

Oracle® Database

2日でデータ・ウェアハウス・ガイド

11g リリース2 (11.2)

部品番号: B56298-01

2009年12月

Oracle Database 2日でデータ・ウェアハウス・ガイド, 11g リリース2 (11.2)

部品番号: B56298-01

Oracle Database 2 Day + Data Warehousing Guide, 11g Release 2 (11.2)

原本部品番号: E10578-02

Copyright © 2007, 2009, Oracle and/or its affiliates. All rights reserved.

制限付権利の説明

このソフトウェアおよび関連ドキュメントの使用と開示は、ライセンス契約の制約条件に従うものとし、知的財産に関する法律により保護されています。ライセンス契約で明示的に許諾されている場合もしくは法律によって認められている場合を除き、形式、手段に関係なく、いかなる部分も使用、複写、複製、翻訳、放送、修正、ライセンス供与、送信、配布、発表、実行、公開または表示することはできません。このソフトウェアのリバース・エンジニアリング、逆アセンブル、逆コンパイルは互換性のために法律によって規定されている場合を除き、禁止されています。

ここに記載された情報は予告なしに変更される場合があります。また、誤りが無いことの保証はいたしかねます。誤りを見つけた場合は、オラクル社までご連絡ください。

このソフトウェアまたは関連ドキュメントが、米国政府機関もしくは米国政府機関に代わってこのソフトウェアまたは関連ドキュメントをライセンスされた者に提供される場合は、次のNoticeが適用されます。

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

このソフトウェアは様々な情報管理アプリケーションでの一般的な使用のために開発されたものです。このソフトウェアは、危険が伴うアプリケーション（人的傷害を発生させる可能性があるアプリケーションを含む）への用途を目的として開発されていません。このソフトウェアを危険が伴うアプリケーションで使用する場合、このソフトウェアを安全に使用するために、適切な安全装置、バックアップ、冗長性（redundancy）、その他の対策を講じることは使用者の責任となります。このソフトウェアを危険が伴うアプリケーションで使用したこと起因して損害が発生しても、オラクル社およびその関連会社は一切の責任を負いかねます。

OracleはOracle Corporationおよびその関連企業の登録商標です。その他の名称は、それぞれの所有者の商標または登録商標です。

このソフトウェアおよびドキュメントは、第三者のコンテンツ、製品、サービスへのアクセス、あるいはそれらに関する情報を提供することがあります。 オラクル社およびその関連会社は、第三者のコンテンツ、製品、サービスに関して一切の責任を負わず、いかなる保証もいたしません。 オラクル社およびその関連会社は、第三者のコンテンツ、製品、サービスへのアクセスまたは使用によって損失、費用、あるいは損害が発生しても、一切の責任を負いかねます。

目次

[図一覧](#)

[表一覧](#)

[タイトルおよび著作権情報](#)

[はじめに](#)

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

[第1部 データ・ウェアハウスの構築](#)

[1 データ・ウェアハウスの概要](#)

- [このガイドについて](#)
 - [このガイドを使用する前に](#)
 - [このガイドの対象外](#)
- [データ・ウェアハウスについて](#)
 - [データ・ウェアハウスの主な特性](#)
 - [Oracleデータ・ウェアハウスの一般的なタスク](#)
- [このガイドで説明するタスク](#)
- [データ・ウェアハウス管理用ツール](#)

[2 データ・ウェアハウス・システムの設定](#)

- [データ・ウェアハウス・システムを設定する一般的な手順](#)
- [環境の準備](#)
 - [適切なハードウェア設定](#)
 - [必要なCPU数およびクロック・スピードについて](#)
 - [必要なメモリーについて](#)
 - [必要なディスク数について](#)
 - [十分なI/O帯域幅を判断する方法について](#)
 - [データ・ウェアハウスのハードウェア構成の検証](#)
 - [ddユーティリティについて](#)
 - [例: ddユーティリティの使用](#)
 - [Orionユーティリティについて](#)
 - [例: Orionユーティリティの使用](#)
- [データ・ウェアハウスのデータベースの設定](#)
 - [メモリー管理パラメータの設定方法](#)
 - [例: 初期化パラメータの設定](#)
 - [重要な他の初期化パラメータの設定](#)
- [Oracle Warehouse Builderへのアクセス](#)
 - [Oracle Warehouse Builderデモンストレーションのインストール](#)

[3 データ・ソースの識別およびメタデータのインポート](#)

- [データ・ソースの概要](#)
- [ソースからメタデータをインポートする一般的な手順](#)
- [OWBのワークスペース、プロジェクトおよびその他のデバイスについて](#)
- [例: フラット・ファイルからのメタデータのインポート](#)
 - [フラット・ファイルの場所の指定](#)
 - [プロジェクトでのモジュールの作成](#)
 - [「インポート・メタデータ・ウィザード」の起動](#)
 - [「フラット・ファイル・サンプル・ウィザード」の使用](#)
 - [フラット・ファイル・データのインポート](#)

4 Oracle Warehouse Builderでのウェアハウスの定義

- [リレーショナル・ターゲット・ウェアハウスを定義する一般的な手順](#)
- [ウェアハウス・ターゲット・スキーマの指定](#)
- [OWBのフラット・ファイルのソースについて](#)
 - [演習: ターゲット・モジュールへの外部表の追加](#)
- [ディメンションについて](#)
 - [演習: ディメンションの理解](#)
 - [レベルについて](#)
 - [レベル属性の定義](#)
 - [階層の定義](#)
 - [ディメンション・ロール](#)
 - [レベルの関係](#)
 - [ディメンションの例](#)
 - [制御行](#)
 - [ディメンションの実装](#)
 - [スター・スキーマ](#)
 - [バインド](#)
- [キューブについて](#)
 - [キューブの定義](#)
 - [キューブ・メジャー](#)
 - [キューブのディメンション](#)
 - [キューブの例](#)
 - [キューブの実装](#)
 - [キューブのリレーショナル実装](#)
 - [バインド](#)

第II部 データ・ウェアハウスへのデータのロード

5 ETLロジックの定義

- [マッピングおよび演算子について](#)
- [マッピングを定義する手順の要約](#)
- [マッピングの作成](#)
 - [演算子のタイプ](#)
- [演算子の追加](#)
 - [ワークスペース・オブジェクトにバインドする演算子の追加](#)
 - [バインドされていない演算子を属性なしで作成](#)
 - [既存のワークスペース・オブジェクトから選択してバインド](#)
- [演算子の編集](#)
- [演算子の接続](#)
 - [属性の接続](#)
 - [グループの接続](#)
 - [例: マッピング・エディタによるステージング領域表の作成](#)
- [マッピング・プロパティの設定](#)
 - [ターゲット・ロード順序](#)
 - [デフォルトにリセット](#)
- [演算子、グループおよび属性プロパティの設定](#)
- [演算子とワークスペース・オブジェクトの同期化](#)

- [演算子の同期化](#)
- [ワークスペース・オブジェクトから演算子への同期化](#)
 - [ワークスペース・オブジェクトに基づく演算子の同期化](#)
- [演算子からワークスペース・オブジェクトへの同期化](#)

6 [ターゲット・スキーマへの配布およびETLロジックの実行](#)

- [配布について](#)
 - [コントロール・センターの概要](#)
 - [配布の物理詳細の構成](#)
 - [配布アクション](#)
 - [配布プロセス](#)
- [オブジェクトの配布](#)
- [ETLジョブの開始](#)
 - [データの表示](#)

第III部 [データ・ウェアハウスのレポート作成](#)

7 [レポートおよび分析のSQL](#)

- [ビジネス問合せに応答するSQLアナリティック機能の使用](#)
 - [ROLLUP関数を使用したレポートへの合計の追加方法](#)
 - [ROLLUP関数を使用するとき](#)
 - [例: ROLLUP関数の使用](#)
 - [CUBE関数を使用した異なるレベルでの合計の分割方法](#)
 - [CUBE関数を使用するとき](#)
 - [例: CUBE関数の使用](#)
 - [GROUPING関数を使用した小計の追加方法](#)
 - [GROUPING関数を使用するとき](#)
 - [例: GROUPING関数の使用](#)
 - [GROUPING SETS関数を使用した集計の結合方法](#)
 - [GROUPING SETS関数を使用するとき](#)
 - [例: GROUPING SETS関数の使用](#)
 - [RANK関数を使用したランキングの計算方法](#)
 - [RANK関数を使用するとき](#)
 - [例: RANK関数の使用](#)
 - [合計に対する相対的な寄与率の計算方法](#)
 - [例: 総計に対する相対的な寄与率の計算](#)
 - [ウィンドウ関数を使用した行間計算の実行方法](#)
 - [例: 行間計算の実行](#)
 - [ウィンドウ関数を使用した移動平均の計算方法](#)
 - [例: 移動平均の計算](#)
- [パーティション外部結合を使用したスパースなデータの処理](#)
 - [パーティション外部結合を使用するとき](#)
 - [例: パーティション外部結合の使用](#)
- [ビジネス問合せを単純化するWITH句の使用](#)
 - [WITH句を使用するとき](#)
 - [例: WITH句の使用](#)

第IV部 [データ・ウェアハウスの管理](#)

8 [データ・ウェアハウスのリフレッシュ](#)

- [データ・ウェアハウスのリフレッシュについて](#)
 - [例: データ・ウェアハウスのリフレッシュ](#)
- [ローリング・ウィンドウを使用したデータのオフロード](#)

- [例: ローリング・ウィンドウの使用](#)

9 データ・ウェアハウスの操作の最適化

- [システムのオーバーロードの回避](#)
 - [システム・パフォーマンスの監視](#)
 - [パラレル実行パフォーマンスの監視](#)
 - [I/Oの監視](#)
 - [データベース・リソース・マネージャの使用](#)
- [索引およびマテリアライズド・ビューの使用の最適化](#)
 - [例: SQLアクセス・アドバイザを使用した索引およびマテリアライズド・ビューの使用の最適化](#)
- [記憶域要件の最適化](#)
 - [データ圧縮による記憶域の改善](#)

10 パフォーマンス・ボトルネックの排除

- [SQLの効率的な実行の検証](#)
 - [オプティマイザ統計の分析](#)
 - [実行計画の分析](#)
 - [例: 実行計画の出力の分析](#)
 - [ヒントを使用したデータ・ウェアハウスのパフォーマンスの向上](#)
 - [例: ヒントを使用したデータ・ウェアハウスのパフォーマンスの向上](#)
 - [アドバイザを使用したSQLパフォーマンスの検証](#)
- [リソース使用量の最小化によるパフォーマンスの向上](#)
 - [パフォーマンスの向上: パーティション化](#)
 - [パフォーマンスの向上: パーティション・プルーニング](#)
 - [パフォーマンスの向上: パーティションワイズ結合](#)
 - [例: SQLアクセス・アドバイザを使用したパーティション化の評価](#)
 - [パフォーマンスの向上: 問合せのリライトおよびマテリアライズド・ビュー](#)
 - [パフォーマンスの向上: 索引](#)
 - [パフォーマンスの向上: 圧縮](#)
 - [パフォーマンスの向上: DBMS_COMPRESSIONパッケージ](#)
 - [パフォーマンスの向上: CREATE TABLEおよびALTER TABLEのtable_compress句](#)
- [最適なリソースの使用](#)
 - [パラレル実行でのパフォーマンスの最適化](#)
 - [パラレル実行の動作方法](#)
 - [並列度の設定](#)
 - [例: 並列度の設定](#)
 - [待機イベントについて](#)

11 データ・ウェアハウスのバックアップおよびリカバリ

- [データ・ウェアハウスのバックアップおよびリカバリの処理方法](#)
- [バックアップおよびリカバリの計画とベスト・プラクティス](#)
 - [ベスト・プラクティスA: ARCHIVELOGモードの使用](#)
 - [停止時間の許容について](#)
 - [ベスト・プラクティスB: Recovery Managerの使用](#)
 - [ベスト・プラクティスC: 読取り専用の表領域の使用](#)
 - [ベスト・プラクティスD: NOLOGGING操作の計画](#)
 - [抽出、変換およびロード](#)
 - [増分バックアップ](#)
 - [ベスト・プラクティスE: すべての表領域の重要度が同等でない場合](#)

12 データ・ウェアハウスのセキュリティ

- [データ・ウェアハウスのセキュリティについて](#)
 - [データ・ウェアハウスのセキュリティの必要性について](#)

- [データ・ウェアハウスのセキュリティのロールおよび権限の使用](#)
- [データ・ウェアハウスの仮想プライベート・データベースの使用](#)
 - [仮想プライベート・データベースの動作方法](#)
- [Oracle Label Securityの概要](#)
 - [Oracle Label Securityの動作方法](#)
 - [データ・ウェアハウスによりラベルを利用する方法](#)
- [データ・ウェアハウスのファイングレイン監査の概要](#)
- [データ・ウェアハウスの透過的データ暗号化の概要](#)

[索引](#)

図一覧

- [3-1 インポート・メタデータ・ウィザード](#)
- [3-2 SOURCEフラット・ファイル・モジュール](#)
- [4-1 PRODUCTSディメンションのスター・スキーマ実装](#)
- [4-2 SALESキューブの実装](#)
- [5-1 プロジェクト・ナビゲータの「マッピング」ノード](#)
- [5-2 マッピング・エディタに表示された演算子のソース](#)
- [5-3 マッピングで接続した演算子](#)
- [5-4 バインドされていないステージング表（属性なし）とソース表](#)
- [5-5 「作成とバインド」ダイアログ・ボックス](#)
- [5-6 「ターゲット・ロード順序」ダイアログ・ボックス](#)
- [5-7 演算子のプロパティ・インスペクタ](#)
- [5-8 演算子の同期化](#)
- [5-9 異なるワークスペース・オブジェクトからの同期化](#)
- [9-1 パラレル実行の監視](#)
- [9-2 I/Oの監視](#)
- [10-1 パーティション化の結果](#)

表一覧

- [2-1 スループット・パフォーマンスの変換](#)
- [4-1 PRODUCTSディメンションのレベルの詳細](#)
- [4-2 PRODUCTSディメンションに作成された制御行](#)
- [4-3 SALESキューブのディメンション](#)
- [5-1 ワークスペース・オブジェクトと同期化される演算子](#)

はじめに

ここでは、次の項目について説明します。

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

対象読者

このガイドは、Oracle Databaseを使用して日常の一般的なデータ・ウェアハウスのタスクを実行するユーザーを対象としています。コンピュータの基本的な知識があり、『Oracle Database 2日でデータベース管理者』を参照済であることが主な前提条件です。

このガイドは、特に次のようなユーザーを対象としています。

- データ・ウェアハウスの管理スキルを習得する必要があるOracle DBA
- データ・ウェアハウスの基礎知識はある程度あるが、Oracleを初めて使用するDBA

ドキュメントのアクセシビリティについて

オラクル社は、障害のあるお客様を含む、すべてのお客様にオラクル社の製品、サービスおよびサポート・ドキュメントをご利用いただけることを目標としています。オラクル社のドキュメントには、ユーザーが障害支援技術を使用して情報を利用できる機能が組み込まれています。HTML形式のドキュメントで用意されており、障害のあるお客様が簡単にアクセスできるようにマークアップされています。標準規格は改善されつつあります。オラクル社はドキュメントをすべてのお客様がご利用できるように、市場をリードする他の技術ベンダーと積極的に連携して技術的な問題に対応しています。オラクル社のアクセシビリティについての詳細情報は、Oracle Accessibility Programの次のWebサイト<http://www.oracle.com/accessibility/>を参照してください。

ドキュメント内のサンプル・コードのアクセシビリティについて

スクリーン・リーダーは、ドキュメント内のサンプル・コードを正確に読めない場合があります。コード表記規則では閉じ括弧のみを行に記述する必要があります。しかし、一部のスクリーン・リーダーは括弧のみの行を読まない場合があります。

外部Webサイトのドキュメントのアクセシビリティについて

このドキュメントにはオラクル社およびその関連会社が所有または管理しないWebサイトへのリンクが含まれている場合があります。オラクル社およびその関連会社は、それらのWebサイトのアクセシビリティに関しての評価や言及は行っておりません。

聴覚に障害があるお客様のOracleサポート・サービスへのアクセス

Oracleサポート・サービスに連絡するには、テレコミュニケーション・リレー・サービス (TRS) を使用してOracleサポート (+1-800-223-1711) までお電話ください。Oracleサポート・サービスの技術者が、Oracleサービス・リクエストのプロセスに従って、技術的な問題を処理し、お客様へのサポートを提供します。TRSの詳細は、<http://www.fcc.gov/cgb/consumerfacts/trs.html>を参照してください。電話番号の一覧は、<http://www.fcc.gov/cgb/dro/trsphonebk.html>を参照してください。

関連ドキュメント

詳細は、次のOracleリソースを参照してください。

- 『Oracle Database 2日でデータベース管理者』

- 『Oracle Databaseデータ・ウェアハウス・ガイド』
- 『Oracle Database管理者ガイド』
- 『Oracle Warehouse Builder Concepts』
- 『Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide』
- 『Oracle Warehouse Builder Sources and Targets Guide』

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。
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。

サポートおよびサービス

次の各項に、各サービスに接続するためのURLを記載します。

Oracleサポート・サービス

オラクル製品サポートの購入方法、およびOracleサポート・サービスへの連絡方法の詳細は、次のURLを参照してください。

<http://www.oracle.com/lang/jp/support/index.html>

製品マニュアル

製品のマニュアルは、次のURLにあります。

<http://www.oracle.com/technology/global/jp/documentation/index.html>

研修およびトレーニング

研修に関する情報とスケジュールは、次のURLで入手できます。

http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=3

その他の情報

オラクル製品やサービスに関するその他の情報については、次のURLから参照してください。

<http://www.oracle.com/lang/jp/index.html>

<http://www.oracle.com/technology/global/jp/index.html>

注意:

ドキュメント内に記載されているURLや参照ドキュメントには、Oracle Corporationが提供する英語の情報も含まれています。日本語版の情報については、前述のURLを参照してください。

第1部

データ・ウェアハウスの構築

ここでは、データ・ウェアハウスの構築方法について説明します。内容は次のとおりです。

- [第1章「データ・ウェアハウスの概要」](#)
- [第2章「データ・ウェアハウス・システムの設定」](#)
- [第3章「データ・ソースの識別およびメタデータのインポート」](#)
- [第4章「Oracle Warehouse Builderでのウェアハウスの定義」](#)

1 データ・ウェアハウスの概要

データ・ウェアハウスの管理、設計および実装を行うユーザーとして、組織内のOracleデータ・ウェアハウスの操作全体を監視し、効率的なパフォーマンスを維持します。

この章の内容は次のとおりです。

- [このガイドについて](#)
- [データ・ウェアハウスについて](#)
- [Oracleデータ・ウェアハウスの一般的なタスク](#)
- [データ・ウェアハウス管理用ツール](#)

このガイドについて

このガイドでは、データ・ウェアハウスを実装して管理するために必要な共通の日常のタスクを実行する方法を説明しています。このガイドの目標は、Oracle Databaseで使用可能なデータ・ウェアハウス・ソリューションの概要を示すことです。基本的なパフォーマンス監視タスクを実行する方法を含め、データ・ウェアハウスの稼働を続けるために必要な共通の管理タスクおよび設計タスクを行う方法を説明します。

このガイドで使用する主要なインタフェースは、Oracle Enterprise Manager (EM)、Oracle Warehouse Builder (OWB) およびSQL*Plusです。

参照

- 『Oracle Enterprise Manager管理』
- 『Oracle Warehouse Builder Concepts』
- 『SQL*Plusクイック・リファレンス』

このガイドを使用する前に

このガイドを使用する前に、次の準備を実行する必要があります。

- Oracle Databaseを管理するOracle Enterprise Manager (EM) の使用方法についての理解を深めます。『Oracle Database 2日でデータベース管理者』を参照してください。
- [「データ・ウェアハウス管理用ツール」](#)に示されている必要なツールを入手します。

このガイドの対象外

このガイドは、データ・ウェアハウスをOracle Databaseに実装するための完全なガイドではありません。このガイドの目的は、タスクがタスク指向方式で実行される理由と時期を説明することです。必要に応じて、現行のタスクを理解して完了するために必要な概念を説明します。

機能および使用上の詳細な手順に関する完全な概念の情報は、次の適切なドキュメントを参照してください。

- 『Oracle Databaseデータ・ウェアハウス・ガイド』

- 『Oracle Warehouse Builder Sources and Targets Guide』
- 管理タスクについては、『Oracle Database管理者ガイド』を参照してください。
- OLAPについては、『Oracle OLAP DML Reference』を参照してください。
- データ・マイニングについては、『Oracle Data Mining Concepts』を参照してください。

データ・ウェアハウスについて

データ・ウェアハウスは、問合せと分析のために設計されたリレーショナル・データベースまたは多次元データベースです。OLTPシステムのドメインであるトランザクション処理用には最適化されません。通常はデータ・ウェアハウスにより、複数のソースから導出された履歴データおよび分析データが統合されます。データ・ウェアハウスにより、トランザクション・ワークロードから分析ワークロードが分割され、組織が複数のソースからデータを統合できるようにします。

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・ウェアハウス内のデータは通常、OLTPアプリケーション、メインフレーム・アプリケーション、または外部データのプロバイダなどのような1つ以上のデータ・ソースから抽出、変換およびロード（ETL）の各プロセスを介してロードされます。

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。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。一般にデータ・ウェアハウスでは、ビジネス・インテリジェンス環境の基盤を提供します。

このガイドでは、スター・スキーマなど、リレーショナル実装について説明しています。

参照

多次元データ・ウェアハウスの詳細は、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください。

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。

- データの一部は、単純化およびパフォーマンスの向上のために非正規化されています。
- 大量の履歴データが使用されます。
- 問合せにより大量のデータを取得することがあります。
- 計画済問合せと非定型問合せの両方とも一般的です。
- データのロードは制御されます。

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。

Oracle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
- 新規リリースのレベルに合せたデータベースおよびソフトウェア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。

このガイドで説明するタスク

このガイドでは、次のタスクについて説明します。

1. データ・ウェアハウスとして使用するためのOracle Databaseを構成します。

[第2章「データ・ウェアハウス・システムの設定」](#)を参照してください。この項では、このガイド全体の演習で参照されるデモンストラーションにアクセスする方法についても説明されています。

2. データを統合する最初のステップを実行します。

[第3章「データ・ソースの識別およびメタデータのインポート」](#)の説明に従います。

3. ウェアハウス内のターゲット・オブジェクトの定義を開始します。

ターゲット・ウェアハウスの外部表、ディメンションおよびキューブを定義する方法については、[第4章「Oracle Warehouse Builderでのウェアハウスの定義」](#)を参照してください。

4. ターゲットにデータを抽出、変換およびロードする計画を定義します。

手順2で指定したソースからデータを抽出し、そのデータを変換し、次に手順3で設計したターゲットにデータをロードするETLロジックを定義する方法については、[第5章「ETLロジックの定義」](#)を参照してください。

5. ターゲット・スキーマへ配布し、ETLロジックを実行します。

マッピングからのコードを使用してターゲット・スキーマを準備する方法、およびその後そのコードを実行する方法については、[第6章「ターゲット・スキーマへの配布およびETLロジックの実行」](#)を参照してください。

6. 効率的なSQLを書きます。

[第7章「レポートおよび分析のSQL」](#)を参照し、説明されているタスクを完了します。この項では、効率的なSQLを書く方法について説明されています。

7. データ・ウェアハウスをリフレッシュします。

[第8章「データ・ウェアハウスのリフレッシュ」](#)を参照し、説明されているタスクを完了します。

8. 操作を最適化します。

[第9章「データ・ウェアハウスの操作の最適化」](#)を参照し、説明されているタスクを完了します。

9. パフォーマンス・ボトルネックを排除します。

[第10章「パフォーマンス・ボトルネックの排除」](#)を参照し、説明されているタスクを完了します。

10. データ・ウェアハウスのバックアップおよびリカバリの基本の一部を確認します。

データ・ウェアハウスをバックアップおよびリカバリする方法に関するいくつかの考慮事項については、[第11章「データ・ウェアハウスのバックアップおよびリカバリ」](#)を参照してください。

11. データ・ウェアハウスのセキュリティの基本の一部を確認します。

安全なデータ・ウェアハウスを作成する方法に関するいくつかの考慮事項については、[第12章「データ・ウェアハウスのセキュリティ」](#)を参照してください。

データ・ウェアハウス管理用ツール

このガイドの手順では、データ・ウェアハウスで目標を達成するための次の製品、ツールおよびユーティリティが示されます。これらが必要となる場合もあります。

- **Oracle Universal Installer**

Oracle Universal Installer (OUI) により、Oracleソフトウェアおよびオプションをインストールします。これによりDatabase Configuration Assistant (DBCA) が自動的に開始され、データベースがインストールされます。OUIおよびDBCAは、Oracle Databaseに含まれています。必要に応じて、『*Oracle Universal Installer and OPatch User's Guide for Windows and UNIX*』を参照してください。

- **Oracle Enterprise Manager**

データベースを管理する主要なツールはWebベースのインタフェースであるOracle Enterprise Managerです。Oracleソフトウェアのインストール、データベースの作成またはアップグレード、およびネットワークの構成後、データベースを管理するOracle Enterprise Managerを使用できます。さらに、Oracle Enterprise Managerではパフォーマンス・アドバイザのインタフェース、およびSQL*Loader、Recovery ManagerなどのOracleユーティリティのインタフェースを使用できます。このガイドの記載よりもさらに詳細な内容を確認する場合は、『*Oracle Enterprise Manager管理*』を参照してください。

- **Oracle Warehouse Builder**

データ・ウェアハウスの移入および保持のための主要な製品であるOracle Warehouse Builderによって、単一の製品内でのETL、データ品質管理およびメタデータ管理が提供されます。

OWBには、Oracle Databaseに格納される統合リポジトリが含まれています。OWBは、Oracle Databaseの機能を使用して、Oracle DatabaseターゲットへのロードおよびOracle Databaseターゲットのメンテナンスに適したコードを生成します。詳細および総合的な手順は、『*Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*』を参照してください。

- **Oracle Tuning Pack**

Oracle Tuning Packでは、データベース全体のチューニング処理を自動化する一連のテクノロジーが提供されるため、データベースの管理コストが大幅に削減され、パフォーマンスと信頼性が向上します。このガイドで使用されているOracle Tuning Packの主要な機能

は、SQLアクセスおよびSQLチューニング・アドバイザーです。『Oracle Databaseライセンス情報』および『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください。

注意

この項でリストされているOUIおよびOWBは、Oracle Databaseに含まれています。OWBの一部のデータ品質機能には、追加のライセンスが必要です。Oracle Tuning Packには、追加のライセンスが必要です。

2 データ・ウェアハウス・システムの設定

この項では、データ・ウェアハウス環境を最初に構成する方法を説明します。内容は次のとおりです。

- [データ・ウェアハウス・システムを設定する一般的な手順](#)
- [環境の準備](#)
- [データ・ウェアハウスのデータベースの設定](#)
- [Oracle Warehouse Builderへのアクセス](#)

データ・ウェアハウス・システムを設定する一般的な手順

この項の手順では、Oracle Databaseをデータ・ウェアハウスとして使用できるように構成する方法を説明します。その後、Oracle Warehouse Builderを構成します。Oracle Warehouse Builderは、Oracle Databaseを強化し、データ管理計画を立案するためのグラフィカル・ユーザー・インタフェースを備えています。

データ・ウェアハウス・システムをセットアップする手順

1. [「環境の準備」](#)の説明に従って、ハードウェアのサイジングおよび構成を行います。
2. Oracle Databaseソフトウェアをインストールします。インストール手順については、『*Oracle Database 2日*でデータベース管理者』または『*Oracle Databaseインストール・ガイド for Linux*』など、使用するプラットフォームのインストール・ガイドを参照してください。
3. [「データ・ウェアハウスのデータベースの設定」](#)の説明に従って、データベースをデータ・ウェアハウスとして使用できるように構成します。
4. Oracle Warehouse Builderソフトウェアにアクセスします。

[「Oracle Warehouse Builderへのアクセス」](#)の説明に従います。その後は、OWBを使用して一般的なデータ・ウェアハウスのタスクを完了する方法の学習に役立つデモンストレーションをインストールします。

環境の準備

データ・ウェアハウスのアーキテクチャの基本的な構成要素は、オンライン・トランザクション処理 (OLTP) システムの基本的な構成要素とほぼ同じです。ただし、データのサイズおよび量が大规模であるため、データ・ウェアハウスのハードウェア構成およびデータ・スループットの要件は固有のものとなります。データ・ウェアハウスのサイジングは、システムに必要なスループットが起点になります。サイジング時には、次の基準を1つ以上使用します。

- ピーク時に間合せによりアクセスされるデータ量および許容できるレスポンス時間。
- 一定期間内にロードされるデータ量。

一般的に、どの特定の時間でも必要とする最も高いスループットを見積もる必要があります。

ハードウェア・ベンダーによって適切なデータ・ウェアハウス・アプリケーション設定が推奨されるため、サイジングに役立ちます。詳細は、任意のハードウェア・ベンダーにご相談ください。

適切なハードウェア設定

データ・ウェアハウスのパフォーマンスを最大限にするには、適切なサイジングと安定したハードウェア構成が必要です。次の項では、適切な構成に重要なポイントを説明します。

- [必要なCPU数およびクロック・スピードについて](#)
- [必要なメモリーについて](#)
- [必要なディスク数について](#)
- [十分なI/O帯域幅を判断する方法について](#)

必要なCPU数およびクロック・スピードについて

中央演算装置（CPU）はデータ・ウェアハウスの計算機能に使用されます。データ・ウェアハウスの操作を実行するには、十分な処理能力を持つCPUが必要です。パラレル操作は同等のシリアル操作よりもCPUに負担がかかります。

必要とするCPU数のガイドラインとして見積もった最も高いスループットを使用します。大まかな見積りとして、次の公式を使用します。

$$\text{<number of CPUs>} = \text{<maximum throughput in MB/s>} / 200$$

この公式は、CPUが1秒間に約200MBまで耐えられることを前提としています。たとえば、システムは、1秒間に1200MBの最高スループットが必要な場合、 $\text{<number of CPUs>} = 1200 / 200 = 6$ CPUを必要とします。このシステムでは、1つのサーバーに6つのCPUを持つ設定を処理できます。2ノード・クラスタ・システムには両方のノードに3つのCPUを設定できます。

必要なメモリーについて

データ・ウェアハウスのメモリーは、大規模なソートなどのメモリーに負担をかける処理を行う際に特に重要になります。大部分の間合せは大量のデータにアクセスするため、データ・キャッシュへのアクセスはデータ・ウェアハウスにとっては重要ではありません。データ・ウェアハウスには、重要なOLTPアプリケーションと同じメモリー要件はありません。

CPU数は必要なメモリー容量のよいガイドラインになります。次の単純な公式を使用すると、選択したCPUで必要なメモリー容量を計算できます。

$$\text{<amount of memory in GB>} = 2 * \text{<number of CPUs>}$$

たとえば、6つのCPUを持つシステムが必要とするメモリー容量は $2 * 6 = 12$ GBになります。大部分の一般的なサーバーはこの要件を満たしています。

必要なディスク数について

データ・ウェアハウス環境での一般的な問題は、必要とする最大容量に基づいた記憶領域のサイジングです。記憶領域要件に排他的に基づいたサイジングは、スループットのボトルネックを作成する傾向があります。

必要なディスク・アレイの数を求めるために、必要とする最大スループットを使用します。また、ディスク・アレイが耐えられるスループットを知るために、記憶領域プロバイダの指定を使用します。記憶領域プロバイダはGb/秒で測定され、最初のスループットの見積りはMB/秒に基づいていることに注意してください。平均ディスク制御の最大スループットは1秒当たり2Gbで、耐えられるスループットに換算すると $(70\% * 2 \text{ Gbit/s}) / 8 = 180 \text{ MB/s}$ となります。

必要なディスク・アレイ数を決定するには、次の公式を使用します。

$$\text{<number of disk controllers>} = \text{<throughput in MB/s>} / \text{<individual controller throughput in MB/s>}$$

たとえば、1秒間に1200MBのシステムは、 $1200 / 180 = 7$ disk arraysとなり、少なくとも7つのディスク・アレイが必要となります。

必要なスループットを維持するために十分な物理ディスクがあるか確認してください。ディスクのスループット数をディスク・ベンダーに尋ね

てください。

十分なI/O帯域幅を判断する方法について

エンドツーエンドのI/Oシステムは、CPUおよびディスクを含んだコンポーネントで構成されます。バランスのよくとれたI/Oシステムには、I/Oシステムのすべてのコンポーネントに対して同等の帯域幅が必要です。これらのコンポーネントには次が含まれます。

- サーバーおよび記憶領域間のコネクタであるホスト・バス・アダプタ（HBA）。
- サーバー間およびストレージ・エリア・ネットワーク（SAN）またはネットワーク・アタッチ・ストレージ（NAS）間のスイッチ。
- ネットワーク接続用のイーサネット・アダプタ（GigE NICまたはInfiniband）。Oracle RAC環境では、I/Oスループットのシステムのサイジング時には含めないノード間のインターコネク用追加のプライベート・ポートが必要です。インターコネクは、ノード間のパラレル実行などのファクタを考慮に入れ、個々にサイジングする必要があります。
- 個別のコンポーネントを接続するワイヤ。

各コンポーネントにはバランスのよくとれたI/Oシステムにするために十分なI/O帯域幅が提供される必要があります。見積もった最初のスループットおよびベンダーによるハードウェアの指定は、必要な個別のコンポーネントの量を決定する基準になります。次の表にある変換を使用して、ベンダーによるビット単位で表された最大スループット数をバイト数で表された耐えられるスループット数に変換します。

表2-1 スループット・パフォーマンスの変換

コンポーネント	ビット	1秒当たりのバイト数
HBA	2Gビット	200MB
16ポート・スイッチ	8×2Gビット	1200MB
Fibre Channel	2Gビット	200MB
GigE NIC	1Gビット	80MB
Inf-2Gビット	2Gビット	160MB

十分なI/O帯域幅を確保するために十分なコンポーネントを持つには、ディスクでのデータのレイアウトが成功と失敗を分ける鍵となります。すべてのディスク・アレイに対して十分なスループットを持つシステムを設定し、問合せが取得するデータが1つのディスクに格納されている場合、必要なスループットを確保することができません。これは、1つのディスクがボトルネックとなるためです。このような状況を避けるためには、できるかぎり多くのディスク、理想はすべてのディスクへデータをストライプ化します。1MBに対して256KBのストライプのデータは、マルチブロック読取り操作および複数のディスクへのデータの振り分け間でのよいバランスを提供します。

データ・ウェアハウスのハードウェア構成の検証

Oracle Databaseのインストールの前に、ハードウェアおよびオペレーティング・システムのレベルで設定を検証する必要があります。オペレーティング・システムで必要なスループットおよびパフォーマンスが実行されない場合、Oracle Databaseはユーザー要件に合わせて実行できなくなります。スループットを検証する2つのツールはddユーティリティおよびOrionです。

ddユーティリティについて

UNIXまたはLinuxでオペレーティング・システムのスループットを検証する基本方法は、ddユーティリティを使用することです。ddユーティリティは、UNIXの一般的なプログラムであり、このユーティリティの主な目的は、生データの低レベル・コピーおよび変換です。ddは、「データセット定義（dataset definition）」の略語です。関連するオーバーヘッドがほとんどないため、ddユーティリティからの出力は信頼性の高い

測定です。Oracle Databaseは、ddユーティリティが実現する最大スループットの約90パーセントに到達できます。

例: ddユーティリティの使用

ddユーティリティを使用する手順

次は、ddの使用に最も重要なオプションです。

```
bs=BYTES: Read BYTES bytes at a time; use 1 MB
count=BLOCKS: copy only BLOCKS input blocks
if=FILE: read from FILE; set to your device
of=FILE: write to FILE; set to /dev/null to evaluate read performance;
write to disk would erase all existing data!!!
skip=BLOCKS: skip BLOCKS BYTES-sized blocks at start of input
```

Oracle Databaseが達成可能な最大スループットを見積もるため、典型的なデータ・ウェアハウス・アプリケーションのワークロード（大規模かつ、ランダムで連続したディスク・アクセスによる）を模倣します。

次のddコマンドは合計2GBを読み取る2つのデバイスに対してランダムで連続したディスク・アクセスを実行します。スループットは次のコマンドを終了するのに必要な時間によって分けられた2GBです。

```
dd bs=1048576 count=200 if=/raw/data_1 of=/dev/null &
dd bs=1048576 count=200 skip=200 if=/raw/data_1 of=/dev/null &
dd bs=1048576 count=200 skip=400 if=/raw/data_1 of=/dev/null &
dd bs=1048576 count=200 skip=600 if=/raw/data_1 of=/dev/null &
dd bs=1048576 count=200 skip=800 if=/raw/data_1 of=/dev/null &
dd bs=1048576 count=200 if=/raw/data_2 of=/dev/null &
dd bs=1048576 count=200 skip=200 if=/raw/data_2 of=/dev/null &
dd bs=1048576 count=200 skip=400 if=/raw/data_2 of=/dev/null &
dd bs=1048576 count=200 skip=600 if=/raw/data_2 of=/dev/null &
dd bs=1048576 count=200 skip=800 if=/raw/data_2 of=/dev/null &
```

