# Oracle8*i*

## 分散システム

リリース 8.1

ORACLE

Oracle8*i* 分散システム リリース 8.1

部品番号：A62766-1

第 1 版 1999 年 5 月（第 1 刷）

原本名：Oracle8*i* Distributed Database Systems, Release 8.1.5

原本部品番号：A67784-01

原本著者：Jason Durbin

原本協力者：William Creekbaum, Steve Bobrowski, Peter Vasterd, John Bellemore, Anupam Bhide, Roger Bodamer, Jacco Draaijer, Diana Foch-Laurentz, Nina Lewis, Raghu Mani, Basab Maulik, Denise Oertel, Paul Raveling, Kendall Scott, Gordon Smith, Katia Tarkhanov, Randy Urbano, Sandy Venning, Eric Voss, and others

# 目次

## 2 分散データベース管理

# 3 分散トランザクション

## C パススルー SQL の DBMS_HS_PASSTHROUGH

## D DBMS_DISTRIBUTED_TRUST_ADMIN パッケージ・リファレンス



## 索引

# はじめに

このマニュアルでは、Oracle8*i* 分散データベース・システムのインプリメント上の問題点について説明します。また、分散システムのインプリメントとメンテナンスに役立つツールとユーティリティについても説明します。

Oracle8*i* 分散システムでは、Oracle8*i* および Oracle8*i* Enterprise Edition 製品の機能および機能性について説明しています。Oracle8*i* および Oracle8*i* Enterprise Edition は、基本的な機能は同じです。ただし、いくつかの高度な機能は、Enterprise Edition でのみ使用可能です。また、これらの機能はオプションの場合もあります。

# このマニュアルの構成

このマニュアルは、2 部で構成されています。

**第 I 部 : 分散システム**

**第 1 章「分散データベースの概要」**

この章では、Oracle の分散データベース・アーキテクチャの基本概念および用語について説明します。分散システムのインプリメントまたはメンテナンスを計画している方は、この章をお読みください。

**第 2 章「分散データベース管理」**

この章では、分散データベースをインプリメントまたはメンテナンスするデータベース管理者（DBA）にとって重要な事項について説明します。

**第 3 章「分散トランザクション」**

この章では、Oracle が 2 フェーズ・コミットのメカニズムを使用して、分散トランザクションの整合性を維持する方法について説明します。

**第 4 章「分散システムのアプリケーション開発」**

この章では、分散システムで実行するアプリケーションを設計する場合に必要な特殊な考慮事項について説明します。

**第 II 部 : 異機種間サービス**

**第 5 章「Oracle の異機種間サービスの理解」**

この章では、Oracle 異機種間サービスの概要を説明します。

**第 6 章「Oracle 異機種間サービスの管理」**

この章では、異機種間サービスのインプリメントとメンテナンス方法を説明します。

**第 7 章「異機種間サービスを使用したアプリケーション開発」**

この章では、Oracle の異機種間サービスを使用するアプリケーションを開発するために必要な情報を提供します。

**付録 A「異機種間サービスの初期化パラメータ」**

この付録では、異機種間サービス固有の初期化パラメータすべてとその値のリストを記載します。

**付録 B「DBMS_HS パッケージ・リファレンス」**

この付録では、DBMS_HS パッケージに関するすべてのインタフェース情報を提供します。DBMS_HS パッケージは、異機種間サービスを管理するために使用します。

**付録 C「パススルー SQL の DBMS_HS_PASSTHROUGH」**

この付録では、パススルー SQL 用の DBMS_HS_PASSTHROUGH パッケージに関するすべてのインタフェース情報を提供します。

**付録 D「DBMS_DISTRIBUTED_TRUST_ADMIN パッケージ・リファレンス」**

この付録では、Trusted Server リストを管理するための、パッケージ DBMS_DISTRIBUTED_TRUST_ADMIN のプロシージャおよびファンクションについて説明します。

## このマニュアルで使用する表記規則

このマニュアルでは、構文例の中で次の規則を使用します。

| | |
|---|---|
| 大文字 | 本文中の英大文字は、示されているとおりに入力する必要があるテキストを示します。次に例を示します。<br>`SQLPLUS username/password`<br>`INTO TABLENAME 'table'` |
| 小文字のイタリック体 | 小文字のイタリック体は、適切な値に置換する必要のある変数を示します。次に例を示します。<br>`VARCHAR (length)` |
| 垂直バー | 垂直バーは、どちらかを選択できることを示します。次に例を示します。<br>`ASC \| DESC` |
| 中カッコ（{}） | 必須項目は中カッコで囲まれています。これは、中カッコで囲まれた選択肢の中から 1 つを選択する必要があることを示します。次に例を示します。<br>`{column_name \| array_def}` |
| 大カッコ | オプション項目は大カッコに囲まれています。次に例を示します。<br>`DECIMAL (digits [ , precision ])` |
| < 演算子 > | SQL 演算子は、<operator> で示します。次に例を示します。<br>`WHERE x <operator> x` |
| 省略記号（...） | 繰返し可能な項目は大カッコで囲み省略記号を付けて示します。次に例を示します。<br>`WHERE column_1 <operator> x`<br>`AND column_2 <operator> y`<br>`[AND ...]` |

# 第I部

## 分散システム

# 1

# 分散データベースの概要

この章では、Oracle の分散データベース・アーキテクチャの基本概念および用語について説明します。次の事項が含まれます。

- Oracle の分散データベース・アーキテク チャ
- 異機種間分散データベース
- 分散データベース・アプリケーションの開発
- Oracle 分散システムの管理
- 各国語サポート

# Oracle の分散データベース・アーキテク チャ

「分散データベース」は、複数のコンピュータに格納された一連のデータベースで、単一のデータベースとしてアプリケーションに認識されます。したがって、ネットワーク上の複数のデータベースに、アプリケーションから同時にアクセスして変更できます。システムの各 Oracle データベースは、ローカルの Oracle サーバーにより制御されますが、各サーバーの協調によりグローバルな分散データベースの一貫性が維持されます。図 1-1 に、代表的な Oracle 分散データベース・システム（以下、分散システム）を示します。

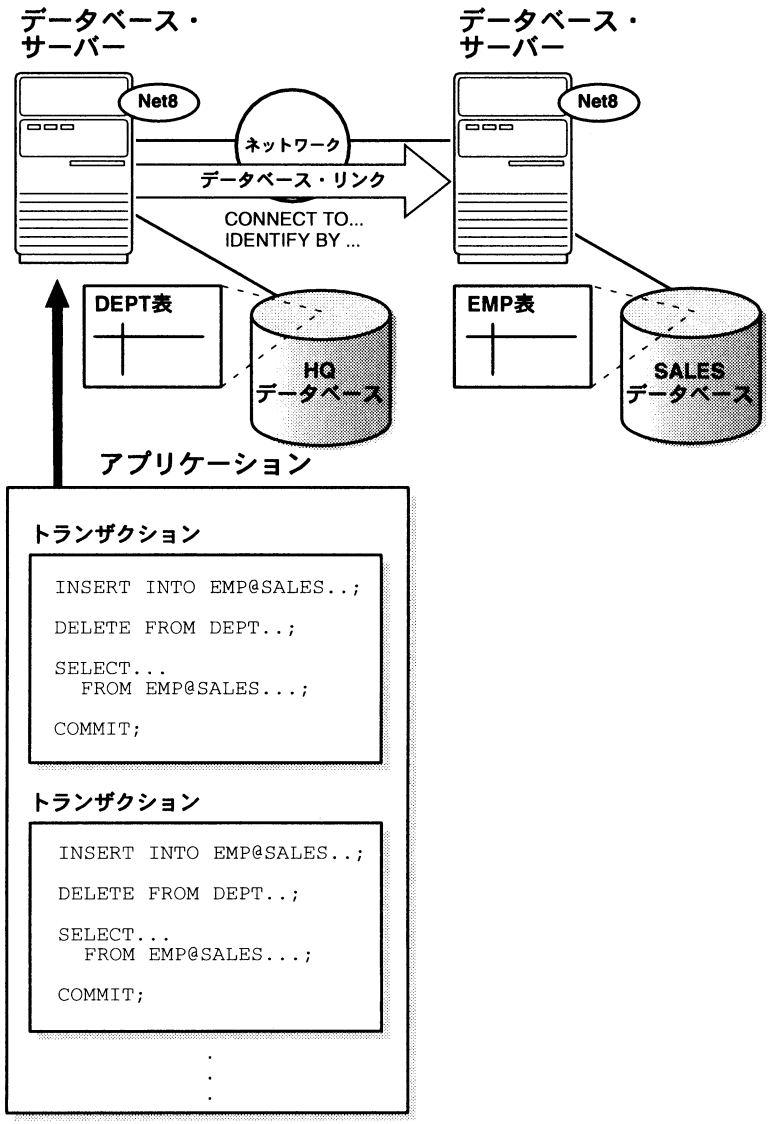
## クライアントとサーバー

「データベース・サーバー」はデータベースを管理する Oracle ソフトウェアで、「クライアント」はサーバー情報を要求するアプリケーションです。システム内の各コンピュータが、1 つの「ノード」です。分散システムのノードは、状況に応じてクライアントまたはサーバー、あるいはその両方として動作します。たとえば、図 1-1 では、HQ データベースを管理するコンピュータは、ローカル・データに対してトリガー文を発行するとき（各トランザクションの 2 番目のトリガー文によって、ローカルの DEPT 表に対する問合せが発行される場合など）には、データベース・サーバーとして動作し、リモート・データに対してトリガー文を発行するとき（各トランザクションの 1 番目のトリガー文が、SALES データベースのリモートの EMP 表に対して発行される場合など）には、クライアントとして動作します。

### 直接接続と間接接続

クライアントは、データベース・サーバーに直接または間接的に接続できます。図 1-1 では、クライアント アプリケーションによって、各トランザクションの 1 番目と 3 番目の文が発行されると、クライアントは中間の HQ データベースに直接接続され、さらにリモート・データが含まれている SALES データベースに間接接続されます。

**図 1-1 Oracle 分散システム**



# ネットワーク

分散システムの個々のデータベースをリンクするには、ネットワークが必要です。次のセクションでは、Oracle 分散システムでのネットワークの問題点について説明します。

## Net8

分散システムにあるすべての Oracle データベースは、ネットワーク上のデータベース間通信を容易にする Oracle のネットワーキング・ソフトウェアである Net8 を使用します。Net8 は、1 つのネットワーク上の異なるコンピュータで動作するクライアントとサーバーを接続するだけでなく、データベース・サーバーが複数のネットワーク上の分散データベース内のリモート・トランザクションや分散トランザクションをサポートできるようにします。

Net8 により、システムを使用するアプリケーションが SQL 要求を送信し、データを受信するために必要な接続性が透過的に確立されます。Net8 は、クライアントから SQL 文を受信し、サポートされている業界標準の通信プロトコルまたはプログラム・インタフェースを介して Oracle サーバーに送信するために、それらをパッケージ化します。また、サーバーから応答を受信し、適切なクライアントに返信するために、それらをパッケージ化します。Net8 は、基礎を形成するネットワーク・オペレーティング・システムとは独立して、すべての処理を実行します。Net8 とその機能の詳細は、『Oracle8*i* Net8 管理者ガイド』を参照してください。

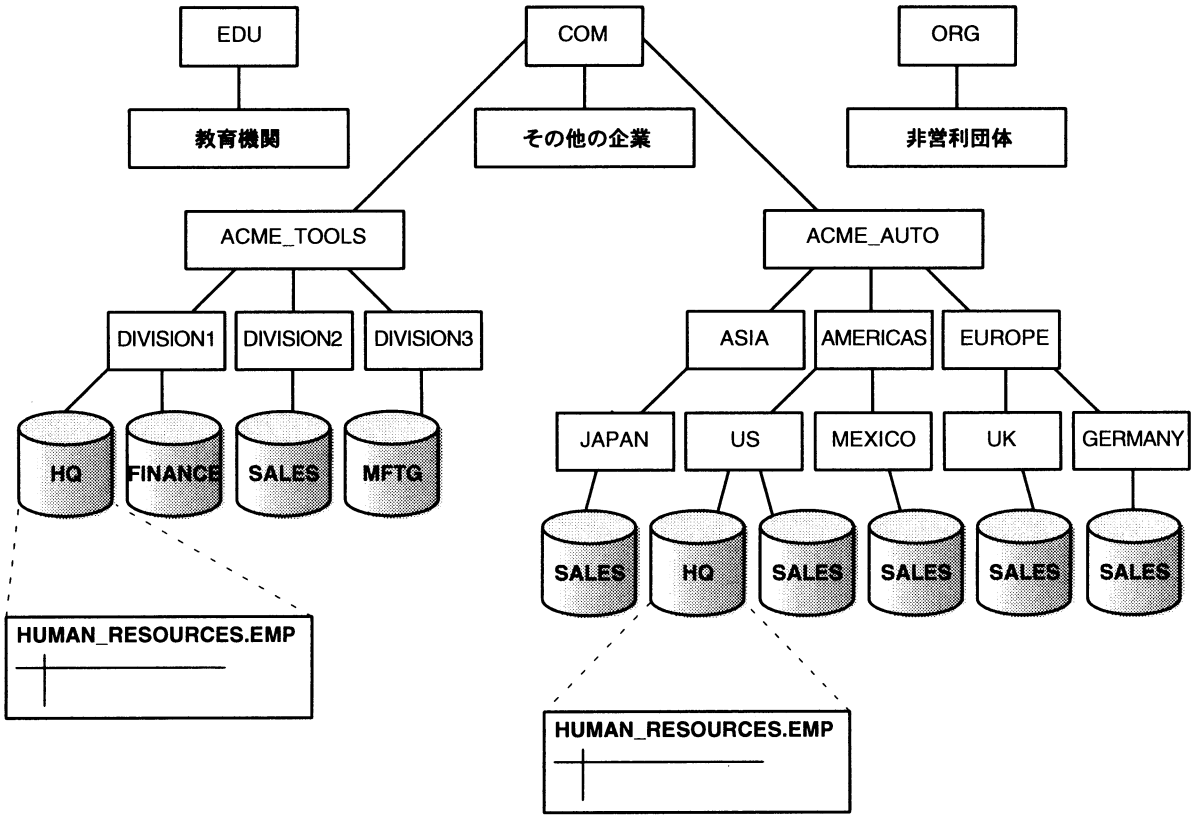
## Oracle Names

任意に Oracle ネットワークで Oracle Names を使用して、グローバルなディレクトリ・サービスを持つシステムを実現できます。Oracle ネットワークが分散システムをサポートする場合は、システムにある各データベースに関する情報の中央リポジトリとして Oracle Names サーバーを使用すると、分散データベース・アクセスの構成を単純化できます。

# データベースとデータベース・リンク

分散データベースの各データベースは、システム内の他のすべてのデータベースとは区別され、それ自体の「グローバル・データベース名」を持ちます。Oracle では、ネットワーク・ドメインの接頭辞として個別のデータベース名を付けて、グローバル・データベース名を形成します。たとえば、図 1-2 は、ネットワーク内でのデータベースの代表的な階層型配置を示しています。

**図 1-2 ネットワーク・ディレクトリとグローバル・データベース名**




複数のデータベースに同じ個別名を付けることができますが、各データベースに一意のグローバル・データベース名が必要です。たとえば、ネットワーク・ドメイン US.AMERICAS.ACME_AUTO.COM と UK.EUROPE.ACME_AUTO.COM はそれぞれ SALES データベースを含みます。

```
SALES.US.AMERICAS.ACME_AUTO.COM
SALES.UK.EUROPE.ACME_AUTO.COM
```

## データベース・リンク

分散システムでアプリケーションの要求を容易にするには、「データベース・リンク」を使用します。データベース・リンクは、Oracle データベースから別のデータベースに一方向の通信パスを定義します。

データベース・リンク名は、そのリンクが指すデータベースのグローバル名と同じであるため、Oracle 分散システムのユーザーにとってデータベース・リンクは、事実上、透過的なものです。

たとえば、次の SQL 文はローカル・データベース内にリモート・データベース SALES.US.AMERICAS.ACME_AUTO.COM へのパスを示すデータベース・リンクを作成します。

```
CREATE DATABASE LINK sales.us.americas.acme_auto.com ... ;
```

データベース・リンクを作成すると、ローカル・データベースに接続されたアプリケーションはリモート・データベース SALES.US.AMERICAS.ACME_AUTO.COM のデータにアクセスできます。次の項では、分散データベース内でリモートのスキーマ・オブジェクトを参照する方法を、データベース・リンクを使用した SQL 文の例も含めて説明します。

---

**注意：** Oracle は、さまざまなタイプのデータベース・リンクをサポートしています。詳細は、2-3 ページの「データベース・リンクのタイプ」を参照してください。

---

## スキーマ・オブジェクト名の解決

スキーマ・オブジェクトへのアプリケーション参照を解決する（このプロセスを「ネーム変換」といいます）には、階層型のアプローチを使用してオブジェクト名を形成します。たとえば、1 つのデータベース内部では、各スキーマは必ず一意の名前を持ち 、1 つのスキーマの内部では、各オブジェクトは必ず一意の名前を持ちます。その結果、スキーマ・オブジェクトの名前は、データベース内で常に一意です。さらにオブジェクトのローカル名へのアプリケーション参照を簡単に解決できます 。

分散データベースでは、表などのスキーマ・オブジェクトにシステム内の全アプリケーションからアクセスできます。Oracle では、グローバル・データベース名を使用して階層型に名前が付けられたモデルを単純に拡張することによって、分散システムで効果的に「グローバル・オブジェクト名」を作成し、スキーマ・オブジェクトへの参照を解決できるようになります。たとえば、リモート表が常駐するデータベースも含めた完全な修飾名を指定すると、問合せでリモート表を参照できます。

```
SELECT * FROM scott.emp@sales.us.americas.acme_auto.com;
```

要求を完了するには、ローカル・データベース・サーバーが、リモートの SALES データベースに接続するデータベース・リンクを暗黙的に使用します。

## バージョンの異なる Oracle Server との接続

Oracle 分散システムでは、異なるバージョンの Oracle データベースを取り込めます。サポートされているリリースの Oracle は、すべて分散システムに加えることができます。ただし、分散データベースを使用して作業するアプリケーションの場合は、システム内の各ノードで使用可能な機能を理解する必要があります。

たとえば、分散データベース・アプリケーションの場合、Oracle7 のデータベースでは Oracle8*i* のデータベースで使用可能なオブジェクトの SQL 拡張要素を認識できません。

## 分散データベースと分散処理

「分散データベース」と「分散処理」という用語は、密接に関連していますが、まったく異なる意味を持ちます。

| | |
|---|---|
| 分散データベース | 分散データベースは、複数のコンピュータに格納された一連のデータベースで、単一のデータベースとしてアプリケーションに認識されます。 |
| 分散処理 | 分散処理は、アプリケーション・システムがネットワーク上の異なるコンピュータにタスクを分散するときに発生します。たとえば、データベース・アプリケーションは、通常、フロントエンド・タスクをクライアント PC または NC に分散し、データベースに対する共有アクセスをバックエンド・データベース・サーバーに管理させます。したがって、分散データベース・アプリケーションの処理システムは、通常は「クライアント・サーバー」データベース・アプリケーション・システムと呼ばれます。 |

Oracle 分散システムは、分散処理アーキテクチャを機能に取り入れています。たとえば、Oracle サーバーが別の Oracle サーバーが管理するデータを要求するときは、そのサーバーはクライアントとして動作します。

## 分散データベースとデータベース・レプリケーション

「分散データベース」と「データベース・レプリケーション」という用語も、密接に関連していますが、異なる意味を持ちます。純粋な分散データベースでは、全データの単一のコピーをシステムが管理し、データベース・オブジェクトをサポートします。分散データベース・アプリケーションでは、通常は分散トランザクションを使用して、ローカル・データとリモート・データの両方にアクセスし、リアルタイムでグローバル・データベースを変更します。

---

**注意：** このマニュアルでは、純粋な分散データベースについて説明します。

---

レプリケーションは、分散システムを構成する複数のデータベースにあるデータベース・オブジェクトをコピーし、維持するプロセスです。レプリケーションは分散データベース・テクノロジに依存して機能しますが、データベース・レプリケーションでは、純粋な分散データベース環境では実現できないアプリケーションの利点を提供できます。

一般的に、レプリケーションには代替のデータ・アクセス・オプションがあるため、パフォーマンスを改善し、アプリケーションの可用性を保護するために有効です。たとえば、ネットワーク・トラフィックを最小に抑え、パフォーマンス最大にするために、アプリケーションでは、通常リモート・サーバーではなく、ローカル・データベースにアクセスします。さらに、ローカル・サーバーに障害が起きても、レプリケート・データが存在する他のサーバーがアクセス可能であれば、アプリケーションは処理を継続できます。

---

**注意：** Oracle のレプリケーション機能の詳細は、『Oracle8*i* レプリケーション・ガイド』を参照してください。

---

# 異機種間分散データベース

Oracle の「異機種間分散データベース・システム」では、少なくとも 1 つのデータベース・システムが非 Oracle システムです。アプリケーションでは、異機種間分散データベース・システムは、ローカルな単一の Oracle データベースとして認識されます。ローカルな Oracle サーバーでは、データが分散し、異機種上にあることを隠すことができます。Oracle サーバーから非 Oracle システムに接続するには、Oracle8*i* 異機種間サービスおよび非 Oracle システムに固有の異機種間サービス・エージェントを使用します。

## 異機種間サービス

異機種間サービスは、Oracle8*i* Server 内の統合化されたコンポーネントで、Oracle の次世代のオープン・ゲートウェイ製品に対するテクノロジを使用可能にします。異機種間サービスでは、将来の Oracle ゲートウェイ製品およびその他異機種間アクセス機能に対して、共通のアーキテクチャと管理メカニズムを提供する一方、初期のリリースの Oracle Open Gateway 製品のユーザーには上位互換機能を提供します。

詳細は、第 5 章「Oracle の異機種間サービスの理解」を参照してください。

## 異機種間サービス・エージェント

異機種間サービスでは、アクセス先の非 Oracle システムごとに、その非 Oracle システムにアクセスするための「エージェント」が必要です。異機種間サービス・エージェントを使用して、非 Oracle システムおよび Oracle サーバー内の異機種間サービス・コンポーネントと通信できます。非 Oracle システムでは、Oracle サーバーのかわりにエージェントによって、SQL、プロシージャおよびトランザクション要求が実行されます。

Gateway バージョン 8 は、プロシージャまたは SQL を使用して非 Oracle システムにアクセスする異機種間サービス・エージェントの Oracle 製品名です。ただし、異機種間サービス・エージェントは、Oracle Transparent Gateway または Oracle Procedural Gateway 以外の製品として使用することも可能です。このマニュアルでは、より一般的な、異機種間サービス・エージェントという用語を使用します。Oracle Open Gateway バージョン 8 を購入済みの場合は、異機種間サービス・エージェントという用語のかわりに「Oracle Open Gateway バージョン 8」を使用できます。

バージョン 8 のゲートウェイのインストレーションおよび構成の詳細は、『Oracle Open Gateway インストレーションおよびユーザーズ・ガイド』を参照してください。

## 機能

異機種間サービスの機能には次のものが含まれます。

- 分散トランザクション。トランザクションは、Oracle システムと非 Oracle システムの両方にまたがることができ、変更内容は、Oracle の 2 フェーズのコミット・メカニズムにより、すべてコミットまたはすべてロールバックされることが保証されます。
- 透過的 SQL アクセス。非 Oracle システムのデータが Oracle 環境に統合され、データが 1 つのローカル・データベースに格納されているかのように認識されます。アプリケーションが発行する SQL 文は、非 Oracle システムが認識できる SQL 文に透過的に変換されます。
- プロシージャ・アクセス。PL/SQL リモート・プロシージャ・コールを使用して、Oracle8*i* Server からメッセージング・システムやキューイング・システムなどのプロシージャ型システムにアクセスできます。
- データ・ディクショナリの変換。Oracle のデータ・ディクショナリ表への参照が含まれる SQL 文を、非 Oracle システムのデータ・ディクショナリ表への参照が含まれる SQL 文に変換して、非 Oracle システムが Oracle サーバーの 1 つとして認識されるようにします。
- パススルー SQL。アプリケーション・プログラマは、必要な場合は、非 Oracle システムの SQL 言語を使用して、Oracle アプリケーションから非 Oracle システムに直接アクセスできます。
- ストアド・プロシージャへのアクセス。SQL ベースの非 Oracle システム内のストアド・プロシージャに、PL/SQL リモート・プロシージャであるかのようにアクセスできます。
- 各国語サポート。異機種間サービスではマルチバイト・キャラクタ・セットがサポートされ、非 Oracle システムと Oracle8*i* サーバーの間でキャラクタ・セットが変換されます。
- マルチスレッド・エージェント。マルチスレッド・エージェントは、オペレーティング・システムのスレッドイン機能を活用します。マルチスレッド・エージェントでは、マルチスレッド・サーバー機能を活用することにより、必要な処理を減少させます。

- エージェント自動登録。エージェント自動登録により、リモート・ホスト上で異機種間サービスの構成データを更新するプロセスが自動化され、異機種間データベース・リンクの正常動作が保証されます。
- 管理インタフェース。アクティブな異機種間サービス・エージェントと、それにアクセスしているユーザー・セッションのグラフィック表現を提供します。

---

**注意：** 使用している異機種間サービス・エージェントまたはOracle Open Gatewayによっては、前述した機能の一部がサポートされていない場合があります。サポートされる機能については、異機種間サービスまたはOracle Open Gatewayのマニュアルを参照してください。

---

# 分散データベース・アプリケーションの開発

分散システム上にアプリケーションを構築するときは、考慮すべき問題点がいくつかあります。次のセクションでは、分散データベースでアプリケーションがデータにアクセスする方法について説明します。

## 分散問合せの最適化

「分散問合せの最適化」は、Oracle8*i* のデフォルト機能です。この機能により、分散SQL文で参照されるリモート表からデータを取り出すときに、サイト間に必要なデータ転送量が減少します。

分散問合せの最適化では、Oracleのコストベース・オプティマイザを使用して、リモート表から必要なデータのみを抽出し、そのデータをリモート・サイトで処理して、結果を最終処理のためにローカル・サイトに戻すSQL文を検索または生成します。これにより、すべての表データを処理のためにローカル・サイトに転送する場合に比べて、データ転送量が減少します。

DRIVING_SITE、NO_MERGEおよびINDEXのヒントなど、コストベース・オプティマイザのヒントを使用すると、Oracleによるデータの処理方法とアクセス方法をさらに制御できます。

詳細は、4-4ページの「分散問合せのチューニング」を参照してください。

## リモートおよび分散SQL文

「リモート問合せ」は、同じリモート・ノードに常駐している1つ以上のリモート表から情報を選択する問合せです。次に例を示します。

```
SELECT * FROM scott.dept@sales.us.americas.acme_auto.com;
```

「リモート更新」は、同じリモート・ノードに常駐している 1 つ以上の表内のデータを変更する更新のことです。

次に例を示します。

```
UPDATE scott.dept@sales.us.americas.acme_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
```

> **注意：** リモート更新には、1 つ以上のリモート・ノードからデータを取り出す副問合せが含まれる場合がありますが、更新は 1 つのリモート・ノードで行われるため、その文はリモート更新として分類されます。

「分散問合せ」は、2 つ以上のノードから情報を取り出します。次に例を示します。

```
SELECT ename, dname
  FROM scott.emp e, scott.dept@sales.us.americas.acme_auto.com d
  WHERE e.deptno = d.deptno;
```

「分散更新」は、2 つ以上のノード上のデータを変更します。プロシージャまたはトリガーなど、異なるノード上のデータにアクセスするリモート更新を 2 つ以上含む PL/SQL サブプログラムの単位を使用すると、分散更新を実行できます。次に例を示します。

```
BEGIN
  UPDATE scott.dept@sales.us.americas.acme_auto.com
    SET loc = 'NEW YORK'
    WHERE deptno = 10;
  UPDATE scott.emp
    SET deptno = 11
    WHERE deptno = 10;
END;
```

プログラム内の文はリモート・ノードに送信され、それらが 1 つの単位として実行され、正常終了または失敗となります。

## リモート・プロシージャ・コール（RPC）

開発者は、分散データベースで作業するアプリケーションをサポートするために、PL/SQL パッケージとプロシージャを作成できます。アプリケーションでは、ローカル・データベースで作業を実行するローカル・プロシージャ・コール、およびリモート・データベースで作業を実行する「リモート・プロシージャ・コール（RPC）」を作成できます。プログラムがリモート・プロシージャをコールする場合は、ローカル・サーバーが全プロシージャ・パラメータをリモート・サーバーに渡します。次に例を示します。

```
BEGIN
 emp_mgmt.del_emp@sales.us.americas.acme_auto.com(1257);
END;
```

分散システムのパッケージとプロシージャを開発する場合、開発者は、リモート・ロケーションで実行すべきプログラム・ユニットの処理内容、およびコール元のアプリケーションに結果を戻す方法を理解した上でコードを作成する必要があります。

# リモート・トランザクションと分散トランザクション

「リモート・トランザクション」は、すべて同じリモート・ノードを参照している、1 つ以上のリモート文を含むトランザクションです。次に例を示します。

```
UPDATE scott.dept@sales.us.americas.acme_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
UPDATE scott.emp@sales.us.americas.acme_auto.com
  SET deptno = 11
  WHERE deptno = 10;
COMMIT;
```

「分散トランザクション」は、分散データベースの 2 つ以上の異なるノード上にあるデータを、個別に、またはグループとして更新する文を 1 つ以上含むトランザクションです。次に例を示します。

```
UPDATE scott.dept@sales.us.americas.acme_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
UPDATE scott.emp
  SET deptno = 11
  WHERE deptno = 10;
COMMIT;
```

**注意：** トランザクションのすべての文が 1 つのリモート・ノードのみを参照する場合、そのトランザクションはリモートであり、分散ではありません。

### 2 フェーズ・コミットのメカニズム

DBMS は、分散システムであってもなくても、トランザクション内のすべての文を 1 単位としてコミットまたはロールバックする必要があります。その結果、トランザクションが適切に設計されていれば、論理データベース内のデータの整合性は常に保証されます。進行中のトランザクションによる影響は、全ノードの他のすべてのトランザクションには認識できないようにする必要があります。これは、問合せ、更新またはリモート・プロシージャ・コールなど、どのタイプの操作を含むトランザクションにも当てはまります。

分散データベースでない場合のトランザクション制御の一般的なメカニズムの詳細は、『Oracle8*i* 概要』を参照してください。分散データベースでは、Oracle は、ネットワーク全体にわたって同一の特性に基づいてトランザクション制御を調整し、ネットワーク障害またはシステム障害が起こった場合でも、データ整合性を維持する必要があります。

Oracle の「2 フェーズ・コミット」メカニズムは、分散トランザクションに参加しているすべてのデータベース・サーバーが、そのトランザクション内の文をすべてコミットまたはすべてロールバックすることを保証します。2 フェーズ・コミット・メカニズムは、整合性制約、リモート・プロシージャ・コールおよびトリガーにより実行される暗黙の DML 操作も保護します。

---

**注意：** Oracle の 2 フェーズ・コミット・メカニズムの詳細は、第 3 章「分散トランザクション」を参照してください。

---

## 分散システムでの透過性

システムを使用するユーザーに対して、最小限の労力で Oracle 分散システムの機能を透過的にできます。透過性の目標は、分散システムを 1 つの Oracle データベースとして認識されるように構築することです。その結果、開発者は分散データベースのアプリケーション開発の困難を回避し、ユーザーは生産性低下から解放されます。次のセクションでは、分散システムでの透過性の詳細を説明します。

### 位置の透過性

Oracle 分散システムには、アプリケーション開発者および管理者が、データベース・オブジェクトの物理的な位置をアプリケーションおよびユーザーから隠す機能があります。アプリケーションが接続されているノードに関わりなく、表などのデータベース・オブジェクトをユーザーが普遍的に参照できる状態を「位置の透過性」があるといいます。位置の透過性には、次のような利点があります。

- ユーザーがデータベース・オブジェクトの物理的な位置を知る必要がないので、リモート・データへのアクセスが簡素化されます。
- 管理者は、エンド・ユーザーまたは既存のデータベース・アプリケーションに影響を与えずに、データベース・オブジェクトを移動できます。

一般的に、管理者と開発者はシノニムを使用して、アプリケーション・スキーマ内の表およびサポート・オブジェクトに対する位置の透過性を確立します。次の文は、リモート・データベースの別々の表に対してデータベースでシノニムを作成します。

```
CREATE PUBLIC SYNONYM emp
  FOR scott.emp@sales.us.americas.acme_auto.com
CREATE PUBLIC SYNONYM dept
  FOR scott.dept@sales.us.americas.acme_auto.com
```

次のような問合せで、リモート表にアクセスできます。

```
SELECT ename, dname
  FROM scott.emp@sales.us.americas.acme_auto.com e,
       scott.dept@sales.us.americas.acme_auto.com d
  WHERE e.deptno = d.deptno;
```

しかし、この場合はシノニムがあるので、リモート表の位置について考慮せずに、簡単な問合せを発行できます。

```
SELECT ename, dname
  FROM emp e, dept d
  WHERE e.deptno = d.deptno;
```

分散システムで動作するアプリケーションに対する位置の透過性を確立するには、シノニムに加えて、ビューおよびストアド・プロシージャも使用できます。

## 文とトランザクションの透過性

Oracle の分散データベース・アーキテクチャでは、問合せ、更新およびトランザクションの透過性も提供されます。たとえば、SELECT、INSERT、UPDATE および DELETE などの標準 SQL コマンドは、分散データベースでない環境で実行する場合とまったく同様に使用できます。さらに、アプリケーションでは、標準 SQL コマンドの COMMIT、SAVEPOINT および ROLLBACK を使用してトランザクションを制御するため、分散トランザクションの制御を行うための複雑なプログラミングやその他の特殊操作は不要です。

- 1 つのトランザクションの文で、ローカル表またはリモート表を必要な数だけ参照できます。
- 分散トランザクションに関係する全ノードが、トランザクションをすべてコミットまたはすべてロールバックするという同じアクションを実行することが保証されます。
- 分散トランザクションのコミットの最中にネットワーク障害またはシステム障害が発生した場合でも、そのトランザクションは自動的、透過的に、グローバルに解決されます。つまり、ネットワークまたはシステムが復元されると、ノードはトランザクションをすべてコミットするか、またはすべてロールバックします。

**内部操作** コミット済みの各トランザクションには、対応する「システム変更番号（SCN）」があり、そのトランザクション内の文による変更を一意に識別します。分散データベースでは、各通信ノードの SCN が調整されるのは次のような場合です。

- 1 つ以上のデータベース・リンクにより記述されているパスを使用して接続が確立されるとき。
- 分散 SQL 文が実行されるとき。
- 分散トランザクションがコミットされるとき。

その他の利点として、分散システムのノード間の SCN を調整することにより、文レベルとトランザクション・レベルの両方でグローバルな分散読取りの一貫性が実現します。必要な場合には、グローバルな時間ベースの分散回復も実行できます。

### レプリケーションの透過性

システムのノード間でデータを透過的にレプリケートするための各種機能も 提供しています。Oracle のレプリケーション機能の詳細は、『Oracle8*i* レプリケーション・ガイド』を参照してください。

## Oracle 分散システムの管理

Oracle 分散システムを対象としたアプリケーション開発には特有の考慮事項があるように、分散データベース管理についても特殊な問題点を理解しておく必要があります。次の項では、Oracle 分散システムでデータベースを管理するための特別なトピックについて説明します。第 6 章「Oracle 異機種間サービスの管理」も参照してください。

## サイト自律性

「サイト自律性」は、分散データベースに参加している各サーバーが、各データベースが非分散データベースとして運用されているかのように、他のすべてのデータベースから独立して管理されることを意味します。

いくつかのデータベースは一緒に動作できますが、各データベースは異なるもので、別々に管理するデータが格納された独立したリポジトリです。Oracle 分散データベースのサイト自律性の利点として次の事項が挙げられます。

- システムのノードは、「互いに手の届く」関係の維持を必要とする会社や共同組織の論理編成をミラー化できます。
- ローカル・データベース管理者がローカル・データを制御します。したがって、各データベース管理者が責任を負うドメインが小さくなり、管理しやすくなります。

- 独立して発生した障害によって分散データベースの他のノードが損傷する危険性が低下します。グローバルな Oracle データベースは、1 つのデータベースとネットワークが使用可能な限り、部分的に使用できます。1 つのデータベースの障害が原因で、すべてのグローバルな操作が停止したり、パフォーマンス上のボトルネックが生じたりすることはありません。
- 管理者は、孤立したシステム障害をシステムの他のノードから独立して回復できます。
- 各ローカル・データベース用のデータ・ディクショナリがあり、ローカル・データにアクセスするためのグローバルなカタログは不要です。
- ノードは、それぞれ独立してソフトウェアをアップグレードできます。

分散システム内の各データベースを独自に管理できるとしても、システムのグローバルな要件を無視することはできません。

たとえば、各データベースで必要となる追加のユーザー・アカウントは、サーバー間接続を容易にするリンクのサポートに必要です。次の項では、これらの特定のトピックについて説明し、システムの個々のノードを管理するときに、分散データベース環境全体のグローバルな側面の必要性を示します。

# 分散データベースのセキュリティ

分散データベースでない環境で使用可能なすべてのセキュリティ機能を、分散システムに対してサポートしています。これらの機能は次のとおりです。

- ユーザーとロールに対するパスワードまたは外部サービスの認証
- クライアントからサーバーへの接続およびサーバー間接続でのログイン・パケットの暗号化

次の項では、Oracle 分散システムを構成するときに考慮するその他のトピックについて説明します。

## ユーザー・アカウントとロールのサポート

分散システムでは、システムを使用するアプリケーションのサポートに必要なユーザー・アカウントおよびロールを慎重に計画する必要があります。

- サーバー間接続の確立に必要なユーザー・アカウントは、分散システムの全データベースで使用可能である必要があります。
- 分散データベースのアプリケーション・ユーザーがアプリケーション権限を使用するために必要なロールは、分散システムの全データベースで提示されている必要があります。

