Oracle® Database
データ・ウェアハウス・ガイド
19c

F16121-03(原本部品番号:E96243-05)

2021年3月

タイトルおよび著作権情報

Oracle Databaseデータ・ウェアハウス・ガイド, 19c

F16121-03

原本著者: Padmaja Potineni

原本協力者: Frederick Kush

原本協力者: Hermann Baer、Mark Bauer、Subhransu Basu、Nigel Bayliss、Donna Carver、Maria Colgan、Benoit Dageville、Luping Ding、Bud Endress、Bruce Golbus、John Haydu、Keith Laker、Paul Lane、Chun-Chieh Lin、William Lee、George Lumpkin、David McDermid、Alex Melidis、Valarie Moore、Ananth Raghavan、Jack Raitto、Andy Rivenes、Lei Sheng、Wayne Smith、Sankar Subramanian、Margaret Taft、Murali Thiyagarajan、Jean-Francois Verrier、Andreas Walter、Andy Witkowski、Min Xiao、Tsae-Feng Yu、Fred Zemke、Mohamed Ziauddin

目次

はじめに

「はじめに」では、次の項目について説明します。

- 対象読者
- 関連ドキュメント
- ドキュメントのアクセシビリティについて
- 表記規則

対象読者

このマニュアルは、データ・ウェアハウスを設計、メンテナンスおよび使用するデータベース管理者、システム管理者およびデータベース・アプリケーション開発者を対象としています。

このマニュアルを使用するには、リレーショナル・データベースの概念、Oracle Serverの基本概念、およびOracleを実行するオペレーティング・システム環境について詳しく理解していることを前提としています。

ドキュメントのアクセシビリティについて

Oracleのアクセシビリティについての詳細情報は、Oracle Accessibility ProgramのWebサイト(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc)を参照してください。

Oracleサポートへのアクセス

サポートを購入したオラクル社のお客様は、My Oracle Supportを介して電子的なサポートにアクセスできます。詳細情報は(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info)か、聴覚に障害のあるお客様は(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs)を参照してください。

関連ドキュメント

このマニュアルに含まれる例の多くでは、Oracleのインストール時にデフォルトでインストールされるシード・データベースのサンプル・スキーマを使用しています。これらのスキーマがどのように作成され、ユーザーがどのように使用できるかについては、『*Oracle Databaseサンプル・スキーマ*』を参照してください。

このマニュアルは、データ・ウェアハウスに関する標準テキストを補足するものです。Oracle固有の性能を中心に説明しており、データ・ウェアハウスの一般的な性能について詳細に説明するものではありません。追加情報は、次の文書を参照してください。

- 『*The Data Warehouse Toolkit*』(Ralph Kimball著、John Wiley and Sons, 1996)
- 『*Building the Data Warehouse*』(William Inmon著、John Wiley and Sons, 1996)

表記規則

このマニュアルでは次の表記規則を使用します。

規則	意味
太字	太字は、操作に関連する Graphical User Interface 要素、または本文中で定義されている用語および用語集に記載されている用語を示します。

規則	意味
イタリック体	イタリックは、ドキュメントのタイトル、強調またはユーザーが特定の値を指定するプレースホルダ変数を示します。
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、サンプル内のコード、画面に表示されるテキスト、または入力するテキストを示します。

このリリースでの『Oracle Databaseデータ・ウェアハウス・ガイド』の変更点

この章の内容は次のとおりです。

- Oracle Database 18c、バージョン18.1での変更点
- Oracle Database 12cリリース2 (12.2.0.1)での変更点
- Oracle Database 12cリリース1 (12.1.0.2)での変更点
- Oracle Database 12cリリース1 (12.1.0.1)での変更点

Oracle Databaseリリース19cにおける変更点

Oracle Database 19c向け『*Oracle Database*データ・ウェアハウス・ガイド』での変更は次のとおりです。

新機能

- LISTAGG関数のDISTINCTキーワードを使用すると、指定した式から重複を除去できます。

  LISTAGG関数を参照してください。

- マテリアライズド・ビューは、ハイブリッド・パーティション表に基づいて作成できます。クエリー・リライトおよびリフレッシュは、ハイブリッド・パーティション表に基づくマテリアライズド・ビューでサポートされています。

  ハイブリッド・パーティション表に基づくマテリアライズド・ビューの作成、ハイブリッド・パーティション表に基づくマテリアライズド・ビューのリフレッシュおよびハイブリッド・パーティション表がある場合のクエリー・リライトについてを参照してください。

- COUNT(DISTINCT)操作を含む問合せのパフォーマンスは、非加算ファクトを含むマテリアライズド・ビューを使用することによって改善できます。

  ビットマップベースのCOUNT(DISTINCT)関数を含むマテリアライズド・ビューの作成およびビットマップベースのCOUNT(DISTINCT)関数に基づくクエリー・リライトおよびマテリアライズド・ビューを参照してください。

Oracle Database 18c、バージョン18.1での変更点

Oracle Database 18c、バージョン18.1向け『*Oracle Database*データ・ウェアハウス・ガイド』での変更は次のとおりです。

新機能

- 近似問合せ処理の機能強化

  ビジネス・インテリジェンス(BI)アプリケーションは、集計関数(分析関数を含む)を広範囲に使用して、一般的なビジネス問合せに応答します。近似問合せ処理を使用すると、既存の分析ワークロードのパフォーマンスが即座に向上し、より高速な非定型のデータ探索が可能になります。ランキング、合計およびカウントのための近似SQL関数が上位Nスタイルの問合せに使用できるようになりました。

  関連項目:

  近似上位N問合せ処理について

- パラレル・パーティション・ワイズ操作の機能強化

パラレル・パーティション・ワイズのSQL操作により問合せのパフォーマンスが大幅に向上し、応答時間が改善されます。パラレル・パーティション・ワイズ結合は一般に、大規模な結合の処理を効率的かつ高速に行うために使用されます。パラレル・パーティション・ワイズ結合に加えて、SQLウィンドウ句があるSQL操作をパラレル・パーティション・ワイズで実行できます。

関連項目:

ウィンドウ関数を使用したパラレル・パーティション・ワイズ操作

- 分析ビューの機能強化
  - SELECT文のWITHまたはFROM句では、分析ビューの問合せに、分析ビューによりアクセスされるデータを集計の前にフィルタ処理するFILTER FACTキーワードを含めることができるようになりました。新しいADDおよびMEASURESキーワードを使用することにより、SELECT文内に分析ビューの計算済メジャーを定義でき、アプリケーションで独自のメジャーを動的に定義できるようになりました。

    関連項目:

    フィルタ処理されたファクトおよび追加メジャーによる分析ビュー問合せ

  - CREATE ATTRIBUTE DIMENSION文のJOIN PATHキーワードでは、スノーフレーク・スタイルのディメンション表の使用をサポートしています。CREATE ANALYTIC VIEW文のREFERENCES DISTINCTキーワードでは、ディメンション属性とファクト・データが同じ表にある非正規化ファクト表の使用がサポートされます。
  - 分析ビューにおいて、計算済メジャー式で使用できる様々な新しい関数がサポートされるようになりました。この新しい関数には、RANK_*、PERCENTILE_*、STATS_*、COVAR_*、HIER_DEPTH、HIER_LEVEL、HIER_MEMBER_NAME、HIER_MEMBER_UNIQUE_NAME、HIER_CAPTIONおよびHIER_DESCRIPTIONがあります。また、計算済メジャーの定義において階層属性を使用できるようになりました。たとえば、CASE文では属性の値に基づく各種の計算式を指定できます。

Oracle Database 12*c*リリース2 (12.2.0.1)での変更点

Oracle Database 12*c*リリース2 (12.2.0.1)向け『*Oracle Database*データ・ウェアハウス・ガイド』での変更は次のとおりです。

新機能

- 属性ディメンション、階層および分析ビュー

  分析ビューは、既存のデータベース表およびビューに格納されているデータの分析問合せを迅速に効率的に作成する方法を提供します。属性ディメンションは、ディメンション表またはビューの列を属性として指定し、属性をレベルに編成します。階層はレベルを階層的に編成します。分析ビューは階層を参照し、ファクト表のデータを参照するメジャーおよびそのデータを使用して行う計算を指定するメジャーを指定します。

  分析ビューの概要を参照してください。

- リアルタイムのマテリアライズド・ビュー

  リアルタイム・マテリアライズド・ビューは、マテリアライズド・ビューが失効とマークされている場合でも最新のデータをユーザーの問合せに提供することによって、マテリアライズド・ビューの可用性をさらに改善します。失効したリアルタイム・マテ

リアライズド・ビューに問合せがアクセスすると、Oracle Databaseは、マテリアライズド・ビューの失効したデータとマテリアライズド・ビュー・ログに格納されているデルタ情報を使用して、その場で最新の問合せ結果を計算します。

リアルタイムのマテリアライズド・ビューの使用を参照してください。

- マテリアライズド・ビューのON STATEMENTリフレッシュ・モード

  ON STATEMENTリフレッシュ・モードでは、実表に対してDML操作が実行されるたびに、トランザクションをコミットしなくてもマテリアライズド・ビューがリフレッシュされます。このモードでは、実表のマテリアライズド・ビュー・ログを保守する必要はありません。

  マテリアライズド・ビューのON STATEMENTリフレッシュについてを参照してください。

- マテリアライズド・ビュー・リフレッシュ統計の管理

  マテリアライズド・ビューのリフレッシュ操作に関する統計が収集され、データベースに保存されます。現在および過去の統計を使用し、マテリアライズド・ビューのリフレッシュ・パフォーマンスを時間の経過に沿って分析できます。

  マテリアライズド・ビューのリフレッシュ操作の監視を参照してください。

- 近似問合せ処理のサポート

  近似問合せ処理は、正確な結果との誤差がごくわずかな近似結果を返すSQL関数を使用する探索問合せに対して非常に高速で応答します。近似結果を返すSQL関数を含む問合せ(または近似問合せ)は、マテリアライズド・ビューを作成するために使用できます。近似問合せに基づくマテリアライズド・ビューは、クエリー・リライトの対象となり、高速リフレッシュできます。

  近似問合せ処理についてを参照してください。

- LISTAGG拡張

  LISTAGG関数は、LISTAGG関数によって返される連結された文字列がVARCHAR2データ型によってサポートされる最大長を超えた場合のシナリオに対する制御を提供するようになりました。

  LISTAGG関数を参照してください。

- SQL関数を使用したデータ・エラーの処理の改善

  CAST演算子は、データ型変換エラーが発生した場合に、ユーザーが指定した値を返すことができるようになりました。VALIDATE_CONVERSION関数は、指定したデータ型への変換が成功するかどうかを判別します。

  SQLを使用したデータ・エラーの処理を参照してください。

- IM列ストアによる問合せパフォーマンスの改善

  インメモリー式(IM式)は、頻繁に評価される問合せ式の結果を計算して、インメモリー列ストア(IM列ストア)移入します。IM式をIM列ストアに移入すると、後続の問合せでそれらを使用でき、問合せ応答時間が速くなります。

  インメモリー仮想列(IM仮想列)では、表に定義されている仮想列をIM列ストアに移入できます。表のすべての仮想列またはサブセットのみをメモリーに移入し、それらの仮想列にアクセスする分析問合せのパフォーマンスを改善できます。

  インメモリー式を使用した問合せパフォーマンスの改善についておよびインメモリー仮想列を使用した問合せパフォーマンスの改善についてを参照してください。

- IM列ストア用の自動データ最適化(ADO)のサポート

  ADOを使用すると、IM列ストアのコンテンツを管理できます。ADOはヒート・マップ統計を使用し、最もメリットがある要素のみがIM列ストアに格納されるようにします。これにより、IM列ストアを使用することによるパフォーマンス上のメリットが高まります。

インメモリー列ストアと自動データ最適化についてを参照してください。

- 表の再定義時の一括更新のパフォーマンスの最適化

  DBMS_REDEFINITIONパッケージのEXECUTE_UPDATEプロシージャは、表に対する一括更新のパフォーマンスを最適化できます。

  バッチ更新および表のオンライン再定義についてを参照してください。

- 表のオンライン再定義中のマテリアライズド・ビューのリフレッシュのサポート

  DBMS_REDEFINITIONパッケージを使用すると、表のオンライン再定義中に高速リフレッシュ可能な依存マテリアライズド・ビューを増分リフレッシュできます。

  表のオンライン再定義中に依存マテリアライズド・ビューをリフレッシュする方法についてを参照してください。

- 外部表のパーティション化のサポート

  外部表のパーティション化では、既存のOracleパーティション化が外部表に拡張されました。これにより、外部ソースがよりよく統合されるようになりました。たとえば、外部表のパーティションとHIVE表のパーティションを連携できます。既存のパーティション・プルーニング・テクニックを使用した、データベースの外部に格納されているデータの問合せパフォーマンスも改善されました。

  簡易データ・アクセスのためのパーティション化を参照してください。

- データベース操作の監視

  最適に実行されていない抽出、変換およびロード(ETL)ジョブを監視し、パフォーマンスのボトルネックを識別するために役立ちます。

  ETL操作の監視の概要を参照してください。

Oracle Database 12cリリース1 (12.1.0.2)での変更点

Oracle Database 12cリリース1 (12.1.0.2)向け『*Oracle Database*データ・ウェアハウス・ガイド』での変更は次のとおりです。

新機能

- Oracle In-Memory Column Store

  SGA内のオプション領域にあり、表、表パーティションおよび個別の列が圧縮列形式で格納されるOracle In-Memory Column Store(IM列ストア)。IM列ストアはデータベースバッファ・キャッシュに代わるものではなく、補完するものです。

  IM列ストアは、主に表スキャンとWHERE句述語の適用のパフォーマンスを改善します。表スキャンの高速化により、オプティマイザがブルーム・フィルタとVECTOR GROUP BY変換を選択する可能性が高くなります。

- 属性クラスタリング

  表の属性クラスタリングにより、表中の特定の列の値に基づく順序に従って、データをディスク上の近接した位置に格納するようにできます。表ゾーン・マップを介したプルーニングの効率が向上するため、表スキャンと、索引を介した表データ参照のI/OおよびCPUコストは低下します。

  関連項目:

属性クラスタリング

- ゾーン・マップ

  ゾーン・マップは、ディスク上でのデータの物理的な場所に基づくデータの自然なプルーニングを可能にします。完全表スキャン時には関連したデータ・ブロックのみにアクセスし、索引スキャン時には関連したデータ列のみにアクセスするため、データ・アクセスのI/OコストとCPUコストを削減できます。

  関連項目:

  ゾーン・マップの使用

- インメモリー集計

  VECTOR GROUP BY操作は、1つ以上の比較的小さな表を大きな表に結合する問合せや、データを集計する問合せのパフォーマンスを向上させます。データ・ウェアハウジングのコンテキストでは、VECTOR GROUP BY集計は、IM列ストアからデータを選択するスター・クエリーの場合によく選択されます。

  VECTOR GROUP BY集計は、1つのファクト表に複数のディメンション表を結合する際の処理を最小化します。パラレル問合せに関連するインフラストラクチャを使用し、パフォーマンスを最大化する、CPU効率のよいアルゴリズムをそこにブレンドします。

- 大きな表の自動キャッシング

  大きな表の自動キャッシングは、バッファ・キャッシュに完全にはおさまらない大きな表の、インメモリー問合せのパフォーマンスを向上させます。そのような表は、大きな表のキャッシュと呼ばれる、オプションの、データベース・バッファ・キャッシュの構成可能な部分に格納できます。

  関連項目:

  大きな表の自動キャッシングによるインメモリー・パラレル問合せのパフォーマンス向上

Oracle Database 12cリリース1 (12.1.0.1)での変更点

Oracle Database 12cリリース1(12.1.0.1)向け『*Oracle Database*データ・ウェアハウス・ガイド』での変更は次のとおりです。

新機能

- パターン一致

  SQLはパターン一致をサポートするよう拡張されたため、様々な順序のパターンを容易に検出することが可能になりました。パターン一致は、株価の監視、ネットワーク侵入検知およびE-Commerce購買追跡など、多くの商用アプリケーションで役立ちます。

  関連項目:

  詳細は、パターン一致用SQLを参照してください。

- 上位N問合せに対するネイティブSQLのサポート

  新しい*row_limiting_clause*により、問合せで戻される行の数を制限できます。オフセットおよび戻される行の数または割合を指定できます。これにより、上位Nレポートが可能になります。

  関連項目:

  詳細は、SQLの行の制限を参照してください。

- バルク・ロード操作に対するオンラインでの統計情報収集

  Oracle Database 12cより、バルク・ロード操作の一部として、表の統計情報がデータベースにより自動的に収集されます。

- 同期リフレッシュ

  同期リフレッシュと呼ばれる新しいタイプのリフレッシュにより、一連の表と表に定義されているマテリアライズド・ビューが、常に同期した状態を維持できます。ここでは、増分データのロードが厳密に制御され、定期的に発生するデータ・ウェアハウスには非常に適しています。

  関連項目:

  詳細は、同期リフレッシュを参照してください。

- ホーム外リフレッシュ

  マテリアライズド・ビューのリフレッシュ・パフォーマンスおよび可用性の向上のため、新しいタイプのリフレッシュが使用できます。このリフレッシュは、リフレッシュの際に外部表を使用するため、ホーム外リフレッシュと呼ばれます。特に、従来のDML文ではうまく対応できない大量のデータ変更を伴う状況を処理する際に効果的です。

  関連項目:

  詳細は、マテリアライズド・ビューのリフレッシュを参照してください。

サポート対象外機能

以前このドキュメントに記載されていた機能の一部は、Oracle Database 12cリリース1ではサポート対象外となっています。サポートされない機能のリストは、*『Oracle Databaseアップグレード・ガイド』*を参照してください。

第I部 データ・ウェアハウス: 基本

この部では、データ・ウェアハウスの基本概念について説明します。

この部は、次の章で構成されています。

- データ・ウェアハウスの概念の概要
- データ・ウェアハウスの論理設計
- データ・ウェアハウスの物理設計
- データ・ウェアハウスの最適化および技法

1 データ・ウェアハウスの概念の概要

この章では、Oracleのデータ・ウェアハウス実装の概要について説明します。この章の内容は次のとおりです。

- データ・ウェアハウスについて
- OLTPとデータ・ウェアハウス環境の比較
- データ・ウェアハウスの一般的なタスク
- データ・ウェアハウス・アーキテクチャ

1.1 データ・ウェアハウスとは

データ・ウェアハウスは、ビジネス・インテリジェンス・アクティビティを可能にするよう設計されたデータベースです。ユーザーの理解を深め、組織のパフォーマンスを向上させる目的があります。これは、トランザクション処理用ではなく、問合せおよび分析用に設計され、通常、トランザクション・データから導出された履歴データが含まれますが、他のソースからのデータを含めることもできます。データ・ウェアハウスは分析処理をトランザクション処理の負荷と分離し、組織で、様々なソースからのデータを整理統合できるようにします。これは、次のような場合に役立ちます。

- 履歴データのメンテナンス
- ビジネスをよりよく理解し改善するためのデータの分析

データ・ウェアハウス環境には、リレーショナル・データベースに加えて、抽出、転送、変換、ロード(ETL)のソリューションや、統計分析、レポート、データ・マイニング機能、クライアント分析ツール、および、データ収集処理や有益かつ実用的な情報へのデータ変換処理、ビジネス・ユーザーへのデータ配信処理を管理するその他のアプリケーションが含まれます。

ビジネス・インテリジェンスの向上という目的を達成するため、データ・ウェアハウスでは複数ソースから収集されたデータを操作します。ソース・データは、内部開発システム、市販アプリケーション、サード・パーティ・データ・シンジケータなどのソースから収集されます。データには取引、製造、マーケティング、人事管理などが含まれます。今日のビッグ・データの世界では、データは、Webサイトでの何十億回ものクリックの一つ一つであったり、複雑な装置に組み込まれたセンサーからの大量のデータ・ストリームの場合もあります。

データ・ウェアハウスは、オンライン・トランザクション処理(OLTP)システムとは別個のものです。データ・ウェアハウスでは、分析ワークロードをトランザクション・ワークロードから分離します。したがって、データ・ウェアハウスは非常に読取り指向のシステムです。書込みおよび更新に対し、読取りデータははるかに大量です。これにより、分析のパフォーマンスが非常に向上する上、トランザクション・システムへの影響を避けられます。多くのソースからのデータを統合するようデータ・ウェアハウス・システムを最適化し、組織内の真の単一ソースにするという主要目的を達成できます。すべてのユーザーが参照可能な、一貫性のあるデータのソースを保持することには、大きな価値があります。多くの論争を防ぎ、意思決定を効率化します。

データ・ウェアハウスには通常、複数月または複数年のデータが格納されており、履歴の分析をサポートします。データ・ウェアハウスのデータは通常、複数のデータ・ソースからの抽出、変換およびロード(ETL)プロセスを通してロードされます。最新のデータ・ウェアハウスは、データ・ウェアハウスをホストするデータベース上で、すべてまたはほとんどのデータ変換が行われる抽出、ロードおよび変換(ELT)アーキテクチャに移行しつつあります。ETLプロセスの定義がデータ・ウェアハウスの設計作業の非常に大きな部分であることに注意することが重要です。同様に、データ・ウェアハウスの稼動後は、ETL操作の速度と信頼性がデータ・ウェアハウスの基盤になります。

データ・ウェアハウスのユーザーは、多くの場合、時間に関連するデータの分析を実行します。例として、昨年の連結売上高、在庫分析、製品別収益および顧客別収益があります。しかし、時間中心か否かは別として、どんなに適切に見えるデータでも、申し分ない設計のデータ・ウェアハウスが要求を十分満たすだけの柔軟性を備えていても、ユーザーはデータを分析することを望みます。ユーザーは、非常に集約されたデータを必要とする場合もあれば、詳細なドリル・ダウンが必要な場合もあります。より

高度な分析に、傾向分析とデータ・マイニングがあり、既存のデータを使用して傾向または今後を予測します。データ・ウェアハウスは、ミドルウェア・ビジネス・インテリジェンス環境で使用されるベースのエンジンとして機能し、エンド・ユーザーにレポート、ダッシュボードその他のインタフェースを提供します。

前述の説明ではデータ・ウェアハウスという用語に注目してきましたが、説明する必要がある重要な用語が他に2つあります。それはデータ・マートとオペレーショナル・データ・ストア(ODS)です。

データ・マートはデータ・ウェアハウス同様の役割を果たしますが、その適用範囲は意図的に制限されています。これは、ある特定の部門または業務に対して提供されます。データ・ウェアハウスに対するデータ・マートのメリットは、適用範囲が制限されているため、高速で作成できる点です。ただし、データ・マートでは非一貫性の問題も発生します。データ・マート全体でデータおよび計算の定義の一貫性を保つため、厳しい統制をとります。この問題は広く知られているため、データ・マートには2つの形式があります。非依存型データ・マートは、ソースのデータが直接入力されるものです。情報が一貫せず、孤立してしまう可能性があります。依存型データ・マートは、既存のデータ・ウェアハウスから入力されるものです。依存型データ・マートでは非一貫性の問題は回避できますが、企業レベルのデータ・ウェアハウスがすでに存在している必要があります。

オペレーショナル・データ・ストアは日常操作のサポートを目的としたものです。ODSのデータはクリーンアップされ、妥当性チェック済ですが、履歴までは対応されていません。つまり、当日のデータのみに対応しています。ODSはデータ・ウェアハウスで扱うことが可能な、豊富な履歴を扱う問合せをサポートするかわりに、データ・ウェアハウスにまだロードされていない最新のデータにアクセスする環境をデータ・ウェアハウスに提供します。ODSは、データ・ウェアハウスのロードのソースとして使用される場合もあります。データ・ウェアハウスのロード技術の進歩に伴い、データ・ロードのソースとしてのODSのニーズは低くなってきました。代替として、常時トリクルフィード・システムにより、データ・ウェアハウスのほぼリアルタイムのロードが可能です。

データ・ウェアハウスを導入する際は、William Inmon氏が提唱するデータ・ウェアハウスの次の特性を十分に理解する必要があります。

- サブジェクト指向
- 統合
- 恒常的
- 時系列

サブジェクト指向

データ・ウェアハウスは、データの分析に役立つように設計されています。たとえば、会社の売上データの詳細が必要な場合は、売上を中心とするデータ・ウェアハウスを作成できます。このデータ・ウェアハウスでは、「昨年、この品目を最も多く購入した顧客は誰だったか」、「来年、この品目を最も多く購入しそうな顧客は誰か」のような質問に答えることができます。このようにデータ・ウェアハウスをサブジェクト(この場合は売上)別に定義できるため、データ・ウェアハウスの使用はサブジェクト指向となります。

統合化

統合化は、サブジェクト指向と密接な関係があります。データ・ウェアハウスには、様々なソースからのデータを一貫性のある形式で格納する必要があります。また、名前の競合や単位間の不一致などの問題を解決する必要があります。これが達成されれば、データ・ウェアハウスは統合化されたことになります。

恒常的

恒常的とは、一度データ・ウェアハウスに入ったデータは変更できないことを意味します。これは、データ・ウェアハウスの目的が、何が発生したかを分析することにあるためです。

時系列

データ・ウェアハウスでは、時系列という用語が意味する、時間経過に伴う変化に重点が置かれています。ビジネスの動向を見いだし、背後に潜むパターンおよび関係性を識別するには、アナリストは大量のデータを必要とします。これは、オンライン・トラン

ザクション処理(OLTP)システムとは非常に対照的です。OLTPシステムでは、パフォーマンス要件のために、履歴データをアーカイブに移動させる必要があります。

1.1.1 データ・ウェアハウスの主な特性

データ・ウェアハウスの主な特性は次のとおりです。

- データはアクセスの簡易性および問合せパフォーマンスの高速化に対応するよう構成されます。
- エンド・ユーザーは時間に敏感で、思考スピードでの応答時間を求めます。
- 大量の履歴データが使用されます。
- 問合せでは通常、数千行もの大量のデータを取得します。
- 定義済問合せと非定型問合せの両方とも一般的です。
- データ・ロードには複数のソースと変換が関与します。

一般的に、データ・ウェアハウスを正常に動作させるためには、高いデータ・スループットによる高速な問合せのパフォーマンスが重要となります。

1.2 OLTPとデータ・ウェアハウス環境の比較

OLTPシステムとデータ・ウェアハウスには重要な相違点があります。この2つのシステムにおける大きな違いの1つは、OLTP環境での一般的なデータ正規化タイプは第3正規形(3NF)ですが、データ・ウェアハウスでは、そうとは限らないということです。

データ・ウェアハウスおよびOLTPシステムの要件は、大きく異なります。次に、典型的なデータ・ウェアハウスとOLTPシステムのいくつかの違いの例を示します。

- 処理負荷

  データ・ウェアハウスは、非定型の問合せおよびデータ分析に適応するように設計されています。データ・ウェアハウスの処理負荷は、事前には不明な場合があります。このため、データ・ウェアハウスは、様々な問合せ操作および分析操作を適切に実行できるように最適化する必要があります。

  OLTPシステムは、事前に定義された操作のみサポートします。これらの操作のみをサポートするように、アプリケーションの特別なチューニングや設計が必要になる場合があります。

- データ修正

  データ・ウェアハウスのデータは、大量データ修正の技術を使用して、ETLプロセスによって定期的に(毎晩、毎週など)更新されます。データ・マイニングなどの分析ツールを使用して、関連する確率から予測を作成したり、顧客を市場セグメントに当てはめたり、顧客プロファイルを作成したりする場合を除き、データ・ウェアハウスのエンド・ユーザーは、データ・ウェアハウスを直接更新することはありません。

  OLTPシステムでは、エンド・ユーザーが機械的に、個々のデータ修正の都度、修正文を発行します。OLTPデータベースは常に最新であり、各ビジネス・トランザクションの現在の状態が反映されます。

- スキーマ設計

  データ・ウェアハウスは、部分的に非正規化されたスキーマを使用して、問合せおよび分析のパフォーマンスを最適化します。

  OLTPシステムは、完全に正規化されたスキーマを使用して、更新/挿入/削除のパフォーマンスを最適化し、データ整合性を保証します。

- 典型的な操作

典型的なデータ・ウェアハウスの問合せでは、膨大な数の列がスキャンされる場合があります。たとえば、「先月のすべての顧客に対する合計売上の検索」などの場合です。

典型的なOLTP操作では、少数のレコードのみがアクセスされます。たとえば、「この顧客に対する現在の注文の取出し」などの場合です。

- 履歴データ

  データ・ウェアハウスには、通常、長い年月分のデータが格納されています。これは、履歴の分析およびレポートをサポートするためです。

  OLTPシステムには、通常、数週間または数か月分のデータのみが格納されています。OLTPシステムには、現行のトランザクション要件を満たすために必要な履歴データのみが格納されます。

1.3 データ・ウェアハウスの一般的なタスク

Oracleデータ・ウェアハウス管理者または設計者として、次のタスクを行うことが予想されます。

- データ・ウェアハウスとして使用するためのOracle Databaseの構成
- データ・ウェアハウスの設計
- データベース・ソフトウェアおよびデータ・ウェアハウス・ソフトウェアの新規リリースへのアップグレードの実行
- スキーマ・オブジェクトの管理(表、索引およびマテリアライズド・ビューなど)
- ユーザーおよびセキュリティの管理
- 抽出、変換およびロード(ETL)の各プロセスに使用するルーチンの開発
- データ・ウェアハウス内のデータに基づいたレポートの作成
- 必要に応じたデータ・ウェアハウスのバックアップおよびリカバリの実行
- データ・ウェアハウスのパフォーマンスの監視と必要に応じた予防処理または修正処理

中小規模のデータ・ウェアハウス環境では、これらのタスクを単独で実行する可能性があります。大企業のような環境の場合、ジョブはデータベース・セキュリティまたはデータベースのチューニングなどの専門を持つ数名のDBAおよび設計者に分割されます。

これらのタスクについては、次で説明されています。

- パーティション化の詳細は、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照してください。
- データベース・セキュリティの詳細は、『Oracle Databaseセキュリティ・ガイド』を参照してください。
- データベースのパフォーマンスの詳細は、『Oracle Databaseパフォーマンス・チューニング・ガイド』および『Oracle Database SQLチューニング・ガイド』を参照してください。
- バックアップおよびリカバリの詳細は、『*Oracle Database*バックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。
- ODIの詳細は、『*Oracle Fusion Middleware Oracle Data Integrator*開発者ガイド』を参照してください。

1.4 データ・ウェアハウス・アーキテクチャ

データ・ウェアハウスおよびそのアーキテクチャは、組織の特定の状況に応じて変化します。一般的なアーキテクチャは、次の3つです。

- データ・ウェアハウス・アーキテクチャ: 基本

- データ・ウェアハウス・アーキテクチャ: ステージング・エリアを伴う
- データ・ウェアハウス・アーキテクチャ: ステージング・エリアおよびデータ・マートを伴う

1.4.1 データ・ウェアハウス・アーキテクチャ: 基本

図1-1に、データ・ウェアハウスの単純なアーキテクチャを示します。エンド・ユーザーは、複数のソース・システムから導出されたデータに、データ・ウェアハウスを通じて直接アクセスします。

図1-1 データ・ウェアハウスのアーキテクチャ

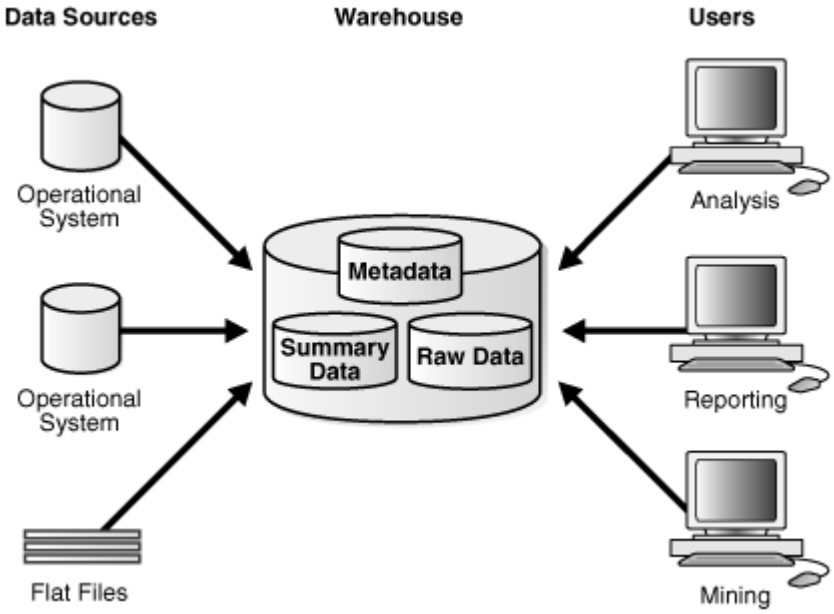


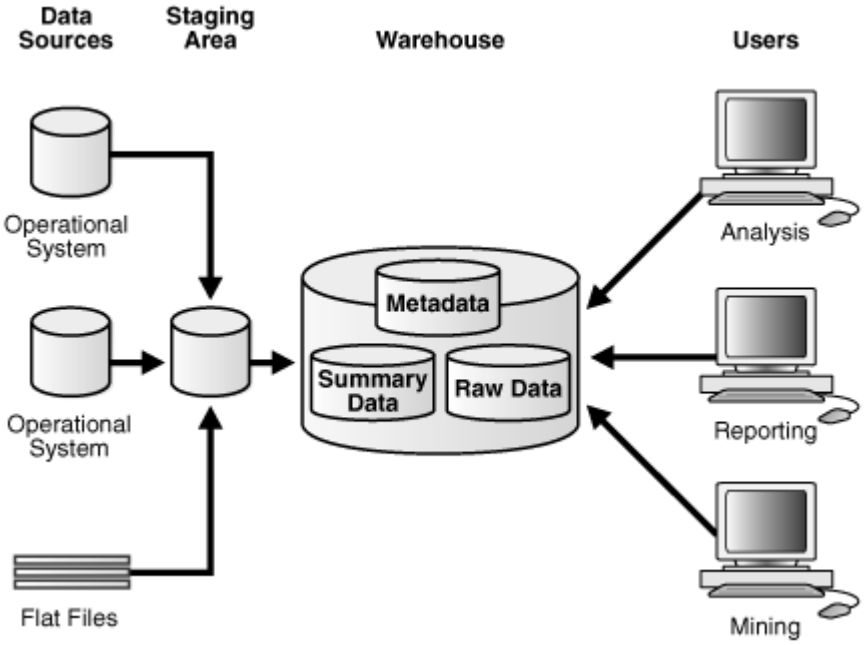
図1-1では、従来のOLTPシステムのメタデータと生データは、付加的なデータであるサマリー・データと同様に示されています。サマリーは、コストが高く、実行に時間のかかる共通の操作を事前に計算し、1秒未満でデータ取得するためのメカニズムです。たとえば、典型的なデータ・ウェアハウス問合せとして、8月の売上などを取り出す問合せがあります。Oracleデータベースでは、サマリーをマテリアライズド・ビューと呼びます。

データ・ウェアハウス・アーキテクチャの中心としての、統合された生データの記憶域は通常、エンタープライズ・データ・ウェアハウス(EDW)と呼ばれます。すべての関連するビジネス情報を、最も詳細な形式で保持することで、EDWは組織のビジネスのあらゆる角度からの考察を可能にします。

1.4.2 データ・ウェアハウス・アーキテクチャ: ステージング・エリアを伴う

図1-2に示すように、業務系データはウェアハウスに格納する前にクレンジングして加工しておく必要があります。これはプログラムで処理できますが、ほとんどのデータ・ウェアハウスではステージング・エリアが使用されます。ステージング・エリアにより、データ・クレンジングおよび複数のソース・システムからの業務系データの統合が簡素化されますが、企業のすべての関連情報が統合される、エンタープライズ・データ・ウェアハウスでは特に有効です。図1-2に、この典型的なアーキテクチャを示します。

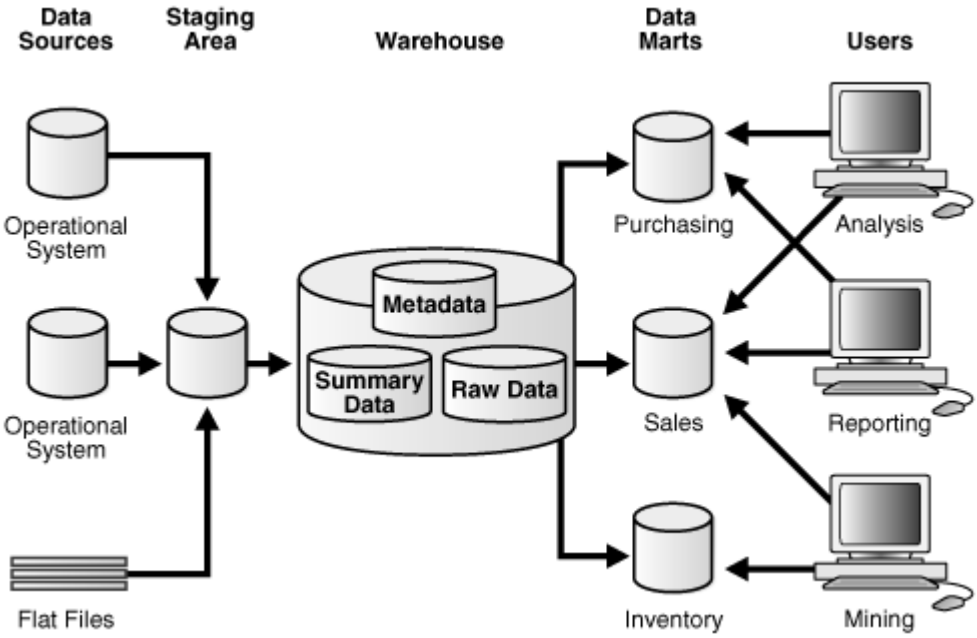
図1-2 データ・ウェアハウス・アーキテクチャ(ステージング・エリアを伴う)



1.4.3 データ・ウェアハウス・アーキテクチャ: ステージング・エリアおよびデータ・マートを伴う

図1-2のアーキテクチャはきわめて一般的ですが、ウェアハウスのアーキテクチャを組織内のグループごとにカスタマイズすることもできます。そのためには、特定の業務に合せて設計された**データ・マート**を追加します。図1-3の例では、発注、受注、在庫が分離されています。この例で、ファイナンシャル・アナリストは発注と受注の履歴データを分析したり、履歴データをマイニングして顧客の行動の予測を作成できます。

図1-3 データ・ウェアハウス・アーキテクチャ(ステージング・エリアおよびデータ・マートを伴う)



ノート:

データ・マートは、物理的にインスタンス化あるいはビューを介して完全に論理的に実装できます。さらに、データ・マートはエンタープライズ・データ・ウェアハウスと共通の場所に配置することも、別のシステムとして構築することもできます。エンタープライズ・データ・ウェアハウスとその周辺のデータ・マートを伴う、エンドツーエンドのデータ・ウェアハウス・アーキテクチャの構築については、このマニュアルでは対象としません。

2 データ・ウェアハウスの論理設計

この章では、データ・ウェアハウス環境の論理設計の作成方法について説明します。内容は次のとおりです。

- データ・ウェアハウスの論理設計と物理設計の比較
- 論理設計の作成
- 第3正規形のスキーマについて
- スター・スキーマについて
- インメモリー列ストアを使用した分析の改善
- 大きな表の自動キャッシングによるインメモリー・パラレル問合せのパフォーマンス向上

2.1 データ・ウェアハウスの論理設計と物理設計の比較

組織でのエンタープライズ・データ・ウェアハウス構築がすでに決定されているとします。そして、ビジネス要件の定義、ビジネスの目的の適用範囲の打合せ、および概念的な設計も完了しているとします。そこで、その要件をシステムに移行できるように変換する必要があります。そのためには、データ・ウェアハウスの論理設計と物理設計を行います。定義する内容は、次のとおりです。

- 具体的なデータ内容
- データ・グループ内およびデータ・グループ間の関係
- データ・ウェアハウスをサポートするシステム環境
- 必要なデータ変換
- データのリフレッシュ頻度

論理設計は、物理設計に比べて概念的で抽象的です。論理設計では、オブジェクト間の論理的な関係を検討します。物理設計では、オブジェクトの格納と取出しの他、転送処理およびバックアップ/リカバリの観点から、最も効果的な方法を検討します。

設計では、エンド・ユーザーのニーズを優先させる必要があります。エンド・ユーザーは、通常、個々のトランザクションではなく、分析を行って集計されたデータを見ます。ただし、実際にデータを見るまで、何が必要であるかがエンド・ユーザーにはわからない場合があります。適切に計画された設計であれば、ユーザーのニーズの変化や発展に応じて、拡張および変更を行うことができます。

まず、論理設計から始めると、情報についての要件に集中でき、実装の詳細は後で検討できます。

2.2 論理設計の作成

論理設計とは、概念的で抽象的な設計です。ここでは、物理的な実装の詳細は説明しません。必要な情報の種類の定義方法のみを扱います。

組織の論理的な情報要件のモデリングに使用できるテクニックの1つが、E-Rモデリングです。E-Rモデリングでは、重要事項(エンティティ)、重要事項のプロパティ(属性)および相互の関連(リレーションシップ)を特定します。

物理設計プロセスでは、データを、エンティティおよび属性と呼ばれる、一連の論理的な関係に配置します。エンティティは、情報の大きいまとまりを表します。エンティティは、リレーショナル・データベースの表にマップされます。属性はエンティティのコンポーネントで、エンティティの一意性を定義します。属性は、リレーショナル・データベースの列にマップされます。

データの一貫性を確実にするには、一意識別子を使用する必要があります。一意識別子は、同じ項目が異なる場所に表示

される場合に、両者を区別できるように表に追加する識別子です。これは通常、物理設計における主キーです。

E-Rモデリングは、完全に論理的なものであり、OLTPとデータ・ウェアハウス・システムの両方に適用されます。データ・ウェアハウス環境で一般的に使用されている、様々な物理スキーマ・モデリング技法、つまり、エンタープライズ・データ・ウェアハウス環境での正規化(3NF)スキーマ、データ・マートでのスターまたはスノーフレーク・スキーマ、またはこれら古典的モデリング技法の両方のコンポーネントを使用した、ハイブリッド・スキーマにも適用可能です。

関連項目:

- ODIの詳細は、*Oracle Fusion Middleware Oracle Data Integratorでの統合プロジェクトの開発*を参照してください。

2.2.1 スキーマとは

スキーマとは、表、ビュー、索引およびシノニムを含むデータベース・オブジェクトのコレクションです。データ・ウェアハウス用に設計されたスキーマ・モデルには、様々な方法でスキーマ・オブジェクトを配置できます。ほとんどのデータ・ウェアハウスでは、ディメンショナル・モデルが使用されます。

データ・ウェアハウス・スキーマの設計には、ソース・データのモデルとユーザー要件を使用できます。会社のエンタープライズ・データ・モデルからソース・データを取得し、これを基にデータ・ウェアハウス用の論理データ・モデルをリバース・エンジニアすることが可能な場合があります。論理データ・ウェアハウス・モデルの物理的な実装には、コンピュータのサイズ、ユーザー数、記憶域の容量、ネットワークの種類、ソフトウェアなどのシステム・パラメータに応じて、多少の変更が必要な場合があります。スキーマの設計の重要な部分は、第3正規形、スターまたはスノーフレーク・スキーマを使用するかどうかということですが、これらについては、後で詳しく説明します。

2.3 第3正規形のスキーマについて

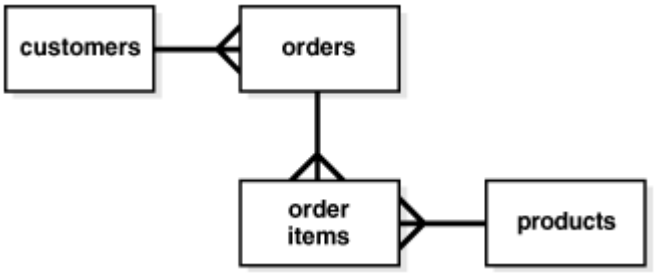
第3正規形の設計では、データの冗長性を最小化し、データの挿入、更新および削除において非一貫性を回避することが求められます。3NFの設計は、オンライン・トランザクション処理(OLTP)システムで長く使用されてきました。OLTPシステムでは、データの挿入、更新および削除時のパフォーマンスおよび精度を最大限にする必要があります。トランザクションは、可能なかぎり早急に処理する必要があります。そうでなければ、業務でイベントのフローを処理できなくなり、売上の喪失または損害を被る可能性があります。そのため3NFの設計により、挿入、更新および削除を減速させる可能性のある、冗長なデータ操作を回避し、表のロックを最小限にします。3NFの設計は、特定のアプリケーションのニーズによるデータの抽出にも有効です。環境に新しいタイプのデータが追加された場合、データ・モデルを比較的容易に拡張し、既存アプリケーションへの影響を最小限に抑えることができます。同様に、データ・ウェアハウスで実行する、まったく新しいタイプの分析がある場合、適切に設計された3NFスキーマであればデータ構造を再設計することなく処理することが可能です。3NFの設計には非常に柔軟性がありますが、代償も伴います。3NFのデータベースは非常に多くの表を使用するため、多数の結合を使用した複雑な問合せが必要です。3NFで構築された、フル・スケールのエンタープライズ・モデルでは、一般的に1,000を超える表がスキーマに記載されています。データ・ウェアハウスでは多くの場合、多数の表の多数の行へのアクセスを必要としますが、含まれる問合せの種類の数に比例して、この設計が理解を妨げるようになり、パフォーマンスが低下します。問合せの作成者が人間であれ、ビジネス・インテリジェンス・ツールおよびアプリケーションであれ、使用可能な表が非常に多数の場合は、データの特定の部分に必要な表を選択および結合することは、複雑な作業になります。問合せジェネレータによって表が即座に選択される場合でも、3NFスキーマは多くの場合、単一の問合せで多数の表を必要とします。問合せに存在する表が多くなるほど、潜在的なデータ・アクセス・パスが増えることになり、データベース問合せオプティマイザの負荷が増大します。最終的には、問合せパフォーマンスが低下します。

3NFシステムにおける問合せパフォーマンス低下の問題は、レポートおよび分析の作成に使用される、コアの問合せに必ずしも

限定されたものではありません。内容を理解するためユーザーがデータのサブセットを参照しているような、より単純なタスクでも発生する可能性があります。同様に、3NFスキーマの複雑さは、問合せおよびレポートの制約に使用される、データの選択リストの作成に影響する可能性があります。これらは比較的軽微な問題に見えますが、このようなプロセスでの応答時間が速いことは、ユーザーの満足度に大きく影響します。

図2-1に、3NFスキーマのごく一部を示します。冗長性の高いデータの格納を避けるため、注文情報が注文と注文品目に分割されていることに注意してください。表間の関係上のカラスの足跡マークは、エンティティ間の1対多の関係を示しています。このように、1つの注文には複数の注文品目があり、1人の顧客には多くの注文があり、さらには1つの製品が多くの注文品目の中にあります。この図では非常に簡単な例を示していますが、データの冗長性を最小化することが、スキーマに多くの表ができることにつながることがわかります。

図2-1 第3正規形スキーマの断片



関連項目:

3NFスキーマの設計の概念

2.3.1 正規化について

正規化とは、各ファクトを1箇所のみに保持し、データの冗長性や挿入、更新および削除の非一貫性を回避するという、高いレベルの目的を持ったデータ設計プロセスです。正規化には複数のレベルがありますが、この項ではそれらの内の最初の3つを説明します。第3正規形(3NF)という用語がいかに基本的なものであるかを考えれば、3NFがどのように実現されるかを見なければ意味がありません。

売上を追跡する場合を考えてみます。追跡の中心になるエンティティは受注ですが、各受注には品名、価格、数量など、各購買品目についての詳細(明細項目と呼ぶ)が含まれます。受注には顧客の氏名や住所なども含まれます。様々な明細項目のある受注もあれば、1つしかない場合もあります。

第1正規形(1NF)では、データ・グループの繰り返しや重複行は存在しません。行と列の各交差部分(フィールド)には、1つの値のみが格納されており、同一のファクトを格納する列のグループは存在しません。重複行を避けるため、主キーが存在します。第1正規形における受注では、表の1つのフィールドには各受注の複数の明細項目は表示されません。また、明細項目を示す複数の列は存在しません。

次の第2正規形(2NF)は、設計は第1正規形のものですが、各非キー列は完全な主キーに依存しています。そこで、明細項目を受注明細項目表に抜き出し、各行が1つの注文の1明細項目を表すようにします。明細項目表を見ると、売上項目の品名は明細項目表の主キーに依存していないことがわかります。つまり、売上項目は独自のエンティティです。したがって、売上項目を品名を示している独自の表に移動します。各項目に対する請求額は、注文により異なる(たとえば、値引によって)ため、明細項目表に残します。この受注表の例では、顧客の氏名および住所は受注表の主キーに依存していません。つまり、顧客は独自のエンティティです。したがって、顧客の氏名および住所列は独自の顧客情報表に移動します。

次の第3正規形の目的は、非キー属性に依存しないことを確認することです。つまり、行のサブジェクト(主キー)に直接関連付けられていない列を取得して、それらを独自の表に配置することが目的になります。したがって、顧客の名前や顧客の市などの

顧客に関する詳細を別個の表に配置してから、customer外部キーをorders表に追加する必要があります。

2NF表と3NF表の相違の別の例として、トーナメント、年、勝者および勝者の誕生日の列を含む、テニス・トーナメントの勝者の表が挙げられます。この場合、勝者の誕生日は、不整合を起こしやすくなります。なぜなら、同じ人が、異なるレコード内で異なる誕生日で表示される可能性があるためです。この潜在的な問題を回避する方法は、表を、トーナメントの勝者用のものと、選手の誕生日用のものに分けることです。

2.3.2 3NFスキーマの設計の概念

次の項では、3NFスキーマの手法を使用したデータ・ウェアハウス環境向けのモデリングの際の、基本概念のいくつかについて説明します。ここでは3NFモデリング(またはさらに高いレベルの正規化)の論理的根拠についての説明ではなく、データ・ウェアハウスに関連する主要コンポーネントに注目することを意図しています。

データ・ウェアハウスに関連する主要な3NFスキーマ設計の概念の一部を次に示します。

- 主キーの識別
- 外部キーの関係および参照整合性制約
- 非正規化

2.3.2.1 主キーの識別

主キーとは、表の中の特定のレコードを一意に識別する属性です。主キーは、単一または複数の列により識別することができます。通常は、可能なかぎり少ない列(理想的には1つか2つ)によって一意の識別を実現することと、更新の可能性の最も低い列または一括で変更されることのない列のいずれかの使用が望ましいとされています。もしデータ・モデルが、その属性を使用した単純な一意の識別につながらない場合は、1つのレコードを一意に識別するために必要な属性が多すぎるか、データが変更されやすいためなので、サロゲート・キーの使用を強くお薦めします。

問合せは多くの表の結合を持つ傾向があり、結合による行の重複を避けるために、レコードを一意に識別するために必要なすべての列が結合条件として必要であるため、3NFスキーマは特に、完全かつ単純な一意の識別に依存します。

2.3.2.2 外部キーの関係および参照整合性制約

データ・ウェアハウス環境における3NFスキーマは、そのOLTPソース・システムのデータ・モデルに通常、類似していますが、そこではデータ・エンティティ間の論理的整合性は、親子関係としても知られる主キーと外部キーの関係によって表現および規定されています。外部キーは、リレーショナル・システムでの1対多の関係を解決し、論理的整合性を保証します。たとえば、注文ヘッダーのない注文明細項目や、存在しない部門で働く従業員を持つことはできません。

そうした参照がOLTPシステムでは常に強制されていますが、データ・ウェアハウス・システムでは通常、これらを強制されない条件である宣言として実装し、ETLプロセスに依存してデータの整合性を保証します。可能な場合は、外部キーと参照整合性制約を強制されない条件として常に定義するようにします。これにより、問合せの最適化およびカーディナリティの見積りを向上させることができます。

2.3.2.3 非正規化

完全な非正規化モデリングでは、論理的なエンティティ(顧客、製品、注文など)が多くの物理表に分解される傾向があり、既知の単純な情報の取得でさえ、多くの表の結合が必要になります。ただし、これは問合せ処理の観点からは問題ではなく、データベース(情報が結合され、常に同時に使用されるため)だけでなく、アプリケーション開発者(コードを記述する)にも不必要な負荷がいくぶん掛かるというだけです。3NFのデータ・ウェアハウス・モデルの非正規化は、論理的な形式ではビューで、物理的な形式では表の非正規化が多少行われるという、ある程度のレベルにとどまることも珍しくありません。

物理的な非正規化には、サブジェクト・ニュートラルな形状を保持するよう注意が必要であり、そのため3NFスキーマの物理的実装には柔軟性があります。

2.4 スター・スキーマについて

スター・スキーマは、埋込み論理データ・マートまたは埋込み物理データ・マートを使用したデータ・ウェアハウス・システムでは、一般的に使用されています。スター・スキーマは、データ・モデルを定義する、ディメンショナル・モデリング法の別名です。ほとんどのディメンショナル・モデリングについての説明は、この分野の先駆的コンサルタントかつライターであるRalph Kimball氏の文献から引用した用語を使用しています。ディメンショナル・モデリングでは複数のスター・スキーマが作成され、その各々が売上追跡や出荷などのビジネス・プロセスに基づいています。各スター・スキーマは1つのデータ・マートと見なすことができ、わずか20個程度のデータ・マートがあれば、1企業のビジネス・インテリジェンスのニーズに対応できます。3NF設計と比較すると、ディメンショナル・モデリングで使用される表の数は微々たるものです。多くのスター・スキーマで保持される表は、1ダース足らずです。スター・スキーマは適合済ディメンションと適合済ファクトを介して一体化されます。したがって、ユーザーは最小限の労力で複数のスター・スキーマからデータを取得できます。

スター・スキーマの目的は、構造の簡素化とデータ取得のパフォーマンス向上です。近年ではほとんどの問合せがレポート・ツールやアプリケーションによって生成されるため、ツールやアプリケーションでの問合せの生成を簡便で信頼性の高いものにすることが不可欠です。実際、多くのビジネス・インテリジェンス・ツールおよびアプリケーションは、スター・スキーマの表現が使用可能なことを見込んで設計されています。

スター・スキーマの説明では、3NFに比べて物理データベースの説明の要約は多くありません。これはディメンショナル・モデリングが実際にビジネス・インテリジェンス・ユーザーのニーズに対して重点を置いているためです。

ディメンショナル・モデリングの形式と、データの冗長性および更新、挿入、削除の非一貫性の危険を最小化する3NFの方法の違いに注意してください。スター・スキーマでは、ユーザーの理解しやすさとデータ取得のよりよいパフォーマンスを意図して、ディメンション表でのデータの冗長性(非正規化)を許容します。スター・スキーマへの一般的な批判は、3NF設計と比較して分析の柔軟性が限定されるというものです。しかし、適切に設計されたディメンショナル・モデルは、新しいタイプの分析が可能になるよう拡張でき、スター・スキーマは最大規模の企業で長年正常に動作しています。

前述のとおり、データ・ウェアハウスの最新の手法は、スター・スキーマと3NFを対抗させるものではありません。むしろ、3NFの基盤レイヤー(3NFのエンタープライズ・データ・ウェアハウス)は基盤データとして、スター・スキーマはアクセスの中核部分およびパフォーマンス最適化レイヤーとしてそれぞれ機能し、2つの技術がともに使用されています。

関連項目:

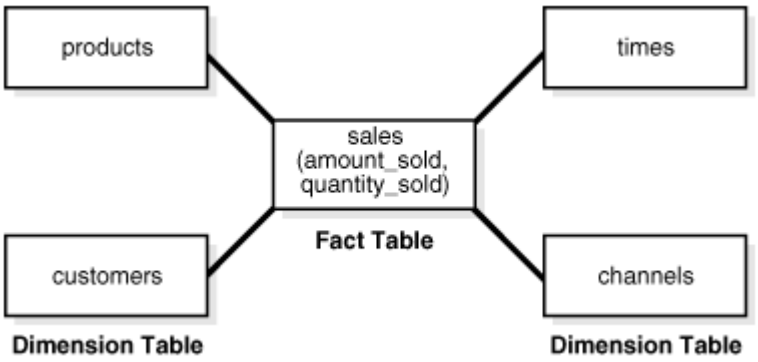
- スター・スキーマのファクトとディメンションについて
- スター・スキーマの設計の概念

2.4.1 スター・スキーマのファクトおよびディメンションについて

スター・スキーマでは、データをファクトとディメンションに分割します。ファクトは売上など、なんらかのイベントの実測値であり、通常は数字です。ディメンションは日付、場所、製品などのファクトの識別に使用するカテゴリです。

スター・スキーマという名前は、スキーマのダイアグラムでは通常、中心のファクト表に付いた線が、ディメンション表に結合しているため、図の印象が星に似ていることに由来します。図2-2は、ファクト表である売上と、ディメンション表である製品、時間、顧客およびチャネルの簡単な例です。

図2-2 スター・スキーマ



関連項目:

- データ・ウェアハウスにおけるファクト表について
- データ・ウェアハウスにおけるディメンション表について

2.4.1.1 データ・ウェアハウスのファクト表について

ファクト表は実測値を保持しています。ファクト表は、多数の行を保持しますが、列は通常多くありません。大企業のファクト表では数十億行を保持することがあります。多くのスター・スキーマでは、ファクト表は全記憶領域の90%を優に超えます。ファクト表には、スキーマのディメンション表の主キーで構成される複合キーがあります。

ファクト表には、ディテール・レベルのファクトまたは集計ファクトのいずれかが含まれます。集計ファクトを含むファクト表は、一般にサマリー表と呼ばれます。通常、ファクト表には同じ集計レベルのファクトが含まれています。ほとんどのファクトは加算的ですが、準加算的なものや非加算的なものもあります。加算ファクトは、単純な加算で集計できます。その最も一般的な例が売上です。非加算ファクトは加算できません。その一例が平均です。準加算ファクトは、ディメンションの一部については集計できますが、他の部分については集計できません。その一例が物理的倉庫に格納された在庫レベルです。倉庫敷地のディメンション全体での加算は可能ですが、時間全体での集計はできません。

ファクト表のデータに行を追加するという観点で、主な方法は次の3つです。

- トランザクションベース

  トランザクションの最も細かなレベルの詳細の行を示します。特定の組合せのディメンションの値についてトランザクションが発生した場合のみ、行が入力されます。これは最も一般的なタイプのファクト表です。

- 定期的スナップショット

  日単位または週単位などの、定期的な時間間隔が終わった時点でデータを示します。前の期間にスナップショットの行が存在する場合は、最後の期間で関連するアクティビティが発生しなくても、そのスナップショットの行が新しい期間に入力されます。このタイプのファクト表は、各トランザクション行からスナップショットの値を計算することが困難な、複雑なビジネス・プロセスで有効です。

- 累積スナップショット

  短期間のプロセスが発生するたびに1行が示されます。行には、短期間のプロセスの主なマイルストンを追跡した、複数の日付が格納されています。他の2つのタイプのファクト表と異なり、累積スナップショットの行は、追跡されたプロセスが前進すると複数回更新されます。

2.4.1.2 データ・ウェアハウスのディメンション表について

ディメンション表はカテゴリ・データを提供し、ファクト・データにコンテキストを与えます。たとえば、売上データのスター・スキーマは、製品、日付、売上場所、宣伝その他のディメンション表を持ちます。ディメンション表の情報は、問合せの制約に使用される値

を選択させるもののため、この表は参照表として機能します。多くのディメンション表の値は、頻繁には更新されません。一例として、市を表す地理のディメンションはほとんど変わりません。しかし、ディメンションの値が変更される場合は、早急かつ確実に更新することが不可欠です。当然、データ・ウェアハウスのディメンションの値が頻繁に変更される場合もあります。企業の顧客ディメンションでは、頻繁な更新ストリームおよび削除ストリームに確実に左右されます。

ディメンション表の重要な側面は、提供される階層情報にあります。ディメンションのデータには通常、最下位レベルの詳細の行と、集約されたディメンション値の行があります。このようなディメンション表内でのロールアップまたは集計は階層と呼ばれ、分析に高い価値を付加します。たとえば、特定の製品が示す、特定の製品カテゴリ内の特定の製品の売上高の割合を計算するとします。各問合せに製品カテゴリのすべての要素を指定するよりは、製品集計用の定義済階層を持つ方が、はるかに容易で信頼性が高くなります。階層情報は非常に価値が高いため、一般的に複数の階層がディメンション表に反映されます。

通常、ディメンション表は説明的なテキストであり、その値は問合せで生成されたレポートの行ヘッダー、列ヘッダーおよびページ・ヘッダーとして使用されます。ディメンション表はファクト表に比べ、はるかに少ない行を保持しますが、非常に幅広くなり、数十列になることがあります。場所のディメンション表は、ロールアップ階層のすべてのレベルを示す列を持ち、表に反映された複数の階層を表します。場所のディメンション表には、街の住所、郵便番号、市、州または県および国などの、地理的ロールアップ用の列があります。同じ表に、販売組織用に設定されたロールアップ階層を含め、販売地区、販売地域、販売地方および特徴に対する各列を持たせることができます。

関連項目:

ディメンションの詳細は、ディメンションを参照してください。

2.4.2 スター・スキーマの設計の概念

ここではスター・スキーマで使用される、主要な用語のいくつかに触れます。すべてを説明するわけではなく、検討の価値のある分野の一部について取り上げます。

データの粒度

モデルを設計する際に最も重要なタスクの一つは、データの粒度と呼ぶ、モデルが提供する詳細さのレベルを検討することです。売上スキーマについて検討してみます。各顧客が購入した個々の品目を格納するほど粒度を細かくするべきでしょうか。それとも粒度を粗くし、各店舗で各製品についての日々の売上合計のみを格納すればよいでしょうか。最新のデータ・ウェアハウスでは、分析力を最大限にするため、粒度の非常に細かいデータの提供に特に重点が置かれています。ディメンショナル・モデリングのエキスパートは一般に、各ファクト表が格納する粒度のレベルは1つのみとすることを推奨しています。単一粒度の表でファクト・データを提供することにより、ファクト表のすべての行の適用範囲に曖昧性がなくなるため、より信頼性の高い問合せおよび表のメンテナンスに対応できます。

複数スター・スキーマでの処理

スター・スキーマの設計方法はデータを異なるプロセスに区分化することを意図しているため、問合せが複数スキーマにまたがった場合、スキーマを横断するための信頼性およびパフォーマンスの高い方法が必要です。この機能を表す用語がデータ・ウェアハウス・バス・アーキテクチャです。データ・ウェアハウス・バス・アーキテクチャは、適合済ディメンションおよび適合済ファクトによって実現可能です。

適合済ディメンション

適合済ディメンションとは、ディメンションが様々なスター・スキーマにわたって同一になるよう設計されていることです。適合済ディメンションでは同一の値、列、名前およびデータ・タイプを、複数のスターにわたり一貫して使用します。適合済ディメンションでは、小さい表の行が大きい表の真のサブセットであるかぎり、ディメンション表の各スキーマのコピー間で同一の行数である必要はあり

ません。

適合済ファクト

複数のファクト表内でファクト列がまったく同じ内容の場合、適合済ファクトとみなされます。このようなファクトは異なる表からのものであっても、計算上確実なものとして組み合せて使用できます。適合済ファクトは、適合状態を示すため、同一の列名にする必要があります。適合されていないファクトは、内容が異なることを示すため、常に異なる名前にする必要があります。

サロゲート・キー

サロゲート・キー、すなわち人工キーは、通常連続した整数で、ディメンション表で推奨されています。サロゲート・キーの使用により、操作の変更からデータが隔離されます。また、小型の整数キーでは、大型で複雑な英数字キーに比べ、よりよいパフォーマンスが期待できます。

逆ディメンション

逆ディメンションとは、ファクト表のディメンション列がディメンション表に結合されていないものです。通常は、注文番号やインボイス番号などの項目です。ファクト表の粒度が注文明細項目または単一トランザクションのレベルの場合に見られます。

ジャンク・ディメンション

ジャンク・ディメンションは、ファクト表のフラグおよびコードのテキスト参照値を保持するため使用される、抽象的ディメンション表です。これらのディメンションをジャンクと呼びますが、価値が低いからではなく、利便性のため各種の列を保持していることから、ガラクタの入った引き出し(junk drawer)にたとえられるからです。ジャンク・ディメンション表における各列の個別値(カーディナリティ)は通常、少数です。

埋込み階層

スター・スキーマを使用した従来のディメンショナル・モデリングでは、各表が単一粒度のデータを格納することを推奨していました。ただし、設計者が表に複数の粒度を格納することを選択する場合があり、これらは通常ロールアップ階層を表します。たとえば、単一の売上ファクト表は、トランザクションレベルのデータと、日レベルの製品別ロールアップおよび月レベルの製品別ロールアップの両方で構成されます。この場合、ファクト表は各行に適用する階層レベルを示す、レベル列を持つ必要があり、この表に対する問合せにはレベルの述語が必要になります。

ファクトレス・ファクト表

ファクトレス・ファクト表には、販売価格や販売数量などのメジャーを含みません。そのかわり、ファクトレス・ファクト表の行は、他のファクト表では表されないイベントを示すために使用されます。ファクトレス・ファクト表のもうひとつの使用法は、有効範囲表です。全製品が販売促進用で、宣伝価格で販売された場合などの特定の状況で発生した、考えられるすべてのイベントを保持します。

緩やかに変化するディメンション

データ・ウェアハウスで確実なことの1つは、データの分類方法が変更されることです。製品名やカテゴリ名は変更されるものです。店舗の特徴も変わります。販売地域に含まれるエリアも変わります。これらの変更のタイミングや程度は必ずしも予測できません。こうした緩やかに変化するディメンションには、どのように対処すればよいのでしょうか。スター・スキーマにより、これらに対処する主な方法は、次の3つです。

- タイプ1

  変更されたディメンションの値を、履歴を残さず単純に上書きします。この場合、時間ベースの分析に問題が発生します。また、ディメンションの古い値に依存する、既存の集計がすべて無効になります。

- タイプ2

  ディメンションの値が変更されると、新規のディメンション行が変更後の値を示し、新しいサロゲート・キーが作成されます。新しい行が有効になった場合およびその行が期限切れになった場合、ディメンションに日付列を含めるか選択できます。

ファクト表を変更する必要はありません。

- タイプ3

  ディメンションの値が変更された場合、以前の値は同じ行の別の列に格納されます。これにより、列の現在の値と以前の値を使用して結果を比較する場合の問合せの生成が簡単になります。

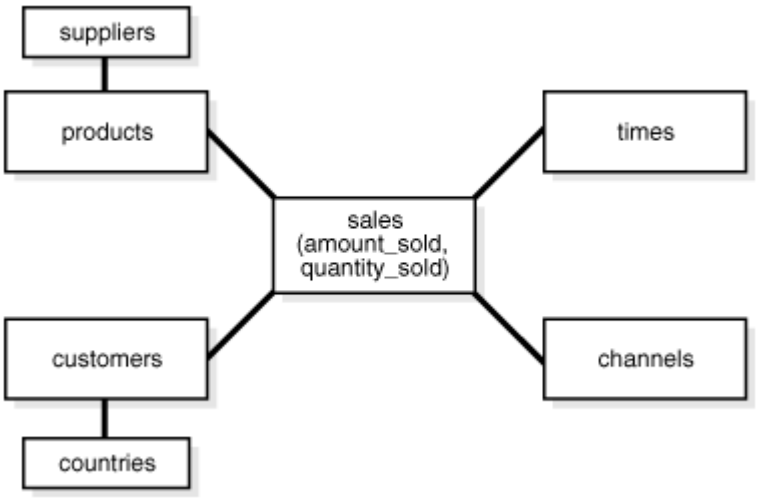
実際には、緩やかに変化するディメンションの対処としては、タイプ2が最も一般的です。

2.4.3 スノーフレーク・スキーマについて

スノーフレーク・スキーマは、スター・スキーマより複雑なデータ・ウェアハウス・モデルであり、スター・スキーマの一種です。このスキーマの図がスノーフレーク(雪片)に似ているため、スノーフレーク・スキーマと呼ばれます。

スノーフレーク・スキーマでは、ディメンションが正規化され、冗長性が排除されます。つまり、ディメンション・データは1つの大規模な表ではなく複数の表にグルーピングされます。たとえば、スター・スキーマのproductディメンション表は、スノーフレーク・スキーマのproducts表、product_category表およびproduct_manufacturer表に正規化できます。これによって、領域が節約されますが、ディメンション表の数が増加し、より多くの外部キー結合が必要になります。その結果、問合せがより複雑になり、問合せのパフォーマンスが低下します。図2-3に、スノーフレーク・スキーマを示します。

図2-3 スノーフレーク・スキーマ



2.5 インメモリー列ストアを使用した分析の改善

インメモリー列ストア(IM列ストア)は任意のシステム・グローバル領域(SGA)です。ここには、迅速にスキャンできるように最適化された圧縮列形式で、表、表のパーティションおよびその他のデータベース・オブジェクトのコピーが格納されています。

列形式はベクトル処理に適しているため、集計、結合、およびいくつかの種類のデータ取得が、従来のディスク上の形式より高速になります。列形式はメモリーの中にのみ存在するもので、ディスク上の形式やバッファ・キャッシュ形式の代替にはなりません。むしろバッファ・キャッシュを補完するものであり、ディスク形式から独立している、トランザクションと整合的な表の追加コピーです。

従来の分析では、分析の問合せで良好なパフォーマンスを得るために対処する必要がある特定の制限または必要条件を持っています。ユーザーのアクセス・パターンを把握し、そのアクセス・パターンでパフォーマンスが最高になるようにデータ構造をカスタマイズする必要があります。既存の索引、マテリアライズド・ビューおよびOLAPキューブを調整する必要があります。いくつかのデータ・マートとレポーティング・データベースには複合ETLがあるため、特別の調整が必要です。さらに、失効したデータに関する分析の実行と、本番データベース上でのOLTP操作のスローダウンのバランスを最適化する必要があります。

Oracle Databaseの範囲内で Oracle In-Memory Column Store(IM列ストア)を活用すると、非定型の問合せとライブ・データの分析のパフォーマンスが向上します。ライブ・トランザクション・データベースは問合せに瞬時に応答するために使用され、それにより、OLTPトランザクションとデータ・ウェアハウス分析に、同じデータベースをシームレスに使用することが可能になりま

す。

IM列ストアは、Oracle Databaseとシームレスに統合され、データ・ウェアハウス環境に次の利点をもたらします。

- 問合せパフォーマンスの向上
  - 予期しないアクセス・パターンの非定型の問合せの処理が速くなります。

    IM列ストアは、大量データの分析の際のスループットを高速化できます。表の列のサブセットを問い合せると、特定のデータ分析タスクに必要な列のみがスキャンされるため、結果がすばやく取得されます。
  - 大量の行のスキャン、および=、<、>およびINなどの演算子を使用するフィルタの適用は、SIMDベクトル処理を使用すると速くなります。
  - 頻繁に評価される式をIM式を使用して格納すると、同じ式を繰り返して計算することが少なくなります。
  - IM仮想列を使用して、指定した仮想列をIM列ストアに移入すると、仮想列を繰り返して評価することが回避されます。
- ブルーム・フィルタを使用した強化された結合パフォーマンス

  特定の種類の結合は、結合される表がIM列ストアに格納されている場合、より高速に実行されます。IM列ストアは、小さいディメンション表に対する述語を大きいファクト表に対するフィルタに変換することによって結合を高速化するハッシュ結合を使用するブルーム・フィルタを活用しています。
- VECTOR GROUP BY変換とベクトル配列処理の使用による効率的な集計。

  データを集計して1つ以上の比較的小さい表をより大きい表に結合する問合せ(スター・クエリーでよく行われます)がより速く実行されます。VECTOR GROUP BYは、オプティマイザによって、コスト見積もりに基づいて選択されます。
- IM列ストア使用に伴う必要な索引、マテリアライズド・ビューおよびOLAPキューブの減少による、記憶域と処理オーバーヘッドの大幅な低下。

関連項目:

IM列ストアの使用の詳細は、Oracle Database In-Memoryガイドを参照してください。

2.5.1 インメモリー式を使用した問合せパフォーマンスの改善について

インメモリー列ストア(IM列ストア)を使用すると、頻繁に評価される式にインメモリー式(IM式)を使用することによって問合せパフォーマンスをさらに向上させることができます。

データ・ウェアハウス環境のほとんどの問合せでは、大きいデータ・セットへの問合せが行われ、複雑な式や計算が含まれているため計算集約的です。IM式を使用すると、頻繁に評価される式を含む問合せのパフォーマンスが向上します。オプティマイザは、繰返し使用される式を自動的に識別して式統計ストア(ESS)に記録します。ESSに取得された式はIM式の候補となります。再使用を促進するには、IM式をマテリアライズしてIM列ストア内のインメモリー式単位(IMEU)に移入します。データベースは、IM式を保守し、それらの式が基にしているソース列への変更と整合性が保たれるようにします。IM式をIM列ストアに移入すると、同じ式が繰り返して計算されることが少なくなります。

たとえば、価格と販売数の積である売上金額はIM式の候補です。IM式を使用しない場合は、すべての問合せおよび問合せによって返されるすべての行で、売上金額の値を再計算する必要があります。IM式を使用する場合は、この頻繁に評価される式をマテリアライズしてIM列ストアに格納できます。これにより、問合せに使用される式を繰返し再計算する必要がなくなります。Oracle Databaseは、IM列ストアに格納されている式の結果を使用するように実行時に問合せをリライトし、問合せのパ

フォーマンスを向上させます。

初期化パラメータINMEMORY_EXPRESSIONS_USAGEは、IM列ストアに移入する必要があるIM式を制御します。DBMS_INMEMORY_ADMINパッケージのプロシージャは、IM式を識別、移入および使用するタイミングを指定します。

関連トピック

- 『Oracle Database In-Memoryガイド』

2.5.2 インメモリー仮想列を使用した問合せパフォーマンスの改善について

インメモリー列ストア(IM列ストア)を使用すると、インメモリー仮想列(IM仮想列)を使用して、指定した仮想列をIM列ストアに移入することによって、仮想列を繰り返して評価することを回避できます。

仮想列は、ユーザーが作成した名前付きの式であり、Oracleによって通常の列のように扱われます。たとえば、SALARY表に列monthly_salaryが含まれている場合は、annual_salaryという仮想列をmonthly_salary * 12として定義できます。IM仮想列は、IM列ストアに移入できる仮想列です。表に定義されている仮想列のすべてまたはサブセットをIM列ストアに移入できます。再計算された仮想列をIM列ストアに格納すると、評価を繰り返すことが回避され、問合せパフォーマンスが向上します。仮想列値は、SIMDベクトル処理などのインメモリー・テクニックを使用してスキャンおよびフィルタ処理することもできます。

初期化パラメータINMEMORY_VIRTUAL_COLUMNSは、IM列ストアが有効にされている表のIM仮想列を作成するかどうかを決定します。

関連トピック

- 『Oracle Database In-Memoryガイド』

2.5.3 インメモリー列ストアと自動データ最適化について

自動データ最適化(ADO)を使用すると、インメモリー列ストア(IM列ストア)のコンテンツを管理できます。

IM列ストアによって提供されるパフォーマンスの利点は、IM列ストアのコンテンツを効果的に管理することによって最適化できます。IM列ストアに格納されることで最もメリットがあるオブジェクトを保持する必要があります。IM列ストアに保持する必要があるオブジェクト、および削除する必要があるオブジェクトを判別するために、IM列ストアを常に監視する必要があります。

自動データ最適化(ADO)は、IM列ストアのコンテンツの管理を自動化します。IM列ストア内のオブジェクトについてヒート・マップ統計が収集され、最もアクティブではないオブジェクトおよび最もアクティブなオブジェクトを判別するために、それらの統計が使用されます。ADOポリシーを定義すると、IM列ストアからオブジェクトを削除するタイミングを指定できます。

データ・ウェアハウス・アプリケーションでは、通常、オブジェクトがアクセスされる頻度は時間の経過に従って減少します。このため、オブジェクトは最初にデータ・ウェアハウスにロードされたときに最も頻繁にアクセスされ、アクティビティ・レベルは徐々に減少します。データ・ウェアハウスのパフォーマンスは、最もアクセスが少ないオブジェクトをIM列ストアから削除するADOポリシーを定義することによって改善できます。

関連トピック

- 『Oracle Database In-Memoryガイド』

2.6 大きな表の自動キャッシングによるインメモリー・パラレル問合せのパフォーマンス向上

大きな表の自動キャッシングは、インメモリー・パラレル問合せのパフォーマンスを向上させます。表がメモリー内に収まらない場合、データベースはアクセス・パターンに基づいてキャッシュするバッファを決定します。大きな表がバッファ・キャッシュに完全におさまらない場合も、これにより効率的なキャッシュが可能になります。

表スキャンのデータを格納するためにはバッファ・キャッシュのオプションの1セクションが使用され、それは「大きな表のキャッシュ」と呼ばれます。大きな表のキャッシュはバッファ・キャッシュと統合され、温度ベースでオブジェクト・レベルの置換アルゴリズムを使用して、大きな表のキャッシュ内容を管理します。これは、バッファ・キャッシュが使用する、アクセス・ベースでブロック・レベルのLRUアルゴリズムとは異なります。

ノート:

大きな表の自動キャッシングは、Oracle Database 12c リリース 1(12.1.0.2)以上で使用できます。

一般的なデータ・ウェアハウジング・ワークロードでは、複数の表がスキャンされます。これらの表の合計サイズがバッファ・キャッシュの合計サイズより大きい場合、パフォーマンスに影響が生じる場合があります。大きな表の自動キャッシングでは、スキャンされた表は、バッファ・キャッシュでなく大きな表のキャッシュに格納されます。大きな表のキャッシュで使用される温度ベースでオブジェクト・レベルの置換アルゴリズムでは、次のようなデータ・ウェアハウジング・ワークロードのパフォーマンスを向上させることができます。

- 「ホット」オブジェクトの選択的なキャッシュ格納

  オブジェクトがアクセスされるたびに、Oracle Databaseはそのオブジェクトの温度をインクリメントします。大きな表のキャッシュにあるオブジェクトは、その温度より高い温度を持つ別のオブジェクトとのみ置換できます。

- スラッシングの回避

  オブジェクトを完全にはキャッシュに格納できない場合、オブジェクトの一部をキャッシュに格納します。

Oracle Real Application Clusters(Oracle RAC)環境では、大きな表の自動キャッシングは、パラレル問合せについてのみサポートされます。単一インスタンス環境では、この機能は、シリアル問合せとパラレル問合せの両方についてサポートされます。

大きな表の自動キャッシングを使用するには、大きな表のキャッシュを有効化する必要があります。連続する問合せに対して大きな表の自動キャッシングを使用するには、DB_BIG_TABLE_CACHE_PERCENT_TARGET初期化パラメータをゼロ以外の値に設定する必要があります。パラレル問合せに対して大きな表の自動キャッシングを使用するには、PARALLEL_DEGREE_POLICYをAUTOまたはADAPTIVEに、そしてDB_BIG_TABLE_CACHE_PERCENT_TARGETをゼロ以外の値に設定する必要があります。

関連項目:

大きな表のキャッシュとその使用方法の詳細は、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照してください。

3 データ・ウェアハウスの物理設計

この章では、データ・ウェアハウス環境の物理設計について説明します。内容は次のとおりです。

- 論理設計から物理設計への変換
- 物理設計について

3.1 論理設計から物理設計への変換

論理設計とは、データ・ウェアハウスを作成する前に、設計を紙に書いたり、Oracle Designerなどのツールで設計したりすることです。物理設計とは、SQL文でデータベースを作成することです。

物理設計では、論理設計時に収集したデータを、物理データベース構造の記述に変換します。物理設計上の決定事項には、主に問合せのパフォーマンスやデータベースのメンテナンスが影響します。たとえば、問合せ要件に適したパーティション化を行うと、実行前に検索対象を絞り込むパーティション・プルーニングをOracle Databaseで活用できます。

関連項目:

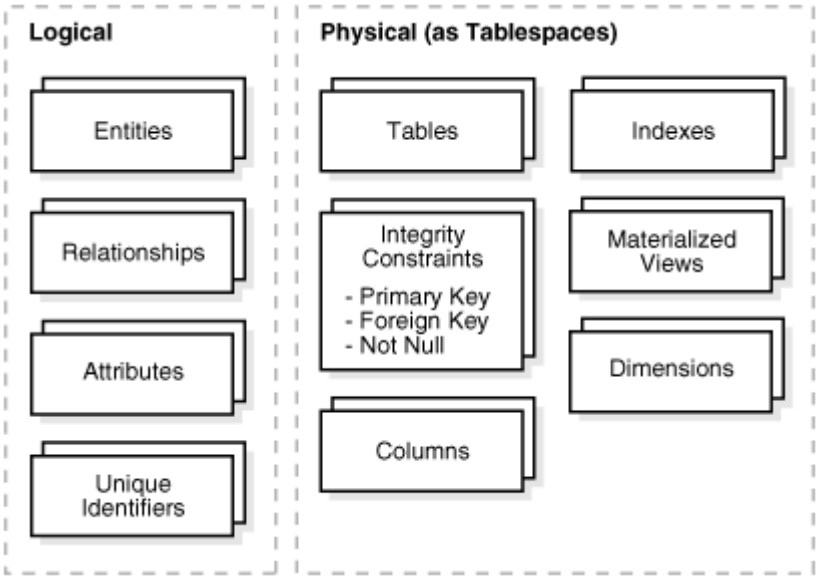
- パーティション化の詳細は、Oracle Database VLDBおよびパーティショニング・ガイドを参照してください。
- 設計事項の概念情報の詳細は、Oracle Database概要を参照してください。

3.2 物理設計について

論理設計の段階で、エンティティ、属性およびリレーションシップで構成されるデータ・ウェアハウスのモデルが定義されています。エンティティは、リレーションシップを使用して相互にリンクされます。属性は、エンティティの説明に使用します。一意識別子(UID)により、エンティティの1つのインスタンスとその他のインスタンスが区別されます。

図3-1に論理設計と物理設計の違いを示します。

図3-1 論理設計と物理設計の比較



物理設計では、必要となるスキーマを実際のデータベース構造に変換します。ここで、次のマッピングを行う必要があります。

- エンティティから表へ

- リレーションシップから外部キー制約へ
- 属性から列へ
- 1次一意識別子から主キー制約へ
- 一意識別子から一意キー制約へ

3.2.1 物理設計の構造

論理設計を物理設計に変換するには、表領域、表、(表または索引構成表の)パーティション、索引(パーティション索引を含む)、ビュー、整合性制約、マテリアライズド・ビューおよびディメンションといった構造の一部または全部を作成する必要があります。

3.2.1.1 データ・ウェアハウスの表領域について

表領域は、使用中のオペレーティング・システム内の物理構造である1つ以上のデータファイルで構成されます。データファイルは、1つの表領域にのみ対応付けられています。設計の観点からは、表領域は物理設計構造のコンテナです。

表領域は、相違点によって分離する必要があります。たとえば、表と索引、小規模な表と大規模な表は分離する必要があります。また、表領域は、可能であれば論理的なビジネス・ユニットを表す必要があります。表領域は、バックアップやリカバリ、またはトランスポータブル表領域メカニズムのための最も大きな単位であるため、論理的なビジネス設計は可用性とメンテナンス操作に影響します。

現在では、巨大なデータファイルを使用でき、非常に大きなデータベースでのパフォーマンスが大幅に改善されています。

3.2.1.2 データ・ウェアハウスにおけるパーティション化について

Oracle Partitioningは、データ・ウェアハウジングの非常に重要な機能であり、管理性、パフォーマンスおよび可用性を改善します。この項では、パーティション化の重要な概念およびそのメリットについて、データ・ウェアハウスにとっての特別な意味に触れながら説明します。

パーティション化により、表、索引および索引構成表をより細かい単位に細分化できるようになります。データベース・オブジェクトの単位をパーティションと呼びます。各パーティションには独自の名前があり、独自の記憶特性を持つことができます。データベース管理者の視点からすると、パーティション・オブジェクトには、まとめて管理することも個別に管理することも可能な複数の単位があります。このため、管理者は、パーティション・オブジェクトをかなり柔軟に管理できます。ただし、ユーザーにとっては、パーティション表は非パーティション表と同じであるため、SQLおよびDMLコマンドを使用してパーティション表にアクセスする際に変更は必要ありません。

表、索引および索引構成表などのデータベース・オブジェクトは、パーティション化キーを使用してパーティション化されます。パーティション化キーは、指定された行がどのパーティションに存在するかを決定する、一連の列です。たとえば、売上表は月次のパーティション化方法を使用して売上日でパーティション化されています。この表は単一かつ通常の表としてどのアプリケーションにも表示されます。ただし、DBAは、場合によっては異なる記憶域の層を使用し、古いデータに対して表の圧縮を適用して各月次パーティションを個別に管理および格納したり、古いデータの領域全体を読取り専用表領域に保管できます。

3.2.1.2.1 データ・ウェアハウスで使用される基本的なパーティション化戦略

Oracle Partitioningには、様々な個別のパーティションに実際にデータを配置する方法を制御する、次の3つの基本的なデータ分散方法が用意されています。

- 範囲

  データはパーティション化キーの値のレンジに基づいて分散されます(日付列がパーティション化キーの場合、January-2012パーティションには、パーティション化キーの値が01-JAN-2012と31-JAN-2012の間の行が含まれます)。データ分散は切れ目のない連続体であり、前のレンジの上限により、レンジの下限が自動的に定義されます。

- リスト

  データ分散は、パーティション化キーの値のリストによって定義されます(地方列がパーティション化キーの場合、North_Americaパーティションには Canada、USAおよびMexicoが含まれます)。特別なDEFAULTパーティションを定義すると、リストで明示的に定義されていないパーティション・キーのすべての値を捕捉できます。

- ハッシュ

  ハッシュ・アルゴリズムは、特定行のパーティションを決定するため、パーティション化キーに対して適用されます。他の2つのデータ分散方法と異なり、ハッシュはデータとパーティション間のいかなる論理マッピングも提供しません。

これらの基本的な方法の他に、次の複数の方法が提供されています。

- 時間隔パーティション化

  管理性を向上させるレンジ・パーティション化の拡張機能です。パーティションが等しい範囲の間隔で定義されます。最初のパーティションを除いたすべてのパーティションは、一致するデータの到着時に要求に応じて自動的に作成されます。

- 参照別のパーティション化

  子表のパーティション化は、主キーと外部キーの関係を通し、親表から継承されます。パーティションのメンテナンスが簡略化され、パーティション・ワイズ結合が使用可能になります。

- 仮想列ベースのパーティション化

  前述のパーティション技術のいずれか1つにより定義され、パーティション化キーは仮想列に基づきます。仮想列はディスクには格納されず、メタデータとしてのみ存在します。この方法により、ビジネス要件により柔軟かつ幅広く対応できます。

前述のデータ分散方法を使用して、単一またはコンポジット・パーティション表として表をパーティション化できます。

- 単一(1レベル)パーティション化

  1つ以上の列をパーティション化キーとして使用し、データ分散方法のいずれかを指定することで表を定義します。たとえば、数値列をパーティション化キーとする表があるとします。2つのパーティションless_than_five_hundredおよびless_than_thousandがあり、less_than_thousandパーティションには、500 <= Partitioning key <1000の条件が当てはまる行が含まれるとします。

  レンジ、リストおよびハッシュの各パーティション表を指定できます。

- コンポジット・パーティション化

- コンポジット・パーティション表の定義には、2つのデータ分散方法の組合せが使用されます。まず、データ配分方法1で表をパーティション化し、その後2つ目のデータ配分方法を使用して、各パーティションをさらに小さなサブパーティションに再分割します。特定のパーティションのサブパーティションをすべて組み合せて、データの論理サブセットを表します。たとえば、レンジ-ハッシュ・コンポジット・パーティション表は、最初にレンジ・パーティション化されます。その後、各レンジ・パーティションがハッシュ・パーティション化技術を使用して、さらにサブパーティション化されます。

関連項目:

- 『Oracle Database VLDBおよびパーティショニング・ガイド』
- ハイブリッド列圧縮の詳細は、『Oracle Database概要』を参照してください。

3.2.1.3 データ・ウェアハウスにおける索引のパーティション化

索引のパーティション化方法の選択に関係なく、基礎となる表のパーティション化方法により索引が結合されるか、結合解除さ

れます。適切な索引パーティション化方法は、ビジネス要件に基づいて選択され、各種アプリケーションのサポートに最適なパーティション化を実現します。Oracle Database 12cでは3タイプのパーティション索引を区別します。

- ローカル索引

  ローカル索引は、パーティション表の索引であり、基礎となるパーティション表に結合され、表のパーティション化方法を継承します。そのため、ローカル索引の各パーティションは、基礎となる表の唯一のパーティションに対応します。結合により、パーティションのメンテナンスの最適化が可能です。たとえば、表のパーティションが削除された場合、Oracle Databaseではそれに対応する索引のパーティションを削除するだけで対応できます。費用のかかる索引のメンテナンスは必要ありません。データ・ウェアハウジング環境では、ローカル索引が最も一般的です。

- グローバル・パーティション索引

  グローバル・パーティション索引は、パーティション表または非パーティション表の索引であり、表とは異なるパーティション化キーまたはパーティション化方法でパーティション化されます。グローバル・パーティション索引では、レンジまたはハッシュの各パーティション化を使用してのパーティション化および基礎となる表との結合解除が可能です。たとえば、表を月別にレンジ・パーティション化し、12のパーティションを持たせることが可能ですが、その表の索引を別のパーティション化キーを使用してハッシュ・パーティション化し、異なる数のパーティションを持たせることが可能です。グローバル・パーティション索引は、データ・ウェアハウジング環境よりはOLTP環境で一般的です。

- グローバル非パーティション索引

  グローバル非パーティション索引は、基本的に非パーティション表の索引と同じものです。索引の構造はパーティション化されておらず、基礎となる表と結合されていません。データ・ウェアハウジング環境での、グローバル非パーティション索引の最も一般的な使用方法は、主キーの制約の施行です。

3.2.1.4 管理性のためのパーティション化について

管理性を向上させるための一般的なパーティション化の使用方法は、データ・ウェアハウスでローリング・ウィンドウ・ロード・プロセスをサポートすることです。DBAが新規データを日単位で表にロードするとします。各パーティションに1日分のデータが保存されるように、その表をレンジ・パーティション化できます。ロード・プロセスは単純に新しいパーティションを追加するだけです。DBAがその他のパーティションを変更する必要がないため、単一のパーティションの追加は、表全体を変更するよりはるかに効率的です。パーティション化を使用する別のメリットは、データを削除する場合にあります。この場合、パーティション全体の削除が可能で、各行を個別に削除する場合に比べて非常に効率的かつ高速です。

3.2.1.5 パフォーマンスのためのパーティション化について

調査または操作対象のデータ量を制限することで、パーティション化によりパフォーマンスに関連する多くの利点がもたらされます。次の2つの機能は特に注目に値します。

- パーティション・プルーニング: パーティション・プルーニングは非常に簡単で、パーティション化を使用してパフォーマンスを向上するための最も実質的な方法でもあります。パーティション・プルーニングにより、問合せのパフォーマンスが大幅に向上します。たとえば、アプリケーションに注文の履歴レコードを含むORDERS表があり、この表が日ごとにパーティション化されているとします。1週間の注文をリクエストする問合せでは、ORDERS表の7つのパーティションにのみアクセスします。表に2年分の履歴データがある場合、この問合せでは730のパーティションではなく7つのパーティションにアクセスします。この問合せは、単純にパーティション・プルーニングの効果で、100倍高速に実行される可能性があります。パーティション・プルーニングは、Oracle製品のその他すべてのパフォーマンス機能と連携します。パーティション・プルーニングは、索引や結合の技術またはパラレル・アクセスの手法と組み合せて使用されます。

- パーティション・ワイズ結合: パーティション化では、パーティション・ワイズ結合と呼ばれる技術が使用され、複数表の結合のパフォーマンスも向上します。パーティション・ワイズ結合は、2つの表を結合する際、少なくとも表の1つが結合キーでパーティション化される場合に適用できます。パーティション・ワイズ結合では、大きな結合が結合された表と同一の

データセットの小さな結合に分割されます。ここでいう同一とは、まったく同一である一連のパーティション化キー値を結合の両側で扱うということです。こうすることにより、これら同一のデータセットの結合のみが結果を生成し、他のデータセットを考慮する必要がないことが保証されます。すでに(物理的に)等しくパーティション化されている結合の表のファクトを使用するか、あるいは実行時に1つの表を透過的に再分割する(再パーティション化する)ことにより、他の表のパーティションに一致する等パーティション化データセットが作成され、全体的な結合が少ない時間で完了します。これにより、シリアル実行およびパラレル実行の両方でパフォーマンスが大幅に向上します。

3.2.1.6 可用性のためのパーティション化について

パーティション化データベース・オブジェクトにより、パーティションの独立性が実現されます。パーティションの独立性のこの特性は、高可用性計画の重要な要素の一部です。たとえば、パーティション表の1つのパーティションが使用できなくなっても、その表のその他すべてのパーティションはオンラインで使用可能なままです。アプリケーションは、そのパーティション表に対して問合せやトランザクションを実行し続けることができ、使用できないパーティションにアクセスする必要がなければ、それらのデータベース操作は正常に実行されます。データベース管理者は、各パーティションが別々の表領域に格納されるよう指定できます。これにより、表内の他のパーティションとは無関係に、個別のパーティションまたは一連のパーティションでのバックアップおよびリカバリ操作が可能になります(パーティションと表領域のマッピングによる)。したがって、万一障害の場合でも、アクティブ・データを構成するパーティションのみでデータベースのリカバリが可能です。その後、都合のよいときに他のパーティションにある非アクティブのデータをリカバリでき、システムの停止時間を削減できます。管理性、パフォーマンスおよび可用性の各メリットを考慮すると、あらゆるデータ・ウェアハウスで採用する必要があります。

関連項目:

『Oracle Database VLDBおよびパーティショニング・ガイド』

3.2.2 データ・ウェアハウスでのビューについて

ビューは、1つ以上の表または他のビューに含まれるデータが、整形された形で表されているものです。ビューは、問合せの出力を取得し、表として処理します。データベースの領域は必要としません。

関連項目:

Oracle Database概要

3.2.3 データ・ウェアハウスでの整合性制約について

整合性制約は、データベースに関連したビジネス・ルールを規定し、表中の無効な情報を防止するために使用されます。データ・ウェアハウスにおける整合性制約は、OLTP環境における制約とは異なります。OLTP環境では、主として無効なデータがレコードに挿入されるのを防止しますが、データ・ウェアハウス環境では、正確さがすでに保証されているため、これは大きな問題ではありません。データ・ウェアハウス環境では、制約はクエリー・リライトにのみ使用されます。NOT NULL制約は、データ・ウェアハウスでは特に一般的です。ある特定の状況では、制約はデータベースの領域を必要とします。このような制約は、基礎となる一意索引の形式になっています。

関連項目:

3.2.4 データ・ウェアハウスでの索引およびパーティション化索引について

索引は、表またはクラスタに関連付けられたオプションの構造体です。データ・ウェアハウス環境では、従来のBツリー索引に加えて、ビットマップ索引がきわめて一般的です。ビットマップ索引は、セット指向の操作に最適化された索引構造です。また、スター型変換など、最適化された一部のデータ・アクセス方法にも必要になります。

索引は、パーティション化できるという点では表と同じですが、パーティション化方法は表の構造に依存しません。索引をパーティション化すると、リフレッシュ時にデータ・ウェアハウスを管理しやすくなり、問合せのパフォーマンスが改善されます。

関連項目:

- データ・ウェアハウスにおける索引のパーティション化
- Oracle Database概要

3.2.5 データ・ウェアハウスでのマテリアライズド・ビューについて

マテリアライズド・ビューには、実行に時間のかかるSQL文の計算が不要になるように、事前の問合せ結果が格納されています。物理設計の観点からすると、マテリアライズド・ビューは表やパーティション表に似ており、透過的に使用されパフォーマンスが向上するという点で、索引のような動作をします。

関連項目:

基本的なマテリアライズド・ビュー

3.2.6 データ・ウェアハウスでのディメンションについて

ディメンションは、データを分類する1つの構造であり、1つ以上の階層から構成されています。ディメンション属性を使用して、ディメンション値を記述できます。通常、ディメンションは説明的なテキスト値です。ファクトと結合した複数の独立したディメンションにより、ビジネス上の問題に対応できます。最も一般的なディメンションは、顧客(customers)、製品(products)および時間(times)です。

ディメンション・スキーマ・オブジェクトは、列または列セット間の階層関係を定義します。階層関係とは、階層内のあるレベルのデータが次のレベルのどのデータに結びつくかという依存関係を指します。ディメンション・オブジェクトは論理関係のコンテナであり、データベースに領域を必要としません。典型的なディメンションには、市、州(または県)、地域および国などがあります。

ディメンション・データは、通常、詳細な最下位レベルで収集されてから、分析に有効な上位レベルの合計に集計されます。このようなディメンション表内でのロールアップまたは集計を階層と呼びます。

この項では、次の項目について説明します。

- ディメンション階層について
- 典型的なディメンション階層

3.2.6.1 ディメンション階層について

階層は、データを構成する順序付けられたレベルを使用する論理構造です。階層は、データ集計の定義に使用できます。たとえば、時間ディメンションであれば、月レベルから四半期レベル、さらに年レベルへとデータを集計する階層を定義できます。階層は、ナビゲーショナル・ドリル・パスの定義およびファミリ構造の作成にも使用できます。

階層内では、各レベルがその上下のレベルと論理的に接続されます。下位レベルのデータ値は、上位レベルのデータ値に集計されます。ディメンションは複数の階層で構成できます。たとえば、製品ディメンションには、2つの階層(製品カテゴリおよび製品仕入先)がある場合があります。

また、ディメンション階層は、要約レベルから詳細レベルまでカテゴリ化されます。階層は、問合せツールによって使用されます。これにより、データをドリルダウンし、詳細度の異なるレベルを参照できるようになります。これは、データ・ウェアハウスの主なメリットの1つです。

階層を設計する場合は、ビジネス構造におけるリレーションシップを考慮する必要があります。たとえば、多階層の部門を持つ販売組織では、構造が複雑になりがちです。

階層では、ディメンション値はファミリ構造で表現されます。あるレベルの値に対して、1つ上のレベルの値はその親になり、1つ下のレベルの値はその子になります。これらのファミリ関係によって、アナリストはデータに迅速にアクセスできます。

関連項目:

- レベルについて
- レベルの関係について

3.2.6.1.1 レベルについて

レベルは、階層における1つの位置を表します。たとえば、時間ディメンションには、月、四半期および年レベルのデータを表す階層があります。レベルには、要約レベルから詳細レベルまであり、ルート・レベルは最上位レベルで、最も要約の進んだレベルです。ディメンション内のレベルは、1つ以上の階層に編成されます。

3.2.6.1.2 レベルの関係について

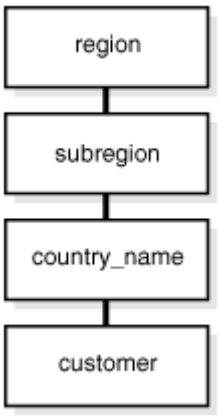
レベル間の関係によって、最も要約された情報(ルート)から最も限定的な情報まで、上位レベルから下位レベルまでの順序付けが指定されます。階層内のレベル間の親子関係が定義されます。

階層は、より複雑なリライトを使用可能にするためにも不可欠なコンポーネントです。たとえば、データベースでは、四半期と年とのディメンションの依存性がわかっていれば、既存の販売収入を四半期別から年別へと集計できます。

3.2.6.2 典型的なディメンション階層

図3-2に、customersをベースとするディメンション階層を示します。

図3-2 一般的なディメンションの階層レベル


region
subregion
country_name
customer

4 データ・ウェアハウスの最適化および技法

この章では、データ・ウェアハウスのスキーマについて説明します。内容は次のとおりです。

- データ・ウェアハウスでの索引の使用方法
- データ・ウェアハウスでの整合性制約の使用方法
- データ・ウェアハウスにおけるパラレル実行について
- データ・ウェアハウスにおける記憶域要件の最適化について
- スター・クエリーおよび3NFスキーマの最適化
- 近似問合せ処理について
- 近似上位N問合せ処理について

4.1 データ・ウェアハウスでの索引の使用方法

索引を使用することにより、データ・ウェアハウスに格納されたデータをより迅速に取得できます。この項ではデータ・ウェアハウスで索引を使用する際の次の側面について説明します。

- データ・ウェアハウスでのビットマップ索引の使用について
- データ・ウェアハウス・アプリケーションの索引に対するメリット
- カーディナリティとビットマップ索引について
- ビットマップ索引の使用対象の判定方法
- データ・ウェアハウスでのビットマップ結合索引の使用
- データ・ウェアハウスでのBツリー索引の使用
- 索引の圧縮の使用
- ローカル索引とグローバル索引の選択基準

4.1.1 データ・ウェアハウスでのビットマップ索引の使用について

ビットマップ索引は、データ・ウェアハウス環境で広く使用されています。この環境では、通常、データおよび非定型の問合せは大量にありますが、同時DMLトランザクションは低いレベルです。このようなアプリケーションでは、ビットマップ索引による次のメリットがあります。

- 大規模な非定型問合せに対する応答時間が削減されます。
- 他の索引付けテクニックと比較すると、領域の使用量が少なくて済みます。
- 比較的CPUの数が少ないハードウェアまたはメモリー量が少ないハードウェアでも、大幅にパフォーマンスが向上します。

大規模な表を従来のBツリー索引で完全に索引付けすると、索引が、表にあるデータの数倍の大きさになる場合があるため、ディスク領域の点で非常にコストが高くなります。通常、ビットマップ索引のサイズは、表内の索引付けされたデータの何分の1かの大きさで済みます。

索引は、指定したキー値を含む表の行へのポインタを保有します。通常の索引には、そのキー値がある行に対応する各キーのROWIDのリストが格納されます。ビットマップ索引では、各キー値のビットマップが、ROWIDのリストのかわりに使用されます。

ビットマップの各ビットは、ROWIDに対応します。ビットが設定されると、対応するROWIDを持つ行に、キー値が含まれることを

意味します。マッピング機能によってビットの位置が実際のROWIDに変換されるため、ビットマップ索引は、通常の索引と同じ機能を提供します。ビットマップ索引は、ビットマップを圧縮して格納します。個別キー値の数が少ない場合は、ビットマップ索引の圧縮率が高くなり、Bツリー索引に対する領域節約の面での優位性がさらに高くなります。

ビットマップ索引は、WHERE句に複数の条件が含まれる問合せに対して最も効率的です。すべての条件ではなく一部の条件のみを満たす行は、表自体がアクセスされる前に除外されます。これによって、応答時間が大幅に削減されます。どの索引を作成するべきか判断できない場合は、SQLアクセス・アドバイザを使用すると、作成するべき索引についてのリコメンデーションが生成されます。ビットマップ索引のビットマップは瞬時に結合されるので、通常は単一列のビットマップ索引を使用するのが最適です。

また、ビットマップ索引は、通常、メンテナンスを行うよりも削除して再作成するほうが簡単であることに注意してください。

4.1.1.1 ビットマップ索引およびNULLについて

ビットマップ索引は、他のほとんどのタイプの索引とは異なり、NULL値を持つ行を含みます。NULLの索引付けは、集計関数COUNTが指定されている問合せなどの、いくつかのタイプのSQL文に有効です。

例4-1 ビットマップ索引

```
SELECT COUNT(*) FROM customers WHERE cust_marital_status IS NULL;
```

この問合せでは、cust_marital_statusのビットマップ索引が使用されます。この問合せでは、Bツリー索引を使用できないことに注意してください。Bツリー索引にはNULL値は格納されないためです。

```
SELECT COUNT(*) FROM customers;
```

NULLデータを持つ行を含めて、表のすべての行に索引が付けられるため、すべてのビットマップ索引をこの問合せに使用できます。NULLに索引が付けられていない場合、オプティマイザは、NOT NULL制約が指定されている列にある索引のみを使用できます。

4.1.1.2 パーティション表のビットマップ索引について

ビットマップ索引をパーティション表に作成できますが、パーティション表に対してローカルである必要があり、グローバル索引にできません。パーティション表に作成できるグローバル索引は、パーティション化または非パーティション化のBツリー索引のみです。

関連項目:

- Oracle Database SQL言語リファレンス
- 『Oracle Database VLDBおよびパーティショニング・ガイド』

4.1.2 データ・ウェアハウス・アプリケーションに対する索引のメリット

ビットマップ索引は、ユーザーがデータの更新ではなく、データの問合せを行うデータ・ウェアハウス・アプリケーションに使用します。この種の索引は、データを変更する同時トランザクションの数が多いOLTPアプリケーションには適していません。

索引はカーディナリティが高い列でメリットがあります。

関連項目:

カーディナリティとビットマップ索引について

パラレル問合せおよびパラレルDMLは、ビットマップ索引でも動作します。ビットマップ索引では、索引のパラレル作成、連結索

引もサポートされます。

4.1.3 カーディナリティとビットマップ索引について

ビットマップ索引は、表の行数に対する個別値の数の比率が小さい列に対して最も効果的です。この比率は、カーディナリティ度と呼ばれます。性別の列は個別値が2つ(男性と女性)しかないため、ビットマップ索引に最適です。ただし、データ・ウェアハウス管理者は、カーディナリティが高い列にビットマップ索引を作成する場合もあります。

たとえば、行が100万ある表では、10,000の個別値を持つ列がビットマップ索引の候補になります。この列のビットマップ索引は、特に、この列が他の索引付けされた列と連結して頻繁に問い合せられる場合に、Bツリー索引よりパフォーマンスが高くなります。実際、典型的なデータ・ウェアハウス環境では、すべての非一意列がビットマップ索引の候補です。

Bツリー索引は、カーディナリティが高いデータ(customer_nameやphone_numberなど、固有な値を多く持つデータ)に対して最も効果的です。データ・ウェアハウスでは、Bツリー索引は、一意の列またはカーディナリティが非常に高い列(ほとんど一意である列)にのみ使用してください。データ・ウェアハウスの索引は、ほとんどがビットマップ索引であっても問題ありません。

非定型問合せなどを行う場合、ビットマップ索引によって問合せのパフォーマンスが大幅に向上します。結果のビットマップをROWIDに変換する前に、対応するブール操作をビットマップに対して直接実行することにより、問合せのWHERE句で指定したANDおよびOR条件は、すぐに解決されます。結果の行数が少ない場合は、全表スキャンを行うことなく、すぐに問合せの結果が戻されます。

次の問合せの出力は、ある会社のcustomers表の一部を示しています。

```
SELECT cust_id, cust_gender, cust_marital_status, cust_income_level
FROM customers;

CUST_ID    C CUST_MARITAL_STATUS  CUST_INCOME_LEVEL
---------- - -------------------- --------------------
...
        70 F                      D: 70,000 - 89,999
        80 F married              H: 150,000 - 169,999
        90 M single               H: 150,000 - 169,999
       100 F                      I: 170,000 - 189,999
       110 F married              C: 50,000 - 69,999
       120 M single               F: 110,000 - 129,999
       130 M                      J: 190,000 - 249,999
       140 M married              G: 130,000 - 149,999
...
```

cust_gender、cust_marital_statusおよびcust_income_levelは、すべてカーディナリティが低い列(MARITAL_STATUSは3つの値のみ、GENDERは2つの値のみ、INCOME_LEVELは12の値のみが存在する列)であるため、これらの列にはビットマップ索引が理想的です。cust_idは一意の列であるため、この列にはビットマップ索引を作成しないでください。かわりに、この列に一意のBツリー索引を作成すると、最も効率的に表示および検索できます。

表4-1に、この例のcust_gender列に対するビットマップ索引を示します。この索引は、2つの別々のビットマップで構成されており、それぞれが性別に対応しています。

表4-1 ビットマップ索引の例

cust_id	gender='M'	gender='F'
cust_id 70	0	1
cust_id 80	0	1

cust_id	gender='M'	gender='F'
cust_id 90	1	0
cust_id 100	0	1
cust_id 110	0	1
cust_id 120	1	0
cust_id 130	1	0
cust_id 140	1	0

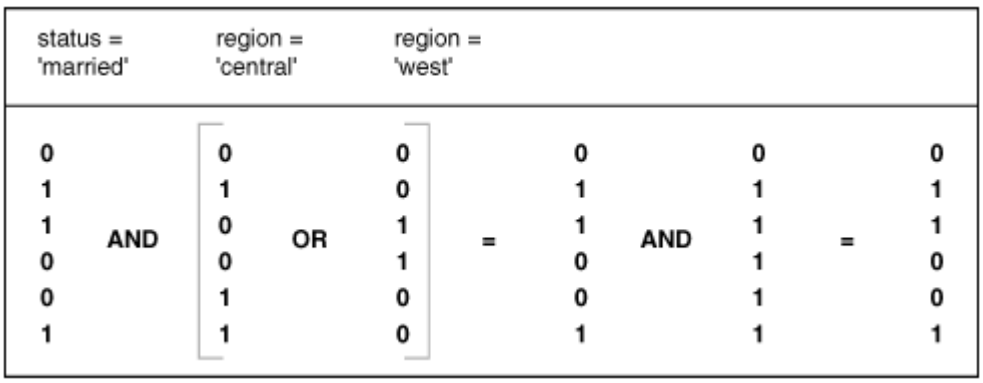
ビットマップの各エントリ(ビット)は、customers表の1つの行に対応します。各ビットの値は、表にある対応する行の値に依存します。たとえば、ビットマップcust_gender='F'には、最初のビットとして1が含まれています。これは、customers表の最初の行にある性別がFであるためです。ビットマップcust_gender='F'の第3ビットは0です。これは、3行目の性別がFでないためです。

会社の顧客の人口統計情報傾向を調査するアナリストが、「既婚者で、所得レベルがGまたはHの顧客がどれだけいるか」と質問するとします。この質問は、次の問合せで表すことができます。

```
SELECT COUNT(*) FROM customers
WHERE cust_marital_status = 'married'
AND cust_income_level IN ('H: 150,000 - 169,999', 'G: 130,000 - 149,999');
```

図4-1に示すように、ビットマップ索引は、単にビットマップにある1の数をカウントすることで、この問合せを効率的に処理できます。結果セットはビットマップのORマージ操作を使用して検索され、ROWIDへの変換は不要です。さらに条件を満たす特定の顧客属性も確認するには、ビットマップからROWIDへの変換後に、結果ビットマップを使用して表にアクセスします。

図4-1 ビットマップ索引を使用した問合せの実行



4.1.4 ビットマップ索引の使用対象の判定方法

ファクト表のみが問い合され、索引付けされた列に対する条件がある場合、またはファクト表が2つ以上のディメンション表と結合され、そのファクト表の外部キー列に対する索引があり、ディメンション表の列に対する条件がある場合に、ビットマップ索引は有効です。

次の条件が当てはまる場合、ファクト表の列はビットマップ索引の候補となります。

- 索引付けされた列の各個別値に対し行が100以上ある。この制限が当てはまる場合、ビットマップ索引は通常の索引よりも大幅に小さくなり、通常の索引よりも早く作成できます。たとえば、数十億の行がある表に、100万の個別値

が含まれる場合などです。

また、次のいずれかの場合も該当します。

- 索引付けされた列が、問合せで制限される(WHERE句で参照される)。

または

- 索引付けされた列が、ディメンション表の外部キーである。この場合、この索引によってスター型変換が行われる可能性が高くなります。

4.1.5 データ・ウェアハウスでのビットマップ結合索引の使用

単一表のビットマップ索引の他に、ビットマップ結合索引を作成できます。これは、複数の表の結合に対するビットマップ索引です。ビットマップ結合索引では、索引付けされる表のビットマップは、結合先の表の値のために作成されます。データ・ウェアハウス環境では、結合条件はディメンション表の主キー列とファクト表の外部キー列の間の内部等価結合です。

ビットマップ結合索引を使用すると、パフォーマンスを大幅に向上させることができます。結合結果を格納することで、ビットマップ結合索引を使用するSQL文では、結合をまったく行わずに済みます。また、ビットマップ結合索引では、結合列に対する通常のビットマップ索引と比較して、個別値の数がさらに少なくなることがほとんどなので、ビットマップの圧縮率が高くなります。その結果、結合列に対する通常のビットマップ索引よりも、使用する領域が小さくなります。

ビットマップ結合索引は、マテリアライズド結合ビューより格納の効率がはるかによく、事前に結合をマテリアライズする方法の代替手段です。これは、マテリアライズド結合ビューでは、ファクト表のROWIDが圧縮されないためです。

Bツリー索引とビットマップ索引では、最大列数の制限が異なります。

関連項目:

- データ・ウェアハウスにおけるビットマップ結合索引の4つの結合モデル
- ビットマップ結合索引の制限事項と要件
- これらの制限の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

4.1.5.1 データ・ウェアハウスのビットマップ結合索引の4つの結合モデル

ビットマップ結合索引は、スター・モデル環境で使用するのが最も一般的です。スター・モデル環境とは、大規模な表の索引列に、1つ以上の小規模な表が結合されている環境のことです。大規模な表のことをファクト表、小規模な表のことをディメンション表と呼びます。次の項では、ビットマップ結合索引でサポートされている4つの結合モデルについて説明します。

次の例は、1つのディメンション表の列と1つのファクト表を結合するビットマップ結合索引を示しています。ビットマップ索引をcustomers表のcust_gender列に作成したカーディナリティとビットマップ索引についての例とは異なり、ここでは、結合列customers(cust_gender)のビットマップ結合索引をファクト表salesに作成します。表salesは、次のようにcust_id値のみを格納します。

```
SELECT time_id, cust_id, amount_sold FROM sales;

TIME_ID   CUST_ID    AMOUNT_SOLD
--------- ---------- -----------
01-JAN-98      29700        2291
01-JAN-98       3380         114
01-JAN-98      67830         553
01-JAN-98     179330           0
```

```
01-JAN-98      127520          195
01-JAN-98       33030          280
...
```

このようなビットマップ結合索引を作成するには、列customers(cust_gender)を表salesに結合する必要があります。結合条件は、ビットマップ結合索引のCREATE文の一部として指定します。

```
CREATE BITMAP INDEX sales_cust_gender_bjix
ON sales(customers.cust_gender)
FROM sales, customers
WHERE sales.cust_id = customers.cust_id
LOCAL NOLOGGING COMPUTE STATISTICS;
```

次の問合せにより、ビットマップ結合索引に格納されるビットマップの作成に使用する結合結果が示されます。

```
SELECT sales.time_id, customers.cust_gender, sales.amount_sold
FROM sales, customers
WHERE sales.cust_id = customers.cust_id;

TIME_ID   C AMOUNT_SOLD
--------- - -----------
01-JAN-98 M        2291
01-JAN-98 F         114
01-JAN-98 M         553
01-JAN-98 M           0
01-JAN-98 M         195
01-JAN-98 M         280
01-JAN-98 M          32
...
```

表4-2に、この例におけるビットマップ結合索引のビットマップ表現を示します。

表4-2 ビットマップ結合索引の例

salesのレコード	**cust_gender='M'**	**cust_gender='F'**
sales のレコード 1	1	0
sales のレコード 2	0	1
sales のレコード 3	1	0
sales のレコード 4	1	0
sales のレコード 5	1	0
sales のレコード 6	1	0
sales のレコード 7	1	0

次の例のように、複数の列または複数の表を使用すると、他のビットマップ結合索引を作成できます。

例4-2 ビットマップ結合索引: 複数のディメンション列と1つのファクト表の結合

ビットマップ結合索引は、次の例のように、1つのディメンション表の複数の列に作成できます。次の例では、shスキーマのcustomers(cust_gender, cust_marital_status)を使用しています。

```
CREATE BITMAP INDEX sales_cust_gender_ms_bjix
ON sales(customers.cust_gender, customers.cust_marital_status)
FROM sales, customers
WHERE sales.cust_id = customers.cust_id
LOCAL NOLOGGING COMPUTE STATISTICS;
```

例4-3 ビットマップ結合索引: 複数のディメンション表と1つのファクト表の結合

ビットマップ結合索引は、次の例のように、複数のディメンション表に作成できます。次の例では、customers(gender)とproducts(category)を使用しています。

```
CREATE BITMAP INDEX sales_c_gender_p_cat_bjix
ON sales(customers.cust_gender, products.prod_category)
FROM sales, customers, products
WHERE sales.cust_id = customers.cust_id
AND sales.prod_id = products.prod_id
LOCAL NOLOGGING COMPUTE STATISTICS;
```

例4-4 ビットマップ結合索引: スノーフレーク・スキーマ

複数の表のビットマップ結合索引を作成できます。この場合、索引付けされた列は、別の表を使用して索引付けされた表に結合されます。たとえば、countries表がsales表と直接結合されていなくても、countries.country_nameの索引を作成できます。かわりに、countries表がcustomers表に結合され、customers表はsales表に結合されます。このタイプのスキーマは、一般にスノーフレーク・スキーマと呼ばれます。

```
CREATE BITMAP INDEX sales_co_country_name_bjix
ON sales(countries.country_name)
FROM sales, customers, countries
WHERE sales.cust_id = customers.cust_id
  AND customers.country_id = countries.country_id
LOCAL NOLOGGING COMPUTE STATISTICS;
```

4.1.5.2 ビットマップ結合索引の制限事項と要件

結合結果を格納する必要があるため、ビットマップ結合索引には次の制限事項があります。

- パラレルDMLはファクト表でのみサポートされます。ディメンション表に対してパラレルDMLを行うと、索引はUNUSABLEのマークが付けられます。
- ビットマップ結合索引を使用する場合、異なるトランザクションで同時に更新できる表は1つのみです。
- 結合に2回同じ表は使用できません。
- 一時表にはビットマップ結合索引は作成できません。
- 索引の列は、すべてディメンション表の列である必要があります。
- ディメンション表の結合列は、主キー列であるか、一意制約を持つ必要があります。
- ファクト表との結合に関与しているディメンション表の1つ以上の列は、主キー列であるか、一意制約を持つ必要があります。
- ディメンション表に複合主キーがある場合、主キーの各列が結合の一部である必要があります。
- 通常のビットマップ索引の作成に関する制限事項も、ビットマップ結合索引に適用されます。たとえば、UNIQUE属性を持つビットマップ索引は作成できません。その他の制限事項については、『Oracle Database SQL言語リファレンス』

を参照してください。

4.1.6 データ・ウェアハウスでのBツリー索引の使用

Bツリー索引は、木をさかさまにしたような構造になっています。索引の最下位層レベルには、実際のデータ値および対応する行へのポインタがあります。これは、本の索引に、各索引エントリに対応するページ数があることとよく似ています。

一般に、典型的な問合せが索引付けされた列を参照して少数の行を取り出すことがわかっている場合に、Bツリー索引を使用します。これらの問合せでは、索引を参照する方が、行を迅速に検索できます。しかし、本の索引にたとえると、本のトピックを1つずつ参照する場合、そのトピックを索引で調べてから該当ページを検索することはありません。本のすべての章を読む方が速いことになります。これと同様に、表内のほとんどの行を取り出す場合、索引を参照して表の行を検索するのでは意味がありません。かわりに、表を読み込むかまたはスキャンします。

データ・ウェアハウスでは、一意キーの索引付けに最もよく使用されるのはBツリー索引です。多くの場合、データ・ウェアハウスにあるこれらの列に索引付けを行う必要はありません。これは、事前に実行されるETL処理の中で一意性はすでに確保されており、通常のデータ・ウェアハウスの問合せは、こうした索引を使用してもパフォーマンスが向上しないためです。Bツリー索引は、第3正規形のスキーマを使用する環境で使用するのが一般的です。ほとんどのデータ・ウェアハウス環境では、ビットマップ索引が、Bツリー索引よりもよく使用されます。

Bツリー索引とビットマップ索引では、最大列数の制限が異なります。これらの制限の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

4.1.7 索引の圧縮の使用

ビットマップ索引は常に、ユーザーの操作を必要とすることなく、独自の方法で圧縮されて格納されます。Bツリー索引は、明示的に圧縮して格納することで、領域を大幅に節約することが可能です。この際、各索引ブロックにはより多くのキーが格納され、I/Oの削減およびパフォーマンスの向上にもつながります。

キーの圧縮によって、Bツリー索引を圧縮できます。この結果、重複値による領域のオーバーヘッドが低減されます。非一意索引では、すべての索引列を圧縮形式で格納できますが、一意索引では、少なくとも1つの索引列は圧縮せずに格納する必要があります。キーの圧縮機能に加え、OLTPの索引圧縮機能はより高い圧縮度を提供しますが、データ・ウェアハウス環境よりもOLTPアプリケーションに適しています。

通常、索引のキーは、グループ化要素と一意要素という2つの要素を持っています。一意要素を持つようにキーが定義されていない場合は、ROWIDをグループ化要素に追加する形で一意要素が提供されます。キーの圧縮では、グループ化要素が分離され、複数の一意要素が共有できるような形で格納されます。圧縮対象として選択された列のカーディナリティによって、実現可能な圧縮率が決まります。したがって、たとえば5つの列からなる一意索引の一意性のほとんどが、後ろの2列によって実現されている場合は、先頭の3列を圧縮して格納するのが最も効率的ということになります。4つの列を圧縮する場合は、重複性はほとんど失われ、圧縮率が低下します。

キーの圧縮によって、索引の領域要件は低くなりますが、索引スキャン時のキー列値の再構築に必要なCPUタイムが長くなる場合があります。また、接頭辞エントリごとに4バイトのオーバーヘッドが生じるので、追加の領域オーバーヘッドも発生します。

関連項目:

- キーの圧縮の詳細は、『Oracle Database管理者ガイド』を参照してください。
- OLTP索引の圧縮の詳細は、『Oracle Database管理者ガイド』を参照してください。

4.1.8 ローカル索引とグローバル索引の選択基準

パーティション表のBツリー索引は、ローカルまたはグローバルにできます。Oracle8i以前のリリースでは、データ・ウェアハウス環境ではグローバル索引を使用しないようにお薦めしていました。これは、パーティションのDDL文(ALTER TABLE ... DROP PARTITIONなど)により索引全体が無効になり、索引の再構築に手間がかかったためです。グローバル索引は、DDLの後に使用不可とマークされなくてもメンテナンス可能です。これにより、データ・ウェアハウス環境におけるグローバル索引の効率が向上しています。

ただし、ローカル索引の方がグローバル索引より一般的です。グローバル索引を使用する必要があるのは、ローカル索引では満たせない特定の要件(非パーティション化キーの一意索引や、パフォーマンス要件など)がある場合です。

パーティション表のビットマップ索引は、常にローカルです。

4.2 データ・ウェアハウスでの整合性制約の使用方法

整合性制約は、データが、データベース管理者によって指定されたガイドラインに従うことを保証するためのメカニズムです。

最も一般的なタイプの制約は、次のとおりです。

- UNIQUE制約

  特定の列が一意であることを保証します。

- NOT NULL制約

  NULL値が許されないことを保証します。

- FOREIGN KEY制約

  2つのキーが主キーと外部キーの関係を共有することを保証します。

制約は、データ・ウェアハウスにおいて、次の目的に使用できます。

- データの正当性

  制約により、不適切なデータの挿入を防止するために、データ・ウェアハウスのデータがデータ整合性および正確さのガイドラインに従っているかどうかが検証されます。

- 問合せ最適化

  Oracle Databaseでは、SQL問合せを最適化するときに、制約が使用されます。制約は、問合せの最適化に多くの点で効果的ですが、マテリアライズド・ビューのクエリー・リライトに特に重要です。

多くのリレーショナル・データベース環境とは異なり、データ・ウェアハウスのデータは通常、抽出、変換、ロード(ETL)プロセス中の、制御された状況下で追加または変更されます。通常、OLTPシステムとは異なり、複数のユーザーがデータ・ウェアハウスを直接更新することはありません。

関連項目:

- データ移動/ETLの概要

この項では、次の項目について説明します。

- 制約の状態の概要
- 一般的なデータ・ウェアハウスの整合性制約

4.2.1 制約の状態の概要

データ・ウェアハウスにおいて最適な制約の使用方法を理解するには、まず、制約の基本的な目的を理解する必要があります。

このような目的をいくつか次に示します。

- 施行

  制約を施行するには、制約がENABLE状態である必要があります。ENABLE状態の制約により、任意の1つ(または複数)の表におけるすべてのデータの変更が、制約の条件を満たすことが保証されます。データ変更操作によってデータが制約に違反する場合、その操作は制約違反エラーとなり正常に実行されません。

- 妥当性チェック

  妥当性チェックを行うために制約を使用するには、制約がVALIDATE状態である必要があります。制約がVALIDATED状態の場合、表に現在存在しているすべてのデータが制約を満たします。

  妥当性チェックは、施行とは関係ありません。業務系システムでの一般的な制約はENABLEDおよびVALIDATED状態ですが、VALIDATED状態であってもENABLED状態でないか、またはその逆(ENABLED状態であってもVALIDATED状態でない)の場合もあります。後者の2つの状態は、データ・ウェアハウスで効果的です。

- 信頼

  特定の制約の条件がTRUEであることがわかっているため、妥当性チェックや制約の施行を必要としない場合があります。しかしそのような場合でも、問合せの最適化やパフォーマンスの改善のために、制約を存在させることができます。この目的で使用する制約は、信頼またはRELY制約と呼ばれ、RELY状態である必要があります。RELY状態は、指定された制約がTRUEであることを信頼してよいことをOracleに通知するメカニズムを提供します。

  RELY状態の影響を受けるのは、VALIDATED状態でない制約のみであることに注意してください。

4.2.2 一般的なデータ・ウェアハウスの整合性制約

この項では、読者が制約の一般的な使用方法を理解していることを前提としています。つまり、ENABLEかつVALIDATEDな状態の制約です。データ・ウェアハウスでは、このような制約の作成およびメンテナンスに非常にコストがかかるため、多くのユーザーにとって、このような制約が効率的でないことは明らかです。この項の内容は次のとおりです。

- データ・ウェアハウスでの一意制約
- データ・ウェアハウスでの外部キー制約
- データ・ウェアハウスでのRELY制約
- データ・ウェアハウスでのNOT NULL制約
- データ・ウェアハウスでの整合性制約とパラレル化
- データ・ウェアハウスでの整合性制約とパーティション化
- データ・ウェアハウスでのビューの制約

4.2.2.1 データ・ウェアハウスでの一意制約

一意制約は、通常、一意索引を使用して規定されます。ただし、表が非常に大規模になる可能性があるデータ・ウェアハウスでは、一意索引を作成すると、処理時間およびディスク領域の点で非常にコストがかかる場合があります。

データ・ウェアハウスにsales表があり、その表にsales_id列が含まれているとします。sales_idは、単一の売上トランザクションを一意に識別し、データ・ウェアハウス管理者は、この列がデータ・ウェアハウス内で一意であることを保証する必要があります。

制約を作成する方法の1つを、次に示します。

```
ALTER TABLE sales ADD CONSTRAINT sales_uk
UNIQUE (prod_id, cust_id, promo_id, channel_id, time_id);
```

デフォルトでは、この制約はENABLEかつVALIDATEDな状態です。Oracleは、この制約をサポートするために、sales_idに一意索引を暗黙的に作成します。ただし、次の3つの理由から、この索引がデータ・ウェアハウスでは不適切な場合があります。

- sales表には数百万または数十億もの行が含まれることが多いため、一意索引は非常に大きくなる可能性があります。
- 一意索引は、問合せの実行にはあまり使用されません。ほとんどのデータ・ウェアハウスの問合せは一意キーについての条件検索を行わないため、この索引を作成してもパフォーマンスが改善される可能性はあまりありません。
- salesがsales_id以外の列でパーティション化されている場合は、一意索引はグローバル索引である必要があります。これによって、sales表でのすべてのメンテナンス操作が悪影響を受ける場合があります。

一意索引は、sales表で変更された個々の行が一意制約を満たすことを保証するために必要です。

データ・ウェアハウス表の場合に使用できる、一意制約にかわる方法を次の文に示します。

```
ALTER TABLE sales ADD CONSTRAINT sales_uk
UNIQUE (prod_id, cust_id, promo_id, channel_id, time_id) DISABLE VALIDATE;
```

この文によって一意キー制約が作成されますが、制約がDISABLED状態であるため、一意索引は必要ありません。この方法によって、制約が一意索引のデメリットの影響を受けずに一意性を保証できるため、多くのデータ・ウェアハウス環境でメリットがあります。

ただし、データ・ウェアハウス管理者が、DISABLE VALIDATE制約を考慮する場合にトレードオフがあります。この制約はDISABLED状態であるため、一意の列を変更するDML文はsales表に対して実行できません。制約が存在する状態でこの表を変更するには、次の2つの方法があります。

- DDLを使用して、この表にデータを追加します(パーティションの交換など)。マテリアライズド・ビューのリフレッシュの例を参照してください。
- この表を変更する前に、制約を削除します。その後、すべての必要なデータ修正を行います。最後に、DISABLED状態の制約を再作成します。この制約を再作成する方が、ENABLED状態の制約を再作成するより効率的です。ただし、この方法では、制約の削除中にsales表に追加されたデータが一意であることは保証されません。

4.2.2.2 データ・ウェアハウスでの外部キー制約

スター・スキーマ・データ・ウェアハウスでは、外部キー制約により、ファクト表とディメンション表のリレーションシップの妥当性がチェックされます。制約の例を次に示します。

```
ALTER TABLE sales ADD CONSTRAINT sales_time_fk
FOREIGN KEY (time_id) REFERENCES times (time_id)
ENABLE VALIDATE;
```

ただし、場合によっては、外部キー制約に異なる状態(特に、ENABLE NOVALIDATE状態)を使用するように選択することがあります。データ・ウェアハウス管理者は、次のいずれかの場合に、ENABLE NOVALIDATE制約を使用することがあります。

- 制約を満たさないデータが表にあるが、データ・ウェアハウス管理者が規定する制約を作成する場合
- 施行済の制約がすぐに必要な場合

データ・ウェアハウスで、新しいデータはファクト表に毎日ロードされ、ディメンション表は週末にのみリフレッシュされるとします。その週の間は、ディメンション表とファクト表が実際には外部キー制約を満たさない可能性があります。それでも、データ・ウェアハウス管理者は、ETLプロセス外で外部キー制約に影響する可能性がある変更が行われないようにするため、この制約を施行する場合があります。つまり、次のように、ETLプロセスの実行後に外部キー制約を毎晩作成できます。

```
ALTER TABLE sales ADD CONSTRAINT sales_time_fk
```

```
FOREIGN KEY (time_id) REFERENCES times (time_id)
ENABLE NOVALIDATE;
```

ENABLE NOVALIDATEを使用すると、制約がTRUEであると考えられる場合にも、施行される制約をすばやく作成できます。ETLプロセスによって、外部キー制約がTRUEであるかどうかが検証されるとします。データベースにこの外部キー制約を再検証させるには時間およびデータベース・リソースが必要となるため、かわりに、データ・ウェアハウス管理者は、ENABLE NOVALIDATEを使用して外部キー制約を作成できます。

4.2.2.3 データ・ウェアハウスでのRELY制約

ETLプロセスでは、通常、ある制約がTRUEかどうかが検証されます。たとえば、ファクト表の受信データにあるすべての外部キーの妥当性チェックを実行します。これは、データ・ウェアハウスに制約を実装するかわりに、制約に従ったデータが提供されることが信頼できることを意味します。RELY制約を次のように作成します。

```
ALTER TABLE sales ADD CONSTRAINT sales_time_fk
FOREIGN KEY (time_id) REFERENCES times (time_id)
RELY DISABLE NOVALIDATE;
```

この文は、主キーがRELY状態であることを前提としています。RELY制約は、データの妥当性チェックに使用されない場合にも、次のことができます。

- マテリアライズド・ビューに対して、より高度なクエリー・リライトを使用可能にします。マテリアライズド・ビューのための基本的なクエリー・リライトを参照してください。
- その他のデータ・ウェアハウス・ツールによって、制約に関する情報をOracleデータ・ディクショナリから直接取り出すことができます。

RELY制約の作成には、コストがほとんどかかりません。また、DML中やロード中にもオーバーヘッドは発生しません。制約がVALIDATED状態ではないため、その作成に必要なデータ処理はありません。

4.2.2.4 データ・ウェアハウスでのNOT NULL制約

クエリー・リライトを使用する際は、NOT NULL制約が必要かどうかを考慮する必要があります。それらが必要となる典型例は、後戻り結合クエリー・リライトを使用する場合です。

関連項目:

- クエリー・リライト使用時のNOT NULL制約の詳細は、マテリアライズド・ビューのための高度なクエリー・リライトを参照してください。

4.2.2.5 データ・ウェアハウスでの整合性制約とパラレル化

すべての制約は、パラレルで妥当性チェックできます。非常に大規模な表で制約の妥当性チェックを行う場合、パフォーマンスの目標を達成するために、パラレル化が必要になります。ある任意の制約操作の並列度は、基礎となる表のデフォルトの並列度によって決定されます。

4.2.2.6 データ・ウェアハウスでの整合性制約とパーティション化

データをパーティション化する前に、制約を作成してメンテナンスできます。データ・ウェアハウスにおけるパーティション化の重要性については、以降の章を参照してください。パーティション化により、他の多くの操作の管理と同様に、制約管理も改善できます。たとえば、マテリアライズド・ビューのリフレッシュには、別々のステージング表にUNIQUEおよびFOREIGN KEYの各制約を作成する例があります。これらの制約はEXCHANGE PARTITION文の実行中にメンテナンスされます。

外部表の場合は、DISABLEモードでのみRELY制約を定義できます。これは、主キー、一意キーおよび外部キーの制約に適用されます。

4.2.2.7 データ・ウェアハウスでのビューの制約

ビューの制約を作成できます。ビューでサポートされるタイプの制約は、RELY制約のみです。

このタイプの制約は、問合せが実表ではなくビューにアクセスする際に、データベース管理者が、ビューとの間のリレーションシップを定義する必要がある場合に有効です。

関連項目:

- 基本的なマテリアライズド・ビュー
- マテリアライズド・ビューのための基本的なクエリー・リライト

4.3 データ・ウェアハウスにおけるパラレル実行について

今日のデータベースには、それがウェアハウスや運用データ・ストア、またはOLTPシステムのどれであるかに限らず、大量の情報が格納されています。しかし、含まれるデータの量が膨大であるため、正しい情報をタイムリーに検索し、提示することは容易ではありません。

パラレル実行は、この課題に対処するための機能です。パラレル実行(パラレル化とも呼ばれます)では、複数のプロセスを使用して単一のタスクを完了させることで、テラバイトのデータを数時間や数日ではなく、数分で処理できます。これにより、通常意思決定支援システム(DSS)およびデータ・ウェアハウスに関連付けられているサイズの大きなデータベース上で、データ集中型の操作のレスポンス時間を大幅に削減できます。また、OLTPシステム上で、索引の作成などのバッチ処理またはスキーマ・メンテナンス操作のためにパラレル実行を実装することもできます。パラレル化の概念は、タスクへの分解であり、これにより1つのプロセスで問合せに関するすべての処理を実行するのではなく、多くのプロセスが同時に各処理を実行します。たとえば、1年の合計売上げを4つのプロセスで計算するのに、1つのプロセスですべての四半期を処理するのではなく、各プロセスが1年の四半期それぞれを処理する場合です。これを使用するとパフォーマンスの大幅な向上が見込めます。

パラレル実行では、次のプロセスのパフォーマンスを向上できます。

- 大規模な表のスキャン、結合またはパーティション索引スキャンを必要とする問合せ
- 大規模な索引の作成
- 大規模な表の作成(マテリアライズド・ビューを含む)
- バルク挿入、更新、マージ、削除

また、パラレル実行を使用して、Oracleデータベース内のオブジェクト型にアクセスできます。たとえば、パラレル実行を使用してラージ・オブジェクト(LOB)にアクセスできます。

大規模なデータ・ウェアハウスでは、優れたパフォーマンスを得るために、常にパラレル実行を使用する必要があります。OLTPアプリケーションの特定の処理(バッチ処理など)も、パラレル実行を行うことでパフォーマンスが大幅に向上します。

この項では、次の項目について説明します。

- パラレル実行を使用する理由
- 自動並列度および文のキューイング
- データ・ウェアハウスにおけるインメモリー・パラレル実行について

4.3.1 パラレル実行を使用する理由

通りにある車の台数を数えるというタスクがあるとします。このタスクを実行するには2つの方法があります。1つは、自分が通りまで行き、車の台数を数えます。もう1つは、友人に協力してもらい、それぞれ通りの反対側から車の台数を数え始め、2人が落ち合ったところでそれぞれの台数を足します。

友人が車を数える速さがあなたと同じだとした場合、通りにあるすべての車を数えるタスクは、自分一人で行う場合と比べて半分の時間で終えることができます。このような場合、処理は直線的に測定できます。つまりリソースの数を2倍にすると、全体の処理時間は半分になります。

データベースの場合も、車を数える例とさほど変わりません。リソースの数を2倍にすることで、処理時間が最初の半分になれば、その処理は直線的に測定できます。直線的に測定できることが、車を数える場合でも、データベース問合せからの応答を配信する場合でも、パラレル処理の最終的な目標です。

関連項目:

- パラレル実行の使用に関する詳細は、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照してください。

次のトピックでは、パラレル実行が役に立つシナリオに関するガイダンスを提供します。

- パラレル実行を実装する場合
- パラレル実行を実装しない場合

4.3.1.1 パラレル実行を実装する場合

パラレル実行は、次のすべての特性を持つシステム上で有効です。

- 対称型マルチプロセッサ(SMP)、クラスタ、または大規模なパラレル・システム
- 十分なI/Oバンド幅
- 稼働中でないCPUまたは断続的に使用されているCPU(CPUの使用率が通常30%未満のシステムなど)
- ソート、ハッシュおよびI/Oバッファなどの追加のメモリー集中処理をサポートする十分なメモリー

システムでこれらの特徴が1つでも欠けていると、パラレル実行を使用してもパフォーマンスが大幅には改善されないことがあります。実際に、使用率の高すぎるシステムまたはI/O帯域幅が小さいシステムでは、パラレル実行によりシステム・パフォーマンスが低下する場合もあります。

パラレル実行のメリットは、DSSおよびデータ・ウェアハウス環境でわかります。OLTPシステムでも、バッチ処理やスキーマ・メンテナンス操作(索引の作成など)の際にはパラレル実行の利点が得られます。平均的で単純なDMLやSELECT文は、少数のレコードや単一のレコードに対してアクセスまたは操作し、OLTPアプリケーションを特徴付けますが、パラレルで実行するメリットはありません。

4.3.1.2 パラレル実行を実装しない場合

パラレル実行は、次の場合では通常有効ではありません。

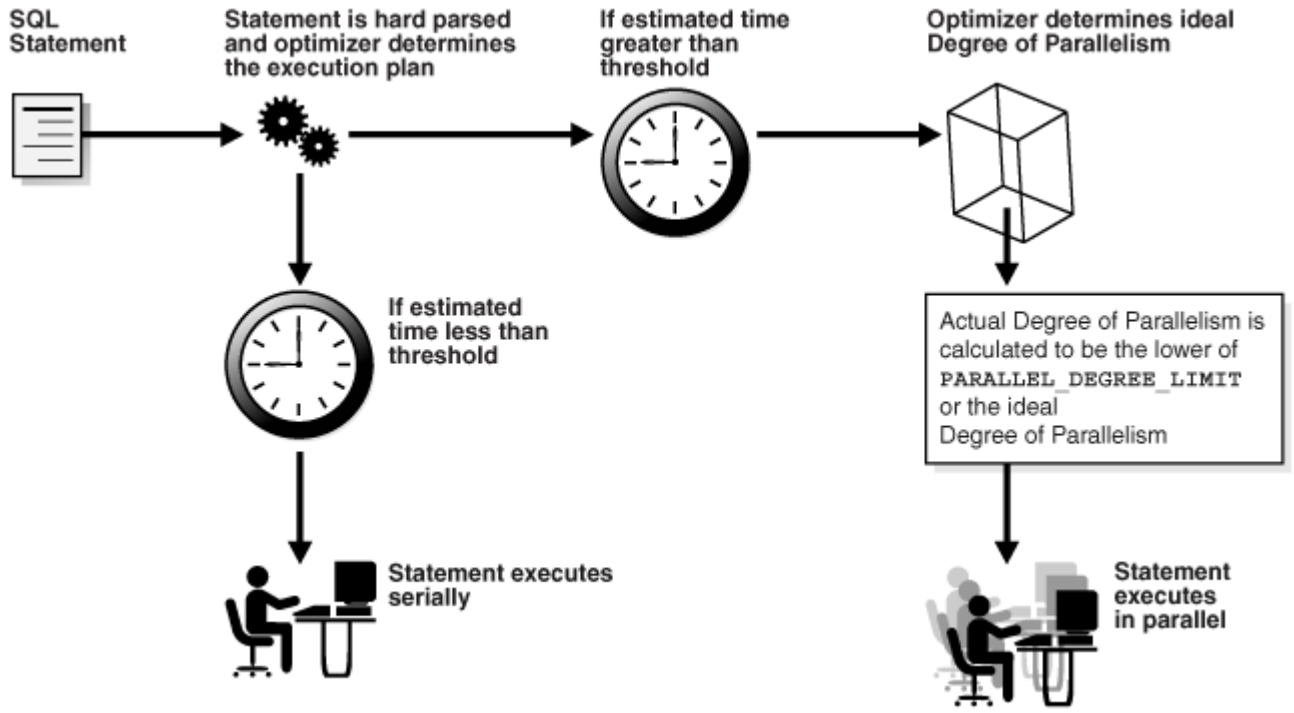
- 標準的な問合せまたはトランザクションが非常に短い(数秒またはそれ以下)環境。これには、ほとんどのオンライン・トランザクション・システムが含まれます。パラレル実行はこのような環境では役立ちません。パラレル実行サーバーの調整に関連するコストが発生するためです。短時間のトランザクションの場合、この調整のコストが並列処理のメリットを上回ります。

- CPU、パラレル実行でもメモリーまたはI/Oリソースが大量に使用されている環境。パラレル実行は追加の使用可能なハードウェア・リソースを利用するように設計されています。そのようなリソースが使用できない場合、パラレル実行はなんのメリットももたらさず、パフォーマンスに悪影響を及ぼす可能性があります。

4.3.2 自動並列度および文のキューイング

名前が示すとおり、自動並列度では、オプティマイザにより決定された、CPU、I/OおよびメモリーのリソースなどのSQL実行コストに基づき、Oracle Databaseが文(DML、DDLおよび問合せ)を実行する並列度(DOP)を決定します。つまり、データベースが問合せを解析し、コストを計算してから、実行に使用するDOPを決定します。コストの最も低い計画がシリアルに実行される可能性もオプションの1つとしてあります。図4-2に、この意思決定プロセスを示します。

図4-2 オプティマイザの計算: シリアルかパラレルか



自動DOPの使用を選択する場合、特にしきい値が比較的低い場合は、多くの文がパラレルに実行している可能性があります。この場合の低いというのはシステムに対する相対的なものであり、絶対的な数量ではありません。

自動DOPを使用して多くの文がパラレルに実行しているという動作が予想されるため、使用可能なパラレル・プロセスの使用量を管理することはより重要になります。つまり、システムは、文をいつ実行するかについてインテリジェントであり、要求されたパラレル・プロセスの数が使用可能かどうかを検証する必要があります。この要求されたプロセスの数というのが、その文のDOPです。

このワークロード管理上の課題への答は、データベース・リソース・マネージャを使用した文のパラレル・キューイングです。文のパラレル・キューイングでは、文が要求したDOPが使用可能な場合に文が実行されます。たとえば、文が64のDOPを要求したときに、この顧客への支援に現在使用できるのが32プロセスのみである場合は、この文はキューに入れられます。

データベース・リソース・マネージャを使用すると、文を消費者グループを介してワークロードに分類できます。各消費者グループには適切な優先度およびパラレル・プロセスの適切なレベルが与えられます。また、各消費者グループはシステム・ロードに基づきパラレル文を格納する固有のキューを持ちます。

関連項目:

- パラレル実行での自動DOPの使用の詳細は、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照してください。
- データベース・リソース・マネージャの使用の詳細は、『Oracle Database管理者ガイド』を参照してください。

4.3.3 データ・ウェアハウスにおけるインメモリー・パラレル実行について

従来、ほとんどの操作に対するパラレル処理では、データベース・バッファ・キャッシュがバイパスされ、ディスクから(ダイレクト・パスI/O経由で)パラレル実行サーバーのプライベート作業領域に直接読み込まれていました。DB_CACHE_SIZEの約2%よりも小さいオブジェクトのみがインスタンスのデータベース・バッファ・キャッシュに入れられますが、パラレルにアクセスされるオブジェクトはこの制限よりも大きいものがほとんどです。つまりこの動作は、そのプライベート処理以外では、使用可能なメモリが利用されることがほとんどないことを示します。しかし、この10年で、ハードウェア・システムはきわめて劇的に進化し、一般的なデータベース・サーバーのメモリ容量は2桁または3桁のギガバイトの領域に達しています。このことが、Oracleの圧縮技術と、Oracle Real Application Clustersの集計されたデータベース・バッファ・キャッシュを利用するOracle Databaseの機能とともに、テラバイトの領域でのオブジェクトのキャッシュを可能にしました。

インメモリー・パラレル実行は、この大きな集計されたデータベース・バッファ・キャッシュを利用します。バッファ・キャッシュを使用してオブジェクトにアクセスするパラレル実行サーバーを持つことにより、大量データのインメモリー・パラレル処理が最大限可能になり、大幅なパフォーマンスの向上が実現されます。

インメモリー・パラレル実行を使用して、SQL文がパラレルに発行されるとき、チェックが実行されて、文がアクセスするオブジェクトをシステムの集計されたバッファ・キャッシュに入れる必要があるかどうかが決定されます。この状況で、オブジェクトは表、索引またはパーティション・オブジェクトの場合には1つまたは複数のパーティションのいずれかです。

関連項目:

- インメモリー・パラレル実行の使用方法の詳細は、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照してください。

4.4 データ・ウェアハウスの記憶域要件の最適化について

データベース・ブロック内の重複値を排除してアーカイブされたデータを圧縮することにより、記憶域要件を削減できます。データ・ウェアハウスの記憶域を改善するためのデータ圧縮の使用では、圧縮データの使用方法について説明します。

圧縮可能なデータベース・オブジェクトには、表およびマテリアライズド・ビューが含まれます。パーティション表については、一部またはすべてのパーティションを圧縮できます。圧縮属性は、表領域、表または表のパーティションに対して宣言できます。表領域レベルで宣言された場合、その表領域で作成されたすべての表はデフォルトで圧縮されます。表(またはパーティションまたは表領域)の圧縮属性は変更可能で、その変更はその表に移動する新規データのみに適用されます。この結果、単一の表またはパーティションにはいくつかの圧縮されたブロックといくつかの通常のブロックが含まれます。これにより、圧縮後もデータのサイズが増加しないことが保証されます。圧縮によってブロックのサイズが増加する場合は、圧縮はそのブロックには適用されません。

4.4.1 データ圧縮によるデータ・ウェアハウスの記憶域の改善

いくつかのパーティションを圧縮するか、またはパーティション化されたヒープ構成表を作成できます。これを実行するには、完全なパーティション化された表を圧縮対象の表として定義するか、またはパーティション・レベルごとに定義するか、いずれかの定義を行います。特定の宣言のないパーティションは表定義から属性を継承し、表レベルの指定がない場合は表領域定義から継承します。

パーティションを圧縮するかまたは未圧縮のままにするかについての決定は、パーティション化されていない表と同じルールに従います。データを論理的に個別パーティションに区切る場合はレンジ・パーティション化とコンポジット・パーティション化を使用できるので、パーティション表は、主に読取り専用のデータ(パーティション)の圧縮部分として適切な候補です。たとえばこれは、古い

データが使用不可になる前の中間の段階としてのすべてのローリング・ウィンドウ操作に役立ちます。データを圧縮することにより、より多くの古いデータをオンラインで保持でき、追加的な記憶域使用の負荷を最小化できます。

また、既存の未圧縮の表パーティションを後で変更したり、新規の圧縮パーティションおよび未圧縮パーティションを追加したり、MERGE PARTITION、SPLIT PARTITIONまたはMOVE PARTITIONなどのデータ移動が必要なパーティション・メンテナンス操作の一部として圧縮属性を変更できます。パーティションにはデータを含めることができ、または空にすることもできます。

部分的または完全に圧縮されたパーティション表のアクセスおよびメンテナンスは、まったく圧縮されていないパーティション表の場合と同じです。また、完全に未圧縮のパーティション表に適用されたすべてのルールも、一部または完全に圧縮されたパーティション表に対して有効です。

データ圧縮を使用するには:

次の例では、1つの圧縮パーティション、costs_oldを使用してレンジ・パーティション化された表を作成します。表およびその他すべてのパーティションの圧縮属性は、表領域レベルから継承されます。

```
CREATE TABLE costs_demo (
   prod_id     NUMBER(6),     time_id     DATE,
   unit_cost   NUMBER(10,2), unit_price  NUMBER(10,2))
PARTITION BY RANGE (time_id)
   (PARTITION costs_old
      VALUES LESS THAN (TO_DATE('01-JAN-2003', 'DD-MON-YYYY')) COMPRESS,
    PARTITION costs_q1_2003
      VALUES LESS THAN (TO_DATE('01-APR-2003', 'DD-MON-YYYY')),
    PARTITION costs_q2_2003
      VALUES LESS THAN (TO_DATE('01-JUN-2003', 'DD-MON-YYYY')),
    PARTITION costs_recent VALUES LESS THAN (MAXVALUE));
```

4.5 スター・クエリーおよび3NFスキーマの最適化

Oracleデータ・ウェアハウスは、スター・スキーマおよび第3正規形のスキーマでの適切な動作が可能です。この項では、両方のタイプのスキーマでのパフォーマンスを最適化する重要な手法について説明します。スター・スキーマおよび3NFスキーマの概念上の背景については、第3正規形のスキーマについておよびスター・スキーマについてを参照してください。

スター・クエリーを使用する場合は、次の点を考慮する必要があります。

- スター・クエリーの最適化
- スター型変換の使用
- 第3正規形スキーマの最適化
- VECTOR GROUP BY集計を使用したスター・クエリーの最適化

4.5.1 スター・クエリーの最適化

スター・クエリーは、ファクト表といくつかのディメンション表を結合するものです。各ディメンション表は、主キーから外部キーへの結合を使用してファクト表に結合されますが、ディメンション表同士は結合されません。オプティマイザによってスター・クエリーが認識されると、スター・クエリーのための効率的な実行計画が生成されます。スター・クエリーのチューニングでは、スター・クエリーのパフォーマンスを向上させる方法について説明します。

4.5.1.1 スター・クエリーのチューニング

スター・クエリーのパフォーマンスを最大限に向上させるためには、次の基本的なガイドラインに従う必要があります。

- ビットマップ索引をファクト表の各外部キー列上に作成する必要があります。

- 初期化パラメータSTAR_TRANSFORMATION_ENABLEDをTRUEに設定する必要があります。これにより、スター・クエリーのための重要なオプティマイザ機能が使用可能になります。この機能は、下位互換性のためにデフォルトでFALSEに設定されています。

データ・ウェアハウスがこれらの条件を満たす場合、そのデータ・ウェアハウスで実行しているほとんどのスター・クエリーは、スター型変換と呼ばれる問合せ実行計画を使用します。スター型変換によって、スター・クエリーの問合せパフォーマンスが向上します。

4.5.2 スター型変換の使用

スター型変換は、元のスター・クエリーのSQLを暗黙的にリライト(または変換)することによる、強力な最適化テクニックです。エンド・ユーザーがスター型変換の詳細を知る必要はありません。Oracle Databaseの問合せオプティマイザでは、該当する場合にスター型変換が自動的に選択されます。

スター型変換は、スター・クエリーを効率的に実行することを目的とした問合せ変換です。Oracle Databaseでは、2つの基本フェーズを使用してスター・クエリーが処理されます。第1フェーズでは、ファクト表から必要な行(結果セット)のみを取り出します。この取出しにはビットマップ索引が使用されるため、非常に効率的です。第2フェーズでは、この結果セットをディメンション表に結合します。たとえば、「西部および南西部販売地域における過去3四半期の食料品部門の売上および利益はどうであったか」というエンド・ユーザーの問合せがあるとします。これは、1つの単純なスター・クエリーです。

この項では、次の項目について説明します。

- ビットマップ索引を使用したスター型変換
- ビットマップ索引を使用したスター型変換の実行計画
- ビットマップ結合索引を使用したスター型変換
- ビットマップ結合索引を使用したスター型変換の実行計画
- Oracleによるスター型変換の使用の選択
- スター型変換の制限

4.5.2.1 ビットマップ索引を使用したスター型変換

スター型変換の前提条件は、ファクト表の各結合列に単一列のビットマップ索引が存在することです。これらの結合列には、すべての外部キー列が含まれます。

たとえば、shサンプル・スキーマのsales表の場合、time_id列、channel_id列、cust_id列、prod_id列およびpromo_id列にビットマップ索引が定義されています。

次のスター・クエリーを考えてみます。

```
SELECT ch.channel_class, c.cust_city, t.calendar_quarter_desc,
   SUM(s.amount_sold) sales_amount
FROM sales s, times t, customers c, channels ch
WHERE s.time_id = t.time_id
AND   s.cust_id = c.cust_id
AND   s.channel_id = ch.channel_id
AND   c.cust_state_province = 'CA'
AND   ch.channel_desc in ('Internet','Catalog')
AND   t.calendar_quarter_desc IN ('1999-Q1','1999-Q2')
GROUP BY ch.channel_class, c.cust_city, t.calendar_quarter_desc;
```

この問合せは、2つのフェーズで処理されます。第1フェーズでは、Oracle Databaseは、ファクト表の外部キー列上のビットマップ索引を使用して、ファクト表から必要な行のみを識別し、取り出します。つまり、Oracle Databaseは、主に次の問合せを使用して、ファクト表から結果セットを取り出します。

```
SELECT ... FROM sales
WHERE time_id IN
  (SELECT time_id FROM times
   WHERE calendar_quarter_desc IN('1999-Q1','1999-Q2'))
   AND cust_id IN
  (SELECT cust_id FROM customers WHERE cust_state_province='CA')
   AND channel_id IN
  (SELECT channel_id FROM channels WHERE channel_desc IN('Internet','Catalog'));
```

元のスター・クエリーがこの副問合せ表現に変換されたことから、これがアルゴリズムの変換ステップになります。ファクト表にアクセスするこの方法は、ビットマップ索引の効果を高めます。ビットマップ索引は、リレーショナル・データベース内に集合ベースの処理方法を提供します。Oracleでは、AND(標準的な集合についての用語で共通部分の意味)、OR(集合の用語で和集合)、MINUS、COUNTなどの集合演算を実行するための非常に高速な方法が実装されています。

このスター・クエリーでは、time_idのビットマップ索引を使用して、1999年第1四半期(1999-Q1)の売上(sales)に対応するファクト表のすべての行集合が識別されます。この集合は、ビットマップ(ファクト表のどの行がこの集合のメンバーであるかを示す1および0(ゼロ)の文字列)として表されます。

同様のビットマップが、1999年第2四半期(1999-Q2)の売上に対応するファクト表の行に取り出されます。ビットマップOR演算を使用して、このQ1の売上集合をQ2の売上集合と組み合せます。

追加の集合演算が、customerディメンションおよびproductディメンションに対して実行されます。この時点で、スター・クエリー処理には3つのビットマップがあります。各ビットマップは、別々のディメンション表に対応し、それぞれ、個々のディメンションの絞込み条件を満たすファクト表の行集合を表します。

これらの3つのビットマップは、ビットマップAND演算を使用して単一ビットマップに結合されます。この最後のビットマップは、ファクト表のうちディメンション表上の絞込み条件をすべて満たす行集合を表します。これは結果セットであり、問合せの評価に必要なファクト表からの正確な行集合です。ここまでのところでは、ファクト表の実際のデータには、まだアクセスしていないことに注意してください。これらの演算はすべて、ビットマップ索引とディメンション表のみを基にしています。ビットマップ索引はデータを圧縮した形で表すため、ビットマップの集合ベース演算は非常に効率的です。

結果セットが識別されると、ビットマップはSALES表から実データへのアクセスに使用されます。エンド・ユーザーの問合せに必要な行のみが、ファクト表から取り出されます。この時点で、すべてのディメンション表が、ビットマップ索引を使用してファクト表に効率的に結合されています。Oracle Databaseでは、各ディメンション表をファクト表に個別に結合するのではなく、すべてのディメンション表を単一の論理結合演算でファクト表に結合しているため、この技法を使用すると優れたパフォーマンスが得られます。

この問合せの第2フェーズでは、ファクト表の行(結果セット)をディメンション表に結合します。Oracleは、最も効率的な方法を使用して、ディメンション表にアクセスおよび結合します。ほとんどのディメンションは非常に小規模なため、これらのディメンション表への最も効率的なアクセス方法は、通常、表スキャンです。大規模なディメンション表については、表スキャンは最も効率的なアクセス方法ではない場合があります。前述の例では、product.departmentのビットマップ索引を使用して、食料品部門のすべての製品が高速で識別されます。Oracle Databaseでは、各ディメンション表のサイズおよびデータ分散に関するオプティマイザの知識に基づいて、オプティマイザが、特定のディメンション表に最適のアクセス方法を自動的に判断します。

同様に、各ディメンション表用の特定の結合方法(および索引付け方法)も、オプティマイザによってインテリジェントに判断されます。多くの場合、ディメンション表を結合するための最も効率的なアルゴリズムはハッシュ結合です。すべてのディメンション表が結合されると、最終結果がユーザーに戻されます。1つの表から一致する行のみを取り出してから、別の表に結合する問合せテクニックは、一般にセミ結合と呼ばれます。

4.5.2.2 ビットマップ索引を使用したスター型変換の実行計画

ビットマップ索引を使用したスター型変換から得られる典型的な実行計画は次のようになります。

```
SELECT STATEMENT
 SORT GROUP BY
```

```
 HASH JOIN
  TABLE ACCESS FULL                          CHANNELS
  HASH JOIN
   TABLE ACCESS FULL                         CUSTOMERS
   HASH JOIN
    TABLE ACCESS FULL                        TIMES
    PARTITION RANGE ITERATOR
     TABLE ACCESS BY LOCAL INDEX ROWID       SALES
      BITMAP CONVERSION TO ROWIDS
       BITMAP AND
        BITMAP MERGE
         BITMAP KEY ITERATION
          BUFFER SORT
           TABLE ACCESS FULL                 CUSTOMERS
          BITMAP INDEX RANGE SCAN            SALES_CUST_BIX
        BITMAP MERGE
         BITMAP KEY ITERATION
          BUFFER SORT
           TABLE ACCESS FULL                 CHANNELS
          BITMAP INDEX RANGE SCAN            SALES_CHANNEL_BIX
        BITMAP MERGE
         BITMAP KEY ITERATION
          BUFFER SORT
           TABLE ACCESS FULL                 TIMES
          BITMAP INDEX RANGE SCAN            SALES_TIME_BIX
```

この計画では、ファクト表は、ビットマップ・アクセス・パスを介してアクセスされます。このパスは、3つのマージされたビットマップのビットマップANDに基づきます。この3つのビットマップは、下位の行ソース・ツリーからビットマップが提供されているBITMAP MERGE行ソースによって生成されます。このような行ソース・ツリーはそれぞれ、副問合せ行ソース・ツリーの値をフェッチするBITMAP KEY ITERATION行ソースで構成されています。この例では、副問合せ行ソース・ツリーは、1つの全表アクセスです。これらの各値について、BITMAP KEY ITERATION行ソースがビットマップをビットマップ索引から取り出します。関係するファクト表の行は、このアクセス・パスを使用して取り出された後に、問合せ結果を生成するためにディメンション表および一時表と結合されます。

4.5.2.3 ビットマップ結合索引を使用したスター型変換

スター型変換において、ビットマップ索引の他にビットマップ結合索引も使用できます。次に示すような追加の索引構造があるとします。

```
CREATE BITMAP INDEX sales_c_state_bjix
ON sales(customers.cust_state_province)
FROM sales, customers
WHERE sales.cust_id = customers.cust_id
LOCAL NOLOGGING COMPUTE STATISTICS;
```

同じスター・クエリーでビットマップ結合索引を使用した場合の処理は、前述の例に似ています。唯一の違いは、単一表のビットマップ索引のかわりに、結合インデックスを使用してスター・クエリーの第1フェーズで顧客データにアクセスすることです。

4.5.2.4 ビットマップ結合索引を使用したスター型変換の実行計画

ビットマップ結合索引を使用したスター型変換の実行計画から得られる典型的な実行計画は次のようになります。

```
SELECT STATEMENT
 SORT GROUP BY
  HASH JOIN
   TABLE ACCESS FULL                         CHANNELS
   HASH JOIN
    TABLE ACCESS FULL                        CUSTOMERS
    HASH JOIN
```

```
    TABLE ACCESS FULL                       TIMES
    PARTITION RANGE ALL
     TABLE ACCESS BY LOCAL INDEX ROWID      SALES
      BITMAP CONVERSION TO ROWIDS
       BITMAP AND
        BITMAP INDEX SINGLE VALUE           SALES_C_STATE_BJIX
        BITMAP MERGE
         BITMAP KEY ITERATION
          BUFFER SORT
           TABLE ACCESS FULL                CHANNELS
          BITMAP INDEX RANGE SCAN           SALES_CHANNEL_BIX
        BITMAP MERGE
         BITMAP KEY ITERATION
          BUFFER SORT
           TABLE ACCESS FULL                TIMES
          BITMAP INDEX RANGE SCAN           SALES_TIME_BIX
```

この計画と前述の計画との唯一の違いは、customerディメンションのビットマップ索引スキャン内の選択のための処理がないことです。これは、customer.cust_state_provinceの結合述語情報を、ビットマップ結合索引sales_c_state_bjixで満たすことができるためです。

4.5.2.5 Oracleによるスター型変換の使用の選択

オプティマイザは、変換なしでも生成できる最適な計画を生成して保存します。変換が有効な場合、オプティマイザは問合せを変換し、必要に応じて、その変換された問合せを使用して最適な計画を生成します。オプティマイザは、この2つの問合せに対する最適な計画のコスト概算を比較して、変換または未変換のどちらの最適な計画を使用するかを決定します。

問合せがファクト表の行の大部分にアクセスする必要があるときは、変換ではなく、全表スキャンを使用する方がよい場合があります。ただし、ディメンション表について、選択による絞込み度合いが高く、ファクト表のわずかな部分のみを取り出す必要がある場合は、変換に基づく計画の方が適していることもあります。

オプティマイザは、多くの基準に基づいて適切であると判断した場合にのみ、ディメンション表に対して副問合せを生成します。副問合せは、すべてのディメンション表に対して生成されるわけではありません。また、オプティマイザが、表および問合せの性質に基づいて、問合せに変換を適用するメリットがないと判断することもあります。このような場合は、最適な通常の計画が使用されます。

4.5.2.6 スター型変換の制限

スター型変換は、次の特性が1つでもある表ではサポートされません。

- ビットマップ・アクセス・パスと非互換の表ヒントがある問合せ。
- ビットマップ索引が少なすぎる表。オプティマイザが副問合せを生成するためには、ファクト表の列にビットマップ索引がある必要があります。
- リモート・ファクト表。ただし、リモート・ディメンション表は、生成された副問合せでは有効です。
- アンチ結合された表。
- 副問合せでディメンション表として使用済の表。
- ビュー・パーティションではなく、実際はマージされていないビューである表。
- ファクト表がマージされていないビューである表
- パーティション・ビューがファクト表として使用されている表

次の場合には、オプティマイザではスター型変換が選択されないことがあります。

- 効率的な単一表アクセス・パスを持つ表
- 小さすぎて変換によるメリットがない表

さらに、次の条件下では、スター型変換で一時表は使用されません。

- データベースが読取り専用モードの場合
- スター・クエリーがシリアル化可能モードでのトランザクションの一部である場合

4.5.3 第3正規形スキーマの最適化

第3正規形(3NF)スキーマの最適化では、次の要件が必要です。

- パワー

  パワーとは、ハードウェア構成のバランスがとれている必要があるということです。多くのデータ・ウェアハウス操作は、大規模な表スキャンおよび大量のランダムI/Oを実行する他のI/O集中型操作に基づいています。最適なパフォーマンスを達成するため、ハードウェア構成のサイズをエンド・ツー・エンドで設定して、このレベルのスループットを維持する必要があります。このタイプのハードウェア構成は、バランスのとれたシステムと呼ばれます。バランスのとれたシステムでは、CPUからディスクに至るまでのすべてのコンポーネントをまとめて編成して、利用可能な最大のI/Oスループットを保証します。

- パーティション化

  大規模な表は、コンポジット・パーティション化(レンジ-ハッシュまたはリスト-ハッシュ)を使用してパーティション化する必要があります。これには次の3つの理由があります。

  - 数TBのデータの管理が容易
  - 必要なデータへのアクセシビリティの向上
  - 効果的でパフォーマンスの高い表の結合

  3NFスキーマ: パーティション化を参照してください。

- パラレル実行

  パラレル実行により、データベース・タスクをパラレル化または小さい作業単位に分割できるため、複数のプロセスを同時に実行できます。パラレル化を使用すると、TB単位のデータを時間単位や日単位ではなく、数分以内でスキャンして処理できます。

  3NFスキーマ: パラレル問合せの実行を参照してください。

4.5.3.1 3NFスキーマ: パーティション化

パーティション化により、表、索引および索引構成表をより細かい単位に細分化できるようになります。データベース・オブジェクトの単位をパーティションと呼びます。各パーティションには独自の名前があり、独自の記憶特性を持つことができます。データベース管理者の視点からすると、パーティション・オブジェクトには、まとめて管理することも個別に管理することも可能な複数の単位があります。

このため、管理者は、パーティション・オブジェクトをかなり柔軟に管理できます。ただし、アプリケーションにとっては、パーティション表は非パーティション表と同じであるため、SQL DMLコマンドを使用してパーティション表にアクセスする際に変更は必要ありません。パーティション化によって、管理性、可用性およびパフォーマンスが向上し、多様なアプリケーションに大きなメリットがもたらされます。

4.5.3.1.1 管理性のためのパーティション化

レンジ・パーティション化により、大量のデータの管理性および可用性を向上できます。2年分の売上データまたは100TBが表

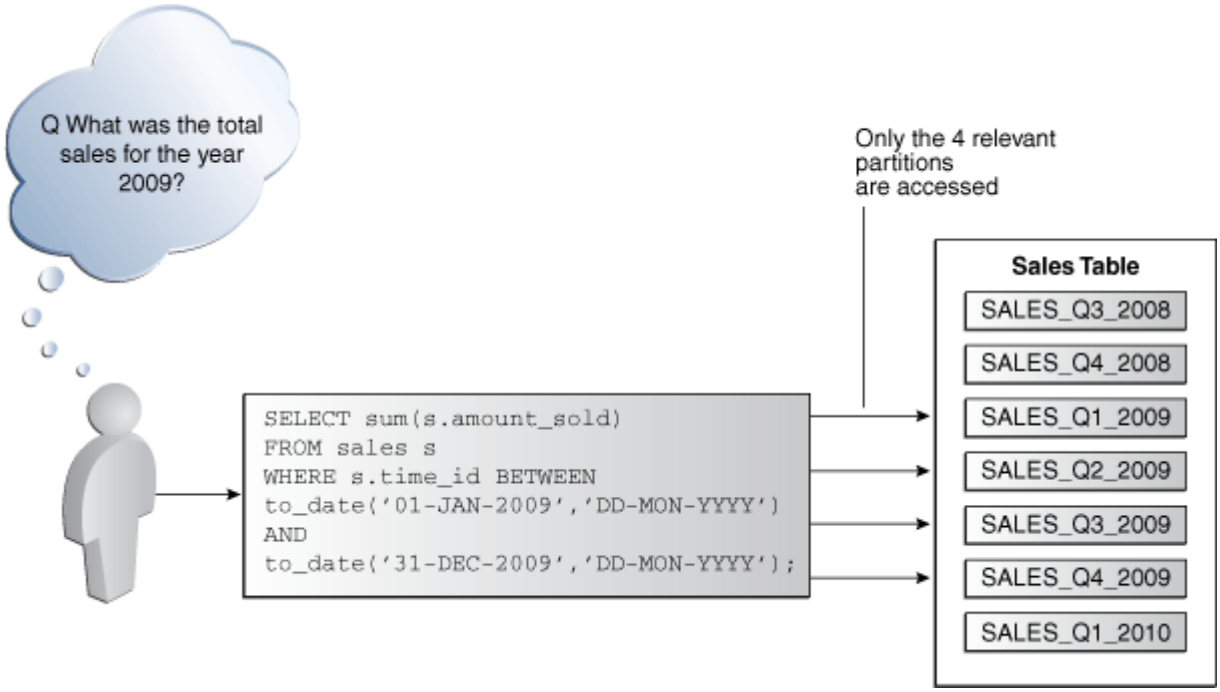
に格納されている事例を検討します。1日の最後に、新しいバッチのデータを表にロードして、最も古い日のデータを削除する必要があります。売上表を日単位でレンジ・パーティション化する場合、パーティション交換ロードを使用して、新しいデータをロードできます。これは、エンド・ユーザー問合せにほとんど影響しない1秒以内の操作です。最も古い日付のデータの削除は、次のコマンドを発行するのみです。

```
SH@DBM1 > ALTER TABLE SALES DROP PARTITION Sales_Q4_2009;
```

4.5.3.1.2 簡易データ・アクセスのためのパーティション化

また、レンジ・パーティション化により、問合せに応答するための必要なデータのみスキャンすることを確認できます。ビジネス・ユーザーが週単位、たとえば、週ごとの売上合計などで売上データに主にアクセスすると想定する場合、日単位でこの表をレンジ・パーティション化すると、全体の表ではなく4つのパーティションのみをスキャンしてビジネス・ユーザーの問合せに応答するため、データに最も効率的な方法でアクセスできます。関係ないパーティションのスキャンを回避する機能は、パーティション・プルーニングと呼ばれます。

図4-3 パーティション・プルーニング



Oracle Database 12cリリース2 (12.2)以降は、外部表にパーティションを定義できます。外部表はデータベース内に存在しない表であり、アクセス・ドライバが用意されている任意の形式が可能です。パーティション化された外部表のファイルは、ファイル・システム、Apache Hive記憶域またはHadoop Distributed File System (HDFS)に格納できます。

外部表をパーティション化すると、問合せパフォーマンスが向上し、データ保守が容易になります。また、外部表でパフォーマンスの最適化の利点(データベースに格納されているパーティション表で利用できるパーティション・プルーニング、パーティション・ワイズ結合など)を活用できます。データベースの表でサポートされるほとんどのパーティション化テクニック(ハッシュ・パーティション化を除く)は、パーティション化された外部表でもサポートされます。ただし、Oracle Databaseは、パーティションの外部ストレージ・ファイルにパーティション条件を満たすデータが含まれることを保証できません。

関連項目:

パーティション外部表の詳細は、『Oracle Database管理者ガイド』を参照してください。

4.5.3.1.3 結合パフォーマンスのパーティション化

パフォーマンスのため、ハッシュによるサブパーティション化を主に使用します。Oracleは、線形ハッシング・アルゴリズムを使用して、

サブパーティションを作成します。ハッシュ・パーティション間でデータを確実に均等に分散するには、ハッシュ・パーティションの数を2の累乗(たとえば、2、4、8など)にすることをお薦めします。各ハッシュ・パーティションのサイズは少なくとも16MB必要です。これより少ない場合は、パラレル問合せで効果的なスキャン率がでません。

ハッシュ・パーティション化の主なパフォーマンスの利点の1つは、パーティション・ワイズ結合です。パーティション・ワイズ結合では、結合がパラレルで実行されるときにパラレル実行サーバー間で交換されるデータ量が最小限に抑えられ、問合せのレスポンス時間が短縮されます。レスポンス時間は大幅に短縮され、CPUとメモリー・リソースの使用率が改善されます。クラスタ化されたデータ・ウェアハウスで、大規模な結合操作で優れたスケーラビリティを実現するために重要なインターコネクト(IPC)でのデータ・トラフィックを制限して、レスポンス時間が大幅に短縮されます。パーティション・ワイズ結合は、結合する表のパーティション・スキームによってフルまたはパーシャルになります。

フル・パーティション・ワイズ結合は、2つの大きい表の結合を複数の小さい結合に分割します。小さい各結合は、結合される表ごとに、パーティションのペアの結合を実行します。フル・パーティション・ワイズ結合方法を選択するオプティマイザには、両方の表を結合キーでパーティション化する必要があります。つまり、同じパーティション化方法を使用した同じ列でパーティション化する必要があります。フル・パーティション・ワイズ結合のパラレル実行はシリアル実行と似ていますが、一度に1つのパーティション・ペアを結合するかわりに複数のパーティション・ペアを複数のパラレル問合せサーバーで結合する点が異なります。パラレルで結合するパーティションの数は、並列度(DOP)によって決定されます。

図4-4 フル・パーティション・ワイズ結合

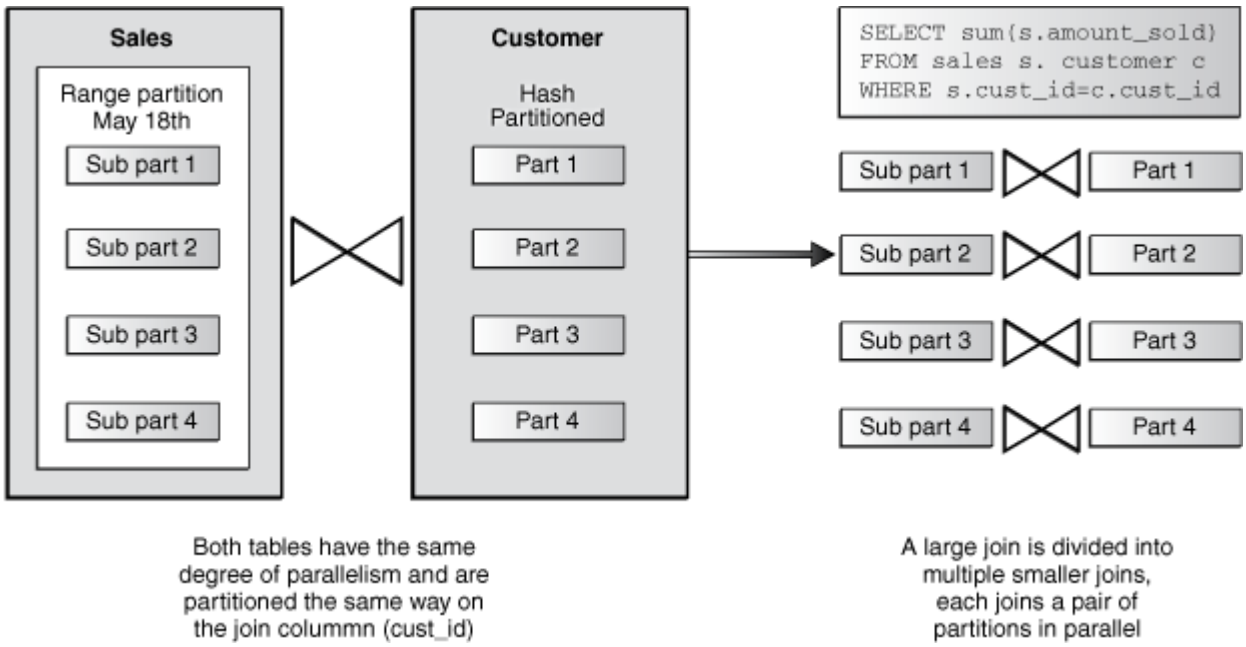


図4-4に、SalesおよびCustomersの2つの表間のフル・パーティション・ワイズ結合のパラレル実行を示します。両方の表の並列度およびパーティション数は同じです。日付フィールドでレンジ・パーティション化され、cust_idフィールドでハッシュを使用してサブパーティション化されます。図に示すように、各パーティション・ペアがデータベースから読み取られ、直接結合されます。データの再分散が必要ないため、特にノード間でIPC通信を最小化します。次の図4-5に、この結合を確認する実行計画を示します。

パラレルでパーティション・ワイズ結合を実行する場合に最適なパフォーマンスを取得するには、各表のパーティションの数が結合に使用する並列度よりも大きい必要があります。パラレル・サーバーよりも多くのパーティションがある場合、各パラレル・サーバーに結合するパーティションの1つのペアが提供され、パラレル・サーバーが結合を完了すると、結合する別のペアのパーティションがリクエストされます。すべてのペアが処理されるまで、このプロセスが繰り返されます。この方法により、動的にロード・バランスできます(たとえば、並列度32の128個のパーティションなど)。

結合している表の1つのみをパーティション化するとどうなるのでしょうか。この場合、オプティマイザはパーシャル・パーティション・ワイズ結合を選択できます。フル・パーティション・ワイズ結合とは異なり、1つの表のみが結合キーでパーティション化されている場合にパーシャル・パーティション・ワイズ結合を適用できます。したがって、パーシャル・パーティション・ワイズ結合は、フル・パーティション・ワイズ結合よりも一般的です。パーシャル・パーティション・ワイズ結合を実行するために、Oracleによって、パーティション

表のパーティション化戦略に基づいて、もう一方の表が動的に再パーティション化されます。もう一方の表が再パーティション化された後は、フル・パーティション・ワイズ結合と同様に実行されます。再分散操作には、パラレル実行サーバー間での行の交換が伴います。データをノード境界で再パーティション化する必要があるため、この操作は、Oracle RAC環境のインターコネクト・トラフィックにつながります。

図4-5 パーシャル・パーティション・ワイズ結合

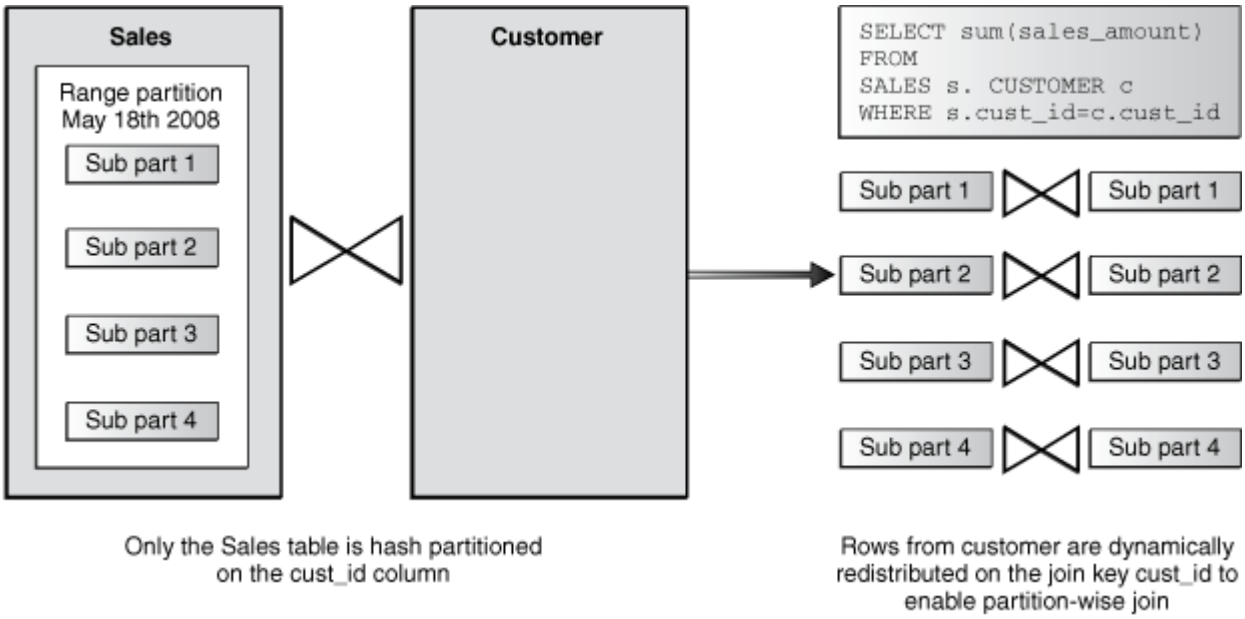


図4-5に、パーシャル・パーティション・ワイズ結合を示します。図4-4と同じ例を使用しますが、顧客表がパーティション化されない点が異なります。結合操作を実行する前に、顧客表の行が結合キーで動的に再分散されます。

4.5.3.2 3NFスキーマ: パラレル問合せの実行

3NFスキーマではパラレル化を複数の方法で活用できますが、ここでは3NFにとって特別な意味のある、パラレル化の1つの側面に注目します。それは大規模問合せでのSQLパラレル実行です。Oracle DatabaseでのSQLパラレル実行は、コーディネータ(通常、問合せコーディネータまたはQCと呼ばれる)およびパラレル・サーバーの原則に基づきます。QCはパラレルSQL文を起動するセッションで、パラレル・サーバーはパラレルで動作する個別のセッションです。QCは処理をパラレル・サーバーに分散するため、パラレルで実行できない最低限の主にロジスティックな処理を実行する必要があります。たとえば、SUM()演算を含むパラレル問合せでは、各パラレル・サーバーで計算された小計それぞれを合計する必要があります。

図4-5のPX COORDINATORで示すとおり、QCはパラレル実行で容易に識別できます。パラレルSQL操作のQCとして動作するプロセスは、実際のユーザー・セッション・プロセスそのものです。パラレル・サーバーは、グローバルに使用可能なパラレル・サーバーのプロセスのプールから取得され、特定の操作に割り当てられます。パラレル・サーバーは、QCの下位のパラレル計画にあるすべての処理を実行します。

Oracle Databaseは、デフォルトで設定不要なパラレル実行をサポートするよう構成されており、2つの初期化パラメータ、parallel_max_serversおよびparallel_min_serversで制御されます。パラレル実行は非常に強力でスケーラブルなフレームワークを提供し、SQL操作をスピードアップしますが、常識的なルールの適用を忘れてはいけません。つまり、パラレル実行によりさらなるパフォーマンスの向上が見込めますが、より多くのリソースが必要になり、同じシステムの他のユーザーや操作に影響する可能性があります。小規模の表や索引(数千レコードまでのもの、数十データ・ブロックまでのもの)では、パラレル実行を有効にしないでください。小規模表のみに該当する操作には、パラレルで実行するメリットがあまりありませんが、大規模表にアクセスする操作で使用するパラレル・サーバーが使用されます。特定の並列度(DOP)でいったん操作が始まると、実行中にDOPを下げる手段がないことにも注意してください。

あるオブジェクトに対して、適切なDOPを決定する一般的な経験則は次のとおりです。

- 200 MB未満のオブジェクトではパラレル化を使用しないでください。

- 200 MBから5GBまでのオブジェクトではDOPを4にする必要があります。
- 5GB超のオブジェクトでは、DOPを32にしてください。

いうまでもなく、サイズ・レンジやDOPにおいてシステムの最適な設定は異なる上、対象のワークロード、ビジネス要件およびハードウェア構成に高く依存します。Oracle RACにおけるインスタンス間パラレル実行の使用の可否では、Oracle RAC環境でのパラレル実行について説明しています。

4.5.3.2.1 Oracle RACにおけるインスタンス間パラレル実行の使用の可否

Oracle Databaseでは、デフォルトでノード間パラレル実行が可能です(複数ノードを含む単一の文のパラレル実行)。前述のとおり、ノード間パラレル実行によってインターコネクト・トラフィックが増大する可能性があるため、Oracle RAC環境でのインターコネクトのサイズが適切である必要があります。サーバーからストレージ・サブシステムへのI/O帯域幅に対し、比較的弱いインターコネクトを使用している場合、パラレル実行を単一ノードまたは限られたノード数に制限することをお薦めします。ノード間パラレル実行は、インターコネクトのサイズが小さいと対応できません。Oracle Database 11*g*以降では、Oracle RACサービスを使用して、クラスタ上でパラレル実行を制御することをお薦めします。

4.5.4 VECTOR GROUP BY集計を使用したスター・クエリーの最適化

VECTOR GROUP BY集計は、データを集計し、1つ以上の比較的小さな表をより大きな表に結合する問合せを最適化します。この変換は、SQLオプティマイザによって、コスト見積もりに基づいて選択されることがあります。データ・ウェアハウジングのコンテキストでは、VECTOR GROUP BYは、インメモリー列表からデータを選択するスター・クエリーの場合によく選択されます。

VECTOR GROUP BY集計は、小さな表と大きな表の間の結合条件を、より大きな表の上のフィルタに変換するという点で、ブルーム・フィルタと類似しています。VECTOR GROUP BY集計は、ファクト表のスキャン後に別のステップとしてデータを集計するのでなく、スキャン時に行うことによって、問合せパフォーマンスをさらに向上させます。

関連項目:

- インメモリー集計を使用する場合
- VECTOR GROUP BYシナリオの詳細は、Oracle Database In-Memoryガイドを参照してください。

4.6 近似問合せ処理について

近似問合せ処理では、SQL関数を使用して、近似処理が許容される探索的問合せに対してリアルタイムの応答を提供します。近似結果を返すSQL関数を含む問合せは、近似問合せと呼ばれます。

ビジネス・インテリジェンス(BI)アプリケーションは、集計関数(分析関数を含む)を広範囲に使用して、一般的なビジネス問合せに応答します。一部のタイプの問合せでは、データ・セットが非常に大きい場合、正確に応答するためにリソースが集中する可能性があります。たとえば、Webサイトの一意の顧客のセッション数をカウントしたり、ある州の各郵便番号内の住宅価格の中央値を算出する場合などです。特定のシナリオでは、近似傾向またはパターン(これらを使用してさらに分析を実行できます)により関心があるため、これらのタイプの問合せには正確な応答が必要ない場合があります。近似問合せ処理は、主にデータ検出アプリケーションで使用され、探索的問合せに対して迅速な応答を返します。通常、ユーザーは大量のデータから対象となるデータ・ポイントを見つけ、これをドリルダウンして詳細をさらに明らかにしようとします。探索的問合せの場合は、正確な値よりも迅速な応答が重要です。

Oracleでは、正確な結果との誤差がごくわずかな近似結果を取得できるSQL関数のセットを提供しています。その他にマテリアライズド・ビューに基づくサマリー集計戦略をサポートする近似関数もあります。近似結果を提供する関数は、次のとおりです。

- APPROX_COUNT_DISTINCT
- APPROX_COUNT_DISTINCT_DETAIL
- APPROX_COUNT_DISTINCT_AGG
- TO_APPROX_COUNT_DISTINCT
- APPROX_MEDIAN
- APPROX_PERCENTILE
- APPROX_PERCENTILE_DETAIL
- APPROX_PERCENTILE_AGG
- TO_APPROX_PERCENTILE
- APPROX_COUNT
- APPROX_RANK
- APPROX_SUM

近似問合せ処理は、既存のコードを変更せずに使用できます。適切な初期化パラメータを設定すると、Oracle Databaseでは、近似結果を返す対応するSQL関数で問合せ内の正確な関数が置き換えられます。

関連項目:

- 近似値を返すSQL関数を使用した、正確な関数を含む問合せの実行
- 近似問合せに基づいたマテリアライズド・ビューの作成
- 近似問合せに基づいたクエリー・リライトおよびマテリアライズド・ビュー
- SQL関数の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

4.6.1 近似値を返すSQL関数を使用した、正確な関数を含む問合せの実行

正確な関数を含む問合せは、問合せを変更せずに、近似結果を返す対応するSQL関数を使用して実行できます。これにより、近似結果を返す対応するSQL関数を使用して、既存のアプリケーションを変更せずに実行できます。

Oracle Databaseには、正確な関数を、実行時に近似結果を返す対応するSQL関数と置き換える必要があることを示す初期化パラメータ(approx_for_aggregation、approx_for_count_distinctおよびapprox_for_percentile)が用意されています。すべての正確な関数は、近似結果を返す対応する関数と実行時に置き換えることができます。対応する近似バージョンと置き換える必要がある関数のリストをより高度に制御する必要がある場合は、実行時に置き換える必要がある関数のタイプを指定できます。たとえば、問合せにCOUNT(DISTINCT)が含まれている場合は、approx_for_aggregationをTRUEに設定すると、この問合せはCOUNT(DISTINCT)ではなくAPPROX_COUNT_DISTINCTを使用して実行されます。

- 指定したSQL関数ではなく、近似結果を返す対応するSQL関数を使用してすべての問合せを実行するには:

  現行セッションまたはデータベース全体に対してapprox_for_aggregation初期化パラメータをTRUEに設定します。このパラメータは、近似結果を返す関数の使用を有効にするためのアンブレラ・パラメータとして機能します。この設定は、APPROX_COUNT_DISTINCTおよびAPPROX_FOR_PERCENTILEパラメータの設定と同等です。

  次のコマンドは、現行セッションに対してapprox_for_aggregationをtrueに設定します。

  ```
  alter session set approx_for_aggregation = TRUE;
  ```

- 問合せのCOUNT(DISTINCT)関数のみをAPPROX_COUNT_DISTINCT関数に置き換えるには:

現行セッションまたはデータベース全体に対してapprox_for_count_distinct初期化パラメータをTRUEに設定します。

- パーセンタイル関数を、近似結果を返す対応する関数に置き換えるには:

  現行セッションまたはデータベース全体に対して、approx_for_percentileをPERCENTILE_CONT、PERCENTILE_DISCまたはALL (すべてのパーセンタイル関数を置き換えます)に設定します。このパラメータのデフォルト値はNONEです。

関連項目:

- 『Oracle Databaseリファレンス』のAPROX_FOR_AGGREGATIONに関する説明
- 『Oracle Databaseリファレンス』のAPPROX_FOR_COUNT_DISTINCTに関する説明
- 『Oracle Databaseリファレンス』のAPPROX_FOR_PERCENTILEに関する説明

4.7 近似上位N問合せ処理について

Oracle Databaseリリース18から、上位N問合せの結果を従来の問合せよりも高速に取得するために、SQL関数APPROX_COUNTおよびAPPROX_SUMをAPPROX_RANKと組み合せて使用できます。

APPROX_COUNT

APPROX_COUNTは式の近似カウントを返します。2番目の引数としてMAX_ERRORが指定された場合、この関数は実際のカウントと近似カウントの間の最大エラーを返します。

この関数は、HAVING句の対応するAPPROX_RANK関数とともに使用する必要があります。問合せでAPPROX_COUNT、APPROX_SUMまたはAPPROX_RANKを使用する場合、問合せで他の集計関数は使用しないでください。

関連項目:

- Oracle Database SQL言語リファレンス
- APPROX_RANK関数

APPROX_SUM

APPROX_SUMは式の近似合計を返します。2番目の引数としてMAX_ERRORが指定された場合、この関数は実際の合計と近似合計の間の最大エラーを返します。

この関数は、HAVING句の対応するAPPROX_RANK関数とともに使用する必要があります。問合せでAPPROX_COUNT、APPROX_SUMまたはAPPROX_RANKを使用する場合、問合せで他の集計関数は使用しないでください。

ノート:

入力が負の数の場合、APPROX_SUM はエラーを返します。

関連項目:

- Oracle Database SQL言語リファレンス
- APPROX_RANK関数

第II部 データ・ウェアハウスの最適化

この部では、データ・ウェアハウスの物理設計について説明します。

この部は、次の章で構成されています。

- 基本的なマテリアライズド・ビュー
- 高度なマテリアライズド・ビュー
- マテリアライズド・ビューのリフレッシュ
- 同期リフレッシュ
- マテリアライズド・ビューのリフレッシュ操作の監視
- ディメンション
- マテリアライズド・ビューのための基本的なクエリー・リライト
- マテリアライズド・ビューのための高度なクエリー・リライト
- 属性クラスタリング
- ゾーン・マップの使用

5 基本的なマテリアライズド・ビュー

この章では、マテリアライズド・ビューの使用方法について説明します。次の項目が含まれます。

- マテリアライズド・ビューを使用したデータ・ウェアハウスの概要
- マテリアライズド・ビューのタイプ
- マテリアライズド・ビューの作成
- マテリアライズド・ビュー・ログの作成
- 近似問合せに基づいたマテリアライズド・ビューの作成
- 既存のマテリアライズド・ビューの登録
- マテリアライズド・ビューに対する索引付けの選択
- マテリアライズド・ビューの削除
- マテリアライズド・ビュー機能の分析

5.1 マテリアライズド・ビューを使用したデータ・ウェアハウスの概要

一般的に、データは月、週または日単位で、1つ以上のオンライン・トランザクション処理(OLTP)データベースからデータ・ウェアハウスに送られます。データは通常、データ・ウェアハウスに追加される前に ステージング・ファイルで処理されます。データ・ウェアハウスのサイズは、一般的に、数百GBから数TBの範囲にわたります。通常、データの大半はいくつかの非常に大規模なファクト表に格納されます。

パフォーマンス向上のためにデータ・ウェアハウスで使用されている1つのテクニックに、サマリーの作成があります。これは、特殊なタイプの集計ビューであり、問合せを実行する前に、効率が悪い結合および集計操作を事前に計算し、その結果をデータベース内の表に格納することで、問合せ実行時間を短縮します。たとえば、サマリー表が、地域別および製品別の売上合計を含むように作成できます。

このマニュアルおよびデータ・ウェアハウスに関するマニュアルで言及されているサマリーまたは集計は、マテリアライズド・ビューと呼ばれるスキーマ・オブジェクトを使用して、Oracle Databaseに作成されます。マテリアライズド・ビューは、問合せパフォーマンスの改善、レプリケートされたデータの提供などの多くの役割で使用できます。

データベース管理者は、サマリーと同じ結果を格納するマテリアライズド・ビューを1つ以上作成します。エンド・ユーザーは、表やビューをディテール・データ・レベルで問い合せます。SQL問合せは、Oracleサーバーのクエリー・リライト・メカニズムにより、サマリー・テーブルを使用するように自動的にリライトされます。このメカニズムにより、問合せから結果を戻すための応答時間が短縮されます。データ・ウェアハウス内のマテリアライズド・ビューは、エンド・ユーザーやデータベース・アプリケーションに対して透過的です。

マテリアライズド・ビューは、通常、クエリー・リライト機能を使用してアクセスされますが、エンド・ユーザーまたはデータベース・アプリケーションは、マテリアライズド・ビューに直接アクセスする問合せも作成できます。ただし、マテリアライズド・ビューに変更があるとそれを参照する問合せに影響するため、ユーザーにこれを許可するかどうかについては慎重に検討する必要があります。

この項では、次の項目について説明します。

- データ・ウェアハウスでのマテリアライズド・ビューについて
- 分散コンピューティングでのマテリアライズド・ビューについて
- モバイル・コンピューティングでのマテリアライズド・ビューについて

- マテリアライズド・ビューの必要性
- サマリー管理のコンポーネント
- データ・ウェアハウスの用語
- マテリアライズド・ビューのスキーマ・デザインについて
- データ・ウェアハウスへのデータのロードについて
- マテリアライズド・ビューの管理作業の概要

5.1.1 データ・ウェアハウスでのマテリアライズド・ビューについて

データ・ウェアハウスでは、マテリアライズド・ビューを使用して、売上合計などの集計データを事前に計算し格納できます。これらの環境では、マテリアライズド・ビューにサマリー・データが格納されるため、マテリアライズド・ビューは、サマリーとして参照されます。また、マテリアライズド・ビューを使用して、集計の有無にかかわらず、結合を事前に計算できます。マテリアライズド・ビューによって、大規模または重要な問合せの、コストの高い結合や集計によって発生するオーバーヘッドを回避できます。

5.1.2 分散コンピューティングでのマテリアライズド・ビューについて

分散環境では、マテリアライズド・ビューを使用して、分散サイトでデータをレプリケートし、各サイトで競合解消方法を使用して実行された更新を同期できます。複製としてのマテリアライズド・ビューによって、本来はリモート・サイトからアクセスする必要があるデータに、ローカルにアクセスできます。マテリアライズド・ビューは、リモート・データ・マートでも有効です。

関連項目:

『Oracle Database Heterogeneous Connectivityユーザーズ・ガイド』

5.1.3 モバイル・コンピューティングでのマテリアライズド・ビューについて

マテリアライズド・ビューを使用して、クライアントとセントラル・サーバー間で定期的にリフレッシュおよび更新を行い、データのサブセットをセントラル・サーバーからモバイル・クライアントにダウンロードすることもできます。この章では、データ・ウェアハウスでのマテリアライズド・ビューの使用方法を中心に説明しています。

関連項目:

『Oracle Database Heterogeneous Connectivityユーザーズ・ガイド』

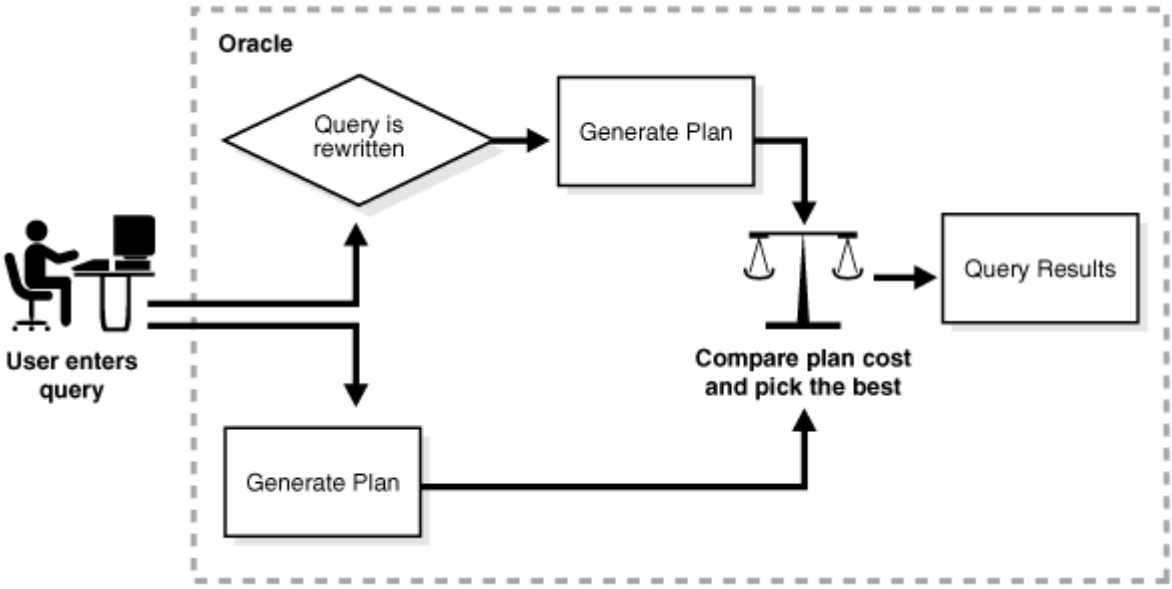
5.1.4 マテリアライズド・ビューの必要性

マテリアライズド・ビューを使用すると、非常に大規模なデータベースに対する問合せの速度が上がります。大規模データベースへの問合せには、多くの場合、表間の結合またはSUMなどの集計(あるいはその両方)が伴います。これらの操作は、時間および処理能力の点でコストが高くなります。作成するマテリアライズド・ビューのタイプによって、マテリアライズド・ビューのリフレッシュ方法およびクエリー・リライトによる使用方法が決まります。

マテリアライズド・ビューは、問合せを実行する前にコストの高い結合および集計操作をデータベース上で事前に計算し、その結果をデータベースに格納することで、問合せのパフォーマンスを改善します。問合せオプティマイザでは、問合せの要求を満たすの

に既存のマテリアライズド・ビューが使用可能かどうか、また必要かどうかが自動的に認識されます。そして、使用可能であれば問合せオプティマイザは、マテリアライズド・ビューを使用するように、問合せを透過的にリライトします。その結果問合せは、ベースとなるディテール表ではなく、マテリアライズド・ビューに対して直接実行されます。一般に、ディテール表ではなくマテリアライズド・ビューを使用するように問合せをリライトすると、応答のパフォーマンスが改善されます。クエリー・リライトがどのように機能するかを図5-1に示します。

図5-1 透過的なクエリー・リライト



クエリー・リライトを使用する場合は、できるだけ多くの問合せを満たすマテリアライズド・ビューを作成します。たとえば、ディテール表またはファクト表に共通して適用される20の問合せがわかっている場合、適切に作成されたマテリアライズド・ビューが5、6個あればこれらの問合せを満たせる場合があります。マテリアライズド・ビュー定義には、任意の数の集計(SUM、COUNT(x)、COUNT(*)、COUNT(DISTINCT x)、AVG、VARIANCE、STDDEV、MINおよびMAX)を含めることができます。また、任意の数の結合を含めることもできます。どのようなマテリアライズド・ビューを作成する必要があるかわからない場合のために、Oracle DatabaseにはSQLアクセス・アドバイザが用意されています。これは、DBMS_ADVISORパッケージに含まれる一連のアドバイザ・プロシージャで、クエリー・リライトのためにマテリアライズド・ビューを設計および評価する場合に有効です。

マテリアライズド・ビューがクエリー・リライトによって使用される場合、そのマテリアライズド・ビューは、基になるディテール表と同じデータベースに格納する必要があります。マテリアライズド・ビューはパーティション化することも、パーティション表にマテリアライズド・ビューを定義することも可能です。また、マテリアライズド・ビューに1つ以上の索引を定義することもできます。

索引とは異なり、SELECT文を使用してマテリアライズド・ビューに直接アクセスできます。ただし、アプリケーションに影響を与えずにSQL文を変更することが難しくなるので、直接的にマテリアライズド・ビューを参照するSQL文は使用しないようにすることをお薦めします。かわりに、問合せがマテリアライズド・ビューを使用するように、クエリー・リライトで透過的にリライトすることをお薦めします。

この章で説明しているのは、データ・ウェアハウスでマテリアライズド・ビューを使用する方法です。マテリアライズド・ビューは、Oracle Replicationでも使用できます。

5.1.5 サマリー管理のコンポーネント

サマリー管理は、次のもので構成されます。

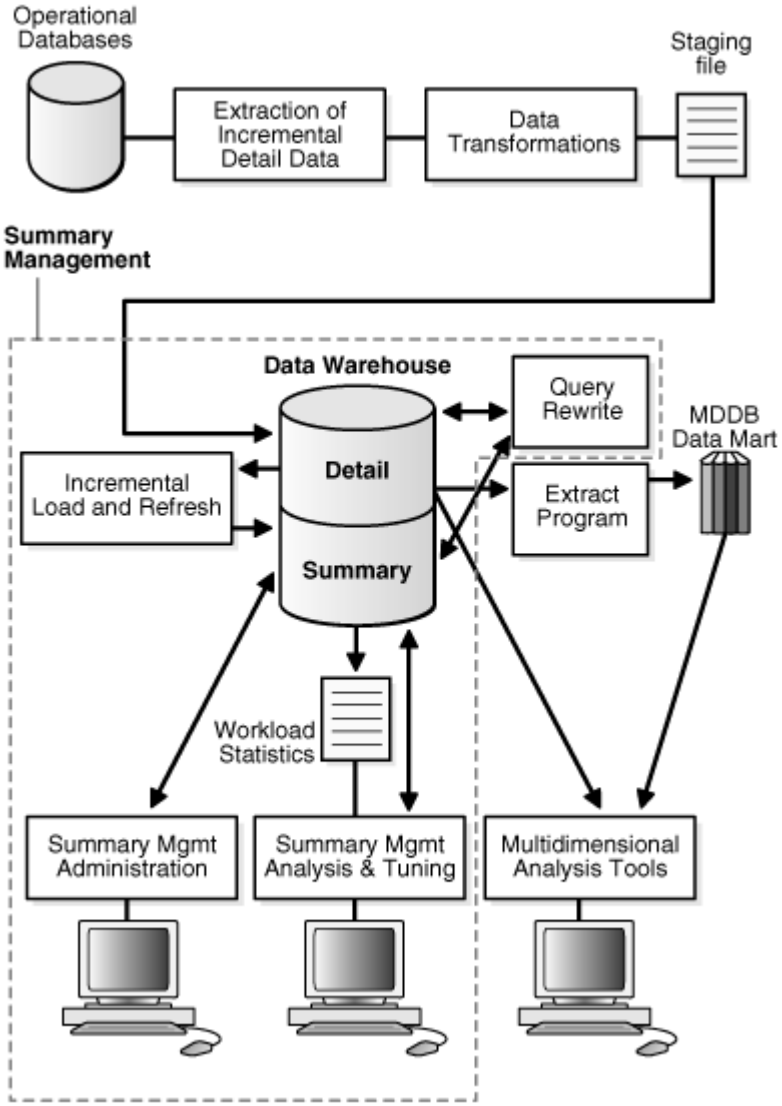
- マテリアライズド・ビューおよびディメンションの定義機能。
- すべてのマテリアライズド・ビューに最新データが確実に含まれるようにするリフレッシュ機能。
- マテリアライズド・ビューを使用するために問合せを透過的にリライトするクエリー・リライト機能。

- 作成するべきマテリアライズド・ビュー、パーティションおよび索引を推奨するSQLアクセス・アドバイザ。
- マテリアライズド・ビューを高速リフレッシュ可能にする方法、および一般的なクエリー・リライトの使用方法を提示するTUNE_MVIEWパッケージ。

サマリー管理機能を使用する際に、スキーマに関する制限事項はありません。また、この機能を使用すると、データベースまたはアプリケーションを再設計しなくても、いくつかの既存DSSデータベース・アプリケーションのパフォーマンスを改善できます。

図5-2に、ウェアハウス・サイクルでのサマリー管理の使用例を示します。データがウェアハウスのディテール・データに変換、ステージングおよびロードされると、サマリー管理プロセスを起動できます。まず、SQLアクセス・アドバイザを使用して、マテリアライズド・ビューをどのように使用するかを計画します。次に、マテリアライズド・ビューを作成し、クエリー・リライトの方法を設計します。マテリアライズド・ビューの動作に問題がある場合は、TUNE_MVIEWを使用すると、マテリアライズド・ビューを最適化できます。

図5-2 サマリー管理の概要



データ・ウェアハウス設計の初期の段階でサマリー管理プロセスを理解しておくと、後で、パフォーマンスの向上、サマリー管理コストの削減および必要な記憶域の削減という大きなメリットを得ることができます。

5.1.6 データ・ウェアハウスの用語

データ・ウェアハウスの基本的な用語の定義は、次のとおりです。

- **ディメンション表**とは、企業のビジネス・エンティティを表します。通常、時間、部門、場所、製品などの階層およびカテゴリ情報として表されます。ディメンション表は、参照表とも呼ばれます。

  ディメンション表は、通常、時間をかけてゆっくり変化し、定期的に変更されることはありません。長時間実行される意

思決定支援問合せで、問合せから戻されるデータをディメンション階層の該当レベルに集計するために使用されます。

- **階層**には、データベースでのビジネスの関係と共通のアクセス・パターンが記述されます。典型的な作業負荷を理解し、ディメンションの分析をすることで、マテリアライズド・ビューを作成できます。詳細は、ディメンションを参照してください。
- **ファクト表**とは、企業のビジネス・トランザクションを表します。

  データ・ウェアハウスのほとんどのデータは、少数の非常に大きなファクト表に格納されます。これらの表は、1つ以上の業務系OLTPデータベースからのデータで定期的に更新されます。

  ファクト表には、売上、個数、在庫などのファクト(メジャーとも呼ぶ)が含まれます。

  - 単純メジャーは、1つの表の数値または文字の列(fact.salesなど)です。
  - 計算済メジャーは、1つの表のメジャーを含む式(fact.revenues - fact.expensesなど)です。
  - マルチ表メジャーは、複数の表で定義される計算済メジャー(fact_a.revenues - fact_b.expensesなど)です。

  ファクト表には、時間、製品、市場など、関連するビジネス・エンティティごとにビジネス・トランザクションを編成する1つ以上の外部キーが含まれます。ほとんどの場合、これらの外部キーはNULLでなく、ファクト表の複合一意キーを形成します。外部キーはそれぞれ ディメンション表の1つの行のみと結合します。
- マテリアライズド・ビューは事前に計算された表で、ファクト表および場合によってはディメンション表からの集計データおよび結合データで構成されます。

5.1.7 マテリアライズド・ビューのスキーマ・デザインについて

データ・ウェアハウス・デザインがこれらのガイドラインに従わない場合でも、サマリー管理によって、クエリー・リライトおよびマテリアライズド・ビューのリフレッシュを含む多くの有効な機能が実行できます。ただし、スキーマ・デザインがこれらのガイドラインに従う場合は、問合せ実行パフォーマンスおよびマテリアライズド・ビューのリフレッシュ・パフォーマンスが大幅に向上し、必要なマテリアライズド・ビューの数を削減できます。

マテリアライズド・ビューの定義には、任意の数の集計および結合を含めることができます。いくつかの点で、マテリアライズド・ビューは索引と同じように動作します。

- マテリアライズド・ビューの目的は、問合せ実行パフォーマンスを向上させることです。
- マテリアライズド・ビューの存在は、SQLアプリケーションに対して透過的であるため、データベース管理者は、SQLアプリケーションの妥当性に影響を与えることなく、いつでもマテリアライズド・ビューを作成または削除できます。
- マテリアライズド・ビューは、記憶領域を消費します。
- マテリアライズド・ビューの内容は、ベースとなるディテール表が変更された場合に、更新される必要があります。

この項では、次の項目について説明します。

- スキーマとディメンション表
- マテリアライズド・ビューのスキーマ・デザインのガイドライン

5.1.7.1 スキーマとディメンション表

正規化または部分的に正規化されているディメンション表(複数の表に格納されているディメンション)の場合は、これらの表がどのように結合されているかを識別します。ディメンション表同士の結合においては、親表の各行と子表の各行の間に1対多の関係が保証されているかどうかに注意してください。また非正規化ディメンションの場合、子の列は親(または属性)の列を一意に決定できなくてはなりません。これらの制約で表されている関係が他の方法で保証されている場合、NOVALIDATEオプションおよびRELYオプションで使用可能にできます。ファクト表とディメンション表の間の結合がこの親子関係をサポートしない場合でも、

CREATE DIMENSION文でディメンションを定義することで、パフォーマンスが大幅に向上します。制限を施行する別の方法には、マテリアライズド・ビューの定義(CREATE MATERIALIZED VIEW文)で外部結合を使用することがあります。

これらのリレーションシップを満たさないスキーマでは、ディメンションを作成しないでください。作成すると、問合せで不適切な結果が戻される場合があります。

5.1.7.2 マテリアライズド・ビューのスキーマ・デザインのガイドライン

サマリー管理の様々なコンポーネントを定義して使用する前に、スキーマ・デザインを調べて、できるだけ次のガイドラインに従う必要があります。ガイドライン1および2は、ガイドライン3より重要です。スキーマ・デザインがガイドライン1および2に従っていない場合、ガイドライン3に従っているかどうかは問題ではありません。ガイドライン1、2および3は、クエリー・リライトのパフォーマンスおよびマテリアライズド・ビューのリフレッシュ・パフォーマンスの両方に影響します。

ディメンションのガイドライン1

(各ディメンションが1つの表に収まるように)ディメンションが非正規化されるか、正規化または部分的に正規化されたディメンションの表間の結合において、親表の各行と子表の各行の間に1対多の関係が保証される必要があります。

子の結合キーに外部キー制約およびNOT NULL制約を追加し、親の結合キーに主キー制約を追加すると、この条件を規定できます。

ディメンションのガイドライン2

ディメンションが非正規化または部分的に非正規化されている場合、ディメンション表のキー列間で階層整合性を保つ必要があります。それぞれの子キー値は、ディメンション表が非正規化されていても、その親キー値を一意に識別する必要があります。非正規化ディメンションの階層整合性は、DBMS_DIMENSIONパッケージのVALIDATE_DIMENSIONプロシージャをコールすることで検証できます。

ディメンションのガイドライン3

ファクト表およびディメンション表では、同様に、各ファクト表の行がディメンション表の1つの行のみと結合することを保証する必要があります。この条件を宣言し、オプションで規定する必要があります。それには、ファクト表のキー列に外部キー制約およびNOT NULL制約を追加し、ディメンション表のキー列に主キー制約を追加するか、外部結合を使用します。データ・ウェアハウスでは、制約規定によるパフォーマンス・オーバーヘッドを回避するために、通常、NOVALIDATE句およびRELY句を使用して制約を使用可能にします。

ディメンションのガイドライン4

各ロード後、マテリアライズド・ビューをリフレッシュする前に、DBMS_DIMENSIONパッケージのVALIDATE_DIMENSIONプロシージャを使用してディメンションの整合性を増分検証します。

増分ロードのガイドライン

ディテール・データの増分ロードは、SQL*Loaderダイレクト・パス・オプション、またはOracleのダイレクト・パス・インタフェースを使用するバルク・ロード・ユーティリティを使用して実行する必要があります。これには、INSERT ... AS SELECT(APPENDまたはPARALLELヒントを利用)などがあり、このヒントによりダイレクト・ローダーのログが挿入時に使用されます。

パーティションのガイドライン

可能な場合は、単調に増加する(できればDATE型の)時間列によって、表のレンジ・パーティション化/コンポジット・パーティション化を行います。

時間ディメンションのガイドライン

時間ディメンションがマテリアライズド・ビューに時間列として表示される場合は、ファクト表の場合と同じ方法でマテリアライズド・ビューをパーティション化して索引を付けます。

制約を使用可能にするために必要な時間、および制約に違反する場合を考慮する必要がある場合は、ENABLE NOVALIDATE文でRELY句を使用して、既存の制約の妥当性チェックを行わずに、制約チェックをONにします。この方法には、制約が1つでも損われた場合、不正確な問合せ結果が戻される可能性があるというデメリットがあります。そのため、データがどれだけ正確か、また、不正確な結果が戻される可能性が大きすぎないかどうかを設計時に判断する必要があります。

関連項目:

- マテリアライズド・ビューのタイプ
- 親表と子表の各行の結合をメンテナンスするメリットの詳細は、ディメンションの作成を参照してください。
- Oracle Database SQL言語リファレンス

5.1.8 データ・ウェアハウスへのデータのロードについて

データ・ウェアハウスまたはデータ・マートにデータをロードする一般的で効率的な方法には、SQL*LoaderでDIRECTまたはPARALLELオプションを使用する方法、データ・ポンプを使用する方法、またはOracleのダイレクト・パスAPIを使用する別のローダー・ツールを使用する方法があります。

ロード方法は、1フェーズまたは2フェーズに分類できます。1フェーズ・ロードでは、データはターゲット表に直接ロードされ、品質保証テストが実行されます。エラーは、マテリアライズド・ビューをリフレッシュする前にDML操作を実行することによって解決されます。多くの削除が行われる場合、ディスク使用率に悪影響を及ぼすことがありますが、一時領域要件およびロード時間が最小化されます。

2フェーズ・ロード・プロセスでは、次の処理が行われます。

- データがウェアハウスの一時表にロードされます。
- 品質保証プロシージャがデータに適用されます。
- ターゲット表に対する参照整合性制約が使用禁止となり、ターゲット・パーティションのローカル索引がUNUSABLEとマークされます。
- INSERT AS SELECTを使用し、PARALLELまたはAPPENDヒントを指定することによって、データが一時領域からターゲット表の適切なパーティションにコピーされます。その後、一時表が削除されます。あるいは、ターゲット表がパーティション化されている場合は、ターゲット表に新しい(空の)パーティションを作成し、ALTER TABLE ... EXCHANGE PARTITIONを使用して一時表をターゲット表に取り込むこともできます。詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- 通常、NOVALIDATEオプション付きで制約が使用可能になります。

ディテール・データのロードおよびディテール・データ上の索引の更新を行うと、必要に応じて、データベースに対する操作ができるようになります。ALTER SYSTEM SET QUERY_REWRITE_ENABLED = FALSE文を発行することにより、すべてのマテリアライズド・ビューがリフレッシュされるまで、クエリー・リライトをシステム・レベルで使用禁止にできます。

QUERY_REWRITE_INTEGRITYをSTALE_TOLERATEDと設定した場合、ALTER SESSION SET QUERY_REWRITE_ENABLED = TRUE文を発行すると、最新ロードのデータが反映されたマテリアライズド・ビューを必要としないユーザーに対して、マテリアライズド・ビューへのアクセスをセッション・レベルで許可できます。この使用例は、QUERY_REWRITE_INTEGRITYがENFORCEDまたはTRUSTEDの場合には適用されません。これは、この2つのモードでは、更新されたデータを持つマテリアライズド・ビューのみがクエリー・リライトで使用されるためです。

関連項目:

DIRECTまたはPARALLELキーワードを指定してSQL*Loaderを使用する場合の制限および考慮点については、*Oracle Databaseユーティリティ*を参照してください。

5.1.9 マテリアライズド・ビューの管理作業の概要

マテリアライズド・ビューの使用目的はパフォーマンスの向上ですが、マテリアライズド・ビュー管理によるオーバーヘッドが、システム管理上の大きな問題になる場合があります。マテリアライズド・ビューに必要な管理アクティビティをレビューまたは評価するときは、次のことを考慮します。

- 最初に作成するマテリアライズド・ビューの特定
- マテリアライズド・ビューの索引付け
- データベース更新のたびに、すべてのマテリアライズド・ビューおよびその索引が適切にリフレッシュされたかどうかの確認
- 使用されたマテリアライズド・ビューのチェック
- 作業負荷パフォーマンスに対する各マテリアライズド・ビューの効率の判断
- マテリアライズド・ビューが使用した領域の計算
- 作成が必要な新しいマテリアライズド・ビューの判断
- 削除が必要な既存のマテリアライズド・ビューの判断
- 有効ではなくなった古いディテールおよびマテリアライズド・ビュー・データのアーカイブ

データ・ウェアハウスまたはデータ・マートを最初に作成および移入した後の主な管理オーバーヘッドは更新処理です。これには次が含まれます。

- 業務系システムによる増分変更の定期的な抽出
- データの変換
- 増分変更が正しく、一貫性があり、完全であるかどうかの検証
- ウェアハウスへのデータのバルク・ロード
- ディテール・データに対する一貫性を保つための索引とマテリアライズド・ビューのリフレッシュ

通常、更新処理は、更新ウィンドウと呼ばれる制限時間内に実行される必要があります。更新ウィンドウは、更新頻度(毎日、毎週など)およびビジネスの特性によって異なります。更新頻度が毎日の場合、更新ウィンドウは2から6時間になります。

次のアクティビティの更新ウィンドウを知る必要があります。

- ディテール・データのロード
- ディテール・データに対する索引の更新または再作成
- データに対する品質保証テストの実行
- マテリアライズド・ビューのリフレッシュ
- マテリアライズド・ビューに対する索引の更新

5.2 マテリアライズド・ビューのタイプ

マテリアライズド・ビュー作成文のSELECT句で、マテリアライズド・ビューに含めるデータが定義されます。指定できる内容に関する制限はごく少数です。必要な数の表を結合できます。表のみでなく、ビュー、インライン・ビュー(SELECT文のFROM句の副問合せ)、副問合せおよびマテリアライズド・ビューなど、他の要素もすべてSELECT句で結合または参照できます。ただし、定義問合せのSELECT構文のリストに副問合せを持つマテリアライズド・ビューは定義できません。WHERE句など、定義問合せのその他の場所に副問合せを記述することは可能です。

マテリアライズド・ビューには、次のタイプがあります。

- 集計を含むマテリアライズド・ビューについて
- 結合のみを含むマテリアライズド・ビューについて
- ネステッド・マテリアライズド・ビューについて

5.2.1 集計を含むマテリアライズド・ビューについて

データ・ウェアハウスでは、通常、マテリアライズド・ビューには例5-1のような集計が含まれています。高速リフレッシュを可能にするには、SELECT構文のリストにすべてのGROUP BY列(ある場合)を含める必要があります。また、集計列にCOUNT(*)およびCOUNT(column)が必要です。さらに、マテリアライズド・ビュー・ログは、マテリアライズド・ビューを定義する問合せで参照されるすべての表に関して存在する必要があります。有効な集計関数は、SUM、COUNT(x)、COUNT(*)、AVG、VARIANCE、STDDEV、MIN、MAXです。また、任意のSQL値式を集計できます。集計を含むマテリアライズド・ビューの高速リフレッシュに関する制限を参照してください。

関連項目:

集計を含むマテリアライズド・ビューの使用要件

結合と集計を含むマテリアライズド・ビューの高速リフレッシュは、実表に対してDML(ダイレクト・ロードまたは従来型のINSERT、UPDATEまたはDELETE)を実行した後に可能になります。ON COMMITまたはON DEMANDでリフレッシュされるように定義できます。REFRESH ON COMMITを指定した場合は、マテリアライズド・ビューのディテール表の1つに対してDMLを実行するトランザクションがコミットされると、マテリアライズド・ビューが自動的にリフレッシュされます。この方法を選択すると、コミット完了までの所要時間は通常より少し長くなることがあります。これは、リフレッシュ操作がコミット・プロセスの一部として実行されるためです。したがって、この方法は、多数のユーザーがマテリアライズド・ビューの実表を同時に変更する場合には適していません。

次に、集計を含むマテリアライズド・ビューの例をいくつか示します。この例のマテリアライズド・ビューは高速リフレッシュされるため、マテリアライズド・ビュー・ログの作成は必須であることに注意してください。

例5-1 **マテリアライズド・ビューの作成**(合計売上数と合計売上金額)

```
CREATE MATERIALIZED VIEW LOG ON products WITH SEQUENCE, ROWID
(prod_id, prod_name, prod_desc, prod_subcategory, prod_subcategory_desc,
prod_category, prod_category_desc, prod_weight_class, prod_unit_of_measure,
 prod_pack_size, supplier_id, prod_status, prod_list_price, prod_min_price)
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON sales
WITH SEQUENCE, ROWID
(prod_id, cust_id, time_id, channel_id, promo_id, quantity_sold, amount_sold)
INCLUDING NEW VALUES;
```

```
CREATE MATERIALIZED VIEW product_sales_mv
PCTFREE 0  TABLESPACE demo
STORAGE (INITIAL 8M)
BUILD IMMEDIATE
REFRESH FAST
ENABLE QUERY REWRITE
AS SELECT p.prod_name, SUM(s.amount_sold) AS dollar_sales,
COUNT(*) AS cnt, COUNT(s.amount_sold) AS cnt_amt
FROM sales s, products p
WHERE s.prod_id = p.prod_id GROUP BY p.prod_name;
```

この例では、製品の合計売上数と合計売上金額を計算するマテリアライズド・ビューproduct_sales_mvが作成されます。これは、prod_id列で表salesおよびproductsを結合することで導出されます。このマテリアライズド・ビューは、作成方法がIMMEDIATEであるため、データがすぐに移入され、クエリー・リライトに使用できます。この例では、デフォルトのリフレッシュ方法はFASTです。これが許されるのは、表productsおよびsalesに関して適切なマテリアライズド・ビュー・ログが作成されているためです。

WITH COMMIT SCN句を含むマテリアライズド・ビュー・ログを使用すると、ローカル・マテリアライズド・ビューの高速リフレッシュのパフォーマンスが向上します。次に例を示します。

```
CREATE MATERIALIZED VIEW LOG ON sales WITH ROWID(prod_id, cust_id, time_id),
  COMMIT SCN INCLUDING NEW VALUES;
```

例5-2 **マテリアライズド・ビューの作成** (計算された売上の合計)

```
CREATE MATERIALIZED VIEW product_sales_mv
PCTFREE 0 TABLESPACE demo
STORAGE (INITIAL 8M)
BUILD DEFERRED
REFRESH COMPLETE ON DEMAND
ENABLE QUERY REWRITE AS
SELECT p.prod_name, SUM(s.amount_sold) AS dollar_sales
FROM sales s, products p WHERE s.prod_id = p.prod_id
GROUP BY p.prod_name;
```

この例では、prod_nameごとの合計売上を計算するマテリアライズド・ビューproduct_sales_mvが作成されます。これは、prod_id列で表salesおよびproductsを結合することで導出されます。このマテリアライズド・ビューは、作成方法がDEFERREDであるため、最初は、どのデータも含みません。作成方法がDEFERREDのマテリアライズド・ビューの最初のリフレッシュには、完全リフレッシュが必要です。このマテリアライズド・ビューがリフレッシュされ、移入されると、クエリー・リライトに使用できます。

例5-3 **マテリアライズド・ビューの作成**(単一表集計)

```
CREATE MATERIALIZED VIEW LOG ON sales WITH SEQUENCE, ROWID
(prod_id, cust_id, time_id, channel_id, promo_id, quantity_sold, amount_sold)
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW sum_sales
PARALLEL
BUILD IMMEDIATE
REFRESH FAST ON COMMIT AS
SELECT s.prod_id, s.time_id, COUNT(*) AS count_grp,
   SUM(s.amount_sold) AS sum_dollar_sales,
   COUNT(s.amount_sold) AS count_dollar_sales,
   SUM(s.quantity_sold) AS sum_quantity_sales,
   COUNT(s.quantity_sold) AS count_quantity_sales
FROM sales s
GROUP BY s.prod_id, s.time_id;
```

この例では、単一表集計を含むマテリアライズド・ビューが作成されます。マテリアライズド・ビューの定義問合せの中ですべての参照列にマテリアライズド・ビュー・ログが作成されているので、このマテリアライズド・ビューは高速リフレッシュが可能です。DMLがsales表に対して適用される場合、コミットが発行されたときに変更がマテリアライズド・ビューに反映されます。

関連項目:

CREATE MATERIALIZED VIEW文およびCREATE MATERIALIZED VIEW LOG文の構文は、Oracle Database SQL言語リファレンスを参照してください。

5.2.1.1 集計を含むマテリアライズド・ビューの使用要件

表5-1に、マテリアライズド・ビューの集計要件を示します。集計Xがある場合、集計Yが必要であり、集計Zはオプションです。

表5-1 集計を含むマテリアライズド・ビューの要件

X	Y	Z
COUNT(expr)	–	–
MIN(expr)		
MAX(expr)		
SUM(expr)	COUNT(expr)	–
SUM(col)(col には NOT NULL 制約あり)	-	
AVG(expr)	COUNT(expr)	SUM(expr)
STDDEV(expr)	COUNT(expr) SUM(expr)	SUM(expr * expr)
VARIANCE(expr)	COUNT(expr) SUM(expr)	SUM(expr * expr)

すべてのタイプの高速リフレッシュを保証するには、常にCOUNT(*)が必要です。さもないと、挿入後の高速リフレッシュのみに制限される場合があります。集計を最も効率よく正確に高速リフレッシュできるように、マテリアライズド・ビューにZ列のオプションの集計も含めることをお薦めします。