検証にはデータベースの記憶領域に含む予定のすべての記憶領域装置を含める必要があります。クラスタ環境の構成時に各ノードからddコマンドを実行する必要があります。

Orionユーティリティについて

Orionは、スループットの測定を目的として、システムでOracleのようなワークロードを模倣するためにオラクル社が提供するツールです。ddユーティリティと比較すると、Orionには次の利点があります。

- Orionのシミュレーションはデータベースが生成するワークロードに似ています。
- Orionを使用すると、1つのシミュレーションで高い信頼性の書取りおよび読取りシミュレーションを実行できます。

オラクル社はデータベースがすでにインストールされている場合でも、最大の達成可能なスループットを検証するためにOrionの使用をお勧めします。

次はサポートされるI/Oワークロードのタイプです。

- 小規模およびランダム
- 大規模および連続的
- 大規模およびランダム
- 混合ワークロード

各タイプのワークロードでOrionは1秒当たりのMB、1秒当たりのI/OおよびI/O待機時間などのパフォーマンス・メトリクスを計測するためのI/O

ロードの異なるレベルで検証を実行できます。データ・ウェアハウス・ワークロードは複数のプロセスによって発行された連続するI/Oスループットによって一般的に特徴づけられています。構築するシステムのタイプに応じて異なるI/Oシミュレーションを実行できます。例は次のとおりです。

- ユーザーまたはアプリケーションがシステムに問い合わせる場合の毎日のワークロード
- ユーザーがシステムにアクセスするまたはしない場合のデータ・ロード
- 索引およびマテリアライズド・ビューの構築
- バックアップ操作

Orionソフトウェアのダウンロード時にブラウザで使用するURL

<http://www.oracle.com/technology/software/tech/orion/index.html>

例: Orionユーティリティの使用

Orionを起動する手順は、次のとおりです。

```
$ orion -run simple -testname mytest -num_disks 8
```

標準出力は次のとおりです。

```
Orion VERSION 10.2
```

```
Command line:
```

```
-run advanced -testname orion14 -matrixpoint -num_large 4 -size_large 1024  
-num_disks 4 -type seq -num_streamIO 8 -simulate raid0 -cache_size 0 -verbose
```

```
This maps to this test:
```

```
Test: orion14  
Small IO size: 8 KB  
Large IO size: 1024 KB  
IO Types: Small Random IOs, Large Sequential Streams  
Number of Concurrent IOs Per Stream: 8  
Force streams to separate disks: No  
Simulated Array Type: RAID 0  
Stripe Depth: 1024 KB  
Write: 0%  
Cache Size: 0 MB  
Duration for each Data Point: 60 seconds  
Small Columns: , 0  
Large Columns: , 4  
Total Data Points: 1
```

```
Name: /dev/vx/rdisk/asm_vol1_1500m Size: 1572864000  
Name: /dev/vx/rdisk/asm_vol2_1500m Size: 1573912576  
Name: /dev/vx/rdisk/asm_vol3_1500m Size: 1573912576  
Name: /dev/vx.rdisk/asm_vol4_1500m Size: 1573912576  
4 FILEs found.
```

```
Maximum Large MBPS=57.30 @ Small=0 and Large=4
```

この例では、特定のワークロードの最大スループットは1秒当たり57.30MBです。

データ・ウェアハウスのデータベースの設定

環境設定およびOracle Databaseソフトウェアのインストール後、データベース・パラメータが正しく設定されていることを確認します。設定する必要のあるデータベース・パラメータは多くありません。

一般的なガイドラインとして、特に理由がないかぎり、データベース・パラメータは変更しないでください。データ・ウェアハウスの設定に

は、Oracle Enterprise Managerを使用します。様々なパラメータ設定を表示するには、「データベース」ページに移動して「サーバー」をクリックします。「データベース構成」で「メモリー・パラメータ」または「すべての初期化パラメータ」をクリックします。

メモリー管理パラメータの設定方法

Oracle Databaseメモリーには、次のコンポーネントがあります。

- 共有メモリー: システム・グローバル領域 (SGA) とも呼ばれます。Oracleインスタンスで使用されるメモリーです。
- セッションベース・メモリー: プログラム・グローバル領域 (PGA) とも呼ばれます。データベース・セッションに使用されるメモリーです。ソートや集計などのデータベース操作の実行に使用されます。

Oracle Databaseは2つのメモリー領域のメモリー・コンポーネントの配布を自動的にチューニングします。次に示す相互に排他的な2つのオプションのいずれかを選択できます。

- MEMORY_TARGETおよびMEMORY_MAX_TARGETを設定します。

または

- SGA_TARGETおよびPGA_AGGREGATE_TARGETを設定します。

最初のオプションを選択した場合、他のパラメータを設定する必要はありません。データベースによって、すべてのメモリーが管理されます。2番目のオプションを選択した場合は、SGAとPGAのサイズを指定する必要があります。その他の設定は、データベースによって行われます。

PGA_AGGREGATE_TARGETパラメータはすべてのセッションの合計PGAが消費するターゲット・メモリー容量です。まずは、次の公式を使用してPGA_AGGREGATE_TARGET値を定義します。

- $PGA_AGGREGATE_TARGET = 3 * SGA_TARGET$ 。このPGA_AGGREGATE_TARGETをメモリーに適用するための十分な物理メモリーがない場合は、PGA_AGGREGATE_TARGETの値を小さくします。

- MEMORY_TARGETおよびMEMORY_MAX_TARGET

MEMORY_TARGETパラメータによってターゲットのメモリー・サイズを設定でき、関連した初期化パラメータMEMORY_MAX_TARGETによって最大ターゲット・メモリー容量を設定できます。データベースは、システム・グローバル領域 (SGA) および集計プログラム・グローバル領域 (PGA) 間の要件に従ってメモリーを再分配し、ターゲット・メモリー・サイズに対してチューニングします。ターゲット・メモリー初期化パラメータは動的であるため、データベースの再起動なしにいつでもターゲット・メモリー・サイズを変更できます。最大メモリー・サイズは上限として機能するため、ターゲット・メモリー・サイズを非常に高い値に設定してしまふことはありません。また特定のSGAコンポーネントが簡単に縮小できないこと、または最小サイズで保持される必要があるため、ターゲット・メモリー・サイズを過度に小さく設定してしまふことも防止します。

例: 初期化パラメータの設定

次で示すように、ALTER SYSTEMの発行により初期化パラメータの設定ができます。

```
ALTER SYSTEM SET SGA_TARGET = 1024M;
```

重要な他の初期化パラメータの設定

初めてデータ・ウェアハウスを使用する際は、Database Configuration Assistant (DBCA) の実行時に選択できるデータ・ウェアハウス・テンプレート・データベースが便利です。ただし、次の初期化パラメータを考慮すれば、どのデータベースでもかまいません。

- COMPATIBLE

COMPATIBLEパラメータは以前のリリースに対してデータベースが持つ互換性のレベルを識別します。最新機能を活用するには、COMPATIBLEパラメータでデータベースのリリース番号に設定します。

- OPTIMIZER_FEATURES_ENABLE

上位のコストベースのオプティマイザ機能を活用するには、このパラメータがデータベースの現行バージョンの値に設定されているか確認します。

- DB_BLOCK_SIZE

デフォルト値の8KBは、データ・ウェアハウスのほとんどの要件に適しています。表圧縮を使用する場合は、より大きなブロック・サイズを指定することを検討してください。

- DB_FILE_MULTIBLOCK_READ_COUNT

DB_FILE_MULTIBLOCK_READ_COUNTパラメータを使用すると、単一オペレーティング・システムの読取りコールでデータベース・ブロックの読取りができます。データ・ウェアハウスの一般的なワークロードは多くの連続したI/Oで構成されているため、多数の小さいI/Oではなく、少数の大きいI/Oを活用できるようにします。このパラメータの設定時、ブロック・サイズおよびオペレーティング・システムの最大I/Oサイズを考慮に入れて、次の公式を使用します。

$DB_FILE_MULTIBLOCK_READ_COUNT * DB_BLOCK_SIZE =$
<maximum operating system I/O size>

最大のオペレーティング・システムのI/Oサイズは64KBから1MBの間で変化します。

- PARALLEL_MAX_SERVERS

PARALLEL_MAX_SERVERSパラメータは、パラレル実行の使用可能な最大プロセス数のリソース制限を設定します。パラレル操作には、操作のあらゆる表に属性を付与する最大並列度 (DOP) として問合せサーバーのプロセス数が最大2倍必要です。

Oracle Databaseはほとんどのシステムにとって十分なデフォルト値をPARALLEL_MAX_SERVERSパラメータに設定します。PARALLEL_MAX_SERVERSのデフォルト値は次のとおりです。

$(CPU_COUNT * PARALLEL_THREADS_PER_CPU * (2 \text{ if } PGA_AGGREGATE_TARGET > 0; \text{ otherwise } 1) * 5)$

上位のDOP属性を持つ表のパラレル問合せにはこの値は十分ではない可能性があります。オラクル社は上位のDOPの問合せを実行するユーザーには、PARALLEL_MAX_SERVERSを次のように設定することをお勧めします。

$2 * DOP * \langle number_of_concurrent_users \rangle$

たとえば、PARALLEL_MAX_SERVERSパラメータに64を設定することで、各問合せは各セットの8つのDOPを持つ2つのスレーブ・セットを使用することを想定し、同時に4つのパラレル問合せを実行できます。

ハードウェア・システムがCPUバインドでもI/Oバインドでもない場合、問合せサーバー・プロセスを追加することによって、システムに同時パラレル実行ユーザーの数を増やすことができます。一方システムがCPUバインドまたはI/Oバインドになる場合、同時ユーザーの追加はすべてのパフォーマンスに弊害をもたらします。PARALLEL_MAX_SERVERSパラメータを慎重に設定することが、同時パラレル操作数の制限に効果的な方法です。

- PARALLEL_ADAPTIVE_MULTI_USER

TRUEまたはFALSEになるPARALLEL_ADAPTIVE_MULTI_USERパラメータは、現行のワークロードに応じて特定の文の並列度を動的に決定するためのアルゴリズムをサーバーが使用するかどうかを定義します。この機能を活用するには、PARALLEL_ADAPTIVE_MULTI_USERをTRUEに設定します。

- QUERY_REWRITE_ENABLED

マテリアライズド・ビューに対して問合せのリライトを活用するには、このパラメータをTRUEに設定する必要があります。このパラメータのデフォルトはTRUEです。

- QUERY_REWRITE_INTEGRITY

QUERY_REWRITE_INTEGRITYパラメータのデフォルトはENFORCEDです。データベースは、有効になっている検証された主キー制約、一意キー制約、および外部キー制約を基準とする場合、完全に最新のマテリアライズド・ビューに対してのみ問合せを上書きします。

TRUSTEDモードでは、オプティマイザはマテリアライズド・ビューのデータが現行であり、ディメンションで宣言された階層関係およびRELY制限が正確であるものとみなします。

- STAR_TRANSFORMATION_ENABLED

最適化されたスター型変換を活用するには、このパラメータをTRUEに設定します。

Oracle Warehouse Builderへのアクセス

Oracle Warehouse Builder (OWB) を使用すると、従来のデータ・ウェアハウスを含む様々なタイプのデータ管理計画を設定および配布できます。

OWBを有効にする手順

1. Oracle DatabaseのEnterprise EditionまたはStandard Editionのいずれかにアクセスしていることを確認します。

Oracle Database 11gには、OWBサーバー・コンポーネントが事前にインストールされています。また、OWBリポジトリのスキーマも含まれています。

2. Oracle DatabaseにインストールされているデフォルトのOWBスキーマを使用するには、まず、次のようにスキーマのロックを解除します。

SYSまたはSYSDBAユーザーとしてSQL*Plusに接続します。次のコマンドを入力します。

```
SQL> ALTER USER OWBSYS ACCOUNT UNLOCK;
```

```
SQL> ALTER USER OWBSYS IDENTIFIED BY owbsys_passwd;
```

3. Warehouse Builderのデザイン・センターを起動します。

Windowsの場合、「スタート」→「プログラム」→「Oracle」→「Warehouse Builder」を選択後、「デザイン・センター」を選択します。

UNIXおよびLinuxの場合、`owb_home/owb/bin/unix`に移動し、`owbclient.sh`を実行します。

4. ワークスペースを定義し、ユーザーをワークスペースに割り当てます。

単一のOWBリポジトリでは、関連付けられているオブジェクト上で作業している一連のユーザーに対応している各ワークスペースで、複数のワークスペースを定義できます。たとえば、開発、テストおよび本番などの各環境に対してワークスペースを作成できます。

簡略化するために、MY_WORKSPACEという1つのワークスペースを作成し、ユーザーを割り当てます。

「デザイン・センター」ダイアログ・ボックスで、「詳細の表示」をクリックした後、「ワークスペース管理」をクリックします。

「リポジトリ・アシスタント」が表示されます。

プロンプトに従い、「リポジトリ・アシスタント」でデフォルト設定を受け入れ、ワークスペースを作成し、ユーザーをワークスペース所有者として割り当てます。

- 作成したユーザー名およびパスワードで「デザイン・センター」にログインします。

参照

『Oracle Warehouse Builder Installation and Administration Guide for Windows and Linux』

Oracle Warehouse Builderデモンストレーションのインストール

この項では、複数のフラット・ファイル・ソースのデータの統合、データの変換および新規リレーショナル・ターゲットへのロード方法について、Oracle Warehouse BuilderのOBEの演習を使用して説明します（OBEはOracle By Exampleの略語です）。

演習および例は、Oracle Technology Network (OTN) (http://www.oracle.com/technology/obe/admin/owb_main.html) で入手できます。このデモンストレーションには、製品への理解を深めるために多様なOWBオブジェクトを作成するフラット・ファイル・データおよびスクリプトが用意されています。OBEのページでは、OWBの追加情報および演習に関する最新情報が提供されています。

このガイドに示されているOWB演習を実行する手順

- OTNデモンストレーションをダウンロードします。
 - OTNの次の場所から、OWBの例の場所にナビゲートします。
http://www.oracle.com/technology/obe/admin/owb_main.html
 - 最新リリースのOracle By Example (OBE) セットのリンクをクリックします。

デモンストレーションは、owbdemo_files.zipというzipアーカイブ内の一連のファイルで構成されています。

このzipアーカイブには、SQLスクリプト、値がカンマ区切りされる形式の2つのファイルおよびTclで記述されたスクリプトが含まれます。

- （オプション）同じリンクから、XSALES表データが含まれるxsales.zipをダウンロードします。
- スクリプトowbdemoinit.tclを編集します。

スクリプトowbdemoinit.tclによって、他のTCLスクリプトで使用される変数を定義および設定します。次の変数を編集して、コンピュータ環境の値と一致させます。

- set tempSPACE TEMP
- set owbclientpwd workspace_owner
- set sysuser sys
- set syspwd pwd
- set host hostname
- set port portnumber
- set service servicename
- set project owb_project_name

- o set owbclient workspace_owner
- o set sourcedir drive:/newowbdemo
- o set indexspace USERS
- o set dataspace USERS
- o set snapspace USERS
- o set sqlpath drive:/oracle/11.1.0/db_1/BIN
- o set sid servicename

4. Warehouse Builderスクリプト・ユーティリティおよびOMB PlusからTclスクリプトを実行します。

Windowsの場合、「スタート」→「プログラム」→「Oracle」→「Warehouse Builder」を選択後、「OMB*Plus」を選択します。

UNIXの場合、owb home/owb/bin/unixに移動し、OMBplus.shを実行します。

OMB+>プロンプトでは、次のコマンドを入力してスクリプトを含むディレクトリに移動します。

```
cd drive:¥¥newowbdemo¥¥
```

次のコマンドを入力して、すべてのTclスクリプトを必要な順序で実行します。

```
source loadall.tcl
```

5. デザイン・センターを起動し、スクリプトowbdemoinit.tclで指定した資格証明を使用して、ワークスペース所有者としてログインします。
6. デモンストレーションに従い、OWBクライアントが正常に設定されたことを確認します。

デザイン・センターでは、接続ナビゲータ内の右側にある「ロケーション」ノードを展開します。「データベース」の展開後、「Oracle」を展開します。「Oracle」ノードには次の場所が含まれます。

```
OWB_REPOSITORY
```

```
SALES_WH_LOCATION
```

OWBデモンストレーションを正常にインストールすると、デザイン・センターがEXPENSE_WHというOracleモジュールとともに表示されます。

3 データ・ソースの識別およびメタデータのインポート

この項では、Oracle Warehouse Builder (OWB) を使用してメタデータをインポートする方法を説明します。

この章の内容は次のとおりです。

- [データ・ソースの概要](#)
- [ソースからメタデータをインポートする一般的な手順](#)
- [OWBのワークスペース、プロジェクトおよびその他のデバイスについて](#)
- [例: フラット・ファイルからのメタデータのインポート](#)

データ・ソースの概要

一般的に、データ・ウェアハウスのソース・システムは通常、トランザクション処理のアプリケーションです。たとえば、売上分析のデータ・ウェアハウスは通常、現行のオーダー・アクティビティを記録しているオーダー・エントリ・システムからデータを抽出します。抽出プロセスの設計には問題が発生する場合があります。ソース・システムが複雑で文書化が不完全な場合、抽出するデータの決定は困難です。また、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も調整されません。これらの問題を処理するには、まずメタデータをインポートしてください。

メタデータはデータ・セットに与えられたオブジェクトのコンテンツを示すデータです。たとえば、表のメタデータは各列のデータ型を示します。

Oracle Databaseを使用する場合は、メタデータのインポートのためのツールとしてOracle Warehouse Builderをお勧めします。OWBにメタデータをインポートした後、メタデータに注釈を付けて、さらにトランザクション処理のアプリケーションから抽出計画を個別に設計できます。

ソースからメタデータをインポートする一般的な手順

メタデータをインポートする手順

1. [「Oracle Warehouse Builderへのアクセス」](#)の指示に従います。
2. [「Oracle Warehouse Builderデモンストレーションのインストール」](#)の説明に従ってOracle Warehouse Builderデモンストレーションをダウンロードおよびインストールします。
3. OWBのプロジェクトを指定します。

[「OWBのワークスペース、プロジェクトおよびその他のデバイスについて」](#)を参照してください。

4. [「例: フラット・ファイルからのメタデータのインポート」](#)に従います。

この例では、ソース・ファイルの場所を指定する方法、および「メタデータのインポート」ウィザードを起動する方法を説明します。ファイル、表およびビューなどのデータ・オブジェクトをインポートするプロセスは同じです。したがって、この例を完了すると、OWBにすべてのデータ・オブジェクトをインポートする一般的な方法を理解したことになります。

OWBのワークスペース、プロジェクトおよびその他のデバイスについて

OWBデモンストレーションをインストールして「デザイン・センター」を起動した後、ワークスペースにログインします。「デザイン・センター」の上部にユーザー名およびワークスペース名が表示されます。

ワークスペースには、関連するプロジェクトで作業する一連のユーザーが含まれます。作成するワークスペースの数を決定するためにセキュリティを考慮することは重要です。共通のモデルは、開発、テストおよび本番に個別のワークスペースを作成することです。このモデルを使用して、開発者などのユーザーによるワークスペースの開発およびテストへのアクセスを可能にしますが、本番ワークスペースへのアクセスは制限されます。

オプションで、ワークスペースをプロジェクトに分割できます。ただし実際は、ワークスペースには通常、アクティブなプロジェクトが1つのみ含まれます。これは、プロジェクトが単純に入れ物であり、セキュリティの実装またはサブジェクト指向のグループ化の確立には適さないためです。セキュリティはワークスペースを介して実装されます。後で説明するように、サブジェクト指向のグループ化の確立はモジュールを介して完了できます。

プロジェクトには、権限に関連するメタデータのセットが含まれます。したがってデータ・ウェアハウスでも、ソースおよびターゲットのすべてのメタデータを同一のプロジェクトに含めます。さらに権限の実装に必要なすべての機能、プロシージャ、変換、マッピングおよびその他のオブジェクトを含めます。プロジェクトには、OWBに作成またはインポートできるオブジェクトの各タイプのノードが含まれます。作成またはインポートできるオブジェクトのタイプの全般を理解するために異なるノードを展開します。

このデモンストレーションでは、プロジェクト・ナビゲータが左側に表示され、2つのプロジェクトが含まれます。MY_PROJECTはデフォルトで、事前にシードされたプロジェクトです。ワークスペース内での単一のアクティブなプロジェクトとしてMY_PROJECTを使用できます。このデモンストレーションでは、OWB_DEMOプロジェクトで作業します。

参照

http://www.oracle.com/technology/obe/11qrl_owb/index.htm

例：フラット・ファイルからのメタデータのインポート

この例では、フラット・ファイルからメタデータをインポートする方法を示します。具体的には、`export.csv`および`expense_categories.csv`の2つのファイルのメタデータをOWB_DEMOプロジェクトにインポートして、プロジェクト・ナビゲータの「ファイル」ノードの下に表示することが目的です。

フラット・ファイルからメタデータをインポートする手順

1. [「フラット・ファイルの場所の指定」](#)の説明に従って、フラット・ファイルの場所を指定します。
2. [「プロジェクトでのモジュールの作成」](#)の説明に従って、OWB_DEMOを編成し、入力フラット・ファイルのメタデータを受け取ります。
3. [「「インポート・メタデータ・ウィザード」の起動」](#)の説明に従って、インポートするファイルを指定します。
4. [「「フラット・ファイル・サンプル・ウィザード」の使用」](#)の説明に従って、メタデータの構成を指定します。
5. [「フラット・ファイル・データのインポート」](#)の説明に従って、両方のフラット・ファイルのメタデータをインポートします。

フラット・ファイルの場所の指定

フラット・ファイルがある場所を指定します。

「デザイン・センター」では、右側の部分が**接続ナビゲータ**というエクスプローラで、「**ロケーション**」というノードが含まれます。この「ロケーション」ノードを使用してソース・データがある場所を指定します。

OWBからアクセスできるソースおよびターゲットのタイプの全般を理解するために「**ロケーション**」ノードおよびその中にあるノードを開きます。

この例では、「**ファイル**」を右クリックし、「**新規**」を選択してフラット・ファイルの場所を定義します。

ファイル・システムの場所を作成するダイアログ・ボックスのプロンプトに従います。定義する各場所はコンピュータのファイル・システムの特定のディレクトリに対応します。したがって、ドライブおよびディレクトリに基づいた場所の名前を付けるように考慮します。このデモンスト

レーションでは、場所にC_NEWOWBDEMO_SOURCEFILESという名前を付けます。

プロジェクトでのモジュールの作成

プロジェクト・ナビゲータで、OWB_DEMOを編成し、入力フラット・ファイルのメタデータを受け取ります。

データ・ウェアハウスの実装では、多数のソース・オブジェクトおよびターゲット・オブジェクトを所有する可能性があります。これらの様々なオブジェクトを編成する方法として、OWBではモジュールを作成する必要があります。モジュールを使用すると、サブジェクト指向のグループ化を確立できます。さらに、各モジュールは、接続ナビゲータに作成する場所に対応します。

この例では、会社の売上データを含むモジュールを作成します。2つのフラット・ファイルに対して1つの場所のみ所有するため、プロジェクト・ナビゲータに1つのモジュールを作成します。OWB_DEMOの下の「ファイル」ノードを右クリックし、「新規」を選択します。新規のモジュールにSALES_EXPENSESという名前を付けます。この場所には、前のステップで定義した場所、C_NEWOWBDEMO_SOURCEFILESを指定します。

「インポート・メタデータ・ウィザード」の起動

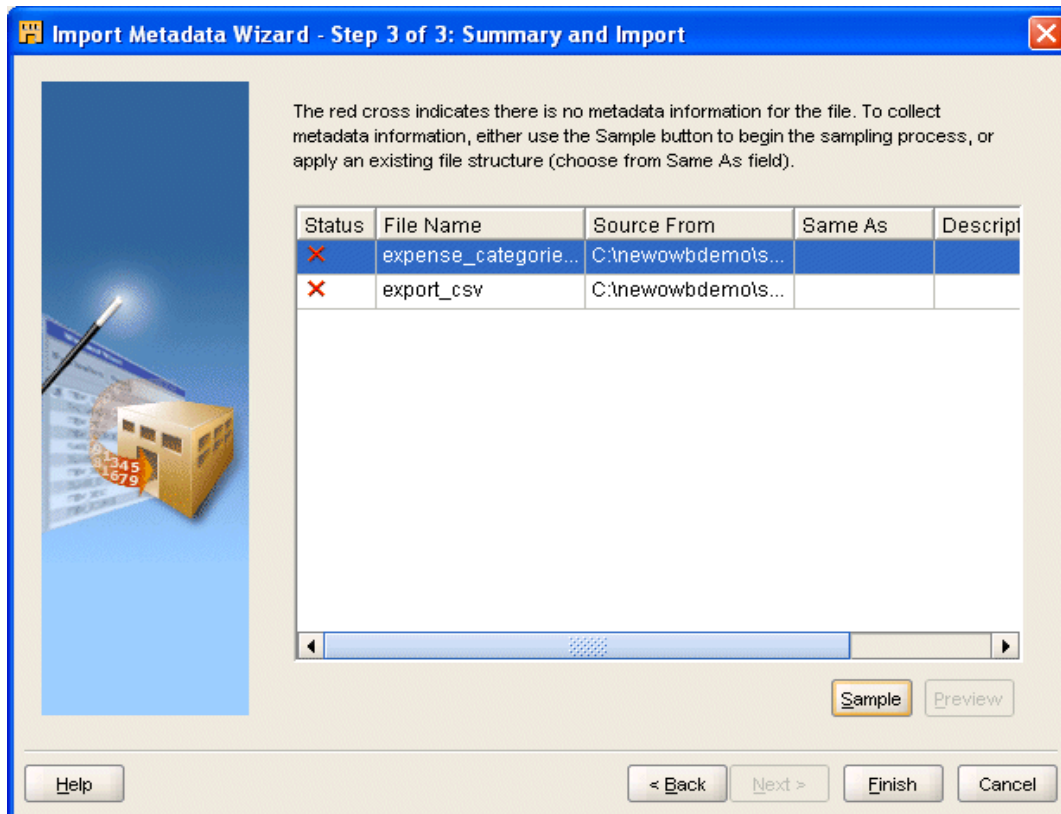
「インポート・メタデータ・ウィザード」を起動します。

モジュール「SALES_EXPENSES」を右クリックし、「新規」を選択して、「インポート・メタデータ・ウィザード」内のプロンプトに従います。ウィザード内のプロンプトは選択したモジュールのタイプによって異なり、したがってインポートするデータ・オブジェクトによっても異なります。

この例では、2つのフラット・ファイルのインポートを選択しました。「インポート・メタデータ・ウィザード」の「サマリー」ページで、ファイルの1つを選択し、次に「サンプル」を選択して「フラット・ファイル・サンプル・ウィザード」を起動します。

次のステップでは、順番に各ファイルのサンプルを作成し、このページの「終了」を選択してメタデータをインポートします。

図3-1 インポート・メタデータ・ウィザード



「図3-1 インポート・メタデータ・ウィザード」の説明

「フラット・ファイル・サンプル・ウィザード」の使用

「フラット・ファイル・サンプル・ウィザード」のプロンプトに従って、メタデータの構成を指定します。

サンプルに指定する文字の数に基づいて、ウィザードによりフラット・ファイル・データが読み込まれ、メタデータの構成のための提案が提供されます。サンプルのサイズが小さすぎる場合は、ウィザードによりデータが誤って読み込まれ、無効な提案が作成される可能性があります。それに従って、ウィザードの設定を変更および調整できます。

この例の目的のため、ウィザードによりファイルが区切られないことが正しく決定され、シングル・レコード・タイプが含まれます。またキャラクター・セットはWE8MSWIN1252です。「フラット・ファイル」ウィザードにあるすべてのデフォルト設定を受け入れます。

ウィザードによりサンプリングできる様々なタイプのファイルに精通するには、ウィザード・ページのオプションを確認してさらに知識を深めるために「ヘルプ」を選択します。

最初のフラット・ファイルをサンプリングした後、「インポート・メタデータ・ウィザード」のサマリーおよびインポートに関するページに戻り、2番目のファイルをサンプリングします。

前のファイルで行ったように、「フラット・ファイル」ウィザードのデフォルト設定を受け入れます。

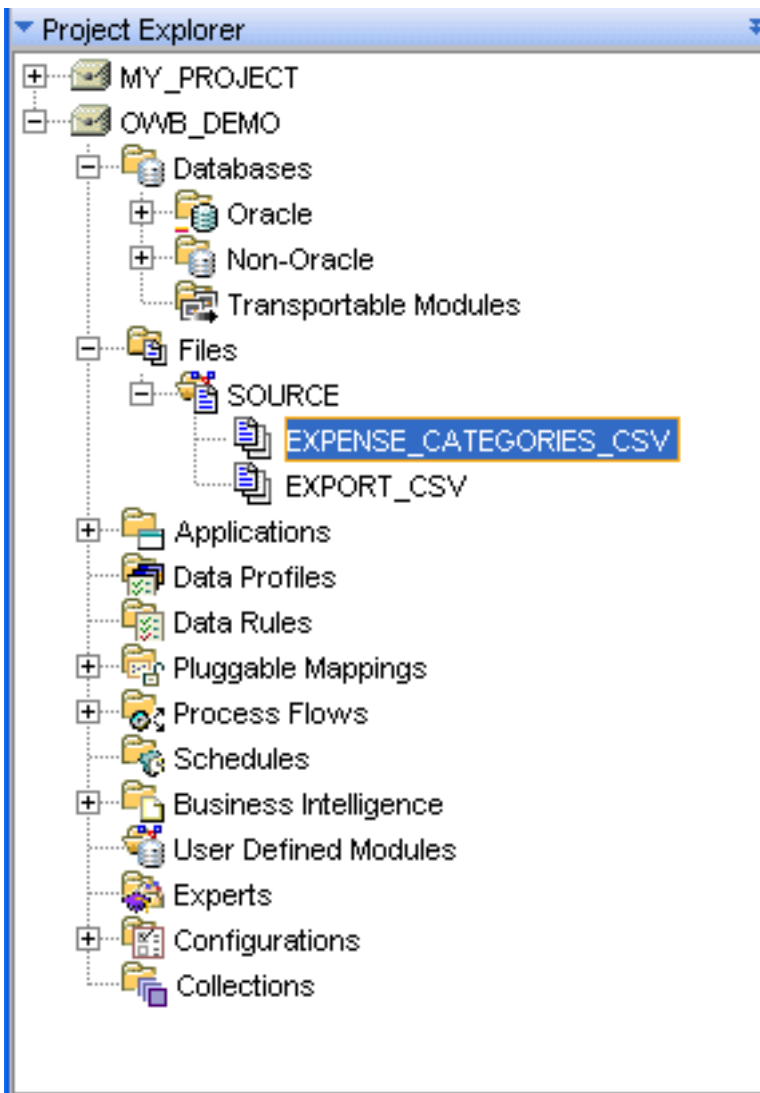
フラット・ファイル・データのインポート

両方のフラット・ファイルのメタデータをインポートします。

サマリーおよびインポートに関するページに再度戻り、「終了」を選択します。

「終了」を選択する場合、ウィザードにより、データのサンプリング時に作成した選択基準に基づいてデータがインポートされます。OWB_DEMOプロジェクトの「ファイル」ノードの下にあるSALES_EXPENSESモジュールの下に、2つのカンマ区切りのファイルが表示されます。

図3-2 SOURCEフラット・ファイル・モジュール



「図3-2 SOURCEプラットフォーム・ファイル・モジュール」の説明

4 Oracle Warehouse Builderでのウェアハウスの定義

Oracle Warehouse Builderを使用することにより、リレーショナルまたはディメンションのデータ・ウェアハウスを設計できます。

OWBでは、物理的な実装からディメンションの設計が明示的に区別されます。簡単なクリック操作で、ディメンション・オブジェクトに対してリレーショナル実装または多次元実装のいずれかを選択できます。したがって、リレーショナル・ターゲット・ウェアハウスまたは多次元ウェアハウスとして同じディメンション・オブジェクトを実装できます。

この項では、リレーショナル・ターゲット・ウェアハウスとして実装されるディメンション・モデルを設計する方法を示します。1つのキューブと2つのディメンションで構成される小規模のデータ・ウェアハウスをモデル化します。複合のスノーフレーク・スキーマをモデル化するためにOWBを使用できますが、このデモンストレーションの目的のために、2つのディメンションへの外部キー参照がある単一のキューブで構成される簡単なスター・スキーマをモデル化します。

この章の内容は次のとおりです。

- [リレーショナル・ターゲット・ウェアハウスを定義する一般的な手順](#)
- [ウェアハウス・ターゲット・スキーマの指定](#)
- [OWBのフラット・ファイルのソースについて](#)
- [ディメンションについて](#)

リレーショナル・ターゲット・ウェアハウスを定義する一般的な手順

この項では、リレーショナル・ターゲット・スキーマを定義する手順を示します。

リレーショナル・ターゲット・ウェアハウスを定義する手順

1. [「ウェアハウス・ターゲット・スキーマの指定」](#)の説明に従って、スキーマをウェアハウス・ターゲット・スキーマとして指定します。
2. ウェアハウス・ターゲット・モジュールにソース・オブジェクトおよびターゲット・オブジェクトを定義またはインポートします。

一般的に、ウェアハウス・ターゲット・モジュール内のいずれかのノードを右クリックし、「新規」または「インポート」のどちらかを選択します。OWBにより、説明のための適切なウィザードが起動されます。追加情報は、「ヘルプ」をクリックします。

ウェアハウス・ターゲット・モジュールに追加するオブジェクトのタイプは、ソース・データおよびデータ・ウェアハウスの目的によって異なります。

このガイドで示される演習を続行するには、[「演習: ターゲット・モジュールへの外部表の追加」](#)および[「演習: ディメンションの理解」](#)を参照してください。

3. 必要に応じて、ソース・オブジェクトおよびターゲット・オブジェクトを構成します。

いくつかのオブジェクトでは追加構成が必要です。ウェアハウス・モジュールに単一のオブジェクトをインポートまたは定義した後、右クリックして「構成」を選択し、設定を確認して必要に応じて変更します。

ウェアハウス・ターゲット・スキーマの指定

従来のデータ・ウェアハウスの実装内のターゲット・スキーマは通常1つのみで、これがデータ・ウェアハウスのターゲットになります。

データ・ウェアハウスのターゲット・スキーマとしてスキーマを指定する手順

1. OWBにスキーマを登録します。

「グローバル・ナビゲータ」パネルで、「セキュリティ」ノードを開きます。「ユーザー」ノードを右クリックし、「新規」を選択します。

「ユーザーの作成」ダイアログ・ボックスで、「DBユーザーの作成」を選択し、プロンプトに従います。さらに情報が必要な場合は、「ヘルプ」をクリックするか、または[F1]キーを押します。

このデモンストレーションでは、新規スキーマを作成し、これをEXPENSE_WHと呼びます。

2. 新規スキーマの位置情報を指定します。

接続ナビゲータ内で右クリックし、「Oracle」ノードの下の「ロケーション」から「新規」を選択します。

「EXPENSE_WH_LOCATION」という場所を作成します。接続をテストするためのオプションを選択します。

3. プロジェクト・ナビゲータで、単一のモジュールとスキーマの場所を関連付けます。

[「例: フラット・ファイルからのメタデータのインポート」](#)で、メタデータのインポート元の場所に対応するモジュールを作成しました。同様に、ターゲット・スキーマの場所に対応するモジュールを作成する必要があります。

OWB_DEMOプロジェクトでは、「データベース」ノードを開き、「Oracle」ノードを右クリックして「新規」を選択します。モジュールを作成するウィザードのプロンプトに従います。モジュールのステータスをWarehouse Targetとして指定することを確認します。

このデモンストレーションでは、モジュールにEXPENSE_WHという名前を付けます。

4. 新規のデータ・ウェアハウスのターゲット・スキーマを習熟します。

プロジェクト・ナビゲータで、新しく定義されたウェアハウス・ターゲット・モジュールのためのノードを開きます。モジュール内に定義またはインポートできるタイプのオブジェクトなど、様々なタイプのオブジェクトがノードの下にリストされます。

OWBのフラット・ファイルのソースについて

ターゲット・モジュールに追加するオブジェクトのタイプは、後で設計するETLロジックに影響を与える場合があります。データ・ソースがフラット・ファイルに由来する場合は、SQL*LoaderコードまたはSQLコードのどちらを生成するか選択できます。コードの各タイプには、固有の利点があります。簡潔に言うと、SQL*Loaderは大量のデータの処理により適している一方、SQLではより広範囲の複合結合および変換が可能です。

OWBでSQL*Loaderを使用するには、[「例: フラット・ファイルからのメタデータのインポート」](#)の説明に従ってフラット・ファイルをインポートします。ただし、SQLを使用するには、[「演習: ターゲット・モジュールへの外部表の追加」](#)の説明に従ってウェアハウス・モジュールに外部表を定義する必要があります。

外部表は、Oracle Database内のデータベース・オブジェクトです。

演習: ターゲット・モジュールへの外部表の追加

外部表は、リレーショナル・フォーマット内のフラット・ファイルからのデータを表し、OWBで通常のソース表のように動作する読取り専用の表です。作成する外部表は、既存のフラット・ファイル内のシングル・レコード・タイプに対応します。

この演習での目的は、前にインポートした2つのフラット・ファイルに必要な外部表を作成することです。2つのファイルの両方にシングル・レコード・タイプがあるので、各ファイルに対して外部表を1つのみ作成する必要があります。

