Oracle9*i*

Streams

リリース 2（9.2）

2003 年 2 月

部品番号：J06285-02

ORACLE®

Oracle9*i* Streams, リリース 2（9.2）

部品番号：J06285-02

原本名：Oracle9*i* Streams, Release 2 (9.2)

原本部品番号：B10007-01, B10008-01

原本著者：Randy Urbano

原本協力者：Valarie Moore, Nimar Arora, Lance Ashdown, Ram Avudaiappan, Sukanya Balaraman, Neerja Bhatt, Ragamayi Bhyravabhotla, Diego Cassinera, Debu Chatterjee, Alan Downing, Lisa Eldridge, Curt Elsbernd, Yong Feng, Jairaj Galagali, Brajesh Goyal, Sanjay Kaluskar, Lewis Kaplan, Anand Lakshminath, Jing Liu, Edwina Lu, Raghu Mani, Pat McElroy, Krishnan Meiyyappan, Shailendra Mishra, Tony Morales, Bhagat Nainani, Anand Padmanaban, Maria Pratt, Arvind Rajaram, Viv Schupmann, Vipul Shah, Neeraj Shodhan, Wayne Smith, Benny Souder, Jim Stamos, Janet Stern, Mahesh Subramaniam, Bob Thome, Ramkumar Venkatesan, Wei Wang, Lik Wong, David Zhang

目次

2 Streams の取得プロセス

3 Streams のステージングと伝播

4 Streams 適用プロセス

6 Streams でのルールの使用方法

7 Streams 競合解消

8 Streams のタグ

9 Streams 異機種間での情報の共有

10 Streams 高可用性環境

第 II 部　Streams の管理

11　Streams 環境の構成

12　取得プロセスの管理

13 ステージングと伝播の管理

14 適用プロセスの管理

15 ルールおよびルールベースの変換の管理

16 その他の Streams 管理タスク

17 Streams 環境の監視

18 Streams 環境のトラブルシューティング

21 簡単な単一のソース・レプリケーションの例

22 単一ソースの異機種間レプリケーションの例

23 複数のソース・レプリケーションの例

24 ルールベースのアプリケーションの例

第 IV 部　付録

A　LCR 用の XML Schema

索引

はじめに

このマニュアルでは、Streams の特徴と機能を説明します。このマニュアルには、Streams の概念と、Streams 環境の構成および管理方法の説明が含まれています。さらに、Streams メッセージ環境、Streams レプリケーション環境およびルールベース・アプリケーションの詳細な構成例も含まれます。

この章の内容は次のとおりです。

対象読者

このマニュアルは、Streams 環境の作成とメンテナンスを担当するデータベース管理者を対象としています。これらの管理者は、次の 1 つ以上のタスクを実行します。

- Streams 環境のプラン作成
- Streams 環境の構成
- Streams 環境における競合解消の構成
- Streams 環境の管理
- Streams 環境の監視
- 必要なトラブルシューティング・アクティビティの実行

このマニュアルを使用するには、リレーショナル・データベースの概念、SQL、分散データベースの管理、アドバンスト・キューイングの概念、PL/SQL および Streams 環境を稼働させるオペレーティング・システムに関する十分な知識が必要です。

このマニュアルの構成

このマニュアルは次の部と章で構成されています。

第 I 部「Streams の概要」

第 I 部の各章では、Streams に関連する概念について説明します。

第 1 章「Streams の概要」

Streams の主な機能と使用方法について説明します。

第 2 章「Streams の取得プロセス」

Streams の取得プロセスの概念について説明します。これには、論理変更レコード（LCR）、データ型、取得される変更のタイプ、サプリメンタル・ロギングおよび取得プロセス・アーキテクチャに関する情報が含まれます。

第 3 章「Streams のステージングと伝播」

Streams 環境でのステージングと伝播の概念について説明します。これには、取得イベントとユーザーがエンキューするイベントの違い、伝播、トランザクション型のキューと非トランザクション型のキューの違いおよび `SYS.AnyData` キューの使用方法などが含まれます。また、キューと伝播のアーキテクチャについても説明します。

第 4 章「Streams 適用プロセス」

Streams の適用プロセスの概念について説明します。これには、適用プロセスによるイベント処理、変更を表に適用する際の考慮事項、DDL 変更を適用する際の条件、トリガーの起動プロパティの制御、適用プロセスに関する最も古い SCN および適用プロセスのアーキテクチャに関する情報が含まれます。

第 5 章「ルール」

ルールの概念について説明します。これには、ルールのコンポーネント、ルール・セットおよびルール関連の権限に関する情報が含まれます。

第 6 章「Streams でのルールの使用方法」

Streams でルールを使用する方法の概要について説明します。これには、表レベルのルール、サブセット・ルール、スキーマ・レベルのルールおよびグローバル・レベルのルールに関する情報が含まれます。また、ルールベースの変換についても説明します。

第 7 章「Streams 競合解消」

競合の概念について説明します。これには、Streams 環境における競合に可能なタイプ、競合検出、競合防止および競合解消に関する情報が含まれます。

第 8 章「Streams のタグ」

Streams タグの概念について説明します。これには、タグ値をルールに使用する方法、適用プロセス用にタグ値を設定する方法およびタグを使用して変更サイクルを回避する方法が含まれます。

第 9 章「Streams 異機種間での情報の共有」

Streams を使用した異機種間での情報共有の概念について説明します。これには、Oracle データベース内の情報を Oracle 以外のデータベースと共有する方法、Oracle 以外のデータベース内の情報を Oracle データベースと共有する方法、および Streams を使用して Oracle 以外の 2 つのデータベース間で情報を共有する方法が含まれます。

第 10 章「Streams 高可用性環境」

高可用性環境を実現するために Streams を使用する場合の概念を説明します。

第 II 部「Streams の管理」

第 II 部の各章では、取得プロセス、ステージング、伝播、適用プロセス、ルール、ルールベースの変換、論理変更レコード（LCR）および Streams タグの管理について説明します。

第 11 章「Streams 環境の構成」

Streams 環境の準備について説明します。これには、Streams 管理者の構成、Streams に重要な初期化パラメータの設定、取得プロセスの準備およびネットワーク接続性の構成に関する手順が含まれます。

第 12 章「取得プロセスの管理」

取得プロセスの管理について説明します。これには、取得プロセスの作成、起動、停止および変更の各手順と、取得プロセスの管理に関連するその他の情報が含まれます。

第 13 章「ステージングと伝播の管理」

Streams 環境でのイベントのステージングと伝播の管理について説明します。これには、Streams キューの作成手順、伝播の有効化、無効化および変更のための手順と、ステージング、伝播およびメッセージングに関連するその他の情報が含まれます。

第 14 章「適用プロセスの管理」

適用プロセスの管理について説明します。これには、適用プロセスの作成、起動、停止および変更の各手順と、適用プロセス・ハンドラの使用、競合解消の構成および例外キューの管理に関する各手順が含まれます。

第 15 章「ルールおよびルールベースの変換の管理」

ルールとルールベースの変換の管理について説明します。これには、ルールおよびルール・セットの管理手順と、ルールに関連する権限の付与と取消しに関する情報が含まれます。また、この章では、ルールベースの変換の作成、変更および削除の手順についても説明します。

第 16 章「その他の Streams 管理タスク」

論理変更レコード（LCR）と Streams タグの管理について説明します。これには、LCR の構成手順およびエンキュー手順と、セッションまたは適用プロセスのタグ値の設定手順および削除手順が含まれます。

第 17 章「Streams 環境の監視」

データ・ディクショナリ・ビューとスクリプトを使用して Streams 環境を監視する方法について説明します。これには、取得プロセス、キュー、伝播、適用プロセス、ルール、ルールベースの変換およびタグを監視する方法が含まれます。

第 18 章「Streams 環境のトラブルシューティング」

Streams 環境に考えられる問題とその解決方法について説明します。これには、取得プロセス、伝播、適用プロセスおよび Streams のルールのトラブルシューティングと、トレース・ファイルとアラート・ログで問題をチェックする方法が含まれます。

第 III 部「環境とアプリケーションの例」

第 III 部の各章では、環境の例を挙げて説明します。

第 19 章「Streams メッセージングの例」

Streams を使用してメッセージ環境を構成する手順の例を示します。

第 20 章「単一データベースの取得および適用の例」

Streamsを使用して単一データベースにおける取得と適用例を構成する手順の例を示します。具体的には、この章では、単一データベースで、表に対する変更を取得し、適用時に DML ハンドラを使用して取得済変更をキューに再エンキューし、さらに変更のサブセットを別の表に適用する例を示します。

第 21 章「簡単な単一のソース・レプリケーションの例」

Streams を使用して単純な単一ソース・レプリケーション環境を構成する手順の例を示します。

第 22 章「単一ソースの異機種間レプリケーションの例」

Streams を使用して単一ソース異機種間レプリケーション環境を構成する手順の例を示します。さらに、この環境にオブジェクトおよびデータベースを追加する手順の例を示します。

第 23 章「複数のソース・レプリケーションの例」

Streams を使用して複数ソース・レプリケーション環境を構成する手順の例を示します。

第 24 章「ルールベースのアプリケーションの例」

Oracle ルール・エンジンを使用するルールベース・アプリケーションに関する手順の例を示します。

第 IV 部「付録」

論理変更レコード（LCR）用の XML Schema に関する付録が含まれています。

付録 A「LCR 用の XML Schema」

LCR 用の XML Schema の定義について説明します。

関連文書

詳細は、次の Oracle リソースを参照してください。

- 『Oracle9*i* データベース概要』
- 『Oracle9*i* データベース管理者ガイド』
- 『Oracle9*i* SQL リファレンス』
- 『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』
- 『PL/SQL ユーザーズ・ガイドおよびリファレンス』
- 『Oracle9*i* データベース・ユーティリティ』
- 『Oracle9*i* Heterogeneous Connectivity Administrator's Guide』
- 『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』
- Oracle Enterprise Manager の Streams ツールに関する Streams オンライン・ヘルプ

特定の内容の詳細情報は、Oracle9*i* マニュアル・セットの他のマニュアルに記載されている場合があります。

このマニュアルに記載されている多数の例は、Oracle とともにデフォルトでインストールされる、シード・データベースのサンプル・スキーマを使用しています。これらのスキーマがどのように作成されているかと、その使用方法については、『Oracle9*i* サンプル・スキーマ』を参照してください。

リリース・ノート、インストレーション・マニュアル、ホワイト・ペーパーまたはその他の関連文書は、次の OTN-J（Oracle Technology Network Japan）に接続すれば、無償でダウンロードできます。

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

OTN-J を使用するには、オンラインでの登録が必要です。OTN-J のユーザー名とパスワードを取得済みであれば、次の OTN-J Web サイトの文書セクションに直接接続できます。

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

表記規則

このマニュアル・セットの本文とコード例に使用されている表記規則について説明します。

本文の表記規則

本文中には、特別な用語が一目でわかるように様々な表記規則が使用されています。次の表に、本文の表記規則と使用例を示します。

規則	意味	例
太字	太字は、本文中に定義されている用語または用語集に含まれている用語、あるいはその両方を示します。	この句を指定する場合は、**索引構成表**を作成します。
固定幅フォントの大文字	固定幅フォントの大文字は、システムにより指定される要素を示します。この要素には、パラメータ、権限、データ型、Recovery Manager キーワード、SQL キーワード、SQL*Plus またはユーティリティ・コマンド、パッケージとメソッド、システム指定の列名、データベース・オブジェクトと構造体、ユーザー名、およびロールがあります。	この句は、NUMBER 列に対してのみ指定できます。 BACKUP コマンドを使用すると、データベースのバックアップを作成できます。 USER_TABLES データ・ディクショナリ・ビューの TABLE_NAME 列を問い合せます。 DBMS_STATS.GENERATE_STATS プロシージャを使用します。
固定幅フォントの小文字	固定幅フォントの小文字は、実行可能ファイル、ファイル名、ディレクトリ名およびサンプルのユーザー指定要素を示します。この要素には、コンピュータ名とデータベース名、ネット・サービス名、接続識別子の他、ユーザー指定のデータベース・オブジェクトと構造体、列名、パッケージとクラス、ユーザー名とロール、プログラム・ユニット、およびパラメータ値があります。 **注意：**一部のプログラム要素には、大文字と小文字の両方が使用されます。この場合は、記載されているとおりに入力してください。	sqlplus と入力して SQL*Plus をオープンします。 パスワードは orapwd ファイルに指定されています。 データ・ファイルと制御ファイルのバックアップを /disk1/oracle/dbs ディレクトリに作成します。 department_id、department_name および location_id の各列は、hr.departments 表にあります。 初期化パラメータ QUERY_REWRITE_ENABLED を true に設定します。 oe ユーザーとして接続します。 これらのメソッドは JRepUtil クラスに実装されます。

規則	意味	例
固定幅フォントの小文字のイタリック	固定幅フォントの小文字のイタリックは、プレースホルダまたは変数を示します。	*parallel_clause* を指定できます。 U*old_release*.SQL を実行します。*old_release* は、アップグレード前にインストールしたリリースです。

コード例の表記規則

コード例は、SQL、PL/SQL、SQL*Plus またはその他のコマンドラインを示します。次のように、固定幅フォントで、通常の本文とは区別して記載しています。

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

次の表に、コード例の記載上の表記規則と使用例を示します。

規則	意味	例
[ ]	大カッコで囲まれている項目は、1 つ以上のオプション項目を示します。大カッコ自体は入力しないでください。	DECIMAL (*digits* [, *precision*])
{ }	中カッコで囲まれている項目は、そのうちの 1 つのみが必要であることを示します。中カッコ自体は入力しないでください。	{ENABLE | DISABLE}
|	縦線は、大カッコまたは中カッコ内の複数の選択肢を区切るために使用します。オプションのうち 1 つを入力します。縦線自体は入力しないでください。	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
...	水平の省略記号は、次のどちらかを示します。	
	■ 例に直接関係のないコード部分が省略されいること。	CREATE TABLE ...AS *subquery*;
	■ コードの一部が繰返し可能であること。	SELECT *col1*, *col2*, ..., *coln* FROM employees;
. . .	垂直の省略記号は、例に直接関係のない数行のコードが省略されていることを示します。	SQL> SELECT NAME FROM V$DATAFILE; NAME ---------- /fs1/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fs1/dbs/tbs_09.dbf 9 rows selected.

規則	意味	例
その他の表記	大カッコ、中カッコ、縦線および省略記号以外の記号は、示されているとおりに入力してください。	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
イタリック	イタリックの文字は、特定の値を指定する必要のあるプレースホルダまたは変数を示します。	`CONNECT SYSTEM/`*`system_password`* `DB_NAME = `*`database_name`*

大文字	大文字は、システムにより指定される要素を示します。これらの用語は、ユーザー定義用語と区別するために大文字で記載されています。大カッコで囲まれている場合を除き、記載されているとおりの順序とスペルで入力してください。ただし、この種の用語は大 / 小文字区別がないため、小文字でも入力できます。	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

小文字	小文字は、ユーザー指定のプログラム要素を示します。たとえば、表名、列名またはファイル名を示します。
	注意：一部のプログラム要素には、大文字と小文字の両方が使用されます。この場合は、記載されているとおりに入力してください。
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

第 I 部

Streams の概要

第 I 部では、Streams の概要について説明します。第 I 部の構成は、次のとおりです。

1

Streams の概要

この章では、Oracle Streams に関連する基本的な概念と用語について説明します。これらの概念の詳細は、他の章を参照してください。

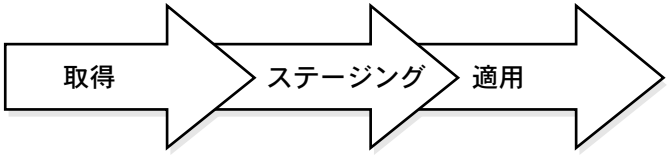
この章の内容は次のとおりです。

Streams の概要

Oracle Streams を使用すると、ストリーム内のデータとイベントを共有できます。ストリームにより、データベース内またはデータベース間でこの情報を伝播できます。ストリームにより、指定の情報が指定の宛先にルーティングされます。その結果、イベントの取得と管理、および他のデータベースやアプリケーションとのイベントの共有に関して、従来のソリューションよりも機能性と柔軟性に優れた新機能が得られます。Streams により、ソリューション間でトレードオフを検討するサイクルから解放されます。Oracle Streams には、分散エンタープライズ・アプリケーション、データ・ウェアハウスおよび高可用性ソリューションの構築と運用に必要な機能が用意されています。Oracle Streams の機能は、すべて同時に使用できます。変更が必要な場合は、既存の機能を損なわずに Streams の新機能を実装できます。

Oracle Streams を使用して、ストリームに入れる情報、ストリームのフローまたはデータベース間でのルーティング、各データベースに送られるストリーム内のイベントに発生する処理およびストリームの終了方法を制御します。Streams の特定の機能を構成することで、特定の要件に対処できます。Streams では、仕様に基づいて、データ操作言語（DML）の変更やデータ定義言語（DDL）の変更など、データベース内のイベントを自動的に取得、ステージングおよび管理できます。また、ユーザー定義イベントをストリームに入れることも可能です。その場合、Streams では、その情報を他のデータベースやアプリケーションに自動的に伝播できます。さらに、Streams では、ユーザーによって設定された仕様に基づいて接続先データベースでイベントを適用できます。図 1-1 に、Streams の情報の流れを示します。

図 1-1 Streams での情報の流れ



Streams の機能

Streams を使用すると、次のことができます。

- データベースでの変更の**取得**。

  表、スキーマまたはデータベース全体に対して行われた変更を取得するように、バックグラウンドの**取得プロセス**を構成できます。取得プロセスは、REDO ログから変更を取得し、取得した変更をそれぞれ論理変更レコード（**LCR**）形式でフォーマットします。REDO ログ内で変更が生成されるデータベースを、**ソース・データベース**と呼びます。

- キューへの**イベント**のエンキュー。Streams のキューでは、LCR およびユーザー・メッセージという 2 種類のイベントをステージングできます。

  取得プロセスは LCR イベントを指定されたキューにエンキューします。キューでは、同じデータベース内で、または他のデータベースと、LCR イベントを共有できます。

  また、ユーザー・アプリケーションでユーザー・イベントを明示的にエンキューすることもできます。このように明示的にエンキューされるイベントは、LCR でもユーザー・メッセージでもかまいません。

- キュー間でのイベントの伝播。これらのキューは、同じデータベースにあっても異なるデータベースにあってもかまいません。

- イベントのデキュー。

  バックグラウンドの**適用プロセス**でイベントをデキューできます。また、ユーザー・アプリケーションでイベントを明示的にデキューすることもできます。

- データベースでのイベントの**適用**。

  適用プロセスを、キュー内のすべてのイベントまたは指定したイベントのみを適用するように構成できます。また、独自の PL/SQL サブプログラムをコールしてイベントを処理するように構成することもできます。

  LCR イベントが適用され、他のタイプのイベントが処理されるデータベースを、**接続先データベース**と呼びます。構成によっては、ソース・データベースと接続先データベースが同一の場合があります。

その他、Streams には次の機能があります。

- 有向ネットワーク
- 自動競合検出および解消
- 変換
- 異機種間での情報の共有

この章では、これらの機能の概要について説明します。詳細は、他の章を参照してください。

Streams を使用する理由

ここでは、Streams を使用する理由について説明します。

メッセージ・キューイング

Streams を使用すると、ユーザー・アプリケーションは各種のメッセージをエンキューし、メッセージをサブスクライバ・キューに伝播し、メッセージをコンシューム（使用）する準備ができたことをユーザー・アプリケーションに通知し、接続先データベースでメッセージをデキューできます。Streams により、SYS.AnyData 型のメッセージをステージングする新しいタイプのキューが導入されます。ほとんどのタイプのメッセージは、SYS.AnyData ラッパーでラップして SYS.AnyData キューでステージングできます。Streams は、マルチ・コンシューマ・キュー、パブリッシュおよびサブスクライブ、コンテンツ・ベースのルーティング、インターネット伝播、変換および他のメッセージ・サブシステムへのゲートウェイなど、メッセージ・キューイング・システムのすべての標準機能がサポートされるアドバンスト・キューイング（AQ）と相互運用できます。

> **関連項目：** AQ の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

データ・レプリケーション

Streams では、データベース・オブジェクトに対する DML と DDL の変更を効率的に取得し、他の 1 つ以上のデータベースにレプリケートできます。Streams の取得プロセスは、ソース・データベース・オブジェクトに対する変更を取得し、それを接続先データベースに伝播して Streams の適用プロセスで適用できる LCR 形式でフォーマットします。

接続先データベースでは、同じデータベース・オブジェクトに対する DML および DDL 変更を許可できます。これらの変更は、環境内の他のデータベースに伝播される場合と、伝播されない場合があります。つまり、Streams 環境は、変更を伝播する 1 つのデータベースを使用して構成する方法と、データベース間で変更が両方向に伝播するように構成する方法があります。また、データが共有される表をすべてのデータベースで同一コピーにする必要はありません。これらの表の構造と内容はデータベースごとに異なっていてもよく、各表の情報をこれらのデータベース間で共有できます。

データ・ウェアハウスのロード

データ・ウェアハウスのロードは、データ・レプリケーションの特殊なケースです。データ・ウェアハウスの作成と管理のうち最も重要なタスクには、既存データのリフレッシュと業務系データベースからの新規データの追加があります。Streams では、本番システムに対して行われた変更を取得し、それをステージング・データベースに送信するか、データ・ウェアハウスまたはオペレーショナル・データ・ストアに直接送信できます。Streams は REDO ログの情報を取得することで、本番システムに発生する不要なオーバーヘッドを回避します。データ変換とユーザー定義の適用手順がサポートされることで、データがロードされるときのデータの再フォーマットやウェアハウス固有のデータ・フィールドの更新に必要な柔軟性が得られます。

関連項目： データ・ウェアハウスの詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

データ保護

データ保護に関する解決策の 1 つは、本番データベースのローカル・コピーまたはリモート・コピーを作成することです。人為的なエラーや災害が発生した場合は、コピーを使用して処理を再開できます。Streams を使用すると、柔軟な高可用性環境を構成できます。また、Oracle Data Guard は内部的に Streams を使用したデータ保護機能であり、この機能を使用して、本番データベースと論理的に等価のスタンバイ・コピーであるロジカル・スタンバイ・データベースを作成し、メンテナンスできます。Streams のレプリケーションの場合と同様に、取得プロセスは REDO ログ内の変更を取得し、これらの変更を LCR 形式でフォーマットします。これらの LCR は、スタンバイ・データベースで適用されます。スタンバイ・データベースは読取り / 書込み用に完全にオープンされており、特殊な索引や他のデータベース・オブジェクトを含んでいる場合があります。したがって、これらのスタンバイ・データベースを更新の適用中に問い合せることができます。これにより、Oracle Data Guard は本番データベースを時間のかかる問合せによる負荷から解放する優れたソリューションとなります。

ロジカル・スタンバイ・データベースと Streams データ・レプリケーション環境の最も顕著な違いは、変更が取得される場所にあります。ロジカル・スタンバイ・データベースでの更新は、できるだけ早期にリモート・サイトに移動することが重要です。これにより、障害発生時にも、トランザクション消失の可能性が最小限ですみます。リモート・データベースで REDO ログを同期式で直接書き込むことで、障害発生時にもデータ消失が発生しません。スタンバイ・システム側では、変更が取得され、適用プロセスによってスタンバイ・データベースに直接適用されます。

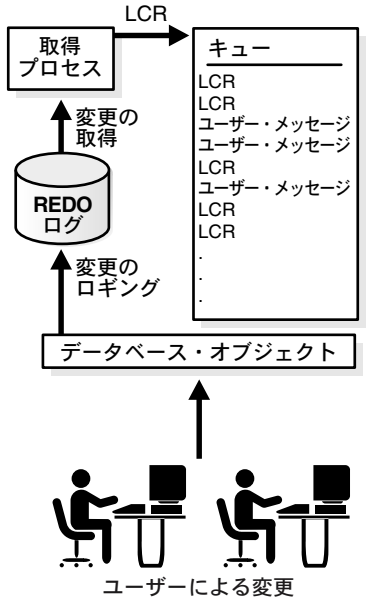
関連項目：

- 第 10 章「Streams 高可用性環境」
- ロジカル・スタンバイ・データベースの詳細は、『Oracle9*i* Data Guard 概要および管理』を参照してください。

取得プロセスの概要

Oracle データベース内でデータベース・オブジェクトに対して行われた変更は、ユーザー・エラーやメディア障害が発生した場合にもリカバリを保証するために、REDO ログに記録されます。取得プロセスは Oracle バックグラウンド・プロセスであり、データベースの REDO ログを読み込んで、データベース・オブジェクトに対する DML と DDL の変更を取得します。取得プロセスは、これらの変更を LCR と呼ばれるイベント形式でフォーマットし、キューにエンキューします。LCR には 2 種類あり、**行 LCR** には、DML 操作によって生じる表内の行に対する変更の情報が含まれ、**DDL LCR** には、データベース・オブジェクトに対する DDL 変更の情報が含まれます。どの変更を取得するかは、ルールを使用して指定します。図 1-2 に、LCR を取得する取得プロセスを示します。

図 1-2 取得プロセス



注意： 取得プロセスは、一部の DML 変更や DDL 変更は取得せず、SYS または SYSTEM スキーマ内で行われた変更も取得しません。

特定のセッションまたは適用プロセスによって生成される REDO エントリ用に、Streams タグを指定できます。これらのタグは、取得プロセスによって取得される LCR の一部となります。タグを使用すると、REDO エントリや LCR に含まれている変更の発生場所が、ローカル・データベースであるか他のデータベースであるかを判断し、発生場所となったデータベースに LCR が送信されるのを回避できます。タグは他の LCR の追跡にも使用できます。また、タグを使用して、LCR ごとに接続先データベースのセットを指定することもできます。

関連項目：

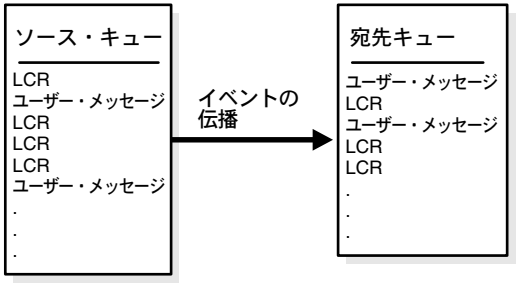
- 取得プロセスと、取得プロセスによって取得される DML 文および DDL 文の詳細は、第 2 章「Streams の取得プロセス」を参照してください。
- 第 8 章「Streams のタグ」

イベントのステージングと伝播の概要

Streams では、キューを使用してイベントが伝播またはコンシュームのためにステージングされます。Streams を使用すると、キューが同一データベース内にあるか異なるデータベース内にあるかに関係なく、あるキューから別のキューにイベントを伝播させることができます。イベントの伝播元となるキューは**ソース・キュー**、イベントを受信するキューは**宛先キュー**と呼ばれます。ソース・キューと宛先キューの間には、1 対多、多対 1 または多対多の関連が可能です。

キュー内でステージングされるイベントは、Streams の適用プロセスまたはユーザー定義のサブプログラムでコンシュームできます。ソース・キューから宛先キューに変更を伝播するように伝播を構成する場合は、ルールを使用して伝播対象となる変更を指定できます。図 1-3 に、ソース・キューから宛先キューへの伝播を示します。

図 1-3 ソース・キューから宛先キューへの伝播

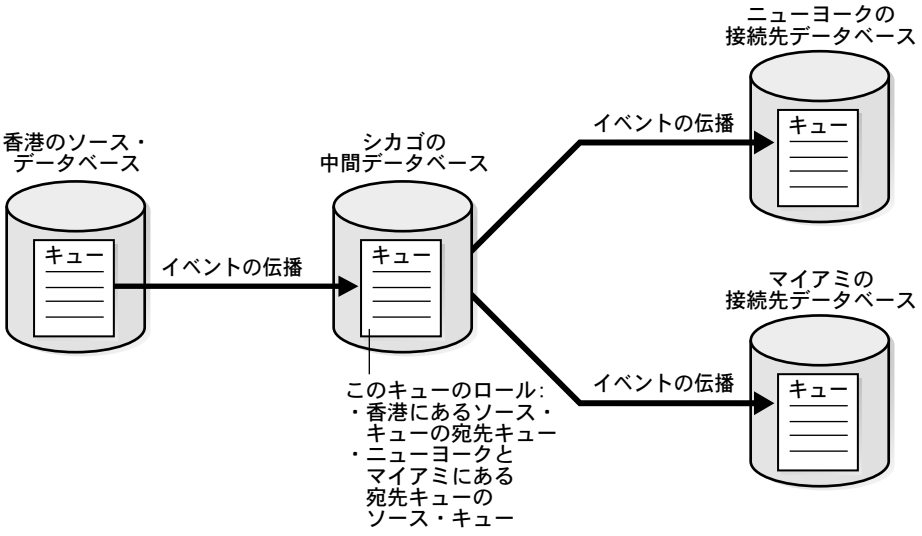


有向ネットワークの概要

Streams では、**有向ネットワーク**を介して変更が共有される環境を構成できます。有向ネットワークとは、伝播されるイベントが接続先データベースに到達する前に 1 つ以上の中間データベースを通過するネットワークです。イベントは、中間データベースで処理される場合も処理されない場合もあります。Streams を使用すると、各接続先データベースに伝播させるイベントを選択し、イベントが接続先データベースに到達するまでのルートを指定できます。

図 1-4 に、有向ネットワーク環境の例を示します。この例では、シカゴにある中間データベースのキューが、ソース・キューと宛先キューの両方を兼ねていることに注意してください。

図 1-4 有向ネットワーク環境の例

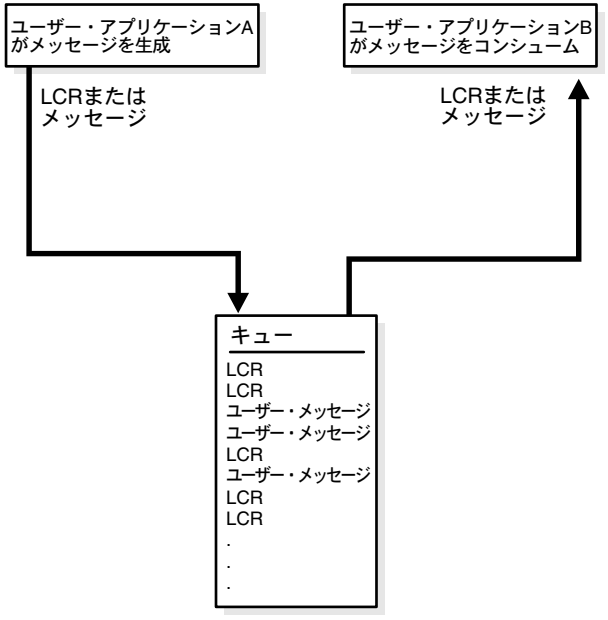


関連項目： ステージングと伝播の詳細は、第 3 章「Streams のステージングと伝播」を参照してください。

イベントの明示的エンキューおよびデキュー

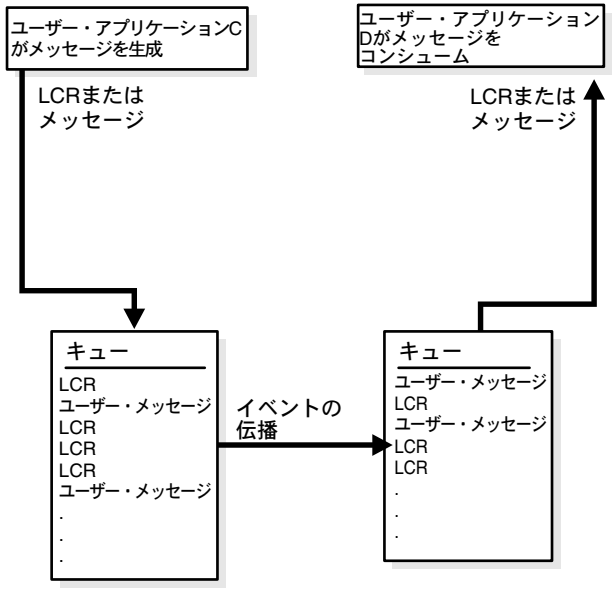
ユーザー・アプリケーションでは、イベントを明示的にエンキューできます。また、これらのイベントを、適用プロセスによって接続先データベースで適用できるように LCR としてフォーマットできます。または、これらのイベントを、他のユーザー・アプリケーションでコンシュームするためにユーザー・メッセージとしてフォーマットすることもできます。この場合、イベントは明示的にデキューされるか、適用プロセスからのコールバックを使用して処理されます。明示的にエンキューされたイベントは、同じキューから明示的にデキューできます。図 1-5 に、同じキューでのイベントの明示的なエンキューとデキューを示します。