5.2.2 結合のみを含むマテリアライズド・ビューについて

マテリアライズド結合ビューのFROM句に関する考慮事項のように、マテリアライズド・ビューに結合のみが含まれ、集計は含まれない場合があります。この例では、sales表をtimes表とcustomers表に結合するマテリアライズド・ビューが作成されます。このタイプのマテリアライズド・ビューを作成するメリットは、コストの高い結合が事前に計算されることです。

関連項目:

テリアライズド結合ビューのFROM句に関する考慮事項

結合のみを含むマテリアライズド・ビューの高速リフレッシュは、実表に対してDML(ダイレクト・パスまたは従来型のINSERT、UPDATEまたはDELETE)を実行した後に可能になります。

結合のみを含むマテリアライズド・ビューは、ON COMMITまたはON DEMANDでリフレッシュされるように定義できます。ON COMMITの場合、リフレッシュは、マテリアライズド・ビューにある1つのディテール表上でDMLを実行するトランザクションのコミット時に実行されます。

REFRESH FASTを指定する場合、Oracle Databaseは、問合せ定義をさらに検証して、いずれかのディテール表が変更された場合の高速リフレッシュの実行を保証します。これらの追加チェックには、次の制限が含まれます。

- 表がパーティション・チェンジ・トラッキング(PCT)をサポートしないかぎり、マテリアライズド・ビュー・ログがディテール表ごとに存在する必要があります。また、マテリアライズド・ビュー・ログが必須の場合は、ROWID列が各マテリアライズド・ビュー・ログに存在していること。
- すべてのディテール表のROWIDが、マテリアライズド・ビュー問合せ定義のSELECT構文のリストにあること。

これらの制限で満たされないものがある場合は、マテリアライズド・ビューをREFRESH FORCEとして作成し、可能なときに高速リフレッシュの効果を得ることができます。表の1つがすべての基準を満たさなくても、他の表がすべての基準を満たしている場合は、すべての基準が満たされている他の表に関しては、マテリアライズド・ビューを高速リフレッシュできます。

最も効率的にリフレッシュが実行されるようにするには、内部結合のように動作する外部結合を定義問合せで使用しないようにします。このような結合が定義問合せに含まれている場合は、内部結合を使用するように定義問合せをリライトすることを検討してください。

関連項目:

- リフレッシュ・パフォーマンスを低下させる条件の詳細は、結合のみを含むマテリアライズド・ビューの高速リフレッシュに関する制限を参照してください。
- マテリアライズド・ビューのパーティション・チェンジ・トラッキング(PCT)リフレッシュについて

5.2.2.1 マテリアライズド結合ビューのFROM句に関する考慮事項

マテリアライズド・ビューに含まれているのが結合のみである場合は、各表のROWID列(およびFROM句のリストに複数回指定されている表)を、マテリアライズド・ビューのSELECT構文のリストに指定する必要があります。

マテリアライズド・ビューのFROM句にリモート表が指定されている場合、マテリアライズド・ビューの増分(高速)リフレッシュを実行するには、そのFROM句の表はすべて同じサイトに配置されている必要があります。また、リモート表を含むマテリアライズド・ビューでは、ON COMMITによるリフレッシュはサポートされません。SCNベースのマテリアライズド・ビュー・ログを除き、マテリアライズド・ビューの各ディテール表のマテリアライズド・ビュー・ログはリモート・サイトに作成されている必要があり、ROWID列は、次の例に示すようにマテリアライズド・ビューのSELECT構文のリストに指定されている必要があります。

例5-4 結合のみを含むマテリアライズド・ビュー

```
CREATE MATERIALIZED VIEW LOG ON sales WITH ROWID;
CREATE MATERIALIZED VIEW LOG ON times WITH ROWID;
CREATE MATERIALIZED VIEW LOG ON customers WITH ROWID;
CREATE MATERIALIZED VIEW detail_sales_mv
PARALLEL BUILD IMMEDIATE
REFRESH FAST AS
SELECT s.rowid "sales_rid", t.rowid "times_rid", c.rowid "customers_rid",
       c.cust_id, c.cust_last_name, s.amount_sold, s.quantity_sold, s.time_id
```

```
FROM sales s, times t, customers c
WHERE  s.cust_id = c.cust_id(+) AND s.time_id = t.time_id(+);
```

また、前述の例にtimes_rid列およびcustomers_rid列が含まれず、リフレッシュ方法がREFRESH FORCEであった場合、このマテリアライズド・ビューを高速リフレッシュできるのは、sales表が更新された場合のみです。times表またはcustomers表が更新された場合は、高速リフレッシュできません。

```
CREATE MATERIALIZED VIEW detail_sales_mv
PARALLEL
BUILD IMMEDIATE
REFRESH FORCE AS
SELECT s.rowid "sales_rid", c.cust_id, c.cust_last_name, s.amount_sold,
   s.quantity_sold, s.time_id
FROM sales s, times t, customers c
WHERE s.cust_id = c.cust_id(+) AND s.time_id = t.time_id(+);
```

5.2.3 ネステッド・マテリアライズド・ビューについて

ネステッド・マテリアライズド・ビューとは、その定義が別のマテリアライズド・ビューに基づいているマテリアライズド・ビューです。ネステッド・マテリアライズド・ビューは、マテリアライズド・ビューの他に、データベース内の他のリレーションも参照する場合があります。

この項では、次の項目について説明します。

- ネステッド・マテリアライズド・ビューを使用する理由
- 結合および集計を含むマテリアライズド・ビューのネストについて
- ネステッド・マテリアライズド・ビューの使用上のガイドライン
- ネステッド・マテリアライズド・ビューの使用上の制限

5.2.3.1 ネステッド・マテリアライズド・ビューを使用する理由

データ・ウェアハウスでは、通常、単一の結合上に多数の集計ビュー(たとえば、異なるディメンションに沿ったロールアップ)を作成します。これらの個別の結合と集計を含むマテリアライズド・ビューに対する増分メンテナンスは、ベースとなる結合が何度も実行される必要があるため、かなりの時間がかかります。

ネステッド・マテリアライズド・ビューを使用すると、結合のみを含む1つのマテリアライズド・ビューに基づいて複数の単一表マテリアライズド・ビューを作成できます。さらに、この種類の単一表集計マテリアライズド・ビューには最適化が実行され、リフレッシュが非常に効率的になります。

例5-5 ネステッド・マテリアライズド・ビュー

マテリアライズド・ビューに対してネステッド・マテリアライズド・ビューを作成できますが、親およびベースのマテリアライズド・ビューには結合または集計が含まれている必要があります。マテリアライズド・ビューの定義問合せに結合または集計が含まれない場合はネストできません。マテリアライズド・ビューの定義のベースとなるすべてのオブジェクト(マテリアライズド・ビューまたは表)には、マテリアライズド・ビュー・ログが必要です。ベースとなるオブジェクトは、すべて表と同様に扱われます。また、マテリアライズド・ビュー用のオプションはすべて使用できます。

次のマテリアライズド・ビューは、shサンプル・スキーマの表と列を使用するネステッド・マテリアライズド・ビューの作成方法を示しています。

```
CREATE MATERIALIZED VIEW LOG ON sales WITH ROWID;
CREATE MATERIALIZED VIEW LOG ON customers WITH ROWID;
CREATE MATERIALIZED VIEW LOG ON times WITH ROWID;

/*create materialized view join_sales_cust_time as fast refreshable at
```

```
   COMMIT time */
CREATE MATERIALIZED VIEW join_sales_cust_time
REFRESH FAST ON COMMIT AS
SELECT c.cust_id, c.cust_last_name, s.amount_sold, t.time_id,
       t.day_number_in_week, s.rowid srid, t.rowid trid, c.rowid crid
FROM sales s, customers c, times t
WHERE s.time_id = t.time_id AND s.cust_id = c.cust_id;
```

join_sales_cust_time表に対するネステッド・マテリアライズド・ビューを作成するには、その表に対してマテリアライズド・ビュー・ログを作成する必要があります。これはjoin_sales_cust_time表に対する単一表集計マテリアライズド・ビューになるため、必要なすべての列をログに記録して、INCLUDING NEW VALUES句を使用する必要があります。

```
/* create materialized view log on join_sales_cust_time */
CREATE MATERIALIZED VIEW LOG ON join_sales_cust_time
WITH ROWID (cust_last_name, day_number_in_week, amount_sold)
INCLUDING NEW VALUES;

/* create the single-table aggregate materialized view sum_sales_cust_time
on join_sales_cust_time as fast refreshable at COMMIT time */
CREATE MATERIALIZED VIEW sum_sales_cust_time
REFRESH FAST ON COMMIT AS
SELECT COUNT(*) cnt_all, SUM(amount_sold) sum_sales, COUNT(amount_sold)
       cnt_sales, cust_last_name, day_number_in_week
FROM join_sales_cust_time
GROUP BY cust_last_name, day_number_in_week;
```

5.2.3.2 結合および集計を含むマテリアライズド・ビューのネストについて

ネステッド・マテリアライズド・ビューの中には、高速リフレッシュできないタイプがあります。このようなタイプのマテリアライズド・ビューを特定するには、EXPLAIN_MVIEWを使用します。DBMS_MVIEW.REFRESHパラメータとともにnested = TRUEパラメータを指定すると、適切な依存順序でネステッド・マテリアライズド・ビューのツリーをリフレッシュできます。たとえば、DBMS_MVIEW.REFRESH ('SUM_SALES_CUST_TIME', nested => TRUE)とコールすると、REFRESHプロシージャによって、まずjoin_sales_cust_timeマテリアライズド・ビューがリフレッシュされ、次にsum_sales_cust_timeマテリアライズド・ビューがリフレッシュされます。

5.2.3.3 ネステッド・マテリアライズド・ビューの使用上のガイドライン

ネステッド・マテリアライズド・ビューを使用するかどうかを決定する場合、次の点に注意する必要があります。

- 高速リフレッシュが必要な場合は、依存しているすべてのマテリアライズド・ビューも高速リフレッシュする必要があります。
- 最高レベルのマテリアライズド・ビューをディテール表と同じ更新レベルに保つには、最高レベルのマテリアライズド・ビューをリフレッシュする前に、ツリー内のすべてのマテリアライズド・ビューが正しい依存順序でリフレッシュされるようにする必要があります。結合および集計を含むマテリアライズド・ビューのネストについてで説明しているように、nested = TRUEパラメータを使用すると、ネスト階層内の中間マテリアライズド・ビューを自動的にリフレッシュできます。nested = TRUEを指定せず、最高レベルのマテリアライズド・ビューの下にあるマテリアライズド・ビューが失効している場合、最高レベルのみのリフレッシュは正常に終了しますが、これはベースとなるマテリアライズド・ビューの観点から更新されているのみで、ツリーの基礎であるディテール表の観点から更新されているわけではありません。
- マテリアライズド・ビューをリフレッシュするときは、ツリー内のすべてのマテリアライズド・ビューがリフレッシュされるようにする必要があります。最高レベルのマテリアライズド・ビューのみをリフレッシュした場合、その下にあるマテリアライズド・ビューは失効するため、これらは明示的にリフレッシュする必要があります。nestedパラメータの値をTRUEに設定してREFRESHプロシージャを実行した場合、指定したマテリアライズド・ビューと、ツリー内の子マテリアライズド・ビューのみがリフレッシュされます。そのツリー内の最高レベルのマテリアライズド・ビューはリフレッシュされません。ツリー内のすべてのマテリアライズ

ド・ビューを確実にリフレッシュする必要がある場合は、nestedパラメータの値をTRUEに設定してREFRESH_DEPENDENTプロシージャを実行します。

- 特定のネステッド・マテリアライズド・ビューでサポートされている唯一のオプションが完全リフレッシュである場合、高速リフレッシュが指定されても、完全リフレッシュが実行されます。
- マテリアライズド・ビューが最新であるかどうかは、そのマテリアライズド・ビューが直接的に参照しているオブジェクトとの比較によって決められます。マテリアライズド・ビューが別のマテリアライズド・ビューを参照している場合、最高レベルのマテリアライズド・ビューが最新であるかどうかは、そのマテリアライズド・ビューが直接的に参照しているマテリアライズド・ビューでの変更と比較することによって決められます。

5.2.3.4 ネステッド・マテリアライズド・ビューの使用上の制限

マテリアライズド・ビューと事前作成マテリアライズド・ビューの両方は同じ表に作成できません。たとえば、表costsとこの表に基づくマテリアライズド・ビューcost_mvがある場合、表costsに対して事前作成マテリアライズド・ビューは作成できません。作成すると、cost_mvがネステッド・マテリアライズド・ビューになります。このような変換方法はサポートされていません。

5.3 マテリアライズド・ビューの作成

マテリアライズド・ビューは、CREATE MATERIALIZED VIEW文またはEnterprise Managerを使用して作成できます。

一般的に、データ・ウェアハウスにはすでにサマリー表または集計表が作成されており、新しいマテリアライズド・ビューを作成して、この作業を繰り返すことはありません。この場合、すでにデータベースに存在する表は、事前作成マテリアライズド・ビューとして登録できます。この方法については、既存のマテリアライズド・ビューの登録を参照してください。

作成するマテリアライズド・ビューを選択した後、各マテリアライズド・ビューに対して次のステップを実行します。

1. マテリアライズド・ビューを設計します。既存のユーザー定義のマテリアライズド・ビューについてはこのステップは不要です。

   マテリアライズド・ビューに多数の行が含まれる場合、適切であれば、マテリアライズド・ビューをパーティション化します(可能な場合)。このパーティション化は、最大または最も頻繁に更新されるディテール表またはファクト表のパーティション化と一致させる必要があります(可能な場合)。パーティション化により、リフレッシュのパフォーマンスが向上します。これは、パラレルDML機能と、場合によってはPCTベースのリフレッシュを利用できるようになるためです。

2. CREATE MATERIALIZED VIEW文を使用して、マテリアライズド・ビューを作成および移入(オプション)します。

   すでにユーザー定義マテリアライズド・ビューが存在する場合は、CREATE MATERIALIZED VIEW文のON PREBUILT TABLE句を使用します。それ以外の場合は、BUILD IMMEDIATE句を使用してマテリアライズド・ビューにすぐに移入するか、BUILD DEFERRED句を使用して後から移入します。BUILD DEFERREDで作成したマテリアライズド・ビューは、最初のCOMPLETE REFRESHが行われるまで、クエリー・リライトで使用できません。ENABLE QUERY REWRITE句を指定していれば、最初のリフレッシュの後で自動的にクエリー・リライトが使用可能になります。

例5-6 マテリアライズド・ビューの作成

この例では、cust_sales_mvというマテリアライズド・ビューが作成されます。

```
CREATE MATERIALIZED VIEW cust_sales_mv
PCTFREE 0 TABLESPACE demo
STORAGE (INITIAL 8M)
PARALLEL
BUILD IMMEDIATE
REFRESH COMPLETE
ENABLE QUERY REWRITE AS
SELECT  c.cust_last_name, SUM(amount_sold) AS sum_amount_sold
FROM customers c, sales s WHERE s.cust_id = c.cust_id
GROUP BY c.cust_last_name;
```

関連項目:

SQL文CREATE MATERIALIZED VIEW、ALTER MATERIALIZED VIEWおよびDROP MATERIALIZED VIEWの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

5.3.1 列の別名リストを含むマテリアライズド・ビューの作成

現在のバージョンでは、マテリアライズド・ビューの作成時、定義問合せのSELECT構文のリストに同じ名前の列が複数含まれている場合は、それらに一意の別名を付与して名前の競合を解消する必要があります。競合を解消しないと、列の定義が曖昧であるというエラー・メッセージが表示され、CREATE MATERIALIZED VIEW文が失敗します。しかし、名前解決のために通常の方法でSELECT句に別名を指定すると、テキストの完全一致のクエリー・リライトの使用が制限され、マテリアライズド・ビューの定義問合せのテキストとユーザーの入力問合せのテキストが同一である場合にのみ、テキストの完全一致のリライトが行われることになります。したがって、マテリアライズド・ビューの定義問合せのSELECT構文のリストに別名を指定し、問合せに別名が指定されなかった場合、テキストの完全一致の比較は失敗します。これは特に、列の別名を大量に使用するDiscovererからの問合せで問題になります。

次に、問題の例を示します。sales_mvを作成する際、SELECT句に列の別名が指定されていますが、入力問合せQ1には別名が指定されていません。このため、テキストの完全一致のリライトは失敗します。マテリアライズド・ビューは次のとおりです。

```
CREATE MATERIALIZED VIEW sales_mv
ENABLE QUERY REWRITE AS
SELECT s.time_id sales_tid, c.time_id costs_tid
FROM sales s, products p, costs c
WHERE s.prod_id = p.prod_id AND c.prod_id = p.prod_id AND
      p.prod_name IN (SELECT prod_name FROM products);
```

入力問合せ文Q1は次のとおりです。

```
SELECT s.time_id, c1.time_id
FROM sales s, products p, costs c1
WHERE s.prod_id = p.prod_id AND c1.prod_id = p.prod_id AND
      p.prod_name IN (SELECT prod_name FROM products);
```

マテリアライズド・ビューの定義問合せは、ユーザーの入力問合せとほとんど同一であり、論理的には同等ですが、クエリー・リライトは行われません。これは、一部の問合せ(WHERE句の副問合せなど)ではリライトが行われる唯一の可能性であるテキストの完全一致に失敗しているためです。

CREATE MATERIALIZED VIEW文には、列の別名リストを追加できます。列の別名リストを使用すると、マテリアライズド・ビューのSELECT句に別名を指定することなく、列名の競合を明示的に解消できます。次の例に、マテリアライズド・ビューの列の別名リストの構文を示します。

```
CREATE MATERIALIZED VIEW sales_mv (sales_tid, costs_tid)
ENABLE QUERY REWRITE AS
SELECT s.time_id, c.time_id
FROM sales s, products p, costs c
WHERE s.prod_id = p.prod_id AND c.prod_id = p.prod_id AND
      p.prod_name IN (SELECT prod_name FROM products);
```

この例では、sales_mvの定義問合せが、ユーザー問合せQ1と完全に一致しています。このため、テキストの完全一致のリライトが行われます。

別名をSELECT句と新しい別名リスト句の両方に指定した場合は、SELECT句に指定した別名ではなく、別名リスト句が使用さ

れます。

5.3.2 ハイブリッド・パーティション表に基づくマテリアライズド・ビューの作成

CREATE MATERIALIZED VIEW文を使用して、ハイブリッド・パーティション表に基づくマテリアライズド・ビューを作成します。

ハイブリッド・パーティション表では、一部のパーティションはデータベース・セグメントに格納されますが、他のパーティションは外部に格納されます。ハイブリッド・パーティション表に基づくマテリアライズド・ビューのSELECT文にパーティション・キーまたはパーティション・マーカーが含まれる場合、そのマテリアライズド・ビューはPCTリフレッシュの要件を満たします。

ハイブリッド・パーティション表に基づいてマテリアライズド・ビューを作成するには、次のようにします。

1. ハイブリッド・パーティション表を作成します。

   次のコマンドは、ハイブリッド・パーティション表hybrid_salesを作成します。

```
CREATE TABLE hybrid_sales(time_id date, customer number, price number, …)
…
PARTITION BY RANGE (time_id)
(
   PARTITION century_19 VALUES LESS THAN (TO_DATE('01-01-1900', 'DD-MM-YYYY'))
       EXTERNAL LOCATION (data_dir1:'sales_1.csv'),
   PARTITION century_20 VALUES LESS THAN (TO_DATE('01-01-2000', 'DD-MM-YYYY'))
       EXTERNAL DEFAULT DIRECTORY data_dir2 LOCATION ('sales_2.csv'),
   PARTITION year_2000 VALUES LESS THAN (TO_DATE('01-01-2001', 'DD-MM-YYYY')),
   PARTITION year_2001 VALUES LESS THAN (TO_DATE('01-01-2002' , 'DD-MM-YYYY'))
);
```

2. ハイブリッド・パーティション表に基づくマテリアライズド・ビューを作成します。

   次のコマンドは、ハイブリッド・パーティション表hybrid_salesに基づくマテリアライズド・ビューhypt_mvを作成します。

```
CREATE MATERIALIZED VIEW HyPT_MV
REFRESH FAST ON DEMAND AS
SELECT time_id, customer_no, sum(price) as total_price
FROM hybrid_sales
GROUP BY time_id, customer_no;
```

   表hybrid_salesに対応するマテリアライズド・ビュー・ログがあるとします。

5.3.3 マテリアライズド・ビューの名前について

マテリアライズド・ビューの名前は、Oracleの標準のネーミング規則に従っている必要があります。ただし、マテリアライズド・ビューがユーザー定義の事前作成表を基にしている場合は、マテリアライズド・ビューの名前はその表名と一致させる必要があります。

すでに表および索引のネーミング規則がある場合は、このネーミング計画をマテリアライズド・ビューに拡張して、マテリアライズド・ビューを簡単に識別できるようにすることが可能です。たとえば、マテリアライズド・ビューをsum_of_salesではなく、sum_of_sales_mvとネーミングすることで、これがマテリアライズド・ビューであり、表またはビューではないことを表すことができます。

5.3.4 マテリアライズド・ビューの記憶域および表の圧縮について

マテリアライズド・ビューは、ユーザー定義の事前作成表を基にしていないかぎり、データベース内の記憶域を必要とし、これを占有します。したがって、マテリアライズド・ビューの記憶域が必要な場合、それが存在する表領域およびエクステントのサイズを指定する必要があります。

マテリアライズド・ビューに必要な領域のサイズがわからない場合は、DBMS_MVIEW.ESTIMATE_MVIEW_SIZEパッケージを使用して、圧縮されていないマテリアライズド・ビューの格納に必要なバイト数を見積ります。この情報は、設計者が、どの表領域にマテ

リアライズド・ビューを常駐させる必要があるかを判断する場合の参考になります。

表の圧縮は、冗長性の高いデータ(多数の外部キーを持つ表など)で使用します。これは、ROLLUP句を使用して作成したマテリアライズド・ビューには特に役立ちます。表の圧縮によりディスクの使用とメモリーの使用(具体的にはバッファ・キャッシュ)が削減され、読取り専用操作のスケールアップが向上します。また、表の圧縮によって、更新コストと引換えに問合せの実行速度が向上します。

関連項目:

- 表の圧縮の詳細は、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照してください。
- 表の圧縮の詳細は、『Oracle Database管理者ガイド』を参照してください。
- STORAGEセマンティクスの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

5.3.5 マテリアライズド・ビューの作成方法について

表5-2に示すように、マテリアライズド・ビューを作成する方法は2つあります。BUILD IMMEDIATEで作成すると、マテリアライズド・ビューの定義がデータ・ディクショナリ内のスキーマ・オブジェクトに追加されます。その後、ファクト表またはディテール表がSELECT文に従ってスキャンされ、その結果がマテリアライズド・ビューに格納されます。スキャンされる表のサイズによっては、作成処理にかなりの時間がかかる場合があります。

BUILD DEFERRED句を使用することもできます。この句は、データなしでマテリアライズド・ビューを作成するため、DBMS_MVIEW.REFRESHパッケージを使用して、後でデータを移入できます。

関連項目:

マテリアライズド・ビューのリフレッシュ

表5-2 作成方法

作成方法	説明
BUILD IMMEDIATE	マテリアライズド・ビューを作成して、データを移入します。
BUILD DEFERRED	マテリアライズド・ビューの定義は作成しますが、データは移入しません。

5.3.6 マテリアライズド・ビューのクエリー・リライトの有効化について

マテリアライズド・ビューを作成する前に、プロシージャDBMS_MVIEW.EXPLAIN_MVIEWをコールすると、使用可能なクエリー・リライトのタイプを確認できます。また、DBMS_ADVISOR.TUNE_MVIEWを使用すると、マテリアライズド・ビューを最適化して、多くのタイプのクエリー・リライトを使用可能にできます。マテリアライズド・ビューの作成後は、DBMS_MVIEW.EXPLAIN_REWRITEを使用して、特定の問合せがリライトされるかどうか(および、リライトされない理由)を調べることができます。

マテリアライズド・ビューが定義された場合でも、クエリー・リライト機能が自動的に使用されることはありません。クエリー・リライトがデフォルトで使用可能にされている場合でも、マテリアライズド・ビューをクエリー・リライトで使用可能にするには、ENABLE QUERY REWRITE句も指定する必要があります。

マテリアライズド・ビューの作成時に、この句を省略するか、DISABLE QUERY REWRITEとして指定した場合は、後でALTER MATERIALIZED VIEW文を使用して、マテリアライズド・ビューのクエリー・リライトを使用可能にできます。

マテリアライズド・ビューをBUILD DEFERREDとして定義する場合は、完全リフレッシュによってマテリアライズド・ビューにデータを移入しないかぎり、クエリー・リライトは使用できません。

5.3.7 クエリー・リライトの制限について

すべてのマテリアライズド・ビューでクエリー・リライトが可能なわけではありません。クエリー・リライトが予想どおりに実行されない場合は、DBMS_MVIEW.EXPLAIN_REWRITEを使用すると、特定の問合せがリライトに適していない理由がわかります。このプロシージャによって、一部のクエリー・リライトが使用可能でないことが示された場合は、プロシージャDBMS_ADVISOR.TUNE_MVIEWを使用すると、マテリアライズド・ビューの定義を変更してクエリー・リライトを使用可能にできるかどうかを確認できます。また、マテリアライズド・ビューが次の条件をすべて満たしているかどうかを確認してください。

- マテリアライズド・ビューでのクエリー・リライトの制限について
- 一般的なクエリー・リライトの制限

5.3.7.1 クエリー・リライトに関するマテリアライズド・ビューの制限について

次の制限を考慮する必要があります。

- マテリアライズド・ビューの定義問合せに、結果の再現が不可能な式(ROWNUM、SYSDATE、結果の再現が不可能なPL/SQLファンクションなど)を含めることはできません。
- 問合せにLONGまたはLONG RAWデータ型やREFオブジェクトの参照を含めることはできません。
- マテリアライズド・ビューがPREBUILTとして登録された場合、WITH REDUCED PRECISIONでオーバーライドされないかぎり、列の精度は対応するSELECT文の精度と一致する必要があります。
- 定義する問合せにオブジェクトやXMLTYPEへの参照を含めることはできません。
- マテリアライズド・ビューは非エディション・オブジェクトであり、エディション・オブジェクトの名前を解決する必要のある評価エディションが指定されていないかぎり、エディション・オブジェクトに依存できません。
- マテリアライズド・ビューは、特定範囲のエディションでのみクエリー・リライトが可能です。CREATEまたはALTER MATERIALIZED VIEW文の中の*query_rewrite_clause*で、マテリアライズド・ビューがクエリー・リライト可能なエディションの範囲を指定できます。

関連項目:

- マテリアライズド・ビューのための高度なクエリー・リライト
- 『*Oracle Database SQL*言語リファレンス』

5.3.7.2 一般的なクエリー・リライトの制限

次の制限を考慮する必要があります。

- 問合せはローカル表およびリモート表の両方を参照する場合があります。このような問合せは、同じ表を参照している適格なマテリアライズド・ビューがローカルで使用できる場合にはリライトできます。
- SYSは、ディテール表もマテリアライズド・ビューも所有できません。
- マテリアライズド・ビューのGROUP BY句に列または式を指定する場合は、その列または式をSELECT構文のリストにも指

定する必要があります。

- 集計関数は、式の最も外側でのみ使用する必要があります。つまり、AVG(AVG(x))やAVG(x)+ AVG(x)などの集計は実行できません。
- CONNECT BY句は使用できません。

関連項目:

- マテリアライズド・ビューのための高度なクエリー・リライト
- Oracle Database SQL言語リファレンス

5.3.8 マテリアライズド・ビューのリフレッシュ・オプションについて

マテリアライズド・ビューを定義する際は、リフレッシュ方法、リフレッシュのタイプ、トラステッド制約を使用可能にするかどうかの3つのリフレッシュ・オプションを指定できます。オプションを指定しない場合は、デフォルトとして、それぞれON DEMAND、FORCEおよびENFORCED制約が使用されます。

関連項目:

- マテリアライズド・ビューのリフレッシュ・モードについて
- マテリアライズド・ビューのリフレッシュのタイプについて
- トラステッド制約の使用とマテリアライズド・ビューのリフレッシュについて

5.3.8.1 マテリアライズド・ビューのリフレッシュ・モードについて

リフレッシュ実行モードは、ON COMMIT、ON DEMANDおよびON STATEMENTです。作成するマテリアライズド・ビューによっては、一部のオプションを使用できない場合があります。表5-3にリフレッシュ・モードを示します。

表5-3 リフレッシュ・モード

リフレッシュ・モード	説明
ON COMMIT	マテリアライズド・ビューのディテール表の 1 つを変更したトランザクションをコミットした場合、リフレッシュが自動的に実行されます。このモードを使用できるのは、マテリアライズド・ビューが高速リフレッシュ可能な場合(つまり複雑でない場合)のみです。このモードを使用するには、ON COMMIT 権限が必要です。
ON DEMAND	ユーザーが DBMS_MVIEW パッケージに含まれている使用可能なリフレッシュ・プロシージャ(REFRESH、REFRESH_ALL_MVIEWS、REFRESH_DEPENDENT)の 1 つを手動で実行した場合、リフレッシュが実行されます。
ON STATEMENT	マテリアライズド・ビューの実表上で DML 操作を実行する際、トランザクションをコミットしなくても、リフレッシュは自動的に行われます。この方法では、マテリアライズド・ビューの実表でマテリアライズド・ビュー・ログを作成する必要はありません。このモードを使用できるのは、マテリアライズド・ビューが高速

リフレッシュ・モード	説明
	リフレッシュ可能な場合のみです。

ON STATEMENTまたはON COMMITの方法を使用する場合、DMLの完了またはコミットまでの時間が通常より若干長くなることがあります。これは、リフレッシュ操作がDMLの一部(ON STATEMENTリフレッシュの場合)またはコミットの一部(ON COMMITリフレッシュの場合)として実行されるためです。したがって、これらの方法は、多数のユーザーがマテリアライズド・ビューの基礎となる表を同時に変更する場合には適していません。

マテリアライズド・ビューで参照される表に対する挿入、更新または削除操作が、そのマテリアライズド・ビューのリフレッシュと同時に実行されると予想され、そのマテリアライズド・ビューに結合と集計が含まれている場合は、ON DEMAND高速リフレッシュではなくON COMMIT高速リフレッシュを使用することをお薦めします。

マテリアライズド・ビューがリフレッシュされなかったと考えられる場合は、アラート・ログまたはトレース・ファイルをチェックしてください。

DMLまたはコミットによるリフレッシュ時にマテリアライズド・ビューがリフレッシュされなかった場合は、トレース・ファイルに示されたエラーを解決してから、DBMS_MVIEWパッケージを使用してリフレッシュ・プロシージャを明示的に起動する必要があります。この操作を行わないかぎり、マテリアライズド・ビューはコミット時に自動的にリフレッシュされません。

5.3.8.2 マテリアライズド・ビューのリフレッシュのタイプについて

COMPLETE、FAST、FORCEおよびNEVERの4つのオプションのいずれかを選択すると、ディテール表からのマテリアライズド・ビューのリフレッシュ方法を指定できます。表5-4にリフレッシュ・オプションを示します。

表5-4 リフレッシュ・オプション

リフレッシュ・オプション	説明
COMPLETE	マテリアライズド・ビューの定義問合せを再計算することでリフレッシュします。
FAST	マテリアライズド・ビュー・ログに記録された情報を使用するか、SQL*Loader ダイレクト・パスまたはパーティション・メンテナンス操作によって、マテリアライズド・ビューに増分変更を適用してリフレッシュします。
FORCE	可能な場合は、FAST リフレッシュが適用されます。それ以外の場合は、COMPLETE リフレッシュが適用されます。
NEVER	マテリアライズド・ビューはリフレッシュ機能によりリフレッシュされないことを示します。

高速リフレッシュ・オプションが使用可能かどうかは、マテリアライズド・ビューのタイプによって異なります。プロシージャDBMS_MVIEW.EXPLAIN_MVIEWをコールすると、高速リフレッシュが可能かどうかを判断できます。

5.3.8.3 トラステッド制約およびマテリアライズド・ビューのリフレッシュについて

また、リフレッシュ時にトラステッド制約およびQUERY_REWRITE_INTEGRITY = TRUSTEDが使用可能かどうかも特定できます。妥当性チェックが行われていないRELY制約は、すべてトラステッド制約です。たとえば、ディメンションまたはマテリアライズド・ビューに定義された、妥当性チェックが行われていない外部キー/主キーのリレーションシップまたは機能依存性は、UNKNOWN状態です。リフレッシュ時のクエリー・リライトが使用可能な場合は、よりパフォーマンスの高いクエリー・リライトを使用可能にすると、これらによってリフレッシュのパフォーマンスを向上できます。リフレッシュにTRUSTED制約を使用できるマテリアライズド・ビューは、リフレッシュの後も、最新であると信頼できる状態(UNKNOWN状態)のままになります。

この状態は、ビューUSER_MVIEWSの列STALENESSに反映されます。同じビューの列UNKNOWN_TRUSTED_FDも、Y(Yes)に設定されます。

マテリアライズド・ビューのこのプロパティは、作成時にREFRESH USING TRUSTED [ENFORCED] CONSTRAINTSを指定するか、またはALTER MATERIALIZED VIEW DDLを使用して定義できます。

表5-5 制約

使用する制約	説明
TRUSTED CONSTRAINTS	リフレッシュの際、トラステッド制約および QUERY_REWRITE_INTEGRITY = TRUSTED を使用できます。これにより、UNKNOWN 状態または FRESH 状態のマテリアライズド・ビューに対し、リフレッシュ時に妥当性チェックが行われていない RELY 制約を使用してリライトを実行できます。 USING TRUSTED CONSTRAINTS 句では、NULL 以外の仮想プライベート・データベース(VPD)ポリシーを持つ表の最上位にマテリアライズド・ビューを作成できます。この場合、マテリアライズド・ビューが正しく動作することを確認します。マテリアライズド・ビューの結果は、VPD ポリシーによってフィルタ処理された行と列に基づいて計算されます。したがって、正しい結果を得るには、マテリアライズド・ビュー定義を VPD ポリシーで調整する必要があります。USING TRUSTED CONSTRAINTS 句を使用しない場合、実表の VPD ポリシーにより、マテリアライズド・ビューは作成されません。
ENFORCED CONSTRAINTS	リフレッシュの際、妥当性チェック済の制約および QUERY_REWRITE_INTEGRITY = ENFORCED を使用できます。これにより、FRESH 状態のマテリアライズド・ビューに対し、リフレッシュ時に妥当性チェック済の規定された制約のみを使用してリライトを実行できます。

マテリアライズド・ビューの高速リフレッシュは、結合列で使用可能な主キー制約および外部キー制約を使用して最適化されます。この外部キー/主キーによる最適化によって、リフレッシュのパフォーマンスは格段に向上します。たとえば、ファクト表とディメンション表の結合を含むマテリアライズド・ビューがあり、最後のリフレッシュ以降、ディメンション表のみに新しい行が挿入されファクト表は変更されていない場合、このマテリアライズド・ビューに関しては何もリフレッシュされません。これは、ディメンション表の結合列の主キー制約およびファクト表の結合列の外部キー制約により、ディメンション表に挿入された新しい行がファクト表のいずれの行とも結合せず、リフレッシュ対象がないためです。リフレッシュの最適化に関する別の例として、ファクト表とディメンション表の両方に対して、最後のリフレッシュ以降に挿入が実行されたとします。この場合、Oracle Databaseによってデルタ・ファクト表とディメンション表の結合のみが実行されます。外部キー/主キーによる最適化を行わない場合、リフレッシュ時に2つの結合が必要になります。デルタ・ファクトとディメンション表の結合、およびデルタ・ディメンションと挿入前のファクト表のイメージの結合です。

このように結合列で主キー制約および外部キー制約を使用して最適化された高速リフレッシュは、制約を強制して、または強制せずに使用できます。前者の場合は、主キー制約および外部キー制約がOracle Databaseによって強制されます。ただし、制約の管理コストがかかります。後者の場合は、アプリケーションによって主キーと外部キーの関係が保証されるため、制約はRELY NOVALIDATEを使用して宣言され、マテリアライズド・ビューはREFRESH FAST USING TRUSTED CONSTRAINTSオプションを使用して定義されます。

5.3.8.4 高速リフレッシュにおける一般的な制限

マテリアライズド・ビューの定義問合せは、次のように制限されています。

- マテリアライズド・ビューには、SYSDATEやROWNUMなど、結果の再現が不可能な式への参照を含めることはできません。
- マテリアライズド・ビューには、RAWまたはLONG RAWデータ型への参照を含めることはできません。
- SELECT構文のリストに副問合せを含めることはできません。

- SELECT句には分析関数(RANKなど)を指定できません。
- XMLIndex索引が定義されている表を参照することはできません。
- MODEL句を含めることはできません。
- 副問合せでHAVING句を指定できません。
- ANY、ALLまたはNOT EXISTSを含むネストした問合せは使用できません。
- [START WITH …] CONNECT BY句を含めることはできません。
- 別々のサイトに複数のディテール表を含めることはできません。
- ON COMMITのマテリアライズド・ビューにリモート・ディテール表を含めることはできません。
- ネステッド・マテリアライズド・ビューには結合または集計が必須です。
- GROUP BY句を含むマテリアライズド結合ビューおよびマテリアライズド集計ビューは、索引構成表からは選択できません。
- リモート・ビューを基にすることはできません。リモート・ビューに基づくマテリアライズド・ビューでは、完全リフレッシュと強制リフレッシュのみがサポートされます。

  高速リフレッシュが必要な場合は、リモート・ビューの基になるリモート表に基づいてマテリアライズド・ビューを作成します。

5.3.8.5 結合のみを含むマテリアライズド・ビューの高速リフレッシュに関する制限

結合のみを含み、集計を含まないマテリアライズド・ビューの定義問合せには、高速リフレッシュに関して次の制限があります。

- 高速リフレッシュにおける一般的な制限のすべての制限が適用されます。
- GROUP BY句または集計を含めることはできません。
- FROMリスト内のすべての表のROWIDが、問合せのSELECT構文のリストにある必要があります。
- マテリアライズド・ビュー・ログが、問合せのFROMリストにあるすべての実表のROWIDを含む必要があります。
- オブジェクト型の列を含む単純結合がSELECT文に指定されている場合、複数の表から高速リフレッシュ可能なマテリアライズド・ビューを作成できません。

また、次に該当する場合は、選択したリフレッシュ方法が効率的に実行されません。

- 定義問合せに、内部結合のように動作する外部結合が使用されている。このような結合が定義問合せに含まれている場合は、内部結合を使用するように定義問合せをリライトすることを検討してください。
- マテリアライズド・ビューのSELECT構文のリストに複数の表の列の式が含まれている。

5.3.8.6 集計を含むマテリアライズド・ビューの高速リフレッシュに関する制限

集計または結合を含むマテリアライズド・ビューの定義問合せには、高速リフレッシュに関して次の制限があります。

- 高速リフレッシュにおける一般的な制限のすべての制限が適用されます。

高速リフレッシュは、ON COMMITおよびON DEMANDマテリアライズド・ビューの両方についてサポートされますが、次の制限が適用されます。

- マテリアライズド・ビューのすべての表にはマテリアライズド・ビュー・ログが必要であり、マテリアライズド・ビュー・ログには次のことが必要です。
  - マテリアライズド・ビューで参照される表のすべての列を含んでいること。
  - ROWIDおよびINCLUDING NEW VALUESで指定すること。

    - 表に挿入/ダイレクト・ロード、削除および更新が混在する場合は、SEQUENCE句を指定すること。
- 高速リフレッシュについてサポートされる集計関数は、SUM、COUNT、AVG、STDDEV、VARIANCE、MINおよびMAXのみです。
- COUNT(*)を指定する必要があります。
- 集計関数は、式の最も外側でのみ使用する必要があります。つまり、AVG(AVG(x))やAVG(x)+ AVG(x)などの集計は実行できません。
- AVG(expr)などの集計ごとに、対応するCOUNT(expr)が存在している必要があります。さらにSUM(expr)を指定することをお薦めします。
- VARIANCE(expr)またはSTDDEV(expr)が指定された場合は、COUNT(expr)およびSUM(expr)を指定する必要があります。さらにSUM(expr *expr)を指定することをお薦めします。
- 定義問合せのSELECT列は、複数の実表の列を含む複合式にはできません。これに対する解決策の1つとして、ネステッド・マテリアライズド・ビューを使用します。
- SELECT構文のリストには、すべてのGROUP BY列が含まれる必要があります。
- マテリアライズド・ビューは1つまたは複数のリモート表に基づいていないものとします。
- マテリアライズド・ビュー・ログのフィルタ列でCHARデータ型を使用する場合、マスター・サイトおよびマテリアライズド・ビューの文字セットは同じである必要があります。
- マテリアライズド・ビューに次のいずれかが含まれる場合は、従来のDMLの挿入およびダイレクト・ロードに対してのみ高速リフレッシュがサポートされます。
    - MINまたはMAX集計を含むマテリアライズド・ビュー
    - SUM(expr)を含むがCOUNT(expr)を含まないマテリアライズド・ビュー
    - COUNT(*)を含まないマテリアライズド・ビュー

  このようなマテリアライズド・ビューは、挿入専用マテリアライズド・ビューと呼ばれます。
- MAXまたはMINを含むマテリアライズド・ビューは、WHERE句がなければ、削除文または混在型のDML文の後に高速リフレッシュできます。

  削除または複合DMLの後の最大/最小高速リフレッシュには、挿入専用の場合と同じ動作はありません。影響されたグループの最大値/最小値を削除し、再計算します。パフォーマンスへの影響を注意する必要があります。
- 定義問合せのFROM句にビューまたは副問合せを含むマテリアライズド・ビューは、そのビューを完全にマージできれば、高速リフレッシュできます。マージされるビューの詳細は、Oracle Database SQL言語リファレンスを参照してください。
- 外部結合がない場合は、WHERE句に任意の絞込み選択および結合を使用できます。
- 外部結合を含み、集計を含むマテリアライズド・ビューは、外部表のみが変更される場合は、従来のDMLおよびダイレクト・ロードの後に高速リフレッシュできます。また、内部結合表の結合列に一意制約が必要です。外部結合がある場合、すべての結合は、ANDで接続し、等価(=)演算子を使用する必要があります。
- CUBE、ROLLUP、グルーピング・セットまたはその連結を含むマテリアライズド・ビューの場合は、次の制限が適用されます。
    - SELECT構文のリストには、グルーピング識別子を含める必要があります。すなわち、GROUP BYのすべての式にGROUPING_ID関数を含めるか、GROUP BYの各式に対して1つずつGROUPING関数を含める必要があります。たとえば、マテリアライズド・ビューのGROUP BY句が「GROUP BY CUBE(a, b)」の場合、マテリアライズド・ビューを高速リフレッシュ可能にするには、SELECT構文のリストに「GROUPING_ID(a, b)」または「GROUPING(a) AND GROUPING(b)」を含める必要があります。

- GROUP BYの結果、グルーピングが重複しないようにします。たとえば、「GROUP BY a, ROLLUP(a, b)」は、グルーピングの結果が「(a), (a, b), AND (a)」と重複するため、高速リフレッシュできません。

関連項目:

集計を含むマテリアライズド・ビューの使用要件

5.3.8.7 UNION ALLを含むマテリアライズド・ビューの高速リフレッシュに関する制限

UNION ALL集合演算子を含むマテリアライズド・ビューは、次の条件が満たされる場合に、REFRESH FASTオプションをサポートします。

- 定義問合せの最上位レベルにUNION ALL演算子を含める必要があります。

  UNION ALL演算子は、1つの例外を除き、副問合せ内に埋め込むことはできません。この例外とは、定義問合せの形式がSELECT * FROM(UNION ALLを含むビューまたは副問合せ)の場合に、UNION ALLを副問合せのFROM句に指定できるということです。この例を次に示します。

```
CREATE VIEW view_with_unionall AS
(SELECT c.rowid crid, c.cust_id, 2 umarker
 FROM customers c WHERE c.cust_last_name = 'Smith'
 UNION ALL
 SELECT c.rowid crid, c.cust_id, 3 umarker
 FROM customers c WHERE c.cust_last_name = 'Jones');

CREATE MATERIALIZED VIEW unionall_inside_view_mv
REFRESH FAST ON DEMAND AS
SELECT * FROM view_with_unionall;
```

  ビューview_with_unionallは、高速リフレッシュの要件を満たしていることに注意してください。

- UNION ALL問合せ内の各問合せブロックは、集計を含む高速リフレッシュ可能マテリアライズド・ビューまたは結合を含む高速リフレッシュ可能マテリアライズド・ビューの要件を満たす必要があります。

  対応するタイプの高速リフレッシュ可能マテリアライズド・ビューで必要となる、適切なマテリアライズド・ビュー・ログを表に作成する必要があります。

  Oracle Databaseでは、SELECT構文のリストとマテリアライズド・ビュー・ログにROWID列が含まれている場合にのみ、結合を含む単一表マテリアライズド・ビューも特殊なケースとして使用できます。これは、ビューview_with_unionallの定義問合せに示されています。

- 各問合せのSELECT構文のリストにはUNION ALLマーカーを含める必要があり、UNION ALL列の各UNION ALLブランチには個別の定数値または文字列値を含める必要があります。さらに、マーカー列は、各問合せブロックのSELECT構文のリスト内で同じ順序の場所に指定する必要があります。UNION ALLマーカーの詳細は、UNION ALLマーカーとクエリー・リライトを参照してください。

- 外部結合、挿入専用集計マテリアライズド・ビューの問合せ、リモート表などの機能は、UNION ALLを含むマテリアライズド・ビューではサポートされません。ただし、レプリケーションで使用されるマテリアライズド・ビューで、結合または集計を含まないものは、UNION ALLまたはリモート表が使用されるときに高速リフレッシュが可能です。

- UNION ALLを含む高速リフレッシュ可能マテリアライズド・ビューを作成するには、COMPATIBILITY初期化パラメータを9.2.0以上に設定する必要があります。

5.3.8.8 リフレッシュの目的の達成について

この章を通して説明しているEXPLAIN_MVIEWプロシージャの他に、DBMS_ADVISOR.TUNE_MVIEWプロシージャを使用してCREATE MATERIALIZED VIEW文を最適化し、REFRESH FASTおよびENABLE QUERY REWRITEという目的を達成できます。

事前作成表のマテリアライズド・ビューのリフレッシュを参照してください。

5.3.8.8.1 事前作成表のマテリアライズド・ビューのリフレッシュ

事前作成オプションで作成されたマテリアライズド・ビューについては、デフォルトでは、索引I_snap$は作成されません。この索引は、高速リフレッシュのパフォーマンスの向上に役立ちます。この索引の作成方法の詳細は、マテリアライズド・ビューに対する索引付けの選択に示されています。

5.3.8.9 ネステッド・マテリアライズド・ビューのリフレッシュ

ネステッド・マテリアライズド・ビューは、そのデータが、基になるディテール表のデータと同期されていれば、最新であるとみなされます。これは、一部のディテール表が失効したマテリアライズド・ビューになる可能性がある場合でも同様です。

ネステッド・マテリアライズド・ビューは、2つの方法でリフレッシュできます。nestedフラグをTRUEに設定したDBMS_MVIEW.REFRESHと、nestedフラグをTRUEに設定した、実表に対するREFRESH_DEPENDENTです。DBMS_MVIEW.REFRESHを使用する場合は、マテリアライズド・ビュー・チェーン全体が、指定したマテリアライズド・ビューからトップダウン方式でリフレッシュされます。つまり、指定したマテリアライズド・ビューと、依存階層にあるそのすべての子マテリアライズド・ビューが順番にリフレッシュされます。DBMS_MVIEW.REFRESH_DEPENDENTを使用する場合は、チェーン全体がボトムアップでリフレッシュされます。つまり、依存階層にあるすべての親マテリアライズド・ビューは、指定した表から順番にリフレッシュされます。

例5-7 ネステッド・マテリアライズド・ビューのリフレッシュの例

次に、ネステッド・マテリアライズド・ビューをリフレッシュする文の例を示します。

```
DBMS_MVIEW.REFRESH('SALES_MV,COST_MV', nested => TRUE);
```

この文ではまず、依存性分析に基づいてsales_mvおよびcost_mvのすべての子マテリアライズド・ビューがリフレッシュされ、次に、指定された2つのマテリアライズド・ビューがリフレッシュされます。

*_MVIEWSビューのSTALE_SINCE列を問い合せると、マテリアライズド・ビューがいつ失効になったのかを確認できます。

5.3.9 マテリアライズド・ビューのORDER BY句

ORDER BY句は、CREATE MATERIALIZED VIEW文で使用できます。これは、マテリアライズド・ビューを最初に作成するときのみ使用されます。完全リフレッシュまたは高速リフレッシュ時には使用されません。

大規模なマテリアライズド・ビューに対する問合せのパフォーマンスを向上させるには、ORDER BY句に指定されている順序で、マテリアライズド・ビューに行を格納します。このように最初に順序付けると、データを物理クラスタ化できます。マテリアライズド・ビューが順序付けられた列上に索引を作成する場合、その索引を使用してマテリアライズド・ビューの行にアクセスすると、物理クラスタ化によるディスクI/Oに対する時間が削減されます。

ORDER BY句は、マテリアライズド・ビューの定義の一部とはみなされません。そのため、Oracle Databaseが様々なタイプのマテリアライズド・ビュー(集計を含まないマテリアライズド結合ビューなど)を検出する方法に違いはありません。同じ理由で、クエリー・リライトは、ORDER BY句の影響を受けません。これは、CREATE TABLE ... ORDER BYの特性と似ています。

5.3.10 Oracle Enterprise Managerを使用したマテリアライズド・ビューの作成

マテリアライズド・ビューは、Enterprise Managerのマテリアライズド・ビュー・オブジェクトを選択することで、作成することもできます。この方法が使用された場合でも、必要な情報は同じです。

5.3.11 マテリアライズド・ビューとNLSパラメータの使用

ある種のマテリアライズド・ビューを使用する場合は、NLSパラメータが作成時と同じに設定されているかどうかを確認する必要があります。この制限を伴うマテリアライズド・ビューは、次のとおりです。

- NLSパラメータの設定に応じて異なる値を戻す式。たとえば、(date > "01/02/03")や(rate <= "2.150")は、NLSパラメータに依存する式です。
- 結合の一方の側が文字データである等価結合。この等価結合の結果は照合に依存します。また、クエリー・リライトの場合に不適切な結果となったり、リフレッシュ操作後にマテリアライズド・ビューの一貫性がなくなるなど、セッションごとに状態が変化することがあります。
- マテリアライズド・ビューのSELECT構文のリスト内、または集計を含むマテリアライズド・ビューの集計内で、文字データへの内部変換を生成する式。この制限は、aとbが数値フィールドである場合のa+bのように、数値データのみを伴う式には適用されません。

5.3.12 マテリアライズド・ビューへのコメントの追加

マテリアライズド・ビューにコメントを追加することができます。

例: マテリアライズド・ビューへのコメントの追加

次の文は、既存のマテリアライズド・ビューに関するコメントをデータ・ディクショナリ・ビューに追加します。

```
COMMENT ON MATERIALIZED VIEW sales_mv IS 'sales materialized view';
```

この文の実行後にコメントを表示するには、カタログ・ビューの{USER, DBA} ALL_MVIEW_COMMENTSを問い合せます。たとえば、次の例を考えてみます。

```
SELECT MVIEW_NAME, COMMENTS
FROM USER_MVIEW_COMMENTS WHERE MVIEW_NAME = 'SALES_MV';
```

出力は次のようになります。

```
MVIEW_NAME                      COMMENTS
-----------      ------------------------
SALES_MV         sales materialized view
```

ノート: COMPATIBILITYが10.0.1以上に設定されている場合、マテリアライズド・ビューのコンテナ表に対してCOMMENT ON TABLEは発行できません。この文を発行すると、次のエラー・メッセージが戻されます。

```
ORA-12098: cannot comment on the materialized view.
```

事前作成表にコメントが設定されている場合にマテリアライズド・ビューを作成すると、そのコメントがマテリアライズド・ビューに継承されます。既存のコメントには、'(from table)'という接頭辞が付けられます。たとえば、売上のサマリー情報を格納するための表sales_summaryが作成されているとします。この表には、'Sales summary data'というコメントがすでに関連付けられています。ここで、この事前作成表をコンテナ表として使用する同じ名前のマテリアライズド・ビューを作成するとします。マテリアライズド・ビューの作成後、コメントは'(from table) Sales summary data'になります。

ただし、事前作成表sales_summaryにコメントが設定されていない場合は、'Sales summary data'というコメントが追加されます。その後、このマテリアライズド・ビューを削除すると、そのコメントが'(from materialized view) Sales summary data'となって事前作成表に渡されます。

5.4 マテリアライズド・ビュー・ログの作成

高速リフレッシュを使用するにはマテリアライズド・ビュー・ログが必要ですが、パーティション・チェンジ・トラッキング・リフレッシュの場合は例外です。つまり、ディテール表でマテリアライズド・ビューのパーティション・チェンジ・トラッキングがサポートされている場合、そのマテリアライズド・ビューを高速リフレッシュするために、そのディテール表にマテリアライズド・ビュー・ログは必要ありません。しかし原則的に、高速リフレッシュを使用する場合は、マテリアライズド・ビュー・ログを作成する必要があります。マテリアライズド・ビュー・ログは、CREATE MATERIALIZED VIEW LOG文を使用して、変更が行われる実表に定義します。対象のマテリアライズド・ビューの上位に別のマテリアライズド・ビューがない場合は、前者のマテリアライズド・ビューにマテリアライズド・ビュー・ログは作成されません。これはネステッド・マテリアライズド・ビューの場合に該当します。マテリアライズド・ビューを高速リフレッシュするには、通常、マテリアライズド・ビュー・ログの定義でROWID句を指定する必要があります。また、集計を含むマテリアライズド・ビューの場合は、マテリアライズド・ビューで参照される表のすべての列、INCLUDING NEW VALUES句およびSEQUENCE句を含む必要があります。集計または結合を含むローカル・マテリアライズド・ビューは、通常、WITH COMMIT SCN句を使用することにより高速リフレッシュのパフォーマンスを向上させることができます。

次のマテリアライズド・ビュー・ログの例は、sales表に対して作成されています。

```
CREATE MATERIALIZED VIEW LOG ON sales WITH ROWID
(prod_id, cust_id, time_id, channel_id, promo_id, quantity_sold, amount_sold)
INCLUDING NEW VALUES;
```

または、次のようにしてコミットSCNベースのマテリアライズド・ビュー・ログを作成することも可能です。

```
CREATE MATERIALIZED VIEW LOG ON sales WITH ROWID
 (prod_id, cust_id, time_id, channel_id, promo_id, quantity_sold, amount_sold),
COMMIT SCN INCLUDING NEW VALUES;
```

混在型のDML操作(複数の表に対するINSERT、UPDATEまたはDELETE操作の組合せ)を実行しないことが確実でないかぎり、キーワードSEQUENCEをマテリアライズド・ビュー・ログ文に含めることをお薦めします。複数の表に対するINSERT文、UPDATE文またはDELETE文の組合せでの高速リフレッシュをサポートするには、マテリアライズド・ビュー・ログにSEQUENCE列が必要です。ただし、マテリアライズド・ビュー・ログの作成後にSEQUENCE列を追加することも可能です。

混在型DML操作の境界は、マテリアライズド・ビューがON COMMITであるかON DEMANDであるかによって決定されます。

- ON COMMITの場合、マテリアライズド・ビューのリフレッシュはトランザクションのコミット時に発生するため、混在型DML文は同じトランザクション内で発生します。
- ON DEMANDの場合、混在型DML文はリフレッシュとリフレッシュの間に発生します。次のマテリアライズド・ビュー・ログの例は、SEQUENCEキーワードを含むsales表に対して作成されています。

  ```
  CREATE MATERIALIZED VIEW LOG ON sales WITH SEQUENCE, ROWID
  (prod_id, cust_id, time_id, channel_id, promo_id,
   quantity_sold, amount_sold) INCLUDING NEW VALUES;
  ```

この項では、次の項目について説明します。

- マテリアライズド・ビュー・ログでのFORCEオプションの使用
- マテリアライズド・ビュー・ログのパージ

5.4.1 マテリアライズド・ビュー・ログでのFORCEオプションの使用

FORCEを指定し、ADD句で指定した項目がすでにそのマテリアライズド・ビュー・ログに指定されていた場合、Oracleはエラーを戻しません。Oracleは、既存の要素をそのまま無視し、ログに存在しない項目をマテリアライズド・ビュー・ログに追加します。たとえば、cust_idなどのフィルタ列を追加したが、この列がすでに存在していた場合、Oracle Databaseはこの重複を無視し、エ

ラーも戻しません。

5.4.2 マテリアライズド・ビュー・ログのパージ

マテリアライズド・ビュー・ログは、マテリアライズド・ビューのリフレッシュ・プロセス中にパージ(消去)できますが、後でパージすることも可能です。後でパージすれば、リフレッシュのパフォーマンスに影響を与えません。パージを実行するタイミングは、次のようにPURGE句を使用して指定できます。

```
CREATE MATERIALIZED VIEW LOG ON sales
PURGE START WITH sysdate NEXT sysdate+1
WITH ROWID
 (prod_id, cust_id, time_id, channel_id, promo_id, quantity_sold, amount_sold)
INCLUDING NEW VALUES;
```

次のような文を発行すると、パージ情報についてUSER_MVIEW_LOGSを問い合せることもできます。

```
SELECT PURGE_DEFERRED, PURGE_INTERVAL, LAST_PURGE_DATE, LAST_PURGE_STATUS
FROM USER_MVIEW_LOGS
WHERE LOG_OWNER "SH" AND MASTER = 'SALES';
```

マテリアライズド・ビュー・ログの作成時にパージを設定できるだけでなく、次のような文を発行することにより既存のマテリアライズド・ビュー・ログを変更することもできます。

```
ALTER MATERIALIZED VIEW LOG ON sales PURGE IMMEDIATE;
```

関連項目:

マテリアライズド・ビュー・ログの構文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

5.5 近似問合せに基づいたマテリアライズド・ビューの作成

近似問合せに基づいたマテリアライズド・ビューは、その定義問合せで近似関数を返すSQL関数を使用します。

サマリーを計算して近似処理を集計し、その後の分析または問合せのためにこれらの結果をマテリアライズド・ビューに格納できます。行のグループ内のすべてのディメンションの近似集計を計算するサマリー近似処理は、詳細集計の実行に使用できます。さらにサマリー・データを集計して、高レベルの分析に使用できる集計近似処理を取得できるため、Oracle Databaseでは、高レベルの集計を計算するために実表を再度スキャンしません。Oracle Databaseでは、高レベルの集計を計算するために実表を再度スキャンしません。これは、既存の集計結果を使用して、高レベルの集計を計算するだけです。たとえば、各州および各国で販売された製品の概数を格納するサマリー近似処理を作成できます。次に、この集計近似処理を使用して、各国内の各製品の概数を返します。

近似結果を返すSQL関数を含むマテリアライズド・ビューを作成するには、次のようにします。

- 適切な関数を含む定義問合せでCREATE MATERIALIZED VIEW文を実行します。

  たとえば、マテリアライズド・ビューの定義問合せでAPPROX_PERCENTILE関数を使用します。

例5-8 **近似問合せに基づいたマテリアライズド・ビューの作成**

次の例では、各日に販売された各製品の概数を格納するマテリアライズド・ビューを作成します。

```
CREATE MATERIALIZED VIEW approx_count_distinct_pdt_mv
ENABLE QUERY REWRITE AS
```

```
SELECT t.calendar_year, t.calendar_month_number, t.day_number_in_month, approx_count_distinct(prod_id)
daily_detail
FROM sales s, times t
WHERE s.time_id = t.time_id
GROUP BY t.calendar_year, t.calendar_month_number, t.day_number_in_month;
```

関連項目:

- 近似問合せに基づいたマテリアライズド・ビューのリフレッシュ
- 近似結果を返すパーセンタイル関数の使用

5.6 ビットマップ・ベースのCOUNT(DISTINCT)関数を含むマテリアライズド・ビューの作成

COUNT(DISTINCT)関数に基づくマテリアライズド・ビューは、整数列に対してビットマップベースの操作を使用することで、より高いパフォーマンスを実現できます。

Oracle Databaseリリース19c以降では、COUNT(DISTINCT)操作の計算を表すためにビットマップ表現を使用するSQL集計関数に基づいてマテリアライズド・ビューを作成できます。これらの関数には、BITMAP_BUCKET_NUMBER、BITMAP_BIT_POSITIONおよびBITMAP_CONSTRUCT_AGGが含まれます。

ビットマップに基づいてマテリアライズド・ビューを作成するには、次のようにします。

1. マテリアライズド・ビューの基になる表に対するマテリアライズド・ビュー・ログが存在することを確認します。
2. CREATE MATERIALIZED VIEWコマンドを使用して、マテリアライズド・ビューを作成します。

   次の例では、SH.SALES表に基づいて、非加算ファクトを含むマテリアライズド・ビューを作成します。

```
SQL> create materialized view mv_sales as
  2  select PROMO_ID,
  3  BITMAP_BUCKET_NUMBER(PROD_ID) bm_bktno,
  4  BITMAP_CONSTRUCT_AGG(BITMAP_BIT_POSITION(PROD_ID),'RAW') bm_details
  5  from sales
  6  group by PROMO_ID,BITMAP_BUCKET_NUMBER(PROD_ID);

Materialized view created.
```

関連トピック

- ビットマップベースのCOUNT(DISTINCT)関数に基づくクエリー・リライトおよびマテリアライズド・ビュー

5.7 既存のマテリアライズド・ビューの登録

一部のデータ・ウェアハウスでは、通常のユーザー表にマテリアライズド・ビューが実装されています。このソリューションによって、マテリアライズド・ビューのパフォーマンスが向上しますが、次のような問題があります。

- すべてのSQLアプリケーションで、クエリー・リライトができるわけではありません。
- あるアプリケーションで定義されたマテリアライズド・ビューには、別のアプリケーションから透過的にアクセスできません。
- 一般に、高速パラレルまたは高速マテリアライズド・ビュー・リフレッシュはサポートされていません。

これらの制限があり、既存のマテリアライズド・ビューが非常に大きく、再作成にコストがかかりすぎる場合があるため、できるだけ、

既存のマテリアライズド・ビューの表を登録する必要があります。ユーザー定義のマテリアライズド・ビューは、CREATE MATERIALIZED VIEW ... ON PREBUILT TABLE文で登録できます。一度登録されたマテリアライズド・ビューでは、クエリー・リライトが使用できるか、いずれかのリフレッシュ方法でメンテナンスできるか、あるいはその両方が可能です。

表の内容は、定義問合せをマテリアライズド・ビューとして登録したときに、定義問合せのマテリアライズ化を反映している必要があります。また、定義問合せの各列は、一致するデータ型を持つ表の列に対応している必要があります。ただし、WITH REDUCED PRECISIONを指定して、定義問合せの列の精度が表の列の精度とは異なるようにすることは可能です。

表およびマテリアライズド・ビューの名前は同じである必要がありますが、表は、表としての個別性を維持し、マテリアライズド・ビューの定義問合せで参照されない列を含むことができます。このような列は、非管理列と呼ばれます。リフレッシュ操作中に行が挿入されると、その行の各非管理列はそのデフォルト値に設定されます。したがって、非管理列は、デフォルト値を持たないかぎり、NOT NULL制約を持つことはできません。

事前作成表に基づくマテリアライズド・ビューは、パラメータQUERY_REWRITE_INTEGRITYがSTALE_TOLERATEDまたはTRUSTEDに設定されている場合に、クエリー・リライトによる選択の対象になります。

関連項目:

整合性レベルの詳細は、マテリアライズド・ビューのための基本的なクエリー・リライトを参照してください。

事前作成表に作成されたマテリアライズド・ビューを削除しても、その表は残り、マテリアライズド・ビューのみが削除されます。

次の例では、ユーザー定義表の登録に必要な2つのステップを示しています。まず、表が作成され、次にマテリアライズド・ビューが表と同じ名前で定義されます。このマテリアライズド・ビューsum_sales_tab_mvは、クエリー・リライトで使用できます。

```
CREATE TABLE sum_sales_tab
PCTFREE 0  TABLESPACE demo
STORAGE (INITIAL 8M) AS
SELECT s.prod_id, SUM(amount_sold) AS dollar_sales,
       SUM(quantity_sold) AS unit_sales
FROM sales s GROUP BY s.prod_id;

CREATE MATERIALIZED VIEW sum_sales_tab_mv
ON PREBUILT TABLE WITHOUT REDUCED PRECISION
ENABLE QUERY REWRITE AS
SELECT s.prod_id, SUM(amount_sold) AS dollar_sales,
    SUM(quantity_sold) AS unit_sales
FROM sales s GROUP BY s.prod_id;
```

この表は領域を節約するために圧縮することもできます。

ユーザー定義のマテリアライズド・ビューは、データの更新サイクルより長いスケジュールでリフレッシュされる場合があります。たとえば、月単位のマテリアライズド・ビューは各月の月末にしか更新されないことがあります。また、マテリアライズド・ビューの値は、常に、リフレッシュが完了した期間のみを参照します。これらのマテリアライズド・ビューに対して直接作成されたレポートでは、現行の(リフレッシュが未完了の)期間にはないデータのみが暗黙的に選択されます。ユーザー定義のマテリアライズド・ビューに時間ディメンションがすでに含まれる場合、次のことに従う必要があります。

- ユーザー定義のマテリアライズド・ビューは、登録してから、更新サイクルごとに高速リフレッシュする必要があります。
- リフレッシュが完了した期間を選択するビューを作成できます。
- レポートが、ユーザー定義のマテリアライズド・ビューを直接参照するのではなく、ビューを参照するように変更する必要があります。

ユーザー定義のマテリアライズド・ビューに時間ディメンションが含まれていない場合、時間ディメンションを含む新しいマテリアライズド・ビューを作成する必要があります(可能な場合)。また、この場合、新しいマテリアライズド・ビューの時間列を、ビューに集計する必要があります。

5.8 マテリアライズド・ビューに対する索引付けの選択

マテリアライズド・ビューに対して行う最も一般的な操作は、問合せの実行および高速リフレッシュですが、各操作には、異なるパフォーマンス要件があります。問合せの実行では、マテリアライズド・ビュー・キー列のすべてのサブセットがアクセスされる必要があり、これらの列のサブセット上で結合および集計が行われる必要がある場合があります。そのため、問合せの実行では、通常、各マテリアライズド・ビュー・キー列上に単一列のビットマップ索引が定義されている場合に、パフォーマンスが最適化されます。

結合のみを含むマテリアライズド・ビューに高速リフレッシュ・オプションを使用する場合は、ROWIDを含む列上に索引を作成して、リフレッシュ操作のパフォーマンスを向上させることをお薦めします。

集計を使用するマテリアライズド・ビューが高速リフレッシュ可能な場合は、CREATE MATERIALIZED VIEW文にUSING NO INDEXが指定されていないかぎり、高速リフレッシュ・プロシージャに適切な索引が作成されます。

マテリアライズド・ビューがパーティション化されている場合、マテリアライズド・ビューに対するパーティション・メンテナンス操作の実行後は索引が使用できなくなるため、高速リフレッシュを行うには索引を再作成する必要があります。

事前作成オプションでマテリアライズド・ビューを作成した場合、I_snap$索引は自動的に作成されません。この索引を使用すると、高速リフレッシュのパフォーマンスが大幅に向上し、次のような文を発行することにより、手動で作成することができます。

```
CREATE UNIQUE INDEX <OWNER>."I_SNAP$_<MVIEW_NAME>" ON <OWNER>.<MVIEW_NAME>
 (SYS_OP_MAP_NONNULL("LOG_DATE"))
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DE
FAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE <TABLESPACE_NAME>;
```

関連項目:

SQLアクセス・アドバイザを使用してマテリアライズド・ビューに最適な索引を判断する方法については、*『Oracle Database SQLチューニング・ガイド』*を参照してください。

5.9 マテリアライズド・ビューの削除

マテリアライズド・ビューを削除するには、DROP MATERIALIZED VIEW文を使用します。たとえば、次の文を考えてみます。

```
DROP MATERIALIZED VIEW sales_sum_mv;
```

この文では、マテリアライズド・ビューsales_sum_mvが削除されます。ある表に対して作成されていたマテリアライズド・ビューが削除されても、元の表は削除されませんが、リフレッシュ・メカニズム機能を使用したメンテナンスやクエリー・リライトによる使用はできなくなります。また、Oracle Enterprise Managerを使用して、マテリアライズド・ビューを削除できます。

5.10 マテリアライズド・ビュー機能の分析

DBMS_MVIEW.EXPLAIN_MVIEWプロシージャを使用すると、マテリアライズド・ビューまたは作成前のマテリアライズド・ビューで可能なことを調べることができます。特に、このプロシージャにより次のことを判断できます。

- マテリアライズド・ビューが高速リフレッシュ可能かどうか
- このマテリアライズド・ビューで実行できるクエリー・リライトのタイプ
- パーティション・チェンジ・トラッキング・リフレッシュの可否

このプロシージャを使用する方法が最も簡単で、DBMS_MVIEW.EXPLAIN_MVIEWプロシージャの使用で説明しています。DBMS_MVIEW.EXPLAIN_MVIEWをコールし、既存のマテリアライズド・ビューのスキーマ名とマテリアライズド・ビュー名を単一パラメータとして渡すのみで済みます。あるいは、作成前のマテリアライズド・ビューについてはSELECT文字列を指定したり、完全なCREATE MATERIALIZED VIEW文を指定したりできます。マテリアライズド・ビューまたは作成前のマテリアライズド・ビューが分析され、結果がデフォルトの表MV_CAPABILITIES_TABLEまたは配列MSG_ARRAYに書き込まれます。

結果をMSG_ARRAYに入れるとき以外は、EXPLAIN_MVIEWをコールする前にutlxmv.sqlスクリプトを実行する必要があるので注意してください。このスクリプトはadminディレクトリにあります。このスクリプトは、現在のスキーマにMV_CAPABILITIES_TABLEを作成します。各種機能の説明は表5-6、可能なすべてのメッセージは表5-7を参照してください。

5.10.1 DBMS_MVIEW.EXPLAIN_MVIEWプロシージャの使用

EXPLAIN_MVIEWプロシージャのパラメータは、次のとおりです。

- stmt_id

  オプション・パラメータ。出力行を特定のEXPLAIN_MVIEWコールと関連付けるためにクライアントが提供する一意識別子です。

- mv

  分析対象となる既存のマテリアライズド・ビューの名前、あるいは作成前のマテリアライズド・ビューの問合せ定義またはCREATE MATERIALIZED VIEW文全体。

- msg-array

  出力を受け取るPL/SQLのVARRAY。

EXPLAIN_MVIEWでは、指定したマテリアライズド・ビューのリフレッシュ機能とリライト機能が分析され、その結果が(複数行形式で)MV_CAPABILITIES_TABLEまたはMSG_ARRAYに挿入されます。

関連項目:

DBMS_MVIEWパッケージの詳細は、『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

この項では、次の項目について説明します。

- DBMS_MVIEW.EXPLAIN_MVIEW宣言
- MV_CAPABILITIES_TABLEの使用
- MV_CAPABILITIES_TABLE.CAPABILITY_NAMEの詳細
- MV_CAPABILITIES_TABLE列の詳細

5.10.1.1 DBMS_MVIEW.EXPLAIN_MVIEW宣言

次のPL/SQL宣言はDBMS_MVIEWパッケージで行われるもので、結果を表またはVARRAYに出力する場合の、既存のマテリアライズド・ビューと作成前のマテリアライズド・ビューを説明するパラメータの順序とデータ型を示します。

MV_CAPABILITIES_TABLEへの出力を指定して、既存または作成前のマテリアライズド・ビューを説明します。

```
DBMS_MVIEW.EXPLAIN_MVIEW (mv           IN VARCHAR2,
                          stmt_id IN VARCHAR2:= NULL);
```

VARRAYへの出力を指定して、既存または作成前のマテリアライズド・ビューを説明します。

```
DBMS_MVIEW.EXPLAIN_MVIEW (mv           IN VARCHAR2,
                          msg_array    OUT SYS.ExplainMVArrayType);
```

5.10.1.2 MV_CAPABILITIES_TABLEの使用

DBMS_MVIEW.EXPLAIN_MVIEWの最も単純な使用方法の1つは、次の構造を持つ、MV_CAPABILITIES_TABLEを使用することです。

```
CREATE TABLE MV_CAPABILITIES_TABLE
(STATEMENT_ID      VARCHAR(30),    -- Client-supplied unique statement identifier
 MVOWNER           VARCHAR(30),    -- NULL for SELECT based EXPLAIN_MVIEW
 MVNAME            VARCHAR(30),    -- NULL for SELECT based EXPLAIN_MVIEW
 CAPABILITY_NAME   VARCHAR(30),    -- A descriptive name of the particular
                                   -- capability:
                                   -- REWRITE
                                   --   Can do at least full text match
                                   --   rewrite
                                   -- REWRITE_PARTIAL_TEXT_MATCH
                                   --   Can do at least full and partial
                                   --   text match rewrite
                                   -- REWRITE_GENERAL
                                   --   Can do all forms of rewrite
                                   -- REFRESH
                                   --   Can do at least complete refresh
                                   -- REFRESH_FROM_LOG_AFTER_INSERT
                                   --   Can do fast refresh from an mv log
                                   --   or change capture table at least
                                   --   when update operations are
                                   --   restricted to INSERT
                                   -- REFRESH_FROM_LOG_AFTER_ANY
                                   --   can do fast refresh from an mv log
                                   --   or change capture table after any
                                   --   combination of updates
                                   -- PCT
                                   --   Can do Enhanced Update Tracking on
                                   --   the table named in the RELATED_NAME
                                   --   column.  EUT is needed for fast
                                   --   refresh after partitioned
                                   --   maintenance operations on the table
                                   --   named in the RELATED_NAME column
                                   --   and to do non-stale tolerated
                                   --   rewrite when the mv is partially
                                   --   stale with respect to the table
                                   --   named in the RELATED_NAME column.
                                   --   EUT can also sometimes enable fast
                                   --   refresh of updates to the table
                                   --   named in the RELATED_NAME column
                                   --   when fast refresh from an mv log
                                   --   or change capture table is not
                                   --   possible.
                                   -- See Table 5-6
 POSSIBLE          CHARACTER(1),   -- T = capability is possible
                                   -- F = capability is not possible
```

```
RELATED_TEXT      VARCHAR(2000),  -- Owner.table.column, alias name, and so on
                                  -- related to this message. The specific
                                  -- meaning of this column depends on the
                                  -- NSGNO column. See the documentation for
                                  -- DBMS_MVIEW.EXPLAIN_MVIEW() for details.
RELATED_NUM       NUMBER,         -- When there is a numeric value
                                  -- associated with a row, it goes here.
MSGNO             INTEGER,        -- When available, QSM message # explaining
                                  -- why disabled or more details when
                                  -- enabled.
MSGTXT            VARCHAR(2000),  -- Text associated with MSGNO.
SEQ               NUMBER);        -- Useful in ORDER BY clause when
                                  -- selecting from this table.
```

MV_CAPABILITIES_TABLEを作成するには、adminディレクトリにあるutlxmv.sqlスクリプトを使用できます。

関連項目:

- パーティション・チェンジ・トラッキングの詳細は、マテリアライズド・ビューのリフレッシュを参照してください。
- パーティション・チェンジ・トラッキングの詳細は、マテリアライズド・ビューのための高度なクエリー・リライトを参照してください。

例5-9 DBMS_MVIEW.EXPLAIN_MVIEW

まず、マテリアライズド・ビューを作成します。あるいは、作成前のマテリアライズド・ビューのSELECT文またはCREATE MATERIALIZED VIEW文全体を使用して、そのマテリアライズド・ビューにEXPLAIN_MVIEWを実行することもできます。

```
CREATE MATERIALIZED VIEW cal_month_sales_mv
BUILD IMMEDIATE
REFRESH FORCE
ENABLE QUERY REWRITE AS
SELECT t.calendar_month_desc,  SUM(s.amount_sold) AS dollars
FROM sales s,  times t WHERE s.time_id = t.time_id
GROUP BY t.calendar_month_desc;
```

次に、マテリアライズド・ビューを指定してEXPLAIN_MVIEWを起動します。各行が論理順に表示されるように、ORDER BY句にSEQ列を使用する必要があります。機能が使用可能でない場合は、P列にN、MSGTXT列に説明が表示されます。複数の理由で機能が使用可能でない場合は、理由ごとに表示されます。

```
EXECUTE DBMS_MVIEW.EXPLAIN_MVIEW ('SH.CAL_MONTH_SALES_MV');

SELECT capability_name,  possible, SUBSTR(related_text,1,8)
  AS rel_text, SUBSTR(msgtxt,1,60) AS msgtxt
FROM MV_CAPABILITIES_TABLE
ORDER BY seq;


CAPABILITY_NAME                P    REL_TEXT     MSGTXT
---------------                -    --------     ------
PCT                            N
REFRESH_COMPLETE               Y
REFRESH_FAST                   N
REWRITE                        Y
PCT_TABLE                      N    SALES        no partition key or PMARKER in select list
PCT_TABLE                      N    TIMES        relation is not a partitioned table
REFRESH_FAST_AFTER_INSERT      N    SH.TIMES     mv log must have new values
```

```
REFRESH_FAST_AFTER_INSERT       N    SH.TIMES     mv log must have ROWID
REFRESH_FAST_AFTER_INSERT       N    SH.TIMES     mv log does not have all necessary columns
REFRESH_FAST_AFTER_INSERT       N    SH.SALES     mv log must have new values
REFRESH_FAST_AFTER_INSERT       N    SH.SALES     mv log must have ROWID
REFRESH_FAST_AFTER_INSERT       N    SH.SALES     mv log does not have all necessary columns
REFRESH_FAST_AFTER_ONETAB_DML   N    DOLLARS      SUM(expr) without COUNT(expr)
REFRESH_FAST_AFTER_ONETAB_DML   N                 see the reason why
                                                  REFRESH_FAST_AFTER_INSERT is disabled
REFRESH_FAST_AFTER_ONETAB_DML   N                 COUNT(*) is not present in the select list
REFRESH_FAST_AFTER_ONETAB_DML   N                 SUM(expr) without COUNT(expr)
REFRESH_FAST_AFTER_ANY_DML      N                 see the reason why
                                                  REFRESH_FAST_AFTER_ONETAB_DML is disabled
REFRESH_FAST_AFTER_ANY_DML      N    SH.TIMES     mv log must have sequence
REFRESH_FAST_AFTER_ANY_DML      N    SH.SALES     mv log must have sequence
REFRESH_PCT                     N                 PCT is not possible on any of the detail
                                                  tables in the materialized view
REWRITE_FULL_TEXT_MATCH         Y
REWRITE_PARTIAL_TEXT_MATCH      Y
REWRITE_GENERAL                 Y
REWRITE_PCT                     N                 PCT is not possible on any detail tables
```

5.10.1.3 MV_CAPABILITIES_TABLE.CAPABILITY_NAMEの詳細

表5-6に、CAPABILITY_NAME列の値の説明を示します。

表5-6 CAPABILITY_NAME列の詳細

CAPABILITY_NAME	説明
PCT	この機能が使用可能な場合は、1 つ以上のディテール関係のパーティション・チェンジ・トラッキングが可能です。この機能が使用可能でない場合、マテリアライズド・ビューで参照されるディテール関係のパーティション・チェンジ・トラッキングは不可能です。
REFRESH_COMPLETE	この機能が使用可能な場合は、マテリアライズド・ビューの完全リフレッシュが可能です。
REFRESH_FAST	この機能が使用可能な場合は、少なくとも特定の状況下での高速リフレッシュが可能です。
REWRITE	この機能が使用可能な場合は、少なくともテキストの完全一致のクエリー・リライトが可能です。この機能が使用可能でない場合、クエリー・リライトは形式を問わず不可能です。
PCT_TABLE	この機能が使用可能な場合は、トップ・レベルの FROM リストで指定されたパーティション表に対して適用可能です。有効にすると、RELATED_TEXT 列に示されたパーティション表に対し、パーティション・チェンジ・トラッキング(PCT)が適用されます。 PCT は、RELATED_TEXT 列で指定された表に対するパーティション・メンテナンス操作後の高速リフレッシュをサポートするために必要です。 また、PCT はマテリアライズド・ビュー・ログからの高速リフレッシュが不可能な場合でも、RELATED_TEXT 列で指定された表の更新に関する高速リフレッシュをサポートします。

CAPABILITY_NAME	説明
	また、PCT は、RELATED_TEXT 列で指定された表に関して、マテリアライズド・ビューの部分的な失効がある場合に、クエリー・リライトをサポートする目的でも必要です。 この機能が使用不可能な場合、PCT は RELATED_TEXT 列で指定された表には適用されません。この場合、RELATED_TEXT 列で指定された表に対するパーティション・メンテナンス操作後に、高速リフレッシュできません。また、RELATED_TEXT 列で指定された表に対する更新の PCT ベースのリフレッシュもできません。最終的に、RELATED_TEXT 列で指定された表に関して、マテリアライズド・ビューの部分的な失効がある場合に、クエリー・リライトをサポートできません。
PCT_TABLE_REWRITE	この機能が使用可能な場合は、トップ・レベルの FROM リストで指定されたパーティション表に対して適用可能です。可能な場合、PCT は RELATED_TEXT 列で指定されたパーティション表に適用されます。 この機能は、RELATED_TEXT 列で指定された表に関して部分的に失効状態であるマテリアライズド・ビューへのクエリー・リライトをサポートするために必要です。 この機能が使用不可能で、このマテリアライズド・ビューが、RELATED_TEXT 列で指定された表に関して部分的に失効状態である場合、クエリー・リライトはサポートされません。
REFRESH_FAST_AFTER_INSERT	この機能が使用可能であれば、少なくとも更新が INSERT 操作のみに制限されている場合は、マテリアライズド・ビュー・ログからの高速リフレッシュが可能です。完全リフレッシュも可能です。この機能が使用可能でない場合、マテリアライズド・ビュー・ログからの高速リフレッシュは、形式を問わず不可能です。
RREFRESH_FAST_AFTER_ONETAB_DML	この機能が使用可能であれば、すべての更新操作が単一表に対して実行される場合は、更新操作のタイプを問わずマテリアライズド・ビュー・ログからの高速リフレッシュが可能です。この機能が使用可能でなければ、更新操作が複数の表に対して実行される場合、マテリアライズド・ビュー・ログからの高速リフレッシュが不可能な場合があります。
REFRESH_FAST_AFTER_ANY_DML	この機能が使用可能であれば、更新操作のタイプや更新される表の数を問わず、マテリアライズド・ビュー・ログからの高速リフレッシュが可能です。この機能が使用可能でなければ、更新操作(INSERT 以外)が複数の表に影響する場合に、マテリアライズド・ビュー・ログからの高速リフレッシュが不可能なことがあります。
REFRESH_FAST_PCT	この機能が使用可能な場合は、PCT を使用した高速リフレッシュが可能です。通常、これは、PCT が可能として示されているディテール表に対するパーティション・メンテナンス操作後に、リフレッシュが可能であることを意味します。
REWRITE_FULL_TEXT_MATCH	この機能が使用可能な場合は、テキストの完全一致のクエリー・リライトが可能です。この機能が使用可能でない場合、テキストの完全一致のクエリー・リライトは不可能です。

CAPABILITY_NAME	説明
REWRITE_PARTIAL_TEXT_MATCH	この機能が使用可能な場合は、少なくともテキストの完全一致および部分一致のクエリー・リライトが可能です。この機能が使用可能でない場合、少なくともテキストの部分一致と一般的なクエリー・リライトは不可能です。
REWRITE_GENERAL	この機能が使用可能な場合は、一般的なクエリー・リライトと、テキストの完全一致および部分一致のクエリー・リライトを含め、クエリー・リライト機能がすべて使用可能です。この機能が使用可能でない場合、少なくとも一般的なクエリー・リライトは不可能です。
REWRITE_PCT	この機能が使用可能な場合、クエリー・リライトでは、QUERY_REWRITE_INTEGRITY = ENFORCED または TRUSTED モードでも、部分的に失効したマテリアライズド・ビューを使用できます。この機能が使用可能でない場合、クエリー・リライトでは、QUERY_REWRITE_INTEGRITY = STALE_TOLERATED モードの場合にのみ、部分的に失効したマテリアライズド・ビューを使用できます。

5.10.1.4 MV_CAPABILITIES_TABLE列の詳細

表5-7に、RELATED_TEXTおよびRELATED_NUM列の意味を示します。

表5-7 MV_CAPABILITIES_TABLE列の詳細

MSGNO	MSGTXT	RELATED_NUM	RELATED_TEXT
NULL	NULL		PCT 機能専用。PCT が使用可能になっている表の[*owner*.]*name*
2066	Oracle エラー内に記載されます	発生した Oracle エラー番号	
2067	SELECT リストにパーティション・キー、PMARKER または結合依存式がありません		PCT がサポートされていない関係の[*owner*.]*name*
2068	リレーションがパーティション化されていません		PCT がサポートされていない関係の[*owner*.]*name*
2069	PCT では複数列で構成されるパーティション・キーをサポートしていません		PCT がサポートされていない関係の[*owner*.]*name*
2070	PCT ではこのタイプのパーティション化をサポートしていません		PCT がサポートされていない関係の[*owner*.]*name*
2071	内部エラー: PCT 障害コードが定義さ	認識されない数値 PCT	PCT がサポートされていない関係の[*owner*.]*name*

MSGNO	MSGTXT	RELATED_NUM	RELATED_TEXT
	れていません	障害コード	
2072	ネストされたマテリアライズド・ビューの高速リフレッシュのための要件は満たされていません		
2077	マテリアライズド・ビュー・ログは最新の全体リフレッシュよりも新しいです		マテリアライズド・ビュー・ログが必要な表の[*owner.*]*table_name*
2078	マテリアライズド・ビュー・ログは新しい値を持つ必要があります		マテリアライズド・ビュー・ログが必要な表の[*owner.*]*table_name*
2079	マテリアライズド・ビュー・ログは ROWID を持つ必要があります		マテリアライズド・ビュー・ログが必要な表の[*owner.*]*table_name*
2080	マテリアライズド・ビュー・ログは主キーを持つ必要があります		マテリアライズド・ビュー・ログが必要な表の[*owner.*]*table_name*
2081	マテリアライズド・ビュー・ログは必要な列をすべて持っているわけではありません		マテリアライズド・ビュー・ログが必要な表の[*owner.*]*table_name*
2082	マテリアライズド・ビュー・ログの問題		マテリアライズド・ビュー・ログが必要な表の[*owner.*]*table_name*
2099	マテリアライズド・ビューは FROM リストのリモート表またはビューを参照しています	SELECT キーワードから問題の表またはビューへのオフセット	問題の表またはビューの[*owner.*]*name*
2126	複数のマスター・サイトです		最初の異なるノードの名前、または、最初の異なるノードがローカルの場合は NULL
2129	結合またはフィルタ条件が複合しています		結合またはフィルタ条件に関連する表の[*owner.*]*name*(または使用可能でない場合は NULL)
2130	式が高速リフレッシュのためにサポートされていません	SELECT キーワードから問題の式へのオフセット	問題の式の SELECT リストの別名
2150	SELECT リストは UNION 演算子を超えて同一である必要があります	SELECT キーワードから SELECT リスト内の最初の異なる選択項目への	SELECT リスト内の最初の異なる選択項目の別名

MSGNO	MSGTXT	RELATED_NUM	RELATED_TEXT
		オフセット	
2182	PCT は結合依存性を介して使用できます		PCT_TABLE_REWRITE が使用可能になっていない関係の[*owner*.]*name*
2183	PCT を有効化する式が分析関数の PARTITION BY またはモデルにありません	認識されない数値 PCT 障害コード	PCT が使用可能になっていない関係の[*owner*.]*name*
2184	PCT を有効化する式はロールアップできません		PCT が使用可能になっていない関係の[*owner*.]*name*
2185	SELECT リストにパーティション・キーまたは PMARKER がありません		PCT_TABLE_REWRITE が使用可能になっていない関係の[*owner*.]*name*
2186	GROUP OUTER JOIN があります		
2187	外部表のマテリアライズド・ビュー		

6 高度なマテリアライズド・ビュー

この章では、マテリアライズド・ビューの高度な使用方法について説明します。次の項目が含まれます。

- パーティション化とマテリアライズド・ビューについて
- 分析処理環境でのマテリアライズド・ビューについて
- マテリアライズド・ビューとモデルについて
- マテリアライズド・ビューのセキュリティ問題について
- マテリアライズド・ビューの無効化
- マテリアライズド・ビューの変更
- リアルタイムのマテリアライズド・ビューの使用

6.1 パーティション化とマテリアライズド・ビューについて

データ・ウェアハウスに格納されているデータの量は膨大であるため、パーティション化は、データベースの設計時に非常に有効なオプションです。ファクト表のパーティション化によって、スケーラビリティの向上とシステム管理の簡素化を実現できます。また、効率的に再作成できるローカル索引を定義できるようになります。ファクト表をパーティション化することにより、マテリアライズド・ビューに対するパーティション・チェンジ・トラッキング(PCT)リフレッシュが可能になるので、マテリアライズド・ビューを高速リフレッシュできる可能性も高くなります。マテリアライズド・ビューのパーティション化は、リフレッシュの面でもメリットがあります。リフレッシュ・プロシージャは、より多くの場合にパラレルDMLを使用できるようになり、PCTベースのリフレッシュは、パーティションの切捨てを使用してマテリアライズド・ビューを効率的にメンテナンスできるようになります。

関連項目:

パーティション化の詳細は、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照してください。

この項では、次の項目について説明します。

- パーティション・チェンジ・トラッキングについて
- マテリアライズド・ビューのパーティション化
- 事前作成表のパーティション化
- ローリング・マテリアライズド・ビュー

6.1.1 パーティション・チェンジ・トラッキングについて

最新の状態かどうかの追跡対象を、マテリアライズド・ビュー全体ではなく、より細かく限定でき、それによるメリットが得られます。これはパーティション・チェンジ・トラッキング(PCT)により実現できます。PCTとは、特定のディテール表のパーティションにより影響を受ける、マテリアライズド・ビュー内の行を識別する方法です。1つ以上のディテール表がパーティション化されている場合は、マテリアライズド・ビュー内で、変更されたディテール・パーティションに対応する特定の行を識別できます。これらの行はパーティションが変更されると失効しますが、他のすべての行は最新のままです。

PCTを使用すると、特定のパーティションに対応するマテリアライズド・ビューの行を識別できます。PCTは、ディテール表に対するパーティション・メンテナンス操作後の高速リフレッシュのサポートにも使用されます。たとえば、ディテール表のパーティションが切り

捨てられるか削除されると、マテリアライズド・ビュー内で影響を受ける行が識別され、削除されます。

QUERY_REWRITE_INTEGRITY = ENFORCEDまたはTRUSTEDモードのときは、マテリアライズド・ビュー全体を失効とみなすのではなく、マテリアライズド・ビューの行のうち最新のものと失効しているものが識別され、最新状態の行をクエリー・リライトで使用できるようになります。DBA_MVIEW_DETAIL_PARTITIONなどの一部のビューでは、どのパーティションが失効しているかまたは最新なのかが具体的に示されます。マテリアライズド・ビューに結合依存の式があることによって、変更対象の表のパーティション・チェンジ・トラッキングが使用可能になる場合は、部分的に失効しているマテリアライズド・ビューへのリライトは行われません。

関連項目:

詳細は、結合依存の式およびパーティション・チェンジ・トラッキングについてを参照してください。

パーティション・チェンジ・トラッキングは、パーティション・レベルおよびサブパーティション・レベルでの失効を追跡しますが(コンポジット・パーティション表に関して)、PCTリフレッシュの粒度のレベルは、パーティション化戦略の最上位レベルのみであることに注意してください。その結果、コンポジット・パーティション表のサブパーティションの1つにあるデータに変更があると、影響を受けた1つのサブパーティションのみが失効としてマークされ、表の残りの部分はリライトが可能になります。ただし、PCTリフレッシュにより、影響を受けたサブパーティションを含む、パーティション全体がリフレッシュされます。

PCTをサポートするには、マテリアライズド・ビューが次の要件を満たしている必要があります。

- マテリアライズド・ビューで参照される1つ以上のディテール表が、パーティション化されている必要があります。
- パーティション表には、レンジ、リストまたはコンポジットのいずれかのパーティション化を使用し、レンジまたはリストはパーティション化戦略の最上位レベルである必要があります。
- 最上位レベルのパーティション・キーは、単一列のみで構成する必要があります。
- マテリアライズド・ビューには、ディテール表のパーティション・キー列、パーティション・マーカー、ROWIDまたは結合依存の式のいずれかを含める必要があります。
- GROUP BY句を使用する場合は、パーティション・キー列、パーティション・マーカー、ROWIDまたは結合依存の式をGROUP BY句に指定する必要があります。
- 分析ウィンドウ関数またはMODEL句を使用する場合は、パーティション・キー列、パーティション・マーカー、ROWIDまたは結合依存の式を、それぞれのPARTITION BY副次句に指定する必要があります。
- データ修正の発生が可能なのは、パーティション表のみです。マテリアライズド・ビューに結合依存の式を含む表に対してPCTリフレッシュを実行する際は、いずれの結合依存の表についても、データ修正が発生していてはなりません。
- COMPATIBILITY初期化パラメータは9.0.0.0.0以上に設定する必要があります。

ビュー、リモート表または外部結合を参照するマテリアライズド・ビューの場合、PCTはサポートされません。

関連項目:

DBMS_MVIEW.PMARKER関数およびパーティション・マーカーの詳細は、『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

この項では、次の項目について説明します。

- パーティション・キーおよびパーティション・チェンジ・トラッキングについて

- 結合依存の式およびパーティション・チェンジ・トラッキングについて
- パーティション・マーカーおよびパーティション・チェンジ・トラッキングについて
- パーティション・チェンジ・トラッキングでの部分的なリライトについて

6.1.1.1 パーティション・キーおよびパーティション・チェンジ・トラッキングについて

パーティション・チェンジ・トラッキングは、マテリアライズド・ビューに必要な情報が存在すれば、ベースとなるディテール表のパーティションの各行を、対応するマテリアライズド・ビューの各行に関連付けることができます。そのためには、ディテール表のパーティション・キー列をSELECT構文のリストに含めます。また、GROUP BYを使用する場合は、GROUP BYリストにも含めます。

例として、日次の顧客売上を格納するマテリアライズド・ビューを考えてみます。次の例では、shサンプル・スキーマおよび3つのディテール表(sales、productsおよびtimes)を使用してマテリアライズド・ビューを作成します。sales表はtime_id列でパーティション化されており、products表はprod_id列でパーティション化されています。times表はパーティション表ではありません。

例6-1 パーティション・キーを使用したマテリアライズドビュー

```
CREATE MATERIALIZED VIEW LOG ON SALES WITH ROWID
   (prod_id, time_id, quantity_sold, amount_sold) INCLUDING NEW VALUES;
CREATE MATERIALIZED VIEW LOG ON PRODUCTS WITH ROWID
   (prod_id, prod_name, prod_desc) INCLUDING NEW VALUES;
CREATE MATERIALIZED VIEW LOG ON TIMES WITH ROWID
   (time_id, calendar_month_name, calendar_year) INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW cust_dly_sales_mv
BUILD DEFERRED REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE AS
SELECT s.time_id, p.prod_id, p.prod_name, COUNT(*),
         SUM(s.quantity_sold), SUM(s.amount_sold),
         COUNT(s.quantity_sold), COUNT(s.amount_sold)
FROM sales s, products p, times t
WHERE  s.time_id = t.time_id AND s.prod_id = p.prod_id
GROUP BY s.time_id, p.prod_id, p.prod_name;
```

cust_dly_sales_mvでは、sales表でPCTが使用可能になります。これは、パーティション化キー列time_idがマテリアライズド・ビューに含まれているためです。

6.1.1.2 結合依存式およびパーティション・チェンジ・トラッキングについて

結合依存の式とは、パーティション化キーでの等価結合を介して直接的または間接的にパーティション化ディテール表に結合されている表の列で構成され、結合キーのディメンション属性またはディメンション階層の親である式のことです。ディテール表へのパスに存在する一連の表は、結合依存の表と呼ばれます。次の点を考慮してください。

```
SELECT s.time_id, t.calendar_month_name
FROM sales s, times t WHERE s.time_id = t.time_id;
```

この問合せでは、times表が結合依存の表です。この表は、パーティション化キー列のtime_idでsales表に結合されているためです。また、calendar_month_nameはtimes.time_idのディメンション階層属性です。これは、calendar_month_nameがtimes.mon_idの属性で、times.mon_idがtimes.time_idのディメンション階層の親であるためです。したがって、times表からの式calendar_month_nameは結合依存の式ということになります。次に、別の例について考えてみます。

```
SELECT s.time_id, y.calendar_year_name
FROM sales s, times_d d, times_m m, times_y y
WHERE s.time_id = d.time_id AND d.day_id = m.day_id AND m.mon_id = y.mon_id;
```

ここで、times表はtimes_d、times_mおよびtimes_yの各表に非正規化されています。times_y表からの式

calendar_year_nameは結合依存の式で、times_d、times_mおよびtimes_yの各表は結合依存の表です。その理由は、times_y表がtimes_m表およびtimes_d表を介して、そのパーティション化キー列time_idで間接的にsales表に結合されているためです。

これによりユーザーは、ディテール表のパーティション化キーよりも上位のレベルでの集計を含むマテリアライズド・ビューを作成できます。次に、月次の顧客売上を格納するマテリアライズド・ビューの例を見てみます。

例6-2 マテリアライズド・ビューの作成: 結合依存の式

マテリアライズド・ビュー・ログは定義済であるとすると、このマテリアライズド・ビューは、次のDDLで作成できます。

```
CREATE MATERIALIZED VIEW cust_mth_sales_mv
BUILD DEFERRED REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE AS
SELECT t.calendar_month_name, p.prod_id, p.prod_name, COUNT(*),
    SUM(s.quantity_sold), SUM(s.amount_sold),
    COUNT(s.quantity_sold), COUNT(s.amount_sold)
FROM sales s, products p, times t
WHERE  s.time_id = t.time_id AND s.prod_id = p.prod_id
GROUP BY t.calendar_month_name, p.prod_id, p.prod_name;
```

ここでは、結合依存の表timesと、times.calendar_month_nameはtimes.time_idによって決定されるディメンション属性であるというリレーションシップを使用して、ディテール表の行を、対応するマテリアライズド・ビューの行に関連付けることができます。これにより、sales表のパーティション・チェンジ・トラッキングが可能になります。さらに、マテリアライズド・ビューにproducts表のパーティション化キー列prod_idが含まれているので、products表のPCTも可能になります。

6.1.1.3 パーティション・マーカーおよびパーティション・チェンジ・トラッキングについて

DBMS_MVIEW.PMARKERファンクションは、マテリアライズド・ビューのカーディナリティ(表の行数に対する個別値の数の比率)を大幅に削減するように設計されています(例については、例6-3を参照)。このファンクションは、指定されたパーティション表内の指定された行のパーティションまたはサブパーティションを、一意に識別するパーティションの識別子を返します。したがって、DBMS_MVIEW.PMARKERファンクションは、SELECT句およびGROUP BY句のパーティション・キー列のかわりに使用されます。

マテリアライズド・ビューでのPL/SQLファンクションの一般的なケースとは異なり、DBMS_MVIEW.PMARKERを使用すると、リライト・モードがQUERY_REWRITE_INTEGRITY = ENFORCEDであっても、そのマテリアライズド・ビューでのリライトは可能です。

PMARKERファンクションの使用例として、指定した年の製品カテゴリ別収益などの典型的な数値の計算を考えてみます。毎月1,000種類の製品が販売されていた場合、マテリアライズド・ビューには12,000行が格納されていることになります。

例6-3 マテリアライズド・ビューにおけるパーティション・マーカーの使用

例として、製品カテゴリごとの年間売上収益を格納するマテリアライズド・ビューを考えます。各製品カテゴリにはおよそ数百種類の製品があるので、products表のパーティション化キー列prod_idをマテリアライズド・ビューに含めると、カーディナリティはかなり高くなります。このマテリアライズド・ビューでは、かわりにDBMS_MVIEW.PMARKERファンクションを使用します。この場合、マテリアライズド・ビューのカーディナリティは、products表のパーティションの数だけ倍加します。

```
CREATE MATERIALIZED VIEW prod_yr_sales_mv
BUILD DEFERRED
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE AS
SELECT DBMS_MVIEW.PMARKER(p.rowid), p.prod_category, t.calendar_year, COUNT(*),
       SUM(s.amount_sold), SUM(s.quantity_sold),
       COUNT(s.amount_sold), COUNT(s.quantity_sold)
FROM   sales s, products p, times t
WHERE  s.time_id = t.time_id AND s.prod_id = p.prod_id
GROUP BY  DBMS_MVIEW.PMARKER (p.rowid), p.prod_category, t.calendar_year;
```

prod_yr_sales_mvのSELECT構文のリストには、products表に対するDBMS_MVIEW.PMARKERファンクションが含まれています。これにより、パーティション・キー列prod_idでグルーピングした場合よりもカーディナリティに与える影響を大幅に抑えながら、products表のパーティション・チェンジ・トラッキングが可能になります。この例では、prod_yr_sales_mvに必要な集計レベルは、products.prod_category別にグルーピングすることです。DBMS_MVIEW.PMARKERファンクションを使用した場合、マテリアライズド・ビューのカーディナリティは、products表のパーティションの数だけ倍加するだけで済みます。通常、この方法では、パーティション・キー列を含めた場合より、カーディナリティへの影響が大幅に小さくなります。

sales表でパーティション・チェンジ・トラッキングが使用可能になることに注意してください。これは、結合依存の式calendar_yearがSELECT構文のリストに含まれているためです。

6.1.1.4 パーティション・チェンジ・トラッキングでの部分的なリライトについて

後続のINSERT文で、表salesのsales_part3パーティションに新しい行が追加されると仮定します。cust_dly_sales_mvは、パーティション・キーを使用して表salesでのPCTが可能になっているため、Oracleはこの時点で、sales_part3パーティションに対応するマテリアライズド・ビューcust_dly_sales_mvの失効している行を識別できます(その他の行は最新状態なので変更されません)。クエリー・リライトは、マテリアライズド・ビューcust_mth_sales_mvおよびprod_yr_sales_mvの最新部分を識別できません。これは、PCTが結合依存の式を使用して表salesで使用可能になっているためです。クエリー・リライトは、PCTがパーティション・キーまたはパーティション・マーカーを使用してディテール表で使用可能になっている場合にのみ、ディテール表への変更発生時にマテリアライズド・ビューの最新部分を識別できます。

6.1.2 マテリアライズド・ビューのパーティション化

次の例に示すように、マテリアライズド・ビューのパーティション化には、Oracleの標準パーティション化句を使用したマテリアライズド・ビューの定義が含まれます。この文では、part_sales_mvというマテリアライズド・ビューが作成されます。このマテリアライズド・ビューは3つのパーティションを使用し、高速リフレッシュが可能で、クエリー・リライトに使用できます。

```
CREATE MATERIALIZED VIEW part_sales_mv
PARALLEL PARTITION BY RANGE (time_id)
(PARTITION month1
      VALUES LESS THAN (TO_DATE('31-12-1998', 'DD-MM-YYYY'))
      PCTFREE 0
      STORAGE (INITIAL 8M)
      TABLESPACE sf1,
 PARTITION month2
      VALUES LESS THAN (TO_DATE('31-12-1999', 'DD-MM-YYYY'))
      PCTFREE 0
      STORAGE (INITIAL 8M)
      TABLESPACE sf2,
 PARTITION month3
      VALUES LESS THAN (TO_DATE('31-12-2000', 'DD-MM-YYYY'))
      PCTFREE 0
      STORAGE (INITIAL 8M)
      TABLESPACE sf3)
BUILD DEFERRED
REFRESH FAST
ENABLE QUERY REWRITE AS
SELECT s.cust_id, s.time_id,
  SUM(s.amount_sold) AS sum_dol_sales, SUM(s.quantity_sold) AS sum_unit_sales
  FROM sales s GROUP BY s.time_id, s.cust_id;
```

6.1.3 事前作成表のパーティション化

マテリアライズド・ビューは、パーティション化された事前作成表に登録できます。マテリアライズド・ビューのパーティション化によるメリットでは、事前作成表のパーティション化の利点について説明します。次に、この例を示します。

```
CREATE TABLE part_sales_tab_mv(time_id, cust_id, sum_dollar_sales, sum_unit_sale)
PARALLEL PARTITION BY RANGE (time_id)
(PARTITION month1
      VALUES LESS THAN (TO_DATE('31-12-1998', 'DD-MM-YYYY'))
      PCTFREE 0
      STORAGE (INITIAL 8M)
      TABLESPACE sf1,
 PARTITION month2
      VALUES LESS THAN (TO_DATE('31-12-1999', 'DD-MM-YYYY'))
      PCTFREE 0
      STORAGE (INITIAL 8M)
      TABLESPACE sf2,
PARTITION month3
      VALUES LESS THAN (TO_DATE('31-12-2000', 'DD-MM-YYYY'))
      PCTFREE 0
      STORAGE (INITIAL 8M)
      TABLESPACE sf3)  AS
SELECT s.time_id, s.cust_id, SUM(s.amount_sold) AS sum_dollar_sales,
  SUM(s.quantity_sold) AS sum_unit_sales
FROM sales s GROUP BY s.time_id, s.cust_id;

CREATE MATERIALIZED VIEW part_sales_tab_mv
ON PREBUILT TABLE
ENABLE QUERY REWRITE AS
SELECT s.time_id,  s.cust_id, SUM(s.amount_sold) AS sum_dollar_sales,
     SUM(s.quantity_sold) AS sum_unit_sales
FROM sales s GROUP BY s.time_id, s.cust_id;
```

この例では、part_sales_tab_mv表は3か月ごとにパーティション化され、事前作成表を使用するためにマテリアライズド・ビューが登録されています。このマテリアライズド・ビューは、ENABLE QUERY REWRITE句を含んでいるため、クエリー・リライトに使用できます。

6.1.3.1 マテリアライズド・ビューのパーティション化によるメリット

マテリアライズド・ビューをディテール表のパーティション化キー列または結合依存の式でパーティション化すると、TRUNCATE PARTITION文を使用してリフレッシュの際にマテリアライズド・ビューの1つ以上のパーティションを削除し、後でそのパーティションに新しいデータを再移入するという作業がより効率的になります。Oracle Databaseは、次の条件とパーティション・チェンジ・トラッキングについてで説明したその他の条件が満たされている場合に、パーティションの切捨てを伴うこの種の高速リフレッシュ(PCTリフレッシュと呼ぶ)を使用します。

- マテリアライズド・ビューが、ディテール表のパーティション化キー列または結合依存の式でパーティション化されている。
- パーティション化キー列または結合式を使用してPCTが使用可能になっている場合、マテリアライズド・ビューはレンジ・パーティション化またはリスト・パーティション化されている必要がある。
- PCTリフレッシュがアトミックではない。

6.1.4 ローリング・マテリアライズド・ビュー

データ・ウェアハウスまたはデータ・マートに時間ディメンションが含まれる場合、通常は最も古い情報をアーカイブしてから、その記憶域を新しい情報に再使用する必要があります。これは、ローリング・ウィンドウ・シナリオと呼ばれます。ファクト表またはマテリアライズド・ビューに時間ディメンションが含まれ、時間属性によって水平にパーティション化されている場合、ロールアウトされるデータの量がレンジ・パーティション化の場合と等しいか、または少なくとも整列していれば、ローリング・マテリアライズド・ビューの管理が、高速で管理コストの低いパーティション管理のみに軽減されます。

データ・ウェアハウスにローリング・マテリアライズド・ビューを作成する場合は、パーティション・メンテナンス操作を実行する頻度を

決定する必要があります。また、ファクト表およびマテリアライズド・ビューをパーティション化して、古いデータが必要なくなったときに必要なシステム管理によるオーバーヘッドを削減する必要があります。また、頻繁には更新されないパーティションに対してはデータ圧縮の使用も考慮します。

レンジ・パーティション化の使用には、制限はありません。たとえば、時間値およびキー値の両方を使用したコンポジット・パーティション化が、データに対して適切なパーティション・ソリューションとなる場合もあります。

関連項目:

CONSIDER FRESHおよび圧縮の詳細は、マテリアライズド・ビューのリフレッシュを参照してください。

6.2 分析処理環境でのマテリアライズド・ビューについて

この項では、分析SQLで使用される概念と、リレーショナル・データベースでこのタイプの問合せを処理する方法を説明します。また、一般的な例を使用してマテリアライズド・ビューを作成する最適な方法を示します。

次のトピックでは、各種環境でのマテリアライズド・ビューの詳細を説明しています。

- マテリアライズド・ビューと分析ビューについて
- マテリアライズド・ビューと階層的キューブについて
- マテリアライズド・ビューのパーティション化によるメリット
- マテリアライズド・ビューの圧縮について
- 集合演算子を含むマテリアライズド・ビューについて

6.2.1 マテリアライズド・ビューと分析ビューについて

分析ビューまたは階層の問合せにまたがるマテリアライズド・ビューの作成は、サポートされていません。

6.2.2 マテリアライズド・ビューと階層的キューブについて

データ・ウェアハウス環境では一般にデータをスター・スキーマ形式で表示しますが、分析SQL問合せではデータを階層的キューブ形式で表示します。階層的キューブにはそのディメンションそれぞれのロールアップ階層に沿って集計されたデータが含まれ、これらの集計はディメンションをまたがって結合されます。ビジネス・インテリジェンス問合せに必要な典型的な集計の集合が含まれます。

例6-4 階層的キューブ

次の2つのディメンションを持つ売上データセットがあり、それぞれのディメンションに4レベルの階層があるとします。

- 時間。(all times)、year、quarterおよびmonthを含みます。
- 製品。(all products)、division、brandおよびitemを含みます。

つまり、階層的キューブ内には16個の集計グループがあることになります。これは、時間の4つのレベルと製品の4つのレベルを掛け合せてキューブが生成されるためです。各ディメンションの4つのレベルを表6-1に示します。

表6-1 時間別および製品別ROLLUP

時間別ROLLUP	製品別ROLLUP

時間別ROLLUP	製品別ROLLUP
year、quarter、month	division、brand、item
year、quarter	division、brand
year	division
all times	all products

ディメンションとレベルの数を増やすと、計算するグループ数が急激に増えることに注意してください。この例には16個のグループが含まれていますが、同数レベルのディメンションを2つ追加するだけで、4×4×4×4 = 256個のグループになります。また、ディメンションに階層が複数あると、同じようにグループ数が増えることも考慮してください。たとえば、時間ディメンションにもう1つの会計月という階層があるとします。会計月は会計四半期、さらに会計年にロールアップします。爆発的に増えるグループの処理は、歴史的に、オンライン分析処理システム用のデータ格納での大きな課題です。

典型的なオンライン分析問合せでは、キューブの様々な部分をスライスおよびダイスして、1つのレベルの集計と別のレベルの集計を比較します。たとえば、問合せにより、2002年1月の食料雑貨部門の売上を求め、2001年全体の食料雑貨部門の売上合計とこの値を比較する場合があります。

6.2.3 マテリアライズド・ビューのパーティション化によるメリット

複数の集計グループを含むマテリアライズド・ビューのパフォーマンスがリフレッシュおよびクエリー・リライト向けに最大化されるのは、適切にパーティション化されている場合です。

ローリング・ウィンドウ・シナリオでのPCTリフレッシュには、時間ディメンションのいくつかのレベルにおける最上位レベルでのパーティション化が必要です。また、このマテリアライズド・ビューへのクエリー・リライトでパーティション・プルーニングを行うには、GROUPING_ID列でのパーティション化が必要です。したがって、これらのマテリアライズド・ビューのパーティション化方法として最も効果的なのは、コンポジット・パーティション化((time, GROUPING_ID)列でのレンジ-リスト・パーティション化)ということになります。この方法でマテリアライズド・ビューをパーティション化すると、次のことが可能になります。

- PCTリフレッシュ: リフレッシュのパフォーマンスが向上します。
- パーティション・プルーニング: 関連する集計グループのみがアクセスされるため、問合せの処理コストが大幅に削減されます。

PCTリフレッシュを使用しない場合は、GROUPING_ID列のみでリスト・パーティション化します。

6.2.4 マテリアライズド・ビューの圧縮について

冗長性の高いデータ(多数の外部キーを持つ表など)を使用するときは、データ圧縮を考慮するようにします。特に、ROLLUP句を使用して作成したマテリアライズド・ビューがこの候補になります。

関連項目:

- データの圧縮の構文および制限の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- 圧縮の詳細は、マテリアライズド・ビューの記憶域および表の圧縮についてを参照してください。

6.2.5 集合演算子を含むマテリアライズド・ビューについて

Oracle Databaseでは、定義問合せに集合演算子が含まれるマテリアライズド・ビューをサポートしています。集合演算子を含むマテリアライズド・ビューをクエリー・リライト対応として作成できるようになりました。マテリアライズド・ビューは、ON COMMITまたはON DEMANDリフレッシュのいずれかを使用してリフレッシュできます。

定義問合せの最上位レベルにUNION ALL演算子が含まれ、UNION ALL内の各問合せブロックが、集計を含むマテリアライズド・ビューまたは結合のみを含むマテリアライズド・ビューの要件を満たす場合は、高速リフレッシュがサポートされます。さらに、マテリアライズド・ビューには、各問合せブロック内の個別の値を含む定数列(UNION ALLマーカー)を含める必要があります。これは、次の例では、列1 markerと2 markerです。

関連項目:

UNION ALLを使用したマテリアライズド・ビューの高速リフレッシュに関する制限の詳細は、UNION ALLを含むマテリアライズド・ビューの高速リフレッシュに関する制限を参照してください。

6.2.5.1 UNION ALLを使用するマテリアライズド・ビューの例

次に、UNION ALLを含む、高速リフレッシュ可能なマテリアライズド・ビューの作成例を示します。

例6-5 2つの結合ビューを伴うUNION ALLを使用するマテリアライズド・ビュー

2つの結合ビューを持つUNION ALLマテリアライズド・ビューを作成するには、マテリアライズド・ビュー・ログにROWID列を含める必要があります。次の例で、UNION ALLマーカーは列1 markerと2 markerです。

```
CREATE MATERIALIZED VIEW LOG ON sales WITH ROWID;
CREATE MATERIALIZED VIEW LOG ON customers WITH ROWID;

CREATE MATERIALIZED VIEW unionall_sales_cust_joins_mv
REFRESH FAST ON COMMIT
ENABLE QUERY REWRITE AS
(SELECT c.rowid crid, s.rowid srid, c.cust_id, s.amount_sold, 1 marker
FROM sales s, customers c
WHERE s.cust_id = c.cust_id AND c.cust_last_name = 'Smith')
UNION ALL
(SELECT c.rowid crid, s.rowid srid, c.cust_id, s.amount_sold, 2 marker
FROM sales s, customers c
WHERE s.cust_id = c.cust_id AND c.cust_last_name = 'Brown');
```

例6-6 結合および集計を伴うUNION ALLを使用するマテリアライズド・ビュー

次の例は、結合を含むマテリアライズド・ビューと集計を含むマテリアライズド・ビューのUNION ALLを示します。この例で注意することは2つです。対応するSELECT構文のリストの列のデータ型が一致するようにするには、NULLまたは定数を使用できます。また、UNION ALLマーカー列には文字列リテラルを指定できます。この例では、'Year' umarker、'Quarter' umarkerまたは'Daily' umarkerです。

```
CREATE MATERIALIZED VIEW LOG ON sales WITH ROWID, SEQUENCE
(amount_sold, time_id)
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON times WITH ROWID, SEQUENCE
  (time_id, fiscal_year, fiscal_quarter_number, day_number_in_week)
INCLUDING NEW VALUES;
```

```
CREATE MATERIALIZED VIEW unionall_sales_mix_mv
REFRESH FAST ON DEMAND AS
(SELECT 'Year' umarker, NULL, NULL, t.fiscal_year,
        SUM(s.amount_sold) amt, COUNT(s.amount_sold), COUNT(*)
 FROM sales s, times t
 WHERE s.time_id = t.time_id
 GROUP BY t.fiscal_year)
UNION ALL
(SELECT 'Quarter' umarker, NULL, NULL, t.fiscal_quarter_number,
        SUM(s.amount_sold) amt, COUNT(s.amount_sold), COUNT(*)
FROM sales s, times t
WHERE s.time_id = t.time_id and t.fiscal_year = 2001
GROUP BY t.fiscal_quarter_number)
UNION ALL
(SELECT 'Daily' umarker, s.rowid rid, t.rowid rid2, t.day_number_in_week,
        s.amount_sold amt, 1, 1
FROM sales s, times t
WHERE s.time_id = t.time_id
AND t.time_id between '01-Jan-01' AND '01-Dec-31');
```

6.3 マテリアライズド・ビューとモデルについて

マテリアライズド・ビューでは、SQLでの配列ベースの計算を可能にするモデルを使用できます。MODEL句による計算は高コストになる場合があるので、モデル計算用とSELECT ... GROUP BY問合せ用の2つのマテリアライズド・ビューを使用した方がよい場合もあります。たとえば、1つの長いマテリアライズド・ビューを使用するかわりに、次のような2つのマテリアライズド・ビューを作成できます。

```
CREATE MATERIALIZED VIEW my_groupby_mv
REFRESH FAST
ENABLE QUERY REWRITE AS
SELECT country_name country, prod_name prod, calendar_year year,
  SUM(amount_sold) sale, COUNT(amount_sold) cnt, COUNT(*) cntstr
FROM sales, times, customers, countries, products
WHERE sales.time_id = times.time_id AND
      sales.prod_id = products.prod_id AND
      sales.cust_id = customers.cust_id AND
      customers.country_id = countries.country_id
GROUP BY country_name, prod_name, calendar_year;

CREATE MATERIALIZED VIEW my_model_mv
ENABLE QUERY REWRITE AS
SELECT country, prod, year, sale, cnt
FROM my_groupby_mv
MODEL PARTITION BY(country) DIMENSION BY(prod, year)
   MEASURES(sale s) IGNORE NAV
(s['Shorts', 2000] =  0.2 * AVG(s)[CV(), year BETWEEN 1996 AND 1999],
s['Kids Pajama', 2000] = 0.5 * AVG(s)[CV(), year BETWEEN 1995 AND 1999],
s['Boys Pajama', 2000] = 0.6 * AVG(s)[CV(), year BETWEEN 1994 AND 1999],
...
<hundreds of other update rules>);
```

2つのマテリアライズド・ビューを使用することにより、マテリアライズド・ビューmy_groupby_mvを段階的にメンテナンスできるようになります。マテリアライズド・ビューmy_model_mvは、より少量のデータセットに基づいているので(my_groupby_mvをベースに作成されているため)、完全リフレッシュでメンテナンスできます。

モデルを含むマテリアライズド・ビューでは、完全リフレッシュまたはPCTリフレッシュのみを使用できます。また、テキストの部分一致のクエリー・リライトのみを使用できます。

関連項目:

モデル計算の詳細は、モデリングのSQLを参照してください。

6.4 マテリアライズド・ビューのセキュリティ問題について

独自のスキーマにマテリアライズド・ビューを作成するには、CREATE MATERIALIZED VIEW権限と、別のスキーマにある参照先の表に対するSELECTまたはREAD権限が必要です。別のスキーマにマテリアライズド・ビューを作成するには、CREATE ANY MATERIALIZED VIEW権限が必要です。また、参照先の表が別のスキーマにある場合、マテリアライズド・ビューの所有者には、その表に対するSELECTまたはREAD権限が必要です。さらに、独自のスキーマに含まれない表を参照するマテリアライズド・ビューでクエリー・リライトを有効にするには、GLOBAL QUERY REWRITE権限、または独自のスキーマに含まれない各表に対するQUERY REWRITEオブジェクト権限が必要です。

マテリアライズド・ビューがビルトインされたコンテナ上にあり、作成者が所有者とは異なる場合は、コンテナ表に対するREAD WITH GRANTまたはSELECT WITH GRANT権限が必要です。

マテリアライズド・ビューを作成するときに、必要なすべての権限が付与されていると考えられても、権限エラーが継続して発生する場合は、権限が明示的に付与されているのではなく、ロールから権限を継承しようとしている可能性があります。参照される表がそれぞれ異なるスキーマにある場合、マテリアライズド・ビューの所有者には、これらの表へのSELECTまたはREAD権限が明示的に付与されている必要があります。

ON COMMIT REFRESHを指定してマテリアライズド・ビューを作成する場合に、定義問合せにある表が所有者のスキーマに含まれないときは、マテリアライズド・ビューの所有者には追加の権限が必要です。その場合、所有者には、ON COMMIT REFRESHシステム権限、または所有者のスキーマに含まれない各表へのON COMMIT REFRESHオブジェクト権限が必要です。

関連項目:

仮想プライベート・データベース(VPD)を含むマテリアライズド・ビューの問合せ

6.4.1 仮想プライベート・データベース(VPD)を含むマテリアライズド・ビューの問合せ

セキュリティ上の問題はありますが、マテリアライズド・ビューを直接的に問い合せている場合、そのマテリアライズド・ビューは、たまたまマテリアライズド・ビュー化しているビューとして機能します。ビューまたはマテリアライズド・ビューを作成する際、その所有者は、作成しているビューまたはマテリアライズド・ビューの基礎となるベース関係にアクセスするための権限を所有している必要があります。この権限を持つ所有者は、他のユーザーがアクセス可能なビューまたはマテリアライズド・ビューを公開できます(そのビューまたはマテリアライズド・ビューへのアクセス権限がユーザーに付与されている場合)。

仮想プライベート・データベースを含むマテリアライズド・ビューを使用する場合も同様です。マテリアライズド・ビューを作成する際、そのマテリアライズド・ビューの所有者には、マテリアライズド・ビューのベース関係に対して有効になっているVPDポリシーが存在しないようにする必要があります。VPDポリシーが存在する場合、マテリアライズド・ビューを作成するときにUSING TRUSTED CONSTRAINTS句を使用する必要があります。マテリアライズド・ビューの所有者は、新しいマテリアライズド・ビューにVPDポリシーを設定できます。マテリアライズド・ビューにアクセスするユーザーは、そのマテリアライズド・ビューに設定されているVPDポリシーの対象になります。ただし、これに加えて、マテリアライズド・ビューの基礎となるベース関係のVPDポリシーの対象になることはありません。基礎となるベース関係のセキュリティ処理は、マテリアライズド・ビューの所有者に対して行われます。

この項では、次の項目について説明します。

- 仮想プライベート・データベースを含むクエリー・リライトの使用
- マテリアライズド・ビューおよび仮想プライベート・データベースに関する制限

6.4.1.1 仮想プライベート・データベースを含むクエリー・リライトの使用

クエリー・リライトを使用してマテリアライズド・ビューにアクセスする場合、マテリアライズド・ビューは索引のようなアクセス構造体として機能します。つまり、この方法でアクセスされるマテリアライズド・ビューのセキュリティ処理は索引のセキュリティ処理とほとんど同じであり、すべてのセキュリティ・チェックは問合せで指定されている関係に対して実行されるということです。索引やマテリアライズド・ビューを使用するのは、データへのアクセス速度を向上させるためであり、追加のセキュリティ・チェックを提供するためではありません。索引やマテリアライズド・ビューの存在によって、セキュリティ・チェックが追加されることはありません。

これは、VPDが存在する状態でクエリー・リライトを使用してマテリアライズド・ビューにアクセスする場合にも当てはまります。この問合せは、問合せの中で指定されている関係に設定されたすべてのVPDポリシーの対象になります。クエリー・リライトは、ディテール関係にアクセスするのではなくマテリアライズド・ビューを使用するように問合せをリライトしますが、これは、リライトが行われなかったとしてもまったく同じ行が戻されると保証できる場合にかぎられます。つまり、クエリー・リライトは、問合せの中で指定されている関係に対するすべてのVPDポリシーをそのまま尊重する必要があるということです。ただし、マテリアライズド・ビュー自体に対するVPDポリシーは、クエリー・リライトを使用してそのマテリアライズド・ビューにアクセスする場合には影響を及ぼしません。これは、問合せの中で指定されている関係に対するVPDポリシーによって、データはすでに保護されているためです。

6.4.1.2 マテリアライズド・ビューおよび仮想プライベート・データベースに関する制限

アクティブなVPDポリシーを持つ関係が問合せに含まれている場合、クエリー・リライトは、テキストの完全一致モードおよび部分一致モードを使用しません。ただし、一般的なリライト手法は使用します。これは、VPDが、VPDポリシーに影響を与えるように問合せを透過的に変換するためです。VPDポリシーを伴う問合せに対してクエリー・リライトがテキスト一致の変換を行うとすると、それはVPDポリシーを否定することになります。

また、マテリアライズド・ビューを作成またはリフレッシュする際、そのマテリアライズド・ビューの所有者は、マテリアライズド・ビューのベース関係に対して有効になっているアクティブなVPDポリシーが存在しないようにする必要があります。このようにしない場合、エラーが戻されます。マテリアライズド・ビューの所有者は、このようなVPDポリシーが存在しないようにするか、ポリシーがNULLを戻すようにする必要があります。これは、VPDがマテリアライズド・ビューの定義問合せを透過的に変更して、マテリアライズド・ビューに格納されている行セットと、マテリアライズド・ビュー定義で指定されている行セットが一致しないようにしてしまうためです。

この制限を回避しながら、必要なVPDが指定された行のサブセットを含むマテリアライズド・ビューを作成する方法の1つは、マテリアライズド・ビューのディテール関係に対するアクティブなVPDポリシーを持たないユーザー・アカウントで、マテリアライズド・ビューを作成することです。また、マテリアライズド・ビューのWHERE句に、VPDポリシーの効果を盛り込んだ条件を指定することもできます。クエリー・リライトは、そのVPDポリシーを含む問合せをリライトする際、問合せに含まれるVPD生成の条件と、マテリアライズド・ビューの作成時に直接指定された条件とを照合します。

6.5 マテリアライズド・ビューの無効化

マテリアライズド・ビューに関連する依存性は、正しい操作が保証されるように、自動的にメンテナンスされます。あるマテリアライズド・ビューが作成されると、マテリアライズド・ビューはその定義で参照したディテール表に依存します。マテリアライズド・ビューの依存する表に対してINSERT、DELETE、UPDATEなどのDML操作またはDDL操作が行われると、そのマテリアライズド・ビューは無効になります。マテリアライズド・ビューを再検証するには、ALTER MATERIALIZED VIEW COMPILE文を使用します。

マテリアライズド・ビューは、参照されたときに自動的に再検証されます。多くの場合、マテリアライズド・ビューは、正常および透過的に再検証されます。ただし、マテリアライズド・ビューが参照している表の列が削除された場合、またはクエリー・リライト権限を持っていなかったマテリアライズド・ビューの所有者に新たに権限が付与された場合は、次の文を使用してマテリアライズド・ビューを再検証する必要があります。

```
ALTER MATERIALIZED VIEW mview_name COMPILE;
```

マテリアライズド・ビューの状態は、データ・ディクショナリ・ビューUSER_MVIEWSまたはALL_MVIEWSを問い合せることでチェックできます。STALENESS列に、そのマテリアライズド・ビューが使用可能かどうかを示すFRESH、STALE、UNUSABLE、UNKNOWN、UNDEFINEDまたはNEEDS_COMPILEのいずれかの値が示されます。状態は自動的にメンテナンスされます。ただし、マテリアライズド・ビューのSTALENESS列がNEEDS_COMPILEとマークされている場合は、ALTER MATERIALIZED VIEW ... COMPILE文を発行してマテリアライズド・ビューを検証し、正しい失効状態にできます。マテリアライズド・ビューの状態がUNUSABLEの場合は、完全リフレッシュを実行して、マテリアライズド・ビューの状態をFRESHに戻す必要があります。リフレッシュしない事前作成表を基にしたマテリアライズド・ビューについては、いったん削除した後、再度作成する必要があります。リモート・マテリアライズド・ビューの失効は追跡されません。そのため、リモート・マテリアライズド・ビューをリライトに使用する場合、これらのビューは信頼できるとみなされます。

6.6 マテリアライズド・ビューの変更

マテリアライズド・ビューでは、次の変更が可能です。

- リフレッシュ・オプション(FAST/FORCE/COMPLETE/NEVER)の変更
- リフレッシュ・モード(ON COMMIT/ON DEMAND)の変更
- 再コンパイルによる変更
- クエリー・リライトに対する使用可能/使用禁止
- CONSIDER FRESH句による変更
- パーティションのメンテナンス操作
- 問合せ時計算の有効化

この他の変更は、すべてマテリアライズド・ビューを削除し再作成することで可能になります。変更操作の成功は、変更の要件が満たされているかどうかによって決まります。たとえば、マテリアライズド・ビュー・ログがすべての実表に存在する場合、高速リフレッシュは成功します。

マテリアライズド・ビューが無効化されている場合は、ALTER MATERIALIZED VIEW文のCOMPILE句を使用できます。このコンパイル処理は高速であり、マテリアライズド・ビューがクエリー・リライトに再使用できるようになります。

関連項目:

- ALTER MATERIALIZED VIEW文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- マテリアライズド・ビューの無効化

6.7 リアルタイムのマテリアライズド・ビューの使用

リアルタイムのマテリアライズド・ビューでは、マテリアライズド・ビューが失効とマークされていても、ユーザー問合せに対して最新のデータが提供されます。

関連項目:

- リアルタイムのマテリアライズド・ビューの概要
- リアルタイムのマテリアライズド・ビューの作成

- リアルタイムのマテリアライズド・ビューへの既存のマテリアライズド・ビューの変換
- リアルタイムのマテリアライズド・ビューを使用するためのクエリー・リライトの有効化
- クエリー・リライト時におけるリアルタイムのマテリアライズド・ビューの使用
- 直接問合せアクセスでのリアルタイムのマテリアライズド・ビューの使用
- リアルタイムのマテリアライズド・ビューのリスト
- リアルタイムのマテリアライズド・ビューのパフォーマンスの向上

6.7.1 リアルタイムのマテリアライズド・ビューの概要

リアルタイムのマテリアライズド・ビューはマテリアライズド・ビューの一種で、データが変更されたためにマテリアライズド・ビューがその実表と同期されていなくても、ユーザー問合せに対して最新のデータを提供します。

SQLセッションが失効許可モードに設定されている場合を除き、失効とマークされているマテリアライズド・ビューをクエリー・リライトで使用することはできません。通常、リアルタイムのデータを必要とする組織では、マテリアライズド・ビューを更新して、実表に加えられた変更を確実に反映するために、ON COMMITリフレッシュ・モードを使用します。ただし、実表に対するDMLの変更が大規模で、非常に頻繁に行われる場合、このモードではリソースの競合が発生し、リフレッシュのパフォーマンスが低下することがあります。リアルタイムのマテリアライズド・ビューは、データをその場で再計算することにより、失効したマテリアライズド・ビューから最新のデータを取得するための軽量のソリューションを提供します。

リアルタイムのマテリアライズド・ビューでは、ログ・ベースやPCTベースのリフレッシュなど、使用可能な任意のホーム外リフレッシュ方法を使用できます。これらはオンデマンド・リフレッシュまたはスケジュールされた自動リフレッシュで使用できますが、ON COMMIT句を使用して指定された自動リフレッシュで使用することはできません。

リアルタイムのマテリアライズド・ビューの利点

- マテリアライズド・ビューの可用性が向上します。
- 失効している可能性があるマテリアライズド・ビューにアクセスするユーザー問合せに対して最新のデータを提供します。

リアルタイムのマテリアライズド・ビューの仕組み

リアルタイムのマテリアライズド・ビューは、問合せ時計算と呼ばれる手法を使用して、失効したマテリアライズド・ビューに最新のデータを提供します。問合せがリアルタイムのマテリアライズド・ビューにアクセスすると、Oracle Databaseではまず、リアルタイムのマテリアライズド・ビューが失効とマークされているかどうかが確認されます。失効していなければ、リアルタイムのマテリアライズド・ビューをそのまま使用して、必要なデータが提供されます。リアルタイムのマテリアライズド・ビューが失効とマークされている場合は、問合せ時計算手法を使用して最新のデータが生成され、適切な問合せ結果が返されます。

リアルタイムのマテリアライズド・ビューは、失効したマテリアライズド・ビューに最新のデータを提供する際にログ・ベースのリフレッシュに似た手法を使用します。変更ログに記録された変更内容と既存のデータを組み合せて、最新のデータを取得します。ただし、ログ・ベースのリフレッシュとは異なり、リアルタイムのマテリアライズド・ビューは、リアルタイムのマテリアライズド・ビュー内のデータを更新するためにマテリアライズド・ビュー・ログを使用しません。かわりに、失効したリアルタイムのマテリアライズド・ビューに問合せがアクセスすると、問合せ時計算を使用して再計算されたデータを直接使用して、問合せに応答します。

リアルタイムのマテリアライズド・ビューを作成するには、マテリアライズド・ビューの定義でON QUERY COMPUTATION句を使用します。

6.7.1.1 リアルタイムのマテリアライズド・ビューの使用に関する制限

リアルタイムのマテリアライズド・ビューを使用する際には、一定の制限が適用されます。

- 次の場合、リアルタイムのマテリアライズド・ビューを使用することはできません。

- 実表に作成された1つ以上のマテリアライズド・ビュー・ログが使用できないか、存在しない。
- 変更シナリオでホーム外のログ・ベースまたはPCTリフレッシュを実行できない。
- ON COMMIT句を使用して自動リフレッシュが指定されている。

- リアルタイムのマテリアライズド・ビューが、1つ以上のベース・マテリアライズド・ビューの上位に定義されているネステッド・マテリアライズド・ビューである場合、すべてのベース・マテリアライズド・ビューが最新である場合にのみ、クエリー・リライトが発生します。1つ以上のベース・マテリアライズド・ビューが失効している場合、クエリー・リライトは、このリアルタイムのマテリアライズド・ビューを使用して実行されません。

リアルタイムのマテリアライズド・ビューに直接アクセスする問合せのカーソルは共有されません。

6.7.1.2 リアルタイムのマテリアライズド・ビューへのアクセスについて

マテリアライズド・ビューの場合と同様に、リアルタイムのマテリアライズド・ビューに保存されているデータにアクセスするには、複数の方法があります。

リアルタイムのマテリアライズド・ビューに保存されているデータには、次のいずれかの方法でアクセスできます。

- クエリー・リライト

  リアルタイムのマテリアライズド・ビューの定義に似たユーザー問合せは、リアルタイムのマテリアライズド・ビューを使用するようにリライトされます。

- リアルタイムのマテリアライズド・ビューの直接アクセス

  ユーザー問合せは、名前を使用して、リアルタイムのマテリアライズド・ビューを直接参照します。

どちらのシナリオでも、リアルタイムのマテリアライズド・ビューの内容に失効データとしてアクセスしたり、適切な結果の問合せ時計算をトリガーできます。問合せ時計算がトリガーされるかどうかは、環境と実際のSQL文によって決まります。

EXPLAIN PLAN文の出力には、特定のユーザー問合せで問合せ時計算が使用されたかどうかを示すメッセージが表示されます。

関連項目:

- 直接問合せアクセスでのリアルタイムのマテリアライズド・ビューの使用
- クエリー・リライト時におけるリアルタイムのマテリアライズド・ビューの使用

6.7.2 リアルタイムのマテリアライズド・ビューの作成

リアルタイムのマテリアライズド・ビューを作成するには、CREATE MATERIALIZED VIEW文でON QUERY COMPUTATION句を使用します。

リアルタイムのマテリアライズド・ビューは、すべての変更シナリオにおいて問合せ時計算に適用できるとはかぎらなくても作成可能です。リアルタイムのマテリアライズド・ビューを作成するための最低限の要件は、INSERT操作のホーム外リフレッシュをサポートしていることです。複合DML操作など、その他の変更シナリオが発生した場合、すべてのタイプのリアルタイム・マテリアライズド・ビューで問合せ時計算を実行できるとはかぎりません。

リアルタイムのマテリアライズド・ビューでは、ログ・ベースのホーム外リフレッシュ・メカニズム(PCTリフレッシュを含む)を使用する必要があります。リアルタイムのマテリアライズド・ビューでON COMMITリフレッシュ・モードを使用することはできません。

リアルタイムのマテリアライズド・ビューを作成するには:

1. リアルタイムのマテリアライズド・ビューのすべての実表にマテリアライズド・ビュー・ログが存在することを確認します。
2. リアルタイムのマテリアライズド・ビューの基になるすべての表についてマテリアライズド・ビュー・ログを作成します。
3. CREATE MATERIALIZED VIEW文でENABLE ON QUERY COMPUTATION句を指定して、リアルタイムのマテリアライズド・ビューを作成します。

例6-7 リアルタイムのマテリアライズド・ビューの作成

この例では、SHスキーマ内のSALESおよびPRODUCTS表から集計されたデータに基づくSUM_SALES_RTMVというリアルタイムのマテリアライズド・ビューを作成します。リアルタイムのマテリアライズド・ビューを作成する前に、必要な前提条件を満たしていることを確認してください。

1. 実表SALESおよびPRODUCTSにマテリアライズド・ビュー・ログを作成します。

   次のコマンドでは、SALES表にマテリアライズド・ビュー・ログが作成されます。

   ```
   CREATE MATERIALIZED VIEW LOG ON sales
   WITH SEQUENCE, ROWID
   (prod_id, quantity_sold, amount_sold)
   INCLUDING NEW VALUES;
   ```

   次のコマンドでは、PRODUCTS表にマテリアライズド・ビュー・ログが作成されます。

   ```
   CREATE MATERIALIZED VIEW LOG ON products
   WITH ROWID
   (prod_id, prod_name, prod_category, prod_subcategory)
   INCLUDING NEW VALUES;
   ```

2. CREATE MATERIALIZED VIEW文でON QUERY COMPUTATION句を指定して、リアルタイムのマテリアライズド・ビューを作成します。このリアルタイムのマテリアライズド・ビューでは高速リフレッシュ方法が使用され、ENABLE QUERY REWRITE句は、クエリー・リライトが有効である必要があることを示しています。

   ```
   CREATE MATERIALIZED VIEW sum_sales_rtmv
   REFRESH FAST ON DEMAND
   ENABLE QUERY REWRITE
   ENABLE ON QUERY COMPUTATION
   AS
   SELECT prod_name, SUM(quantity_sold) AS sum_qty, COUNT(quantity_sold) AS cnt_qty,
   SUM(amount_sold) AS sum_amt,
        COUNT(amount_sold) AS cnt_amt, COUNT(*) AS cnt_star
   FROM sales, products
   WHERE sales.prod_id = products.prod_id
   GROUP BY prod_name;
   ```

SUM_SALES_RTMVリアルタイム・マテリアライズド・ビューを作成した後、次の問合せを実行するとします。

```
SELECT prod_name, SUM(quantity_sold), SUM(amount_sold)
FROM sales, products
WHERE sales.prod_id = products.prod_id
GROUP BY prod_name;
```

SUM_SALES_RTMVが失効していない場合、このリアルタイムのマテリアライズド・ビューに保存されているデータを使用して、問合せ結果が返されます。ただし、SUM_SALES_RTMVが失効しており、問合せをリライトして問合せ時計算でこのマテリアライズド・ビューを使用するコストが実表アクセスより低い場合は、SALESおよびPRODUCTS表のマテリアライズド・ビュー・ログ内のデルタ変更をリアルタイムのマテリアライズド・ビューSUM_SALES_RTMV内のデータと組み合せることにより、問合せに応答します。

6.7.3 リアルタイムのマテリアライズド・ビューへの既存のマテリアライズド・ビューの変換

リアルタイムのマテリアライズド・ビューの前提条件を満たしている場合、定義を変更し、問合せ時計算を有効にすることにより、既存のマテリアライズド・ビューをリアルタイムのマテリアライズド・ビューに変換できます。

マテリアライズド・ビューをリアルタイムのマテリアライズド・ビューに変換するには:

- マテリアライズド・ビューの定義を変更し、ALTER MATERIALIZED VIEW文でON QUERY COMPUTATION句を使用して問合せ時計算を有効にします。

リアルタイムのマテリアライズド・ビューを通常のマテリアライズド・ビューに変換するには、ALTER MATERIALIZED VIEW文でDISABLE ON QUERY COMPUTATION句を使用して問合せ時計算を無効にします。

例6-8 リアルタイムのマテリアライズド・ビューへのマテリアライズド・ビューの変換

マテリアライズド・ビューSALES_RTMVは、SALES、TIMESおよびPRODUCTSの各表に基づいており、高速リフレッシュを使用します。3つの実表すべてにマテリアライズド・ビュー・ログが存在します。このマテリアライズド・ビューを変更し、リアルタイムのマテリアライズド・ビューに変換することにします。

1. マテリアライズド・ビューの定義を変更し、ON QUERY COMPUTATION句を指定してリアルタイムのマテリアライズド・ビューに変換します。

```
ALTER MATERIALIZED VIEW sales_rtmv ENABLE ON QUERY COMPUTATION;
```

2. DBA_MVIEWSビューに問い合せて、SALES_RTMVで問合せ時計算が有効になっているかどうかを確認します。

```
SELECT mview_name, on_query_computation
FROM dba_mviews
WHERE mview_name = 'SALES_RTMV';
```

6.7.4 リアルタイムのマテリアライズド・ビューを使用するためのクエリー・リライトの有効化

クエリー・リライト・メカニズムによりユーザー問合せをリライトしてリアルタイムのマテリアライズド・ビューを使用するには、リアルタイムのマテリアライズド・ビューでクエリー・リライトを有効にする必要があります。

リアルタイムのマテリアライズド・ビューの作成時にクエリー・リライトを有効にすることも、リアルタイムのマテリアライズド・ビューの定義を変更することにより、後から有効にすることもできます。クエリー・リライトを有効にするには、ENABLE QUERY REWRITE句を使用します。

既存のリアルタイムのマテリアライズド・ビューでクエリー・リライトを有効にするには:

- ALTER MATERIALIZED VIEWコマンドを実行し、ENABLE QUERY REWRITE句を指定します。

例6-9 リアルタイムのマテリアライズド・ビューでのクエリー・リライトの有効化

リアルタイムのマテリアライズド・ビューmy_rtmvは、高速リフレッシュ・メカニズムを使用します。このリアルタイムのマテリアライズド・ビューの定義を変更し、クエリー・リライト・メカニズムで問合せをリライトする際に、このリアルタイムのマテリアライズド・ビューを考慮するように指定することにします。

次のコマンドでは、my_rtmvでクエリー・リライトが有効化されます。

```
ALTER MATERIALIZED VIEW my_rtmv ENABLE QUERY REWRITE;
```

6.7.5 クエリー・リライト時におけるリアルタイムのマテリアライズド・ビューの使用

リアルタイムのマテリアライズド・ビューでクエリー・リライトが有効になっている場合、リアルタイムのマテリアライズド・ビューが失効していても、クエリー・リライトでは、リアルタイムのマテリアライズド・ビューを使用してユーザー問合せに対して結果を提供できます。

ネストされたリアルタイムのマテリアライズド・ビューがクエリー・リライトの対象となるのは、そのベースとなるリアルタイムのマテリアライズド・ビューがすべて最新である場合のみです。

ユーザー問合せが実行されると、クエリー・リライトはまず、必要なデータを提供するために最新のマテリアライズド・ビューを使用できるかどうかを確認します。適切なマテリアライズド・ビューがない場合、クエリー・リライトは、ユーザー問合せをリライトするために使用できるリアルタイムのマテリアライズド・ビューを探します。リアルタイムのマテリアライズド・ビューでは最新のデータを計算する際になんらかのオーバーヘッドが発生するため、リアルタイムのマテリアライズド・ビューより最新のマテリアライズド・ビューが優先されます。次に、コスト・ベースのオプティマイザによって、問合せ時計算を使用するSQL問合せのコストが特定され、そのユーザー問合せに応答するためにリアルタイムのマテリアライズド・ビューを使用するかどうかが決定されます。

現在のSQLセッションのQUERY_REWRITE_INTEGRITYモードがSTALE_TOLERATEDに設定されている場合、クエリー・リライトの際に問合せ時計算は使用されません。STALE_TOLERATEDリライト・モードは、問合せに対応するために最新の結果が必要でないため、問合せ時計算が不要であることを示します。

クエリー・リライトでリアルタイムのマテリアライズド・ビューを使用するには:

1. QUERY_REWRITE_INTEGRITYがENFORCEDまたはTRUSTEDモードに設定されていることを確認します。QUERY_REWRITE_INTEGRITYモードをSTALE_TOLERATEDモードに設定しないでください。
2. リアルタイムのマテリアライズド・ビューを定義するために使用されたSQL問合せと一致するユーザー問合せを実行します。

   リアルタイムのマテリアライズド・ビューを利用するようにリライトできる問合せでは、問合せ時計算でリアルタイムのマテリアライズド・ビューが使用されます。

   リアルタイムのマテリアライズド・ビューを使用して問合せがリライトされたことを確認するには、EXPLAIN PLANを使用します。

例6-10 クエリー・リライト時におけるリアルタイムのマテリアライズド・ビューの使用

この例では、クエリー・リライトを有効にしてリアルタイムのマテリアライズド・ビューを作成し、ユーザー問合せに対してデータを提供するために、それがクエリー・リライトで使用されたことを確認します。

1. 作成するリアルタイムのマテリアライズド・ビューの実表であるSALES表にマテリアライズド・ビュー・ログを作成します。
2. クエリー・リライトを有効にして、リアルタイムのマテリアライズド・ビューmav_sum_salesを作成します。

```
CREATE MATERIALIZED VIEW mav_sum_sales
REFRESH FAST ON DEMAND
ENABLE ON QUERY COMPUTATION
ENABLE QUERY REWRITE
AS
SELECT prod_id, sum(quantity_sold) as sum_qty, count(quantity_sold) as cnt_qty,
       sum(amount_sold) sum_amt, count(amount_sold) cnt_amt, count(*) as cnt_star
FROM sales
GROUP BY prod_id;
```

3. 次の問合せを実行します。

```
SELECT prod_id, sum(quantity_sold), sum(amount_sold)
FROM  sales
WHERE prod_id < 1000
GROUP BY prod_id;
```

   この問合せが、リアルタイムのマテリアライズド・ビューmav_sum_salesを定義するために使用されたものに似ている点に注目してください。この問合せに似た定義を持つ他のマテリアライズド・ビューが存在しないため、クエリー・リライトでは、mav_sum_salesリアルタイム・マテリアライズド・ビューを使用して問合せ結果を特定できます。クエリー・リライトが行われたことを確認するには、SQLカーソル・キャッシュをチェックするか(たとえばDBMS_XPLANを使用)、SQLモニターを使用す

るか、またはEXPLAIN PLANを使用します。

mav_sum_salesを使用する、内部でリライトされた問合せは、次のような文になります。

```
SELECT prod_id, sum_qty, sum_amt
FROM  mav_sum_sales
WHERE prod_id < 1000;
```

4. 問合せ結果を提供するためにリアルタイムのマテリアライズド・ビューが使用されたことを確認します。EXPLAIN PLAN文を使用して、問合せの実行計画を表示します。

   次の実行計画は、リアルタイムのマテリアライズド・ビューへの直接アクセスを示しています。マテリアライズド・ビューが失効している場合、実行計画はさらに複雑になり、未処理のDML操作に応じて、他のオブジェクト(マテリアライズド・ビュー・ログなど)へのアクセスも含まれます。

```
EXPLAIN PLAN for SELECT prod_id, sum(quantity_sold), sum(amount_sold) FROM sales WHERE prod_id
< 1000 GROUP BY prod_id;
SELECT plan_table_output FROM table(dbms_xplan.display('plan_table',null,'serial'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
-
Plan hash value: 13616844
------------------------------------------------------------------------------------------------
-
| Id  | Operation             | Name          | Rows  | Bytes | Cost (%CPU) | Time      |
------------------------------------------------------------------------------------------------
-
| 0   | SELECT STATEMENT      |               | 92    | 3588  | 3  (0)      | 00:00:01  |
| *1  | MAT_VIEW ACCESS FULL  | MAV_SUM_SALES | 92    | 3588  | 3  (0)      | 00:00:01  |
------------------------------------------------------------------------------------------------
-
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("PROD_ID"<1000)

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)

17 rows selected.
```

6.7.6 直接問合せアクセスでのリアルタイムのマテリアライズド・ビューの使用

リアルタイムのマテリアライズド・ビューの名前を問合せで参照することにより、リアルタイムのマテリアライズド・ビューに直接アクセスできます。

ユーザー問合せで指定されたリアルタイムのマテリアライズド・ビューが最新である場合、必要なデータがリアルタイムのマテリアライズド・ビューから直接フェッチされます。リアルタイムのマテリアライズド・ビューが失効している場合は、FRESH_MVヒントを使用して問合せ時計算を実行し、最新のデータを取得する必要があります。Oracle Databaseでは、ユーザー問合せで直接アクセスされるリアルタイムのマテリアライズド・ビューについては問合せ時計算が自動的に実行されることはありません。

リアルタイムのマテリアライズド・ビューに直接アクセスする際に、失効したリアルタイムのマテリアライズド・ビューから最新のデータを取得するには:

- ユーザー問合せでFRESH_MVヒントを使用して、問合せ時計算を実行する必要があることを示します。

例6-11 リアルタイムのマテリアライズド・ビューの作成と問合せでの使用

この例では、SALES_NEW表に基づくリアルタイムのマテリアライズド・ビューMY_RTMVを作成します。SALES_NEW表は、SH.SALES表のコピーとして作成されています。リアルタイムのマテリアライズド・ビューを作成した後、実表に1行を挿入します。次に、ユーザー問合せでマテリアライズド・ビュー名を使用することにより、fresh_mvヒントを使用してリアルタイムのマテリアライズド・ビューから最新のデータにアクセスします。

1. 実表sales_newにマテリアライズド・ビュー・ログを作成します。

   リアルタイムのマテリアライズド・ビューを作成する場合、実表のマテリアライズド・ビュー・ログは必須です。

```
CREATE MATERIALIZED VIEW LOG on sales_new
WITH sequence, ROWID (prod_id, cust_id, time_id, channel_id, promo_id, quantity_sold,
amount_sold)
INCLUDING NEW VALUES;
```

2. sales_newを実表とする、my_rtmvというリアルタイムのマテリアライズド・ビューを作成します。

   ON QUERY COMPUTATION句は、リアルタイムのマテリアライズド・ビューを作成することを示しています。指定されているリフレッシュ・モードは、ログ・ベースの高速リフレッシュです。このリアルタイムのマテリアライズド・ビューでは、クエリー・リライトが有効です。

```
CREATE MATERIALIZED VIEW my_rtmv
REFRESH FAST
ENABLE ON QUERY COMPUTATION
ENABLE QUERY REWRITE
AS
SELECT prod_id, cust_id, channel_id, sum(quantity_sold) sum_q, count(quantity_sold) cnt_q,
avg(quantity_sold) avg_q,
               sum(amount_sold) sum_a, count(amount_sold) cnt_a, avg(amount_sold) avg_a
FROM sales_new
GROUP BY prod_id, cust_id, channel_id;
```

3. リアルタイムのマテリアライズド・ビューの実表であるsales_newに1行を挿入し、この変更をコミットします。

```
INSERT INTO sales_new (prod_id, cust_id, time_id, channel_id, promo_id, quantity_sold,
amount_sold)
        VALUES (116,100450, sysdate,9,9999,10,350);

COMMIT;
```

4. リアルタイムのマテリアライズド・ビューに直接問い合せて、前のステップでリアルタイムのマテリアライズド・ビューの実表に追加された行のデータを表示します。

```
SELECT * from my_rtmv
WHERE prod_id = 116 AND cust_id=100450 AND channel_id = 9;

PROD_ID   CUST_ID   CHANNEL_ID   SUM_Q   CNT_Q    AVG_Q    SUM_A    CNT_A    AVG_A
-------   -------   ----------   -----   -----    -----    -----    -----    ------
116       100450    9            1       1        1        11.99    1        11.99
```

   問合せ結果には、このデータの更新された値が表示されていません。これは、リアルタイムのマテリアライズド・ビューがまだリフレッシュされておらず、その実表に加えられた変更内容が反映されていないためです。

5. リアルタイムのマテリアライズド・ビューに問い合せる際にFRESH_MVヒントを指定して、実表で更新された行を表示します。

```
SELECT /*+ fresh_mv */ * FROM my_rtmv
WHERE prod_id = 116 AND cust_id=100450 AND channel_id = 9;

PROD_ID   CUST_ID   CHANNEL_ID  SUM_Q   CNT_Q   AVG_Q    SUM_A    CNT_A    AVG_A
```

```
-------  -------  ----------  -----  -----  -----  -----  -----  ------
116      100450   9           11     2      5.5    361.99 2      180.995
```

ここでは、更新された行が表示されています。これは、FRESH_MVヒントによって、リアルタイムのマテリアライズド・ビューの問合せ時計算がトリガーされ、最新のデータが再計算されるためです。

6.7.7 リアルタイムのマテリアライズド・ビューのリスト

データ・ディクショナリ・ビューALL_MVIEWS、DBA_MVIEWSおよびUSER_MVIEWSのON_QUERY_COMPUTATION列は、マテリアライズド・ビューがリアルタイムのマテリアライズド・ビューであるかどうかを示します。

ON_QUERY_COMPUTATION列の値Yは、リアルタイムのマテリアライズド・ビューを示します。

ユーザー・スキーマ内のリアルタイムのマテリアライズド・ビューをすべてリストするには:

- USER_MVIEWSビューに問い合せ、ON_QUERY_COMPUTATION列がYに設定されているマテリアライズド・ビューの詳細を表示します。

例6-12 現在のユーザーのスキーマ内のリアルタイムのマテリアライズド・ビューのリスト

```
SELECT owner, mview_name, rewrite_enabled, staleness
FROM user_mviews
WHERE on_query_computation = 'Y';

OWNER   MVIEW_NAME        REWRITE_ENABLED   STALENESS
------  ------------    ------------------ ------------
SH      SALES_RTMV          N                FRESH
SH      MAV_SUM_SALES       Y                FRESH
SH      MY_SUM_SALES_RTMV   Y                FRESH
SH      NEW_SALES_RTMV      Y                STALE
```

6.7.8 リアルタイムのマテリアライズド・ビューのパフォーマンスの向上

リアルタイムのマテリアライズド・ビューを使用するユーザー問合せのパフォーマンスを向上させるには、特定のガイドラインに従います。

リアルタイムのマテリアライズド・ビューでは、次のガイドラインを使用します。

- リアルタイムのマテリアライズド・ビューを使用する可能性がある問合せのパフォーマンスを向上させるために、リアルタイムのマテリアライズド・ビューを頻繁にリフレッシュします。

  リアルタイムのマテリアライズド・ビューは、実表に対するデルタ変更を既存のマテリアライズド・ビュー・データと組み合せることで動作するため、計算されるデルタ変更が少なければ、問合せの応答時間が短縮されます。未処理のDML操作が多くなると、問合せ時計算が複雑になる(コストも高くなる)可能性があり、実表の直接アクセスの方が効率的になることもあります(クエリー・リライトの場合)。

- オプティマイザが問合せのコストを正確に特定できるように、実表、リアルタイムのマテリアライズド・ビューおよびマテリアライズド・ビュー・ログの統計を収集します。

  クエリー・リライトの場合、コスト・ベースのリライト・メカニズムでは、オプティマイザを使用して、リライトされた問合せを使用する必要があるかどうかが決定されます。オプティマイザは、統計を使用してコストを特定します。

7 マテリアライズド・ビューのリフレッシュ

この章では、データ・ウェアハウス環境でマテリアライズド・ビューを使用する際に良好なパフォーマンスおよびデータの整合性を維持する主要な要素である、マテリアライズド・ビューのリフレッシュ方法について説明します。

この章の内容は次のとおりです。

- マテリアライズド・ビューのリフレッシュについて
- マテリアライズド・ビューのリフレッシュのヒント
- パーティション表付きマテリアライズド・ビューの使用
- パーティション化によるデータ・ウェアハウス・リフレッシュの改善
- リフレッシュ中のDML操作の最適化

7.1 マテリアライズド・ビューのリフレッシュについて

データベースでは、実表の変更後にマテリアライズド・ビューをリフレッシュして、マテリアライズド・ビューのデータを管理します。

リフレッシュ操作の実行には、索引再構築のための一時領域が必要で、リフレッシュ操作そのものを実行するために追加の領域が必要な場合もあります。サイトによっては、マテリアライズド・ビューを同時にすべてリフレッシュすることが適切ではない場合もあります。ベースとなるディテール・データが更新されると、このデータを使用するすべてのマテリアライズド・ビューが失効します。したがって、マテリアライズド・ビューのリフレッシュを遅延させる場合は、選択したリライトの整合性レベルを信頼して、失効したマテリアライズド・ビューがクエリー・リライトに使用できるかどうかを判断するか、または、ALTER SYSTEM SET QUERY_REWRITE_ENABLED = FALSE文でクエリー・リライトを一時的に使用禁止にできます。マテリアライズド・ビューのリフレッシュ後に、ALTER SYSTEM SET QUERY_REWRITE_ENABLEDにTRUEを指定すると、現行のデータベース・インスタンスにあるすべてのセッションに対して、クエリー・リライトをデフォルトの状態である使用可能に戻すことができます。マテリアライズド・ビューをリフレッシュすると、そのすべての索引が自動的に更新されます。完全リフレッシュの場合は、リフレッシュ中にすべての索引を再作成できるように、一時ソート領域が必要です。これは、完全リフレッシュでは、新しいデータ・ボリューム全体が挿入される前に表が切り捨てられるか、削除されるためです。索引再作成のための一時領域が不足している場合は、リフレッシュ操作の前に明示的に各索引を削除するか、UNUSABLEマークを付ける必要があります。

マテリアライズド・ビューのリフレッシュのタイプについて

リフレッシュ方法は、増分または完全リフレッシュです。ログベースのリフレッシュおよびパーティション・チェンジ・トラッキング(PCT)リフレッシュと呼ばれる、2つの増分リフレッシュ方法があります。増分リフレッシュは一般にFASTリフレッシュと呼ばれ、通常は、完全リフレッシュよりも高速に実行されます。

完全リフレッシュは、マテリアライズド・ビューをBUILD IMMEDIATEで定義した場合に、最初に作成するときに実行されます。ただし、そのマテリアライズド・ビューが事前作成表を参照する場合、またはBUILD DEFERREDで定義された場合は除きます。ユーザーは、マテリアライズド・ビューを作成した後いつでも完全リフレッシュを実行できます。完全リフレッシュを実行すると、マテリアライズド・ビューを定義する問合せも実行されます。この処理は、特にデータベースが大量のデータを読み取って処理する必要がある場合に遅くなることがあります。

増分リフレッシュにより、マテリアライズ・ビューを最初から再作成する必要がなくなります。このように、変更のみを処理するため、リフレッシュ時間を大幅に削減できます。マテリアライズド・ビューは、必要時に、または定期的にリフレッシュできます。また、実表と同じデータベース内のマテリアライズド・ビューは、トランザクションによって実表の変更がコミットされるたびにリフレッシュできます。

ログベースの高速リフレッシュ方法を使用する場合、実表に対する変更はマテリアライズド・ビュー・ログまたはダイレクト・ローダー・ログに記録されます。マテリアライズド・ビュー・ログは、実表への変更を記録するスキーマ・オブジェクトであるため、実表で

定義されているマテリアライズド・ビューを増分リフレッシュできます。マテリアライズド・ビュー・ログは、それぞれ1つの実表に関連付けられています。マテリアライズド・ビュー・ログは、実表と同じデータベースおよびスキーマに格納されます。

PCTリフレッシュ方法は、変更された実表がパーティション化されている場合に使用でき、変更された実表のパーティションを使用して、マテリアライズド・ビュー内の影響を受けるパーティションまたはデータの部分を識別することができます。実表に対するパーティション・メンテナンス操作がある場合、増分リフレッシュにはこの方法しか使用できません。PCTリフレッシュを実行すると、影響を受けるマテリアライズド・ビューのパーティションまたは影響を受けるデータの部分にあるデータがすべて削除され、最初から再計算されます。

マテリアライズド・ビューのリフレッシュ・モードについて

マテリアライズド・ビューを作成する場合、リフレッシュをON DEMANDで行うか、ON COMMITで行うかを指定できます。

マテリアライズド・ビューで参照される表に対する挿入、更新または削除操作が、そのマテリアライズド・ビューのリフレッシュと同時に実行されると予想され、そのマテリアライズド・ビューに結合と集計が含まれている場合は、ON DEMAND高速リフレッシュではなくON COMMIT高速リフレッシュを使用することをお薦めします。

ON COMMITの場合は、トランザクションがコミットされるたびに、マテリアライズド・ビューが変更されます。このため、マテリアライズド・ビューに常に最新データが含まれていることが保証されます。または、ON DEMANDを指定すると、マテリアライズド・ビューのリフレッシュが発生する時期を制御できます。ON DEMANDマテリアライズド・ビューの場合、リフレッシュはDBMS_SYNC_REFRESHパッケージまたはDBMS_MVIEWパッケージで提供されるリフレッシュ方法で実行できます。

- DBMS_SYNC_REFRESHパッケージには、Oracle Database 12cリリース1で導入された新しいリフレッシュ方法である同期リフレッシュのAPIが含まれています。詳細は、同期リフレッシュを参照してください。
- DBMS_MVIEWパッケージに含まれるAPIの使用方法については、この章で説明します。リフレッシュ操作の基本タイプには、完全リフレッシュ、高速リフレッシュ、パーティション・チェンジ・トラッキング(PCT)リフレッシュの3つがあります。これらの基本タイプは、Oracle Database 12cリリース1で、ホーム外リフレッシュとよばれる新しいリフレッシュにより機能強化されています。

DBMS_MVIEWパッケージには、リフレッシュ操作を実行するための3つのAPIが含まれています。

- DBMS_MVIEW.REFRESH

  1つまたは複数のマテリアライズド・ビューをリフレッシュします。
- DBMS_MVIEW.REFRESH_ALL_MVIEWS

  すべてのマテリアライズド・ビューをリフレッシュします。
- DBMS_MVIEW.REFRESH_DEPENDENT

  特定のマスター表またはマテリアライズド・ビュー、あるいはマスター表やマテリアライズド・ビューのリストに依存するマテリアライズド・ビューをすべてリフレッシュします。

マテリアライズド・ビューのリフレッシュ方法

これらの各リフレッシュ・オプションに対して、リフレッシュの実行方法に関する2つの技法、つまり、ホーム内リフレッシュとホーム外リフレッシュがあります。ホーム内リフレッシュは、マテリアライズド・ビューに対して直接リフレッシュ文を実行します。ホーム外リフレッシュは1つ以上の外部表を作成し、その外部表に対してリフレッシュ文を実行し、次にマテリアライズド・ビューまたは影響を受けるマテリアライズド・ビューのパーティションをその外部表で切り替えます。ホーム内リフレッシュとホーム外リフレッシュは両方とも、特定のリフレッシュ・シナリオで優れたパフォーマンスを発揮します。ただし、ホーム外リフレッシュを使用すると、リフレッシュ時、特にリフレッシュ文を完了するのに時間がかかる場合に、マテリアライズド・ビューの高可用性が実現されます。

また、ホーム外リフレッシュ・メカニズムを採用して、同期リフレッシュと呼ばれる新しいリフレッシュ方法が、Oracle Database 12cリリース1で導入されています。これは、ファクト表およびそれらのマテリアライズド・ビューの両方が同じ方法でパーティション化されている場合、またはそれらのパーティションが機能依存性で関連付けられている場合の、データ・ウェアハウスの一般的な使

用シナリオを対象にしています。

このリフレッシュ・アプローチにより、一連の表と表に定義されているマテリアライズド・ビューが、常に同期した状態を維持できます。このリフレッシュ方法では、ユーザーは、実表の内容を直接変更するのではなく、整合性を確保するために、実表とマテリアライズド・ビューに同時にこれらの変更を適用する、同期リフレッシュ・パッケージによって提供されるAPIを使用する必要があります。同期リフレッシュ方法は、増分データのロードが厳密に制御され、定期的に発生するデータ・ウェアハウスには非常に適しています。

関連項目:

- ホーム外リフレッシュ・オプションについて
- キューブ・マテリアライズド・ビューのリフレッシュの詳細は、『Oracle OLAPユーザーズ・ガイド』を参照してください。

7.1.1 マテリアライズド・ビューの完全リフレッシュについて

完全リフレッシュは、マテリアライズド・ビューを最初にBUILD IMMEDIATEで定義した場合に実行されます。ただし、そのマテリアライズド・ビューが事前作成表を参照する場合は除きます。BUILD DEFERREDを使用するマテリアライズド・ビューの場合は、初めて使用する前に完全リフレッシュを実行する必要があります。完全リフレッシュは、どのマテリアライズド・ビューの場合も必要に応じて要求できます。このリフレッシュでは、ディテール表が読み込まれ、マテリアライズド・ビューの結果が計算されます。これは、読み込まれて処理されるデータが大量の場合には、時間がかかる処理です。そのため、完全リフレッシュを要求する前に、必ずその処理時間を考慮してください。

また、すでに作成済のマテリアライズド・ビューが次の項で説明する高速リフレッシュの条件を満たしていないために、完全リフレッシュしか使用できない場合もあります。

7.1.2 マテリアライズド・ビューの高速リフレッシュについて

ほとんどのデータ・ウェアハウスでは、そのディテール・データが定期的に増分更新されます。マテリアライズド・ビューのスキーマ・デザインについてで説明しているように、SQL*Loaderまたはバルクロードユーティリティーを使用して、ディテール・データの増分ロードを実行できます。通常、マテリアライズド・ビューの高速リフレッシュは効率的です。これは、すべてのマテリアライズド・ビューを再計算するのではなく、変更分のみが既存のデータに適用されるためです。このように、変更のみを処理するため、リフレッシュ時間を大幅に削減できます。

7.1.3 マテリアライズド・ビューのパーティション・チェンジ・トラッキング(PCT)リフレッシュについて

ディテール表に対するパーティション・メンテナンス操作がある場合、高速リフレッシュにはこの方法しか使用できません。マテリアライズド・ビューに対するPCTベースのリフレッシュは、パーティション・チェンジ・トラッキングについてで説明しているすべての条件を満たした場合にのみ有効です。

ディテール表に対するパーティション・メンテナンス操作がない場合、DBMS_MVIEWパッケージのプロシージャでリフレッシュのFASTメソッド(method => 'F')を要求すると、PCTリフレッシュが選択される前に、ログベースのルールの高速リフレッシュを試行するヒューリスティックなルールが使用されます。同様に、FORCEメソッド(method => '?')を要求すると、ログベースの高速リフレッシュ、PCTリフレッシュ、完全リフレッシュの試行順序に基づいてリフレッシュ方法が選択されます。または、PCTメソッド(method => 'P')を要求することもできます。Oracleは、すべてのPCT要件が満たされている場合はPCTメソッドを使用します。

マテリアライズド・ビューのパーティション化によるメリットで説明されている条件を満たしており、PCTリフレッシュ・プロセスを効率的に実行できる場合は、マテリアライズド・ビューに対してTRUNCATE PARTITIONを使用できます。

関連項目:

- パーティション・チェンジ・トラッキングの詳細は、パーティション・チェンジ・トラッキングについてを参照してください。

7.1.4 ホーム外リフレッシュ・オプションについて

Oracle Database 12cリリース1より、マテリアライズド・ビューのリフレッシュ・パフォーマンスおよび可用性を向上させる新しいリフレッシュ・オプションが利用可能となります。このリフレッシュ・オプションはホーム外リフレッシュと呼ばれます。これは、マテリアライズド・ビューのコンテナ表に直接変更を適用する既存のホーム内リフレッシュとは対照的に、リフレッシュの際に外部表を使用するためです。ホーム外リフレッシュ・オプションは、FAST ('F')、COMPLETE ('C')、PCT ('P')、FORCE ('?')など、既存のすべてのリフレッシュ方法とともに使用できます。ホーム外リフレッシュは特に、従来のDML文ではうまく対応できない大量のデータ変更を伴う状況を処理する際に効果的です。また、リフレッシュされるマテリアライズド・ビューは、リフレッシュ文の実行の際にダイレクト・アクセスおよびクエリー・リライトに使用できるため、非常に高い可用性を得ることができます。さらに、マテリアライズド・ビューのコンテナ表が徐々に断片化されたり、リフレッシュの中間結果が表示されるなどの問題の可能性を回避するのにも役立ちます。

ホーム外リフレッシュでは、マテリアライズド・ビューの全体または影響を受ける部分は、1つ以上の外部表に計算されます。パーティション化されたマテリアライズド・ビューの場合、パーティション・レベルのチェンジ・トラッキングが可能で、マテリアライズド・ビューにローカル索引が定義されていれば、ホーム外の方法では外部表にも同じローカル索引が構築されます。このリフレッシュ・プロセスは、マテリアライズド・ビューと外部表の切替え、または影響を受けるパーティションと外部表とのパーティション交換により行われます。切替えまたはパーティション交換操作中にクエリー・リライトがサポートされないことに注意してください。リフレッシュの間、外部表には効率的なダイレクト・ロードによって移入が行われます。

この項では、次の項目について説明します。

- ホーム外リフレッシュのタイプ
- ホーム外リフレッシュの制限および考慮事項

7.1.4.1 ホーム外リフレッシュのタイプ

ホーム外リフレッシュには次の3つのタイプがあります。

- ホーム外高速リフレッシュ

  これは、ホーム内高速リフレッシュより高い可用性を提供します。また、変更がマテリアライズド・ビューの多くの部分に影響を与える場合には、より良好なパフォーマンスが得られます。

- ホーム外PCTリフレッシュ

  これは、ホーム内PCTリフレッシュより高い可用性を提供します。パーティション化されたマテリアライズド・ビューおよびパーティション化されていないマテリアライズド・ビューには、2つの異なるアプローチがあります。切捨ておよびダイレクト・ロードが実現可能ではない場合、変更が比較的大規模であればホーム外リフレッシュを使用する必要があります。切捨ておよびダイレクト・ロードが実現可能な場合、パフォーマンスの観点からは、ホーム内リフレッシュをお薦めします。可用性の観点からは、望ましいのは常にホーム外リフレッシュです。

- ホーム外完全リフレッシュ

  これは、ホーム内完全リフレッシュより高い可用性を提供します。

DBMS_MVIEWパッケージのリフレッシュ・インタフェースをmethod = ?およびout_of_place = trueと指定して使用すると、まずホーム外高速リフレッシュが試みられ、次にホーム外PCTリフレッシュ、最後にホーム外完全リフレッシュが試みられます。次に例を示します。

```
DBMS_MVIEW.REFRESH('CAL_MONTH_SALES_MV', method => '?',
   atomic_refresh => FALSE, out_of_place => TRUE);
```

7.1.4.2 ホーム外リフレッシュの制限および考慮事項

ホーム外リフレッシュには、対応するホーム内リフレッシュを使用する場合に適用されるすべての制限が適用されます。また、これには、次の制限があります。

- マテリアライズド結合ビューおよびマテリアライズド集計ビューのみが使用可能です。
- ON COMMITリフレッシュは使用できません。
- リモート・マテリアライズド・ビュー、キューブ・マテリアライズド・ビュー、オブジェクト・マテリアライズド・ビューは使用できません。
- LOB列は使用できません。
- マテリアライズド・ビューでマテリアライズド・ビューのログ、トリガーまたは制約(NOT NULLを除く)が定義されている場合、使用できません。
- マテリアライズド・ビューがCLUSTERING句を含む場合、使用できません。
- CREATEまたはALTER MATERIALIZED VIEWセッション、またはALTER TABLEセッション内の完全リフレッシュには適用されません。
- アトミック・モードは使用できません。atomic_refreshをTRUEに指定し、out_of_placeをTRUEに指定すると、エラーが表示されます。

ホーム外PCTリフレッシュには、次の制限があります。

- UNION ALLやグルーピング・セットは使用できません。

ホーム外高速リフレッシュには、次の制限があります。

- UNION ALL、グルーピング・セットおよび外部結合は使用できません。
- 複数の表が複合DML文で変更されている場合、マテリアライズド結合ビューでは使用できません。

ホーム外リフレッシュでは、リフレッシュの間の外部表および索引用に追加の記憶域が必要です。したがって、使用可能な表領域が十分にあるか、自動拡張がオンになっている必要があります。

ホーム外PCTリフレッシュのパーティション交換は、マテリアライズド・ビューのグローバル索引に影響を及ぼします。したがって、マテリアライズド・ビューのコンテナ表にグローバル索引が定義されている場合、Oracleでは、パーティション交換の前にグローバル索引を無効にし、パーティション交換の後でグローバル索引を再構築します。この再構築はさらなるオーバーヘッドとなります。

7.1.5 マテリアライズド・ビューのON COMMITリフレッシュについて

ON COMMITメソッドを使用すると、マテリアライズド・ビューを自動的にリフレッシュできます。したがって、マテリアライズド・ビューが定義されている表の更新トランザクションがコミットされるたびに、その変更内容がマテリアライズド・ビューに自動的に反映されます。このアプローチを使用するメリットは、マテリアライズド・ビューのリフレッシュに注意する必要がないことです。唯一のデメリットは、余分な処理が必要なため、コミット完了までの所要時間が少し長くなることです。ただし、データ・ウェアハウスでは、同時プロセスで同じ表の更新が試みられることはまずないため、これは問題ではありません。

7.1.6 マテリアライズド・ビューのON STATEMENTリフレッシュについて

ON STATEMENTリフレッシュ・モードを使用するマテリアライズド・ビューは、マテリアライズド・ビューの実表のいずれかに対してDML操作が実行されるたびに自動的にリフレッシュされます。

ON STATEMENTリフレッシュ・モードを使用すると、実表が変更された場合、マテリアライズド・ビューにすぐに反映されます。トランザクションをコミットしたり、マテリアライズド・ビュー・ログを実表に保持する必要はありません。DML文が後でロールバックされた場合、マテリアライズド・ビューに加えられた対応する変更もロールバックされます。

ON STATEMENTリフレッシュ・モードを使用するには、マテリアライズド・ビューが高速リフレッシュ可能である必要があります。高速リフレッシュのパフォーマンスを向上させるために、ファクト表のROWID列に索引が自動的に作成されます。

ON STATEMENTリフレッシュ・モードの利点は、マテリアライズド・ビュー・ログを保持するというオーバーヘッドを伴うことなく、マテリアライズド・ビューが実表内のデータと常に同期される点です。ただし、このモードでは、DML操作の一環としてマテリアライズド・ビューがリフレッシュされるため、DML操作の実行に要する時間が長くなることがあります。

関連項目:

ON STATEMENT句の制限については、『Oracle Database SQL言語リファレンス』を参照してください

例7-1 ON STATEMENTリフレッシュを使用するマテリアライズド・ビューの作成

この例では、ON STATEMENTリフレッシュ・モードを使用し、sh.sales、sh.customersおよびsh.productsの各表に基づくマテリアライズド・ビューsales_mv_onstatを作成します。実表のいずれかに対してDML操作が実行されると、マテリアライズド・ビューは自動的にリフレッシュされます。マテリアライズド・ビューをリフレッシュするために、DML操作の後にコミットは必要ありません。

```
CREATE MATERIALIZED VIEW sales_mv_onstat
REFRESH FAST ON STATEMENT USING TRUSTED CONSTRAINT
AS
SELECT s.rowid sales_rid, c.cust_first_name first_name, c.cust_last_name last_name,
       p.prod_name prod_name,
    s.quantity_sold quantity_sold, s.amount_sold amount_sold
FROM sh.sales s, sh.customers c, sh.products p
WHERE s.cust_id = c.cust_id and s.prod_id = p.prod_id;
```

7.1.7 DBMS_MVIEWパッケージによる手動リフレッシュについて

マテリアライズド・ビューのON DEMANDリフレッシュを行う場合は、次の表のように4つのリフレッシュ方法から1つ指定できます。デフォルト・オプションは、マテリアライズド・ビューの作成時に定義できます。表7-1にリフレッシュ・オプションを示します。

表7-1 ON DEMANDリフレッシュ方法

リフレッシュ・オプション	パラメータ	説明
COMPLETE	C	マテリアライズド・ビューの定義問合せを再計算することでリフレッシュします。
FAST	F	マテリアライズド・ビューへの変更を増分的に適用することでリフレッシュします。 ローカル・マテリアライズド・ビューの場合は、オプティマイザが最も効率的と判断したリフレッシュ方法が選択されます。考えられるリフレッシュ方法は、ログベースの FAST と FAST_PCT です。
FAST_PCT	P	ディテール表で変更されたパーティションの影響を受けるマテリアライズド・ビューの行

リフレッシュ・オプション	パラメータ	説明
		を再計算することでリフレッシュします。
FORCE	?	高速リフレッシュを試みます。それができない場合は、完全リフレッシュを行います。 ローカル・マテリアライズド・ビューの場合は、オプティマイザが最も効率的と判断したリフレッシュ方法が選択されます。考えられるリフレッシュ方法は、ログベースの FAST、FAST_PCT および COMPLETE です。

ON DEMANDリフレッシュを実行するために、DBMS_MVIEWパッケージの3つのリフレッシュ・プロシージャを使用できます。それぞれに、一連の固有のパラメータがあります。

関連項目:

DBMS_MVIEWパッケージの詳細は、『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

7.1.8 REFRESHを使用した特定のマテリアライズド・ビューのリフレッシュ

DBMS_MVIEW.REFRESHプロシージャを使用して、1つ以上のマテリアライズド・ビューをリフレッシュできます。一部のパラメータはレプリケーションにのみ使用されるため、ここでは説明しません。このプロシージャを使用するために必要なパラメータは次のとおりです。

- カンマで区切られた、リフレッシュ対象のマテリアライズド・ビューのリスト
- リフレッシュ方法: F-Fast、P-Fast_PCT、?-Force、C-Complete
- 使用するロールバック・セグメント
- エラー発生後のリフレッシュ(TRUEまたはFALSE)

  ブール・パラメータ。TRUEに設定すると、number_of_failures出力パラメータは失敗したリフレッシュの数に設定され、通常のエラー・メッセージによって失敗の発生が示されます。インスタンスに関するアラート・ログに、リフレッシュ・エラーの詳細が示されます。デフォルト値のFALSEに設定すると、最初にエラーが発生したときにリフレッシュが停止し、リスト内の残りのマテリアライズド・ビューはリフレッシュされません。
- 次の4つのパラメータは、レプリケーション・プロセスで使用されます。ウェアハウスのリフレッシュの場合は、FALSE, 0,0,0に設定してください。
- アトミック・リフレッシュ(TRUEまたはFALSE)

  TRUEに設定すると、すべてのリフレッシュが1トランザクションで行われます。FALSEに設定されると、各マテリアライズド・ビューは別々のトランザクションで非アトミックにリフレッシュされます。FALSEに設定すると、パラレルDMLによってリフレッシュが最適化され、マテリアライズド・ビューのDDLが切り捨てられます。マテリアライズド・ビューをアトミック・モードでリフレッシュする際、リライト整合性モードがstale_toleratedに設定されている場合にクエリー・リライトを使用できます。リフレッシュがネステッド・ビューで実行される場合は、アトミック・リフレッシュは保証されません。
- ホーム外リフレッシュを使用するかどうか

このパラメータは、既存のすべてのリフレッシュ方法(F、P、C、?)とともに使用できます。したがって、Fとout_of_place = trueを指定すると、ホーム外高速リフレッシュが試みられます。同様に、Pとout_of_place = trueを指定すると、ホーム外PCTリフレッシュが試みられます。

たとえば、マテリアライズド・ビューcal_month_sales_mvに高速リフレッシュを行うには、DBMS_MVIEWパッケージを次のようにコールします。

```
DBMS_MVIEW.REFRESH('CAL_MONTH_SALES_MV', 'F', '', TRUE, FALSE, 0,0,0,
   FALSE, FALSE);
```

複数のマテリアライズド・ビューを同時にリフレッシュすることが可能です。また、その際、すべてに同じリフレッシュ方法を使用する必要はありません。それぞれに異なるリフレッシュ方法を適用するには、マテリアライズド・ビューのリスト順に複数のメソッド・コードを指定します(カンマなし)。たとえば、次のように指定すると、cal_month_sales_mvは完全リフレッシュされ、fweek_pscat_sales_mvは高速リフレッシュされます。

```
DBMS_MVIEW.REFRESH('CAL_MONTH_SALES_MV, FWEEK_PSCAT_SALES_MV', 'CF', '',
  TRUE, FALSE, 0,0,0, FALSE, FALSE);
```

リフレッシュ方法を指定しなければ、マテリアライズド・ビューの定義で指定したデフォルトのリフレッシュ方法が使用されます。

7.1.9 REFRESH_ALL_MVIEWSを使用したすべてのマテリアライズド・ビューのリフレッシュ

リフレッシュするマテリアライズド・ビューを指定する場合、プロシージャDBMS_MVIEW.REFRESH_ALL_MVIEWSを使用する方法もあります。このプロシージャでは、すべてのマテリアライズド・ビューがリフレッシュされます。リフレッシュに失敗したマテリアライズド・ビューがあると、その数がレポートされます。

このプロシージャのパラメータは次のとおりです。

- 失敗数(OUT変数)
- リフレッシュ方法: F-Fast、P-Fast_PCT、?-Force、C-Complete
- エラー発生後のリフレッシュ(TRUEまたはFALSE)

  ブール・パラメータ。TRUEに設定すると、number_of_failures出力パラメータは失敗したリフレッシュの数に設定され、通常のエラー・メッセージによって失敗の発生が示されます。インスタンスに関するアラート・ログに、リフレッシュ・エラーの詳細が示されます。デフォルト値のFALSEに設定すると、最初にエラーが発生したときにリフレッシュが停止し、リスト内の残りのマテリアライズド・ビューはリフレッシュされません。

- アトミック・リフレッシュ(TRUEまたはFALSE)

  TRUEに設定すると、すべてのリフレッシュが1トランザクションで行われます。FALSEに設定されると、各マテリアライズド・ビューは別々のトランザクションで非アトミックにリフレッシュされます。FALSEに設定すると、パラレルDMLによってリフレッシュが最適化され、マテリアライズド・ビューのDDLが切り捨てられます。マテリアライズド・ビューをアトミック・モードでリフレッシュする際、リライト整合性モードがstale_toleratedに設定されている場合にクエリー・リライトを使用できます。リフレッシュがネステッド・ビューで実行される場合は、アトミック・リフレッシュは保証されません。

- ホーム外リフレッシュを使用するかどうか

  このパラメータは、既存のすべてのリフレッシュ方法(F、P、C、?)とともに使用できます。したがって、Fとout_of_place = trueを指定すると、ホーム外高速リフレッシュが試みられます。同様に、Pとout_of_place = trueを指定すると、ホーム外PCTリフレッシュが試みられます。

次に、すべてのマテリアライズド・ビューをリフレッシュする例を示します。

```
DBMS_MVIEW.REFRESH_ALL_MVIEWS(failures,'C','', TRUE, FALSE, FALSE);
```

7.1.10 REFRESH_DEPENDENTを使用した依存マテリアライズド・ビューのリフレッシュ

3番目のプロシージャDBMS_MVIEW.REFRESH_DEPENDENTでは、特定の表または表のリストに依存するマテリアライズド・ビューのみがリフレッシュされます。たとえば、変更がorders表には反映されるが、customer payments表には反映されないとします。orders表を参照するマテリアライズド・ビューのみをリフレッシュするには、REFRESH_DEPENDENTプロシージャをコールします。

このプロシージャのパラメータは次のとおりです。

- 失敗数(OUT変数)
- 依存する表
- リフレッシュ方法: F-Fast、P-Fast_PCT、?-Force、C-Complete
- 使用するロールバック・セグメント
- エラー発生後のリフレッシュ(TRUEまたはFALSE)

  ブール・パラメータ。TRUEに設定すると、number_of_failures出力パラメータは失敗したリフレッシュの数に設定され、通常のエラー・メッセージによって失敗の発生が示されます。インスタンスに関するアラート・ログに、リフレッシュ・エラーの詳細が示されます。デフォルト値のFALSEに設定すると、最初にエラーが発生したときにリフレッシュが停止し、リスト内の残りのマテリアライズド・ビューはリフレッシュされません。

- アトミック・リフレッシュ(TRUEまたはFALSE)

  TRUEに設定すると、すべてのリフレッシュが1トランザクションで行われます。FALSEに設定されると、各マテリアライズド・ビューは別々のトランザクションで非アトミックにリフレッシュされます。FALSEに設定すると、パラレルDMLによってリフレッシュが最適化され、マテリアライズド・ビューのDDLが切り捨てられます。マテリアライズド・ビューをアトミック・モードでリフレッシュする際、リライト整合性モードがstale_toleratedに設定されている場合にクエリー・リライトを使用できます。リフレッシュがネステッド・ビューで実行される場合は、アトミック・リフレッシュは保証されません。

- ネストされているかどうか

  TRUEに設定すると、依存順序に基づいて指定した表のセットのすべての依存マテリアライズド・ビューがリフレッシュされ、マテリアライズド・ビューが実表に対して最新の状態になります。

- ホーム外リフレッシュを使用するかどうか

  このパラメータは、既存のすべてのリフレッシュ方法(F、P、C、?)とともに使用できます。したがって、Fとout_of_place = trueを指定すると、ホーム外高速リフレッシュが試みられます。同様に、Pとout_of_place = trueを指定すると、ホーム外PCTリフレッシュが試みられます。

customers表を参照するすべてのマテリアライズド・ビューの完全リフレッシュを実行するには、次のように指定します。

```
DBMS_MVIEW.REFRESH_DEPENDENT(failures, 'CUSTOMERS', 'C', '', FALSE, FALSE, FALSE);
```

7.1.11 リフレッシュへのジョブ・キューの使用について

ジョブ・キューを使用して、複数のマテリアライズド・ビューをパラレルにリフレッシュできます。キューが使用できない場合、高速リフレッシュでは各ビューがフォアグラウンド・プロセスで順次リフレッシュされます。キューを使用可能にするには、JOB_QUEUE_PROCESSESパラメータを設定する必要があります。このパラメータでバックグラウンド・ジョブ・キュー・プロセスの数を定義することにより、同時に実行可能なマテリアライズド・ビューの数が決まります。同時リフレッシュの数と各リフレッシュの並列度の

バランスが調整されます。マテリアライズド・ビューがリフレッシュされる順序は、ネステッド・マテリアライズド・ビューで指定されている依存性、および他のマテリアライズド・ビューに対するクエリー・リライトによる効率的なリフレッシュの可能性によって決まります(詳細は、マテリアライズド・ビューのリフレッシュのスケジューリングを参照)。このパラメータが有効なのは、atomic_refreshがFALSEに設定されている場合のみです。

DBMS_MVIEW.REFRESHを実行中のプロセスに割込みが入るか、そのインスタンスが停止すると、ジョブ・キュー・プロセスで実行中だったリフレッシュ・ジョブが再度キューに入れられ、引き続き実行されます。これらのジョブを削除するには、DBMS_JOB.REMOVEプロシージャを使用します。

関連項目:

- DBMS_JOBパッケージの詳細は、『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

7.1.12 高速リフレッシュが可能なパターン

すべてのマテリアライズド・ビューの高速リフレッシュが可能であるとは限りません。したがって、パッケージDBMS_MVIEW.EXPLAIN_MVIEWを使用して、マテリアライズド・ビューに使用可能なリフレッシュ方法を判断します。

マテリアライズド・ビューを高速リフレッシュできるようにする方法がわからない場合は、DBMS_ADVISOR.TUNE_MVIEWプロシージャを使用して、高速リフレッシュできるマテリアライズド・ビューの作成に必要な文を含むスクリプトを指定します。

関連項目:

- Oracle Database SQLチューニング・ガイド
- DBMS_MVIEWパッケージの詳細は、基本的なマテリアライズド・ビューを参照してください。

7.1.13 近似問合せに基づいたマテリアライズド・ビューのリフレッシュ

Oracle Databaseは、近似問合せを使用して定義されているマテリアライズド・ビューの高速リフレッシュを実行します。

近似問合せには、近似結果を返すSQL関数が含まれます。近似問合せを含むマテリアライズド・ビューのリフレッシュは、マテリアライズド・ビューの実表で実行されるDML操作によって異なります。

- 挿入操作の場合は、詳細パーセンタイルを含むマテリアライズド・ビューに対して高速リフレッシュが使用されます。
- 削除操作または削除につながるDML操作(UPDATEやMERGEなど)の場合は、マテリアライズド・ビューにWHERE句が含まれない場合のみ、近似集計を含むマテリアライズド・ビューに対して高速リフレッシュが使用されます。

マテリアライズド・ビュー・ログは、高速リフレッシュを必要とするマテリアライズド・ビューのすべての実表に存在する必要があります。

- 近似問合せに基づいたマテリアライズド・ビューをリフレッシュするには:

  DBMS_REFRESH.REFRESHプロシージャを実行して、マテリアライズド・ビューの高速リフレッシュを実行します。

例7-2 **近似問合せに基づいたマテリアライズド・ビューのリフレッシュ**

次の例では、近似問合せに基づいたマテリアライズド・ビューpercentile_per_pdtの高速リフレッシュを実行します。

```
exec DBMS_MVIEW.REFRESH('percentile_per_pdt', method => 'F');
```

関連項目:

- 近似問合せ処理について
- 近似問合せに基づいたマテリアライズド・ビューの作成
- 近似問合せに基づいたクエリー・リライトおよびマテリアライズド・ビュー

7.1.14 表のオンライン再定義中に依存マテリアライズド・ビューをリフレッシュする方法について

DBMS_REDEFINITIONパッケージを使用して表のオンライン再定義を実行しているときに、再定義されている表に依存する高速リフレッシュ可能なマテリアライズド・ビューの増分リフレッシュを実行できます。

Oracle Database 12cリリース2 (12.2)より前は、再定義が実行されている表の依存マテリアライズド・ビューをリフレッシュするには、再定義処理が完了した後に完全なリフレッシュを手動で実行する必要がありました。

表のオンライン再定義中に、依存マテリアライズド・ビューを増分リフレッシュするには、DBMS_REDEFINITON.REDEF_TABLEプロシージャのrefresh_dep_mviewsパラメータにYを設定します。表のオンライン再定義中に依存マテリアライズド・ビューをリフレッシュできるのは、マテリアライズド・ビューが高速リフレッシュ可能で、ROWIDベースのマテリアライズド・ビューまたはマテリアライズド結合ビューではない場合のみです。これらの制限に従っていないマテリアライズド・ビューはリフレッシュされません。

次の依存マテリアライズド・ビューを持つmy_sales表について考えてみます。

- my_sales_pk_mv: 高速リフレッシュ可能で主キーベースのマテリアライズド・ビュー
- my_sales_rid_mv: 高速リフレッシュ可能でROWIDベースのマテリアライズド・ビュー
- my_sales_mjv: 高速リフレッシュ可能なマテリアライズド結合ビュー
- my_sales_mav: 高速リフレッシュ可能なマテリアライズド集計ビュー
- my_sales_rmv: 完全リフレッシュのみが可能なマテリアライズド・ビュー

次のコマンドを実行した場合、高速リフレッシュはmy_sales_pk_mvおよびmy_sales_mavマテリアライズド・ビューでのみ実行されます。

```
DBMS_REDEFINITION.REDEF_TABLE(
uname => 'SH',
tname => 'MY_SALES',
table_compression_type => 'ROW STORE COMPRESS ADVANCED',
refresh_dep_mviews => 'Y');
```

関連項目:

Oracle Database管理者ガイド

7.1.15 パラレル化の推奨初期化パラメータ

パラレル化を効果的にするには、次の初期化パラメータを正しく設定する必要があります。

- PARALLEL_MAX_SERVERSは、パラレル化に対応できるように十分高い値にします。リフレッシュ文に必要なスレーブの数

を考慮する必要があります。たとえば、並列度が8であれば、16のスレーブ・プロセスが必要です。

- インスタンスでソートと結合のメモリー使用を自動的に管理するには、PGA_AGGREGATE_TARGETを設定する必要があります。メモリー・パラメータを手動で設定する場合は、SORT_AREA_SIZEをHASH_AREA_SIZE未満にする必要があります。
- OPTIMIZER_MODEは、all_rowsと同じ値にします。

すべての表および索引を効率的に分析すると、最適化が向上します。

関連項目:

『Oracle Database VLDBおよびパーティショニング・ガイド』

7.1.16 リフレッシュの監視

ジョブの実行中は、V$SESSION_LONGOPSビューを問い合せて、各マテリアライズド・ビューのリフレッシュの進行状況を確認できます。

```
SELECT * FROM V$SESSION_LONGOPS;
```

どのジョブがどのキューに入っているかを確認するには、次の文を使用します。

```
SELECT * FROM DBA_JOBS_RUNNING;
```

7.1.17 マテリアライズド・ビューのステータスのチェック

マテリアライズド・ビューのステータスをチェックできるように、DBA_MVIEWS、ALL_MVIEWS、USER_MVIEWSの3つのビューが用意されています。マテリアライズド・ビューが最新か失効しているかをチェックするには、次の文を発行します。

```
SELECT MVIEW_NAME, STALENESS, LAST_REFRESH_TYPE, COMPILE_STATE
FROM USER_MVIEWS ORDER BY MVIEW_NAME;

MVIEW_NAME            STALENESS      LAST_REF       COMPILE_STATE
----------            ---------      --------       -------------
CUST_MTH_SALES_MV     NEEDS_COMPILE  FAST           NEEDS_COMPILE
PROD_YR_SALES_MV      FRESH          FAST           VALID
```

compile_state列がNEEDS COMPILEとなっている場合、表示されているその他の列の値は信頼できず、実際のステータスが反映されていない場合があります。マテリアライズド・ビューを再検証するには、次の文を発行します。

```
ALTER MATERIALIZED VIEW [materialized_view_name] COMPILE;
```

続いて、SELECT文を再度発行します。

7.1.17.1 パーティションの最新状態の表示

一部のビューでは、実表のパーティションの状態を確認し、マテリアライズド・ビューのデータの最新および失効の範囲を判別できます。該当のビューは次のとおりです。

- *_USER_MVIEWS

  マテリアライズド・ビューのパーティション・チェンジ・トラッキング(PCT)情報の判別に使用します。

- *_USER_MVIEW_DETAIL_RELATIONS

  マテリアライズド・ビューの基になるディテール表のパーティション情報を表示するのに使用します。

- *_USER_MVIEW_DETAIL_PARTITION

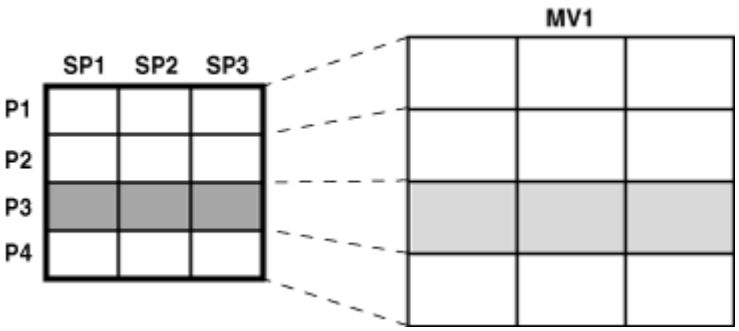
  最新のパーティションを判別するのに使用します。

- *_USER_MVIEW_DETAIL_SUBPARTITION

  最新のサブパーティションを判別するのに使用します。

これらのビューの使用例を次に示します。図7-1は、レンジ-リスト・パーティション化された表とそれに基づくマテリアライズド・ビューを示しています。P1、P2、P3はパーティションで、SP1、SP2、SP3はサブパーティションです。

図7-1 PCTによる最新状態の判別



関連項目:

最新状態の判別に使用するビューの使用例

7.1.17.1.1 最新状態の判別に使用するビューの使用例

この項では、マテリアライズド・ビューとそのディテール表のPCTおよび最新状態の情報を判別するための例を示します。

例7-3 マテリアライズド・ビューのPCTステータスの確認

マテリアライズド・ビューのPCT情報にアクセスするには、次のようにUSER_MVIEWSを問い合せます。

```
SELECT MVIEW_NAME, NUM_PCT_TABLES, NUM_FRESH_PCT_REGIONS,
   NUM_STALE_PCT_REGIONS
FROM USER_MVIEWS
WHERE MVIEW_NAME = MV1;

MVIEW_NAME NUM_PCT_TABLES NUM_FRESH_PCT_REGIONS NUM_STALE_PCT_REGIONS
---------- -------------- ---------------------- ----------------------
       MV1              1                      9                      3
```

例7-4 マテリアライズド・ビューのディテール表におけるPCTステータスの確認

PCTディテール表の情報にアクセスするには、次のようにUSER_MVIEW_DETAIL_RELATIONSを問い合せます。

```
SELECT MVIEW_NAME, DETAILOBJ_NAME, DETAILOBJ_PCT,
   NUM_FRESH_PCT_PARTITIONS, NUM_STALE_PCT_PARTITIONS
FROM USER_MVIEW_DETAIL_RELATIONS
WHERE MVIEW_NAME = MV1;

MVIEW_NAME  DETAILOBJ_NAME  DETAIL_OBJ_PCT  NUM_FRESH_PCT_PARTITIONS  NUM_STALE_PCT_PARTITIONS
----------  --------------  --------------  -------------------------  -------------------------
       MV1              T1               Y                          3                          1
```

例7-5 最新のパーティションの確認

パーティションのPCTの最新状態の情報にアクセスするには、次のようにUSER_MVIEW_DETAIL_PARTITIONを問い合せます。

```
SELECT MVIEW_NAME,DETAILOBJ_NAME,DETAIL_PARTITION_NAME,
   DETAIL_PARTITION_POSITION,FRESHNESS
FROM USER_MVIEW_DETAIL_PARTITION
WHERE MVIEW_NAME = MV1;

MVIEW_NAME  DETAILOBJ_NAME  DETAIL_PARTITION_NAME  DETAIL_PARTITION_POSITION  FRESHNESS
----------  --------------  ---------------------  -------------------------  ---------
       MV1              T1                     P1                          1      FRESH
       MV1              T1                     P2                          2      FRESH
       MV1              T1                     P3                          3      STALE
       MV1              T1                     P4                          4      FRESH
```

例7-6 最新のサブパーティションの確認

サブパーティションのPCTの最新状態の情報にアクセスするには、次のようにUSER_MVIEW_DETAIL_SUBPARTITIONを問い合せます。

```
SELECT MVIEW_NAME,DETAILOBJ_NAME,DETAIL_PARTITION_NAME, DETAIL_SUBPARTITION_NAME,
    DETAIL_SUBPARTITION_POSITION,FRESHNESS
FROM USER_MVIEW_DETAIL_SUBPARTITION
WHERE MVIEW_NAME = MV1;

MVIEW_NAME DETAILOBJ DETAIL_PARTITION DETAIL_SUBPARTITION_NAME DETAIL_SUBPARTITION_POS FRESHNESS
---------- --------- ---------------- ------------------------ ------------------------ ---------
       MV1        T1               P1                      SP1                        1     FRESH
       MV1        T1               P1                      SP2                        1     FRESH
       MV1        T1               P1                      SP3                        1     FRESH
       MV1        T1               P2                      SP1                        1     FRESH
       MV1        T1               P2                      SP2                        1     FRESH
       MV1        T1               P2                      SP3                        1     FRESH
       MV1        T1               P3                      SP1                        1     STALE
       MV1        T1               P3                      SP2                        1     STALE
       MV1        T1               P3                      SP3                        1     STALE
       MV1        T1               P4                      SP1                        1     FRESH
       MV1        T1               P4                      SP2                        1     FRESH
       MV1        T1               P4                      SP3                        1     FRESH
```

7.1.18 マテリアライズド・ビューのリフレッシュのスケジュール

ほとんどの場合、データベースには複数のマテリアライズド・ビューがあります。一部のマテリアライズド・ビューは、他のマテリアライズド・ビューに対してリライトして計算されているものもあります。これは、ネステッド・マテリアライズド・ビューがある、または一部の階層の異なるレベルにマテリアライズド・ビューがあるデータ・ウェアハウス環境では、非常に一般的なことです。

このような場合、BUILD DEFERREDとしてマテリアライズド・ビューを作成し、DBMS_MVIEWパッケージのいずれかのリフレッシュ・プロシージャを発行してすべてのマテリアライズド・ビューをリフレッシュする必要があります。Oracle Databaseによって、依存性が計算され、正しい順序でマテリアライズド・ビューがリフレッシュされます。階層的キューブのマテリアライズド・ビューの例で説明されている完全な階層的キューブの例を参照してください。すべてのマテリアライズド・ビューがBUILD DEFERREDとして作成されているとします。マテリアライズド・ビューをBUILD DEFERREDとして作成すると、すべてのマテリアライズド・ビューのメタデータのみが作成されます。これで、DBMS_MVIEWパッケージのリフレッシュ・プロシージャのいずれかをコールし、すべてのマテリアライズド・ビューを正しい順序でリフレッシュできます。

```
DECLARE numerrs PLS_INTEGER;
BEGIN DBMS_MVIEW.REFRESH_DEPENDENT (
   number_of_failures => numerrs, list=>'SALES', method => 'C');
DBMS_OUTPUT.PUT_LINE('There were ' || numerrs || ' errors during refresh');
END;
/
```

プロシージャは、マテリアライズド・ビューを依存性の順序(sales_hierarchical_mon_cube_mvから始まり、sales_hierarchical_qtr_cube_mv、sales_hierarchical_yr_cube_mv、sales_hierarchical_all_cube_mvという順序)でリフレッシュします。各マテリアライズド・ビューは、リストにおける1つ前のマテリアライズド・ビューに対してリライトされます。

同じ種類のリライトは、PCTリフレッシュを行う際にも使用できます。PCTリフレッシュでは、ディテール表で変更された行に対応するマテリアライズド・ビューの行が再計算されます。リフレッシュ時に最新のマテリアライズド・ビューが他にある場合は、ディテール表ではなくそのマテリアライズド・ビューが直接使用されます。

したがって、指定した方法に関係なく、マテリアライズド・ビューのリストをDBMS_MVIEWパッケージのいずれかのリフレッシュ・プロシージャに渡すことは有益です。これによって、プロシージャは、マテリアライズド・ビューに対してリフレッシュを実行する順序を認識できます。

7.2 マテリアライズド・ビューのリフレッシュのヒント

この項の内容は次のとおりで、マテリアライズド・ビューのリフレッシュのヒントを含みます。

- 集計を含むマテリアライズド・ビューのリフレッシュのヒント
- 集計を含まないマテリアライズド・ビューのリフレッシュのヒント
- ネステッド・マテリアライズド・ビューのリフレッシュのヒント
- UNION ALLでの高速リフレッシュのヒント
- コミットSCNベースのマテリアライズド・ビュー・ログを使用した高速リフレッシュのヒント
- マテリアライズド・ビューのリフレッシュ後のヒント

7.2.1 集計を含むマテリアライズド・ビューのリフレッシュのヒント

ここでは、集計を含むマテリアライズド・ビューのリフレッシュ機能を使用する場合のガイドラインを示します。

- 高速リフレッシュの場合は、ROWID、SEQUENCEおよびINCLUDING NEW VALUES句を使用して、マテリアライズド・ビューに関連するすべてのディテール表のマテリアライズド・ビュー・ログを作成します。

  マテリアライズド・ビュー・ログには、マテリアライズド・ビューに使用されると思われる表の列をすべて含めます。

  高速リフレッシュは、マテリアライズド・ビュー・ログでSEQUENCEオプションが省略されていても可能な場合があります。すべてのディテール表で挿入または削除しか発生しないと判断できる場合、マテリアライズド・ビュー・ログにはSEQUENCE句は不要です。ただし、複数の表が更新される可能性があるか必要な場合、または特定の更新の手順が不明な場合は、SEQUENCE句が含まれているかどうかを確認してください。

- Oracleのバルク・ロード・ユーティリティまたはダイレクト・パス・インサート(ロードに対するAPPENDヒント付きのINSERT)を使用します。Oracle Database 12cより、データベースではバルク・ロード操作の一部として(CTASおよびIAS)、索引の作成時の統計の収集方法に似た表統計の自動収集を行います。データ・ロードの際に統計を収集することによって、スキャン操作の追加を回避し、ユーザーがデータを使用できるようになるとすぐに必要な統計を提供できます。

  これは、従来の挿入に比べてはるかに効率的です。ロード中はすべての制約を使用禁止にし、ロード終了後に使用可能に戻します。ダイレクト・ロードと従来型DMLのどちらを使用する場合も、マテリアライズド・ビュー・ログは必要であるため注意してください。

  マテリアライズド・ビューに対する従来型の複合DML操作、ダイレクト・パス・インサートおよび高速リフレッシュの順序を最適化してください。従来のDMLおよびダイレクト・ロードと混在させて高速リフレッシュを使用できます。高速リフレッシュでは、次に示すように、ダイレクト・ロードのみが発生することがわかっていれば、大幅な最適化を実行できます。

  1. ディテール表へのダイレクト・パス・インサート(SQL*LoaderまたはINSERT /*+ APPEND */)を行います。

2. マテリアライズド・ビューのリフレッシュ

3. 従来型の複合DML操作を行います。

4. マテリアライズド・ビューをリフレッシュします。

高速リフレッシュを、ディテール表に対する従来型の複合DML(INSERT、UPDATEおよびDELETE)と併用できます。ただし、高速リフレッシュで処理中に大幅な最適化を実行できるのは、次のように、表に対して挿入または削除のみが行われていることが検出される場合です。

5. ディテール表に対するDML INSERTまたはDELETE

6. 複数のマテリアライズド・ビューのリフレッシュ

7. ディテール表のDML更新

8. マテリアライズド・ビューをリフレッシュします。

さらに最適化するには、INSERTとDELETEを分離します。

可能であれば、最後にリフレッシュを1つのみ発行するのではなく、前述のように各種のデータ変更の後にリフレッシュを実行します。可能でない場合は、従来のDMLを挿入対象の表に限定すると、リフレッシュのパフォーマンスが大幅に改善されます。DELETEとダイレクト・ロードの混在を回避してください。

さらに、ON COMMITリフレッシュの場合、Oracleは、コミットされたトランザクションで実行されたDMLのタイプを記録します。したがって、同じトランザクションの他の表には、ダイレクト・パス・インサートおよびDMLを実行しないでください。Oracleがリフレッシュ・フェーズを最適化できない可能性があります。

ON COMMITマテリアライズド・ビューの場合は、各トランザクションの最後にリフレッシュが自動的に発生します。DML文を分離できない場合があり、その場合はトランザクションを短くすると有効です。ただし、ディテール表に対して多数の変更を行う場合は、更新のたびにリフレッシュするより、それらの更新を1回のトランザクションで実行し、コミット時に一度にマテリアライズド・ビューをリフレッシュする方が効率的です。

- 次の方法を使用できるため、表をパーティション化することをお薦めします。

  1. パラレルDML

     大量のロードまたはリフレッシュの場合は、パラレルDMLを使用可能にすると、処理時間を短縮できます。

  2. パーティション・チェンジ・トラッキング(PCT)高速リフレッシュ

     マテリアライズド・ビューの高速リフレッシュは、ディテール表に対するパーティション・メンテナンス操作後に実行できます。マテリアライズド・ビューでPCTを使用可能にする方法の詳細は、パーティション・チェンジ・トラッキングについてを参照してください。

- また、マテリアライズド・ビューをパーティション化すると、リフレッシュでパラレルDMLを使用してマテリアライズド・ビューを更新できるため、パフォーマンスが改善されます。たとえば、ディテール表とマテリアライズド・ビューがパーティション化されており、PARALLEL句があるとします。次の順序により、Oracleではマテリアライズド・ビューのリフレッシュをパラレル化できます。

  1. ディテール表にバルク・ロードします。

  2. ALTER SESSION ENABLE PARALLEL DML文でパラレルDMLを使用可能にします。

  3. マテリアライズド・ビューをリフレッシュします。

- DBMS_MVIEW.REFRESHを使用してリフレッシュを行うには、パラメータatomic_refreshをFALSEに設定します。

  1. COMPLETEリフレッシュでは、これにより、マテリアライズド・ビューの既存の行がTRUNCATEによって削除されます。

これはDELETEより高速です。

2. PCTリフレッシュでは、マテリアライズド・ビューが適切にパーティション化されている場合は、これにより、マテリアライズド・ビューの影響を受けるパーティションの行がTRUNCATE PARTITIONによって削除されます。これはDELETEより高速です。

3. FASTまたはFORCEリフレッシュでは、COMPLETEリフレッシュまたはPCTリフレッシュが選択された場合、これにより、前述のTRUNCATE最適化を使用できます。

- JOB_QUEUESを指定してDBMS_MVIEW.REFRESHを使用する場合は、atomicをFALSEに設定してください。このように設定しなければ、JOB_QUEUESは使用されません。ジョブ・キュー・プロセスの数を、プロセッサ数より大きくなるように設定します。

  ジョブ・キューが使用可能になっており、リフレッシュするマテリアライズド・ビューが多い場合は、個別にコールするより1つのコマンドですべてをリフレッシュするほうが高速です。

- リフレッシュしたマテリアライズド・ビューをクエリー・リライトで使用できるようにするには、REFRESH FORCEを使用します。最適なリフレッシュ方法が選択されます。高速リフレッシュができない場合は、完全リフレッシュが行われます。

- 1回のプロシージャ・コールで、すべてのマテリアライズド・ビューをリフレッシュします。これにより、ネステッド・マテリアライズド・ビューで指定された依存性、および他のマテリアライズド・ビューに対するクエリー・リライトを使用した効率的なリフレッシュを考慮した正しい順序によるすべてのマテリアライズド・ビューのリフレッシュが、Oracleによってスケジューリングされます。

7.2.2 集計を含まないマテリアライズド・ビューのリフレッシュのヒント

結合を含み、集計を含まないマテリアライズド・ビューは、集計を含むマテリアライズド・ビューよりもかなりサイズが大きくなる傾向があるため、ディテール表の結合列ROWIDのそれぞれに索引を設定すると、リフレッシュ・パフォーマンスが大幅に向上します。たとえば、次のマテリアライズド・ビューを考えてみます。

```
CREATE MATERIALIZED VIEW detail_fact_mv BUILD IMMEDIATE AS
SELECT s.rowid "sales_rid", t.rowid "times_rid", c.rowid "cust_rid",
   c.cust_state_province, t.week_ending_day, s.amount_sold
FROM sales s, times t, customers c
WHERE s.time_id = t.time_id AND s.cust_id = c.cust_id;
```

この場合は、sales_rid、times_ridおよびcust_rid列に索引を作成します。リフレッシュを起動する前に、セッションでのパラレルDMLを可能にするとともにパーティション化も行ってください。リフレッシュ・パフォーマンスが大幅に向上します。

このタイプのマテリアライズド・ビューは、DMLがディテール表に実行される場合でも高速リフレッシュも可能です。このタイプのマテリアライズド・ビューでも、単一表集計マテリアライズド・ビューと同じ手順に従ってください。つまり、1つのタイプの変更(ダイレクト・パス・インサートまたはDML)を行うたびにマテリアライズド・ビューをリフレッシュします。これは、Oracle Databaseが、1つのタイプの変更のみが行われたと検出した場合に、大幅な最適化を行うためです。

また、複数の表をすべてロードしてからリフレッシュを行うより、表を1つロードするたびにリフレッシュを起動することをお薦めします。

ON COMMITリフレッシュの場合、Oracleは、コミットされたトランザクションで実行されたDMLのタイプを記録します。そのため、同じトランザクションの他の表には、できるだけダイレクト・パス・ロードおよび従来型DMLを実行しないでください。Oracleがリフレッシュ・フェーズを最適化できない可能性があります。たとえば、次のような方法はお薦めできません。

1. ファクト表への新規データのダイレクト・ロード
2. 記憶域表へのDML
3. コミット

また、異なるタイプの従来型DML文は、できるだけ混在させないでください。これも、高速リフレッシュ時の様々な最適化を妨げる要因になります。たとえば、次のような文は使用しないでください。

1. ファクト表への挿入
2. ファクト表からの削除
3. コミット

多数の更新が必要な場合は、できるかぎり1回のトランザクションにまとめてください。これによって、リフレッシュは、更新のたびにではなく、コミット時に1回のみですみます。

データ・ウェアハウス環境では、マテリアライズド・ビューがパラレル句を含む場合、次のステップで行うことをお薦めします。

1. ファクト表にバルク・ロードします。
2. パラレルDMLを使用可能にします。
3. ALTER SESSION ENABLE PARALLEL DML文を発行します。
4. マテリアライズド・ビューをリフレッシュします。

7.2.3 ネステッド・マテリアライズド・ビューのリフレッシュのヒント

ネステッド・マテリアライズド・ビューのベースとなるオブジェクトは、マテリアライズド・ビューのリフレッシュ時にはすべて通常の表として扱われます。ON COMMITリフレッシュ・オプションが指定されている場合は、すべてのマテリアライズド・ビューがコミット時に適切な順番でリフレッシュされます。Oracleは、一部順序付けられたマテリアライズド・ビューの集合を作成し、リフレッシュ完了後にはすべてのマテリアライズド・ビューが最新になっているように、リフレッシュを行います。マテリアライズド・ビューの状態は、適切なビュー(USER、DBA_、ALL_MVIEWS)に問い合せることでチェックできます。

マテリアライズド・ビューのいずれかがON DEMANDリフレッシュとして定義されている場合(リフレッシュ方法がFAST、FORCE、COMPLETEのいずれであるかにかかわらず)、ネステッド・マテリアライズド・ビューは、(最新かどうかにかかわらず)他のマテリアライズド・ビューの現在の内容との比較によってリフレッシュされます。そのため、正しい順番で(マテリアライズド・ビュー間の依存性を考慮して)リフレッシュする必要があります。これには、ネストされた階層の一番上にあるマテリアライズド・ビューに対してリフレッシュ・プロシージャを呼び出し、nestedパラメータをTRUEに指定します。

コミット中にリフレッシュに失敗した場合は、リフレッシュされていないマテリアライズド・ビューのリストがアラート・ログに書き込まれます。それらをすべて依存マテリアライズド・ビューとともに手動でリフレッシュする必要があります。

ネステッド・マテリアライズド・ビューのリフレッシュには、通常のマテリアライズド・ビューの場合と同じDBMS_MVIEWプロシージャを使用します。

これらのプロシージャは、ネステッド・マテリアライズド・ビューに対して使用されると、次のように動作します。

- 他のマテリアライズド・ビュー上に作成されているマテリアライズド・ビューmy_mvにREFRESHが適用されると、my_mvは、nested => TRUEを指定しないかぎり、他のマテリアライズド・ビューの現在の内容を反映するようにリフレッシュされます(つまり、他のマテリアライズド・ビューが最初にリフレッシュされることはありません)。
- REFRESH_DEPENDENTをマテリアライズド・ビューmy_mvに適用すると、nested => TRUEを指定しないかぎり、my_mvに直接依存しているマテリアライズド・ビューのみがリフレッシュされます(my_mvに依存しているマテリアライズド・ビューにさらに依存しているマテリアライズド・ビューはリフレッシュされません)。
- REFRESH_ALL_MVIEWSを使用すると、マテリアライズド・ビューがリフレッシュされる順序は、ネステッド・マテリアライズド・ビュー間の依存性が考慮されます。
- GET_MV_DEPENDENCIESで、あるオブジェクトに直接依存するマテリアライズド・ビューのリストが作成されます。

7.2.4 UNION ALLでの高速リフレッシュのヒント

マテリアライズド・ビュー定義にメンテナンス列を指定することで、UNION ALLを使用するマテリアライズド・ビューに高速リフレッシュを使用できます。たとえば、UNION ALL演算子を持つマテリアライズド・ビューは、次のように高速リフレッシュできます。

```
CREATE MATERIALIZED VIEW fast_rf_union_all_mv AS
SELECT x.rowid AS r1, y.rowid AS r2, a, b, c, 1 AS marker
FROM x, y WHERE x.a = y.b
UNION ALL
SELECT p.rowid, r.rowid, a, c, d, 2 AS marker
FROM p, r WHERE p.a = r.y;
```

メンテナンス・マーカー列(例ではMARKER)の形式は、numeric_or_string_literal AS column_aliasである必要があります。ここで各UNION ALLメンバーは、numeric_or_string_literalに対して別個の値を持ちます。

7.2.5 コミットSCNベースのマテリアライズド・ビュー・ログを使用した高速リフレッシュのヒント

実表のマテリアライズド・ビュー・ログにWITH COMMIT SCN句を含めるようにすると、多くの場合、高速リフレッシュのパフォーマンスを大幅に改善できます。WITH COMMIT SCNを使用してマテリアライズド・ビュー・ログの処理を最適化することで、高速リフレッシュの処理時間を短縮できます。次の例は、この句の使用方法を示しています。

```
CREATE MATERIALIZED VIEW LOG ON sales WITH ROWID
 (prod_id, cust_id, time_id, channel_id, promo_id, quantity_sold, amount_sold),
COMMIT SCN INCLUDING NEW VALUES;
```

マテリアライズド・ビューのリフレッシュではコミットSCNベースのマテリアライズド・ビュー・ログが自動的に使用されるので、リフレッシュ時間が短縮されます。

COMMIT SCNを利用できるのは、新しいマテリアライズド・ビュー・ログのみです。既存のマテリアライズド・ビュー・ログの場合は、削除して再作成しないかぎり、変更してCOMMIT SCNを追加できません。

タイムスタンプ・ベースのマテリアライズド・ビュー・ログを使用した実表とコミットSCNベースのマテリアライズド・ビュー・ログを使用した実表の両方でマテリアライズド・ビューが作成されると、エラー(ORA-32414)が発生し、これらのマテリアライズド・ビュー・ログが高速リフレッシュに関して相互に互換性がないことを示すメッセージが表示されます。

7.2.6 マテリアライズド・ビューのリフレッシュ後のヒント

ロードまたは増分ロードを行い、ディテール表索引を再作成した後は、整合性制約(ある場合)を使用可能に戻し、そのディテール表から導出されたマテリアライズド・ビューおよびマテリアライズド・ビュー索引をリフレッシュする必要があります。データ・ウェアハウス環境では、参照整合性制約は、通常、NOVALIDATEまたはRELYオプションで使用可能にできます。リフレッシュ操作を行う前に、そのリフレッシュ操作をリカバリ可能にする必要があるかどうかを決定する必要があります。マテリアライズド・ビューのデータは冗長で、いつでもディテール表から再作成できるため、マテリアライズド・ビューのロギングは使用禁止にすることをお薦めします。ロギングを無効にし、増分リフレッシュをリカバリの必要なしで実行するには、リフレッシュの前にALTER MATERIALIZED VIEW ... NOLOGGING文を使用します。

ON COMMIT方法でマテリアライズド・ビューをリフレッシュする場合は、リフレッシュ操作後、アラート・ログalert_*SID*.logおよびトレース・ファイルora_*SID*_number.trcで、エラーが発生していないかどうかをチェックします。

7.3 パーティション表付きマテリアライズド・ビューの使用

データ・ウェアハウスの主なメンテナンス・コンポーネントは、ディテール・データの変更時におけるマテリアライズド・ビューの同期化

(リフレッシュ)です。このとき基礎となるディテール表をパーティション化していると、リフレッシュ・タスクの所要時間を短縮できます。これは、パーティション化によりパラレルDMLを使用してマテリアライズド・ビューを更新できるようになるためです。また、パーティション・チェンジ・トラッキングも使用可能になります。

パーティション・チェンジ・トラッキングによるマテリアライズド・ビューの高速リフレッシュでは、PCTリフレッシュに関する追加情報が提供されています。

7.3.1 パーティション・チェンジ・トラッキングによるマテリアライズド・ビューの高速リフレッシュ

データ・ウェアハウスでは、通常、ディテール表の変更により、DROP、EXCHANGE、MERGEおよびADD PARTITIONなどのパーティション・メンテナンス操作が必要になります。このような操作の後でマテリアライズド・ビューをメンテナンスするには、手動メンテナンス(CONSIDER FRESHも参照)または完全リフレッシュを使用する必要がありました。現在では、パーティション・チェンジ・トラッキング(PCT)リフレッシュと呼ばれる高速リフレッシュ機能を使用できます。

PCTを使用可能にするには、ディテール表をパーティション化する必要があります。この機能では、マテリアライズド・ビュー自体をパーティション化する必要はありません。PCTリフレッシュが可能な場合は、ユーザーに対して透過的に処理が発生します。PCTの要件は、パーティション・チェンジ・トラッキングについてを参照してください。

この機能の使用例を次に示します。

- マテリアライズド・ビューのPCT高速リフレッシュ: 使用例1
- マテリアライズド・ビューのPCT高速リフレッシュ: 使用例2
- マテリアライズド・ビューのPCT高速リフレッシュ: 使用例3

7.3.1.1 マテリアライズド・ビューのPCT高速リフレッシュ: 使用例1

この使用例では、salesはtime_id列を使用してパーティション化された表で、productsはprod_category列でパーティション化されています。表timesはパーティション表ではありません。

1. マテリアライズド・ビューを作成します。次のマテリアライズド・ビューは、PCTの要件を満たしています。

```
CREATE MATERIALIZED VIEW cust_mth_sales_mv
BUILD IMMEDIATE
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE AS
SELECT s.time_id, s.prod_id, SUM(s.quantity_sold), SUM(s.amount_sold),
       p.prod_name, t.calendar_month_name, COUNT(*),
       COUNT(s.quantity_sold), COUNT(s.amount_sold)
FROM sales s, products p, times t
WHERE  s.time_id = t.time_id AND s.prod_id = p.prod_id
GROUP BY t.calendar_month_name, s.prod_id, p.prod_name, s.time_id;
```

2. DBMS_MVIEW.EXPLAIN_MVIEWプロシージャを実行して、PCTリフレッシュが可能な表を判断します。

```
MVNAME              CAPABILITY_NAME   POSSIBLE  RELATED_TEXT  MSGTXT
------------------  ----------------  --------  ------------  -----------------
CUST_MTH_SALES_MV   PCT               Y         SALES
CUST_MTH_SALES_MV   PCT_TABLE         Y         SALES
CUST_MTH_SALES_MV   PCT_TABLE         N         PRODUCTS      no partition key
                                                              or PMARKER
                                                              in SELECT list
CUST_MTH_SALES_MV   PCT_TABLE         N         TIMES         relation is not
                                                              partitionedtable
```

EXPLAIN_MVIEWから抜粋したサンプル出力からもわかるように、sales表に対して実行されるパーティション・メンテナンス操作では、PCT高速リフレッシュが可能です。ただし、products表に対するパーティション・メンテナンス操作や更新の後は、cust_mth_sales_mvに十分な情報が含まれていないため、PCTリフレッシュは実行できません。times表はパーティション化されていないため、PCTリフレッシュできないことに注意してください。Oracle DatabaseでPCTリフレッシュが適用されるのは、更新されたすべての表に関してPCTをサポートするだけの十分な情報がマテリアライズド・ビューにあると判断できる場合です。DBA_MVIEWSやDBA_MVIEW_DETAIL_PARTITIONなどのビューを使用すると、最新および失効したパーティションを確認できます。

このプロシージャの使用方法およびPCT関連のビューの詳細は、マテリアライズド・ビュー機能の分析を参照してください。

3. 少し後の時点で、sales表の1つのパーティションのSPLIT操作が必要になったとします。

```
ALTER TABLE SALES
SPLIT PARTITION month3 AT (TO_DATE('05-02-1998', 'DD-MM-YYYY'))
INTO (PARTITION month3_1 TABLESPACE summ,
      PARTITION month3 TABLESPACE summ);
```

4. sales表になんらかのデータを挿入します。
5. DBMS_MVIEW.REFRESHプロシージャを使用して、cust_mth_sales_mvを高速リフレッシュします。

```
EXECUTE DBMS_MVIEW.REFRESH('CUST_MTH_SALES_MV', 'F',
   '',TRUE,FALSE,0,0,0,FALSE);
```

この使用例では、高速リフレッシュを行うとPCTリフレッシュが自動的に実行されます。ただし、パーティション・メンテナンス操作によって表が更新され、PCTが不可能な場合には、高速リフレッシュは行われません。これについては、マテリアライズド・ビューのPCT高速リフレッシュ: 使用例2を参照してください。

マテリアライズド・ビューのPCT高速リフレッシュ: 使用例1は、次のように、PMARKER句を使用してマテリアライズド・ビューを作成した場合にも該当します。

```
CREATE MATERIALIZED VIEW cust_sales_marker_mv
BUILD IMMEDIATE
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE AS
SELECT DBMS_MVIEW.PMARKER(s.rowid) s_marker, SUM(s.quantity_sold),
  SUM(s.amount_sold), p.prod_name, t.calendar_month_name, COUNT(*),
  COUNT(s.quantity_sold), COUNT(s.amount_sold)
FROM sales s, products p, times t
WHERE  s.time_id = t.time_id AND s.prod_id = p.prod_id
GROUP BY DBMS_MVIEW.PMARKER(s.rowid),
         p.prod_name, t.calendar_month_name;
```

7.3.1.2 マテリアライズド・ビューのPCT高速リフレッシュ: 使用例2

この使用例では、最初の3つのステップは、マテリアライズド・ビューのPCT高速リフレッシュ: 使用例1と同じです。その後、sales表に対するパーティションのSPLIT操作を実行します。ただし、マテリアライズド・ビューをリフレッシュする前に、times表にレコードを挿入します。

1. マテリアライズド・ビューのPCT高速リフレッシュ: 使用例1と同じです。
2. マテリアライズド・ビューのPCT高速リフレッシュ: 使用例1と同じです。
3. マテリアライズド・ビューのPCT高速リフレッシュ: 使用例1と同じです。
4. マテリアライズド・ビューのPCT高速リフレッシュ: 使用例1と同じSPLIT操作の発行後に、times表になんらかのデータを挿入します。

```
ALTER TABLE SALES
```

```
SPLIT PARTITION month3 AT (TO_DATE('05-02-1998', 'DD-MM-YYYY'))
INTO (PARTIITION month3_1 TABLESPACE summ,
      PARTITION month3 TABLESPACE summ);
```

5. cust_mth_sales_mvをリフレッシュします。

```
EXECUTE DBMS_MVIEW.REFRESH('CUST_MTH_SALES_MV', 'F',
    '', TRUE, FALSE, 0, 0, 0, FALSE, FALSE);
ORA-12052: cannot fast refresh materialized view SH.CUST_MTH_SALES_MV
```

