Oracle® Warehouse Builder

トランスフォーメーション・ガイド

10*g* リリース 1（10.1）

部品番号 : B13520-02

2007 年 11 月

ORACLE®

Oracle Warehouse Builder トランスフォーメーション・ガイド , 10*g* リリース 1（10.1）

部品番号 : B13520-02

原本名 : Oracle Warehouse Builder Transformation Guide, 10*g* Release 1 (10.1)

原本部品番号 : B12151-02

原本著者 : Jean-Pierre Dijcks

原本協力者 : Shirinne Alison, Kavita Nayar, Padmaja Potineni

目次

3 SQL 変換

A 緩やかに変化する次元の使用

索引

はじめに

ここでは、次のトピックについて説明します。

目的

Oracle Warehouse Builder は、データの移動と変換、ビジネス・インテリジェンス・システムの開発と実装、メタデータの管理、Oracle データベースとメタデータの作成と管理などを担当するユーザーのために用意された総合的なツールセットです。このガイドでは、Warehouse Builder 変換の特徴であるファンクションとプロシージャについて説明します。

対象読者

このガイドは、次のようなデータ・ウェアハウス技術者を対象としています。

- ビジネス・インテリジェンス・アプリケーション開発者
- ウェアハウスのアーキテクト、設計者および開発者 — 特に、SQL 開発者と PL/SQL 開発者
- データ・アナリストおよび抽出、変換、ロードのルーチンを開発するユーザー
- データ・ウェアハウスを基盤とした大規模な製品の開発者
- ウェアハウス管理者
- システム管理者
- その他の MIS 専門家

このガイドの情報を利用するには、リレーショナル・データベース管理システムやデータ・ウェアハウスの設計に関する概念に慣れている必要があります。データ・ウェアハウスについては、『Oracle データ・ウェアハウス・ガイド』を参照してください。また、Oracle Database、SQL*Plus、SQL*Loader、Oracle Enterprise Manager、Oracle Workflow などの、Oracle リレーショナル・データベース・ソフトウェア製品の知識も必要です。

このマニュアルの構成

このガイドでは、データや関数の型に従って変換をグループ化して説明します。たとえば、日付に対して実行するすべての変換は、1 つの項にまとめられています。各項では、すべての関数がアルファベット順に記載され、効率的に検索できるようになっています。それぞれの変換の説明には、Warehouse Builder で使用される構文、変換の目的、一般的な使用例やビジネス上の使用例が含まれます。

Warehouse Builder には、PL/SQL で構築されたカスタム・ファンクションのセットが組み込まれており、開発者は標準的な作業を効率的に実行できるようになっています。これらの変換は接頭辞の WB_ で始まり、各章の後半にそれぞれの説明があります。

『Oracle Warehouse Builder トランスフォーメーション・ガイド』は、次の章と付録で構成されています。

- 第 1 章「Warehouse Builder での変換の概要」では、SQL 変換と PL/SQL 変換について概説します。
- 第 2 章「変換」では、マッピング・エディタで使用可能な組込み済の変換とマッピングでのそれらの定義方法について説明します。
- 第 3 章「SQL 変換」は、Warehouse Builder で使用できる PL/SQL プロシージャとファンクション、SQL 関数のリファレンスです。
- 付録 A「緩やかに変化する次元の使用」では、緩やかに変化する次元の様々なタイプについて概説します。また、ケース・スタディ・シナリオを通じて、Warehouse Builder を使用して緩やかに変化する次元の様々なタイプを設計および配布する方法について説明します。

10*g* リリース 1（10.1）の新機能

マッピング・エディタの拡張 : マッピング・デバッガ

マッピング・エディタでのマッピングに使用可能な拡張デバッグ機能が新しく用意されました。マッピング・デバッガを使用することで、マッピングの論理設計エラーを特定できます。この新機能では、ブレーク・ポイントとウォッチの設定やテスト・データの対話的な変更など、総合的なデバッグ機能を使用して、マッピングのデータ・フローをステップごとに実行できます。

複数ターゲットのサポートの拡張 : 相関コミット

このリリースでは、複数のターゲットを含むマッピングに新しいコミット方法が導入されました。前のリリースの Warehouse Builder では、独立したコミットが実行されていました。つまり、Warehouse Builder は、ターゲットごとに、他のターゲットに依存せずに、コミットとロールバックを実行していました。現行の Warehouse Builder では、このオプションに加えて、相関コミットも実行できます。Warehouse Builder は、すべてのターゲットを集合的に考慮し、すべてのターゲットで一律にデータのコミットやロールバックを行います。ソース内の各行が、すべての関連ターゲットに一律に影響することが重要である場合は、相関コミットを使用します。

ダイレクト・パーティション交換ロード

前のリリースの Warehouse Builder では、パーティション交換の前に追加処理が必要なマッピングでは、デフォルトで一時表が作成されました。これは、マッピングにリモート・ソースが含まれている場合や複数のソースが結合されている場合に発生しました。このリリースからは、一時表を作成しなくても、ソースを直接ターゲットに交換できるようになりました。以前実行したマッピングでロードしたファクト表を即時に公開するには、マッピングにダイレクト・パーティション交換ロードを使用してください。

データの品質に関する機能

- **複数の Name and Address ソフトウェア・プロバイダ :** このリリースから、Warehouse Builder は、動作保証された複数の Name and Address ソフトウェア・プロバイダと連携できるようになりました。サードパーティ・ベンダーに許諾された Name and Address ソフトウェアを Warehouse Builder で使用できます。これにより、プロジェクトにとって最適な Name and Address プロバイダを選択できるようになります。
- **Name and Address 演算子ウィザード :** 前のリリースでは、マッピング・キャンバスと演算子の「構成プロパティ」シートを使用して、Name and Address 演算子を定義しました。便宜性の向上を図り、Warehouse Builder では、ウィザードと演算子エディタを使用して、Name and Address 演算子を作成および編集できるようになりました。
- **Match-Merge 演算子 :** Warehouse Builder には、Oracle Pure Integrate で以前使用できたデータの品質に関する機能が組み込まれています。マッピング・エディタで使用可能な Match-Merge 演算子を使用して、レコードの照合とマージ用にビジネス・ルールを定義できます。Match-Merge 演算子と Name and Address 演算子を使用すると、Name and Address データで一意のハウスホールドを識別するプロセスがサポートされます。

メタデータ変更管理

前のリリースでは、OMB Plus スクリプト・ユーティリティを使用してメタデータ変更管理を実行できました。このリリースからは、Warehouse Builder Client ユーザー・インタフェースのメタデータ変更管理機能にもアクセスできます。メタデータ変更管理では、メタデータ・オブジェクトのスナップショットを取り、バックアップ管理と履歴管理に使用できます。スナップショットはナビゲーション・ツリーのすべてのオブジェクトでサポートされており、オブジェクト自身（表やモジュールなど）に関する情報だけでなく、オブジェクト内のオブジェクト（モジュール内の表など）に関する情報も保存できます。

Oracle Warehouse Builder 機能の拡張

- **セキュリティ :** Warehouse Builder では、高度なリポジトリ・セキュリティと監査オプションを、セキュリティ要件に応じて実装できるようになりました。高度なセキュリティ・オプションには、次のものが含まれます。

  プロアクティブ・セキュリティ : カスタマイズした PL/SQL セキュリティ実装パッケージを Warehouse Builder Design Repository にプラグインすることで、企業内で定義されているセキュリティ・ルールに基づいて、各ユーザーにアクセス制御を設定できるようになりました。

  リアクティブ・セキュリティ : Warehouse Builder では、メタデータの履歴に基づいて監査情報を追跡し、このような監査情報からセキュリティ・ポリシーを判断できます。

  データ管理 : Warehouse Builder では、技術管理者ではなく、特定の個人またはグループがメタデータの一部を所有できます。このため、メタデータの所有権は、メタデータ・セキュリティ管理の重要な要素になります。

- **RAC のサポート :** Warehouse Builder 10*g* リリース 1（10.1）では、RAC 機能のサポートが拡張され、実行時にネット・サービス名を使用できるようになりました。これにより、ランタイム環境を再構成せずに、クラスタ内のノードの保守を計画できます。また、ランタイム・サービスの可用性が向上しました。たとえば、サービス・インスタンスかその関連ノードのどちらかが障害を起こすかサービス停止になった場合、別ノードのランタイム・サービス・インスタンスが引き継ぐことができます。Warehouse Builder Design Repository も RAC クラスタ内で使用できますが、このリリースでは、RAC のフェイルオーバー機能は利用できません。

フラット・ファイル・サポートの拡張

- **ZONED データ型のサポート :** ゾーン 10 進データを含む固定形式のデータ・ファイルをロードできるようになりました。フラット・ファイル・サンプル・ウィザードで、インポートするフラット・ファイルに ZONED データ型を指定します。ZONED データの形式は、1 バイトにつき 1 桁の 10 進数の文字列であり、符号は最終バイトに組み込まれます（COBOL の場合は、SIGN TRAILING フィールドになります）。このフィールドの長さは、指定した精度（桁数）に等しくなります。小数点以下の桁数であるスケールを指定することもできます。

- **DECIMAL データ型のサポート :** DECIMAL データは、最終バイトを除くバイトごとに 2 桁のパック 10 進形式になっており、最終バイトには数字と符号が含まれます。DECIMAL データ型には精度とスケールが含まれるので、端数値を表現できます。

データベース接続性の拡張

Warehouse Builder では、データベースで共有できるパブリック・データベース・リンクを作成できるようになりました。リポジトリ所有者および `CREATE PUBLIC DATABASE LINK` 権限を持つユーザーが、パブリック・データベース・リンクを作成できます。

HP-UX と AIX のサポート

このリリースから、HP-UX プラットフォームおよび AIX プラットフォームで、Warehouse Builder を使用できるようになりした。前のリリースから使用可能になった UNIX プラットフォーム（Solaris および Linux）と Windows プラットフォーム（NT、2000 および XP）は、引き続き使用できます（OLAP Bridge 機能は Windows プラットフォームでのみ使用でき、Name and Address Server は Windows と Solaris プラットフォームでのみ使用できます）。

Public API

このリリースからは、Warehouse Builder に Public API が組み込まれました。API を利用するには、ローカル・マシンのフォルダに、次のファイルを解凍して抽出します。

```
<owb home directory>¥owb¥lib¥int¥pubapi_javadoc.jar
```

ファイル `index.html` をダブルクリックしてください。API の使用方法は、ヘルプ・リンクを選択して参照してください。

リリース 9.0.4 の追加機能

Oracle Warehouse Builder では、次の新機能が導入されました。

Warehouse Builder コンソールの変更

- **拡張ナビゲーション・ツリー :** Warehouse Builder コンソールに表示されるナビゲーション・ツリーが拡張されプロジェクト間のナビゲーションが向上し、メタデータ・リポジトリ・オブジェクトへの直接アクセスが容易になりました。以前は、一度に 1 つのプロジェクトしか表示できませんでしたが、すべてのプロジェクトをツリーに表示できるようになりました。また、プロジェクト・ノードを開き、アクティブなプロジェクトの内容を表示できるようになりました。モジュール・ツリーは、別ウィンドウに表示されなくなりました。
- **ウィザード、エディタ、プロパティ・シート :** すべての Warehouse Builder ウィザード、エディタおよびプロパティ・シートをナビゲーション・ツリーから起動できるようになりました。
- **ビジネス・エリアからコレクションに名前変更 :** 前のリリースでは、ウェアハウス・モジュール内にビジネス・エリアを作成して Warehouse Builder でオブジェクトを編成し、Oracle Discoverer などのツールにメタデータをエクスポートできました。このリリースからは、すべての機能でビジネス・エリアがコレクションに変更され、コレクションへのメタデータのインポートと、コレクションからのメタデータのエクスポートなどの機能が拡張されました。
- **ファクト表からキューブに名前変更 :** このリリースでは、ファクトおよびファクト表という用語が、OLAP 業界標準に合せて、キューブに名前変更されました。
- **論理名からビジネス名に名前変更 :** このリリースでは、オブジェクトの論理名という表現で参照されていたものが、すべてビジネス名に変更されました。
- **Warehouse Builder コンソールのツールバー :** ユーティリティ・ドローワは削除され、Warehouse Builder コンソールの横と上部のツールバーが上部にマージされ、最も重要な機能が 1 箇所に統合されました。

配布の拡張

- **配布管理オブジェクトの追加 :** このリリースでは、配布ソースおよびターゲットへの接続管理に使用できる、ロケーション、コネクタおよびランタイム・リポジトリ接続という 3 つのオブジェクトが導入されました。ロケーションは、配布の物理的なロケーションを定義します。コネクタは、ロケーション間の関係を定義します。ランタイム・リポジトリ接続は、Runtime Repository に関する情報を提供します。これらのオブジェクトを使用することで、1 つのターゲット設計に複数の配布ターゲットを作成できます。
- **単一の配布管理インタフェース :** デプロイメント・マネージャは、すべてのオブジェクトの配布および配布済マッピング、変換、プロセス・フローの実行を管理する単一のインタフェースを提供します。これまでに配布されたオブジェクトの履歴に直接アクセスすることもできます。デプロイメント・マネージャでは、このようなすべてのタスクを 1 つのインタフェースから実行できるだけでなく、Warehouse Builder ではランタイム・メタデータが追跡され、これまでに配布されたものの履歴を参照できるようになりました。

Warehouse Builder メタデータ・ブラウザの拡張

- **設計メタデータの参照 :** Warehouse Builder Design Browser が、外部表、ロケーション、コネクタなどのすべての新しい公開オブジェクトを含むように拡張されました。また、Design Browser はスタンドアロン環境でも実行可能になり、単独ユーザーの場合は、Oracle Application Server をインストールする必要がなくなりました。
- **ランタイム・メタデータの参照 :** Warehouse Builder Runtime Audit Viewer が、Web ベースのレポートを提供する Runtime Audit Browser に置換されました。Runtime Audit Browser では、前のリリースで使用できたものより広範な配布および実行監査レポートが提供されます。この監査データは、Runtime Repository に保存された情報から取得され、配布データと実行データの両方を含みます。

Warehouse Builder のプログラムで規定されたアクセスの拡張

- **Warehouse Builder Public API:** このリリースからは、Oracle Warehouse Builder 機能へのプログラムで規定されたアクセスに、新しく提供された Public API を使用できるようになりました。この API は Public Java API の完全なセットで、アプリケーション・プログラマが独自のアプリケーションに Warehouse Builder 機能およびサービスを組み込む際に使用できます。

- **Warehouse Builder スクリプト言語 :** Oracle MetaBase（OMB）Scripting Language では、Warehouse Builder グラフィカル・ユーザー・インタフェースにアクセスせずに、すべての Warehouse Builder 機能にアクセスできます。開発者は、Warehouse Builder のスクリプト・ユーティリティである OMB Plus を使用し、Warehouse Builder のメタデータと機能にアクセスできます。これにより、開発者は、Warehouse Builder をプログラムから使用でき、必要に応じて機能を拡張できます。OMB Scripting Language の詳細は、『Oracle Warehouse Builder スクリプト・リファレンス』を参照してください。

メタデータ管理の拡張

- **セキュリティ :** 各自のセキュリティ要件に応じて、オプションのリポジトリ・セキュリティと監査システムを実装できるようになりました。識別可能な複数のユーザーが同一の Warehouse Builder Design Repository にアクセスできるマルチ・ユーザー・アカウント・システムを作成できます。また、カスタマイズした PL/SQL セキュリティ実装パッケージを Warehouse Builder Design Repository にプラグインすることで、企業内で定義されているセキュリティ・ルールに基づいて、各ユーザーにアクセス制御を設定できます。

- **メタデータ変更管理（メタデータ・スナップショット）:** このリリースからは、メタデータ・オブジェクトのスナップショットを取り、バックアップ管理と履歴管理に使用できます。スナップショットはナビゲーション・ツリーのすべてのオブジェクトでサポートされており、オブジェクト自身（表やモジュールなど）に関する情報だけでなく、オブジェクト内のオブジェクト（モジュール内の表など）に関する情報も保存できます。

- **複数言語サポート（MLS）:** この機能を使用すると、表示するビジネス名と説明をリポジトリのベース言語以外の言語で格納できます。ビジネス名と説明を各種言語に翻訳し、ターゲット・ユーザーの移入言語を使用して EUL に配布できます。

- **ユーザー定義プロパティによる拡張性 :** Warehouse Builder OMB Plus スクリプト・ユーティリティを使用して、任意の Warehouse Builder オブジェクトに追加プロパティを定義できます。スクリプト・ユーティリティでユーザー定義プロパティを定義した後は、ユーザー・インタフェース、Oracle MetaBase（OMB）Scripting Language、Warehouse Builder Java API および Warehouse Builder Design Browser からユーザー定義プロパティにアクセスできます。これにより、Warehouse Builder の拡張性が強化され、他のビジネス・インテリジェンス製品との統合が容易になります。

- **Metadata Loader（インポートおよびエクスポート）の柔軟性の拡張 :** 2 つの機能が新しく追加され、この製品の柔軟性が拡張されました。1 つ目の機能では、コレクションからメタデータを直接エクスポート可能になりました。2 つ目の機能は、Metadata Loader コマンドライン・ユーティリティから使用できます。この機能を使用すると、ファーストクラス・オブジェクトのインポート時に適用するアクションのタイプを指定でき、柔軟性が向上します。

プロセス・フロー・エディタ

このリリースからは、Warehouse Builder に付属するプロセス・フロー・エディタを使用して、プロセス・フローを作成および定義できます。マッピングで以前定義した外部プロセス演算子は、ユーザー定義プロセスにアップグレードされ、プロセス・フロー・モジュールに含まれます。プロセス・フローは 1 つの Warehouse Builder 設計環境に統合され、Oracle Workflow 設計クライアントを使用してこの機能を実行する必要はなくなりました。Warehouse Builder プロセス・フロー・エディタはマッピングのセマンティックをネイティブに理解できるため、FTP や電子メールなどのアクティビティをモデル化できます。

パフォーマンスの改善

- **マッピング・ユーザー・インタフェース :**「マッピング」という名前の定義済表示セットが、このリリースで新しく追加されました。この表示セットを選択すると、効果的にマッピングまたは使用されている列のみが、マッピングに表示されます。
- **マッピング圧縮 :** この機能では、特定のマッピングで使用されていない演算子と属性間の接続が自動的に検出され、リポジトリから削除されます。これにより、大量のデータ・フローを要する大規模マッピングのロードと保存のパフォーマンスが劇的に向上します。
- **Metadata Loader（インポートおよびエクスポート）:** インポートおよびエクスポート機能では、新しい圧縮機能を個々のマッピングに対して使用できます。つまり、Metadata Loader では、実際に使用されているマッピング・オブジェクトにのみエクスポートとインポートが実行されます。

Oracle Database との統合

- **OLAP 統合 :** Warehouse Builder では、様々なデータ・ソースから、多次元 OLAP オブジェクトを ROLAP モデルまたは MOLAP モデルとして設計、配布およびロードできます。データがロードされたら、BI ツールとアプリケーションを使用して、ビジネス上の疑問に答える、複雑な分析問合せを実行できます。Warehouse Builder を使用すると、1 つのキューブ設計とディメンション設計から、リレーショナル・オブジェクトと多次元オブジェクトの両方を作成および管理できます。
- **アドバンスト・キュー（AQ）統合 :** Warehouse Builder では、データ・ウェアハウスの設計時に、アドバンスト・キュー定義をインポートし、AQ をデータ・ソースおよびターゲットとして使用できます。アドバンスト・キュー機能を Messaging Gateway と組み合せると、Warehouse Builder データ・ソースとして、MQ Series と Tibco 上のメッセージ・アプリケーションをサポートできます。AQ では、ソース・システムからターゲットに、取得した変更データを伝搬することもできます。AQ を統合する機能は、今後、リアルタイム・データ・ウェアハウスを提供する基礎になります。
- **外部表 :** このリリースからは、外部表を使用して、リレーショナル以外のファイル・ソースのデータをリレーショナルの読取り専用形式で表現できます。Oracle Database から、既存の外部表をインポートできます。また、フラット・ファイル定義に基づいて、Warehouse Builder 内に外部表を作成することもできます。Warehouse Builder では、外部表を Oracle Database に配布する適切な DDL が自動生成されます。
- **Oracle Database マルチ・テーブル・インサート :** Warehouse Builder では、Oracle Database 機能が利用され、ターゲット・データベースが Oracle Database の場合は、マルチ・テーブル・インサートが生成されます。これにより、1 回の操作で複数の表にデータを挿入するように、マッピングを最適化できます。
- **Oracle Database テーブル・ファンクション :** Warehouse Builder では、テーブル・ファンクション演算子が導入され、ターゲット・システムへのロード時のパフォーマンスを改善できるようになりました。この演算子を使用して、入力行のセットを操作して、カーディナリティが異なる別の行セットを返すことができるカスタム・コードを開発します。従来のファンクションとは異なり、テーブル・ファンクションでは、物理表のように問い合せることができる行セットが出力されます。

フラット・ファイル・サポートの拡張

- **ターゲットと非バインドのフラット・ファイル :** このリリースでは、マッピングの作成時に、新しい非バインドのフラット・ファイル・オブジェクトを作成できます。Warehouse Builder では、カンマ区切りの単一レコードによる新しいタイプのフラット・ファイルが、指定したロケーションに作成されます。この機能により、リレーショナル・オブジェクトの内容をフラット・ファイルに、より容易にロードできます。
- **フラット・ファイルのアウトバウンド調整 :** アウトバウンド調整では、新しいリポジトリ・オブジェクトをマッピング・フラット・ファイルから作成できます。その結果、フラット・ファイルがそのリポジトリで新規である場合、カンマ区切りのファイルが指定したロケーションに新しく作成されます。この機能により、リレーショナル・オブジェクトの内容をフラット・ファイルに迅速にダンプすることが、より容易になります。

- **デリミタ付きファイルの論理レコード :** フラット・ファイル・サンプル・ウィザードも拡張され、改善されたユーザー・インタフェースを使用して、デリミタ付きファイルの論理レコードを定義できるようになりました。
- **位置ベースのマスター / ディテールのロード :** 位置ベースのマスター / ディテール・フラット・ファイルが、追加されたマッピング演算子を使用して簡単にロードできるようになりました。
- **SQL プロパティの拡張 :** Warehouse Builder にインポートするフラット・ファイルの SQL プロパティを指定できるようになりました。フラット・ファイル・フィールドごとに、SQL プロパティ値を事前定義できます。このため、フラット・ファイル・ソースをリレーショナル・ターゲットにマッピングする場合、これらの事前定義済 SQL プロパティ値が、ターゲット列のデフォルトになります。これらの値は、リレーショナル・ターゲット列の構築時または外部表の列の作成時に使用されます。

マッピング・エディタの拡張

- **マッピング・ユーザー・インタフェース :** プロパティ関連のタブ・セットが新しくなり、マッピング演算子と属性プロパティを迅速に作成および編集できるようになりました。
- **ピボット演算子とアンピボット演算子 :** このリリースからは、ピボット演算子とアンピボット演算子をマッピングに追加できます。ピボット演算子では、属性の 1 行を複数の行に変換できます。アンピボット演算子では、複数の入力行が 1 つの出力行に変換されます。
- **Name and Address 演算子の拡張 :** Name and Address 演算子が拡張され、新しい入力ロールと出力属性が組み込まれました。このリリースからは、United States Postal Service Code Accuracy Support System（CASS）レポートもサポートされます。

Warehouse Builder UNIX プラットフォームのサポート

このリリースから、UNIX プラットフォーム（Solaris および Linux）と Windows プラットフォーム（NT、2000 および XP）で Warehouse Builder を使用できるようになりました。Warehouse Builder のすべてのコンポーネントを使用できますが、このリリースでは、Name and Address ライブラリは Linux では使用できません（OLAP Bridge 機能は Windows プラットフォームでのみ使用でき、Name and Address Server は Windows と Solaris プラットフォームでのみ使用できます）。

表記規則

このマニュアルでは、Windows NT、Windows 2000 および Windows XP の各オペレーティング・システムを総称して Windows と記載します。Oracle Database の SQL*Plus インタフェースは、SQL と記載する場合があります。

例では、特に明記しないかぎり、各行末に改行が挿入されています。そのため、入力の行末で [Enter] キーを押す必要があります。

次の表に、その規則と使用例を示します。

規則	意味
. . .	垂直の省略記号は、例に直接関連しない複数の行が省略されていることを示します。
...	文またはコマンド中の横の省略記号は、その例に直接関係のない文やコマンドの一部が省略されていることを示します。
太字	テキスト中の太字は、インタフェースのボタンやリンクを示します。太字は主題を強調する目的でも使用します。
`Unicode` テキスト	Unicode テキストは、コード自体、ファイルのディレクトリや名前、リテラル・コマンドを示します。
イタリックの `Unicode` テキスト	イタリックの Unicode テキストは、ユーザーが値を指定するパラメータを示します。

規則	意味
[]	大カッコは、カッコ内の項目を任意に選択することを表します。 大カッコは、入力しないでください。

関連ドキュメント

次のマニュアルがあります。

- 『Oracle Warehouse Builder ユーザーズ・ガイド』
- 『Oracle Warehouse Builder インストレーションおよび構成ガイド』
- 『Oracle Warehouse Builder トランスフォーメーション・ガイド』
- 『Oracle Warehouse Builder スクリプト・リファレンス』
- 『Oracle Warehouse Builder リリース・ノート』

Warehouse Builder ドキュメント・セットに加えて、『Oracle データ・ウェアハウス・ガイド』も参照できます。

1

Warehouse Builder での変換の概要

データの変換は、抽出、変換およびロード（ETL）ツールの持つ主要な機能の 1 つです。データ変換では、単一値ベースの算術計算から複合ベースのデータ変換までの広範囲に及ぶデータ操作やデータ変更が必要になります。Oracle Warehouse Builder には、複数の事前定義済変換および事前定義済ファンクションとプロシージャのライブラリがデータ変換用に組み込まれています。また、Warehouse Builder では、SQL と PL/SQL を変換言語として使用することで、Oracle Database のエンジンが持つ能力を最大限に活用できます。

このガイドでは、Warehouse Builder 変換の特徴であるファンクションとプロシージャについて説明します。このガイドは基本的なユーザー・マニュアルではないため、読者はマッピングの作成方法および変換と演算子のマッピングへの追加方法を理解している必要があります。また、このガイドでは、カスタムなどの概念については説明されません。

この章では、次のトピックについて説明します。

詳細は、『Oracle Warehouse Builder ユーザーズ・ガイド』を参照してください。

概要

オラクル社は、総合的な ETL ソリューションを提供します。このソリューションの主要コンポーネントは ETL ツールの Oracle Warehouse Builder です。Warehouse Builder には、ETL プロセスのモデル化と作成を可能にするコンポーネントが用意されています。開発者は Warehouse Builder の直観的なユーザー・インタフェース（UI）を使用して、オープン・リポジトリに保存されるオブジェクトを設計および定義できます。初期設計が完了したら、この設計をランタイム・プラットフォームに配布できます。Oracle Database がランタイム・プラットフォームとなるため、Warehouse Builder は ETL エンジン・ベースのツールとしてではなく、コード生成ツールとして使用されます。Warehouse Builder には、リポジトリでレポートを実行するレポート・ツールも組み込まれています。また、他の Oracle 問合せツールと統合することもできます。

SQL と PL/SQL は用途が広く定評のある言語として、多くの情報専門家によって広く使用されているため、Warehouse Builder を使用することで、他の変換言語を開発する時間と費用を削減できます。Warehouse Builder を使用すると、既存の知識および定評があってオープンで標準的な技術を使用してソリューションを作成できます。

Warehouse Builder でのデータ変換

Warehouse Builder で設計した ETL プロセスは、PL/SQL パッケージに変換できます。変換した PL/SQL パッケージを Oracle Database に配布し、実行可能なパッケージとして保存します。

PL/SQL を使用して、ソースからターゲットに移動するデータを変換することもできます。ウェアハウス・ソリューションを迅速に開発できるようにするため、Warehouse Builder には、PL/SQL で記述されたカスタム・プロシージャとファンクションが用意されています。Warehouse Builder では、PL/SQL の再利用に加えて、独自の PL/SQL 変換も作成できます。最終的なプロセスは Oracle Database で実行されるため、Warehouse Builder では、Oracle Database でサポートされるすべての構成メンバーがサポートされます。

Warehouse Builder マッピング・エディタでは、SQL コード・コンポーネントを使用するデータ変換を設計することもできます。たとえば、異種データ・ソースを結合したり、データ・ストリームを複数の出力ストリームに分割するようなアクティビティを SQL コンポーネントとして実装できます。これにより、ソースからターゲットにデータを移動する SQL コードが生成され、効率的なソリューションを開発できます。

SQL 規格

オラクル社は、業界で認められた規格に準拠するように努め、SQL 標準化委員会に積極的に参加しています。業界で認められた委員会には、ANSI（米国規格協会）と ISO（国際標準化機構）が含まれます。ISO は、IEC（国際電気標準会議）と連携しています。ANSI と ISO/IEC は、ともに、リレーショナル・データベースの標準言語として SQL を認めています。新しい SQL 規格がこれらの組織によって同時に公開される際は、組織によって使用される表記規則に従ってこれらの規格に名前が付けられます。命名規則は異なる場合がありますが、規格が技術的に同一であることに注意してください。

最新の SQL 規格は、1999 年 7 月に採用され、SQL:99 と呼ばれています。この規格の正式名称は、次のとおりです。

- ANSI X3.135-1999, "Database Language SQL", Parts 1 ("Framework"), 2 ("Foundation"), and 5 ("Bindings")
- ISO/IEC 9075:1999, "Database Language SQL", Parts 1 ("Framework"), 2 ("Foundation"), and 5 ("Bindings")

SQL の動作方法

SQL は、Oracle などのリレーショナル・データベースにインタフェースを提供するデータのサブ言語です。すべての SQL 文は、データベースに対する命令です。アプリケーション・プログラマ、データベース管理者、管理者、エンド・ユーザーなどの様々なタイプのユーザーが SQL を使用しています。

SQL 言語には、次のような特徴があります。

- データのセットを、個別単位ではなくグループとして処理します。
- データを自動的にナビゲートできます。
- 複雑で強力なスタンドアロン型の文を使用します。

フロー制御文は、SQL の一部ではありませんでしたが、SQL の最近認められたオプション部分 ISO/IEC 9075-5: 1996 に含まれています。フロー制御文は永続保存モジュール（PSM）として一般的に知られており、SQL の拡張機能である Oracle の PL/SQL は PSM に類似しています。

SQL では、論理レベルでデータを処理できます。実装の詳細は、データの操作時にのみ必要になります。たとえば、表から行のセットを取得するには、条件を定義して行をフィルタ処理できます。条件を満たすすべての行が 1 ステップで取得され、ユーザー、別の SQL 文またはアプリケーションに 1 単位として渡されます。行を 1 つずつフィルタ処理したり、データの保存と取得を手動で行う必要はありません。すべての SQL 文に対してオプティマイザが使用されます。これは、Oracle の一部であり、特定データに最も効率的にアクセスする方法が判断されます。Oracle では、オプティマイザのパフォーマンスの改善に使用できる技法も提供されます。

SQL には、次のような様々なタスクを実行する文があります。

- データの問合せ
- 表に対する行の挿入、更新および削除
- オブジェクトの作成、置換、変更および削除
- データベースとそのオブジェクトへのアクセスの制御
- データベースの一貫性と整合性の保証