ターゲット・ウェアハウス・モジュールに外部表を追加する手順

1. プロジェクト・ナビゲータで、「データベース」ノード、続いて「Oracle」ノードを開きます。
2. 外部表を作成するターゲット・モジュールを開きます。

つまり、EXPENSE_WHモジュールが開かれます。

3. 「外部表」ノードを右クリックして「新規」を選択します。

OWBで、外部表を作成するウィザードが表示されます。プロンプトに従います。

外部表にEXPENSE_CATEGORIESという名前を付けます。フラット・ファイルを選択するように求められた場合は、EXPENSE_CATEGORIES_CSVを選択します。

4. 前のステップを繰り返し、EXPORT_CSVを表すEXPENSE_DATAと呼ばれる外部表を作成します。
5. 2つの外部表に対して物理ファイル・システムの詳細を構成します。

モジュールから1つの外部表を右クリックし、「構成」を選択します。「データファイル」ノードを右クリックし、「作成」を選択します。デフォルト名、NEW_DATAFILE_1を受け入れます。外部表がデータを継承するフラット・ファイルの名前を入力します。したがってデータファイル名は、1つの外部表に対してはexpense_categories.csvとし、もう1つの外部表に対してはexport.csvとします。

ディメンションについて

ディメンションは、データを編成する構造です。一般的に使用されるディメンションの例には、顧客、時間および製品があります。

リレーショナル・ディメンションの場合は、ディメンションの使用によって、問合せパフォーマンスが向上します。これは、ユーザーが既知の階層をドリルダウンしてデータを分析することがよくあるためです。階層の例には、年、四半期、月、日という時間階層があります。Oracle Databaseでは、詳細表ではなくマテリアライズド・ビューからデータを取り出す問合せをリライトして、これらの定義済階層を使用します。

一般的なリレーショナル・ディメンション表には、次の特徴があります。

- ウェアハウス・キーと呼ばれる、値が移入された単一列の主キーがあります。

ウェアハウス・キーは、ディメンションを管理し、ディメンション履歴を保持する技術をサポートし、キューブのサイズを削減します。
- ディメンション・オブジェクトとして明示的に定義された1つ以上の階層があります。階層によって、Oracleサーバーによる問合せリライトの数が最大化されます。
- ディメンションは、スター・スキーマのデータのプライマリ組織単位です。一般的に使用されるディメンションの例は、カスタマ、「製品」および「時間」です。

ディメンションは、一連のレベルと、これらのレベルに定義されている一連の階層で構成されます。ディメンションを作成する際は、次の内容を定義する必要があります。

- **ディメンション属性:** ディメンション・メンバーの特徴を説明する特性です。ディメンション属性には、名前およびデータ型があります。
- **レベル:** データの集計のレベルを定義します。たとえば、PRODUCTSディメンションには、合計、グループおよび製品のレベルを設定できます。
- **レベル属性:** レベル・メンバーの特徴を説明する特性です。ディメンションの各レベルに、一連のレベル属性があります。

- **階層:** データの編成方法として、順序付けされたレベルまたは一連のデータ値（値ベース階層の場合）を使用する論理構造です。階層構造は一連のレベルの親子関係を説明します。

演習: デイメンションの理解

デイメンションの基本概念および設計を理解するには、この演習で事前定義のデイメンションを確認します。

デイメンションの理解を深めるための手順

1. データ・オブジェクト・エディタでPRODUCTSデイメンションを開きます。

「プロジェクト・ナビゲータ」パネルで、「OWB_DEMO」、「データベース」、「Oracle」、「SALES_WH」にナビゲートして、「デイメンション」を開きます。「PRODUCTS」をダブルクリックします。

OWBはデータ・オブジェクト・エディタを起動します。データ・オブジェクト・エディタは、様々なデータベースまたはデイメンション・オブジェクトの設計、作成および管理が簡単にできる単一インタフェースです。

2. デイメンション属性を監視します。

デイメンション属性は、デイメンション・メンバーの記述特性です。デイメンション属性には、名前とデータ型があり、デイメンション内の1つ以上のレベルに適用できます。これらは、データを格納するためのレベル属性として実装されます。

デイメンション属性のリストには、デイメンション内のあらゆるレベルに必要なすべての属性が含まれる必要があります。

たとえば、PRODUCTSデイメンションには、説明というデイメンション属性があります。この属性は、すべてのレベルの合計、グループおよび製品に適用でき、その中には、これらのレベルの各メンバーに対する説明が格納されます。

3. レベルを監視します。

デイメンションのレベルは、データの集計レベルを表します。デイメンションには、少なくとも1つのレベルが含まれている必要があります。ただし、値ベース階層が含まれるデイメンションは除きます。各レベルには、レベル属性およびレベル識別子が必要です。

たとえば、PRODUCTSデイメンションには、合計、グループおよび製品のレベルを設定できます。

レベルについて

各レベルには、サロゲート識別子とビジネス識別子の2つの識別子が必要です。デイメンションを作成するときは、そのデイメンションのサロゲート識別子およびビジネス識別子（複合ビジネス識別子の場合は属性）としてマーク付けされたデイメンション属性が、各レベルで実装される必要があります。

サロゲート識別子は、デイメンションのすべてのレベルにまたがって、各レベルのレコードを一意に識別します。この識別子は、単一の属性で構成する必要があります。サロゲート識別子を使用すると、ファクトを最下位デイメンション・レベルのみでなく、どのデイメンション・レベルにもフックできます。

リレーショナル実装を使用しているデイメンションの場合、サロゲート識別子のデータ型はNUMBERであることが必要です。サロゲート識別子の値は、すべてのデイメンション・レベルにまたがって一意である必要があるため、すべてのデイメンション・レベルのサロゲート識別子を同じ順序を使用して生成します。

リレーショナル実装の場合、サロゲート識別子は次の目的を果します。

- 子レベルが親レベルとは別の表に格納されている場合、各子レベルのレコードには、親レコードのサロゲート識別子が格納されます。
- ファクト表の各キューブ・レコードには、参照先のデイメンション・レコードのサロゲート識別子のみが格納されます。サロゲート識別子を格納することによって、キューブを実装するファクト表のサイズが削減されます。

ビジネス識別子はユーザーが選択する属性リストで構成されます。ビジネス識別子は、特定のレベルで一意であることが必要であり、常にデータ・ソースの自然キーから導出されます。ビジネス識別子はメンバーを一意に識別します。たとえば、製品レベルのビジネス識別子として統一商品コード (UPC) を使用できます。UPCは、各商品に対応する一意のコードです。

ビジネス識別子は、次のように機能します。

- ビジネス期間のレコードを識別します。
- ファクトとディメンション間または2つのレベル間の論理的なリンクを提供します。
- サロゲート・キーを参照可能にします。

ディメンションに子レベルを移入する場合は、その親レベルのビジネス識別子を指定する必要があります。キューブを移入する場合は、そのキューブが参照するディメンション・レベルのビジネス識別子を指定する必要があります。

親識別子は、値ベース階層の親参照に注釈を付けるために使用されます。

たとえば、値ベース階層のEMPLOYEEディメンションに、ID、FIRST_NAME、LAST_NAME、EMAIL、PHONE、JOB_ID、HIRE_DATEおよびMANAGER_IDのディメンション属性があるとします。このディメンションでは、IDがサロゲート識別子で、MANAGER_IDが親識別子です。

レベル属性の定義

レベル属性とはレベル・メンバーを説明する特性です。ディメンションの各レベルには一連のレベル属性があります。レベル属性を定義するには、そのレベルで実装するディメンション属性を単純に選択します。レベル属性には個別名とデータ型があります。データ型は、そのレベル属性に実装されているディメンション属性から継承されます。レベル属性の名前は、実装されているディメンション属性の名前とは異なる名前に変更できます。

すべてのレベルは、一連のディメンション属性のサロゲート識別子およびビジネス識別子としてマークされた属性を実装する必要があります。

階層の定義

ディメンション階層は、データの編成方法として、順序付けされたレベルまたは一連のデータ値 (値ベース階層の場合) を使用する論理構造です。階層構造は一連のレベルの親子関係を説明します。レベル・ベース階層には少なくとも1つのレベルが必要です。複数の階層の一部を1つのレベルにできます。

たとえば、TIMEディメンションには次の2つの階層が指定されます。

会計階層: 会計年度 > 会計年度四半期 > 会計年度月 > 会計年度週 > 日

カレンダー階層: カレンダー年 > カレンダー四半期 > カレンダー月 > 日

すべての階層は、完全な1対nの関係になります。親レベルの1つのレコードは子レベルの複数のレコードに対応します。一方、子レベルの1つのレコードは階層内で1つの親レコードにのみ対応します。

ディメンション・ロール

ディメンション・ロールとはディメンションの別名です。データ・ウェアハウスでは、キューブが同じディメンションを複数回参照する場合、参照する都度そのディメンションを格納する必要はありません。同じディメンションに対する複数回の参照は混乱の原因となります。そこで、ディメンションを参照するたびに別名を作成することで、結合を瞬時に理解できるようになります。このような場合は、同じディメンションがキューブ内の異なるディメンション・ロールを実行します。

たとえば、売上レコードには次の3つの時間値があります。

- 受注記帳時間
- 受注出荷時間

- 受注履行時間

3つのTIMEディメンションを作成し、それらのディメンションにデータを移入するかわりに、ディメンション・ロールを使用できます。1つのTIMEディメンションのモデルを作成し、そのディメンションに3つのロール（受注記帳時間、受注出荷時間、受注履行時間）を作成します。SALESキューブでは、受注時間、出荷時間および受注履行時間のディメンションを参照できます。

ディメンションがデータベースに格納される場合は、1つのディメンションのみが作成され、このディメンションを各ディメンション・ロールが参照します。一方、ディメンションがOLAPカタログに格納される場合は、OWBによって、各ディメンション・ロールに1つのディメンションが作成されます。したがって、TIMEディメンションに3つのロールがある場合、OLAPカタログには3つのディメンションが作成されます。ただし、3つのディメンションはすべて同一の基礎となる表にマッピングされます。これは、OLAPカタログがディメンション・ロールをサポートしていないための対策です。

注意

ディメンション・ロールは、リレーショナル実装を持つディメンションに対してのみ作成できます。

レベルの関係

レベルの関係とは、ディメンション階層内のレベル間のアソシエーションです。レベルの関係は、階層内の親レベルへの参照を格納するレベル属性を使用して実装されます。

たとえば、PRODUCTSディメンションに「合計 > グループ > 製品」という階層があるとします。OWBでは、2つのレベルの関係（製品からグループ、グループから合計）が作成されます。2つの新しい属性には、このレベルの関係（製品レベルでの関係とグループ・レベルでの関係）が実装されます。これらの属性には、親レベルのサロゲートIDが格納されます。

ディメンションの例

ディメンションの一例として、製品データの編成に使用するPRODUCTSディメンションを紹介します。[表4-1](#)に、PRODUCTSディメンションの各レベルと、ディメンション内の各レベルのサロゲート識別子とビジネス識別子をリストします。

表4-1 PRODUCTSディメンションのレベルの詳細

レベル	属性名	識別子
合計	ID	サロゲート
	名前	ビジネス
	説明	
グループ	ID	サロゲート
	名前	ビジネス
	説明	
製品	ID	サロゲート
	UPC	ビジネス

	名前	
	説明	
	パッケージ・タイプ	
	パッケージ・サイズ	

PRODUCTSディメンションの階層は、次のとおりです。

階層1: 合計 > グループ > 製品

制御行

OWBにより、ファクト・データを各レベルのディメンションにリンクできる制御行が作成されます。たとえば、TIMEディメンションを2つの異なるキューブで再利用して、予算データを月レベルで記録し、実際のデータを日レベルで記録する場合などです。ディメンションは制御行と一緒にロードされるため、追加定義をすることなくこのアクションを実行できます。ディメンション階層の各メンバーは、単一のレコードを使用して表示されます。

すべての制御行には-2から始まる負のディメンション・キー値があります。上位レベルのレベル値では行が生成され、ファクト表に対して一意でリンクする行として動作できます。このリンクの下位レベルまたは制御行は無効化されます。

[「ディメンションの例」](#)で説明されているPRODUCTSディメンションを検討します。4つの製品カテゴリを含む表からデータをこのディメンションにロードします。OWBは[表4-2](#)に示されるディメンションに制御行を挿入します。これらの行によりどのディメンション・レベルでもキューブにリンクできます。表にはすべてのディメンション属性値が含まれるわけではありません。

表4-2 PRODUCTSディメンションに作成された制御行

ディメンション・キー	合計名	カテゴリ名	製品名
-3	TOTAL		
-9	TOTAL	ハードウェア	
-10	TOTAL	ソフトウェア	
-11	TOTAL	エレクトロニクス	
-12	TOTAL	周辺機器	

ディメンションの実際の行数を取得するには、NULL行を除外するWHERE句を含めることで行数を数えます。たとえば、PRODUCTSでの数を取得するには、PRODUCTS内のNULL行を除外するWHERE句を含めて行数を数えます。

ディメンションの実装

ディメンションの実装とは、ディメンションおよびそのデータを物理的に格納する方法を指定することです。OWBによって、ディメンション・オブジェクトに対して複数のタイプの実装（多次元実装を含む）を行うことができます。ただし、このガイドでは、リレーショナル実装について

のみ説明します。

スター・スキーマ

スター・スキーマ実装では、OWBによって単一の表にディメンション・データが格納されます。同じ表またはビューに複数のディメンション・レベルのデータが格納されるため、表にディメンション・キーの列を指定する必要があります。ディメンション・キーの列は、ディメンションの主キーです。この列はキューブの外部キー参照も形成します。

各レベルには、ディメンション属性のサブセットが実装されます。デフォルトでは、レベル属性の名前はディメンション属性の名前と同じです。すべてのレベルのデータが同じ表に格納されることで生じる名前の衝突を回避するため、OWBでは、スター表の名前の指定に、次のガイドラインが使用されます。

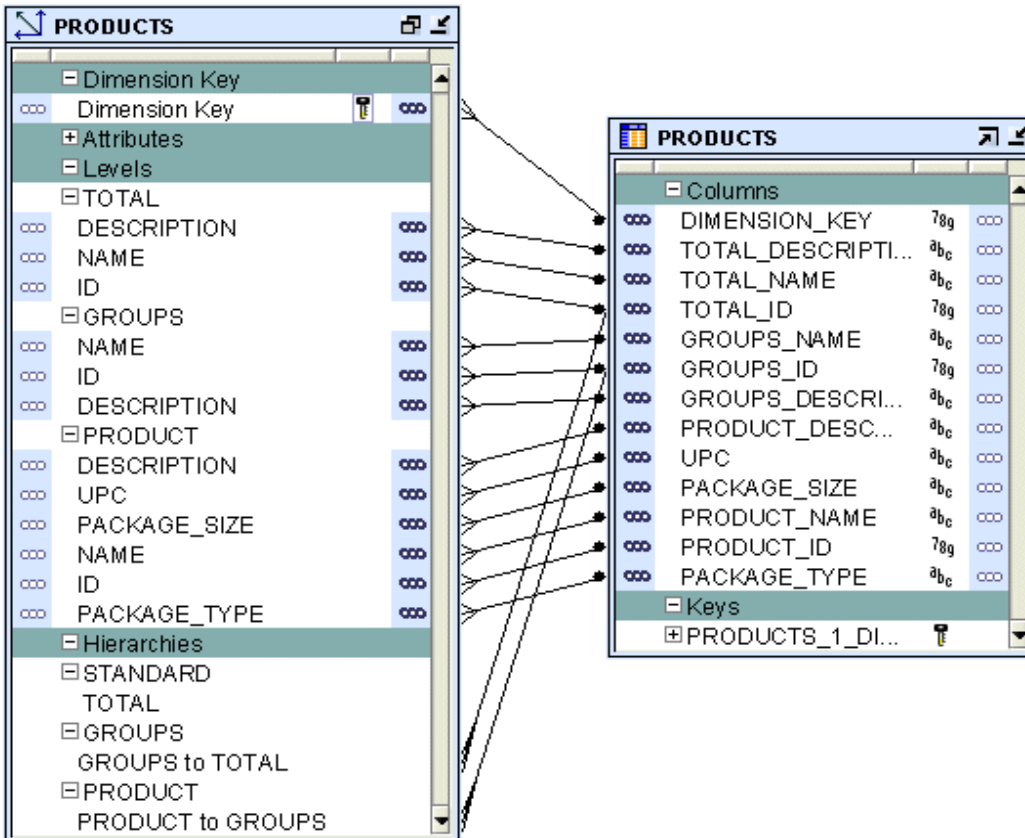
- レベル属性の名前が一意でない場合は、OWBによってレベル名に接頭辞が付けられます。
- レベル属性の名前が一意の場合は、接頭辞は使用されません。

注意

接頭辞が使用されないようにするには、「ディメンションの作成」ウィザードまたはデータ・オブジェクト・エディタで、レベル属性の名前を明示的に変更する必要があります。

たとえば、スター・スキーマを使用してPRODUCTSディメンションを実装する場合、OWBは単一の表を使用し、ディメンションにすべてのレベルを実装します。

図4-1 PRODUCTSディメンションのスター・スキーマ実装



「図4-1 PRODUCTSディメンションのスター・スキーマ実装」の説明

バインド

バインドを実行する場合は、各属性のデータおよびディメンション内のレベルの関係を格納するデータベース列を指定します。ディメンションに対して自動バインドまたは手動バインドを実行できます。

自動バインド:

自動バインドを実行すると、OWBによって、ディメンション・オブジェクトの属性がそのデータを格納するデータベース列にバインドされます。自動バインドを初めて実行するときは、ディメンション・データの格納に使用される表も作成されます。

すでにバインド済みのディメンションに対して自動バインドを実行すると、OWBでは次のルールが使用されます。

- ディメンションの実装方法が同じ場合は、OWBによって既存の実装オブジェクトにディメンション・オブジェクトが再バインドされます。

たとえば、スター・スキーマ実装方法を使用してPRODUCTSディメンションを作成し、自動バインドを実行したとします。ディメンション・データはProductsと呼ばれる表に格納されます。後日、ディメンション定義を変更しましたが、実装方法はスターのままです。この状況でPRODUCTSディメンションを自動バインドすると、OWBによって同じ実装表にPRODUCTSディメンションの属性が再バインドされます。

- ディメンションの実装方法を変更すると、OWBによって、古い実装オブジェクトが削除され、一連の新しい実装表が作成されます。古い実装オブジェクトを保持する場合は、自動バインドを実行する前に、ディメンション・オブジェクトをアンバインドする必要があります。実装方法の詳細は、[「スター・スキーマ」](#)を参照してください。

たとえば、スター・スキーマ実装方法を使用してPRODUCTSディメンションを作成し、それを実装表にバインドしたとします。その後、このディメンションを編集し、実装方法をスノーフレークに変更しました。変更したPRODUCTSディメンションの自動バインドを実行すると、OWBによって、ディメンション・データが格納されていた表は削除され、新しい実装表が作成されて、ディメンション属性および関係が新しい実装表にバインドされます。

自動バインドを実行する手順

1. プロジェクト・ナビゲータで、ディメンションを右クリックして「**エディタを開く**」を選択します。

このディメンションのデータ・オブジェクト・エディタが表示されます。

2. 「ディメンショナル」タブで、「ディメンション」ノードを右クリックして「**バインド**」を選択します。

あるいは、キャンバスで「ディメンション」ノードを選択し、「**オブジェクト**」メニューから「**バインド**」を選択します。

「バインド」オプションが有効になっていない場合は、ディメンションがリレーショナル・ディメンションであるかどうか、また「手動」オプションが「記憶域」タブの「実装」セクションで設定されていないかどうかをチェックします。

[「スター・スキーマ」](#)に説明されているように、自動バインドによって実装設定が使用されます。

手動バインド:

通常は、手動バインドを使用して既存の表をディメンションにバインドします。自動バインドまたは再バインドが不要な場合は、手動バインドを使用します。

ディメンションの手動バインドを実行する手順

1. ディメンション・データの格納に使用する実装オブジェクト（表またはビュー）を作成します。

リレーショナル・ディメンションの場合は、ディメンションのサロゲート識別子のロードに使用する順序を作成します。既存の順序を使用することもできます。

2. プロジェクト・ナビゲータで、ディメンションを右クリックして「**エディタを開く**」を選択します。

そのディメンションのデータ・オブジェクト・エディタが表示されます。キャンバスでは「ディメンショナル」タブがアクティブになります。

3. ディメンションを右クリックして「**詳細の表示**」を選択します。

OWBによって、ディメンションと同じ名前の新しいタブが表示されます。

4. パレットから、実装オブジェクトを表す演算子をキャンバスにドラッグ・アンド・ドロップします。

OWBによって、「新規または既存の<オブジェクト>を追加」ダイアログ・ボックスが表示されます。たとえば、ディメンション・データが表に格納されている場合は、パレットからキャンバスに表演算子をドラッグ・アンド・ドロップすると、「新規または既存の表を追加」ダイアログ・ボックスが表示されます。

5. 「**既存の<オブジェクト>を選択**」オプションを選択し、選択ツリーに表示されるオブジェクトのリストからデータ・オブジェクトを選択します。
6. 「**OK**」をクリックします。

追加したオブジェクトを表すノードがキャンバスに表示されます。

7. ディメンション・データの格納に複数のデータ・オブジェクトが使用される場合は、各データ・オブジェクトについてステップ4から6を実行します。
8. ディメンションの各レベルの属性を、そのデータを格納する列にマッピングします。ディメンション属性の上でマウスを押したまま、属性値を格納する列にドラッグ・アンド・ドロップします。

レベルの関係も、そのデータを格納するデータベース列にマッピングします。

たとえば、[「ディメンションの例」](#)に記載されているPRODUCTSディメンションの場合、PRODUCTSディメンションの**グループ・レベルの名前**属性は、Products_tab表のGroup_name属性に格納されます。**名前**属性の上でマウスを押したまま、Products_tab表のGroup_name属性にドラッグ・アンド・ドロップします。

キューブについて

キューブには、メジャーおよび1つ以上のディメンションへのリンクが含まれます。キューブの軸にはディメンション・メンバーが組み込まれ、キューブの本体にはメジャーの値が格納されます。ほとんどのメジャーは加法的なメジャーです。たとえば、売上データは、キューブ・エッジに時間、製品および顧客のディメンション値が指定され、キューブ本体にバリュー売上とドル売上上のメジャーから値が格納されるキューブに編成できます。

キューブは、外部キー制約によってディメンション表にリンクされます。これらの制約は、データ整合性が最も重要とされるデータ・ウェアハウス環境には不可欠です。この制約によって、データ・ウェアハウスを使用する日常業務での参照整合性が順守されます。

データ分析アプリケーションでは、通常、多くのディメンションにわたってデータが集計されます。これによって、データの変則的または特異なパターンを確認できます。キューブの使用は、この種の操作を実行する最も効率的な方法です。リレーショナル実装で、ウェアハウス・キーが設定されているディメンションを設計する場合、キューブの行の長さは一般的に短くなります。これは、ウェアハウス・キーの長さが対応する本来のデータより短いからです。その結果、キューブ・データに必要な記憶領域は少なくなります。

一般的なキューブには、次の要素が含まれます。

- 一連の外部キー参照列、またはデータ・リストの場合は人工キーまたは一連のウェアハウス・キー列に定義された主キー。キューブがデータ・リストの場合、外部キー参照列ではキューブの各行は一意に識別されません。
- 表を対応するディメンションにリンクする一連の外部キー参照列。

キューブの定義

キューブは、一連のディメンションにわたって定義されたメジャーの集合で構成されます。キューブを作成するには、次の要素を定義する必要があります。

- [キューブ・メジャー](#)

- [キューブのディメンション](#)

キューブ・メジャー

メジャーとは、調査および分析可能なデータで、通常は加法的な数値になります。メジャーの例としては、売上、原価および収益があります。キューブには、1つ以上のディメンションが必要です。メジャーの集計も可能です。ただし、集計できるのは数値メジャーのみです。

キューブのディメンション

キューブは一連のディメンションで定義されます。キューブはディメンション内で最下位レベル以外のレベルを参照できます。

純粋なリレーショナル実装を使用するキューブでは、ディメンション・ロールによって、複数回同じディメンションを再使用できます。ディメンション・ロールの詳細は、[「ディメンション・ロール」](#)を参照してください。

キューブを有効にする前に、キューブが参照するディメンションすべてが有効になっていることを確認してください。

ディメンション参照を定義する場合に指定する要素

- キューブが参照するディメンションとディメンション内のレベル。

リレーショナル実装を使用するキューブでは、ディメンションの中間レベルを参照できます。OWBでは、レベルの非サロゲート識別子（ビジネス・キーなど）への参照をサポートしています。

- リレーショナル実装を使用するディメンションの場合は、各ディメンションのディメンション・ロール。ディメンション・ロールは、キューブ内でディメンション参照が実行するロールを示します。ディメンション・ロールの指定はオプションです。

キューブの例

SALESキューブには、集計された売上データが格納されます。このキューブにはvalue_salesおよびDollar_salesという2つのメジャーが含まれています。

- value_sales: 売上額を売上数量に換算して格納します。
- Dollar_sales: 売上額を格納します。

[表4-3](#)に、SALESキューブのディメンションを示します。この表には、キューブが参照するディメンションの名前とディメンションのレベルがリストされています。

表4-3 SALESキューブのディメンション

ディメンション名	レベル名
Products	製品
Customers	顧客
Times	日

キューブの実装

キューブを実装する際は、キューブの物理的な記憶域の詳細を指定します。ディメンションの場合と同様に、OWBによって、リレーショナル形式または多次元形式でキューブを実装できます。このガイドでは、リレーショナル実装について説明します。

キューブのリレーショナル実装

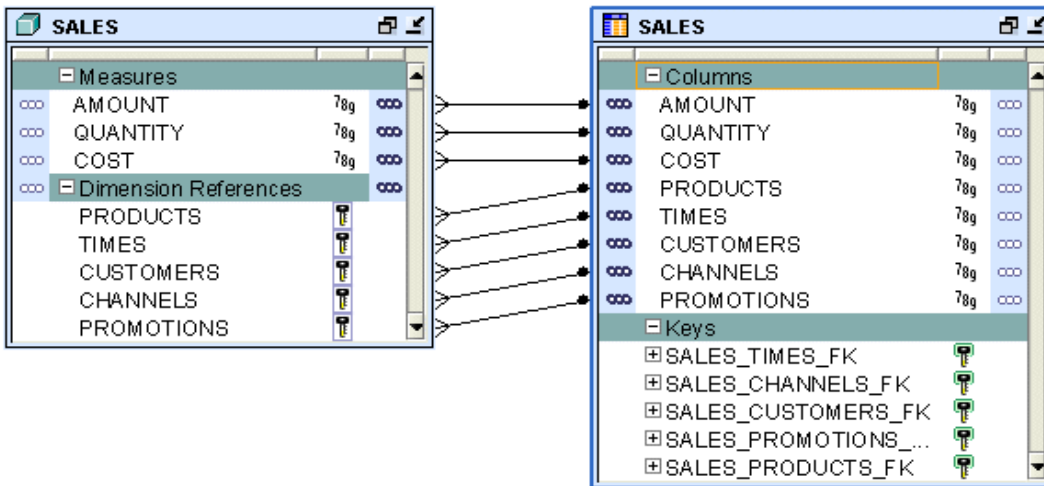
キューブ・データの格納に使用するデータベース・オブジェクトは、ファクト表と呼ばれます。キューブは1つのファクト表のみを使用して実装する必要があります。ファクト表にはキューブのメジャーおよびディメンション参照の列が含まれます。

キューブを実装する手順

- キューブ・データを格納する表またはマテリアライズド・ビューを選択します。
- 各メジャーについて、メジャー・データを格納する列を選択します。
- 各ディメンション参照について、ディメンション参照を格納する列を選択します。

各ディメンション参照は、ファクト表の列およびファクト表からディメンション表への外部キー（オプション）に対応します。ファクト表からディメンション表への1対nの関係が順守される必要があります。

図4-2 SALESキューブの実装



「図4-2 SALESキューブの実装」の説明

バインド

バインドを実行する場合は、キューブのメジャーとディメンション参照の各データを格納するデータベース列を指定します。キューブに対して自動バインドまたは手動バインドを実行できます。

自動バインド:

自動バインドを実行すると、OWBによって、キューブ・データを格納する表が作成され、キューブのメジャーおよび参照がデータベースの列にバインドされます。自動バインドを実行する手順の詳細は、[「自動バインド」](#)を参照してください。

キューブの自動バインドを実行する場合は、キューブが参照するディメンションを自動バインドした後に、キューブを自動バインドします。キューブが最後に自動バインドされた後に、そのキューブによって参照されるディメンションが自動バインドされた場合は、キューブを配布できません。

たとえば、TIMESディメンションとPRODUCTSディメンションを参照するSALESキューブを作成し、そのキューブの自動バインドを実行したとします。その後、PRODUCTSディメンションの定義を変更しました。ここで、再度SALESキューブを自動バインドしようとすると、OWBによってエラーが生成されます。最初にPRODUCTSディメンションを自動バインドした後に、キューブを自動バインドする必要があります。

手動バインド:

手動バインドでは、最初にキューブ・データを格納する表またはビューを作成し、次にそのデータを格納するデータベースの列にキューブの参

照およびメジャーをマッピングする必要があります。あるいは、既存のデータベース表またはビューを使用して、キューブ・データを格納することもできます。

キューブの手動バインドを実行する手順

1. キューブ・データを格納する表またはビューを作成します。

2. プロジェクト・ナビゲータで、キューブを右クリックして「**エディタを開く**」を選択します。

そのキューブのデータ・オブジェクト・エディタが開きます。キャンバスでは「**ディメンショナル**」タブがアクティブになります。

3. キューブを右クリックして「**詳細の表示**」を選択します。

OWBによって、キューブと同じ名前の新しいタブが表示されます。

4. パレットから、実装オブジェクトを表す演算子をキャンバスにドラッグ・アンド・ドロップします。

OWBによって、「新規または既存の<オブジェクト>を追加」ダイアログ・ボックスが表示されます。たとえば、キューブ・データが表に格納されている場合は、パレットからキャンバスに演算子をドラッグ・アンド・ドロップすると、「新規または既存の表を追加」ダイアログ・ボックスが表示されます。

5. 「**既存の<オブジェクト>を選択**」オプションを選択し、選択ツリーに表示されるオブジェクトのリストからデータ・オブジェクトを選択します。

6. 「**OK**」をクリックします。

追加したオブジェクトを表すノードがキャンバスに表示されます。

7. キューブのメジャーとディメンション参照をキューブ・データが格納される列にマッピングします。メジャーまたはディメンション参照をマウス・ボタンで押したまま、そのメジャーまたはディメンション参照を格納するデータ・オブジェクト属性にドラッグ・アンド・ドロップします。

第II部

データ・ウェアハウスへのデータのロード

ここでは、データ・ウェアハウスへのデータのロード方法について説明します。内容は次のとおりです。

- [第5章「ETLロジックの定義」](#)
- [第6章「ターゲット・スキーマへの配布およびETLロジックの実行」](#)

5 ETLロジックの定義

Oracle Warehouse Builderでデータ・オブジェクトの定義を作成およびインポートした後は、データをソースからターゲットに移動する抽出、変換およびロード（ETL）の各操作を設計できます。OWBでは、マッピングを使用してこれらの操作を設計します。

この章の内容は次のとおりです。

- [マッピングおよび演算子について](#)
- [マッピングを定義する手順の要約](#)
- [マッピングの作成](#)
- [演算子の追加](#)
- [演算子の編集](#)
- [演算子の接続](#)
- [マッピング・プロパティの設定](#)
- [演算子、グループおよび属性プロパティの設定](#)
- [演算子とワークスペース・オブジェクトの同期化](#)

マッピングおよび演算子について

マッピングとはデータをソースから抽出し、変換してターゲットにロードする一連の操作を表します。これによりデータ・フローおよびデータで実行される操作がビジュアル表示されます。マッピングをOWBで設計する場合は、マッピング・エディタ・インタフェースを使用します。

マッピングの基本となる設計要素は演算子です。演算子を使用して、データ・フローでソースおよびターゲットを表現します。また、演算子を使用して、ソースからターゲットへのデータの変換方法を定義することもできます。ソースとして選択する演算子は、マッピングの設計方法に影響を与えます。OWBでは、選択された演算子に基づいて、次のいずれかのマッピング生成言語にマッピングが割り当てられます。

- PL/SQL
- SQL*Loader
- ABAP

各コード言語では、マッピングを設計する際、特定のルールに従う必要があります。

このガイドでは、PL/SQLマッピングを定義する方法を説明します。他のタイプのマッピングを定義するには、『Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide』を参照してください。PL/SQLマッピングを定義する基本ルールは、PL/SQLマッピングには、フラット・ファイル演算子またはSAP/R3ソース以外のすべてのタイプのソース演算子を含めることができます。

マッピングを定義する手順の要約

マッピングを定義する場合に参照する項

1. [マッピングの作成](#)
2. [演算子の追加](#)
3. [演算子の編集](#)
4. [演算子の接続](#)
5. [マッピング・プロパティの設定](#)
6. [演算子、グループおよび属性プロパティの設定](#)
7. *Warehouse Builder*のオンライン・ヘルプのマッピング構成のリファレンス
8. マッピング・デザインに問題がない場合、ツールバーの「生成」アイコンを選択してコードを生成します。

後続の手順

マッピングを設計してマッピングのコードを生成した後、プロセス・フローを作成するか、直接配布を続行して実行できます。

プロセス・フローを使用して、マッピングを相互に関連付けます。たとえば、あるマッピングが完了すると電子メール通知がトリガーされて別のマッピングが起動するように、プロセス・フローを設計できます。

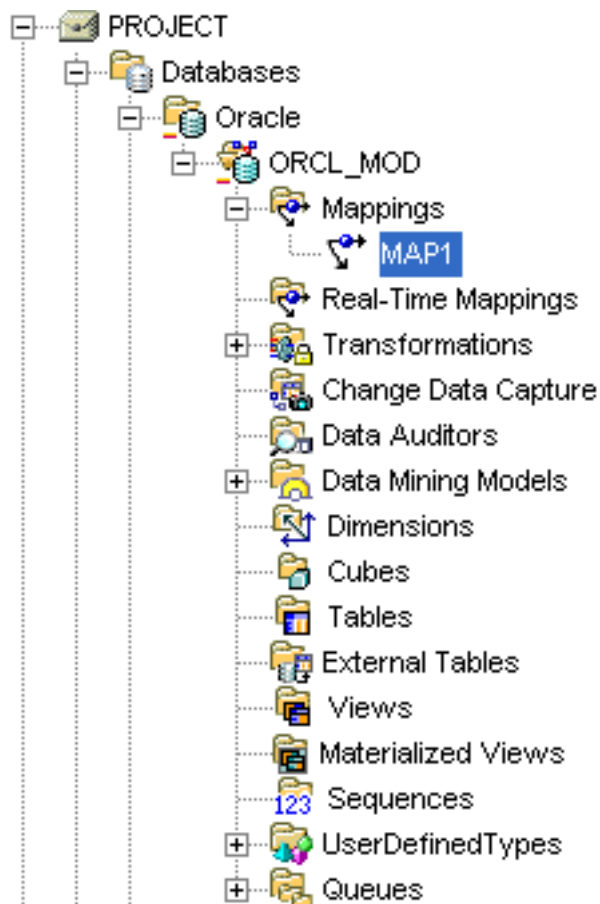
マッピングおよび作成した関連するプロセス・フローを配布し、マッピングを実行します。

マッピングの作成

マッピングを作成する手順

1. プロジェクト・ナビゲータの「マッピング」ノードにナビゲートします。このノードは、ウェアハウス・ターゲット・モジュールの下、「データベース」フォルダの下の「Oracle」フォルダにあります。

図5-1 プロジェクト・ナビゲータの「マッピング」ノード



「図5-1 プロジェクト・ナビゲータの「マッピング」ノード」の説明

2. 「マッピング」を右クリックして「新規」を選択します。

「マッピングの作成」ダイアログ・ボックスが表示されます。

3. 新しいマッピングの名前と説明（オプション）を入力します。

マッピング名の指定と説明の記述に関するルールを表示するには、「ヘルプ」を選択します。

4. 「OK」をクリックします。

マッピングの定義が保存され、その名前がプロジェクト・ナビゲータ内に挿入されます。また、そのマッピングのマッピング・エディタが表示され、タイトル・バーにマッピングの名前が表示されます。

以前に作成したマッピングを開く手順

1. プロジェクト・ナビゲータで、ウェアハウス・ターゲット・モジュールを「データベース」フォルダに移動して、次にOracle Databaseのフォルダに移動します。
2. 「マッピング」ノードを開きます。
3. 次のいずれかの方法で、マッピング・エディタを開きます。
 - マッピングをダブルクリックします。
 - マッピングを選択し、「編集」メニューから「エディタを開く」を選択します。
 - マッピングを選択し、[Ctrl]キーを押しながら[0]キーを押します。