PCT高速リフレッシュを実行できない表に対してDMLが発生しているため、このマテリアライズド・ビューは高速リフレッシュできません。この状況を回避するには、パーティション追跡高速リフレッシュが使用可能になっているディテール表に対するパーティション・メンテナンス操作の直後に、高速リフレッシュを実行することをお薦めします。

マテリアライズド・ビューのPCT高速リフレッシュ: 使用例2の状況が発生した場合は、2つの方法が考えられます。つまり、完全リフレッシュを実行するか、後述のようにCONSIDER FRESHオプションに切り替えるかです。ただし、CONSIDER FRESHとパーティション・チェンジ・トラッキングの高速リフレッシュには、互換性がないため注意する必要があります。ALTER MATERIALIZED VIEW cust_mth_sales_mv CONSIDER FRESH文が発行されると、完全リフレッシュが完了するまで、このマテリアライズド・ビューにはPCTリフレッシュが適用されなくなります。さらに、手動でマテリアライズド・ビューをリフレッシュしないかぎり、CONSIDER FRESHを使用しないでください。

データ・ウェアハウスで一般的な状況は、データのローリング・ウィンドウを使用することです。この場合、たとえばディテール表とマテリアライズド・ビューに過去12か月分のデータが含まれる場合があります。毎月、1か月分の新規データが表に追加され、最も古い月が削除(またはアーカイブ)されます。PCTリフレッシュは、この場合にマテリアライズド・ビューをメンテナンスする非常に効率的なメカニズムです。

7.3.1.3 マテリアライズド・ビューのPCT高速リフレッシュ: 使用例3

1. 通常、新規データは、新規パーティションを追加し、それを新規データを含む表と交換することでディテール表に追加されます。

```
ALTER TABLE sales ADD PARTITION month_new ...
ALTER TABLE sales EXCHANGE PARTITION month_new month_new_table
```

2. 次に、最も古いパーティションが削除されるか切り捨てられます。

```
ALTER TABLE sales DROP PARTITION month_oldest;
```

3. ここで、マテリアライズド・ビューがPCTリフレッシュの要件をすべて満たしているとします。

```
EXECUTE DBMS_MVIEW.REFRESH('CUST_MTH_SALES_MV', 'F', '', TRUE, FALSE, 0, 0, 0, FALSE, FALSE);
```

高速リフレッシュでは、PCTが使用可能であることが自動的に検出され、PCTリフレッシュが実行されます。

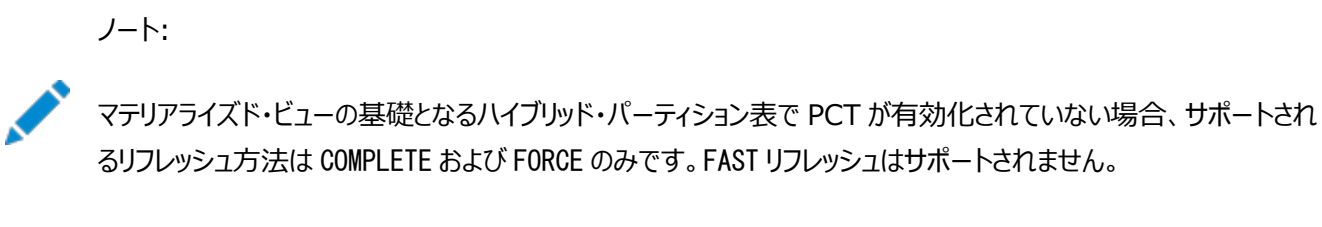
7.4 ハイブリッド・パーティション表に基づくマテリアライズド・ビューのリフレッシュ

ハイブリッド・パーティション表に基づくマテリアライズド・ビューをリフレッシュするには、完全リフレッシュ方法、高速リフレッシュ方法またはPCTリフレッシュ方法を使用できます。

Oracle Databaseでは外部ソースでのデータのメンテナンス方法を制御できないため、外部パーティションのデータが最新であることは保証されず、その最新状態はUNKNOWNとマークされます。外部パーティションのデータは、信頼できる整合性モードまたは失効許可モードでのみ使用できます。

外部パーティションから発生するデータのリフレッシュは、コストが高くなり、多くの場合は不要な操作となる(ソース・データが変更

されていない場合)可能性があります。DBMS_MVIEW.REFRESHプロシージャのskip_ext_data属性を使用して、外部パーティションに対応するマテリアライズド・ビュー・データのリフレッシュをスキップできます。この属性をTRUEに設定すると、外部パーティションに対応するマテリアライズド・ビュー・データは再計算されず、状態がUNKNOWNの信頼できるモードのままになります。デフォルトでは、skip_ext_dataはFALSEです。



ノート:

マテリアライズド・ビューの基礎となるハイブリッド・パーティション表で PCT が有効化されていない場合、サポートされるリフレッシュ方法は COMPLETE および FORCE のみです。FAST リフレッシュはサポートされません。

例7-7 **ハイブリッド・パーティション表に基づくマテリアライズド・ビューのリフレッシュ**

マテリアライズド・ビューhypt_mvの内部パーティションyear_2000が失効していると仮定します。このマテリアライズド・ビューは、ハイブリッド・パーティション表に基づいています。カタログ・ビューUSER_MVIEW_DETAIL_PARTITIONを問い合せると、次のように表示されます。

```
SELECT mview_name, detail_partition_name, freshness, last_refresh_time
        from USER_MVIEW_DETAIL_PARTITION;

MVIEW_NAME      DETAIL_PARTITION_NAME   FRESHNESS       LAST_REFRESH_TIME
----------      ---------------------   ---------       -----------------
HyPT_MV                 century_19                                  UNKNOWN         2016-10-31
20:48:00.20
HyPT_MV         century_20                                  UNKNOWN         2016-10-31 20:48:00.20
HyPT_MV                 year_2000                                       STALE
2016-10-31 20:48:00.20
HyPT_MV                 year_2001                                       FRESH
2016-10-31 20:48:00.20
```

次のコマンドを使用して、マテリアライズド・ビューの高速リフレッシュを実行します。

```
DBMS_MVIEW.REFERSH('HyPT_MV', 'F', skip_ext_data => false);
```

リフレッシュ後にカタログ・ビューUSER_MVIEW_DETAIL_PARTITIONを問い合せると、次のように表示されます。

```
SELECT mview_name, detail_partition_name, freshness, last_refresh_time
        from USER_MVIEW_DETAIL_PARTITION;

MVIEW_NAME      DETAIL_PARTITION_NAME   FRESHNESS       LAST_REFRESH_TIME
----------      ---------------------   ---------       -----------------
HyPT_MV                 century_19              UNKNOWN         2016-10-31 21:32:17.00
HyPT_MV         century_20              UNKNOWN         2016-10-31 21:32:17.00
HyPT_MV                 year_2000                       FRESH                   2016-10-31 21:32:17.00
HyPT_MV                 year_2001                       FRESH                   2016-10-31 20:48:00.20
```

内部パーティションyear_2000のみがリフレッシュされています。パーティションyear_2001は、すでに最新であるためリフレッシュされませんでした。skip_ext_dataがFALSEに設定されている場合、外部パーティションの完全リフレッシュおよび内部パーティションの高速リフレッシュが実行されます。

7.5 パーティション化によるデータ・ウェアハウス・リフレッシュの改善

ETL(抽出、変換、ロード)がスケジュールに基づいて実行され、オリジナルのソース・システムに対して行われた変更が反映されます。このステップ中に、新しいクリーン・データを本番データ・ウェアハウス・スキーマに物理的に挿入し、この新しいデータがエンド・ユーザーにも利用できるように、必要に応じてその他のステップ(索引の作成、制約の妥当性チェック、データのバックアップ作

成など)をすべて実行します。このデータがすべてデータ・ウェアハウスにロードされた後に、最新データが反映されるようにマテリアライズド・ビューを更新する必要があります。

データ・ウェアハウスのパーティション化方法によって、データ・ウェアハウスのロード・プロセスにおけるリフレッシュ操作の効率が決まります。実際、データ・ウェアハウスの表および索引をパーティション化する方法を選択するときには、ロード・プロセスが重要な考慮点となることがあります。

非常に大規模なデータ・ウェアハウス表(スター・スキーマのファクト表など)のパーティション化方法は、データ・ウェアハウスのロード・パラダイムをベースにする必要があります。

ほとんどのデータ・ウェアハウスには、新しいデータが定期的にロードされます。たとえば、毎晩、毎週、毎月などのペースで、データ・ウェアハウスに新しいデータが格納されます。週末または月末にロードされるデータは、通常、その週またはその月のトランザクションに対応しています。このように、非常に一般的なケースでは、データ・ウェアハウスは時間ごとにロードされます。そのため、データ・ウェアハウス表には、日付列でのパーティション化が適しています。たとえば、次のデータ・ウェアハウスの例では、新しいデータがsales表に毎月ロードされるとします。また、sales表は月別にパーティション化されているとします。表salesに新しい月(2001年1月)のデータを追加するロード・ステップは、次のようになります。

1. 別に用意したsales_01_2001表に新しいデータを格納します。このデータは、データ・ウェアハウスの外部から直接sales_01_2001にロードできます。また、前にデータ・ウェアハウスで実行されたデータ変換操作の結果をロードすることも可能です。sales_01_2001の列、データ型などは、sales表と同一です。このsales_01_2001表の統計情報データを収集します。
2. sales_01_2001に索引を作成し、制約を追加します。この場合も、sales_01_2001の索引および制約は、salesの索引および制約と同一とします。索引はパラレルで作成できます。また、NOLOGGINGおよびCOMPUTE STATISTICSオプションを使用する必要があります。次に例を示します。

   ```
   CREATE BITMAP INDEX sales_01_2001_customer_id_bix
     ON sales_01_2001(customer_id)
         TABLESPACE sales_idx NOLOGGING PARALLEL 8 COMPUTE STATISTICS;
   ```

   sales表に存在するすべての制約が、sales_01_2001表に適用される必要があります。これには参照整合性制約も含まれます。典型的な制約の例としては、次のものがあります。

   ```
   ALTER TABLE sales_01_2001 ADD CONSTRAINT sales_customer_id
         REFERENCES customer(customer_id) ENABLE NOVALIDATE;
   ```

   パーティション表salesにグローバル索引構造により強制適用される主キーまたは一意キーがある場合は、索引構造を作成せずに、次のようにしてsales_pk_jan01の制約が有効になるようにしてください。

   ```
   ALTER TABLE sales_01_2001 ADD CONSTRAINT sales_pk_jan01
   PRIMARY KEY (sales_transaction_id) DISABLE VALIDATE;
   ```

   ENABLE句で制約を作成すると、パーティション表のローカル索引構造と一致しない一意索引が作成されます。非パーティション表に、パーティション表の既存のグローバル索引と交換される索引構造を作成しないでください。EXCHANGEコマンドが失敗します。

3. sales_01_2001表をsales表に追加します。

   この新しいデータをsales表に追加するには、2つの操作が必要です。まず、sales表に新しいパーティションを追加します。ALTER TABLE ... ADD PARTITION文を使用します。これによって、sales表に空のパーティションが追加されます。

   ```
   ALTER TABLE sales ADD PARTITION sales_01_2001
   VALUES LESS THAN (TO_DATE('01-FEB-2001', 'DD-MON-YYYY'));
   ```

   この後、EXCHANGE PARTITION操作によって、新しく作成した表をこのパーティションに追加できます。この操作によって、

新しい空のパーティションが、新しくロードされた表と交換されます。

```
ALTER TABLE sales EXCHANGE PARTITION sales_01_2001 WITH TABLE sales_01_2001
INCLUDING INDEXES WITHOUT VALIDATION UPDATE GLOBAL INDEXES;
```

EXCHANGE操作は、sales_01_2001表にすでにあった索引および制約を保存します。一意制約(sales_transaction_idの一意制約など)の場合は、前述のようにUPDATE GLOBAL INDEXES句を使用できます。これにより、グローバル索引構造はパーティション・メンテナンス操作の一部として自動的にメンテナンスされ、プロセス全体でアクセス可能な状態に保たれます。外部キー制約のみの場合は、EXCHANGE操作はすぐに処理を終了します。

同期リフレッシュを使用する場合は、ステップ3を実行するかわりに、DBMS_SYNC_REFRESH.REGISTER_PARTITION_OPERATIONパッケージを使用してsales_01_2001表を登録する必要があります。詳細は、同期リフレッシュを参照してください。

このパーティション化テクニックには、重要な利点があります。第1に、新しいデータのロードに使用するリソースが最小限に抑えられます。新しいデータは、完全に別々の表にロードされるため、索引および制約の処理は、その新しいパーティションのみに適用されます。sales表が50GBで、12のパーティションを持つとすると、新しい月のデータ・サイズは約4GBになります。索引を作成する必要があるのは新しい月のデータのみです。残りの46GBのデータに関する索引は、まったく変更する必要はありません。このパーティション化方法では、ロード処理時間は、sales表全体のサイズではなく、新しいデータの量に比例します。

第2に、同時問合せへの影響を最小限に抑えて、新しいデータをロードできます。データのロードに関連する操作は、すべて別のsales_01_2001表に対して発生しています。したがって、sales表の既存のデータや索引はデータのリフレッシュ処理中にはまったく影響を受けません。このリフレッシュ処理の間、sales表およびその索引に、処理が加えられないようにできます。

第3に、グローバル索引が存在する場合は、交換コマンドの一部として段階的にメンテナンスされます。このメンテナンスは、既存のグローバル索引構造の可用性には影響しません。

EXCHANGE操作は、公開機能と見ることができます。データ・ウェアハウスの管理者がsales_01_2001表をsales表に交換するまで、エンド・ユーザーは新しいデータを参照できません。EXCHANGEが実行されると、その直後に、sales表にアクセスするすべてのエンド・ユーザー問合せからsales_01_2001データが参照できるようになります。

パーティション化は、新しいデータの追加のみでなく、データの削除やアーカイブにも有効です。多くのデータ・ウェアハウスがデータのローリング・ウィンドウを保持しています。たとえば、データ・ウェアハウスには最近36か月のsalesデータが格納されているとします。sales表に新しいパーティションを追加できるのと同じように(前述参照)、古いパーティションも即座に(かつ他に影響を及ぼさずに)sales表から削除できます。パーティションの追加には2つの効果(リソース使用の削減およびエンド・ユーザーへの影響の最小化)がありますが、これはパーティションの削除にも当てはまります。

パーティション表からデータを削除しても、必ずしも古いデータがデータベースから物理的に削除されるわけではありません。パーティション表からデータを削除するには、他に次の2つの方法があります。1つ目は、古いデータが含まれているパーティションを削除して割り当てられている領域を解放することで、データベースからすべてのデータを物理的に削除する方法です。

```
ALTER TABLE sales DROP PARTITION sales_01_1998;
```

2つ目は、古いパーティションを、同じ構造を持つ空の表と交換する方法です。この空の表は、ロード処理のステップ1と2で説明したのと同じステップで作成します。新しい空の表スタブの名前をsales_archive_01_1998とすると、次のSQL文でパーティションsales_01_1998が空になります。

```
ALTER TABLE sales EXCHANGE PARTITION sales_01_1998
WITH TABLE sales_archive_01_1998 INCLUDING INDEXES WITHOUT VALIDATION
UPDATE GLOBAL INDEXES;
```

古いデータは、交換された非パーティション表sales_archive_01_1998としてまだ存在していることに注意してください。

すべてのパーティションを別々の表領域に格納するという方法でパーティション表が設定されていた場合は、実際のデータ(表領域)を削除する前に、Oracle Databaseのトランスポータブル表領域フレームワークを使用して、この表をアーカイブ(または転送)できます。

場合によっては、古いデータをすぐに削除するのではなく、パーティション表の一部として保持することが必要な場合があります。このデータはもう重要ではありませんが、この古い読取り専用のデータにアクセスする問合せがまだ存在する可能性があります。古いデータの使用領域を最小化するには、Oracleのデータ圧縮を使用できます。1つ以上の圧縮パーティションがすでにパーティション表の一部に含まれていることも想定されます。

関連項目:

- トランスポータブル表領域の詳細は、トランスポータブル表領域を使用した転送を参照してください。
- 表の圧縮の詳細は、『Oracle Database管理者ガイド』を参照してください。
- パーティション化と表の圧縮については、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照してください。

7.5.1 データ・ウェアハウスのリフレッシュ・シナリオ

典型的な使用例では、古いデータを圧縮するのみでなく、いくつかの古いパーティションを将来のバックアップの最小処理単位にマージすることが必要な場合があります。バックアップ(パーティション)の最小処理単位がどの四半期についても四半期ベースであり、最も古い月と最新の月の差が36か月より多いものとします。この場合、sales_01_1998、sales_02_1998およびsales_03_1998を新しい圧縮パーティションsales_q1_1998に圧縮およびマージすることになります。

1. 別の表領域に新しいマージ・パーティションをパラレルに作成します。パーティションは、次のMERGE操作の一部として圧縮されます。

```
ALTER TABLE sales MERGE PARTITIONS sales_01_1998, sales_02_1998, sales_03_1998
 INTO PARTITION sales_q1_1998 TABLESPACE archive_q1_1998
COMPRESS UPDATE GLOBAL INDEXES PARALLEL 4;
```

2. パーティションのMERGE操作により、新しいマージ・パーティションに対するローカル索引は無効になります。したがってこれらを再構築する必要があります。

```
ALTER TABLE sales MODIFY PARTITION sales_q1_1998
REBUILD UNUSABLE LOCAL INDEXES;
```

かわりに、パーティション表の外部に新しい圧縮表を作成して、これと交換する方法も選択できます。どちらの方法でも、パフォーマンスと一時領域の使用量は同程度です。

1. マージされた新しい情報を格納する中間的な表を作成します。次の文は、デフォルトで、元の表からNOT NULL制約をすべて継承します。

```
CREATE TABLE sales_q1_1998_out TABLESPACE archive_q1_1998
NOLOGGING COMPRESS PARALLEL 4 AS SELECT * FROM sales
WHERE time_id >=  TO_DATE('01-JAN-1998','dd-mon-yyyy')
  AND time_id < TO_DATE('01-APR-1998','dd-mon-yyyy');
```

2. 既存の表salesの他に、表sales_q1_1998_outに対して等価な索引構造を作成します。
3. 既存の表salesを、新しい圧縮表sales_q1_1998_outと交換するために準備します。交換される表には、実際には3つのパーティションにまたがるデータが含まれているため、参照中のレンジ境界を持つ、一致するパーティションを1つ作成

する必要があります。既存のパーティションを2つ削除するだけで済みます。低い方のパーティションsales_01_1998とsales_02_1998を削除する必要があることに注意してください。レンジ・パーティションの下限は、常に前のパーティションの上限(上限を含まない)によって定義されます。

```
ALTER TABLE sales DROP PARTITION sales_01_1998;
ALTER TABLE sales DROP PARTITION sales_02_1998;
```

4. これで、表sales_q1_1998_outをパーティションsales_03_1998と交換できます。パーティションの名前から想定されるものとは異なり、この境界は1998年の第1四半期をカバーします。

```
ALTER TABLE sales EXCHANGE PARTITION sales_03_1998
WITH TABLE sales_q1_1998_out INCLUDING INDEXES WITHOUT VALIDATION
UPDATE GLOBAL INDEXES;
```

どちらの方法も、多少異なるビジネス・シナリオに適用されます。MERGE PARTITIONを使用する方法では、影響を受けるパーティションのローカル索引構造は無効になりますが、データは常時アクセス可能なまま保たれます。影響を受けるパーティションに対して、使用できない索引構造の1つを介してアクセスしようとすると、エラーが発生します。使用が制限される時間は、ローカル・ビットマップ索引構造を再作成するための時間とほぼ同程度になります。ほとんどの場合、これは無視できます。パーティション表のこの部分はそれほど頻繁にアクセスされないためです。

ただし、CTAS方法では索引構造が使用できない時間はゼロに近く短縮されますが、パーティション表にすべてのデータが揃わない特定の時間枠があります。これは、2つのパーティションを削除したためです。使用が制限される時間は、表を交換するための時間とほぼ同程度になります。グローバル索引の有無およびその数によって、この時間枠は変わります。既存のグローバル索引がない場合、この時間枠はほんの数秒です。

これらの例は、データ・ウェアハウスのローリング・ウィンドウのロードの使用例を単純化したものです。実際のデータ・ウェアハウスのリフレッシュ特性は、さらに複雑です。ただし、このローリング・ウィンドウを使用すると、リフレッシュ特性がさらに複雑になっても、十分な効果を得ることができます。

パーティション表に単一または複数の圧縮パーティションを初めて追加するときは、その前にローカル・ビットマップ索引をすべて削除するか使用不可にマークする必要があることに注意してください。最初に圧縮パーティションを追加した後は、圧縮パーティションに関するその後のどの操作においても追加の処置は必要ありません。これは、どのような方法で圧縮表に圧縮パーティションが追加されたかには関係ありません。

関連項目:

- パーティション化と表の圧縮については、『*Oracle Database VLDBおよびパーティショニング・ガイド*』を参照してください。
- パーティション化と表の圧縮の詳細は、『*Oracle Database*管理者ガイド』を参照してください。

7.5.2 データ・ウェアハウスのリフレッシュにパーティション化を使用する使用例

この項では、リフレッシュとともにパーティション化が使用される次の2つの典型的なシナリオについて説明します。

- データ・ウェアハウスのリフレッシュのためのパーティション化: 使用例1
- データ・ウェアハウスのリフレッシュのためのパーティション化: 使用例2

7.5.2.1 データ・ウェアハウスのリフレッシュのためのパーティション化: 使用例1

データは毎日ロードされます。ただし、データ・ウェアハウスには2年分のデータを格納するため、1日単位のパーティションは適切ではありません。

ソリューションは、週別または月別(適切な方)にパーティション化することです。INSERTを使用して、新しいデータを既存のパーティションに追加します。INSERT操作が影響するパーティションは1つのみなので、前述の利点はそのまま残ります。INSERT操作は、パーティションが表の一部である場合に行うことができます。単一パーティションへのINSERTはパラレル化できます。

```
INSERT /*+ APPEND*/ INTO sales PARTITION (sales_01_2001)
SELECT * FROM new_sales;
```

このsalesパーティションの索引もパラレルでメンテナンスされます。このメソッドのかわりにEXCHANGE操作を使用することもできます。そのためには、sales表のsales_01_2001パーティションを交換し、INSERT操作を使用します。この方法を使用するのは、索引をメンテナンスするよりも、削除して再作成する方が効率的な場合です。

7.5.2.2 データ・ウェアハウスのリフレッシュのためのパーティション化: 使用例2

新しいデータは、主に最近の日、週、月などのものから構成されますが、以前の期間のデータも含まれます。

解決策 1

パラレルSQL操作(CREATE TABLE ... AS SELECTなど)を使用して、新しいデータを以前の期間のデータから分離します。日付が古いデータは、他のテクニックを使用して別に処理します。

新しいデータは、必ず時間ベースであるとは限りません。データ・ウェアハウスがビジネス・ニーズに基づいて複数の業務系システムから新規データを受け取るようにすることもできます。たとえば、直接チャネルからの売上データが、間接チャネルからのデータとは別にデータ・ウェアハウスに格納されることもあります。また、業務上の理由から、直接データと間接データを別のパーティションに保存することも適しています。

解決策2

Oracleでは、コンポジット・レンジ-リスト・パーティション化をサポートしています。sales表の主要なパーティション化方法は、例に示すようにtime_idに基づいたレンジ・パーティション化にできます。ただし、サブパーティション化は、チャネル属性に基づいたリストです。これで各サブパーティションを互いに(それぞれ別個のチャネルごとに)独立にロードして、前述のローリング・ウィンドウ操作で追加できます。パーティション化方式は、最も最適化された方法でビジネス・ニーズを処理します。

7.6 リフレッシュ中のDML操作の最適化

DMLパフォーマンスは、次の方法で最適化できます。

- 効率的なMERGE操作の実装
- データ・ウェアハウスにおける参照整合性の維持
- データ・ウェアハウスからのデータのパージ

7.6.1 効率的なMERGE操作の実装

ソース・システムから抽出したデータは、データ・ウェアハウスに挿入する必要のある新しいレコードの単なるリストではありません。この新しいデータセットは、新しいレコードと変更レコードの組合せで構成されます。たとえば、OLTPシステムから抽出したデータのほとんどが新しい売上トランザクションによるものだとします。これらのレコードは、ウェアハウスのsales表に挿入されますが、その中には、商品の返品や、最初にデータ・ウェアハウスにロードしたときに不備があったトランザクションの修正など、以前のトランザクションに対する変更を反映しているものがある場合があります。このようなレコードについては、sales表の更新が必要です。

new_sales表があり、この表にsales表に適用される挿入項目と更新項目の両方が格納されている例を考えてみます。データ・ウェアハウスのロード・プロセス全体を設計するときに、次のような処理方法で、このnew_sales表にレコードを格納することにします。

- new_sales表のレコードの任意のsales_transaction_idがsales表にすでに存在する場合、new_sales表からsales_dollar_amountおよびsales_quantity_soldの値をsales表の既存の行に追加することで、sales表を更新します。
- それ以外の場合は、new_sales表から新しいレコード全体をsales表に挿入します。

このUPDATE-ELSE-INSERT操作は、通常はマージと呼ばれます。マージは、1つのSQL文で実行できます。

例7-8 MERGE操作

```
MERGE INTO sales s USING new_sales n
ON (s.sales_transaction_id = n.sales_transaction_id)
WHEN MATCHED THEN
UPDATE SET s.sales_quantity_sold = s.sales_quantity_sold + n.sales_quantity_sold,
 s.sales_dollar_amount = s.sales_dollar_amount + n.sales_dollar_amount
WHEN NOT MATCHED THEN INSERT (sales_transaction_id, sales_quantity_sold,
sales_dollar_amount)
VALUES (n.sales_transcation_id, n.sales_quantity_sold, n.sales_dollar_amount);
```

ターゲット表に対する無条件のUPDATE ELSE INSERT機能でMERGE文を使用することに加えて、次のことも可能です。

- UPDATEのみ、またはINSERTのみの文を実行する。
- 追加のWHERE条件をMERGE文のUPDATEまたはINSERT部分に適用する。
- 特定の条件がTRUEの場合、UPDATE操作で行を削除する。

例7-9 INSERT句の省略

一部のデータ・ウェアハウス・アプリケーションでは、新しい行を履歴情報に追加することはできず、更新のみが可能なものがあります。また、更新ではなく新しい情報の挿入のみが必要な場合があります。次の文は、UPDATEのみ使用してINSERT機能のみを実行する例です。

```
MERGE USING Product_Changes S     -- Source/Delta table
INTO Products D1                  -- Destination table 1
ON (D1.PROD_ID = S.PROD_ID)       -- Search/Join condition
WHEN MATCHED THEN UPDATE          -- update if join
SET D1.PROD_STATUS = S.PROD_NEW_STATUS
```

例7-10 UPDATE句の省略

次の文は、UPDATEを省略する例です。

```
MERGE USING New_Product S           -- Source/Delta table
INTO Products D2                    -- Destination table 2
ON (D2.PROD_ID = S.PROD_ID)         -- Search/Join condition
WHEN NOT MATCHED THEN               -- insert if no join
INSERT (PROD_ID, PROD_STATUS) VALUES (S.PROD_ID, S.PROD_NEW_STATUS)
```

INSERT句を省略した場合は、ソース表とターゲット表に対して通常の結合が実行されます。UPDATE句を省略した場合は、ソース表とターゲット表に対してアンチ結合が実行されます。これによって、ソース表とターゲット表がより効率的に結合されます。

例7-11 UPDATE句のスキップ

特定の行を表にマージする際に、UPDATE操作をスキップしなければならない場合があります。この場合、MERGEのUPDATE句にオプションのWHERE句を使用できます。その結果、UPDATE操作は、指定した条件がTRUEの場合にのみ実行されるようになります。

次の文は、UPDATE操作をスキップする例を示しています。

```
MERGE
USING Product_Changes S                        -- Source/Delta table
INTO Products P                                -- Destination table 1
ON (P.PROD_ID = S.PROD_ID)                     -- Search/Join condition
WHEN MATCHED THEN
UPDATE                                         -- update if join
SET P.PROD_LIST_PRICE = S.PROD_NEW_PRICE
WHERE P.PROD_STATUS <> "OBSOLETE"              -- Conditional UPDATE
```

次に、条件P.PROD_STATUS <> "OBSOLETE"がTRUEではない場合にUPDATE操作がどのようにスキップされるかを示します。条件の述語は、ターゲット表とソース表の両方を指すことができます。

例7-12 MERGE文の条件付き挿入

特定の行を表にマージする際に、INSERT操作をスキップしなければならない場合があります。この場合、MERGEのINSERT句にオプションのWHERE句を追加します。その結果、INSERT操作は、指定した条件がTRUEの場合にのみ実行されるようになります。次に例を示します。

```
MERGE USING Product_Changes S                        -- Source/Delta table
INTO Products P                                      -- Destination table 1
ON (P.PROD_ID = S.PROD_ID)                           -- Search/Join condition
WHEN MATCHED THEN UPDATE                             -- update if join
SET P.PROD_LIST_PRICE = S.PROD_NEW_PRICE
WHERE P.PROD_STATUS <> "OBSOLETE"                    -- Conditional
WHEN NOT MATCHED THEN
INSERT (PROD_ID, PROD_STATUS, PROD_LIST_PRICE)       -- insert if not join
VALUES (S.PROD_ID, S.PROD_NEW_STATUS, S.PROD_NEW_PRICE)
WHERE S.PROD_STATUS <> "OBSOLETE";                   -- Conditional INSERT
```

この例は、条件S.PROD_STATUS <> "OBSOLETE"がTRUEではない場合にINSERT操作がスキップされ、条件がTRUEのときにのみINSERTが実行されることを示します。条件の述語は、ソース表のみを指すことができます。条件の述語は、ソース表のみを指すことができます。

例7-13 MERGE文でのDELETE句の使用

表を移入または更新する際に、表をクレンジングする場合があります。この場合、次の例のように、MERGE文でDELETE句を使用できます。

```
MERGE USING Product_Changes S
INTO Products D ON (D.PROD_ID = S.PROD_ID)
WHEN MATCHED THEN
UPDATE SET D.PROD_LIST_PRICE =S.PROD_NEW_PRICE, D.PROD_STATUS = S.PROD_NEWSTATUS
DELETE WHERE (D.PROD_STATUS = "OBSOLETE")
WHEN NOT MATCHED THEN
INSERT (PROD_ID, PROD_LIST_PRICE, PROD_STATUS)
VALUES (S.PROD_ID, S.PROD_NEW_PRICE, S.PROD_NEW_STATUS);
```

このように、行がproductsで更新される際に、削除条件D.PROD_STATUS = "OBSOLETE"がチェックされ、条件がTRUEであれば行が削除されます。

DELETE操作は、完全なDELETE文の削除とは異なります。MERGE先の行のみが削除されます。DELETEの影響を受ける行のみが、このMERGE文で更新される行です。したがって、目的の表で指定した行が削除条件に合致しても、ON句の条件の下で結合を実行しない場合は削除されません。

例7-14 MERGE文の無条件の挿入

ソースのすべての行を表に挿入する場合があります。この場合、ソース表とターゲット表の結合を避けることができます。1=0など、常にFALSEになる特殊な一定の結合条件を指定することで、このようなMERGE文は最適化され、結合条件は無効になります。

```
MERGE USING New_Product S       -- Source/Delta table
INTO Products P                 -- Destination table 1
ON (1 = 0)                      -- Search/Join condition
WHEN NOT MATCHED THEN           -- insert if no join
INSERT (PROD_ID, PROD_STATUS) VALUES (S.PROD_ID, S.PROD_NEW_STATUS)
```

7.6.2 データ・ウェアハウスの参照整合性の維持

データ・ウェアハウス環境によっては、参照整合性を保証するために新しいデータを表に挿入する必要がある場合もあります。たとえば、キャッシュ・レジスタから直接データを取り出す業務系システムからsalesを導出するデータ・ウェアハウスがあるとします。salesは毎晩リフレッシュされます。ただし、productディメンション表のデータは別の業務系システムから導出されます。productディメンション表の変更には比較的時間がかかるため、この表は週に1回しかリフレッシュされないことがあります。新製品が月曜日に導入されたとすると、その製品のproduct_idがデータ・ウェアハウスのproduct表に挿入される前に、データ・ウェアハウスのsalesデータ内にproduct_idが表示される可能性があります。

この新製品の売上トランザクションは有効ですが、その売上データは、productディメンション表とsalesファクト表間の参照整合性制約を満たしません。この場合、新しい売上トランザクションを禁止するより、sales表にその売上トランザクションを挿入する方を選ぶのが普通です。ただし、sales表とproduct表間の参照整合性関係もメンテナンスする必要があります。これは、新しい行を不明な製品のプレースホルダとしてproduct表に挿入することによって可能になります。

前述の例のように、sales表への新しいデータは別のnew_sales表に格納されるとします。パラレル化が可能な単一のINSERT文で、product表が新製品を反映するように変更できます。

```
INSERT INTO product
  (SELECT sales_product_id, 'Unknown Product Name', NULL, NULL ...
   FROM new_sales WHERE sales_product_id NOT IN
  (SELECT product_id FROM product));
```

7.6.3 データ・ウェアハウスからのデータのパージ

データ・ウェアハウスから大量のデータを削除する必要がある場合もあります。前述のローリング・ウィンドウでは、古いデータをデータ・ウェアハウスからロールアウトして新しいデータの領域を確保するという、非常に一般的な例を取り上げました。

ただし、それ以外の場合でも、データをデータ・ウェアハウスから削除する必要がある場合があります。たとえば、ある小売会社が、以前にXYZ Software社の製品を販売し、その後XYZ Softwareが廃業したとします。データ・ウェアハウスを業務で利用しているユーザーが、XYZ Software社に関するデータはもう必要ないと判断し、このデータを削除することになりました。

大量のデータを削除する方法の1つに、パラレル削除による方法があります。例を次に示します。

```
DELETE FROM sales WHERE sales_product_id IN (SELECT product_id
   FROM product WHERE product_category = 'XYZ Software');
```

このSQL文では、パーティションごとに1つのパラレル処理が起動されます。このアプローチは一連のDELETE文よりもはるかに効率的であり、sales表内のデータを移動する必要はありません。ただし、いくつかのデメリットもあります。行の大部分を削除する場合、DELETE文は既存のパーティションに多数の行スロットを残します。その後、新しいデータがローリング・ウィンドウ・テクニックによってロードされても(またはダイレクト・パス・インサートまたはダイレクト・パス・ロードによってロードされても)、この記憶領域は再使用されません。また、DELETE文はパラレル化できますが、それより効率的な方法もあります。別の方法としては、XYZ Software社以外の製品カテゴリのデータをすべて維持したまま、sales表全体を再作成する方法があります。

```
CREATE TABLE sales2 AS SELECT * FROM sales, product
WHERE sales.sales_product_id = product.product_id
```

```
AND product_category <> 'XYZ Software'
NOLOGGING PARALLEL (DEGREE 8)
#PARTITION ... ; #create indexes, constraints, and so on
DROP TABLE SALES;
RENAME SALES2 TO SALES;
```

この方法は、パラレルDELETEより効率的です。ただし、sales表を2回インスタンス化することになるため、この方法もディスク領域の使用量の面ではコストが高くなります。

ディスク領域の使用量が少ない別の方法として、sales表を一度に1パーティションずつ再作成する方法があります。

```
CREATE TABLE sales_temp AS SELECT * FROM sales WHERE 1=0;
INSERT INTO sales_temp
SELECT * FROM sales PARTITION (sales_99jan), product
WHERE sales.sales_product_id = product.product_id
AND product_category <> 'XYZ Software';
<create appropriate indexes and constraints on sales_temp>
ALTER TABLE sales EXCHANGE PARTITION sales_99jan WITH TABLE sales_temp;
```

sales表の各パーティションに対して、このプロセスを繰り返します。

8 同期リフレッシュ

この章では、データ・ウェアハウス内で表とマテリアライズド・ビューへの変更を同期させる方法について説明します。この方法は、表とマテリアライズド・ビューに対する更新の同期に基づいており、同期リフレッシュと呼ばれます。

この章の内容は次のとおりです。

- マテリアライズド・ビューの同期リフレッシュについて
- マテリアライズド・ビューでの同期リフレッシュの使用
- 同期リフレッシュ・グループの使用方法
- 同期リフレッシュのための変更データの指定と準備
- 同期リフレッシュ操作のトラブルシューティング
- 同期リフレッシュ適格性分析の実行
- 同期リフレッシュのセキュリティに関する考慮事項の概要

8.1 マテリアライズド・ビューの同期リフレッシュについて

同期リフレッシュは、Oracle Database 12*c*リリース1で導入されたリフレッシュ方法で、表一式とそこで定義されたマテリアライズド・ビューが常に同期した状態を維持できるようにします。ここでは、増分データのロードが厳密に制御され、定期的に発生するデータ・ウェアハウスには非常に適しています。

ほとんどのデータ・ウェアハウスでは、ファクト表は時間ディメンションに沿ってパーティション化され、通常増分データ・ロードは主に最近の期間に対する変更で構成されます。同期リフレッシュはこのような特性を利用し、リフレッシュのパフォーマンスおよびスループットを大幅に向上させます。これにより、計画済問合せと非定型問合せの両方において、問合せパフォーマンスが高速になります。これはデータ・ウェアハウスを成功させるために重要なことです。

この項では、同期リフレッシュの主要な要件と基本的な概念について説明します。内容は次のとおりです。

- 同期リフレッシュとは
- 同期リフレッシュを使用する理由
- 同期リフレッシュのための表とマテリアライズド・ビューの登録
- リフレッシュのための変更データの指定
- 同期リフレッシュの準備と実行
- 同期リフレッシュのためのマテリアライズド・ビューの適格性ルールと制限事項

8.1.1 同期リフレッシュとは

同期リフレッシュはデータ・ウェアハウスにおける表とマテリアライズド・ビューの新しい管理方法で、これによって表とマテリアライズド・ビューが同時にリフレッシュされます。従来のリフレッシュ方法の場合、変更は実表に適用され、マテリアライズド・ビューは次のいずれかのリフレッシュ方法で別個にリフレッシュされます。

- マテリアライズド・ビューのログが使用できる場合、これを使用したログベースの増分(高速)リフレッシュ
- PCTリフレッシュ(使用可能な場合)
- 完全リフレッシュ

同期リフレッシュは、ログベースの増分(高速)リフレッシュとPCTリフレッシュの手法のいくつかの要素を組み合せたものですが、これら2つの方法とは異なり、ON DEMANDマテリアライズド・ビューに対してのみ適用可能です。同期リフレッシュと他のリフレッシュ方法には、主に次の3つの相違点があります。

- 同期リフレッシュでは、表とマテリアライズド・ビューの登録が必要です。
- 同期リフレッシュでは、以前指定したルールに従ってデータへの変更を指定する必要があります。
- 同期リフレッシュは、リフレッシュ操作を準備と実行の2つのステップに分割して機能します。このアプローチにより、他の方法に比べ、パフォーマンスが優れている、詳細に制御できるなどの重要な利点が得られます。

同期リフレッシュAPIは、DBMS_SYNC_REFRESHという新しいパッケージで定義されています。このパッケージの詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください。

8.1.2 同期リフレッシュを使用する理由

同期リフレッシュでは、データ・ウェアハウスでマテリアライズド・ビューをリフレッシュするために使用される従来のタイプのリフレッシュ方法に対し、次のような利点があります。

- 実表への変更のロードと、それ自体独立したマテリアライズド・ビューの非常に効果的なリフレッシュを連携させます。
- オプティマイザがクエリーをリライトするためにマテリアライズド・ビューを使用できない時間を減らします。
- これは、データ・ウェアハウスで一般的に使用されるマテリアライズド・ビューの幅広いクラス(マテリアライズド集計ビューおよびマテリアライズド結合ビュー)に適しています。ファクト表と同様にマテリアライズド・ビューをパーティション化する必要があります。マテリアライズド・ビューが現在パーティション化されていない場合は、同期リフレッシュを利用できるように効果的にパーティション化できます。
- パーティション化およびデータ・ウェアハウスのロード・サイクルの性質を十分に活用して、リフレッシュ・プロシージャを通してマテリアライズド・ビューと実表間の同期を保証します。
- 一般的なデータ・ウェアハウスでは、データの準備として、1つ以上のソースからのデータの抽出、クレンジング、整合性のためのフォーマットおよびデータ・ウェアハウス・スキーマへの変換があります。データの準備領域はステージング領域と呼ばれ、データ・ウェアハウスの実表がステージング領域の表からロードされます。同期リフレッシュ方法では、変更データをステージング・ログにロードできるため、この方法はこのモデルに適しています。
- ステージング・ログは、従来の高速リフレッシュ方法のマテリアライズド・ビュー・ログと同じ役割を果たします。ただし、重要な相違点が1つあります。従来の高速リフレッシュ方法では、まず実表が更新され、その後変更がマテリアライズド・ビュー・ログからマテリアライズド・ビューに適用されます。ところが同期リフレッシュ方法では、ステージング・ログからの変更がマテリアライズド・ビューのリフレッシュに適用される一方、実表にもこれが適用されます。
- データ・ウェアハウス内のほとんどのマテリアライズド・ビューは通常、ファクト表およびディメンション表が外部キーと主キーの関係で結合されているスター・スキーマまたはスノーフレーク・スキーマを採用します。同期リフレッシュ方法では、ファクト表にのみ行が追加されるものから、ファクト表とディメンション表への無作為な変更まで、考えられるすべての変更データ・ロード・シナリオにおいて、両方のスキーマを処理できます。
- ステージング・ログに変更ロード・データを指定するかわりに、影響を受けるパーティションと交換するデータを含む外部表の形式で変更データを直接実表に指定することもできます。この機能は、DBMS_SYNC_REFRESHパッケージのREGISTER_PARTITION_OPERATIONプロシージャによって提供されます。

8.1.3 同期リフレッシュのための表とマテリアライズド・ビューの登録

同期リフレッシュを実際に実行する前に、該当する表とマテリアライズド・ビューを登録する必要があります。同期リフレッシュでは、表とマテリアライズド・ビューを登録するため、次の方法を提供しています。

- 表は、そこに**ステージング・ログ**を作成することによって、同期リフレッシュに登録されます。ステージング・ログは、CREATE MATERIALIZED VIEW LOG文を使用して作成されます。この構文は、従来の増分リフレッシュに使用されるなじみのあるマテリアライズド・ビュー・ログに加え、ステージング・ログも作成するよう、このリリースで拡張されました。表にステージング・ログを作成すると、同期リフレッシュに登録されたと見なされ、同期リフレッシュ・プロシージャによってのみ変更可能となります。言い換えると、ステージング・ログが定義されている表は同期リフレッシュに登録され、ユーザーが直接変更することはできません。
- マテリアライズド・ビューは、DBMS_SYNC_REFRESHパッケージのREGISTER_MVIEWSプロシージャを使用して同期リフレッシュに登録されます。REGISTER_MVIEWSプロシージャは、暗黙的に同期リフレッシュ・グループと呼ばれる関連オブジェクトのグループを作成します。**同期リフレッシュ・グループ**は、互いに依存しているため単一のエンティティとして一緒にリフレッシュする必要のあるすべての関連マテリアライズド・ビューおよび表で構成されます。

関連項目:

- CREATE MATERIALIZED VIEW LOG文の詳細は、Oracle Database SQL言語リファレンスを参照してください。
- DBMS_SYNC_REFRESHパッケージの詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください。

8.1.4 リフレッシュのための変更データの指定

他のリフレッシュ方法の場合、マテリアライズド・ビューの実表を直接変更でき、変更データの指定の問題は生じません。しかし同期リフレッシュでは、以前指定した特定のルールに従い、DBMS_SYNC_REFRESHパッケージにより提供されるAPIを使用して、変更データを指定および準備する必要があります。

変更データを指定するには、次の2つの方法があります。

- 外部表に変更データを指定し、これをREGISTER_PARTITION_OPERATIONプロシージャに登録します。

  詳細は、同期リフレッシュの変更データの取得時におけるパーティション操作の使用を参照してください。
- ステージング・ログに変更データを指定し、これらをPREPARE_STAGING_LOGプロシージャで処理します。ステージング・ログのフォーマットおよび移入に関するルールは、同期リフレッシュの変更データの取得時におけるステージング・ログの使用で説明されています。表ごとに、リフレッシュ操作を実行する前に、PREPARE_STAGING_LOGプロシージャを実行する必要があります。

8.1.5 同期リフレッシュの準備と実行

変更データの準備ができたら、実際のリフレッシュ操作を実行できます。同期リフレッシュでは、新しいリフレッシュ実行方法を取ります。これは、リフレッシュ操作を準備と実行の2つのステップに分割して機能します。これは、同期リフレッシュと他のリフレッシュ方法の主な相違点の1つで、重要ないくつかの利点があります。

準備ステップでは、ファクト表のパーティションとマテリアライズド・ビューのパーティション間のマッピングを決定します。このステップでは、増分変更データのロードによって変更されたファクト表のパーティションのみに対応する新しい表を計算します。**外部表**と呼ばれるこれらの表が計算されると、実行ステップで実際にリフレッシュ操作が行われます。この操作は、外部表と、ファクト表またはマテリアライズド・ビュー内の該当するパーティションの交換です。

リフレッシュ実行ステップを2つのフェーズに分割し、それぞれに別のプロシージャを与えることにより、同期リフレッシュでは、リフレッシュ実行プロセスを制御できるだけでなく、システム全体のパフォーマンスを向上させることができます。これは、リフレッシュ・プロセスによって変更されるためダイレクト・アクセスまたはオプティマイザがマテリアライズド・ビューを使用できない時間を最小限に抑え

ることよって実現しています。準備フェーズでは、マテリアライズド・ビューとその表は変更されません。これは、この時点ではすべてのリフレッシュ変更が外部表に記録されるためです。したがって、マテリアライズド・ビューは、これを読み取る必要のあるすべての問合せで使用可能です。表とマテリアライズド・ビューが変更されるのは実行の間のみです。実行パフォーマンスは、主にディメンション表への変更の数の影響を受けます。この数が少ない場合、パーティション交換操作自体は非常に高速であるため、パフォーマンスは非常に良好になります。

DBMS_SYNC_REFRESHパッケージでは、これら2つのステップを実行するPREPARE_REFRESHプロシージャとEXECUTE_REFRESHプロシージャを提供しています。

関連項目:

- Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス

8.1.6 同期リフレッシュのためのマテリアライズド・ビューの適格性ルールと制限事項

マテリアライズド・ビューを同期リフレッシュで使用できるようにするための主な要件として、マテリアライズド・ビューは、そのファクト表のパーティション・キーから導出できるキーでパーティション化される必要があります。以降の項では、同期リフレッシュを使用するためのその他の要件について説明します。

この項では、次の項目について説明します。

- 同期リフレッシュの制限: パーティション化
- 同期リフレッシュの制限: リフレッシュ・オプション
- 同期リフレッシュの制限: 制約
- 同期リフレッシュの制限: 表
- 同期リフレッシュの制限: マテリアライズド・ビュー
- 同期リフレッシュの制限: 集計を含むマテリアライズド・ビュー

8.1.6.1 同期リフレッシュの制限: パーティション化

同期リフレッシュを使用するためには、主に2つの要件があります。

- マテリアライズド・ビューは、ファクト表と同じディメンションでパーティション化される必要があります。
- ファクト表のパーティション・キーが機能的にマテリアライズド・ビューのパーティション・キーを決定する必要があります。

機能的に決定するとは、外部キー制約関係に基づき、マテリアライズド・ビューのパーティション・キーはファクト表のパーティション・キーから導出できることを意味します。この条件が満たされるのは、マテリアライズド・ビューのパーティション・キーが、ファクト表のパーティション・キーと同じであるか、スター・スキーマやスノーフレーク・スキーマのようにファクト表とディメンション表の結合により関連付けられている場合です。たとえば、ファクト表がTIME_KEYなどの日付列でパーティション化されている場合、マテリアライズド・ビューは、TIME_KEY、MONTH、YEARなどでパーティション化できます。

同期リフレッシュは、ファクト表とマテリアライズド・ビューにおいて2種類のパーティション化(最上位のパーティション化タイプがレンジの場合、レンジ・パーティション化とコンポジット・パーティション化)をサポートしています。

8.1.6.2 同期リフレッシュの制限: リフレッシュ・オプション

マテリアライズド・ビューを定義する際は、リフレッシュ方法、トラステッド制約を使用可能にするかどうか、実行するリフレッシュのタイプの3つのリフレッシュ・オプションを指定できます。オプションを指定しない場合は、デフォルトとして、それぞれON DEMAND、

ENFORCED制約、FORCEが使用されます。同期リフレッシュの場合、これらのオプションの最初の2つの値は、それぞれON DEMANDとTRUSTED制約である必要があります。同期リフレッシュでは、リフレッシュのタイプには特定の値を必要としません。したがって、FAST、FORCE、COMPLETEのいずれでもかまいません。

8.1.6.3 同期リフレッシュの制限: 制約

ファクト表とディメンション表の関係は、表の外部キーおよび主キー制約によって宣言されます。同期リフレッシュは、リフレッシュの実行の際にこれらの制約を信頼し、マテリアライズド・ビューの定義でUSING TRUSTED CONSTRAINTSが指定されている必要があります。これによって、リフレッシュの際、UNKNOWNまたはFRESHの状態で、妥当性チェックが行われていないRELY制約の使用およびマテリアライズド・ビューに対するリライトが許可されます。

表が同期リフレッシュ用に登録されている場合、その制約はVALIDATEまたはNOVALIDATEの状態です。表がディメンション表の場合、同期リフレッシュでは、リフレッシュ実行プロセスの間、この状態が維持されます。

ただし、表がファクト表の場合は、リフレッシュ実行の間、同期リフレッシュにより制約がNOVALIDATEの状態にマークされます。これにより、同期リフレッシュ方法の基本であるパーティション交換の間、既存のデータで制約を検証する必要がなくなり、リフレッシュ実行のパフォーマンスが向上します。

同期リフレッシュではファクト表での制約は強制されないため、ユーザー自身が提供されたデータの整合性を検証する必要があります。

8.1.6.4 同期リフレッシュの制限: 表

同期リフレッシュで使用できるようにするには、表は次の条件を満たす必要があります。

- 表にはVPDやトリガーを定義できません。
- 表にRAW型を含めることはできません。
- 表はリモートにできません。
- 同期リフレッシュ用に登録された各表のステージング・ログ・キーは、ステージング・ログ・キーについてで説明する要件を満たす必要があります。

8.1.6.5 同期リフレッシュの制限: マテリアライズド・ビュー

他にも、同期リフレッシュ用に登録されたマテリアライズド・ビューに固有の制約がいくつかあります。

- 問合せの定義にROWID列を使用することはできません。これは、元のパーティションを外部表と置き換えるパーティション交換を使用するため、関係ありません。したがって、問合せの定義にROWID列は含めません。
- 同期リフレッシュでは、ネステッド・マテリアライズド・ビュー、UNION ALLマテリアライズド・ビュー、副問合せ、マテリアライズド・ビュー定義での複雑な問合せはサポートしていません。問合せの定義は、スター・スキーマまたはスノーフレーク・スキーマに準拠している必要があります。
- SQLコンストラクトの分析ウィンドウ関数(RANKなど)、MODEL句およびCONNECT BY句もサポートされていません。
- ビュー、リモート表または外部結合を参照するマテリアライズド・ビューでは、同期リフレッシュはサポートされません。
- マテリアライズド・ビューには、SYSDATEやROWNUMなど、結果の再現が不可能な式への参照を含めることはできません。

一般に、PCTリフレッシュ、高速リフレッシュおよび一般的なクエリー・リライトに適用される制限のほとんどが、同期リフレッシュにも適用されます。これらの制限は次の項を参照してください。

- マテリアライズド・ビューでのクエリー・リライトの制限について
- 一般的なクエリー・リライトの制限

- 高速リフレッシュにおける一般的な制限

8.1.6.6 同期リフレッシュの制限: 集計を含むマテリアライズド・ビュー

集計を含むマテリアライズド・ビューの場合、同期リフレッシュでは、高速リフレッシュと同様に次の制限があります。

- サポートされているのは、SUM、COUNT、AVG、STDDEV、VARIANCE、MINおよびMAXのみです。
- COUNT(*)を指定する必要があります。
- 集計関数は、式の最も外側でのみ使用する必要があります。つまり、AVG(AVG(x))やAVG(x)+ AVG(x)などの集計は実行できません。
- AVG(expr)などの集計ごとに、対応するCOUNT(expr)が存在している必要があります。さらにSUM(expr)を指定することをお薦めします。
- VARIANCE(expr)またはSTDDEV(expr)が指定された場合は、COUNT(expr)およびSUM(expr)を指定する必要があります。さらにSUM(expr *expr)を指定することをお薦めします。

8.2 マテリアライズド・ビューでの同期リフレッシュの使用

同期リフレッシュは、様々な点で他のリフレッシュ方法とは異なります。1つは、他のリフレッシュ方法はDBMS_MVIEWパッケージで宣言されるのに対し、同期リフレッシュのAPIはDBMS_SYNC_REFRESHと呼ばれる新しいパッケージに含まれる点です。もう1つは、オブジェクトが一度同期リフレッシュに登録されると、これらに他のリフレッシュ方法を使用できなくなる点です。

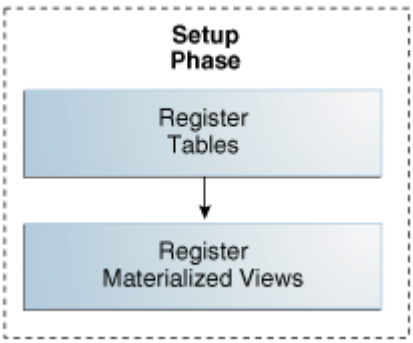
同期リフレッシュに関連する操作は、大まかに次の3つのフェーズに分割できます。

- 同期リフレッシュ・ステップ1: 登録フェーズ
- 同期リフレッシュ・ステップ2: 同期リフレッシュ・フェーズ
- 同期リフレッシュ・ステップ3: 登録解除フェーズ

8.2.1 同期リフレッシュのステップ1: 登録フェーズ

このフェーズでは(図8-1)、使用するオブジェクトを同期リフレッシュに登録します。このフェーズの2つのステップでは、まず表を登録し、次にマテリアライズド・ビューを登録します。表はステージング・ログを作成することで登録し、マテリアライズド・ビューはREGISTER_MVIEWSプロシージャによって登録します。ステージング・ログは、CREATE MATERIALIZED LOG … FOR SYNCHRONOUS REFRESH文を使用して作成されます。表にすでに通常のマテリアライズド・ビュー・ログがある場合は、ALTER MATERIALIZED LOG … FOR SYNCHRONOUS REFRESH文を使用して、これをステージング・ログに変換できます。

図8-1 登録フェーズ



例8-1に示すような文を使用してステージング・ログを作成できます。

例8-1 表の登録

```
CREATE MATERIALIZED VIEW LOG ON fact
```

```
FOR SYNCHRONOUS REFRESH USING st_fact;
```

表にマテリアライズド・ビューがある場合、次のような文を使用して、これをステージング・ログに変更できます。

```
ALTER MATERIALIZED VIEW LOG ON fact
FOR SYNCHRONOUS REFRESH USING st_fact;
```

例8-2に示すような文を使用してマテリアライズド・ビューを登録できます。

例8-2 マテリアライズド・ビューの登録

```
EXECUTE DBMS_SYNC_REFRESH.REGISTER_MVIEWS('MV1');
```

複数のマテリアライズド・ビューを一度に登録できます。

```
EXECUTE DBMS_SYNC_REFRESH.REGISTER_MVIEWS('mv2, mv2_year, mv1_halfmonth');
```

8.2.2 同期リフレッシュのステップ2: 同期リフレッシュ・フェーズ

図8-2に、同期リフレッシュ・フェーズを示します。このフェーズを繰り返し使用して、同期リフレッシュを実行できます。このフェーズの主な3つのステップは次のとおりです。

1. リフレッシュ操作のための変更データを準備します。表に変更データを指定し、これをREGISTER_PARTITION_OPERATIONプロシージャに登録するか、ステージング・ログにデータを移入して提供することができます。ステージング・ログは、次のステップに進む前にPREPARE_STAGING_LOGプロシージャで処理する必要があります。

   例8-12に例を示します。

2. リフレッシュ操作の最初のステップを実行します(PREPARE_REFRESH)。この操作は、外部表の準備およびロードを行うため、時間がかかる可能性があります。

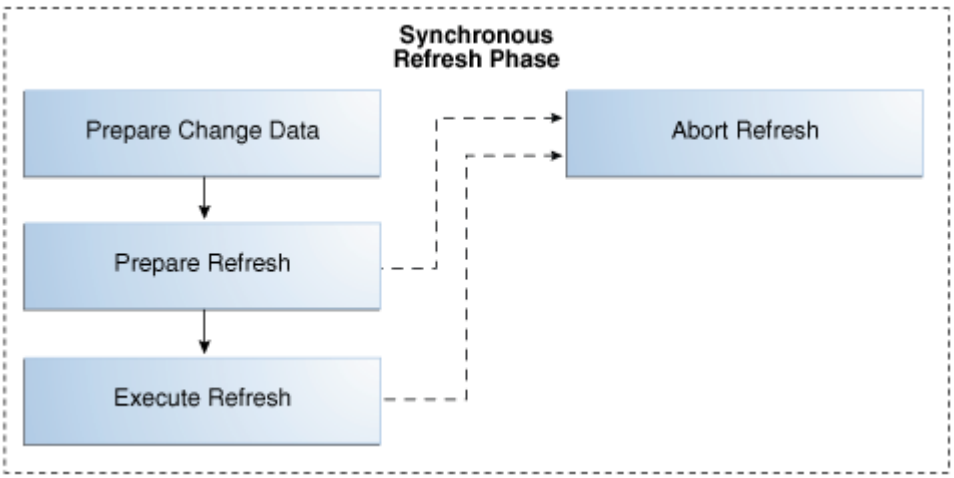
   例8-16に例を示します。

3. リフレッシュ操作の2番目かつ最後のステップを実行します(EXECUTE_REFRESH)。これは通常一連のパーティション交換操作で構成されるため、非常に速く実行されます。

   例8-20に例を示します。

図8-2において、実線の矢印は標準の制御フローを示し、エラー処理の場合には破線の矢印が使用されています。いずれかのリフレッシュ操作(PREPARE_REFRESHまたはEXECUTE_REFRESH)でユーザー・エラーが生じた場合は、ABORT_REFRESHプロシージャを使用して表およびマテリアライズド・ビューをリフレッシュ操作の前の状態に戻し、問題を修正して、もう一度最初からリフレッシュ操作を行います。

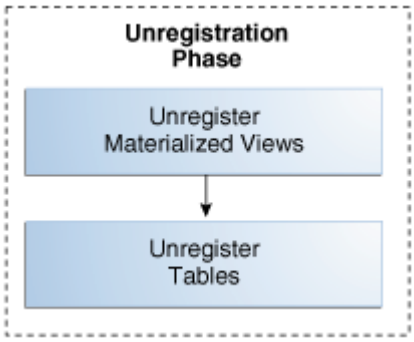
図8-2 リフレッシュ・フェーズ



8.2.3 同期リフレッシュのステップ3: 登録解除フェーズ

同期リフレッシュの使用を停止することにした場合、図8-3に示すとおり、マテリアライズド・ビューを登録解除する必要があります。まず、UNREGISTER_MVIEWSプロシージャを使用してマテリアライズド・ビューを登録解除します。次に、ステージング・ログを削除するか、通常のログに変更することによって、表を登録解除します。ALTER MATERIALIZED LOG … FOR FAST REFRESH文を使用してステージング・ログを通常のマテリアライズド・ビュー・ログに変換する場合、マテリアライズド・ビューは標準の高速リフレッシュ方法で管理できます。

図8-3 登録解除フェーズ



例8-3は、単一のマテリアライズド・ビューMV1を登録解除する方法を示しています。

例8-3 マテリアライズド・ビューの登録解除

```
EXECUTE DBMS_SYNC_REFRESH.UNREGISTER_MVIEWS('MV1');
```

次の方法では複数のマテリアライズド・ビューを一度に登録解除できます。

```
EXECUTE DBMS_SYNC_REFRESH.UNREGISTER_MVIEWS('mv2, mv2_year, mv1_halfmonth');
```

DBA_SR_OBJ_ALLビューを問い合せることで、マテリアライズド・ビューが登録解除されたかどうかを確認できます。

例8-4は、ステージング・ログの削除方法を示しています。

例8-4 表の登録解除

```
DROP MATERIALIZED VIEW LOG ON fact;
```

次のようにして表をマテリアライズド・ビュー・ログに変更することもできます。

```
ALTER MATERIALIZED VIEW LOG ON fact
FOR FAST REFRESH;
```

DBA_SR_OBJ_ALLビューを問い合せることで、表が登録解除されたかどうかを確認できます。

8.3 同期リフレッシュ・グループの使用方法

同期リフレッシュの際立った特徴として、表およびそのマテリアライズド・ビューへの変更が一緒にロードされリフレッシュされることがあげられます。このため同期リフレッシュという名前になっています。同期リフレッシュで管理される表およびマテリアライズド・ビューでは、オブジェクトを登録する必要があります。表は、ステージング・ログが作成されると同期リフレッシュに対して登録され、マテリアライズド・ビューはREGISTER_MVIEWSプロシージャにより登録されます。

同期リフレッシュでは、複数の表に構築されたマテリアライズド・ビュー(これらのうち1つ以上に変更がある)のリフレッシュをサポートしています。制約によって関連付けられている表は、データの整合性を保証するにはすべて一緒にリフレッシュする必要があります。さらに、同期リフレッシュに対して登録されている表の最上位にいくつかのマテリアライズド・ビューを構築することが可能ですが、その場合、これらのマテリアライズド・ビューは一緒にリフレッシュする必要があります。

ユーザーがこれらの依存性を追跡し、適切な表のセットでリフレッシュ・コマンドを発行するかわりに、Oracle Databaseでは一緒にリフレッシュする必要のある最低限の表とマテリアライズド・ビューのセットを自動生成します。これらのセットは同期リフレッシュ・グループと呼ばれます。各同期リフレッシュ・グループは、GROUP_ID値で識別されます。

同期リフレッシュの実行に関連する3つのプロシージャ(PREPARE_REFRESH、EXECUTE_REFRESHおよびABORT_REFRESH)は、入力として、同期リフレッシュ・グループを識別する単一のグループIDまたはグループIDのリストを取ります。

同期リフレッシュに対して登録された各表またはマテリアライズド・ビューにはGROUP_ID値が割り当てられます。これは依存関係が変わると変更される可能性があります。これは、REGISTER_MVIEWSプロシージャおよびUNREGISTER_MVIEWSプロシージャを発行すると発生します。後続の例は、様々なシナリオにおける同期リフレッシュ・グループを示しています。

GROUP_ID値は時とともに変化する可能性があるため、同期リフレッシュ・プロシージャの起動の際には、実際のGROUP_ID値は使用せず、かわりにDBMS_SYNC_REFRESH.GET_GROUP_ID関数を使用することをお薦めします。この関数は入力としてマテリアライズド・ビュー名を取り、マテリアライズド・ビューのGROUP_ID値を戻します。

関連項目:

DBMS_SYNC_REFRESH.REGISTER_MVIEWSプロシージャの使用方法については、『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

この項では、次の項目について説明します。

- 同期リフレッシュ・グループでの一般的なアクションの例
- 複数の同期リフレッシュ・グループの使用例

8.3.1 同期リフレッシュ・グループでの一般的なアクションの例

rdbms/demoディレクトリの同期リフレッシュ・デモ・スクリプトでは、実行されることの多い一般的な操作を見ることができます。メイン・スクリプトはsyncref_run.sqlで、そのログはsyncref_run.logです。次の例8-5、例8-6および例8-7は、GET_GROUP_ID関数を使用できる、異なるコンテキストを示しています。

例8-5 グループに登録されたオブジェクトの表示

この例は、グループに登録されたオブジェクトを表示する方法を示しています。

```
EXECUTE DBMS_SYNC_REFRESH.REGISTER_MVIEWS('MV1');
SELECT NAME, TYPE, STAGING_LOG_NAME FROM USER_SR_OBJ
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1')
ORDER BY TYPE, NAME;

NAME       TYPE       STAGING_LOG_NAME
---------- ---------- ----------------
MV1        MVIEW
FACT       TABLE      ST_FACT
STORE      TABLE      ST_STORE
TIME       TABLE      ST_TIME
```

例8-6 リフレッシュ操作の起動

この例は、リフレッシュ操作の起動方法を示しています。

```
EXECUTE DBMS_SYNC_REFRESH.PREPARE_REFRESH( -
        DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));
EXECUTE DBMS_SYNC_REFRESH.EXECUTE_REFRESH( -
```

```
        DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));
SELECT NAME, TYPE, STATUS FROM USER_SR_OBJ_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1')
ORDER BY TYPE, NAME;
```

例8-7 グループに登録されたオブジェクトのステータスの確認

この例は、EXECUTE_REFRESH操作の後に、グループに登録されたオブジェクトのステータスを確認する方法を示しています。

```
SELECT NAME, TYPE, STATUS FROM USER_SR_OBJ_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1')
ORDER BY TYPE, NAME;

NAME            TYPE       STATUS
--------------- ---------- ----------------
MV1             MVIEW      COMPLETE
FACT            TABLE      COMPLETE
STORE           TABLE      COMPLETE
TIME            TABLE      COMPLETE
```

8.3.2 複数の同期リフレッシュ・グループの使用例

次のAPIを使用して一度に複数のリフレッシュ・グループを使用できます。

- GET_GROUP_ID_LIST

  入力にマテリアライズド・ビューのリストを取り、それらのグループIDをリストで戻します。

- GET_ALL_GROUP_IDS

  システム内のすべてのグループのグループIDをリストで戻します。

- リフレッシュ準備プロシージャ(PREPARE_REFRESH、EXECUTE_REFRESHおよびABORT_REFRESH)は複数のグループで機能します。これらのオーバーロードされたバージョンでは、一度にグループIDのリストを受け入れます。

例8-8 同期リフレッシュ・グループの準備

この例は、MV1、MV2およびMV3の同期リフレッシュ・グループを準備する方法を示しています。

```
EXECUTE DBMS_SYNC_REFRESH.PREPARE_REFRESH(
        DBMS_SYNC_REFRESH.GET_GROUP_ID_LIST('MV1, MV2, MV3'));
```

これら3つのマテリアライズド・ビューがすべて異なるグループである必要はないことに注意してください。マテリアライズド・ビューのうち2つが同じグループでもう1つが別のグループでも、あるいは3つすべてが同じグループでもかまいません。グループIDまたはグループIDのリストを受け入れるため、PREPARE_REFRESHはオーバーロードされるので、前述のコールはどのような場合でも機能します。

例8-9 同期リフレッシュ・グループの実行

この例は、システム内のすべての同期リフレッシュ・グループのリフレッシュを準備および実行する方法を示しています。

```
EXECUTE DBMS_SYNC_REFRESH.PREPARE_REFRESH(
        DBMS_SYNC_REFRESH.GET_ALL_GROUP_IDS);

EXECUTE DBMS_SYNC_REFRESH.EXECUTE_REFRESH(
        DBMS_SYNC_REFRESH.GET_ALL_GROUP_IDS);
```

8.4 同期リフレッシュの変更データの指定および準備

同期リフレッシュでは、PREPARE_REFRESHプロシージャおよびEXECUTE_REFRESHプロシージャへの入力として機能する変更デー

タを指定および準備する必要があります。変更データを指定するには、次の2つの方法があります。

- 同期リフレッシュの変更データの取得時におけるパーティション操作の使用の説明に従って、外部表に変更データを指定し、これをREGISTER_PARTITION_OPERATIONプロシージャに登録します。
- 同期リフレッシュの変更データの取得時におけるステージング・ログの使用の説明に従って、ステージング・ログに変更データを指定し、これらをPREPARE_STAGING_LOGプロシージャで処理します。

変更データに関する注意点を次に示します。

- 2つの方法は互いに排他的ではなく、同じ表であっても同時に使用することが可能です。ただし、指定した変更で競合が存在することはできません。たとえば、変更の数が少ないパーティションではステージング・ログを使用して変更を指定できますが、別のパーティションに大規模な変更がある場合は、そのパーティションの変更を外部表に指定することができます。
- ディメンション表の場合、変更の指定にはステージング・ログのみを使用できます。
- 同期リフレッシュでは、ファクト表およびディメンション表内の任意の組合せの変更を処理できますが、大量の変更がファクト表の少数のパーティションのみに行われるという、データ・ウェアハウスの最も一般的な使用シナリオ向けに最適化されます。
- 同期リフレッシュでは、一般的にデータ・ウェアハウスで使用される、パーティションの追加などの非破壊的パーティション・メンテナンス操作(PMOPS)の使用について、制限を設けていません。このようなPMOPSの使用は、変更データの指定に使用される方法に直接関係しません。
- 同期リフレッシュでは、グループ内のすべてのステージング・ログが準備されている必要があります。これはステージング・ログに登録されている変更がない場合でも同様です。

8.4.1 同期リフレッシュの変更データを取得する際のパーティション操作の実行

REGISTER_PARTITION_OPERATIONプロシージャを使用して、直接変更データを提供できます。この方法はファクト表にのみ適用可能です。変更されるファクト表パーティションごとに、そのパーティションのデータを含む外部表を提供する必要があります。同期リフレッシュのデモ(syncref_run.sqlおよびsyncref_run.log)に例が含まれています。ステップは次のとおりです。

1. 置換するパーティションの外部表を作成します。これは、ファクト表と同じ制約を持つ必要があり、任意の表領域に作成できます。

```
CREATE TABLE      fact_ot_fp3(
   time_key       DATE NOT NULL REFERENCES time(time_key),
   store_key      INTEGER NOT NULL REFERENCES store(store_key),
   dollar_sales   NUMBER (6,2),
   unit_sales     INTEGER)
   tablespace     syncref_fp3_tbs;
```

2. このパーティションのデータを外部表に挿入します。
3. パーティション交換用にこの表を登録します。

```
begin
   DBMS_SYNC_REFRESH.REGISTER_PARTITION_OPERATION(
       partition_op                 => 'EXCHANGE',
       schema_name                  => 'SYNCREF_USER',
       base_table_name              => 'FACT',
       partition_name               => 'FP3',
       outside_partn_table_schema   => 'SYNCREF_USER',
       outside_partn_table_name     => 'FACT_OT_FP3');
   end;
  /
```

```
/
```

外部表を登録し、リフレッシュを実行すると、Oracle DatabaseはEXECUTE_REFRESH時に次の操作を実行します。

```
ALTER TABLE FACT EXCHANGE PARTITION fp3 WITH TABLE fact_ot_fp3
INCLUDING INDEXES WITHOUT VALIDATION;
```

ただし、この文を自分自身で直接発行することはできません。発行すると、Oracle Databaseでは次のエラーが表示されます。

```
ORA-31908: Cannot modify the contents of a table with a staging log.
```

EXCHANGE操作の他に、REGISTER_PARTITION_OPERATIONプロシージャで登録できる2つのパーティション操作として、DROPとTRUNCATEがあります。

例8-10 DROP操作の登録

この例は、次の文を使用して最初のパーティション(FP1)の削除を指定する方法を示しています。

```
begin
 DBMS_SYNC_REFRESH.REGISTER_PARTITION_OPERATION(
              partition_op                => 'DROP',
              schema_name                 => 'SYNCREF_USER',
              base_table_name             => 'FACT',
              partition_name              => 'FP1');
end;
/
```

かわりにパーティションを切り捨てる場合は、partition_opパラメータにDROPではなくTRUNCATEを指定できます。

3つのパーティション操作(EXCHANGE、DROPおよびTRUNCATE)は、表の内容を変更するため破壊的PMOPSと呼ばれます。次のパーティション操作は破壊的ではなく、同期リフレッシュに登録される表で直接実行できます。

- ADD PARTITION
- SPLIT PARTITION
- MERGE PARTITIONS
- MOVE PARTITION
- RENAME PARTITION

データ・ウェアハウスでは、これらのパーティション操作は一般に大量のデータを管理するために使用され、同期リフレッシュはその使用方法に制約を設けません。Oracle Databaseでの要件は、PREPARE_REFRESHコマンドが発行される前にこれらの操作を実行することのみです。これは、PREPARE_REFRESHプロシージャは、ファクト表パーティションとマテリアライズド・ビュー・パーティション間のマッピングを計算しており、PREPARE_REFRESHプロシージャとEXECUTE_REFRESHプロシージャの間にパーティション・メンテナンスが行われると、Oracle DatabaseはEXECUTE_REFRESHでこれを検出し、エラーを表示するためです。

USER_SR_PARTN_OPSカタログ・ビューを使用して、登録されたパーティション操作を表示できます。

```
SELECT   TABLE_NAME, PARTITION_OP, PARTITION_NAME,
         OUTSIDE_TABLE_SCHEMA ot_schema, OUTSIDE_TABLE_NAME ot_name
FROM     USER_SR_PARTN_OPS
ORDER BY TABLE_NAME;

TABLE_NAME PARTITION_ PARTITION_NAME  OT_SCHEMA       OT_NAME
---------- ---------- --------------- --------------- --------------------
FACT       EXCHANGE   FP3             SYNCREF_USER    FACT_OT_FP3

1 row selected.
```

これらのパーティション操作は、同期リフレッシュ操作で使用され、EXECUTE_REFRESHプロシージャにより自動的に登録解除されます。したがって、EXECUTE_REFRESHの後にUSER_SR_PARTN_OPSを問い合せても行は表示されません。

パーティションの登録後に誤りを見つけたり、方針が変わった場合には、UNREGISTER_PARTITION_OPERATIONコマンドを使用して元に戻すことができます。

```
begin
 DBMS_SYNC_REFRESH.UNREGISTER_PARTITION_OPERATION(
               partition_op                    => 'EXCHANGE',
               schema_name                     => 'SYNCREF_USER',
               base_table_name                 => 'FACT',
               partition_name                  => 'FP3');
end;
/
```

8.4.2 同期リフレッシュの変更データを取得する際のステージング・ログの操作

同期リフレッシュにおいて、ステージング・ログは、増分リフレッシュでのマテリアライズド・ビュー・ログに類似した役割を果たします。これらはDDL文で作成され、マテリアライズド・ビュー・ログに変更できます。ただし、マテリアライズド・ビュー・ログとは異なり、変更は指定した形式でユーザーがステージング・ログにロードする必要があります。ステージング・ログの各行は、一意に識別するためのキーを持つ必要があります。これは**ステージング・ログ・キー**と呼ばれ、ステージング・ログ・キーについてで定義されています。

ユーザーはステージング・ログにデータを移入する必要があります。ステージング・ログは、実表内のすべての列とCHAR(2)型の追加制御列DMLTYPE$$で構成されます。これには、行が挿入されていることを示す'I'、削除を示す'D'、および更新される行の新しい値と古い値をそれぞれ示す'UN'と'UO'の値が必要です。最後の2つはペアで表す必要があります。

ステージング・ログはPREPARE_STAGING_LOGプロシージャにより検証され、同期リフレッシュ操作(PREPARE_REFRESHおよびEXECUTE_REFRESH)により使用されます。PREPARE_STAGING_LOGによる検証でエラーが検出された場合、これらは例外表で取得されます。ビューUSER_SR_STLOG_EXCEPTIONSを問い合せることで、例外の詳細を取得できます。

同期リフレッシュでは、同期リフレッシュ・グループのPREPARE_REFRESHをコールする前に、グループ内のすべての表のステージング・ログをPREPARE_STAGING_LOGで処理する必要があります。これは、表に変更データがなく、ステージング・ログが空の場合でも必要です。

この項では、次の項目について説明します。

- ステージング・ログ・キーについて
- ステージング・ログ・ルールについて
- NULLに更新される列について
- ステージング・ログの使用例
- ステージング・ログの準備におけるエラー処理

8.4.2.1 ステージング・ログ・キーについて

実表でステージング・ログを作成するには、実表にキーが必要です。表に主キーがある場合、その主キーは表のステージング・ログでステージング・ログ・キーと見なされます。すべてのディメンション表に主キーがあることに注意してください。

ファクト表の場合、主キーがあるのはあまり一般的ではありません。表に主キーがない場合、そのディメンション表の外部キーである列がそのステージング・ログ・キーを構成します。

ステージング・ログ・キーは次のように説明できます。

- 実表の主キー。ファクト表に主キーがある場合は、サロゲート・キーと呼ばれることもあります。

- ファクト表の外部キーのセット。これはファクト表に主キーがない場合に当てはまります。このような場合というのはデータ・ウェアハウスでは一般的ですが、必ずしもそうとはかぎりません。

ステージング・ログのロード・ルールについては、ステージング・ログ・ルールについてを参照してください。

PREPARE_STAGING_LOGプロシージャは、各キー値が一度だけ指定されていることを確認します。ステージング・ログにデータを移入する際、同じキー値の行が複数回変更される場合には、ユーザーがその変更を統合する必要があります。このプロセスは**変更**の**統合**と呼ばれます。変更の統合を行う際には、次のようにする必要があります。

- 同じ行の削除/挿入は、行'UO'および'UN'による更新操作に統合する。
- 複数の更新を1つの更新に統合する。
- 同じ行での挿入/更新/削除など、NULL変更がステージング・ログに表示されないようにする。
- 複数の更新が後に続く挿入を、1つの挿入に統合する。

8.4.2.2 ステージング・ログ・ルールについて

各行には、主キーを含むすべての列に対して非NULL値が含まれる必要があります。ステージング・ログ内の各キーを1種類の操作に対してのみ指定できるように、変更はすべて統合する必要があります。

挿入される行(DMLTYPE$$が'I')の場合、ステージング・ログ内のすべての列に、実表内の対応する列での制約に準拠した有効な値を提供する必要があります。挿入される行のキーが実表に存在することはできません。

削除される行(DMLTYPE$$が'D')の場合、キー以外の列値はオプションです。同様に、更新される列の古い値を指定する行(DMLTYPE$$が'UO')の場合、キー以外の列値はオプションですが、重要な例外として、値がNULLに更新される列があります。これについてはこの後で説明します。

更新される列の新しい値を指定する行(DMLTYPE$$が'UN')の場合、キー以外の列値は変更された列の値を除いてオプションです。

8.4.2.3 NULLに更新される列について

列がNULLに更新される場合、その古い値は指定する必要があります。指定しないと、Oracle Databaseは、これを更新で値が変更されないままの列と区別できない場合があります。

たとえば、表T1に3つの列c1、c2およびc3があるとします。(c1, c2, c3) = (1, 5, 10)の行があり、ステージング・ログに次の情報を指定するとします。

DMLTYPE$$	C1	C2	C3
UO	1	NULL	NULL
UN	1	NULL	11

古い値を指定しない場合、新しい行は(1, 5, 11)または(1, NULL, 11)となります。しかし、このように指定した場合、明らかに新しい行は(1, 5, 11)となります。c2にNULLを指定する場合は、UO列に次のように古い値を指定する必要があります。

DMLTYPE$$	C1	C2	C3
UO	1	5	NULL

DMLTYPE$$	C1	C2	C3
UN	1	NULL	11

c2の古い値は5(この列に対して以前更新された正しい値)であるため、その新しい値はNULLで、新しい行は(1, NULL, 11)となります。

8.4.2.4 ステージング・ログの使用例

この項では、ステージング・ログの使用例について説明します。

PREPARE_STAGING_LOGプロシージャには、オプションでPSL_MODEという3番目のパラメータを指定できます。これによって、例8-11に示すように、ステージング・ログで指定される3種類のDML文のいずれかまたはすべてを信頼できるものとして扱えるかどうかを指定でき、PREPARE_STAGING_LOGプロシージャによる検証に依存しないようにできます。

例8-11 信頼できるDML文の指定

```
EXECUTE DBMS_SYNC_REFRESH.PREPARE_STAGING_LOG('syncref_user', 'store',
                                    DBMS_SYNC_REFRESH.INSERT_TRUSTED +
                                    DBMS_SYNC_REFRESH.DELETE_TRUSTED);
```

このコールでは、STOREのステージング・ログのINSERTおよびDELETEのDML文の検証はスキップしますが、UPDATE DML文の検証は行います。

例8-12 ステージング・ログの準備

この例は、デモsyncref_run.sqlから取り出したものです。これは、ユーザーが削除および更新操作用にすべての列の値を指定していることを示しています。これらの値が使用可能な場合は、これをお薦めします。

```
INSERT INTO st_store (dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE)
VALUES ('I',    5, 5, 'Store 5', '03060');

INSERT INTO st_store (dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE)
VALUES ('I',    6, 6, 'Store 6', '03062');

INSERT INTO st_store (dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE)
VALUES ('UO',   4, 4, 'Store 4', '03062');

INSERT INTO st_store (dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE)
VALUES ('UN',   4, 4, 'Stor4NewNam', '03062');

INSERT INTO st_store (dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE)
VALUES ('D',    3, 3, 'Store 3', '03060');

EXECUTE DBMS_SYNC_REFRESH.PREPARE_STAGING_LOG('syncref_user', 'store');

-- display initial contents of st_store

SELECT dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE
FROM st_store
ORDER BY STORE_KEY ASC, dmltype$$ DESC;

DM  STORE_KEY  STORE_NUMBER  STORE_NAME    ZIPCODE
--  ---------  ------------  ----------    -------
D           3             3  Store 3         03060
UO          4             4  Store 4         03062
UN          4             4  Stor4NewNam     03062
I           5             5  Store 5         03060
```

```
I            5            5  Store 6         03062

5 rows selected.
```

例8-13 削除および更新レコードの不足している値の充填

この例は、ユーザーが削除および更新操作用にすべての値を指定しない場合、PREPARE_STAGING_LOGプロシージャの実行時にOracle Databaseが不足している値を埋めることを示しています。

```
INSERT INTO st_store (dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE)
VALUES ('D',  3, NULL, NULL, NULL);

INSERT INTO st_store (dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE)
VALUES ('UO',  4, NULL, NULL, NULL);

INSERT INTO st_store (dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE)
VALUES ('UN',  4, NULL, NULL, '03063');

EXECUTE DBMS_SYNC_REFRESH.PREPARE_STAGING_LOG('syncref_user', 'store');

SELECT dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE
FROM ST_STORE ORDER BY STORE_KEY ASC, dmltype$$ DESC;

DM STORE_KEY STORE_NUMBER STORE_NAME  ZIPCODE
-- --------- ------------ ----------- ---------
D          3            3 Store 3     03060
UO         4            4 Store 4     03062
UN         4            4 Store 4     03063
```

例8-14 列のNULLへの更新

この例は、列をNULLに更新する方法を示しています。列値をNULLに更新する場合、UOレコードに古い値を指定する必要があります。

この例の目的は、店舗の郵便番号を4から03063に変更し、その名前をNULLにすることです。古い郵便番号の値を指定することはできますが、'UO'行にはstore_nameの古い値を指定する必要があります。指定しない場合、store_nameは変更されません。

```
INSERT INTO st_store (dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE)
VALUES ('UO',  4, NULL, 'Store 4', NULL);

INSERT INTO st_store (dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE)
VALUES ('UN',  4, NULL, NULL, '03063');

EXECUTE DBMS_SYNC_REFRESH.PREPARE_STAGING_LOG('syncref_user', 'store');

SELECT dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE
FROM st_store ORDER BY STORE_KEY ASC, dmltype$$ DESC;

DM STORE_KEY STORE_NUMBER STORE_NAME  ZIPCODE
-- --------- ------------ ----------- --------
UO         4            4 Store 4     03062
UN         4            4             03063
```

例8-15 ステージング・ログ統計の表示

この例は、USER_SR_STLOG_STATSカタログ・ビューを使用してステージング・ログ統計を表示する方法を示しています。

```
SELECT   TABLE_NAME, STAGING_LOG_NAME, NUM_INSERTS, NUM_DELETE, NUM_UPDATES
FROM     USER_SR_STLOG_STATS
```

```
ORDER BY TABLE_NAME;

TABLE_NAME STAGING_LOG_NAME NUM_INSERTS NUM_DELETES NUM_UPDATES
---------- ---------------- ----------- ----------- -----------
FACT       ST_FACT                    4           1           1
STORE      ST_STORE                   2           1           1
TIME       ST_TIME                    1           0           0

3 rows selected.
```

EXECUTE_REFRESHプロシージャの最後に同じ問合せを使用すると、取得する行はありません。これは、同期リフレッシュによりすべての変更データが使用されたことを示します。

8.4.2.5 ステージング・ログの準備におけるエラー処理

表がPREPARE_STAGING_LOGプロシージャで処理される際、これはその表のみに関係する変更データの仕様でエラーを検出および報告します。たとえば、挿入される行のキーがまだ実表に存在していないことや、削除または更新する行のキーが存在することを確認します。ただし、PREPARE_STAGING_LOGプロシージャは、表での参照整合性制約に関連するエラーを検出できません。つまり、複数の表が関与する変更データの仕様に不整合があった場合、このプロシージャでエラーを検出することはできません。このようなエラーは、EXECUTE_REFRESHプロシージャで検出されます。

8.5 同期リフレッシュ操作のトラブルシューティング

この項では、2つの同期リフレッシュ・プロシージャ、PREPARE_REFRESHおよびEXECUTE_REFRESHのステータスの監視方法および、発生する可能性のあるエラーのトラブルシューティング方法について説明します。同期リフレッシュを使いこなすには、発生する可能性のある様々なエラーのタイプとその対処方法を知っておく必要があります。

エラーの原因として最も可能性の高いものの1つは、変更データの準備での誤りです。これらのエラーは、EXECUTE_REFRESHプロシージャが実行されると、参照制約違反として表示されます。そのような場合、グループのステータスはABORTに設定されます。これらのエラーを認識して対処する方法を理解することが重要です。

この項では、次のトピックについて説明します。

- リフレッシュ操作のステータスの概要
- PREPARE_REFRESHによるSTATUSフィールドの設定方法
- PREPARE_REFRESHを使用した同期リフレッシュの準備の例
- 同期リフレッシュでのEXECUTE_REFRESHによるSTATUSフィールドの設定方法
- EXECUTE_REFRESHを使用した同期リフレッシュの実行の例
- 制約違反のあるEXECUTE_REFRESHの例

8.5.1 リフレッシュ操作のステータスの概要

DBMS_SYNC_REFRESHパッケージは、リフレッシュ実行プロセスを制御する3つのプロシージャを提供しています。同期リフレッシュはPREPARE_REFRESHプロシージャで開始します。これは、リフレッシュ操作全体を計画し、リフレッシュのための計算作業の大部分を行います。その後、リフレッシュを実行するEXECUTE_REFRESHプロシージャが続きます。提供されている3番目のプロシージャはABORT_REFRESHで、これは、いずれかのプロシージャが失敗した場合にエラーからのリカバリに使用されます。

USER_SR_GRP_STATUSカタログ・ビューおよびUSER_SR_OBJ_STATUSカタログ・ビューには、現在のグループのリフレッシュ操作のステータスに関するすべての情報が含まれます。

- USER_SR_GRP_STATUSビューは、グループ全体としてのステータスを示します。

- OPERATIONフィールド(PREPAREまたはEXECUTE)は、そのグループでの現在のリフレッシュ・プロシージャの実行状況を示します。
- STATUSフィールド(RUNNING、COMPLETE、ERROR-SOFT、ERROR-HARD、ABORT、PARTIAL)は、操作のステータスを示します。これらについては、後で詳しく説明します。
- グループは、そのグループIDで識別されます。

- USER_SR_OBJ_STATUSビューは、各個別オブジェクトのステータスを示します。
  - オブジェクトは、その所有者、名前、タイプ(TABLEまたはMVIEW)およびグループIDによって識別されます。
  - STATUSフィールド(NOT PROCESSED、ABORTまたはCOMPLETE)。これらについては、後で詳しく説明します。

8.5.2 PREPARE_REFRESHによるSTATUSフィールドの設定方法

新しいPREPARE_REFRESHジョブを起動すると、グループのSTATUSはRUNNINGに設定され、グループ内のオブジェクトのSTATUSはNOT PROCESSEDに設定されます。PREPARE_REFRESHジョブが終了してもオブジェクトのステータスには変化がありませんが、グループのステータスは次の3つの値のいずれかに変わります。

- COMPLETE: ジョブが正常に完了した場合。
- ERROR_SOFT: ジョブでORA-01536「表領域%sに対する領域割当て制限を使い果たしました。」エラーが発生した場合。
- ERROR_HARD: その他(つまり、ジョブでORA-01536以外のなんらかのエラーが発生した場合)。

PREPARE_REFRESHプロシージャの使用時の注意点

- グループ内のオブジェクトのNOT PROCESSEDステータスは、PREPARE_REFRESHジョブによってそのオブジェクトのデータが変更されていないことを示しています。データの変更はEXECUTE_REFRESHステップでのみ発生します。この際、ステータスは適宜変更されます。これについては後で説明します。
- STATUSがERROR_SOFTの場合、表領域の領域割当を増やすことでORA-01536エラーを修正し、PREPARE_REFRESHを再開できます。かわりに、ABORT_REFRESHを使用してリフレッシュを中断させることもできます。
- STATUS値がERROR_HARDの場合は、ABORT_REFRESHによるリフレッシュの中断が唯一の選択肢となります。
- PREPARE_REFRESHプロシージャの終了後のSTATUS値がRUNNINGである場合は、エラーが発生しています。Oracleサポート・サービスに連絡してください。
- PREPARE_REFRESHプロシージャではリソースを大量に使用する場合があるため、ERROR_HARDのSTATUS値は、リソースの枯渇と関連している可能性があります。問題を特定できない場合は、Oracleサポート・サービスに連絡してください。ただし、問題を特定してそれを修正できる場合には、まずABORT_REFRESHを実行し、次にPREPARE_REFRESHプロシージャを実行することによって、同期リフレッシュの使用を続行できる可能性があります。
- 新しいPREPARE_REFRESHジョブを起動できるのは、そのグループの前のリフレッシュ操作(存在する場合)が正常終了しているか、中断された場合のみであることに留意してください。
- PREPARE_REFRESHプロシージャの最終的なSTATUS値がCOMPLETEではない場合、EXECUTE_REFRESHステップには進めません。PREPARE_REFRESHが正常に機能しない場合は、登録解除フェーズに進み、グループ内のオブジェクトを他のリフレッシュ方法で管理できます。

8.5.3 PREPARE_REFRESHを使用した同期リフレッシュの準備の例

この項では、リフレッシュの準備の際の一般的な例を示します。

例8-16 ステータスがCOMPLETEで正常終了するPREPARE_REFRESH

この例は、正常に完了するPREPARE_REFRESHプロシージャを示しています。

```
EXECUTE DBMS_SYNC_REFRESH.PREPARE_REFRESH( DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));

PL/SQL procedure successfully completed.

SELECT OPERATION, STATUS
FROM USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1');

OPERATION  STATUS
---------  ------
PREPARE    COMPLETE
```

例8-17 ステータスがERROR_SOFTで失敗するPREPARE_REFRESH

この例は、ORA-01536が発生するPREPARE_REFRESHプロシージャを示しています。

```
EXECUTE DBMS_SYNC_REFRESH.PREPARE_REFRESH( DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));
BEGIN DBMS_SYNC_REFRESH.PREPARE_REFRESH(DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1')); END;

*
ERROR at line 1:
ORA-01536: space quota exceeded for tablespace 'DUMMY_TS'
ORA-06512: at "SYS.DBMS_SYNC_REFRESH", line 63
ORA-06512: at "SYS.DBMS_SYNC_REFRESH", line 411
ORA-06512: at "SYS.DBMS_SYNC_REFRESH", line 429
ORA-06512: at line 1PL/SQL procedure successfully completed.

SELECT OPERATION, STATUS
FROM USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1');

OPERATION  STATUS
---------  ------
PREPARE    ERROR_SOFT
```

例8-18 PREPARE_REFRESHの再開の成功

この例は、例8-17の続きです。ORA-01536エラーが発生したら、DUMMY_TSの表領域を増やし、PREPARE_REFRESHプロシージャを再実行します。今回は正常に終了します。PREPARE_REFRESHプロシージャは停止した場所から処理を再開することに注意してください。PREPARE_REFRESHプロシージャの使用方法は通常と変わりなく、プロシージャが再開されていることを示すパラメータや設定は必要ありません。

```
EXECUTE DBMS_SYNC_REFRESH.PREPARE_REFRESH(DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));

PL/SQL procedure successfully completed.

SELECT OPERATION, STATUS
FROM USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1');

OPERATION  STATUS
---------  ------
PREPARE    COMPLETE
```

例8-19 PREPARE_REFRESHの中断

この例は、PREPARE_REFRESHプロシージャが失敗し、STATUS値がERROR_HARDであることを前提としています。ABORT_REFRESHプロシージャを実行して準備ジョブを中断します。STATUS値が最終的にERROR_HARDからABORTに変わることに注意してください。

```
SELECT OPERATION, STATUS
FROM USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1');

OPERATION  STATUS
---------  ------
PREPARE    ERROR_HARD

EXECUTE DBMS_SYNC_REFRESH.ABORT_REFRESH( DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));

PL/SQL procedure successfully completed.

SELECT OPERATION, STATUS
FROM USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1');

OPERATION  STATUS
---------  ------
PREPARE    ABORT
```

8.5.4 同期リフレッシュでのEXECUTE_REFRESHによるSTATUSフィールドの設定方法

EXECUTE_REFRESHプロシージャでは、同期リフレッシュ・グループ内のオブジェクトのグループをサブグループに分割し、各サブグループがアトミックにリフレッシュされます。最初のサブグループは実表で構成されます。同期リフレッシュ・グループ内の各マテリアライズド・ビューは別個のサブグループに配置され、アトミックにリフレッシュされます。

EXECUTE_REFRESHプロシージャの場合、STATUSフィールドの考えられる最終状態は、COMPLETE、PARTIALおよびABORTです。

- STATUS = COMPLETE

  実表とすべてのマテリアライズド・ビューが正常にリフレッシュされると、この状態になります。

- STATUS = ABORT

  この状態は、実表のサブグループのリフレッシュが失敗したことを示しています。表とマテリアライズド・ビュー内のデータは整合していますが、変更されていません。この状態が生じた場合は、失敗に関連するエラーがあると考えられます。これが制約違反などのユーザー・エラーである場合は、問題を修正して同期リフレッシュ操作を最初(つまり、グループPREPARE_REFRESHおよびEXECUTE_REFRESH内の各表のPREPARE_STAGING_LOG)から行うことができます。これがユーザー・エラーではない場合は、Oracleサポート・サービスに連絡してください。

- STATUS = PARTIAL

  実表はすべて正常にリフレッシュされ、マテリアライズド・ビューはすべてではなく一部のみが正常にリフレッシュされると、この状態になります。正常にリフレッシュされた表とマテリアライズド・ビュー内のデータは互いに整合しています。その他のマテリアライズド・ビューは失効しており、リフレッシュを完了させる必要があります。この状態が生じた場合は、失敗に関連するエラーがあると考えられます。最も可能性が高いのは、ユーザー・エラーではなく、Oracleサポート・サービスに報告する必要のあるOracleエラーです。この状態では、次の2つの選択肢があります。

  - EXECUTE_REFRESHプロシージャの実行を再度試みます。この場合、EXECUTE_REFRESHは、失敗したマテリアライズド・ビューのリフレッシュを、PCTリフレッシュやCOMPLETEリフレッシュなどの別のリフレッシュ方法で再試行します。すべてのマテリアライズド・ビューが成功すると、ステータスはCOMPLETEに設定されます。そうでない場合、ステータスはPARTIALのままとなります。

- ABORT_REFRESHプロシージャを起動して、マテリアライズド・ビューを中断します。これにより、すべてのマテリアライズド・ビューおよび実表への変更がロールバックされます。これらはすべて、ステージング・ログ内の変更または登録されたパーティション操作が適用される前の元の状態と同じデータを持つことになります。

EXECUTE_REFRESHプロシージャでのエラーの場合、USER_SR_GRP_STATUSビュー内の次のフィールドも役立ちます。

- NUM_MVS_COMPLETED: 正常にリフレッシュ操作を完了したマテリアライズド・ビューの数が含まれています。
- NUM_MVS_ABORTED: 中断されたマテリアライズド・ビューの数が含まれています。
- ERRORおよびERROR_MESSAGE: 操作で発生したエラーが記録されます。

EXECUTE_REFRESHプロシージャの最後には、グループ内のオブジェクトのステータスがUSER_SR_OBJ_STATUSビューで次のようにマークされます。

- 変更が正常に適用された場合、オブジェクトのステータスはCOMPLETEに設定されます。
- 変更が正常に適用されなかった場合、オブジェクトのステータスはABORTに設定されます。この場合、オブジェクトはリフレッシュ操作の前と同じ状態です。ERRORおよびERROR_MESSAGEフィールドには、操作で発生したエラーが記録されます。
- 変更が適用されなかった場合、オブジェクトのステータスはNOT PROCESSEDのままとなります。

8.5.5 EXECUTE_REFRESHを使用した同期リフレッシュの実行例

この項では、リフレッシュの実行の際の一般的な例を示します。

例8-20 正常に完了するEXECUTE_REFRESH

例8-20は、正常に完了するEXECUTE_REFRESHプロシージャを示しています。

```
EXECUTE DBMS_SYNC_REFRESH.EXECUTE_REFRESH( DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));

PL/SQL procedure successfully completed.

SELECT OPERATION, STATUS
FROM USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1');

OPERATION  STATUS
---------  ------
EXECUTE    COMPLETE
```

例8-21 部分的に成功するEXECUTE_REFRESH

例8-21は、部分的に成功するEXECUTE_REFRESHプロシージャを示しています。この例で、EXECUTE_REFRESHプロシージャは、実表のリフレッシュ後、すべてのマテリアライズド・ビューのリフレッシュが完了する前に失敗します。結果として生じるグループのステータスはPARTIALで、QSM-03280エラー・メッセージがスローされます。

```
EXECUTE DBMS_SYNC_REFRESH.EXECUTE_REFRESH(DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));
BEGIN DBMS_SYNC_REFRESH.EXECUTE_REFRESH(DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1')); END;

*
ERROR at line 1:
ORA-31928: Synchronous refresh error
QSM-03280: One or more materialized views failed to refresh successfully.
ORA-06512: at "SYS.DBMS_SYNC_REFRESH", line 63
ORA-06512: at "SYS.DBMS_SYNC_REFRESH", line 411
ORA-06512: at "SYS.DBMS_SYNC_REFRESH", line 446
ORA-06512: at line 1
```

EXECUTE_REFRESHプロシージャの後で、グループ自体のステータスをチェックします。操作フィールドがEXECUTEに設定され、ステータスがPARTIALであることに注意してください。

```
SELECT OPERATION, STATUS FROM USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1');

OPERATION  STATUS
---------  -------------
EXECUTE    PARTIAL
```

USER_SR_GRP_STATUSビューを問い合せることによって、中断したマテリアライズド・ビューの数が1で、失敗したマテリアライズド・ビューがMV1であることがわかります。

グループ内のオブジェクトのステータスを調べると、STOREおよびTIMEは変更されていないため、それらのステータスはNOT PROCESSEDです。

```
SELECT NAME, TYPE, STATUS FROM USER_SR_OBJ_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1')
ORDER BY TYPE, NAME;

NAME             TYPE       STATUS
---------------- ---------- ----------------
MV1              MVIEW      ABORT
MV1_HALFMONTH    MVIEW      COMPLETE
MV2              MVIEW      COMPLETE
MV2_YEAR         MVIEW      COMPLETE
FACT             TABLE      COMPLETE
STORE            TABLE      NOT PROCESSED
TIME             TABLE      NOT PROCESSED

7 rows selected.

SELECT NUM_TBLS, NUM_MVS, NUM_MVS_COMPLETED, NUM_MVS_ABORTED
FROM USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1');

NUM_TBLS NUM_MVS NUM_MVS_COMPLETED NUM_MVS_ABORTED
-------- ------- ----------------- ---------------
       3       4                 3               1
```

この時点で、もう一度EXECUTE_REFRESHプロシージャの実行を試みることができます。再試行に成功し、失敗したマテリアライズド・ビューが成功すると、グループのステータスはCOMPLETEに設定されます。そうでない場合、ステータスはPARTIALのままとなります。例8-22を参照してください。リフレッシュ・プロシージャを中断して元の状態に戻すこともできます。例8-23を参照してください。

例8-22 PARTIALステータス後のリフレッシュの再試行

例8-22は例8-21の続きを示しています。EXECUTE_REFRESHプロシージャを再試行し、成功します。

```
EXECUTE DBMS_SYNC_REFRESH.EXECUTE_REFRESH(DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));

PL/SQL procedure successfully completed.

--Check the status of the group itself after the EXECUTE_REFRESH operation;
--note that the operation field is set to EXECUTE and status is COMPLETE.

SELECT OPERATION, STATUS
FROM USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1');
```

```
OPERATION  STATUS
---------  ---------
EXECUTE    COMPLETE
```

USER_SR_GRP_STATUSビューを問い合せることによって、中断したマテリアライズド・ビューの数が0で、MV1のステータスがCOMPLETEであることがわかります。グループ内のオブジェクトのステータスを調べると、STOREおよびTIMEは変更されていないため、それらのステータスはNOT PROCESSEDです。

```
SELECT NAME, TYPE, STATUS FROM USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1')
ORDER BY TYPE, NAME;

NAME             TYPE       STATUS
---------------- ---------- ----------------
MV1              MVIEW      COMPLETE
MV1_HALFMONTH    MVIEW      COMPLETE
MV2              MVIEW      COMPLETE
MV2_YEAR         MVIEW      COMPLETE
FACT             TABLE      COMPLETE
STORE            TABLE      NOT PROCESSED
TIME             TABLE      NOT PROCESSED

7 rows selected.

SELECT NUM_TBLS, NUM_MVS, NUM_MVS_COMPLETED, NUM_MVS_ABORTED
FROM  USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1');

NUM_TBLS NUM_MVS NUM_MVS_COMPLETED NUM_MVS_ABORTED
-------- ------- ----------------- ---------------
       3       4                 4               0
```

表とマテリアライズド・ビューを調べ、変更データ内の変更がそれらに正しく適用されていること、およびマテリアライズド・ビューと表が互いに整合していることを確認できます。

例8-23 PARTIALステータスのリフレッシュの中断

例8-23は、PARTIAL状態のリフレッシュ・プロシージャの中断を示しています。

```
EXECUTE DBMS_SYNC_REFRESH.ABORT_REFRESH(DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));

PL/SQL procedure successfully completed.
```

ABORT_REFRESHプロシージャの後で、グループ自体のステータスをチェックします。操作フィールドがEXECUTEに設定され、ステータスがABORTであることに注意してください。

```
SELECT OPERATION, STATUS FROM USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1');

OPERATION  STATUS
---------- -------
EXECUTE    ABORT
```

USER_SR_GRP_STATUSビューを問い合せることによって、すべてのマテリアライズド・ビューとファクト表が中断されたことがわかります。グループ内のオブジェクトのステータスを調べます。STOREおよびTIMEは変更されていないため、それらのステータスはNOT PROCESSEDです。

```
SELECT NAME, TYPE, STATUS FROM USER_SR_GRP_STATUS
```

```
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1')
ORDER BY TYPE, NAME;

NAME             TYPE       STATUS
---------------- ---------- ----------------
MV1              MVIEW      ABORT
MV1_HALFMONTH    MVIEW      ABORT
MV2              MVIEW      ABORT
MV2_YEAR         MVIEW      ABORT
FACT             TABLE      ABORT
STORE            TABLE      NOT PROCESSED
TIME             TABLE      NOT PROCESSED

7 rows selected.

SELECT NUM_TBLS, NUM_MVS, NUM_MVS_COMPLETED, NUM_MVS_ABORTED
FROM USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1');

NUM_TBLS NUM_MVS NUM_MVS_COMPLETED NUM_MVS_ABORTED
-------- ------- ----------------- ---------------
       3       4                 0               4
```

表とマテリアライズド・ビューを調べ、これらがすべて元の状態で、変更データの変更が適用されていないことを確認できます。

8.5.6 制約違反のあるEXECUTE_REFRESHの例

同期リフレッシュ方法では、変更データは表とマテリアライズド・ビューに同時にロードされ、同期した状態を保ちます。その他のリフレッシュ方法では、変更データはまず表にロードされ、その際に有効なすべての制約がチェックされます。同期リフレッシュ方法では、外部表はユーザーが信頼するデータを使用して準備され、制約違反は実行時間を省くため無効にされます。次の例は、EXECUTE_REFRESHプロシージャによって検出される制約違反を示しています。このような場合、EXECUTE_REFRESHプロシージャの最終的なステータスはABORTとなります。変更データの問題を特定して修正し、もう一度同期リフレッシュ・フェーズを開始する必要があります。

例8-24 子キーの制約違反

例8-24では、rdbms/demoディレクトリのファイルsyncref_run.sqlと同じ表が使用され、同じデータが移入されるものとします。特に、表STOREには、主キーSTORE_KEYの値が1から4の4つの行があり、FACT表には、store 3を含む4つすべての店舗を参照する行があります。

親キーの制約違反を示すため、STOREのステージング・ログにSTORE_KEYが3の行の削除を移入します。他の表へのその他の変更はありません。EXECUTE_REFRESHプロシージャを実行すると、次のようにORA-02292エラーで失敗します。

```
INSERT INTO st_store (dmltype$$, STORE_KEY, STORE_NUMBER, STORE_NAME, ZIPCODE)
VALUES ('D',   3, 3, 'Store 3', '03060');

-- Prepare the staging logs
EXECUTE DBMS_SYNC_REFRESH.PREPARE_STAGING_LOG('syncref_user', 'fact');
EXECUTE DBMS_SYNC_REFRESH.PREPARE_STAGING_LOG('syncref_user', 'time');
EXECUTE DBMS_SYNC_REFRESH.PREPARE_STAGING_LOG('syncref_user', 'store');

-- Prepare the refresh
EXECUTE DBMS_SYNC_REFRESH.PREPARE_REFRESH(DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));

-- Execute the refresh
EXECUTE DBMS_SYNC_REFRESH.EXECUTE_REFRESH( -
           DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));
BEGIN DBMS_SYNC_REFRESH.EXECUTE_REFRESH(DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1')); END;
```

```
*
ERROR at line 1:
ORA-02292: integrity constraint (SYNCREF_USER.SYS_C0031765) violated - child record found
ORA-06512: at line 1
ORA-06512: at "SYS.DBMS_SYNC_REFRESH", line 63
ORA-06512: at "SYS.DBMS_SYNC_REFRESH", line 411
ORA-06512: at "SYS.DBMS_SYNC_REFRESH", line 446
ORA-06512: at line 1
```

EXECUTE_REFRESHプロシージャの後で、グループ自体のステータスを確認します。操作フィールドがEXECUTEに設定され、ステータスがABORTであることに注意してください。

```
SELECT OPERATION, STATUS
FROM USER_SR_GRP_STATUS
WHERE GROUP_ID = DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1');

OPERATION  STATUS
---------- --------------
EXECUTE    ABORT
```

実表とMV1の内容をチェックすると、変更がなく、これらすべてに元の値が含まれていることがわかります。

8.6 同期リフレッシュ適格性分析の実行

CAN_SYNCREF_TABLE関数は、表とその依存マテリアライズド・ビューが同期リフレッシュで使用可能かどうかを示します。その分析についての説明も行います。表とビューが使用できない場合、その原因を調査し、可能であれば適切な処置を行うことができます。同期リフレッシュで使用できるようにするには、表が前述の様々な条件を満たす必要があります。

CAN_SYNCREF_TABLE関数は、次の2つの方法で起動できます。

- CAN_SYNCREF_TABLE関数の出力を格納する表を使用します

  出力表を使用するための基本構文は、次のとおりです。

  ```
  can_syncref_table(schema_name   IN VARCHAR2,
                    table_name    IN VARCHAR2,
                    statement_id  IN VARCHAR2)
  ```

- CAN_SYNCREF_TABLE関数の出力を格納するVARRAYを使用します

  CAN_SYNCREF_TABLE関数の出力を表ではなくVARRAYに送るには、次のようにプロシージャをコールします。

  ```
  can_syncref_table(schema_name   IN VARCHAR2,
                    table_name    IN VARCHAR2,
                    output_array  IN OUT Sys.CanSyncRefTypeArray)
  ```

スクリプトutlcsrt.sqlを実行すると、出力表SYNCREF_TABLEを作成できます。

表8-1 CAN_SYNCREF_TABLE

パラメータ	説明
schema_name	実表のスキーマの名前。
base_table_name	実表の名前。

パラメータ	説明
statement_id	出力がユーザーのスキーマ内の SYNCREF_TABLE という表に送られる際に CAN_SYNCREF_TABLE 関数のコールに関係する行を識別する文字列(VARCHAR2(30))。
output_array	同期リフレッシュにおける元表およびその依存マテリアライズド・ビューの適格性に関する情報を含む出力配列(CAN_SYNCREF_TABLE によって記録される)。

ノート:

CAN_SYNCREF_TABLE 関数には、1 つの statement_id または output_array パラメータのみを指定する必要があります。

8.6.1 SYNCREF_TABLEを使用した同期リフレッシュ適格性分析の結果の格納

CAN_SYNCREF_TABLE関数の出力は、表SYNCREF_TABLEに送ることができます。ユーザーはSYNCREF_TABLEの作成を担います。これは必要がなくなったら削除できます。SYNCREF_TABLEの形式は次のとおりです。

```
CREATE TABLE SYNCREF_TABLE(
          statement_id              VARCHAR2(30),
          schema_name               VARCHAR2(30),
          table_name                VARCHAR2(30),
          mv_schema_name            VARCHAR2(30),
          mv_name                   VARCHAR2(30),
          eligible                  VARCHAR2(1),    -- 'Y' , 'N'
          seq_num                   NUMBER,
          msg_number                NUMBER,
          message                   VARCHAR2(4000)
);
```

同じ表でこのプロシージャを起動するたびに、異なるstatement_idパラメータを指定する必要があります。そうしないとエラーがスローされます。statement_id、schema_nameおよびtable_nameフィールドは、特定の表およびstatement_idの結果を識別します。

各行には、表またはその依存マテリアライズド・ビューの適格性に関する情報が含まれます。CAN_SYNCREF_TABLE関数は、NULLまたは非NULLの、mv_schema_nameとmv_nameの両方の値が各行にあることを保証します。これらの行には、次のような意味があります。

- mv_schema_name値がNULLでmv_nameがNULLの場合、ELIGIBLEフィールドは表が同期リフレッシュで使用できるかどうかを表します。表が使用できない場合、MSG_NUMBERおよびMESSAGEフィールドにその理由が示されます。
- mv_schema_name値がNOT NULLでmv_nameがNOT NULLの場合、ELIGIBLEフィールドはマテリアライズド・ビューが同期リフレッシュで使用できるかどうかを表します。マテリアライズド・ビューが使用できない場合、MSG_NUMBERおよびMESSAGEフィールドにその理由が示されます。

同じ表でこのプロシージャを起動するたびに、異なるstatement_idパラメータを指定する必要があります。そうしないと、エラーがスローされます。statement_id、schema_nameおよびtable_nameフィールドは、特定の表およびstatement_idの結果を識別します。

8.6.2 VARRAYを使用した同期リフレッシュ適格性分析の結果の格納

CAN_SYNCREF_TABLE関数の出力は、PL/SQLのVARRAYに保存できます。この配列の要素はCanSyncRefMessage型で、SYSスキーマで次の例のように事前定義されています。

```
TYPE CanSyncRefMessage IS OBJECT (
          schema_name         VARCHAR2(30),
          table_name          VARCHAR2(30),
          mv_schema_name      VARCHAR2(30),
          mv_name             VARCHAR2(30),
          eligible            VARCHAR2(1),     -- 'Y', 'N'
          seq_num             NUMBER,
          msg_number          NUMBER,
          message             VARCHAR2(4000)
);
```

配列型CanSyncRefArrayTypeは、CanSyncRefMessageオブジェクトのVARRAYで、SYSスキーマで次のように事前定義されています。

```
TYPE  CanSyncRefArrayType AS VARRAY(256) OF CanSyncRefMessage;
```

各CanSyncRefMessageレコードは、同期リフレッシュに対する実表または依存マテリアライズド・ビューの適格性に関するメッセージを提供します。フィールドの意味は、SYNCREF_TABLE内の対応するフィールドの意味と同じです。ただし、CAN_SYNCREF_TABLEプロシージャがVARRAYパラメータとともにコールされた場合、statement_idは提供されない(要求されないため)ので、SYNCREF_TABLEにはCanSyncRefMessageにないstatement_idフィールドがあります。

CanSyncRefArrayTypeのデフォルトのサイズ制限は、256要素です。256以上の要素を必要とする場合、CanSyncRefArrayをSYSとして接続し、再定義します。次のコマンドは、SYSユーザーとして接続した場合、CanSyncRefArrayを再定義し、制限を2048要素に設定します。

```
CREATE OR REPLACE TYPE CanSyncRefArrayType AS VARRAY(2048) OF SYS.CanSyncRefMessage;
/
GRANT EXECUTE ON SYS.CanSyncRefMessage TO PUBLIC;

CREATE OR REPLACE PUBLIC SYNONYM CanSyncRefMessage FOR SYS.CanSyncRefMessage;
/
GRANT EXECUTE ON SYS.CanSyncRefArrayType TO PUBLIC;

CREATE OR REPLACE PUBLIC SYNONYM CanSyncRefArrayType FOR SYS.CanSyncRefArrayType;
/
```

8.6.3 デモ・スクリプト

rdbms/demoディレクトリの同期リフレッシュ・デモ・スクリプトには、CAN_SYNCREF_APIなど、様々な同期リフレッシュ操作の最も一般的なシナリオの例が含まれています。メイン・スクリプトはsyncref_run.sqlで、そのログはsyncref_run.logです。ファイルsyncref_cst.sqlは、DO_CSTとDO_CST_ARRの2つのプロシージャを定義します。これらはCAN_SYNCREF_TABLE関数の使用を簡素化し、使いやすい形式で情報を画面に表示します。この形式は、syncref_cst.sqlファイルに記載されています。

8.7 同期リフレッシュのセキュリティに関する考慮事項の概要

DBMS_SYNC_REFRESHパッケージの実行権限は、PUBLICに対して付与されており、すべてのユーザーがこのパッケージのプロシージャを実行して、自身が所有するオブジェクトで同期リフレッシュを実行できます。データベース管理者は、データベース内のすべての表およびマテリアライズド・ビューで同期リフレッシュ操作を実行できます。

一般に、DBA権限のないユーザーが別のユーザーの表で同期リフレッシュを使用する場合、その表における完全な読取りおよび

書込み権限が必要です。つまり、その表またはマテリアライズド・ビューのSELECT、INSERT、UPDATEおよびDELETE権限を持つ必要があります。ユーザーは、SELECT権限でなくREAD権限を持つことができます。次のように、例外が2つあります。

- PURGE_REFRESH_STATS関数およびALTER_REFRESH_STATS_RETENTION関数

  これら2つのファンクションは、パージ・ポリシーを実装し、デフォルトのリテンション期間を変更するために使用できます。これらの関数を実行できるのはデータベース管理者のみです。

- CAN_SYNCREF_TABLE関数

  これは、指定された表に関連付けられたすべてのマテリアライズド・ビューの同期リフレッシュの適格性を調査するアドバイザ関数です。したがって、このファンクションでは、指定した表に関連付けられたすべてのマテリアライズド・ビューに対するREADまたはSELECT権限が必要です。

9 マテリアライズド・ビューのリフレッシュ操作の監視

ここでは、リフレッシュ統計を使用して、マテリアライズド・ビューのリフレッシュ操作のパフォーマンスを監視する方法について説明します。

この章のトピックは、次のとおりです:

9.1 マテリアライズド・ビュー・リフレッシュ統計について

Oracle Databaseでは、マテリアライズド・ビューのリフレッシュ操作に関する統計が収集および保存されます。これらの統計には、データ・ディクショナリ・ビューを使用してアクセスできます。

現在と過去両方のマテリアライズド・ビューのリフレッシュ操作に関する統計がデータベースに保存されます。過去のマテリアライズド・ビュー・リフレッシュ統計では、データベースにおけるマテリアライズド・ビューのリフレッシュ・パフォーマンスを時間の経過に沿って把握および分析できます。リフレッシュ統計は、様々なレベルの粒度で収集可能です。

マテリアライズド・ビュー・リフレッシュ統計の管理には、次のような機能が用意されています。

- マテリアライズド・ビューのリフレッシュ操作に関するレポート機能
  - マテリアライズド・ビューのリフレッシュ操作に関する現在と過去両方の統計を表示できます。
  - 実際のリフレッシュ実行時間に関する統計を表示できます。
  - 実際のリフレッシュ実行時間に関する統計を使用して、マテリアライズド・ビューのリフレッシュのパフォーマンスを時間の経過に沿って追跡できます。
- マテリアライズド・ビューのリフレッシュ・パフォーマンスに関する診断機能

  現在および過去の詳細な統計を使用して、マテリアライズド・ビューのリフレッシュ操作のパフォーマンスをすばやく分析できます。たとえば、マテリアライズド・ビューのリフレッシュに時間がかかる場合、リフレッシュ統計を使用して、処理速度の低下の原因がシステム負荷の増大であるのか、多種多様な変更データであるのかを特定できます。

9.2 マテリアライズド・ビュー・リフレッシュ統計の管理の概要

Oracle Databaseでは、定義したデータベース設定に基づいて、マテリアライズド・ビュー・リフレッシュ統計の収集および保存が管理されます。デフォルトでは、データベース全体のマテリアライズド・ビューのリフレッシュ操作に関する基本的な統計が収集および保存されます。

マテリアライズド・ビュー・リフレッシュ統計の管理には、次の設定を制御するポリシーの定義が含まれます。

- マテリアライズド・ビュー・リフレッシュ統計の詳細レベル
- マテリアライズド・ビュー・リフレッシュ統計の保存期間

マテリアライズド・ビュー・リフレッシュ統計を管理するポリシーを定義するには、次の手法を使用します。

- データベース全体に適用されるデフォルト設定の定義

  DBMS_MVIEW_STATS.SET_SYSTEM_DEFAULTプロシージャでは、データベース全体のマテリアライズド・ビュー・リフレッシュ統計の収集および保存を管理するデフォルト設定を定義できます。

- 個々のマテリアライズド・ビューの収集および保存ポリシーの定義

  DBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMSプロシージャでは、個々のマテリアライズド・ビューのレベルで統計の収集および保存を管理することにより、マテリアライズド・ビュー・リフレッシュ統計をよりきめ細かく制御できます。マテリアライズド・ビュー・レベルで指定された設定は、データベース・レベルの設定より優先されます。

ノート:

- マテリアライズド・ビュー・リフレッシュ統計の収集
- マテリアライズド・ビュー・リフレッシュ統計の保存

9.3 マテリアライズド・ビュー・リフレッシュ統計が保存されるデータ・ディクショナリ・ビューについて

Oracle Databaseでは、マテリアライズド・ビュー・リフレッシュ統計がデータ・ディクショナリに保存されます。マテリアライズド・ビューのリフレッシュの収集レベルを設定することにより、収集されるリフレッシュ統計の詳細レベルを制御できます。

マテリアライズド・ビューのリフレッシュ操作はそれぞれ、一意のリフレッシュIDを使用して識別されます。1つのリフレッシュ操作によって、複数のマテリアライズド・ビューがリフレッシュされることもあります。たとえば、REFRESH_DEPENDENTプロシージャを使用して1つのマテリアライズド・ビューをリフレッシュすると、同じリフレッシュ操作の一環として、指定したマテリアライズド・ビューに依存するマテリアライズド・ビューもすべてリフレッシュされます。その結果、この操作の一環としてリフレッシュされたすべてのマテリアライズド・ビューのリフレッシュIDは同じになります。

表9-1 マテリアライズド・ビュー・リフレッシュ統計が保存されるデータ・ディクショナリ・ビュー

ビュー名	説明
DBA_MVREF_STATS	リフレッシュ操作のリフレッシュ ID や基本的なタイミング統計など、マテリアライズド・ビューのリフレッシュに関する基本的な統計が保存されます。 このビューには、リフレッシュ統計が収集されている各マテリアライズド・ビューに関する次の情報が格納されます。 - マテリアライズド・ビューの名前

ビュー名	説明
	● 使用されたリフレッシュ方法 ● リフレッシュ操作の開始時と終了時におけるマテリアライズド・ビュー内の行数 ● マテリアライズド・ビューのリフレッシュに使用されたステップの数 ノート: これは、集計または結合のみを含むマテリアライズド・ビューの高速リフレッシュ用に移入されたビューです。他のタイプのマテリアライズド・ビューのリフレッシュでは移入されません。
DBA_MVREF_RUN_STATS	それぞれのマテリアライズド・ビュー・リフレッシュ操作に関する次のような詳細情報が保存されます。 ● マテリアライズド・ビューのリスト、リフレッシュ方法、消去オプションなど、リフレッシュ操作の実行時に指定したパラメータ ● リフレッシュ操作でリフレッシュされたマテリアライズド・ビューの数 ● 開始時刻、終了時刻、経過時間など、リフレッシュ操作に関する詳細なタイミング統計
DBA_MVREF_CHANGE_STATS	マテリアライズド・ビューのリフレッシュ操作に関連付けられた実表の変更データ・ロード情報が格納されます。 詳細には、実表名、マテリアライズド・ビュー名、挿入された行数、更新された行数、削除された行数、ダイレクト・ロード・インサートの数、PMOP の詳細、リフレッシュ操作の開始時の行数などが含まれます。
DBA_MVREF_STMT_STATS	1 つのマテリアライズド・ビュー・リフレッシュ操作に含まれる各リフレッシュ文に関連する情報が格納されます。 これには、マテリアライズド・ビュー名、リフレッシュ ID、リフレッシュ文、リフレッシュ文の SQLID、文の実行計画などの情報が含まれます。

関連項目:

9.4 マテリアライズド・ビュー・リフレッシュ統計の収集

Oracle Databaseでは、マテリアライズド・ビューのリフレッシュ操作に関する基本的な統計が収集されます。これらの統計はデータ・ディクショナリに保存され、マテリアライズド・ビューのリフレッシュ操作のパフォーマンスを分析するために使用できます。

関連項目:

- マテリアライズド・ビュー・リフレッシュ統計の収集について
- マテリアライズド・ビュー・リフレッシュ統計の収集に関するデフォルト設定の指定
- マテリアライズド・ビュー・リフレッシュ統計の収集レベルの変更

9.4.1 マテリアライズド・ビュー・リフレッシュ統計の収集について

Oracle Databaseではデフォルトで、すべてのマテリアライズド・ビューのリフレッシュ操作に関する基本的なリフレッシュ統計が収集されます。

Oracle Databaseでは、マテリアライズド・ビュー・リフレッシュ統計を収集する際の粒度やレベルを制御できます。統計は、データベース内のすべてのマテリアライズド・ビューについて収集することも、マテリアライズド・ビューの特定のセットについて収集することもできます。データベース内の一部のマテリアライズド・ビューのみを監視する場合は、マテリアライズド・ビュー・レベルで統計を収集できます。マテリアライズド・ビューのリフレッシュ・パターンは大きく異なることがあるため、選択した一連のマテリアライズド・ビューについてリフレッシュ統計を収集すると役立ちます。

収集レベルによって、データベースがマテリアライズド・ビューのリフレッシュ操作について収集する統計の量が定義されます。基本的な統計を収集することも、マテリアライズド・ビューのリフレッシュ操作の際に使用されたパラメータや実行されたSQL文など、より詳細な情報を収集することもできます。

マテリアライズド・ビュー・リフレッシュ統計の収集レベルを指定するCOLLECTION_LEVELパラメータを設定するには、DBMS_MVIEW_STATSパッケージ内のプロシージャを使用します。COLLECTION_LEVELパラメータに設定可能な値は次のとおりです。

- NONE

  マテリアライズド・ビューのリフレッシュ操作に関する統計は収集されません。

- TYPICAL

  マテリアライズド・ビューのリフレッシュ操作に関する基本的なリフレッシュ統計のみが収集されます。これがデフォルトの設定です。

- ADVANCED

  リフレッシュ操作で使用されたパラメータや実行されたSQL文など、マテリアライズド・ビューのリフレッシュ操作に関する詳細な統計が収集されます。

9.4.2 マテリアライズド・ビュー・リフレッシュ統計の収集に関するデフォルト設定の指定

DBMS_MVIEW_STATS.SET_SYSTEM_DEFAULTプロシージャを使用すると、データベース・レベルでマテリアライズド・ビュー・リフレッ

シュ統計の収集を管理するためのデフォルトを設定できます。

個々のマテリアライズド・ビュー・レベルで異なる設定を指定することにより、システム・デフォルトを上書きすることもできます。デフォルト設定が上書きされていないマテリアライズド・ビューでは、システムのデフォルト設定が使用されます。

Oracle Databaseではデフォルトで、データベース全体のマテリアライズド・ビューのリフレッシュ操作に関する基本的な統計が収集および保存されます。統計の収集レベルを変更することにより、統計の収集を無効にしたり、デフォルト設定を変更できます。

データベース・レベルでマテリアライズド・ビュー・リフレッシュ統計のデフォルトの収集レベルを設定するには:

- DBMS_MVIEW_STATS.SET_SYSTEM_DEFAULTプロシージャを実行し、COLLECTION_LEVELパラメータを設定します。

例9-1 データベースのマテリアライズド・ビュー・リフレッシュ統計収集レベルの設定

この例では、マテリアライズド・ビュー・リフレッシュ統計のデフォルトの収集レベルをADVANCEDに設定して、マテリアライズド・ビューのリフレッシュ操作に関する詳細な統計が収集および保存されるように指定します。

```
DBMS_MVIEW_STATS.SET_SYSTEM_DEFAULT ('COLLECTION_LEVEL','ADVANCED');
```

例9-2 マテリアライズド・ビューのリフレッシュに関する統計の収集の無効化

この例では、マテリアライズド・ビュー・リフレッシュ統計のデフォルトの収集レベルをNONEに設定して、統計の収集を無効化します。

```
DBMS_MVIEW_STATS.SET_SYSTEM_DEFAULT ('COLLECTION_LEVEL','NONE');
```

関連項目:

Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス

9.4.3 マテリアライズド・ビュー・リフレッシュ統計の収集レベルの変更

DBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMSプロシージャを使用すると、マテリアライズド・ビュー・リフレッシュ統計の収集を管理する設定を変更できます。

統計の収集の動作は、データベース全体について変更することも、1つ以上のマテリアライズド・ビューについて変更することもできます。収集に関する新しい設定は、データベース・レベルのデフォルト設定や、指定したマテリアライズド・ビューの以前の設定より優先されます。たとえば、COLLECTION_LEVELのシステム・デフォルトがデータベースでTYPICALに設定されているとします。その後、DBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMSプロシージャを使用して、マテリアライズド・ビューMV1およびMV2の収集レベルをADVANCEDに変更します。データベース内の残りのマテリアライズド・ビューでは、引き続きTYPICAL収集レベルが使用されます。

データベース・レベルまたはマテリアライズド・ビュー・レベルでマテリアライズド・ビュー・リフレッシュ統計の収集レベルを変更するには:

- DBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMSプロシージャを実行し、COLLECTION_LEVELパラメータを目的の値に設定します。

例9-3 データベース全体のマテリアライズド・ビュー統計収集レベルの設定

次の例では、マテリアライズド・ビュー・リフレッシュ統計の収集レベルをデータベース・レベルでTYPICALに変更します。1つ以上のマテリアライズド・ビュー名のかわりにNULLを指定すると、この設定がデータベース全体に適用されることを示します。

```
DBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMS (NULL, 'TYPICAL');
```

例9-4 複数のマテリアライズド・ビューのマテリアライズド・ビュー統計収集レベルの設定

この例では、SHスキーマ内のマテリアライズド・ビューSALES_2013_MVおよびSALES_2014_MVの収集レベルをADVANCEDに設定します。保存期間は60日に設定します。この設定は、データベース・レベルで指定されている可能性があるデフォルト設定より優先されます。

```
DBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMS ('SH.SALES_2013_MV, SH.SALES_2014_MV','ADVANCED',60);
```

関連項目:

Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス

9.5 マテリアライズド・ビュー・リフレッシュ統計の保存

Oracle Databaseでは、保存期間で指定した期間、収集されたマテリアライズド・ビュー・リフレッシュ統計が保存されます。

関連項目:

- マテリアライズド・ビュー・リフレッシュ統計の保存について
- マテリアライズド・ビュー・リフレッシュ統計のデフォルトの保存期間の指定
- マテリアライズド・ビュー・リフレッシュ統計の保存期間の変更

9.5.1 マテリアライズド・ビュー・リフレッシュ統計の保存について

保存期間によって、マテリアライズド・ビュー・リフレッシュ統計がデータ・ディクショナリに保存される期間(日数)が定義されます。保存期間に達すると、収集された統計は自動的に消去されます。

マテリアライズド・ビュー・リフレッシュ統計の保存期間は、データベース・レベルで設定することも、マテリアライズド・ビュー・レベルで設定することもできます。DBMS_MVIEW_STATS.SET_SYSTEM_DEFAULTまたはDBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMSのRETENTION_PERIODパラメータを使用すると、マテリアライズド・ビュー・リフレッシュ統計をデータ・ディクショナリに保存しておく必要がある期間を指定できます。

9.5.2 マテリアライズド・ビュー・リフレッシュ統計のデフォルトの保存期間の指定

DBMS_MVIEW_STATS.SET_SYSTEM_DEFAULTプロシージャでは、データベース・レベルでマテリアライズド・ビュー・リフレッシュ統計の保存を管理するためのデフォルトを設定できます。

Oracle Databaseではデフォルトで、収集日から365日間、マテリアライズド・ビュー・リフレッシュ統計が保存されます。保存期間に達すると、統計はデータ・ディクショナリから消去されます。個々のマテリアライズド・ビュー・レベルで異なる設定を指定することにより、システムのデフォルト設定を上書きすることもできます。デフォルト設定が上書きされていないマテリアライズド・ビューでは、引き続きシステムのデフォルト設定が使用されます。

リフレッシュ統計がデータベースから消去されることがないように指定するには、保存期間を-1に設定します。

データベース全体のデフォルトの保存期間を新しく指定するには:

- DBMS_MVIEW_STATS.SET_SYSTEM_DEFAULTプロシージャのRETENTION_PERIODパラメータを目的の日数に設定します。

例9-5 マテリアライズド・ビュー・リフレッシュ統計の保存期間の設定

この例では、マテリアライズド・ビュー・リフレッシュ統計のデフォルトの保存期間をデータベース全体について60日に設定します。

```
DBMS_MVIEW_STATS.SET_SYSTEM_DEFAULT ('RETENTION_PERIOD',60);
```

例9-6 マテリアライズド・ビュー・リフレッシュ統計の消去の禁止

この例では、マテリアライズド・ビュー・リフレッシュ統計の保存期間を-1に設定することにより、デフォルトの保存期間に達しても、リフレッシュ統計が自動的に消去されることがないように指定します。この設定を使用する場合、DBMS_MVIEW_STATS.PURGE_REFRESH_STATSプロシージャを使用して、リフレッシュ統計をデータ・ディクショナリから明示的に消去する必要があります。

```
DBMS_MVIEW_STATS.SET_SYSTEM_DEFAULT ('RETENTION_PERIOD',-1);
```

関連項目:

Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス

9.5.3 マテリアライズド・ビュー・リフレッシュ統計の保存期間の変更

DBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMSプロシージャでは、マテリアライズド・ビュー・リフレッシュ統計に設定されている保存期間を変更できます。

保存期間は、データベース全体について変更することも、1つ以上のマテリアライズド・ビューについて変更することもできます。特定のマテリアライズド・ビューのみについて保存期間を変更した場合、データベース内の残りのマテリアライズド・ビューでは、引き続き既存の保存期間が使用されます。

システムのデフォルト設定では、基本的なマテリアライズド・ビュー・リフレッシュ統計が収集され、60日間保存されるとします。ただし、マテリアライズド・ビューの特定のセットについては、詳細な統計を収集し、それらの統計を45日間保存することにします。この場合、マテリアライズド・ビューの特定のセットについては、COLLECTION_LEVELをADVANCEDに、RETENTION_PERIODを45に設定します。

データベース・レベルまたはマテリアライズド・ビュー・レベルでマテリアライズド・ビュー・リフレッシュ統計の保存期間を変更するには:

- DBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMSプロシージャを実行し、RETENTION_PERIODパラメータを目的の値に設定します。

例9-7 保存期間に関するマテリアライズド・ビュー・リフレッシュ統計のデフォルト設定の使用

この例では、SHスキーマ内のマテリアライズド・ビューSALES_MVの収集レベルをTYPICALに設定します。保存期間にはNULLを使用するため、このマテリアライズド・ビューでは、保存期間についてはシステム全体のデフォルト設定が使用されます。

```
DBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMS ('SH.SALES_MV','TYPICAL',NULL);
```

例9-8 マテリアライズド・ビューの保存期間の設定

この例では、SH.SALES_MVの収集レベルをADVANCEDに、保存期間を45日に設定します。これは、このマテリアライズド・ビューに設定されている既存の保存期間より優先されます。

```
DBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMS ('SH.SALES_MV','ADVANCED',45);
```

関連項目:

Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス

9.6 マテリアライズド・ビュー・リフレッシュ統計の設定の表示

データ・ディクショナリ・ビューには、マテリアライズド・ビュー・リフレッシュ統計を管理するデフォルト設定とマテリアライズド・ビュー固有の設定の両方が保存されます。

マテリアライズド・ビュー・リフレッシュ統計の収集および保存に関するデータベース・レベルのデフォルト設定を表示するには:

- DBA_MVREF_STATS_SYS_DEFAULTSビューのparameter_nameおよびvalue列を問い合せます。

1つ以上のマテリアライズド・ビューのリフレッシュ統計の収集および保存に関する設定を表示するには:

- mv_ownerおよびmv_name列を使用してデータをフィルタ処理することにより、DBA_MVREF_STATS_PARAMSビューのparameter_nameおよびvalue列を問い合せます。

例9-9 マテリアライズド・ビュー・リフレッシュ統計を管理するためのデータベース・レベルのデフォルト設定の表示

次の問合せでは、マテリアライズド・ビュー・リフレッシュ統計を管理するためのデータベース・レベルのデフォルト設定が表示されます。

```
SELECT parameter_name, value from DBA_MVREF_STATS_SYS_DEFAULTS;

PARAMETER_NAME                 VALUE
---------------                --------
COLLECTION_LEVEL               TYPICAL
RETENTION_PERIOD               45
```

例9-10 一連のマテリアライズド・ビューのリフレッシュ統計に関する設定の表示

次の問合せでは、SHスキーマが所有するすべてのマテリアライズド・ビューのリフレッシュ統計に関する設定が表示されます。

```
SELECT mv_name,collection_level,retention_period
FROM DBA_MVREF_STATS_PARAMS
WHERE mv_owner = 'SH';

MV_NAME                 COLLECTION_LEVEL        RETENTION_PERIOD
--------                -----------------       ------------------
MY_RTMV                 ADVANCED                60
NEW_SALES_RTMV          ADVANCED                45
MY_SUM_SALES_RTMV       TYPICAL                 31
SALES_RTMV              TYPICAL                 -1
CAL_MONTH_SALES_MV      TYPICAL                 45

5 rows selected.
```

9.7 マテリアライズド・ビュー・リフレッシュ統計の消去

DBMS_MVIEW_STATS.PURGE_REFRESH_STATSプロシージャを使用すると、指定した期間より古いマテリアライズド・ビュー・リフレッシュ統計をデータ・ディクショナリから明示的に消去できます。

デフォルトでは、指定した保存期間が経過すると、データ・ディクショナリからマテリアライズド・ビュー・リフレッシュ統計が削除されます。設定に応じて、消去がデータベース全体について実行されることもあれば、指定した一連のマテリアライズド・ビューについて

実行されることもあります。DBMS_MVIEW_STATS.PURGE_REFRESH_STATSプロシージャを使用すると、設定した保存期間を変更することなく、指定した時間より古いリフレッシュ統計を明示的に消去できます。リフレッシュ統計の明示的な消去は保存期間の現在の設定より優先されますが、その設定が変更されることはありません。

データベースに保存されているマテリアライズド・ビュー・リフレッシュ統計を消去するには:

- DBMS_MVIEW_STATS.PURGE_REFRESH_STATSプロシージャを実行します。

  統計を消去する必要があるマテリアライズド・ビューと、統計を消去するまでの期間を指定します。

例9-11 マテリアライズド・ビューのリフレッシュ統計の消去

マテリアライズド・ビューSALES_MVのリフレッシュ統計の保存期間は60日であるとします。過去60日間のリフレッシュ統計はいつでも使用可能です。ただし、領域の制約により、30日が経過した統計を消去することにします。そのためには、DBMS_MVIEW_STATS.PURGE_REFRESH_STATSプロシージャを使用します。

SALES_MVに設定されている保存期間は変更されません。この消去は、1回かぎりの操作です。

```
DBMS_MVIEW_STATS.PURGE_REFRESH_STATS (' SH.SALES_MV' ,30);
```

例9-12 すべてのマテリアライズド・ビューのリフレッシュ統計の消去

この例では、データベース内のすべてのマテリアライズド・ビューについて、20日より古いマテリアライズド・ビュー・リフレッシュ統計を消去します。1つ以上のマテリアライズド・ビューのかわりにNULLを指定すると、この設定がデータベース全体に適用されることを示します。

```
DBMS_MVIEW_STATS.PURGE_REFRESH_STATS (NULL,20);
```

関連項目:

Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス

9.8 マテリアライズド・ビュー・リフレッシュ統計の表示

リフレッシュ統計が保存されているデータ・ディクショナリ・ビューに問い合せることにより、マテリアライズド・ビューのリフレッシュ操作に関する現在と過去両方の統計を表示できます。

収集レベルの設定に応じて、マテリアライズド・ビュー・リフレッシュ統計は、DBA_MVREFS_STATS、DBA_MVREF_RUN_STATS、DBA_MVREF_CHANGE_STATSおよびDBA_MVREF_STMT_STATSのうちの1つ以上のビューに保存されます。これらのビューすべてについて、対応するUSER_バージョンがあります。各ビューには、必要に応じて1つ以上のビューを結合するために使用できるREFRESH_ID列があります。

関連項目:

- マテリアライズド・ビューの基本的なリフレッシュ統計の表示
- それぞれのマテリアライズド・ビュー・リフレッシュ操作に関する詳細な統計の表示
- マテリアライズド・ビュー・リフレッシュ操作時の変更データ統計の表示
- マテリアライズド・ビュー・リフレッシュ操作に関連付けられたSQL文の表示

9.8.1 マテリアライズド・ビューの基本的なリフレッシュ統計の表示

マテリアライズド・ビューのリフレッシュ操作に関する基本的な統計を表示するには、DBA_MVREF_STATSビューを使用します。

マテリアライズド・ビューのリフレッシュ操作はそれぞれ、一意のリフレッシュIDを使用して識別されます。DBA_MVREF_STATSビューには、リフレッシュID、リフレッシュ方法、リフレッシュされたマテリアライズド・ビューの名前、基本的な実行時間、およびリフレッシュ操作のステップの数が保存されます。

マテリアライズド・ビューのリフレッシュ操作に関する基本的なリフレッシュ統計を表示するには:

- 必要な列のリストを指定してDBA_MVREF_STATSビューに問い合せ、条件を使用して必要なデータをフィルタ処理します。

例9-13 マテリアライズド・ビューのリフレッシュ操作に関する基本的な統計の表示

次の問合せでは、SH.NEW_SALES_RTMVマテリアライズド・ビューのリフレッシュ操作に関するいくつかのリフレッシュ統計が表示されます。情報には、リフレッシュ方法、リフレッシュ時間、リフレッシュ操作の開始時におけるマテリアライズド・ビューの行数、およびリフレッシュ操作の終了時における行数が含まれます。

```
SELECT refresh_id, refresh_method, elapsed_time, initial_num_rows, final_num_rows
FROM dba_mvref_stats
WHERE mv_name = 'NEW_SALES_RTMV' and mv_owner = 'SH';

REFRESH_ID    REFRESH_METHOD    ELAPSED_TIME    INITIAL_NUM_ROWS    FINAL_NUM_ROWS
----------    --------------    ------------    ----------------    ----------------
49            FAST              0               766                 788
61            FAST              1               788                 788
81            FAST              1               788                 798

3 rows selected.
```

例9-14 リフレッシュ時間に基づいたマテリアライズド・ビューの表示

次の例では、リフレッシュ操作に10分より長くかかったマテリアライズド・ビューの名前を表示します。elapsed_timeは秒数で指定するため、問合せでは600を使用します。

```
SELECT mv_owner, mv_name, refresh_method
FROM dba_mvref_stats
WHERE elapsed_time > 600;
```

9.8.2 それぞれのマテリアライズド・ビュー・リフレッシュ操作に関する詳細な統計の表示

DBA_MVREF_RUN_STATSビューには、マテリアライズド・ビューのリフレッシュ操作に関する詳細な統計が保存されます。リフレッシュ操作が複数のマテリアライズド・ビューに影響する場合、影響を受けるすべてのマテリアライズド・ビューについて詳細な統計が使用可能です。

マテリアライズド・ビューをリフレッシュするには、DBMS_MVIEWパッケージ内のREFRESH、REFRESH_DEPENDENTまたはREFRESH_ALLプロシージャのいずれかを使用します。各プロシージャには、リフレッシュの実行方法を指定する様々なパラメータがあります。DBA_MVREF_RUN_STATSビューには、リフレッシュ操作で指定されたパラメータ、リフレッシュされたマテリアライズド・ビューの数、実行時間およびログ消去時間に関する情報が格納されます。

マテリアライズド・ビューのリフレッシュ操作に関する詳細なリフレッシュ統計を表示するには:

- 必要な列のリストを指定してDBA_MVREF_RUN_STATSビューに問い合せ、条件を使用して必要なデータをフィルタ処理します。

例9-15 1つのリフレッシュ操作でリフレッシュされたすべてのマテリアライズド・ビューのリスト

次の例では、指定したリフレッシュIDの一部としてリフレッシュされたマテリアライズド・ビューおよび各マテリアライズド・ビューのリフレッシュ時間を表示します。

```
SELECT mviews, elapsed_time, complete_stats_available
FROM dba_mvref_run_stats
WHERE refresh_id = 100;

MVIEWS              ELAPSED_TIME      COMPLETE_STATS_AVAIALBE
--------            ------------      -------------------------
"SH"."SALES_RTMV"   1                 Y
```

例9-16 マテリアライズド・ビューのリフレッシュ操作時に指定されたパラメータの表示

次の例では、リフレッシュID 81のリフレッシュ操作時にリフレッシュされたマテリアライズド・ビューのリストおよび指定されたパラメータの一部を表示します。

```
SELECT mviews, refresh_after_errors, purge_option, parallelism, nested
FROM dba_mvref_run_stats
WHERE run_owner = 'SH' and refresh_id=81;

MVIEWS                  R     PURGE_OPTION    PARALLELISM   NESTED
------                  -     ------------    ------------  -------
"SH"."SALES_RTMV"       N       1               0       N
```

例9-17 マテリアライズド・ビューのリフレッシュ操作に関する詳細な統計の表示

次の例では、リフレッシュID 156のリフレッシュ操作に関する詳細な統計を表示します。この詳細には、リフレッシュされたマテリアライズド・ビューの数、マテリアライズド・ビューの所有者と名前、およびリフレッシュに要した時間が含まれます。

```
SELECT num_mvs, mv_owner, mv_name, r.elapsed_time
FROM dba_mvref_stats s, dba_mvref_run_stats r
WHERE s.refresh_id = r.refresh_id and refresh_id = 156;

NUM_MVS     MV_OWNER    MV_NAME                ELAPSED_TIME
--------    --------    --------               ------------
1           SH          SALES_RTMV             5
```

関連項目:

Oracle Databaseリファレンス

9.8.3 マテリアライズド・ビュー・リフレッシュ操作時の変更データ統計の表示

DBA_MVREF_CHANGE_STATSビューには、マテリアライズド・ビューのリフレッシュ操作に関する詳細な変更データ統計が保存されます。これには、リフレッシュされた実表、挿入された行数、更新された行数、削除された行数、パーティション・メンテナンス操作(PMOP)の詳細などが含まれます。

DBA_MVREF_CHANGE_STATSビューを、マテリアライズド・ビュー・リフレッシュ統計を格納する他のビューと結合すると、より詳細な統計を提供できます。

マテリアライズド・ビューのリフレッシュ操作に関する詳細な変更データ統計を表示するには:

- 必要な列のリストを指定してDBA_MVREF_CHANGE_STATSビューに問い合せ、条件を使用して必要なデータをフィルタ処理します。

例9-18 リフレッシュ操作によってPMOPが発生したかどうかの確認

次の例では、リフレッシュID 1876のリフレッシュ操作について実表名およびPMOPの詳細を表示します。問合せの出力には、マテリアライズド・ビューの実表ごとにレコードが1つずつ表示されます。

```
SELECT tbl_name, mv_name, pmops_occurred, pmop_details
FROM dba_mvref_change_stats
WHERE refresh_id =1876;

TBL_NAME      MV_NAME     PMOPS_OCCURRED    PMOP_DETAILS
---------     --------    --------------    ------------
MY_SALES      SALES_RTMV     N
```

例9-19 リフレッシュ操作時に変更された行数の表示

この例では、SH.MY_SALESマテリアライズド・ビューのリフレッシュ操作における各実表に関する詳細(表内の行数、挿入された行数、更新された行数、削除された行数、ダイレクト・ロード・インサートの数、およびPMOP操作の詳細)を表示します。

```
SELECT tbl_name, num_rows, num_rows_ins, num_rows_upd, num_rows_del, num_rows_dl_ins, pmops_occurred,
pmop_details
FROM dba_mvref_change_stats
WHERE mv_name = 'MY_SALES' and mv_owner = 'SH';
```

関連項目:

Oracle Databaseリファレンス

9.8.4 マテリアライズド・ビュー・リフレッシュ操作に関連付けられたSQL文の表示

マテリアライズド・ビューのリフレッシュ操作で使用されたすべてのSQL文に関する情報を表示するには、DBA_MVREF_STMT_STATSビューに問い合せます。

各リフレッシュ操作は、それぞれがSQL文を使用して実行される複数のステップで構成される場合があります。リフレッシュ操作のそれぞれのステップについて、ステップ番号とSQL文を表示できます。

マテリアライズド・ビューのリフレッシュ操作に関連付けられたSQL文を表示するには:

- 必要な列のリストを指定してDBA_MVREF_STMT_STATSビューに問い合せ、条件を使用して必要なデータをフィルタ処理します。

例9-20 リフレッシュ操作における各ステップのSQL文の表示

次の例では、リフレッシュID 1278のマテリアライズド・ビュー・リフレッシュ操作について、マテリアライズド・ビュー名、マテリアライズド・ビューのリフレッシュに使用されたSQL文、および実行時間を表示します。

```
SELECT mv_name, step, stmt, execution_time
FROM dba_mvref_stmt_stats
WHERE refresh_id = 1278;
```

例9-21 マテリアライズド・ビューの現在のリフレッシュで使用されたリフレッシュ文の表示

この例では、MY_SALESマテリアライズド・ビューのリフレッシュに使用された個々のSQL文を表示します。1つのリフレッシュ操作は、それぞれがSQL文を実行する複数のステップで構成される場合があります。この例で表示される詳細には、ステップ番号、SQL文のSQL ID、実行されたSQL文、およびSQL文の実行時間が含まれます。

```
SELECT step, sqlid, stmt, execution_time
```

```
FROM DBA_MVREF_STATS M, DBA_MVREF_STMT_STATS S
WHERE M.refresh_id = S.refresh_id and M.mv_name = 'MY_SALES'
ORDER BY step;
```

関連項目:

Oracle Databaseリファレンス

9.9 リフレッシュ統計を使用したマテリアライズド・ビューのリフレッシュ・パフォーマンスの分析

データ・ディクショナリ・ビューに保存されているマテリアライズド・ビュー・リフレッシュ統計を使用して、マテリアライズド・ビューのリフレッシュ・パフォーマンスを分析できます。

リフレッシュ統計で提供される詳細な情報を使用すると、マテリアライズド・ビューのリフレッシュ操作とそのパフォーマンスを把握および分析できます。リフレッシュ統計では通常、マテリアライズド・ビューの重要なリフレッシュ操作や長時間実行されているリフレッシュ操作を分析します。マテリアライズド・ビューのリフレッシュに通常より時間がかかる場合、過去のリフレッシュ時間や変更データを分析することで、時間がかかる要因となっている可能性がある差異(今回はリフレッシュする必要があるデータが5倍多いなど)を特定できます。

マテリアライズド・ビューのリフレッシュ・パフォーマンスを分析するには:

1. マテリアライズド・ビューのリフレッシュ統計を一定期間にわたって収集するための収集レベルおよび保存期間を設定します。
   これらをデータベース・レベルで設定することも、マテリアライズド・ビュー・レベルで設定することもできます。
2. リフレッシュ・パフォーマンスを分析する必要があるマテリアライズド・ビューを特定します。
   通常は、データベース内のマテリアライズド・ビューの特定のセットのリフレッシュ・パフォーマンスを分析します。この場合、これらのマテリアライズド・ビューのリフレッシュ統計に関する設定を要件に従って変更できます。
3. (分析するマテリアライズド・ビューについて)一定期間にわたって複数のリフレッシュ操作が実行されると、Oracle Databaseにより、目的のリフレッシュ統計が収集されます。
4. リフレッシュ統計が保存されているデータ・ディクショナリ・ビューに問い合せ、目的のマテリアライズド・ビューのリフレッシュ動作を時間の経過に沿って分析し、リフレッシュ動作を把握します。
   データベースには過去と現在両方の統計が保存され、これらを分析することでリフレッシュ動作を把握できます。

10 ディメンション

この章では、データ・ウェアハウスでのディメンションの使用方法について説明します。内容は次のとおりです。

- ディメンションの概要
- ディメンションの作成
- ディメンションの表示
- ディメンションおよび制約の使用
- ディメンションの妥当性チェック
- ディメンションの変更
- ディメンションの削除

10.1 ディメンションの概要

ディメンションとは、エンド・ユーザーがビジネス上の質問に答えることができるように、データを分類する構造です。最も一般的なディメンションは、顧客(customers)、製品(products)および時間(times)です。たとえば、衣料品の小売店チェーンの各店舗では、各種衣類の売上と再生利用に関するデータを収集および格納している場合があります。小売店チェーンの管理者は、データ・ウェアハウスを作成して、全店舗にわたる長期間の製品の売上を分析できます。また、次のような質問に答えることができます。

- 1つの製品の宣伝が、宣伝していない関係製品の売上に対してどのような影響があるか
- 宣伝前後の製品の売上はいくらか
- 宣伝が各種物流チャネルにどのように影響するか

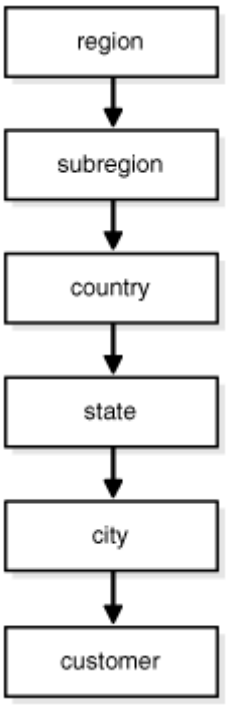
小売店のデータ・ウェアハウス・システムのデータには、ディメンションとファクトという2つの重要なコンポーネントがあります。ディメンションは、製品、顧客、宣伝、チャネルおよび時間です。ディメンションを識別する1つの方法は、製品に関するすべての情報を含む製品表や、宣伝に関するすべての情報を含む宣伝表など、参照している表を調べることです。ファクトは、売上(売上数量)および利益です。データ・ウェアハウスには、1日ごとの各製品の売上に関するファクトが含まれます。

このようなデータ・ウェアハウスの典型的なリレーショナル実装が、スター・スキーマです。ファクト情報はファクト表に格納され、ディメンション情報はディメンション表に格納されます。たとえば、それぞれの売上トランザクション・レコードは、顧客別、製品別、販売チャネル別、宣伝別、および日付(時間)別に一意に定義されます。

Oracle Databaseでは、ディメンション情報自体がディメンション表に格納されます。また、データベース・オブジェクト・ディメンションにより、ディメンション情報を階層形式に編成してグループ化できます。これは、制約では表すことのできない列間または列グループ(階層レベル)間の1:nの関係を表します。階層内でのレベルを上げることはデータのロールアップと呼ばれ、レベルを下げることはデータのドリルダウンと呼ばれます。小売店の例では、次のようになります。

- 時間ディメンションでは、月は四半期に、四半期は年に、年は全年にそれぞれロールアップされます。
- 製品ディメンションでは、製品はサブカテゴリに、サブカテゴリはカテゴリに、カテゴリは全製品にそれぞれロールアップされます。
- 顧客ディメンション内では、顧客は**市**にロールアップされます。市は州にロールアップされます。州は**国**にロールアップされます。国は**地区**にロールアップされます。最後に、地区は**地域**にロールアップされます。これを図10-1に示します。

図10-1 顧客ディメンションのロールアップの例



通常、データ分析はディメンション階層の高いレベルから開始され、必要に応じて徐々に階層がドリルダウンされます。

ディメンション・スキーマ・オブジェクト(ディメンション)を定義する必要はありません。ただし、アプリケーションでディメンショナル・モデリングを使用する場合は、ディメンションを注意深く作成すると、クエリー・リライトで複雑なタイプのリライトを実行できるようになり、大きなメリットが得られます。また、ディメンションは、マテリアライズド・ビューの特定のタイプのリフレッシュ操作およびSQLアクセス・アドバイザに対してもメリットをもたらします。ディメンションが必須になるのは、SQLアクセス・アドバイザ(マテリアライズド・ビューおよび索引管理用のGUIツール)を使用して、ワークロードを発生させることなく、作成、削除または保持すべきマテリアライズド・ビューおよび索引を推奨させる場合のみです。

この章で説明するディメンションの関係を完全に満たさないスキーマには、メリットの有無に関係なく、ディメンションを作成しないでください。作成すると、問合せで不適切な結果が戻される場合があります。

関連項目:

- スキーマの詳細は、データ・ウェアハウスの最適化および技法を参照してください。
- クエリー・リライトの詳細は、マテリアライズド・ビューのための基本的なクエリー・リライトを参照してください。
- SQLアクセス・アドバイザの詳細は、『Oracle Database SQLチューニング・ガイド』を参照してください。

10.1.1 データ・ウェアハウスのディメンションの要件

- 親と子の間に、1:nの関係がある必要があります。親は、1つ以上の子を持つことができますが、子は1つの親しか持てません。
- 階層レベルとその依存ディメンション属性の間に、1:1の属性関係がある必要があります。たとえば、fiscal_month_desc列がある場合、可能な属性関係はfiscal_month_desc対fiscal_month_nameになります。スキップ・レベルにある関係の行にNULL値を持つレベル列がある場合にNULLレベルをスキップするには、その行にもNULL値を持つ属性関係の列がなければなりません。
- 親レベルと子レベルの列が異なる表に存在する場合、これらの間の結合も1:nの結合関係がある必要があります。SKIP WHEN NULL句を使用しない場合、子表の各行は、親表の1つの行のみと結合している必要があります。この関係は、子の結合キーがNULLでないこと、子の結合キーと親の結合キーの参照整合性が保持されること、および親の

結合キーは一意であることが必要であるため、参照整合性のみより強力です。

- SKIP WHEN NULL句を使用しない場合は、各階層レベルの列がNULLでないこと、および階層の整合性が保たれていることを確認する必要があります(必要に応じて、データベース制約を使用してください)。
- オプションの結合キーは、階層内のスキップ・レベル(存在する場合)にあるスキップ対象外の子であるCHILDLEVと、そのスキップ・レベル(同様に、存在する場合)にあるスキップ対象外の最も近い祖先であるANCLEVを接続する結合キーです。また、この結合キーは、異なる関係にわたってCHILDLEVおよびANCLEVが定義されている場合にのみ使用可能です。
- ディメンションの階層は、相互にオーバーラップすることも切り離されることもあります。ただし、階層レベルの列を2つ以上のディメンションに関連付けることはできません。
- ディメンションの図式内で循環を形成する階層の定義はサポートされません。たとえば、階層レベルは、それ自体とは直接的にも間接的にもつなげることはできません。

> ノート:
>
> ディメンション・オブジェクトに格納される情報は宣言のみです。前述の依存関係は、ディメンション・オブジェクトの作成のみでは施行されません。ディメンションの妥当性チェックで説明するように、すべてのディメンション定義は DBMS_DIMENSION.VALIDATE_DIMENSION プロシージャで検証する必要があります。

10.2 ディメンションの作成

ディメンション・オブジェクトを作成する前に、ディメンション・データを含み得るディメンション表が、データベース内に存在する必要があります。たとえば、顧客ディメンションを作成する場合は、市、州および国の情報を含む1つ以上の表が存在する必要があります。スター・スキーマ・データ・ウェアハウスには、これらのディメンション表がすでに存在しています。したがって、どれが使用されるかを簡単に識別できます。

図10-1のようなディメンションの階層を定義できます。たとえば、市は州および国の子です(市レベルのデータは州レベルまで集計できるため)。この階層情報は、データベース・オブジェクト・ディメンションに格納されます。

複数の表に格納されているディメンションが正規化、もしくは部分的に正規化されている場合、これらの表がどのように結合されているか把握する必要があります。ディメンション表同士の結合においては、親表の各行と子表の各行の間に1対多の関係が保証されているかどうかに注意してください。また非正規化ディメンションの場合、子の列は親(または属性)の列を一意に決定できなくてはなりません。制約を使用してこれらの関係を表す場合、制約によって表される関係が他の手段で保証されるのであれば、NOVALIDATE句およびRELY句を使用して制約を使用可能にできます。

ディメンションのNULLのレベルは必要に応じてスキップできます。たとえば、プエルトリコの例を考えてみます。プエルトリコを北アメリカ地域に含めたいが、州のカテゴリには含めたくないとします。これを実行するには、SKIP WHEN NULL句を使用します。詳細は、この章の後半に記載されているサンプル・ディメンションを参照してください。構文および制限については、『Oracle Database SQL言語リファレンス』を参照してください。

ディメンションを作成するには、CREATE DIMENSION文またはOracle Enterprise Managerのディメンション・ウィザードのいずれかを使用します。CREATE DIMENSION文中では、LEVEL句を使用してディメンション・レベルの名前を指定します。

図10-1に示すとおり、この顧客ディメンションには地理的にロールアップする階層が1つあります(図では子レベルと親レベルの間に矢印が描かれています)。

この図式内の各矢印は、すべての子に対して親が1つのみあることを示します。たとえば、各市は1つの州のみに含まれる必要があり、各州は1つの国のみに含まれる必要があります。2つ以上の国に属する州は、階層の整合性に違反します。また、州に属

さない市(ワシントンD.C.など)を含めたい場合は、SKIP WHEN NULL句を使用する必要があります。階層の整合性は、集計を含むマテリアライズド・ビューに対する管理機能を正確に操作するために必要です。

たとえば、product、subcategoryおよびcategoryのレベルを含むディメンションproducts_dimを宣言できます。

```
CREATE DIMENSION products_dim
       LEVEL product           IS (products.prod_id)
       LEVEL subcategory       IS (products.prod_subcategory)
       LEVEL category          IS (products.prod_category) ...
```

ディメンション内の各レベルは、データベースにある表の1つ以上の列に対応付けられている必要があります。したがって、レベルproductはproducts表の列prod_idで識別され、レベルsubcategoryは同じ表の列prod_subcategoryで識別されます。

この例では、データベース表は非正規化され、すべての列は同じ表に存在します。ただし、これはディメンション作成の前提条件ではありません。JOIN KEY句を使用した、正規化スキーマ設計を持つディメンションcustomers_dimの作成方法については、正規化ディメンション表を使用したディメンションの作成を参照してください。

次のステップでは、HIERARCHY文でレベル間の関係を宣言し、階層に名前を付けます。階層関係とは、階層内の1つのレベルから次のレベルに対する機能的な依存関係です。前に定義したレベルの名前を使用して、CHILD OF関係によって、各子のレベル値が1つのみの親レベル値に対応付けられていることが示されます。次の文では、prod_rollupという階層を宣言し、product、subcategoryおよびcategory間の関係を定義しています。

```
HIERARCHY prod_rollup
 (product         CHILD OF
  subcategory     CHILD OF
  category)
```

1:nの階層関係に加えて、ディメンションには、階層レベルとその依存ディメンション属性との間の1:1の属性関係も含まれます。たとえば、『Oracle Databaseサンプル・スキーマ』で定義されているように、ディメンションtimes_dimにはfiscal_month_desc、fiscal_month_nameおよびdays_in_fiscal_monthの各列があります。この関係の定義は、次のとおりです。

```
LEVEL fis_month IS TIMES.FISCAL_MONTH_DESC
...
ATTRIBUTE fis_month DETERMINES
      (fiscal_month_name, days_in_fiscal_month)
```

ATTRIBUTE ... DETERMINES句は、fis_monthをfiscal_month_nameとdays_in_fiscal_monthに関係付けます。これが単方向の依存関係であることに注意してください。保証されているのは、特定のfiscal_month(1999-11など)について、fiscal_month_nameおよびdays_in_fiscal_monthに一致する値がそれぞれ1つ(Novemberおよび28など)のみであることです。各会計年度のNovemberであるfiscal_month_nameに基づいて特定のfiscal_month_descを判断することはできません。

この例では、fiscal_month_descではなくfiscal_month_nameで問合せが発行されたとします。この1:1の関係は属性とレベルの間に存在するため、fiscal_month_descを含んでいる集計済のマテリアライズド・ビューをディメンション情報に後戻り結合し、データの識別に使用できます。

ディメンション定義の例を次に示します。

```
CREATE DIMENSION products_dim
    LEVEL product           IS (products.prod_id)
    LEVEL subcategory       IS (products.prod_subcategory) [SKIP WHEN NULL]
    LEVEL category          IS (products.prod_category)
    HIERARCHY prod_rollup (
         product         CHILD OF
         subcategory     CHILD OF
         category)
    ATTRIBUTE product DETERMINES
```

```
     (products.prod_name, products.prod_desc,
      prod_weight_class, prod_unit_of_measure,
      prod_pack_size, prod_status, prod_list_price, prod_min_price)
    ATTRIBUTE subcategory DETERMINES
     (prod_subcategory, prod_subcategory_desc)
    ATTRIBUTE category DETERMINES
     (prod_category, prod_category_desc);
```

または、次の例に示すように、*attribute_clause*のかわりに*extended_attribute_clause*を使用することもできます。

```
CREATE DIMENSION products_dim
    LEVEL product           IS (products.prod_id)
    LEVEL subcategory       IS (products.prod_subcategory)
    LEVEL category          IS (products.prod_category)
    HIERARCHY prod_rollup (
         product         CHILD OF
         subcategory     CHILD OF
         category
        )
    ATTRIBUTE product_info LEVEL product DETERMINES
     (products.prod_name, products.prod_desc,
      prod_weight_class, prod_unit_of_measure,
      prod_pack_size, prod_status, prod_list_price, prod_min_price)
    ATTRIBUTE subcategory DETERMINES
     (prod_subcategory, prod_subcategory_desc)
    ATTRIBUTE category DETERMINES
     (prod_category, prod_category_desc);
```

ディメンションの設計、作成およびメンテナンスは、データ・ウェアハウス・スキーマの設計、作成およびメンテナンスの一部です。ディメンションを作成したら、データ・ウェアハウスにおけるディメンションの要件に記載されている要件を満たしているか検証します。

関連項目:

- ディメンション情報の使用方法の詳細は、マテリアライズド・ビューのための基本的なクエリー・リライトを参照してください。
- CREATE DIMENSION文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

10.2.1 属性列の削除および作成

CREATE DIMENSION文でATTRIBUTE句を使用すると、階層レベルによって一意に決定される1つ以上の列を指定できます。

*extended_attribute_clause*を使用して、階層レベルによって決定される複数の列を作成する場合は、すべての属性列を削除することなく1つの属性列を削除できます。また、CREATEまたはALTER DIMENSION文で各ATTRIBUTE句に属性名を指定すると、ATTRIBUTE句ごとに属性名が付けられ、複数のレベル-列関係を個々に指定できるようになります。

次に、すべての列を削除することなく1つの列を削除できるようにする文を示します。

```
CREATE DIMENSION products_dim
LEVEL product         IS (products.prod_id)
LEVEL subcategory     IS (products.prod_subcategory)
LEVEL category        IS (products.prod_category)
HIERARCHY prod_rollup (
          product        CHILD OF
          subcategory    CHILD OF category)
ATTRIBUTE product DETERMINES
         (products.prod_name, products.prod_desc,
```

```
          prod_weight_class, prod_unit_of_measure,
          prod_pack_size,prod_status, prod_list_price, prod_min_price)
ATTRIBUTE subcategory_att DETERMINES
         (prod_subcategory, prod_subcategory_desc)
ATTRIBUTE category DETERMINES
         (prod_category, prod_category_desc);

ALTER DIMENSION products_dim
DROP ATTRIBUTE subcategory_att LEVEL subcategory COLUMN prod_subcategory;
```

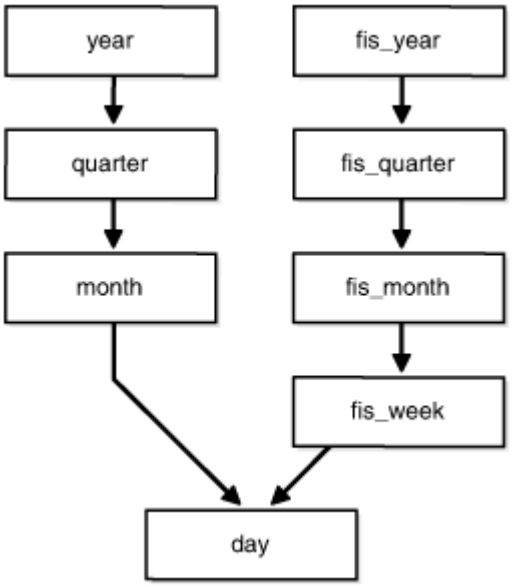
関連項目:

CREATE DIMENSION文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

10.2.2 結合作成時における複数の階層の使用

単一のディメンション定義に、複数の階層を含めることができます。小売店で、ある品目の売上を長期間追跡すると想定します。最初のステップは、売上が追跡される時間の時間ディメンションを定義することです。図10-2に、2つの時間階層を持つディメンションtimes_dimを示します。

図10-2 2つの時間階層を持つtimes_dimディメンション



この図から、次のCREATE DIMENSION文によって定義された非正規化time_dimディメンションの階層を作成できます。

```
CREATE DIMENSION times_dim
   LEVEL day          IS times.time_id
   LEVEL month        IS times.calendar_month_desc
   LEVEL quarter      IS times.calendar_quarter_desc
   LEVEL year         IS times.calendar_year
   LEVEL fis_week     IS times.week_ending_day
   LEVEL fis_month    IS times.fiscal_month_desc
   LEVEL fis_quarter IS times.fiscal_quarter_desc
   LEVEL fis_year     IS times.fiscal_year
   HIERARCHY cal_rollup    (
             day      CHILD OF
             month    CHILD OF
             quarter CHILD OF
             year
   )
```

```
  HIERARCHY fis_rollup    (
            day         CHILD OF
            fis_week    CHILD OF
            fis_month   CHILD OF
            fis_quarter CHILD OF
            fis_year
  ) <attribute determination clauses>;
```

10.2.3 正規化ディメンション表を使用したディメンションの作成

ディメンションの定義に使用される表は、正規化または非正規化されている場合があります。また、個々の階層は、正規化または非正規化できます。1つの階層のレベルが同じ表のものである場合は、完全な非正規化階層と呼ばれます。たとえば、times_dimディメンション内のcal_rollupは、非正規化階層です。1つの階層のレベルが異なる表に存在する場合、このような階層は、完全なまたは部分的な正規化階層です。この項では、正規化階層の定義方法を示します。

顧客の所在地が市、州および国別に追跡されているとします。このデータは、customers表およびcountries表に格納されます。データ・エンティティcust_idおよびcountry_idが別々の表から取り出されるため、顧客ディメンションcustomers_dimは部分的に正規化されます。ディメンション定義内のJOIN KEY句では、階層内のレベルの結合方法が指定されます。ディメンションの文の一部を次に示します。

```
CREATE DIMENSION customers_dim
   LEVEL customer  IS (customers.cust_id)
   LEVEL city      IS (customers.cust_city)
   LEVEL state     IS (customers.cust_state_province)
   LEVEL country   IS (countries.country_id)
   LEVEL subregion IS (countries.country_subregion)
   LEVEL region IS (countries.country_region)
   HIERARCHY geog_rollup (
      customer        CHILD OF
      city            CHILD OF
      state           CHILD OF
      country         CHILD OF
      subregion       CHILD OF
      region
   JOIN KEY (customers.country_id) REFERENCES country);
```

SKIP WHEN NULL句を使用する場合、JOIN KEY句を使用して階層内に存在しないレベルを持つレベル同士をリンクできます。たとえば、次の文によって、SKIP WHEN NULLと宣言された州レベルを市および国に結合できます。

```
JOIN KEY (city.country_id) REFERENCES country;
```

これにより、顧客レベルおよび市レベルの行が、国レベル、地区レベルおよび地域レベルの行に関連付けられます。

10.3 ディメンションの表示

ディメンションは、次のいずれかの方法で参照できます。

- Oracle Enterprise Managerによるディメンションの表示
- DESCRIBE_DIMENSIONプロシージャによるディメンションの表示

10.3.1 Oracle Enterprise Managerによるディメンションの表示

Oracle Enterprise Managerを使用すると、データ・ウェアハウス内にあるすべてのディメンションを表示できます。すべてのディメンションを表示するには、「**スキーマ**」アイコンから「**ディメンション**」オブジェクトを選択します。特定のディメンションを選択すると、

その定義された階層、レベルおよび属性がグラフィカルに表示されます。

10.3.2 DESCRIBE_DIMENSIONプロシージャによるディメンションの表示

ディメンションの定義を表示するには、DBMS_DIMENSIONパッケージのDESCRIBE_DIMENSIONプロシージャを使用します。たとえば、ディメンションが次の文でshサンプル・スキーマに作成されているとします。

```
CREATE DIMENSION channels_dim
        LEVEL channel       IS (channels.channel_id)
        LEVEL channel_class IS (channels.channel_class)
        HIERARCHY channel_rollup (
                channel CHILD OF channel_class)
        ATTRIBUTE channel DETERMINES (channel_desc)
        ATTRIBUTE channel_class DETERMINES (channel_class);
```

次のようにしてDESCRIBE_DIMENSIONプロシージャを実行します。

```
SET SERVEROUTPUT ON FORMAT WRAPPED;  --to improve the display of info
EXECUTE DBMS_DIMENSION.DESCRIBE_DIMENSION('SH.CHANNELS_DIM');
```

結果が次のように出力されます。

```
EXECUTE DBMS_DIMENSION.DESCRIBE_DIMENSION('SH.CHANNELS_DIM');
  DIMENSION SH.CHANNELS_DIM
    LEVEL CHANNEL IS SH.CHANNELS.CHANNEL_ID
    LEVEL CHANNEL_CLASS IS SH.CHANNELS.CHANNEL_CLASS

    HIERARCHY CHANNEL_ROLLUP (
              CHANNEL CHILD OF
              CHANNEL_CLASS)

    ATTRIBUTE CHANNEL LEVEL CHANNEL DETERMINES
SH.CHANNELS.CHANNEL_DESC
    ATTRIBUTE CHANNEL_CLASS LEVEL CHANNEL_CLASS DETERMINES
SH.CHANNELS.CHANNEL_CLASS
```

10.4 ディメンションおよび制約の使用

制約は、ディメンションに対して重要な役割を果します。データ・ウェアハウスでは、完全な参照整合性が有効化されますが、常にそうである必要はありません。これは、通常、業務系データベースは完全な参照整合性を持っており、データ・ウェアハウスに送られるデータがすでに設定されている整合性ルールに違反しないことを保証できるためです。

制約を使用可能にし、妥当性チェックの時間を考慮する必要がある場合に、次のようにNOVALIDATE句を使用することをお薦めします。

```
ENABLE NOVALIDATE CONSTRAINT pk_time;
```

主キーおよび外部キーも実装する必要があります。ファクト表の参照整合性制約およびNOT NULL制約によって、マテリアライズド・ビューの有用性を拡張するクエリー・リライトの機能で利用する情報が提供されます。

また、次のようにRELY句を使用して、制約が正しいことをクエリー・リライトに提示する必要があります。

```
ALTER TABLE time MODIFY CONSTRAINT pk_time RELY;
```

この情報は、クエリー・リライトにも使用されます。詳細は、マテリアライズド・ビューのための基本的なクエリー・リライトを参照してください。

SKIP WHEN NULL句を使用する場合、NOT NULL制約を持たない参照レベル列が少なくとも1つ必要です。

10.5 ディメンションの妥当性チェック

ディメンション・オブジェクトの情報は宣言のみで、データベースでは規定されません。ディメンションによって表された関係が不適切な場合は、不適切な結果が戻される可能性があります。したがって、DBMS_DIMENSION.VALIDATE_DIMENSIONプロシージャを定期的に使用して、CREATE DIMENSIONで指定される関係を検証する必要があります。

このプロシージャは使いやすく、指定するパラメータは次の4つのみです。

- dimension: 所有者と名前。
- incremental: このディメンションの表の新しい行のみをチェックするにはTRUEに設定。
- check_nulls: SKIP WHEN NULL句を含むレベルにないすべての列がNULLではないことを検証するにはTRUEに設定。
- statement_id: プロシージャの各実行結果を識別するためにユーザーが指定する一意の識別子。

次の例では、shスキーマ内のTIME_FNディメンションの妥当性チェックが行われます。

```
@utldim.sql
EXECUTE DBMS_DIMENSION.VALIDATE_DIMENSION ('SH.TIME_FN', FALSE, TRUE,
  'my first example');
```

VALIDATE_DIMENSIONプロシージャを実行する前に、提供されているスクリプトutldim.sqlを実行して、ローカル表DIMENSION_EXCEPTIONSを作成しておく必要があります。VALIDATE_DIMENSIONプロシージャでエラーが検出されると、この表に書き込まれます。この表を問い合せると、検出されたエラーを識別できます。次に例を示します。

```
SELECT * FROM dimension_exceptions
WHERE statement_id = 'my first example';

STATEMENT_ID       OWNER  TABLE_NAME  DIMENSION_NAME   RELATIONSHIP   BAD_ROWID
------------       -----  ----------  --------------   ------------   ----------
my first example   SH     MONTH       TIME_FN          FOREIGN KEY    AAAAuwAAJAAAARwAAA
```

ただし、この表を問い合せるより、無効な行のROWIDを使用して、制約に違反する実際の行を取り出す問合せの方が適している場合があります。この例では、TIME_FNディメンションが、month表をチェックしています。これにより、制約に違反する行が検出されています。ROWIDを使用すると、次のように、month表のうち問題の原因となっている行を正確に確認できます。

```
SELECT * FROM month
WHERE rowid IN (SELECT bad_rowid
                FROM dimension_exceptions
                WHERE statement_id = 'my first example');

MONTH    QUARTER   FISCAL_QTR    YEAR    FULL_MONTH_NAME    MONTH_NUMB
------   -------   ----------    ----    ---------------    ----------
199903     19981         19981    1998              March             3
```

10.6 ディメンションの変更

ALTER DIMENSION文を使用すると、ディメンションを変更できます。このコマンドを使用して、レベル、階層または属性をディメンションに対して追加または削除できます。

図10-2の時間ディメンションの場合では、属性fis_year、階層fis_rollupまたはレベルfiscal_yearを削除できます。また、次のようにf_yearという新しいレベルを追加できます。

```
ALTER DIMENSION times_dim DROP ATTRIBUTE fis_year;
```

```
ALTER DIMENSION times_dim DROP HIERARCHY fis_rollup;
ALTER DIMENSION times_dim DROP LEVEL fis_year;
ALTER DIMENSION times_dim ADD LEVEL f_year IS times.fiscal_year;
```

ディメンションの作成時に*extended_attribute_clause*を使用した場合は、すべての属性列を削除することなく1つの属性列を削除できます。これは、「属性列の削除および作成」に示されているように、次のような文で行うことができます。

```
ALTER DIMENSION product_dim
DROP ATTRIBUTE size LEVEL prod_type COLUMN Prod_TypeSize;
```

ディメンション内でさらに依存性を持つオブジェクトの削除を試みると、そのディメンションの変更はOracle Databaseにより拒否されます。ディメンションが参照しているスキーマ・オブジェクトのいずれかを変更した場合、そのディメンションは無効になります。たとえば、ディメンションが定義されている表が変更されると、そのディメンションは無効になります。

次の文のように、SKIP WHEN NULL句を含むレベルを追加することによってディメンションを変更できます。

```
ALTER DIMENSION times_dim
ADD LEVEL f_year IS times.fiscal_year SKIP WHEN NULL;
```

ただし、SKIP WHEN NULL句を含むレベルは変更できません。かわりに、そのレベルを削除して再作成する必要があります。

ディメンションの状態をチェックするには、ALL_DIMENSIONSデータ・ディクショナリ・ビューにあるinvalid列の内容を参照します。ディメンションを再検証するには、次のようにCOMPILEオプションを使用します。

```
ALTER DIMENSION times_dim COMPILE;
```

ディメンションは、Oracle Enterprise Managerを使用しても変更または削除できます。

10.7 ディメンションの削除

ディメンションは、DROP DIMENSION文を使用して削除できます。たとえば、次のような文を発行します。

```
DROP DIMENSION times_dim;
```

11 マテリアライズド・ビューのための基本的なクエリー・リライト

この章では、Oracleのクエリー・リライトについて説明します。次の内容が含まれます。

- クエリー・リライトの概要
- クエリー・リライトの有効化
- クエリー・リライトの例

11.1 クエリー・リライトの概要

実表にデータが大量に格納されていると、必要な集計やこれらの表間の結合の計算にコストや時間が多くかかります。こうした場合の問合せは数分から数時間を要することもあります。マテリアライズド・ビューにはすでに計算された集計と結合が含まれているため、Oracle Databaseでは、クエリー・リライトと呼ばれる非常に強力なプロセスが採用されており、マテリアライズド・ビューを使用した問合せに迅速に応答します。

クエリー・リライトが利用可能になることは、マテリアライズド・ビューを作成しメンテナンスすることで得られる主要なメリットの1つです。クエリー・リライトでは、複数の表やビューに対するSQL文を、ディテール表に定義された1つ以上のマテリアライズド・ビューに対してアクセスする文に変換します。この変換はエンド・ユーザーやアプリケーションに対して透過的に処理され、SQL文内のマテリアライズド・ビューに対する介入や参照は不要です。クエリー・リライトは透過的な処理なので、マテリアライズド・ビューは、アプリケーション・コード内のSQLを無効にせず索引のように追加したり削除したりできます。Oracleによるクエリー・リライト条件では、リライト対象の問合せが満たす必要がある条件について説明します。

11.1.1 クエリー・リライトとオプティマイザについて

問合せは、その問合せにクエリー・リライトが必要かどうかを判断するチェックを受けます。チェック結果に問題があると、その問合せはマテリアライズド・ビューではなく、ディテール表に対する問合せになります。リライトできないと、応答時間や処理能力についての効率が低下する場合があります。

オプティマイザでは、マテリアライズド・ビューが関わる問合せをリライトする場合の判断に2つの方法が使用されます。最初の方法で、問合せのSQLテキストとマテリアライズド・ビュー定義のSQLテキストが照合されます。この方法で判断できない場合、問合せとマテリアライズド・ビューの結合、選択、データ列、グルーピング列および集計関数を比較するという、より一般的な方法が取られます。

クエリー・リライトは、次のSQL文の問合せおよび副問合せに対応します。

- SELECT
- CREATE TABLE … AS SELECT
- INSERT INTO … SELECT

また、集合演算子UNION、UNION ALL、, INTERSECT、MINUS、およびDML文の副問合せ(INSERT、DELETE、UPDATEなど)にも対応します。

マテリアライズド・ビューを使用するように問合せがリライトされるかどうかには、ディメンション、制約およびリライトの整合性レベルが影響します。また、問合せのリライトはREWRITEヒントやNOREWRITEヒント、およびQUERY_REWRITE_ENABLEDセッション・パラメータを使用して有効化したり無効化したりできます。

問合せでクエリー・リライトが可能かどうか、また可能な場合はどのマテリアライズド・ビューが使用されるかについては、DBMS_MVIEW.EXPLAIN_REWRITEプロシージャで示されます。また、このプロシージャでは問合せをリライトできない理由もわかります。

11.1.2 Oracleによるクエリー・リライト条件

問合せは、次の一定の条件が満たされた場合にのみリライトされます。

- セッションで、クエリー・リライトが使用可能である必要があります。
- マテリアライズド・ビューに対するクエリー・リライトが使用可能である必要があります。
- リライトの整合性レベルが、マテリアライズド・ビューの使用を許可している必要があります。たとえば、マテリアライズド・ビューが最新のものではなく、かつクエリー・リライトの整合性がENFORCEDに設定されている場合、マテリアライズド・ビューは使用できません。
- 問合せの要求結果の一部またはすべてが、1つ以上のマテリアライズド・ビューに格納されている、事前計算された結果から取得可能である必要があります。

オプティマイザでは、こうした条件のテストに、ユーザーが制約やディメンションを使用して宣言したデータ関係が使用される場合があります。そうしたデータ関係には、階層、参照整合性、キー・データの一意性などがあります。

11.2 クエリー・リライトの有効化

クエリー・リライトを使用可能にするには、次の手順を実行する必要があります。

1. 個々のマテリアライズド・ビューに、ENABLE QUERY REWRITE句を指定します。

   マテリアライズド・ビューでのクエリー・リライトの有効化の説明に従ってこのステップを完了しないと、クエリー・リライトでマテリアライズド・ビューを使用できません。

2. セッション・パラメータQUERY_REWRITE_ENABLEDをTRUE(デフォルト)またはFORCEに設定します。

   クエリー・リライトの初期化パラメータを参照してください。

3. 初期化パラメータOPTIMIZER_MODEを、ALL_ROWS、FIRST_ROWSまたはFIRST_ROWS_*n*に設定して、コストベース・オプティマイザを使用します。

   クエリー・リライトの初期化パラメータを参照してください。

DBMS_ADVISOR.TUNE_MVIEWプロシージャを使用すると、CREATE MATERIALIZED VIEW文を最適化して、一般的なQUERY REWRITEを使用可能にできます。

11.2.1 マテリアライズド・ビューのクエリー・リライトの有効化

ENABLE QUERY REWRITEは、ALTER MATERIALIZED VIEW文を使用して指定するか、マテリアライズド・ビューの作成時に次のように指定できます。

```
CREATE MATERIALIZED VIEW join_sales_time_product_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_id, p.prod_name, t.time_id, t.week_ending_day,
       s.channel_id, s.promo_id, s.cust_id, s.amount_sold
FROM   sales s, products p, times t
WHERE  s.time_id=t.time_id  AND s.prod_id = p.prod_id;
```

NOREWRITEヒントを使用すると、QUERY_REWRITE_ENABLEDパラメータをオーバーライドして、SQL文のクエリー・リライトを使用禁止にできます。REWRITEヒント(mv_nameと併用)を使用すると、クエリー・リライトを使用できるマテリアライズド・ビューを、ヒントで指定するビューのみに制限できます。

11.2.2 クエリー・リライトの初期化パラメータについて

クエリー・リライトの動作は、特定のデータベース初期化パラメータによって制御されます。

表11-1 クエリー・リライト動作を制御する初期化パラメータ

初期化パラメータ名	初期化パラメータ値	クエリー・リライトの動作
OPTIMIZER_MODE	ALL_ROWS (デフォルト)、FIRST_ROWS または FIRST_ROWS_*n*	OPTIMIZER_MODE を FIRST_ROWS に設定すると、コストと発見的方法の組合せによって、最初の数行を高速に配信するために最適な計画が求められます。FIRST_ROWS_*n* に設定すると、コストベースのアプローチが使用され、初めの *n* 行(n = 1、10、100、1000)を戻すまでの時間が最も速くなるように最適化されます。
QUERY_REWRITE_ENABLED	TRUE (デフォルト)、FALSE または FORCE	このオプションを使用すると、オプティマイザのクエリー・リライト機能が有効化されてマテリアライズド・ビューを利用できるようになり、パフォーマンスが向上します。FALSE に設定すると、オプティマイザのクエリー・リライト機能が無効になり、リライトされていない問合せの見積り問合せコストの方が低くなる場合にも、マテリアライズド・ビューを使用した問合せのリライトが行われません。 FORCE に設定すると、オプティマイザのクエリー・リライト機能が有効になり、リライトされない問合せの見積り問合せコストの方が低くなる場合にも、マテリアライズド・ビューを使用して問合せがリライトされます。
QUERY_REWRITE_INTEGRITY	STALE_TOLERATED、TRUSTED または ENFORCED (デフォルト)	このパラメータはオプションです。ただし、設定されている場合は、「初期化パラメータ値」列に指定されている値のいずれかを指定する必要があります。 デフォルトでは、整合性レベルは ENFORCED に設定されます。このモードでは、すべての制約の妥当性チェックを行う必要があります。そのため、ENABLE NOVALIDATE RELY を使用した場合、一部のクエリー・リライトが動作しないことがあります。この環境でクエリー・リライトを使用可能にする(つまり、制約の妥当性チェックが行われないようにする)には、整合

初期化パラメータ名	初期化パラメータ値	クエリー・リライトの動作
		性レベルを TRUSTED や STALE_TOLERATED のような低いレベルに設定する必要があります。

関連トピック

- クエリー・リライトの精度について

11.2.3 クエリー・リライトの制御

マテリアライズド・ビューをクエリー・リライトに使用できるのは、ENABLE QUERY REWRITE句が指定されている場合のみです。この句はマテリアライズド・ビューを最初に作成するときに指定するか、後でALTER MATERIALIZED VIEW文を使用して指定します。

前述のセッション・パラメータは、すべてのセッションについて、ALTER SYSTEM SET文を使用して設定でき、また初期化ファイルでも設定できます。所定のユーザー・セッションでALTER SESSIONを使用すると、そのセッションのみでクエリー・リライトの使用を禁止または可能にできます。次に例を示します。

```
ALTER SESSION SET QUERY_REWRITE_ENABLED = TRUE;
```

クエリー・リライトのレベルはセッションごとに設定できるので、各ユーザーが異なる整合性レベルで作業できます。次のような文を使用できます。

```
ALTER SESSION SET QUERY_REWRITE_INTEGRITY = STALE_TOLERATED;
ALTER SESSION SET QUERY_REWRITE_INTEGRITY = TRUSTED;
ALTER SESSION SET QUERY_REWRITE_INTEGRITY = ENFORCED;
```

11.2.4 クエリー・リライトの精度について

クエリー・リライトは、初期化パラメータQUERY_REWRITE_INTEGRITYによって制御される3つのレベルのリライト整合性を提供します。

QUERY_REWRITE_INTEGRITYパラメータに対して設定できる値は、次のとおりです。

- ENFORCED

  これがデフォルト・モードです。オプティマイザでは、マテリアライズド・ビューの最新データのみが使用され、ENABLED VALIDATEDになっている主/一意/外部キー制約に基づいた関係のみが使用されます。

- TRUSTED

  TRUSTEDモードの場合、オプティマイザでは、ディメンションで宣言された関係およびRELY制約が適切であることが信頼の対象になります。このモードでは、事前作成マテリアライズド・ビューや、ビューに基づくマテリアライズド・ビューが使用され、施行された関係と同様に施行されていない関係も使用されます。また、宣言されたがENABLED VALIDATEDでない主/一意キー制約、およびディメンションを使用して指定されたデータ関係も信頼されます。このモードではより高度なクエリー・リライト機能を使用できますが、ユーザーが宣言し、信頼された関係に不正確なものがあった場合、不正確な結果が生成される危険性もあります。

- STALE_TOLERATED

  STALE_TOLERATEDモードの場合、オプティマイザでは最新データを含むマテリアライズド・ビューの他に、有効だが失効データを含むマテリアライズド・ビューも使用されます。このモードでは、リライト機能を最大限に使用できますが、不正確な結果が生成される危険性もあります。

リライト整合性が最も安全なレベルであるENFORCEDに設定されている場合、オプティマイザでは、問合せの結果がディテール表に直接アクセスした場合と同じであることを保証するために、施行された主キー制約および参照整合性制約のみが使用されます。

リライト整合性をENFORCED以外のレベルに設定すると、次のような状況において、リライトした場合とリライトしなかった場合の出力が異なることがあります。

- マテリアライズド・ビューが、データのマスター・コピーと同期されていない場合。これは、通常、マテリアライズド・ビューの1つ以上のディテール表に対するバルク・ロードまたはDML操作の後に、マテリアライズド・ビューのリフレッシュ・プロシージャが保留状態にあるために発生します。データ・ウェアハウス・サイトによっては、この状況が最適な場合もあります。これは、一部のマテリアライズド・ビューでは一定の間隔でリフレッシュされることが一般的であるためです。
- ディメンション・オブジェクトに含まれる関係が無効の場合。たとえば、階層内のあるレベルの値が、正確に1つの親の値にロールアップされないことがあります。
- 事前作成マテリアライズド・ビュー表に格納された値が不適切な場合。
- 施行されていない表またはビューの制約により不正なデータ関係が定義されているため、間違った答えが生じている場合。

QUERY_REWRITE_INTEGRITYは、初期化パラメータ・ファイルで設定するか、ALTER SYSTEM文またはALTER SESSION文を使用して設定できます。

11.2.5 クエリー・リライトの有効化の権限について

マテリアライズド・ビューの使用は、そのマテリアライズド・ビューに対してユーザーが保持する権限ではなく、問合せ内のディテール表またはビューに対してユーザーが保持する権限に基づきます。

GRANT QUERY REWRITEシステム権限では、自スキーマ内のマテリアライズド・ビューから直接参照される表がすべて自スキーマ内にある場合にのみ、そのマテリアライズド・ビューに対するクエリー・リライトが有効になります。GRANT GLOBAL QUERY REWRITE権限では、マテリアライズド・ビューが別のスキーマ内のオブジェクトを参照する場合にも、マテリアライズド・ビューに対するクエリー・リライトが有効になります。また、自スキーマの外部にある表およびビューの場合、QUERY REWRITEオブジェクト権限を使用することもできます。

クエリー・リライト用にマテリアライズド・ビューを使用するための権限は、定義者権限のプロシージャに対する権限と似ています。

11.2.6 サンプル・スキーマおよびマテリアライズド・ビュー

次の項では、shサンプル・スキーマおよびいくつかのマテリアライズド・ビューを使用して、オプティマイザでデータ関係を利用してクエリー・リライトが行われる仕組みを説明します。

この章におけるクエリー・リライトの例では、主として次に示すマテリアライズド・ビューを参照します。これらのマテリアライズド・ビューは、必ずしもshスキーマの最も効率的な実装を表しているわけではありません。リライト機能を表すためのベースにすぎません。この章には、特定の機能についての例も記載されています。

次に示すのは、結合および集計を含むマテリアライズド・ビューです。

```
CREATE MATERIALIZED VIEW sum_sales_pscat_week_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_subcategory, t.week_ending_day,
       SUM(s.amount_sold) AS sum_amount_sold
FROM   sales s, products p, times t
WHERE  s.time_id=t.time_id  AND s.prod_id=p.prod_id
GROUP BY p.prod_subcategory, t.week_ending_day;

CREATE MATERIALIZED VIEW sum_sales_prod_week_mv
```

```
ENABLE QUERY REWRITE AS
SELECT p.prod_id, t.week_ending_day, s.cust_id,
       SUM(s.amount_sold) AS sum_amount_sold
FROM   sales s, products p, times t
WHERE  s.time_id=t.time_id AND s.prod_id=p.prod_id
GROUP BY p.prod_id, t.week_ending_day, s.cust_id;

CREATE MATERIALIZED VIEW sum_sales_pscat_month_city_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_subcategory, t.calendar_month_desc, c.cust_city,
       SUM(s.amount_sold) AS sum_amount_sold,
       COUNT(s.amount_sold) AS count_amount_sold
FROM   sales s, products p, times t, customers c
WHERE  s.time_id=t.time_id AND s.prod_id=p.prod_id AND s.cust_id=c.cust_id
GROUP BY p.prod_subcategory, t.calendar_month_desc, c.cust_city;
```

次に示すのは、結合のみを含むマテリアライズド・ビューです。

```
CREATE MATERIALIZED VIEW join_sales_time_product_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_id, p.prod_name, t.time_id, t.week_ending_day,
       s.channel_id, s.promo_id, s.cust_id, s.amount_sold
FROM   sales s, products p, times t
WHERE  s.time_id=t.time_id AND s.prod_id = p.prod_id;

CREATE MATERIALIZED VIEW join_sales_time_product_oj_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_id, p.prod_name, t.time_id, t.week_ending_day,
       s.channel_id, s.promo_id, s.cust_id, s.amount_sold
FROM   sales s, products p, times t
WHERE  s.time_id=t.time_id AND s.prod_id=p.prod_id(+);
```

これは必須ではありませんが、できるだけマテリアライズド・ビューに関する統計情報を収集して、問合せをリライトするかどうかをオプティマイザで判断できるようにすることをお薦めします。収集する場合、オブジェクトごとに収集するか、統計情報のない新規に作成したオブジェクトすべてを対象として収集するかを選択できます。オブジェクトごとの場合を次のjoin_sales_time_product_mvの例で示します。

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS ( -
   'SH','JOIN_SALES_TIME_PRODUCT_MV', estimate_percent   => 20, -
    block_sample   => TRUE, cascade   => TRUE);
```

次に、統計情報のない新規作成したオブジェクトすべてを収集対象にする場合の例を示します。

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS ( 'SH', -
   options            => 'GATHER EMPTY', -
   estimate_percent   =>  20, block_sample  =>  TRUE, -
   cascade            =>  TRUE);
```

11.2.7 クエリー・リライトの発生を確認する方法

クエリー・リライトは透過的に行われるので、問合せがリライトされたかどうかを確認するには、特別なステップを実行する必要があります。問合せが高速に実行された場合、リライトが発生したと考えられますが、これは確認にはなりません。そのため、EXPLAIN PLAN文またはDBMS_MVIEW.EXPLAIN_REWRITEプロシージャを使用して、クエリー・リライトが発生したことを確認します。詳細は、クエリー・リライトが発生したことの確認を参照してください。

11.3 クエリー・リライトの例

このような例からも、マテリアライズド・ビューを使用したクエリー・リライトの効果は明らかです。

次のcal_month_sales_mvというマテリアライズド・ビューの場合を考えてみます。このマテリアライズド・ビューを使用すると、月ごとの販売額(ドル)の合計を表示できます。

```
CREATE MATERIALIZED VIEW cal_month_sales_mv
ENABLE QUERY REWRITE AS
SELECT t.calendar_month_desc, SUM(s.amount_sold) AS dollars
FROM sales s, times t WHERE s.time_id = t.time_id
GROUP BY t.calendar_month_desc;
```

たとえば、その店舗における販売数量が通常の月では100万前後だとします。この場合、このマテリアライズド集計ビューには、事前に計算された月ごとの合計販売額(ドル)が用意されます。

次の問合せについて考えてみます。この問合せは、その店舗における会計月ごとの総販売数量を問い合せるためのものです。

```
SELECT t.calendar_month_desc, SUM(s.amount_sold)
FROM sales s, times t WHERE s.time_id = t.time_id
GROUP BY t.calendar_month_desc;
```

前述のマテリアライズド・ビューおよびクエリー・リライト機能がない場合、Oracle Databaseがsales表に直接アクセスし、総販売数量を計算した上でその結果を返します。その場合、sales表から膨大な数の行が読み込まれ、対象となるディスクへのアクセスに伴って問合せの応答時間は必ず増大します。また、問合せに結合があるので、膨大な数の行に対する結合の計算が必要になり、問合せへの応答はさらに遅くなります。

ここで、マテリアライズド・ビューcal_month_sales_mvがあると、クエリー・リライトによって前述の問合せが透過的にリライトされ、次の問合せに書き換えられます。

```
SELECT calendar_month, dollars
FROM cal_month_sales_mv;
```

マテリアライズド・ビューcal_month_sales_mvの行数はほんの数十行で、結合も存在しないため、Oracle Databaseにより結果は即座に戻されます。

12 マテリアライズド・ビューのための高度なクエリー・リライト

この章では、Oracleの高度なクエリー・リライトについて説明します。次の内容が含まれます。

- Oracleによる問合せのリライト方法
- クエリー・リライトのタイプ
- その他のクエリー・リライトの考慮事項
- 同等化を使用した高度なクエリー・リライト
- 同等化を使用した結果キャッシュ・マテリアライズド・ビューの作成
- 近似問合せに基づいたクエリー・リライトおよびマテリアライズド・ビュー
- クエリー・リライトが発生したことの確認
- クエリー・リライトを改善するための設計上の考慮事項

12.1 Oracleによる問合せのリライト方法

オプティマイザが問合せをリライトするには、いくつかの方法があります。クエリー・リライトが可能かどうかを判断する最初のステップは、問合せが次の前提条件を満たしているかどうかを確認することです。

- マテリアライズド・ビュー内の結合がSQLで使用されている。
- 問合せに回答するのに十分なデータがマテリアライズド・ビューまたは複数のビューにある。

次に、オプティマイザは問合せをリライトする方法を判断する必要があります。最も簡単な例は、マテリアライズド・ビューに格納されている結果が、問合せによって要求されているものと正確に一致する場合です。オプティマイザは、問合せのテキストとマテリアライズド・ビュー定義のテキストを比較して、このような判断を行います。このテキスト一致の方法は最も簡単ですが、このタイプのクエリー・リライトに使用できる問合せの数は多くありません。

テキスト比較によるテストで判断できなかった場合、オプティマイザは、結合、選択、グルーピング、集計およびフェッチされた列データに基づいて、一連の一般的なチェックを実行します。これは、問合せの様々な句(SELECT、FROM、WHERE、HAVINGまたはGROUP BY)をマテリアライズド・ビューのものと比較することによって行われます。

クエリー・リライト方法1: テキスト一致リライトまたは一般的なクエリー・リライト方法のいずれかのタイプのクエリー・リライトを使用できます。

次のトピックでは、オプティマイザについてさらに詳しく説明します。

- コストベース・オプティマイザとクエリー・リライトについて
- 一般的なクエリー・リライト方法
- クエリー・リライトで行われるチェックについて
- ディメンションを使用したクエリー・リライトについて

12.1.1 コストベースの最適化およびクエリー・リライトについて

問合せがリライトされると、Oracleのコストベース・オプティマイザによって、リライトされた問合せのコストと元の問合せのコストが比較され、コストの低い方の実行計画が選択されます。

クエリー・リライトは、コストベース・オプティマイザとともに使用できます。Oracle Databaseは、リライトを使用または使用せずに

入力問合せを最適化し、最も効率的な方法を選択します。オプティマイザは、1つ以上の問合せブロックを一度に1つずつリライトすることで、問合せをリライトします。

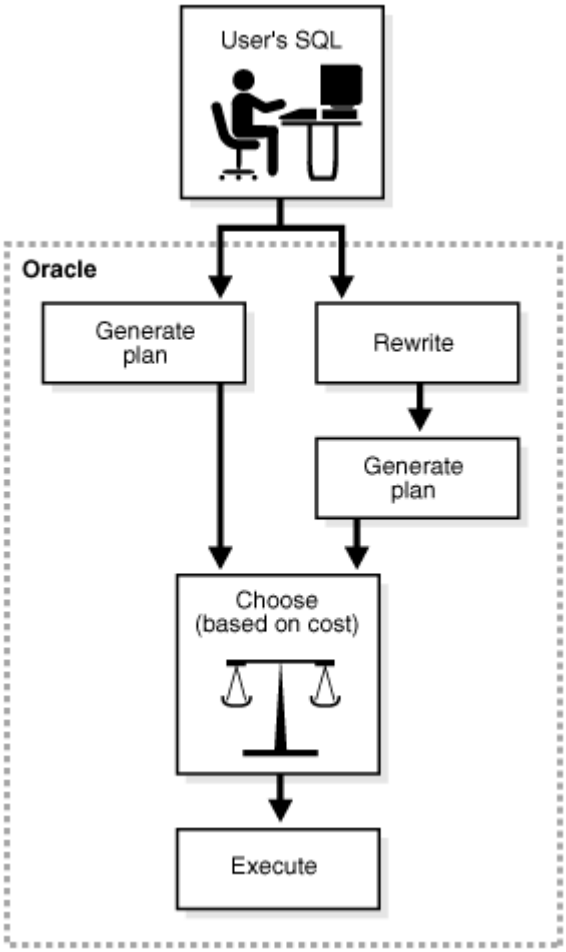
クエリー・リライトが複数のマテリアライズド・ビューの中から問合せブロックをリライトするものを選択できる場合、読み込まれるデータ量の最も少ないものが選択されます。リライト用のマテリアライズド・ビューが選択されると、オプティマイザはリライトされた問合せがさらに別のマテリアライズド・ビューでリライト可能かどうかをテストします。このプロセスは、リライトができなくなるまで繰り返されます。その後、リライトされた問合せは最適化され、元の問合せも最適化されます。オプティマイザでは、この2つの最適化したものが比較され、コストの低い方が選択されます。

最適化はコストを基準にするため、問合せに関係する表と、マテリアライズド・ビューを示す表の両方についての統計情報を収集することが重要です。表の行数などの統計情報は、リライトされた問合せのコスト計算に使用する基本的な尺度です。その作成にはDBMS_STATSパッケージを使用します。

インライン・ビューまたは名前付きビューを含む問合せも、クエリー・リライトの対象になります。問合せに名前付きビューが含まれている場合、マテリアライズド・ビューと問合せを一致させるために、ビュー名が使用されます。問合せにインライン・ビューが含まれている場合、インライン・ビューは、マテリアライズド・ビューと問合せを一致させる前に、マージされる場合があります。

図12-1に、リライト処理中に使用されるコストベースのアプローチを図示します。

図12-1 クエリー・リライト・プロセス



12.1.2 一般的なクエリー・リライト方法

オプティマイザでは、様々なクエリー・リライト方法の中から選択して、問合せに対する回答を行います。テキストの一致によるリライトが不可能な場合、一般的なクエリー・リライトと呼ばれるリライト方法が使用されます。この高度なテクニックを使用するメリットは、多数の異なる問合せに対する回答に1つ以上のマテリアライズド・ビューを使用できる点です。したがって、クエリー・リライトを発生させるために、問合せとマテリアライズド・ビューが常に一致している必要はありません。

一般的なクエリー・リライト方法を使用する場合、オプティマイザでは、主キー制約、外部キー制約、ディメンション・オブジェクト

など、依存可能なデータ関係を使用します。たとえば、主キーと外部キーの関係によって、外部キー表の各行が主キー表の1つ以下の行と結合することがオプティマイザに示されます。さらに、外部キーにNOT NULL制約がある場合は、外部キー表の各行が主キー表の1つの行と正確に結合することが示されます。ディメンション・オブジェクトは、日、月、年などの関係を表し、これは、データを日レベルから月レベルにロールアップするために使用できます。

データの結合、グルーピングまたは集計操作によって生成される結果の種類が示されるため、このデータ関係は特に重要です。そのため、このようなデータ関係がデータベースに存在する場合、大規模な問合せ集合のリライトを可能にするには、制約およびディメンションを宣言する必要があります。

関連項目:

クエリー・リライトに制約とディメンションが必要な場合

12.1.2.1 クエリー・リライトで制約およびディメンションが必要となる場合

表12-1に、異なるタイプのクエリー・リライトにディメンションおよび制約が必要な場合を示します。クエリー・リライトのこれらのタイプについては、この章の中で説明しています。

表12-1 クエリー・リライトのディメンションおよび制約要件

クエリー・リライトのタイプ	ディメンション	主キー/外部キー/Not Null制約
SQL テキスト一致	必要なし	必要なし
後戻り結合	必要あり または	必要あり
集計可能性	必要なし	必要なし
集計ロールアップ	必要なし	必要なし
ディメンションを使用したロールアップ	必須	必要なし
データのフィルタリング	必要なし	必要なし
PCT リライト	必要なし	必要なし
複数のマテリアライズド・ビュー	必要なし	必要なし

12.1.3 クエリー・リライトで行われるチェックについて

クエリー・リライトが発生するには、データが様々なチェックをパスする必要があります。これらのチェックには、次のものがあります。

- クエリー・リライトでの結合互換性チェック
- クエリー・リライトでのデータ充足性チェック
- クエリー・リライトでのグルーピング互換性チェック

- クエリー・リライトでの集計可能性チェック

12.1.3.1 クエリー・リライトの結合互換性チェック

このチェックでは、問合せの結合がマテリアライズド・ビューの結合と比較されます。一般に、この比較によって、結合は次の3つに分類されます。

- 問合せおよびマテリアライズド・ビューの両方に発生する共通結合。この結合は、共通のサブグラフを形成します。

  「共通結合」を参照してください。

- 問合せのみで発生し、マテリアライズド・ビューでは発生しないデルタ結合。この結合は、問合せのデルタ・サブグラフを形成します。

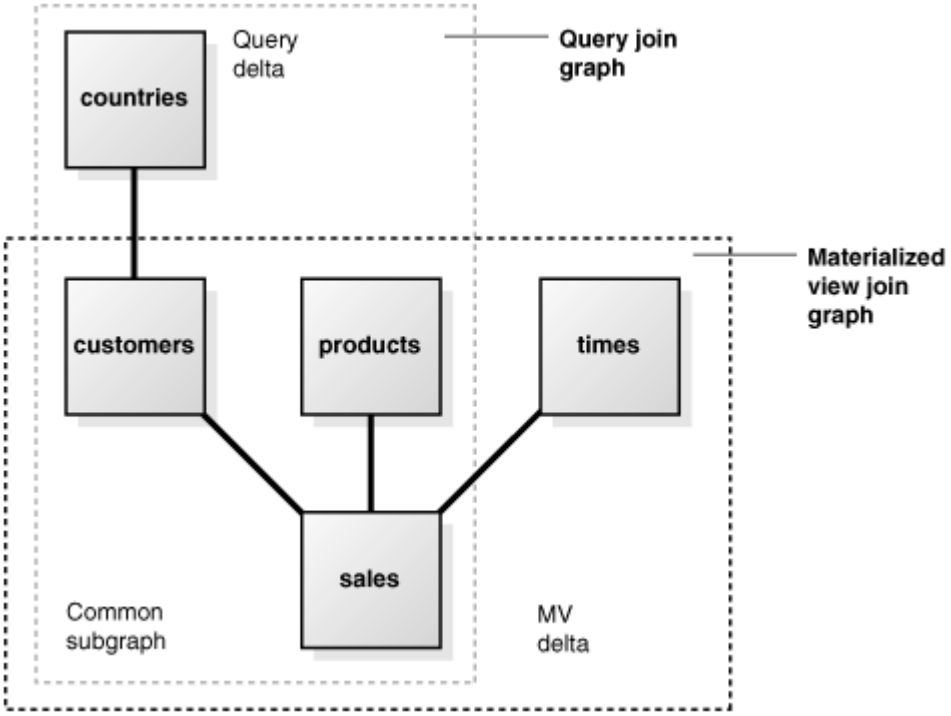
  問合せデルタ結合を参照してください。

- マテリアライズド・ビューのみで発生し、問合せでは発生しないデルタ結合。この結合は、マテリアライズド・ビューのデルタ・サブグラフを形成します。

  マテリアライズド・ビュー・デルタ結合を参照してください。

図12-2に、これらの結合を示します。

図12-2 クエリー・リライトのサブグラフ



関連項目:

クエリー・リライトで行われるチェックについて

12.1.3.1.1 共通結合

両者間の共通結合の組は同型であるか、または問合せの結合がマテリアライズド・ビューの結合から導出可能である必要があります。たとえば、マテリアライズド・ビューが表Aの表Bとの外部結合を含み、かつ、問合せが表Aの表Bとの内部結合を含んでいる場合、内部結合の結果は、外部結合の結果からアンチ結合行をフィルタすることで導出できます。たとえば、次の問合せを考えてみます。

```
SELECT p.prod_name, t.week_ending_day, SUM(s.amount_sold)
FROM   sales s, products p, times t
WHERE  s.time_id=t.time_id AND s.prod_id = p.prod_id
AND mv.week_ending_day BETWEEN TO_DATE('01-AUG-1999', 'DD-MON-YYYY')
                           AND TO_DATE('10-AUG-1999', 'DD-MON-YYYY')
GROUP BY p.prod_name, mv.week_ending_day;
```

次に、この問合せとマテリアライズド・ビューjoin_sales_time_product_mvとの間の共通結合を示します。

```
s.time_id = t.time_id AND s.prod_id = p.prod_id
```

これらは正確に一致し、次のようにリライトできます。

```
SELECT p.prod_name, mv.week_ending_day, SUM(s.amount_sold)
FROM   join_sales_time_product_mv
WHERE  mv.week_ending_day BETWEEN TO_DATE('01-AUG-1999','DD-MON-YYYY')
                          AND   TO_DATE('10-AUG-1999','DD-MON-YYYY')
GROUP BY mv.prod_name, mv.week_ending_day;
```

問合せは、マテリアライズド・ビューjoin_sales_time_product_oj_mvを使用しても結果を得ることができます。この場合、問合せの内部結合がマテリアライズド・ビューの外部結合から導出可能です。リライトされた問合せでは、(ユーザーには透過的に)アンチ結合行がフィルタによって排除されます。リライトされた問合せの構造は次のとおりです。

```
SELECT mv.prod_name, mv.week_ending_day, SUM(mv.amount_sold)
FROM   join_sales_time_product_oj_mv mv
WHERE  mv.week_ending_day BETWEEN TO_DATE('01-AUG-1999','DD-MON-YYYY')
  AND  TO_DATE('10-AUG-1999','DD-MON-YYYY') AND mv.prod_id IS NOT NULL
GROUP BY mv.prod_name, mv.week_ending_day;
```

一般に、結合のみを含むマテリアライズド・ビューに外部結合を使用する場合は、マテリアライズド・ビューの外部結合の右側に主キーまたはROWIDを指定する必要があります。たとえば、前述の例のjoin_sales_time_product_oj_mvでは、salesとproductsの両方に主キーがあります。

結合のみを含むマテリアライズド・ビューの別の例に、セミ結合リライトの例があります。つまり、単一表を持つEXISTSまたはIN副問合せが含まれる問合せです。次のような、売上が1,000ドルを超えた製品をレポートする問合せを考えてみます。

```
SELECT DISTINCT p.prod_name
FROM products p
WHERE EXISTS (SELECT p.prod_id, SUM(s.amount_sold) FROM sales s
              WHERE p.prod_id=s.prod_id HAVING SUM(s.amount_sold) > 1000)
              GROUP BY p.prod_id);
```

この問合せは、次のように表すこともできます。

```
SELECT DISTINCT p.prod_name
FROM products p WHERE p.prod_id IN (SELECT s.prod_id FROM sales s
                    WHERE s.amount_sold > 1000);
```

この問合せには、products表とsales表の間のセミ結合(s.prod_id = p.prod_id)が含まれています。

この問合せは、外部キー制約がアクティブの場合、join_sales_time_product_mvマテリアライズド・ビューを使用するか、主キーがアクティブの場合はjoin_sales_time_product_oj_mvマテリアライズド・ビューを使用してリライトできます。どちらのマテリアライズド・ビューも、s.prod_id=p.prod_idが含まれます。これは、問合せ内のセミ結合を導出する場合に使用します。この問合せは次のように、join_sales_time_product_mvを使用してリライトされます。

```
SELECT mv.prod_name
FROM (SELECT DISTINCT mv.prod_name FROM  join_sales_time_product_mv mv
```

```
      WHERE mv.amount_sold > 1000);
```

マテリアライズド・ビューjoin_sales_time_product_mvがtime_idでパーティション化された場合、salesとproductsの間の元の結合が回避されたため、この問合せは元の問合せより効率的になる傾向があります。この問合せは次のように、join_sales_time_product_oj_mvを使用してリライトされる場合もあります。

```
SELECT mv.prod_name
FROM (SELECT DISTINCT mv.prod_name FROM join_sales_time_product_oj_mv mv
      WHERE mv.amount_sold > 1000 AND mv.prod_id IS NOT NULL);
```

セミ結合を使用したリライトは、結合のみを含むマテリアライズド・ビューに制限されており、結合および集計を含むマテリアライズド・ビューの場合には使用できません。

関連項目:

クエリー・リライトで行われるチェックについて

12.1.3.1.2 問合せデルタ結合

問合せデルタ結合は、問合せでのみ使用される結合で、マテリアライズド・ビューでは使用されません。問合せに指定できるデルタ結合の数や種類に制限はなく、問合せがマテリアライズド・ビューを使用してリライトされた場合は、簡単に保持されます。保持された結合が正常に機能するには、マテリアライズド・ビューに結合キーが含まれている必要があります。リライトの際、マテリアライズド・ビューは問合せデルタ内の適切な表に結合されます。たとえば、次の問合せを考えてみます。

```
SELECT p.prod_name, t.week_ending_day, c.cust_city, SUM(s.amount_sold)
FROM   sales s, products p, times t, customers c
WHERE  s.time_id=t.time_id AND s.prod_id = p.prod_id
AND    s.cust_id = c.cust_id
GROUP BY p.prod_name, t.week_ending_day, c.cust_city;
```

マテリアライズド・ビューjoin_sales_time_product_mvを使用した場合、共通結合はs.time_id=t.time_idおよびs.prod_id=p.prod_idになります。問合せのデルタ結合はs.cust_id=c.cust_idです。リライトされたフォームは、次のようにjoin_sales_time_product_mvマテリアライズド・ビューをcustomers表に結合します。

```
SELECT mv.prod_name, mv.week_ending_day, c.cust_city, SUM(mv.amount_sold)
FROM   join_sales_time_product_mv mv, customers c
WHERE  mv.cust_id = c.cust_id
GROUP BY mv.prod_name, mv.week_ending_day, c.cust_city;
```

関連項目:

クエリー・リライトで行われるチェックについて

12.1.3.1.3 マテリアライズド・ビュー・デルタ結合

マテリアライズド・ビューのデルタ結合は、マテリアライズド・ビューのみで使用される結合で、問合せでは使用されません。マテリアライズド・ビュー内のすべてのデルタ結合は、共通結合の結果に関して可逆式である必要があります。可逆式結合は、共通結合の結果が制限されないことを保証します。**可逆式**結合では、表Aと表Bが結合された場合、表Aの行は表Bの行と常に一致し、どのデータも消失しません。このため、可逆式結合と呼ばれます。たとえば、外部キーを使用する各行は、外部キーにNULLが許可されない場合、主キーを使用した1つの行と一致します。そのため、可逆式結合を保証するには、適切な結合

キーに外部キー制約、主キー制約およびNOT NULL制約を指定する必要があります。または、表Aと表Bの結合が外部結合の場合(Aが外部表の場合)、結合は表Aのすべての行を保持するため可逆式となります。

マテリアライズド・ビュー内のすべてのデルタ結合は、共通結合の結果に関して重複していない必要があります。非重複結合は、共通結合の結果が重複されないことを保証します。たとえば、非重複結合では、表Aと表Bが結合された場合、表Aの行は表Bの1つ以下の行と一致し、重複は発生しません。非重複結合を保証するには、主キー制約または一意キー制約を使用して、表Bのキーを一意の値に制約する必要があります。

salesとtimesを結合する次の問合せを考えてみます。

```
SELECT t.week_ending_day, SUM(s.amount_sold)
FROM   sales s, times t
WHERE s.time_id = t.time_id AND t.week_ending_day BETWEEN TO_DATE
 ('01-AUG-1999', 'DD-MON-YYYY') AND TO_DATE('10-AUG-1999', 'DD-MON-YYYY')
GROUP BY week_ending_day;
```

マテリアライズド・ビューjoin_sales_time_product_mvは、salesとproductsの間の結合(s.prod_id=p.prod_id)が追加されています。これは、join_sales_time_product_mvのデルタ結合です。この結合が可逆式および非重複である場合は、問合せをリライトできます。s.prod_idがp.prod_idに対する外部キーで、かつ、NULLではない場合がその例です。したがって、問合せは次のようにリライトされます。

```
SELECT week_ending_day, SUM(amount_sold)
FROM   join_sales_time_product_mv
WHERE  week_ending_day BETWEEN TO_DATE('01-AUG-1999', 'DD-MON-YYYY')
                       AND     TO_DATE('10-AUG-1999', 'DD-MON-YYYY')
GROUP BY week_ending_day;
```

問合せは、外部キー制約が必要とされないマテリアライズド・ビューjoin_sales_time_product_mv_ojを使用してリライトすることもできます。このビューには、salesとproductsの間の外部結合(s.prod_id=p.prod_id(+))が含まれています。このため、この結合は可逆式になります。p.prod_idが主キーの場合、非重複条件も満たされ、オプティマイザは問合せを次のようにリライトします。

```
SELECT week_ending_day, SUM(amount_sold)
FROM   join_sales_time_product_oj_mv
WHERE  week_ending_day BETWEEN TO_DATE('01-AUG-1999', 'DD-MON-YYYY')
  AND  TO_DATE('10-AUG-1999', 'DD-MON-YYYY')
GROUP BY week_ending_day;
```

問合せは、外部キー制約が必要とされないマテリアライズド・ビューjoin_sales_time_product_mv_ojを使用してリライトすることもできます。このビューには、salesとproductsの間の外部結合(s.prod_id=p.prod_id(+))が含まれています。このため、この結合は可逆式になります。p.prod_idが主キーの場合、非重複条件も満たされ、オプティマイザは問合せを次のようにリライトします。

```
SELECT week_ending_day, SUM(amount_sold)
FROM   join_sales_time_product_oj_mv
WHERE  week_ending_day BETWEEN TO_DATE('01-AUG-1999', 'DD-MON-YYYY')
  AND  TO_DATE('10-AUG-1999', 'DD-MON-YYYY')
GROUP BY week_ending_day;
```

shスキーマでは、salesとproductsの間の主キー/外部キーの関係がすでに可逆式になっているため、join_sales_time_product_mv_ojの定義には外部結合が不要であることに注意してください。ただしこれは、単に具体例を示すためのものです。sales.prod_idがNULL値可能で、したがって結合条件sales.prod_id = products.prod_idに可逆性がなくなる場合、外部結合は必要となります。

現在の制限事項では、外部結合を含むリライトのほとんどは、結合のみを使用したマテリアライズド・ビューに制限されます。外

部結合を含むマテリアライズド集計ビューを使用したリライトのサポートは制限されているため、この種のマテリアライズド・ビューは、マテリアライズド・ビューのデルタ結合の可逆性を保証するために、外部キー制約に依存する必要があります。

関連項目:

クエリー・リライトで行われるチェックについて

12.1.3.1.4 結合の等価性の認識

クエリー・リライトは、等価な結合であるとの認識に基づいて多くの変換を行うことができます。クエリー・リライトでは、次の構成体は結合に対して等価であると認識されます。

```
WHERE table1.column1 = F(args)    /* sub-expression A */
AND table2.column2 = F(args)      /* sub-expression B */
```

F(args)がDETERMINISTICとして宣言されているPL/SQLファンクションで、Fを起動するときの引数がいずれも同じである場合、副次式Aと副次式Bの組合せは、table1.column1とtable2.column2の結合として認識できます。つまり、次の式は、前述の式と等価になります。

```
WHERE table1.column1 = F(args)          /* sub-expression A */
AND table2.column2 = F(args)            /* sub-expression B */
AND table1.column1 = table2.column2     /* join-expression J */
```

結合式Jは副次式Aと副次式Bから推論できるため、推論された結合を使用して、マテリアライズド・ビュー内の対応する結合table1.column1 = table2.column2に一致させることができます。

12.1.3.2 クエリー・リライトのデータ充足性チェック

このチェックでは、オプティマイザは、問合せが要求した列データが、マテリアライズド・ビューから取得可能かどうかを判断します。このために、1つの列と別の列との同等化が使用されます。たとえば、表Aと表Bの間の内部結合が結合述部A.X = B.Xに基づく場合、結合の結果、列A.Xのデータは列B.Xのデータと同等になります。このデータ・プロパティが、問合せ内の列A.Xとマテリアライズド・ビュー内の列B.Xの一致、またはその逆に使用されます。たとえば、次の問合せを考えてみます。

```
SELECT p.prod_name, s.time_id, t.week_ending_day, SUM(s.amount_sold)
FROM sales s, products p, times t
WHERE s.time_id=t.time_id AND s.prod_id = p.prod_id
GROUP BY p.prod_name, s.time_id, t.week_ending_day;
```

この問合せは、マテリアライズド・ビューにs.time_idが含まれない場合でも、join_sales_time_product_mvを使用して回答されます。かわりにこの問合せには、結合条件s.time_id=t.time_idを介してs.time_idと同等のt.time_idが含まれます。したがって、オプティマイザで次のリライトが選択される場合があります。

```
SELECT prod_name, time_id, week_ending_day, SUM(amount_sold)
FROM join_sales_time_product_mv
GROUP BY prod_name, time_id, week_ending_day;
```

12.1.3.3 クエリー・リライトのグルーピング互換性チェック

このチェックは、マテリアライズド・ビューと問合せの両方にGROUP BY句がある場合にのみ必要です。オプティマイザは、まず、問合せが要求したデータのグルーピングが、マテリアライズド・ビューに格納されているデータのグルーピングと同じかどうかを判断します。つまり、グルーピングのレベルは、問合せとマテリアライズド・ビューで同じです。マテリアライズド・ビューで問合せの列および式をすべてグルーピングし、追加の列または式もグルーピングする場合、クエリー・リライトでは、問合せのグルーピング列およびグルーピン

グ式でマテリアライズド・ビューを再集計して、問合せで要求された同じ結果を導出できます。

12.1.3.4 クエリー・リライトの集計可能性チェック

このチェックは、問合せおよびマテリアライズド・ビューの両方に集計が含まれている場合にのみ必要になります。このチェックでは、オプティマイザは、問合せが要求した集計が、マテリアライズド・ビューに格納された1つ以上の集計から導出または計算可能かどうかを判断します。たとえば、問合せがAVG(X)を要求し、マテリアライズド・ビューがSUM(X)およびCOUNT(X)を含む場合、AVG(X)はSUM(X)/COUNT(X)で計算できます。

グルーピング互換性チェックによって、マテリアライズド・ビューに格納された集計のロールアップが必要であると判断された場合、次に、集計可能性チェックによって、問合せが要求した各集計が、マテリアライズド・ビューの集計を使用してロールアップできるかどうかが判断されます。

12.1.4 ディメンションを使用したクエリー・リライトについて

この項では、リライト環境でディメンションを使用する際の次の側面について説明します。

- クエリー・リライト環境でディメンションを使用する利点
- クエリー・リライトでディメンションを定義する方法

12.1.4.1 クエリー・リライト環境でディメンションを使用する利点

ディメンションは、列同士の階層(親子)関係を定義します。これらの列は、同じ表の列である必要はありません。

ディメンションを定義すると、列間の機能依存性を確立できるため、クエリー・リライトの実行性が高まります。また、ディメンションにより、制約では表現できない表内の関係を表現することもできます。ディメンション定義に追加の領域は必要ありません。むしろ、ディメンション定義を行うと、使用しているスキーマ内のディメンション内およびディメンション間の関係を記述するメタデータが構築されます。マテリアライズド・ビューを作成する前に行う最初のステップは、スキーマを調べてディメンションを定義することです。これによって問合せのリライトの機会が大幅に増す場合があるためです。

12.1.4.2 クエリー・リライト用にディメンションを定義する方法

任意のスキーマに対して、次のステップを使用してディメンションを作成してください。

1. スキーマ内のすべてのディメンションおよびディメンション表を指定する
2. 各ディメンション内の階層を特定する
3. 階層の各レベル内の属性依存性を特定する
4. データ・ウェアハウスのファクト表から各ディメンションへの結合を特定する

ディメンションで宣言された関係をクエリー・リライトで利用するには、パラメータQUERY_REWRITE_INTEGRITYをTRUSTEDまたはSTALE_TOLERATEDに設定する必要があります。

スキーマ内のすべてのディメンションおよびディメンション表を指定する

複数の表に収まるようにディメンションが正規化されている場合、ディメンション表間の結合において、子表の各行が親表の1つの行のみと結合していることが保証されていることを確認します。また非正規化ディメンションの場合、子の列が親(または属性)列を一意に決定できなくてはなりません。これらの規則を順守できなかった場合、問合せから不正確な結果が戻される場合があります。

各ディメンション内の階層を特定する

たとえば、日(day)は月(month)の子(日レベルのデータを月レベルまで集計可能)であり、四半期(quarter)は年(year)の子です。

階層の各レベル内の属性依存性を特定する

たとえば、calendar_month_nameが月の属性であることを指定します。

データ・ウェアハウスのファクト表から各ディメンションへの結合を特定する

次に、各結合において、ファクト表の各行がディメンションの1つの行のみと結合することが保証されることを確認します。この条件を宣言し、オプションで規定する必要があります。それには、ファクト表のキー列に外部キー制約およびNOT NULL制約を追加し、親表の結合キーに主キー制約を追加します。これらの関係が、プロシージャ(ロード・プロセスなど)を処理する他のデータによって保証される場合、これらの制約は、NOVALIDATEオプションを使用して有効化できます。このオプションを使用すると、表のすべての行がその制約に準拠することを検証するのに必要な時間を節約できます。妥当性チェックが行われないすべての制約について、クエリー・リライトで使用できるようにするためには、RELY句も必要になります。