SQL では、これらのすべてのタスクが、1 つの一貫した言語に統合されています。

リレーショナル・データベースの共通言語としての SQL

すべての主要なリレーショナル・データベース管理システムでは SQL がサポートされており、SQL で作成されたすべてのプログラムは、わずかに修正するだけで、あるデータベースから別のデータベースに移植できます。

つまり、組織に蓄えられたすべての SQL の知識は、Warehouse Builder に完全に移植できます。Warehouse Builder を使用して、既存の複雑なカスタム・コードをインポートし保守できます。このようなカスタム変換は、後で Warehouse Builder マッピングで使用できます。

2

変換

Warehouse Builder マッピング・エディタには、事前作成済の変換演算子が数多く用意されています。これらの演算子を使用して、ソースからターゲットへのデータの移動方法を定義するときの標準的な変換を定義できます。

変換演算子とは、事前に作成された PL/SQL ファンクション、プロシージャ、パッケージ・ファンクションおよびパッケージ・プロシージャです。これらは入力データを受け取って操作を実行し、出力データを生成します。

この章では、次のトピックについて説明します。

関連情報は、『Oracle Warehouse Builder ユーザーズ・ガイド』を参照してください。

標準 SQL 演算子

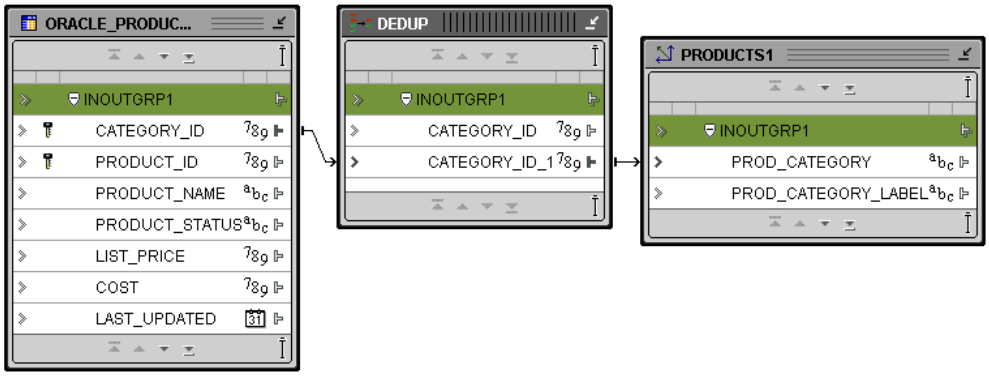
SQL は、データの抽出と変換を行う強力なメカニズムです。SQL では、異種ソースを 1 つのデータ・ストリームに結合して、それを必要に応じて変換し、さらには複数ストリームに分割して複数ターゲットに出力するというような一連の処理を実行できます。Warehouse Builder では、標準 SQL 演算子を使用して、これらのアクティビティを実行します。

デュプリケータ解除演算子（DISTINCT）

多くの場合、ソース・データには重複値が含まれます。たとえば、ディメンション内の高位レベルは粒度の低いレベルで存在するため、ソース内で重複します。このような行の選択時に必要になるのは、複数行ではなく、レベル・レコードごとに 1 行のみを選択することです。

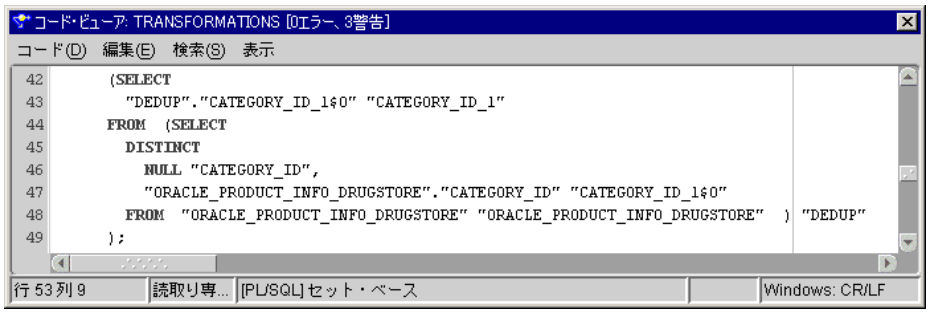
これを可能にするには、図 2-1 のように、デュプリケータ解除演算子をマッピングに追加します。ターゲットで必要となるすべての行は、デュプリケータ解除演算子を通過する必要があります。行セットがデュプリケータ解除演算子を迂回して、ターゲットに直接ヒットすることはありません。

図 2-1 マッピングでのデュプリケータ解除演算子



デュプリケータ解除演算子では、図 2-2 のように、生成される抽出問合せに DISTINCT キーワードが生成されます。

図 2-2 DISTINCT キーワードへの変換



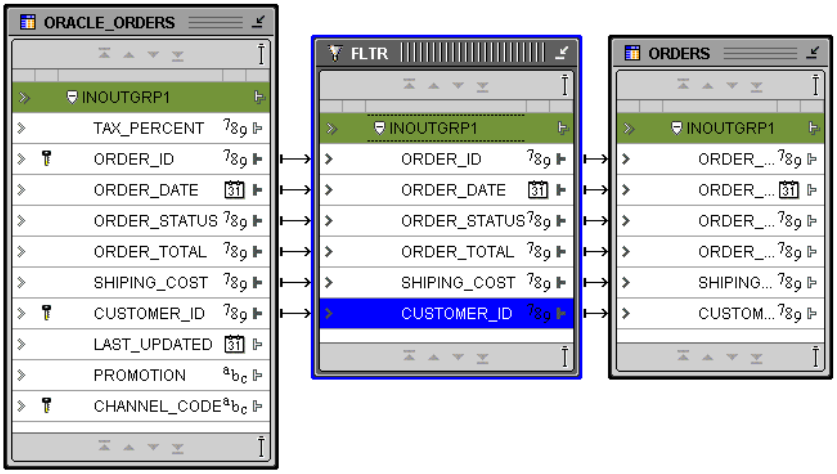
フィルタ（WHERE）

フィルタでは、問合せソースから選択されるデータ・セットを制限できます。フィルタ処理では、データ・セットに適用される句に基づいて、抽出や処理が行われる行数が制限されます。このフィルタ句は、サポートされているすべてのデータ型に基づき、定数を含むこともできます。

図 2-3 では、注文ステータスを使用して注文が制限されます。注文は記帳される必要があり、その最新の更新日付が抽出日付になる必要があります。結果は切り捨てられ、タイムスタンプがない日付に対して照合されたことが保証されます。したがって、ロードする日に修正された（つまりステータスが記帳済に設定された）記帳済注文のみがロードされます。これが、変更データの取得を実装する簡単な方法です。

ターゲットで必要となるすべての行は、フィルタ演算子を通過する必要があります。行セットがフィルタ演算子を迂回して、ターゲットに直接ヒットすることはありません。

図 2-3 マッピングでのフィルタ



マッピングにフィルタを定義したら、図 2-4 のように、Expression Builder を使用してフィルタ句を指定する必要があります。フィルタ条件は複雑になることがあるため、Expression Builder を使用してフィルタ句を検証してから、ターゲット・システムに配布してください。

図 2-4 Expression Builder でのフィルタ句の定義

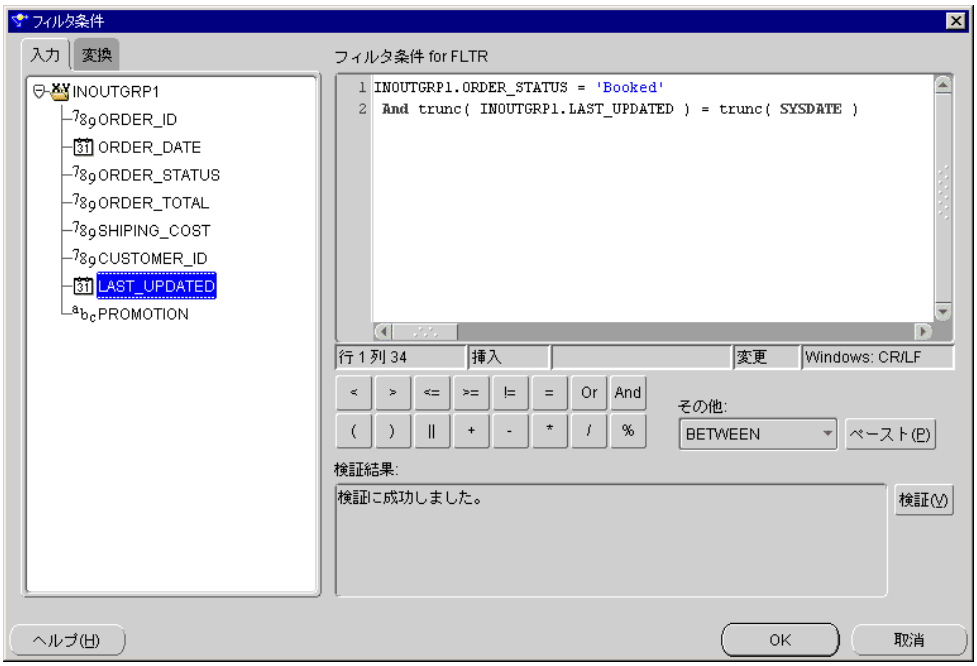
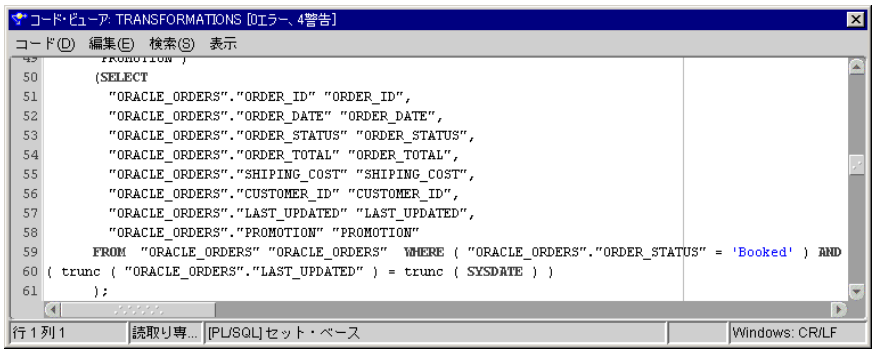


図 2-5 のように、抽出プログラムでは、フィルタ句は WHERE 句に翻訳されます。

図 2-5 フィルタ演算子用に生成されたコード



生成された抽出コードにはオブジェクトの物理列名が含まれますが、Expression Builder には相対名が表示されます。論理モデリング・フェーズでビジネス名を使用している場合、Warehouse Builder では、ビジネス名が実際のデータベース・オブジェクトの物理名に自動的に変換されます。

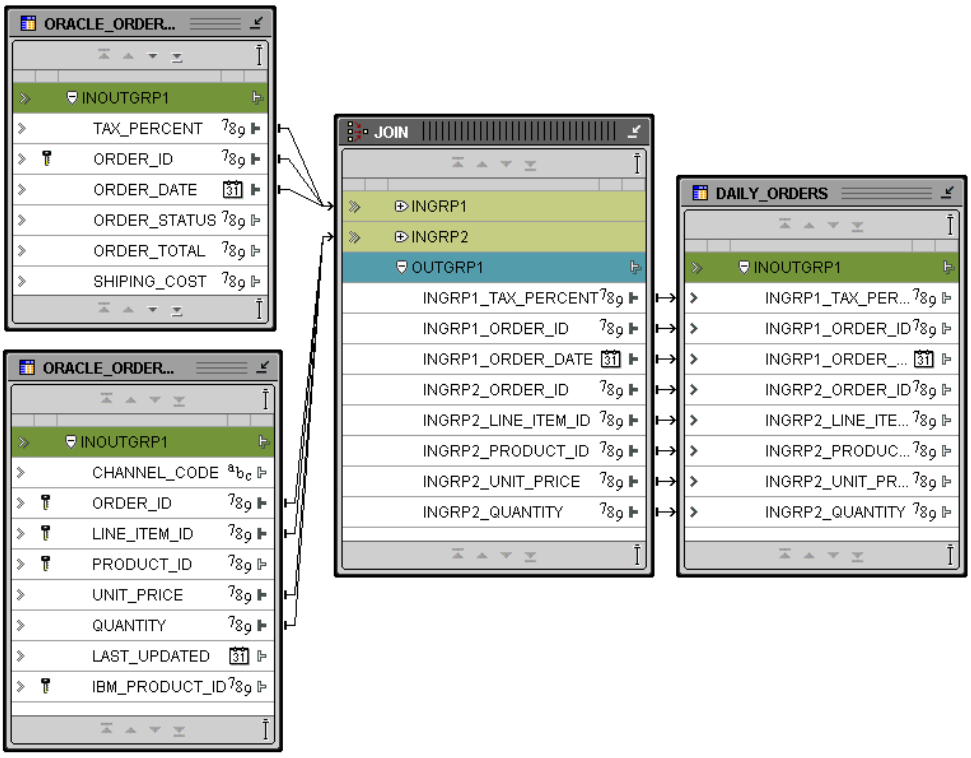
結合子（完全外部結合）

結合とは、複数の表、ビューまたはマテリアライズド・ビューの行を組み合せる問合せのことです。Oracle では、問合せの FROM 句に複数の表が指定されている場合に結合が実行されます。問合せの選択リストには、指定されている表の任意の列を選択できます。

ほとんどの結合問合せには WHERE 句条件が含まれ、異なる表にある 2 つの列が比較されます。Oracle では、結合を実現するために、結合条件が TRUE に評価された 2 行がペアとなり、各表にあるそれぞれの行が結合されます。結合問合せの WHERE 句には、結合条件に加えて、1 つの表のみの列を参照する別の条件を含めることもできます。この条件では、結合問合せによって返される行をさらに制限できます。

Warehouse Builder の結合子では、内部結合、外部結合、自己結合、等価結合および非等価結合もサポートされます。Oracle9*i* Database で実行する場合は、完全外部結合もサポートされます。キー参照演算子は、Warehouse Builder で使用される外部結合の例です。結合の詳細は、『Oracle Database SQL リファレンス』を参照してください。

図 2-6 関連する 2 表の 1 つの結果セットへの結合



Warehouse Builder Design Repository のメタデータに主キーと外部キーの関係が含まれる場合、Warehouse Builder では、その情報に基づいて、結合条件が事前に移入されます。図 2-6 では、2 つのソース表が結合され、正規化されている注文表セットのデータが組み合されて 1 つの表に挿入されます。注文行は一般的にカーディナリティが高いため（注文行の表には、注文ごとに多くのレコードが存在するので）、結果セットでもカーディナリティが高くなります。

結合問合せの 2 つの表に結合条件が含まれない場合、Warehouse Builder では、デカルト積が返され、一方の表の各行がもう一方の表の各行にそれぞれ組み合されます。デカルト積では常に多くの行が生成されるので、効果的でない場合があります。たとえば、それぞれ 100 行を含む 2 表のデカルト積では、10,000 行が含まれます。デカルト積が必要でない場合は、結合条件を常に組み込む必要があります。

図 2-7 主キー / 外部キー関係に基づく結合条件

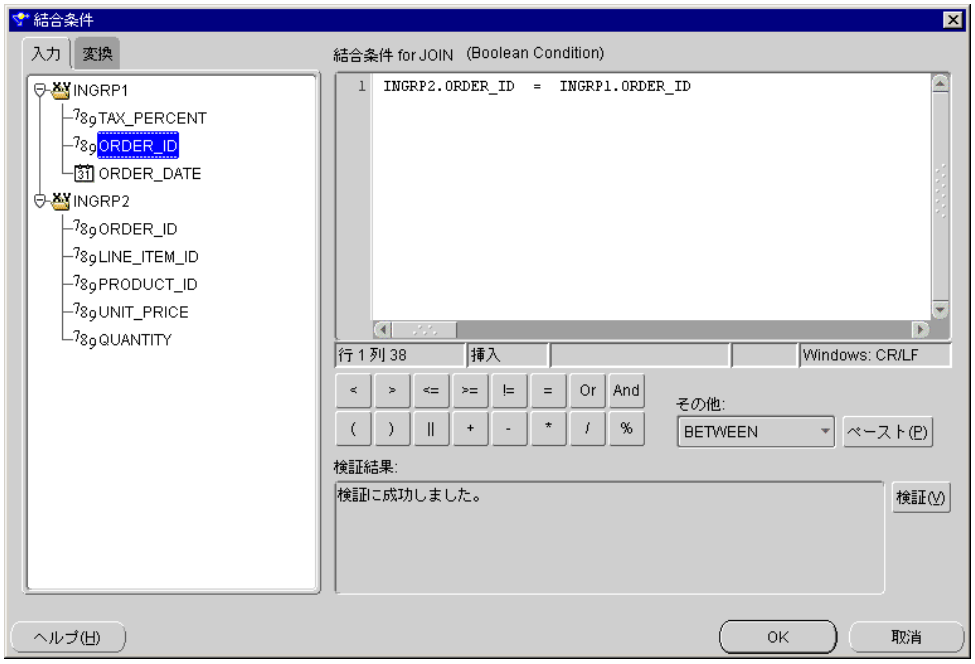
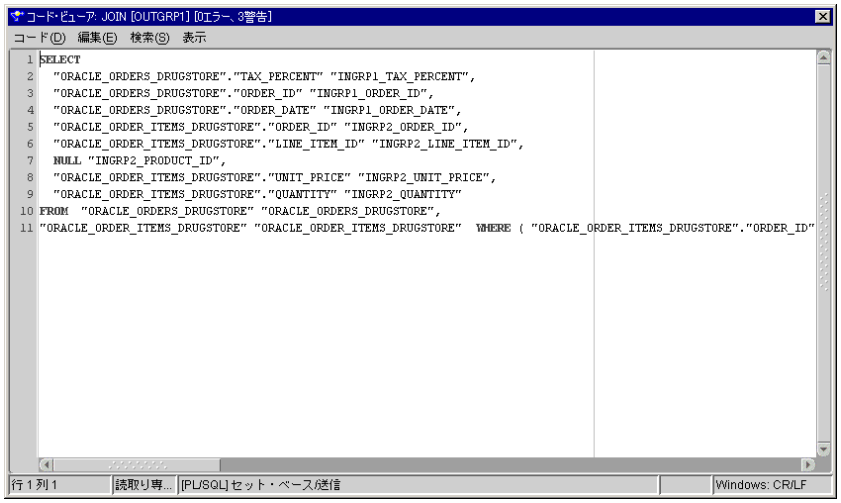


図 2-7 のように、Expression Builder を使用すると、無限に複雑な結合句を定義できます。Warehouse Builder では、図 2-8 のように、生成された SQL コードでこれらの句が WHERE 句に変換されます。この例では、FROM 句に 2 つのソース表が指定され、WHERE 句によって、order_id 列を基にこの両表が結合されます。

図 2-8 結合演算子用に生成されたコード

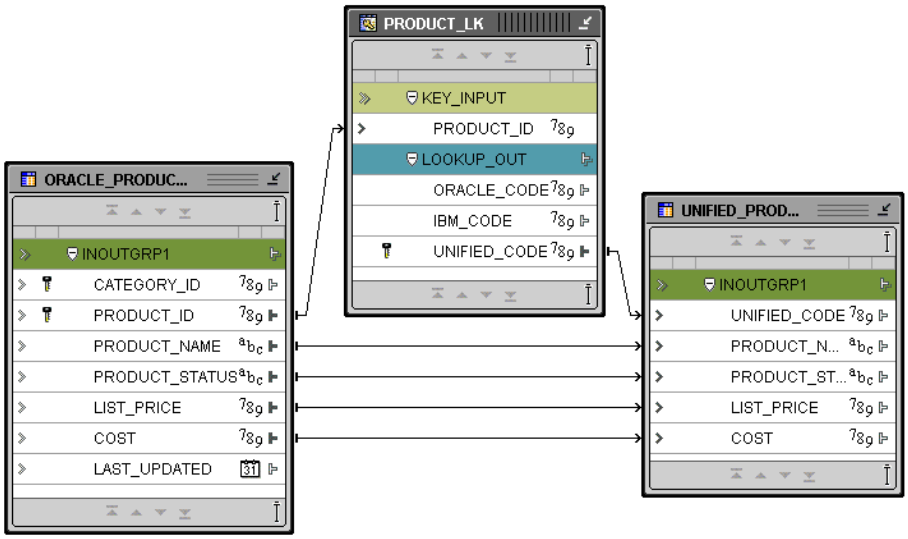


キー参照

データ・ウェアハウスの設計では、サロゲート・キーを使用するかどうかを一般的に判断します。サロゲート・キーは通常は整数で、ソース・システムの大きな論理識別子を置換する際に使用します。サロゲート・キーを使用すると、主キーにリンクしている表により使用される領域を減らしたり、ソース固有の識別子を一般的な識別子で置き換えてレポートを可能にすることができます。数値サロゲート・キーの作成方法の詳細は、2-10 ページの「順序（CURRVAL、NEXTVAL）」を参照してください。

サロゲート・キーを使用すると、ソースとターゲットの間でレコードの 1 次識別子の不整合が発生します。ターゲットでの結合が正しいデータで実行されることを保証するには、キー参照を実行する必要があります。キー参照では、論理キーがサロゲートの論理キーに変換されます。図 2-9 は、マッピングでのキー参照の例です。

図 2-9 マッピングでのキー参照



Warehouse Builder には、キー参照を実行する方法が 2 通りあります。事前定義済の PL/SQL 変換を使用する方法と SQL 演算子を使用する方法です。SQL 演算子では、表、ビュー、マテリアライズド・ビュー、ディメンションなどの参照オブジェクトが、元の識別子を含む表に結合されます。

参照条件では、結合句を手動で作成する必要がありません。専用の UI により、図 2-10 のように、結合句を指定できます。キー参照演算子の出力属性ごとに、デフォルトを指定できます。

図 2-10 検索条件の定義

過度な制限とソース行の欠落を避けるため、参照表は演算子によって外部結合され、すべてのソース行が問合せの一部になります。参照値を使用できない場合、演算子は NULL 値を返します。問合せで NVL を使用するデフォルト値でこれを置き換えると、NULL をデフォルトで置き換えることができます。

図 2-11 外部結合と NVL で生成されるコード

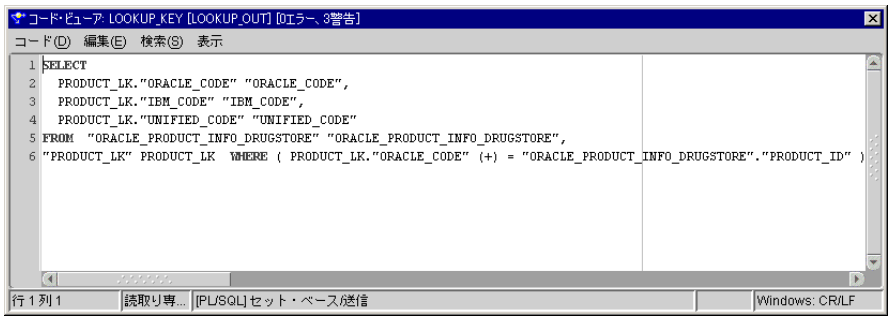


図 2-11 では、参照表である PRODUCT_LK 表で外部結合が実行されています。UNIFIED_CODE の NULL を返す行は、NVL 関数によって置換され、-1 の値が反映されます。このため、デフォルト値が、システム生成コードを妨害することはありません。詳細は、3-58 ページの「NVL」を参照してください。

ピボット演算子

ピボット演算子では、属性の 1 行を複数の行に変換できます。行の代わりに属性に含まれるデータを変換する場合は、マッピングでこの演算子を使用します。たとえば、クロス集計フォーマットのデータなど、リレーショナル以外のデータ・ソースからデータを抽出する場合にこの演算子を使用します。

例 : 売上データのピボット

外部表 SALES_DAT には、フラット・ファイルのデータが含まれます。このデータには、販売担当者ごとに 1 行が含まれており、月ごとに列が分割されています。外部表の詳細は、『Oracle Warehouse Builder ユーザーズ・ガイド』を参照してください。図 2-12 は、クロス集計フォーマットになっている SALES_DAT のデータです。

図 2-12 SALES_DAT

ID	Reg	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
0675	4	10.5	11.4	9.5	8.7	7.4	7.5	7.8	9.7				
0676	3	9.5	10.5	10.3	7.6	8.0	7.8	8.7	8.9				
0679	3	8.7	7.4	7.5	7.8	9.7	10.3	7.6	8.0				
0683	2	9.5	10.5	10.3	9.5	8.7	7.4	7.8	8.7				
0684	1	11.4	9.5	8.7	7.4	7.5	10.3	9.5	8.7				
0687	1	9.5	8.7	7.4	7.8	8.7	7.4	7.5	7.8				
0690	1	8.7	7.4	7.8	8.7	11.4	9.5	8.7	7.4				

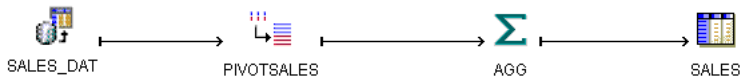
表 2-1 は、Warehouse Builder でピボット操作を実行した後のデータの例です。これまで複数の列（M1、M2、M3...）に含まれていたデータが、1 つの属性（MONTHLY_SALES）に変わっています。SALES_DAT の 1 つの ID 行が、ピボット後のデータでは 12 行に該当します。

表 2-1 ピボット操作後のデータ

REP	MONTH	MONTHLY_SALES	REGION
0675	Jan	10.5	4
0675	Feb	11.4	4
0675	Mar	9.5	4
0675	Apr	8.7	4
0675	May	7.4	4
0675	Jun	7.5	4
0675	Jul	7.8	4
0675	Aug	9.7	4
0675	Sep	NULL	4
0675	Oct	NULL	4
0675	Nov	NULL	4
0675	Dec	NULL	4

この例のピボット変換を実行するには、図 2-13 のようなマッピングを作成します。

図 2-13 マッピングでのピボット演算子



このマッピングでは、外部表からデータが読み取られ、データがピボットおよび集計され、セット・ベースのモードでターゲットに書き込まれます。ピボット後、ターゲットにデータを直接ロードする必要はありません。ターゲット演算子にデータを向ける前後に、一連の演算子の中でピボット演算子を使用できます。フィルタ、結合子、集合演算などの演算子を配置してから、ピボット演算子を配置できます。Warehouse Builder のピボット済データは行ごとの操作ではないので、セット・ベースのモードでマッピングを実行することもできます。

順序（CURRVAL、NEXTVAL）

順序演算子では、データベースから順序番号を生成できます。本来のセマンティックの意味を含まない識別子を作成することもできます。このような識別子は、多くの場合、サロゲート・キーと呼ばれます。この場合は、Warehouse Builder 内でキー参照を使用して元の値を参照し、この生成されたキーを演繹できます。

順序番号を生成する場合、トランザクションのコミットかロールバックのどちらを行ったかに関係なく、順序演算子の値は増加します。2 人のユーザーが、同じ順序演算子の値を同時に増加した場合、それぞれのユーザーが取得する順序番号では、別のユーザーも順序番号を生成しているので、ギャップが生じることがあります。片方のユーザーは、別のユーザーによって生成された順序番号を取得できません。ユーザーが順序値を生成すると、そのユーザーは、順序演算子の値が別のユーザーによって増加されていても、その値にアクセスし続けることができます。

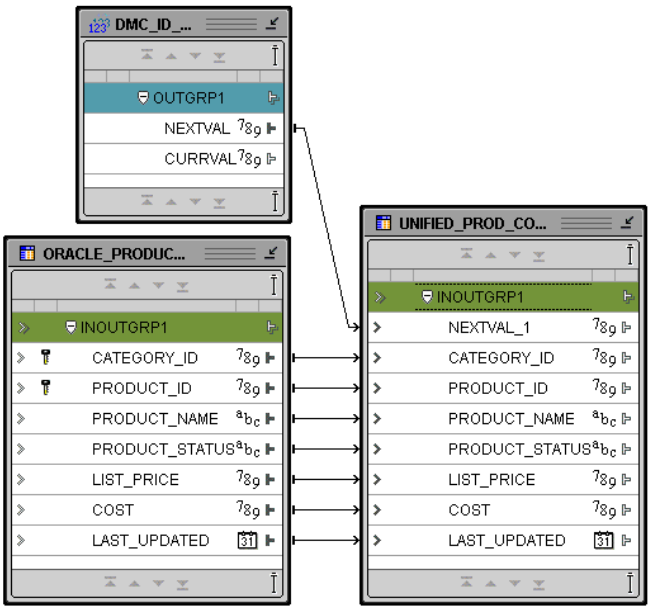
順序番号は、表に依存せずに生成されるので、1 つ以上の表に同じ順序演算子を使用できます。後でロールバックされたトランザクションで生成して使用された、それぞれの順序番号は、スキップされたようになることがあります。また、1 人のユーザーは、別のユーザーが同じ順序演算子を使用していることを認識できません。

SQL 文の順序値には、順序演算子の現行値を返す CURRVAL 擬似列、または順序演算子の値を増加して新しい値を返す NEXTVAL 擬似列でアクセスできます。

注意： CURRVAL 機能を使用するには、NEXTVAL を先にコールする必要があります。

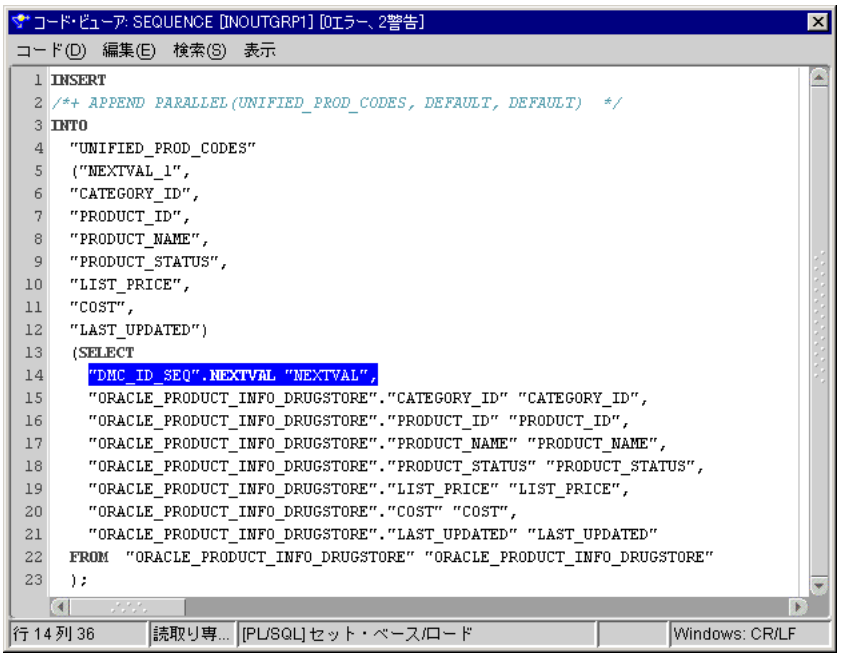
図 2-14 は、Warehouse Builder マッピングで順序演算子を使用する方法を示しています。

図 2-14 サロゲート・キーの生成



この例では、Warehouse Builder の順序オブジェクトで、擬似列 CURRVAL と NEXTVAL を使用でき、適切な列をマッピングに選択できます。NEXTVAL 列は、図 2-15 のように、挿入文の生成で一般的に使用します。