- 。 マッピングを右クリックし、「**エディタを開く**」を選択します。

OWBによって、マッピング・エディタが表示されます。

演算子のタイプ

マッピングを設計するときは、マッピング・エディタ・パレットから演算子を選択し、キャンバスにドラッグできます。

- **Oracleソース/ターゲット演算子:** これらの演算子を使用して、Oracle表、ビュー、マテリアライズド・ビューなどのOracle Databaseオブジェクトを示します。
- **リモートおよびOracle以外のソース演算子とターゲット演算子:** これらの演算子の使用には、特別な要件があります。
- **データ・フロー演算子:** データ・フロー演算子を使用して、データを変換します。
- **事前/事後処理演算子:** マッピングの実行前または後に処理を実行するには、事前/事後処理演算子を使用します。マッピングに対する値を指定するには、マッピング・パラメータ演算子を使用します。
- **プラグgable・マッピング演算子:** プラuggable・マッピング演算子は、マッピング演算子の再利用可能なグループで、単一の演算子として機能します。

演算子の追加

演算子をマッピングに追加する手順は、選択する演算子のタイプによって異なります。これは、一部の演算子はワークスペース・オブジェクトにバインドされ、その他の演算子はバインドされないためです。一般的に、データ・ソースまたはターゲット演算子を追加すると、OWBによって、OWBワークスペースのそのオブジェクトのバージョンと、マッピング・エディタ用の個別のバージョンが維持されます。たとえば、表演算子をマッピングに追加した場合は、OWBによってその表のコピーがワークスペースで維持されます。個々のバージョンは、1つにバインドされません。つまり、マッピングのバージョンがワークスペースのバージョンにバインドされます。

2つのバージョンを区別するために、この項では、ワークスペース内のオブジェクトを総称的にワークスペース・オブジェクトと呼び、具体的には、ワークスペース表、ワークスペース・ビューのように表現します。また、マッピングの演算子は表演算子、ビュー演算子のように表現します。したがって、ディメンションをマッピングに追加する場合、マッピングのディメンションはディメンション演算子、ワークスペースのディメンションはワークスペース・ディメンションと表されます。

OWBでは、ユーザーがこれらのオブジェクトの変更定義を同期化できるように、一部の演算子に個別のワークスペース・オブジェクトが保持されます。たとえば、ワークスペース表の新規メタデータ定義を再インポートした場合は、これらの変更内容をマッピングの表演算子に伝播する必要があります。反対に、マッピングの表演算子を変更した場合は、これらの変更内容を関連するワークスペース表に伝播する必要があります。これらのタスクは、同期化と呼ばれるプロセスによって実現します。OWBでは、自動的に同期化またはマッピング・エディタから手動で同期化できます。

演算子をマッピングに追加する手順

1. マッピング・エディタを開きます。
2. 「**マッピング**」メニューから「**追加**」を選択し、演算子を選択します。または、ツールボックスからマッピング・エディタのキャンバスに演算子のアイコンをドラッグ・アンド・ドロップします。

ワークスペース・オブジェクトにバインドできる演算子を選択すると、マッピング・エディタに「**マッピング<演算子名>の追加**」ダイアログ・ボックスが表示されます。このダイアログ・ボックスの使用の詳細は、「**ヘルプ**」を選択してください。

ワークスペース・オブジェクトにバインドできない演算子を選択した場合は、演算子を作成するためのウィザードまたはダイアログ・ボックスが表示されます。

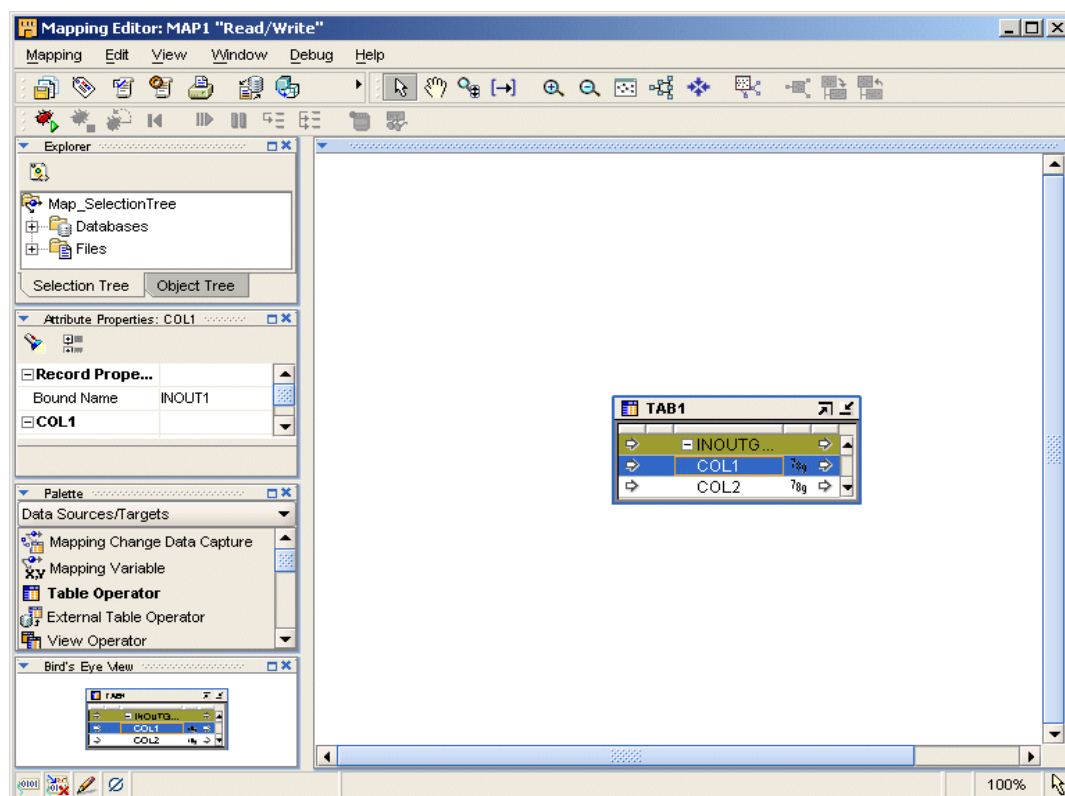
3. OWBに表示される指示に従い、「**OK**」をクリックします。

マッピング・エディタでは、キャンバスに最大化された演算子が表示されます。左上の隅に演算子名が表示されます。各属性名および

データ型を表示できます。

演算子を最小化するには、右上隅の矢印をクリックします。マッピング・エディタのキャンバスに演算子がアイコンとして表示されず。

図5-2 マッピング・エディタに表示された表演算子のソース



「図5-2 マッピング・エディタに表示された表演算子のソース」の説明

ワークスペース・オブジェクトにバインドする演算子の追加

ワークスペース・オブジェクトにバインドできる演算子を追加すると、マッピング・エディタに「マッピング<演算子名>の追加」ダイアログ・ボックスが表示されます。次のオプションから1つ選択します。

- [バインドされていない演算子を属性なしで作成](#)
- [既存のワークスペース・オブジェクトから選択してバインド](#)

バインドされていない演算子を属性なしで作成

このオプションは、マッピング・エディタを使用して、新規ステージング領域表や新規ターゲット表など、新しいワークスペース・オブジェクトを定義する場合に使用します。

「バインドされていない演算子を属性なしで作成」を選択した後、新しいオブジェクトの名前を入力します。演算子が属性なしでキャンバスに表示されます。

これで、演算子の属性を追加および定義できるようになりました（「[演算子の編集](#)」を参照）。次に、ターゲット・モジュールに新規ワークスペース・オブジェクトを作成するために、演算子を右クリックして「作成とバインド」を選択します。

このオプションをマッピング設計で使用する方法的例は、「[例：マッピング・エディタによるステージング領域表の作成](#)」を参照してください。

既存のワークスペース・オブジェクトから選択してバインド

このオプションは、ワークスペース内の定義済またはインポート済のオブジェクトに基づいて、演算子を追加する場合に使用します。

接頭辞を入力してオブジェクトを検索するか、選択したモジュール内のオブジェクトの表示リストからオブジェクトを選択します。

複数のアイテムを選択するには、[Ctrl]キーを押しながら各アイテムをクリックします。連続したアイテムのグループを選択するには、選択範囲の最初のオブジェクトをクリックし、[Shift]キーを押しながら最後のオブジェクトをクリックします。

演算子は、マッピングと同じモジュール内のワークスペース・オブジェクトまたは別のモジュールのワークスペース・オブジェクトに基づいて追加できます。別のモジュールのワークスペース・オブジェクトを選択すると、マッピング・エディタによってコネクタが作成されます（コネクタが存在していない場合）。このコネクタによって、マッピングのロケーションとワークスペース・オブジェクトのロケーション間のデータの移動パスが確立されます。

演算子の編集

各演算子には、エディタが関連付けられています。その演算子エディタを使用して、演算子、グループおよび属性に関する一般的な情報と構造的な情報を指定します。演算子エディタでは、グループおよび属性を追加または削除したり、名前を変更できます。演算子の名前も変更できません。

演算子の編集は、ロード・プロパティや条件付き動作の割当てとは異なります。ロード・プロパティや条件付き動作を指定するには、プロパティ・ウィンドウ（[「演算子、グループおよび属性プロパティの設定」](#)を参照）を使用します。

演算子、グループまたは属性を編集する手順

1. マッピング・エディタのキャンバスから演算子を選択します。

あるいは、演算子内のグループまたは属性を選択します。

2. 選択したアイテムを右クリックし、「**詳細をオープン**」を選択します。

マッピング・エディタに演算子エディタが表示され、「名前」タブ、「グループ」タブ、および各タイプの演算子のグループの「入力/出力」タブが表示されます。

いくつかの演算子には追加のタブが含まれます。たとえば、Match Merge演算子には一致ルールおよびマージ・ルールを定義するタブが含まれます。

3. 各タブに表示されるプロンプトに従い、「**OK**」をクリックして終了します。

タブの実行方法については、「**ヘルプ**」を参照してください。

演算子の接続

マッピング・ソース演算子、データを変換する演算子およびターゲット演算子を選択した後は、これらの演算子を接続できます。データ・フローの接続により、ソースから演算子を通してターゲットに至るデータ・フローの様子が視覚的に表されます。

演算子は、次のいずれかの方法で接続できます。

- **属性の接続**: 一度に個々の演算子属性を相互に接続します。
- **グループの接続**: 2つのグループ間ですべての属性を接続するための条件を定義します。
- **演算子ウィザードの使用**: ピボット演算子、Name and Address演算子などの演算子は、ウィザードを使用してデータ・フロー接続を定義できます。

属性の接続

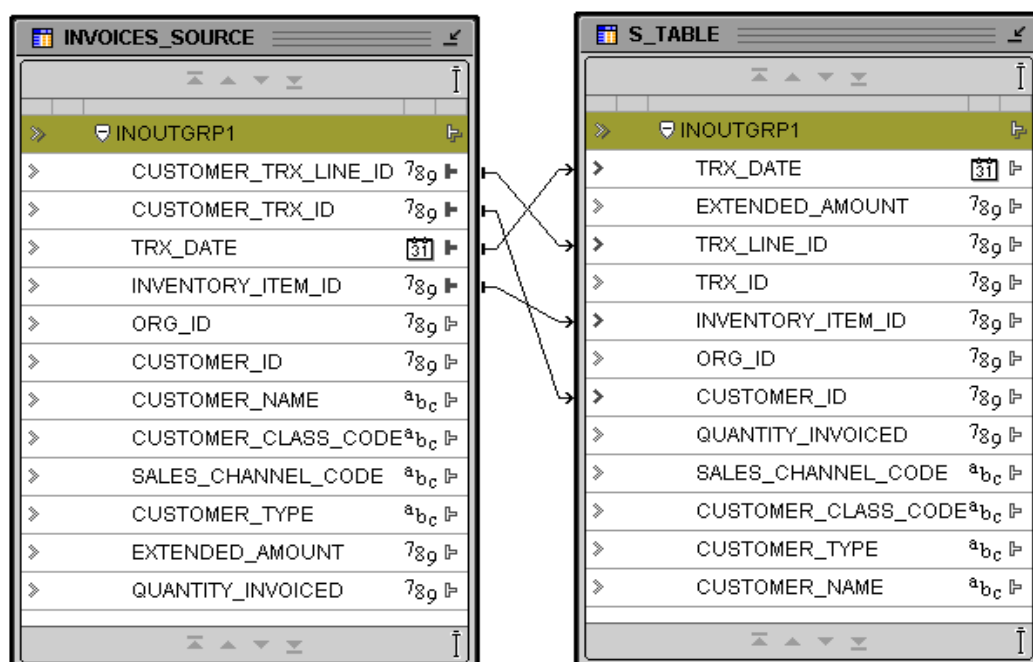
一方の演算子の単一の出力属性から、他方の演算子の単一の入力属性に線を引きます。

属性を接続する手順

1. マウス・ボタンをクリックしたまま、ポインタを出力属性の上に置きます。
2. 出力属性からデータのフロー先となる入力属性にマウスをドラッグします。

マウスをドラッグするとき、接続を示す行がマッピング・エディタ・キャンパスに表示されます。
3. 入力属性の上でマウス・ボタンを放します。
4. ステップ1から3を繰り返し、必要なデータ・フロー接続をすべて作成します。

図5-3 マッピングで接続した演算子



「図5-3 マッピングで接続した演算子」の説明

属性を接続するときは、次のルールに留意します。

- 同じ入力属性には2度接続できません。
- 同じ演算子内の属性には接続できません。
- 入力専用の属性から接続することも、出力専用の属性に接続することもできません。
- 確立されているカーディナリティに矛盾する方法では演算子を接続できません。かわりに、ジョイナ演算子を使用します。

グループの接続

グループを接続すると、属性が自動的にコピーされるか、詳細情報を入力するように指示されます。

演算子グループを既存の属性を持たないターゲット・グループに接続すると、属性が自動的にコピーされ、各属性が接続されます。これは、「例: マッピング・エディタによるステージング領域表の作成」に示すようなマッピングを設計する場合に便利です。

例: マッピング・エディタによるステージング領域表の作成

マッピング・エディタでバインドされていない演算子を使用すると、ステージング領域表をすばやく作成できます。

次の手順は、既存のソース表に基づいてステージング表を作成する方法を示しています。この手順は、ビュー、マテリアライズド・ビュー、フラット・ファイルおよび変換の作成にも使用できます。

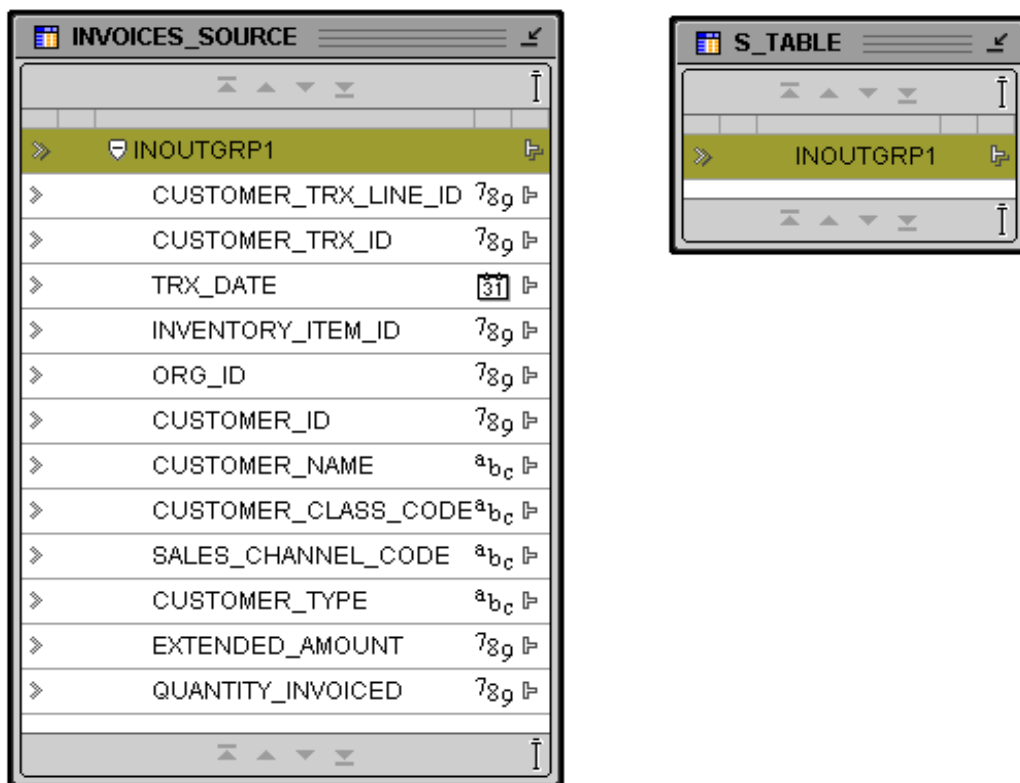
ソース表をステージング表にマップする手順

1. マッピング・エディタで、ソース表を追加します。

メニュー・バーから「マッピング」、「追加」、「データ・ソース/ターゲット」の順に選択します。「データ・ソース/ターゲット」メニューで、「演算子」を選択します。

2. 「演算子の追加」ダイアログ・ボックスを使用して、マッピング内のソース表の演算子を選択およびバインドします。「演算子の追加」ダイアログ・ボックスから、「バインドされていない演算子を属性なしで作成」を選択します。

図5-4 バインドされていないステージング表（属性なし）とソース表



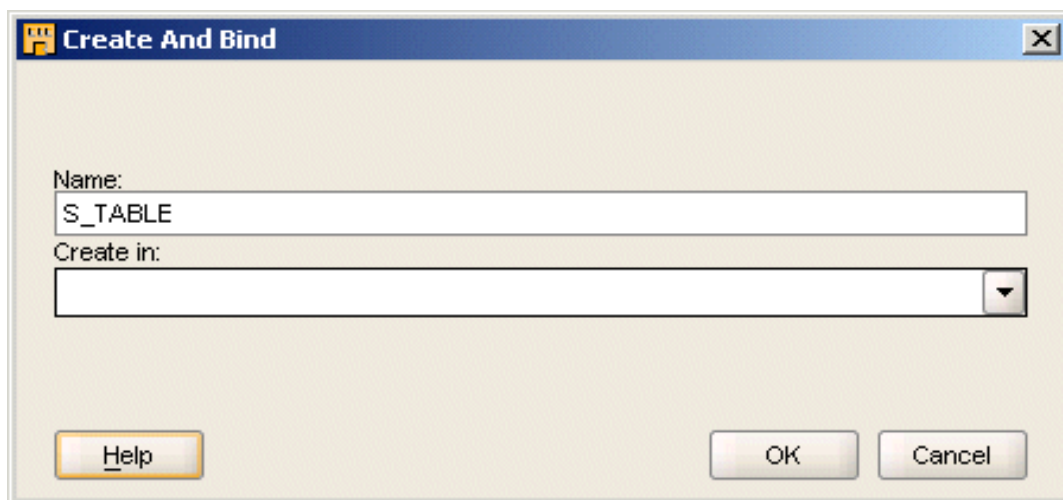
「図5-4 バインドされていないステージング表（属性なし）とソース表」の説明

3. ソース演算子のグループにマウス・ポインタを置き、マウス・ボタンを押したままにします。
4. ステージング領域表グループまでマウスをドラッグします。

OWBによって、ソース属性がステージング領域表にコピーされ、2つの演算子が接続されます。

5. マッピング・エディタでは、マッピングに追加したバインドされていない表を選択します。右クリックして「作成とバインド」を選択します。

図5-5 「作成とバインド」 ダイアログ・ボックス



「図5-5 「作成とバインド」 ダイアログ・ボックス」の説明

6. 「作成場所」に、表を作成するターゲット・モジュールを指定します。

OWBによって、指定したターゲット・モジュールに新しい表が作成されます。

マッピング・プロパティの設定

マッピング・キャンバスで空白を選択すると、エディタの左側にある「プロパティ・インスペクタ」にマッピング・プロパティが表示されます。マッピングについて次のプロパティを設定できます。

- [ターゲット・ロード順序](#)

ターゲット・ロード順序

マッピングに1つのターゲットのみが含まれている場合、あるいはマッピングがSQL*LoaderまたはABAPマッピングの場合は、ターゲット・ロード順序が適用されません。デフォルトの設定を受け入れてマッピングの設計を続行してください。

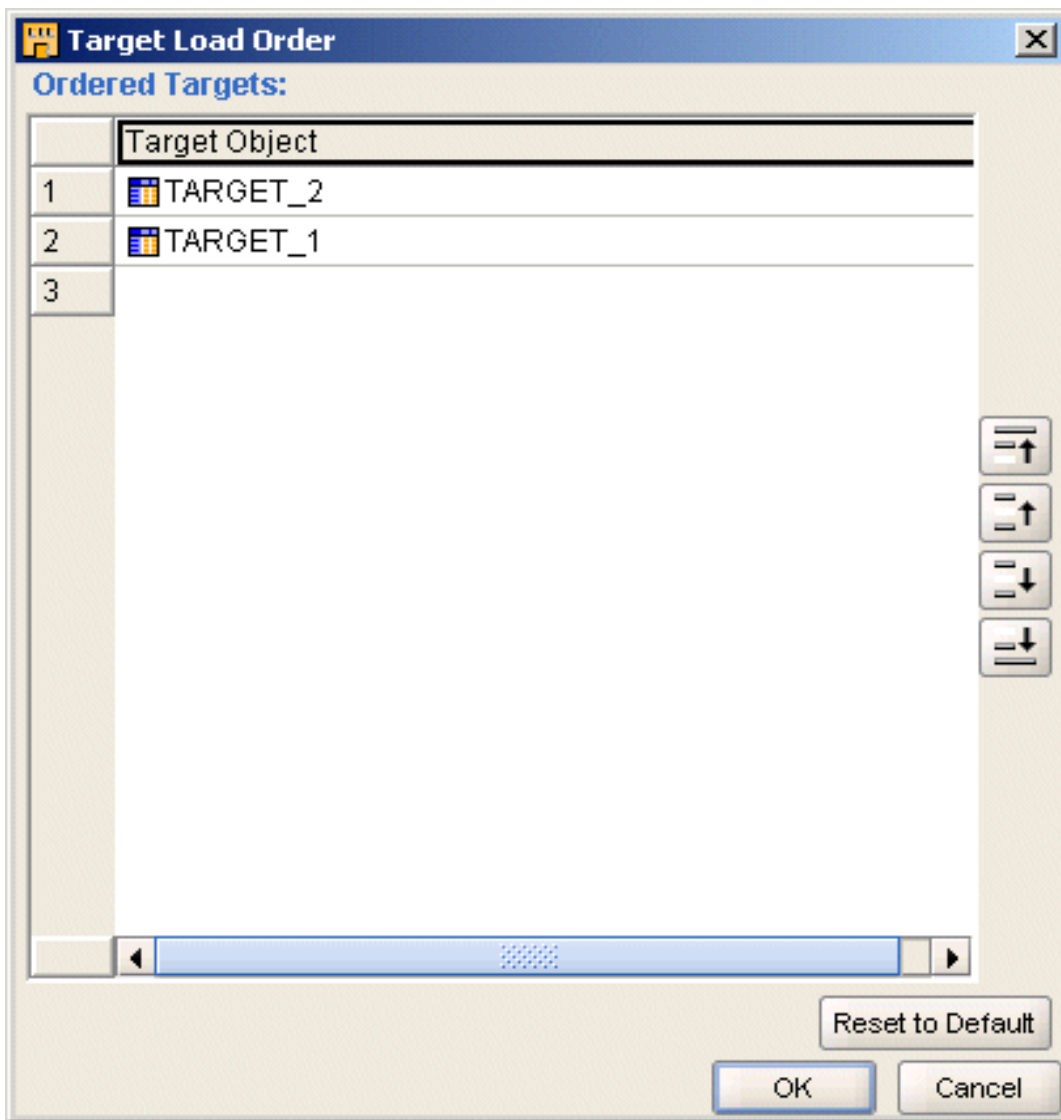
複数のターゲットがあるPL/SQLマッピングを設計すると、OWBにより、ターゲットをロードするデフォルトの順序が計算されます。ターゲット間に外部キー関連を定義すると、OWBにより、親の次に子をロードするデフォルトの順序が作成されます。外部キー関係を作成しない場合、またはターゲット表に再帰的な関連がある場合は、OWBにより、無作為な順序がデフォルトとして割り当てられます。

デフォルトのロード順序は、「ターゲット・ロード順序」プロパティを設定することで上書きできます。ターゲットの順序を変更する際に間違った変更を実行した場合は、[「デフォルトにリセット」](#)オプションを選択することで、デフォルトの順序に戻ります。

複数のターゲットのロード順序を指定する手順

1. マッピング・キャンバスの空白をクリックすると、プロパティ・ウィンドウの右上隅にマッピング・プロパティが表示されます。
2. 「ターゲットのロード順序のマップ」プロパティに進み、右側にある省略記号ボタンをクリックします。

図5-6 「ターゲット・ロード順序」 ダイアログ・ボックス



「図5-6 「ターゲット・ロード順序」 ダイアログ・ボックス」の説明

- ロード順序を変更するには、ターゲットを選択し、右側のシャトル・ボタンを使用してリスト内でターゲットを上または下に移動します。

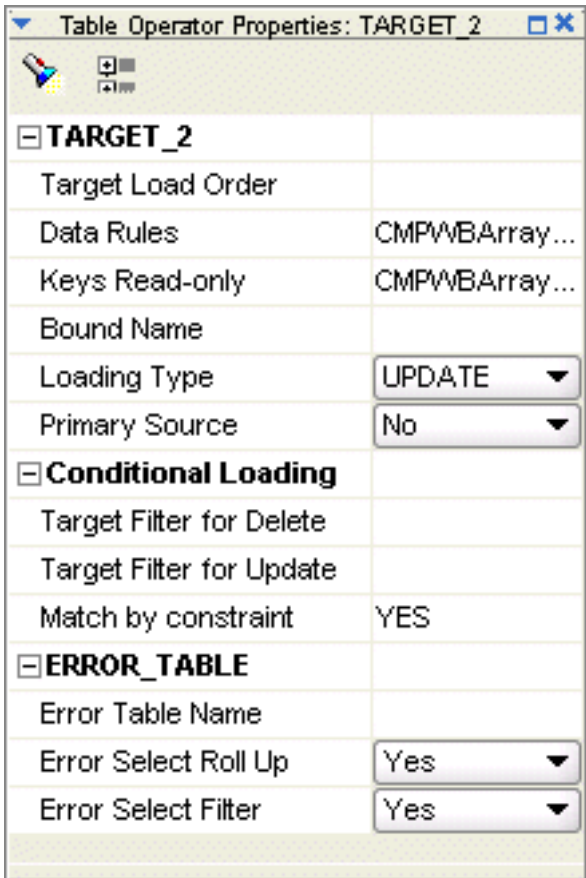
デフォルトにリセット

「デフォルトにリセット」ボタンは、ターゲット・ロード順序を再計算する場合に使用します。ターゲットの順序変更を誤った場合、または順序を割り当てた後に当初の順序が無効になるようなマッピングの設計変更があった場合は、再計算が必要になります。

演算子、グループおよび属性プロパティの設定

キャンバスでオブジェクトを選択すると、エディタの左側にあるプロパティ・インスペクタに、関連するプロパティが表示されます。

図5-7 表演算子のプロパティ・インスペクタ



「図5-7 演算子のプロパティ・インスペクタ」の説明

次のタイプのプロパティを表示および設定できます。

- **演算子のプロパティ:** 演算子全体に影響を与えるプロパティです。設定できるプロパティは演算子のタイプに依存します。
- **グループのプロパティ:** これは属性のグループに影響を与えるプロパティです。ほとんどの演算子には、グループのプロパティが設定されていません。グループ・プロパティが設定される演算子の例としては、スプリッタ演算子やデュプリケータ解除演算子があります。
- **属性のプロパティ:** これはソース演算子とターゲット演算子の属性に関するプロパティです。属性のプロパティの例としては、データ型、精度、スケールがあります。

演算子とワークスペース・オブジェクトの同期化

マッピングに使用する多くの演算子に対応する定義は、OWBワークスペースにあります。このことは、表演算子やビュー演算子など、ソース演算子とターゲット演算子に当てはまります。また、順序演算子や変換演算子など、定義が複数のマッピングにまたがって使用される他の演算子にも当てはまります。このような演算子を変更するときは、変更内容をワークスペース・オブジェクトに伝播する必要があります。

変更の伝播方向は、次の選択肢から決定できます。

ワークスペース・オブジェクトから演算子への同期化: 本番環境でマッピングの使用を開始した後、ソースまたはターゲットに対してETL設計に影響する変更が行われる場合があります。通常、このような変更を管理する最善の方法は、*Warehouse Builder*のオンライン・ヘルプで説明されているように、OWBの依存性マネージャを使用することです。依存性マネージャを使用すると、変更による影響が自動的に評価され、影響を受けるすべてのマッピングがまとめて同期化されます。また、[「ワークスペース・オブジェクトから演算子への同期化」](#)で説明されているように、マッピング・エディタで、オブジェクトを手動で同期化することもできます。

演算子からワークスペース・オブジェクトへの同期化: マッピングの演算子を変更する場合は、変更内容に対応するワークスペース定義に伝播させます。たとえば、インポートしてマッピングに使用したソースの場合、属性に複合物理名が付いている場合があります。

同期化がリフレッシュとは異なることに注意してください。リフレッシュ・コマンドでは、マルチユーザー環境で他のユーザーが行った変更にあわせて、確実に最新の状態が保たれます。同期化では、対応するワークスペース・オブジェクトと演算子が照合されます。

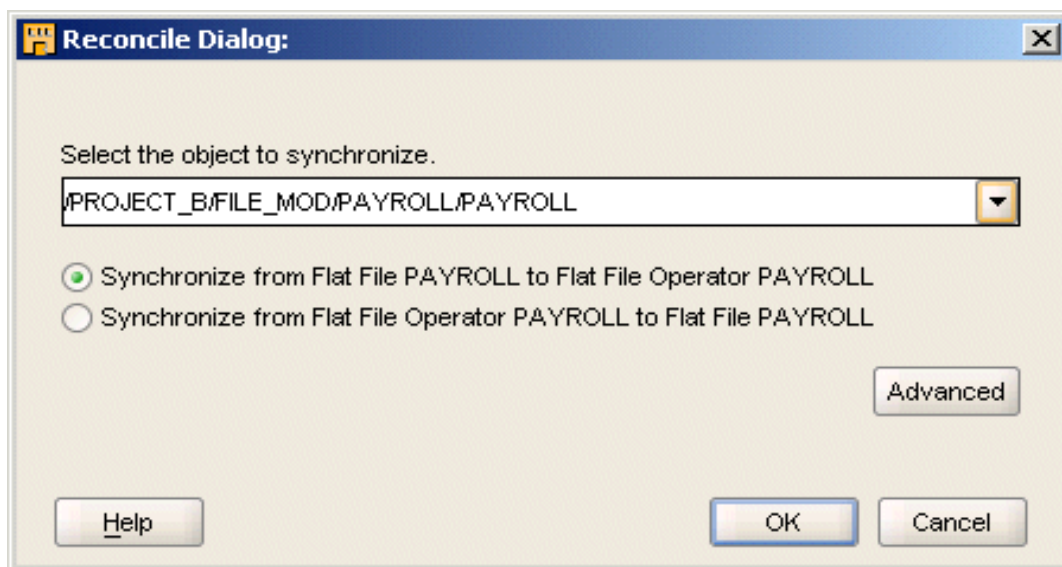
演算子の同期化

同期化する場合は、演算子を1つ選択し、指定したワークスペース・オブジェクトの定義とその演算子を同期化します。

演算子を同期化する手順

1. マッピング・エディタのキャンバスで演算子を選択します。
2. 「編集」メニューから、「同期化」を選択するか、演算子のヘッダーを右クリックして「同期化」を選択します。

図5-8 演算子の同期化



「図5-8 演算子の同期化」の説明

3. デフォルトでは、OWBにより、選択した演算子とワークスペース内の関連するオブジェクトを同期化するためのオプションが選択されます。デフォルトを受け入れるか、リスト・ボックスから別のワークスペース・オブジェクトを選択します。

この手順では、[ワークスペース・オブジェクトから演算子への同期化](#)、または[演算子からワークスペース・オブジェクトへの同期化](#)のいずれかを選択します。

4. (オプション) 「拡張」をクリックして「一致方針」を設定します。

「一致方針」の使用方法を参照するには、「ヘルプ」を選択します。

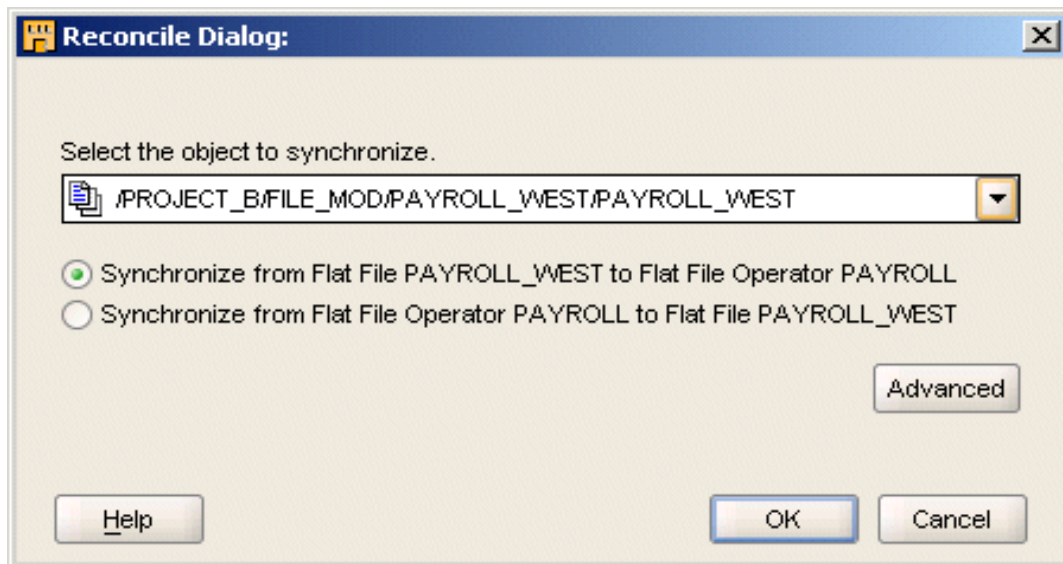
5. 「OK」をクリックします。

ワークスペース・オブジェクトから演算子への同期化

マッピング・エディタでは、次のような理由により、ワークスペース・オブジェクトから演算子への同期化を実行できます。

- **手動による変更の伝播:** ワークスペース・オブジェクトの変更を関連する演算子に伝播します。ワークスペース・オブジェクトの変更には、構造的な変更、属性名の変更、属性のデータ型の変更などがあります。ワークスペース・オブジェクトの変更を複数のマッピングに自動的に伝播する方法は、*Warehouse Builder*のオンライン・ヘルプを参照してください。
- **演算子と新規ワークスペース・オブジェクトの同期化:** たとえば、マッピングをデータ・ウェアハウスの特定のバージョンから新しいバージョンに移行し、各バージョンの別のオブジェクト定義を維持する場合、演算子と新規のワークスペース・オブジェクトを関連付けることができます。

図5-9 異なるワークスペース・オブジェクトからの同期化



「図5-9 異なるワークスペース・オブジェクトからの同期化」の説明

- **表によるマッピングのプロトタイプ作成:** 設計環境で作業している場合は、表を使用したETLロジックの設計を選択できます。ただし、本番環境では、他のワークスペース・オブジェクト・タイプ（ビュー、マテリアライズド・ビュー、キューブなど）に基づいたマッピングが可能です。

ワークスペース・オブジェクトに基づく演算子の同期化

表5-1に、演算子および同期化できるワークスペース・オブジェクトのタイプを示します。

表5-1 ワークスペース・オブジェクトと同期化される演算子

同期対象: 演算子	同期基準: ワークスペース・オブジェクトのタイプ
キューブ演算子	表、ビュー、マテリアライズド・ビュー、フラット・ファイル、ディメンションおよびキューブ
ディメンション演算子	表、外部表、ビュー、マテリアライズド・ビュー、フラット・ファイル、ディメンションおよびキューブ
外部表演算子	表、外部表、ビュー、マテリアライズド・ビュー、フラット・ファイル、ディメンションおよびキューブ
フラット・ファイル演算子	表、外部表、ビュー、マテリアライズド・ビュー、フラット・ファイル、ディメンションおよびキューブ
キー参照演算子	表のみ
マテリアライズド・ビュー演算子	表、外部表、ビュー、マテリアライズド・ビュー、ファイル、ディメンションおよびキューブ
マッピング後プロセス演算子	変換のみ
マッピング前プロセス演算子	変換のみ
順序演算子	順序のみ

表演算子	表、外部表、ビュー、マテリアライズド・ビュー、フラット・ファイル、ディメンションおよびキューブ
変換演算子	変換のみ
ビュー演算子	表、外部表、ビュー、マテリアライズド・ビュー、ファイル、ディメンションおよびキューブ

外部表演算子から同期化するとき、OWBは、関連付けられたフラット・ファイルではなく、ワークスペース外部表のみに基づいて演算子を更新することに注意してください。

演算子からワークスペース・オブジェクトへの同期化

マッピングの演算子を変更した場合は、それらの変更をワークスペース・オブジェクトに伝播する必要があります。同期化によって、表、ビュー、マテリアライズド・ビュー、変換およびフラット・ファイルの各演算子の変更を伝播できます。