分散システム内のノードに対してデータベース・リンクを作成する場合は、リンクを使用するサーバー間接続のために各サイトが必要とするユーザー・アカウントおよびロールを確認してください。システム内の異なるタイプのデータベース・リンクのサポートに必要なユーザー・アカウントの詳細は、2-3 ページの「データベース・リンクのタイプ」を参照してください。

## グローバル・ユーザーとグローバル・ロール

一般的に、分散環境ではユーザーが多数のネットワーク・サービスにアクセスする必要があります。各ネットワーク・サービスにアクセスする各ユーザーに対して別個の認証を構成する必要がある場合、特にそれが大規模なシステムの場合には、セキュリティ管理の取り扱いが難しくなります。

グローバル認証サービスの使用は、分散環境に対するセキュリティ管理を簡素化するための一般的なテクニックです。

Oracle クライアント / サーバーまたは分散データベース環境では、ユーザーおよびロールに対するグローバル認証をサポートするためのオプションが 2 つあります。

- Oracle Security Server は、Oracle ネットワークで認証の集中化と分散化をサポートする製品です。

> **注意：** Oracle8 で使用可能だったグローバル・ユーザー機能は修正中であり、現在はベータ版のユーザーのみ使用可能です。Oracle8*i* では、今後のリリースに組み込まれる予定です。

- グローバルなデータベース・ユーザーおよびロールの認証を非 Oracle の認証サービス（DCE など）フレームワーク内で操作する必要がある場合、Oracle 分散データベース環境では、Net8 の Advanced Networking Option を使用できます。Net8 Advanced Networking Option はオプション製品で、Net8 および Oracle 分散データベース・システムのセキュリティの拡張に使用できる多数の機能が組み込まれています。詳細は、『Oracle8*i* Advanced Security Option 管理者ガイド』を参照してください。

## データの暗号化

Net8 Advanced Networking Option は、データの不正な解読と改ざんを防止するネットワーク・データの暗号化とチェックサムを、Net8 とその関連製品が使用できるようにします。また、RSA データ・セキュリティ RC4 またはデータ暗号化規格（DES）の暗号化アルゴリズムを使用して、認証されていない状態でのデータ表示を防止します。

伝送中にデータが改ざん、削除または再生されていないことを保証するために、Advanced Networking Option のセキュリティ・サービスは、暗号による保護メッセージ・ダイジェストを生成し、ネットワークを介して送信する各パケットにそのダイジェストを組み込むことができます。

Net8 の Advanced Networking Option のさまざまな機能の詳細は、『Oracle8*i* Net8 管理者ガイド』および『Oracle8*i* Advanced Security Option 管理者ガイド』を参照してください。

# Oracle 分散データベースの管理ツール

Oracle 分散システムを管理するときに使用するツールについて、データベース管理者にはいくつかの選択肢があります。

- Oracle Enterprise Manager
- サードパーティの管理ツール
- SNMP サポート

## Enterprise Manager

Enterprise Manager は、Oracle のデータベース管理ツールです。Enterprise Manager のグラフィカル・コンポーネント（Enterprise Manager/GUI）によって、便利なグラフィカル・ユーザー・インタフェース（GUI）を使用したデータベース管理タスクを実行できます。

Enterprise Manager のライン・モード・コンポーネントは、ライン・モード・インタフェースを提供します。

Enterprise Manager は、使いやすいインタフェースを介して管理機能を提供します。Enterprise Manager は次の目的に使用できます。

- 従来の管理タスク（データベースの起動、シャットダウン、バックアップおよび回復など）の実行。手動で SQL コマンドを入力してこれらのタスクを実行するより、Enterprise Manager のグラフィカル・インタフェースを使用して、マウスで指示しクリックすることによって、コマンドを速く便利に実行できます。
- 複数のタスクの同時実行。Enterprise Manager では複数のウィンドウを同時にオープンできるので、複数の管理タスクおよび非管理タスクを同時に実行できます。
- 複数のデータベースの管理。Enterprise Manager を使用すると、単一のデータベースを管理することも、複数のデータベースを同時に管理することもできます。
- データベース管理タスクの集中化。世界中のあらゆる Oracle プラットフォームで稼働中のローカル・データベースおよびリモート・データベースの両方を管理できます。さらに、これら Oracle プラットフォームは、Net8 がサポートするあらゆるネットワーク・プロトコルを使用して接続できます。
- SQL、PL/SQL および Enterprise Manager のコマンドを動的に実行します。Enterprise Manager を使用して文を入力、編集および実行できます。また、Enterprise Manager は、実行された文の履歴を保守します。

  したがって、入力し直さなくても文を再実行できます。分散システム内で長い文を繰り返し実行する必要がある場合には、この機能が特に有用です。

- グラフィカル・ユーザー・インタフェースが使用不能な場合や不要な場合は、Enterprise Manager のライン・モード・インタフェースを使用して管理タスクを実行できます。

## サードパーティの管理ツール

現在、60 を超える企業が、Oracle のデータベースとネットワークの管理の助けとなる 150 以上もの製品を生産しており、真にオープンな環境を提供しています。

## SNMP サポート

ネットワーク管理機能とは別に、「シンプル・ネットワーク管理プロトコル」（SNMP）のサポートによって、SNMP ベースのネットワーク管理システムによる Oracle サーバーへの検索と問合せができます。SNMP は標準的に広く受け入れられており、次のような普及度の高い多くのネットワーク管理システムで採用されています。

- HP 社の OpenView
- Digital 社の POLYCENTER Manager（NetView 上）
- IBM 社の NetView/6000
- Novell 社の NetWare Management System
- SunSoft 社の SunNet Manager

**追加情報：**『Oracle SNMP サポート・リファレンス・ガイド』を参照してください。

# 各国語サポート

Oracle は、クライアントとサーバーが異なる文字セットを使用するクライアント / サーバー環境をサポートします。クライアントが使用する文字セットは、そのクライアント・セッションの NLS_LANG パラメータの値により定義されます。サーバーが使用する文字セットは、そのサーバーのデータベースの文字セットです。この 2 つの文字セットが異なる場合、文字セット間でのデータ変換が自動的に行われます。各国語サポート機能の詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

# 2

# 分散データベース管理

この章では、分散データベースをインプリメントまたはメンテナンスするデータベース管理者（DBA）にとって重要な事項について説明します。

この章で説明する項目は次のとおりです。

- グローバル・データベース名とグローバル・オブジェクト名
- データベース・リンクのタイプ
- 位置の透過性についてのテクニック
- 文の透過性

# グローバル・データベース名とグローバル・オブジェクト名

1 つの分散システム内では、各データベースがそれぞれ一意のグローバル名を持つ必要があります。「グローバル・データベース名」により、システムの各データベースが識別されます。グローバル・データベース名は 2 つのコンポーネントで構成されます。8 文字以下のデータベース名（SALES など）とそのデータベースが所属するドメイン名（後述）です。

グローバル・データベース名のドメイン名コンポーネントは、インターネットの標準規則に従っていることが必要です。ドメイン名の中の各レベルはドットで区切られ、ドメイン名の順序は、左から右に、リーフ（葉）からルート（根）への順にする必要があります。データベース名とドメイン名は、初期化パラメータ DB_NAME および DB_DOMAIN により判断されます。これらの初期化パラメータの指定の詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

データベース・リンクには、そのデータベース・リンクが参照するリモート・データベースのグローバル・データベース名と同じ名前を付ける必要があります。初期化パラメータの GLOBAL_NAMES を TRUE に設定していると、データベース・リンク名はリモート・データベースのデータベース名と同じになります。初期化パラメータ GLOBAL_NAMES の指定の詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

**注意：** 初期化パラメータ GLOBAL_NAMES を FALSE に設定した場合、グローバル名を使用する必要はありません。ただし、Oracle アドバンスト・レプリケーションを含む各種の有効な機能ではグローバル名が必要であるため、グローバル名の使用を強くお薦めします。

グローバル名を使用可能にすると、データベース・リンク名は、そのリンクが指すデータベースのグローバル名と同じであるため、分散データベースのユーザーにとってデータベース・リンクは、事実上透過的なものとなります。たとえば、次の文は、ローカル・データベース内にデータベース・リンクを作成します。

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com ... ;
```

Oracle では、次の命名計画を使用し、スキーマ・オブジェクトのグローバルな命名にグローバル・データベース名を使用します。

```
<schema>.<schema_object>@<global_database_name>
```

この場合

| | |
|---|---|
| <schema> | スキーマは、データの論理構造、つまりスキーマ・オブジェクトの集合です。スキーマは、データベース・ユーザーが所有し、そのユーザーと同じ名前を持ちます。各ユーザーは単一のスキーマを所有します。 |
| <schema_object> | スキーマ・オブジェクトは、表、ビュー、シノニム、プロシージャ、パッケージまたはデータベース・リンクのような論理データの構造体です。 |

| | |
|---|---|
| <global_database_name> | リモート・データベースを個別に識別する名前。この名前は、リモート・データベースの初期化パラメータ DB_NAME と DB_DOMAIN を連結したものと同じである必要があります。 |

たとえば、事前に定義したデータベース・リンクを使用し、ユーザーまたはアプリケーションはグローバル・オブジェクト名を使用してリモート・データを参照できます。

```
SELECT * FROM scott.emp@sales.division3.acme.com;
```

# データベース・リンクのタイプ

分散システム全体で、アプリケーションによるデータおよびスキーマ・オブジェクトへのアクセスをサポートするために、管理者は必要なデータベース・リンクをすべて作成する必要があります。次の項では、Oracle が提供するデータベース・リンクの各タイプを比較します。

## プライベート、パブリックおよびグローバル・データベース・リンク

Oracle では、プライベート、パブリックおよびグローバル・データベース・リンクを作成できます。

| | |
|---|---|
| プライベート・データベース・リンク | データベースの特定のスキーマで、「プライベート・データベース・リンク」を作成できます。プライベート・データベース・リンクの 所有者またはスキーマの PL/SQL サブプログラムのみが、プライベート・データベース・リンクを使用して、対応するリモート・データベースにあるデータおよびデータベース・オブジェクトにアクセスできます。 |
| パブリック・データベース・リンク | データベースに「パブリック・データベース・リンク」を作成できます。すべてのユーザーおよびデータベース内の PL/SQL サブプログラムは、パブリック・データベース・リンクを使用して、対応するリモート・データベースにあるデータとデータベース・オブジェクトにアクセスできます。 |
| グローバル・データベース・リンク | Oracle ネットワークが Oracle Names を使用する場合は、システムにあるネーム・サーバーがネットワーク上の全 Oracle データベースに対して「グローバル・データベース・リンク」を自動的に作成し、管理します。あらゆるデータベースのユーザーおよび PL/SQL サブプログラムは、グローバル・データベース・リンクを使用して、対応するリモート・データベースにあるデータおよびデータベース・オブジェクトにアクセスできます。 |

分散データベースで採用するデータベース・リンクのタイプは、そのシステムを使用するアプリケーションの仕様要件によって判断されます。

データベース・リンクの各タイプを使用する上での利点と欠点を考慮してください。

- 「プライベート・データベース・リンク」の場合、それを使用して特定のリモート・データベースにアクセスできるのは、そのプライベート・リンク所有者または同一スキーマ内のサブプログラムのみであるため、パブリックまたはグローバル・リンクよりも高い安全性があります。
- 多数のユーザーがリモート Oracle データベースへのアクセス・パスを必要とする場合、管理者は「パブリック・データベース・リンク」を 1 つ作成し、データベースの全ユーザーに対応させることができます。
- Oracle ネットワークが Oracle Names を使用している場合、管理者はシステムにあるデータベースすべての「グローバル・データベース・リンク」を容易に管理できます。データベース・リンクの管理は、集中化されていて簡単です。

### プライベート・データベース・リンクの作成

プライベート・データベース・リンクを作成するには、次のように指定します。

```
CREATE DATABASE LINK ...;
```

詳細は、『Oracle8*i* SQL リファレンス』および以降の各項を参照してください。

### パブリック・データベース・リンクの作成

パブリック・データベース・リンクを作成するには、PUBLIC キーワードを使用します。

```
CREATE PUBLIC DATABASE LINK ...;
```

詳細は、『Oracle8*i* SQL リファレンス』および以降の各項を参照してください。

### グローバル・データベース・リンクの作成

「グローバル・データベース・リンク」は、Oracle Name Server で定義する必要があります。詳細は、『Oracle8*i* Net8 管理者ガイド』を参照してください。

## データベース・リンクのセキュリティ・オプション

データベース・リンクによって、あるデータベースから別のデータベースへの通信パスが定義されます。アプリケーションがデータベース・リンクを使用して、リモート・データベースにアクセスすると、そのローカル・アプリケーションの要求のために、リモート・データベース内でデータベース・セッションが確立されます。

プライベートまたはパブリック・データベース・リンクの作成時、そのリンクが接続するリモート・データベース内のスキーマを、固定ユーザー、現ユーザーまたは接続ユーザーのどれにするか決定できます。

## 固定ユーザー・データベース・リンク

「固定ユーザー・データベース・リンク」を作成するには、リモート・データベースへのアクセスに必要な資格証明（この場合は、ユーザー名とパスワード）をリンクの定義に組み込みます。

```
CREATE DATABASE LINK ... CONNECT TO username IDENTIFIED BY password ...;
```

アプリケーションが固定ユーザー・データベース・リンクを使用する場合、ローカル・サーバーは、常にリモート・データベースにある固定リモート・スキーマへ接続を確立します。また、アプリケーションがリモート・データベースにアクセスするためにリンクを使用する場合、ローカル・サーバーは、ネットワークを介してユーザーの資格証明を送信します。固定ユーザー・データベース・リンクを使用する分散データベースを、非保護ネットワークでサポートする場合は、サーバー間接続のログイン・パケットを暗号化することを考慮してください。

## 接続ユーザーと現ユーザーのデータベース・リンク

接続ユーザーと現ユーザーのデータベース・リンクでは、リンクの定義に資格証明を含みません。リモート・データベースへの接続に使用する資格証明は、データベース・リンクを参照するユーザー、およびアプリケーションで実行される操作に応じて変更できます。2 つのタイプのデータベース・リンクの違いを理解するには、最初に、接続ユーザーと現ユーザーの概念を理解する必要があります。

- 「接続ユーザー」は、データベース・アプリケーションを使用してデータベースに接続するユーザーです。たとえば、SQL*Plus を起動し、Oracle データベースに SCOTT として接続すると、接続ユーザーは SCOTT となります。
- 「現ユーザー」は、データベース操作を実行するセキュリティ・コンテキストによって判断されます。たとえば、Oracle データベースにユーザー SCOTT として接続して、プロシージャ SALES.DEL_EMP を実行する場合、プロシージャ DEL_EMP を実行している間の現ユーザーは、デフォルトで SALES になります。ストアド・プロシージャはその所有者のセキュリティ・コンテキスト内で実行されるためです。

接続ユーザーと現ユーザーのデータベース・リンクは、これらユーザーのタイプの違いを理解して応用すれば区別できます。「接続ユーザーのデータベース・リンク」の場合、リモート・データベースで実行されている操作は、常に「ローカル・データベースの接続ユーザー」のセキュリティ・コンテキスト内で発生します。

「現ユーザーのデータベース・リンク」の場合、リモート・データベースで実行されている操作は、常に「ローカル・データベースの現ユーザー」のセキュリティ・コンテキスト内で発生します。

たとえば、ユーザー SCOTT がプロシージャ SALES.DEL_EMP をコールし、そのプロシージャがリモート・データベースから社員レコードを削除する場合について考えます。そのプロシージャが接続ユーザーのデータベース・リンクを使用してリモート・データベースにアクセスする場合、リモート・データベースにある社員レコードの削除は、ローカル・データベース内の接続ユーザーである SCOTT として行われます。ただし、そのプロシージャが現ユーザーのデータベース・リンクを使用してリモート・データベースにアクセスする場合、リモート・データベースにある社員レコードの削除は、ローカル・データベースの現ユーザー SALES として行われます。

接続ユーザーのデータベース・リンクを作成するには、単に CONNECT TO 句を省略します。次は、接続ユーザーのデータベース・リンク作成例です。

```
CREATE DATABASE LINK sales.division3.acme.com USING 'sales';
```

現ユーザーのデータベース・リンクを作成する構文は、次のとおりです。

```
CREATE DATABASE LINK ... CONNECT TO CURRENT_USER ...;
```

現ユーザーのデータベース・リンクを使用するには、その現ユーザーは、Oracle Security Server で認証されたグローバル・ユーザーである必要があります。

---

**注意：** Oracle8 で使用可能だったグローバル・ユーザーの機能は修正中であり、現在はベータ版のユーザーのみ使用可能です。Oracle8*i* では、今後のリリースに組み込まれる予定です。

---

データベース・リンク作成の構文の詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

## 共有データベース・リンク

標準のデータベース・リンクを使用してリモート・サーバーを参照するあらゆるアプリケーションは、ローカル・データベースとリモート・データベース間の接続を確立します。多数のユーザーがアプリケーションを同時に実行すると、ローカル・データベースとリモート・データベース間の接続数を増大させる結果となります。

「共有データベース・リンク」を使用すると、ローカル・サーバーとリモート・サーバー間に必要なネットワーク接続数を制限できます。共有データベース・リンクを使用するには、ローカル・サーバーをマルチスレッド・サーバー（MTS）モードで実行する必要があります。リモート・サーバーの実行モードは、マルチスレッド・サーバー・モードでも、専用サーバー・モードでもかまいません。

**注意：**共有データベース・リンクにより、ローカル・サーバーとリモート・サーバー間に必要な接続数は減少しますが、使用方法を誤ると、単に標準の（共有でない）データベース・リンクを使用した場合よりも、物理的な接続とプロセスの数が増大します。

共有サーバーで共有データベース・リンクを使用するシステムをインプリメントする前に、この項に記載されている情報を十分に理解してください。

## 共有データベース・リンクのプロパティ

共有データベース・リンクは、次の 2 つの点で標準のデータベース・リンクとは異なります。

- 共有データベース・リンク用に作成されたネットワーク接続は、同一のデータベース・リンクのスキーマ・オブジェクトを使用するユーザーとの間で共有できます。ユーザーが特定の共有サーバー・プロセスからあるリモート・サーバーへの接続を必要とする場合、共有プロセスは、そのリモート・サーバーにすでに確立されている接続を再使用できます（その接続が同一のデータベース・リンクで同じ共有サーバー上に確立されていた場合）。
- 共有データベース・リンクを使用すると、ネットワーク接続はローカル・サーバーにある共有サーバーから直接確立されます。標準の（非共有）データベース・リンクでは、ローカル・サーバーがマルチスレッド・サーバーの場合、この接続はローカル・ディスパッチャを介して行われ、そのローカル・ディスパッチャ用のコンテキスト・スイッチが必要であり、データはディスパッチャを通過する必要があります。

## 共有データベース・リンクを使用するべき場合

アプリケーションとマルチスレッド・サーバーの構成を慎重に検討して、共有リンクを使用するかどうかを判断してください。

たとえば、標準のパブリック・データベース・リンクを使用するアプリケーションを設計した後で、100 人のユーザーから同時に接続を要求されると、100 の直接ネットワーク接続が必要になります。

ただし、アプリケーションで共有データベース・リンクを使用し、ローカルの MTS モード・データベースに共有サーバーが 10 あれば、同一の（共有パブリック）データベース・リンクを使用する 100 人のユーザーに必要なリモート・サーバーへのネットワーク接続数は 10（またはそれ以下）になります。ローカルの各共有サーバーが必要とするリモート・サーバーへの接続数は 1 です。

## 共有データベース・リンクを使用するべきではない場合

共有データベース・リンクは、どのような状況でも有効なわけではありません。たとえば、リモート・サーバーにアクセスするユーザーが 1 ユーザーのみであるとします。そのユーザーが共有データベース・リンクを定義しており、ローカル・データベースには共有サーバーが 10 あれば、1 人のユーザーはリモート・サーバーへのネットワーク接続を最大 10 まで要求する可能性があります。各共有サーバーがそのユーザーに使用されている可能性があるため、すべての共有サーバーがリモート・サーバーへの接続を確立している場合があります。

標準データベース・リンクの場合、必要（可能）なネットワーク接続は1つで済むため、この状況では明らかにが望ましいと言えます。レッスン：単一ユーザーを想定したシナリオでは、共有データベース・リンクを使用すると、ネットワーク接続数が増加します。したがって、同一のデータベース・リンクを必要とするユーザー数が多いと予想される場合にのみ、共有データベース・リンクを使用してください。一般的に、このケースではパブリック・データベース・リンクですが、同一のローカル・スキーマ（したがって、同じプライベート・データベース・リンク）を使用するクライアント数が多いと予想される場合は、プライベート・データベース・リンクについても、同じことが当てはまります。

データベース・リンクにアクセスするユーザー数が、ローカル・データベースのサーバー数を大幅に上回る場合は、共有データベース・リンクを使用するという経験則が成立します。

## 共有データベース・リンクの設定

共有データベース・リンクを作成するには、SQL の CREATE DATABASE LINK コマンドで SHARED キーワードを使用します。

```
CREATE SHARED DATABASE LINK dblink_name
[CONNECT TO username IDENTIFIED BY password] | [CONNECT TO CURRENT_USER]
AUTHENTICATED BY schema_name IDENTIFIED BY password
[USING 'service_name'];
```

構文の詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

SHARED キーワードを使用するときには、AUTHENTICATED BY 句も必要です。リモート・データベース には、指定された USERID/PASSWORD のアカウントおよび CREATE SESSION 権限が必要です。それ以外の権限は必要ありません。

AUTHENTICATED BY 句で指定されたスキーマは、セキュリティ上の理由でのみ使用され、「ダミー」スキーマと見なすことができます。このスキーマは、共有データベース・リンクの使用時に影響を受けず、共有データベース・リンクのユーザーに影響を与えません。認証されていないクライアントがデータベース・リンク・ユーザーを装って、権限を持たない情報にアクセスするのを防止するには、AUTHENTICATED BY 句が必要です。

## 共有データベース・リンクの構成

共有データベース・リンクは、2種類の構成で使用できます。

**専用サーバーへの共有データベース・リンク** 第1番目の構成は、ローカル・サーバーの共有サーバーがリモートの専用サーバーを所有し、直接的なネットワーク接続がその共有サーバーとリモートの専用サーバー間に存在する構成です。この構成の利点は、ローカル共有サーバーとリモート専用サーバー間で直接的なネットワーク転送を実現できることです。こ

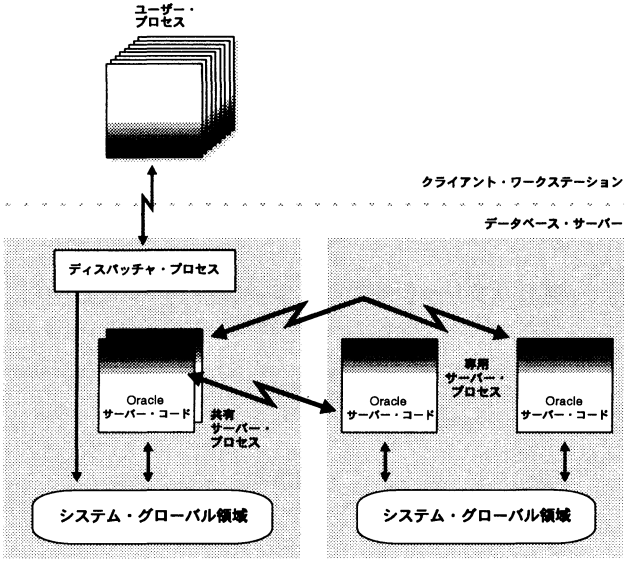
の構成の欠点は、多数のバックエンド・サーバーが必要になることです。図 2-1 を参照してください。

---

**注意：** リモート・サーバーは、マルチスレッド・サーバーまたは専用サーバーとして構成できます。ローカル・サーバーとリモート・サーバー間の接続には、専用接続を使用します。リモート・サーバーがマルチスレッド・サーバーとして構成される場合は、サービス名の定義で（SERVER=DEDICATED）句を使用してこの構成を指定することにより、専用サーバーの接続を強制できます。

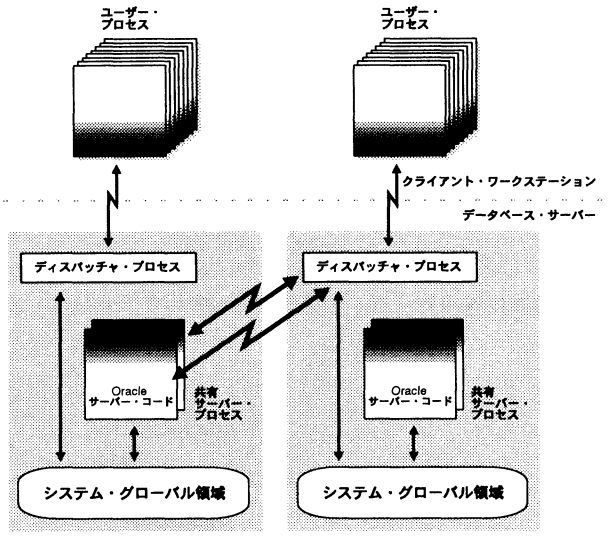
---

**図 2-1 専用サーバー・プロセスへの共有データベース・リンク**




**マルチスレッド・サーバーへの共有データベース・リンク** 第 2 番目は、リモート・サーバーで共有サーバーを使用する構成です。この構成では、必要となる専用サーバーの数は増加しませんが、リモート・サーバー上のディスパッチャを通過する必要があります。図 2-2 を参照してください。ローカル・サーバーとリモート・サーバーの両方を、マルチスレッド・サーバーとして構成する必要がある点に注意してください。

**図 2-2 マルチスレッド・サーバーへの共有データベース ・リンク**




## 例

**例 1: パブリック固定ユーザーのデータベース・リンク** 次の文は、パブリック固定ユーザーのデータベース・リンクを作成します。

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO SCOTT IDENTIFIED BY TIGER
USING 'sales';
```

ローカル・データベースに接続したどのユーザーも、データベース・リンク SALES.DIVISION3.ACME.COM を使用してリモート・データベースに接続できます。各ユーザーは、リモート・データベースの同じリモート・スキーマ SCOTT に接続されます。SCOTT のリモート・スキーマにある EMP 表にアクセスするには、SQL 問合せを発行できます。

```
SELECT * FROM emp@sales.division3.acme.com;
```

各アプリケーションまたはユーザー・セッションは、サーバー上の共通アカウントへの接続をそれぞれ別個に作成することに注意してください。アプリケーションまたはユーザーのセッション中は、リモート・データベースへの接続はオープンのままです。

**例 2: パブリック固定ユーザーの共有データベース・リンク** 次の文は、パブリック固定ユーザーの共有データベース・リンクを作成する例です。

```
CREATE SHARED PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO scott IDENTIFIED BY tiger
AUTHENTICATED BY scott IDENTIFIED BY tiger
USING 'sales';
```

> **注意：** ローカル・データベースは、マルチスレッド・サーバー・モードで構成する必要があります。

ローカル MTS モードのサーバーに接続したユーザーは、このデータベース・リンクを使用して、リモート・データベース SALES に接続し（共有サーバー・プロセスを介して）、SCOTT スキーマにある表を問合せできます。

前述の例では、各ローカル共有サーバーはリモート・サーバーへの接続を 1 つ確立できます。ローカル共有サーバー・プロセスが、データベース・リンク SALES.DIVISION3.ACME.COM を介してリモート・サーバーにアクセスする必要がある場合、ローカル共有サーバー・プロセスは確立済みネットワーク接続を再使用します。

**例 3: パブリック接続ユーザーのデータベース・リンク** 次の文は、パブリック接続ユーザーのデータベース・リンクを作成します。

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
USING 'sales';
```

ローカル・データベースに接続したどのユーザーも、データベース・リンク SALES.DIVISION3.ACME.COM を使用できます。ローカル・データベースでデータベース・リンクを使用する接続ユーザーが、リモート・スキーマを決定します。SCOTT が接続ユーザーで、データベース・リンクを使用する場合は、データベース・リンクの接続先は SCOTT のリモート・スキーマです。FORD が接続ユーザーで、データベース・リンクを使用する場合は、データベース・リンクの接続先は FORD のリモート・スキーマです。

次の文で、リモート・スキーマ FORD がスキーマ・オブジェクト EMP を解決できない場合は、ローカル・データベースのユーザー FORD にはエラーが発生します。つまり、SALES.DIVISION3.ACME.COM のスキーマ FORD に、表、ビューまたは（パブリック）シノニムである EMP がないと、エラーが戻されます。

```
SELECT * FROM emp@sales.division3.acme.com;
```

**例 4: パブリック接続ユーザーの共有データベース・リンク** 次の文は、パブリック接続ユーザーの共有データベース・リンクを作成します。

```
CREATE SHARED PUBLIC DATABASE LINK sales.division3.acme.com AUTHENTICATED
BY ward IDENTIFIED BY orange
USING 'sales';
```

**注意：** ローカル・データベース・サーバーは、マルチスレッド・サーバー・モードで構成する必要があります。

ローカル・サーバーに接続したどのユーザーも、この共有データベース・リンクを使用してリモート・データベースに接続し、対応するリモート・スキーマで表を問合せできます。

前述の例では、各ローカル共有サーバーはリモート・サーバーへの接続を 1 つ確立します。ローカル共有サーバー・プロセスが、データベース・リンク SALES.DIVISION3.ACME.COM を介してリモート・サーバーにアクセスする必要がある場合、その接続ユーザーが別のユーザーでも、ローカル共有サーバー・プロセスは確立済みネットワーク接続を再使用します。

このデータベース・リンクが頻繁に使用されると、最終的にローカル・データベースのすべての共有サーバーにリモート接続が確立されます。その時点で、新しいユーザーがこの共有データベース・リンクを使用するときでも、リモート・サーバーへの物理的な接続はそれ以上増加しなくなります。

**例 5: パブリック現ユーザーのデータベース・リンク** 次の文は、パブリック現ユーザーのデータベース・リンクを作成します。

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO CURRENT_USER
USING 'sales';
```

**注意：** このデータベース・リンクを使用するには、現ユーザーをグローバル・ユーザーにする必要があります（グローバル・ユーザーには、Oracle Security Server による認証が必要です）。

**注意：** Oracle8 で使用可能だったグローバル・ユーザーの機能は修正中であり、現在はベータ版のユーザーのみ使用可能です。Oracle8*i* では、今後のリリースに組み込まれる予定です。

SCOTT は、リモート表 EMP から行を削除するローカル・プロシージャ FIRE_EMP を作成し、FORD に実行権限を付与します。

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
   DELETE FROM emp@sales.division3.acme.com
   WHERE empno=enum;
END;

GRANT EXECUTE ON FIRE_EMP TO FORD;
```

FORD がプロシージャ SCOTT.FIRE_EMP を実行すると、プロシージャは SCOTT の権限で実行されます。現ユーザーのデータベース・リンクが使用されたため、接続は SCOTT のリモート・スキーマに確立されます。(かわりに接続ユーザーのデータベース・リンクが使用された場合は、接続が FORD のリモート・スキーマに確立されます。) この場合、SCOTT は必ずグローバル・ユーザーですが、FORD はグローバル・ユーザーであってもグローバル・ユーザーでなくてもかまわない点に注意してください。

---

**注意：** Oracle8 で使用可能だったグローバル・ユーザーの機能は修正中であり、現在はベータ版のユーザーのみ使用可能です。Oracle8*i* では、今後のリリースに組み込まれる予定です。

---

SCOTT のリモート・スキーマに接続する固定ユーザーのデータベース・リンクを使用しても同様の実行ができる点に注意してください。ただし、固定ユーザーのデータベース・リンクでは、SCOTT のユーザー名とパスワードはデータベース内で DBA が読込み可能な形式なので、セキュリティが低下する可能性があります。

## 接続修飾子

状況によっては、異なる通信経路で接続しながら、同じリモート・データベースを指す同一タイプ（パブリックなど）の各種データベース・リンクが必要になる場合があります。たとえば、リモート・データベースが Oracle Parallel Server を使用中のときに、リモート・データベースの特定のインスタンスに接続を確立できるようにするために、ローカル・ノードでいくつかのパブリック・データベース・リンクを定義する場合などです。

そのような機能を手助けするために、Oracle ではデータベース・リンク名にオプションの接続修飾子を使用してデータベース・リンクを作成できます。データベース・リンクを作成するとき、接続修飾子は、符号（"@"）で区切ったデータベース・リンク名の末尾の部分に指定します。たとえば、リモート・データベース HQ.ACME.COM が Oracle Parallel Server により管理される 場合を仮定します。データベース HQ には、HQ_1 と HQ_2 という名前の 2 つのインスタンスがあります。ローカル・データベースには、データベース HQ のリモート・インスタンスへの経路を定義する次のパブリック・データベース・リンクを含めることができます。

```
CREATE PUBLIC DATABASE LINK hq.acme.com@hq_1
       USING 'string_to_hq_1';

CREATE PUBLIC DATABASE LINK hq.acme.com@hq_2
       USING 'string_to_hq_2';

CREATE PUBLIC DATABASE LINK hq.acme.com
      USING 'string_to_hq';
```

前述の例では、接続修飾子が単にデータベース・リンク名の拡張子になっていることに注意してください。接続修飾子は、必ずしも接続の確立方法を指定するものではありません。この情報は、USING 句のサービス名で指定されます。3 番目の例では、接続修飾子は指定されていません。この場合、接続修飾子が指定されると、インスタンスが USING 文字列によって決定されます。

接続修飾子を使用して特定のインスタンスを指定するには、グローバル・オブジェクト名の最後に修飾子を挿入します。

```
SELECT * FROM scott.emp@hq.acme.com@hq_1
```

# データベース・リンクの解決

SQL 文にグローバル・オブジェクト名への参照が含まれていると、グローバル・オブジェクト名に指定されたデータベース名と一致する名前のデータベース・リンクが検索されます。この検索は、指定されたリモート・データベースへのパスを確認するために実行されます。

常に次の順序で、データベース・リンクの一致が検索されます。

1. SQL 文を発行したユーザーのスキーマにあるプライベート・データベース・リンク
2. ローカル・データベースのパブリック・データベース・リンク
3. グローバル・データベース・リンク（Oracle Name Server が使用可能な場合のみ）

SQL 文で完全なグローバル・データベース名を指定すると、つまり、データベースとドメイン・コンポーネントの両方が指定されると、明確に指定されたグローバル・データベース名と一致するデータベース・リンク（プライベート、パブリックおよびグローバル）のみが検索されます。ドメインの一部が指定されたときは、完全なグローバル・データベース名が指定されたとみなされます。しかし、SQL 文でグローバル・データベース名を部分的に指定すると、つまり、データベース・コンポーネントのみが指定されると、完全なグローバル・データベース名を形成するために、データベース名にローカル・データベースのネットワーク・ドメイン・コンポーネントが追加されます。そして、組み立てられたグローバル・データベース名と一致するデータベース・リンク（プライベート、パブリックおよびネットワーク）のみが検索されます。一致するデータベース・リンクが見つからなかった場合は、エラーが戻され、SQL 文は実行できません。

**最適化 :** グローバル・オブジェクト名がローカル・データベース内のオブジェクトを参照し、接続修飾子が指定されていない場合は、Oracle は自動的にそのオブジェクトがローカルであることを検出し、オブジェクト参照を解決するためにデータベース・リンクを検索および使用しません。

接続修飾子が指定されているかどうかに関わらず、グローバル・オブジェクト参照は拡張されます。接続修飾子が指定されている場合は、オブジェクト参照を解決するために（接続修飾子も含めて）一致するデータベース・リンクのみが使用されます。

最初の一致が検出されたときに、データベース・リンクの検索を停止するわけではありません。リモート・データベース（リモート・アカウントとサービス名の両方）への完全なパスが確認されるまで、プライベート、パブリックおよびネットワーク・データベース・リンクを検索する必要があります。

最初の一致ではリモート・スキーマが判別されます。つまり、CONNECT 句が未指定の場合は、接続ユーザーのデータベース・リンクとして使用されます。"CONNECT TO username IDENTIFIED BY password" 句が指定された場合は、固定ユーザーのデータベース・リンクが使用されます。" CONNECT TO CURRENT_USER" 句が指定された場合は、現ユーザーのデータベース・リンクとして使用されます。

最初の一致で USING 句が指定されていない場合は、データベース文字列を指定するリンクが見つかるまで、検索が継続されます。一致するデータベース・リンクが検出されても、データベース文字列が認識できない場合は、エラーが戻ります。

完全なパスが確認されると、同じローカル・セッションのための同一の接続はまだオープンしていないとみなされ、リモート・セッションが作成されます。

# スキーマ・オブジェクト名の解決

ローカル Oracle データベースが、指定したリモート・データベースに接続されると、SQL 文を発行したローカル・ユーザーにかわって、リモート・ユーザーが対応した SQL 文を発行したかのように、オブジェクト解決が継続されます。つまり、固定ユーザーのデータベース・リンクが使用される場合は、指定されたスキーマで、接続ユーザーのデータベース・リンクが使用される場合は、接続ユーザーのリモート・スキーマ（シノニムも含む）で、さらに現ユーザーのデータベース・リンクが使用される場合は、現ユーザーのリモート・スキーマで、それぞれオブジェクト解決が進められます。オブジェクトが見つからない場合は、リモート・データベースのパブリック・オブジェクトが検査されます。

オブジェクトが解決されない場合は、確立したリモート・セッションはそのままですが、SQL 文は実行できません。

## オブジェクト名解決の例

次の例は、分散システムにあるグローバル・オブジェクト名の解決です。

次の各例では、リモート・データベースは SALES.DIVISION3.ACME.COM 、ローカル・データベースは HQ.DIVISION3.ACME.COM、Oracle Name Server は（したがって、グローバル・データベース・リンクも）使用可能ではないと想定しています。

**例 1** この例は、プライベート・データベース・リンクとパブリック・データベース・リンクを使用して、完全なグローバル・オブジェクト名を解決する方法、およびリモート・データベースへの適切なパスを確認する方法を示しています。

たとえば、リモート表 EMP はスキーマ TSMITH に含まれていると仮定します。

ローカル・データベースで発行された次の文について考えます。

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
        CONNECT TO guest IDENTIFIED BY network
        USING 'dbstring';

CONNECT jward/bronco;

CREATE DATABASE LINK sales.division3.acme.com
       CONNECT TO tsmith IDENTIFIED BY radio;

UPDATE tsmith.emp@sales.division3.acme.com
        SET deptno = 40
       WHERE deptno = 10;
```