12.1.4.2.1 時間ディメンションを作成するSQL文の例

```
CREATE DIMENSION times_dim
LEVEL day IS TIMES.TIME_ID
LEVEL month IS TIMES.CALENDAR_MONTH_DESC
LEVEL quarter IS TIMES.CALENDAR_QUARTER_DESC
LEVEL year IS TIMES.CALENDAR_YEAR
LEVEL fis_week IS TIMES.WEEK_ENDING_DAY
LEVEL fis_month  IS TIMES.FISCAL_MONTH_DESC
LEVEL fis_quarter IS TIMES.FISCAL_QUARTER_DESC
LEVEL fis_year IS TIMES.FISCAL_YEAR
        HIERARCHY cal_rollup
        (day  CHILD OF month CHILD OF quarter CHILD OF year)
        HIERARCHY fis_rollup
        (day  CHILD OF fis_week CHILD OF fis_month CHILD OF fis_quarter
        CHILD OF fis_year)

        ATTRIBUTE day DETERMINES
        (day_number_in_week, day_name, day_number_in_month,
         calendar_week_number)

        ATTRIBUTE month DETERMINES
        (calendar_month_desc, calendar_month_number, calendar_month_name,
         days_in_cal_month, end_of_cal_month)

        ATTRIBUTE quarter DETERMINES
        (calendar_quarter_desc, calendar_quarter_number,days_in_cal_quarter,
         end_of_cal_quarter)

        ATTRIBUTE year DETERMINES
        (calendar_year,  days_in_cal_year, end_of_cal_year)

        ATTRIBUTE fis_week DETERMINES
        (week_ending_day, fiscal_week_number);
```

12.2 クエリー・リライトのタイプ

多数の行または大規模な表同士の結合に対して計算が必要な集計を持つ問合せは、コストが高くなる可能性があり、結果を戻すまでに長い時間がかかる場合があります。クエリー・リライトは、事前計算済の結果が格納されているマテリアライズド・ビューを使用してそのような問合せを透過的にリライトするので、問合せに対して瞬時に結果を戻すことができます。これらのマテリアライズド・ビューは、大きく2つのグループに分類できます。すなわち、マテリアライズド集計ビューとマテリアライズド結合ビューです。マテリアライズド集計ビューは、元の表の列の事前に計算された集計値を格納する表です。同様に、マテリアライズド結合ビューは、元の表の列間の事前に計算された結合を格納する表です。クエリー・リライトは、入力問合せを変換して、マテリアライズド・ビューの列から結果をフェッチします。これらの列には事前に計算された結果がすでに格納されているので、入力問合せに対して

瞬時に結果を戻すことができます。キューブ・マテリアライズド・ビューのクエリー・リライトに関する考慮事柄については、『Oracle OLAPユーザーズ・ガイド』を参照してください。

この項では、問合せのリライトに使用できる次の方法について説明します。

- クエリー・リライト方法1: テキスト一致リライト
- クエリー・リライト方法2: 後戻り結合
- クエリー・リライト方法3: 集計可能性
- クエリー・リライト方法4: 集計ロールアップ
- クエリー・リライト方法5: ディメンションを使用したロールアップ
- クエリー・リライト方法6: マテリアライズド・ビューにデータのサブセットが1つのみ含まれる場合
- パーティション・チェンジ・トラッキング(PCT)リライト
- 複数のマテリアライズド・ビューを使用したクエリー・リライトについて

12.2.1 クエリー・リライト方法1: テキスト一致リライト

クエリー・リライト・エンジンは常に、問合せをリライトする際に対象となり得るすべてのマテリアライズド・ビューの定義のテキストと、入力問合せのテキストとの比較を最初に試みます。これは、一般的なリライトに必要な複雑な分析を行うコストと比較して、単純なテキスト比較を行うオーバーヘッドが通常わずかであるためです。

クエリー・リライト・エンジンは、テキストの完全一致のリライトおよびテキストの部分一致によるリライトという、2つのテキスト一致方法を使用します。テキストが完全に一致する場合は、問合せのテキスト全体がマテリアライズド・ビュー定義のテキスト全体(SELECT文全体)と比較されます。テキストの比較中は、空白は無視されます。たとえば、次のマテリアライズド・ビューsum_sales_pscat_month_city_mvがあるとします。

```
CREATE MATERIALIZED VIEW sum_sales_pscat_month_city_mv
ENABLE QUERY REWRITE AS
   SELECT p.prod_subcategory, t.calendar_month_desc, c.cust_city,
   SUM(s.amount_sold) AS sum_amount_sold,
   COUNT(s.amount_sold) AS count_amount_sold
   FROM sales s, products p, times t, customers c
   WHERE s.time_id=t.time_id
      AND     s.prod_id=p.prod_id
      AND     s.cust_id=c.cust_id
  GROUP BY p.prod_subcategory, t.calendar_month_desc, c.cust_city;
```

次の問合せについて考えてみます。

```
SELECT p.prod_subcategory, t.calendar_month_desc, c.cust_city,
       SUM(s.amount_sold) AS sum_amount_sold,
       COUNT(s.amount_sold) AS count_amount_sold
       FROM sales s, products p, times t, customers c
       WHERE s.time_id=t.time_id
         AND     s.prod_id=p.prod_id
         AND     s.cust_id=c.cust_id
       GROUP BY p.prod_subcategory, t.calendar_month_desc, c.cust_city;
```

この問合せは、sum_sales_pscat_month_city_mv(空白は除外)と一致し、次のようにリライトされます。

```
SELECT mv.prod_subcategory, mv.calendar_month_desc, mv.cust_city,
       mv.sum_amount_sold, mv.count_amount_sold
FROM   sum_sales_pscat_month_city_mv;
```

テキストの完全一致が失敗した場合、オプティマイザはテキストの部分一致を試みます。この方法では、問合せのFROM句から始まるテキストが、マテリアライズド・ビュー定義のFROM句から始まるテキストと比較されます。したがって、次の問合せはリライトできます。

```
SELECT p.prod_subcategory, t.calendar_month_desc, c.cust_city,
       AVG(s.amount_sold)
FROM   sales s, products p, times t, customers c
WHERE  s.time_id=t.time_id AND s.prod_id=p.prod_id
AND    s.cust_id=c.cust_id
GROUP BY p.prod_subcategory, t.calendar_month_desc, c.cust_city;
```

この問合せは、次のようにリライトされます。

```
SELECT mv.prod_subcategory, mv.calendar_month_desc, mv.cust_city,
       mv.sum_amount_sold/mv.count_amount_sold
FROM   sum_sales_pscat_month_city_mv mv;
```

テキストの部分一致によるリライト方法の場合、問合せが要求した売上集計の平均は、マテリアライズド・ビューに格納された売上の合計および売上集計の件数を使用して計算されます。

どちらのテキストの一致も成功しなかった場合、オプティマイザは、一般的なクエリー・リライト方法を使用します。

テキスト一致リライトでは、大文字と小文字の違いに意味があるコンテキストとそうでないコンテキストが区別されます。たとえば、次の2つの文は同等です。

```
SELECT X, 'aBc' FROM Y

Select x, 'aBc' From y
```

12.2.2 クエリー・リライト方法2: 後戻り結合

問合せが要求した列データがマテリアライズド・ビューから取得できない場合、オプティマイザは、さらに、機能依存性と呼ばれるデータ関係を基に取得できないかを判断します。列内のデータによって別の列のデータを決定できる場合、そのような関係は、機能依存性または機能決定性と呼ばれます。たとえば、1つの表にprod_idという主キー列、およびprod_nameという別の列があるとします。prod_id値を指定した場合、対応付けられたprod_nameを参照できます。この逆は真ではありません。つまり、prod_name値は一意のprod_idに関連付けられている必要はありません。

問合せが要求した列データが、マテリアライズド・ビューに含まれない場合、要求された列データを機能的に決定するキーがマテリアライズド・ビューに含まれていれば、それらの列データは、要求された列データを含む表をマテリアライズド・ビューに後戻り結合することによって取得できます。たとえば、次の問合せを考えてみます。

```
SELECT p.prod_category, t.week_ending_day, SUM(s.amount_sold)
FROM   sales s, products p, times t
WHERE  s.time_id=t.time_id  AND s.prod_id=p.prod_id AND p.prod_category='CD'
GROUP BY p.prod_category, t.week_ending_day;
```

マテリアライズド・ビューsum_sales_prod_week_mvにはp.prod_idが含まれていますが、p.prod_categoryは含まれていません。ただし、prod_idは機能的にprod_categoryを決定するため、sum_sales_prod_week_mvをproductsに再び結合して、prod_categoryを取り出すことができます。オプティマイザは、sum_sales_prod_week_mvを次のように使用して、この問合せをリライトします。

```
SELECT p.prod_category, mv.week_ending_day, SUM(mv.sum_amount_sold)
FROM   sum_sales_prod_week_mv mv, products p
WHERE  mv.prod_id=p.prod_id AND p.prod_category='CD'
GROUP BY p.prod_category, mv.week_ending_day;
```

この場合、products表は、マテリアライズド・ビューに結合されていたものが、リライトされた問合せで再び結合されたため、後戻り結合表と呼ばれます。

機能依存性は、次の2つの方法で宣言できます。

- 主キー制約を使用する方法(前述の例を参照)
- ディメンションのDETERMINES句を使用する方法

別の列を決定する列を主キーにできない場合、ディメンション定義のDETERMINES句のみが、機能依存性を宣言できる唯一の方法になることがあります。たとえば、products表は、prod_id、prod_nameおよびprod_subcategoryの各列を持つ非正規化ディメンション表です。prod_subcategoryはprod_subcat_descおよびprod_categoryを機能的に決定します。また、prod_categoryは、prod_cat_descを機能的に決定します。

最初の機能依存性は、prod_idを主キーとして宣言することで確立されますが、2番目の機能依存性は、prod_subcategory列に重複値が含まれるため、この方法では確立できません。そのような場合は、ディメンションのDETERMINES句を使用すると、2番目の機能依存性を宣言できます。

次のディメンション定義は、機能依存性の宣言方法を示します。

```
CREATE DIMENSION products_dim
        LEVEL product           IS (products.prod_id)
        LEVEL subcategory       IS (products.prod_subcategory)
        LEVEL category          IS (products.prod_category)
        HIERARCHY prod_rollup (
                product         CHILD OF
                subcategory     CHILD OF
                category
        )
        ATTRIBUTE product DETERMINES products.prod_name
        ATTRIBUTE product DETERMINES products.prod_desc
        ATTRIBUTE subcategory DETERMINES products.prod_subcat_desc
        ATTRIBUTE category DETERMINES products.prod_cat_desc;
```

階層prod_rollupは、1:n機能依存性でもある階層関係を宣言します。1:1機能依存性は、prod_subcategoryが機能的にprod_subcat_descを決定するように、DETERMINES句を使用して宣言されます。

次のマテリアライズド・ビューが作成されているものとします。

```
CREATE MATERIALIZED VIEW sum_sales_pscat_week_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_subcategory, t.week_ending_day,
       SUM(s.amount_sold) AS sum_amount_sole
FROM sales s, products p, times t
WHERE s.time_id = t.time_id AND s.prod_id = p.prod_id
GROUP BY p.prod_subcategory, t.week_ending_day;
```

次の問合せを考えてみます。

```
SELECT p.prod_subcategory_desc, t.week_ending_day, SUM(s.amount_sold)
FROM   sales s, products p, times t
WHERE  s.time_id=t.time_id AND s.prod_id=p.prod_id
AND    p.prod_subcat_desc LIKE '%Men'
GROUP BY p.prod_subcat_desc, t.week_ending_day;
```

この問合せは、sum_sales_pscat_week_mvをproducts表に結合することによりリライトできます。これにより、prod_subcat_descを使用して選択述語の評価を行うことが可能になります。ただし、結合は、products表の主キーでないprod_subcategory列をベースにするため、重複が許可されます。これは、個別値を選択するインライン・ビューを使用すること

によって実現され、このビューは、次のリライトされた問合せで示すように、マテリアライズド・ビューと結合されます。

```
SELECT iv.prod_subcat_desc, mv.week_ending_day, SUM(mv.sum_amount_sold)
FROM  sum_sales_pscat_week_mv mv,
     (SELECT DISTINCT prod_subcategory, prod_subcat_desc
      FROM products) iv
WHERE  mv.prod_subcategory=iv.prod_subcategory
AND iv.prod_subcat_desc LIKE '%Men'
GROUP BY iv.prod_subcat_desc, mv.week_ending_day;
```

このようなリライトが可能なのは、prod_subcategoryが、ディメンションで宣言されたようにprod_subcategory_descを機能的に決定することによります。

12.2.3 クエリー・リライト方法3: 集計可能性

問合せで要求された集計がマテリアライズド・ビューに格納された1つ以上の集計から導出または計算可能かどうかをオプティマイザで判断する場合にも、クエリー・リライトは可能です。たとえば、問合せがAVG(X)を要求し、マテリアライズド・ビューがSUM(X)およびCOUNT(X)を含む場合、AVG(X)はSUM(X)/COUNT(X)で計算できます。

また、マテリアライズド・ビューに格納された集計のロールアップが必要と判断され、ロールアップが可能な場合、クエリー・リライトでは、マテリアライズド・ビューの集計を使用して、問合せで要求された各集計もロールアップします。

たとえば、州の値が同じグループのSUM(sales)集計をすべて合計することによって、市レベルのSUM(sales)を州レベルのSUM(sales)にロールアップできます。ただし、COUNT(sales)またはSUM(sales)がマテリアライズド・ビューで使用可能でない場合、細分性が低いレベルにAVG(sales)をロールアップすることはできません。同様に、COUNT(sales)およびSUM(sales)がマテリアライズド・ビューで使用可能でない場合、VARIANCE(sales)またはSTDDEV(sales)はロールアップできません。たとえば、次の問合せを考えてみます。

```
ALTER TABLE times MODIFY CONSTRAINT time_pk RELY;
ALTER TABLE customers MODIFY CONSTRAINT customers_pk RELY;
ALTER TABLE sales MODIFY CONSTRAINT sales_time_pk RELY;
ALTER TABLE sales MODIFY CONSTRAINT sales_customer_fk RELY;
SELECT  p.prod_subcategory, AVG(s.amount_sold) AS avg_sales
FROM  sales s, products p WHERE s.prod_id = p.prod_id
GROUP BY p.prod_subcategory;
```

salesとtimes、およびsalesとcustomersの結合が可逆式で非重複の場合は、この文では、マテリアライズド・ビューsum_sales_pscat_month_city_mvをリライトに使用できます。さらに、問合せはprod_subcategoryによるグルーピング、マテリアライズド・ビューはprod_subcategory、calendar_month_descおよびcust_cityによるグルーピングであるため、マテリアライズド・ビューに格納された集計がロールアップされる必要があります。オプティマイザは、次のように問合せをリライトします。

```
SELECT mv.prod_subcategory, SUM(mv.sum_amount_sold)/COUNT(mv.count_amount_sold)
   AS avg_sales
FROM sum_sales_pscat_month_city_mv mv
GROUP BY mv.prod_subcategory;
```

SUMのような集計には、A+Bなどの算術式を引数とすることができます。オプティマイザでは、問合せ内の集計SUM(A+B)と、マテリアライズド・ビューに格納されている集計SUM(A+B)またはSUM(B+A)とを一致させるよう処理します。つまり、問合せ内の集計の引数とマテリアライズド・ビューの同様の集計の引数を一致させる際に、式の同等化を使用します。そのためには、同等である別々の2つの式が同じ標準形になるように、Oracleは集計引数式を標準的な形式に変換します。たとえば、A*(B-C)、A*B-C*A、(B-C)*Aおよび-A*C+A*Bはすべて同一の標準形に変換され、それによってこれらは正しく一致します。

12.2.4 クエリー・リライト方法4: 集計ロールアップ

問合せが要求したデータのグルーピングが、マテリアライズド・ビューに格納されているデータのグルーピングより細分性が低いレベルにある場合でも、オプティマイザはマテリアライズド・ビューを使用して問合せをリライトできます。たとえば、マテリアライズド・ビューsum_sales_pscat_week_mvは、prod_subcategoryとweek_ending_dayによるグルーピングです。この問合せは、prod_subcategoryによるグルーピング(より細分性が低いグルーピング)です。

```
ALTER TABLE times MODIFY CONSTRAINT time_pk RELY;
ALTER TABLE sales MODIFY CONSTRAINT sales_time_fk RELY;
SELECT p.prod_subcategory, SUM(s.amount_sold) AS sum_amount
FROM   sales s, products pWHERE  s.prod_id=p.prod_id
GROUP BY p.prod_subcategory;
```

したがって、オプティマイザは次のようにこの問合せをリライトします。

```
SELECT mv.prod_subcategory, SUM(mv.sum_amount_sold)
FROM   sum_sales_pscat_week_mv mv
GROUP BY mv.prod_subcategory;
```

12.2.5 クエリー・リライト方法5: ディメンションを使用したロールアップ

階層の異なるレベルでレポートが必要な場合、ディメンションが定義済であれば、階層内のレベルごとにマテリアライズド・ビューを作成する必要はありません。これは、クエリー・リライトが、ディメンション内の関係情報を使用して、マテリアライズド・ビューのデータを階層内の必要なレベルにロールアップできるためです。

次の例では、問合せはprod_categoryでグルーピングされたデータを要求し、マテリアライズド・ビューはprod_subcategoryでグルーピングされたデータを格納しています。prod_subcategoryがCHILD OF prod_categoryである場合(前述のディメンション例を参照)、マテリアライズド・ビューに格納されているグルーピングされたデータは、問合せがリライトされた場合にprod_categoryによってさらにグルーピングできます。つまり、マテリアライズド・ビューに格納されたprod_subcategoryレベルの(細分性がより高い)集計は、prod_categoryレベルの(細分性がより低い)集計にロールアップできます。

たとえば、次の問合せを考えてみます。

```
SELECT p.prod_category, t.week_ending_day, SUM(s.amount_sold) AS sum_amount
FROM   sales s, products p, times t
WHERE  s.time_id=t.time_id AND s.prod_id=p.prod_id
GROUP BY p.prod_category, t.week_ending_day;
```

prod_subcategoryは機能的にprod_categoryを決定するため、sum_sales_pscat_week_mvは、prod_category列データを取り出すためにproducts表の後戻り結合で使用されます。その後、集計は、次に示すようにprod_categoryレベルにロールアップされます。

```
SELECT pv.prod_category, mv.week_ending_day, SUM(mv.sum_amount_sold)
FROM   sum_sales_pscat_week_mv mv,
       (SELECT DISTINCT prod_subcategory, prod_category
        FROM products) pv
WHERE mv.prod_subcategory= pv.prod_subcategory
GROUP BY pv.prod_category, mv.week_ending_day;
```

12.2.6 クエリー・リライト方法6: マテリアライズド・ビューにデータのサブセットのみがある場合

Oracleではクエリー・リライトにHAVING句またはWHERE句によって1つの表または複数の表からデータを絞り込んでいるマテリアライズド・ビューを使用することをサポートします。たとえば、New Hampshire在住の顧客に限定したものなどです。つまりWHERE

句は、WHERE state = 'New Hampshire'となります。

このタイプのクエリー・リライトを実行するには、Oracleは、問合せで要求されたデータがマテリアライズド・ビューに格納されているデータに含まれているか、そのサブセットかどうかを判断する必要があります。以降の項では、Oracleでこの問題が解決され、ディテール表のデータのフィルタ処理済部分を含むマテリアライズド・ビューを使用するように問合せがリライトされる場合の条件について説明します。

フィルタ処理済データでクエリー・リライトを実行できるかどうかを判断するために、問合せとマテリアライズド・ビューの両方に選択述語(非結合)が含まれている場合に選択互換性チェックが実行されます。このチェックは、WHERE句およびHAVING句で行われます。マテリアライズド・ビューに選択が含まれ、問合せに含まれていなければ、マテリアライズド・ビューの方が問合せより限定的であるため、選択互換性チェックは失敗します。問合せに選択述語があり、マテリアライズド・ビューになければ、選択互換性チェックは不要です。

マテリアライズド・ビューのWHEREまたはHAVING句は、結合または絞込み選択(あるいはその両方)を使用でき、問合せのリライトにも使用されます。式を含む述語句や、特定の列の値に基づいて行を選択する述語句は、非結合述語の一例です。

この項では、次の項目について説明します。

- マテリアライズド・ビューにデータのサブセットが1つのみ含まれる場合のクエリー・リライトの定義
- マテリアライズド・ビューにデータのサブセットが1つのみ含まれる場合の選択カテゴリ
- クエリー・リライトの選択述語の例
- クエリー・リライトでのHAVING句の処理について
- マテリアライズド・ビューにINリストが含まれる場合のクエリー・リライトについて

12.2.6.1 マテリアライズド・ビューにデータのサブセットのみがある場合のクエリー・リライトの定義

クエリー・リライトでデータの1つのサブセットのみを処理する際に可能な事柄について説明する前に、次の定義について説明します。

- *join relop*

  (=、<、<=、>、>=)のいずれかです。

- *selection relop*

  (=、<、<=、>、>=、!=、[NOT] BETWEEN | IN| LIKE |NULL)のいずれかです。

- *join predicate*

  (*column1 join relop column2*)形式です。列は、現在の問合せブロックで同じFROM句に記述された異なる表の列です。したがって、外部参照などは使用できません。

- *selection predicate*

  左辺式 *relop* 右辺式、の形式です。すべての非結合述語は選択述語です。通常、左辺には列、右辺には値が含まれます。たとえば、color='red'の場合、左辺はcolor、右辺は'red'、関係演算子は(=)です。

12.2.6.2 マテリアライズド・ビューにデータのサブセットのみがある場合の選択のカテゴリ

選択述語は、次のように分類されます。

- 単純

  単純な選択述語は、*expression relop constant*という形式です。

- 複雑

複雑な選択述語は、*expression relop expression*という形式です。

- 範囲

  範囲選択述語は、WHERE (cust_last_name BETWEEN 'abacrombe' AND 'anakin')などの形式です。

  関係演算子(<、<=、>、>=)を含む単純な選択述語も、範囲選択述語とみなされます。

- INリスト

  WHERE(prod_id) IN (102, 233, ...)など、単一列と複数列のINリストです。

  (column1='v1' OR column1='v2' OR column1='v3' OR ....)形式の選択は、グループとして扱われ、INリストに分類されます。

- IS [NOT] NULL
- [NOT] LIKE
- その他

  その他の選択述語は、データの境界を判断できない場合です。たとえば、EXISTSなどです。

問合せの選択述語とマテリアライズド・ビューの選択述語の比較では、両者の選択述語の左辺が比較されます。

左辺の選択述語が一致した場合、右辺値でデータの包含がチェックされます。つまり、問合せの選択述語の右辺値が、マテリアライズド・ビューの選択述語の右辺値に含まれている必要があります。

選択述語で式を使用することもできます。この処理は次のようになります。

```
expression relational operator constant
```

*expression*には、Oracle Databaseで許可されている任意の算術式を使用できます。マテリアライズド・ビューと問合せの式は一致する必要があります。Oracleでは、A+BやB+Aなど、論理的に等価の式は常に同じ式として認識されます。

また、演算子または演算子として定義されるユーザー定義関数の左辺と右辺に式を持つ問合せを使用することもできます。クエリー・リライトは、マテリアライズド・ビューと問合せの複雑な選択述語が論理的に等価のときに発生します。これは、テキストの完全一致とは異なり、式が等価であるかぎり、条件を異なる順序で指定してもリライトできることを意味します。

12.2.6.3 クエリー・リライトの選択述語の例

次に、データがフィルタ処理される際にクエリー・リライトが可能な様々な例を示します。

例12-1 単一値の選択述語

問合せに次の句が含まれているとします。

```
WHERE prod_id = 102
```

また、マテリアライズド・ビューに次の句が含まれているとします。

```
WHERE prod_id BETWEEN 0 AND 200
```

この例では、左辺の選択述語がprod_idで一致し、問合せの右辺値102がマテリアライズド・ビューの範囲内にあるため、クエリー・リライトは可能です。

例12-2 境界付き範囲の選択述語

選択述語には、境界付き範囲(上限値と下限値を持つ範囲)を使用できます。たとえば、問合せに次の句が含まれているとします。

```
WHERE prod_id > 10 AND prod_id < 50
```

また、マテリアライズド・ビューに次の句が含まれているとします。

```
WHERE prod_id BETWEEN 0 AND 200
```

この場合、選択述語はprod_idで一致し、問合せの範囲はマテリアライズド・ビューの範囲内にあります。この例では、問合せの選択が両方とも同じ列に基づいていることがわかります。

例12-3 式を使用した選択述語

問合せに次の句が含まれているとします。

```
WHERE (sales.amount_sold * .07) BETWEEN 1.00 AND 100.00
```

また、マテリアライズド・ビューに次の句が含まれているとします。

```
WHERE (sales.amount_sold * .07) BETWEEN 0.0 AND 200.00
```

この例では、選択述語が(sales.amount_sold *.07)で一致し、問合せの右辺値がマテリアライズド・ビューの範囲内にあるため、クエリー・リライトは可能です。このような複雑な選択述語では、左辺と右辺がマテリアライズド・ビューの範囲内で一致している必要があります。

例12-4 完全一致の選択述語

問合せに次の句が含まれているとします。

```
WHERE (cost.unit_price * 0.95) > (cost_unit_cost  * 1.25)
```

また、マテリアライズド・ビューに次の句が含まれているとします。

```
WHERE (cost.unit_price * 0.95) > (cost_unit_cost  * 1.25)
```

左辺と右辺がマテリアライズド・ビューに一致し、*selection_relop*が同一の場合、通常はリライトされた問合せから選択述語を削除できます。それ以外の場合は、選択述語によりマテリアライズド・ビューから余分なデータを除外する必要があります。

クエリー・リライトで、リライトされた問合せから選択述語を削除できる場合は、選択述語のすべての列がマテリアライズド・ビュー内になくてもかまわないため、より多くのリライトを実行できます。これにより、マテリアライズド・ビューのデータが問合せより限定的でないことが保証されます。

例12-5 問合せでの追加選択述語

問合せの選択述語はマテリアライズド・ビューの選択述語と一致する必要はありませんが、一致する場合は、右辺値がマテリアライズド・ビューに含まれている必要があります。たとえば、問合せに次の句が含まれているとします。

```
WHERE prod_name = 'Shorts' AND prod_category = 'Men'
```

また、マテリアライズド・ビューに次の句が含まれているとします。

```
WHERE prod_category = 'Men'
```

この例では、prod_categoryのみが選択述語一致です。問合せには、マテリアライズド・ビューとは一致しない選択述語がありますが、マテリアライズド・ビューで、prod_nameを選択するか、ディテール表に後戻り結合してprod_nameを取得できる列を選択する場合は許容され、クエリー・リライトが可能です。唯一の要件は、選択述語prod_nameをマテリアライズド・ビューに適用する方法がクエリー・リライトに必要とされることです。

例12-6 問合せ内の選択述語が少ないためにリライトが発生しない場合

問合せに次の句が含まれているとします。

```
WHERE prod_category = 'Men'
```

また、マテリアライズド・ビューに次の句が含まれているとします。

```
WHERE prod_name = 'Shorts' AND prod_category = 'Men'
```

この例では、マテリアライズド・ビューの選択述語prod_nameが一致しません。マテリアライズド・ビューのみが製品Shortsを含んでいるので、マテリアライズド・ビューの方が問合せより限定的です。したがって、クエリー・リライトは発生しません。

例12-7 複数列のINリストの選択述語

クエリー・リライトでは、問合せに複数列のINリストがあり、その各列がマテリアライズド・ビューの単一列のINリストからの個々の列と完全に一致している場合も、チェックされます。たとえば、問合せに次の句が含まれているとします。

```
WHERE (prod_id, cust_id) IN ((1022, 1000), (1033, 2000))
```

また、マテリアライズド・ビューに次の句が含まれているとします。

```
WHERE prod_id IN (1022,1033) AND cust_id IN (1000, 2000)
```

この例では、マテリアライズド・ビューのINリストは、問合せの複数列のINリスト内の列と一致しています。さらに、問合せの選択の右辺値は、マテリアライズド・ビューに含まれているため、リライトが発生します。

例12-8 INリストを使用した選択述語

選択互換性では、マテリアライズド・ビューに複数列のINリストがあり、その各列が問合せ内のINリストの1つ以上の列と完全に一致している場合も、チェックされます。たとえば、問合せに次の句が含まれているとします。

```
WHERE prod_id = 1022 AND cust_id IN (1000, 2000)
```

また、マテリアライズド・ビューに次の句が含まれているとします。

```
WHERE (prod_id, cust_id) IN ((1022, 1000), (1022, 2000))
```

この例では、マテリアライズド・ビューのINリストの列は、問合せの選択述語の列と一致しています。さらに、問合せの選択述語の右辺値は、マテリアライズド・ビューに含まれています。したがって、リライトは成功します。

例12-9 複数の選択述語または式

問合せに次の句が含まれているとします。

```
WHERE (city_population > 15000 AND city_population < 25000
   AND state_name = 'New Hampshire')
```

また、マテリアライズド・ビューに次の句が含まれているとします。

```
WHERE (city_population < 5000 AND state_name = 'New York') OR
   (city_population BETWEEN 10000 AND 50000 AND state_name = 'New Hampshire')
```

この例では、問合せに単一の離接詞(ANDで区切られた選択述語のグループ)があります。また、マテリアライズド・ビューには、ORで区切られた離接詞が2つあります。問合せの単一の離接詞はマテリアライズド・ビューの2番目の離接詞に含まれているため、選択互換性チェックにパスします。マテリアライズド・ビューには、問合せで必要とするより多くのデータが含まれているため、問合せをリライトできることは明らかです。

12.2.6.4 クエリー・リライトでのHAVING句の処理について

問合せで指定された範囲がマテリアライズド・ビューで指定された範囲内にあれば、SUM(s.amount_sold) BETWEEN 10000 AND 20000のように、問合せのHAVING句で集計値の範囲が指定されている場合でも、クエリー・リライトが可能です。

```
CREATE MATERIALIZED VIEW product_sales_mv
BUILD IMMEDIATE
```

```
REFRESH FORCE
ENABLE QUERY REWRITE AS
SELECT p.prod_name, SUM(s.amount_sold) AS dollar_sales
FROM products p, sales s
WHERE p.prod_id = s.prod_id
GROUP BY prod_name
HAVING SUM(s.amount_sold) BETWEEN 5000 AND 50000;
```

したがって、次の問合せはリライトできます。

```
SELECT p.prod_name, SUM(s.amount_sold) AS dollar_sales
FROM products p, sales s WHERE p.prod_id = s.prod_id
GROUP BY prod_name
HAVING SUM(s.amount_sold) BETWEEN 10000 AND 20000;
```

この問合せは、次のようにリライトされます。

```
SELECT mv.prod_name, mv.dollar_sales FROM product_sales_mv mv
WHERE mv.dollar_sales BETWEEN 10000 AND 20000;
```

12.2.6.5 マテリアライズド・ビューにINリストが含まれる場合のクエリー・リライトについて

マテリアライズド・ビューにINリストが含まれる際、クエリー・リライトを使用できます。たとえば、次のマテリアライズド・ビューの定義があるとします。

```
CREATE MATERIALIZED VIEW popular_promo_sales_mv
BUILD IMMEDIATE
REFRESH FORCE
ENABLE QUERY REWRITE AS
SELECT p.promo_name, SUM(s.amount_sold) AS sum_amount_sold
FROM  promotions p, sales s
WHERE s.promo_id = p.promo_id
AND p.promo_name IN ('coupon', 'premium', 'giveaway')
GROUP BY promo_name;
```

次の問合せはリライトできます。

```
SELECT p.promo_name, SUM(s.amount_sold)
FROM  promotions p, sales s
WHERE s.promo_id = p.promo_id AND p.promo_name IN ('coupon', 'premium')
GROUP BY p.promo_name;
```

この問合せは、次のようにリライトされます。

```
SELECT * FROM popular_promo_sales_mv mv
WHERE mv.promo_name IN ('coupon', 'premium');
```

12.2.7 パーティション・チェンジ・トラッキング(PCT)リライト

PCTリライトでは、オプティマイザは、一部のみが最新のマテリアライズド・ビューを使用して、問合せを最新のデータで正確にリライトできます。そのために、ディテール表内で更新されたパーティションが追跡されます。その次に、ディテール表内の更新されたパーティションに基づくマテリアライズド・ビュー内の行が追跡されます。これにより、オプティマイザは、マテリアライズド・ビューの最新と認識される部分を使用できるようになります。最新かどうかに関する詳細は、DBA_MVIEWSビュー、DBA_DETAIL_RELATIONSビューおよびDBA_MVIEW_DETAIL_PARTITIONビューで確認できます。これらのビューの使用例はパーティションの最新状態の表示を参照してください。

オプティマイザでは、QUERY_REWRITE_INTEGRITY = ENFORCEDモードまたはTRUSTEDモードでPCTリライトを使用します。

STALE_TOLERATEDモードでは、データが最新かどうかが考慮されないため、PCTリライトは使用されません。また、PCTリライトが実行されるためにはWHERE句が必要です。

PCTリライトは、パーティション化を伴う場合でも使用できますが、ハッシュ・パーティション化はサポートされていません。次のトピックでは、PCTの使用における側面について説明します。

- レンジ・パーティション表に基づいたPCTリライト
- レンジ-リスト・パーティション表に基づいたPCTリライト
- リスト・パーティション表に基づいたPCTリライト
- PCTリライトとPMARKER
- PMARKERとしてROWIDを使用したPCTリライト

12.2.7.1 レンジ・パーティション表に基づいたPCTリライト

次に示すのは、マテリアライズド・ビューのPCTがパーティション・キーで有効になっており、さらに基になる実表がtimeキーでレンジ・パーティション化されている場合のPCTリライトの例です。

```
CREATE TABLE part_sales_by_time (time_id, prod_id, amount_sold,
       quantity_sold)
  PARTITION BY RANGE (time_id)
  (
    PARTITION old_data
      VALUES LESS THAN (TO_DATE('01-01-1999', 'DD-MM-YYYY'))
      PCTFREE 0
      STORAGE (INITIAL 8M),
    PARTITION quarter1
      VALUES LESS THAN (TO_DATE('01-04-1999', 'DD-MM-YYYY'))
      PCTFREE 0
      STORAGE (INITIAL 8M),
    PARTITION quarter2
      VALUES LESS THAN (TO_DATE('01-07-1999', 'DD-MM-YYYY'))
      PCTFREE 0
      STORAGE (INITIAL 8M),
    PARTITION quarter3
      VALUES LESS THAN (TO_DATE('01-10-1999', 'DD-MM-YYYY'))
      PCTFREE 0
      STORAGE (INITIAL 8M),
    PARTITION quarter4
      VALUES LESS THAN (TO_DATE('01-01-2000', 'DD-MM-YYYY'))
      PCTFREE 0
      STORAGE (INITIAL 8M),
    PARTITION max_partition
      VALUES LESS THAN (MAXVALUE)
      PCTFREE 0
      STORAGE (INITIAL 8M)
  )
  AS
  SELECT s.time_id, s.prod_id, s.amount_sold, s.quantity_sold
  FROM sales s;
```

ここで、1日ごとに販売された製品の総数を含むマテリアライズド・ビューを作成します。

```
CREATE MATERIALIZED VIEW  sales_in_1999_mv
  BUILD IMMEDIATE
  REFRESH FORCE ON DEMAND
  ENABLE QUERY REWRITE
```

```
  AS
  SELECT s.time_id, s.prod_id, p.prod_name, SUM(quantity_sold)
  FROM part_sales_by_time s, products p
  WHERE p.prod_id = s.prod_id
    AND s.time_id BETWEEN TO_DATE('01-01-1999', 'DD-MM-YYYY')
    AND TO_DATE('31-12-1999', 'DD-MM-YYYY')
  GROUP BY s.time_id, s.prod_id, p.prod_name;
```

次に示す問合せが、マテリアライズド・ビューsales_in_1999_mvでリライトされます。

```
SELECT s.time_id, p.prod_name, SUM(quantity_sold)
  FROM part_sales_by_time s, products p
  WHERE p.prod_id = s.prod_id
    AND s.time_id < TO_DATE('01-02-1999', 'DD-MM-YYYY')
    AND s.time_id >= TO_DATE('01-01-1999', 'DD-MM-YYYY')
  GROUP BY s.time_id, p.prod_name;
```

part_sales_by_timeのquarter4に行を追加する場合は、次のようになります。

```
INSERT INTO part_sales_by_time
  VALUES (TO_DATE('26-12-1999', 'DD-MM-YYYY'),38920,2500, 20);

commit;
```

このとき、マテリアライズド・ビューsales_in_1999_mvは失効状態になります。PCTリライトを使用すると、マテリアライズド・ビューの最新部分のデータのみを要求する問合せをリライトできます。マテリアライズド・ビューsales_in_1999_mvは、SELECT句およびGROUP BY句にtime_idが含まれています。したがってPCTは有効になっており、次の問合せはquarter4のデータを要求しないように正しくリライトされます。

```
SELECT s.time_id, p.prod_name, SUM(quantity_sold)
  FROM part_sales_by_time s, products p
  WHERE p.prod_id = s.prod_id
  AND s.time_id < TO_DATE('01-07-1999', 'DD-MM-YYYY')
  AND s.time_id >= TO_DATE('01-03-1999', 'DD-MM-YYYY')
  GROUP BY s.time_id, p.prod_name;
```

複数のマテリアライズド・ビューによるリライトが無効になっている場合、次の問合せはリライトできません。デフォルトでは、複数のマテリアライズド・ビューによるリライトが有効になっているため、次の問合せはマテリアライズド・ビューと実表を使用してリライトされます。

```
SELECT s.time_id, p.prod_name, SUM(quantity_sold)
  FROM part_sales_by_time s, products p
  WHERE p.prod_id = s.prod_id
  AND s.time_id < TO_DATE('31-10-1999', 'DD-MM-YYYY') AND
      s.time_id > TO_DATE('01-07-1999', 'DD-MM-YYYY')
  GROUP BY s.time_id, p.prod_name;
```

12.2.7.2 レンジ-リスト・パーティション表に基づいたPCTリライト

ディテール表がレンジ-リスト・パーティション化されている場合、このディテール表に依存するマテリアライズド・ビューは、パーティション化とサブパーティション化の両方のレベルでPCTをサポートできます。パーティション化キーとサブパーティション化キーの両方がマテリアライズド・ビューにある場合、より細かくPCTを実行できます。これにより、マテリアライズド・ビューのより細かな部分に対してマテリアライズド・ビューをリフレッシュでき、失効したマテリアライズド・ビューでより多くの問合せをリライトできるようになります。また、マテリアライズド・ビューにパーティション化キーしかない場合は、PCTは粗く実行されます。

次のレンジ-リスト・パーティション表を考えてみます。

```
CREATE TABLE sales_par_range_list
```

```
 (calendar_year, calendar_month_number, day_number_in_month,
  country_name, prod_id, prod_name, quantity_sold, amount_sold)
PARTITION BY RANGE (calendar_month_number)
SUBPARTITION BY LIST (country_name)
 (PARTITION q1 VALUES LESS THAN (4)
 (SUBPARTITION q1_America VALUES
 ('United States of America', 'Argentina'),
   SUBPARTITION q1_Asia VALUES ('Japan', 'India'),
   SUBPARTITION q1_Europe VALUES ('France', 'Spain', 'Ireland')),
   PARTITION q2 VALUES LESS THAN (7)
  (SUBPARTITION q2_America VALUES
   ('United States of America', 'Argentina'),
   SUBPARTITION q2_Asia VALUES ('Japan', 'India'),
   SUBPARTITION q2_Europe VALUES ('France', 'Spain', 'Ireland')),
     PARTITION q3 VALUES LESS THAN (10)
  (SUBPARTITION q3_America VALUES
   ('United States of America', 'Argentina'),
   SUBPARTITION q3_Asia VALUES ('Japan', 'India'),
   SUBPARTITION q3_Europe VALUES ('France', 'Spain', 'Ireland')),
     PARTITION q4 VALUES LESS THAN (13)
  (SUBPARTITION q4_America VALUES
   ('United States of America', 'Argentina'),
   SUBPARTITION q4_Asia VALUES ('Japan', 'India'),
   SUBPARTITION q4_Europe VALUES ('France', 'Spain', 'Ireland')))
  AS SELECT t.calendar_year, t.calendar_month_number,
     t.day_number_in_month, c1.country_name, s.prod_id,
     p.prod_name, s.quantity_sold, s.amount_sold
  FROM times t, countries c1, products p, sales s, customers c2
  WHERE s.time_id = t.time_id AND s.prod_id = p.prod_id AND
        s.cust_id = c2.cust_id AND c2.country_id = c1.country_id AND
        c1.country_name IN ('United States of America', 'Argentina',
          'Japan', 'India', 'France', 'Spain', 'Ireland');
```

製品に関する各年の月ごとの合計売上高を持つ次のマテリアライズド・ビューsum_sales_per_year_month_mvを考えてみます。

```
CREATE MATERIALIZED VIEW  sum_sales_per_year_month_mv
BUILD IMMEDIATE
REFRESH FORCE ON DEMAND
ENABLE QUERY REWRITE AS
SELECT s.calendar_year, s.calendar_month_number,
         SUM(s.amount_sold) AS sum_sales, COUNT(*) AS cnt
FROM sales_par_range_list s WHERE s.calendar_year > 1990
GROUP BY s.calendar_year, s.calendar_month_number;
```

sales_per_country_mvは、レンジ・パーティション化レベルでsales_par_range_listに対するPCTをサポートします。これは、レンジ・パーティション・キーcalendar_month_numberが、SELECTおよびGROUP BYリストにあるためです。

```
INSERT INTO sales_par_range_list
   VALUES (2001, 3, 25, 'Spain', 20, 'PROD20',  300,  20.50);
```

この文では、calendar_month_number = 3およびcountry_name = 'Spain'を含む行を挿入します。この行は、パーティションq1、サブパーティションEuropeに挿入されます。このINSERT文の後、sales_par_range_listのパーティションq1に関して、sum_sales_per_year_month_mvは失効します。したがって、次の文のように、sales_par_range_listのこのパーティションのデータにアクセスする入力問合せは、すべてリライトできません。

次の問合せでは、パーティションq1およびq2のデータにアクセスします。q1が更新され、マテリアライズド・ビューはq1に関して失効しているため、PCTリライトは使用できません。

```
SELECT s.calendar_year, SUM(s.amount_sold) AS sum_sales, COUNT(*) AS cnt
```

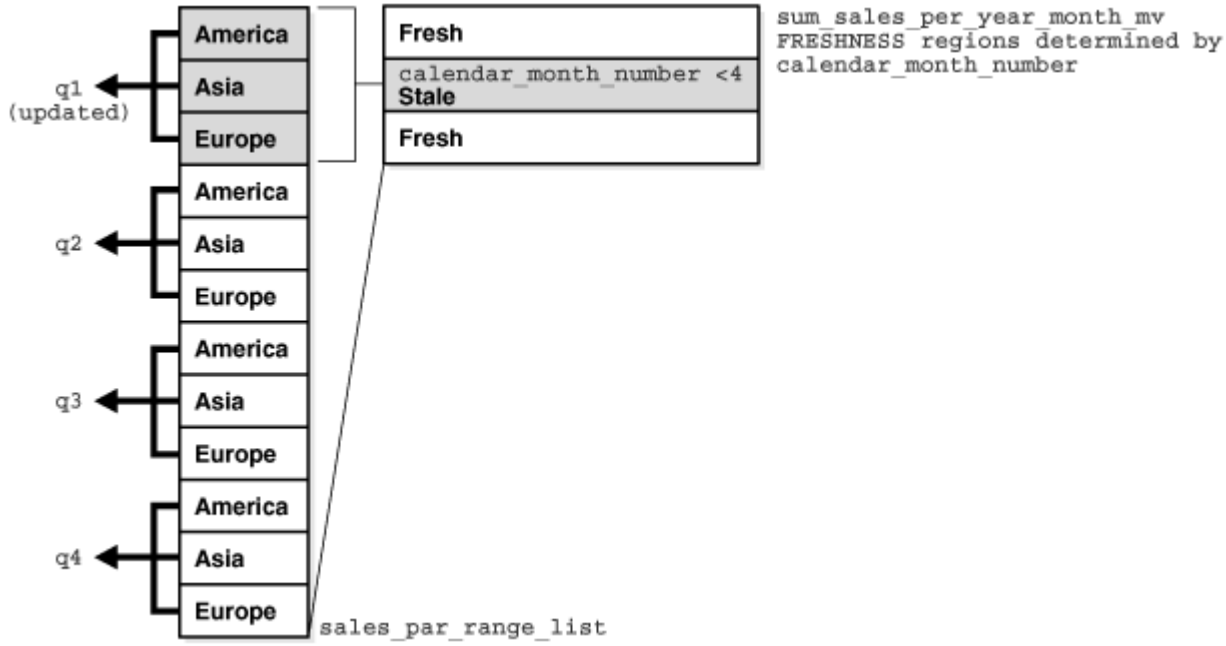
```
FROM sales_par_range_list s
WHERE s.calendar_year = 2000
  AND s.calendar_month_number BETWEEN 2 AND 6
GROUP BY s.calendar_year;
```

次に、INSERT文の後にリライトを行う文の例を示します。これは最新のデータにアクセスするためです。

```
SELECT s.calendar_year, SUM(s.amount_sold) AS sum_sales, COUNT(*) AS cnt
FROM sales_par_range_list s
WHERE s.calendar_year = 2000 AND s.calendar_month_number BETWEEN 5 AND 9
GROUP BY s.calendar_year;
```

図12-3に、失効した部分と最新の部分を示します。

図12-3 PCTリライトとレンジ-リスト・パーティション化



12.2.7.3 リスト・パーティション表に基づいたPCTリライト

マテリアライズド・ビューのSELECTおよびGROUP BYにLISTパーティション化キーがある場合、PCTはマテリアライズド・ビューでサポートされます。サポートされているパーティション化のタイプに関係なく、ディテール表のパーティション・マーカーまたはROWIDがマテリアライズド・ビューにある場合、PCTは該当する特定のディテール表のマテリアライズド・ビューでサポートされます。

```
CREATE TABLE sales_par_list
(calendar_year, calendar_month_number, day_number_in_month,
 country_name, prod_id, quantity_sold, amount_sold)
 PARTITION BY LIST (country_name)
 (PARTITION America
      VALUES ('United States of America', 'Argentina'),
  PARTITION Asia
      VALUES ('Japan', 'India'),
  PARTITION Europe
      VALUES ('France', 'Spain', 'Ireland'))
  AS SELECT t.calendar_year, t.calendar_month_number,
         t.day_number_in_month, c1.country_name, s.prod_id,
         s.quantity_sold, s.amount_sold
  FROM times t, countries c1, sales s, customers c2
  WHERE s.time_id = t.time_id and s.cust_id = c2.cust_id and
        c2.country_id = c1.country_id and
        c1.country_name IN ('United States of America', 'Argentina',
       'Japan', 'India', 'France', 'Spain', 'Ireland');
```

マテリアライズド・ビューが表sales_par_listで作成され、その中にリスト・パーティション化キーがある場合、PCTリライトでは、このマテリアライズド・ビューをリライト対象として使用します。

この機能を理解するために、毎年の各国における全製品の合計売上高を持つマテリアライズド・ビューの作成例を次に示します。ビューは、ディテール表sales_par_listおよびproductsに依存しています。

```
CREATE MATERIALIZED VIEW sales_per_country_mv
BUILD IMMEDIATE
REFRESH FORCE ON DEMAND
ENABLE QUERY REWRITE AS
SELECT s.calendar_year AS calendar_year, s.country_name AS country_name,
 p.prod_name AS prod_name, SUM(s.amount_sold) AS sum_sales, COUNT(*) AS cnt
FROM sales_par_list s, products p
WHERE s.prod_id = p.prod_id AND s.calendar_year <= 2000
GROUP BY s.calendar_year, s.country_name, prod_name;
```

sales_per_country_mvは、sales_par_listに対するPCTをサポートします。これは、リスト・パーティション・キーcountry_nameが、SELECTおよびGROUP BYリストにあるためです。表productsはパーティション化されません。したがってsales_per_country_mvはこの表に対するPCTをサポートしていません。

入力問合せがマテリアライズド・ビューの最新の部分にのみアクセスする場合に、sales_per_country_mvが失効していても、問合せは、(ENFORCEDモードまたはTRUSTEDモードで)sales_per_country_mvに関して、リライトできます。FRESHであるマテリアライズド・ビューの部分を判別できるのは、更新された表がマテリアライズド・ビューでPCTを使用できる場合のみです。したがって、PCTを使用できない表が更新された場合、マテリアライズド・ビューのFRESHの部分を判別できないため、特定のマテリアライズド・ビューの最新データによるリライトはできません。

sales_per_country_mvは、sales_par_listのPCTはサポートしますが、表productのPCTはサポートしません。表productsが更新されても、マテリアライズド・ビューのどの部分がFRESHであるかを識別できないため、sales_per_country_mvでのPCTリライトはできません。

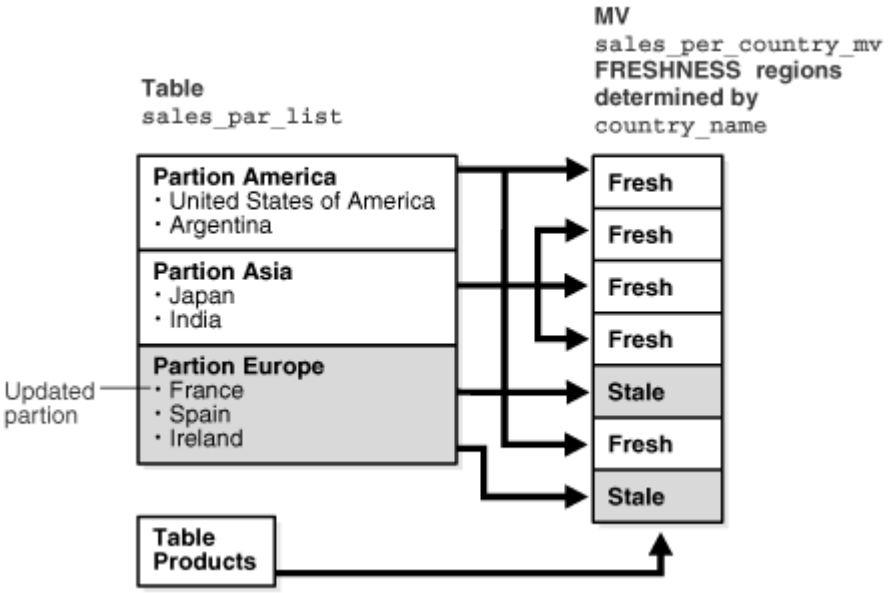
次の文は、sales_par_listを更新します。

```
INSERT INTO sales_par_list VALUES (2000, 10, 22, 'France', 900, 20, 200.99);
```

この文では、表sales_par_list内のパーティションEuropeに行を挿入しています。この結果、sales_per_country_mvは失効しますが、このマテリアライズド・ビューは表sales_par_listに対するPCTをサポートするため、(ENFORCEDモードおよびTRUSTEDモードでの)PCTリライトが可能となります。マテリアライズド・ビューの最新の部分および失効した部分は、パーティション化されたディテール表sales_par_listに基づいて識別されます。

図12-4に、この例における最新の部分と失効した部分を示します。

図12-4 PCTリライトとリスト・パーティション化



次の問合せについて考えてみます。

```
SELECT s.country_name, p.prod_name, SUM(s.amount_sold) AS sum_sales,
   COUNT(*) AS cnt
FROM sales_par_list s, products p
WHERE s.prod_id = p.prod_id AND s.calendar_year = 2000
  AND s.country_name IN ('United States of America', 'Japan')
GROUP BY s.country_name, p.prod_name;
```

この問合せでは、sales_par_listのパーティションAmericaおよびAsiaにアクセスします。これらのパーティションは未更新であり、この問合せのアクセス対象はマテリアライズド・ビューのFRESHの部分に限定されるため、失効したマテリアライズド・ビューsales_per_country_mvでリライトできます。

問合せは、次のように、sales_per_country_mvでリライトされます。

```
SELECT country_name, prod_name, SUM(sum_sales) AS sum_sales, SUM(cnt) AS cnt
FROM sales_per_country_mv WHERE calendar_year = 2000
  AND country_name IN ('United States of America', 'Japan')
GROUP BY country_name, prod_name;
```

次の問合せを考えてみます。

```
SELECT s.country_name,  p.prod_name,
 SUM(s.amount_sold) AS sum_sales, COUNT(*) AS cnt
FROM sales_par_list s, products p
WHERE  s.prod_id = p.prod_id AND s.calendar_year = 1999
  AND s.country_name IN ('Japan', 'India', 'Spain')
GROUP BY s.country_name, p.prod_name;
```

この問合せでは、sales_par_listのパーティションEuropeおよびAsiaにアクセスします。パーティションEuropeは更新されていますが、マテリアライズド・ビューの必要なデータが失効しているため、この問合せはsales_per_country_mvでリライトできません。

入力問合せがマテリアライズド・ビューのFRESHの部分にアクセスする場合、DML、ダイレクト・ロード、パーティション・メンテナンス操作(PMOP)など、sales_par_listに対する任意の種類の更新後に、リライトできるようになります。

12.2.7.4 PCTリライトとPMARKER

パーティション・マーカーを指定して、特定のパーティションのすべての行が同じpmarker値を持つと、クエリー・リライト機能が、ディテール表の全パーティションにアクセスする問合せのリライトに制限されます。つまり、ディテール表のパーティションの一部にアクセスする問合せの場合、マテリアライズド・ビューのFRESHの部分にデータが対応していても、リライトは行われません。この場合、マテリアライズド・ビューのFRESHの部分は、pmarker値によって決定されます。マテリアライズド・ビューの最新の行を決定するに

は、データの新しさとマーカー値を関連付けます。これにより、特定のpmarker値を持つマテリアライズド・ビューのすべての行が、FRESHまたはSTALEになります。

次に、各月について、sales_par_listの各ディテール表のパーティションにおける、全製品の合計売上高を表すマテリアライズド・ビューを作成する例を示します。次に示すように、このマテリアライズド・ビューはディテール表productsにも依存します。

```
CREATE MATERIALIZED VIEW sales_per_dt_partition_mv
BUILD IMMEDIATE
REFRESH FORCE ON DEMAND
ENABLE QUERY REWRITE AS
SELECT s.calendar_year AS calendar_year, p.prod_name AS prod_name,
       DBMS_MVIEW.PMARKER(s.rowid) pmarker,
       SUM(s.amount_sold) AS sum_sales, COUNT(*) AS cnt
FROM sales_par_list s, products p
WHERE s.prod_id = p.prod_id AND s.calendar_year > 2000
GROUP BY s.calendar_year, DBMS_MVIEW.PMARKER(s.rowid), p.prod_name;
```

マテリアライズド・ビューsales_per_dt_partition_mvでは、ディテール表のパーティションごとの合計売上高を提供します。このマテリアライズド・ビューは、パーティション・マーカーがSELECT句およびGROUP BY句にあるため、表sales_par_listに対するPCTリライトをサポートします。表12-2に、この例でのパーティションの名前およびpmarkerを示します。

表12-2 パーティションの名前およびPmarker

パーティション名	Pmarker
America	1000
Asia	1001
Europe	1002

次のように、sales_par_listを更新します。

```
DELETE FROM sales_par_list WHERE country_name = 'India';
```

これで、表sales_par_listのパーティションAsiaから行が削除されました。この結果、sales_per_dt_partition_mvは失効しますが、このマテリアライズド・ビューは表sales_par_listに対するPCT(pmarkerベース)をサポートするため、(ENFORCEDモードおよびTRUSTEDモードでの)PCTリライトが可能となります。

次の問合せを考えてみます。

```
SELECT p.prod_name, SUM(s.amount_sold) AS sum_sales, COUNT(*) AS cnt
FROM sales_par_list s, products p
WHERE  s.prod_id = p.prod_id AND s.calendar_year = 2001 AND
       s.country_name IN ('United States of America', 'Argentina')
GROUP BY p.prod_name;
```

ディテール表のパーティションに対応するすべてのデータがアクセスされ、このデータに関してマテリアライズド・ビューはFRESHであるため、この問合せはsales_per_dt_partition_mvでリライトできます。この問合せは、未更新のパーティションAmericaのすべてのデータにアクセスします。

問合せは、次のように、sales_per_dt_partition_mvでリライトされます。

```
SELECT prod_name, SUM(sum_sales) AS sum_sales, SUM(cnt) AS cnt
FROM sales_per_dt_partition_mv
WHERE calendar_year = 2001 AND pmarker = 1000
```

```
GROUP BY prod_name;
```

12.2.7.5 PMARKERとしてROWIDを使用したPCTリライト

マテリアライズド・ビューは、GROUP BY句があり、SELECTおよびGROUP BY句にパーティション・キーまたはパーティション・マーカーが指定されている場合、PCTリライトをサポートします。pmarkerまたはパーティション・キーのかわりに、パーティション表のROWIDを使用できます。この場合、内部的にROWIDがpmarkerに変換されます。次の表を考えてみます。

```
CREATE TABLE product_par_list
(prod_id, prod_name, prod_category,
 prod_subcategory, prod_list_price)
 PARTITION BY LIST (prod_category)
 (PARTITION prod_cat1
      VALUES ('Boys', 'Men'),
  PARTITION prod_cat2
      VALUES ('Girls', 'Women'))
 AS
   SELECT prod_id, prod_name, prod_category,
      prod_subcategory, prod_list_price
   FROM products;
```

表sales_par_listおよびproduct_par_listのマテリアライズド・ビューを作成する例を次に示します。

```
CREATE MATERIALIZED VIEW  sum_sales_per_category_mv
BUILD IMMEDIATE
REFRESH FORCE ON DEMAND
ENABLE QUERY REWRITE AS
SELECT p.rowid prid, p.prod_category,
       SUM (s.amount_sold) sum_sales, COUNT(*) cnt
FROM sales_par_list s, product_par_list p
WHERE s.prod_id = p.prod_id and s.calendar_year <= 2000
GROUP BY p.rowid, p.prod_category;
```

pmarkerでのリライトに適用されるすべての制限が、ここでも適用されます。入力問合せは、リライト対象の問合せの全パーティションにアクセスする必要があります。この場合に使用されるpmarker表を次に示します。

```
product_par_list       pmarker value
----------------       -------------
prod_cat1                      1000
prod_cat2                      1001
prod_cat3                      1002
```

次のように、product_par_listを更新します。

```
DELETE FROM product_par_list WHERE prod_name = 'MEN';
```

この結果、product_par_listのパーティションprod_list1に関して、sum_sales_per_category_mvは失効します。

次の問合せを考えてみます。

```
SELECT p.prod_category, SUM(s.amount_sold) AS sum_sales, COUNT(*) AS cnt
FROM sales_par_list s, product_par_list p
WHERE  s.prod_id = p.prod_id AND p.prod_category IN
       ('Girls', 'Women') AND s.calendar_year <= 2000
GROUP BY p.prod_category;
```

ディテール表のパーティションに対応するすべてのデータがアクセスされ、このデータに関してマテリアライズド・ビューはFRESHであるため、この問合せはsum_sales_per_category_mvでリライトできます。この問合せは、未更新のパーティションprod_cat2のす

べてのデータにアクセスします。次に、sum_sales_per_category_mvでリライトされた問合せを示します。

```
SELECT prod_category, sum_sales, cnt
FROM sum_sales_per_category_mv WHERE DBMS_MVIEW.PMARKER(srid) IN (1000)
GROUP BY prod_category;
```

12.2.8 複数のマテリアライズド・ビューを使用したクエリー・リライトについて

クエリー・リライトは、複数のマテリアライズド・ビューを使用した問合せのリライトができるように拡張されました。クエリー・リライトにおいて、すべてのデータを戻すマテリアライズド・ビューのセットがないと判断された場合、残りのデータは実表から取得されます。

複数のマテリアライズド・ビューを使用したクエリー・リライトでは、PCTやINリストを使用したリライトなど、様々なタイプのリライトを活用できる他、それらを組み合せて使用することもできます。以下では、現在クエリー・リライトが可能な問合せについて、具体例をいくつか取り上げます。

2つのマテリアライズド・ビュー、cust_avg_credit_mv1とcust_avg_credit_mv2について考えてみます。
cust_avg_credit_mv1では、各郵便番号について、1940年から1950年に生まれたすべての顧客の平均与信限度額を問い合せます。cust_avg_credit_mv2では、各郵便番号について、1951年から1970年に生まれた顧客の平均与信限度額を問い合せます。

この例で定義されるマテリアライズド・ビューを次に示します。

```
CREATE MATERIALIZED VIEW cust_avg_credit_mv1
ENABLE QUERY REWRITE
AS SELECT cust_postal_code, cust_year_of_birth,
       SUM(cust_credit_limit) AS sum_credit,
       COUNT(cust_credit_limit) AS count_credit
FROM customers
WHERE cust_year_of_birth BETWEEN 1940 AND 1950
GROUP BY cust_postal_code, cust_year_of_birth;

CREATE MATERIALIZED VIEW cust_avg_credit_mv2
ENABLE QUERY REWRITE
AS SELECT cust_postal_code, cust_year_of_birth,
       SUM(cust_credit_limit) AS sum_credit,
       COUNT(cust_credit_limit) AS count_credit
FROM customers
WHERE cust_year_of_birth > 1950 AND cust_year_of_birth <= 1970
GROUP BY cust_postal_code, cust_year_of_birth;
```

問合せ1: マテリアライズド・ビューと問合せにおける1つの範囲の一致

各郵便番号について、1940年から1970年に生まれたすべての顧客の平均与信限度額を問い合せる問合せを考えてみます。この問合せは、cust_year_of_birthに対してBETWEENで指定した範囲と一致します。

```
SELECT cust_postal_code, AVG(cust_credit_limit) AS avg_credit
FROM customers c
WHERE cust_year_of_birth BETWEEN 1940 AND 1970
GROUP BY cust_postal_code;
```

前述の問合せは、次のように、すべてのデータを取得するために、2つのマテリアライズド・ビューでリライトできます。

```
SELECT v1.cust_postal_code,
SUM(v1.sum_credit)/SUM(v1.count_credit) AS avg_credit
FROM (SELECT cust_postal_code, sum_credit, count_credit
 FROM cust_avg_credit_mv1
 GROUP BY cust_postal_code
 UNION ALL
```

```
SELECT cust_postal_code, sum_credit, count_credit
FROM cust_avg_credit_mv2
GROUP BY cust_postal_code) v1
GROUP BY v1.cust_postal_code;
```

インライン・ビューにUNION ALLの問合せが使用されているのは、再集計の必要があるためです。また、クエリー・リライトがどのように集計をカウントしてこのロールアップを実行したかを確認してください。

問合せ2: マテリアライズド・ビューに含まれるデータ外の問合せ

マテリアライズド・ビューで指定している範囲が、問合せで問い合せる範囲を超える場合、リライトされる問合せにはフィルタ(選択述語)が追加され、マテリアライズド・ビューによって戻される不要な行が削除されます。これは、次のような問合せの場合です。

```
SELECT cust_postal_code, SUM(cust_credit_limit) AS sum_credit
FROM customers c
WHERE cust_year_of_birth BETWEEN 1945 AND 1955
GROUP BY cust_postal_code;
```

問合せ2は、次のようにリライトされます。

```
SELECT v1.cust_postal_code, SUM(v1.sum_credit)
FROM
(SELECT cust_postal_code, SUM(sum_credit) AS sum_credit
FROM cust_avg_credit_mv1
WHERE cust_year_of_birth BETWEEN 1945 AND 1950
GROUP BY cust_postal_code
UNION ALL
SELECT cust_postal_code, SUM(sum_credit) AS sum_credit
FROM cust_birth_mv2
WHERE cust_year_of_birth > 1950 AND cust_year_of_birth <= 1955
GROUP BY cust_postal_code) v1
GROUP BY v1.cust_postal_code;
```

問合せ3: マテリアライズド・ビューよりも多くのデータの問合せ

問合せで、前述の2つのマテリアライズド・ビューに含まれているデータよりも多くのデータを問い合せる場合を考えます。その場合でも、両方のマテリアライズド・ビューおよび実表のデータを使用してリライトされます。次の例では、集計がないマテリアライズド・ビューの新しいセットを定義し、両方のマテリアライズド・ビューおよび実表のデータを使用してリライトを行います。

```
CREATE MATERIALIZED VIEW cust_birth_mv1
ENABLE QUERY REWRITE
AS SELECT cust_last_name, cust_first_name, cust_year_of_birth
FROM customers WHERE cust_year_of_birth BETWEEN 1940 AND 1950;

CREATE MATERIALIZED VIEW cust_avg_credit_mv2
ENABLE QUERY REWRITE
AS SELECT cust_last_name, cust_first_name, cust_year_of_birth
FROM customers
WHERE cust_year_of_birth > 1950 AND cust_year_of_birth <= 1970;
```

この問合せでは、1940年から1990年に生まれたすべての顧客を必要としています。

```
SELECT cust_last_name, cust_first_name
FROM customers c WHERE cust_year_of_birth BETWEEN 1940 AND 1990;
```

クエリー・リライトは、1971年から1990年に生まれた顧客にアクセスするため、実表にアクセスする必要があります。したがって、問合せ3は次のようにリライトされます。

```
SELECT cust_last_name, cust_first_name
FROM cust_birth_mv1
```

```
UNION ALL
SELECT cust_last_name, cust_first_name
FROM cust_birth_mv2
UNION ALL
SELECT cust_last_name, cust_first_name
FROM customers c
WHERE cust_year_of_birth > 1970 AND cust_year_of_birth <= 1990;
```

問合せ4: 複数の選択列のデータの問合せ

次に、1945年から1960年に生まれ、与信限度額が1,000から10,000のすべての顧客を問い合せる問合せを考えてみます。これは、複数の選択列のデータを問い合せるため、複数選択の問合せです。

```
SELECT cust_last_name, cust_first_name
FROM customers WHERE cust_year_of_birth BETWEEN 1945 AND 1960 AND
   cust_credit_limit BETWEEN 1000 AND 10000;
```

図12-5は、2つの選択問合せを示します。これらは、次の項で説明する2つの選択マテリアライズド・ビューでリライトできます。

図12-5 複数のマテリアライズド・ビューを使用したクエリー・リライト

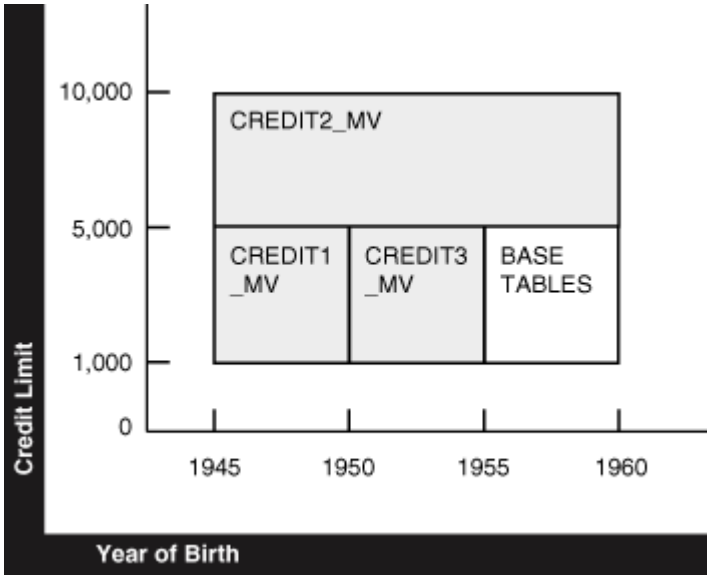


図12-5のグラフは、この問合せを満たすために使用できるマテリアライズド・ビューを表したものです。credit_mv1は、1945から1950年生まれで、与信限度額が1,000から5,000の顧客を問い合せます。credit_mv2は、1945から1960年生まれで、与信限度額が5,000より多く、10,000以下の顧客を問い合せます。credit_mv3は、1950から1955年生まれで、与信限度額が1,000から5,000の顧客を問い合せます。

この場合のマテリアライズド・ビューの定義を次に示します。

```
CREATE MATERIALIZED VIEW credit_mv1
ENABLE QUERY REWRITE
AS SELECT cust_last_name, cust_first_name,
    cust_credit_limit, cust_year_of_birth
FROM customers
WHERE cust_credit_limit BETWEEN 1000 AND 5000
AND cust_year_of_birth BETWEEN 1945 AND 1950;

CREATE MATERIALIZED VIEW credit_mv2
ENABLE QUERY REWRITE
AS SELECT cust_last_name, cust_first_name,
   cust_credit_limit, cust_year_of_birth
FROM customers
WHERE cust_credit_limit > 5000
   AND cust_credit_limit <= 10000 AND cust_year_of_birth
```

```
   BETWEEN 1945 AND 1960;

CREATE MATERIALIZED VIEW credit_mv3
ENABLE QUERY REWRITE AS
SELECT cust_last_name, cust_first_name,
   cust_credit_limit, cust_year_of_birth
FROM customers
WHERE cust_credit_limit BETWEEN 1000 AND 5000
  AND cust_year_of_birth > 1950 AND cust_year_of_birth <= 1955;
```

問合せ4は、ほとんどのデータにアクセスするこれら3つのマテリアライズド・ビューを使用してリライトできます。ただし、一部のデータはこれら3つのマテリアライズド・ビューからは取得できないため、クエリー・リライトは実表にもアクセスして、1955年から1960年生まれで、与信限度額が1,000から5,000である顧客のデータを取得します。次のようにリライトされます。

```
SELECT cust_last_name, cust_first_name
FROM credit_mv1
UNION ALL
SELECT cust_last_name, cust_first_name
FROM credit_mv2
UNION ALL
SELECT cust_last_name, cust_first_name
FROM credit_mv3
UNION ALL
SELECT cust_last_name, cust_first_name
FROM customers
WHERE cust_credit_limit BETWEEN 1000 AND 5000
  AND cust_year_of_birth > 1955 AND cust_year_of_birth <= 1960;
```

この例では、複数のマテリアライズド・ビューを使用して複数選択の問合せをリライトする方法を示しています。この例は、3つのマテリアライズド・ビュー間でデータが重複しないように単純化されています。これに対し、クエリー・リライトでは同様のリライトを実行できます。

問合せ5: 範囲および制約範囲

この例では、単一選択マテリアライズド・ビューを使用して複数選択の問合せをリライトする方法を示します。この例では、問合せに2つの範囲指定があり、マテリアライズド・ビューには1つの制約範囲指定があります。問合せでは、1945年から1960年生まれで、与信限度額が1,000から10,000の顧客を問い合せます。一方、credit_mv1では与信限度額が1,000から5,000の顧客のみを問い合せるとします。credit_mv1はcust_year_of_birthの選択述語で制約されていないので、問合せの出生年値の全範囲をカバーしています。

図12-6 制約されたマテリアライズド・ビューの選択述語

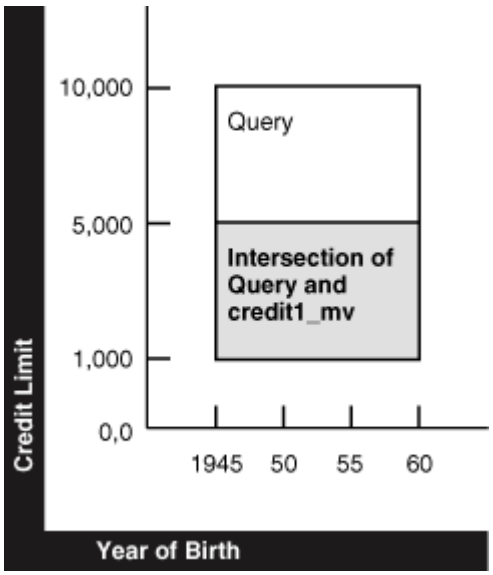


図12-6では、下部の領域がcredit_mv1のデータを表しています。

新しいcredit_mv1は、次のように定義されます。

```
CREATE MATERIALIZED VIEW credit_mv1
ENABLE QUERY REWRITE
AS SELECT cust_last_name, cust_first_name,
   cust_credit_limit, cust_year_of_birth
FROM customers WHERE cust_credit_limit BETWEEN 1000 AND 5000;
```

問合せは次のとおりです。

```
SELECT cust_last_name, cust_first_name
FROM customers WHERE cust_year_of_birth BETWEEN 1945 AND 1960
  AND cust_credit_limit BETWEEN 1000 AND 10000;
```

最終的にリライトされた問合せは、次のようになります。

```
SELECT cust_last_name, cust_first_name
FROM credit_mv1 WHERE cust_year_of_birth BETWEEN 1945 AND 1960
UNION ALL
SELECT cust_last_name, cust_first_name
FROM customers WHERE cust_year_of_brith BETWEEN 1945 AND 1960
  AND cust_credit_limit > 5000 AND cust_credit_limit <= 10000;
```

問合せ6: 単一列のINリストが指定された問合せと単一列の範囲が指定されたマテリアライズド・ビュー

複数のマテリアライズド・ビューのクエリー・リライトでは、入力問合せのINリストを処理し、同じ選択列の範囲指定を持つマテリアライズド・ビューで問合せをリライトできます。前述の範囲指定のみの例に対する自然な流れとして、INリストでは範囲内の離散値を表すものとします。

次の例では、1つの列のINリストの選択述語を持つ問合せと、1つの列の範囲選択述語を持つマテリアライズド・ビューを示します。1945、1950、1955、1960、1965、1970または1975年に生まれた、各国の顧客数を問い合せる問合せを考えてみます。この問合せは、cust_year_of_birthのINリストで制約されます。

```
SELECT c2.country_name, count(c1.country_id)
FROM customers c1, countries c2
WHERE c1.country_id = c2.country_id AND
   c1.cust_year_of_birth IN (1945, 1950, 1955, 1960, 1965, 1970, 1975)
GROUP BY c2.country_name;
```

次の2つのマテリアライズド・ビューを考えてみます。cust_country_birth_mv1では、1940年から1950年に生まれた、各国の顧客数を問い合せます。cust_country_birth_mv2では、1951年から1970年に生まれた、各国の顧客数を問い合せます。前述の問合せは、1945、1950、1955、1960、1965および1970年生まれの各国の顧客総数を取得するために、これら2つのマテリアライズド・ビューでリライトできます。1975年生まれの顧客数を取得するためには、実表にアクセスする必要があります。

この例で定義されるマテリアライズド・ビューを次に示します。

```
CREATE MATERIALIZED VIEW cust_country_birth_mv1
ENABLE QUERY REWRITE
AS SELECT c2.country_name, c1.cust_year_of_birth,
 COUNT(c1.country_id) AS count_customers
FROM customers c1, countries c2
WHERE c1.country_id = c2.country_id AND
      cust_year_of_birth BETWEEN 1940 AND 1950
GROUP BY c2.country_name, c1.cust_year_of_birth;

CREATE MATERIALIZED VIEW cust_country_birth_mv2
```

```
ENABLE QUERY REWRITE
AS SELECT c2.country_name, c1.cust_year_of_birth,
 COUNT(c1.country_id) AS count_customers
FROM customers c1, countries c2
WHERE c1.country_id = c2.country_id AND cust_year_of_birth > 1950
AND cust_year_of_birth <= 1970
GROUP BY c2.country_name, c1.cust_year_of_birth;
```

したがって、問合せ6は次のようにリライトされます。

```
SELECT v1.country_name, SUM(v1.count_customers)
FROM (SELECT country_name, SUM(count_customers) AS count_customers
FROM cust_country_birth_mv1
WHERE cust_year_of_birth IN (1945, 1950)
GROUP BY country_name
UNION ALL
SELECT country_name, SUM(count_customers) AS count_customers
FROM cust_country_birth_mv2
WHERE cust_year_of_birth IN (1955, 1960, 1965, 1970)
GROUP BY country_name
UNION ALL
SELECT c2.country_name, COUNT(c1.country_id) AS count_customers
FROM customers c1, countries c2
WHERE c1.country_id = c2.country_id AND cust_year_of_birth IN (1975)
GROUP BY c2.country_name) v1
GROUP BY v1.country_name;
```

問合せ7: 複数のマテリアライズド・ビューを使用したPCTリライト

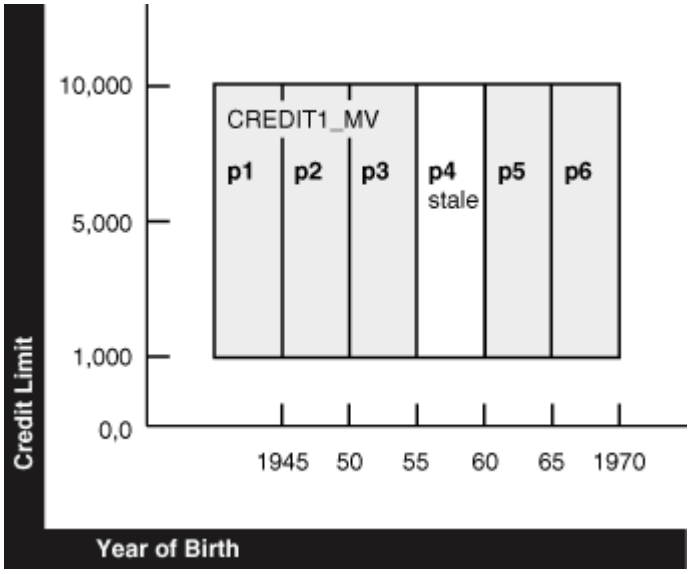
複数のマテリアライズド・ビューを使用したリライトでも、PCTリライトを利用できます。PCTリライトは、マテリアライズド・ビューの失効時にマテリアライズド・ビューの最新部分のみを使用して問合せをリライトする機能です。この機能は、ENFORCEDまたはTRUSTEDの整合性モード、および、複数のマテリアライズド・ビューによるリライトで使用され、最新データはマテリアライズド・ビューの最新部分から取得し、失効データは実表から取得できます。したがって、リライトされる問合せでは、1つ以上のマテリアライズド・ビューの最新部分のみに対してUNION ALLを行い、残りのデータを実表から取得して、結果を戻します。そのため、ここではPCTのすべての規則と条件も適用されます。マテリアライズド・ビューではPCTを有効化し、実表に対する変更は、マテリアライズド・ビューの最新部分と失効部分を明確に識別できるように実行する必要があります。

この例では、1945年から1964年生まれで、与信限度額が1,000から10,000の顧客を問い合せる問合せを考えます。また、顧客(customers)表はcust_date_of_birthでパーティション化されており、PCTを有効化したマテリアライズド・ビューcredit_mv1でも、1945年から1964年生まれで、与信限度額が1,000から10,000の顧客を問い合せるものとします。

```
SELECT cust_last_name, cust_first_name
FROM customers WHERE cust_credit_limit BETWEEN 1000 AND 10000;
```

図12-7では、実表のパーティションp1からp6に関して、マテリアライズド・ビューの最新部分および失効部分をそれぞれグレーと白で示しています。

図12-7 PCTリライトと複数のマテリアライズド・ビューによるリライト



ここでは、ENFORCEDモードで、顧客表のp1、p2、p3、p5およびp6が最新部分、パーティションp4が失効部分だとします。これは、問合せに回答するのにcredit_mv1のすべてのパーティションが使用できないことを意味します。リライトされる問合せでは、顧客パーティションp4の結果を他の特定のマテリアライズド・ビューから取得するか、この例で示すように、実表から取得する必要があります。次に、表がどのようにパーティション化されているのかがわかるcustomers表の表定義の一部を示します。

```
CREATE TABLE customers
(PARTITION BY RANGE (cust_year_of_birth)
 PARTITION p1 VALUES LESS THAN (1945),
 PARTITION p2 VALUES LESS THAN (1950),
 PARTITION p3 VALUES LESS THAN (1955),
 PARTITION p4 VALUES LESS THAN (1960),
 PARTITION p5 VALUES LESS THAN (1965),
 PARTITION p6 VALUES LESS THAN (1970);
```

次に、前述の例のマテリアライズド・ビューの定義を示します。

```
CREATE MATERIALIZED VIEW credit_mv1
ENABLE QUERY REWRITE
AS SELECT cust_last_name, cust_first_name,
   cust_credit_limit, cust_year_of_birth
FROM customers
WHERE cust_credit_limit BETWEEN 1000 AND 10000
AND cust_year_of_birth BETWEEN 1945 AND 1964;
```

このマテリアライズド・ビューでは、表customersに関してPCTが有効化されています。

リライトされた問合せは、次のようになります。

```
SELECT cust_last_name, cust_first_name FROM credit_mv1
WHERE cust_credit_limit BETWEEN 1000 AND 10000 AND
   (cust_year_of_birth >= 1945 AND cust_year_of_birth < 1955 OR
    cust_year_of_birth BETWEEN 1945 AND 1964)
UNION ALL
SELECT cust_last_name, cust_first_name
FROM customers WHERE cust_credit_limit BETWEEN 1000 AND 10000
  AND cust_year_of_birth < 1960 AND cust_year_of_birth >= 1955;
```

12.3 その他のクエリー・リライトの考慮事項

次のトピックでは、クエリー・リライトが可能なその他のケースについて説明します。

- ネステッド・マテリアライズド・ビューを使用したクエリー・リライトについて

- インライン・ビューがある場合のクエリー・リライトについて
- リモート表を使用したクエリー・リライトについて
- 表複製がある場合のクエリー・リライトについて
- デート・フォールディングを使用したクエリー・リライトについて
- ビューの制約を使用したクエリー・リライトについて
- 集合演算子を含むマテリアライズド・ビューを使用したクエリー・リライト
- グルーピング・セットがある場合のクエリー・リライトについて
- ウィンドウ関数がある場合のクエリー・リライト
- クエリー・リライトおよび式の一致
- クエリー・リライトでのカーソルの共有とバインド変数
- クエリー・リライトでの式の処理

12.3.1 ネステッド・マテリアライズド・ビューを使用したクエリー・リライトについて

クエリー・リライトでは、ネステッド・マテリアライズド・ビューが繰り返し利用されます。Oracle Databaseでは、まず、集計および結合を持つマテリアライズド・ビューを使用して、クエリー・リライトを試みます。次に、結合のみを含むマテリアライズド・ビューを使用して試みます。いずれかのリライトが成功した場合、Oracleはリライトがなくなるまで、そのプロセスを繰り返します。たとえば、次のようにマテリアライズド・ビューjoin_sales_time_product_mvとsum_sales_time_product_mvを作成したとします。

```
CREATE MATERIALIZED VIEW join_sales_time_product_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_id, p.prod_name, t.time_id, t.week_ending_day,
       s.channel_id, s.promo_id, s.cust_id, s.amount_sold
FROM   sales s, products p, times t
WHERE  s.time_id=t.time_id AND s.prod_id = p.prod_id;

CREATE MATERIALIZED VIEW sum_sales_time_product_mv
ENABLE QUERY REWRITE AS
SELECT mv.prod_name, mv.week_ending_day, COUNT(*) cnt_all,
       SUM(mv.amount_sold) sum_amount_sold,
       COUNT(mv.amount_sold) cnt_amount_sold
FROM join_sales_time_product_mv mv
GROUP BY mv.prod_name, mv.week_ending_day;
```

次の問合せを考えてみます。

```
SELECT p.prod_name, t.week_ending_day, SUM(s.amount_sold)
FROM sales s, products p, times t
WHERE s.time_id=t.time_id AND s.prod_id=p.prod_id
GROUP BY p.prod_name, t.week_ending_day;
```

Oracleは、join_sales_time_product_mvがリライトに適していると判断します。リライトされた問合せの形式は、次のとおりです。

```
SELECT mv.prod_name, mv.week_ending_day, SUM(mv.amount_sold)
FROM join_sales_time_product_mv mv
GROUP BY mv.prod_name, mv.week_ending_day;
```

リライトが発生したため、Oracleはこのプロセスを再度試みます。ここでは、前述の問合せは、単一表集計マテリアライズド・ビューsum_sales_store_timeを使用して、次の形式にリライトされます。

```
SELECT mv.prod_name, mv.week_ending_day, mv.sum_amount_sold
FROM sum_sales_time_product_mv mv;
```

12.3.2 インライン・ビューがある場合のクエリー・リライトについて

Oracle Databaseでは、次の2つの場合にインライン・ビューによるクエリー・リライトをサポートしています。

- マテリアライズド・ビューに含まれるインライン・ビューのテキストが問合せのテキストと完全に一致している場合
- 問合せの中に含まれるインライン・ビューが、マテリアライズド・ビューに含まれるインライン・ビューと同等である場合

2つのインライン・ビューが同等であるとみなされるのは、それぞれのSELECT構文のリストおよびGROUP BYリストが同等であり、FROM句に同一または同等のオブジェクトが含まれ、WHERE句の中のすべての選択述語を含む結合グラフが同等であり、さらにHAVING句が同等である場合です。

次の例は、インライン・ビューを持つ問合せに対し、テキスト一致リライトおよび一般的なインライン・ビュー・リライトを使用したマテリアライズド・ビューによるリライトがどのように行われるかを示します。インライン・ビューを含む次のマテリアライズド・ビューを考えてみます。

```
CREATE MATERIALIZED VIEW SUM_SALES_MV
ENABLE QUERY REWRITE AS
SELECT mv_iv.prod_id, mv_iv.cust_id,
sum(mv_iv.amount_sold) sum_amount_sold
FROM (SELECT prod_id, cust_id, amount_sold
FROM sales, products
WHERE sales.prod_id = products.prod_id) MV_IV
GROUP BY mv_iv.prod_id, mv_iv.cust_id;
```

次の問合せに含まれるインライン・ビューのテキストは、マテリアライズド・ビューのインライン・ビューのテキストと完全に一致しています。そのためこの問合せでは、リライトが行えるように、インライン・ビューが内部的にマテリアライズド・ビューのインライン・ビューと置き換えられます。

```
SELECT iv.prod_id, iv.cust_id,
SUM(iv.amount_sold) sum_amount_sold
FROM (SELECT prod_id, cust_id, amount_sold
FROM sales, products
WHERE sales.prod_id = products.prod_id) IV
GROUP BY iv.prod_id, iv.cust_id;
```

次の問合せに含まれるインライン・ビューのテキストは、前述のマテリアライズド・ビューに含まれるインライン・ビューのテキストと完全には一致していません。問合せのインライン・ビューの中の結合述語が置き換えられていることに注意してください。この問合せのテキストは、マテリアライズド・ビューに含まれるインライン・ビューのテキストに一致していませんが、クエリー・リライトでは、この問合せのインライン・ビューが、マテリアライズド・ビューに含まれるインライン・ビューと同等であると判断されます。ここでも同様に、この問合せに対してリライトが行えるよう、問合せのインライン・ビューが内部的にマテリアライズド・ビューのインライン・ビューと置き換えられます。

```
SELECT iv.prod_id, iv.cust_id,
SUM(iv.amount_sold) sum_amount_sold
FROM (SELECT prod_id, cust_id, amount_sold
FROM sales, products
WHERE products.prod_id = sales.prod_id) IV
GROUP BY iv.prod_id, iv.cust_id;
```

これらの問合せはいずれも、次のようにSUM_SALES_MVでリライトされます。

```
SELECT prod_id, cust_id, sum_amount_sold
FROM SUM_SALES_MV;
```

集合演算子、GROUPING SET句、ネストした副問合せ、ネストしたインライン・ビュー、およびリモート表を含む問合せでは、一般的なインライン・ビュー・リライトはサポートされていません。

12.3.3 リモート表を使用したクエリー・リライトについて

Oracle Databaseは、単一のリモート・データベース・サイトの表を参照するマテリアライズド・ビューでのクエリー・リライトをサポートしています。ただしマテリアライズド・ビューは、問合せの発行が行われているサイトに存在する必要があります。リモート表の更新内容は、即座にはローカル・サイトに伝播されないため、クエリー・リライトはstale_toleratedモードでのみ実行されます。マテリアライズド・ビューにはない列が含まれている問合せでは、後戻り結合という手法を使用してクエリー・リライトが行われます。ただし、後戻り結合表がローカル・サイトにない場合は、クエリー・リライトは実行されません。また、リモート表の制約情報はリモート・サイトでは使用できないため、クエリー・リライトでは制約情報は使用されません。

次の問合せには、単一のリモート・サイトにある表が含まれています。

```
SELECT p.prod_id, t.week_ending_day, s.cust_id,
       SUM(s.amount_sold) AS sum_amount_sold
FROM sales@remotedbl s, products@remotedbl p, times@remotedbl t
WHERE s.time_id=t.time_id AND s.prod_id=p.prod_id
GROUP BY p.prod_id, t.week_ending_day, s.cust_id;
```

次のマテリアライズド・ビューはローカル・サイトにありますが、参照する表はすべてリモート・サイトにあります。

```
CREATE MATERIALIZED VIEW sum_sales_prod_week_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_id, t.week_ending_day, s.cust_id,
       SUM(s.amount_sold) AS sum_amount_sold
FROM sales@remotedbl s, products@remotedbl p, times@remotedbl t
WHERE s.time_id=t.time_id AND s.prod_id=p.prod_id
GROUP BY p.prod_id, t.week_ending_day, s.cust_id;
```

この問合せはリモート表を参照しますが、そのリライトは次のように前述のマテリアライズド・ビューを使用して行われます。

```
SELECT prod_id, week_ending_day, cust_id,  sum_amount_sold
FROM sum_sales_prod_week_mv;
```

12.3.4 表複製がある場合のクエリー・リライトについて

同じ表に対する複数の参照または自己結合を含む問合せのクエリー・リライトを行うには、2つの異なる方法があります。第1の方法では、問合せとマテリアライズド・ビューの定義で、表に対する複数の参照の別名が同じであるようにする必要があります。別名が異なる場合は、Oracleは第2の方法を試みます。この方法では、問合せの複数の参照がマテリアライズド・ビューの複数の参照に一致するように、問合せの結合とマテリアライズド・ビューの結合が比較されます。

マテリアライズド・ビューと問合せの例を次に示します。この例では、問合せには表の列の参照がないため、テキストの完全一致は機能しません。ただし、表の参照の別名は一致するため、一般的なクエリー・リライトは可能です。

shサンプル・スキーマで自己結合のリライトの可能性を示すために、ファクト表にある実際の出荷日と支払日を含むように、次のような追加が行われ、同じディメンション表timesを参照しているとします。これはあくまでも例であり、結果は戻されません。

```
ALTER TABLE sales ADD (time_id_ship DATE);
ALTER TABLE sales ADD (CONSTRAINT time_id_book_fk FOREIGN key (time_id_ship)
 REFERENCES times(time_id) ENABLE NOVALIDATE);
ALTER TABLE sales MODIFY CONSTRAINT time_id_book_fk RELY;
ALTER TABLE sales ADD (time_id_paid DATE);
ALTER TABLE sales ADD (CONSTRAINT time_id_paid_fk FOREIGN KEY (time_id_paid)
 REFERENCES times(time_id) ENABLE NOVALIDATE);
ALTER TABLE sales MODIFY CONSTRAINT time_id_paid_fk RELY;
```

これで、マテリアライズド・ビューを次のように定義できます。

```
CREATE MATERIALIZED VIEW sales_shipping_lag_mv
ENABLE QUERY REWRITE AS
SELECT t1.fiscal_week_number, s.prod_id,
         t2.fiscal_week_number - t1.fiscal_week_number AS lag
FROM times t1, sales s, times t2
WHERE t1.time_id = s.time_id AND t2.time_id = s.time_id_ship;
```

次の問合せはテキストの完全一致テストにはパスしませんが、表の別名が一致するためリライトされます。

```
SELECT s.prod_id, t2.fiscal_week_number - t1.fiscal_week_number AS lag
FROM times t1, sales s, times t2
WHERE t1.time_id = s.time_id AND t2.time_id = s.time_id_ship;
```

Oracle Databaseでは、問合せ内で複数インスタンス化された表のインスタンスが、マテリアライズド・ビュー内の対応する表のインスタンスと正確に一致することを保証するために、他のチェックを実行することに注意してください。たとえば、次の例では、表timesの複数インスタンスで使用される別名は、マテリアライズド・ビューの表timesの複数インスタンスと一致しないと判断されます。

次の問合せでは、複数インスタンス化された表timeの別名は一致しますが、t2で別名化されたtimeのインスタンス間の結合条件とは互換性がないため、sales_shipping_lag_mvを使用してリライトできません。

```
SELECT s.prod_id, t2.fiscal_week_number - t1.fiscal_week_number AS lag
FROM times t1, sales s, times t2
WHERE t1.time_id = s.time_id AND t2.time_id = s.time_id_paid;
```

この問合せは、t2で別名化されたtime表のインスタンスをs.time_id_paid列で結合しますが、マテリアライズド・ビューは、t2で別名化されたtime表のインスタンスをs.time_id_ship列で結合します。結合条件が異なるため、Oracleではリライトできないことが適切に判断されます。

次の問合せでは、表timesのマテリアライズド・ビューsales_shipping_lag_mvにおいて一致する別名がありません。ただし、クエリー・リライトによって問合せとマテリアライズド・ビューの結合が比較され、timesの複数インスタンスと正しく一致します。

```
SELECT s.prod_id, x2.fiscal_week_number - x1.fiscal_week_number AS lag
FROM times x1, sales s, times x2
WHERE x1.time_id = s.time_id AND x2.time_id = s.time_id_ship;
```

12.3.5 デート・フォールディングを使用したクエリー・リライトについて

デート・フォールディングによるリライトは、式の一致によるリライトの特別な形式です。このリライトでは、問合せの日付範囲が、より大きい単位の日付で表される同等の日付範囲にフォールドされます。フォールドされた日付範囲にある大きい単位の日付で表された式は、マテリアライズド・ビューの等価の式と一致します。より大きい単位(月、四半期、年など)の日付への日付範囲のフォールドは、列の基礎となるデータ型がOracle DATEの場合に実行されます。式の一致は、式の標準形の使用をベースに実行されます。

DATEは組込みデータ型で、秒、日、月などの順序付けられた時間単位を表し、時間階層(秒->分->時間->日->月->四半期->年)が組み込まれています。DATEについてのハードコード化されたこの情報が、小さい単位の日付から大きい単位の日付に日付範囲をフォールドするときに使用されます。つまり、日付値は、月、四半期、年の初め、または月、四半期、年の終わりにフォールドできます。たとえば、日付値1-jan-1999は、年1999、四半期1999-1または月1999-01のいずれかの最初にフォールドできます。また、日付値30-sep-1999は、四半期1999-03または月1999-09のいずれかの最後にフォールドできます。

ノート:

デート・フォールディングのしくみにより、BETWEEN および日付列を使用する際には注意が必要です。BETWEEN および日付列を使用するには、後の日付を 1 日増やすのが最善の方法です。つまり、date_col BETWEEN '1-jan-1999' AND '30-jun-1999'を使用するかわりに、date_col BETWEEN '1-jan-1999' AND '1-jul-1999'を使用してください。また、TRUNC(date_col) BETWEEN '1-jan-1999' AND '30-jun-1999'のように、TRUNC 関数を使用しても同等の結果を取得できます。ただし、TRUNC では、時間値が取り除かれます。

日付値は順序付けされているため、日付範囲の方がより大きい単位の整数を表している場合、日付列に対して指定されている範囲述語は小さい単位から大きい単位にフォールドできます。たとえば、範囲述語date_col >= '1-jan-1999' AND date_col < '30-jun-1999'は、日付値から特定の日付コンポーネントを抽出するTO_CHAR関数を使用して、月範囲または四半期範囲にフォールドできます。

日付値のフォールドによってデータを集計するメリットは、データを圧縮できることです。デート・フォールディングが実行されない場合、データは最小単位で集計されます。その結果、データの格納に必要なディスク領域が多くなり、マテリアライズド・ビューをスキャンするためのI/Oも増加します。

1998年の製品別の売上合計を問い合せる問合せを考えてみます。

```
SELECT p.prod_category, SUM(s.amount_sold)
FROM sales s, products p
WHERE s.prod_id=p.prod_id AND s.time_id >= TO_DATE('01-jan-1998', 'dd-mon-yyyy')
  AND s.time_id <   TO_DATE('01-jan-1999', 'dd-mon-yyyy')
GROUP BY p.prod_category;

CREATE MATERIALIZED VIEW sum_sales_pcat_monthly_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_category, TO_CHAR(s.time_id,'YYYY-MM') AS month,
       SUM(s.amount_sold) AS sum_amount
FROM sales s, products p
WHERE s.prod_id=p.prod_id
GROUP BY p.prod_category, TO_CHAR(s.time_id, 'YYYY-MM');

SELECT p.prod_category, SUM(s.amount_sold)
FROM sales s, products p
WHERE s.prod_id=p.prod_id
AND TO_CHAR(s.time_id, 'YYYY-MM') >= '01-jan-1998'
AND TO_CHAR(s.time_id, 'YYYY-MM') < '01-jan-1999'
GROUP BY p.prod_category;

SELECT mv.prod_category, mv.sum_amount
FROM sum_sales_pcat_monthly_mv mv
WHERE month >= '01-jan-1998' AND month < '01-jan-1999';
```

問合せに指定されている範囲は、年、四半期または月の整数を表します。prod_typeごとに事前集計された売上を含むマテリアライズド・ビューmv3が次のように定義されているとします。

```
CREATE MATERIALIZED VIEW mv3
ENABLE QUERY REWRITE AS
SELECT prod_name, TO_CHAR(sales.time_id,'yyyy-mm')
   AS month, SUM(amount_sold) AS sum_sales
FROM sales, products WHERE sales.prod_id = products.prod_id
GROUP BY prod_name, TO_CHAR(sales_time_id, 'yyyy-mm');
```

問合せは、まず日付範囲を月単位にフォールドし、次に月を表す式をmv3の月式と一致させることでリライトされます。このリライトを、次の2つのステップ(日付範囲のフォールドおよび実際のリライト)で示します。

```
SELECT prod_name, SUM(amount_sold) AS sum_sales
FROM sales, products
WHERE sales.prod_id = products.prod_id AND TO_CHAR(sales.time_id, 'yyyy-mm') >=
      TO_CHAR('01-jan-1998', 'yyyy-mm') AND TO_CHAR(sales.time_id, '01-jan-1999',
 'yyyy-mm') < TO_CHAR(TO_DATE(''01-jan-1999'', ''dd-mon-yyyy''), ''yyyy-mm'')
GROUP BY prod_name;

SELECT prod_name, sum_sales
FROM mv3 WHERE month >=
      TO_CHAR(TO_DATE('01-jan-1998', 'dd-mon-yyyy'), 'yyyy-mm')
  AND month < TO_CHAR(TO_DATE('01-jan-1999', 'dd-mon-yyyy'), 'yyyy-mm');
```

mv3の事前集計済の売上がprod_nameと月ではなくprod_nameと年に基づいたものであれば、問合せは日付範囲を年単位にフォールドして年の式と一致させることでリライトされます。

12.3.6 ビューの制約を使用したクエリー・リライトについて

データ・ウェアハウス・アプリケーションでは、リレーショナル・スキーマの整合性制約を識別することにより、データベースの多次元キューブが認識されます。整合性制約は、ファクト表とディメンション表間の主キーと外部キーの関係を表します。アプリケーションは、データ・ディクショナリを問い合せれば整合性制約とデータベース内のキューブを認識できます。ただし、スキーマの複雑さとセキュリティ上の理由から、データベース管理者がファクト表とディメンション表のビューを定義する環境では、これは機能しません。このような環境では、アプリケーションはキューブを正しく識別できません。ビュー間で制約の定義を可能にすると、実表の制約をビューにも適用し、制限付きの環境でもアプリケーションにキューブを認識させることができます。

ビューの制約の定義は宣言の性質を持っていますが、ビューの操作には実表に定義された整合性制約が適用され、ビューに対する制約は実表に対する制約を通じて実施できます。データの正確性と正当性のみでなく、元のベース・オブジェクトを使用したマテリアライズド・ビューのクエリー・リライトのためにも、実表の制約を定義する必要があります。

関連項目:

ビューの制約の制限について

マテリアライズド・ビューのリライトでは、クエリー・リライトのために制約が広範囲に使用されます。これは、可逆式結合の判断と、マテリアライズド・ビューの結合に問合せの結合との互換性があるかどうかの判断、リライトが可能かどうかの判断に使用されます。

ビューに対する制約の有効な状態は、DISABLE NOVALIDATEのみです。ただし、ビューの制約の状態としてRELYまたはNORELYを選択すると、より洗練されたクエリー・リライトが使用可能になります。たとえば、RELY状態のビューの制約では、問合せの整合性レベルがTRUSTEDに設定されている場合に、問合せをリライトできます。表12-3に、ビューの制約が可逆式結合の判断に使用される場合を示します。

ビューの制約は、問合せの整合性レベルがENFORCEDの場合には使用できないため注意してください。このレベルでは、最大レベルのENABLE VALIDATEが施行されます。

表12-3 ビューの制約とリライトの整合性モード

制約の状態	RELY	NORELY
ENFORCED	不可	不可
TRUSTED	可能	不可

制約の状態	RELY	NORELY
STALE_TOLERATED	はい	不可

例12-10 ビューの制約

ビューのリライト機能を理解するには、shサンプル・スキーマを次のように拡張する必要があります。

```
CREATE VIEW time_view AS
SELECT time_id, TO_NUMBER(TO_CHAR(time_id, 'ddd')) AS day_in_year FROM times;
```

これで、ビューとファクト表間の外部キーと主キーの関係を(RELYモードで)設定できます。次の制約を追加すると、リライトは表12-3のようになります。たとえば、リライトはTRUSTEDモードで機能します。

```
ALTER VIEW time_view ADD (CONSTRAINT time_view_pk
   PRIMARY KEY (time_id) DISABLE NOVALIDATE);
ALTER VIEW time_view MODIFY CONSTRAINT time_view_pk RELY;
ALTER TABLE sales ADD (CONSTRAINT time_view_fk FOREIGN KEY (time_id)
   REFERENCES time_view(time_id) DISABLE NOVALIDATE);
ALTER TABLE sales MODIFY CONSTRAINT time_view_fk RELY;
```

次のマテリアライズド・ビューの定義を考えてみます。

```
CREATE MATERIALIZED VIEW sales_pcat_cal_day_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_category, t.day_in_year, SUM(s.amount_sold) AS sum_amount_sold
FROM time_view t, sales s, products p
WHERE t.time_id = s.time_id AND p.prod_id = s.prod_id
GROUP BY p.prod_category, t.day_in_year;
```

次の問合せも、ディメンション表productsが省略されていますが、主キーと外部キーの関係なしでリライトされます。これは、salesとproducts間の結合が可逆式結合であるためです。

```
SELECT t.day_in_year, SUM(s.amount_sold) AS sum_amount_sold
FROM time_view t, sales s WHERE t.time_id = s.time_id
GROUP BY t.day_in_year;
```

ただし、マテリアライズド・ビューsales_pcat_cal_day_mvがビューtime_viewでのみ定義されている場合、次の問合せはsalesとtime_viewの間の結合が明記されていないためリライトできません。これは、マテリアライズド・ビューのデルタ結合の可逆性を示せないことによります。前述のように制約が追加されていれば、この問合せもリライト可能です。

```
SELECT p.prod_category, SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p WHERE p.prod_id = s.prod_id
GROUP BY p.prod_category;
```

shスキーマに加えた変更を元に戻すには、次の文を発行します。

```
ALTER TABLE sales DROP CONSTRAINT time_view_fk;
DROP VIEW time_view;
```

12.3.6.1 ビューの制約の制限について

参照整合性制約定義にビューが関係する場合、つまり、ビューに外部キーまたは参照されるキーがある場合、制約に指定できるモードはDISABLE NOVALIDATEのみです。

ビューのRELY制約が許可されるのは、DISABLE NOVALIDATEモードで参照される一意または主キー制約もRELY制約である場合のみです。

参照整合性制約に対応付けるON DELETEアクションは指定できません(DELETE CASCADEなど)。ただし、ビューの制約はDISABLE NOVALIDATEモードであるため、ビューおよびその実表に対するDELETE、UPDATEおよびINSERT操作は許可されます。

12.3.7 ハイブリッド・パーティション表がある場合のクエリー・リライトについて

クエリー・リライトでは、ハイブリッド・パーティション表内の外部パーティションの最新状態がUNKNOWNであるとみなします。したがって、1つ以上の外部パーティションのデータを要求する問合せは、TRUSTEDまたはSTALE_TOLERATED整合性モードでのみリライトできます。

ハイブリッド・パーティション表に基づくマテリアライズド・ビューのSELECTリストにパーティション・キーまたはパーティション・マーカーが含まれる場合、そのマテリアライズド・ビューはパーティション追跡の対象になります。PCTが有効化されていないハイブリッド・パーティション表に基づくマテリアライズド・ビューの場合、使用可能な整合性モードはSTALE_TOLERATEDのみです。

ハイブリッド・パーティション表がレンジ・パーティション表またはリスト・パーティション表である場合のみ、ハイブリッド・パーティション表に対する問合せを、ENFORCEDおよびTRUSTED整合性モードでPCTリライトを使用してリライトできます。

例12-11 ハイブリッド・パーティション表に基づくクエリー・リライトおよびマテリアライズド・ビュー

ハイブリッド・パーティション表hybrid_salesでは、ENFORCED整合性モードが使用されます。内部パーティションの1つが失効しています。

次の問合せが実行されます。

```
SELECT customer_no, sum(price) as sum_price
FROM hybrid_sales WHERE
  time_id > TO_DATE( '01-01-1950' ) and time_id < TO_DATE( '06-01-2001' )
GROUP BY customer_no;
```

この問合せは、ハイブリッド・パーティション表を使用するようにリライトできます。PCTリライトは、マテリアライズド・ビューから最新のパーティションを選択し、すべての失効したパーティションおよび外部パーティションを実表から直接選択します。リライトされた問合せは、次のようになります。

```
SELECT v1.customer_no, SUM(v1.total_price) sum_price
FROM
  (SELECT customer_no, SUM(total_price) FROM Hybrid_sales WHERE
    time_id > TO_DATE( '01-01-1950' ) and time_id < TO_DATE( '01-01-2000' )
   GROUP BY customer_no
   UNION ALL
   SELECT customer_no, SUM(total_price) FROM HyPT_MV WHERE
    time_id > TO_DATE( '01-01-2000' ) and time_id < TO_DATE( '01-01-2001' )
   GROUP BY customer_no
   UNION ALL
   SELECT customer_no, SUM(total_price) FROM Hybrid_sales WHERE
    time_id > TO_DATE( '01-01-2001' ) and time_id < TO_DATE( '06-01-2001' )
   GROUP BY customer_no
) v1
GROUP BY v1.customer_no;
```

12.3.8 集合演算子を含むマテリアライズド・ビューを使用したクエリー・リライト

集合演算子を含むマテリアライズド・ビューでクエリー・リライトを使用できます。この場合、リライトを行うために、問合せとマテリアライズド・ビューが文字どおりに一致している必要はありません。たとえば、次のマテリアライズド・ビューを考えてみます。このマテリアライズド・ビューでは、San FranciscoまたはLos Angelesの男性顧客の郵便番号を使用します。

```
CREATE MATERIALIZED VIEW cust_male_postal_mv
ENABLE QUERY REWRITE AS
SELECT c.cust_city, c.cust_postal_code
```

```
FROM customers c
WHERE c.cust_gender = 'M' AND c.cust_city = 'San Francisco'
UNION ALL
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_gender = 'M' AND c.cust_city = 'Los Angeles';
```

次の問合せを行うと、San FranciscoまたはLos Angelesの男性顧客の郵便番号が表示されます。

```
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_city = 'Los Angeles' AND c.cust_gender = 'M'
UNION ALL
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_city = 'San Francisco' AND c.cust_gender = 'M';
```

リライトされた問合せは次のようになります。

```
SELECT mv.cust_city, mv.cust_postal_code
FROM cust_male_postal_mv mv;
```

リライトされた問合せでは、UNION ALLを削除し、マテリアライズド・ビューに置き換えています。通常、クエリー・リライトでは、既存の一般的な適性ルールを使用して、UNION ALLの下位にあるSELECT副次選択が、問合せとマテリアライズド・ビューで同等かどうかを判断する必要があります。

UNION ALLマーカーとクエリー・リライトを参照してください。

たとえば、San Francisco、PalmdaleまたはLos Angelesの男性顧客の郵便番号を取得する問合せの場合、前述の例と同じリライトが可能ですが、クエリー・リライトでは、次のように実表でUNION ALLを保持する必要があります。

```
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_city= 'Palmdale' AND c.cust_gender ='M'
UNION ALL
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_city = 'Los Angeles' AND c.cust_gender = 'M'
UNION ALL
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_city = 'San Francisco' AND c.cust_gender = 'M';
```

リライトされた問合せは、次のようになります。

```
SELECT mv.cust_city, mv.cust_postal_code
FROM cust_male_postal_mv mv
UNION ALL
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_city = 'Palmdale' AND c.cust_gender = 'M';
```

これにより、マテリアライズド・ビューcust_male_postal_mvを使用して、UNION ALLのサブセットをリライトできるケースがクエリー・リライトで検出されます。

UNION、UNION ALLおよびINTERSECTは可換であるため、問合せまたはマテリアライズド・ビューで副次選択が出現する順序に関係なく、クエリー・リライトでリライトできます。ただし、MINUSは可換ではありません。たとえば、A MINUS Bは、B MINUS Aと同等ではありません。したがって、問合せおよびマテリアライズド・ビューにおいてMINUS演算子の副次選択の出現順序は、リライト

する順序と同じである必要があります。たとえば、customer_oldという古いバージョンの顧客表があり、London在住の男性顧客に限定した現行の顧客表との相違を探すとします。つまり、古い顧客表にはなく、現行の顧客表にあるそうした顧客を探します。MINUS演算子を使用して、これを行う方法の例を次に示します。

```
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_city= 'Los Angeles' AND c.cust_gender = 'M'
MINUS
SELECT c.cust_city, c.cust_postal_code
FROM customers_old c
WHERE c.cust_city = 'Los Angeles' AND c.cust_gender = 'M';
```

副次選択の順序を変更すると、異なる回答になります。これは、MINUSが可換でないことを示します。

12.3.8.1 UNION ALLマーカーとクエリー・リライト

マテリアライズド・ビューにUNION ALL演算子が1つ以上ある場合、UNION ALLマーカーを含めることができます。UNION ALLマーカーは、マテリアライズド・ビューの各行が影響を受けているUNION ALL副次選択を識別するために使用します。クエリー・リライトでは、このマーカーを使用して、マテリアライズド・ビューのどの行が特定のUNION ALL副次選択に属しているかを識別できます。これは、問合せがマテリアライズド・ビューのデータのサブセットのみを必要とする場合、または問合せの副次選択が文字どおりマテリアライズド・ビューの副次選択と一致しない場合に有効です。例として、San Franciscoの男性顧客およびLos Angelesの女性顧客の郵便番号を取得する問合せを次に示します。

```
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_gender = 'M' and c.cust_city = 'San Francisco'
UNION ALL
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_gender = 'F' and c.cust_city = 'Los Angeles';
```

問合せの回答は、次のマテリアライズド・ビューを使用して取得できます。

```
CREATE MATERIALIZED VIEW cust_postal_mv
ENABLE QUERY REWRITE AS
SELECT 1 AS marker, c.cust_gender, c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_city = 'Los Angeles'
UNION ALL
SELECT 2 AS marker, c.cust_gender, c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_city = 'San Francisco';
```

リライトされた問合せは、次のようになります。

```
SELECT mv.cust_city, mv.cust_postal_code
FROM cust_postal_mv mv
WHERE mv.marker = 2 AND mv.cust_gender = 'M'
UNION ALL
SELECT mv.cust_city, mv.cust_postal_code
FROM cust_postal_mv mv
WHERE mv.marker = 1 AND mv.cust_gender = 'F';
```

最初の副次選択のWHERE句には、mv.marker = 2およびmv.cust_gender = 'M'があります。これらは、UNION ALLの2番目の副次選択で男性顧客を表す行のみを選択します。2番目の副次選択のWHERE句には、mv.marker = 1およびmv.cust_gender = 'F'があります。これらは、UNION ALLの最初の副次選択で女性顧客を表す行のみを選択します。クエリー・リライトでは、重複行または個別行を削除する集合演算子は使用できないことに注意してください。たとえば、UNIONでは

重複が削除されるため、クエリー・リライトでは削除された行を識別できません。次に例を示します。

```
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_city= 'Palmdale' AND c.cust_gender ='M'
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_gender = 'M' and c.cust_city = 'San Francisco'
UNION ALL
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_gender = 'F' and c.cust_city = 'Los Angeles';
```

UNION ALLマーカーを使用してリライトされた問合せは、次のようになります。

```
SELECT c.cust_city, c.cust_postal_code
FROM customers c
WHERE c.cust_city= 'Palmdale' AND c.cust_gender ='M'
UNION ALL
SELECT mv.cust_city, mv.cust_postal_code

FROM cust_postal_mv mv
WHERE mv.marker = 2 AND mv.cust_gender = 'M'
UNION ALL
  SELECT mv.cust_city, mv.cust_postal_code
  FROM cust_postal_mv mv
  WHERE mv.marker = 1 AND mv.cust_gender = 'F';
```

マーカーを使用する場合に準拠する必要があるルールを次に示します。

- 定数または文字列を指定し、すべてのUNION ALL副次選択と同じデータ型にする必要があります。
- 定数を指定し、UNION ALL副次選択ごとに別個の値にします。複数の副次選択で同じ値は再使用できません。
- すべての副次選択に対して同じ順序の場所に指定する必要があります。

12.3.9 グルーピング・セットがある場合のクエリー・リライトについて

この項では、グルーピング・セットがある場合のクエリー・リライトの使用について、次の考慮すべき事柄を説明します。

- GROUP BY拡張機能を使用したクエリー・リライトについて
- 拡張GROUP BYを持つ問合せをリライトするためのヒント

12.3.9.1 GROUP BY拡張機能を使用したクエリー・リライトについて

GROUP BY句の拡張機能としてGROUPING SETS、CUBE、ROLLUPおよびこれらの連結が使用できます。これらの拡張機能により、必要なグルーピングを問合せのGROUP BY句で選択的に指定できます。たとえば、次の例は、グルーピング・セットを使用した一般的な問合せです。

```
SELECT p.prod_subcategory, t.calendar_month_desc, c.cust_city,
  SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, customers c, products p, times t
WHERE s.time_id=t.time_id AND s.prod_id = p.prod_id AND s.cust_id = c.cust_id
GROUP BY GROUPING SETS ((p.prod_subcategory, t.calendar_month_desc),
   (c.cust_city, p.prod_subcategory));
```

GROUP BY拡張機能を持つ問合せに対するベース・グルーピングという語は、GROUP BY句に存在するすべての一意式を意味します。前述の問合せでは、グルーピング(p.prod_subcategory, t.calendar_month_desc, c.cust_city)が

ベース・グルーピングです。

拡張機能は、ユーザーの問合せやマテリアライズド・ビューを定義する問合せに使用できます。いずれの場合も、マテリアライズド・ビューのリライトが適用され、リライトの機能を次の使用例に分けて識別できます。

- マテリアライズド・ビューが単純GROUP BYを持ち、問合せが拡張GROUP BYを持つ場合
- マテリアライズド・ビューが拡張GROUP BYを持ち、問合せが単純GROUP BYを持つ場合
- マテリアライズド・ビューと問合せの両方が拡張GROUP BYを持つ場合

12.3.9.1.1 マテリアライズド・ビューが単純GROUP BYを持ち、問合せが拡張GROUP BYを持つ場合

問合せに拡張GROUP BY句が含まれている場合、ベース・グルーピングがOracleによるクエリー・リライト条件で説明したリライト・ルールにリストされているマテリアライズド・ビューを使用してリライトできる場合は、マテリアライズド・ビューでリライトできます。たとえば、次の問合せを考えてみます。

```
SELECT p.prod_subcategory, t.calendar_month_desc, c.cust_city,
  SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, customers c, products p, times t
WHERE s.time_id=t.time_id AND s.prod_id = p.prod_id AND s.cust_id = c.cust_id
GROUP BY GROUPING SETS
((p.prod_subcategory, t.calendar_month_desc),
   (c.cust_city, p.prod_subcategory));
```

ベース・グルーピングは(p.prod_subcategory, t.calendar_month_desc, c.cust_city, p.prod_subcategory))であり、これによってOracleでは、次のようにsum_sales_pscat_month_city_mvを使用して問合せをリライトできます。

```
SELECT mv.prod_subcategory, mv.calendar_month_desc, mv.cust_city,
  SUM(mv.sum_amount_sold) AS sum_amount_sold
FROM sum_sales_pscat_month_city_mv mv
GROUP BY GROUPING SETS
((mv.prod_subcategory, mv.calendar_month_desc),
  (mv.cust_city, mv.prod_subcategory));
```

問合せでEXPAND_GSET_TO_UNIONヒントを使用している場合は、特殊な状況が発生します。EXPAND_GSET_TO_UNIONの使用例は、拡張GROUP BYを持つ問合せをリライトするためのヒントを参照してください。

12.3.9.1.2 マテリアライズド・ビューが拡張GROUP BYを持ち、問合せが単純GROUP BYを持つ場合

拡張GROUP BYを持つマテリアライズド・ビューをリライトに使用するためには、さらに2つの条件を満たしている必要があります。

- グルーピング識別名が含まれていること。これは、すべてのGROUP BY式のGROUPING_ID関数です。たとえば、マテリアライズド・ビューのGROUP BY句がGROUP BY CUBE(a, b)の場合は、SELECT構文のリストにGROUPING_ID(a, b)を含める必要があります。
- マテリアライズド・ビューのGROUP BYの結果、重複するグルーピングが発生しないこと。たとえば、GROUP BY GROUPING SETS((a, b), (a, b))があると、マテリアライズド・ビューは一般的なリライトの対象外になります。

拡張GROUP BYを持つマテリアライズド・ビューには、複数のグルーピングが含まれています。Oracleは、問合せを計算できる最低コストのグルーピングを検索し、それをリライトに使用します。たとえば、次のマテリアライズド・ビューを考えてみます。

```
CREATE MATERIALIZED VIEW sum_grouping_set_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_category, p.prod_subcategory, c.cust_state_province, c.cust_city,
  GROUPING_ID(p.prod_category,p.prod_subcategory,
              c.cust_state_province,c.cust_city) AS gid,
  SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, customers c
WHERE s.prod_id = p.prod_id AND s.cust_id = c.cust_id
```

```
GROUP BY GROUPING SETS
((p.prod_category, p.prod_subcategory, c.cust_city),
  (p.prod_category, p.prod_subcategory, c.cust_state_province, c.cust_city),
  (p.prod_category, p.prod_subcategory));
```

この場合、次の問合せはリライトされます。

```
SELECT p.prod_subcategory, c.cust_city, SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, customers c
WHERE s.prod_id = p.prod_id AND s.cust_id = c.cust_id
GROUP BY p.prod_subcategory, c.cust_city;
```

この問合せは、マテリアライズド・ビューに最も近似したグルーピングでリライトされます。つまり、次のように、(prod_category, prod_subcategory, cust_city)グルーピングが使用されます。

```
SELECT prod_subcategory, cust_city, SUM(sum_amount_sold) AS sum_amount_sold
FROM sum_grouping_set_mv
WHERE gid = grouping identifier of (prod_category,prod_subcategory, cust_city)
GROUP BY prod_subcategory, cust_city;
```

12.3.9.1.3 マテリアライズド・ビューと問合せの両方が拡張GROUP BYを持つ場合

マテリアライズド・ビューと問合せの両方にGROUP BY拡張機能が含まれている場合、Oracleではリライトに対してグルーピングの一致とUNION ALLリライトという2つの方法が使用されます。最初にグルーピングの一致が試されます。問合せのグルーピングがマテリアライズド・ビューのグルーピングと一致し、しかもロールアップなしで一致する場合、マテリアライズド・ビューからグルーピングが選択されます。たとえば、次の問合せを考えてみます。

```
SELECT p.prod_category, p.prod_subcategory, c.cust_city,
       SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, customers c
WHERE s.prod_id = p.prod_id AND s.cust_id = c.cust_id
GROUP BY GROUPING SETS
((p.prod_category, p.prod_subcategory, c.cust_city),
  (p.prod_category, p.prod_subcategory));
```

この問合せは、sum_grouping_set_mvの2つのグルーピングと一致するため、Oracleは問合せを次のようにリライトします。

```
SELECT prod_subcategory, cust_city, sum_amount_sold
FROM sum_grouping_set_mv
WHERE gid = grouping identifier of (prod_category,prod_subcategory, cust_city)
   OR gid = grouping identifier of (prod_category,prod_subcategory)
```

グルーピングの一致が失敗した場合、UNION ALLリライトと呼ばれる一般的なリライト方法が試みられます。Oracleは、最初に、拡張GROUP BY句を持つ問合せを、同等なUNION ALL問合せで表します。元の問合せのグルーピングはすべて別個のUNION ALLブランチで置き換えられます。ブランチは、単純GROUP BY句を持ちます。たとえば、次の問合せを考えてみます。

```
SELECT p.prod_category, p.prod_subcategory, c.cust_state_province,
  t.calendar_month_desc, SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, customers c, times t
WHERE s.prod_id = p.prod_id AND s.cust_id = c.cust_id
GROUP BY GROUPING SETS
((p.prod_subcategory, t.calendar_month_desc),
  (t.calendar_month_desc),
  (p.prod_category, p.prod_subcategory, c.cust_state_province),
  (p.prod_category, p.prod_subcategory));
```

これは、最初に、次の4つのブランチを持つUNION ALLで表されます。

```
SELECT null, p.prod_subcategory, null,
```

```
    t.calendar_month_desc, SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, customers c, times t
WHERE s.prod_id = p.prod_id AND s.cust_id = c.cust_id
GROUP BY p.prod_subcategory, t.calendar_month_desc
UNION ALL
  SELECT null, null, null,
    t.calendar_month_desc, SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, customers c, times t
WHERE s.prod_id = p.prod_id AND s.cust_id = c.cust_id
GROUP BY t.calendar_month_desc
UNION ALL
SELECT p.prod_category, p.prod_subcategory, c.cust_state_province,
    null, SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, customers c, times t
WHERE s.prod_id = p.prod_id AND s.cust_id = c.cust_id
GROUP BY p.prod_category, p.prod_subcategory, c.cust_state_province
UNION ALL
  SELECT p.prod_category, p.prod_subcategory, null,
    null, SUM(s.amount_sold) AS sum_amount_sold
  FROM sales s, products p, customers c, times t
  WHERE s.prod_id = p.prod_id AND s.cust_id = c.cust_id
  GROUP BY p.prod_category, p.prod_subcategory;
```

次に、個々のブランチは、Oracleによるクエリー・リライト条件のルールを使用して個別にリライトされます。マテリアライズド・ビューsum_grouping_set_mvを使用して、ブランチ3(マテリアライズド・ビューのロールアップが必要)とブランチ4(マテリアライズド・ビューと正確に一致)のみがリライトできます。リライトされないブランチは、元の拡張GROUP BY形式に変換されます。この結果、問合せは次のようにリライトされます。

```
SELECT null, p.prod_subcategory, null,
    t.calendar_month_desc, SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, customers c, times t
WHERE s.prod_id = p.prod_id AND s.cust_id = c.cust_id
GROUP BY GROUPING SETS
  ((p.prod_subcategory, t.calendar_month_desc),
    (t.calendar_month_desc),)
UNION ALL
  SELECT prod_category, prod_subcategory, cust_state_province,
    null, SUM(sum_amount_sold) AS sum_amount_sold
  FROM sum_grouping_set_mv
  WHERE gid = <grouping id of (prod_category,prod_subcategory, cust_city)>
  GROUP BY p.prod_category, p.prod_subcategory, c.cust_state_province
UNION ALL
  SELECT prod_category, prod_subcategory, null,
    null, sum_amount_sold
  FROM sum_grouping_set_mv
  WHERE gid = <grouping id of (prod_category,prod_subcategory)>
```

拡張GROUP BYを持つ問合せは等価なUNION ALLで表され、リライトの最適化のために再帰的に実行されることに注意してください。リライトできないグルーピングはUNION ALLの最後のブランチに残り、かわりに実データがアクセスされます。

12.3.9.2 拡張GROUP BYを持つ問合せのリライトに関するヒント

EXPAND_GSET_TO_UNIONヒントを使用すると、GROUP BY拡張機能を持つ問合せを、等価なUNION ALL問合せへ強制的に拡張できます。このヒントは、マテリアライズド・ビューが単純GROUP BY句のみを持つ環境で使用できます。この場合、各ブランチが別個のマテリアライズド・ビューで独立してリライトできるため、リライトの柔軟性が増します。

12.3.10 ウィンドウ関数がある場合のクエリー・リライト

ウィンドウ関数は、累積集計、移動集計および集中集計の計算に使用されます。これらの関数は、SUM、AVG、MIN、MAX、COUNT、VARIANCE、STDDEV、FIRST_VALUEおよびLAST_VALUEの各集計とともに使用されます。ウィンドウ関数を持つ問合せは、テキストの完全一致のリライトを使用してリライトできます。これには、マテリアライズド・ビューの定義も問合せに完全に一致する必要があります。マテリアライズド・ビューにウィンドウ関数がなく、問合せ内の集計がマテリアライズド・ビューにあり、結合互換性チェックなどの他のすべての適格性チェックにパスした場合に、ウィンドウ関数を持つ問合せはリライトできます。問合せのウィンドウ関数は、マテリアライズド・ビューのウィンドウ関数と比較されます。比較には標準的な形式のフォーマットが使用されます。これによって、複雑なウィンドウ関数でもリライトを実行できます。

クエリー・リライトの際に、ウィンドウ関数を持つ問合せでロールアップが必要となる場合、その問合せは、可能な場合、集計を持つ内部問合せとウィンドウ関数を持つ外部問合せに分割されます。これにより、クエリー・リライトでは、ウィンドウ関数が適用される前に内部問合せの集計をリライトできます。唯一の例外は、問合せにウィンドウ関数とグルーピング・セットの両方がある場合です。この場合、グルーピング・セットがあるため、クエリー・リライトでは問合せを分割できず、問合せはリライトされません。

12.3.11 クエリー・リライトおよび式の一致

マテリアライズド・ビューの列が、問合せの式と一致する事前に計算された式を表している場合、問合せに使用される式は、マテリアライズド・ビューの単純列に置き換えることができます。マテリアライズド・ビューを使用するように問合せをリライトできれば、その方が高速になります。これは、マテリアライズド・ビューには事前計算済の計算が含まれており、式の計算を実行する必要がないためです。

式の一致は、まず式を標準的な形式に変換し、次にそれらが同等かどうかを比較することによって行われます。したがって、2つの異なる式は、互いに同等であれば通常は一致します。さらに、問合せの式全体がマテリアライズド・ビュー内の式と一致しなかった場合は、副式内の一致が検索されます。式を最大限に一致させるために、副式はトップダウンで検索されます。

年齢の範囲(1-10、11-20、21-30など)ごとの売上の合計を問い合せる問合せについて考えてみます。

```
CREATE MATERIALIZED VIEW sales_by_age_bracket_mv
ENABLE QUERY REWRITE AS
SELECT TO_CHAR((2000-c.cust_year_of_birth)/10-0.5,999) AS age_bracket,
       SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, customers c WHERE s.cust_id=c.cust_id
GROUP BY TO_CHAR((2000-c.cust_year_of_birth)/10-0.5,999);
```

次の問合せは式の一致を使用してリライトされます。

```
SELECT TO_CHAR(((2000-c.cust_year_of_birth)/10)-0.5,999), SUM(s.amount_sold)
FROM sales s, customers c WHERE s.cust_id=c.cust_id
GROUP BY TO_CHAR((2000-c.cust_year_of_birth)/10-0.5,999);
```

この問合せは、次のような年齢の範囲の式(つまり、2000 - c.cust_year_of_birth)/10-0.5)の標準的な形式の一致に基づいてsales_by_age_bracket_mvでリライトされます。

```
SELECT age_bracket, sum_amount_sold FROM sales_by_age_bracket_mv;
```

12.3.11.1 部分的に失効したマテリアライズド・ビューを使用したクエリー・リライト

ディテール表のあるパーティションが更新されると、マテリアライズド・ビューの特定セクションのみが失効としてマークされます。マテリアライズド・ビューには、その特定の行またはグループに対応する表のパーティションを識別できる情報が必要です。最も単純な使用例は、マテリアライズド・ビューのSELECT構文のリスト内で表のパーティション化キーを使用する場合です。これが、行を失効パーティションにマップするには最も簡単な方法であるためです。部分的に失効したマテリアライズド・ビューを使用する場合のポイントは、次のとおりです。

- クエリー・リライトでENFORCEDまたはTRUSTEDモードのマテリアライズド・ビューを使用できるのは、そのマテリアライズド・ビューのうち問合せに対する回答に使用される行がFRESHであると認識される場合です。
- マテリアライズド・ビュー内の最新行は、そのビューのWHERE句に選択述語を追加することで識別されます。回答がこの(制限付き)マテリアライズド・ビュー内に含まれる場合は、このマテリアライズド・ビューで問合せをリライトできます。

ファクト表salesは、次のようにtime_idの範囲に基づいてパーティション化されます。

```
PARTITION BY RANGE (time_id)
(PARTITION SALES_Q1_1998
           VALUES LESS THAN (TO_DATE('01-APR-1998', 'DD-MON-YYYY')),
 PARTITION SALES_Q2_1998
           VALUES LESS THAN (TO_DATE('01-JUL-1998', 'DD-MON-YYYY')),
 PARTITION SALES_Q3_1998
           VALUES LESS THAN (TO_DATE('01-OCT-1998', 'DD-MON-YYYY')),
...
```

次のように、time_idでグルーピングされているマテリアライズド・ビューがあるとします。

```
CREATE MATERIALIZED VIEW sum_sales_per_city_mv
ENABLE QUERY REWRITE AS
SELECT s.time_id, p.prod_subcategory, c.cust_city,
       SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, customers c
WHERE s.cust_id = c.cust_id AND s.prod_id = p.prod_id
GROUP BY time_id, prod_subcategory, cust_city;
```

また、パーティションsales_q4_2000に割り当てられる2000年12月に関する新規データが挿入されるとします。テストの目的で、salesに対して任意のDML操作を適用し、このマテリアライズド・ビューが最新の場合に次の問合せがsales_q1_2000のデータを要求すると、これ以外のパーティションを変更できます。たとえば、次のようにします。

```
INSERT INTO SALES VALUES(17, 10, '01-DEC-2000', 4, 380, 123.45, 54321);
```

リフレッシュが完了するまで、マテリアライズド・ビューは一般に失効状態で、enforcedモードでの無制限のリライトには使用できません。ただし、表salesはパーティション化されており、すべてのパーティションが変更されたわけではないため、変更されていないパーティションはすべてOracleによって識別できます。オプティマイザは、次のように問合せを定義するマテリアライズド・ビューに選択述語を暗黙的に追加することにより、マテリアライズド・ビューの最新行(最後のリフレッシュ操作以降に行われた更新の影響を受けないデータ)を識別できます。

```
SELECT s.time_id, p.prod_subcategory, c.cust_city,
       SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, customers c
WHERE s.cust_id = c.cust_id AND s.prod_id = p.prod_id
AND   s.time_id < TO_DATE('01-OCT-2000','DD-MON-YYYY')
OR s.time_id >= TO_DATE('01-OCT-2001','DD-MON-YYYY'))
GROUP BY time_id, prod_subcategory, cust_city;
```

部分的に失効したマテリアライズド・ビューが最新であるかどうかは、論理ベースではなくパーティションごとに追跡されることに注意してください。salesファクト表のパーティション化方法は四半期ベースであるため、2000年12月に変更があると、パーティションsales_q4_2000全体が失効します。

2000年の第1四半期と第2四半期の売上を要求する次の問合せがあるとします。

```
SELECT s.time_id, p.prod_subcategory, c.cust_city,
       SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, customers c
WHERE s.cust_id = c.cust_id AND s.prod_id = p.prod_id
AND   s.time_id BETWEEN TO_DATE('01-JAN-2000', 'DD-MON-YYYY')
```

```
AND TO_DATE('01-JUL-2000', 'DD-MON-YYYY')
GROUP BY time_id, prod_subcategory, cust_city;
```

Oracle Databaseでは、マテリアライズド・ビュー内の該当する行範囲が最新であるとわかっているため、前述の問合せをマテリアライズド・ビューでリライトできます。リライトされた問合せは、次のようになります。

```
SELECT time_id, prod_subcategory, cust_city, sum_amount_sold
FROM sum_sales_per_city_mv
WHERE time_id BETWEEN TO_DATE('01-JAN-2000', 'DD-MON-YYYY')
AND TO_DATE('01-JUL-2000', 'DD-MON-YYYY');
```

マテリアライズド・ビューのSELECT(およびGROUP BYリスト)には、パーティション化キーのかわりにパーティション・マーカー(ROWIDを持つパーティションを識別するファンクション)を使用できます。マテリアライズド・ビューを使用すると、パーティション全体を含むパーティション化キーの範囲を指定する選択述語がある問合せなど、特定のパーティション(パーティション・マーカーで識別可能)のデータのみを必要とする問合せをリライトできます。パーティション・マーカー関数DBMS_MVIEW.PMARKERの詳細は、高度なマテリアライズド・ビューを参照してください。

次の例は、パーティション・キー列を直接使用するのではなく、マテリアライズド・ビューでパーティション・マーカーを使用する方法を示しています。

```
CREATE MATERIALIZED VIEW sum_sales_per_city_2_mv
ENABLE QUERY REWRITE AS
SELECT DBMS_MVIEW.PMARKER(s.rowid) AS pmarker,
       t.fiscal_quarter_desc, p.prod_subcategory, c.cust_city,
       SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, customers c, times t
WHERE s.cust_id = c.cust_id AND s.prod_id = p.prod_id
AND   s.time_id = t.time_id
GROUP BY DBMS_MVIEW.PMARKER(s.rowid),
    p.prod_subcategory, c.cust_city, t.fiscal_quarter_desc;
```

パーティションsales_q1_2000が最新であり、sales表の他のパーティションに対してDML変更が行われたことがわかっているとします。テストの目的で、salesに対して任意のDML操作を適用し、マテリアライズド・ビューが最新のときにsales_q1_2000以外のパーティションを変更できます。次に例を示します。

```
INSERT INTO SALES VALUES(17, 10, '01-DEC-2000', 4, 380, 123.45, 54321);
```

マテリアライズド・ビューsum_sales_per_city_2_mvは、これで一般に失効しているとみなされますが、Oracle Databaseでは、このマテリアライズド・ビューを使用して次の問合せをリライトできます。この問合せは、次に示すように、データをパーティションsales_q1_2000に制限し、cust_cityの特定値のみを選択します。

```
SELECT p.prod_subcategory, c.cust_city, SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, customers c, times t
WHERE s.cust_id = c.cust_id AND s.prod_id = p.prod_id AND s.time_id = t.time_id
AND c.cust_city= 'Nuernberg'
AND s.time_id >=TO_DATE('01-JAN-2000','dd-mon-yyyy')
AND s.time_id <  TO_DATE('01-APR-2000','dd-mon-yyyy')
GROUP BY prod_subcategory, cust_city;
```

PMARKERファンクションを含む部分的に失効したマテリアライズド・ビューでのリライトは、前述の例で示すように、1つ以上のパーティションの完全なデータ内容がアクセスされ、述語条件がパーティション化されたファクト表自体にある場合にのみ実行できることに注意してください。

DBMS_MVIEW.PMARKERファンクションでは、各パーティションごとにまったく別の値が与えられます。これにより、パーティション化キー自体と比べて、潜在的なマテリアライズド・ビュー内の行数は大幅に減りますが、このキーに関する詳細な情報もすべて放棄することになります。わかっている情報は、パーティション番号、したがって、境界値の上限と下限のみです。これは、レンジ・パーティ

ション列のカーディナリティ、つまり行数を減らすこととのトレードオフとなります。

パーティションsales_q1_2000のp_marker値を31070とすると、前述の問合せはマテリアライズド・ビューに対して次のようにリライトできます。

```
SELECT mv.prod_subcategory, mv.cust_city, SUM(mv.sum_amount_sold)
FROM sum_sales_per_city_2_mv mv
WHERE mv.pmarker = 31070 AND mv.cust_city= 'Nuernberg'
GROUP BY prod_subcategory, cust_city;
```

したがって、問合せは失効データにアクセスせずにマテリアライズド・ビューに対してリライトできます。

12.3.12 クエリー・リライト中のカーソルの共有とバインド変数

クエリー・リライトがサポートされるのは、問合せにユーザー・バインド変数が含まれ、実際のバインド値がクエリー・リライトで不要な場合です。バインド変数の実際の値がクエリー・リライトで必要な場合、このクエリー・リライトはバインド値に依存していると言えます。クエリー・リライト時にユーザー・バインド変数は使用できないため、クエリー・リライトがバインド値に依存している場合、問合せのリライトはできません。例として、次のマテリアライズド・ビューcustomer_mvを考えてみます。このマテリアライズド・ビューには、WHERE句に述語(customer_id >= 1000)があります。

```
CREATE MATERIALIZED VIEW customer_mv
ENABLE QUERY REWRITE AS
SELECT cust_id, prod_id,  SUM(amount_sold) AS total_amount
FROM sales WHERE cust_id >= 1000
GROUP BY cust_id, prod_id;
```

次の問合せを考えてみます。この問合せには、WHERE句にユーザー・バインド変数(:user_id)があります。

```
SELECT cust_id, prod_id, SUM(amount_sold) AS sum_amount
FROM sales WHERE cust_id > :user_id
GROUP BY cust_id, prod_id;
```

マテリアライズド・ビューcustomer_mvには、WHERE句に選択述語があるため、クエリー・リライトは、ユーザー・バインド変数user_idの実際の値に依存して包含を計算します。user_idはクエリー・リライト時に使用できず、クエリー・リライトはuser_idのバインド値に依存しているため、この問合せはリライトできません。

前述の例ではWHERE句にユーザー・バインド変数がありますが、ユーザー・バインド変数が問合せのどの部分にあっても、結果は同じです。つまり、ユーザー・バインド変数が問合せのどの箇所にあるかに関係なく、クエリー・リライトがその値に依存している場合、その問合せはリライトできません。

次の問合せを考えてみます。この問合せでは、SELECT構文のリストにユーザー・バインド変数(:user_id)があります。

```
SELECT cust_id + :user_id, prod_id, SUM(amount_sold) AS total_amount
FROM sales WHERE cust_id >= 2000
GROUP BY cust_id, prod_id;
```

この場合、ユーザー・バインド変数user_idの値はクエリー・リライト時に必要ないため、前述の問合せはリライトされます。

```
SELECT cust_id + :user_id, prod_id, total_amount
FROM customer_mv;
```

12.3.13 クエリー・リライトでの式の処理

式がTO_DATE('12-SEP-1999','DD-Mon-YYYY')のように定数に評価される場合は、一部の式でのリライトもサポートされます。たとえば、既存のマテリアライズド・ビューが次のように定義されているとします。

```
CREATE MATERIALIZED VIEW sales_on_valentines_day_99_mv
```

```
BUILD IMMEDIATE
REFRESH FORCE
ENABLE QUERY REWRITE AS
SELECT s.prod_id, s.cust_id, s.amount_sold
FROM times t, sales s WHERE s.time_id = t.time_id
AND t.time_id = TO_DATE('14-FEB-1999', 'DD-MON-YYYY');
```

したがって、次の問合せはリライトできます。

```
SELECT s.prod_id, s.cust_id, s.amount_sold
FROM sales s, times t WHERE s.time_id = t.time_id
AND t.time_id = TO_DATE('14-FEB-1999', 'DD-MON-YYYY');
```

この問合せは、次のようにリライトされます。

```
SELECT * FROM sales_on_valentines_day_99_mv;
```

TO_DATEが使用される際は常に、与えられた日付マスクがNLS_DATE_FORMATで指定された日付マスクと同じである場合にのみクエリー・リライトが発生します。

12.4 同等化を使用した高度なクエリー・リライト

2つのSQL文が機能的に同等であることを宣言できる場合に使用可能な特殊なタイプのクエリー・リライトがあります。この機能を使用すると、内部アプリケーション情報をデータベースに配置でき、データベースがこの情報を活用することにより、問合せのパフォーマンスが向上します。これを実行するには、機能的に同等な(同じ行と列を戻す)SELECT文を2つ宣言して、一方のSELECT文がパフォーマンスに関して有利なように指定します。

この高度なリライトの機能は通常、様々な問合せパフォーマンスの問題や事項に適用できます。この機能はどのようなアプリケーションでも使用でき、複雑なユーザー問合せに対するリライトに作用して、一般的に内部アプリケーション情報を持つユーザーが特別に作成した、より単純でパフォーマンスの高い問合せで回答を得ることができます。

SQL文の変換および飛躍的なパフォーマンス向上のためのチューニングを実現する、内部アプリケーション情報を持つことができるシナリオには様々なものがあります。実行する最適化のタイプには、非常に単純なものや、問合せを大幅に再構成するような高度なものもあります。ただし、入力SQL問合せはアプリケーションで生成される場合が多く、このような問合せの形式および構造は制御できません。

この機能へのアクセス権を取得するには、SYSDBAとして接続し、リライトの同等化を宣言するデータベース管理者に実行アクセス権を明示的に付与する必要があります。詳細は、『*Oracle Database PL/SQLパッケージおよびタイプ・リファレンス*』を参照してください。

この項では、このようなタイプの高度なリライト機能を理解するために、一部の例で多次元データを使用しています。リソース使用率を最適化するため、アプリケーションでは、複雑なSQL、カスタムCコードまたはテーブル・ファンクションを使用して、データベースからデータを取得する場合があります。ただし、エンド・ユーザーに関するかぎり、こうした複雑な操作を行うことはほとんどありません。ユーザーにとっては、SELECT ... GROUP BYなどの一般的な問合せで回答を取得するのが理想的です。

例12-12 同等化を使用したリライト

この例では、指定の代替問合せを使用して特定のユーザー問合せを実行する必要があることを、Oracleに対して宣言します。Oracleはこの関係を認識し、ユーザーから問合せが要求されるたびに、代替問合せを使用して透過的にリライトします。したがって、ユーザーは複雑な集計計算のSQLを学習したり記述したりする必要はありません。

ここにsales_factおよびgeog_dimという2つの実表があります。次の文を発行すると、ロールアップを使用して市、州および地域ごとの合計売上を計算できます。

```
SELECT g.region, g.state, g.city,
```

```
GROUPING_ID(g.city, g.state, g.region), SUM(sales)
FROM sales_fact f, geog_dim g WHERE f.geog_key = g.geog_key
GROUP BY ROLLUP(g.region, g.state, g.city);
```

アプリケーションでは、迅速に結果を戻せるように、この問合せがマテリアライズされることもあります。ただし、生成されるマテリアライズド・ビューによって、非常に多くのディスク領域が占有されてしまいます。ただし、市から州、州から地域にロールアップするディメンションがある場合、次のようにDECODE文を使用して3つのグルーピング列を1つの列に簡単に圧縮できます。これは、埋込み合計と呼ばれます。

```
DECODE (gid, 0, city, 1, state, 3, region, 7, "grand_total")
```

この場合、最低レベルの階層を使用して、情報全体を表します。たとえば、BostonがBoston, MA, New England Regionを表し、CAがCA, Western Regionを表す場合などです。アプリケーションでは、この埋込み合計の結果をembedded_total_salesなどの表に格納します。

ただし、結果を取得する際に、すべてのデータ列(city、state、region)が必要な場合もあります。アプリケーションでは、結果を効率的かつ迅速に戻すため、次のようにカスタム・テーブル・ファンクション(et_function)を使用して、表embedded_total_salesから拡張形式でデータを取り出す場合があります。

```
SELECT * FROM TABLE (et_function);
```

つまりこの機能によって、アプリケーションでは、前述のユーザー問合せと代替問合せの同等化を次のように宣言できるようになります。

```
DBMS_ADVANCED_REWRITE.DECLARE_REWRITE_EQUIVALENCE (
   'EMBEDDED_TOTAL',
   'SELECT g.region, g.state, g.city,
    GROUPING_ID(g.city, g.state, g.region), SUM(sales)
    FROM sales_fact f, geog_dim g
    WHERE f.geog_key = g.geog_key
    GROUP BY ROLLUP(g.region, g.state, g.city)',
    'SELECT * FROM TABLE(et_function)');
```

このDECLARE_REWRITE_EQUIVALENCEの呼出しにより、EMBEDDED_TOTALという同等化の宣言が作成されます。この宣言では、指定されたSOURCE_STMTおよびDESTINATION_STMTが機能的に同等であり、パフォーマンスに関してはDESTINATION_STMTが優先されることを示しています。いったん、このような宣言をDBAが作成すれば、ユーザーは内部で実行されている領域の最適化に関する知識を持つ必要がありません。

またこの機能により、アプリケーションでは、SQL問合せの特殊な部分的なマテリアライズを実行できます。たとえば、例12-13で示す3つのリレーションのUNION ALLを使用して、ロールアップを実行することもできます。

例12-13 同等化を使用したリライト(UNION ALL)

```
CREATE MATERIALIZED VIEW T1
AS SELECT g.region, g.state, g.city, 0 AS gid, SUM(sales) AS sales
FROM sales_fact f, geog_dim g WHERE f.geog_key = g.geog_key
GROUP BY g.region, g.state, g.city;

CREATE MATERIALIZED VIEW T2 AS
SELECT t.region, t.state, SUM(t.sales) AS sales
FROM T1 GROUP BY t.region, t.state;

CREATE VIEW T3 AS
SELECT t.region, SUM(t.sales) AS sales
FROM T2 GROUP BY t.region;
```

ROLLUP(region, state, city)問合せは、次のものと同等になります。

```
SELECT * FROM T1 UNION ALL
SELECT region, state, NULL, 1 AS gid, sales FROM T2 UNION ALL
SELECT region, NULL, NULL, 3 AS gid, sales FROM T3 UNION ALL
SELECT NULL, NULL, NULL, 7 AS gid, SUM(sales) FROM T3;
```

この同等化を指定することにより、Oracle Databaseは問合せの効率的な2番目の形式を使用して、ユーザーが問い合せたROLLUP問合せを計算できます。

```
DBMS_ADVANCED_REWRITE.DECLARE_REWRITE_EQUIVALENCE (
   'CUSTOM_ROLLUP',
   'SELECT g.region, g.state, g.city,
   GROUPING_ID(g.city, g.state, g.region), SUM(sales)
   FROM sales_fact f, geog_dim g
   WHERE f.geog_key = g.geog_key
   GROUP BY ROLLUP(g.region, g.state, g.city ',
   ' SELECT * FROM T1
   UNION ALL
   SELECT region, state, NULL, 1 as gid, sales FROM T2
   UNION ALL
   SELECT region, NULL, NULL, 3 as gid, sales FROM T3
   UNION ALL
   SELECT NULL, NULL, NULL, 7 as gid, SUM(sales) FROM T3');
```

また、この機能を使用すると、概念的には単純でもSQLで表現すると非常に複雑になる特殊な集計計算が可能になります。この場合アプリケーションは、指定のカスタム集計関数を使用し、内部的に複雑なSQLで計算を行うよう、ユーザーへ通知します。

例12-14 同等化を使用したリライト(カスタム集計を使用)

アプリケーション・ユーザーが市、州、地域ごとの売上および特定の季節の売上情報を調べるとします。たとえば、New Englandのユーザーが、冬季の各月におけるNew Englandの各市の売上情報を必要としているとします。アプリケーションでは、前述の集計を計算する特殊な集計Seasonal_Aggが提供されます。この場合、従来のサマリー問合せを行いますが、SUM(sales)ではなくSeasonal_Agg(sales, region)を使用します。

```
SELECT g.region, t.calendar_month_name, Seasonal_Agg(f.sales, g.region) AS sales
FROM sales_fact f, geog_dim g, times t
WHERE f.geog_key = g.geog_key AND f.time_id = t.time_id
GROUP BY g.region, t.calendar_month_name;
```

アプリケーションでは、ユーザーが追加の計算を行うSQLを記述するよう指示されることはなく、この機能により自動的に計算が行われます。この例の場合、Seasonal_Aggはスプレッドシート機能を使用して計算されます(モデリングのSQLを参照)。Seasonal_Aggはユーザー定義の集計ですが、問合せの回答に行を追加することが必要となります。このような処理は、単純なPL/SQLファンクションでは容易には行えません。

```
DBMS_ADVANCED_REWRITE.DECLARE_REWRITE_EQUIVALENCE (
   'CUSTOM_SEASONAL_AGG',
   SELECT g.region, t.calendar_month_name, Seasonal_Agg(sales, region) AS sales
   FROM sales_fact f, geog_dim g, times t
   WHERE f.geog_key = g.geog_key AND f.time_id = t.time_id
   GROUP BY g.region, t.calendar_month_name',
   'SELECT g,region, t.calendar_month_name, SUM(sales) AS sales
   FROM sales_fact f, geog_dim g
   WHERE f.geog_key = g.geog_key AND t.time_id = f.time_id
   GROUP BY g.region, g.state, g.city, t.calendar_month_name
   DIMENSION BY g.region, t.calendar_month_name
   (sales ['New England', 'Winter'] = AVG(sales) OVER calendar_month_name IN
    ('Dec', 'Jan', 'Feb', 'Mar'),
   sales ['Western', 'Summer' ] = AVG(sales) OVER calendar_month_name IN
```

```
   ('May', 'Jun', 'July', 'Aug'), .);
```

12.5 同等化を使用した結果キャッシュ・マテリアライズド・ビューの作成

結果キャッシュ・マテリアライズド・ビュー(RCMV)という特殊なマテリアライズド・ビューを使用すると、クエリー・リライトの実行時に結果キャッシュを使用できます。結果キャッシュ・マテリアライズド・ビューでは、より少ない領域ですばやくアクセスできるという結果キャッシュの主なメリットを得られる上に、クエリー・リライトを実行できないという通常のデメリットから解放されます。

このタイプのマテリアライズド・ビューの使用例を次に示します。

例12-15 結果キャッシュ・マテリアライズド・ビュー

まず、必要な権限を付与します。

```
CONNECT / AS SYSDBA
GRANT CREATE MATERIALIZED VIEW TO sh;
GRANT EXECUTE ON DBMS_ADVANCED_REWRITE TO sh;
```

次に、結果キャッシュ・マテリアライズド・ビューを作成します。

```
CONNECT sh/sh
begin
   sys.DBMS_ADVANCED_REWRITE.Declare_Rewrite_Equivalence
    (
     Name               => 'RCMV_SALES',
     Source_Stmt        =>
      'select channel_id, prod_id, sum(amount_sold), count(amount_sold)
       from sales
       group by prod_id, channel_id',
     Destination_Stmt   =>
      'select * from
      (select /*+ RESULT_CACHE(name=RCMV_SALES) */
                channel_id, prod_id, sum(amount_sold), count(amount_sold)
         from sales
         group by prod_id, channel_id)',
       Validate         => FALSE,
       Rewrite_Mode     => 'GENERAL'
       );
end;
/

ALTER SESSION SET query_rewrite_integrity = stale_tolerated;
```

EXPLAIN PLANを見ながら、異なるすべての問合せがRCMV_SALESにリライトされることを確認します。

```
EXPLAIN PLAN FOR
  SELECT channel_id, SUM(amount_sold) FROM sales GROUP BY channel_id;
@?/rdbms/admin/utlxpls

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 3903632134
------------------------------------------------------------------------------
|Id |         Operation       | Name  |Rows|Bytes|Cost(%CPU)| Time |Pstart|Pstop|
------------------------------------------------------------------------------
| 0 | SELECT STATEMENT        |       |  4 |   64| 1340 (68)|00:00:17|    |    |
| 1 |  HASH GROUP BY          |       |  4 |   64| 1340 (68)|00:00:17|    |    |
| 2 |   VIEW                  |       | 204| 3264| 1340 (68)|00:00:17|    |    |
| 3 |    RESULT CACHE         |3gps5zr86gyb53y36js9zuay2s| | | |     |    |    |
```

```
| 4 |     HASH GROUP BY       |        | 204| 2448| 1340 (68)|00:00:17|     |     |
| 5 |      PARTITION RANGE ALL|        |918K|  10M|  655 (33)|00:00:08|  1  | 28  |
| 6 |       TABLE ACCESS FULL | SALES  |918K|  10M|  655 (33)|00:00:08|  1  | 28  |
-------------------------------------------------------------------------------------

Result Cache Information (identified by operation id):
------------------------------------------------------
   3 - column-count=4; dependencies=(SH.SALES); name="RCMV_SALES"

18 rows selected.
```

キャッシュされた結果を作成する問合せを実行します。

```
SELECT channel_id, SUM(amount_sold)
FROM sales
GROUP BY channel_id;

CHANNEL_ID  SUM(AMOUNT_SOLD)
----------  ----------------
         2        26346342.3
         4          13706802
         3        57875260.6
         9         277426.26
```

結果キャッシュ内でマテリアライズド・ビューがマテリアライズされていることを確認します。

```
CONNECT / AS SYSDBA

SELECT name, scan_count hits, block_count blocks, depend_count dependencies
FROM V$RESULT_CACHE_OBJECTS
WHERE name = 'RCMV_SALES';

NAME         HITS    BLOCKS   DEPENDENCIES
----------   ----    ------   ------------
RCMV_SALES      0         5              1
```

最後に、RCMV問合せの同等化を削除します。

```
begin
  sys.DBMS_ADVANCED_REWRITE.Drop_Rewrite_equivalence('RCMV_SALES');
end;
/
```

結果キャッシュの詳細は、『Oracle Database SQLチューニング・ガイド』を参照してください。

12.6 近似問合せに基づいたクエリー・リライトおよびマテリアライズド・ビュー

近似結果を返すSQL関数を含む問合せは、一致するマテリアライズド・ビューを使用するように自動的にリライトされます(マテリアライズド・ビューを使用してこれらの問合せに応答できる場合)。

近似結果を返すSQL関数を含む問合せを、近似問合せに基づいたマテリアライズド・ビューを使用してリライトする場合は、マテリアライズド・ビューに対してクエリー・リライトが有効になっていることを確認してください。また、データベース・レベルまたは現行セッションで、クエリー・リライトを有効にする必要があります。

次のように定義されたマテリアライズド・ビューapprox_count_distinct_pdt_mvについて検討します。

```
CREATE MATERIALIZED VIEW approx_count_distinct_pdt_mv
ENABLE QUERY REWRITE AS
```

```
SELECT t.calendar_year, t.calendar_month_number, t.day_number_in_month,
approx_count_distinct_detail(prod_id) daily_detail
FROM sales s, times t
WHERE s.time_id = t.time_id
GROUP BY t.calendar_year, t.calendar_month_number, t.day_number_in_month;
```

approx_count_distinct_pdt_mvの定義問合せに一致する問合せが実行中で、かつこの項で説明した前提条件を満たしている場合、この問合せは、このマテリアライズド・ビューを使用するように自動的にリライトされます。次の問合せは、問合せに対して生成された実行計画によって指定された、approx_count_distinct_pdt_mvを使用するようにリライトされます。

```
SELECT t.calendar_year, t.calendar_month_number, t.day_number_in_month, approx_count_distinct(prod_id)
FROM sales s, times t
WHERE s.time_id = t.time_id
GROUP BY t.calendar_year, t.calendar_month_number, t.day_number_in_month;

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------------
-------
Plan hash value: 2307354865
-------------------------------------------------------------------------------------------------------
-------
| Id  | Operation                    | Name                          | Rows  | Bytes | Cost (%CPU)|
Time     |
-------------------------------------------------------------------------------------------------------
-------
|   0 | SELECT STATEMENT             |                               |  1460 | 74460 |   205   (0)|
00:00:01 |
|   1 |  MAT_VIEW REWRITE ACCESS FULL| APPROX_COUNT_DISTINCT_PDT_MV |  1460 | 74460 |   205   (0)|
00:00:01 |
-------------------------------------------------------------------------------------------------------
-------

8 rows selected.
```

また、次の問合せも、実行計画によって指定されたapprox_count_distinct_pdt_mvを使用するようにリライトされます。この問合せは、approx_count_distinct_pdt_mvの定義問合せで定義されたよりも高レベルでデータを集計することに注意してください。

```
SELECT t.calendar_year, t.calendar_month_number, approx_count_distinct(prod_id)
FROM sales s, times t
WHERE s.time_id = t.time_id
GROUP BY t.calendar_year, t.calendar_month_number;

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------------
--------
Plan hash value: 827336432
-------------------------------------------------------------------------------------------------------
--------
| Id  | Operation                    | Name                          | Rows  | Bytes | Cost (%CPU)|
Time     |
-------------------------------------------------------------------------------------------------------
--------
|   0 | SELECT STATEMENT             |                               |    34 |  1632 |   206   (1)|
00:00:01 |
|   1 |  HASH GROUP BY APPROX        |                               |    34 |  1632 |   206   (1)|
00:00:01 |
|   2 |   MAT_VIEW REWRITE ACCESS FULL| APPROX_COUNT_DISTINCT_PDT_MV |  1460 | 70080 |   205   (0)|
00:00:01 |
```

```
-------------------------------------------------------------------------------------------------------------
--------

9 rows selected.
```

近似関数を含むマテリアライズド・ビューを使用するための正確な関数による問合せのリライト

正確な関数を、近似値を返す対応するSQL関数と置き換えるデータベース初期化パラメータを設定する場合、同じ関数の近似バージョンを使用して定義されているマテリアライズド・ビューを使用するように、正確な関数を含む問合せをオプティマイザでリライトできます。対応する近似関数を使用するように問合せをリライトする必要はありません。

たとえば、approx_for_count_distinctパラメータがTRUEに設定されている場合、オプティマイザは、マテリアライズド・ビューapprox_count_distinct_pdt_mvを使用するように次の問合せをリライトします。

```
ALTER SESSION SET approx_for_count_distinct = TRUE;

SELECT t.calendar_year, t.calendar_month_number, COUNT (DISTINCT prod_id)
FROM sales s, times t
WHERE s.time_id = t.time_id
GROUP BY t.calendar_year, t.calendar_month_number;

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------------------
--------
Plan hash value: 827336432
-------------------------------------------------------------------------------------------------------------
--------
| Id  | Operation                     | Name                          | Rows  | Bytes | Cost (%CPU)|
Time     |
-------------------------------------------------------------------------------------------------------------
--------
|   0 | SELECT STATEMENT              |                               |    34 |  1632 |   206   (1)|
00:00:01 |
|   1 |  HASH GROUP BY APPROX         |                               |    34 |  1632 |   206   (1)|
00:00:01 |
|   2 |   MAT_VIEW REWRITE ACCESS FULL| APPROX_COUNT_DISTINCT_PDT_MV |  1460 | 70080 |   205   (0)|
00:00:01 |
-------------------------------------------------------------------------------------------------------------
--------

9 rows selected.
```

前述の実行プランが、前の例で問合せがapprox_count_distinctを使用したときに生成された実行計画と同じであることに注意してください。

関連項目:

- 近似問合せ処理について
- 近似集計について
- 近似問合せに基づいたマテリアライズド・ビューの作成

12.7 ビットマップベースのCOUNT(DISTINCT)関数に基づくクエリー・リライトおよびマテリアライズド・ビュー

整数列に対するCOUNT(DISTINCT)操作を含む問合せは、ビットマップベースの関数を含むマテリアライズド・ビューを使用するようにリライトできます。

マテリアライズド・ビューのクエリー・リライトを有効化して、このマテリアライズド・ビューを使用してSQL問合せをリライトできるようにします。

例12-16 COUNT(DISTINCT)を含むマテリアライズド・ビューを使用したクエリー・リライト

マテリアライズド・ビューmv_salesは、次のコマンドを使用して作成されました。

```
create materialized view mv_sales as
  select PROMO_ID, BITMAP_BUCKET_NUMBER(PROD_ID) bm_bktno,
  BITMAP_CONSTRUCT_AGG(BITMAP_BIT_POSITION(PROD_ID),'RAW') bm_details
  from sales
  group by PROMO_ID,BITMAP_BUCKET_NUMBER(PROD_ID);
```

マテリアライズド・ビューmv_salesに対してクエリー・リライトが有効化されています。

SQL問合せで、mv_salesマテリアライズド・ビュー定義に含まれる数値列に対してCOUNT(DISTINCT)操作を実行すると、問合せはマテリアライズド・ビューを使用するようにリライトされます。次の実行計画は、マテリアライズド・ビューが使用されたことを示しています。

```
SQL> EXPLAIN PLAN FOR select PROMO_ID,count(distinct PROD_ID) from sales group by PROMO_ID order by
PROMO_ID;

Explained.

SQL> SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
------
Plan hash value: 2440767223

-----------------------------------------------
| Id  | Operation               | Name       |
-----------------------------------------------
|   0 | SELECT STATEMENT        |            |
|   1 |  SORT GROUP BY          |            |
|   2 |   VIEW                  |            |
|   3 |    HASH GROUP BY        |            |
|   4 |     MAT_VIEW ACCESS FULL| MV_SALES   |
-----------------------------------------------

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)

15 rows selected.
```

例12-17 ビットマップベースのCOUNT(DISTINCT)およびロールアップを使用したクエリー・リライト

次のコマンドを使用して、マテリアライズド・ビューmv_sales_amountが作成されました。

```
create materialized view mv_sales_amount AS
SELECT PROMO_ID, CHANNEL_ID,
```

```
       BITMAP_BUCKET_NUMBER(PROD_ID) as bm_bktno,
       BITMAP_CONSTRUCT_AGG(BITMAP_BIT_POSITION(PROD_ID)) as bm_details,
       SUM(AMOUNT_SOLD) as amount_sold
FROM sales
GROUP BY PROMO_ID, CHANNEL_ID, BITMAP_BUCKET_NUMBER(PROD_ID);
```

マテリアライズド・ビューmv_sales_amountに対してクエリー・リライトが有効化されています。

次に示すSQLコマンドの実行計画は、COUNT(DISTINCT)関数を含む問合せを満たすクエリー・リライトを示しています。クエリー・リライトは、様々な集計レベルで、他の集計がある状況で実行されます。

```
EXPLAIN PLAN FOR
SELECT PROMO_ID, COUNT(DISTINCT PROD_ID), SUM(AMOUNT_SOLD)
FROM sales
GROUP BY PROMO_ID;
-------------------------------------------------------------------------------------------------
| Id | Operation          | Name            | Rows| Bytes| Cost (%CPU) | Time     |
-------------------------------------------------------------------------------------------------
| 0 | SELECT STATEMENT    |                 | 163 | 6357 | 8 (13)      | 00:00:01 |
| 1 | HASH GROUP BY       |                 | 163 | 6357 | 8 (13)      | 00:00:01 |
| 2 | VIEW                |                 | 163 | 6357 | 8 (13)      | 00:00:01 |
| 3 | HASH GROUP BY       |                 | 163 | 324K | 8 (13)      | 00:00:01 |
| 4 | MAT_VIEW ACCESS FULL| MV_SALES_AMOUNT | 163 | 324K | 7 (0)       | 00:00:01 |
-------------------------------------------------------------------------------------------------
```

12.8 クエリー・リライトが発生したことの確認

クエリー・リライトは透過的に行われるので、問合せがリライトされたかどうかを確認するには、特別なステップを実行する必要があります。問合せが高速に実行された場合、リライトが発生したと考えられますが、これは確認にはなりません。そのため、EXPLAIN PLAN文またはDBMS_MVIEW.EXPLAIN_REWRITEプロシージャを使用して、クエリー・リライトが発生したことを確認します。

この項では、次の項目について説明します。

- クエリー・リライトでのEXPLAIN PLANの使用
- クエリー・リライトでのEXPLAIN_REWRITEプロシージャの使用

12.8.1 クエリー・リライトでのEXPLAIN PLANの使用

EXPLAIN PLAN機能の使用方法については、『*Oracle Database SQL言語リファレンス*』を参照してください。クエリー・リライトの場合、チェックする必要があるのは、操作(OPERATION)にMAT_VIEW REWRITE ACCESSが示されていることのみです。示されている場合、クエリー・リライトが発生したということになります。次の例では、マテリアライズド・ビューcal_month_sales_mvを作成します。

```
CREATE MATERIALIZED VIEW cal_month_sales_mv
ENABLE QUERY REWRITE AS
SELECT  t.calendar_month_desc, SUM(s.amount_sold) AS dollars
FROM sales s, times t WHERE s.time_id = t.time_id
GROUP BY t.calendar_month_desc;
```

次のSQL文でEXPLAIN PLANが使用された場合、結果はデフォルトのPLAN_TABLE表に格納されます。ただし、PLAN_TABLEはutlxplan.sqlスクリプトを使用して最初に作成する必要があります。EXPLAIN PLANでは問合せが実際には実行されないことに注意してください。

```
EXPLAIN PLAN FOR
SELECT  t.calendar_month_desc, SUM(s.amount_sold)
FROM  sales s, times t WHERE s.time_id = t.time_id
```

```
GROUP BY t.calendar_month_desc;
```

PLAN_TABLEから得られる情報のうちクエリー・リライトに関して必要なものは、この問合せの実行方法を特定するための操作OBJECT_NAMEのみです。したがって次のように、出力には操作MAT_VIEW REWRITE ACCESSが表示されます。

```
SELECT OPERATION, OBJECT_NAME FROM PLAN_TABLE;

OPERATION                 OBJECT_NAME
--------------------      ----------------------
SELECT STATEMENT
MAT_VIEW REWRITE ACCESS    CALENDAR_MONTH_SALES_MV
```

12.8.2 クエリー・リライトでのEXPLAIN_REWRITEプロシージャの使用

問合せがリライトされない場合、その原因を容易には特定できないことがあります。クエリー・リライトの対象になるかどうかを制御するルールはきわめて複雑で、制約、ディメンション、クエリー・リライトの整合性モード、マテリアライズド・ビューが最新かどうか、問合せ自体のタイプなど、様々な要因が関係します。また、クエリー・リライトで特定のマテリアライズド・ビューが選択された理由も知る必要があります。そのため、問合せをリライトできる場合にそれを知らせ、リライトできない場合はその理由を知らせるDBMS_MVIEW.EXPLAIN_REWRITEプロシージャが用意されています。DBMS_MVIEW.EXPLAIN_REWRITEの結果を使用すると、問合せをできるかぎりリライトさせるために必要となる適切な措置を講じることができます。

EXPLAIN_REWRITE文で指定した問合せは、実際には実行されないことに注意してください。

この項では、次の項目について説明します。

- DBMS_MVIEW.EXPLAIN_REWRITEの構文
- REWRITE_TABLEを使用したEXPLAIN_REWRITEの出力の表示
- VARRAYを使用したEXPLAIN_REWRITEの出力の表示
- EXPLAIN_REWRITEのメリットに関する統計情報
- EXPLAIN_REWRITEでの32KBを超える問合せテキストのサポート
- EXPLAIN_REWRITEおよび複数のマテリアライズド・ビューについて
- EXPLAIN_REWRITEの出力について

12.8.2.1 DBMS_MVIEW.EXPLAIN_REWRITEの構文

DBMS_MVIEW.EXPLAIN_REWRITEからの出力を取得するには、2つの方法があります。1つは表を使用する方法、もう1つはVARRAYを作成する方法です。出力表を使用するための基本構文は、次のとおりです。

```
DBMS_MVIEW.EXPLAIN_REWRITE (
    query           VARCHAR2,
    mv              VARCHAR2(30),
    statement_id    VARCHAR2(30));
```

utlxrw.sqlスクリプトを実行して、REWRITE_TABLEという名前の出力表を作成できます。

queryパラメータはSQL問合せを表すテキスト文字列です。mvパラメータには、schema.mvという形式の完全修飾されたマテリアライズド・ビュー名を指定します。これはオプションのパラメータです。指定されていない場合、EXPLAIN_REWRITEは、目的の問合せのリライトに必要なすべてのマテリアライズド・ビューに関するメッセージを戻します。schemaを省略してmvのみを指定すると、EXPLAIN_REWRITEでは現在のスキーマにあるマテリアライズド・ビューが検索されます。

EXPLAIN_REWRITEの出力を表ではなくVARRAYに送る場合は、このプロシージャを次のようにコールする必要があります。

```
DBMS_MVIEW.EXPLAIN_REWRITE (
    query           [VARCHAR2 | CLOB],
    mv               VARCHAR2(30),
    output_array     SYS.RewriteArrayType);
```

問合せが256文字未満の場合は、SQL*PlusからEXECUTEコマンドを使用して簡単にEXPLAIN_REWRITEを起動できます。それ以外の場合は、/rdbms/demo/smxrw*の例に示されているように、PL/SQLのBEGIN... ENDブロックを使用する方法をお薦めします。

12.8.2.2 REWRITE_TABLEを使用したEXPLAIN_REWRITEの出力の表示

EXPLAIN_REWRITEの出力は、表REWRITE_TABLEに送ることができます。この出力表を作成するには、スクリプトutlxrw.sqlを実行します。このスクリプトはadminディレクトリにあります。REWRITE_TABLEの形式は、次のとおりです。

```
CREATE TABLE REWRITE_TABLE(
  statement_id          VARCHAR2(30),   -- id for the query
  mv_owner              VARCHAR2(30),   -- owner of the MV
  mv_name               VARCHAR2(30),   -- name of the MV
  sequence              INTEGER,        -- sequence no of the msg
  query                 VARCHAR2(2000), -- user query
  query_block_no        INTEGER,        -- block no of the current subquery
  rewritten_txt         VARCHAR2(2000), -- rewritten query
  message               VARCHAR2(512),  -- EXPLAIN_REWRITE msg
  pass                  VARCHAR2(3),    -- rewrite pass no
  mv_in_msg             VARCHAR2(30),   -- MV in current message
  measure_in_msg        VARCHAR2(30),   -- Measure in current message
  join_back_tbl         VARCHAR2(30),   -- Join back table in message
  join_back_col         VARCHAR2(30),   -- Join back column in message
  original_cost         INTEGER,        -- Cost of original query
  rewritten_cost        INTEGER,        -- Cost of rewritten query
  flags                 INTEGER,        -- associated flags
  reserved1             INTEGER,        -- currently not used
  reerved2              VARCHAR2(10))   -- currently not used;
```

例12-18 REWRITE_TABLEを使用したEXPLAIN_REWRITE

PL/SQLコールの例を次に示します。

```
EXECUTE DBMS_MVIEW.EXPLAIN_REWRITE -
('SELECT p.prod_name, SUM(amount_sold) ' || -
'FROM sales s, products p ' || -
'WHERE s.prod_id = p.prod_id ' || -
' AND prod_name > ''B%'' ' || -
' AND prod_name < ''C%'' ' || -
'GROUP BY prod_name', -
'TestXRW.PRODUCT_SALES_MV', -
'SH');

SELECT message FROM rewrite_table ORDER BY sequence;
MESSAGE
--------------------------------------------------------------------------------
QSM-01033: query rewritten with materialized view, PRODUCT_SALES_MV
1 row selected.
```

デモ・ファイルxrwutl.sqlには、EXPLAIN_REWRITEのより詳細な出力を表示するためのプロシージャが用意されています。詳細は、EXPLAIN_REWRITEの出力についてを参照してください。

次に、いくつかのマテリアライズド・ビューがある中で、結果的にsales_mvが最適なマテリアライズド・ビューとして選択された理由を、1つの具体例を基にして詳しく説明します。

```
DECLARE
  qrytext VARCHAR2(500)  :='SELECT cust_first_name, cust_last_name,
 SUM(amount_sold) AS dollar_sales FROM sales s, customers c WHERE s.cust_id=
 c.cust_id GROUP BY cust_first_name, cust_last_name';
    idno    VARCHAR2(30) :='ID1';
BEGIN
  DBMS_MVIEW.EXPLAIN_REWRITE(qrytext, '', idno);
END;
/
SELECT message FROM rewrite_table ORDER BY sequence;


SQL> MESSAGE
--------------------------------------------------------------------------------
QSM-01082: Joining materialized view, CAL_MONTH_SALES_MV, with table, SALES, not possible
QSM-01022: a more optimal materialized view than PRODUCT_SALES_MV was used to rewrite
QSM-01022: a more optimal materialized view than FWEEK_PSCAT_SALES_MV was used to rewrite
QSM-01033: query rewritten with materialized view, SALES_MV
```

12.8.2.3 VARRAYを使用したEXPLAIN_REWRITEの出力の表示

EXPLAIN_REWRITEの出力はPL/SQLのVARRAYに保存できます。この配列の要素はRewriteMessage型で、SYSスキーマで次のように事前定義されています。

```
TYPE RewriteMessage IS OBJECT(
  mv_owner        VARCHAR2(30),   -- MV's schema
  mv_name         VARCHAR2(30),   -- Name of the MV
  sequence        NUMBER(3),      -- sequence no of the msg
  query_text      VARCHAR2(2000), -- User query
  query_block_no  NUMBER(3),      -- block no of the current subquery
  rewritten_text  VARCHAR2(2000), -- rewritten query text
  message         VARCHAR2(512),  -- EXPLAIN_REWRITE error msg
  pass            VARCHAR2(3),    -- Query rewrite pass no
  mv_in_msg       VARCHAR2(30),   -- MV in current message
  measure_in_msg  VARCHAR2(30),   -- Measure in current message
  join_back_tbl   VARCHAR2(30),   -- Join back table in current msg
  join_back_col   VARCHAR2(30),   -- Join back column in current msg
  original_cost   NUMBER(10),     -- Cost of original query
  rewritten_cost  NUMBER(10),     -- Cost rewritten query
  flags           NUMBER,         -- Associated flags
  reserved1       NUMBER,         -- For future use
  reserved2       VARCHAR2(10)    -- For future use
);
```

配列型RewriteArrayTypeは、RewriteMessageオブジェクトのVARRAYで、SYSスキーマに次のように事前定義されています。

- TYPE RewriteArrayType AS VARRAY(256) OF RewriteMessage;
- この配列型を使用すると、配列変数を宣言して、それをEXPLAIN_REWRITE文の中に指定できます。
- 各RewriteMessageレコードには、リライト処理に関するメッセージがあります。
- パラメータはREWRITE_TABLEと同じですが、出力としてVARRAYを使用する場合、statement_idは使用しません。
- mv_ownerフィールドでは、メッセージに関連するマテリアライズド・ビューの所有者が定義されます。
- mv_nameフィールドでは、メッセージに関連するマテリアライズド・ビューの名前が定義されます。
- sequenceフィールドでは、メッセージに必要な順序が定義されます。
- query_textフィールドには、分析対象となる問合せテキストの最初の2000文字が含まれます。

- messageフィールドには、queryのリライト処理に関連するメッセージのテキストが含まれます。
- flags、reserved1およびreserved2フィールドは、将来のために予約されています。

例12-19 VARRAYを使用したEXPLAIN_REWRITE

次のマテリアライズド・ビューを考えてみます。

```
CREATE MATERIALIZED VIEW avg_sales_city_state_mv
ENABLE QUERY REWRITE AS
SELECT c.cust_city, c.cust_state_province, AVG(s.amount_sold)
FROM sales s, customers c WHERE s.cust_id = c.cust_id
GROUP BY c.cust_city, c.cust_state_province;
```

このマテリアライズド・ビューを次の問合せで使用してみます。

```
SELECT c.cust_state_province, AVG(s.amount_sold)
FROM sales s, customers c WHERE s.cust_id = c.cust_id
GROUP BY c.cust_state_province;
```

ただし、このマテリアライズド・ビューでは、問合せはリライトされません。これでは、リライトに必要な情報がすべてマテリアライズド・ビューにあるように思われるため、不慣れなユーザーは混乱しがちです。特定のマテリアライズド・ビューからAVGを計算できないことは、DBMS_MVIEW.EXPLAIN_REWRITEから判断できます。問題は、ここではROLLUPが必要であり、AVGではCOUNTまたはSUMのROLLUPが必要であることです。

前述の問合せのPL/SQLブロックの例は、出力としてVARRAYを使用すると次のようになります。

```
SET SERVEROUTPUT ON
DECLARE
  Rewrite_Array SYS.RewriteArrayType := SYS.RewriteArrayType();
  querytxt VARCHAR2(1500) := 'SELECT c.cust_state_province,
  AVG(s.amount_sold)
   FROM sales s, customers c WHERE s.cust_id = c.cust_id
   GROUP BY c.cust_state_province';
  i NUMBER;
BEGIN
  DBMS_MVIEW.EXPLAIN_REWRITE(querytxt, 'AVG_SALES_CITY_STATE_MV',
  Rewrite_Array);
  FOR i IN 1..Rewrite_Array.count
  LOOP
    DBMS_OUTPUT.PUT_LINE(Rewrite_Array(i).message);
  END LOOP;
END;
/
```

このEXPLAIN_REWRITE文の出力を次に示します。

```
QSM-01065: materialized view, AVG_SALES_CITY_STATE_MV, cannot compute
  measure, AVG, in the query
QSM-01101: rollup(s) took place on mv, AVG_SALES_CITY_STATE_MV
QSM-01053: NORELY referential integrity constraint on table, CUSTOMERS,
  in TRUSTED/STALE TOLERATED integrity mode
PL/SQL procedure successfully completed.
```

12.8.2.4 EXPLAIN_REWRITEのメリットに関する統計情報

EXPLAIN_REWRITEの出力には、original_costおよびrewritten_costという2つの列があり、クエリー・コストの見積りに役立ちます。original_cost列は、クエリー・リライトを使用禁止にした場合のクエリー・コストに関するオプティマイザの見積りを提供し、rewritten_cost列は、マテリアライズド・ビューを使用して問合せをリライトした場合の見積りを提供します。これらのコスト

値は、特定の問合せがリライトから受けるメリットを調べる場合に使用できます。

12.8.2.5 EXPLAIN_REWRITEでの32KBを超える問合せテキストのサポート

このリリースでは、サイズの大きな問合せでも処理できるよう、EXPLAIN_REWRITEプロシージャが強化されました。入力問合せテキストをVARCHARデータ型のかわりにCLOBデータ型で定義できるようになりました。これにより、EXPLAIN_REWRITEで最大4GBの問合せが許容されます。

次に、CLOBを使用して表に出力する場合のEXPLAIN_REWRITEの構文を示します。

```
DBMS_MVIEW.EXPLAIN_REWRITE(
   query          IN CLOB,
   mv             IN VARCHAR2,
   statement_id   IN VARCHAR2);
```

2番目の引数mvおよび3番目の引数statement_idには、NULLを指定できます。同様に、CLOBを使用してVARRAYに出力する場合のEXPLAIN_REWRITEの構文を示します。

```
DBMS_MVIEW.EXPLAIN_REWRITE(
   query           IN CLOB,
   mv              IN VARCHAR2,
   msg_array       IN OUT SYS.RewriteArrayType);
```

表に出力する場合と同様、2番目の引数mvにはNULLを指定できます。CLOBでの長い問合せテキストは、DBMS_LOBパッケージのプロシージャを使用して生成できます。

12.8.2.6 EXPLAIN_REWRITEおよび複数のマテリアライズド・ビューについて

複数のマテリアライズド・ビューを使用したEXPLAIN_REWRITEの構文は、単一のマテリアライズド・ビューを使用した場合の構文と同じです。ただし、複数のマテリアライズド・ビューはカンマで区切った文字列として指定します。たとえば、マテリアライズド・ビューの特定のセットmv1、mv2およびmv3を問合せquery_txtのリライトに使用できるかどうかを判断し、使用できない場合にその原因を調べるには、次のようにEXPLAIN_REWRITEを使用します。

```
DBMS_MVIEW.EXPLAIN_REWRITE(query_txt, 'mv1, mv2, mv3')
```

マテリアライズド・ビューの特定のセットを使用して問合せquery_txtがリライトされた場合、次のメッセージが表示されます。

```
QSM-01127: query rewritten with materialized view(s), mv1, mv2, and mv3.
```

1つ以上のマテリアライズド・ビューの特定のセットを使用したクエリー・リライトが失敗した場合、リライトに使用されなかった各マテリアライズド・ビューについてEXPLAIN_REWRITEによって失敗の原因が出力されます。

12.8.2.7 EXPLAIN_REWRITEの出力について

/rdbms/demo/smxrw.sqlには、EXPLAIN_REWRITEの使用方法を示した具体例がいくつか用意されています。またデモのxrw領域には、SYS.XRWというユーティリティがあり、EXPLAIN_REWRITEプロシージャの出力内容の選択に使用できます。EXPLAIN_REWRITEにより問合せの評価が行われると、その出力には、リライトされた問合せテキスト、問合せのブロック番号、リライトされた問合せのコストなどの情報が反映されます。ユーティリティSYS.XRWでは、ユーザーにより指定されたフィールドが一定の書式に基づいて出力されるため、内容の判読が容易です。構文は次のとおりです。

```
SYS.XRW(list_of_mvs, list_of_commands, query_text),
```

ただし、list_of_mvsは、クエリー・リライトで使用されると予想されるマテリアライズド・ビューを表します。マテリアライズド・ビューが複数存在する場合は、それぞれをカンマで区切る必要があります。list_of_commandsは、次のいずれかのフィールドを表します。

```
QUERY_TXT:      User query text
REWRITTEN_TXT:  Rewritten query text
QUERY_BLOCK_NO: Query block number to identify each query blocks in
                case the query has subqueries or inline views
PASS:           Pass indicates whether a given message was generated
                before or after the view merging process of query rewrite.
COSTS:          Costs indicates the estimated execution cost of the
                original query and the rewritten query
```

次の例は、このユーティリティの使用方法を示したものです。

```
DROP MATERIALIZED VIEW month_sales_mv;

CREATE MATERIALIZED VIEW month_sales_mv
  ENABLE QUERY REWRITE
  AS
  SELECT t.calendar_month_number, SUM(s.amount_sold) AS sum_dollars
  FROM sales s, times t
  WHERE s.time_id = t.time_id
GROUP BY t.calendar_month_number;

SET SERVEROUTPUT ON
DECLARE
  querytxt VARCHAR2(1500) := 'SELECT t.calendar_month_number,
         SUM(s.amount_sold) AS sum_dollars FROM sales s, times t
  WHERE s.time_id = t.time_id GROUP BY t.calendar_month_number';
BEGIN
  SYS.XRW('MONTH_SALES_MV', 'COSTS, PASS, REWRITTEN_TXT, QUERY_BLOCK_NO', querytxt);
END;
/
```

SYS.XRWの出力は次のようになります。SYS.XRWの出力には、リライト前およびリライト後の問合せのコスト、リライトされた問合せテキスト、問合せのブロック番号、およびビューのマージ処理の前後でメッセージが生成されたかどうかが反映されていることがわかります。

```
================================================================================
>> MESSAGE  : QSM-01151: query was rewritten
>> RW QUERY : SELECT MONTH_SALES_MV.CALENDAR_MONTH_NUMBER CALENDAR_MONTH_NUMBER,
MONTH_SALES_MV.SUM_DOLLARS SUM_DOLLARS FROM SH.MONTH_SALES_MV MONTH_SALES_MV
>> ORIG COST: 19.952763130792                  RW COST: 1.80687108
================================================================================
>>
------------------------ ANALYSIS OF QUERY REWRITE ------------------------
>>
>> QRY BLK #: 0
>> MESSAGE  : QSM-01209: query rewritten with materialized view,
   MONTH_SALES_MV, using text match algorithm
>> RW QUERY : SELECT MONTH_SALES_MV.CALENDAR_MONTH_NUMBER CALENDAR_MONTH_NUMBER,
   MONTH_SALES_MV.SUM_DOLLARS SUM_DOLLARS FROM SH.MONTH_SALES_MV MONTH_SALES_MV
>> ORIG COST: 19.952763130792                  RW COST: 1.80687108
>> MESSAGE OUTPUT BEFORE VIEW MERGING...
============================ END OF MESSAGES ============================
PL/SQL procedure successfully completed.
```

12.9 クエリー・リライトを改善するための設計上の考慮事項

この項では、クエリー・リライトのメリットを最大限引き出す上で役立つ、設計上の考慮事項について説明します。これらは、クエリー・リライトを使用するために必須ではありません。また、これらのガイドラインに従っても、リライトが保証されるわけではありませ

ん。考慮する必要がある一般的なルールは次のとおりです。

- クエリー・リライトの考慮事項: 制約
- クエリー・リライトの考慮事項: ディメンション
- クエリー・リライトの考慮事項: 外部結合
- クエリー・リライトの考慮事項: テキストの一致
- クエリー・リライトの考慮事項: 集計
- クエリー・リライトの考慮事項: グルーピング条件
- クエリー・リライトの考慮事項: 式の一致
- クエリー・リライトの考慮事項: デート・フォールディング
- クエリー・リライトの考慮事項: 統計情報
- クエリー・リライトの考慮事項: ヒント

12.9.1 クエリー・リライトの考慮事項: 制約

マテリアライズド・ビューで参照されるすべての内部結合に、外部キー列に対する追加のNOT NULL制約を使用した参照整合性(外部キー/主キー制約)があることを確認します。制約は通常、大量のオーバーヘッドを伴うため、それらをNO VALIDATEおよびRELYにして、パラメータQUERY_REWRITE_INTEGRITYをSTALE_TOLERATEDまたはTRUSTEDに設定することも可能です。ただし、QUERY_REWRITE_INTEGRITYをENFORCEDに設定した場合、リライト機能の効果を最大限に高めるには、すべての制約をENABLED状態、ENFORCED状態およびVALIDATED状態に設定する必要があります。

予期しない結果となる可能性があるため、ON DELETE句は使用しないようにする必要があります。

12.9.2 クエリー・リライトの考慮事項: ディメンション

正規化されたディメンション表または非正規化ディメンション表の階層関係および機能依存性は、ディメンションのHIERARCHY句およびDETERMINES句を使用して表現できます。ディメンションは、制約では表現できない表内関係を表現できます。ディメンションで宣言された関係をクエリー・リライトで利用するには、パラメータQUERY_REWRITE_INTEGRITYをTRUSTEDまたはSTALE_TOLERATEDに設定する必要があります。

12.9.3 クエリー・リライトの考慮事項: 外部結合

制約を回避するもう1つの方法は、マテリアライズド・ビューに外部結合を使用する方法です。クエリー・リライトは、BのROWIDまたは列B.bがマテリアライズド・ビュー内で使用可能な場合、(A.a=B.b)などの問合せ内の内部結合を、マテリアライズド・ビュー(A.a = B.b(+))内の外部結合から導出できます。外部結合を使用したリライトのほとんどは、結合のみを使用したマテリアライズド・ビューによってサポートされます。それを活用するためには、外部結合を使用したマテリアライズド・ビューは、外部結合の内部表のROWIDまたは主キーを格納する必要があります。たとえば、マテリアライズド・ビュー join_sales_time_product_mv_ojには、外部結合の内部表の主キーprod_idおよびtime_idが格納されます。

12.9.4 クエリー・リライトの考慮事項: テキストの一致

極端に複雑で実行に時間がかかる問合せについて、その処理時間を短縮する必要がある場合は、問合せと同じテキストを使用したマテリアライズド・ビューを作成します。マテリアライズド・ビューには問合せの結果が格納されるため、複合結合の実行に要する時間や全データの中から必要なデータを検索するのに要する時間を節約できます。

12.9.5 クエリー・リライトの考慮事項: 集計

クエリー・リライトで最大の効果を得ることができるよう、問合せの集計を計算するために必要なすべての集計がマテリアライズド・ビューに存在していることを確認します。集計の条件は、増分リフレッシュの条件とよく似ています。たとえば、AVG(x)が問合せ内にある場合、COUNT(x)およびAVG(x)、またはSUM(x)およびCOUNT(x)をマテリアライズド・ビューに格納する必要があります。高速リフレッシュの要件については、高速リフレッシュにおける一般的な制限を参照してください。

12.9.6 クエリー・リライトの考慮事項: グルーピング条件

階層のより低いレベルでデータを集計すると、より高いレベルでデータを集計するより効率的です。より低いレベルは、より多くのクエリー・リライトに使用されるためです。ただし、それによって、より多くの領域が必要となることに注意してください。たとえば、州でグルーピングするのではなく、市でグルーピングした場合などです(領域の制約で禁止されていない場合)。

オーバーラップした、または階層的に関係付けられたGROUP BY列を使用した複数のマテリアライズド・ビューを作成するかわりに、それらすべてのGROUP BY列を使用した単一のマテリアライズド・ビューを作成します。たとえば、市でグルーピングされたマテリアライズド・ビュー、および月でグルーピングされた別のマテリアライズド・ビューを使用するかわりに、市および月でグルーピングされた1つのマテリアライズド・ビューを使用します。

ディメンション内のレベルに対応する列にGROUP BYを使用し、機能的に依存している列には使用しません。これは、クエリー・リライトは、ディメンション内のDETERMINES句をベースにして、機能依存性を自動的に使用できるためです。たとえば、prod_nameでグルーピングするかわりに、prod_idでグルーピングします(属性prod_idによりprod_nameが決定されることを示すディメンションがあれば、prod_nameに関するクエリー・リライトを使用可能にできます)。

12.9.7 クエリー・リライトの考慮事項: 式の一致

複数の問合せに共通の副次選択がある場合、SELECT列の1つとして共通の副次選択を使用したマテリアライズド・ビューを作成すると効果的です。その場合、共通の副次選択の事前計算によって、複数の問合せにおけるパフォーマンスが向上します。

12.9.8 クエリー・リライトの考慮事項: デート・フォールディング

月、四半期、年などのフォールドされた日付単位ごとにデータを集計するマテリアライズド・ビューを作成する場合、年コンポーネントは、常に接頭辞として使用し、接尾辞としては使用しません。たとえば、TO_CHAR(date_col,'yyyy-q')は、日付を四半期にフォールドし、年の順序にそろえますが、TO_CHAR(date_col, 'q-yyyy')は、日付を四半期にフォールドし、四半期の順序にそろえます。前者は順序を維持しますが、後者は順序を維持しません。このため、年の接頭辞なしで作成されたすべてのマテリアライズド・ビューは、デート・フォールディングのリライトには使用できません。

12.9.9 クエリー・リライトの考慮事項: 統計情報

マテリアライズド・ビューを使用した最適化は、コストベースで実行されます。オプティマイザがコストベースの選択をするには、マテリアライズド・ビューと問合せの表の両方の統計情報が必要です。そのため、マテリアライズド・ビューにはDBMS_STATSパッケージを使用して収集された情報が必要です。

12.9.10 クエリー・リライトの考慮事項: ヒント

この項の内容は、次のとおりです。

- クエリー・リライト: REWRITEヒントおよびNOREWRITEヒント
- クエリー・リライト: REWRITE_OR_ERRORヒント
- クエリー・リライト: 複数のマテリアライズド・ビューでのリライトのヒント

- クエリー・リライト: EXPAND_GSET_TO_UNIONヒント

12.9.10.1 クエリー・リライト: REWRITEヒントおよびNOREWRITEヒント

SQL文のSELECTブロックにヒントを含めると、クエリー・リライトの発生を制御できます。問合せにNOREWRITEヒントを使用すると、オプティマイザが問合せをリライトすることを回避できます。

問合せに引数を持たないREWRITEヒントを使用すると、オプティマイザでは、コストにかかわらずマテリアライズド・ビュー(ある場合)によるリライトが強制的に実行されます。引数を持つREWRITE(mv1,mv2,...)ヒントを使用すると、指定した名前のリストから最適なマテリアライズド・ビューを選択してリライトを強制できます。

リライトを防止するには、次の文を使用できます。

```
SELECT /*+ NOREWRITE */ p.prod_subcategory, SUM(s.amount_sold)
FROM   sales s, products p WHERE  s.prod_id = p.prod_id
GROUP BY p.prod_subcategory;
```

sum_sales_pscat_week_mvを使用してリライトを強制するには(リライトが可能な場合)、次の文を使用します。

```
SELECT /*+ REWRITE (sum_sales_pscat_week_mv) */
       p.prod_subcategory,  SUM(s.amount_sold)
FROM   sales s, products p WHERE  s.prod_id=p.prod_id
GROUP BY p.prod_subcategory;
```

リライト・ヒントの有効範囲は問合せブロックです。SQL文が複数の問合せブロック(SELECT句)からなる場合、文全体のリライトを制御するには、各問合せブロックにリライト・ヒントを指定する必要があります。

12.9.10.2 クエリー・リライト: REWRITE_OR_ERRORヒント

問合せでREWRITE_OR_ERRORヒントを使用すると、問合せがリライトに失敗した場合、次のエラーが発生します。

```
ORA-30393: a query block in the statement did not rewrite
```

たとえば、次の問合せでは、使用するクエリー・リライトに適切なマテリアライズド・ビューがない場合、ORA-30393エラーを発行します。

```
SELECT /*+ REWRITE_OR_ERROR */ p.prod_subcategory, SUM(s.amount_sold)
FROM sales s, products p WHERE s.prod_id = p.prod_id
GROUP BY p.prod_subcategory;
```

12.9.10.3 クエリー・リライト: 複数のマテリアライズド・ビューのリライトのヒント

複数のマテリアライズド・ビューを使用する際にリライトを制御するヒントが2つあります。NO_MULTIMV_REWRITEヒントを使用すると、2つ以上のマテリアライズド・ビューによる問合せのリライトが行われなくなり、NO_BASETABLE_MULTIMV_REWRITEヒントを使用すると、マテリアライズド・ビューと実表の組合せによる問合せのリライトが行われなくなります。

12.9.10.4 クエリー・リライト: EXPAND_GSET_TO_UNIONヒント

EXPAND_GSET_TO_UNIONヒントを使用すると、GROUP BY拡張機能を持つ問合せを、強制的に等価なUNION ALL問合せに拡張できます。詳細は、拡張GROUP BYを持つ問合せをリライトするためのヒントを参照してください。

13 属性クラスタリング

属性クラスタリングは、特定の列の内容に基づいて、物理的に近接しているデータをクラスタリングする表レベルのディレクティブです。論理的に同種のデータを物理的に近接させて格納すると、処理するデータ量を大幅に低減でき、ワークロードに含まれる特定の問合せのパフォーマンスを向上させることができます。

この章の内容は次のとおりです。

- 属性クラスタリングについて
- 属性クラスタリングの操作
- 属性クラスタリング情報

13.1 属性クラスタリングについて

属性がクラスタリングされた表では、表中の一群の列、または他の表中の一群の列の値に基づく順序に従って、データをディスク上の近接した位置に格納します。

指定された列の線形順序に応じてクラスタリングすることも、多次元クラスタリングを可能にする機能を使用してクラスタリングすること(インターリーブ・クラスタリング)も可能です。属性クラスタリングは、ゾーン・マップ、Exadata Storage Index、およびインメモリー最小/最大プルーニングの有効性を向上させます。クラスタリングされた列を修飾する問合せは、クラスタリングされたリージョンのみにアクセスします。属性クラスタリングがパーティション表に対して定義されている場合、クラスタリングはすべてのパーティションに適用されます。

属性クラスタリングは、表のディレクティブ・プロパティです。すべてのDML操作に対して強制されるわけではなく、直接パス挿入操作、データ移動または表の作成にのみ影響を及ぼします。表に対する従来のDML操作は、属性クラスタリングの影響を受けません。これは、データをクラスタリングする操作はすべて、現在の作業データ・セットに対してのみ行われる操作であることを意味します。これは、たとえばCTAS操作の一部として発生するような、手動で適用されるORDER BYコマンドとは対照的です。

この項では、次の項目について説明します。

- データのクラスタリングの方法
- 属性クラスタリングのタイプ
- 例: 属性クラスタリングされた表
- 属性クラスタリングを使用するためのガイドライン
- 属性クラスタリングされた表の利点
- 表の属性クラスタリングの定義について
- 属性クラスタリングの実行が必須である場合の指定

13.1.1 データをクラスタリングする方法

データをクラスタリングする方法には次のようなものがあります。

- 属性クラスタリングが定義された表の、1つ以上の列に基づくクラスタリング。
- 属性クラスタリングが定義された表に結合されている、1つ以上の列に基づくクラスタリング。結合された列に基づくクラスタリングは、**結合属性クラスタリング**と呼ばれます。表は主キーと外部キーの関係を介して接続されますが、外部キーは強制される必要はありません。

通常はスター・クエリーがディメンション階層を修飾するため、ファクト表が1つ以上のディメンション表の列(属性)に基づいてクラスタリングされる場合は有益であることがあります。結合属性クラスタリングを使用すると、1つ以上のディメンション表をファクト表に結合し、その後でディメンション階層列によってファクト表データをクラスタリングできます。1つ以上のディメンション表にある列上でファクト表をクラスタリングする場合、ディメンション表への結合がディメンション表の主キーまたは一意キーの上にある必要があります。表データはディメンション階層によってクラスタリングされるため、スター・クエリーのコンテキストにおける結合属性クラスタリングは階層クラスタリングとも呼ばれますが、その各階層は階層列の順序付きリストからできています(たとえば、nation、stateおよびcityの各列がlocation階層を構成しています)。

ノート: Oracle表クラスタとは対照的に、結合属性でクラスタリングされた表は、表のグループからのデータを、同じデータベース・ブロックには格納しません。例として、属性クラスタリングされた表salesが、ディメンション表productsと結合されている場合を見てみましょう。sales表にはsales表の行のみが含まれますが、行の順序は、products表から結合された列の値に基づきます。適切な結合は、データ移動、直接パス挿入、およびCTAS操作の際に実行されます。

13.1.2 属性クラスタリングのタイプ

属性クラスタリングはユーザー定義の表ディレクティブで、表内の1つ以上の列に対するデータ・クラスタリングです。ディレクティブは、表の作成または変更時に指定できます。

Oracle Databaseには、次のタイプの属性クラスタリングが用意されています。

- 線形順序の属性クラスタリング
- インターリーブ順序の属性クラスタリング

データのクラスタリングは、使用されている属性クラスタリングのタイプに関係なく、単一の表に基づいて行うことも、複数の表を結合することによって行うこと(結合属性クラスタリング)もできます。

13.1.2.1 線形順序の属性クラスタリング

線形順序では、指定された列の順序によってデータが格納されます。これがクラスタリングのタイプのデフォルトです。たとえば、表SALESの(prod_id、channel_id)列における線形順序では、先にprod_idを基準にソートされ、その後でchannel_idを基準にソートされます。ソートされたデータは、近接するクラスタリング済の列のデータとともにディスクに格納されます。

線形順序は、主キーと外部キーの関係を介して接続される単一または複数の表に対して定義できます。

指定した列の順序に基づいて属性クラスタリングを実行するには、CLUSTERING ... BY LINEAR ORDERディレクティブを使用します。

列の線形順序に基づく属性クラスタリングは、次のシナリオでの使用が最適です。

- 問合せで、単一の表内のCLUSTERING句に含まれている列の接頭辞が指定されている

  たとえば、sales上での問合せで、顧客ID、または顧客IDと製品IDの組合せがよく指定される場合、cust_id, prod_idという列の順序を使用して表内のデータをクラスタリングすることが考えられます。

- CLUSTERING句で使用される列に、許容できるレベルのカーディナリティがある

  属性クラスタリングされた表の利点に記載されたシナリオで取得できるデータの削減の程度は、ある列の述語から取得されるデータの削減に正比例します。

ゾーン・マップと線形クラスタリングの組合せは、I/Oの削減に非常に効果的です。

13.1.2.2 インターリーブ順序の属性クラスタリング

インターリーブ順序では、Z次カーブ・フィッティングに基づいて、特別な多次元クラスタリング・テクニックを使用します。その場合、列値(データ・ポイント)の多次元ローカリティを保存しつつ、複数列の属性値(多次元データ・ポイント)を単一の一次元の値に

マップします。インターリーブ順序は、単一の表と複数の表のいずれの上でもサポートされます。線形順序とは異なり、クラスタリング定義の先頭列が存在しなくても、属性クラスタリングされた表の利点に記載されているシナリオにおけるI/Oプルーニングの利点を得られます。

列は個別に使用することも、列グループにグループ化することもできます。個別の列または列グループは、クラスタ内の多次元データ・ポイントの1つを構成するためにそれぞれ使用されます。グループ化された列は「(」と「)」で囲まれ、粒度の粗いものから細かいものへと続くディメンション階層に従う必要があります。たとえば、(product_category, product_subcategory)のようにします。

インターリーブ順序によってクラスタリングを実行するには、CLUSTERING ... BY INTERLEAVED ORDERディレクティブを使用します。

インターリーブ・クラスタリングは、複数の列上にある多様な述語によるSQL操作に最適です。これはしばしば次元モデルに対するスター・クエリーの場合に起こることで、その場合、問合せの述語はディメンション表上にあり、述語の数は様々です。ディメンション表からの列に基づいてファクト表がクラスタリングされる環境では、インターリーブ結合された属性クラスタリングの使用が最も一般的です。ディメンション表からの列は、高い確率で、階層(たとえば、製品カテゴリとサブ・カテゴリの階層)を含みます。この場合、ファクト表のクラスタリングは、階層を構成するディメンション列の上で発生します。スター・スキーマの結合属性クラスタリングが階層クラスタリングと呼ばれることがあるのはこのためです。たとえば、sales上の問合せで異なるディメンションからの列が指定されている場合、これらのディメンションの列に応じてsales表内のデータをクラスタリングすることが考えられます。

ゾーン・マップとインターリーブ・クラスタリングの組合せは、スター・スキーマ問合せのI/Oプルーニングに非常に効果的です。また、ゾーン・マップを使用する問合せで非常に効率的なI/Oプルーニングを可能にし、同じ列の値が互いに近接していて簡単に圧縮できるため、圧縮率も向上します。

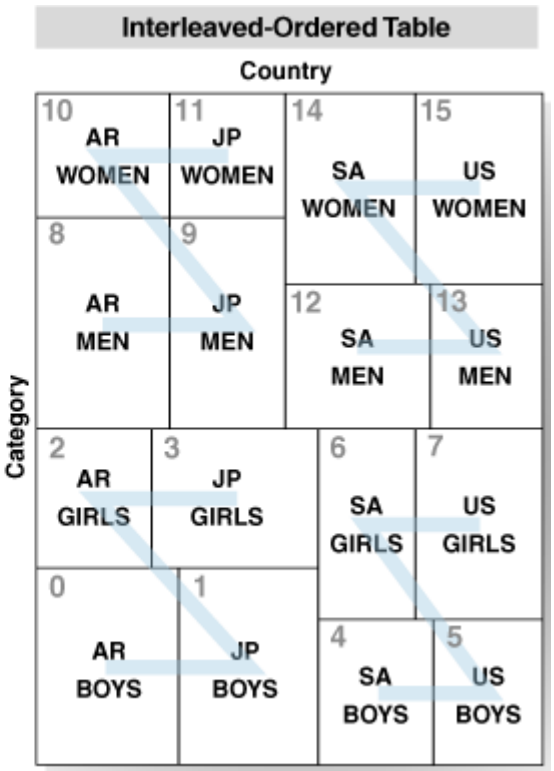
13.1.3 例: 属性クラスタリングされた表

クラスタリングされた表のようすの一例を、図13-1に示します。表salesに列(category、country)があると想定しています。左側の表は、線形順序を使用してクラスタリングされており、右側の表はインターリーブ順序を使用してクラスタリングされています。インターリーブ順序の表では、指定のカテゴリおよび国のデータが含まれるディスク上に連続したリージョンがあることになります。

図13-1 属性クラスタリングされた表

Linear-Ordered Table

Category	Country
BOYS	AR
BOYS	JP
BOYS	SA
BOYS	US
GIRLS	AR
GIRLS	JP
GIRLS	SA
GIRLS	US
MEN	AR
MEN	JP
MEN	SA
MEN	US
WOMEN	AR
WOMEN	JP
WOMEN	SA
WOMEN	US



13.1.4 属性クラスタリングを使用するためのガイドライン

属性クラスタリングされた表を定義する場合の考慮事項を次に示します。

- ゾーン・プルーニングとそれに関連するI/O削減を容易にするには、属性クラスタリングとゾーン・マップを組み合せて使用します。
- カーディナリティが中または低い列では、述語を使用して問い合せられることが多い大きな表の使用を検討します。
- ディメンション階層による問合せが多い場合はファクト表を検討します。
- パーティション表の場合は、パーティション・キーに相関する列を含めることを検討します(ゾーン・マップのパーティション・プルーニングを容易にするため)。
- 線形順序の場合、列を接頭辞から接尾辞へという順序でリストします。
- ディメンション階層を構成する列は、まとめてグループ化します。これにより列グループが構成されます。列グループそれぞれの中では、粒度が粗いものから細かいものへという順序で列をリストします。
- 4つ以上のディメンション表がある場合、フィルタで指定されることが最も多いディメンションを含めます。クラスタリングの効果を高くするには、ディメンションの数を2または3に制限します。
- カーディナリティが中または低い列では、索引のかわりに属性クラスタリングを使用することを検討します。
- ディメンション表の主キーがディメンション階層の値で構成されている場合(たとえば、主キーが年、四半期、月、日の各値から構成される場合)、対応する外部キーは、ディメンション階層のかわりにクラスタリング列として作成します。

13.1.5 属性クラスタ表の利点

- 索引の使用に関連するコストが不要
- ゾーン・マップとともに使用すると、ランダムI/Oまたはフル表スキャンを実行せずにクラスタリングされたリージョンにアクセス可能
- 次のいずれかと組み合せて使用することでI/O削減が可能
    - Oracle Exadata Storage Index
    - Oracle In-memory最小/最大プルーニング
    - ゾーン・マップ

  属性クラスタリングは、フィルタ述語として使用される属性に基づいたデータ・クラスタリングです。Exadata Storage IndexとOracle In-memory最小/最大プルーニングが各物理リージョンに格納された列の最小値と最大値を追跡するため、クラスタリングすると、データにアクセスするために必要なI/Oが低減されます。

  ゾーン・マップを使用してI/Oプルーニングすると、表スキャンと索引スキャンのI/OコストとCPUコストを大幅に削減できます。
- スター・スキーマで、ディメンション列に基づくファクト表のクラスタリングを可能にします。

  従来の表クラスタなどのテクニックでは、他の表の列を基準にした順序を使用できません。スター・スキーマでは、大部分の問合せはディメンション表を修飾し、ファクト表を修飾しないため、ファクト表の列を基準にするクラスタリングは効果的ではありません。Oracle Databaseは、ディメンション表の列上でのクラスタリングをサポートしています。
- データ圧縮率を改善し、それにより間接的に表スキャンのコストを改善します。

  クラスタリングを使用すると、同じ値を持つクラスタリングされた列がディスク上で近接している可能性が高くなり、それに

よりデータベースによる圧縮が容易になるため、圧縮率が改善されることもあります。

- 属性クラスタリングが索引選択基準に関わる場合、索引範囲スキャン操作のための表参照操作と単一のブロックI/O操作が最小になります。
- OLTPアプリケーションで、接頭辞を修飾し、線形順序の属性クラスタリングを使用する問合せのI/Oを削減します。
- インターリーブ順序による属性クラスタリングで、クラスタリング列のサブセットにおけるI/O削減を可能にします。

  表データが複数の列上で順序づけられている場合(索引で整理された表など)、問合せでI/Oを少なく抑えるには、列の接頭辞を指定する必要があります。対照的に、BY INTERLEAVED表では、問合せが接頭辞でない順序で複数の表から列を指定する場合に、I/Oプルーニングが有益です。

13.1.6 表の属性クラスタリングの定義について

属性クラスタリング情報は、表メタデータの一部です。表の属性クラスタリングを定義できるのは、表が最初に作成されたときか、その後で表定義を変更したときです。

表の属性クラスタリングを定義するには、CREATE TABLE文のCLUSTERING句を使用します。属性クラスタリングのタイプを指定するには、BY LINEAR ORDERまたはBY INTERLEAVED ORDERを含めます。

関連項目:

- 線形順序で属性クラスタリングされた表の作成
- インターリーブ順序で属性クラスタリングされた表の作成

表の作成時に属性クラスタリングを定義しなかった場合、表定義を変更してクラスタリングを追加できます。既存の表の属性クラスタリングを定義するには、ALTER TABLE ... ADD CLUSTERING文を使用します。

関連項目:

既存の表への属性クラスタリングの追加

13.1.7 属性クラスタリングの実行が必須である場合の指定

クラスタリングを実行すると、DML操作の際に表とクラスタリング・データが再編成されるため、コストが高くなる場合があります。Oracle Databaseでは、従来のDML、従来の挿入、更新およびマージにおけるデータのクラスタリングは必須ではありません。

クラスタリングは、次の2つの方法で行うことができます。1つめの方法は、表上での特定のDML操作に対するクラスタリングの自動実行です。これは、クラスタリングがトリガーされる操作を表メタデータの一部として定義することによって行われます。2つめの方法は、ヒントを使用したDML操作の属性クラスタリングの制御およびDDL操作における属性クラスタリングの表レベル設定のオーバーライドの説明に従ってクラスタリングを実行する必要があることを明示的に指定することです。この場合、メタデータ定義にクラスタリングが含まれない場合でも、表のクラスタリングを実行できます。

表定義の一部として、次の操作がトリガーされたときに属性クラスタリングを実行する必要があると指定できます。

- 直接パス挿入操作

  直接パス挿入操作の際に属性クラスタリングを実行する必要があると指定するには、ON LOADオプションをYESに設定

します。

- データ移動操作

  データ移動操作時にクラスタリングを実行する必要があることを指定するには、ON DATA MOVEMENTオプションをYESに設定します。これには、表のオンライン再定義と、次のパーティション操作が含まれます: MOVE、MERGE、SPLITおよびCOALESCE。

ON LOADオプションとON DATA MOVEMENTオプションは、CREATE TABLEまたはALTER TABLE文にも含めることができます。YES ON LOADもYES ON DATA MOVEMENTも指定されない場合、クラスタリングは自動実行されません。

後でゾーン・マップ作成用に使用される可能性がある表の自然クラスタリングを定義するメタデータとしてのみ使用されます。この場合、ロード時にクラスタリングを実行するかどうかはユーザー次第です。

関連項目:

ON LOADオプションおよびON DATA MOVEMENTオプションを使用する例については、既存の表への属性クラスタリングの追加を参照してください。

13.2 属性クラスタリングの操作

この項では、属性クラスタリングを含む一般的なタスクについて、次のように分けて説明します。

- 属性クラスタリングされた表の権限
- 線形順序で属性クラスタリングされた表の作成
- インターリーブ順序で属性クラスタリングされた表の作成
- 属性クラスタリングのメンテナンス

13.2.1 属性クラスタリングされた表の権限

表の属性クラスタリングを定義するには、表に対してCREATEまたはALTER権限がある必要があります。さらに、結合属性クラスタリングの場合、結合された表に対してSELECT権限かREAD権限を持っている必要もあります。

関連項目:

CREATE TABLEのCLUSTERING句の構文とセマンティクスの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

13.2.2 線形順序で属性クラスタリングされた表の作成

線形順序では、指定した列の順序によってデータが格納されます。ORDER BY句と同じです。線形順序は、スター・スキーマで、単一または複数の表の列に対してサポートされます。線形順序の属性クラスタリングの例には、線形順序の属性クラスタリング表の例があげられています。

関連項目:

属性クラスタリングの制限に関する詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

13.2.2.1 線形順序の属性クラスタリングの例

例13-1と例13-2は、線形順序の説明です。

例13-1 線形順序による表の作成

sales上での問合せで、顧客ID、または顧客IDと製品IDの組合せがよく指定されると想定します。属性クラスタリングされた表を作成して、属性クラスタリングされた表の利点に記載されたシナリオで、問合せにI/O削減の利点がもたらされるようにできます。

次の文は、sales表を線形順序で作成します。

```
CREATE TABLE sales (
  prod_id        NUMBER(6) NOT NULL,
  cust_id        NUMBER NOT NULL,
  time_id        DATE NOT NULL,
  channel_id     CHAR(1) NOT NULL,
  promo_id       NUMBER(6) NOT NULL,
  quantity_sold  NUMBER(3) NOT NULL,
  amount_sold    NUMBER(10,2) NOT NULL
)
CLUSTERING
  BY LINEAR ORDER (cust_id, prod_id);
```

このクラスタリングされた表は、cust_id上の述語か、cust_idとprod_id両方にある述語を含んでいる問合せに有効です。

例13-2 線形順序の、結合を備えた表の作成

productsディメンション表において、prod_id列上に一意キーまたは主キーがあると想定します。この表の他の列には、prod_name、prod_desc、prod_category、prod_subcategoryおよびprod_statusが含まれますが、それに限定されるわけではありません。my_salesファクト表に対する問合せには、多くの場合、次のいずれかが含まれます。

- cust_id上の述語
- cust_idおよびprod_category上の述語
- cust_id、prod_categoryおよびprod_subcategory上の述語

my_sales表に属性クラスタリングを定義すると、CLUSTERING句に含まれる述語を含む問合せに有効です。

```
CREATE TABLE my_sales (
  prod_id        NUMBER(6) NOT NULL,
  cust_id        NUMBER NOT NULL,
  time_id        DATE NOT NULL,
  channel_id     CHAR(1) NOT NULL,
  promo_id       NUMBER(6) NOT NULL,
  quantity_sold  NUMBER(3) NOT NULL,
  amount_sold    NUMBER(10,2) NOT NULL
)
CLUSTERING
  my_sales JOIN products ON (my_sales.prod_id = products.prod_id)
  BY LINEAR ORDER (cust_id, prod_category, prod_subcategory);
```

関連項目:

BY LINEAR ORDER句の構文とセマンティクスの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

13.2.3 インターリーブ順序で属性クラスタリングされた表の作成

インターリーブ順序では、Z次ソートに類似した、特別な多次元クラスタリング・テクニックを使用します。ほとんどの場合によく使用されるが、必ずしもそのすべてを使用するわけではない一群の述語がある場合に特に有効です。インターリーブ順序は、データウェアハウスにおいて、スター・スキーマのディメンション階層に対して有効です。インターリーブ順序の属性クラスタリングの例には、線形順序の属性クラスタリング表の例があげられています。

関連項目:

属性クラスタリングの制限に関する詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

13.2.3.1 インターリーブ順序の属性クラスタリングの例

例13-3と例13-4は、インターリーブ順序の説明です。

問合せでゾーン・マップを使用したプルーニングを活用できるように、属性クラスタリングされた表を作成することもできます。属性クラスタリングによるゾーン・マップの作成には、属性クラスタリングを使用したゾーン・マップ定義の例があげられています。

例13-3 インターリーブ順序による表の作成

sales上での問合せで、時刻ID、または時刻IDと製品IDの組合せがよく指定されると想定します。インターリーブ属性クラスタリングを設定してsalesを作成するには、次のコマンドを使用します。

```
CREATE TABLE sales (
  prod_id        NUMBER(6) NOT NULL,
  cust_id        NUMBER NOT NULL,
  time_id        DATE NOT NULL,
  channel_id     CHAR(1) NOT NULL,
  promo_id       NUMBER(6) NOT NULL,
  quantity_sold  NUMBER(3) NOT NULL,
  amount_sold    NUMBER(10,2) NOT NULL
)
CLUSTERING
  BY INTERLEAVED ORDER (time_id, prod_id);
```

このクラスタリングされた表は、次のいずれかを含む問合せで有効です。

- time_id上の述語
- prod_id上の述語
- time_idおよびprod_id上の述語

例13-4 インターリーブ順序と結合のある表の作成

大規模なデータ・ウェアハウスでは、データがスター・スキーマを使用して組織化されることがよくあります。ディメンション表は親子階層を使用し、外部キーでファクト表に接続されます。ファクト表をインターリーブ順序でクラスタリングすると、表スキャンの際、データベースがディメンション列の値をスキップする特別な機能を使用できるようになります。クラスタリングで外部キー関係が必須でない点に注意してください。ただし、Oracle Databaseでは、ディメンション表上に主キーまたは一意のキーが必要です。

次のコマンドは、salesファクト表に対する、インターリーブ順序を使用した属性クラスタリングを定義します。

```
CREATE TABLE sales (
  prod_id        NUMBER(6) NOT NULL,
  cust_id        NUMBER NOT NULL,
  time_id        DATE NOT NULL,
  channel_id     CHAR(1) NOT NULL,
  promo_id       NUMBER(6) NOT NULL,
  quantity_sold  NUMBER(3) NOT NULL,
  amount_sold    NUMBER(10,2) NOT NULL
)
CLUSTERING
  sales JOIN products ON (sales.prod_id = products.prod_id)
  BY INTERLEAVED ORDER ((time_id), (prod_category, prod_subcategory));
```

このクラスタリングされた表は、次のいずれかを含む問合せで有効です。

- time_id上の述語
- prod_category上の述語
- prod_categoryおよびprod_subcategory上の述語
- time_idおよびprod_id上の述語
- time_id、prod_categoryおよびprod_subcategory上の述語

関連項目:

CREATE TABLE文とCLUSTERING句の詳細は、Oracle Database SQL言語リファレンスを参照

13.2.4 属性クラスタリングのメンテナンス

表の属性クラスタリング定義は、いつでも追加、削除および更新できます。変更された定義は既存の表データに影響せず、将来の操作のためのディレクティブとしてのみ使用できます。

次のメンテナンス操作は、表のメタデータを変更します。

- 既存の表への属性クラスタリングの追加
- 属性クラスタリング定義の変更
- 既存の表の属性クラスタリングの削除

実行時に表の属性クラスタリング定義をオーバーライドすることもできます。実行時に属性クラスタリング動作に影響するメンテナンス操作は次のとおりです。

- ヒントを使用したDML操作の属性クラスタリングの制御
- DDL操作における属性クラスタリングの表レベル設定のオーバーライド
- 表のオンライン再定義時の表データのクラスタリング

13.2.4.1 既存の表への属性クラスタリングの追加

クラスタリングのある表を作成する場合、デフォルトでゾーン・マップがともに作成されます。ただし、WITHOUT ZONEMAPを使用することにより、これを明示的に抑制することができます。こうするのは、クラスタリング列と、クラスタリング列に相関する追加の列に関するゾーン・マップを作成する、デフォルトのものでなく固有のゾーン・マップ・ストレージ・オプションを使用するといった場合です。

現在属性クラスタリングを使用していない既存の表に属性クラスタリングを追加するには、ALTER TABLE ... ADD

CLUSTERINGコマンドを使用します。

次のコマンドは、SALESファクト表に属性クラスタリングを追加します。変更後の表では、結合されたディメンション表CUSTOMERSおよびPRODUCTSに基づくインターリーブ・クラスタリングが使用されます。

```
ALTER TABLE sales
ADD CLUSTERING sales JOIN customers ON (sales.cust_id = customers.cust_id)
                     JOIN products ON (sales.prod_id = products.prod_id)
    BY INTERLEAVED ORDER ((prod_category, prod_subcategory),
                          (country_id, cust_state_province, cust_city))
    YES ON LOAD YES ON DATA MOVEMENT
    WITHOUT MATERLALIZED ZONEMAP;
```

表にクラスタリングを追加する場合、既存のデータはクラスタリングされません。既存のデータを強制的にクラスタリングするには、ALTER TABLE. . . MOVE文を使用する必要があります。その操作はパーティションごとに行います。

次のコマンドは、sales表内のデータをクラスタリングします。

```
ALTER TABLE sales MOVE PARTITION sales_1995 UPDATE INDEXES ALLOW CLUSTERING;
```

ゾーン・マップの詳細は、ゾーン・マップについてを参照してください。

13.2.4.2 属性クラスタリング定義の変更

ALTER TABLE ... MODIFY CLUSTERING文を使用すると、表に対して属性クラスタリングをいつトリガーするかを変更できます。クラスタリング定義を変更しても、既存の表データには影響ありません。変更された定義は、その後のデータ移動または直接パス挿入操作のみに適用されます。

次のコマンドは、SALES表のクラスタリング定義を変更し、データ移動時にクラスタリングを有効化します。

```
ALTER TABLE sales MODIFY CLUSTERING YES ON DATA MOVEMENT;
```

表定義を変更し、属性クラスタリングに基づいてゾーン・マップを作成したり削除したりすることもできます。次の文は、SALES表の定義を変更して、ゾーン・マップを追加します。

```
ALTER TABLE sales MODIFY CLUSTERING WITH MATERIALIZED ZONEMAP;
```

属性クラスタリングされた表SALESの定義を変更して、ゾーン・マップを削除するには、次の文を使用します。

```
ALTER TABLE sales MODIFY CLUSTERING WITHOUT MATERIALIZED ZONEMAP;
```

13.2.4.3 既存の表の属性クラスタリングの削除

既存の表に対して属性クラスタリングが定義されている場合、属性クラスタリングを削除するには、ALTER TABLE ... DROP CLUSTERING文を使用します。クラスタリング定義を削除しても、既存の表データにはいかなる影響もありません。

次のコマンドは、SALES表の属性クラスタリングを削除します。

```
ALTER TABLE sales DROP CLUSTERING;
```

13.2.4.4 ヒントを使用したDML操作の属性クラスタリングの制御

ヒントを使用すると、クラスタリングの使用を強制したり、直接パス挿入操作時に使用を抑制したりできます。表に対してクラスタリングを強制するにはCLUSTERINGヒントを使用し、クラスタリングの使用を抑制するにはNO_CLUSTERINGヒントを使用します。

次のコマンドは、SALES表にデータを挿入する際に、属性クラスタリングを無効にします。この表は、YES ON LOADオプションで作成されています。

```
INSERT /*+ APPEND NO_CLUSTERING */ INTO sales SELECT * FROM external_sales;
```

ヒントの詳細は、ゾーン・マップ使用の制御を参照してください。

13.2.4.5 DDL操作における属性クラスタリングの表レベル設定のオーバーライド

新しいデータ・セグメントの作成(分割またはマージ操作)や、表、パーティションまたはサブパーティションの移動などのデータ移動DDL操作の際に、属性クラスタリング定義をオーバーライドできます。たとえば、表がNO ON DATA MOVEMENTオプションを使用して定義されている場合、ALTER TABLE ... ALLOW CLUSTERING文を使用して、データ移動時にこの表のデータをクラスタリングすることができます。

次のコマンドを使用すると、NO ON DATA MOVEMENTオプションを使用して定義されているSALES表のsales_2010パーティションでのデータ移動の際に、クラスタリングが可能になります。

```
ALTER TABLE sales MOVE PARTITION sales_2010 UPDATE INDEXES ALLOW CLUSTERING;
```

同様に、YES ON DATA MOVEMENTオプションを使用して定義されている表のデータ移動の際のクラスタリングを無効にするには、データ移動に使用するALTER TABLEコマンドにDISALLOW CLUSTERING句を含めます。

13.2.4.6 表のオンライン再定義時の表データのクラスタリング

表のオンライン再定義を使用すると、表の可用性に大幅に影響を及ぼすことなく表の論理構造または物理構造を変更できます。表は、その再定義プロセスの大部分で、問合せおよびDMLを使用してアクセスできます。

表をオンラインで再定義して、属性クラスタリングを以前に使用したことがない表に属性クラスタリングを追加できます。DBMS_REDEFINITIONパッケージは、表をオンラインで再定義して、属性クラスタリングを追加できるようにします。

関連項目:

DBMS_REDEFINITIONパッケージの詳細は、Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンスを参照してください

例13-5 属性クラスタリングされた表のオンライン再定義

sales表を再定義して、amount_soldのデータ型をnumberからfloatに変更し、表に属性クラスタリングを追加し、オンライン再定義時にデータをクラスタリングする場合を考えます。

SHスキーマのsales表を再定義し、表のオンライン再定義の際に表のデータをクラスタリングするには、次のステップを使用します。

1. DBMS_REDEFINITIONパッケージのCAN_REDEF_TABLEプロシージャを起動して、表をオンラインで再定義できることを確認します。

   次のコマンドは、sales表がオンラインで再定義できることを確認します。

   ```
   exec DBMS_REDEFINITION.CAN_REDEF_TABLE('SH','SALES');
   ```

2. 再定義される表で使用する物理属性と論理属性を持つ仮表を、SHスキーマで作成します。

   次のコマンドは、仮表sales_interimを作成します。amount_sold列のデータ型はbinary_doubleで、CLUSTERING句で属性クラスタリングの実行方法を指定します。

   ```
   CREATE TABLE sales_interim
   (
    PROD_ID        NUMBER(6) PRIMARY KEY,
    CUST_ID        NUMBER NOT NULL,
    TIME_ID        DATE NOT NULL,
    CHANNEL_ID     CHAR(1) NOT NULL,
    PROMO_ID       NUMBER(6),
   ```

```
 QUANTITY_SOLD  NUMBER(3) NOT NULL,
 AMOUNT_SOLD    binary_double
)
CLUSTERING sales_interim JOIN customers ON
          (sales_interim.cust_id = customers.cust_id)
                JOIN products ON (sales_interim.prod_id = products.prod_id)
    BY INTERLEAVED ORDER ( (prod_category, prod_subcategory),
                           (country_id, cust_state_province, cust_city));
```

3. DBMS_REDEFINITON.START_REDEF_TABLEプロシージャを使用して、表のオンライン再定義プロセスを開始します。このプロセスの最中も、sales表に対する問合せとDMLは可能です。

   次のコマンドは、SALES表の再定義プロセスを開始します。

```
exec DBMS_REDEFINITION.START_REDEF_TABLE(uname => 'SH',orig_table  => 'SALES', int_table =>
'SALES_INTERIM', options_flag => DBMS_REDEFINITION.CONS_USE_ROWID);
```

4. オプションで、仮表を元の表と同期化します。

   再定義の開始後に、元の表の上で多数のDML文が実行された可能性がある場合、同期をお薦めします。このステップにより、再定義プロセスを完了するための時間が短縮されます。

   次のコマンドは、sales_interim表を元のsales表と同期します。

```
exec DBMS_REDEFINITION.SYNC_INTERIM_TABLE('SH', 'SALES', 'SALES_INTERIM');
```

5. DBMS_REDEFINITION.FINISH_REDEF_TABLEプロシージャを使用して、表のオンライン再定義を完了します。

   次のコマンドは、sales表のオンライン再定義を完了します。

```
exec DBMS_REDEFINITION.FINISH_REDEF_TABLE('SH', 'SALES', 'SALES_INTERIM');
```

13.3 属性クラスタリング情報

Oracle Databaseには、属性クラスタリングに関する情報を含むデータ・ディクショナリ・ビュー一式が用意されています。この項では、それらのビューを使用して属性クラスタリングに関する情報を取得する方法を説明します。

この項では、次の項目について説明します。

- 表に対して属性クラスタリングが定義されているかどうかの判断
- 表の属性クラスタリング情報の表示
- 属性クラスタリングが実行される列に関する情報の表示
- 属性クラスタリングが実行されるディメンションと結合に関する情報の表示

13.3.1 表に対して属性クラスタリングが定義されているかどうかの判断

表に対して属性クラスタリングが定義されているかどうかは、ビューDBA_TABLES、USER_TABLESおよびALL_TABLESのCLUSTERING列で指定されています。表に属性クラスタリングが定義されている場合、CLUSTERING列がYESと表示され、それ以外の場合はNOと表示されます。

次の問合せは、SHスキーマの表の名前を表示し、属性クラスタリングを使用しているかどうかを示します。

```
SELECT TABLE_NAME, CLUSTERING FROM DBA_TABLES WHERE OWNER='SH';

TABLE_NAME      CLUSTERING
```

```
-----------    ------------
SALES             YES
PRODUCTS          NO
MY_SALES          YES
```