次のような理由により、演算子からワークスペース・オブジェクトへの同期化を実行します。

- **変更の伝播:** 演算子の変更を関連するワークスペース・オブジェクトに伝播します。演算子または属性のビジネス名を変更すると、OWBによってそのビジネス名の最初の30文字がバウンド名として伝播されます。
- **ワークスペース・オブジェクトの置換:** 同期化により、既存のワークスペース・オブジェクトを置換します。

演算子からの同期化は、他の演算子とワークスペース・オブジェクトとの依存関係には影響を与えません。

6 ターゲット・スキーマへの配布およびETLロジックの実行

この章の内容は次のとおりです。

- [配布について](#)
- [オブジェクトの配布](#)
- [ETLジョブの開始](#)

配布について

配布は、OWBリポジトリ内の論理オブジェクトから、ターゲット・ロケーション内に物理オブジェクトを作成するプロセスです。

マッピングまたはプロセス・フローの配布には、次の手順が含まれます。

- 必要に応じて、PL/SQL、SQL*LoaderまたはABAPスクリプトを生成します。
- スクリプトをデザイン・センターからコントロール・センターにコピーします。また、SQL*Loader制御ファイルもコントロール・センターにコピーします。
- 新しいコネクタを配布します。つまり、ロケーション間にデータベース・リンクとデータベース・ディレクトリを作成します。

マッピングまたはプロセス・フローの配布の後、スクリプトを明示的に開始する必要があります。

配布できるのは、ユーザーにCOMPILE権限があるオブジェクトのみです。デフォルトでは、リポジトリ内のすべてのオブジェクトに対してこの権限があります。ただし、リポジトリの所有者が異なるセキュリティ・ポリシーを設定している場合があります。

デザイン・センターのナビゲーション・ツリーから直接配布するか、またはコントロール・センター・マネージャを使用して配布できます。

注意

オブジェクトは常にOWBを使用して保守してください。配布した物理オブジェクトをSQLで手動で変更しないでください。そうしないと、論理オブジェクトと物理オブジェクトの同期がとれなくなり、予測できない結果を引き起こす可能性があります。

注意

オブジェクトを配布するたびに、OWBでは、すべての設計オブジェクトに対するすべての変更がリポジトリに自動的に保存されます。「プリファレンス」ダイアログ・ボックスの「コミットを要求」を選択すると、警告メッセージの表示を選択できます。

コントロール・センターの概要

コントロール・センターには、すべての配布に関する詳細情報が格納されます。次の情報が含まれており、オブジェクト単位またはジョブ単位でアクセスできます。

- 各オブジェクトの現在の配布ステータス
- 各オブジェクトに対するすべての配布試行の履歴
- 各マッピングおよびプロセス・フローに対するすべてのETL開始試行の履歴

- すべての配布ジョブからのメッセージの完全ログ

コントロール・センターは、ターゲット・ロケーションと同じデータベース内にスキーマとして実装されています。各リポジトリにはデフォルトのコントロール・センターがあり、これは、リポジトリ所有者のスキーマ内に作成されています。たとえば、REP_OWNERリポジトリの所有者には、REP_OWNERというスキーマがあり、このスキーマには、デザイン・センターとデフォルトのコントロール・センターの両方からのメタデータが格納されます。

ローカル・システムに配布するには、デフォルトのコントロール・センターを使用できます。または、別のシステムに配布するために、追加のコントロール・センターを作成できます。一度に1つのコントロール・センターのみアクティブにできます。

コントロール・センター・マネージャでは、配布のすべての側面を表示および管理できる総合的な配布コンソールが提供されます。ここから、アクティブなコントロール・センターに格納されている情報にアクセスできます。

リポジトリ・ブラウザを使用して、配布データ・レポートにアクセスすることもできます (*Warehouse Builder*のオンライン・ヘルプを参照)。

新規コントロール・センターを作成する手順

1. 接続ナビゲータで、「コントロール・センター」を右クリックして「新規」を選択します。

「コントロール・センターの作成」ダイアログ・ボックスが表示されます。

2. このダイアログ・ボックスを完了します。詳細は、「ヘルプ」ボタンをクリックしてください。

コントロール・センターは、構成の作成ウィザードを使用して作成することもできます。

コントロール・センターをアクティブにする手順

1. コントロール・センターを使用するように、構成を作成または編集します。

「新規構成の作成」を参照してください。

2. その構成をアクティブ化します。

配布の物理詳細の構成

OWBでは、オブジェクトの論理設計が配布の物理詳細から分離されています。この分離は、構成パラメータに物理詳細を格納することによって実現します。構成と呼ばれるオブジェクトには、すべての構成設定が格納されます。各配布ロケーションには、それぞれのオブジェクト・パラメータに異なる設定を指定して、異なる構成を作成できます。

配布前に、ターゲット・オブジェクト、マッピングおよびモジュールの構成を確認してください。

オブジェクトが配布可能になる条件は、次のとおりです。

- そのターゲット・ロケーションが完全に定義され、有効であり、オブジェクトのモジュールに対して選択されている必要があります。
- その配布可能パラメータが選択されている (デフォルト) 必要があります。
- 検証および生成がエラーなしに実行されている必要があります。

配布アクション

デザイン・センターで定義した新規オブジェクトは、ただちにコントロール・センター・マネージャにリストされます。各オブジェクトには表示可能なデフォルトの配布アクションがあります。このデフォルトは以前のアクションによって設定され、オブジェクトのタイプによって異なります。デフォルトは、コントロール・センター・マネージャで別の配布アクションを選択することで上書きできます。

次の配布アクションがあります。

- **作成:** ターゲット・ロケーションにオブジェクトを作成します。指定した名前のオブジェクトがすでに存在している場合は、エラーが発生する可能性があります。
- **アップグレード:** 可能な場合、データを喪失することなくオブジェクトを変更します。アップグレードは、元に戻して再実行できます。このアクションは、スケジュールなど、一部のオブジェクト・タイプには使用できません。
- **削除:** ターゲット・ロケーションからオブジェクトを削除します。
- **置換:** オブジェクトを削除して再作成します。このアクションは、アップグレードより高速ですが、データはすべて削除されます。

配布プロセス

データ・システムのライフサイクル期間中は、通常、配布プロセスで次の手順を実行します。

1. オブジェクト設定および使用するコントロール・センターが指定された構成を選択します。
2. オブジェクトをターゲット・ロケーションに配布します。個別に配布、1つずつ順に配布、すべて一度に配布のいずれも可能です。
3. 配布の結果を確認します。オブジェクトの配布に失敗した場合は、問題を修正し、再試行してください。
4. ETLプロセスを開始します。
5. ユーザーの要求やソース・データの変更などにあわせて、ターゲット・オブジェクトの設計を変更します。
6. 変更したオブジェクトの配布アクションを「アップグレード」または「置換」に設定します。
7. これらの手順を繰り返します。

注意

OWBでは、配布前に、リポジトリに対するすべての変更が自動的に保存されます。

オブジェクトの配布

配布は、生成したコードを使用して、メタデータからターゲット・ロケーションに物理オブジェクトを作成するプロセスです。オブジェクトは、プロジェクト・ナビゲータから配布することも、コントロール・センター・マネージャを使用して配布することもできます。OWBでは、オブジェクトを自動的に検証して生成します。

プロジェクト・ナビゲータからの配布は、デフォルト・アクションに制限されています。デフォルト・アクションは、「作成」、「置換」、「削除」または「更新」に設定されている可能性があります。このデフォルト・アクションを上書きするには、配布プロセスを完全に制御するコントロール・センター・マネージャを使用します。

プロジェクト・ナビゲータから配布する手順

該当するオブジェクトを選択して、ツールバーの「配布」アイコンをクリックします。

ステータス・メッセージが、「デザイン・センター」ウィンドウの下部に表示されます。配布が完了したことを通知する場合は、配布前にプリファレンスで「配布完了メッセージの表示」を選択します。

コントロール・センター・マネージャを開く手順

1. プロジェクトを開きます。
2. 「ツール」メニューから「コントロール・センター・マネージャ」を選択します。

ETLジョブの開始

ETLは、データをそのソース・ロケーションから抽出し、マッピングで定義したとおりに変換し、ターゲット・オブジェクトにロードするプロセスです。ETLを開始するときは、データをジョブとしてOWBジョブ・キューに送信します。このジョブは、すぐに開始できます。また、Oracle Enterprise Managerのスケジューラなど、スケジューラを使用する場合は、スケジュールされた時間に開始できます。ETLは、配布と同様に、プロジェクト・ナビゲータから開始することも、コントロール・センター・マネージャを使用して開始することもできます。SQLスクリプトを実行するOWB外部のツールをして、ETLを開始することもできます。

プロジェクト・ナビゲータからETLを開始する手順

マッピングまたはプロセス・フローを選択して、「設計」メニューから「開始」を選択します。

データの表示

ETLを完了した後は、OWB内のデータ・オブジェクトを確認し、結果が期待どおりであることを簡単に検証できます。

データを表示する手順

プロジェクト・ナビゲータで、オブジェクトを右クリックして「データ」を選択します。「データ・ビューア」が開き、オブジェクトの内容が表示されます。

第III部

データ・ウェアハウスのレポート作成

ここでは、データ・ウェアハウスの管理方法について説明します。内容は次のとおりです。

- [第7章「レポートおよび分析のSQL」](#)

7 レポートおよび分析のSQL

この項では、ビジネス問合せから導出された効果的なビジネス・レポートの作成方法を説明します。この項の内容は次のとおりです。

- [ビジネス問合せに回答するSQLアナリティック機能の使用](#)
- [パーティション外部結合を使用したスパースなデータの処理](#)
- [ビジネス問合せを単純化するWITH句の使用](#)

ビジネス問合せに回答するSQLアナリティック機能の使用

Oracle Databaseでは、集計および分析SQL関数のファミリの導入によってSQLの分析処理機能を強化しています。これらの関数によって、ランキング、パーセンタイルおよび移動平均の計算など、次の問合せに回答できます。

- 上位10製品の国別売上は何ですか。
- 在庫のある製品の毎週の移動平均はどうですか。
- 総売上高の何パーセントが第4四半期に発生していますか。
- 第4四半期の平均割引は年間の平均割引よりどれくらい高いですか。
- 国内の精油所の20パーセントが閉鎖された場合、既存の精油所の収益性ランキングはどうなりますか。

集計関数は、合計の異なるタイプを導出でき、追加計算のこれらの合計を使用できるデータ・ウェアハウスの基礎となる部分です。データ・ウェアハウスの集計パフォーマンスを向上させるには、Oracle DatabaseはGROUP BY列に対するいくつかの拡張を提供します。CUBE、ROLLUP、GROUPINGおよびGROUPING SETS関数により、より早く簡単に問合せおよびレポート作成ができます。ROLLUP関数は、最も個人的で詳細なものから総計まで、集計レベルを上げながらSUM、COUNT、MAX、MINおよびAVGなどの集計を計算します。CUBE関数はROLLUPに似た拡張で、単一の文で集計可能なすべての組合せを計算できます。

分析関数は行のグループに基づいて集計値を計算します。これらの関数は各グループに対して複数の行を戻すという点で集計関数とは異なります。この行のグループは**ウィンドウ**と呼ばれます。このウィンドウを使用すると、移動平均や累積合計などを計算できます。行ウィンドウは各行に対して定義されます。このウィンドウは現行の行の計算を実行するために使用する行の範囲を決定します。ウィンドウ・サイズは時間などの論理間隔、または行の物理的な行の数に基づくことができます。一部の関数はウィンドウとのみ使用され、ウィンドウ関数として参照されません。

パフォーマンスを向上させるために、集計および分析関数をパラレルで実行できます。つまり、複数のプロセスでこれらの関数すべてを同時に実行できます。これらの機能によって計算、分析およびレポート作成がより簡単で効果的になり、データ・ウェアハウ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。

高度なSQLおよびPL/SQL機能を活用でき、Oracle Databaseはビジネス問合せをSQLに変更します。この項ではこれらの高度な機能を説明します。この項の内容は次のとおりです。

- [ROLLUP関数を使用したレポートへの合計の追加方法](#)
- [CUBE関数を使用した異なるレベルでの合計の分割方法](#)
- [GROUPING関数を使用した小計の追加方法](#)
- [GROUPING SETS関数を使用した集計の結合方法](#)

- [RANK関数を使用したランキングの計算方法](#)
- [合計に対する相対的な寄与率の計算方法](#)
- [ウィンドウ関数を使用した行間計算の実行方法](#)
- [ウィンドウ関数を使用した移動平均の計算方法](#)

ROLLUP関数を使用したレポートへの合計の追加方法

ROLLUP関数によって、SELECT文で総計やディメンションの特定のグループを横断する複数の小計レベルを計算できます。ROLLUP関数はGROUP BY句の単純な拡張であるため、この構文は非常に簡単に使用できます。ROLLUP関数は非常に効果的で、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関数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関数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するためです。ROLLUPは、その引数として、グループ化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、グループ化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関数を使用するとき

タスクに小計が含まれる場合、特に小計が時間または地理などの階層ディメンションに基づいている場合、ROLLUP関数は有効です。また、ROLLUP関数によって、マテリアライズド・ビューのメンテナンスが簡素化および高速化できます。

例：ROLLUP関数の使用

ビジネス・レポートの準備時に共通して要求されるのは、収入額順で、異なる製品カテゴリにわたる四半期の総売上高の検索です。次の問合せはこれを達成し、後に実行するより複雑な問合せの構築の開始点に使用されます。

```
SELECT t.calendar_quarter_desc quarter
, p.prod_category category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
FROM times t
, products p
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY t.calendar_quarter_desc, p.prod_category
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);
```

QUARTER	CATEGORY	REVENUE
2001-01	Software/Other	\$860,819.81
2001-01	Electronics	\$1,239,287.71
2001-01	Hardware	\$1,301,343.45
2001-01	Photo	\$1,370,706.38
2001-01	Peripherals and Accessories	\$1,774,940.09
2001-02	Software/Other	\$872,157.38
2001-02	Electronics	\$1,144,187.90
2001-02	Hardware	\$1,557,059.59
2001-02	Photo	\$1,563,475.51
2001-02	Peripherals and Accessories	\$1,785,588.01
2001-03	Software/Other	\$877,630.85
2001-03	Electronics	\$1,017,536.82
2001-03	Photo	\$1,607,315.63
2001-03	Hardware	\$1,651,454.29
2001-03	Peripherals and Accessories	\$2,042,061.04
2001-04	Software/Other	\$943,296.36
2001-04	Hardware	\$1,174,512.68
2001-04	Electronics	\$1,303,838.52
2001-04	Photo	\$1,792,131.39
2001-04	Peripherals and Accessories	\$2,257,118.57

この問合せは便利ですが、同じレポート上で異なるカテゴリの総計を確認する必要がある可能性があります。次の例は、元の実行問合せに総計を追

加するためのROLLUP関数の使用方法を説明しています。

```
SELECT t.calendar_quarter_desc quarter
, p.prod_category category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
FROM times t
, products p
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY ROLLUP(t.calendar_quarter_desc, p.prod_category)
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);
```

QUARTER	CATEGORY	REVENUE
2001-01	Software/Other	\$860,819.81
2001-01	Electronics	\$1,239,287.71
2001-01	Hardware	\$1,301,343.45
2001-01	Photo	\$1,370,706.38
2001-01	Peripherals and Accessories	\$1,774,940.09
2001-01		\$6,547,097.44
2001-02	Software/Other	\$872,157.38
2001-02	Electronics	\$1,144,187.90
2001-02	Hardware	\$1,557,059.59
2001-02	Photo	\$1,563,475.51
2001-02	Peripherals and Accessories	\$1,785,588.01
2001-02		\$6,922,468.39
2001-03	Software/Other	\$877,630.85
2001-03	Electronics	\$1,017,536.82
2001-03	Photo	\$1,607,315.63
2001-03	Hardware	\$1,651,454.29
2001-03	Peripherals and Accessories	\$2,042,061.04
2001-03		\$7,195,998.63
2001-04	Software/Other	\$943,296.36
2001-04	Hardware	\$1,174,512.68
2001-04	Electronics	\$1,303,838.52
2001-04	Photo	\$1,792,131.39
2001-04	Peripherals and Accessories	\$2,257,118.57
2001-04		\$7,470,897.52
		\$28,136,461.98

CUBE関数を使用した異なるレベルでの合計の分割方法

CUBE関数は指定されたグループ化列の集合を取り、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。多次元分析によって、CUBE関数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関数および追加の組合せに含まれるすべての値が含まれます。

CUBE関数を使用するとき

複数の表にわたるレポートを必要とする状況では、CUBE関数の使用を検討してください。複数の表にわたるレポートに必要なデータは、CUBE関数を使用する単一のSELECT文で生成できます。ROLLUPと同様に、CUBE関数もマテリアライズド・ビューの生成に便利です。CUBE関数を含む問合せがパラレルで実行されるより、マテリアライズド・ビューを移入した方がより高速になることに注意してください。

例：CUBE関数の使用

四半期の総額のみでなく、選択した期間における異なる製品カテゴリの総額を入手する場合、次の例のように、CUBE関数を使用してこれらの計算ができます。

CUBE関数を使用する手順

```
SELECT t.calendar_quarter_desc quarter
, p.prod_category category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
FROM times t
, products p
```

```

, sales
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY CUBE(t.calendar_quarter_desc, p.prod_category)
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);

```

QUARTER	CATEGORY	REVENUE
2001-01	Software/Other	\$860,819.81
2001-01	Electronics	\$1,239,287.71
2001-01	Hardware	\$1,301,343.45
2001-01	Photo	\$1,370,706.38
2001-01	Peripherals and Accessories	\$1,774,940.09
2001-01		\$6,547,097.44
2001-02	Software/Other	\$872,157.38
2001-02	Electronics	\$1,144,187.90
2001-02	Hardware	\$1,557,059.59
2001-02	Photo	\$1,563,475.51
2001-02	Peripherals and Accessories	\$1,785,588.01
2001-02		\$6,922,468.39
2001-03	Software/Other	\$877,630.85
2001-03	Electronics	\$1,017,536.82
2001-03	Photo	\$1,607,315.63
2001-03	Hardware	\$1,651,454.29
2001-03	Peripherals and Accessories	\$2,042,061.04
2001-03		\$7,195,998.63
2001-04	Software/Other	\$943,296.36
2001-04	Hardware	\$1,174,512.68
2001-04	Electronics	\$1,303,838.52
2001-04	Photo	\$1,792,131.39
2001-04	Peripherals and Accessories	\$2,257,118.57
2001-04		\$7,470,897.52
	Software/Other	\$3,553,904.40
	Electronics	\$4,704,850.95
	Hardware	\$5,684,370.01
	Photo	\$6,333,628.91
	Peripherals and Accessories	\$7,859,707.71
		\$28,136,461.98

GROUPING関数を使用した小計の追加方法

ROLLUPおよびCUBE関数を使用する際には、2つの課題があります。第一に、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し、指定された小計の正確な小計レベルをどのように探し出すかということです。合計に対する割合を計算する場合に小計を使用する必要があるため、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に、格納されるNULL値およびROLLUP、またはCUBE関数によって作成されるNULL値の両方が問合せ結果に含まれる場合、どう処理するかという問題です。この2つをどのように区別するかが問題になります。

この問題はGROUPING関数で処理します。単一の列を引数として使用し、ROLLUPまたはCUBE関数により作成されたNULL値が発生した場合、GROUPING関数は1を返します。つまり、NULL値が小計の行であることを示す場合、GROUPINGは1を返します。格納されたNULL値を含むその他のタイプの値では0を返します。

GROUPING関数を使用するとき

NULL値またはROLLUPまたはCUBE操作で作成されたNULL値を処理する必要がある場合、GROUPING関数を使用します。NULL値を使用する1つの理由に、NULLフィールドに説明を置くことです。たとえば、番号が総計を示すテキストなどです。

例：GROUPING関数の使用

値が総計を示している場合にわかりにくいため、レポート内の説明の列がさらに必要になる場合があります。GROUPING関数を使用すると、次の例のように、問合せ結果に総計を示すラベルを挿入することができます。

GROUPING関数を使用する手順

```

SELECT DECODE(GROUPING(t.calendar_quarter_desc)
, 0, t.calendar_quarter_desc
, 1, 'TOTAL'

```

```

) quarter
, DECODE (GROUPING(p.prod_category)
, 0, p.prod_category
, 1, 'TOTAL'
) category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
FROM times t
, products p
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY CUBE(t.calendar_quarter_desc, p.prod_category)
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);

```

QUARTER	CATEGORY	REVENUE
2001-01	Software/Other	\$860,819.81
2001-01	Electronics	\$1,239,287.71
2001-01	Hardware	\$1,301,343.45
2001-01	Photo	\$1,370,706.38
2001-01	Peripherals and Accessories	\$1,774,940.09
2001-01	TOTAL	\$6,547,097.44
2001-02	Software/Other	\$872,157.38
2001-02	Electronics	\$1,144,187.90
2001-02	Hardware	\$1,557,059.59
2001-02	Photo	\$1,563,475.51
2001-02	Peripherals and Accessories	\$1,785,588.01
2001-02	TOTAL	\$6,922,468.39
2001-03	Software/Other	\$877,630.85
2001-03	Electronics	\$1,017,536.82
2001-03	Photo	\$1,607,315.63
2001-03	Hardware	\$1,651,454.29
2001-03	Peripherals and Accessories	\$2,042,061.04
2001-03	TOTAL	\$7,195,998.63
2001-04	Software/Other	\$943,296.36
2001-04	Hardware	\$1,174,512.68
2001-04	Electronics	\$1,303,838.52
2001-04	Photo	\$1,792,131.39
2001-04	Peripherals and Accessories	\$2,257,118.57
2001-04	TOTAL	\$7,470,897.52
TOTAL	Software/Other	\$3,553,904.40
TOTAL	Electronics	\$4,704,850.95
TOTAL	Hardware	\$5,684,370.01
TOTAL	Photo	\$6,333,628.91
TOTAL	Peripherals and Accessories	\$7,859,707.71
TOTAL	TOTAL	\$28,136,461.98

GROUPING SETS関数を使用した集計の結合方法

GROUP BY句内でGROUPING SETS関数を使用して、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の全データを計算せずに複数のディメンションにおよぶ正確な指定ができ、すべてのディメンションの総計は必要ありません。

GROUPING SETS関数を使用するとき

データ・キューブ内の特定の小計が必要ですべての小計が入手できない場合、GROUPING SETS関数を使用します。

例：GROUPING SETS関数の使用

販売チャネルに基づいた営業番号の総計を参照する場合、チャネル・クラスごとの総計を取得するため別の問合せを追加するかわりに、次の例のように、GROUPING SETS関数を使用できます。

GROUPING SETS関数を使用する手順

```

SELECT DECODE (GROUPING(t.calendar_quarter_desc)
, 0, t.calendar_quarter_desc
, 1, 'TOTAL'

```

```

) quarter
, DECODE(GROUPING(c.channel_class)
, 0, c.channel_class
, 1, '--all--'
) channel
, DECODE(GROUPING(p.prod_category)
, 0, p.prod_category
, 1, 'TOTAL'
) category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
FROM times t
, products p
, channels c
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND c.channel_id = s.channel_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY GROUPING SETS(c.channel_class,
CUBE(t.calendar_quarter_desc, p.prod_category))
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);

```

QUARTER	CHANNEL	CATEGORY	REVENUE
2001-01	--all--	Software/Other	\$860,819.81
2001-01	--all--	Electronics	\$1,239,287.71
2001-01	--all--	Hardware	\$1,301,343.45
2001-01	--all--	Photo	\$1,370,706.38
2001-01	--all--	Peripherals and Accessories	\$1,774,940.09
2001-01	--all--	TOTAL	\$6,547,097.44
2001-02	--all--	Software/Other	\$872,157.38
2001-02	--all--	Electronics	\$1,144,187.90
2001-02	--all--	Hardware	\$1,557,059.59
2001-02	--all--	Photo	\$1,563,475.51
2001-02	--all--	Peripherals and Accessories	\$1,785,588.01
2001-02	--all--	TOTAL	\$6,922,468.39
2001-03	--all--	Software/Other	\$877,630.85
2001-03	--all--	Electronics	\$1,017,536.82
2001-03	--all--	Photo	\$1,607,315.63
2001-03	--all--	Hardware	\$1,651,454.29
2001-03	--all--	Peripherals and Accessories	\$2,042,061.04
2001-03	--all--	TOTAL	\$7,195,998.63
2001-04	--all--	Software/Other	\$943,296.36
2001-04	--all--	Hardware	\$1,174,512.68
2001-04	--all--	Electronics	\$1,303,838.52
2001-04	--all--	Photo	\$1,792,131.39
2001-04	--all--	Peripherals and Accessories	\$2,257,118.57
2001-04	--all--	TOTAL	\$7,470,897.52
TOTAL	--all--	Software/Other	\$3,553,904.40
TOTAL	--all--	Electronics	\$4,704,850.95
TOTAL	--all--	Hardware	\$5,684,370.01
TOTAL	--all--	Photo	\$6,333,628.91
TOTAL	Indirect	TOTAL	\$6,709,496.66
TOTAL	--all--	Peripherals and Accessories	\$7,859,707.71
TOTAL	Others	TOTAL	\$8,038,529.96
TOTAL	Direct	TOTAL	\$13,388,435.36
TOTAL	--all--	TOTAL	\$28,136,461.98

RANK関数を使用したランキングの計算方法

ビジネス情報の処理には、複雑なランキング、小計、移動平均およびリード/ラグ比較を含む高度な計算が必要です。これらの集計および分析タスクはビジネス・インテリジェンス問合せに不可欠です。ウィンドウ関数の使用によって解決します。

RANK関数を使用するとき

複雑な問合せを実行したり問合せ結果を分析する場合には、RANK関数を使用します。

例：RANK関数の使用

四半期の収入番号のランクを示す追加の列を確認する場合、次の例のように、RANK関数を使用します。

RANK関数を使用する手順

```
SELECT DECODE(GROUPING(t.calendar_quarter_desc)
, 0, t.calendar_quarter_desc
, 1, 'TOTAL'
) quarter
, DECODE(GROUPING(t.calendar_quarter_desc) + GROUPING(p.prod_category)
, 0, RANK() OVER (PARTITION BY t.calendar_quarter_desc
ORDER BY SUM(s.amount_sold))
, 1, null
) ranking
, DECODE(GROUPING(c.channel_class)
, 0, c.channel_class
, 1, '--all--'
) channel
, DECODE(GROUPING(p.prod_category)
, 0, p.prod_category
, 1, 'TOTAL'
) category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
FROM times t
, products p
, channels c
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND c.channel_id = s.channel_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY GROUPING SETS(c.channel_class,
CUBE(t.calendar_quarter_desc, p.prod_category))
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);
```

QUARTER	RANKING	CHANNEL	CATEGORY	REVENUE
2001-01	1	--all--	Software/Other	\$860,819.81
2001-01	2	--all--	Electronics	\$1,239,287.71
2001-01	3	--all--	Hardware	\$1,301,343.45
2001-01	4	--all--	Photo	\$1,370,706.38
2001-01	5	--all--	Peripherals and Accessories	\$1,774,940.09
2001-01		--all--	TOTAL	\$6,547,097.44
2001-02	1	--all--	Software/Other	\$872,157.38
2001-02	2	--all--	Electronics	\$1,144,187.90
2001-02	3	--all--	Hardware	\$1,557,059.59
2001-02	4	--all--	Photo	\$1,563,475.51
2001-02	5	--all--	Peripherals and Accessories	\$1,785,588.01
2001-02		--all--	TOTAL	\$6,922,468.39
2001-03	1	--all--	Software/Other	\$877,630.85
2001-03	2	--all--	Electronics	\$1,017,536.82
2001-03	3	--all--	Photo	\$1,607,315.63
2001-03	4	--all--	Hardware	\$1,651,454.29
2001-03	5	--all--	Peripherals and Accessories	\$2,042,061.04
2001-03		--all--	TOTAL	\$7,195,998.63
2001-04	1	--all--	Software/Other	\$943,296.36
2001-04	2	--all--	Hardware	\$1,174,512.68
2001-04	3	--all--	Electronics	\$1,303,838.52
2001-04	4	--all--	Photo	\$1,792,131.39
2001-04	5	--all--	Peripherals and Accessories	\$2,257,118.57
2001-04		--all--	TOTAL	\$7,470,897.52
TOTAL		--all--	Software/Other	\$3,553,904.40
TOTAL		--all--	Electronics	\$4,704,850.95
TOTAL		--all--	Hardware	\$5,684,370.01
TOTAL		--all--	Photo	\$6,333,628.91
TOTAL			Indirect TOTAL	\$6,709,496.66
TOTAL		--all--	Peripherals and Accessories	\$7,859,707.71
TOTAL		Others	TOTAL	\$8,038,529.96
TOTAL		Direct	TOTAL	\$13,388,435.36
TOTAL		--all--	TOTAL	\$28,136,461.98

In this example, the PARTITION BY clause defines the boundaries for the RANK function.

合計に対する相対的な寄与率の計算方法

共通のビジネス・インテリジェンス要求は、特定の時間間隔に基づく総収入に対する各製品カテゴリの寄与率を計算します。

例：総計に対する相対的な寄与率の計算

四半期ベースで収入番号の差異を取得します。次の例のように、PARTITION BY製品カテゴリを持つウィンドウ関数を使用して実行できます。

総計に対する相対的な寄与率を計算する手順

```
SELECT DECODE(GROUPING(t.calendar_quarter_desc)
, 0, t.calendar_quarter_desc
, 1, 'TOTAL'
) quarter
, DECODE(GROUPING(t.calendar_quarter_desc) + GROUPING(p.prod_category)
, 0, RANK() OVER (PARTITION BY t.calendar_quarter_desc
ORDER BY SUM(s.amount_sold))
, 1, null
) RANKING
, DECODE(GROUPING(c.channel_class)
, 0, c.channel_class
, 1, '--all--'
) channel
, DECODE(GROUPING(p.prod_category)
, 0, p.prod_category
, 1, 'TOTAL'
) category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
, TO_CHAR(100 * RATIO_TO_REPORT(SUM(s.amount_sold))
OVER (PARTITION BY (TO_CHAR(GROUPING(p.prod_category) ||
t.calendar_quarter_desc)), '990D0') percent
FROM times t
, products p
, channels c
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND c.channel_id = s.channel_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY GROUPING SETS(c.channel_class,
CUBE(t.calendar_quarter_desc, p.prod_category))
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);
```

QUARTER	RANKING	CHANNEL	CATEGORY	REVENUE	PERC
2001-01	1	--all--	Software/Other	\$860,819.81	13.1
2001-01	2	--all--	Electronics	\$1,239,287.71	18.9
2001-01	3	--all--	Hardware	\$1,301,343.45	19.9
2001-01	4	--all--	Photo	\$1,370,706.38	20.9
2001-01	5	--all--	Peripherals	\$1,774,940.09	27.1
2001-01		--all--	TOTAL	\$6,547,097.44	100.0
2001-02	1	--all--	Software/Other	\$872,157.38	12.6
2001-02	2	--all--	Electronics	\$1,144,187.90	16.5
2001-02	3	--all--	Hardware	\$1,557,059.59	22.5
2001-02	4	--all--	Photo	\$1,563,475.51	22.6
2001-02	5	--all--	Peripherals	\$1,785,588.01	25.8
2001-02		--all--	TOTAL	\$6,922,468.39	100.0
2001-03	1	--all--	Software/Other	\$877,630.85	12.2
2001-03	2	--all--	Electronics	\$1,017,536.82	14.1
2001-03	3	--all--	Photo	\$1,607,315.63	22.3
2001-03	4	--all--	Hardware	\$1,651,454.29	22.9
2001-03	5	--all--	Peripherals	\$2,042,061.04	28.4
2001-03		--all--	TOTAL	\$7,195,998.63	100.0
2001-04	1	--all--	Software/Other	\$943,296.36	12.6
2001-04	2	--all--	Hardware	\$1,174,512.68	15.7
2001-04	3	--all--	Electronics	\$1,303,838.52	17.5
2001-04	4	--all--	Photo	\$1,792,131.39	24.0
2001-04	5	--all--	Peripherals	\$2,257,118.57	30.2
2001-04		--all--	TOTAL	\$7,470,897.52	100.0
TOTAL		--all--	Software/Other	\$3,553,904.40	12.6

```

TOTAL    --all-- Electronics  $4,704,850.95 16.7
TOTAL    --all-- Hardware    $5,684,370.01 20.2
TOTAL    --all-- Photo      $6,333,628.91 22.5
TOTAL    Indirect TOTAL      $6,709,496.66 11.9
TOTAL    --all-- Peripherals $7,859,707.71 27.9
TOTAL    Others TOTAL        $8,038,529.96 14.3
TOTAL    Direct TOTAL        $13,388,435.36 23.8
TOTAL    --all-- TOTAL      $28,136,461.98 50.0

```

"Peripherals" was used instead of "Peripherals and Accessories" to save space.

ウィンドウ関数を使用した行間計算の実行方法

ビジネス・インテリジェンスに共通する問題は、特定の結果がどのように他の結果と関連しているかということです。単一問合せでこれを行うには、ウィンドウ関数を使用して単一の文で行間計算を実行できます。

例：行間計算の実行

各四半期の総収入に対するそれぞれの製品カテゴリの寄与率を調べる場合、次の例のように、ウィンドウ関数RATIO_TO_REPORTを使用できます。四半期ごとのRATIO_TO_REPORTから総計を除外するには、GROUPING(p.prod_category)を持つ連結を使用する必要があることに注意してください。

行間計算を行う手順

```

SELECT DECODE(GROUPING(t.calendar_quarter_desc)
, 0, t.calendar_quarter_desc
, 1, 'TOTAL'
) quarter
, DECODE(GROUPING(t.calendar_quarter_desc) + GROUPING(p.prod_category)
, 0, RANK() OVER (PARTITION BY t.calendar_quarter_desc
ORDER BY SUM(s.amount_sold))
, 1, null
) RANKING
, DECODE(GROUPING(c.channel_class)
, 0, c.channel_class
, 1, '--all--'
) channel
, DECODE(GROUPING(p.prod_category)
, 0, p.prod_category
, 1, 'TOTAL'
) category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
, TO_CHAR(100 * RATIO_TO_REPORT(SUM(s.amount_sold))
OVER (PARTITION BY (TO_CHAR(GROUPING(p.prod_category) ||
t.calendar_quarter_desc)), '990D0') percent
, DECODE(GROUPING(t.calendar_quarter_desc) + GROUPING(p.prod_category)
, 0, TO_CHAR(SUM(s.amount_sold) - LAG(SUM(s.amount_sold), 1)
OVER (PARTITION BY p.prod_category
ORDER BY t.calendar_quarter_desc), 'L999G990D00')
, 1, null
) q_q_diff
FROM times t
, products p
, channels c
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND c.channel_id = s.channel_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY GROUPING SETS(c.channel_class,
CUBE(t.calendar_quarter_desc, p.prod_category))
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);

```

QUARTER	RANKING	CHANNEL	CATEGORY	REVENUE	PERC	Q_Q_DIFF
2001-01	1	--all--	Software/Other	\$860,819.81	13.1	
2001-01	2	--all--	Electronics	\$1,239,287.71	18.9	
2001-01	3	--all--	Hardware	\$1,301,343.45	19.9	
2001-01	4	--all--	Photo	\$1,370,706.38	20.9	

```

2001-01 5 --all-- Peripherals $1,774,940.09 27.1
2001-01 --all-- TOTAL $6,547,097.44 100.0
2001-02 1 --all-- Software/Other $872,157.38 12.6 $11,337.57
2001-02 2 --all-- Electronics $1,144,187.90 16.5 -$95,099.81
2001-02 3 --all-- Hardware $1,557,059.59 22.5 $255,716.14
2001-02 4 --all-- Photo $1,563,475.51 22.6 $192,769.13
2001-02 5 --all-- Peripherals $1,785,588.01 25.8 $10,647.92
2001-02 --all-- TOTAL $6,922,468.39 100.0
2001-03 1 --all-- Software/Other $877,630.85 12.2 $5,473.47
2001-03 2 --all-- Electronics $1,017,536.82 14.1 -$126,651.08
2001-03 3 --all-- Photo $1,607,315.63 22.3 $43,840.12
2001-03 4 --all-- Hardware $1,651,454.29 22.9 $94,394.70
2001-03 5 --all-- Peripherals $2,042,061.04 28.4 $256,473.03
2001-03 --all-- TOTAL $7,195,998.63 100.0
2001-04 1 --all-- Software/Other $943,296.36 12.6 $65,665.51
2001-04 2 --all-- Hardware $1,174,512.68 15.7 -$476,941.61
2001-04 3 --all-- Electronics $1,303,838.52 17.5 $286,301.70
2001-04 4 --all-- Photo $1,792,131.39 24.0 $184,815.76
2001-04 5 --all-- Peripherals $2,257,118.57 30.2 $215,057.53
2001-04 --all-- TOTAL $7,470,897.52 100.0
TOTAL --all-- Software/Other $3,553,904.40 12.6
TOTAL --all-- Electronics $4,704,850.95 16.7
TOTAL --all-- Hardware $5,684,370.01 20.2
TOTAL --all-- Photo $6,333,628.91 22.5
TOTAL Indirect TOTAL $6,709,496.66 11.9
TOTAL --all-- Peripherals $7,859,707.71 27.9
TOTAL Others TOTAL $8,038,529.96 14.3
TOTAL Direct TOTAL $13,388,435.36 23.8
TOTAL --all-- TOTAL $28,136,461.98 50.0

```

"Peripherals" was used instead of "Peripherals and Accessories" to save space.