図 2-15 順序演算子用に生成されたコード

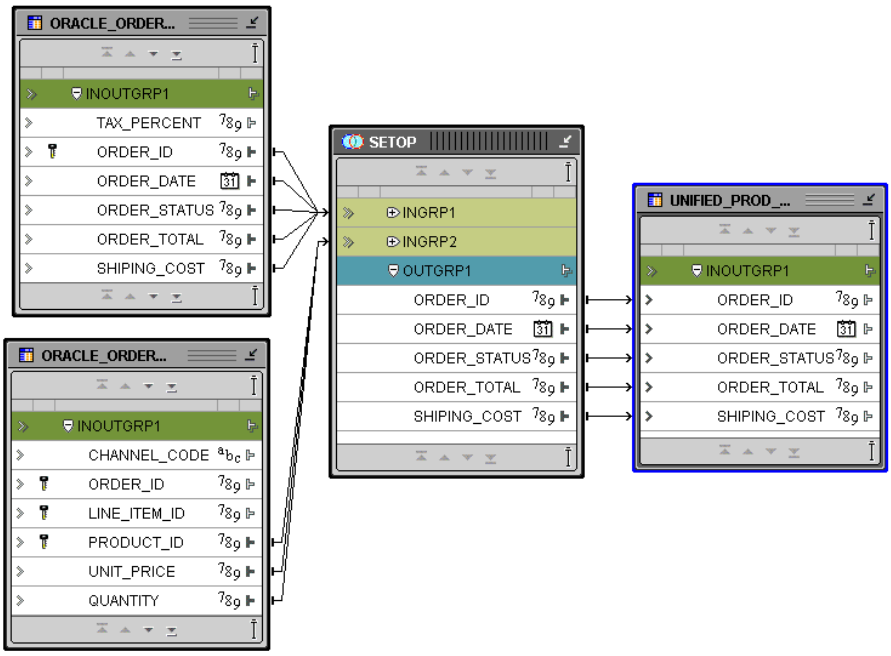


複数のセッションで順序演算子を使用できるので、番号が連続するとはかぎりません。順序演算子では、20 個の値の範囲というように、特定数の値がキャッシュされます。これは、セッションの終了時に失われます。

集合（UNION、UNION ALL、INTERSECT、MINUS）

集合演算子では、2 つのコンポーネントに対する問合せの結果が、1 つの結果に組み合されます。結合子とは異なり、集合演算子では、結果セットの結合に WHERE 句は必要ありません。セット・ベースの演算子では、データが 1 つの出力に追加されますが、列リストが結合されて 1 つの組み合さった列リストが形成されることはありません。結合子では、別々の行が 1 行に組み合さりますが、集合演算子では、図 2-16 のように、すべてのデータ行が共通の行に組み合さります。

図 2-16 セット・ベースの変換の適用



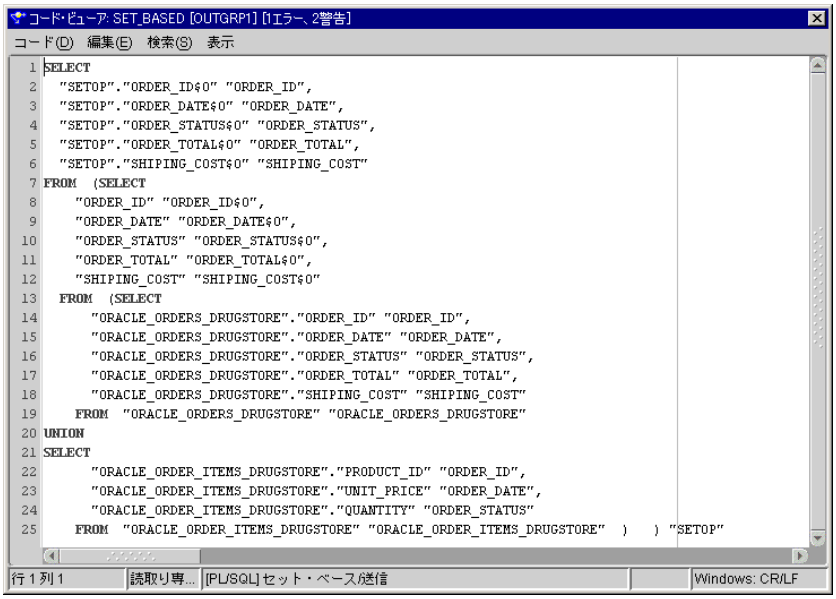
Warehouse Builder では、この演算子のモードとして UNION、UNION ALL、INTERSECT および MINUS がサポートされています。表 2-2 に、各モードにおける演算子の戻り値を示します。詳細は、『Oracle Database SQL リファレンス』を参照してください。

表 2-2 集合演算子の戻り値

演算子	戻り値
UNION	どちらかの問合せによって選択された、すべての行
UNION ALL	どちらかの問合せによって選択された、すべての重複を含むすべての行
INTERSECT	両方の問合せによって選択された、すべての個別行
MINUS	最初の問合せによって選択され、次の問合せによって選択されない、すべての個別行

UNION（または UNION ALL）演算子は、異種ソースの製品リストや顧客リストを組み合せるときなどに、最も一般的に使用されます。表（多くの場合はステージング表）のフォーマットが一致する場合、UNION では、1 つに統合された製品リストにレコードが組み合さります。生成されたレコードはウェアハウスにロードしたり、クレンジングしてからウェアハウスに保存できます。図 2-17 は、2 つの製品表で UNION を実行する例です。

図 2-17 2 つの製品表に対する UNION の実行



演算子のモードを変更すると（各演算子は 1 つのアクションのみ実行可能）、生成コードが変更され、UNION キーワードは選択したモードに置換されます。

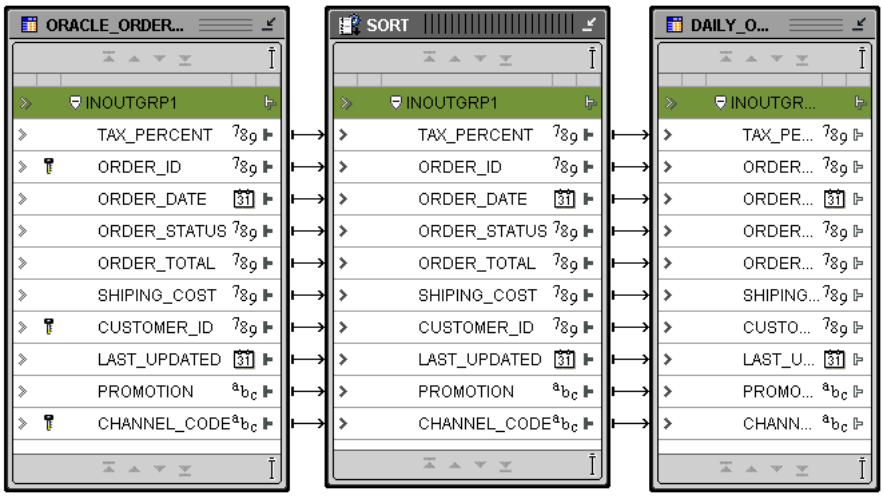
ソーター（ORDER BY）

データ・ウェアハウスのほとんどのデータは、一般的にバッチでロードされるので、ローディング・ルーチンで問題が発生することがあります。たとえば、注文のバッチでは、同じ注文番号が、異なるステータスを持つ注文行ごとに複数回現れることがあります。ある注文のステータスが、1 日のうちに、「CREATED」、「UPDATED」、「BOOKED」と移行したとします。

SQL の SELECT 文では、順序がデフォルトで保証されないので、ターゲット表での挿入と更新が、誤った順序で実行される可能性があります。ステータスが「UPDATED」の行が最後に処理されると、最新ステータスが「BOOKED」にもかかわらず、「UPDATED」がその日の最新値になってしまいます。

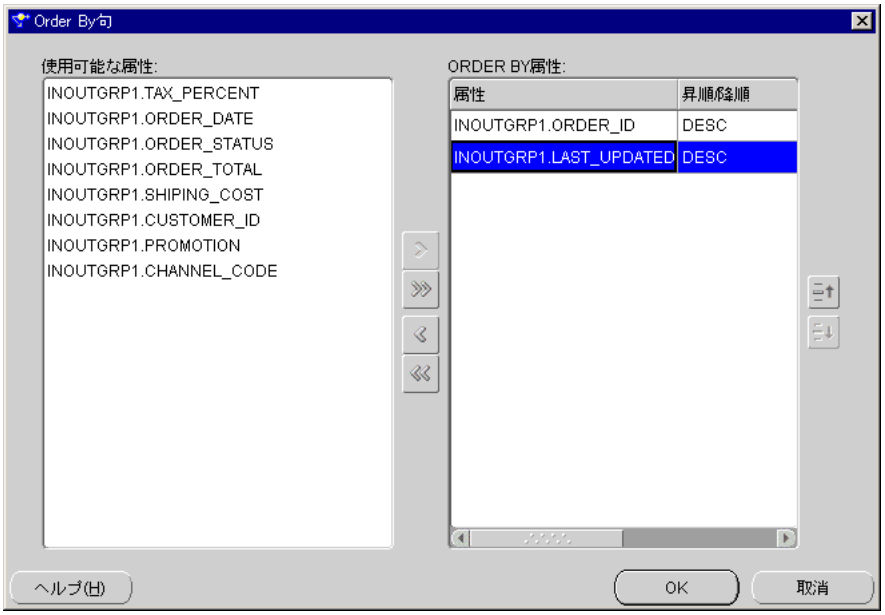
Warehouse Builder では、ソーター演算子を使用して順序付けした抽出問合せを作成することで、この問題を解決できます。

図 2-18 マッピングでのデータの順序付け



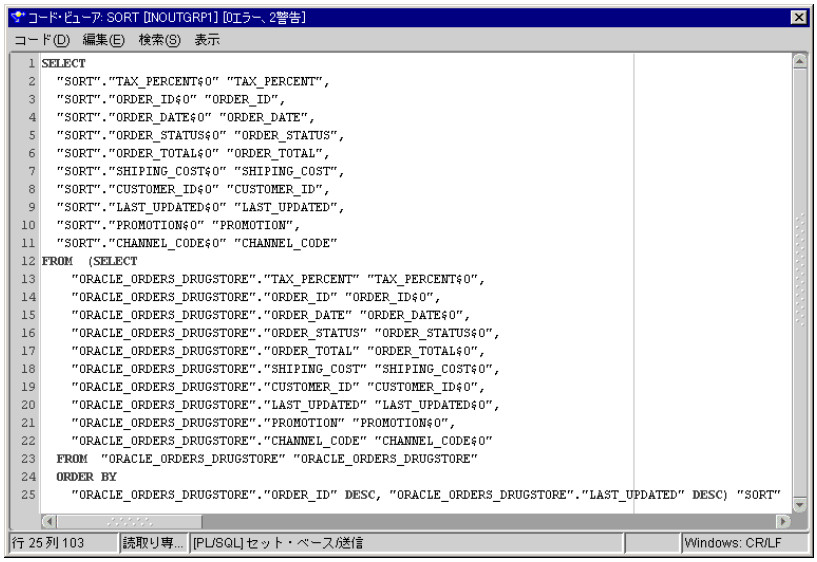
ソーターでは、SQL 文に ORDER BY 句が作成され、ORDER BY 句に指定された列を基準にソートされます。ORDER BY 句で最初にリストされる列は、後にリストされる列より優先されます。図 2-18 のように、最初の順序付けは注文番号を基準に実行され、注文番号の同一グループ内では、最新更新日付を基準に順序付けが実行されます。ターゲットでは最後の変更が最新更新日付となるため、注文の正しい最新ステータスがターゲットに反映されます。

図 2-19 ソートとソート順序の設定



GROUP BY 句内での順序付けは、ASCENDING（昇順）または DESCENDING（降順）で実行できます。デフォルトは、図 2-19 のように DESCENDING（降順）です。

図 2-20 生成された SQL の ORDER BY 句



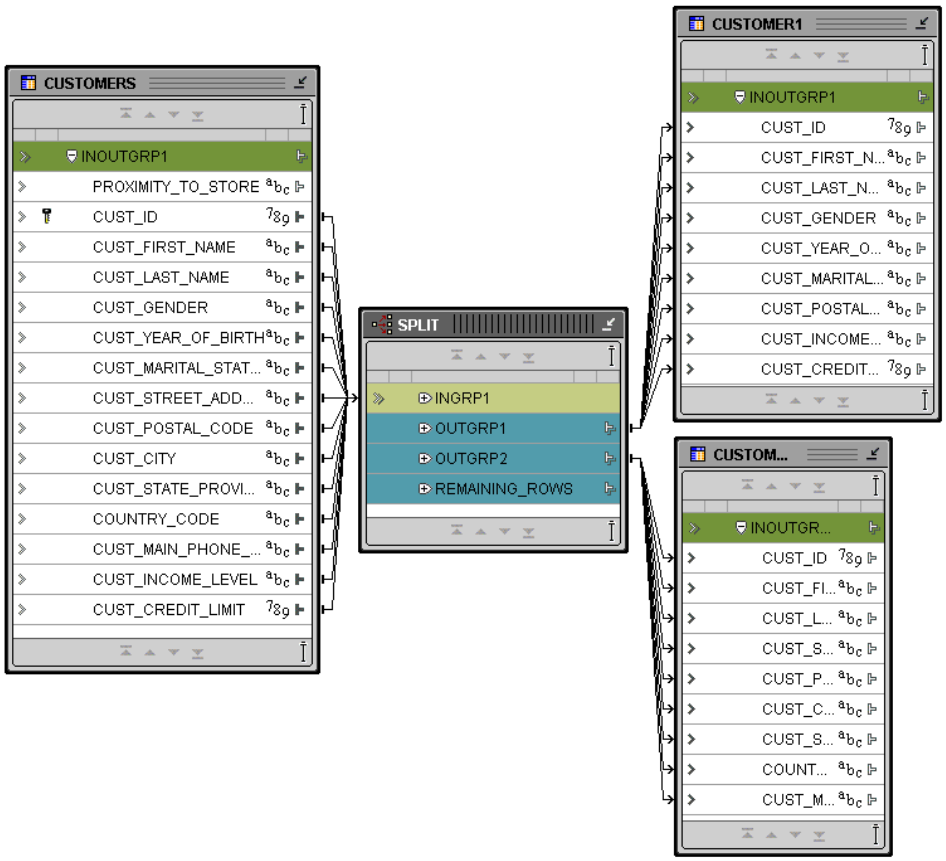
ORDER BY 句での属性の順序を変更すると、生成される SQL 内での順序が変更され、マッピングの動作も変更されます。図 2-20 に、生成された ORDER BY 句を示します。

スプリッタ（WHERE による複数表分割）

ウェアハウス環境では、データ移動条件に基づいて、データを様々なターゲットに移動する必要があることがあります。Warehouse Builder では、複数のフィルタでデータを移動するかわりに、スプリッタを使用できます。この演算子は、入力データを取り、指定した分割条件に基づいて、データの複数のフローを出力します。

たとえば、CUSTOMER 表を住所と純粋な顧客情報に分割する場合は、図 2-21 のように、1 つの行を 2 つの表に挿入する必要があります。

図 2-21 条件のない分割の実行



この例では、分割条件がスプリッタに追加されていません。OUTGRP1 と OUTGRP2 には条件がありませんが、OUTGRP1 の列セットが 1 つのターゲットにマッピングされ、OUTGRP2 の列セットが別のターゲットにマッピングされます。1 人の顧客に複数の住所がある場合などに顧客数を減らすには、上位フローにデュプリケータ解除を追加し、1 つの住所につき 1 人の顧客を取得します。cust_id を両方のターゲットにマッピングすると、関係が常に保たれます。

現在のところ、生成されるコードは、データの 2 つの個別ストリームです。それぞれのターゲットは、データ受信者として扱われます。データを挿入する場合は、図 2-22 と図 2-23 のように、2 つの挿入文が 1 つのパッケージに生成されます。

図 2-22 顧客表の挿入

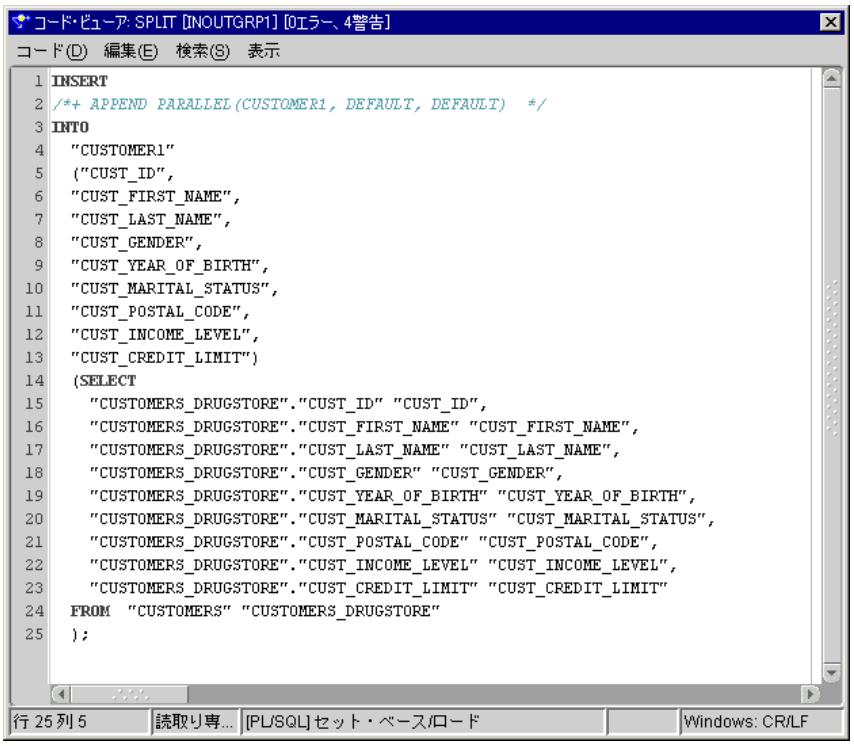
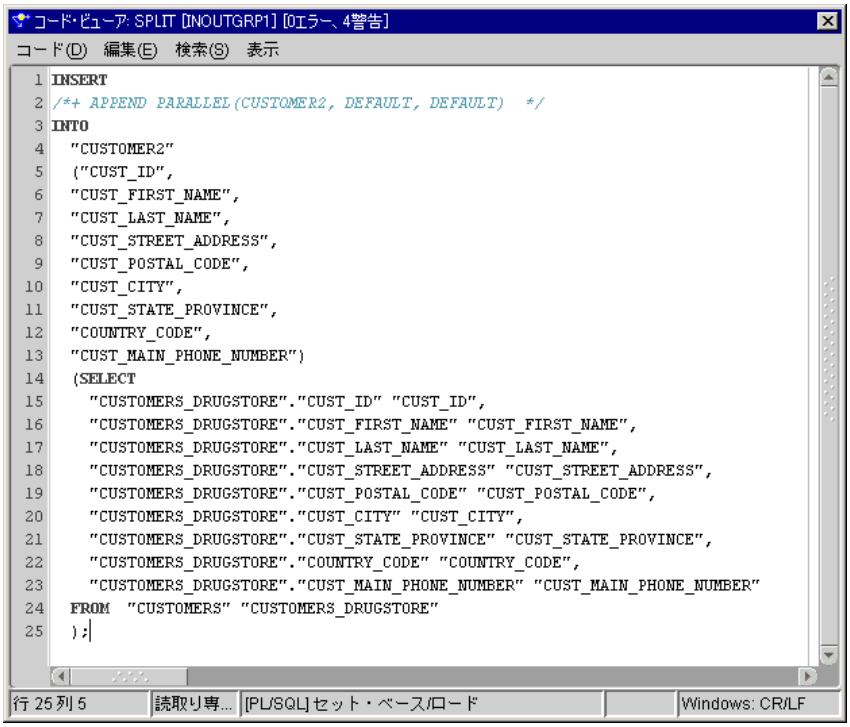


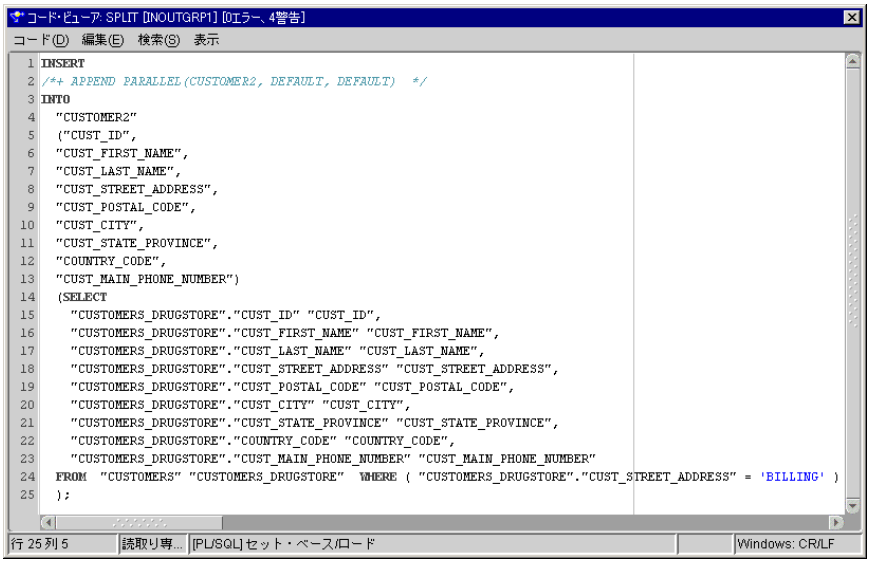
図 2-23 住所データの挿入



次の例では、住所を分割して、住所表に請求先住所のみを保持します。OUTGRP2 に条件を追加すると、請求先住所のみを選択できます。この場合は、図 2-24 のように、WHERE 句がコードに追加されます。

出力グループでどの句も満たされない場合、そのデータは、通常の出力グループに保持されないすべてのデータを含むデフォルト・グループに追加されます。

図 2-24 請求先住所のみの挿入



条件フィルタにスプリッタを使用することもできます。たとえば、これを使用すると、メイン・ブランチのエラー・データを別のエラー表に分割できます。

テーブル・ファンクション

通常のファンクションは同時に 1 つの行でしか動作しませんが、テーブル・ファンクションでは、同じ複雑な PL/SQL ロジックを行のセットに適用し、パフォーマンスを上げることができます。

Warehouse Builder では、テーブル・ファンクション演算子をマッピングに追加し、行のセットを入力できます。この行セットは、テーブル・ファンクション内の PL/SQL ロジックで変換されてから、次の演算子に出力されます。

図 2-25 マッピングでのテーブル・ファンクション

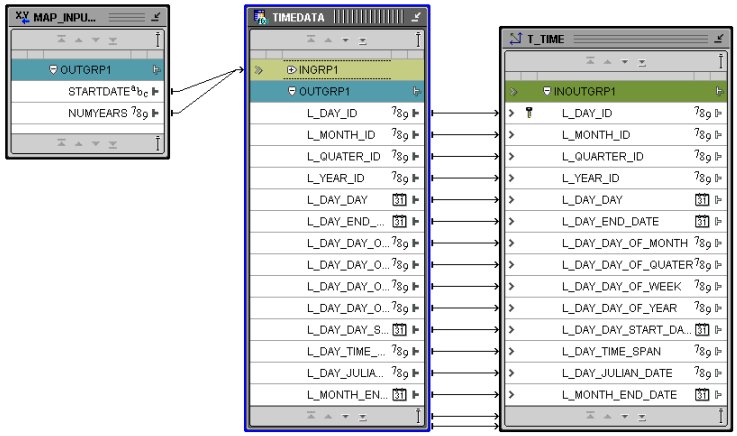


図 2-25 では、時間ディメンションがテーブル・ファンクションからロードされます。このテーブル・ファンクションは、SELECT 文の FROM 句に追加されます。このテーブル・ファンクションは行セット・プロバイダの役割を果たし、複雑なカレンダー・データ生成とロードを 1 つの insert as select 文で実行できるようにします。

アンピボット演算子

アンピボット演算子では、複数の入力行が 1 つの出力行に変換されます。ソースから 1 回抽出し、次にソース・データの属性によってグループ化されるソース行のセットから 1 行を生成することもできます。ピボット演算子と同じように、アンピボット演算子は、マッピングの任意の場所に配置できます。

例 : 売上データのアンピボット

表 2-3 は、リレーショナル表 SALES のデータの代表的なサンプルです。クロス集計フォーマットでは、「MONTH」列に、1 年の各月を表す 12 の文字列値のいずれかが挿入されます。すべての売上数値は、1 つの列「MONTHLY_SALES」に挿入されます。

表 2-3 クロス集計フォーマットのデータ

REP	MONTH	MONTHLY_SALES	REGION
0675	Jan	10.5	4
0676	Jan	9.5	3
0679	Jan	8.7	3
0675	Feb	11.4	4
0676	Feb	10.5	3
0679	Feb	7.4	3
0675	Mar	9.5	4
0676	Mar	10.3	3
0679	Mar	7.5	3
0675	Apr	8.7	4
0676	Apr	7.6	3
0679	Apr	7.8	3

図 2-26 は、Warehouse Builder で表をアンピボットした後のリレーショナル表 SALES のデータです。「MONTH」列にあった Jan、Feb、Mar などのデータは、12 の異なる属性（M1、M2、M3...）に対応しています。「MONTHLY_SALES」にあった売上数値は、月ごとの 12 の属性に分散されました。

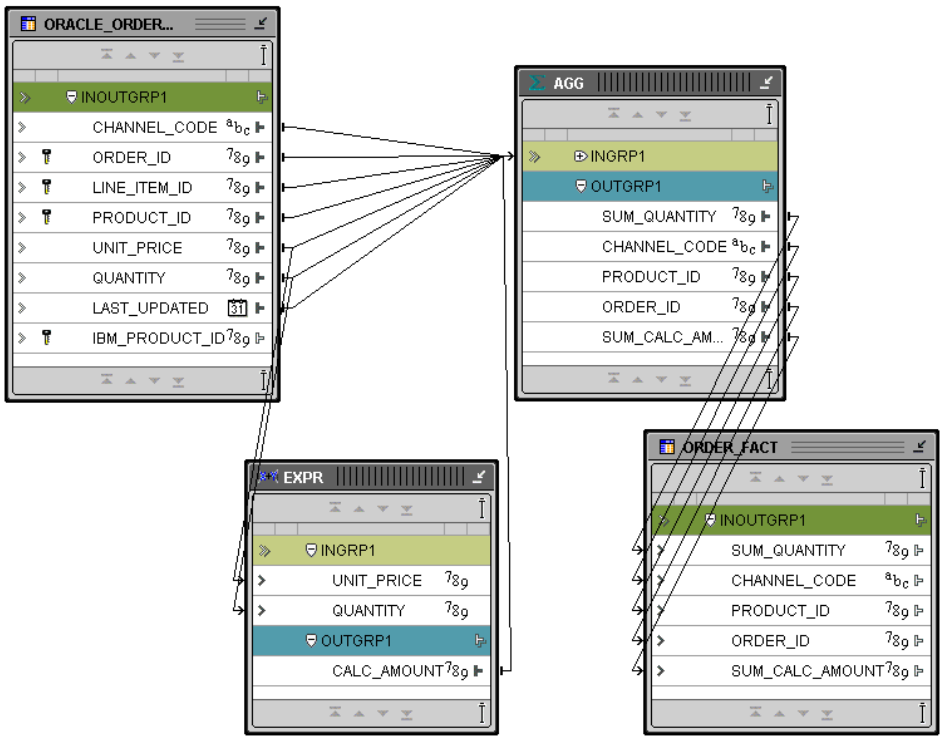
図 2-26 クロス集計フォーマットからアンピボットされたデータ

ID	Reg	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
0675	4	10.5	11.4	9.5	8.7	7.4	7.5	7.8	9.7				
0676	3	9.5	10.5	10.3	7.6	8.0	7.8	8.7	8.9				
0679	3	8.7	7.4	7.5	7.8	9.7	10.3	7.6	8.0				
0683	2	9.5	10.5	10.3	9.5	8.7	7.4	7.8	8.7				
0684	1	11.4	9.5	8.7	7.4	7.5	10.3	9.5	8.7				
0687	1	9.5	8.7	7.4	7.8	8.7	7.4	7.5	7.8				
0690	1	8.7	7.4	7.8	8.7	11.4	9.5	8.7	7.4				

集計（GROUP BY、HAVING）

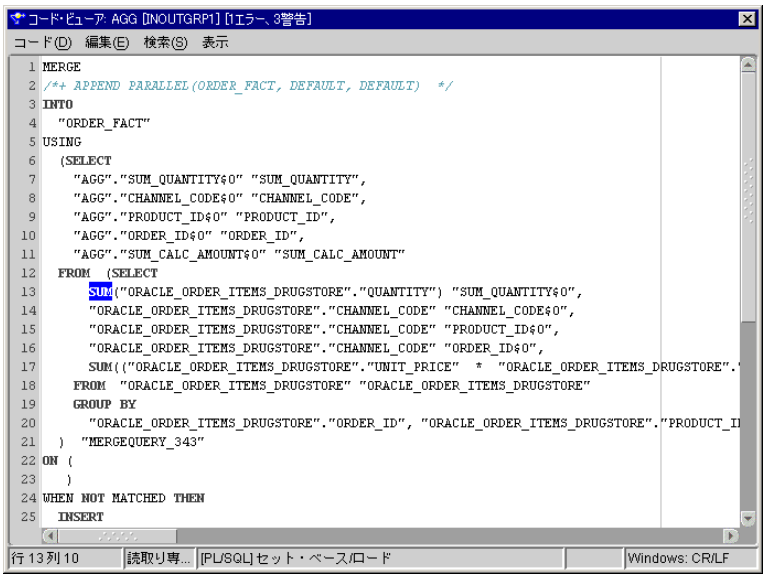
ファクト・データの集計は、一般的な変換操作です。Warehouse Builder では、マッピングに 1 つの集計演算子を追加して、複数の集計を実行できます。Warehouse Builder では、個別のエディタが提供され、複雑な集計を作成できるようになっています。集計演算子では、属性ごとに異なる集計関数をコールできますが、各集計演算子でサポートされるのは、1 つの GROUP BY 句と 1 つの HAVING 句のみです。たとえば、図 2-27 のように、製品および注文ディメンション間で注文を集計する場合があります。

図 2-27 注文情報の集計



この例のターゲット表で挿入と更新が許可されている場合は（更新は、ディメンション・キー値または集計ポイントで照合）、図 2-28 に示すような問合せが、Warehouse Builder により生成されます。

図 2-28 集計データのマージ



SQL 文は、集計演算子のプロパティを使用して作成できます。それぞれの属性には独自の集計タイプが保持され、HAVING 句と GROUP BY 句は、図 2-29、図 2-30 および図 2-31 のように、演算子上で修正されます。

図 2-29 GROUP BY 句

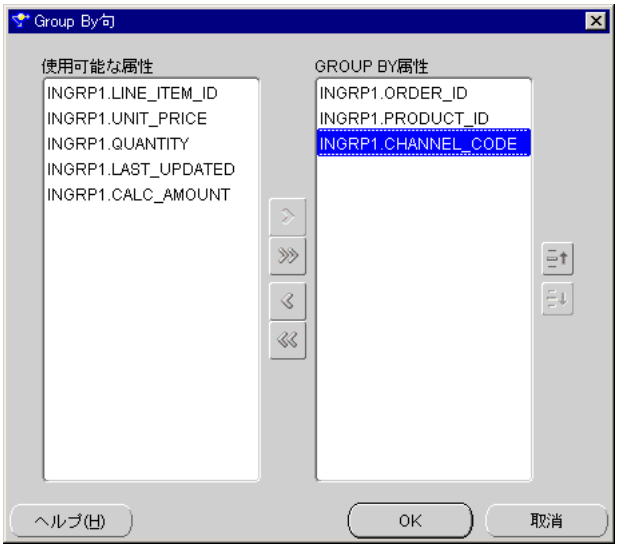


図 2-30 属性ごとの集計関数

集計演算子では次の関数を使用できます。

AVG

構文

```
avg::=AVG(expr)
```

目的

AVG は、演算子に指定されている GROUP BY 句を使用して expr の平均値を返します。つまり、Warehouse Builder では、演算子に渡されるデータの平均値が出力グループ属性として返されます。