Oracle は、完全なグローバル・オブジェクト名が JWARD の UPDATE 文で参照されることを認識します。したがって、ローカル・データベースで、一致する名前のデータベース・リンクの検索が開始されます。スキーマ JWARD で一致するプライベート・データベース・リンクが検出されます。ただし、プライベート・データベース・リンク JWARD.SALES.DIVISION3.ACME.COM では、リモート・アカウントのみが示され、リモート・データベース SALES への完全なパスは示されません。そこで、Oracle は一致するパブリック・データベース・リンクを検索し、検出します。Oracle は、このパブリック・データベース・リンクからサービス名を取得します。一致したプライベート固定ユーザーのデータベース・リンクから取得したリモート・アカウントと結合して、完全なパスが確認され、ユーザー TSMITH/RADIO でリモート・データベース SALES への接続の確立に進みます。

リモート・データベースは、この時点で EMP 表へのオブジェクト参照を解決できます。Oracle は、指定したスキーマ TSMITH で検索し、参照する表 EMP を検出します。これ以上の解決は不要です。リモート・データベースは文の実行を完了し、ローカル・データベースに結果が戻されます。

**例 2** この例は、プライベート・データベース・リンクとパブリック・データベース・リンクを使用して、部分的なグローバル・オブジェクト名を解決し、リモート ・データベースへの適切なパスを確認する方法を示しています。

リモート表 EMP はスキーマ TSMITH に含まれ、EMP という名前のリモート・パブリック・シノニムは前述の EMP 表を指すと仮定します。また、例 1 でパブリック・データベース・リンクが作成されているとします。

ローカル・データベースで発行された次の文について考えます。

```
CONNECT scott/tiger;

CREATE DATABASE LINK sales.division3.acme.com;

DELETE FROM emp@sales
        WHERE empno = 4299;
```

Oracle は、部分的なグローバル・オブジェクト名が SCOTT の DELETE 文で参照されていることを認識します。最初に、ローカル・データベースのドメイン名を使用して、グローバル・オブジェクト名が完全なグローバル・オブジェクト名に拡張され、その結果次の文になります。

```
DELETE FROM emp@sales.division3.acme.com
        WHERE empno = 4299;
```

次に、ローカル・データベースで、一致する名前を使用してデータベース・リンクの検索が開始されます。スキーマ SCOTT で、一致するプライベート接続ユーザーのデータベース・リンクが検出されます。ただし、プライベート・データベース・リンクでは、パスが示されません。Oracle は、接続したユーザー名 / パスワードをパスのリモート・アカウント部分として使用して、一致するパブリック・データベース・リンクを検索し、検出します。Oracle は、パブリック・データベース・リンクからデータベース文字列を取得します。この時点で、完全なパスが確認され、SCOTT/TIGER としてリモート・データベースに接続できます。

SCOTT としてリモート・データベースに接続すると、EMP への参照が解決されます。最初に、スキーマ SCOTT でオブジェクト名 EMP を検索しますが、検出しません。

次に、リモート・データベースでは EMP という名前のパブリック・シノニムが検索され、検出されます。続いて、リモート・データベースでは文の実行を完了し、ローカル・データベースに結果が戻されます。

## ビュー、シノニム、プロシージャおよびグロ ーバル名の解決

リモート・スキーマ・オブジェクトは、ビュー、シノニムまたは PL/SQL プログラム・ユニット（プロシージャ、トリガーなど）の定義の中のグローバル・オブジェクト名により参照できます。完全なグローバル・オブジェクト名が、ビュー、シノニムまたはプログラム・ユニットの定義で参照される場合は、参照されているグローバル・オブジェクト名を拡張しないで、指定されたとおりオブジェクトの定義が格納されます。ただし、部分的なグローバル・オブジェクト名（つまり、ドメイン名のないデータベース名のみ）が、ビュー、シノニムまたはプログラム・ユニットの定義で参照される場合は、ローカル・データベースのグローバル・データベース名のドメイン・コンポーネントを使用して、その部分的な名前が拡張される必要があります。

次のリストは、ビュー、シノニムおよびプログラム・ユニットに対する部分的なグローバル・オブジェクト名の拡張を完了する時点を説明します。

- ビューが作成されると、問合せの定義にある部分的なグローバル・オブジェクト名は拡張されず、定義している問合せのテキストがそのままデータ・ディクショナリに格納されます。かわりに、ビューを使用して文が解析されるたびに、部分的なグローバル・オブジェクト名が拡張されます。
- シノニムが作成されると、部分的なグローバル・オブジェクト名が拡張されます。シノニムの定義は、拡張したグローバル・オブジェクト名も含めてデータ・ディクショナリに格納されます。
- プログラム・ユニットがコンパイルされるたびに、部分的なグローバル・オブジェクト名が拡張されます。

部分的なグローバル・オブジェクト名を使用してリモート・データを参照するビュー、シノニムおよびプロシージャを作成するときには、前述の動作を考慮してください。ローカル・データベースのグローバル・データベース名が変更される場合（まれに発生します）、ビューおよびプロシージャは、グローバル・データベース名が変更される前にアクセスしたデータベースとは異なるデータベースを参照しようとします。一方、シノニムは実行時にデータベース・リンク名を拡張しないため、変更は生じません。状況によって、この動作は望ましい場合とそうでない場合があります。たとえば、SALES.UK.ACME.COM および HQ.UK.ACME.COM という 2 つのデータベースがあり、SALES データベースに次のビューとシノニムが含まれている場合を考えます。

```
CREATE VIEW employee_names AS
        SELECT ename FROM scott.emp@hq;

CREATE SYNONYM employee FOR scott.emp@hq;
```

シノニム EMPLOYEE の定義が拡張され、格納されます。

```
"scott.emp@hq.uk.acme.com"
```

会社が再編成されることになりました。最初に、営業部門と人事部門の両方が米国に移動する状況を考えます。その結果、対応するグローバル・データベース名が、それぞれ SALES.US.ACME.COM と HQ.US.ACME.COM に変更されます。この場合、EMPLOYEE_NAMES ビューの定義問合せも、このビューの使用時に正しいデータベースに拡張されます。

```
"SELECT ename FROM scott.emp@hq.us.acme.com"
```

ただし、シノニム EMPLOYEE の定義は、以前のデータベース名 HQ.UK.ACME.COM を引き続き参照します。

今度は、営業部門のみが米国に移動する場合を考えます。その結果、対応する新規グローバル・データベース名は SALES.US.ACME.COM となりますが、人事部部門のデータベースは HQ.UK.ACME.COM のままです。この場合、EMPLOYEE_NAMES ビューの定義問合せは、このビューの使用時には、存在しないグローバル・データベース名に拡張されます。

```
"SELECT ename FROM scott.emp@hq.us.acme.com"
```

かわりに、シノニム EMPLOYEE は、正しいデータベース HQ.UK.ACME.COM を引き続き参照します。

まとめると、ビュー、シノニムおよびプロシージャの定義について、どのようなときに部分的および完全なグローバル・オブジェクト名を使用するかを判断する必要があります。データベース名は固定的なものとし、ネットワーク内で無意味なデータベース移動をしないように配慮することが必要です。

## データベース・リンクの削除

表またはビューを削除できるのと同様に、データベース・リンクも削除できます。コマンド構文は次のとおりです。

```
DROP DATABASE LINK dblink;
```

たとえば、データベース・リンク NY_FIN を削除するには、コマンドを次のように入力します。

```
DROP DATABASE LINK ny_fin;
```

## 使用可能なデータベース・リンクのリスト

各データベースのデータ・ディクショナリには、そのデータベース内のすべてのデータベース・リンクの定義が格納されています。USER/ALL/DBA_DB_LINKS データ・ディクショナリ・ビューには、ローカル・データベースで定義されたデータベース・リンクが表示されます。

すべてのユーザーは、データ・ディクショナリに問い合せて、自分が使用可能なデータベース・リンクがどれかを調べることができます。データ・ディクショナリの参照方法の詳細は、『Oracle8*i* 概要』または『Oracle8*i* SQL リファレンス』を参照してください。

## アクティブ・データベース・リンクの数の制限

OPEN_LINKS 初期化パラメータを使用して、1 つのユーザー・プロセスからのリモート・データベースへの接続数を制限できます。このパラメータは、1 つのユーザー・セッションが 1 つの SQL 文中で同時に使用できるリモート接続の数を制御します。詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

# 位置の透過性についてのテクニック

分散システムのユーザーは、作業するデータベースの位置と機能を意識する必要はありません（多くの場合、意識すべきではありません）。次のセクションで示されるように、DBA とネットワーク管理者は、データベースの分散性をユーザーに対して透過的な状態に保持できます。

## ビューと位置の透過性

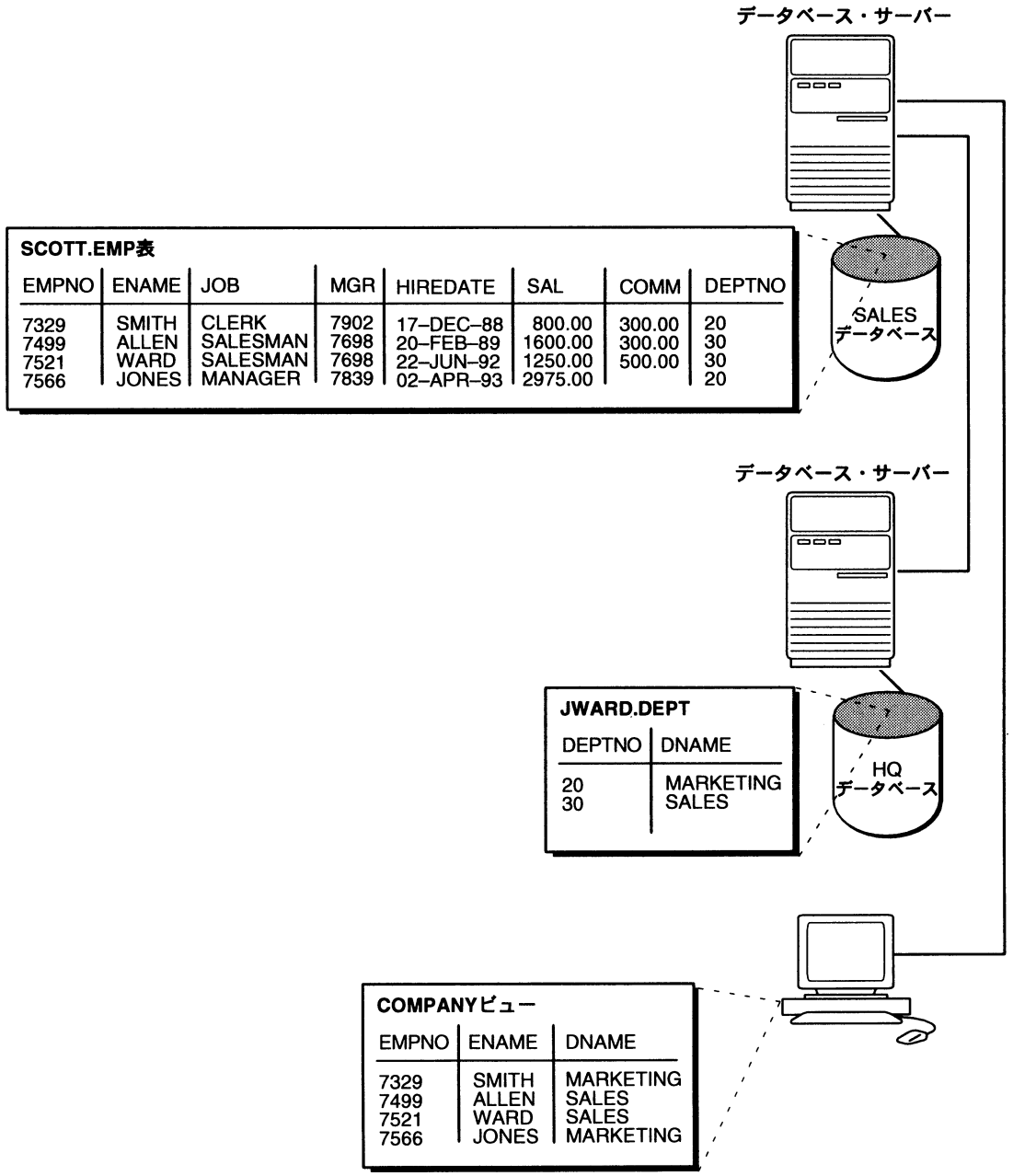
ローカル・ビューはローカル表およびリモート表に対する位置の透過性を提供します。

たとえば、EMP 表がローカル・データベースに格納されているとします。これとは別に、DEPT 表がリモート・データベースに格納されているとします。

これらの表の位置および関係をシステムのユーザーにとって透過的にするには、ローカル・サーバーとリモート・サーバーのデータを結合する COMPANY という名前のビューをローカル・データベースに作成します。

```
CREATE VIEW company AS
SELECT empno, ename, dname
FROM emp a, dept@hq.acme.com b
HERE a.deptno = b.deptno;
```

**図 2-3 ビューと位置の透過性**



ユーザーがこのビューにアクセスした場合、ユーザーはデータの物理的な格納場所や、複数の表のデータにアクセスしているかどうかはわからず、また知る必要もありません。したがって、必要な情報の入手が容易になります。次に例を示します。

```
SELECT * FROM company;
```

この文は、ローカル・データベース表とリモート・データベース表の両方からデータを取り出します。

図 2-3 は、この例の位置の透過性を示しています。

### ビューと権限

ローカル・ビューがリモート表またはリモート・ビューを参照するとします。このローカル・ビューの所有者が付与できるのは、自分のビューに関するオブジェクト権限のうち、リモート・ユーザーに付与されている権限のみです。(リモート・ユーザーはデータベース・リンクのタイプで暗黙指定されます)。これは、ローカル・データを参照するビューの権限管理に似ています。

## シノニムと位置の透過性

シノニムは、分散システム内の位置も含めたオブジェクトの基礎を形成する個別性を隠すため、分散環境でも非分散環境でもきわめて有効です。基礎となるオブジェクトの改名または移動が必要な場合は、シノニムを再定義するだけですみます。シノニムに基づくアプリケーションは、変更せずにそのまま使用できます。シノニムは、分散システムのユーザーに対する SQL 文も単純化できます。

シノニムは、任意の表、型、ビュー、スナップショット、シーケンス、プロシージャ、ファンクションまたはパッケージに対して作成できます。すべてのシノニムは、それを作成したデータベースのデータ・ディクショナリに格納されます。データベース・リンクを介したリモート表へのアクセスを簡素化するために、シノニムを使用して、特定のオブジェクト名と位置をシノニムのユーザーから切り離して、リモート・データに 1 ワードでアクセスできるようにします。シノニムを作成するための構文は次のとおりです。

```
CREATE [PUBLIC] synonym_name
FOR [schema.]object_name[@database_link_name]
```

この場合、項目の意味は次のとおりです。

| | |
|---|---|
| [PUBLIC] | すべてのユーザーがこのシノニムを使用できることを指定します。このパラメータを省略すると、シノニムはプライベートになり、作成者のみが使用可能になります。パブリック・シノニムを作成できるのは、CREATE PUBLIC SYNONYM システム権限を持つユーザーのみです。 |
| synonym_name | ユーザーとアプリケーションが参照する代替オブジェクト名を指定します。 |
| schema | object_name に指定したオブジェクトのスキーマを指定します。このパラメータを省略すると、作成者のスキーマがオブジェクトのスキーマとして使用されます。 |
| object_name | 表、ビュー、シーケンス、スナップショット、型、プロシージャ、ファンクションまたはパッケージのいずれかを適切に指定します。 |
| database_link_name | object_name で指定されたオブジェクトが配置されたリモート・データベースとスキーマを識別するデータベース・リンクを指定します。 |

分散システム内のデータベースすべてに、データベース HQ に格納された表 SCOTT.EMP のパブリック・シノニムが定義されるとします。

```
CREATE PUBLIC SYNONYM emp FOR scott.emp@hq.acme.com;
```

この場合、SCOTT.EMP@HQ.ACME.COM の位置はパブリック・シノニムにより隠されているので、社員管理アプリケーションは、そのアプリケーションをどこで使用するかに関係なく設計できます。

アプリケーション内の SQL 文は、パブリック・シノニム EMP を参照することによって表にアクセスします。

さらに、EMP 表を HQ データベースから HR データベースに移動した場合でも、システムの各ノードで変更する必要があるのはパブリック・シノニムのみです。社員管理アプリケーションは、すべてのノードで正常に働きます。

シノニムは、そのスキーマで一意の名前を持つオブジェクトである必要があります。スキーマに同じ名前のスキーマ・オブジェクトとパブリック・シノニムが含まれている場合は、スキーマを所有するユーザーがその名前を参照するたびに、スキーマ・オブジェクトが検索されます。

## シノニムと権限

シノニムは実際のオブジェクトの参照です。特定のスキーマ・オブジェクトに対するシノニムにアクセスするユーザーには、スキーマ・オブジェクト自体への権限も必要です。たとえば、ユーザーがシノニムにアクセスしようとしても、そのシノニムが特定する表に対する権限を持っていなければ、エラーが発生し、該当の表またはビューが存在しないことが示されます。

あるローカル・シノニムがリモート・オブジェクトの別名であるとします。このローカル・シノニムの所有者は、このシノニムに対するオブジェクト権限を他のローカル・ユーザーに付与できません。この動作はローカルの表またはビューに対する別名のシノニムの権限管理とは異なります。シノニムがリモート・オブジェクトの別名になっている場合は、そのシノニムに対するローカル権限は付与できません。リモート・オブジェクトに対する権限も付与する結果になり、それは許されていないからです。したがって、シノニムを位置の透過性のために使用している場合は、ローカル権限管理は実行できません。シノニムを構成するオブジェクトのセキュリティは、リモート・ノードでのみ制御されます。たとえば、ユーザー ADMIN は、EMP_SYN シノニムに対するオブジェクト権限は何も付与できません。

ビュー定義またはプロシージャ定義で参照されているデータベース・リンクと異なり、シノニムで参照されているデータベース・リンクの解決は、そのシノニムに対する参照を実際に解析する時点で、スキーマが所有しているプライベート・リンクを最初に検索することにより行われます。

したがって、予定通りにオブジェクト解決が実行されるようにするには、ローカルにあるオブジェクトのスキーマをシノニムの定義で指定することが特に重要です。

# プロシージャと位置の透過性

「プロシージャ」と呼ばれる PL/SQL プログラム・ユニットも位置の透過性を提供できます。2 つのオプションがあります。

- ローカル・プロシージャがリモート・データを参照
- ローカル・シノニムがリモート・プロシージャを参照

シノニムを使用する位置の透過性は第 2 番目のオプションで提供されます。このオプションについては、2-22 ページの「シノニムと位置の透過性」を参照してください。第 1 番目のオプションについては、次の項で説明します。

## リモート・データを参照するローカル・プロシージャ

プロシージャまたはファンクション（スタンドアロンまたはパッケージ）には、リモート・データを参照する SQL 文を含めることができます。たとえば、次の文により作成されるプロシージャを考えます。

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
DELETE FROM emp@hq.acme.com
WHERE empno = enum;
END;
```

ユーザーまたはアプリケーションが FIRE_EMP プロシージャをコールするときに、変更されるのがリモート表かどうかはわかりません。

ローカル・プロシージャ、ビューまたはシノニムを使用してリモート・データをプロシージャ内の文が間接的に参照する場合、位置の透過性は2番目のレイヤーで実現されます。たとえば、次の文はローカル・シノニムを定義します。

```
CREATE SYNONYM emp FOR emp@hq.acme.com;
```

このシノニムを使用して、次の文でFIRE_EMPプロシージャを定義できます。

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
DELETE FROM emp WHERE empno = enum;
END;
```

表EMP@HQが改名または移動された場合は、表を参照するローカル・シノニムのみに変更が必要です。プロシージャおよびこのプロシージャをコールするアプリケーション側では、何も変更する必要はありません。

### プロシージャと権限

ローカル・プロシージャに、リモートの表またはビューを参照する文が含まれているとします。ローカル・プロシージャの所有者は、どのユーザーにもEXECUTE権限を付与できます。これにより、そのユーザーには、プロシージャを実行する権限に加え、間接的にリモート・データにアクセスする権限が付与されます。

一般に、プロシージャはセキュリティに役立ちます。プロシージャ内部で参照されるオブジェクトについての権限は、コール元のユーザーに明示的に付与する必要はありません。

## 文の透過性

リモート表を参照するために、次の標準DML文を使用できます。

- SELECT（問合せ）
- INSERT
- UPDATE
- DELETE
- SELECT...FOR UPDATE
- LOCK TABLE

結合、集約操作、副問合せおよびSELECT ... FOR UPDATEが含まれる問合せは、ローカルおよびリモートの表とビューを必要な数だけ参照できます。たとえば、次の問合せは、2つのリモート表からの情報を結合します。

```
SELECT empno, ename, dname FROM scott.emp@sales.division3.acme.com e,
jward.dept@hq.acme.com d
  WHERE d.deptno = d.deptno;
```

UPDATE 文、INSERT 文、DELETE 文および LOCK TABLE 文は、ローカル表とリモート表の両方を参照できます。リモート・データを更新するにはプログラミングは不要です。たとえば、次の文は、ローカル・データベースのスキーマ JWARD で、表 EMP から行を選択することにより、スキーマ SCOTT.SALES のリモート表 EMP に新しい行を挿入します。

```
INSERT INTO scott.emp@sales.division3.acme.com
  SELECT * FROM jward.emp;
```

## 制限事項

文の透過性には、いくつかの制限が適用されます。

- 単一の SQL 文の中では、参照される LONG 列、LONG RAW 列、順序、更新される表およびロックされる表は、すべて同じノードにあることが必要です。
- Oracle ではリモートのデータ定義言語（DDL）の文（CREATE、ALTER および DROP など）は使用できません。

ANALYZE 文の LIST CHAINED ROWS 句は、リモート表を指定できません。

## 環境依存型 SQL 関数の値

分散システムでは、文（または文の一部）がどこで実行されるかにかかわらず、SYSDATE、USER、UID および USERENV などの環境依存型の SQL 関数は、ローカル・サーバーに関して常に評価されます。

**注意：** USERENV 関数は、問合せによる使用のみサポートされています。

## リモート文および分散文用の共有 SQL

リモート文または分散文が共有 SQL を使用する仕組みは、基本的にはローカル文の場合と同じです。SQL テキストおよび参照オブジェクトが一致しており、バインド変数のあらゆるバインド型は同一である必要があります。可能であれば、共有 SQL 領域は、文（または分解された問合せ文）のローカルおよびリモート上の処理に使用されます。

# 3

# 分散トランザクション

この章では、Oracle8*i* が分散トランザクションの整合性をどのようにして維持するかについて説明します。この章で説明する項目は次のとおりです。

- 分散トランザクション管理
- 準備フェーズとコミット・フェーズ
- セッション・ツリー
- ケース・スタディ
- システム変更番号の調整
- 読取り専用分散トランザクション
- 1 ノード当たりの分散トランザクション数の制限
- 分散トランザクションに関する問題のトラブルシューティング
- インダウト・トランザクションの手動上書き
- インダウト・トランザクションの手動コミット
- 接続保持時間の変更
- 分散トランザクション回復機能のテスト

# 分散トランザクション管理

分散トランザクションに関与する全参加者（ノード）は、そのトランザクションに関して同じ動作を実行する必要があります。つまり、すべてのノードがコミットするかまたはロールバックする必要があります。

Oracle8*i* では、分散トランザクションのコミットまたはロールバックが自動的に制御、監視され、2 フェーズ・コミットと呼ばれるトランザクション管理のメカニズムを使用して、グローバル・データベース（トランザクションに関係するデータベースの集合）の整合性が維持されています。このメカニズムは、完全に透過的です。ユーザーやアプリケーション開発者側では、これを使用するためのプログラミングはまったく不要です。

次のセクションでは、2 フェーズ・コミットのメカニズムの働きについて説明します。

# 準備フェーズとコミット・フェーズ

分散トランザクションのコミットは、2 つのフェーズに区別されます。

| | |
|---|---|
| 準備フェーズ | グローバル・コーディネータ（開始ノード）は参加ノードに対して、準備（エラーがあってもコミットまたはロールバックすることを確約する）を要求します。 |
| コミット・フェーズ | すべての参加ノードが準備完了を示す応答をコーディネータに戻すと、コーディネータはすべてのノードにトランザクションのコミットを要求します。準備のできていない参加ノードが 1 つでもあれば、コーディネータはすべてのノードにトランザクションのロールバックを要求します。 |

あるユーザーが COMMIT 文により分散トランザクションをコミットすると、前述の両方のフェーズが自動的に実行されます。次の各セクションでは、それぞれのフェーズについてさらに詳しく説明します。

## 準備フェーズ

分散トランザクションをコミットする最初のフェーズは準備フェーズで、ここではトランザクションのコミットは実際には実行されません。かわりに、分散トランザクションで参照されるすべてのノードが（3-8 ページの「コミット・ポイント・サイト」で説明するコミット・ポイント・サイトを除く）、コミットの準備をするように指示されます。

準備フェーズの実行により、ノードは十分な情報を記録し、発生した障害に関係なくその後にトランザクションをコミットまたは異常終了できるようにします（異常終了の場合は、ロールバックが実行されます）。

ノードが準備完了を示す応答をリクエスタに送ると、その準備完了ノードは、後で該当のトランザクションをコミットまたはロールバックできること、およびそのトランザクションのコミットまたはロールバックについて単独決定をしないことを約束したことになります。

---

**注意：** ノードの準備完了後に開始された問合せは、すべてのフェーズが完了するまでは（エラーが発生しなければ、きわめて短時間）、関連するロック済みのデータにアクセスできません。

---

ノードは、準備するよう指示された場合、次の 3 つの応答のいずれかを戻すことができます。

| | |
|---|---|
| prepared（準備完了） | ノードのデータは分散トランザクション内の文により変更され、ノードは正常に準備を完了しています。 |
| read-only（読取り専用） | このノードのデータは、変更されていないか、または変更できません（問合せの対象のみ）。したがって、準備も不要です。 |
| abort（異常終了） | ノードの準備は正常に完了できていません。 |

## 準備フェーズでのノードの処理

準備フェーズを完了するために、各ノードは次の処理をします。

- ノードは、その下位ノード（後で参照されるノード）に準備要求を出します。
- ノードは、トランザクションがそのノードまたはその下位ノードのどれかのデータを変更するかどうかをチェックします。変更がない場合、ノードは次のステップをスキップして、読取り専用のメッセージ（次を参照）で回答します。
- データが変更された場合、ノードは、トランザクションをコミットするために必要なすべてのリソースを割り当てます。
- ノードは、該当のトランザクションが実行した変更に対応するすべてのエントリを、ローカルの REDO ログに書き込みます。
- ノードは、該当のトランザクションについて保持されているロックが障害発生時にも確実に保持されるようにします。
- ノードは、分散トランザクションで参照元のノードに、準備したメッセージを返します。あるいは、ノードの準備またはその下位ノードの 1 つの準備が失敗した場合は、異常終了のメッセージ（次を参照）を返します。

これらの処理により、トランザクションがその後そのノードでコミットまたはロールバックできることが保証されます。この後、準備完了ノードは、COMMIT または ROLLBACK を受け取るまで待機します。ノードの準備が完了すると、そのトランザクションは「インダウト」と呼ばれます。

### 読取り専用応答

あるノードが準備を要求され、データベースに影響する SQL 文がそのノードのデータを変更しなかった場合は、そのノードは、参照元のノードに読取り専用メッセージで応答します。このようなノードは第 2 のフェーズ（コミット・フェーズ）には参加しません。読取り専用分散トランザクションの詳細は、3-17 ページの「読取り専用分散トランザクション」を参照してください。

### 準備の失敗

準備に失敗した場合、ノードは次の処理を実行します。

- ノードは、該当のトランザクションが現在保持しているすべてのリソースを解放し、そのトランザクションのローカル部分をロールバックします。
- ノードは、分散トランザクションの参照元のノードに「異常終了のメッセージ」を返します。

これらの処理は、そのトランザクションをロールバックしてグローバル・データベースのデータの整合性を保証するために、その分散トランザクションに参加している他のノードに波及します。

さらにまた、この処理によって分散トランザクションの基本規則が守られます。このトランザクションに参加するすべてのノードは、同じ論理時刻に、このトランザクションを一斉にコミットするか、または一斉にロールバックします。

## コミット・フェーズ

分散トランザクションのコミット処理の第 2 のフェーズは、コミット・フェーズです。このフェーズに移る前に、分散トランザクションで参照されている全ノードには、トランザクションをコミットするために必要なリソースが保証されています。つまり、該当するすべてのノードは準備完了の状態になっています。

したがって、コミット・フェーズは次の各ステップで構成されます。

1. グローバル・コーディネータは、すべてのノードにメッセージを送って、トランザクションをコミットするように指示します。
2. 各ノードは、分散トランザクションのローカル部分をコミット（ロックをリリース）し、トランザクションがコミットしたことを示すために、追加の REDO エントリをローカル REDO ログに記録します。

コミット・フェーズが完了すると、分散システムのすべてのノードのデータは相互に一貫した状態になっています。

準備とコミットの両フェーズ中に、ネットワーク障害またはシステム障害が原因で、各種の障害が発生する場合があります。障害状況の説明と 2 フェーズ・コミット中に発生した障害の解決方法は、3-19 ページの「分散トランザクションに関する問題のトラブルシューティング」を参照してください。

# セッション・ツリー

分散トランザクションで文が発行されると、トランザクションに関与する全ノードの「セッション・ツリー」が定義されます。セッション・ツリーは、セッションとその役割の関係を記述する階層型モデルです。同じ分散トランザクションのセッション・ツリーに含まれるすべてのノードは、次の 1 つ以上の役割を持っています。

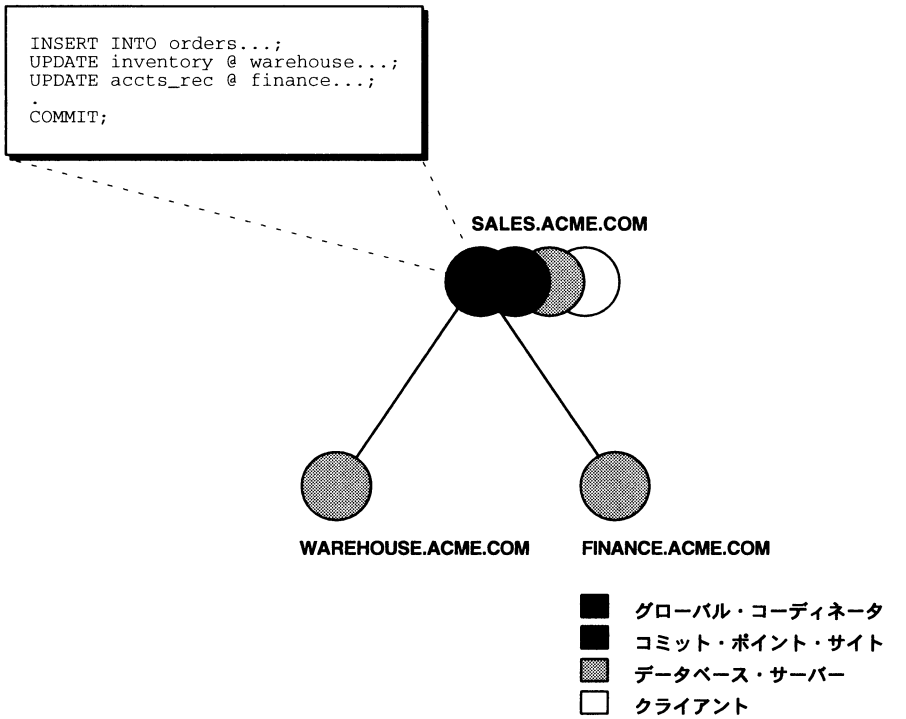
- クライアント
- データベース・サーバー
- グローバル・コーディネータ
- ローカル・コーディネータ
- コミット・ポイント・サイト（3 - 8 ページを参照）

分散トランザクションでノードが果たす役割は、次の条件により決まります。

- トランザクションがローカルかリモートか
- ノードのコミット・ポイント強度（3 - 8 ページを参照）
- 要求されたすべてのデータが 1 つのノードで使用可能なのか、それともトランザクションを完了するには他のノードを参照する必要があるのか
- ノードが読取り専用であるかどうか

図 3-1 に、簡単なセッション・ツリーを示します。

**図 3-1 簡単なセッション・ツリーの例**




## クライアント

ノードは、別のノードのデータベースからの情報を参照するときに、クライアントとして働きます。参照されるノードは「データベース・サーバー」です。前述の図の例では、ノード SALES.ACME.COM は、WAREHOUSE および FINANCE の各データベースを提供するノード（データベース・サーバー）のクライアントです。

## サーバーとデータベース・サーバー

サーバーは、分散トランザクションで直接参照されるノード、または別のノードによりそのデータベースのデータが要求されているために、トランザクションへの参加を要求されるノードです。データベースをサポートするノードは、データベース・サーバーとも呼ばれます。

図 3-1 では、SALES データベースを保有するノードのアプリケーションが、WAREHOUSE および FINANCE データベースを保有するノードのデータにアクセスする分散トランザクションを開始します。

したがって、SALES.ACME.COM の役割はクライアント・ノードであり、WAREHOUSE と FINANCE は両方ともデータベース・サーバーです。この例で、アプリケーションは SALES データベースの情報の変更も要求しているため、SALES はデータベース・サーバーであり、クライアントでもあります。

## ローカル・コーディネータ

分散トランザクションで役割を果たすために他のノードのデータを参照する必要があるノードは、「ローカル・コーディネータ」と呼ばれます。図 3-1 では、SALES.ACME.COM は、グローバル・コーディネータですが、直接参照するノード WAREHOUSE.ACME.COM および FINANCE.ACME.COM を調整するため、ローカル・コーディネータと考えることもできます。

ローカル・コーディネータは、次の処理を実行することにより、直接通信するノード間のトランザクションの調整を担当します。

- それらのノードとの間で、トランザクション状態情報の受渡しをします。
- それらのノードに問合せを渡します。
- それらのノードから問合せを受け取り、それを他のノードに渡します。
- 問合せの結果をその開始元のノードに戻します。

## グローバル・コーディネータ

分散トランザクションを起動したノード（分散トランザクションを発行するデータベース・アプリケーションの直接の接続先）をグローバル・コーディネータといいます。このノードは、セッション・ツリーの親、つまりルートになります。グローバル・コーディネータは、分散トランザクション中に次の操作を実行します。

- 分散トランザクションの SQL 文、リモート・プロシージャ・コールなどはすべて、グローバル・コーディネータから直接参照されるノードに送られ、次のようにセッション・ツリーが形成されます。

  たとえば、図 3-1 では、ノード SALES.ACME.COM で発行されたトランザクションが、データベース・サーバー WAREHOUSE.ACME.COM および FINANCE.ACME.COM からの情報を参照します。

  したがって、SALES.ACME.COM はこの分散トランザクションのグローバル・コーディネータです。

  - グローバル・コーディネータは、コミット・ポイント・サイト（次を参照）以外のすべての直接参照ノードに、トランザクションについての準備を指示します。

- すべてのノードが正常に準備を完了すると、グローバル・コーディネータは、コミット・ポイント・サイトにトランザクションのグローバル・コミットの開始を指示します。
- 1つ以上の異常終了メッセージがあった場合は、グローバル・コーディネータはすべてのノードにトランザクションのグローバル・ロールバックの開始を指示します。

# コミット・ポイント・サイト

コミット・ポイント・サイトの仕事は、グローバル・コーディネータの指示に従ってコミットまたはロールバックを開始することです。システム管理者は、すべてのノードにコミット・ポイント強度（次を参照）を割り当てることにより、セッション・ツリーでコミット・ポイント・サイトになるノードを常に1つ指定します。コミット・ポイント・サイトとして選択されるノードは、最も重要なデータ（最も広く使用されるデータ）を格納するノードである必要があります。

コミット・ポイント・サイトは、次の2つの点で、分散トランザクションに関連する他のサイトとは明らかに異なります。

- コミット・ポイント・サイトが準備状態になることはありません。これは潜在的な利点です。なぜなら、コミット・ポイント・サイトに最重要データが格納されていれば、障害状況に陥った場合でも、このデータがインダウトになることはないからです。(障害状況では、失敗したノードは準備状態のままになり、インダウト・トランザクションが解決されるまで、データに対する必要なロックを保持したままとなります。)
- 実際には、コミット・ポイント・サイトでの分散トランザクションの結果により、すべてのノードでそのトランザクションがコミットされるか、ロールバックされるかが決定されます。グローバル・コーディネータは、すべてのノードがコミット・ポイント・サイトと同じ方法でトランザクションを完了できるようにします。