13.3.2 表の属性クラスタリング情報の表示

表の属性クラスタリングの詳細を取得するには、次のいずれかのデータ・ディクショナリ・ビューを使用します。

- DBA_CLUSTERING_TABLES - データベースにあるすべての属性クラスタリングされた表を記述します。
- ALL_CLUSTERING_TABLES - ユーザーにアクセスできる、属性クラスタリングされた表を記述します。
- USER_CLUSTERING_TABLES - ユーザーが所有する、属性クラスタリングされた表を記述します。

次の問合せは、SALES表の属性クラスタリングの詳細を表示します。この詳細には、属性クラスタリングのタイプと、その表に対してクラスタリングが有効化されている操作が含まれます。出力はページに収まるようにあらかじめ折り返されています。

```
SELECT owner, table_name, clustering_type, on_load, on_datamovement, with_zonemap
FROM DBA_CLUSTERING_TABLES WHERE table_name='SALES';

OWNER    TABLE_NAME CLUSTERING_TYPE  ON_LOAD  ON_DATAMOVEMENT  WITH_ZONEMAP
------   ---------- ---------------  -------- ---------------  -------------
SH       SALES       LINEAR          YES      YES              YES

SELECT owner, table_name, clustering_type, on_load, on_datamovement
FROM DBA_CLUSTERING_TABLES WHERE table_name='SALES';

OWNER    TABLE_NAME CLUSTERING_TYPE  ON_LOAD  ON_DATAMOVEMENT
------   ---------- ---------------  -------- ---------------
SH       SALES       LINEAR          YES      YES
```

13.3.3 属性クラスタリングが実行される列に関する情報の表示

表に対して属性クラスタリングが定義されている列に関する情報を取得するには、次のデータ・ディクショナリ・ビューの1つを使用します。

- DBA_CLUSTERING_KEYS
- ALL_CLUSTERING_KEYS
- USER_CLUSTERING_KEYS

たとえば、表SALESのデータは、線形順序を使用してクラスタリングされています。表がクラスタリングされている列を表示するには、次のコマンドを使用します。出力はページに収まるようにあらかじめ折り返されています。

```
SELECT detail_owner, detail_name, detail_column, position
FROM DBA_CLUSTERING_KEYS
WHERE table_name='SALES';

DETAIL_OWNER    DETAIL_NAME       DETAIL_COLUMN      POSITION
------------   --------------   -----------------   ---------
SH             SALES             PROD_ID               2
SH             SALES             TIME_ID               1
```

13.3.4 属性クラスタリングが実行されるディメンションと結合に関する情報の表示

ファクト表がクラスタリングされているディメンション表に関する情報を表示するには、DBA_CLUSTERING_DIMENSIONS、ALL_CLUSTERING_DIMENSIONSまたはUSER_CLUSTERING_DIMENSIONSデータ・ディクショナリ・ビューを問い合せます。

ファクト表とディメンション表の結合に関する詳細を表示するには、DBA_CLUSTERING_JOINS、ALL_CLUSTERING_JOINSまたはUSER_CLUSTERING_JOINSビューを問い合せます。出力はページに収まるようにあらかじめ折り返されています。

次の問合せは、ファクト表SALESが属性クラスタリングされているディメンション表を表示します。

```
SELECT * FROM DBA_CLUSTERING_DIMENSIONS WHERE table_name='MY_SALES';

OWNER      TABLE_NAME      DIMENSION_OWNER        DIMENSION_NAME
------   --------------   -------------------  ---------------------
SH         MY_SALES           SH                    PRODUCTS
```

次の問合せは、ファクト表my_salesとディメンション表productsを結合するために使用される列を表示します。出力はページに収まるようにあらかじめ折り返されています。

```
SELECT tab1_owner, tab1_name,tab1_column
FROM DBA_CLUSTERING_JOINS
WHERE table_name='MY_SALES';

TAB1_OWNER       TAB1_NAME     TAB1_COLUMN
-----------   -------------   --------------------
SH               MY_SALES        PROD_ID
```

14 ゾーン・マップの使用

ゾーン・マップは、表に対して作成できる、独立したアクセス構造です。表スキャンおよび索引スキャンにゾーン・マップを使用すると、表列上の述語に基づいて、表のディスク・ブロックや、パーティション表の完全なパーティションをプルーニングできる可能性があります。ゾーン・マップは、属性クラスタリングの有無にかかわらず使用できます。

この章の内容は次のとおりです。

- ゾーン・マップについて
- ゾーン・マップの操作
- ゾーン・マップのリフレッシュと失効
- ゾーン・マップを使用したプルーニングの実行
- ゾーン・マップ情報の表示

14.1 ゾーン・マップについて

ゾーン・マップは表に対して作成される独立したアクセス構造で、表のゾーンに関する情報が格納されます。ゾーン・マップを使用すると、表列上の述語に適合しないデータ・ブロックをデータベースがプルーニングできます。**ゾーン**とは、ディスク上で連続している一群のデータ・ブロックです。

従来のゾーン・マップは、ディスク・ユニット、ブロックまたはエクステントごとに表内の列の最小値と最大値を格納します。問合せがクラスタリング列を修飾する場合、I/Oプルーニングが発生します。Oracle Databaseのゾーン・マップは、一定範囲のブロック(ゾーンと呼ばれます)について、列の最小値と最大値を格納します。ファクト表がディメンション表との外部結合を介してディメンション属性により属性クラスタリングされていれば、クラスタリングされたファクト表の述語に基づくI/Oプルーニングの実行に加え、ゾーン・マップはディメンション表の述語上でもプルーニングします。

1つの表につき最大1つのゾーン・マップを定義できます。パーティション表の場合、すべてのパーティション(およびサブパーティション)について1つのゾーン・マップが存在します。パーティション表のゾーン・マップは、ゾーンごと、パーティションごと、サブパーティションごとの最小値と最大値を追跡します。表にディメンション表との外部結合がある場合、ゾーン・マップ定義にディメンション列の最小値と最大値を含めることができます。

この項では、次の項目について説明します。

- ゾーン・マップと索引の違い
- ゾーン・マップと属性クラスタリング
- ゾーン・マップのタイプ
- ゾーン・マップの利点
- ゾーン・マップが有効なシナリオ
- ゾーン・マップのメンテナンスについて

14.1.1 ゾーン・マップと索引の違い

ゾーン・マップは、粗い索引構造のようなものです。ただし、索引とは次のような根本的な違いがあります。

- ゾーン・マップには、行ごとではなくゾーンごとの情報が格納されます。このため、索引よりはるかにコンパクトです。
- 索引はDML処理との同期を維持しますが、ゾーン・マップは、そのようにアクティブには管理されません。このため、たとえ

ゾーン・マップにREFRESH ON COMMITオプションが付けられていても、コミットまたはロールバックが発生するまで、トランザクション範囲内では失効していることがあります。

- ゾーン・マップには、あるゾーンに関する失効した情報と、残りのゾーンに関する新しい情報が混在することがありますが、その場合もOracle Databaseでは、ファクト表のスキャン時に、そのゾーン・マップを使用してI/Oプルーニングを実行します。

14.1.2 ゾーン・マップと属性クラスタリング

属性クラスタリングは、ゾーン・マップにとって必須の前提条件ではありません。ゾーン・マップは、属性クラスタリングの有無にかかわらず使用できます。このため、ゾーン・マップがなくても属性クラスタリングを指定でき、クラスタリングがなくても表上でゾーン・マップを作成できます。

データ・ウェアハウジング環境では、ETL処理のデータを合理的にクラスタリングしているのが普通です(たとえば、時間列や地理的リージョンによるクラスタリング)。クラスタリングにより、列の最小値と最大値が、属性クラスタリングされた表内の連続的なデータ・ブロックと相関する傾向が高まるため、ゾーン・マップを使用するプルーニングの効率が向上します。ゾーン・マップを使用すると、属性クラスタリングにより実行されるデータの順序付けを利用できるため、プルーニングが効率化します。表スキャンと索引スキャンの際(たとえば、rowidによるフェッチ)、ゾーン・マップを使用すると、表列の述語に適合しないデータ・ブロックもプルーニングできます。

関連項目:

属性クラスタリングの詳細は、属性クラスタリングについてを参照してください。

14.1.3 ゾーン・マップのタイプ

ゾーン・マップには、次の2つのタイプがあります。

- 基本的なゾーン・マップは単一の表の上で定義され、この表の一部の列の最小値と最大値を保持します。
- 結合ゾーン・マップは、1つ以上の別の表への外部結合を持っており、その別の表内の一部の列の最小値と最大値を保持している表の上で定義されます。これらの結合条件はマスターと詳細の関係でも、ファクト表とディメンション表の間のスター・スキーマでも一般的です。

  スター・クエリーの場合、複数のディメンション表が、ファクト表とのPK-FK関係を介して結合されます。ここで、結合ゾーン・マップは、ファクト表のゾーンに対応するディメンション表の列の最小値と最大値を維持します。

14.1.4 ゾーン・マップの利点

- 表または表パーティションの順次スキャンまたは索引スキャン時のI/Oを削減
- ゾーン・マップ列がパーティション化キーと相関している場合に、パーティション表やコンポジット・パーティション表の非キー列に基づくパーティション・プルーニングが可能
- インターリーブ順序による属性クラスタリングで、クラスタリング列のサブセットにおけるI/O削減を可能にします。
- 索引の使用に関連する記憶域コストが不要

関連項目:

14.1.5 ゾーン・マップが有効なシナリオ

ゾーン・マップは、次のシナリオで有効です。

- 表スキャンが、使用頻度の高い述語を使用して実行される

  ゾーン・マップを使用すると、列述語によって除外されたゾーンをOracle Databaseがスキャンせずにすみます。

- 結合が、ディメンション階層列上で使用頻度の高い述語を使用して、ファクト表とディメンション表の間で定義される

  ファクト表の行を、属性値上の述語によって除外されるゾーンをプルーニングしつつ、ディメンション属性値により順序付けることができます。

- パーティション表内の列に、パーティション・キーと相関する値が含まれる

  これにより、非キー列に基づくパーティション・プルーニングが容易になります。たとえば、日付でパーティション化されている表には、パーティション・キーとよく相関する他の日付列や、時間の経過に応じて変化または循環する連続値を含む列がしばしばあります。

- データ・クラスタリングがゾーン・マップ列の値に対して実行される

  属性クラスタリングは、特にこの目的のために設計されています。あるいは、データに内在する順序を利用するのが適切です(たとえば、順次的にロードされる、時間に基づく列値や、ロード時にソートされるデータ)。

- 頻度が高くカーディナリティが低い索引範囲スキャンが、表の上で実行される

  属性クラスタリングを単独で使用して、圧縮係数を改善できます。ゾーン・マップを使用すると、除外されたゾーンからの参照をプルーニングすることによって、索引スキャンの効率を改善できます。あるいは、ゾーン・マップを索引のかわりに使用できます。

14.1.6 ゾーン・マップのメンテナンスについて

ゾーン・マップは表に基づいているため、基礎になる表に対する変更はなんであれ、ゾーン・マップの状態に影響します。表の上で実行される操作により異なりますが、ゾーン・マップの一部またはすべてのゾーンが影響を受けます。基礎になる表の変更により影響がおよぶゾーン・マップにはメンテナンスが必要です。

ゾーン・マップのメンテナンスは、次のうち1つ以上の作業で構成されます。

- 影響を受けるゾーン・マップの有効性チェック
- 影響を受けるゾーンの失効追跡
- 影響を受ける失効済ゾーン・マップのリフレッシュ(ゾーン・マップに対して設定されているリフレッシュ・モードにより異なる)

ゾーン・マップが基づいている実表の構造が変更された場合(たとえば、最小値と最大の値がゾーン・マップによって維持されている表列が削除されたなどの場合)、ゾーン・マップは**無効**になります。無効なゾーン・マップは問合せで使用されず、そのゾーン・マップはOracle Databaseではメンテナンスされません。ただし、ゾーン・マップと関連していない実表の構造が変更された場合(たとえば、表に新規の列が追加された場合など)、ゾーン・マップは有効なままですが、コンパイルが必要です。Oracle Databaseは、それ以降の操作(問合せによるゾーン・マップの使用など)の際に、ゾーン・マップを自動的にコンパイルします。または、ALTER MATERIALIZED ZONEMAPコマンドのCOMPILE句を使用してゾーン・マップをコンパイルすることもできます。

関連項目:

基礎になる表のデータが変更された場合、その変化の影響を受けるゾーンは**「失効」**とマークされます。失効したゾーン・マップでは、データだけは最新ではありませんが、定義はまだ有効です。Oracle Databaseは、様々なタイプの操作による、基礎になる表上でのゾーン・マップの失効を自動的に追跡します。基礎になる表の上で実行される操作のタイプに応じて、Oracle Databaseは、ゾーン・マップ全体を失効とマークするか、ゾーン・マップ内の一部のゾーンのみを失効とマークします。

この項では、次の項目について説明します。

- ゾーン・マップのメンテナンスを必要とする操作
- ゾーン・マップが自動的にリフレッシュされるシナリオ

14.1.6.1 ゾーン・マップのメンテナンスを必要とする操作

ゾーン・マップのメンテナンスが必要になるのは、基礎になる表の1つ以上に対して次の操作が行われる場合です。

- DML(挿入、削除、更新、従来型ロード)。
- 直接パス挿入およびロード。
- パーティションのメンテナンス操作(MOVE、SPLIT、MERGE、DROP、TRUNCATEおよびEXCHANGE)、表データの移動、および表のオンライン再定義。

14.1.6.2 ゾーン・マップが自動的にリフレッシュされるシナリオ

前述の操作の影響を受けたゾーン・マップをOracle Databaseが自動的にリフレッシュするかどうかは、ゾーン・マップ・リフレッシュ・モードにより決定されます。

Oracle Databaseでは、次のものに影響を受けるゾーン・マップについて自動リフレッシュが実行されます。

- リフレッシュ・モードがREFRESH ON COMMITの場合、DML操作。REFRESH ON COMMITモードのゾーン・マップは、トランザクション的には最新のままです。リフレッシュは、トランザクションがコミットされたときに実行されます。
- リフレッシュ・モードがREFRESH ON LOADの場合、直接パス挿入またはロード。

  REFRESH ON LOADで構成されているゾーン・マップは、基礎になる表の上でのDMLまたはPMOP操作により失効することがあります。
- リフレッシュ・モードがREFRESH ON DATA MOVEMENTである場合、PMOP(MOVE、SPLIT、MERGE、DROP)または表の移動。

  REFRESH ON DATA MOVEMENTで構成されているゾーン・マップは、DML、直接パス挿入またはロード、PMOP(TRUNCATE、EXCHANGE)または基礎になる表のオンライン再定義の後、失効することがあります。
- リフレッシュ・モードがREFRESH ON DATA MOVEMENTである場合の、直接パス挿入またはロード、PMOP(MOVE、SPLIT、MERGE、DROP)または表の移動。

  REFRESH ON LOAD DATA MOVEMENTで構成されているゾーン・マップは、DML、PMOP(TRUNCATE、EXCHANGE)または基礎になる表のオンライン再定義の後、失効することがあります。

リフレッシュ・モードがREFRESH ON DEMANDの場合、Oracle Databaseは基礎になる表でのなんらかの操作に影響を受けるゾーン・マップを自動リフレッシュしません。REFRESH ON DEMANDが設定されているゾーン・マップは、手動でリフレッシュする必要があります。

関連項目:

- ゾーン・マップのリフレッシュと失効
- ゾーン・マップのメンテナンス

14.2 ゾーン・マップの操作

この項では、ゾーン・マップを含む一般的なタスクについて、次のように分けて説明します。

- ゾーン・マップに必要な権限
- ゾーン・マップの作成
- ゾーン・マップの変更
- ゾーン・マップの削除
- ゾーン・マップのコンパイル
- ゾーン・マップ使用の制御
- ゾーン・マップのメンテナンス

14.2.1 ゾーン・マップに必要な権限

- 独自のスキーマのゾーン・マップを作成、変更または削除するには、CREATE MATERIALIZED ZONEMAP権限を持っている必要があります。
- 他のスキーマのゾーン・マップを作成するには、CREATE ANY MATERIALIZED ZONEMAP権限を持っている必要があります。
- 別のスキーマの表の上に、独自のスキーマのゾーン・マップを作成するには、SELECT ANY TABLEまたはREAD ANY TABLE権限を持っている必要があります。
- 他のスキーマの表を使用して他のスキーマのゾーン・マップを作成するには、SELECT ANY TABLEおよびCREATE ANY MATERIALIZED ZONEMAP権限を持っている必要があります。SELECT ANY TABLE権限のかわりにREAD ANY TABLE権限を持つこともできます。
- 他のスキーマのゾーン・マップを変更するには、ALTER ANY MATERIALIZED ZONEMAP権限を持っている必要があります。
- 他のスキーマのゾーン・マップを削除するには、DROP ANY MATERIALIZED ZONEMAP権限を持っている必要があります。

14.2.2 ゾーン・マップの作成

ゾーン・マップは属性クラスタリングとともに表の上に作成できますが、ゾーン・マップは属性クラスタリングから独立しています。ゾーン・マップは、属性クラスタリングにかかわりなく、独立して作成できます。

ゾーン・マップが使用する記憶域の構造は、それが定義される表のデフォルト表領域に作成されます。

関連項目:

- ゾーン・マップ作成の構文の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください
- ゾーン・マップの制限事項の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

この項では、次の項目について説明します。

- 属性クラスタリングによるゾーン・マップの作成
- 属性クラスタリングから独立したゾーン・マップの作成

14.2.2.1 属性クラスタリングによるゾーン・マップの作成

WITH MATERIALIZED ZONEMAP副次句を使用して、ゾーン・マップを作成できます。表に属性クラスタリングを定義する場合、または、後でクラスタリング定義を変更する場合、この副次句を使用できます。

属性クラスタリングを使用してゾーン・マップを作成するには、次のトピックで説明されているステップを使用します。

- 線形属性クラスタリングによる基本的なゾーン・マップの作成
- インターリーブ属性クラスタリングによる結合ゾーン・マップの作成
- 属性クラスタリング後のゾーン・マップ作成

関連項目:

属性クラスタリングの詳細は、属性クラスタリングを参照してください。

14.2.2.1.1 線形属性クラスタリングによる基本的なゾーン・マップの作成

salesの問合せで、顧客ID、または顧客IDと製品IDの組合せがよく指定される事態を想定します。問合せでゾーン・マップを使用したプルーニングを活用できるように、属性クラスタリングされた表を作成することができます。次のような表を作成するとします。

```
CREATE TABLE sales (
   prod_id NUMBER NOT NULL,
   cust_id NUMBER NOT NULL,
   time_id        DATE NOT NULL,
   channel_id     NUMBER NOT NULL,
   promo_id       NUMBER NOT NULL,
   quantity_sold  NUMBER(10,2),
   amount_sold NUMBER(10,2)
)
CLUSTERING
BY LINEAR ORDER (cust_id, prod_id)
YES ON LOAD YES ON DATA MOVEMENT
WITH MATERIALIZED ZONEMAP;
```

列(cust_id、prod_id)上にゾーン・マップZMAP$_SALESが作成されます。ここで、ZMAP$_SALESは、Oracle Databaseによって自動的に生成されるゾーン・マップの名前です。ただし、インターリーブ属性クラスタリングによる結合ゾーン・マップの作成で説明するとおり、WITH MATERIALIZED ZONEMAPに続けてカッコに入れることでゾーン・マップの名前を指定できます。

列cust_idとprod_idの両方、または接頭辞cust_idを修飾する問合せは、自然プルーニングを受けます。次の例は、表スキャン時のデータベースのプルーニングがどのように可能になるかを示しています。

アプリケーションは、次の問合せを発行します。

```
SELECT * FROM sales WHERE cust_id = 100;
```

表はBY LINEAR ORDERでクラスタリングされているため、データベースはcust_idの値に100が含まれるゾーンのみ読み取る必要があります。

アプリケーションは、次の問合せを発行します。

```
SELECT * FROM sales WHERE cust_id = 100 AND prod_id = 2300;
```

表はBY LINEAR ORDERでクラスタリングされているため、データベースはcust_idの値に100、prod_idの値に2300が含まれるゾーンのみ読み取る必要があります。

14.2.2.1.2 インターリーブ属性クラスタリングによる結合ゾーン・マップの作成

salesファクト表と、その2つのディメンション表、customersおよびproductsを含むデータ・ウェアハウスを想定します。大部分の問合せは、customers表階層(country_id, cust_state_province, cust_city)およびproducts階層(prod_category, prod_subcategory)上の述語を持っています。次の部分文に示すように、sales表のインターリーブ順序を使用できます。

```
CREATE TABLE sales (
   prod_id NUMBER NOT NULL,
   cust_id NUMBER NOT NULL,
   amount_sold NUMBER(10,2))
CLUSTERING
sales JOIN products ON (sales.prod_id = products.prod_id)
JOIN customers ON (sales.cust_id = customers.cust_id)
BY INTERLEAVED ORDER
(
(products.prod_category, products.prod_subcategory),
(customers.country_id, customers.cust_state_province, customers.cust_city)
)
YES ON LOAD YES ON DATA MOVEMENT
WITH MATERIALIZED ZONEMAP (sales_zmap);
```

属性クラスタリングされた表に、sales_zmapと呼ばれるゾーン・マップが作成されます。このクラスタリング句では、ディメンション表の結合列に主キーまたは一意キーの制約がある必要があります。1つのディメンションからのインターリーブ順序の列の場合、たとえば(prod_category, prod_subcategory)のように「(」と「)」で囲んだ別のグループがクラスタリング句になければならないことに注意してください。さらに、列はディメンション内の階層に従う必要があり(たとえばprod_category, prod_subcategoryの自然階層)、グループ内の列の順序は階層内のものに従う必要があります。これにより、ディメンション表に存在する階層に応じて、データが効果的にクラスタリングされます。

14.2.2.1.3 属性クラスタリング後のゾーン・マップ作成

データベースにsalesと呼ばれている表が存在すると想定します。次のコマンドを使用して、sales表に属性クラスタリングを定義することができます。

```
ALTER TABLE sales ADD CLUSTERING BY INTERLEAVED ORDER (cust_id, prod_id)
      YES ON LOAD YES ON DATA MOVEMENT;
```

このコマンドは、表定義に属性クラスタリングを追加しますが、sales表内の既存のデータはクラスタリングしません。sales表上でデータ移動操作を実行すると、YES ON DATA MOVEMENTオプションがあるため、そのデータはクラスタリングされます。

次のコマンドは、sales表内のデータをクラスタリングします。

```
ALTER TABLES sales MOVE;
```

sales表内のデータがクラスタリングされた後、次のコマンドを使用してクラスタリングを変更することによって、sales表上にゾーン・マップを定義できます。

```
ALTER TABLE sales MODIFY CLUSTERING WITH MATERIALIZED ZONEMAP (sales_zmap);
```

その後、必要なら、次のコマンドを使用してクラスタリングを変更することによってゾーン・マップを削除できます。

```
ALTER TABLE sales MODIFY CLUSTERING WITHOUT MATERIALIZED ZONEMAP;
```

14.2.2.2 属性クラスタリングから独立したゾーン・マップの作成

表上にゾーン・マップを作成するには、CREATE MATERIALIZED ZONEMAPコマンドを使用します。このゾーン・マップは属性クラスタリングから独立しています。これは、クラスタリングされた表にもクラスタリングされていない表にも作成できるということです。また、ゾーン・マップに使用される列の組合せは、属性クラスタリングに使用される列の組合せと同じでも異なっていてもかまいません。

ゾーン・マップを作成する場合、ゾーン・マップが基づく表の列を指定する必要があります。

属性クラスタリングから独立したゾーン・マップを作成するには、次のトピックで説明されているステップを使用します。

- 属性クラスタリングから独立した基本ゾーン・マップの作成
- 属性クラスタリングから独立した結合ゾーン・マップの作成

14.2.2.2.1 属性クラスタリングから独立した基本ゾーン・マップの作成

sales表上の問合せで、顧客ID、製品ID、またはその2つの列の組合せがよく指定されると想定します。例14-1に示すように、プルーニングが問合せを効率化できるように、sales表の顧客ID列と製品ID列の上にゾーン・マップを作成できます。

例14-1 属性クラスタリングから独立した基本ゾーン・マップの作成

次の文を使用して、sales表の上にゾーン・マップsales_zmapを作成できます。

```
CREATE MATERIALIZED ZONEMAP sales_zmap ON sales (cust_id, prod_id);
```

この文は、次のCREATE...AS文と同じです。

```
CREATE MATERIALIZED ZONEMAP sales_zmap
REFRESH ON LOAD DATA MOVEMENT
AS
SELECT SYS_OP_ZONE_ID(rowid),MIN(cust_id),MAX(cust_id),MIN(prod_id),MAX(prod_id)
FROM sales
GROUP BY SYS_OP_ZONE_ID(rowid);
```

この文では、ゾーン・マップで使用するSYS_OP_ZONE_ID(rowid)ファンクションが使用されています。SYS_OP_ZONE_IDファンクションは、ファクト表行のrowidが指定されると、隣接するディスク・ブロック(ゾーン)の個別の範囲を特定します。このファンクションは、ゾーン・マップのパーティション・プルーニングと高速リフレッシュを実行して、パーティション・レベルで最小値と最大値の範囲を維持する助けになります。ゾーン・マップで使用する場合、隣接するデータ・ブロック群から単一のゾーンにすべての行をマップすると便利です。

14.2.2.2.2 属性クラスタリングから独立した結合ゾーン・マップの作成

salesファクトと複数のディメンションを含むデータ・ウェアハウスを想定します。大部分の問合せは、customers表階層(cust_state_province, cust_city)上の述語を持っています。例14-2に示すように、sales表ではインターリーブ順序を使用できます。

例14-2 属性クラスタリングから独立した結合ゾーン・マップの作成

結合ゾーン・マップには、ゾーン・マップが作成されている表から、他の1つ以上の表への外部結合が含まれます。次の文に示すように、結合ゾーン・マップはスター・スキーマ設定で最も一般的に使用され、ファクト表の列でなく、ディメンション表の列の最小値と最大値を追跡します。

```
CREATE MATERIALIZED ZONEMAP sales_zmap
REFRESH ON LOAD DATA MOVEMENT
AS
SELECT SYS_OP_ZONE_ID(s.rowid), MIN(cust_state_province),
       MAX(cust_state_province), MIN(cust_city), MAX(cust_city)
FROM sales s, customers c
WHERE s.cust_id = c.cust_id(+)
```

```
GROUP BY SYS_OP_ZONE_ID(s.rowid);
```

14.2.3 ゾーン・マップの変更

ALTER MATERIALIZED ZONEMAP文を使用してゾーン・マップを変更できます。

例14-3 ゾーン・マップの使用不能化

次の文は、ゾーン・マップを使用不能にします。これは、問合せがこのゾーン・マップをもう使用せず、Oracle Databaseによるゾーン・マップのメンテナンスも行われないことを意味します。

```
ALTER MATERIALIZED ZONEMAP sales_zmap UNUSABLE;
```

例14-4 ゾーン・マップの完全リフレッシュの実行

次の文は、ゾーン・マップの完全リフレッシュを実行します。

```
ALTER MATERIALIZED ZONEMAP sales_zmap REBUILD COMPLETE;
```

ゾーン・マップがそれ以前に使用不能とマークされている場合、再構築の過程で使用可能にされます。

例14-5 ゾーン・マップのリフレッシュ

次の文は、可能な場合、高速リフレッシュを実行します。それ以外の場合は完全リフレッシュを実行します。

```
ALTER MATERIALIZED ZONEMAP sales_zmap REBUILD;
```

例14-6 ゾーン・マップのプルーニングの無効化

次の文はプルーニングを無効にします。これはパフォーマンス測定の際に必要になることがあります。

```
ALTER MATERIALIZED ZONEMAP sales_zmap DISABLE PRUNING;
```

例14-7 ゾーン・マップのプルーニングの有効化

次の文は、以前に無効化されている可能性があるゾーン・マップのプルーニングを有効化します。

```
ALTER MATERIALIZED ZONEMAP sales_zmap ENABLE PRUNING;
```

例14-8 ゾーン・マップのリフレッシュの無効化

次の文は、ロード時のリフレッシュとデータ移動をオフにし、それにより、ゾーン・マップがリフレッシュされる方法とタイミングを制御できるようにします。

```
ALTER MATERIALIZED ZONEMAP sales_zmap REFRESH ON DEMAND;
```

例14-9 コミット時のゾーン・マップ・リフレッシュの有効化

次の文は、各トランザクションのコミット時におけるゾーン・マップのリフレッシュをオンにします。

```
ALTER MATERIALIZED ZONEMAP sales_zmap REFRESH ON COMMIT;
```

関連項目:

- ゾーン・マップ変更の構文については、『*Oracle Database SQL言語リファレンス*』を参照してください。

14.2.4 ゾーン・マップの削除

次のように、DROP MATERIALIZED ZONEMAP文を発行することによってゾーン・マップを削除できます。

```
DROP MATERIALIZED ZONEMAP sales_zmap;
```

関連項目:

ゾーン・マップ削除の構文については、『*Oracle Database SQL言語リファレンス*』を参照してください。

14.2.5 ゾーン・マップのコンパイル

ゾーン・マップが基づいている実表に関するDDL操作はすべて、ゾーン・マップのコンパイル状態に影響を及ぼします。このため、ゾーン・マップがまだ有効かどうか確認するには、ゾーン・マップを定義する問合せをコンパイルする必要があります。この動作はマテリアライズド・ビューと類似しています。それも実表で実行されるDDLの影響を受けるためです。Oracle Databaseは、DDL操作後、ゾーン・マップを初めて使用しようとするときにコンパイルします。ただし、次のようなDDL変更文を使用して、ゾーン・マップを明示的にコンパイルすることができます。

```
ALTER MATERIALIZED ZONEMAP sales_zmap COMPILE;
```

ゾーン・マップのコンパイル結果は、DDLによって実行された個別処理に応じて、有効か無効のいずれかです。たとえば、ファクト表に列を追加するためにDDLが実行された場合、ゾーン・マップはコンパイル後に有効になります。一方、定義問合せで参照された列を削除するためにDDLが実行された場合、ゾーン・マップはコンパイル後に無効になります。

注意すべきポイントは次のとおりです。

- クラスタリング句に出現する列が削除されると、クラスタリングは削除されます。また、クラスタリングの一部として作成されたゾーン・マップがあれば、そのゾーン・マップも削除されます。
- スター・スキーマからのディメンション表が削除され、それがファクト表上のクラスタリングに含まれていた場合、そのファクト表のクラスタリングは削除されます。また、クラスタリングの一部として作成されたゾーン・マップがあれば、そのゾーン・マップは削除されます。
- ユーザーがクラスタリング句に含まれるディメンション表上の必須の主キーまたは一意キーを削除した場合、クラスタリングは無効にされます(一部のタイプのPMOPがそれ以降にロードまたはデータ移動操作を実行しても、データのクラスタリングが行われません)。

14.2.6 ゾーン・マップ使用の制御

SQLワークロード全体または特定のSQL文に対してゾーン・マップの使用を制御することができます。

この項では、次の項目について説明します。

- SQLワークロード全体に対するゾーン・マップ使用の制御
- 特定のSQL文に対するゾーン・マップ使用の制御

14.2.6.1 SQLワークロード全体に対するゾーン・マップ使用の制御

オブジェクト・レベルでゾーン・マップの使用を制御できます。オブジェクト・レベルの変更は、SQLワークロードのすべての文に適用されます。ゾーン・マップを作成すると、DISABLE PRUNINGを指定してデフォルトをオーバーライドしないかぎり、プルーニングに利用できます。たとえば、次の文は、プルーニング無効でゾーン・マップを作成します。

```
CREATE MATERIALIZED ZONEMAP sales_zmap
    DISABLE PRUNING ON sales(cust_id, prod_id);
```

このゾーン・マップは、Oracle Databaseによって作成、メンテナンスされますが、ワークロード内のいかなるSQLでも使用されません。次のALTER MATERIALIZED ZONEMAP文を使用すると、プルーニングで利用できるようにできます。

```
ALTER MATERIALIZED ZONEMAP sales_zmap ENABLE PRUNING;
```

同様に、次の文を使用すると、ゾーン・マップをプルーニングに利用できないようにできます。

```
ALTER MATERIALIZED ZONEMAP sales_zmap DISABLE PRUNING;
```

14.2.6.2 特定のSQL文に対するゾーン・マップ使用の制御

ヒントを使用して、個別のSQL文レベルでゾーン・マップの使用を制御できます。ゾーン・マップに対してプルーニングが無効にされている場合、ヒントを使用してゾーン・マップの使用を制御できないことに注意してください。プルーニングは有効なままにしておき、個別のSQL文で否定的なヒントを指定することで、ヒントを介してより細かく制御できます。

プルーニングによるゾーン・マップの使用を無効にするには、NO_ZONEMAPヒントを使用します。次の例は、データをプルーニングしますが、ゾーン・マップの使用を無効にします。

例14-10 スキャン・プルーニング: NO_ZONEMAPヒントを使用したゾーン・マップの無効化

```
SELECT /*+ NO_ZONEMAP (S SCAN) */* FROM sales S
WHERE s.time_id BETWEEN '1-15-2008' AND '1-31-2008';
```

例14-11 結合プルーニング: NO_ZONEMAPヒントを使用したゾーン・マップの無効化

```
SELECT /*+ NO_ZONEMAP (S JOIN) */* FROM sales s
WHERE s.time_id BETWEEN '1-15-2008' AND '1-31-2008';
```

例14-12 パーティション・プルーニング: NO_ZONEMAPヒントを使用したゾーン・マップの無効化

```
SELECT /*+ NO_ZONEMAP (S PARTITION) */* FROM sales S
WHERE s.time_id BETWEEN '1-15-2008' AND '1-31-2008';
```

14.2.7 ゾーン・マップのメンテナンス

ゾーン・マップの作成時、または後でゾーン・マップ定義を変更するときに、ゾーン・マップをメンテナンスする方法を指定できます。ゾーン・マップ・メンテナンスに関する考慮事項を参照してください。

関連項目:

ゾーン・マップのメンテナンスについて

ゾーン・マップのメンテナンス方法を指定するには、CREATE MATERIALIZED ZONEMAPまたはALTER MATERIALIZED ZONEMAP文でREFRESH句を使用します。CREATE MATERIALIZED ZONEMAP文でREFRESH句を省略する場合、使用されるデフォルトはREFRESH ON LOAD DATA MOVEMENTとなり、直接パス・ロード時と一部のデータ移動操作時の、Oracle Databaseによるゾーン・マップのメンテナンスが有効になります。

次の文は、メンテナンスをユーザーが手動で管理するゾーン・マップを作成します。

```
CREATE MATERIALIZED ZONEMAP sales_zmap
REFRESH ON DEMAND
ON sales (cust_id, prod_id);
```

次の文は、メンテナンスが各トランザクションのコミット終了時にOracle Databaseによって管理されるゾーン・マップを作成します。

```
CREATE MATERIALIZED ZONEMAP sales_zmap
REFRESH ON COMMIT
ON sales (cust_id, prod_id);
```

コミット時にリフレッシュされるため、前述のゾーン・マップは決して失効しません。

既存のゾーン・マップのメンテナンスを変更するには、ALTER MATERIALIZED ZONEMAP文を使用します。

例14-13 データ移動時におけるゾーン・マップ・メンテナンスの有効化

次の文は、Oracle Databaseによる、データ移動操作時(たとえばMOVE、SPLIT、MERGEおよびDROP)のゾーン・マップ・メンテナンスを有効にします。

```
ALTER MATERIALIZED ZONEMAP sales_zmap REFRESH ON DATA MOVEMENT;
```

例14-14 直接パス・ロード時におけるゾーン・マップ・メンテナンスの有効化

次の文は、Oracle Databaseによる直接パス・ロード操作時(INSERT /*+ APPEND */文など)のゾーン・マップ・メンテナンスを有効にします。

```
ALTER MATERIALIZED ZONEMAP sales_zmap REFRESH ON LOAD;
```

例14-15 データの移動およびロード時におけるゾーン・マップ・メンテナンスの有効化

次の文は、Oracle Databaseによる、データ移動およびロード操作時のゾーン・マップ・メンテナンスを有効にします。

```
ALTER MATERIALIZED ZONEMAP sales_zmap REFRESH ON LOAD DATA MOVEMENT;
```

REFRESH ON LOAD DATA MOVEMENTがデフォルトのオプションである点に注意してください。

14.2.7.1 ゾーン・マップ・メンテナンスに関する考慮事項

ゾーン・マップのメンテナンスと失効の追跡に関する、注意が必要な問題のいくつかを次に示します。

- ファクト表に対するDML/パラレルDML操作

  ゾーン・マップが作成されると、従来のDML操作による行の変更を追跡するための内部トリガーが作成されます。たとえば、sales表に新しい行が挿入されると、そのトリガーがrowidからzone_idを計算し、ゾーン・マップ内の対応する集計行を失効としてマークします。このため、ゾーン・マップの失効はゾーンごとに追跡されます。これは、ファクト表に対してDMLが実行された後でさえ、そのゾーン・マップが、最新のゾーンのMIN/MAX集計を使用するプルーニングに使用されることがまだあることを意味します。

  ファクト表更新時点で、更新される列がゾーン・マップによって参照されていない場合、ゾーン・マップの失効は影響を受けません。それ以外の場合、更新された行に対応するゾーンは、内部トリガーによって失効とマークされます。

- ファクト表への直接ロード(INSERT /*+ APPEND */)操作

  直接ロードが最高水位標より上にデータを挿入する場合も、新しく追加される行は、そのゾーン・マップに対してすでに計算済のゾーンに属することができます。このため、Oracle Databaseは、MIN/MAX集計が新しく追加されるデータに影響を受ける可能性のある既存のゾーンを識別し、そのようなゾーンを失効とマークします。また、まだ最新状態であるゾーンのMIN/MAX集計を利用することによって、ゾーン・マップへの直接ロードにもかかわらず、プルーニングでのゾーン・マップ利用をOracle Databaseで継続できます。ゾーン・マップにREFRESH ON LOADオプションがある場合、Oracle Databaseはロード終了時にゾーン・マップ・リフレッシュを実行します。

- ファクト表でのデータ移動(たとえば、パーティション・メンテナンス操作)

データ移動操作には、パーティション・メンテナンス操作や、パーティション/表のオンライン再定義が含まれます。ただし、データ移動(パーティション移動など)により、ゾーン・マップの古いパーティションに属する既存のゾーンは廃止とされますが、新しいパーティションに属するゾーンは、ゾーン・マップがリフレッシュされるまで計算されません。ゾーン・マップがリフレッシュされているかどうかに関係なく、Oracle Databaseはデータ移動操作後も、プルーニングのためのゾーン・マップを使用し続けます。ゾーン・マップにREFRESH ON DATA MOVEMENTオプションが付けられている場合、Oracle Databaseはデータ移動操作の終了時にリフレッシュを実行します。

- ディメンション表のデータ移動

  この操作は、ゾーン・マップに影響を及ぼしません。

- ディメンション表に対するなんらかのDML

  この操作はゾーン・マップ全体を失効させるため、完全リフレッシュが必要になります。ただし、1つ例外があります。それが更新操作であり、更新された列群がゾーン・マップによって参照されない場合、影響はありません。

- ディメンション表への直接ロード

  この操作はゾーン・マップ全体を失効させます。ゾーン・マップに対してREFRESH ON LOADオプションが指定されている場合、Oracle Databaseはロード操作の直後にゾーン・マップ・リフレッシュを実行します。

- ファクト表またはディメンション表へのDDL

  DDL操作時に、ゾーン・マップは失効不明(つまり「失効」が' unknown' に設定される)、要コンパイル(つまり、compile_stateが' needs_compileに設定される)とマークされます。この状態では、Oracle Databaseはプルーニングでゾーン・マップを使用しません。ただし、DDL操作の後でゾーン・マップを初めて使用するときにOracle Databaseはゾーン・マップをコンパイルし、その結果に基づいて、無効および失効の状態を適切に設定します。たとえば、MIN/MAX集計がゾーン・マップに格納されている列をDDL操作が削除した場合、ゾーン・マップのコンパイルは失敗するため、ゾーン・マップのcompile_stateは' compilation error' に設定され、「失効」は' unknown' のまま、「無効」は' yes' に設定されます。

14.3 ゾーン・マップのリフレッシュと失効

実表のデータが変更されると、Oracle Databaseはゾーン・マップを失効とマークするか、ゾーン・マップ内の個別のゾーンを失効とマークします。失効したゾーン・マップはプルーニングで使用されませんが、失効したゾーンがあるゾーン・マップはまだプルーニングで使用されます。ゾーン・マップをリフレッシュしてゾーンを更新し、プルーニングで使用できるようにする必要があります。

この項では、次の項目について説明します。

- ゾーン・マップの失効について
- ゾーン・マップのリフレッシュについて
- ゾーン・マップのリフレッシュ

14.3.1 ゾーン・マップの失効について

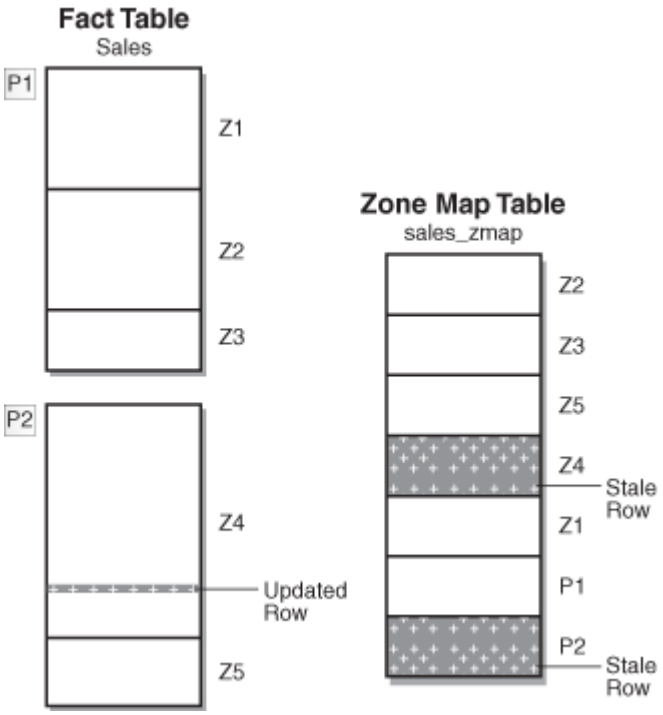
ゾーン・マップが基づいている表のデータが変更された場合、変更された行に対応するゾーンは失効とマークされます。ゾーン・マップをリフレッシュしてゾーンを最新にする必要があります。

ファクト表のパーティション内の行が更新された場合、ゾーンに対応するパーティション表内の行は、更新のため、失効とマークされます。これにより集計パーティション・レベルの情報は自動的に無効になり、プルーニングはゾーン・レベルでのみ可能になります。この特定のパーティションに対応するゾーン・マップ内の行も、更新のため、失効とマークされます。

これは図14-1に、P2のZ4における更新で示されており、対応するZ4は失効とマークされます。ただし、ゾーン・マップがまだ使用

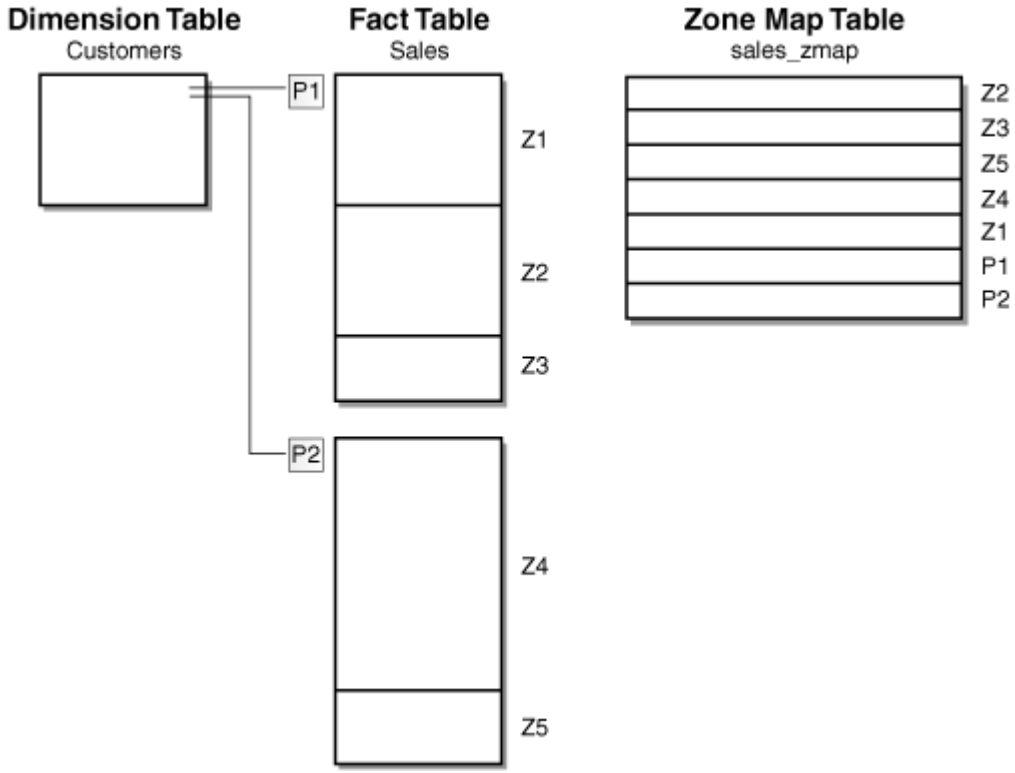
可能である点に注意してください。ゾーン・マップが部分的に失効しているかぎり、Z4に対応する表データは常に読み取られます(Z4でプルーニングは実行されません)。

図14-1 部分的に失効しているゾーン・マップ



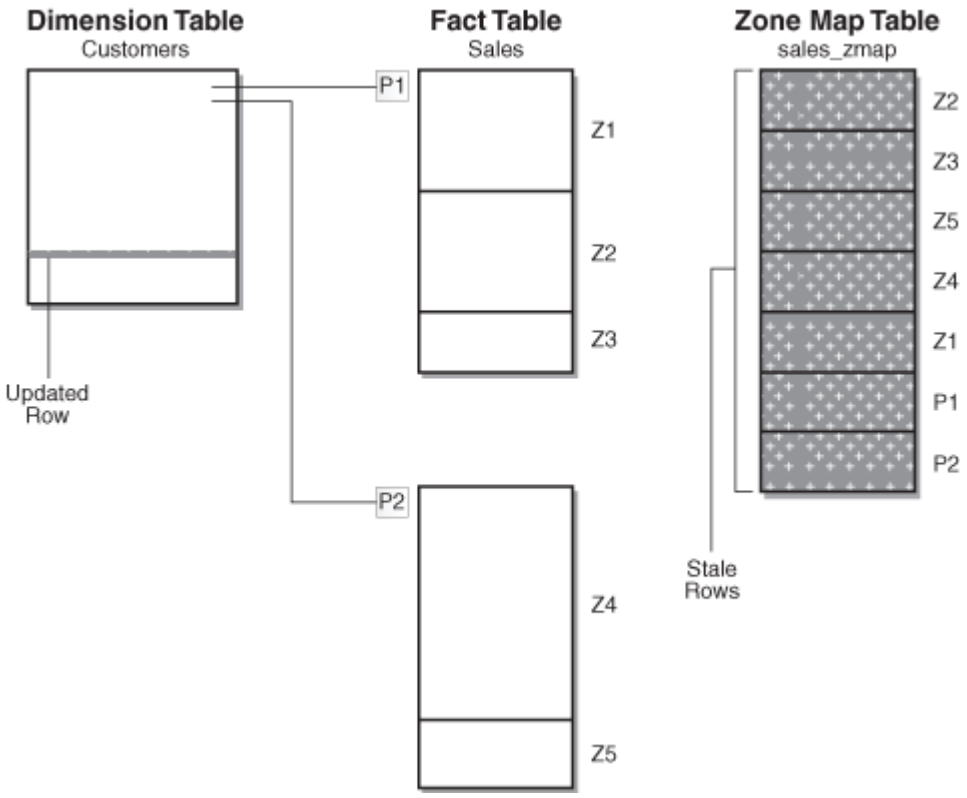
ディメンション表がファクト表に追加されると、ステータスは図14-2に似たものになります。

図14-2 ディメンション表を伴うゾーン・マップ



ディメンション表に対してなんらかのDMLが実行された場合、図14-3に示されているように、ゾーン・マップは完全に失効します。ゾーン・マップが完全に失効するため、完全にリフレッシュされるまで、プルーニングで使用できません。ゾーン・マップをリフレッシュするには、ALTER MATERIALIZED ZONEMAP文のREBUILDオプションを使用します。

図14-3 ディメンション表を伴うゾーン・マップと失効



14.3.2 ゾーン・マップのリフレッシュについて

Oracle Databaseでは、基礎になる表が変更された後、リフレッシュしてゾーン・マップをメンテナンスする必要があります。ゾーン・マップで使用されるリフレッシュ方法は、完全リフレッシュまたは増分リフレッシュです。REFRESH COMPLETE句を使用して指定される完全リフレッシュには、ゾーン・マップ内のすべてのゾーンの再構築が必要です。大量のデータを処理する必要があるため、完全リフレッシュには時間がかかります。REFRESH FAST句を使用して指定される増分リフレッシュは、最後のリフレッシュ以後に発生した変更のみを処理します。この方法では、ゼロから再構築せずにゾーン・マップをリフレッシュできます。ゾーン・マップはマテリアライズド・ビューを使用して内部的に実装されますが、ゾーン・マップの高速リフレッシュの実行には、実表上のマテリアライズド・ビュー・ログは必要ありません。

リフレッシュ・モードは、ゾーン・マップのリフレッシュをトリガーする操作を指定します。次のいずれかのリフレッシュ・モードを使用します。

- ON COMMIT

  実表に対する変更のコミット時にゾーン・マップがリフレッシュされます。

- ON DEMAND

  DML操作またはパーティション・メンテナンス操作の後、手動でゾーン・マップをリフレッシュする必要があります。

- ON DATA MOVEMENT

  実表上でデータ移動操作が実行されたときにゾーン・マップがリフレッシュされます。

- ON LOAD

  実表上で直接パス挿入操作が実行されたときにゾーン・マップがリフレッシュされます。

- ON LOAD DATA MOVEMENT

  実表上で直接パス挿入操作またはデータ移動操作が実行されたときにゾーン・マップがリフレッシュされます。これはデフォルトです。

デフォルトでは、ゾーン・マップはロード時およびデータ移動時にリフレッシュされます。このデフォルトをオーバーライドするには、ゾーン・マップの作成または変更時に、次のリフレッシュ・モードの1つを指定します: ON COMMIT、ON LOAD、ON DATA MOVEMENTまたはON LOAD。

14.3.3 ゾーン・マップのリフレッシュ

REFRESHオプションを指定しないでゾーン・マップを作成した場合、Oracle Databaseはデフォルトで、直接ロードの後、および特定のデータ移動操作の後に、ゾーン・マップのメンテナンスを実行します。例外は、DML操作(たとえば削除、挿入および更新)です。これらの操作の場合、Oracle Databaseは、ゾーン・マップ、またはゾーン・マップ内の一部のゾーンを、適切に失効とマークします。ゾーン・マップのリフレッシュ・メンテナンスを手動で制御するには、REFRESH ON DEMANDオプションを指定する必要があります。

次のコマンドは、リフレッシュ・メンテナンスが無効化されたゾーン・マップを作成します。この場合、基礎になる表が変更された後、ゾーン・マップを手動でリフレッシュする必要があります。

```
CREATE MATERIALIZED ZONEMAP sales_zmap
      ON sales (time_id, cust_id)
      REFRESH ON DEMAND;
```

Oracle Databaseには、ゾーン・マップをリフレッシュする次の2つの方法があります。

- ALTER MATERIALIZED ZONEMAPコマンドを使用したゾーン・マップのリフレッシュ
- DBMS_MVIEWパッケージを使用したゾーン・マップのリフレッシュ

14.3.3.1 ALTER MATERIALIZED ZONEMAPコマンドを使用したゾーン・マップのリフレッシュ

ゾーン・マップをリフレッシュするには、ALTER MATERIALIZED ZONEMAPコマンドのREBUILDオプションを使用します。

次のコマンドは、ゾーン・マップの完全リフレッシュを実行します。

```
ALTER MATERIALIZED ZONEMAP sales_zmap REBUILD COMPLETE;
```

次のコマンドは、ゾーン・マップが完全に失効しているか、使用不能とマークされている場合に、完全リフレッシュを実行します。それ以外の場合は、増分(高速)リフレッシュが実行されます。

```
ALTER MATERIALIZED ZONEMAP sales_zmap REBUILD;
```

関連項目:

ゾーン・マップ・リフレッシュの構文については、『*Oracle Database SQL言語リファレンス*』を参照してください。

14.3.3.2 DBMS_MVIEWパッケージを使用したゾーン・マップのリフレッシュ

DBMS_MVIEWパッケージのREFRESHプロシージャを使用してゾーン・マップをリフレッシュできます。

DBMS_MVIEW.REFRESHプロシージャを使用する場合、Oracle Databaseは、次に示すようなゾーン・マップのrefresh_methodパラメータに指定されている値に応じてリフレッシュを実行します。

- C: 完全リフレッシュを実行します。
- F - 高速リフレッシュを実行します。高速リフレッシュができない場合は、エラーが発行されます。
- ? - 可能な場合は高速リフレッシュを実行します。それ以外の場合は完全リフレッシュを実行します。

  値が指定されない場合、これがデフォルトとして使用されます。

REFRESHプロシージャの使用例を次に示します。

```
EXECUTE DBMS_MVIEW.REFRESH('sales_zmap','C');
```

14.4 ゾーン・マップを使用したプルーニングの実行

ゾーン・マップの主要な利点は、表スキャンのI/O削減です。プルーニングは、レコードの自然な場所に関する情報を活用して不必要なI/Oを回避します。SQL文に、ゾーン・マップで追跡される列にある述語が含まれる場合、データベースは各ゾーンについて述語値を最小値および最大値と比較して、表スキャン時にブロックのどのゾーンを読み取る、またはスキップするべきかを決定します。

ゾーン・マップ・プルーニングの候補には、次の述語が含まれます。

- 関係述語=、<=、<、>、>=
- (column_name relational_predicate constant形式のもの。たとえば、WHERE country_name='US'やWHERE country_name=:name)
- INリスト(たとえば、WHERE product_name IN ('a','b'))
- 接尾辞%が付けられたLIKE述語(たとえば、company_name LIKE 'ORA%')

この項では、次の項目について説明します。

- ゾーン・マップを使用したプルーニングの実行方法
- 例: ゾーン・マップと属性クラスタリングによるプルーニングの実行

14.4.1 ゾーン・マップを使用したプルーニングの実行方法

この項では、次の例を使用して、ゾーン・マップと属性クラスタリングでプルーニングがどのように実行されるかを説明します。

- ゾーン・マップを使用した表のプルーニング
- ゾーン・マップと属性クラスタリングを使用したパーティション表のプルーニング

14.4.1.1 ゾーン・マップを使用した表のプルーニング

この例は、述語が定数を含む問合せにおいてデータをプルーニングできるゾーン・マップの作成を示しています。表14-1に示すlineitem表は、次の文を使用して作成されます。

```
CREATE TABLE lineitem
  ( orderkey      NUMBER
  , shipdate      DATE
  , receiptdate   DATE
  , destination   VARCHAR2(50)
  , quantity      NUMBER);
```

この表には、1ブロックにつき2行のデータ・ブロックが4つ含まれると想定します。表14-1は、表の8つの行を示しています。

表14-1 lineitem表のデータ・ブロック

ブロック	orderkey	shipdate	receiptdate	destination	quantity
1	1	1-1-2011	1-10-2011	San_Fran	100
1	2	1-2-2011	1-10-2011	San_Fran	200
2	3	1-3-2011	1-5-2011	San_Fran	100

ブロック	orderkey	shipdate	receiptdate	destination	quantity
2	4	1-5-2011	1-10-2011	San_Diego	100
3	5	1-10-2011	1-15-2011	San_Fran	100
3	6	1-12-2011	1-16-2011	San_Fran	200
4	7	1-13-2011	1-20-2011	San_Fran	100
4	8	1-15-2011	1-30-2011	San_Jose	100

次に、CREATE MATERIALED ZONEMAP文を使用して、lineitem表上にゾーン・マップを作成します。

```
CREATE MATERIALIZED ZONEMAP lineitem_zmap
     ON lineitem (orderkey, shipdate, receiptdate);
```

各ゾーンには2つのブロックが含まれ、orderkey、shipdateおよびreceiptdate列の最小値と最大値が格納されます。表14-2は、ゾーン・マップを示しています。

表14-2 lineitem表のゾーン・マップ

ブロックの範囲	min orderkey	max orderkey	min shipdate	max shipdate	min receiptdate	max receiptdate
1-2	1	4	1-1-2011	1-5-2011	1-9-2011	1-10-2011
3-4	5	8	1-10-2011	1-15-2011	1-15-2011	1-30-2011

次の問合せを実行すると、データベースはゾーン・マップを読み、日付1-3-2011が日付の最小値と最大値の間にある、ブロック1および2のみをスキャンします。

```
SELECT * FROM lineitem WHERE shipdate = '1-3-2011';
```

14.4.1.2 ゾーン・マップと属性クラスタリングを使用したパーティション表のプルーニング

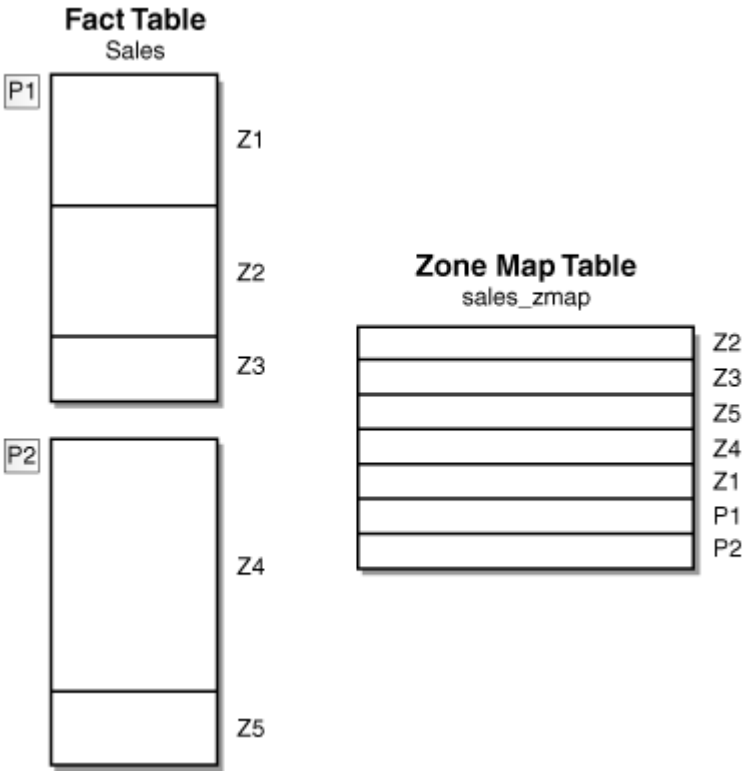
この次の文は、パーティション表上に、属性クラスタリングを伴うゾーン・マップを作成します。

```
CREATE TABLE sales
(
   prod_id        NUMBER NOT NULL,
   cust_id        NUMBER NOT NULL,
   time_id        DATE NOT NULL,
   channel_id     NUMBER NOT NULL,
   promo_id       NUMBER NOT NULL,
   quantity_sold  NUMBER(10,2) NOT NULL,
   amount_sold    NUMBER(10,2)
)
CLUSTERING sales JOIN products ON (sales.prod_id = products.prod_id)
   BY LINEAR ORDER (products.prod_id)
   WITH MATERIALIZED ZONEMAP (sales_zmap)
PARTITION BY HASH (amount_sold)
     ( PARTITION p1, PARTITION p2);
```

図14-4は、パーティション表salesのゾーン・マップの作成を示しています。5つのゾーンの各々について、ゾーン・マップには、ゾーン・マップで追跡される列の最小値と最大値が格納されます。述語が各ゾーンに格納された列の最小値と最大値の間にない場合、そのゾーンを読み取る必要はありません。一例として、ゾーンZ4が列prod_idの最小値10と最大値100を追跡している場合、このゾーンには述語prod_id = 200に一致するレコードは決して存在しないため、ゾーンZ4は読み取られません。

パーティション表では、パーティション上でもゾーン・レベル上でもプルーニングが発生することがあります。ゾーン・マップの集計パーティション・レベルの情報により、指定された述語の組合せに対して、一致するデータの可能性が否定される場合、パーティション全体がプルーニングされます。それ以外の場合は、ゾーン・レベルごとにパーティションのプルーニングが発生します。

図14-4 パーティション・ファクト表のゾーン・マップ



14.4.2 例: ゾーン・マップと属性クラスタリングによるプルーニングの実行

この項では、ゾーン・マップと属性クラスタリングを使用したプルーニングの実行の例を示します。例は、例14-16のように作成されたmy_sales表に基づきます。

例14-16 my_sales表の作成

my_sales表は、ゾーン・マップを含む、結合属性クラスタリングされた表です。SHスキーマのsales表に基づき、次の文を使用して作成されています。

```
CREATE TABLE my_sales
PARTITION BY LIST (channel_id)
 (PARTITION mysales_chan_c VALUES ('C'),
  PARTITION mysales_chan_i VALUES ('I'),
  PARTITION mysales_chan_p VALUES ('P'),
  PARTITION mysales_chan_s VALUES ('S'),
  PARTITION mysales_chan_t VALUES ('T'))
CLUSTERING
  my_sales JOIN customers ON (my_sales.cust_id = customers.cust_id)
  BY INTERLEAVED ORDER ((my_sales.time_id),
                        (customers.country_id,
                         customers.cust_state_province,
                         customers.cust_city))
  WITH MATERIALIZED ZONEMAP (mysales_zmap)
AS SELECT * FROM sales;
```

この項では、次の項目について説明します。

- 例: パーティションと表スキャン・プルーニング
- 例: ゾーン・マップ結合プルーニング

14.4.2.1 例:パーティションと表スキャン・プルーニング

この例は、ゾーン・マップがゾーンとパーティション(または、コンポジット・パーティション表内のサブパーティション)をどのようにプルーニングできるかを示しています。

1. my_sales表を作成します。例14-16には、この表を作成するために使用する構文が含まれています。
2. 次の文を使用して、customersディメンションに結合されているmy_sales表を問合せます。

```
SELECT c.cust_city, SUM(quantity_sold)
   FROM   my_sales s, customers c
   WHERE  s.cust_id = c.cust_id
      AND    c.country_id = 'US'
      AND    c.cust_state_province = 'CA'
      AND    s.promo_id < 50
   GROUP BY c.cust_city;
```

3. 次の文を使用して計画を表示します。

```
SELECT *
FROM TABLE(dbms_xplan.display_cursor(FORMAT => 'BASIC PREDICATE PARTITION'));

-------------------------------------------------------------------------------
| Id  | Operation                          | Name      | Pstart| Pstop |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |           |       |       |
|   1 |  HASH GROUP BY                     |           |       |       |
|*  2 |   HASH JOIN                        |           |       |       |
|   3 |    JOIN FILTER CREATE              | :BF0000   |       |       |
|*  4 |     TABLE ACCESS FULL              | CUSTOMERS |       |       |
|   5 |    JOIN FILTER USE                 | :BF0000   |       |       |
|   6 |     PARTITION LIST ITERATOR        |           |KEY(ZM)|KEY(ZM)|
|*  7 |      TABLE ACCESS FULL WITH ZONEMAP | MY_SALES  |KEY(ZM)|KEY(ZM)|
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
   2 - access("S"."CUST_ID"="C"."CUST_ID")
   4 - filter(("C"."CUST_STATE_PROVINCE"='CA' AND
      "C"."COUNTRY_ID"='US'))
   7 - filter((SYS_ZMAP_FILTER('/* ZM_PRUNING */ SELECT "ZONE_ID$",
      CASE WHEN BITAND(zm."ZONE_STATE$",1)=1 THEN 1 ELSE CASE WHEN
      (zm."MIN_2_COUNTRY_ID" > :1 OR zm."MAX_2_COUNTRY_ID" < :2 OR
      zm."MIN_3_CUST_STATE_PROVINCE" > :3 OR zm."MAX_3_CUST_STATE_PROVINCE" <
      :4) THEN 3 ELSE 2 END END FROM "SH"."MYSALES_ZMAP" zm WHERE
      zm."ZONE_LEVEL$"=0 ORDER BY zm."ZONE_ID$"',SYS_OP_ZONE_ID(ROWID),'US','U
      S','CA','CA')<3 AND "S"."PROMO_ID"<50 AND
      SYS_OP_BLOOM_FILTER(:BF0000,"S"."CUST_ID")))
```

7行目は、ゾーン・マップが使用されることを示します。ゾーン・マップ・パーティション・リスト・イテレータ"KEY(ZM)"に注意してください。

14.4.2.2 例: ゾーン・マップ結合プルーニング

この例は、ゾーン・マップと属性クラスタリングを使用する結合プルーニングを示しています。ディメンションの主キーがディメンション階層の値から構成されている場合、対応する外部キーでファクト表をクラスタリングすれば十分です。この例では、times.time_idが(calendar_year, calendar_month_number, day_number_in_month)のように構成されています。このため、time_idは、カレンダ時間階層および財務時間階層に変換されます。財務階層とカレンダ階層のいずれかについて述語が存在する場合、timesとmy_salesの間の結合をプルーニングすることができます。

1. my_sales表を作成します。例14-16には、この表を作成するために使用する構文が含まれています。
2. 次の文を使用して、timesに結合されたmy_sales表を問い合せます。

```
SELECT SUM(quantity_sold)
FROM   my_sales s, times tWHERE  s.time_id = t.time_id AND t.calendar_year = '1999';
```

3. 次の文を使用して計画を表示します。

```
SELECT *
FROM TABLE(dbms_xplan.display_cursor(FORMAT => 'BASIC PREDICATE PARTITION'));

-------------------------------------------------------------------------------
| Id  | Operation                                | Name    | Pstart| Pstop |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |         |       |       |
|   1 |  SORT AGGREGATE                          |         |       |       |
|*  2 |   HASH JOIN                              |         |       |       |
|   3 |    JOIN FILTER CREATE                    | :BF0000 |       |       |
|*  4 |     TABLE ACCESS FULL                    | TIMES   |       |       |
|   5 |    JOIN FILTER USE                       | :BF0000 |       |       |
|   6 |     PARTITION LIST ALL                   |         |     1 |     5 |
|*  7 |      TABLE ACCESS FULL WITH ZONEMAP      | MY_SALES|     1 |     5 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

PLAN_TABLE_OUTPUT
-----------------------------------------------------
   2 - access("S"."TIME_ID"="T"."TIME_ID")
   4 - filter("T"."CALENDAR_YEAR"=1999)
   7 - filter((SYS_ZMAP_FILTER('/* ZM_PRUNING */ SELECT "ZONE_ID$",
      CASE WHEN BITAND(zm."ZONE_STATE$",1)=1 THEN 1 ELSE CASE WHEN
      ((ORA_RAWCOMPARE(zm."MIN_1_TIME_ID",:1,8)>0 OR
      ORA_RAWCOMPARE(zm."MAX_1_TIME_ID",:2,8)<0)) THEN 3 ELSE 2 END END FROM
      "SH"."MYSALES_ZMAP" zm WHERE zm."ZONE_LEVEL$"=0 ORDER BY
      zm."ZONE_ID$"',SYS_OP_ZONE_ID(ROWID),SYSVARCOL,SYSVARCOL)<3 AND
      SYS_OP_BLOOM_FILTER(:BF0000,"S"."TIME_ID")))
```

7行目は、一致するtime_idゾーンを結合して、ゾーン・マップを使用していることを示します。

14.5 ゾーン・マップ情報の表示

ゾーン・マップとメジャーについての情報は、データ・ディクショナリ・ビューに格納されます。

この項では、次の項目について説明します。

- データベース内ゾーン・マップの詳細表示
- ゾーン・マップのメジャーの表示

14.5.1 データベース内ゾーン・マップの詳細表示

データベース内のゾーン・マップに関する情報を表示するには、次のデータ・ディクショナリ・ビューの1つを使用します。

- DBA_ZONEMAPS: データベース内のすべてのゾーン・マップを表示
- ALL_ZONEMAPS: ユーザーにアクセスできるゾーン・マップを表示
- USER_ZONEMAPS: ユーザーが所有するゾーン・マップを表示

次の問合せは、名前、実表、タイプ、リフレッシュ・モードおよび現在のユーザーが所有するゾーン・マップの失効を表示し、ゾーン・マップが属性クラスタリングとともに作成されているかどうかを示します。

```
SELECT zonemap_name,fact_table,hierarchical,with_clustering,refresh_mode,stale
       FROM USER_ZONEMAPS;

ZONEMAP_NAME    FACT_TABLE  HIERARCHICAL  WITH_CLUSTERING REFRESH_MODE      STALE
------------    ----------  ------------  --------------- ------------      -----
ZMAP$_MY_SALES  MY_SALES      NO          YES             LOAD DATAMOVEMENT   NO
```

次の問合せは、ユーザーにアクセスできるすべてのゾーン・マップのステータスを表示します。PRUNINGがDISABLEDのゾーン・マップは、I/Oプルーニングで使用されません。無効とマークされているゾーン・マップは、基礎になる実表の構造が変更しているため、再コンパイルする必要があります。

```
SQL> SELECT zonemap_name,pruning,refresh_method,invalid,complie_state
    FROM all_zonemaps;

ZONEMAP_NAME     PRUNING    REFRESH_METHOD INVALID  UNUSABLE   COMPILE_STATE
------------    ---------   -------------- -------  ------     -------------
SALES_ZMAP       ENABLED    FORCE           NO      NO         VALID
ZMAP$_MY_SALES   DISABLED   FORCE           NO      NO         VALID
```

14.5.2 ゾーン・マップのメジャーの表示

ゾーン・マップ内のメジャーに関する情報を表示するには、次のビューの1つを使用します。

- DBA_ZONEMAP_MEASURES: データベース内のすべてのゾーン・マップのメジャーを表示
- ALL_ZONEMAP_MEASURES: ユーザーにアクセスできるゾーン・マップのメジャーを表示
- USER_ZONEMAP_MEASURES: ユーザーが所有するゾーン・マップを表示

次の問合せは、ゾーン・マップ、メジャー、および現在のユーザーにアクセスできる各ゾーンについて最小値/最大値がメンテナンスされている列を表示します。

```
SELECT zonemap_name, measure, agg_function
FROM ALL_ZONEMAP_MEASURES;

ZONEMAP_NAME      MEASURE                          AGG_FUNCTION
---------------   ---------------------------      -------------
ZMAP$_MY_SALES   "SH"."MY_SALES"."PROD_ID"         MAX
ZMAP$_MY_SALES   "SH"."MY_SALES"."PROD_ID"         MIN
ZMAP$_MY_SALES   "SH"."MY_SALES"."CUST_ID"         MAX
ZMAP$_MY_SALES   "SH"."MY_SALES"."CUST_ID"         MIN
```

第III部 データ移動/ETL

この部では、データ・ウェアハウスの管理に必要な作業について説明します。

この部は、次の章で構成されています。

- データ移動/ETLの概要
- データ・ウェアハウスにおける抽出
- データ・ウェアハウスにおける転送
- データ・ウェアハウスにおけるロードおよび変換

15 データ移動/ETLの概要

この章では、データ・ウェアハウス環境でのデータの抽出、転送、変換およびロード・プロセスについて説明します。次のトピックが含まれています:

- データ・ウェアハウスにおけるETLの概要
- データ・ウェアハウスのETLツール

15.1 データ・ウェアハウスにおけるETLの概要

データ・ウェアハウスの目的はビジネス分析ですが、そのためには、定期的にデータをウェアハウスにロードする必要があります。このロードを行うには、1つ以上の業務系システムからデータを抽出し、データ・ウェアハウスにコピーする必要があります。データ・ウェアハウス環境では、多数のシステムにわたる大量のデータを統合、再配置および連結し、結果として、統一された新たな情報ベースをビジネス・インテリジェンスに提供することが課題になります。

ソース・システムからデータを抽出し、データ・ウェアハウスに取り込むプロセスは、一般的にETLと呼ばれます。これは、抽出(Extraction)、変換(Transformation)およびロード(Loading)の略です。ETLは1つの広範囲なプロセスであり、明確に定義された3つのステップの組合せではありません。ETLという略語は簡略すぎるように見えます。これでは、転送フェーズが示されておらず、他の3つのフェーズがそれぞれ独立しているかのような印象を受けます。いずれにせよ、このプロセス全体はETLとして知られています。

ETLの方法論およびタスクは以前から周知されており、必ずしもデータ・ウェアハウス環境に固有ではありません。あらゆる企業において、様々な独自のアプリケーションやデータベース・システムが、ITのバックボーンとなっています。データをアプリケーション間やシステム間で共有し、統合し、少なくとも2つのアプリケーションに同じ情報が表示されるようにする必要があります。データの共有のほとんどは、ETLのようなメカニズムで対処されています。

15.1.1 データ・ウェアハウスにおけるETLの基本

ETLプロセス中に発生する処理を考えてみます。このプロセスで主に発生する処理には、次のタスクがあります。

- データ・ウェアハウスにおけるデータの抽出
- データ・ウェアハウスにおけるデータの転送

15.1.1.1 データ・ウェアハウスのデータの抽出

抽出中には、必要なデータが識別され、データベース・システムやアプリケーションなど、多数の異なるソースから抽出されます。通常、必要なデータの特定のサブセットは識別できないため、必要以上のデータを抽出する必要があり、関連データの識別は後の時点で行われます。ソース・システムの機能(オペレーティング・システムのリソースなど)によっては、この抽出プロセス中になんらかの変換が行われる場合もあります。抽出されるデータのサイズは、ソース・システムとビジネス状況に応じて数百KBから数GBになります。これは、2つの(論理的に)同一の抽出処理間の時間デルタにもあてはまります。タイム・スパンは、日数/時間数、分数やほぼリアルタイムまで様々です。たとえば、Webサーバーのログ・ファイルは、きわめて短期間のうちに数百MBにまで大きくなることがあります。

15.1.1.2 データ・ウェアハウスのデータの転送

データの抽出後は、それをターゲット・システムまたは中間システムに物理的に転送し、さらに処理する必要があります。選択した転送方法によっては、このプロセス中にもなんらかの変換を行うことができます。たとえば、ゲートウェイを通じてリモート・ターゲットに直接アクセスするSQL文では、SELECT文の中で2つの列を連結できます。

この項では、スケーラビリティに重点を置いた例を示します。Oracle Databaseを長く使用していれば、PL/SQLによる複雑な

データ変換ロジックのプログラミングについてはエキスパートであると言えます。ここでは、このような多数のデータ操作の代替手段を提案し、Oracleの新しいSQL機能、特にETLとパラレル問合せインフラストラクチャを活用する実装に重点を置いて説明します。

15.2 データ・ウェアハウスのETLツール

ETLプロセスの設計およびメンテナンスは、データ・ウェアハウス・プロジェクトで最も困難でリソースを大量に必要とする部分だと考えられています。多数のデータ・ウェアハウス・プロジェクトでは、このプロセスを管理するためにETLツールが使用されます。たとえば、Oracle Data Integrator (ODI)にはETL機能が搭載されており、本来のデータベース機能を活用できます。他のデータ・ウェアハウス・ビルダーもありますが、これらは独自のETLツールおよびプロセスをデータベース内またはデータベース外に作成します。

ETLをデータ・ウェアハウスの日常操作および今後の強化のサポートの一部として順調に実装するには、抽出、変換およびロードのサポートの他にも重要なタスクがあります。通常、これらのタスクは、データ・ウェアハウス設計およびデータ・フロー設計のサポートとともに、ODIなどのETLツールで取り扱われます。

OracleはETLツールではなく、ETLの完全なソリューションを提供するものでもありません。ただし、Oracleには、ETLツールとカスタマイズされたETLソリューションの両方で使用できる豊富な機能が搭載されています。Oracleは、Oracleデータベース間でデータを転送し、大量のデータを変換し、新しいデータをデータ・ウェアハウスに迅速にロードするための技術を提供します。

15.2.1 データ・ウェアハウスにおける日次操作

継続的なロードおよび変換を、特定の順序でスケジュールして処理する必要があります。操作または操作の一部が成功したか失敗したかに応じて、結果を追跡する必要があります。また、場合によっては後続の代替プロセスを開始できます。操作の進行状況の制御とビジネス・ワークフローの定義は、通常はOracle Data Integrator (ODI)などのETLツールで扱われます。

15.2.2 データ・ウェアハウスの発展

データ・ウェアハウスは生きたITシステムであり、ソースとターゲットが変化することがあります。このような変化は、古いETLプロセス・フロー情報を上書きまたは削除せずに、システムの稼働期間中を通じてメンテナンスし、追跡する必要があります。ウェアハウス内の情報の信頼性のレベルを築き、維持するために、ウェアハウス内の個々のレコードの処理フローは、将来の適切な時期に、いつでも再構築が可能です。

16 データ・ウェアハウスにおける抽出

この章では、抽出について説明します。抽出とは、業務系システムからデータを取り出してデータ・ウェアハウスまたはステージング・システムに移動するプロセスです。この章の内容は次のとおりです。

- データ・ウェアハウスにおける抽出の概要
- データ・ウェアハウスにおける抽出方法の概要
- データ・ウェアハウスにおける抽出の例

16.1 データ・ウェアハウスにおける抽出の概要

抽出とは、データ・ウェアハウス環境で使用できるように、ソース・システムからデータを抽出する操作です。これは、ETLプロセスの最初のステップです。抽出後のデータは、変換してデータ・ウェアハウスにロードできます。

通常、データ・ウェアハウスのソース・システムになるのは、トランザクション処理データベース・アプリケーションです。たとえば、販売分析データ・ウェアハウスのソース・システムの例としては、現在の受注状況をすべて記録する注文入力システムがあります。

抽出プロセスの設計および作成は、通常はETLプロセスおよびデータ・ウェアハウスのプロセス全体の中で、最も時間がかかる作業の1つです。ソース・システムが非常に複雑で、詳細にドキュメント化されていないため、どのデータを抽出する必要があるかを決定できない場合もあります。通常、データは1度だけ抽出されるのではなく、変更があったすべてのデータをデータ・ウェアハウスに提供して最新の状態に保つために、定期的に何回も抽出されます。さらに、データ・ウェアハウスの抽出プロセスの必要性に応えようとしても、通常ソース・システムは変更できず、そのパフォーマンスまたは可用性を調整することもできません。

これらは、抽出およびETL全体で考慮する必要がある重要な考慮点です。ただし、この章では、各種のソースおよび抽出方法に関する技術的な考慮点について重点的に説明します。ここでは、データ・ウェアハウス・チームが抽出対象のデータをすでに識別していることを前提に、ソース・データベースからデータを抽出する一般的なテクニックについて説明します。

このプロセスの設計では、主に次の2つのカテゴリに関する意思決定を行うことになります。

- どの抽出方法を選択するか

  これは、ソース・システム、転送プロセスおよびウェアハウスのリフレッシュの所要時間に影響します。

- 抽出されたデータを後続の処理にどんな方法で提供するか

  これは、転送方法と、データのクレンジングおよび変換のニーズに影響します。

16.2 データ・ウェアハウスにおける抽出方法の概要

どの抽出方法を選択する必要があるかは、ソース・システムに大きく左右され、ターゲットとなるデータ・ウェアハウス環境でのビジネス・ニーズも考慮する必要があります。処理負荷の増大やパフォーマンスへの影響を考えると、データの増分抽出機能を強化するために新たなロジックをソース・システムに追加するというのはあまり現実的な方法ではありません。顧客には既製のアプリケーション・システムに対する追加が許されていない場合さえもあります。

この項では、次の項目について説明します。

- 論理的抽出方法
- 物理的抽出方法
- チェンジ・トラッキング方法

16.2.1 論理的抽出方法

論理的抽出には、次の2種類があります。

- 全体抽出
- 増分抽出

全体抽出

データ全体がソース・システムから抽出されます。この抽出では、ソース・システムで現在使用可能なデータがすべて反映されるため、前回の抽出成功以降にデータソースに対して行われた変更を追跡する必要がありません。ソース・データはそのまま提供され、ソース側では追加の論理情報(タイムスタンプなど)は不要です。全体抽出の例には、特定の表のエクスポート・ファイルや、ソース表全体をスキャンするリモートSQL文があります。

増分抽出

特定の時点で、履歴で適切に定義されたイベント以降に変更があったデータのみが抽出されます。このイベントは、最終抽出時刻でも、会計期間の最終記帳日のように複雑なビジネス・イベントでもかまいません。このデルタ変更を識別するには、この特定の時間事象以降に変更があった情報をすべて識別する機能が必要です。この情報は、最終変更日時のタイムスタンプを反映するアプリケーション列のようなソース・データ自体、または適切な追加メカニズムにより起点となったトランザクションに加え、変更が追跡されるチェンジ・テーブルから得ることができます。ほとんどの場合は、後者の方法を使用すると、ソース・システムに抽出のロジックを追加することになります。

抽出プロセスに対して、なんらかの形で変更をキャプチャするテクニックを使用しているデータ・ウェアハウスはあまりありません。そのかわり、ソース・システムからすべての表をデータ・ウェアハウスまたはステージング・エリアに抽出し、ソース・システムからの前回の抽出と比較して変更データを識別します。この方法では、ソース・システムが重大な影響を受けることはありませんが、データ・ボリュームが非常に大きい場合は、特に、データ・ウェアハウスの処理に対しては、多大な負荷になります。

16.2.2 物理的抽出方法

選択した論理的抽出方法と、ソース側の機能と制限に応じて、抽出データを2つのメカニズムで物理的に抽出できます。つまり、データをソース・システムからオンラインで抽出する方法と、オフライン構成から抽出する方法があります。オフライン構成は既存のものを使用する場合と、抽出ルーチンで生成する場合があります。

物理的抽出には、次の2つの方法があります。

- オンライン抽出
- オフライン抽出

オンライン抽出

データはソース・システム自体から直接抽出されます。抽出プロセスでは、ソース・システムに直接接続してソース表自体にアクセスするか、データが事前構成済の方法(スナップショット・ログやチェンジ・テーブルなど)で格納されている中間システムに接続できます。なお、中間システムは必ずしもソース・システムと物理的に異なるとは限りません。

オンライン抽出では、分散トランザクションでオリジナルのソース・オブジェクトを使用しているか、準備したソース・オブジェクトを使用しているかを考慮する必要があります。

オフライン抽出

データはソース・システムから直接抽出されるのではなく、オリジナル・ソース・システム外部で明示的にステージングされます。データはすでに既存の仕組み(REDOログ、アーカイブ・ログまたはトランスポータブル表領域など)を持っているか、または抽出ルーチンにより作成されています。

次の仕組みを考慮する必要があります。

- フラット・ファイル

  定義済の汎用フォーマットのデータ。さらに処理するには、ソース・オブジェクトに関する追加情報が必要です。

- ダンプ・ファイル

  Oracle固有のフォーマット。選択したユーティリティによって、含んでいるオブジェクトに関する情報がある場合とない場合があります。

- REDOログおよびアーカイブ・ログ

  情報は特殊な追加ダンプ・ファイルにあります。

- トランスポータブル表領域

  Oracleデータベース間で大量のデータを抽出および移動する強力な手段。この機能を使用して、データの抽出および転送を行う例については、データ・ウェアハウスにおける転送を参照してください。他の抽出テクニックに比べてパフォーマンスと管理性が大幅に向上するため、できるかぎりトランスポータブル表領域を使用することをお薦めします。

  エクスポート/インポート使用の詳細は、『Oracle Databaseユーティリティ』を参照してください。

16.2.3 チェンジ・トラッキング方法

抽出における重要な考慮点として、増分抽出があります。これはチェンジ・トラッキングとも呼ばれます。業務系システムからデータ・ウェアハウスへのデータ抽出を夜間に行う場合、データ・ウェアハウスに必要なのは、前回の抽出以降に変更されたデータ(過去24時間で変更されたデータ)のみです。また、チェンジ・トラッキングは、ほぼリアルタイムまたはオンタイムのデータ・ウェアハウスを提供するキー有効化テクノロジでもあります。

変更された最新データのみを効率的に識別して抽出できれば、データ抽出ボリュームがわずかで済み、抽出プロセス(およびETLプロセスの下流の処理)を大幅に効率化できます。ただし、多くのソース・システムでは最新の変更データの識別は困難であり、また、システムの操作に影響を及ぼすこともあります。チェンジ・トラッキングは、データ抽出における最も困難な技術課題です。

多くの場合、チェンジ・トラッキングは抽出プロセスの一部として望ましいものであるため、この項では独自開発のチェンジ・キャプチャ機能をOracle Databaseソース・システムに実装するいくつかのテクニックについて説明します。

- タイムスタンプ
- パーティション化
- トリガー

これらのテクニックはソース・システムの特性に基づいていますが、ソース・システムに変更が必要なこともあります。したがって、これらのテクニックを実装する前に、ソース・システムのユーザーが慎重にそれぞれを評価する必要があります。

これらのテクニックは、それぞれ前述のデータ抽出テクニックと連携して機能します。たとえば、データがファイルにアンロードされているか、分散問合せによってアクセスされている場合は、タイムスタンプを使用できます。

タイムスタンプ

業務系システムの中には、表にタイムスタンプ列を持つものもあります。タイムスタンプは、ある行が最後に変更された日付および時間を示します。業務系システムの表にタイムスタンプを含む列があれば、そのタイムスタンプ列を使用することで最新のデータを簡単に識別できます。たとえば、orders表から今日のデータを抽出するには、次の問合せが有効です。

```
SELECT * FROM orders
WHERE TRUNC(CAST(order_date AS date),'dd') =
  TO_DATE(SYSDATE,'dd-mon-yyyy');
```

業務系ソース・システムにタイムスタンプ情報がない場合、タイムスタンプを含むようにシステムを変更できるとは限りません。このような変更が必要な場合は、まず、業務系システムの表を変更して、新しくタイムスタンプ列を追加します。次に、特定の行を変更する操作が行われるたびにタイムスタンプ列を更新するトリガーを作成します。

パーティション化

ソース・システムの中には、レンジ・パーティション化を使用しているものもあります。これには、ソース表が日付キーによってパーティション化され、新しいデータが簡単に識別できるようになっているものなどがあります。たとえば、orders表から抽出する場合は、orders表が週別にパーティション化されていれば、現在の週のデータを簡単に識別できます。

トリガー

業務系システムにトリガーを作成することで、最新の更新レコードを追跡できます。これらのトリガーをタイムスタンプ列と連携させて使用し、ある行が最後に変更された正確な日付および時間を識別することもできます。そのためには、データ変更を獲得する必要がある各ソース表にトリガーを作成します。ソース表でDML文が実行されるたびに、このトリガーによってタイムスタンプ列が現在の時間に更新されます。このようにして、行が最後に変更された正確な日付および時間が、タイムスタンプ列に反映されます。

同様に内部化されたトリガー・ベースのテクニックが、Oracleのマテリアライズド・ビュー・ログに使用されています。これらのログは、マテリアライズド・ビューが変更データを識別するために使用し、エンド・ユーザーもアクセスできます。ただし、マテリアライズド・ビュー・ログの形式はドキュメント化されないので、時間が経過すると変化する場合があります。

マテリアライズド・ビュー・ログはトリガーに依存しますが、このデータ変更システムの作成およびメンテナンスが主にデータベースで管理されるという点が効果的です。

トリガー・ベース・テクニックはソース・システムのパフォーマンスに影響する場合があるため、本番ソース・システムに実装する前にその点を慎重に検討する必要があります。

16.3 データ・ウェアハウスにおける抽出の例

次の2つのデータ抽出方法があります。

- データファイルを使用した抽出
- 分散処理による抽出

16.3.1 データファイルを使用した抽出

ほとんどのデータベース・システムには、内部データベース・フォーマットからデータをフラット・ファイルにエクスポートまたはアンロードする機能が搭載されています。メインフレーム・システムからの抽出では、COBOLプログラムが使用されることがありますが、データベースの多くは、エクスポートまたはアンロード用のユーティリティを搭載しています。

データの抽出では、必ずしもデータベースの全構造がフラット・ファイルにアンロードされるわけではありません。多くの場合、データベース表全体またはオブジェクト全体をアンロードすることが適切です。前回の抽出以降にソース・システムで行われた変更のような、特定の表のサブセットのみ、または複数の表を結合した結果のみをアンロードする方がより適切な場合もあります。抽出テクニックによって、これらの2つのいずれが適切かが異なります。

ソース・システムがOracleデータベースの場合、データをファイルに抽出するには、次のように何通りかの方法があります。

- SQL*Plusによるフラット・ファイルへの抽出
- OCIまたはPro*Cプログラムによるフラット・ファイルへの抽出
- エクスポート・ユーティリティによるエクスポート・ファイルへの抽出
- 外部表によるエクスポート・ファイルへの抽出

16.3.1.1 SQL*Plusによるフラット・ファイルへの抽出

データ抽出の最も基本的なテクニックは、SQL*PlusでSQL問合せを実行して、問合せの出力をファイルに送ることです。たとえば、フラット・ファイルcountry_city.logを抽出するには、次のSQLスクリプトを実行します。このファイルでは、列値の間にデリミタとしてパイプ記号が使用されており、表countriesおよびcustomersからのアメリカの市のリストが含まれています。

```
SET echo off SET pagesize 0 SPOOL country_city.log
SELECT distinct t1.country_name ||'|'|| t2.cust_city
FROM countries t1, customers t2 WHERE t1.country_id = t2.country_id
AND t1.country_name= 'United States of America';
SPOOL off
```

出力ファイルの抽出フォーマットは、SQL*Plusのシステム変数を使用して指定できます。

この抽出テクニックは、カスタマイズされたフォーマットで結果を格納するのに効果的です。外部表データ・ポンプ・アンロード機能を使用すると、任意のSQL操作の結果を抽出することもできます。前述の例では、結合の結果を抽出します。

この抽出テクニックは、複数の同時SQL*Plusセッションを起動し、各セッションで、抽出対象データの異なる部分を表す別々の問合せを実行することで、パラレル化できます。たとえば、orders表からデータを抽出するとき、そのorders表が月別にレンジ・パーティション化されており、orders_jan1998、orders_feb1998などのパーティションがあるとします。orders表から1年分のデータを抽出するには、12の同時SQL*Plusセッションを起動し、各セッションでパーティションを1つずつ抽出します。このようなセッションを行うSQLスクリプトの例を、次に示します。

```
SPOOL order_jan.dat
SELECT * FROM orders PARTITION (orders_jan1998);
SPOOL OFF
```

これらの12のSQL*Plusプロセスによって、データが12の別々のファイルに同時にスプールされます。必要な場合は、抽出後に(オペレーティング・システムのユーティリティを使用して)これらのファイルを連結できます。ターゲットへのロードにSQL*Loaderを使用する場合は、この12のファイルをそのまま使用して12のSQL*Loaderセッションでパラレル・ロードできます。例については、データ・ウェアハウスにおける転送を参照してください。

orders表がパーティション化されていない場合でも、論理的または物理的な基準に基づいて抽出をパラレル化できます。論理的な方法は、次のように列値の論理的な範囲に基づいています。

```
SELECT ... WHERE order_date
BETWEEN TO_DATE('01-JAN-99') AND TO_DATE('31-JAN-99');
```

物理的な方法は、値の範囲に基づいています。データ・ディクショナリを参照することで、orders表を構成するOracle Databaseデータ・ブロックを識別できます。この情報を使用して、orders表からデータを抽出するためのROWIDレンジ問合せの集合を導出できます。

```
SELECT * FROM orders WHERE rowid BETWEEN value1 and value2;
```

複雑なSQL問合せによる抽出もパラレル化できますが、1つの複雑な問合せを複数のコンポーネントに分割することが困難な場合があります。特に、独立したプロセスを調整してグローバルな一貫性を持つビューを保証するのは困難な場合があります。SQL*Plusの方法と異なり、外部表データ・ポンプ・アンロード機能では透過的にパラレル化できます。

パラレル化のすべてのテクニックは、ソース・システムのCPUおよびI/Oリソースを大量に使用します。抽出テクニックをパラレル化する前に、ソース・システムへの影響を評価する必要があることに注意してください。

16.3.1.2 OCIまたはPro*Cプログラムによるフラット・ファイルへの抽出

OCIプログラム(または、Pro*CプログラムなどOracle Call Interfaceを使用する他のプログラム)も、データ抽出に使用できます。これらのテクニックによって、通常、SQL*Plusを使用する方法より高いパフォーマンスが得られますが、プログラミングには時

間がかかります。SQL*Plusの方法と同様に、OCIプログラムもSQL問合せの結果の抽出に使用できます。また、SQL*Plusの方法で説明したパラレル化のテクニックもOCIプログラムに対して簡単に適用できます。

OCIまたはSQL*Plusを抽出に使用する場合は、データ自体のみでなく追加情報が必要になります。少なくとも、抽出される列に関する情報が必要です。また、抽出フォーマットがわかっていれば役立ちます。これには、個別の列間のセパレータなどがあります。

16.3.1.3 エクスポート・ユーティリティによるエクスポート・ファイルへの抽出

エクスポート・ユーティリティによって、表(データ含む)をOracle Databaseエクスポート・ファイルにエクスポートできます。SQL問合せの結果を抽出するSQL*PlusおよびOCIの方法とは異なり、エクスポート・ユーティリティではデータベースのオブジェクトを抽出するメカニズムが提供されます。そのため、エクスポート・ユーティリティは前述の2つの方法とは大きく異なる点があります。

- エクスポート・ファイルには、データの他にメタデータも含まれます。エクスポート・ファイルには、表の生データのみでなく表を再作成するための情報も含まれ、索引、制約、権限付与、およびその表に関係する他の属性があればそれらも含まれます。
- 1つのエクスポート・ファイルにはシングル・オブジェクトのサブセットまたは多数のデータベース・オブジェクトが含まれ、スキーマ全体が含まれることもあります。
- エクスポート・ユーティリティは、複雑なSQL問合せの結果のエクスポートには直接使用できません。エクスポートは、個々のデータベース・オブジェクトのサブセットの抽出にのみ使用できます。
- エクスポート・ユーティリティの出力は、インポート・ユーティリティを使用して処理する必要があります。

Oracle Databaseでは、下位互換性用の元のエクスポート/インポート・ユーティリティ、および高パフォーマンスでスケーラブルなパラレル抽出のためのデータ・ポンプ・エクスポート/インポート・インフラストラクチャを提供しています。詳細は、『Oracle Databaseユーティリティ』を参照してください。

16.3.1.4 外部表によるエクスポート・ファイルへの抽出

エクスポート・ユーティリティの他に、外部表を使用して任意のSELECT操作の結果を抽出できます。データは、プラットフォームに依存しないOracle内部データ・ポンプ形式で格納され、ターゲット・システムで通常の外部表として処理できます。

次の例では、パラレルの結合操作の結果を、指定した4つのファイルに抽出します。データ抽出用に使用できる外部表のタイプは、Oracle内部形式ORACLE_DATAPUMPのみです。

```
CREATE DIRECTORY def_dir AS '/net/private/jdoe/WORK/FEATURES/et';
DROP TABLE extract_cust;
CREATE TABLE extract_cust
ORGANIZATION EXTERNAL
(TYPE ORACLE_DATAPUMP DEFAULT DIRECTORY def_dir ACCESS PARAMETERS
(NOBADFILE NOLOGFILE)
LOCATION ('extract_cust1.exp', 'extract_cust2.exp', 'extract_cust3.exp',
          'extract_cust4.exp'))
PARALLEL 4 REJECT LIMIT UNLIMITED AS
SELECT  c.*, co.country_name, co.country_subregion, co.country_region
FROM customers c, countries co where co.country_id=c.country_id;
```

指定した抽出ファイルの合計数により、書込み操作の最大並列度が制限されます。抽出をパラレル化しても、文のSELECT部分は自動的にパラレル化されないことに注意してください。

エクスポート/インポートを使用する場合とは異なり、外部表データ・ポンプ・アンロードを使用すると、外部表のメタデータは作成されたファイルには含まれません。外部表の適切なメタデータを抽出するには、次の文に示すとおりDBMS_METADATAパッケージを使用します。

```
SET LONG 2000
```

```
SELECT DBMS_METADATA.GET_DDL('TABLE','EXTRACT_CUST') FROM DUAL;
```

16.3.2 分散処理による抽出

分散問合せテクノロジを使用すると、1つのOracleデータベースから、他のOracleデータベースやOracleゲートウェイ・テクノロジで接続されている従来型システムなど、各種のソース・システムにある表へ直接問い合せることができます。具体的には、データ・ウェアハウスまたはステージング・データベースから、接続されているソース・システムにある表やデータに直接アクセスできます。ゲートウェイは、分散問合せテクノロジのもう1つの形式です。ゲートウェイでは、Oracleデータベース(データ・ウェアハウスなど)からリモートの非Oracleデータベースに格納されているデータベース表にアクセスできます。2つのOracleデータベース間でデータを移動するには、これが最も簡単な方法です。抽出と変換のプロセスを1ステップで行うことができるうえ、プログラミングに必要な時間も最小限に抑えられます。ただし、これが常に実現可能であるとは限りません。

部署名を含む従業員名リストをソース・データベースから抽出し、それをデータ・ウェアハウスに格納するとします。Oracle Net接続および分散問合せテクノロジを使用すると、この作業は、次の1つのSQL文で実現できます。

```
CREATE TABLE country_city AS SELECT distinct t1.country_name, t2.cust_city
FROM countries@source_db t1, customers@source_db t2
WHERE t1.country_id = t2.country_id
AND t1.country_name='United States of America';
```

この文により、データ・マートにローカル表country_cityが作成され、ソース・システム上の表countriesおよびcustomersからデータが移入されます。

このテクニックは少量のデータ移動には理想的です。ただし、データは単一のOracle Net接続を介してソース・システムからデータ・ウェアハウスに転送されます。そのため、このテクニックのスケーラビリティには限界があります。データが大量になると、ファイル・ベースのデータ抽出および転送テクニックの方が、よりスケーラブルで適切であることがあります。

関連項目:

- 分散問合せの詳細は、*Oracle Database Heterogeneous Connectivityのユーザーズ・ガイド*を参照してください。
- 分散問合せの詳細は、*『Oracle Database概要』*を参照してください。

17 データ・ウェアハウスにおける転送

この章では、データ・ウェアハウスへのデータの転送について説明します。

- データ・ウェアハウスにおける転送の概要
- データ・ウェアハウスにおける転送メカニズムの概要

17.1 データ・ウェアハウスにおける転送の概要

転送とは、データをあるシステムから別のシステムへ移動させる操作です。データ・ウェアハウス環境では、最も一般的な転送要件は、次のデータ移動です。

- ソース・システムからステージング・データベースまたはデータ・ウェアハウス・データベースへ
- ステージング・データベースからデータ・ウェアハウスへ
- データ・ウェアハウスからデータ・マートへ

転送は、ETLプロセスの中でも単純な部分であることが多く、一般的にプロセスの他の部分と統合できます。たとえば、データ・ウェアハウスにおける抽出に示すように、分散問合せテクノロジにはデータ抽出と転送の両方のメカニズムが備わっています。

17.2 データ・ウェアハウスにおける転送メカニズムの概要

ウェアハウスにおけるデータ転送には、次の3つの基本的な方法があります。

- フラット・ファイルを使用した転送
- 分散処理による転送
- トランスポータブル表領域を使用した転送

17.2.1 フラット・ファイルを使用した転送

データ転送の最も一般的な方法は、FTPや他のリモート・ファイル・システム・アクセス・プロトコルなどのメカニズムを使用して、フラット・ファイルを転送する方法です。データは、データ・ウェアハウスにおける抽出で説明したテクニックで、ソース・システムからフラット・ファイルにアンロードまたはエクスポートされ、FTPまたは同等のメカニズムによってターゲット・プラットフォームに転送されます。

ソース・システムとデータ・ウェアハウスでは、異なるオペレーティング・システムとデータベース・システムを使用していることが多いため、データ変換を最小限に抑えて異機種システム間でデータを交換するには、フラット・ファイルを使用するのが最も単純な方法です。ただし、同機種システム間でのデータ転送でも、フラット・ファイルは最も効率的で扱いやすいデータ転送メカニズムです。

17.2.2 分散処理による転送

分散問合せは、ゲートウェイを使用するかどうかを問わずデータ抽出の効果的なメカニズムです。これらのメカニズムでは、ターゲット・システムに直接データを転送するため、抽出と変換を1ステップで処理します。時間とシステム・リソースに許される影響度によっては、これらのメカニズムは抽出と変換の両方に適しています。

フラット・ファイルの転送とは異なり、転送の成否は分散問合せまたはトランザクションの結果とともに即時に認識されます。

関連項目:

- 詳細は、データ・ウェアハウスにおける抽出を参照してください。

17.2.3 トランスポータブル表領域を使用した転送

Oracleトランスポータブル表領域により、大量のデータを2つのOracleデータベース間で最も高速に移動させることができます。トランスポータブル表領域の導入以前は、最もスケーラブルなデータ転送メカニズムは、生データを含むフラット・ファイルの移動によるものでした。このようなメカニズムでは、データをまずソース・データベースからアンロードまたはエクスポートし、転送後に、ターゲット・データベースにロードまたはインポートする必要がありました。トランスポータブル表領域ではこのアンロードと再ロードのステップは不要です。

トランスポータブル表領域を使用することによって、Oracleデータファイル(表データ、索引およびその他のOracleデータベース・オブジェクトのほぼすべてを含む)を、あるデータベースから他のデータベースへ直接転送できます。さらに、トランスポータブル表領域は、インポートおよびエクスポートと同様に、データのみでなくメタデータを転送するメカニズムも提供します。

トランスポータブル表領域には、いくつかの制限があり、ソース・システムおよびターゲット・システムでは、Oracle8*i*(またはそれ以上)が稼働していること、互換性のある文字セットを使用すること、Oracle Database 10*g*より前では同一のオペレーティング・システムで稼働することが必要です。オペレーティング・システム間で表領域を転送する方法の詳細は、*『Oracle Database管理者ガイド』*を参照してください。

データ・ウェアハウスにおけるトランスポータブル表領域の最も一般的な用途は、ステージング・データベースからデータ・ウェアハウスへのデータ移動、またはデータ・ウェアハウスからデータ・マートへのデータ移動です。

この項では、次の項目について説明します。

- トランスポータブル表領域を使用したデータ・ウェアハウスへのデータの転送: 例
- トランスポータブル表領域の他の用途

17.2.3.1 トランスポータブル表領域を使用したデータ・ウェアハウスへのデータの転送: 例

売上データを含む1つのデータ・ウェアハウス、および毎月リフレッシュされる複数のデータ・マートがあるとします。また、1か月分の売上データをデータ・ウェアハウスからデータ・マートに移動するとします。

次のステップを使用して、トランスポータブル表領域を作成します。

1. 転送するデータを専用の表領域に配置する
2. メタデータのエクスポート
3. データファイルおよびエクスポート・ファイルをターゲット・システムにコピーする
4. メタデータのインポート

転送するデータを専用の表領域に配置する

現在の月のデータを転送するには、まず、別々に用意した表領域にそのデータを配置する必要があります。この例では、ts_temp_sales表領域に今月のデータをコピーするとします。CREATE TABLE ... AS SELECT文を使用すると、当月のデータを効率的にこの表領域にコピーできます。

```
CREATE TABLE temp_jan_sales NOLOGGING TABLESPACE ts_temp_sales
AS SELECT * FROM sales
WHERE time_id BETWEEN '31-DEC-1999' AND '01-FEB-2000';
```

この操作の完了後に、表領域ts_temp_salesを読取り専用に設定します。

```
ALTER TABLESPACE ts_temp_sales READ ONLY;
```

表領域は、その表領域を変更するアクティブなトランザクションがなくなるまで転送できません。表領域を読取り専用に設定することによって、転送が可能になります。

ts_temp_sales表領域は、トランスポータブル表領域機能が使用する一時的なデータ格納領域として、特別に作成したものの場合もあります。データファイルおよびエクスポート・ファイルをターゲット・システムにコピーするの後、この表領域を読取り/書込みに設定できます。必要に応じて、temp_jan_sales表を削除したり、その表領域を他のデータ転送やそれ以外の目的で再使用できます。

トランスポータブル表領域操作では、任意の表領域にあるすべてのオブジェクトが転送されます。この例では1つの表のみが転送されていますが、ts_temp_sales表領域には複数の表が含まれることもあります。たとえば、データ・マートのリフレッシュは、最新の月の売上トランザクションによってのみでなく、CUSTOMER表の新規コピーによって行われることもあります。これらの2つの表は、どちらも同じ表領域に転送できます。また、この表領域には、索引など他のデータベース・オブジェクトも含めることができ、それらも同様に転送されます。

また、トランスポータブル表領域操作では、複数の表領域を同時に転送できます。これによって、非常に大量のデータも簡単にデータベース間で移動できます。ただし、トランスポータブル表領域機能で転送できるのは、他の表領域に依存性を持たないデータベース・オブジェクトの完全な集合を含む表領域の集合のみであることに注意してください。たとえば、索引は、元となる表がなければ転送できません。また、パーティションも表の残りの部分がないと転送できません。DBMS_TTSパッケージを使用すると、表領域がトランスポータブルであるかどうかをチェックできます。

関連項目:

DBMS_TTSパッケージの詳細は、『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ステップ1では、1月の売上データを別々の表領域にコピーしました。ただし、別々の表領域にデータを移動しなくても、トランスポータブル表領域を利用できる場合もあります。SALES表がデータ・ウェアハウス内で月別にパーティション化されており、各パーティションが専用の表領域にある場合、1月のデータを含む表領域を直接転送できることがあります。たとえば、表領域ts_sales_jan2000に、1月のパーティションsales_jan2000が配置されているとします。この場合、1月の売上データを一時的にts_temp_salesにコピーするのではなく、ts_sales_jan2000表領域を転送できることがあります。

ただし、表領域ts_sales_jan2000を転送するには、特別に作成された表領域の場合と同じ条件を満たす必要があります。第1に、この表領域をREAD ONLYに設定する必要があります。第2に、パーティション表のパーティション1つのみを転送することはできない(パーティション表の残りの部分も転送する必要がある)ため、1月のデータのみを転送するには、(ALTER TABLE文を使用して)1月のパーティションを別の表に変換する必要があります。EXCHANGE操作は非常に高速です。ただし、1月のデータは基礎となるsales表の一部ではなくなるため、メタデータのエクスポート後に変換してsales表に戻さないかぎり、ユーザーからはアクセスできなくなります。1月のデータは、データファイルおよびエクスポート・ファイルをターゲット・システムにコピーするのステップの完了後に変換してsales表に戻すことができます。

メタデータのエクスポート

転送した表領域に含まれるオブジェクトを記述するメタデータをエクスポートするには、エクスポート・ユーティリティを使用します。これまでの例に合せてエクスポート・コマンドを記述すると、次のようになります。

```
EXP TRANSPORT_TABLESPACE=y TABLESPACES=ts_temp_sales FILE=jan_sales.dmp
```

この操作では、エクスポート・ファイルjan_sales.dmpが生成されます。このエクスポート・ファイルは、含まれているのがメタデータのみのためサイズは小さくなっています。この場合、エクスポート・ファイルには、列名、列のデータ型などの表temp_jan_salesを説明する情報、およびターゲットOracleデータベースがts_temp_sales内のオブジェクトへのアクセスに必要とするその他の情報すべてが含まれます。

データファイルおよびエクスポート・ファイルをターゲット・システムにコピーする

ts_temp_salesを構成するデータファイル、およびエクスポート・ファイルjan_sales.dmpを、いずれかのフラット・ファイル転送メカニズムを使用してデータ・マート・プラットフォームにコピーします。データファイルのコピーが終わると、必要に応じて表領域ts_temp_salesをREAD WRITEモードに設定できます。

メタデータのインポート

ファイルをデータ・マートにコピーした後に、メタデータをデータ・マートにインポートします。

```
IMP TRANSPORT_TABLESPACE=y DATAFILES='/db/tempjan.f'
    TABLESPACES=ts_temp_sales FILE=jan_sales.dmp
```

この時点で、表領域ts_temp_salesおよび表temp_sales_janが、データ・マート内でアクセス可能になります。この新しいデータをデータ・マートの表に取込むことができます。

temp_sales_jan表からデータ・マートのsales表にデータを挿入するには、2通りの方法があります。

```
INSERT /*+ APPEND */ INTO sales SELECT * FROM temp_sales_jan;
```

この操作の後に、temp_sales_jan表(およびts_temp_sales表領域全体)を削除できます。

または、データ・マートのsales表が月別にパーティション化されている場合に、新しく転送した表領域およびtemp_sales_jan表をそのデータ・マートの永続部分にすることが可能です。temp_sales_jan表は、データ・マートのsales表のパーティションになることができます。

```
ALTER TABLE sales ADD PARTITION sales_00jan VALUES
  LESS THAN (TO_DATE('01-feb-2000','dd-mon-yyyy'));
ALTER TABLE sales EXCHANGE PARTITION sales_00jan
  WITH TABLE temp_sales_jan INCLUDING INDEXES WITH VALIDATION;
```

17.2.3.2 トランスポータブル表領域の他の用途

前述の例では、データ・ウェアハウスにデータを転送する代表的な例を示しました。トランスポータブル表領域は、この他にも様々な目的に使用できます。データ・ウェアハウス環境では、トランスポータブル表領域は、Oracleデータベース間で大量のデータを移動する(インポート/エクスポート、またはSQL*Loaderのような)ユーティリティとみなすことができます。CREATE TABLE ... AS SELECT文やINSERT ... AS SELECT文などのパラレル・データ移動操作とともに使用すると、トランスポータブル表領域は、様々な目的でのデータの高速転送用の重要なメカニズムを提供します。

18 データ・ウェアハウスにおけるロードおよび変換

この章は、データ・ウェアハウスの作成および管理に有効な情報について説明します。内容は次のとおりです。

- データ・ウェアハウスにおけるロードおよび変換の概要
- データ・ウェアハウスのロード・メカニズム
- データ・ウェアハウスにおける変換メカニズム
- エラーのロギングおよび処理のメカニズム
- ロードおよび変換の使用例

18.1 データ・ウェアハウスにおけるロードおよび変換の概要

データ変換は最も複雑で、抽出、変換およびロード(ETL)プロセスの中で最も処理時間がかかることがあります。データ変換では、単純なデータ変換からかなり複雑なデータのクレンジング・テクニックまで実行できます。変換は、データベース外(フラット・ファイルなど)でも実装されることがありますが、ほとんどの場合はOracleデータベース内で行います。

この章では、Oracle Database内でスケーラブルで効果的なデータ変換を実装するテクニックについて説明します。この章の例は比較的単純です。実際のデータ変換は、通常、はるかに複雑です。ただし、この章で説明した変換テクニックは、実際のデータ変換要件のほとんどを満たしており、その他の方法よりスケーラブルで、少ないプログラミングで済みます。

この章では、データ・ウェアハウスで発生する一般的な変換をすべて説明しているわけではありません。これらの変換を実装するために使用できる基本的なテクニックの種類を示し、最適なテクニックの選択方法を説明します。

18.1.1 データ・ウェアハウス: 変換フロー

アーキテクチャの観点では、データ変換には次の方法があります。

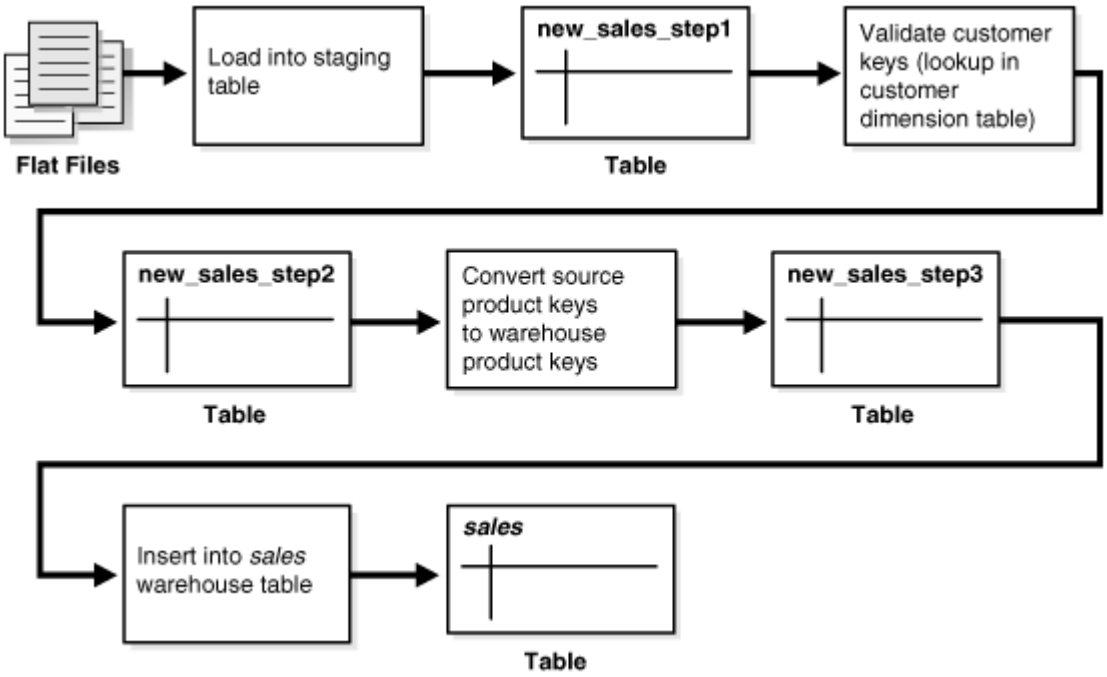
- データ・ウェアハウスにおけるマルチステージ・データ変換
- データ・ウェアハウスにおけるパイプライン・データ変換
- データ・ウェアハウスにおけるステージング・エリア

18.1.1.1 データ・ウェアハウスでのマルチステージ・データ変換

ほとんどのデータ・ウェアハウスのデータ変換のロジックは、複数のステップから構成されています。たとえば、sales表に挿入するために新しいレコードを変換する場合、各ディメンション・キーの妥当性チェックを行うには、個別のロジック変換ステップに従う必要がある場合があります。

図18-1は、変換ロジックをグラフィカルに示したものです。

図18-1 マルチステージ・データ変換



Oracle Databaseを変換エンジンとして使用する場合、一般的な方法では、変換をそれぞれ別々のSQL操作として実装し、各ステップの処理結果を格納するために別々の一時的なステージング表(図18-1の表new_sales_step1やnew_sales_step2など)を作成します。また、ロードしてから変換するこの方法では、変換プロセス全体にチェックポイント取得を実行するスキームが提供されます。このスキームによって、プロセスの監視および再起動が簡単になります。ただし、マルチステージングには、領域と時間の必要性が増大するというデメリットもあります。

また、多数の単純なロジック変換を、単一のSQL文または単一のPL/SQLプロシージャに結合することも可能です。このような結合を行うと、各ステップを個別に実行するよりパフォーマンスが向上することがありますが、同時に、個々の変換の変更、追加、削除や、失敗した変換からのリカバリが困難になる場合があります。

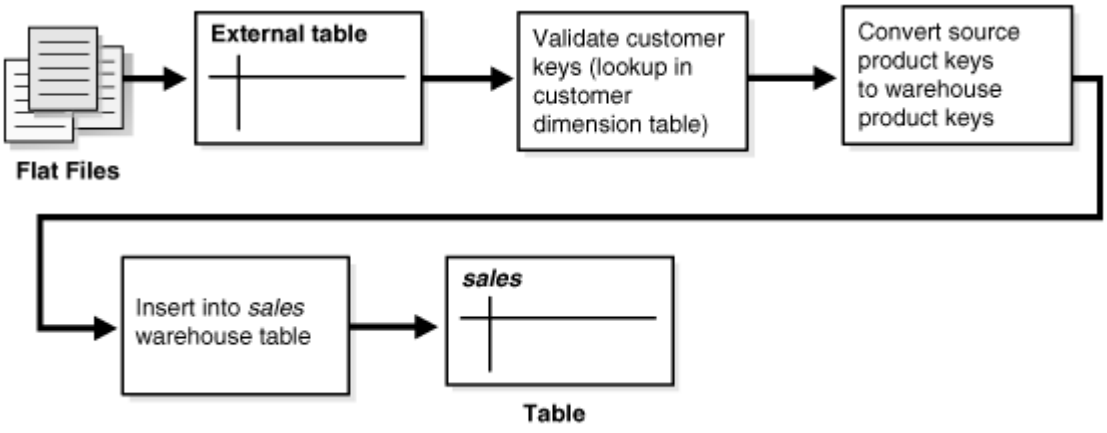
18.1.1.2 データ・ウェアハウスでのパイプライン・データ変換

ETLプロセス・フローは動的に変更でき、データベースはETLソリューションに不可欠の部品になります。

新機能により従来は必要だったプロセス・ステップの一部が廃止になりますが、改造してデータ・フローとデータ変換を強化し、よりスケーラブルで中断のないものにできます。タスクは変換してからロードするという(ほとんどのタスクがデータベース外部で実行される)シリアル・プロセスや、ロードしてから変換するというプロセスから拡張され、ロードしながら変換するプロセスへとシフトします。

Oracleでは、ETLシナリオに関連するすべての問題とタスクに対処できるように、多様な新機能が用意されています。データベースは、汎用的なソリューションに対処するのではなく、ツールキット機能を提供するということを理解する必要があります。基礎となるデータベースは、特定の顧客のニーズに合せて最も適切なETLプロセス・フローを有効化する必要があり、技術的な観点での要件や制約がないようにする必要があります。図18-2に、以降で説明する新機能を示します。

図18-2 パイプライン・データ変換



18.1.1.3 データ・ウェアハウスのステージング領域

ロードの全体的な速度は、生データをいかに速くステージング領域から読み取ることができるか、データベースのターゲット表に書き込むことができるかにより決まります。生データはできるだけ多くの物理ディスクにステージングし、生データの読取りがロード時のボトルネックにならないようにすることを強くお薦めします。

データをステージングするのに最適な場所はOracle Database File System(DBFS)です。DBFSはSecureFile LOBとしてデータベースに格納されたファイルへのアクセスに使用できるマウント可能なファイルシステムを作成します。DBFSはローカルのファイルシステムのように見える共有ネットワーク・ファイルシステムを提供するという点でNFSと似ています。DBFSはデータ・ウェアハウスとは別のデータベースに作成し、DIRECT_IOオプションを使用してマウントして、ファイルシステムとの間の生データ・ファイルの移動時にシステム・ページ・キャッシュの競合が起きないようにします。DBFSの設定の詳細は、『Oracle Database SecureFilesおよびラージ・オブジェクト開発者ガイド』を参照してください。

18.1.2 バッチ更新および表のオンライン再定義について

表の一括更新は、EXECUTE_UPDATEプロシージャを使用して最適化できます。REDOログに更新が記録されないため、パフォーマンスが最適化されます。

DBMS_REDEFINITION.EXECUTE_UPDATEプロシージャを使用すると、UPDATE文を直接挿入モードで実行できます。この操作中はREDOが記録されないため、メディア・リカバリを使用して再定義やデータの更新をリカバリできません。リカバリを可能にするには、再定義を開始する前にデータベースまたは表領域のバックアップを実行することをお薦めします。

関連項目:

Oracle Database管理者ガイド

18.1.3 ETL操作の監視の概要

ETLは複雑になり、パフォーマンスが低下する可能性があるため、Oracle Databaseでは、ETLの計画に含まれるデータベース操作を監視およびレポートできるユーザー・インタフェースを提供しています。

データベース操作は、一連の関連するデータベース・タスクが含まれているユーザー定義の論理オブジェクトです。たとえば、エンド・ユーザーまたはアプリケーション・コードによって定義されたETL処理ジョブです。各データベース操作は、名前および実行IDによって一意に識別でき、複数回実行できます。

データベース操作の監視は、最適とまでいかないジョブのトラブルシューティングに大変有効で、任意のステップにおいて、どこで、どのようにリソースが消費されているのかを特定するのに役立ちます。関連情報の追跡、パフォーマンス・ボトルネックの特定およびデータベース・パフォーマンスの問題の調整のための時間の短縮が可能になります。Oracle Database 12cリリース2(12.2)以降では、DBMS_SQL_MONITOR.BEGIN_OPERATION関数にセッションIDおよびシリアル番号を指定することによって、任意のセッションでデータベース操作を開始できます。

関連項目:

Oracle Database SQLチューニング・ガイド

18.2 データ・ウェアハウスでのロード・メカニズム

データ・ウェアハウスへのロードには、次のメカニズムを使用できます。

- SQL*Loaderを使用したデータ・ウェアハウスのロード
- 外部表を使用したデータ・ウェアハウスのロード
- OCIおよびダイレクト・パスAPIを使用したデータ・ウェアハウスのロード
- エクスポート/インポートを使用したデータ・ウェアハウスのロード

18.2.1 SQL*Loaderを使用したデータ・ウェアハウスのロード

データ変換をデータベース内で行うには、生データがデータベースでアクセス可能になっている必要があります。その方法の1つは、データベースにロードすることです。データをOracleデータ・ウェアハウスに転送するいくつかのテクニックについては、データ・ウェアハウスにおける転送を参照してください。データを転送する最も一般的なテクニックは、フラット・ファイルを使用する方法です。

データをフラット・ファイルからOracleデータ・ウェアハウスへ移動するには、SQL*Loaderを使用します。このデータのロード中に、SQL*Loaderを使用して、基本的なデータ変換の実装を行うこともできます。ダイレクト・パスSQL*Loaderを使用すると、データ型変換や単純なNULL処理などの基本的なデータ操作は、データのロード中に自動的に解決できます。パフォーマンスの理由から、ほとんどのデータ・ウェアハウスでは、ダイレクト・パス・ロードが選択されます。

従来型パス・ローダーは、ダイレクト・パス・ローダーより広範囲な機能をデータ変換で使用できます。SQL関数は、列の値がロードされるときに、すべての列に適用できます。これによって、データのロード中に、豊富な機能を使用して変換できるようになります。ただし、従来型パス・ローダーはダイレクト・パス・ローダーより低速です。このような理由から、従来型パス・ローダーを使用するのは、主に少量のデータをロードおよび変換する場合にしてください。

データ・ウェアハウスでは、直接パス・モードを使用してバッチ更新を実行することによって、REDOデータを保守するためのオーバーヘッドを回避できます。バッチ更新は、表のオンライン再定義中に表に対して実行できます。

ここでは、SQL*Loader制御ファイルの単純な例を示します。このファイルでは、外部ファイルsh_sales.datからshサンプル・スキーマのsales表にデータがロードされます。外部フラット・ファイルsh_sales.datは、日次レベルで集計された売上トランザクション・データで構成されています。この外部ファイルのすべての列がsales表にロードされるわけではありません。この外部ファイルは、shサンプル・スキーマの2番目のファクト表をロードするためのソースとしても使用されます。これには、外部表が使用されます。

次に、sales表をロードする制御ファイル(sh_sales.ctl)を示します。

```
LOAD DATA INFILE sh_sales.dat APPEND INTO TABLE sales
FIELDS TERMINATED BY "|"
(PROD_ID, CUST_ID, TIME_ID, CHANNEL_ID, PROMO_ID, QUANTITY_SOLD, AMOUNT_SOLD)
```

このロードには、次のコマンドを使用できます。

```
$  sqlldr control=sh_sales.ctl direct=true
Username:
Password:
```

SQL*Loader Expressモードの場合、制御ファイルは使用しません。かわりに、表の列定義を使用して入力データ型を特定します。

関連項目:

- 詳細は、『Oracle Databaseユーティリティ』を参照してください。

- DBMS_REDEFINITIONを使用した一括更新の詳細は、『Oracle Database管理者ガイド』を参照してください。

18.2.2 外部表を使用したデータ・ウェアハウスのロード

外部データソースを処理するもう1つのアプローチは、外部表を使用することです。Oracleの外部表機能では、外部データを直接およびパラレルに問い合せて結合できる仮想表として使用できます。この場合、外部データを最初にデータベースにロードする必要はありません。そのまま、SQL、PL/SQLおよびJavaを使用して外部データにアクセスできます。

外部表を使用すると、ロード・フェーズを変換フェーズとパイプライン化できます。データ・ストリーミングを中断せずに、変換プロセスをロード・プロセスとマージできます。データベースで比較や変換などの処理を行うために、データベース内でデータをステージングする必要はなくなります。たとえば、外部表からのSELECTと組み合せたダイレクト・パスのINSERT AS SELECT文で、従来型ロードの変換機能を使用できます。Oracle Database 12cより、データベースではバルク・ロード操作の一部として(CTASおよびIAS)、索引の作成時の統計の収集方法に似た表統計の自動収集を行います。データ・ロードの際に統計を収集することによって、スキャン操作の追加を回避し、ユーザーがデータを使用できるようになるとすぐに必要な統計を提供できます。

外部表と通常の表の主な違いは、外部で編成された表は読取り専用であることです。DML操作(UPDATE/INSERT/DELETE)は実行できず、索引も作成できません。

外部表は既存のSQL*Loader機能の大部分に準拠しており、ほとんどの場合において、より優れた機能を提供します。外部表は、外部ソース全体を既存のデータベース・オブジェクトと結合する必要がある環境、またはデータを複雑な方法で変換する必要がある環境で特に有効です。たとえば、SQL*Loaderとは異なり、任意のSQL変換を適用してダイレクト・パス・インサートを使用できます。さらに、ファイル(圧縮されたデータファイルなど)を処理する実行対象のプログラム(zcatなど)を指定し、その出力(圧縮されていないデータファイルなど)をOracle Databaseで利用できるようにすることもできます。つまり、大量の圧縮データをロードする際に、先にそのデータをディスクに展開する必要がないということです。

外部ファイルsh_sales.gzに表される、売上トランザクション・データ全体の構造を表す外部表sales_transactions_extを作成できます。製品部門では、製品と時間単位の原価分析が特に重要です。したがって、shスキーマにファクト表costを作成します。処理するソース・データは、salesファクト表の場合と同じです。ただし、提供される全ディメンション情報を調査するわけではないため、costファクト表のデータはsalesファクト表のデータより疎い密度になっています。たとえば、すべての異なる物流チャネルが1つに集計されます。

ディメンションの一部は使用されないため、前述のような詳細情報の集計を行わないとcostファクト表にデータをロードできません。

外部表フレームワークは、そのソリューションを提供します。SQL*Loaderでは集計を適用する前にデータをロードする必要がありましたが、それとは異なり、次のように単一のSQL DML文中にロードと変換を組み合せることができます。ターゲット表に挿入する前にデータを一時的にステージングする必要はありません。

オブジェクト・ディレクトリがすでに存在し、sh_sales.gzファイルを含むディレクトリと不良ファイルおよびログ・ファイルを含むディレクトリを指定する必要があります。

```
CREATE TABLE sales_transactions_ext
(PROD_ID NUMBER, CUST_ID NUMBER,
 TIME_ID DATE, CHANNEL_ID NUMBER,
 PROMO_ID NUMBER, QUANTITY_SOLD NUMBER,
 AMOUNT_SOLD NUMBER(10,2), UNIT_COST NUMBER(10,2),
 UNIT_PRICE NUMBER(10,2))
ORGANIZATION external (TYPE oracle_loader
  DEFAULT DIRECTORY data_file_dir ACCESS PARAMETERS
  (RECORDS DELIMITED BY NEWLINE CHARACTERSET US7ASCII
    PREPROCESSOR EXECDIR:'zcat'
    BADFILE log_file_dir:'sh_sales.bad_xt'
```

```
    LOGFILE log_file_dir:'sh_sales.log_xt'
    FIELDS TERMINATED BY "|" LDRTRIM
    ( PROD_ID, CUST_ID,
      TIME_ID         DATE(10) "YYYY-MM-DD",
      CHANNEL_ID, PROMO_ID, QUANTITY_SOLD, AMOUNT_SOLD,
      UNIT_COST, UNIT_PRICE))
  location ('sh_sales.gz')
)REJECT LIMIT UNLIMITED;
```

これで、外部表をデータベースから使用し、外部データの一部の列にのみアクセスし、データをグルーピングして、costsファクト表に挿入できます。

```
INSERT /*+ APPEND */ INTO COSTS
(TIME_ID, PROD_ID, UNIT_COST, UNIT_PRICE)
SELECT TIME_ID, PROD_ID, AVG(UNIT_COST), AVG(amount_sold/quantity_sold)
FROM sales_transactions_ext GROUP BY time_id, prod_id;
```

関連項目:

- 外部表の構文の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。
- 使用例は『*Oracle Databaseユーティリティ*』を参照してください。

18.2.2.1 DBMS_CLOUDを使用したオブジェクト・ストア・データの外部表の作成

DBMS_CLOUD PL/SQLパッケージを使用すると、データ・ウェアハウスをクラウド内のオブジェクト・ストアに接続できます。

DBMS_CLOUDは、外部表を作成し、クラウドに格納されているファイルおよびオブジェクトからデータへのアクセスを可能にするAPIを提供します。テキスト・ファイル、Parquetファイル、Avroファイルおよびクラウド内のデータ・ポンプ・ファイルから外部表にデータをロードできます。

オブジェクト・ストアに対する認証は、ユーザー名とパスワードを含む、個別に作成された資格証明オブジェクトを介して取得されます。オブジェクト・ストア管理者は、これらの資格証明を指定し、ストア内のデータにアクセスするための適切な権限をユーザーにプロビジョニングする必要があります。

このパッケージは、Oracle Object Storage、Microsoft Azure Blob StorageおよびAmazon S3からのファイルのロードをサポートしています。

関連項目:

DBMS_CLOUD APIについて説明しているDatabase PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス。

18.2.3 OCIおよびダイレクト・パスAPIを使用したデータ・ウェアハウスのロード

OCIおよびダイレクト・パスAPIが頻繁に使用されるのは、変換と計算がデータベースの外部で実行され、フラット・ファイルのステージングを必要としない場合です。

18.2.4 エクスポート/インポートを使用したデータ・ウェアハウスのロード

エクスポートとインポートは、データがそのままターゲット・システムに挿入される場合に使用されます。複雑な抽出は実行できま

せん。詳細は、データ・ウェアハウスにおける抽出を参照してください。

18.3 データ・ウェアハウスでの変換メカニズム

データベース内でデータを変換するには、次の方法があります。

- SQLを使用したデータの変換
- PL/SQLを使用したデータの変換
- テーブル・ファンクションを使用したデータの変換

18.3.1 SQLを使用したデータの変換

データがデータベースにロードされると、SQL操作を使用してデータ変換を実行できます。SQLデータ変換を実装するには、次の4つの基本的なテクニックがあります。

- CREATE TABLE ... AS SELECTおよびINSERT /*+APPEND*/ AS SELECT
- UPDATEを使用したデータの変換
- MERGEを使用したデータの変換
- マルチテーブル・インサートを使用したデータの変換

18.3.1.1 CREATE TABLE ... AS SELECTおよびINSERT /*+APPEND*/ AS SELECT

CREATE TABLE ... AS SELECT文(CTAS)は、大規模なデータセットを操作する場合に強力なツールです。後述する例のように、多くのデータ変換は標準SQLで記述でき、CTASにより、SQL問合せを効果的に実行してその問合せ結果を新しいデータベース表に格納する機能が提供されます。INSERT /*+APPEND*/ ...AS SELECT文の機能は、既存のデータベース表に対して同様の機能を提供します。

データ・ウェアハウス環境では、CTASは、最大のパフォーマンスを得るためにNOLOGGINGモードで、パラレルで実行されます。

簡単で一般的なデータ変換は、データの置換えです。データ置換えによる変換では、1つの列のいくつかまたはすべての値が変更されます。たとえば、sales表にchannel_id列があるとします。この列は、指定された売上トランザクションが企業自体の売上(直接販売)によるものか、または販売店(間接販売)によるものかを指定するために使用されます。

データ・ウェアハウスの複数のソース・システムからデータを受け取る場合があるとします。これらのソース・システムのうち、直接販売のみを処理し、間接販売チャネルには関与していないソース・システムがあるとします。データ・ウェアハウスがこのシステムから売上データを最初に受け取ると、すべての売上レコードのsales.channel_idフィールドはNULL値になります。これらのNULL値は、適切なキー値に設定する必要があります。たとえば、ターゲットとなるsales表の文への挿入の一部としてSQL関数を使用すると、この操作を効率的に行うことができます。ソース表sales_activity_directの構造は、次のとおりです。

```
DESC sales_activity_direct
Name           Null?    Type
-------------  -----    ----------------
SALES_DATE              DATE
PRODUCT_ID              NUMBER
CUSTOMER_ID             NUMBER
PROMOTION_ID            NUMBER
AMOUNT                  NUMBER
QUANTITY                NUMBER
```

次のSQL文は、sales_activity_directのデータをサンプル・スキーマのsales表に挿入します。ここでは、売上日時(sales_date)の値を午前0時に切り捨てるSQL関数を使用し、固定チャネルIDとして3を割り当てます。

```
INSERT /*+ APPEND NOLOGGING PARALLEL */
INTO sales SELECT product_id, customer_id, TRUNC(sales_date), 3,
       promotion_id, quantity, amount
FROM sales_activity_direct;
```

18.3.1.2 UPDATEを使用したデータの変換

データ置換を実装するもう1つのテクニックは、UPDATE文を使用してsales.channel_id列を変更することです。UPDATEによって正しい結果が戻されます。ただし、データ置換による変換で多数(またはすべての行)の変更が必要であれば、UPDATEではなくCTAS文を使用する方が効率的な場合があります。

18.3.1.3 MERGEを使用したデータの変換

Oracle Databaseのマージ機能では、表または表外の単一表ビューに行を条件付きで更新または挿入できるように、SQLキーワードMERGEを導入することでSQLが拡張されます。条件は、ON句に指定します。これは、純粋な大量ロードと並んでデータ・ウェアハウスの同期化における最も一般的な操作です。

例18-1 SQLを使用したマージ操作

次の例では、マージの各種実装について説明します。この例では、ディメンション表productsの新規データをデータ・ウェアハウスに転送し、挿入または更新する必要があるものとします。表products_deltaの構造はproductsと同じです。

```
MERGE INTO products t USING products_delta s
ON (t.prod_id=s.prod_id)
WHEN MATCHED THEN UPDATE SET
  t.prod_list_price=s.prod_list_price, t.prod_min_price=s.prod_min_price
WHEN NOT MATCHED THEN INSERT (prod_id, prod_name, prod_desc, prod_subcategory,
  prod_subcategory_desc, prod_category, prod_category_desc, prod_status,
  prod_list_price, prod_min_price)
VALUES (s.prod_id, s.prod_name, s.prod_desc, s.prod_subcategory,
  s.prod_subcategory_desc, s.prod_category, s.prod_category_desc,
  s.prod_status, s.prod_list_price, s.prod_min_price);
```

18.3.1.4 マルチテーブル・インサートを使用したデータの変換

多くの場合、外部データソースは、様々なターゲット・オブジェクトに挿入できるように、論理属性に基づいて分離する必要があります。また、通常、データ・ウェアハウス環境では、同じソース・データが複数のターゲット・オブジェクトに分岐しています。マルチテーブル・インサートでは、この種の変換用に新しいSQL文が用意されており、ビジネス上の変換ルールに応じてデータを複数または1つのターゲットにできます。この挿入は、ビジネス・ルールに基づいて条件付きで行う方法と、無条件で行う方法があります。

これには、複数の表がターゲットとなっている場合に、INSERT ... SELECT文を使用できるというメリットがあります。これにより、2つの別の方法でのデメリットを回避できます。従来は、*n*個の独立したINSERT … SELECT文を取り扱う必要があり、同じソース・データを*n*回処理するため、変換による作業負荷も*n*倍になっていました。または、プロシージャによるアプローチを選択し、挿入の処理方法を行ごとに判断する必要がありました。このソリューションには、SQLで使用可能な高速のアクセス・パスへのダイレクト・アクセスが欠けています。

既存の文を使用する場合と同様に、新しいINSERT ... SELECT文も、パラレル化してダイレクト・ロード機能とともに使用することでパフォーマンスを改善できます。

例18-2 無条件の挿入

次の文では、sales_activity_directに格納されているトランザクションの売上情報が日次で集計され、今日のsalesおよびcostsファクト表に挿入されます。

```
INSERT ALL
   INTO sales VALUES (product_id, customer_id, today, 3, promotion_id,
```

```
                       quantity_per_day, amount_per_day)
   INTO costs VALUES (product_id, today, promotion_id, 3,
                       product_cost, product_price)
SELECT TRUNC(s.sales_date) AS today, s.product_id, s.customer_id,
  s.promotion_id, SUM(s.amount) AS amount_per_day, SUM(s.quantity)
  quantity_per_day, p.prod_min_price*0.8 AS product_cost, p.prod_list_price
  AS product_price
FROM sales_activity_direct s, products p
WHERE s.product_id = p.prod_id AND TRUNC(sales_date) = TRUNC(SYSDATE)
GROUP BY TRUNC(sales_date), s.product_id, s.customer_id, s.promotion_id,
  p.prod_min_price*0.8, p.prod_list_price;
```

例18-3 条件付きのALL挿入

次の文では、有効な宣伝を伴うすべての売上トランザクションについて、salesおよびcosts表に1行が挿入され、ある顧客による複数の同一注文に関する情報が別の表cum_sales_activityに格納されます。売上トランザクションには、2行を挿入できるものと、まったく挿入できないものがあります。

```
INSERT ALL
WHEN promotion_id IN (SELECT promo_id FROM promotions) THEN
   INTO sales VALUES (product_id, customer_id, today, 3, promotion_id,
                       quantity_per_day, amount_per_day)
   INTO costs VALUES (product_id, today, promotion_id, 3,
                       product_cost, product_price)
WHEN num_of_orders > 1 THEN
   INTO cum_sales_activity VALUES (today, product_id, customer_id,
     promotion_id, quantity_per_day, amount_per_day, num_of_orders)
SELECT TRUNC(s.sales_date) AS today, s.product_id, s.customer_id,
   s.promotion_id, SUM(s.amount) AS amount_per_day, SUM(s.quantity)
   quantity_per_day, COUNT(*) num_of_orders, p.prod_min_price*0.8
   AS product_cost, p.prod_list_price AS product_price
FROM sales_activity_direct s, products p
WHERE s.product_id = p.prod_id
AND TRUNC(sales_date) = TRUNC(SYSDATE)
GROUP BY TRUNC(sales_date), s.product_id, s.customer_id,
 s.promotion_id, p.prod_min_price*0.8, p.prod_list_price;
```

例18-4 条件付きのFIRST挿入

次の文では、製品注文の合計数量と重量に従って、適切な出荷リストに挿入されます。例外は大量注文の場合で、重量区分が大きすぎないかぎり急送されます。数量のない注文として表されているこの単純な例では、すべての不適切な注文は別の表に格納されます。ここでは、適切な表large_freight_shipping、express_shipping、default_shippingおよびincorrect_sales_orderが存在するものとします。

```
INSERT FIRST WHEN (sum_quantity_sold > 10 AND prod_weight_class < 5) AND
sum_quantity_sold >=1) OR (sum_quantity_sold > 5 AND prod_weight_class > 5) THEN
  INTO large_freight_shipping VALUES
      (time_id, cust_id, prod_id, prod_weight_class, sum_quantity_sold)
  WHEN sum_amount_sold > 1000 AND sum_quantity_sold >=1 THEN
  INTO express_shipping VALUES
      (time_id, cust_id, prod_id, prod_weight_class,
       sum_amount_sold, sum_quantity_sold)
WHEN (sum_quantity_sold >=1) THEN INTO default_shipping VALUES
      (time_id, cust_id, prod_id, sum_quantity_sold)
ELSE INTO incorrect_sales_order VALUES (time_id, cust_id, prod_id)
SELECT s.time_id, s.cust_id, s.prod_id, p.prod_weight_class,
       SUM(amount_sold) AS sum_amount_sold,
       SUM(quantity_sold) AS sum_quantity_sold
FROM sales s, products p
```

```
WHERE s.prod_id = p.prod_id AND s.time_id = TRUNC(SYSDATE)
GROUP BY s.time_id, s.cust_id, s.prod_id, p.prod_weight_class;
```

例18-5 条件付き挿入と無条件の挿入の混在型

次の例では、新規顧客がcustomers表に挿入され、cust_credit_limitが4501以上のすべての新規顧客がさらなる販促対象として別個の表に格納されます。

```
INSERT FIRST WHEN cust_credit_limit >= 4500 THEN INTO customers
  INTO customers_special VALUES (cust_id, cust_credit_limit)
  ELSE INTO customers
SELECT * FROM customers_new;
```

関連項目:

MERGE操作の詳細は、マテリアライズド・ビューのリフレッシュを参照してください。

18.3.2 PL/SQLを使用したデータの変換

データ・ウェアハウス環境では、PL/SQLなどの手続き型言語を使用して、複雑な変換をOracle Databaseで実装できます。CTASが表全体を操作し、パラレル化を強調するのに対して、PL/SQLでは、行ベースの方法で、非常に高度な変換規則に対応できます。たとえば、PL/SQLプロシージャを使用して、複数のカーソルをオープンして複数のソース表からデータを読み取り、複雑なビジネス・ルールを使用してこのデータを結合できます。これによって、変換されたデータを1つ以上のターゲット表に挿入できます。標準SQL文を使用して、同じ手順の操作を表現するのは困難または不可能です。

手続き型言語を使用すると、複雑なETL処理内で特定の変換(または多数の変換ステップ)をカプセル化し、中間のステージング・エリアからデータを読み取り、出力として新しい表オブジェクトを生成できます。以前に生成された変換の入力表と後続の変換には、この特定の変換により生成される表が使用されます。また、ETLプロセス全体でこのようにカプセル化された変換ステップをシームレスに統合できるため、相互の行セットがストリーム化され、中間的なステージングが不要になります。テーブル・ファンクションを使用すると、このような動作を実装できます。

18.3.3 テーブル・ファンクションを使用したデータの変換

テーブル・ファンクションにより、PL/SQL、CまたはJavaで実装された変換のパイプライン実行またはパラレル実行がサポートされます。 前述のシナリオは、中間的なステージング表を使用せずに実行でき、各種変換ステップでのデータ・フローは中断されません。テーブル・ファンクションに関する詳細情報は、テーブル・ファンクションに記載されています。

18.3.3.1 テーブル・ファンクション

テーブル・ファンクションは、出力として行セットを生成できる関数として定義されます。また、テーブル・ファンクションは入力として行セットを使用できます。Oracle9iまでのPL/SQLファンクションには、次のようなデメリットがありました。

- 入力としてカーソルを使用できません。
- パラレル化またはパイプライン化できません。

現在、ファンクションにこのような制限はありません。テーブル・ファンクションでは、次のことができるため、データベース機能が拡張されます。

- 1つの関数から複数行を戻すことができます。
- SQL副問合せ(複数行の選択)の結果を関数に直接渡すことができます。

- 関数に入力としてカーソルを使用できます。
- 関数をパラレル化できます。
- 結果セットが作成されるとすぐに次の処理に段階的に渡します。これは段階的パイプライン処理と呼ばれます。

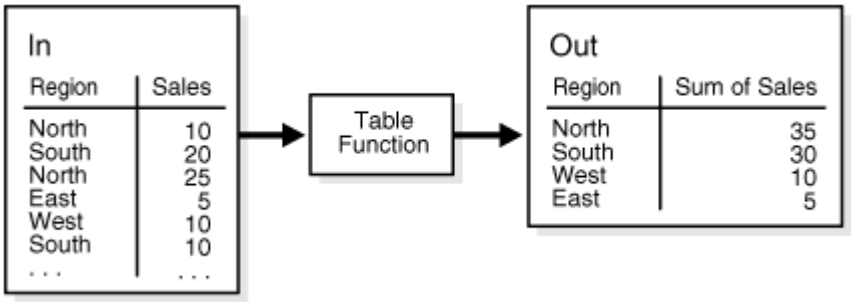
テーブル・ファンクションは、ネイティブなPL/SQLインタフェースを使用してPL/SQLで定義するか、Oracle Data Cartridge Interface(ODCI)を使用してJavaまたはCで定義できます。

関連項目:

- 詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- 詳細は、『*Oracle Databaseデータ・カートリッジ開発者ガイド*』を参照してください。

行セットを入力してSUM操作の実行後に行セットを出力する典型的な集計を、図18-3に示します。

図18-3 テーブル・ファンクションの例



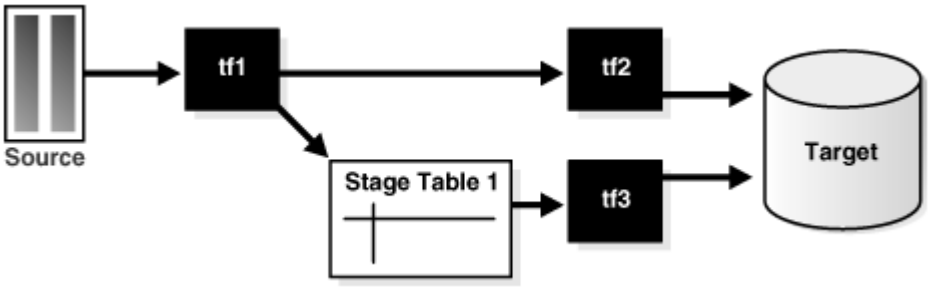
この操作のための疑似コードは次のようになります。

```
INSERT INTO Out SELECT * FROM ("Table Function"(SELECT * FROM In));
```

このテーブル・ファンクションは、InでのSELECTの結果を入力として使用し、レコード・セットを異なるフォーマットでダイレクト挿入用の出力としてOutに渡します。

また、テーブル・ファンクションでは、データをアトミック・トランザクションの適用範囲内で分岐させることができます。これは、効率的なロギング・メカニズムや、他の独立した変換の分岐など、様々な場合に使用できます。このような使用例では、単一のステージング表が必要です。

図18-4 分岐を伴うパイプライン・パラレル変換



このための疑似コードは次のようになります。

```
INSERT INTO target SELECT * FROM (tf2(SELECT *
FROM (tf1(SELECT * FROM source))));
```

この場合は、targetに挿入され、アトミック・トランザクションの適用範囲内でtf1の一部としてStage Table 1にも挿入されます。

```
INSERT INTO target SELECT * FROM tf3(SELT * FROM stage_table1);
```

関連項目:

- テーブル・ファンクションの詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- PL/SQL以外の言語で実装されるテーブル・ファンクションの詳細は、『*Oracle Databaseデータ・カートリッジ開発者ガイド*』を参照してください。

テーブル・ファンクションの例を実行する前に作成するオブジェクト

次の例は、テーブル・ファンクションの基本を示しており、これらの関数内に実装された複雑なビジネス・ルールは使用されていません。あくまでも具体例を示すことが目的であり、すべてPL/SQLで実装されています。

テーブル・ファンクションはレコード・セットを戻し、入力としてカーソルを使用できます。この例を使用する前に、shサンプル・スキーマの他に次のデータベース・オブジェクトを設定する必要があります。

```
CREATE TYPE product_t AS OBJECT (
      prod_id                    NUMBER(6)
    , prod_name                  VARCHAR2(50)
    , prod_desc                  VARCHAR2(4000)
    , prod_subcategory           VARCHAR2(50)
    , prod_subcategory_desc      VARCHAR2(2000)
    , prod_category              VARCHAR2(50)
    , prod_category_desc         VARCHAR2(2000)
    , prod_weight_class          NUMBER(2)
    , prod_unit_of_measure       VARCHAR2(20)
    , prod_pack_size             VARCHAR2(30)
    , supplier_id                NUMBER(6)
    , prod_status                VARCHAR2(20)
    , prod_list_price            NUMBER(8,2)
    , prod_min_price             NUMBER(8,2)
);
/
CREATE TYPE product_t_table AS TABLE OF product_t;
/
COMMIT;

CREATE OR REPLACE PACKAGE cursor_PKG AS
  TYPE product_t_rec IS RECORD (
      prod_id                     NUMBER(6)
    , prod_name                   VARCHAR2(50)
    , prod_desc                   VARCHAR2(4000)
    , prod_subcategory            VARCHAR2(50)
    , prod_subcategory_desc       VARCHAR2(2000)
    , prod_category               VARCHAR2(50)
    , prod_category_desc          VARCHAR2(2000)
    , prod_weight_class           NUMBER(2)
    , prod_unit_of_measure        VARCHAR2(20)
    , prod_pack_size              VARCHAR2(30)
    , supplier_id                 NUMBER(6)
    , prod_status                 VARCHAR2(20)
    , prod_list_price             NUMBER(8,2)
    , prod_min_price              NUMBER(8,2));
  TYPE product_t_rectab IS TABLE OF product_t_rec;
  TYPE strong_refcur_t IS REF CURSOR RETURN product_t_rec;
  TYPE refcur_t IS REF CURSOR;
END;
/
```

```
REM artificial help table, used later
CREATE TABLE obsolete_products_errors (prod_id NUMBER, msg VARCHAR2(2000));
```

例18-6 テーブル・ファンクションの例: 基本的な例

この例は、prod_category Electronicsを除くすべての廃止製品を表示する単純なフィルタ処理です。このテーブル・ファンクションは、結果セットとしてレコード・セットを戻し、入力として弱い型指定を持つREF CURSORを使用しています。

```
CREATE OR REPLACE FUNCTION obsolete_products(cur cursor_pkg.refcur_t)
RETURN product_t_table
IS
    prod_id                    NUMBER(6);
    prod_name                  VARCHAR2(50);
    prod_desc                  VARCHAR2(4000);
    prod_subcategory           VARCHAR2(50);
    prod_subcategory_desc      VARCHAR2(2000);
    prod_category              VARCHAR2(50);
    prod_category_desc         VARCHAR2(2000);
    prod_weight_class          NUMBER(2);
    prod_unit_of_measure       VARCHAR2(20);
    prod_pack_size             VARCHAR2(30);
    supplier_id                NUMBER(6);
    prod_status                VARCHAR2(20);
    prod_list_price            NUMBER(8,2);
    prod_min_price             NUMBER(8,2);
    sales NUMBER:=0;
    objset product_t_table := product_t_table();
    i NUMBER := 0;
BEGIN
   LOOP
     -- Fetch from cursor variable
     FETCH cur INTO prod_id, prod_name, prod_desc, prod_subcategory,
    prod_subcategory_desc, prod_category, prod_category_desc, prod_weight_class,
    prod_unit_of_measure, prod_pack_size, supplier_id, prod_status,
    prod_list_price, prod_min_price;
     EXIT WHEN cur%NOTFOUND; -- exit when last row is fetched
     -- Category Electronics is not meant to be obsolete and will be suppressed
     IF prod_status='obsolete' AND prod_category != 'Electronics' THEN
     -- append to collection
     i:=i+1;
     objset.extend;
     objset(i):=product_t( prod_id, prod_name, prod_desc, prod_subcategory,
     prod_subcategory_desc, prod_category, prod_category_desc,
     prod_weight_class, prod_unit_of_measure, prod_pack_size, supplier_id,
     prod_status, prod_list_price, prod_min_price);
     END IF;
   END LOOP;
   CLOSE cur;
   RETURN objset;
END;
/
```

このテーブル・ファンクションをSQL文に使用すると、次の結果が表示されます。ここでは、出力用にSQLの機能を追加使用しています。

```
SELECT DISTINCT UPPER(prod_category), prod_status
FROM TABLE(obsolete_products(
   CURSOR(SELECT prod_id, prod_name, prod_desc, prod_subcategory,
   prod_subcategory_desc, prod_category, prod_category_desc, prod_weight_class,
```

```
   prod_unit_of_measure, prod_pack_size,
   supplier_id, prod_status, prod_list_price, prod_min_price
          FROM products)));
```

例18-7 テーブル・ファンクションの例: REF CURSORを使用したフィルタ処理

この例は、例18-6と同じフィルタ処理を実装しています。2つの主な違いは次のとおりです。

- この例では、強い型指定を持つREF CURSORを入力として使用しており、次に示す例の1つのように、そのカーソルのオブジェクトに基づいてパラレル化できます。
- テーブル・ファンクションは、レコードの作成直後に段階的に結果セットを戻します。

```
CREATE OR REPLACE FUNCTION
  obsolete_products_pipe(cur cursor_pkg.strong_refcur_t) RETURN product_t_table
PIPELINED
PARALLEL_ENABLE (PARTITION cur BY ANY) IS
    prod_id                  NUMBER(6);
    prod_name                VARCHAR2(50);
    prod_desc                VARCHAR2(4000);
    prod_subcategory         VARCHAR2(50);
    prod_subcategory_desc    VARCHAR2(2000);
    prod_category            VARCHAR2(50);
    prod_category_desc       VARCHAR2(2000);
    prod_weight_class        NUMBER(2);
    prod_unit_of_measure    VARCHAR2(20);
    prod_pack_size          VARCHAR2(30);
    supplier_id             NUMBER(6);
    prod_status             VARCHAR2(20);
    prod_list_price         NUMBER(8,2);
    prod_min_price          NUMBER(8,2);
    sales NUMBER:=0;
BEGIN
 LOOP
     -- Fetch from cursor variable
     FETCH cur INTO prod_id, prod_name, prod_desc, prod_subcategory,
     prod_subcategory_desc, prod_category, prod_category_desc,
     prod_weight_class, prod_unit_of_measure, prod_pack_size, supplier_id,
     prod_status, prod_list_price, prod_min_price;
     EXIT WHEN cur%NOTFOUND; -- exit when last row is fetched
     IF prod_status='obsolete' AND prod_category !='Electronics' THEN
       PIPE ROW (product_t( prod_id, prod_name, prod_desc, prod_subcategory,
 prod_subcategory_desc, prod_category, prod_category_desc, prod_weight_class,
 prod_unit_of_measure, prod_pack_size, supplier_id, prod_status,
 prod_list_price, prod_min_price));
     END IF;
   END LOOP;
   CLOSE cur;
   RETURN;
END;
/
```

次のようなテーブル・ファンクションを使用できます。

```
SELECT DISTINCT prod_category,
                DECODE(prod_status,'obsolete','NO LONGER AVAILABLE','N/A')
FROM TABLE(obsolete_products_pipe(
  CURSOR(SELECT prod_id, prod_name, prod_desc, prod_subcategory,
         prod_subcategory_desc, prod_category, prod_category_desc,
         prod_weight_class, prod_unit_of_measure, prod_pack_size,
         supplier_id, prod_status, prod_list_price, prod_min_price
```

```
         FROM products)));
```

ここで、入力表productsの並列度を変更し、再度同じ文を発行します。

```
ALTER TABLE products PARALLEL 4;
```

セッション統計は、文がパラレル化されていることを示します。

```
SELECT * FROM V$PQ_SESSTAT WHERE statistic='Queries Parallelized';

STATISTIC              LAST_QUERY  SESSION_TOTAL
--------------------   ----------  -------------
Queries Parallelized            1              3

1 row selected.
```

例18-8 テーブル・ファンクションの例: 永続表への結果の移入

テーブル・ファンクションでは、結果を永続表の構造に分岐させることもできます。この例では、誤ってステータスobsoleteに設定されている特定のprod_category (デフォルトはElectronics)の製品を除き、それ以外のすべての廃止製品が関数によってフィルタ処理されます。テーブル・ファンクションの結果セットは、他のすべての廃止製品カテゴリで構成されます。検出された間違ったprod_idは、別個の表構造obsolete_products_errorに格納されます。テーブル・ファンクションが自律型トランザクションの一部である場合、サブプログラム呼出しでのエラーを回避するために、各PIPE ROW文の前にCOMMITまたはROLLBACKを実行する必要があることに注意してください。その結果セットは、他のすべての廃止製品カテゴリで構成されます。さらに、通常の変数をテーブル・ファンクションとともに使用する方法を示します。

```
CREATE OR REPLACE FUNCTION obsolete_products_dml(cur cursor_pkg.strong_refcur_t,
 prod_cat varchar2 DEFAULT 'Electronics') RETURN product_t_table
PIPELINED
PARALLEL_ENABLE (PARTITION cur BY ANY) IS
    PRAGMA AUTONOMOUS_TRANSACTION;
    prod_id                    NUMBER(6);
    prod_name                  VARCHAR2(50);
    prod_desc                  VARCHAR2(4000);
    prod_subcategory           VARCHAR2(50);
    prod_subcategory_desc      VARCHAR2(2000);
    prod_category              VARCHAR2(50);
    prod_category_desc         VARCHAR2(2000);
    prod_weight_class          NUMBER(2);
    prod_unit_of_measure       VARCHAR2(20);
    prod_pack_size             VARCHAR2(30);
    supplier_id                NUMBER(6);
    prod_status                VARCHAR2(20);
    prod_list_price            NUMBER(8,2);
    prod_min_price             NUMBER(8,2);
    sales                      NUMBER:=0;
BEGIN
   LOOP
     -- Fetch from cursor variable
     FETCH cur INTO prod_id, prod_name, prod_desc, prod_subcategory,
  prod_subcategory_desc, prod_category, prod_category_desc, prod_weight_class,
  prod_unit_of_measure, prod_pack_size, supplier_id, prod_status,
     prod_list_price, prod_min_price;
     EXIT WHEN cur%NOTFOUND; -- exit when last row is fetched
     IF prod_status='obsolete' THEN
       IF prod_category=prod_cat THEN
          INSERT INTO obsolete_products_errors VALUES
          (prod_id, 'correction: category '||UPPER(prod_cat)||' still
   available');
```

```
          COMMIT;
       ELSE
       PIPE ROW (product_t( prod_id, prod_name, prod_desc, prod_subcategory,
 prod_subcategory_desc, prod_category, prod_category_desc, prod_weight_class,
 prod_unit_of_measure, prod_pack_size, supplier_id, prod_status,
 prod_list_price, prod_min_price));
       END IF;
     END IF;
   END LOOP;
   CLOSE cur;
   RETURN;
END;
/
```

次の問合せでは、誤ってステータスobsoleteに設定されているprod_category Electronicsを除き、すべての廃止製品グループが示されます。

```
SELECT DISTINCT prod_category, prod_status FROM TABLE(obsolete_products_dml(
CURSOR(SELECT prod_id, prod_name, prod_desc, prod_subcategory,
  prod_subcategory_desc, prod_category, prod_category_desc, prod_weight_class,
  prod_unit_of_measure, prod_pack_size, supplier_id, prod_status,
  prod_list_price, prod_min_price
FROM products)));
```

このように、prod_category Electronicsには、誤って廃止された製品があります。

```
SELECT DISTINCT msg FROM obsolete_products_errors;
```

2番目の入力変数を利用すると、Electronics以外の対象となる別の製品グループを指定できます。

```
SELECT DISTINCT prod_category, prod_status
FROM TABLE(obsolete_products_dml(
CURSOR(SELECT prod_id, prod_name, prod_desc, prod_subcategory,
  prod_subcategory_desc, prod_category, prod_category_desc, prod_weight_class,
  prod_unit_of_measure, prod_pack_size, supplier_id, prod_status,
  prod_list_price, prod_min_price
FROM products),'Photo'));
```

テーブル・ファンクションは通常の表と同様に使用できるため、次のようにネストできます。

```
SELECT DISTINCT prod_category, prod_status
FROM TABLE(obsolete_products_dml(CURSOR(SELECT *
FROM TABLE(obsolete_products_pipe(CURSOR(SELECT prod_id, prod_name, prod_desc,
 prod_subcategory, prod_subcategory_desc, prod_category, prod_category_desc,
 prod_weight_class, prod_unit_of_measure, prod_pack_size, supplier_id,
 prod_status, prod_list_price, prod_min_price
FROM products))))));
```

Oracle DatabaseのETLの最大のメリットはツールキット機能です。この機能を後述の機能と組み合せると、ETL処理を改善してスピードアップできます。たとえば、入力として外部表を使用し、それを既存の表と結合し、パラレル化されたテーブル・ファンクションの入力に使用して、複雑なビジネス・ロジックを処理できます。このテーブル・ファンクションはMERGE操作の入力ソースとして使用できるため、フラット・ファイルとして提供されたデータ・ウェアハウス用の新規情報をストリーム化し、ETLプロセスを通じて単一の文中で処理できます。

18.4 エラーのロギングおよび処理のメカニズム

外部ソースなど、特に様々なソースからのデータを処理する場合、データをロードおよび変換する際に、制約に従っていないデー

タに遭遇する場合がよくあります。この不適切なデータによって、長時間のロードや変換の操作が中断させられると、多くの時間とリソースが無駄になります。

次のトピックでは、エラーの2つの主な原因とその対処方法について説明します。

- ビジネス・ルールの違反
- データ・ルールの違反(データ・エラー)

18.4.1 ビジネス・ルールの違反

論理的に制約に従っていないデータは、データを使用する前に認識されているビジネス・ルールに違反します。ほとんどの場合、この種のエラー処理は、ロードまたは変換プロセスに組み込まれます。ただし、すべてのレコードに対するエラー識別にかかるコストが膨大となり、かつデータ・ルール違反であってもビジネス・ルールが適用される可能性があるような状況(数百の列をテストして、それらがNOT NULLかどうかを確認するなど)では、プログラマは多くの場合、既知の論理エラーの場合でも一般的に処理する方を選択します。この例は、データ・エラーのシナリオに示しています。

論理的なルールの組込みは、簡単な場合(データの入力ストリームに対してフィルタ条件を適用するなど)もあれば、複雑な場合(不適切なデータを異なる変換ワークフローに送り込むなど)もあります。次に、例をいくつか示します。

- SQLを使用した論理データ・エラーのフィルタ処理。特定の条件を満たさないデータを、処理する前にフィルタにかけて除外します。
- 論理データ・エラーの識別と選別。これは、複雑なケースでは、例18-6に示すようにプロシージャによるアプローチで、単純なケースでは、例18-1に示すようにSQLを使用して行うことができます。

18.4.2 データ・ルールの違反(データ・エラー)

論理エラーと異なり、データ・ルールの違反は、ロード・プロセスや変換プロセスでは通常想定していません。操作では処理されないこのような想定外のデータ・ルールの違反(データ・エラー)によって、処理は失敗します。データ・ルールの違反は、データベース内で発生し、文を失敗させるエラー状況です。この例として、データ型の変換エラーや制約違反があります。

かつてのSQLでは、大量処理の一部として行レベルでデータ・エラーを処理する方法がありませんでした。データベース内のデータ・エラーを処理する唯一の手段は、PL/SQLを使用することでした。現在では、DML操作の続行中でもデータ・エラーのログを特別なエラー表に記録できるようになりました。SQL関数を使用してデータ変換エラーを処理できるようにもなりました。

次の項では、様々な例外処理戦略について簡単に説明します。

- SQLを使用したデータ・エラーの処理
- PL/SQLを使用したデータ・エラーの処理
- エラー・ロギング表を使用したデータ・エラーの処理

18.4.2.1 SQLを使用したデータ・エラーの処理

データ変換プロセスで使用される外部データは場合によって適切でなく、その結果、データ変換エラーが発生することがあります。特定のSQL関数を使用して、データ変換エラーを処理できます。

データ変換エラーを処理するSQL関数を使用するには、COMPATIBLEパラメータを12.2に設定する必要があります。

SQL関数を使用したデータ変換エラーの処理には、次の方法を使用できます。

- 有効または無効なデータの明示的なフィルタ処理

  VALIDATE_CONVERSION関数は、目的のデータ型に変換できない、問題のあるデータを特定します。所定の式を指定されたデータ型に変換できる場合は1を返し、それ以外の場合は0を返します。

- SQLデータ型変換関数内でのエラー処理

  CAST、TO_NUMBER、TO_BINARY_FLOAT、TO_BINARY_DOUBLE、TO_DATE、TO_TIMESTAMP、TO_TIMESTAMP_TZ、TO_DSINTERVALおよびTO_YMINTERVAL関数は、データ型変換エラーが発生した場合、エラーのかわりにユーザー指定の値を返すことができます。これにより、ETLプロセス中の失敗が減少します。

  ユーザー指定の値が返されるのは、式を評価するときではなく、式の変換中にエラーが発生した場合のみです。CAST関数では、特定のデータ型の引数として書式文字列とNLSパラメータ文字列を指定することもできます。

例18-9 VALIDATE_CONVERSIONおよびCASTを使用したデータ変換エラーの処理

TMP_PRODUCTS表からPRODUCTS表にデータをロードするとします。列の数と名前はどちらの表でも同じですが、prod_id列のデータ型が異なります。PRODUCTS表のprod_id列のデータ型はNUMBERです。TMP_PRODUCTS表のprod_id列のデータは数値ですが、データ型はVARCHAR2です。PRODUCTS表へのデータのロード中に、prod_id列のデータ型変換エラーを処理するには、適切でないprod_id値を含む行をフィルタ処理して除外するか、NUMBERに変換できないprod_id値にデフォルト値を割り当てます。

次のコマンドは、TMP_PRODUCTS表からPRODUCTS表にデータをロードします。tmp_products.prod_idを数値に正常に変換できる行のみが挿入されます。

```
INSERT INTO PRODUCTS
   (SELECT prod_id, prod_name, prod_desc, prod_category_id, prod_category_name,
      prod_category_desc, prod_list_price
    FROM tmp_products
    WHERE VALIDATE_CONVERSION(prod_id AS NUMBER)=1);
```

CAST関数を使用すると、データ型変換エラーの原因となるprod_id値を処理できます。次のINSERTコマンドは、TMP_PRODUCTS表からPRODUCTS表にデータをロードします。prod_idで使用されているCAST関数によって、データ型変換エラーが発生した場合にデフォルト値0がprod_idに割り当てられます。これにより、データ型変換エラーが原因でロード操作が失敗することがなくなります。

```
INSERT INTO PRODUCTS
   (SELECT CAST(prod_id AS NUMBER DEFAULT 0 ON CONVERSION ERROR), prod_name,
      prod_desc, prod_category_id, prod_category_name, prod_category_desc,
      prod_list_price
    FROM tmp_products);
```

関連項目:

CAST関数およびVALIDATE_CONVERSION関数、およびそれらでサポートされるデータ型の詳細は、Oracle Database SQL言語リファレンスを参照してください。

18.4.2.2 PL/SQLを使用したデータ・エラーの処理

次の文は、PL/SQLを使用してエラー処理を行う方法の例を示しています。ここでは、すべてのエラーを捕捉するために、プロシージャによるレコードレベルの処理を使用する必要があります。この文は、エラー・ロギング表を使用したデータ・エラーの処理で説明している文とほぼ同じです。

```
DECLARE
errm number default 0;
BEGIN
FOR crec IN (SELECT product_id, customer_id, TRUNC(sales_date) sd,
                    promotion_id, quantity, amount
             FROM sales_activity_direct) loop
```

```
BEGIN
 INSERT INTO sales VALUES (crec.product_id, crec.customer_id,
                           crec.sd, 3, crec.promotion_id,
                           crec.quantity, crec.amount);
exception
WHEN others then
 errm := sqlerrm;
 INSERT INTO sales_activity_error
        VALUES (errm, crec.product_id, crec.customer_id, crec.sd,
                crec.promotion_id, crec.quantity, crec.amount);
END;
END loop;
END;
/
```

18.4.2.3 エラー・ロギング表を使用したデータ・エラーの処理

DMLエラー・ロギングによって既存のDML機能が拡張され、ユーザーがエラー・ロギング表の名前を指定すると、DML操作中に検出されたエラーがOracle Databaseによってその表に記録されるようになりました。これにより、どのようなエラーが検出されてもDML操作を完了でき、後でエラーのある行において対処措置をとることができます。

このDMLエラー・ロギング表は、ターゲット列に起こり得るエラーを格納可能なデータ型を使用した、DML操作のターゲット表の列のすべてまたはサブセットを表すユーザー定義の列のセット、およびいくつかの必須の制御列から構成されています。たとえば、ターゲット表のNUMBER列のTO_NUMデータ型変換エラーを格納するには、エラー・ロギング表にVARCHAR2データ型が必要となります。DMLエラー・ロギング表を作成するには、DBMS_ERRLOGパッケージを使用する必要があります。このパッケージおよびロギング表の構造の詳細は、『*Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス*』を参照してください。

DMLターゲット表とエラー・ロギング表の間の列名のマッピングによって、制御列以外のどの列がDML操作でロギングされるかが決まります。

次の文は、DMLエラー・ロギングによってSQLを使用したデータの変換の例を強化する方法を示します。

```
INSERT /*+ APPEND PARALLEL */
INTO sales SELECT product_id, customer_id, TRUNC(sales_date), 3,
   promotion_id, quantity, amount
FROM sales_activity_direct
LOG ERRORS INTO sales_activity_errors('load_20040802')
REJECT LIMIT UNLIMITED
```

すべてのデータ・エラーのログが、オプションのタグload_20040802で識別される表sales_activity_errorsに記録されます。このINSERT文は、データ・エラーが存在する場合でも成功します。この文を使用する前にDMLエラー・ロギング表を作成しておく必要がある点に注意してください。

REJECT LIMIT Xを指定していた場合、この文は失敗し、エラー・メッセージ「エラーX=1」が示されます。このエラー・メッセージは、拒否制限(REJECT LIMIT)によって異なる場合があります。文が失敗した場合、DML文のみがロールバックされ、DMLエラー・ロギング表への挿入操作はロールバックされません。エラー・ロギング表には、X+1個の行が含まれます。

DMLエラー・ロギング表は、実行ユーザーとは異なるスキーマに配置できますが、その場合、表の完全な名前を指定する必要があります。状況に応じてDMLエラー・ロギング表の名前を省略することもできます。その場合、DBMS_ERRLOGパッケージによって生成される表のデフォルト名が使用されます。

Oracle Databaseは、DML操作中に次のエラーを記録します。

- 大きすぎる列値
- 制約違反(NOT NULL制約、一意制約、参照制約およびチェック制約)

- トリガー実行中に発生したエラー
- 副問合せの列と表の対応列との間の型変換で発生したエラー
- パーティション・マッピング・エラー

次の状態では、文の実行が失敗しロールバックは実行されますが、エラー・ロギング機能は起動されません。

- 遅延制約の違反
- 領域不足エラー
- 一意制約や索引の違反を招くすべてのINSERT操作(INSERTまたはMERGE)
- 一意制約や索引の違反を招くすべてのUPDATE操作(UPDATEまたはMERGE)

また、LONG、LOBまたはオブジェクト型の列については、エラー・ロギング表においてエラーを追跡できません。エラー・ロギングを使用する際の制約の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

DMLエラー・ロギングは、あらゆる種類のDML操作に適用できます。次の項では、これらの例について説明します。

外部ロード・ユーティリティとしてのSQL*Loaderにも、データ・エラーのロギング機能がありますが、データベース内部に統合されたETL処理のメリットがありません。

18.5 ロードおよび変換の使用例

ここでは、典型的なロードおよび変換タスクの例を示します。

- キー参照のシナリオ
- ビジネス・ルール違反のシナリオ
- データ・エラーのシナリオ
- ピボットのシナリオ

18.5.1 キー参照のシナリオ

典型的な変換として、キー参照があります。たとえば、売上トランザクション・データが小売データ・ウェアハウスにロードされているとします。データ・ウェアハウスのsales表にはproduct_id列がありますが、ソース・システムから抽出された売上トランザクション・データには、製品IDではなくUPCコード(Uniform Price Codes)があります。そのため、新しい売上トランザクション・データをsales表に挿入できるようにするには、最初にUPCコードを製品IDに変換する必要があります。

この変換を実行するには、product_id値をUPCコードに関係付ける参照表が必要です。この表は、productディメンション表か、またはこの変換をサポートするために特別に作成された、データ・ウェアハウスにある別の表です。この例では、product_id列およびupc_code列を持つ、productという表があると想定しています。

このデータ置換による変換は、次のCTAS文を使用して実装できます。

```
CREATE TABLE temp_sales_step2 NOLOGGING PARALLEL AS SELECT sales_transaction_id,
  product.product_id sales_product_id, sales_customer_id, sales_time_id,
  sales_channel_id, sales_quantity_sold, sales_dollar_amount
FROM  temp_sales_step1, product
WHERE temp_sales_step1.upc_code = product.upc_code;
```

このCTAS文は、有効な各UPCコードを、有効なproduct_id値に変換します。各UPCコードが有効であることをETLプロセスが保証している場合は、この文のみで変換全体を実装できます。

18.5.2 ビジネス・ルール違反のシナリオ

前述の例で、有効なUPCコードが付いていない新規の売上データ(論理データ・エラー)も処理する必要がある場合は、次のようにCTAS文を追加使用して無効な行を識別できます。

```
CREATE TABLE temp_sales_step1_invalid NOLOGGING PARALLEL AS
SELECT * FROM temp_sales_step1 s
WHERE NOT EXISTS (SELECT 1 FROM product p WHERE p.upc_code=s.upc_code);
```

無効なデータは別の表temp_sales_step1_invalidに格納され、ETLプロセスで別々に処理できます。

無効なデータを処理する別の方法として、次の文に示すように、元のCTASを変更して外部結合を使用します。

```
CREATE TABLE temp_sales_step2 NOLOGGING PARALLEL AS
SELECT sales_transaction_id, product.product_id sales_product_id,
   sales_customer_id, sales_time_id, sales_channel_id, sales_quantity_sold,
   sales_dollar_amount
FROM  temp_sales_step1, product
WHERE temp_sales_step1.upc_code = product.upc_code (+);
```

外部結合を使用すると、無効なUPCコードを含んでいた売上トランザクションに、NULLのproduct_idが割り当てられます。これらのトランザクションは後で処理できます。または、NULLのproduct_idで値を別の表に分離するマルチテーブル・インサートも使用できます。これは、予想されたエラー数がデータ全体の量と比較して相対的に少ないときに有効なアプローチとなる場合があります。後続の処理では、大きなターゲット表を扱う必要がなくなり、小さな表のみを扱うことになります。

```
INSERT /*+ APPEND PARALLEL */ FIRST
WHEN sales_product_id IS NOT NULL THEN
   INTO temp_sales_step2
   VALUES (sales_transaction_id, sales_product_id,
   sales_customer_id, sales_time_id, sales_channel_id,
   sales_quantity_sold, sales_dollar_amount)
ELSE
   INTO temp_sales_step1_invalid
   VALUES (sales_transaction_id, sales_product_id,
   sales_customer_id, sales_time_id, sales_channel_id,
   sales_quantity_sold, sales_dollar_amount)
SELECT sales_transaction_id, product.product_id sales_product_id,
   sales_customer_id, sales_time_id, sales_channel_id,
   sales_quantity_sold, sales_dollar_amount
FROM temp_sales_step1, product
WHERE temp_sales_step1.upc_code = product.upc_code (+);
```

このソリューションでは、空の表temp_sales_step2およびtemp_sales_step1_invalidがあらかじめ存在している必要がある点に注意してください。

無効なUPCコードを処理するには、他にも方法があります。あるデータ・ウェアハウスではNULL値のproduct_idをsales表に挿入するように選択されます。また、無効なUPCコードがすべて処理されるまで、バッチ全体のどの新規データもsales表に挿入できないデータ・ウェアハウスもあります。どの方法が適切かは、データ・ウェアハウスのビジネス要件によって決まります。特定の要件に関係なく、例外処理は、変換と同じく基本的なSQLによって処理されます。

18.5.3 データ・エラーのシナリオ

データの質が不明の場合、ビジネス・ルール違反のシナリオで示した例を、次に示すように想定外のデータ・エラー(データ型の変換エラーなど)を処理するように強化できます。

```
INSERT /*+ APPEND PARALLEL */ FIRST
WHEN sales_product_id IS NOT NULL THEN
```

```
INTO temp_sales_step2
VALUES (sales_transaction_id, sales_product_id,
   sales_customer_id, sales_time_id, sales_channel_id,
   sales_quantity_sold, sales_dollar_amount)
LOG ERRORS INTO sales_step2_errors('load_20040804')
REJECT LIMIT UNLIMITED
ELSE
INTO temp_sales_step1_invalid
VALUES (sales_transaction_id, sales_product_id,
   sales_customer_id, sales_time_id, sales_channel_id,
   sales_quantity_sold, sales_dollar_amount)
LOG ERRORS INTO sales_step2_errors('load_20040804')
REJECT LIMIT UNLIMITED
SELECT sales_transaction_id, product.product_id sales_product_id,
   sales_customer_id, sales_time_id, sales_channel_id,
   sales_quantity_sold, sales_dollar_amount
FROM temp_sales_step1, product
WHERE temp_sales_step1.upc_code = product.upc_code (+);
```

この文は、表temp_sales_step1_invalidに有効な製品UPCコードが存在しないという論理データ・エラー、およびsales_step2_errorsという名前のDMLエラー・ロギング表において可能性のあるその他のすべてのエラーを追跡します。エラー・ロギング表は、複数のDML操作で使用できます。

または、NOT NULL制約を使用して、データベース・レベルで有効なUPCコードが存在するというビジネス・ルールを実施する方法もあります。外部結合を使用すると、有効なUPCコードを持たないすべての注文はNULL値にマップされ、データ・エラーとして扱われます。次の文では、これらのエラーを追跡するために、このDMLエラー・ロギング機能が使用されています。

```
INSERT /*+ APPEND PARALLEL */
INTO temp_sales_step2
VALUES (sales_transaction_id, sales_product_id,
   sales_customer_id, sales_time_id, sales_channel_id,
   sales_quantity_sold, sales_dollar_amount)
SELECT sales_transaction_id, product.product_id sales_product_id,
   sales_customer_id, sales_time_id, sales_channel_id,
   sales_quantity_sold, sales_dollar_amount
FROM temp_sales_step1, product
WHERE temp_sales_step1.upc_code = product.upc_code (+)
LOG ERRORS INTO sales_step2_errors('load_20040804')
REJECT LIMIT UNLIMITED;
```

エラー・ロギング表には、DML操作の失敗の原因となったすべてのレコードが格納されます。この内容を利用して、あらゆるエラーを分析および修正できます。エラー・ロギング表の内容は、DML操作自体が成功したかどうかに関わりなく、すべてのDML操作に対して保持されます。拒否制限(REJECT LIMIT)に達したため、次のSQL文が失敗したものと仮定します。

```
SQL> INSERT /*+ APPEND NOLOGGING PARALLEL */ INTO sales_overall
2 SELECT * FROM sales_activity_direct
3 LOG ERRORS INTO err$_sales_overall ('load_test2')
4 REJECT LIMIT 10;
SELECT * FROM sales_activity_direct
*
ERROR at line 2:
ORA-01722: invalid number
```

エラー・ロギング表の名前err$_sales_overallは、DBMS_ERRLOGパッケージを使用して導出されたデフォルトの名前です。詳細は、『*Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス*』を参照してください。

生成されたエラー・メッセージは、最初のエラー制限に達した後で発生しています。次のエラー(番号11)は、エラーを生じさせたものです。表示されているエラー・メッセージは、制限を超過したエラーを基準としているため、たとえば、9番目のエラーは、11番

目のエラーとは異なる場合があります。

ターゲット表sales_overallには入力されているレコードがありません(空の表だったと考えられる)が、エラー・ロギング表には、11個(REJECT LIMIT + 1)の行が含まれます。

```
SQL> SELECT COUNT(*) FROM sales_overall;
COUNT(*)
----------
0

SQL> SELECT COUNT(*) FROM err$_sales_overall;
COUNT(*)
----------
11
```

DMLエラー・ロギング表は、すべてのエラー・ロギング表に必須であるいくつかの固定の制御列で構成されます。Oracleエラー番号の他に、エラー・メッセージも格納されます。多くの場合、エラー・メッセージにより、データ・エラーの根本的原因を分析および解決するための詳細な情報が示されます。次のDMLエラー・ロギング表のSQL出力は、この違いを示しています。2番目の出力には、NOT NULL違反によって拒否された行の追加情報が含まれています。

```
SQL> SELECT DISTINCT ora_err_number$ FROM err$_sales_overall;

ORA_ERR_NUMBER$
---------------
           1400
           1722
           1830
           1847

SQL> SELECT DISTINCT ora_err_number$, ora_err_mesg$ FROM err$_sales_overall;

ORA_ERR_NUMBER$       ORA_ERR_MESG$
           1400       ORA-01400: cannot insert NULL into
                      ("SH"."SALES_OVERALL"."CUST_ID")
           1400       ORA-01400: cannot insert NULL into
                      ("SH"."SALES_OVERALL"."PROD_ID")
           1722       ORA-01722: invalid number
           1830       ORA-01830: date format picture ends before
                      converting entire input string
           1847       ORA-01847: day of month must be between 1 and last
                      day of month
```

関連項目:

制御列の詳細は、『*Oracle Database*管理者ガイド』を参照してください。

18.5.4 ピボットのシナリオ

データ・ウェアハウスは、多数の異なるソースからデータを受け取ることができます。これらのソース・システムには、リレーショナル・データベースではないものもあり、データ・ウェアハウスとは大きく異なるフォーマットでデータが格納されている場合があります。たとえば、売上レコードの集合を、次のフォームの非リレーショナル・データベースから受け取ったとします。

```
product_id, customer_id, weekly_start_date, sales_sun, sales_mon, sales_tue,
  sales_wed, sales_thu, sales_fri, sales_sat
```

入力表は次のようになります。

```
SELECT * FROM sales_input_table;

PRODUCT_ID CUSTOMER_ID WEEKLY_ST  SALES_SUN  SALES_MON  SALES_TUE  SALES_WED SALES_THU  SALES_FRI
SALES_SAT
---------- ----------- --------- ---------- ---------- ---------- ---------- ---------- ----
------
       111         222 01-OCT-00        100        200        300        400        500        600
700
       222         333 08-OCT-00        200        300        400        500        600        700
800
       333         444 15-OCT-00        300        400        500        600        700        800
900
```

データ・ウェアハウスでは、次のような一般的なリレーショナル形式でshサンプル・スキーマのファクト表salesにこれらのレコードを格納します。

```
prod_id, cust_id, time_id, amount_sold
```

ノート:

この例では、簡潔にするために表の多数の列を無視しているため、sales 表の多数の制約が使用禁止になっています。

これには、入力ストリームの各レコードが、データ・ウェアハウスのsales表の7つのレコードに変換されるように変換処理を作成する必要があります。通常、この操作はピボットと呼ばれ、Oracle Databaseでは複数の方法でこの操作を行えます。

前述の例の結果は、次のようになります。

```
SELECT prod_id, cust_id, time_id, amount_sold FROM sales;

   PROD_ID    CUST_ID   TIME_ID   AMOUNT_SOLD
---------- ----------   --------- -----------
       111         222   01-OCT-00         100
       111         222   02-OCT-00         200
       111         222   03-OCT-00         300
       111         222   04-OCT-00         400
       111         222   05-OCT-00         500
       111         222   06-OCT-00         600
       111         222   07-OCT-00         700
       222         333   08-OCT-00         200
       222         333   09-OCT-00         300
       222         333   10-OCT-00         400
       222         333   11-OCT-00         500
       222         333   12-OCT-00         600
       222         333   13-OCT-00         700
       222         333   14-OCT-00         800
       333         444   15-OCT-00         300
       333         444   16-OCT-00         400
       333         444   17-OCT-00         500
       333         444   18-OCT-00         600
       333         444   19-OCT-00         700
       333         444   20-OCT-00         800
       333         444   21-OCT-00         900
```

例18-10 ピボットの例

次の例では、マルチテーブル・インサート構文を使用して、デモ表sh.salesに異なる構造を持つ入力表からデータを挿入しています。マルチテーブルINSERT文は次のようになります。

```
INSERT ALL INTO sales (prod_id, cust_id, time_id, amount_sold)
      VALUES (product_id, customer_id, weekly_start_date, sales_sun)
      INTO sales (prod_id, cust_id, time_id, amount_sold)
      VALUES (product_id, customer_id, weekly_start_date+1, sales_mon)
      INTO sales (prod_id, cust_id, time_id, amount_sold)
      VALUES (product_id, customer_id, weekly_start_date+2, sales_tue)
      INTO sales (prod_id, cust_id, time_id, amount_sold)
      VALUES (product_id, customer_id, weekly_start_date+3, sales_wed)
      INTO sales (prod_id, cust_id, time_id, amount_sold)
      VALUES (product_id, customer_id, weekly_start_date+4, sales_thu)
      INTO sales (prod_id, cust_id, time_id, amount_sold)
      VALUES (product_id, customer_id, weekly_start_date+5, sales_fri)
      INTO sales (prod_id, cust_id, time_id, amount_sold)
      VALUES (product_id, customer_id, weekly_start_date+6, sales_sat)
SELECT product_id, customer_id, weekly_start_date, sales_sun,
      sales_mon, sales_tue, sales_wed, sales_thu, sales_fri, sales_sat
FROM sales_input_table;
```

この文では、ソース表が1回のみスキャンされ、毎日の適切なデータが挿入されます。

関連項目:

- ピボットの詳細は、ピボット操作を参照してください。
- pivot_clauseの構文については、『Oracle Database SQL言語リファレンス』を参照してください。

第IV部 リレーショナル分析

この部では、データ・ウェアハウスのパフォーマンスを向上させる方法について説明します。次の章で構成されています。

- 分析計算およびレポート用SQL関数
- データ・ウェアハウスにおける集計のためのSQL
- パターン一致用SQL
- モデリングのSQL
- 高度な分析SQL

19 分析計算およびレポート用SQL関数

次のトピックでは、Oracleにおける分析SQLの機能および手法について説明します。これらのトピックは、データ・ウェアハウスの観点から表されていますが、分析およびレポートを必要とするすべてのアクティビティに適用可能です。

- 分析計算およびレポート用SQL関数の概要
- ランキング関数、ウィンドウ関数およびレポート関数
- 分析用の高度な集計
- ピボット操作
- レポート用のデータの稠密化
- 稠密化したデータに対する時系列の計算
- その他の分析およびレポートの機能
- SQLの行の制限

19.1 分析計算およびレポート用SQL関数の概要

Oracle Databaseでは、大規模な分析SQL関数のファミリを用意しています。このような分析用の関数を使用すると、次の計算が可能です。

- ランキングとパーセンタイル(百分位数)
- 変動ウィンドウの計算
- LAG/LEAD分析
- FIRST/LAST分析
- 線形回帰統計

ランキング関数には、累積分散、パーセント・ランクおよびNタイルなどがあります。変動ウィンドウの計算では、合計や平均などの変動集計および累積集計の検索が可能になります。LAG/LEAD分析では、行間の直接参照が可能になるため、周期ごとの変更を計算できます。FIRST/LAST分析では、順序付けされたグループ内の最初の値または最後の値を検索できます。

分析およびレポートに役立つその他のSQL要素には、CASE式およびパーティション外部結合があります。CASE式では、様々な状況で有効なif-thenロジックが提供されます。パーティション外部結合は、特定のディメンションを選択的に稠密化しながら、その他のディメンションをスパースなままにできるANSI外部結合構文のバリアントです。これにより、たとえば、その他のディメンションはスパースなままで、クロス集計レポートに表示されるディメンションをレポート・ツールで選択的に稠密化できます。

パフォーマンスを向上させるには、分析関数をパラレル化する方法があります。つまり、複数のプロセスですべての文を同時に実行できます。こうした機能によって計算がより容易かつ効率的になるので、データベースのパフォーマンス、スケーラビリティおよび簡易性が向上します。

分析関数は、表19-1に示すように分類されます。

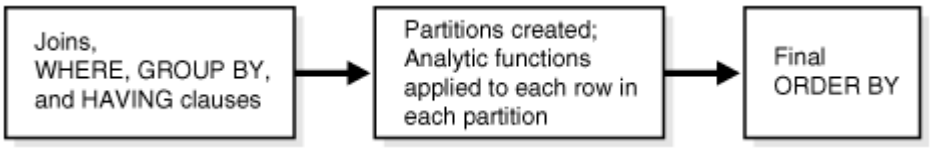
表19-1 **分析関数およびその使用目的**

タイプ	使用目的
ランキング	結果セットのランク、パーセンタイルおよび n タイルの値を計算する。

タイプ	使用目的
ウィンドウ	累積集計および変動集計を計算する。SUM、AVG、MIN、MAX、COUNT、VARIANCE、STDDEV、FIRST_VALUE、LAST_VALUE および新しい統計関数とともに動作する。DISTINCT キーワードは MAX および MIN を除くウィンドウ関数ではサポートされません。
レポート	市場占有率などのシェアを計算する。SUM、AVG、MIN、MAX、COUNT(DISTINCT 付き/なし)、VARIANCE、STDDEV、RATIO_TO_REPORT および新しい統計関数とともに動作する。集計モードで DISTINCT キーワードがサポートされるレポート関数では、DISTINCT を使用できる。
LAG/LEAD	現在行から指定した行数を移動した行の値を検索する。
FIRST/LAST	順序付けされたグループ内の最初または最後の値。
線形回帰	線形回帰およびその他の統計情報(傾き、切片など)を計算する。
逆パーセンタイル	データセット内で指定されたパーセンタイルと一致する値。
仮説ランクおよび仮説分布	行が指定されたデータセットに挿入された場合に与えられるランクまたはパーセンタイル。

こうした処理を行うため、分析関数ではSQL処理に新しい要素がいくつか追加されています。これらの要素は、既存のSQL上に作成され、柔軟で強力な計算式を可能にします。分析関数には、いくつかの例外を除きこの追加要素群が含まれます。図19-1に、処理フローを表します。

図19-1 処理順序



次に、分析関数における重要な概念を示します。

- 処理順序

  分析関数を使用した問合せ処理は、3つのステップで実行されます。第1に、すべての結合、WHERE、GROUP BYおよびHAVING句が実行されます。第2に、結果セットを分析関数で使用できるようになり、その計算がすべて実行されます。第3に、問合せの最後にORDER BY句がある場合、ORDER BYが処理され、正確な出力順序付けが可能になります。処理順序は、図19-1のとおりです。

- 結果セット・パーティション

  分析関数を使用すると、問合せの結果セットをパーティションと呼ばれる行グループに分割できます。分析関数で使用される**パーティション**という用語は、表パーティション機能とは無関係です。この章でパーティションという用語が使用される場合、分析関数に関する意味のみを示します。パーティションはGROUP BY句で定義されているグループの後に作成されるため、SUMやAVGなどのすべての集計結果で使用できます。パーティションは、必要な列または式に基づいて分割することもできます。問合せ結果セットは、すべての行を持つ1つのパーティション、少数の大きなパーティション、またはそれぞれが数行しか持たない多数の小さなパーティションに分割可能です。

- ウィンドウ

  パーティションの各行に対して、スライドするデータ・ウィンドウを定義できます。このウィンドウで、カレント行の計算に使用される行の範囲が決まります。ウィンドウの大きさは、行の物理数値または時間などのロジカル・インターバルに基づきます。ウィンドウには、開始行および終了行があります。ウィンドウの定義によっては、ウィンドウの片方または両方の末端を移動できます。たとえば、累積合計関数として定義されているウィンドウでは、開始行がパーティションの最初の行に固定されますが、終了行はパーティションの開始点から最後の行までスライドします。一方、移動平均として定義されているウィンドウでは、開始点および終了点の両方がスライドし、一定の物理範囲または論理範囲が維持されます。

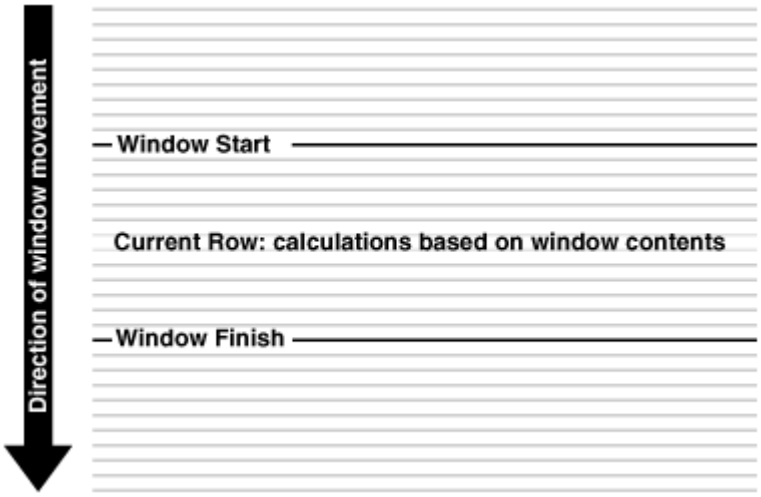
  ウィンドウのサイズは、パーティション内の行全体と同じ大きさからパーティション内の1行のスライド・ウィンドウとしてまで様々に設定できます。ウィンドウがパーティションの境界に近い場合、関数からは使用可能な行の結果のみが戻されます。結果がユーザーの必要とするものではないことは警告されません。

  ウィンドウ関数を使用するとカレント行が計算に含まれるため、*n*個の項目を処理する場合は(*n*-1)を指定します。

- カレント行

  分析関数を使用して実行される各計算は、パーティション内のカレント行に基づいて行われます。カレント行は、ウィンドウの開始および終了を判断する際の参照点として機能します。たとえば、中央の移動平均計算は、カレント行とその前の6行およびその後の6行にわたるウィンドウのように定義できます。この場合、図19-2のように、13行の大きさのスライド・ウィンドウが作成されます。

図19-2 スライド・ウィンドウの例



19.2 ランキング関数、ウィンドウ関数およびレポート関数

この項では、ランキング、ウィンドウおよびレポート用の基本的な分析関数について説明します。次の項目が含まれます。

- ランキング関数
- ウィンドウ関数
- レポート関数
- LAG/LEAD関数
- FIRST_VALUE関数、LAST_VALUE関数およびNTH_VALUE関数

19.2.1 ランキング関数

ランキング関数では、メジャーの集合の値に基づいたデータセット内の他のレコードとの比較により、レコードのランクが計算されます。ランキング関数の種類を次に示します。

- RANK関数およびDENSE_RANK関数
- ボトムNランキング関数
- CUME_DIST関数
- PERCENT_RANK関数
- NTILE関数
- ROW_NUMBER関数

19.2.1.1 RANK関数およびDENSE_RANK関数

RANK関数およびDENSE_RANK関数では、グループ内での項目のランク付けができます。たとえば、昨年カリフォルニアでよく売れた製品の上位3位までを検索する場合などに利用できます。次の構文で示すように、ランキングを実行する関数には2種類あります。

```
RANK ( ) OVER ( [query_partition_clause] order_by_clause )
DENSE_RANK ( ) OVER ( [query_partition_clause] order_by_clause )
```

RANKとDENSE_RANKの違いは、同じ値の項目がある場合、DENSE_RANKではランキングの順序に抜けができないという点です。つまり、ある競争についてDENSE_RANKを使用してランキングした結果、3人が同点で第2位であった場合、3人全員が第2位となり、次点の人が第3位になります。RANK関数でも3人全員が第2位になりますが、次点の人は第5位になります。

次に、RANKに関する注意点を示します。

- デフォルトのソート順は昇順です。必要に応じて降順に変更できます。
- オプションのPARTITION BY句の式を使用すると、問合せ結果セットがRANK関数の適用範囲となるグループ群に分割されます。つまり、グループが変更されるたびに、RANKがリセットされます。実際には、PARTITION BY句の値式でリセットの境界が定義されます。
- PARTITION BY句がない場合、ランクは問合せ結果セット全体にわたって計算されます。
- ORDER> BY句によって、ランキングが実行されるメジャー(<value expression>)が指定され、各グループ(またはパーティション)でソートされる行の順序が定義されます。各パーティション内でデータがソートされると、各行が1からランク付けされます。
- NULLS FIRST | NULLS LAST句によって、順序付けされた行セットでNULLの位置が最初になるか最後になるかが示されます。順序付けによって、NULLが、NULL以外の値より高いか低いかが比較されます。順序が昇順であった場合、NULLS FIRSTはNULLが他のどのNULL以外の値よりも小さいことを示し、NULLS LASTはNULL以外の値よりも大きいことを示します。降順では、その逆になります。例: ランキング関数でのNULLの処理の例を参照してください。
- NULLS FIRST | NULLS LAST句が省略されている場合、NULL値の順序付けはASC引数またはDESC引数に依存します。NULL値は、他のどの値よりも大きいとみなされます。順序付け順序がASCの場合、NULLは最後に表示されます。それ以外の場合は、最初に表示されます。NULLは他のNULLと同等とみなされるため、NULLが表示されている順序は確定的ではありません。

19.2.1.1.1 RANK関数およびDENSE_RANK関数でのランキング順序

次の例では、RANKの[ASC | DESC]オプションによるランキング順序の変化を示します。

例19-1 ランキング順序

```
SELECT channel_desc, TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$,
   RANK() OVER (ORDER BY SUM(amount_sold)) AS default_rank,
   RANK() OVER (ORDER BY SUM(amount_sold) DESC NULLS LAST) AS custom_rank
FROM sales, products, customers, times, channels, countries
```

```
WHERE sales.prod_id=products.prod_id AND sales.cust_id=customers.cust_id
  AND customers.country_id = countries.country_id AND sales.time_id=times.time_id
  AND sales.channel_id=channels.channel_id
  AND times.calendar_month_desc IN ('2000-09', '2000-10')
  AND country_iso_code='US'
GROUP BY channel_desc;

CHANNEL_DESC         SALES$          DEFAULT_RANK CUSTOM_RANK
-------------------- --------------- ------------ -----------
Direct Sales              1,320,497            3           1
Partners                    800,871            2           2
Internet                    261,278            1           3
```

この結果のデータはメジャーSALES$で順序付けされていますが、一般にRANK関数では、データがメジャーでソートされるという保証はありません。結果のデータがSALES$でソートされるようにするには、SELECT文の最後にORDER BY句で明示的にソートを指定する必要があります。

19.2.1.1.2 複数の式でのランキング

ランキング関数では、セット内にある同じ内容の値について解決する必要があります。最初の式で同じ内容の値が解決されない場合、2番目の式が同じ内容の値の解決に使用され、以降同様に続きます。たとえば、売上高(ドル)に基づいて2か月間の販売チャネルのうち3つをランキングし、単位売上で同じ内容の値を解決する問合せの例を次に示します。(ここでは、この問合せ用に同じ内容の値を作成するためだけにTRUNC関数を使用しています)。

例19-2 複数の式でのランキング

```
SELECT channel_desc, calendar_month_desc, TO_CHAR(TRUNC(SUM(amount_sold),-5),
  '9,999,999,999') SALES$, TO_CHAR(SUM(quantity_sold), '9,999,999,999')
  SALES_Count, RANK() OVER (ORDER BY TRUNC(SUM(amount_sold), -5)
  DESC, SUM(quantity_sold) DESC) AS col_rank
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id AND sales.cust_id=customers.cust_id
  AND sales.time_id=times.time_id AND sales.channel_id=channels.channel_id
  AND times.calendar_month_desc IN ('2000-09', '2000-10')
  AND channels.channel_desc<>'Tele Sales'
GROUP BY channel_desc, calendar_month_desc;

CHANNEL_DESC         CALENDAR SALES$          SALES_COUNT     COL_RANK
-------------------- -------- --------------- --------------- ---------
Direct Sales         2000-10       1,200,000          12,584         1
Direct Sales         2000-09       1,200,000          11,995         2
Partners             2000-10         600,000           7,508         3
Partners             2000-09         600,000           6,165         4
Internet             2000-09         200,000           1,887         5
Internet             2000-10         200,000           1,450         6
```

sales_count列で、3組の同じ内容の値が解決されています。

この問合せの上位5つの結果のみを表示したい場合は、ORDER BY COL_RANK FETCH FIRST 5 ROWS ONLY文を追加します。詳細は、SQLの行の制限を参照してください。

19.2.1.1.3 例: RANKとDENSE_RANKの違い

RANK関数とDENSE_RANK関数の違いを例19-3に示します。

例19-3 RANKとDENSE_RANK

```
SELECT channel_desc, calendar_month_desc,
   TO_CHAR(TRUNC(SUM(amount_sold),-5), '9,999,999,999') SALES$,
      RANK() OVER (ORDER BY TRUNC(SUM(amount_sold),-5) DESC) AS RANK,
```

```
DENSE_RANK() OVER (ORDER BY TRUNC(SUM(amount_sold),-5) DESC) AS DENSE_RANK
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id
  AND sales.cust_id=customers.cust_id
  AND sales.time_id=times.time_id AND sales.channel_id=channels.channel_id
  AND times.calendar_month_desc IN ('2000-09', '2000-10')
  AND channels.channel_desc<>'Tele Sales'
GROUP BY channel_desc, calendar_month_desc;

CHANNEL_DESC         CALENDAR SALES$              RANK DENSE_RANK
-------------------- -------- -------------- --------- ----------
Direct Sales         2000-09       1,200,000         1          1
Direct Sales         2000-10       1,200,000         1          1
Partners             2000-09         600,000         3          2
Partners             2000-10         600,000         3          2
Internet             2000-09         200,000         5          3
Internet             2000-10         200,000         5          3
```

DENSE_RANKでは、最大のランク値がデータセット内の個別値の数を示すことに注意します。

19.2.1.1.4 グループ内でのランキング: 例

RANK関数は、グループ内部を処理対象として使用できます。この場合、グループが変更されるたびに、ランクがリセットされます。これを可能にするには、PARTITION BY句を使用します。PARTITION BY副次句のグループ式を使用すると、データセットが、RANKの操作対象となるグループに分割されます。たとえば、ドル単位の売上高によって各チャネル内で製品を順序付ける場合は、次の文を発行できます。

例19-4 グループごとのランキングの例1

```
SELECT channel_desc, calendar_month_desc, TO_CHAR(SUM(amount_sold),
 '9,999,999,999') SALES$, RANK() OVER (PARTITION BY channel_desc
  ORDER BY SUM(amount_sold) DESC) AS RANK_BY_CHANNEL
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id AND sales.cust_id=customers.cust_id
  AND sales.time_id=times.time_id AND sales.channel_id=channels.channel_id
  AND times.calendar_month_desc IN ('2000-08', '2000-09', '2000-10', '2000-11')
  AND channels.channel_desc IN ('Direct Sales', 'Internet')
GROUP BY channel_desc, calendar_month_desc;

CHANNEL_DESC         CALENDAR SALES$          RANK_BY_CHANNEL
-------------------- -------- -------------- ---------------
Direct Sales         2000-08       1,236,104               1
Direct Sales         2000-10       1,225,584               2
Direct Sales         2000-09       1,217,808               3
Direct Sales         2000-11       1,115,239               4
Internet             2000-11         284,742               1
Internet             2000-10         239,236               2
Internet             2000-09         228,241               3
Internet             2000-08         215,107               4

8 rows selected.
```

単一の問合せブロックに、複数のランキング関数を含めることができます。これらの各関数によって、種々のグループにデータがパーティション化(種々の境界上でリセット)されます。そうしたグループは、相互に排他的にできます。次の問合せでは、各月(rank_of_product_per_region)および各チャネル(rank_of_product_total)内において、ドル単位の売上高に基づき製品が順序付けされます。

例19-5 グループごとのランキングの例2

```
SELECT channel_desc, calendar_month_desc, TO_CHAR(SUM(amount_sold),
  '9,999,999,999') SALES$, RANK() OVER (PARTITION BY calendar_month_desc
  ORDER BY SUM(amount_sold) DESC) AS RANK_WITHIN_MONTH, RANK() OVER (PARTITION
 BY channel_desc ORDER BY SUM(amount_sold) DESC) AS RANK_WITHIN_CHANNEL
FROM sales, products, customers, times, channels, countries
WHERE sales.prod_id=products.prod_id AND sales.cust_id=customers.cust_id
  AND customers.country_id = countries.country_id AND sales.time_id=times.time_id
  AND sales.channel_id=channels.channel_id
  AND times.calendar_month_desc IN ('2000-08', '2000-09', '2000-10', '2000-11')
  AND channels.channel_desc IN ('Direct Sales', 'Internet')
GROUP BY channel_desc, calendar_month_desc;

CHANNEL_DESC  CALENDAR    SALES$     RANK_WITHIN_MONTH  RANK_WITHIN_CHANNEL
------------- --------    ---------  -----------------  -------------------
Direct Sales   2000-08    1,236,104                  1                    1
Internet       2000-08      215,107                  2                    4
Direct Sales   2000-09    1,217,808                  1                    3
Internet       2000-09      228,241                  2                    3
Direct Sales   2000-10    1,225,584                  1                    2
Internet       2000-10      239,236                  2                    2
Direct Sales   2000-11    1,115,239                  1                    4
Internet       2000-11      284,742                  2                    1
```

19.2.1.1.5 例: キューブおよびロールアップのグループごとのランキング

RANKなどの分析関数は、CUBE、ROLLUPまたはGROUPING SETS演算子によるグルーピングに基づいて再設定されます。これは、CUBE、ROLLUPおよびGROUPING SETS問合せで作成されたグループにランクを割り当てる場合に有効です。GROUPING関数の詳細は、データ・ウェアハウスにおける集計のためのSQLを参照してください。

CUBEおよびROLLUP問合せの例を次に示します。

```
SELECT  channel_desc, country_iso_code, SUM(amount_sold) SALES$,
  RANK() OVER (PARTITION BY GROUPING_ID(channel_desc, country_iso_code)
  ORDER BY SUM(amount_sold) DESC) AS RANK_PER_GROUP
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id
  AND countries.country_id = customers.country_id AND sales.channel_id = channels.channel_id
  AND channels.channel_desc IN ('Direct Sales', 'Internet') AND times.calendar_month_desc='2000-07'
  AND country_iso_code IN ('GB', 'US', 'JP')
GROUP BY cube(channel_desc, country_iso_code);

CHANNEL_DESC    CO  SALES$         RANK_PER_GROUP
--------------  --  -------------  --------------
Direct Sales      US    616539.04              1
Direct Sales      GB     83869.96              2
Internet              US        82595.71              3
Direct Sales      JP     79047.78              4
Internet              JP        7103.39               5
Internet              GB        6477.98               6
Direct Sales                 779456.78         1
Internet                         96177.08              2
                          US   699134.75              1
                          GB   90347.94               2
                          JP   86151.17               3
                                      875633.86       1
```

19.2.1.1.6 例: ランキング関数でのNULLの扱い

NULLは、通常の値と同様に処理されます。また、ランクの計算では、NULL値は別のNULL値と同等であると想定されています。

NULLは、メジャーに設定したASC | DESCオプション、およびNULLS FIRST | NULLS LAST句に従って高低にソートされ、適切にランク付けされます。次の例では、NULLが様々なケースにおいてどのようにランク付けされるかを示します。

```
SELECT  times.time_id time,  sold,
  RANK() OVER (ORDER BY (sold) DESC NULLS LAST) AS NLAST_DESC,
  RANK() OVER (ORDER BY (sold) DESC NULLS FIRST) AS NFIRST_DESC,
  RANK() OVER (ORDER BY (sold) ASC NULLS FIRST) AS NFIRST,
  RANK() OVER (ORDER BY (sold) ASC NULLS LAST) AS NLAST
FROM
  (
   SELECT time_id, SUM(sales.amount_sold)  sold
   FROM sales, products, customers, countries
   WHERE sales.prod_id=products.prod_id
    AND customers.country_id = countries.country_id
    AND sales.cust_id=customers.cust_id
    AND prod_name IN ('Envoy Ambassador', 'Mouse Pad') AND country_iso_code ='GB'
   GROUP BY  time_id)
 v, times
WHERE v.time_id (+) = times.time_id
  AND calendar_year=1999
  AND calendar_month_number=1
ORDER BY sold  DESC NULLS LAST;

TIME            SOLD NLAST_DESC NFIRST_DESC     NFIRST      NLAST
--------- ---------- ---------- ----------- ---------- ----------
25-JAN-99    3097.32          1          18         31         14
17-JAN-99    1791.77          2          19         30         13
30-JAN-99     127.69          3          20         29         12
28-JAN-99     120.34          4          21         28         11
23-JAN-99      86.12          5          22         27         10
20-JAN-99      79.07          6          23         26          9
13-JAN-99       56.1          7          24         25          8
07-JAN-99      42.97          8          25         24          7
08-JAN-99      33.81          9          26         23          6
10-JAN-99      22.76         10          27         21          4
02-JAN-99      22.76         10          27         21          4
26-JAN-99      19.84         12          29         20          3
16-JAN-99      11.27         13          30         19          2
14-JAN-99       9.52         14          31         18          1
09-JAN-99                    15           1          1         15
12-JAN-99                    15           1          1         15
31-JAN-99                    15           1          1         15
11-JAN-99                    15           1          1         15
19-JAN-99                    15           1          1         15
03-JAN-99                    15           1          1         15
15-JAN-99                    15           1          1         15
21-JAN-99                    15           1          1         15
24-JAN-99                    15           1          1         15
04-JAN-99                    15           1          1         15
06-JAN-99                    15           1          1         15
27-JAN-99                    15           1          1         15
18-JAN-99                    15           1          1         15
01-JAN-99                    15           1          1         15
22-JAN-99                    15           1          1         15
29-JAN-99                    15           1          1         15
05-JAN-99                    15           1          1         15
```

19.2.1.2 APPROX_RANK関数

APPROX_RANK関数は、値のグループにおける近似値を返します。

この関数では、オプションのPARTITION BY句の後に必須のORDER BY ... DESC句が必要です。PARTITION BYキーはGROUP BYキーのサブセットである必要があります。ORDER BY句にはAPPROX_COUNTまたはAPPROX_SUMのいずれかを含める必要があります。

APPROX_RANK関数の構文は次のとおりです。

```
SELECT expr_1[, expr_2, … expr_j], APPROX_*(expr_k) agg_1[, APPROX_*(expr_l) agg_2…]
FROM table_name
WHERE …
GROUP BY expr_1[, expr_2, …expr_j]
HAVING APPROX_RANK(PARTITION BY partition_by_clause ORDER BY APPROX_*(expr_k) DESC) <= N1
[AND APPROX_RANK(PARTITION BY partition_by_clause ORDER BY APPROX_*(expr_l) DESC) <= N2…)];
```

次の例では、部門ごとの給与合計上位10位のジョブを返します。各ジョブについて、給与合計とランキングも表示されます。

```
SELECT deptno, job, APPROX_SUM(sal), APPROX_RANK(PARTITION BY deptno ORDER BY APPROX_SUM(sal) DESC) rk
FROM emp
GROUP BY deptno, job
HAVING APPROX_RANK(PARTITION BY deptno ORDER BY APPROX_SUM(sal) DESC) <= 10;

    DEPTNO JOB       APPROX_SUM(SAL)         RK
---------- --------- --------------- ----------
        10 CLERK                1300          3
        10 MANAGER              2450          2
        10 PRESIDENT            5000          1
        20 CLERK                1900          3
        20 MANAGER              2975          2
        20 ANALYST              6000          1
        30 CLERK                 950          3
        30 MANAGER              2850          2
        30 SALESMAN             5600          1
```

次の例では、合計給与が上位2位、その職種の従業員数が上位3位に入るジョブを部門ごとに返します。

```
SELECT deptno, job, APPROX_SUM(sal), APPROX_COUNT(*)
FROM emp
GROUP BY deptno, job
HAVING APPROX_RANK(PARTITION BY deptno ORDER BY APPROX_SUM(sal) DESC) <= 2
AND APPROX_RANK(PARTITION BY deptno ORDER BY APPROX_COUNT(*) DESC) <= 3;

    DEPTNO JOB       APPROX_SUM(SAL) APPROX_COUNT(*)
---------- --------- --------------- ---------------
        10 MANAGER              2450               1
        10 PRESIDENT            5000               1
        20 MANAGER              2975               1
        20 ANALYST              6000               2
        30 MANAGER              2850               1
        30 SALESMAN             5600               4
```

次の例では、MAX_ERROR属性を使用して近似集計の精度をレポートします。

```
SELECT deptno, job, APPROX_SUM(sal) sum_sal, APPROX_SUM(sal,'MAX_ERROR') sum_sal_err
FROM emp
GROUP BY deptno, job
HAVING APPROX_RANK(PARTITION BY deptno ORDER BY APPROX_SUM(sal) DESC) <= 2;

    DEPTNO JOB          SUM_SAL SUM_SAL_ERR
```

```
---------- --------- ---------- -----------
        10 MANAGER         2450           0
        10 PRESIDENT       5000           0
        20 MANAGER         2975           0
        20 ANALYST         6000           0
        30 MANAGER         2850           0
        30 SALESMAN        5600           0
```

関連項目:

- Oracle Database SQL言語リファレンス

19.2.1.3 ボトムNランキング関数

ボトムNはトップNに似ていますが、ランク式内の順序付け順序が異なります。前述の例の場合でいうと、降順のかわりに昇順でSUM(s_amount)を順序付けできます。

19.2.1.4 CUME_DIST関数

CUME_DIST関数(統計書によっては、パーセンタイルの逆と定義されている関数)によって、値の集合に対する特定の値の相対位置が計算されます。この順序は、昇順または降順にできます。デフォルトは昇順です。CUME_DISTで戻される値の範囲は、0(ゼロ)から1です。サイズNの集合Sに含まれる値XのCUME_DISTを計算するには、次の計算式を使用します。

```
CUME_DIST(x) =  number of values in S coming before
   and including x in the specified order/ N
```

構文は次のとおりです。

```
CUME_DIST ( ) OVER ( [query_partition_clause] order_by_clause )
```

CUME_DIST関数の様々なオプションは、意味的にはRANK関数のオプションと同様です。デフォルトの順序は昇順であり、最小値が最小のCUME_DISTを取得します(他のすべての値はこの値の後に順序付けされます)。NULLは、RANK関数の場合と同様に処理されます。NULL以外の値と同様に処理されるため、分子および分母の両方が考慮されます。次の例では、各月におけるチャネル別の売上の累積分布がわかります。

```
SELECT calendar_month_desc AS MONTH, channel_desc,
     TO_CHAR(SUM(amount_sold) , '9,999,999,999') SALES$,
     CUME_DIST() OVER (PARTITION BY calendar_month_desc ORDER BY
         SUM(amount_sold) ) AS CUME_DIST_BY_CHANNEL
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id AND sales.cust_id=customers.cust_id
  AND sales.time_id=times.time_id AND sales.channel_id=channels.channel_id
  AND times.calendar_month_desc IN ('2000-09', '2000-07','2000-08')
GROUP BY calendar_month_desc, channel_desc;

MONTH    CHANNEL_DESC         SALES$         CUME_DIST_BY_CHANNEL
-------- -------------------- -------------- --------------------
2000-07  Internet                140,423           .333333333
2000-07  Partners                611,064           .666666667
2000-07  Direct Sales          1,145,275                    1
2000-08  Internet                215,107           .333333333
2000-08  Partners                661,045           .666666667
2000-08  Direct Sales          1,236,104                    1
2000-09  Internet                228,241           .333333333
2000-09  Partners                666,172           .666666667
```

```
2000-09  Direct Sales              1,217,808                    1
```

19.2.1.5 PERCENT_RANK関数

PERCENT_RANKはCUME_DISTと似ていますが、分子に行カウントではなくランク値が使用されます。したがって、値グループに対する値の相対的なパーセント・ランクが戻されます。この関数は、一般的なスプレッドシートで使用できます。ある行のPERCENT_RANKは次のように計算されます。

```
(rank of row in its partition - 1) / (number of rows in the partition - 1)
```

PERCENT_RANKでは、0(ゼロ)から1の範囲の値が戻されます。ランク1の行は、PERCENT_RANKが0(ゼロ)になります。構文は次のとおりです。

```
PERCENT_RANK () OVER ([query_partition_clause] order_by_clause)
```

19.2.1.6 NTILE関数

NTILEを使用すると、三分位数、四分位数、十分位数およびその他の一般的な集計統計情報を簡単に計算できます。この関数では、順序付けられたパーティションが**バケット**と呼ばれる特定数のグループに分割され、バケット番号がパーティションの各行に割り当てられます。NTILE計算は、データセットを4分割、3分割およびその他のグループ数に分割できるため、非常に便利です。

各バケットは、それぞれに同数の行が割り当てられるか、他のバケットとの差が最大でも1行となるように計算されます。たとえば、パーティションに100の行があって、バケットが4つになるようにNTILE関数に要求した場合、最初の25行に1の値、次の25行に2の値が割り当てられ、残りも同様に割り当てられます。これらのバケットは、等度数バケットと呼ばれます。

パーティションの行数が所定のバケット数に等分に(余りなく)分割されない場合、各バケットに割り当てられる行数には最大で1行の差ができます。余りの行は、バケット番号が最小のバケットから順に、バケットごとに1行ずつ分配されます。たとえば、NTILE(5)関数を持つパーティションに103の行がある場合、最初の21行は第1バケットに、次の21行は第2バケットに、次の21行は第3バケットに、次の20行は第4バケットに、最後の20行は第5バケットに分割されます。

NTILE関数の構文は次のとおりです。

```
NTILE (expr) OVER ([query_partition_clause] order_by_clause)
```

NTILE(N)のNは、定数(5など)または式になります。

この関数では、RANKやCUME_DISTと同様に、グループごとの計算用にPARTITION BY句、メジャーおよびそのソート順序の指定用にORDER BY句、および特定のNULL処理用にNULLS FIRST | NULLS LAST句を使用できます。例として、各月の総売上を4つのバケットのそれぞれに割り当てる場合を次に示します。

```
SELECT calendar_month_desc AS MONTH , TO_CHAR(SUM(amount_sold),
 '9,999,999,999')
 SALES$, NTILE(4) OVER (ORDER BY SUM(amount_sold)) AS TILE4
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id AND sales.cust_id=customers.cust_id
  AND sales.time_id=times.time_id AND sales.channel_id=channels.channel_id
  AND times.calendar_year=2000 AND prod_category= 'Electronics'
GROUP BY calendar_month_desc;

MONTH    SALES$              TILE4
-------- -------------- ----------
2000-02         242,416          1
2000-01         257,286          1
2000-03         280,011          1
2000-06         315,951          2
```

```
2000-05          316,824          2
2000-04          318,106          2
2000-07          433,824          3
2000-08          477,833          3
2000-12          553,534          3
2000-10          652,225          4
2000-11          661,147          4
2000-09          691,449          4
```

再生可能な結果を得るには、NTILE ORDER BY文を完全に指定する必要があります。同等の値は、隣接バケット間で分散されます。確定的な結果を確実に得るには、一意キーで順序付けを行う必要があります。

19.2.1.7 ROW_NUMBER関数

ROW_NUMBER関数では、(ORDER BYで定義されたとおり、1から順番に)一意の番号がパーティション内の各行に割り当てられます。構文は次のとおりです。

```
ROW_NUMBER ( ) OVER ( [query_partition_clause] order_by_clause )
```

例19-6 ROW_NUMBER

```
SELECT channel_desc, calendar_month_desc,
   TO_CHAR(TRUNC(SUM(amount_sold), -5), '9,999,999,999') SALES$,
   ROW_NUMBER() OVER (ORDER BY TRUNC(SUM(amount_sold), -6) DESC) AS ROW_NUMBER
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id AND sales.cust_id=customers.cust_id
  AND sales.time_id=times.time_id AND sales.channel_id=channels.channel_id
  AND times.calendar_month_desc IN ('2001-09', '2001-10')
GROUP BY channel_desc, calendar_month_desc;

CHANNEL_DESC         CALENDAR SALES$         ROW_NUMBER
-------------------- -------- -------------- ----------
Direct Sales         2001-10       1,000,000          1
Direct Sales         2001-09       1,100,000          2
Internet             2001-09         500,000          3
Partners             2001-09         600,000          4
Partners             2001-10         600,000          5
Internet             2001-10         700,000          6
```

上述の結果には同等の値のペアが3組あります。NTILEと同様、ROW_NUMBERも不確定的な関数なので、同じ内容の値のそれぞれにおいて行番号が変わる場合があります。確定的な結果を確実に得るには、一意キーで順序付けを行う必要があります。ほとんどの場合、同じ内容の値を解決する列を新しく問合せに追加し、それをORDER BYの指定で使用する必要があります。

19.2.2 ウィンドウ関数

一連のウィンドウ関数を使用すると、累積集計、移動集計および集中集計を計算できます。この種類の関数では、対応するウィンドウ内にある他の行に基づいて、表内の各行に対する値が戻されます。集計ウィンドウ関数群では、SUM、AVERAGE、COUNT、MAX、MINを始めとする多数の関数の移動および累積バージョンの計算が可能です。この種の関数は、問合せのSELECT句およびORDER BY句でのみ使用できます。集計ウィンドウ関数群には、ウィンドウ内の最初の値が戻されるFIRST_VALUE、ウィンドウ内の最後の値が戻されるLAST_VALUEといった便利な関数があります。これらの関数を使用すると、自己結合なしで表の複数の行にアクセスできます。ウィンドウ関数の構文は次のとおりです。

```
analytic_function([ arguments ])
   OVER (analytic_clause)

 where analytic_clause =
     [ query_partition_clause ]
```

```
     [ order_by_clause [ windowing_clause ] ]

and query_partition_clause =
    PARTITION BY
      { value_expr[, value_expr ]...
      }

 and windowing_clause =
     { ROWS | RANGE }
     { BETWEEN
       { UNBOUNDED PRECEDING
     | CURRENT ROW
     | value_expr { PRECEDING | FOLLOWING }
     }
     AND
     { UNBOUNDED FOLLOWING
     | CURRENT ROW
     | value_expr { PRECEDING | FOLLOWING }
     }
   | { UNBOUNDED PRECEDING
     | CURRENT ROW
     | value_expr PRECEDING
     }
   }
```

DISTINCTキーワードはMAXおよびMINを除くウィンドウ関数ではサポートされません。

関連項目:

構文および制限の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

この項では、次の項目について説明します。

- ウィンドウ関数に入力したNULLの処理について
- 論理オフセットを指定したウィンドウ関数
- 集中集計関数
- 重複がある場合の集計ウィンドウ関数
- 行ごとに変動するウィンドウ・サイズ
- 物理オフセットを指定した集計ウィンドウ関数の例
- ウィンドウ関数を使用したパラレル・パーティション・ワイズ操作

19.2.2.1 ウィンドウ関数に入力したNULLの取扱いについて

ウィンドウ関数のNULLの処理方法は、SQL集計関数のNULLの処理方法と同じです。その他の処理方法は、ユーザー定義の関数によって、またはウィンドウ関数でDECODEかCASE式を使用することによって取得できます。

19.2.2.2 論理オフセットを指定したウィンドウ関数

論理オフセットは、RANGE 10 PRECEDINGなどの定数または定数を求める式で指定するか、RANGE INTERVAL N DAY/MONTH/YEAR PRECEDINGなどのインターバル指定またはインターバルを求める式によって指定できます。

論理オフセットでは、NUMERIC(オフセットが数値の場合)またはDATE(インターバルが指定される場合)と互換性のある型の式を

1つのみ、関数のORDER BY式リストに指定できます。

RANGEキーワードを使用する分析関数で、ウィンドウとして次の2つのいずれかを指定する場合は、ORDER BY句に複数のソート・キーを指定できます。

- RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW。このウィンドウの短縮形はRANGE UNBOUNDED PRECEDINGです。この短縮形を使用することもできます。
- RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING。

ウィンドウの境界がこれらの条件に合致しない場合、分析関数のORDER BY句に指定できるソート・キーは1つのみとなります。

例19-7 累積集計関数

2000年の四半期別、顧客ID別の累積amount_soldの例を次に示します。

```
SELECT c.cust_id, t.calendar_quarter_desc, TO_CHAR (SUM(amount_sold),
  '9,999,999,999.99') AS Q_SALES, TO_CHAR(SUM(SUM(amount_sold))
OVER (PARTITION BY c.cust_id ORDER BY c.cust_id, t.calendar_quarter_desc
ROWS UNBOUNDED
PRECEDING), '9,999,999,999.99') AS CUM_SALES
  FROM sales s, times t, customers c
  WHERE s.time_id=t.time_id AND s.cust_id=c.cust_id AND t.calendar_year=2000
    AND c.cust_id IN (2595, 9646, 11111)
  GROUP BY c.cust_id, t.calendar_quarter_desc
  ORDER BY c.cust_id, t.calendar_quarter_desc;

   CUST_ID CALENDA Q_SALES           CUM_SALES
---------- ------- ----------------- -----------------
      2595 2000-01            659.92            659.92
      2595 2000-02            224.79            884.71
      2595 2000-03            313.90          1,198.61
      2595 2000-04          6,015.08          7,213.69
      9646 2000-01          1,337.09          1,337.09
      9646 2000-02            185.67          1,522.76
      9646 2000-03            203.86          1,726.62
      9646 2000-04            458.29          2,184.91
     11111 2000-01             43.18             43.18
     11111 2000-02             33.33             76.51
     11111 2000-03            579.73            656.24
     11111 2000-04            307.58            963.82
```

この例では、分析関数SUMによって各行のウィンドウが定義されます。ウィンドウはパーティションの先頭から開始され(UNBOUNDED PRECEDING)、デフォルトではカレント行で終了します。

この例では、それ自体がSUMである値に対してSUMを実行しているため、ネストされたSUMが必要です。ネストされた集計は、分析集計関数できわめて頻繁に使用されます。

例19-8 移動集計関数

次に、時間ベースのウィンドウの例を示します。このウィンドウは、特定の顧客について、今月と過去2か月間の売上の移動平均を示すものです。

```
SELECT c.cust_id, t.calendar_month_desc, TO_CHAR (SUM(amount_sold),
      '9,999,999,999') AS SALES, TO_CHAR(AVG(SUM(amount_sold))
OVER (ORDER BY c.cust_id, t.calendar_month_desc ROWS 2 PRECEDING),
      '9,999,999,999') AS MOVING_3_MONTH_AVG
FROM sales s, times t, customers c
WHERE s.time_id=t.time_id AND s.cust_id=c.cust_id
   AND t.calendar_year=1999 AND c.cust_id IN (6510)
```

```
GROUP BY c.cust_id, t.calendar_month_desc
ORDER BY c.cust_id, t.calendar_month_desc;

   CUST_ID CALENDAR SALES          MOVING_3_MONTH
---------- -------- -------------- --------------
      6510 1999-04            125            125
      6510 1999-05          3,395          1,760
      6510 1999-06          4,080          2,533
      6510 1999-07          6,435          4,637
      6510 1999-08          5,105          5,207
      6510 1999-09          4,676          5,405
      6510 1999-10          5,109          4,963
      6510 1999-11            802          3,529
```

ウィンドウ計算は問合せにより取り出されたデータの境界を越えることはないため、出力データの3か月の移動平均計算のうち、最初の2行は指定したよりも小さいインターバルに基づいていることに注意してください。結果セットの境界で見られるこのような異なるウィンドウ・サイズを考慮する必要があります。つまり、必要な内容のみが含まれるように問合せの変更が必要になることがあります。