ウィンドウ関数を使用した移動平均の計算方法

ウィンドウ関数を使用して移動集計を作成できます。移動関数は物理行の数に基づくか、論理時間間隔である可能性があります。ウィンドウ関数はPARTITIONキーワードを使用し、パーティションの各行に対してデータのウィンドウの変動を定義できます。このウィンドウでは現行の行の計算を実行するために使用する行の範囲を決定します。ウィンドウ・サイズは行の物理的な数または時間などの論理間隔に基づきます。ウィンドウには開始行および終了行が含まれます。この定義によっては、ウィンドウは片方の行または両方の行へ移動する場合があります。たとえば、累積SUM関数を定義したウィンドウには、パーティションの最初の行に固定された開始行が含まれ、終了行は開始点からパーティションの最後の行までスライドします。反対に、移動平均を定義したウィンドウには開始点と終了点の変動が含まれるため、定数の物理的範囲または論理的範囲が維持されます。

ウィンドウ関数は一般的に、移動および累積バージョンのSUM、AVERAGE、COUNT、MAX、MIN、および他の多くの関数の計算に使用されます。また、問合せのSELECT句、ORDER BY句でのみ使用できます。ウィンドウ関数には、ウィンドウで最初の値を戻すFIRST_VALUE関数、およびウィンドウで最終値を戻すLAST_VALUE関数が含まれます。これらの関数によって自己結合なしに表内の複数行にアクセスできます。

例：移動平均の計算

次の例は、論理的な時間間隔を使用した、各製品の製品収入の7日間の変動平均を取得する問合せを示しています。

変動平均を計算する手順は、次のとおりです。

```

SELECT time_id
, prod_name
, TO_CHAR(revenue, 'L999G990D00') revenue
, TO_CHAR(AVG(revenue) OVER (PARTITION BY prod_name ORDER BY time_id
RANGE INTERVAL '7' DAY PRECEDING), 'L999G990D00') mv_7day_avg
FROM
( SELECT s.time_id, p.prod_name, SUM(s.amount_sold) revenue
FROM products p
, sales s
WHERE p.prod_id = s.prod_id
AND s.time_id BETWEEN TO_DATE('25-JUN-2001', 'dd-MON-yyyy')
AND TO_DATE('16-JUL-2001', 'dd-MON-yyyy')
AND p.prod_name LIKE '%Memory%'
AND p.prod_category = 'Photo'
GROUP BY s.time_id, p.prod_name
)
ORDER BY time_id, prod_name;

```

TIME_ID	PROD_NAME	REVENUE	MV_7DAY_AVG
26-JUN-01	256MB Memory Card	\$560.15	\$560.15
30-JUN-01	256MB Memory Card	\$844.00	\$702.08
02-JUL-01	128MB Memory Card	\$3,283.74	\$3,283.74
02-JUL-01	256MB Memory Card	\$3,903.32	\$1,769.16
03-JUL-01	256MB Memory Card	\$699.37	\$1,501.71
08-JUL-01	128MB Memory Card	\$3,283.74	\$3,283.74
08-JUL-01	256MB Memory Card	\$3,903.32	\$2,835.34
10-JUL-01	256MB Memory Card	\$138.82	\$1,580.50

パーティション外部結合を使用したスパースなデータの処理

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**（売上などの重要なファクトを含むデータ・ウェアハウスの表）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、製品が特定の期間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 Database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、結合のもう一方の表を含む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のUNION操作です。時間ディメンションのみではなく他のディメンションのギャップを埋めるためにも、このタイプの結合を使用してください。

パーティション外部結合を使用するとき

結果セットの欠落した行を埋める場合または時系列計算を処理する場合、パーティション外部結合を使用します。

例：パーティション外部結合の使用

特定の製品の数週間の売上を表示する必要がある場合があります。この例では、フォト・カテゴリのメモリー・カードを使用します。これらの製品は頻繁に売れるものではなく、数週間売れないこともあるためです。簡単に比較するため、次の例のように、パーティション外部結合を使用してデータを稠密にする必要があります。

パーティション外部結合を使用する手順は、次のとおりです。

```
SELECT tim.week_ending_day
, rev.prod_name product
, nvl(SUM(rev.amount_sold),0) revenue
FROM (SELECT p.prod_name, s.time_id, s.amount_sold
      FROM products p
      , sales s
      WHERE s.prod_id = p.prod_id
      AND p.prod_category = 'Photo'
      AND p.prod_name LIKE '%Memory%'
      AND s.time_id BETWEEN TO_DATE('25-JUN-2001','dd-MON-yyyy')
      AND TO_DATE('16-JUL-2001','dd-MON-yyyy')
      ) rev
PARTITION BY (prod_name)
RIGHT OUTER JOIN (SELECT time_id, week_ending_day FROM times
                  WHERE week_ending_day
                  BETWEEN TO_DATE('01-JUL-2001','dd-MON-yyyy')
                  AND TO_DATE('16-JUL-2001','dd-MON-yyyy')
                  ) tim
ON (rev.time_id = tim.time_id)
GROUP BY tim.week_ending_day
, rev.prod_name
ORDER BY tim.week_ending_day
, rev.prod_name;
```

WEEK_ENDI	PRODUCT	REVENUE
01-JUL-01	128MB Memory Card	0
01-JUL-01	256MB Memory Card	1404.15

08-JUL-01	128MB Memory Card	6567.48
08-JUL-01	256MB Memory Card	8506.01
15-JUL-01	128MB Memory Card	0
15-JUL-01	256MB Memory Card	138.82

ビジネス問合せを単純化するWITH句の使用

異なるタイプの結合およびウィンドウ関数を活用し、多くの表にアクセスする問合せは、非常に複雑になります。WITH句を使用すると、問合せの増分的な構築によりこの複雑さを解消できます。複雑な問合せで問合せブロックが複数回発生する場合、SELECT文の同じ問合せブロックを再使用できます。Oracle Databaseは問合せブロックの結果を取得して、ユーザーの一時表領域に結果を格納します。

WITH句を使用するとき

同じ問合せブロックに対して複数の参照が問合せに含まれ、結合および集計がある場合、WITH句を使用します。

例：WITH句の使用

2001年7月締め最初の3週間のフォト・カテゴリのメモリー・カード製品の売上を比較すると想定します。次の問合せはこの期間に売れなかった一部の製品を考慮し、前の週と関連する売上で増加または減少を戻します。最終的に、問合せは特定の週におけるメモリー・カードの売上に対する寄与率の割合を取得します。WITH句の使用のために、問合せの個々のセクションの複雑さは解消されます。

WITH句を使用する手順は、次のとおりです。

```
WITH sales_numbers AS
( SELECT s.prod_id, s.amount_sold, t.week_ending_day
  FROM sales s
    , times t
    , products p
 WHERE s.time_id = t.time_id
   AND s.prod_id = p.prod_id
   AND p.prod_category = 'Photo'
   AND p.prod_name LIKE '%Memory%'
   AND t.week_ending_day BETWEEN TO_DATE('01-JUL-2001', 'dd-MON-yyyy')
                        AND TO_DATE('16-JUL-2001', 'dd-MON-yyyy')
)
, product_revenue AS
( SELECT p.prod_name product, s.week_ending_day, SUM(s.amount_sold) revenue
  FROM products p
    LEFT OUTER JOIN (SELECT prod_id, amount_sold, week_ending_day
                    FROM sales_numbers) s
    ON (s.prod_id = p.prod_id)
 WHERE p.prod_category = 'Photo'
   AND p.prod_name LIKE '%Memory%'
 GROUP BY p.prod_name, s.week_ending_day
)
, weeks AS
( SELECT distinct week_ending_day week FROM times WHERE week_ending_day
   BETWEEN TO_DATE('01-JUL-2001', 'dd-MON-yyyy')
   AND TO_DATE('16-JUL-2001', 'dd-MON-yyyy')
)
, complete_product_revenue AS
( SELECT w.week, pr.product, nvl(pr.revenue,0) revenue
  FROM product_revenue pr
    PARTITION BY (product)
    RIGHT OUTER JOIN weeks w
    ON (w.week = pr.week_ending_day)
)
SELECT week
, product
, TO_CHAR(revenue, 'L999G990D00') revenue
, TO_CHAR(revenue - lag(revenue,1) OVER (PARTITION BY product
  ORDER BY week), 'L999G990D00') w_w_diff
, TO_CHAR(100 * RATIO_TO_REPORT(revenue) OVER (PARTITION BY week), '990D0') percentage
FROM complete_product_revenue
ORDER BY week, product;
```

WEEK	PRODUCT	REVENUE	W_W_DIFF	PERCENT
-----	-----	-----	-----	-----

01-JUL-01	128MB Memory Card	\$0.00		0.0
01-JUL-01	256MB Memory Card	\$1,404.15		100.0
01-JUL-01	64MB Memory Card	\$0.00		0.0
08-JUL-01	128MB Memory Card	\$6,567.48	\$6,567.48	43.6
08-JUL-01	256MB Memory Card	\$8,506.01	\$7,101.86	56.4
08-JUL-01	64MB Memory Card	\$0.00	\$0.00	0.0
15-JUL-01	128MB Memory Card	\$0.00	-\$6,567.48	0.0
15-JUL-01	256MB Memory Card	\$138.82	-\$8,367.19	100.0
15-JUL-01	64MB Memory Card	\$0.00	\$0.00	0.0

第IV部

データ・ウェアハウスの管理

ここでは、データ・ウェアハウスのメンテナンス方法について説明します。内容は次のとおりです。

- [第8章「データ・ウェアハウスのリフレッシュ」](#)
- [第9章「データ・ウェアハウスの操作の最適化」](#)
- [第10章「パフォーマンス・ボトルネックの排除」](#)
- [第11章「データ・ウェアハウスのバックアップおよびリカバリ」](#)
- [第12章「データ・ウェアハウスのセキュリティ」](#)

8 データ・ウェアハウスのリフレッシュ

この章の内容は次のとおりです。

- [データ・ウェアハウスのリフレッシュについて](#)
- [ローリング・ウィンドウを使用したデータのオフロード](#)

データ・ウェアハウスのリフレッシュについて

データ・ウェアハウスを定期的に更新し、現在の情報を導出します。データの更新プロセスは、**リフレッシュ処理**と呼ばれます。

抽出、変換およびロード (ETL) は、スケジュールに基づいて実行され、元のソース・システムへの変更が反映されます。この手順の間、新規の更新されたデータを本番データのウェアハウス・スキーマに物理的に挿入し、ユーザーがこのデータを使用できるようにするために、他のすべての必要な手順 (索引の作成、制限の検証、バックアップ・コピーの作成) に従います。このデータをすべてデータ・ウェアハウスにロードすると、マテリアライズド・ビューを更新して最新のデータを反映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ニング・スキームは、データ・ウェアハウスのロード・プロセスでの操作のリフレッシュの効率性を決定する際に重要です。ロード・プロセスは、データ・ウェアハウス表のパーティショニング・スキームの選択時に考慮されます。

多くのデータ・ウェアハウスでは、定期的に新規データがロードされます。たとえば、日、週または月ごとに新規データがデータ・ウェアハウスに取り込まれます。週末または月末にロードされたデータは通常、その週またはその月のトランザクションに対応します。この一般的なシナリオでは、データ・ウェアハウスは時間ごとにロードされます。ここで、データ・ウェアハウス表をデータ列上でパーティション化することをお勧めします。データ・ウェアハウスの例として、新規データが毎月、売上表にロードされると想定します。さらに、sales表は毎月、パーティション化されるものとし、これらの手順では、ロード・プロセスによる新しい月 (2006年第1四半期) のsales表へのデータの追加処理方法を示します。

例: データ・ウェアハウスのリフレッシュ

多くの問合せでは、日付で問合せが制限されているproducts、customersおよびsales表の一部の列が要求されます。マテリアライズド・ビューを利用すると、3つの表に対する問合せの大部分が高速化されます。マテリアライズド・ビューが作成される事前作成表を使用して、sales表のパーティション計画を持つ動機のマテリアライズド・ビューのパーティション計画を選択します。

次に、マテリアライズド・ビューのリフレッシュに関する例を示します。ここでは、パーティション交換ロード操作を使用します。この例はshスキーマのsales表に基づいています。

マテリアライズド・ビューをリフレッシュする手順

1. マテリアライズド・ビューに基づいた表を作成します。

```
CREATE TABLE sales_prod_cust_mv
( time_id DATE
, prod_id NUMBER
, prod_name VARCHAR2(50)
, cust_id NUMBER
, cust_first_name VARCHAR2(20)
, cust_last_name VARCHAR2(40)
, amount_sold NUMBER
, quantity_sold NUMBER
)
PARTITION BY RANGE (time_id)
( PARTITION p1999 VALUES LESS THAN (TO_DATE('01-JAN-2000','DD-MON-YYYY'))
, PARTITION p2000 VALUES LESS THAN (TO_DATE('01-JAN-2001','DD-MON-YYYY'))
, PARTITION p2001h1 VALUES LESS THAN (TO_DATE('01-JUL-2001','DD-MON-YYYY'))
, PARTITION p2001h2 VALUES LESS THAN (TO_DATE('01-JAN-2002','DD-MON-YYYY'))
, PARTITION p2001q1 VALUES LESS THAN (TO_DATE('01-APR-2002','DD-MON-YYYY'))
, PARTITION p2002q2 VALUES LESS THAN (TO_DATE('01-JUL-2002','DD-MON-YYYY'))
, PARTITION p2002q3 VALUES LESS THAN (TO_DATE('01-OCT-2002','DD-MON-YYYY'))
, PARTITION p2002q4 VALUES LESS THAN (TO_DATE('01-JAN-2003','DD-MON-YYYY'))
)
```

```

, PARTITION p2003q1 VALUES LESS THAN (TO_DATE('01-APR-2003', 'DD-MON-YYYY'))
, PARTITION p2003q2 VALUES LESS THAN (TO_DATE('01-JUL-2003', 'DD-MON-YYYY'))
, PARTITION p2003q3 VALUES LESS THAN (TO_DATE('01-OCT-2003', 'DD-MON-YYYY'))
, PARTITION p2003q4 VALUES LESS THAN (TO_DATE('01-JAN-2004', 'DD-MON-YYYY'))
, PARTITION p2004q1 VALUES LESS THAN (TO_DATE('01-APR-2004', 'DD-MON-YYYY'))
, PARTITION p2004q2 VALUES LESS THAN (TO_DATE('01-JUL-2004', 'DD-MON-YYYY'))
, PARTITION p2004q3 VALUES LESS THAN (TO_DATE('01-OCT-2004', 'DD-MON-YYYY'))
, PARTITION p2004q4 VALUES LESS THAN (TO_DATE('01-JAN-2005', 'DD-MON-YYYY'))
, PARTITION p2005q1 VALUES LESS THAN (TO_DATE('01-APR-2005', 'DD-MON-YYYY'))
, PARTITION p2005q2 VALUES LESS THAN (TO_DATE('01-JUL-2005', 'DD-MON-YYYY'))
, PARTITION p2005q3 VALUES LESS THAN (TO_DATE('01-OCT-2005', 'DD-MON-YYYY'))
, PARTITION p2005q4 VALUES LESS THAN (TO_DATE('01-JAN-2006', 'DD-MON-YYYY'))
, PARTITION p2006q1 VALUES LESS THAN (TO_DATE('01-APR-2006', 'DD-MON-YYYY'))
) PARALLEL COMPRESS;

```

2. sales表の初期表をロードします。

```

ALTER SESSION ENABLE PARALLEL DML;
INSERT /*+ PARALLEL smv */ INTO sales_prod_cust_mv smv
SELECT /*+ PARALLEL s PARALLEL c */ s.time_id
, s.prod_id
, p.prod_name
, s.cust_id
, cust_first_name
, c.cust_last_name
, SUM(s.amount_sold)
, SUM(s.quantity_sold)
FROM sales s
, products p
, customers c
WHERE s.cust_id = c.cust_id
AND s.prod_id = p.prod_id
GROUP BY s.time_id
, s.prod_id
, p.prod_name
, s.cust_id
, c.cust_first_name
, c.cust_last_name;
COMMIT;

```

3. マテリアライズド・ビューを作成します。

```

CREATE MATERIALIZED VIEW sales_prod_cust_mv
ON PREBUILT TABLE
ENABLE QUERY REWRITE
AS SELECT s.time_id
, s.prod_id
, p.prod_name
, s.cust_id
, c.cust_first_name
, c.cust_last_name
, SUM(s.amount_sold) amount_sold
, SUM(s.quantity_sold) quantity_sold
FROM sales s
, products p
, customers c
WHERE s.cust_id = c.cust_id
AND s.prod_id = p.prod_id
GROUP BY s.time_id
, s.prod_id
, p.prod_name
, s.cust_id
, c.cust_first_name
, c.cust_last_name;

```

4. 新規パーティションで交換される個別の表をロードします。

```

CREATE TABLE sales_q1_2006 PARALLEL COMPRESS
AS SELECT * FROM sales
WHERE 0 = 1;

```

```

/* This would be the regular ETL job */

ALTER SESSION ENABLE PARALLEL DML;

INSERT /* PARALLEL qs */ INTO sales_q1_2006 qs
SELECT /* PARALLEL s */ prod_id
, cust_id
, add_months(time_id,3)
, channel_id
, promo_id
, quantity_sold
, amount_sold
FROM sales PARTITION(sales_q4_2005) s;

COMMIT;

CREATE BITMAP INDEX bmp_indx_prod_id ON sales_q1_2006 (prod_id);
CREATE BITMAP INDEX bmp_indx_cust_id ON sales_q1_2006 (cust_id);
CREATE BITMAP INDEX bmp_indx_time_id ON sales_q1_2006 (time_id);
CREATE BITMAP INDEX bmp_indx_channel_id ON sales_q1_2006 (channel_id);
CREATE BITMAP INDEX bmp_indx_promo_id ON sales_q1_2006 (promo_id);

ALTER TABLE sales_q1_2006 ADD CONSTRAINT sales_q_prod_fk
FOREIGN KEY (prod_id) REFERENCES products(prod_id) ENABLE NOVALIDATE;

ALTER TABLE sales_q1_2006 ADD CONSTRAINT sales_q_cust_fk
FOREIGN KEY (cust_id) REFERENCES customers(cust_id) ENABLE NOVALIDATE;

ALTER TABLE sales_q1_2006 ADD CONSTRAINT sales_q_time_fk
FOREIGN KEY (time_id) REFERENCES times(time_id) ENABLE NOVALIDATE;

ALTER TABLE sales_q1_2006 ADD CONSTRAINT sales_q_channel_fk
FOREIGN KEY (channel_id) REFERENCES channels(channel_id) ENABLE NOVALIDATE;

ALTER TABLE sales_q1_2006 ADD CONSTRAINT sales_q_promo_fk
FOREIGN KEY (promo_id) REFERENCES promotions(promo_id) ENABLE NOVALIDATE;

BEGIN
  DBMS_STATS.GATHER_TABLE_STATS('SH','SALES_Q1_2006');
END;
/

```

5. マテリアライズド・ビューのパーティションで交換される個別の表を作成およびロードします。

```

CREATE TABLE sales_mv_q1_2006 PARALLEL COMPRESS
AS SELECT * FROM sales_prod_cust_mv
WHERE l = 0;

ALTER SESSION ENABLE PARALLEL DML;

INSERT /*+ PARALLEL smv */ INTO sales_mv_q1_2006 smv
SELECT /*+ PARALLEL s PARALLEL c */ s.time_id
, s.prod_id
, p.prod_name
, s.cust_id
, c.cust_first_name
, c.cust_last_name
, SUM(s.amount_sold)
, SUM(s.quantity_sold)
FROM sales_q1_2006 s
, products p
, customers c
WHERE s.cust_id = c.cust_id
AND s.prod_id = p.prod_id
GROUP BY s.time_id
, s.prod_id
, p.prod_name
, s.cust_id
, c.cust_first_name
, c.cust_last_name;

COMMIT;

```

6. 統計を収集します。

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS('SH','SALES_MV_Q1_2006');
END;
```

- パーティションを交換します。

```
ALTER TABLE sales
EXCHANGE PARTITION sales_q1_2006
WITH TABLE sales_q1_2006
INCLUDING INDEXES WITHOUT VALIDATION;
```

```
ALTER TABLE sales_prod_cust_mv
EXCHANGE PARTITION p2006q1
WITH TABLE sales_mv_q1_2006
INCLUDING INDEXES WITHOUT VALIDATION;
```

- マテリアライズド・ビューが再度フレッシュになったことがデータベースに通知されます。

```
ALTER MATERIALIZED VIEW sales_prod_cust_mv CONSIDER FRESH;
```

このシナリオでは、事前作成表を使用し、この制約はRELY制約ではないため、問合せのリライト機能はquery_rewrite_integrityをSTALE_TOLERATEDに設定した場合にのみ有効です。

ローリング・ウィンドウを使用したデータのオフロード

データを削除したりアーカイブする際にローリング・ウィンドウを使用すると特に効果的です。たとえば、最新の36か月の売上データをデータ・ウェアハウスに格納する場合にローリング・ウィンドウを使用します。新規パーティションは、新しい月のsales表にそれぞれ追加され、古いパーティションはsales表から削除される場合があります。この方法では、ウェアハウスに常に36か月分のデータが保持されます。

例：ローリング・ウィンドウの使用

次にshスキーマのsales表のローリング・ウィンドウの例を示します。

ローリング・ウィンドウを使用する手順

- 2005年12月の売上を追加します。

```
ALTER TABLE sales
ADD PARTITION sales_12_2005 VALUES LESS THAN ('01-JAN-2006');
```

既存の索引を再作成する必要があります。

- 1999年のパーティションを削除します。

```
ALTER TABLE sales
DROP PARTITION sales_1999;
```

9 データ・ウェアハウスの操作の最適化

この項では、データ・ウェアハウスのパフォーマンスを最適化する方法を説明します。内容は次のとおりです。

- [システムのオーバーロードの回避](#)
- [索引およびマテリアライズド・ビューの使用の最適化](#)
- [記憶域要件の最適化](#)

システムのオーバーロードの回避

この項では、システムのオーバーロードを識別して回避する方法を説明します。一般的に、『*Oracle Database 2日*でパフォーマンス・チューニング・ガイド』の説明に従って、自動診断機能である自動データベース診断モニター (ADDM) を使用してデータベースでのパフォーマンスの問題を識別する必要があります。この項では、システムにおけるパフォーマンスの問題を回避する別の方法を説明します。また次の項が含まれます。

- [システム・パフォーマンスの監視](#)
- [データベース・リソース・マネージャの使用](#)

システム・パフォーマンスの監視

この項では、重要なメトリックを定期的に監視することによりシステムのオーバーロードを回避する方法の詳細を提供します。Oracle Enterprise Managerの「データベース・パフォーマンス」ページを介してこれらのメトリックを監視できます。この項の内容は次のとおりです。

- [パラレル実行パフォーマンスの監視](#)
- [I/Oの監視](#)

パラレル実行パフォーマンスの監視

この項では、パラレル実行パフォーマンスを監視する方法を説明します。多くのパラレル文がダウングレードされているとします。これは、パフォーマンスの問題を示します。想定よりも低い並列度で文が実行されると大幅に時間がかかり、文がダウングレードされたかどうかによって実行時間が異なります。パラレル文がダウングレードする原因には次のようなものが考えられます。

- 最初の並列度が高く、低くする必要がある場合。
- 使用可能なパラレル・サーバーが十分になく、システムがオーバーロードしている場合。

パラレル実行を監視する手順

1. データベースのホームページで「パフォーマンス」をクリックします。

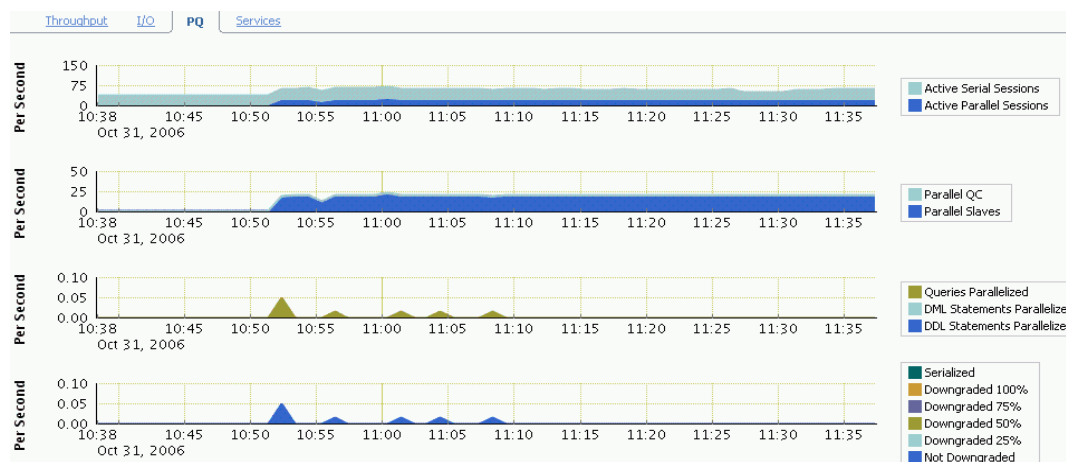
「パフォーマンス」ページが表示されます。

2. ページを下へスクロールします。リンクのリストの下で、「PQ」をクリックします。

「PQ」ページが表示されます。次の項目に対してパラレル問合せのパフォーマンス特性が表示されます。

- パラレル・セッション
- パラレル・スレーブ
- DMLおよびDDLのパラレル化
- シリアライズおよび文のダウングレード

図9-1 パラレル実行の監視



「図9-1 パラレル実行の監視」の説明

I/Oの監視

この項では、I/Oパフォーマンスを監視する方法を説明します。システムのスループットがシステム構成に基づいて予想していたものより大幅に低く（第2章「[データ・ウェアハウス・システムの設定](#)」を参照）、ユーザーがパフォーマンスに関して不満がある場合は、パフォーマンスに問題があると考えられます。適切に構成されたシステムで通常のデータ・ウェアハウス・ワークロードを処理する場合、単一のI/Oブロックに対して、おおむね良好なI/Oと、待ち時間が比較的少ない（30分以下）が予想されます。

I/Oパフォーマンス監視する手順

1. データベースのホームページで「パフォーマンス」をクリックします。

「パフォーマンス」ページが表示されます。

2. ページを下へスクロールします。リンクのリストの下で、「I/O」をクリックします。

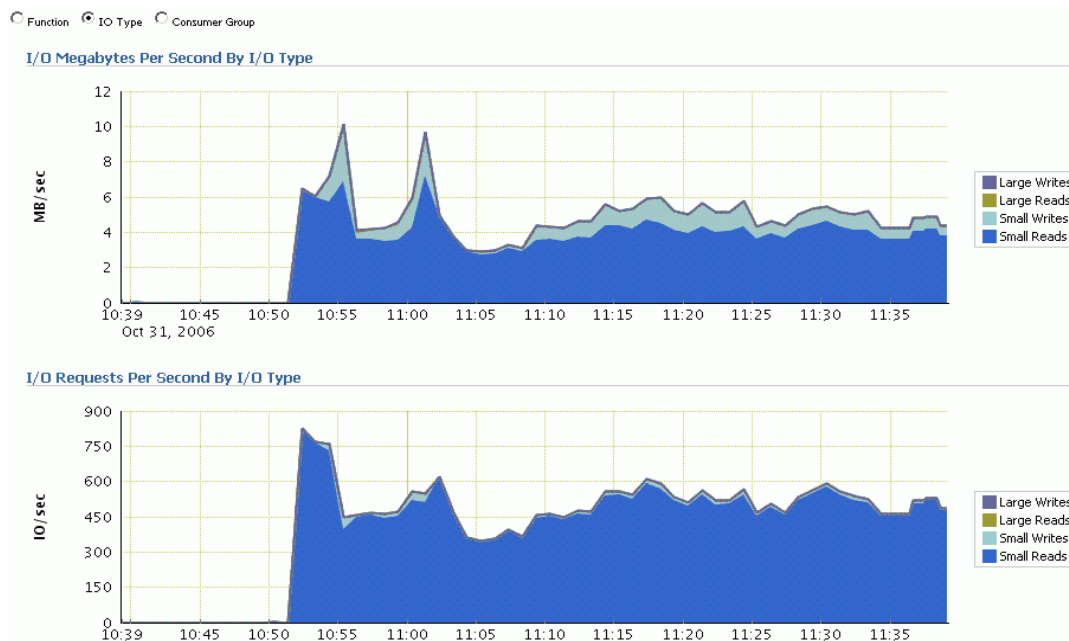
I/Oページが表示され、「ファンクションごとのI/O MB/秒」と「ファンクションごとのI/Oリクエスト数/秒」が表示されます。

3. 読取り/書込み操作に関する詳細は、「I/Oタイプ」を選択します。

次の項目に対してI/O詳細が表示されます。

- 大規模の書込み
- 大規模の読取り
- 小規模の書込み
- 小規模の読取り

図9-2 I/Oの監視



「図9-2 I/Oの監視」の説明

データベース・リソース・マネージャの使用

データベース・リソース・マネージャには、Oracleシステム内の作業に優先度を設定する機能があります。たとえば、優先度の高いジョブのあるユーザーは、オンライン作業のレスポンス時間を最短に抑えるためにリソースを取得する一方、バッチ・ジョブやレポートなどの優先度の低いジョブのあるユーザーに対しては、レスポンス時間が遅くなります。このように優先度を割り当てることにより、リソースをより細かく制御できます。また、コンシューマ・グループに対して、コンシューマ・グループの自動切替え、最大アクティブ・セッションの制御、問合せ実行時間の見積りおよびUNDOプール割当て制限などの機能があります。

各コンシューマ・グループの同時にアクティブなセッションの最大数を指定できます。この制限に到達した場合、データベース・リソース・マネージャにより、すべての後続のリクエストが並べられ、既存のアクティブ・セッションが完了した後にのみそれらが実行されます。

データベース・リソース・マネージャはOracle Databaseの一部で、データベース内部の異なるプロセスを区別できます。結果として、データベース・リソース・マネージャにより、データベース内部で実行されている個別の操作に優先度を割り当てることができます。

データベース・リソース・マネージャを使用して、次のことを実行できます。

- システムにかかる負荷およびユーザー数に関係なく、処理リソースの最低量をユーザーに保証します。
- CPU時間の割合を異なるユーザーおよびアプリケーションに割り当てることにより、使用可能な処理リソースを配分します。データ・ウェアハウスでは、バッチ・ジョブよりもリレーショナル・オンライン分析処理 (ROLAP) アプリケーションに対する割当ての方が多くなります。
- 管理者が定義した基準に基づいて、ユーザーを1つのグループから他のグループに自動的に切り替えます。あるユーザー・グループのメンバーが、指定時間より長くかかるセッションを作成した場合、そのセッションはリソース要件の異なる他のユーザー・グループに自動的に切り替えられます。
- 特定のリソース割当て方法を使用するようにインスタンスを構成します。この方法は、インスタンスを停止して再起動する必要なく、日中設定から夜間設定などのように動的に変更できます。

索引およびマテリアライズド・ビューの使用の最適化

索引およびマテリアライズド・ビューを適切に使用することにより、データ・ウェアハウスのパフォーマンスを向上させることができます。SQLアクセス・アドバイザーの主な利点は、現行のワークロードを推奨事項の基準として使用できることです。

例: SQLアクセス・アドバイザーを使用した索引およびマテリアライズド・ビューの使用の最適化

この例では、特定の索引およびマテリアライズド・ビューを利用しているシステムで実行されているワークロードを所有すると仮定します。

索引およびマテリアライズド・ビューを最適化する手順

1. 「アドバイザー・セントラル」ページから、「SQLアドバイザー」をクリックします。

「アドバイザー」ページが表示されます。
2. 「アドバイザー」ページから、「SQLアクセス・アドバイザー」をクリックします。

「SQLアクセス・アドバイザー」ページが表示されます。
3. 「デフォルト・オプションを使用」を選択して、「続行」をクリックします。

「ワークロード・ソース」ページが表示されます。
4. 「既存のSQLチューニング・セットを使用します。」で、ワークロード・ソースを選択します。に移動し、「選択」をクリックします。次に「次へ」をクリックします。

「推奨オプション」ページが表示されます。
5. 「索引」、「マテリアライズド・ビュー」および「包括モード」を選択します。「次へ」をクリックします。

「スケジュール」ページが表示されます。
6. 「発行」をクリックします。

「推奨」ページが表示されます。
7. 「名前」フィールドに名前を入力し、開始する必要がある場合は「即時」を選択します。「次へ」をクリックします。

「確認」ページが表示されます。
8. 「発行」をクリックします。

「確認」ページが表示されます。
9. タスク名を選択し、「結果の表示」をクリックします。

「推奨」、「SQL文」または「詳細」で追加情報を表示できます。

記憶域要件の最適化

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。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。パーティション表では、パーティションの一部を圧縮するかまたはすべてのパーティションを圧縮するかを選択できます。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も適用されます。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。

データ圧縮による記憶域の改善

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。ただし、範囲の権限およびデータを論理的に個別のパーティションに区切るコンポジット・パーティション化のため、パーティション表などは、主に読取り専用のデータ（パーティション）の圧縮する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、追加の記憶域の使用量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。

一部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、完全に未圧縮のパーティション表と同じです。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手順

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。表の圧縮属性および他のすべてのパーティション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));
```

10 パフォーマンス・ボトルネックの排除

この項では、パフォーマンスの問題を識別して削減する方法を説明します。内容は次のとおりです。

- [SQLの効率的な実行の検証](#)
- [リソース使用量の最小化によるパフォーマンスの向上](#)
- [最適なリソースの使用](#)

SQLの効率的な実行の検証

システムの正常な実行を保証するために重要なことは、パフォーマンスの問題を排除することです。この項では、これらのボトルネックを検索して排除するいくつかの方法を説明します。内容は次のとおりです。

- [オプティマイザ統計の分析](#)
- [実行計画の分析](#)
- [ヒントを使用したデータ・ウェアハウスのパフォーマンスの向上](#)
- [アドバイザを使用したSQLパフォーマンスの検証](#)

オプティマイザ統計の分析

オプティマイザ統計は、データベース内のデータベースおよびオブジェクトについてより詳細に説明するデータが収集されたものです。これらの統計はデータ・ディクショナリに格納され、問合せオプティマイザにより使用されて各SQL文に最適な実行計画を選択します。オプティマイザ統計には、次が含まれます。

- 表統計（行数、ブロックおよび平均の行長さ）
- 列統計（列内の固有の値の数、列内のNULL値の数およびデータ配分）
- 索引統計（リーフ・ブロックの数、レベルおよびクラスタリング・ファクタ）
- システム統計（CPUとI/Oのパフォーマンスおよび使用率）

オプティマイザ統計は、データ・ディクショナリに格納されます。オプティマイザ統計は、次に類似するデータ・ディクショナリ・ビューを使用して参照できます。

```
SELECT * FROM DBA_SCHEDULER_JOBS WHERE JOB_NAME = 'GATHER_STATS_JOB';
```

データベース内のオブジェクトは常に変更されるので、これらのデータベース・オブジェクトを正確に示すために統計を定期的に更新する必要があります。統計は、Oracle Databaseにより自動的に維持されるか、またはDBMS_STATSパッケージを使用して手動でオプティマイザ統計を維持することもできます。

実行計画の分析

SQL文を実行するには、Oracle Databaseでは多数のステップを実行する必要があります。これらの各ステップでは、データベースから物理的にデータ行を取得するか、文を発行するユーザー用になんらかの方法でそれらのデータを用意しておきます。Oracle Databaseが文の実行に使用するステップの組合せは、実行計画と呼ばれます。実行計画には、文がアクセスする各表のアクセス・パスおよび適切な結合方法を使用した表の

順序（結合順序）が含まれます。

EXPLAIN PLAN文を使用して、オプティマイザがSQL文用に選択した実行計画を調査できます。その文が発行されると、オプティマイザにより実行計画が選択され、計画を説明するデータがデータベース表に挿入されます。単純にEXPLAIN PLAN文を発行し出力表を問い合わせることで実行できます。

EXPLAIN PLAN文を使用する一般的なガイドラインは次のとおりです。

- SQLスクリプトであるUTLXPLAN.SQLを使用して、スキーマ内にあるPLAN_TABLEというサンプルの出力表を作成します。
- SQL文より前にEXPLAIN PLAN FOR句を含めます。
- EXPLAIN PLAN文を発行した後、Oracle Databaseにより提供されたスクリプトの1つまたはパッケージの1つを使用して最新の計画の表の出力を表示します。
- EXPLAIN PLAN出力内の実行順序は、最も右側にインデントされた行から始まります。2つの行が等しくインデントされている場合は、通常、上位の行が最初に実行されます。

例：実行計画の出力の分析

次の文では、2つのEXPLAIN PLAN文の出力を示し、1つは動的プルーニングを使用し、もう1つは静的プルーニングを使用します。

EXPLAIN PLANの出力を分析する手順

```
EXPLAIN PLAN FOR
SELECT p.prod_name
, c.channel_desc
, SUM(s.amount_sold) revenue
FROM products p
, channels c
, sales s
WHERE s.prod_id = p.prod_id
AND s.channel_id = c.channel_id
AND s.time_id BETWEEN '01-12-2001' AND '31-12-2001'
GROUP BY p.prod_name
, c.channel_desc;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

WITHOUT TO_DATE

```
-----
| Id | Operation                | Name | Rows | Bytes | Cost | Time | Pstart | Pstop |
|    |                          |      |      |      |      |      |        |       | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT          |      | 252 | 15876 | 305(1) | 00:00:06 | | |
| 1 | HASH GROUP BY             |      | 252 | 15876 | 305(1) | 00:00:06 | | |
| *2 | FILTER                    |      |    |    |    |    |    |    |
| *3 | HASH JOIN                  |      | 2255 | 138K | 304(1) | 00:00:06 | | |
| 4 | TABLE ACCESS FULL        | PRODUCTS | 72 | 2160 | 2(0) | 00:00:01 | | |
| 5 | MERGE JOIN                 |      | 2286 | 75438 | 302(1) | 00:00:06 | | |
| 6 | TABLE ACCESS BY INDEX ROWID | CHANNELS | 5 | 65 | 2(0) | 00:00:01 | | |
| 7 | INDEX FULL SCAN           | CHANNELS_PK | 5 | 1(0) | 00:00:01 | | |
| *8 | SORT JOIN                  |      | 2286 | 45720 | 299(1) | 00:00:06 | | |
| 9 | PARTITION RANGE ITERATOR   |      | 2286 | 45720 | 298(0) | 00:00:06 | KEY | KEY |
| 10 | TABLE ACCESS BY LOCAL INDEX ROWID | SALES | 2286 | 45720 | 298(0) | 00:00:06 | KEY | KEY |
| 11 | BITMAP CONVERSION TO ROWIDS |      |    |    |    |    |    |    |
| *12 | BITMAP INDEX RANGE SCAN    | SALES_TIME_BIX |    |    |    |    |    |    | KEY | KEY |
-----
```

Predicate Information (identified by operation id):

```
-----
2 - filter(TO_DATE('01-12-2001') <= TO_DATE('31-12-2001'))
3 - access("S"."PROD_ID"="P"."PROD_ID")
8 - access("S"."CHANNEL_ID"="C"."CHANNEL_ID")
   filter("S"."CHANNEL_ID"="C"."CHANNEL_ID")
12 - access("S"."TIME_ID" >= '01-12-2001' AND "S"."TIME_ID" <= '31-12-2001')
```

Note the values of KEY KEY for Pstart and Pstop.

WITH TO_DATE

Id	Operation	Name	Rows	Bytes	Cost(%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		252	15876	31 (20)	00:00:01		
1	HASH GROUP BY		252	15876	31 (20)	00:00:01		
*2	HASH JOIN		21717	1336K	28 (11)	00:00:01		
3	TABLE ACCESS FULL	PRODUCTS	72	2160	2 (0)	00:00:01		
*4	HASH JOIN		21717	699K	26 (12)	00:00:01		
5	TABLE ACCESS FULL	CHANNELS	5	65	3 (0)	00:00:01		
6	PARTITION RANGE SINGLE		21717	424K	22 (10)	00:00:01	20	20
*7	TABLE ACCESS FULL	SALES	21717	424K	22 (10)	00:00:01	20	20

Predicate Information (identified by operation id):

```
2 - access("S"."PROD_ID"="P"."PROD_ID")
4 - access("S"."CHANNEL_ID"="C"."CHANNEL_ID")
7 - filter("S"."TIME_ID">=TO_DATE('2001-12-01 00:00:00', 'yyyy-mm-dd hh24:mi:ss') AND
          "S"."TIME_ID"<=TO_DATE('2001-12-31 00:00:00', 'yyyy-mm-dd hh24:mi:ss'))
```

Note the values of 20 20 for Pstart and Pstop.

最初の実行計画は、PstartおよびPstopそれぞれのKEY値を使用して動的ブルーニングを表示します。動的ブルーニングは、実行時間にデータベースによりアクセスするパーティションを特定する必要があることを意味します。静的ブルーニングの場合は、解析時間にデータベースにより、より効果的な実行のためにアクセスするパーティションを把握します。

明示的な日付変換を使用して実行計画を頻繁に改善できます。明示的な日付変換の使用は、最適なパーティション・ブルーニングおよび索引の使用のためのベスト・プラクティスです。

ヒントを使用したデータ・ウェアハウスのパフォーマンスの向上

ヒントによって、通常はオブティマイザによって行われる決定を行うことができます。オブティマイザによって管理されていないデータについての情報をアプリケーション開発者として保持している場合があります。ヒントによってメカニズムが提供され、特定の基準に基づく問合せ実行プランを選択するためにオブティマイザに通知されます。

たとえば、ユーザーが特定の問合せに対して特定の索引が選択可能であることを把握している場合があります。この情報に基づいて、オブティマイザが選択するより、より効率的な実行プランが選択できる場合があります。このような場合、ヒントを使用して、オブティマイザに通知し、最適な実行プランを使用します。

デフォルトでは、Oracle Warehouse Builderには、一般的なデータ・ロードを最適化するヒントが含まれます。

参照

『Oracle Warehouse Builder Sources and Targets Guide』

例：ヒントを使用したデータ・ウェアハウスのパフォーマンスの向上

システムがアイドル状態である間に昨年の売上表のサマリーを迅速に実行することを想定します。この場合は、次の文を発行できます。

データ・ウェアハウスのパフォーマンスを向上させるためにヒントを使用する手順

```
SELECT /*+ PARALLEL(s,16) */ SUM(amount_sold)
FROM sales s
WHERE s.time_id BETWEEN TO_DATE('01-JAN-2005', 'DD-MON-YYYY')
AND TO_DATE('31-DEC-2005', 'DD-MON-YYYY');
```

データ・ウェアハウスでヒントを使用するもう1つの一般的な方法は、圧縮の使用によるレコードの効率的なロードを保証することです。次のSQLに示すように、APPENDヒントを使用します。

...