例

次の例では、OE.EMPLOYEES 表のすべての従業員の平均給与が計算されます。

```
SELECT AVG(salary) "Average" FROM employees;

Average
--------
6425
```

COUNT

構文

```
count::=COUNT(expr)
```

目的

COUNT は、演算子に指定されている GROUP BY 句を使用して問合せの行数を返します。expr を指定すると、COUNT は expr が NULL でない行の数を返します。すべての行数または expr の個別値のみを計算できます。アスタリスク（*）を指定すると、この関数は、重複と NULL を含むすべての行数を返します。COUNT では、NULL が値として返されることはありません。

例

次の例では、集計関数として COUNT を使用しています。

```
SELECT COUNT(commission_pct) "Count" FROM employees;

Count
----------
35
```

MAX

構文

```
max::=MAX(attribute)
```

目的

MAX は、演算子に指定されている GROUP BY 句を使用して attribute の最大値を返します。つまり、集計演算子は、attribute に渡されるデータの最小値を出力グループ属性として返します。

例

次の例では、HR.EMPLOYEES 表の最高給与額が返されます。

```
SELECT MAX(salary) "Maximum" FROM employees;

Maximum
----------
24000
```

MIN

構文

```
min::= MIN(attribute)
```

目的

MIN は、演算子に指定されている GROUP BY 句を使用して attribute の最小値を返します。つまり、集計演算子は、attribute に渡されるデータの最小値を出力グループ属性として返します。

例

次の文では、HR.EMPLOYEES 表にある最も早い雇用日付が返されます。

```
SELECT MIN(hire_date) "Earliest" FROM employees;

Earliest
---------
17-JUN-87
```

NONE

構文

```
none::=Group By (attribute)
```

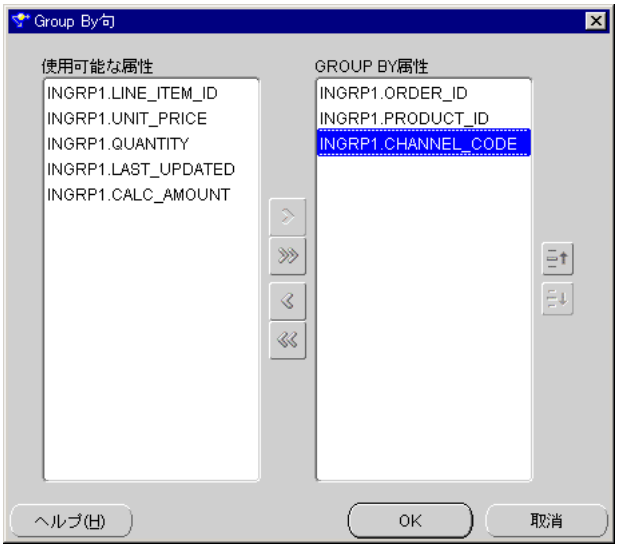
目的

GROUP BY 句に属性を追加した場合、この属性の集計に使用されるアクションを識別するには、NONE を使用します。attribute の集計演算子に NONE を指定すると、Group By 句にその属性が自動的に追加されます（その反対も可能）。その他の関数のように、NONE の使用により、SQL 文で集計が行われることはありません。

例

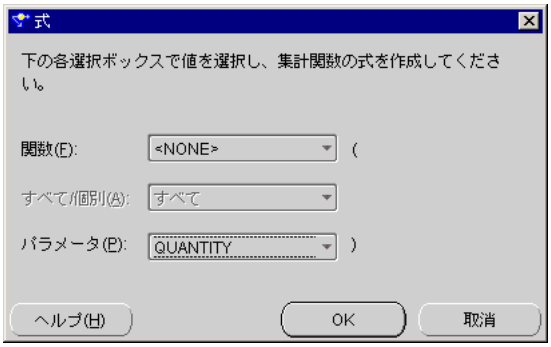
図 2-31 では、右側に GROUP BY 属性が表示されています。属性を左側から右側に移動すると、集計アクションが自動的に NONE に切り替わります。

図 2-31「Group By 句」ダイアログ



逆に（集計関数ダイアログと同様に）、NONE を選択すると、GROUP BY 句に属性が移動し、右側に表示されます。

図 2-32 属性に集計を選択しない設定



STDDEV

構文

```
stddev::= STDDEV(attribute)
```

目的

STDDEV では、attribute に指定される数値セットの標本標準偏差が返されます。Warehouse Builder でのこの属性は、集計演算子に入力される行で構成されるのが一般的です。

STDDEV と STDDEV_SAMP は異なります。入力データが 1 行の場合、STDDEV ではゼロが返されますが、STDDEV_SAMP では NULL が返されます。Oracle では、VARIANCE 集計関数に定義されている分散の平方根として標準偏差が計算されます。

例

次の例では、サンプルの HR.EMPLOYEES 表の給与値の標準偏差が返されます。

```
SELECT STDDEV(salary) "Deviation"
FROM employees;

Deviation
----------
3909.36575
```

STDDEV_POP

構文

```
stddev_pop::=STDDEV_POP(sttribute)
```

目的

STDDEV_POP は、母集団の標準偏差を計算し、集計演算子の出力属性に母集団分散の平方根を返します。

この attribute は数値表現で、この関数は NUMBER 型の値を返します。

この関数は、VAR_POP 関数の平方根と同じです。VAR_POP が NULL を返す場合、この関数も NULL を返します。

例

次の例では、サンプル表 SH.SALES にある売上量の母集団標準偏差と標本標準偏差が返されます。

```
SELECT STDDEV_POP(amount_sold) "Pop",
STDDEV_SAMP(amount_sold) "Samp"
FROM sales;

Pop         Samp
----------  ----------
944.290101  944.290566
```

STDDEV_SAMP

構文

```
stddev_samp::=STDDEV_SAMP(attribute)
```

目的

STDDEV_SAMP は、累積の標本標準偏差を計算し、集計演算子の出力属性に標本分散の平方根を返します。

この attribute は数値表現で、この関数は NUMBER 型の値を返します。この関数は、VAR_SAMP 関数の平方根と同じです。VAR_SAMP が NULL を返す場合、この関数も NULL を返します。

例
次の例では、サンプル表 SH.SALES にある売上量の母集団標準偏差と標本標準偏差が返されます。

```
SELECT STDDEV_POP(amount_sold) "Pop",
STDDEV_SAMP(amount_sold) "Samp"
FROM sales;

Pop         Samp
----------  ----------
944.290101  944.290566
```

SUM

構文
```
sum::=SUM(attribute)
```

目的
SUM は、attribute の値の合計を返します。Warehouse Builder での attribute は、一般的に集計演算子に入力される行セットです。これには、前の変換の式またはソース・システムの式が含まれます。

例
次の例では、サンプル HR.EMPLOYEES 表のすべての給与の合計が計算されます。

```
SELECT SUM(salary) "Total"
FROM employees;

Total
----------
691400
```

VAR_POP

構文
```
var_pop::= VAR_POP(attribute)
```

目的
VAR_POP は、セットの NULL を破棄した後で、集計演算子の出力属性に数値セットの母集団分散を返します。この attribute は数値表現で、この関数は NUMBER 型の値を返します。この関数が空セットに適用された場合は、NULL が返されます。

この関数は、次の計算を行います。

```
(SUM(attribute² ) - SUM(attribute)² / COUNT(attribute)) / COUNT(attribute)
```

例
次の例では、HR.EMPLOYEES 表の給与の母集団分散が返されます。

```
SELECT VAR_POP(salary) FROM employees;

VAR_POP(SALARY)
---------------
15140307.5
```

VAR_SAMP

構文

```
var_samp::= VAR_SAMP(attribute)
```

目的

VAR_SAMP は、セットの NULL を破棄した後で、集計演算子の出力属性に数値セットの標本分散を返します。式は数値表現で、この関数は NUMBER 型の値を返します。この関数が空セットに適用された場合は、NULL が返されます。

この関数は、次の計算を行います。

```
(SUM(attribute² ) - SUM(attribute)² / COUNT(attribute)) / (COUNT(attribute) - 1)
```

この関数は VARIANCE に似ていますが、入力セットが 1 要素の場合は、VARIANCE では 0 が返され、VAR_SAMP では NULL が返されます。

例

次の例では、サンプル HR.EMPLOYEES 表の給与の標本分散が返されます。

```
SELECT VAR_SAMP(salary) FROM employees;

VAR_SAMP(SALARY)
----------------
15283140.5
```

VARIANCE

構文

```
variance::=VARIANCE(attribute)
```

目的

VARIANCE は、attribute の分散を返し、集計演算子の出力属性にその結果を出力します。Warehouse Builder では、attribute の分散は次のように計算されます。

- attribute の行数が 1 の場合は 0
- attribute の行数が 1 より大きい場合は VAR_SAMP

例

次の例では、サンプル HR.EMPLOYEES 表の全給与の分散が計算されます。

```
SELECT VARIANCE(salary) "Variance"
FROM employees;

Variance
----------
15283140.5
```

定数

多くの変換には、定数値が必要になります。Warehouse Builder には、直接関数としていくつかの定数があります。Expression Builder を使用すると、Warehouse Builder で定数を作成できます。次の項では、特殊関数として使用される定数について説明します。

SYSDATE

構文

```
sysdate::=SYSDATE
```

目的

SYSDATE は、現在の日時を返し、引数を必要としません。分散型 SQL 文では、この関数により、ローカル・データベースの日時が返されます。CHECK 制約の条件にこの関数を使用することはできません。

例

次の例では、現在の日時が返されます。

```
SELECT TO_CHAR (SYSDATE, 'MM-DD-YYYY HH24:MI:SS')"NOW"
FROM DUAL;

NOW
-------------------
04-13-2001 09:45:51
```

SYSTIMESTAMP

構文

```
systimestamp::=SYSTIMESTAMP
```

目的

SYSTIMESTAMP 関数は、小数点以下の秒とデータベースのタイム・ゾーンを含むシステム日付を返します。戻り値の型は TIMESTAMP WITH TIME ZONE です。

例

次の例ではシステムの日付が返されます。

```
SELECT SYSTIMESTAMP FROM DUAL;

SYSTIMESTAMP
---------------------------------------------------------------------
28-MAR-00 12.38.55.538741 PM -08:00
```

データ・クレンジング演算子

Warehouse Builder マッピング・エディタには、データ・クレンジング変換を実行する演算子が含まれています。この項では、このような演算子について説明します。

Name and Address

Warehouse Builder には、Name and Address のクレンジングを可能にする演算子が含まれています（9.0.2.56.0 バージョン）。Name and Address 演算子では、名前データとアドレスデータの解析、標準化、郵便照合、ジオコーディングがサポートされます。Name and Address 解析では、分離していない入力が、個別の名前コンポーネントまたはアドレス・コンポーネントに分割されます。たとえば、次のような入力アドレスがあるとします。

```
Mr. Joe A. Smith Sr.
8500 Normandale Lake Blvd Suite 710
Bloomington MN 55438
```

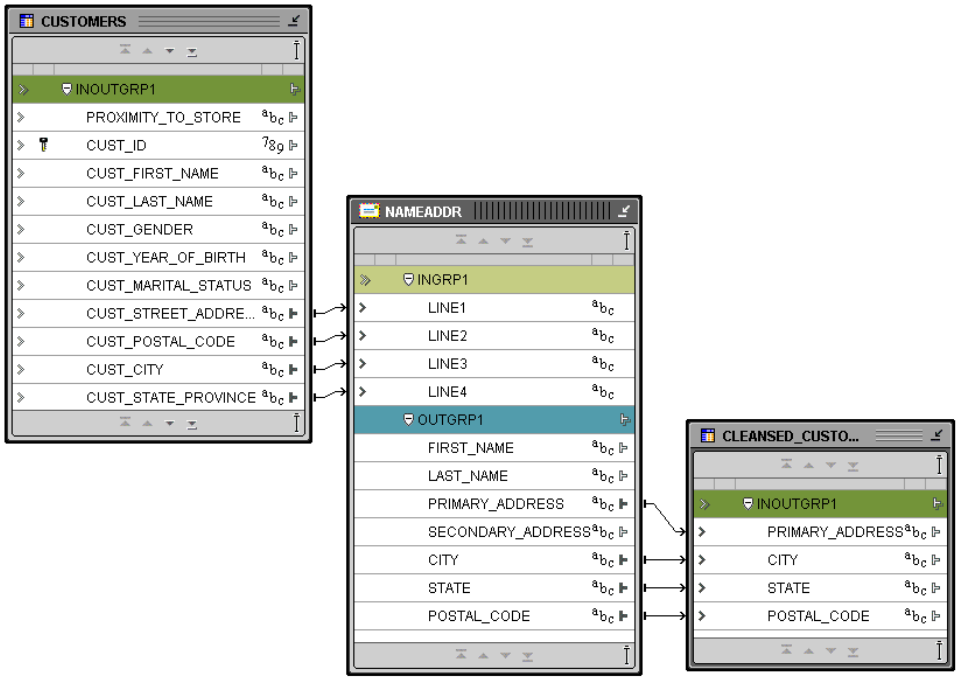
これは、次のようなアドレス・コンポーネントの短縮形リストに解析されます。

```
Pre name: MR
First name: JOE
First name standardized: JOSEPH
Post name: SR
Street name: NORMANDALE LAKE
Primary address: NORMANDALE LAKE BLVD
Secondary address: STE 710
Last Line: BLOOMINGTON MN  55437-3813
Latitude: 44.854876
```

Name and Address の標準化では、郵便サービスで容認されるか、レコード照合に適している標準バージョンにコンポーネントが修正されます。上の例では、Suite が STE に、Joe が JOSEPH に標準化されています。

郵便照合では、入力アドレスが郵便データベース・エントリと照合され、アドレスの確認や訂正が行われます。上の例では、郵便番号が 55437-3813 に訂正されています。

図 2-33 Name and Address 演算子



ジオコーディング（米国のみで使用可能）では、人口調査データと場所データが集計されます。Warehouse Builder では、緯度と経度を現在使用できます。後のバージョンでは、小調査地区、大都市統計地域、FIPS コードなどの人口調査データが使用できるようになります。

Match-Merge 演算子

Match-Merge 演算子は、データ品質演算子であり、データを照合してからマージする際に使用できます。

レコードを照合する場合は、表のどのレコードが同じデータを参照するかをビジネス・ルールから判断します。レコードをマージする場合は、照合したレコードのデータを 1 つのレコードに統合します。

この項では、Match-Merge 演算子をマッピングで使用する方法について、例をあげて説明します。Match-Merge 演算子と Name and Address 演算子を使用すると、Name and Address データで一意のハウスホールドを識別するプロセスがサポートされます。

例 : 顧客データの照合とマージ

Match-Merge 演算子を活用し、顧客メーリング・リストを管理する方法について考察します。10,000 行を含む顧客データの表で、同一人物を参照するレコードを検索するには、照合を使用します。たとえば、姓と名が同一であるレコードを選別する照合ルールを定義できます。照合により、5 行が同一人物を参照していることが検出されたとします。このようなレコードは、1 つの新しいレコードにマージできます。たとえば、一致した 5 つのレコードのうち、住所が最も長いレコードから値を取得するマージ・ルールを作成できます。新しくマージした表には、顧客 1 人につき 1 レコードが含まれます。

表 2-4 は、Match-Merge 演算子を使用する前の同一人物を参照するレコードです。

表 2-4 サンプル・レコード

Row	FirstName	LastName	SSN	Address	Unit	Zip
1	Jane	Doe	NULL	123 Main Street	NULL	22222
2	Jane	Doe	111111111	NULL	NULL	22222
3	J.	Doe	NULL	123 Main Street	Apt 4	22222
4	NULL	Smith	111111111	123 Main Street	Apt 4	22222
5	Jane	Smith-Doe	111111111	NULL	NULL	22222

表 2-5 は、Match-Merge 演算子の使用後に 1 つになった Jane Doe のレコードです。新しいレコードでは、サンプルの様々な行からデータが取得されています。

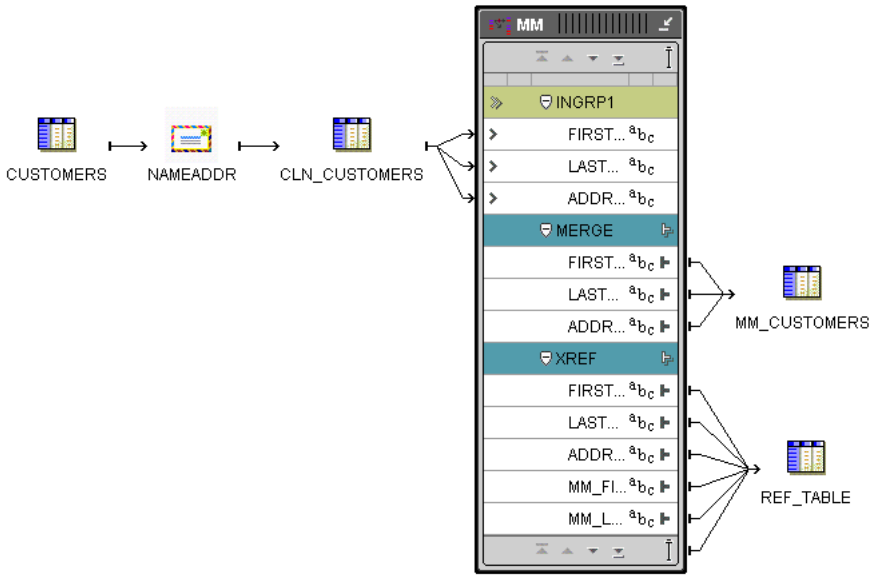
表 2-5 Jane Doe のマージ済レコード

First Name	Last Name	SSN	Address	Unit	Zip
Jane	Doe	111111111	123 Main Street	Apt 4	22222

Match-Merge 演算子によるマッピングの設計

図 2-34 に、Match-Merge 演算子を使用して設計できるマッピングを示します。Match-Merge 演算子の前に、Name and Address 演算子 NAMEADDR とステージング表 CLN_CUSTOMERS があることに注意してください。マッピングは、Name and Address 演算子を使用しても使用しなくても設計できます。データのクレンジングと標準化を行ってから、時間がかかる照合操作とマージ操作を実行する場合は、Match-Merge 演算子の前に Name and Address 演算子を置いてください。

図 2-34 マッピングでの Match-Merge 演算子



Name and Address 演算子を組み込むかどうかを決める場合は、マッピングの設計時に、次の事項に注意してください。

- **PL/SQL 出力 :** Match-Merge 演算子では 2 つの出力を生成できますが、どちらも PL/SQL 出力のみです。MERGE グループには、マージ済データが含まれます。XREF グループはオプションのグループであり、これを設計すると、マージ・プロセスをドキュメント化できます。
- **行ベースの操作モード :** Match-Merge 演算子は、レコードを照合する場合、各行をソースの後続行と比較し、行ベースのコードのみを生成します。このため、このようなマッピングは、行ベース・モードのみで実行できます。
- **Match-Merge 演算子の前の SQL ベース演算子 :** Match-Merge 演算子では、PL/SQL 出力のみが生成されます。SQL コードのみを生成する演算子を組み込む場合は、Match-Merge 演算子に先行するようにマッピングを設計する必要があります。たとえば、結合、キー参照、集合などの演算子は、Match-Merge 演算子の前に置く必要があります。セット・ベースのコードを生成する演算子を Match-Merge 演算子の後に置くマッピング設計は無効で、Warehouse Builder では、このようなマッピングのコードは生成されません。
- **SQL 入力 :** 例外が 1 つありますが、Match-Merge 演算子では SQL 入力が必要です。Name and Address 演算子など、PL/SQL 出力のみを生成する演算子を Match-Merge 演算子の前に置く場合は、まずデータをステージング表にロードする必要があります。
- **Match-Merge 演算子のデータ詳細化 :** 大きなデータ詳細化を可能にするには、1 つの Match-Merge 演算子からさらに別の Match-Merge 演算子に、XREF 出力をマッピングします。このシナリオが、Match-Merge 演算子の SQL 入力ルールの例外です。別の設計要素により、2 番目の Match-Merge 演算子は、PL/SQL を受け取ります。

Match-Merge 演算子の詳細は、『Oracle Warehouse Builder ユーザーズ・ガイド』を参照してください。

3

SQL 変換

この章では、次のトピックについて説明します。

関連情報は、次を参照してください。

- 『Oracle Database SQL リファレンス』
- 『Oracle Warehouse Builder ユーザーズ・ガイド』

はじめに

次の項では、変換ライブラリについて説明し、Warehouse Builder におけるカスタム変換の使用方法について概説します。

変換について

Warehouse Builder では、次の変換タイプがサポートされます。

- **ユーザー変換パッケージ :** このカテゴリには、ユーザーが定義するパッケージ・ファンクションとパッケージ・プロシージャが含まれます。
- **定義済変換 :** これらのカテゴリは事前定義済にあり、組込みおよびシードされたファンクションとプロシージャで構成されます。
- **ファンクション :** ファンクションのカテゴリは、すべてのウェアハウス・モジュールに自動的に作成されます。このカテゴリには、変換として使用されるスタンドアロン・ファンクションがすべて含まれます。これらのファンクションは、ユーザーが定義するか、またはデータベースからインポートします。ファンクション変換は、任意の数（0 を含む）の入力パラメータを受け取って 1 つの結果値を生成します。
- **プロシージャ :** プロシージャのカテゴリは、すべてのウェアハウス・モジュールに自動的に作成されます。このカテゴリには、変換として使用されるスタンドアロン・プロシージャがすべて含まれます。これらのプロシージャは、ユーザーが定義するか、またはデータベースからインポートします。プロシージャ変換は、任意の数（0 を含む）の入力パラメータを受け取って任意の数（0 を含む）の出力パラメータを生成します。
- **インポートされたパッケージ :** このカテゴリは、PL/SQL パッケージをインポートすると作成されます。パッケージの本体は修正できますが、ファンクションまたはプロシージャの署名となるパッケージ・ヘッダーは修正できません。パッケージは、変換ライブラリのプロパティ・シートに表示できます。

パブリック変換について

Warehouse Builder では、プロジェクトごとにパブリック変換が作成されます。ライブラリには、標準の事前定義済変換と、Design Repository 内に定義されたカスタム変換が含まれます。

パブリック変換には次のタイプがあります。

- **カスタム :** Design Repository 内に定義されたファンクションおよびプロシージャとして分類される、再利用可能な変換の集まり。
- **事前定義済 :** カスタムにプロシージャを定義する際に使用できる、事前定義のファンクションの集まり。

カスタム変換を作成する際、プロジェクト内で共有するには、カスタムに追加します。1 つのモジュールに専用の変換の場合は、そのモジュール内のパブリック変換に追加します。

カスタム

カスタムには、Design Repository 間で共有する変換を格納します。デフォルトのカテゴリは次のとおりです。

- **ファンクション :** このカテゴリにはスタンドアロン・ファンクションが格納されます。
- **プロシージャ :** このカテゴリにはスタンドアロン・プロシージャが格納されます。

事前定義済

事前定義済には、一連の標準変換が次のカテゴリに分類されて格納されます。

- Administration
- Character
- Conversion

- Date
- Numeric
- Other
- XML

変換へのアクセス

変換には、Expression Builder、「マッピング変換の追加」ダイアログまたは新規変換ウィザードからアクセスします。また、Warehouse Builder コンソールのナビゲーション・ツリーからもアクセスできます。独自の変換を作成し、ニーズに従って変換を編成することもできます。

PL/SQL パッケージのインポート

インポート・ウィザードを使用して、PL/SQL ファンクション、プロシージャおよびパッケージを Warehouse Builder プロジェクトにインポートできます。

インポートした PL/SQL を使用する際には、次の点に注意してください。

- インポートした PL/SQL ファンクションおよびプロシージャは、編集、保存および配布できます。
- インポートした PL/SQL パッケージは編集できません。
- ラップされた PL/SQL オブジェクトは読取りできません。
- インポートしたパッケージは、プロパティ・シートで表示および変更できます。
- インポートしたパッケージの本体は編集できますが、仕様は編集できません。

変換を編集するには、変換プロパティ・シートを使用します。プロパティの編集は必ず一貫して行ってください。たとえば、パラメータの名前を変更したら、実装コード内の名前も変更する必要があります。変換のプロパティは、マッピング・エディタから「演算子のプロパティ」シートを使用して表示できます。この設定は読取り専用です。

Administration

Administration 変換つまりファンクションは、ETL プロセスで定期的に実行されるアクションです。Administration 変換は主に DBA 関連の領域で実行され、パフォーマンスの改善に焦点があてられます。たとえば、多くの場合、表のロード時に無効にした制約は、ロード後に再び有効にする必要があります。Warehouse Builder では、Administration 変換により、これらの目的に使用する事前構築済ファンクションが提供されます。

Warehouse Builder の管理ファンクションは、すべてカスタム・ファンクションです。次に、これらをアルファベット順に説明します。

WB_ABORT

構文

```
WB_ABORT(p_code, p_message)
```

p_code は強制終了コードで、-20000 から -29999 までの間にする必要があります。*p_message* は、指定する強制終了メッセージです。

目的

WB_ABORT では、Warehouse Builder コンポーネントのアプリケーションを強制終了できます。WB_ABORT はマッピング後プロセスから実行できます。また、マッピング内の変換として実行できます。

例

この管理ファンクションを使用して、アプリケーションを強制終了します。マッピングでエラーが発生した場合は、マッピング後プロセスでこのファンクションを使用して、配布を強制終了できます。

WB_ANALYZE_SCHEMA

構文

```
WB_ANALYZE_SCHEMA
```

このファンクションでは、パラメータは必要ありません。

目的

ウェアハウスにデータをロードした後は、統計をリフレッシュして、コストベース・オプティマイザがウェアハウスで推奨するパフォーマンス最適化戦略を確実にする必要があります。WB_ANALYZE_SCHEMA は、スキーマを分析する DBMS_DDL.ANALYZE_OBJECT をコールし、この統計を提供します。スキーマ全体を分析するので、表の数と表内の行数によっては、時間がかかることがあります。

例

この管理パッケージを使用し、ロードしたスキーマで分析コマンドを自動的に実行できます。これは、依存関係図の最終マッピングのマッピング後プロセスを使用して実行できます。データベース・スキーマにプロシージャを配布し、クライアント・ツールで作成された SQL 文を使用して OEM からこのプロシージャを起動することもできます。

WB_ANALYZE_TABLE

構文

```
WB_ANALYZE_TABLE(p_name)
```

p_name は、分析コマンドを実行する表です。

目的

ウェアハウスにデータをロードした後は、統計をリフレッシュして、コストベース・オプティマイザがウェアハウスで推奨するパフォーマンス最適化戦略を確実にする必要があります。WB_ANALYZE_TABLE は、ユーザー・スキーマの特定の表を分析する DBMS_DDL.ANALYZE_OBJECT をコールし、この統計を提供します。表を分析するので、表の行数によっては、時間がかかることがあります。

例

この管理パッケージを使用し、ロードした表で分析コマンドを自動的に実行できます。これは、表にデータをロードするマッピングのマッピング後プロセスを使用して実行できます。

WB_COMPILE_PLSQL

構文

```
WB_COMPILE_PLSQL(p_name, p_type)
```

p_name はコンパイルするオブジェクトの名前で、*p_type* はコンパイルするオブジェクトのタイプです。適切なタイプは次のとおりです。

```
'PACKAGE'
'PACKAGE BODY'
```

```
'PROCEDURE'
'FUNCTION'
'TRIGGER'
```

目的

このプログラム・ユニットでは、データベース内にあるストアド・オブジェクトがコンパイルされます。

WB_DISABLE_ALL_CONSTRAINTS

構文

```
WB_DISABLE_ALL_CONSTRAINTS(p_name)
```

p_name は、制約を無効にする表の名前です。

目的

このプログラム・ユニットでは、プログラム・コールで指定した表によって所有される、すべての制約が無効になります。

データ・セットのロードを高速にするために、表の制約を無効にできます。データは、検証されずにロードされます。これは、比較的クリーンなデータ・セットで主に実行します。

例

次の例では、表 OE.CUSTOMERS の制約を無効にします。

```
select constraint_name
,decode(constraint_type
       , 'C', 'Check'
       , 'P', 'Primary'
       ) Type
,      status
from user_constraints
where table_name = 'CUSTOMERS';
5 rows selected


CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   ENABLED
CUST_LNAME_NN                  Check   ENABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   ENABLED
CUSTOMER_ID_MIN                Check   ENABLED
CUSTOMERS_PK                   Primary ENABLED
```

すべての制約を無効にするには、SQL*Plus または Warehouse Builder で、次を実行します。

```
Execute WB_DISABLE_ALL_CONSTRAINTS('CUSTOMERS');

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   DISABLED
CUST_LNAME_NN                  Check   DISABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   DISABLED
CUSTOMER_ID_MIN                Check   DISABLED
CUSTOMERS_PK                   Primary DISABLED

5 rows selected
```

注意： この文では、カスケード・オプションを使用し、キーを無効にして依存関係を解除できます。

WB_DISABLE_ALL_TRIGGERS

構文

```
WB_DISABLE_ALL_TRIGGERS(p_name)
```

p_name は、トリガーを無効にする表の名前です。

目的

このプログラム・ユニットでは、プログラム・コールで指定した表によって所有される、すべてのトリガーが無効になります。表の所有者は、現行ユーザー（変数 USER）である必要があります。このアクションでは、トリガーが停止し、パフォーマンスが改善されます。

例

次の例では、表 OE.OC_ORDERS のすべてのトリガーを無効にします。

使用可能なトリガー:

```
select trigger_name
,      status
from user_triggers
where table_name = 'OC_ORDERS';

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     ENABLED
ORDERS_ITEMS_TRG               ENABLED
```

すべてのトリガーを無効にするには、SQL*Plus または Warehouse Builder で、次を実行します。

```
Execute WB_DISABLE_ALL_TRIGGERS ('OC_ORDERS');

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     DISABLED
ORDERS_ITEMS_TRG               DISABLED
```

WB_DISABLE_CONSTRAINT

構文

```
WB_DISABLE_CONSTRAINT(p_constraintname, p_tablename)
```

p_constraintname は無効にする制約名で、*p_tablename* は指定した制約を無効にする表の名前です。

目的

このプログラム・ユニットでは、プログラム・コールで指定した表によって所有される特定の制約が無効になります。ユーザーは、現行ユーザー（変数 USER）です。

データ・セットのロードを高速にするために、表の制約を無効にできます。データは、検証されずにロードされます。これにより、オーバーヘッドが削減されます。これは、比較的クリーンなデータ・セットで主に実行します。

例

次の例では、表 OE.CUSTOMERS の特定の制約を無効にします。

```
select constraint_name
, decode(constraint_type
, 'C', 'Check'
, 'P', 'Primary'
) Type
, status
from user_constraints
where table_name = 'CUSTOMERS';

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   ENABLED
CUST_LNAME_NN                  Check   ENABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   ENABLED
CUSTOMER_ID_MIN                Check   ENABLED
CUSTOMERS_PK                   Primary ENABLED

5 rows selected
```

指定した制約を無効にするには、SQL*Plus または Warehouse Builder で、次を実行します。

```
Execute WB_DISABLE_CONSTRAINT('CUSTOMERS_PK','CUSTOMERS');

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   ENABLED
CUST_LNAME_NN                  Check   ENABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   ENABLED
CUSTOMER_ID_MIN                Check   ENABLED
CUSTOMERS_PK                   Primary ENABLED

5 rows selected
```