分散トランザクションは、すべてのノードが準備を完了し、コミット・ポイント・サイトがそのトランザクションをコミットした時点で（参加ノードの一部がまだ準備状態になっているだけでトランザクションが実際にはコミットされていなくても）コミットされたものとみなされます。

コミット・ポイント・サイトのREDOログは、このノードで分散トランザクションがコミットされると同時に更新されます。同じように、コミット・ポイント・サイトでコミットされていない場合には、分散トランザクションはコミットされていないとみなされます。

## コミット・ポイント強度

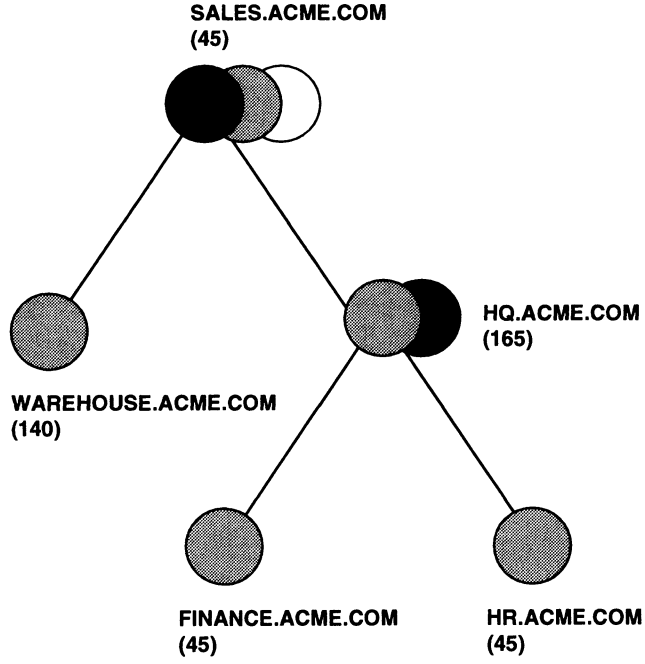
データベース・サーバーとして働くすべてのノードには、コミット・ポイント強度を割り当てる必要があります。データベース・サーバーが分散トランザクションで参照される場合、そのコミット・ポイント強度の値が、2フェーズ・コミットで果たす役割を決定します。特に、コミット・ポイント強度は、特定のノードが分散トランザクションにおけるコミット・ポイント・サイトであるかどうかを決定します。

この値は、初期化パラメータ COMMIT_POINT_STRENGTH を使用して指定されます（3 - 8 ページを参照）。コミット・ポイント・サイトは、準備フェーズの開始時に決定されます。

コミット・ポイント・サイトは、該当のトランザクションに参加しているノードからのみ選択されます。そのサイトが決定されると、グローバル・コーディネータは、参加しているすべてのノードに準備メッセージを送信します。グローバル・コーディネータにより直接参照されるノードの中で、最高位のコミット・ポイント強度を持つノードが選択されます。次に、最初に選択されたノードが、そのサーバー（このトランザクションについて取得する必要がある情報を持っている他のノード）の中に、さらに高いコミット・ポイント強度を持つものがあるかどうかを判断します。

トランザクション内で直接参照されている最高のコミット・ポイント強度を持つノード、またはそのサーバーのうちさらに高いコミット・ポイント強度を持つものが、コミット・ポイント・サイトになります。図 3-2 は、各ノードのコミット・ポイント強度（カッコ内）をサンプルのセッション・ツリーで表し、コミット・ポイント・サイトとして選択されたノードを示しています。

**図 3-2 コミット・ポイント強度とコミット・ポイント・サイトの判断**




次の条件は、コミット・ポイント・サイトを判断する場合に適用されます。

- 読取り専用ノード（トランザクションのローカル・データを変更しないノード）は、コミット・ポイント・サイトとして指定できません。

- グローバル・コーディネータから直接参照される複数のノードが、同じコミット・ポイント強度の場合、Oracle8*i* は、それらのノードの 1 つをコミット・ポイント・サイトとして指定します。
- 分散トランザクションがロールバックで終了した場合、準備フェーズおよびコミット・フェーズは必要とされず、したがってコミット・ポイント・サイトが決定されることはありません。かわりに、グローバル・コーディネータがすべてのノードに ROLLBACK 文を送信し、分散トランザクションの処理を終了させます。

コミット・ポイント強度は、分散トランザクション内でのコミット・ポイント・サイトを決定するだけです。コミット・ポイント・サイトはトランザクションの状態についての情報を格納するため、コミット・ポイント・サイトは、他のノードがトランザクションの状態についての情報を必要としたときに、頻繁に信頼性が低下したり使用不能になるノードであってはなりません。

図 3-2 が示すように、コミット・ポイント・サイトとグローバル・コーディネータは、セッション・ツリーの異なるノードであってもかまいません。

各ノードのコミット・ポイント強度は、初期接続を確立したときに、コーディネータ（複数の場合もあります）に通知されます。コーディネータは、直接通信した各ノードのコミット・ポイント強度を保持するため、2 フェーズ・コミット時にコミット・ポイント・サイトを効率よく選択できます。したがって、コミットが発生するたびにコーディネータとノードの間でコミット・ポイント強度を交換する必要はありません。

## インスタンスのコミット・ポイント強度の指定

準備またはコミット・フェーズ中にエラーが発生した場合に、一番重要なサーバーが確実に「非ブロック化」されるように、各ノードに対するコミット・ポイント強度を指定します。

ノードのコミット・ポイント強度は初期化パラメータ COMMIT_POINT_STRENGTH で設定します。値の範囲は 0 ～ 255 の整数です。たとえば、データベースのコミット・ポイント強度を 200 に設定するには、データベースのパラメータ・ファイルに次の 1 行を挿入します。

```
COMMIT_POINT_STRENGTH=200
```

**追加情報：** デフォルト値については、Oracle オペレーティング・システム固有のマニュアルを参照してください。

# ケース・スタディ

このケース・スタディでは次の各事項について説明します。

- セッション・ツリーの定義
- コミット・ポイント・サイトの決定方法
- 準備要求メッセージの送信時期
- トランザクションが実際にコミットする時期
- トランザクションについてローカルに格納される情報

## シナリオ

会社には、独立した Oracle8*i* サーバー SALES.ACME.COM および WAREHOUSE.ACME.COM があります。販売レコードが SALES データベースに挿入されると、それに対応付けられたレコードが WAREHOUSE データベースで更新されます。

## プロセス

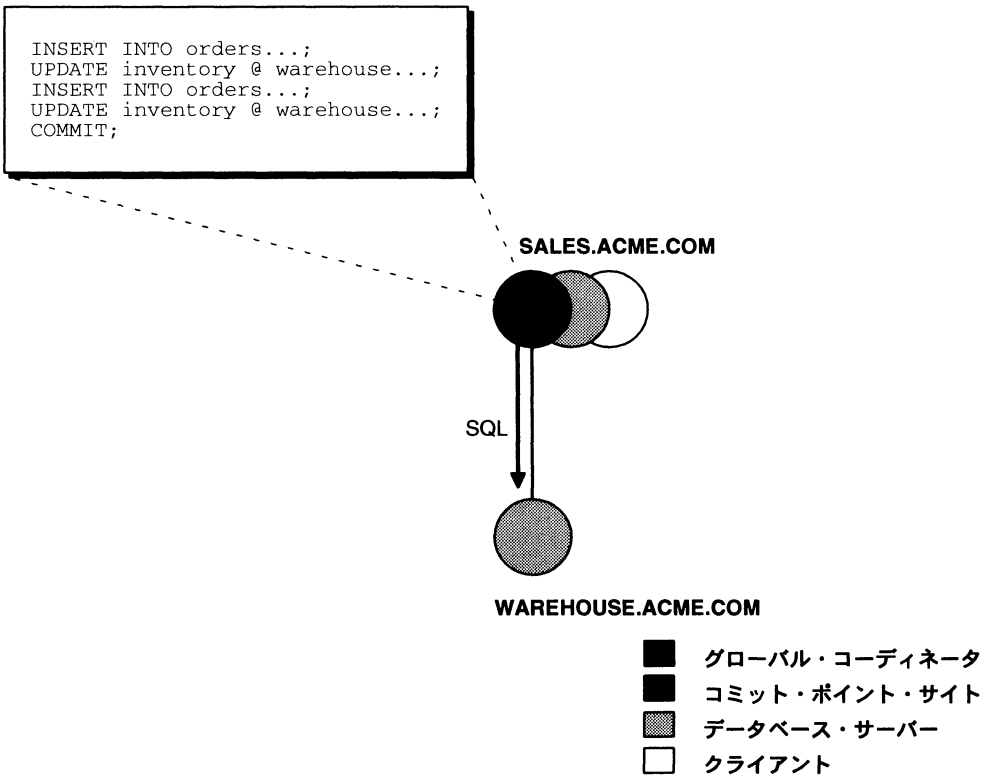
注文を入力する分散トランザクションでは、次のステップが実行されます。

1. アプリケーションが SQL 文を発行します。

   販売部門で、販売員がデータベース・アプリケーションを使用して注文を入力し、その後にコミットします。アプリケーションは、多数の SQL 文を発行してその注文を SALES データベースに入力し、WAREHOUSE データベースの在庫目録を更新します。

   これらの SQL 文はすべて単一の分散トランザクションの一部分であり、発行されたすべての SQL 文が全体で 1 単位として成功または失敗することになります。これによって、注文を入力したのにそれを反映するために在庫目録が更新されない可能性を防止できます。実際に、トランザクションは、グローバル・データベースのデータの一貫性を保証します。トランザクションにある各 SQL 文が実行されると、図 3-3 のようなセッション・ツリーが定義されます。

**図 3-3 セッション・ツリーの定義**




次の点に注意してください。

- SALES データベースで実行中の受注アプリケーションが、トランザクションを開始します。したがって、SALES.ACME.COM はこの分散トランザクションのグローバル・コーディネータです。
- 受注アプリケーションは、新規販売レコードを SALES データベースに挿入し、倉庫の在庫目録を更新します。したがって、ノード SALES.ACME.COM および WAREHOUSE.ACME.COM はどちらもデータベース・サーバーです。さらに、SALES.ACME.COM は在庫目録を更新するので、これは WAREHOUSE.ACME.COM のクライアントです。

これで、この分散トランザクションについてのセッション・ツリーの定義が完成しました。

ツリー内の各ノードがローカル・データを参照するSQL文を実行するために必要なデータ・ロックを獲得していることに注意してください。これらのロックは、SQL文が実行された後も、2フェーズ・コミットが完了するまでそのまま残りです。

2. アプリケーションがCOMMIT文を発行します。

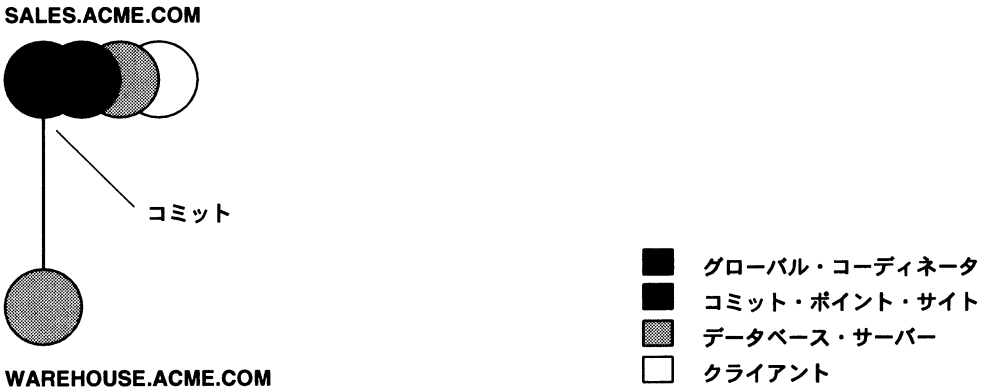
   注文を入力するトランザクションの最後の文が発行されることになります。これは、準備フェーズから始まる2フェーズ・コミットを開始するCOMMIT文です。

3. グローバル・コーディネータがコミット・ポイント・サイトを判断します。

   コミット・ポイント・サイトは、COMMIT文の直後に判断されます。図3-4に示すように、グローバル・コーディネータSALES.ACME.COMがコミット・ポイント・サイトになることが確認されます。

   コミット・ポイント・サイトの決定方法の詳細は、3-10ページの「インスタンスのコミット・ポイント強度の指定」を参照してください。

**図3-4 コミット・ポイント・サイトの決定**




4. グローバル・コーディネータが準備要求メッセージを送ります。

   コミット・ポイント・サイトの確認後、グローバル・コーディネータは、直接参照されるセッション・ツリーのすべてのノード（コミット・ポイント・サイトを除く）に準備メッセージを送ります。この例では、WAREHOUSE.ACME.COMが準備を要求される唯一のノードです。

   WAREHOUSE.ACME.COMは準備をしようとします。ノードがトランザクションのうちローカルに依存する部分をコミットできること、およびコミット情報をそのローカルREDOログに記録できることを保証できる状態になると、そのノードは準備が正常に完了したことになります。

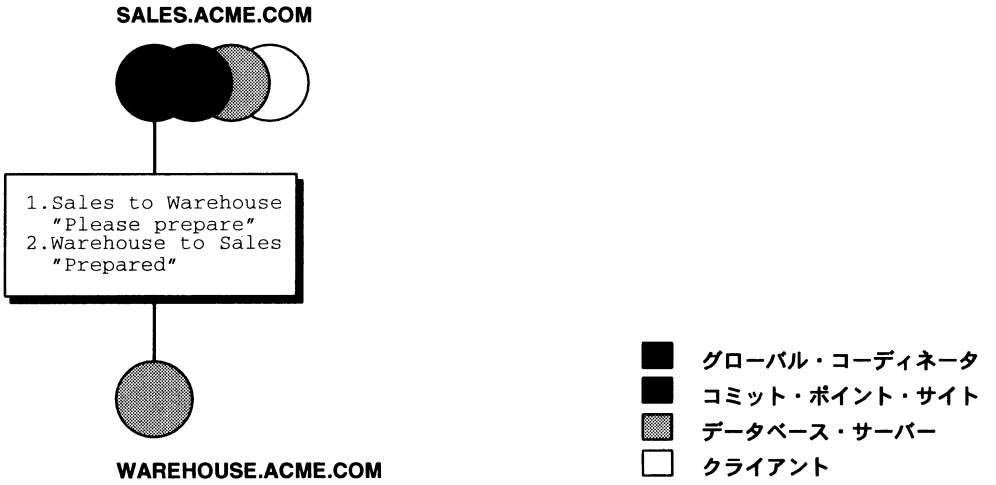
この例では、SALES.ACME.COM はコミット・ポイント・サイト（準備不要の）なので、WAREHOUSE.ACME.COM のみが準備要求メッセージを受け取ります。WAREHOUSE.ACME.COM は、準備完了メッセージの応答を SALES.ACME.COM に送ります。

各ノードは、準備を完了すると、準備の要求元のノードにメッセージを戻します。それらの応答に応じて、2 つのうちのどちらかの処理が実行されます。

- 準備を要求されたいずれかのノードが、グローバル・コーディネータに対して異常終了メッセージによる応答をした場合、グローバル・コーディネータは全ノードにトランザクションをロールバックするように指示し、プロセスは完了します。
- 準備を要求されたすべてのノードが、グローバル・コーディネータに対して準備完了または読取り専用のメッセージによる応答をした場合、つまり、すべてのノードが正常に準備を完了した場合は、グローバル・コーディネータは、コミット・ポイント・サイトにトランザクションのコミットを要求します。

図 3-5 は、この例のステップ 4 の部分を示したものです。

**図 3-5 PREPARE メッセージの送信と確認**




5. コミット・ポイント・サイトがコミットします。

SALES.ACME.COM は、WAREHOUSE.ACME.COM が準備完了の応答を受け取り、コミット・ポイント・サイト（この例では、自分自身）にトランザクションをコミットするように指示します。コミット・ポイント・サイトは、この時点でトランザクションをローカルにコミットし、その事実をローカル REDO ログに記録します。

WAREHOUSE.ACME.COM がまだコミットしていない場合でも、このトランザクションの結果は決定されます。つまり、コミットするノードの機能が遅れても、トランザクションは全ノードでコミットされるとみなされます。

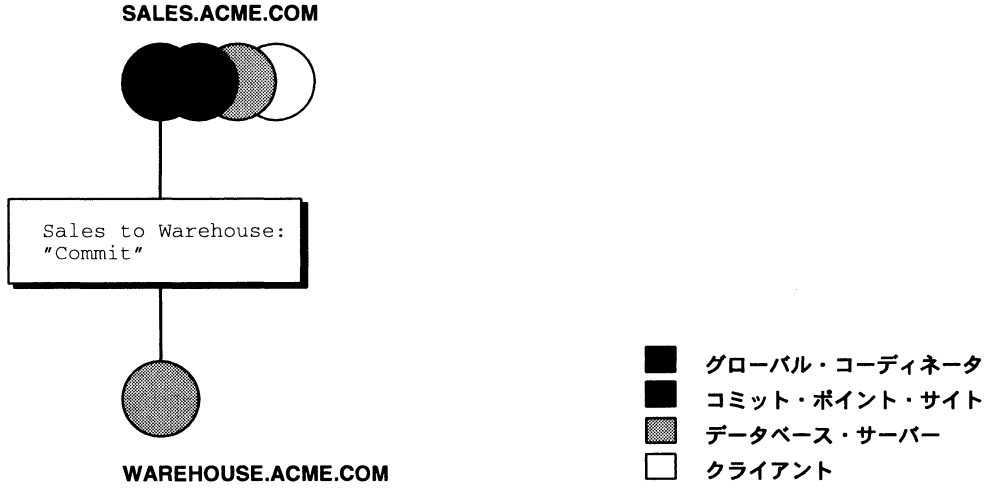
6. コミット・ポイント・サイトがグローバル・コーディネータにコミットを通知します。

   コミット・ポイント・サイトは、ここで、トランザクションのコミットが完了したことをグローバル・コーディネータに知らせます。この例では、コミット・ポイント・サイトとグローバル・コーディネータが同じノードなので、操作は不要です。コミット・ポイント・サイトは、該当の分散トランザクションに参加している他のすべてのノードでトランザクションがコミットされたことをグローバル・コーディネータが確認するまで、自分がトランザクションをコミットしたことを覚えています。

   グローバル・コーディネータは、コミット・ポイント・サイトでのコミット完了の通知を受けた後で、他のすべての直接参照ノードにコミットを指示します。今度は、ローカル・コーディネータがそのサーバーにコミットを指示し、この処理が順次実行されます。グローバル・コーディネータも含め、各ノードがトランザクションをコミットし、該当の REDO ログ・エントリをローカルに記録します。各ノードがコミットするにつれて、そのトランザクションについてローカルに保持されていたリソース・ロックが解放されます。

   図 3-6 は、この例のステップ 6 を示したものです。SALES.ACME.COM は、コミット・ポイント・サイトおよびグローバル・コーディネータの両方として機能するため、すでにトランザクションをローカルにコミットしています。SALES は、ここで WAREHOUSE.ACME.COM にトランザクションをコミットするように指示します。

**図 3-6 グローバル・コーディネータとその他のサーバーによるトランザクションのコミット**




7. グローバル・コーディネータおよびコミット・ポイント・サイトがコミットを完了します。

   すべての参照ノードとグローバル・コーディネータがトランザクションをコミットした後で、グローバル・コーディネータはコミット・ポイント・サイトにそれを通知します。

   このメッセージを待機していたコミット・ポイント・サイトは、この分散トランザクションに関する状態情報を消去し、終了したことをグローバル・コーディネータに通知します。つまり、コミット・ポイント・サイトは、この分散トランザクションのコミットに関する記憶をすべて消去します。2 フェーズ・コミットに関与するすべてのノードは、トランザクションを正常にコミット済みであり、今後その状態情報を確認することはないので、この消去に問題はありません。

   コミット・ポイント・サイトが、トランザクションに関する記憶を消去したことをグローバル・コーディネータに通知すると、グローバル・コーディネータは、そのトランザクション自体についての記憶を消去することにより、トランザクションを完了させます。

   これで COMMIT フェーズが完了し、分散トランザクションが終了します。

   ここまでで説明したステップはすべて自動的に実行され、1 秒もかからずに完了します。

# システム変更番号の調整

コミット済みの各トランザクションには、それに対応付けられたシステム変更番号（SCN）、つまりそのトランザクション内でSQL文が行った変更を一意に識別するための番号があります。分散システムでは、通信ノードのSCNが調整されるのは次のような場合です。

- 1つ以上のデータベース・リンクにより記述されているパスを使用する接続が発生する場合。
- 分散SQL文が実行される場合（実行フェーズが完了する）。
- 分散トランザクションがコミットする場合。

その他の利点として、分散システムのノード間をSCNが調整することにより、文レベルとトランザクション・レベルの両方でグローバルな分散読取りの一貫性が実現します。必要な場合には、グローバルな時間ベースの分散回復も達成できます。

準備フェーズを通じて、Oracle8*i*はトランザクションに関与する全ノードで最高位となるSCNを決定します。次に、トランザクションは、コミット・ポイント・サイトでその最高位のSCNを使用してコミットします。その後に、そのコミットSCNが、コミット決定とともにすべての準備完了ノードに送信されます。

# 読取り専用分散トランザクション

分散トランザクションのすべてまたは一部が読取り専用になるのは、次の3つの場合です。

- 分散トランザクションが部分的に読取り専用になるのは、次の場合です。
  - 1つ以上のノードで問合せのみを発行している場合。
  - レコードが一切変更されない場合。
  - 整合性制約の違反が生じたか、またはトリガーによる制約に違反したために、変更がロールバックされた場合。

  これらの場合はいずれも、ノードは準備を要求されたときに自分が読取り専用であることを認識します。読取り専用ノードは、それぞれのローカル・コーディネータに、応答として読取り専用メッセージを戻します。これにより、Oracleは後続の処理からそれらの読取り専用ノードを排除するため、コミット・フェーズの完了が速くなります。
- 分散トランザクションが完全に読取り専用（いずれのノードでもデータが変更されない）でありながら、トランザクションがSET TRANSACTION READ ONLY文で起動されていない場合があります。

  この場合は、すべてのノードは準備フェーズの実行中に自分が読取り専用であることを認識するため、コミット・フェーズは不要になります。ただし、グローバル・コーディネータは、すべてのノードが読取り専用かどうかがわからないため、準備フェーズに含まれている操作を実行する必要があります。

- 分散トランザクションが完全に読取り専用であり（すべてのノードで問合せのみ）、トランザクションが SET TRANSACTION READ ONLY 文で起動されている場合があります。この場合は、トランザクションでは問合せの実行のみが可能で、グローバル・コーディネータは 2 フェーズ・コミットのための処理をする必要はありません。他のトランザクションによる変更が原因でグローバル・トランザクション・レベルの読込みの一貫性が損なわれることはありません。これは、SCN の調整により、分散システムの各ノードで読込みの一貫性が自動的に保証されているからです。

# 1 ノード当たりの分散トランザクション数の制限

初期化パラメータ DISTRIBUTED_TRANSACTIONS は、指定したインスタンスがクライアントとおよびサーバーの両方として同時に参加できる分散トランザクションの数を制御します。この制限に到達した後で、後続のユーザーがリモート・データベースを参照する SQL 文を発行しようとすると、その文はロールバックされ、エラー・メッセージが戻されます。

```
ORA-2042:too many global transactions
```

たとえば、あるインスタンスについて DISTRIBUTED_TRANSACTIONS を 10 に設定したとします。この場合、最大 10 セッションが、分散トランザクションを同時に処理できます。11 番目のセッションが分散アクセスを要求する DML 文を発行すると、そのセッションにエラー・メッセージが戻され、その文はロールバックされます。

定常的に多数の分散トランザクションに参加するインスタンスがあり、現行の制限の結果として上記のエラー・メッセージが頻繁に戻される場合は、データベース管理者は、初期化パラメータ DISTRIBUTED_TRANSACTIONS の値を大きくすることを検討する必要があります。制限を大きくすることによって、より多数のユーザーが同時に分散トランザクションを発行できるようになります。

DISTRIBUTED_TRANSACTIONS 初期化パラメータをゼロに設定すると、どのセッションでも分散 SQL 文を発行できません。

また、ローカル・インスタンスの起動時には、RECO バックグラウンド・プロセスは起動されません。（これ以前に発生したネットワークまたはシステムの障害により）インダウトになっている分散トランザクションは、Oracle8*i* では自動的に解決できません。

したがって、この初期化パラメータをゼロに設定して分散トランザクションの発生を防止できるのは、新規インスタンスを起動するときで、前回のインスタンスのシャットダウン後に、インダウト分散トランザクションが残っていないことが確実な場合のみです。

**追加情報：** 詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

# 分散トランザクションに関する問題のトラブルシューティング

ネットワーク障害またはシステム障害が原因で、次のようなタイプの問題が発生することがあります。

- 障害発生時に処理中であった 2 フェーズ・コミットが、セッション・ツリーの全ノードで完了できないことがあります。
- 障害が持続した場合（たとえばネットワークが長時間ダウンした場合）、その間は、インダウト・トランザクションにより排他的にロックされたデータを他のトランザクションの文に使用できなくなります。

次の各項で、この状況について説明します。

## 2 フェーズ・コミットへの割込み障害

分散トランザクションをコミットするユーザー・プログラムには、次のいずれかのエラー・メッセージにより問題が通知されます。

```
ORA-02050:transaction ID rolled back,
          some remote dbs may be in-doubt
ORA-02051: transaction ID committed,
          some remote dbs may be in-doubt
ORA-02054: transaction ID in-doubt
```

健全なアプリケーションは、前述のエラーを受け取るとトランザクションについての情報を保存します。この情報は、後で手動による分散トランザクション回復が必要になった場合に使用できます。

---

**注意：** これらのエラー・メッセージが表示される障害のケースは、このマニュアルで対象としている範囲を超えており、システムを管理する上で必要なものではありません。

---

ネットワーク障害またはシステム障害により 1 つ以上のインダウト分散トランザクションが生じた場合でも、そのノードの管理者がなんらかのアクションをする必要はありません。Oracle8*i* の自動回復機能は、ネットワークまたはシステムの障害が解決すれば、セッション・ツリーの全ノードが同じ結果（つまり、すべてコミットまたはすべてロールバック）になるように、インダウト・トランザクションを透過的に完了させます。

ただし、停止状態が長引く場合、管理者はトランザクションのコミットまたはロールバックを強制実行し、ロックされたデータをすべて解放することができます。アプリケーションは、このような可能性を考慮に入れておく必要があります。

# 準備フェーズにおけるデータ・アクセスの障害

ユーザーがSQL文を発行すると、Oracle8*i*は、文を正常に実行するために必要なリソースをロックしようとします。ただし、要求されたデータが、まだコミットされていない他のトランザクションの文により現在保持されていて、一定時間を過ぎてもロックされたままになっていると、タイムアウトが発生します。次の2つのシナリオを考えます。

## トランザクション・タイムアウト

リモート・データベースでのロックが必要なDML SQL文の場合、要求されたデータに対するロックを別のトランザクション（分散または非分散）が現在所有していると、必要なロックを適用できないことがあります。このようなロックのため要求元のSQL文のブロックが続くと、タイムアウトが発生し、そのSQL文はロールバックされ、次のようなエラー・メッセージがユーザーに戻されます。

```
ORA-02049: time-out: distributed transaction waiting for lock
```

変更されたデータはないので、タイムアウトの結果必要になるアクションはありません。アプリケーションは、デッドロックが発生した場合と同様の処理をします。該当の文を実行したユーザーは、後で同じ文の再実行を試みることができます。ロックがまだ持続している場合は、ユーザーは管理者に連絡し、問題を報告する必要があります。

前述のような状況でのタイムアウトの間隔は、初期化パラメータDISTRIBUTED_LOCK_TIMEOUTを使用して制御できます。この間隔は、秒単位で指定します。たとえば、あるインスタンスのタイムアウト間隔を30秒に設定するには、対応付けられているパラメータ・ファイルに次の行を組み込みます。

```
DISTRIBUTED_LOCK_TIMEOUT=30
```

このタイムアウト間隔では、トランザクションが使用不可能なリソースが空くのを30秒待った後でも先に進めなかった場合は、前述したタイムアウト・エラーが発生します。

**追加情報：**初期化パラメータおよびパラメータ・ファイルの編集の詳細は、『Oracle8*i*リファレンス・マニュアル』を参照してください。

## インダウト・トランザクションからのロック

ローカル・データベースのロックを必要とする問合せまたはDML文は、インダウト分散トランザクションのリソースのロックにより、無期限にロックを阻止されることがあります。この場合は、次のエラー・メッセージがただちにユーザーに戻されます。

```
ORA-01591: lock held by in-doubt distributed transaction <id>
```

この場合、SQL文がただちにロールバックされます。該当の文を実行したユーザーは、後で同じ文の再実行を試みることができます。ロックが持続する場合は、管理者に連絡し、インダウト分散トランザクションのIDも含めて問題を報告する必要があります。

2 フェーズ・コミットの重要部分で障害が発生する可能性は低いことを考えると、前述のような状況が発生する可能性はまずありません。このような障害が発生した場合でも、ネットワーク障害またはシステム障害からの回復が早ければ、問題は自動的に解決され、手動で対処する必要はありません。したがって、ユーザーまたはデータベース管理者が検出する前に問題は通常解決されます。

# インダウト・トランザクションの手動上書き

データベース管理者は、ローカルのインダウト分散トランザクションの COMMIT または ROLLBACK を手動で強制実行することができます。ただし、特定のインダウト・トランザクションは、次のような状況の場合にのみ手動で上書きしてください。

- インダウト・トランザクションが、他のトランザクションに必要なデータをロックしている場合。エラー・メッセージ ORA-01591 が出てトランザクションが実行できないという連絡がユーザーから届く場合は、この状況が発生しています。
- インダウト・トランザクションのために、他のトランザクションがロールバック・セグメントのエクステンションを使用できない状態になっている 場合。インダウト分散トランザクションのローカル・トランザクション ID の最初の部分は、ロールバック・セグメントの ID に対応しています。この対応は、データ・ディクショナリ・ビュー DBA_2PC_PENDING および DBA_ROLLBACK_SEGS にリストされています。
- 2 フェーズ・コミットのフェーズを完了できない障害が、許容時間間隔以内に解消されない場合。このようなケースの例としては、通信ネットワークが破損した場合や、データベースが破損して回復処理を完了するまでに相当な時間が必要な場合などが考えられます。

通常、インダウト分散トランザクションをローカルで強制的に処理する場合は、他のデータベースの管理者と相談の上で決定してください。この決定を誤ると、データベースに追跡困難で手動訂正を必要とする矛盾が生じることがあります。

前述の状況に該当しない場合は、Oracle8*i* の自動回復機能により、常にトランザクションを完了できます。ただし、前述のどれかの基準に当てはまる場合は、管理者はインダウト・トランザクションのローカル上書きを検討する必要があります。

トランザクションをローカルに強制終了させる決定をくだした場合、データベース管理者は、次の目標を念頭に置いて使用可能な情報を分析します。

- 該当のトランザクションをコミットまたはロールバックしたノードを探します。すでにトランザクションを解決したノードが見つかった場合は、そのノードで実行された処理を手本にすることができます。
- 該当の分散トランザクションについて、DBA_2PC_PENDING の TRAN_COMMENT 列の中に情報が入っているかどうかを調べます。COMMIT コマンドの COMMENT パラメータには、コメントを組み込めます。たとえば、インダウト分散トランザクションのコメントに、そのトランザクションの起動元とトランザクションのタイプが示されていることがあります。

```
COMMIT COMMENT 'Finance/Accts_pay/Trans_type 10B';
```

- 該当の分散トランザクションについて、DBA_2PC_PENDING の ADVICE 列の中に情報が入っているかどうかを調べます。アプリケーションは、SQL コマンド ALTER_SESSION の ADVISE パラメータを使用して、分散トランザクションの個別の部分のコミットを強制するのか、またはロールバックを強制するのかについて、事前にアドバイスを指示できます。

  準備フェーズの間に各ノードに送信されるアドバイスは、現行トランザクションにおいて該当のデータベースに対して最後の DML 文を実行した時に有効なアドバイスです。

  たとえば、社員レコードをあるノードの EMP 表から別のノードの EMP 表に移動する分散トランザクションを考えます。このトランザクションは、次の順序の SQL 文を組み込むことによって、(各ノードで管理者が独立してインダウト・トランザクションを強制処理した場合にも) レコードを保護できます。

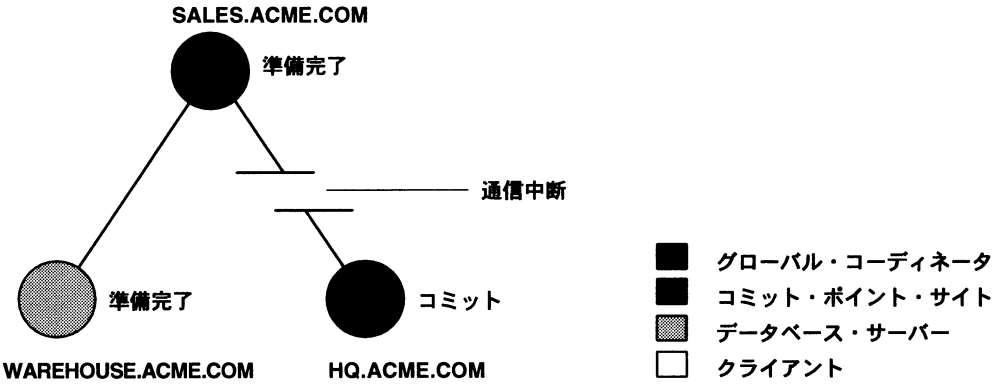
```
ALTER SESSION ADVISE COMMIT;
INSERT INTO emp@hq ... ;     /*advice to commit at HQ */
ALTER SESSION ADVISE ROLLBACK;
DELETE FROM emp@sales ... ; /*advice to roll back at SALES*/

ALTER SESSION ADVISE NOTHING;
```

  インダウト・トランザクションを手動で強制処理した場合、最悪の事態が発生しても、移動中の社員レコードが各ノードにコピーされるだけです。レコードが消失することはありません。

## 手動による上書きの例

次の例では、分散トランザクションのコミット中に発生した障害と、インダウト分散トランザクションのローカル部分のコミットまたはロールバックを手動で強制する前に情報を入手する方法を示します。図 3-7 は、この例を示しています。

**図 3-7 インダウト分散トランザクションの例**




この障害のケースでは、準備フェーズは完了しています。ただし、コミット・ポイント・サイトがトランザクションをコミットしたにもかかわらず、コミット・フェーズ中に、コミット・ポイント・サイトのコミット・メッセージ（コミット・ポイント・サイトでトランザクションがコミットされたことをグローバル・コーディネータに知らせるメッセージ）がグローバル・コーディネータに戻されませんでした。

ここでは、WAREHOUSE データベースの管理者であるとします。インダウト・トランザクションが他のトランザクションにとって重要なものであるため、在庫管理データがロックされました。インダウト・トランザクションがコミットまたはロールバックされるまで、このロックを保持する必要があるため、データはアクセス不能になっています。しかも、販売部門（SALES）と経営本部（HQ）との間の通信リンクは、すぐには解決できないことがわかっています。

そこで、次のステップを使用してインダウト・トランザクションのローカル部分を手動で強制処理することに決めました。

1. ユーザーからのフィードバックを記録します。
2. ローカル DBA_2PC_PENDING ビューに問い合せて、グローバル・トランザクション ID を取得し、インダウト・トランザクションについてのその他の情報を入手します。
3. ローカル DBA_2PC_NEIGHBORS ビューに問い合せて、セッション・ツリーのトレースを開始し、インダウト・トランザクションをすでに解決したノードを見つけます。
4. 正常な通信が再確立された後で、複合結果フラグをチェックします。

次の各項で、この例の各ステップについて詳しく説明します。

## ステップ 1: ユーザー・フィードバックの記録

インダウト・トランザクションのロックと競合するローカル・データベース・システムのユーザーは、次のエラー・メッセージを受け取ります。

```
ORA-01591: lock held by in-doubt distributed transaction 1.21.17
```

ここで、1.21.17 は、この例のインダウト分散トランザクションのローカル・トランザクション ID です。ローカル・データベース管理者は、強制処理の対象になるインダウト・トランザクションを識別するために、問題を報告したユーザーからこの ID 番号を入手し記録する必要があります。

## ステップ 2: DBA_2PC_PENDING への問合せ

インダウト・トランザクションについての情報を取得するために、ローカル DBA_2PC_PENDING（3 - 29 ページも参照）に問い合せます。

```
SELECT * FROM sys.dba_2pc_pending WHERE local_tran_id = '1.21.17';
```

たとえば、上記の問合せを WAREHOUSE で発行すると、次の情報が戻されます。

**図 3-8 DBA_2PC_PENDING の問合せ結果**

```
Column Name          Value
-------------------- -------------------------------------
LOCAL_TRAN_ID        1.21.17
GLOBAL_TRAN_ID       SALES.ACME.COM.55d1c563.1.93.29
STATE                prepared
MIXED                no
ADVICE
TRAN_COMMENT         Sales/New Order/Trans_type 10B
FAIL_TIME            31-MAY-91
FORCE_TIME
RETRY_TIME           31-MAY-91
OS_USER              SWILLIAMS
OS_TERMINAL          TWA139:
HOST                 system1
DB_USER              SWILLIAMS
COMMIT#
```

グローバル・トランザクション ID は、同じ分散トランザクションの各ノードについて同一の共通トランザクション ID です。この形式は次のとおりです。

```
global_database_name.hhhhhhhh.local_transaction_id
```