```
INSERT /* +APPEND */ INTO my_materialized_view
...
```

アドバイザを使用したSQLパフォーマンスの検証

SQLチューニング・アドバイザおよびSQLアクセス・アドバイザを使用すると、特定のSQL文またはSQL文のセットを調べるアドバイザ・モードで問合せ最適化を起動でき、SQL文の効率を向上させる推奨事項が提供されます。SQLチューニング・アドバイザおよびSQLアクセス・アドバイザはSQLプロファイルの作成、SQL文の再構築、付加的な索引またはマテリアライズド・ビューの作成、および最適化統計のリフレッシュなどの様々なタイプの推奨事項を作成できます。さらに、Oracle Enterprise Managerを使用すると、数回のマウス・クリックでこれらの推奨事項を受け入れ、実装できます。

SQLアクセス・アドバイザは、主に索引およびマテリアライズド・ビューの追加および削除などのスキーマ変更の推奨事項を作成する場合に使用します。また、パーティション計画も推奨します。SQLチューニング・アドバイザは、SQLプロファイルの作成、SQL文の再構築などの他のタイプの推奨事項の作成に使用します。新しい索引を作成してパフォーマンスが大幅に向上できる場合、SQLチューニング・アドバイザは索引の作成を推奨する可能性があります。ただし、これらの推奨事項は、典型的なSQL文のセットを含んだSQLワークロードを使用してSQLアクセス・アドバイザを実行し、検証する必要があります。

例: SQLチューニング・アドバイザを使用したSQLパフォーマンスの検証

SQLチューニング・アドバイザを使用して、単一または複数のSQL文をチューニングできます。複数のSQL文をチューニングする場合、SQLチューニング・アドバイザはSQL文間の相互依存を認識しないことに注意してください。かわりに、多数のSQL文に対してSQLチューニング・アドバイザを実行すると有効です。

SQLチューニング・アドバイザを実行してSQLパフォーマンスを検証する手順は、次のとおりです。

1. 「セントラル・アドバイザ」ページに移動して、「SQLアドバイザ」をクリックします。

「SQLアドバイザ」ページが表示されます。

2. 「SQLチューニング・アドバイザのスケジュール」をクリックします。

「SQLチューニング・アドバイザのスケジュール」ページが表示されます。推奨される名前は「名前」フィールドに表示され、変更が可能です。その後「包括」を選択して実行する包括的な分析を取得します。「スケジュール」の「即時」を選択します。適切なSQLチューニング・セットを選択した後、「OK」をクリックします。

3. 「処理中」ページが表示されます。その後、「推奨」ページにパフォーマンス向上の推奨事項が表示されます。「推奨の表示」をクリックします。

「推奨」ページが表示されます。

4. 推奨事項は、「実装」をクリックして実装できる索引を作成することです。または、必要に応じてSQLアクセス・アドバイザも実行できます。

リソース使用量の最小化によるパフォーマンスの向上

リソースの使用量を最小化できるので、次の機能を使用してデータ・ウェアハウスのパフォーマンスを向上させることができます。

- [パフォーマンスの向上: パーティション化](#)
- [パフォーマンスの向上: 問合せのリライトおよびマテリアライズド・ビュー](#)
- [パフォーマンスの向上: 索引](#)
- [パフォーマンスの向上: 圧縮](#)

パフォーマンスの向上：パーティション化

データ・ウェアハウスにはサイズの大きな表が含まれていることが多く、これらのサイズの大きな表を管理する技術およびこれらの表全体に良質な問合せのパフォーマンスを提供する技術の両方が必要になります。この項では、パーティション化とこれらの要件に対処するための主要な方法を説明します。データ・ウェアハウスの問合せのパフォーマンスに関連する2つの機能は、パーティション・プルーニングとパーティションワイズ結合です。

パフォーマンスの向上：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、データ・ウェアハウスに必須のパフォーマンス機能です。パーティション・プルーニングでは、オプティマイザによりSQL文のFROM句およびWHERE句が分析され、パーティション・アクセス・リストの作成時に不必要なパーティションが排除されます。これにより、Oracle Databaseを使用してSQL文に関連するパーティションでのみ操作を実行できます。レンジ・パーティション化列またはリスト・パーティション化列にある範囲述語、LIKE述語、等価述語およびIN-リスト述語を使用する場合とハッシュ・パーティション化列にある等価述語およびIN-リストを使用する場合は、Oracle Databaseによりパーティションがプルーニングされます。

パーティション・プルーニングによりディスクから取得するデータ量を大幅に削減し処理時間の使用を短縮できるので、問合せのパフォーマンスおよびリソース使用量が向上します。グローバルなパーティション索引を使用して異なる列の索引および表をパーティション化する場合、基礎となる表が排除できない場合でもパーティション・プルーニングにより索引パーティションが排除されます。

実際のSQL文に応じて、Oracle Databaseにより静的プルーニングおよび動的プルーニングが使用されます。静的プルーニングは事前にアクセスされたパーティションの情報とともにコンパイル時に発生し、一方動的プルーニングは実行時に発生し、これは文によってアクセスされる正確なパーティションを事前に把握していることを意味します。静的プルーニングのサンプルの使用例は、パーティション・キー列に固定リテラルがあるWHERE条件を含むSQL文です。動的プルーニングの例は、WHERE条件内の演算子または関数の使用です。

パーティション・プルーニングは、プルーニングが発生するオブジェクトの統計に影響するので、文の実行計画にも影響します。

パフォーマンスの向上：パーティションワイズ結合

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場合にパラレル実行サーバー間で交換されるデータの量を最小化することにより問合せのレスポンス時間が削減されます。これによりレスポンス時間が大幅に削減され、CPUおよびメモリー・リソース両方の使用量を改善します。Oracle Real Application Clusters環境では、パーティションワイズ結合は相互接続のデータ通信を回避するか、少なくとも制限します。このことは大規模な結合操作に対する良質なスケーラビリティを実現するために重要です。

パーティションワイズ結合では、全体的または部分的です。Oracle Databaseにより使用する結合のタイプが決定されます。

例：SQLアクセス・アドバイザを使用したパーティション化の評価

データ・ウェアハウス環境では、常にパーティション化を考慮する必要があります。

パーティション化を評価する手順

1. 「アドバイザ・セントラル」 ページで、「SQLアドバイザ」をクリックします。
「SQLアドバイザ」 ページが表示されます。
2. 「SQLアクセス・アドバイザ」をクリックします。
「SQLアクセス・アドバイザ」 ページが表示されます。
3. 「初期オプション」 から、「デフォルト・オプションを使用」を選択し、「続行」をクリックします。
4. 「ワークロード・ソース」 から、「現在と最近のSQLアクティビティ」を選択し、「次へ」をクリックします。
「推奨オプション」 ページが表示されます。
5. 「パーティション化」を選択し、次に「包括モード」を選択して、「次へ」をクリックします。

「スケジュール」 ページが表示されます。

6. 「タスク名」 フィールドにSQLACCESStest1を入力し、「次へ」をクリックします。

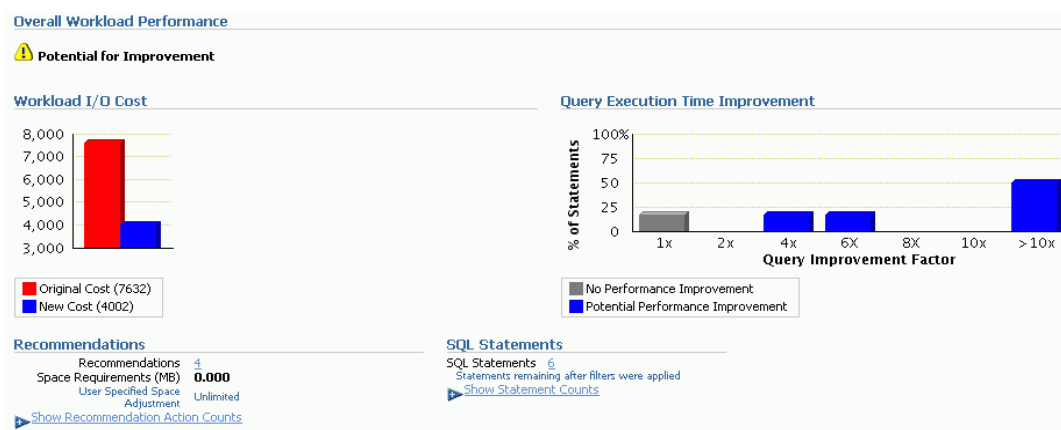
「確認」 ページが表示されます。「発行」をクリックします。

7. 「発行」をクリックします。

「確認」 ページが表示されます。

8. タスクを選択し、「結果の表示」をクリックします。「タスクの結果」 ページが表示され、パーティション化の結果として可能な向上が示されます。

図10-1 パーティション化の結果



[「図10-1 パーティション化の結果」の説明](#)

パフォーマンスの向上：問合せのリライトおよびマテリアライズド・ビュー

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の合計などの集計データを計算して格納できます。また、それらを使用して集計を加えてまたは集計を除いて結合を計算でき、コストの高い計算と同様、頻繁に実行されるサイズの大きな表間のコストの高い結合にも便利です。マテリアライズド・ビューにより、大規模な結合または問合せを処理する前にサマリーが作成されたデータを計算して格納するので、大規模または重要なクラスの問合せのためのコストの高い結合および集計に関連付けられたオーバーヘッドは排除されます。これらの環境でマテリアライズド・ビューは多くの場合、サマリーと呼ばれます。

マテリアライズド・ビューを作成して保持する主な利点の1つは、表またはビューで表されるSQL文を、詳細表で定義される1つ以上のマテリアライズド・ビューにアクセスする文に変換する問合せのリライト機能を利用できることです。変換はユーザーおよびアプリケーションに対して透過的で、SQL文内のマテリアライズド・ビューを使用したり参照する必要はありません。問合せのリライト機能は透過的なので、索引と同様にアプリケーション・コード内のSQLを無効にすることなく、マテリアライズド・ビューを追加または削除できます。

基礎となる表に大量のデータが含まれる場合、必要な集計の計算またはこれらの表間の結合の計算は、コストが高く時間がかかる処理になります。このような場合、問合せにより応答が返されるまで数分または数時間かかります。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ので、Oracle Databaseによって強力な問合せのリライト・プロセスが使用され、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。

パフォーマンスの向上：索引

ビットマップ索引はデータ・ウェアハウス環境において広く使用されています。データ・ウェアハウス環境では通常、大量のデータおよび非定型問合せがありますが、DMLトランザクションが同時に発生することは稀です。索引が表内のデータのサイズの数倍である場合があるため、サイズの大きな表に従来のBツリーを完全に索引付けすると、ディスク領域に関してコストが非常に高くなる可能性があります。ビットマップ索引は通常、表内の索引付けされたデータのサイズのほんの一部にすぎません。このようなアプリケーションの場合、ビットマップ索引には次のような特性があります。

- 大規模な非定型問合せに対するレスポンス時間の削減

- 他の索引の方法と比較した場合の記憶域要件の減少
- 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上での大幅なパフォーマンスの向上
- パラレルDMLおよびロード中の効率的なメンテナンス

パフォーマンスの向上：圧縮

パフォーマンスの向上に効果的な場合、Oracle Databaseでは、バルク・ロード操作時にロード中のデータが圧縮されます。データ量が非常に少ない小さいセグメントの場合は、圧縮を指定しても圧縮は行われません。データ変換および圧縮は内部的に処理され、圧縮を使用するためにアプリケーションを変更する必要はありません。

この機能を構成するために特にインストールが必要なものはありません。ただし、この機能を使用するには、データベース互換パラメータを11.2.0以上に設定する必要があります。

注意:

Oracle Exadata Storage Serverでは、ハイブリッド列単位圧縮など、その他の圧縮テクノロジーを使用できます。詳細は、Oracle Exadataのドキュメントを参照してください。

パフォーマンスの向上：DBMS_COMPRESSIONパッケージ

PL/SQLパッケージDBMS_COMPRESSIONによって、アプリケーションに対して適切な圧縮レベルを簡単に選択できる圧縮アドバイザ・インタフェースが提供されます。圧縮アドバイザは、データベース内のオブジェクトを分析し、達成可能な圧縮率を見積ります。

参照:

『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』のDBMS_COMPRESSIONに関する説明

パフォーマンスの向上：CREATE TABLEおよびALTER TABLEのtable_compress句

CREATE TABLEおよびALTER TABLEのtable_compress句で提供されるCOMPRESSは、圧縮レベルのパラメータを取ります。COMPRESSを使用して、データ・セグメントを圧縮してディスク使用量を削減するかどうかをデータベースに指示します。この句は、OLTPシステムと比較して、挿入操作と更新操作の量が少ないOLAP環境およびデータ・ウェアハウスで役立ちます。

注意:

表で圧縮を有効にするには、表の作成時に有効にするか、または表を変更して有効にする必要があります。

参照

『Oracle Database SQL言語リファレンス』

最適なリソースの使用

可能な場合にパラレルで操作を実行すると、システムでのリソースの使用を最大化できます。特定の時点でリソースによる制限がない場合、データベース操作をより速く実行できます。操作はCPUリソース、I/O容量、メモリーまたは（クラスタ内の）相互接続のトラフィックによって制限される場合があります。データベース操作のパフォーマンスを向上させるには、パフォーマンスの問題に焦点を当て、その問題を排除する必要があります（問題が他のリソースに移行する可能性を考慮し、排除する必要があります）。Oracle Databaseでは、使用可能なリソースの使用を最大化する機能を提供し、また不要なリソースの使用も回避します。

パラレル実行でのパフォーマンスの最適化

パラレル実行を使用すると、通常ディシジョン・サポート・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、特定のタイプのオンライン・トランザクション処理

(OLTP) およびハイブリッド・システム上で、パラレル実行の実装もできます。パラレル実行はパラレル化とも呼ばれます。パラレル化のタスクの概念を簡潔に説明すると、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年間ですべての四半期を1つのプロセスで処理せずに、四半期それぞれを4つのプロセスで処理します。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、次のプロセスを向上できます。

- サイズの大きな表のスキャン、結合またはパーティション化された索引のスキャンを要求する問合せ
- 大規模な索引の作成
- サイズの大きな表の作成（マテリアライズド・ビューを含む）
- 大量データの挿入、更新、マージおよび操作の削除

また、パラレル実行を使用して、Oracle Database内のオブジェクト型にアクセスできます。たとえば、パラレル実行により、ラージ・オブジェクト (LOB) にアクセスできます。

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帯域幅
- 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

これらの特性を持たないシステムでは、パラレル実行を使用しても大幅にパフォーマンスが向上しない場合があります。実際に、大容量を使用しているシステムまたはI/O帯域幅の少ないシステム上では、システムのパフォーマンスが低下する可能性があります。

パラレル実行の動作方法

パラレル実行では、SQL文の実行タスクが複数の小さな単位に分割され、各ユニットは個別のプロセスで実行されます。また、受信するデータ（表、索引、パーティション）も**グラニュル**と呼ばれる単位に分類されます。ユーザー・シャドウ・プロセスは、パラレル実行のコーディネータまたは問合せコーディネータとしての役割を持ちます。問合せコーディネータは次のタスクを実行します。

- 問合せの解析および並列度の判別
- 1セットまたは2セットのスレーブ（スレッドまたはプロセス）の割当て
- 問合せの制御およびパラレル問合せスレーブへの手順の送信
- パラレル問合せスレーブによるスキャンが必要な表または索引の判別
- ユーザーに対する最終的な出力の作成

並列度の設定

パラレル実行コーディネータでは、2つ以上のインスタンスのパラレル実行サーバーでSQL文を処理する場合があります。単一の操作に関連付けられているパラレル実行サーバーの数は、**並列度** (DOP) と呼ばれます。

単一の操作は、ORDER BY操作または索引なしの列表に対して結合を実行する全表スキャンなどのSQL文の一部です。

並列度を指定する方法は次のとおりです。

- PARALLELヒントを持つ文レベルでの指定
- ALTER SESSION FORCE PARALLEL文を発行してセッション・レベルでの指定
- 表の定義内で表レベルでの指定
- 索引の定義内で索引レベルでの指定

例：並列度の設定

表でDOPを4に設定する場合を想定します。

並列度を設定する手順

次の文を発行します。