注意： この文では、カスケード・オプションを使用し、キーを無効にして依存関係を解除できます。

WB_DISABLE_TRIGGER

構文

```
WB_DISABLE_TRIGGER(p_name)
```

p_name は、無効にするトリガー名です。

目的

このプログラム・ユニットでは、指定したトリガーが無効になります。トリガーの所有者は、現行ユーザー（変数 USER）である必要があります。

例

次の例では、表 OE.OC_ORDERS のトリガーを無効にします。

```
select trigger_name, status
from user_triggers
where table_name = 'OC_ORDERS';

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     ENABLED
ORDERS_ITEMS_TRG               ENABLED
```

指定したトリガーを無効にするには、SQL*Plus または Warehouse Builder で、次を実行します。

```
Execute WB_DISABLE_TRIGGER ('ORDERS_TRG');

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     DISABLED
ORDERS_ITEMS_TRG               DISABLED
```

WB_ENABLE_ALL_CONSTRAINTS

構文

```
WB_ENABLE_ALL_CONSTRAINTS(p_name)
```

p_name は、制約を有効にする表の名前です。

目的

このプログラム・ユニットでは、プログラム・コールで指定した表によって所有される、すべての制約が有効になります。

データ・セットのロードを高速にするために、表の制約を無効にできます。データのロード後には、このプログラム・ユニットを使用して、制約を再び有効にする必要があります。

例

次の例では、表 OE.CUSTOMERS の制約を有効にします。

```
select constraint_name
, decode(constraint_type
, 'C', 'Check'
, 'P', 'Primary)
Type
, status
from user_constraints
where table_name = 'CUSTOMERS';

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   DISABLED
CUST_LNAME_NN                  Check   DISABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   DISABLED
CUSTOMER_ID_MIN                Check   DISABLED
CUSTOMERS_PK                   Primary DISABLED

5 rows selected
```

すべての制約を有効にするには、SQL*Plus または Warehouse Builder で、次を実行します。

```
Execute WB_ENABLE_ALL_CONSTRAINTS('CUSTOMERS');

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   ENABLED
CUST_LNAME_NN                  Check   ENABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   ENABLED
CUSTOMER_ID_MIN                Check   ENABLED
CUSTOMERS_PK                   Primary ENABLED

5 rows selected
```

WB_ENABLE_ALL_TRIGGERS

構文

```
WB_ENABLE_ALL_TRIGGERS(p_name)
```

p_name は、トリガーを有効にする表の名前です。

目的

このプログラム・ユニットでは、プログラム・コールで指定した表によって所有される、すべてのトリガーが有効になります。表の所有者は、現行ユーザー（変数 USER）である必要があります。

例

次の例では、表 OE.OC_ORDERS のすべてのトリガーを有効にします。

```
select trigger_name
,      status
from user_triggers
where table_name = 'OC_ORDERS';

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     DISABLED
ORDERS_ITEMS_TRG               DISABLED
```

指定した制約を有効にするには、SQL*Plus または Warehouse Builder で、次を実行します。

```
Execute WB_ENABLE_ALL_TRIGGERS ('OC_ORDERS');

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     ENABLED
ORDERS_ITEMS_TRG               ENABLED
```

WB_ENABLE_CONSTRAINT

構文

```
WB_ENABLE_CONSTRAINT(p_constraintname, p_tablename)
```

p_constraintname は有効にする制約名で、*p_tablename* は指定した制約を有効にする表の名前です。

目的

このプログラム・ユニットでは、プログラム・コールで指定した表によって所有される特定の制約が有効になります。ユーザーは、現行ユーザー（変数 USER）です。データ・セットのロードを高速にするために、表の制約を無効にできます。ロードの完了後には、制約を再び有効にする必要があります。このプログラム・ユニットでは、制約を個別に有効にする方法について説明します。

例

次の例では、表 OE.CUSTOMERS の特定の制約を有効にします。

```
select constraint_name
,      decode(constraint_type
       , 'C', 'Check'
       , 'P', 'Primary'
       ) Type
,      status
from user_constraints
where table_name = 'CUSTOMERS';

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   DISABLED
CUST_LNAME_NN                  Check   DISABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   DISABLED
CUSTOMER_ID_MIN                Check   DISABLED
CUSTOMERS_PK                   Primary DISABLED

5 rows selected
```

指定した制約を有効にするには、SQL*Plus または Warehouse Builder で、次を実行します。

```
Execute WB_ENABLE_CONSTRAINT('CUSTOMERS_PK','CUSTOMERS');

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   DISABLED
CUST_LNAME_NN                  Check   DISABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   DISABLED
CUSTOMER_ID_MIN                Check   DISABLED
CUSTOMERS_PK                   Primary DISABLED

5 rows selected
```

WB_ENABLE_TRIGGER

構文

```
WB_ENABLE_TRIGGER(p_name)
```

p_name は、有効にするトリガー名です。

目的

このプログラム・ユニットでは、指定したトリガーが有効になります。トリガーの所有者は、現行ユーザー（変数 USER）である必要があります。

例

次の例では、表 OE.OC_ORDERS のトリガーを有効にします。

```
select trigger_name
,      status
from user_triggers
where table_name = 'OC_ORDERS';

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     DISABLED
ORDERS_ITEMS_TRG               ENABLED
```

指定した制約を有効にするには、SQL*Plus または Warehouse Builder で、次を実行します。

```
Execute WB_ENABLE_TRIGGER ('ORDERS_TRG');

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     ENABLED
ORDERS_ITEMS_TRG               ENABLED
```

WB_TRUNCATE_TABLE

構文

```
WB_TRUNCATE_TABLE(p_name)
```

p_name は、切り捨てる表の名前です。

目的

このプログラム・ユニットでは、コマンド・コールで指定した表が切り捨てられます。表の所有者は、現行ユーザー（変数 USER）である必要があります。表切捨てコマンドを有効にするために、このコマンドでは、すべての参照制約の無効化と再有効化が行われます。このコマンドをマッピング前プロセスで使用してステージング表を明示的に切り捨て、このステージング表のすべてのデータが新しくロードされたデータであることを保証します。

例

次の例では、表 OE.OC_ORDERS を切り捨てます。

```
select count(*) from oc_orders;

  COUNT(*)
----------
       105
```

指定した制約を有効にするには、SQL*Plus または Warehouse Builder で、次を実行します。

```
Execute WB_TRUNCATE_TABLE ('OC_ORDERS');

  COUNT(*)
----------
         0
```

Character

Character 変換により、Warehouse Builder ユーザーは、Character オブジェクトで変換を実行できます。ここでは、Character 変換をアルファベット順に説明します。Warehouse Builder で提供されるカスタム・ファンクションには、WB_ が接頭辞として付きます。

Warehouse Builder では、次の Character 変換を使用できます。

ASCII

構文

```
ascii::=ASCII(attribute)
```

目的

ASCII は、attribute に指定した最初の文字を、データベース・キャラクタ・セットの 10 進数表現に変換して返します。attribute には、データ型 CHAR、VARCHAR2、NCHAR または NVARCHAR2 を指定できます。戻り値のデータ型は NUMBER です。データベース・キャラクタ・セットが 7 ビット ASCII の場合は、このファンクションは ASCII 値を返します。データベース・キャラクタ・セットが EBCDIC コードの場合は、このファンクションは EBCDIC 値を返します。対応する EBCDIC キャラクタ・ファンクションはありません。

例

次の例では、文字 Q の ASCII 10 進数表現が返されます。

```
SELECT ASCII('Q') FROM DUAL;
ASCII('Q')
----------
81
```

CHR

構文

```
chr::=CHR(attribute)
```

目的

CHR は、attribute に指定された数値と同等の 2 進数が表す文字を、データベース・キャラクタ・セットまたは各国語キャラクタ・セットのどちらかで返します。

USING NCHAR_CS を指定しない場合、このファンクションは、attribute と同等の 2 進数が表す文字を、データベース・キャラクタ・セットの VARCHAR2 値として返します。
Expression Builder で USING NCHAR_CS を指定すると、このファンクションは、attribute と同等の 2 進数が表す文字を各国語キャラクタ・セットの NVARCHAR2 値として返します。

例

次の例は、データベース・キャラクタ・セットが WE8ISO8859P1 として定義されている ASCII ベース・マシンで実行します。

```
SELECT CHR(67)||CHR(65)||CHR(84) "Dog" FROM DUAL;

Dog
---
CAT
```

WE8EBCDIC1047 キャラクタ・セットが定義されている EBCDIC ベース・マシンで同じ結果を出すには、上の例を次のように修正します。

```
SELECT CHR(195)||CHR(193)||CHR(227) "Dog"
FROM DUAL;

Dog
---
CAT
```

次の例では、UTF8 キャラクタ・セットが使用されています。

```
SELECT CHR (50052 USING NCHAR_CS) FROM DUAL;
CH
--
Ä
```

CONCAT

構文

```
concat::=CONCAT(attribute1, attribute2)
```

目的

CONCAT は、attribute1 を attribute2 と連結して返します。attribute1 と attribute2 のデータ型は、CHAR または VARCHAR2 にできます。返される文字列のデータ型は、attribute1 と同じキャラクタ・セットに含まれる VARCHAR2 になります。このファンクションは、連結演算子（||）と同じです。

例

次の例では、ネストを使用して、3 つの文字列を連結しています。

```
SELECT CONCAT(CONCAT(last_name, '''s job category is '),
job_id) "Job"
FROM employees
WHERE employee_id = 152;

Job
----------------------------------------------------
Hall's job category is SA_REP
```

CONVERT

構文

```
convert::=CONVERT(attribute, dest_char_set, source_char_set)
```

目的

CONVERT は、attribute に指定されている文字列を、あるキャラクタ・セットから別のキャラクタ・セットに変換します。戻り値のデータ型は VARCHAR2 です。

- attribute1 引数は、変換する値です。データ型は CHAR または VARCHAR2 にできます。
- *dest_char_set* 引数は、変換後の *attribute1* に使用するキャラクタ・セットの名前です。
- source_char_set 引数は、attribute1 のデータベースへの格納に使用されているキャラクタ・セットの名前です。デフォルト値は、データベース・キャラクタ・セットです。

変換後のキャラクタ・セット引数と変換前のキャラクタ・セット引数は、キャラクタ・セットの名前を含むリテラルまたは列にすることができます。Character 変換が完全に対応するには、変換前キャラクタ・セットで定義されている文字表現が、すべて変換後キャラクタ・セットに含まれる必要があります。変換後キャラクタ・セットに文字がない場合、文字は置換文字で置き換えられます。置換文字は、キャラクタ・セット定義の一部として定義できます。

例

次の例では、Latin-1 文字列から ASCII にキャラクタ・セットが変換されます。結果は、WE8ISO8859P1 データベースから US7ASCII データベースに、同じ文字列をインポートすることと同じです。

```
SELECT CONVERT('Ä Ê Í Õ Ø A B C D E ', 'US7ASCII', 'WE8ISO8859P1')
FROM DUAL;

CONVERT('ÄÊÍÕØABCDE'
--------------------
A E I ? ? A B C D E ?
```

一般的なキャラクタ・セットは次のとおりです。

- US7ASCII: US 7 ビット ASCII キャラクタ・セット
- WE8DEC: 西ヨーロッパ 8 ビット・キャラクタ・セット
- WE8HP: HP 西ヨーロッパ Laserjet 8 ビット・キャラクタ・セット
- F7DEC: DEC フランス 7 ビット・キャラクタ・セット
- WE8EBCDIC500: IBM 西ヨーロッパ EBCDIC コード・ページ 500
- WE8PC850: IBM PC コード・ページ 850
- WE8ISO8859P1: ISO 8859-1 西ヨーロッパ 8 ビット・キャラクタ・セット

INITCAP

構文

```
initcap::=INITCAP(attribute)
```

目的

INITCAP は、attribute に指定された各語句の先頭文字を大文字に、その他すべての文字を小文字にして返します。語句は、空白または英数字以外の文字で区切られます。attribute のデータ型は、CHAR または VARCHAR2 にできます。戻り値のデータ型は、attribute と同じです。

例

次の例では、文字列の各語句の先頭文字が大文字になります。

```
SELECT INITCAP('the soap') "Capitals" FROM DUAL;
Capitals
---------
The Soap
```

INSTR/INSTRB

構文

```
instr::=INSTR(attribute1, attribute2, n, m)
instrb::=INSTRB(attribute1, attribute2, n, m)
```

目的

INSTR は、attribute1 の先頭から n 番目の文字から検索し、attribute2 が m 番目に出現する位置を返します。検索により合致した attribute2 の先頭文字の、attribute1 における文字位置を返します。INSTRB は、文字の代わりにバイトを使用します。

n を負にすると、Oracle は attribute1 の末尾から反対方向に検索します。m の値は正にする必要があります。n と m のデフォルト値は 1 です。つまり、Oracle は attribute1 の先頭文字から検索し、attribute2 の最初の出現位置を返します。戻り値は、n の値に関係なく、attribute1 の先頭から数えた位置になり、文字で表現されます。検索が成功しなかった場合、つまり、attribute1 の n 番目以降に attribute2 が m 回出現しない場合、戻り値は 0 になります。

例

次の例では、文字列「CORPORATE FLOOR」の先頭から 3 番目の文字から、文字列「OR」が検索されます。CORPORATE FLOOR で、OR が 2 回目に出現した位置が返されます。

```
SELECT INSTR('CORPORATE FLOOR','OR', 3, 2) "Instring" FROM DUAL;

Instring
----------
14
```

次の例では、末尾から 3 番目の文字から検索が始まります。

```
SELECT INSTR('CORPORATE FLOOR','OR', -3, 2)
"Reversed Instring"
FROM DUAL;

Reversed Instring
-----------------
2
```

次の例では、ダブルバイト・データベース・キャラクタ・セットを使用します。

```
SELECT INSTRB('CORPORATE FLOOR','OR',5,2) "Instring in bytes"
FROM DUAL;

Instring in bytes
-----------------
27
```

LENGTH/LENGTHB

構文

```
length::=LENGTH(attribute)
lengthb::=LENGTHB(attribute)
```

目的

長さ関連のファンクションでは、文字の長さが返されます。LENGTH は、入力キャラクタ・セットにより定義された文字列を使用して長さを計算します。LENGTHB は、文字の代わりにバイトを使用します。attribute のデータ型は、CHAR または VARCHAR2 にできます。戻り値のデータ型は NUMBER です。attribute のデータ型が CHAR の場合は、後続するすべての空白が長さに含まれます。attribute に NULL 値が含まれる場合は、NULL を返します。

例

次の例では、シングルバイト・データベース・キャラクタ・セットとマルチバイト・データベース・キャラクタ・セットをそれぞれ使用して、LENGTH ファンクションを実行します。

```
SELECT LENGTH('CANDIDE') "Length in characters"
FROM DUAL;

Length in characters
--------------------
7
```

次の例では、ダブルバイト・データベース・キャラクタ・セットを使用します。

```
SELECT LENGTHB ('CANDIDE') "Length in bytes"
FROM DUAL;

Length in bytes
---------------
14
```

LOWER

構文

```
lower::=LOWER(attribute)
```

目的

LOWER は、attribute のすべての文字を小文字にして返します。attribute のデータ型は、CHAR または VARCHAR2 にできます。戻り値のデータ型は、attribute と同じです。

例

次の例では、小文字の文字列が返されます。

```
SELECT LOWER('MR. SCOTT MCMILLAN') "Lowercase"
FROM DUAL;

Lowercase
--------------------
mr. scott mcmillan
```

LPAD

構文

```
lpad::=LPAD(attribute1, n, attribute2)
```

目的

LPAD は、attribute1 の長さが n になるまで、attribute2 の文字シーケンスを左側に埋め込んで返します。attribute2 のデフォルトは、空白 1 個です。attribute1 が n より長い場合、このファンクションは n 文字分の attribute1 を返します。

attribute1 と attribute2 のデータ型は、CHAR または VARCHAR2 にできます。返される文字列のデータ型は VARCHAR2 になり、attribute1 と同じキャラクタ・セットに含まれます。引数 n は、画面に表示される戻り値の全長です。ほとんどのキャラクタ・セットでは、戻り値の文字数にもなります。しかし、一部のマルチバイト・キャラクタ・セットでは、表示される文字列の長さが、文字列の文字数と異なることがあります。

例

次の例では、文字「*.」が文字列の左に埋め込まれます。

```
SELECT LPAD('Page 1',15,'*.') "LPAD example"
FROM DUAL;

LPAD example
---------------
*.*.*.*.*Page 1
```

LTRIM

構文

```
ltrim::=LTRIM(attribute, set)
```

目的

LTRIM は、attribute の左から文字を削除しますが、set で指定した文字と同じ文字で attribute の左端にあるものをすべて削除します。set のデフォルトは、空白 1 個です。attribute を文字リテラルにする場合は、一重引用符で囲む必要があります。Warehouse Builder は、attribute の先頭文字からスキャンを始め、set とは異なる文字になるまで、set と同じ文字をすべて削除します。次に結果を返します。

attribute と set のデータ型は、CHAR または VARCHAR2 にできます。返される文字列のデータ型は VARCHAR2 になり、attribute と同じキャラクタ・セットに含まれます。

例

次の例では、左端にある x と y のすべてが文字列から削除されます。

```
SELECT LTRIM('xyxXxyLAST WORD','xy') "LTRIM example"
FROM DUAL;

LTRIM example
------------
XxyLAST WORD
```

NLSSORT

構文

```
nlssort::=NLSSORT(attribute, nlsparam)
```

目的

NLSSORT は、attribute のソートに使用するバイトの文字列を返します。パラメータ attribute の型は VARCHAR2 です。このファンクションを使用すると、文字列のバイナリ値でなく基準の言語ソートに基づいて比較されます。

nlsparam の値は、'NLS_SORT = sort' という形式にできます。sort には、言語ソート基準の名前または BINARY を指定します。nlsparams を省略すると、このファンクションはセッションのデフォルトのソート基準を使用します。

例

次の例では、2 つの値を含む表が作成され、NLSSORT ファンクションによって戻り値をどのように順序付けできるかが示されています。

```
CREATE TABLE test (name VARCHAR2(15));
INSERT INTO TEST VALUES ('Gaardiner');
INSERT INTO TEST VALUES ('Gaberd');

SELECT * FROM test ORDER BY name;

NAME
------
Gaardiner
Gaberd

SELECT *
   FROM test
   ORDER BY NLSSORT(name, 'NLSSORT = XDanish');

Name
------
Gaberd
Gaardiner
```

NLS_INITCAP

構文

```
nls_initcap::=NLS_INITCAP(attribute, nlsparam)
```

目的

NLS_INITCAP は、attribute に指定された各語句の先頭文字を大文字に、その他すべての文字を小文字にして返します。語句は、空白または英数字以外の文字で区切られます。

nlsparam の値は、'NLS_SORT = sort' という形式にできます。sort には、言語ソート基準の名前または BINARY を指定します。言語ソート基準は、大 / 小文字の変換を行うときの特殊な言語要件を扱います。これらの要件によって、戻り値が attribute と異なる長さになる場合があります。'nlsparams' を省略すると、このファンクションはセッションのデフォルトのソート基準を使用します。

例

次の例では、言語ソート基準とファンクションで、戻り値がどのように異なるかを示します。

```
SELECT NLS_INITCAP('ijsland') "InitCap"
   FROM dual;

InitCap
---------
Ijsland

SELECT NLS_INITCAP('ijsland','NLS_SORT=XDutch) "InitCap"
   FROM dual;

InitCap
---------
IJsland
```

NLS_LOWER

構文

```
nls_lower::=NLS_LOWER(attribute, nlsparam)
```

目的

NLS_LOWER は、attribute のすべての文字を小文字にして返します。attribute と nlsparam のデータ型は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB、NCLOB のいずれかにできます。返される文字列のデータ型は VARCHAR2 になり、attribute と同じキャラクタ・セットに含まれます。nlsparam の値は、'NLS_SORT = sort' という形式にできます。sort には、言語ソート基準の名前または BINARY を指定します。

例

次の例では、言語ソート基準 XGerman を使用して、文字列 'citta'' が返されます。

```
SELECT NLS_LOWER('CITTA''','NLS_SORT=XGerman) "Lowercase"
   FROM DUAL;

Lowercase
------------
citta'
```

NLS_UPPER

構文

```
nls_upper::=NLS_UPPER(attribute, nlsparam)
```

目的

NLS_UPPER は、attribute のすべての文字を大文字にして返します。attribute と nlsparam のデータ型は、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB、NCLOB のいずれかにできます。返される文字列のデータ型は VARCHAR2 になり、attribute と同じキャラクタ・セットに含まれます。nlsparam の値は、'NLS_SORT = sort' という形式にできます。sort には、言語ソート基準の名前または BINARY を指定します。

例

次の例では、すべての文字が大文字に変換された文字列が返されます。

```
SELECT NLS_UPPER('große') "Uppercase"
   FROM DUAL;

Uppercase
------------
GROßE

SELECT NLS_UPPER('große', 'NLS_SORT=XGerman) "Uppercase"
   FROM DUAL;

Uppercase
------------
GROSSE
```

REPLACE

構文

```
replace::=REPLACE(attribute, 'search_string', 'replace_string')
```

目的

REPLACE は、attribute の中の search_string と一致する語をすべて replacement_string に置き換えて返します。replacement_string を省略するか NULL にすると、search_string と一致する語がすべて削除されます。search_string を NULL にすると、attribute が返されます。

search_string、replacement_string および attribute のデータ型は、CHAR または VARCHAR2 にできます。返される文字列のデータ型は VARCHAR2 になり、attribute と同じキャラクタ・セットに含まれます。

このファンクションでは、TRANSLATE ファンクションによって提供される機能のスーパーセットが提供されます。TRANSLATE では、単一文字が 1 対 1 で置き換えられます。REPLACE では、1 つの文字列を別の文字列で置き換えることができ、さらには文字列を削除することもできます。

例

次の例では、「J」が「BL」で置き換わります。

```
SELECT REPLACE('JACK and JUE','J','BL') "Changes"
FROM DUAL;
Changes
--------------
BLACK and BLUE
```

RPAD

構文

```
rpad::=RPAD(attribute1, n, attribute2)
```

目的

RPAD は、attribute1 の長さが n になるまで、attribute2 を必要な回数だけ右側に埋め込んで返します。attribute2 のデフォルトは、空白 1 個です。attribute1 が n より長い場合、このファンクションは n 文字分の attribute1 を返します。

attribute1 と attribute2 のデータ型は、CHAR または VARCHAR2 にできます。返される文字列のデータ型は VARCHAR2 になり、attribute1 と同じキャラクタ・セットに含まれます。

引数 n は、画面に表示される戻り値の全長です。ほとんどのキャラクタ・セットでは、戻り値の文字数にもなります。しかし、一部のマルチバイト・キャラクタ・セットでは、表示される文字列の長さが、文字列の文字数と異なることがあります。

例

次の例では、長さが 12 文字になるまで、文字「ab」が名前の右に埋め込まれます。

```
SELECT RPAD('MORRISON',12,'ab') "RPAD example"
FROM DUAL;
RPAD example
-----------------
MORRISONabab
```

RTRIM

構文

```
rtrim::=RTRIM(attribute, set)
```

目的

RTRIM は、set で指定した文字と同じ文字で、attribute の右端にあるものをすべて削除して返します。set のデフォルトは、空白 1 個です。attribute を文字リテラルにする場合は、一重引用符で囲む必要があります。RTRIM は、LTRIM と同じように動作します。attribute と set のデータ型は、CHAR または VARCHAR2 にできます。返される文字列のデータ型は VARCHAR2 になり、attribute と同じキャラクタ・セットに含まれます。

例

次の例では、文字列の右側から文字「xy」が削除されます。

```
SELECT RTRIM('BROWNINGyxXxy','xy') "RTRIM e.g."
FROM DUAL;

RTRIM e.g
-------------
BROWNINGyxX
```

SOUNDEX

構文

```
soundex::=SOUNDEX(attribute)
```

目的

SOUNDEX は、attribute の発音表現を含む文字列を返します。このファンクションでは、スペルは異なるが英語の発音が似ている語句を比較できます。

発音表現は、Donald E. Knuth 氏の『The Art of Computer Programming, Volume 3: Sorting and Searching』で、次のように定義されています。

- 文字列の先頭文字を残し、先頭文字以外の a、e、h、i、o、u、w、y をすべて削除します。
- 次のように、残りの文字（先頭文字以外）に数値を割り当てます。
  - b、f、p、v = 1
  - c、g、j、k、q、s、x、z = 2
  - d、t = 3
  - l = 4
  - m、n = 5
  - r = 6
- 元の名前（手順 1 を実行する前）で同じ数値の複数の文字が隣り合う場合または h と w を除けば隣り合う場合は、最初の文字を残してその他すべてを省略します。
- 0 を埋め込んだ最初の 4 バイトを返します。

attribute のデータ型は、CHAR または VARCHAR2 にできます。戻り値のデータ型は、attribute と同じです。

例

次の例では、姓の発音表現が「Smyth」になる従業員が返されます。

```
SELECT last_name, first_name
FROM hr.employees
WHERE SOUNDEX(last_name)
= SOUNDEX('SMYTHE');


LAST_NAME  FIRST_NAME
---------- ----------
Smith      Lindsey
```

SUBSTR

構文

```
substr::=SUBSTR(attribute, position, substring_length)
substrb::=SUBSTRB(attribute, position, substring_length)
```

目的

サブストリング関連のファンクションは、attribute の指定された位置から始まる、substring_length に指定された長さの文字を返します。SUBSTR 句は、入力キャラクタ・セットにより定義される文字を使用して長さを計算します。SUBSTRB は、文字の代わりにバイトを使用します。

- position を 0 にすると、1 として扱われます。
- position を正にした場合、Warehouse Builder は、attribute の先頭から数えて最初の文字を特定します。
- position を負にした場合、Warehouse Builder は、attribute の末尾から反対方向に数えます。
- substring_length を省略すると、Warehouse Builder は、attribute の末尾までの文字をすべて返します。substring_length を 1 より小さくすると、null が返されます。

attribute のデータ型は、CHAR または VARCHAR2 にできます。戻り値のデータ型は、attribute と同じです。浮動小数点数を引数として SUBSTR に渡すと、整数に自動的に変換されます。

例

次の例では、「ABCDEFG」の指定されたサブストリングが返されます。

```
SELECT SUBSTR('ABCDEFG',3,4) "Substring"
FROM DUAL;

Substring
---------
CDEF

SELECT SUBSTR('ABCDEFG',-5,4) "Substring"
FROM DUAL;

Substring
---------
CDEF
```

ダブルバイト・データベース・キャラクタ・セットの場合は、次のようになります。

```
SELECT SUBSTRB('ABCDEFG',5,4.2) "Substring with bytes"
FROM DUAL;

Substring with bytes
--------------------
CD
```

TO_DATE

構文

```
to_date::=TO_DATE(attribute, fmt, nlsparam)
```

目的

TO_DATE は、データ型が CHAR または VARCHAR2 の attribute を、データ型 DATE の値に変換します。fmt は、attribute のフォーマットを指定する日付フォーマットです。fmt を省略した場合は、attribute をデフォルトの日付フォーマットにする必要があります。fmt をユリウス暦を表す「J」にした場合は、attribute を整数にする必要があります。このファンクションの nlsparam は、Conversion 変換の TO_CHAR ファンクションと同じ目的で動作します。

attribute 引数に DATE 値を指定して TO_DATE ファンクションを使用しないでください。fmt またはデフォルトの日付フォーマットによっては、返される DATE 値の最初の 2 桁が、元の attribute と異なる場合があります。

例

次の例では、文字列が日付に変換されます。

```
SELECT TO_DATE(
'January 15, 1989, 11:00 A.M.',
'Month dd, YYYY, HH:MI A.M.',
'NLS_DATE_LANGUAGE = American')
FROM DUAL;

TO_DATE
---------
15-JAN-89
```

TO_MULTI_BYTE

構文

```
to_multi_byte::=TO_MULTI_BYTE(attribute)
```

目的

TO_MULTI_BYTE は、attribute のすべてのシングルバイト文字を、対応するマルチバイト文字に変換して返します。attribute のデータ型は、CHAR または VARCHAR2 にできます。戻り値のデータ型は、attribute と同じです。attribute のシングルバイト文字のうち、対応するマルチバイト文字がないものは、出力文字列でシングルバイト文字として表示されます。

このファンクションは、シングルバイト文字とマルチバイト文字の両方がデータベース・キャラクタ・セットに含まれている場合にかぎって便利です。

例

次の例では、シングルバイトの「A」がマルチバイトに変換されます。

```
'A' in UTF8:
SELECT dump(TO_MULTI_BYTE( 'A')) FROM DUAL;
DUMP(TO_MULTI_BYTE('A'))
------------------------
Typ=1 Len=3: 239,188,161
```

TO_NUMBER

構文

```
to_number::=TO_NUMBER(attribute, fmt, nlsparam)
```

目的

TO_NUMBER は、オプション・フォーマット model_fmt で指定されたフォーマットの数字を含む CHAR または VARCHAR2 データ型値の attribute を、NUMBER データ型の値に変換します。

例

次の例では、文字列データが数値に変換されます。

```
UPDATE employees
SET salary = salary + TO_NUMBER('100.00', '9G999D99')
WHERE last_name = 'Perkins';
```

このファンクションの nlsparam 文字列は、数値変換の TO_CHAR ファンクションと同じ目的で動作します。

```
SELECT TO_NUMBER('-AusDollars100','L9G999D99',
' NLS_NUMERIC_CHARACTERS = '',.''
NLS_CURRENCY = ''AusDollars''
') "Amount"
FROM DUAL;

Amount
----------
-100
```

TO_SINGLE_BYTE

構文

```
to_single_byte::=TO_SINGLE_BYTE(attribute)
```

目的

TO_SINGLE_BYTE は、attribute のすべてのマルチバイト文字を、対応するシングルバイト文字に変換して返します。attribute のデータ型は、CHAR または VARCHAR2 にできます。戻り値のデータ型は、attribute と同じです。attribute のマルチバイト文字のうち、対応するシングルバイト文字がないものは、出力でマルチバイト文字として表示されます。

このファンクションは、シングルバイト文字とマルチバイト文字の両方がデータベース・キャラクタ・セットに含まれている場合にかぎって便利です。

例

次の例では、UTF8 のマルチバイト文字「A」が、シングルバイトの ASCII 文字「A」に変換されます。

```
SELECT TO_SINGLE_BYTE( CHR(15711393)) FROM DUAL;
T
-
A
```

TRANSLATE

構文

```
translate::=TRANSLATE(attribute, from_string, to_string)
```

目的

TRANSLATE は、attribute の中の、from_string の各文字と一致するすべての文字を to_string に指定された文字で置き換えて返します。attribute の文字のうち from_string に含まれない文字は、置き換わりません。引数 from_string には、to_string より多くの文字を含めることができます。この場合、from_string の末尾にある余分な文字は、to_string の文字と対応しません。こうした余分な文字が attribute にある場合、余分な文字は戻り値から削除されます。

to_string に空文字列を使用し、from_string と一致するすべての文字を戻り値から削除することはできません。空文字列は NULL として解釈されます。このファンクションは、NULL 引数が含まれる場合、null を返します。

例

次の文では、ライセンス番号が変換されます。すべての文字「ABC...Z」は「X」に、すべての数字「012 . . . 9」は「9」に変換されます。

```
SELECT TRANSLATE('2KRW229',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'9999999999XXXXXXXXXXXXXXXXXXXXXXXXXX') "License"
FROM DUAL;
License
--------
9XXX999
```