19.2.2.3 集中集計関数

カレント行の前後に集中する集計ウィンドウ関数の計算は簡単です。次の例では、すべての顧客について、1999年12月末の1週間における売上の集中移動平均を計算します。ここで、カレント行の前後1日(カレント行も含む)の合計売上の平均がわかります。

例19-9 集中集計

```
SELECT t.time_id, TO_CHAR (SUM(amount_sold), '9,999,999,999')
AS SALES, TO_CHAR(AVG(SUM(amount_sold)) OVER
  (ORDER BY t.time_id
   RANGE BETWEEN INTERVAL '1' DAY PRECEDING AND
   INTERVAL '1' DAY FOLLOWING), '9,999,999,999') AS CENTERED_3_DAY_AVG
   FROM sales s, times t
   WHERE s.time_id=t.time_id AND t.calendar_week_number IN (51)
     AND calendar_year=1999
   GROUP BY t.time_id
   ORDER BY t.time_id;

TIME_ID   SALES          CENTERED_3_DAY
--------- -------------- --------------
20-DEC-99        134,337        106,676
21-DEC-99         79,015        102,539
22-DEC-99         94,264         85,342
23-DEC-99         82,746         93,322
24-DEC-99        102,957         82,937
25-DEC-99         63,107         87,062
26-DEC-99         95,123         79,115
```

ウィンドウ計算は問合せにより取り出されたデータの境界は越えないため、出力データの各製品の集中移動平均計算のうち、最初の行と最後の行は2日にのみ基づいています。前述の例のように、結果セットの境界で見られるこのような異なるウィンドウ・サイズを考慮する必要があります。場合によっては、問合せの調整が必要となります。

19.2.2.4 重複がある場合の集計ウィンドウ関数

次の例では、同じ内容の値のデータが存在する場合、つまり単一の順序値に対して複数の行が戻される場合に、集計ウィンドウ関数で値がどのように計算されるかを示します。次の問合せでは、特定の期間における複数の顧客に対する売上高が取得されます。(実データセットを定義するためにインライン・ビューを使用していますが、特に意味はないので無視してかまいません。)

この問合せには、カレント行の日付から10日前まで実行される変動ウィンドウが定義されています。この例のウィンドウ句の定義にRANGEキーワードが使用されていることに注意してください。これは、このウィンドウが、範囲内にある各値に対して多くの行を保持できることを意味しています。この場合、重複するデータ値を持つ行が3組あります。

例19-10 論理オフセットを指定した集計ウィンドウ関数

```
SELECT time_id, daily_sum, SUM(daily_sum) OVER (ORDER BY time_id
RANGE BETWEEN INTERVAL '10' DAY PRECEDING AND CURRENT ROW)
AS current_group_sum
FROM (SELECT time_id, channel_id, SUM(s.quantity_sold)
AS daily_sum
FROM customers c, sales s, countries
WHERE c.cust_id=s.cust_id
  AND c.country_id = countries.country_id
  AND s.cust_id IN (638, 634, 753, 440 ) AND s.time_id BETWEEN '01-MAY-00'
  AND '13-MAY-00' GROUP BY time_id, channel_id);

TIME_ID    DAILY_SUM CURRENT_GROUP_SUM
--------- ---------- -----------------
06-MAY-00          7                 7   /* 7 */
10-MAY-00          1                 9   /* 7 + (1+1) */
10-MAY-00          1                 9   /* 7 + (1+1) */
11-MAY-00          2                15   /* 7 + (1+1) + (2+4) */
11-MAY-00          4                15   /* 7 + (1+1) + (2+4) */
12-MAY-00          1                16   /* 7 + (1+1) + (2+4) + 1 */
13-MAY-00          2                23   /* 7 + (1+1) + (2+4) + 1 + (5+2) */
13-MAY-00          5                23   /* 7 + (1+1) + (2+4) + 1 + (5+2) */
```

この例の出力では、5月6日と5月12日を除くすべての日付について2つの行が戻されています。出力の右側のコメントにある数字をチェックして、値がどのように計算されているかを確認します。カッコ内の各グループは、単一の日について戻された値を表します。

この例は、ROWSキーワードではなくRANGEキーワードを使用した場合にのみ当てはまることに注意してください。RANGEを使用すると、分析関数のORDER BY句でORDER BY式を1つしか使用できないことにも注意が必要です。ROWSキーワードを使用した場合は、分析関数のORDER BY句で複数のORDER BY式を使用できます。

19.2.2.5 行ごとに変動するウィンドウ・サイズ

特定の条件に応じて、行ごとにウィンドウのサイズを変えると便利な場合があります。特定の日付に対してウィンドウを大きくし、その他の日付には小さくする必要がある場合などです。たとえば、3営業日にわたる株価の移動平均を計算するとします。すべての営業日で毎日の行数が同じで、非営業日については格納されていない場合は、物理ウィンドウ関数を使用できます。この条件が満たされない場合に移動平均を計算するには、ウィンドウ・サイズ・パラメータで式を使用します。

ウィンドウ・サイズ指定における式は、様々なソースで作成できます。式には、時間表などの表の列を使用できます。また、カレント行の値に基づいてウィンドウの適切な境界を戻す関数も使用できます。仮想の株価データベースに関する次の文では、RANGE句にユーザー定義関数を使用してウィンドウ・サイズが設定されています。

```
SELECT t_timekey, AVG(stock_price)
        OVER (ORDER BY t_timekey RANGE fn(t_timekey) PRECEDING) av_price
FROM stock, time WHERE st_timekey = t_timekey
ORDER BY t_timekey;
```

この文では、t_timekeyが日付フィールドです。*fn*は、次の仕様を持つPL/SQLファンクションとします。

fn(t_timekey)は次の値を戻します。

- t_timekeyが月曜日または火曜日の場合は4。

- 2 それ以外の場合
- 前に休日があっても、カウントは正しく調整されます。

日付列に関して、ウィンドウ関数でORDER BY句に数値が使用される形でウィンドウが指定されている場合、この数値は日数に変換されます。単にfn(t_timekey)とするのではなく、インターバル・リテラル変換関数をNUMTODSINTERVAL(fn(t_timekey), 'DAY')のように使用しても、同じ処理を行えます。また、INTERVALデータ型の値を戻すPL/SQLファンクションを記述することもできます。

19.2.2.6 物理オフセットを指定した集計ウィンドウ関数の例

行単位で表されるウィンドウでは、順序付け式は、結果を確定的にするために一意である必要があります。たとえば、次の問合せは、この結果セット内でtime_idが一意でないため、確定的ではありません。

例19-11 物理オフセットを指定した集計ウィンドウ関数

```
SELECT t.time_id, TO_CHAR(amount_sold, '9,999,999,999') AS INDIV_SALE,
       TO_CHAR(SUM(amount_sold) OVER (PARTITION BY t.time_id ORDER BY t.time_id
ROWS UNBOUNDED PRECEDING), '9,999,999,999') AS CUM_SALES
FROM sales s, times t, customers c
WHERE s.time_id=t.time_id AND s.cust_id=c.cust_id
  AND t.time_id IN
   (TO_DATE('11-DEC-1999'), TO_DATE('12-DEC-1999'))
  AND c.cust_id
BETWEEN 6500 AND 6600
ORDER BY t.time_id;

TIME_ID   INDIV_SALE    CUM_SALES
--------- ----------    ---------
12-DEC-99          23           23
12-DEC-99           9           32
12-DEC-99          14           46
12-DEC-99          24           70
12-DEC-99          19           89
```

この問題に対処するには、結果セットにprod_id列を追加してtime_idおよびprod_idの両方を順序付けするという方法があります。

19.2.2.7 ウィンドウ関数を使用したパラレル・パーティション・ワイズ操作

SQLウィンドウ関数には問合せのパーティション化句を含めることができ、この句で使用される式に基づいて問合せ結果をグループにパーティション化できます。パーティション表に対するパラレル問合せの場合、パラレル・パーティション・ワイズ操作の要件が満たされていれば、この句で定義されたパーティションをパーティション・ワイズ操作を実行するのに使用できます。これにより、パーティション表に対する高速なSQLウィンドウ問合せが実現されます。

関連項目:

『Oracle Database VLDBおよびパーティショニング・ガイド』

19.2.3 レポート関数

問合せの処理後に、結果の行数や列の平均値といった集計値をパーティション内で簡単に計算でき、他のレポート関数でも使用可能にできます。集計レポート関数では、パーティション内のすべての行に対して同一の集計値が戻されます。この種の関数

におけるNULLへの対応は、SQL集計関数と同じです。構文は次のとおりです。

```
{SUM | AVG | MAX | MIN | COUNT | STDDEV | VARIANCE  ... }
  ([ALL | DISTINCT] {value expression1 [,...] | *})
   OVER ([PARTITION BY value expression2[,...]])
```

さらに、次の条件が適用されます。

- アスタリスク(*)は、COUNT(*)でのみ使用できます。
- DISTINCTは、対応する集計関数で許可されている場合にのみサポートされます。
- *value expression1*および*value expression2*には、列参照または集計を含む有効な式を指定できます。
- PARTITION BY句には、ウィンドウ関数の計算対象とするグループを定義します。PARTITION BY句がない場合、この関数は問合せ結果セット全体に対して計算されます。

関連項目:

RATIO_TO_REPORT関数

レポート関数は、SELECT句またはORDER BY句でのみ使用できます。レポート関数の主なメリットは、単一の問合せブロックでデータの複数のパスを実行し、問合せパフォーマンスをスピードアップできることです。「売上が市全体の売上の10%以上である販売員の数をカウントする」といった問合せで、別々の問合せブロック間の結合が必要ありません。

たとえば、「各製品カテゴリについて、最大の売上を記録した地域を検索する」という質問について考えてみます。MAX集計レポート関数を使用する同等のSQL問合せは、次のようになります。

```
SELECT prod_category, country_region, sales
FROM (SELECT SUBSTR(p.prod_category,1,8) AS prod_category, co.country_region,
 SUM(amount_sold) AS sales,
MAX(SUM(amount_sold)) OVER (PARTITION BY prod_category) AS MAX_REG_SALES
FROM sales s, customers c, countries co, products p
WHERE s.cust_id=c.cust_id AND c.country_id=co.country_id
   AND s.prod_id =p.prod_id AND s.time_id = TO_DATE('11-OCT-2001')
GROUP BY prod_category, country_region)
WHERE sales = MAX_REG_SALES;
```

集計レポート関数MAX(SUM(amount_sold))を含む内部問合せでは、次の内容が戻されます。

```
PROD_CAT COUNTRY_REGION            SALES MAX_REG_SALES
-------- -------------------- ---------- -------------
Electron Americas                581.92        581.92
Hardware Americas                925.93        925.93
Peripher Americas               3084.48       4290.38
Peripher Asia                   2616.51       4290.38
Peripher Europe                 4290.38       4290.38
Peripher Oceania                 940.43       4290.38
Software Americas                4445.7        4445.7
Software Asia                   1408.19        4445.7
Software Europe                 3288.83        4445.7
Software Oceania                 890.25        4445.7
```

問合せ結果の全体は、次のようになります。

```
PROD_CAT COUNTRY_REGION            SALES
-------- -------------------- ----------
```

```
Electron Americas                581.92
Hardware Americas                925.93
Peripher Europe                 4290.38
Software Americas                4445.7
```

例19-12 集計レポートの例

集計レポートをネストされた問合せと組み合せると、複雑な問合せに対する応答が効率化されます。たとえば、最も重要な製品サブカテゴリで最も売れ行きがよい製品を知りたいとします。製品カテゴリ内の売上の20%以上を占める製品サブカテゴリについて、サブカテゴリごとに売上上位5位までの製品を検索する問合せは、次のようになります。

```
SELECT SUBSTR(prod_category,1,8) AS CATEG, prod_subcategory, prod_id, SALES
FROM (SELECT p.prod_category, p.prod_subcategory, p.prod_id,
      SUM(amount_sold) AS SALES,
      SUM(SUM(amount_sold)) OVER (PARTITION BY p.prod_category) AS CAT_SALES,
      SUM(SUM(amount_sold)) OVER
         (PARTITION BY p.prod_subcategory) AS SUBCAT_SALES,
      RANK() OVER  (PARTITION BY p.prod_subcategory
         ORDER BY SUM(amount_sold) ) AS RANK_IN_LINE
     FROM sales s, customers c, countries co, products p
     WHERE s.cust_id=c.cust_id
       AND c.country_id=co.country_id AND  s.prod_id=p.prod_id
       AND s.time_id=to_DATE('11-OCT-2000')
     GROUP BY p.prod_category, p.prod_subcategory, p.prod_id
     ORDER BY prod_category, prod_subcategory)
  WHERE SUBCAT_SALES>0.2*CAT_SALES AND RANK_IN_LINE<=5;
```

19.2.3.1 RATIO_TO_REPORT関数

RATIO_TO_REPORT関数では、値の集合の合計に対して、ある値の割合が計算されます。式value expressionがNULLと評価される場合、RATIO_TO_REPORTもNULLと評価されますが、分母の値の合計を計算する際には0(ゼロ)として扱われます。構文は次のとおりです。

```
RATIO_TO_REPORT ( expr ) OVER ( [query_partition_clause] )
```

ここでは、次の事項が適用されます。

- exprには、列参照または集計を含む任意の有効な式を指定できます。
- PARTITION BY句には、RATIO_TO_REPORT関数の計算対象とするグループを定義します。PARTITION BY句がない場合、この関数は問合せ結果セット全体に対して計算されます。

例19-13 RATIO_TO_REPORT

チャネル別売上のRATIO_TO_REPORTを計算するには、次のような構文を使用できます。

```
SELECT ch.channel_desc, TO_CHAR(SUM(amount_sold),'9,999,999') AS SALES,
     TO_CHAR(SUM(SUM(amount_sold)) OVER (), '9,999,999') AS TOTAL_SALES,
     TO_CHAR(RATIO_TO_REPORT(SUM(amount_sold)) OVER (), '9.999')
     AS RATIO_TO_REPORT
FROM sales s, channels ch
WHERE s.channel_id=ch.channel_id  AND s.time_id=to_DATE('11-OCT-2000')
GROUP BY ch.channel_desc;

CHANNEL_DESC         SALES      TOTAL_SALE RATIO_
-------------------- ---------- ---------- ------
Direct Sales             14,447     23,183   .623
Internet                    345     23,183   .015
Partners                  8,391     23,183   .362
```

19.2.4 LAG/LEAD関数

LAGおよびLEAD関数は、行の相対位置を確実に認識できる場合の、値の比較に役立ちます。この2つの関数は、カレント行からターゲット行までの行数を指定する形で使用します。これらの関数では、表の複数の行に対して自己結合せずに同時にアクセスできるため、処理速度が向上します。LAG関数では、現在位置から指定したオフセット分のみ前の行がアクセスされ、LEAD関数では、現在位置から指定したオフセット分のみ後の行がアクセスされます。LAGおよびLEAD関数の構文ではこうした関数の構文について説明します。

NTH_VALUE関数と関連があり、これを単純化したものであると考えられるのがLAG関数とLEAD関数です。LAGとLEADでは、指定した物理オフセットの行の値を取得できるだけです。これでは不十分な場合は、NTH_VALUEを使用して、いわゆる論理オフセットまたは相対位置に基づく行の値を取得できます。NTH_VALUE関数ではIGNORE NULLSオプションを指定できます。つまり、条件を指定したり、特定の条件に基づき行を除外できるという意味で、その機能を強化できます。例19-17では、数量が8未満の行はフィルタにより除外されています。これはLAGやLEADではできません。目的の行までのオフセットがわからないためです。

詳細は、NTH_VALUE関数および*Oracle Database SQL言語リファレンス*を参照してください。

19.2.4.1 LAG/LEADの構文

これらの関数の構文は次のとおりです。

```
{LAG | LEAD} ( value_expr [, offset] [, default] ) [RESPECT NULLS|IGNORE NULLS]
   OVER ( [query_partition_clause] order_by_clause )
```

*offset*はオプションのパラメータで、デフォルトは1です。*default*はオプションのパラメータで、*offset*が表またはパーティションの境界外となる場合に戻される値です。IGNORE NULLSを指定した場合、戻される値は、NULLの行を無視した上で、指定されたLAGまたはLEADのオフセットの行から取得されます。

例19-14 LAG/LEAD

この例は、LAGおよびLEADの一般的な使用例を示しています。

```
SELECT time_id, TO_CHAR(SUM(amount_sold),'9,999,999') AS SALES,
  TO_CHAR(LAG(SUM(amount_sold),1) OVER (ORDER BY time_id),'9,999,999') AS LAG1,
  TO_CHAR(LEAD(SUM(amount_sold),1) OVER (ORDER BY time_id),'9,999,999') AS LEAD1
FROM sales
WHERE time_id>=TO_DATE('10-OCT-2000') AND time_id<=TO_DATE('14-OCT-2000')
GROUP BY time_id;

TIME_ID   SALES      LAG1       LEAD1
--------- ---------- ---------- ----------
10-OCT-00    238,479                 23,183
11-OCT-00     23,183    238,479     24,616
12-OCT-00     24,616     23,183     76,516
13-OCT-00     76,516     24,616     29,795
14-OCT-00     29,795     76,516
```

LAG/LEAD関数を使用して、スパースなデータに対する周期ごとの比較問合せを実行する方法は、レポート用のデータの稠密化を参照してください。

例19-15 IGNORE NULLSを使用したLAG/LEAD

この例は、LAGおよびLEADでIGNORE NULLSオプションを指定する一般的な使用例を示しています。

```
SELECT prod_id, channel_id, SUM(quantity_sold) quantity,
 CASE WHEN SUM(quantity_sold) < 5000 THEN SUM(amount_sold) ELSE NULL END amount,
 LAG(CASE WHEN SUM(quantity_sold) < 5000 THEN SUM(amount_sold) ELSE NULL END)
 IGNORE NULLS OVER (PARTITION BY prod_id ORDER BY channel_id) lag
```

```
FROM sales
WHERE prod_id IN (18,127,138)
GROUP BY prod_id, channel_id;

 PROD_ID  CHANNEL_ID  QUANTITY   AMOUNT          LAG
--------  ----------  --------   -----------     ----------
      18           2      2888    4420923.94
      18           3      5615                   4420923.94
      18           4      1088    1545729.81     4420923.94
     127           2      4508     274088.08
     127           3      9626                    274088.08
     127           4      1850     173682.67      274088.08
     138           2      1120      127390.3
     138           3      3878     393111.15       127390.3
     138           4       543      71203.21      393111.15

9 rows selected.
```

19.2.5 FIRST_VALUE関数、LAST_VALUE関数およびNTH_VALUE関数

この項では、次のトピックで説明するFIRST_VALUE、LAST_VALUEおよびNTH_VALUE関数について説明します。

- FIRST_VALUE関数およびLAST_VALUE関数
- NTH_VALUE関数

19.2.5.1 FIRST_VALUE関数およびLAST_VALUE関数

FIRST_VALUE関数やLAST_VALUE関数を使用すると、ウィンドウの最初や最後の行を選択できます。これらの行は、計算上の基準行として使用されるので特に重要です。これらの行を使用する例としては、たとえば、日付で順序付けされた売上データが保持されるパーティションについて、「その期間の最初の販売日(FIRST_VALUE)と比較した各日の売上はどの程度か」といった質問をする場合があります。

FIRST_VALUEでIGNORE NULLSオプションを使用すると、セット内で最初のNULL以外の値が戻されます。すべての値がNULLの場合は、NULLが戻されます。LAST_VALUEでIGNORE NULLSを使用すると、セット内で最後のNULL以外の値が戻されます。すべての値がNULLの場合は、NULLが戻されます。IGNORE NULLSオプションは、在庫表を適切に移入する場合に特に役立ちます。

これらの関数の構文は次のとおりです。

```
FIRST_VALUE|LAST_VALUE ( <expr> ) [RESPECT NULLS|IGNORE NULLS] OVER (analytic clause );
```

例19-16 FIRST_VALUE

この例では、FIRST_VALUEでIGNORE NULLSオプションを使用しています。

```
SELECT prod_id, channel_id, time_id,
  CASE WHEN MIN(amount_sold) > 9.5
    THEN MIN(amount_sold) ELSE NULL END amount_sold,
  FIRST_VALUE(CASE WHEN MIN(amount_sold) > 9.5
    THEN min(amount_sold) ELSE NULL END)
    IGNORE NULLS OVER (PARTITION BY prod_id
                       ORDER BY channel_id DESC, time_id
                       ROWS BETWEEN UNBOUNDED PRECEDING
                       AND UNBOUNDED FOLLOWING) nv FROM sales
                       WHERE prod_id = 115 AND time_id BETWEEN '18-DEC-01'
                       AND '22-DEC-01' GROUP BY prod_id, channel_id, time_id
                       ORDER BY prod_id;

  PROD_ID CHANNEL_ID TIME_ID    AMOUNT_SOLD         NV
```

```
---------- ---------- --------- ---------- ----------
       115          4 18-DEC-01                  9.66
       115          4 19-DEC-01                  9.66
       115          4 20-DEC-01                  9.66
       115          4 22-DEC-01                  9.66
       115          3 18-DEC-01         9.66     9.66
       115          3 19-DEC-01         9.66     9.66
       115          3 20-DEC-01         9.66     9.66
       115          3 21-DEC-01         9.66     9.66
       115          3 22-DEC-01         9.66     9.66
       115          2 18-DEC-01         9.67     9.66
       115          2 19-DEC-01         9.67     9.66
       115          2 21-DEC-01         9.67     9.66
       115          2 22-DEC-01         9.67     9.66

13 rows selected.
```

19.2.5.2 NTH_VALUE関数

NTH_VALUE関数では、ウィンドウ内の任意の行から列値を見つけることができます。たとえば、ある企業の株価について、ある年で5番目に高かった終値を検索する場合に使用できます。

NTH_VALUE関数と関連があり、これを単純化したものであると考えられるのがLAG関数とLEAD関数です。LAGとLEADでは、指定した物理オフセットの行の値を取得できるだけです。これでは不十分な場合は、NTH_VALUEを使用して、いわゆる論理オフセットまたは相対位置に基づく行の値を取得できます。NTH_VALUE、FIRST_VALUEおよびLAST_VALUE関数では、IGNORE NULLSオプションを使用して、条件を指定したり、特定の条件に基づいて行を除外できるという意味で、その機能を強化できます。例19-17では、数量が8未満の行はフィルタにより除外されています。これはLAGやLEADではできません。目的の行までのオフセットがわからないためです。

詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

この関数の構文は次のとおりです。

```
NTH_VALUE (<expr>, <n expr>) [FROM FIRST | FROM LAST]
[RESPECT NULLS | IGNORE NULLS] OVER (<window specification>)
```

- exprには、列、定数、バインド変数、またはこれらを含む式を指定できます。
- nには、列、定数、バインド変数、またはこれらを含む式を指定できます。
- RESPECT NULLSは、NULLを処理するデフォルトのメカニズムです。exprのNULL値を計算に含めるか、計算から除外するかを決定します。デフォルトはRESPECT NULLSです。
- FROM FIRSTおよびFROM LASTオプションでは、オフセットnが最初の行からか最後の行からかを指定できます。デフォルトはFROM FIRSTです。
- IGNORE NULLSを指定すると、メジャー値内のNULLをスキップできます。

例19-17 NTH_VALUE

次の例は、prod_idが10から20の範囲の各製品について、昇順で2番目のchannel_idのamount_sold値を戻します。

```
SELECT prod_id, channel_id, MIN(amount_sold),
  NTH_VALUE(MIN(amount_sold), 2) OVER (PARTITION BY prod_id ORDER BY channel_id
  ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) NV
FROM sales
WHERE prod_id BETWEEN 10 AND 20 GROUP BY prod_id, channel_id;

   PROD_ID CHANNEL_ID MIN(AMOUNT_SOLD)         NV
```

```
---------- ---------- ---------------- ----------
        13          2           907.34      906.2
        13          3            906.2      906.2
        13          4           842.21      906.2
        14          2          1015.94    1036.72
        14          3          1036.72    1036.72
        14          4           935.79    1036.72
        15          2           871.19     871.19
        15          3           871.19     871.19
        15          4           871.19     871.19
        16          2           266.84     266.84
        16          3           266.84     266.84
        16          4           266.84     266.84
        16          9            11.99     266.84
...
```

19.3 分析用の高度な集計

Oracle Databaseは、高度な集計を実行するための複数のSQL関数を提供しています。また、特定の正確な関数には、近似結果を返す対応する関数が提供されています。

この項では、次に示す高度な分析集計関数について説明します。

- 近似集計について
- LISTAGG関数
- FIRST/LAST関数
- 逆パーセンタイル関数
- 仮説ランク関数
- 線形回帰関数
- 統計集計について
- ユーザー定義集計について

19.3.1 近似集計について

近似集計は、近似結果を返すSQL関数を使用して計算されます。これらは、正確な値を必要とせず、近似処理が許容されるデータ探索問合せで主に使用されます。

APPROX_COUNT_DISTINCT関数は、指定された式に対して個別値を含む行の概数を返します。APPROX_COUNT_DISTINCT_DETAILおよびAPPROX_COUNT_DISTINCT_AGG関数を使用すると、指定されたグループ内の近似個別値カウントの変動する集計レベルを計算できます。これらの集計結果は、その後の分析またはユーザー問合せに応答するために表またはマテリアライズド・ビューに格納できます。

APPROX_COUNT_DISTINCT_DETAIL関数は、WHERE句にリストされるすべてのディメンションのタプルを含むベースレベルのサマリーをバイナリ形式で作成します。APPROX_COUNT_DISTINCT_AGG関数は、APPROX_COUNT_DISTINCT_DETAIL関数によって生成されたデータを使用して、高レベルのタプルをバイナリ形式で抽出します。これにより、元の計算(この場合はAPPROX_COUNT_DISTINCTを使用した計算)を再実行する必要がなくなります。バイナリ形式を使用する集計データは、TO_APPROX_COUNT_DISTINCTを使用して判読可能な形式に変換されます。

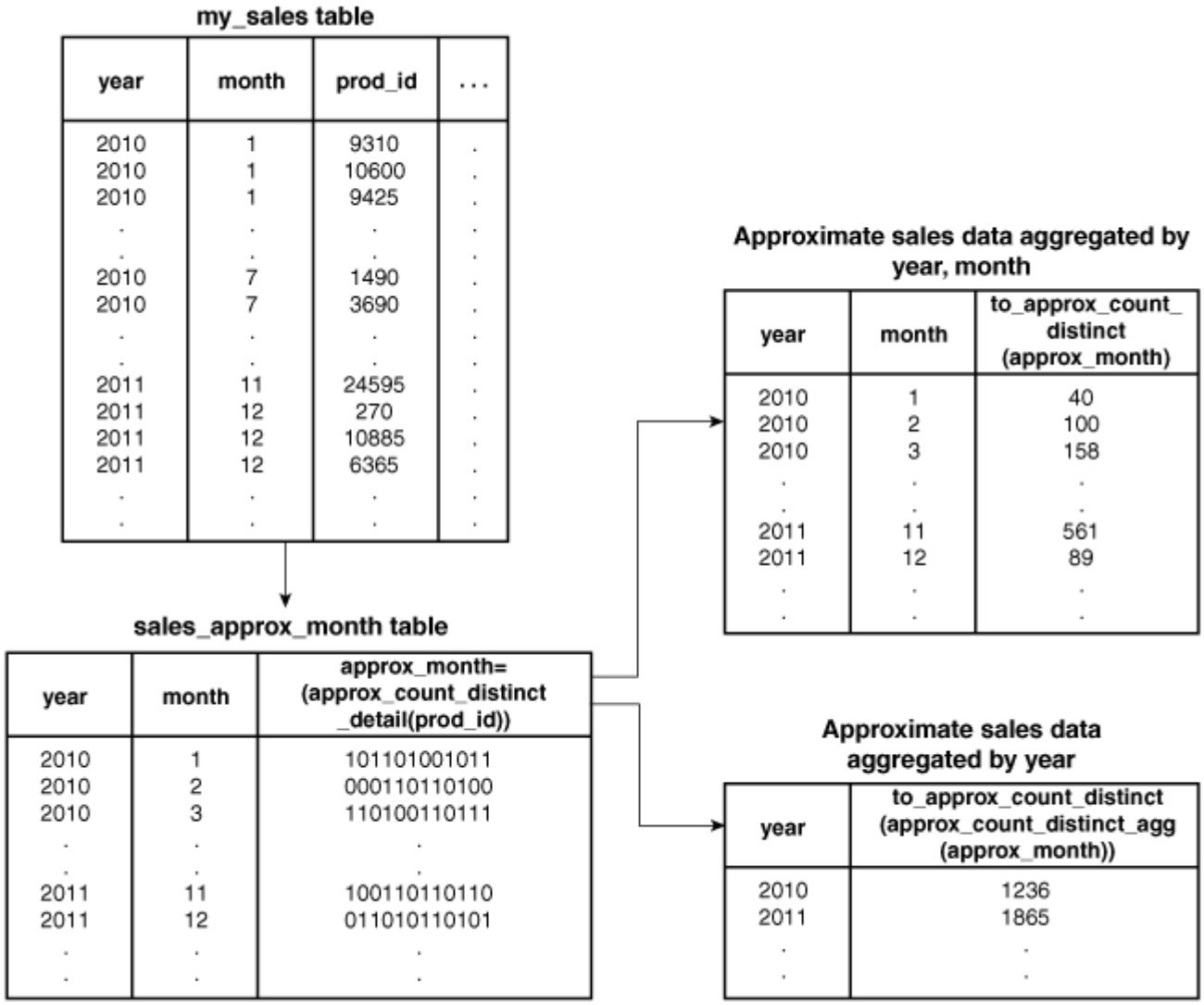
図19-3は、APPROX_COUNT_DISTINCT_DETAILを使用して、各月に販売された各製品の概数を取得する例を示しています。次のような問合せを使用して、my_sales表から選択された販売データは年および月ごとに集計され、SALES_APPROX_MONTH

表に格納されます。

```
INSERT INTO sales_approx_month
   (SELECT year, month, APPROX_COUNT_DISTINCT_DETAIL(prod_id) approx_month
      FROM my_sales
      GROUP BY year, month);
```

approx_monthに格納される値はバイナリ値であることに注意してください。TO_APPROX_COUNT_DISTINCT関数を使用して、これらのバイナリ値を判読可能な形式で表示します。年および月ごとに集計された各製品の販売数を表示するには、approx_month列に対してTO_APPROX_COUNT_DISTINCT関数を使用します。年ごとに集計されたデータを表示するには、approx_month列に格納されたデータに対してTO_APPROX_COUNT_DISTINCT関数およびAPPROX_COUNT_DISTINCT_AGG関数を使用します。

図19-3 SQL関数を使用した近似集計の表示



各年に販売された各製品の概数を別の方法で計算するには、APPROX_COUNT_DISTINCT_AGGを使用して、SALES_APPROX_MONTH表に格納された月次詳細を集計し、この結果を表またはマテリアライズド・ビューに格納します。

近似パーセンタイル結果を返すSQL関数のプロパティ

近似パーセンタイル結果を提供するSQL関数には、APPROX_PERCENTILE、APPROX_PERCENTILE_DETAILおよびAPPROX_PERCENTILE_AGGがあります。これらの関数には、次の追加プロパティがあります。

- ERROR_RATE

  近似計算のエラー率を計算することにより、補間されたパーセンタイル値の精度を示します。

- CONFIDENCE

エラー率の精度の信頼度を示します(エラー率が指定されている場合)。

- DETERMINISTIC

  近似処理の計算に使用されるアルゴリズムを制御します。

  一貫性のある繰返し可能な結果が必要な場合は、DETERMINISTICを使用します。通常、これは結果を他のユーザーと共有する必要がある場合です。

関連項目:

- 近似結果を返すパーセンタイル関数の使用
- 『Oracle Database SQL言語リファレンス』のAPPROX_COUNT_DISTINCTに関する項
- 関数とERROR_RATE、CONFIDENCEおよびDETERMINISTICプロパティの詳細は、Oracle Database SQL言語リファレンスのAPPROX_COUNT_DISTINCT_DETAILに関する項を参照してください。
- 『Oracle Database SQL言語リファレンス』のAPPROX_COUNT_DISTINCT_AGGに関する項
- 『Oracle Database SQL言語リファレンス』のTO_APPROX_COUNT_DISTINCTに関する項

19.3.2 LISTAGG関数

LISTAGG関数は、ORDER BY句に基づいて各グループ内のデータを順序付けてから、メジャー列の値を連結します。

Oracle Database 12cリリース2 (12.2)より前のリリースでは、LISTAGG関数によって返された連結値が、戻り値のデータ型でサポートされる最大長を超える場合、次のエラーが返されます。

ORA-01489: 文字列を連結した結果、長さが最大長を超えました

Oracle Database 12cリリース2 (12.2)以降では、戻り値のデータ型でサポートされる最大長内に収まるように戻り文字列を切り捨てて、戻り値が切り捨てられたことを示す切捨てリテラルを表示できます。切捨ては、最後の完全なデータ値の後に実行されるため、不完全なデータ値が表示されることはありません。

LISTAGG関数の構文は次のとおりです。

```
LISTAGG ( [ALL] [DISTINCT] <measure_column> [,<delimiter>] [ON OVERFLOW TRUNCATE [truncate_literal] |
ON OVERFLOW ERROR  [WITH | WITHOUT COUNT]])
        WITHIN GROUP (ORDER BY <oby_expression_list>)
```

DISTINCTを指定すると、重複する値がリストから削除されます。

measure_columnには、列、定数、バインド変数、またはこれらを含む式を指定できます。

データ型でサポートされる最大長内に戻り文字列が収まっていない場合、エラーを表示することも、戻り文字列を切り捨てて、切捨てリテラルを表示することもできます。デフォルトはON OVERFLOW ERRORで、切捨てが発生したときにエラーを表示します。

truncate_literalには、NULL、文字列リテラルまたは定数式を指定できます。戻り値のデータ型でサポートされる最大長より長い値をLISTAGGが返した場合、最後のデリミタの後、値のリストの末尾にこれが追加されます。デフォルト値は省略記号(...)です。

WITH COUNTは、戻り値のデータ型でサポートされる最大長を超えたためにLISTAGGの出力から切り捨てられたデータ値の件数を表示します。このオプションがデフォルトです。文字列が切り捨てられたときにLISTAGG関数の末尾に件数を表示しない場合は、WITHOUT COUNTを使用します。

delimiterには、NULL (デフォルト値)、文字列リテラル、バインド変数または定数式を指定できます。これは必須パラメータで

す。デリミタを指定しない場合は、NULLがデリミタとして使用されます。

oby_expression_listには、任意の順序付けオプションを含む式のリストを指定できます。昇順または降順でソートしたり(ASCまたはDESC)、NULLのソート順を制御したり(NULLS FIRSTまたはNULLS LAST)するときに使用します。デフォルトは、ASCENDINGおよびNULLS LASTです。

関連項目:

VARCHAR2データ型でサポートされる最大長の詳細は、Oracle Database SQL言語リファレンスを参照してください。

19.3.2.1 集計として使用するLISTAGG

LISTAGG関数を集計として使用できます。

例19-18 集計として使用するLISTAGG

次の例では、LISTAGGを集計として使用しています。

```
SELECT prod_id, LISTAGG(cust_first_name||' '||cust_last_name, '; ')
  WITHIN GROUP (ORDER BY amount_sold DESC) cust_list
FROM sales, customers
WHERE sales.cust_id = customers.cust_id AND cust_gender = 'M'
  AND cust_credit_limit = 15000 AND prod_id BETWEEN 15 AND 18
  AND channel_id = 2 AND time_id > '01-JAN-01'
GROUP BY prod_id;

PROD_ID  CUST_LIST
-------  ----------------------------------------------
     15  Hope Haarper; Roxanne Crocker; ... Mason Murray
     16  Manvil Austin; Bud Pinkston; ... Helga Nickols
     17  Opal Aaron; Thacher Rudder; ... Roxanne Crocker
     18  Boyd Lin; Bud Pinkston; ... Erik Ready
```

この出力は、読みやすく変更されています。この場合、省略記号は、最後の顧客名の前のいくつかの値が出力から省略されていることを示します。

例19-19 戻り文字列が最大許容長を超えるLISTAGG

この例では、GROUP BY句で指定した各グループ内のデータが順序付けられ、cust_first_nameおよびcust_last_name列の値が連結されます。連結された名前のリストが、VARCHAR2データ型でサポートされる最大長を超える場合、リストは最後の完全な名前まで切り捨てられます。リストの末尾に、「...」というオーバーフロー・リテラルが追加され、その後に、切り捨てられた値の件数が表示されます。

```
SELECT country_region,
  LISTAGG(s.CUST_FIRST_NAME||' '|| s.CUST_LAST_NAME, ';' ON OVERFLOW TRUNCATE WITH COUNT) WITHIN GROUP
(ORDER BY s.cust_id) AS customer_names
FROM countries c, customers s
WHERE c.country_id = s.country_id
GROUP BY c.country_region
ORDER BY c.country_region;

COUNTRY_REGION
--------------------
CUSTOMER_NAMES
--------------------------------------------------------------------------------
Africa
```

```
Laurice Lincoln;Kirsten Newkirk;Verna Yarborough;Chloe Dwyer;Betty Sampler;Terry
 Hole;Waren Parkburg;Uwe Feldman;Douglas Hanson;Woodrow Lazar;Alfred Doctor;Stac
.
.
Zwolinsky;Buzz Milenova;Abbie Venkayala

COUNTRY_REGION
--------------------
CUSTOMER_NAMES
--------------------------------------------------------------------------------
Americas
Linette Ingram;Vida Puleo;Gertrude Atkins;Sibil Haul;Raina Cassidy;Kaula Daley;G
abriela Sean;Dolores Moore;Erica Vandermark;Madallyn Ladd;Carolyn Hinkle;Leonora
.
.
emphill;Urban Smyth;Murry Ivy;Steven Lauers;...(21482)

COUNTRY_REGION
--------------------
CUSTOMER_NAMES
--------------------------------------------------------------------------------
Asia
Harriett Charles;Willa Fitz;Faith Fischer;Gay Nance;Maggie Cain;Neda Clatterbuck
;Justa Killman;Penelope Oliver;Mandisa Grandy;Marette Overton;Astrid Rice;Poppy
.
.
ob Gentile;Lynn Hardesty;Mabel Barajas;...(1648)

COUNTRY_REGION
--------------------
CUSTOMER_NAMES
--------------------------------------------------------------------------------
Europe
Abigail Kessel;Anne Koch;Buick Emmerson;Frank Hardy;Macklin Gowen;Rosamond Kride
r;Raina Silverberg;Gloria Saintclair;Macy Littlefield;Yuri Finch;Bertilde Sexton
.
.
el Floyd;Lincoln Sean;Morel Gregory;Kane Speer;...(30284)

COUNTRY_REGION
--------------------
CUSTOMER_NAMES
--------------------------------------------------------------------------------
Middle East
Dalila Rockwell;Alma Elliott;Cara Jeffreys;Joy Sandstrum;Elizabeth Barone;Whitby
 Burnns;Geoffrey Door;Austin Dutton;Tobin Newcomer;Blake Overton;Lona Kimball;Lo
.
.
edy;Brandon Moy;Sydney Fenton

COUNTRY_REGION
--------------------
CUSTOMER_NAMES
--------------------------------------------------------------------------------
Oceania
Fredericka Umstatt;Viola Nettles;Alyce Reagan;Catherine Odenwalld;Mauritia Linde
green;Heidi Schmidt;Ray Wade;Cicily Graham;Myrtle Joseph;Joan Morales;Brenda Obr
.
.
;Fredie Elgin;Gilchrist Lease;Guthrey Cain;...(793)
```

```
6 rows selected.
```

例19-20 DISTINCTを使用して繰返し値が削除されたLISTAGG

この例では、GROUP BY句で指定された各グループ内のデータを順序付け、prod_cateogry列およびprod_desc列の値を連結します。連結された名前のリストが、VARCHAR2データ型でサポートされる最大長を超える場合、リストは最後の完全な文字列まで切り捨てられます。DISTINCTキーワードは、指定したメジャー列の重複値を削除する必要があることを指定します。

```
SELECT cust_id, LISTAGG( DISTINCT prod_category||':'||prod_desc,' ; ' ON OVERFLOW TRUNCATE WITH COUNT)
WITHIN GROUP (ORDER BY amount_sold)
FROM sh.sales, sh.products
WHERE sales.prod_id=products.prod_id
AND amount_sold > 200 AND products.prod_id BETWEEN 10 and 15
AND time_id > '01-JAN-01'
GROUP BY cust_id;
```

19.3.2.2 集計レポートとして使用するLISTAGG

LISTAGG関数を集計レポートとして使用できます。

例19-21 集計レポートとして使用するLISTAGG

この例では、LISTAGGを集計レポートとして使用しています。各期間内の各製品の最も低い原価を抽出します。

```
connect sh/sh
set lines 120 pages 20000
column list format A40

SELECT time_id, prod_id, LISTAGG(MIN(unit_cost),';')
      WITHIN GROUP (ORDER BY prod_id) OVER (PARTITION BY time_id) lowest_unit_cost
  FROM sh.sales_transactions_ext
  WHERE time_id BETWEEN '20-DEC-01' AND '22-DEC-01' AND prod_id BETWEEN 120 AND 125
  GROUP BY time_id, prod_id;

    TIME_ID      PROD_ID   LOWEST_UNIT_COST
    --------- ----------   -----------------------------------
    20-DEC-01         121    9.11;9.27;15.84;43.95
    20-DEC-01         122    9.11;9.27;15.84;43.95
    20-DEC-01         123    9.11;9.27;15.84;43.95
    21-DEC-01         120    9.11;9.27
    21-DEC-01         121    9.11;9.27
    22-DEC-01         120    9.11;9.27;15.84;43.95;16.06;12.66
    22-DEC-01         121    9.11;9.27;15.84;43.95;16.06;12.66
    22-DEC-01         122    9.11;9.27;15.84;43.95;16.06;12.66
    22-DEC-01         123    9.11;9.27;15.84;43.95;16.06;12.66
    22-DEC-01         124    9.11;9.27;15.84;43.95;16.06;12.66
    22-DEC-01         125    9.11;9.27;15.84;43.95;16.06;12.66
```

19.3.3 FIRST/LAST関数

FIRST/LAST集計関数を使用すると、データセットをランク付けした上で、最上位または最下位の行に対する処理ができます。最上位または最下位の行が発見された後には、目的とする任意の列に対して集計関数が適用されます。つまり、FIRST/LASTでは、列Aを対象としてランク付けした結果、最上位または最下位となった行の、列Bにおける値に対する集計の結果が戻されます。この場合、自己結合や副問合せが不要になり、より高いパフォーマンスが得られるので便利です。これらの関数の構文としては、各グループに対し1つの戻り値を生成する通常の集計関数(MIN、MAX、SUM、AVG、COUNT、VARIANCE、STDDEV)を最初に記述します。その上で、FIRST/LAST関数で使用するランキングを指定するため、KEEPで始まる新しい句を追加します。

これらの関数の構文は次のとおりです。

```
aggregate_function KEEP ( DENSE_RANK FIRST | LAST ORDER BY
  expr [ DESC | ASC ] [NULLS { FIRST | LAST }]
  [, expr [ DESC | ASC ] [NULLS { FIRST | LAST }]]...)
[OVER query_partitioning_clause]
```

ORDER BY句には、複数の式を指定できます。

この項では、次の項目について説明します。

- 通常の集計としてのFIRSTおよびLAST
- 集計レポートとしてのFIRSTおよびLAST

19.3.3.1 通常の集計としてのFIRSTおよびLAST

FIRST/LAST集計ファミリは、通常の集計関数として使用できます。

例19-22 FIRST/LASTの例1

次に示す問合せでは、製品の最低価格と定価を比較できます。メンズ・ウェア・カテゴリにおける製品サブカテゴリごとに、次の内容が戻されます。

- 最低価格が最も低い製品の定価
- 最も低い最低価格
- 最低価格が最も高い製品の定価
- 最も高い最低価格

```
SELECT prod_subcategory, MIN(prod_list_price)
  KEEP (DENSE_RANK FIRST ORDER BY (prod_min_price)) AS LP_OF_LO_MINP,
MIN(prod_min_price) AS LO_MINP,
MAX(prod_list_price) KEEP (DENSE_RANK LAST ORDER BY (prod_min_price))
 AS LP_OF_HI_MINP,
MAX(prod_min_price) AS HI_MINP
FROM products WHERE prod_category='Electronics'
GROUP BY prod_subcategory;

PROD_SUBCATEGORY LP_OF_LO_MINP LO_MINP  LP_OF_HI_MINP  HI_MINP
---------------- ------------- -------  ------------- ----------
Game Consoles          299.99  299.99          299.99     299.99
Home Audio             499.99  499.99          599.99     599.99
Y Box Accessories        7.99    7.99           20.99      20.99
Y Box Games              7.99    7.99           29.99      29.99
```

19.3.3.2 集計レポートとしてのFIRSTおよびLAST

FIRST/LAST集計ファミリは、集計レポート関数としても使用できます。たとえば、人数の増加が年間で最大の月や最小の月の計算などができます。これらの関数の構文は、他の集計レポートの構文と似ています。

例19-22のFIRST/LASTの例で考えてみます。この場合、「個々の製品の定価を検索し、それをサブカテゴリ内で最低価格が最も高い製品および最も低い製品の定価と比較するとどうなるか」という問題になります。

FIRST/LASTを集計レポートとして使用し、こうした情報をドキュメンテーション(Documentation)というサブカテゴリに関して検索する問合せを次に示します。

例19-23 FIRST/LASTの例2

```
SELECT prod_id, prod_list_price,
    MIN(prod_list_price) KEEP (DENSE_RANK FIRST ORDER BY (prod_min_price))
       OVER(PARTITION BY (prod_subcategory)) AS LP_OF_LO_MINP,
    MAX(prod_list_price) KEEP (DENSE_RANK LAST ORDER BY (prod_min_price))
       OVER(PARTITION BY (prod_subcategory)) AS LP_OF_HI_MINP
FROM products WHERE prod_subcategory = 'Documentation';

   PROD_ID PROD_LIST_PRICE LP_OF_LO_MINP LP_OF_HI_MINP
---------- --------------- ------------- -------------
        40           44.99         44.99         44.99
        41           44.99         44.99         44.99
        42           44.99         44.99         44.99
        43           44.99         44.99         44.99
        44           44.99         44.99         44.99
        45           44.99         44.99         44.99
```

FIRSTおよびLAST関数を集計レポートとして使用すると、その結果を簡単に「最高給与に対する給与の割合」などの計算に簡単に組み込むことができます。

19.3.4 逆パーセンタイル関数

値の集合の累積分布(パーセンタイル)を求めるには、CUME_DIST関数を使用できます。ただし、逆の操作(特定のパーセンタイルを計算するための値の検索)は、簡単でも効率的でもありません。これを簡単に操作できるように、PERCENTILE_CONT関数とPERCENTILE_DISC関数が導入されました。この2つの関数は、通常の集計関数としてのみでなく、ウィンドウおよびレポート関数としても使用できます。

これらの関数では、ソートの指定と、0から1のパーセンタイル値をとるパラメータが必要とされます。ソート指定の処理には、ORDER BY句で1つの式を使用します。通常の集計関数として使用すると、ソートされた集合ごとに単一の値が戻されます。

PERCENTILE_CONTは内挿法により計算される連続関数で、PERCENTILE_DISCは不連続値を想定するステップ関数です。PERCENTILE_CONTおよびPERCENTILE_DISCは、他の集計と同様、グルーピングされた問合せの行グループを対象として操作する関数ですが、次のような違いがあります。

- 0から1までの値をとるパラメータが必要です。パラメータにこの範囲外の値を指定した場合、エラーとなります。このパラメータには、定数に評価される式を指定する必要があります。
- ソート指定が必要です。ソートは、ORDER BY句に単一の式を指定する形で指定します。複数の式は使用できません。

Oracle Database 12cリリース2 (12.2)以降では、近似逆分布関数APPROX_PERCENTILEは、そのソート指定に従ってそのパーセンタイル値に該当する近似の補間された値を戻します。

関連項目:

- 通常の集計の構文
- 逆パーセンタイルの例
- 集計レポートとしての使用
- 逆パーセンタイル関数に関する制限
- 近似結果を返すパーセンタイル関数の使用

19.3.4.1 通常の集計の構文

```
[PERCENTILE_CONT | PERCENTILE_DISC]( constant expression )
    WITHIN GROUP ( ORDER BY single order by expression
[ASC|DESC] [NULLS FIRST| NULLS LAST])
```

19.3.4.2 逆パーセンタイルの例

次の問合せを使用して、この項の例で使用したデータのうち17行を戻します。

```
SELECT cust_id, cust_credit_limit, CUME_DIST()
   OVER (ORDER BY cust_credit_limit) AS CUME_DIST
FROM customers WHERE cust_city='Marshal';

   CUST_ID CUST_CREDIT_LIMIT  CUME_DIST
---------- ----------------- ----------
     28344              1500 .173913043
      8962              1500 .173913043
     36651              1500 .173913043
     32497              1500 .173913043
     15192              3000 .347826087
    102077              3000 .347826087
    102343              3000 .347826087
      8270              3000 .347826087
     21380              5000  .52173913
     13808              5000  .52173913
    101784              5000  .52173913
     30420              5000  .52173913
     10346              7000 .652173913
     31112              7000 .652173913
     35266              7000 .652173913
      3424              9000 .739130435
    100977              9000 .739130435
    103066             10000 .782608696
     35225             11000 .956521739
     14459             11000 .956521739
     17268             11000 .956521739
    100421             11000 .956521739
     41496             15000          1
```

PERCENTILE_DISC(x)の計算では、x以上の値が最初に見つかるまで、各グループ内のCUME_DIST値がスキャンされます。xは、指定した百分位数です。このサンプルの問合せではPERCENTILE_DISC(0.5)となり、結果は次のように5,000となります。

```
SELECT PERCENTILE_DISC(0.5) WITHIN GROUP
  (ORDER BY cust_credit_limit) AS perc_disc, PERCENTILE_CONT(0.5) WITHIN GROUP
  (ORDER BY cust_credit_limit) AS perc_cont
 FROM customers WHERE cust_city='Marshal';

PERC_DISC   PERC_CONT
---------   ---------
     5000        5000
```

PERCENTILE_CONTの計算では、順序付けを行った後に、行と行の線形内挿法によって結果が導かれます。
PERCENTILE_CONT(x)を計算するため、まず行番号=RN=(1+x*(n-1))が計算されます。*n*はグループ内の行数、*x*は指定した百分位数です。さらに、行番号CRN = CEIL(RN)およびFRN = FLOOR(RN)の行の値の線形内挿法により、この集計関数の最終結果が計算されます。

最終結果は、PERCENTILE_CONT(X) =(CRN = FRN = RN)の場合は(行RNの式の値)、それ以外の場合は(CRN -

RN)*(行FRNの式の値)+(RN -FRN)*(行CRNの式の値)となります。

前述の問合せの例で、PERCENTILE_CONT(0.5)を計算する場合を考えてみます。ここで、*n*は17です。どちらのグループも、行番号はRN = (1 + 0.5*(n-1))= 9となります。これを式に当てはめると(FRN=CRN=9)となり、結果として行9からの値が戻されます。

別の例として、PERCENTILE_CONT(0.66)を計算する場合を考えてみます。行番号は、RN=(1 + 0.66*(n-1))= (1 + 0.66*16)= 11.67と計算されます。その結果、PERCENTILE_CONT(0.66) = (12-11.67)*(行11の値)+(11.67-11)*(行12の値)となります。結果は次のようになります。

```
SELECT PERCENTILE_DISC(0.66) WITHIN GROUP
  (ORDER BY cust_credit_limit) AS perc_disc, PERCENTILE_CONT(0.66) WITHIN GROUP
  (ORDER BY cust_credit_limit) AS perc_cont
FROM customers WHERE cust_city='Marshal';

 PERC_DISC  PERC_CONT
---------- ----------
      9000       8040
```

逆パーセンタイル関数は、他の既存の集計関数と同様に問合せのHAVING句に使用できます。

19.3.4.3 集計レポートとしての使用

集計関数PERCENTILE_CONTおよびPERCENTILE_DISCは、集計レポート関数としても使用できます。その場合の構文は、他の集計レポートの場合と同様です。

```
[PERCENTILE_CONT | PERCENTILE_DISC](constant expression)
WITHIN GROUP ( ORDER BY single order by expression
[ASC|DESC] [NULLS FIRST| NULLS LAST])
OVER ( [PARTITION BY value expression [,...]] )
```

次の問合せでは同じ内容(この結果セットに含まれる顧客の与信限度額の中央値)が計算されますが、後述の出力のように結果セットの各行について結果がレポートされます。

```
SELECT cust_id, cust_credit_limit, PERCENTILE_DISC(0.5) WITHIN GROUP
     (ORDER BY cust_credit_limit) OVER () AS perc_disc,
  PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY cust_credit_limit)
  OVER () AS perc_cont
FROM customers WHERE cust_city='Marshal';

   CUST_ID CUST_CREDIT_LIMIT  PERC_DISC  PERC_CONT
---------- ----------------- ---------- ----------
     28344              1500       5000       5000
      8962              1500       5000       5000
     36651              1500       5000       5000
     32497              1500       5000       5000
     15192              3000       5000       5000
    102077              3000       5000       5000
    102343              3000       5000       5000
      8270              3000       5000       5000
     21380              5000       5000       5000
     13808              5000       5000       5000
    101784              5000       5000       5000
     30420              5000       5000       5000
     10346              7000       5000       5000
     31112              7000       5000       5000
     35266              7000       5000       5000
      3424              9000       5000       5000
    100977              9000       5000       5000
```

103066	10000	5000	5000
35225	11000	5000	5000
14459	11000	5000	5000
17268	11000	5000	5000
100421	11000	5000	5000
41496	15000	5000	5000

19.3.4.4 逆パーセンタイル関数の制限

PERCENTILE_DISCの場合は、ORDER BY句の式に、ソート可能なデータ型(数値、文字列、日付など)を使用できます。一方、PERCENTILE_CONTの評価には線形内挿法が使用されるため、ORDER BY句の式には数値型または日付/時刻型(インターバルを含む)を指定する必要があります。式がDATE型の場合、内挿の結果はその型の最小単位に丸められます。DATE型の場合、内挿された値は最も近い秒に丸められ、インターバル型の場合は最も近い秒(INTERVAL DAY TO SECOND)または月(INTERVAL YEAR TO MONTH)に丸められます。

他の集計と同様に、逆パーセンタイル関数でも結果の評価時にNULLは無視されます。たとえば、集合内の中央値を求める場合、Oracle DatabaseではNULLが無視され、NULL以外の値から中央値が求められます。ORDER BY句にはNULLS FIRST/NULLS LASTオプションも使用できますが、NULLが無視されるので、こうしたオプションも結果的には無視されます。

19.3.4.5 近似結果を返すパーセンタイル関数の使用

Oracle Databaseには、近似パーセンタイル結果を返すSQL関数のセットが用意されています。これらの関数を使用して、品質の監視、ソーシャル・メディア・アクティビティの追跡、パフォーマンスの監視およびデータ・セット内の外れ値の検索を行うことができます。

次のSQL関数は、近似パーセンタイル結果を計算および表示します。

- APPROX_PERCENTILE

  ソート指定に従ってそのパーセンタイル値に該当する補間された近似値を返します。これは、正確な結果とわずかに誤差がありますが、PERCENTILE_CONTよりはるかに高速に大量のデータを処理できます。

- APPROX_PERCENTILE_DETAIL

  GROUP BY句を使用して指定された一連のデータ内の、詳細と呼ばれる近似パーセンタイル情報を計算します。この関数で作成された詳細情報はバイナリ形式で格納されており、TO_APPROX_PERCENTILEとAPPROX_PERCENT_DETAIL_AGGの両方の関数で使用されることを想定しています。

- APPROX_PERCENTILE_AGG

  APPROX_PERCENTILE_DETAIL関数を使用して作成された詳細の集計を実行します。

- TO_APPROX_PECENTILE

  詳細または集計の結果(BLOB値として格納されている)を判読可能な形式で表示します。

詳細および高レベルの集計データは、その後の分析のために表またはマテリアライズド・ビューに格納できます。

例: 国または州内の近似パーセンタイル販売データの表示

この例では、APPROX_PERCENTILE_DETAILを使用して、パーセンタイル計算を1回実行し、この結果を表に格納し、格納されたデータに基づいて近似集計を実行します。TO_APPROX_PERCENTILE関数は、パーセンタイル計算の結果を判読可能な形式で表示するために使用されます。

1. APPROX_PERCENTILE_DETAILを使用して、各州の販売額の近似パーセンタイルを計算し、approx_sales_percentile_detailと呼ばれる表にこの結果を格納します。

```
CREATE TABLE approx_sales_percentile_detail AS
```

```
SELECT c.country_id country, c.cust_state_province state,
approx_percentile_detail(amount_sold) detail
FROM sales s, customers c
WHERE s.cust_id = c.cust_id
GROUP BY c.country_id, c.cust_state_province;
```

2. TO_APPROX_PERCENTILEを使用して、詳細を問い合せ、表に格納されている値を集計してこれらの値を判読可能な形式で表示します。

   次の文は、APPROX_PERCENTILE_AGG関数を使用して、approx_sales_percentile_detail表に格納されている詳細データをさらに集計します。TO_APPROX_PERCENTILE関数は、集計結果を判読可能な形式で表示します。

```
SELECT country, to_approx_percentile(approx_percentile_agg(detail),0.5) median_amt_sold
FROM approx_sales_percentile_detail
GROUP BY country
ORDER BY country;

   COUNTRY   MEDIAN_AMT_SOLD
----------    ---------------
     52769            33.5
     52770           35.92
     52771           44.99
     52772           35.55
     52773           29.61
     52774           35.55
     52775           42.09
     52776           34.67
     52777            38.1
     52778           38.35
     52779           38.67
     52782           36.89
     52785           22.99
     52786           44.99
     52787           27.99
     52788           27.13
     52789           37.79
     52790           33.69

18 rows selected.
```

関連項目:

Oracle Database SQL言語リファレンスのAPROX_PERCENTILE、APPROX_PERCENTILE_DETAIL、APPROX_PERCENTILE_AGGおよびTO_APPROX_PERCENTILE

19.3.5 仮説ランク関数

この種の関数では、what-if分析に役立つ機能が提供されます。たとえば、行を他の行集合に仮に挿入すると、行のランクがどうなるかという問題があるとします。

この集計ファミリでは、仮に挿入する行と順序付けられた行のグループを引数として1つ以上取り、対象とするグループに行を仮に挿入した場合の、その行のRANK、DENSE_RANK、PERCENT_RANKまたはCUME_DISTが戻されます。

```
[RANK | DENSE_RANK | PERCENT_RANK | CUME_DIST]( constant expression [, ...] )
```

```
WITHIN GROUP ( ORDER BY order by expression [ASC|DESC] [NULLS FIRST|NULLS LAST][, ...] )
```

ここで、*constant expression*は定数に評価される式です。このような式を引数として複数個、関数に渡す場合もあります。ORDER BY句には、ランキングの基準となるソート順を定義する式を1つ以上含めることができます。ORDER BY句内の各式には、ASC、DESC、NULLS FIRST、NULLS LASTオプションを使用できます。

例19-24 仮説ランク関数および仮説分布関数の例1

この項で使用されているproducts表の定価データを使用して、価格50ドルのセーターの仮のRANK、PERCENT_RANKおよびCUME_DISTを計算し、各セーター・サブカテゴリに当てはまるかどうかを調べる場合を考えます。この場合の問合せと結果は次のようになります。

```
SELECT cust_city,
  RANK(6000) WITHIN GROUP (ORDER BY CUST_CREDIT_LIMIT DESC) AS HRANK,
  TO_CHAR(PERCENT_RANK(6000) WITHIN GROUP
  (ORDER BY cust_credit_limit),'9.999') AS HPERC_RANK,
  TO_CHAR(CUME_DIST (6000) WITHIN GROUP
  (ORDER BY cust_credit_limit),'9.999') AS HCUME_DIST
FROM customers
WHERE cust_city LIKE 'Fo%'
GROUP BY cust_city;

CUST_CITY                          HRANK HPERC_ HCUME_
------------------------------ ---------- ------ ------
Fondettes                              13   .455   .478
Fords Prairie                          18   .320   .346
Forest City                            47   .370   .378
Forest Heights                         38   .456   .464
Forestville                            58   .412   .418
Forrestcity                            51   .438   .444
Fort Klamath                           59   .356   .363
Fort William                           30   .500   .508
Foxborough                             52   .414   .420
```

逆パーセンタイル集計とは異なり、仮説ランク関数および仮説分布関数の場合、ソート指定のORDER BY句には複数の式を使用できます。ORDER BY句の式の数と引数の数は同じにする必要があり、引数は、対応するORDER BY句の式と同じ型または互換性のある型の定数式にする必要があります。次の例では、複数の仮説ランク関数において2つの引数が使用されています。

例19-25 仮説ランク関数および仮説分布関数の例2

```
SELECT prod_subcategory,
  RANK(10,8) WITHIN GROUP (ORDER BY prod_list_price DESC,prod_min_price)
  AS HRANK, TO_CHAR(PERCENT_RANK(10,8) WITHIN GROUP
  (ORDER BY prod_list_price, prod_min_price),'9.999') AS HPERC_RANK,
  TO_CHAR(CUME_DIST (10,8) WITHIN GROUP
  (ORDER BY prod_list_price, prod_min_price),'9.999') AS HCUME_DIST
FROM products WHERE prod_subcategory LIKE 'Recordable%'
GROUP BY prod_subcategory;

PROD_SUBCATEGORY          HRANK   HPERC_   HCUME_
--------------------      -----   ------   ------
Recordable CDs                4     .571     .625
Recordable DVD Discs          5     .200     .333
```

これらの関数は、他の集計関数と同様に問合せのHAVING句に使用できます。集計レポート関数や集計ウィンドウ関数としては使用できません。

19.3.6 線形回帰関数

この回帰関数では、数値の組の集合に対する微分最小2乗法で求めた回帰直線の適合がサポートされます。この関数は、集計関数としても、ウィンドウ関数やレポート関数としても使用できます。

回帰関数には次のようなものがあります。

- REGR_COUNT関数
- REGR_AVGY関数およびREGR_AVGX関数
- REGR_SLOPE関数およびREGR_INTERCEPT関数
- REGR_R2関数
- REGR_SXX関数、REGR_SYY関数およびREGR_SXY関数

Oracleではこの種の関数は、(e1, e2)の組の集合に対して、e1またはe2のいずれかがNULLである組をすべて排除した後に適用されます。e1は従属変数の値(y)として、e2は独立変数(x)として解釈されます。どちらの式も、数値である必要があります。

すべての回帰関数は、データからの単一パスの間に同時に計算されます。こうした回帰関数は、多くの場合COVAR_POP、COVAR_SAMPおよびCORR関数と組み合せて使用されます。

関連項目:

- 線形回帰統計の例
- 線形回帰計算の例

19.3.6.1 REGR_COUNT関数

REGR_COUNTでは、回帰直線の適合に使用されるNULL以外の数値の組の数が戻されます。この関数を空の集合に適用した場合(または、e1およびe2の両方がNULLでない(e1, e2)の組が存在しない場合)、0(ゼロ)が戻ります。

19.3.6.2 REGR_AVGY関数およびREGR_AVGX関数

REGR_AVGYでは回帰直線の従属変数の平均、REGR_AVGXでは回帰直線の独立変数の平均が計算されます。REGR_AVGYでは、(e1, e2)の組のうちe1またはe2のいずれかがNULLであるものが排除され、その後で第1の引数(e1)の平均が計算されます。同様に、REGR_AVGXでは、NULLが排除された後に第2の引数(e2)の平均が計算されます。どちらの関数も、空の集合に適用した場合にはNULLが戻ります。

19.3.6.3 REGR_SLOPE関数およびREGR_INTERCEPT関数

REGR_SLOPE関数では、NULL以外の(e1, e2)の組に適合する回帰直線の傾きが計算されます。

REGR_INTERCEPT関数では、回帰直線のy切片が計算されます。傾きまたは回帰平均がNULLの場合、REGR_INTERCEPTからはNULLが戻ります。

19.3.6.4 REGR_R2関数

REGR_R2関数では、回帰直線の確定係数(通常は「Rの2乗」または「適合度」)が計算されます。

回帰直線が定義される場合(線の傾きがNULLではない場合)、REGR_R2からは0(ゼロ)から1の値が戻ります。それ以外の場合はNULLが戻ります。値が1に近づくほど、回帰直線がデータに適合します。

19.3.6.5 REGR_SXX関数、REGR_SYY関数およびREGR_SXY関数

REGR_SXX関数、REGR_SYY関数およびREGR_SXY関数は、回帰分析用の様々な診断統計情報の計算に使用されます。これらの関数では、(e1, e2)の組のうちe1またはe2のいずれかがNULLであるものが排除された後で、次の計算が実行されます。

```
REGR_SXX:    REGR_COUNT(e1,e2) * VAR_POP(e2)
REGR_SYY:    REGR_COUNT(e1,e2) * VAR_POP(e1)
REGR_SXY:    REGR_COUNT(e1,e2) * COVAR_POP(e1, e2)
```

19.3.6.6 線形回帰統計の例

線形回帰分析を伴う一般的な診断統計の一部を表19-2に示します。Oracleにより、これらすべてを計算できます。

表19-2 一般的な診断統計情報およびその式

統計情報のタイプ	式
調整 R2	1-((1 - REGR_R2)*((REGR_COUNT-1)/(REGR_COUNT-2)))
標準誤差	SQRT((REGR_SYY-(POWER(REGR_SXY,2)/REGR_SXX))/(REGR_COUNT-2))
2 乗の総計	REGR_SYY
2 乗の回帰合計	POWER(REGR_SXY,2) / REGR_SXX
2 乗の残差合計	REGR_SYY - (POWER(REGR_SXY,2)/REGR_SXX)
傾きの t 統計	REGR_SLOPE * SQRT(REGR_SXX) / (標準誤差)
y 切片の t 統計量	REGR_INTERCEPT / ((標準誤差)* SQRT((1/REGR_COUNT)+(POWER(REGR_AVGX,2)/REGR_SXX))

19.3.6.7 線形回帰計算の例

この例では、製品の販売数量をその製品の定価の線形関数として表す、微分最小2乗法で求めた回帰直線を計算します。計算は、販売チャネル別にグルーピングされます。SLOPE、INTCPTおよびRSQRの値は、それぞれ、回帰直線の傾き、切片および確定係数です。(整数)値COUNTは、販売数量データと定価データの両方を使用できる各チャネルの製品の数量です。

```
SELECT s.channel_id, REGR_SLOPE(s.quantity_sold, p.prod_list_price) SLOPE,
  REGR_INTERCEPT(s.quantity_sold, p.prod_list_price) INTCPT,
  REGR_R2(s.quantity_sold, p.prod_list_price) RSQR,
  REGR_COUNT(s.quantity_sold, p.prod_list_price) COUNT,
  REGR_AVGX(s.quantity_sold, p.prod_list_price) AVGLISTP,
  REGR_AVGY(s.quantity_sold, p.prod_list_price) AVGQSOLD
FROM sales s, products p WHERE s.prod_id=p.prod_id
  AND p.prod_category='Electronics' AND s.time_id=to_DATE('10-OCT-2000')
GROUP BY s.channel_id;

CHANNEL_ID      SLOPE     INTCPT       RSQR      COUNT    AVGLISTP   AVGQSOLD
---------- ---------- ---------- ---------- ---------- ---------- ----------
         2          0          1          1         39 466.656667          1
         3          0          1          1         60     459.99          1
         4          0          1          1         19 526.305789          1
```

19.3.7 統計集計について

Oracle Databaseには、一連のSQL統計関数と統計パッケージDBMS_STAT_FUNCSが用意されています。この項では、新しい関数の一部を基本的な構文とともに示します。

DBMS_STAT_FUNCSパッケージの詳細は『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』を、構文およびセマンティックについては『*Oracle Database SQL言語リファレンス*』を参照してください。

この項では、次の項目について説明します。

- 記述統計情報
- 仮説の検証 - パラメトリック検定
- クロス集計統計情報
- 仮説の検証 - ノンパラメトリック検定
- ノンパラメトリック相関

19.3.7.1 記述統計情報

次の記述統計情報を計算できます。

- データセットの中央値

```
Median (expr) [OVER (query_partition_clause)]
```

- データセットのモード

```
STATS_MODE (expr)
```

Oracle Database 12cリリース2 (12.2)以降では、近似逆分布関数APPROX_MEDIANは、指定した式の近似中央値を返します。

関連項目:

Oracle Database SQL言語リファレンス

19.3.7.2 仮説の検証 - パラメトリック検定

次の記述統計情報を計算できます。

- 1標本のT-検定

```
STATS_T_TEST_ONE (expr1, expr2 (a constant) [, return_value])
```

- 対標本のT-検定

```
STATS_T_TEST_PAIRED (expr1, expr2 [, return_value])
```

- 独立標本のT-検定。併合分散

```
STATS_T_TEST_INDEP (expr1, expr2 [, return_value])
```

- 独立標本のt-検定。非併合分散

```
STATS_T_TEST_INDEPU (expr1, expr2 [, return_value])
```

- F-検定

```
STATS_F_TEST (expr1, expr2 [, return_value])
```

- 1方向ANOVA

```
STATS_ONE_WAY_ANOVA (expr1, expr2 [, return_value])
```

19.3.7.3 クロス集計統計情報

次の構文を使用して、クロス集計統計情報を計算できます。

```
STATS_CROSSTAB (expr1, expr2 [, return_value])
```

次のいずれかの値が戻されます。

- カイ2乗の測定値
- カイ2乗の測定値の有意性
- カイ2乗の自由度
- ファイ係数、クラメールのV統計
- 一致係数
- コーエンのカッパ

19.3.7.4 仮説の検証 - ノンパラメトリック検定

次の構文を使用して、仮説統計情報を計算できます。

```
STATS_BINOMIAL_TEST (expr1, expr2, p [, return_value])
```

- 2項検定/ウィルコクソンの符号付き順位検定

```
STATS_WSR_TEST (expr1, expr2 [, return_value])
```

- マン・ホイットニー検定

```
STATS_MW_TEST (expr1, expr2 [, return_value])
```

- コルモゴロフ・スミルノフ検定

```
STATS_KS_TEST (expr1, expr2 [, return_value])
```

19.3.7.5 ノンパラメトリック相関

次のパラメトリック統計情報を計算できます。

- スピアマンのロー係数

```
CORR_S (expr1, expr2 [, return_value])
```

- ケンドールのタウb係数

```
CORR_K (expr1, expr2 [, return_value])
```

これらの関数に加え、今回のリリースにはPL/SQLパッケージDBMS_STAT_FUNCSも用意されています。このパッケージには、分布適合をサポートする関数とともに、記述統計関数SUMMARYが含まれています。SUMMARY関数は、様々な記述統計情報を含む表の数値列を集計します。分布適合関数は5つあり、それぞれ正規分布、一様分布、ワイブル分布、ポアソン分布、指数分

布をサポートします。

19.3.8 ユーザー定義集計について

Oracleでは、ユーザー定義集計関数と呼ばれる独自の関数を作成できます。この種の関数は、PL/SQL、JavaおよびCなどのプログラミング言語で記述し、マテリアライズド・ビューで分析関数または集計関数として使用できます。構文および制限の詳細は、『*Oracle Databaseデータ・カートリッジ開発者ガイド*』を参照してください。

この種の関数のメリットを次に示します。

- きわめて複雑な関数を完全に手続き型言語でプログラミングできます。
- ユーザー定義関数をパラレル処理用にプログラミングする場合に、他のテクニックに比べて高いスケーラビリティが得られます。
- オブジェクトのデータ型を処理できます。

ユーザー定義集計関数の単純な例として、偏りの統計を考えてみます。この計算では、データセットの分散が平均に対して一方に偏っているかどうかが計算されます。分散の一方の最後尾が他方より極端に大きいかどうかが示されます。ユーザー定義集計関数udskewを作成し、前述の例の与信限度額データに適用した場合、SQL文とその結果は次のようになります。

```
SELECT USERDEF_SKEW(cust_credit_limit) FROM customers
WHERE cust_city='Marshal';

USERDEF_SKEW
============
0.583891
```

ユーザー定義集計関数を作成する前に、ニーズを通常のSQLで満たすことができるかどうかを考慮する必要があります。SQLでは、特にCASE式を使用すれば、多数の複雑な計算を直接行うことができます。

通常のSQLを使用しても開発を簡素化することは可能です。また、SQLでは多くの問合せ操作がすでに適切にパラレル化されています。前述の例でも、偏りの統計は、長くはなりますが標準的なSQLを使用して作成できます。

19.4 ピボット操作

ビジネス・インテリジェンス問合せから戻されたデータは、多くの場合、クロス集計形式で表されていると最も便利に使用できます。SELECT文においてpivot_clauseを使用すると、行を回転して列とするクロス集計の問合せを記述し、回転のプロセスでデータを集計できます。ピボット操作はデータ・ウェアハウスにおける重要なテクニックです。この操作では、入力する複数行がデータ・ウェアハウス内でより少数の行や、より多数の行(通常の場合)に変換されます。ピボット操作時には、ピボット列の値リストにある各項目に集計演算子が適用されます。ピボット列には不定的な式を含めることはできません。式に対するピボットが必要な場合、ビュー内でその式に別名を与えてからPIVOT操作を行います。基本的な構文は次のようになります。

```
SELECT ....
FROM <table-expr>
   PIVOT
     (
      aggregate-function(<column>) AS <alias>
      FOR <pivot-column> IN (<value1>, <value2>,..., <valuen>)
        ) AS <alias>
WHERE .....
```

pivot_clause構文については、『Oracle Database SQL言語リファレンス』を参照してください。

この項では、次の項目について説明します。

- ピボットの例で使用するビューの作成
- ピボットの例
- 複数列に対するピボット操作
- ピボット操作: 複数の集計
- ソース・データ内のNULLとPIVOTで生成されたNULLとの識別
- ワイルド・カードおよび副問合せのXML操作によるピボット操作

19.4.1 ピボットの例に使用するビューの作成

ピボットとアンピボットの例は、sales_viewビューに基づいています。

例19-26 ピボットの例のためのSALES_VIEWビューの作成

次の例は、ピボットの使用を示すためのベースとして使用するsales_viewビューを作成します。

```
CREATE VIEW sales_view AS
SELECT
 prod_name product, country_name country, channel_id channel,
 SUBSTR(calendar_quarter_desc, 6,2) quarter,
 SUM(amount_sold) amount_sold, SUM(quantity_sold) quantity_sold
FROM sales, times, customers, countries, products
WHERE sales.time_id = times.time_id AND
  sales.prod_id = products.prod_id AND
  sales.cust_id = customers.cust_id AND
  customers.country_id = countries.country_id
GROUP BY prod_name, country_name, channel_id,
 SUBSTR(calendar_quarter_desc, 6, 2);
```

19.4.2 ピボットの例

次の文は、例19-26の説明に従って作成したビューsales_viewのchannel列に対する一般的なピボットを示しています。

```
SELECT * FROM
  (SELECT product, channel, amount_sold
   FROM sales_view
   ) S PIVOT (SUM(amount_sold)
   FOR CHANNEL IN (3 AS DIRECT_SALES, 4 AS INTERNET_SALES,
                   5 AS CATALOG_SALES, 9 AS TELESALES))
ORDER BY product;

PRODUCT                   DIRECT_SALES  INTERNET_SALES  CATALOG_SALES  TELESALES
-----------------------   ------------  --------------  -------------  ---------
...
Internal 6X CD-ROM           229512.97        26249.55
Internal 8X CD-ROM           286291.49        42809.44
Keyboard Wrist Rest          200959.84        38695.36                   1522.73
...
```

出力では、ピボット値それぞれについて、DIRECT_SALES、INTERNET_SALES、CATALOG_SALESおよびTELESALESという別名の付いた4つの列が新しく作成されています。この出力には合計が表示されます。別名を付けない場合、列ヘッダーはINリストの値となります。

19.4.3 複数列に対するピボット操作

ピボットは複数の列に対して実行できます。次の文は、例19-26の説明に従って作成したビューsales_viewに対する一般的な複数列のピボットを示しています。

```
SELECT *
FROM
     (SELECT product, channel, quarter, quantity_sold
      FROM sales_view
     ) PIVOT (SUM(quantity_sold)
                FOR (channel, quarter) IN
                  ((5, '02') AS CATALOG_Q2,
                   (4, '01') AS INTERNET_Q1,
                   (4, '04') AS INTERNET_Q4,
                   (2, '02') AS PARTNERS_Q2,
                   (9, '03') AS TELE_Q3
                  )
               );

PRODUCT              CATALOG_Q2  INTERNET_Q1  INTERNET_Q4  PARTNERS_Q2   TELE_Q3
-------              ----------  -----------  -----------  -----------   -------
...
Bounce                                  347          632          954
...
Smash Up Boxing                         129          280          560
...
Comic Book Heroes                        47          155          275
...
```

この例では、複数列のINリストが指定され、そのINリストのメンバーに一致するような列ヘッダーが与えられています。

19.4.4 ピボット操作: 複数の集計

次の例に示すように、複数の集計を使用してピボットを実行できます。この例では、例19-26で作成したsales_viewから複数の集計に対するピボットを実行しています。

```
SELECT *
FROM
     (SELECT product, channel, amount_sold, quantity_sold
      FROM sales_view
     ) PIVOT (SUM(amount_sold) AS sums,
              SUM(quantity_sold) AS sumq
              FOR channel IN (5, 4, 2, 9)
               )
ORDER BY product;

PRODUCT               5_SUMS  5_SUMQ    4_SUMS   4_SUMQ      2_SUMS   2_SUMQ    9_SUMS   9_SUMQ
--------------        ------  ------    ------   ------      ------   ------    ------   ------
O/S Doc Set English           142780.36     3081   381397.99     8044   6028.66      134
O/S Doc Set French             55503.58     1192   132000.77     2782
...
```

この問合せでは、ピボットの値、アンダースコア文字および集計列の別名を連結することにより、列ヘッダーが作成されます。生成された列見出しの長さが列名の最大長より長い場合、ORA-00918エラーが戻されます。このエラーを回避するには、AS *alias*を使用して、ピボット列見出しや集計値列の名前、またはその両方のためのより短い列の別名を指定します。ピボット値への別名使用については、複数列に対するピボット操作で説明しています。

関連項目:

列名の最大長に関する詳細は、『*Oracle Database SQL*言語リファレンス』を参照してください。

19.4.5 ソース・データ内のNULLとPIVOTで生成されたNULLとの識別

ソース・データ内に存在するNULL値と、PIVOTを使用することで生成されたNULL値は、識別できます。次の例では、PIVOTによって生成されるNULLについて解説します。

次の問合せでは5列分の行が戻されます。列prod_idと、ピボット操作結果の列Q1、Q1_COUNT_TOTAL、Q2、Q2_COUNT_TOTALの5列です。prod_idの一意の各値について、Q1_COUNT_TOTALには行のqtrの値がQ1である行の合計数が戻されます。Q2_COUNT_TOTALには行qtrの値がQ2である行の合計数が戻されます。

ここでは、次の構造からなるsales2という表があると仮定します。

```
PROD_ID   QTR  AMOUNT_SOLD
-------   ---  -----------
100        Q1           10
100        Q1           20
100        Q2         NULL
200        Q1           50

SELECT *
FROM sales2
        PIVOT
         ( SUM(amount_sold), COUNT(*) AS count_total
           FOR qtr IN ('Q1', 'Q2')
         );

PROD_ID    "Q1"   "Q1_COUNT_TOTAL"  "Q2"       "Q2_COUNT_TOTAL"
-------    ----   ----------------  ---------  ----------------
    100      20                 2   NULL <1>                 1
    200      50                 1   NULL <2>                 0
```

この結果から、prod_idが100の場合、四半期(quarter)がQ1の売上行が2行、四半期がQ2の売上行が1行あること、prod_idが200の場合、四半期がQ1の売上行が1行、四半期がQ2の売上行はないことがわかります。したがって、Q2_COUNT_TOTALの内容から、NULL<1>は元の表内の行のうちメジャーがNULL値のものに由来し、NULL<2>は元の表内でprod_idが200の行が四半期Q2にはなかったために表示されていると識別できます。

19.4.6 ワイルド・カードおよび副問合せのXML操作によるピボット操作

ピボットの対象とする列でワイルド・カード引数または副問合せを使用するには、PIVOT XML構文を利用する方法があります。PIVOT XMLを使用した場合、操作の出力は適切に整形されたXMLになります。

次の例に、ワイルド・カード・キーワードANYを使用した場合を示します。この例では、sales_view内のチャネルの値をすべて含むXMLが出力されます。

```
SELECT *
FROM
   (SELECT product, channel, quantity_sold
    FROM sales_view
    ) PIVOT XML(SUM(quantity_sold)
                FOR channel IN (ANY)
               );
```

ビューsales_viewを作成する構文については、例19-26を参照してください。

キーワードANYは、PIVOT操作においてはXML操作の一部としてのみ利用できます。この出力には、データセット内にチャネルが存在する場合のデータが含まれます。また、複数の値を戻すには集計関数ではGROUP BY句を指定する必要がありますが、pivot_clauseでは明示的なGROUP BY句は含まれていません。かわりに、pivot_clauseでは暗黙的なGROUP BYを実行します。

次の例では、副問合せを使用する場合を示します。この例では、すべてのチャネルの値と、各チャネルに対応する売上データを含むXMLが出力されます。

```
SELECT *
FROM
     (SELECT product, channel, quantity_sold
      FROM sales_view
     ) PIVOT XML(SUM(quantity_sold)
                 FOR channel IN (SELECT DISTINCT channel_id FROM CHANNELS)
               );
```

この出力ではデータが稠密化され、各製品についてあり得るチャネルがすべて含まれる状態となります。

19.5 アンピボット操作

アンピボット操作では、PIVOT操作を元に戻しません。かわりに、データを列から行に回転します。ピボットされたデータを操作している場合にUNPIVOT操作を行っても、PIVOTまたはその他の手段で作成された集計は元に戻せません。

アンピボットを解説するため、まず、所定の年の四半期を示す4列からなる、ピボットされた表を作成します。次のコマンドは、例19-26で作成したビューsales_viewに基づく表を作成します。

```
CREATE TABLE pivotedTable AS
SELECT *
FROM (SELECT product, quarter, quantity_sold, amount_sold
      FROM sales_view)
    PIVOT
    (
       SUM(quantity_sold) AS sumq, SUM(amount_sold) AS suma
       FOR quarter IN ('01' AS Q1, '02' AS Q2, '03' AS Q3, '04' AS Q4));
```

表の内容は次のようになります。

```
SELECT *
FROM pivotedTable
ORDER BY product;

PRODUCT         Q1_SUMQ  Q1_SUMA  Q2_SUMQ  Q2_SUMA   Q3_SUMQ  Q3_SUMA   Q4_SUMQ    Q4_SUMA
--------------- -------  -------  -------  --------  -------  --------  -------    ---------
1.44MB External    6098   58301.33    5112   49001.56    6050   56974.3     5848     55341.28
128MB Memory       1963  110763.63    2361  132123.12    3069  170710.4     2832    157736.6
17" LCD            1492 1812786.94    1387 1672389.06    1591 1859987.66    1540   1844008.11
...
```

次のUNPIVOT操作では、一連の四半期(quarter)の列が回転されて行になります。各製品について、各四半期を表す4つの行が作成されます。

```
SELECT product, DECODE(quarter, 'Q1_SUMQ', 'Q1', 'Q2_SUMQ', 'Q2', 'Q3_SUMQ', 'Q3',
   'Q4_SUMQ', 'Q4') AS quarter, quantity_sold
FROM  pivotedTable
   UNPIVOT INCLUDE NULLS
       (quantity_sold
```

```
        FOR quarter IN (Q1_SUMQ, Q2_SUMQ, Q3_SUMQ, Q4_SUMQ))
ORDER BY product, quarter;

PRODUCT                          QU    QUANTITY_SOLD
-------                          --    -------------
1.44MB External 3.5" Diskette    Q1             6098
1.44MB External 3.5" Diskette    Q2             5112
1.44MB External 3.5" Diskette    Q3             6050
1.44MB External 3.5" Diskette    Q4             5848
128MB Memory Card                Q1             1963
128MB Memory Card                Q2             2361
128MB Memory Card                Q3             3069
128MB Memory Card                Q4             2832
...
```

この例ではINCLUDE NULLSが使用されています。デフォルト設定のEXCLUDE NULLSを使用することもできます。

さらに、次のように、2つの列を使用してアンピボット操作を行うことができます。

```
SELECT product, quarter, quantity_sold, amount_sold
FROM  pivotedTable
   UNPIVOT INCLUDE NULLS
       (
        (quantity_sold, amount_sold)
        FOR quarter IN ((Q1_SUMQ, Q1_SUMA) AS 'Q1', (Q2_SUMQ, Q2_SUMA) AS 'Q2', (Q3_SUMQ, Q3_SUMA) AS
'Q3', (Q4_SUMQ, Q4_SUMA) AS 'Q4'))
ORDER BY product, quarter;

PRODUCT                          QU   QUANTITY_SOLD   AMOUNT_SOLD
-------------------------------  --   -------------   ------------
1.44MB External 3.5" Diskette    Q1             6098       58301.33
1.44MB External 3.5" Diskette    Q2             5112       49001.56
1.44MB External 3.5" Diskette    Q3             6050       56974.3
1.44MB External 3.5" Diskette    Q4             5848       55341.28
128MB Memory Card                Q1             1963      110763.63
128MB Memory Card                Q2             2361      132123.12
128MB Memory Card                Q3             3069      170710.4
128MB Memory Card                Q4             2832      157736.6
```

19.6 レポート用のデータの稠密化

データは通常、スパースな形式で格納されています。つまり、ディメンション値の特定の組合せで値が存在しない場合、ファクト表には行が存在していません。しかし、ファクト・データが存在しない場合でも、ディメンション値のすべての組合せの行を表示し、データを稠密な形式で表示する必要がある場合があります。たとえば、特定の期間において製品が販売されていない場合でも、その期間の売上値をゼロとして製品の横に表示する場合などです。さらに、データが時間ディメンションに沿って稠密であれば、時系列の計算を簡単に実行できます。これは、稠密なデータが期間ごとに一定数の行を占めているため、物理オフセットを指定した分析ウィンドウ関数の使用が単純化されるためです。データの稠密化は、スパース・データを稠密な形式に変換するプロセスです。

スパース性の問題を解決するために、パーティション外部結合を使用して時系列または他のディメンションとのギャップを埋めることができます。この結合は、問合せで定義した各論理パーティションに外部結合を適用することで、従来の外部結合の構文を拡張したものです。Oracleでは、PARTITION BY句で指定した式に基づいて、問合せの行を論理的にパーティション化します。パーティション外部結合の結果は、論理的にパーティション化された表にある各パーティションの外部結合と、結合のもう一方の表のUNIONです。

このタイプの結合を使用すると、時間ディメンションに限らず、あらゆるディメンションのギャップを埋めることができます。比較の基

準として最も頻繁に使用されるのは時間ディメンションであるため、ここにあげた例のほとんどでは、時間のディメンションを対象としています。

この項では、次の項目について説明します。

- パーティション結合の構文について
- スパースなデータの例
- データのギャップ補完
- 2つのディメンションのギャップ補完
- 在庫表のギャップ補完
- ギャップを埋めるデータ値の計算

19.6.1 パーティション結合の構文について

パーティション外部結合の構文は、SQLのJOIN句にPARTITION BY句と式のリストを追加したものです。リストの式は、外部結合が適用されるグループを指定します。次に、パーティション外部結合で通常使用する2種類の構文を示します。

```
SELECT .....
FROM table_reference
PARTITION BY (expr [, expr ]... )
RIGHT OUTER JOIN table_reference

SELECT .....
FROM table_reference
LEFT OUTER JOIN table_reference
PARTITION BY {expr [,expr ]...)
```

FULL OUTER JOINは、パーティション外部結合ではサポートされないので注意してください。

19.6.2 スパースなデータの例

標準的なスパース・ディメンションの状態を次の例に示します。2000から2001年の20から30週における製品Bounceの1週間の売上と年度累計の売上が計算されます。

```
SELECT SUBSTR(p.Prod_Name,1,15) Product_Name, t.Calendar_Year Year,
 t.Calendar_Week_Number Week, SUM(Amount_Sold) Sales
FROM Sales s, Times t, Products p
WHERE s.Time_id = t.Time_id AND s.Prod_id = p.Prod_id AND
 p.Prod_name IN ('Bounce') AND t.Calendar_Year IN (2000,2001) AND
 t.Calendar_Week_Number BETWEEN 20 AND 30
GROUP BY p.Prod_Name, t.Calendar_Year, t.Calendar_Week_Number;

PRODUCT_NAME          YEAR       WEEK      SALES
--------------- ---------- ---------- ----------
Bounce                2000         20        801
Bounce                2000         21    4062.24
Bounce                2000         22    2043.16
Bounce                2000         23    2731.14
Bounce                2000         24    4419.36
Bounce                2000         27    2297.29
Bounce                2000         28    1443.13
Bounce                2000         29    1927.38
Bounce                2000         30    1927.38
Bounce                2001         20    1483.3
Bounce                2001         21    4184.49
```

```
Bounce                2001          22     2609.19
Bounce                2001          23     1416.95
Bounce                2001          24     3149.62
Bounce                2001          25     2645.98
Bounce                2001          27     2125.12
Bounce                2001          29     2467.92
Bounce                2001          30     2620.17
```

この例では、データが稠密化されていれば22行(1年11週間の2年分)になるはずです。しかし、2000年の25週と26週、2001年の26週と28週がないので、実際には18行のみです。

19.6.3 データのギャップ補完

スパースなデータの例に示す問合せのスパース・データを取り、時間データの稠密なセットでパーティション外部結合を実行できます。次の問合せでは、元の問合せをvという別名にし、tという別名にしたtimes表からデータを選択します。ここでは、時系列にギャップはないので、22行を取得します。追加された4行には、NVL関数で設定されたSales値0があります。

```
SELECT Product_Name, t.Year, t.Week, NVL(Sales,0) dense_sales
FROM
 (SELECT SUBSTR(p.Prod_Name,1,15) Product_Name,
  t.Calendar_Year Year, t.Calendar_Week_Number Week, SUM(Amount_Sold) Sales
  FROM Sales s, Times t, Products p
  WHERE s.Time_id = t.Time_id AND s.Prod_id = p.Prod_id AND
   p.Prod_name IN ('Bounce') AND t.Calendar_Year IN (2000,2001) AND
   t.Calendar_Week_Number BETWEEN 20 AND 30
  GROUP BY p.Prod_Name, t.Calendar_Year, t.Calendar_Week_Number) v
PARTITION BY (v.Product_Name)
RIGHT OUTER JOIN
 (SELECT DISTINCT Calendar_Week_Number Week, Calendar_Year Year
  FROM Times
  WHERE Calendar_Year IN (2000, 2001)
  AND Calendar_Week_Number BETWEEN 20 AND 30) t
ON (v.week = t.week AND v.Year = t.Year)
ORDER BY t.year, t.week;

PRODUCT_NAME          YEAR       WEEK DENSE_SALES
--------------- ---------- ---------- -----------
Bounce                2000         20         801
Bounce                2000         21     4062.24
Bounce                2000         22     2043.16
Bounce                2000         23     2731.14
Bounce                2000         24     4419.36
Bounce                2000         25           0
Bounce                2000         26           0
Bounce                2000         27     2297.29
Bounce                2000         28     1443.13
Bounce                2000         29     1927.38
Bounce                2000         30     1927.38
Bounce                2001         20      1483.3
Bounce                2001         21     4184.49
Bounce                2001         22     2609.19
Bounce                2001         23     1416.95
Bounce                2001         24     3149.62
Bounce                2001         25     2645.98
Bounce                2001         26           0
Bounce                2001         27     2125.12
Bounce                2001         28           0
Bounce                2001         29     2467.92
Bounce                2001         30     2620.17
```

この問合せでは、時間ディメンションのインライン・ビューに、20から30の週のWHERE条件が置かれています。これは、結果セットのサイズを大きくしないために導入されたものです。

19.6.4 2つのディメンションのギャップ補完

n次元のデータは通常、(n-2)ページのディメンションの稠密な2次元のクロス集計として表示されます。これには、クロス集計内にある2つのディメンションのディメンション値をすべて埋める必要があります。次に示す例では、2つのディメンションのギャップを埋めるために、パーティション外部結合機能が使用されています。

```
WITH v1 AS
 (SELECT p.prod_id, country_id, calendar_year,
    SUM(quantity_sold) units, SUM(amount_sold) sales
  FROM sales s, products p, customers c, times t
  WHERE s.prod_id in (147, 148) AND t.time_id = s.time_id AND
    c.cust_id = s.cust_id AND p.prod_id = s.prod_id
  GROUP BY p.prod_id, country_id, calendar_year),
v2 AS                                  --countries to use for densifications
  (SELECT DISTINCT country_id
  FROM customers
  WHERE country_id IN (52782, 52785, 52786, 52787, 52788)),
v3 AS                                   --years to use for densifications
  (SELECT DISTINCT calendar_year FROM times)
SELECT v4.prod_id, v4.country_id, v3.calendar_year, units, sales
FROM
  (SELECT prod_id, v2.country_id, calendar_year, units, sales
   FROM v1 PARTITION BY (prod_id)
   RIGHT OUTER JOIN v2                  --densifies on country
   ON (v1.country_id = v2.country_id)) v4
PARTITION BY (prod_id,country_id)
RIGHT OUTER JOIN v3                     --densifies on year
ON (v4.calendar_year = v3.calendar_year)
ORDER BY 1, 2, 3;
```

この問合せでは、WITH副問合せのファクタリング句v1で、製品、国、年レベルの売上データを集計します。この結果はスパースですが、製品ごとのすべての国、年の組合せを表示する場合もあります。このために、製品の値に基づいたv1の各パーティションを取り、最初に国ディメンションでそれを外部結合します。これによって、製品ごとの国の値がすべて提供されます。次に、結果を取り、それを製品と国の値でパーティション化し、さらに時間ディメンションで外部結合します。これで、製品と国の組合せごとに時間の値がすべて提供されます。

```
   PROD_ID COUNTRY_ID CALENDAR_YEAR      UNITS      SALES
---------- ---------- ------------- ---------- ----------
       147      52782          1998
       147      52782          1999         29     209.82
       147      52782          2000         71     594.36
       147      52782          2001        345    2754.42
       147      52782          2002
       147      52785          1998          1       7.99
       147      52785          1999
       147      52785          2000
       147      52785          2001
       147      52785          2002
       147      52786          1998          1       7.99
       147      52786          1999
       147      52786          2000          2      15.98
       147      52786          2001
       147      52786          2002
       147      52787          1998
```

```
       147      52787          1999
       147      52787          2000
       147      52787          2001
       147      52787          2002
       147      52788          1998
       147      52788          1999
       147      52788          2000          1        7.99
       147      52788          2001
       147      52788          2002
       148      52782          1998        139     4046.67
       148      52782          1999        228     5362.57
       148      52782          2000        251     5629.47
       148      52782          2001        308     7138.98
       148      52782          2002
       148      52785          1998
       148      52785          1999
       148      52785          2000
       148      52785          2001
       148      52785          2002
       148      52786          1998
       148      52786          1999
       148      52786          2000
       148      52786          2001
       148      52786          2002
       148      52787          1998
       148      52787          1999
       148      52787          2000
       148      52787          2001
       148      52787          2002
       148      52788          1998          4      117.23
       148      52788          1999
       148      52788          2000
       148      52788          2001
       148      52788          2002
```

19.6.5 在庫表のギャップ補完

在庫表は通常、様々な製品の有効単位数を追跡します。この表は、イベントが発生した場合にのみ製品の行を格納するため、スパースです。sales表の場合、イベントは売上であり、在庫表の場合、イベントは製品の有効数量の変化です。たとえば、次の在庫表を考えてみます。

```
CREATE TABLE invent_table (
product VARCHAR2(10),
time_id DATE,
quant NUMBER);

INSERT INTO invent_table VALUES
 ('bottle', TO_DATE('01/04/01', 'DD/MM/YY'), 10);
INSERT INTO invent_table VALUES
 ('bottle', TO_DATE('06/04/01', 'DD/MM/YY'), 8);
INSERT INTO invent_table VALUES
 ('can', TO_DATE('01/04/01', 'DD/MM/YY'), 15);
INSERT INTO invent_table VALUES
 ('can', TO_DATE('04/04/01', 'DD/MM/YY'), 11);
```

在庫表には、次の行があります。

```
PRODUCT    TIME_ID   QUANT
---------- --------- -----
```

```
bottle     01-APR-01    10
bottle     06-APR-01     8
can        01-APR-01    15
can        04-APR-01    11
```

レポートの目的で、この在庫データを異なる表示にする場合があります。たとえば、各製品の時間の値をすべて表示する必要があるような場合です。これには、パーティション外部結合を使用します。さらに、存在しない期間で新たに挿入された行については、最も近い既存の期間から持ち越された数量列の値を表示する場合があります。これには、分析ウィンドウ関数LAST_VALUEの値を使用します。次に、問合せとその目的とする出力を示します。

```
WITH v1 AS
 (SELECT time_id
  FROM times
  WHERE times.time_id BETWEEN
   TO_DATE('01/04/01', 'DD/MM/YY')
   AND TO_DATE('07/04/01', 'DD/MM/YY'))
SELECT product, time_id, quant quantity,
  LAST_VALUE(quant IGNORE NULLS)
    OVER (PARTITION BY product ORDER BY time_id)
    repeated_quantity
FROM
 (SELECT product, v1.time_id, quant
  FROM invent_table PARTITION BY (product)
  RIGHT OUTER JOIN v1
  ON (v1.time_id = invent_table.time_id))
ORDER BY 1, 2;
```

内部問合せは、各製品内の時間でパーティション外部結合を計算します。内部問合せは、時間ディメンションでデータを稠密化します(つまり、時間ディメンションは、週の各曜日の行を持つことになります)。ただし、メジャー列quantityは、新たに追加された行に対してNULLとなります(次の結果のquantity列の出力を参照してください)。

外部問合せでは、分析関数LAST_VALUEを使用しています。この関数を適用すると、製品単位でデータがパーティション化され、時間ディメンション列(time_id)でデータが順序付けられます。行ごとに、この関数は、IGNORE NULLSオプションを基に、ウィンドウ内で最後の非NULL値を検索します。このオプションは、LAST_VALUEとFIRST_VALUEの両方で使用できます。次の結果では、repeated_quantity列に目的の出力が表示されます。

```
PRODUCT    TIME_ID   QUANTITY REPEATED_QUANTITY
---------- --------- -------- -----------------
bottle     01-APR-01       10                10
bottle     02-APR-01                         10
bottle     03-APR-01                         10
bottle     04-APR-01                         10
bottle     05-APR-01                         10
bottle     06-APR-01        8                 8
bottle     07-APR-01                         8
can        01-APR-01       15                15
can        02-APR-01                         15
can        03-APR-01                         15
can        04-APR-01       11                11
can        05-APR-01                         11
can        06-APR-01                         11
can        07-APR-01                         11
```

19.6.6 ギャップを埋めるデータ値の計算

データのギャップ補完、2つのディメンションのギャップ補完、在庫表のギャップ補完の各項の例は、パーティション外部結合を使

用して1つ以上のディメンションのギャップを埋める方法を示しています。ただし、パーティション外部結合で生成された結果セットでは、PARTITION BYリストに含まれない列にNULL値があります。通常、これらはメジャー列です。分析SQL関数を使用すると、こうしたNULL値を非NULL値に置き換えることができます。

たとえば、次の問合せは、2000年の64MBメモリー・カードおよびDVD-Rディスク(製品IDは122と136)の月の売上合計を計算します。この問合せでは、すべての月のデータを稠密化するために、パーティション外部結合を使用します。売上が存在しない月に対しては、分析SQL関数AVGを使用して、製品の売上が存在する各月の平均売上と平均数量を計算します。

SQL*Plusで作業している場合は、次の2つのコマンドを実行すると、列ヘッダーが折り返されて結果が読みやすくなります。

```
col computed_units  heading  'Computed|_units'
col computed_sales  heading  'Computed|_sales'

WITH V AS
 (SELECT substr(p.prod_name,1,12) prod_name, calendar_month_desc,
     SUM(quantity_sold) units, SUM(amount_sold) sales
   FROM sales s, products p, times t
   WHERE s.prod_id IN (122,136) AND calendar_year = 2000
     AND t.time_id = s.time_id
     AND p.prod_id = s.prod_id
   GROUP BY p.prod_name, calendar_month_desc)
SELECT v.prod_name, calendar_month_desc, units, sales,
  NVL(units, AVG(units) OVER (PARTITION BY v.prod_name)) computed_units,
  NVL(sales, AVG(sales) OVER (PARTITION BY v.prod_name)) computed_sales
FROM
  (SELECT DISTINCT calendar_month_desc
   FROM times
   WHERE calendar_year = 2000) t
   LEFT OUTER JOIN V
   PARTITION BY (prod_name)
   USING (calendar_month_desc);
                                                  computed   computed
PROD_NAME    CALENDAR      UNITS      SALES      _units     _sales
------------ -------- ---------- ---------- ---------- ----------
64MB Memory  2000-01         112    4129.72        112    4129.72
64MB Memory  2000-02         190       7049        190       7049
64MB Memory  2000-03          47    1724.98         47    1724.98
64MB Memory  2000-04          20      739.4         20      739.4
64MB Memory  2000-05          47    1738.24         47    1738.24
64MB Memory  2000-06          20      739.4         20      739.4
64MB Memory  2000-07                          72.6666667    2686.79
64MB Memory  2000-08                          72.6666667    2686.79
64MB Memory  2000-09                          72.6666667    2686.79
64MB Memory  2000-10                          72.6666667    2686.79
64MB Memory  2000-11                          72.6666667    2686.79
64MB Memory  2000-12                          72.6666667    2686.79
DVD-R Discs, 2000-01         167     3683.5        167     3683.5
DVD-R Discs, 2000-02         152    3362.24        152    3362.24
DVD-R Discs, 2000-03         188    4148.02        188    4148.02
DVD-R Discs, 2000-04         144    3170.09        144    3170.09
DVD-R Discs, 2000-05         189    4164.87        189    4164.87
DVD-R Discs, 2000-06         145    3192.21        145    3192.21
DVD-R Discs, 2000-07                           124.25    2737.71
DVD-R Discs, 2000-08                           124.25    2737.71
DVD-R Discs, 2000-09           1      18.91          1      18.91
DVD-R Discs, 2000-10                           124.25    2737.71
DVD-R Discs, 2000-11                           124.25    2737.71
DVD-R Discs, 2000-12           8     161.84          8     161.84
```

19.7 稠密化したデータに対する時系列の計算

稠密化は、レポートのみを目的としたものではありません。稠密化によって、一部の計算、特に時系列の計算が可能となります。時系列の計算は、データが時間ディメンションに沿って稠密であれば、より簡単になります。稠密なデータは、期間ごとに一定数の行を占めているため、物理オフセットを指定した分析ウィンドウ関数の使用が単純化されます。

実例として、データのギャップ補完に示す例を使用し、その問合せに分析関数を追加します。次の機能強化した例では、週の売上とともに、週次年度累計売上を計算します。時系列の稠密化においてパーティション外部結合により挿入されたNULL値は、通常どおりに処理されます。つまり、SUM関数はこの値を0として処理します。

```
SELECT Product_Name, t.Year, t.Week, NVL(Sales,0) Current_sales,
 SUM(Sales)
   OVER (PARTITION BY Product_Name, t.year ORDER BY t.week) Cumulative_sales
FROM
 (SELECT SUBSTR(p.Prod_Name,1,15) Product_Name, t.Calendar_Year Year,
   t.Calendar_Week_Number Week, SUM(Amount_Sold) Sales
  FROM Sales s, Times t, Products p
  WHERE s.Time_id = t.Time_id AND
   s.Prod_id = p.Prod_id AND p.Prod_name IN ('Bounce') AND
   t.Calendar_Year IN (2000,2001) AND
   t.Calendar_Week_Number BETWEEN 20 AND 30
  GROUP BY p.Prod_Name, t.Calendar_Year, t.Calendar_Week_Number) v
PARTITION BY (v.Product_Name)
RIGHT OUTER JOIN
(SELECT DISTINCT
 Calendar_Week_Number Week, Calendar_Year Year
 FROM Times
 WHERE Calendar_Year in (2000, 2001)
 AND Calendar_Week_Number BETWEEN 20 AND 30) t
ON (v.week = t.week AND v.Year = t.Year)
ORDER BY t.year, t.week;

PRODUCT_NAME          YEAR       WEEK CURRENT_SALES CUMULATIVE_SALES
--------------- ---------- ---------- ------------- ----------------
Bounce                2000         20           801              801
Bounce                2000         21       4062.24          4863.24
Bounce                2000         22       2043.16           6906.4
Bounce                2000         23       2731.14          9637.54
Bounce                2000         24       4419.36          14056.9
Bounce                2000         25             0          14056.9
Bounce                2000         26             0          14056.9
Bounce                2000         27       2297.29         16354.19
Bounce                2000         28       1443.13         17797.32
Bounce                2000         29       1927.38          19724.7
Bounce                2000         30       1927.38         21652.08
Bounce                2001         20        1483.3           1483.3
Bounce                2001         21       4184.49          5667.79
Bounce                2001         22       2609.19          8276.98
Bounce                2001         23       1416.95          9693.93
Bounce                2001         24       3149.62         12843.55
Bounce                2001         25       2645.98         15489.53
Bounce                2001         26             0         15489.53
Bounce                2001         27       2125.12         17614.65
Bounce                2001         28             0         17614.65
Bounce                2001         29       2467.92         20082.57
Bounce                2001         30       2620.17         22702.74
```

この項では、次の項目について説明します。

- 1つの時間レベルでの周期ごとの比較: 例
- 複数の時間レベルでの周期ごとの比較: 例
- ディメンションのカスタム・メンバーの作成: 例

19.7.1 1つの時間レベルでの周期ごとの比較: 例

複数の期間にわたって値を比較するには、この方法をどのように使用しますか。特に、週レベルで複数年度にわたる売上の比較については、どのように計算しますか。次の問合せは、同じ行で、2000年と2001年の各製品の週次年度累計売上を戻します。

この例では、はじめにWITH句がきています。これによって、問合せの読みやすさが向上し、パーティション外部結合に集中できます。SQL*Plusで作業している場合は、次のコマンドを実行すると、列ヘッダーが折り返されて結果が読みやすくなります。

```
col Weekly_ytd_sales_prior_year  heading  'Weekly_ytd|_sales_|prior_year'

WITH v AS
  (SELECT SUBSTR(p.Prod_Name,1,6) Prod, t.Calendar_Year Year,
     t.Calendar_Week_Number Week, SUM(Amount_Sold) Sales
   FROM Sales s, Times t, Products p
   WHERE s.Time_id = t.Time_id AND
     s.Prod_id = p.Prod_id AND p.Prod_name in ('Y Box') AND
     t.Calendar_Year in (2000,2001) AND
     t.Calendar_Week_Number BETWEEN 30 AND 40
   GROUP BY p.Prod_Name, t.Calendar_Year, t.Calendar_Week_Number)
SELECT Prod , Year, Week, Sales,
  Weekly_ytd_sales, Weekly_ytd_sales_prior_year
FROM
  (SELECT Prod, Year, Week, Sales, Weekly_ytd_sales,
     LAG(Weekly_ytd_sales, 1) OVER
       (PARTITION BY Prod , Week ORDER BY Year) Weekly_ytd_sales_prior_year
   FROM
     (SELECT v.Prod  Prod , t.Year Year, t.Week Week,
        NVL(v.Sales,0) Sales, SUM(NVL(v.Sales,0)) OVER
          (PARTITION BY v.Prod , t.Year ORDER BY t.week) weekly_ytd_sales
      FROM v
      PARTITION BY (v.Prod )
      RIGHT OUTER JOIN
        (SELECT DISTINCT Calendar_Week_Number Week, Calendar_Year Year
         FROM Times
         WHERE Calendar_Year IN (2000, 2001)) t
      ON (v.week = t.week AND v.Year = t.Year)
     ) dense_sales
  ) year_over_year_sales
WHERE Year = 2001 AND Week BETWEEN 30 AND 40
ORDER BY 1, 2, 3;
                                                          Weekly_ytd
                                                             _sales_
PROD         YEAR       WEEK      SALES WEEKLY_YTD_SALES prior_year
------ ---------- ---------- ---------- ---------------- ----------
Y Box        2001         30    7877.45          7877.45          0
Y Box        2001         31   13082.46         20959.91    1537.35
Y Box        2001         32   11569.02         32528.93    9531.57
Y Box        2001         33   38081.97          70610.9   39048.69
Y Box        2001         34   33109.65        103720.55   69100.79
Y Box        2001         35          0        103720.55   71265.35
Y Box        2001         36     4169.3        107889.85   81156.29
Y Box        2001         37   24616.85         132506.7   95433.09
```

```
Y Box        2001        38    37739.65        170246.35   107726.96
Y Box        2001        39      284.95          170531.3    118817.4
Y Box        2001        40    10868.44        181399.74   120969.69
```

インライン・ビューdense_salesのFROM句では、集計ビューvと時間ビューtのパーティション外部結合を使用して、時間ディメンションに沿って売上データのギャップを埋めます。パーティション外部結合の結果が、分析関数SUM ... OVERで処理され、週次年度累計売上(weekly_ytd_sales列)が計算されます。したがって、dense_salesビューは、集計ビューにないものも含めて、週次年度累計売上データを計算します。次に、インライン・ビューyear_over_year_salesは、LAG関数を使用して、1年前の週次年度累計売上を計算します。weekly_ytd_sales_prior_yearというラベルが付けられたLAG関数には、2000年と2001年の同じ週の行を単一のパーティションに組み合せるPARTITION BY句を指定します。1のオフセットをLAG関数に渡し、前年の週次年度累計売上を取得します。最も外側の問合せブロックでは、条件yr = 2001でyear_over_year_salesのデータを選択します。したがって、この問合せでは各製品について、2001年と2000年の指定した週の週次年度累計売上が戻されます。

19.7.2 複数の時間レベルでの周期ごとの比較: 例

前の例では、単一の時間レベルでの比較を作成する方法を示しましたが、1回の問合せで複数の時間レベルを処理できればより便利となります。たとえば、年、四半期、月、曜日レベルで前期との売上を比較できます。ここでは、時間階層のすべてのレベルで、年度累計売上の年度ごとの比較を実行する問合せの作成方法について説明します。

このタスクを実行するには、いくつかのステップを踏みます。目標は、1回の問合せで日、週、月、四半期、年レベルでの比較を実行することです。ステップは次のとおりです。

1. cube_prod_timeというビューを作成します。このビューは、timesとproductsにわたって集計した売上の階層的キューブを保持します。

   階層的キューブ・ビューの作成を参照してください。

2. キューブのエッジとして使用する時間ディメンションのビューを作成します。完全な日付セットを保持する時間エッジは、cube_prod_timeビューでスパース・データにパーティション外部結合されます。

   日付値の完全なセットであるビューedge_timeの作成を参照してください。

3. 最後に、パフォーマンスを最大にするために、マテリアライズド・ビューmv_prod_timeを作成します。これにはcube_prod_timeと同じディメンションを使用します。

   パフォーマンス向上をサポートするマテリアライズド・ビューmv_prod_timeの作成を参照してください。

4. 比較問合せを作成します。

   比較問合せの作成を参照してください。

階層的キューブの詳細は、データ・ウェアハウスにおける集計のためのSQLを参照してください。マテリアライズド・ビューは、次の項で定義します。

階層的キューブ・ビューの作成

次に示すマテリアライズド・ビューは、すでにシステムに存在している場合もあります。存在していない場合は、ここで作成します。作成が必要な場合は、処理時間をかけないように、問合せでは2つの製品に制限するので注意してください。

```
CREATE OR REPLACE VIEW cube_prod_time AS
SELECT
  (CASE
     WHEN ((GROUPING(calendar_year)=0 )
       AND (GROUPING(calendar_quarter_desc)=1 ))
       THEN (TO_CHAR(calendar_year) || '_0')
     WHEN ((GROUPING(calendar_quarter_desc)=0 )
```

```
       AND (GROUPING(calendar_month_desc)=1 ))
       THEN (TO_CHAR(calendar_quarter_desc) || '_1')
     WHEN ((GROUPING(calendar_month_desc)=0 )
       AND (GROUPING(t.time_id)=1 ))
       THEN (TO_CHAR(calendar_month_desc) || '_2')
     ELSE (TO_CHAR(t.time_id) || '_3')
  END) Hierarchical_Time,
  calendar_year year, calendar_quarter_desc quarter,
  calendar_month_desc month, t.time_id day,
  prod_category cat, prod_subcategory subcat, p.prod_id prod,
  GROUPING_ID(prod_category, prod_subcategory, p.prod_id,
    calendar_year, calendar_quarter_desc, calendar_month_desc,t.time_id) gid,
  GROUPING_ID(prod_category, prod_subcategory, p.prod_id) gid_p,
  GROUPING_ID(calendar_year, calendar_quarter_desc,
    calendar_month_desc, t.time_id) gid_t,
  SUM(amount_sold) s_sold, COUNT(amount_sold) c_sold, COUNT(*) cnt
FROM SALES s, TIMES t, PRODUCTS p
WHERE s.time_id = t.time_id AND
  p.prod_name IN ('Bounce', 'Y Box') AND s.prod_id = p.prod_id
GROUP BY
  ROLLUP(calendar_year, calendar_quarter_desc, calendar_month_desc, t.time_id),
  ROLLUP(prod_category, prod_subcategory, p.prod_id);
```

このビューは2つの製品に限定されるため、2200強の行のみが戻されます。Hierarchical_Time列には、時間階層の全レベルの文字列表現があるので注意が必要です。Hierarchical_Time列で使用されるCASE式は、マーカー(_0, _1, ...)を各日付文字列に付加して、値の時間レベルを示します。_0は年レベルを表し、_1は四半期レベル、_2は月レベル、_3は日レベルを表します。GROUP BY句は、時間ディメンションおよび製品ディメンションのロールアップ階層を指定する、連結されたROLLUPです。GROUP BY句によって、階層的キューブの内容が決まります。

日付値の完全なセットであるビューedge_timeの作成

edge_timeを使用して、パーティション外部結合による階層的キューブの時間のギャップを埋めます。edge_timeのHierarchical_Time列が、ビューcube_prod_timeのHierarchical_Time列とのパーティション結合に使用されます。edge_timeは次の文で定義します。

```
CREATE OR REPLACE VIEW edge_time AS
SELECT
  (CASE
     WHEN ((GROUPING(calendar_year)=0 )
       AND (GROUPING(calendar_quarter_desc)=1 ))
       THEN (TO_CHAR(calendar_year) || '_0')
     WHEN ((GROUPING(calendar_quarter_desc)=0 )
       AND (GROUPING(calendar_month_desc)=1 ))
       THEN (TO_CHAR(calendar_quarter_desc) || '_1')
     WHEN ((GROUPING(calendar_month_desc)=0 )
       AND (GROUPING(time_id)=1 ))
       THEN (TO_CHAR(calendar_month_desc) || '_2')
     ELSE (TO_CHAR(time_id) || '_3')
   END) Hierarchical_Time,
   calendar_year yr, calendar_quarter_number qtr_num,
   calendar_quarter_desc qtr, calendar_month_number mon_num,
   calendar_month_desc mon, time_id - TRUNC(time_id, 'YEAR') + 1 day_num,
   time_id day,
GROUPING_ID(calendar_year, calendar_quarter_desc,
  calendar_month_desc, time_id) gid_t
FROM TIMES
GROUP BY ROLLUP
 (calendar_year, (calendar_quarter_desc, calendar_quarter_number),
 (calendar_month_desc, calendar_month_number), time_id);
```

パフォーマンス向上をサポートするマテリアライズド・ビューmv_prod_timeの作成

このマテリアライズド・ビューの定義は、これまでに定義したビューcube_prod_timeの定義と同じです。同じ問合せとなるので、cube_prod_timeへの参照はマテリアライズド・ビューmv_prod_timeを使用するように書き換えられます。次のマテリアライズド・ビューは、すでにシステムに存在している場合もあります。存在していない場合は、ここで作成します。作成が必要な場合は、処理時間をかけないように、問合せでは2つの製品に制限するので注意してください。

```
CREATE MATERIALIZED VIEW mv_prod_time
REFRESH COMPLETE ON DEMAND AS
SELECT
  (CASE
     WHEN ((GROUPING(calendar_year)=0 )
       AND (GROUPING(calendar_quarter_desc)=1 ))
       THEN (TO_CHAR(calendar_year) || '_0')
     WHEN ((GROUPING(calendar_quarter_desc)=0 )
       AND (GROUPING(calendar_month_desc)=1 ))
       THEN (TO_CHAR(calendar_quarter_desc) || '_1')
     WHEN ((GROUPING(calendar_month_desc)=0 )
       AND (GROUPING(t.time_id)=1 ))
       THEN (TO_CHAR(calendar_month_desc) || '_2')
     ELSE (TO_CHAR(t.time_id) || '_3')
  END) Hierarchical_Time,
  calendar_year year, calendar_quarter_desc quarter,
  calendar_month_desc month, t.time_id day,
  prod_category cat, prod_subcategory subcat, p.prod_id prod,
  GROUPING_ID(prod_category, prod_subcategory, p.prod_id,
    calendar_year, calendar_quarter_desc, calendar_month_desc,t.time_id) gid,
  GROUPING_ID(prod_category, prod_subcategory, p.prod_id) gid_p,
  GROUPING_ID(calendar_year, calendar_quarter_desc,
    calendar_month_desc, t.time_id) gid_t,
  SUM(amount_sold) s_sold, COUNT(amount_sold) c_sold, COUNT(*) cnt
FROM SALES s, TIMES t, PRODUCTS p
WHERE s.time_id = t.time_id AND
  p.prod_name IN ('Bounce', 'Y Box') AND s.prod_id = p.prod_id
GROUP BY
  ROLLUP(calendar_year, calendar_quarter_desc, calendar_month_desc, t.time_id),
  ROLLUP(prod_category, prod_subcategory, p.prod_id);
```

比較問合せの作成

ここまでで比較問合せの段階が設定されました。すべての時間レベルでの周期ごとの比較計算を取得できます。それには、時間ディメンションに沿った稠密なデータで、分析関数を階層的キューブに適用する必要があります。

次に、各時間レベルで実行可能な計算の一部を示します。

- すべての時間レベルでの前期の売上の合計
- 前の期間からの売上の変化
- すべての時間レベルでの1年前の同じ期間の売上の合計
- 昨年の同じ期間からの売上の変化

次の例では、この4つの計算をすべて実行します。ビューcube_prod_timeおよびedge_timeのパーティション外部結合を使用して、dense_cube_prod_timeという稠密なデータのインライン・ビューを作成します。続いて、前の単一レベルの例と同様に、問合せでLAG関数を使用します。外部WHERE句は、2001年8月の各日、8月全体、2001年の第3四半期の3つのレベルで時間を指定します。結果の最後の2行には、月レベルと四半期レベルの集計が含まれます。SQL*Plusを使用している場合は、次のコマンドで列ヘッダーを調整すると、結果が読みやすくなります。このコマンドを実行すると、列ヘッダーが折り返されて行が短

くなります。

```
col sales_prior_period heading 'sales_prior|_period'
col variance_prior_period heading 'variance|_prior|_period'
col sales_same_period_prior_year heading 'sales_same|_period_prior|_year'
col variance_same_period_p_year heading 'variance|_same_period|_prior_year'
```

次に、現在の売上を前期の売上および1年前の売上と比較する問合せを示します。

```
SELECT SUBSTR(prod,1,4) prod, SUBSTR(Hierarchical_Time,1,12) ht,
  sales, sales_prior_period,
  sales - sales_prior_period variance_prior_period,
  sales_same_period_prior_year,
  sales - sales_same_period_prior_year variance_same_period_p_year
FROM
 (SELECT cat, subcat, prod, gid_p, gid_t,
    Hierarchical_Time, yr, qtr, mon, day, sales,
    LAG(sales, 1) OVER (PARTITION BY gid_p, cat, subcat, prod,
      gid_t ORDER BY yr, qtr, mon, day)
      sales_prior_period,
    LAG(sales, 1) OVER (PARTITION BY gid_p, cat, subcat, prod,
      gid_t, qtr_num, mon_num, day_num ORDER BY yr)
      sales_same_period_prior_year
  FROM
   (SELECT c.gid, c.cat, c.subcat, c.prod, c.gid_p,
      t.gid_t, t.yr, t.qtr, t.qtr_num, t.mon, t.mon_num,
      t.day, t.day_num, t.Hierarchical_Time, NVL(s_sold,0) sales
    FROM cube_prod_time c
    PARTITION BY (gid_p, cat, subcat, prod)
    RIGHT OUTER JOIN edge_time t
    ON ( c.gid_t = t.gid_t AND
      c.Hierarchical_Time = t.Hierarchical_Time)
   ) dense_cube_prod_time
 )                        --side by side current and prior year sales
WHERE prod IN (139) AND gid_p=0 AND    --1 product and product level data
  ( (mon IN ('2001-08' ) AND gid_t IN (0, 1)) OR --day and month data
  (qtr IN ('2001-03' ) AND gid_t IN (3)))        --quarter level data
ORDER BY day;

                                          variance    sales_same     variance
                            sales_prior     _prior _period_prior _same_period
PROD HT                SALES     _period    _period         _year  _prior_year
---- ------------ ---------- ----------- ---------- ------------- ------------
139  01-AUG-01_3           0           0          0             0            0
139  02-AUG-01_3     1347.53           0    1347.53             0      1347.53
139  03-AUG-01_3           0     1347.53   -1347.53         42.36       -42.36
139  04-AUG-01_3       57.83           0      57.83        995.75      -937.92
139  05-AUG-01_3           0       57.83     -57.83             0            0
139  06-AUG-01_3           0           0          0             0            0
139  07-AUG-01_3      134.81           0     134.81        880.27      -745.46
139  08-AUG-01_3     1289.89      134.81    1155.08             0      1289.89
139  09-AUG-01_3           0     1289.89   -1289.89             0            0
139  10-AUG-01_3           0           0          0             0            0
139  11-AUG-01_3           0           0          0             0            0
139  12-AUG-01_3           0           0          0             0            0
139  13-AUG-01_3           0           0          0             0            0
139  14-AUG-01_3           0           0          0             0            0
139  15-AUG-01_3       38.49           0      38.49       1104.55     -1066.06
139  16-AUG-01_3           0       38.49     -38.49             0            0
139  17-AUG-01_3       77.17           0      77.17       1052.03      -974.86
```

```
139  18-AUG-01_3     2467.54       77.17     2390.37            0     2467.54
139  19-AUG-01_3           0     2467.54    -2467.54       127.08     -127.08
139  20-AUG-01_3           0           0           0            0           0
139  21-AUG-01_3           0           0           0            0           0
139  22-AUG-01_3           0           0           0            0           0
139  23-AUG-01_3     1371.43           0     1371.43            0     1371.43
139  24-AUG-01_3      153.96     1371.43    -1217.47       2091.3    -1937.34
139  25-AUG-01_3           0      153.96     -153.96            0           0
139  26-AUG-01_3           0           0           0            0           0
139  27-AUG-01_3     1235.48           0     1235.48            0     1235.48
139  28-AUG-01_3       173.3     1235.48    -1062.18       2075.64    -1902.34
139  29-AUG-01_3           0       173.3      -173.3            0           0
139  30-AUG-01_3           0           0           0            0           0
139  31-AUG-01_3           0           0           0            0           0
139  2001-08_2       8347.43     7213.21     1134.22      8368.98      -21.55
139  2001-03_1       24356.8    28862.14    -4505.34     24168.99      187.81
```

最初のLAG関数(sales_prior_period)により、gid_p、cat、subcat、prodおよびgid_tのデータがパーティション化され、すべての時間ディメンション列で行が順序付けされます。この関数ではオフセット1を渡すことで前期の売上値が取得されます。2番目のLAG関数(sales_same_period_prior_year)では、この他にqtr_num、mon_numおよびday_numの各列でもデータがパーティション化され、yrで順序付けされます。その結果、オフセット1によって1年前の同じ期間の売上が計算されます。最も外側のSELECT句は、分散を計算します。

19.7.3 ディメンションのカスタム・メンバーの作成: 例

多くの分析SQLタスクでは、ディメンションでカスタム・メンバーを定義すると役立ちます。たとえば、分析用に特定の期間を定義するとします。パーティション外部結合を使用して、メンバーを一時的にディメンションに追加できます。新しくSQLに導入されたMODEL句は、ディメンション内の新しいメンバーが関与する複雑なシナリオの作成に適しています。詳細は、モデリングのSQLを参照してください。

タスクの例として、timeディメンションの新しいメンバーを定義する場合はどうするかを示します。ここでは、timeディメンションで月レベルの13番目のメンバーを作成します。この13番目の月は、2001年の各四半期の最初の月における各製品の売上の合計として定義します。

これを行うには、2つのステップがあります。ここでは、前の例で作成したビューと表を使用します。この2つのステップは必須です。まず、適切なディメンションに追加される新しいメンバーを持つビューを作成します。ビューでは、UNION ALL演算子を使用して新しいメンバーが追加されます。カスタム・メンバーを使用して問合せを実行するには、CASE式およびパーティション外部結合を使用します。

timeディメンションの新しいメンバーは、次のビューで作成されます。

```
CREATE OR REPLACE VIEW time_c AS
(SELECT * FROM edge_time
UNION ALL
SELECT '2001-13_2', 2001, 5, '2001-05', 13, '2001-13', null, null,
8 -- <gid_of_mon>
FROM DUAL);
```

この文では、(前の例で定義した)edge_timeビューとユーザー定義の13番目の月のUNION ALLを実行することで、ビューtime_cが定義されます。標準メンバーからカスタム・メンバーを区別するために、gid_t値として8が選択されています。UNION ALLでは、DUAL表からSELECTを実行することで、13番目の月のメンバーの属性が指定されます。グルーピングid(列gid_t)は8に設定され、四半期の番号は5に設定されます。

したがって、2番目のステップは、問合せでインライン・ビューを使用して、cube_prod_timeとtime_cのパーティション外部結合を実行することです。このステップでは、製品集計の各レベルで13番目の月の売上データを作成します。メインの問合せでは、分

析関数SUMがCASE式とともに使用され、各四半期の最初の月における売上合計として定義された13番目の月が計算されます。

```
SELECT * FROM  (SELECT SUBSTR(cat,1,12) cat, SUBSTR(subcat,1,12) subcat,
     prod, mon, mon_num,
     SUM(CASE WHEN mon_num IN (1, 4, 7, 10)
           THEN s_sold
           ELSE NULL
         END)
       OVER (PARTITION BY gid_p, prod, subcat, cat, yr) sales_month_13
   FROM
     (SELECT c.gid, c.prod, c.subcat, c.cat, gid_p,
        t.gid_t, t.day, t.mon, t.mon_num,
        t.qtr, t.yr, NVL(s_sold,0) s_sold
      FROM cube_prod_time c
      PARTITION BY (gid_p, prod, subcat, cat)
      RIGHT OUTER JOIN time_c t
      ON (c.gid_t = t.gid_t AND
        c.Hierarchical_Time = t.Hierarchical_Time)
     )
  )
WHERE mon_num=13;

CAT          SUBCAT             PROD MON         MON_NUM SALES_MONTH_13
------------ ------------ ---------- -------- ---------- --------------
Electronics  Game Console          16 2001-13          13      762334.34
Electronics  Y Box Games          139 2001-13          13       75650.22
Electronics  Game Console             2001-13          13      762334.34
Electronics  Y Box Games              2001-13          13       75650.22
Electronics                           2001-13          13      837984.56
                                      2001-13          13      837984.56
```

SUM関数では、CASEが使用され、データが各年の1、4、7、10月に制限されます。データセットが2製品のみと小さいため、結果のロールアップ値は、必然的により低いレベルの集計の繰り返しになります。ロールアップ値のより現実的なセットでは、Game ConsoleおよびY Box Gamesサブカテゴリからより多くの製品を、基底のマテリアライズド・ビューに含めることができます。

19.8 その他の分析およびレポートの機能

この項では、次に示す追加の分析機能について説明します。

- WIDTH_BUCKET関数
- 線形代数
- CASE式
- SQL分析での高頻度項目セット

19.8.1 WIDTH_BUCKET関数

WIDTH_BUCKET関数は、特定の式について、この式の結果に対して評価後に割り当てられるバケット番号を戻します。WIDTH_BUCKETの構文では、WIDTH_BUCKETの構文について説明します。

この関数を使用すると、ヒストグラムを生成できます。ヒストグラムでは、データ集合はインターバル・サイズ(最大値から最小値まで)が等しい等幅バケットに分割されます。各バケットに保持される行数は変動します。関連する関数NTILEでは、等度数バケットが作成されます。

ヒストグラムを生成できるのは、数値または日付のデータ型の場合のみです。したがって、最初の3つのパラメータは、すべて数値

型またはすべて日付型の式にする必要があります。他の型の式は使用できません。最初のパラメータがNULLの場合、結果はNULLです。2番目または3番目のパラメータがNULLの場合、NULL値は日付または数値のディメンションの範囲について終了点(または点)を示すことができないため、エラー・メッセージが戻されます。最後のパラメータ(バケット数)は、正の整数値に評価される数値型の式にする必要があります。0(ゼロ)、NULLまたは負の値の場合はエラーになります。

バケットには、0から(n+1)の番号が付いています。バケット0には最小値未満の値のカウントが保持されます。バケット(n+1)には、指定した最大値以上の値のカウントが保持されます。

19.8.1.1 WIDTH_BUCKETの構文

WIDTH_BUCKETは、パラメータとして4つの式をとります。最初のパラメータは、ヒストグラムの対象となる式です。2番目と3番目のパラメータは、最初のパラメータについて許容範囲の端点を示す式です。4番目のパラメータは、バケット数を示します。

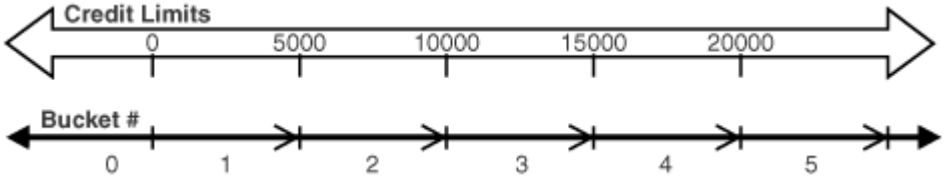
```
WIDTH_BUCKET(expression, minval expression, maxval expression, num buckets)
```

表customersからの次のデータを考えてみます。これは、17人の顧客の与信限度額を示しています。このデータは、例19-27の問合せで収集されます。

```
CUST_ID    CUST_CREDIT_LIMIT
---------  -----------------
    10346               7000
    35266               7000
    41496              15000
    35225              11000
     3424               9000
    28344               1500
    31112               7000
     8962               1500
    15192               3000
    21380               5000
    36651               1500
    30420               5000
     8270               3000
    17268              11000
    14459              11000
    13808               5000
    32497               1500
   100977               9000
   102077               3000
   103066              10000
   101784               5000
   100421              11000
   102343               3000
```

このcustomers表には、cust_credit_limit 列に1500から15000までの値があり、WIDTH_BUCKET (cust_credit_limit, 0, 20000, 4)を使用すると、これらの値を1から4の番号が付いた4つの等幅バケットに割り当てることができます。理想的には、各バケットは実際の数値直線のクローズ/オープン・インターバルです。たとえば、バケット番号2は5000.0000から9999.9999...のスコアに割り当てられており、[5000, 10000)は5,000がインターバルに含まれ、10,000は除外されることを示す場合があります。範囲[0, 20,000)外の他の値に対処するために、0未満の値は番号0で指定されたアンダーフロー・バケットに割り当てられ、20,000以上の値は番号5(通常はバケット数+1)で指定されたオーバーフロー・バケットに割り当てられます。バケットの割当て方法については、図19-4を参照してください。

図19-4 バケットの割当て



バケットの境界は、WIDTH_BUCKET (cust_credit_limit, 20000, 0, 4)のように逆順で指定することもできます。境界が逆順になっている場合、バケットはオープン/クローズ・インターバルとなります。この例で、バケット番号1は(15000, 20000]、バケット番号2は(10000, 15000]、バケット番号4は(0, 5000]となります。オーバーフロー・バケットの番号は0 (20000, +infinity)、アンダーフロー・バケットの番号は5 (-infinity, 0)です。

バケット・カウント・パラメータが0または負の値の場合は、エラーになります。

例19-27 WIDTH_BUCKET

次の問合せは、customers表の与信限度額のバケット番号を示しています。境界は、それぞれ通常の順序および逆順で指定されています。0から20,000の範囲を使用します。

```
SELECT cust_id, cust_credit_limit,
 WIDTH_BUCKET(cust_credit_limit,0,20000,4) AS WIDTH_BUCKET_UP,
 WIDTH_BUCKET(cust_credit_limit,20000, 0, 4) AS WIDTH_BUCKET_DOWN
FROM customers WHERE cust_city = 'Marshal';

   CUST_ID CUST_CREDIT_LIMIT WIDTH_BUCKET_UP WIDTH_BUCKET_DOWN
---------- ----------------- --------------- -----------------
     10346              7000               2                 3
     35266              7000               2                 3
     41496             15000               4                 2
     35225             11000               3                 2
      3424              9000               2                 3
     28344              1500               1                 4
     31112              7000               2                 3
      8962              1500               1                 4
     15192              3000               1                 4
     21380              5000               2                 4
     36651              1500               1                 4
     30420              5000               2                 4
      8270              3000               1                 4
     17268             11000               3                 2
     14459             11000               3                 2
     13808              5000               2                 4
     32497              1500               1                 4
    100977              9000               2                 3
    102077              3000               1                 4
    103066             10000               3                 3
    101784              5000               2                 4
    100421             11000               3                 2
    102343              3000               1                 4
```

19.8.2 線形代数

線形代数は数学の一分野であり、実践的にも幅広く応用されています。線形代数を使用して表現できる事柄を研究や分析の対象として含む領域は多く、その一部として例をあげれば、統計学(多重線形回帰分析や主成分分析)、データ・マイニング(クラスタリングや分類)、バイオインフォマティクス(マイクロアレイ・データの分析)、オペレーションズ・リサーチ(サプライ・チェーンなどの最適化問題)、経済学(消費者需要データの分析)、金融(資産分配問題)など様々です。無償で利用できる線形代数用ライブラリも、様々な種類のものが提供されています。OracleのUTL_NLAパッケージでは、強力な線形代数ライブラリとして定評のあるBLASとLAPACKのためのPL/SQL行列データ型やPL/SQLラッパー・サブプログラムを利用できます。

線形代数の土台となっているのは行列演算です。これまで、PL/SQLで行列演算を実行するためには、PL/SQLのネイティブ・データ型を基にした行列の表現を考案した上で、行列演算のルーチンを一から記述する必要がありました。そのためには、きわめて手間のかかるプログラミングが必要となり、それを実装した後のパフォーマンスにも限界がありました。一方、開発者が独自のルーチンを作成せず、データを外部パッケージに送って処理しようとすると、データのやりとりに時間がかかる場合があります。UTL_NLAパッケージを使用すれば、データをOracle内部で処理できるのみでなく、手間のかかるプログラミングも不要になり、高速な実装が実現されます。

関連項目:

UTL_NLAパッケージおよび線形代数の使用方法の詳細は、『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

例19-28 線形代数

ビジネス分析にOracleの線形代数サポートをどのように活用するかについて、1つの例を紹介します。UTL_NLAパッケージを使用して作成された多重線形回帰アプリケーションを起動します。この多重線形回帰アプリケーションは、OLS_Regressionというオブジェクトに実装されます。OLS Regressionオブジェクトのサンプル・ファイルは、$ORACLE_HOME/plsql/demoにあります。

小売業者がそのマーケティング・プログラムの有効性を分析する場合を考えます。店舗ごとにそのマーケティング費用が、メディア広告(media)、販促活動(promo)、割引券(disct)およびダイレクト・メール(dmail)の各プログラムに割り当てられます。回帰分析を行って、平均的な店舗におけるある年の売上高(sales)と4つのマーケティング・プログラムにかかる費用との間に線形関係を見出します。マーケティング・データが次の表に格納されているとします。

```
sales_marketing_data (
  /* Store information*/
  store_no    NUMBER,
  year        NUMBER,
  /* Sales revenue (in dollars)*/
  sales       NUMBER,    /* sales amount*/
  /* Marketing expenses (in dollars)*/
  media       NUMBER,    /*media advertisements*/
  promo       NUMBER,    /*promotions*/
  disct       NUMBER,    /*discount coupons*/
  dmail       NUMBER,    /*direct mailers*/
```

ここで係数を基にして、次のような売上高とマーケティングの線形モデルを作成できます。

```
Sales Revenue = a  + b Media Advisements
                   + c Promotions
                   + d Discount Coupons
                   + e Direct Mailer
```

このモデルは、OLS Regressionオブジェクトを参照する次のようなビューとして実装できます。

```
CREATE OR REPLACE VIEW sales_marketing_model (year, ols)
   AS SELECT year,
        OLS_Regression(
        /* mean_y => */
        AVG(sales),
        /* variance_y => */
        var_pop(sales),
        /* MV mean vector => */
        UTL_NLA_ARRAY_DBL (AVG(media),AVG(promo),
                           AVG(disct),AVG(dmail)),
```

```
        /* VCM variance covariance matrix => */
        UTL_NLA_ARRAY_DBL (var_pop(media),covar_pop(media,promo),
                           covar_pop(media,disct),covar_pop(media,dmail),
                           var_pop(promo),covar_pop(promo,disct),
                           covar_pop(promo,dmail),var_pop(disct),
                           covar_pop(disct,dmail),var_pop(dmail)),
        /* CV covariance vector => */
  UTL_NLA_ARRAY_DBL (covar_pop(sales,media),covar_pop(sales,promo),
                           covar_pop(sales,disct),covar_pop(sales,dmail)))
 FROM sales_marketing_data
 GROUP BY year;
```

マーケティング・プログラム・マネージャはこのビューを使用することで、「売上高とマーケティングに関するこのモデルは2004年のデータに適合しているか、つまり、この重相関はある許容値(たとえば0.9)よりも大きいか」などの分析を実行できます。では、相関が複数の場合は、許容値(0.9など)よりも大きくなりますか。これに対応する問合せをSQLで記述するとすれば、次のようになります。

```
SELECT model.ols.getCorrelation(1)
       AS "Applicability of Linear Model"
FROM sales_marketing_model model
WHERE year = 2004;
```

また、次のような問題を解決することもできます。「2003年にマーケティング・プログラムをまったく実施しなかった店舗の予想基準売上高はいくらか」、「2004年のマーケティング・プログラムではどの要素が最も効果的であったか、つまり、費用増分に対する売上高増分の割合が最大であったプログラムはどれか」など。

19.8.3 CASE式

Oracleでは、現在、単純および検索CASE文をサポートしています。CASE文は、目的がDECODE文に似ていますが、DECODE文以上の柔軟性および機能性が提供されます。従来のDECODE文より簡単に理解でき、パフォーマンスも向上します。一般に、カテゴリを年齢などのバケット(たとえば20から29、30から39など)に分割する場合に使用します。

単純CASE文の構文は、次のとおりです。

```
CASE expr WHEN comparison_expr THEN return_expr
 [, WHEN comparison_expr THEN return_expr]... [ELSE else_expr] END
```

単純CASE式は、expr値がcomparison_exprと等価かどうかを調べます。

検索CASE文の構文は、次のとおりです。

```
CASE WHEN condition THEN return_expr [, WHEN condition THEN return_expr]
   ... ELSE else_expr] END
```

検索CASE式では、等価性のみではなく、あらゆる種類の条件を指定できます。

指定できる引数の最大数は65,535です。WHEN ... THENの各組は、2つの引数として数えられます。この制限を超えないようにするには、return_expr自体がCASE式になるようにCASE式をネストします。

関連項目:

CASEを使用してヒストグラムを作成する方法については、CASE文を使用したヒストグラムの作成を参照してください。

例19-29 CASE

ある会社のすべての従業員の平均給与を検索するとします。従業員の給与が2000ドル未満の場合、問合せにはかわりに2000ドルを使用します。CASE文を使用しない場合、この問合せは次のようになります。

```
SELECT AVG(foo(e.salary)) FROM employees e;
```

これは、hrサンプル・スキーマに対して実行する問合せです。この場合のfooは、入力が2001以上の場合はその入力値を、それ以外の場合は2000を戻す関数です。この問合せは、各行で関数を起動する必要があるため、パフォーマンスを考慮する必要があります。また、独自関数を記述すると、開発の負荷も大きくなることがあります。

データベースでPL/SQLを使用せずにCASE式を使用すると、この問合せは次のように記述できます。

```
SELECT AVG(CASE when e.salary > 2000 THEN e.salary ELSE 2000 end)
  AS avg_sal_2k_floor
FROM employees e;
```

CASE式を使用すると、独自関数を開発する必要がなく、高速で実行できます。

例19-30 独立したサブセットを集計するCASE

データの複数のサブセットに対して集計を実行する際、通常のGROUP BYで対応できない場合は、CASEを集計関数の内部で使用できます。たとえば、前述の例において、それぞれ独自のCASE式を持つ複数のAVG列をSELECT構文のリストに含めることができます。このようにすると、給与が0から2000または2000から5000の範囲にある全従業員の平均給与を求める次のような問合せを作成できます。次のようになります。

```
SELECT AVG(CASE WHEN e.sal BETWEEN 0 AND 2000 THEN e.sal ELSE null END) avg2000,
    AVG(CASE WHEN e.sal BETWEEN 2001 AND 5000 THEN e.sal ELSE null END) avg5000
 FROM emps e;
```

この問合せでは、独立したサブセット・データの集計結果を別々の列に表示していますが、CASE式をGROUP BY句に追加すると、集計結果を単一の列の複数の行に表示できます。次の項では、CASEを使用してヒストグラムを作成する2つのアプローチとともに、このアプローチの柔軟性を示します。

19.8.3.1 CASE文を使用したヒストグラムの作成

ユーザー定義バケット(バケット数および各バケットの幅の両方)を含むヒストグラムを作成する場合は、CASE文を使用します。次に、CASE文で作成されたヒストグラムの例を2つ示します。最初の例では、ヒストグラムの合計が複数の列に示され、単一の行が戻されます。2番目の例では、ヒストグラムはラベル列および単一の合計列で示され、複数の行が戻されます。

例19-31 ヒストグラムの例1

```
SELECT SUM(CASE WHEN cust_credit_limit BETWEEN  0 AND 3999 THEN 1 ELSE 0 END)
  AS "0-3999",
SUM(CASE WHEN cust_credit_limit BETWEEN  4000 AND 7999 THEN 1 ELSE 0 END)
  AS "4000-7999",
SUM(CASE WHEN cust_credit_limit BETWEEN  8000 AND 11999 THEN 1 ELSE 0 END)
  AS "8000-11999",
SUM(CASE WHEN cust_credit_limit BETWEEN  12000 AND 16000 THEN 1 ELSE 0 END)
  AS "12000-16000"
FROM customers WHERE cust_city = 'Marshal';

    0-3999  4000-7999 8000-11999 12000-16000
---------- ---------- ---------- -----------
         8          7          7           1
```

例19-32 ヒストグラムの例2

```
SELECT (CASE WHEN cust_credit_limit BETWEEN  0 AND 3999 THEN  ' 0 - 3999'
   WHEN cust_credit_limit BETWEEN  4000 AND 7999 THEN ' 4000 - 7999'
```

```
   WHEN cust_credit_limit BETWEEN  8000 AND 11999 THEN  ' 8000 - 11999'
   WHEN cust_credit_limit BETWEEN  12000 AND 16000 THEN '12000 - 16000' END)
  AS BUCKET, COUNT(*) AS Count_in_Group
FROM customers WHERE cust_city = 'Marshal' GROUP BY
 (CASE WHEN cust_credit_limit BETWEEN  0 AND 3999 THEN ' 0 - 3999'
 WHEN cust_credit_limit BETWEEN  4000 AND 7999 THEN ' 4000 - 7999'
 WHEN cust_credit_limit BETWEEN  8000 AND 11999 THEN  ' 8000 - 11999'
 WHEN cust_credit_limit BETWEEN  12000 AND 16000 THEN '12000 - 16000' END);

BUCKET         COUNT_IN_GROUP
-------------- --------------
 0 - 3999                   8
 4000 - 7999                7
 8000 - 11999               7
12000 - 16000               1
```

19.8.4 SQL分析の高頻度項目セット

一定のイベントがどのくらいの頻度で発生するか(たとえば、ある顧客が食料品店でどのくらいの頻度でミルクを購入するか)をカウントするかわりに、複数のイベントがどのくらいの頻度で同時に発生するか(たとえば、ある顧客が食料品店でどのくらいの頻度でミルクとシリアルを同時に購入するか)をカウントできます。こうした複数のイベントをカウントするには、高頻度項目セット(名前のとおり複数項目のセット)というものを使用します。項目セットの例としては、特定の顧客が食料品店での1回の買い物で購入したすべての製品(通常はマーケット・バスケットと呼ばれる)、1回のセッションでユーザーがアクセスしたWebページ、特定の顧客が利用する金融サービスなどがあります。

高頻度項目セットを使用する実際の目的は、最も頻繁に発生する項目セットを見つけることにあります。食料品店のPOSデータを分析する場合であれば、たとえば、同時に購入される項目の組として最も多いものはミルクとバナナであるということがわかるわけです。このため、高頻度項目セットは長い間、小売業界のビジネス・インテリジェンス環境で最も一般的なマーケット・バスケット分析用ツールとして使用されてきました。高頻度項目セットの計算機能はデータベースに統合されています。操作はリレーショナル表の上で行われ、SQLを介してアクセスします。データベースと統合することで、次のような大きな利点があります。

- これまで高頻度項目セット操作に依存していたアプリケーションの場合は、より単純な実装が可能になり、パフォーマンスが大幅に向上します。
- これまで高頻度項目セットを使用していなかったSQLベース・アプリケーションの場合は、簡単に拡張してこの機能を利用できるようになります。

高頻度項目セットの分析は、PL/SQLパッケージDBMS_FREQUENT_ITEMSETSで実行されます。詳細は、*『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』*を参照してください。また、高頻度項目セットの使用例はビジネス・インテリジェンス問合せの例4: 高頻度項目セットを参照してください。

19.9 SQLの行の制限

特定の行数または行の割合によって、SQL問合せから戻される行を制限できます。場合によっては、問合せ結果を順序付けてから戻される行数を制限する必要が生じることもあります。最初に行をソートしてから戻す行数を制限する問合せは、通常、上位N問合せと呼ばれます。これにより、「給料の高い従業員の上位10人はだれか」などのような基本的な質問のレポートまたは単純なビューを簡単に作成できます。これは、参照用にデータセットの最初の数行を示すユーザー・インタフェースにも役立ちます。上位N問合せを発行する際に、オフセットを指定することもできます。オフセットにより、問合せ結果セットの先頭の数行が除外されます。すると問合せは、オフセットの後の最初の行から、指定の数または割合の行を戻します。オフセットにより、一般的な質問を変更し、最も給料の高い従業員に関する質問で上位10人の従業員をスキップして給料のランキングが11位から20位までの従業員のみが戻されるようにすることができます。同様に、給料ごとに従業員の問合せを行い、上位10人の従業員をスキップして残りの従業員のうち上位10%を戻すこともできます。

戻される行を制限する問合せは、かねてより、ROW_NUMBERウィンドウ関数、ROWNUM疑似列およびその他の手法を使用して実現可能でしたが、*row_limiting_clause*のANSI SQL標準構文を使用して、より簡単に記述できるようになりました。この句を使用する際、問合せにORDER BY句を含めることによって、上位N問合せで必要となる確定的なソート順を保証できます。*row_limiting_clause*句は、SELECTの最後の部分として、ORDER BY句の後に置かれ、FETCHまたはOFFSETのいずれかで開始されます。基本的な構文は次のようになります。

```
[ OFFSET offset { ROW | ROWS } ]
[ FETCH { FIRST | NEXT } [ { rowcount | percent PERCENT } ]
    { ROW | ROWS } { ONLY | WITH TIES } ]
```

この構文については、次の項で例示します。

OFFSET

これは、行の制限が開始される前にスキップする行数を指定します。*offset*は数値である必要があります。負の数を指定すると、オフセットは0と見なされます。NULLを指定したり、問合せで戻される行数以上の数を指定すると、戻される行数は0行となります。*offset*に少数が含まれている場合、その少数部分は切り捨てられます。この句を指定しない場合、*offset*は0で、行の制限は最初の行から開始されます。読みやすくするため、Oracleでは、ROWまたはROWS(これらは同等です)を使用するオプションを提供しています。

FETCH

これは、戻す行数または行の割合を指定します。この句を指定しない場合、*offset* + 1行目からすべての行が戻されます。WITH TIESキーワードを使用すると、問合せには、限定された最後の行のソート・キーと一致するすべての行も含まれます。

問合せで戻される行数の制限方法について説明するため、次の文について考えてみましょう。

```
SELECT employee_id, last_name
FROM employees
ORDER BY employee_id
FETCH FIRST 5 ROWS ONLY;

EMPLOYEE_ID LAST_NAME
----------- ----------
        100 King
        101 Kochhar
        102 De Haan
        103 Hunold
        104 Ernst
```

この文では、employee_idの値が小さい方から5人の従業員が戻されます。

次の5人を戻すには、この文にOFFSETを追加します。

```
SELECT employee_id, last_name
FROM employees
ORDER BY employee_id
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;

EMPLOYEE_ID LAST_NAME
----------- ----------
        105 Austin
        106 Pataballa
        107 Lorentz
        108 Greenberg
        109 Faviet
```

この文で、FETCH FIRSTとFETCH NEXTは同等ですが、OFFSETが使用されている場合には、FETCH NEXTの方が明確です。

オフセットは、次の文のように、10などの大きな値にできます。

```
SELECT employee_id, last_name
FROM employees
ORDER BY employee_id
OFFSET 10 ROWS FETCH NEXT 5 ROWS ONLY;

EMPLOYEE_ID LAST_NAME
----------- ----------
        110 Chen
        111 Sciarra
        112 Urman
        113 Popp
        114 Raphaely
```

固定の数ではなく割合で値を戻すようにすることもできます。これを示すため、次の文では、給料が下位5%の従業員を戻します。

```
SELECT employee_id, last_name, salary
FROM employees
ORDER BY salary
FETCH FIRST 5 PERCENT ROWS ONLY;

EMPLOYEE_ID LAST_NAME             SALARY
----------- -------------------- ------
        132 Olson                  2100
        128 Markle                 2200
        136 Philtanker             2200
        127 Landry                 2400
        135 Gee                    2400
        119 Colmenares             2500
```

この結果セットでは、5%は6行です。OFFSETを使用する場合、割合の計算はオフセットが適用される前の結果セット全体に基づくため、これが重要となります。次の文は、OFFSETの使用例です。

```
SELECT employee_id, last_name, salary
FROM employees
ORDER BY salary, employee_id
OFFSET 6 ROWS FETCH FIRST 5 PERCENT ROWS ONLY;

EMPLOYEE_ID LAST_NAME             SALARY
----------- -------------------- ------
        131 Marlow                 2500
        140 Patel                  2500
        144 Vargas                 2500
        182 Sullivan               2500
        191 Perkins                2500
        118 Himuro                 2500
```

この文では依然として6行が戻されますが、結果セットの7番目の行から開始されます。ORDER BY句に追加される追加のemployee_idは、確定的なソートを保証するためのものでした。

WITH TIESを使用して、同等の値を戻すオプションもあります。これによって、給料が下位5%の従業員に加え、最後にフェッチされた行と同じ給料のすべての従業員も戻されます。

```
SELECT employee_id, last_name, salary
FROM employees
ORDER BY salary
FETCH FIRST 5 PERCENT ROWS WITH TIES;
```

```
EMPLOYEE_ID LAST_NAME             SALARY
----------- -------------------- ------
        132 Olson                  2100
        128 Markle                 2200
        136 Philtanker             2200
        127 Landry                 2400
        135 Gee                    2400
        119 Colmenares             2500
        131 Marlow                 2500
        140 Patel                  2500
        144 Vargas                 2500
        182 Sullivan               2500
        191 Perkins                2500
```

同じ問合せを発行することもできますが、次の文を使用すると、最初の5つの値がスキップされます。

```
SELECT employee_id, last_name, salary
FROM employees
ORDER BY salary
OFFSET 5 ROWS FETCH FIRST 5 PERCENT ROWS WITH TIES;

EMPLOYEE_ID LAST_NAME             SALARY
----------- -------------------- ------
        119 Colmenares             2500
        131 Marlow                 2500
        140 Patel                  2500
        144 Vargas                 2500
        182 Sullivan               2500
        191 Perkins                2500
```

19.9.1 SQLの行の制限における制限事項および考慮事項

*row_limiting_clause*句には、次の制限事項があります。

- この句は、*for_update_clause*と同時には指定できません。
- この句を指定すると、順序疑似列のCURRVALまたはNEXTVALを、選択リストに含めることができなくなります。
- 定義する問合せにこの句が含まれている場合、マテリアライズド・ビューは増分リフレッシュに適応しません。

関連項目:

構文および制限の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

20 データ・ウェアハウスにおける集計のためのSQL

この章では、データ・ウェアハウスの基本的な側面であるSQLによる集計処理について説明します。この項の内容は、次のとおりです。

- データ・ウェアハウスにおける集計SQLの概要
- ROLLUP(GROUP BYの拡張)
- CUBE(GROUP BYの拡張)
- GROUPING関数
- GROUPING SETS式
- 複合列とグルーピングについて
- 連結グルーピングとデータ集計について
- データ・ウェアハウスで集計を使用する場合の考慮事項
- WITH句を使用した計算
- SQLでの階層的キューブの処理

20.1 データ・ウェアハウスにおける集計SQLの概要

集計は、データ・ウェアハウスの基本的な処理です。ウェアハウスにおける集計パフォーマンスを向上させるため、Oracle Databaseでは次の機能が用意されています。

- GROUP BY句を拡張するCUBEおよびROLLUP
- 3つのGROUPING関数
- GROUPING SETS式
- ピボット操作

SQLに対するCUBE、ROLLUPおよびGROUPING SETS拡張により、問合せとレポートがより簡単で高速になります。CUBE、ROLLUPおよびグルーピング・セットでは、行を様々にグルーピングした文のUNION ALLと同じ単一の結果セットが生成されます。ROLLUPでは、SUM、COUNT、MAX、MINおよびAVGなどの集計を、最も詳細なものから総計まで、レベルを上げながら集計が作成されます。CUBEはROLLUPと同様の拡張で、一文で集計可能なすべての組合せについて計算を行うことができます。CUBE、ROLLUPおよびGROUPING SETS拡張を使用すると、GROUP BY句で必要なグルーピングを指定できます。これにより、CUBE操作を実行せずに複数のディメンション間で効率的に分析できます。CUBEの計算は、大きな処理負荷が生じますが、キューブをグルーピング・セットに置き換えると、パフォーマンスを大幅に改善できます。

パフォーマンスを向上させるために、CUBE、ROLLUPおよびGROUPING SETSをパラレル化できます。つまり、複数のプロセスで、すべての文を同時に実行できます。これらの機能によって集計計算がより効率的になるため、データベースのパフォーマンスおよびスケーラビリティが向上します。

3つのGROUPING関数を使用すると、各行が属するグループを識別し、小計行をソートして結果にフィルタを適用できます。

この項では、次の項目について説明します。

- 複数ディメンション間の分析について
- 集計パフォーマンスの最適化について

- データ・ウェアハウジング: 集計のシナリオ

20.1.1 複数ディメンション間の分析について

意思決定支援システムの重要な概念の1つは、多次元分析です。必要なディメンションをすべて組み合せて企業を調査します。ディメンションという用語は、質問の指定に使用される任意のカテゴリという意味で使用されます。最も一般的には、時間、地理、製品、部門、流通チャネルなどのディメンションが指定されますが、企業活動が多方面にわたるのと同様に、可能なディメンションの数にも制限はありません。特定のディメンション値の集合に対応付けられたイベントまたはエンティティは、通常、ファクトと呼ばれます。ファクトには、売上件数または国内通貨での売上金額、利益、顧客数、生産量など、追跡する価値があるすべてのものが含まれます。

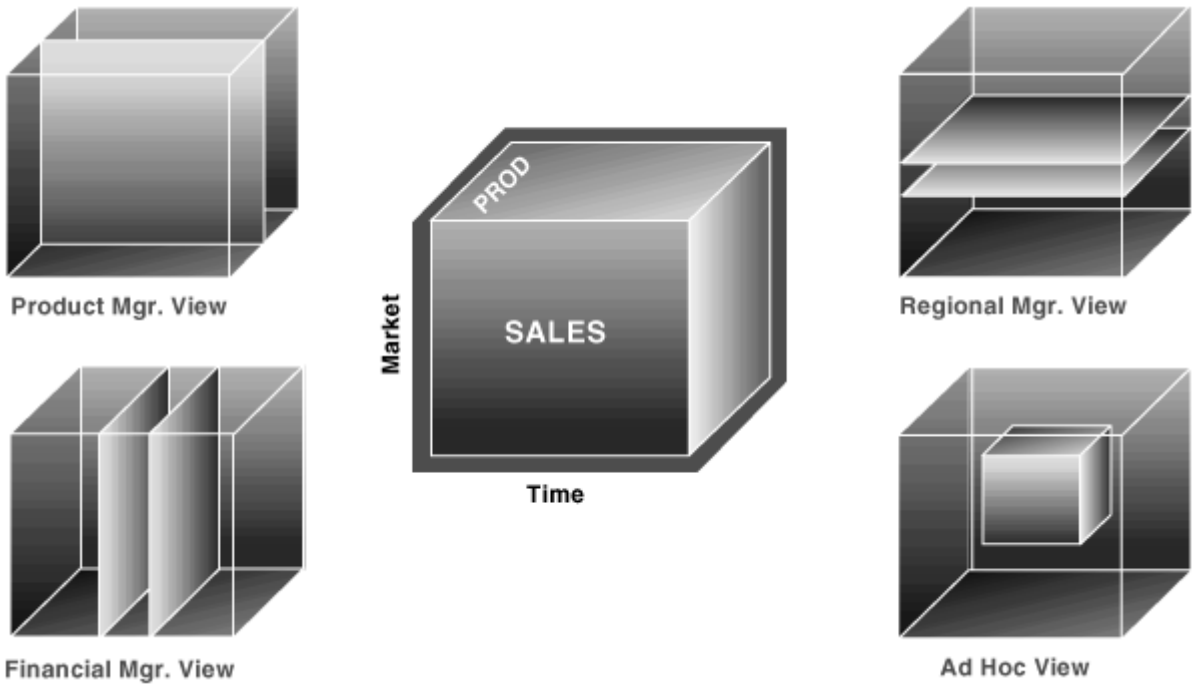
多次元的な問合せの例を次に示します。

- 1999年および2000年について、すべての製品の総売上を、州から国、地域単位へと地理ディメンションの集計レベルを上げながら表示します。
- 1999年と2000年における南アメリカの地域別経費を示す事業のクロス集計分析を作成します。可能な小計をすべて組み込みます。
- 自動車製品に関する2000年の販売収入に従って、アジアでの販売代理店の上位10社をリストし、そのコミッションのランキングを作成します。

これらの問合せには、すべて複数のディメンションが伴います。多次元による多くの質問では、通常は時間、地理または予算にまたがって集計されたデータと、データ・セットの比較が必要になります。

アナリストは一般に、多数のディメンションを持つデータを視覚的に表現する際、データ・キューブ(n個のディメンションの共通部分にファクトが格納される領域)を使用します。図20-1に、あるデータ・キューブと、このデータ・キューブが様々なグループによって異なる方法で利用される様子を示します。キューブには、製品、市場、売上および時間のディメンションで編成された売上データが格納されています。これが単方向ディメンションであることに注意してください。実際のデータは通常の表に物理的に格納されます。キューブ・データは、詳細データと集計データの両方で構成されます。

図20-1 論理キューブおよび異なるユーザーごとのビュー



キューブからデータのスライスを取り出すことができます。これらは、表20-1に示すようなクロス集計レポートに対応します。地域マネージャは、異なる市場に適用されるキューブ・スライスを比較することで、データを解析します。これとは対照的に、製品マネージャは、異なる製品に適用するスライスを比較します。非定型作業を行うユーザーは、サブセット・キューブ内で、様々に

データを絞り込んで処理できます。

多次元の質問への回答には、多くの場合、数百万行にもなる膨大な量のデータへのアクセスおよび問合せが伴います。巨大な組織によって生成される大量のディテール・データは、最低レベルでは解析できないため、情報の集計ビューが不可欠です。合計やカウントなど、多数のディメンションにまたがる集計は、多次元分析にとってきわめて重要です。したがって、分析作業には、便利で効率的なデータ集計が必要となります。

20.1.2 最適化された集計のパフォーマンスについて

多次元での処理のみでなく、すべてのタイプの処理が、拡張された集計機能の効果を得ることができます。トランザクション処理システムや、財務システム、製造システムでは、大量のシステム・リソースを必要とする膨大な数の成果レポートが生成されます。これらのレポート作成時の効率が向上することで、システムの負荷が削減されます。実際、データを詳細レベルから高度なレベルまで集計する場合には、どのようなコンピュータ処理でも集計パフォーマンスの最適化によるメリットが得られます。

これらの拡張された集計機能の提供によって、次に示すような多くの効果が得られます。

- 大量の作業にも少量のSQLコードしか必要としない単純化されたプログラム
- より高速で高効率の問合せ処理
- 集計作業がサーバー側に移行されることによる、クライアント処理の負荷およびネットワーク通信量の削減
- 類似した問合せで既存の作業を効率化できることによる、集計のキャッシング機会の増加

20.1.3 データ・ウェアハウス: 集計シナリオ

GROUP BY拡張の使用例を示すために、この章ではサンプル・スキーマのshのデータを使用します。この章の例はすべて、この会社のデータを例として使用します。この架空の会社は世界中で販売を行っており、売上を金額情報と数量情報の両面から追跡しています。多数のデータ行があるため、問合せは、この例のように通常はWHERE句で厳密に制限され、結果は少数の行に限定されます。

表20-1は、クロス集計レポートの例です。このレポートは、2000年9月のインターネット販売および直接販売における、米国(US)とフランス(France)のcountry_idおよびchannel_desc別の総売上を示しています。

表20-1 単純なクロス集計レポート(小計付き)

チャネル	フランス	米国	合計
インターネット	9,597	124,224	133,821
直接販売	61,202	638,201	699,403
合計	70,799	762,425	833,224

値の数が9個のみのこのような単純なレポートでも、4つの小計および1つの総計が生成されています。このレポートに必要な値の半数は、SUM(amount_sold)を要求してGROUP BY(channel_desc, country_id)を行うような問合せのみでは計算されません。上位レベルの集計を取得するには、さらに問合せが必要になります。小計の計算について改善されたデータベース・コマンドによって、問合せ、レポートおよび分析的な操作で大きな効果を得ることができます。

```
SELECT channels.channel_desc, countries.country_iso_code,
  TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id AND
  sales.channel_id= channels.channel_id AND channels.channel_desc IN
```

```
  ('Direct Sales', 'Internet') AND times.calendar_month_desc='2000-09'
  AND customers.country_id=countries.country_id
  AND countries.country_iso_code IN ('US','FR')
GROUP BY CUBE(channels.channel_desc, countries.country_iso_code);

CHANNEL_DESC         CO SALES$
-------------------- -- --------------
                           833,224
                     FR     70,799
                     US    762,425
Internet                   133,821
Internet             FR      9,597
Internet             US    124,224
Direct Sales               699,403
Direct Sales         FR     61,202
Direct Sales         US    638,201
```

集計の例でのNULLの解釈

GROUP BYの拡張機能によって戻されるNULLは、不明値を意味する従来のNULLとは限りません。このNULLは、その行が小計であることを示す場合があります。データベース・システムに値以外のものを導入するのを避けるため、こうした小計値には特別なタグは付けられていません。

関連項目:

小計を表すNULLとデータに格納されるNULLを区別する方法の詳細は、GROUPING関数を参照してください。

20.2 ROLLUP(GROUP BYの拡張)

ROLLUPを使用すると、SELECT文により、指定したディメンション・グループの小計を複数のレベルで計算できます。総計も計算できます。ROLLUPは、GROUP BY句の単純な拡張であるため、その構文は非常に簡単です。ROLLUPによる拡張は非常に効率的で、問合せにかかるオーバーヘッドは最小限に抑えられます。

ROLLUPのアクションは簡単です。これは、最も詳細なレベルから総計まで、ROLLUP句で指定されたグループ・リストに従ってロールアップする小計を作成します。ROLLUPは、その引数として、グルーピング列の順序付けリストを取ります。最初に、GROUP BY句で指定された標準の集計値を計算します。次に、グルーピング列のリストを右から左に移動しながら、順番に高いレベルの小計を作成します。最後に、総計を作成します。

ROLLUPは、n+1のレベルで小計を作成します。ここで、nはグルーピング列の数です。たとえば、time、regionおよびdepartment(n=3)のグルーピング列でROLLUPを指定した問合せの場合、結果セットには4つの集計レベルの行が含まれます。

ROLLUPを使用するときにデータの圧縮が必要となることがあります。これは、古いパーティションに対する更新が少ない場合に特に役立ちます。

この項では、次の項目について説明します。

- ROLLUPを使用するとき
- ROLLUPの構文
- 部分的ROLLUP

20.2.1 ROLLUPを使用するとき

小計を伴う作業では、ROLLUPによる拡張を使用します。

- 時間や地理などの階層的なディメンションに従って小計する場合に非常に有効です。たとえば、問合せでROLLUP(y, m, day)またはROLLUP(country, state, city)のように指定できます。
- サマリー表を使用しているデータ・ウェアハウス管理者の場合は、ROLLUPを使用することでサマリー表のメンテナンスが簡単になり、時間を短縮できます。

20.2.2 ROLLUPの構文

ROLLUPは、SELECT文のGROUP BY句で使用します。形式は次のとおりです。

```
SELECT … GROUP BY ROLLUP(grouping_column_reference_list)
```

例20-1 ROLLUP

この例では、shサンプル・スキーマ・データを使用します。このデータは、図20-1で使用されているものと同じです。ROLLUPは、3つのディメンションにまたがっています。

```
SELECT channels.channel_desc, calendar_month_desc,
       countries.country_iso_code,
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id
  AND sales.cust_id=customers.cust_id
  AND customers.country_id = countries.country_id
  AND sales.channel_id = channels.channel_id
  AND channels.channel_desc IN ('Direct Sales', 'Internet')
  AND times.calendar_month_desc IN ('2000-09', '2000-10')
  AND countries.country_iso_code IN ('GB', 'US')
GROUP BY
  ROLLUP(channels.channel_desc, calendar_month_desc, countries.country_iso_code);

CHANNEL_DESC         CALENDAR CO SALES$
-------------------- -------- -- --------------
Internet             2000-09  GB          16,569
Internet             2000-09  US         124,224
Internet             2000-09             140,793
Internet             2000-10  GB          14,539
Internet             2000-10  US         137,054
Internet             2000-10             151,593
Internet                                 292,387
Direct Sales         2000-09  GB          85,223
Direct Sales         2000-09  US         638,201
Direct Sales         2000-09             723,424
Direct Sales         2000-10  GB          91,925
Direct Sales         2000-10  US         682,297
Direct Sales         2000-10             774,222
Direct Sales                           1,497,646
                                       1,790,032
```

丸めのため、結果が常に加算されるとは限らないことに注意してください。

この問合せでは、次の行集合が戻されます。

- ROLLUPを使用しないでGROUP BYによって生成される通常の集計行。

- channel_descとcalendar_monthの組合せごとに、country_idをまたがって集計される第1レベルの小計。
- channel_desc値ごとに、calendar_month_descとcountry_idをまたがって集計される第2レベルの小計。
- 総計行。

Live SQL:

関連する例を表示および実行するには、Oracle Live SQL (Oracle LiveSQL: GROUP BY を使用する ROLLUP)にアクセスしてください。

20.2.3 部分的ROLLUP

一部の小計のみを含めるためのロールアップもできます。このような部分的ロールアップで使用する構文は次のとおりです。

```
GROUP BY expr1, ROLLUP(expr2, expr3);
```

この場合、GROUP BY句は3つ(2+1)の集計レベルで小計を作成します。つまり、(expr1, expr2, expr3)のレベル、(expr1, expr2)のレベル、および(expr1)のレベルです。

例20-2 部分的ROLLUP

```
SELECT channel_desc, calendar_month_desc, countries.country_iso_code,
   TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id
  AND customers.country_id = countries.country_id
  AND sales.channel_id= channels.channel_id
  AND channels.channel_desc IN ('Direct Sales', 'Internet')
  AND times.calendar_month_desc IN ('2000-09', '2000-10')
  AND countries.country_iso_code IN ('GB', 'US')
GROUP BY channel_desc, ROLLUP(calendar_month_desc, countries.country_iso_code);

CHANNEL_DESC         CALENDAR CO SALES$
-------------------- -------- -- --------------
Internet             2000-09  GB          16,569
Internet             2000-09  US         124,224
Internet             2000-09             140,793
Internet             2000-10  GB          14,539
Internet             2000-10  US         137,054
Internet             2000-10             151,593
Internet                                 292,387
Direct Sales         2000-09  GB          85,223
Direct Sales         2000-09  US         638,201
Direct Sales         2000-09             723,424
Direct Sales         2000-10  GB          91,925
Direct Sales         2000-10  US         682,297
Direct Sales         2000-10             774,222
Direct Sales                           1,497,646
```

この問合せでは、次の行集合が戻されます。

- ROLLUPを使用しないでGROUP BYによって生成される通常の集計行。
- channel_descとcalendar_month_descの組合せごとに、country_idをまたがって集計される第1レベルの小計。

- channel_desc値ごとに、calendar_month_descとcountry_idをまたがって集計される第2レベルの小計。
- 総計行は生成されません。

20.3 CUBE(GROUP BYの拡張)

CUBEは、指定されたグルーピング列の集合を取り、それらが取り得るすべての組合せに対して小計を作成します。多次元分析の観点では、CUBEは、指定されたディメンションを持つデータ・キューブに対して計算されるすべての小計を生成します。CUBE(time, region, department)を指定した場合、結果セットには、同等のROLLUP文および追加組合せ内にある値がすべて含まれます。たとえば、図20-1では、すべての地域にわたる部門の合計(279,000および319,000)はROLLUP(time, region, department)句では計算されませんが、CUBE(time, region, department)句では計算されます。*CUBE*に対して指定された列がn個ある場合、戻される小計の組合せは2から*n*個になります。CUBEの構文に、3つのディメンションを持つキューブの例を示します。

関連項目:

構文および制限の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

この項では、次の項目について説明します。

- CUBEを使用するとき
- CUBEの構文
- 部分的CUBE
- CUBEを使用しない小計の計算

20.3.1 CUBEを使用するとき

クロス集計レポートを必要とする状況では、CUBEの使用を考慮してください。クロス集計レポートに必要なデータは、CUBEを使用して単一のSELECTで作成できます。ROLLUPと同様に、CUBEもサマリー表の作成に有効です。CUBE問合せがパラレルに実行されるよりサマリー表を利用した方が高速です。

CUBEは、通常、1つのディメンションの異なるレベルを表す列を使用する問合せより、複数のディメンションの列を使用する問合せに最も適しています。たとえば、一般的に要求されるクロス集計作成では、month、stateおよびproductのすべての組合せに対する小計が必要です。これらは、3つの独立したディメンションであり、取り得るすべての組合せに対する小計を処理した分析が一般的です。反対に、year、monthおよびdayが取り得るすべての組合せを示すクロス集計作成では、timeディメンションに階層があるため、必要な値はいくつかに限られています。年間を通して合計された、毎月の日別利益のような小計は、ほとんどの分析では必要ありません。「年間の毎月16日の総売上はいくらか」といった質問を必要とするユーザーは、比較的少数です。ロールアップ計算を効率的に処理する例は、ROLLUPおよびCUBEでの階層処理を参照してください。

20.3.2 CUBEの構文

CUBEは、SELECT文のGROUP BY句で使用します。形式は次のとおりです。

```
SELECT …  GROUP BY CUBE (grouping_column_reference_list)
```

例20-3 問合せにおけるCUBEキーワード

```
SELECT channel_desc, calendar_month_desc, countries.country_iso_code,
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
```

```
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id AND
  sales.channel_id= channels.channel_id
 AND customers.country_id = countries.country_id
 AND channels.channel_desc IN
  ('Direct Sales', 'Internet') AND times.calendar_month_desc IN
  ('2000-09', '2000-10') AND countries.country_iso_code IN ('GB', 'US')
GROUP BY CUBE(channel_desc, calendar_month_desc, countries.country_iso_code);

CHANNEL_DESC         CALENDAR CO SALES$
-------------------- -------- -- --------------
                                    1,790,032
                              GB      208,257
                              US    1,581,775
                    2000-09           864,217
                    2000-09   GB      101,792
                    2000-09   US      762,425
                    2000-10           925,815
                    2000-10   GB      106,465
                    2000-10   US      819,351
Internet                              292,387
Internet                      GB       31,109
Internet                      US      261,278
Internet            2000-09           140,793
Internet            2000-09   GB       16,569
Internet            2000-09   US      124,224
Internet            2000-10           151,593
Internet            2000-10   GB       14,539
Internet            2000-10   US      137,054
Direct Sales                        1,497,646
Direct Sales                  GB      177,148
Direct Sales                  US    1,320,497
Direct Sales        2000-09           723,424
Direct Sales        2000-09   GB       85,223
Direct Sales        2000-09   US      638,201
Direct Sales        2000-10           774,222
Direct Sales        2000-10   GB       91,925
Direct Sales        2000-10   US      682,297
```

この問合せは、3つのディメンションにまたがるCUBE集計を示しています。

20.3.3 部分的CUBE

部分的CUBEは、特定のディメンションに制限してCUBE演算子の外側の列に進むという点で、部分的ROLLUPに似ています。この場合、取り得るすべての組合せに対する小計は、CUBEリスト内(カッコ内)のディメンションに制限され、GROUP BYリスト内の前の項目と組み合されます。

部分的CUBEの構文は、次のとおりです。

```
GROUP BY expr1, CUBE(expr2, expr3)
```

この構文例では2×2、つまり次の4つの小計が計算されます。つまり、次のようになります。

- (expr1, expr2, expr3)
- (expr1, expr2)
- (expr1, expr3)
- (expr1)

例20-4 問合せにおける部分的CUBE

salesデータベースを使用して、次の文を発行できます。

```
SELECT channel_desc, calendar_month_desc, countries.country_iso_code,
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id = times.time_id
  AND sales.cust_id = customers.cust_id
  AND customers.country_id=countries.country_id
  AND sales.channel_id = channels.channel_id
  AND channels.channel_desc IN ('Direct Sales', 'Internet')
  AND times.calendar_month_desc IN ('2000-09', '2000-10')
  AND countries.country_iso_code IN ('GB', 'US')
GROUP BY channel_desc, CUBE(calendar_month_desc, countries.country_iso_code);

CHANNEL_DESC         CALENDAR CO SALES$
-------------------- -------- -- --------------
Internet                              292,387
Internet                      GB       31,109
Internet                      US      261,278
Internet             2000-09          140,793
Internet             2000-09  GB       16,569
Internet             2000-09  US      124,224
Internet             2000-10          151,593
Internet             2000-10  GB       14,539
Internet             2000-10  US      137,054
Direct Sales                        1,497,646
Direct Sales                  GB      177,148
Direct Sales                  US    1,320,497
Direct Sales         2000-09          723,424
Direct Sales         2000-09  GB       85,223
Direct Sales         2000-09  US      638,201
Direct Sales         2000-10          774,222
Direct Sales         2000-10  GB       91,925
Direct Sales         2000-10  US      682,297
```

20.3.4 CUBEを使用しない小計の計算

ROLLUPの場合と同様に、UNION ALL文と組み合された複数のSELECT文によって、CUBEを使用する場合と同じ情報が収集できます。ただし、この場合は多数のSELECT文が必要となります。nディメンションのキューブの場合、2から*n*個のSELECT文が必要です。3ディメンションの場合も、UNION ALLでリンクされたSELECT文を発行することになります。SELECT文が多すぎるため、処理の効率が悪くなり、SQL文が極端に長くなります。

可能なすべての組合せを計算する際にディメンションを1つのみ追加すると、どのような影響があるかを考えてみます。SELECT文の数は、2倍の16になります。CUBE句で使用される列が増加するほど、UNION ALLを使用する方法と比較した場合の効果も大きくなります。

20.4 GROUPING関数

ROLLUPおよびCUBEの使用については、2つの課題があります。第1の課題は、どの結果セット行が小計であるかをプログラム上でどのように判断するか、および指定された小計の正確な集計レベルをどのように探し出すかということです。合計に対する割合などを計算する場合に小計がよく必要となるため、どの行が求める小計であるかを判断する簡単な方法が必要です。第2の課題は、格納されるNULL値と、ROLLUPまたはCUBEによって作成される「NULL」値の両方が参照結果に含まれる場合、どう処理するかということです。この2つをどのように区別するかが問題になります。ここでは、このような場合について例をあげて説明します。

関連項目:

構文および制限の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

この項では、次の項目について説明します。

- GROUPING関数
- GROUPINGを使用するとき
- GROUPING_ID関数
- GROUP_ID関数

20.4.1 GROUPING関数

GROUPINGは、このような問題を処理します。単一の列を引数として使用し、ROLLUPまたはCUBE操作によってNULLが作成された場合に、GROUPINGは1を戻します。つまり、NULLが小計の行であることを示す場合、GROUPINGは1を戻します。格納されたNULLなど、その他のタイプの値では0(ゼロ)を戻します。

GROUPINGは、SELECT文のリスト部分で使用します。形式は次のとおりです。

```
SELECT …  [GROUPING(dimension_column)…]  …
  GROUP BY …    {CUBE | ROLLUP| GROUPING SETS}  (dimension_column)
```

例20-5 GROUPINGによる列のマスク

次の例では、GROUPINGを使用して、例20-2に示した結果セットに対する一連のマスク列を作成します。マスク列は、プログラムで簡単に分析できます。

```
SELECT channel_desc, calendar_month_desc, country_iso_code,
TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$, GROUPING(channel_desc) AS Ch,
   GROUPING(calendar_month_desc) AS Mo, GROUPING(country_iso_code) AS Co
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id
  AND sales.cust_id=customers.cust_id
  AND customers.country_id = countries.country_id
  AND sales.channel_id= channels.channel_id
  AND channels.channel_desc IN ('Direct Sales', 'Internet')
  AND times.calendar_month_desc IN ('2000-09', '2000-10')
  AND countries.country_iso_code IN ('GB', 'US')
GROUP BY ROLLUP(channel_desc, calendar_month_desc, countries.country_iso_code);

CHANNEL_DESC         CALENDAR CO SALES$               CH         MO         CO
-------------------- -------- -- -------------- ---------- ---------- ----------
Internet             2000-09  GB          16,569          0          0          0
Internet             2000-09  US         124,224          0          0          0
Internet             2000-09             140,793          0          0          1
Internet             2000-10  GB          14,539          0          0          0
Internet             2000-10  US         137,054          0          0          0
Internet             2000-10             151,593          0          0          1
Internet                                 292,387          0          1          1
Direct Sales         2000-09  GB          85,223          0          0          0
Direct Sales         2000-09  US         638,201          0          0          0
Direct Sales         2000-09             723,424          0          0          1
Direct Sales         2000-10  GB          91,925          0          0          0
Direct Sales         2000-10  US         682,297          0          0          0
```

```
Direct Sales          2000-10              774,222          0          0          1
Direct Sales                             1,497,646          0          1          1
                                         1,790,032          1          1          1
```

プログラムは、T、RおよびD列に対するマスク「0 0 0」によって、ディテール行を簡単に識別できます。第1レベルの小計行は「0 0 1」のマスクを持ち、第2レベルの小計行はマスク「0 1 1」、全体の総計行はマスク「1 1 1」を持ちます。

例20-6に示すようにGROUPINGおよびDECODE関数を使用して、結果セットを読みやすくできます。

例20-6 可読性を高めるためのGROUPING

```
SELECT DECODE(GROUPING(channel_desc), 1, 'Multi-channel sum', channel_desc) AS
 Channel, DECODE (GROUPING (country_iso_code), 1, 'Multi-country sum',
 country_iso_code) AS Country, TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id
  AND sales.cust_id=customers.cust_id
  AND customers.country_id = countries.country_id
  AND sales.channel_id= channels.channel_id
  AND channels.channel_desc IN ('Direct Sales', 'Internet')
  AND times.calendar_month_desc= '2000-09'
  AND country_iso_code IN ('GB', 'US')
GROUP BY CUBE(channel_desc, country_iso_code);

CHANNEL              COUNTRY           SALES$
-------------------- ----------------- --------------
Multi-channel sum    Multi-country sum        864,217
Multi-channel sum    GB                       101,792
Multi-channel sum    US                       762,425
Internet             Multi-country sum        140,793
Internet             GB                        16,569
Internet             US                       124,224
Direct Sales         Multi-country sum        723,424
Direct Sales         GB                        85,223
Direct Sales         US                       638,201
```

前述の文を理解するために、channel_desc列を処理する最初の列指定に注目してください。前述の文の最初の行を考えてみます。

```
SELECT DECODE(GROUPING(channel_desc), 1, 'Multi-Channel sum', channel_desc)AS Channel
```

ここで、channel_descの値は、GROUPING関数を含むDECODE関数で決定されます。行の値がROLLUPまたはCUBEによって作成された集計である場合、GROUPING関数は1を戻し、それ以外の場合は0(ゼロ)を戻します。次に、DECODE関数は、GROUPING関数の結果を処理します。1が戻された場合は、テキスト「All Channels」が戻されます。0(ゼロ)が戻された場合、データベースからchannel_descの値が戻されます。データベースから戻される値は、「Internet」のような実際の値または格納されたNULLです。country_idを表示する2番目の列指定も同様に処理されます。

20.4.2 GROUPINGを使用するとき

GROUPING関数は、2種類のNULLの識別に役立つのみでなく、小計行のソートまたは結果のフィルタも可能です。例20-7では、CUBEによって作成された小計のサブセットを取り出し、基本レベルの集計は取り出しません。HAVING句で、GROUPING関数を使用する列を制約します。

例20-7 HAVINGと組み合せたGROUPING

```
SELECT channel_desc, calendar_month_desc, country_iso_code, TO_CHAR(
SUM(amount_sold), '9,999,999,999') SALES$, GROUPING(channel_desc) CH, GROUPING
  (calendar_month_desc)  MO, GROUPING(country_iso_code) CO
```

```
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id
  AND customers.country_id = countries.country_id
  AND sales.channel_id= channels.channel_id
  AND channels.channel_desc IN ('Direct Sales', 'Internet')
  AND times.calendar_month_desc IN ('2000-09', '2000-10')
  AND country_iso_code IN ('GB', 'US')
GROUP BY CUBE(channel_desc, calendar_month_desc, country_iso_code)
HAVING (GROUPING(channel_desc)=1 AND GROUPING(calendar_month_desc)= 1
  AND GROUPING(country_iso_code)=1) OR (GROUPING(channel_desc)=1
  AND GROUPING (calendar_month_desc)= 1) OR (GROUPING(country_iso_code)=1
  AND GROUPING(calendar_month_desc)= 1);

CHANNEL_DESC         C CO SALES$               CH         MO         CO
-------------------- - -- -------------- ---------- ---------- ----------
                       US      1,581,775          1          1          0
                       GB        208,257          1          1          0
Direct Sales                 1,497,646          0          1          1
Internet                       292,387          0          1          1
                             1,790,032          1          1          1
```

例20-7でグループが正確に指定されていることを確認するために、例20-7の結果セットと例20-2の結果セットを比較します。前者には、timeおよびdepartmentについて集計された年間合計、地域合計および総計のみが含まれています。

20.4.3 GROUPING_ID関数

特定の行のGROUP BYレベルを調べるには、問合せでGROUP BY列ごとにGROUPING関数情報を戻す必要があります。そのためにGROUPING関数を使用する場合は、各GROUP BY列にGROUPING関数を使用するもう1つの列が必要です。たとえば、4列のGROUP BY句は4つのGROUPING関数を使用して分析する必要があります。これは、SQLで記述するには不便であり、問合せに必要な列の数が増加します。問合せの結果セットを表に格納する場合は、マテリアライズド・ビューの場合と同様に余分な列により記憶領域が使用されます。

このような問題に対処するために、GROUPING_ID関数を使用できます。GROUPING_IDは、正確なGROUP BYレベルを判断できるように、単一の数値を戻します。GROUPING_IDは、行ごとに、該当するGROUPING関数を使用した場合に生成される1と0のセットを取り、それを連結してビット・ベクトルを形成します。このビット・ベクトルは2進数として扱われ、GROUPING_ID関数はこの数値の10進値を戻します。たとえば、式CUBE(a, b)でグルーピングする場合、可能な値は表20-2のようになります。

表20-2 CUBE(a, b)のGROUPING_IDの例

集計レベル	ビット・ベクトル	GROUPING_ID
a, b	0 0	0
a	0 1	1
b	1 0	2
総計	1 1	3

GROUPING_IDでは、グルーピング・セット指定により作成されたグルーピングが明確に区別されるため、マテリアライズド・ビューのリフレッシュおよびリライトに非常に有効です。

20.4.4 GROUP_ID関数

GROUP BYの拡張は強力で柔軟性があり、重複するグルーピングを含む複雑な結果セットの出力も可能です。GROUP_ID関数を使用すると、重複するグルーピングを区別できます。特定レベルで計算される複数の行集合がある場合、GROUP_IDは最初の集合の行すべてに値0を割り当てます。特定のグルーピングに関する重複行の他の集合にはすべて、1から始まる上位の値が割り当てられます。たとえば、重複するグルーピングを生成する次の問合せを考えてみます。

例20-8 問合せにおけるGROUP_ID

```
SELECT country_iso_code, SUBSTR(cust_state_province,1,12), SUM(amount_sold),
  GROUPING_ID(country_iso_code, cust_state_province) GROUPING_ID, GROUP_ID()
FROM sales, customers, times, countries
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id
  AND customers.country_id=countries.country_id AND times.time_id= '30-OCT-00'
  AND country_iso_code IN ('FR', 'ES')
GROUP BY GROUPING SETS (country_iso_code,
ROLLUP(country_iso_code, cust_state_province));

CO SUBSTR(CUST_ SUM(AMOUNT_SOLD) GROUPING_ID GROUP_ID()
-- ------------ ---------------- ----------- ----------
ES Alicante              135.32           0          0
ES Valencia             4133.56           0          0
ES Barcelona              24.22           0          0
FR Centre                  74.3           0          0
FR Aquitaine             231.97           0          0
FR Rhtne-Alpes           1624.69           0          0
FR Ile-de-Franc          1860.59           0          0
FR Languedoc-Ro           4287.4           0          0
                         12372.05           3          0
ES                        4293.1           1          0
FR                        8078.95           1          0
ES                        4293.1           1          1
FR                        8078.95           1          1
```

この問合せでは、(country_id, cust_state_province)、(country_id)、(country_id)および()というグルーピングが生成されます。グルーピング(country_id)が2度繰り返されていることに注意してください。GROUPING SETSの構文については、GROUPING SETS式を参照してください。

この関数を使用すると、結果にフィルタを適用して重複するグルーピングを排除できます。たとえば、問合せにHAVING句の条件GROUP_ID()=0を追加し、前述の例から重複するグルーピング(region)を排除できます。

20.5 GROUPING SETS式

GROUP BY句の中でGROUPING SETS式を使用して、作成するグループの集合を選択的に指定できます。これにより、CUBE全体を計算せずに、複数のディメンションにまたがる正確な指定ができます。GROUPING SETSの構文の内容はGROUPING SETSの構文です。

次に例を示します。

```
SELECT channel_desc, calendar_month_desc, country_iso_code,
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id AND
  sales.channel_id= channels.channel_id AND channels.channel_desc IN
 ('Direct Sales', 'Internet') AND times.calendar_month_desc IN
 ('2000-09', '2000-10') AND country_iso_code IN ('GB', 'US')
GROUP BY GROUPING SETS((channel_desc, calendar_month_desc, country_iso_code),
```

```
    (channel_desc, country_iso_code), (calendar_month_desc, country_iso_code));
```

この文は複合列を使用していることに注意してください。詳細は、複合列とグルーピングについてを参照してください。この文では、次の3つのグルーピングにまたがる集計が計算されます。

- (channel_desc, calendar_month_desc, country_iso_code)
- (channel_desc, country_iso_code)
- (calendar_month_desc, country_iso_code)

前述の文を次の代替文と比較します。次の文は、CUBE操作とGROUPING_ID関数を使用して必要な行を戻します。

```
SELECT channel_desc, calendar_month_desc, country_iso_code,
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$,
       GROUPING_ID(channel_desc, calendar_month_desc, country_iso_code) gid
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id AND
  sales.channel_id= channels.channel_id AND channels.channel_desc IN
 ('Direct Sales', 'Internet') AND times.calendar_month_desc IN
 ('2000-09', '2000-10') AND country_iso_code IN ('GB', 'US')
GROUP BY CUBE(channel_desc, calendar_month_desc, country_iso_code)
HAVING GROUPING_ID(channel_desc, calendar_month_desc, country_iso_code)=0
  OR GROUPING_ID(channel_desc, calendar_month_desc, country_iso_code)=2
  OR GROUPING_ID(channel_desc, calendar_month_desc, country_iso_code)=4;
```

この文では、8(2×2×2)のグルーピングがすべて計算されますが、必要としているのは上の3グループのみです。

もう1つの方法は次の文ですが、複数の組合せがあるため長くなります。この文では、実表を3回スキャンする必要があり、非効率的です。CUBEとROLLUPは、きわめて限定的な意味を持つグルーピング・セットとみなすことができます。たとえば、次の文を考えてみます。

```
CUBE(a, b, c)
```

この文は、次の文と同等です。

```
GROUPING SETS ((a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), ())
ROLLUP(a, b, c)
```

さらに、この文は、次の文と同等です。

```
GROUPING SETS ((a, b, c), (a, b), ())
```

20.5.1 GROUPING SETSの構文

GROUPING SETSの構文では、同じ問合せで複数のグルーピングを定義できます。GROUP BYでは、指定したグルーピングがすべて計算され、UNION ALLと組み合されます。たとえば、次の文を考えてみます。

```
GROUP BY GROUPING sets (channel_desc, calendar_month_desc, country_id )
```

この文は、次の文と同等です。

```
GROUP BY channel_desc UNION ALL
GROUP BY calendar_month_desc UNION ALL GROUP BY country_id
```

表20-3に、グルーピング・セット指定および同等のGROUP BY指定を示します。一部の例では複合列が使用されているため注意してください。

表20-3 GROUPING SETS文および同等のGROUP BY文

GROUPING SETS文	同等のGROUP BY文
GROUP BY GROUPING SETS(a, b, c)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY c
GROUP BY GROUPING SETS(a, b, (b, c))	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY b, c
GROUP BY GROUPING SETS((a, b, c))	GROUP BY a, b, c
GROUP BY GROUPING SETS(a, (b), ())	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY ()
GROUP BY GROUPING SETS(a, ROLLUP(b, c))	GROUP BY a UNION ALL GROUP BY ROLLUP(b, c)

問合せブロックにまたがって検索して実行計画を生成するオプティマイザがなければ、UNIONに基づく問合せでは実表salesを複数回スキャンする必要があります。通常、ファクト表は大型のため、これはきわめて非効率的です。GROUPING SETS文を使用すると、必要なすべてのグルーピングを同じ問合せブロック内で使用できます。

20.6 複合列およびグルーピングについて

複合列は、グルーピングの計算中に1単位として処理される列のコレクションです。次の文のように、列をカッコで囲んで指定します。

```
ROLLUP (year, (quarter, month), day)
```

この文では、データは年から四半期にまたがってはロールアップされませんが、かわりに、UNION ALLの次のグルーピングと同等化されます。

- (year, quarter, month, day)
- (year, quarter, month),
- (year)
- ()

(quarter, month)は複合列を形成し、1単位として処理されます。通常、複合列は、ROLLUP、CUBE、GROUPING SETSおよび連結されたグルーピングに有効です。たとえば、CUBEまたはROLLUPでは、複合列は特定レベルにまたがる集計がスキップされることを意味します。つまり、次の文になります。

```
GROUP BY ROLLUP(a, (b, c))
```

これは、次の操作に相当します。

```
GROUP BY a, b, c UNION ALL
GROUP BY a UNION ALL
GROUP BY ()
```

(b, c)は1単位として処理され、(b, c)にまたがるロールアップは適用されません。たとえば、zの場合、(b, c)およびGROUP BY式はGROUP BY ROLLUP(a, z)に減少します。これを次の通常のロールアップと比較します。

```
GROUP BY ROLLUP(a, b, c)
```

これは、次のようになります。

```
GROUP BY a, b, c UNION ALL
```

```
GROUP BY a, b UNION ALL
GROUP BY a UNION ALL
GROUP BY ().
```

同様に、次の文は4つのGROUP BYと同等です。

```
GROUP BY CUBE((a, b), c)

GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY c UNION ALL
GROUP By ()
```

GROUPING SETSでは、複合列はGROUP BYの特定レベルを示すために使用されます。複合列の他の例については、表20-3を参照してください。

例20-9 複合列

CUBEとROLLUPで必要な集計レベルは、完全には制御できません。たとえば、次の文を考えてみます。

```
SELECT channel_desc, calendar_month_desc, country_iso_code,
 TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id
  AND customers.country_id = countries.country_id
  AND sales.channel_id= channels.channel_id
  AND channels.channel_desc IN ('Direct Sales', 'Internet')
  AND times.calendar_month_desc IN ('2000-09', '2000-10')
  AND country_iso_code IN ('GB', 'US')
GROUP BY ROLLUP(channel_desc, calendar_month_desc, country_iso_code);
```

この文では、Oracleにより次のグルーピングが計算されます。

- (channel_desc, calendar_month_desc, country_iso_code)
- (channel_desc, calendar_month_desc)
- (channel_desc)
- ()

これらのうち、1、3および4番目のグルーピングのみが必要な場合に、計算をこれらのグルーピングに制限するには、複合列を使用する必要があります。複合列を使用すると、月と国をロールアップ中に1単位として処理することで計算を制限できます。カッコ内の列は、CUBEおよびROLLUPの計算中に1単位として処理されます。つまり、次のようになります。

```
SELECT channel_desc, calendar_month_desc, country_iso_code,
    TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id AND
      sales.channel_id= channels.channel_id  AND channels.channel_desc IN
 ('Direct Sales', 'Internet') AND times.calendar_month_desc IN
 ('2000-09', '2000-10') AND country_iso_code IN ('GB', 'US')
GROUP BY ROLLUP(channel_desc, (calendar_month_desc, country_iso_code));

CHANNEL_DESC         CALENDAR CO SALES$
-------------------- -------- -- --------------
Internet             2000-09  GB         228,241
Internet             2000-09  US         228,241
Internet             2000-10  GB         239,236
Internet             2000-10  US         239,236
Internet                                 934,955
Direct Sales         2000-09  GB       1,217,808
```

```
Direct Sales          2000-09  US       1,217,808
Direct Sales          2000-10  GB       1,225,584
Direct Sales          2000-10  US       1,225,584
Direct Sales                            4,886,784
                                        5,821,739
```

20.7 連結グルーピングおよびデータ集計

連結グルーピングにより、一貫した方法でグルーピングの有効な組合せを生成できます。連結グルーピングで指定したグルーピングにより、各グルーピング・セットからのグルーピングのクロス積が得られます。クロス積操作により、ごく少数の連結グルーピングで多数の最終グループを生成できます。複数のグルーピング・セット、キューブおよびロールアップをカンマで区切って指定するのみで、連結グルーピングを指定できます。連結グルーピング・セットの例を次に示します。

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

このSQLは、次のグルーピングを定義します。

```
(a, c), (a, d), (b, c), (b, d)
```

グルーピング・セットの連結は、次の理由できわめて有効です。

- 問合せの開発が簡単

  すべてのグルーピングを手動で列挙する必要がありません。

- アプリケーションで使用

  分析用アプリケーションで生成されるSQLでは、グルーピング・セットの連結を伴うことがよくあります。この場合、それぞれのグルーピング・セットがディメンションに必要なグルーピングを定義します。

例20-10 連結グルーピング

GROUP BY句に複数のグルーピングを指定することもできます。たとえば、各製品の売上値をtimeディメンション(year、monthおよびday)のすべてのレベルとgeographyディメンション(region)のすべてのレベルにまたがってロールアップして集計する場合は、次の文を発行できます。

```
SELECT channel_desc, calendar_year, calendar_quarter_desc, country_iso_code,
  cust_state_province, TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id = times.time_id AND sales.cust_id = customers.cust_id
 AND sales.channel_id = channels.channel_id AND countries.country_id =
    customers.country_id AND channels.channel_desc IN
   ('Direct Sales', 'Internet') AND times.calendar_month_desc IN ('2000-09',
 '2000-10') AND countries.country_iso_code IN ('GB', 'FR')
GROUP BY channel_desc, GROUPING SETS (ROLLUP(calendar_year,
   calendar_quarter_desc),
ROLLUP(country_iso_code, cust_state_province));
```

その結果、次のようなグルーピングとなります。

- (channel_desc, calendar_year, calendar_quarter_desc)
- (channel_desc, calendar_year)
- (channel_desc)
- (channel_desc, country_iso_code, cust_state_province)
- (channel_desc, country_iso_code)

- (channel_desc)

これは、次のクロス積です。

- channel_desc
- ROLLUP(calendar_year, calendar_quarter_desc)。これは、((calendar_year, calendar_quarter_desc), (calendar_year), ())と同等です。
- ROLLUP(country_iso_code, cust_state_province)。これは、((country_iso_code, cust_state_province), (country_iso_code), ())と同等です。

出力には(channel_desc)グループが2つ含まれていることに注意してください。余分な(channel_desc)グループを排除するには、問合せでGROUP_ID関数を使用します。

連結結合のもう1つの例を例20-11に示します。この例は、2つのグルーピング・セットのクロス積を示しています。

例20-11 連結グルーピング(2つのグルーピング・セットのクロス積)

```
SELECT country_iso_code, cust_state_province, calendar_year,
calendar_quarter_desc, TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id AND
 countries.country_id=customers.country_id  AND
  sales.channel_id= channels.channel_id AND channels.channel_desc IN
 ('Direct Sales', 'Internet') AND times.calendar_month_desc IN
 ('2000-09', '2000-10') AND country_iso_code IN ('GB', 'FR')
GROUP BY GROUPING SETS (country_iso_code, cust_state_province),
         GROUPING SETS (calendar_year, calendar_quarter_desc);
```

この文では、次のようなグルーピングが計算されます。

- (country_iso_code, year)、(country_iso_code, calendar_quarter_desc)、(cust_state_province, year)および(cust_state_province, calendar_quarter_desc)

20.7.1 連結グルーピングと階層的データ・キューブ

連結グルーピングの最も重要な用途の1つは、階層的データ・キューブに必要な集計を生成することです。階層的キューブは、データが各ディメンションのロールアップ階層に沿って集計され、これらの集計がディメンション間で組み合されるデータセットです。ビジネス・インテリジェンス問合せに必要な典型的な集計の集合が含まれます。連結グルーピングを使用すると、*n*個のROLLUPを使用するのみで階層キューブに必要な集計をすべて生成し、不要な集計の生成を回避できます。*n*はディメンション数です。

shサンプル・スキーマ・データセットのディメンションが3つのみで、それぞれにマルチレベルの階層があるとします。

- time: year、quarter、month、day(weekは別の階層内)
- product: category、subcategory、prod_name
- geography: region、subregion、country、state、city

このデータは、階層レベルごとに1列を使用して表され、ディメンションの列は合計12列および売上高を保持する列となります。

ビジネス・インテリジェンスのニーズに合せ、ディメンションの様々な組合せについて特定の集計を計算して格納できます。例20-12では、「day」を除くすべてのレベルの集計を作成しますが、これでは作成する行数が多すぎます。特に、各ディメンション内でROLLUPを使用して有効な集計を生成する必要があります。各ディメンションでROLLUPベースの集計を生成した後に、それを他のディメンションと組み合せます。これにより、階層的キューブが生成されます。これは12のディメンション列すべてを使用するCUBEとまったく同じではないため注意してください。2から12乗(4,096)の集計グループが作成されますが、そのうちで必要としているのはごく少数です。連結グルーピング・セットを使用すると、必要な集計のみを簡単に生成できます。例20-12に、GROUP

BY句が必要とされる例を示します。

例20-12 連結グルーピングと階層的キューブ

```
SELECT calendar_year, calendar_quarter_desc, calendar_month_desc,
  country_region, country_subregion, countries.country_iso_code,
 cust_state_province, cust_city, prod_category_desc, prod_subcategory_desc,
 prod_name, TO_CHAR(SUM (amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries, products
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id AND
  sales.channel_id= channels.channel_id AND sales.prod_id=products.prod_id AND
  customers.country_id=countries.country_id AND channels.channel_desc IN
 ('Direct Sales', 'Internet') AND times.calendar_month_desc IN
 ('2000-09', '2000-10') AND prod_name IN ('Envoy Ambassador',
 'Mouse Pad') AND countries.country_iso_code IN ('GB', 'US')
GROUP BY ROLLUP(calendar_year, calendar_quarter_desc, calendar_month_desc),
  ROLLUP(country_region, country_subregion, countries.country_iso_code,
         cust_state_province, cust_city),
  ROLLUP(prod_category_desc, prod_subcategory_desc, prod_name);
```

GROUP BY指定のロールアップにより、ディメンションごとに4つずつ、次のグループが生成されます。

表20-4 階層的CUBEの例

時間別ROLLUP	製品別ROLLUP	地理別ROLLUP
year、quarter、month	category、subcategory、name	region、subregion、country、state、city region、subregion、country、state region、subregion、country
year、quarter	category、subcategory	region、subregion
year	category	region
all times	all products	all geographies

前述のSQLで指定されている連結グルーピングでは、表に示したROLLUP集計を使用してクロス積が実行されます。クロス積により、階層的データ・キューブに必要な96(4×4×6)の集計グループが作成されます。96個のグルーピング・セット式を必要とするような内容を、3つのROLLUP式を使用して置き換えることには、重要なメリットがあります。つまり、簡潔なSQLはエラーの可能性が大幅に減少すること、メンテナンスがはるかに容易であること、そして問合せを大幅に最適化できることです。より多数のディメンションとレベルを持つキューブによる連結グルーピングの使用方法を指定すると、さらに大きなメリットが得られます。

階層的キューブの詳細は、SQLでの階層的キューブの処理を参照してください。

20.8 データ・ウェアハウスでの集計の使用に関する考慮事項

この項の内容は、次のとおりです。

- ROLLUPおよびCUBEでの階層処理
- ROLLUPおよびCUBEでの列の容量
- GROUP BYの拡張機能とともに使用するHAVING句

- GROUP BYの拡張機能とともに使用するORDER BY句
- ROLLUPおよびCUBEとともに他の集計関数を使用する場合
- インメモリー集計の使用

20.8.1 ROLLUPおよびCUBEでの階層処理

ROLLUPおよびCUBEは、システムにあるどの階層メタデータからも独立して動作します。計算は、主にそれらを使用するSELECT文で指定された列を基にして実行されます。この方法では、階層メタデータを使用できるかどうかにかかわらず、CUBEおよびROLLUPが使用可能になります。階層ディメンションでレベルを処理するには、ROLLUPを使用し、別の列を使用して明示的にレベルを示すことが最も簡単な方法です。次に、簡単な例を示します。この例では、月は四半期にロールアップされ、四半期は年にロールアップされます。

例20-13 ROLLUPおよびCUBEでの階層処理

```
SELECT calendar_year, calendar_quarter_number,
    calendar_month_number, SUM(amount_sold)
FROM sales, times, products, customers, countries
WHERE sales.time_id=times.time_id
  AND sales.prod_id=products.prod_id
  AND customers.country_id = countries.country_id
  AND sales.cust_id=customers.cust_id
  AND prod_name IN ('Envoy Ambassador', 'Mouse Pad')
  AND country_iso_code = 'GB' AND calendar_year=1999
GROUP BY ROLLUP(calendar_year, calendar_quarter_number, calendar_month_number);

CALENDAR_YEAR CALENDAR_QUARTER_NUMBER CALENDAR_MONTH_NUMBER SUM(AMOUNT_SOLD)
------------- ----------------------- --------------------- ----------------
         1999                       1                     1          5521.34
         1999                       1                     2         22232.95
         1999                       1                     3         10672.63
         1999                       1                               38426.92
         1999                       2                     4         23658.05
         1999                       2                     5          5766.31
         1999                       2                     6         23939.32
         1999                       2                               53363.68
         1999                       3                     7         12132.18
         1999                       3                     8         13128.96
         1999                       3                     9         19571.96
         1999                       3                               44833.1
         1999                       4                    10         15752.18
         1999                       4                    11          7011.21
         1999                       4                    12          14257.5
         1999                       4                               37020.89
         1999                                                      173644.59
                                                                   173644.59
```

20.8.2 ROLLUPおよびCUBEでの列の容量

CUBE、ROLLUPおよびGROUPING SETSは、GROUP BY句の列容量を制限しません。GROUP BY句で処理できる列は、拡張機能使用の有無にかかわらず255列以内です。ただし、CUBEでは組合せの数が膨大になるため、CUBEによる拡張で多数の列を指定することは望ましくありません。CUBEに対する20列のリストで、結果セットに2から20の組合せが作成されたとします。膨大なCUBEリストは、システム・リソースを極限まで使用するため、そのような問合せでは、パフォーマンスおよびシステムにかかる負荷を慎重にテストする必要があります。

20.8.3 GROUP BYの拡張機能とともに使用するHAVING句

SELECT文のHAVING句は、GROUP BYの使用による影響を受けません。HAVING句で指定する条件は、結果セットの小計行および小計以外の行の両方に適用されます。問合せでHAVING句から小計行または小計以外の行を排除する必要がある場合もあります。これは、HAVING句とともにGROUPINGまたはGROUPING_ID関数を使用することによって可能になります。この例については、例20-7および関連するSQLを参照してください。

20.8.4 GROUP BYの拡張機能とともに使用するORDER BY句

多くの場合、問合せでは行を特定の方法で順序付けする必要があり、これはORDER BY句で行われます。ORDER BY句はGROUP BYの計算が完了した後に適用されるため、SELECT文のORDER BY句はGROUP BYの使用による影響を受けません。

ORDER BY指定では、結果セットの集計行と非集計行が区別されないため注意してください。たとえば、売上高を降順でリストし、各グループの最後に小計を置く必要があるとします。売上高を降順で順序付けするのみでは、小計(最大値)が各グループの最初に置かれるため不十分です。したがって、ORDER BY句の列には、集計列と非集計列を区別する列を含める必要があります。この要件は、ORDER BYをGROUP BYへの集計拡張とともに使用する問合せでは、通常、1つ以上のGROUPING関数を使用する必要があることを意味します。

20.8.5 ROLLUPおよびCUBEとともに他の集計関数を使用する場合

この章の例では、SUM関数で使用するROLLUPおよびCUBEを示しています。これは最も一般的な集計タイプですが、これらの拡張は、GROUP BY句で使用できるその他のすべての関数(COUNT、AVG、MIN、MAX、STDDEVおよびVARIANCE)で使用することもできます。COUNTは、クロス集計分析で必要になる場合が多く、2番目に使用頻度の高い関数と考えられます。

20.8.6 インメモリー集計の使用

通常、分析問合せは、データに対して複雑な集計を実行することによって、パターンや傾向を検出しようとします。インメモリー集計は、KEY VECTORおよびVECTOR GROUP BY操作を使用して集計に関係する問合せブロックを最適化し、単一の大きな表から複数の小さな表への結合を実行します(たとえば典型的なスター・クエリーの場合)。これらの操作は、結合と集計において、効果的なインメモリー配列を使用します。基礎となる表がインメモリー列ストア(IM列ストア)に格納されている場合、特に効果的です。

VECTOR GROUP BY変換は、効率的なインメモリー配列ベースの集計を可能にする最適化変換です。それにより、表スキャンの際、インメモリー配列に集計値が蓄積されます。その結果、結合と、結合および集計のパフォーマンスが向上します。

VECTOR GROUP BY変換は、スター型変換と同様の2つの部分を持つ処理で、次のステップを含んでいます。

1. ディメンション表がスキャンされ、任意のWHERE句の述語が適用されます。これらのスキャンの結果に基づいて、KEY VECTORと呼ばれる新しいデータ構造が作成されます。

   KEY VECTORはブルーム・フィルタと同様に、ファクト表のスキャン時に追加のフィルタ述語として結合の述語を適用できますが、ファクト表のスキャン後ではなく、スキャン時にOracle DatabaseがGROUP BYまたは集計を行うことも可能にします。

2. ファクト表スキャンの結果は、KEY VECTOR作成の過程で作成される一時表に再び結合されます。

これら2つのフェーズを組み合せると、複合的な集計における多数の表結合の効率が劇的に改善されます。どちらのフェーズも、問合せの実行計画に表示されます。

例20-14: VECTOR GROUP BY変換を使用する集計

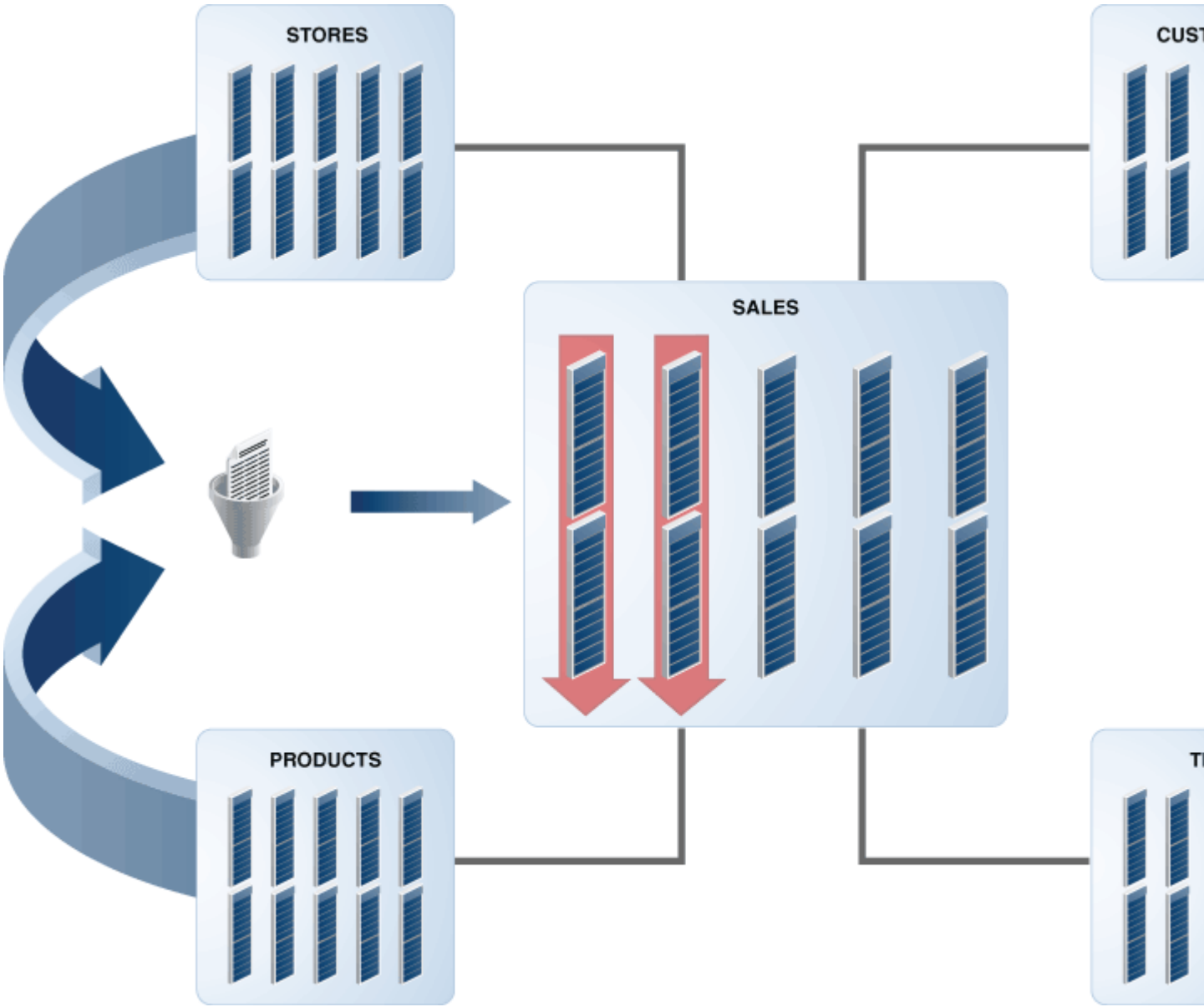
products、customersおよびtimesの各ディメンションをsalesファクト表に結合する次の問合せを検討します。

```
SELECT p.department_name, c.customer_id, t.fiscal_year, SUM(sales)
```

```
FROM PRODUCTS p, CUSTOMERS c, TIMES t, SALES s
WHERE p.product_id = s.product_id AND c.customer_id = s.customer_id
     AND t.time_id = s.time_id
GROUP BY p.department_name, c.customer_id, t.fiscal_year;
```

IM列ストアが設定されている場合、この問合せはオプティマイザにより、ベクトル結合とVECTOR GROUP BY集計を使用するように書きなおされます。図20-2は、VECTOR GROUP BYを使用する集計がどのように実行されるかを説明しています。ディメンション表PRODUCTS、CUSTOMERSおよびTIMESの述語は、ファクト表SALESでフィルタに変換されます。GROUP BYは、インメモリー配列を使用することにより、SALES表のスキャンと同時に実行されます。

図20-2 Oracle In-Memory Column Storeを使用するVECTOR GROUP BY



20.9 WITH句を使用した計算

WITH句(旧称はsubquery_factoring_clause)を使用すると、同じ問合せブロックが複雑な問合せに複数回発生する場合に、SELECT文中で再使用できます。WITHは、SQL-99標準の一部です。これは、問合せに同じ問合せブロックの参照が複数あり、結合と集計が存在する場合に特に便利です。WITH句を使用すると、Oracleでは問合せブロックの結果が取り出され、それがユーザーの一時表領域に格納されます。システムの構成に応じて、結果は共有一時表領域またはローカルの一時表領域に格納できます。Oracle Databaseでは、WITH句の再帰的使用はサポートされないため注意してください。ただし、部品表で使用する問合せや、親子階層から親子孫階層に拡張する問合せなどで使用されるWITH句の再帰的使用はサポートしていま

す。詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

ノート:

前のリリースで、一時表領域という用語で示されていた対象は現在、共有一時表領域と呼ばれています。

次の問合せは、WITH句を使用してパフォーマンスを改善し、SQLをより単純に記述できる一例です。この問合せでは、各チャネルの売上合計が計算され、channel_summaryという名前で保持されます。次に、各チャネルの総売上がチェックされ、総売上の3分の1を超えているチャネルの売上があるかどうかが調べられます。WITH句を使用すると、channel_summaryデータは1回のみ計算され、大きいsales表の余分なスキャンを回避できます。

例20-15 WITH句

```
WITH channel_summary AS (SELECT channels.channel_desc, SUM(amount_sold)
AS channel_total FROM sales, channels
WHERE sales.channel_id = channels.channel_id GROUP BY channels.channel_desc)
SELECT channel_desc, channel_total
FROM channel_summary WHERE channel_total > (SELECT SUM(channel_total) * 1/3
FROM channel_summary);

CHANNEL_DESC         CHANNEL_TOTAL
-------------------- -------------
Direct Sales            57875260.6
```

この例は、分析計算およびレポート用SQL関数で説明する集計レポート関数を使用しても、効率的に実行できます。

20.10 SQLでの階層的キューブの処理

この項では、階層的キューブの処理の例を示します。次の項目が含まれます。

- SQLでの階層的キューブの指定
- SQLでの階層的キューブの問合せ

20.10.1 SQLでの階層的キューブの指定

Oracle Databaseでは、簡単で効率的なSQL問合せで階層的キューブを指定できます。このような階層的キューブは、多くの分析用SQL製品で論理キューブと呼ばれているものと同じです。データを階層的キューブ形式で指定するには、GROUP BY句に対する拡張機能の1つである連結グルーピング・セットを使用して、階層的データ・キューブに必要な集計を生成できます。連結ロールアップ(各ディメンションの階層に沿ってロールアップしてから、複数のディメンションにまたがってそのデータを連結すること)を使用すると、階層的キューブに必要な集計をすべて生成できます。

例20-16 連結ROLLUP

2次元の例(例20-12と同様)の階層的キューブの作成に必要なGROUP BY句は、次のとおりです。次の簡単な構文で連結ロールアップが実行されます。

```
GROUP BY ROLLUP(year, quarter, month), ROLLUP(Division, brand, item)
```

この連結ロールアップでは、表20-4にリストされているROLLUP集計を使用してクロス積を実行します。クロス積により、階層的データ・キューブに必要な16(4×4)の集計グループが作成されます。

20.10.2 SQLでの階層的キューブの問合せ

分析アプリケーションではデータをキューブとして扱いますが、必要なのはキューブの特定のスライスおよび領域のみです。連結ロールアップ(階層的キューブ)により、リレーショナル・データをキューブとして扱えます。複雑な分析用問合せを処理する基本的な手法は、キューブの中の必要なスライスを正確に指定する外側の問合せ内に、階層的キューブ問合せを入れるというものです。Oracle Databaseでは、スライス問合せの中にネストされている階層キューブの処理を最適化します。強力なアルゴリズムを多数適用することにより、今までにないような速度と規模でこのような問合せを処理できます。これにより、SQLの分析ツールや分析アプリケーションで一貫した問合せスタイルを使用して、非常に複雑な問合せでも処理できます。

例20-17 階層的キューブの問合せ

次の分析問合せを考えてみます。この問合せは、スライス問合せの中にネストされている階層的キューブ問合せで構成されています。

```
SELECT month, division, sum_sales FROM
  (SELECT year, quarter, month, division, brand, item, SUM(sales) sum_sales,
      GROUPING_ID(grouping-columns) gid
   FROM sales, products, time
   WHERE join-condition
   GROUP BY ROLLUP(year, quarter, month),
            ROLLUP(division, brand, item))
WHERE division = 25 AND month = 200201 AND gid = gid-for-Division-Month;
```

内側に指定されている階層的キューブは、ディメンションが2つと各ディメンションにレベルが4つ含まれる単純なキューブを定義しています。これにより、16のグループ(4つの時間レベル×4つの製品レベル)が生成されます。問合せ内のGROUPING_ID関数は、引数内の*grouping-columns*の集計レベルに基づいて、各行が属するグループを識別します。

外側の問合せは、この問合せに必要な制約を適用し、Divisionを値25に、Monthを値200201(この場合は2002年1月を表す)に限定します。概念的には、この問合せはキューブからデータの小さいかたまりをスライスし(切り取り)ます。GID列に対する外側の問合せの制約(問合せで*gid-for-division-month*により示されている)は、データがdivisionとmonthとの組合せとしてグループ化されていることを示すキーの値です。GID制約により、monthとdivisionというGROUP BY句のレベルで集計された行のみが選択されます。

Oracle Databaseでは、外側の問合せの条件に基づき、問合せ処理から不要な集計グループが排除されます。前述の外側の問合せの条件により、結果セットはdivisionおよびmonthを集計する1つのグループに限定されます。year、month、brandおよびitemを含むその他のグループは、ここではすべて不要です。グループ・プルーニング最適化ではこれを認識し、この問合せを次のように変換します。

```
SELECT month, division, sum_sales
FROM (SELECT  null, null,  month, division, null, null, SUM(sales) sum_sales,
      GROUPING_ID(grouping-columns) gid
      FROM sales, products, time WHERE join-condition
   GROUP BY month, division)
WHERE division = 25 AND month = 200201 AND gid = gid-for-Division-Month;
```

太字の部分が変更されたSQLを示します。これで、内側の問合せには、monthとdivisionを含む単純なGROUP BY句が含まれます。year列、quarter列、brand列およびitem列は、単純化されたGROUP BY句に合うようにNULLに変換されています。これで、問合せではグループを1つのみ要求するので、16個のグループのうち15個が処理から除外され、処理量が大幅に削減されます。より多くのディメンションとレベルを持つキューブでは、グループ・プルーニングによる節約はさらに大きくなる可能性があります。グループ・プルーニング変換処理は、GROUP BY句のすべての拡張機能(ROLLUP、CUBEおよびGROUPING SETS)に適用されます。

オプティマイザにより前述の問合せは単純なGROUP BYに変換されましたが、グループが事前計算されてマテリアライズド・ビューに

格納されていると、応答時間をさらに高速化できます。オンライン分析問合せではキューブの任意のスライスを求めることがあるため、多数のグループを事前に計算してマテリアライズド・ビューに格納しておくことが必要になります。これは次の項で説明します。

この項では、次の項目について説明します。

- 階層的キューブを格納するマテリアライズド・ビューを作成するSQL
- 階層的キューブのマテリアライズド・ビューの例

20.10.2.1 階層的キューブを格納するマテリアライズド・ビューを作成するSQL

分析用SQLでは、複数ユーザーに対する応答時間が速いことが要求されます。そのため、キューブの大部分を事前に計算してマテリアライズド・ビューに保持する必要があります。

データ・ウェアハウス設計者は、マテリアライズド・ビューのデータ量を正確に選択できます。データ・ウェアハウスでは、階層的キューブ全体を完全にマテリアライズド・ビューにして保持できます。その場合は記憶領域の量が最も多くなりますが、キューブ内のすべての問合せにすばやく応答できます。または、部分的にマテリアライズド・ビューにしたものをデータ・ウェアハウスに格納することもできます。この場合、記憶領域は節約されますが、高速に応答できるのは問合せ全体の一部のみに限定されます。問合せが、データセットで考えられるすべてのレベルの集計グループを対象としている場合は、階層キューブ全体をマテリアライズするのが最適の方法となることもあります。

これは、各ディメンションの集計階層が他の各ディメンションと組み合せて事前に計算されることを意味します。したがって、階層キューブ全体を事前に計算するには、小さい集計グループの集合より多くのディスク領域が必要であり、作成やリフレッシュの回数も増えます。処理時間およびディスク領域と問合せパフォーマンスとのトレードオフについて、作成を決定する前に考慮する必要があります。また、ディスク領域要件を少なくするためにデータ圧縮の使用も検討します。

関連項目:

- 表の圧縮の構文および制限の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。
- 表の圧縮の詳細は、『*Oracle Database*管理者ガイド』を参照してください。
- 表の圧縮の詳細は、マテリアライズド・ビューの記憶域および表の圧縮についてを参照してください。

20.10.2.2 階層的キューブのマテリアライズド・ビューの例

この項では、完全な階層的キューブおよび部分的な階層的キューブのマテリアライズド・ビューを示します。例の多くは、機能を示すもので、実際に実行されるものではありません。

ローリング・ウィンドウを使用することが多いデータ・ウェアハウスでは、複数のマテリアライズド・ビューに対し、必要な時間のレベルごとに1つずつの階層的キューブを格納してください。これにより、完全な階層的キューブは、sales_hierarchical_mon_cube_mv、sales_hierarchical_qtr_cube_mv、sales_hierarchical_yr_cube_mvおよびsales_hierarchical_all_cube_mvという4つのマテリアライズド・ビューに格納されることになります。

次の文では、3つのコンポジット・パーティション化されたマテリアライズド・ビューと1つのリスト・パーティション化されたマテリアライズド・ビューのセットに、完全な階層的キューブを作成します。

例20-18 完全な階層的キューブのマテリアライズド・ビュー

```
CREATE MATERIALIZED VIEW sales_hierarchical_mon_cube_mv
PARTITION BY RANGE (mon)
SUBPARTITION BY LIST (gid)
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE AS
```

```
SELECT calendar_year yr, calendar_quarter_desc qtr, calendar_month_desc mon,
    country_id, cust_state_province, cust_city,
    prod_category, prod_subcategory, prod_name,
    GROUPING_ID(calendar_year, calendar_quarter_desc, calendar_month_desc,
                country_id, cust_state_province, cust_city,
                prod_category, prod_subcategory, prod_name) gid,
    SUM(amount_sold) s_sales, COUNT(amount_sold) c_sales,
    COUNT(*) c_star
FROM sales s, products p, customers c, times t
WHERE s.cust_id = c.cust_id AND s.prod_id = p.prod_id AND s.time_id = t.time_id
GROUP BY calendar_year, calendar_quarter_desc, calendar_month_desc,
  ROLLUP(country_id, cust_state_province, cust_city),
  ROLLUP(prod_category, prod_subcategory, prod_name),
...;

CREATE MATERIALIZED VIEW sales_hierarchical_qtr_cube_mv
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE AS
SELECT calendar_year yr, calendar_quarter_desc qtr,
    country_id, cust_state_province, cust_city,
    prod_category, prod_subcategory, prod_name,
    GROUPING_ID(calendar_year, calendar_quarter_desc,
                country_id, cust_state_province, cust_city,
                prod_category, prod_subcategory, prod_name) gid,
    SUM(amount_sold) s_sales, COUNT(amount_sold) c_sales,
    COUNT(*) c_star
FROM sales s, products p, customers c, times t
WHERE s.cust_id = c.cust_id AND s.prod_id = p.prod_id
      AND s.time_id = t.time_id
GROUP BY calendar_year, calendar_quarter_desc,
  ROLLUP(country_id, cust_state_province, cust_city),
  ROLLUP(prod_category, prod_subcategory, prod_name),
PARTITION BY RANGE (qtr)
 SUBPARTITION BY LIST (gid)
...;

CREATE MATERIALIZED VIEW sales_hierarchical_yr_cube_mv
PARTITION BY RANGE (year)
SUBPARTITION BY LIST (gid)
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE AS
SELECT calendar_year yr, country_id, cust_state_province, cust_city,
    prod_category, prod_subcategory, prod_name,
    GROUPING_ID(calendar_year, country_id, cust_state_province, cust_city,
                prod_category, prod_subcategory, prod_name) gid,
    SUM(amount_sold) s_sales, COUNT(amount_sold) c_sales, COUNT(*) c_star
FROM sales s, products p, customers c, times t
WHERE s.cust_id = c.cust_id AND s.prod_id = p.prod_id AND s.time_id = t.time_id
GROUP BY calendar_year,
  ROLLUP(country_id, cust_state_province, cust_city),
  ROLLUP(prod_category, prod_subcategory, prod_name),
...;

CREATE MATERIALIZED VIEW sales_hierarchical_all_cube_mv
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE AS
SELECT country_id, cust_state_province, cust_city,
    prod_category, prod_subcategory, prod_name,
    GROUPING_ID(country_id, cust_state_province, cust_city,
                prod_category, prod_subcategory, prod_name) gid,
```

```
    SUM(amount_sold) s_sales, COUNT(amount_sold) c_sales, COUNT(*) c_star
FROM sales s, products p, customers c, times t
WHERE s.cust_id = c.cust_id AND s.prod_id = p.prod_id AND s.time_id = t.time_id
GROUP BY ROLLUP(country_id, cust_state_province, cust_city),
         ROLLUP(prod_category, prod_subcategory, prod_name),
PARTITION BY LIST (gid)
...;
```