ここで、*global_database_name* はグローバル・コーディネータ（トランザクションの起動元）のデータベース名、*hhhhhhhh* はグローバル・コーディネータの内部データベース ID（16 進数 8 桁）、*local_tran_id* はグローバル・コーディネータで割り当てられた対応するローカル・トランザクション ID です。したがって、グローバル・コーディネータの場合は、グローバル・トランザクション ID の最後の部分とローカル・トランザクション ID とが一致するはずです。この例では、これらの番号が一致しないため、WAREHOUSE がグローバル・コーディネータではないことがわかります。

このノードのトランザクションは準備完了状態になっています。したがって、WAREHOUSE は、そのコーディネータからコミット・メッセージまたはロールバック・メッセージが送信されるのを待ちます。

トランザクションのコメントまたはアドバイスには、このトランザクションについての情報が組み込まれている場合があります。その場合は、そのコメントを参考にしてください。この例では、起動元（販売受注アプリケーション）とトランザクション・タイプがトランザクションのコメントに組み込まれています。この情報に、トランザクションのローカル部分をコミットするかロールバックするかを決定するために役立つ事項が示されていることがあります。

役に立ちそうなコメントがインダウト・トランザクションに含まれていない場合は、なんらかの追加の管理作業を実行してセッション・ツリーをトレースし、該当のトランザクションをすでに解決したノードを見つける必要があります。

## ステップ 3: DBA_2PC_NEIGHBORS への問合せ

このステップの目的は、セッション・ツリーをさかのぼってコーディネータを検索し、最終的にはグローバル・コーディネータに到達することです。途中で、トランザクションを解決したコーディネータが見つかる場合があります。見つからなかった場合は、最終的にコミット・ポイント・サイトに達するまで作業を進めます。この方法で、必ずインダウト・トランザクションを解決済みの状態にできます。

DBA_2PC_NEIGHBORS ビューには、インダウト・トランザクションに関連した接続についての情報が表示されます。各接続についての情報は、その接続がインバウンドかアウトバウンドかに応じて異なります。

- 接続がインバウンドの場合は、このノードは別のノード（別のノードのサーバー）に従属しています。この場合は、DATABASE 列に、そのノードに接続されているクライアント・データベースの名前がリストされ、DBUSER_OWNER 列に、インダウト・トランザクションで使用されたデータベース・リンク接続用のローカル・アカウントがリストされます。
- 接続がアウトバウンドの場合は、ノードは他のサーバーのクライアントであります。この場合は、DATABASE 列に、リモート・ノードに接続するデータベース・リンクの名前がリストされます。DBUSER_OWNER 列には、インダウト・トランザクション用のデータベース・リンクの所有者がリストされます。

さらに、INTERFACE 列には、ローカル・ノードまたは従属ノードがコミット・ポイント・サイトかどうかが示されています。

セッション・ツリーをトレースするために、ローカル DBA_2PC_NEIGHBORS ビューを問合せできます。この例の場合は、WAREHOUSE データベースにあるこのビューを問い合せます。

```
SELECT * FROM sys.dba_2pc_neighbors
   WHERE local_tran_id = '1.21.17'
   ORDER BY sess#, in_out;

Column Name          Value
-------------------- -------------------------------------
LOCAL_TRAN_ID        1.21.17
IN_OUT               in
DATABASE             SALES.ACME.COM
DBUSER_OWNER         SWILLIAMS
INTERFACE            N
DBID                 000003F4
SESS#                1
BRANCH               0100
```

このビューの中で特に注目する列は、IN_OUT、DATABASE、DBUSER_OWNER および INTERFACE の各列です。この例では、DATABASE 列に指定されているように、IN_OUT 列は WAREHOUSE データベースが SALES データベースのサーバーであることを示しています。

DB_OWNER 列に示されているように、WAREHOUSE との接続はデータベース・リンクを経由して SWILLIAMS のアカウントで確立されます。また、INTERFACE 列に示されているように、WAREHOUSE や WAREHOUSE の下位ノードのいずれも、コミット・ポイント・サイトではありません。この時点で、それぞれ判明したノードの管理者に連絡し、グローバル・トランザクション ID を使用してステップ 2 および 3 を各ノードで実行するように依頼します。

---

**注意：** 別のネットワークでこれらのノードに直接接続する場合は、ステップ 2 および 3 を繰り返します。

---

たとえば、SALES および HQ でステップ 2 および 3 を実行すると、それぞれ次の結果が戻されます。

## SALES.ACME.COM での保留トランザクションの状態の手動チェック

```
SELECT * FROM sys.dba_2pc_pending
   WHERE global_tran_id = 'SALES.ACME.COM.55d1c563.1.93.29';

Column Name          Value
-------------------- ---------------------------------------
LOCAL_TRAN_ID        1.93.29
GLOBAL_TRAN_ID       SALES.ACME.COM.55d1c563.1.93.29
STATE                prepared
MIXED                no
ADVICE
TRAN_COMMENT         Sales/New Order/Trans_type 10B
FAIL_TIME            31-MAY-91
FORCE_TIME
RETRY_TIME           31-MAY-91
OS_USER              SWILLIAMS
OS_TERMINAL          TWA139:
HOST                 system1
DB_USER              SWILLIAMS
COMMIT#

SELECT * FROM dba_2pc_neighbors
   WHERE global_tran_id = 'SALES.ACME.COM.55d1c563.1.93.29'
   ORDER BY sess#, in_out;
```

SALES では、このトランザクションについて 3 つの行があります（1 行は WAREHOUSE への接続用、1 行は HQ への接続用、1 行はユーザーが確立する接続用です）。SALES および HQ の接続についての行に対応する情報を次に示します。

```
Column Name          Value
-------------------- ---------------------------------------
LOCAL_TRAN_ID        1.93.29
IN_OUT               OUT
DATABASE             WAREHOUSE.ACME.COM
DBUSER_OWNER         SWILLIAMS
INTERFACE            N
DBID                 55d1c563
SESS#                1
BRANCH               1
```

```
Column Name          Value
-------------------- -------------------------------------
LOCAL_TRAN_ID        1.93.29
IN_OUT               OUT
DATABASE             HQ.ACME.COM
DBUSER_OWNER         ALLEN
INTERFACE            C
DBID                 00000390
SESS#                1
BRANCH               1
```

前述の問合せから得られる情報により、次のような事実が判明します。

- SALES は、ローカル・トランザクション ID とグローバル・トランザクション ID が一致しているため、グローバル・コーディネータです。このノードから 2 つのアウトバウンド接続が確立されていることと、インバウンド接続がないことにも注意してください（つまり、このノードは別のノードのサーバーではありません）。
- HQ またはそのサーバーの 1 つ（この例ではありません）がコミット・ポイント・サイトです。

## HQ.ACME.COM での保留 トランザクションの状態の手動チェック

```
SELECT * FROM dba_2pc_pending
   WHERE global_tran_id = 'SALES.ACME.COM.55d1c563.1.93.29';

Column Name          Value
-------------------- -------------------------------------
LOCAL_TRAN_ID        1.45.13
GLOBAL_TRAN_ID       SALES.ACME.COM.55d1c563.1.93.29
STATE                COMMIT
MIXED                NO
ACTION
TRAN_COMMENT         Sales/New Order/Trans_type 10B
FAIL_TIME            31-MAY-91
FORCE_TIME
RETRY_TIME           31-MAY-91
OS_USER              SWILLIAMS
OS_TERMINAL          TWA139:
HOST                 SYSTEM1
DB_USER              SWILLIAMS
COMMIT#              129314
```

この時点で、トランザクションを解決済みのノードが見つかりました。トランザクションはすでにコミットされています。したがって、インダウト・トランザクションをローカル・データベースで強制的にコミットできます（インダウト・トランザクションを手動でコミットまたはロールバックする場合の説明は、後述の項を参照してください）。調査結果が役立つと思われる他の管理者にも連絡をとっておくことをお薦めします。

## ステップ 4: 複合結果のチェック

トランザクションを手動で強制的にコミットまたはロールバックした後は、対応する行が保留トランザクション表に残っています。トランザクションの STATE（状態）は、トランザクションをどのように強制処理したかに応じて、forced commit（強制コミット）または forced abort（強制異常終了）に変更されています。

## 保留トランザクション表（DBA_2PC_PENDING）

Oracle8*i* のすべてのデータベースには「保留トランザクション表」があり、これは 2 フェーズ・コミットのフェーズを介して進行した分散トランザクションについての情報を格納する特別な表です。DBA_2PC_PENDING データ・ディクショナリ・ビューを参照することにより、データベースの保留トランザクション表を問合せできます。

保留トランザクション表にエントリを持つ各トランザクションは、次のカテゴリの 1 つに（DBA_2PC_PENDING.STATE に示されているように）分類されます。

| | |
|---|---|
| collecting<br>（収集中） | このカテゴリは、通常はグローバル・コーディネータまたはローカル・コーディネータにのみ適用されます。ノードは、準備可能かどうかを決定する前に、現在、他のデータベース・サーバーから情報を収集中です。 |
| prepared<br>（準備完了） | ノードは準備が完了しています。準備完了メッセージによる肯定応答をローカル・コーディネータに送信済みの場合も、まだ送信していない場合もあります。ただし、コミット・メッセージはまだ受け取っていません。ノードは、準備完了状態のままになり、トランザクションのコミットに必要なすべてのローカル・リソースのロックを保持しています。 |
| committed<br>（コミット済み） | このノード（任意のタイプ）はトランザクションをコミットしましたが、同じトランザクションに参加しているその他のノードはまだコミットしていない可能性があります。つまり、トランザクションは 1 つ以上のノードでまだ保留されています。 |

| | |
|---|---|
| forced commit（強制コミット） | 保留トランザクションは、データベース管理者の判断で強制的にコミットすることができます。このエントリは、データベース管理者がローカル・ノードでトランザクションを手動コミットした場合に発生します。 |
| forced abort（強制異常終了（ロールバック）） | 保留トランザクションは、データベース管理者の判断で強制的にロールバックすることができます。このエントリは、データベース管理者がローカル・ノードでこのトランザクションを手動でロールバックした場合に発生します。 |

その他に、保留トランザクション表の中では、複合結果フラグ（DBA_2PC_PENDING.MIXED に示される）にも注目してください。データベース管理者は、保留トランザクションを強制的にコミットまたはロールバックする場合、選択を誤る可能性があります（たとえば、ローカル管理者がトランザクションをロールバックし、その他のノードはそれをコミットした場合など）。

誤った決定は、自動的に検出され、対応する保留トランザクションのレコードの損傷フラグが設定されます（MIXED=yes）。

RECO（リカバラ）バックグラウンド・プロセスは、保留トランザクション表の情報を使用して、インダウト・トランザクションの状態を最終的に決定します。保留トランザクション表の情報は、保留中の分散トランザクションの自動回復プロシージャを手動で上書きすることを決定した場合に、データベース管理者も使用できます。

RECO により自動的に解決されたトランザクションは、すべて保留トランザクション表から自動的に除去されます。また、管理者が正しく解決したインダウト・トランザクションについての情報も、すべて（RECO が通信を再確立したときにチェックされ）保留トランザクション表から自動的に除去されます。ただし、管理者が解決した行のうちノード間での複合結果を生じたすべての行は、手動で削除するまではすべての参加ノードの保留トランザクション表の中に残ります。

# インダウト・トランザクションの手動コミット

ローカル・データベース管理者がインダウト・トランザクションを手動で強制的にコミットする方法は 2 つあります。DBA は、Enterprise Manager の「Transaction Object List」の「Force Commit」オプションを使用するか、または、SQL コマンド COMMIT に、FORCE オプションおよびコミットするインダウト・トランザクションのローカルまたはグローバル・トランザクション ID を示すテキスト文字列を指定して使用できます。

## Enterprise Manager でのコミットまたはロールバックの強制

インダウト・トランザクションをコミットするには、「Transaction Object List」から該当のトランザクションを選択し、「Transaction」メニューから「Force Commit」を選択します。

インダウト・トランザクションをロールバックするには、「Transaction Object List」から該当のトランザクションを選択し、「Transaction」メニューから「Force Rollback」を選択します。

**注意：** インダウト・トランザクションは、セーブポイントまでロールバックできません。

## インダウト・トランザクションの手動コミットまたは手動ロールバック

次の SQL 文を使用して、インダウト・トランザクションをコミットします。

```
COMMIT FORCE 'transaction_name';
```

インダウト・トランザクションを手動ロールバックするには、SQL コマンド ROLLBACK を使用し、FORCE オプションと、ロールバックするインダウト・トランザクションのローカルまたはグローバル・トランザクション ID を示すテキスト文字列を指定します。たとえば、2.9.4 というローカル・トランザクション ID を持つインダウト・トランザクションをロールバックするには、次の文を使用します。

```
ROLLBACK FORCE '2.9.4';
```

**注意：** インダウト・トランザクションは、セーブポイントまでロールバックできません。

### インダウト・トランザクションの手動コミットまたはロールバックに必要な権限

自分自身が発行したインダウト・トランザクションを手動で強制的にコミットまたはロールバックするには、FORCE TRANSACTION システム権限が必要です。他のユーザーの分散トランザクションを強制的にコミットまたはロールバックするには、FORCE ANY TRANSACTION システム権限が必要がです。どちらの権限も、明示的に取得するか、またはロールを介して取得できます。

---

**注意：** インダウト分散トランザクションの強制コミットまたは強制ロールバックは、オペレータの現行トランザクションには影響しません。

---

### ローカル保留トランザクション表とロールバック / コミットの強制

すべての例で、トランザクションはローカル・ノードでコミットまたはロールバックされ、ローカル保留トランザクション表は、このトランザクション行の STATE 列に強制コミットまたは強制異常終了の値を記録します。

### SCN の指定

トランザクションのコミットを強制する場合、オプションで、トランザクションの SCN を指定することができます。この機能を使用すると、他のノードでコミットされたときに割り当てられた SCN を使用してインダウト・トランザクションをコミットできます。

この方法で、障害が発生した場合でも、分散トランザクションのコミット時刻の同期を保つことができます。SCN を指定するのは、別のノードですでにコミットされている同じトランザクションの SCN を判断できる場合のみにしてください。

たとえば、*global_id* というグローバル・トランザクション ID を持つトランザクションを手動でコミットするとします。最初に、問題のトランザクションに参加したリモート・データベースの DBA_2PC_PENDING ビューを問い合せます。

そのノードでトランザクションのコミットに使用された SCN をメモします。ローカル・ノードでそのトランザクションをコミットするときに、それと同じ SCN（10 進数）を指定します。たとえば、SCN が 829381993 であれば、次のコマンドを使用します。

```
COMMIT FORCE 'global_id', 829381993;
```

## 接続保持時間の変更

分散トランザクションが失敗した場合、ローカル・サイトからリモート・サイトへの接続がすぐにはクローズしないようにできます。接続はクローズせずに、通信がすぐに回復した場合に接続を再確立しなくてもよいように、オープンのままになります。接続をオープンにしておく時間の長さは、データベース・パラメータ DISTRIBUTED_RECOVERY_CONNECTION_HOLD_TIME を使用して設定できます。

大きな値を設定すると、障害の後で再接続するためのコストは最小限になりますが、ローカル・データベースが消費するリソースが増大します。逆に、値が小さいと、障害発生中にロックされたままになるリソースのコストは最小限になりますが、障害後の再接続のコストが増大します。このパラメータのデフォルト値は 200 秒です。詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

## 分散トランザクション数の制限の設定

データベース・パラメータ DISTRIBUTED_TRANSACTIONS は、1 つのデータベースが参加できる分散トランザクションの最大数を設定します。データベースが多数の分散トランザクションの一部である場合は、このパラメータを大きい値に設定する必要があります。デフォルト値はオペレーティング・システムに応じて異なります。

これに対して、サイトで異常に多数のネットワーク障害が発生している場合は、このパラメータの値を一時的に小さくすることができます。これにより、サイトが参加するインダウト・トランザクションの数を制限し、サイトのロック・データの量、および解決を要するインダウト・トランザクションの数を制限できます。

このパラメータの詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

# 分散トランザクション回復機能のテスト

必要があれば、分散トランザクション障害を強制的に発生させて、RECO がトランザクションのローカル部分を自動的に解決するのを観察できます。また、分散トランザクションを強制的に失敗させて、インダウト分散トランザクションを手動で解決し、結果を観察することもできます。

次の各項では、使用可能な機能と、その操作を実行するためのステップについて説明します。

## 分散トランザクションの強制終了

COMMIT 文の COMMENT パラメータには、コメントを組み込むことができます。分散トランザクションの 2 フェーズ・コミットのフェーズ中に意図的に失敗させるには、次のコメントを COMMENT パラメータに組み込みます。

```
COMMIT COMMENT 'ORA-2PC-CRASH-TEST-n';
```

*n* は、次の整数の 1 つです。

| n | 結果 |
|---|---|
| 1 | 収集の後で、コミット・ポイント・サイトをクラッシュさせます。 |
| 2 | 収集の後で、非コミット・ポイント・サイトをクラッシュさせます。 |
| 3 | 準備前にクラッシュさせます（非コミット・ポイント・サイト）。 |
| 4 | 準備後にクラッシュさせます（非コミット・ポイント・サイト）。 |
| 5 | コミット前に、コミット・ポイント・サイトをクラッシュさせます。 |
| 6 | コミット後に、コミット・ポイント・サイトをクラッシュさせます。 |
| 7 | コミット前に、非コミット・ポイント・サイトをクラッシュさせます。 |
| 8 | コミット後に、非コミット・ポイント・サイトをクラッシュさせます。 |
| 9 | 記憶消去前に、コミット・ポイント・サイトをクラッシュさせます。 |
| 10 | 記憶消去前に、非コミット・ポイント・サイトをクラッシュさせます。 |

たとえば、ローカル・コミット・ポイント強度がリモート・コミット・ポイント強度より大きく、両方のノードが更新される場合は、次の文を発行すると、次に示すようなメッセージが戻されます。

```
COMMIT COMMENT 'ORA-2PC-CRASH-TEST-7';

ORA-02054: transaction #.##.## in-doubt
ORA-02059: ORA-CRASH-TEST-n in commit comment
```

この時点で、該当のインダウト分散トランザクションが DBA_2PC_PENDING ビューに表示されます。RECO は、使用可能になっていれば、自動的にトランザクションを迅速に解決します。

### 2 フェーズ・コミットを意図的に失敗させるために必要な権限

前述のコメントを使用して 2 フェーズ・コミット・フェーズの失敗を意図的に発生させることができるのは、ローカル・セッションとリモート・セッションが FORCE ANY_TRANSACTION システム権限を持っている場合のみです。この権限を持っていない場合は、クラッシュ・コメントを指定した COMMIT 文を発行すると、エラーが戻されます。

## リカバラ（RECO）のバック グラウンド・プロセス

Oracle8*i* インスタンスの RECO バックグラウンド・プロセスは、分散トランザクションに関係する障害を自動的に解決します。ノードの RECO バックグラウンド・プロセスは、指数関数的に増加する時間間隔で、インダウト分散トランザクションのローカル部分の回復を試みます。

RECO は、失敗したトランザクションに参加していた他のノードへの接続のために、既存接続を使用するかまたは新規接続を確立することができます。接続が確立されると、RECO は自動的にすべてのインダウト・トランザクションを解決します。解決済みのインダウト・トランザクションに対応する行はすべて、各データベースの保留トランザクション表から自動的に除去されます。

## RECO を使用禁止または 使用可能にする方法

リカバラ・バックグラウンド・プロセス RECO は、ALTER SYSTEM コマンドに ENABLE/DISABLE_DISTRIBUTED_RECOVERY オプションを指定して、それぞれ使用可能または使用禁止にすることができます。たとえば、2 フェーズ・コミットの失敗を強行して、一時的に RECO を使用禁止にし、インダウト・トランザクションを手動で解決するとします。次の文は、RECO を使用禁止にします。

```
ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;
```

逆に、次の文は、RECO を使用可能にしてインダウト・トランザクションが自動的に解決されるようにします。

```
ALTER SYSTEM ENABLE DISTRIBUTED RECOVERY;
```

---

**注意：** シングル・プロセス・インスタンス（MS-DOS が動作している PC など）には、独立したバックグラウンド・プロセスがないため、RECO プロセスはありません。したがって、分散システムに参加するシングル・プロセス・インスタンスを起動する場合は、前述の文を使用して分散回復を手動で使用可能にする必要があります。

---

**追加情報：** シングル・プロセス・インスタンスの分散トランザクションの回復の詳細は、Oracle オペレーティング・システム固有のマニュアルを参照してください。

# 4

# 分散システムのアプリケーション開発

この章では、分散システムで実行するアプリケーションを設計する場合に必要となる、特殊な考慮事項について説明します。Oracle では、特別に分散環境で動作するようにアプリケーションを設計する必要はほとんどありません。詳細は、『Oracle8*i* 概要』を参照してください。

この章で説明する項目は次のとおりです。

- アプリケーションのデータの分散に影響する要因
- データベース・リンクにより確立される接続の制御
- 分散システムの参照整合性
- 分散問合せ
- リモート・プロシージャのエラー処理

Oracle8*i* アプリケーションのインプリメントの詳細は、『Oracle8*i* 管理者ガイド』を参照してください。この章では、Oracle8*i* 分散データベース環境を使用する場合の開発に固有の情報のみを説明します。Oracle 環境におけるアプリケーション開発の詳細は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』も参照してください。

# アプリケーションのデータの分散に影響する要因

分散データベース環境では、開発者はデータベース管理者と協議して、データの最も適切な格納場所を判断する必要があります。そのときに考慮すべき主な事項には次のものがあります。

- 各位置からポストされるトランザクションの数
- 各ノードで使用するデータ（表の部分）の量
- ネットワークのパフォーマンス特性および信頼性
- 各ノードの速度、ディスクの容量
- 使用できなくなった場合のノードまたはリンクの重要度
- 表相互の参照整合性の必要度

# データベース・リンクにより確立される接続の制御

SQL 文またはリモート・プロシージャ・コールでグローバル・オブジェクト名が参照されると、ローカル・ユーザーにかわって、データベース・リンクがリモート・データベース内のセッションとの接続を確立します。リモート接続およびセッションが作成されるのは、その接続がまだローカル・ユーザー・セッション用として確立されていない場合のみです。

リモート・データベースに対して確立された接続およびセッションは、アプリケーション（またはユーザー）が明示的にそれを終了させない限り、ローカル・ユーザーのセッションが終了するまで持続します。確立されたリモート接続をデータベース・リンクにより終了する方法は、アプリケーションで不要になったコストの高い接続（長距離電話接続など）を切断するときに役立ちます。

アプリケーション開発者またはユーザーは、CLOSE DATABASE LINK パラメータを指定した ALTER SESSION コマンドを使用して、リモート接続およびセッションをクローズ（終了）できます。たとえば、次の問合せを発行したとします。

```
SELECT * FROM emp@sales;
COMMIT;
```

次の文は、SALES データベース・リンクが指し示すリモート・データベース内のセッションを終了させます。

```
ALTER SESSION CLOSE DATABASE LINK sales;
```

ユーザー・セッションでのデータベース・リンク接続をクローズするには、ALTER SESSION システム権限が必要です。

**注意：** データベース・リンクをクローズする前に、リンクに使用したカーソルをすべてクローズし、リンクを使用する現行のトランザクションがあれば、それも終了する必要があります。

# 分散システムの参照整合性

Oracle では、宣言参照整合性制約を、分散システムの複数ノードにまたがって定義することはできません（つまり、ある表についての宣言参照整合性制約で、リモート表の主キーまたは一意キーを参照する外部キーを指定することはできません）。ただし、ノード間にわたる親子表の関連は、トリガーを使用して維持できます。参照整合性を適用するトリガーの詳細は、『Oracle8*i* 概要』を参照してください。

**注意：** トリガーを使用して、分散データベースの複数のノードで参照整合性の定義を決定する場合は、ネットワーク障害によって親表だけでなく子表のアクセス可能性も制限されることに注意してください。

たとえば、子表が SALES データベース内にあり、親表が HQ データベース内にあるとします。この 2 つのデータベース間のネットワーク接続に障害が生じると、子表を対象とするある種の DML 文（子表に行を挿入するもの、または子表の中の外部キー値を更新するもの）は、処理を続行できなくなります。これは、参照整合性トリガーが、HQ データベース内の親表にアクセスする必要があるためです。

# 分散問合せ

分散問合せは、ローカルの Oracle により、対応するいくつかのリモート問合せに分解され、実行するためにリモート・ノードに送られます。リモート・ノードでは問合せを実行し、その結果をローカル・ノードに送り返します。これを受けたローカル・ノードでは、必要な後処理があればそれを実行し、その結果をユーザーまたはアプリケーションに戻します。

たとえば、整合性制約違反が原因で分散文の一部が失敗した場合、Oracle はエラー番号 ORA-02055 を戻します。後続の文またはプロシージャ・コールでは、ロールバックまたはセーブポイントまでのロールバックが発行されるまでは、エラー番号 ORA-02067 が戻されます。

アプリケーションを設計するときは、分散更新の一部が失敗したことを示すエラー・メッセージが戻されたかどうかをチェックするように設定しておく必要があります。障害が検出された場合は、アプリケーションを先へ進める前に、トランザクション全体のロールバック（またはセーブポイントまでのロールバック）をする必要があります。

## 分散問合せのチューニング

分散問合せを最適化する最も効率的な方法は、リモート・データベースへのアクセスを最小限に抑え、必要なデータのみを取り出すことです。特に、分散問合せで2つの異なるリモート・データベースから5つのリモート表を参照する場合に、複雑なフィルタ（WHERE r1.salary + r2.salary > 50000など）を使用していると、問合せをリライトすることによりパフォーマンスを改善できます。具体的には、リモート・データベースに一度にアクセスを行うようにし、リモート・サイトでフィルタを適用するようにリライトします（これにより、問合せ実行サイトに転送されるデータ量が減少します）。そのリライトのためには、「連結インライン・ビュー」を使用します。

このことを念頭において、次の用語を定義する必要があります。

- **連結**：2つ以上の表が同じデータベースに配置されること。
- **インライン・ビュー**：親SELECT文中の表を代替するSELECT文。次の埋込みSELECT文（太字）は、インライン・ビューの一例です。

```
SELECT e.empno, e.ename, d.deptno, d.dname
   FROM (SELECT empno, ename from emp@orc1.world) e, dept d;
```

- **連結インライン・ビュー**：単一データベースからのみ複数の表のデータを選択するインライン・ビュー（リモート・データベースへのアクセス回数が減少し、分散問合せのパフォーマンスが改善されます）。

分散問合せはどのように記述してもかまいませんが、分散問合せのパフォーマンスを高めるために、できる限り連結インライン・ビューを使用することをお薦めします。

Oracleのコストベース最適化では、連結インライン・ビューによるパフォーマンス向上が反映されるように、透過的に多くの分散問合せをリライトします。

## コストベース最適化

コストベース最適化手法では、連結インライン・ビューを使用して問合せをリライトできるだけでなく、参照表から収集された統計と、オプティマイザによって実行される計算に従って、分散問合せを最適化できます。たとえば、コストベースの最適化によって、次の問合せが分析されます（CREATE TABLE文の内側の問合せが分析されることに注目してください）。

```
CREATE TABLE AS (SELECT l.a, l.b, r1.c, r1.d, r1.e, r2.b, r2.c
   FROM local l, remote1 r1, remote2 r2
      WHERE l.c = r.c AND r1.c = r2.c AND r.e > 300);
```

この問合せは、次のようにリライトされます。

```
CREATE TABLE AS (SELECT l.a, l.b, v.c, v.d, v.e
   FROM (SELECT r1.c, r1.d, r1.e, r2.b, r2.c FROM remote1 r1, remote2 r2
      WHERE r1.c = r2.c AND r1.e > 300) v, local l
   WHERE l.c = r1.c);
```

インライン・ビューに別名 V が割り当てられ、この SELECT 文中で表として参照できます。連結インライン・ビューの作成によって、リモート・サイトで実行される問合せの量が減少し、コストを伴うネットワーク・トラフィックも減少します。

## コストベース最適化のセットアップ

分散問合せのパフォーマンスを改善するため、コストベース最適化を使用するようにシステムをセットアップすると、この操作はユーザーに対して透過的になります。つまり、問合せの発行時に最適化が自動的に発生します（他のタイプの問合せのパフォーマンスも改善されます。詳細は、『Oracle8*i* チューニング』を参照）。

Oracle のオプティマイザを活用するようにシステムをセットアップするには、次のタスクを完了する必要があります。

- データベース環境のセットアップ
- 表の分析（表の統計を生成するため）

**環境のセットアップ** コストベース最適化を使用可能にするには、OPTIMIZER_MODE パラメータを CHOOSE または COST に設定する必要があります。このパラメータは、パラメータ・ファイル（INIT.ORA）内で OPTIMZER_MODE パラメータを変更して永続的に設定する方法と、ALTER SESSION コマンドを発行してセッション・レベルで設定する方法があります。

パラメータ・ファイル（INIT.ORA）内で OPTIMZER_MODE パラメータを設定する方法は、『Oracle8*i* チューニング』を参照してください。

OPTIMIZER_MODE をセッション・レベルで設定するには、次の ALTER SESSION 文を発行します（この設定は、現行のセッション中のみ有効です）。

```
ALTER SESSION OPTIMIZER_MODE = CHOOSE;
```

または

```
ALTER SESSION OPTIMIZER_MODE = COST;
```

コストベース最適化方法を使用するようにシステムを構成する方法の詳細は、『Oracle8*i* チューニング』を参照してください。

**表の分析** コストベース最適化で問合せに最も効率的なパスを選択させるには、分散問合せに関係する表について正確な統計を提供する必要があります。

表の統計を生成するには、ANALYZE コマンドを実行するのが最も簡単な方法です。たとえば、分散問合せで EMP 表と DEPT 表を参照する場合は、次のコマンドを実行して必要な統計を生成します。

```
ANALYZE TABLE emp COMPUTE STATISTICS;
ANALYZE TABLE dept COMPUTE STATISTICS;
```

**注意：** ANALYZE コマンドを実行するには、表に関してローカルに接続する必要があります。次のコマンドは実行できません。

```
ANALYZE TABLE remote@remote.com COMPUTE STATISTICS;
```

先にリモート・サイトに接続してから、前述の ANALYZE 文を実行してください。

ANALYZE 文の使用方法に関する追加情報は、『Oracle8*i* SQL リファレンス』を参照してください。

一度に複数のオブジェクトに関する統計を生成する方法は、『Oracle8*i* チューニング』の「統計情報の作成」を参照してください。また、統計を最新の状態に保つプロセスを自動化し、コストベース最適化のパフォーマンスと精度を高める方法は、『Oracle8*i* チューニング』の「自動統計収集」を参照してください。

## コストベース最適化の機能

「分散問合せのチューニング」で説明したように、オプティマイザの主なタスクは、連結インライン・ビューを使用するように分散問合せをリライトすることです。この最適化は、次の 3 つのステップで実行されます。

1. マージ可能なビューがすべてマージされます。
2. 連結問合せブロックのテストが実行されます。
3. 連結インライン・ビューを使用してオプティマイザで問合せがリライトされます。

問合せは、リライト後に実行され、ユーザーにデータ・セットが戻されます。

**コストベース最適化の制限事項** コストベース最適化はユーザーにとって透過的に実行されますが、この方法ではパフォーマンスを改善できない分散問合せの使用例もいくつかあります。特に、分散問合せに次のいずれかが含まれている場合、コストベース最適化は非効率的です。

- 集約操作
- 副問合せ
- 複合 SQL

このうち 1 つでも分散問合せに含まれている場合は、必ず「ヒントを使用したコストベース最適化の拡張」を読んでください。問合せを変更し、ヒントを使用して分散問合せのパフォーマンスを改善する方法が記載されています。

## ヒントを使用したコストベース最適化の拡張

オプティマイザで処理できない分散問合せ（「コストベース最適化の制限事項」を参照）がある場合は、ヒントを使用してコストベース最適化の機能を拡張できます。特に、連結インライン・ビューを使用する独自の問合せを記述する場合は、CBO に対して分散問合せをリライトしないように指示する必要があります。

また、データベース環境に関して特殊な知識（統計、負荷、ネットワークおよび CPU の制限、分散問合せなど）を持っている場合は、コストベース最適化のガイドとなるヒントを指定できます。

分散問合せを最適化するために、その問合せに関する知識に基づいてヒントを提供します。特に、データベース環境に関する知識に基づく連結インライン・ビューを使用して、独自に最適化した問合せを記述している場合は、オプティマイザでリライトされないように NO_MERGE ヒントを指定してください。

このテクニックが特に役立つのは、分散問合せが集約操作、副問合せまたは複合 SQL を含んでいる場合です。この種の分散問合せはオプティマイザでリライトできないため、NO_MERGE を指定すると、4-6 ページの「コストベース最適化の機能」で説明したステップがスキップされます。

DRIVING_SITE ヒントを使用すると、問合せ実行サイトの役割を果たすリモート・サイトを定義できます。このテクニックが特に役立つのは、多量のデータがリモート・サイトにあり、そのリモート・サイトから問合せを実行し、結果として得られたデータ・セットをローカル・サイトに戻す方が効率的な場合です。

**NO_MERGE** NO_MERGE ヒントを指定すると、インライン・ビューは非連結 SQL 文にマージされなくなります（4-6 ページの「コストベース最適化の機能」のステップ 1 を参照）。このヒントは SELECT 文に埋め込むことができます。ヒントは、引数としてインライン・ビューを指定して SELECT 文の先頭に配置するか、インライン・ビューを定義する問合せブロックに挿入できます。

**引数を使用する場合：**

```
SELECT /*+NO_MERGE(v)*/ t1.x, v.avg_y
   FROM t1, (SELECT x, AVG(y) AS avg_y FROM t2 GROUP BY x) v,
   WHERE t1.x = v.x AND t1.y = 1;
```

**問合せブロック内：**

```
SELECT t1.x, v.avg_y
   FROM t1, (SELECT /*+NO_MERGE*/ x, AVG(y) AS avg_y FROM t2 GROUP BY x) v,
   WHERE t1.x = v.x AND t1.y = 1;
```

データベース環境に関する知識に基づいて、最適化された問合せを開発している場合は、このヒントを使用してください。詳細は、『Oracle8*i* チューニング』の「NO_MERGE」を参照してください。

**DRIVING_SITE** DRIVING_SITE ヒントを使用すると、問合せを実行するサイトを指定できます。できるだけ、コストベース最適化に実行場所を判断させることをお薦めします。ただし、オプティマイザを上書きする場合（統計が陳腐化した場合や、特定のマシンのパフォーマンスが大幅に低下した場合など）は、DRIVING_SITE ヒントで実行サイトを指定できます。DRIVING_SITE ヒントを含む SELECT 文は、次のようになります。

```
SELECT /*+DRIVING_SITE(dept)*/ * FROM emp, dept@remote.com
   WHERE emp.deptno = dept.deptno;
```

詳細は、『Oracle8*i* チューニング』の「DRIVING_SITE」を参照してください。分散問合せのチューニングの詳細は、4-4 ページの「分散問合せのチューニング」および『Oracle8*i* チューニング』を参照してください。

# 最適化の検証

分散問合せのチューニングでは、問合せの実行計画を分析する作業が重要です。分析から受け取るフィードバックは、データベースをテストして検証する上で重要な要素となります。コストベース最適化によって最適化する分散問合せの実行計画を、手動（ヒントの使用、連結インライン・ビューの定義など）で最適化する分散問合せの実行計画と比較する場合は、この検証作業がさらに重要です。実行計画、EXPLAIN PLAN コマンドおよび結果の解釈方法の詳細は、『Oracle8*i* チューニング』を参照してください。

## データベースの準備

分散問合せの実行計画を表示するには、その実行計画を格納するためのデータベースを準備する必要があります。この準備は、スクリプトを実行すれば簡単です。次のスクリプトを実行し、データベースに実行計画を格納できるように準備してください。

```
@utlxplan.sql
```

> **注意：** UTLXPLAN.SQL ファイルの位置は、オペレーティング・システムによって異なります。

UTLXPLAN.SQL ファイルを実行すると、実行プランを一時的に格納するための PLAN_TABLE が現行のスキーマ内に作成されます。

## 実行計画の生成

実行計画を格納するためのデータベースを準備すれば、指定した問合せの実行計画を表示する準備ができたことになります。SQL 文を直接実行するかわりに、EXPLAIN PLAN FOR 句に文を指定してください。たとえば、次のように実行します。

```
EXPLAIN PLAN FOR
   SELECT d.dname FROM dept d
      WHERE d.deptno IN
         (SELECT deptno FROM emp@orc2.world
            GROUP BY deptno
               HAVING COUNT (deptno) >3);
```

## 実行計画の表示

前述の SQL 文を実行すると、実行計画は前に作成した PLAN_TABLE に一時的に格納されます。実行計画の結果を表示するには、次のスクリプトを実行します。

```
@utlxpls.sql
```

> **注意：** UTLXPLS.SQL ファイルの位置は、オペレーティング・システムによって異なります。

UTLXPLS.SQL ファイルを実行すると、指定した SELECT 文の実行計画が表示されます。結果は、次のような形式で表示されます。

```
Plan Table
--------------------------------------------------------------------------------
| Operation                 | Name     | Rows | Bytes| Cost  | Pstart| Pstop |
--------------------------------------------------------------------------------
| SELECT STATEMENT          |          |      |      |       |       |       |
|  NESTED LOOPS             |          |      |      |       |       |       |
|   VIEW                    |          |      |      |       |       |       |
|    REMOTE                 |          |      |      |       |       |       |
|   TABLE ACCESS BY INDEX RO|DEPT      |      |      |       |       |       |
|    INDEX UNIQUE SCAN      |PK_DEPT   |      |      |       |       |       |
--------------------------------------------------------------------------------
```