次の文では、文字が削除されて数字が残った状態でライセンス番号が返されます。

```
SELECT TRANSLATE('2KRW229',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ', '0123456789') "Translate example"
FROM DUAL;

Translate example
-----------------
2229
```

TRIM

構文

```
trim::=TRIM(attribute)
```

目的

TRIM では、先頭の空白または末尾の空白、あるいはその両方を文字列から削除できます。このファンクションは、データ型が VARCHAR2 の値を返します。値の最大長は、attribute の長さです。

例

次の例では、先頭の空白と末尾の空白が文字列から削除されます。

```
SELECT TRIM ('   Warehouse    ') "TRIM Example"
FROM DUAL;
TRIM example
------------
Warehouse
```

UPPER

構文

```
upper::=UPPER(attribute)
```

目的

UPPER は、attribute のすべての文字を大文字にして返します。attribute のデータ型は、CHAR または VARCHAR2 にできます。戻り値のデータ型は、attribute と同じです。

例

次の例では、大文字の文字列が返されます。

```
SELECT UPPER('Large') "Uppercase"
FROM DUAL;
Upper
-----
LARGE
```

WB.LOOKUP_CHAR

構文

```
WB.LOOKUP_CHAR (table_name
, column_name
, key_column_name
, key_value
)
```

table_name は参照を実行する表の名前で、column_name は参照結果などで返される VARCHAR2 列の名前です。key_column_name は参照表で照合するキーとして使用される NUMBER 列の名前です。key_value は、照合を実行する key_column_name にマッピングされるキー列の値です。

目的

照合キーとして NUMBER 列を使用し、数値でキー参照を実行して、データベース表から VARCHAR2 値を返します。

例

参照表 LKP1 として、次の表を使用するとします。

```
KEY_COLUMN    TYPE    COLOR
10            Car     Red
20            Bike    Green
```

次のようなコールで、このパッケージを使用します。

```
WB.LOOKUP_CHAR ('LKP1'
, 'TYPE'
, 'KEYCOLUMN'
, 20
)
```

この場合は、この変換の出力として値 'Bike' が返されます。この出力は、インライン・ファンクション・コールの結果として、マッピングで処理されます。

> **注意：** このファンクションは、行ベースのキー参照です。セット・ベースの参照は、参照演算子を使用した場合にサポートされます。

WB.LOOKUP_CHAR

Syntax

```
WB.LOOKUP_CHAR (table_name
, column_name
, key_column_name
, key_value
)
```

table_name は参照を実行する表の名前で、column_name は参照結果などで返される VARCHAR2 列の名前です。key_column_name は参照表で照合するキーとして使用される VARCHAR2 列の名前です。key_value は、照合を実行する key_column_name にマッピングされる値などのキー列値です。

目的

照合キーとして VARCHAR2 列を使用し、VARCHAR2 値でキー参照を実行して、データベース表から VARCHAR2 値を返します。

例

参照表 LKP1 として、次の表を使用するとします。

```
KEYCOLUMN  TYPE  COLOR
ACV        Car   Red
ACP        Bike  Green
```

次のようなコールで、このパッケージを使用します。

```
WB.LOOKUP_CHAR ('LKP1'
, 'TYPE'
, 'KEYCOLUMN'
, 'ACP'
)
```

この場合は、この変換の出力として値 'Bike' が返されます。この出力は、インライン・ファンクション・コールの結果として、マッピングで処理されます。

> **注意：** このファンクションは、行ベースのキー参照です。セット・ベースの参照は、参照演算子を使用した場合にサポートされます。

WB_IS_SPACE

構文

```
WB_IS_SPACE(attibute)
```

目的

文字列値に空白のみが含まれているかどうかをチェックします。メインフレーム・ソースでは、固定長フォーマットにファイルを合せるため、一部のフィールドに多くの空白が含まれます。このファンクションでは、この空白をチェックできます。このファンクションは、常にブール値を返します。

例

WB_IS_SPACE は、attribute に空白のみが含まれる場合、true を返します。

Date

Warehouse Builder ユーザーは、Date 変換により、日付属性で変換を実行できます。ここでは、この変換について順番に説明します。Warehouse Builder で提供されるカスタム・ファンクションは、すべて WB_<function name> というフォーマットになっています。

次の項では、Warehouse Builder で提供されるすべての Date 変換について、アルファベット順に説明します。

ADD_MONTHS

構文

```
add_months::=ADD_MONTHS(attribute, n)
```

目的

ADD_MONTHS は、attribute プラス n か月の日付を返します。引数 n は、任意の整数にすることができます。これは、一般的に、attribute または定数に追加されます。

attribute の日付が月末である場合、または算出される月の日数が attribute の日付より少ない場合、結果は算出される月の最終日になります。他の場合、結果は attribute と同じ日付になります。

例

次の例では、サンプル表 employees で、hire_date の 1 か月後が返されます。

```
SELECT TO_CHAR(ADD_MONTHS(hire_date,1), 'DD-MON-YYYY') "Next month"
FROM employees
WHERE last_name = 'Baer';
Next Month
-----------
07-JUL-1994
```

LAST_DAY

構文

```
last_day::=LAST_DAY(attribute)
```

目的

LAST_DAY は、attribute の日付が含まれる月の最終日の日付を返します。

例

次の文では、現行月の残っている日数を判断できます。

```
SELECT SYSDATE,
LAST_DAY(SYSDATE) "Last",
LAST_DAY(SYSDATE) - SYSDATE "Days Left"
FROM DUAL;
SYSDATE   Last Days Left
--------- --------- ----------
23-OCT-97 31-OCT-97 8
```

MONTHS_BETWEEN

構文

```
months_between::=MONTHS_BETWEEN(attribute1, attribute2)
```

目的

MONTHS_BETWEEN は、attribute1 の日付と attribute2 の日付の間にある月数を返します。attribute1 が attribute2 より後である場合、結果は正になります。前である場合、結果は負になります。

attribute1 と attribute2 が月の同じ日付または異なる月の末日の場合は、結果は常に整数になります。他の場合は、Oracle は、1 か月を 31 日として、attribute1 と attribute2 の時刻の差を考慮し、結果の小数部を計算します。

例

次の例では、2 つの日付の間で月数を計算します。

```
SELECT MONTHS_BETWEEN
(TO_DATE('02-02-1995','MM-DD-YYYY'),
TO_DATE('01-01-1995','MM-DD-YYYY') ) "Months"
FROM DUAL;

Months
----------
1.03225806
```

NEW_TIME

構文

```
new_time::=NEW_TIME(attribute, zone1, zone2)
```

目的

NEW_TIME は、タイム・ゾーン zone1 の日時値である attribute を、zone2 の日時に変換して返します。このファンクションを使用する前に、NLS_DATE_FORMAT パラメータを設定して、24 時間制の時刻にする必要があります。

引数 zone1 と zone2 は、次の中の任意のテキスト文字列にすることができます。

- AST、ADT: 大西洋標準時または夏時間
- BST、BDT: ベーリング標準時または夏時間
- CST、CDT: 中央標準時または夏時間
- EST、EDT: 東部標準時または夏時間
- GMT: グリニッジ標準時
- HST、HDT: アラスカ・ハワイ標準時または夏時間

- MST、MDT: 山岳部標準時または夏時間
- NST: ニューファンドランド標準時
- PST、PDT: 太平洋標準時または夏時間
- YST、YDT: ユーコン標準時または夏時間

例

次の例では、大西洋標準時を太平洋標準時に変換して返します。

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
SELECT NEW_TIME
(TO_DATE('11-10-99 01:23:45', 'MM-DD-YY HH24:MI:SS'),
'AST', 'PST') "New Date and Time"
FROM DUAL;

New Date and Time
--------------------
09-NOV-1999 21:23:45
```

NEXT_DAY

構文

```
next_day::=NEXT_DAY(attribute, attribute2)
```

目的

NEXT_DAY は、attribute1 の日付よりも後の日付で、attribute2 の文字列により指定された曜日の最初の日付を返します。引数 attribute2 は、セッションの日付言語で表現した曜日（完全な名前か短縮形）にする必要があります。必要最低文字数は、短縮形の文字数です。有効な短縮形に続く文字は無視されます。戻り値には、引数 attribute1 と同じ時、分、秒が含まれます。

例

次の例では、2001 年 2 月 2 日以降の最初の火曜日にあたる日付が返されます。

```
SELECT NEXT_DAY('02-FEB-2001','TUESDAY') "NEXT DAY"
FROM DUAL;
 NEXT DAY
-----------
06-FEB-2001
```

ROUND（日付）

構文

```
round_date::=ROUND(attribute, fmt)
```

目的

ROUND は、attribute の日付を、フォーマット・モデル fmt で指定される単位に丸めて返します。fmt を省略した場合、日付は最も近い日に丸められます。

例
次の例では、次の年の初日に日付が丸められます。

```
SELECT ROUND (TO_DATE ('27-OCT-00'),'YEAR') "New Year"
FROM DUAL;

New Year
---------
01-JAN-01
```

SYSDATE

構文
```
sysdate::=SYSDATE
```

目的
SYSDATE は、現在の日時を返します。戻り値のデータ型は DATE です。この関数は引数を必要としません。分散型 SQL 文では、この関数により、ローカル・データベースの日時が返されます。CHECK 制約の条件にこの関数を使用することはできません。

例
次の例では、現在の日時が返されます。

```
SELECT TO_CHAR(SYSDATE, 'MM-DD-YYYY HH24:MI:SS')"NOW" FROM DUAL;

NOW
-------------------
04-13-2001 09:45:51
```

TO_CHAR（日時）

構文
```
to_char_date::=TO_CHAR(attribute, fmt, nlsparam)
```

目的
TO_CHAR は、DATE データ型の attribute を、日付フォーマット fmt で指定されるフォーマットの VARCHAR2 データ型値に変換します。fmt を省略した場合、日付はデフォルト日付フォーマットの VARCHAR2 値に変換されます。

nlsparams には、月日の名前と短縮形を返す言語を指定します。この引数は、'NLS_DATE_LANGUAGE = language' という形式にできます。nlsparams を省略すると、このファンクションはセッションのデフォルト日付言語を使用します。

例
次の例では、データベースのシステム日付で様々な変換が行われます。

```
select to_char(sysdate) no_fmt
from dual;
NO_FMT
---------
26-MAR-02

select to_char(sysdate, 'dd-mm-yyyy') fmted
from dual;

FMTED
----------
26-03-2002
```

TRUNC（日付）

構文

```
trunc_date::=TRUNC(attribute, fmt)
```

目的

TRUNC は、attribute の日付の時刻部分を、フォーマット・モデル fmt で指定される単位に切り捨てて返します。fmt を省略した場合、日付は最も近い日に丸められます。

例

次の例では、日付が切り捨てられます。

```
SELECT TRUNC(TO_DATE('27-OCT-92','DD-MON-YY'), 'YEAR') "New Year"
FROM DUAL;

New Year
---------
01-JAN-92
```

WB_CAL_MONTH_NAME

構文

```
WB_CAL_MONTH_NAME(attribute)
```

目的

このファンクション・コールでは、attribute で指定した日付の月名が、完全な長さの名前で返されます。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_CAL_MONTH_NAME(sysdate)
from dual;

WB_CAL_MONTH_NAME(SYSDATE)
---------------------------
March

select WB_CAL_MONTH_NAME('26-MAR-2002')
from dual;

WB_CAL_MONTH_NAME('26-MAR-2002')
---------------------------------
March
```

WB_CAL_MONTH_OF_YEAR

構文

```
WB_CAL_MONTH_OF_YEAR(attribute)
```

目的

WB_CAL_MONTH_OF_YEAR は、attribute で指定される日付の月（1 から 12）を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_CAL_MONTH_OF_YEAR(sysdate) month
from dual;

     MONTH
----------
         3

select WB_CAL_MONTH_OF_YEAR('26-MAR-2002') month
from dual;

     MONTH
----------
         3
```

WB_CAL_MONTH_SHORT_NAME

構文

```
WB_CAL_MONTH_SHORT_NAME(attribute)
```

目的

WB_CAL_MONTH_SHORT_NAME は、attribute で指定される日付の月の短縮名（「Jan」など）を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_CAL_MONTH_SHORT_NAME (sysdate) month
from dual;

MONTH
---------
Mar

select WB_CAL_MONTH_SHORT_NAME ('26-MAR-2002') month
from dual;

MONTH
---------
Mar
```

WB_CAL_QTR

構文

```
WB_CAL_QTR(attribute)
```

目的

WB_CAL_QTR は、attribute で指定される日付のグレゴリオ暦の四半期（1 月から 3 月の場合は 1）を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_CAL_QTR (sysdate) quarter
from dual;

   QUARTER
----------
         1

select WB_CAL_QTR ('26-MAR-2002') quarter
from dual;

   QUARTER
----------
         1
```

WB_CAL_WEEK_OF_YEAR

構文

```
WB_CAL_WEEK_OF_YEAR(attribute)
```

目的

WB_CAL_WEEK_OF_YEAR は、日付 attribute の年の週（1 から 53）を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_CAL_WEEK_OF_YEAR (sysdate) w_of_y
from dual;

    W_OF_Y
----------
        13

select WB_CAL_WEEK_OF_YEAR ('26-MAR-2002') w_of_y
from dual;

    W_OF_Y
----------
        13
```

WB_CAL_YEAR

構文

```
WB_CAL_YEAR(attribute)
```

目的

WB_CAL_YEAR は、attribute で指定された日付の年度の数値表現を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_CAL_YEAR (sysdate) year
from dual;

      YEAR
```

```
----------
      2002

select WB_CAL_YEAR ('26-MAR-2002') w_of_y
from dual;

      YEAR
----------
      2002
```

WB_CAL_YEAR_NAME

構文

```
WH_CAL_YEAR_NAME(attribute)
```

目的

WB_CAL_YEAR_NAME は、attribute で指定される日付の年度のスペル・アウト文字名を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_CAL_YEAR_NAME (sysdate) name
from dual;

NAME
-----------------------------------------------
Two Thousand Two

select WB_CAL_YEAR_NAME ('26-MAR-2001') name
from dual;

NAME
-----------------------------------------------
Two Thousand One
```

WB_DATE_FROM_JULIAN

構文

```
WB_DATE_FROM_JULIAN(attribute)
```

目的

WB_DATE_FROM_JULIAN は、ユリウス暦の日付 attribute を通常の日付に変換します。

例

次の例では、指定したユリウス暦の日付に対する値が返されます。

```
select to_char(WB_DATE_FROM_JULIAN(3217345),'dd-mon-yyyy') JDate
from dual;

JDATE
-----------
08-sep-4096
```

WB_DAY_NAME

構文

```
WB_DAY_NAME(attribute)
```

目的

WB_DAY_NAME は、日付 attribute の完全な曜日名を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_DAY_NAME (sysdate) name
from dual;

NAME
--------------------------------------------
Thursday

select WB_DAY_NAME ('26-MAR-2002') name
from dual;

NAME
--------------------------------------------
Tuesday
```

WB_DAY_OF_MONTH

構文

```
WB_DAY_OF_MONTH(attribute)
```

目的

WB_DAY_OF_MONTH は、attribute で指定される日付の、月内での日付番号を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_DAY_OF_MONTH (sysdate) num
from dual;

       NUM
----------
        28

select WB_DAY_OF_MONTH ('26-MAR-2002') num
from dual

       NUM
----------
        26
```

WB_DAY_OF_WEEK

構文

```
WB_DAY_OF_WEEK(attribute)
```

目的

WB_DAY_OF_WEEK は、attribute で指定される日付の、週内での日付番号をデータベース・カレンダに基づいて返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_DAY_OF_WEEK (sysdate) num
from dual;

       NUM
----------
         5

select WB_DAY_OF_WEEK ('26-MAR-2002') num
from dual;


       NUM
----------
         3
```

WB_DAY_OF_YEAR

構文

```
WB_DAY_OF_YEAR(attribute)
```

目的

WB_DAY_OF_YEAR は、attribute で指定される日付の、年度内での日付番号を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_DAY_OF_YEAR (sysdate) num
from dual;

       NUM
----------
        87

select WB_DAY_OF_YEAR ('26-MAR-2002') num
from dual;

       NUM
----------
        85
```

WB_DAY_SHORT_NAME

構文

```
WB_DAY_SHORT_NAME(attribute)
```

目的

WB_DAY_SHORT_NAME は、日付 attribute の曜日を 3 文字で表現した短縮名を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_DAY_SHORT_NAME  (sysdate) abbr
from dual;

ABBR
-------------------------------------
Thu

select WB_DAY_SHORT_NAME  ('26-MAR-2002') abbr
from dual;

NUM
-------------------------------------
Tue
```

WB_DECADE

構文

```
WB_DECADE(attribute)
```

目的

WB_DECADE は、attribute で指定される日付の、世紀内での年度を 10 年単位までで返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_DECADE  (sysdate) dcd
from dual;

       DCD
----------
         2

select WB_DECADE  ('26-MAR-2002') DCD
from dual;

       DCD
----------
         2
```

WB_HOUR12

構文

```
WB_HOUR12(attribute)
```

目的

WB_HOUR12 は、attribute に指定される日付の時間部分（12 時間設定）を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_HOUR12 (sysdate) h12
from dual;

       H12
----------
         9

select WB_HOUR12 ('26-MAR-2002') h12
from dual;

       H12
----------
        12
```

注意： 2 番目の例のように、タイムスタンプを含まない日付の場合、Oracle は深夜 12:00 のタイムスタンプを使用するので、この場合は 12 が返されます。

WB_HOUR12MI_SS

構文

```
WB_HOUR12MI_SS(attribute)
```

目的

WB_HOUR12MI_SS は、HH12:MI:SS というフォーマットで、attribute のタイムスタンプを返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_HOUR12MI_SS (sysdate) h12miss
from dual;

H12MISS
-------------------------------------
09:08:52

select WB_HOUR12MI_SS ('26-MAR-2002') h12miss
from dual;

H12MISS
-------------------------------------
12:00:00
```

注意： 2 番目の例のように、タイムスタンプを含まない日付の場合、Oracle は深夜 12:00 のタイムスタンプを使用するので、この場合は 12:00:00 が返されます。

WB_HOUR24

構文

```
WB_HOUR24(attribute)
```

目的

WB_HOUR24 は、attribute に指定される日付の時間部分（24 時間設定）を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_HOUR24 (sysdate) h24
from dual;

       H24
----------
         9

select WB_HOUR24 ('26-MAR-2002') h24
from dual;

       H24
----------
         0
```

注意： 2 番目の例のように、タイムスタンプを含まない日付の場合、Oracle は 00:00:00 というタイムスタンプを使用するので、この場合はこのタイムスタンプが返されます。

WB_HOUR24MI_SS

構文

```
WB_HOUR24MI_SS(attribute)
```

目的

WB_HOUR24MI_SS は、HH24:MI:SS というフォーマットで、attribute のタイムスタンプを返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_HOUR24MI_SS (sysdate) h24miss
from dual;

H24MISS
-----------------------------------
09:11:42

select WB_HOUR24MI_SS ('26-MAR-2002') h24miss
```

```
from dual;

H24MISS
-----------------------------------
00:00:00
```

> **注意：** 2 番目の例のように、タイムスタンプを含まない日付の場合、Oracle は 00:00:00 というタイムスタンプを使用するので、この場合はこのタイムスタンプが返されます。

WB_IS_DATE

構文

```
WB_IS_DATE(attribute, fmt)
```

目的

attribute に有効な日付が含まれているかどうかをチェックします。このファンクションで返されるブール値は、attribute に有効な日付が含まれている場合、true に設定されます。Fmt は、オプションの日付フォーマットです。fmt を省略した場合は、データベース・セッションの日付フォーマットが使用されます。

このファンクションは、データを検証してから表にロードする場合に使用できます。この方法により、値が表にロードされエラーの原因となる前に、値を変換できます。

例

WB_IS_DATE は、attribute に有効な日付が含まれる場合、PL/SQL の true を返します。

WB_JULIAN_FROM_DATE

構文

```
WB_JULIAN_FROM_DATE(attribute)
```

目的

WB_JULIAN_FROM_DATE は、attribute に指定される日付のユリウス暦の日付を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_JULIAN_FROM_DATE (sysdate) jdate
from dual;

     JDATE
----------
   2452362

select WB_JULIAN_FROM_DATE ('26-MAR-2002') jdate
from dual;

     JDATE
----------
   2452360
```

WB_MI_SS

構文

WB_MI_SS(attribute)

目的

WB_MI_SS は、attribute に指定される日付の時間部分の分と秒を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_MI_SS (sysdate) mi_ss
from dual;

MI_SS
-----------------------------------------
33:23

select WB_MI_SS ('26-MAR-2002') mi_ss
from dual;

MI_SS
-----------------------------------------
00:00
```

注意： 2 番目の例のように、タイムスタンプを含まない日付の場合、Oracle は 00:00:00 というタイムスタンプを使用するので、この場合はこのタイムスタンプが返されます。

WB_WEEK_OF_MONTH

構文

WB_WEEK_OF_MONTH(attribute)

目的

WB_WEEK_OF_MONTH は、attribute に指定される日付の、カレンダ月での週番号を返します。

例

次の例では、sysdate および指定した日付文字列に対する値が返されます。

```
select WB_WEEK_OF_MONTH (sysdate) w_of_m
from dual;

    W_OF_M
----------
         4

select WB_WEEK_OF_MONTH ('26-MAR-2002') w_of_m
from dual;

    W_OF_M
----------
         4
```

Numeric

ここでは、Numeric 変換をアルファベット順に説明します。Warehouse Builder で提供されるカスタム・ファンクションには、WB_ が接頭辞として付きます。

次の項では、Warehouse Builder で提供されるすべての Numeric 変換について、アルファベット順に説明します。

ABS

構文

```
abs::=ABS(attribute)
```

目的

ABS は、attribute の絶対値を返します。

例

次の例では、-15 の絶対値が返されます。

```
SELECT ABS(-15) "Absolute" FROM DUAL;
Absolute
----------
15
```

ACOS

構文

```
acos::= ACOS(attribute)
```

目的

ACOS は、attribute の逆余弦を返します。引数 attribute は、-1 から 1 の範囲内に指定する必要があり、このファンクションによって、ラジアン表記で 0 から π の範囲の値が返されます。

例

次の例では、.3 の逆余弦が返されます。

```
SELECT ACOS(.3) "Arc_Cosine" FROM DUAL;

Arc_Cosine
----------
1.26610367
```

ASIN

構文

```
asin::=ASIN(attribute)
```

目的

ASIN は、attribute の逆正弦を返します。引数 attribute は、-1 から 1 の範囲内に指定する必要があり、このファンクションによって、ラジアン表記で -π/2 から π/2 の範囲の値が返されます。

例
次の例では、.3 の逆正弦が返されます。

```
SELECT ACOS(.3) "Arc_Sine" FROM DUAL;

Arc_Sine
----------
.304692654
```

ATAN

構文
```
atan::=ATAN(attribute)
```

目的
ATAN は、attribute の逆正接を返します。引数 attribute に制限はなく、このファンクションによって、ラジアン表記で -π /2 から π /2 の範囲の値が返されます。

例
次の例では、.3 の逆正接が返されます。

```
SELECT ATAN(.3) "Arc_Tangent" FROM DUAL;

Arc_Tangent
----------
.291456794
```

ATAN2

構文
```
atan2::=ATAN2(attribute1, attribute2)
```

目的
ATAN2 は、attribute1 と attribute2 の逆正接を返します。引数 attribute1 に制限はなく、このファンクションによって、ラジアン表記で -π から π の値が返され、これは attribute1 と attribute2 の符号に依存します。ATAN2(attribute1,attribute2) は ATAN2(attribute1/attribute2) と同じです。

例
次の例では、.3 と .2 の逆正接が返されます。

```
SELECT ATAN2(.3,.2) "Arc_Tangent2" FROM DUAL;

Arc_Tangent2
------------
.982793723
```

COS

構文

```
cos::=COS(attribute)
```

目的

COS は、attribute（度数で表記された角度）の余弦を返します。

例

次の例では、180 度の余弦が返されます。

```
SELECT COS(180 * 3.14159265359/180) "Cosine" FROM DUAL;

Cosine
------
     -1
```

COSH

構文

```
cosh::=COSH(attribute)
```

目的

COSH は、attribute の双曲余弦を返します。

例

次の例では、0 の双曲余弦が返されます。

```
SELECT COSH(0) "Hyperbolic Cosine" FROM DUAL;

Hyperbolic Cosine
-----------------
                1
```

CEIL

構文

```
ceil::=CEIL(attribute)
```

目的

CEIL は、attribute 以上の最小整数値を返します。

例

次の例では、15.7 以上の最小整数値が返されます。

```
 SELECT CEIL(15.7) "Ceiling" FROM DUAL;
 Ceiling
----------
16
```

EXP

構文

```
exp::=EXP(attribute)
```

目的

EXP は、attribute で指定される値を n として、n 乗した自然対数の底を返します。自然対数の底の値は、2.71828183... です。

例

次の例では、4 乗した自然対数の底が返されます。

```
SELECT EXP(4) "e to the 4th power" FROM DUAL;

e to the 4th power
------------------
54.59815
```

FLOOR

構文

```
floor::=FLOOR(attribute)
```

目的

FLOOR は、attribute の数値以下の最大整数値を返します。

例

次の例では、15.7 以下の最大整数値が返されます。

```
SELECT FLOOR(15.7) "Floor" FROM DUAL;

Floor
----------
15
```

LN

構文

```
ln::=LN(attribute)
```

目的

LN は、attribute の自然対数を返します。ここで attribute は 0 より大きい値です。

例

次の例では、95 の自然対数が返されます。

```
SELECT LN(95) "Natural Logarithm" FROM DUAL;

Natural Logarithm
-----------------
4.55387689
```

LOG

構文

```
log::=LOG(attribute1, attribute2)
```

目的

LOG は、attribute1 を底とする attribute2 の対数を返します。底 attribute1 は 0 と 1 を除く任意の正の数で、attribute2 は任意の正の数です。

例

次の例では、100 の対数が返されます。

```
SELECT LOG(10,100) "Log base 10 of 100" FROM DUAL;

Log base 10 of 100
------------------
                 2
```

MOD

構文

```
mod::=MOD(attribute1, attribute2)
```

目的

MOD は、attribute1 を attribute2 で割った剰余を返します。attribute2 が 0 の場合は、attribute1 を返します。

例

次の例では、11 を 4 で割った剰余が返されます。

```
 SELECT MOD(11,4) "Modulus" FROM DUAL;
 Modulus
----------
3
```

POWER

構文

```
power::=POWER(attribute1, attribute2)
```

目的

POWER は、attribute2 で指定される値を n として、n 乗した attribute1 を返します。底 attribute1 と指数 attribute2 は任意の数値にできますが、attribute1 が負の場合は、attribute2 を整数にする必要があります。

例

次の例では、3 の 2 乗が返されます。

```
SELECT POWER(3,2) "Raised" FROM DUAL;
Raised
----------
9
```

ROUND（数値）

構文

```
round_number::=ROUND(attribute1, attribute2)
```

目的

ROUND は、attribute1 を、小数点以下第 attribute2 位に丸めて返します。attribute2 を省略すると、attribute1 は第 0 位に丸められます。attribute2 を負にすると、小数点の左側の数値が丸められます。attribute2 は整数にする必要があります。

例

次の例では、小数点第 1 位に数値が丸められます。

```
SELECT ROUND(15.193,1) "Round" FROM DUAL;

Round
----------
15.2
 The following example rounds a number one digit to the left of the decimal point:
 SELECT ROUND(15.193,-1) "Round" FROM DUAL;
 Round
----------
20
```

SIGN

構文

```
sign::=SIGN(attribute)
```

目的

SIGN は、attribute< 0 の場合、-1 を返します。attribute = 0 の場合は 0 を返し、attribute > 0 の場合は 1 を返します。これは、正の数値のみが期待されるメジャーの検証に使用できます。

例

次の例では、ファンクションの引数 -15 が 0 より小さいことがわかります。

```
SELECT SIGN(-15) "Sign" FROM DUAL;
 Sign
----------
-1
```

SIN

構文

```
sin::=SIN(attribute)
```

目的

SIN は、attribute（角度として表記）の正弦を返します。

例
次の例では、30 度の正弦が返されます。

```
SELECT SIN(30 * 3.14159265359/180) "Sine of 30 degrees" FROM DUAL;

Sine of 30 degrees
------------------
                .5
```

SINH

構文
```
sinh::=SINH(attribute)
```

目的
SINH は、attribute の双曲正弦を返します。

例
次の例では、1 の双曲正弦が返されます。

```
SELECT SINH(1) "Hyperbolic Sine of 1" FROM DUAL;

Hyperbolic Sine of 1
--------------------
          1.17520119
```

SQRT

構文
```
sqrt::=SQRT(attribute)
```

目的
SQRT は、attribute の平方根を返します。attribute の値を負にすることはできません。SQRT は、実数の結果を返します。

例
次の例では、26 の平方根が返されます。

```
SELECT SQRT(26) "Square root" FROM DUAL;

Square root
-----------
5.09901951
```

TAN

構文
```
tan::=TAN(attrin=bute)
```

目的
TAN は、attribute（ラジアンで表記された角度）の正接を返します。

例

次の例では、135 度の正接が返されます。

```
SELECT TAN(135 * 3.14159265359/180) "Tangent of 135 degrees" FROM DUAL;

Tangent of 135 degrees
----------------------
                    -1
```

TANH

構文

```
tanh::=TANH(attribute)
```

目的

TANH は、attribute の双曲正接を返します。

例

次の例では、5 の双曲正接が返されます。

```
SELECT TANH(5) "Hyperbolic tangent of 5" FROM DUAL;

Hyperbolic tangent of 5
-----------------------
             .462117157
```

TO_CHAR（数値）

構文

```
to_char_number::=to_char(attribute, fmt, nlsparam)
```

目的

TO_CHAR は、NUMBER データ型の attribute を、オプションの数値フォーマット fmt を使用して、VARCHAR2 データ型値に変換します。fmt を省略すると、attribute は、有効桁数と同じ長さの VARCHAR2 値に変換されます。nlsparam には、数値フォーマット要素により返される次の文字を指定します。

- 小数点の文字
- グループ・セパレータ
- ローカル通貨記号
- 国際通貨記号

この引数は、次の形式にできます。

```
'NLS_NUMERIC_CHARACTERS = ''dg''
 NLS_CURRENCY = ''text''
 NLS_ISO_CURRENCY = territory '
```

文字 d は小数点文字、g はグループ・セパレータを表します。これらは、異なるシングルバイト文字にする必要があります。引用した文字列内では、2 つの一重引用符でパラメータ値を囲む必要があります。通貨記号には、10 個の文字を使用できます。

nlsparam またはパラメータの 1 つを省略した場合、このファンクションは、セッションのデフォルト・パラメータ値を使用します。

例

次の例では、通貨記号の左に空白が埋め込まれて出力されます。

```
SELECT TO_CHAR(-10000,'L99G999D99MI') "Amount"
FROM DUAL;
Amount
--------------
$10,000.00-
SELECT TO_CHAR(-10000,'L99G999D99MI'
'NLS_NUMERIC_CHARACTERS = '',.''
NLS_CURRENCY = ''AusDollars'' ') "Amount"
FROM DUAL;

Amount
-------------------
AusDollars10.000,00-
```

TRUNC（数値）

構文

```
trunc_number::=TRUNC(attribute, m)
```

目的

TRUNC は、attribute を、小数点以下第 m 位に切り捨てて返します。m を省略すると、attribute は第 0 位に切り捨てられます。m を負にすると、小数点の左側の第 m 位までが切り捨てられます（0 になります）。