図 1-5 単一キューでのイベントの明示的なエンキューおよびデキュー



イベントがキュー間で伝播する場合、ソース・キューに明示的にエンキューされたイベントは、適用プロセスの介入なしにユーザー・アプリケーションが宛先キューから明示的にデキューできます。図 1-6 に、ソース・キューへのイベントの明示的なエンキュー、宛先キューへの伝播および宛先キューからのイベントの明示的なデキューを示します。

図 1-6 イベントの明示的エンキュー、伝播およびデキュー



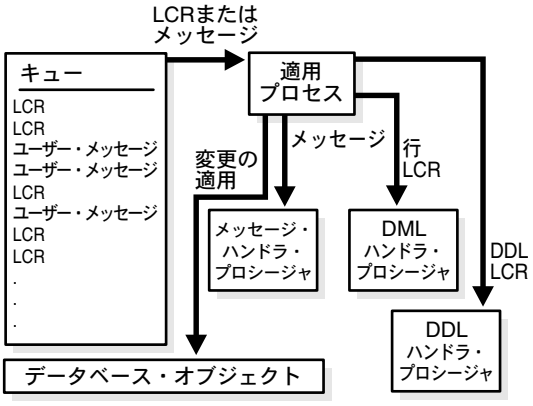
関連項目： イベントの明示的エンキューおよびデキューの詳細は、3-10 ページの「SYS.AnyData 型のキューとユーザー・メッセージ」を参照してください。

適用プロセスの概要

適用プロセスは Oracle バックグラウンド・プロセスであり、キューからイベントをデキューし、各イベントをデータベース・オブジェクトに直接適用するか、適用ハンドラと呼ばれるユーザー定義プロシージャにパラメータとして渡します。これらの適用ハンドラには、メッセージ・ハンドラ、DML ハンドラおよび DDL ハンドラがあります。

通常、適用プロセスは、イベントを稼働中のローカル・データベースに適用しますが、異機種間データベース環境では、Oracle 以外のリモート・データベースにイベントを適用するように構成できます。キュー内のどのイベントを適用するかは、ルールを使用して指定します。図 1-7 に、LCR とユーザー・メッセージを処理する適用プロセスを示します。

図 1-7 適用プロセス



関連項目： 第 4 章「Streams 適用プロセス」

自動競合検出および解消

適用プロセスは、LCR を直接適用するときに、競合を自動的に検出します。通常、競合が生じるのは、ソース・データベース内と接続先データベース内で同じ行がほぼ同時に変更される場合です。

競合が発生した場合は、競合がビジネス・ルールに従って確実に解消されるメカニズムが必要です。Streams には、様々なビルトイン競合解消ハンドラが用意されています。これらのビルトイン・ハンドラを使用すると、データベースごとに、ビジネス・ルールに従って競合を解消する競合解消システムを定義できます。Oracle のビルトイン競合解消ハンドラでは解消できない固有の状況がある場合は、独自の競合解消ハンドラを作成できます。

競合が解消されない場合や、ハンドラ・プロシージャにエラーが発生した場合、エラーを発生させたトランザクション内のすべてのイベントは、後で分析して可能であれば再実行できるように例外キューに保存されます。

関連項目： 第 7 章「Streams 競合解消」

ルールの概要

Streams では、どの情報をどのような場合に共有するかを、**ルール**を使用して制御できます。ルールは、SQL 問合せの WHERE 句での条件に類似した条件を指定します。関連するルールは**ルール・セット**としてグループ化できます。ルールの構成要素は、次のとおりです。

- **ルール条件**は、1 つ以上の式と演算子を組み合せてブール値を戻します。ブール値は、イベントに基づく TRUE、FALSE または NULL（不明）のいずれかです。
- **ルール評価コンテキスト**では、ルール条件内で参照できる外部データを定義します。外部データは、外部変数または表データ、あるいはその両方として存在します。
- **ルール・アクション・コンテキスト**はルールに関連付けられたオプションの情報であり、ルールが評価されるときにルール・エンジンのクライアントによって解析されます。

たとえば、表を所有するスキーマの名前が hr、表名が departments である場合に TRUE と評価されるように指定するには、次のルール条件を Streams で使用します。

```
:dml.get_object_owner() = 'hr' AND :dml.get_object_name() = 'departments'
```

Streams 環境では、このルール条件を次のように使用できます。

- 取得プロセスに対して、hr.departments 表に対する DML 変更を取得するように指示します。
- 伝播に対して、hr.departments 表に対する DML 変更を伝播するように指示します。
- 適用プロセスに対して、DML 変更を hr.departments 表に適用するように指示します。

Streams では、タスクはルールに基づいて実行されます。これらのタスクには、取得プロセスによる変更の取得、伝播による変更の伝播および適用プロセスによる変更の適用などがあります。これらのタスクに関するルールを次の 3 つのレベルで定義できます。

- 表ルール
- スキーマ・ルール
- グローバル・ルール

表ルールの概要

表ルールを定義すると、指定した表に変更が加えられた場合に Streams のタスクが実行されます。たとえば、取得プロセスに対して、hr.employees 表に対する変更を取得するように指示するルールを定義できます。このルールの場合、hr.employees 表に 1 行が挿入されると、取得プロセスはその挿入を取得して LCR にフォーマットし、LCR をエンキューします。

スキーマ・ルールの概要

スキーマ・ルールを定義すると、指定したスキーマ内のデータベース・オブジェクトと、将来そのスキーマに追加されるデータベース・オブジェクトに変更が加えられた場合に、Streams のタスクが実行されます。たとえば、伝播に対して、hr スキーマに対する DML 変更と DDL 変更をソース・キューから宛先キューに伝播するように指示する 2 つのルールを定義できます。これらのルールの場合、ソース・キューには次の変更を定義する LCR が含まれているとします。

- hr.loc_city_ix 索引の変更
- hr.jobs 表の 1 行の更新

どちらの変更も hr スキーマ内のデータベース・オブジェクトが対象であるため、伝播はこれらの変更をソース・キューから宛先キューに伝播します。

グローバル・ルールの概要

グローバル・ルールを定義すると、データベース内のデータベース・オブジェクトに変更が加えられた場合に Streams のタスクが実行されます。グローバルな DML 取得ルールの場合、取得プロセスはデータベース内のデータベース・オブジェクトに対するすべての DML 変更を取得します。グローバルな DDL 伝播または適用ルールの場合、Streams のタスクはキュー内のすべての DDL 変更について実行されます。

注意： 取得プロセスは、特定のタイプの変更と表の列の特定のデータ型に対する変更を取得しません。また、SYS および SYSTEM スキーマ内での変更は取得しません。

関連項目：

- 第 5 章「ルール」
- 第 6 章「Streams でのルールの使用方法」

変換の概要

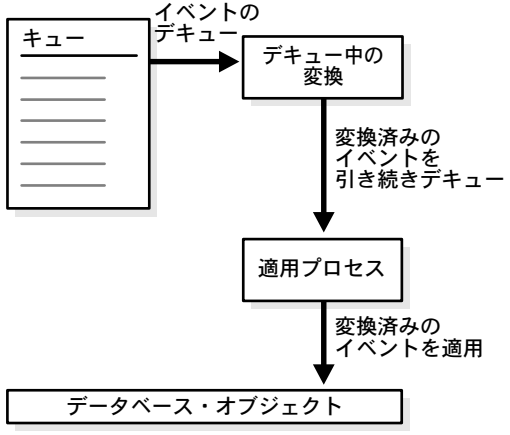
ルールベースの変換は、ルールが TRUE に評価される場合に発生するイベントの変更です。たとえば、イベントについて表内にある特定の列のデータ型を変更する必要がある場合は、ルールベースの変換を使用できます。この場合、入力として列の NUMBER データ型の論理変更レコード（LCR）を含む SYS.AnyData オブジェクトを取り、同じ列の VARCHAR2 データ型の LCR を含む SYS.AnyData オブジェクトを戻す PL/SQL ファンクションを変換に使用できます。

変換は、次の時点で発生します。

- イベントのエンキュー時。すべての接続先データベースに適切な方法でイベントをフォーマットする場合に役立ちます。
- イベントの伝播時。リモート・サイトに送信する前にデータをサブセット化する場合に役立ちます。
- イベントのデキュー時。特定の接続先データベースに適切な方法でイベントをフォーマットする場合に役立ちます。

図 1-8 に、適用時のルールベースの変換を示します。

図 1-8 適用中の変換



関連項目： 6-23 ページ「ルールベースの変換」

異機種間での情報共有の概要

Streams では、Oracle データベース間のみでなく、Oracle データベースと Oracle 以外のデータベースの間でも情報の共有がサポートされます。ここでは、このサポートの概要について説明します。

関連項目： 第 9 章「Streams 異機種間での情報の共有」

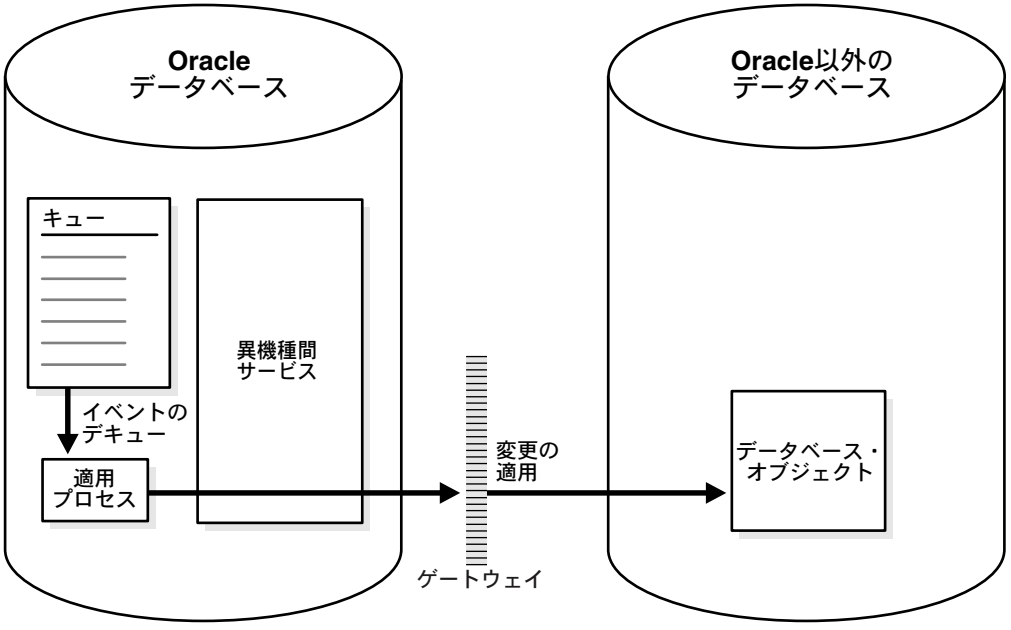
Oracle データベースから Oracle 以外のデータベースへのデータ共有の概要

ソース・データベースが Oracle で、接続先データベースが Oracle 以外の場合、Oracle 以外の接続先データベースには次の Streams メカニズムがありません。

- イベントを受信するためのキュー
- イベントをデキューして適用する適用プロセス

Oracle ソース・データベースの DML 変更を Oracle 以外の接続先データベースと共有するために、Oracle データベースはプロキシとして機能し、通常は接続先データベースで実行される手順の一部を実行します。つまり、Oracle 以外の接続先データベースを対象とするイベントは、Oracle データベース自体でデキューされ、Oracle データベースの適用プロセスは異機種間サービスを使用し、ゲートウェイを介してネットワーク接続経由でイベントを Oracle 以外のデータベースに適用します。図 1-9 に、Oracle 以外のデータベースとデータを共有している Oracle データベースを示します。

図 1-9 Oracle データベースから Oracle 以外のデータベースへの異機種間データ共有

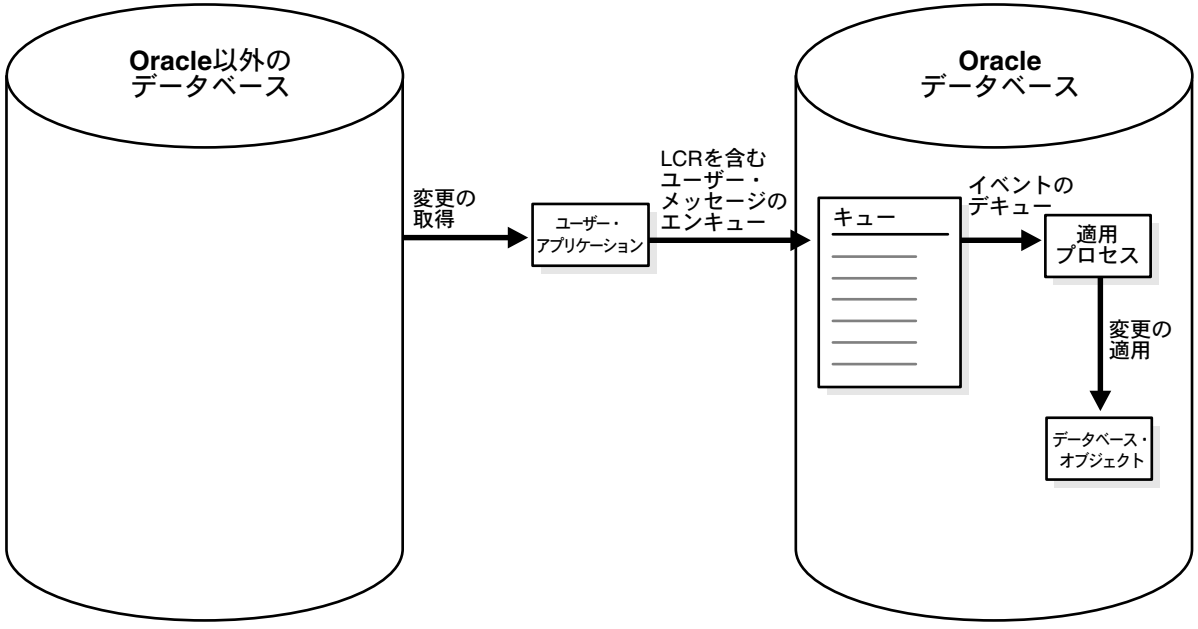


関連項目： 異機種間サービスの詳細は、『Oracle9*i* Heterogeneous Connectivity Administrator's Guide』を参照してください。

Oracle 以外のデータベースから Oracle データベースへのデータ共有の概要

Oracle 以外のデータベースから変更を取得して Oracle データベースに伝播させるには、カスタム・アプリケーションが必要です。このアプリケーションは、トランザクション・ログを読み込むか、トリガーを使用するか、または他のなんらかの方法で、Oracle 以外のデータベースに対する変更を取得します。このアプリケーションではトランザクションをアセンブルし、順序付けし、それぞれの変更を論理変更レコード（LCR）に変換する必要があります。次に、PL/SQL インタフェースを使用して、適用プロセスで処理できるように LCR を Oracle データベースのキューにエンキューします。図 1-10 に、Oracle データベースとデータを共有している Oracle 以外のデータベースを示します。

図 1-10 Oracle 以外のデータベースから Oracle データベースへの異機種間データ共有



Streams 構成の例

図 1-11 に、単一データベース内の情報を共有するように Streams を構成する方法を示します。図 1-12 には、2 つの異なるデータベース間で情報を共有するように Streams を構成する方法を示します。

図 1-11 単一データベースでの Streams 構成

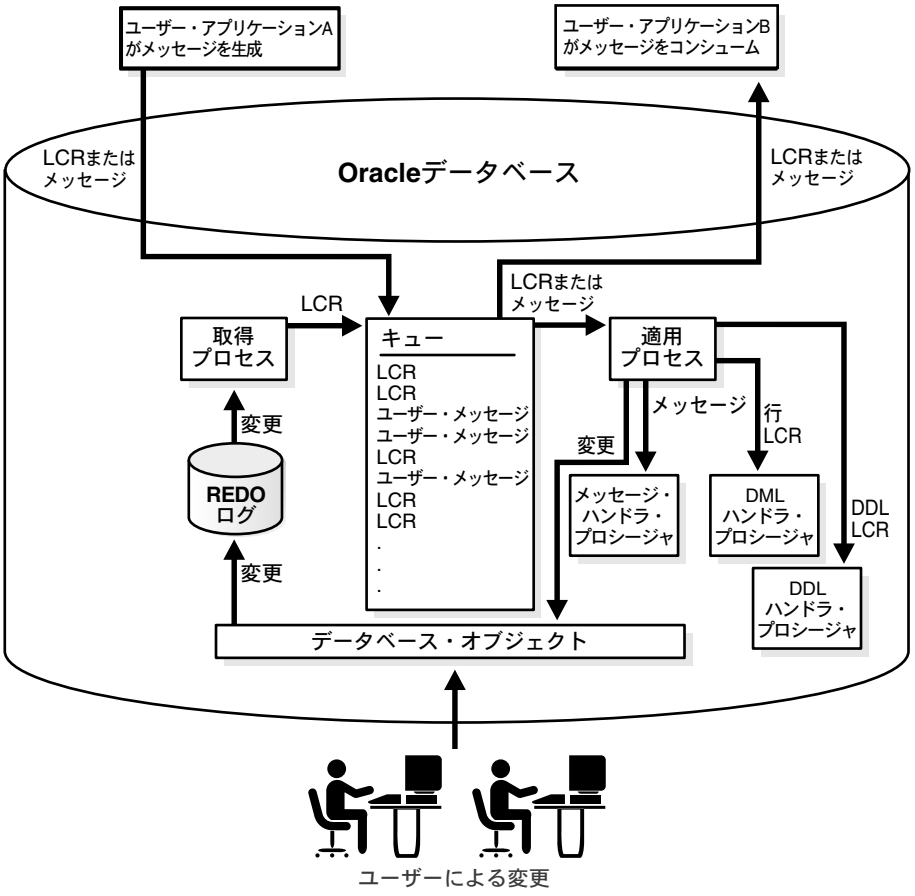
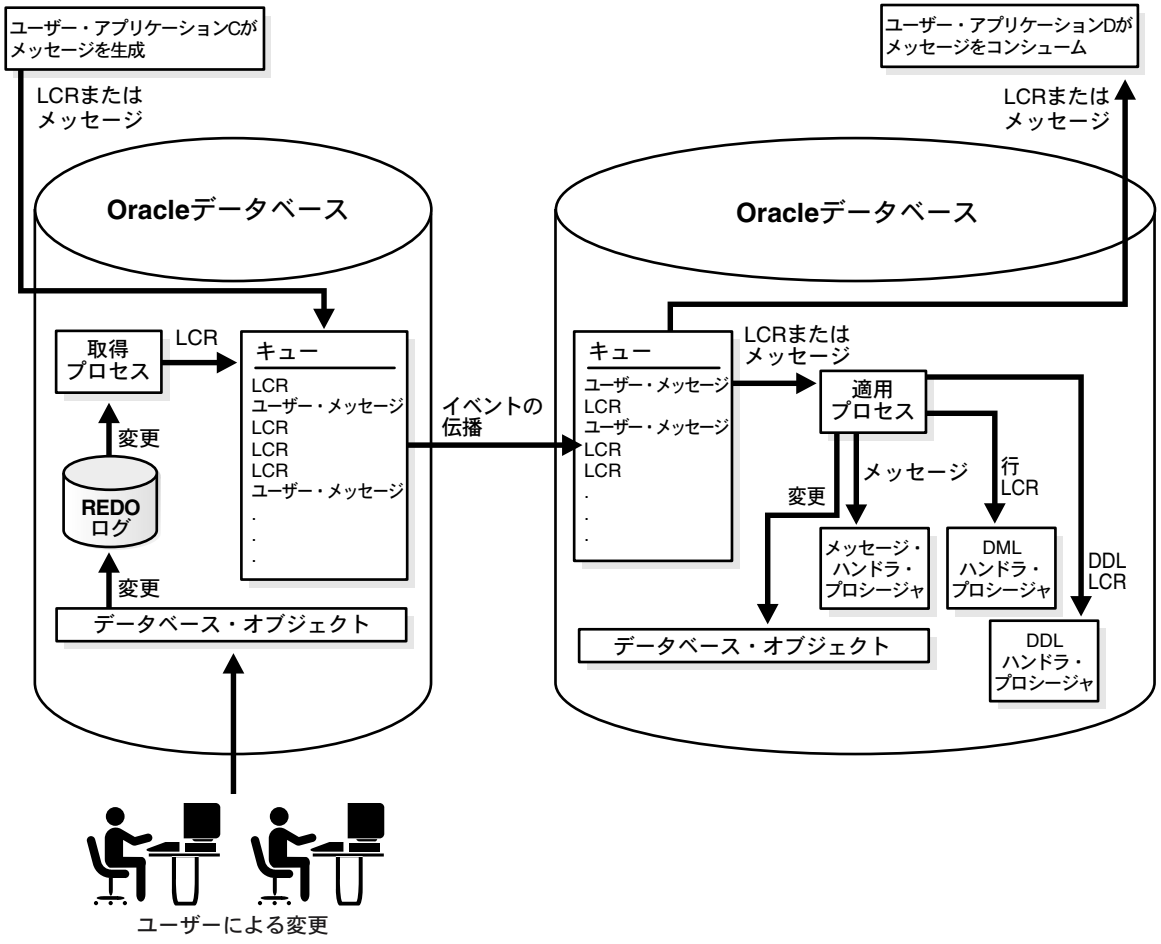


図 1-12 データベース間で情報を共有する Streams 構成



Streams 環境用の管理ツール

Streams 環境の構成、管理および監視には、複数のツールを使用できます。オラクル社が提供する PL/SQL パッケージは主要な構成および管理ツールですが、Oracle Enterprise Manager の Streams ツールにも環境を管理できるように構成、管理および監視機能が用意されています。また、Streams データ・ディクショナリ・ビューには、Streams 環境に関する情報が常に表示されます。

オラクル社が提供する PL/SQL パッケージ

オラクル社が提供する次の PL/SQL パッケージには、Streams 環境の構成と管理に使用できるプロシージャとファンクションが含まれています。

> **関連項目：** これらのパッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

DBMS_STREAMS_ADM パッケージ

DBMS_STREAMS_ADM パッケージには、表、スキーマ、データベースの各レベルでの取得、伝播および適用に関する単純なルールを追加および削除するための管理インタフェースが用意されています。また、このパッケージには、キューを作成し、データ・ディクショナリ情報など、Streams のメタデータを管理するためのプロシージャも含まれています。このパッケージは、Streams レプリケーション環境で共通のタスクを完了するための簡便な方法として提供されます。DBMS_CAPTURE_ADM、DBMS_PROPAGATION_ADM および DBMS_APPLY_ADM など、他のパッケージを使用すると、これと同じタスクのみでなく、追加のカスタマイズを必要とするタスクを完了できます。

DBMS_CAPTURE_ADM パッケージ

DBMS_CAPTURE_ADM パッケージには、取得プロセスを起動、停止および構成するための管理インタフェースが用意されています。取得される変更のソースは REDO ログで、リポジトリはキューです。また、このパッケージには、接続先データベースでインスタンス化するために、ソース・データベースでデータベース・オブジェクトを準備するための管理プロシージャも用意されています。

DBMS_PROPAGATION_ADM パッケージ

DBMS_PROPAGATION_ADM パッケージには、ソース・キューから宛先キューへの伝播を構成するための管理インタフェースが用意されています。

DBMS_APPLY_ADM パッケージ

DBMS_APPLY_ADM パッケージには、適用プロセスを起動、停止および構成するための管理インタフェースが用意されています。

DBMS_RULE_ADM パッケージ

DBMS_RULE_ADM パッケージには、ルール、ルール・セットおよびルール評価コンテキストを作成および管理するための管理インタフェースが用意されています。

DBMS_RULE パッケージ

DBMS_RULE パッケージには、ルール・セットを評価する EVALUATE プロシージャが含まれています。このプロシージャの目的は、データに基づいて条件を満たすルールのリストを生成することです。

DBMS_STREAMS パッケージ

DBMS_STREAMS パッケージには、SYS.AnyData オブジェクトを論理変更レコード（LCR）オブジェクトに変換し、Streams 属性の情報を戻し、さらにセッションで生成された REDO エントリにバイナリ・タグで注釈を付けるためのインタフェースが用意されています。このタグは、ルールに REDO エントリまたは LCR 内のバイナリ・タグの仕様が含まれている取得プロセス、伝播ジョブまたは適用プロセスの動作に影響します。

Streams のデータ・ディクショナリ・ビュー

Streams 環境内の各データベースには、Streams のデータ・ディクショナリ・ビューがあります。これらのビューでは、ローカル Streams のルール、オブジェクト、取得プロセス、伝播および適用プロセスに関する管理情報が保持されます。これらのビューを使用して、Streams 環境を監視できます。

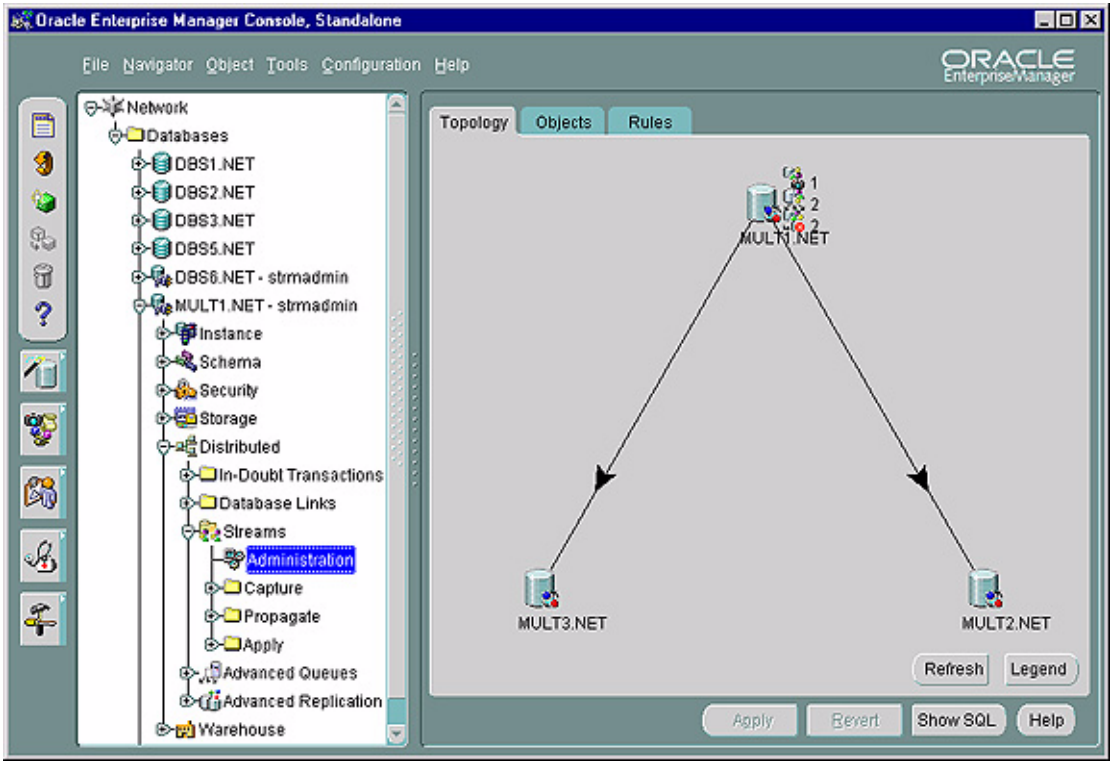
関連項目：

- 第 17 章「Streams 環境の監視」
- これらのデータ・ディクショナリ・ビューの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

Oracle Enterprise Manager の Streams ツール

Streams 環境の構成、管理および監視を容易にするために、Oracle では Oracle Enterprise Manager のコンソールに Streams ツールを用意しています。また、Streams ツールを使用すると、Streams 構成スクリプトを生成し、それを変更して実行し、Streams 環境を構成できます。Streams ツールの主な資料は、オンライン・ヘルプです。図 1-13 に、Streams ツールの「**Topology**」タブを示します。

図 1-13 Streams ツール



関連項目： Streams ツールの使用方法の詳細は、Oracle Enterprise Manager のオンライン・ヘルプを参照してください。

2

Streams の取得プロセス

この章では、Streams の取得プロセスの概念とアーキテクチャについて説明します。

この章の内容は次のとおりです。

- REDO ログと取得プロセス
- 論理変更レコード（LCR）
- 取得ルール
- 取得されるデータ型
- 取得される変更のタイプ
- Streams 環境内のサプリメンタル・ロギング
- インスタンス化
- 取得プロセスの開始 SCN、取得済 SCN および適用済 SCN
- Streams の取得プロセスと RESTRICTED SESSION
- Streams の取得プロセスと Oracle Real Application Clusters
- 取得プロセスのアーキテクチャ

関連項目： 第 12 章「取得プロセスの管理」

REDO ログと取得プロセス

各 Oracle データベースには、2 つ以上の REDO ログ・ファイルのセットがあります。データベースの REDO ログ・ファイルを総称して、データベースの REDO ログと呼びます。REDO ログの主要機能は、データベースに対して行われた変更をすべて記録することです。

REDO ログは、人為的エラーやメディア障害が発生した場合のリカバリ可能性を保証するために使用されます。取得プロセスはオプションの Oracle バックグラウンド・プロセスであり、データベースの REDO ログを読み込んで、データベース・オブジェクトに対する DML 変更と DDL 変更を取得します。取得プロセスが REDO ログから変更を取得するように構成されている場合、変更が生成されたデータベースはソース・データベースと呼ばれます。

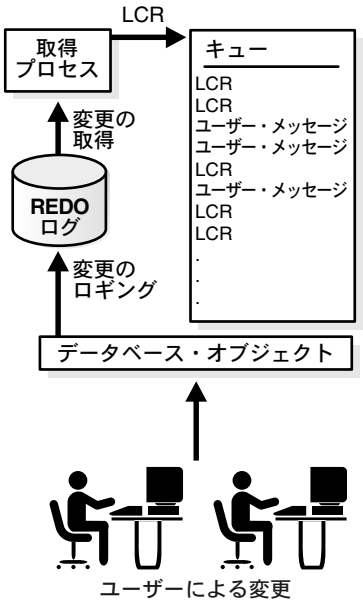
論理変更レコード（LCR）

取得プロセスは、REDO ログから取得した変更を LCR 形式に再フォーマットします。LCR は、データベース変更を記述する特定のフォーマットを持ったオブジェクトです。取得プロセスは、行 LCR および DDL LCR という 2 種類の LCR を取得します。