独自の連結インライン・ビューを記述したりヒントを使用して、分散問合せを手動で最適化する場合は、手動による最適化の前後に実行計画を生成することをお薦めします。2 つの実行計画を使用して、手動による最適化の効率を比較し、必要に応じて最適化に変更を加えて、分散問合せのパフォーマンスを改善できます。

リモート・サイトで実行される SQL 文を表示する場合は、次の SELECT 文を実行します。

```
SELECT other FROM plan_table WHERE operation = 'REMOTE';
```

出力は、次のようになります。

```
SELECT DISTINCT "A1"."DEPTNO" FROM "EMP" "A1"
   GROUP BY "A1"."DEPTNO" HAVING COUNT("A1"."DEPTNO")>3
```

**注意：** OTHER 列の内容全体を表示できない場合は、次のように実行する必要があります。

```
SET LONG 9999999
```

# リモート・プロシージャのエラー処理

プロシージャをローカルまたはリモート位置で実行する場合、次の 4 種類の例外状態が生じることがあります。

- PL/SQL のユーザー定義の例外。これは、EXCEPTION キーワードを使用して宣言する必要があります。
- PL/SQL の事前定義の例外。たとえば NO_DATA_FOUND キーワードなどです。
- SQL エラー。たとえば ORA-00900 および ORA-02015 などです。
- アプリケーション例外。これは、RAISE_APPLICATION_ERROR() プロシージャにより生成されます。

ローカル・プロシージャを使用しているときは、次の例に示すような例外ハンドラを作成することにより、上記のすべてのメッセージを検出できます。

```
BEGIN
  ...
EXCEPTION
   WHEN ZERO_DIVIDE THEN
   /* ...handle the exception */
END;
```

WHEN 句には例外名が必要である点に注意してください。発生した例外に名前がない場合は ( たとえば RAISE_APPLICATION_ERROR で生成される例外など)、次の例に示すように、PRAGMA_EXCEPTION_INIT を使用して名前を割り当てることができます。

```
DECLARE
  null_salary EXCEPTION;
  PRAGMA EXCEPTION_INIT(null_salary, -20101);
BEGIN
  ...
  RAISE_APPLICATION_ERROR(-20101, 'salary is missing');
...
EXCEPTION
  WHEN null_salary THEN
  ...
END;
```

リモート・プロシージャをコールすると、ローカル・プロシージャの例外ハンドラによって例外が処理されます。コール側プロシージャが、前述の例のように例外を処理できるように、リモート・プロシージャは、ローカルのコール側プロシージャにエラー番号を戻す必要があります。エラー番号を戻さなければ、PL/SQL でユーザー定義された例外は、ローカル・プロシージャに常に ORA-06510 を戻すことに注意してください。

したがって、ユーザー定義された 2 つの異なる例外については、エラー数値に基づいた区別はできません。その他のリモート例外は、すべてローカルな例外と同じ方法で処理できます。

# 第II部

## 異機種間サービス

# 5

# Oracle の異機種間サービスの理解

この章では、Oracle の異機種間サービスの基本概念について説明します。この章で説明する項目は次のとおりです。

- 異機種間サービス
- 異機種間サービスのサービス
- 異機種間サービスの使用

# 異機種間サービス

異機種間サービスは、Oracle8*i* Server 内の統合化されたコンポーネントで、Oracle サーバーから非 Oracle システムにアクセスするための一般的なテクノロジを提供します。異機種間サービスは、次のことができます。

- Oracle SQL を使用すると、Oracle サーバー内にデータが格納されているかのように、非 Oracle システムに格納されたデータに透過的にアクセスできます。
- Oracle プロシージャ・コールを使用すると、Oracle 分散環境から、非 Oracle システム、非 Oracle サービスまたはアプリケーション・プログラム・インタフェース（API）に透過的にアクセスできます。

特定の非 Oracle システムにアクセスするには、異機種間サービス・エージェントが必要になります。

> **注意：**「非 Oracle システム」という用語は、SQL を使用してアクセスされる Oracle 以外のデータストア（データベース）と、プロシージャ的にアクセスされるシステムの両方を意味します。

## 異機種間サービス・エージェント

異機種間サービスは、Oracle8*i* Server の一般的なテクノロジを提供しますが、特定の非 Oracle システムにアクセスする場合は、異機種間サービス・エージェントが必要になります。異機種間サービス・エージェントは、Oracle Open Gateway バージョン 8 以降で提供予定です。

Oracle Open Gateway は、異機種間サービスを使用する製品ファミリの 1 つです。その他の異機種間サービスを基盤とした製品も開発される予定です。それらの製品は、オラクルまたはサードパーティによって開発され、Oracle Open Gateway の製品ファミリには含まれない場合もあります。「異機種間サービス・エージェント」という用語は、異機種間サービスを基盤とするすべての製品を示し、Oracle Open Gateway もその中に含まれます。

# 異機種間サービスのサービス

異機種間サービスには 3 つのサービスがあります。

- トランザクション・サービス
- SQL サービス
- プロシージャ・サービス

## トランザクション・サービス

「トランザクション・サービス」を使用すると、非 Oracle システムを Oracle のトランザクションおよびセッションに統合できます。ユーザーは、Oracle のユーザー・セッション中に、データベース・リンクを介して非 Oracle システムに初めてアクセスするときに、非 Oracle システム内で認証された（ユーザー名とパスワード）セッションを透過的にセットアップできます。非 Oracle システムのセッションは、Oracle のユーザー・セッションが終了したときに透過的に終了します。さらに、1 つ以上の非 Oracle システムが Oracle 分散トランザクションに参加できます。アプリケーションがトランザクションをコミットすると、Oracle の 2 フェーズ・コミット・プロトコルが非 Oracle システムにアクセスして、分散トランザクションを透過的に調整します。実際は、非 Oracle システムそのものが 2 フェーズ・コミットをサポートしない場合でも、Oracle サーバーが非 Oracle システムとの分散トランザクションをサポートします。

SQL サービスおよびプロシージャ・サービスの両方で、トランザクション・サービスを使用します。Oracle のオブジェクト・トランザクション・サービスでは、トランザクション・サービスのインプリメントのみを行うエージェントを使用します。

異機種間分散トランザクションの詳細は、6-9 ページの「トランザクション・サービスのビュー」を参照してください。

## SQL サービス

「SQL サービス」は、SQL を使用して非 Oracle システムに透過的にアクセスするために使用されます。アプリケーションの SQL 要求に非 Oracle システムのデータが必要な場合、異機種間サービスによって、Oracle の SQL 要求は非 Oracle システムが理解できる SQL 要求に変換され、非 Oracle データがアクセスされ、そのデータが Oracle Server の処理（後処理）用に使用可能になります。

SQL サービスには次の機能があります。

- Oracle の SQL を非 Oracle システムが理解できる SQL 言語に変換します。
- Oracle のデータ・ディクショナリ表の SQL 要求を、非 Oracle システムのデータ・ディクショナリの要求に変換します。
- 非 Oracle システムのデータ型を Oracle のデータ型にマップします。

## プロシージャ・サービス

異機種間サービスを使用すると、ユーザーは、Oracle8*i* Server から、メッセージング・システムやキューイング・システムなどの任意のプロシージャ型非 Oracle システムにアクセスできます。非 Oracle システムは、PL/SQL リモート・プロシージャ・コール（RPC）を使用して Oracle サーバーからコールされます。異機種間サービスは、PL/SQL のコールを、非 Oracle システムのプロシージャまたはファンクションに変換します。

プロシージャ・サービスを使用すると、分散外部プロシージャを作成できるため、第三世代言語（3GL）のルーチンを PL/SQL からコールすることが可能になります。PL/SQL の外部プロシージャと同様、分散外部プロシージャは、PL/SQL のプロシージャおよびファンクションの名前と引数を 3GL のルーチン名と引数にマップします。外部プロシージャと分散外部プロシージャは、同じメカニズムを使用して、PL/SQL から 3GL のルーチンをコールします。外部プロシージャは、Oracle8*i* Server に対して、ローカルで特定の目的のタスクを実行するために設計されていますが、分散外部プロシージャは、非 Oracle システムにアクセスするために設計されています。分散外部プロシージャと外部プロシージャの主な違いは次のとおりです。

- 分散外部プロシージャによって、Oracle8*i* Server は非 Oracle システムで認証されたセッションを開始し、非 Oracle システムとの分散トランザクションを調整できます。
- 分散外部プロシージャは、リモート・システム上で実行されると見なされるため、分散外部プロシージャは、データベース・リンクを介して起動されます。

PL/SQL の外部プロシージャの詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

# 異機種間サービスの使用

異機種間サービスを使用すると、非 Oracle システムがリモート Oracle サーバーとして認識されます。非 Oracle システムの表をアクセスまたは操作したり、プロシージャを実行するには、非 Oracle システムへのデータベース・リンクを作成します。非 Oracle システムの表およびプロシージャは、データベース・リンクを使用して表およびプロシージャを修飾してアクセスできます。これは、Oracle のリモート・サーバーの表およびプロシージャにアクセスする方法と同じです。

非 Oracle システムが参照される場合は、異機種間サービスが、SQL 文または PL/SQL リモート・プロシージャ・コールを非 Oracle システムに適切な文に変換します。

たとえば、非 Oracle システムがアクセスされるときは、データベース・リンクを介します。

```
SELECT *
FROM EMP@non_Oracle_system;
```

異機種間サービスは、Oracle SQL 文を非 Oracle システムの SQL に変換し、非 Oracle システムで SQL 文を実行します。

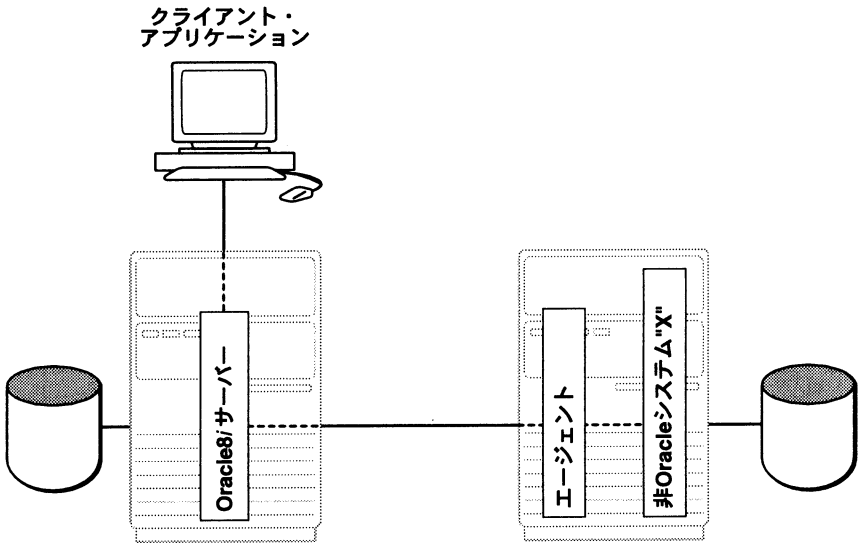
# 異機種間サービス・プロセスのアーキテクチャ

Oracle8*i* Server から特定の非 Oracle システムにアクセスするには、エージェントが必要です。Oracle サーバーはエージェントと通信し、エージェントが特定の非 Oracle システムと通信します。

図 5-1 でに示すように、エージェントは非 Oracle システムと同じマシンに常駐できますが、必ずそこに常駐する必要はありません。また、エージェントは、Oracle8*i* Server と同じマシンに常駐したり、その他のマシンに常駐することもできます。エージェントは、Net8 を介して Oracle8*i* Server からアクセスできる必要があります。また、エージェントは、非 Oracle システム固有の通信メカニズムを使用して、非 Oracle システムからアクセスできる必要があります。

ユーザー・セッションが Oracle8*i* Server のデータベース・リンクを通じて非 Oracle システムにアクセスすると、Net8 リスナーはエージェント・プロセスを開始します。このエージェント・プロセスは、ユーザー・セッションが切断されるまで、またはデータベース・リンクが明示的にクローズされるまで、実行し続けます。

**図 5-1 異機種非 Oracle システムへのアクセス**



## 分散外部プロシージャのプロセス・アーキテクチャ

分散外部プロシージャによって、PL/SQL のプロシージャは、リモートの動的リンク・ライブラリ（DLL）内の 3GL ルーチンにマップされます。分散外部プロシージャが実行されると、エージェントによって、3GL のルーチンが含まれるオペレーティング・システムの動的リンク・ライブラリがエージェント・プロセスにロードされ、PL/SQL のプロシージャが 3GL のルーチンにマップされ、3GL のルーチンが起動されます。3GL のルーチンの処理が終了したら、引数と戻り値がコール側の PL/SQL プログラムに戻されます。図 5-2 を参照してください。

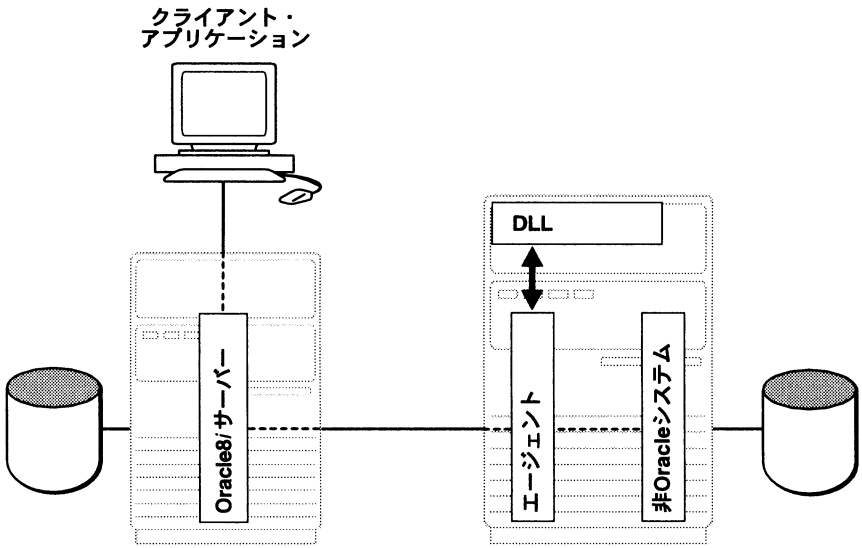
---

**注意：** プラットフォームによっては、動的リンク・ライブラリが「共有ライブラリ」として参照されます。

---

分散外部プロシージャを使用して 非 Oracle システムにアクセスするには、その非 Oracle システム用に特別に設計されたエージェントが必要です。そのエージェントには、非 Oracle システムでセッションをセットアップするための非 Oracle システム固有のコードが含まれ、分散外部プロシージャによって非 Oracle システムで実行されたトランザクションが Oracle 分散トランザクションに統合されます。

**図 5-2 Oracle8*i*、エージェントと動的ライブラリ**




**DLL = ダイナミック・リンク・ライブラリ**

たとえば、キューイング・システムにアクセス可能なエージェントがあるとします。キューにメッセージを入れるには、Oracle のアプリケーションで次の文を実行します。

```
SQL> EXECUTE enqueue@queuing_system('We are out of stock');
SQL> COMMIT;
```

エンキュー・プロシージャは、動的リンク・ライブラリに存在します。前述の文が実行されると、Net8 リスナーがエージェント・プロセスを作成します。エージェント・プロセスではエンキュー・プロシージャが含まれる DLL をロードし、エンキュー・プロシージャを実行して、キューイング・システムにメッセージを書き出します。トランザクションを COMMIT すると、エージェントは Oracle サーバーの代理として、キューイング・システムにトランザクションのコミットを要求します。

Oracle ユーザー・セッションの継続中、または "ALTER SESSION CLOSE DATABASE LINK queing_system" コマンドで明示的にデータベース・リンクをクローズするまで、エージェント・プロセスの実行は継続されます。

# 6

# Oracle 異機種間サービスの管理

この章では、異機種間の分散環境をメンテナンスするために必要なデータベース管理作業について説明します。この章で説明する項目は次のとおりです。

- 非 Oracle システムへのアクセスの設定
- 異機種間サービス・データ・ディクショナリの構造
- データ・ディクショナリ・ビュー
- DBMS_HS パッケージ ( 初期化パラメータの設定 )
- 分散外部プロシージャのセキュリティ
- エージェント自動登録

# 非 Oracle システムへのアクセスの設定

この項では非 Oracle システムへのアクセスを構成するための一般的なステップを説明します。特定のエージェントのインストレーションの詳細は、使用しているエージェントのインストレーションおよびユーザーズ・ガイドを参照してください。特定のエージェントの構成は、この項で示す構成と多少異なる場合があります。

ステップは、次のようになります。

1. 異機種間サービス・データ・ディクショナリをインストールします。
2. 異機種間サービス・エージェントにアクセスする環境をセットアップします。
3. 非 Oracle システムへのデータベース・リンクを作成します。
4. 接続をテストします。
5. オプションで、分散外部プロシージャを登録します。

## 異機種間サービス・データ・ディクショナリのインストール

異機種間サービス用のデータ・ディクショナリ表およびビューをインストールするには、異機種間サービス・データ・ディクショナリ表およびビュー、パッケージをすべて作成するスクリプトを実行する必要があります。ほとんどのシステムでは、このスクリプト名は CATHS.SQL で、$ORACLE_HOME/rdbms/admin に格納されています。

---

**注意：** データ・ディクショナリ表、ビューおよびパッケージは、Oracle8*i* Server にすでにインストールされている可能性があります。SYS.HS_FDS_CLASS など、異機種間サービス・データ・ディクショナリ・ビューの存在をチェックすることによって、これを確認できます。

---

## 異機種間サービス・エージェントにアクセスする環境の設定

Oracle8*i* Server は、非 Oracle システムとの接続を開始するために、Net8 リスナーを介してエージェント・プロセスを起動します。Oracle8*i* Server からエージェントへ接続するために必要な作業は、次のとおりです。

1. Oracle8*i* Server で使用するエージェントの Net8 サービス名を設定します。Net8 サービス名の記述子には、Net8 リスナーへのアクセスに必要な、プロトコルに固有の情報が含まれます。サービス名の記述子には、その接続が Oracle8*i* の異機種間サービスを使用することを示す（HS=OK）句を含める必要があります。
2. リスナーは、Oracle8*i* Server からの要求をリスニングし、異機種間サービス・エージェントを起動するようにセットアップする必要があります。リスナーによって異機種間サービス・エージェントが開始されるように *listener.ora* ファイルを変更して、リスナーを（再）起動する必要があります。

### Net8 サービス名の記述子の例

次に示すのは、*tnsnames.ora* のサービス名のエントリの例です。

```
MegaBase6_sales= (DESCRIPTION=
                    (ADDRESS=(PROTOCOL=tcp)
                             (HOST=dlsun206)
                             (PORT=1521))

                    (CONNECT_DATA = (SID=SalesDB))

                    (HS = OK))
```

このサービス名の記述は、*tnsnames.ora*、Oracle Names サーバーまたは Oracle のネーム・アダプタを使用するサードパーティのネーム・サーバーで定義します。Net8 サービス名の定義方法の詳細は、使用しているエージェントのインストレーションおよびユーザーズ・ガイドを参照してください。

### LISTENER.ORA のエントリの例

次に示すのは、*listener.ora* 内のリスナーのエントリの例です。

```
LISTENER =
   (ADDRESS_LIST =
      (ADDRESS=(PROTOCOL=tcp)
               (HOST = dlsun206)
               (PORT = 1521)
      )
  )
...
SID_LIST_LISTENER =
  (SID_LIST =
      (SID_DESC = (SID_NAME=SalesDB)
                  (ORACLE_HOME=/home/oracle/megabase/8.1.3)
                  (PROGRAM=tg4mb80)
      )
  )
```

エージェントの実行可能ファイルは、PROGRAM キーワードによって定義されます。エージェントの実行可能ファイルは、$ORACLE_HOME/bin ディレクトリ内に存在する必要があります。SID_NAME は通常、エージェントの初期化パラメータ・ファイルを定義するために使用されます。

## 非 Oracle システムへのデータベース・リンクの作成

非 Oracle システムとのデータベース・リンクを設定するには、CREATE DATABASE LINK コマンドを使用して、プライベートまたはパブリックのデータベース・リンクを作成します。

CREATE DATABASE LINK コマンドの USING 句で使用される「サービス名」は、Net8 のサービス名です。

たとえば、MegaBase リリース 6 サーバーの Sales データベースへのデータベース・リンクは、次のように作成できます。

```
CREATE DATABASE LINK salesdb
USING 'MegaBase6_sales';
```

**関連項目：** データベース・リンクの作成方法の詳細は、第 2 章「分散データベース管理」を参照してください。

## 接続のテスト

非 Oracle システムとの接続をテストするには、SQL 文または PL/SQL 文でデータベース・リンクを使用します。非 Oracle システムが SQL ベースのデータベースであれば、次のように、データベース・リンクを使用して既存の表またはビューからの検索を実行できます。

```
SELECT *
FROM product@salesdb
WHERE product_name like '%pencil%';
```

非 Oracle システムに初めてアクセスするときに、異機種間サービス・エージェントによって、情報が異機種間サービスのデータ・ディクショナリにアップロードされます。アップロードされる情報には、次の情報が含まれます。

- **非 Oracle システムの機能：** エージェントによって、非 Oracle システムの機能が指定されます。たとえば、結合または GROUP BY を実行できるかどうかを指定します。
- **SQL 変換情報：** エージェントによって、Oracle のファンクションおよびオペレータを、非 Oracle システムのファンクションおよびオペレータに変換する方法が指定されます。
- **データ・ディクショナリ変換：** エージェントによって、Oracle データ・ディクショナリ表を非 Oracle システムの表およびビューに変換する方法が指定されます。これにより、

非 Oracle システムのデータ・ディクショナリ情報が、Oracle データ・ディクショナリであるかのように使用可能になります。

---

**注意：** ほとんどのエージェントは、初回のアクセス時に、Oracle8*i* データ・ディクショナリに、情報が自動的にアップロードします。ただし、エージェント・ベンダーによっては、Oracle8*i* Server 上で実行する必要があるスクリプトを提供している場合もあります。

---

## 分散外部プロシージャの登録（オプション）

このステップは、分散外部プロシージャをサポートするエージェントにのみ必要なステップです。分散外部プロシージャを使用すると、ユーザーは、非 Oracle システムにプロシージャ・コールを行えます。エージェント・ベンダーが分散外部プロシージャを作成した場合、これらの分散外部プロシージャを Oracle8*i* Server に登録するためのスクリプトまたはインストーラが提供されます。

分散外部プロシージャを使用して非 Oracle システムにアクセスする場合は、リモート・プロシージャの実行に PL/SQL のリモート・プロシージャ・コールを使用します。

```
execute foo@non_oracle_system(1,2,3)
procedure successfully completed.
```

---

**注意：** 非 Oracle システムのストアド・プロシージャの実行には通常、分散外部プロシージャを使用する必要はありません。

---

---

**注意：** 分散外部プロシージャの登録方法の詳細は、使用しているエージェントのインストレーションおよびユーザーズ・ガイドを参照してください。非 Oracle システムで実行可能な分散外部プロシージャは、エージェント・ベンダーによって定義されます。実行可能なプロシージャのリストについては、使用しているエージェントのインストレーションおよびユーザーズ・ガイドを参照してください。

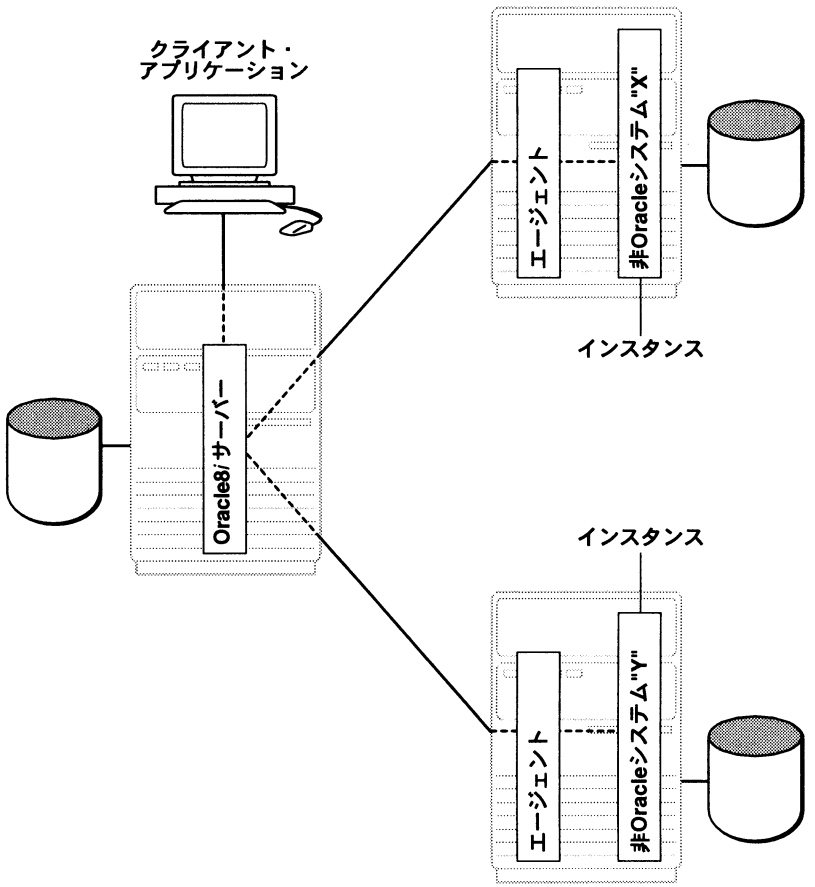
---

# 異機種間サービス・データ・ディクショナリの構造

Oracle8*i* Server からアクセスする非 Oracle システムは、それぞれが非 Oracle システムの「インスタンスおよびクラス」と見なされます。同じ Oracle8*i* Server から、複数の非 Oracle システムにアクセスできます。図 6-1 を参照してください。

Oracle8*i* Server は、アクセス対象となる非 Oracle システムごとに、その機能（SQL 変換、データ・ディクショナリ変換）を知る必要があります。この情報は、Oracle8*i* のデータ・ディクショナリに格納されています。

**図 6-1 インスタンス**




この情報がアクセス対象の非 Oracle システムごとに個別に格納されると、格納するデータ・ディクショナリ情報の量が多くなり、時には冗長になる可能性があります。たとえば、同じタイプの 3 つの非 Oracle システム・インスタンスにアクセスする必要がある場合、同じ機能、同じ SQL 変換および同じデータ・ディクショナリ変換が格納されることになります。

不必要な冗長性を回避するために、この情報はデータ・ディクショナリ内で「クラスおよびインスタンス」別に編成されています。「クラス」では非 Oracle システムのタイプが定義され、インスタンスでは特定の非 Oracle システムのクラスの特性が定義されます。インスタンス情報はクラス情報より優先され、クラス情報はサーバーから与えられるデフォルトより優先されるので注意してください。

複数の同じクラス（タイプ）の非 Oracle システムにアクセスする場合は、初期化パラメータなど、特定の情報をインスタンス・レベルで設定する必要があります。異機種間サービスでは、クラス情報とインスタンス情報の両方を格納します。複数インスタンスは同じクラス情報を共有できますが、非 Oracle システムの各インスタンスは、それぞれのインスタンス情報を持ちます。

たとえば、Oracle8*i* Server サーバーが、Megabase リリース 5 の 3 タイプのインスタンスと、Megabase リリース 6 の 2 タイプのインスタンスにアクセスする場合を考えます。Megabase のリリース 5 および 6 は、機能が異なるものとします。データ・ディクショナリには、2 つのクラス定義および 5 つのインスタンス定義が含まれます。この 2 つのクラス定義のうち、1 つはリリース 5 用で、もう 1 つはリリース 6 用です。

# データ・ディクショナリ・ビュー

異機種間サービスのデータ・ディクショナリ・ビューには、次の情報が含まれます。

- Oracle8*i* データ・ディクショナリにアップロードされるインスタンスとクラスの名前。アップロードされるクラスおよびインスタンスの情報は、それぞれ HS_FDS_CLASS ビューと HS_FDS_INST ビューで表示できます。
- SQL 変換など、クラスまたはインスタンスごとに定義された機能。機能情報は、HS_..._CAPS ビューで表示できます。
- クラスまたはインスタンスごとに定義されるデータ・ディクショナリ変換。データ・ディクショナリ変換情報は、HS_..._DD ビューで表示できます。
- クラスまたはインスタンスごとに定義される初期化パラメータ。初期化パラメータ情報は、HS_...INIT ビューで表示できます。
- Oracle8*i* Server からアクセスできる分散外部プロシージャ。

**表 6-1 異機種間サービスのデータ・ディクショナリ・ビュー**

| ビュー名 | 説明 |
|---|---|
| HS_FDS_CLASS | この Oracle8*i* Server からアクセスできるクラスを識別するビュー |
| HS_FDS_INST | この Oracle8*i* Server からアクセスできるインスタンスを識別するビュー |
| HS_CLASS_INIT | 各クラスの初期化パラメータを識別するビュー |
| HS_INST_INIT | 各インスタンスの初期化パラメータを識別するビュー |
| HS_BASE_DD | 異機種間サービスによってサポートされるすべてのデータ・ディクショナリの変換表名を識別するビュー |
| HS_CLASS_DD | 各クラスのデータ・ディクショナリ変換を識別するビュー |
| HS_INST_DD | 各インスタンスのデータ・ディクショナリ変換を識別するビュー |
| HS_BASE_CAPS | 異機種間サービスによってサポートされているすべての機能を識別するビュー |
| HS_CLASS_CAPS | 各クラスの機能を識別するビュー |
| HS_INST_CAPS | 各インスタンスの機能を識別するビュー |
| HS_EXTERNAL_OBJECTS | 分散外部プロシージャと、それに対応するライブラリに関する情報を提供するビュー |

ビューは、4 グループに分割できます。

- 一般ビュー
- トランザクション・サービスに使用されるビュー
- SQL サービスに使用されるビュー
- プロシージャ・サービスに使用されるビュー

ほとんどのデータ・ディクショナリ・ビューは、クラスとインスタンスの両方に対して定義されます。したがって、ほとんどの種類の情報には、"..._CLASS" および "..._INST" ビューが定義されています。

**関連項目：** クラスおよびインスタンスの詳細は、6-5 ページの「異機種間サービス・データ・ディクショナリの構造」を参照してください。

Oracle のデータ・ディクショナリ表と同じように、これらのビューは読取り専用です。基礎表の内容は、SQL を使用して変更しないでください。基礎表を変更するには、"DBMS_HS" パッケージで使用可能なプロシージャを使用します。詳細は、6-13 ページの「DBMS_HS パッケージ」を参照してください。

**関連項目：** これらのビューの詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

## 異機種間サービスの一般データ・ディクショナリ・ビュー

すべてのサービスに共通のビューには、次の情報が含まれます。

- Oracle8*i* データ・ディクショナリにアップロードされるインスタンスとクラスの名前。アップロードされるクラスおよびインスタンスの名前は、それぞれ HS_FDS_CLASS ビューおよび HS_FDS_INST ビューを介して表示できます。
- 異機種間サービスの初期化パラメータの情報。この情報は、HS_CLASS_INIT ビューと HS_INST_INIT ビューを介して表示できます。

たとえば、Oracle8*i* Server から、MegaBase リリース 5 およびリリース 6 の両方にアクセスできるとします。初めてエージェントにアクセスした後、Oracle8*i* Server にアップロードされた情報は、次のようになります。

```
select * from hs_fds_class;

FDS_CLASS_NAME FDS_CLASS_COMMENTS FDS_CLASS_ID
---------------------- ------------------------------ ------------
MegaBase5              Uses ODBC HS driver, R1.0                 1
MegaBase6              Uses ODBC HS driver, R1.0                21
```

2 つのクラスがアップロードされています。MegaBase リリース 5 のサーバーにアクセスするためのクラスと、MegaBase リリース 6 のサーバーにアクセスするためのクラスです。Oracle8*i* Server のデータ・ディクショナリには、MegaBase5 と MegaBase6 の両方の機能情報、SQL 変換およびデータ・ディクショナリ変換が含まれることになります。

この情報に加えて、Oracle8*i* Server データ・ディクショナリには、アクセスされる非 Oracle システム・インスタンスごとに、HS_FDS_INST ビューにインスタンス情報が含まれています。

## トランザクション・サービスのビュー

分散トランザクションに非 Oracle システムが関係する場合は、非 Oracle システム（およびエージェント）のトランザクション機能によって、そのシステムが分散トランザクションに参加できるかどうかが決まります。トランザクション機能は、HS_CLASS_CAPS および HS_INST_CAPS 機能表に格納されています。

非 Oracle システム（およびエージェント）が、2 フェーズ・コミット・プロトコルをサポートできるかどうかは、次の 5 タイプのいずれかに指定される "2PC type" の機能によって決まります。

| | |
|---|---|
| 読取り専用（RO） | 非 Oracle システムは、SQL の SELECT 文でのみ問合せできます。プロシージャ・コールは、データ書込みを行うと見なされるため、使用できません。 |
| シングル・サイト（SS） | 非 Oracle システムは、リモート・トランザクションを処理できますが、分散トランザクションは処理できません。つまり、2 フェーズ・コミット・プロトコルには参加できません。 |
| コミット確認（CC） | 非 Oracle システムは、分散トランザクションに参加できます。Oracle の 2 フェーズ・コミット・プロトコルに参加できますが、コミット・ポイント・サイトとしてのみです。つまり、データは準備できませんが、グローバル・コーディネータから要求された場合に必要な特定のトランザクションの結果を記録できます。 |
| 2 フェーズ・コミット | 非 Oracle システムは、分散トランザクションに参加できます。Oracle の 2 フェーズ・コミット・プロトコルに、通常の 2 フェーズ・コミット・ノードとして参加できます。ただし、コミット・ポイント・サイトとしては参加できません。つまり、データは準備できますが、グローバル・コーディネータから要求された場合に必要な特定のトランザクションの結果を記録できません。 |
| 2 フェーズ・コミット確認 | 非 Oracle システムは、分散トランザクションに参加できます。Oracle の 2 フェーズ・コミット・プロトコルに、通常の 2 フェーズ・コミット・ノードとしても、コミット・ポイント・サイトとしても参加できます。つまり、データを準備でき、グローバル・コーディネータから要求された場合に必要な特定のトランザクションの結果を記録できます。 |

ドライバおよび非 Oracle システムがサポートするトランザクション・モデルは、異機種間サービスのデータ・ディクショナリ・ビュー HS_CLASS_CAPS および HS_INST_CAPS から問合せできます。

機能の一つに "2PC type" が見つかります。

```
SELECT cap_description, translation
FROM hs_class_caps
WHERE cap_description LIKE '2PC%'
AND fds_class_name='MegaBase6';

CAP_DESCRIPTION TRANSLATION
--------------------------------------- -----------
2PC type (RO-SS-CC-PREP/2P-2PCC) CC
```

非 Oracle システムおよびエージェントによって分散トランザクションがサポートされている場合は、非 Oracle システムは他の Oracle8*i* Server のように扱われます。2 フェーズ・コミット・プロトコルで障害が発生した場合は、トランザクションは自動的に回復します。障害が残った場合は、データベース管理者がインダウト・トランザクションを手動で上書きする必要があります。分散トランザクションの詳細は、第 3 章「分散トランザクション」を参照してください。

### 分散外部プロシージャのトランザクション

分散外部プロシージャでは、非 Oracle システムのデータが変更されるかどうかわかりません。Oracle は異機種の分散データベースの一貫性を保つために、分散外部プロシージャによって非 Oracle システムが更新されると想定します。

このため、分散外部プロシージャは、それがアクセスされた唯一のノードであるのか、または他のノードもアクセスされたのかによって、リモート・トランザクションかまたは分散トランザクションに参加することになります。したがって、分散外部プロシージャを使用するためには、エージェントは少なくとも「単一サイト」のトランザクション・モデルをサポートしている必要があります。

## SQL サービスのビュー

SQL サービスに固有のデータ・ディクショナリ・ビューには、次の情報が含まれます。

- 非 Oracle データ・ソースの SQL 機能および SQL 変換
- Oracle のデータ・ディクショナリ・ビューを非 Oracle システムのデータ・ディクショナリに変換するための、データ・ディクショナリのマップ

### 機能と変換のビュー

HS_..._CAPS データ・ディクショナリ表には、非 Oracle システムのデータ・ソースの SQL 機能および SQL 変換についての情報が含まれています。

HS_..CAPS では、特定の SQL 言語機能をインプリメントするのが、非 Oracle データ・ストアであるのか、または Oracle サーバーであるのかを指定します。機能がオフのときは、Oracle8*i* は、その機能が含まれる非 Oracle データ・ソースに SQL 文を送信しませんが、後処理を実行することは可能です。

### データ・ディクショナリ変換のビュー

非 Oracle システムを Oracle8*i* Server として認識させるために、非 Oracle システムのデータ・ディクショナリは、Oracle のデータ・ディクショナリであるかのように問合せできます。これは、定義済みのデータ・ディクショナリ変換によって可能になります。これらの変換は、HS_..._DD ビューに格納されます。

たとえば、次の SELECT 文は、MegaBase のデータ・ディクショナリ表から EMP 表についての情報を取り出す MegaBase の問合せに変換されます。

```
SELECT *
FROM USER_TABLES@salesdb
WHERE UPPER(TABLE_NAME)='EMP';
```

データ・ディクショナリ表は、「変換」されず、「模造」されることがあります。非 Oracle データ・ソースのデータ・ディクショナリに必要な情報が格納されていないために、データ・ディクショナリを変換できない場合、異機種間サービスによってディクショナリ表が使用可能であるように見えますが、その表には情報が含まれていません。