これにより、sales表に対するパーティション・メンテナンス操作の際に、sales_hierarchical_mon_cube_mv、sales_hierarchical_qtr_cube_mvおよびsales_hierarchical_yr_cube_mvの各マテリアライズド・ビューでPCTリフレッシュを使用できるようになります。実表に大幅な変更があり、ログ・ベースの高速リフレッシュがPCTリフレッシュよりも遅くなると予想される場合にも、PCTリフレッシュを使用できます。FORCEメソッド(method => '?')を指定して、DBMS_MVIEWパッケージのリフレッシュ・サブプログラムを実行すると、Oracle Databaseはリフレッシュに最適な方法を選択します。PCTリフレッシュの詳細は、マテリアライズド・ビューのパーティション・チェンジ・トラッキング(PCT)リフレッシュについてを参照してください。

sales_hierarchical_qtr_cube_mvにはtimes表の列は含まれないため、このマテリアライズド・ビューではPCTリフレッシュは有効になりません。ただし、FORCEメソッド(method => '?')を指定すれば、DBMS_MVIEWパッケージのリフレッシュ・サブプログラムをコールできます。Oracle Databaseは、リフレッシュに最適な方法を選択します。

部分的なキューブ(つまり、完全なキューブからのグルーピングのサブセット)が必要な場合は、キューブを「連合キューブ」として格納することをお薦めします。連合キューブは、必要な各グルーピングを別個のマテリアライズド・ビューに格納します。

```
CREATE MATERIALIZED VIEW sales_mon_city_prod_mv
PARTITION BY RANGE (mon)
...
BUILD DEFERRED
REFRESH FAST ON DEMAND
  USING TRUSTED CONSTRAINTS
ENABLE QUERY REWRITE AS
SELECT calendar_month_desc mon, cust_city, prod_name, SUM(amount_sold) s_sales,
       COUNT(amount_sold) c_sales, COUNT(*) c_star
FROM sales s, products p, customers c, times t
WHERE s.cust_id = c.cust_id AND s.prod_id = p.prod_id
AND s.time_id = t.time_id
GROUP BY calendar_month_desc, cust_city, prod_name;

CREATE MATERIALIZED VIEW sales_qtr_city_prod_mv
PARTITION BY RANGE (qtr)
...
BUILD DEFERRED
REFRESH FAST ON DEMAND
  USING TRUSTED CONSTRAINTS
ENABLE QUERY REWRITE AS
SELECT calendar_quarter_desc qtr, cust_city, prod_name,SUM(amount_sold) s_sales,
COUNT(amount_sold) c_sales, COUNT(*) c_star
FROM sales s, products p, customers c, times t
WHERE s.cust_id = c.cust_id AND s.prod_id =p.prod_id AND s.time_id = t.time_id
GROUP BY calendar_quarter_desc, cust_city, prod_name;

CREATE MATERIALIZED VIEW sales_yr_city_prod_mv
PARTITION BY RANGE (yr)
...
BUILD DEFERRED
REFRESH FAST ON DEMAND
USING TRUSTED CONSTRAINTS
ENABLE QUERY REWRITE AS
SELECT calendar_year yr, cust_city, prod_name, SUM(amount_sold) s_sales,
       COUNT(amount_sold) c_sales, COUNT(*) c_star
```

```
FROM sales s, products p, customers c, times t
WHERE s.cust_id = c.cust_id AND s.prod_id =p.prod_id AND s.time_id = t.time_id
GROUP BY calendar_year, cust_city, prod_name;

CREATE MATERIALIZED VIEW sales_mon_city_scat_mv
PARTITION BY RANGE (mon)
...
BUILD DEFERRED
REFRESH FAST ON DEMAND
  USING TRUSTED CONSTRAINTS
ENABLE QUERY REWRITE AS
SELECT calendar_month_desc mon, cust_city, prod_subcategory,
       SUM(amount_sold) s_sales, COUNT(amount_sold) c_sales, COUNT(*) c_star
FROM sales s, products p, customers c, times t
WHERE s.cust_id = c.cust_id AND s.prod_id =p.prod_id AND s.time_id =t.time_id
GROUP BY calendar_month_desc, cust_city, prod_subcategory;

CREATE MATERIALIZED VIEW sales_qtr_city_cat_mv
PARTITION BY RANGE (qtr)
...
BUILD DEFERRED
REFRESH FAST ON DEMAND
  USING TRUSTED CONSTRAINTS
ENABLE QUERY REWRITE AS
SELECT calendar_quarter_desc qtr, cust_city, prod_category cat,
       SUM(amount_sold) s_sales, COUNT(amount_sold) c_sales, COUNT(*) c_star
FROM sales s, products p, customers c, times t
WHERE s.cust_id = c.cust_id AND s.prod_id =p.prod_id AND s.time_id =t.time_id
GROUP BY calendar_quarter_desc, cust_city, prod_category;

CREATE MATERIALIZED VIEW sales_yr_city_all_mv
PARTITION BY RANGE (yr)
...
BUILD DEFERRED
REFRESH FAST ON DEMAND
  USING TRUSTED CONSTRAINTS
ENABLE QUERY REWRITE AS
SELECT calendar_year yr, cust_city, SUM(amount_sold) s_sales,
       COUNT(amount_sold) c_sales, COUNT(*) c_star
FROM sales s, products p, customers c, times t
WHERE s.cust_id = c.cust_id AND s.prod_id = p.prod_id AND s.time_id = t.time_id
GROUP BY calendar_year, cust_city;
```

これらのマテリアライズド・ビューは、BUILD DEFERREDとして作成でき、次に DBMS_MVIEW.REFRESH_DEPENDENT(number_of_failures, 'SALES', 'C' ...)を実行すると、ディテール表salesで定義されている各マテリアライズド・ビューの完全リフレッシュが、最も効率的な順序でスケジューリングされます。詳細は、マテリアライズド・ビューのリフレッシュのスケジューリングを参照してください。

各マテリアライズド・ビューは、SELECT構文のリストで時間レベル(月、四半期、年)にパーティション化されるので、PCTは各マテリアライズド・ビューのsales表で使用できます。これによって、FASTおよびCOMPLETEリフレッシュ方法に加え、PCTリフレッシュ方法も適用できます。

21 パターン一致用SQL

一連の行でのパターン認識は要望の多い機能でしたが、現在までSQLでは使用できませんでした。多数の回避策がありましたが、これらの回避策は記述が難しく、理解しにくく、実行も非効率的でした。Oracle Database 12cからは、MATCH_RECOGNIZE句を使用して、効率的に実行するネイティブSQLでこの機能を実現できるようになります。この章では、この実現方法について説明しており、次の項で構成されています。

- データ・ウェアハウスにおけるパターン一致の概要
- パターン一致の基本トピック
- パターン一致の詳細
- パターン一致の高度なトピック
- パターン一致のルールと制限
- パターン一致の例

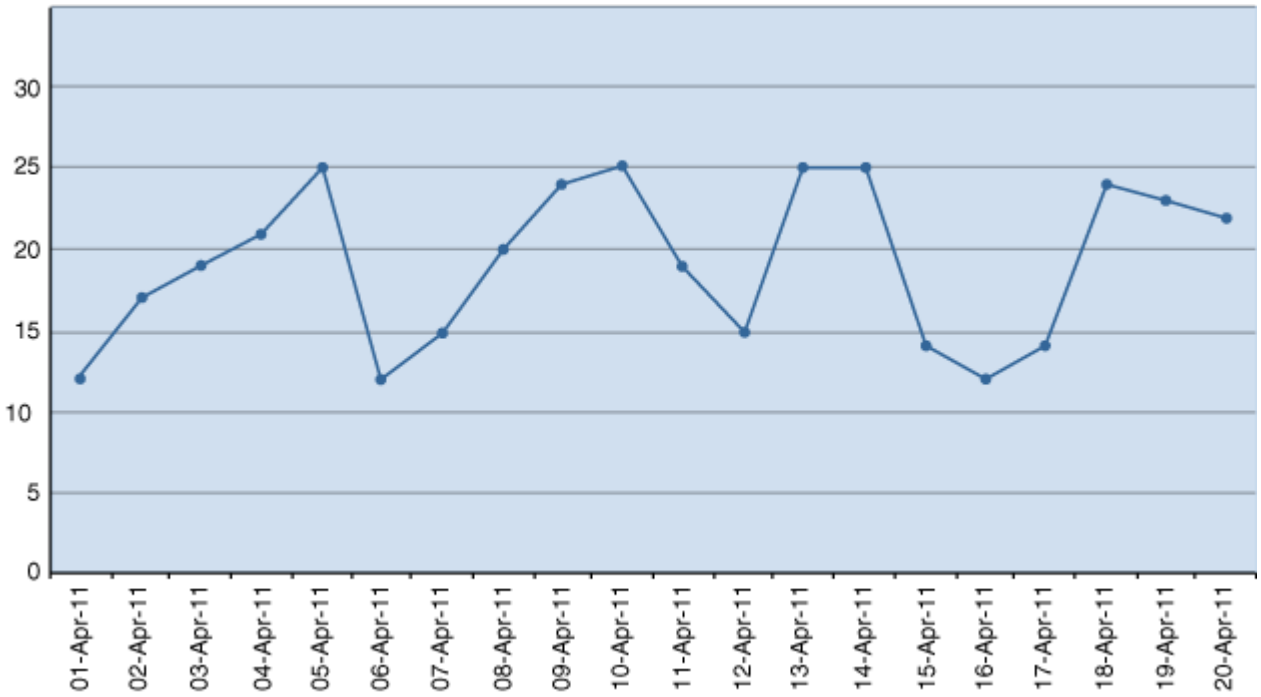
21.1 データ・ウェアハウスでのパターン一致の概要

SQLでのパターン一致は、MATCH_RECOGNIZE句を使用して実行されます。MATCH_RECOGNIZEにより、次のタスクが実行可能になります。

- PARTITION BY句およびORDER BY句を含むMATCH_RECOGNIZE句で使用されるデータを、論理的にパーティション化し、順序付けます。
- MATCH_RECOGNIZE句のPATTERN句を使用して、シークする行のパターンを定義します。これらのパターンでは、正規表現構文が使用されます。これは、強力かつ表現力の豊かな機能であり、ユーザーが定義するパターン変数に適用されます。
- 行をDEFINE句にある行パターン変数にマップするために必要な論理条件を指定します。
- MEASURES句で、SQL問合せの他のパートで使用可能な式であるメジャーを定義します。

単純なパターン一致として、図21-1に示す株価チャートを考えてみましょう。

図21-1 株価チャート



パターン一致によって様々なタイプの計算を実行するのみでなく、図21-1に示すV字形およびW字形のような価格パターンを識別できます。たとえば、ユーザーが実行する計算には、監視回数や下方傾向または上方傾向の平均値が含まれる場合もあります。

この項では、次の項目について説明します。

- パターン一致を使用する理由
- パターン一致におけるデータの処理方法
- パターン一致の特別な機能について

21.1.1 パターン一致を使用する理由

複数の行にわたって発生するパターンを認識する能力は、多様な作業で重要です。たとえば、一連のイベント主導によるあらゆる種類のビジネス・プロセス(異常な動作の検出が要求されるセキュリティ・アプリケーション、または価格設定、取引量およびその他の動作のパターンをシークする会計アプリケーションなど)があります。その他の一般的な用途として、詐欺行為検出アプリケーションやセンサー・データ分析があります。この総合的な領域を1つの用語で説明するのは複雑なイベント処理であるため、パターン一致の利用が大いに役立ちます。

ここで、例21-1の問合せを考えてみましょう。この問合せでは、図21-1に示す株価を使用します。この図は、後に続くCREATE文およびINSERTを使用してデータベースに取り込むことができます。この問合せでは、株価が底値をうって上昇したすべてのケースが検出されます。これは一般的にV字形と呼ばれます。この問合せを調べる前に、出力を調べてみましょう。3行のみ出力されていますが、これは、一致ごとに1行だけを報告するようにコードが作成されていて、3つの一致が見つかったためです。MATCH_RECOGNIZE句では、一致ごとに1行を出力するか、一致ごとにすべての行を出力するかを選択できます。この例では、一致ごとに1行という短い出力が採用されています。

例21-1 パターン一致: 一致ごとに1行を出力する単純なV字形

```
CREATE TABLE Ticker (SYMBOL VARCHAR2(10), tstamp DATE, price NUMBER);

INSERT INTO Ticker VALUES('ACME', '01-Apr-11', 12);
INSERT INTO Ticker VALUES('ACME', '02-Apr-11', 17);
INSERT INTO Ticker VALUES('ACME', '03-Apr-11', 19);
INSERT INTO Ticker VALUES('ACME', '04-Apr-11', 21);
INSERT INTO Ticker VALUES('ACME', '05-Apr-11', 25);
INSERT INTO Ticker VALUES('ACME', '06-Apr-11', 12);
```

```
INSERT INTO Ticker VALUES('ACME', '07-Apr-11', 15);
INSERT INTO Ticker VALUES('ACME', '08-Apr-11', 20);
INSERT INTO Ticker VALUES('ACME', '09-Apr-11', 24);
INSERT INTO Ticker VALUES('ACME', '10-Apr-11', 25);
INSERT INTO Ticker VALUES('ACME', '11-Apr-11', 19);
INSERT INTO Ticker VALUES('ACME', '12-Apr-11', 15);
INSERT INTO Ticker VALUES('ACME', '13-Apr-11', 25);
INSERT INTO Ticker VALUES('ACME', '14-Apr-11', 25);
INSERT INTO Ticker VALUES('ACME', '15-Apr-11', 14);
INSERT INTO Ticker VALUES('ACME', '16-Apr-11', 12);
INSERT INTO Ticker VALUES('ACME', '17-Apr-11', 14);
INSERT INTO Ticker VALUES('ACME', '18-Apr-11', 24);
INSERT INTO Ticker VALUES('ACME', '19-Apr-11', 23);
INSERT INTO Ticker VALUES('ACME', '20-Apr-11', 22);

SELECT *
FROM Ticker MATCH_RECOGNIZE (
     PARTITION BY symbol
     ORDER BY tstamp
     MEASURES  STRT.tstamp AS start_tstamp,
               LAST(DOWN.tstamp) AS bottom_tstamp,
               LAST(UP.tstamp) AS end_tstamp
     ONE ROW PER MATCH
     AFTER MATCH SKIP TO LAST UP
     PATTERN (STRT DOWN+ UP+)
     DEFINE
        DOWN AS DOWN.price < PREV(DOWN.price),
        UP AS UP.price > PREV(UP.price)
     ) MR
ORDER BY MR.symbol, MR.start_tstamp;

SYMBOL     START_TST BOTTOM_TS END_TSTAM
---------- --------- --------- ---------
ACME       05-APR-11 06-APR-11 10-APR-11
ACME       10-APR-11 12-APR-11 13-APR-11
ACME       14-APR-11 16-APR-11 18-APR-11
```

この問合せでは、何を行っているのでしょうか。次に、MATCH_RECOGNIZE句の各行について説明します。

- PARTITION BYはTicker表のデータを論理グループに分割します。論理グループの各グループには銘柄記号が1つずつ含まれています。
- ORDER BYは、各論理グループのデータをtstampによって順序付けます。
- MEASURESは、V字形の始まりのタイムスタンプ(start_tstamp)、V字形の底部のタイムスタンプ(bottom_tstamp)およびV字形の終わりのタイムスタンプ(end_tstamp)という3つのメジャーを定義します。メジャーbottom_tstampおよびend_tstampは、LAST()関数を使用して、取得された値が各パターン一致内のタイムスタンプの最終値であることを確認します。
- ONE ROW PER MATCHは、検出されたパターン一致ごとに1行が出力されることを意味します。
- AFTER MATCH SKIP TO LAST UPは、一致が見つかるたびに、UPパターン変数の最終行で検索が再開されることを意味します。パターン変数とはMATCH_RECOGNIZE文で使用される変数であり、DEFINE句の中で定義されます。
- PATTERN (STRT DOWN+ UP+)は、検索中のパターンに3つのパターン変数、STRT、DOWNおよびUPがあることを示します。DOWNとUPの後のプラス記号(+)は、少なくとも1行をそれぞれにマップする必要があることを示しています。パターンでは、パターンを検索するうえで非常に表現力の豊かな正規表現が定義されます。

- DEFINEによって、行パターン変数STRT、DOWNおよびUPに行をマップするため満たす必要のある条件が指定されます。STRTには条件がないため、任意の行をSTRTにマップできます。なぜ条件のないパターン変数があるのでしょうか。これは、一致をテストするための開始点として使用されます。DOWNとUPは両方ともPREV()関数を利用して、現在行の価格と前の行の価格とを比較します。DOWNは、行の価格が前の行の価格よりも低い場合に一致します。したがって、V字形の下方(左)部分を定義します。行の価格が前の行の価格よりも高い場合は、その行をUPにマップできます。

次の2つの図によって、例21-1で返された結果が理解しやすくなります。図21-2は、PATTERN句で指定された特定のパターン変数にマップされた日付を示しています。パターン変数と日付とのマッピングが使用可能になった後、その情報はMEASURES句で使用されてメジャー値が計算されます。メジャーの結果を、図21-3に示します。

図21-2 日付とパターン変数とのマッピングを示す株価チャート

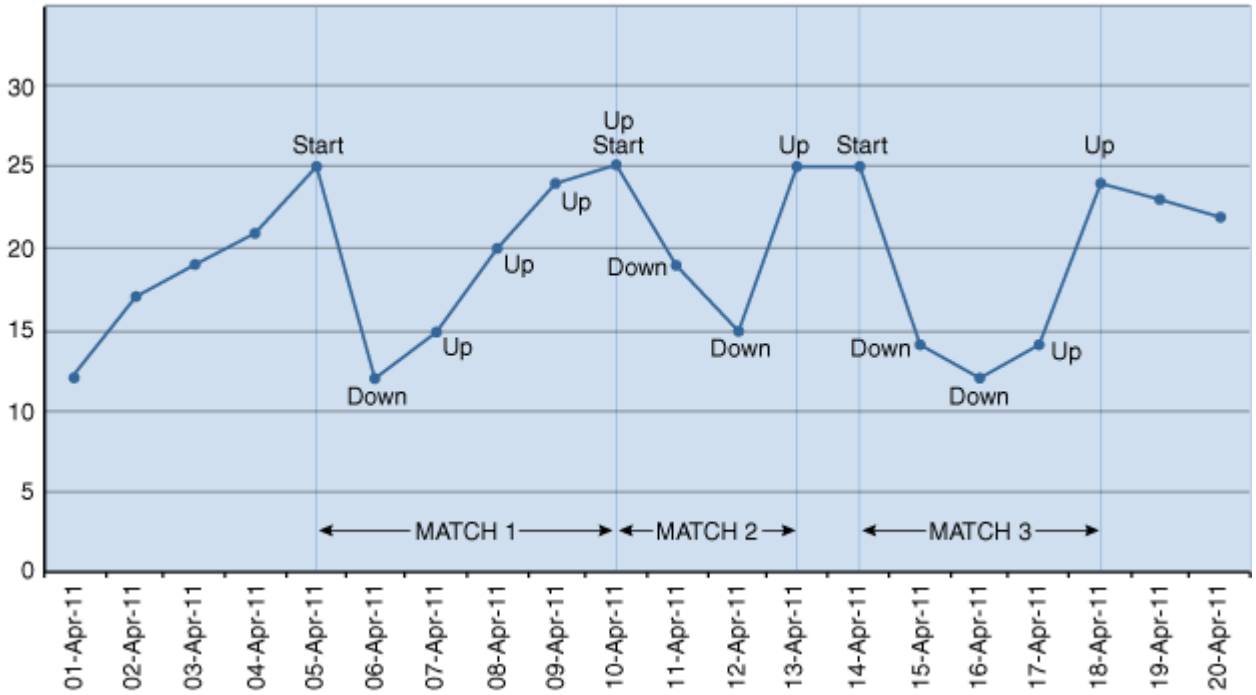


図21-2では、パターン変数にマップされた日付ごとにラベルを付けています。このマッピングは、PATTERN句で指定されたパターンとDEFINE句で指定された論理条件に基づいています。細い縦線は、このパターンに関して見つかった3つの一致の境界を示しています。それぞれの一致では、最初の日付にそれにマップされたSTRTパターン変数(*Start*とラベル表示)が表示され、続いてDOWNパターン変数にマップされた1つ以上の日付、最後にUPパターン変数にマップされた1つ以上の日付が表示されます。

問合せの中でAFTER MATCH SKIP TO LAST UPを指定したため、2つの隣接する一致で1つの行が共有される場合があります。つまり、1つの日付に2つの変数がマップされることがあります。たとえば、10-Aprilにパターン変数UPとSTRTの両方がマップされるとします。その場合、April 10は一致1の終わりであり、一致2の始まりでもあります。

図21-3 メジャーが対応している日付を示す株価チャート

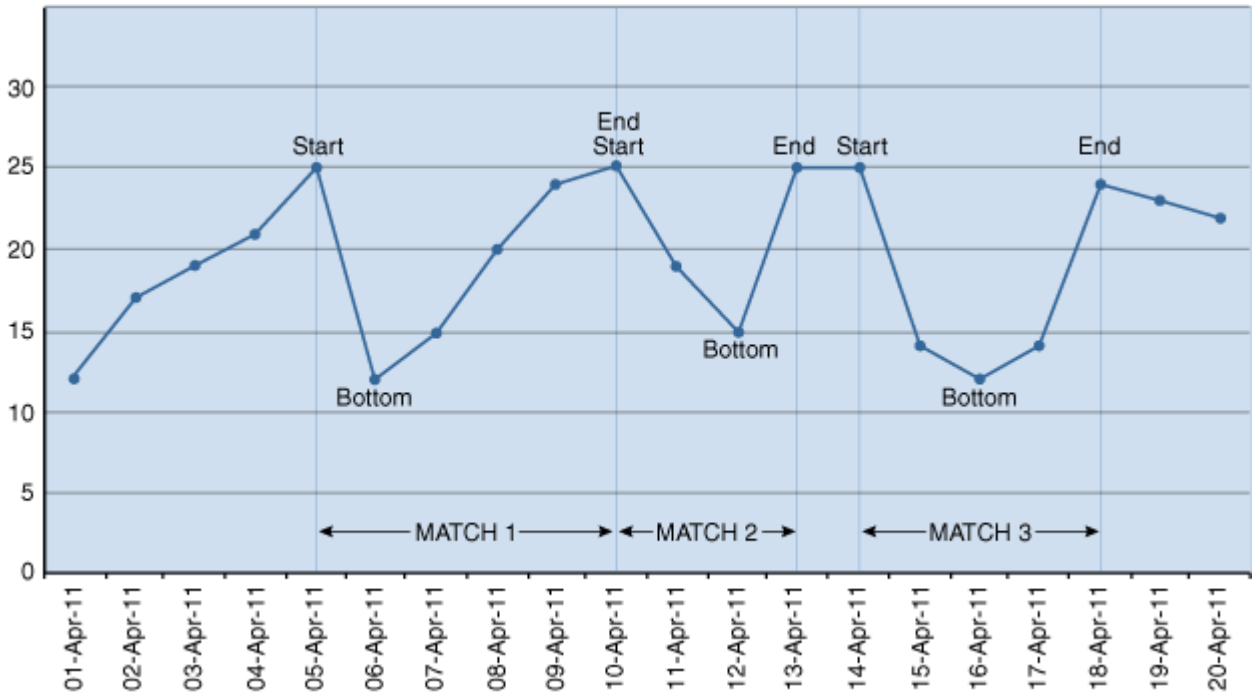


図21-3のラベル表示は、問合せのMEASURES句で定義されたメジャーのみ、すなわちSTART (問合せのstart_tstamp)、BOTTOM (問合せのbottom_tstamp)およびEND (問合せのend_tstamp)のみを対象としています。図21-2と同様に、細い縦線は、このパターンに関して見つかった3つの一致の境界を示しています。すべての一致には、Start日付、Bottom日付およびEnd日付が定義されています。図21-2と同様に、日付10-Aprilが2つの一致に検出されました。つまり、一致1のENDメジャー、一致2のSTARTメジャーです。図21-3のラベル表示された日付は、図21-2のパターン変数マッピングに基づいたメジャー定義にどの日付が対応しているかを示しています。

図21-3でラベル表示された日付は、例の出力で前出の9個の日付に対応しています。出力の1行目には一致1の日付が、出力の2行目には一致2の日付が、出力の3行目には一致3の日付がそれぞれ表示されます。

21.1.2 パターン一致におけるデータの処理方法

MATCH_RECOGNIZE句で、次のステップが実行されます。

1. 行パターン入力表がPARTITION BY句に従ってパーティション化されます。各パーティションは、パーティション化列の値と同じ値が設定された入力表の行のセットで構成されています。
2. 行パターンの各パーティションは、ORDER BY句に従って順序付けられます。
3. 順序付けられた行パターンの各パーティションは、PATTERNとの一致がないか検索されます。
4. パターン一致は、ORDER BY句で指定した順序で行パターン・パーティションの行を考慮し、最初の行に出現する一致をシークすることによって実行されます。

   一連の行でのパターン一致は増分プロセスであり、パターンに一致するかどうかを1行ずつ順次調べていきます。この増分処理モデルでは、完全なパターンが認識されるまでは、どのステップにおいても部分的な一致を認識できたにすぎず、将来どんな行が追加されるか、その行にどんな変数がマップされるかは不明です。

   最初の行で一致が見つからない場合は、パーティション内の次の行に検索が進み、その行から開始して一致が見つかるかどうか確認されます。
5. 一致が見つかると、行パターン一致によって行パターンのメジャー列が計算されます。これは、MEASURES句で定義された式に相当します。
6. 最初の例で示したように、パターン一致でONE ROW PER MATCHを使用すると、見つかった一致ごとに1行が生成されます。ALL ROWS PER MATCHを使用した場合、一致した行はすべてパターン一致の出力に含められます。

7. AFTER MATCH SKIP句は、空以外の一致が見つかった後で、行パターン・パーティション内で行パターン一致が再開される場所を判別します。前出の例では、行パターン一致は、一致が最後に見つかった行で再開されています(AFTER MATCH SKIP TO LAST UP)。

21.1.3 パターン一致の特別な機能について

次の機能があります。

- 正規表現は、システムでデータのパターンを検索するための堅牢で、長く確立されてきた方法です。Perl言語の正規表現の機能は、パターン一致ルールの設計上の目標として採用されており、Oracle Database 12*c* リリース1では、これらのルールのサブセットを実装してパターン一致に対応しています。
- Oracleの正規表現は、行パターン変数が文字または文字のセットで定義されるのではなく、ブール条件によって定義されている点で、通常の正規表現と異なります。
- パターン一致では正規表現の表記法を使用してパターンを表現していますが、前の行と行パターン変数とのマッピング方法に依存するようパターン変数を定義することも可能なため、これは実際には、より高度な機能です。DEFINE句では、パターン変数を他のパターン変数上に構築できます。
- 行パターン変数の定義とメジャーの定義では、副問合せが使用可能です。

21.2 パターン一致の基本トピック

この項では、次の内容を説明します。

- 基本的なパターン一致の例
- パターン一致のタスクとキーワード
- パターン一致の構文

21.2.1 基本的なパターン一致の例

この項では、一致するパターンの基本的な例を示しています。

例21-2 一致ごとに全行を出力する単純なV字形のパターン一致

この例の1行目は、SQL*Plusを使用している場合のフォーマッティングを改善するものです。

```
column var_match format a4

SELECT *
FROM Ticker MATCH_RECOGNIZE (
     PARTITION BY symbol
     ORDER BY tstamp
     MEASURES  STRT.tstamp AS start_tstamp,
               FINAL LAST(DOWN.tstamp) AS bottom_tstamp,
               FINAL LAST(UP.tstamp) AS end_tstamp,
               MATCH_NUMBER() AS match_num,
               CLASSIFIER() AS var_match
     ALL ROWS PER MATCH
     AFTER MATCH SKIP TO LAST UP
     PATTERN (STRT DOWN+ UP+)
     DEFINE
        DOWN AS DOWN.price < PREV(DOWN.price),
        UP AS UP.price > PREV(UP.price)
     ) MR
ORDER BY MR.symbol, MR.match_num, MR.tstamp;
```

```
SYMBOL     TSTAMP    START_TST BOTTOM_TS END_TSTAM  MATCH_NUM VAR_      PRICE
---------- --------- --------- --------- --------- ---------- ---- ----------
ACME       05-APR-11 05-APR-11 06-APR-11 10-APR-11          1 STRT         25
ACME       06-APR-11 05-APR-11 06-APR-11 10-APR-11          1 DOWN         12
ACME       07-APR-11 05-APR-11 06-APR-11 10-APR-11          1 UP           15
ACME       08-APR-11 05-APR-11 06-APR-11 10-APR-11          1 UP           20
ACME       09-APR-11 05-APR-11 06-APR-11 10-APR-11          1 UP           24
ACME       10-APR-11 05-APR-11 06-APR-11 10-APR-11          1 UP           25
ACME       10-APR-11 10-APR-11 12-APR-11 13-APR-11          2 STRT         25
ACME       11-APR-11 10-APR-11 12-APR-11 13-APR-11          2 DOWN         19
ACME       12-APR-11 10-APR-11 12-APR-11 13-APR-11          2 DOWN         15
ACME       13-APR-11 10-APR-11 12-APR-11 13-APR-11          2 UP           25
ACME       14-APR-11 14-APR-11 16-APR-11 18-APR-11          3 STRT         25
ACME       15-APR-11 14-APR-11 16-APR-11 18-APR-11          3 DOWN         14
ACME       16-APR-11 14-APR-11 16-APR-11 18-APR-11          3 DOWN         12
ACME       17-APR-11 14-APR-11 16-APR-11 18-APR-11          3 UP           14
ACME       18-APR-11 14-APR-11 16-APR-11 18-APR-11          3 UP           24

15 rows selected.
```

この問合せでは、何を行っているのでしょうか。これは例21-1の問合せに類似していますが、MEASURES句の項目、ALL ROWS PER MATCHの変更、および問合せの終わりにあるORDER BYの変更が異なります。MEASURES句では、次のように追加されています。

- MATCH_NUMBER() AS match_num

  この例では一致ごとに複数行が出力されるため、どの行がどの一致のメンバーであるかを把握する必要があります。MATCH_NUMBERでは、特定の一致の各行に同じ数値が割り当てられます。たとえば、行パターン・パーティションで最初に見つかった一致のすべての行に、match_numの値として1が割り当てられます。一致の番号付けは、行パターン・パーティションごとに繰り返し1から開始します。

- CLASSIFIER() AS var_match

  どの行がどの変数にマップされたかを把握するには、CLASSIFIER関数を使用します。この例では、いくつかの行がSTRT変数にマップされ、別のいくつかの行がDOWN変数にマップされ、さらに別の行がUP変数にマップされます。

- FINAL LAST()

  FINALを指定し、bottom_tstampに対してLAST()関数を使用することによって、一致ごとの各行でV字形の底部に対して同じ日付が表示されます。同様に、FINAL LAST()をend_tstampメジャーに適用すると、一致ごとの各行でV字形の終わりに対して同じ日付が表示されます。この構文が使用されない場合、表示される日付は行ごとの実行値になります。

他の2つの行で、次のように変更されました。

- ALL ROWS PER MATCH: 例21-1では一致ごとに1行のみを出力するサマリーが行ONE ROW PER MATCHを使用して表示されましたが、この例では一致ごとにすべての行を表示するよう要求しています。
- 最終行のORDER BY: これはMATCH_NUMを利用するように変更されたため、同じ一致内のすべての行が時系列にまとめられます。

April 10の行は2つのパターン一致に存在するため、2回表示されます。これは、最初の一致の最終日と2番目の一致の初日に相当しています。

例21-3 変数に集計を使用するパターン一致

例21-3では、パターン一致問合せで集計関数が使用されていることを強調しています。

```
SELECT *
FROM Ticker MATCH_RECOGNIZE (
  PARTITION BY symbol
  ORDER BY tstamp
  MEASURES
    MATCH_NUMBER() AS match_num,
    CLASSIFIER() AS var_match,
    FINAL COUNT(UP.tstamp) AS up_days,
    FINAL COUNT(tstamp) AS total_days,
    RUNNING COUNT(tstamp) AS cnt_days,
    price - STRT.price AS price_dif
  ALL ROWS PER MATCH
  AFTER MATCH SKIP TO LAST UP
  PATTERN (STRT DOWN+ UP+)
  DEFINE
    DOWN AS DOWN.price < PREV(DOWN.price),
    UP AS UP.price > PREV(UP.price)
  ) MR
ORDER BY MR.symbol, MR.match_num, MR.tstamp;

SYMBOL TSTAMP    MATCH_NUM VAR_ UP_DAYS TOTAL_DAYS CNT_DAYS PRICE_DIF PRICE
------ --------- --------- ---- ------- ---------- -------- --------- -----
ACME   05-APR-11         1 STRT       4          6        1         0    25
ACME   06-APR-11         1 DOWN       4          6        2       -13    12
ACME   07-APR-11         1 UP         4          6        3       -10    15
ACME   08-APR-11         1 UP         4          6        4        -5    20
ACME   09-APR-11         1 UP         4          6        5        -1    24
ACME   10-APR-11         1 UP         4          6        6         0    25
ACME   10-APR-11         2 STRT       1          4        1         0    25
ACME   11-APR-11         2 DOWN       1          4        2        -6    19
ACME   12-APR-11         2 DOWN       1          4        3       -10    15
ACME   13-APR-11         2 UP         1          4        4         0    25
ACME   14-APR-11         3 STRT       2          5        1         0    25
ACME   15-APR-11         3 DOWN       2          5        2       -11    14
ACME   16-APR-11         3 DOWN       2          5        3       -13    12
ACME   17-APR-11         3 UP         2          5        4       -11    14
ACME   18-APR-11         3 UP         2          5        5        -1    24

15 rows selected.
```

この問合せでは、何を行っているのでしょうか。これは、集計関数COUNT()を使用する3つのメジャーを追加して、例21-2上に作成されています。また、式で修飾列と非修飾列を使用できる方法を示すメジャーも追加されています。

- up_daysメジャー(FINAL COUNTを使用)によって、一致ごとにUPパターン変数にマップされる日数が表示されます。これを確認するには、図21-2に示す一致ごとのUPのラベル数をカウントします。
- total_daysメジャー(これもFINAL COUNTを使用)によって、非修飾列の使用が導入されます。このメジャーではパターン変数なしでFINAL count(tstamp)を指定してtstamp列を修飾しているため、1つの一致に含まれているすべての行のカウント数が返されます。
- cnt_daysメジャーにより、RUNNINGキーワードが導入されます。このメジャーは、1つの一致内で行を区別するのに役立つ実行数を示します。tstamp列を修飾するパターン変数もないため、これは1つの一致のすべての行に適用されます。この場合、これはデフォルトであるため、RUNNINGキーワードを明示的に使用する必要はありません。詳細は、実行中および最終セマンティクスとキーワードの比較を参照してください。
- price_difメジャーは、一致の初日の株価と比較した各日の株価の差異を表示します。式price - STRT.priceでは、非修飾列priceが修飾列STRT.priceとともに使用されているケースが見られます。

例21-4 W字形のパターン一致

この例は、W字形を示しています。

```
SELECT *
FROM Ticker MATCH_RECOGNIZE (
  PARTITION BY symbol
  ORDER BY tstamp
  MEASURES
    MATCH_NUMBER() AS match_num,
    CLASSIFIER()  AS  var_match,
    STRT.tstamp AS start_tstamp,
    FINAL LAST(UP.tstamp) AS end_tstamp
  ALL ROWS PER MATCH
  AFTER MATCH SKIP TO LAST UP
  PATTERN (STRT DOWN+ UP+ DOWN+ UP+)
  DEFINE
    DOWN AS DOWN.price < PREV(DOWN.price),
    UP AS UP.price > PREV(UP.price)
  ) MR
ORDER BY MR.symbol, MR.match_num, MR.tstamp;

SYMBOL     TSTAMP     MATCH_NUM  VAR_  START_TST  END_TSTAM      PRICE
---------- --------- ----------  ----  ---------  --------- ----------
ACME       05-APR-11           1  STRT  05-APR-11  13-APR-11          25
ACME       06-APR-11           1  DOWN  05-APR-11  13-APR-11          12
ACME       07-APR-11           1  UP    05-APR-11  13-APR-11          15
ACME       08-APR-11           1  UP    05-APR-11  13-APR-11          20
ACME       09-APR-11           1  UP    05-APR-11  13-APR-11          24
ACME       10-APR-11           1  UP    05-APR-11  13-APR-11          25
ACME       11-APR-11           1  DOWN  05-APR-11  13-APR-11          19
ACME       12-APR-11           1  DOWN  05-APR-11  13-APR-11          15
ACME       13-APR-11           1  UP    05-APR-11  13-APR-11          25
```

この問合せでは、何を行っているのでしょうか。これは、例21-1に導入されたコンセプトに基づいて作成されており、データ内のV字形ではなくW字形をシークしていきます。問合せ結果は1つのW字形として表示されます。W字形を見つけるために、PATTERN正規表現を定義している行がDOWNの後にUPが続く2回連続したパターン(PATTERN (STRT DOWN+ UP+ DOWN+ UP+))をシークできるように変更されました。このパターン指定は、2つのV字形が間を空けていないW字形のみと一致できることを示しています。たとえば、価格が変動しないフラットな間隔が存在し、その間隔が2つのV字形の間に発生する場合、このパターンとデータの一致は生じません。返されたデータを説明するために、出力はALL ROWS PER MATCHに設定されます。MEASURES句のFINAL LAST(UP.tstamp)は、UPに最後にマップされた行のタイムスタンプ値を返します。

21.2.2 パターン一致のタスクとキーワード

この項では、パターン一致における次のタスクとキーワードについて説明します。

PARTITION BY: 行を論理的にグループに分割する

通常は、入力データを分析のために論理グループに分割する必要が生じます。株価の例では、一度に1つの株価のみに適用されるようにパターン一致を分割しています。これは、PARTITION BYキーワードを使用して行います。PARTITION BYを使用して、行パターン入力表の行を1列ごと、または複数列ごとにパーティション化されるように指定します。一致はパーティション内で見つかり、パーティションの境界を超えることはありません。

PARTITION BYが存在しない場合は、行パターン入力表のすべての行で1つの行パターン・パーティションが構成されます。

ORDER BY: パーティション内の行を論理的に順序付ける

入力データを論理パーティションに分割した後で、それぞれのパーティションの中でデータを順序付ける必要が生じます。行の順序付けが行われないと、パターン一致を確認するための信頼できるシーケンスが得られません。ORDER BYキーワードを使用して、1つの行パターン・パーティションでの行の順序が指定されます。

[ONE ROW | ALL ROWS] PER MATCH: 一致ごとにサマリーまたは詳細を選択する

一致に関してサマリー・データが必要な場合と、詳細が必要な場合があります。そのような場合は、次のSQLキーワードを使用して対応できます。

- ONE ROW PER MATCH

  一致ごとに1つのサマリー行が生成されます。これはデフォルトです。

- ALL ROWS PER MATCH

  複数行にわたる一致の場合、その一致で行ごとに1つの出力行が生成されます。

この出力については、行パターンの出力で説明しています。

MEASURES: エクスポートの計算をパターン一致から定義する

パターン一致句の使用によって、広範囲の分析に役立つ式を作成できます。これらの句を出力の列として表示するには、MEASURES句を使用します。MEASURES句では行パターンのメジャーの列が定義され、その値は特定の一致に関連する式を評価して計算されます。

PATTERN: 一致する行パターンを定義する

PATTERN句の使用により、一致する必要のあるパターン変数、そのパターン変数の一致順序、および一致する必要のある行数を定義することができます。PATTERN句では、一致検索に対する正規表現が指定されます。

行パターン一致は、1つの行パターン・パーティション内で連続する行のセットで構成されます。一致の各行はそれぞれ1つのパターン変数にマップされます。行とパターン変数とのマッピングは、PATTERN句の正規表現に準拠している必要があり、DEFINE句のすべての条件を満たす必要があります。

DEFINE: プライマリ・パターン変数を定義する

PATTERN句はパターン変数に依存するため、これらの変数を定義する句を用意する必要があります。これらの変数はDEFINE句で指定されます。

DEFINEは必須の句であり、行を特定のパターン変数にマップするために満たしている必要のある条件の指定に使用されます。

パターン変数は定義を必要としません。任意の行を未定義のパターン変数にマップできます。

AFTER MATCH SKIP: 一致が見つかった後で一致プロセスを再開する

問合せで一致が見つかった後に、次の一致を正しい場所で検索する必要があります。前の一致の終わりが次の一致の始まりと重なる一致を検索する必要があるでしょうか。その他のタイプの一致が必要ですか。パターン一致は、再開場所の指定に非常に柔軟に対応できます。AFTER MATCH SKIP句では、空以外の一致が見つかった後の行パターン一致を再開する場所が判別されます。この句のデフォルトはAFTER MATCH SKIP PAST LAST ROWであり、現在の一致の最終行の次の行からパターン一致が再開されます。

MATCH_NUMBER: どの行がどの一致のメンバーであるかを検索する

指定した行パーティションの中で、パターンに対する一致が多数見つかる場合があります。これらのすべての一致をどのように見分ければよいでしょうか。これを実行するには、MATCH_NUMBER関数を使用します。1つの行パターン・パーティション内の一致には、見つかった順に1から順次番号が付けられます。行パターン・パーティション間では順序付けが継承されないため、一致の番号付は行パターン・パーティションごとに1から繰り返し開始されます。

CLASSIFIER: どのパターン変数がどの行に適用されるかを検索する

現在どのMATCH_NUMBERが表示されているかを知るだけでなく、特定の行にパターンのどの部分が当てはまるのかを知る必要が生じることもあります。これを実行するには、CLASSIFIER関数を使用します。行の分類子は、行パターン一致によってその行がマップされているパターン変数になります。CLASSIFIER関数は、行のマップ先である変数の名前を値として持つ文字列を返します。

21.2.3 パターン一致の構文

パターン一致の構文は次のとおりです。

```
table_reference ::=
  {only (query_table_expression) | query_table_expression }[flashback_query_clause]
   [pivot_clause|unpivot_clause|row_pattern_recognition_clause] [t_alias]

row_pattern_recognition_clause ::=
  MATCH_RECOGNIZE (
   [row_pattern_partition_by ]
   [row_pattern_order_by ]
   [row_pattern_measures ]
   [row_pattern_rows_per_match ]
   [row_pattern_skip_to ]
   PATTERN (row_pattern)
   [ row_pattern_subset_clause]
   DEFINE row_pattern_definition_list
   )

row_pattern_partition_by ::=
   PARTITION BY column[, column]...

row_pattern_order_by ::=
   ORDER BY column[, column]...

row_pattern_measures ::=
   MEASURES row_pattern_measure_column[, row_pattern_measure_column]...

row_pattern_measure_column ::=
   expression AS c_alias

row_pattern_rows_per_match ::=
   ONE ROW PER MATCH
  | ALL ROWS PER MATCH

row_pattern_skip_to ::=
   AFTER MATCH {
    SKIP TO NEXT ROW
   | SKIP PAST LAST ROW
   | SKIP TO FIRST variable_name
   | SKIP TO LAST variable_name
   | SKIP TO variable_name}

row_pattern ::=
   row_pattern_term
  | row_pattern "|" row_pattern_term

row_pattern_term ::=
   row_pattern_factor
  | row_pattern_term row_pattern_factor

row_pattern_factor ::=
   row_pattern_primary [row_pattern_quantifier]
```

```
row_pattern_quantifier ::=
    *[?]
   |+[?]
   |?[?]
   |"{"[unsigned_integer ],[unsigned_integer]"}"[?]
   |"{"unsigned_integer "}"

row_pattern_primary ::=
   variable_name
   |$
   |^
   |([row_pattern])
   |"{-" row_pattern"-}"
   | row_pattern_permute

row_pattern_permute ::=
   PERMUTE (row_pattern [, row_pattern] ...)

row_pattern_subset_clause ::=
   SUBSET row_pattern_subset_item [, row_pattern_subset_item] ...

row_pattern_subset_item ::=
   variable_name = (variable_name[ , variable_name]...)

row_pattern_definition_list ::=
   row_pattern_definition[, row_pattern_definition]...

row_pattern_definition ::=
   variable_name AS condition
```

パターン一致の中で行われる行パターンの操作の構文は、次のとおりです。

```
function ::=
 single_row_function
| aggregate_function
| analytic_function
| object_reference_function
| model_function
| user_defined_function
| OLAP_function
| data_cartridge_function
| row_pattern_recognition_function

row_pattern_recognition_function ::=
 row_pattern_classifier_function
| row_pattern_match_number_function
| row_pattern_navigation_function
| row_pattern_aggregate_function

row_pattern_classifier_function ::=
  CLASSIFIER( )

row_pattern_match_number_function ::=
  MATCH_NUMBER( )

row_pattern_navigation_function ::=
   row_pattern_navigation_logical
  | row_pattern_navigation_physical
  | row_pattern_navigation_compound
```

```
row_pattern_navigation_logical ::=
   [RUNNING|FINAL] {FIRST|LAST} (expression[,offset])

row_pattern_navigation_physical ::=
   {PREV|NEXT}(expression[, offset])

row_pattern_navigation_compound ::=
   {PREV | NEXT} (
     [RUNNING| FINAL] {FIRST|LAST} (expression[, offset]) [,offset])
```

パターン一致句の中でのset関数指定の構文は、次のとおりです。

```
row_pattern_aggregate_function ::=
   [RUNNING | FINAL] aggregate_function
```

21.3 パターン一致の詳細

この項では、パターン一致の構文で説明した項目の詳細に加えて、追加のトピックについても説明しています。内容の一部がやむを得ず複雑になっていることに留意してください。パターン一致には、難解な詳細に特別の注意が求められる場合があります。

- PARTITION BY: 行を論理的にグループに分割する
- ORDER BY: パーティション内の行を論理的に順序付ける
- [ONE ROW | ALL ROWS] PER MATCH: 一致ごとにサマリーまたは詳細を選択する
- MEASURES: 問合せに使用する計算を定義する
- PATTERN: 一致する行パターンを定義する
- SUBSET: 共用体行パターン変数を定義する
- DEFINE: プライマリ・パターン変数を定義する
- AFTER MATCH SKIP: 一致が見つかった後の一致プロセスの再開場所を定義する
- MEASURESおよびDEFINEの式
- 行パターンの出力

21.3.1 PARTITION BY: 行を論理的にグループに分割する

通常は、分析のために入力データを論理グループに分割する必要があります。株価の例では、一度に1つの株価のみに適用されるようにパターン一致が分割されています。これを実行するには、PARTITION BY句を使用します。PARTITION BYは、入力表の行が1列または複数列のパーティションに区分されるように指定します。一致はパーティション内で見つかり、パーティションの境界を超えることはありません。

PARTITION BYが存在しない場合は、行パターン入力表のすべての行で1つの行パターン・パーティションが構成されます。

21.3.2 ORDER BY: パーティション内の行を論理的に順序付ける

ORDER BY句は、1つの行パターン・パーティション内での行の順序の指定に使用します。行パターン・パーティションの2つの行の順序がORDER BYによって確定されていない場合、MATCH_RECOGNIZE句の結果は非確定的になります。つまり、問合せが実行されるたびに一貫性のある結果が得られない場合があります。

21.3.3 [ONE ROW | ALL ROWS] PER MATCH: 一致ごとにサマリーまたは詳細を選択する

一致に関してサマリー・データが必要な場合と、詳細が必要な場合があります。これは、次のSQLを使用して処理できます。

- ONE ROW PER MATCH

  一致ごとに1つのサマリー行が生成されます。これはデフォルトです。

- ALL ROWS PER MATCH

  複数行にわたる一致の場合、その一致で行ごとに1つの出力行が生成されます。

この出力については、行パターンの出力で説明しています。

MATCH_RECOGNIZE句では、行がまったく出力されない一致が見つかることがあります。空の一致の場合、ONE ROW PER MATCHはサマリー行を返します。つまり、PARTITION BY列では空の一致が発生した行から値が取り込まれ、メジャー列は行の空のセットに対して評価されます。

ALL ROWS PER MATCHには、次の3つのサブオプションがあります。

- ALL ROWS PER MATCH SHOW EMPTY MATCHES
- ALL ROWS PER MATCH OMIT EMPTY MATCHES
- ALL ROWS PER MATCH WITH UNMATCHED ROWS

これらのオプションについては、パターン一致の高度なトピックで説明しています。

21.3.4 MEASURES: 問合せに使用する計算を定義する

MEASURES句は、パターン出力表の列のリストを定義します。各パターンのメジャー列は、対応するパターン・メジャー式により指定された値の列名を使用して定義されます。

値の式は、パターン変数に関連して定義されます。値の式には、set関数、パターン・ナビゲーション操作、CLASSIFIER()、MATCH_NUMBER()、および入力表の任意の列に対する列参照を含めることができます。詳細は、MEASURESおよびDEFINEの式を参照してください。

21.3.5 PATTERN: 一致する行パターンを定義する

PATTERNキーワードは、パーティション内の行の順序どおりにパターンが認識されるように指定します。パターンの各変数名はブール条件に対応しています。ブール条件は、構文のDEFINEコンポーネントを使用して後で指定されます。

PATTERN句は、正規表現の指定に使用します。正規表現の概念と詳細についての説明は、本資料では対象外です。正規表現に習熟していない場合は、他の資料を利用して習熟するようにしてください。

PATTERN句の正規表現は、丸括弧で囲まれています。PATTERNでは、次の演算子を使用する場合があります。

- 連結

  連結は、一致の対象であるパターン内の2つ以上の項目を順序どおりにリストするために使用します。2つの連続する項目の間に演算子符号がない場合、項目は連結されます。例: PATTERN (A B C)。

- 数量詞

  数量子は、一致で受け入れられる反復数を定義するPOSIX演算子です。POSIX拡張正規表現の構文は、従来のUNIX正規表現の構文に似ています。数量子の選択肢を次に示します。

  - *: 0以上の反復

- +: 1以上の反復
- ?: 0または1の反復
- {n}: n個の反復(n > 0)
- {n,}: n個以上の反復(n >= 0)
- {n,m}: nからm個(これを含む)の間の反復(0 <= n <= m、0 < m)
- {,m} : 0からm個(これを含む)の間の反復(m > 0)
- 最短一致数量子: 数量子の後に疑問符を追加して示します(*?、+?、??、{n,}?、{ n, m }?、{,m}?)。最短一致の数量子と非最短一致の数量子の相違は、最短一致数量子と強欲な数量子の比較を参照してください。

次に、数量子演算子の使用例を示します。

- A*は、Aの0以上の反復に一致します。
- A{3,6}は、Aの3から6個の反復に一致します。
- A{,4}は、Aの0から4個の反復に一致します。

- 代替

  代替では、複数の使用可能な正規表現のリスト内の1つの正規表現に一致させます。代替リストは、それぞれの正規表現の間に縦線(|)を入れて作成されます。代替項目は、指定された順序で優先されます。一例として、PATTERN (A | B | C)は、最初にAとの一致を試みます。Aが一致しない場合は、Bとの一致を試みます。Bが一致しない場合は、Cとの一致を試みます。

- グループ化

  グループ化では、正規表現の一部分を単一ユニットとして扱い、数量子などの正規表現演算子をそのグループに適用できるようにします。グループ化はカッコを使って作成されます。一例として、PATTERN ((A B){3} C)はグループ(A B)との一致を3回試みて、Cの1回の出現を探します。

- PERMUTE

  詳細は、すべての順列の表現方法を参照してください。

- 除外

  ALL ROWS PER MATCHの出力から除外されるパターンの部分は、{-と-}で括られます。パターンの部分を出力から除外する方法.を参照してください。

- アンカー

  アンカーは、行ではなく位置に関して機能します。アンカーは、パーティションの始まりまたは終わりの位置と一致します。

  - ^は、パーティションの最初の行の前の位置と一致します。
  - $は、パーティションの最後の行の後の位置と一致します。

  たとえば、PATTERN (^A+$)は、パーティションのすべての行がAの条件を満たす場合のみ、一致します。その結果、パーティション全体にわたって一致します。

- 空のパターン()は、空の行セットと一致します。

この項では、次の項目について説明します。

- 最短一致数量子と強欲な数量子の比較

- 演算子の優先順位

21.3.5.1 最短一致数量子と強欲な数量子の比較

パターン数量子を強欲な数量子といい、適用されている正規表現で最大限のインスタンスの一致を試みます。その例外が、接尾辞として疑問符?を持つパターン数量子です。これらは、最短一致数量子と呼ばれます。これらは、適用されている正規表現で最小限のインスタンスの一致を試みます。

1つのパターン変数に付加される強欲な数量子と最短一致数量子の相違は、「A*はできるかぎり多くの行をAにマップしようとするのに対し、A*?はできるかぎり少ない行をAにマップしようとする」のように説明できます。次に例を示します。

```
PATTERN (X Y* Z)
```

このパターンは3つの変数名、X、YおよびZで構成され、Yは*で数量化されています。これは、連続する入力行が次の条件を満たしたときに、パターン一致が認識され、報告されることを示しています。

- 1つの行が変数Xを定義する条件を満たし、その後に変数Yを定義する条件を満たす0行以上の行が続き、さらにその後に変数Zを定義する条件を満たす行が1つ続く。

パターン一致プロセスの際、1つの行がXにマップされ、0行以上の行がYにマップされた後、それに続く行を変数YとZの両方にマップできる場合(これはYとZの両方の定義条件を満たします)、このとき、Yの数量子*は強欲な数量子であるので、その行はZよりもYに優先的にマップされます。この強欲という特性のため、YはZよりも優先され、Yに対してより多くの行がマップされます。パターン表現がYで最短一致数量子*?を使用するPATTERN (X Y*? Z)であった場合、ZはYよりも優先されます。

21.3.5.2 演算子の優先順位

正規表現の要素の優先順位は、降順で次のとおりです。

- row_pattern_primary

  これらの要素には、プライマリ・パターン変数(SUBSET: 共用体行パターン変数を定義するで説明するSUBSET句を使用して作成されていないパターン変数)、アンカー、PERMUTE、挿入句、除外構文および空パターンがあります。

- 数量詞

  row_pattern_primaryは、0または1つの数量子を持つ場合があります。

- 連結
- 代替

代替の優先順位はPATTERN(A B | C D)で説明できます。これは、PATTERN ((A B) | (C D))と同等です。ただし、PATTERN (A (B | C) D)と同等ではありません。

数量子の優先順位はPATTERN (A B *)によって説明できます。これは、PATTERN (A (B*))と同等です。ただし、PATTERN ((A B)*)と同等ではありません。

数量子が別の数量子の直後に続かない場合もあります。たとえば、PATTERN(A**)は禁止されています。

プライマリ・パターン変数がパターン内に複数回出現することは許可されています(例: IPATTERN (X Y X))。

21.3.6 SUBSET: 共用体行パターン変数を定義する

それ自身の変数名を使って参照可能な複数のパターン変数のグループ化を作成すると便利な場合があります。このようなグループ化を共用体行パターン変数といい、SUBSET句を使って作成できます。SUBSETで作成された共用体行パターン変数は、MEASURES句およびDEFINE句に使用できます。SUBSET句はオプションです。これは、共用体行パターン変数の宣言に使用します。たとえば、SUBSETを使用して、STRT変数とDOWN変数(この場合のSTRTはパターンの開始点、DOWNはV字形の下方(左)部

分)の共用体にマップされるすべての行に基づいて平均値を計算する問合せがあるとします。

例21-5では、共用体行パターン変数の作成について説明しています。

例21-5 共用体行パターン変数の定義

```
SELECT *
FROM Ticker MATCH_RECOGNIZE(
     PARTITION BY symbol
     ORDER BY tstamp
     MEASURES FIRST(STRT.tstamp) AS strt_time,
              LAST(DOWN.tstamp) AS bottom,
              AVG(STDN.Price) AS stdn_avgprice
     ONE ROW PER MATCH
     AFTER MATCH SKIP TO LAST UP
     PATTERN (STRT DOWN+ UP+)
     SUBSET STDN= (STRT, DOWN)
     DEFINE
        UP AS UP.Price > PREV(UP.Price),
        DOWN AS DOWN.Price < PREV (DOWN.Price)
);

SYMBOL    STRT_TIME BOTTOM    STDN_AVGPRICE
------    --------- --------- -------------
ACME      05-APR-11 06-APR-11          18.5
ACME      10-APR-11 12-APR-11    19.6666667
ACME      14-APR-11 16-APR-11            17
```

この例では、1つの共用体行パターン変数、STDNを宣言し、それをSTRTにマップされる行とDOWNにマップされる行の共用体として定義しています。1つの問合せに複数の共用体行パターン変数を使用できます。次に例を示します。

```
PATTERN (W+ X+ Y+ Z+)
SUBSET XY = (X, Y),
       WZ = (W, Z)
```

SUBSET項目の右側は、カッコで括られた個別のプライマリ行パターン変数のカンマ区切りリストです。これは、(左辺の)共用体行パターン変数を(右辺の)プライマリ行パターン変数の共用体として定義します。

右側のパターン変数のリストには、共用体行パターン変数がまったく含まれない場合があります(共用体の共用体がないため)。

一致ごとに、ユニバーサル行パターン変数と呼ばれる暗黙的な共用体行パターン変数が1つ存在します。ユニバーサル行パターン変数は、すべてのプライマリ行パターン変数の共用体です。たとえば、パターンにプライマリ・パターン変数A、BおよびCが存在する場合、ユニバーサル行パターン変数は引数(A, B, C)を持つSUBSET句と同等になります。したがって、一致のすべての行はユニバーサル行パターン変数にマップされます。MEASURES句またはDEFINE句の中の非修飾列参照はすべて、ユニバーサル行パターン変数によって暗黙的に修飾されます。ユニバーサル行パターン変数を明示的に指定するキーワードはありません。

21.3.7 DEFINE: プライマリ・パターン変数を定義する

DEFINEは必須の句であり、プライマリ・パターン変数を定義する条件の指定に使用されます。この例では次のようになります。

```
DEFINE UP AS UP.Price > PREV(UP.Price),
DOWN AS DOWN.Price < PREV(DOWN.Price)
```

UPは条件UP.Price > PREV (UP.Price)によって定義され、DOWNは条件DOWN.Price < PREV (DOWN.Price)によって定義されます。(PREVは、前の行の式を評価する行パターンのナビゲーション操作です。行パターンのナビゲーション操作の完全なセットについては、行パターンのナビゲーション操作を参照してください。)

パターン変数は定義を必要としません。定義が存在しない場合は、パターン変数に任意の行をマップできます。

共用体行パターン変数(SUBSET: 共用体行パターン変数を定義するのSUBSETの説明を参照)はDEFINEでは定義できませんが、パターン変数の定義で参照できます。

パターン変数の定義は、別のパターン変数を参照できます。これについては、例21-6で説明しています。

例21-6 パターン変数の定義

```
SELECT *
FROM Ticker MATCH_RECOGNIZE (
     PARTITION BY Symbol
     FROM Ticker
     MATCH_RECOGNIZE (
     PARTITION BY Symbol
     ORDER BY tstamp
     MEASURES FIRST (A.tstamp) AS A_Firstday,
              LAST (D.tstamp) AS D_Lastday,
              AVG (B.Price) AS B_Avgprice,
              AVG (D.Price) AS D_Avgprice
     PATTERN (A B+ C+ D)
     SUBSET BC = (B,C)
     DEFINE A AS Price > 100,
            B AS B.Price > A.Price,
            C AS C.Price < AVG (B.Price),
            D AS D.Price > MAX (BC.Price)
) M
```

この例では、次のようになります。

- Aの定義は、ユニバーサル行パターン変数を暗黙的に参照しています(Priceが非修飾列参照のため)。
- Bの定義は、パターン変数Aを参照しています。
- Cの定義は、パターン変数Bを参照しています。
- Dの定義は、共用体行パターン変数BCを参照しています。

条件が試行一致でのパーティションの連続行で評価され、現在の行がこのパターンで許可されているパターン変数に試験的にマップされます。正常にマップされるためには、条件がtrueに評価される必要があります。

前述の例では次のようになります。

```
A AS Price > 100
```

Priceは現在行のPriceを参照します。なぜなら、プライマリ行パターン変数にマップされている最後の行が現在行であり、これがAに試験的にマップされているからです。この例では、かわりに、A.Priceを使用しても同じ結果になります。

```
B AS B.Price > A.Price
```

B.Priceは現在行のPriceを参照しています(Bが定義中であるため)が、A.PriceはAに最後にマップされた行を参照しています。このパターンを調べてみると、Aにマップされた行は、マッピングの対象である最初の行のみです。

```
C AS C.Price < AVG(B.Price)
```

ここでは、C.Priceは現在の行でPriceを参照していますが、これはCが定義中であるためです。集計AVG(つまり、挿入Price)は、すでにBにマップされたすべての行の平均値として計算されています。

```
D AS D.Price > MAX(BC.Price)
```

パターン変数Dはパターン変数Cと類似していますが、ブール条件における共用体行パターン変数の使用について説明しています。このケースでは、MAX(BC.Price)は、変数Bまたは変数Cにマップされている行の最大価格値を返します。ブール条件のセマ

ンティクスについては、MEASURESおよびDEFINEの式で詳しく説明しています。

21.3.8 AFTER MATCH SKIP: 一致が見つかった後の一致プロセスの再開場所を定義する

AFTER MATCH SKIP句では、空以外の一致が見つかった後の行パターン一致を再開する場所が判別されます。この句のデフォルトは、AFTER MATCH SKIP PAST LAST ROWです。使用できるオプションは次のとおりです。

- AFTER MATCH SKIP TO NEXT ROW

  現在の一致の最初の行の後の行でパターン一致を再開します。

- AFTER MATCH SKIP PAST LAST ROW

  現在の一致の最後の行の次の行でパターン一致を再開します。

- AFTER MATCH SKIP TO FIRST *pattern_variable*

  パターン変数にマップされた最初の行でパターン一致を再開します。

- AFTER MATCH SKIP TO LAST *pattern_variable*

  パターン変数にマップされた最後の行でパターン一致を再開します。

- AFTER MATCH SKIP TO *pattern_variable*

  AFTER MATCH SKIP TO LAST *pattern_variable*と同じです。

AFTER MATCH SKIP TO FIRSTまたはAFTER MATCH SKIP TO [LAST]を使用すると、pattern_variableに行がマップされないことがあります。次に例を示します。

```
AFTER MATCH SKIP TO A
PATTERN (X A* X),
```

例に示したパターン変数Aには、Aにマップされた行がない場合があります。Aにマップされた行がない場合は、スキップする行も存在しないため、ランタイム例外が生成されます。別の問題として、AFTER MATCH SKIPが最後の一致が開始された行と同じ行でパターン一致を再開しようとすることがあります。次に例を示します。

```
AFTER MATCH SKIP TO X
PATTERN (X Y+ Z),
```

この例では、AFTER MATCH SKIP TO Xは、前の一致が見つかった行と同じ行でパターン一致を再開しようとしています。その結果、無限ループに入り、このシナリオに対してランタイム例外が生成されます。

AFTER MATCH SKIP構文は、空以外の一致の後で一致のスキャン再開場所を判別するのみです。空の一致が見つかった場合は、1行をスキップします(SKIP TO NEXT ROWが指定された場合と同じ)。したがって、空の一致によってこれらの例外が発生することはありません。これらの例外のいずれか1つを取得する問合せは、たとえば次のように再作成してください。

```
AFTER MATCH SKIP TO A
PATTERN (X (A | B) Y)
```

これにより、行がBにマップされたときに実行時エラーが発生しますが、これはA.にマップされた行がないためです。AまたはBのいずれかにスキップする場合は、次のようにします。

```
AFTER MATCH SKIP TO C
PATTERN (X (A | B) Y)
SUBSET C = (A, B)
```

修正された例では、AまたはBのどちらが一致しているかに関係なく、実行時エラーが生じる可能性はありません。

もう1つ、例を示します。

```
AFTER MATCH SKIP TO FIRST A
PATTERN (A* X)
```

この例では、最初の一致の後で、一致の最初の行にスキップしたため(A*が一致している場合)、または存在しない行にスキップしたために(A*が一致していない場合)、例外を取得しています。この例の場合は、SKIP TO NEXT ROWを選択することをお薦めします。

ALL ROWS PER MATCHをAFTER MATCH SKIP PAST LAST ROW以外のスキップ・オプションとともに使用すると、連続する一致が重なることがあります。その場合、行パターン入力表の行Rが複数の一致に出現する可能性があります。その場合、行パターン出力表には、行が関係している一致ごとに1行ずつ表示されます。行パターン入力表の行が複数の一致に関係している場合は、MATCH_NUMBER関数を使用してそれぞれの一致を区別できます。行が複数の一致に関係している場合は、一致ごとに異なる分類子を使用できます。

21.3.9 MEASURESおよびDEFINEの式

パターン一致では、次のように、行パターン一致に固有のスカラー式が使用できます。

- 行パターンのナビゲーション操作(関数PREV、NEXT、FIRSTおよびLASTを使用)。行パターンのナビゲーション操作については、行パターンのナビゲーション操作で説明しています。
- 行パターン・パーティション内の行パターン一致の連番を返すMATCH_NUMBER関数(MATCH_NUMBER: どの行がどの一致にあるかを検索するで説明)。
- 行のマップ先であるプライマリ行パターン変数の名前を返すCLASSIFIER関数(CLASSIFIER: どのパターン変数がどの行に適用されるかを検索するで説明)。

MEASURES句とDEFINE句の式の構文とセマンティクスは、次の例外を除き、同じです。

- DEFINE句は実行中のセマンティクスのみをサポートします。
- MEASURES句のデフォルトは実行中のセマンティクスですが、最終セマンティクスもサポートしています。この区別については、実行中セマンティクスと最終セマンティクスの比較で説明しています。

式の使用

この項では、パターン一致で式を使用する際の考慮点について説明します。内容は次のとおりです。

- MATCH_NUMBER: どの行がどの一致にあるかを検索する
- CLASSIFIER: どのパターン変数がどの行に適用されるかを検索する
- 行パターンの列参照
- 集計
- 行パターンのナビゲーション操作

21.3.9.1 MATCH_NUMBER: どの行がどの一致にあるかを検索する

1つの行パターン・パーティション内の一致には、見つかった順に1から順次番号が付けられます。行パターン・パーティション間では順序付けが継承されないため、一致の番号付は行パターン・パーティションごとに1から繰り返し開始されます。
MATCH_NUMBER()は、行パターン・パーティション内の一致の連番を表すスケール0(ゼロ)の数値を返す関数です。

MATCH_NUMBER()を使用している前の例では、これはMEASURES句で使用されています。MATCH_NUMBER()をDEFINE句で使用することもできます。その場合は、一致番号に依存する条件の定義に使用できます。

21.3.9.2 CLASSIFIER: どのパターン変数がどの行に適用されるかを検索する

CLASSIFIER関数は、行のマップ先であるパターン変数の名前を値として持つ文字列を返します。CLASSIFIER関数は、MEASURES句とDEFINE句の両方で使用可能です。

DEFINE句では、CLASSIFIER関数は現在行のマップ先であるプライマリ・パターン変数の名前を返します。

MEASURES句の場合:

- ONE ROW PER MATCHが指定された場合、問合せはMEASURES句の処理時に一致の最後の行を使用しているため、CLASSIFIER関数は一致の最後の行のマップ先であるパターン変数の名前を返します。
- ALL ROWS PER MATCHが指定された場合、CLASSIFIER関数は、見つかった一致の行ごとに、行のマップ先であるパターン変数の名前を返します。

空の一致の開始行に対する分類子はNULL値になります。

21.3.9.3 行パターンの列参照

行パターン列参照は、次に示すような明示的または暗黙的なパターン変数によって修飾された列名です。

```
A.Price
```

Aはパターン変数であり、Priceは列名です。修飾子のない列名(Priceなど)は、一致のすべての行のセットを参照するユニバーサル行パターン変数により暗黙的に修飾されます。列参照は、その他の構文要素、特に集計とナビゲーション演算子の中にネストできます。(ただし、行パターン一致でのネストは、MATCH_RECOGNIZE句で禁止されたネストでFROM句に関して説明する制限を受けます。)

パターン列参照は、次のように分類されています。

- 集計(SUMなど)の中でネストする場合: 集計行パターンの列参照。
- 行パターンのナビゲーション操作(PREV、NEXT、FIRSTおよびLAST)の中でネストする場合: ナビゲートされた行パターンの列参照。
- その他: 通常の行パターンの列参照。

集計または行パターンのナビゲーション操作内のパターン列参照はすべて、同じパターン変数で修飾される必要があります。次に例を示します。

```
PATTERN (A+ B+)
DEFINE B AS AVG(A.Price + B.Tax) > 100
```

この例では、AとBが別々のパターン変数であるため、構文エラーになります。集計セマンティクスには単一の行セットが必要です。A.Price + B.Taxを評価する際に使用される単一の行セットを構成する方法はありません。ただし、次の方法は受け入れられます。

```
DEFINE B AS AVG (B.Price + B.Tax) > 100
```

この例では、集計内のすべてのパターン列参照がBによって修飾されています。

非修飾の列参照が、一致内のすべての行のセットを参照するユニバーサル行パターン変数により暗黙的に修飾されます。次に例を示します。

```
DEFINE B AS AVG(Price + B.Tax) > 1000
```

この例では、非修飾の列参照Priceがユニバーサル行パターン変数によって暗黙的に修飾されているのに対し、B.TaxはBによって明示的に修飾されているため、構文エラーが生じます。ただし、次の方法は受け入れられます。

```
DEFINE B AS AVG (Price + Tax) > 1000
```

この例では、PriceとTaxが両方とも、ユニバーサル行パターン変数によって暗黙的に修飾されています。

21.3.9.4 集計

集計(COUNT、SUM、AVG、MAXおよびMIN)は、MEASURES句とDEFINE句の両方で使用できます。DISTINCTキーワードはサポート対象外であることに留意してください。行パターン一致で使用される場合、集計は、実行中または最終のセマンティクスのいずれかを使用して特定のパターン変数にマップされた行のセット上で動作します。次に例を示します。

```
MEASURES SUM (A.Price) AS RunningSumOverA,
         FINAL SUM(A.Price) AS FinalSumOverA
ALL ROWS PER MATCH
```

この例では、Aがパターン変数です。最初のパターン・メジャー、RunningSumOverAはRUNNINGまたはFINALのいずれも指定していないため、RUNNINGがデフォルトになります。これは、現在の一致によってAにマップされる、現在まで(現在行も含む)の行におけるPriceの合計として計算されることを示しています。2番目のパターン・メジャー、FinalSumOverAは、現在の一致によってAにマップされるすべての行(現在行より後になる行も含む)にわたるPriceの合計を計算します。最終的な集計は、DEFINE句ではなく、MEASURES句のみで使用可能です。

集計に含まれている非修飾の列参照は、現在のパターン一致のすべての行を参照するユニバーサル行パターン変数によって暗黙的に修飾されます。次に例を示します。

```
SUM (Price)
```

現在の行パターン一致のすべての行にわたるPriceの実行中の合計が計算されます。

集計に含まれている列参照はすべて、同じパターン変数で修飾する必要があります。次に例を示します。

```
SUM (Price + A.Tax)
```

Priceはユニバーサル行パターン変数によって暗黙的に修飾されているのに対し、A.Taxは Aによって明示的に修飾されているため、構文エラーが発生します。

COUNT集合にはパターン一致用に特別の構文があるため、COUNT(A.*)を指定できます。COUNT(A.*)は、現在のパターン一致によってパターン変数Aにマップされる行数です。COUNT(*)については、*がユニバーサル行パターン変数の行を暗黙的にカバーするため、COUNT(*)は現在のパターン一致の行数になります。

21.3.9.5 行パターンのナビゲーション操作

4つの関数、PREV、NEXT、FIRSTおよびLASTがあり、物理的または論理的なオフセットによって行パターン内でのナビゲーションを可能にします。

21.3.9.5.1 PREVとNEXT

PREV関数は、パーティション内で前の行を使用する式の評価に使用できます。これは物理行に関して動作し、特定の変数にマップされた行に制限されません。前の行が存在しない場合は、NULL値が返されます。次に例を示します。

```
DEFINE A AS PREV (A.Price) > 100
```

この例によると、現在行の前の行の価格が100より大きい場合は、現在行をAにマップできます。前の行が存在しない場合(つまり、現在行が行パターン・パーティションの第1行である場合)、PREV(A.Price)はNULLになり、条件はTrueではありません。したがって、第1行はAにマップできません。

パターン変数Aを定義する際に別のパターン変数(Bなど)を使用できます。さらに、条件によってPREV()関数をその別のパターン変数に適用できます。これは、次の状況に似ています。

```
DEFINE A AS PREV (B.PRICE) > 100
```

この場合、PREV()関数でナビゲーション用に使用される開始行は、パターン変数Bにマップされた最後の行になります。

PREV関数はオプションの負以外の整数の引数を受け入れて、前の行に対して物理オフセットを指定できます。次のようになります:

- PREV (A.Price, 0)はA.Priceと同等になります。
- PREV (A.price, 1)は、PREV (A.Price)と同等になります。ノート: 1はデフォルトのオフセットです。
- PREV (A.Price, 2)は、実行中セマンティクスによる、Aで表した行の2行前の行にあるPriceの値です。(Aにマップされた行がない場合、または前に2行が存在しない場合、PREV (A.Price, 2)はNULLになります。)

オフセットは、列や副問合せではなく、ランタイム定数(リテラル、バインド変数およびそれらを含む式)である必要があります。

NEXT関数は、PREV関数の前方向バージョンです。これを使用して、物理オフセットを使用し、行パターン・パーティション内で前方向に行を参照することができます。構文はPREVと同じですが、関数の名前が異なります。次に例を示します。

```
DEFINE A AS NEXT (A.Price) > 100
```

この例では、行パターン・パーティションで前方向に1行を参照しています。パターン一致では、行とパターン変数との将来的なマッピング予測が困難であるため、DEFINE句で現在行より後を参照する集計はサポートされません。NEXT関数は、将来的な行のマッピングを知る必要のない物理オフセットを基準として将来の行にナビゲートするため、この原則に違反しません。

たとえば、前後それぞれ2行の平均値の2倍以上の値を持つ単独行を見つけるには、NEXTを使用して表現できます。

```
PATTERN ( X )
DEFINE X AS X.Price > 2 * ( PREV (X.Price, 2)
       + PREV (X.Price, 1)
       + NEXT (X.Price, 1)
       + NEXT (X.Price, 2) ) / 4
```

PREVまたはNEXTを評価している行は、引数内のパターン変数にマップされるとはかぎりません。たとえば、この例では、PREV (X.Price, 2)を評価している行は一致に含まれていません。パターン変数の目的は、最終的に到達する行ではなく、オフセット元となる行を特定することです。(パターン変数の定義がPREV()またはNEXT()の中でそれ自身を参照している場合、オフセット元の行として現在行を参照していることになります。)これについては、パターン一致におけるPREVおよびNEXT内でのFIRSTとLASTのネストで詳しく説明しています。

PREVとNEXTは、複数の列参照とともに使用することもできます。次に例を示します。

```
DEFINE A AS PREV (A.Price + A.Tax) < 100
```

PREVまたはNEXTの最初の引数として複雑な式を使用する場合、すべての修飾子は同じパターン変数である必要があります(この例では、A)。

PREVとNEXTは、常に実行中セマンティクスを持っています。キーワードRUNNINGとFINALをPREVまたはNEXTとともに使用することはできません。(実行中および最終セマンティクスとキーワードの比較の項を参照してください。)最終セマンティクスを取得するには、たとえば、パターン一致におけるPREVおよびNEXT内でのFIRSTとLASTのネストで説明するように、PREV (FINAL LAST (A.Price))を使用します。

21.3.9.5.1.1 FIRSTとLAST

PREVおよびNEXT関数とは対照的に、FIRSTおよびLAST関数は、パターン変数にマップされた行のみをナビゲートします。つまり、これらの関数は物理オフセットではなく、論理オフセットを使用します。FIRSTは、パターン変数にマップされた行グループの最初の行で評価された式の値を返します。次に例を示します。

```
FIRST (A.Price)
```

Aにマップされている行がない場合、この値はNULLになります。

同様に、LASTは、パターン変数にマップされた行のグループの最後の行で評価された式の値を返します。次に例を示します。

```
LAST (A.Price)
```

この例では、Aにマップされた最終行でA.Priceが評価されます(このような行が存在しない場合は、NULLになります)。

FIRSTおよびLAST演算子は、オプションの負以外の整数の引数を受け入れて、パターン変数にマップされた行のセット内で論理オフセットを指定できます。次に例を示します。

```
FIRST (A.Price, 1)
```

この行では、Aにマップされた2番目の行でPriceを評価します。表21-1のデータセットとマッピングについて考えてみます。

表21-1 パターンおよび行

行	価格	マッピング
R1	10	A
R2	20	B
R3	30	A
R4	40	C
R5	50	A

次のようになります。

- FIRST (A.Price) = FIRST (A.Price, 0) = LAST (A.Price, 2) = 10
- FIRST (A.Price, 1) = LAST (A.Price, 1) = 30
- FIRST (A.Price, 2) = LAST (A.Price, 0) = LAST (A.Price) = 50
- FIRST (A.Price, 3)およびLAST (A.Price, 3)はNULLです。

オフセットは、パターン変数Aにマップされる行のセット{R1, R3, R5}の中で移動する論理オフセットです。PREVまたはNEXTの場合のような物理オフセットではありません。

オプションの整数引数は、列や副問合せではなく、ランタイム定数(リテラル、バインド変数およびそれらを含む式)である必要があります。

FIRSTまたはLASTの最初の引数には、少なくとも1つの行パターンの列参照が設定されている必要があります。したがって、FIRST(1)は構文エラーになります。

FIRSTまたはLASTの最初の引数には、複数の行パターンの列参照が設定されていることがありますが、その場合は、すべての修飾子は同じパターン変数である必要があります。たとえば、FIRST (A.Price + B.Tax)は構文エラーですが、FIRST (A.Price + A.Tax)は受け入れられます。

FIRSTおよびLASTは、実行中セマンティクスと最終セマンティクスを両方ともサポートしています。RUNNINGキーワードはデフォルトであり、DEFINE句で唯一サポートされているオプションです。MEASURESで最終セマンティクスにアクセスするには、次のようにキーワードFINALを使用します。

```
MEASURES FINAL LAST (A.Price) AS FinalPrice
ALL ROWS PER MATCH
```

21.3.9.6 実行中および最終セマンティクスとキーワードの比較

この項では、RUNNINGおよびFINALを使用する際の留意点について説明します。

21.3.9.6.1 実行中セマンティクスと最終セマンティクスの比較

一連の行でのパターン一致は、通常は増分プロセスと見なされ、行がパターンに一致するかどうかを1行ずつ順次に調べていきます。この増分処理モデルでは、完全なパターンが認識されるまでは、どのステップにおいても部分的な一致を認識できたにすぎず、将来どんな行が追加されるか、その行がどの変数にマップされるかは不明です。したがって、パターン一致では、DEFINE句のブール条件にある行パターンの列参照には実行中セマンティクスが設定されます。つまり、パターン変数によって表現される行セットは、すでにパターン変数にマップされたものであり、現在行も含めて現在行までが対象となりますが、将来の行は対象外です。

完全な一致が確立されると、最終セマンティクスを持つことができるようになります。最終セマンティクスは、一致に成功した最終行では実行中セマンティクスと同じです。DEFINEでは完全一致が達成されたかどうか不確かなため、最終セマンティクスはMEASURESでのみ使用可能です。

キーワードRUNNINGおよびFINALは、それぞれ実行中セマンティクスまたは最終セマンティクスの指定に使用します。これらのキーワードのルールは、RUNNINGおよびFINALキーワードの比較で説明しています。

MEASURESおよびDEFINEでの式評価の基本ルールは次のとおりです。

- パターン変数に関連する式を行グループに対して計算する場合は、そのパターン変数にマップされた行のセットが使用されます。セットが空の場合、COUNTは0になり、パターン変数に関連するその他の式はすべてNULLになります。
- 式で単一行での評価が必要な場合は、セットの最後の行が使用されます。セットが空の場合、式はNULLになります。

たとえば、次の図21-7の表と問合せを考えてみます。

図21-7 実行中セマンティクスと最終セマンティクスの比較

```
SELECT M.Symbol, M.Tstamp, M.Price, M.RunningAvg, M.FinalAvg
FROM TICKER MATCH_RECOGNIZE (
     PARTITION BY Symbol
     ORDER BY tstamp
     MEASURES RUNNING AVG (A.Price) AS RunningAvg,
              FINAL AVG (A.Price) AS FinalAvg
     ALL ROWS PER MATCH
     PATTERN (A+)
     DEFINE A AS A.Price >= AVG (A.Price)
     ) M
;
```

表21-2に示す次のデータの順序付き行パターン・パーティションを考えてみます。

表21-2 パターンおよびパーティション化されたデータ

行	記号	タイムスタンプ	価格
R1	XYZ	09-Jun-09	10
R2	XYZ	10-Jun-09	16

行	記号	タイムスタンプ	価格
R3	XYZ	11-Jun-09	13
R4	XYZ	12-Jun-09	9

次のロジックを使用して一致を見つけることができます。

- 行パターン・パーティションの最初の行で、行R1をパターン変数Aに試験的にマップします。この時点で、変数Aにマップされた行のセットは{R1}になります。このマッピングが正常に行われたかどうかを確認するには、次のように述語を評価します。

  A.Price >= AVG (A.Price)

  左辺のA.Priceは、セットの最終行である単一行で、実行中セマンティクスを使用して評価する必要があります。セットの最後の行はR1であるため、A.Priceは10になります。

  右辺のAVG (A.Price)は、セットの行を使用して計算される集計です。この平均は10/1 = 10となります。

  述語が10 >= 10であるかと尋ねるので、答は「はい」となり、マッピングは成功します。ただし、パターンA+は強欲なので、問合せではできるかぎり多くの行を一致させようとする必要があります。

- 行パターン・パーティションの2番目の行では、R2をパターン変数Aに試験的にマップします。この時点では2つの行がAにマップされているため、セットは{R1, R2}となります。述語を評価して、マッピングが成功したかどうかを確認してください。

  A.Price >= AVG (A.Price)

  左辺のA.Priceは、セットの最終行である単一行で、実行中セマンティクスを使用して評価する必要があります。セットの最終行はR2であるため、A.Priceは16になります。右辺のAVG (A.Price)は、セットの行を使用して計算される集計です。この平均値は(10+16)/2 = 13です。したがって、述語は16 >= 13かどうかと尋ねてきます。答は「はい」になるため、マッピングは成功しています。

- 行パターン・パーティションの3番目の行では、R3をパターン変数Aに試験的にマップします。現在3つの行がAにマップされているため、セットは{R1, R2, R3}となります。述語を評価して、マッピングが成功したかどうかを確認してください。

  A.Price >= AVG (A.Price)

  左辺のA.PriceはR3で評価されます。したがって、A.Priceは13です。

  右辺のAVG (A.Price)は、セットの行を使用して計算される集計です。この平均値は(10+16+13)/3 = 13です。したがって、述語は13 >= 13かどうかと尋ねてきます。答は「はい」になるため、マッピングは成功しています。

- 行パターン・パーティションの4番目の行では、R4をパターン変数Aに試験的にマップします。この時点で、セットは{R1, R2, R3, R4}となります。述語を評価して、マッピングが成功したかどうかを確認してください。

  A.Price >= AVG (A.Price)

  左辺のA.PriceはR4で評価されます。したがって、A.Priceは9です。

  右辺のAVG (A.Price)は、セットの行を使用して計算される集計です。この平均値は(10+16+13+9)/4 = 12です。したがって、述語は9 >= 12かどうかと尋ねてきます。答えは「いいえ」となるため、マッピングは成功しません。

R4はAの定義を満たさなかったため、A+との最長一致は{R1, R2, R3}になります。A+は強欲な数量子を持っているため、この一致が優先されます。

DEFINE句で計算された平均値は、実行中の平均値です。MEASURESでは、特にALL ROWS PER MATCHが指定されている場合に、最終集計と実行中の集計を区別できます。MEASURES句にキーワードRUNNINGおよびFINALが使用されていることに留意し

てください。この区別は、表21-3の例の結果で確認できます。

表21-3 行パターンのナビゲーション

記号	タイムスタンプ	価格	実行中の平均値	最終平均値
XYZ	2009-06-09	10	10	13
XYZ	2009-06-10	16	13	13
XYZ	2009-06-11	13	13	13

パターン変数にマップされた行のセットが空の場合があります。空のセットで評価する場合:

- COUNTは0です。
- その他のすべての集計、行パターンのナビゲーション操作または通常のパターンの列参照はNULLになります。

次に例を示します。

```
PATTERN ( A? B+ )
DEFINE A AS A.Price > 100,
       B AS B.Price > COUNT (A.*) * 50
```

前の例を使用して、次の表21-4のデータの順序付き行パターン・パーティションを考えてみます。

表21-4 パターンおよび行

行	価格
R1	60
R2	70
R3	40

次のようにして、このデータで一致を見つけることができます。

- 行R1をパターン変数Aに試験的にマップします。(数量子?は、最初にAとの単独の一致を検索し、それが失敗した場合は、空の一致が一致するA?として取り込まれることを意味します)。マッピングが成功したかどうかを確認するために、述語A.Price > 100が評価されます。A.Priceが60であるため、述語はfalseとなり、Aとのマッピングは成功しません。
- Aとのマッピングが失敗したため、空の一致が一致するA?として取り込まれます。
- 行R1をBに試験的にマップします。このマッピングを確認するための述語は、B.Price > COUNT (A.*) * 50です。

  Aにマップされた行がないため、COUNT (A.*)は0になります。B.Price = 60は0よりも大きいため、マッピングは成功です。
- 同様に、行R2および R3も正常にB.にマップできます。さらに行が存在しないため、これが完全な一致になります。つまり、Aにマップされた行はなく、行{R1, R2, R3}がBにマップされています。

パターン変数は前方参照、つまり、まだ一致していないパターン変数の参照を行うことができます。次に例を示します。

```
PATTERN (X+ Y+)
DEFINE X AS COUNT (Y.*) > 3,
```

```
Y AS Y.Price > 10
```

この例は有効な構文です。ただし、1行がXにマップされた時点で、Yにマップされた行が存在していなかったため、この例では一致しません。したがって、COUNT(Y.*)は0で、3よりも小さくなります。将来、4つの行がYに正常にマップされることがあっても、これは変わりません。表21-5でこのデータセットを考えてみます。

表21-5 パターンおよび行

行	価格
R1	2
R2	11
R3	12
R4	13
R5	14

{R2, R3, R4, R5}のYへのマッピングは成功します。その理由は、この4つの行がすべてYに定義されているブール条件を満たすためです。その場合、R1をXにマップして、完全一致を成功させることができるとも考えられます。ただし、パターンX+ Y+によると、Yに行がマップされる前に少なくとも1行がXにマップされている必要があるため、パターン一致のルールではこの一致は見つかりません。

21.3.9.6.2 RUNNINGおよびFINALキーワードの比較

RUNNINGおよびFINALは、実行中または最終セマンティクスのいずれが要求されているかを指定するキーワードです。RUNNINGとFINALは、集計および、行パターンのナビゲーション操作FIRSTおよびLASTとともに使用できます。

集計FIRSTおよびLASTは、行パターン一致の問合せで次の場所に発生させることができます。

- DEFINE句の中。DEFINE句を処理している場合、問合せはまだ一致の認識中であるため、実行中セマンティクスのみがサポートされます。
- MEASURES句の中。MEASURES句を処理している場合、問合せは一致の認識を終了したため、最終セマンティクスを考慮できるようになります。次の2つのサブケースがあります。
  - ONE ROW PER MATCHが指定された場合、問合せは概念上、一致の最後の行に配置されるため、実行中セマンティクスと最終セマンティクスの相違は事実上なくなります。
  - ALL ROWS PER MATCHが指定された場合、行パターンの出力表には一致の行ごとに1行が表示されます。この状況では、ユーザーが実行中の値と最終の値の両方を確認したい場合があるため、パターン一致ではキーワードRUNNINGとFINALを用意して、その区別に対応しています。

この分析に基づいて、パターン一致は次のように指定します。

- MEASURESでは、キーワードRUNNINGおよびFINALを使用して、集計FIRSTまたはLASTに必要なセマンティクスを指定できます。キーワードは演算子の前に記述します。たとえば、RUNNING COUNT (A.*)またはFINAL SUM (B.Price)とします。
- MEASURESおよびDEFINEのいずれでも、デフォルトはRUNNINGです。

- DEFINEでは、FINALは使用できません。必要に応じて、RUNNINGを使用してより明確にすることもできます。
- MEASURESでONE ROW PER MATCHを指定した場合、すべての集計FIRSTおよびLASTは、一致の最後の行が認識された後で計算されるため、デフォルトのRUNNINGセマンティクスはFINALセマンティクスと事実上同じになります。ユーザーは、このような場合にFINALがデフォルトになる式を作成したり、FINALを記述して明確さを高めることもできます。
- 通常の列参照には、実行中セマンティクスが設定されています。(ALL ROWS PER MATCHの場合、MEASURESで最終セマンティクスを取得するには、通常の列参照ではなく、FINAL LAST行パターンのナビゲーション操作を使用してください。)

21.3.9.6.3 通常の行パターンの列参照

通常の行パターンの列参照とは、集計もナビゲーションも行われないものです。次に例を示します。

```
A.Price
```

RUNNINGおよびFINALキーワードの比較では、通常の行パターンの列参照には常に実行中セマンティクスが設定されていると説明しています。これは次のことを意味します。

- DEFINEでは、通常の列参照は、現在行も含めて現在行までで、パターン変数にマップされた最後の行を参照します。このような行がない場合、値はNULLになります。
- MEASURESには、次の2つのサブケースがあります。
  - ALL ROWS PER MATCHが指定された場合は、現在行も表記され、セマンティクスはDEFINEの場合と同じになります。
  - ONE ROW PER MATCHが指定された場合、問合せは概念上、一致の最後の行に配置されます。通常の列参照は、パターン変数にマップされた最後の行を参照します。変数がどの行にもマップしていない場合、値はNULLになります。

これらのセマンティクスはLAST演算子と同じであり、RUNNINGが暗黙的にデフォルトになります。したがって、X.Priceなどの通常の列参照はRUNNING LAST (X.Price)と同等になります。

21.3.10 行パターンの出力

MATCH_RECOGNIZEの結果を行パターンの出力表といいます。行パターンの出力表の形状(行タイプ)は、ONE ROW PER MATCHまたはALL ROWS PER MATCHの選択によって異なります。

ONE ROW PER MATCHが指定されたか示唆された場合、行パターンの出力表の列は宣言順に行パターンのパーティション化列になり、その後に行パターンのメジャー列が宣言順に続きます。表には少なくとも1つの列が必要なため、少なくとも行パターンのパーティション化列が1つ、または行パターンのメジャー列が1つ存在している必要があります。

ALL ROWS PER MATCHが指定された場合、行パターンの出力表の列として、宣言順による行パターンのパーティション化列、宣言順による順序付け列、宣言順による行パターンのメジャー列、そして最後に行パターンの入力表のその他の列が行パターンの入力表での発生順に表示されます。

パターンのメジャー列の名前と宣言されたタイプは、MEASURES句によって判別されます。メジャー以外の列の名前と宣言されたタイプは、パターンの入力表の対応する列から継承されます。

関連項目:

相関名を行パターン出力に割り当てる方法については相関名および行パターンの出力を参照してください。

21.3.10.1 相関名および行パターンの出力

行パターンの出力表に、次のような相関名を割り当てることができます。

```
SELECT M.Matchno
FROM Ticker MATCH_RECOGNIZE (...
     MEASURE MATCH_NUMBER() AS Matchno
     ...
) M
```

この例では、Mが行パターンの出力表に割り当てられた相関名です。相関名を割り当てる利点は、相関名を使用すると、この例のM.Matchnoのように、行パターンの出力表の列名を修飾できることです。FROM句に他の表がある場合、これはあいまいな列名を解決する際に特に重要です。

21.4 パターン一致の高度なトピック

この項では、次の高度なトピックについて説明します。

- パターン一致におけるPREVおよびNEXT内でのFIRSTとLASTのネスト
- パターン一致における空の一致または一致しない行の処理
- パターンの部分を出力から除外する方法
- すべての順列の表現方法

21.4.1 パターン一致のPREVおよびNEXT内のFIRSTとLASTのネスト

FIRSTおよびLASTは、特定のパターン変数にすでにマップしている行のセット内でのナビゲーションを提供します。PREVおよびNEXTは、特定の行の物理オフセットを使用するナビゲーションを提供します。これらの種類のナビゲーションは、PREVまたはNEXT内でFIRSTまたはLASTをネストすることによって、組み合せられます。これによって、次のような式が使用可能になります。

```
PREV (LAST (A.Price + A.Tax, 1), 3)
```

この例では、Aはパターン変数である必要があります。行パターンの列参照が設定されている必要があり、複合演算子のすべてのパターン変数が同等(この例ではA)である必要があります。

この複合演算子は、次のように評価されます。

1. 内部演算子LASTは、パターン変数Aにマップされた行のセットのみで処理を行います。このセットで、最後から*1*つ前の行を検索します。(該当する行がない場合、結果はNULLになります。)
2. 外部演算子PREVは、行パターン・パーティションで、ステップ1で見つかった行から3行前まで戻ります。(該当する行がない場合、結果はNULLになります。)
3. Rを、ステップ2で見つかった行を参照する実装依存の範囲変数にします。式A.Price + A.Taxで、パターン変数Aのすべての出現をRに置換します。この結果得られた式R.Price + R.Taxが評価され、複合ナビゲーション操作の値が決定します。

たとえば、表21-6のデータセットとマッピングを考えてみます。

表21-6 データセットとマッピング

行	価格	税金	マッピング
R1	10	1	

行	価格	税金	マッピング
R2	20	2	A
R3	30	3	B
R4	40	4	A
R5	50	5	C
R6	60	6	A

PREV (LAST (A.Price + A.Tax, 1), 3)を評価するには、次のステップを使用できます。

- Aにマップされた行のセットは{R2, R4, R6}となります。LASTはこのセットで処理を行い、終わりからオフセットして行R4に到達します。
- PREVはR4の3行前の物理オフセットを実行し、R1に到達します。
- Rを、R1を指し示す範囲変数にします。R.Price + R.Taxが評価されて、10+1 = 11となります。

このネストは、ネストした関数の一般的な評価として定義されていないことに留意してください。内部演算子LASTは、実際には式A.Price + A.Taxを評価しません。これは、この式を使用してパターン変数(A)を指定した後で、その変数にマップされた行内をナビゲートします。外部演算子PREVは、行に対してさらに物理ナビゲーションを実行します。式A.Price + A.Taxは実際にはほとんど評価されません。これは、最終的に到達される行がパターン変数Aにマップされるとはかぎないためです。この例では、R1はどのパターン変数にもマップされません。

21.4.2 パターン一致での空の一致または一致しない行の処理

ALL ROWS PER MATCHには、次の3つのサブオプションがあります。

- ALL ROWS PER MATCH SHOW EMPTY MATCHES
- ALL ROWS PER MATCH OMIT EMPTY MATCHES
- ALL ROWS PER MATCH WITH UNMATCHED ROWS

これらのオプションについては、次の内容で説明します。

- パターン一致における空の一致の処理
- パターン一致における一致しない行の処理

21.4.2.1 パターン一致での空の一致の処理

パターンの中には、空の一致が使用できるものがあります。たとえば、PATTERN (A*)は、Aにマップされた0行以上の行による一致が可能です。

空の一致では行とパターン変数とのマッピングは行われませんが、空の一致には開始行が存在します。たとえば、パーティションの1行目に空の一致が存在することも、2行目またはそれ以降に空の一致が存在することも考えられます。空の一致には、他のすべての一致と同じように、開始行の順序位置に基づいて順次一致番号が割り当てられます。

ONE ROW PER MATCHを使用した場合、空の一致によって出力表に1行が表示されます。空の一致に対する行パターンのメジャーは、次のように計算されます。

- MATCH_NUMBER()の値は、空の一致の順次一致番号です。
- どのCOUNTも0になります。
- その他のすべての集計、行パターンのナビゲーション操作または通常の行パターンの列参照はNULLになります。

ALL ROWS PER MATCHに関しては、空の一致には行が存在しないため、空の一致に対して出力行を生成するかどうかという問題が生じます。これに対応するには、次の2つのオプションがあります。

- ALL ROWS PER MATCH SHOW EMPTY MATCHES: このオプションを使用すると、空の一致では行パターンの出力表に1つの行が生成されます。
- ALL ROWS PER MATCH OMIT EMPTY MATCHES: このオプションを使用すると、空の一致は行パターンの出力表から省略されます。(このため、順次一致番号付けでギャップが生じる場合があります。)

ALL ROWS PER MATCHのデフォルトはSHOW EMPTY MATCHESです。このオプションを使用すると、空の一致では行パターンの出力行に1行が生成されます。この行では、次のようになります。

- CLASSIFIER()関数の値はNULLになります。
- MATCH_NUMBER ()関数の値は、空の一致の順次一致番号になります。
- 通常の行パターンの列参照の値は、NULLになります。
- 集計または行パターンのナビゲーション操作の値は、空の行セットを使って計算されます(したがって、どのCOUNTも0になり、他のすべての集計および行パターンのナビゲーション操作はNULLになります)。
- 行パターンの入力表の列に対応している列の値は、空の一致の開始行の対応する列と同じです。

21.4.2.2 パターン一致での一致しない行の処理

行パターンの入力表の行の中には、空の一致の開始行にもならず、空以外の一致によってマップされることもないものがあります。このような行を、一致しない行といいます。

オプションALL ROWS PER MATCH WITH UNMATCHED ROWSは、空の一致と一致しない行の両方を表示します。空の一致は、SHOW EMPTY MATCHESの場合と同様に処理されます。一致しない行を表示する場合、すべての行パターンのメジャーはNULLになります。これは、外部結合のNULL拡張側に類似しています。したがって、COUNTおよびMATCH_NUMBERを使用して、一致しない行と空の一致の開始行を区別できる場合があります。除外構文{- -}は、WITH UNMATCHED ROWSの意図に反して、使用が禁止されています。詳細は、パターンの部分を出力から除外する方法を参照してください。

パターンで空の一致を使用したり、一致しない行をパターンに含めることはできません。その理由は、行パターンの入力表の行をプライマリ行パターン変数にマップできない場合、その行は、パターンで空の一致が使用できることを前提とした場合に、空の一致の開始行のまま残り、一致しない行とは見なされないためです。そのため、パターンで空の一致が使用できる場合は、ALL ROWS PER MATCH SHOW EMPTY MATCHESを使用する出力はALL ROWS PER MATCH WITH UNMATCHED ROWSを使用する出力と同じになります。したがって、WITH UNMATCHED ROWSは主として、空の一致が使用できないパターンで使用されます。ただし、パターンに空の一致または一致しない行があるかどうか不確かな場合は、WITH UNMATCHED ROWSを指定する場合もあります。

ALL ROWS PER MATCH WITH UNMATCHED ROWSをデフォルトのスキップ動作(AFTER MATCH SKIP PAST LAST ROW)とともに使用する場合、入力の行ごとに1行ずつが出力に発生します。

その他のスキップ動作はWITH UNMATCHED ROWSを使用して実行できます。その場合、行は複数の一致とのマップが可能であり、行パターンの出力表に複数回表示できます。一致しない行は、出力に1回のみ表示されます。

21.4.3 パターンの部分を出力から除外する方法

ALL ROWS PER MATCHをOMIT EMPTY MATCHESまたはSHOW EMPTY MATCHESサブオプションのいずれかとともに使用する場合、

PATTERNに部分的に一致する行が行パターンの出力表から除外されることがあります。除外された部分は、PATTERN句の中で{-と-}で括られます。

たとえば、次の例では、価格が最低でも10から始まる価格上昇の最長期間を検出しています。

例21-8 価格上昇期間

```
SELECT M.Symbol, M.Tstamp, M.Matchno, M.Classfr, M.Price, M.Avgp
FROM Ticker MATCH_RECOGNIZE (
     PARTITION BY Symbol
     ORDER BY tstamp
     MEASURES FINAL AVG(S.Price) AS Avgp,
              CLASSIFIER() AS Classfr,
              MATCH_NUMBER() AS Matchno
     ALL ROWS PER MATCH
     AFTER MATCH SKIP TO LAST B
     PATTERN ( {- A -} B+ {- C+ -} )
     SUBSET S = (A,B)
     DEFINE
        A AS A.Price >= 10,
        B AS B.Price > PREV(B.Price),
        C AS C.Price <= PREV(C.Price)
)  M
ORDER BY symbol, tstamp;

SYMBOL     TSTAMP       MATCHNO CLAS      PRICE         AVGP
---------- --------- ---------- ---- ---------- ----------
ACME       02-APR-11          1 B            17         18.8
ACME       03-APR-11          1 B            19         18.8
ACME       04-APR-11          1 B            21         18.8
ACME       05-APR-11          1 B            25         18.8
ACME       07-APR-11          2 B            15         19.2
ACME       08-APR-11          2 B            20         19.2
ACME       09-APR-11          2 B            24         19.2
ACME       10-APR-11          2 B            25         19.2
ACME       13-APR-11          3 B            25           20
ACME       17-APR-11          4 B            14   16.6666667
ACME       18-APR-11          4 B            24   16.6666667
```

行パターンの出力表にはBにマップされた行のみが表示され、AおよびCにマップされた行は出力から除外されます。除外された行は行パターンの出力表に表示されませんが、共用体パターン変数の定義または、DEFINEまたはMEASURESのスカラー式の計算から除外されることはありません。たとえば、プライマリ・パターン変数AおよびCの定義、共用体パターン変数Sの定義、または前出の例のAvgp行パターンのメジャーを参照してください。

除外構文はALL ROWS PER MATCH WITH UNMATCHED ROWSでは使用できません。

除外構文はONE ROW PER MATCHでは使用できますが、この場合は一致ごとに1つのサマリー行しか存在しないため、効果はありません。

21.4.4 すべての順列の表現方法

PERMUTE構文を使用して、より単純なパターンの順列であるパターンを表現できます。たとえば、PATTERN (PERMUTE (A, B, C))は、次のように、3つのパターン変数A、BおよびCのすべての順列の代替と同じです。

```
PATTERN (A B C | A C B | B A C | B C A | C A B | C B A)
```

PERMUTEは辞書編集上で展開され、並べ替える各要素は他の要素からカンマで区切る必要があります。(この例では、3つのパターン変数A、BおよびCはアルファベット順にリストされているため、これは展開された順列もアルファベット順にリストされる辞書

編集式の展開に基づいています。)これは、展開に記述された順序で代替が試みられることを示しているため、重要です。したがって、 (A B C)との一致が(A C B)との一致の前に試みられるというように順次処理されます。最初に成功した試みを勝者と呼ぶことができます。

別の例を示します。

```
PATTERN (PERMUTE (X{3}, B C?, D))
```

これは、次のルールと同等です。

```
PATTERN ((X{3} B C? D)
| (X{3} D B C?)
| (B C? X{3} D)
| (B C? D X{3})
| (D X{3} B C?)
| (D B C? X{3}))
```

パターン要素B C?はカンマで区切られていないため、単一ユニットとして処理されます。

21.5 パターン一致のルールと制限

この項では、次のルールと制限について説明します。

- パターン一致における入力表の要件
- MATCH_RECOGNIZE句で禁止されたネスト
- 連結MATCH_RECOGNIZE句
- 集計の制限

21.5.1 パターン一致の入力表の要件

行パターンの入力表はMATCH_RECOGNIZEに対する入力引数です。表またはビュー、あるいは名前付き問合せ(WITH句で定義)を使用できます。行パターンの入力表は、導出表(インライン・ビューとしても知られています)にもできます。次に例を示します。

```
FROM (SELECT S.Name, T.Tstamp, T.Price
      FROM Ticker T, SymbolNames S
      WHERE T.Symbol = S.Symbol)
MATCH_RECOGNIZE (...) M
```

行パターンの入力表は結合表にはできません。回避策として、次のような導出表を使用できます。

```
FROM (SELECT * FROM A LEFT OUTER JOIN B ON (A.X = B.Y))
MATCH_RECOGNIZE (...) M
```

パターン入力表の列名は明確である必要があります。SQLではベース表またはビューにあいまいな列名を使用できないため、行パターンの入力表がベース表またはビューの場合は問題ありません。行パターンの入力表が導出表の場合のみ、問題が生じます。たとえば、EmpおよびDeptという2つの表の結合があり、それぞれの表にNameという列があるとします。次は、構文エラーです。

```
FROM (SELECT D.Name, E.Name, E.Empno, E.Salary
      FROM Dept D, Emp E
      WHERE D.Deptno = E.Deptno)
MATCH_RECOGNIZE (
      PARTITION BY D.Name
   ...)
```

この例は、変数DがMATCH_RECOGNIZE句の中に表示されていないため、エラーになります(Dの有効範囲は導出表のみです)。

次のように書き直しても効果的ではありません。

```
FROM (SELECT D.Name, E.Name, E.Empno, E.Salary
      FROM Dept D, Emp E
      WHERE D.Deptno = E.Deptno)
MATCH_RECOGNIZE (
      PARTITION BY Name
   ...)
```

この書き直しによって、MATCH_RECOGNIZE句での変数Dの使用がなくなります。ただし、今回は導出表にNameという列が2つ存在するため、Nameがあいまいになるというエラーが生じます。この対処方法は、次に示すように、導出表自体の中の列名を明確にすることです。

```
FROM (SELECT D.Name AS Dname, E.Name AS Ename,
         E.Empno, E.Salary
      FROM Dept D, Emp E
      WHERE D.Deptno = E.Deptno)
MATCH_RECOGNIZE (
      PARTITION BY Dname
   ...)
```

関連項目:

『Oracle Database SQL言語リファレンス』

21.5.2 MATCH_RECOGNIZE句で禁止されたネスト

次の種類のネストは、MATCH_RECOGNIZE句では禁止されています。

- 1つのMATCH_RECOGNIZE句を別のこの句の中にネストすること。
- MEASURES句またはDEFINE副次句での外部参照。つまり、MATCH_RECOGNIZE句は、行パターンの入力表を除き、外部問合せブロック内の表を参照できません。
- 相関副問合せは、MEASURESまたはDEFINEでは使用できません。また、MEASURESまたはDEFINEの副問合せはパターン変数を参照できません。
- MATCH_RECOGNIZE句は、再帰的問合せで使用できません。
- SELECT FOR UPDATE文では、MATCH_RECOGNIZE句を使用できません。

21.5.3 連結MATCH_RECOGNIZE句

次の例のように、1つのMATCH_RECOGNIZE句の出力を別のこの句の入力に投入することは可能です。

```
SELECT ...
FROM ( SELECT *
       FROM Ticker
            MATCH_RECOGNIZE (...) )
       MATCH_RECOGNIZE (...)
```

この例では、最初のMATCH_RECOGNIZE句が導出表にあり、これが2番目のMATCH_RECOGNIZEに入力を提供します。

21.5.4 集計の制限

集計関数COUNT、SUM、AVG、MAXおよびMINは、MEASURES句とDEFINE句の両方で使用できます。DISTINCTキーワードはサポートされていません。

21.6 パターン一致の例

この項では、次のタイプの高度なパターン一致の例について説明します。

- パターン一致の例: 株式市場
- パターン一致の例: セキュリティ・ログの分析
- パターン一致の例: セッション化
- パターン一致の例: 会計トラッキング

21.6.1 パターン一致の例: 株式市場

この項では、株価とパターンに関連した共通タスクに基づくパターン一致の例について説明します。

例21-9 指定規模の株価下落

例21-9の問合せでは、現在の株価が前日の終値を特定の率(この例では8%)以上下回っている株式を示しています。

```
CREATE TABLE Ticker3Wave (SYMBOL VARCHAR2(10), tstamp DATE, PRICE NUMBER);

INSERT INTO Ticker3Wave VALUES('ACME', '01-Apr-11', 1000);
INSERT INTO Ticker3Wave VALUES('ACME', '02-Apr-11', 775);
INSERT INTO Ticker3Wave VALUES('ACME', '03-Apr-11', 900);
INSERT INTO Ticker3Wave VALUES('ACME', '04-Apr-11', 775);
INSERT INTO Ticker3Wave VALUES('ACME', '05-Apr-11', 900);
INSERT INTO Ticker3Wave VALUES('ACME', '06-Apr-11', 775);
INSERT INTO Ticker3Wave VALUES('ACME', '07-Apr-11', 900);
INSERT INTO Ticker3Wave VALUES('ACME', '08-Apr-11', 775);
INSERT INTO Ticker3Wave VALUES('ACME', '09-Apr-11', 800);
INSERT INTO Ticker3Wave VALUES('ACME', '10-Apr-11', 550);
INSERT INTO Ticker3Wave VALUES('ACME', '11-Apr-11', 900);
INSERT INTO Ticker3Wave VALUES('ACME', '12-Apr-11', 800);
INSERT INTO Ticker3Wave VALUES('ACME', '13-Apr-11', 1100);
INSERT INTO Ticker3Wave VALUES('ACME', '14-Apr-11', 800);
INSERT INTO Ticker3Wave VALUES('ACME', '15-Apr-11', 550);
INSERT INTO Ticker3Wave VALUES('ACME', '16-Apr-11', 800);
INSERT INTO Ticker3Wave VALUES('ACME', '17-Apr-11', 875);
INSERT INTO Ticker3Wave VALUES('ACME', '18-Apr-11', 950);
INSERT INTO Ticker3Wave VALUES('ACME', '19-Apr-11', 600);
INSERT INTO Ticker3Wave VALUES('ACME', '20-Apr-11', 300);

SELECT *
FROM Ticker3Wave MATCH_RECOGNIZE (
     PARTITION BY symbol
     ORDER BY tstamp
     MEASURES B.tstamp AS timestamp,
              A.price AS Aprice,
              B.price AS Bprice,
              ((B.price - A.price)*100) / A.price AS PctDrop
    ONE ROW PER MATCH
    AFTER MATCH SKIP TO B
    PATTERN (A B)
    DEFINE
```

```
       B AS (B.price - A.price) / A.price  < -0.08
   );

SYMBOL  TIMESTAMP    APRICE     BPRICE    PCTDROP
------  ---------  ---------- -------   ----------
ACME    02-APR-11  1000           775        -22.5
ACME    04-APR-11   900           775   -13.888889
ACME    06-APR-11   900           775   -13.888889
ACME    08-APR-11   900           775   -13.888889
ACME    10-APR-11   800           550   -31.25
ACME    12-APR-11   900           800   -11.111111
ACME    14-APR-11  1100           800   -27.272727
ACME    15-APR-11   800           550   -31.25
ACME    19-APR-11   950           600   -36.842105
ACME    20-APR-11   600           300   -50.0

10 rows selected.
```

例21-10 元値に戻った時点での指定規模の価格下落

例21-10の問合せは、例21-9で定義したパターンを拡張したものです。8%を超える価格下落を示す株式を見つけます。また、株価が元値を下回ったままであるゼロ以上の追加日数をシークします。そして、株価が初期値と等しいか、それを超えるまで上昇した時点が特定されます。このパターンが発生する日数を知っておくと役立つため、これがここに含められています。start_price列は一致の開始値で、end_price列は開始値以上の株価になったときの一致の終値です。

```
SELECT *
 FROM Ticker3Wave MATCH_RECOGNIZE (
   PARTITION BY symbol
   ORDER BY tstamp
   MEASURES
      A.tstamp      as start_timestamp,
      A.price       as start_price,
      B.price       as drop_price,
      COUNT(C.*)+1  as cnt_days,
      D.tstamp      as end_timestamp,
      D.price       as end_price
   ONE ROW PER MATCH
   AFTER MATCH SKIP PAST LAST ROW
   PATTERN (A B C* D)
   DEFINE
      B as (B.price - A.price)/A.price < -0.08,
      C as C.price < A.price,
      D as D.price >= A.price
   );

SYMBOL     START_TIM START_PRICE DROP_PRICE  CNT_DAYS  END_TIMES  END_PRICE
---------- --------- ----------- ----------  --------  --------- ----------
ACME       01-APR-11        1000         775        11  13-APR-11        1100
ACME       14-APR-11         800         550         1  16-APR-11         800
```

例21-11 取引履歴でV字形とU字形の両方を検索する

例21-11では、パターンを定義する際に、考えられるすべてのデータ動作を考慮にいれることの重要性を示しています。表TickerVUは、最初の例の表Tickerとほとんど同じですが、3つ目の底部の行に2日間の等価日、April 16および17がある点が異なります。このように底部がフラットな価格下落をU字形といいます。元の例、例21-1では、変更されたデータがV字形に似たロットであることを認識し、その出力にU字形を含めることができるでしょうか。いいえ、できません。問合せを次のように変更する必要があります。

```
CREATE TABLE TickerVU (SYMBOL VARCHAR2(10), tstamp DATE, PRICE NUMBER);

INSERT INTO TickerVU values('ACME', '01-Apr-11', 12);
INSERT INTO TickerVU values('ACME', '02-Apr-11', 17);
INSERT INTO TickerVU values('ACME', '03-Apr-11', 19);
INSERT INTO TickerVU values('ACME', '04-Apr-11', 21);
INSERT INTO TickerVU values('ACME', '05-Apr-11', 25);
INSERT INTO TickerVU values('ACME', '06-Apr-11', 12);
INSERT INTO TickerVU values('ACME', '07-Apr-11', 15);
INSERT INTO TickerVU values('ACME', '08-Apr-11', 20);
INSERT INTO TickerVU values('ACME', '09-Apr-11', 24);
INSERT INTO TickerVU values('ACME', '10-Apr-11', 25);
INSERT INTO TickerVU values('ACME', '11-Apr-11', 19);
INSERT INTO TickerVU values('ACME', '12-Apr-11', 15);
INSERT INTO TickerVU values('ACME', '13-Apr-11', 25);
INSERT INTO TickerVU values('ACME', '14-Apr-11', 25);
INSERT INTO TickerVU values('ACME', '15-Apr-11', 14);
INSERT INTO TickerVU values('ACME', '16-Apr-11', 12);
INSERT INTO TickerVU values('ACME', '17-Apr-11', 12);
INSERT INTO TickerVU values('ACME', '18-Apr-11', 24);
INSERT INTO TickerVU values('ACME', '19-Apr-11', 23);
INSERT INTO TickerVU values('ACME', '20-Apr-11', 22);
```

例21-1の元の問合せを、この表名を使用するように変更して実行した場合、どうなるでしょうか。

```
SELECT *
FROM TickerVU MATCH_RECOGNIZE (
     PARTITION BY symbol
     ORDER BY tstamp
     MEASURES STRT.tstamp AS start_tstamp,
              DOWN.tstamp AS bottom_tstamp,
              UP.tstamp AS end_tstamp
    ONE ROW PER MATCH
    AFTER MATCH SKIP TO LAST UP
    PATTERN (STRT DOWN+ UP+)
    DEFINE DOWN AS DOWN.price < PREV(DOWN.price),
           UP AS UP.price > PREV(UP.price)
) MR
ORDER BY MR.symbol, MR.start_tstamp;

SYMBOL     START_TST BOTTOM_TS END_TSTAM
---------- --------- --------- ---------
ACME       05-APR-11 06-APR-11 10-APR-11
ACME       10-APR-11 12-APR-11 13-APR-11
```

この問合せは、出力の3つの行(価格下落ごとに1つ)を表示するかわりに、2つの行のみを表示します。これは、価格下落の底部でのフラットなデータの広がりを処理する変数が定義されていないためです。ここで、DEFINE句にフラットなデータを処理する変数を追加し、その変数をPATTERN句に使用して変更した問合せを使用してみます。

```
SELECT *
FROM TickerVU MATCH_RECOGNIZE (
     PARTITION BY symbol
     ORDER BY tstamp
     MEASURES STRT.tstamp AS start_tstamp,
              DOWN.tstamp AS bottom_tstamp,
              UP.tstamp AS end_tstamp
     ONE ROW PER MATCH
     AFTER MATCH SKIP TO LAST UP
     PATTERN (STRT DOWN+ FLAT* UP+)
```

```
     DEFINE
         DOWN AS DOWN.price < PREV(DOWN.price),
         FLAT AS FLAT.price = PREV(FLAT.price),
         UP AS UP.price > PREV(UP.price)
) MR
ORDER BY MR.symbol, MR.start_tstamp;

SYMBOL     START_TST   BOTTOM_TS   END_TSTAM
---------- ---------   ---------   ---------
ACME       05-APR-11   06-APR-11   10-APR-11
ACME       10-APR-11   12-APR-11   13-APR-11
ACME       14-APR-11   16-APR-11   18-APR-11
```

3つの価格下落がすべてデータに含まれている出力が得られます。ここで学んだことは、データ・シーケンスで可能なバリエーションをすべて考慮し、それらの可能性を必要に応じてPATTERN句、DEFINE句およびMEASURES句に含めることです。

表21-12 エリオット波動パターンの検索: 逆V字形の複数のインスタンス

例21-12は、逆V字形の複数の連続パターンを持つエリオット波動という、単純な株価パターンのクラスを示しています。この特別なケースにおいて、パターン式は1日以上の上昇の後に1日以上の下落のパターンを検索します。このシーケンスは5回連続して、途切れずに現れる必要があります。つまり、このパターンは/¥/¥/¥/¥/¥のようになります。

```
SELECT MR_ELLIOTT.*
FROM Ticker3Wave MATCH_RECOGNIZE (
     PARTITION BY symbol
     ORDER BY tstamp
     MEASURES
               COUNT(*) as CNT,
               COUNT(P.*) AS CNT_P,
               COUNT(Q.*) AS CNT_Q,
               COUNT(R.*) AS CNT_R,
               COUNT(S.*) AS CNT_S,
               COUNT(T.*) AS CNT_T,
               COUNT(U.*) AS CNT_U,
               COUNT(V.*) AS CNT_V,
               COUNT(W.*) AS CNT_W,
               COUNT(X.*) AS CNT_X,
               COUNT(Y.*) AS CNT_Y,
               COUNT(Z.*) AS CNT_Z,
               CLASSIFIER() AS CLS,
     MATCH_NUMBER() AS MNO
     ALL ROWS PER MATCH
     AFTER MATCH SKIP TO LAST Z
     PATTERN (P Q+ R+ S+ T+ U+ V+ W+ X+ Y+ Z+)
     DEFINE
        Q AS Q.price > PREV(Q.price),
        R AS R.price < PREV(R.price),
        S AS S.price > PREV(S.price),
        T AS T.price < PREV(T.price),
        U AS U.price > PREV(U.price),
        V AS V.price < PREV(V.price),
        W AS W.price > PREV(W.price),
        X AS X.price < PREV(X.price),
        Y AS Y.price > PREV(Y.price),
        Z AS Z.price < PREV(Z.price)
   ) MR_ELLIOTT
ORDER BY symbol, tstamp;

SYMB TSTAMP    CNT  CNT_P CNT_Q CNT_R CNT_S CNT_T CNT_U CNT_V CNT_W CNT_X CNT_Y CNT_Z CLS MNO PRICE
```

```
---- --------- ---- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- --- --- -----
ACME 02-APR-11    1     1     0     0     0     0     0     0     0     0     0     0 P     1   775
ACME 03-APR-11    2     1     1     0     0     0     0     0     0     0     0     0 Q     1   900
ACME 04-APR-11    3     1     1     1     0     0     0     0     0     0     0     0 R     1   775
ACME 05-APR-11    4     1     1     1     1     0     0     0     0     0     0     0 S     1   900
ACME 06-APR-11    5     1     1     1     1     1     0     0     0     0     0     0 T     1   775
ACME 07-APR-11    6     1     1     1     1     1     1     0     0     0     0     0 U     1   900
ACME 08-APR-11    7     1     1     1     1     1     1     1     0     0     0     0 V     1   775
ACME 09-APR-11    8     1     1     1     1     1     1     1     1     0     0     0 W     1   800
ACME 10-APR-11    9     1     1     1     1     1     1     1     1     1     0     0 X     1   550
ACME 11-APR-11   10     1     1     1     1     1     1     1     1     1     1     0 Y     1   900
ACME 12-APR-11   11     1     1     1     1     1     1     1     1     1     1     1 Z     1   800

11 rows selected.
```

例21-13 エリオット波動の検索と受入れ可能な行カウントの範囲の指定

例21-12と同様、例21-13でも逆V字形のエリオット波形が指定されますが、この場合は正規表現を使用して、一致させる連続行数をパターン変数ごとに指定します。これは、範囲として指定されます。構文"{3,4}"を使用して、3つまたは4つの連続一致をシークするように各パターン変数を設定します。出力にはパターンの1つの完全一致に対してすべての行が表示され、各パターン変数の始まりと終わりの正確なタイミングが表示されます。変数WおよびXではそれぞれ4行が一致しているのに対し、変数YおよびZではそれぞれ3行しか一致していない点に留意してください。

```
CREATE TABLE tickerwavemulti (symbol VARCHAR2(10), tstamp DATE, price NUMBER);

INSERT INTO tickerwavemulti VALUES('ACME', '01-May-10', 36.25 );
INSERT INTO tickerwavemulti VALUES('BLUE', '01-May-10', 177.85);
INSERT INTO tickerwavemulti VALUES('EDGY', '01-May-10', 27.18);
INSERT INTO tickerwavemulti VALUES('ACME', '02-May-10', 36.47);
INSERT INTO tickerwavemulti VALUES('BLUE', '02-May-10', 177.25);
INSERT INTO tickerwavemulti VALUES('EDGY', '02-May-10', 27.41);
INSERT INTO tickerwavemulti VALUES('ACME', '03-May-10', 36.36);
INSERT INTO tickerwavemulti VALUES('BLUE', '03-May-10', 176.16);
INSERT INTO tickerwavemulti VALUES('EDGY', '03-May-10', 27.43);
INSERT INTO tickerwavemulti VALUES('ACME', '04-May-10', 36.25);
INSERT INTO tickerwavemulti VALUES('BLUE', '04-May-10', 176.28);
INSERT INTO tickerwavemulti VALUES('EDGY', '04-May-10', 27.56);
INSERT INTO tickerwavemulti VALUES('ACME', '05-May-10', 36.36);
INSERT INTO tickerwavemulti VALUES('BLUE', '05-May-10', 177.72);
INSERT INTO tickerwavemulti VALUES('EDGY', '05-May-10', 27.31);
INSERT INTO tickerwavemulti VALUES('ACME', '06-May-10', 36.70);
INSERT INTO tickerwavemulti VALUES('BLUE', '06-May-10', 178.36);
INSERT INTO tickerwavemulti VALUES('EDGY', '06-May-10', 27.23);
INSERT INTO tickerwavemulti VALUES('ACME', '07-May-10', 36.50);
INSERT INTO tickerwavemulti VALUES('BLUE', '07-May-10', 178.93);
INSERT INTO tickerwavemulti VALUES('EDGY', '07-May-10', 27.08);
INSERT INTO tickerwavemulti VALUES('ACME', '08-May-10', 36.66);
INSERT INTO tickerwavemulti VALUES('BLUE', '08-May-10', 178.18);
INSERT INTO tickerwavemulti VALUES('EDGY', '08-May-10', 26.90);
INSERT INTO tickerwavemulti VALUES('ACME', '09-May-10', 36.98);
INSERT INTO tickerwavemulti VALUES('BLUE', '09-May-10', 179.15);
INSERT INTO tickerwavemulti VALUES('EDGY', '09-May-10', 26.73);
INSERT INTO tickerwavemulti VALUES('ACME', '10-May-10', 37.08);
INSERT INTO tickerwavemulti VALUES('BLUE', '10-May-10', 180.39);
INSERT INTO tickerwavemulti VALUES('EDGY', '10-May-10', 26.86);
INSERT INTO tickerwavemulti VALUES('ACME', '11-May-10', 37.43);
INSERT INTO tickerwavemulti VALUES('BLUE', '11-May-10', 181.44);
INSERT INTO tickerwavemulti VALUES('EDGY', '11-May-10', 26.78);
INSERT INTO tickerwavemulti VALUES('ACME', '12-May-10', 37.68);
```

```
INSERT INTO tickerwavemulti VALUES('BLUE', '12-May-10', 183.11);
INSERT INTO tickerwavemulti VALUES('EDGY', '12-May-10', 26.59);
INSERT INTO tickerwavemulti VALUES('ACME', '13-May-10', 37.66);
INSERT INTO tickerwavemulti VALUES('BLUE', '13-May-10', 181.50);
INSERT INTO tickerwavemulti VALUES('EDGY', '13-May-10', 26.39);
INSERT INTO tickerwavemulti VALUES('ACME', '14-May-10', 37.32);
INSERT INTO tickerwavemulti VALUES('BLUE', '14-May-10', 180.65);
INSERT INTO tickerwavemulti VALUES('EDGY', '14-May-10', 26.31);
INSERT INTO tickerwavemulti VALUES('ACME', '15-May-10', 37.16);
INSERT INTO tickerwavemulti VALUES('BLUE', '15-May-10', 179.51);
INSERT INTO tickerwavemulti VALUES('EDGY', '15-May-10', 26.53);
INSERT INTO tickerwavemulti VALUES('ACME', '16-May-10', 36.98);
INSERT INTO tickerwavemulti VALUES('BLUE', '16-May-10', 180.00);
INSERT INTO tickerwavemulti VALUES('EDGY', '16-May-10', 26.76);
INSERT INTO tickerwavemulti VALUES('ACME', '17-May-10', 37.19);
INSERT INTO tickerwavemulti VALUES('BLUE', '17-May-10', 179.24);
INSERT INTO tickerwavemulti VALUES('EDGY', '17-May-10', 26.63);
INSERT INTO tickerwavemulti VALUES('ACME', '18-May-10', 37.45);
INSERT INTO tickerwavemulti VALUES('BLUE', '18-May-10', 180.48);
INSERT INTO tickerwavemulti VALUES('EDGY', '18-May-10', 26.84);
INSERT INTO tickerwavemulti VALUES('ACME', '19-May-10', 37.79);
INSERT INTO tickerwavemulti VALUES('BLUE', '19-May-10', 181.21);
INSERT INTO tickerwavemulti VALUES('EDGY', '19-May-10', 26.90);
INSERT INTO tickerwavemulti VALUES('ACME', '20-May-10', 37.49);
INSERT INTO tickerwavemulti VALUES('BLUE', '20-May-10', 179.79);
INSERT INTO tickerwavemulti VALUES('EDGY', '20-May-10', 27.06);
INSERT INTO tickerwavemulti VALUES('ACME', '21-May-10', 37.30);
INSERT INTO tickerwavemulti VALUES('BLUE', '21-May-10', 181.19);
INSERT INTO tickerwavemulti VALUES('EDGY', '21-May-10', 27.17);
INSERT INTO tickerwavemulti VALUES('ACME', '22-May-10', 37.08);
INSERT INTO tickerwavemulti VALUES('BLUE', '22-May-10', 179.88);
INSERT INTO tickerwavemulti VALUES('EDGY', '22-May-10', 26.95);
INSERT INTO tickerwavemulti VALUES('ACME', '23-May-10', 37.34);
INSERT INTO tickerwavemulti VALUES('BLUE', '23-May-10', 181.21);
INSERT INTO tickerwavemulti VALUES('EDGY', '23-May-10', 26.71);
INSERT INTO tickerwavemulti VALUES('ACME', '24-May-10', 37.54);
INSERT INTO tickerwavemulti VALUES('BLUE', '24-May-10', 181.94);
INSERT INTO tickerwavemulti VALUES('EDGY', '24-May-10', 26.96);
INSERT INTO tickerwavemulti VALUES('ACME', '25-May-10', 37.69);
INSERT INTO tickerwavemulti VALUES('BLUE', '25-May-10', 180.88);
INSERT INTO tickerwavemulti VALUES('EDGY', '25-May-10', 26.72);
INSERT INTO tickerwavemulti VALUES('ACME', '26-May-10', 37.60);
INSERT INTO tickerwavemulti VALUES('BLUE', '26-May-10', 180.72);
INSERT INTO tickerwavemulti VALUES('EDGY', '26-May-10', 26.47);
INSERT INTO tickerwavemulti VALUES('ACME', '27-May-10', 37.93);
INSERT INTO tickerwavemulti VALUES('BLUE', '27-May-10', 181.54);
INSERT INTO tickerwavemulti VALUES('EDGY', '27-May-10', 26.73);
INSERT INTO tickerwavemulti VALUES('ACME', '28-May-10', 38.17);
INSERT INTO tickerwavemulti VALUES('BLUE', '28-May-10', 182.93);
INSERT INTO tickerwavemulti VALUES('EDGY', '28-May-10', 26.89);

SELECT MR_EW.*
FROM tickerwavemulti MATCH_RECOGNIZE (
     PARTITION by symbol
     ORDER by tstamp
     MEASURES V.tstamp AS START_T,
              Z.tstamp AS END_T,
              COUNT(V.price) AS CNT_V,
              COUNT(W.price) AS UP__W,
              COUNT(X.price) AS DWN_X,
```

```
              COUNT(Y.price) AS UP__Y,
              COUNT(Z.price) AS DWN_Z,
    MATCH_NUMBER() AS MNO
    ALL ROWS PER MATCH
    AFTER MATCH SKIP TO LAST Z
    PATTERN (V W{3,4} X{3,4} Y{3,4} Z{3,4})
    DEFINE
       W AS W.price > PREV(W.price),
       X AS X.price < PREV(X.price),
       Y AS Y.price > PREV(Y.price),
       Z AS Z.price < PREV(Z.price)
) MR_EW
ORDER BY symbol, tstamp;

SYMB TSTAMP    START_T   END_T     CNT_V UP__W DWN_X UP__Y DWN_Z   MNO   PRICE
---- --------- --------- --------- ----- ----- ----- ----- ----- ----- -------
ACME 08-MAY-10 08-MAY-10                1     0     0     0     0     1   36.66
ACME 09-MAY-10 08-MAY-10                1     1     0     0     0     1   36.98
ACME 10-MAY-10 08-MAY-10                1     2     0     0     0     1   37.08
ACME 11-MAY-10 08-MAY-10                1     3     0     0     0     1   37.43
ACME 12-MAY-10 08-MAY-10                1     4     0     0     0     1   37.68
ACME 13-MAY-10 08-MAY-10                1     4     1     0     0     1   37.66
ACME 14-MAY-10 08-MAY-10                1     4     2     0     0     1   37.32
ACME 15-MAY-10 08-MAY-10                1     4     3     0     0     1   37.16
ACME 16-MAY-10 08-MAY-10                1     4     4     0     0     1   36.98
ACME 17-MAY-10 08-MAY-10                1     4     4     1     0     1   37.19
ACME 18-MAY-10 08-MAY-10                1     4     4     2     0     1   37.45
ACME 19-MAY-10 08-MAY-10                1     4     4     3     0     1   37.79
ACME 20-MAY-10 08-MAY-10 20-MAY-10      1     4     4     3     1     1   37.49
ACME 21-MAY-10 08-MAY-10 21-MAY-10      1     4     4     3     2     1   37.30
ACME 22-MAY-10 08-MAY-10 22-MAY-10      1     4     4     3     3     1   37.08

15 rows selected.
```

例21-14 一致の中間にスキップして重複する一致を確認する

例21-14では、AFTER MATCH SKIP TO句の重複している一致の検索能力を強調しています。これが使用するパターンは、パターン変数Q、R、SおよびTで構成されるW字形をシークするという単純なものです。W字の各辺に対し、行は1行または複数行にできます。この一致では、AFTER MATCH SKIP TO句も活用します。一致が見つかると、W字形の中間点である最後のR値のみにスキップします。これによって、問合せは、W字形の後半部分が次に重なるW字形の前半部分となる一致を見つけることができます。次の出力では、一致1がApril 5に終わっていても、一致2は重複していてApril 3に始まっていることがわかります。

```
SELECT MR_W.*
FROM Ticker3Wave MATCH_RECOGNIZE (
     PARTITION BY symbol
     ORDER BY tstamp
     MEASURES
        MATCH_NUMBER() AS MNO,
        P.tstamp AS START_T,
        T.tstamp AS END_T,
        MAX(P.price) AS TOP_L,
        MIN(Q.price) AS BOTT1,
        MAX(R.price) AS TOP_M,
        MIN(S.price) AS BOTT2,
        MAX(T.price) AS TOP_R
     ALL ROWS PER MATCH
     AFTER MATCH SKIP TO LAST R
     PATTERN ( P Q+ R+ S+ T+ )
     DEFINE
```

```
        Q AS Q.price < PREV(Q.price),
        R AS R.price > PREV(R.price),
        S AS S.price < PREV(S.price),
        T AS T.price > PREV(T.price)
) MR_W
ORDER BY symbol, mno, tstamp;

SYMB TSTAMP      MNO START_T   END_T     TOP_L BOTT1 TOP_M BOTT2 TOP_R PRICE
---- --------- ----- --------- --------- ----- ----- ----- ----- ----- -----
ACME 01-APR-11     1 01-APR-11            1000                       1000
ACME 02-APR-11     1 01-APR-11            1000   775                  775
ACME 03-APR-11     1 01-APR-11            1000   775   900            900
ACME 04-APR-11     1 01-APR-11            1000   775   900   775      775
ACME 05-APR-11     1 01-APR-11 05-APR-11  1000   775   900   775   900   900
ACME 03-APR-11     2 03-APR-11             900                       900
ACME 04-APR-11     2 03-APR-11             900   775                 775
ACME 05-APR-11     2 03-APR-11             900   775   900           900
ACME 06-APR-11     2 03-APR-11             900   775   900   775     775
ACME 07-APR-11     2 03-APR-11 07-APR-11   900   775   900   775   900   900
ACME 05-APR-11     3 05-APR-11             900                       900
ACME 06-APR-11     3 05-APR-11             900   775                 775
ACME 07-APR-11     3 05-APR-11             900   775   900           900
ACME 08-APR-11     3 05-APR-11             900   775   900   775     775
ACME 09-APR-11     3 05-APR-11 09-APR-11   900   775   900   775   800   800
ACME 07-APR-11     4 07-APR-11             900                       900
ACME 08-APR-11     4 07-APR-11             900   775                 775
ACME 09-APR-11     4 07-APR-11             900   775   800           800
ACME 10-APR-11     4 07-APR-11             900   775   800   550     550
ACME 11-APR-11     4 07-APR-11 11-APR-11   900   775   800   550   900   900
ACME 09-APR-11     5 09-APR-11             800                       800
ACME 10-APR-11     5 09-APR-11             800   550                 550
ACME 11-APR-11     5 09-APR-11             800   550   900           900
ACME 12-APR-11     5 09-APR-11             800   550   900   800     800
ACME 13-APR-11     5 09-APR-11 13-APR-11   800   550   900   800  1100  1100
ACME 11-APR-11     6 11-APR-11             900                       900
ACME 12-APR-11     6 11-APR-11             900   800                 800
ACME 13-APR-11     6 11-APR-11             900   800  1100           1100
ACME 14-APR-11     6 11-APR-11             900   800  1100   800     800
ACME 15-APR-11     6 11-APR-11             900   800  1100   550     550
ACME 16-APR-11     6 11-APR-11 16-APR-11   900   800  1100   550   800   800
ACME 17-APR-11     6 11-APR-11 17-APR-11   900   800  1100   550   875   875
ACME 18-APR-11     6 11-APR-11 18-APR-11   900   800  1100   550   950   950

33 rows selected.
```

例21-15 指定した時間間隔内で発生する大量取引を検索する

例21-15では、取引の多い株式、つまり、連結期間内で大量のトランザクションがあった株式を見つけます。この例で、大量の取引は、1時間以内に3つのトランザクションが発生し、各トランザクションでは30,000を超える株が取引されると定義されています。パターンが条件を満たさない取引を受け入れられるように、Bなどのパターン変数を含めることが重要です。B変数がないと、パターンでは、条件を満たしているトランザクションが3回連続して発生したケースのみが一致します。

この例の問合せは、表stockT04を使用しています。

```
CREATE TABLE STOCKT04 (symbol varchar2(10), tstamp TIMESTAMP,
                       price NUMBER, volume NUMBER);

INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 12.00.00.000000 PM', 35, 35000);
INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 12.05.00.000000 PM', 35, 15000);
INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 12.10.00.000000 PM', 35,  5000);
```

```
INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 12.11.00.000000 PM', 35, 42000);
INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 12.16.00.000000 PM', 35,  7000);
INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 12.19.00.000000 PM', 35,  5000);
INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 12.20.00.000000 PM', 35,  5000);
INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 12.33.00.000000 PM', 35, 55000);
INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 12.36.00.000000 PM', 35, 15000);
INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 12.48.00.000000 PM', 35, 15000);
INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 12.59.00.000000 PM', 35, 15000);
INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 01.09.00.000000 PM', 35, 55000);
INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 01.19.00.000000 PM', 35, 55000);
INSERT INTO STOCKT04 VALUES('ACME', '01-Jan-10 01.29.00.000000 PM', 35, 15000);

SELECT *
FROM stockT04 MATCH_RECOGNIZE (
     PARTITION BY symbol
     ORDER BY tstamp
     MEASURES FIRST (A.tstamp) AS in_hour_of_trade,
              SUM (A.volume) AS sum_of_large_volumes
     ONE ROW PER MATCH
     AFTER MATCH SKIP PAST LAST ROW
     PATTERN (A B* A B* A)
     DEFINE
        A AS ((A.volume > 30000) AND
        ((A.tstamp - FIRST (A.tstamp)) < '0    01:00:00.00' )),
        B AS ((B.volume <= 30000) AND ((B.tstamp - FIRST (A.tstamp)) < '0
        01:00:00.00'))
);

SYMBOL IN_HOUR_OF_TRADE                SUM_OF_LARGE_VOLUMES
------ ------------------------------  --------------------
ACME   01-JAN-10 12.00.00.000000 PM                  132000

1 row selected.
```

21.6.2 パターン一致の例: セキュリティ・ログの分析

この項の例では、エラー・メッセージを発行し、認証チェックを行って、イベントをシステム・ファイルに格納するコンピュータ・システムについて説明しています。セキュリティ上の問題やその他の問題があるかどうかを判別するには、システム・ファイルを分析する必要があります。このアクティビティは、ソフトウェアでファイルを精査して問題点を検索するために、ログ精査とも呼ばれます。これらの例のソース・データは非常に多くの場所をとるため、表示されていません。これらの例では、AUTHENLOG表はログ・ファイルに由来しています。

例21-16 4つ以上の連続する同一メッセージ

この例の問合せでは、考えられる3つの'errtype'値であるerror、noticeおよびwarnのセットからの4つ以上の連続する同一メッセージの発生をシークします。

```
SELECT MR_SEC.ERRTYPE,
       MR_SEC.MNO     AS Pattern,
       MR_SEC.CNT     AS Count,
       SUBSTR(MR_SEC.MSG_W, 1, 30)   AS Message,
       MR_SEC.START_T AS Starting_on,
       MR_SEC.END_T   AS Ending_on
FROM AUTHENLOG
MATCH_RECOGNIZE(
    PARTITION BY errtype
    ORDER BY tstamp
    MEASURES
```

```
        S.tstamp         AS START_T,
        W.tstamp         AS END_T,
        W.message        AS MSG_W,
        COUNT(*)         AS CNT,
        MATCH_NUMBER()   AS MNO
    ONE ROW PER MATCH
    AFTER MATCH SKIP PAST LAST ROW
    PATTERN ( S W{3,} )
    DEFINE  W  AS  W.message = PREV (W.message)
    ) MR_SEC
ORDER BY ErrType, Pattern;

ERRTYP PATTERN COUNT MESSAGE              STARTING_ON                    ENDING_ON
------ ------- ----- -------------------  ------------------------------ ------------------------------
error        1     4 script not found or  09-JAN-10 12.00.06.000006 PM 09-JAN-10 12.00.15.000015 PM
error        2     4 File does not exist  04-FEB-10 12.00.18.000018 PM 04-FEB-10 12.00.23.000023 PM
error        3     4 File does not exist  06-FEB-10 12.00.25.000025 PM 06-FEB-10 12.00.33.000033 PM
error        4     4 File does not exist  13-FEB-10 12.00.19.000019 PM 14-FEB-10 12.00.07.000007 PM
error        5     5 File does not exist  28-FEB-10 12.00.27.000027 PM 28-FEB-10 12.00.34.000034 PM
error        6     4 script not found or  05-APR-10 12.00.19.000019 PM 05-MAR-10 12.00.23.000023 PM
error        7     4 File does not exist  07-MAR-10 12.00.31.000031 PM 08-MAR-10 12.00.02.000002 PM
error        8     4 File does not exist  14-MAR-10 12.00.19.000019 PM 15-MAR-10 12.00.00.000000 PM
error        9     4 File does not exist  20-MAR-10 12.00.02.000002 PM 20-MAR-10 12.00.06.000006 PM
error       10     5 File does not exist  28-APR-10 12.00.24.000024 PM 28-APR-10 12.00.31.000031 PM
error       11     5 script not found or  01-MAY-10 12.00.15.000015 PM 02-MAY-10 12.00.11.000011 PM
error       12     5 user jsmith: authen  02-MAY-10 12.00.54.000054 PM 03-MAY-10 12.00.11.000011 PM
error       13     4 File does not exist  09-MAY-10 12.00.46.000046 PM 10-MAY-10 12.00.01.000001 PM
error       14     4 File does not exist  20-MAY-10 12.00.42.000042 PM 20-MAY-10 12.00.47.000047 PM
error       15     4 user jsmith: authen  21-MAY-10 12.00.08.000008 PM 21-MAY-10 12.00.18.000018 PM
error       16     4 File does not exist  24-MAY-10 12.00.07.000007 PM 25-MAY-10 12.00.01.000001 PM
error       17     4 user jsmith: authen  12-JUN-10 12.00.00.000000 PM 12-JUN-10 12.00.07.000007 PM
error       18     4 script not found or  12-JUN-10 12.00.18.000018 PM 13-JUN-10 12.00.01.000001 PM
error       19     4 File does not exist  17-JUN-10 12.00.23.000023 PM 17-JUN-10 12.00.30.000030 PM
error       20     5 File does not exist  21-JUN-10 12.00.31.000031 PM 22-JUN-10 12.00.01.000001 PM
error       21     4 user jsmith: authen  22-JUN-10 12.00.36.000036 PM 22-JUN-10 12.00.56.000056 PM
error       22     4 File does not exist  08-JUL-10 12.00.29.000029 PM 08-JUL-10 12.00.32.000032 PM
error       23     6 user jsmith: authen  10-JUL-10 12.00.43.000043 PM 11-JUL-10 12.00.06.000006 PM
error       24     4 File does not exist  12-JUL-10 12.00.09.000009 PM 12-JUL-10 12.00.22.000022 PM
error       25     4 File does not exist  26-JUL-10 12.00.18.000018 PM 27-JUL-10 12.00.04.000004 PM
error       26     4 File does not exist  03-AUG-10 12.00.02.000002 PM 03-AUG-10 12.00.11.000011 PM
error       27     4 File does not exist  23-AUG-10 12.00.04.000004 PM 23-AUG-10 12.00.18.000018 PM
error       28     5 File does not exist  24-AUG-10 12.00.09.000009 PM 26-AUG-10 12.00.00.000000 PM
error       29     4 script not found or  09-SEP-10 12.00.03.000003 PM 09-SEP-10 12.00.09.000009 PM
error       30     4 script not found or  11-SEP-10 12.00.22.000022 PM 11-SEP-10 12.00.31.000031 PM
error       31     4 script not found or  23-SEP-10 12.00.09.000009 PM 23-SEP-10 12.00.16.000016 PM
error       32     5 script not found or  17-OCT-10 12.00.02.000002 PM 18-OCT-10 12.00.09.000009 PM
error       33     4 File does not exist  20-OCT-10 12.00.35.000035 PM 21-OCT-10 12.00.00.000000 PM
error       34     5 File does not exist  21-OCT-10 12.00.16.000016 PM 21-OCT-10 12.00.35.000035 PM
error       35     4 File does not exist  26-OCT-10 12.00.25.000025 PM 26-OCT-10 12.00.35.000035 PM
error       36     4 user jsmith: authen  26-OCT-10 12.00.43.000043 PM 26-OCT-10 12.00.49.000049 PM
error       37     4 user jsmith: authen  01-NOV-10 12.00.35.000035 PM 01-NOV-10 12.00.39.000039 PM
error       38     4 File does not exist  09-NOV-10 12.00.46.000046 PM 10-NOV-10 12.00.09.000009 PM
error       39     4 user jsmith: authen  11-NOV-10 12.00.14.000014 PM 11-NOV-10 12.00.30.000030 PM
error       40     4 user jsmith: authen  22-NOV-10 12.00.46.000046 PM 23-NOV-10 12.00.07.000007 PM
error       41     4 script not found or  03-DEC-10 12.00.14.000014 PM 03-DEC-10 12.00.27.000027 PM
error       42     5 File does not exist  07-DEC-10 12.00.02.000002 PM 07-DEC-10 12.00.37.000037 PM
error       43     4 user jsmith: authen  11-DEC-10 12.00.06.000006 PM 11-DEC-10 12.00.11.000011 PM
error       44     4 user jsmith: authen  19-DEC-10 12.00.26.000026 PM 20-DEC-10 12.00.04.000004 PM
error       45     4 user jsmith: authen  25-DEC-10 12.00.11.000011 PM 25-DEC-10 12.00.17.000017 PM
error       46     4 File does not exist  04-JAN-11 12.00.09.000009 PM 04-JAN-11 12.00.19.000019 PM
```

```
error       47     4 user jsmith: authen  10-JAN-11 12.00.23.000023 PM 11-JAN-11 12.00.03.000003 PM
error       48     4 File does not exist  11-JAN-11 12.00.14.000014 PM 11-JAN-11 12.00.24.000024 PM
notice       1     4 Child 3228: Release  08-JAN-10 12.00.38.000038 PM 09-JAN-10 12.00.02.000002 PM
notice       2     4 Child 3228: Release  16-JAN-10 12.00.10.000010 PM 17-JAN-10 12.00.13.000013 PM
notice       3     4 Child 1740: Startin  28-JAN-10 12.00.17.000017 PM 28-JAN-10 12.00.22.000022 PM
notice       4     4 Child 1740: Child p  08-MAR-10 12.00.37.000037 PM 08-MAR-10 12.00.40.000040 PM
notice       5     4 Child 3228: All wor  19-APR-10 12.00.10.000010 PM 19-APR-10 12.00.15.000015 PM
notice       6     4 Child 1740: Acquire  02-MAY-10 12.00.38.000038 PM 02-MAY-10 12.00.46.000046 PM
notice       7     4 Child 1740: Starting 09-MAY-10 12.00.03.000003 PM 09-MAY-10 12.00.08.000008 PM
notice       8     4 Child 3228: Child pr 18-MAY-10 12.00.38.000038 PM 18-MAY-10 12.00.45.000045 PM
notice       9     4 Child 3228: All work 25-JUL-10 12.00.04.000004 PM 25-JUL-10 12.00.09.000009 PM
notice      10     4 Child 3228: All work 24-AUG-10 12.00.11.000011 PM 24-AUG-10 12.00.18.000018 PM
notice      11     4 Child 1740: Starting 19-SEP-10 12.00.05.000005 PM 19-SEP-10 12.00.15.000015 PM
notice      12     4 Child 1740: Acquired 06-OCT-10 12.00.07.000007 PM 06-OCT-10 12.00.13.000013 PM
notice      13     4 Child 1740: Starting 09-JAN-11 12.00.12.000012 PM 09-JAN-11 12.00.18.000018 PM
warn         1  3448 The ScriptAlias dire 01-JAN-10 12.00.00.000000 PM 17-JAN-11 12.00.18.000018 PM

62 rows selected.
```

例21-17 4つ以上の連続する認証失敗

この例では、IP開始アドレスに関係なく、4つ以上の連続する認証失敗を検索します。出力には、最初が5行、最後のものが4行の2つの一致が表示されています。

```
SELECT MR_SEC2.ERRTYPE  AS Authen,
       MR_SEC2.MNO      AS Pattern,
       MR_SEC2.CNT      AS Count,
       MR_SEC2.IPADDR   AS On_IP,
       MR_SEC2.TSTAMP   AS Occurring_on
FROM AUTHENLOG
MATCH_RECOGNIZE(
    PARTITION BY errtype
    ORDER BY tstamp
    MEASURES
        COUNT(*)          AS CNT,
        MATCH_NUMBER()    AS MNO
    ALL ROWS PER MATCH
    AFTER MATCH SKIP TO LAST W
    PATTERN ( S W{3,} )
    DEFINE S AS S.message LIKE '%authenticat%',
           W AS W.message = PREV (W.message)
    ) MR_SEC2
ORDER BY Authen, Pattern, Count;

AUTHEN     PATTERN      COUNT  ON_IP           OCCURRING_ON
------     -------  ---------  ------------    ---------------------------
error            1          1  10.111.112.3    02-MAY-10 12.00.54.000054 PM
error            1          2  10.111.112.6    03-MAY-10 12.00.07.000007 PM
error            1          3  10.111.112.6    03-MAY-10 12.00.08.000008 PM
error            1          4  10.111.112.6    03-MAY-10 12.00.09.000009 PM
error            1          5  10.111.112.6    03-MAY-10 12.00.11.000011 PM
error            2          1  10.111.112.5    21-MAY-10 12.00.08.000008 PM
error            2          2  10.111.112.6    21-MAY-10 12.00.16.000016 PM
error            2          3  10.111.112.4    21-MAY-10 12.00.17.000017 PM
error            2          4  10.111.112.6    21-MAY-10 12.00.18.000018 PM
error            3          1  10.111.112.5    12-JUN-10 12.00.00.000000 PM
error            3          2  10.111.112.4    12-JUN-10 12.00.04.000004 PM
error            3          3  10.111.112.3    12-JUN-10 12.00.06.000006 PM
error            3          4  10.111.112.3    12-JUN-10 12.00.07.000007 PM
error            4          1  10.111.112.5    22-JUN-10 12.00.36.000036 PM
```

```
error            4          2  10.111.112.5     22-JUN-10 12.00.50.000050 PM
error            4          3  10.111.112.5     22-JUN-10 12.00.53.000053 PM
error            4          4  10.111.112.6     22-JUN-10 12.00.56.000056 PM
error            5          1  10.111.112.4     10-JUL-10 12.00.43.000043 PM
error            5          2  10.111.112.6     10-JUL-10 12.00.48.000048 PM
error            5          3  10.111.112.6     10-JUL-10 12.00.51.000051 PM
error            5          4  10.111.112.3     11-JUL-10 12.00.00.000000 PM
error            5          5  10.111.112.5     11-JUL-10 12.00.04.000004 PM
error            5          6  10.111.112.3     11-JUL-10 12.00.06.000006 PM
error            6          1  10.111.112.4     26-OCT-10 12.00.43.000043 PM
error            6          2  10.111.112.4     26-OCT-10 12.00.47.000047 PM
error            6          3  10.111.112.4     26-OCT-10 12.00.48.000048 PM
error            6          4  10.111.112.5     26-OCT-10 12.00.49.000049 PM
error            7          1  10.111.112.3     01-NOV-10 12.00.35.000035 PM
error            7          2  10.111.112.5     01-NOV-10 12.00.37.000037 PM
error            7          3  10.111.112.5     01-NOV-10 12.00.38.000038 PM
error            7          4  10.111.112.3     01-NOV-10 12.00.39.000039 PM
error            8          1  10.111.112.6     11-NOV-10 12.00.14.000014 PM
error            8          2  10.111.112.5     11-NOV-10 12.00.20.000020 PM
error            8          3  10.111.112.6     11-NOV-10 12.00.24.000024 PM
error            8          4  10.111.112.3     11-NOV-10 12.00.30.000030 PM
error            9          1  10.111.112.5     22-NOV-10 12.00.46.000046 PM
error            9          2  10.111.112.5     22-NOV-10 12.00.51.000051 PM
error            9          3  10.111.112.3     23-NOV-10 12.00.06.000006 PM
error            9          4  10.111.112.3     23-NOV-10 12.00.07.000007 PM
error           10          1  10.111.112.5     11-DEC-10 12.00.06.000006 PM
error           10          2  10.111.112.4     11-DEC-10 12.00.07.000007 PM
error           10          3  10.111.112.5     11-DEC-10 12.00.08.000008 PM
error           10          4  10.111.112.6     11-DEC-10 12.00.11.000011 PM
error           11          1  10.111.112.5     19-DEC-10 12.00.26.000026 PM
error           11          2  10.111.112.5     20-DEC-10 12.00.01.000001 PM
error           11          3  10.111.112.4     20-DEC-10 12.00.03.000003 PM
error           11          4  10.111.112.3     20-DEC-10 12.00.04.000004 PM
error           12          1  10.111.112.4     25-DEC-10 12.00.11.000011 PM
error           12          2  10.111.112.4     25-DEC-10 12.00.12.000012 PM
error           12          3  10.111.112.4     25-DEC-10 12.00.16.000016 PM
error           12          4  10.111.112.3     25-DEC-10 12.00.17.000017 PM
error           13          1  10.111.112.6     10-JAN-11 12.00.23.000023 PM
error           13          2  10.111.112.6     11-JAN-11 12.00.00.000000 PM
error           13          3  10.111.112.3     11-JAN-11 12.00.02.000002 PM
error           13          4  10.111.112.4     11-JAN-11 12.00.03.000003 PM

55 rows selected.
```

例21-18 同じIPアドレスからの認証失敗

例21-18の問合せは例21-17とほぼ同じですが、3回以上連続して発生した認証失敗を同じIP開始アドレスから検索しています。

```
SELECT MR_S3.MNO AS Pattern, MR_S3.CNT AS Count,
       MR_S3.ERRTYPE AS Type, MR_S3.IPADDR AS On_IP_addr,
       MR_S3.START_T AS Starting_on, MR_S3.END_T AS Ending_on
FROM AUTHENLOG
MATCH_RECOGNIZE(
    PARTITION BY errtype
    ORDER BY tstamp
    MEASURES
        S.tstamp         AS START_T,
        W.tstamp         AS END_T,
        W.ipaddr         AS IPADDR,
```

```
        COUNT(*)          AS CNT,
        MATCH_NUMBER()    AS MNO
    ONE ROW  PER MATCH
    AFTER MATCH SKIP TO LAST W
    PATTERN ( S W{2,} )
    DEFINE  S AS S.message LIKE '%authenticat%',
            W  AS  W.message = PREV (W.message)
                   AND W.ipaddr = PREV (W.ipaddr)
    ) MR_S3
ORDER BY Type, Pattern;

PATTERN COUNT  TYPE    ON_IP_ADDR     STARTING_ON                    ENDING_ON
------- -----  -----   ------------   ----------------------------   ----------------------------
      1     4  error   10.111.112.6   03-MAY-10 12.00.07.000007 PM   03-MAY-10 12.00.11.000011 PM
      2     3  error   10.111.112.5   22-JUN-10 12.00.36.000036 PM   22-JUN-10 12.00.53.000053 PM
      3     3  error   10.111.112.4   27-JUN-10 12.00.03.000003 PM   27-JUN-10 12.00.08.000008 PM
      4     3  error   10.111.112.6   19-JUL-10 12.00.15.000015 PM   19-JUL-10 12.00.17.000017 PM
      5     3  error   10.111.112.4   26-OCT-10 12.00.43.000043 PM   26-OCT-10 12.00.48.000048 PM
      6     3  error   10.111.112.4   25-DEC-10 12.00.11.000011 PM   25-DEC-10 12.00.16.000016 PM
      7     3  error   10.111.112.5   12-JAN-11 12.00.01.000001 PM   12-JAN-11 12.00.08.000008 PM

7 rows selected.
```

21.6.3 パターン一致の例: セッション化

セッション化とは、通常は1つのセッションで複数のイベントが関与するユーザー・アクティビティの個別のセッションを定義するプロセスです。パターン一致によって、セッション化のための問合せの表現が容易になります。たとえば、Webサイトへのビジターが標準的なセッション中に何ページを閲覧するか把握したい場合があります。通信プロバイダの場合は、2人のユーザー間の電話セッションで接続が切れてユーザーがリダイアルする場合の特徴を把握することもあります。企業は、ユーザー・セッション動作を理解することによって重要な価値を引き出すことができます。これは、サービスの提供と向上、価格設定、マーケティングなどを定義するうえで、企業にとって役立つためです。

次の例では、Webサイトのクリックストリームに関連したセッション化の導入例を2つ示し、さらに電話に関わる例を示しています。

例21-19 クリックストリーム・データに関する単純なセッション化

例21-19では、クリックストリーム・データを分析するための単純なセッション化を説明しています。一連の行の目的は、セッションを検出し、各セッションにセッションIDを割り当てて、各入力行をセッションIDとともに表示することです。次のデータは、すべてのページ・リクエストを追跡するWebサーバーのシステム・ログから得られます。それぞれの行がページをリクエストするユーザーのイベントである行セットから開始します。この単純な例では、ユーザーIDであるパーティション・キーと、ユーザーがページをリクエストした時間を示すタイムスタンプがデータに含まれています。Webシステム・ログでは、指定のページをユーザーがリクエストした時期が表示されますが、そのページの閲覧をユーザーが停止した時期は示されません。

例21-19では、セッションは、同じパーティション・キー(User_ID)を持つ1つ以上の時系列の行のシーケンスで、タイムスタンプ間の時間差は指定のしきい値未満であるものとして定義されています。この場合、しきい値は10時間単位です。行のタイムスタンプが10単位より離れている場合は、別個のセッションの行と見なされます。ここで使用されている10単位のしきい値は任意の数字です。実際のケースごとに、アナリストの判断で最適なしきい値の時間差を決める必要があります。Webサイト訪問のセッションを分離する場合、履歴的にみて、30分という時間差が一般にしきい値として使用されています。

最初に、クリックストリーム・イベントの表を作成してください。

```
CREATE TABLE Events(
   Time_Stamp NUMBER,
   User_ID VARCHAR2(10)
  );
```

次に、データを挿入します。次の挿入文は順序付けられており、読み取りやすいように空白を入れて、パーティションとセッションを確認できるようにしています。実際には、イベントはタイムスタンプの順序で到着し、異なるユーザー・セッションの行が混在しています。

```
  INSERT INTO Events(Time_Stamp, User_ID) VALUES ( 1, 'Mary');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (11, 'Mary');

  INSERT INTO Events(Time_Stamp, User_ID) VALUES (23, 'Mary');

  INSERT INTO Events(Time_Stamp, User_ID) VALUES (34, 'Mary');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (44, 'Mary');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (53, 'Mary');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (63, 'Mary');

  INSERT INTO Events(Time_Stamp, User_ID) VALUES ( 3, 'Richard');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (13, 'Richard');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (23, 'Richard');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (33, 'Richard');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (43, 'Richard');

  INSERT INTO Events(Time_Stamp, User_ID) VALUES (54, 'Richard');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (63, 'Richard');

  INSERT INTO Events(Time_Stamp, User_ID) VALUES ( 2, 'Sam');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (12, 'Sam');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (22, 'Sam');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (32, 'Sam');

  INSERT INTO Events(Time_Stamp, User_ID) VALUES (43, 'Sam');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (47, 'Sam');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (48, 'Sam');

  INSERT INTO Events(Time_Stamp, User_ID) VALUES (59, 'Sam');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (60, 'Sam');
  INSERT INTO Events(Time_Stamp, User_ID) VALUES (68, 'Sam');
```

次の行パターン一致の問合せでは、それぞれの入力行がSession_IDとともに表示されます。前述したように、イベントの分離が10時間単位以下であれば、それらのイベントは同じセッションに含まれていると見なされます。このセッションしきい値は、パターン変数のDEFINE句で表されます。

```
SELECT time_stamp, user_id, session_id
FROM Events MATCH_RECOGNIZE
         (PARTITION BY User_ID ORDER BY Time_Stamp
          MEASURES match_number() AS session_id
          ALL ROWS PER MATCH
          PATTERN (b s*)
          DEFINE
             s AS (s.Time_Stamp - prev(Time_Stamp) <= 10)
         )
ORDER BY user_id, time_stamp;
```

出力は次のようになります。

```
TIME_STAMP USER_ID    SESSION_ID
---------- ---------- ----------
         1 Mary                1
        11 Mary                1
        23 Mary                2
        34 Mary                3
```

```
        44 Mary                3
        53 Mary                3
        63 Mary                3
         3 Richard             1
        13 Richard             1
        23 Richard             1
        33 Richard             1
        43 Richard             1
        54 Richard             2
        63 Richard             2
         2 Sam                 1
        12 Sam                 1
        22 Sam                 1
        32 Sam                 1
        43 Sam                 2
        47 Sam                 2
        48 Sam                 2
        59 Sam                 3
        60 Sam                 3
        68 Sam                 3

24 rows selected.
```

例21-20 集計による単純なセッション化

例21-19に示すように詳細レベルの行にセッション番号を割り当てると、ただちに分析プロセスが開始されます。セッション化データのビジネス値は、セッション別の集計後でないと出力されません。

この例では、データが集計され、セッションごとに、Session_ID、User_ID、セッションごとの集計イベント数および合計セッション期間の列を含む行が1行出力されます。この出力によって、各ユーザーがセッションごとに行ったクリック回数と、各セッションの持続時間を容易に確認できます。また、この問合せからのデータを利用して、セッション期間の最大値、最小値、平均値など、その他の多くの分析を実行することもできます。

```
SELECT session_id, user_id, start_time, no_of_events, duration
FROM Events MATCH_RECOGNIZE
     (PARTITION BY User_ID
      ORDER BY Time_Stamp
      MEASURES MATCH_NUMBER() session_id,
               COUNT(*) AS no_of_events,
               FIRST(time_stamp) start_time,
               LAST(time_stamp) - FIRST(time_stamp) duration
      PATTERN (b s*)
      DEFINE
         s AS (s.Time_Stamp - PREV(Time_Stamp) <= 10)
     )
ORDER BY user_id, session_id;
```

出力は次のようになります。

```
SESSION_ID USER_ID    START_TIME NO_OF_EVENTS   DURATION
---------- ---------- ---------- ------------ ----------
         1 Mary                1            2         10
         2 Mary               23            1          0
         3 Mary               34            4         29
         1 Richard             3            5         40
         2 Richard            54            2          9
         1 Sam                 2            4         30
         2 Sam                43            3          5
         3 Sam                59            3          9
```

```
8 rows selected.
```

例21-21 接続の切断を伴う電話のセッション化

クリックストリーム・データを示した例21-19および例21-20では、ページ閲覧の終了時間を示す明示的な終了点がソース・データにありませんでした。ユーザー・アクティビティの明確な終了点が指定されたとしても、終了点では、ユーザーがセッションを終了しようとしていることを示していない場合もあります。携帯電話サービスの使用中に接続が切れたユーザーを考えてみます。通常、このユーザーはリダイアルして通話を続けます。このシナリオでは、同じ電話番号のペアが関連する複数の通話が1回の電話セッションに含まれると考える必要があります。

例21-21に、電話のセッション化を示します。この例では通話の詳細レコード・データをセッション化のベースに使用します。ここで通話データ・レコード行にはStart_Time、End_Time、Caller_ID、Callee_IDが含まれます。後に示す問合せでは、次の処理が行われます。

- データをcaller_idおよびcallee_idによってパーティション化します。
- 連続する通話間の時間差が60秒というしきい値以内の場合に、発信者から受信者への通話を1つのセッションにグループ化するセッションを検索します。このしきい値は、パターン変数BのDEFINE句で指定されます。
- セッションごとに、次の値を返します(MEASURES句を参照してください)。
  - session_id、発信者および受信者
  - セッション内で通話が再開された回数
  - 合計有効通話時間(セッション中に電話が接続されていた合計時間)
  - 合計中断期間(セッション中に電話が切断されていた合計時間)

```
SELECT  Caller, Callee, Start_Time, Effective_Call_Duration,
                (End_Time - Start_Time) - Effective_Call_Duration
                     AS Total_Interruption_Duration, No_Of_Restarts, Session_ID
FROM my_cdr MATCH_RECOGNIZE
        ( PARTITION BY Caller, Callee ORDER BY Start_Time
           MEASURES
                A.Start_Time               AS Start_Time,
                End_Time                   AS End_Time,
                SUM(End_Time - Start_Time) AS Effective_Call_Duration,
                COUNT(B.*)                 AS No_Of_Restarts,
                MATCH_NUMBER()             AS Session_ID
           PATTERN (A B*)
           DEFINE B AS B.Start_Time - PREV(B.end_Time) < 60
         );
```

前出の問合せでは大量のデータを意味のあるものにする必要があり、それには相当の領域が消費されるため、ここではINSERT文は含まれていません。サンプル出力は次のようになります。

```
SQL> desc my_cdr
Name            Null?        Type
--------------  ----------   ----------
CALLER          NOT NULL     NUMBER(38)
CALLEE          NOT NULL     NUMBER(38)
START_TIME      NOT NULL     NUMBER(38)
END_TIME        NOT NULL     NUMBER(38)

SELECT * FROM my_cdr ORDER BY 1, 2, 3, 4;

CALLER   CALLEE  START_TIME   END_TIME
```

```
------   ------  ----------   ---------
     1        7        1354        1575
     1        7        1603        1829
     1        7        1857        2301
     1        7        2320        2819
     1        7        2840        2964
     1        7       64342       64457
     1        7       85753       85790
     1        7       85808       85985
     1        7       86011       86412
     1        7       86437       86546
     1        7      163436      163505
     1        7      163534      163967
     1        7      163982      164454
     1        7      214677      214764
     1        7      214782      215248
     1        7      216056      216271
     1        7      216297      216728
     1        7      216747      216853
     1        7      261138      261463
     1        7      261493      261864
     1        7      261890      262098
     1        7      262115      262655
     1        7      301931      302226
     1        7      302248      302779
     1        7      302804      302992
     1        7      303015      303258
     1        7      303283      303337
     1        7      383019      383378
     1        7      383407      383534
     1        7      424800      425096

30 rows selected.

CALLER CALLEE START_TIME EFFECTIVE_CALL TOTAL_INTERUPTION NO_OF_RE SESSION_ID
------ ------- ---------- -------------- ----------------- -------- ----------
    1        7       1354           1514                96        4          1
    1        7      64342            115                 0        0          2
    1        7      85753            724                69        3          3
    1        7     163436            974                44        2          4
    1        7     214677            553                18        1          5
    1        7     216056            752                45        2          6
    1        7     261138           1444                73        3          7
    1        7     301931           1311                95        4          8
    1        7     383019            486                29        1          9
    1        7     424800            296                 0        0         10

10 rows selected.
```

21.6.4 パターン一致の例: 会計トラッキング

一般的な会計アプリケーションでは、疑わしい会計パターンを検索します。例21-22は、異常であると定義した特定の基準を満たしているため、疑わしく見える資金移動を検出する方法を示しています。

例21-22 疑わしい資金移動

例21-22では、資金の移動時に疑わしく見えるパターンを検索しています。このケースでは、それを、30日以内に3回以上の少額($2000未満)の資金移動があり、続けて最後の少額の資金移動から10日以内に大量の($1,000,000を上回る)資

金移動が発生した場合と定義しています。話を分かりやすくするために、表とデータは非常に基本的なものになっています。

最初に、必要なデータを含む表を作成します。

```
CREATE TABLE event_log
     ( time          DATE,
       userid        VARCHAR2(30),
       amount        NUMBER(10),
       event         VARCHAR2(10),
       transfer_to   VARCHAR2(10));
```

その後、データをevent_logに挿入します。

```
INSERT INTO event_log VALUES
   (TO_DATE('01-JAN-2012', 'DD-MON-YYYY'), 'john', 1000000, 'deposit', NULL);
INSERT INTO event_log VALUES
   (TO_DATE('05-JAN-2012', 'DD-MON-YYYY'), 'john', 1200000, 'deposit', NULL);
INSERT INTO event_log VALUES
   (TO_DATE('06-JAN-2012', 'DD-MON-YYYY'), 'john', 1000, 'transfer', 'bob');
INSERT INTO event_log VALUES
   (TO_DATE('15-JAN-2012', 'DD-MON-YYYY'), 'john', 1500, 'transfer', 'bob');
INSERT INTO event_log VALUES
   (TO_DATE('20-JAN-2012', 'DD-MON-YYYY'), 'john', 1500, 'transfer', 'allen');
INSERT INTO event_log VALUES
   (TO_DATE('23-JAN-2012', 'DD-MON-YYYY'), 'john', 1000, 'transfer', 'tim');
INSERT INTO event_log VALUES
   (TO_DATE('26-JAN-2012', 'DD-MON-YYYY'), 'john', 1000000, 'transfer', 'tim');
INSERT INTO event_log VALUES
   (TO_DATE('27-JAN-2012', 'DD-MON-YYYY'), 'john', 500000, 'deposit', NULL);
```

次に、この表を問い合せることができます。

```
SELECT userid, first_t, last_t, amount
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE
   (PARTITION BY userid ORDER BY time
    MEASURES FIRST(x.time) first_t, y.time last_t, y.amount amount
    PATTERN ( x{3,} y )
    DEFINE x AS (event='transfer' AND amount < 2000),
           y AS (event='transfer' AND amount >= 1000000 AND
                 LAST(x.time) - FIRST(x.time) < 30 AND
                 y.time - LAST(x.time) < 10));

USERID        FIRST_T      LAST_T       AMOUNT
----------    ---------    ---------    -------
john          06-JAN-12    26-JAN-12    1000000
```

この文では、1つ目の太字テキストが少額の資金移動を示し、2つ目の太字テキストが大量の資金移動を示し、3つ目の太字テキストが30日以内に少額の資金移動が発生したことを示し、4つ目の太字テキストが最後の少額の資金移動から10日以内に大量の資金移動が発生したことを示しています。

この文をさらに改良して、次のように、疑わしい資金移動の受取人を含めるようにすることができます。

```
SELECT userid, first_t, last_t, amount, transfer_to
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE
   (PARTITION BY userid ORDER BY time
    MEASURES z.time first_t, y.time last_t, y.amount amount,
             y.transfer_to transfer_to
    PATTERN ( z x{2,} y )
```

```
    DEFINE z AS (event='transfer' AND amount < 2000),
           x AS (event='transfer' AND amount <= 2000 AND
                 PREV(x.transfer_to) <> x.transfer_to),
           y AS (event='transfer' AND amount >= 1000000 AND
                 LAST(x.time) - z.time < 30 AND
                 y.time - LAST(x.time) < 10 AND
                 SUM(x.amount) + z.amount < 20000);

USERID        FIRST_T      LAST_T       AMOUNT      TRANSFER_TO
----------    ---------    ---------    -------     -----------
john          15-JAN-12    26-JAN-12    1000000     tim
```

この文では、1つ目の太字テキストが最初の少額の資金移動を示し、次の太字テキストが異なる口座への2回以上の少額の資金移動を示し、3つ目の太字テキストが$20,000未満の少額の資金移動すべての合計を示しています。

22 モデリングのSQL

この章では、SQLモデリングの使用方法について説明します。次の内容が含まれます。

- データ・ウェアハウスにおけるSQLモデリングの概要
- SQLモデリングの基本的なトピック
- SQLモデリングの高度なトピック
- SQLモデリングのパフォーマンスに関する考慮事項
- SQLモデリングの例

22.1 データ・ウェアハウスにおけるSQLモデリングの概要

MODEL句によって、SQL計算の機能性と柔軟性が向上します。MODEL句を使用すると、問合せ結果から多次元配列を作成し、この配列に(ルールと呼ばれる)式を適用して新しい値を計算できます。ルールは、基本的な算術式から再帰型を使用した連立方程式まで様々です。一部のアプリケーションでは、MODEL句の機能をPCベースのスプレッドシートと置き換えることができます。SQLのモデルには、Oracle Databaseが持つスケーラビリティ、管理性、コラボレーション機能、セキュリティ機能などの長所が生かされています。問合せのコア・エンジンは、無制限の量のデータを処理できます。データベース内でモデルを定義および実行することにより、別個のモデリング環境間で大規模なデータセットを転送する必要がなくなります。モデルはワークグループ間で簡単に共有できるため、すべてのアプリケーションで計算の一貫性を維持できます。モデルの共有と同様に、アクセスもOracleのセキュリティ機能を使用して厳密に制御できます。豊富な機能を備えたMODEL句を使用することで、あらゆるタイプのアプリケーションを強化できます。

MODEL句を使用すると、パーティション列、ディメンション列およびメジャー列の3つのグループに問合せの列を対応付けることによって、多次元配列を作成できます。これらの要素は、次のタスクを実行します。

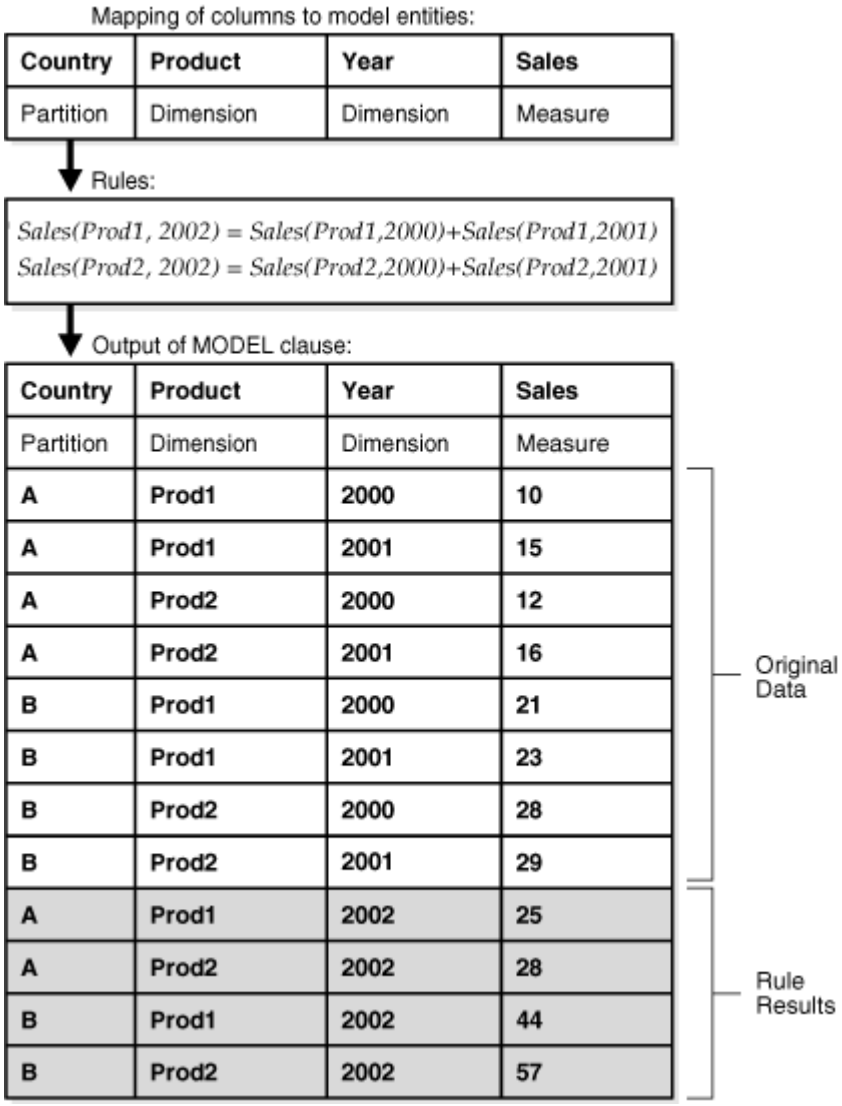
- パーティション列では、分析計算およびレポート用SQL関数で説明した分析関数のパーティションと同様の方法で結果セットの論理ブロックを定義します。MODEL句のルールは、他のパーティションとは別個にパーティションごとに適用されます。したがって、パーティションは、MODEL句での計算をパラレル化するための境界点となります。
- ディメンション列では、多次元配列を定義します。この列は、パーティション内のセルの識別に使用します。デフォルトでは、ディメンションの1つの完全な組合せはパーティション内の1つのセルのみを識別する必要があります。デフォルト・モードの場合、リレーショナル表のキーと同様のものとみなされます。
- メジャーは、スター・スキーマのファクト表のメジャーと等価です。通常、メジャーには販売単位やコストなどの数値が格納されています。各セルにアクセスするには、各セルのディメンションの完全な組合せを指定します。各パーティションには、特定のディメンションの組合せと一致するセルが含まれることがあります。

MODEL句ではルールを指定して、パーティション列およびディメンション列で定義された多次元配列内のセルのメジャー値を操作できます。ルールは、ディメンション値を直接指定して、メジャー列値へのアクセスおよび更新を行います。ルールで参照を使用することにより、モデルの可読性が向上します。ルールは簡潔かつ柔軟なもので、表現力を最大限に高めるためにワイルド・カードおよびループ・コンストラクトを使用できます。Oracle Databaseでは、効率的にルールを評価し、可能なかぎりモデル計算をパラレル化し、MODEL句を他のSQL句とシームレスに統合します。したがって、データベースのビジネス・モデルを計算する上でMODEL句はスケーラブルで管理性に優れた方法であるといえます。

図22-1に、SQLのモデリング機能の概要を示します。図は3つの部分から構成されています。最上部では、一般的な表をパーティション列、ディメンション列およびメジャー列に分割する概念を示しています。中央の部分では、2002年のProd1およびProd2の値を計算する2つのルールを示しています。最下部では、仮想データを含む表にルールを適用した問合せの出力を示しています。白の部分の出力はデータベースから取得した元のデータを表し、グレーの部分の出力はルールで計算された行を表し

ます。パーティションAの結果はパーティションBの結果とは独立して計算されていることに注目してください。

図22-1 モデルの要素



この項では、次の項目について説明します。

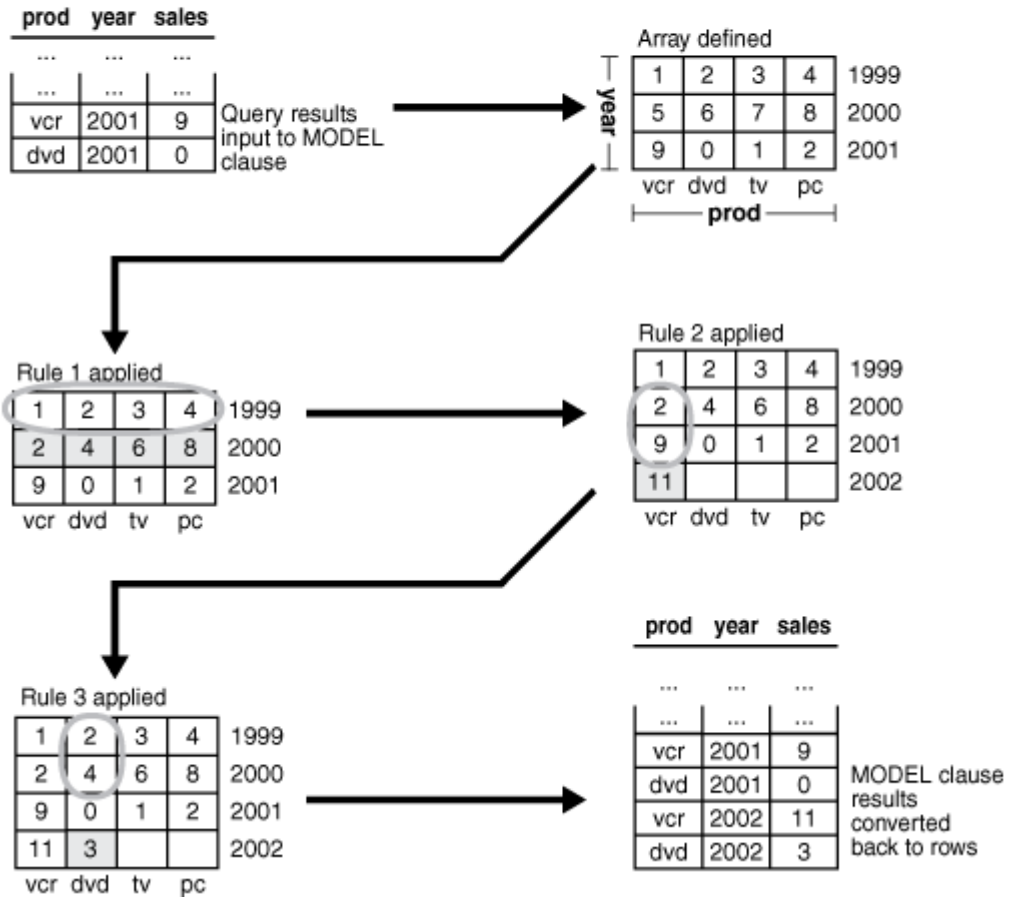
- SQL Modelでのデータの処理方法
- データ・ウェアハウスでSQLモデリングを使用する理由
- SQLモデリングの機能について

22.1.1 SQL Modelでのデータの処理方法

図22-2に、単純なMODEL句における処理フローを示します。ここでは、3つのルールを含んだMODEL句を通してデータの処理を理解していきます。ルールの1つでは既存の値を更新し、残りの2つのルールでは予測用に新しい値を作成します。図では、問合せで取得したデータの行がMODEL句に入力され、配列に再配置されていることが示されています。配列が定義されると、ルールが1つずつデータに適用されます。図22-2のグレーのセルはルールによって作成された新しいデータを表し、楕円で囲まれたセルは新しい値のソース・データを表しています。最後に、更新された値と新規に作成された値の両方を含むデータが行形式に再配置され、問合せの結果として表されます。この問合せでは、表へのデータの挿入はまったく行われていないことに注目してください。

図22-2 MODELの処理フロー

```
MODEL
DIMENSION BY (prod, year)
MEASURES (sales s)
RULES UPSERT
(s[ANY, 2000]=s[CV(prod), CV(year -1)*2],  --Rule 1
 s[vcr, 2002]=s[vcr, 2001]+s[vcr, 2000],   --Rule 2
 s[dvd, 2002]=AVG(s)[CV(prod), year<2001]) --Rule 3
```



22.1.2 データ・ウェアハウスでSQLモデリングを使用する理由

Oracleのモデリングでは、高度な計算をデータに対して実行できます。通常は、ビジネス・ルールをデータに適用してレポートを生成する場合などに使用します。Oracle Databaseでは、モデリングの計算をデータベースに統合しているため、パフォーマンスと管理性が飛躍的に向上します。次の問合せについて考えてみます。

```
SELECT SUBSTR(country, 1, 20) country,
      SUBSTR(product, 1, 15) product, year, sales
FROM sales_view
WHERE country IN ('Italy', 'Japan')
MODEL
  PARTITION BY (country) DIMENSION BY (product, year)
  MEASURES (sales sales)
  RULES
  (sales['Bounce', 2002] = sales['Bounce', 2001] + sales['Bounce', 2000],
   sales['Y Box', 2002] = sales['Y Box', 2001],
   sales['All_Products', 2002] = sales['Bounce', 2002] + sales['Y Box', 2002])
ORDER BY country, product, year;
```

この問合せでは、SQLモデリングの例のベース・スキーマに示すように、sales_viewのデータをcountryでパーティション化して、3つのルールで定義されたモデル計算を国ごとに実行します。このモデルでは、2002年のBounceの売上を2000年と2001年の売上の合計として計算し、2002年のY Boxの売上を2001年の売上と同じ値に設定します。また、2002年については、同年のBounceとY Boxの売上の合計となる、新製品カテゴリAll_Productsを導入しています(sales_viewには、製品All_Productsはありません)。この問合せの出力を次に示します。この中で太字は新しい値を示します。

```
COUNTRY              PRODUCT              YEAR       SALES
-------------------- -------------- ---------- ----------
Italy                Bounce               1999     2474.78
Italy                Bounce               2000     4333.69
Italy                Bounce               2001      4846.3
Italy                Bounce               2002     9179.99
...
Italy                Y Box                1999    15215.16
Italy                Y Box                2000    29322.89
Italy                Y Box                2001    81207.55
Italy                Y Box                2002    81207.55
...
Italy                All_Products         2002    90387.54
...
Japan                Bounce               1999      2961.3
Japan                Bounce               2000     5133.53
Japan                Bounce               2001      6303.6
Japan                Bounce               2002    11437.13
...
Japan                Y Box                1999    22161.91
Japan                Y Box                2000    45690.66
Japan                Y Box                2001    89634.83
Japan                Y Box                2002    89634.83
...
Japan                All_Products         2002   101071.96
...
```

BounceとY Boxの売上値は入力データ内に存在するのに対し、All_Productsの値は導出されていることに注目してください。

22.1.3 SQLモデリングの機能について

Oracle DatabaseのMODEL句の機能を次に示します。

- ディメンション値を使用したセルのアドレッシング

  個々の行のメジャー列は、多次元配列のセルのように処理され、ディメンション値を使用して参照および更新できます。たとえば、ファクト表ft(country, year, sales)では、countryおよびyearをディメンション列に、salesをメジャーに指定し、特定の国および年の売上をsales[country='Spain', year=1999]として参照できます。この場合、1999年のSpainの売上値を取得できます。また、同じ内容を短縮形式で表したsales['Spain', 1999]を使用することもできます。ただし、両者の表記には、わずかな意味の違いがあります。詳細は、SQLモデリングでのセル参照についてを参照してください。

- シンボリック配列の計算

  データの処理に、ルールと呼ばれる一連の式を指定できます。ルールでは、個々のセル、セルの集合または範囲で関数を呼び出すことができます。個々のセルを対象にした例を次に示します。

```
sales[country='Spain',year=2001] = sales['Spain',2000]+ sales['Spain',1999]
```

  この場合、2001年のSpainの売上は、1999年と2000年におけるSpainの売上の合計に設定されています。次に、セルの範囲を対象とした例を示します。

```
sales[country='Spain',year=2001] =
   MAX(sales)['Spain',year BETWEEN 1997 AND 2000]
```

  この場合、2001年のSpainの売上は、1997年から2000年のSpainにおける最高の売上と等しくなるよう設定されています。

- UPSERT、UPSERT ALLおよびUPDATEオプション

  UPSERTオプション(デフォルト)を使用すると、入力データに存在しないセルの値を作成できます。参照先のセルがデータに存在する場合、セルが更新されます。参照先のセルがデータに存在せず、なおかつルールが適切な表記法を使用している場合は、セルが挿入されます。UPSERT ALLオプションでは、より幅広い種類のルールに対してUPSERTの動作を適用できます。反対に、UPDATEオプションでは、新しいセルは一切挿入されません。

  これらのオプションは、グローバルに指定(すべてのルールに適用)することも、ルールごとに指定することもできます。ルール・レベルでオプションを指定すると、グローバル・オプションはオーバーライドされます。次の規則を考慮してください。

```
UPDATE sales['Spain', 1999] = 3567.99,
UPSERT sales['Spain', 2001] = sales['Spain', 2000]+ sales['Spain', 1999]
```

  最初のルールでは、1999年のSpainにおける売上のセルを更新します。2番目のルールでは、2001年のSpainにおける売上のセルが存在する場合は更新し、存在しない場合は新しいセルを作成します。

- ディメンションのワイルド・カード指定

  ANYおよびIS ANYを使用すると、ディメンションの値をすべて指定できます。たとえば、次の文があるとします。

```
sales[ANY, 2001] = sales['Japan', 2000]
```

  このルールでは、2001年のすべての国の売上を2000年のJapanの売上値と同じに設定しています。この場合、ディメンションの値はNULLも含め、すべてANYの指定を満たしています。次のように、IS ANY述語を使用して同じ処理を指定することもできます。

```
sales[country IS ANY, 2001] = sales['Japan', 2000]
```

- CV関数を使用したディメンション値へのアクセス

  ルールの右辺でCV関数を使用すると、ルールの左辺で参照されるセルのディメンション列の値にアクセスできます。これによって、類似した計算を実行する複数のルールを単一のルールに結合できるため、指定が簡潔になります。たとえば、次のルールを結合できます。

```
sales[country='Spain', year=2002] = 1.2 * sales['Spain', 2001],
sales[country='Italy', year=2002] = 1.2 * sales['Italy', 2001],
sales[country='Japan', year=2002] = 1.2 * sales['Japan', 2001]
```

  これらは、次の単一のルールに結合できます。

```
sales[country IN ('Spain', 'Italy', 'Japan'), year=2002] = 1.2 *
   sales[CV(country), 2001]
```

  この場合、CV関数は、左辺のcountryディメンションの値をルールの右辺に渡します。

- 順序付け計算

  セルの集合を更新するルールでは、結果はディメンション値の順序に依存する場合があります。ルールでORDER BYを指定すると、ディメンション値を特定の順序に強制できます。たとえば、次のルールを考えてみます。

```
sales[country IS ANY, year BETWEEN 2000 AND 2003] ORDER BY year =
  1.05 * sales[CV(country), CV(year)-1]
```

  これによって、yearは時系列的に昇順で参照されます。

- ルールの自動順序付け

  AUTOMATIC ORDERキーワードを使用すると、セル間の依存関係に基づいてMODEL句のルールを自動的に順序付けで

きます。たとえば、次の割当ての場合、2番目と3番目のルールが最初のルールよりも先に処理されます。これは、最初のルールが最後の2つのルールに依存しているためです。

```
RULES AUTOMATIC ORDER
{sales[c='Spain', y=2001] = sales[c='Spain', y=2000]
  + sales[c='Spain', y=1999]
sales[c='Spain', y=2000] = 50000,
sales[c='Spain', y=1999] = 40000}
```

- 反復ルール評価

  反復ルール評価を指定すると、終了条件が満たされるまで、反復的にルールが評価されます。次の指定を考えてみます。

```
MODEL DIMENSION BY (x) MEASURES (s)
  RULES ITERATE (4) (s[x=1] = s[x=1]/2)
```

  この文では、式s[x=1] = s[x=1]/2の評価を4回繰り返すように指定しています。反復回数は、MODEL句のITERATEオプションで指定しています。また、UNTIL句を使用すると、終了条件も指定できます。

  反復ルール評価は、ビジネス・アプリケーションのエンティティ間の再帰的関係をモデル化する場合に重要な機能です。たとえば、ローンの総額が金利によって異なる一方で、金利はローンの額に応じて異なります。

- 参照モデル

  1つのモデルには、読取り専用の配列である参照モデルを複数含めることができます。ルールは、複数の参照モデルのセルを参照できます。ルールでセルを更新または挿入できるのは、メイン・モデルと呼ばれる1つの多次元配列のみです。参照モデルを使用すると、異なるディメンションにモデルを関係付けることができます。たとえば、ファクト表ft(country, year, sales)の他に、通貨の換算率の表cr(country, ratio)があるとします。この表には、ディメンション列としてcountry、メジャーとしてratioがあります。この表の各行は、当該国の通貨のUSドルへの換算率を示します。この2つの表は、次のルールで使用できます。

```
dollar_sales['Spain',2001] = sales['Spain',2000] * ratio['Spain']
```

- スケーラブルな計算

  データをパーティション化して、他のパーティションとは関係なく各パーティション内のルールを評価できます。これにより、パーティションに基づいてモデル計算をパラレル化できます。たとえば、次のモデルがあるとします。

```
MODEL PARTITION BY (country) DIMENSION BY (year) MEASURES (sales)
  (sales[year=2001] = AVG(sales)[year BETWEEN 1990 AND 2000]
```

  データはcountryでパーティション化されています。各パーティション内では、2001年の売上高は1990年から2000年の平均売上高になるよう計算されます。パーティションはパラレルに処理できるため、スケーラブルなモデルの実行が実現します。

22.2 SQLモデリングの基本的なトピック

この項では、モデルの基本的な概念および使用方法について説明します。次の内容が含まれます。

- SQLモデリングの例のベース・スキーマ
- MODEL句の構文
- SQLモデリングのキーワード
- SQLモデリングでのセル参照について

- SQLモデリングのルールについて
- SQLモデリングのルールの評価順序
- SQLモデリングのルールのグローバルおよびローカル・キーワード
- UPDATE、UPSERTおよびUPSERT ALLの動作
- SQLモデリングでのNULLおよび欠損セルの処理
- SQLモデリングでの参照モデルについて

22.2.1 SQLモデリングの例のベース・スキーマ

この章の例は、shサンプル・スキーマから導出される次のビューsales_viewをベースにしています。

```
CREATE VIEW sales_view AS
SELECT country_name country, prod_name product, calendar_year year,
  SUM(amount_sold) sales, COUNT(amount_sold) cnt,
  MAX(calendar_year) KEEP (DENSE_RANK FIRST ORDER BY SUM(amount_sold) DESC)
  OVER (PARTITION BY country_name, prod_name) best_year,
  MAX(calendar_year) KEEP (DENSE_RANK LAST ORDER BY SUM(amount_sold) DESC)
  OVER (PARTITION BY country_name, prod_name) worst_year
FROM sales, times, customers, countries, products
WHERE sales.time_id = times.time_id AND sales.prod_id = products.prod_id AND
  sales.cust_id =customers.cust_id AND customers.country_id=countries.country_id
GROUP BY country_name, prod_name, calendar_year;
```

この問合せでは、country、productおよびyearでグルーピングされた売上データのSUMおよびCOUNT集計を計算します。また、対象国で各製品の売上高が最も高かった年をレポートします。これを製品のbest_yearと呼びます。また、worst_yearは、売上高が最も低かった年を示します。

22.2.2 MODEL句の構文

MODEL句を使用すると、SQL問合せブロックでデータの多次元計算を定義できます。多次元アプリケーションでは、ファクト表は、従属メジャーまたは属性として機能する列とともに各行を一意に識別する列から構成されます。MODEL句では、多次元配列を定義するPARTITION、DIMENSIONおよびMEASUREの各列、この多次元配列を処理するルール、および処理オプションを指定できます。

MODEL句は、パーティション内の配列計算を表す更新のリストを含み、SQL問合せブロックの一部となります。この構造は次のとおりです。

```
MODEL
[<global reference options>]
[<reference models>]
[MAIN <main-name>]
  [PARTITION BY (<cols>)]
  DIMENSION BY (<cols>)
  MEASURES (<cols>)
  [<reference options>]
  [RULES]  <rule options>
  (<rule>, <rule>,..., <rule>)
  <global reference options> ::= <reference options> <ret-opt>
   <ret-opt> ::= RETURN {ALL|UPDATED} ROWS
  <reference options> ::=
  [IGNORE NAV | [KEEP NAV]
  [UNIQUE DIMENSION | UNIQUE SINGLE REFERENCE]
  <rule options> ::=
  [UPDATE | UPSERT | UPSERT ALL]
```

```
  [AUTOMATIC ORDER | SEQUENTIAL ORDER]
  [ITERATE (<number>)  [UNTIL <condition>]]
  <reference models> ::= REFERENCE ON <ref-name> ON (<query>)
  DIMENSION BY (<cols>) MEASURES (<cols>) <reference options>
```

各ルールは割当てを表します。ルールの左辺は、セルまたはセルの集合を参照します。ルールの右辺には、定数、ホスト変数、個々のセルまたはセルの範囲による集計を含む式を指定できます。たとえば、例22-1の問合せについて考えてみます。これはSQLモデリングの例のベース・スキーマで説明されているとおりに作成されたビューsales_viewに基づいています。

例22-1 MODEL句を使用した単純な問合せ

```
SELECT SUBSTR(country,1,20) country, SUBSTR(product,1,15) product, year, sales
FROM sales_view
WHERE country in ('Italy', 'Japan')
MODEL
  RETURN UPDATED ROWS
  MAIN simple_model
  PARTITION BY (country)
  DIMENSION BY (product, year)
  MEASURES (sales)
  RULES
   (sales['Bounce', 2001] = 1000,
    sales['Bounce', 2002] = sales['Bounce', 2001] + sales['Bounce', 2000],
    sales['Y Box', 2002] = sales['Y Box', 2001])
ORDER BY country, product, year;
```

この問合せでは、国ItalyおよびJapanのsales_viewの行についてモデル計算を定義しています。このモデルには、simple_modelという名前が付いています。countryでデータをパーティション化し、各パーティション内にproductとyearで2次元配列を定義しています。この配列の各セルは、salesメジャーの値を保持します。このモデルの最初のルールでは、2001年のBounceの売上を1000に設定しています。残りの2つのルールでは、2002年のBounceの売上は2001年と2000年の売上の合計であり、2002年のY Boxの売上は前年の売上と同じであることを定義しています。

RETURN UPDATED ROWSを指定すると、前述の問合せは、モデル計算によって更新または挿入された行のみを戻します。デフォルトのRETURN ALL ROWSを使用した場合、MODEL句によって更新または挿入された行のみでなく、すべての行を取得します。問合せの出力は次のようになります。

```
COUNTRY              PRODUCT              YEAR      SALES
-------------------- -------------------- ---------- ----------
Italy                Bounce               2001        1000
Italy                Bounce               2002     5333.69
Italy                Y Box                2002    81207.55
Japan                Bounce               2001        1000
Japan                Bounce               2002     6133.53
Japan                Y Box                2002    89634.83
```

MODEL句は、データベース表に対して行の更新または挿入を行っていないことに注目してください。次の問合せで、sales_viewが変更されていないことを示し、このことを確認します。

```
SELECT SUBSTR(country,1,20) country, SUBSTR(product,1,15) product, year, sales
FROM sales_view
WHERE country IN ('Italy', 'Japan');

COUNTRY              PRODUCT              YEAR      SALES
-------------------- -------------------- ---------- ----------
Italy                Bounce               1999     2474.78
Italy                Bounce               2000     4333.69
Italy                Bounce               2001      4846.3
...
```

MODEL句で行った2001年のBounceの売上値の更新は、データベースに反映されていません。データベース表の行を更新または挿入するには、INSERT、UPDATEまたはMERGE文を使用する必要があります。

前述の例では、PARTITION BY、DIMENSION BYおよびMEASURESリストで列を指定しています。パーティション・キー、ディメンション・キーおよびメジャーとして、定数、ホスト変数、単一行関数、集計関数、分析関数またはこれらを含む式を指定することもできます。ただし、これらはPARTITION BY、DIMENSION BYおよびMEASURESリスト内で別名化する必要があります。ルール、SELECTリストおよびORDER BY問合せ内の式を参照するには、別名を使用する必要があります。次の例では、式および別名の使用方法を示します。

```
SELECT country, p product, year, sales, profits
FROM sales_view
WHERE country IN ('Italy', 'Japan')
MODEL
  RETURN UPDATED ROWS
  PARTITION BY (SUBSTR(country,1,20) AS country)
  DIMENSION BY (product AS p, year)
  MEASURES (sales, 0 AS profits)
  RULES
   (profits['Bounce', 2001] = sales['Bounce', 2001] * 0.25,
    sales['Bounce', 2002] = sales['Bounce', 2001] + sales['Bounce', 2000],
    profits['Bounce', 2002] = sales['Bounce', 2002] * 0.35)
ORDER BY country, year;

COUNTRY    PRODUCT     YEAR    SALES          PROFITS
-------    ---------   ----    --------      --------
Italy      Bounce      2001      4846.3      1211.575
Italy      Bounce      2002     9179.99     3212.9965
Japan      Bounce      2001      6303.6        1575.9
Japan      Bounce      2002    11437.13     4002.9955
```

別名「0 AS profits」は、profitsメジャーのすべてのセルを0に初期化します。MODEL句の構文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

22.2.3 SQLモデリングのキーワード

この項では、SQLモデリングで使用されるキーワードの定義について説明します。次の項目が含まれます。

- 値の割当てとNULLの処理
- 計算定義

22.2.3.1 値の割当てとNULLの処理

- UPSERT

  既存のセルのメジャー値を更新します。目的のセルが存在せず、なおかつルールが適切な表記法を使用している場合は、そのセルが挿入されます。セル参照にシンボリック参照が1つでも含まれる場合、セルの挿入は行われません。

- UPSERT ALL

  UPSERTと似ていますが、より幅広いルール表記法で新しいセルを挿入できる点が異なります。

- UPDATE

  既存のセルの値を更新します。セルの値が存在しない場合、更新は行われません。

- IGNORE NAV

  数値セルの場合、使用できない値を0として処理します。つまり、問合せ結果セットによってMODEL句に指定されていないセルは、計算でゼロとして処理されます。これは、モデルのすべてのメジャーに対してグローバル・レベルで使用できます。

- KEEP NAV

  使用できないセルの値を変更せずにそのまま維持します。これは、グローバル・レベルでIGNORE NAVを指定した場合に例外を設けるのに役立ちます。これはデフォルトなので、省略可能です。

22.2.3.2 計算定義

- MEASURES

  モデルで変更または作成される値の集合です。

- RULES

  値をメジャーに割り当てる式です。

- AUTOMATIC ORDER

  すべてのルールが、論理的な依存関係に基づいた順序で評価されます。

- SEQUENTIAL ORDER

  ルールが、記述された順序で評価されます。これはデフォルトです。

- UNIQUE DIMENSION

  これがデフォルトです。MODEL句のPARTITION BYおよびDIMENSION BYの列の組合せが、モデル内のあらゆるセルを一意に識別する必要があることを意味します。この一意性は、必要に応じて、問合せの実行時に明示的に検証されます。その場合、処理速度低下の原因となることがあります。

- UNIQUE SINGLE REFERENCE

  PARTITION BYおよびDIMENSION BY句は、ルールの右辺のシングル・ポイント参照を一意に識別します。これは、問合せの実行時に一意性の明示的なチェックを行わないため、処理時間を短縮できる場合があります。

- RETURN [ALL|UPDATED] ROWS

  選択した行すべてを戻すか、ルールによって更新された行のみを戻すかを指定できます。デフォルト値は、ALLです。UPDATED ROWSも指定可能です。

22.2.4 SQLモデリングでのセル参照について

MODEL句では、リレーションはセルの多次元配列として扱われます。多次元配列のセルにはメジャー値が含まれ、このセルは、PARTITION BYキーによって定義された各パーティション内でDIMENSION BYキーを使用して索引付けされます。たとえば、SQLモデリングの例のベース・スキーマの説明に従って作成されたビューsales_viewで実行される次の問合せを考えてみます。

```
SELECT country, product, year, sales, best_year, best_year
FROM sales_view
MODEL
  PARTITION BY (country)
  DIMENSION BY (product, year)
  MEASURES (sales, best_year)
  (<rules> ..)
ORDER BY country, product, year;
```

この例では、countryでデータをパーティション化し、各パーティション内にproductとyearで2次元配列を定義しています。この配列のセルには、salesおよびbest_yearという2つのメジャーが格納されます。

DIMENSION BYキーを指定してセルのメジャー値にアクセスすることを、セル参照といいます。セル参照の例を次に示します。

```
sales[product= 'Bounce', year=2000]
```

この例では、製品Bounceおよび年2000で参照されるセルのsales値にアクセスしています。セル参照では、DIMENSION BY

キーを、前述のセル参照のようにシンボリックに指定するか、sales['Bounce', 2000]のように位置ベースで指定できます。

この項では、次の項目について説明します。

- シンボリック・ディメンション参照
- 位置ベースのディメンション参照

22.2.4.1 シンボリック・ディメンション参照

シンボリック・ディメンション参照(シンボリック参照)では、DIMENSION BYキーの値をブール式で指定します。たとえば、セル参照sales[year >= 2001]の場合、DIMENSION BYキーyearのシンボリック参照があり、yearの値が2001以上のセルをすべて指定します。また、sales[product = 'Bounce', year >= 2001]も、productおよびyearディメンションのシンボリック参照の一例です。

22.2.4.2 位置ベースのディメンション参照

位置ベースのディメンション参照(位置参照)は、定数、またはディメンションに対して指定された定数式です。たとえば、セル参照sales['Bounce']の場合、productディメンションの位置参照があり、製品Bounceのsales値にアクセスします。セル参照の定数(または定数式)は、DIMENSION BYキーに対して指定された列の順序と比較されます。次に、ディメンションの位置参照の使用例を示します。

```
sales['Bounce', 2001]
```

DIMENSION BYキーがproductとyearで両者の順序が同じ場合、Bounceと2001のsales値にアクセスします。

指定方法に応じて、セル参照はシングル・セル参照とマルチセル参照に分類されます。

22.2.5 SQLモデリングのルールについて

モデル計算はルールで記述します。このルールに基づいて、PARTITION BY、DIMENSION BYおよびMEASURES句で定義された多次元配列のセルの操作が行われます。ルールは、左辺が1つのセルまたはセルの範囲を表し、右辺が定数、バインド変数、個々のセル、セルの範囲の集計関数を含む式になっている代入文です。ルールでは、表現力を最大限に高めるためにワイルド・カードおよびループ・コンストラクトを使用できます。次に、ルールの例を示します。

```
sales['Bounce', 2003] = 1.2 * sales['Bounce', 2002]
```

このルールは、製品Bounceの2003年の売上は2002年の売上より20%増加するということを表しています。

このルールは、左辺と右辺がともにシングル・セル参照であるため比較的単純です。マルチセル参照、集計、ネステッド・セル参照を使用した複雑なルールも記述できます。

シングル・セル参照

このタイプのルールには、左辺に定数によるシングル・セル参照があり、右辺にシングル・セル参照があるものが含まれます。次に、例をいくつか示します。

```
sales[product='Finding Fido', year=2003] = 100000
sales['Bounce', 2003] = 1.2 * sales['Bounce', 2002]
sales[product='Finding Fido', year=2004] = 0.8 * sales['Standard Mouse Pad',
  year=2003] + sales['Finding Fido', 2003]
```

右辺のマルチセル参照

マルチセル参照は、ルールの右辺で使用できます。この場合、集計関数を適用して単一値に変換する必要があります。分析集計関数(逆パーセンタイル関数、仮説ランク関数、仮説分布関数など)や統計集計関数(CORRELATION関数、REGR_SLOPE関数など)など既存のすべての集計関数、およびユーザー定義の集計関数を使用できます。RANKや

MOVING_AVGなどのウィンドウ関数も使用可能です。たとえば、次のルールでは、2003年のBounceの売上は、1998年から2002年の期間における最高の売上より100増加すると計算します。

```
sales['Bounce', 2003] = 100 + MAX(sales)['Bounce', year BETWEEN 1998 AND 2002]
```

次に、逆パーセンタイル関数PERCENTILE_DISCの使用例を示します。ここでは、2003年のFinding Fidoの売上高は、2003年より前のすべての年におけるFinding Fido、Standard Mouse PadおよびBoatの売上の中央値より30%増加すると想定しています。

```
sales[product='Finding Fido', year=2003] = 1.3 *
  PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY sales) [product IN ('Finding
  Fido','Standard Mouse Pad','Boat'), year < 2003]
```

集計関数は、ルールの右辺でのみ使用できます。集計関数の引数には、MODEL句のメジャー、バインド変数、定数、またはこれらを含む式を指定できます。たとえば、次のルールでは、2003年のBounceの売上は、1998年から2002年の売上の加重平均になると計算します。

```
sales['Bounce', 2003] =
   AVG(sales * weight)['Bounce', year BETWEEN 1998 AND 2002]
```

左辺のマルチセル参照

ルールでは、次のように左辺にマルチセル参照を指定できます。

```
sales['Standard Mouse Pad', year > 2000] =
   0.2 * sales['Finding Fido', year=2000]
```

このルールでは、左辺のセルの範囲(製品Standard Mouse Padのセルおよび2001年以降のyearのセル)にアクセスして、これらのセルのsalesメジャーを右辺の式で計算される値に割り当てます。前述のルールによる計算は、「2001年以降のStandard Mouse Padの売上は、2000年のFinding Fidoの売上の20%になる」ということを表しています。この計算は、右辺のセル参照(右辺の式)が左辺で参照されるすべてのセルと同じである点で単純です。

CV関数の使用

CV関数を使用すると相対索引付けが可能です。相対索引付けでは、左辺で参照されるセルのディメンション値が右辺のセル参照で使用されます。CV関数は、引数にディメンション・キーを取り、左辺で現在参照されているセルのDIMENSION BYキーの値を提供します。次の例を考えてみます。

```
sales[product='Standard Mouse Pad', year>2000] =
  sales[CV(product), CV(year)] + 0.2 * sales['Finding Fido', 2000]
```

左辺がセルStandard Mouse Padおよび2001を参照する場合、右辺の式は次のようになります。

```
sales['Standard Mouse Pad', 2001] + 0.2 * sales['Finding Fido', 2000]
```

同様に、左辺がセルStandard Mouse Padおよび2002を参照する場合、右辺の式は次のようになります。

```
sales['Standard Mouse Pad', 2002] + 0.2 * sales['Finding Fido', 2000]
```

また、CV()のように引数を指定せずにCVを使用することもできます。この場合、位置参照となります。CV()は、セル参照の外部で使用することもできます。ただし、この場合は、引数に対象のディメンション名を指定する必要があります。前述のルールは、次のように記述することもできます。

```
sales[product='Standard Mouse Pad', year>2000] =
  sales[CV(), CV()] + 0.2 * sales['Finding Fido', 2000]
```

最初のCV()参照はCV(product)に対応し、2番目の参照はCV(year)に対応します。CV関数は、右辺のセル参照でのみ使

用できます。別のCV関数の使用例を次に示します。

```
sales[product IN ('Finding Fido','Standard Mouse Pad','Bounce'), year
  BETWEEN 2002 AND 2004] = 2 * sales[CV(product), CV(year)-10]
```

このルールは、「Finding Fido、Standard Mouse Pad、Bounceの各製品について、2002年から2004年の売上は10年前の2倍になる」という意味です。

ANYワイルド・カードの使用

セル参照でワイルド・カードANYを使用すると、NULLを含むすべてのディメンション値に一致させることができます。ANYは、ルールの左辺と右辺の両方で使用できます。たとえば、「2003年の全製品の売上は、2002年の売上より10%増加する」という計算を行う場合、ルールは次のようになります。

```
sales[product IS ANY, 2003] = 1.1 * sales[CV(product), 2002]
```

位置参照を使用する場合は、次のようになります。

```
sales[ANY, 2003] = 1.1 * sales[CV(), 2002]
```

ANYは、位置ベースで指定した場合でも、シンボリック参照として扱われます。これは、ANYの実際の意味が(dimension IS NOT NULL OR dimension IS NULL)であるためです。

ネステッド・セル参照

セル参照はネストできます。つまり、ディメンション値を指定するセル参照をセル参照内で使用できます。たとえば、ネステッド・セル参照のメジャーがbest_yearの場合、次のようになります。

```
sales[product='Bounce', year = best_year['Bounce', 2003]]
```

ネステッド・セル参照best_year['Bounce', 2003]は、ディメンション・キーyearに値を提供し、yearのシンボリック参照で使用されます。メジャーbest_yearおよびworst_yearは、年(y)と製品(p)の組合せごとに、製品pの売上が最高または最低だった年を示します。次のルールでは、2003年のStandard Mouse Padの売上高が、Finding Fidoの売上が最高および最低だった年のStandard Mouse Padの平均売上高となるものとして計算します。

```
sales['Standard Mouse Pad', 2003] = (sales[CV(), best_year['Finding Fido',
  CV(year)]] + sales[CV(), worst_year['Finding Fido', CV(year)]]) / 2
```

Oracle Databaseで許容されるネストのレベルは1つのみです。また、ネステッド・セル参照として使用できるのは、シングル・セル参照のみです。マルチセル参照の集計は、ネステッド・セル参照で使用できません。

22.2.6 SQLモデリング・ルールの評価順序

デフォルトでは、ルールはMODEL句内での出現順に評価されます。MODEL句でオプションのキーワードSEQUENTIAL ORDERを指定すると、評価順序を明示できます。ルールの評価順序が順次的になっているSQLモデルを、順次順序モデルと呼びます。たとえば、次のRULESの指定では、指定した順序でルールが評価されます。

```
RULES SEQUENTIAL ORDER
 (sales['Bounce', 2001] =
    sales['Bounce', 2000] + sales['Bounce', 1999],    --Rule R1
  sales['Bounce', 2000] = 50000,                      --Rule R2
  sales['Bounce', 1999] = 40000)                      --Rule R3
```

また、AUTOMATIC ORDERオプションを使用すると、Oracle Databaseによってルールの評価順序が自動的に決定されます。Oracleは、各ルール内のセル参照を調べ、ルール間の依存関係を確認します。ルールR1の左辺で参照されるセルが、別のルールR2の右辺で参照される場合、R2はR1に依存しているとみなされます。したがって、ルールR1は、ルールR2より先に評価され

る必要があります。前述の例にAUTOMATIC ORDERを指定すると、次のようになります。

```
RULES AUTOMATIC ORDER
 (sales['Bounce', 2001] = sales['Bounce', 2000] + sales['Bounce', 1999],
  sales['Bounce', 2000] = 50000,
  sales['Bounce', 1999] = 40000)
```

ルール2とルール3は、ルール1より先に任意の順序で評価されます。これは、ルール1がルール2とルール3に依存しており、ルール2とルール3より後に評価される必要があるためです。ルール2とルール3は相互に依存していないため、評価順序は任意になります。相互に独立したルールの場合、任意の順序で評価できます。前述の例のように、評価順序が自動になっているSQLモデルを自動順序モデルと呼びます。

自動順序モデルの場合、同一セルへの複数の割当ては禁止されています。つまり、セルのメジャーを割り当てることができるのは1回のみです。結果が不確定的となる場合は、エラーが戻されます。たとえば、次のルール指定の場合、sales['Bounce', 2001]が複数回割り当てられているため、エラーが生成されます。

```
RULES AUTOMATIC ORDER
 (sales['Bounce', 2001] = sales['Bounce', 2000] + sales['Bounce', 1999],
  sales['Bounce', 2001] = 50000,
  sales['Bounce', 2001] = 40000)
```

2001年の製品Bounceの売上を割り当てるルールは相互に依存していないため、ルール間で特定の評価順序が決まりません。この場合、sales['Bounce', 2001]は、40000、50000、1999年と2000年のBounceの合計売上のいずれにもなり得るため、評価順序が無原則となり、不確定的な結果になります。Oracle Databaseでは、これを防止するために、AUTOMATIC ORDERが指定された場合は複数の割当てを禁止しています。ただし、順次順序モデルでは、複数の割当ては可能です。前述の例でAUTOMATIC ORDERのかわりにSEQUENTIAL ORDERを指定した場合、sales['Bounce', 2001]の結果は40000になります。

22.2.7 SQLモデリング・ルールのグローバルおよびローカル・キーワード

UPDATE、UPSERT、UPSERT ALL、IGNORE NAVおよびKEEP NAVオプションは、RULES句の中でグローバル・レベルで指定できます。この場合、すべてのルールが指定のモードで処理されます。これらのオプションは、ルールごとにローカル・レベルで指定することもできます。この場合、グローバルな動作はオーバーライドされます。たとえば、次の指定を考えてみます。

```
RULES UPDATE
(UPDATE s['Bounce',2001] = sales['Bounce',2000] + sales['Bounce',1999],
 UPSERT s['Y Box', 2001] = sales['Y Box', 2000] + sales['Y Box', 1999],
   sales['Mouse Pad', 2001] = sales['Mouse Pad', 2000] +
   sales['Mouse Pad',1999])
```

UPDATEオプションをグローバル・レベルで指定しているため、最初と3番目のルールは更新モードで処理されます。2番目のルールは、UPSERTキーワードを指定しているため、アップサート・モードで処理されます。3番目のルールは、オプションが未指定のため、グローバル・オプションの更新動作を継承します。

22.2.8 UPDATE、UPSERTおよびUPSERT ALLの動作

UPDATE、UPSERTまたはUPSERT ALLのいずれかを選択することで、ルールで指定されたセルの処理方法を指定できます。MODEL句のルールは、デフォルトでUPSERTとして処理されますが、アップサートであることを明確にするために、明示的にUPSERTキーワードを指定してもかまいません。

次の各項で、これら3つのオプションの動作について説明します。

- UPDATEの動作

- UPSERTの動作
- UPSERT ALLの動作

22.2.8.1 UPDATEの動作

UPDATEオプションを指定すると、完全な更新モードが強制されます。このモードでは、左辺の参照先のセルが存在しない場合、ルールは無視されます。ルールの左辺で参照されるセルが存在する場合、そのメジャーが右辺の式の値で更新されます。それ以外の場合でセル参照が位置ベースの場合は、右辺の式の値に等しいメジャー値で新しいセルが作成され、多次元配列に挿入されます。セル参照が位置ベースでない場合、セルは挿入されません。セルの指定にシンボリック参照が1つでも含まれる場合は、アップサートのルールでも挿入はできないことに注意してください。たとえば、次のルールがあるとします。

```
sales['Bounce', 2003] = sales['Bounce', 2001] + sales ['Bounce', 2002]
```

製品Bounceと2003年のセルが存在する場合、2001年と2002年のBounceの合計売上でセルが更新され、存在しない場合はセルが作成されます。次のように、シンボリック参照を使用して同じルールを作成した場合、更新は行われません。

```
sales[prod= 'Bounce', year= 2003] = sales['Bounce', 2001] + sales ['Bounce', 2002]
```

22.2.8.2 UPSERTの動作

UPSERTを使用すると、セルが存在せず、セル参照に含まれているのが定数で修飾された位置参照のみである場合に、ルールの左辺で参照されているセルに対応する新しいセルが作成されます。FORループで作成されるセル参照(SQLモデリングの高度なトピックを参照)は、位置参照として扱われることに注意してください。このため、FORループで作成される値は、新しいセルの挿入に使用されます。たとえば、2004年以降の年のセルがないとします。この場合、次のルールを考えてみます。

```
UPSERT sales['Bounce', year = 2004] = 1.1 * sales['Bounce', 2002]
```

この場合、シンボリック参照year = 2004があるため、新しいセルは作成されません。ただし、次の点を考慮してください。

```
UPSERT sales['Bounce', 2004] = 1.1 * sales['Bounce', 2002]
```

この場合、2004年の製品Bounceのセルが新規作成されます。ただし、いずれかの参照がANYの場合、新しいセルは作成されません。これは、ANYが、NULLを含むすべてのディメンション値を許可する述語であるためです。たとえば、ディメンションdに対する参照ANYは、(d IS NOT NULL OR d IS NULL)という述語と同じ意味になります。

UPSERTルールの左辺のセル参照でFORループが使用されている場合、アップサート対象のセルのリストは、各ディメンションのすべての個別値のクロス積を計算することで生成されます。FORループを含むUPSERTを使用してディメンションの稠密化(レポート用のデータの稠密化を参照)を行うことは可能ですが、稠密化にはパーティション外部結合操作を使用するのが一般的です。

22.2.8.3 UPSERT ALLの動作

UPSERT ALLを使用すると、存在述語(比較、IN、ANYなど)を左辺に含むモデル・ルールにUPSERTの動作を適用できます。たとえば次のルールでは、ANYを使用して、San Francisco、San JoseおよびOaklandの組合せであるBay Areaを作成しています。

```
SELECT product, time, city, s sales
FROM cube_subquery
MODEL PARTITION BY (product)
DIMENSION BY (time, city) MEASURES(sales s
RULES UPSERT ALL
(s[ANY, 'Bay Area'] =
   s[CV(), 'San Francisco'] + s[CV(), 'San Jose'] + s[CV(), 'Oakland']
s['2004', ANY] = s['2002', CV()] + s['2003', CV()]);
```

この例の最初のルールでは、単純に、個別の各time値用のBay Areaセルを挿入しています。2番目のルールでは、Bay

Areaを含む個別の各city値用の2004セルを挿入しています。この例は、左辺で使用されている存在述語がANY述語であるという比較的単純なものですが、UPSERT ALLはより複雑な計算で使用することも可能です。

UPSERT ALLを使用する場合(特に、複数のシンボリック・ディメンション参照がある場合)は、その動作を正確に理解しておく必要があります。その動作は、FORループを使用するUPSERTルールとは異なります。

Oracle Databaseでは、UPSERT ALLルールの評価の際、次のステップが実行され、アップサート対象となるセル参照のリストが作成されます。

1. セル参照のすべてのシンボリック述語を満たす既存のセルを検索します。
2. 記号参照があるディメンションのみを使用して、これらのセルの異なるディメンション値の組合せを検索します。
3. これらの値の組合せと、位置参照で指定されたディメンション値とのクロス積を実行します。
4. ステップ3の結果を使用して、新しいセルを配列にアップサートします。

22.2.8.3.1 例: UPSERT ALLの動作

UPSERT ALLの動作で説明している4つのステップを説明するために、3つのディメンションを持つ抽象的なデータおよびモデルを使用した簡単な例を示します。このモデルは、(product, time, city)でディメンション化され、salesというメジャーを持つものとします。この例では、都市zに新しい売上値をアップサートしますが、この売上値は都市yからコピーしてきます。

```
UPSERT ALL sales[ANY, ANY, 'z']= sales[CV(product),CV(time),'y']
```

ソースのデータセットには、次の4つの行があります。

```
PROD    TIME   CITY   SALES
   1    2002      x      10
   1    2003      x      15
   2    2002      y      21
   2    2003      y      24
```

次に、前述の4つのステップを、このデータに当てはめて詳しく説明します。

1. ルールのシンボリック述語はANYなので、前述の行すべてが対象になります。
2. 条件に合致する、シンボリック述語で参照されたセルの重複しないディメンションの組合せは、(1, 2002)、(1, 2003)、(2, 2002)および(2, 2003)です。
3. これらのディメンションの組合せと、位置参照で指定されたセルのクロス積を求めます。この例では、単純に値zとのクロス積となり、結果のセル参照は(1, 2002, z)、(1, 2003, z)、(2, 2002, z)および(2, 2003, z)となります。
4. ステップ3で求めたセルが、都市yに基づいて計算された売上でアップサートされます。都市yの製品1には値がないので、製品1用に作成されたセルの売上値はNULLになります。当然のことながら、ルールが異なる場合には、新しいすべてのセルに非NULLの結果が生成されることもあります。結果セットには、元の4つの行に加えて、次のように新しい4つの行が追加されます。

```
PROD    TIME   CITY    SALES
   1    2002      x       10
   1    2003      x       15
   2    2002      y       21
   2    2003      y       24
   1    2002      z     NULL
   1    2003      z     NULL
   2    2002      z       21
   2    2003      z       24
```

この結果は、すべてのディメンションの値をすべて使用したクロス積ではないことに注意してください。すべてのディメンションの値を

すべて使用した場合は、(1,2002, y)、(2,2003, x)などのセルも生成されます。これは、あくまでも既存の行で見つかったディメンションの組合せを使用して生成された結果です。

22.2.9 SQLモデリングでのNULLおよび欠損セルの処理

モデルを使用するアプリケーションでは、NULLによるセル・メジャーの不確定値のみでなく、欠損セルによる不確定性にも対応する必要があります。シングル・セル参照で参照されるセルがデータ内で欠損している場合、このようなセルを欠損セルと呼びます。MODEL句では、NULLのセルおよび欠損セルについてANSI SQL規格に準拠したデフォルトの処理方法があり、ビジネス・ロジックに応じて適切な方法でこうしたセルを処理するオプションも用意されています。たとえば、NULLを算術演算でゼロとして処理するオプションなどがあります。

デフォルトでは、NULLのセル・メジャー値は、SQLの他の箇所と同じ方法で処理されます。たとえば、次のルールがあるとします。

```
sales['Bounce', 2001] = sales['Bounce', 1999] + sales['Bounce', 2000]
```

1999年と2000年のBounceの売上値のいずれかがNULLの場合、右辺の式はNULLと評価されます。同様に、ルール内の集計関数でも、通常の動作と同じ方法でNULL値を処理します。つまり、NULL値は集計処理時に無視されます。

欠損セルは、NULLメジャー値を含むセルとして処理されます。たとえば、前述のルールでは、Bounceと2000のセルがない場合、NULL値として処理され、右辺の式はNULLと評価されます。

この項では、次の項目について説明します。

- 欠損セルとNULLの区別
- 欠損セルおよびNULLのデフォルト値の使用
- セル参照でのNULLの使用

22.2.9.1 欠損セルとNULLの区別

PRESENTV関数およびPRESENTNNV関数を使用すると、欠損セルの識別およびNULL値との区別が可能です。これらの関数の引数には、PRESENTV(cell, expr1, expr2)のようにシングル・セル参照および2つの式を指定します。PRESENTV関数は、セルcellがMODEL句に入力されたデータに存在する場合、最初の式expr1を戻します。それ以外の場合は、2番目の式expr2を戻します。次の例を考えてみます。

```
PRESENTV(sales['Bounce', 2000], 1.1*sales['Bounce', 2000], 100)
```

製品Bounceおよび2000年のセルが存在する場合、対応する売上値に1.1を掛けて戻します。これに該当しない場合は100を戻します。製品Bounceおよび2000年のセルがNULLの場合、前述の指定ではNULLが戻されます。

PRESENTNNV関数は、セルの有無のチェックに加えて、セルがNULLかどうかのチェックも行います。セルが存在してNULLでない場合は最初の式expr1を、それ以外の場合は2番目の式expr2を戻します。次の例を考えてみます。

```
PRESENTNNV(sales['Bounce', 2000], 1.1*sales['Bounce', 2000], 100)
```

この例では、sales['Bounce', 2000]が存在し、NULLでない場合、1.1*sales['Bounce', 2000]を戻します。それ以外の場合は100を戻します。