LCR の取得後に、取得プロセスは LCR を含むイベントをキューにエンキューします。取得プロセスには常に単一の `SYS.AnyData` キューが関連付けられており、このキューにのみイベントをエンキューします。複数のキューを作成し、各キューに異なる取得プロセスを関連付けることができます。図 2-1 に、LCR を取得する取得プロセスを示します。

注意： 取得プロセスを関連付けることができるのは `SYS.AnyData` キューのみで、型付きのキューに関連付けることはできません。

図 2-1 取得プロセス



関連項目：

- 16-2 ページ「論理変更レコード（LCR）の管理」
- LCR の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

行 LCR

行 LCR では、1 行のデータに対する変更、または 1 行の単一 LOB 列に対する変更が記述されます。この変更は、データ操作言語（DML）文または LOB のピース単位更新によるものです。たとえば、DML 文では、表に複数の行を挿入またはマージしたり、表の複数の行を更新したり、表から複数の行を削除する場合があります。そのため、単一の DML 文で複数の行 LCR が生成される可能性があります。つまり、取得プロセスでは DML 文によって変更される行ごとに LCR が作成されます。さらに、DML 文自体が、多数の DML 文を含むトランザクションの一部になっている場合があります。

取得された行 LCR には、トランザクション制御文も含まれている場合があります。これらの行 LCR には、`COMMIT` や `ROLLBACK` などのディレクティブが含まれています。これらの行 LCR は内部的であり、ソース・データベースと接続先データベースの間でトランザクションの一貫性を保つために適用プロセスによって使用されます。

それぞれの行 LCR には、次の情報が含まれます。

- 行の変更が発生したソース・データベースの名前
- 変更を生成した DML 文のタイプ（INSERT、UPDATE、DELETE、LOB ERASE、LOB WRITE または LOB TRIM）
- 行に変更があった表を含むスキーマの名前
- 変更があった行を含む表の名前
- LCR の追跡に使用できる RAW タグ
- DML 文が実行されたトランザクションの識別子
- 変更が REDO ログに書き込まれたときのシステム変更番号（SCN）
- 変更に関連する古い値。DML 文のタイプが UPDATE または DELETE の場合、これらの古い値には、DML 文の実行前に変更があった行の一部またはすべての列が含まれます。DML 文のタイプが INSERT の場合、古い値はありません。
- 変更に関連する新規の値。DML 文のタイプが UPDATE または INSERT 文の場合、これらの新規の値には、DML 文が実行された後の、変更があった行の一部またはすべての列が含まれます。DML 文のタイプが DELETE の場合、新規の値はありません。

DDL LCR

DDL LCR では、データ定義言語（DDL）の変更が記述されます。DDL 文は、データベースの構造を変更します。たとえば、DDL 文でデータベース・オブジェクトを作成、変更または削除できます。

それぞれの DDL LCR には、次の情報が含まれます。

- DDL の変更が発生したソース・データベースの名前。
- 変更を生成した DDL 文のタイプ（ALTER TABLE または CREATE INDEX など）。
- DDL 文が実行されたデータベース・オブジェクトを所有しているユーザーのスキーマ名。
- DDL 文が実行されたデータベース・オブジェクトの名前。
- DDL 文が実行されたデータベース・オブジェクトのタイプ（TABLE または PACKAGE など）。
- DDL 文のテキスト。
- ログオン・ユーザー。セッションで DDL 文が実行されたユーザーです。
- DDL テキストのオブジェクトに対するスキーマが指定されていない場合に使用されるスキーマ。

- 実表の所有者。DDL 文が表に依存する場合、実表の所有者は依存先となる表の所有者です。
- 実表の名前。DDL 文が表に依存する場合、実表の名前は依存先となる表の名前です。
- LCR の追跡に使用できる RAW タグ。
- DDL 文が実行されたトランザクションの識別子。
- 変更が REDO ログに書き込まれた時点の SCN。

注意： 行 LCR と DDL LCR の両方に、変更が発生したデータベースのソース・データベース名が含まれています。取得された LCR が伝播によって伝播されるか、適用プロセスによって適用される場合は、伝播と適用に伴う問題を回避するために、取得プロセスによる変更の取得が開始された後はソース・データベースの名前を変更しないことをお薦めします。

関連項目： DDL 文の全タイプのリストは、『Oracle Call Interface プログラマーズ・ガイド』の表「SQL コマンド・コード」を参照してください。

取得ルール

取得プロセスでは、定義したルールに基づいて変更が取得されます。各ルールでは、取得プロセスが変更を取得するデータベース・オブジェクトと、取得する変更のタイプを指定します。取得ルールは次のレベルで指定できます。

- 表ルールは、特定の表に対する DML 変更または DDL 変更を取得します。
- スキーマ・ルールは、特定のスキーマ内のデータベース・オブジェクトに対する DML 変更または DDL 変更を取得します。
- グローバル・ルールは、データベース内のすべての DML 変更または DDL 変更を取得します。

関連項目：

- 第 5 章「ルール」
- 第 6 章「Streams でのルールの使用方法」

注意： 取得プロセスは、特定のタイプの変更と表の列の特定のデータ型に対する変更を取得しません。また、`SYS` および `SYSTEM` スキーマ内での変更は取得しません。

取得されるデータ型

表に対する変更を取得するときに、取得プロセスは次のデータ型の列に対する変更を取得できます。

- CHAR
- VARCHAR2
- NCHAR
- NVARCHAR2
- NUMBER
- DATE
- CLOB
- BLOB
- RAW
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

取得プロセスで、リストされていないデータ型の列を含む表に、ルールを満たす変更が検出されると、エラーが発生します。取得プロセスは、データ型 NCLOB、LONG、LONG RAW、BFILE、ROWID、UROWID およびユーザー定義型（オブジェクト型、REF、VARRAY および NESTED TABLE など）の列における DML 変更は取得しません。取得プロセスにエラーが発生すると、エラーの原因となった LCR がトレース・ファイルに書き込まれ、ORA-00902 エラーが発生し、取得プロセスが無効化されます。

関連項目：

- 4-9 ページ「適用されるデータ型」
- これらのデータ型の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

取得される変更のタイプ

取得プロセスは、データベースとそのオブジェクトに対して行われた特定タイプの変更のみを取得できます。次の項では、取得できる DML 変更と DDL 変更のタイプを説明します。取得プロセスでは、取得できない変更は無視されます。

注意： 取得プロセスは、SYS および SYSTEM スキーマ内での変更は取得しません。

関連項目： 適用プロセスによって適用される変更と無視される変更のタイプについては、第 4 章「Streams 適用プロセス」を参照してください。

取得される DML 変更のタイプ

特定の表に対する DML 変更を取得するように指定すると、取得プロセスはそれらの表に対する次のタイプの DML 変更を取得します。

- INSERT
- UPDATE
- DELETE
- MERGE
- LOB のピース単位更新

注意：

- 取得プロセスは、各 MERGE 変更を INSERT 変更または UPDATE 変更に変換します。MERGE は、行 LCR では有効なコマンド・タイプではありません。
- 取得プロセスは、CALL、EXPLAIN PLAN または LOCK TABLE 文を取得しません。
- 取得プロセスでは、一時表、索引構成表またはオブジェクト表に対する DML 変更は取得できません。
- 順序を複数のデータベースで共有する場合、これらのデータベースで個々の行に使用される順序値が異なっている可能性があります。また、実際の順序値に対する変更は取得されません。たとえば、ユーザーが NEXTVAL を参照するか、順序を設定する場合、取得プロセスではこれらの操作によって生じる変更は取得されません。

関連項目：

- 4-10 ページ「表に DML 変更を適用する際の考慮事項」
- 様々なデータベースで 2 つの異なる行の順序生成値が同じになるのを回避する方針については、7-6 ページの「Streams 環境内の一意性競合の防止」を参照してください。

取得プロセスによって無視される DDL 変更のタイプ

取得プロセスでは、次のタイプの DDL 変更を除き、取得プロセスのルール・セット内にあるルールを満たす DDL 変更が取得されます。

- ALTER DATABASE
- CREATE CONTROLFILE
- CREATE DATABASE
- CREATE PFILE
- CREATE SPFILE

取得プロセスでは、取得プロセスのルール・セット内にあるルールを満たす DDL 文が取得されますが、DDL 文が CREATE TABLE AS SELECT 文である場合を除き、DDL 文の結果は取得されません。たとえば、取得プロセスで ANALYZE 文が取得されても、ANALYZE 文により生成された統計は取得されません。ただし、取得プロセスで CREATE TABLE AS SELECT 文が取得される場合は、この文自体と選択されたすべての行（INSERT 行 LCR）が取得されます。

取得プロセスによって取得される一部のタイプの DDL 変更は、適用プロセスで適用できません。適用プロセスは、適用できない操作を指定する DDL LCR を受信すると、DDL LCR を無視して、それに関する情報を適用プロセスのトレース・ファイルに記録します。

関連項目： 4-20 ページ「DDL 変更の適用に関する考慮事項」

取得プロセスによって無視されるその他のタイプの変更

次のタイプの変更は、取得プロセスによって無視されます。

- セッション制御文 ALTER SESSION および SET ROLE
- システム制御文 ALTER SYSTEM
- PL/SQL プロシージャの起動

また、取得プロセスが変更を取得する表またはスキーマでは、DBMS_REDEFINITION パッケージを使用したオンラインの表の再定義はサポートされません。

SQL 操作用の NOLOGGING および UNRECOVERABLE キーワード

SQL 操作に NOLOGGING または UNRECOVERABLE キーワードを使用した場合、SQL 操作によって生じる変更は取得プロセスで取得できません。したがって、SQL 操作により生じる変更を取得プロセスで取得する必要がある場合、これらのキーワードは使用しないでください。

ロギング属性を指定しているオブジェクトが FORCE LOGGING モードのデータベースまたは表領域に存在する場合、このデータベースまたは表領域が FORCE LOGGING モードでなくなるまで、Oracle では NOLOGGING または UNRECOVERABLE 設定は無視されます。データベースの現行ロギング・モードを判別するには、V$DATABASE 動的パフォーマンス・ビューの FORCE_LOGGING 列を問い合せます。

注意： UNRECOVERABLE キーワードは推奨できないため、*logging_clause* では NOLOGGING キーワードに置き換えられています。UNRECOVERABLE は下位互換性のためサポートされていますが、必要に応じて NOLOGGING キーワードを使用することをお薦めします。

関連項目： NOLOGGING および UNRECOVERABLE キーワード、FORCE LOGGING モード、および *logging_clause* の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

ダイレクト・パス・ロードの UNRECOVERABLE 句

ダイレクト・パス・ロードで SQL*Loader 制御ファイルに UNRECOVERABLE 句を使用した場合、ダイレクト・パス・ロードにより生じる変更は取得プロセスで取得できません。したがって、ダイレクト・パス・ロードにより生じる変更を取得プロセスで取得する必要がある場合、UNRECOVERABLE 句は使用しないでください。

ソース・データベースで変更をログに記録せずにダイレクト・パス・ロードを実行し、ソース・データベースの接続先データベースで同様のダイレクト・パス・ロードを実行しない場合、ソース・データベースのロード済オブジェクトに変更が加えられると、これらの接続先データベースで適用エラーが発生する可能性があります。この場合、ソース・データベースでの取得プロセスはこれらのオブジェクトに対する変更を取得し、1 つ以上の伝播で接続先データベースに対する変更を伝播できますが、これらのオブジェクトが接続先データベースに存在しないか、またはオブジェクトが接続先データベースに存在しても、これらの変更に関連する行が存在しない可能性があります。

したがって、ダイレクト・パス・ロードで UNRECOVERABLE 句を使用し、取得プロセスがロード済オブジェクトに対する変更を取得するように構成されている場合、適用エラーを回避するために、接続先データベースにロード済オブジェクトおよびロード済データが含まれることを確認してください。これらのオブジェクトが接続先データベースで確実に存在するようにするには、ソース・データベースで実行したダイレクト・パス・ロードと同様のダイレクト・パス・ロードを、これらの各接続先データベースで実行する方法があります。

FORCE LOGGING モードのデータベースまたは表領域にオブジェクトをロードする場合、Oracle ではダイレクト・パス・ロード時に UNRECOVERABLE 句が無視され、ロード済変更がログに記録されます。データベースの現行ロギング・モードを判別するには、V$DATABASE 動的パフォーマンス・ビューの FORCE_LOGGING 列を問い合せます。

関連項目： ダイレクト・パス・ロードおよび SQL*Loader の詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。

Streams 環境内のサプリメンタル・ロギング

サプリメンタル・ロギングでは、UPDATE 操作が実行されるたびに REDO ログに列データが追加されます。この種の更新には、LOB のピース単位更新が含まれます。取得プロセスは、この追加情報を取得して LCR に含めます。

サプリメンタル・ロギングには、データベース・サプリメンタル・ロギングと表サプリメンタル・ロギングの 2 種類があります。データベース・サプリメンタル・ロギングではデータベース全体のサプリメンタル・ロギングが指定されますが、表サプリメンタル・ロギングでは、特定の表のサプリメンタル・ロギングに使用するログ・グループを指定できます。表サプリメンタル・ロギングを使用する場合は、無条件ログ・グループと条件付きログ・グループのどちらかを選択できます。

無条件ログ・グループでは、表が更新されるたびに、指定した列が更新の影響を受けるかどうかに関係なく、その列のビフォア・イメージがログに記録されます。このログ・グループは、ALWAYS ログ・グループと呼ばれることがあります。条件付きログ・グループでは、ログ・グループ内の少なくとも 1 列が更新される場合にのみ、指定したすべての列のビフォア・イメージがログに記録されます。

データベース・レベルのサプリメンタル・ロギング、表レベルの無条件ログ・グループおよび表レベルの条件付きログ・グループにより、更新文またはピース単位の LOB 更新でログに記録する古い値が決定されます。

取得プロセスによって取得された LCR を、1 つ以上の適用プロセスを使用して適用する予定の場合は、接続先データベースの表内にある次のタイプの列について、ソース・データベースでサプリメンタル・ロギングを使用可能にする必要があります。

- 接続先データベースで変更が適用される表の主キーに使用されるソース・データベースの列はすべて、ログ・グループに無条件でログを記録するか、主キー列のデータベース・サプリメンタル・ロギングによってログを記録する必要があります。
- 変更を適用する適用プロセスの並列性が 2 以上の場合、ソース・データベースの複数列に基づく接続先データベースの一意制約は、条件付きでログに記録する必要があります。 一意制約がソース・データベースの単一列に基づく場合、サプリメンタル・ロギングを指定する必要はありません。
- 変更を適用する適用プロセスの並列性が 2 以上の場合、ソース・データベースの複数列に基づく接続先データベースの外部キー制約は、条件付きでログに記録する必要があります。 外部キーがソース・データベースの単一列に基づく場合、サプリメンタル・ロギングを指定する必要はありません。

- 接続先データベースで適用プロセスの代替キー列に使用されるソース・データベースの列は、無条件でログに記録する必要があります。表の代替キー列を指定するには、DBMS_APPLY_ADM パッケージの SET_KEY_COLUMNS プロシージャを使用します。
- ソース・データベースの複数の列が接続先データベースの列リストに使用されている場合、適用時に競合解消用の列リストで指定された列は、条件付きでログに記録する必要があります。
- 接続先データベースでの更新操作または LOB に対するピース単位の更新用に指定された、DML ハンドラまたはエラー・ハンドラで使用されるソース・データベースの列は、無条件でログに記録する必要があります。
- ルールまたはルールベースの変換で使用されるソース・データベースの列は、無条件でログに記録する必要があります。
- 接続先データベースで表の行のサブセット化を指定する場合は、宛先表に含まれるソース・データベースの列またはサブセット条件に含まれるソース・データベースの列を、無条件でログに記録する必要があります。適用プロセスの行のサブセット化条件を指定するには、DBMS_STREAMS_ADM パッケージの ADD_SUBSET_RULES プロシージャで dml_condition パラメータを使用します。

ソース・データベースでこれらのタイプの列にサプリメンタル・ロギングを使用しないと、これらの列が関係する変更は接続先データベースで正しく適用されない可能性があります。

注意： LOB、LONG およびユーザー定義型は、サプリメンタル・ログ・グループに含めることはできません。

関連項目：

- 12-8 ページ「ソース・データベースでのサプリメンタル・ロギングの指定」
- サプリメンタル・ロギングの使用方法については、『Oracle9*i* Data Guard 概要および管理』および『Oracle9*i* データベース管理者ガイド』を参照してください。
- 表に対する適用プロセスの動作の詳細は、4-10 ページの「表に DML 変更を適用する際の考慮事項」を参照してください。
- サプリメンタル・ロギングおよびキー列の詳細は、4-10 ページの「制約」を参照してください。
- 6-23 ページ「ルールベースの変換」
- サプリメンタル・ロギングおよび列リストの詳細は、7-11 ページの「列リスト」を参照してください。

インスタンス化

取得プロセスを使用して、データベース・オブジェクトに対する変更をソース・データベースから接続先データベースにレプリケートする予定の場合は、接続先データベースにデータベース・オブジェクトのコピーが含まれる必要があります。接続先データベースにコピーが存在しない場合は、変更をレプリケートする前に、接続先データベースでオブジェクトを**インスタンス化**する必要があります。オブジェクトのインスタンス化とは、ソース・データベースのオブジェクトに基づいて、接続先データベースでオブジェクトを物理的に作成することを意味します。オブジェクトが表である場合、ソース・データベースと接続先データベースのオブジェクトは完全一致でなくても構いませんが、2 つのデータベース間で表データの一部またはすべてをレプリケートする場合は、表をインスタンス化したときにレプリケートするデータに一貫性がある必要があります。通常、インスタンス化はエクスポート / インポートを使用して行われます。

単一のデータベース内または複数のデータベース間で 1 つのデータベース・オブジェクトを共有する Streams 環境では、ソース・データベースとはオブジェクトに対する変更が REDO ログ内で生成されるデータベースです。このような変更が取得プロセスで取得されているか、今後取得され、その変更がローカルに適用されるか、他のデータベースに伝播されてから接続先データベースで適用される場合は、接続先データベースでソース・データベース・オブジェクトのインスタンス化が必要になることがあります。どのイベントについても、常にオブジェクトをインスタンス化のために準備する必要があります。インスタンス化のためにオブジェクトを準備することで、オブジェクトに対する変更を接続先データベースで適用する必要がある最も古い SCN が設定されます。

DBMS_CAPTURE_ADM パッケージの次のプロシージャでは、インスタンス化のためにデータベース・オブジェクトの準備が行われます。

- PREPARE_TABLE_INSTANTIATION では、1 つの表がインスタンス化のために準備されます。
- PREPARE_SCHEMA_INSTANTIATION では、スキーマ内の全データベース・オブジェクトと、将来そのスキーマに追加される全データベース・オブジェクトが、インスタンス化のために準備されます。
- PREPARE_GLOBAL_INSTANTIATION では、データベース内の全オブジェクトと、将来そのデータベースに追加される全オブジェクトが、インスタンス化のために準備されます。

これらのプロシージャでは、インスタンス化のために各オブジェクトの最小 SCN が記録されます。オブジェクトの最小 SCN より後の SCN は、そのオブジェクトのインスタンス化に使用できます。また、前述のプロシージャでは、インスタンス化の準備対象となる、表、スキーマまたはデータベースに対する変更を、取得、伝播または適用する、取得プロセス、伝播および適用プロセスのために、Streams データ・ディクショナリが移入されます。

オブジェクトの取得ルールまたは伝播ルールの条件を追加または変更するときは、次のいずれかの条件が満たされる場合、常に適切なプロシージャを実行して、ソース・データベースでそのオブジェクトをインスタンス化のために準備する必要があります。

- オブジェクトに対する変更を取得するように指示する取得プロセスのルール・セットに、1 つ以上のルールを追加する場合。DBMS_STREAMS_ADM パッケージを使用して取得プロセスのルール・セットにルールを追加すると、インスタンス化の準備を行う適切なプロシージャがソース・データベースで自動的に実行されます。DBMS_RULE_ADM パッケージを使用してこれらのルールを追加する場合は、インスタンス化のために手動で準備する必要があります。
- オブジェクトに対する変更を取得するように指示する取得プロセスのルール・セットで、ルールについて 1 つ以上の条件を変更する場合。
- オブジェクトに対する変更を伝播するように指示する伝播のルール・セットに、1 つ以上のルールを追加する場合。
- オブジェクトに対する変更を伝播するように指示する伝播のルール・セットで、ルールについて 1 つ以上の条件を変更する場合。

前述の条件が 1 つでも満たされる場合は、ルールを追加または変更したリモート・データベースにオブジェクトが存在していても、ソース・データベースでインスタンス化のためにこれらのデータベース・オブジェクトの準備を行って、ソース・オブジェクトに関する情報が必要な関連する Streams データ・ディクショナリを移入する必要があります。

関連する Streams データ・ディクショナリは、ローカル・ディクショナリとすべてのリモート・ディクショナリについて非同期に移入されます。インスタンス化の準備を行うプロシージャでは、ソース・データベースの REDO ログに情報が追加されます。ローカルの Streams データ・ディクショナリには、取得プロセスがこれらの REDO エントリを取得するとオブジェクトに関する情報が移入され、リモートの Streams データ・ディクショナリには情報が伝播されると移入されます。

エクスポート / インポートを使用して表をインスタンス化した場合、インスタンス化された表について表サプリメンタル・ログ・グループの仕様が保持されます。つまり、インスタンス化の後、インポート・データベースのインポートされた表のログ・グループ仕様は、エクスポート・データベースの同じ表のログ・グループの仕様と同じです。インポート・データベースで表のサプリメンタル・ログ・グループの仕様を保持しない場合は、インポート後に特定のサプリメンタル・ログ・グループを削除できます。全データベースのエクスポート / インポートを実行する場合でも、エクスポート / インポート中はデータベース・サプリメンタル・ロギングの仕様は保持されません。

関連項目：

- 12-10 ページ「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」
- 表およびデータベース・サプリメンタル・ロギングの詳細は、2-10 ページの「Streams 環境内のサプリメンタル・ロギング」を参照してください。
- サプリメンタル・ログ・グループを追加および削除する方法の詳細は、12-8 ページの「ソース・データベースでのサプリメンタル・ロギングの指定」を参照してください。

取得プロセスの開始 SCN、取得済 SCN および適用済 SCN

この項では、取得プロセスで重要となるシステム変更番号（SCN）値を説明します。1 つ以上の取得プロセスのシステム変更番号を表示するには、DBA_CAPTURE データ・ディクショナリ・ビューを問い合せます。

開始 SCN

開始 SCN は、取得プロセスが変更の取得を開始する SCN です。取得プロセスを初めて起動すると、デフォルトで開始 SCN は取得プロセスが作成されたときの SCN となります。たとえば、取得プロセスを作成から 2 日後に起動した場合、REDO ログからの変更の取得は 2 日前の作成時点から開始されます。

取得プロセスの作成時に異なる開始 SCN を指定したり、取得プロセスを変更して開始 SCN を設定できます。開始 SCN には、データベースの最初の取得プロセスを作成した時点以後の SCN を指定する必要があります。

関連項目：

- 12-10 ページ「取得プロセスの開始 SCN の設定」
- 取得プロセスを変更する方法については、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_CAPTURE_ADM.ALTER_CAPTURE プロシージャの項を参照してください。

取得済 SCN

取得済 SCN は、取得プロセスで取得された最新の変更に対応する SCN です。

適用済 SCN

取得プロセスの**適用済 SCN** は、関連する適用プロセスによりデキューされた最新イベントの SCN です。この SCN より小さい番号のすべてのイベントは、この取得プロセスで取得された変更を適用するすべての適用プロセスによりデキューされています。

Streams の取得プロセスと RESTRICTED SESSION

システム起動時に STARTUP RESTRICT 文を発行して制限付きセッションを有効化した場合、データベースの停止時に取得プロセスが実行中であった場合も、取得プロセスは起動しません。制限付きセッションを無効化すると、データベースの停止時に実行中であった各取得プロセスが起動されます。

SQL 文 ALTER SYSTEM および ENABLE RESTRICTED SESSION 句により実行中のデータベースで制限付きセッションが有効化された場合、実行中の取得プロセスには影響しません。これらの取得プロセスは引き続き実行され、変更の取得が行われます。制限付きセッションで停止済の取得プロセスが起動された場合、この取得プロセスは制限付きセッションが無効化されるまで起動されません。

Streams の取得プロセスと Oracle Real Application Clusters

Streams の取得プロセスは、Real Application Clusters 環境での変更を取得するように構成できます。1 つ以上の取得プロセスと Real Application Clusters を同じ環境で使用する場合、その環境は次の要件を満たす必要があります。

- 取得プロセスによって取得される変更を含むすべてのアーカイブ・ログは、Real Application Clusters 環境の全インスタンスで使用可能であることが必要です。Real Application Clusters 環境においては、取得プロセスで常にアーカイブ REDO ログが読み込まれ、すべてのインスタンスによる変更が読み込まれます。
- DBMS_CAPTURE_ADM.START_CAPTURE プロシージャのコールは、取得プロセスで使用されるキューの所有インスタンス上で実行する必要があります。取得プロセスを操作する他のプロシージャとファンクションのコールは、どのインスタンスからでも実行できます。
- サプリメンタル・ロギングは、実行中の各インスタンスで指定する必要があります。実行中のインスタンスごとに指定した後は、インスタンスを停止して再起動する場合に再指定する必要はなく、新規インスタンスについても指定する必要はありません。
- ARCHIVE_LAG_TARGET 初期化パラメータを、0（ゼロ）より大きい値に設定する必要があります。この初期化パラメータでは、ログ・ファイルが自動的に切り替えられるまでの期間を指定します。LogMiner では、すべての LCR が SCN で順序付けられます。そのためには、すべてのインスタンスからのアーカイブ・ログ・ファイルが必要です。このパラメータをログ・ファイルが自動的に切り替わるように設定すると、LogMiner は、あるインスタンスのトランザクションが他のインスタンスよりはるかに少ない場合に、必要以上に長時間待機しなくなります。

取得プロセスで使用されるキューを含むキュー表の所有者インスタンスが使用不可能になると、キューの所有権は自動的にクラスタ内の別のインスタンスに移ります。この状況が発生した場合は、取得プロセスを再起動するために、キューの所有者インスタンスに接続し、START_CAPTURE プロシージャを実行してください。DBA_QUEUE_TABLES データ・ディクショナリ・ビューは、キュー表の所有者インスタンスに関する情報を示します。取得プロセスが Real Application Clusters 環境で永続的な起動 / 停止状態を維持するのは、キューを所

有するデータベース・インスタンスが再起動される前に、キューの所有者インスタンスに変更がない場合のみです。

また、単一の取得プロセスで使用されるパラレル実行プロセスは、Real Application Clusters 環境では単一のインスタンスで実行されます。

関連項目：

- 3-17 ページ「Streams キューと Oracle Real Application Clusters」
- 4-27 ページ「Streams の適用プロセスと Oracle Real Application Clusters」
- ARCHIVE_LAG_TARGET 初期化パラメータの詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。
- DBA_QUEUE_TABLES データ・ディクショナリ・ビューの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。
- 2-27 ページ「取得プロセスの永続的状態」
- 2-10 ページ「Streams 環境内のサプリメンタル・ロギング」

取得プロセスのアーキテクチャ

取得プロセスは Oracle バックグラウンド・プロセスであり、そのプロセス名は cp*nn* です。この場合、*nn* は取得プロセス番号です。有効な取得プロセス名は、cp01 ～ cp99 です。取得プロセスは、LogMiner のインフラストラクチャを使用して REDO ログから変更を取得します。LogMiner は、Streams によって自動的に構成されます。取得プロセスの作成、変更、起動、停止および削除と、取得プロセスで取得される変更を制御する取得ルールの定義を行うことができます。

取得プロセスを作成するユーザーは、取得ルール評価および取得ルールベースの変換を実行するユーザーです。さらにこのユーザーは、取得プロセスで使用されるキューに取得済イベントをエンキューします。このユーザーは、取得プロセスで使用されるルール・セットの実行権限、ルール・セットで使用されるすべての変換ファンクションの実行権限、および取得プロセス・キューへのイベントのエンキュー権限など、前述のアクションの実行に必要な権限を持っている必要があります。

関連項目： 必要な権限については、11-2 ページの「Streams 管理者の構成」を参照してください。

この項の内容は、次のとおりです。

取得プロセスのコンポーネント

取得プロセスのコンポーネントは、parallelism 取得プロセス・パラメータに指定する設定に応じて異なります。parallelism を 3 以上の値に設定すると、取得プロセスは次のパラレル実行サーバーを使用して変更を同時に取得します。

- REDO ログを読み込んで変更を検索する 1 つの**リーダー・サーバー**。
- リーダー・サーバーによって検出された変更を LCR 形式にフォーマットする多数の**プリペアラ・サーバー**。プリペアラ・サーバーの数は、parallelism 取得プロセス・パラメータに指定した数から 2 を差し引いた数になります。
- プリペアラ・サーバーによって作成された LCR をマージして SCN の順序を保つ 1 つの**ビルダー・サーバー**。LCR のマージ後に、ビルダー・サーバーはそれを取得プロセスに関連付けられたキューにエンキューします。

たとえば、parallelism を 5 に設定すると、取得プロセスは合計 5 つのパラレル実行サーバーを使用します。合計 5 つのパラレル実行サーバーが使用可能な場合、リーダー・サーバーが 1 つ、プリペアラ・サーバーが 3 つおよびビルダー・サーバーが 1 つになります。

parallelism を 2 以下に設定すると、取得プロセス自体（*cpnn*）がパラレル実行サーバーを使用せずにすべての作業を実行します。

関連項目：

- parallelism パラメータの詳細は、2-23 ページの「取得プロセスの並列性」を参照してください。
- パラレル実行サーバーの管理については、『Oracle9*i* データベース管理者ガイド』を参照してください。

LogMiner の構成

取得プロセスは、LogMiner を使用して REDO ログに記録された変更を取得します。この項では、1 つ以上の取得プロセスで使用できるように LogMiner を構成する方法について説明します。

LogMiner の表のための代替表領域

LogMiner の表には、LogMiner で使用されるデータ・ディクショナリ表と一時表があります。デフォルトでは、LogMiner の表はすべて SYSTEM 表領域を使用するように作成されますが、SYSTEM 表領域に領域が足りないために LogMiner の表が収まらない場合があります。したがって、データベースで取得プロセスを作成する前に、LogMiner の表に使用する代替表領域を作成することをお薦めします。LogMiner の表をすべて代替表領域に再作成するには、DBMS_LOGMNR_D.SET_TABLESPACE ルーチンを使用します。

関連項目：

- 11-13 ページ「LogMiner 用の代替表領域の指定」
- LogMiner の表に代替表領域を使用する方法の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。
- SET_TABLESPACE プロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

単一データベース内の複数の取得プロセス

各取得プロセスは 1 つの LogMiner セッションを使用し、インスタンスに許容されるアクティブな LogMiner セッションの最大数は LOGMNR_MAX_PERSISTENT_SESSIONS 初期化パラメータで制御されます。この初期化パラメータのデフォルト設定は 1 です。したがって、データベース内で複数の取得プロセスを使用するには、LOGMNR_MAX_PERSISTENT_SESSIONS 初期化パラメータを取得プロセス数より大きい値に設定します。