例

次の例では、数値が切り捨てられます。

```
SELECT TRUNC(15.79,1) "Truncate"
FROM DUAL;
Truncate
----------
15.7
 SELECT TRUNC(15.79,-1) "Truncate"
FROM DUAL;
 Truncate
----------
10
```

WB.LOOKUP_NUM（NUMBER 列用）

構文

```
 WB.LOOKUP_NUM (table_name
, column_name
, key_column_name
, key_value
)
```

TABLE_NAME は、参照を実行する表の名前です。COLUMN_NAME は、参照結果などで返される NUMBER 列の名前です。KEY_COLUMN_NAME は、参照表で照合するキーとして使用する NUMBER 列の名前です。KEY_VALUE は、照合を実行する key_column_name にマッピングされる値などのキー列値です。

目的

照合キーとして NUMBER 列を使用し、キー参照を実行して、データベース表から NUMBER 値を返します。

例

参照表 LKP1 として、次の表を使用するとします。

```
KEYCOLUMN  TYPE_NO  TYPE
10         100123   Car
20         100124   Bike
```

次のようなコールで、このパッケージを使用します。

```
WB.LOOKUP_CHAR('LKP1'
, 'TYPE_NO'
, 'KEYCOLUMN'
, 20
)
```

この場合は、この変換の出力として値 100124 が返されます。この出力は、インライン・ファンクション・コールの結果として、マッピングで処理されます。

注意： このファンクションは、行ベースのキー参照です。セット・ベースの参照は、参照演算子を使用した場合にサポートされます。

WB.LOOKUP_NUM（VARCHAR2 列用）

構文

```
WB.LOOKUP_CHAR(table_name
, column_name
, key_column_name
, key_value
)
```

TABLE_NAME は、参照を実行する表の名前です。COLUMN_NAME は、参照結果などで返される NUMBER 列の名前です。KEY_COLUMN_NAME は、参照表で照合するキーとして使用する VARCHAR2 列の名前です。KEY_VALUE は、照合を実行する key_column_name にマッピングされる値などのキー列値です。

目的

照合キーとして VARCHAR2 列を使用し、キー参照を実行して、データベース表から NUMBER 値を返します。

例

参照表 LKP1 として、次の表を使用するとします。

```
KEYCOLUMN  TYPE_NO  TYPE
ACV        100123   Car
ACP        100124   Bike
```

次のようなコールで、このパッケージを使用します。

```
WB.LOOKUP_CHAR ('LKP1'
, 'TYPE'
, 'KEYCOLUMN'
, 'ACP'
)
```

この場合は、この変換の出力として値 100124 が返されます。この出力は、インライン・ファンクション・コールの結果として、マッピングで処理されます。

注意： このファンクションは、行ベースのキー参照です。セット・ベースの参照は、2-7 ページの「キー参照」で説明する参照演算子を使用した場合にサポートされます。

WB_IS_NUMBER

構文

```
WB_IS_NUMBER(attibute, fmt)
```

目的

attribute に有効な数値が含まれているかどうかをチェックします。このファンクションで返されるブール値は、attribute に有効な数値が含まれている場合、true に設定されます。fmt は、オプションの数値フォーマットです。fmt を省略した場合は、セッションの数値フォーマットが使用されます。

このファンクションは、データを検証してから表にロードする場合に使用できます。この方法により、値が表にロードされエラーの原因となる前に、値を変換できます。

例

WB_IS_NUMBER は、attribute に有効な数値が含まれる場合、PL/SQL の true を返します。

OLAP

Warehouse Builder ユーザーは、OLAP 変換により、リレーショナル・ディメンションおよびリレーショナル・キューブに格納されたデータを、アナリティック・ワークスペースにロードできます。

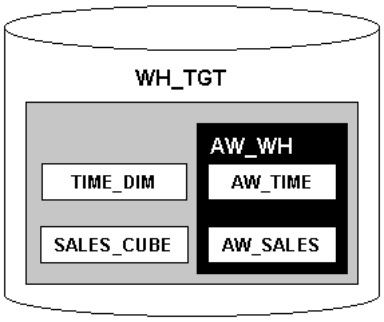
Warehouse Builder では、次の OLAP 変換を使用できます。

- WB_OLAP_LOAD_CUBE（3-54 ページ）
- WB_OLAP_LOAD_DIMENSION（3-54 ページ）
- WB_OLAP_LOAD_DIMENSION_GENUK（3-55 ページ）

WB_OLAP_LOAD_CUBE、WB_OLAP_LOAD_DIMENSION、WB_OLAP_LOAD_DIMENSION_GENUK の各変換は、Warehouse Builder でキューブのクローニングに使用します。これらの OLAP 変換は、データベースのバージョンが Oracle Database 9*i* または Oracle Database 10*g* リリース 1 の場合にのみ使用します。

これらの OLAP 変換の説明に使用する例は、図 3-1 に示すシナリオに基づいています。

図 3-1 OLAP 変換の例



リレーショナル・ディメンション TIME_DIM とリレーショナル・キューブ SALES_CUBE は、スキーマ WH_TGT に格納されています。WH_TGT スキーマには、ディメンションとキューブがロードされたアナリティック・ワークスペース AW_WH も作成されます。

WB_OLAP_LOAD_CUBE

構文

```
wb_olap_load_cube::=WB_OLAP_LOAD_CUBE(olap_aw_owner, olap_aw_name, olap_cube_owner,
olap_cube_name, olap_tgt_cube_name)
```

olap_aw_owner は、アナリティック・ワークスペースを保有するデータベース・スキーマの名前です。olap_aw_name は、キューブのデータを格納するアナリティック・ワークスペースの名前です。olap_cube_owner は、関連するリレーショナル・キューブを保有するデータベース・スキーマの名前です。olap_cube_name は、リレーショナル・キューブの名前です。olap_tgt_cube_name は、アナリティック・ワークスペースに入っているキューブの名前です。

目的

WB_OLAP_LOAD_CUBE は、リレーショナル・キューブからアナリティック・ワークスペースにデータをロードします。これにより、キューブのデータを詳細に分析できます。これは、クローニング元のリレーショナル・キューブからアナリティック・ワークスペースに、キューブのデータをロードするためのものです。これは、キューブをロードするための DBMS_AWM パッケージのいくつかのプロシージャのラッパーです。

例

次の例では、リレーショナル・キューブ SALES_CUBE からアナリティック・ワークスペース AW_WH のキューブ AW_SALES に、データをロードします。

```
WB_OLAP_LOAD_CUBE('WH_TGT', 'AW_WH', 'WH_TGT', 'SALES_CUBE', 'AW_SALES')
```

WB_OLAP_LOAD_DIMENSION

構文

```
wb_olap_load_dimension::=WB_OLAP_LOAD_DIMENSION(olap_aw_owner, olap_aw_name, olap_
dimension_owner, olap_dimension_name, olap_tgt_dimension_name)
```

olap_aw_owner は、アナリティック・ワークスペースを保有するデータベース・スキーマの名前です。olap_aw_name は、ディメンションのデータを格納するアナリティック・ワークスペースの名前です。olap_dimension_owner は、関連するリレーショナル・ディメンションが格納されるデータベース・スキーマの名前です。olap_dimension_name は、リレーショナル・ディメンションの名前です。olap_tgt_dimension_name は、アナリティック・ワークスペースに入っているディメンションの名前です。

目的

WB_OLAP_LOAD_DIMENSION は、リレーショナル・ディメンションからアナリティック・ワークスペースにデータをロードします。これにより、ディメンションのデータを詳細に分析できます。これは、クローニング元のリレーショナル・ディメンションからアナリティック・ワークスペースに、ディメンションのデータをロードするためのものです。これは、ディメンションをロードするための DBMS_AWM パッケージのいくつかのプロシージャのラッパーです。

例

次の例では、リレーショナル・ディメンション TIME_DIM からアナリティック・ワークスペース AW_WH のディメンション AW_TIME に、データをロードします。

```
WB_OLAP_LOAD_DIMENSION('WH_TGT', 'AW_WH', 'WH_TGT', 'TIME_DIM', 'AW_TIME')
```

WB_OLAP_LOAD_DIMENSION_GENUK

構文

```
wb_olap_load_dimension_genuk::=WB_OLAP_LOAD_DIMENSION_GENUK(olap_aw_owner, olap_aw_
name, olap_dimension_owner, olap_dimension_name, olap_tgt_dimension_name)
```

olap_aw_owner は、アナリティック・ワークスペースを保有するデータベース・スキーマの名前です。olap_aw_name は、ディメンションのデータを格納するアナリティック・ワークスペースの名前です。olap_dimension_owner は、関連するリレーショナル・ディメンションが格納されるデータベース・スキーマの名前です。olap_dimension_name は、リレーショナル・ディメンションの名前です。olap_tgt_dimension_name は、アナリティック・ワークスペースに入っているディメンションの名前です。

目的

WB_OLAP_LOAD_DIMENSION_GENUK は、リレーショナル・ディメンションからアナリティック・ワークスペースにデータをロードします。これにより、すべてのレベルで一意のディメンション識別子が生成されます。これは、クローニング元のリレーショナル・ディメンションからアナリティック・ワークスペースに、ディメンションのデータをロードするためのものです。これは、ディメンションをロードするための DBMS_AWM パッケージのいくつかのプロシージャのラッパーです。

キューブがクローニングされた場合、「ディメンションのサロゲート・キーを生成」オプションで「はい」を選択すると、ディメンションを再ロードするときに、WB_OLAP_LOAD_DIMENSION_GENUK プロシージャを使用する必要があります。このプロシージャでは、アナリティック・ワークスペースのすべてのレベルでサロゲート識別子が生成されます。これは、アナリティック・ワークスペースでは、すべてのレベルの識別子が、すべてのレベルのディメンションで一意である必要があるためです。

例

キューブのクローニングによって、ディメンション TIME_DIM が OLAP サーバーに配布されている場合を考えます。パラメータ generate surrogate keys for Dimension が true に設定されているとします。ここで、リレーショナル・ディメンション TIME_DIM からアナリティック・ワークスペース AW_WH のディメンション AW_TIME にデータを再ロードするには、次の構文を使用します。

```
WB_OLAP_LOAD_CUBE('WH_TGT', 'AW_WH', 'WH_TGT', 'TIME_DIM', 'AW_TIME')
```

XML

Warehouse Builder ユーザーは、XML 変換により、XML オブジェクトで変換を実行できます。Warehouse Builder ユーザーは、この変換により、XML 文書とアドバンスト・キューをロードおよび変換できます。

XML ソースのロードを可能にするため、Warehouse Builder では、この章で詳しく説明されているカスタム・ファンクションを使用して、データベースの XML 機能にアクセスします。

WB_XML_LOAD

構文

```
WB_XML_LOAD(control_file)
```

目的

WB_XML_LOAD は、XML 文書からデータを抽出し、そのデータをデータベース・ターゲットにロードします。control_file（XML 文書）には、XML 文書のソース、ターゲットおよびランタイム制御を指定します。変換を定義した後、Warehouse Builder でのマッピングでは、マッピング前トリガーまたはマッピング後トリガーとしてこの変換がコールされます。

例

次の例のスクリプトは、ファイル products.xml に保存されている XML 文書からデータを抽出し、そのデータをターゲット表 books にロードする Warehouse Builder 変換を実装します。

```
begin
wb_xml_load('<OWBXMLRuntime>'
||
'<XMLSource>'
||
' <file>¥ora817¥GCCAPPS¥products.xml</file>'
||
'</XMLSource>'
||
'<targets>'
||
' <target XSLFile="¥ora817¥XMLstyle¥GCC.xsl">books</target>'
||
'</targets>'
||
'</OWBXMLRuntime>'
);
end;
```

制御ファイルの詳細は、『Oracle Warehouse Builder ユーザーズ・ガイド』を参照してください。

WB_XML_LOAD_F

構文

```
WB_XML_LOAD_F(control_file)
```

目的

WB_XML_LOAD_F は、XML 文書からデータを抽出し、そのデータをデータベース・ターゲットにロードします。このファンクションは、ロード中に読み込まれた XML 文書の数を返します。control_file（XML 文書）には、XML 文書のソース、ターゲットおよびランタイム制御を指定します。変換を定義した後、Warehouse Builder でのマッピングでは、マッピング前トリガーまたはマッピング後トリガーとしてこの変換がコールされます。

例

次の例のスクリプトは、ファイル products.xml に保存されている XML 文書からデータを抽出し、そのデータをターゲット表 books にロードする Warehouse Builder 変換を実装します。

```
begin
wb_xml_load_f('<OWBXMLRuntime>'
||
'<XMLSource>'
||
' <file>¥ora817¥GCCAPPS¥products.xml</file>'
||
'</XMLSource>'
||
'<targets>'
||
' <target XSLFile="¥ora817¥XMLstyle¥GCC.xsl">books</target>'
||
'</targets>'
||
'</OWBXMLRuntime>'
);
end;
```

処理される型および control_file の詳細は、『Oracle Warehouse Builder ユーザーズ・ガイド』を参照してください。

Conversion

Warehouse Builder ユーザーは、Conversion 変換により、値の条件付き変換を実行するファンクションを実行できます。このファンクションでは、SQL 内で if - then 構造が実行されます。たとえば、NVL では、指定された任意の値で NULL 値を置き換えたり、入力が NULL の場合に value を出力する機能が提供されます。

CASE

CASE 式では、プロシージャを起動せずに、SQL 文で IF...THEN...ELSE ロジックを使用できます。decode の代わりに、この文を使用してください。

構文

```
case_expression::=CASE attrubute1 WHEN inputvalue THEN outputvalue
[WHEN inputvalue THEN outputvalue]...
ELSE elsevalue
END
```

目的

単純な CASE 式では、attribute1 が inputvalue に等しい、最初の WHEN ... THEN ペアが検索され、outputvalue が返されます。すべての WHEN ... THEN ペアがこの条件を満たさず、ELSE 句が存在する場合は、elsevalue が返されます。それ以外の場合は、null が返されます。

すべての式（attribute1、inputvalue および outputvalue）は、同じデータ型にする必要があり、CHAR または VARCHAR2 を使用できます。

単純な CASE の例

次の文では、サンプル表 oe.customers の顧客ごとに、そのクレジット制限が $100 に等しい場合は「Low」、$5000 に等しい場合は「High」、それ以外の場合は「Medium」と一覧表示されます。

```
SELECT cust_last_name,
   CASE credit_limit WHEN 100 THEN 'Low'
   WHEN 5000 THEN 'High'
   ELSE 'Medium' END
   FROM customers;

CUST_LAST_NAME       CASECR
-------------------- ------
...
Bogart               Medium
Nolte                Medium
Loren                Medium
Gueney               Medium
```

検索 CASE の例

次の文では、最低給与として $2000 を使用して、サンプル表 oe.employees で従業員の平均給与が検索されます。

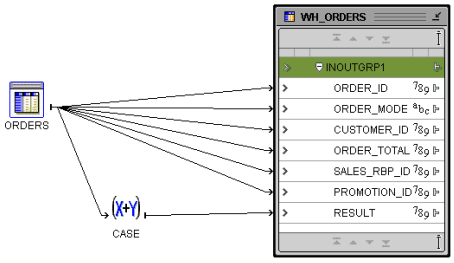
```
SELECT AVG(CASE WHEN e.salary > 2000 THEN e.salary
   ELSE 2000 END) "Average Salary" from employees e;

Average Salary
--------------
    6461.68224
```

Warehouse Builder での例

Warehouse Builder では、式を使用して CASE 文を保持できます。Expression Builder では、この文を作成して生成コードに組み込むことができます。この例を、図 3-2「CASE マッピングの例」に示します。

図 3-2 CASE マッピングの例



NVL

構文

```
nvl::=NVL(attribute1, attrbitute2)
```

目的

attribute1 が NULL の場合、NVL は attribute2 を返します。attribute1 が NULL でない場合、NVL は attribute1 を返します。引数 attribute1 と attribute2 は、任意のデータ型にできます。データ型が異なる場合は、attribute2 が attribute1 のデータ型に変換されてから比較されます。Warehouse Builder では、NVL の 3 つのバリエーションが提供され、すべての入力値がサポートされます。

attribute1 が文字データでないかぎり、戻り値のデータ型は、attribute1 と常に同じデータ型になります。文字データの場合は、戻り値のデータ型は、attribute1 のキャラクタ・セットの VARCHAR2 になります。

例

次の例では、従業員名とコミッションのリストが返されますが、従業員がコミッションを受け取っていない場合、コミッションは「Not Applicable」になります。

```
SELECT last_name, NVL(TO_CHAR(commission_pct), 'Not Applicable') "COMMISSION"
FROM employees
WHERE last_name LIKE 'B%';

LAST_NAME                 COMMISSION
------------------------- ----------------------------------------
Baer                      Not Applicable
Baida                     Not Applicable
Banda                     .11
Bates                     .16
Bell                      Not Applicable
Bernstein                 .26
Bissot                    Not Applicable
Bloom                     .21
Bull                      Not Applicable
```

Other

Warehouse Builder には、特定のデータ型に制限されない様々なファンクションを実行できる変換も組み込まれています。この項では、このようなタイプの変換について説明します。

NLS_CHARSET_DECL_LEN

構文

```
nls_charset_decl_len::=NLS_CHARSET_DECL_LEN(byte_count,charset_id)
```

目的

NLS_CHARSET_DECL_LEN は、NCHAR 列の宣言幅（文字数）を返します。引数 byte_count は列の幅で、引数 charset_id は列のキャラクタ・セット ID です。

例

次の例では、マルチバイト・キャラクタ・セットを使用するとき、200 バイトの列にある文字数を返します。

```
SELECT NLS_CHARSET_DECL_LEN(200, nls_charset_id('ja16eucfixed')) FROM DUAL;

NLS_CHARSET_DECL_LEN(200,NLS_CHARSET_ID('JA16EUCFIXED'))
---------------------------------------------------------
                                                      100
```

NLS_CHARSET_ID

構文

```
nls_charset_id::= NLS_CHARSET_ID(text)
```

目的

NLS_CHARSET_ID は、キャラクタ・セット名 text に対応するキャラクタ・セット ID 番号を返します。引数 text は、実行時の VARCHAR2 値です。text 値に 'CHAR_CS' を指定すると、サーバーのデータベース・キャラクタ・セット ID 番号が返されます。text 値に 'NCHAR_CS' を指定すると、サーバーの各国語キャラクタ・セット ID 番号が返されます。

無効なキャラクタ・セット名を指定すると、NULL が返されます。

例

次の例では、キャラクタ・セットのキャラクタ・セット ID 番号が返されます。

```
SELECT NLS_CHARSET_ID('ja16euc') FROM DUAL;

NLS_CHARSET_ID('JA16EUC')
--------------------------
                       830
```

NLS_CHARSET_NAME

構文

```
nls_charset_name::= NLS_CHARSET_NAME(number)
```

目的

NLS_CHARSET_NAME は、ID number に対応するキャラクタ・セットの名前を返します。キャラクタ・セット名は、データベース・キャラクタ・セットで VARCHAR2 値として返されます。

number が有効なキャラクタ・セット ID として認識されない場合、このファンクションは NULL を返します。

例

次の例では、キャラクタ・セット ID 番号 2 に対応するキャラクタ・セットが返されます。

```
SELECT NLS_CHARSET_NAME(2) FROM DUAL;

NLS_CH
--------
WE8DEC
```

UID

構文

```
uid::=UID()
```

目的

UID は、変換を含むセッションの実行時にログオンするユーザーなど、セッション・ユーザーを一意に識別する整数を返します。分散型の SQL 文の場合、UID ファンクションは、ローカル・データベース上のユーザーを識別します。

監査情報のログをターゲット表に記録し、マッピングを実行しているユーザーを識別する場合、このファンクションを使用します。

例

次の例では、このセッションにログインしたユーザーのローカル・データベース・ユーザー ID が返されます。

```
select uid from dual;

       UID
----------
        55
```

USER

構文

```
user::=USER()
```

目的

USER は、データ型が VARCHAR2 のセッション・ユーザー名（ログオン中のユーザー）を返します。

Oracle は、空白を埋め込んだ比較セマンティクスとこのファンクションの値を比較します。分散型の SQL 文の場合、UID ファンクションと USER ファンクションは、ローカル・データベース上のユーザーを識別します。

監査情報のログをターゲット表に記録し、マッピングを実行しているユーザーを識別する場合、このファンクションを使用します。

例

次の例では、このセッションにログイン中のローカル・データベース・ユーザーが返されます。

```
select user from dual;

USER
------------------------------
OWB9I_RUN
```

A

緩やかに変化する次元の使用

この付録では、緩やかに変化する次元の様々なタイプについて説明します。また、ケース・スタディ・シナリオを通じて、Warehouse Builder を使用して緩やかに変化する次元の様々なタイプを設計および配布する方法について説明します。詳細は、Ralph Kimball 氏の『データウェアハウス・ツールキット』など、データ・ウェアハウスに関する解説書を参照してください。

注意： Warehouse Builder では、Oracle9*i* Database リリース 2 以降のデータベース・サーバーでのみ、緩やかに変化する次元の使用がサポートされます。

この付録では、次のトピックについて説明します。

緩やかに変化する次元について

緩やかに変化する次元（Slowly Changing Dimension）は、現行データと履歴データの両方をデータ・ウェアハウスで時間の経過に従って管理する、適切に定義された戦略です。最初に、ビジネス要件に基づいて、使用する緩やかに変化する次元のタイプを決める必要があります。緩やかに変化する次元の 3 つの主なタイプについて、表 A-1 で説明します。

表 A-1 緩やかに変化する次元のタイプ

タイプ	用途	説明	履歴データの保持
タイプ 1	上書き	1 バージョンのディメンション・レコードのみが存在します。変更時に、レコードは上書きされ、履歴データは保存されません。	いいえ
タイプ 2	別のディメンション・レコードの作成	同じディメンション・レコードの複数のバージョンが存在し、修正時に新バージョンが作成されても、旧バージョンは保持されます。	はい
タイプ 3	現行値フィールドの作成	同じディメンション・レコードで、古い値と現行の値の 2 つのバージョンが存在し、現行値が修正されても古い値は保持されます。	はい

緩やかに変化する次元のタイプを選択したら、次の手順を実行してディメンションを作成します。

- 履歴データを保存する新しいディメンションを作成します。
- ソース・システムから事前定義済のディメンション・ターゲットに、データを抽出、変換およびロードするマッピングを作成します。
- ディメンションとマッピングの両方を生成し、Oracle9*i* Database リリース 2 以降のデータベースに配布します。
- マッピングを実行します。

ケース・スタディ・シナリオ

緩やかに変化する次元を使用するには、次を使用している必要があります。

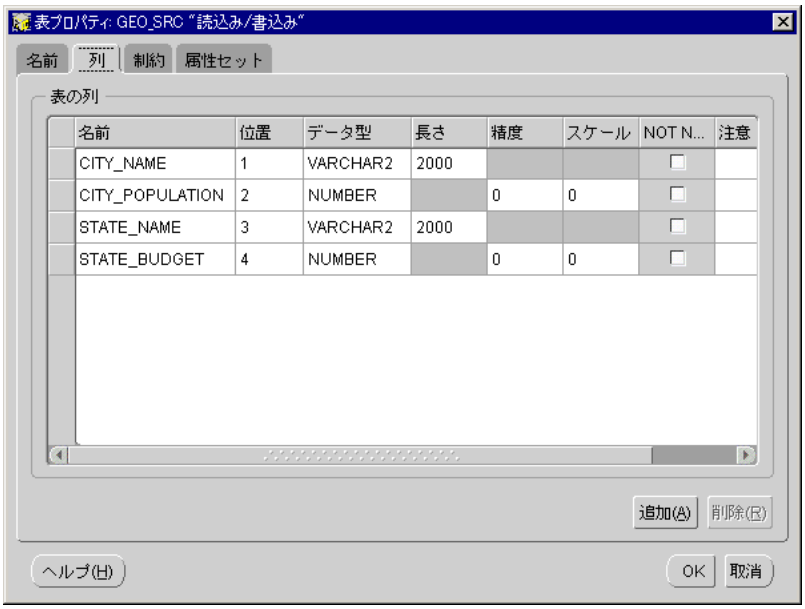
- Warehouse Builder 9.0.4 以降
- Oracle9*i* Database リリース 2 以降のデータベース

この付録では、次の項で説明するソース・システムとターゲット・システムを使用して、緩やかに変化する次元を構成する方法について説明します。スター・スキーマを使用して、同一ディメンション表ターゲットに、すべてのレベルのデータを保存します。これは、最も一般的に使用される戦略の 1 つです。

ソース・システム

このケース・スタディでは、地理データ・ソース表 GEO_SRC を使用します。図 A-1 に、GEO_SRC 表の属性を示します。

図 A-1 GEO_SRC 表のプロパティ



ターゲット・システム

作成するターゲット・ウェアハウスには、次が含まれます。

- サロゲート・キーの移入に使用する順序 DIM_ID
- タイプ 1 の緩やかに変化する次元として使用するディメンション GEO_DIM
- GEO_SRC から GEO_DIM にデータをロードするマッピング
- タイプ 2 の緩やかに変化する次元として使用するディメンション GEO_DIM_TYPE2
- GEO_SRC から GEO_DIM にデータをロードするマッピング
- タイプ 3 の緩やかに変化する次元として使用するディメンション GEO_DIM_TYPE3
- GEO_SRC から GEO_DIM_TYPE3 にデータをロードするマッピング

このケース・スタディでは、地理ディメンションを例として使用します。一般的に、地理ディメンションには、市区町村と州（都道府県）という、2 つのレベルがあります。市区町村レベルは地理階層の最下位レベルであり、州（都道府県）レベルは上位レベルです。図 A-2 に示す簡単な市区町村レベルには、次の属性が含まれます。

- **ID:** 市区町村レベルのサロゲート・キー
- **NAME:** 市区町村レベルの自然キー
- **POPULATION:** 市区町村の人口

図 A-2 市区町村レベル・ディメンションのプロパティ

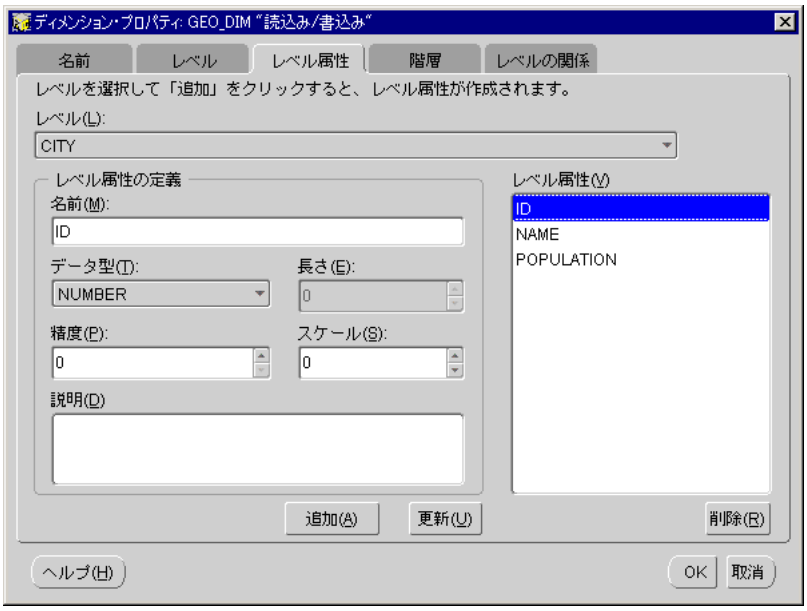
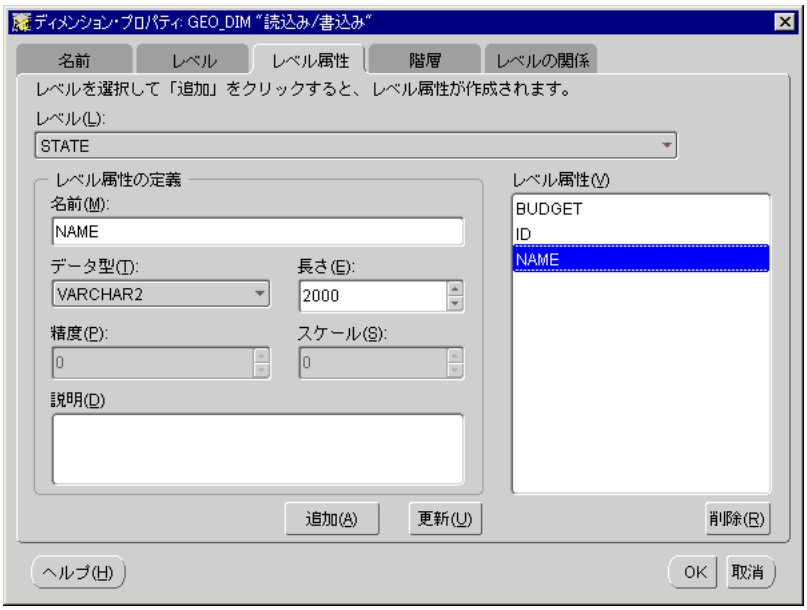


図 A-3 に示す簡単な州（都道府県）レベルには、次の属性が含まれます。

- **ID:** 州（都道府県）レベルのサロゲート・キー
- **NAME:** 州（都道府県）レベルの自然キー
- **BUDGET:** 州（都道府県）の予算

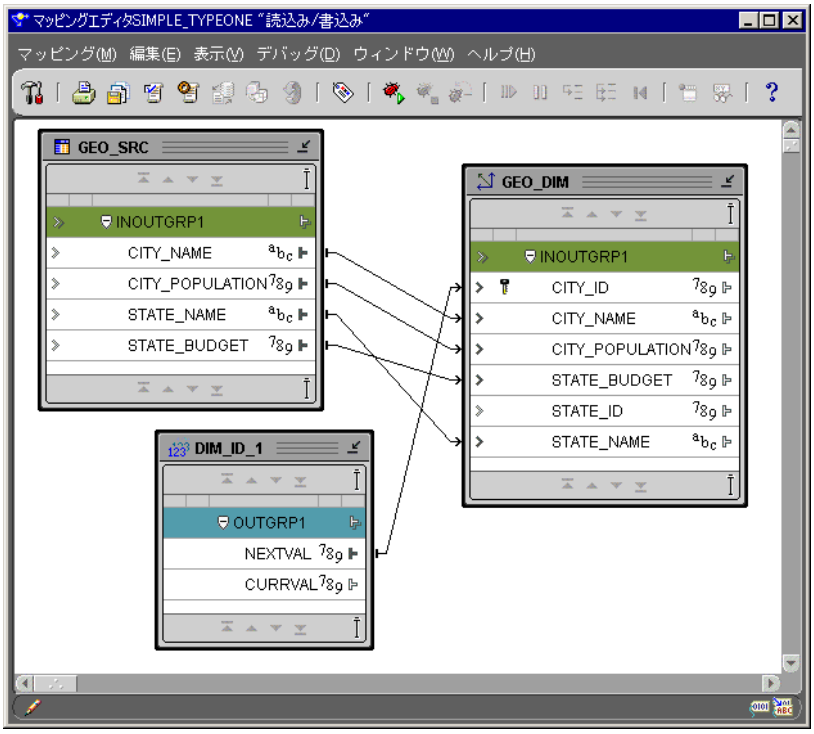
図 A-3 州（都道府県）レベル・ディメンションのプロパティ



タイプ 1 の緩やかに変化する次元の使用

タイプ 1 の緩やかに変化する次元では、履歴が保持されず、ディメンション・レコードの最新値のみが保存されます。ディメンション GEO_DIM を定義したら、マッピングに使用してデータをロードできます。タイプ 1 の緩やかに変化する次元にロードするには、ソースからデータを抽出し、それをターゲットに直接ロードします。GEO_SRC はソース表で、ここからディメンション DEO_DIM にデータをロードします。図 A-4 は、この例で使用するマッピングです。