アプリケーションでは、モデルにIS PRESENT述語を使用して、セルの有無を明示的にチェックできます。この述語は、セルが存在する場合はTRUEを、存在しない場合はFALSEを戻します。PRESENTNNVを使用した前述の例を、IS PRESENTを使用して記述すると次のようになります。

```
CASE WHEN sales['Bounce', 2000] IS PRESENT AND sales['Bounce', 2000] IS NOT NULL
THEN 1.1 * sales['Bounce', 2000]
ELSE 100
END
```

PRESENTVおよびPRESENTNNV関数と同様に、IS PRESENT述語は、入力データ(MODEL句の実行前に存在していたデータ)にセルが存在するかどうかをチェックします。これを使用すると、UPSERTルールで新規に挿入されたセルの複数のメジャーを初期化できます。たとえば、製品Bounceと2003年のセルがデータに存在しない場合、セルのsales値とprofit値をそれぞれ1000と500に初期化するには、次のように記述します。

```
RULES
 (UPSERT sales['Bounce', 2003] =
    PRESENTV(sales['Bounce', 2003], sales['Bounce', 2003], 1000),
  UPSERT profit['Bounce', 2003] =
    PRESENTV(profit['Bounce', 2003], profit['Bounce', 2003], 500))
```

この例で使用されているPRESENTV関数は、入力データ内のセルの有無に応じて、TRUEまたはFALSEを戻します。評価順序に基づいて一方のルールでBounceと2003のセルが挿入される場合でも、他方のルールのPRESENTV関数は、引き続きFALSEと評価します。この動作は、ルール評価の事前処理ステップとみなすことができます。このステップでは、PRESENTV関数、PRESENTNNV関数およびIS PRESENT述語のすべてを評価し、対応する値ですべて置き換えます。

22.2.9.2 欠損セルおよびNULLのデフォルト値の使用

MODEL句では、デフォルトで欠損セルを、NULLメジャー値を含むセルとして処理します。この動作にするには、オプションのKEEP NAVキーワードをMODEL句で指定します。アプリケーションのデフォルトとして、欠損セルおよびNULLをなんらかの値にするには、IS PRESENT述語、IS NULL述語、PRESENTV関数およびPRESENTNNV関数を使用します。ただし、シングル・セル参照およびルールが多い場合は、処理が複雑になる場合があります。デフォルトのKEEP NAVオプションのかわりに、IGNORE NAVオプションを使用すると、NULLおよび欠損セルを次のデフォルト値に設定できます。

- 数値データの場合は0
- 文字/文字列データの場合は空の文字列
- 日付型のデータの場合は01-JAN-2001
- 前述以外のデータ型の場合はNULL

次の問合せについて考えてみます。

```
SELECT product, year, sales
FROM sales_view
WHERE country = 'Poland'
MODEL
  DIMENSION BY (product, year) MEASURES (sales sales) IGNORE NAV
  RULES UPSERT
  (sales['Bounce', 2003] = sales['Bounce', 2002] + sales['Bounce', 2001]);
```

この例では、MODEL句の入力データに、製品Bounceおよび2002年のセルがありません。IGNORE NAVオプションが指定されていることにより、sales['Bounce', 2002]のデフォルト値はNULLではなく、0になります(salesが数値型のため)。したがって、sales['Bounce', 2003]の値は、sales['Bounce', 2001]の値と同じになります。

22.2.9.3 セル参照でのNULLの使用

セル参照でNULL値を使用するには、次のいずれかの方法を使用します。

- ワイルド・カードANYを使用した位置参照(たとえば、sales[ANY])
- IS ANY述語を使用したシンボリック参照(たとえば、sales[product IS ANY])
- NULLの位置参照(たとえば、sales[NULL])
- IS NULL述語を使用したシンボリック参照(たとえば、sales[product IS NULL])

シンボリック参照sales[product = NULL]では、productディメンションのNULLのチェックは行われません。この場合の動作は、SQLによるNULLの標準の取扱いに準拠します。

22.2.10 SQLモデリングの参照モデルについて

ルールが処理される多次元配列の他に、1つ以上の読取り専用の多次元配列があります。前者は、メイン・モデル、後者は参照モデルと呼ばれます。参照モデルは、MODEL句で作成および参照でき、メイン・モデルの参照表のように機能します。参照モデルは、メイン・モデルと同様に問合せブロックで定義されます。また、参照モデルには、ディメンションとメジャーを示すDIMENSION BY句およびMEASURES句があります。参照モデルは、次の副次句で作成されます。

```
REFERENCE model_name ON (query) DIMENSION BY (cols) MEASURES (cols)
   [reference options]
```

メイン・モデルと同様、参照モデルの多次元配列は、ルールの評価前に作成されます。ただし、メイン・モデルとは異なり、参照モデルは読取り専用のため、作成した後にセルの更新や新しいセルの挿入を行うことはできません。したがって、メイン・モデルのルールは、参照モデルに対して、セルへのアクセスは可能ですが、セルの更新や新しいセルの挿入はできません。次に、参照モデルとして通貨の換算表を使用した例を示します。

```
CREATE TABLE dollar_conv_tbl(country VARCHAR2(30), exchange_rate NUMBER);
INSERT INTO dollar_conv_tbl VALUES('Poland', 0.25);
INSERT INTO dollar_conv_tbl VALUES('France', 0.14);
...
```

2003年のPolandとFranceの算出売上をUSドルに換算するには、次のコマンドのようにドル換算表を参照モデルとして使用できます。ビューsales_viewは、SQLモデリングの例のベース・スキーマの説明に従って作成されたものです。

```
SELECT country, year, sales, dollar_sales
FROM sales_view
GROUP BY country, year
MODEL
  REFERENCE conv_ref ON (SELECT country, exchange_rate FROM dollar_conv_tbl)
    DIMENSION BY (country) MEASURES (exchange_rate) IGNORE NAV
  MAIN conversion
    DIMENSION BY (country, year)
    MEASURES (SUM(sales) sales, SUM(sales) dollar_sales) IGNORE NAV
RULES
(dollar_sales['France', 2003] = sales[CV(country), 2002] * 1.02 *
   conv_ref.exchange_rate['France'],
   dollar_sales['Poland', 2003] =
      sales['Poland', 2002] * 1.05 * exchange_rate['Poland']);
```

この例の要点を次に示します。

- 1つのディメンション参照モデルconv_refが表dollar_conv_tblの行に作成され、そのメジャーexchange_rateは、メイン・モデルのルールで参照されています。
- メイン・モデルconversionには、countryとyearの2つのディメンションがあり、参照モデルconv_refには、ディメンションcountryが1つあります。
- 参照モデルのメジャーexchange_rateへのアクセス方法に違いがあります。Franceの場合、model_name.measure_nameの表記法に準拠したconv_ref.exchange_rateを使用しているため明示的です。Polandの場合、単純なmeasure_nameの表記法に準拠した参照exchange_rateを使用しています。メイン・モデルと参照モデル間での列名の曖昧性を解決するには、前者の表記法を使用する必要があります。

次の例では、増加率がルールでハードコード化されています。Franceの増加率は2%、Polandは5%です。ただし、これらは別個の表に存在するため、各表の上位に参照モデルを定義できます。表growth_rate(country, year, rate)を次のよう

に定義するとします。

```
CREATE TABLE growth_rate_tbl(country VARCHAR2(30),
   year NUMBER, growth_rate NUMBER);
INSERT INTO growth_rate_tbl VALUES('Poland', 2002, 2.5);
INSERT INTO growth_rate_tbl VALUES('Poland', 2003, 5);
...
INSERT INTO growth_rate_tbl VALUES('France', 2002, 3);
INSERT INTO growth_rate_tbl VALUES('France', 2003, 2.5);
```

次の問合せでは、すべての国を対象とした2003年のドルでの算出売上高を計算します。

```
SELECT country, year, sales, dollar_sales
FROM sales_view
GROUP BY country, year
MODEL
  REFERENCE conv_ref ON
          (SELECT country, exchange_rate FROM dollar_conv_tbl)
           DIMENSION BY (country c) MEASURES (exchange_rate) IGNORE NAV
  REFERENCE growth_ref ON
          (SELECT country, year, growth_rate FROM growth_rate_tbl)
           DIMENSION BY (country c, year y) MEASURES (growth_rate) IGNORE NAV
  MAIN projection
   DIMENSION BY (country, year) MEASURES (SUM(sales) sales, 0 dollar_sales)
  IGNORE NAV
  RULES
  (dollar_sales[ANY, 2003] = sales[CV(country), 2002] *
   growth_rate[CV(country), CV(year)] *
   exchange_rate[CV(country)]);
```

この問合せでは、異なるディメンションのオブジェクトを処理および関連付ける、MODEL句の機能を示しています。参照モデルconv_refにはディメンションが1つあり、参照モデルgrowth_refとメイン・モデルにはディメンションが2つあります。参照モデルでのシングル・セル参照のディメンションは、CV関数を使用して指定されているので、メイン・モデルのセルと参照モデルが関連付けられます。この指定により、メイン・モデルと参照モデル間でリレーショナル結合が実行されます。

参照モデルは、キーを順序番号に変換し、順序番号を使用した計算(たとえば、減算に前期を使用する場合など)を実行してから、順序番号をキーに再変換する場合にも有効です。たとえば、順序番号を年に割り当てる次のビューがあるとします。

```
CREATE or REPLACE VIEW year_2_seq (i, year) AS
SELECT ROW_NUMBER() OVER (ORDER BY calendar_year), calendar_year
FROM (SELECT DISTINCT calendar_year FROM TIMES);
```

このビューで、参照表を2つ定義できます。1つは、整数から年に変換するi2yです。これは、順序番号を整数にマップします。もう1つは、年から整数に変換するy2iで、逆のマッピングを行います。参照y2i.i[year]およびy2i.i[year] - 1は、それぞれ今年と昨年の順序番号を戻します。参照i2y.y[y2i.i[year]-1]は、昨年のyearキーの値を戻します。次の問合せでは、参照モデルのこのような使用方法を示します。

```
SELECT country, product, year, sales, prior_period
FROM sales_view
MODEL
  REFERENCE y2i ON (SELECT year, i FROM year_2_seq) DIMENSION BY (year y)
   MEASURES (i)
  REFERENCE i2y ON (SELECT year, i FROM year_2_seq) DIMENSION BY (i)
   MEASURES (year y)
  MAIN projection2 PARTITION BY (country)
  DIMENSION BY (product, year)
  MEASURES (sales, CAST(NULL AS NUMBER) prior_period)
(prior_period[ANY, ANY] = sales[CV(product), i2y.y[y2i.i[CV(year)]-1]])
```

```
ORDER BY country, product, year;
```

前述の例では、参照モデルのセル参照がネストされていることがわかります。参照モデルy2iのセル参照は、i2yのセル参照内でネストされ、これがさらにメインSQLモデルのセル参照内でネストされています。参照モデルのセル参照は、無制限のレベルでネストできます。ただし、メインSQLモデルのセル参照の場合は、2つのレベルまでしかネストできません。

最後に、参照モデルの指定および使用方法に関する制限を次に示します。

- 参照モデルでは、PARTITION BY句は指定できません。
- 参照モデルが定義された問合せブロックは、他の問合せと相関させることができません。
- 参照モデルには、一意の名前を付ける必要があります。
- 参照モデルのセルへの参照は、すべてシングル・セル参照である必要があります。

22.3 SQLモデリングの高度なトピック

この項では、SQLモデリングのより高度なトピックについて説明します。

- SQLモデリングでのFORループ
- SQLモデリングでの反復モデル
- AUTOMATIC ORDERモデルでのルールの依存関係
- SQLモデリングでの順序付きルール
- SQLモデリングでの分析関数
- SQLモデリングでのUNIQUE DIMENSIONとUNIQUE SINGLE REFERENCE
- モデリング用SQLを使用する場合の規則および制限事項

22.3.1 SQLモデリングでのFORループ

MODEL句では、ルール内でFORコンストラクトを使用して、計算をよりコンパクトに表現できます。これは、ルールの左辺と右辺の両方で使用できます。FORループは、ルールの左辺に指定された場合は位置参照として扱われます。たとえば、次の計算を考えてみます。この計算では、2004年の製品の売上高は、2003年の売上高より10%増加すると見積ります。

```
RULES UPSERT
(sales['Bounce', 2004] = 1.1 * sales['Bounce', 2003],
 sales['Standard Mouse Pad', 2004] = 1.1 * sales['Standard Mouse Pad', 2003],
...
 sales['Y Box', 2004] = 1.1 * sales['Y Box', 2003])
```

この計算では、UPSERTオプションが使用されているため、これらの製品および2004年のセルが多次元配列にない場合、これらのセルが挿入されます。この場合、製品の数と同じ数のルールを指定する必要があるため、処理が困難になります。次のように、FORコンストラクトを使用すると、この計算をコンパクトかつ完全に同じセマンティックで表現できます。

```
RULES UPSERT
(sales[FOR product IN ('Bounce', 'Standard Mouse Pad', ..., 'Y Box'), 2004] =
   1.1 * sales[CV(product), 2003])
```

この例と同様の指定をFORキーワードを使用せずに記述すると、次のようになります。

```
RULES UPSERT
(sales[product IN ('Bounce', 'Standard Mouse Pad', ..., 'Y Box'), 2004] =
   1.1 * sales[CV(product), 2003])
```

この場合、UPSERTを指定していても、UPDATEの処理方法が適用されます。つまり、既存のセルは更新されますが、この指定では新しいセルは作成されません。その理由は、productに対するマルチセル参照がシンボリック参照であり、シンボリック参照では新しいセルを挿入できないためです。FORコンストラクトは、1つのルールから位置参照で複数のルールを生成するマクロとみなすことができます。これによりUPSERTの処理方法が保持されます。概念上、次のルールがあるとします。

```
sales[FOR product IN ('Bounce', 'Standard Mouse Pad', ..., 'Y Box'),
      FOR year IN (2004, 2005)] = 1.1 * sales[CV(product), CV(year)-1]
```

これは、次のルールの順序付きコレクションとして扱うことができます。

```
sales['Bounce', 2004] = 1.1 * sales[CV(product), CV(year)-1],
sales['Bounce', 2005] = 1.1 * sales[CV(product), CV(year)-1],
sales['Standard Mouse Pad', 2004] = 1.1 *
  sales[CV(product), CV(year)-1],
sales['Standard Mouse Pad', 2005] = 1.1 * sales[CV(product),
 CV(year)-1],
...
sales['Y Box', 2004] = 1.1 * sales[CV(product), CV(year)-1],
sales['Y Box', 2005] = 1.1 * sales[CV(product), CV(year)-1]
```

前述の例の場合、FORコンストラクトの形式は、FOR ディメンション IN *(値のリスト)*となっています。リスト内の値は、定数の式やシングル・セル参照など、単一値の式でなければなりません。前述の例では、FORコンストラクトは、productとyearに別々にあります。単一のFORコンストラクトですべてのディメンションを指定し、複数列のINリストを使用して値を指定することも可能です。たとえば、2004年のBounceの売上高、2005年のStandard Mouse Padの売上高、2004年と2005年のY Boxの売上高のみを見積る場合、次のように記述できます。これは、次のように記述できます。

```
sales[FOR (product, year) IN (('Bounce', 2004), ('Standard Mouse Pad', 2005),
  ('Y Box', 2004), ('Y Box', 2005))] =
     1.1 * sales[CV(product), CV(year)-1]
```

n個のディメンションd1, ..., dnがあり、リスト内にm個の値がある場合、このFORコンストラクトの形式は、FOR (d1, ..., dn) IN ((d1_val1, ..., dn_val1), ..., (d1_valm, ..., dn_valm)]とする必要があります。

FOR内のディメンションの値のリストは、表または副問合せから取得することもできます。このようなケースに対応するためにOracle Databaseでは、FOR ディメンション IN *(副問合せ)*などの形式のFORコンストラクトが用意されています。たとえば、対象の製品が表interesting_productsに格納されるとします。この場合、次のルールでは、2004年と2005年の製品の売上高を見積ります。

```
sales[FOR product IN (SELECT product_name FROM interesting_products)
    FOR year IN (2004, 2005)] = 1.1 * sales[CV(product), CV(year)-1]
```

別の例として、new_countryという新しいcountryを導入し、その売上高を、Polandにおいて売上高が存在するすべての製品および年から流用するシナリオを考えてみます。これを実現するために、次の文を発行します。

```
SELECT country, product, year, s
FROM sales_view
MODEL
DIMENSION BY (country, product, year)
MEASURES (sales s) IGNORE NAV
RULES UPSERT
(s[FOR (country, product, year) IN
       (SELECT DISTINCT 'new_country', product, year
        FROM sales_view
        WHERE country = 'Poland')] = s['Poland',CV(),CV()])
ORDER BY country, year, product;
```

ビューsales_viewは、SQLモデリングの例のベース・スキーマの説明に従って作成されたものです。

この指定に含まれている、副問合せを評価することで生成される複数列のINリストに注目してください。INリストの取得に使用される副問合せは、外部の問合せブロックと相関させることはできません。

このルールで作成されるアップサートのリストは、各ディメンションの個別値のクロス積です。たとえば、countryの値が10個、yearの値が5個、productの値が3個ある場合は、150個のセルを含むアップサートのリストが生成されます。

対象の値が不連続な領域の値であるとわかっている場合は、FORコンストラクトのFOR dimension FROM value1 TO value2 [INCREMENT | DECREMENT] value3を使用できます。これを指定すると、value1からvalue2の範囲内にあり、value1から始まり、value3ずつ増加(または減少)する値を取得できます。value1、value2およびvalue3の各値には、単一値の式を指定する必要があります。たとえば、次のルールがあるとします。

```
sales['Bounce', FOR year FROM 2001 TO 2005 INCREMENT 1] =
  sales['Bounce', year=CV(year)-1] * 1.2
```

これは、意味的および順序的に次のルールと同等です。

```
sales['Bounce', 2001] = sales['Bounce', 2000] * 1.2,
sales['Bounce', 2002] = sales['Bounce', 2001] * 1.2,
...
sales['Bounce', 2005] = sales['Bounce', 2004] * 1.2
```

このタイプのFORコンストラクトは、数値、日付および日時データ型のディメンションに使用できます。増加/減少式のvalue3の型には、数値型のディメンションの場合は数値を、日付または日時型のディメンションの場合は数値またはインターバルを指定する必要があります。また、value3は、正の値である必要があります。FOR year FROM 2005 TO 2001 INCREMENT -1を使用した場合、エラーが戻されます。この場合、FOR year FROM 2005 TO 2001 DECREMENT 1またはFOR year FROM 2001 TO 2005 INCREMENT 1を使用する必要があります。

文字列値を生成する場合は、FORコンストラクトのFOR dimension LIKE string FROM value1 TO value2 [INCREMENT | DECREMENT] value3を使用できます。文字列stringには、%文字を1つ含める必要があります。これを指定すると、value1からvalue2の範囲内にありvalue3の値ごとに増減する値で%を置き換えた文字列を取得できます。たとえば、次のルールがあるとします。

```
sales[FOR product LIKE 'product-%' FROM 1 TO 3 INCREMENT 1, 2003] =
sales[CV(product), 2002] * 1.2
```

これは、次のルールと同等です。

```
sales['product-1', 2003] = sales['product-1', 2002] * 1.2,
sales['product-2', 2003] = sales['product-2', 2002] * 1.2,
sales['product-3', 2003] = sales['product-3', 2002] * 1.2
```

SEQUENTIAL ORDERモデルの場合、FORコンストラクトで表されるルールは、生成された順に評価されます。一方、AUTOMATIC ORDERを指定した場合、ルールの評価順序は依存関係に基づいて決定されます。たとえば、次のようにルールで複数のルールが表される場合の評価順序を考えてみます。

```
sales['Bounce', FOR year FROM 2004 TO 2001 DECREMENT 1] =
  1.1 * sales['Bounce', CV(year)-1]
```

SEQUENTIAL ORDERモデルの場合、ルールは次の順序で生成されます。

```
sales['Bounce', 2004] = 1.1 * sales['Bounce', 2003],
sales['Bounce', 2003] = 1.1 * sales['Bounce', 2002],
sales['Bounce', 2002] = 1.1 * sales['Bounce', 2001],
sales['Bounce', 2001] = 1.1 * sales['Bounce', 2000]
```

一方、AUTOMATIC ORDERモデルの場合、次と同等の順序になります。

```
sales['Bounce', 2001] = 1.1 * sales['Bounce', 2000],
sales['Bounce', 2002] = 1.1 * sales['Bounce', 2001],
sales['Bounce', 2003] = 1.1 * sales['Bounce', 2002],
sales['Bounce', 2004] = 1.1 * sales['Bounce', 2003]
```

関連項目:

FORループを含む式の評価

22.3.1.1 FORループを含む式の評価

FORループ・コンストラクトは、1つのディメンション、またはすべてのディメンション(INリストで複数列を指定する場合)の単一値参照を生成するための反復の仕組みを提供します。左辺にFORループを含む式を評価するプロセスでは、基本的に、これらのFORループによって生成される各単一値参照について式の右辺を評価し、この単一値参照で指定されたセルにその結果を割り当てるという処理が行われます。これらの単一参照値の生成は、「FORループの展開」と呼ばれます。これら展開されたセルは、展開プロセス時の生成順で評価されます。

展開がどのように行われるかは、ルールに指定されたUPSERT、UPDATEまたはUPDATE ALLの動作と、そのルール固有の特性によって異なります。これを理解するには、問合せの処理に伴う、問合せ計画の作成と問合せの実行という2つのステップを説明する必要があります。問合せ計画の作成では、効率的な問合せ実行計画を作成するために、特定のルール参照が解決されます。問合せの実行では、残りの未解決の参照がすべて解決されます。FORループの展開は、問合せ計画の作成時に行われる場合もあれば、問合せの実行時に行われる場合もあります。以降の各項で、展開方法がどのように決定されるかを詳しく説明します。

関連項目:

- UPDATEルールおよびUPSERTルールでの展開
- UPSERT ALLルールでの展開
- 式の左辺でFORループ式を使用する場合の制限事項

22.3.1.1.1 UPDATEルールおよびUPSERTルールでの展開

UPDATEルールまたはUPSERTルールでは、ルールの左辺の展開でシングル・セル参照が生成されることが保証される場合、展開は問合せの実行時に行われます。展開プロセスでシングル・セル参照を生成できない場合、展開は問合せ計画の作成時に行われ、展開プロセスで生成される参照ごとに同じ式のコピーが作成されます。たとえば、次の式では、展開によってシングル・セル参照が生成されるので、展開は問合せの実行時に行われます。

```
sales[FOR product IN ('prod1', 'prod2'), 2003] = sales[CV(product), 2002] * 1.2
```

一方、次の式では、別のディメンションに対する述語が使用されているため、参照値の展開で単一値参照は生成されません。

```
sales[FOR product in ('prod1', 'prod2'), year >= 2003]
   = sales[CV(product), 2002] * 1.2
```

この式には、yearディメンションに対する単一値参照がありません。そのため、たとえFORループがproductディメンションで展開されるとしても、式の左辺に単一値参照は存在しないことになります。これは、展開が問合せ計画の作成時に行われ、元の式は

物理的に次の式に置き換えられることを意味します。

```
sales['prod1', year >= 2003]  = sales[CV(product), 2002] * 1.2,
sales['prod2', year >= 2003]  = sales[CV(product), 2002] * 1.2
```

MODEL句内で実行される分析と最適化は、問合せ計画の作成時における展開の後(展開がこのタイミングで行われる場合)に行われます。したがって、これ以降のすべての処理は、あたかも複数のルールが明示的にMODEL句に指定されているかのように行われます。これらのケースでは、問合せ計画の作成時に展開を行うことによって、より正確な分析と、より効果的な式の評価の最適化が可能になります。ただし、この場合は式の数が増加する可能性があり、それによって式の合計数が上限を超えるとエラーが発生するのでノートが必要です。

22.3.1.1.2 UPSERT ALLでの展開: ルール

UPSERT ALLを使用するルールでは、FORループは様々な方法で展開されます。UPSERT ALLルールのFORループは、使用されている述語に関係なく、常に問合せの実行時に展開されます。この動作によって、次の項で説明するFORループのいくつかの制限が回避されます。ただし、制限が少なくなることと、問合せ計画がより効果的に最適化されるようになることは、トレードオフの関係にあります。UPSERT ALLルールは、一般的に、同じような内容のUPSERTルールまたはUPDATEルールよりも処理が低速です。モデルを設計する際は、この点を考慮する必要があります。

22.3.1.1.3 式の左辺でFORループ式を使用する場合の制限事項

FORループ・コンストラクトの使用に関する制限事項は、展開処理が問合せ計画の作成時に行われるか、問合せの実行時に行われるかによって異なります。FORループを左辺に含む式が(前項で説明した理由によって)問合せ計画の作成時に展開される場合、展開するうえで評価する必要のある式は、問合せ計画の作成時に使用可能な値を持つ定数の式でなければなりません。たとえば、次の文を考えてみます。

```
sales[For product like 'prod%' from ITERATION_NUMBER
to ITERATION_NUMBER+1, year >= 2003] =  sales[CV(product), 2002]*1.2
```

UPSERT ALLが指定されていない場合、このルールは問合せ計画の作成時に展開されます。開始式と終了式を評価するために必要なITERATION_NUMBERの値は、問合せ計画の作成時には不明であるため、このルールが問合せの実行時に展開されない場合には、エラーが発生します。しかし、次のルールであれば、問合せ計画の作成時にエラーなしで展開できます。ITERATION_NUMBERがFORループ内の式として指定されていますが、このルールの場合、その値は展開するために必要ではありません。

```
sales[For product in ('prod'||ITERATION_NUMBER, 'prod'||(ITERATION_NUMBER+1)),
 year >= 2003] =  sales[CV(product), 2002]*1.2
```

次のいずれかを含む式は、問合せ計画の作成時に評価できません。

- ネステッド・セル参照
- 参照モデルの参照
- ITERATION_NUMBER references

このような式の結果を必要とするFORループを含むルールを問合せ計画の作成時に展開すると、エラーが発生します。問合せの実行時に展開する場合には、これらの式によってエラーが発生することはありません。

FORループ・コンストラクトに副問合せを含む式で、コンパイル時の展開が必要な場合、その副問合せは、展開を行えるようにするため、問合せ計画の作成時に評価されます。副問合せを問合せ計画の作成時に評価すると、カーソルを共有できなくなる場合があります。これは、同じ問合せであっても発行のたびに再コンパイルが必要になる可能性があることを意味します。このような式の展開を問合せの実行時まで遅延させると、コンパイル時の評価は不要になり、式がカーソルの共有性に影響を与えることもなくなります。

式のFORループに含まれる副問合せは、式が問合せの実行時に展開される場合には、WITH句内の表を参照できます。その式が問合せ計画の作成時に展開される場合は、エラーが発生します。

22.3.2 SQLモデリングの反復モデル

MODEL句のITERATEオプションを使用すると、ルールを特定の回数で反復的に評価できます。回数は、ITERATE句の引数として指定できます。ITERATEは、SEQUENTIAL ORDERモデルにのみ指定できます。このようなモデルは、反復モデルと呼ばれます。次の例を考えてみます。

```
SELECT x, s FROM DUAL
MODEL
  DIMENSION BY (1 AS x) MEASURES (1024 AS s)
  RULES UPDATE ITERATE (4)
(s[1] = s[1]/2);
```

表DUALにある行はただ1つです。したがって、このモデルでは、xでディメンション化され、メジャーsおよび単一の要素s[1] = 1024を持つ1次元配列を定義します。ルールs[1] = s[1]/2の評価は、4回繰り返されます。この問合せの結果は、列xと列sにそれぞれ値1と値64を持つ単一の行になります。ITERATE句の反復回数の引数には、正の整定数を指定する必要があります。オプションで早期終了条件を指定して、最大反復回数に達する前にルールの評価を停止することもできます。この条件は、ITERATEのUNTIL副次句で指定し、反復の終了時にチェックされます。したがって、ITERATEを指定した場合、最低1回は反復が行われます。ITERATE句の構文を次に示します。

```
ITERATE (number_of_iterations) [ UNTIL (condition) ]
```

反復評価は、指定の反復回数が終了した後または終了条件の評価がTRUEになった時点のいずれか早い時点で停止します。

場合によっては、反復の過程でのセル値の変更に基づいた終了条件が必要になることもあります。Oracle Databaseでは、このような条件を指定するためのメカニズムを備えています。この場合、UNTIL条件で現行の反復の前後におけるセルの値にアクセスできます。OracleのPREVIOUS関数は、引数にシングル・セル参照を取り、前の反復の後に存在するようになったセルのメジャー値を戻します。また、システム変数ITERATION_NUMBERを使用すると、現行の反復回数を取得できます。反復回数は、値0から開始し、1回の反復が終了するたびに増加します。PREVIOUSおよびITERATION_NUMBERを使用することにより、複雑な終了条件を作成できます。

次の反復モデルを考えてみます。この反復モデルでは、最大1000回までの継続的な反復の過程でs[1]の値が1未満に変化するまで、ルールを反復するよう指定しています。

```
SELECT x, s, iterations FROM DUAL
MODEL
  DIMENSION BY (1 AS x) MEASURES (1024 AS s, 0 AS iterations)
  RULES ITERATE (1000) UNTIL ABS(PREVIOUS(s[1]) - s[1]) < 1
 (s[1] = s[1]/2, iterations[1] = ITERATION_NUMBER);
```

最新の値は正か負かわからない場合があるため、終了条件の指定には絶対値関数(ABS)が役立ちます。このモデルのルールは11回反復されます。これは、11回目の反復後にs[1]の値が0.5になるためです。この問合せの結果は、x、sおよび繰返し回数に対して、それぞれ1、0.5および10の値を持つ単一行になります。

PREVIOUS関数は、UNTIL条件でのみ使用できます。ただし、ITERATION_NUMBERは、メイン・モデルの任意の場所に指定できます。次の例では、ITERATION_NUMBERをセル参照で使用しています。

```
SELECT country, product, year, sales
FROM sales_view
MODEL
  PARTITION BY (country) DIMENSION BY (product, year) MEASURES (sales sales)
  IGNORE NAV
  RULES ITERATE(3)
```

```
(sales['Bounce', 2002 + ITERATION_NUMBER] = sales['Bounce', 1999
  + ITERATION_NUMBER]);
```

この文では、Bounceの売上を配列1999-2001のセルから配列2002-2005にコピーします。

ビューsales_viewは、SQLモデリングの例のベース・スキーマの説明に従って作成されたものです。

22.3.3 AUTOMATIC ORDERモデルでのルールの依存関係

Oracle Databaseでは、ルールの依存関係に基づいて、AUTOMATIC ORDERモデルでのルールの評価順序を決定します。ルールは、依存対象のルールが評価された後でのみ評価されます。ルールを評価するためのアルゴリズムは、依存関係分析およびモデルのルール間に循環依存があるかどうかに基づきます。循環依存には、「ルールAはルールBに依存し、ルールBはルールAに依存する」形式と「ルールがそのルール自身に依存する」自己循環の形式があります。前者の例を次に示します。

```
sales['Bounce', 2002] = 1.5 * sales['Y Box', 2002],
sales['Y Box', 2002] = 100000 / sales['Bounce', 2002
```

後者の例を次に示します。

```
sales['Bounce', 2002] = 25000 / sales['Bounce', 2002]
```

ただし、次のルールの場合、左辺と右辺で異なるメジャーにアクセスしているため、自己循環はありません。

```
projected_sales['Bounce', 2002] = 25000 / sales['Bounce', 2002]
```

Oracle Databaseでは、AUTOMATIC ORDERモデルの分析でルールに循環依存がないと判断した場合、依存関係の順序でルールを評価します。たとえば、次のAUTOMATIC ORDERモデルを考えてみます。

```
MODEL DIMENSION BY (prod, year) MEASURES (sale sales) IGNORE NAV
  RULES AUTOMATIC ORDER
 (sales['SUV', 2001] = 10000,
  sales['Standard Mouse Pad', 2001] = sales['Finding Fido', 2001]
    * 0.10 + sales['Boat', 2001] * 0.50,
  sales['Boat', 2001] = sales['Finding Fido', 2001]
    * 0.25 + sales['SUV', 2001]* 0.75,
  sales['Finding Fido', 2001] = 20000)
```

ルール2はルール3と4に依存し、ルール3はルール1と4に依存し、ルール1と4はどのルールにも依存していません。この場合、Oracleではルールの依存関係が非循環であると判断して、有効な評価順序(1、4、3、2)または(4、1、3、2)のいずれかでルールを評価します。このタイプのルール評価は、ACYCLICアルゴリズムと呼ばれます。

Oracle Databaseでは、ルール間に循環依存がなくても、モデルが非循環であることを確認できない場合があります。これは、セル参照内に複雑な式がある場合などで発生します。このような場合、Oracle Databaseではルールが循環依存になっていると仮定し、CYCLICアルゴリズムを使用して、ルールおよびデータに基づいて反復的にモデルを評価します。収束に達した時点で反復は終了し、結果が戻されます。収束とは、これ以上モデルを実行しても、モデルのどのセルの値も変化しない状態のことをいいます。循環依存がない場合、必ず収束に達します。

AUTOMATIC ORDERモデルに循環依存のルールがある場合、Oracle Databaseでは前述のCYCLICアルゴリズムを使用します。アルゴリズムが試行される反復回数内で収束に達した場合、結果が生成されます。それ以外の場合は、循環検出エラーがレポートされます。この問題を回避するには、ルールを手動で順序付け、SEQUENTIAL ORDERを指定します。

22.3.4 SQLモデリングの順序付けルール

順序付きルールは、左辺にORDER BYが指定されているルールです。このルールは、ORDER BYで指定された順序でセルにアクセスし、右辺の計算を適用します。ルールの左辺にANYまたはシンボリック参照がある場合、ORDER BY句を指定しないと、ルール

の結果はセルのアクセス順序に依存しているため不確定的であるという内容のエラーが戻される可能性があります。次のSEQUENTIAL ORDERモデルを考えてみます。

```
SELECT t, s
FROM sales, times
WHERE sales.time_id = times.time_id
GROUP BY calendar_year
MODEL
  DIMENSION BY (calendar_year t) MEASURES (SUM(amount_sold) s)
  RULES SEQUENTIAL ORDER
  (s[ANY] = s[CV(t)-1]);
```

この問合せでは、すべての年tについて、ある年の売上高の値sを前年の売上高の値に設定するよう試みます。ただし、このルールの結果は、セルのアクセス順序に依存しています。年の昇順でセルにアクセスする場合、結果は表22-1の3列目に示す結果になります。降順でセルにアクセスする場合、結果は4列目に示す結果になります。

表22-1 順序付きルール

t	s	昇順の場合	降順の場合
1998	1210000982	null	null
1999	1473757581	null	1210000982
2000	2376222384	null	1473757581
2001	1267107764	null	2376222384

セルを降順で考慮する場合、表の4列目の結果を取得するには、次のように指定する必要があります。

```
SELECT t, s
FROM sales, times
WHERE sales.time_id = times.time_id
GROUP BY calendar_year
MODEL
  DIMENSION BY (calendar_year t) MEASURES (SUM(amount_sold) s)
  RULES SEQUENTIAL ORDER
  (s[ANY] ORDER BY t DESC = s[CV(t)-1]);
```

一般的に、左辺のセル参照と一致するセル間で順序が一意となるかぎり、任意のORDER BY指定を使用できます。ルールのORDER BYの式には、定数、メジャー、ディメンション・キーを含めることができます。また、順序付けオプション[ASC | DESC] [NULLS FIRST | NULLS LAST]を指定すると、目的の順序を取得できます。

AUTOMATIC ORDERモデルのルールにORDER BYを指定することもできます。この場合、ルールの評価時に特定の順序でセルが考慮されます。ルールにORDER BYを指定した場合、そのルールが自己循環とみなされることはありません。たとえば、自己循環の式を使用した次のAUTOMATIC ORDERモデルを非循環にするとします。

```
MODEL
  DIMENSION BY (calendar_year t) MEASURES (SUM(amount_sold) s)
  RULES AUTOMATIC ORDER
  (s[ANY] = s[CV(t)-1])
```

この場合、ORDER BYを使用して、アクセスする評価対象のセルの順序を指定する必要があります。次に例を示します。

```
s[ANY] ORDER BY t = s[CV(t) - 1]
```

これにより、Oracle Databaseは式の評価にACYCLICアルゴリズムを使用して、結果を確実に生成します。

22.3.5 SQLモデリングの分析関数

分析関数(ウィンドウ関数とも呼ばれる)は、ルールの右辺で使用できます。分析関数を使用すると、MODEL句で、より複雑な式を柔軟に計算できるようになります。次の例では、分析関数とMODEL句を組み合せて使用します。まず、GROUPING_ID関数を使用して集計の各レベルの識別子を計算するビューsales_rollup_timeを作成します。次に、quarterレベルとyearレベルの両方で売上高の累積合計を計算する問合せの中で、このビューを使用します。

```
CREATE OR REPLACE VIEW sales_rollup_time
AS
SELECT country_name country, calendar_year year, calendar_quarter_desc quarter,
GROUPING_ID(calendar_year, calendar_quarter_desc) gid, SUM(amount_sold) sale,
COUNT(amount_sold) cnt
FROM sales, times, customers, countries
WHERE sales.time_id = times.time_id AND sales.cust_id = customers.cust_id
  AND customers.country_id = countries.country_id
GROUP BY country_name, calendar_year, ROLLUP(calendar_quarter_desc)
ORDER BY gid, country, year, quarter;

SELECT country, year, quarter, sale, csum
FROM sales_rollup_time
WHERE country IN ('United States of America', 'United Kingdom')
MODEL DIMENSION BY (country, year, quarter)
MEASURES (sale, gid, 0 csum)
(
csum[any, any, any] =
  SUM(sale) OVER (PARTITION BY country, DECODE(gid,0,year,null)
ORDER BY year, quarter
ROWS UNBOUNDED PRECEDING)
)
ORDER BY country, gid, year, quarter;

COUNTRY                          YEAR QUARTER       SALE       CSUM
------------------------------ ---------- ------- ---------- ----------
United Kingdom                   1998 1998-01  484733.96  484733.96
United Kingdom                   1998 1998-02  386899.15  871633.11
United Kingdom                   1998 1998-03  402296.49  1273929.6
United Kingdom                   1998 1998-04  384747.94 1658677.54
United Kingdom                   1999 1999-01  394911.91  394911.91
United Kingdom                   1999 1999-02  331068.38  725980.29
United Kingdom                   1999 1999-03  383982.61  1109962.9
United Kingdom                   1999 1999-04  398147.59 1508110.49
United Kingdom                   2000 2000-01  424771.96  424771.96
United Kingdom                   2000 2000-02  351400.62  776172.58
United Kingdom                   2000 2000-03  385137.68 1161310.26
United Kingdom                   2000 2000-04   390912.8 1552223.06
United Kingdom                   2001 2001-01  343468.77  343468.77
United Kingdom                   2001 2001-02  415168.32  758637.09
United Kingdom                   2001 2001-03  478237.29 1236874.38
United Kingdom                   2001 2001-04  437877.47 1674751.85
United Kingdom                   1998         1658677.54 1658677.54
United Kingdom                   1999         1508110.49 3166788.03
United Kingdom                   2000         1552223.06 4719011.09
United Kingdom                   2001         1674751.85 6393762.94
...  /*and similar output for the US*/
```

分析関数を使用する際は、いくつかの固有の制限事項が適用されます。詳細は、モデリング用SQLを使用する場合の規則お

よび制限事項を参照してください。

22.3.6 SQLモデリングのUNIQUE DIMENSIONとUNIQUE SINGLE REFERENCE

MODEL句のデフォルトの動作では、モデルへの入力で各行を一意に識別するために、PARTITION BYおよびDIMENSION BYキーが必要です。Oracleは、データの一意性を検証し、データが一意でない場合はエラーを戻します。PARTITION BYおよびDIMENSION BYキーでの入力行セットの一意性によって、シングル・セル参照がアクセスするモデル内のセルが1つに限定されることが保証されます。MODEL句でオプションのキーワードUNIQUE DIMENSIONを指定すると、この動作を明示できます。たとえば、次の問合せは、SQLモデリングの例のベース・スキーマの説明に従って作成されたビューsales_viewに対して実行されます。

```
SELECT country, product, sales
FROM sales_view
WHERE country IN ('France', 'Poland')
MODEL UNIQUE DIMENSION
  PARTITION BY (country) DIMENSION BY (product) MEASURES (sales sales)
  IGNORE NAV RULES UPSERT
(sales['Bounce'] = sales['All Products'] * 0.24);
```

この問合せは、モデルに入力された行セットがcountryおよびproductでは一意でないため(yearも必要)、次の一意性違反エラーを戻します。

```
ERROR at line 2:ORA-32638: Non unique addressing in MODEL dimensions
```

ただし、次の問合せの場合、このようなエラーは戻されません。

```
SELECT country, product, year, sales
FROM sales_view
WHERE country IN ('Italy', 'Japan')
MODEL UNIQUE DIMENSION
  PARTITION BY (country) DIMENSION BY (product, year) MEASURES (sales sales)
  RULES UPSERT
(sales['Bounce', 2003] = sales['All Products', 2002] * 0.24);
```

この場合、MODEL句への入力は、次に示すようにcountry、productおよびyearで一意です。

```
COUNTRY    PRODUCT                          YEAR   SALES
-------    ------------------------------   ----   --------
Italy      1.44MB External 3.5" Diskette    1998    3141.84
Italy      1.44MB External 3.5" Diskette    1999    3086.87
Italy      1.44MB External 3.5" Diskette    2000    3440.37
Italy      1.44MB External 3.5" Diskette    2001     855.23
...
```

UNIQUE SINGLE REFERENCEキーワードを指定すると、一意性を簡単にチェックできます。これにより、処理時間を節約できます。この場合、MODEL句は、ルールの右辺に現れるシングル・セル参照のみの一意性をチェックします。したがって、一意性違反エラーを戻した問合せでUNIQUE DIMENSIONのかわりにUNIQUE SINGLE REFERENCEを指定すると、エラーが発生しなくなります。

UNIQUE DIMENSIONとUNIQUE SINGLE REFERENCEのもう1つの違いは、左辺にシングル・セル参照があるルールで更新できるセルの数です。UNIQUE DIMENSIONの場合、左辺のシングル・セル参照に1つのセルのみが一致するようなルールで更新できる行は最大でも1つです。これは、入力行セットがPARTITION BYおよびDIMENSION BYキーで一意であるためです。UNIQUE SINGLE REFERENCEの場合、左辺のシングル・セル参照に一致するすべてのセルがルールで更新されます。

22.3.7 モデリング用SQLを使用する場合の規則および制限事項

MODEL句を使用する場合は、次の一般規則および制限事項が適用されます。

- 更新可能な列は、メインSQLモデルのMEASURES副次句で指定された列のみです。参照モデルのメジャーは更新できません。
- MODEL句が評価されるのは、SELECT DISTINCT句を除く問合せブロック内のすべての句およびORDER BY句が評価された後です。SELECT構文のリスト内の句および式は、MODEL句の後に評価されます。
- 問合せにMODEL句がある場合、問合せのSELECTおよびORDER BYリストに集計関数または分析関数は指定できません。これらの関数が必要な場合は、PARTITION BY、DIMENSION BYおよびMEASURESリストで指定可能であり、別名化する必要があります。これにより、SELECTまたはORDER BY句で別名を使用できるようになります。次の例では、分析関数RANKをMODEL句のMEASURESリストで指定および別名化し、その別名をSELECT構文のリストで使用して、外部の問合せがランクに基づいて結果の行を順序付けできるようにしています。

```
SELECT country, product, year, s, RNK
FROM (SELECT country, product, year, s, rnk
      FROM sales_view
      MODEL
        PARTITION BY (country) DIMENSION BY (product, year)
        MEASURES (sales s, year y, RANK() OVER (ORDER BY sales) rnk)
        RULES UPSERT
          (s['Bounce Increase 90-99', 2001] =
             REGR_SLOPE(s, y) ['Bounce', year BETWEEN 1990 AND 2000],
           s['Bounce', 2001] = s['Bounce', 2000] *
          (1+s['Bounce increase 90-99', 2001])))
WHERE product <> 'Bounce Increase 90-99'
ORDER BY country, year, rnk, product;
```

- ルールの右辺にマルチセル参照がある場合、集計関数を適用して、マルチセル参照の複数のメジャー値を単一値に変換する必要があります。これには、通常の集計関数、分析集計関数(逆パーセンタイル関数、仮説ランク関数、仮説分布関数)、ユーザー定義の集計関数など、任意の種類の集計関数を使用できます。
- UPSERTを指定できるのは、左辺に位置ベースのシングル・セル参照があるルールのみです。それ以外のルールはすべて、UPSERTオプションを指定した場合でも、UPDATEになります。
- FORループでは、負の値の増加は禁止されています。また、空のFORループも禁止されています。たとえば、FOR d FROM 2005 TO 2001 INCREMENT -1は無効です。かわりに、FOR d FROM 2005 TO 2001 DECREMENT 1を使用する必要があります。FOR d FROM 2005 TO 2001 INCREMENT 1は、空のループを指定しているため無効です。
- FORコンストラクト内を除き、ルールではネストした問合せ式(副問合せ)は使用できません。たとえば、次の文を発行した場合は無効になります。

```
SELECT *
FROM sales_view WHERE country = 'Poland'
MODEL DIMENSION BY (product, year)
  MEASURES (sales sales)
  RULES UPSERT
  (sales['Bounce', 2003] = sales['Bounce', 2002] +
   (SELECT SUM(sales) FROM sales_view));
```

これは、ルールの右辺に副問合せがあるためです。前述の問合せは、次の有効な方法で記述しなおすことができます。

```
SELECT *
FROM sales_view WHERE country = 'Poland'
MODEL DIMENSION BY (product, year)
```

```
MEASURES (sales sales, (SELECT SUM(sales) FROM sales_view) AS grand_total)
RULES UPSERT
(sales['Bounce', 2003] =sales['Bounce', 2002] +
 grand_total['Bounce', 2002]);
```

- ルールの左辺に指定されたFORコンストラクトにも副問合せを使用できます。ただし、次の制限があります。
  - 相関させることはできません。
  - 戻り値の行数は10,000行未満に制限する必要があります。
  - WITH句で定義した問合せにすることはできません。
  - カーソルは共有できません。

ネステッド・セル参照には、次の制限事項があります。

- ネステッド・セル参照は、シングル・セル参照である必要があります。ネステッド・セル参照での集計はサポートされていません。したがって、s['Bounce', MAX(best_year)['Bounce', ANY]]のような指定は無効になります。
- メイン・モデルのネステッド・セル参照の場合、サポートされているネストのレベルは1レベルのみです。たとえば、s['Bounce', best_year['Bounce', 2001]]は有効ですが、s['Bounce', best_year['Bounce', best_year['Bounce', 2001]]]は無効になります。
- AUTOMATIC ORDERモデルでは、ルールの左辺にあるネステッド・セル参照をモデルのルールで更新しないようにしてください。この制限によって、参照メジャーの更新が原因でルールの依存関係が無原則に変更される(そのため、結果が不確定的になる)のを防止できます。

  SEQUENTIAL ORDERモデルのネステッド・セル参照の場合、このような制限はありません。また、SEQUENTIALまたはAUTOMATIC ORDERのどちらのモデルでも、ルールの右辺にあるネステッド・セル参照には、この制限は適用されません。

参照モデルには、次の制限事項があります。

- 参照モデルを定義する問合せは、外部の問合せと相関させることはできません。ただし、副問合せやビューなどを持つ問合せにすることはできます。
- 参照モデルでは、PARTITION BY句は指定できません。
- 参照モデルは更新できません。

ウィンドウ関数には、次の制限事項があります。

- OVER句に指定できる式は、定数の式、メジャー、MODEL句のPARTITION BYとDIMENSION BYのキー、および単一のセルの式です。集計は、OVER句の内部には指定できません。次の式は許容されます。

  ```
  rnk[ANY, ANY, ANY] = RANK() OVER (PARTITION BY prod, country ORDER BY sale)
  ```

  次の式は許容されません。

  ```
  rnk[ANY, ANY, ANY] = RANK() OVER (PARTITION BY prod, country ORDER BY SUM(sale))
  ```

- ルールの右辺にウィンドウ関数が含まれる場合、同じルールの左辺にはORDER BY句を指定できません。
- ウィンドウ関数と集計関数の両方を同じルールの右辺には指定できません。
- ウィンドウ関数は、UPDATEルールの右辺でのみ使用できます。
- ルールの左辺にFORループがある場合、ウィンドウ関数は同じルールの右辺で使用できません。

22.4 SQLモデリングのパフォーマンスに関する考慮事項

以降の項では、MODEL句を使用する場合のパフォーマンスに影響するトピックについて説明します。

- パラレル実行とSQLモデリング
- 集計計算とSQLモデリング
- EXPLAIN PLANを使用したモデル問合せの理解

22.4.1 パラレル実行とSQLモデリング

MODEL句の計算のスケーラビリティは、使用するプロセッサの数に応じて向上します。また、スケーラビリティは、PARTITION BY句で定義したパーティション間でMODEL計算をパラレルに実行することにより実現します。データは、PARTITION BYキーの値に基づいて(パラレル問合せスレーブと呼ばれる)処理要素に分散されます。これにより、PARTITION BYキーの値が同じ行は、すべて同じスレーブに配置されます。この場合、パーティションの内部処理では、論理的に処理されるパーティションと内部的に処理されるパーティションは1対1の一致とはなりません。これにより、各スレーブは、他のスレーブとは関係なくMODEL句の計算を終了できます。データは、ハッシュ・ベースまたはレンジ・ベースでパーティション化できます。次のMODEL句を考えてみます。

```
MODEL
  PARTITION BY (country) DIMENSION BY (product, time) MEASURES (sales)
  RULES UPDATE
  (sales['Bounce', 2002] = 1.2 * sales['Bounce', 2001],
   sales['Car', 2002] = 0.8 * sales['Car', 2001])
```

ここで、入力データは、PARTITION BYキーcountryに基づいてスレーブ間でパーティション化されます。このパーティション化は、ハッシュ・ベースまたはレンジ・ベースで行うことができます。各スレーブは、受け取ったデータのルールを評価します。

モデル計算のパラレル化は、MODEL句の指定方法によって制御または制限されます。MODEL句にPARTITION BYキーがない場合、(後述の例外を除き)計算はパラレル化できません。PARTITION BYキーのカーディナリティが非常に低い場合、並列度は制限されます。このような場合、Oracleでは、パーティション化に使用可能なDIMENSION BYキーを識別します。たとえば、前述の例とほぼ同等だがPARTITION BYキーがない、次のようなMODEL句があるとします。

```
MODEL
  DIMENSION BY (country, product, time) MEASURES (sales)
  RULES UPDATE
  (sales[ANY, 'Bounce', 2002] = 1.2 * sales[CV(country), 'Bounce', 2001],
   sales[ANY, 'Car', 2002] = 0.8 * sales[CV(country), 'Car', 2001])
```

この場合、Oracle Databaseでは、DIMENSION BYキーcountryがパーティション化に使用可能であると識別し、内部のパーティション化の処理にregionを使用します。データは、countryのスレーブ間でパーティション化され、パラレル実行が行われます。

22.4.2 集計計算とSQLモデリング

MODEL句の集計の処理には2通りの方法があります。1つはパーティション内のデータをスキャンして集計する一般的な方法、もう1つはウィンドウ形式の集計を行う効率的な方法です。最初の方法の例を次に示します。ここでは、新しいディメンション・メンバーALL_2002_productsを導入し、その値が全製品の2002年の合計売上になるよう計算します。

```
MODEL PARTITION BY (country) DIMENSION BY (product, time) MEASURES (sale sales)
RULES UPSERT
 (sales['ALL_2002_products', 2002] = SUM(sales)[ANY, 2002])
```

この場合、集計の合計を評価するために、各パーティションをスキャンして、全製品の2002年のセルを検索して集計します。ルールの左辺が複数のセルを参照する場合は、左辺で参照されるセルごとにパーティションがスキャンされ、右辺の集計が計算されます。たとえば、次の例を考えてみます。

```
MODEL PARTITION BY (country) DIMENSION BY (product, time)
  MEASURES (sale sales, 0 avg_exclusive)
  RULES UPDATE
  (avg_exclusive[ANY, 2002] =  AVG(sales)[product <> CV(product), CV(time)])
```

このルールは、2002年の全製品を対象にメジャーavg_exclusiveを計算します。メジャーavg_exclusiveは、現行の製品を除外したすべての製品の平均売上として定義されています。この場合、2002年の全製品についてパーティションのデータがスキャンされ集計が計算されるため、コストが高くなる可能性があります。

Oracle Databaseでは、このような集計の評価を最適化するために、集計関数で使用される、ウィンドウ形式の計算のシナリオが複数用意されています。これらのシナリオでは、左辺にマルチセル参照を持つルールを指定し、移動平均や累積合計などのウィンドウ計算を実行します。次の例を検討してください:

```
MODEL PARTITION BY (country) DIMENSION BY (product, time)
  MEASURES (sale sales, 0 mavg)
  RULES UPDATE
  (mavg[product IN ('Bounce', 'Y Box', 'Mouse Pad'), ANY] =
   AVG(sales)[CV(product), time BETWEEN CV(time)
   AND CV(time) - 2])
```

この例では、3年間にわたる製品Bounce、Y BoxおよびMouse Padの売上の移動平均を計算します。左辺で参照されるすべてのセルについてパーティションがスキャンされると、集計の評価が非常に非効率的になります。Oracleでは、ウィンドウ形式で計算が識別され、効率的に評価されます。この場合、productおよびtimeの入力をソートしてから、データを1回スキャンして移動平均を計算します。このルールは、次のように製品Bounce、Y BoxおよびMouse Padの売上データに適用する集計関数として表すことができます。

```
AVG(sales) OVER (PARTITION BY product ORDER BY time
RANGE BETWEEN 2 PRECEDING AND CURRENT ROW)
```

この計算形式は、WINDOW (IN MODEL) SORTと呼ばれます。この形式の集計を適用できるのは、ルールの左辺にORDER BYが指定されていないマルチセル参照があり、ルールの右辺に単純な集計(SUM、COUNT、MIN、MAX、STDEV、VAR)があり、右辺のいずれか1つのディメンションにブール型の述語(<、<=、>、>=、BETWEEN)があり、右辺のその他のディメンションがすべてCVで修飾されている場合です。

22.4.3 EXPLAIN PLANを使用したモデル問合せの理解

OracleのEXPLAIN PLAN機能は、モデルに完全対応しています。問合せに対するEXPLAIN PLANのメインの出力では、モデルおよび使用されたアルゴリズムを示す行が表示されます。参照モデルの場合、プランの出力ではキーワードREFERENCEでタグ付けられます。また、いずれかのルールでウィンドウ形式の集計計算を許可する場合、プランにWINDOW (IN MODEL) SORTという注釈が付けられます。

EXPLAIN PLANを調べることにより、モデルの評価で使用されたアルゴリズムを確認できます。モデルにSEQUENTIAL ORDERが指定されている場合、ORDEREDが表示されます。AUTOMATIC ORDERモデルの場合は、評価用のアルゴリズムがACYCLICかCYCLICかに応じて、ACYCLICまたはCYCLICが表示されます。さらに、左辺のすべてのセル参照がシングル・セル参照であり、ルールの右辺の集計が単純で非個別的な算術集計(SUM、COUNT、AVGなど)の場合、アルゴリズムがORDEREDおよびACYCLICだと、プランの出力に注釈FASTが付けられます。この場合、ルールの評価が非常に効率的になるため、注釈FASTが付きます。したがって、EXPLAIN PLANの出力では、MODEL {ORDERED [FAST] | ACYCLIC [FAST] | CYCLIC}と表示されます。

この項では、次の項目について説明します。

- ORDERED FASTの使用例
- ORDEREDの使用例

- ACYCLIC FASTの使用例
- ACYCLICの使用例
- CYCLICの使用例

ORDERED FASTの使用例

このモデルでは、ルールの左辺にシングル・セル参照のみがあり、最初のルールの右辺にある集計AVGは単純で非個別的な集計になっています。

```
EXPLAIN PLAN FOR
SELECT country, product, year, sales
FROM sales_view
WHERE country IN ('Italy', 'Japan')
MODEL UNIQUE DIMENSION
  PARTITION BY (country) DIMENSION BY (product, year) MEASURES (sales sales)
  RULES UPSERT
  (sales['Bounce', 2003] = AVG(sales)[ANY, 2002] * 1.24,
   sales['Y Box', 2003] = sales['Bounce', 2003] * 0.25);
```

ORDEREDの使用例

次の例の場合、2番目のルールの左辺がマルチセル参照になっているため、FASTは選択されません。

```
EXPLAIN PLAN FOR
SELECT country, product, year, sales
FROM sales_view
WHERE country IN ('Italy', 'Japan')
MODEL UNIQUE DIMENSION
  PARTITION BY (country) DIMENSION BY (product, year) MEASURES (sales sales)
  RULES UPSERT
  (sales['Bounce', 2003] = AVG(sales)[ANY, 2002] * 1.24,
   sales[prod <> 'Bounce', 2003] = sales['Bounce', 2003] * 0.25);
```

ACYCLIC FASTの使用例

このモデルのルールは非循環になっているため、EXPLAIN PLANではACYCLICが表示されます。この場合、FASTも選択されます。

```
EXPLAIN PLAN FOR
SELECT country, product, year, sales
FROM sales_view
WHERE country IN ('Italy', 'Japan')
MODEL UNIQUE DIMENSION
  PARTITION BY (country) DIMENSION BY (product, year) MEASURES (sales sales)
  RULES UPSERT AUTOMATIC ORDER
  (sales['Y Box', 2003] = sales['Bounce', 2003] * 0.25,
   sales['Bounce', 2003] = sales['Bounce', 2002] / SUM(sales)[ANY, 2002] * 2 *
   sales['All Products', 2003],
   sales['All Products', 2003] = 200000);
```

ACYCLICの使用例

このモデルのルールは非循環です。2番目のルールでは、2002年の売上高の中央値を示すPERCENTILE_DISC集計があります。この集計は、単純な集計関数ではありません。そのため、FASTは選択されず、EXPLAIN PLANではACYCLICのみが表示されます。

```
SELECT country, product, year, sales
FROM sales_view
WHERE country IN ('Italy', 'Japan')
```

```
MODEL UNIQUE DIMENSION
  PARTITION BY (country) DIMENSION BY (product, year) MEASURES (sales sales)
  RULES UPSERT AUTOMATIC ORDER
  (sales['Y Box', 2003] = sales['Bounce', 2003] * 0.25,
   sales['Bounce',2003] = PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY
   sales)[ANY,2002] / SUM(sales)[ANY, 2002] * 2 * sales['All Products', 2003],
   sales['All Products', 2003] = 200000);
```

CYCLICの使用例

2番目と3番目のルールは循環依存になっているため、このモデルではCYCLICアルゴリズムが選択されます。

```
EXPLAIN PLAN FOR
SELECT country, product, year, sales
FROM sales_view
WHERE country IN ('Italy', 'Japan')
MODEL UNIQUE DIMENSION
  PARTITION BY (country) DIMENSION BY (product, year) MEASURES (sales sales)
  IGNORE NAV RULES UPSERT AUTOMATIC ORDER
  (sales['All Products', 2003] = 200000,
   sales['Y Box', 2003] = sales['Bounce', 2003] * 0.25,
   sales['Bounce', 2003] = sales['Y Box', 2003] +
    (sales['Bounce', 2002] / SUM(sales)[ANY, 2002] * 2 *
     sales['All Products', 2003]));
```

22.5 SQLモデリングの例

この項の各例では、sales_view (SQLモデリングの例のベース・スキーマで作成されたもの)に加え、次のビューが定義されていると想定します。このビューは、製品別および国別の月単位による合計売上高と販売数量を示します。

```
CREATE VIEW sales_view2 AS
SELECT country_name country, prod_name product, calendar_year year,
  calendar_month_name month, SUM(amount_sold) sale, COUNT(amount_sold) cnt
FROM sales, times, customers, countries, products
WHERE sales.time_id = times.time_id AND
      sales.prod_id = products.prod_id AND
      sales.cust_id = customers.cust_id AND
      customers.country_id = countries.country_id
GROUP BY country_name, prod_name, calendar_year, calendar_month_name;
```

この項には次の例が含まれます:

- SQLモデリング例1: 売上高の差の計算
- SQLモデリング例2: 変化率の計算
- SQLモデリング例3: 正味現在価値の計算
- SQLモデリング例4: 連立方程式を使用した計算
- SQLモデリング例5: 回帰を使用した計算
- SQLモデリング例6: 貸付金の割賦償還額の計算

22.5.1 SQLモデリング例1: 売上高の差の計算

ItalyとSpainの売上高および両国の売上高の差を製品別に示します。差は、country = 'Diff Italy-Spain'を使用して新しい行に配置されます。

```
SELECT product, country, sales
```

```
FROM sales_view
WHERE country IN ('Italy', 'Spain')
GROUP BY product, country
MODEL
  PARTITION BY (product) DIMENSION BY (country) MEASURES (SUM(sales) AS sales)
  RULES UPSERT
  (sales['DIFF ITALY-SPAIN'] = sales['Italy'] - sales['Spain']);
```

この例を実行するために必要なビューについては、SQLモデリングの例を参照してください。

22.5.2 SQLモデリング例2: 変化率の計算

2000年10月から11月と同年11月から12月のそれぞれの期間において、各国での各製品の売上高が同じ月間変化率で増加(または減少)した場合、会社全体および国ごとの第4四半期の売上高を計算します。

```
SELECT country, SUM(sales)
FROM (SELECT product, country, month, sales
      FROM sales_view2
      WHERE year=2000 AND month IN ('October','November')
MODEL
  PARTITION BY (product, country) DIMENSION BY (month) MEASURES (sale sales)
  RULES
   (sales['December']=(sales['November'] /sales['October']) *sales['November']))
GROUP BY GROUPING SETS ((),(country));
```

この例を実行するために必要なビューについては、SQLモデリングの例を参照してください。

22.5.3 SQLモデリング例3: 正味現在価値の計算

一連の定期的なキャッシュ・フローの正味現在価値(NPV)を計算するとします。シナリオには、2つのプロジェクトがあります。それぞれ、時間0での負のキャッシュ・フローで表される初期投資から開始します。初期投資以降の3年間のキャッシュ・フローは正の値になります。最初に、次の文で表(cash_flow)を作成し、データを移入します。

```
CREATE TABLE cash_flow (year DATE, i INTEGER, prod VARCHAR2(3), amount NUMBER);
INSERT INTO cash_flow VALUES (TO_DATE('1999', 'YYYY'),  0,  'vcr',  -100.00);
INSERT INTO cash_flow VALUES (TO_DATE('2000', 'YYYY'),  1,  'vcr',   12.00);
INSERT INTO cash_flow VALUES (TO_DATE('2001', 'YYYY'),  2,  'vcr',  10.00);
INSERT INTO cash_flow VALUES (TO_DATE('2002', 'YYYY'),  3,  'vcr',  20.00);
INSERT INTO cash_flow VALUES (TO_DATE('1999', 'YYYY'),  0,  'dvd',  -200.00);
INSERT INTO cash_flow VALUES (TO_DATE('2000', 'YYYY'),  1,  'dvd',  22.00);
INSERT INTO cash_flow VALUES (TO_DATE('2001', 'YYYY'),  2,  'dvd',  12.00);
INSERT INTO cash_flow VALUES (TO_DATE('2002', 'YYYY'),  3,  'dvd',  14.00);
```

この例を実行するために必要なビューについては、SQLモデリングの例を参照してください。

割引率を0.14としてNPVを計算する場合、次の文を発行します。

```
SELECT year, i, prod, amount, npv
FROM cash_flow
MODEL PARTITION BY (prod)
  DIMENSION BY (i)
  MEASURES (amount, 0 npv, year)
  RULES
    (npv[0] = amount[0],
     npv[i !=0] ORDER BY i =
       amount[CV()]/ POWER(1.14,CV(i)) + npv[CV(i)-1]);

YEAR                I PRO      AMOUNT        NPV
--------- ---------- --- ---------- ----------
```

```
01-AUG-99          0 dvd       -200        -200
01-AUG-00          1 dvd         22 -180.70175
01-AUG-01          2 dvd         12 -171.46814
01-AUG-02          3 dvd         14 -162.01854
01-AUG-99          0 vcr       -100        -100
01-AUG-00          1 vcr         12 -89.473684
01-AUG-01          2 vcr         10 -81.779009
01-AUG-02          3 vcr         20 -68.279579
```

22.5.4 SQLモデリング例4: 連立方程式を使用した計算

利子負担率を純所得(純所得=給与-税金-利子)の30%にするとします。利子は総所得から控除できます。税率は給与の38%、資産売却益の28%です。給与は100,000ドル、資産売却益は15,000ドルとします。純所得、税額および利子負担率は不明です。純所得は利子に依存し、利子は純所得に依存していることから、これは連立方程式であるため、ITERATE句が含まれていることに注目してください。

この例を実行するために必要なビューについては、SQLモデリングの例を参照してください。

最初に、表ledgerを作成します。

```
CREATE TABLE  ledger  (account  VARCHAR2(20), balance  NUMBER(10,2) );
```

次の5つの行を挿入します。

```
INSERT INTO ledger VALUES  ('Salary', 100000);
INSERT INTO ledger VALUES  ('Capital_gains', 15000);
INSERT INTO ledger VALUES  ('Net', 0);
INSERT INTO ledger VALUES  ('Tax', 0);
INSERT INTO ledger VALUES  ('Interest', 0);
```

続いて、次の文を発行します。

```
SELECT s, account
FROM ledger
MODEL
  DIMENSION  BY (account) MEASURES (balance s)
  RULES  ITERATE (100)
  (s['Net']=s['Salary']-s['Interest']-s['Tax'],
 s['Tax']=(s['Salary']-s['Interest'])*0.38 + s['Capital_gains']*0.28,
  s['Interest']=s['Net']*0.30);
```

出力は次のようになります(端数は切り捨てられています)。

```
         S ACCOUNT
---------- --------------------
    100000 Salary
     15000 Capital_gains
48735.2445 Net
36644.1821 Tax
14620.5734 Interest
```

22.5.5 SQLモデリング例5: 回帰を使用した計算

2001年のBounceの売上高は、過去3年(1998年から2000年)と同様、2000年と比べて増加する見通しです。この増加額を計算するには、次のように回帰関数REGR_SLOPEを使用します。ここでは、次の期間の値を計算するため、傾きを2000年の値に追加すれば十分です。

```
SELECT * FROM
```

```
 (SELECT country, product, year, projected_sale, sales
  FROM sales_view
  WHERE country IN ('Italy', 'Japan') AND product IN ('Bounce')
MODEL
  PARTITION BY (country) DIMENSION BY (product, year)
  MEASURES (sales sales, year y, CAST(NULL AS NUMBER) projected_sale) IGNORE NAV
  RULES UPSERT
  (projected_sale[FOR product IN ('Bounce'), 2001] =
             sales[CV(), 2000] +
             REGR_SLOPE(sales, y)[CV(), year BETWEEN 1998 AND 2000]))
ORDER BY country, product, year;
```

この例を実行するために必要なビューについては、SQLモデリングの例を参照してください。

出力は次のようになります。

```
COUNTRY    PRODUCT            YEAR   PROJECTED_SALE  SALES
-------    -------            ----   --------------  -------
Italy      Bounce             1999                    2474.78
Italy      Bounce             2000                    4333.69
Italy      Bounce             2001           6192.6   4846.3
Japan      Bounce             1999                    2961.3
Japan      Bounce             2000                    5133.53
Japan      Bounce             2001          7305.76   6303.6
```

22.5.6 SQLモデリング例6: 貸付金の割賦償還額の計算

この例では、貸付金のファクト表から選択した住宅ローンに関する情報を使用して、任意の数の顧客を対象とした貸付金の割賦償還額の表を作成します。最初に、次の2つの表を作成し、必要なデータを挿入します。

- mortgage_facts

  顧客名など個々の顧客のローンに関する情報、行に格納されているローンに関するファクト、およびそのファクトの値を保持します。この例で格納されるファクトには、ローン(Loan)、年間利子率(Annual_Interest)およびローンの支払回数(Payments)があります。また、2名の顧客(SmithとJones)の値も挿入されます。

  ```
  CREATE TABLE mortgage_facts (customer VARCHAR2(20), fact VARCHAR2(20),
    amount  NUMBER(10,2));
  INSERT INTO mortgage_facts  VALUES ('Smith', 'Loan', 100000);
  INSERT INTO mortgage_facts  VALUES ('Smith', 'Annual_Interest', 12);
  INSERT INTO mortgage_facts  VALUES ('Smith', 'Payments', 360);
  INSERT INTO mortgage_facts  VALUES ('Smith', 'Payment', 0);
  INSERT INTO mortgage_facts  VALUES ('Jones', 'Loan', 200000);
  INSERT INTO mortgage_facts  VALUES ('Jones', 'Annual_Interest', 12);
  INSERT INTO mortgage_facts  VALUES ('Jones', 'Payments', 180);
  INSERT INTO mortgage_facts  VALUES ('Jones', 'Payment', 0);
  ```

- mortgage

  計算の出力情報を保持します。これには、顧客、支払回数(pmt_num)、支払に適用される元本金額(principalp)、支払に適用される利子(interestp)、ローンの残高(mort_balance)の列があります。新しいセルをパーティションにアップサートするには、事前に各パーティションに少なくとも1つの行を作成しておく必要があります。したがって、2名の顧客が支払を行う前に、顧客の値を貸付金の表に入力します。このシード情報は、表mortgage_factsに基づいてSQLのINSERT文を使用すると、簡単に生成できます。

```
CREATE TABLE mortgage_facts (customer VARCHAR2(20), fact VARCHAR2(20),
   amount  NUMBER(10,2));

INSERT INTO mortgage_facts  VALUES ('Smith', 'Loan', 100000);
```

```
INSERT INTO mortgage_facts  VALUES ('Smith', 'Annual_Interest', 12);
INSERT INTO mortgage_facts  VALUES ('Smith', 'Payments', 360);
INSERT INTO mortgage_facts  VALUES ('Smith', 'Payment', 0);
INSERT INTO mortgage_facts  VALUES ('Smith', 'PaymentAmt', null);
INSERT INTO mortgage_facts  VALUES ('Jones', 'Loan', 200000);
INSERT INTO mortgage_facts  VALUES ('Jones', 'Annual_Interest', 12);
INSERT INTO mortgage_facts  VALUES ('Jones', 'Payments', 180);
INSERT INTO mortgage_facts  VALUES ('Jones', 'Payment', 0);
INSERT INTO mortgage_facts  VALUES ('Jones', 'PaymentAmt', null);


CREATE TABLE mortgage (customer VARCHAR2(20), pmt_num NUMBER(4),
   principalp NUMBER(10,2), interestp NUMBER(10,2), mort_balance NUMBER(10,2));

INSERT INTO mortgage VALUES ('Jones',0, 0, 0, 200000);
INSERT INTO mortgage VALUES ('Smith',0, 0, 0, 100000);
```

この例を実行するために必要なビューについては、SQLモデリングの例を参照してください。

次のSQL文は複雑であるため、必要に応じて個々の行に注釈を付けています。これらの行の詳細については後述します。

```
SELECT c, p, m, pp, ip
FROM MORTGAGE
MODEL                                                              --See 1
REFERENCE R ON
   (SELECT customer, fact, amt                                     --See 2
    FROM mortgage_facts
    MODEL DIMENSION BY (customer, fact) MEASURES (amount amt)      --See 3
      RULES
       (amt[any, 'PaymentAmt']= (amt[CV(),'Loan']*
          Power(1+ (amt[CV(),'Annual_Interest']/100/12),
                   amt[CV(),'Payments']) *
         (amt[CV(),'Annual_Interest']/100/12)) /
         (Power(1+(amt[CV(),'Annual_Interest']/100/12),
                amt[CV(),'Payments']) - 1)
       )
    )
  DIMENSION BY (customer cust, fact) measures (amt)                 --See 4
MAIN amortization
  PARTITION BY (customer c)                                        --See 5
  DIMENSION BY (0 p)                                               --See 6
  MEASURES (principalp pp, interestp ip, mort_balance m, customer mc)   --See 7
  RULES
    ITERATE(1000) UNTIL (ITERATION_NUMBER+1 =
r.amt[mc[0],'Payments'])                                           --See 8
    (ip[ITERATION_NUMBER+1] = m[CV()-1] *
       r.amt[mc[0], 'Annual_Interest']/1200,                       --See 9
     pp[ITERATION_NUMBER+1] = r.amt[mc[0], 'PaymentAmt'] - ip[CV()],    --See 10
      m[ITERATION_NUMBER+1] = m[CV()-1] - pp[CV()]                 --See 11
    )
ORDER BY c, p;
```

次の番号は、例で示した番号と対応しています。

1: これは、メイン・モデル定義の開始です。

2から4: これらの行では、参照モデルRの開始と終了をマークしています。このモデルは、各顧客のローンについて月々の支払額を計算するSELECT文を定義します。このSELECT文では、3のラベルが付けられた行で開始し、ルールが1つ指定された独自のMODEL句を使用します。このルールでは、表mortgage_factsの情報に基づいてamtの値を定義します。4のラベルが付けられた

行で定義した、顧客名custおよびファクトの値factによってディメンション化されている、参照モデルRによって戻されるメジャーはamtです。

参照モデルが計算されると、メイン・モデルでは、その値を使用して他の計算を実行します。参照モデルが計算されると、メイン・モデルでは、その値を使用して他の計算を実行します。参照モデルRは、mortgage_factの既存の各行の1行を戻し、さらに顧客ごとに新規に計算された行を戻します。この場合、ファクト・タイプはPaymentでamtが月々の支払額です。Rの出力から特定の金額を使用する場合は、式r.amt[<customer_name>,<fact_name>]で指定します。

5: これは、メイン・モデル定義の続きです。cに別名化したcustomerで出力をパーティション化します。

6: メイン・モデルを定数値0でディメンション化し、pに別名化します。これは、行の支払回数を表します。

7: メジャーを4つ定義しています。principalp (pp)は月々のローンに適用される元本金額、interestp (ip)は月々に支払われる利子、mort_balance (m)はローンの支払後における貸付金の残高値を示します。customer (mc)はパーティション化をサポートするために使用されます。

8: ルールのブロックを開始します。ルールの計算は、最大1000回実行されます。計算は顧客ごとに各月につき1回実行されるため、ローンに指定できる最大の月数は1000です。反復処理は、ITERATION_NUMBER+1が参照Rから導出される支払額に等しくなった時点で停止します。参照Rの値は、REFERENCE句で定義したamt(金額)メジャーであることに注意してください。この参照値は、r.amt[<customer_name>,<fact>]として処理されます。反復行で使用される式"r.amt[mc[0], 'Payments']"は、参照Rからの金額に解決し、この場合、顧客名はmc[0]で解決される値です。各パーティションに含まれる顧客は1つのみのため、mc[0]が持つことができる値は1つのみです。これにより、"r.amt[mc[0], 'Payments']"から、REFERENCE句の現行の顧客に関する支払回数の値が得られます。つまり、顧客の支払回数と同じ回数でルールが実行されます。

9から11: このブロックの最初の2つのルールは、8で説明したr.amt参照と同じタイプです。違いは、ルールipがファクトの値をAnnual_Interestとして定義している点です。各ルールは、他のいずれかのメジャー値を参照することに注意してください。各ルールの左辺で使用されている式"[ITERATION_NUMBER+1]"によって、新しいディメンション値が作成され、メジャーが結果セットにアップサートされます。これにより、各顧客のすべての支払について月々の割賦償還額を示す行が結果に含まれるようになります。

この例の最終行によって、顧客別およびローンの支払回数別に結果がソートされます。

23 高度な分析SQL

この章では、高度なビジネス・インテリジェンス問合せを実行するためのテクニックについて説明します。この章は、様々なSQL機能を組み合せて複雑な分析を実行する方法をより深く理解することを目的としています。この章で扱う機能は、「データ・ウェアハウスにおける集計のためのSQL」、「分析計算およびレポート用SQL関数」および「モデリングのSQL」でも個別に説明していますが、各機能を個々に見ていくだけでは、それらを組み合せて使用する方法を十分に理解することはできません。ここでは、各機能を組み合せることによって、どのような分析が可能になるかを示します。

「高度な」ビジネス・インテリジェンス問合せとは一体何でしょうか。「高度な」という形容詞が最もふさわしいのは、複数のディメンション階層を伴うことの多い複数ステップの問合せです。複数ステップの問合せでは、最終的な結果は、複数セットの取得データおよび複数の計算ステップによって導かれ、取得されるデータには、ディメンション階層の複数のレベルが含まれる場合があります。高度な問合せの主な例としては、複数の条件に基づく市場シェアの計算や、データのギャップ補完を必要とする売上予測があげられます。

この章では、ネストしたインライン・ビュー、CASE式、パーティション外部結合、MODEL句、WITH句、分析SQL関数などの使用方法を、例を示しながら説明します。必要に応じて、問合せ計画についても説明します。この章の内容は次のとおりです。

- ビジネス・インテリジェンス問合せの例

23.1 ビジネス・インテリジェンス問合せの例

この章の問合せは、様々なビジネス・インテリジェンス・タスクを示しています。これらの問合せのトピックおよび各問合せで使用されている機能は次のとおりです。

- 複雑な複数ステップの条件に基づく市場シェアの変化率。ネストしたインライン・ビュー、CASE式および分析SQL関数の使用方法を示します。

  ビジネス・インテリジェンス問合せの例1: 計算セット内での製品の市場シェアの変化率を参照してください。

- データのギャップ補完を伴う売上予測。MODEL句をパーティション外部結合およびCASE式と組み合せて使用する方法を示します。

  ビジネス・インテリジェンス問合せの例2: 欠損データを補完する売上予測を参照してください。

- 購入額別のバケットに顧客をグループ化する顧客分析。WITH句(問合せの副ファクタリング)と、分析SQL関数percentile_contおよびwidth_bucketの使用方法を示します。

  ビジネス・インテリジェンス問合せの例3: 顧客のバケットへのグループ化による顧客分析を参照してください

- 項目セットにグループ化する顧客項目。DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONALをテーブル・ファンクションとして使用する高頻度項目セットの計算方法を示します。

  ビジネス・インテリジェンス問合せの例4: 高頻度項目セットを参照してください。

23.1.1 ビジネス・インテリジェンス問合せの例1: 計算セット内での製品の市場シェアの変化率

この例では、収益が20%以上増加したアカウントについて、現在の3か月期間における売上上位20%を占める製品グループの市場シェアの、前年の同期間との変化率を求めます。

ここでは、市場シェアを「総売上高に占める当該製品の売上高の割合」と定義します。このように定義するのは、shサンプル・スキーマに競合企業のデータがなく、自社製品と競合企業製品の売上を比較する通常のシェア計算を行うことができないためで

す。ただし、ここでのシェア計算に必要な作業は、競合的な市場シェアの計算と論理的にはそれほど変わりません。

次に、この問合せで取得する情報を示します。これらの情報は、この順序で取得する必要があります。

1. 指定した3か月期間の間に、前年の同期間と比較して購入額が20%以上増加した都市。都市は1か国に限定され、売上に宣伝は関与していないことに注意してください。
2. 前のステップで求めた都市グループにおける、売上上位20%の製品。つまり、この顧客グループ全体での製品別総売上高を求め、最も売上の多い製品20%を選択します。
3. 前のステップで求めた各製品の売上高の割合。つまり、前のステップで求めた製品グループを使用して、全製品の総売上高に占める各製品の売上高の割合を求めます。前年の同期間における割合を求め、2つの年における割合の変化率を計算します。

この例で使用しているテクニックは次のとおりです。

- この問合せは、WITH句およびネストしたインライン・ビューを使用して実行されます。各インライン・ビューにはそのデータ要素の内容を示す別名が付けられており、各インライン・ビューの境界はコメント行で示されています。インライン・ビューは高機能ですが、WITH句を最大限に使用するような形で問合せを構成すると、可読性およびメンテナンス性が大幅に向上します。
- この問合せでは、WITH句の使用を控えめにしています。一部のネストしたインライン・ビューは、WITH句の独立した副次句として記述することも可能です。たとえば、メインの問合せでは、1つの値を戻すのに2つのインライン・ビューを使用しています。これらのインライン・ビューは、シェア計算の分母を戻すために使用されています。ここでインライン・ビューを使用せずに、その処理をWITH句で行うようにすると、可読性が向上します。顧客の購買分析に関するビジネス・インテリジェンス問合せの例3: 顧客のバケットへのグループ化による顧客分析を参照して、WITH句を最大限に使用する問合せと比較してください。
- SUM関数に対する引数の中で、CASE式を使用している点に注目してください。これらのCASE式をWHERE句の後に追加のデータ・フィルタとして機能させることで、SQLの記述が簡潔になっています。またこれらの式によって、指定の日付の売上列と、別の日付の別の列を合計することが可能になっています。

```
WITH prod_list AS                       --START: Top 20% of products
   ( SELECT prod_id prod_subset, cume_dist_prod
    FROM                                --START: All products Sales for city subset
    ( SELECT s.prod_id, SUM(amount_sold),
         CUME_DIST() OVER (ORDER BY SUM(amount_sold)) cume_dist_prod
      FROM sales s, customers c, channels ch,  products p, times t
      WHERE s.prod_id = p.prod_id AND p.prod_total_id = 1 AND
            s.channel_id = ch.channel_id AND ch.channel_total_id = 1 AND
            s.cust_id = c.cust_id AND
            s.promo_id = 999 AND
            s.time_id  = t.time_id AND t.calendar_quarter_id = 1776 AND
            c.cust_city_id IN
       (SELECT cust_city_id             --START: Top 20% of cities
       FROM
         (
           SELECT cust_city_id, ((new_cust_sales - old_cust_sales)
                / old_cust_sales ) pct_change, old_cust_sales
           FROM
           (
              SELECT cust_city_id, new_cust_sales, old_cust_sales
              FROM
              (          --START: Cities AND sales for 1 country in 2 periods
                SELECT cust_city_id,
                  SUM(CASE WHEN t.calendar_quarter_id = 1776
                     THEN amount_sold  ELSE  0  END ) new_cust_sales,
                  SUM(CASE WHEN t.calendar_quarter_id = 1772
```

```
                    THEN amount_sold ELSE 0 END) old_cust_sales
               FROM sales s, customers c, channels ch,
                    products p, times t
               WHERE s.prod_id = p.prod_id AND p.prod_total_id = 1 AND
                     s.channel_id = ch.channel_id AND ch.channel_total_id = 1 AND
                     s.cust_id = c.cust_id AND c.country_id = 52790 AND
                     s.promo_id = 999 AND
                     s.time_id  = t.time_id AND
                    (t.calendar_quarter_id = 1776 OR t.calendar_quarter_id =1772)
               GROUP BY cust_city_id
             ) cust_sales_wzeroes
             WHERE old_cust_sales > 0
          )  cust_sales_woutzeroes
         )         --END: Cities and sales for country in 2 periods
         WHERE old_cust_sales > 0 AND  pct_change >= 0.20)
                                  --END: Top 20% of cities
GROUP BY s.prod_id
)  prod_sales                     --END: All products sales for city subset
    WHERE cume_dist_prod > 0.8    --END: Top 20% products
)
                                  --START: Main query bloc
SELECT  prod_id, ( (new_subset_sales/new_tot_sales)
              -    (old_subset_sales/old_tot_sales)
                 ) *100  share_changes
FROM
(                                 --START: Total sales for country in later period
  SELECT  prod_id,
     SUM(CASE WHEN t.calendar_quarter_id = 1776
                   THEN amount_sold  ELSE  0  END )  new_subset_sales,
        (SELECT SUM(amount_sold) FROM sales s, times t, channels ch,
                customers c, countries co, products p
          WHERE s.time_id  = t.time_id AND t.calendar_quarter_id = 1776 AND
                s.channel_id = ch.channel_id AND ch.channel_total_id = 1 AND
                s.cust_id = c.cust_id AND
                c.country_id = co.country_id AND co.country_total_id = 52806 AND
                s.prod_id = p.prod_id AND p.prod_total_id = 1 AND
                s.promo_id = 999
        )   new_tot_sales,

                                 --END: Total sales for country in later period
                                 --START: Total sales for country in earlier period
     SUM(CASE WHEN t.calendar_quarter_id = 1772
                   THEN amount_sold  ELSE  0  END)  old_subset_sales,
        (SELECT SUM(amount_sold) FROM sales s, times t, channels ch,
                customers c, countries co, products p
          WHERE s.time_id  = t.time_id AND t.calendar_quarter_id = 1772 AND
                s.channel_id = ch.channel_id AND ch.channel_total_id = 1 AND
                s.cust_id = c.cust_id AND
                c.country_id = co.country_id AND co.country_total_id = 52806 AND
                       s.prod_id = p.prod_id AND p.prod_total_id = 1 AND
                s.promo_id = 999
        )   old_tot_sales
                                 --END: Total sales for country in earlier period
 FROM sales s, customers c, countries co, channels ch, times t
 WHERE s.channel_id = ch.channel_id AND ch.channel_total_id = 1 AND
       s.cust_id = c.cust_id AND
       c.country_id = co.country_id AND co.country_total_id = 52806 AND
       s.promo_id = 999 AND
       s.time_id  = t.time_id AND
      (t.calendar_quarter_id = 1776 OR t.calendar_quarter_id = 1772)
```

```
        AND s.prod_id IN
    (SELECT prod_subset FROM prod_list)
  GROUP BY prod_id);
```

23.1.2 ビジネス・インテリジェンス問合せの例2: 欠落データを埋める売上予測

この問合せは、2000年と2001年の売上に基づいて2002年の売上を予測します。2000年と2001年の売上について最大の変化率を求めてから、これを2002年の売上に追加します。これは単純な計算ですが、1つ注意が必要な点があります。それは、多くの製品において、2000年と2001年に売上データのない月が存在することです。これらの空白値には、実際の売上が存在する月から算出した、その年の平均売上値を埋め込みます。またこの問合せでは、国別の通貨値をUSドルに変換します。最終的に、この問合せは2002年の予測値のみを戻します。

この例で使用しているテクニックは次のとおりです。

- MODEL句の前に、クロス結合を使用して考え得るすべてのデータ行を事前定義することで、MODEL句で必要となる処理の量を減らしています。
- 通貨の変換を行うために、MODEL句で参照モデルを使用しています。
- CV関数を多用することで、必要なルールの総数を3つにまで減らしています。
- 最も興味深い処理を行っているのは最後のルールです。ここでは、ネストしたルールを使用して通貨の変換係数を求めています。この式で必要となる国名を提供するため、参照モデルとメイン・モデルの両方で、国をそれぞれディメンションc、メジャーccとして定義しています。

この例の処理は、通貨の変換係数の参照表を作成するところから始まります。この表は、各国の各月の変換係数を格納します。表に挿入する行を指定するために、クロス結合を使用している点に注目してください。変換係数を設定する国は、例の目的に合せて1か国(カナダ)のみとしています。

```
CREATE TABLE currency (
   country         VARCHAR2(20),
   year            NUMBER,
   month           NUMBER,
   to_us           NUMBER);

INSERT INTO currency
(SELECT distinct
SUBSTR(country_name,1,20), calendar_year, calendar_month_number, 1
FROM countries
CROSS JOIN times t
WHERE calendar_year IN (2000,2001,2002)
);
UPDATE currency set to_us=.74 WHERE country='Canada';
```

次に示すのは、予測用の問合せです。まず、2つの副次句を含むWITH句を指定しています。最初の副次句では、2000年、2001年および2002年の製品ごとの月次売上高を国別に求めています。2つ目の副次句では、月レベルの個別時間値のリストを求めています。

```
WITH  prod_sales_mo AS       --Product sales per month for one country
(
SELECT country_name c, prod_id p, calendar_year  y,
   calendar_month_number  m, SUM(amount_sold) s
FROM sales s, customers c, times t, countries cn, promotions p, channels ch
WHERE  s.promo_id = p.promo_id AND p.promo_total_id = 1 AND
       s.channel_id = ch.channel_id AND ch.channel_total_id = 1 AND
       s.cust_id=c.cust_id  AND
       c.country_id=cn.country_id AND country_name='France' AND
```

```
       s.time_id=t.time_id  AND t.calendar_year IN  (2000, 2001,2002)
GROUP BY cn.country_name, prod_id, calendar_year, calendar_month_number
),
                    -- Time data used for ensuring that model has all dates
time_summary AS(  SELECT DISTINCT calendar_year cal_y, calendar_month_number cal_m
  FROM times
  WHERE  calendar_year IN  (2000, 2001, 2002)
)
                   --START: main query block
SELECT c, p, y, m, s,  nr FROM (
SELECT c, p, y, m, s,  nr
FROM prod_sales_mo s
                   --Use partitioned outer join to make sure that each combination
                   --of country and product has rows for all month values
  PARTITION BY (s.c, s.p)
     RIGHT OUTER JOIN time_summary ts ON
        (s.m = ts.cal_m
         AND s.y = ts.cal_y
        )
MODEL
  REFERENCE curr_conversion ON
      (SELECT country, year, month, to_us
      FROM currency)
      DIMENSION BY (country, year y,month m) MEASURES (to_us)
                                --START: main model
   PARTITION BY (s.c c)
   DIMENSION BY (s.p p, ts.cal_y y, ts.cal_m m)
   MEASURES (s.s s, CAST(NULL AS NUMBER) nr,
             s.c cc ) --country is used for currency conversion
   RULES (
                      --first rule fills in missing data with average values
      nr[ANY, ANY, ANY]
         = CASE WHEN s[CV(), CV(), CV()] IS NOT NULL
              THEN s[CV(), CV(), CV()]
              ELSE ROUND(AVG(s)[CV(), CV(), m BETWEEN 1 AND 12],2)
           END,
                      --second rule calculates projected values for 2002
      nr[ANY, 2002, ANY] = ROUND(
         ((nr[CV(),2001,CV()] - nr[CV(),2000, CV()])
          / nr[CV(),2000, CV()]) * nr[CV(),2001, CV()]
         + nr[CV(),2001, CV()],2),
                      --third rule converts 2002 projections to US dollars
      nr[ANY,y != 2002,ANY]
         = ROUND(nr[CV(),CV(),CV()]
           * curr_conversion.to_us[ cc[CV(),CV(),CV()], CV(y), CV(m)], 2)
)
ORDER BY c, p, y, m)
WHERE y = '2002'
ORDER BY c, p, y, m;
```

23.1.3 ビジネス・インテリジェンス問合せの例3: 顧客のバケットへのグループ化による顧客分析

顧客を理解する1つの方法として重要なのは、顧客の購買パターンを調べ、顧客ごとの利益率を明らかにすることです。これは、その顧客に営業活動を行う価値があるかどうか、どのような活動が有効かを判断するのに役立ちます。shサンプル・スキーマのデータセットには多数の顧客が含まれているため、利益率の分析は、高レベルのビューから始めるのが適しています。ここでは、顧客利益率のヒストグラムのためのデータを求め、利益率を10の範囲(ヒストグラム分析では通常「バケット」と呼ばれる)に分

割します。国ごとに、月の集計レベルで次のデータを求めます。

- 10の等幅バケットからなる、顧客利益率のヒストグラムに必要なデータ。つまり、10の利益率バケットのそれぞれにグループ化される顧客の数を求めます。結果として取得される行は10行のみですが、大がかりな計算が必要になります。

利益率バケットごとに、次のデータも求めます。

- 顧客ごとの月間トランザクション数の中央値(同日の単一の顧客による単一のチャネルを介した購買を1つのトランザクションとして扱う)
- 顧客ごとの(各国通貨での)トランザクション・サイズの中央値
- 利益が最大の製品と最小の製品
- トランザクション数の中央値とトランザクション・サイズの中央値の、前年からの変化率

この例で使用しているテクニックは次のとおりです。

- WITH句を使用して問合せの可読性を向上させます。必要なデータを論理的なチャンクに分割し、各チャンクを専用のWITHの副次句で記述することで、ネストしたインライン・ビューと比較して可読性およびメンテナンス性が大幅に向上しています。WITH句を最大限に使用すると、メインのSELECT句で、取得したデータに対する計算を行わずに済みます。このことも、問合せの可読性およびメンテナンス性の向上に寄与しています。
- 等幅のヒストグラム・バケットを得るためのwidth_bucket、およびトランザクション・サイズとトランザクション数の中央値を求めるためのpercentile_contという2つの分析SQL関数を使用します。

この問合せには、データ・ウェアハウスの設計に起因する分析上の問題点が含まれています。それは、shのデータには各トランザクションのエントリおよびトランザクション数のエントリが含まれていないため、これらの数値については仮定を設定する必要があるということです。この問合せでは、最小主義に則って、同じ日に単一の顧客が単一のチャネルを介して購入した製品をすべて同じトランザクションに属するものとみなします。実際には、同じ日に同じチャネルで複数の購買を行っている顧客もいるはずなので、このアプローチでは、トランザクション数は必然的に実際よりも少なくなります。

次に示す問合せは、先頭部分cust_prod_mon_profitに対するマテリアライズド・ビューを作成してから実行する必要があることに注意してください。また、そのマテリアライズド・ビューを作成する前に、索引を2つ作成しておくことも必要です。このような準備ステップをしないままこの問合せを実行すると、場合によっては終了までにかなりの時間がかかります。必要となる2つの索引および問合せの主要部分は次のとおりです。

```
CREATE BITMAP INDEX costs_chan_bix
      ON costs (channel_id)
      LOCAL NOLOGGING COMPUTE STATISTICS;

CREATE BITMAP INDEX costs_promo_bix
      ON costs (promo_id)
      LOCAL NOLOGGING COMPUTE STATISTICS;

WITH cust_prod_mon_profit AS
-- profit by cust, prod, day, channel, promo
  (SELECT s.cust_id, s.prod_id, s.time_id,
          s.channel_id, s.promo_id,
          s.quantity_sold*(c.unit_price-c.unit_cost) profit,
          s.amount_sold dol_sold, c.unit_price price, c.unit_cost cost
   FROM sales s, costs c
   WHERE s.prod_id=c.prod_id
     AND s.time_id=c.time_id
     AND s.promo_id=c.promo_id
     AND s.channel_id=c.channel_id
     AND s.cust_id in (SELECT cust_id FROM customers cst
                       WHERE cst.country_id = 52770
```

```
     AND s.time_id IN (SELECT time_id FROM times t
                       WHERE t.calendar_month_desc = '2000-12'
   ),
-- Transaction Definition:  All products sold through a single channel to a
-- single cust on a single day are assumed to be sold in 1 transaction.
-- Some products in a transacton
-- may be on promotion
-- A customers  daily transaction amount is the sum of ALL products
-- purchased in the same channel in the same day
cust_daily_trans_amt  AS
(  SELECT cust_id, time_id, channel_id,  SUM(dol_sold) cust_daily_trans_amt
   FROM  cust_prod_mon_profit
   GROUP BY cust_id, time_id, channel_id
--A customers  monthly transaction count is the count of all channels
--used to purchase items in the same day, over all days in the month.
--It is really a count of the minimum possible number of transactions
cust_purchase_cnt  AS(  SELECT cust_id,  COUNT(*) cust_purchase_cnt
   FROM  cust_daily_trans_amt
   GROUP BY cust_id
),
--  Total profit for a customer over 1 month
cust_mon_profit AS
( SELECT cust_id, SUM(profit) cust_profit
       FROM  cust_prod_mon_profit
       GROUP BY cust_id
-- Minimum and maximum profit across all customer
-- sets endpoints for histogram data.
min_max_p AS
-- Note max profit + 0.1 to allow 10th bucket to include max value
(SELECT 0.1 + MAX(cust_profit) max_p, MIN(cust_profit) min_p
FROM cust_mon_profit),
-- Profitability bucket found for each customer
cust_bucket AS
(SELECT cust_id, cust_profit,
   width_bucket(cust_profit,
         min_max_p.min_p,
FROM cust_mon_profit,  min_max_p
-- Aggregated data needed for each bucket
histo_data AS
(  SELECT bucket,
     bucket*(( max_p-min_p) /10) top_end , count(*)  histo_count
   FROM  cust_bucket, min_max_p
   GROUP BY bucket, bucket*(( max_p - min_p) /10)
-- Median count of transactions per cust per month median_trans_count AS
-- Find median count of transactions per cust per month
(SELECT cust_bucket.bucket,
      PERCENTILE_CONT(0.5) WITHIN GROUP
          (ORDER BY cust_purchase_cnt.cust_purchase_cnt) median_trans_count
    FROM cust_bucket, cust_purchase_cnt
    WHERE cust_bucket.cust_id=cust_purchase_cnt.cust_id
    GROUP BY cust_bucket.bucket
-- Find Mmedian transaction size for custs by profit bucket
cust_median_trans_size AS
(  SELECT cust_bucket.bucket,
      PERCENTILE_CONT(0.5) WITHIN GROUP
          (ORDER BY cust_daily_trans_amt.cust_daily_trans_amt)
           cust_median_trans_ size
    FROM cust_bucket, cust_daily_trans_amt
    WHERE cust_bucket.cust_id=cust_daily_trans_amt.cust_id
    GROUP BY cust_bucket.bucket
```

```
-- Profitability of each product sold within each bucket
bucket_prod_profits AS
(  SELECT  cust_bucket.bucket, prod_id, SUM(profit) tot_prod_profit
   FROM  cust_bucket, cust_prod_mon_profit
   WHERE  cust_bucket.cust_id=cust_prod_mon_profit.cust_id
   GROUP BY cust_bucket.bucket, prod_id
),  -- Most and least profitable product by bucket
prod_profit AS
(  SELECT bucket, MIN(tot_prod_profit) min_profit_prod,
                  MAX(tot_prod_profit) max_profit_prod
   FROM bucket_prod_profits
   GROUP BY bucket
-- Main query block
SELECT  histo_data.bucket, histo_data.histo_count,
        median_trans_count.median_trans_count,
        cust_median_trans_size.cust_median_trans_size,
        prod_profit.min_profit_prod, prod_profit.max_profit_prod
FROM  histo_data, median_trans_count, cust_median_trans_size,
      prod_profit
WHERE  histo_data.bucket=median_trans_count.bucket
  AND  histo_data.bucket=cust_median_trans_size.bucket
  AND  histo_data.bucket=prod_profit.bucket;
```

23.1.4 ビジネス・インテリジェンス問合せの例4: 高頻度項目セット

あるマーケティング・マネージャが、自社製品の各種ホワイト・ペーパーについて、ユーザーがそれぞれのセッションでどのような種類のものをダウンロードするかを把握しようとしているとします。その場合このマネージャが必要としている情報は、ホワイト・ペーパーのどのようなグルーピングが頻度の高い項目であるかということです。そのような情報は、ダウンロードされたホワイト・ペーパーごとにユーザーIDとセッションIDがWebサイトのアクティビティ・ログに記録されていれば、すでに組み込まれている高頻度項目機能を使用することで容易に取得できます。まず、個々のホワイト・ペーパーについてダウンロード数を集計したリストを次に示します。(この例では、Oracle製品のホワイト・ペーパー名を使用しています。)

```
White paper titles                                                  #
------------------------------------------------------------        ----
Table Compression in Oracle Database 10g                             696
Field Experiences with Large Data Warehouses                         439
Key Data Warehouse Features: A Comparative Performance Analysis      181
Materialized Views in Oracle Database 10g                            167
Parallel Execution in Oracle Database 10g                            166
```

次に示すのは、このような分析に使用できる問合せの一例です。この問合せには、テーブル・ファンクションとしてDBMS_FREQUENT_ITEMSET.FI_TRANSACTIONALが使用されています。問合せの構造の詳細は、『*Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス*』を参照してください。この問合せでは、同一のセッションの中でダウンロードされたホワイト・ペーパーの組合せの項目セットが戻されます。

```
SELECT itemset, support, length, rnk
FROM
  (SELECT itemset, support, length,
   RANK() OVER (PARTITION BY length ORDER BY support DESC) rnk
FROM
(SELECT CAST(itemset AS fi_char) itemset, support, length, total_tranx
 FROM table(DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL
  (CURSOR(SELECT session_id, command
          FROM web_log
          WHERE time_stamp BETWEEN '01-APR-2002' AND '01-JUN-2002'),
          (60/2600), 2, 2, CURSOR(SELECT 'a' FROM DUAL WHERE 1=0),
          CURSOR(SELECT 'a' FROM DUAL WHERE 1=0)))))
```

```
   WHERE rnk <= 10;
```

上位3項目が次のような結果であったとします。

```
White paper titles                                                  #
------------------------------------------------------------      -----
Table Compression in Oracle Database 10g                            115
Field Experiences with Large Data Warehouses

Data Warehouse Performance Enhancements with Oracle Database 10g    109
Oracle Performance and Scalability in DSS Environments

Materialized Views in Oracle Database 10g                           107
Query Optimization in Oracle Database 10g
```

この分析からは、興味深い結果が導き出されています。個々のホワイト・ペーパーのダウンロード数のリストを見ると、「Table Compression in Oracle10*g*」がすべてのホワイト・ペーパーの中で最上位にあるので、ダウンロード数の多かったホワイト・ペーパーの組合せでも、その大部分にこのホワイト・ペーパーが含まれていると予想されます。しかし、上位3つの組合せのうち、このホワイト・ペーパーが含まれる組合せは1つしかありません。

高頻度項目セットを使用してWebのログ情報を分析すれば、ダウンロード数の多いホワイト・ペーパーを表示するだけの簡単なレポートよりもはるかに多くの情報を抽出することが可能になります。この結果から、このWebサイトの訪問者はセッションごとに特定のテーマに沿った情報を探す傾向が強いということがわかります。スケーラビリティに関心のある訪問者は、当然、圧縮についてのホワイト・ペーパーと大規模データ・ウェアハウスについてのホワイト・ペーパーをダウンロードし、複合問合せ機能に関心のある訪問者は、問合せの最適化についてのホワイト・ペーパーとマテリアライズド・ビューについてのホワイト・ペーパーをダウンロードすると考えられます。このような情報は、マーケティング・マネージャにとっては、将来的にどのような内容のホワイト・ペーパーを執筆すればよいかを判断するのに役立ち、Webデザイナにとっては、Webサイトの構成についてのヒントとして役立ちます。

詳細は、SQL分析での高頻度項目セットを参照してください。

第V部 分析ビュー

分析ビューを使用すると、データベースの表およびビューの階層およびディメンションの大量のデータに対する複雑な分析問合せを簡単に作成できます。

分析ビューについては、次のトピックで説明しています。

- 分析ビューの概要
- 属性ディメンションおよび階層オブジェクト
- 分析ビュー・オブジェクト

24 分析ビューの概要

分析ビューは、メタデータ・オブジェクトであり、ユーザーがデータベースの表およびビューのデータに対する複雑な階層およびディメンションの問合せをすばやく簡単に作成できます。

次の項では、分析ビューの一般的な考慮事項について説明します。

- 分析ビューとは
- 分析ビューの権限
- 分析ビューのアプリケーション・プログラミング・インタフェース
- 分析ビューのコンパイル状態
- データの検証
- 分析ビューの分類
- アプリケーション・コンテナとの分析ビューの共有
- 分析ビュー・オブジェクトの変更または削除
- 例のデータとスクリプト

24.1 分析ビューとは

分析ビューは、既存のデータベース表およびビューに格納されているデータの分析問合せを迅速に効率的に作成する方法を提供します。

分析ビューは、ディメンション・モデルを使用してデータを編成します。これらを使用すると、集計および計算をデータ・セットに簡単に追加でき、比較的単純なSQLで問い合せることができるデータをビューに表示できます。

標準のリレーショナル・ビューと同様、分析ビューには次の特徴があります。

- メタデータ・オブジェクトです(データを格納しません)。
- SQLを使用して問い合せることができます。
- 表、ビュー、外部表などの他のデータベース・オブジェクトからデータにアクセスできます。
- 複数の表を単一のビューに結合できます。

分析ビューには、次の特徴もあります。

- ディメンションおよび階層概念を持つリッチ・ビジネス・モデルを使用してデータを編成します。
- 階層データを使用したシステム生成列を含みます。
- データを自動的に集計します。
- ビジネス・モデルに基づいた構文を使用して簡単に定義される埋込みメジャー計算を含みます。
- プレゼンテーション・メタデータを含みます。

分析ビューの定義には、ナビゲーション、結合、集計および計算のルールが含まれるため、これらのルールを問合せに含める必要がありません。結合、集計およびメジャー計算を表す単純な表や複雑なSELECT文を含めるのではなく、単純SQLを使用してスマート分析ビューを問い合せることができます。この方法には、次のようないくつかの利点があります。

- 簡略でより高速なアプリケーション開発。複雑なSELECT文を記述または生成するよりも、分析ビュー内で計算を定義

する方が簡単です。

- データベースで1回定義すると、多数のアプリケーションで再使用できる計算ルール。これにより、エンド・ユーザーは、一貫性のない結果を考慮することなくレポート作成ツールを自由に選択できます。

分析ビューは、特に次のユーザーに役立ちます。

- データ・ウェアハウスのアーキテクトまたは設計者
- ビジネス・インテリジェンス・アプリケーションの開発者
- データベース・アナリスト

データ・ウェアハウス・アーキテクトにとって、分析ビューは、データ・ウェアハウスのデータをアプリケーション開発者およびビジネス・ユーザーに表示するためのツールです。BIアプリケーションで提供されるツールでは、問合せの生成、データの取得および結果の表示を行います。

分析ビューのコンポーネント

分析ビューのコンポーネント・オブジェクトの構成内容は次のとおりです。

- 属性ディメンション。これは、表またはビューを参照し、列を属性やレベルなどの高レベルのオブジェクトに編成するメタデータ・オブジェクトです。ディメンションおよび階層に関連するメタデータの多くは、属性ディメンション・オブジェクトで定義されます。
- 階層。これは、属性ディメンション・オブジェクトを参照するビューおよび階層関係を使用してデータを編成するビューのタイプです。ディメンションおよび階層に関連するデータは、階層から選択されます。
- 分析ビュー・オブジェクト。これは、ファクト・データを表示するビューのタイプです。分析ビューは、ファクト表と階層の両方を参照します。階層とメジャー・データはどちらも分析ビューから選択できます。
- 導出分析ビューは、SELECT文のWITHおよびFROM句で定義され、既存の分析ビューに基づきます。

データ・ディクショナリ・ビュー(ALL_ANALYTIC_VIEW_COLUMNSなど)には、分析ビューのコンポーネント・オブジェクトのメタデータおよび他の情報が含まれます。

DBMS_HIERARCHY PL/SQLパッケージには、分析ビューおよび階層オブジェクトを検証するための関数、および検証関数によって生成されたメッセージの記録に使用できる表を作成するプロシージャが含まれます。

分析ビューのデータ・ソース

属性ディメンションおよび分析ビューは、通常、スター・スキーマ・ディメンション表およびファクト表をデータ・ソースとして使用します。大きいデータ・セットの場合、インメモリー列ストアの表により、分析ビューは最適な問合せパフォーマンスが得られます。分析ビューは、スノーフレーク・スキーマ、非正規化表、外部表およびリモート表とともに使用することもできます。

データ・ソースは、属性ディメンションまたは分析ビュー定義でusing_clauseを使用して指定します。データ・ソースの別名を指定できます。

データ・ソースへのアクセスに必要な権限を持つデータベース・ユーザーは、分析ビュー・オブジェクトを作成できます。作成者はビジネス・モデルを定義し、属性ディメンション、階層および分析ビューを作成して、データの問合せ方法およびこのモデルの実装方法を指定します。

マテリアライズド・ビューと分析ビュー

分析ビューまたは階層の問合せにまたがるマテリアライズド・ビューの作成は、サポートされていません。マテリアライズド・ビューは、分析ビューのcache_clauseのMEASURE_GROUP句で使用できます。

分析ビュー・オブジェクトの制約

分析ビューの問合せで最適な問合せパフォーマンスを得るには、スター・スキーマの問合せに通常使用する制約と同じ制約を使用する必要があります。属性ディメンションまたは分析ビューでは、ソース表またはビューで特定の制約を定義または有効化する必要はありません。また、属性ディメンションまたは分析ビューの定義により、これらの表またはビューに追加の制約は導入されません。階層の属性ディメンションまたは分析ビューで使用される表またはビューのデータが、メタデータ定義に固有の論理制約に準拠していることを検証するために、PL/SQL関数VALIDATE_HIERARCHYおよびVALIDATE_ANALYTIC_VIEWを使用できます。

分析ビューの命名規則

属性ディメンション、階層および分析ビューの命名規則、およびこれらのコンポーネント(属性、レベル、メジャーなど)の命名規則は、標準のデータベース識別子ルールに従います。識別子を囲むには、二重引用符を使用できます(拡張文字および大文字小文字混在など)。それ以外の場合は、標準の大文字および限られた文字ルールが適用されます。

24.2 分析ビューの権限

分析ビュー、属性ディメンションおよび階層で使用可能なシステム権限およびオブジェクト権限について説明します。

システム権限

次のシステム権限により、ユーザーは分析ビューのコンポーネント・オブジェクトを作成、変更または削除できます。

システム権限	説明
CREATE ANALYTIC VIEW	権限を付与したスキーマ内での分析ビューの作成。
CREATE ANY ANALYTIC VIEW	SYS を除く任意のスキーマ内での分析ビューの作成。
CREATE ATTRIBUTE DIMENSION	権限を付与したスキーマ内での属性ディメンションの作成。
CREATE ANY ATTRIBUTE DIMENSION	SYS を除く任意のスキーマ内での属性ディメンションの作成。
CREATE HIERARCHY	権限を付与したスキーマ内での階層の作成。
CREATE ANY HIERARCHY	SYS を除く任意のスキーマ内での階層の作成。
ALTER ANY ANALYTIC VIEW	SYS を除く任意のスキーマ内での分析ビューの名前の変更。
ALTER ANY ATTRIBUTE DIMENSION	SYS を除く任意のスキーマ内での属性ディメンションの名前の変更。
ALTER ANY HIERARCHY	SYS を除く任意のスキーマ内での階層の名前の変更。
DROP ANY ANALYTIC VIEW	SYS を除く任意のスキーマ内での分析ビューの削除。
DROP ANY ATTRIBUTE DIMENSION	SYS を除く任意のスキーマ内での属性ディメンションの削除。
DROP ANY HIERARCHY	SYS を除く任意のスキーマ内での階層の削除。

システム権限	説明
SELECT ANY TABLE	任意のスキーマ内での任意の分析ビューまたは階層の問合せまたは表示。

オブジェクト権限

次のオブジェクト権限により、ユーザーは分析ビューのコンポーネント・オブジェクトを問い合せたり、名前を変更できます。

オブジェクト権限	許可される操作
ALTER	分析ビュー、属性ディメンションまたは階層の名前の変更。
READ	SELECT 文でのオブジェクトの問合せ。
SELECT	SELECT 文でのオブジェクトの問合せ。

例24-1 システム権限の付与

次の文は、CREATEシステム権限をユーザーav_userに付与します。

```
GRANT CREATE ATTRIBUTE DIMENSION TO av_user;
GRANT CREATE HIERARCHY TO av_user;
GRANT CREATE ANALYTIC VIEW TO av_user;
GRANT SELECT ANY TABLE TO av_user;
```

例24-2 オブジェクト権限の付与

次の文は、すべてのオブジェクト権限をユーザーav_user2に付与し、ALTER権限を取り消します。

```
GRANT ALL ON "AV_USER".SALES_AV TO "AV_USER2";
REVOKE ALTER ON "AV_USER".SALES_AV FROM "AV_USER2";
```

24.3 分析ビューのアプリケーション・プログラミング・インタフェース

分析ビューのアプリケーション・プログラミング・インタフェースは、SQL DDL文、PL/SQLプロシージャと関数およびデータ・ディクショナリ・ビューで構成されています。

これらのインタフェースは、次の項に示されています。

- 分析ビューを作成および管理するためのSQL DDL文
- 分析ビュー用のPL/SQLパッケージ
- 分析ビューのデータ・ディクショナリ・ビュー

分析ビューを作成および管理するためのSQL DDL文

分析ビュー・オブジェクトは、次のSQL DDL文を使用して作成および管理できます。

- CREATE ANALYTIC VIEW
- CREATE ATTRIBUTE DIMENSION
- CREATE HIERARCHY
- ALTER ANALYTIC VIEW
- ALTER ATTRIBUTE DIMENSION
- ALTER HIERARCHY
- DROP ANALYTIC VIEW

- DROP ATTRIBUTE DIMENSION
- DROP HIERARCHY

これらの文の詳細は、*Oracle Database SQL*言語リファレンスのCREATE ANALYTIC VIEWおよびその他の文を参照してください。

フィルタ処理されたファクトおよび追加メジャーのSQL SELECT文の句

SELECT文のWITHおよびFROM句では、階層のメジャー値の集計前に階層メンバーをフィルタ処理する1つ以上の一時分析ビューを定義できます。問合せで使用される追加のメジャーを定義することもできます。フィルタ処理されたファクトおよび追加メジャーは既存の永続分析ビューに基づいていますが、これらは永続分析ビューの定義そのものを変更するわけではありません。

関連項目:

フィルタ処理されたファクトおよび追加メジャーによる分析ビュー問合せ

分析ビュー用のPL/SQLパッケージ

分析ビュー・オブジェクトおよび階層オブジェクトのデータは、DBMS_HIERARCHYパッケージの次のプロシージャを使用して検証できます。

- CREATE_VALIDATE_LOG_TABLEプロシージャ
- VALIDATE_ANALYTIC_VIEW関数
- VALIDATE_CHECK_SUCCESS関数
- VALIDATE_HIERARCHY関数

このパッケージの詳細は、*Oracle Database PL/SQL*パッケージおよびタイプ・リファレンスのDBMS_HIERARCHYに関する項を参照してください。

分析ビューのデータ・ディクショナリ・ビュー

次のデータ・ディクショナリ・ビューには、分析ビュー・オブジェクトに関する情報が含まれています。接頭辞ALLが付いたビューのみが表示されます。また、各ビューには、対応するDBAバージョンとUSERバージョンがあります。

分析ビューのビュー

- ALL_ANALYTIC_VIEW_ATTR_CLASS
- ALL_ANALYTIC_VIEW_BASE_MEAS
- ALL_ANALYTIC_VIEW_CALC_MEAS
- ALL_ANALYTIC_VIEW_CLASS
- ALL_ANALYTIC_VIEW_COLUMNS
- ALL_ANALYTIC_VIEW_DIM_CLASS
- ALL_ANALYTIC_VIEW_DIMENSIONS
- ALL_ANALYTIC_VIEW_HIER_CLASS
- ALL_ANALYTIC_VIEW_HIERS
- ALL_ANALYTIC_VIEW_KEYS
- ALL_ANALYTIC_VIEW_LEVEL_CLASS
- ALL_ANALYTIC_VIEW_LEVELS
- ALL_ANALYTIC_VIEW_LVLGRPS
- ALL_ANALYTIC_VIEW_MEAS_CLASS

- ALL_ANALYTIC_VIEWS

属性ディメンションのビュー

- ALL_ATTRIBUTE_DIM_ATTR_CLASS
- ALL_ATTRIBUTE_DIM_ATTRS
- ALL_ATTRIBUTE_DIM_CLASS
- ALL_ATTRIBUTE_DIM_JOIN_PATHS
- ALL_ATTRIBUTE_DIM_KEYS
- ALL_ATTRIBUTE_DIM_LEVEL_ATTRS
- ALL_ATTRIBUTE_DIM_LEVELS
- ALL_ATTRIBUTE_DIM_LVL_CLASS
- ALL_ATTRIBUTE_DIM_ORDER_ATTRS
- ALL_ATTRIBUTE_DIM_TABLES
- ALL_ATTRIBUTE_DIMENSIONS

階層のビュー

- ALL_HIER_CLASS
- ALL_HIER_COLUMNS
- ALL_HIER_HIER_ATTR_CLASS
- ALL_HIER_HIER_ATTRIBUTES
- ALL_HIER_JOIN_PATHS
- ALL_HIER_LEVEL_ID_ATTRS
- ALL_HIER_LEVELS
- ALL_HIERARCHIES

これらのビューの詳細は、*Oracle Database*リファレンスのALL_ANALYTIC_VIEWSおよびその他のビューを参照してください。

24.4 分析ビューのコンパイル状態

属性ディメンション、階層または分析ビューを作成または変更すると、Oracle Databaseで、オブジェクトのメタデータの内部の妥当性が確認されます。

分析ビューのSQL DDL CREATEおよびALTER文には、FORCEおよびNOFORCEオプションがあり、デフォルトではNOFORCEが使用されます。別のオブジェクトに依存するメタデータの検証はオプションで、FORCEおよびNOFORCEオプションによって決定されます。

NOFORCEを指定してコンパイルに失敗すると、CREATEまたはALTER操作が失敗し、エラーが発生します。FORCEを指定すると、コンパイルに失敗してもCREATEまたはALTERは成功します。

COMPILEキーワードを指定すると、コンパイルを明示的に起動できます。コンパイルは、問合せ中に必要に応じて暗黙的に起動されます。オブジェクトがコンパイルされない場合は、問合せによってエラーが返され、暗黙的にコンパイルできません。

コンパイル状態は、ALL_ATTRIBUTE_DIMENSIONS、ALL_HIERARCHIESおよびALL_ANALYTIC_VIEWSデータ・ディクショナリ・ビュー(および対応するDBAおよびUSERビュー)のCOMPILE_STATE列に記録されます。状態は次のいずれかです。

値	説明

値	説明
VALID	オブジェクトがエラーなしでコンパイルされました。
INVALID	一部の変更に再コンパイルが必要か、オブジェクトはコンパイルされたがエラーが発生しました。

分析ビュー・オブジェクトに対するSQL DDL操作により、依存オブジェクトの状態がINVALIDに変更されます。たとえば、属性ディメンションに対する変更により、属性ディメンション、および属性ディメンションによってディメンション化される分析ビューを使用するすべての階層の状態がINVALIDに変更されます。また、属性ディメンションおよび分析ビューによって使用される表またはビューに対するDDL変更によって、これらのオブジェクトの状態がINVALIDに変更されます。

ALL_OBJECTSデータ・ディクショナリ・ビューには、VALIDまたはINVALIDのSTATUS列があります。属性ディメンション、階層および分析ビューの場合、STATUS値はCOMPILE_STATEに関連付けられています。COMPILE_STATEがVALIDの場合、STATUS値はVALIDです。COMPILE_STATEがINVALIDの場合、STATUSはINVALIDです。

24.5 データの検証

問合せ結果の正確さを保証するには、階層およびアナリティックのデータを検証する必要があります。

階層または分析ビューのデータを検証するには、PL/SQLパッケージDBMS_HIERARCHYの関数を使用します。VALIDATE_HIERARCHY関数およびVALIDATE_ANALYTIC_VIEW関数は、データを検証して結果を表に格納します。これらの関数のオプションの引数は表の名前です。CREATE_VALIDATE_LOG_TABLEプロシージャは、この目的のために使用できる表を作成します。表を指定しない場合は、VALIDATE_HIERARCHY関数およびVALIDATE_ANALYTIC_VIEW関数によって表が作成されます。

関連付けられている属性ディメンションまたは分析ビューで使用される表に対するSQL DDLまたはDMLによる変更、または属性ディメンション、階層または分析ビュー自体へのDDLによる変更を行うと、階層の状態がINVALIDに変更されます。

データ・セキュリティ・ポリシーが階層、分析ビュー、または関連付けられている属性ディメンションによって使用される表やビューに適用されている場合は、検証の状態を判別できず、VALIDATE_STATEはVALIDに設定されません。VALIDATE_HIERARCHY関数またはVALIDATE_ANALYTIC_VIEW関数を実行すると、その時点のそのユーザーにおいて階層または分析ビューが有効であるかどうかが示されます。

属性ディメンションによって使用される表またはビューに対するSQL DMLによる変更が、データ・ディクショナリへの問合せまたはVALIDATE_HIERARCHY関数の実行と階層または分析ビューへの問合せの実行の間に行われた場合、階層は無効になることがあります。問合せで階層が有効であるようにするには、読取り専用トランザクションの作成(たとえば、SET TRANSACTION READ ONLY)、検証関数の実行、検証の成功の確認、問合せの実行、およびCOMMIT文やROLLBACK文によるトランザクションの終了を行います。

24.6 分析ビューの分類

分類により、属性ディメンション、階層および分析ビュー・オブジェクトに対して、さらにはこれらのコンポーネント(属性ディメンション・キー、属性、レベル、メジャーなど)に説明的なメタデータが提供されます。

アプリケーションは、分類を使用して、階層および分析ビューに関する情報を表示できます。分類は、表および列に関するコメントに似ていますが、コメントは単一の値です。同じオブジェクトに、任意の数の分類を指定できます。言語別に値を変更できます。分類値は常にテキスト・リテラルで、その最大長は4000バイトです。

分類は、SQL問合せには関係ありませんが、ツールまたはアプリケーションで使用するデータ・ディクショナリ・ビューで使用できます。

CAPTIONおよびDESCRIPTION分類には、分類をサポートするすべてのオブジェクトに対するDDLショートカットがあります。

分類値の言語を指定できます。言語を指定する場合は、有効なNLS_LANGUAGE値である必要があります。言語を指定しない場合、分類の言語値はNULLで、デフォルトのデータベース言語が使用されます。

CAPTIONおよびDESCRIPTIONのDDLショートカットは、NULL言語にのみ適用されます。特定の言語のCAPTIONおよびDESCRIPTION分類を指定するには、完全なCLASSIFICATION構文を使用する必要があります。

SQLツールでは、NULL言語の値はデフォルトとして解釈されます。たとえば、ツールで属性ディメンションのCAPTIONを検索しているとします。ツールは、最初に現在のNLS_LANGUAGEに一致する言語を持つCAPTIONを検索します。見つかった場合、そのCAPTION値を使用します。見つからない場合、NULL言語の値を持つCAPTIONを検索して、それを使用します。SQLロジックは、ユーザー、ツールまたはアプリケーションによって決まります。

言語によって異なる説明的なメタデータを階層のメンバーに提供するには、階層属性MEMBER_NAME、MEMBER_CAPTIONおよびMEMBER_DESCRIPTIONを使用します。

24.7 アプリケーション・コンテナとの分析ビューの共有

分析ビューをアプリケーション・コンテナと共有できます。

分析ビュー・オブジェクトの定義では、SHARING句を使用して、属性ディメンション、階層、分析ビュー・メタデータまたはオブジェクトを、アプリケーション・コンテナと共有できます。この句の値は、次のとおりです。

値	説明
NONE	共有しません。これがデフォルト値です。
METADATA	メタデータのみ共有します。
OBJECT	オブジェクト(データを含む)を共有します。

METADATAを指定すると、オブジェクトの定義のみがアプリケーション・コンテナと共有されます。

OBJECTを指定すると、属性ディメンション、階層または分析ビュー・オブジェクト(オブジェクトのデータ・ソースを含む)がアプリケーション・コンテナと共有されます。

24.8 分析ビュー・オブジェクトの変更または削除

SQL DDL文を使用すると、オブジェクトの名前変更またはオブジェクトの削除を行うことができます。

名前以外の分析ビュー・オブジェクトの特性を変更するには、CREATE OR REPLACE文を使用して、目的の変更が適用されたオブジェクトに置き換えます。

例24-3 属性ディメンションの名前変更

次の例は、属性ディメンションを名前変更します。

```
ALTER ATTRIBUTE DIMENSION product_attr_dim RENAME TO myproduct_attr_dim;
```

例24-4 属性ディメンションの削除

次の例は、属性ディメンションを削除します。

```
DROP ATTRIBUTE DIMENSION myproduct_attr_dim;
```

24.9 例のデータとスクリプト

この項では、分析ビューの例が基にしているデータについて説明します。分析ビュー・コンポーネント・オブジェクトを作成するためのSQL文も含まれています。

データおよび分析ビュー・コンポーネントについては、次のトピックで説明しています。

- 例のデータとスクリプトについて
- Create Attribute Dimension文
- Create Hierarchy文
- Create Analytic View文

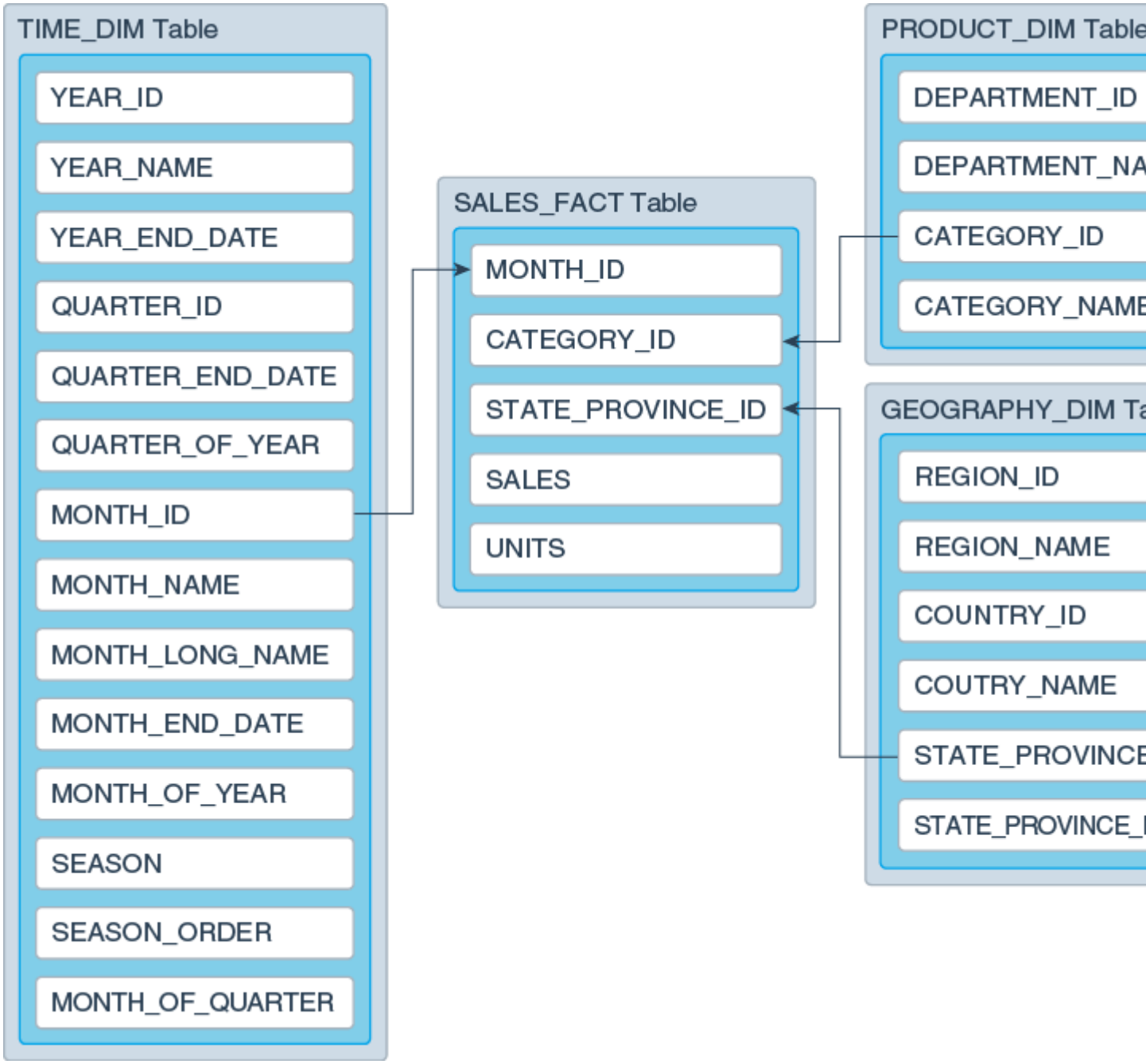
24.9.1 例のデータとスクリプトについて

例で使用されるデータは、単一のファクト表の売上データ、および3つのディメンション表(期間、製品および地理的データ)で構成されています。

例に使用されている表、分析ビュー・コンポーネント・オブジェクトおよび問合せを作成するSQLスクリプトは、Oracle Live SQL Webサイト(https://livesql.oracle.com/apex/livesql/file/index.html)で表示および実行できます。

データは、次の図に示すスター・スキーマ表にあります。

図24-1 分析ビューの例の表



SALES_FACT表のMONTH_ID、DEPARTMENT_IDおよびSTATE_PROVINCE_ID列は、それぞれTIME_DIM、PRODUCT_DIMおよびGEOGRAPHY_DIMディメンション表の外部キーです。

各ディメンション表で、_ID列はキーとして使用され、_NAME列は記述子として使用されます。他の列は、ソートまたはレポートのための属性として使用できます。

_ID列および_NAME列のデータには1対1の関係があります。期間でソートするには、TIME_DIM表の_END_DATE列を使用します。

24.9.2 Create Attribute Dimension文

このトピックには、例の属性ディメンションを作成するためのSQL文が含まれています。

time_attr_dim属性ディメンションの作成

time_attr_dim属性ディメンションは、TIME_DIMディメンション表に基づいています。次の文は属性ディメンションを作成します。

```
CREATE OR REPLACE ATTRIBUTE DIMENSION time_attr_dim
```

```
DIMENSION TYPE TIME
USING time_dim
ATTRIBUTES
 (year_id
   CLASSIFICATION caption VALUE 'YEAR_ID'
   CLASSIFICATION description VALUE 'YEAR ID',
  year_name
    CLASSIFICATION caption VALUE 'YEAR_NAME'
    CLASSIFICATION description VALUE 'Year',
  year_end_date
    CLASSIFICATION caption VALUE 'YEAR_END_DATE'
    CLASSIFICATION description VALUE 'Year End Date',
  quarter_id
    CLASSIFICATION caption VALUE 'QUARTER_ID'
    CLASSIFICATION description VALUE 'QUARTER ID',
  quarter_name
    CLASSIFICATION caption VALUE 'QUARTER_NAME'
    CLASSIFICATION description VALUE 'Quarter',
  quarter_end_date
    CLASSIFICATION caption VALUE 'QUARTER_END_DATE'
    CLASSIFICATION description VALUE 'Quarter End Date',
  quarter_of_year
    CLASSIFICATION caption VALUE 'QUARTER_OF_YEAR'
    CLASSIFICATION description VALUE 'Quarter of Year',
  month_id
    CLASSIFICATION caption VALUE 'MONTH_ID'
    CLASSIFICATION description VALUE 'MONTH ID',
  month_name
    CLASSIFICATION caption VALUE 'MONTH_NAME'
    CLASSIFICATION description VALUE 'Month',
  month_long_name
    CLASSIFICATION caption VALUE 'MONTH_LONG_NAME'
    CLASSIFICATION description VALUE 'Month Long Name',
  month_end_date
    CLASSIFICATION caption VALUE 'MONTH_END_DATE'
    CLASSIFICATION description VALUE 'Month End Date',
  month_of_quarter
    CLASSIFICATION caption VALUE 'MONTH_OF_QUARTER'
    CLASSIFICATION description VALUE 'Month of Quarter',
  month_of_year
    CLASSIFICATION caption VALUE 'MONTH_OF_YEAR'
    CLASSIFICATION description VALUE 'Month of Year',
  season
    CLASSIFICATION caption VALUE 'SEASON'
    CLASSIFICATION description VALUE 'Season',
  season_order
    CLASSIFICATION caption VALUE 'SEASON_ORDER'
    CLASSIFICATION description VALUE 'Season Order')
LEVEL month
  LEVEL TYPE MONTHS
  CLASSIFICATION caption VALUE 'MONTH'
  CLASSIFICATION description VALUE 'Month'
  KEY month_id
  MEMBER NAME month_name
  MEMBER CAPTION month_name
  MEMBER DESCRIPTION month_long_name
  ORDER BY month_end_date
  DETERMINES (month_end_date,
    quarter_id,
    season,
```

```
    season_order,
    month_of_year,
    month_of_quarter)
LEVEL quarter
  LEVEL TYPE QUARTERS
  CLASSIFICATION caption VALUE 'QUARTER'
  CLASSIFICATION description VALUE 'Quarter'
  KEY quarter_id
  MEMBER NAME quarter_name
  MEMBER CAPTION quarter_name
  MEMBER DESCRIPTION quarter_name
  ORDER BY quarter_end_date
  DETERMINES (quarter_end_date,
    quarter_of_year,
    year_id)
LEVEL year
  LEVEL TYPE YEARS
  CLASSIFICATION caption VALUE 'YEAR'
  CLASSIFICATION description VALUE 'Year'
  KEY year_id
  MEMBER NAME year_name
  MEMBER CAPTION year_name
  MEMBER DESCRIPTION year_name
  ORDER BY year_end_date
  DETERMINES (year_end_date)
LEVEL season
  LEVEL TYPE QUARTERS
  CLASSIFICATION caption VALUE 'SEASON'
  CLASSIFICATION description VALUE 'Season'
  KEY season
  MEMBER NAME season
  MEMBER CAPTION season
  MEMBER DESCRIPTION season
LEVEL month_of_quarter
  LEVEL TYPE MONTHS
  CLASSIFICATION caption VALUE 'MONTH_OF_QUARTER'
  CLASSIFICATION description VALUE 'Month of Quarter'
  KEY month_of_quarter;
```

product_attr_dim属性ディメンションの作成

product_attr_dim属性ディメンションは、PRODUCT_DIMディメンション表に基づいています。次の文は属性ディメンションを作成します。

```
CREATE OR REPLACE ATTRIBUTE DIMENSION product_attr_dim
USING product_dim
ATTRIBUTES
 (department_id
    CLASSIFICATION caption VALUE 'DEPARTMENT_ID'
    CLASSIFICATION description VALUE 'DEPARTMENT ID',
  department_name
    CLASSIFICATION caption VALUE 'DEPARTMENT_NAME'
    CLASSIFICATION description VALUE 'Department',
  category_id
    CLASSIFICATION caption VALUE 'CATEGORY_ID'
    CLASSIFICATION description VALUE 'CATEGORY ID',
  category_name
    CLASSIFICATION caption VALUE 'CATEGORY_NAME'
    CLASSIFICATION description VALUE 'Category')
LEVEL DEPARTMENT
```

```
  CLASSIFICATION caption VALUE 'DEPARTMENT'
  CLASSIFICATION description VALUE 'Department'
  KEY department_id
  MEMBER NAME department_name
  MEMBER CAPTION department_name
  ORDER BY department_name
LEVEL CATEGORY
  CLASSIFICATION caption VALUE 'CATEGORY'
  CLASSIFICATION description VALUE 'Category'
  KEY category_id
  MEMBER NAME category_name
  MEMBER CAPTION category_name
  ORDER BY category_name
  DETERMINES(department_id)
ALL MEMBER NAME 'ALL PRODUCTS';
```

geography_attr_dim属性ディメンションの作成

geography_attr_dim属性ディメンションは、GEOGRAPHY_DIMディメンション表に基づいています。次の文は属性ディメンションを作成します。

```
CREATE OR REPLACE ATTRIBUTE DIMENSION geography_attr_dim
USING geography_dim
ATTRIBUTES
 (region_id
    CLASSIFICATION caption VALUE 'REGION_ID'
    CLASSIFICATION description VALUE 'REGION ID',
  region_name
    CLASSIFICATION caption VALUE 'REGION_NAME'
    CLASSIFICATION description VALUE 'Region',
  country_id
    CLASSIFICATION caption VALUE 'COUNTRY_ID'
    CLASSIFICATION description VALUE 'COUNTRY ID',
  country_name
    CLASSIFICATION caption VALUE 'COUNTRY_NAME'
    CLASSIFICATION description VALUE 'Country',
  state_province_id
    CLASSIFICATION caption VALUE 'STATE_PROVINCE_ID'
    CLASSIFICATION description VALUE 'STATE-PROVINCE ID',
  state_province_name
    CLASSIFICATION caption VALUE 'STATE_PROVINCE_NAME'
    CLASSIFICATION description VALUE 'State-Province')
LEVEL REGION
  CLASSIFICATION caption VALUE 'REGION'
  CLASSIFICATION description VALUE 'Region'
  KEY region_id
  MEMBER NAME region_name
  MEMBER CAPTION region_name
  ORDER BY region_name
LEVEL COUNTRY
  CLASSIFICATION caption VALUE 'COUNTRY'
  CLASSIFICATION description VALUE 'Country'
  KEY country_id
  MEMBER NAME country_name
  MEMBER CAPTION country_name
  ORDER BY country_name
  DETERMINES(region_id)
LEVEL STATE_PROVINCE
  CLASSIFICATION caption VALUE 'STATE_PROVINCE'
  CLASSIFICATION description VALUE 'State-Province'
```

```
  KEY state_province_id
  MEMBER NAME state_province_name
  MEMBER CAPTION state_province_name
  ORDER BY state_province_name
  DETERMINES(country_id)
ALL MEMBER NAME 'ALL CUSTOMERS';
```

24.9.3 Create Hierarchy文

このトピックには、例の階層を作成するためのSQL文が含まれています。

time_attr_dimを使用した階層の作成

次の文は、time_attr_dim属性ディメンションを使用する階層を作成します。

```
CREATE OR REPLACE HIERARCHY time_hier
  CLASSIFICATION caption VALUE 'CALENDAR'
  CLASSIFICATION description VALUE 'CALENDAR'
USING time_attr_dim
  (month CHILD OF
   quarter CHILD OF
   year);
--
CREATE OR REPLACE HIERARCHY time_season_hier
  CLASSIFICATION caption VALUE 'SEASONS'
  CLASSIFICATION description VALUE 'Seasons'
USING time_attr_dim
  (month CHILD OF
   season);
--
CREATE OR REPLACE HIERARCHY time_year_season_hier
USING time_attr_dim
  (month CHILD OF
   season CHILD OF
   year);
--
CREATE OR REPLACE HIERARCHY time_month_of_qtr_hier
  CLASSIFICATION caption VALUE 'MONTH_OF_QUARTER'
  CLASSIFICATION description VALUE 'Month of Quarter'
USING time_attr_dim
  (month CHILD OF
   month_of_quarter);
```

product_attr_dimを使用した階層の作成

次の文は、product_attr_dim属性ディメンションを使用する階層を作成します。

```
CREATE OR REPLACE HIERARCHY product_hier
  CLASSIFICATION caption VALUE 'PRODUCT'
  CLASSIFICATION description VALUE 'Product'
USING product_attr_dim
  (CATEGORY
   CHILD OF department);
```

geography_attr_dimを使用した階層の作成

次の文は、geography_attr_dim属性ディメンションを使用する階層を作成します。

```
CREATE OR REPLACE HIERARCHY geography_hier
  CLASSIFICATION caption VALUE 'GEOGRAPHY'
```

```
  CLASSIFICATION description VALUE 'Geography'
USING geography_attr_dim
  (state_province
   CHILD OF country
   CHILD OF region);
```

24.9.4 Create Analytic View文

このトピックには、例の分析ビューを作成するためのSQL文が含まれています。

sales_av分析ビューの作成

次の文は、SALES_FACTファクト表を使用する分析ビューを作成します。

```
CREATE OR REPLACE ANALYTIC VIEW sales_av
  CLASSIFICATION caption VALUE 'Sales AV'
  CLASSIFICATION description VALUE 'Sales Analytic View'
  CLASSIFICATION created_by VALUE 'Harold C. Ehrlicher'
USING sales_fact
DIMENSION BY
  (time_attr_dim
    KEY month_id REFERENCES month_id
    HIERARCHIES (
      time_hier DEFAULT,
      time_season_hier,
      time_year_season_hier,
      time_month_of_qtr_hier),
   product_attr_dim
    KEY category_id REFERENCES category_id
    HIERARCHIES (
      product_hier DEFAULT),
   geography_attr_dim
    KEY state_province_id
    REFERENCES state_province_id
    HIERARCHIES (
      geography_hier DEFAULT)
   )
MEASURES
 (sales FACT sales
    CLASSIFICATION caption VALUE 'Sales'
    CLASSIFICATION description VALUE 'Sales'
    CLASSIFICATION format_string VALUE '$9,999.99',
  units FACT units
    CLASSIFICATION caption VALUE 'Units'
    CLASSIFICATION description VALUE 'Units Sold'
    CLASSIFICATION format_string VALUE '9,999',
  sales_prior_period AS
    (LAG(SALES) OVER (HIERARCHY time_hier OFFSET 1))
      CLASSIFICATION caption VALUE 'Sales Prior Period'
      CLASSIFICATION description VALUE 'Sales Prior_Period'
      CLASSIFICATION format_string VALUE '$9,999.99',
  sales_chg_prior_period AS
    (LAG_DIFF(SALES) OVER (HIERARCHY time_hier OFFSET 1))
      CLASSIFICATION caption VALUE 'Sales Change Prior Period'
      CLASSIFICATION description VALUE 'Sales Change Prior Period'
      CLASSIFICATION format_string VALUE '$9,999.99',
  sales_qtr_ago AS
    (LAG(SALES) OVER (HIERARCHY time_hier OFFSET 1
      ACROSS ANCESTOR AT LEVEL quarter))
        CLASSIFICATION caption VALUE 'Sales Qtr Ago'
```

```
      CLASSIFICATION description VALUE 'Sales Qtr Ago'
      CLASSIFICATION format_string VALUE '$9,999.99',
sales_chg_qtr_ago AS
  (LAG_DIFF(SALES) OVER (HIERARCHY time_hier OFFSET 1
    ACROSS ANCESTOR AT LEVEL quarter))
      CLASSIFICATION caption VALUE 'Sales Change Qtr Ago'
      CLASSIFICATION description VALUE 'Sales Change Qtr Ago'
      CLASSIFICATION format_string VALUE '$9,999.99',
sales_pct_chg_qtr_ago AS
  (LAG_DIFF_PERCENT(SALES) OVER (HIERARCHY time_hier OFFSET 1
    ACROSS ANCESTOR AT LEVEL quarter))
      CLASSIFICATION caption VALUE 'Sales Percent Change Qtr Ago'
      CLASSIFICATION description VALUE 'Sales Percent Change Qtr Ago'
      CLASSIFICATION format_string VALUE '999.99',
sales_yr_ago AS
  (LAG(SALES) OVER (HIERARCHY time_hier OFFSET 1
    ACROSS ANCESTOR AT LEVEL year))
      CLASSIFICATION caption VALUE 'Sales Year Ago'
      CLASSIFICATION description VALUE 'Sales Year Ago'
      CLASSIFICATION format_string VALUE '$9,999.99',
sales_chg_yr_ago AS
  (LAG_DIFF(SALES) OVER (HIERARCHY time_hier OFFSET 1
    ACROSS ANCESTOR AT LEVEL year))
      CLASSIFICATION caption VALUE 'Sales Change Year Ago'
      CLASSIFICATION description VALUE 'Sales Change Year Ago'
      CLASSIFICATION format_string VALUE '$9,999.99',
sales_pct_chg_yr_ago AS
  (LAG_DIFF_PERCENT(SALES) OVER (HIERARCHY time_hier OFFSET 1
    ACROSS ANCESTOR AT LEVEL year))
      CLASSIFICATION caption VALUE 'Sales Percent Change Year Ago'
      CLASSIFICATION description VALUE 'Sales Percent Change Year Ago'
      CLASSIFICATION format_string VALUE '999.99',
sales_qtd AS
  (SUM(sales) OVER (HIERARCHY time_hier
    BETWEEN UNBOUNDED PRECEDING AND CURRENT MEMBER
    WITHIN ANCESTOR AT LEVEL quarter))
      CLASSIFICATION caption VALUE 'Sales Quarter to Date'
      CLASSIFICATION description VALUE 'Sales Quarter to Date'
      CLASSIFICATION format_string VALUE '$9,999.99',
sales_ytd AS
  (SUM(sales) OVER (HIERARCHY time_hier
    BETWEEN UNBOUNDED PRECEDING AND CURRENT MEMBER
    WITHIN ANCESTOR AT LEVEL year))
      CLASSIFICATION caption VALUE 'Sales Year to Date'
      CLASSIFICATION description VALUE 'Sales Year to Date'
      CLASSIFICATION format_string VALUE '$9,999.99',
sales_2011 AS
  (QUALIFY (sales, time_hier = year['11']))
    CLASSIFICATION caption VALUE 'Sales CY2011'
    CLASSIFICATION description VALUE 'Sales CY2011'
    CLASSIFICATION format_string VALUE '$9,999.99',
sales_pct_chg_2011 AS
  ((sales - (QUALIFY (sales, time_hier = year['11']))) /
  (QUALIFY (sales, time_hier = year['11'])))
    CLASSIFICATION caption VALUE 'Sales Pct Change CY2011'
    CLASSIFICATION description VALUE 'Sales Pct Change CY2011'
    CLASSIFICATION format_string VALUE '999.99',
sales_share_time_parent AS
  (SHARE_OF(sales HIERARCHY time_hier PARENT))
    CLASSIFICATION caption VALUE 'Sales Share of Time Parent'
```

```
      CLASSIFICATION description VALUE 'Sales Share of Time Parent'
      CLASSIFICATION format_string VALUE '999.99',
  sales_share_season_parent AS
    (SHARE_OF(sales HIERARCHY time_season_hier PARENT))
      CLASSIFICATION caption VALUE 'Sales Share of Season Parent'
      CLASSIFICATION description VALUE 'Sales Share of Season Parent'
     CLASSIFICATION format_string VALUE '999.99',
  sales_share_prod_parent AS
    (SHARE_OF(sales HIERARCHY product_hier PARENT))
      CLASSIFICATION caption VALUE 'Sales Share of Product Parent'
      CLASSIFICATION description VALUE 'Sales Share of Product Parent'
      CLASSIFICATION format_string VALUE '999.99',
  sales_share_dept AS
    (SHARE_OF(sales HIERARCHY product_hier LEVEL department))
      CLASSIFICATION caption VALUE 'Sales Share of Product Parent'
      CLASSIFICATION description VALUE 'Sales Share of Product Parent'
      CLASSIFICATION format_string VALUE '999.99',
  sales_share_geog_parent AS
   (SHARE_OF(sales HIERARCHY geography_hier PARENT))
      CLASSIFICATION caption VALUE 'Sales Share of Geography Parent'
      CLASSIFICATION description VALUE 'Sales Share of Geography Parent'
      CLASSIFICATION format_string VALUE '999.99',
  sales_share_region AS
    (SHARE_OF(sales HIERARCHY geography_hier LEVEL region))
      CLASSIFICATION caption VALUE 'Sales Share of Geography Parent'
      CLASSIFICATION description VALUE 'Sales Share of Geography Parent'
      CLASSIFICATION format_string VALUE '999.99'
  )
DEFAULT MEASURE SALES;
```

25 属性ディメンションおよび階層オブジェクト

属性ディメンションはデータ・ソースを参照し、属性およびレベルを指定します。階層はレベルを階層的に編成します。

属性ディメンションおよび階層については、次のトピックで説明しています。

- 属性ディメンションおよび階層について
- 属性および階層属性
- レベルの順序
- レベル・キー
- 属性の関係の決定

25.1 属性ディメンションおよび階層について

属性ディメンションは、データ・ソース、属性およびレベルを指定します。階層はレベルを階層的に編成します。

属性ディメンションは、使用するデータ・ソースを指定し、そのソースの列を属性として指定します。一部またはすべての属性のレベルを指定し、レベル間の属性の関係を決定します。

階層では、属性ディメンションのレベル間の階層リレーションシップを定義します。属性ディメンションおよび階層は、分析ビュー・オブジェクトのディメンション・メンバーを提供します。

ディメンションおよび階層に関連するメタデータの多くは、属性ディメンションで定義されます。階層は、使用する属性ディメンションのすべてのメタデータを継承します。これにより、属性およびレベルのメタデータが多くの階層で再使用され、一貫性が促進され、階層の定義を単純化できます。

属性ディメンションについて

属性ディメンションには次の特性があります。

- データ・ソース(通常はスター・スキーマまたはスノーフレーク・スキーマのディメンション表ですが、非正規化表、ビュー、外部表またはリモート表である場合があります。ディメンション表の各列は階層に設定できます)
- ディメンション・タイプ(STANDARDまたはTIME)
- 属性(データ・ソースの列)
- レベル(同じレベルの集計にあるすべての値のグループを表します)
- 階層属性(階層によって使用され、レベル間の階層関係を示す)
- 1つのメンバーのみを持つ暗黙的なALLレベル(属性ディメンションを使用する階層の最も高いレベルです)
- 任意の数の階層が使用できます

属性ディメンションには次のオプションの特性もあります。

- メタデータまたはメタデータとデータをアプリケーション・コンテナと共有するように指定できます
- レベル・メンバーの順序を指定できます
- 属性ディメンション自体、属性、一部の階層属性、レベルおよびALLメンバーの分類を指定できます。アプリケーションが問合せに使用したり、問合せ結果を表示するために使用したりできるメタデータが分類によって提供されます

含まれているレベルによって判別される属性は、階層の列になる属性を指定するため、その階層を参照する分析ビューの属性

となります。

属性ディメンションおよびレベル・タイプについて

属性ディメンションは、STANDARDタイプまたはTIMEタイプにすることができます。機能的には、STANDARDタイプおよびTIMEタイプの属性ディメンションは同じです。ただし、TIMEタイプの属性ディメンションの各レベルでは、レベル・タイプを指定する必要があります(レベル・メンバーの値はそのタイプである必要はありません)。たとえば、TIMEタイプの属性ディメンションでは、レベル・タイプQUARTERSのSEASONという名前のレベルを作成できますが、値は季節の名前になります。レベル・タイプは任意に選択した用途に使用できます。

STANDARDタイプの属性ディメンションのレベルは、STANDARDタイプです。STANDARDタイプの属性ディメンションのレベルには、レベル・タイプを指定する必要はありません。

TIMEタイプの属性ディメンションのレベルは、次のレベル・タイプのいずれかである必要があります。

- YEARS
- HALF_YEARS
- QUARTERS
- MONTHS
- WEEKS
- DAYS
- HOURS
- MINUTES
- SECONDS

階層について

階層の特性は次のとおりです。

- 属性ディメンション
- 属性ディメンションのレベルの階層順序
- レベルの各属性(判別された属性を含む)の列
- 階層属性の列
- 階層の各レベルの各メンバーの行、および単一の最上位の集計値を表す暗黙的なALLレベルの行
- 属性ディメンションから継承したメタデータ
- SQLのSELECT文のFROM句に使用できます

階層には次のオプションの特性もあります。

- メタデータまたはメタデータとデータをアプリケーション・コンテナと共有するように指定できます
- 階層自体および階層属性の分類を指定できます

例25-1 単純な属性ディメンション

属性ディメンションは、キー属性のみを定義したレベルと属性のリストのような単純なものにすることができます。この例は、TIME_DIM表のYEAR_ID、QUARTER_IDおよびMONTH_ID列のみを属性として指定した属性ディメンションを作成しています。

```
CREATE OR REPLACE ATTRIBUTE DIMENSION time_attr_dim
DIMENSION TYPE TIME
USING time_dim       -- References the TIME_DIM table
```

```
ATTRIBUTES           -- A list of table columns to be used as attributes
 (year_id,
  quarter_id,
  month_id)
LEVEL MONTH          -- A level
  LEVEL TYPE MONTHS  -- The level type
  KEY month_id       -- Attribute with unique values
LEVEL QUARTER
  LEVEL TYPE QUARTERS
  KEY quarter_id
LEVEL YEAR
  LEVEL TYPE YEARS
  KEY year_id;
```

TIME_DIM表の詳細は、例のデータとスクリプトについてを参照してください。

TIME_DIM表の各_ID列は、属性リストに含まれています。デフォルトでは、属性の名前はディメンション表の列の名前です。属性に異なる名前を指定するには、定義でAS alias句を使用します。

レベルはKEYプロパティ(レベルで唯一の必須のプロパティです)を使用して各属性に作成されます。

例25-2 単純な階層

```
CREATE OR REPLACE HIERARCHY time_hier  -- Hierarchy name
USING time_attr_dim      -- Refers to the TIME_ATTR_DIM attribute dimension
 (month CHILD OF         -- Levels in the attribute dimension
  quarter CHILD OF
  year);
```

階層には、属性ディメンションの各属性および階層属性の列があります。

```
SELECT column_name from ALL_HIER_COLUMNS WHERE HIER_NAME = 'TIME_HIER';
```

	COLUMN_NAME
1	DEPTH
2	HIER_ORDER
3	IS_LEAF
4	LEVEL_NAME
5	MEMBER_CAPTION
6	MEMBER_DESCRIPTION
7	MEMBER_NAME
8	MEMBER_UNIQUE_NAME
9	MONTH_ID
10	PARENT_LEVEL_NAME
11	PARENT_UNIQUE_NAME
12	QUARTER_ID
13	YEAR ID

次の文は、TIME_ATTR_DIMが例25-1で定義された属性ディメンションである場合に、TIME_HIERから属性列および一部の階層列を選択しています。

```
SELECT year_id, quarter_id, month_id,
       member_name, member_unique_name
       member_caption, member_description
  FROM time_hier
  ORDER BY hier_order;
```

問合せ結果の一部を示します。

	YEAR_ID	QUARTER_ID	MONTH_ID	MEMBER_NAME	MEMBER_UNIQUE_NAME	MEMBER_CAPTION	MEMBER_DE
1	(null)	(null)	(null)	ALL	[ALL].[ALL]	(null)	(null)
2	11	(null)	(null)	11	[YEAR].&[11]	(null)	(null)
3	11	111	(null)	111	[QUARTER].&[11]&[111]	(null)	(null)
4	11	111	Feb-11	Feb-11	[MONTH].&[11]&[111]&[Feb-11]	(null)	(null)
5	11	111	Jan-11	Jan-11	[MONTH].&[11]&[111]&[Jan-11]	(null)	(null)
6	11	111	Mar-11	Mar-11	[MONTH].&[11]&[111]&[Mar-11]	(null)	(null)
7	11	211	(null)	211	[QUARTER].&[11]&[211]	(null)	(null)
8	11	211	Apr-11	Apr-11	[MONTH].&[11]&[211]&[Apr-11]	(null)	(null)
9	11	211	Jun-11	Jun-11	[MONTH].&[11]&[211]&[Jun-11]	(null)	(null)
10	11	211	May-11	May-11	[MONTH].&[11]&[211]&[May-11]	(null)	(null)
11	11	311	(null)	311	[QUARTER].&[11]&[311]	(null)	(null)
12	11	311	Aug-11	Aug-11	[MONTH].&[11]&[311]&[Aug-11]	(null)	(null)
13	11	311	Jul-11	Jul-11	[MONTH].&[11]&[311]&[Jul-11]	(null)	(null)
14	11	311	Sep-11	Sep-11	[MONTH].&[11]&[311]&[Sep-11]	(null)	(null)

25.2 属性および階層属性

通常、属性ディメンションの属性はソース表またはビューの列を参照しています。階層属性は、階層のメンバーに関する情報を提供します。

属性ディメンションで、属性は参照するソース表またはビューの列を指定します。属性のデフォルト名は表の列名です。属性に異なる名前を指定するには、SQLのSELECT句の別名に似た構文を使用します。レベルは属性を使用して定義し、属性間の関係はレベルを使用して定義します。属性は、階層に含まれるレベル、およびレベルに定義されている属性の関係に応じて、階層の列として表示されます。

階層属性は次のとおりです。

- DEPTHは階層メンバーのレベルの深さです。ALLレベルは深さ0 (ゼロ)です
- HIER_ORDERは階層内のメンバーの順序です
- IS_LEAFは、メンバーが階層の最も低い(リーフ)レベルにあるかどうかを示すブール値です
- LEVEL_NAMEは、属性ディメンションの定義内のレベルの名前です
- MEMBER_NAMEは、属性ディメンションの定義内のメンバーの名前です
- 属性ディメンションまたは階層の定義に値を指定していない場合、MEMBER_CAPTIONはNULLです
- 属性ディメンションまたは階層の定義に値を指定していない場合、MEMBER_DESCRIPTIONはNULLです
- MEMBER_UNIQUE_NAMEは、階層内で一意であることが保証されている名前であり、レベル名、祖先およびキー属性値を連結したものです
- PARENT_LEVEL_NAMEは、現在のメンバーの親のレベルの名前です
- PARENT_UNIQUE_NAMEは、現在のメンバーの親のMEMBER_UNIQUE_NAMEです

階層属性値は、レベルおよび系統で構成されます。系統にはメンバーのキー値が含まれています。系統の各コンポーネントは角括弧で囲み、コンポーネントはピリオドで区切ります。コンポーネント値に右角括弧が含まれている場合は、2つの右角括弧を使用して表されます。

例25-3 階層属性への値の設定

これは、属性ディメンションおよび階層についての単純な属性ディメンションに基づいた階層に対する問合せの結果の一部です。

	YEAR_ID	QUARTER_ID	MONTH_ID	MEMBER_NAME	MEMBER_UNIQUE_NAME	MEMBER_CAPTION	MEMBER_DES
1	(null)	(null)	(null)	ALL	[ALL].[ALL]	(null)	(null)
2	11	(null)	(null)	11	[YEAR].&[11]	(null)	(null)
3	11	111	(null)	111	[QUARTER].&[11]&[111]	(null)	(null)
4	11	111	Feb-11	Feb-11	[MONTH].&[11]&[111]&[Feb-11]	(null)	(null)
5	11	111	Jan-11	Jan-11	[MONTH].&[11]&[111]&[Jan-11]	(null)	(null)
6	11	111	Mar-11	Mar-11	[MONTH].&[11]&[111]&[Mar-11]	(null)	(null)
7	11	211	(null)	211	[QUARTER].&[11]&[211]	(null)	(null)
8	11	211	Apr-11	Apr-11	[MONTH].&[11]&[211]&[Apr-11]	(null)	(null)
9	11	211	Jun-11	Jun-11	[MONTH].&[11]&[211]&[Jun-11]	(null)	(null)
10	11	211	May-11	May-11	[MONTH].&[11]&[211]&[May-11]	(null)	(null)
11	11	311	(null)	311	[QUARTER].&[11]&[311]	(null)	(null)
12	11	311	Aug-11	Aug-11	[MONTH].&[11]&[311]&[Aug-11]	(null)	(null)
13	11	311	Jul-11	Jul-11	[MONTH].&[11]&[311]&[Jul-11]	(null)	(null)
14	11	311	Sep-11	Sep-11	[MONTH].&[11]&[311]&[Sep-11]	(null)	(null)

この階層は機能していますが、いくつかの重要な機能が欠けています。MEMBER_NAME列は読みやすいとは言えず、MEMBER_CAPTION列およびMEMBER_DESCRIPTION列にはデータが返されていません。

time_attr_dim属性ディメンションのこの新しい定義には、TIME_DIM表の_NAME列が含まれています。レベルの定義では、階層属性MEMBER_NAME、MEMBER_CAPTIONおよびMEMBER_DESCRIPTIONの値が含まれている属性を指定しています。この定義は、レベル・メンバーの指定値を持つ属性ディメンションを使用する階層を提供します。

```
CREATE OR REPLACE ATTRIBUTE DIMENSION time_attr_dim
DIMENSION TYPE TIME
USING time_dim
ATTRIBUTES
 (year_id,
  year_name,
  quarter_id,
  quarter_name,
  month_id,
  month_name,
  month_long_name)
LEVEL MONTH
  LEVEL TYPE MONTHS
  KEY month_id
  MEMBER NAME month_name
  MEMBER CAPTION month_name
  MEMBER DESCRIPTION month_long_name
LEVEL QUARTER
  LEVEL TYPE QUARTERS
  KEY quarter_id
  MEMBER NAME quarter_name
  MEMBER CAPTION quarter_name
  MEMBER DESCRIPTION quarter_name
LEVEL YEAR
  LEVEL TYPE YEARS
  KEY year_id
  MEMBER NAME year_name
  MEMBER CAPTION year_name
  MEMBER DESCRIPTION year_name;
```

この文は、TIME_HIER階層から属性列およびいくつかの階層列を選択します。

```
SELECT year_id, quarter_id, month_id,
       member_name, member_unique_name,
       member_caption, member_description
```

```
  FROM time_hier
  ORDER BY hier_order;
```

問合せ結果の一部を示します。

	YE...	QUARTER_ID	MONTH_ID	MEMBER_NAME	MEMBER_UNIQUE_NAME	MEMBER_CAPTION	MEMBER_DE
1	(null)	(null)	(null)	ALL	[ALL].[ALL]	(null)	(null)
2	11	(null)	(null)	CY2011	[YEAR].&[11]	CY2011	CY2011
3	11	111	(null)	Q1CY2011	[QUARTER].&[11]&[111]	Q1CY2011	Q1CY2011
4	11	111	Feb-11	Feb-11	[MONTH].&[11]&[111]&[Feb-11]	Feb-11	February 20
5	11	111	Jan-11	Jan-11	[MONTH].&[11]&[111]&[Jan-11]	Jan-11	January 201
6	11	111	Mar-11	Mar-11	[MONTH].&[11]&[111]&[Mar-11]	Mar-11	March 2011
7	11	211	(null)	Q2CY2011	[QUARTER].&[11]&[211]	Q2CY2011	Q2CY2011
8	11	211	Apr-11	Apr-11	[MONTH].&[11]&[211]&[Apr-11]	Apr-11	April 2011
9	11	211	Jun-11	Jun-11	[MONTH].&[11]&[211]&[Jun-11]	Jun-11	June 2011
10	11	211	May-11	May-11	[MONTH].&[11]&[211]&[May-11]	May-11	May 2011
11	11	311	(null)	Q3CY2011	[QUARTER].&[11]&[311]	Q3CY2011	Q3CY2011
12	11	311	Aug-11	Aug-11	[MONTH].&[11]&[311]&[Aug-11]	Aug-11	August 201
13	11	311	Jul-11	Jul-11	[MONTH].&[11]&[311]&[Jul-11]	Jul-11	July 2011
14	11	311	Sep-11	Sep-11	[MONTH].&[11]&[311]&[Sep-11]	Sep-11	September 2

期間の順序は、時系列の計算のレポートのためにはまだ正しくありません。たとえば、1月の前に2月があります。レベルのソート順を指定する例については、レベルの順序を参照してください。

25.3 レベルの順序

属性ディメンション・レベル・メンバーの順序を指定できます。

属性ディメンションのレベル定義のORDER BY句を使用すると、レベルのメンバーの順序を指定できます。デフォルトでは、属性ディメンション・レベルの値は、MEMBER_NAME値によってアルファベット順にソートされます。メンバー名を指定しない場合、レベルはKEY属性値で順序付けされます。

ORDER BY句では、NULL値を順序の最初または最後のどちらにするかも指定します。属性がレベルによって判別されない場合は、MINまたはMAX式を指定できます(デフォルトはMINです)。

例25-4 最終日の追加

この例は、最終日属性をtime_attr_dim属性ディメンションの定義に追加しています。

```
CREATE OR REPLACE ATTRIBUTE DIMENSION time_attr_dim
DIMENSION TYPE TIME
USING time_dim
ATTRIBUTES
 (year_id,
  year_name,
  year_end_date,
  quarter_id,
  quarter_name,
  quarter_end_date,
  month_id,
  month_name,
  month_long_name,
  month_end_date)
LEVEL MONTH
  KEY month_id
  MEMBER NAME month_name
  MEMBER CAPTION month_name
```

```
  MEMBER DESCRIPTION month_long_name
  ORDER BY month_end_date
LEVEL QUARTER
  KEY quarter_id
  MEMBER NAME quarter_name
  MEMBER CAPTION quarter_name
  MEMBER DESCRIPTION quarter_name
  ORDER BY quarter_end_date
LEVEL YEAR
  KEY year_id
  MEMBER NAME year_name
  MEMBER CAPTION year_name
  MEMBER DESCRIPTION year_name
  ORDER BY year_end_date;
```

これはtime_hier階層の定義です。

```
CREATE OR REPLACE HIERARCHY time_hier
USING time_attr_dim
 (month CHILD OF
  quarter CHILD OF
  year);
```

この問合せには階層の順序属性が含まれています。

```
SELECT year_id,
  quarter_id,
  month_id,
  member_name,
  hier_order
FROM time_hier
ORDER BY hier_order;
```

問合せ結果の一部を示します。

	YEAR_ID	QUARTER_ID	MONTH_ID	MEMBER_NAME	HIER_ORDER
1	(null)	(null)	(null)	ALL	0
2	11	(null)	(null)	CY2011	1
3	11	111	(null)	Q1CY2011	2
4	11	111	Jan-11	Jan-11	3
5	11	111	Feb-11	Feb-11	4
6	11	111	Mar-11	Mar-11	5
7	11	211	(null)	Q2CY2011	6
8	11	211	Apr-11	Apr-11	7
9	11	211	May-11	May-11	8
10	11	211	Jun-11	Jun-11	9
11	11	311	(null)	Q3CY2011	10
12	11	311	Jul-11	Jul-11	11
13	11	311	Aug-11	Aug-11	12
14	11	311	Sep-11	Sep-11	13

レベルのメンバーが最終日でソートされるようになりました。

25.4 レベル・キー

レベルのキー属性は、レベル・メンバーのデータ・ソースを指定します。

属性ディメンション・レベルは、レベルのメンバーを指定するキー属性およびオプションの代替キー属性を指定します。

レベルには、単一の属性、または複合キーの場合は複数の属性によって定義されるキーがある必要があります。キーの各個別値によって、そのレベルの属性ディメンション・メンバーが定義されます。

レベルは、1つ以上の代替キーを持つこともできます。代替キーはレベル・キーと1対1の関係である必要があります。代替キーとして指定する属性は、レベルの各メンバーのキー属性に対して一意の値である必要があります。

例25-5 PRODUCT_ATTR_DIM属性ディメンションの作成

この例は、product_attr_dim属性ディメンションを作成しています。LEVEL句には、キーと代替キーが指定されています。

```
CREATE OR REPLACE ATTRIBUTE DIMENSION product_attr_dim
USING product_dim
ATTRIBUTES
 (department_id,
  department_name,
  category_id,
  category_name)
LEVEL DEPARTMENT
  KEY department_id
  ALTERNATE KEY department_name
  MEMBER NAME department_name
  MEMBER CAPTION department_name
  ORDER BY department_name
LEVEL CATEGORY
  KEY category_id
  ALTERNATE KEY category_name
  MEMBER NAME category_name
  MEMBER CAPTION category_name
  ORDER BY category_name
  DETERMINES(department_id)
ALL MEMBER NAME 'ALL PRODUCTS';
```

25.5 属性の関係の決定

レベルの属性によって他の属性の値が判別されるように指定できます。

属性ディメンション定義のDETERMINES句を使用すると、レベルのキー属性と他の属性の関係を指定できます。ある属性の各値に対する別の属性の値が1つのみである場合は、ある属性の値によって別の属性の値が決まります。たとえば、MONTH_IDの各値に対しては1つのQUARTER_ID値のみがあり、MONTH_IDによってQUARTER_IDが決まります。

レベルによって判別される属性は、その属性ディメンションを使用する階層に含められます。DETERMINES句に指定する属性は、別々のレベル・メンバーで値が同じであってもかまいません。レベルによってキーおよび代替キーの属性が暗黙的に決定されますが、DETERMINES句の属性と異なり、それらの属性は一意の値である必要があります。

DETERMINES句によって指定される関係では、次のことを行うことができます。

- 階層によって返される行数の変更
- 特定の属性によって特定の行のデータが返されるかどうかの制御
- 分析ビューの問合せ時に生成されるSQLの単純化

判別される属性を指定すると、階層または分析ビューがメンバーの一意の値を判別するために役立ちます。属性がレベルによって判別される場合は、判別される属性と階層メンバーの関係を識別する属性値を問合せに明示的に指定する必要がありません。たとえば、QUALIFYの計算では一意に識別された階層メンバーが必要となります。DETERMINES句で属性を省略した場合は、QUALIFYの計算を使用する分析ビューのメジャーで、一意のメンバーを識別するためにそれらの属性を明示的に指定する必要があります。

判別される属性とキーおよび代替キーの属性の関係は、属性ディメンションまたは属性ディメンションを使用する階層で検証または施行されません。関係を検証するには、ソース表またはビューのデータを検査するPL/SQLプロシージャDBMS_HIERARCHY.VALIDATE_HIERARCHYを使用します。

使用上のノート

DETERMINES句を使用する場合は、次のことを考慮してください。

- 低いレベルのキーによって親レベルの値が決まる場合は、階層内の親レベルのKEY属性をDETERMINES句に含めます。低いレベルは、判別された祖先のレベルの属性を継承します。このため、親レベルのキー属性値を低いレベルのDETERMINES句に含めることをお薦めします。
- MEMBER NAME、MEMBER CAPTION、MEMBER DESCRIPTIONおよびORDER BYプロパティの値は、KEY属性値によって決まることが想定されます。それらのプロパティの属性をDETERMINES句に含める必要はありません。ただし、それらの属性のデータがKEY属性の各値に対して1つの値のみを持つことを確認する必要があります。

例25-6 DETERMINES句の追加

この例は、time_attr_dimのレベルにDETERMINES句を追加しています。

```
CREATE OR REPLACE ATTRIBUTE DIMENSION time_attr_dim
DIMENSION TYPE TIME
USING time_dim
ATTRIBUTES
 (year_id,
  year_name,
  year_end_date,
  quarter_id,
  quarter_name,
  quarter_end_date,
  month_id,
  month_name,
  month_long_name,
  month_end_date)
LEVEL MONTH
  LEVEL TYPE MONTHS
  KEY month_id
  MEMBER NAME month_name
  MEMBER CAPTION month_name
  MEMBER DESCRIPTION month_long_name
  ORDER BY month_end_date
  DETERMINES (quarter_id)
LEVEL QUARTER
  LEVEL TYPE QUARTERS
  KEY quarter_id
  MEMBER NAME quarter_name
  MEMBER CAPTION quarter_name
  MEMBER DESCRIPTION quarter_name
  ORDER BY quarter_end_date
  DETERMINES (year_id)
LEVEL YEAR
  LEVEL TYPE YEARS
  KEY year_id
  MEMBER NAME year_name
  MEMBER CAPTION year_name
  MEMBER DESCRIPTION year_name
  ORDER BY year_end_date;
```

TIME_HIER階層からLEVEL_NAME、_IDおよびMEMBER_UNIQUE_NAME列を選択します。

```
SELECT level_name,
  year_id,
  quarter_id,
  month_id,
  member_unique_name
FROM time_hier
ORDER BY hier_order;
```

前の問合せの次の結果に示されているように、月、四半期および年の属性の間の関係が階層で認識されるようになりました。MEMBER_UNIQUE_NAME値は、レベル名およびKEY属性値のみから作成されるようになりました。例25-3に示されているような完全な系統を含める必要はなくなりました。

	LEVEL_NAME	YEAR_ID	QUARTER_ID	MONTH_ID	MEMBER_UNIQUE_NAME
1	ALL	(null)	(null)	(null)	[ALL].[ALL]
2	YEAR	11	(null)	(null)	[YEAR].&[11]
3	QUARTER	11	111	(null)	[QUARTER].&[111]
4	MONTH	11	111	Jan-11	[MONTH].&[Jan-11]
5	MONTH	11	111	Feb-11	[MONTH].&[Feb-11]
6	MONTH	11	111	Mar-11	[MONTH].&[Mar-11]
7	QUARTER	11	211	(null)	[QUARTER].&[211]
8	MONTH	11	211	Apr-11	[MONTH].&[Apr-11]
9	MONTH	11	211	May-11	[MONTH].&[May-11]
10	MONTH	11	211	Jun-11	[MONTH].&[Jun-11]
11	QUARTER	11	311	(null)	[QUARTER].&[311]
12	MONTH	11	311	Jul-11	[MONTH].&[Jul-11]
13	MONTH	11	311	Aug-11	[MONTH].&[Aug-11]
14	MONTH	11	311	Sep-11	[MONTH].&[Sep-11]

26 分析ビュー・オブジェクト

分析ビューは、スター・スキーマやスノーフレーク・スキーマ、または集計データ、メジャー計算および説明的なメタデータを持つフラット(非正規化)ファクト表のコンテンツを簡単に拡張できるようにするため、およびデータにアクセスするのに必要なSQLを簡略化するために使用できるビューのタイプです。

分析ビューについては、次のトピックで説明しています。

- 分析ビューについて
- 分析ビューのメジャー
- 分析ビューの作成
- 計算済メジャーの例
- 属性のレポート
- フィルタ処理されたファクトおよび追加メジャーによる分析ビュー問合せ

26.1 分析ビューについて

分析ビューでは、データに対して階層/ディメンション・モデルを重ねて表示します。

分析ビューは、ディメンション表およびスター・スキーマまたはスノーフレーク・スキーマのファクト表で定義されています。ディメンション属性とファクト・データが同じ表にある非正規化表で分析ビューを定義することもできます。階層はディメンション表で定義されています。分析ビューは階層およびファクト表を参照します。

分析ビューはスター・スキーマとしてモデル化されたデータで定義されますが、データがスター・スキーマに格納されている必要はありません。ビューを使用して、他の形式で格納されたデータを分析ビューに示すことができます。一般に、スター・スタイル問合せで適切に動作する表またはビューは、分析ビューでも適切に機能します。ビューにおいてはデータ・セットが小さいほうが適切に動作します。データ・セットが大きい場合は、スター・スキーマの表においてより適切に動作します。パフォーマンスが最も良好なスキーマは、Oracle Database In-Memoryオプションを使用してインメモリー列ストアにロードされたスター・スキーマです。

インメモリー列ストアとともに使用した場合、分析ビューはSQL実行計画を最適化してインメモリー集計(つまり、ベクトル変換実行計画)を利用します。分析ビューでは、集計レベルの問合せをさらに高速化するために、マテリアライズド・ビューを利用できます(マテリアライズド・ビューはインメモリー列ストアにロードできます)。

分析ビューの最小要件は次のとおりです。

- ディメンション表(またはビュー)。この表には、値の一意のリストを提供し、ファクト表に結合される主キーが必要です。
- 少なくとも1つのファクト(メジャー)列およびディメンション表の主キーに結合されるキー列があるファクト表。

通常、分析ビューには次の特性があります。

- 複数のディメンション表により定義され、これによりデータのスライスおよびダイスが可能になります。
- 1つまたは複数のディメンション表に、異なるレベルの集計のデータが含まれています(例: 日、月、四半期および年)。

分析ビューは、属性ディメンション、階層および分析ビューの3種類のオブジェクトで構成されます。

属性ディメンションは、表またはビューを参照し、列を属性やレベルなどの高レベルのオブジェクトに編成するメタデータ・オブジェクトです。ディメンションおよび階層に関連するメタデータの多くは、属性ディメンション・オブジェクトで定義されます。

階層は、ビューのタイプです。階層は、属性ディメンション・オブジェクトを参照します。階層は、階層メンバー間の階層関係を使用してデータを編成します。階層の問合せは、詳細および集計レベルのキー(階層値)およびそれらの値の属性を返します。

分析ビューは、ファクト・データを返すビューのタイプです。分析ビューは、ファクト表と階層の両方を参照します。階層とメジャー・データのいずれも、分析ビューから選択されます。

26.2 分析ビューのメジャー

分析ビューのメジャーには、ファクト・データと計算、またはデータに対して実行する他の操作を指定します。

分析ビューの定義では、1つ以上のベース・メジャーおよび計算済メジャーを指定できます。

ベース・メジャー

ベース・メジャーはファクト表の列への参照です。オプションで分析ビューのデフォルトの集計方法をオーバーライドするmeas_aggregate_clauseを指定できます。各ベース・メジャーにはデフォルトの集計を指定できます。集計には、SUM、AVGなどの単純な操作、または属性ディメンションによって異なる複雑にネストした操作を使用できます。

default_aggregate_clauseを使用すると、meas_aggregate_clauseがないベース・メジャーにデフォルトの集計方法を指定できます。default_aggregate_clauseのデフォルト値はSUMです。

計算済メジャー

計算済メジャーは式であり、ユーザー定義の式、または多くの事前定義済のアナリティック計算のうちの1つを使用できます。計算済メジャーの式には、他のメジャー、行関数および階層関数を含めることができます。階層関数では、階層内の関連するメンバーの識別および処理に基づいて計算できます。式は分析ビューの他のメジャーを参照することもありますが、ファクト列を参照しないこともあります。計算では他のメジャーを参照できるため、ネストを使用して複雑な計算を簡単に作成できます。

計算済メジャーの式の定義では、分析ビューのメジャーを定義した順序に関係なく、分析ビューの他のメジャーを使用できます。唯一の制限は、計算に循環を使用できないことです。

分析ビューの定義において計算済メジャーを使用する以外に、分析ビューを問い合せるSELECT文に計算済メジャーを追加できます。これを行うには、文のWITHまたはFROM句でADD MEASURESキーワードを使用します。計算済メジャーの構文は、分析ビューの定義においてであれSELECT文内であれ同じです。

計算済メジャーの式のカテゴリを次に示します。

- 分析ビューのメジャー式
- 分析ビューの階層式
- 単純式
- 単一行関数式
- 複合式
- 日時式
- 期間式

分析ビューのメジャーの式には、次のような操作があります。

- LEADおよびLAG
- 修飾データ参照(QDR)
- ランク
- 関連するメンバー
- シェア

- ウィンドウの計算

関連トピック

- 追加メジャーによる分析ビュー問合せ

26.3 分析ビューの作成

分析ビューを作成する場合は、1つ以上の階層、および各階層に結合できる少なくとも1つのメジャー列を持つファクト表を指定します。

単純な分析ビューの作成

分析ビューには、ファクト表への参照、および階層に結合できるメジャーがある必要があります。

例26-1 単純な分析ビューの作成

この分析ビューでは、TIME_HIER階層およびSALE_FACT表が使用されています。単一のメジャーSALESが含まれています。

```
CREATE OR REPLACE ANALYTIC VIEW sales_av
USING sales_fact                      -- Refers to the SALES_FACT table
DIMENSION BY                          -- List of attribute dimensions
  (time_attr_dim                      -- TIME_ATTR_DIM attribute dimension
    KEY month_id REFERENCES month_id  -- Dimension key joins to fact column
    HIERARCHIES (                     -- List of hierarchies that use
      time_hier DEFAULT))             -- the attribute dimension
MEASURES                              -- List of measures
 (sales FACT sales)                   -- SALES measure references SALES column
DEFAULT MEASURE SALES;                -- Default measure of the analytic view
```

分析ビューから選択する問合せにフィルタがない場合は、大量の行が返される可能性があります。ただし、この問合せでは、SALES_AV分析ビューには86行のみを返す単一の階層が含まれています。

```
SELECT *
  FROM sales_av HIERARCHIES(time_hier)
  ORDER BY time_hier.hier_order;
```

返される値の一部を次に示します。

MEMBER_DESCRIPTION	LEVEL_NAME	HIER_ORDER	DEPTH	IS_LEAF	PARENT_LEVEL_NAME	PARENT_UNIQUE_NAME	SALES
(null)	ALL	0	0	0	(null)	(null)	364185
CY2011	YEAR	1	1	0	ALL	[ALL].[ALL]	67551
Q1CY2011	QUARTER	2	2	0	YEAR	[YEAR].&[1]	16252
January 2011	MONTH	3	3	1	QUARTER	[QUARTER].&[111]	5456
February 2011	MONTH	4	3	1	QUARTER	[QUARTER].&[111]	5
March 2011	MONTH	5	3	1	QUARTER	[QUARTER].&[111]	5630
Q2CY2011	QUARTER	6	2	0	YEAR	[YEAR].&[1]	17151
April 2011	MONTH	7	3	1	QUARTER	[QUARTER].&[211]	5563
May 2011	MONTH	8	3	1	QUARTER	[QUARTER].&[211]	5839
June 2011	MONTH	9	3	1	QUARTER	[QUARTER].&[211]	5748
Q3CY2011	QUARTER	10	2	0	YEAR	[YEAR].&[1]	16910

別のベース・メジャーの追加

分析ビューに別のベース・メジャーを追加するには、MEASURESリストにそのメジャーを含めます。

例26-2 分析ビューへのベース・メジャーの追加

```
CREATE OR REPLACE ANALYTIC VIEW sales_av
```

```
USING sales_fact
DIMENSION BY
  (time_attr_dim
    KEY month_id REFERENCES month_id
    HIERARCHIES (
      time_hier DEFAULT))
MEASURES
 (sales FACT sales,
  units FACT units)                    -- Add the UNITS base measure
DEFAULT MEASURE SALES;
```

分析ビューの問合せでは大量の行が返されることがあるため、通常、問合せではフィルタを使用して結果を制限します。この問合せは、WHERE句で期間をYEARレベルにフィルタ処理し、そのレベルのSALESおよびUNITSのデータのみが返されるようにしています。

```
SELECT time_hier.member_name as TIME,
 sales,
 units
FROM
 sales_av HIERARCHIES(time_hier)
WHERE time_hier.level_name = 'YEAR'
ORDER BY time_hier.hier_order;
```

返される値は次のとおりです。

	TIME	SALES	UNITS
1	CY2011	6755115980.73	24462444
2	CY2012	6901682398.95	24400619
3	CY2013	7240938717.57	24407259
4	CY2014	7579746352.89	24402666
5	CY2015	7941102885.15	24475206

分析ビューへの階層の追加

通常、分析ビューには1つ以上の属性ディメンションを使用する複数の階層があります。

例26-3 分析ビューへの階層の追加

この例は、属性ディメンションおよび階層を分析ビューのDIMENSION BYリストに追加しています。

```
CREATE OR REPLACE ANALYTIC VIEW sales_av
USING sales_fact
DIMENSION BY
  (time_attr_dim
    KEY month_id REFERENCES month_id
    HIERARCHIES (
      time_hier DEFAULT),
   product_attr_dim
    KEY category_id REFERENCES category_id
    HIERARCHIES (
      product_hier DEFAULT),
   geography_attr_dim
    KEY state_province_id
    REFERENCES state_province_id
    HIERARCHIES (
      geography_hier DEFAULT)
   )
MEASURES
 (sales FACT sales,
  units FACT units
```

```
  )
DEFAULT MEASURE sales;
```

次の問合せは、PRODUCT_HIER階層およびGEOGRAPHY_HIER階層をFROM句のHIERARCHIESフェーズに追加しています。

```
SELECT time_hier.member_name AS Time,
 product_hier.member_name AS Product,
 geography_hier.member_name AS Geography,
 sales,
 units
FROM
 sales_av HIERARCHIES (time_hier, product_hier, geography_hier)
WHERE time_hier.level_name in ('YEAR')
  AND product_hier.level_name in ('DEPARTMENT')
  AND geography_hier.level_name in ('REGION')
ORDER BY time_hier.hier_order,
  product_hier.hier_order,
  geography_hier.hier_order;
```

問合せは50行を戻します。次の図は、最初の20行のみを示しています。

	TIME	PRODUCT	GEOGRAPHY	SALES	UNITS
1	CY2011	Cameras and Camcorders	Africa	45634583.27	179017
2	CY2011	Cameras and Camcorders	Asia	202690278.2	797358
3	CY2011	Cameras and Camcorders	Europe	30943543.81	123376
4	CY2011	Cameras and Camcorders	North America	74533750.43	292386
5	CY2011	Cameras and Camcorders	Oceania	1475539.04	6015
6	CY2011	Cameras and Camcorders	South America	86722190.7	340517
7	CY2011	Computers	Africa	637720605.62	2021863
8	CY2011	Computers	Asia	2841504727.29	8982201
9	CY2011	Computers	Europe	440217216.49	1389905
10	CY2011	Computers	North America	1043206792.26	3296076
11	CY2011	Computers	Oceania	21628470.68	68639
12	CY2011	Computers	South America	1212972617.95	3832038
13	CY2011	Portable Music and Video	Africa	12006274.15	323900
14	CY2011	Portable Music and Video	Asia	53059837.6	1434864
15	CY2011	Portable Music and Video	Europe	8257455.15	222346
16	CY2011	Portable Music and Video	North America	19477358.02	527553
17	CY2011	Portable Music and Video	Oceania	398916.14	10999
18	CY2011	Portable Music and Video	South America	22665823.93	613391
19	CY2012	Cameras and Camcorders	Africa	46521568.18	178894
20	CY2012	Cameras and Camcorders	Asia	206589367.58	795253

例に使用されている表、分析ビュー・コンポーネント・オブジェクトおよび問合せを作成するSQLスクリプトは、Oracle Live SQL Webサイト(https://livesql.oracle.com/apex/livesql/file/index.html)で表示および実行できます。

26.4 計算済メジャーの例

計算済メジャーは、分析ビューのMEASURES句に*measure_name* AS (式)の形式で追加する式です。

LAG式の追加

この例は、LAG操作を使用する計算済メジャーをSALES_AV分析ビューに追加しています。

例26-4 LAG式の追加

```
CREATE OR REPLACE ANALYTIC VIEW sales_av
USING sales_fact
DIMENSION BY
  (time_attr_dim
    KEY month_id REFERENCES month_id
    HIERARCHIES (
      time_hier DEFAULT),
   product_attr_dim
    KEY category_id REFERENCES category_id
    HIERARCHIES (
      product_hier DEFAULT),
   geography_attr_dim
    KEY state_province_id REFERENCES state_province_id
    HIERARCHIES (
      geography_hier DEFAULT)
   )
MEASURES
 (sales FACT sales,
  units FACT units,
  sales_prior_period AS       -- Add a calculated measure.
    (LAG(sales) OVER (HIERARCHY time_hier OFFSET 1))
  )
DEFAULT MEASURE SALES;
```

YEARレベルとQUARTERレベルでSALESメジャーとSALES_PRIOR_PERIODメジャーを選択します。

```
SELECT time_hier.member_name as TIME,
 sales,
 sales_prior_period
FROM
 sales_av HIERARCHIES(time_hier)
WHERE time_hier.level_name IN ('YEAR','QUARTER')
ORDER BY time_hier.hier_order;
```

この問合せ結果の一部では、LAG式によって同じレベル内の前の期間が返されています。

	TIME	SALES	SALES_PRIOR_PERIOD
1	CY2011	6755115980.73	(null)
2	Q1CY2011	1625299627.35	(null)
3	Q2CY2011	1715160208.04	1625299627.35
4	Q3CY2011	1691017692.94	1715160208.04
5	Q4CY2011	1723638452.4	1691017692.94
6	CY2012	6901682398.95	6755115980.73
7	Q1CY2012	1644857783.16	1723638452.4
8	Q2CY2012	1752414181.93	1644857783.16
9	Q3CY2012	1732373411.73	1752414181.93
10	Q4CY2012	1772037022.13	1732373411.73
11	CY2013	7240938717.57	6901682398.95
12	Q1CY2013	1723571457.57	1772037022.13
13	Q2CY2013	1840985832.41	1723571457.57

SHARE OF式

シェアのメジャーは、親の行、祖先の行または現在のレベルのすべての行と現在の行の比率を計算します。たとえば、地理的メンバーとそのメンバーの親の比率です。シェアのメジャーは、SHARE OF式を使用して指定します。

例26-5 SHARE OF式の使用

この例は、SHARE OF操作を使用する計算済メジャーをSALES_AV分析ビューに追加しています。

```
CREATE OR REPLACE ANALYTIC VIEW sales_av
USING sales_fact
DIMENSION BY
  (time_attr_dim
    KEY month_id REFERENCES month_id
    HIERARCHIES (
      time_hier DEFAULT),
   product_attr_dim
    KEY category_id REFERENCES category_id
    HIERARCHIES (
      product_hier DEFAULT),
   geography_attr_dim
    KEY state_province_id REFERENCES state_province_id
    HIERARCHIES (
      geography_hier DEFAULT)
   )
MEASURES
 (sales FACT sales,
  units FACT units,
  -- Share of calculations
 sales_shr_parent_prod AS
   (SHARE_OF(sales HIERARCHY product_hier PARENT)),
 sales_shr_parent_geog AS
   (SHARE_OF(sales HIERARCHY geography_hier PARENT)),
 sales_shr_region AS
   (SHARE_OF(sales HIERARCHY geography_hier LEVEL REGION))
  )
DEFAULT MEASURE SALES;
```

SALES_SHR_PARENT_PRODメジャーは、CATEGORYレベルまたはDEPARTMENTレベルのSALES値とPRODUCT_HIER階層内の親のSALES値の比率を計算します(Total Server ComputersとComputersのSALESの比率など)。

この問合せは、PRODUCT_HIER階層の各レベルのCY2014のSALESメジャーおよびSALES_SHR_PARENT_PRODメジャーを選択しています。

```
SELECT time_hier.member_name AS Time,
 product_hier.member_name    AS Product,
 product_hier.level_name     AS Prod_Level,
 sales,
 ROUND(sales_shr_parent_prod,2) AS sales_shr_parent_prod
FROM
 sales_av HIERARCHIES (time_hier, product_hier)
WHERE time_hier.year_name = 'CY2014'
AND time_hier.level_name = 'YEAR'
ORDER BY product_hier.hier_order;
```

問合せの結果は次のとおりです。

	TIME	PRODUCT	PROD_LEVEL	SALES	SALES_SHR_PARENT_PROD
1	CY2014	ALL PRODUCTS	ALL	7579746352.89	(null)
2	CY2014	Cameras and Camcorders	DEPARTMENT	496952312.98	0.07
3	CY2014	Camcorders and Accessories	CATEGORY	154489927.29	0.31
4	CY2014	Cameras and Accessories	CATEGORY	342462385.69	0.69
5	CY2014	Computers	DEPARTMENT	6952712285.9	0.92
6	CY2014	All Computer Furniture	CATEGORY	23214339.8	0
7	CY2014	Computer Printers and Supplies	CATEGORY	1677409104.44	0.24
8	CY2014	PDAs	CATEGORY	7747497.6	0
9	CY2014	Total Personal Computers	CATEGORY	5133182348.24	0.74
10	CY2014	Total Server Computers	CATEGORY	111158995.82	0.02
11	CY2014	Portable Music and Video	DEPARTMENT	130081754.01	0.02
12	CY2014	Total iPlayer Family	CATEGORY	130081754.01	1

SALE_SHR_REGIONメジャーは、REGIONレベルのSALESに対するSTATEレベルまたはCOUNTRYレベルのSALESのシェアを計算します(たとえば、北米のSALESに対する米国カリフォルニア州のSALESの比率です)。

この問合せは、CY2014年の米国の州のSALESメジャーとSALES_SHR_REGIONメジャーの値を返します。

```
SELECT time_hier.member_name AS Time,
 geography_hier.member_name  AS Geography,
 geography_hier.level_name   AS Geog_Level,
 sales,
 ROUND(sales_shr_region,2)   AS sales_shr_region
FROM
 sales_av HIERARCHIES (time_hier, geography_hier)
WHERE time_hier.year_name = 'CY2014'
AND time_hier.level_name = 'YEAR'
AND geography_hier.country_name = 'United States'
AND geography_hier.level_name = 'STATE_PROVINCE'
ORDER BY geography_hier.hier_order;
```

問合せの結果は次のとおりです。

	TIME	GEOGRAPHY	GEOG_LEVEL	SALES	SALES_SHR_REGION
1	CY2014	California - US	STATE_PROVINCE	10990458.69	0.01
2	CY2014	Florida - US	STATE_PROVINCE	50867372.16	0.04
3	CY2014	Georgia - US	STATE_PROVINCE	57369538.31	0.05
4	CY2014	Illinois - US	STATE_PROVINCE	58648087.2	0.05
5	CY2014	Massachusetts - US	STATE_PROVINCE	41954923.92	0.03
6	CY2014	Michigan - US	STATE_PROVINCE	61579430.16	0.05
7	CY2014	Missouri - US	STATE_PROVINCE	56495320.12	0.04
8	CY2014	Nevada - US	STATE_PROVINCE	31457133.25	0.02
9	CY2014	New York - US	STATE_PROVINCE	49942020.98	0.04
10	CY2014	Ohio - US	STATE_PROVINCE	69715139.36	0.05
11	CY2014	Pennsylvania - US	STATE_PROVINCE	54751342.31	0.04
12	CY2014	Rhode Island - US	STATE_PROVINCE	28485913.48	0.02
13	CY2014	Tennessee - US	STATE_PROVINCE	24783302.86	0.02
14	CY2014	Texas - US	STATE_PROVINCE	44151509.32	0.03
15	CY2014	Virginia - US	STATE_PROVINCE	28255742.07	0.02
16	CY2014	Washington - US	STATE_PROVINCE	47650667.4	0.04

QDR式

qdr_expressionは、QUALIFYキーワードを使用して、メジャーの値を単一のディメンション・メンバーに制限します。例としては、CY2011年の売上、または現在の期間とCY2011の売上の相違率があります。QUALIFY式はKEY属性値を参照します。

例26-6 QUALIFY式の使用

SALES_2011メジャーおよびSALES_PCT_CHG_2011メジャーを使用してSALES_AV分析ビューを作成します。

```
CREATE OR REPLACE ANALYTIC VIEW sales_av
USING sales_fact
DIMENSION BY
  (time_attr_dim
    KEY month_id REFERENCES month_id
    HIERARCHIES (
      time_hier DEFAULT),
   product_attr_dim
    KEY category_id REFERENCES category_id
    HIERARCHIES (
      product_hier DEFAULT),
   geography_attr_dim
    KEY state_province_id REFERENCES state_province_id
    HIERARCHIES (
      geography_hier DEFAULT)
   )
MEASURES
 (sales FACT sales,
  units FACT units,
 -- Sales for CY2011
 sales_2011 AS
    (QUALIFY (sales, time_hier = year['11'])),
  -- Sales percent change from 2011.
  sales_pct_chg_2011 AS
    ((sales - (QUALIFY (sales, time_hier = year['11']))) /
    (QUALIFY (sales, time_hier = year['11'])))
  )
DEFAULT MEASURE SALES;
```

問合せのフィルタに関係なく、SALES_2011は常にCY2011年のデータを返します。SALES_PCT_CHG_2011メジャーは、現在の期間とCY2011の相違率を計算します。

この問合せは、YEARレベルおよびREGIONレベルでSALES、SALES_2011およびSALES_PCT_CHG_2011を選択します。

```
SELECT time_hier.member_name   AS Time,
 geography_hier.member_name    AS Geography,
 sales,
 sales_2011,
 ROUND(sales_pct_chg_2011,2) as sales_pct_chg_2011
FROM
 sales_av HIERARCHIES (time_hier, geography_hier)
WHERE time_hier.level_name = 'YEAR'
AND geography_hier.level_name = 'REGION'
ORDER BY geography_hier.hier_order,
 time_hier.hier_order;
```

問合せ結果の一部を示します。各行のSALES_2011には、CY2011のSALESが返されています。

	TIME	GEOGRAPHY	SALES	SALES_2011	SALES_PCT_CHG_2011
1	CY2011	Africa	695361463.04	695361463.04	0
2	CY2012	Africa	715142588.19	695361463.04	0.03
3	CY2013	Africa	746220583.3	695361463.04	0.07
4	CY2014	Africa	781333432.78	695361463.04	0.12
5	CY2015	Africa	818560024.79	695361463.04	0.18
6	CY2011	Asia	3097254843.09	3097254843.09	0
7	CY2012	Asia	3163782733.74	3097254843.09	0.02
8	CY2013	Asia	3322778863.3	3097254843.09	0.07
9	CY2014	Asia	3479067417.8	3097254843.09	0.12
10	CY2015	Asia	3644177245.26	3097254843.09	0.18

26.5 属性のレポート

階層の属性ディメンションの属性を使用し、分析ビューでそのデータを集計できます。

属性を使用して、データをフィルタ処理したり、レポートで表示したりできます。属性ごとにデータを取得(集計)することもできます。属性を使用して分析ビューに計算済メジャーを作成できます。分析ビューにはその属性の集計行が示されます。

例26-7 SEASON属性の使用

この例は、SEASONおよびSEASON_ORDERを属性として持つ属性ディメンションを最初に作成しています。これにより、階層および分析ビューでそれらの属性のメタデータを再使用して、属性を他のレベルに関連付けることができます。たとえば、SEASONはMONTH値によって決まります。

```
-- Create a time attribute dimension with a SEASON attribute.
CREATE OR REPLACE ATTRIBUTE DIMENSION time_attr_dim
DIMENSION TYPE TIME
USING time_dim
ATTRIBUTES
 (year_id,
  year_name,
  year_end_date,
  quarter_id,
  quarter_name,
  quarter_end_date,
  month_id,
  month_name,
  month_long_name,
  month_end_date,
  season,
  season_order)
LEVEL month
  LEVEL TYPE MONTHS
  KEY month_id
  MEMBER NAME month_name
  MEMBER CAPTION month_name
  MEMBER DESCRIPTION month_long_name
  ORDER BY month_end_date
  DETERMINES (quarter_id, season, season_order)
LEVEL quarter
  LEVEL TYPE QUARTERS
  KEY quarter_id
  MEMBER NAME quarter_name
  MEMBER CAPTION quarter_name
  MEMBER DESCRIPTION quarter_name
  ORDER BY quarter_end_date
  DETERMINES (year_id)
```

```
LEVEL year
  LEVEL TYPE YEARS
  KEY year_id
  MEMBER NAME year_name
  MEMBER CAPTION year_name
  MEMBER DESCRIPTION year_name
  ORDER BY year_end_date
LEVEL season
  LEVEL TYPE QUARTERS
  KEY season
  MEMBER NAME season
  MEMBER CAPTION season
  MEMBER DESCRIPTION season
  ORDER BY season_order;
```

MONTHがSEASONの子である階層を作成します。

```
CREATE OR REPLACE HIERARCHY time_season_hier
USING time_attr_dim
  (month CHILD OF
   season);
```

TIME_SEASON_HIER階層からデータを選択します。

```
SELECT member_name,
  member_unique_name,
  level_name,
  hier_order
FROM time_season_hier
ORDER BY hier_order;
```

問合せの結果では、TIME_SEASON_HIER階層によってALLレベル、SEASONSおよびMONTHSの行が返されています。この図は、返された最初の20行を示しています。

	MEMBER_NAME	MEMBER_UNIQUE_NAME	LEVEL_NAME	HIER_ORDER
1	ALL	[ALL].[ALL]	ALL	0
2	Spring	[SEASON].&[Spring]	SEASON	1
3	Mar-11	[MONTH].&[Mar-11]	MONTH	2
4	Apr-11	[MONTH].&[Apr-11]	MONTH	3
5	May-11	[MONTH].&[May-11]	MONTH	4
6	Mar-12	[MONTH].&[Mar-12]	MONTH	5
7	Apr-12	[MONTH].&[Apr-12]	MONTH	6
8	May-12	[MONTH].&[May-12]	MONTH	7
9	Mar-13	[MONTH].&[Mar-13]	MONTH	8
10	Apr-13	[MONTH].&[Apr-13]	MONTH	9
11	May-13	[MONTH].&[May-13]	MONTH	10
12	Mar-14	[MONTH].&[Mar-14]	MONTH	11
13	Apr-14	[MONTH].&[Apr-14]	MONTH	12
14	May-14	[MONTH].&[May-14]	MONTH	13
15	Mar-15	[MONTH].&[Mar-15]	MONTH	14
16	Apr-15	[MONTH].&[Apr-15]	MONTH	15
17	May-15	[MONTH].&[May-15]	MONTH	16
18	Summer	[SEASON].&[Summer]	SEASON	17
19	Jun-11	[MONTH].&[Jun-11]	MONTH	18
20	Jul-11	[MONTH].&[Jul-11]	MONTH	19

次に、SEASONの集計データを提供する分析ビューを作成します。

```
CREATE OR REPLACE ANALYTIC VIEW sales_av
USING sales_fact
DIMENSION BY
  (time_attr_dim
    KEY month_id REFERENCES month_id
    HIERARCHIES (
      time_hier DEFAULT,
      time_season_hier),
   product_attr_dim
    KEY category_id REFERENCES category_id
    HIERARCHIES (
      product_hier DEFAULT),
   geography_attr_dim
    KEY state_province_id
    REFERENCES state_province_id
    HIERARCHIES (
      geography_hier DEFAULT)
   )
MEASURES
 (sales FACT sales,
  units FACT units
  )
DEFAULT MEASURE SALES;
```

分析ビューからYEARおよびSEASONごとにSALESを直接選択できるようになりました。この問合せは、TIME_HIER階層およびTIME_SEASON_HIER階層からYEARレベルおよびSEASONレベルで選択します。

```
SELECT  time_hier.member_name      AS Time,
 time_season_hier.member_name      AS Season,
 ROUND(sales)                      AS Sales
FROM sales_av HIERARCHIES (time_hier, time_season_hier)
WHERE time_hier.level_name = 'YEAR'
  AND time_season_hier.level_name = 'SEASON'
ORDER BY time_hier.hier_order,
  time_season_hier.hier_order;
```

この問合せ結果の一部は、返された最初の12行を示しています。

	TIME	SEASON	SALES
1	CY2011	Spring	1703419821
2	CY2011	Summer	1708263225
3	CY2011	Fall	1708053033
4	CY2011	Winter	1635379902
5	CY2012	Spring	1740590320
6	CY2012	Summer	1749187977
7	CY2012	Fall	1751494549
8	CY2012	Winter	1660409552
9	CY2013	Spring	1828526039
10	CY2013	Summer	1835363103
11	CY2013	Fall	1838698718
12	CY2013	Winter	1738350857

例に使用されている表、分析ビュー・コンポーネント・オブジェクトおよび問合せを作成するSQLスクリプトは、Oracle Live SQL Webサイト(https://livesql.oracle.com/apex/livesql/file/index.html)で表示および実行できます。

26.6 フィルタ処理されたファクトおよび追加メジャーによる分析ビュー問合せ

分析ビューからSELECTする問合せには、分析ビューによってアクセスされるファクト・データを計算の前にフィルタ処理するFILTER FACTキーワードや、その問合せに対して追加の計算済メジャーを定義するADD MEASURESキーワードが含まれる場合があります。

関連トピック

- フィルタ処理されたファクトによる分析ビュー問合せ
- 追加メジャーによる分析ビュー問合せ
- フィルタ処理されたファクトおよび複数の追加メジャーによる分析ビュー問合せ

26.6.1 フィルタ処理されたファクトによる分析ビュー問合せ

分析ビューの問合せにおいて、分析ビューで上位レベルの階層メンバーのデータが集計される前に、ファクト・データをフィルタ処理できます。

分析ビューによって返される集計レコードの値は、分析ビューの階層、集計演算子、およびファクト表に含まれている行によって決まります。分析ビューを問い合せるSELECT文の述語は、分析ビューによって返される行を制限しますが、集計レコードの計算には影響しません。

SELECT文でFILTER FACTキーワードを使用すると、データが分析ビューで集計される前にファクト・レコードをフィルタ処理でき、指定された階層メンバーに対してのみ集計値が生成されます。

例26-8 集計前フィルタ述語がある場合とない場合の問合せ

次の問合せでは、sales_av分析ビューから階層メンバー名および売上値を選択します。問合せの述語により、階層メンバーがYEARレベルのものに制限されます。このフィルタ処理はメジャー値の集計には影響しません。

```
SELECT time_hier.member_name, TO_CHAR(sales, '999,999,999,999') AS sales
  FROM sales_av HIERARCHIES(time_hier)
  WHERE time_hier.level_name = 'YEAR'
  ORDER BY time_hier.hier_order;
```

問合せの結果は次のとおりです。結果にはYEARレベルの階層メンバーの集計されたメジャー値が含まれています。

```
MEMBER_NAME          SALES
-----------  -------------
CY2011       6,755,115,981
CY2012       6,901,682,399
CY2013       7,240,938,718
CY2014       7,579,746,353
CY2015       7,941,102,885
```

次の問合せでは、集計前に階層メンバーをフィルタ処理するインライン分析ビューを定義します。

```
SELECT time_hier.member_name, TO_CHAR(sales, '999,999,999,999') AS sales
  FROM ANALYTIC VIEW (                      -- inline analytic view
  USING sales_av HIERARCHIES(time_hier)
  FILTER FACT (time_hier TO level_name = 'MONTH'
               AND TO_CHAR(month_end_date, 'Q') IN (1, 2)
               )
  )
  WHERE time_hier.level_name = 'YEAR')
  ORDER BY time_hier.hier_order;
```

問合せの結果は次のとおりです。インライン分析ビューのFILTER FACT句により、最初の2四半期の月以外がすべて除外されます。結果にはこれらの四半期のYEARレベルの集計値が含まれています。集計には第3四半期と第4四半期の値は含まれま

せん。

```
MEMBER_NAME          SALES
-----------  -------------
CY2011       3,340,459,835
CY2012       3,397,271,965
CY2013       3,564,557,290
CY2014       3,739,283,051
CY2015       3,926,231,605
```

関連トピック

- フィルタ処理されたファクトおよび複数の追加メジャーによる分析ビュー問合せ

26.6.2 追加メジャーによる分析ビュー問合せ

ADD MEASURESのキーワードを使用して、分析ビューの問合せにメジャー計算を追加できます。

例26-9 FROM句でメジャーを追加する計算

この例には、sales_av分析ビューを使用して問合せに計算済メジャーshare_salesを追加するインライン分析ビューが含まれています。

```
SELECT time_hier.member_name AS "Member",
       TO_CHAR(sales, '999,999,999,999') AS "Sales",
       ROUND(share_sales, 2) AS "Share of Sales"
  FROM ANALYTIC VIEW (
    USING sales_av HIERARCHIES (time_hier)
    ADD MEASURES (
      share_sales as (SHARE_OF(sales HIERARCHY time_hier PARENT))
    )
  )
  WHERE time_hier.level_name IN ('ALL', 'YEAR')
  ORDER BY time_hier.hier_order;
```

問合せの結果は次のとおりです。

```
Member          Sales   Share of Sales
------  -------------  --------------
ALL     36,418,586,336
CY2011   6,755,115,981            0.19
CY2012   6,901,682,399            0.19
CY2013   7,240,938,718            0.2
CY2014   7,579,746,353            0.21
CY2015   7,941,102,885            0.22
```

例26-10 WITH句でメジャーを追加する計算

この例では前の例と同じ分析ビューが定義されていますが、SELECT文のWITH句を使用してそうしています。

```
WITH my_av ANALYTIC VIEW AS (
  USING sales_av HIERARCHIES (time_hier)
  ADD MEASURES (
    share_sales as (SHARE_OF(sales HIERARCHY time_hier PARENT))
  )
)
SELECT time_hier.member_name AS "Member",
       TO_CHAR(sales, '999,999,999,999') AS "Sales",
       ROUND(share_sales, 2) AS "Share of Sales"
  FROM my_av
  WHERE time_hier.level_name IN ('ALL', 'YEAR')
```

```
  ORDER BY time_hier.hier_order;
```

問合せの結果は前の例と同じです。

```
Member          Sales   Share of Sales
------  --------------  --------------
ALL     36,418,586,336
CY2011   6,755,115,981            0.19
CY2012   6,901,682,399            0.19
CY2013   7,240,938,718             0.2
CY2014   7,579,746,353            0.21
CY2015   7,941,102,885            0.22
```

関連トピック

- フィルタ処理されたファクトおよび複数の追加メジャーによる分析ビュー問合せ

26.6.3 フィルタ処理されたファクトおよび複数の追加メジャーによる分析ビュー問合せ

分析ビューの問合せでは、事前集計フィルタおよび追加メジャーを指定できます。

例26-11 問合せフィルタ・ファクトおよび複数の計算済メジャーを使用する問合せ

この問合せのWITH句の分析ビューは、sales_av分析ビューに基づいています。my_av分析ビューにより、time_hier階層のメンバーがQUARTERレベルの第1および第2四半期に、そしてgeography_hier階層のメンバーがCOUNTRYレベルのMexicoおよびCanadaにフィルタ処理されます。そして前期の売上と、前期売上からの売上の変化率を計算する計算済メジャーを追加します。

```
WITH my_av ANALYTIC VIEW AS (
  USING sales_av HIERARCHIES (time_hier, geography_hier)
  FILTER FACT (time_hier TO level_name = 'QUARTER'
               AND (quarter_name LIKE 'Q1%' OR quarter_name LIKE 'Q2%'),
               geography_hier TO level_name = 'COUNTRY'
               AND country_name IN ('Mexico', 'Canada'))
  ADD MEASURES (sales_pp AS
                   (LAG(sales) OVER (HIERARCHY time_hier OFFSET 1)),
                sales_pp_pct_change AS
                   (LAG_DIFF_PERCENT(sales) OVER (HIERARCHY time_hier OFFSET 1)))
  )
 SELECT time_hier.member_name AS time,
   geography_hier.member_name AS geography,
   sales,
   sales_pp,
   ROUND(sales_pp_pct_change,3) AS "Change"
 FROM my_av HIERARCHIES (time_hier, geography_hier)
 WHERE time_hier.level_name IN ('YEAR') AND
        geography_hier.level_name = 'REGION'
 ORDER BY time_hier.hier_order;
```

結果は次のようになります。

```
TIME    GEOGRAPHY            SALES     SALES_PP     Change
------  -------------  ------------  -----------  ------
CY2011  North America  229,884,616
CY2012  North America  233,688,485   229,884,616    .017
CY2013  North America  245,970,470   233,688,485    .053
CY2014  North America  256,789,511   245,970,470    .044
CY2015  North America  270,469,199   256,789,511    .053
```

関連トピック

- フィルタ処理されたファクトによる分析ビュー問合せ
- 追加メジャーによる分析ビュー問合せ

用語集

加算的

加算することでサマリーできるファクト(またはメジャー)を示します。加算ファクトは、最も一般的なタイプのファクトです。たとえば、販売価格、原価、収益などがこれに該当します。「非加算的」および「準加算的」と対比してください。

アドバイザ

「SQLアクセス・アドバイザ」を参照してください。

集計

サマリーされたデータ。たとえば、特定製品の売上数量を1日、1か月、四半期および1年ごとに集計できます。

集計操作

複数のデータ値を1つの値に集約する処理。たとえば、1日単位で集めた販売データを週レベルに集計したり、週のデータを月レベルに集計するなどの処理がこれに該当します。その後、データは集計データとして参照できます。集計という用語はサマリーと同義であり、集計データはサマリー・データと同義です。

分析ビュー

ファクト・データの集計、計算および結合をカプセル化するビューのタイプ。分析ビューは、ディメンション・モデルを使用してデータを編成します。これらを使用すると、集計および計算をデータ・セットに簡単に追加でき、比較的単純なSQLで問い合せることができるデータをビューに表示できます。

祖先

階層内で、特定の値よりも上位のレベルにある値。たとえば、Timeディメンションでは、値1999は値Q1-99とJan-99の祖先です。

属性

1つ以上のレベルの特徴を説明した特性。たとえば、衣料品製造業の製品ディメンションには品目と呼ばれるレベルが含まれ、その中に色という属性があります。属性は、エンド・ユーザーが類似の特性に基づいてデータを選択できる論理グループを表します。

リレーショナル・モデルにおける属性は、エンティティの特性として定義されます。Oracle Database 10*g*の場合、属性は単一レベルの各要素を特徴付けるディメンションの列です。

属性ディメンション

データ・ソース、および属性ディメンションの属性であるデータ・ソースの列を指定します。メンバーのレベルを指定して、レベル間の属性リレーションシップを決定します。属性ディメンションは、階層および分析ビューによって使用されます。

カーディナリティ

OLTPの観点では、表内の行数を指します。データ・ウェアハウスの観点では、一般に、列内の個別値の数を指します。データ・ウェアハウスのほとんどのDBAにとっては、カーディナリティ度のほうがより重要な問題点です。

子

階層内で、特定の値の直下のレベルにある値のことです。たとえば、Timeディメンションでは、値Jan-99は値Q1-99の子です。子値が複数の階層に属している場合は、1つの値が複数の親の子になることもあります。

クレンジング

ソース・データの非一貫性を解決し、異常を修正する処理。通常は、ETL処理の一部です。

コモン・ウェアハウス・メタデータ(CWM)

Oracleデータ・ウェアハウスおよび意思決定支援で使用される標準リポジトリ。CWMリポジトリ・スキーマは他の製品が共有できるスタンドアロン製品で、それぞれ、その製品が作成するCWMリポジトリ内のオブジェクトのみを所有します。

クロス積

複数セットの要素群を組み合せる方法。たとえば、2つの列がある場合、最初の列の各要素は2番目の列の各要素と組み合せられます。単純例を次に示します。

```
Col1   Col2   Cross Product
----   ----   -------------
a      c      ac
b      d      ad
              bc
              bd
```

クロス積は、グルーピング・セットの連結時に行われます。データ・ウェアハウスにおける集計のためのSQLを参照してください。

データ・マート

販売、マーケティング、金融など、特定のビジネス分野に対して設計されたデータ・ウェアハウス。依存型のデータ・マートの場合、データは企業全体のデータ・ウェアハウスから導出されます。非依存型のデータ・マートの場合、データはソースから直接収集されます。

データソース

ウェアハウスにデータを提供するデータベース、アプリケーション、リポジトリまたはファイル。

データ・ウェアハウス

トランザクション処理用ではなく、問合せおよび分析用に設計されたリレーショナル・データベース。データ・ウェアハウスには、通常、トランザクション・データから導出された履歴データが含まれますが、別のソースからのデータを含めることもできます。データ・ウェアハウスにより、分析ワークロードとトランザクション・ワークロードを分離できます。また企業は、複数のソースのデータを統合できるようになります。

データ・ウェアハウス環境は、リレーショナル・データベースに加え、ETLソリューション、分析SQLエンジン、クライアント分析ツール、およびデータ収集とビジネス・ユーザーへのデータ配信の処理を管理するその他のアプリケーションで構成されることが多いです。

カーディナリティ度

表内の列の個別値の数を表内の行の合計数で割ったものです。これは、作成する索引を決定する際に特に重要です。通常、カーディナリティ度の低い列にはビットマップ索引、カーディナリティ度の高い列にはBツリー索引を使用します。原則として、カーディナリティ度が1%未満の場合にビットマップ索引を使用します。

非正規化

表内の冗長性を許す処理。「正規化」と対比してください。

導出ファクト(またはメジャー)

算術演算またはデータ変換を使用して既存のデータから生成されたファクト(またはメジャー)。例としては、平均、合計、割合、差などがあります。

ディテール

「ファクト表」を参照してください。

ディテール表

「ファクト表」を参照してください。

ディメンション

一般に、2通りの方法で使用されます。

- データセットのメンバーを指定するために使用される特性を示す一般的な用語。売上指向のデータ・ウェアハウスにおける最も一般的なディメンションは、時間、地理および製品の3つです。ほとんどのディメンションが階層を持ちます。
- 問合せがディメンションをナビゲートできるようにデータベース内に定義されたオブジェクト。Oracle Database 10*g*の場合、ディメンションは、1組の列セット間の階層(親/子)関係を定義するデータベース・オブジェクトです。Oracle Expressの場合、ディメンションは値リストで構成されるデータベース・オブジェクトです。

ディメンション表

時間、部門、所在地、製品などの階層情報および分類情報として表される企業のビジネス・エンティティを記述します。参照表と呼ばれる場合もあります。

ディメンション値

ディメンションを構成するリストの一要素。たとえば、コンピュータ会社では、製品ディメンションにLAPPCやDESKPCなどのディメンション値を持ちます。地理ディメンション内の値には、BostonおよびParisなどが含まれる場合があります。また、時間ディメンション内の値には、MAY96やJAN97などが含まれる場合があります。

ドリル

1つの項目から一連の関連項目にナビゲートすることです。ドリル操作は通常、ある階層内のレベル内(またはレベル間)での上下へのナビゲートを伴います。データを選択する際、階層内でドリルダウンすれば階層が開き、ドリルアップすれば階層が閉じます。

ドリルダウン

ビューを拡張して、親の値に関連付けられている子の値を階層内に含めることです。

ドリルアップ

階層内で親の値に関連付けられている子の値のリストを閉じることです。

要素

オブジェクトまたはプロセス。たとえば、ディメンションはオブジェクト、マッピングはプロセスであり、両方とも要素です。

エンタープライズ・データ・ウェアハウス

生データが1つの記憶域の場所に統合され、データ・ウェアハウス・アーキテクチャの中心として使用されているデータ・ウェアハウス。

エンティティ

データベースのモデル化に使用されます。リレーショナル・データベースでは通常、表にマップされます。

ELT

ELTは、Extraction(抽出)、Loading(ロード)、Transformation(変換)およびTransportation(転送)の略です。これは、古いETLの最新バージョンです。

ETL

ETLは、Extraction(抽出)、Transformation(変換)およびLoading(ロード)の略です。ETLとは、ソース・データにアクセスして操作を行い、データ・ウェアハウスへロードする方法を意味します。これらの処理の実行順序は様々です。

ETLのかわりに、ETT(extraction, transformation, transportation)やETM(extraction, transformation, move)が使用される場合もあります。

抽出

ETLの初期フェーズにおいてソースからデータを取り出す処理。

ファクト

調査や分析の対象となるデータで、通常は数値データや加算的データ。たとえば、販売価格、原価、収益などがこれに該当します。ファクトとメジャーは同じ意味で、ファクトは主にリレーショナル環境で使用され、メジャーは主に多次元環境で使用されます。導出ファクト(またはメジャー)は、算術演算やデータ変換を使用して既存のデータから生成されます。

ファクト表

ファクトを格納する、スター・スキーマ内の表。多くの場合、ファクト表には、ファクトを格納する列と、ディメンション表の外部キーとなる列の2種類の列があります。通常、ファクト表の主キーは、その表のすべての外部キーで構成されるコンポジット・キーです。

ファクト表には、詳細レベルのファクトまたは集計されたファクト(集計されたファクトを含むファクト表は、サマリー表と呼ばれることが多い)のいずれかが含まれています。通常、ファクト表には同じ集計操作レベルのファクトが含まれています。

高速リフレッシュ

マテリアライズド・ビューに対して変更されたデータのみを適用する操作。この操作によって、マテリアライズド・ビューを一から再作成する必要がなくなります。

ファイルから表へのマッピング

フラット・ファイルからウェアハウス内の表へのデータのマップ。

階層

データを編成する手段として順序付けされたレベルを使用する論理構造。データ集計を定義するために使用できます。たとえば、時間ディメンションでは、階層を使用して月レベルから四半期レベル、年レベルへとデータを集計できます。階層は、Oracleでディメンション・オブジェクトの一部として定義できます。また、ドリル操作のナビゲーション・パスの定義にも使用できますが、この場合、階層内のレベルは必ずしも集計された合計を示している必要はありません。

階層は、属性ディメンションのレベル間の階層関係を定義するビューのタイプであるデータ・ディクショナリ・オブジェクトである場合もあります。属性ディメンションおよび階層は、分析ビューのディメンション・メンバーを提供します。

レベル

階層内での位置。たとえば、時間ディメンションは、月レベル、四半期レベルおよび年レベルのデータを表す階層を持ちます。

レベル値の表

ディメンションおよび階層の一部として作成したレベルの値またはデータを格納するデータベース表。

マッピング

ソース・オブジェクトとターゲット・オブジェクトとの間の関係およびデータ・フローに関する定義。

マテリアライズド・ビュー

ファクト表(場合によってはディメンション表)の集計データまたは結合データで構成される事前計算表。サマリーまたは集計表とも呼ばれます。

マテリアライズド・ビュー・ログ

特定のマテリアライズド・ビューに関する詳細を記録するログ。高速リフレッシュを使用するにはマテリアライズド・ビュー・ログが必要ですが、パーティション・チェンジ・トラッキング・リフレッシュの場合は例外です。

メジャー

「ファクト」を参照してください。

メタデータ

データおよびその他の構造(オブジェクト、ビジネス・ルール、ビジネス・プロセスなど)を記述するデータ。たとえば、データ・ウェアハウスのスキーマ設計は、通常、メタデータとしてリポジトリに格納され、データ・ウェアハウスの作成と移入に使用するスクリプトを生成するために使用されます。メタデータはリポジトリに含まれます。

データの例: ソースからターゲットへの変換に関する定義、データ・ウェアハウスの作成と移入に使用されます。情報の例: 表、列、関連項目の定義、関連するモデル・ツール内に格納されます。ビジネス・ルールの例: 1,000個を販売した後10パーセントの値引を行います。

モデル

作成する内容を示すオブジェクト。典型的なスタイル、計画、設計。また、データ・ウェアハウスの構造を定義するメタデータにもなります。

非加算

加算することでサマリーできないファクト(またはメジャー)を示します。非加算の例には、平均があります。「加算的」および「準加算的」と対比してください。

正規化

リレーショナル・データベースにおいて、データを複数の表に分離することによりデータの冗長性を取り除くプロセス。「非正規化」と対比してください。

データを複数の表に分割し、データの冗長性を排除する処理。

OLTP

「オンライン・トランザクション処理(OLTP)」を参照してください。

オンライン・トランザクション処理(OLTP)

オンライン・トランザクション処理。OLTPシステムは、高速で信頼性の高いトランザクション処理用に最適化されています。データ・ウェアハウス・システムに比べると、ほとんどのOLTPシステムには、比較的少数の行と多数の表のグループが含まれます。

パラレル実行

いくつかのプロセスが作業の一部を処理できるようにタスクを分解することです。複数のCPUがそれぞれの部分を同時に実行すると、パフォーマンスを大きく向上させることができます。

パラレル化

いくつかのプロセスが作業の一部を処理できるようにタスクを分解することです。複数のCPUがそれぞれの部分を同時に実行すると、パフォーマンスを大きく向上させることができます。

親

階層内の所定の値より上のレベルにある値。たとえば、時間ディメンションでは、値Q1-99(99年第1四半期)は、子値Jan-99(99年1月)の親とされる場合があります。

パーティション

非常に大きな表および索引は、操作が難しく時間がかかる可能性があります。管理性を改善するために、表と索引をパーティションと呼ばれる小さい部分に分解できます。

パーティション・チェンジ・トラッキング(PCT)

マテリアライズド・ビューの失効を、パーティション・レベルまたはサブパーティション・レベルで追跡する方法。

パターン一致

MATCH_RECOGNIZE句を使用して一連の行のパターンを認識する方法。

ピボット

入力ストリーム内の各レコードが、データ・ウェアハウスの適切な表にある多数のレコードに変換される変換処理。これは、リレーショナルでないデータベースからデータを取り出す際に特に重要です。

クエリー・リライト

マテリアライズド・ビュー(事前に計算されたもの)を使用して問合せに迅速に答えるメカニズム。

リフレッシュ

マテリアライズド・ビューを変更して新しいデータを反映するメカニズム。

リライト

「クエリー・リライト」を参照してください。

スキーマ

関連するデータベース・オブジェクトの集まり。リレーショナル・スキーマは、データベース・ユーザーIDでグルーピングされ、表やビューなどのオブジェクトを含みます。このマニュアルでは、shというサンプル・スキーマを使用しています。特殊なタイプのスキーマとして、スノーフレーク・スキーマおよびスター・スキーマの2つがあります。

準加算的

全ディメンションについてではなく、一部のディメンションによって加算することでサマリーできるファクト(またはメジャー)を示します。準加算の例には、人数や手持在庫があります。「加算的」および「非加算的」と対比してください。

スライスおよびダイス

データの取得および操作を指す非公式用語。データ・ウェアハウスは、それぞれの軸がディメンションを表したデータのキューブ(立方体)と見ることができます。データをスライスするとは、ディメンションの一部または全部のメジャーと値を指定してキューブのピース(スライス)を取得することです。データ・スライスの取得時に、スライスを細切れ(ダイス)したように多数の小さなピースにし、データ列と行を移動したり並べ替えることもできます。適切にスライスおよびダイスされたシステムでは、大量のデータのナビゲーションが容易になります。

スノーフレーク・スキーマ

各ディメンション表の一部または全部が正規化されたタイプのスター・スキーマ。

ソース

データ・ウェアハウス内のデータが導出されるデータベース、アプリケーション、ファイルまたはその他のデータ保管場所。

ソース・システム

データ・ウェアハウス内のデータが導出されるデータベース、アプリケーション、ファイルまたはその他のデータ保管場所。

ソース・テーブル

ソース・データベース内の表。

SQLアクセス・アドバイザ

SQLアクセス・アドバイザは、ユーザーが目標とするパフォーマンスを実現できるように、特定のワークロードに適切なマテリアライズド・ビューのセット、マテリアライズド・ビュー・ログ、パーティション、索引を推奨します。Oracle Enterprise ManagerのGUIであり、DBMS_ADVISORパッケージと同様の機能を持ちます。

ステージング・エリア

ウェアハウスに入る前にデータが処理される場所。

ステージング・ファイル

ウェアハウスに入る前のデータ処理に使用されるファイル。

スター・クエリー

ファクト表および多数のディメンション表を結合するものです。各ディメンション表は、主キーから外部キーへの結合を使用してファクト表に結合されます。ただし、ディメンション表同士は結合されません。

スター・スキーマ

多次元データ・モデルを表現するように設計されたリレーショナル・スキーマ。1つ以上のファクト表と、外部キーを介して関連付けられている1つ以上のディメンション表で構成されます。

サブジェクト領域

組織の役割、知識領域を表現したり、識別するための分類方法。通常、1つのデータ・マートは、販売、マーケティングまたは地域などの1つのサブジェクト領域をサポートするために開発されます。

サマリー

「マテリアライズド・ビュー」を参照してください。

サマリー・アドバイザ

現在は、SQLアクセス・アドバイザに置き換えられています。

ターゲット

ETL処理過程において中間的または最終的な結果を保持します。ETL処理全体のターゲットは、データ・ウェアハウスです。

第3正規形

正規化を通してデータの冗長性を最小化する、古典的なリレーショナル・データベース・モデリング技法。

第3正規形スキーマ

OLTPシステムで一般的に使用されているものと同じ種類の正規化を使用するスキーマ。大規模なデータ・ウェアハウス、特に、データのロード要求が多く、データ・マートへのデータの入力および長時間実行問合せの実行に使用される環境用として選択されることがあります。「スノーフレーク・スキーマ」および「スター・スキーマ」と比較してください。

変換

データを操作する処理。コピー操作以外の操作は変換です。変換の例には、複数のソース・テーブルからのデータのクレンジング、集計および統合があります。

転送

コピーまたは変換したデータをソースからデータ・ウェアハウスに移動する処理。「変換」と比較してください。

一意識別子

同じ項目が複数の場所に表示される場合に、その項目を区別することを目的とする識別子。

更新ウィンドウ

ウェアハウスの更新に使用できる時間の長さ。たとえば、夜間の8時間がウェアハウスの更新に当てられます。

更新頻度

新しいデータでデータ・ウェアハウスが更新される頻度。たとえば、ウェアハウスはOLTPシステムから毎晩更新できます。

妥当性チェック

メタデータ定義および構成パラメータを検証する処理。

バージョニング

新規要件および変更に対応して、データ・ウェアハウス・プロジェクトの新規バージョンを作成する機能。

索引

数字

A

B

C

D

E

F

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

- タイプ 14.1.3
- SQL文の場合の使用方法 14.2.6.2
- SQLワークロードの場合の使用方法 14.2.6.1
- 使用例 14.1.5
- 属性クラスタリング 14.1.2