また、単一データベース上で複数の取得プロセスを実行する場合は、各インスタンスのシステム・グローバル領域（SGA）のサイズを増やす操作が必要になることがあります。SGA のサイズを増やすには、SGA_MAX_SIZE 初期化パラメータを使用します。また、データベース上の取得プロセスごとに、共有プールのサイズを 10MB ずつ増やす必要があります。

注意： 様々な取得プロセスからの LCR を別個に保存するために、各取得プロセスでは個別のキューを使用することをお薦めします。

関連項目： LOGMNR_MAX_PERSISTENT_SESSIONS 初期化パラメータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

取得プロセスの作成

取得プロセスを作成するには、DBMS_STREAMS_ADM パッケージを使用する方法と、DBMS_CAPTURE_ADM パッケージを使用する方法があります。一部の構成オプションにはデフォルトが自動的に使用されるため、DBMS_STREAMS_ADM パッケージを使用して取得プロセスを作成する方が簡単です。また、DBMS_STREAMS_ADM パッケージを使用すると、取得プロセスのルール・セットが作成され、そのルール・セットにルールが自動的に追加されます。DBMS_STREAMS_ADM パッケージは、レプリケーション環境で使用するように設計されています。それに対して DBMS_CAPTURE_ADM パッケージを使用して取得プロセスを作成するのは、より柔軟性があり、ルール・セットとルールの作成は取得プロセスの作成前または後に行われます。DBMS_STREAMS_ADM パッケージまたは DBMS_RULE_ADM パッケージのプロシージャを使用すると、取得プロセスのルール・セットにルールを追加できます。

DBMS_STREAMS_ADM パッケージのプロシージャを使用して取得プロセスを作成すると、取得プロセスで変更が取得される表に対して、DBMS_CAPTURE_ADM パッケージのプロシージャが自動的に実行されます。次の表に、DBMS_STREAMS_ADM パッケージのプロシージャを実行した場合に DBMS_CAPTURE_ADM パッケージ内で実行されるプロシージャを示します。

DBMS_STREAMS_ADM パッケージ内で実行するプロシージャ	自動的に実行される DBMS_CAPTURE_ADM パッケージ内のプロシージャ
ADD_TABLE_RULES	PREPARE_TABLE_INSTANTIATION
ADD_SCHEMA_RULES	PREPARE_SCHEMA_INSTANTIATION
ADD_GLOBAL_RULES	PREPARE_GLOBAL_INSTANTIATION

インスタンス化を行うために複数回実行されることがあります。リモート・サイトで表、スキーマまたはデータベースをインスタンス化する予定の場合は、DBMS_CAPTURE_ADM パッケージの CREATE_CAPTURE プロシージャで取得プロセスを作成するときに、適切なプロシージャを手動で実行して、変更が取得される各表、スキーマまたはデータベースをインスタンス化のために準備する必要があります。

注意： データベースで取得プロセスを作成した後は、データベースの DBID または DBNAME を変更しないでください。

関連項目： 取得プロセスの作成に使用できる次のプロシージャの詳細は、第 12 章「取得プロセスの管理」および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

- `DBMS_STREAMS_ADM.ADD_TABLE_RULES`
- `DBMS_STREAMS_ADM.ADD_SCHEMA_RULES`
- `DBMS_STREAMS_ADM.ADD_GLOBAL_RULES`
- `DBMS_CAPTURE_ADM.CREATE_CAPTURE`

取得プロセス作成中のデータ・ディクショナリの複製

データベース用の最初の取得プロセスが作成されると、Streams は、取得プロセスと伝播で使用できるように、Streams データ・ディクショナリと呼ばれる複製データ・ディクショナリを移入します。Streams データ・ディクショナリは、取得プロセスの作成時点では、1 次データ・ディクショナリと一貫性のある状態になっています。

1 次データ・ディクショナリ内の情報は REDO ログから取得される変更に適用できないため、取得プロセスには Streams データ・ディクショナリが必要です。このような変更は、取得プロセスによって取得される数分前または数時間前に発生した可能性があります。たとえば、次の使用例を考えます。

1. 取得プロセスは、表に対する変更を取得するように構成されています。
2. データベース管理者が取得プロセスを停止します。取得プロセスが停止されると、その時点で取得中だった変更の SCN が記録されます。
3. ユーザー・アプリケーションは、取得プロセスの停止中も引き続き表に対する変更を行います。
4. 取得プロセスが、停止から 3 時間後に再起動されます。

この場合、データ整合性を確保するために、取得プロセスは停止された時点から REDO ログ内の変更の取得を開始する必要があります。取得プロセスは、停止時に記録した SCN から開始します。

REDO ログには RAW データが含まれており、データベース・オブジェクト名や表の列名は含まれていません。かわりに、データベース・オブジェクトと列には、それぞれオブジェクト番号と内部列番号が使用されます。したがって、変更を取得する場合、取得プロセスはデータ・ディクショナリを参照して変更の詳細を判別する必要があります。

Streams データ・ディクショナリは、取得プロセスによって DDL 文が処理されるときに必要に応じて更新されます。取得プロセスが変更を取得してから現時点までの間に関連する表に対する DDL 変更があった場合、1 次データ・ディクショナリには取得された変更に関して正しい情報が含まれていない可能性があります。ただし、Streams データ・ディクショナリでは、1 次データ・ディクショナリ内の情報のサブセットがバージョニングされるため、取得された変更について常に正しい時刻が反映されます。

取得プロセスは表に関係する DDL 変更を取得するかどうかを判断するときに、変更に関する情報を Streams データ・ディクショナリに自動的に追加します。また、表の新バージョンについて、Streams データ・ディクショナリ情報を取得するかどうかを判断します。これらの判断を行うために、DDL 文によって作成または変更された表の名前と所有者を含む部分的な情報を使用して、取得ルール・セットが評価されます。Streams データ・ディクショナリ情報を取得して伝播することで、各宛先キューで使用可能になり、伝播と適用プロセスで使用できるようになります。

取得ルール・セットで少なくとも 1 つのルールが TRUE に評価されるか（true_rules）、さらに情報を与えることによって TRUE に評価される可能性がある場合（maybe_rules）は、その表に関する Streams データ・ディクショナリ情報が取得されます。このルールは、DML ルールでも DDL ルールでもかまいません。ソース・データベースの取得プロセスでは、表がインスタンス化のために準備されるときに、同様のルール評価が実行されます。

データ・ディクショナリは最初の取得プロセスの作成時に複製されるため、データベースの最初の取得プロセスを作成するには多少時間がかかることがあります。所要時間は、データベース内のデータベース・オブジェクトの数に応じて異なります。

データ・ディクショナリは、データベース用に一度のみ複製されます。追加の取得プロセスは、データベース内で最初に作成された取得プロセスと同じ Streams データ・ディクショナリを使用します。Streams データ・ディクショナリはマルチバージョニングされるため、各取得プロセスは Streams データ・ディクショナリと同期状態になります。

関連項目：

- 2-24 ページ「取得プロセス・ルールの評価」
- 12-10 ページ「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」
- 3-24 ページ「伝播用の Streams データ・ディクショナリ」
- 4-31 ページ「適用プロセス用の Streams データ・ディクショナリ」

Streams データ・ディクショナリの必要性を示す使用例 取得プロセスが表 t1 に対する変更を取得するように構成されている場合を考えます。この表には列 a および b があり、この表に対して 3 つの異なる時点で次の変更が行われるとします。

時刻 1: 値 a=7 および b=15 を挿入

時刻 2: 列 c を追加

時刻 3: 列 b を削除

なんらかの理由で取得プロセスがこれ以前の時点から変更を取得している場合、1 次データ・ディクショナリと Streams データ・ディクショナリ内の関連バージョンには異なる情報が含まれていることになります。表 2-1 に、現在の時刻が変更取得時刻とは異なる場合の、Streams データ・ディクショナリ内の情報の使用方法を示します。

表 2-1 1 次データ・ディクショナリ内と取得データ・ディクショナリ内の表 t1 に関する情報

現在の時刻	変更取得時刻	1 次データ・ディクショナリ	Streams データ・ディクショナリ
1	1	表 t1 には列 a および b が含まれています。	表 t1 には、時刻 1 における列 a および b が含まれています。
2	1	表 t1 には、列 a、b および c が含まれています。	表 t1 には、時刻 1 における列 a および b が含まれています。
3	1	表 t1 には列 a および c が含まれています。	表 t1 には、時刻 1 における列 a および b が含まれています。

取得プロセスは、実際の時刻が 3 のときに、時刻 1 に行われた挿入による変更を取得します。取得プロセスが 1 次データ・ディクショナリを使用していた場合は、列 a に値 7 が挿入され、列 c に値 15 が挿入されたと想定する可能性があります。これは、この 2 つが 1 次データ・ディクショナリ内の時刻 3 における表 t1 の列であるためです。ただし、実際には、値 15 が列 b に挿入されています。

取得プロセスは Streams データ・ディクショナリを使用するため、このエラーは回避されます。Streams データ・ディクショナリは取得プロセスと同期化され、表 t1 に時刻 1 の時点で列 a および b があることを引き続き記録します。そのため、取得された変更は値 15 が列 b に挿入されたことを示します。

ARCHIVELOG モードと取得プロセス

取得プロセスは、可能な場合はオンライン REDO ログを読み込み、それ以外の場合はアーカイブ REDO ログを読み込みます。このため、変更を取得するように取得プロセスを構成している場合は、データベースを ARCHIVELOG モードで実行する必要があります。アーカイブ REDO ログ・ファイルは、それを必要とする取得プロセスがないことが確実になるまで、使用可能にしておく必要があります。REDO ログ内のすべてのトランザクションがすべてのダウンストリーム・データベースで適用されるまで、REDO ログを使用可能にしてください。関連適用プロセスによりデキューされた最新のメッセージの SCN を判別するには、DBA_CAPTURE データ・ディクショナリ・ビューの APPLIED_SCN 列を使用します。この SCN より小さい番号のすべての変更は、取得プロセスで取得された変更を適用する適用プロセスすべてでデキューされています。

取得プロセスに遅れが生じると、オンライン REDO ログの読込みからアーカイブ REDO ログの読込みへとシームレスに推移し、取得プロセスが追いつくと、アーカイブ REDO ログの読込みからオンライン REDO ログの読込みへとシームレスに推移します。

> **関連項目：** ARCHIVELOG モードでデータベースを実行する方法については、『Oracle9*i* データベース管理者ガイド』を参照してください。

取得プロセスのパラメータ

作成後の取得プロセスは無効化されているため、初めて起動する前に環境に合せて取得プロセス・パラメータを設定できます。取得プロセス・パラメータでは、取得プロセスの動作を制御します。たとえば、time_limit 取得プロセス・パラメータを使用すると、取得プロセスが自動的に停止するまでの実行時間を指定できます。取得プロセス・パラメータの設定後に、取得プロセスを開始します。

関連項目：

- 12-7 ページ「取得プロセスのパラメータの設定」
- この項では、使用可能な取得プロセス・パラメータの一部のみを説明しています。すべての取得プロセス・パラメータの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_CAPTURE_ADM.SET_PARAMETER プロシージャの項を参照してください。

取得プロセスの並列性

parallelism 取得プロセス・パラメータでは、取得プロセスで使用するプリペアラ・サーバーの数を制御します。プリペアラ・サーバーは、REDO ログ内で検出された変更を同時に LCR 形式にフォーマットします。

注意：

- parallelism パラメータをリセットすると、取得プロセスが自動的に停止され、再起動されます。
- parallelism パラメータを使用可能なパラレル実行サーバー数より大きい数に設定すると、取得プロセスが無効化される場合があります。parallelism 取得プロセス・パラメータの設定時には、PROCESSES および PARALLEL_MAX_SERVERS 初期化パラメータが適切に設定されているかどうかを確認してください。

関連項目： プリペアラ・サーバーの詳細は、2-17 ページの「取得プロセスのコンポーネント」を参照してください。

取得プロセスの自動再起動

取得プロセスは、特定の上限に達した時点で自動的に停止するように構成できます。time_limit 取得プロセス・パラメータで取得プロセスの実行時間を指定し、message_limit 取得プロセス・パラメータで取得プロセスが取得できるイベントの数を指定します。取得プロセスは、これらのいずれかの上限に達した時点で自動的に停止します。

disable_on_limit パラメータでは、取得プロセスが上限に達した時点で無効化されるか、再起動するかを制御します。disable_on_limit パラメータを y に設定すると、取得プロセスは上限に達した時点で無効化され、明示的に指定するまで再起動されません。ただし、disable_on_limit パラメータを n に設定すると、取得プロセスは上限に達した時点で自動的に停止し、再起動します。

取得プロセスは、再起動されると最後に停止した時点から変更の取得を開始します。再起動された取得プロセスは新規のセッション識別子を取得し、取得プロセスに関連付けられているパラレル実行サーバーも新規のセッション識別子を取得します。ただし、取得プロセス番号（cp*nn*）は変わりません。

取得プロセス・ルールの評価

実行中の取得プロセスは、次の一連のアクションを完了して変更を取得します。

1. REDO ログ内で変更を検索します。
2. REDO ログ内で変更の事前フィルタ処理を実行します。この手順で、取得プロセスはオブジェクト・レベルとスキーマ・レベルでルール・セット内のルールを評価し、REDO ログ内で見つかった変更を 2 つのカテゴリに分類します。カテゴリの一方は、LCR への変換を必要とする変更で、他方は LCR に変換しない変更です。

   事前フィルタ処理は、不完全な情報を使用して実行される安全な最適化です。この手順では、次のように、それ以後に処理対象となる関連する変更が識別されます。

   - 変換後に 1 つ以上のルールを TRUE に評価できる場合は、変更が LCR に変換されます。
   - 変換後に TRUE に評価できるルールがないことを取得プロセスが保証できる場合、変更は LCR に変換されません。
3. 1 つ以上のルールが TRUE に評価される変更は、事前フィルタ処理に基づいて LCR に変換されます。
4. LCR のフィルタ処理を実行します。この手順で、取得プロセスは各 LCR 内の情報に関するルールを評価して、LCR を 2 つのカテゴリに分けます。カテゴリの一方は、エンキューする必要のある LCR で、他方は廃棄する必要のある LCR です。
5. ルールに基づいて、エンキューしない LCR を廃棄します。
6. 残っている取得済み LCR を、取得プロセスに関連付けられているキューにエンキューします。

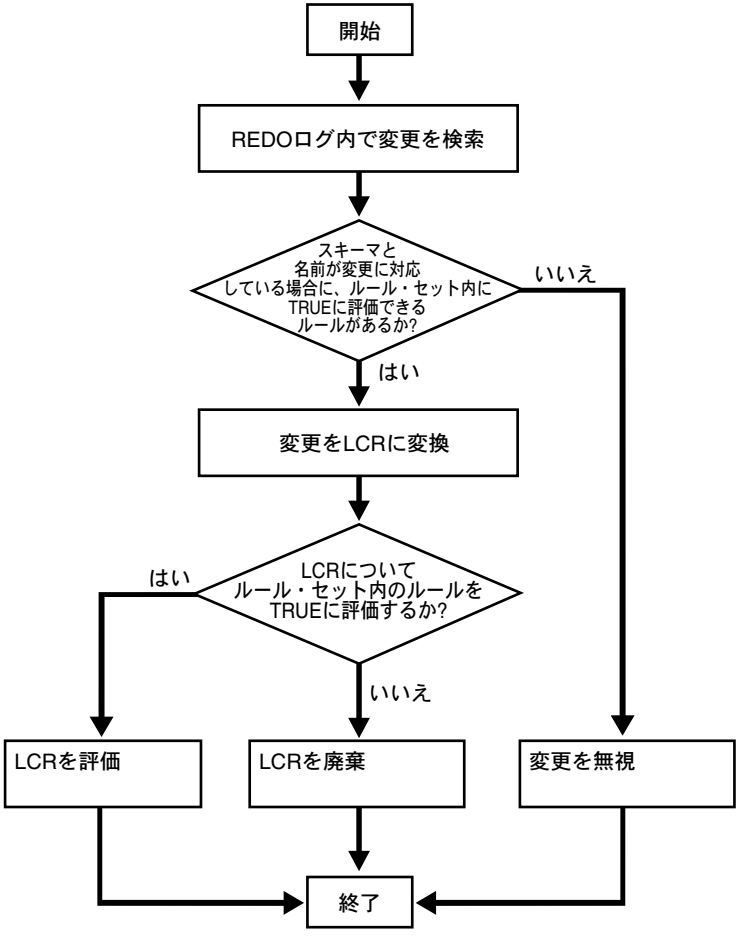
たとえば、取得プロセスに対して、department_id が 50 である hr.employees 表に対する変更を取得するというルールが定義されていると想定します。この取得プロセスに他のルールは定義されておらず、その parallelism パラメータは 1 に設定されています。

このルールがある場合に、hr.employees 表に対する UPDATE 文によって表の 50 行が変更されるとします。この取得プロセスは、各行の変更ごとに次の一連のアクションを実行します。

1. REDO ログ内で UPDATE 文によって発生する次の変更を検索します。
2. 変更が hr.employees 表に対する UPDATE 文によって発生しており、取得する必要があると判断します。変更が異なる表に対するものの場合、取得プロセスでは無視されます。
3. 変更を取得して LCR に変換します。
4. LCR をフィルタ処理し、department_id が 50 である行が関係しているかどうかを判断します。
5. LCR に含まれる行の department_id が 50 の場合は、取得プロセスに関連付けられているキューに LCR をエンキューします。LCR に含まれる行の department_id が 50 でないか department_id が含まれていない場合は、LCR を廃棄します。

図 2-2 に、取得プロセス・ルールの評価をフロー・チャート形式で示します。

図 2-2 取得プロセス・ルールの評価を示すフロー・チャート



取得プロセスの永続的状態

取得プロセスは、永続的状態を保持します。つまり、取得プロセスは、データベースが停止され、再起動されても、最新の状態に保たれます。たとえば、データベースの停止時に取得プロセスが実行中だった場合は、データベースが再起動されると取得プロセスは自動的に起動されます。ただし、データベースの停止時に取得プロセスが停止していた場合は、データベースを再起動しても取得プロセスは停止したままです。

3

Streams のステージングと伝播

この章では、キューにあるイベントのステージングと、キュー間でのイベントの伝播に関連する概念について説明します。

この章の内容は次のとおりです。

- イベントのステージングと伝播の概要
- 取得イベントとユーザー・エンキュー・イベント
- キュー間でのイベントの伝播
- SYS.AnyData 型のキューとユーザー・メッセージ
- Streams キューと Oracle Real Application Clusters
- Streams のステージングと伝播のアーキテクチャ

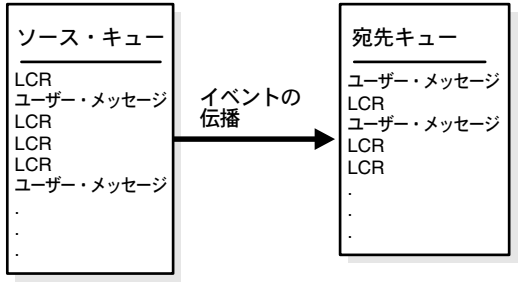
関連項目： 第 13 章「ステージングと伝播の管理」

イベントのステージングと伝播の概要

Streams では、SYS.AnyData 型のキューを使用してイベントがステージングされます。Streams キューでステージングできるイベントには、論理変更レコード（LCR）とユーザー・メッセージの 2 種類があります。LCR はデータベース・オブジェクトに対する変更情報を含むオブジェクトで、ユーザー・メッセージはユーザーまたはアプリケーションが作成するカスタム・メッセージです。どちらのタイプのイベントも SYS.AnyData 型であり、単一データベース内またはデータベース間で情報を共有するために使用できます。

ステージングされたイベントは、コンシュームまたは伝播、あるいはその両方が可能です。これらのイベントは、適用プロセスでコンシュームするか、明示的にデキューするユーザー・アプリケーションでコンシュームできます。イベントを 1 つ以上の他のキューに伝播するように Streams を構成している場合、またはメッセージ保存が指定されている場合は、イベントをコンシューム後もキューに残すことができます。この場合の他のキューは、同じデータベースにあっても異なるデータベースにあってもかまいません。どちらの場合も、イベントの伝播元となるキューは**ソース・キュー**、イベントを受信するキューは**宛先キュー**と呼ばれます。ソース・キューと宛先キューの間には、1 対多、多対 1 または多対多の関連が可能です。図 3-1 に、ソース・キューから宛先キューへの伝播を示します。

図 3-1 ソース・キューから宛先キューへの伝播



伝播の作成、変更および削除と、伝播するイベントを制御する伝播ルールの定義ができます。ソース・キューを所有するユーザーが、イベントを伝播するユーザーです。このユーザーは、イベントの伝播に必要な権限を持っている必要があります。これには、次の権限が含まれます。

- 伝播で使用するルール・セットの実行権限
- ルール・セットで使用するすべての変換ファンクションの実行権限
- 宛先キューが同じデータベースにある場合は、宛先キューのエンキュー権限

伝播によってイベントがリモート・データベース内の宛先キューに伝播される場合、ソース・キューの所有者は伝播のデータベース・リンクを使用できる必要があり、リモート・データベースでデータベース・リンクの接続先となるユーザーは、宛先キューに対するエンキュー権限を持っている必要があります。

注意： Streams の伝播では、接続修飾子は使用できません。

関連項目： メッセージ保存の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

取得イベントとユーザー・エンキュー・イベント

イベントをエンキューするには、次の 2 つの方法があります。

- 取得プロセスは、取得した変更を LCR を含むイベントの形式でエンキューします。最初に取得プロセスによって取得されエンキューされた LCR を含んでいるイベントは、**取得イベント**と呼ばれます。
- ユーザー・アプリケーションは、`SYS.AnyData` 型のユーザー・メッセージをエンキューします。これらのユーザー・メッセージには、LCR や他のタイプのメッセージを含めることができます。ユーザーまたはアプリケーションによって明示的にエンキューされたユーザー・メッセージは、**ユーザー・エンキュー・イベント**と呼ばれます。適用プロセスからコールされたユーザー・プロシージャによってエンキューされたイベントも、ユーザー・エンキュー・イベントです。

そのため、各取得イベントには LCR が含まれていますが、ユーザー・エンキュー・イベントには LCR が含まれていない場合もあります。取得イベントまたはユーザー・エンキュー・イベントを伝播すると、宛先キューにイベントがエンキューされます。

イベントをデキューするには、次の 2 つの方法があります。

- 適用プロセスは、取得イベントまたはユーザー・エンキュー・イベントをデキューします。イベントに LCR が含まれている場合、適用プロセスはそれを直接適用するか、ユーザー指定のプロシージャをコールして処理させることができます。イベントに LCR が含まれていない場合、適用プロセスではメッセージ・ハンドラと呼ばれるユーザー指定のプロシージャを起動してイベントが処理可能です。
- ユーザー・アプリケーションは、ユーザー・エンキュー・イベントを明示的にデキューして処理します。取得イベントはユーザー・アプリケーションではデキューできず、適用プロセスでデキューする必要があります。ただし、適用プロセスによってコールされたユーザー・プロシージャがイベントを明示的にエンキューする場合、そのイベントはユーザー・エンキュー・イベントであり、最初は取得イベントであったとしても明示的にデキューできます。

デキューされたイベントは、デキュー時と同じデータベースで発生した場合と、異なるデータベースで発生した場合があります。

関連項目：

- 取得プロセスの詳細は、第 2 章「Streams の取得プロセス」を参照してください。
- イベントをキューにエンキューする方法については、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。
- 適用プロセスの詳細は、第 4 章「Streams 適用プロセス」を参照してください。
- 16-2 ページ「論理変更レコード（LCR）の管理」

キュー間でのイベントの伝播

Streams を使用すると、異なるデータベースにある 2 つのキュー間でイベントを伝播するように構成できます。Streams では、ジョブ・キューを使用してイベントが伝播されます。

伝播は、常にソース・キューと宛先キューの間で行われます。伝播は常に 2 つのキュー間で行われますが、単一のキューが多数の伝播に関わることができます。つまり、単一のソース・キューがイベントを複数の宛先キューに伝播でき、単一の宛先キューが複数のソース・キューからイベントを受信できます。ただし、特定のソース・キューと特定の宛先キューの間に許される伝播は 1 つのみです。また、単一のキューが、ある伝播の宛先キューと他の伝播のソース・キューを兼ねることができます。

伝播では、ソース・キューにあるイベントすべてを宛先キューに伝播するか、イベントのサブセットのみを伝播できます。また、単一の伝播で取得イベントとユーザー・エンキュー・イベントの両方を伝播できます。ルールを使用すると、ソース・キューにあるどのイベントが宛先キューに伝播されるかを制御できます。

Streams 環境の設定方法によっては、変更が発生元のサイトに送り返される場合があります。変更が無限ループ内で循環しないように環境を構成していることを確認してください。Streams タグを使用すると、このような変更の循環ループを回避できます。

関連項目：

- 13-7 ページ「Streams の伝播および伝播ジョブの管理」
- AQ の伝播のインフラストラクチャの詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。
- 第 8 章「Streams のタグ」

伝播ルール

伝播では、定義したルールに基づいてイベントが伝播されます。LCR イベントの場合、各ルールでは、伝播によって変更が伝播されるデータベース・オブジェクトと、伝播される変更のタイプを指定します。LCR イベントの伝播ルールは次のレベルで指定できます。

- 表ルールは、特定の表に対する DML または DDL 変更を伝播します。
- スキーマ・ルールは、特定のスキーマ内のデータベース・オブジェクトに対する DML または DDL 変更を伝播します。
- グローバル・ルールは、ソース・キュー内のすべての DML または DDL 変更を伝播します。

非 LCR イベントの場合は、伝播を制御する独自のルールを作成できます。

条件を指定するキュー・サブスクライバによって、システムでルールが生成されます。キューへのすべてのサブスクライバ用のルール・セットが組み合されて、サブスクリプションをより効率的にする単一のシステム生成ルール・セットとなります。

関連項目：

- 第 5 章「ルール」
- 第 6 章「Streams でのルールの使用方法」

保証付きのイベント配信

ユーザー・エンキュー・イベントが宛先キューに正常に伝播されるのは、宛先キューへのエンキューがコミットされる時点です。取得イベントが宛先キューに正常に伝播されるのは、次の両方のアクションが完了した時点です。

- イベントが、宛先キューに関連付けられた関連するすべての適用プロセスによって処理される場合。
- イベントが宛先キューからその関連宛先キューすべてに正常に伝播される場合。

2 つの Streams キュー間でイベントが正常に伝播されると、宛先キューはそのことを確認します。ソース・キューがイベントを複数の宛先キューに伝播するように構成されている場合は、各宛先キューがソース・キューにイベント伝播の確認を送信し終わるまで、イベントはソース・キューに残ります。各宛先キューがイベントの正常な伝播を確認し、ソース・キュー・データベース内のすべてのローカル・コンシューマがイベントをコンシュームし終わると、ソース・キューはイベントを削除できます。

この確認システムにより、イベントが常にソース・キューから宛先キューに確実に伝播されますが、構成によってはソース・キューが最適サイズよりも大きくなる可能性があります。ソース・キューが大きくなると、SGA メモリーの使用量が増加し、ディスク領域の使用量も増加する場合があります。

ソース・キューの拡大には、通常、次の 2 つの原因があります。

- なんらかの理由（ネットワークの問題など）でイベントを指定された宛先キューに伝播できない場合、イベントはソース・キューに無限に残ります。この状況がソース・キューの拡大の原因となることがあります。そのため、キューを定期的に監視して問題を早期に検出する必要があります。
- ソース・キューがイベントを複数の宛先キューに伝播しており、1 つ以上の接続先データベースが、他のキューよりもはるかに低速でイベントが正常に伝播されたことを確認する場合を考えます。この場合、低速の接続先データベースでは、高速の接続先データベースによってすでに確認済みのイベントのバックログが作成されるため、ソース・キューが大きくなることがあります。このような環境では、複数の取得プロセスを作成してソース・データベースでの変更を取得することを考慮してください。これにより、あるソース・キューを低速の接続先データベースに、別のソース・キューを高速の接続先データベースに使用できます。

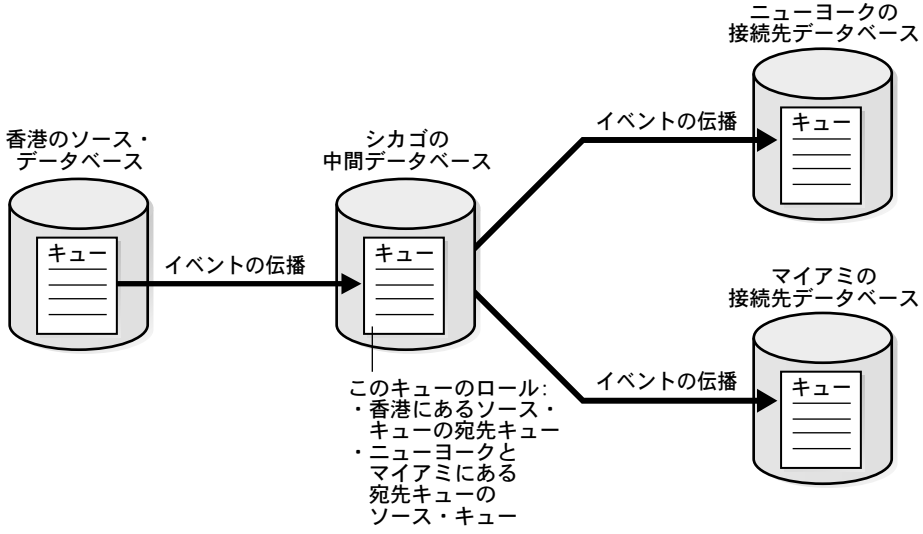
関連項目：

- 第 2 章「Streams の取得プロセス」
- 17-12 ページ「Streams キューの監視」

有向ネットワーク

有向ネットワークとは、伝播されるイベントが接続先データベースに到達する前に 1 つ以上の中間データベースを通過するネットワークです。イベントは、中間データベースで適用プロセスによって処理される場合と、処理されない場合があります。Streams を使用すると、各接続先データベースに伝播させるイベントを選択し、イベントが接続先データベースに到達するまでのルートを指定できます。図 3-2 に、有向ネットワーク環境の例を示します。

図 3-2 有向ネットワーク環境の例



有向ネットワークを使用する利点は、ソース・データベースと接続先データベースの間に物理的なネットワーク接続を必要としないことです。そのため、イベントをデータベース間で伝播させる必要があるが、これらのデータベースを稼働させているコンピュータ間にダイレクト・ネットワーク接続がない場合も、1 つ以上の中間データベースがソース・データベースを接続先データベースに接続しているかぎり、ネットワークを再構成せずにイベントを伝播させることができます。

有向ネットワークを使用している場合に、中間サイトが長期間停止されるか、中間サイトが削除されると、ネットワークと Streams 環境の再構成が必要になる場合があります。

キューによる転送と適用による転送

有向ネットワークにおける中間データベースでは、キューによる転送または適用による転送を使用してイベントを伝播できます。**キューによる転送**は、中間データベースで受信されたイベントが転送されることを意味します。イベントのソース・データベースは、イベントが発生したデータベースです。