Oracle8*i* のデータ・ディクショナリ・ビューや表が非 Oracle システム用に変換または模造されたかの情報を取り出すには、HS_CLASS_DD または HS_INST_DD ビューに次の問合せを発行します。

```
SELECT DD_TABLE_NAME, TRANSLATION_TYPE
FROM   HS_CLASS_DD
WHERE FDS_CLASS_NAME='MegaBase6';


DD_TABLE_NAME                  T
------------------------------ -
ALL_ARGUMENTS                  M
ALL_CATALOG                    T
ALL_CLUSTERS                   T
ALL_CLUSTER_HASH_EXPRESSIONS   M
ALL_COLL_TYPES                 M
ALL_COL_COMMENTS               T
ALL_COL_PRIVS                  M
ALL_COL_PRIVS_MADE             M
ALL_COL_PRIVS_RECD             M
...
```

変換タイプ 'T' は変換が存在することを指定します。変換タイプが 'M' の場合は、データ・ディクショナリ表が模造されていることを示します。

## 分散外部プロシージャのビュー

分散外部プロシージャおよびリモート・ライブラリは、Oracle8*i* Server で管理されます。エージェント・ベンダーは、分散外部プロシージャとそのライブラリを登録するスクリプトを提供します。これら登録されたプロシージャとライブラリに関する情報は、HS_EXTERNAL_OBJECTS データ・ディクショナリ・ビューに格納されます。次の情報が含まれます。

- 分散外部プロシージャまたはリモート・ライブラリの名前。
- 3GL ルーチンに関する情報を提供する PL/SQL のプロトタイプ。名前および引数、ライブラリ名が含まれます。

- 分散外部プロシージャが論理的に常駐する非 Oracle システムのインスタンス名。

# DBMS_HS パッケージ

DBMS_HS パッケージには、アプリケーションの開発者やデータベース管理者が異機種間サービスの初期化パラメータ、機能、インスタンス名、クラス名などを設定または設定解除するための、ファンクションやプロシージャが含まれています。

異機種間サービスに関するすべての DBMS_HS パッケージ・インタフェースのリストは、付録 B「DBMS_HS パッケージ・リファレンス」を参照してください。

## 初期化パラメータの設定

初期化パラメータは、Oracle8*i* Server または異機種間サービス・エージェントで設定できます。Oracle8*i* Server で初期化パラメータを設定するには、DBMS_HS パッケージを使用する必要があります。詳細は、使用しているエージェントのインストレーションおよびユーザーズ・ガイドを参照してください。エージェントと Oracle8*i* Server の両方に同じ初期化パラメータが設定された場合は、Oracle8*i* Server の初期化パラメータの値が優先されます。

初期化パラメータには、次の 2 種類があります。

- 汎用初期化パラメータ
- 非 Oracle データ・ストアのクラス固有の初期化パラメータ

汎用初期化パラメータは、異機種間サービスによって定義されます。汎用初期化パラメータの詳細は、付録 A「異機種間サービスの初期化パラメータ」を参照してください。

非 Oracle データ・ストアのクラス固有の初期化パラメータは、エージェント・ベンダーが定義しています。非 Oracle データ・ストアのクラス固有の初期化パラメータには、必須のものもあります。たとえば、非 Oracle システムに接続するときに必要な接続情報が初期化パラメータに含まれる場合があります。非 Oracle データ・ストアのクラス固有のパラメータについては、使用しているエージェントのインストレーション・ガイドおよびユーザーズ・ガイドを参照してください。

汎用的な初期化パラメータと非 Oracle データ・ストアのクラス固有の HS 初期化パラメータは、どちらも Oracle サーバーで DBMS_HS パッケージの CREATE_INST_INIT プロシージャを使用して設定できます。

たとえば、HS_DB_DOMAIN 初期化パラメータは次のように設定します。

```
DBMS_HS.CREATE_INST_INIT
          (FDS_INST_NAME   => 'SalesDB',
           FDS_CLASS_NAME  => 'MegaBase6',
           INIT_VALUE_NAME => 'HS_DB_DOMAIN',
           INIT_VALUE      => 'US.SALES.COM');
```

**関連項目：** 初期化パラメータの詳細は、付録 A「異機種間サービスの初期化パラメータ」を参照してください。

## 初期化パラメータの設定解除

Oracle8*i* Server で異機種間サービスの初期化パラメータの設定を解除するには、DROP_INST_INIT プロシージャを使用する必要があります。たとえば、HS_DB_DOMAIN エントリを削除するには、次のようにします。

```
DBMS_HS.DROP_INST_INIT
          (FDS_INST_NAME   => 'SalesDB',
           FDS_CLASS_NAME  => 'MegaBase6',
           INIT_VALUE_NAME => 'HS_DB_DOMAIN');
```

**注意：** DBMS_HS パッケージの詳細は、付録 B「DBMS_HS パッケージ・リファレンス」を参照してください。

# 分散外部プロシージャのセキュリティ

分散外部プロシージャを実行する権限の制御方法については、使用しているエージェント固有のマニュアルを参照してください。

# エージェント自動登録

エージェント自動登録により、リモート・ホスト上のエージェントを記述する異機種間サービスの構成データの更新プロセスが自動化され、異機種間データベース・リンクの正常動作が保証されます。エージェント自動登録はデフォルト動作なので注意してください。エージェント自動登録機能を使用しない場合は、Oracle 初期化パラメータ HS_AUTOREGISTER の値を FALSE に設定する必要があります。詳細は、6-17 ページの「Oracle サーバーの初期化パラメータ HS_AUTOREGISTER」を参照してください。

サーバーとエージェントは、HS 接続の操作を構成し、制御するために、次の 3 タイプの情報を使用します。

- **HS 初期化パラメータ :** これらのパラメータは、各種接続に固有の詳細な操作の制御を提供します。
- **機能定義 :** これらの定義は、非 Oracle データ・ソースでサポートされる SQL 言語機能などの詳細を識別します。
- **データ・ディクショナリ変換 :** これらの変換は、Oracle データ・ディクショナリの表とビューへの参照を、非 Oracle データ・ソース固有の同等な機能にマップします。

このマニュアルでは、この 3 組の情報をまとめて HS 構成データと呼びます。

## エージェント自動登録の利点

HS 構成データ（前項で説明した DBMS_HS_ADMIN パッケージを使用して設定）は、Oracle サーバーのデータ・ディクショナリに格納されます。エージェントは、リモートであるため別個に管理されている場合があるので、次のように、状況によってはサーバーとエージェントの間で構成が一致しなくなるおそれがあります。

- 新規エージェントがあるマシンにインストールされると、既存のサーバーには、そのエージェントの HS 構成データを表す HS データ・ディクショナリの内容が存在しません。
- 新規サーバーがインストールされると、そのサーバーには既存のエージェントと非 Oracle データ・ストアに関して必要な HS 構成データが存在しません。
- 非 Oracle インスタンスが旧バージョンから新バージョンにアップグレードされると、HS 構成データの変更が必要になります。
- リモート・サイトの HS エージェントが新バージョンにアップグレードされるか、パッチが適用されると、HS 構成データの変更が必要になります。
- 非 Oracle サイトの DBA が、チューニングやテストの目的でエージェントのセットアップを変更すると、その内容が HS 構成データに影響します。

エージェント自動登録機能を使用すると、このような状況でも異機種間サービスが正常に動作できます。

特に、Oracle サーバーと HS エージェントがどちらもバージョン 8.0.3 以上であれば、エージェント自動登録によって、両者間のインターオペラビリティが強化されます。そのための基本的メカニズムとして、エージェントからサーバーに HS 構成データ（HS データ・ディクショナリの内容）をアップロードする機能が用意されています。

自動登録によって、Oracle サーバーのデータ・ディクショナリに格納する HS 構成データの更新が自動化されます（サーバーの初期化パラメータ HS_AUTOREGISTER で有効化されている場合（後述））。このようなデータ・ディクショナリの更新は、エージェント自動登録によるアップロードを、過去に未登録エージェントの初回使用時に一度だけ実行することにより実現されます。インスタンス情報は、サーバーのデータ・ディクショナリに格納されるのではなく、接続ごとにアップロードされます。

## エージェント自動登録の機能

HS エージェント自動登録の機能は、次のとおりです。

- Oracle サーバーに対して、エージェントと非 Oracle データ・ストアを識別します。
- エージェントは、自分自身と接続先 Oracle8*i* サーバーの両方が使用する、異機種間サービスの初期化パラメータを定義できます。
- 接続の初期化中に、機能定義とデータ・ディクショナリ変換が HS エージェントからアップロードされます。

サーバーとエージェントがどちらもリリース 8.1 以上の場合、クラス情報のアップロードは、サーバーのデータ・ディクショナリ内でクラスが定義されていない場合にのみ発生するので注意してください。同様に、インスタンス情報は、サーバーのデータ・ディクショナリ内でインスタンスが定義されていない場合にのみアップロードされます。

このために必要な情報は、エージェントから提供される次の名前を使用して、サーバーのデータ・ディクショナリ内でアクセスされます。

- FDS_CLASS
- FDS_CLASS_VERSION

## FDS_CLASS および FDS_CLASS_VERSION

FDS_CLASS と FDS_CLASS_VERSION は、個々の HS エージェントおよびバージョンごとに、Oracle またはサードパーティ・ベンダーによって定義されます。これらの名前は、Oracle 異機種間サービスによって FDS_CLASS_NAME 形式の名前に変更され、サーバーのデータ・ディクショナリ内のクラス情報にアクセスするための主キーとして使用されます。

FDS_CLASS では、アクセスする非 Oracle データ・ストアのタイプを指定し、FDS_CLASS_VERSION では、非 Oracle データ・ストアとそれにアクセスするエージェントのバージョン番号を指定する必要があります。エージェントのコンポーネント（エージェントの実行可能ファイルやアップロード可能な定義）に変更があった場合は、新リリースを一意に識別するように FDS_CLASS_VERSION も変更する必要があるので注意してください。

**注意：** この情報は、各接続の初期化時にアップロードされます。

## FDS_INST_NAME

「インスタンス固有の情報」は、サーバーのデータ・ディクショナリに格納できます。インスタンス名 FDS_INST_NAME は、エージェントを管理する DBA によって構成されます。その方法は、使用中のエージェントに応じて異なります。Oracle サーバーは、FDS_INST_NAME を、FDS_INST_INIT、FDS_INST_CAPS および FDS_INST_DD のビュー内で同じ名前を持つ列の主キーとして使用し、データ・ディクショナリ内でインスタンス固有の構成情報を検索します。

FDS_INST_NAME を使用するサーバーのデータ・ディクショナリ・アクセスでは、構成情報の各行を一意に識別するために FDS_CLASS_NAME も使用されます。たとえば、データベースをクラス "MegaBase8.0.4" からクラス "MegaBase8.1.3" に移植すると、両方のデータベースはインスタンス名 "Scott" で同時に動作でき、別々の構成情報を使用できます。

クラス情報と違って、インスタンス情報はサーバーのデータ・ディクショナリに自動登録されません。

- サーバーのデータ・ディクショナリにインスタンス情報が含まれている場合、その情報はインスタンス構成を詳細に定義するために DBA が定義したセットアップ詳細を表します。インスタンス情報は、エージェントからサーバーにアップロードされません。
- サーバーのデータ・ディクショナリにインスタンス情報が含まれていない場合は、接続先のエージェントによって作成されたインスタンス情報が、サーバーにアップロードされます。アップロードされたデータは、サーバーのデータ・ディクショナリには格納されません。

## Oracle サーバーの初期化パラメータ HS_AUTOREGISTER

Oracle サーバーの初期化パラメータ HS_AUTOREGISTER を使用すると、HS エージェントの自動登録を使用可能または使用禁止にできます。このパラメータを TRUE に設定すると、以前に未登録であったエージェント・クラスや新規エージェント・バージョンが、サーバーのデータ・ディクショナリにアップロードされます。

このパラメータの説明と構文は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

このパラメータには、デフォルト値（TRUE）を使用することをお薦めします。この設定により、サーバーのデータ・ディクショナリの内容が、HS 接続に使用されたクラス機能とデータ・ディクショナリ変換の定義を、常に正しく表すことが保証されます。

# 7

# 異機種間サービスを使用したアプリケーション開発

この章では、異機種間サービスを使用するアプリケーションの開発者へ情報を提供します。

この章で説明する項目は次のとおりです。

- 異機種間サービスを使用したアプリケーション開発
- パススルー SQL
- バルク・フェッチ

# 異機種間サービスを使用したアプリケーション開発

アプリケーションを開発する場合、非 Oracle システムのアクセスを考慮する必要はありません。異機種間サービスによって、非 Oracle システムは別の Oracle8*i* Server であるかのように認識されます。

しかし、非 Oracle システム用の SQL 言語を使用して非 Oracle システムにアクセスする必要が出てくる場合があります。これを可能にするために、異機種間サービスにはパススルー SQL 機能があります。この機能を使用すると、アプリケーション・プログラマは、非 Oracle システムで固有の SQL 文を直接実行できます。

さらに、異機種間サービスは、非 Oracle システム、エージェントおよび Oracle サーバーの間のラージ・データ・セットのデータ転送を最適化するバルク・フェッチをサポートします。この章では、そのようなデータ転送を調整する方法についても説明します。

## パススルー SQL

パススルー SQL 機能を使用すると、アプリケーション開発者が送信する文が、Oracle8*i* Server に解釈されずに非 Oracle システムに直接送信されます。これは、Oracle で同義の文がない操作を非 Oracle システムで実現する場合に役立ちます。PL/SQL パッケージ DBMS_HS_PASSTHROUGH を使用して、非 Oracle システムで直接それらの文を実行できます。パススルー・パッケージで実行する文は、標準の「透過的」な SQL 文のように同じトランザクションで実行されます。

DBMS_HS_PASSTHROUGH パッケージは、概念的に非 Oracle システム上に存在します。パッケージ内のプロシージャおよびファンクションは、非 Oracle システムとの適切なデータベース・リンクを使用して起動する必要があります。

## パススルー SQL を使用する上での考慮

非 Oracle システムでトランザクションを（暗黙に）コミットまたはロールバックするパススルー SQL 文を実行する場合は、トランザクションの動作に注意する必要があります。たとえば、あるシステムでは、データ定義言語（DDL）の文が実行されると、トランザクションが暗黙にコミットします。Oracle サーバーはバイパスされるため、その Oracle サーバーは非 Oracle システムでのコミットを認識しません。これは、Oracle サーバーのトランザクションはコミットされなくても、非 Oracle システムのデータはコミットできることを意味します。

Oracle サーバーのトランザクションがロールバックされると、Oracle サーバーと非 Oracle サーバー間に不整合（つまり、グローバル・データの不整合）が発生します。

アプリケーションが通常の COMMIT を実行すれば、Oracle サーバーは、非 Oracle システムと分散トランザクションを調整できることに注意してください。パススルー機能を実行する文は、分散トランザクションの一部となります。

# パススルー SQL 文の実行

次の表に、DBMS_HS_PASSTHROUGH パッケージによって提供されるファンクションとプロシージャを示します。これらを使用すると、パススルー SQL 文を実行できます。次の項では、それらの使用方法について説明します。文は 2 つのクラスに分けられます。

- 非問合せ（INSERT、DELETE、UPDATE および DDL 文）
- 問合せ（SELECT 文）

| プロシージャ / ファンクション | 説明 |
|---|---|
| OPEN_CURSOR | カーソルをオープンする。 |
| CLOSE_CURSOR | カーソルをクローズする。 |
| PARSE | 文を解析する。 |
| BIND_VARIABLE | IN 変数をバインドする。 |
| BIND_OUT_VARIABLE | OUT 変数をバインドする。 |
| BIND_INOUT_VARIABLE | IN OUT 変数をバインドする。 |
| EXECUTE_NON_QUERY | 非問合せを実行する。 |
| EXECUTE_IMMEDIATE | バインド変数を使用せずに非問合せを実行する。 |
| FETCH_ROW | 問合せから行をフェッチする。 |
| GET_VALUE | SELECT 文から列の値を取り出す、あるいは OUT バインド・パラメータを取り出す。 |

## 非問合せの実行

非問合せ文を実行するには、EXECUTE_IMMEDIATE ファンクションを使用します。たとえば、データベース・リンク "SalesDB" を使用してアクセスできる非 Oracle システムで DDL 文を実行するには、次のように実行します。

```
DECLARE
  num_rows INTEGER;

BEGIN
  num_rows := DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE@SalesDB
              ('CREATE TABLE DEPT (n SMALLINT, loc CHARACTER(10))');
END;
```

変数 num_rows は、実行により影響を受ける行数が割り当てられます。DDL 文では 0（ゼロ）が戻されます。

EXECUTE_IMMEDIATE を使用した問合せを実行できません。また、バインド変数も使用できません。

## バインド変数

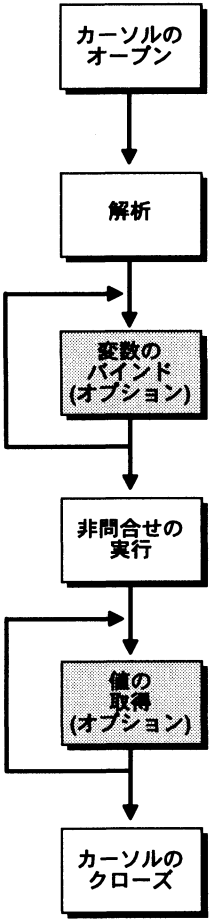
バインド変数を使用すると、同じ SQL 文を異なる値で複数回使用でき、SQL 文の解析に必要な回数を削減できます。たとえば、特定の表に 4 行挿入するときは、SQL 文を 1 度だけ解析し、バインドおよび SQL 文の実行を各行ごとに行います。各 SQL 文には、0（ゼロ）または複数のバインド変数を使用できます。

バインド変数を使用してパススルー SQL 文を実行するには、次のことを行う必要があります。

- カーソルをオープンします。
- SQL 文を非 Oracle システムで解析します。
- 変数をバインドします。
- SQL 文を非 Oracle システムで実行します。
- カーソルをクローズします。

図 7-1 は、バインド変数を使用して非問合せを実行するフロー・ダイアグラムです。

**図 7-1 非問合せパススルー SQL のフロー・ダイアグラム**




**IN バインド変数** バインド変数を文に指定する方法は、非 Oracle システムの構文によって決まります。たとえば、Oracle では、次のようにコロンを付けてバインド変数を定義します。

```
UPDATE EMP
SET SAL=SAL*1.1
WHERE ENAME=:ename
```

この文では、:ename がバインド変数です。非 Oracle システムでは、次のように疑問符を付けてバインド変数を指定する必要がある場合もあります。

```
UPDATE EMP
SET SAL=SAL*1.1
WHERE ENAME= ?
```

バインド変数のステップでは、この各バインド変数にホスト・プログラムの変数（この場合は、PL/SQL）を位置的に関連付ける必要があります。

たとえば、前述の文を実行するには次の PL/SQL プログラムを使用します。

```
DECLARE
  c INTEGER;
  nr INTEGER;
BEGIN
  c := DBMS_HS_PASSTHROUGH.OPEN_CURSOR@SalesDB;
  DBMS_HS_PASSTHROUGH.PARSE@SalesDB(c,
        'UPDATE EMP SET SAL=SAL*1.1 WHERE ENAME=?');
  DBMS_HS_PASSTHROUGH.BIND_VARIABLE(c,1,'JONES');
  nr:=DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY@SalesDB(c);
  DBMS_OUTPUT.PUT_LINE(nr||' rows updated');
  DBMS_HS_PASSTHROUGH.CLOSE_CURSOR@salesDB(c);
END;
```

**OUT バインド変数** 一部の非 Oracle システムでは、OUT バインド変数もサポートされています。OUT バインド変数は、バインド変数の値は、SQL 文の実行後までわかりません。

OUT バインド変数は、SQL 文の実行後に代入されますが、非 Oracle システムでは、SQL 文の実行前にそのバインド変数が OUT バインド変数であることがわかっている必要があります。BIND_OUT_VARIABLE プロシージャを使用して、そのバインド変数が OUT バインド変数であることを指定する必要があります。

SQL 文の実行後に、GET_VALUE プロシージャを使用して、OUT バインド変数の値を取り出すことができます。

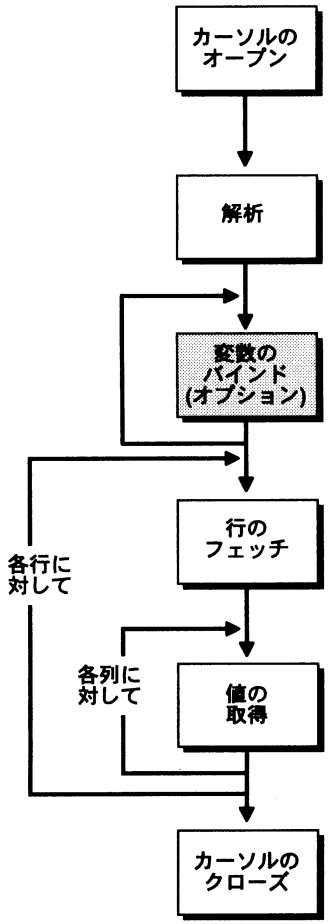
**IN OUT バインド変数** バインド変数は、IN 変数と OUT 変数の両方であってもかまいません。つまり、バインド変数の値は、SQL 文の実行前に指定する必要がありますが、SQL 文の実行後に変更される可能性もあります。

IN OUT バインド変数は、SQL 文の実行前に、BIND_INOUT_VARIABLE プロシージャを使用して、値を指定する必要があります。SQL 文の実行後は、GET_VALUE プロシージャを使用して、バインド変数の新しい値を取り出します。

## 問合せの実行

問合せと非問合せの違いは、問合せが結果セットを取り出すことです。結果セットは、カーソルを介した反復によって、取り出されます。SELECT 文が解析されると、FETCH_ROW プロシージャを使用して、結果セットの各行をフェッチできます。行がフェッチされた後は、GET_VALUE プロシージャを使用して、プログラム変数に選択リスト項目を取り出します。すべての行がフェッチされたら、カーソルをクローズします。図 7-2 を参照してください。

**図 7-2 問合せ用のパススルー SQL**




行をすべてフェッチする必要はありません。カーソルのオープン後は、たとえば、行を 2 ～ 3 行フェッチした後など、いつでもカーソルをクローズできます。

---

**注意：** 一度に 1 行ずつフェッチする場合も、異機種間サービスは、複数の行を非 Oracle データ・システムからバッファリングし、一度にフェッチすることにより、Oracle8*i* Server と非 Oracle システム間の往復回数を最適化します。

---

次の例で、問合せを実行します。

```
DECLARE
   val VARCHAR2(100);
   c INTEGER;
   nr INTEGER;
BEGIN
  c := DBMS_HS_PASSTHROUGH.OPEN_CURSOR@SalesDB;
  DBMS_HS_PASSTHROUGH.PARSE@SalesDB(c,
    'select ename
     from emp
     where deptno=10');
  LOOP
    nr := DBMS_HS_PASSTHROUGH.FETCH_ROW@SalesDB(c);
    EXIT WHEN nr = 0;
    DBMS_HS_PASSTHROUGH.GET_VALUE@SalesDB(c, 1, val);
    DBMS_OUTPUT.PUT_LINE(val);
  END LOOP;
  DBMS_HS_PASSTHROUGH.CLOSE_CURSOR@SalesDB(c);
END;
```

SELECT 文の解析後は、ファンクション FETCH_ROW によって "0" が戻されるまで、各行は LOOP 内でフェッチされ、表示されます。

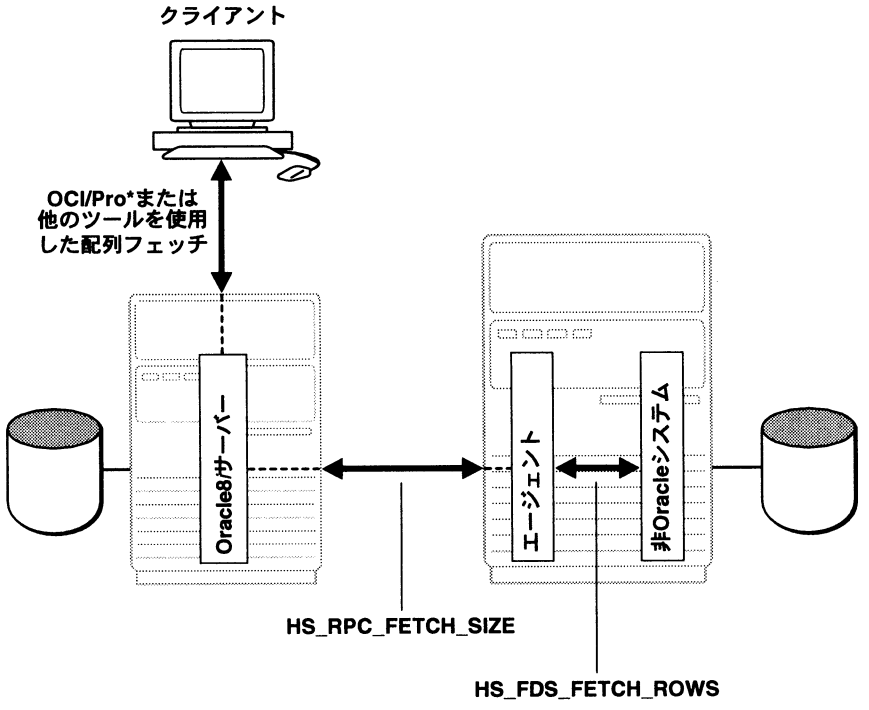
# バルク・フェッチ

アプリケーションが非 Oracle システムからデータをフェッチすると、異機種間サービスを使用して次のようにデータが転送されます。

- 非 Oracle システムからエージェント・プロセスへ
- エージェント・プロセスから Oracle サーバーへ
- Oracle サーバーからアプリケーションへ

Oracle では、3 種類のデータ転送をすべて最適化できます。図 7-3 を参照してください。

図 7-3 データ転送の最適化




## OCI、Oracle プリコンパイラまたは他のツールを使用した配列フェッチ

配列フェッチの使用により、アプリケーションと Oracle8*i* Server 間のデータ転送を最適化できます。配列フェッチおよびネットワーク 1 往復当たりのデータ送信量の指定方法については、アプリケーション開発ツールのマニュアルを参照してください。

## Oracle8*i* Server とエージェント間の配列フェッチ

非 Oracle システムからのデータの取り出しでは、異機種間サービスの初期化パラメータ HS_RPC_FETCH_SIZE によって、エージェントと Oracle8*i* Server 間でフェッチごとに送信されるバイト数を定義します。Oracle サーバーに返送されたバイト数が指定数に累積されるまで、または結果セットの最終行が非 Oracle システムからフェッチされるまで、エージェントは非 Oracle システムからデータをフェッチします。

## エージェントと外部データストア間の配列フェッチ

初期化パラメータ HS_FDS_FETCH_ROWS によって、非 Oracle システムから取り出される行数が決定されます。配列フェッチがエージェントでサポートされる必要があります。エージェントが配列フェッチをサポートしていることを確認するには、使用しているエージェントのマニュアルを参照してください。

## 再ブロック化

デフォルトでは、サーバーに返送するために十分なデータが取り出されるまで、エージェントは非 Oracle システムからデータをフェッチします。つまり、非 Oracle システムからフェッチしたバイト数が、HS_RPC_FETCH_SIZE の値以上になるまでフェッチされます。言い換えると、エージェントと Oracle サーバー間のデータは、HS_RPC_FETCH_SIZE で指定されたサイズに「再ブロック化」されます。

非 Oracle システムが配列フェッチをサポートする場合は、Oracle サーバーに配列フェッチすることにより、非 Oracle システムからフェッチしたデータが HS_RPC_FETCH_SIZE の値に達するまで待たずに、ただちに送信できます。つまり、非 Oracle システムから Oracle サーバーにデータを流し、再ブロック化を使用禁止にします。初期化パラメータ HS_RPC_FETCH_REBLOCKING の値を ' オフ ' に設定することで、これを実現できます。

たとえば、HS_RPC_FETCH_SIZE を 64KB、HS_FDS_FETCH_ROWS を 100 行に設定します。各行は約 600 バイトのサイズで、100 行で約 60KB と仮定します。HS_RPC_FETCH_REBLOCKING が ' オン ' に設定されていると、エージェントは非 Oracle システムから 100 行をフェッチし始めます。

データは 60KB しかないため、エージェントはデータを Oracle サーバーに返送しません。エージェントは、次の 100 行を非 Oracle システムからフェッチします。データは 120KB になるので、最初の 64KB が Oracle サーバーに返送されます。

ここで、エージェントには 56KB のデータが残っています。Oracle サーバーに次の 64KB のデータを送信する前に、エージェントは非 Oracle システムから別の 100 行をフェッチします。初期化パラメータ HS_RPC_FETCH_REBLOCKING を ' オフ ' に設定することにより、最初の 100 行が Oracle8*i* Server にただちに返送されます。

# A

# 異機種間サービスの初期化パラメータ

この付録では、異機種間サービスの初期化パラメータについて説明します。

初期化パラメータは、エージェントサイトでエージェントに固有のメカニズムを使用するか、または Oracle サーバーで DBMS_HS パッケージを使用して設定できます。DBMS_HS パッケージを使用して初期化パラメータを設定および削除する方法の詳細は、第 6 章「Oracle 異機種間サービスの管理」を参照してください。

- HS_COMMIT_POINT_STRENGTH
- HS_DB_DOMAIN
- HS_DB_INTERNAL_NAME
- HS_DB_NAME
- HS_DESCRIBE_CACHE_HWM
- HS_LANGUAGE
- HS_NLS_DATE_FORMAT
- HS_NLS_DATE_LANGUAGE
- HS_NLS_NCHAR
- HS_OPEN_CURSORS
- HS_ROWID_CACHE_SIZE
- HS_RPC_FETCH_REBLOCKING
- HS_FDS_FETCH_ROWS
- HS_RPC_FETCH_SIZE

# HS_COMMIT_POINT_STRENGTH

| サービス： | 一般 |
|---|---|
| デフォルト値： | 0 |
| 値の範囲： | 0 ～ 255 |

## 目的

このパラメータ HS_COMMIT_POINT_STRENGTH は、Oracle8*i* 用のパラメータ COMMIT_POINT_STRENGTH と同じ働きをします。

HS_COMMIT_POINT_STRENGTH は、分散トランザクションの中でコミット・ポイント・サイトになるサイトの重要度に関する値を設定します。最大のコミット・ポイント強度を持つ Oracle サーバーまたは非 Oracle システムがコミット・ポイント・サイトになります。非 Oracle システムがコミット・ポイント・サイトにならないようにする場合は、HS_COMMIT_POINT_STRENGTH をゼロに設定してください。

非 Oracle システムが、2 フェーズ・プロトコルに通常の 2 フェーズ・コミット・パートナおよびコミット・ポイント・サイトとして参加できる場合にのみ、HS_COMMIT_POINT_STRENGTH が重要になります。これは、トランザクション・モデルが 2 フェーズ・コミット・コンファーム (2PCC) である場合のみです。

異機種間分散トランザクションの詳細は、第 6 章「Oracle 異機種間サービスの管理」を参照してください。分散トランザクションおよびコミット・ポイント・サイトの詳細は、第 3 章「分散トランザクション」を参照してください。

# HS_DB_DOMAIN

| サービス： | 一般 |
|---|---|
| デフォルト値： | WORLD |
| 値の範囲： | 1 ～ 119 文字 |

## 目的

HS_DB_DOMAIN は、非 Oracle システムに対して、一意のネットワーク・サブアドレスを指定します。HS_DB_DOMAIN の使用方法は、Oracle サーバー用の同等のパラメータに似ています。『Oracle8*i* 管理者ガイド』および『Oracle8*i* リファレンス・マニュアル』を参照してください。Oracle Name Server を使用する場合は、HS_DB_DOMAIN が必要です。パラメータ HS_DB_NAME と HS_DB_DOMAIN は、非 Oracle システムのグローバル名を定義します。

**注意：** HS_DB_NAME と HS_DB_DOMAIN の組合せも一意にする必要があります。

# HS_DB_INTERNAL_NAME

| | |
|---|---|
| **サービス:** | **一般** |
| **デフォルト値:** | 01010101 |
| **値の範囲:** | 1 ～ 16 文字の 16 進文字 |

## 目的

HS_DB_INTERNAL_NAME は、異機種間サービス・エージェントによって接続されるインスタンスを識別する一意の 16 進数を指定します。このパラメータの値は、グローバル・ネーム・サービスがアクティブになったときに、トランザクション ID の一部として使用されます。一意でない数値を選択すると、トランザクションについて 2 フェーズ・コミット回復アクションが必要になったときに問題が発生します。

# HS_DB_NAME

| | |
|---|---|
| **サービス:** | **一般** |
| **デフォルト値:** | HO |
| **値の範囲:** | 1 ～ 8 文字 |

## 目的

非 Oracle システムに指定するデータ・ストアに対する一意の英数字名です。この名前は、コオペラティブ・サーバー環境内の非 Oracle システムを識別します。HS_DB_NAME と HS_DB_DOMAIN は、非 Oracle システムのグローバル名を定義します。

# HS_DESCRIBE_CACHE_HWM

| サービス： | 一般 |
|---|---|
| デフォルト値： | 100 |
| 値の範囲： | 1 ～ 4000 |

## 目的

HS_DESCRIBE_CACHE_HWM は、異機種間サービスで使用する記述キャッシュの最大登録数を指定します。この制限は、記述キャッシュの高水位標を表します。このキャッシュには、異機種間サービスが非 Oracle データストアを再アクセスするより再利用するようにマップされた表の記述が格納されています。高水位標を高くすると、特に数多くのマップされた表にアクセスするときに、パフォーマンスが改善されます。ただし、高水位標を高くすると、メモリー使用量を犠牲にしてパフォーマンスが改善することに留意してください。

# HS_LANGUAGE

| サービス: | 一般 |
|---|---|
| デフォルト値: | システム固有 |
| 値の範囲: | なし |

## 目的

初期化パラメータ HS_LANGUAGE は、非 Oracle データ・ソースのキャラクタ・セット、言語および地域に関する情報を異機種間サービスに提供します。初期化パラメータ HS_LANGUAGE の値は、次の形式で記述する必要があります。

```
<language>[_<territory>.<character_set>]
```

**注意：** 各国語サポートの初期化パラメータは、エラー・メッセージ、SQL サービスのデータおよび分散外部プロシージャのパラメータに影響します。

## キャラクタ・セット

Oracle8*i* Server と非 Oracle データ・ソースのキャラクタ・セットが同じであることが理想的です。キャラクタ・セットが同じでない場合、異機種間サービスは非 Oracle データ・ソースのキャラクタ・セットを Oracle8*i* キャラクタ・セットに変換しようとします（その逆の場合も同様）。これによりパフォーマンスは低下し、異機種間サービスによっては、あるキャラクタ・セットから別のキャラクタ・セットに文字を変換できないこともあります。

## 言語

初期化パラメータ HS_LANGUAGE の言語部分は、次を定義します。

- 日付の曜日および月の名前
- 日付および時間の AD、BC、PM および AM 記号
- デフォルトのソート・メカニズム

HS_LANGUAGE は、汎用的な異機種間サービス・メッセージ（ORA-25000 ～ ORA-28000）のエラー・メッセージの言語を定義しません。これらのメッセージは、Oracle サーバーのセッションの設定によって制御されます。

**注意：** 曜日と月の名前、日付と時間の AD、BC、PM および AM 記号は、初期化パラメータ HS_NLS_DATE_LANGUAGE を使用して言語とは別に設定できます。

## 地域

初期化パラメータ HS_LANGUAGE の地域句は、日と週のナンバリング、デフォルトの日付書式、小数点の文字と桁グループ・セパレータおよび ISO と各国通貨記号に関する規則を指定します。

- 初期化パラメータ HS_NLS_DATE_FORMAT を使用して、日付書式を上書きできます。
- Oracle8*i* Server と非 Oracle データ・ソース間の各国語サポートのレベルは、ドライバがどのようにインプリメントされているかに応じて異なります。各国語サポートのレベルの詳細は、使用しているプラットフォームのインストレーション・ガイドおよびユーザーズ・ガイドを参照してください。

# HS_NLS_DATE_FORMAT

| サービス: | 一般 |
|---|---|
| デフォルト値: | 値は HS_LANGUAGE パラメータにより決定されます。 |
| 値の範囲: | なし |

## 目的

HS_NLS_DATE_FORMAT は、ターゲット・システムが使用する日付の書式を定義します。このパラメータは、Oracle サーバー用の NLS_DATE_FORMAT パラメータと同じ機能を持ちます。date_format の値は、『Oracle8*i* リファレンス・マニュアル』にリストされている有効な日付マスクにすることができますが、ターゲット・システムの日付書式と一致している必要があります。たとえば、ターゲット・システムが「1995 年 2 月 14 日」という日付を「1995/02/14」という書式で格納する場合は、パラメータを次のように設定します。

```
'YYYY/MM/DD'
```

# HS_NLS_DATE_LANGUAGE

| | |
|---|---|
| **サービス:** | **一般** |
| **デフォルト値:** | 値は HS_LANGUAGE パラメータにより決定されます。 |
| **値の範囲:** | なし |

## 目的

HS_NLS_DATE_LANGUAGE パラメータは、非 Oracle システムから受け取る文字日付値で使用する言語を定義します。日付書式は、言語に依存しないように定義できます。たとえば、'DD/MM/YY' という形式の場合、文字日付の 3 つのコンポーネントはすべて数字です。ただし、'DD-MON-YY' という形式の場合は、月を示すコンポーネントは 3 文字に省略された名前です。この略称は、言語に大きく依存します。たとえば、April の略称は APR で、フランス語の場合は、AVR（Avril）です。

異機種間サービスでは、非 Oracle システムからフェッチする文字日付の値は、この形式であると仮定します。また、異機種間サービスから非 Oracle システムへ送信するときも、この形式の文字日付のバインド値が使用されます。