図 A-4 タイプ 1 の緩やかに変化する次元のマッピング



タイプ 1 の緩やかに変化する次元を作成するには、次のステップを完了する必要があります。

- ステップ 1: サロゲート・キーの移入
- ステップ 2: ターゲット・プロパティの構成
- ステップ 3: コードの生成

ステップ 1: サロゲート・キーの移入

新しいディメンション・レコードのサロゲート・キーに一意の数値を確実に割り当てるには、順序演算子を使用して、GEO_DIM のサロゲート・キー列 CITY_ID（最下位レベルのキー）にマッピングします。

ステップ 2: ターゲット・プロパティの構成

図 A-5 のように、GEO_DIM 演算子のプロパティを構成し、データを正しくロードできるようにします。最初に、GEO_DIM のロード・タイプを「UPDATE/INSERT」に構成する必要があります。

図 A-5 ディメンションのマッピングのプロパティ

また、マッピングした各列を構成する必要があります。

- CITY_ID はサロゲート・キーで、行の挿入時にのみロードします。
- CITY_NAME は自然キーで、行の挿入時にのみロードします。行の更新時は照合します。
- CITY_POPULATION は、行の挿入時と更新時の両方でロードします。STATE_NAME と STATE_BUDGET は、同様に構成します。

ステップ 3: コードの生成

ウェアハウス・モジュールの構成プロパティから構成できるターゲット・データベース・タイプが Oracle9*i* Database に設定されている場合は、コード生成時に Merge 機能が保証されます。

タイプ 2 の緩やかに変化する次元の使用

タイプ 2 の緩やかに変化する次元では、毎回、ディメンション・レコードの別バージョンを作成し、既存バージョンを履歴としてマークします。これに対処するには、有効日付列や失効日付列などの追加メタデータをディメンション表に作成する必要があります。これらの列を使用して、現行バージョンと履歴バージョンを次のように区別します。

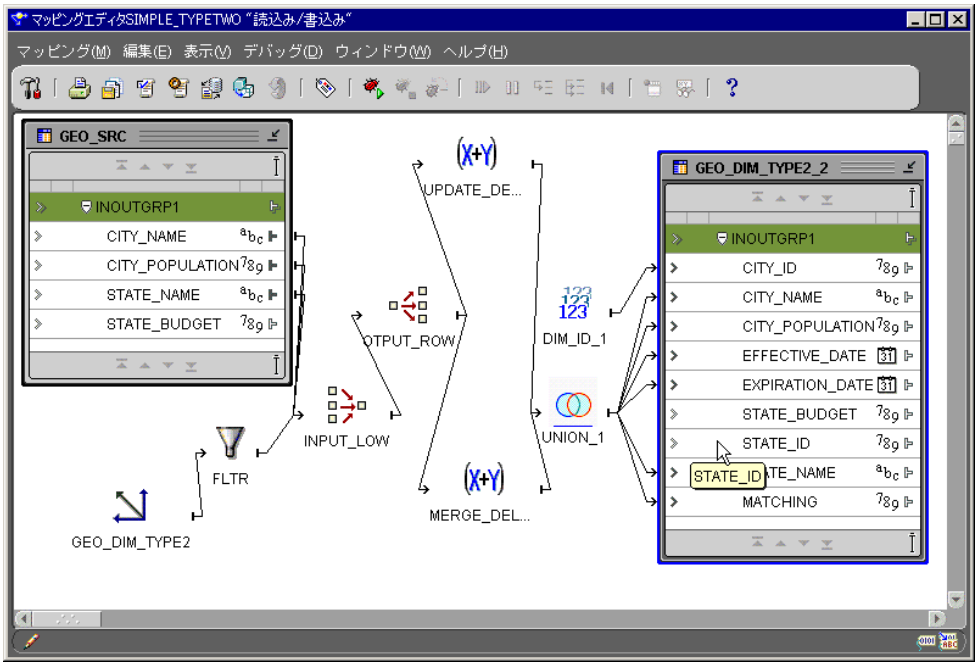
- 有効日付列には、バージョンの有効日付を格納します。これは、開始日付とも呼ばれます。
- 失効日付列には、バージョンの失効日付を格納します。これは、終了日付とも呼ばれます。
- 現行バージョンの失効日付値は、常に NULL またはデフォルト日付値に設定します。

値の変更時に、履歴データを保存する列を決める必要もあります。この列はトリガー列として定義し、メタデータの要素として表す必要があります。

ディメンション GEO_DIM_TYPE2 を定義したら、マッピングに使用してデータをロードできます。GEO_SRC はサンプル・ソース表で、ここから GEO_DIM_TYPE2 にデータをロードします。

タイプ 2 の緩やかに変化する次元にロードするには、ソースから抽出したデータを適切に変換してから、ターゲットにロードする必要があります。これを行うには、図 A-6 のようなマッピングを作成します。このマッピングでは、最初にデータが GEO_SRC から抽出され、一連の演算子で変換されてから、最終的に GEO_DIM_TYPE2 にロードされます。

図 A-6 タイプ 2 の緩やかに変化する次元のマッピング



データの実際の変換方法について、特に注目してください。Warehouse Builder では、タイプ 2 の緩やかに変化する次元で必要となる、すべての演算子がサポートされています。Warehouse Builder を使用すると、タイプ 2 の緩やかに変化する次元の ETL プロセス全体を 1 つのマッピングで実行できます。データの変換方法については、ステップごとに説明します。

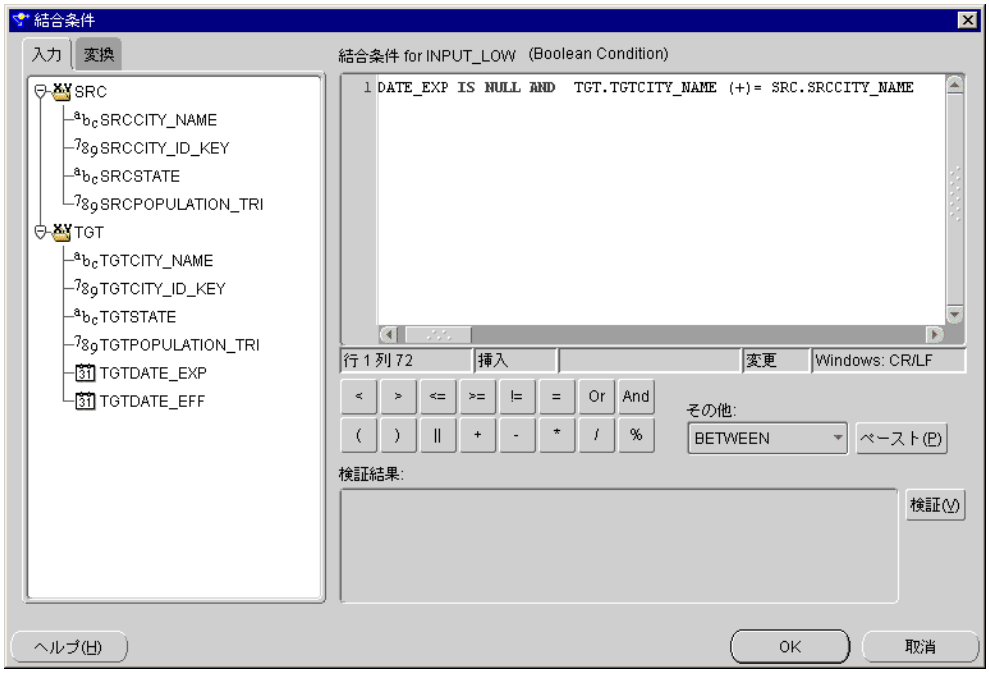
タイプ 2 の緩やかに変化する次元を作成するには、次のステップを完了する必要があります。

- ステップ 1: 一致の検出
- ステップ 2: 結合結果の分割
- ステップ 3: マージ行の識別
- ステップ 4: 式 UPDATE_DELTA_ROW の使用
- ステップ 5: 式 MERGE_DELTA_ROW の使用
- ステップ 6: サロゲート・キーの移入
- ステップ 7: ターゲット・プロパティの構成
- ステップ 8: コードの生成

ステップ 1: 一致の検出

最初に、GEO_SRC のソース行ごとに、それが GEO_DIM_TYPE2 の現行ディメンション・レコードに一致するかどうかを調べる必要があります。これを行うには、結合子を使用して、自然キー列を結合条件にした外部結合を行い GEO_SRC と GEO_DIM_TYPE2 を排他的に照合します。図 A-7 は、この条件に使用する式です。

図 A-7 Input_Row 式



GEO_SRC の照合は、履歴のディメンション・レコードではなく、GEO_DIM_TYPE2 の現行のディメンション・レコードにのみ行う必要があることに注意してください。このためには、フィルタ演算子を適用して、履歴レコードを照合から除外します。

ステップ 2: 結合結果の分割

結合後の出力データには、ソース・データ行と照合したターゲット行の両方が含まれます。結合子の各出力行は、次のグループに分類する必要があります。

- 新バージョンを作成するか、現行バージョンを上書きする OPEN_SET
- 現行バージョンを履歴としてマークする CLOSE_SET

このように分類するには、スプリッタを使用して、結合子出力を OPEN_SET グループと CLOSE_SET グループに分割します。

結合子出力行が GEO_DIM_TYPE2 の任意の現行バージョンと一致する GEO_SRC 行またはどのバージョンとも一致しない GEO_SRC 行のどちらかから取得されている場合は、OPEN_SET グループに分類します。OPEN_SET グループのスプリッタ条件を指定して、これを実行します。

次の 2 つの条件の両方が満たされる場合、結合子出力行は CLOSE_SET グループに分類します。

- GEO_DIM_TYPE2 の任意の現行バージョンと一致する GEO_SRC 行から取得されている。
- GEO_DIM_TYPE2 のどのトリガー列も GEO_SRC のトリガー列と等しくない。

CLOSE_SET グループのスプリッタ条件およびこの 2 つの条件句を指定します。

ステップ 3: マージ行の識別

OPEN_SET と CLOSE_SET を使用して、GEO_DIM_TYPE2 にロードする、次の 2 つのデルタ・セットを計算します。

- CLOSE_SET からロードし、GEO_DIM_TYPE2 を更新します。これは、UPDATE_DELTA_ROW とも呼ばれます。
- OPEN_SET からロードし、GEO_DIM_TYPE2 の更新と挿入を行います。これは、MERGE_DELTA_ROW とも呼ばれます。

両方のタスクを実行するには、式を使用します。UPDATE_DELTA_ROW と MERGE_DELTA_ROW は、それぞれ CLOSE_SET と OPEN_SET の出力から、2 つの異なる式演算子として作成されます。両方の式演算子の出力グループは、集合演算子を活用することで UNION になり、この出力行セットは、GEO_DIM_TYPE2 に直接マッピングする準備が整っています。

ステップ 4: 式 UPDATE_DELTA_ROW の使用

UPDATE_DELTA_ROW は、一致する現行バージョンを履歴としてマークするために、最終ターゲット行が上書きされる行セットを表します。具体的には、ターゲットの失効タイムスタンプを現在のシステム日付の値で更新する必要があります。この操作は、現行バージョンを閉じることにもなります。これを実行するには、属性 DATE_EXP の式を SYSDATE に指定します。

残りの列は、更新の必要がありません。元のターゲット列の値が、対応する式に指定されます。

ステップ 5: 式 MERGE_DELTA_ROW の使用

MERGE_DELTA_ROW は、行セットを表します。最終ターゲット行を上書きし、次を行います。

- 現行バージョンがターゲットで一致しないか、一致するバージョンに任意のトリガー列とは異なる値が含まれる場合は、別の現行バージョンを作成します。
- その他の場合、一致バージョンを直接更新します。

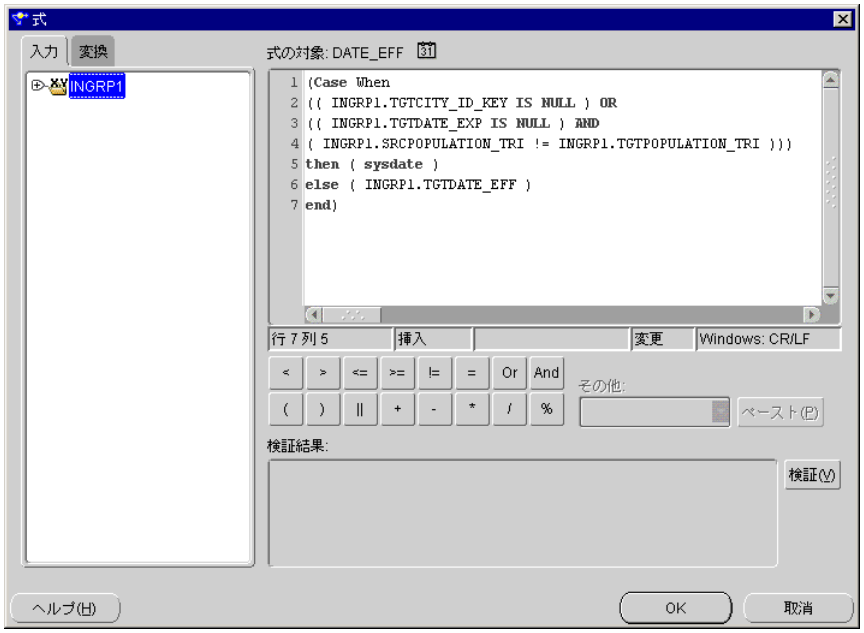
具体的には、最終ターゲット列ごとに式を構築し、Case When (...) Then (...) Else (...) End のような CASE 式をインスタンス化して、上の 2 つのシナリオを区別する必要があります。Warehouse Builder には、使いやすい Expression Builder がサポートされており、これを簡単に実行できます。

DATE_EFF または任意の有効タイムスタンプ列には、次を実行する式を指定します。

- 別のバージョンを作成する場合、現在のシステム時刻（SYSDATE）を維持します。
- その他の場合、ターゲットから派生した有効タイムスタンプ値を維持します（一致バージョンの更新）。

図 A-8 は、DATE_EFF または任意の有効タイムスタンプ列の式を指定する方法の例です。

図 A-8 DATE_EFF の式

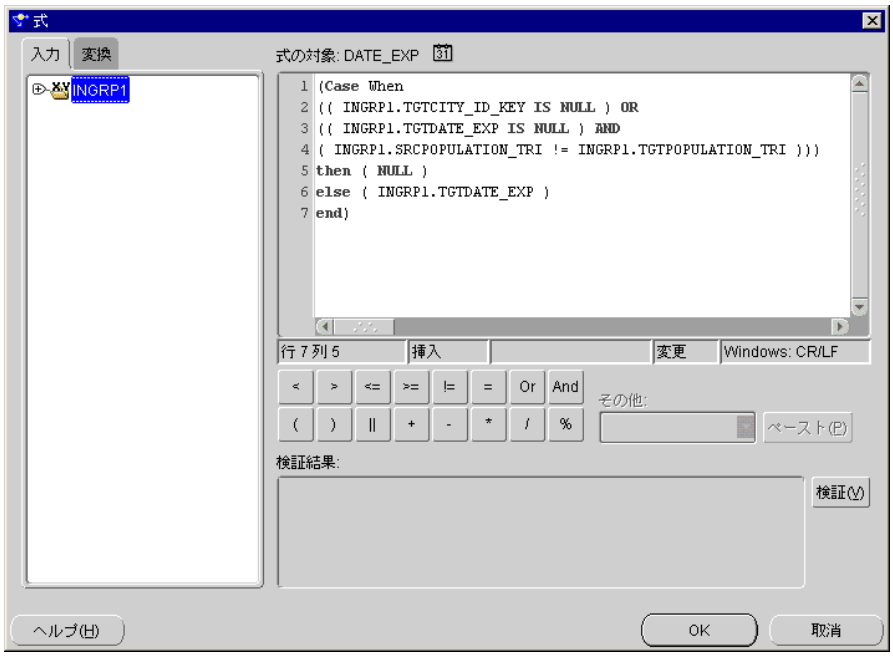


DATE_EXP または任意の失効タイムスタンプ列には、次を実行する式を指定します。

- 別のバージョンを作成する場合、デフォルト値（NULL、または 01/01/2004 などの将来のタイムスタンプ）を保持し、任意のバージョンを現行としてマークします。
- その他の場合、ターゲットから派生した失効タイムスタンプ値を維持します（一致バージョンの更新）。

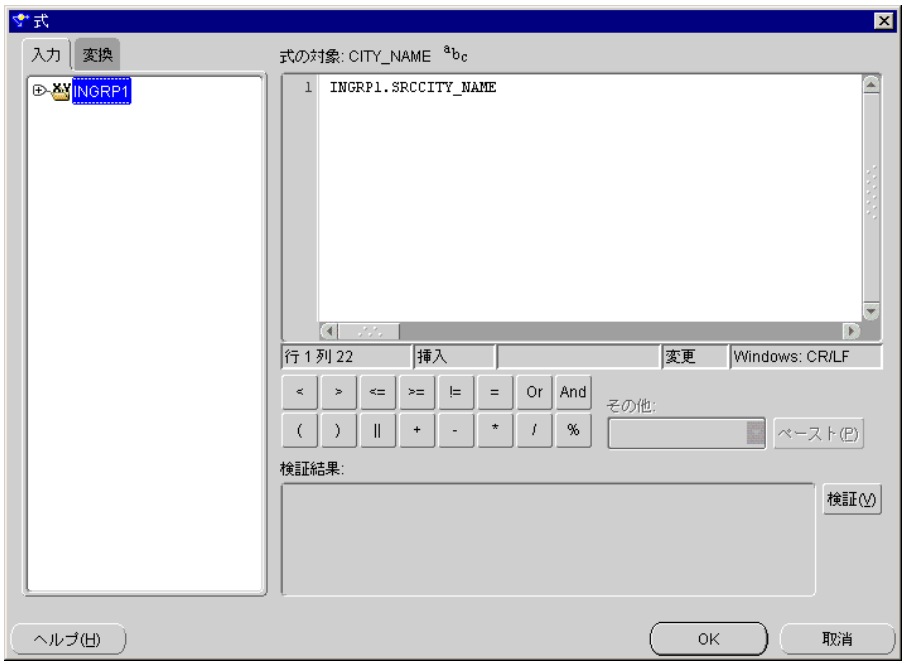
図 A-9 は、DATE_EXP または任意の失効タイムスタンプ列の式を指定する方法の例です。

図 A-9 DATE_EXP の式



CITY_NAME または任意の自然キー列では、ソースから取得した自然キー値で常に上書きします。図 A-10 は、CITY_NAME または任意の自然キー列に式を指定する方法の例です。

図 A-10 CITY_NAME の式

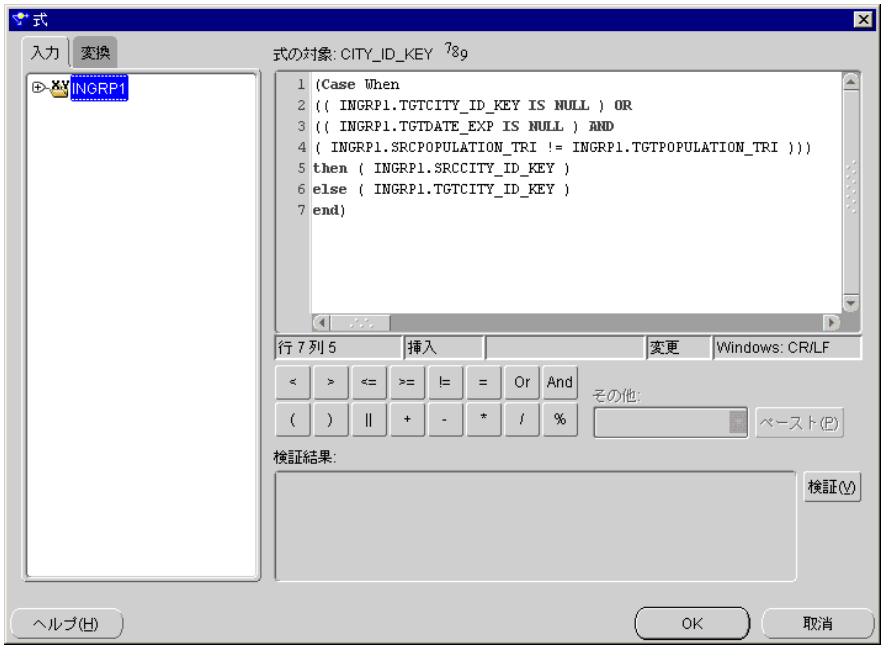


CITY_ID_KEY または任意のサロゲート・キー列では、ターゲットから導出したサロゲート・キー値を維持し、次のことを行う必要があります。

- 一致するバージョンを更新する場合は、最終ターゲット行と照合し、更新を実行します。
- 新バージョンを作成する場合、導出したターゲット・サロゲート・キーは、NULL になることがあります。順序番号は後で導入され、ディメンション・レコードの作成で、一意のサロゲート・キー値が確実に割り当てられます。

図 A-11 は、CITY_ID_KEY または任意のサロゲート・キー列の式を指定する方法の例です。

図 A-11 CITY_ID_KEY の式



STATE_NAME または任意のトリガー以外の列では、ソースから導出した値で常に上書きします。

CITY_POPULATION または任意のトリガー列では、ソースから導出した値で常に上書きします。

ステップ 6: サロゲート・キーの移入

新しいディメンション・レコードのサロゲート・キーとして一意の数値を確実に割り当てるには、順序演算子を使用して、GEO_DIM_TYPE2 のサロゲート・キー列 CITY_ID を挿入します。

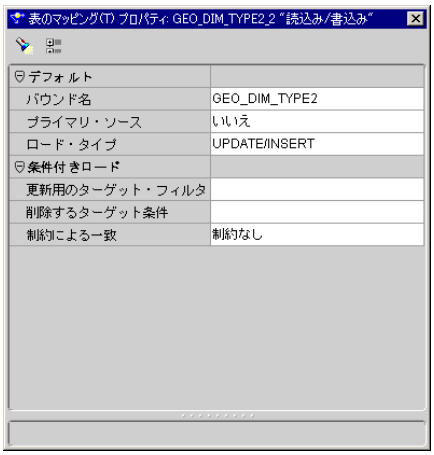
UNION から導出したターゲット・サロゲート・キーは、ロード中に最終ターゲット・サロゲート・キーとの照合に使用します。これを行うには、別の属性 MATCHING を最終ターゲットに作成した後で、導出したターゲット・サロゲート・キー CITY_ID_KEY からその属性にマッピングします。

MATCHING 属性は、最終ターゲットの一意のキーを表し、データを適切にロードするための一致基準として選択されます。ここでは、最終ターゲット・サロゲート・キー列 CITY_ID を MATCHING 属性として使用する必要があります。これを行うには、バウンド名を CITY_ID と同一に設定します。

ステップ 7: ターゲット・プロパティの構成

図 A-12 は、GEO_DIM_TYPE2 演算子のプロパティを構成し、データを適切にロードできるようにする方法の例です。最初に、GEO_DIM_TYPE2 のロード・タイプを「UPDATE/INSERT」に構成する必要があります。

図 A-12「UPDATE/INSERT」を使用した構成



また、マッピングした各列を構成する必要があります。

- CITY_ID はサロゲート・キーで、行の挿入時にのみロードします。
- CITY_NAME は自然キーで、行の挿入時にのみロードします。
- CITY_POPULATION は、行の挿入時と更新時の両方でロードします。STATE_NAME、EFFECTIVE_DATE および EXPIRATION_DATE は、同様に構成します。
- MATCHING は、行の更新時に照合します。

ステップ 8: コードの生成

ターゲット・データベース・タイプ（ウェアハウス・モジュール構成プロパティから構成可能）が Oracle9*i* Database に設定されている場合は、コード生成時に Merge 機能が保証されます。

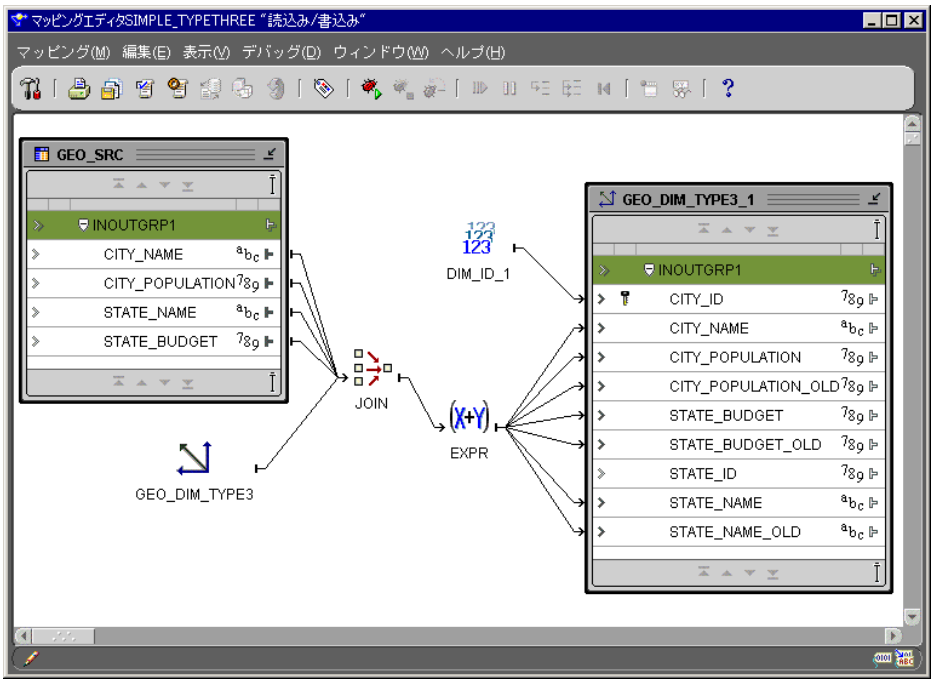
タイプ 3 の緩やかに変化する次元の使用

タイプ 3 の緩やかに変化する次元では、現行値フィールドを作成し、以前の値とは別にディメンション・レコードの現行値を保持します。これを行うには、現行値用と以前の値の保持用の 2 つの列をデータ・フィールドごとに作成する必要があります。

ディメンション GEO_DIM_TYPE3 を定義したら、マッピングに使用してデータをロードできます。GEO_SRC はサンプル・ソース表で、ここから GEO_DIM_TYPE3 にデータをロードします。

タイプ 3 の緩やかに変化する次元にロードするには、ソースからデータを抽出し、それを変換し、ターゲットに直接ロードします。これは、次のマッピング図に従って実行します。この図では、最初に GEO_SRC からデータが抽出され、それが一連の演算子により変換された後に、最終的に GEO_DIM_TYPE3 にロードされます。図 A-13 は、このマッピングの例です。

図 A-13 タイプ 3 の緩やかに変化する次元のマッピング



タイプ 3 の緩やかに変化する次元を作成するには、次のステップを完了する必要があります。

- ステップ 1: 一致の検出
- ステップ 2: 現行値の移入
- ステップ 3: 式による古い値列の移入
- ステップ 4: サロゲート・キーの移入
- ステップ 5: ターゲット・プロパティの構成
- ステップ 6: コードの生成

ステップ 1: 一致の検出

最初に、GEO_SRC のソース行ごとに、それが GEO_DIM_TYPE3 の現行ディメンション・レコードに一致するかどうかを調べる必要があります。これを行うには、結合子を使用して、自然キー列を結合条件にした外部結合を行い GEO_SRC と GEO_DIM_TYPE3 を排他的に照合します。

ステップ 2: 現行値の移入

タイプ 3 の緩やかに変化する次元では、ターゲットの現行値列をソースの列で常に上書きします。これを行うには、結合子出力からターゲット GEO_DIM_TYPE3 に、マッピング線を直接作成します。

ステップ 3: 式による古い値列の移入

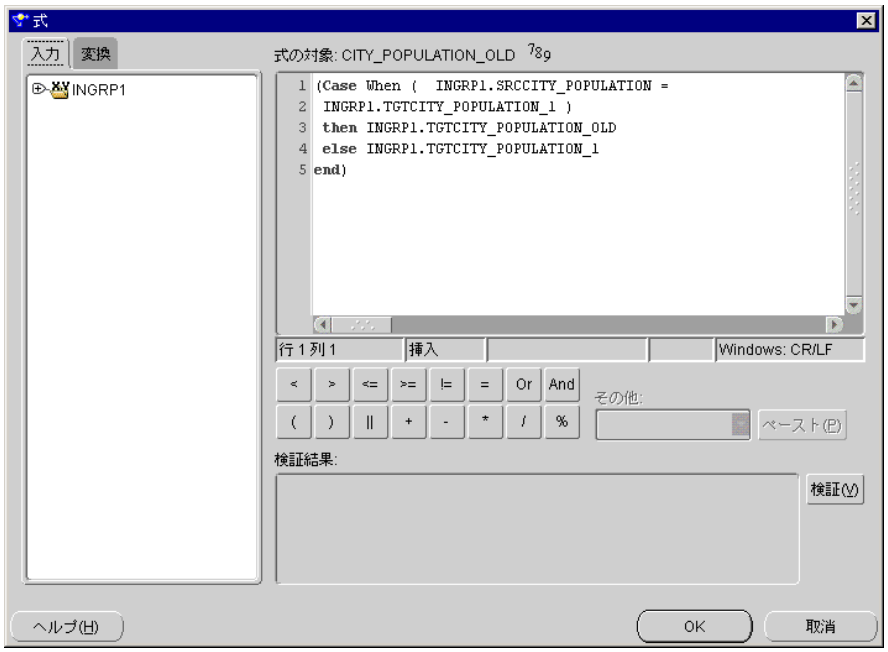
タイプ 3 の緩やかに変化する次元では、CITY POPULATION_OLD、CITY STATE_BUDGET_OLD、CITY STATE_NAME_OLD など、ターゲットの古い値列を上書きするタイミングと方法が問題になります。

特に、次のことを行う必要があります。

- ターゲットの現行値がソースの値と異なる場合は、古い値列をターゲットの現行値列で上書きします。
- その他の場合は、古い値列は変更しません。

これを行うには、前の結合子の結果から式を構築し、CASE 式を使用して式をインスタンス化します。図 A-14 は、CASE 式を使用して式をインスタンス化する方法の例です。

図 A-14 CASE 式



ステップ 4: サロゲート・キーの移入

タイプ 1 の緩やかに変化する次元のステップと同じです。詳細は、A-5 ページの「ステップ 1: サロゲート・キーの移入」を参照してください。

ステップ 5: ターゲット・プロパティの構成

タイプ 1 の緩やかに変化する次元のステップと同様です。

- GEO_DIM_TYPE3 のロード・タイプを「UPDATE/INSERT」に構成する必要があります。
- CITY_ID はサロゲート・キーで、行の挿入時にのみロードします。
- CITY_NAME は自然キーで、行の挿入時にのみロードします。行の更新時は照合します。
- 他は、行の挿入時と更新時の両方でロードします。

ステップ 6: コードの生成

ターゲット・データベース・タイプ（ウェアハウス・モジュール構成プロパティから構成可能）が Oracle9*i* Database に設定されている場合は、コード生成時に Merge 機能が保証されます。

緩やかに変化する次元の配布とロード

ディメンションとマッピングを構成したら、デプロイメント・マネージャを使用して配布および実行します。最適なパフォーマンスを得るには、Oracle9*i* Database をターゲットにして、セットベース・モードでマッピングを実行してください。

索引

か

さ

た

は

ま

や