適用による転送は、中間データベースで転送されるイベントが、最初に適用プロセスによって処理されることを意味します。この種のイベントは、中間データベースで取得プロセスによって再取得されてから転送されます。適用による転送を使用すると、イベントは中間データベースで再取得されるため、中間データベースはイベントの新しいソース・データベースとなります。

Streams 環境のプランを作成するときには、キューによる転送と適用による転送に次の違いがあることに注意してください。

- キューによる転送を使用すると、取得または伝播に伴う変換がなければ、イベントは変更されずにそのまま有向ネットワークを介して伝播されます。適用による転送を使用すると、イベントは中間データベースで適用されてから再取得されます。競合解消、適用ハンドラまたは適用変換によって変更される場合があります。
- キューによる転送を使用する場合、接続先データベースでは、各ソース・データベースからのイベントを別個の適用プロセスで適用する必要があります。適用による転送を使用すると、中間データベースでイベントが再取得されることによって、変更が接続先データベースに到達する時点でソース・データベースの数が減少する場合があるため、接続先データベースに必要な適用プロセスが少数で済む可能性があります。
- キューによる転送を使用する場合は、ソース・データベースと接続先データベースの間に 1 つ以上の中間データベースが位置します。適用による転送を使用する場合、イベントは中間データベースで再取得されるため、イベントのソース・データベースは、接続先データベースと直接接続している中間データベースと同じでもかまいません。

単一の Streams 環境で、キューによる転送と適用による転送を併用できます。

キューによる転送の利点 キューによる転送を適用による転送と比べると、次のような利点があります。

- イベントが一度のみ取得されるため、パフォーマンスを改善できます。
- イベントが 1 つ以上の中間データベースで適用されてから再取得されることがないため、イベントを発生元データベースから接続先データベースへと伝播させる所要時間が短縮されます。つまり、キューによる転送を使用すると、待機時間を短縮できる場合があります。
- イベントのソース・データベースは、イベントに含まれている LCR に対して `GET_SOURCE_DATABASE_NAME` メンバー・プロシージャを実行すると簡単に判断できます。適用による転送を使用する場合は、イベントの発生元で Streams タグと適用ハンドラを使用する必要があるかどうかを判断してください。
- 別々の適用プロセスを使用すると、依存性が低減し、複数の適用コーディネータと適用リーダー・プロセスによって作業が実行されるため、パラレル適用によって拡張性が向上し、スループットが増大します。
- 1 つの中間データベースが停止した場合は、キューを再度ルーティングし、取得サイトで開始 SCN をリセットして、エンド・トゥ・エンドの取得、伝播および適用を再構成できます。

  適用による転送を使用すると、使用不可能な中間データベースから接続先データベースへのダウンストリームでは、この中間データベースの SCN 情報が使用されているため、イベントについてエンド・トゥ・エンドの取得、伝播および適用の再構成に必要な作業量が実質的に増大します。この SCN 情報がなければ、接続先データベースでは変更を正しく適用できません。

適用による転送の利点 適用による転送をキューによる転送と比べると、次のような利点があります。

- 各データベースは、複数のリモート・ソース・データベースではなく直接接続しているデータベースからのみ変更を適用できるため、Streams 環境の構成が容易です。
- 中間データベースによって変更が適用される大規模な Streams 環境では、適用プロセスが少数で済むため、環境の監視と管理が容易です。変更を適用する中間データベースでは、変更を受信するソース・データベースごとに 1 つの適用プロセスが必要です。適用による転送環境では、中間データベースのソース・データベースは、直接接続されているデータベースのみです。キューによる転送環境では、中間データベースのソース・データベースは、中間データベースに直接接続されているかどうかに関係なく環境内の他のすべてのソース・データベースです。
- 複数のソースがある Streams 環境では、そのままデータベースを追加できます。各データベースでオブジェクトに対するすべての DML を停止し、そのオブジェクトに関係するすべての LCR が取得、伝播および適用されるまで待機する必要はありません。新規データベースは、Streams 環境の残りのデータベースに接続する 1 つのデータベースからインスタンス化されます。これに対して、キューによる転送環境では、複数のソースとの共有オブジェクトに関する現行のデータすべてが単一のデータベースに含まれるわけではないため、環境に新規データベースを追加するときには DML を停止する必要があります。

関連項目：

- 第 4 章「Streams 適用プロセス」
- キューによる転送を使用する環境の例は、第 22 章「単一ソースの異機種間レプリケーションの例」を参照してください。
- 適用による転送を使用する環境の例は、8-11 ページの「プライマリ・データベースが複数のセカンダリ・データベースとデータを共有している場合」を参照してください。

SYS.AnyData 型のキューとユーザー・メッセージ

Streams では、SYS.AnyData 型のキューを使用したメッセージングが可能です。これらのキューは Streams キューと呼ばれます。Streams キューでは、SYS.AnyData 型のペイロードを持つユーザー・メッセージをステージングできます。SYS.AnyData 型のペイロードには、異なるデータ型のペイロード用のラッパーを使用できます。特定の型のメッセージしかステージングできないキューは、型付きのキューと呼ばれます。

メッセージ・ペイロードに SYS.AnyData ラッパーを使用すると、パブリッシャ・アプリケーションでは様々な型のメッセージを 1 つのキューにエンキューし、サブスクライバ・アプリケーションではこれらのメッセージをデキュー API を使用して明示的に、または適用プロセスを使用して暗黙的にデキューできます。サブスクライバ・アプリケーションがリモートの場合は、メッセージをリモート・サイトに伝播し、サブスクライバ・アプリケーションでリモート・データベースのローカル・キューからメッセージをデキューできます。また、リモート・サブスクライバ・アプリケーションでは、PL/SQL や OCI など、様々な標準プロトコルを使用して、ソース・キューからメッセージを直接デキューできます。

Streams と相互運用されるアドバンスト・キューイング（AQ）では、マルチ・コンシューマ・キュー、パブリッシュおよびサブスクライブ、コンテンツ・ベースのルーティング、インターネット伝播、変換および他のメッセージ・サブシステムへのゲートウェイなど、メッセージ・キューイング・システムのすべての標準機能がサポートされます。

関連項目：

- 13-17 ページ「Streams メッセージ環境の管理」
- 第 19 章「Streams メッセージングの例」
- AQ の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

ユーザー・メッセージ・ペイロード用の SYS.AnyData ラッパー

ほぼすべての型のペイロードは、SYS.AnyData ペイロードでラップできます。そのためには、SYS.AnyData 型の Convert*data_type* 静的ファンクションを使用します。この場合、*data_type* はラップするオブジェクトの型です。これらのファンクションは、入力としてオブジェクトを取り、SYS.AnyData オブジェクトを戻します。

次のデータ型は、SYS.AnyData ラッパーでラップできません。

- NESTED TABLE
- NCLOB
- ROWID および UROWID

次のデータ型は、SYS.AnyData ラッパーで直接ラップできますが、SYS.AnyData ラッパーでラップされるユーザー定義型ペイロードには含められません。

- CLOB
- BLOB
- BFILE
- VARRAY

関連項目：

- 13-18 ページ「SYS.AnyData ラッパーでのユーザー・メッセージ・ペイロードのラップ」
- SYS.AnyData 型の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ユーザー・メッセージのエンキューとデキューのためのプログラムによる環境

アプリケーションでは、次のプログラムによる環境を使用して、ユーザー・メッセージを Streams キューとの間でエンキューおよびデキューできます。

- PL/SQL（DBMS_AQ パッケージ）
- JMS
- OCI

ここでは、これらのインタフェースを使用して、Streams キューとの間でユーザー・メッセージのエンキューとデキューを行う方法について説明します。

関連項目： これらのプログラム・インタフェースの詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

PL/SQL を使用したユーザー・メッセージのエンキュー

LCR を含むユーザー・メッセージを、PL/SQL を使用して Streams キューにエンキューするには、最初にエンキューする LCR を作成します。行 LCR の作成には SYS.LCR$_ROW_RECORD 型のコンストラクタ、DDL LCR の作成には SYS.LCR$_DDL_RECORD 型のコンストラクタを使用します。次に、SYS.AnyData.ConvertObject ファンクションを使用して LCR を SYS.AnyData ペイロードに変換し、DBMS_AQ.ENQUEUE プロシージャを使用してエンキューします。

LCR 以外のオブジェクトを含むユーザー・メッセージを、PL/SQL を使用して Streams キューにエンキューするには、SYS.AnyData.Convert* ファンクションの 1 つを使用してオブジェクトを SYS.AnyData ペイロードに変換し、DBMS_AQ.ENQUEUE プロシージャを使用してエンキューします。

関連項目：

- 13-17 ページ「Streams メッセージ環境の管理」
- 第 19 章「Streams メッセージングの例」

OCI または JMS を使用したユーザー・メッセージのエンキュー

LCR を含むユーザー・メッセージを、JMS または OCI を使用して Streams キューにエンキューするには、LCR を XML 形式で表す必要があります。LCR を構成するには、oracle.xdb.XMLType クラスを使用します。LCR は SYS スキーマに定義されます。Oracle ホームの rdbms/admin/ ディレクトリにある catxlcr.sql スクリプトを使用して、LCR スキーマを SYS スキーマにロードする必要があります。

OCI を使用してメッセージをエンキューするには、型付きのキューにエンキューする場合と同じ操作を実行します。型付きのキューとは、特定の型のメッセージしかステージングできないキューです。JMS を使用してメッセージをエンキューするには、DBMS_AQ、DBMS_AQIN および DBMS_AQJMS パッケージに対する EXECUTE 権限を持っている必要があります。

LCR 以外のユーザー・メッセージには、ユーザー定義型のメッセージや JMS 型のメッセージがあります。JMS には、次の型があります。

- javax.jms.TextMessage
- javax.jms.MapMessage
- javax.jms.StreamMessage
- javax.jms.ObjectMessage
- javax.jms.BytesMessage

ユーザー定義型を使用する場合は、JPublisher を使用してメッセージ用の Java クラスを生成する必要があります。これにより、ORAData インタフェースが実装されます。メッセージを Streams キューにエンキューするには、メソッド QueueSender.send または TopicPublisher.publish を使用できます。

関連項目：

- 19-33 ページ「JMS を使用したイベントのエンキューとデキュー」
- メッセージを XML 形式で表す方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』および『Oracle9*i* XML データベース開発者ガイド - Oracle XML DB』を参照してください。
- `oracle.jms` Java パッケージの詳細は、『Oracle9*i* Java パッケージ・プロシージャ・リファレンス』を参照してください。
- OCI を使用したメッセージのエンキューの詳細は、『Oracle Call Interface プログラマーズ・ガイド』の `OCIAQenq` ファンクションの項を参照してください。

PL/SQL を使用したユーザー・メッセージのデキュー

PL/SQL を使用してユーザー・メッセージを Streams キューからデキューするには、`DBMS_AQ.DEQUEUE` プロシージャを使用し、ペイロードとして `SYS.AnyData` を指定します。ユーザー・メッセージには、LCR が含まれている場合と、他の型のオブジェクトが含まれている場合があります。

関連項目：

- 13-17 ページ「Streams メッセージ環境の管理」
- 第 19 章「Streams メッセージングの例」

OCI または JMS を使用したユーザー・メッセージのデキュー

Streams キューでは、XML 形式の LCR を含むユーザー・メッセージは `oracle.xdb.XMLType` として表されます。LCR 以外のメッセージには、次のいずれかのフォーマットがあります。

- JMS の型（`javax.jms.TextMessage`、`javax.jms.MapMessage`、`javax.jms.StreamMessage`、`javax.jms.ObjectMessage` または `javax.jms.BytesMessage`）
- ユーザー定義型

JMS を使用して Streams キューからメッセージをデキューするには、メソッド `QueueReceiver`、`TopicSubscriber` または `TopicReceiver` を使用できます。キューには、`SYS.AnyData` ラッパーでラップされた様々な型のオブジェクトが含まれている可能性があるため、SQL の型とそれに対応する Java クラスのリストを JMS セッションの型マップに登録する必要があります。JMS の型は、すでに型マップに登録されています。

たとえば、キューに oracle.xdb.XMLType として表される LCR メッセージと person 型および address 型のメッセージが含まれると想定します。クラス JPerson.java および JAddress.java は、それぞれ person および address の ORAData マッピングです。メッセージをデキューする前に、型マップを次のように移入する必要があります。

```
java.util.Dictionary map = ((AQjmsSession)q_sess).getTypeMap();

map.put("SCOTT.PERSON", Class.forName("JPerson"));
map.put("SCOTT.ADDRESS", Class.forName("JAddress"));
map.put("SYS.XMLTYPE", Class.forName("oracle.xdb.XMLType"));  // For LCRs
```

QueueReceiver または TopicPublisher とともにメッセージの選択指定を使用する場合、選択指定には次の 1 つ以上の組合せによる SQL92 の式を含めることができます。

- JMS メッセージ・ヘッダー・フィールドまたはプロパティ。JMSPriority、JMSCorrelationID、JMSType、JMSXUserI、JMSXAppID、JMSXGroupID および JMSXGroupSeq などです。次に JMS メッセージ・フィールドの例を示します。

  ```
  JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
  ```

- ユーザー定義のメッセージのプロパティ。次に例を示します。

  ```
  color IN ('RED', 'BLUE', 'GREEN') AND price < 30000
  ```

- PL/SQL ファンクション。次に例を示します。

  ```
  hr.GET_TYPE(tab.user_data) = 'HR.EMPLOYEES'
  ```

OCI を使用してメッセージをデキューするには、型付きのキューからデキューする場合と同じ操作を実行します。

関連項目：

- 19-33 ページ「JMS を使用したイベントのエンキューとデキュー」
- メッセージを XML 形式で表す方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』および『Oracle9*i* XML データベース開発者ガイド - Oracle XML DB』を参照してください。
- oracle.jms Java パッケージの詳細は、『Oracle9*i* Java パッケージ・プロシージャ・リファレンス』を参照してください。
- OCI を使用したメッセージのデキューの詳細は、『Oracle Call Interface プログラマーズ・ガイド』の OCIAQdeq ファンクションの項を参照してください。

メッセージの伝播と SYS.AnyData キュー

Streams 環境では、SYS.AnyData キューと型付きのキューを相互運用できます。型付きのキューでは、特定の型を持つメッセージしかステージングできません。表 3-1 に、キュー間で可能な伝播のタイプを示します。

表 3-1 各種キュー間での伝播

ソース・キュー	宛先キュー	変換
SYS.AnyData	SYS.AnyData	なし。
型付き	SYS.AnyData	暗黙的。 **注意：**伝播が可能なのは、型付きのキューにあるメッセージが 3-16 ページの「ユーザー定義型のメッセージ」で説明する制限を満たしている場合のみです。
SYS.AnyData	型付き	メッセージをフィルタ処理するためのルールとユーザー定義変換が必要です。
型付き	型付き	アドバンスト・キューイング（AQ）のルールに従います（詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照）。

特定の型を持つペイロードを含むメッセージを SYS.AnyData ソース・キューから型付きの宛先キューに伝播させるには、変換を実行する必要があります。型付きのキューに伝播できるのは、そのキューと同じ型のペイロードを含むメッセージのみです。

Simple Object Access Protocol（SOAP）を使用して Streams キューと直接対話することはできませんが、SOAP を Streams と併用することで、Streams キューと型付きのキューの間でメッセージを伝播できます。SOAP を使用してメッセージを Streams キューにエンキューするには、型付きのキューから Streams キューへの伝播を構成します。構成後、SOAP を使用してメッセージを型付きのキューにエンキューします。型付きのキューから Streams キューへメッセージが自動的に伝播されます。

SOAP を使用して Streams キューに含まれるメッセージをデキューする場合は、Streams キューから型付きのキューへの伝播を構成します。Streams キューから型付きのキューへメッセージが自動的に伝播されます。そうすると、SOAP を使用してメッセージにアクセス可能になります。

注意： 取得プロセスを使用した変更の取得や、適用プロセスを使用した変更の適用など、特定の Streams 機能を構成するには、SYS.AnyData キューを使用する必要があります。

関連項目：

- 13-22 ページ「SYS.AnyData キューと型付きのキューの間でのメッセージの伝播」
- SOAP および型付きのキューの使用方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

ユーザー定義型のメッセージ

Streams 環境でユーザー定義型のメッセージをエンキュー、伝播またはデキューする予定の場合は、これらのメッセージに使用されるそれぞれの型が、メッセージをキュー内でステージングできる各データベースに存在する必要があります。一部の環境では、有向ネットワークが使用され、メッセージは中間データベースを介してルーティングされてから宛先に到達します。このような環境では、ある型のメッセージが特定の中間データベースでエンキューまたはデキューされることがない場合も、その型が各中間データベースに存在する必要があります。

また、それぞれの型は次の要件を満たす必要があります。

- 型名が各データベースで同じであること。
- 型が各データベースで同じスキーマにあること。
- 型の shape が各データベースで正確に一致すること。
- どのデータベースでも、型には継承や型進化は使用しないこと。
- 型には、VARRAY、NESTED TABLE、LOB、ROWID または UROWID を含めないこと。

オブジェクト識別子（OID）は、各データベースで一致する必要はありません。

関連項目：

- ユーザー定義型のメッセージ・ペイロードを `SYS.AnyData` メッセージでラップする方法については、3-10 ページの「ユーザー・メッセージ・ペイロード用の SYS.AnyData ラッパー」を参照してください。
- 3-6 ページ「有向ネットワーク」

Streams キューと Oracle Real Application Clusters

Real Application Clusters 環境で、取得イベントとユーザー・エンキュー・イベントをステージングして伝播するように、Streams キューを構成できます。Real Application Clusters 環境では、キューのバッファを使用できるのは所有者インスタンスのみです。様々なインスタンスが異なるキューにバッファを使用できます。キュー・バッファについては後述します。キュー・バッファは、取得イベントのみを含む Streams キューに関連付けられたシステム・グローバル領域（SGA）メモリーです。

Real Application Clusters 環境では、ユーザー・エンキュー・イベントのみを含む Streams キューの動作は型付きのキューと同じです。ただし、現在または将来、Real Application Clusters 環境で Streams キューに取得イベントが含まれる場合、次の要件を満たす必要があります。

- DBMS_STREAMS_ADM パッケージの SET_UP_QUEUE プロシージャを使用して、取得イベントがある Streams キューを含む各キュー表を作成すること。キュー表のいずれかのキューに取得イベントが含まれる場合、DBMS_AQADM パッケージを使用してキュー表を作成または変更する操作はサポートされません。
- 取得イベントを処理し、特定の Streams キューを使用する取得プロセスと適用プロセスは、すべてキューの所有者インスタンスで起動すること。
- 取得イベントを Real Application Clusters の接続先データベースに伝播する各伝播は、宛先キューの所有者インスタンスを参照するインスタンス固有のデータベース・リンクを使用すること。伝播が他のインスタンスに接続すると、伝播でエラーが発生します。
- すべてのインスタンスで AQ タイム・マネージャを実行すること。したがって、各インスタンス上で、AQ_TM_PROCESSES 初期化パラメータを 1 以上に設定してください。

宛先キューを含むキュー表の所有者インスタンスが使用不可能になると、キューの所有権は自動的にクラスタ内の別のインスタンスに移ります。この状況が発生した場合は、リモート・ソース・キューからのデータベース・リンクを、宛先キューを所有するインスタンスに接続するように、手動で再構成する必要があります。DBA_QUEUE_TABLES データ・ディクショナリ・ビューは、キュー表の所有者インスタンスに関する情報を示します。キュー表には、複数のキューが含まれている場合があります。この場合、キュー表内の各キューの所有者インスタンスは、キュー表と同じになります。

関連項目：

- 2-15 ページ「Streams の取得プロセスと Oracle Real Application Clusters」
- 4-27 ページ「Streams の適用プロセスと Oracle Real Application Clusters」
- 3-19 ページ「キュー・バッファ」
- DBA_QUEUE_TABLES データ・ディクショナリ・ビューの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。
- キューと Real Application Clusters の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

Streams のステージングと伝播のアーキテクチャ

通常、Streams キューと伝播には AQ のインフラストラクチャが使用されます。ただし、AQ キューではキュー表内のすべてのイベントがステージングされるのに対して、Streams キューにはキュー・バッファがあり、そこで共有メモリー内の取得イベントがステージングされます。この項では、キュー・バッファとその Real Application Clusters 環境での使用方法、さらに伝播ジョブおよび保護キューとそれらの Streams での使用方法について説明します。また、トランザクション型のキューで取得イベントおよびユーザー・エンキュー・イベントがどのように処理されるかと、取得イベントを伝播するデータベースでの Streams データ・ディクショナリの必要性についても説明します。

関連項目：

- AQ のインフラストラクチャの詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。
- DBMS_JOB パッケージの詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

キュー・バッファ

キュー・バッファは、取得イベントのみを含む Streams キューに関連付けられたシステム・グローバル領域（SGA）メモリーです。キュー・バッファにより、Oracle では、常にキュー表に格納するかわりに SGA にバッファリングすることで、取得イベントを最適化できます。この取得イベントのバッファリングは、Streams キュー内で取得イベントがステージングされるデータベースで発生します。この種のデータベースは、ソース・データベース、中間データベースまたは接続先データベースのいずれでもかまいません。LCR ユーザー・エンキュー・イベントと LCR 以外のユーザー・エンキュー・イベントは、キュー・バッファではなく常にキュー表内でステージングされます。

キュー・バッファによってパフォーマンスは改善されますが、バッファを含むインスタンスが正常または異常な状態で停止すると、キュー・バッファの内容は失われます。インスタンス上でデータベース全体のリカバリが実行されると、Streams はこのような場合に自動的にリカバリします。

単一データベースでは、結合されたすべてのキュー・バッファにより共有プールが最大 10% まで使用されます。取得イベントの保持に使用できる共有メモリーが足りないと、キュー・バッファがオーバーフローする場合があります。キュー・バッファをオーバーフローした取得イベントは、適切な AQ$_*queue_table_name*_p 表に格納されます。*queue_table_name* はこのキューのキュー表名です。キュー・バッファ内のイベントが失われると、キュー・バッファとそのキュー表の同期状態を保つために、キュー・バッファからあふれたイベントも続いて削除されます。また、トランザクションが例外キューに移動すると、トランザクション内のすべてのイベントは、キュー・バッファではなくキュー表内でステージングされます。

関連項目：

- SGA の詳細は、『Oracle9*i* データベース概要』を参照してください。
- 16-27 ページ「接続先データベースにおけるデータベースの Point-in-Time リカバリの実行」

伝播ジョブ

Streams 伝播は、DBMS_JOBS パッケージを使用して内部的に構成されます。したがって、伝播ジョブはイベントをソース・キューから宛先キューに伝播するメカニズムです。DBMS_JOBS パッケージを使用して構成される他のジョブと同様に、伝播ジョブには所有者が存在し、必要に応じてジョブ・キュー・プロセス（Jnnn）を使用してジョブが実行されます。

伝播ジョブは、複数の伝播で使用される場合があります。データベースのすべての宛先キューは、単一の伝播ジョブを介して単一のソース・キューからイベントを受信します。Streams では、複数の宛先キューに単一の伝播ジョブを使用することで、同じデータベースにある複数の宛先キューが同じメッセージを受信する場合にも、1 つのイベントが接続先データベースに確実に一度のみ送信されます。メッセージが同じデータベースに複数回送信されることはないため、通信リソースが節減されます。

注意：

- 現在、データベース・リンクが複数の宛先キューにイベントを伝播するために複数の伝播で使用されている場合でも、そのデータベース・リンクを使用するすべてのイベントが、単一の伝播ジョブによって伝播されます。
- ソース・キューの所有者が伝播を実行しますが、伝播ジョブの所有者はそれを作成したユーザーです。この 2 人のユーザーは、同一であっても異なっていてもかまいません。

伝播のスケジューリングと Streams 伝播

伝播スケジュールでは、伝播ジョブでソース・キューから宛先キューにイベントを伝播する頻度を指定します。したがって、伝播ジョブを使用するすべての伝播で伝播スケジュールが同じとなります。デフォルトの伝播スケジュールは、次のいずれかのプロシージャを使用して伝播ジョブを作成するときに、その伝播ジョブに対して設定されます。

- DBMS_STREAMS_ADM パッケージの ADD_GLOBAL_PROPAGATION_RULE プロシージャ
- DBMS_STREAMS_ADM パッケージの ADD_SCHEMA_PROPAGATION_RULE プロシージャ
- DBMS_STREAMS_ADM パッケージの ADD_TABLE_PROPAGATION_RULE プロシージャ
- DBMS_PROPAGATION_ADM パッケージの CREATE_PROPAGATION プロシージャ

デフォルトのスケジュールのプロパティは、次のとおりです。

- 開始時刻は SYSDATE() です。
- 継続時間は無限を意味する NULL です。
- 次回の時刻は NULL で、これは現行の継続時間が終了した直後に伝播が再開されることを意味します。
- 待機時間は 5 秒間で、これは同じ宛先キューへの伝播を必要とするメッセージがないときに、メッセージがキューにエンキューされてから宛先キューに伝播されるまでの待機時間です。

伝播ジョブのデフォルト・スケジュールを変更する場合は、DBMS_AQADM パッケージの ALTER_PROPAGATION_SCHEDULE プロシージャを使用します。

関連項目： 13-12 ページ「伝播ジョブのスケジュールの変更」

伝播ジョブと RESTRICTED SESSION

システム起動時に STARTUP RESTRICT 文を発行することで制限付きセッションが有効化された場合、伝播スケジュールが有効化された伝播ジョブでのイベントの伝播は行われません。制限付きセッションが無効化されると、有効化されて実行準備が整っている各伝播スケジュールは、使用可能なジョブ・キュー・プロセスがあるときに実行されます。

SQL 文 ALTER SYSTEM および ENABLE RESTRICTED SESSION 句により実行中のデータベースで制限付きセッションが有効化された場合、実行中の伝播ジョブは完了するまで引き続き実行されます。ただし、伝播スケジュールに対して発行された新規の伝播ジョブは開始されません。したがって、有効化されたスケジュールの伝播が最終的に停止する可能性があります。

保護キュー

保護キューは、エンキューやデキューなどのキュー操作を実行できる 1 人以上のデータベース・ユーザーに、AQ エージェントを明示的に関連付ける必要のあるキューです。保護キューの所有者はキューに対してすべてのキュー操作を実行できますが、他のユーザーは、**保護キュー・ユーザー**として構成されていないかぎり、保護キューに対するキュー操作を実行できません。Streams では、保護キューを使用して、適切なユーザーと Streams プロセスのみがキューとの間でイベントのエンキューとデキューを確実に行えます。

DBMS_STREAMS_ADM パッケージの SET_UP_QUEUE プロシージャを使用して作成された Streams キューは、すべて保護キューです。SET_UP_QUEUE プロシージャを使用してキューを作成すると、queue_user パラメータで指定したユーザーは、可能であればキューの保護キュー・ユーザーとして自動的に構成されます。また、キュー・ユーザーには、キューに対する ENQUEUE および DEQUEUE 権限が付与されます。キューとの間でイベントのエンキューとデキューを行うには、キュー・ユーザーは DBMS_AQ パッケージに対する EXECUTE 権限も必要です。SET_UP_QUEUE プロシージャでは、この権限は付与されません。

キュー・ユーザーを保護キュー・ユーザーとして構成するために、SET_UP_QUEUE プロシージャでは、存在しない場合はユーザー名と同じ名前で AQ エージェントが作成されます。ユーザーは、このエージェントを使用してキューに対するキュー操作を実行する必要があります。この名前を持つエージェントがすでに存在し、特定のキュー・ユーザーにのみ関連付けられている場合は、そのエージェントが使用されます。SET_UP_QUEUE では、エージェントとユーザーを指定して、DBMS_AQADM パッケージの ENABLE_DB_ACCESS プロシージャが実行されます。SET_UP_QUEUE が作成しようとしたエージェントがすでに存在し、queue_user で指定した以外のユーザーに関連付けられている場合は、エラーが発生します。この場合は、DBMS_AQADM パッケージの ALTER_AQ_AGENT または DROP_AQ_AGENT プロシージャを使用して、それぞれ既存のエージェントの名前を変更するか、削除する必要があります。その後に、SET_UP_QUEUE を再試行してください。

取得プロセスまたは適用プロセスを作成すると、Streams プロセスに関連付けられている保護キューの AQ エージェントが自動的に構成され、Streams プロセスを実行するユーザーは、自動的にこのキューの保護キュー・ユーザーとして指定されます。したがって、取得プロセスは保護キューを自動的にエンキューするように構成され、適用プロセスは保護キューから自動的にデキューするように構成されます。

取得プロセスの場合、取得プロセスの作成プロシージャを起動したユーザーが、取得プロセスを実行するユーザーとなります。適用プロセスの場合は、apply_user として指定したユーザーが、適用プロセスを実行するユーザーとなります。apply_user を指定しなければ、適用プロセスの作成プロシージャを起動したユーザーが、適用プロセスを実行するユーザーとなります。

また、DBMS_APPLY_ADM パッケージの ALTER_APPLY プロシージャを使用して適用プロセスの apply_user を変更すると、指定した apply_user は適用プロセスで使用されるキューの保護キュー・ユーザーとして構成されます。ただし、古い適用ユーザーは、キューの保護キュー・ユーザーとして構成されたままです。古い適用ユーザーを削除するには、そのユーザーと関連 AQ エージェントを指定して、DBMS_AQADM パッケージの DISABLE_DB_ACCESS プロシージャを実行します。また、不要になったエージェントは削除できます。DBA_AQ_AGENT_PRIVS データ・ディクショナリ・ビューを問い合せると、AQ エージェントとその関連ユーザーを確認できます。

DBMS_AQADM パッケージを使用して SYS.AnyData キューを作成する場合は、CREATE_QUEUE_TABLE プロシージャを実行するときに secure パラメータを使用して、キューが保護キューかどうかを指定します。このプロシージャを実行するときに secure パラメータに true を指定すると、キューは保護キューになります。DBMS_AQADM パッケージを使用して保護キューを作成し、ユーザーが保護キューに対してキュー操作を実行できるようにする必要がある場合は、これらの保護キュー・ユーザーを手動で構成する必要があります。

DBMS_STREAMS_ADM パッケージの SET_UP_QUEUE プロシージャを使用して保護キューを作成し、キュー所有者ではなく queue_user パラメータで指定していないユーザーに、キューに対する操作の実行を許可する必要がある場合は、そのユーザーを手動でキューの保護キュー・ユーザーとして構成できます。また、SET_UP_QUEUE プロシージャを再実行して、キューに異なる queue_user を指定することもできます。この場合、SET_UP_QUEUE はキューの作成をスキップしますが、queue_user で指定したユーザーがキューの保護キュー・ユーザーとして構成されます。

取得プロセスまたは適用プロセスを削除しても、これらのプロセスの保護キュー・ユーザーとして構成されていたユーザーは、引き続きキューの保護キュー・ユーザーのままです。保護キュー・ユーザーとしてのユーザーを削除するには、ユーザーごとに DBMS_AQADM パッケージの DISABLE_DB_ACCESS プロシージャを実行します。また、不要になったエージェントは削除できます。

関連項目：

- 13-3 ページ「保護キューでのユーザー操作の有効化」
- 13-5 ページ「保護キューでのユーザー操作の無効化」
- AQ エージェントと DBMS_AQADM パッケージの使用方法の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