# HS_NLS_NCHAR

| | |
|---|---|
| **サービス:** | **一般** |
| **デフォルト値:** | 値は HS_LANGUAGE パラメータにより決定されます。 |
| **値の範囲:** | なし |

HS_NLS_NCHAR パラメータは、非 Oracle データ・ソースの各国キャラクタ・セットの値を異機種間サービスに伝えるために使用します。値は、Oracle の NLSRTL ライブラリがサポートするキャラクタ・セットのキャラクタ・セット ID である必要があります。

HS_LANGUAGE パラメータも参照してください。

# HS_OPEN_CURSORS

| サービス： | 一般 |
|---|---|
| デフォルト値： | 50 |
| 値の範囲： | なし |

## 目的

HS_OPEN_CURSORS パラメータは、非 Oracle システムのインスタンスへの 1 つの接続にオープンできる最大カーソル数を定義します。

この値が、Oracle server でのオープン・カーソルの数を超えないようにしてください。したがって、Oracle サーバーでは、HS_OPEN_CURSORS 初期化パラメータと同じ値に設定することをお薦めします。

# HS_ROWID_CACHE_SIZE

| | |
|---|---|
| **サービス:** | **一般** |
| **デフォルト値:** | 3 |
| **値の範囲:** | 1 ～ 32767 |

## 目的

HS_ROWID_CACHE_SIZE は、非 Oracle システムの ROWID を格納する異機種間サービスのキャッシュ・サイズを指定します。キャッシュには、SQL 文または SELECT FOR UPDATE 文にある WHERE CURRENT OF 句をサポートするために必要な非 Oracle システムの ROWID を格納しています。

キャッシュが満杯になると、そのキャッシュの第 1 スロットが再利用され、次は第 2 スロットと続きます。非 Oracle システムの ROWID は、HS_ROWID_CACHE_SIZE までキャッシュされます。

# HS_RPC_FETCH_REBLOCKING

| サービス： | 一般 |
|---|---|
| デフォルト値： | ON |
| 値の範囲： | OFF または ON |

## 目的

このパラメータによって、異機種間サービスが、ORACLE サーバーと非 Oracle データストアに接続された異機種間サービス・エージェントとの間で、データ転送のパフォーマンスを最適化するかどうかが制御されます。詳細は、第 7 章「異機種間サービスを使用したアプリケーション開発」を参照してください。

"OFF" の値は、フェッチ・データの再ブロック化を使用禁止にします。これは、データがエージェントからサーバーにただちに送信されることを意味します。"ON" の値は、再ブロック化を可能にします。これは、非 Oracle システムからフェッチされたデータがエージェントにバッファリングされた後、フェッチしたデータ量が HS_RPC_FETCH_SIZE 以上になるまで、Oracle サーバーには送信されないことを意味します。

# HS_FDS_FETCH_ROWS

| | |
|---|---|
| **サービス:** | **一般** |
| **デフォルト値:** | 20 |
| **値の範囲:** | 10 進数の整数（行数） |

## 目的

このパラメータは、エージェントが非 Oracle データストアから一度にフェッチする行数を指定します。

各異機種間サービス・エージェントには、通常、この変数の範囲に関する独自の上限があります。外部データストアが配列フェッチをサポートしない場合は、このパラメータの値を 1 にする必要があります。詳細は、該当するエージェントのマニュアルを参照してください。

詳細は、第 7 章「異機種間サービスを使用したアプリケーション開発」を参照してください。

# HS_RPC_FETCH_SIZE

| | |
|---|---|
| **サービス:** | **一般** |
| **デフォルト値:** | 4000 |
| **値の範囲:** | 10 進数の整数（バイト数） |

## 目的

この初期化パラメータは、サーバーとエージェント・プロセス間のデータ転送レートを最適化するために、内部データのバッファリングを調整します。値を大きくすると、1 往復当りのデータ転送をさらに最適化できます。ただし、非 Oracle システムからフェッチしたデータが HS_RPC_FETCH_SIZE と等しくなるまで、サーバーにデータは返送されないため、特定の問合せの応答時間は長くなります。

詳細は、第 7 章「異機種間サービスを使用したアプリケーション開発」を参照してください。

# B

# DBMS_HS パッケージ・リファレンス

この付録では、異機種間サービスを管理するための DBMS_HS パッケージに関するすべてのインタフェース情報を提供します。異機種間サービスの管理の詳細は、第 6 章「Oracle 異機種間サービスの管理」を参照してください。

この付録では次の事項を参照できます。

- DBMS_HS.CREATE_FDS_INST
- DBMS_HS.CREATE_INST_INIT
- DBMS_HS.DROP_FDS_INST
- DBMS_HS.DROP_INST_INIT

# DBMS_HS.CREATE_FDS_INST

## 目的

非 Oracle システムのインスタンスを Oracle8*i* Server に登録します。これは、非 Oracle システム・インスタンスの論理名で、異機種間サービスと呼ばれます。登録済みのインスタンスに関する情報は、ビュー HS_FDS_INST を介して参照可能です。

## インタフェースの説明

```
PROCEDURE create_fds_inst(
FDS_INST_NAME      IN VARCHAR2,
FDS_CLASS_NAME     IN VARCHAR2,
FDS_INST_COMMENTS IN VARCHAR2 )
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| FDS_INST_NAME | Oracle8*i* Server に登録される非 Oracle システム・インスタンス。 |
| FDS_CLASS_NAME | 非 Oracle システムのインスタンスに対応付けられたクラス。 |
| FDS_INST_COMMENTS | 非 Oracle システムのインスタンスを説明する最大 255 バイトのコメント。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-24270 | そのインスタンスはすでに存在します。 |
| ORA-24274 | オブジェクトを作成できませんでした。存在する CLASS または INSTANCE 名を渡しましたか？ |

## 関連項目

DBMS_HS.DROP_FDS_INST

# DBMS_HS.CREATE_INST_INIT

## 目的

非 Oracle システムのインスタンスに対して初期化変数を作成します。初期化変数の作成方法の詳細は、第 6 章「Oracle 異機種間サービスの管理」を参照してください。設定可能な初期化パラメータについては、付録 A「異機種間サービスの初期化パラメータ」を参照してください。この情報は、ビュー HS_INST_INIT を介して参照可能です。

## インタフェースの説明

```
PROCEDURE create_inst_init(
FDS_INST_NAME IN VARCHAR2
FDS_CLASS_NAME IN VARCHAR2
INIT_VALUE_NAME IN VARCHAR2,
INIT_VALUE IN VARCHAR2,
INIT_VALUE_TYPE IN VARCHAR2);
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| FDS_INST_NAME | 初期化パラメータの適用が必要な非 Oracle システムのインスタンス。 |
| FDS_CLASS_NAME | 非 Oracle システムのインスタンスに対応付けられたクラス。 |
| INIT_VALUE_NAME | 初期化パラメータの名前。有効な値については、付録 A「異機種間サービスの初期化パラメータ」を参照してください。 |
| INIT_VALUE | 初期化パラメータの値。 |
| INIT_VALUE_TYPE | 次のいずれかです。<br>■ 初期化パラメータをエージェント・プロセスの環境変数として設定する必要がある場合は 'T'。<br>■ 正規の初期化パラメータ（デフォルト）の場合は 'F'。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-24270 | このインスタンスについてその初期化パラメータはすでに定義されています。 |
| ORA-24272 | INIT_VALUE_TYPE は 'T' または 'F' です。 |
| ORA-24274 | 初期化パラメータを作成できませんでした。存在する CLASS または INSTANCE 名を渡しましたか？ |

## 関連項目

DBMS_HS.DROP_INST_INIT

# DBMS_HS.DROP_FDS_INST

## 目的

非 Oracle システムのインスタンスの登録を抹消します。ビュー HS_FDS_INST には、登録済みのインスタンスに関する情報が含まれます。

## インタフェースの説明

```
PROCEDURE drop_fds_inst(
FDS_INST_NAME  IN VARCHAR2,
FDS_CLASS_NAME IN VARCHAR2)
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| FDS_INST_NAME | Oracle8*i* Server から登録解除される非 Oracle システムのインスタンス。 |
| FDS_CLASS_NAME | 非 Oracle システムのインスタンスに対応付けられたクラス。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-24274 | インスタンスを削除できませんでした。存在する CLASS または INSTANCE 名を渡しましたか？ |

## 関連項目

DBMS_HS.CREATE_FDS_INST

# DBMS_HS.DROP_INST_INIT

## 目的

特定の非 Oracle システムのインスタンスの初期化変数を削除します。特定のインスタンスについて定義された初期化パラメータは、HS_INST_INIT および HS_ALL_INIT ビューを使用して問合せできます。

## インタフェースの説明

```
PROCEDURE drop_inst_init(
FDS_INST_NAME   IN VARCHAR2,
FDS_CLASS_NAME  IN VARCHAR2,
INIT_VALUE_NAME IN VARCHAR2)
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| FDS_INST_NAME | 初期化パラメータの削除が必要な非 Oracle システムのインスタンス。 |
| FDS_CLASS_NAME | 非 Oracle システムのインスタンスに対応付けられたクラス。 |
| INIT_VALUE_NAME | 初期化パラメータの名前。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-24274 | 初期化パラメータを削除できませんでした。存在する CLASS または INSTANCE 名を渡しましたか？ |

## 関連項目

DBMS_HS.CREATE_INST_INIT

# C

# パススルー SQL の DBMS_HS_PASSTHROUGH

この付録では、異機種間サービスの機能であるパス・スルー SQL のための、パッケージ DBMS_HS_PASSTHROUGH 内のプロシージャとファンクションについて説明します。このパッケージの使用方法の詳細は、第 7 章「異機種間サービスを使用したアプリケーション開発」を参照してください。

この付録では次の事項を参照できます。

- DBMS_HS_PASSTHROUGH.BIND_VARIABLE
- DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW
- DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE
- DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE_RAW
- DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE
- DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE_RAW
- DBMS_HS_PASSTHROUGH.CLOSE_CURSOR
- DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE
- DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY
- DBMS_HS_PASSTHROUGH.FETCH_ROW
- DBMS_HS_PASSTHROUGH.GET_VALUE
- DBMS_HS_PASSTHROUGH.GET_VALUE_RAW
- DBMS_HS_PASSTHROUGH.OPEN_CURSOR
- DBMS_HS_PASSTHROUGH.PARSE

# DBMS_HS_PASSTHROUGH.BIND_VARIABLE

## 目的

"IN" 変数に PL/SQL プログラム変数を所定の位置にバインドします。変数のバインド方法は、第 7 章「異機種間サービスを使用したアプリケーション開発」を参照してください。

## インタフェースの説明

```
PROCEDURE BIND_VARIABLE (c IN BINARY_INTEGER NOT NULL,
                         pos IN BINARY_INTEGER NOT NULL,
                         val    IN <dty>
                         [,name IN VARCHAR2])
```

<dty> は次のいずれかになります。

- DATE
- NUMBER
- VARCHAR2

RAW 型の変数をバインドするには、プロシージャ DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW を使用します。

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| c | パス・スルー SQL 文に関連付けられたカーソル。カーソルは、OPEN_CURSOR ルーチンを使用してオープンし、PARSE ルーチンを使用して解析する必要があります。 |
| pos | SQL 文でのバインド変数の位置。1 から始まります。 |
| val | バインド変数に渡す必要のある値。 |
| name | バインド変数に名前を付けるためのオプションのパラメータ。たとえば、"SELECT * FROM emp WHERE ename=:ename" では、バインド変数 ":ename" の位置は 1 で、名前は ":ename" です。このパラメータは、非 Oracle システムが位置的なバインドではなく「名前付きのバインド」をサポートしている場合に使用できます。ただし、それでも位置を渡す必要があることに注意してください。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-28550 | 渡されたカーソルが無効です。 |
| ORA-28552 | プロシージャを正しい順序で実行できません。最初にカーソルをオープンしてから、SQL 文を解析しましたか？ |
| ORA-28553 | バインド変数の位置が範囲外です。 |
| ORA-28555 | NOT NULL パラメータに NULL 値が渡されました。 |

## 純粋さのレベル

定義される純粋さのレベル : WNDS、RNDS

## 関連項目

DBMS_HS_PASSTHROUGH.OPEN_CURSOR DBMS_HS_PASSTHROUGH.PARSE DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW

# DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW

## 目的

RAW 型の IN 変数をバインドします。

## インタフェースの説明

```
PROCEDURE BIND_VARIABLE_RAW
(c IN BINARY_INTEGER NOT NULL,
pos IN BINARY_INTEGER NOT NULL,
val     IN RAW
 [,name IN VARCHAR2])
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| c | パス・スルー SQL 文に関連付けられたカーソル。カーソルは、OPEN_CURSOR ルーチンを使用してオープンし、PARSE ルーチンを使用して解析する必要があります。 |
| pos | SQL 文でのバインド変数の位置。1 から始まります。 |
| val | バインド変数に渡す必要のある値。 |
| name | バインド変数に名前を付けるためのオプションのパラメータ 。たとえば、"SELECT * FROM emp WHERE ename=:ename" では、バインド変数 ":ename" の位置は 1 で、名前は ":ename" です。このパラメータは、非 Oracle システムが位置的なバインドではなく「名前付きのバインド」をサポートしている場合に使用できます。ただし、それでも位置を渡す必要があることに注意してください。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-28550 | 渡されたカーソルが無効です。 |
| ORA-28552 | プロシージャを正しい順序で実行できません。最初にカーソルをオープンしてから、SQL 文を解析しましたか？ |
| ORA-28553 | バインド変数の位置が範囲外です。 |
| ORA-28555 | NOT NULL パラメータに NULL 値が渡されました。 |

## 純粋さのレベル

定義される純粋さのレベル : WNDS、RNDS

## 関連項目

DBMS_HS_PASSTHROUGH.OPEN_CURSOR DBMS_HS_PASSTHROUGH.PARSE DBMS_HS_PASSTHROUGH.BIND_VARIABLE DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE

# DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE

## 目的

OUT 変数に PL/SQL プログラム変数をバインドします。OUT パラメータのバインディングの詳細は、第 7 章「異機種間サービスを使用したアプリケーション開発」を参照してください。

## インタフェースの説明

```
PROCEDURE BIND_OUT_VARIABLE
c IN BINARY_INTEGER NOT NULL,
pos IN BINARY_INTEGER NOT NULL,
val      OUT <dty>,
[,name IN VARCHAR2])
```

<dty> は次のいずれかになります。

- DATE
- NUMBER
- VARCHAR2

RAW データ型の OUT 変数のバインディングについては、「BIND_OUT_VARIABLE_RAW」を参照してください。

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| c | パス・スルー SQL 文に関連付けられたカーソル。カーソルは、OPEN_CURSOR ルーチンを使用してオープンし、PARSE ルーチンを使用して解析する必要があります。 |
| pos | SQL 文でのバインド変数の位置。1 から始まります。 |
| val | OUT バインド変数がその値を格納する変数。パッケージは、変数の「サイズ」のみを記憶します。SQL 文の実行後に、GET_VALUE を使用して OUT パラメータの値を取り出すことができます。取り出した値のサイズが、BIND_OUT_VARIABLE を使用して渡されたパラメータのサイズを超えないようにしてください。 |
| name | バインド変数に名前を付けるためのオプションのパラメータ。たとえば、"SELECT * FROM emp WHERE ename=:ename" では、バインド変数 ":ename" の位置は 1 で、名前は ":ename" です。このパラメータは、非 Oracle システムが位置的なバインドではなく「名前付きのバインド」をサポートしている場合に使用できます。ただし、それでも位置を渡す必要があることに注意してください。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-28550 | 渡されたカーソルが無効です。 |
| ORA-28552 | プロシージャを正しい順序で実行できません。最初にカーソルをオープンしてから、SQL 文を解析しましたか？ |
| ORA-28553 | バインド変数の位置が範囲外です。 |
| ORA-28555 | NOT NULL パラメータに NULL 値が渡されました。 |

## 純粋さのレベル

定義される純粋さのレベル：WNDS、RNDS

## 関連項目

DBMS_HS_PASSTHROUGH.OPEN_CURSOR DBMS_HS_PASSTHROUGH.PARSE DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE_RAW DBMS_HS_PASSTHROUGH.BIND_VARIABLE DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW DBMS_HS_PASSTHROUGH.GET_VALUE

# DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE_RAW

## 目的

RAW データ型の OUT 変数に PL/SQL プログラム変数をバインドします。OUT パラメータのバインディングの詳細は、第 7 章「異機種間サービスを使用したアプリケーション開発」を参照してください。

## インタフェースの説明

```
PROCEDURE BIND_OUT_VARIABLE
c IN BINARY_INTEGER NOT NULL,
pos IN BINARY_INTEGER NOT NULL,
val OUT RAW,
,name IN VARCHAR2])
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| c | パス・スルー SQL 文に関連付けられたカーソル。カーソルは、OPEN_CURSOR ルーチンを使用してオープンし、PARSE ルーチンを使用して解析する必要があります。 |
| pos | SQL 文でのバインド変数の位置。1 から始まります。 |
| val | OUT バインド変数がその値を格納する変数。パッケージは、変数の「サイズ」のみを記憶します。SQL 文の実行後に、GET_VALUE を使用して OUT パラメータの値を取り出すことができます。取り出した値のサイズが、BIND_OUT_VARIABLE_RAW を使用して渡されたパラメータのサイズを超えないようにしてください。 |
| name | バインド変数に名前を付けるためのオプションのパラメータ。たとえば、"SELECT * FROM emp WHERE ename=:ename" では、バインド変数 ":ename" の位置は 1 で、名前は ":ename" です。このパラメータは、非 Oracle システムが位置的なバインドではなく「名前付きのバインド」をサポートしている場合に使用できます。ただし、それでも位置を渡す必要があることに注意してください。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-28550 | 渡されたカーソルが無効です。 |
| ORA-28552 | プロシージャを正しい順序で実行できません。最初にカーソルをオープンしてから、SQL 文を解析しましたか？ |
| ORA-28553 | バインド変数の位置が範囲外です。 |
| ORA-28555 | NOT NULL パラメータに NULL 値が渡されました。 |

## 純粋さのレベル

定義される純粋さのレベル : WNDS、RNDS

## 関連項目

DBMS_HS_PASSTHROUGH.OPEN_CURSOR DBMS_HS_PASSTHROUGH.PARSE DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE DBMS_HS_PASSTHROUGH.BIND_VARIABLE DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW DBMS_HS_PASSTHROUGH.GET_VALUE

# DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE

## 目的

IN OUT バインド変数をバインドします。IN OUT パラメータのバインディングの詳細は、第 7 章「異機種間サービスを使用したアプリケーション開発」を参照してください。

## インタフェースの説明

```
PROCEDURE BIND_INOUT_VARIABLE
c IN BINARY_INTEGER NOT NULL,
pos IN BINARY_INTEGER NOT NULL,
val       IN OUT <dty>,
,name IN VARCHAR2]
```

<dty> は次のいずれかになります。

- DATE
- NUMBER
- VARCHAR2

RAW データ型の IN OUT 変数のバインディングについては、「BIND_INOUT_VARIABLE_RAW」を参照してください。

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| c | パス・スルー SQL 文に関連付けられたカーソル。カーソルは、OPEN_CURSOR ルーチンを使用してオープンし、PARSE ルーチンを使用して解析する必要があります。 |
| pos | SQL 文でのバインド変数の位置。1 から始まります。 |
| val | この値は、2 つの目的で使用されます。<br>■ SQL 文が実行される前に、IN 値を提供する<br>■ out 値のサイズを判別する |
| name | バインド変数に名前を付けるためのオプションのパラメータ。たとえば、"SELECT * FROM emp WHERE ename=:ename" では、バインド変数 ":ename" の位置は 1 で、名前は ":ename" です。このパラメータは、非 Oracle システムが位置的なバインドではなく「名前付きのバインド」をサポートしている場合に使用できます。ただし、それでも位置を渡す必要があることに注意してください。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-28550 | 渡されたカーソルが無効です。 |
| ORA-28552 | プロシージャを正しい順序で実行できません。最初にカーソルをオープンしてから、SQL 文を解析しましたか？ |
| ORA-28553 | バインド変数の位置が範囲外です。 |
| ORA-28555 | NOT NULL パラメータに NULL 値が渡されました。 |

## 純粋さのレベル

定義される純粋さのレベル : WNDS、RNDS

## 関連項目

DBMS_HS_PASSTHROUGH.OPEN_CURSOR DBMS_HS_PASSTHROUGH.PARSE DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE_RAW DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE_RAW DBMS_HS_PASSTHROUGH.BIND_VARIABLE DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW DBMS_HS_PASSTHROUGH.GET_VALUE

# DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE_RAW

## 目的

RAW データ型の IN OUT バインド変数をバインドします。IN OUT パラメータのバインディングの詳細は、第 7 章「異機種間サービスを使用したアプリケーション開発」を参照してください。

## インタフェースの説明

```
PROCEDURE BIND_INOUT_VARIABLE
c IN BINARY_INTEGER NOT NULL,
pos IN BINARY_INTEGER NOT NULL,
val IN OUT RAW,
[,name IN VARCHAR2]);
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| c | パス・スルー SQL 文に関連付けられたカーソル。カーソルは、OPEN_CURSOR ルーチンを使用してオープンし、PARSE ルーチンを使用して解析する必要があります。 |
| pos | SQL 文でのバインド変数の位置。1 から始まります。 |
| val | この値は、2 つの目的で使用されます。<br>■ SQL 文が実行される前に、IN 値を提供する<br>■ out 値のサイズを判別する |
| name | バインド変数に名前を付けるためのオプションのパラメータ。たとえば、"SELECT * FROM emp WHERE ename=:ename" では、バインド変数 ":ename" の位置は 1 で、名前は ":ename" です。このパラメータは、非 Oracle システムが位置的なバインドではなく「名前付きのバインド」をサポートしている場合に使用できます。ただし、それでも位置を渡す必要があることに注意してください。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-28550 | 渡されたカーソルが無効です。 |
| ORA-28552 | プロシージャを正しい順序で実行できません。最初にカーソルをオープンしてから、SQL 文を解析しましたか？ |
| ORA-28553 | バインド変数の位置が範囲外です。 |
| ORA-28555 | NOT NULL パラメータに NULL 値が渡されました。 |

## 純粋さのレベル

定義される純粋さのレベル : WNDS、RNDS

## 関連項目

DBMS_HS_PASSTHROUGH.OPEN_CURSOR DBMS_HS_PASSTHROUGH.PARSE DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE_RAW, DBMS_HS_PASSTHROUGH.BIND_VARIABLE DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW DBMS_HS_PASSTHROUGH.GET_VALUE

# DBMS_HS_PASSTHROUGH.CLOSE_CURSOR

## 目的

このファンクションは、SQL 文が非 Oracle システムで実行された後に、カーソルをクローズし、関連付けられているメモリーを解放します。カーソルがオープンされていない場合は、「操作なし」です。

## インタフェースの説明

```
PROCEDURE CLOSE_CURSOR (c IN BINARY_INTEGER NOT NULL);
```

### パラメータの説明

| パラメータ | 説明 |
|---|---|
| c | リリースされるカーソル。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-28555 | NOT NULL パラメータに NULL 値が渡されました。 |

### 純粋さのレベル

定義される純粋さのレベル : WNDS、RNDS

## 関連項目

DBMS_HS_PASSTHROUGH.OPEN_CURSOR

# DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE

## 目的

このファンクションは、SQL 文をただちに実行します。有効な SQL コマンド（SELECT を除く）を、ただちに実行できます。文にバインド変数を含めることはできません。文は引数で VARCHAR2 として渡されます。SQL 文は、内部的には OPEN_CURSOR、PARSE、EXECUTE_NON_QUERY、CLOSE_CURSOR の PASSTHROUGH SQL プロトコル・シーケンスを使用して実行されます。

## インタフェースの説明

```
FUNCTION EXECUTE_IMMEDIATE ( S IN VARCHAR2 NOT NULL )
RETURN BINARY_INTEGER;
```

### パラメータの説明

| パラメータ | 説明 |
|---|---|
| s | ただちに実行される文の VARCHAR2 変数。 |

### 戻り値

SQL 文の実行により影響のある行数。

### 例外：

| 例外 | 説明 |
|---|---|
| ORA-28544 | オープン・カーソルの最大に達しました。 |
| ORA-28551 | SQL 文が無効です。 |
| ORA-28555 | NOT NULL パラメータに NULL 値が渡されました。 |

## 純粋さのレベル

定義される純粋さのレベル：NONE

## 関連項目

DBMS_HS_PASSTHROUGH.OPEN_CURSOR
DBMS_HS_PASSTHROUGH.PARSE
DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY
DBMS_HS_PASSTHROUGH.CLOSE_CURSOR

# DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY

## 目的

このファンクションは、SQL 文を実行します。SQL 文が SELECT 文であってはいけません。SQL 文を実行する前に、カーソルをオープンし、SQL 文を解析する必要があります。

## インタフェースの説明

```
FUNCTION EXECUTE_NON_QUERY (c IN BINARY_INTEGER NOT NULL)
RETURN BINARY_INTEGER
```

### パラメータの説明

| パラメータ | 説明 |
|---|---|
| c | パス・スルー SQL 文に関連付けられたカーソル。カーソルは、OPEN_CURSOR ルーチンを使用してオープンし、PARSE ルーチンを使用して解析する必要があります。 |

### 戻り値

非 Oracle システムで、SQL 文により影響のある行数。

### 例外

| 例外 | 説明 |
|---|---|
| ORA-28550 | 渡されたカーソルが無効です。 |
| ORA-28552 | BIND_VARIABLE プロシージャを正しい順序で実行できません。最初にカーソルをオープンしてから、SQL 文を解析しましたか？ |
| ORA-28555 | NOT NULL パラメータに NULL 値が渡されました。 |

## 純粋さのレベル

定義される純粋さのレベル：NONE

## 関連項目

DBMS_HS_PASSTHROUGH.OPEN_CURSOR DBMS_HS_PASSTHROUGH.PARSE

# DBMS_HS_PASSTHROUGH.FETCH_ROW

## 目的

結果セットから行をフェッチします。結果セットは、SQL SELECT 文によって 定義されます。フェッチする行がこれ以上ない場合は、例外 NO_DATA_FOUND に なります。行をフェッチする前に、カーソルをオープンし、SQL 文を解析する必要があります。

## インタフェースの説明

```
FUNCTION FETCH_ROW
(c IN BINARY_INTEGER NOT NULL
[,first IN BOOLEAN])
RETURN BINARY_INTEGER;
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| c | パス・スルー SQL 文に関連付けられたカーソル。カーソルは、OPEN_CURSOR ルーチンを使用してオープンし、PARSE ルーチンを使用して解析する必要があります。 |
| first | SELECT 文を再実行するオプション・パラメータ。可能な値は次のとおりです。<br>■ TRUE: SELECT 文を再実行します。<br>■ FALSE: 次の行をフェッチするか、または初めて実行された場合は SELECT 文を実行して行をフェッチします（デフォルト）。 |

### 戻り値

フェッチした行数。このファンクションでは、最後の行をすでにフェッチ済みの場合、"0" が戻されます。

### 例外

| 例外 | 説明 |
|---|---|
| ORA-28550 | 渡されたカーソルが無効です。 |
| ORA-28552 | プロシージャを正しい順序で実行できません。最初にカーソルをオープンしてから、SQL 文を解析しましたか？ |
| ORA-28555 | NOT NULL パラメータに NULL 値が渡されました。 |

## 純粋さのレベル

定義される純粋さのレベル : WNDS

## 関連項目

DBMS_HS_PASSTHROUGH.OPEN_CURSOR DBMS_HS_PASSTHROUGH.PARSE

# DBMS_HS_PASSTHROUGH.GET_VALUE

## 目的

このプロシージャには次の 2 つの目的があります。

- 行がフェッチされた後で、SELECT 文の選択リスト項目を取り出すこと。
- SQL 文が実行された後で、OUT バインド値を取り出すこと。

## インタフェースの説明

```
PROCEDURE GET_VALUE
 (c IN BINARY_INTEGER NOT NULL,
  pos IN BINARY_INTEGER NOT NULL,
  val    OUT <dty>);
```

<dty> は次のいずれかになります。

- DATE
- NUMBER
- VARCHAR2

RAW データ型の値の取出しについては、GET_VALUE_RAW を参照してください。

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| c | パス・スルー SQL 文に関連付けられたカーソル。カーソルは、OPEN_CURSOR ルーチンを使用してオープンし、PARSE ルーチンを使用して解析する必要があります。 |
| pos | SQL 文でのバインド変数または選択リスト項目の位置。1 から始まります。 |
| val | OUT バインド変数または選択リスト項目がその値を格納する変数。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-1403 | 最後の行をフェッチした後に GET_VALUE を実行すると、NO_DATA_FOUND 例外が戻されます（たとえば、FETCH_ROW によって "0" が戻された場合 )。 |
| ORA-28550 | 渡されたカーソルが無効です。 |
| ORA-28552 | プロシージャを正しい順序で実行できません。最初にカーソルをオープンしてから、SQL 文を解析および実行（またはフェッチ）しましたか？ |
| ORA-28553 | バインド変数の位置が範囲外です。 |
| ORA-28555 | NOT NULL パラメータに NULL 値が渡されました。 |

## 純粋さのレベル

定義される純粋さのレベル : WNDS

## 関連項目

DBMS_HS_PASSTHROUGH.OPEN_CURSOR DBMS_HS_PASSTHROUGH.PARSE DBMS_HS_PASSTHROUGH.FETCH_ROW DBMS_HS_PASSTHROUGH.GET_VALUE_RAW DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE_RAW DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE_RAW

# DBMS_HS_PASSTHROUGH.GET_VALUE_RAW

## 目的

RAW データ型が対象である以外は GET_VALUE と同じです。

## インタフェースの説明

```
PROCEDURE GET_VALUE_RAW
(c IN BINARY_INTEGER NOT NULL,
 pos IN BINARY_INTEGER NOT NULL,
 val OUT RAW);
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| c | パス・スルー SQL 文に関連付けられたカーソル。カーソルは、OPEN_CURSOR ルーチンを使用してオープンし、PARSE ルーチンを使用して解析する必要があります。 |
| pos | SQL 文でのバインド変数または選択リスト項目の位置。1 から始まります。 |
| val | OUT バインド変数または選択リスト項目がその値を格納する変数。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-1403 | 最後の行をフェッチした後に GET_VALUE を実行すると、NO_DATA_FOUND 例外が戻されます（たとえば、FETCH_ROW によって "0" が戻された場合 )。 |
| ORA-28550 | 渡されたカーソルが無効です。 |
| ORA-28552 | プロシージャを正しい順序で実行できません。最初にカーソルをオープンしてから、SQL 文を解析および実行（またはフェッチ）しましたか？ |
| ORA-28553 | バインド変数の位置が範囲外です。 |
| ORA-28555 | NOT NULL パラメータに NULL 値が渡されました。 |

## 純粋さのレベル

定義される純粋さのレベル : WNDS

## 関連項目

DBMS_HS_PASSTHROUGH.OPEN_CURSOR DBMS_HS_PASSTHROUGH.PARSE DBMS_HS_PASSTHROUGH.FETCH_ROW DBMS_HS_PASSTHROUGH.GET_VALUE DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE_RAW DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE_RAW

# DBMS_HS_PASSTHROUGH.OPEN_CURSOR

## 目的

非 Oracle システムでパス・スルー SQL 文を実行するためのカーソルをオープンします。このファンクションは、すべてのタイプの SQL 文についてコールする必要があります。ファンクションは、後続のコールで必要になるカーソルを戻します。このコールにより、メモリーが割り当てられます。対応付けられたメモリーの割当てを解除するには、プロシージャ DBMS_HS_PASSTHROUGH.CLOSE_CURSOR をコールしてください。

## インタフェースの説明

```
FUNCTION OPEN_CURSOR RETURN BINARY_INTEGER;
```

### 戻り値

後続のプロシージャおよびファンクション・コールで使用されるカーソル。

### 例外

| 例外 | 説明 |
|---|---|
| ORA-28554 | オープン・カーソルの最大数を超えました。異機種間サービスの OPEN_CURSORS 初期化パラメータを大きくしてください。 |

## 純粋さのレベル

定義される純粋さのレベル : WNDS、RNDS

## 関連項目

DBMS_HS_PASSTHROUGH.CLOSE_CURSOR

# DBMS_HS_PASSTHROUGH.PARSE

## 目的

非 Oracle システムで SQL 文を解析します。

## インタフェースの説明

```
PROCEDURE GET_VALUE_RAW
(c IN BINARY_INTEGER NOT NULL,
 stmt IN VARCHAR2 NOT NULL);
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| c | パス・スルー SQL 文に関連付けられたカーソル。カーソルはファンクション OPEN_CURSOR を使用してオープンされる必要があります。 |
| stmt | 解析される文。 |

### 例外

| 例外 | 説明 |
|---|---|
| ORA-28550 | 渡されたカーソルが無効です。 |
| ORA-28551 | SQL 文が適切でありません。 |
| ORA-28555 | NOT NULL パラメータに NULL 値が渡されました。 |

## 純粋さのレベル

定義される純粋さのレベル : WNDS、RNDS

## 関連項目

DBMS_HS_PASSTHROUGH.OPEN_CURSOR DBMS_HS_PASSTHROUGH.PARSE DBMS_HS_PASSTHROUGH.FETCH_ROW DBMS_HS_PASSTHROUGH.GET_VALUE DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE_RAW DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE_RAW

# D

# DBMS_DISTRIBUTED_TRUST_ADMIN パッケージ・リファレンス

この付録では、Trusted Server リストを管理するための、パッケージ DBMS_DISTRIBUTED_TRUST_ADMIN のプロシージャおよびファンクションについて説明します。

> **注意：** Oracle8 で使用可能だった Oracle Security Serve 機能は修正中であり、現在はベータ版のユーザーのみ使用可能です。Oracle8*i* では、今後のリリースに組み込まれる予定です。

データ・ディクショナリ・ビュー TRUSTED_SERVERS を使用して、データベースから信頼されている（いない）データベースを参照できます。

この付録では次の事項を参照できます。

- DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ ALL
- DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_ ALL
- DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER (SERVER IN VARCHAR2)
- DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER (SERVER IN VARCHAR2)

# DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ ALL

## 目的

DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ALL は、Trusted Database リストを空にして、すべてのサーバーが信頼されていないことを指定するエントリを挿入します。TRUSTED_SERVERS ビューには、現在どのサーバーも信頼されていないことを示す、"UNTRUSTED ALL" が表示されます。その後、DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER を使用して、特定のサーバーのアクセスを可能にできます。

## インタフェースの説明

```
PROCEDURE deny_all
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| なし | |

### 例外

| 例外 | 説明 |
|---|---|
| なし | |

## 純粋さのレベル

定義される純粋さのレベル：なし

## 関連項目

DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_ALL

# DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_ ALL

## 目的

DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_ALL は、Trusted Database リストを空にして、Oracle Security Server などのように中央認可レベルで信頼されている、すべてのサーバーのアクセスを可能にすることを指定します。

TRUSTED_SERVERS ビューには、現在、すべてのサーバーが Oracle Security Server などのように中央認可レベルで信頼されていることを示す、"TRUSTED ALL" が表示されます。

特定のサーバーを信頼されないようにするには、DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER を使用できます。

## インタフェースの説明

```
PROCEDURE allow_all
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| なし | |

### 例外

| 例外 | 説明 |
|---|---|
| なし | |

## 純粋さのレベル

定義される純粋さのレベル : なし

## 関連項目

DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ALL

DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER

# DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER (SERVER IN VARCHAR2)

## 目的

指定するサーバーが信頼されていることを確認してください（事前に "deny all" を指定した場合を含む）。

Trusted Server リストに "deny all" エントリが含まれている場合は、このプロシージャによって、特定のデータベース（たとえば DBx）が信頼されていることを示す指定が追加されます。

Trusted Server リストに "allow all" エントリが含まれていて、このリストに "deny DBx" エントリがない場合、このプロシージャを実行しても、何も変更されません。

Trusted Server リストに "allow all" エントリが含まれていて、このリストに "deny DBx" がある場合 、"deny DBx" エントリは削除されます。

## インタフェースの説明

```
PROCEDURE allow_server(server IN VARCHAR2) SERVER_NAME
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| SERVER | 信頼されるサーバーの一意の完全修飾名 |

### 例外

| 例外 | 説明 |
|---|---|
| なし | |

## 純粋さのレベル

定義される純粋さのレベル : なし

# DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER (SERVER IN VARCHAR2)

## 目的

指定するサーバーが信頼されていないことを確認してください（事前に "allow all" を指定した場合を含む）。

Trusted Server リストに "allow all" エントリが含まれていると、このプロシージャによって、特定のデータベース（たとえば DBx）が信頼されていないことを示す指定が追加されます。

Trusted Server リストに "deny all" エントリが含まれていて、このリストに "allow DBx" エントリがない場合、このプロシージャによって何も変更されません。

Trusted Server リストに "deny all" エントリが含まれていて、このリストに "allow DBx" エントリがある場合、このプロシージャによって "allow DBx" エントリは削除されます。

## インタフェースの説明

```
PROCEDURE deny_server(server IN VARCHAR2)
```

### パラメータおよび説明

| パラメータ | 説明 |
|---|---|
| SERVER | 信頼されないサーバーの一意の完全修飾名 |

### 例外

| 例外 | 説明 |
|---|---|
| なし | |

## 純粋さのレベル

定義される純粋さのレベル：なし

# 索引

## 数字



## A



## B



## C



## D

## E



## F



## G



## H



## I



## L



## N



## O



## P

## R



## S



## T



## U



## あ



## い



## え

## お



## か



## き



## く



## け

## こ



## さ



## し



## す



## せ

## た



## ち



## て



## と

## ね



## は



## ひ



## ふ

## へ



## ほ



## ま



## め



## ゆ

## よ



## ら



## り



## れ



## ろ