```
ALTER TABLE orders PARALLEL 4;
```

待機イベントについて

待機 イベントはサーバー・プロセスによって増分した統計で、サーバー・プロセスは、イベントが完了し、処理の続行が可能となるまで待機する必要があることを示します。セッションの待機理由は様々で、大量の入力に対する待機、ディスクへの書込み操作などのサービスを完了するためのオペレーティング・システムに対する待機、またはロックやラッチを待機する時間が含まれます。

各セッションがリソースを待機する間、有効な動作は実行されません。待機イベントの多くが問題の原因となります。待機イベント・データにより、ラッチ競合、バッファ競合およびI/O競合などのパフォーマンスに影響を与える可能性のある問題の症状が明らかになります。

11 データ・ウェアハウスのバックアップおよびリカバリ

この項で説明する、データ・ウェアハウスのバックアップおよびリカバリに関する考慮事項には、次の内容が含まれます。

- [データ・ウェアハウスのバックアップおよびリカバリの処理方法](#)
- [バックアップおよびリカバリの計画とベスト・プラクティス](#)

データ・ウェアハウスのバックアップおよびリカバリの処理方法

バックアップおよびリカバリは、管理者にとって最も重要なタスクであり、データ・ウェアハウスについても同様のことが言えます。ただし、データ・ウェアハウスはデータベースのサイズが大きいため、管理者にとってはバックアップおよびリカバリに関する新たな課題が浮上することになります。

データ・ウェアハウスは、無数のリソースからデータが取得されるという点において特徴的で、最終的にデータベースに挿入される前に変換されますが、これは主にサイズが非常に大きいためです。大規模なデータ・ウェアハウスのリカバリの管理は大変なタスクで、従来のOLTPのバックアップ計画およびリカバリ計画では十分に要件を満たしません。

データ・ウェアハウスとOLTPシステムの相違点は次のとおりです。

- 通常、データ・ウェアハウスのサイズははるかに大規模です。
- データ・ウェアハウスは、運用システムとは異なる可用性の要件を備えています。ビジネスにおける決断でも、データ・ウェアハウスからの情報に依存し、たとえばサービス・デスクが操作できないような状況では、事態はさらに悪化します。また、データ・ウェアハウスのサイズのため、一定水準のデータ・ウェアハウスの可用性を保証するには非常に高いコストがかかります。
- データ・ウェアハウスは、通常ETL（抽出、変換およびロード）と呼ばれるより優れた制御のプロセスを介して移入されます。この結果、データ・ウェアハウス内の更新は正確に把握され、データ・ソースからの再生成が可能です。
- 単一のデータ・ウェアハウスには通常、多くの履歴データが格納され、大抵の場合は変更されるオブジェクトではありません。変更されないデータは、1回のみバックアップする必要があります。

システム設計の一部としてバックアップ計画を計画し、バックアップする対象およびバックアップする頻度を考慮する必要があります。バックアップ設計で最も重要な変数は、バックアップまたはリカバリの実行に使用可能なリソースの量とリカバリ時間目標（システム全体またはシステムの一部の使用不可能な状態の許容時間）です。

バックアップおよびリカバリ計画を計画する場合は、NOLOGGING操作を考慮する必要があります。従来のリカバリは、バックアップをリストアしてアーカイブ・ログから変更を適用するものですが、NOLOGGING操作には適用されません。さらに、NOLOGGING操作により操作されたデータに依存する後続の操作は失敗します。バックアップおよびリカバリ計画を設計する場合は、NOLOGGING操作を考慮する必要があります。

NOLOGGING操作が行われている場合は、バックアップを作成することはありません。

次の2つの計画のうち1つを計画するか、または2つの計画の組合せを計画します。

- ETL計画。リカバリできないトランザクションを含まないバックアップをリカバリし、バックアップと失敗の間で行われたETLをリプレイします。
- 増分バックアップ計画。その他のリカバリできないトランザクションが行われた後、すぐにバックアップを実行します。変更済データ・ブロックに基づいた増分バックアップを有効にするトラッキング・ファイル機能が提供されます。Recovery Managerによりトラッキング・ファイル機能が利用されます。

バックアップおよびリカバリの計画とベスト・プラクティス

バックアップおよびリカバリ計画を考案することは、大変なタスクです。また、数百GBの保護対象のデータを所有していて失敗した場合にリカ

バリが必要である場合は、計画は非常に複雑になります。

次のベスト・プラクティスにより、ウェアハウスのバックアップおよびリカバリ計画の実装をサポートします。

- [ベスト・プラクティスA: ARCHIVELOGモードの使用](#)
- [ベスト・プラクティスB: Recovery Managerの使用](#)
- [ベスト・プラクティスC: 読取り専用の表領域の使用](#)
- [ベスト・プラクティスD: NOLOGGING操作の計画](#)
- [ベスト・プラクティスE: すべての表領域の重要度が同等でない場合](#)

ベスト・プラクティスA: ARCHIVELOGモードの使用

アーカイブREDOログはデータベースへの変更の記録を構成するため、データがまったく失われない場合のリカバリに非常に重要です。Oracle Databaseは、次の2つのモードのいずれかで実行されます。

- ARCHIVELOG: 再利用される前に満杯になったオンラインのREDOログ・ファイルがアーカイブされます。
- NOARCHIVELOG: 再利用される前に満杯になったオンラインのREDOログ・ファイルがアーカイブされません。

ARCHIVELOGモードでデータベースを実行すると、次の利点があります。

- データベースはインスタンス障害およびメディア障害の両方から完全にリカバリされます。
- データベースを開いて使用可能な状態で、ユーザーはバックアップを実行できます。
- Oracle Databaseにより、多重化されたアーカイブ・ログによるそのアーカイブ・ログ上の単一点障害の回避がサポートされます。
- ユーザーは、表領域のPoint-in-Timeリカバリ (TSPITR) の実行機能のようなより多くのリカバリ・オプションを使用できます。
- アーカイブREDOログは、プライマリ・データベースの完全なコピーであるフィジカル・スタンバイ・データベースに転送および適用できます。
- データベースはインスタンス障害およびメディア障害の両方から完全にリカバリされます。

NOARCHIVELOGモードでデータベースを実行すると、次の結果が得られます。

- 停止後、データベースが完全に閉じられた状態で、ユーザーはデータベースのバックアップのみ実行できます。
- 最後のバックアップ後はすべてのトランザクションの損失の原因となるため、通常、メディア・リカバリ・オプションのみがデータベース全体をリストアします。

停止時間の許容について

データベースが開いている間、または閉じている間でもOracle Databaseのバックアップは作成できます。データベースの計画停止時間は業務を混乱させ、特に複数のタイムゾーンにいるユーザーを最長24時間サポートする世界規模の企業では顕著です。このような場合は、データベースへの割込みを最小化するようにバックアップ計画を設計することが重要です。

業種によっては、停止時間に余裕がある企業もあります。ビジネス計画全体で、停止時間は最小限またはまったくない状態が必要とされる場合は、バックアップ計画にオンライン・バックアップを実装する必要があります。データベースは、バックアップのために停止する必要がありま

せん。オンライン・バックアップでは、データベースをARCHIVELOGモードにする必要があります。

基本的に、ARCHIVELOGモードを使用しない理由などありません。どんなデータ・ウェアハウス（つまりすべての重要なデータベース）にとってもARCHIVELOGモードの使用が必要となります。特に、データ・ウェアハウスのサイズ（およびデータ・ウェアハウスをバックアップする時間）を考慮すると、データ・ウェアハウスがNOARCHIVELOGモードを使用する場合に必要なデータ・ウェアハウスのオフライン・バックアップの作成は、通常、実行可能ではありません。

もちろん、サイズの大きいデータ・ウェアハウスは大規模なデータの変更が発生する可能性があり、大規模なデータの変更があると大量のログ・ファイルが生成されます。大量のアーカイブ・ログ・ファイルの管理に適応するために、Recovery Managerにより、アーカイブ済のログ・ファイルを圧縮するオプションが提供されます。これにより、リカバリのためのより迅速なアクセシビリティを実現するために、より多くのアーカイブ・ログをディスクに保存できます。

要約すると、ベスト・プラクティスは、データベースをアーカイブ・ログ・モードにしてオンライン・バックアップのオプションとPoint-in-Timeリカバリのオプションを使用可能にすることです。

ベスト・プラクティスB: Recovery Managerの使用

Recovery Managerを採用する理由は多数あります。Recovery Managerをバックアップおよびリカバリ計画に統合する理由のいくつかは、次の項目が提供されるからです。

- 拡張性の高いレポート
- 増分バックアップ
- 停止時間のないバックアップ
- バックアップおよびリストアの検証
- バックアップおよびリストアの最適化
- メディア・マネージャとの簡単な統合
- ブロック・メディア・リカバリ
- アーカイブ・ログの検証および管理
- 破損ブロックの検出

ベスト・プラクティスC: 読取り専用の表領域の使用

データ・ウェアハウスが直面する最大の問題の1つは、一般的にデータ・ウェアハウスが大規模であるということです。強力なバックアップのハードウェアを使用しても、バックアップに数時間かかります。したがって、バックアップするデータの量の最小化を考慮することは、バックアップのパフォーマンスの向上のために重要なことの1つです。読取り専用の表領域は、データ・ウェアハウス内にバックアップするデータの量を削減するための最も簡単なメカニズムです。

読取り専用の表領域の利点は、データを1回のみバックアップすればよいという点です。つまり、データ・ウェアハウスに5年間の履歴データが含まれる場合は、データの最初の4年間は読取り専用にできます。理論上は、データベースの通常のバックアップでデータの20%のみがバックアップされます。これにより、データ・ウェアハウスのバックアップに必要な時間を大幅に削減できます。

ほとんどのデータ・ウェアハウスでは、時間でレンジ・パーティション化された表にデータが格納されます。通常のデータ・ウェアハウスでは、一般的に30日間から1年間までの範囲の期間で、データはアクティブです。この期間中、履歴データを更新でき、また変更できます（たとえば、小売業者が購入日後の30日間までは返品を受け入れる場合は、売上データの記録をこの期間中に変更できます）。ただし、一度データが特定の日付に到達すると、多くの場合データは静的であると認識されます。

パーティション化を利用すると、ユーザーは読取り専用のデータの静的な部分を作成できます。Recovery Managerにより、読取り専用のパーティションまたは表ではなく、読取り専用の表領域がサポートされます。読取り専用の表領域を利用してバックアップ・ウィンドウを削減するには、読取り専用の表領域の定数データ・パーティションの格納計画を考案する必要があります。ローリング・ウィンドウを実装する2つの計画

は次のとおりです。

- データ全体が静的になったら、読取り/書込み表領域から読取り専用表領域にパーティションを移動する定期的なプロセスを実装します。

この場合のベスト・プラクティスは、データベースをARCHIVELOGモードにしてオンライン・バックアップおよびPoint-in-Timeリカバリのオプションを使用可能にすることです。

- 各表領域に少数のパーティションを持つ一連の表領域を作成し、その表領域内のデータが年数を重ねるにつれて定期的に1つの表領域を読取り/書込み両用から読取り専用に変更します。

1つ考慮する必要があるのは、データのバックアップはリカバリ・プロセスの半分にすぎないということです。テープのシステムを構成する場合、4時間でデータ・ウェアハウスの読取り/書込み両用の部分をバックアップでき、データベースの80%が読取り専用であるときに完全なリカバリが必要な場合は、必然的にテープのシステムではデータベースをリカバリするために20時間かかります。

要約すると、ベスト・プラクティスは、静的表および静的パーティションを読取り専用の表領域に配置することです。読取り専用の表領域を1回のみバックアップする必要があります。

ベスト・プラクティスD: NOLOGGING操作の計画

一般的に、データ・ウェアハウスの最優先事項はパフォーマンスです。データ・ウェアハウスにより、オンラインのユーザーに快適な問合せのパフォーマンスが提供される必要があるだけでなく、大量のデータを最短时间内にロードするために、ETLプロセス中、データ・ウェアハウスは効率的である必要があります。

データ・ウェアハウスにより利用される共通の最適化の1つは、NOLOGGINGモードを使用して大量データ操作を実行することです。NOLOGGINGモードをサポートするデータベース操作は、直接パス・ロードおよび直接パス・インサート、索引の作成および表の作成です。1つの操作がNOLOGGINGモードで実行される場合は、データはREDOログに書き込まれません（より正確に言うと、サイズの小さいメタデータのセットのみREDOログに書き込まれます）。このモードは、データ・ウェアハウスの広範囲で使用され、大量データ操作のパフォーマンスを50%まで改善できます。

ただし、リカバリをサポートするために必要なデータがログ・ファイルに書き込まれることはないで、そのかわりに、従来のバックアップ・メカニズムを使用してNOLOGGING操作をリカバリできません。さらに、NOLOGGING操作が発生するデータに対する後続の操作は、たとえこれらの操作がNOLOGGING操作を使用していなかったとしてもリカバリされません。NOLOGGING操作により提供されるパフォーマンスの向上のため、一般的にETLプロセス中は、データ・ウェアハウスによるNOLOGGINGモードの使用をお勧めします。

バックアップおよびリカバリ計画を考案する場合は、NOLOGGING操作の存在を考慮する必要があります。データベースがNOLOGGING操作に依存する場合、ログ・ファイルによりNOLOGGING操作はリカバリできないので、従来のリカバリ計画（最新のテープのバックアップからリカバリし、アーカイブ・ログ・ファイルを適用する）は適用されません。

覚えておく必要のある第1の原則は、NOLOGGING操作が発生しているときにバックアップを作成しないことです。現在この原則を規定していないので、DBAはNOLOGGING操作がバックアップ操作に重複しないようなバックアップ・ジョブおよびETLジョブをスケジュールする必要があります。

NOLOGGING操作が存在するバックアップおよびリカバリには、2つのアプローチがあります。ETLバックアップまたは増分バックアップです。データ・ウェアハウス内でNOLOGGING操作を使用していない場合は、次のオプションのいずれも選択する必要はありません。アーカイブ・ログを使用してデータ・ウェアハウスをリカバリできます。ただし次のオプションは、リカバリの場合のアーカイブ・ログ・ベースのアプローチにおいていくつかのパフォーマンスの面で優れています。

- [抽出、変換およびロード](#)
- [増分バックアップ](#)

抽出、変換およびロード

ETLプロセスでは、データをデータにロード（再ロード）するためにいくつかのOracleの機能またはツール、およびメソッドの組合せを使用します。これらの機能またはツールの構成要素は次のとおりです。

- **トランスポータブル表領域。** Oracleトランスポータブル表領域機能により、ユーザーはOracle Database間で表領域を迅速に移動させることができます。この方法は、データベース間で大量データを移動するのに最も効率的です。Oracle Databaseにより、プラットフォーム

フォーム間で表領域を転送できる機能が提供されます。ソース・プラットフォームとターゲット・プラットフォームが異なるエンディアンネスである場合、Recovery Managerによりターゲット・フォーマットに転送されている表領域が変換されます。

- SQL*Loader。SQL*Loaderにより、外部フラット・ファイルから1つのOracle Database上の複数の表にデータがロードされます。SQL*Loaderには、データファイル内のデータの形式にほとんど制限のない強力なデータ解析エンジンがあります。
- データ・ポンプ（エクスポート/インポート）。Oracle Databaseにより、Oracle Data Pumpテクノロジーが提供され、これにより1つのデータベースから他のデータベースにデータおよびメタデータを高速で移動することが可能です。このテクノロジーは、データ移動ユーティリティであるデータ・ポンプ・エクスポートおよびデータ・ポンプ・インポートの基本となります。
- 外部表。外部表機能とは、既存のSQL*Loader機能を補完するものです。この機能により、データベース内の表にあるデータと同様に、外部ソースのデータにアクセスできるようになります。

ETL計画およびNOLOGGING操作

アプローチの1つは、定期的にデータベースのバックアップをして、その週全体のETLプロセスを再作成するために必要なデータファイルを保存することです。リカバリが必要な場合、データ・ウェアハウスは最新のバックアップからリカバリされます。次に、従来のリカバリのシナリオで行われていたようにアーカイブREDOログを適用してロールフォワードを行うかわりに、データ・ウェアハウスはETLプロセスをリプレイすることでロールフォワードを行います。この実例では、ETLプロセスは簡単にリプレイできることを前提とし、通常、各ETLプロセスの抽出ファイルのセットの保存も含まれます（たとえば不正なデータ送信の修復の必要があるかどうかを識別できるように、データ・ウェアハウスの多くはすでにベスト・プラクティスとしてこれを行っています）。

このアプローチのサンプル実装は、毎週末データ・ウェアハウスのバックアップを作成し、毎晩のETLプロセスをサポートするために必要なファイルを保存することです。したがって、データベースをリカバリするためには最大7日間分のETLプロセスが再適用される必要があります。データ・ウェアハウス管理者は、テープからのリカバリ・スピードおよび前のETL実行からのパフォーマンス・データに基づいて簡単に時間の長さを計画してデータ・ウェアハウスをリカバリできます。

基本的にデータ・ウェアハウス管理者は、NOLOGGING操作を介してETLプロセスでのより高いパフォーマンスを獲得しており、その際は多少複雑であり自動化されていないリカバリ・プロセスを通過する必要があります。データ・ウェアハウス管理者の多くは、これを価値のある代償だと理解しています。

このアプローチの弱点は、データ・ウェアハウスで発生した関連する変更のすべてを追跡する負荷がデータ・ウェアハウス管理者にかかるということです。このアプローチでは、ETLプロセスの範囲に入らない変更は取得されません。たとえば、いくつかのデータ・ウェアハウスでは、エンドユーザーが所有する表およびデータ構造を作成します。それらの変更はリカバリ時に失われます。この制限はエンドユーザーに伝達される必要があります。かわりに、管理者がエンドユーザーに個別の表領域にあるすべてのプライベート・データベース・オブジェクトを作成するように指示し、リカバリの間、DBAはETLプロセスをリプレイするアプローチを使用して残りのデータベースをリカバリする一方で従来のリカバリを使用してこの表領域をリカバリすることもできます。

要約すると、ベスト・プラクティスは、リカバリできない（NOLOGGING）トランザクションを含まないバックアップをリストアすることです。次にETLプロセスをリプレイしてデータを再ロードします。

ブロック変更トラッキング・ファイルのサイジング

ブロック変更トラッキング・ファイルのサイズは次の項目に比例します。

- データベース・サイズ（バイト）。ブロック変更トラッキング・ファイルには、データベース内のデータファイル・ブロックを表すデータが含まれます。データはおおよそデータベースのサイズ全体の250000分の1です。
- 有効なスレッドの数。すべてのReal Application Clusters (RAC) インスタンスは同一のブロック変更トラッキング・ファイルにアクセスできますが、インスタンスにより、ロックまたはノード間のブロック・スワッピングなしでトラッキング・ファイルの異なる領域が更新されます。個別のインスタンスに対してではなく、データベース全体に対してブロック変更トラッキング・ファイルを有効化できます。
- 変更済ブロックのメタデータ。ブロック変更トラッキング・ファイルは最後のバックアップ以来の変更に加えて、前のバックアップ間のすべての変更の記録を保持しています。トラッキング・ファイルは、最大で8個のバックアップの変更履歴を保持します。トラッキング・ファイルに8個のバックアップの変更履歴が含まれる場合は、Oracle Databaseにより最も古い変更履歴情報が上書きされます。

たとえばスレッドを1つのみ所有し、Recovery Managerリポジトリに8個のバックアップを保持する500GBのデータベースの場合は、20MBのブロック変更トラッキング・ファイルが必要です。

$((\text{Threads} * 2) + \text{number of old backups}) * (\text{database size in bytes})$

250000

増分バックアップ

NOLOGGING操作が存在する、より自動化されたバックアップおよびリカバリ計画では、Recovery Managerの増分バックアップ機能を利用します。Recovery Managerが最初にリリースされて以来、増分バックアップはRecovery Managerの一部を構成しています。増分バックアップには、前のバックアップ後に変更されたブロックのみをバックアップする機能があります。データファイルの増分バックアップでは、データファイル内のすべての使用済ブロックのバックアップを必要とするのではなく、基本的にブロックごとのデータの変更が取得されます。結果として取得されるバックアップのセットは、データファイル内のすべてのブロックが変更されないのであれば、通常すべてのデータファイルのバックアップより小さくて効率的です。

Oracle Databaseには、変更トラッキング・ファイル機能を実装して増分を高速化する機能があります。ブロック変更トラッキングを有効にする場合は、Oracle Databaseにより、すべてのデータベースの変更の物理的な場所が追跡されます。Recovery Managerにより自動的に変更トラッキング・ファイルが使用され、増分バックアップ中に読み込む必要のあるブロックが決定されて、バックアップするブロックに直接アクセスします。

増分アプローチ

このアプローチを使用する通常のバックアップおよびリカバリ計画は、毎週末データ・ウェアハウスをバックアップし、ETLプロセスが完了した後は毎晩データ・ウェアハウスの増分バックアップを取得します。従来のバックアップと同様に、増分バックアップとNOLOGGING操作を同時に実行しないように注意してください。データ・ウェアハウスをリカバリするために、データベースのバックアップはリストアされ、毎晩の増分バックアップは再適用されます。アーカイブ・ログではNOLOGGING操作は取得されませんが、NOLOGGING操作からのデータは増分バックアップに存在します。さらに、前のアプローチとは異なり、このバックアップおよびリカバリ計画は、Recovery Managerを使用して完全に管理できます。

ETLをリプレイするアプローチおよび増分バックアップのアプローチの両方とも、多くのNOLOGGING操作で構成されているワークロードであるデータベースの効率的で安全なバックアップおよびリカバリのソリューションとしてお勧めします。バックアップおよびリカバリ計画においてNOLOGGING操作を考慮することが最も重要です。

要約すると、ベスト・プラクティスは、NOLOGGING操作のためオブジェクトをリカバリできない状態にしておくダイレクト・ロードの後、ブロック変更トラッキング機能を実装して増分バックアップを作成することです。

ベスト・プラクティスE: すべての表領域の重要度が同等でない場合

データベース内の各表領域を同様に処理することが最も単純なバックアップおよびリカバリのシナリオですが、オラクル社は、DBAが必要に応じて各表領域に対してバックアップおよびリカバリのシナリオを考案できるような柔軟性を提供しています。

バックアップおよびリカバリという観点で見ると、データ・ウェアハウス内のすべての表領域が同等に重要だということはありません。DBAはこの情報を利用して、より効率的なバックアップおよびリカバリ計画を必要に応じて考案できます。バックアップおよびリカバリの基本粒度は表領域なので、異なる表領域には異なるバックアップおよびリカバリ計画が存在する可能性があります。最も基本的なレベルでは、一時表領域がバックアップされることはありません (Recovery Managerにより規定されるルール)。

さらに、いくつかのデータ・ウェアハウスでは、明示的な一時表領域ではないにもかかわらず、エンドユーザーが一時表領域と増分の結果を格納するスクラッチ領域専用の表領域であるために、基本的に一時表領域として機能している表領域があります。ビジネス要件によっては、これらの表領域をバックアップおよびリストアする必要はありません。かわりに、これらの表領域が失われた場合は、エンドユーザーは所有するデータ・オブジェクトを再作成します。

データ・ウェアハウスの多くは、その他のデータより重要な一部データを持っています。たとえば、データ・ウェアハウス内の売上データは非常に重要で、リカバリの段階では、このデータはできるだけ早くオンラインにする必要があります。しかし、同一のデータ・ウェアハウス内では、企業のWebサイトからのクリック・ストリーム・データを格納している表の重要度はかなり低くなります。ビジネスを継続する上で、このデータが数日間オフラインになること、またはデータベース・ファイル損失時にクリック・ストリーム・データが数日間失われることは許容範囲内です。このシナリオでは、売上データを含む表領域は頻繁にバックアップする必要があり、一方でクリック・ストリーム・データを含む表領域は毎週1回または2回バックアップする必要があります。

12 データ・ウェアハウスのセキュリティ

この項では、データ・ウェアハウスのセキュリティの考慮事項について説明します。内容は次のとおりです。

- [データ・ウェアハウスのセキュリティについて](#)
- [データ・ウェアハウスのセキュリティのロールおよび権限の使用](#)
- [データ・ウェアハウスの仮想プライベート・データベースの使用](#)
- [Oracle Label Securityの概要](#)
- [データ・ウェアハウスのファイニングレイン監査の概要](#)
- [データ・ウェアハウスの透過的データ暗号化の概要](#)

データ・ウェアハウスのセキュリティについて

データ・ウェアハウジングは、セキュリティに対して固有の課題を投げかけます。業務データ・ウェアハウスは、巨大なシステムであることが多く、多様なセキュリティ上のニーズに応えるとともに、数多くのユーザーに対応しています。また、データ・ウェアハウスは、柔軟性のある、強力なセキュリティ・インフラストラクチャを必要としますが、そのセキュリティ機能は、厳密なパフォーマンスとスケーラビリティなど条件付きの環境で動作する必要があります。

データ・ウェアハウスのセキュリティの必要性について

よく知られたセキュリティに関する多くの基本要件は、他のシステムと同等にデータ・ウェアハウスにも適用されます。アプリケーションでは、不正ユーザーによるデータへのアクセスまたは修正を防止する必要があります。また、アプリケーションおよび基礎データは、簡単にハッカーが盗用できるものであってはなりません。データは、正当なユーザーが必要に応じてデータを使用できるようにする必要があります。さらに、ユーザーが実行するアクティビティのレコードをシステムで保持する必要があります。

ウェアハウスには複数のソースから統合されたデータが含まれるため、情報を盗もうとする側から見ると、企業で最も利益を得られるターゲットとなる可能性があるため、セキュリティの要件はデータ・ウェアハウスでは特に重要です。さらに、強固なセキュリティ・インフラストラクチャにより、有効性が大幅に改善され、データ・ウェアハウス環境のコストが削減されます。

データ・ウェアハウスのセキュリティの一般的な顧客シナリオには次のような事例があります。

- ある企業は、多くの部署や子会社で広く使用される企業データ・ウェアハウスを管理しています。その企業では、各部署の従業員には自分の部署に関連するデータのみを表示し、本社の従業員には全体像が表示されるセキュリティ・インフラストラクチャを必要としています。
- ある企業のデータ・ウェアハウスには、個人情報が格納されています。プライバシーに関連する法では、そのような個人情報の使用が規制される場合があります。データ・ウェアハウスでは、このような法律に沿うようにデータを管理する必要があります。
- ある企業は、データ・ウェアハウスのデータをクライアントに販売します。クライアントには、購入または申し込んだデータのみが表示されるようにし、他のクライアントのデータの表示は許可しません。

データ・ウェアハウスのセキュリティのロールおよび権限の使用

システム権限、オブジェクト権限およびロールは、基本的なデータベース・セキュリティを提供します。また、ユーザーによるデータへのアクセスを制御し、ユーザーが実行可能なSQL文の種類を制限するように設計されています。ロールは異なるレベルのデータベース・アクセスを作成するために使用できる権限のグループです。たとえば、表およびプログラムを作成できるように、アプリケーション開発者に対してロールを作成できます。

必要な権限を所有する場合にかぎり、他のユーザーに権限およびロールを付与できます。ロールおよび権限の付与は、管理者レベルで開始します。データベースの作成で管理ユーザーsysにより、すべてのシステム権限および事前定義されたOracleロールの作成と付与が行われます。ユーザーsysは、他のユーザーに権限およびロールを付与でき、これらを付与されたユーザーに対して、他のユーザーに特定の権限を付与する権限を付与することもできます。明示的に付与された権限がない場合、ユーザーはデータベース内のどの情報にもアクセスできません。

ロールおよび権限はデータのセキュリティを規定し、データ・ウェアハウスにとってこれらのロールおよび権限の使用は不可欠です。これは多くのアプリケーションおよびツールを介してユーザーがデータにアクセスするためです。

データ・ウェアハウスの仮想プライベート・データベースの使用

仮想プライベート・データベース (VPD) によって、表、ビューまたはシノニムで直接、粒度の細かいレベルにセキュリティを規定できます。これは、セキュリティ・ポリシーが表、ビューまたはシノニムに直接連結されており、ユーザーがデータにアクセスする際にセキュリティをバイパスする方法がない場合に自動的に適用されるためです。述語を使用する動的に付与されているSQL文によって、VPDはデータへのアクセスを低いレベルに制限し、セキュリティ・ポリシーをデータベース・オブジェクトに関連付けます。また、完全なデータ分離の保証があれば、複数のユーザーが単一のデータベース・サーバーに格納された重要なデータへ直接セキュア・アクセスできます。VPDを使用すると、銀行顧客には顧客自身のアカウント履歴のみを見せ、複数の企業（競争企業同士）のデータを扱う企業の場合は、同じデータ・ウェアハウスからデータを扱っている企業にそれぞれのデータのみを見せることができます。さらに、VPDはセキュリティ関連列への制御アクセスを提供するので、従業員は自身の給与のみを確認できるようになります。

VPDは透過アプリケーションです。セキュリティはデータベース層で規定され、データベース内でのデータ・アクセスを制御するためのアプリケーション固有のロジックを考慮します。標準のアプリケーションおよびカスタム・ビルドのアプリケーションは、アプリケーション・コードの単一行の変更を行うことなく、ファイングレイン・アクセス制御を活用できます。

Enterprise内では、VPDによりアプリケーションの配布にかかる所有コストが削減され、データにアクセスするすべてのアプリケーション内ではなくウェアハウス内にセキュリティが構築されます。セキュリティは、ユーザーによるデータへのアクセスに関係なくデータベースによって規定されているためにより強固になりましたが、非定型の問合せのツールまたは新しいレポート・ライターを介してデータへアクセスするユーザーはセキュリティを経由できません。多くのエンドユーザー・ツールと同様、様々なアプリケーションをサポートするEnterprise Data Warehouseにおいて、仮想プライベート・データベースは主要なテクノロジーです。

仮想プライベート・データベースの動作方法

仮想プライベート・データベースは表、ビューまたはシノニムを使用するセキュリティ・ポリシーとの関連付けにより有効になります。管理者はPL/SQL DBMS_RLSパッケージを使用して、ポリシー・ファンクションとデータベース・オブジェクトをバインドします。連結したセキュリティ・ポリシーを使用するオブジェクトへの直接アクセスまたは間接アクセスにより、データベースはポリシーを実装したファンクションを参照します。ポリシー・ファンクションは、データベースがユーザーのSQL文に追加する述語 (WHERE句) を戻します。そのため、ユーザーによるデータへのアクセスの修正は透過的で動的です。

アプリケーション・コンテキストによって、アクセス条件は組織、サブスクリバ番号、アカウント番号または地位など、データベース管理者が重要と考える事実上すべての属性に基づくことができます。たとえば、ユーザーが顧客、営業担当またはマーケティング・アナリストかどうか関係なく、売上データのウェアハウスは顧客番号に基づいたアクセスを規定できます。この場合、顧客はWeb上で注文履歴を確認でき（顧客自身の注文のみ）、営業担当は複数の注文を見ることができます（自身の企業の顧客の注文のみ）。また、アナリストの場合、過去の2四半期におけるすべての売上を分析できます。アプリケーション・コンテキストは、特定のオブジェクトでファイングレイン・アクセス制御ポリシーに適用する可能性のあるセキュアなデータ・キャッシュとして機能します。ユーザーがデータベースにログインすると、Oracle Databaseはユーザー・セッションのキャッシュ情報にアプリケーション・コンテキストを設定します。アプリケーション・コンテキストの情報は、特定のアプリケーションに関連した情報に基づいて開発者によって定義されます。たとえば、地域別売上データを問い合わせるレポート・アプリケーションは、ユーザーの地位および部門に関するアクセス制御に基づくことができます。この場合、アプリケーションはユーザーのログイン時に各ユーザーのアプリケーション・コンテキストを最初に設定し、各ユーザーの地位および部門の従業員と部署に関する表から問い合わせられたデータを使用してコンテキストを移入できます。地域別売上表でVPDポリシーを実装しているパッケージは、ユーザーの地位および部門の各問合せを移入するためにこのアプリケーション・コンテキストを参照します。アプリケーション・コンテキストはパフォーマンスの劣化を引き起こす可能性のある実行中の副問合せを不要とみなします。

Oracle Label Securityの概要

Oracle Databaseのセキュリティ・オプションであるOracle Label Securityはラベルベースのアクセス制御を規定するために仮想プライベート・データベース (VPD) を拡張します。Oracle Label Securityは完全なVPD対応アプリケーションで、ラベル付きデータ管理によってVPDを拡大します。セキュアなデータ・ウェアハウスの配布を簡略化し、基本的なセキュリティを提供します。

ラベルベースのアクセス制御を使用すると、表の行に機密性ラベルが割り当てられ、これらのラベルに基づいてデータへのアクセスが制御されて、適切なセキュリティ・ラベルがマークされたデータを確認できます。たとえば、組織が企業の機密情報とパートナーに関する情報を区別する場合です。または、特定の主要なパートナーと共有できる機密情報、および会計部門や営業部門などの特定の企業内部のグループのみがアクセス可能な機密情報などがある場合です。ラベル・データの管理機能は、適切なデータ・アクセス・レベルで適切な人に適切な情報を提供するために組織にとって非常に有利です。

Oracle Label Securityの動作方法

Oracle Label Securityは、ラベルのコレクション、ユーザーの認可およびセキュリティ強制オプションであるポリシーを使用します。一度作成されると、ポリシーはすべてのアプリケーション・スキーマまたは特定のアプリケーション表に適用できます。Oracle Label Securityは単一のデータ・ウェアハウス内で複数のポリシー定義をサポートします。ラベル定義、ユーザーの認可および強制オプションはポリシーごとに定義されます。たとえば、マーケティングのポリシーにはマーケティングのみのラベルやマネージャおよび上級副社長などのラベルが含まれます。

Oracle Label Securityは行に含まれるラベル、各データベース・セッションと関連付けられたラベル、およびセッションに割り当てられたOracle Label Security権限に基づくデータベース表の列へのアクセスを調整します。これにより、ユーザーに標準のデータベース・システムおよびオブジェクト権限が付与された後にアプリケーション表でアクセス調整が行われます。たとえば、ユーザーに表に対するSELECT権限があるとします。ユーザーが表でSELECT文を実行すると、Oracle Label Securityは選択した行を評価し、ユーザーに割り当てられた権限およびアクセス・ラベルに基づいて選択した行にアクセスできるかどうかを決定します。また、Oracle Label SecurityはUPDATE文、DELETE文およびINSERT文に対してセキュリティ・チェックも行います。ラベルは表およびマテリアライズド・ビューに適用できます。マテリアライズド・ビューによりパフォーマンスが増加し、ラベルによりセキュリティが増加するために、データ・ウェアハウス環境で目的の柔軟性、スピードおよびスケラビリティを実現します。

データ・ウェアハウスによりラベルを利用する方法

Oracle Label Securityを使用すると、複数のソースから利便性、管理性および集中管理によって、1つの大きなシステムに情報を統合できます。このセキュリティ・オプションはアプリケーション自体で行われるので、PL/SQLプログラミングを行う必要がありません。これにより、情報の統合およびデータ自体でのセキュリティの制限によるリスクの最小化が可能になり、行レベルでのデータへのアクセス制御によって精度の高いアクセス・セキュリティが提供されます。

データ・ウェアハウスのファイングレイン監査の概要

ファイングレイン監査により、コンテンツに基づいたデータ・アクセスの監視ができます。監査するレコードの列および条件を指定でき、指定列を使用した接続で使用したDML文の特定のタイプに対する監査の制限を条件に含むことができます。また、監査イベントの発生時におけるルーチン名を指定できます。このルーチンは管理者に通知またはアラートするか、エラーおよび異常を処理できます。たとえば、中央税務当局で使用すると、納税申告へのアクセスを追跡し従業員による詮索を防止できます。特定の表でSELECT文を発行した特定のユーザーを把握するだけでは十分ではなく、日常業務に必要ではない情報にアクセスしようとする際のさらに細かいレベルでの監査が強固なセキュリティに必要です。この場合、列または行のSELECT文には仕事に無関係な情報が含まれます。ファイングレイン監査はこの機能を提供します。

ファイングレイン監査はDBMS_FGAパッケージまたはデータベース・トリガーを使用してユーザー・アプリケーションで実装できます。

データ・ウェアハウスの透過的データ暗号化の概要

透過的データ暗号化によって、オペレーティング・システムのファイルに格納されるデータとして、データベース列にある機密データを暗号化できます。データベース外部のセキュリティ・モジュールで暗号キーの管理および記憶領域をセキュアに保ちます。既存のアプリケーションに暗号化したルーチンを埋め込む必要がなく、コストおよび暗号化の複雑さを大幅に削減しました。いくつかの単純なコマンドを使用することで、機密のアプリケーション・データは暗号化されます。

多くの暗号化ソリューションは、アプリケーション・コード内の暗号化ファンクションに対する特定のコールを必要とします。これには高いコストが必要で、アプリケーションの広範囲にわたる理解、およびソフトウェアの書込みと保持の技術が要求されます。一般的に、ほとんどの組織には既存のアプリケーションを修正して暗号化ルーチンに対するコールを作成する時間や専門家が不足しています。透過的データ暗号化は、Oracle Databaseに埋め込まれた暗号化により暗号化問題を処理します。これはSQL* Loaderではなく直接パス・ロードを使用して行われることに注意してください。

SQLを介して実行されるアプリケーション・ロジックは変更することなく実行を継続します。言い換えれば、アプリケーションはアプリケーション表にデータを挿入するために同じ構文を使用でき、Oracle Databaseはディスクへ情報を書き込む前にデータを自動的に暗号化します。後続のSELECTオプションには透過的に復号化されたデータが含まれ、アプリケーションは通常の作業を継続します。

索引

[A](#) [B](#) [C](#) [D](#) [E](#) [G](#) [I](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [あ](#) [か](#) [さ](#) [た](#) [は](#) [ま](#) [や](#) [ら](#)

A

ARCHIVELOGモード
ベスト・プラクティス, [\[1\]](#)

B

Bツリー索引, [\[1\]](#)

C

COMPATIBLE, [\[1\]](#)
COMPRESS句, [\[1\]](#)
CPU, [\[1\]](#)
CUBE関数, [\[1\]](#)

D

DBMS_COMPRESSIONパッケージ, [\[1\]](#)
DBMS_STATSパッケージ, [\[1\]](#)
DB_BLOCK_SIZE, [\[1\]](#)
DB_FILE_MULTIBLOCK_READ_COUNT, [\[1\]](#)
ddユーティリティ, [\[1\]](#)

E

EXPLAIN PLAN文, [\[1\]](#)

G

GROUPING SETS関数, [\[1\]](#)
GROUPING関数, [\[1\]](#)

I

I/O, [\[1\]](#), [\[2\]](#)
監視, [\[1\]](#)

M

MERGE PARTITION, [\[1\]](#)
MOVE PARTITION, [\[1\]](#)

N

NOLOGGING
ベスト・プラクティス, [\[1\]](#)

O

OPTIMIZER_FEATURES_ENABLE, [\[1\]](#)
Oracle Label Security, [\[1\]](#)
データ・ウェアハウス, [\[1\]](#)
Orionユーティリティ, [\[1\]](#)

P

PARALLEL_ADAPTIVE_MULTI_USER, [\[1\]](#)
PARALLEL_MAX_SERVERS, [\[1\]](#)

Q

QUERY_REWRITE_ENABLED, [\[1\]](#)
QUERY_REWRITE_INTEGRITY, [\[1\]](#)

R

RANK関数, [\[1\]](#)
Recovery Manager
 ベスト・プラクティス, [\[1\]](#)
ROLLUP関数, [\[1\]](#)

S

SPLIT PARTITION, [\[1\]](#)
SQLアクセス・アドバイザー, [\[1\]](#)
 索引の最適化, [\[1\]](#)
 パーティション化の評価, [\[1\]](#)
SQLチューニング・アドバイザー, [\[1\]](#)
STAR_TRANSFORMATION_ENABLED, [\[1\]](#)

T

table_compress句, [\[1\]](#)

U

UTLXPLAN. SQLスクリプト, [\[1\]](#)

V

VPD
 動作, [\[1\]](#)

W

WITH句, [\[1\]](#)

あ

圧縮, [\[1\]](#)
圧縮アドバイザー・インタフェース, [\[1\]](#)
一般的なタスク
 データ・ウェアハウス, [\[1\]](#)

移動平均の計算, [\[1\]](#)
ウィザード
 「外部表の作成」ウィザード, [\[1\]](#)
演算子
 接続, [\[1\]](#)
 編集, [\[1\]](#)
 リポジトリ・オブジェクトとの同期化, [\[1\]](#)
演算子の属性, [\[1\]](#)
演算子のプロパティ
 設定, [\[1\]](#)
演算子, マッピング, [\[1\]](#)
 接続, [\[1\]](#)
 タイプ, [\[1\]](#)
 追加, [\[1\]](#)
 編集, [\[1\]](#)
オペティマイザ統計, [\[1\]](#)

か

階層
 説明, [\[1\]](#)
外部表
 作成用ウィザード, [\[1\]](#)
 新規定義の作成, [\[1\]](#)
 定義, [\[1\]](#)
 「フラット・ファイル」も参照
 「外部表の作成」ウィザード, [\[1\]](#)
仮想プライベート・データベース
 データ・ウェアハウス, [\[1\]](#)
記憶域
 最適化, [\[1\]](#)
キューブ, [\[1\]](#)
 ディメンション, [\[1\]](#)
 メジャー, [\[1\]](#)
 リレーショナル実装, [\[1\]](#)
 例, [\[1\]](#)
行間計算
 実行, [\[1\]](#)
クエリー・リライト, [\[1\]](#), [\[2\]](#), [\[3\]](#), [\[4\]](#), [\[5\]](#)
グループ
 接続, [\[1\]](#)
 プロパティの設定, [\[1\]](#)
グループ・プロパティ, [\[1\]](#)
権限
 データ・ウェアハウスのセキュリティ, [\[1\]](#)

さ

索引
 SQLアクセス・アドバイザーを使用した最適化, [\[1\]](#)
 最適化, [\[1\]](#)
 ビットマップ, [\[1\]](#)
索引統計, [\[1\]](#)
作成
 マッピング, [\[1\]](#)
システム統計, [\[1\]](#)

実行計画, [\[1\]](#)
分析, [\[1\]](#)

実装
スター・スキーマ, [\[1\]](#)
リレーショナル・キューブ, [\[1\]](#)

自動バインド・ルール, [\[1\]](#)

初期化パラメータ
設定, [\[1\]](#)

スパース性, [\[1\]](#)

セキュリティ
データ・ウェアハウス, [\[1\]](#)

接続
演算子, [\[1\]](#)
グループ, [\[1\]](#)
属性, [\[1\]](#)

設定
マッピング・プロパティ, [\[1\]](#)

相対的な寄与率の計算, [\[1\]](#)

属性
接続, [\[1\]](#)
プロパティの設定, [\[1\]](#)
レベル属性, [\[1\]](#)

属性のプロパティ, 設定, [\[1\]](#)

た

待機イベント, [\[1\]](#)

稠密化, [\[1\]](#)

追加
マッピング演算子, [\[1\]](#)

ツール
データ・ウェアハウス, [\[1\]](#)

定義
マッピング, [\[1\]](#)

ディスク
要件, [\[1\]](#)

ディメンション
階層, [\[1\]](#)
実装, [\[1\]](#)
スター・スキーマ実装, [\[1\]](#)
ディメンション・ロール, [\[1\]](#)
例, [\[1\]](#)
レベル属性, [\[1\]](#)
レベルの関係, [\[1\]](#)

データ圧縮, [\[1\]](#)

データ・ウェアハウス
一般的なタスク, [\[1\]](#)
主な特性, [\[1\]](#)
設定, [\[1\]](#), [\[2\]](#)
ツール, [\[1\]](#)

データ・ウェアハウスのセキュリティ, [\[1\]](#)

データのオフロード
ローリング・ウィンドウ, [\[1\]](#)

透過的データ暗号化
データ・ウェアハウス, [\[1\]](#)

同期化
演算子トリポジットリ・オブジェクト, [\[1\]](#)

統計
オブティマイザ, [\[1\]](#)

は

- バインド
 - 自動バインド, ルール, [\[1\]](#)
- バックアップ, [\[1\]](#)
- バックアップとリカバリ, [\[1\]](#)
 - ベスト・プラクティス, [\[1\]](#)
- パーティション, [\[1\]](#)
- パーティション外部結合, [\[1\]](#)
- パーティション・プルーニング, [\[1\]](#)
- パーティションワイズ結合, [\[1\]](#)
- ハードウェア構成, [\[1\]](#)
 - 確認, [\[1\]](#)
- パフォーマンス
 - SQLアクセス・アドバイザーの使用, [\[1\]](#)
 - SQLチューニング・アドバイザーの使用, [\[1\]](#)
 - 圧縮による向上, [\[1\]](#)
 - クエリー・リライトの使用, [\[1\]](#)
 - ヒントの使用, [\[1\]](#)
- パラメータ
 - 初期化, [\[1\]](#)
 - メモリー管理, [\[1\]](#)
- パラレル実行, [\[1\]](#)
 - 動作, [\[1\]](#)
- パラレル問合せ, [\[1\]](#), [\[2\]](#)
 - 監視, [\[1\]](#)
- 必要なメモリー, [\[1\]](#)
- 表
 - 「外部表」も参照
- 表統計, [\[1\]](#)
- ヒント, [\[1\]](#)
- ファイングレイン監査
 - データ・ウェアハウス, [\[1\]](#)
- フラット・ファイル
 - 「外部表」も参照
- プロパティ
 - 設定, [\[1\]](#)
 - マッピング, [\[1\]](#)
- 分析SQL, [\[1\]](#)
- 分析機能, [\[1\]](#)
- 並列度 (DOP), [\[1\]](#)
- ベスト・プラクティス
 - ARCHIVELOGモード, [\[1\]](#)
 - NOLOGGING, [\[1\]](#)
 - Recovery Manager, [\[1\]](#)
 - バックアップとリカバリ, [\[1\]](#)
 - 表領域の相違, [\[1\]](#)
 - 読取り専用表領域, [\[1\]](#)

ま

- マッピング
 - 作成, [\[1\]](#)
 - 説明, [\[1\]](#)
 - 定義, [\[1\]](#)
 - プロパティの設定, [\[1\]](#)
- マッピング演算子, 設定, [\[1\]](#)

マッピング演算子
 接続, [\[1\]](#)
 説明, [\[1\]](#)
 タイプ, [\[1\]](#)
 追加, [\[1\]](#)
 編集, [\[1\]](#)
 リポジトリ・オブジェクトとの同期化, [\[1\]](#)
マテリアライズド・ビュー, [\[1\]](#)
メモリー, [\[1\]](#)
メモリー管理
 パラメータ, [\[1\]](#)

や

読取り専用表領域
 ベスト・プラクティス, [\[1\]](#)

ら

リソース使用量, [\[1\]](#)
 最小化, [\[1\]](#)
リソース・マネージャ, [\[1\]](#)
リフレッシュ, [\[1\]](#)
 データ・ウェアハウス, [\[1\]](#)
リライト
 問合せ, [\[1\]](#), [\[2\]](#)
列統計, [\[1\]](#)
レンジ・パーティション化された表, [\[1\]](#)
ローリング・ウィンドウ, [\[1\]](#)
ロール
 ディメンション・ロール, [\[1\]](#)
 データ・ウェアハウスのセキュリティ, [\[1\]](#)

この図には、サンプリングされなかった2つのファイルが表示されています。右下に「サンプル」ボタンがあります。

この図には、OWB_DEMOというプロジェクトが示されています。「ファイル」ノードの下に、SOURCEというモジュールがあります。

このスクリーンショットは、PRODUCTSディメンション内の各属性がマップされる表の列をダイアグラムで表したものです。

スクリーンショットには、SALESキューブのリレーショナル実装が表示されます。左側にはSALESキューブがあり、右側にはSALES表があります。すべてのメジャーおよびディメンション参照は、SALESと呼ばれる表に格納されます。

この図は、プロジェクト・ナビゲータ上でのマッピング・ノードを示しています。マッピング・モジュールでMAP1オブジェクトが現在選択されています。

このスクリーンショットは、キャンバス上の表演算子を示しています。上部にはメニュー・バーがあります。メニュー・バーの下にはマッピング・エディタで様々な操作を実行するためのツールバーの2つの行があります。ツールバーの下でマッピング・エディタは半分に分かれます。左半分には、上から順に「ナビゲータ」ペイン、「マッピングのプロパティ」ペイン、「パレット」ペイン、および「鳥瞰図」ペインが並んでいます。右半分には「表演算子」ソースを使用する「マッピング」キャンバス・ペインが表示されます。「マッピング」エディタ・ページの下部にはインジケータ・バーが表示されます。

このスクリーンショットは、マッピングで接続した演算子を示しています。左側にはINVOICES_SOURCE表が表示されています。右側にはS_TABLE表が表示されています。INVOICES_SOURCE表はS_TABLE表の入力として使用されます。

このスクリーンショットは、バインドされていないステージング表（属性なし）とソース表を示しています。左側がINVOICES_SOURCE表で、右側がS_TABLE表です。

このスクリーンショットは、「作成とバインド」ダイアログ・ボックスを示しています。このダイアログ・ボックスには、上から順に「名前」フィールドおよび「作成」フィールドがあります。ダイアログ・ボックスの左下隅には「ヘルプ」ボタンがあります。ダイアログ・ボックスの右下隅には、左から順に「OK」ボタンおよび「取消」ボタンがあります。

このスクリーンショットは「ターゲット・ロード順序」ダイアログ・ボックスを示しています。このダイアログ・ボックスでは、「順序設定されたターゲット」が表形式で表示されます。表にはターゲット・オブジェクトという名前の列があります。中央右側の隅に、ターゲットを最上部に上げるボタン、ターゲットを1レベル上げるボタン、ターゲットを1レベル下げるボタン、ターゲットを最下部に下げるボタンの4つのボタンがあります。右下隅の表の下には「デフォルトにリセット」ボタンがあります。ダイアログ・ボックスの右下隅には、左から順に、「OK」ボタンと「取消」ボタンの2つのボタンがあります。

このスクリーンショットは、表演算子の「プロパティ・インスペクタ」を示しています。これには上から順に縮小可能な行オプション、TARGET_2、条件付きロード、およびERROR_TABLEが含まれます。TARGET_2には上から順にターゲット・ロード順序行、データ・ルール行、読取り専用キー行、バウンド名行、ロード・タイプ行、およびプライマリ・ソース行が含まれます。条件付きロードには、上から順に削除用ターゲット・フィルタ行、更新用ターゲット・フィルタ行、および制約による一致行が含まれます。ERROR_TABLEには上から順にエラー表名行、ロールアップの選択のエラー行、およびフィルタの選択のエラー行が含まれます。

このスクリーンショットは、演算子の同期化のための「調整」ダイアログ・ボックスを示しています。ダイアログ・ボックスの中央には同期化するオブジェクトを選択するためのリストがあります。リストの下には、上から順に「フラット・ファイルPAYROLL_WESTからフラット・ファイル演算子PAYROLLに同期」と「フラット・ファイル演算子PAYROLLからフラット・ファイルPAYROLL_WESTに同期」の2つのラジオ・ボタンがあります。ラジオ・ボタンの下の右隅には「拡張」ボタンがあります。ダイアログ・ボックスの左下隅には「ヘルプ」ボタンがあります。ダイアログ・ボックスの右下隅には、左から順に「OK」ボタンと「取消」ボタンがあります。

このスクリーンショットは、異なるワークスペース・オブジェクトからの同期化のための「調整」ダイアログ・ボックスを示しています。ダイアログ・ボックスの中央には同期化するオブジェクトを選択するためのリストがあります。リストの下には、上から順に「フラット・ファイルPAYROLL_WESTからフラット・ファイル演算子PAYROLLに同期」と「フラット・ファイル演算子PAYROLLからフラット・ファイルPAYROLL_WESTに同期」の2つのラジオ・ボタンがあります。ラジオ・ボタンの下の右隅には「拡張」ボタンがあります。ダイアログ・ボックスの左下隅には「ヘルプ」ボタンがあります。ダイアログ・ボックスの右下隅には、左から順に「OK」ボタンと「取消」ボタンがあります。

この図の最上位の行にはパラレル・セッションが示されています。その次の行にはパラレル・スレーブが示されています。3つ目の行にはDMLおよびDDLのパラレル化が示されています。最下位の行には、シリアライズ、および文がダウングレードされているかどうかを示されます。

これは、「I/Oタイプ」ページのスクリーンショットです。上部に「I/OタイプごとのI/O MB/秒」、下部に「I/OタイプごとのI/Oリクエスト数/秒」が示されています。

これは、パーティション評価後の「タスクの結果」ページの図です。ページの上部に「ワークロードの全体的なパフォーマンス」と示されています。その下に「改善の可能性」と示されています。2つのグラフが示されており、左側には元のコストが新しいコストの2倍であることを示す「ワークロードのI/Oコスト」のグラフ、右側には4倍、6倍または10倍以上の改善の可能性を示す「問合せ実行時間の向上」のグラフが表示されています。