トランザクション型のキューと非トランザクション型のキュー

トランザクション型のキューは、複数のユーザー・エンキュー・イベントを 1 つのトランザクションとして適用されるセットにグループ化できるキューです。つまり、適用プロセスは、グループ内のユーザー・エンキュー・イベントをすべて適用した後に COMMIT を実行します。DBMS_STREMS_ADM パッケージの SET_UP_QUEUE プロシージャでは、常にトランザクション型のキューが作成されます。

非トランザクション型のキューは、各ユーザー・エンキュー・イベントが個別のトランザクションであるキューです。つまり、適用プロセスは、適用する各ユーザー・エンキュー・イベントの後に COMMIT を実行します。どちらの場合も、ユーザー・エンキュー・イベントには、ユーザーが作成した LCR が含まれている場合と、含まれていない場合があります。

トランザクション型のキューと非トランザクション型のキューの違いが重要になるのは、ユーザー・エンキュー・イベントの場合のみです。適用プロセスは、常にソース・データベースで実行されるトランザクションを保つトランザクション内で取得イベントを適用します。表 3-2 に、イベントの型とキューの型ごとに、適用プロセスの動作を示します。

表 3-2 トランザクション型のキューと非トランザクション型のキューに対する適用プロセスの動作

イベント型	トランザクション型のキュー	非トランザクション型のキュー
取得イベント	適用プロセスはオリジナルのトランザクションを保ちます。	適用プロセスはオリジナルのトランザクションを保ちます。
ユーザー・エンキュー・イベント	ユーザーが指定したユーザー・エンキュー・イベントのグループを、1 つのトランザクションとして適用します。	各ユーザー・エンキュー・イベントをそれぞれのトランザクション内で適用します。

関連項目：

- 13-2 ページ「Streams のキューの管理」
- メッセージのグループ化の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

伝播用の Streams データ・ディクショナリ

取得プロセスが作成されると、Streams データ・ディクショナリと呼ばれる複製データ・ディクショナリが自動的に移入されます。Streams データ・ディクショナリは、ソース・データベースにある 1 次データ・ディクショナリ内の一部の情報のマルチバージョン・コピーです。Streams データ・ディクショナリでは、取得プロセスによってルールが評価されて LCR が作成されると、ソース・データベースからのオブジェクト番号、オブジェクトのバージョン情報および内部の列番号が、表名、列名および列のデータ型にマップされます。このマッピングによって、イベントには名前ではなく番号を格納できるため、各取得イベントが最小サイズに保たれます。

ソース・データベースから取得イベントを伝播するデータベースでルールを評価するために、ソース・データベースにある Streams データ・ディクショナリ内のマッピング情報が必要になる場合があります。このマッピング情報を伝播で使用できるように、Oracle は Streams の伝播がある各サイトでマルチバージョンの Streams データ・ディクショナリを自動的に移入します。ソース・データベースにある Streams データ・ディクショナリからの関連情報を含む内部メッセージは、ソース・データベースから取得イベントを受信する他のすべてのデータベースに自動的に送信されます。

キュー内のこれらの内部メッセージに含まれている Streams データ・ディクショナリ情報は、伝播によって伝播される場合と、伝播されない場合があります。どの Streams データ・ディクショナリ情報が伝播されるかは、伝播のルール・セットによって決定されます。伝播が表の Streams データ・ディクショナリ情報を検出すると、ソース・データベース名、表名および表所有者などの特定の情報を使用して、その伝播のルール・セットが評価されます。

ルール・セットで少なくとも 1 つのルールが `TRUE` に評価されるか（`true_rules`）、さらに情報を与えることによって `TRUE` に評価される可能性がある場合（`maybe_rules`）は、Streams データ・ディクショナリ情報が伝播されます。このルールは、DML ルールでも DDL ルールでもかまいません。

Streams データ・ディクショナリ情報が宛先キューに伝播されると、宛先キューにエンキューされるのみでなく、宛先キューを含むデータベースの Streams データ・ディクショナリにも取り込まれます。したがって、有向ネットワーク構成で宛先キューを読み込む伝播は、Streams データ・ディクショナリの移入を待たずに、LCR を即時に転送できます。

関連項目：

- 2-20 ページ「取得プロセス作成中のデータ・ディクショナリの複製」
- 第 6 章「Streams でのルールの使用方法」

4

Streams 適用プロセス

この章では、Streams の適用プロセスの概念とアーキテクチャについて説明します。

この章の内容は次のとおりです。

関連項目： 第 14 章「適用プロセスの管理」

適用プロセスの概要

適用プロセスはオプションの Oracle バックグラウンド・プロセスであり、特定のキューから論理変更レコード（LCR）とユーザー・メッセージをデキューして、それぞれを直接適用するか、ユーザー定義プロシージャにパラメータとして渡します。適用プロセスによってデキューされる LCR には、適用プロセスが接続先データベース内のデータベース・オブジェクトに適用できるデータ操作言語（DML）変更またはデータ定義言語（DDL）変更が含まれています。適用プロセスによってデキューされるユーザー定義メッセージは SYS.AnyData 型であり、ユーザー作成の LCR など、どのようなユーザー・メッセージでも含めることができます。

適用プロセスで適用されるイベントは、**適用ユーザー**により適用されます。適用ユーザーとは、すべての DML 文と DDL 文を適用し、ユーザー定義の適用ハンドラを実行するユーザーです。

注意： 適用プロセスを関連付けることができるのは SYS.AnyData キューのみで、型付きのキューに関連付けることはできません。

適用ルール

適用プロセスでは、定義したルールに基づいて変更が適用されます。各ルールでは、適用プロセスが変更を適用するデータベース・オブジェクトと、適用する変更のタイプを指定します。適用ルールは次のレベルで指定できます。

- 表ルールは、特定の表に DML 変更または DDL 変更を適用します。サブセット・ルールは、特定の表に対する変更のサブセットを含む表ルールです。
- スキーマ・ルールは、特定のスキーマ内のデータベース・オブジェクトに DML または DDL 変更を適用します。
- グローバル・ルールは、適用プロセスに関連付けられたキュー内のすべての DML 変更または DDL 変更を適用します。

非 LCR イベントの場合は、適用プロセスの動作を制御する独自のルールを作成できます。

関連項目：

- 第 5 章「ルール」
- 第 6 章「Streams でのルールの使用方法」

適用プロセスによるイベント処理

適用プロセスは、キュー内のイベントを処理するための柔軟なメカニズムです。環境に合せて 1 つ以上の適用プロセスを構成する場合には、考慮を必要とするオプションがあります。この項では、適用プロセスで適用できるイベントの型と適用方法について説明します。

適用プロセスによる取得イベントとユーザー・エンキュー・イベントの処理

1 つの適用プロセスで、取得イベントまたはユーザー・エンキュー・イベントを適用できます。両方を適用することはできません。接続先データベースのキューに取得イベントとユーザー・エンキュー・イベントの両方が含まれている場合、接続先データベースにはイベントを処理する適用プロセスが少なくとも 2 つ必要です。

DBMS_STREAMS_ADM パッケージのプロシージャを使用して適用プロセスを作成した場合、その適用プロセスは取得イベントにのみ適用されます。DBMS_APPLY_ADM パッケージの CREATE_APPLY プロシージャを使用して適用プロセスを作成する場合は、apply_captured パラメータを使用して、取得イベントとユーザー・エンキュー・イベントのどちらを適用するかを指定します。このプロシージャで作成された適用プロセスの場合、apply_captured パラメータはデフォルトで false に設定されます。

イベントが発生したデータベースは、取得イベントの適用プロセスには重要ですが、ユーザー・エンキュー・イベントの適用プロセスには重要ではありません。取得イベントの場合、ソース・データベースは、REDO ログで変更が生成されたデータベースです。適用プロセスでは、1 つのソース・データベースの変更のみが適用されるように、取得された各 LCR のソース・データベースが決定される必要があります。データベース管理者は、適用プロセスで、このソース・データベースにある取得プロセスの 1 つのみから変更が適用されることを確認する必要があります。ユーザー・エンキュー・イベントの場合、イベントの発生元データベースに関する情報は、イベントがユーザー・エンキュー LCR の場合でも適用プロセスで無視されます。1 つの適用プロセスにおいて、異なるデータベースで発生したユーザー・エンキュー・イベントが適用可能です。

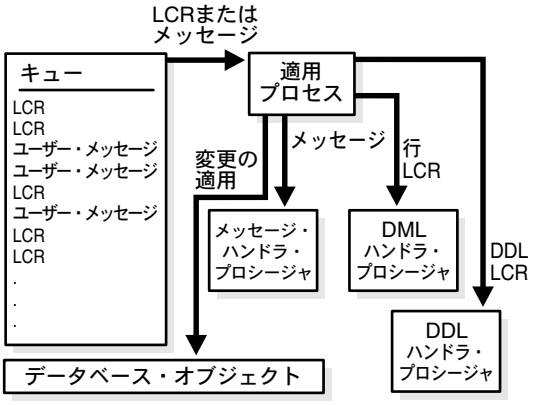
関連項目：

- 取得イベントとユーザー・エンキュー・イベントの詳細は、3-2 ページの「イベントのステージングと伝播の概要」を参照してください。
- CREATE_APPLY プロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

イベント処理オプション

イベント処理オプションは、適用プロセスが受信するイベントが LCR イベントであるかどうかに応じて異なります。図 4-1 に、適用プロセスのイベント処理オプションを示します。

図 4-1 適用プロセス



LCR イベントの処理

各適用プロセスで適用できるのは、1 つのソース・データベースからの取得イベントのみです。これは、これらのイベント内の LCR を処理するには、ソース・データベースで依存性、意味のあるトランザクションの順序付けおよびトランザクションの境界を認識する必要があるためです。複数のデータベースから取得された LCR を、単一の宛先キューに送信できます。ただし、単一のキューに複数のデータベースから取得された LCR が含まれている場合は、これらの LCR を取り出す適用プロセスが複数必要です。これらの各適用プロセスは、正確に 1 つのソース・データベースから取得された LCR のみを受信するように、ルールを使用して構成する必要があります。LCR を含むユーザー・エンキュー・イベント（取得イベントではなく）の場合、複数のソース・データベースからのユーザー・エンキュー・イベントであっても、1 つの適用プロセスで適用可能です。

また、各適用プロセスで適用できるのは、1 つの取得プロセスからの取得イベントのみです。ソース・データベース上で複数の取得プロセスが実行されていて、接続先データベースで複数の取得プロセスからの LCR が適用される場合は、各取得プロセスからの変更を適用するために 1 つの適用プロセスが必要です。このような環境では、取得プロセスまたは適用プロセスで使用される各 Streams キューに、特定のソース・データベースの最大 1 つの取得プロセスからの取得イベントを入れることをお薦めします。各取得プロセスが異なるソース・データベースで変更を取得している場合は、キューに複数の取得プロセスからの LCR を含めることができます。

適用プロセスを構成するときに、LCR を含む取得イベントまたはユーザー・エンキュー・イベントの処理方法を次から指定できます。LCR イベントを直接適用する方法と、LCR イベントをユーザー・プロシージャにパラメータとして渡して処理させる方法があります。ここでは、これらのオプションについて説明します。

LCR イベントを直接適用する場合 このオプションを使用すると、適用プロセスではユーザー・プロシージャを実行せずに LCR イベントが適用されます。適用プロセスは、データベース・オブジェクトに LCR 内の変更を正常に適用します。また、競合または適用エラーが発生した場合は、競合ハンドラまたはエラー・ハンドラと呼ばれるユーザー指定プロシージャを使用してエラーを解決しようとします。

競合ハンドラが競合を解消できる場合は、LCR が適用されるか、LCR 内の変更が廃棄されます。エラー・ハンドラがエラーを解決できる場合は、必要に応じて LCR を適用する必要があります。エラー・ハンドラは、適用前に LCR を変更することでエラーを解決できる場合があります。エラー・ハンドラがエラーを解決できない場合、適用プロセスはトランザクションとそれに関連付けられたすべての LCR を例外キューに入れます。

ユーザー・プロシージャをコールして LCR イベントを処理する場合 このオプションを使用すると、適用プロセスは LCR イベントをユーザー・プロシージャにパラメータとして渡して処理させます。ユーザー・プロシージャは、LCR イベントをカスタマイズされた方法で処理できます。

DML 文によって生じた行 LCR を処理するユーザー・プロシージャは DML ハンドラと呼ばれ、DDL 文によって生じた DDL LCR を処理するユーザー・プロシージャは DDL ハンドラと呼ばれます。適用プロセスでは多数の DML ハンドラを使用できますが、DDL ハンドラは 1 つのみで、適用プロセスによってデキューされるすべての DDL LCR を処理します。

適用プロセスに関連付けられている表ごとに、行 LCR 内の次のタイプの各操作を処理するように個別の DML ハンドラを設定できます。

- `INSERT`
- `UPDATE`
- `DELETE`
- `LOB_UPDATE`

たとえば、`hr.employees` 表には、`INSERT` 操作を処理する DML ハンドラが 1 つと、`UPDATE` 操作を処理する別の DML ハンドラがあるとします。

ユーザー・プロシージャは、LCR のカスタマイズされた処理に使用できます。たとえば、ソース・データベースで特定の表に対する個々の挿入によって、接続先データベースで複数の表に挿入する必要がある場合は、そのために表に対する `INSERT` 操作を処理するユーザー・プロシージャを作成できます。また、DDL 変更を適用前にログに記録する場合は、そのために DDL 操作を処理するユーザー・プロシージャを作成できます。

DML ハンドラでは、ユーザー・プロシージャによって設定された名前付きセーブポイントが対象の場合を除き、コミットもロールバックも行いません。DDL ハンドラ内で DDL を実行するには、LCR 用の EXECUTE メンバー・プロシージャを起動します。

DML ハンドラを設定するには、DBMS_APPLY_ADM パッケージの SET_DML_HANDLER プロシージャを使用します。この設定がデータベース内のすべての適用プロセスで使用されます。DDL ハンドラを特定の適用プロセスに関連付けるには、DBMS_APPLY_ADM パッケージの CREATE_APPLY または ALTER_APPLY プロシージャで ddl_handler パラメータを使用します。

エラー・ハンドラは DML ハンドラの場合と同じ方法で作成しますが、SET_DML_HANDLER プロシージャを実行する際に、error_handler パラメータを true に設定する点は異なります。このように設定すると、ハンドラが起動するのは、適用プロセスで指定の表において指定の操作が行われて、行 LCR の適用が試行されたときに、適用エラーが発生した場合のみとなります。

注意： SET_DML_HANDLER プロシージャを実行する場合、ハンドラが使用されるオブジェクトを指定すると、指定オブジェクトがローカルの接続先データベースに存在するかどうかが Oracle によりチェックされます。オブジェクトが存在しない場合、エラーが発生します。したがって、ソース・データベースと接続先データベースでオブジェクト名が異なる場合、行 LCR が適用される前に、ルールベースの変換を使用して行 LCR のオブジェクト名を変換してください。

関連項目：

- 行 LCR と DDL LCR の詳細は、2-2 ページの「論理変更レコード（LCR）」を参照してください。
- LCR 型用の EXECUTE メンバー・プロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 6-23 ページ「ルールベースの変換」

LCR 以外のユーザー・メッセージの処理

LCR を含まないユーザー・エンキュー・イベントは、適用プロセスのルール・セット内の少なくとも 1 つのルールを満たしている場合、適用プロセス用に指定されたメッセージ・ハンドラにより処理されます。メッセージ・ハンドラとは、環境に合せてカスタマイズされた方法で LCR 以外のユーザー・メッセージを処理できるユーザー定義プロシージャです。

メッセージ・ハンドラには、1 つ以上のリモート・データベースの更新や他のリモート・アクションを実行する必要のあるアプリケーションを含む環境で使用すると利点があります。これらのアプリケーションでユーザー・メッセージをローカル・データベースのキューにエンキューし、Streams で各ユーザー・メッセージを接続先データベースの適切なキューに伝播できます。複数の接続先がある場合、Streams はこれらのメッセージを自動的に各接続先に伝播して処理するためのインフラストラクチャを提供します。接続先が 1 つしかない場合も、Streams はソース・データベースのアプリケーションと接続先データベースのアプリケーションの間にレイヤーを提供します。これにより、リモート・データベースのアプリケーションが使用不可能になった場合も、ソース・データベースのアプリケーションは引き続き正常に動作できます。

たとえば、メッセージ・ハンドラは、ユーザー・メッセージを電子メール・メッセージとしてフォーマットできます。この場合、ユーザー・メッセージには、`from`、`to`、`subject`、`text_of_message` など、電子メール・メッセージで使用される属性が含まれます。メッセージ・ハンドラは、これらのユーザー・メッセージを電子メール・メッセージに変換し、電子メール・ゲートウェイを介して送信できます。

適用プロセス用のメッセージ・ハンドラを指定するには、`DBMS_APPLY_ADM` パッケージの `CREATE_APPLY` または `ALTER_APPLY` プロシージャで `message_handler` パラメータを使用します。Streams の適用プロセスでは常に、非 LCR メッセージにはキュー内の他のイベントに対する依存性がないものとみなされます。したがって、環境内でこれらのメッセージ間に依存性が存在する場合は、適用プロセスの並列性を 1 に設定することをお薦めします。

イベント処理オプションのまとめ

表 4-1 に、前項で説明した 1 つ以上のイベント・ハンドラを使用する場合に使用可能なイベント処理オプションのまとめを示します。行 LCR と DDL LCR は適用プロセスで直接適用できるため、イベント・ハンドラはオプションです。ただし、LCR 以外のユーザー・メッセージを処理するには、メッセージ・ハンドラが必須です。また、イベントが適用プロセスのルール・セット内の少なくとも 1 つのルールを満たしている場合にのみ、適用プロセスでこのイベントがデキューされます。

表 4-1 イベント処理オプションのまとめ

イベントの型	適用プロセスのデフォルト動作	ユーザー・プロシージャ	ユーザー・プロシージャの有効範囲
行 LCR	DML を実行	DML ハンドラまたはエラー・ハンドラ	1 つの表に対する 1 つの操作
DDL LCR	DDL を実行	DDL ハンドラ	適用プロセス全体
LCR 以外のユーザー・メッセージ	エラー・トランザクションを作成（メッセージ・ハンドラが存在しない場合）	メッセージ・ハンドラ	適用プロセス全体

注意：

- 適用ハンドラは、LCR の EXECUTE メンバー・プロシージャをコールして LCR を実行できます。
- 適用された DDL LCR は、すべて自動的にコミットされます。したがって、DDL ハンドラが DDL LCR の EXECUTE メンバー・プロシージャをコールすると、自動的にコミットが実行されます。
- 必要な場合、適用ハンドラは Streams セッション・タグを設定できます。

関連項目：

- LCR 型用の EXECUTE メンバー・プロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 第 8 章「Streams のタグ」

適用されるデータ型

表にデータ操作言語（DML）変更の行 LCR を適用する場合、適用プロセスでは次のデータ型の列に対する変更が適用されます。

- CHAR
- VARCHAR2
- NCHAR
- NVARCHAR2
- NUMBER
- DATE
- CLOB
- BLOB
- RAW
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

適用プロセスは、データ型 NCLOB、LONG、LONG RAW、BFILE、ROWID、UROWID およびユーザー定義型（オブジェクト型、REF、VARRAY および NESTED TABLE など）の列における DML 変更は適用しません。サポートされていないデータ型の列に関する情報を含む行 LCR を適用しようとすると、適用プロセスにエラーが発生します。その場合、適用プロセスでは LCR を含むトランザクションが例外キューに移動されます。

関連項目：

- 2-6 ページ「取得されるデータ型」
- これらのデータ型の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

表に DML 変更を適用する際の考慮事項

ここでは、表に DML 変更を適用する際の考慮事項について説明します。

制約

接続先データベースの主キー列が、更新のたびにソース・データベースの REDO ログに確実に記録されるようにする必要があります。ソース・データベースの複数列のデータが含まれる接続先データベースの一意制約または外部キー制約には、ソース・データベースでの追加ロギングが必要です。

列をソース・データベースのログに確実に記録させるには、様々な方法があります。たとえば、列の値が更新されるたびに、その列がログに記録されます。また、Oracle には、指定した列のロギングを自動化するサプリメンタル・ロギングと呼ばれる機能があります。

ソース・データベースの単一列のデータのみが含まれる接続先データベースの一意キー制約と外部キー制約については、サプリメンタル・ロギングは不要です。ただし、ソース・データベースの複数列のデータが含まれる制約については、接続先データベースの制約で使用されるソース・データベースのすべての列が含まれる、条件付きのサプリメンタル・ログ・グループを作成する必要があります。

通常、一意キー制約と外部キー制約には、ソース・データベースおよび接続先データベースで同じ列が含まれます。ただし、適用ハンドラまたはルールベースの変換により、ソース・データベースの複数列制約が接続先データベースの単一キー列に結合される場合があります。また、適用ハンドラまたはルールベースの変換により、ソース・データベースの単一キー列が接続先データベースの複数列制約に分割される場合があります。このような場合は、ソース・データベースの制約内の列数により、条件付きのサプリメンタル・ログ・グループが必要かどうかが決定されます。ソース・データベースの制約に複数の列が存在する場合、ソース・データベースに、すべての制約列が含まれる条件付きのサプリメンタル・ログ・グループが必要です。ソース・データベースの制約に 1 列のみが存在する場合、キー列にサプリメンタル・ロギングは不要です。

関連項目： 2-10 ページ「Streams 環境内のサプリメンタル・ロギング」

代替キー列

可能な場合は、適用プロセスによって変更が適用される各表に主キーを用意する必要があります。主キーを使用できない場合は、各表に各行の一意識別子として使用できる列セットを含めることをお薦めします。Streams 環境で使用する予定の表に主キーまたは一意列セットがない場合は、それに応じてこれらの表を変更することを考慮してください。

競合を検出してエラーを正確に処理するためには、Oracle は異なるデータベースで対応する行を一意に識別して合致させることができる必要があります。デフォルトでは、Streams は表の主キーを使用してその表の各行を識別します。接続先データベースの表に主キーがない場合や、キーとして主キー以外の列を使用する必要がある場合は、接続先データベースで代替キーを指定できます。代替キーとは、Oracle が適用中に表の各行を識別するために使用できる列または列セットです。

表の代替主キーを指定するには、DBMS_APPLY_ADM パッケージの SET_KEY_COLUMNS プロシージャを使用します。主キーとは異なり、代替キー列には NULL を指定できます。また、代替キー列は、接続先データベースですべての適用プロセスについて、指定された表の既存の主キーよりも優先されます。

接続先データベースで表の代替キーを指定する場合に、これらの列がソース・データベースで同じ表の主キーでなければ、ソース・データベースで代替キー列を含む無条件のサプリメンタル・ログ・グループを作成する必要があります。

代替キー列および主キー制約がどちらもない場合、適用プロセスでは、LOB 列および LONG 列を除く表内のすべての列がキー列として使用されます。この場合、ソース・データベースでこれらの列を含む無条件のサプリメンタル・ログ・グループを作成する必要があります。行 LCR では必要な列が少ないため、表の主キー制約がない場合は代替キー列を使用することをお薦めします。

注意：

- 列ごとに、代替キー列として NOT NULL 列を指定することをお薦めします。また、代替キーにすべての列を含む 1 つの索引を作成する必要があります。これらのガイドラインに従うと、データベースでは関連する行を効率的に検索できるため、LOB の更新、削除およびピース単位更新のパフォーマンスが改善されます。
- アプリケーションには、表の主キー列または代替キー列の更新を許可しないでください。これにより、データベースでは行を識別してデータの整合性を保つことができます。

関連項目：

- 『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_APPLY_ADM.SET_KEY_COLUMNS プロシージャの項を参照してください。
- 2-10 ページ「Streams 環境内のサプリメンタル・ロギング」

Streams ルールを使用した行のサブセット化

DBMS_STREAMS_ADM パッケージの ADD_SUBSET_RULES プロシージャを使用すると、特定の表の各行のサブセットをメンテナンスできます。そのためには、このプロシージャの dml_condition パラメータで、SELECT 文の WHERE 句の条件と同様の条件を指定します。特定の表について、特定の適用プロセスに許されるサブセット・ルールは 1 つのみです。

注意： 1 つ以上の LOB 列を含む表に対するサブセット・ルールの作成は、サポートされていません。

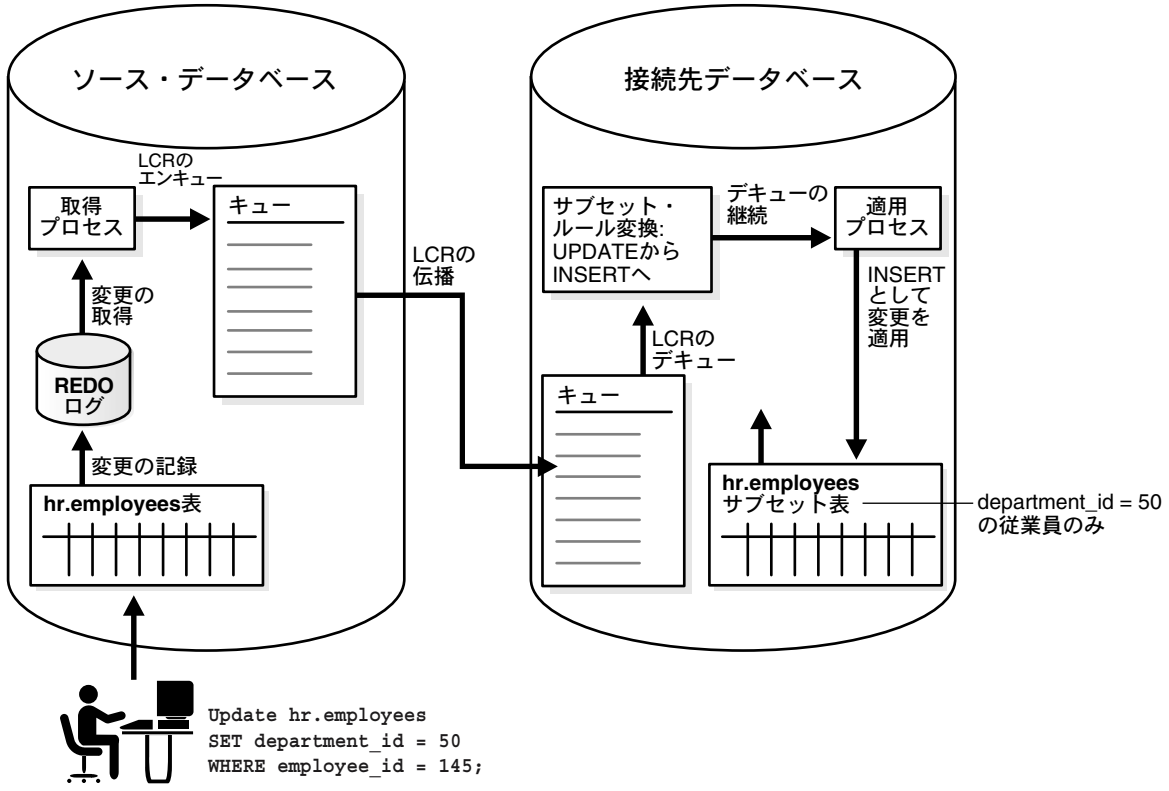
関連項目： 6-6 ページ「表ルールとサブセット・ルール」

行の移行

サブセット・ルールを使用すると、取得された更新操作を、適用プロセスによって適用されるときに挿入操作または削除操作に変換されることがあります。この自動変換は**行の移行**と呼ばれ、ルールのアクション・コンテキストで指定した内部の LCR 変換によって実行されます。

たとえば、サブセット・ルールを使用してサブセット条件 department_id = 50 を指定し、従業員の department_id が 50 の場合に適用プロセスで変更を hr.employees 表にのみ適用するように指定する場合を考えます。接続先データベースの表が、department_id が 50 である従業員のレコードのみが含まれるサブセット表であると想定します。ソース・データベースが従業員に対する変更を取得し、それが従業員の department_id を 80 から 50 に変更するというものであれば、接続先データベースにサブセット・ルールを持つ適用プロセスは、更新操作を挿入操作に変換してこの変更を適用します。この変換が必要となるのは、従業員の行が宛先の表に存在しないためです。

図 4-2 サブセット・ルールの変換例



同様に、取得された更新によって従業員の department_id が 50 から 20 に変更される場合、このサブセット・ルールを持つ適用プロセスは更新操作を削除操作に変換します。

適用プロセスが変更を表に適用するときに行の移行を実行できる場合に、表に対するローカル変更を許可していると、適用プロセスは表のすべての行がサブセットの条件を満たしているかどうかを確認できません。たとえば、hr.employees 表の条件が department_id = 50 であるとします。ユーザーまたはアプリケーションが department_id として 30 を持つ従業員の行を挿入すると、この行は表に残り、適用プロセスでは削除されません。同様に、ユーザーまたはアプリケーションが 1 行をローカルに更新して department_id を 30 に変更した場合、この行も表に残ります。この種のエラーを回避するために、サブセットの表に対して実行されるすべての DML がサブセットの条件を満たすことを確認するようにお薦めします。

サプリメンタル・ロギングと行のサブセット化

接続先データベースで表のサブセット・ルールを指定する場合は、接続先データベースにある表のすべての列と、サブセット条件に含まれるすべての列について、ソース・データベースで無条件のサプリメンタル・ログ・グループを指定する必要があります。場合によっては、サブセット・ルールを指定すると、適用プロセスによって更新が挿入に変換される場合があり、この場合は一部またはすべての列に補足情報が必要になることがあります。

たとえば、データベース dbs2.net 上で hr.locations 表の postal_code 列に対するサブセット・ルールを指定し、さらにこの表に対する変更のソース・データベースが dbs1.net の場合は、dbs2.net の hr.locations 表に存在するすべての列、および postal_code 列（この列が接続先データベースの表に存在しない場合も含む）について、dbs1.net のサプリメンタル・ロギングを指定します。

関連項目： 2-10 ページ「Streams 環境内のサプリメンタル・ロギング」

列の不一致に対する適用プロセスの動作

列の不一致とは、ソース・データベースの表の列と接続先データベースの同じ表の列が一致しないことです。Streams 環境に列の不一致がある場合は、ルールベースの変換または DML ハンドラを使用して、適用プロセスで適用される行 LCR の列を接続先データベースにある関連表の列と一致させます。ここでは、一般的な列の不一致に対する適用プロセスの動作について説明します。

関連項目：

- 6-23 ページ「ルールベースの変換」および 15-11 ページ「ルールベースの変換の管理」
- 適用プロセスのハンドラの詳細は、4-4 ページの「LCR イベントの処理」を参照してください。
- LCR の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

接続先データベースの列の欠落

ソース・データベースの表にある 1 つ以上の列が接続先データベースの表から欠落していると、適用プロセスにエラーが発生し、エラーの原因となったトランザクションが例外キューに移動します。この種のエラーは、適用前に LCR から欠落している列を排除するルールベースの変換または DML ハンドラを作成することで回避できます。特に、変換またはハンドラでは、行 LCR に DELETE_COLUMN メンバー・プロシージャを使用して、余分な列を削除できます。

接続先データベースの余分な列

接続先データベースの表の列数がソース・データベースの表よりも多い場合、適用プロセスの動作は余分な列が依存性の計算に必要かどうかに応じて異なります。余分な列が依存性の計算に使用されない場合、適用プロセスでは変更が宛先の表に適用されます。この場合、接続先データベースに余分な列に関する列のデフォルトが存在すると、すべての挿入でこれらの列にはデフォルトが使用されます。それ以外の場合、挿入されるこれらの列は NULL になります。

ただし、余分な列が依存性の計算に使用される場合、適用プロセスではこれらの変更を含むトランザクションが例外キューに置かれます。依存性の計算には、次のタイプの列が必要です。

- どのような変更の場合も、すべてのキー列。
- INSERT および DELETE 文の場合は、制約に関係するすべての列。
- UPDATE 文の場合、一意キー制約列や外部キー制約列などの制約列に変更がある場合は、その制約に関係するすべての列。

列のデータ型の不一致

接続先データベースにある表の列のデータ型が、ソース・データベースにある同じ列のデータ型と一致しない場合、適用プロセスでは一致しない列の変更を含むトランザクションが例外キューに置かれます。この種のエラーを回避するために、データ型を変換するルールベースの変換または DML ハンドラを作成できます。

競合解消と適用プロセス

複数のデータベース間でデータが共有される Streams 構成では、競合が発生する可能性があります。変更の取得対象となる表とこれらの変更が適用される表について、DML 変更が許可されている場合は、競合が発生する可能性があります。

たとえば、ソース・データベースのトランザクションによって行が更新されるのとほぼ同時に、それと同じ行が接続先データベースの異なるトランザクションによって更新されることがあります。この場合、2 つのデータベース間でデータ整合性が重要であれば、変更が接続先データベースに伝播するときに、適用プロセスに対して、接続先データベースで変更を保持するか、ソース・データベースからの変更で置換するように指示する必要があります。データ競合が発生した場合は、競合をビジネス・ルールに従って確実に解消するメカニズムが必要です。

Streams は競合を自動的に検出し、更新の競合の場合は、更新の競合ハンドラが構成されていれば、それを使用して解消しようとします。Streams には様々なビルトイン・ハンドラが用意されており、ビジネス・ルールに従って競合を解消するデータベース用の競合解消システムを定義できます。ビルトイン競合解消ハンドラで解消できない固有の状況がある場合は、独自のカスタム競合解消ハンドラを作成してエラー・ハンドラまたは DML ハンドラ内で使用できます。

関連項目： 第 7 章「Streams 競合解消」

ハンドラと行 LCR の処理

次のどのハンドラでも、行 LCR を処理できます。

- DML ハンドラ
- エラー・ハンドラ
- 更新の競合ハンドラ

ここでは、これらのハンドラに関係する使用例について説明します。

- 関連ハンドラがない場合
- 関連する更新の競合ハンドラがある場合
- DML ハンドラはあるが関連する更新の競合ハンドラがない場合
- DML ハンドラおよび関連する更新の競合ハンドラがある場合
- エラー・ハンドラはあるが関連する更新の競合ハンドラがない場合
- エラー・ハンドラおよび関連する更新の競合ハンドラがある場合

DML ハンドラとエラー・ハンドラを、同じ表に対する同じ操作に同時に使用することはできません。したがって、両方のハンドラが起動される使用例はありません。

関連ハンドラがない場合

行 LCR に関連ハンドラがない場合、適用プロセスは行 LCR で指定された変更を直接適用しようとします。適用プロセスが行 LCR を適用できる場合は、表のその行が変更されます。適用中に競合またはエラーが発生すると、エラーになった行 LCR を含むトランザクションがロールバックされ、適用プロセスのルール・セットを満たすトランザクション内の LCR がすべて例外キューに移動します。

関連する更新の競合ハンドラがある場合

関連する更新の競合ハンドラは構成されているが、他の関連ハンドラが構成されていない場合を考えます。適用プロセスは、行 LCR で指定された変更を直接適用しようとします。適用プロセスが行 LCR を適用できる場合は、表のその行が変更されます。

一意性競合や削除の競合など、更新の競合以外の条件によって適用中にエラーが発生すると、エラーになった行 LCR を含むトランザクションがロールバックされ、適用プロセスのルール・セットを満たすトランザクション内の LCR がすべて例外キューに移動します。

適用中に更新の競合が発生すると、関連する更新の競合ハンドラが起動します。更新の競合ハンドラによって競合が正常に解消されると、適用プロセスは更新の競合の解消方法に応じて LCR を適用または廃棄し、適用プロセスのルール・セットを満たすトランザクション内の他の LCR を引き続き適用します。更新の競合ハンドラが競合を解消できない場合は、エラーになった行 LCR を含むトランザクションがロールバックされ、適用プロセスのルール・セットを満たすトランザクション内の LCR がすべて例外キューに移動します。

DML ハンドラはあるが関連する更新の競合ハンドラがない場合

適用プロセスが行 LCR を DML ハンドラに渡し、関連する更新の競合ハンドラが構成されていない場合を考えます。

DML ハンドラによって行 LCR が処理されます。DML ハンドラのデザイナが、この処理を全面的に制御します。一部の DML ハンドラは、SQL 操作を実行するか、行 LCR の EXECUTE メンバー・プロシージャを実行する場合があります。DML ハンドラが行 LCR の EXECUTE メンバー・プロシージャを実行すると、適用プロセスは行 LCR を適用しようとします。この行 LCR は、DML ハンドラによって変更されている可能性があります。

DML ハンドラによって実行される SQL 操作が失敗するか、EXECUTE メンバー・プロシージャの実行に失敗した場合、DML ハンドラは例外処理を試みることができます。DML ハンドラが例外を呼び出さなければ、適用プロセスは DML ハンドラが行 LCR で適切なアクションを実行したものとみなして、適用プロセスのルール・セットを満たすトランザクション内の他の LCR を引き続き適用します。

DML ハンドラは、例外を処理できなければ例外を呼び出します。この場合は、DML ハンドラが処理した行 LCR を含むトランザクションがロールバックされ、適用プロセスのルール・セットを満たすトランザクション内の LCR がすべて例外キューに移動します。

DML ハンドラおよび関連する更新の競合ハンドラがある場合

適用プロセスが行 LCR を DML ハンドラに渡し、関連する更新の競合ハンドラが構成されている場合を考えます。

DML ハンドラによって行 LCR が処理されます。DML ハンドラのデザイナが、この処理を全面的に制御します。一部の DML ハンドラは、SQL 操作を実行するか、行 LCR の EXECUTE メンバー・プロシージャを実行する場合があります。DML ハンドラが行 LCR の EXECUTE メンバー・プロシージャを実行すると、適用プロセスは行 LCR を適用しようとします。この行 LCR は、DML ハンドラによって変更されている可能性があります。

DML ハンドラによって実行される SQL 操作が失敗するか、更新の競合以外の理由で EXECUTE メンバー・プロシージャの実行に失敗した場合の動作は、4-17 ページの「DML ハンドラはあるが関連する更新の競合ハンドラがない場合」で説明した動作と同じです。一意性競合と削除の競合は、更新の競合ではないことに注意してください。

更新の競合が原因で EXECUTE メンバー・プロシージャの実行に失敗した場合の動作は、このプロシージャの conflict_resolution パラメータの設定に応じて次のように異なります。

conflict_resolution パラメータが true に設定されている場合 conflict_resolution パラメータが true に設定されている場合は、関連する更新の競合ハンドラが起動します。

更新の競合ハンドラによって競合が正常に解消され、DML ハンドラによって実行された他のすべての操作が正常終了すると、DML ハンドラは例外を呼び出さずに終了し、適用プロセスはそのルール・セットを満たすトランザクション内の他の LCR を引き続き適用します。

更新の競合ハンドラが競合を解消できない場合、DML ハンドラは例外の処理を試みることができます。DML ハンドラが例外を呼び出さなければ、適用プロセスは DML ハンドラが行 LCR で適切なアクションを実行したものとみなして、適用プロセスのルール・セットを満たすトランザクション内の他の LCR を引き続き適用します。DML ハンドラは、例外を処理できなければ例外を呼び出します。この場合は、行 LCR を含むトランザクションがロールバックされ、適用プロセスのルール・セットを満たすトランザクション内の LCR がすべて例外キューに移動します。

conflict_resolution パラメータが false に設定されている場合 conflict_resolution パラメータが false に設定されている場合、関連する更新の競合ハンドラは起動しません。この場合の動作は、4-17 ページの「DML ハンドラはあるが関連する更新の競合ハンドラがない場合」で説明した動作と同じです。

エラー・ハンドラはあるが関連する更新の競合ハンドラがない場合

適用プロセスが行 LCR を適用するときにエラーが発生した場合を考えます。このエラーは、競合または他の条件が原因となっている可能性があります。表操作用のエラー・ハンドラはありますが、関連する更新の競合ハンドラは構成されていません。

行 LCR がエラー・ハンドラに渡されます。エラー・ハンドラによって行 LCR が処理されます。エラー・ハンドラのデザイナが、この処理を全面的に制御します。一部のエラー・ハンドラは、SQL 操作を実行するか、行 LCR の EXECUTE メンバー・プロシージャを実行する場合があります。エラー・ハンドラが行 LCR の EXECUTE メンバー・プロシージャを実行すると、適用プロセスは行 LCR を適用しようとします。この行 LCR は、エラー・ハンドラによって変更されている可能性があります。

エラー・ハンドラによって実行される SQL 操作が失敗するか、EXECUTE メンバー・プロシージャの実行に失敗した場合、エラー・ハンドラは例外処理を試みることができます。エラー・ハンドラが例外を呼び出さなければ、適用プロセスはエラー・ハンドラが行 LCR で適切なアクションを実行したものとみなして、適用プロセスのルール・セットを満たすトランザクション内の他の LCR を引き続き適用します。

エラー・ハンドラは、例外を処理できなければ例外を呼び出します。この場合は、DML ハンドラが処理した行 LCR を含むトランザクションがロールバックされ、適用プロセスのルール・セットを満たすトランザクション内の LCR がすべて例外キューに移動します。

エラー・ハンドラおよび関連する更新の競合ハンドラがある場合

適用プロセスが行 LCR を適用するときにエラーが発生した場合を考えます。表操作用のエラー・ハンドラがあり、関連する更新の競合ハンドラが構成されているとします。

エラー処理のために起動されるハンドラは、次のようにエラーのタイプに応じて異なります。

- 一意性競合や削除の競合など、更新の競合以外の条件によってエラーが発生すると、エラー・ハンドラが起動します。この場合の動作は、4-18 ページの「エラー・ハンドラはあるが関連する更新の競合ハンドラがない場合」で説明した動作と同じです。
- 更新の競合が原因でエラーが発生すると、更新の競合ハンドラが起動します。更新の競合ハンドラによって競合が正常に解消されると、適用プロセスはトランザクションのうち適用プロセスのルール・セットを満たす他の LCR を引き続き適用します。この場合、エラー・ハンドラは起動されません。

  更新の競合ハンドラが競合を解消できない場合は、エラー・ハンドラが起動します。エラー・ハンドラが例外を呼び出さなければ、適用プロセスはエラー・ハンドラが行 LCR で適切なアクションを実行したものとみなして、適用プロセスのルール・セットを満たすトランザクション内の他の LCR を引き続き適用します。エラー・ハンドラは、LCR を処理できなければ例外を呼び出します。この場合は、DML ハンドラが処理した行 LCR を含むトランザクションがロールバックされ、適用プロセスのルール・セットを満たすトランザクション内の LCR がすべて例外キューに移動します。

  関連項目：

  - 行 LCR 用の `EXECUTE` メンバー・プロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
  - 14-13 ページ「DML ハンドラの管理」
  - 14-20 ページ「エラー・ハンドラの管理」
  - 14-28 ページ「Streams の競合解消の管理」

DDL 変更の適用に関する考慮事項

ここでは、表に DDL 変更を適用する際の考慮事項について説明します。

- 適用プロセスによって無視される DDL 変更のタイプ
- Streams 環境内のデータベース構造
- 接続先データベースに必須の現行スキーマ・ユーザー
- システム生成名
- CREATE TABLE AS SELECT 文

関連項目： 2-8 ページ「取得プロセスによって無視される DDL 変更のタイプ」

適用プロセスによって無視される DDL 変更のタイプ

適用プロセスでは、次のタイプの DDL 変更はサポートされません。これらのタイプの DDL 変更は適用されません。

- ALTER MATERIALIZED VIEW
- ALTER MATERIALIZED VIEW LOG
- CREATE DATABASE LINK
- CREATE SCHEMA AUTHORIZATION
- CREATE MATERIALIZED VIEW
- CREATE MATERIALIZED VIEW LOG
- DROP DATABASE LINK
- DROP MATERIALIZED VIEW
- DROP MATERIALIZED VIEW LOG
- RENAME

また、次のタイプの CREATE TABLE および ALTER TABLE 文は、適用プロセスでは無視されます。

- Streams 環境では、クラスタ化表に対する CREATE TABLE AS SELECT 文はサポートされません。
- 索引構成表を作成する CREATE TABLE 文。
- 索引構成表を変更する ALTER TABLE 文。

適用プロセスは、適用できない操作を指定する DDL LCR を受信すると、DDL LCR を無視して、次のメッセージと無視された DDL テキストを適用プロセスのトレース・ファイルに記録します。

```
Apply process ignored the following DDL:
```

変更を含む DDL LCR が適用プロセスのルール・セットにあるルールを満たしている場合、適用プロセスでは他のすべてのタイプの DDL 変更が適用されます。また、適用プロセスでは、有効なユーザー・エンキュー DDL LCR が適用可能です。

注意：

- 適用プロセスでは、ALTER TABLE jobs RENAME などの ALTER *object_type object_name* RENAME 変更が適用されます。したがって、オブジェクト名を変更する DDL 変更を適用する必要がある場合は、RENAME 文のかわりに ALTER *object_type object_name* RENAME 文を使用してください。
- マテリアライズド・ビューは、スナップショットのシノニムです。マテリアライズド・ビュー上の文に等価のスナップショットは、適用プロセスでは無視されます。

Streams 環境内のデータベース構造

取得された DDL 変更を接続先データベースで正常に適用するには、接続先データベースの構造をソース・データベースと同じにする必要があります。または、DDL 文で異なるデータベース構造情報を指定しないでください。データベース構造には、データ・ファイル、表領域、ロールバック・セグメントと、データベース・オブジェクトをサポートする他の物理構造と論理構造が含まれます。

たとえば、表に対する取得された DDL 変更を接続先データベースで正常に適用するには、次の条件を満たす必要があります。

- ソース・データベースと接続先データベースで、CREATE TABLE 文に同じ記憶域パラメータを指定する必要があります。
- DDL 文で特定の表領域またはロールバック・セグメントを参照する場合は、ソース・データベースと接続先データベースで、その表領域またはロールバック・セグメントに同じ名前と互換仕様を使用する必要があります。

  ただし、DDL 文で表領域とロールバック・セグメントが指定されていなければ、デフォルトの表領域とロールバック・セグメントが使用されます。この場合の表領域やロールバック・セグメントは、ソース・データベースと接続先データベースで異なっていてもかまいません。
- ソース・データベースと接続先データベースで、同じパーティション化仕様を使用する必要があります。

接続先データベースに必須の現行スキーマ・ユーザー

接続先データベースで DDL LCR を正常に適用するには、DDL LCR で current_schema として指定されたユーザーが接続先データベースに存在する必要があります。現行のスキーマは、DDL テキストでオブジェクトに対するスキーマが指定されていない場合に使用されるスキーマです。

関連項目：

- データベース構造の詳細は、『Oracle9*i* データベース概要』を参照してください。
- DDL LCR の current_schema 属性の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

システム生成名

ソース・データベースで DDL 変更を取得して、それを接続先データベースで適用する予定の場合は、システム生成名を使用しないでください。DDL 文の結果としてオブジェクト名がシステムによって生成される場合、通常、オブジェクト名はソース・データベースとそこからの DDL 変更を適用する各接続先データベースで異なるものになります。オブジェクト名が異なると、将来の DDL 変更に適用エラーが発生する可能性があります。

たとえば、ソース・データベースで次の DDL 文を実行する場合を考えます。

```
CREATE TABLE sys_gen_name (n1 NUMBER  NOT NULL);
```

この文の結果、システム生成名を持つ NOT NULL 制約が作成されます。たとえば、NOT NULL 制約の名前が sys_001500 であるとします。この変更が接続先データベースで適用されると、この制約のシステム生成名が sys_c1000 となる場合があります。

ソース・データベースで次の DDL 文を実行する場合を考えます。

```
ALTER TABLE sys_gen_name DROP CONSTRAINT sys_001500;
```

この DDL 文は、ソース・データベースでは成功しますが、接続先データベースでは失敗し、適用エラーになります。

この種のエラーを回避するには、DDL 文によって作成されるすべてのオブジェクトの名前を明示的に指定します。たとえば、NOT NULL 制約の名前を明示的に指定するには、次の DDL 文を実行します。

```
CREATE TABLE sys_gen_name (n1 NUMBER CONSTRAINT sys_gen_name_nn NOT NULL);
```

CREATE TABLE AS SELECT 文

CREATE TABLE AS SELECT 文から発生した変更を適用する場合、適用プロセスでは次の 2 つの手順が実行されます。

1. CREATE TABLE AS SELECT 文が接続先データベースで実行されますが、表の構造のみが作成されます。表に行は挿入されません。CREATE TABLE AS SELECT 文が失敗すると、適用プロセスがエラーになります。それ以外の場合は、文が自動コミットされ、適用プロセスによって手順 2 が実行されます。
2. 適用プロセスによって、CREATE TABLE AS SELECT 文の結果としてソース・データベースで挿入された行が、接続先データベースで対応する表に挿入されます。取得プロセス、伝播または適用プロセスでは、これらの挿入を伴うすべての行 LCR が、それぞれのルール・セットに基づいて廃棄される場合があります。この場合、接続先データベースでは表が空のままになります。

注意： Streams 環境では、クラスタ化表に対する CREATE TABLE AS SELECT 文はサポートされません。

トリガー起動プロパティ

DBMS_DDL パッケージの SET_TRIGGER_FIRING_PROPERTY プロシージャを使用すると、DML または DDL トリガーの起動プロパティを制御できます。このプロシージャでは、トリガーの起動プロパティを 1 回起動するように設定するかどうかを指定できます。起動プロパティが 1 回起動するように設定されていると、次の場合にはトリガーは起動しません。

- 適用プロセスによって関連する変更が行われる場合。
- DBMS_APPLY_ADM パッケージの EXECUTE_ERROR または EXECUTE_ALL_ERRORS プロシージャを使用して 1 つ以上の適用が実行エラーになったために、関連する変更が生じた場合。

トリガーは、1 回起動するように設定されていなければ、前述のどちらの場合にも起動します。

デフォルトで、DML トリガーと DDL トリガーが 1 回起動するように設定されます。トリガーの起動プロパティは、DBMS_DDL パッケージの IS_TRIGGER_FIRE_ONCE ファンクションを使用してチェックできます。

たとえば、hr スキーマ内で、employees 表の job_id 列または department_id 列のデータが更新されると、update_job_history トリガーによって job_history 表に 1 行が追加されます。この場合、Streams 環境に次の構成が存在すると考えます。

- 取得プロセスは、dbs1.net データベースでこの 2 つの表に対する変更を取得します。
- 伝播は、これらの変更を dbs2.net データベースに伝播させます。

- 適用プロセスは、これらの変更を dbs2.net データベースで適用します。
- どちらのデータベースでも、hr スキーマに update_job_history トリガーが存在します。

この使用例で、dbs2.net で update_job_history トリガーが 1 回起動するように設定されていなければ、これらのアクションの結果は次のようになります。

1. dbs1.net で、employees 表の従業員に関する job_id 列が更新されます。
2. dbs1.net で update_job_history トリガーが起動し、変更を記録する job_history 表に 1 行が追加されます。
3. dbs1.net の取得プロセスは、employees 表と job_history 表の両方に対する変更を取得します。
4. 伝播は、これらの変更を dbs2.net データベースに伝播させます。
5. dbs2.net データベースの適用プロセスは、両方の変更を適用します。
6. 適用プロセスによって employees 表が更新されると、dbs2.net で update_job_history トリガーが起動します。

この場合、employees 表に対する変更は、dbs2.net データベースで 2 回記録されます。1 回は適用プロセスが変更を job_history 表に適用する時点で、もう 1 回は update_job_history トリガーが起動して、適用プロセスによって employees 表に対して行われた変更を記録する時点です。

このように、データベース管理者は、適用プロセスによって変更が行われるときに、dbs2.net データベースで update_job_history トリガーが起動しないようにする必要がある場合があります。同様に、データベース管理者は、適用エラー・トランザクションの実行が原因でトリガーが起動しないようにする必要がある場合もあります。update_job_history トリガーの起動プロパティを 1 回起動するように設定すると、適用プロセスによって変更が employees 表に適用されるときにも、実行されたエラー・トランザクションによって employees 表が更新されるときにも、dbs2.net でトリガーは起動しません。

また、ON SCHEMA 句を使用してスキーマ・トリガーを作成した場合、スキーマにより関連する変更が実行される場合にのみ、スキーマ・トリガーが起動します。したがって、適用プロセスにより変更が適用されるとき、適用ユーザーがスキーマ・トリガーで指定されたスキーマと同じ場合にのみ起動するように設定されたスキーマ・トリガーが、常に起動します。スキーマ・トリガーが 1 回起動するように設定されているとき、適用ユーザーがスキーマ・トリガーで指定されたスキーマと同じかどうかにかかわらず、適用プロセスでの変更適用時にトリガーは起動しません。

たとえば、ソース・データベースおよび接続先データベースの hr スキーマで常に起動するスキーマ・トリガーを指定した場合に、接続先データベースの適用ユーザーが strmadmin である場合、hr ユーザーがソース・データベースで関連する変更を実行するときにはトリガーが起動しますが、この変更が接続先データベースで適用されるときにはトリガーは起動しません。ただし、接続先データベースの strmadmin スキーマで常に起動するスキーマ・

トリガーを指定した場合は、ソース・データベースのトリガー仕様にかかわらず、関連する変更が適用プロセスにより適用されるときには必ずこのトリガーが起動します。

注意： 1回起動するように設定できるのは、DML トリガーと DDL トリガーのみです。他のタイプのトリガーは、すべて常に起動します。

関連項目： SET_TRIGGER_FIRING_PROPERTY プロシージャを使用してトリガーの起動プロパティを設定する方法の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

インスタンス化 SCN および非処理 SCN

複数のデータベース間で情報を共有する Streams 環境の場合、ソース・データベースとは REDO ログ内で変更が生成されるデータベースです。次の特性を持つ環境を考えます。

- 取得プロセスは、ソース・データベースで表に対する変更を取得します。
- 表に対する変更は、接続先データベースに伝播し、そこで適用されます。
- 接続先データベースには、すでに変更が取得、伝播および適用される一部またはすべての表が含まれています。

この環境では、接続先データベースに存在する表はインスタンス化されません。つまり、これらの表はすでに接続先データベースに存在するため、ソース・データベースでのエクスポートと接続先データベースでのインポートによって接続先データベースで作成されることはありません。かわりに、接続先データベースの適用プロセスに対して、ソース・データベースの各表について特定のシステム変更番号（SCN）より後にコミットされた変更を適用するよう明示的に指示する必要があります。この SCN は、表の**インスタンス化 SCN** によって指定されます。

データベース・オブジェクトのインスタンス化 SCN によって、データベース・オブジェクトに対する変更を含む LCR のうち、適用プロセスで無視される LCR と適用される LCR が制御されます。ソース・データベースからのデータベース・オブジェクトに関する LCR のコミット SCN が、接続先データベースでそのデータベース・オブジェクトのインスタンス化 SCN 以下であれば、接続先データベースの適用プロセスでは LCR が廃棄されます。それ以外の場合は、適用プロセスによって LCR が適用されます。

また、接続先データベースの共有データベース・オブジェクトに複数のソース・データベースがある場合は、ソース・データベースごとにインスタンス化 SCN を設定する必要があります。このインスタンス化 SCN は、ソース・データベースごとに異なっていてもかまいません。インスタンス化 SCN は、エクスポート / インポートを使用するか、または DBMS_APPLY_ADM パッケージのプロシージャを使用して設定できます。

また、Streams では各データベース・オブジェクトについて**非処理 SCN** が記録されます。非処理 SCN は、その下にインスタンス化 SCN を設定できない SCN です。この値は、オブジェクトがインスタンス化のために準備された時点でのソース・データベースの SCN 値に相当します。

データベース・オブジェクトのインスタンス化 SCN および非処理 SCN を表示するには、DBA_APPLY_INSTANTIATED_OBJECTS データ・ディクショナリ・ビューを問い合せます。

関連項目：

- 14-33 ページ「接続先データベースでのインスタンス化 SCN の設定」
- 12-10 ページ「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」

適用プロセスに関する最も古い SCN

適用プロセスが実行されている場合、**最も古い SCN** とは現在デキューおよび適用中のトランザクションの最も古い SCN です。停止した適用プロセスの場合、最も古い SCN とは適用プロセスが停止した時点で適用中だったトランザクションの最も古い SCN です。

最も古い SCN が重要になるのは、次の 2 つの一般的な使用例の場合です。

- 適用プロセスを実行中のデータベースを特定の時点までリカバリする必要がある場合。
- 適用プロセス用の変更を取得するために、既存の取得プロセスの使用を停止し、別の取得プロセスを使用する場合。

どちらの場合も、DBA_APPLY_PROGRESS データ・ディクショナリ・ビューを問い合せて、適用プロセス用の最も古い SCN を判別する必要があります。このビューの OLDEST_MESSAGE_NUMBER 列に、最も古い SCN が表示されます。次に、適用プロセス用の変更を取得する取得プロセスの開始 SCN を、最も古い SCN と同じ値に設定します。取得プロセスが他の適用プロセス用の変更を取得している場合は、取得プロセスの開始 SCN をリセットすると、これらの他の適用プロセスは重複する LCR イベントを受信することがあります。この場合、これらの適用プロセスでは重複する LCR イベントが自動的に廃棄されます。

関連項目： 2-14 ページ「取得プロセスの開始 SCN、取得済 SCN および適用済 SCN」

適用プロセスの最低水位標および最高水位標

適用プロセスの**最低水位標**は、すべてのイベントが適用済となっているシステム変更番号（SCN）です。つまり、最低水位標番号以下の SCN でコミットされたイベントは確実に適用されており、それを超える SCN でコミットされた一部のイベントも適用されている可能性があります。最低水位標は**適用済 SCN** と呼ばれることもあります。

適用プロセスの**最高水位標**は、その番号を超えて適用されたイベントのない SCN です。つまり、最高水位標を超えた SCN でコミットされたイベントは適用されていません。

1 つ以上の適用プロセスについて最低水位標と最高水位標を表示するには、V$STREAMS_APPLY_COORDINATOR および ALL_APPLY_PROGRESS データ・ディクショナリ・ビューを問い合せます。

Streams の適用プロセスと RESTRICTED SESSION

システム起動時に STARTUP RESTRICT 文を発行して制限付きセッションを有効化した場合は、適用プロセスは起動しません。これは、データベースの停止時に適用プロセスが実行中であった場合も同様です。制限付きセッションを無効化すると、停止しなかった各適用プロセスが起動します。

ALTER SYSTEM 文の ENABLE RESTRICTED SESSION 句を使用して実行中のデータベースで制限付きセッションを有効化した場合は、実行中の適用プロセスには影響しません。これらの適用プロセスは引き続き実行され、イベントの適用が行われます。停止した適用プロセスは、制限付きセッション内では、制限付きセッションが無効化されるまで起動しません。

Streams の適用プロセスと Oracle Real Application Clusters

Streams の適用プロセスは、Real Application Clusters 環境での変更を適用するように構成できます。適用プロセスを使用して Real Application Clusters 環境で取得された LCR を適用する場合は、DBMS_APPLY_ADM パッケージの START_APPLY プロシージャのコールを、適用プロセスで使用されるキューの所有者インスタンス上で実行する必要があります。

適用プロセスを操作する他のプロシージャとファンクションのコールは、どのインスタンスからでも実行できます。また、ユーザー・エンキュー・イベントを適用する適用プロセスは、どのインスタンスでも起動できます。

適用プロセスで使用されるキューを含むキュー表の所有者インスタンスが使用不可能になると、キューの所有権は自動的にクラスタ内の別のインスタンスに移ります。この状況が発生した場合は、適用プロセスを再起動するために、キューの所有者インスタンスに接続し、START_APPLY プロシージャを実行してください。DBA_QUEUE_TABLES データ・ディクショナリ・ビューは、キュー表の所有者インスタンスに関する情報を示します。適用プロセスが Real Application Clusters 環境で永続的な起動 / 停止状態を維持するのは、キューを所有するデータベース・インスタンスが再起動される前に、キューの所有者インスタンスに変更がない場合のみです。

また、Real Application Clusters 環境では、適用コーディネータ・プロセス、それに対応する適用リーダー・サーバーおよびすべての適用サーバー・プロセスが、単一のインスタンスで実行されます。

関連項目：

- 3-17 ページ「Streams キューと Oracle Real Application Clusters」
- 2-15 ページ「Streams の取得プロセスと Oracle Real Application Clusters」
- DBA_QUEUE_TABLES データ・ディクショナリ・ビューの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。
- 4-34 ページ「適用プロセスの永続的状態」

適用プロセスのアーキテクチャ

適用プロセスの作成、変更、起動、停止および削除と、適用プロセスによってキューからデキューされるイベントを制御する適用ルールの定義を行うことができます。デフォルトでは、適用プロセスを作成したユーザーが、変更を適用するユーザーとなります。このユーザーは、変更の適用に必要な権限を持っている必要があります。

関連項目： 必要な権限については、11-2 ページの「Streams 管理者の構成」を参照してください。

この項の内容は、次のとおりです。

- 適用プロセスのコンポーネント
- 適用プロセスの作成
- 適用プロセス用の Streams データ・ディクショナリ
- 適用プロセスのパラメータ
- 適用プロセスの永続的状態
- 例外キュー

適用プロセスのコンポーネント

適用プロセスのコンポーネントは、次のとおりです。

- イベントをデキューする**リーダー・サーバー**。リーダー・サーバーはパラレル実行サーバーであり、LCR 間の依存性を計算してイベントをトランザクションにアセンブルします。次に、アセンブルしたトランザクションをコーディネータに戻すと、コーディネータはそれをアイドル状態の適用サーバーに割り当てます。
- リーダー・サーバーからトランザクションを取得して適用サーバーに渡す**コーディネータ・プロセス**。コーディネータ・プロセス名は ap*nn* で、*nn* はコーディネータ・プロセス番号です。有効なコーディネータ・プロセス名は、ap01 ～ ap99 です。
- LCR を DML 文または DDL 文としてデータベース・オブジェクトに適用するか、LCR を適切なハンドラに渡す 1 つ以上の**適用サーバー**。LCR 以外のメッセージの場合、適用サーバーはイベントをメッセージ・ハンドラに渡します。各適用サーバーは、パラレル実行サーバーです。適用サーバーにエラーが発生すると、ユーザー指定のエラー・ハンドラを使用してエラーを解決しようとします。エラーを解決できない場合、適用サーバーはトランザクションをロールバックし、すべてのイベントを含めてトランザクション全体を例外キューに置きます。

  適用サーバーが完了したトランザクションをコミットする時点で、このトランザクションは適用済みになっています。適用サーバーがトランザクションを例外キューに置いてコミットした場合も、このトランザクションは適用済みになっています。

適用サーバーで処理中のトランザクションに、適用済みとして認識されていない別のトランザクションとの依存性がある場合、適用サーバーはコーディネータに通知して指示があるまで待機します。コーディネータはすべての適用サーバーを監視して、トランザクションが適切な順序で適用され、コミットされることを確認します。

たとえば、次の 2 つのトランザクションを考えます。

1. 表に 1 行が挿入されます。
2. 同じ行が、特定の列の値を変更するように更新されます。

この場合、行は表に挿入されるまで更新できないため、トランザクション 2 はトランザクション 1 に依存します。この 2 つのトランザクションがソース・データベースで REDO ログから取得され、接続先データベースに伝播され、そこで適用されるとします。適用サーバー A は挿入トランザクションを処理し、適用サーバー B は更新トランザクションを処理します。

適用サーバー A が挿入トランザクションを適用する前に、適用サーバー B で更新トランザクションを適用する準備が完了すると、適用サーバー B はコーディネータから指示があるまで待機します。適用サーバー A が挿入トランザクションを適用すると、コーディネータ・プロセスは更新トランザクションを適用するように適用サーバー B に対して指示します。

適用プロセスの作成

適用プロセスを作成するには、DBMS_STREAMS_ADM パッケージを使用する方法と、DBMS_APPLY_ADM パッケージを使用する方法があります。一部の構成オプションにはデフォルトが自動的に使用されるため、DBMS_STREAMS_ADM パッケージを使用して適用プロセスを作成する方が簡単です。また、DBMS_STREAMS_ADM パッケージを使用すると、自動的に適用プロセスのルール・セットが作成され、そのルール・セットにルールが追加されます。DBMS_STREAMS_ADM パッケージは、レプリケーション環境で使用するように設計されています。または、DBMS_APPLY_ADM パッケージを使用すると適用プロセスを柔軟に作成できます。また、ルール・セットとルールの作成は適用プロセスの作成前または後に行います。

DBMS_STREAMS_ADM パッケージのプロシージャで作成した適用プロセスでイベントを適用できるのは、ローカル・データベースのみで、適用できるのは取得イベントのみです。リモート・データベースでイベントを適用する適用プロセスや、ユーザー・エンキュー・イベントを適用する適用プロセスを作成するには、DBMS_APPLY_ADM パッケージの CREATE_APPLY プロシージャを使用します。

DBMS_STREAMS_ADM パッケージで作成した適用プロセスによって適用される変更では、接続先データベースの REDO ログ内に値 00（2 つのゼロ）のタグが生成されます。ただし、CREATE_APPLY プロシージャを使用する場合は、このタグの値を設定できます。また、タグを設定するには、DBMS_APPLY_ADM パッケージの ALTER_APPLY プロシージャを使用する方法もあります。

DBMS_APPLY_ADM パッケージの CREATE_APPLY プロシージャを実行して適用プロセスを作成する場合は、apply_captured、apply_database_link および apply_tag パラメータにデフォルト以外の値を指定できます。DBMS_STREAMS_ADM パッケージまたは DBMS_RULE_ADM パッケージのプロシージャを使用すると、適用プロセスのルール・セットにルールを追加できます。

データベースに複数の適用プロセスを作成すると、各適用プロセスは相互にまったく依存しません。これらの適用プロセスは、同じソース・データベースからの LCR を適用する場合にも、相互に同期化されません。

> **関連項目：** 適用プロセスの作成に使用できる次のプロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
>
> - DBMS_STREAMS_ADM.ADD_TABLE_RULES
> - DBMS_STREAMS_ADM.ADD_SUBSET_RULES
> - DBMS_STREAMS_ADM.ADD_SCHEMA_RULES
> - DBMS_STREAMS_ADM.ADD_GLOBAL_RULES
> - DBMS_CAPTURE_ADM.CREATE_APPLY

適用プロセス用の Streams データ・ディクショナリ

取得プロセスが作成されると、Streams データ・ディクショナリと呼ばれる複製データ・ディクショナリが自動的に移入されます。Streams データ・ディクショナリは、ソース・データベースにある 1 次データ・ディクショナリ内の一部の情報のマルチバージョン・コピーです。Streams データ・ディクショナリでは、取得プロセスによってルールが評価されて LCR が作成されると、ソース・データベースからのオブジェクト番号、オブジェクトのバージョン情報および内部の列番号が、表名、列名および列のデータ型にマップされます。取得イベントでは通常は内部で名前ではなく番号を使用できるため、このマッピングによって各取得イベントが最小サイズに保たれます。

取得イベントが取得または伝播中にパラメータとしてユーザー変換に渡されないかぎり、取得イベントを適用するデータベースで LCR の内容を解析するには、ソース・データベースの Streams データ・ディクショナリ内のマッピング情報が必要です。このマッピング情報を適用プロセスで使用できるように、Oracle は Streams の適用プロセスがある各接続先データベースでマルチバージョンの Streams データ・ディクショナリを自動的に移入します。ソース・データベースにある Streams データ・ディクショナリからの関連情報は、ソース・データベースからの取得イベントを適用する他のすべてのデータベースに自動的に伝播されます。

関連項目： 2-20 ページ「取得プロセス作成中のデータ・ディクショナリの複製」

適用プロセスのパラメータ

作成後の適用プロセスは無効化されているため、初めて起動する前に環境に合せて適用プロセス・パラメータを設定できます。適用プロセス・パラメータでは、適用プロセスの動作を制御します。たとえば、`time_limit` 適用プロセス・パラメータを使用すると、適用プロセスが自動的に停止するまでの実行時間を指定できます。適用プロセス・パラメータの設定後に、適用プロセスを開始します。

関連項目：

- 14-11 ページ「適用プロセス・パラメータの設定」
- この項では、使用可能な適用プロセス・パラメータの一部のみを説明しています。すべての適用プロセス・パラメータの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の `DBMS_APPLY_ADM.SET_PARAMETER` プロシージャの項を参照してください。

適用プロセスの並列性

parallelism 適用プロセス・パラメータでは、トランザクションを同時に適用できる適用サーバーの数を指定します。たとえば、parallelism を 5 に設定すると、適用プロセスでは合計 5 つの適用サーバーが使用されます。また、リーダー・サーバーはパラレル実行サーバーです。そのため、データベースで 6 つのパラレル実行サーバーが使用可能な場合に、parallelism を 5 に設定すると、適用プロセスでは合計 6 つのパラレル実行サーバーが使用されます。適用プロセスでは、常に 1 つ以上のパラレル実行サーバーが使用されます。

注意：

- parallelism パラメータをリセットすると、適用プロセスは自動的に停止し、現在実行中のトランザクションが適用されるときに再起動されます。トランザクションのサイズによっては、この間に少し時間がかかる場合があります。
- parallelism パラメータを使用可能なパラレル実行サーバー数より大きい数に設定すると、適用プロセスが無効化される場合があります。parallelism 適用プロセス・パラメータの設定時には、PROCESSES および PARALLEL_MAX_SERVERS 初期化パラメータが適切に設定されているかどうかを確認してください。

関連項目：

- 適用サーバーとリーダー・サーバーの詳細は、4-29 ページの「適用プロセスのコンポーネント」を参照してください。
- パラレル実行サーバーの管理については、『Oracle9*i* データベース管理者ガイド』を参照してください。

コミットのシリアル化

適用サーバーが、ソース・データベースでのコミット順序とは異なる順序に従って、接続先データベースでトランザクションを適用する場合があります。ソース・データベースでのコミット順序とは異なる順序で適用できるのは、非依存トランザクションのみです。依存トランザクションは、常にソース・データベースでコミットされたのと同じ順序に従って、接続先データベースで適用されます。

commit_serialization 適用パラメータを使用して、適用サーバーがソース・データベースとは異なる順序に従って接続先データベースで非依存トランザクションを適用できるかどうかを制御します。このパラメータの設定値は次のとおりです。

- full: 適用プロセスは、ソース・データベースでのコミット時と同じ順序に従って、適用済みトランザクションをコミットします。この設定がデフォルトです。
- none: 適用プロセスは、任意の順序でトランザクションをコミットできます。この値を指定すると、最大限のパフォーマンスが得られます。

none を指定すると、接続先データベースにソース・データベースとは異なる順序でコミット変更が格納される可能性があります。たとえば、ソース・データベースでは、2 つの非依存トランザクションが次の順序でコミットされるとします。

1. トランザクション A
2. トランザクション B

接続先データベースでは、これらのトランザクションが逆の順序でコミットされる可能性があります。

1. トランザクション B
2. トランザクション A

適用プロセスの自動再起動

適用プロセスは、事前定義済みの特定の上限に達した時点で自動的に停止するように構成できます。time_limit 適用プロセス・パラメータで適用プロセスの実行時間を指定し、transaction_limit 適用プロセス・パラメータで適用プロセスが適用できるトランザクションの数を指定します。適用プロセスは、これらの上限に達した時点で自動的に停止します。

disable_on_limit パラメータでは、適用プロセスが上限に達した時点で無効化されるか、再起動するかを制御します。disable_on_limit パラメータを y に設定すると、適用プロセスは上限に達した時点で無効化され、明示的に指定するまで再起動されません。ただし、disable_on_limit パラメータを n に設定すると、適用プロセスは上限に達した時点で自動的に停止し、再起動します。

再起動された適用プロセスは新規のセッション識別子を取得し、適用プロセスに関連付けられているパラレル実行サーバーも新規のセッション識別子を取得します。ただし、コーディネータ・プロセス番号（ap*nn*）は変わりません。

エラー発生時の停止または継続

disable_on_error 適用プロセス・パラメータを使用すると、適用プロセスをエラー発生時に無効化するか、エラー発生後も引き続きトランザクションの適用を許可するかを指定できます。

関連項目： 4-34 ページ「例外キュー」

適用プロセスの永続的状態

適用プロセスは、永続的状態を保持します。つまり、適用プロセスは、データベースが停止され、再起動されても、最新の状態に保たれます。たとえば、データベースの停止時に適用プロセスが実行中だった場合は、データベースが再起動されると適用プロセスは自動的に起動されます。ただし、データベースの停止時に適用プロセスが停止していた場合は、データベースを再起動しても適用プロセスは停止したままです。

例外キュー

例外キューは、各キュー表に関連付けられています。DBMS_STREAMS_ADM パッケージの SET_UP_QUEUE プロシージャを使用して Streams キューを作成するとき、Streams キューで使用されるキュー表に例外キューが存在しない場合は、例外キューが自動的に作成されます。データベースのすべての例外キュー内の Streams 適用エラーに関する情報を表示するには、DBA_APPLY_ERROR データ・ディクショナリ・ビューを問い合せます。

例外キューには、データベースで実行中の適用プロセスにより正常に適用できなかったトランザクションに関する情報が格納されます。トランザクションには多数のイベントが含まれる場合があり、適用中に未処理エラーが発生すると、適用プロセスでそのルール・セットを満たすトランザクション内の全イベントが例外キューに自動的にコピーされます。例外キューに移動した最後のエラーはエラー・スタックの最上部にあります。

例外キューには、ローカル接続先データベースで発生したエラーに関する情報のみが格納されます。Streams 環境内の他のデータベースで実行中の適用プロセスに関するエラー情報は格納されません。

エラーの原因となった条件を訂正し、エラーになったトランザクションを再実行できます。たとえば、表の 1 行を変更して、エラーの原因となった条件を訂正できます。エラーの原因となった条件を訂正した後、EXECUTE_ERROR および EXECUTE_ALL_ERRORS プロシージャを使用して例外キュー内でトランザクションを再実行するか、DELETE_ERROR および DELETE_ALL_ERRORS プロシージャを使用して例外キューからトランザクションを削除できます。これらのプロシージャは、いずれも DBMS_APPLY_ADM パッケージにあります。

例外キュー内のトランザクションを再実行する場合に、そのトランザクションを実行するユーザーとして、最初に例外キューにエラーを置いたユーザーを指定する方法と、トランザクションを再実行するユーザーを指定する方法があります。また、例外キュー内のトランザクションを再実行する場合は、適用プロセスの現行の Streams タグが使用されます。

再実行されたトランザクションでは、関連する適用ハンドラと競合解消ハンドラが使用されます。エラーを解決するために例外キュー内の LCR を実行前に変更する必要がある場合は、例外キュー内でエラーの原因となった LCR を処理するように DML ハンドラを構成できます。この場合、DML ハンドラはなんらかの方法で LCR を変更し、同じエラーの繰返しを回避できます。DBMS_APPLY_ADM パッケージの EXECUTE_ERROR または EXECUTE_ALL_ERRORS プロシージャを使用して、LCR を含むエラーを再実行すると、LCR が DML ハンドラに渡されます。

関連項目：

- 14-31 ページ「適用エラーの管理」
- 17-34 ページ「適用エラーのチェック」
- 17-35 ページ「適用エラーの詳細情報の表示」
- 14-13 ページ「DML ハンドラの管理」
- DBMS_APPLY_ADM パッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- DBA_APPLY_ERROR データ・ディクショナリ・ビューの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

5
ルール

この章では、ルールに関連する概念について説明します。

この章の内容は次のとおりです。

関連項目：

- 第 6 章「Streams でのルールの使用方法」
- 第 15 章「ルールおよびルールベースの変換の管理」
- 第 24 章「ルールベースのアプリケーションの例」

ルールの構成要素

ルールとは、イベントの発生時に条件が満たされている場合に、クライアントでアクションを実行できるようにするデータベース・オブジェクトです。ルールは、Oracle の組込み部分である**ルール・エンジン**によって評価されます。ユーザー作成アプリケーションと Streams などの Oracle の機能の両方が、ルール・エンジンのクライアントとなることができます。

ルールの構成要素は、次のとおりです。

- ルール条件
- ルール評価コンテキスト（オプション）
- ルール・アクション・コンテキスト（オプション）

各ルールは、SQL 問合せの WHERE 句の条件と同様の条件として指定します。関連するルールをグループ化して**ルール・セット**を作成できます。1 つのルールを 1 つ以上のルール・セットに含めることができます。また、どのルール・セットにも含めなくてもかまいません。

> **注意：** ルールを評価させるには、ルール・セットに含める必要があります。

ルール条件

ルール条件は、1 つ以上の式と演算子を組み合せてブール値を戻します。ブール値は、TRUE、FALSE または NULL（不明）のいずれかです。**式**は、1 つ以上の値と、評価結果が値になる演算子の組合せです。値には、表のデータ、変数のデータ、SQL 関数または PL/SQL ファンクションから戻されるデータを使用できます。たとえば、次の条件は 2 つの式（department_id および 30）と 1 つの演算子（=）で構成されています。

```
department_id = 30
```

この論理条件は、department_id 列が 30 の場合は、特定の行について TRUE と評価されます。この場合、値は表の department_id 列のデータです。

複数の条件を AND、OR および NOT 条件演算子で結合して複合条件を形成し、1 つのルール条件に含めることができます。たとえば、次の複合条件を考えます。

```
department_id = 30 OR job_title = 'Programmer'
```

このルール条件には、OR 条件演算子で結合された 2 つの条件が含まれています。どちらかの条件が TRUE と評価されると、ルール条件が TRUE と評価されます。条件演算子が OR ではなく AND の場合、ルール条件全体が TRUE と評価されるためには、両方の条件が TRUE と評価される必要があります。

ルール条件内の変数

ルール条件には変数を使用できます。その場合は、各変数の先頭にコロン（:）を付けます。ルール条件に使用する変数の例を次に示します。

```
:x = 55
```

変数を使用すると、表に格納されていないデータを参照できます。また、変数を使用すると、共通して発生する式を置換することでパフォーマンスを改善できる場合があります。これは、同じ式が 2 回以上評価されるかわりに、変数が 1 回評価されるためです。

ルール条件には、サブプログラムのコールの評価も含めることができます。これらの条件は、他の条件と同じ方法で評価されます。つまり、値 TRUE、FALSE または不明として評価されます。次の例に、従業員がマネージャかどうかを判断する単純なファンクション is_manager のコールを含む条件を示します。

```
is_manager(employee_id) = 'Y'
```

この場合、employee_id の値は employee_id 列を持つ表のデータによって判断されます。

ユーザー定義型を変数に対して使用できます。したがって、変数では属性を指定できます。変数に属性を指定する場合、各属性には変数に対する部分データが含まれます。ルール条件内では、属性はドット表記法を使用して指定します。たとえば、次の条件は、変数 y の属性 z の値が 9 の場合に TRUE と評価されます。

```
:y.z = 9
```

> **関連項目：** ユーザー定義型の詳細は、『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

単純ルール条件

単純ルール条件とは、次のいずれかの形式で表される条件を指します。

- *simple_rule_expression operator constant*
- *constant operator simple_rule_expression*

単純ルール条件では、*simple_rule_expression* は次のいずれかです。

- 表の列
- 変数
- 変数の属性
- 引数を取らないメソッドの結果。メソッドの結果は変数メソッド・ファンクションから戻すことができるため、式は数値または文字型です。

表の列、変数、および変数の属性については、すべての数値型（NUMBER、FLOAT、DOUBLE、INTEGER）と文字型（CHAR、VARCHAR2）がサポートされます。他の型の式を使用すると、単純ルール条件ではなくなります。

単純ルール条件では、*operator* は次のいずれかです。

- <=
- <
- =
- >
- >=

他の型の演算子を使用すると、単純ルール条件ではなくなります。

constant は固定値です。定数は次のいずれかです。

- 12 または 5.4 などの数値
- x または $ などの文字
- this is a string などの文字列

したがって、次の条件が単純ルール条件です。

- tab1.col = 5
- :v1 > 'aaa'
- :v2.a1 < 10.01
- :v3.m() = 10

単純ルール条件を含むルールは、単純なルールと呼ばれます。演算子 AND および OR を使用して 2 つ以上の単純ルール条件をルールに結合しても、ルールの単純さは保たれます。ただし、ルールの条件内で NOT 条件演算子を使用すると、ルールは単純ではなくなります。たとえば、次の条件を含むルールは単純なルールです。

- tab1.col = 5 AND :v1 > 'aaa'
- tab1.col = 5 OR :v1 > 'aaa'

単純なルールは、次の理由で重要です。

- 単純なルールは、ルール・エンジンによって内部的に索引を付けられます。
- 単純なルールは、SQL を実行せずに評価できます。
- 単純なルールは、部分データを使用して評価できます。

クライアントが DBMS_RULE.EVALUATE を使用してイベントを評価する場合、クライアントは、simple_rules_only パラメータを TRUE に指定することにより、単純なルールのみを評価するよう指定できます。

> **関連項目：** 条件、式および演算子の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

ルール評価コンテキスト

ルール評価コンテキストは、ルール条件内で参照できる外部データを定義するデータベース・オブジェクトです。外部データは、変数または表データ、あるいはその両方として存在します。次のように類推するとわかりやすくなります。ルール条件が SQL 問合せの WHERE 句であると考えると、ルール評価コンテキスト内の外部データは、その問合せの FROM 句で参照される情報です。つまり、有効な WHERE 句を作成するには、ルール条件内の式が評価コンテキスト内の表、表の別名および変数を参照する必要があります。

ルール評価コンテキストは、外部データを参照するルール条件の解析と評価に必要な情報を提供します。たとえば、ルールで変数を参照する場合、ルール評価コンテキスト内の情報には変数の型を含める必要があります。また、ルールで表の別名を参照する場合、評価コンテキストには表の別名を定義する情報を含める必要があります。

ルールによって参照されるオブジェクトは、それに関連付けられたルール評価コンテキストによって決定されます。ルール所有者は、表に対する SELECT 権限や型に対する EXECUTE 権限など、これらのオブジェクトへのアクセスに必要な権限を持つ必要があります。ルール条件は、評価コンテキストを所有するスキーマ内で解決されます。

たとえば、次の情報を含むルール評価コンテキスト hr_evaluation_context を考えます。

- 表の別名 dep は、hr.departments 表に対応します。
- 変数 loc_id1 および loc_id2 は、どちらも NUMBER 型です。

hr_evaluation_context ルール評価コンテキストは、次のルール条件の評価に必要な情報を提供します。

```
dep.location_id IN (:loc_id1, :loc_id2)
```

この場合、loc_id1 または loc_id2 変数で渡された値のどちらかに対応する location_id 列に行の値があると、hr.departments 表のその行についてはルール条件が TRUE と評価されます。hr_evaluation_context ルール評価コンテキスト内に情報がなければ、ルールを適切に解析または評価できません。また、dep 表の別名での location_id の指定にドット表記法を使用していることに注意してください。

明示的変数と暗黙的変数

ルール条件内で参照される変数の値をルールが評価されるときに明示的に指定するか、変数の値を特定のイベントに暗黙的に使用可能にすることができます。

明示的変数は、評価時にコール元から提供されます。これらの値は、DBMS_RULE.EVALUATE プロシージャの実行時に variable_values パラメータで指定します。

暗黙的変数には、評価時には値が与えられません。暗黙的変数の値は、変数値の評価ファンクションをコールすることで取得されます。このファンクションは、DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT プロシージャを使用して評価コンテキストの作成中に variable_types リストを指定するときに定義します。暗黙的変数の値が評価中に指定されると、その値によって変数値の評価ファンクションから戻される値がオーバーライドされます。

特に、variable_types リストは SYS.RE$VARIABLE_TYPE_LIST 型であり、SYS.RE$VARIABLE_TYPE 型の変数のリストです。リストにある SYS.RE$VARIABLE_TYPE の各インスタンス内で、暗黙的変数の値の決定に使用されるファンクションを variable_value_function 属性として指定します。

変数が明示的であるか暗黙的であるかは、アプリケーション・デザイナがルール・エンジンを使用して選択します。暗黙的変数を使用する理由は、次のとおりです。

- DBMS_RULE.EVALUATE プロシージャのコール元は、変数に関する情報を認識する必要がありません。この場合は、ルール・エンジンを使用してアプリケーションの複雑さを軽減できます。たとえば、変数では、評価対象のデータに基づいて値を戻すファンクションをコールできます。
- コール元には、変数の値の評価ファンクションに対する実行権限がなくてもかまいません。
- DBMS_RULE.EVALUATE プロシージャのコール元は、イベントに基づいた変数値を認識しません。このため、変数値に機密情報が含まれている場合に、セキュリティを向上できます。
- 変数の使用頻度が低い場合があり、また変数の値は必要に応じていつでも導出できます。この種の変数を暗黙的変数にすると、DBMS_RULE.EVALUATE プロシージャのコール元では、一般的ではない多数の変数を指定する必要がなくなります。

たとえば、次のルール条件では、変数 x および y の値を明示的に指定できますが、変数 max の値は max ファンクションを実行することで戻されます。

```
:x = 4 AND :y < :max
```

また、変数 x および y を暗黙的変数に、変数 max を明示的変数にすることもできます。このように、ルール条件内では、明示的変数と暗黙的変数に構文上の違いはありません。変数が明示的であるか暗黙的であるかは、DBA_EVALUATION_CONTEXT_VARS データ・ディクショナリ・ビューを問い合せることで判断できます。明示的変数の場合は、VARIABLE_VALUE_FUNCTION フィールドが NULL になっています。暗黙的変数の場合、このフィールドにはその変数によってコールされるファンクションの名前が含まれます。

関連項目：

- DBMS_RULE および DBMS_RULE_ADM パッケージと、Oracle が提供するルールのタイプの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- DBA_EVALUATION_CONTEXT_VARS データ・ディクショナリ・ビューの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

評価コンテキストとルール・セットおよびルールとの関連付け

1 つのルール評価コンテキストを、複数のルールまたはルール・セットに関連付けることができます。次のリストに、ルールが評価されるときに使用される評価コンテキストを示します。

- 評価コンテキストがルールに関連付けられている場合は、ルールが評価されるたびにその評価コンテキストが使用され、評価対象のルール・セットに関連付けられている評価コンテキストは無視されます。
- ルールに評価コンテキストがなくても、DBMS_RULE_ADM パッケージの ADD_RULE プロシージャを使用してルール・セットに追加するときに、ルール用の評価コンテキストを指定した場合、ルール・セットが評価されるときには、ADD_RULE プロシージャで指定した評価コンテキストがルールに使用されます。
- ルール評価コンテキストがルールに関連付けられておらず、ADD_RULE プロシージャでも指定されていない場合、ルール・セットが評価されるときには、ルール・セットの評価コンテキストがルールに使用されます。

注意： ルールに評価コンテキストがない場合に、そのルールを評価コンテキストのないルール・セットに追加しようとすると、ADD_RULE プロシージャの実行時に評価コンテキストを指定しないかぎりエラーが発生します。

評価ファンクション

ルール評価コンテキストで実行する評価ファンクションを作成できるようにオプションが用意されています。次のような理由で、評価ファンクションの使用を選択する場合があります。

- ルール・エンジンをバイパスし、かわりに評価ファンクションを使用してイベントを評価する場合
- イベントをフィルタ処理して、一部のイベントは評価ファンクションを使用して評価し、他のイベントはルール・エンジンを使用して評価する場合

DBMS_RULE_ADM パッケージの CREATE_EVALUATION_CONTEXT プロシージャを使用してルール評価コンテキストを作成するときに、evaluation_function パラメータにファンクション名を指定して、ファンクションをルール評価コンテキストに関連付けることができます。その場合、評価ファンクションは、評価コンテキストを使用する任意のルール・セットの評価中に、ルール・エンジンによって起動されます。DBMS_RULE.EVALUATE プロシージャのファンクションには各パラメータが必要です。各パラメータには、DBMS_RULE.EVALUATE プロシージャ内の対応するパラメータと同じ型を使用する必要がありますが、パラメータの名前は異なる場合があります。

評価ファンクションから戻される値は、次のとおりです。

- DBMS_RULE_ADM.EVALUATION_SUCCESS: ユーザーにより指定された評価ファンクションが、ルール・セットの評価を正常終了しました。ルール・エンジンは、評価ファンクションによって取得された評価結果を DBMS_RULE.EVALUATE プロシージャを使用してルール・エンジン・クライアントに戻します。
- DBMS_RULE_ADM.EVALUATION_CONTINUE: ルール・エンジンは、評価ファンクションが存在しないかのようにルール・セットを評価します。評価ファンクションは使用されず、評価ファンクションから戻される結果はすべて無視されます。
- DBMS_RULE_ADM.EVALUATION_FAILURE: ユーザーにより指定された評価ファンクションが失敗しました。ルール・セットの評価は停止し、ルール・エンジンは、評価ファンクションによって取得された評価結果を DBMS_RULE.EVALUATE プロシージャを使用してルール・エンジン・クライアントに戻します。

ルール・エンジンを常にバイパスする場合は、評価ファンクションは EVALUATION_SUCCESS または EVALUATION_FAILURE を戻す必要があります。ただし、イベントをフィルタ処理して一部のイベントを評価ファンクションで評価し、他のイベントをルール・エンジンで評価する場合、評価ファンクションが 3 つの戻り値すべてを戻すことがあります。評価にルール・エンジンを使用する必要がある場合は、EVALUATION_CONTINUE を戻します。

評価コンテキストに対して評価ファンクションを指定していると、その評価コンテキストがルール・セットまたはルールによって使用される場合、評価のあいだ、その評価ファンクションが実行されます。

> **関連項目：** DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT プロシージャ内に指定する評価ファンクションの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ルール・アクション・コンテキスト

ルール・アクション・コンテキストには、イベントのルールの評価時にルール・エンジンのクライアントによって解析されるルールに関連付けられたオプションの情報が含まれています。ルール・エンジンのクライアントは、ユーザー作成アプリケーション、または Streams などの Oracle の内部機能です。各ルールのアクション・コンテキストは 1 つのみです。アクション・コンテキスト内の情報は、名前 / 値ペアの配列を含む型である SYS.RE$NV_LIST 型です。

ルール・アクション・コンテキスト情報は、ルールが TRUE に評価される場合に、ルール・エンジンのクライアントによって実行されるアクションのコンテキストを提供します。ルール・エンジンは、アクション・コンテキストを解析しません。かわりに、ルールが TRUE と評価されると、アクション・コンテキスト情報を戻します。これにより、ルール・エンジンのクライアントではアクション・コンテキスト情報を解析できます。

たとえば、イベントが会社への新規従業員の追加として定義されているとします。従業員情報が hr.employees 表に格納されている場合は、この表に行が挿入されるたびにイベントが発生します。会社は、新規従業員を追加するときに多数のアクションを実行するが、アクションは従業員の所属部門に応じて異なるように指定する必要があります。これらのアクションの 1 つは、従業員を部門関連のコースに登録することです。

この使用例では、会社は適切なアクション・コンテキストを使用して、各部門用のルールを作成できます。この場合、ルールが TRUE と評価された場合に戻されるアクション・コンテキストでは、従業員が参加する必要のあるコースの番号を指定します。3 つの部門がある場合のルール条件とアクション・コンテキストを次に示します。

ルール名	ルール条件	アクション・コンテキストの名前 / 値ペア
rule_dep_10	department_id = 10	course_number, 1057
rule_dep_20	department_id = 20	course_number, 1215
rule_dep_30	department_id = 30	NULL

これらのアクション・コンテキストでは、クライアント・アプリケーションに次の指示が戻されます。

- rule_dep_10 ルールのアクション・コンテキストはクライアント・アプリケーションに対して、新規従業員をコース番号 1057 に登録するように指示します。
- rule_dep_20 ルールのアクション・コンテキストはクライアント・アプリケーションに対して、新規従業員をコース番号 1215 に登録するように指示します。
- rule_dep_30 ルールの NULL アクション・コンテキストはクライアント・アプリケーションに対して、新規従業員をどのコースにも登録しないように指示します。

各アクション・コンテキストには、0 個以上の名前 / 値ペアを含めることができます。アクション・コンテキストに複数の名前 / 値ペアが含まれている場合、リスト内の名前はそれぞれ一意であることが必要です。この例では、ルール・エンジンからアクション・コンテキス

トが戻されるクライアント・アプリケーションでは、戻されたコース番号を持つコースに新規従業員が登録されます。NULL アクション・コンテキストが戻される場合や、アクション・コンテキストにコース番号が含まれていない場合、クライアント・アプリケーションでは従業員はコースに登録されません。

複数のクライアントで同じルールを使用する場合や、アクション・コンテキストから複数の名前 / 値ペアが戻されるようにする必要がある場合は、1 つのアクション・コンテキストで複数の名前 / 値ペアをリストできます。たとえば、会社が新規従業員を部門の電子メール・リストにも追加する場合を考えます。この場合、rule_dep_10 ルールのアクション・コンテキストに次の 2 つの名前 / 値ペアを含めることができます。

名前	値
course_number	1057
dist_list	admin_list

名前 / 値ペアで指定する名前については、次のことを考慮する必要があります。

- 異なるアプリケーションで同じアクション・コンテキストを使用する場合は、異なる名前または名前の接頭辞を使用してネーミングの競合を回避してください。
- Oracle が提供するアクション・コンテキスト名との競合を回避するために、名前には $ と # を使用しないでください。

Streams では、ルールベースの変換にアクション・コンテキストが使用されます。また、サブセット・ルールが指定されている場合は、UPDATE 操作を含む LCR に必要な内部変換にも、アクション・コンテキストが使用されます。

RE$NV_LIST 型の ADD_PAIR メンバー・プロシージャを使用すると、アクション・コンテキストに名前 / 値ペアを追加できます。RE$NV_LIST 型の REMOVE_PAIR メンバー・プロシージャを使用すると、アクション・コンテキストから名前 / 値ペアを削除できます。アクション・コンテキスト内の既存の名前 / 値ペアを変更する場合は、最初に REMOVE_PAIR メンバー・プロシージャを使用して削除してから、ADD_PAIR メンバー・プロシージャを使用して適切な名前 / 値ペアを追加する必要があります。

関連項目：

- 4-12 ページ「Streams ルールを使用した行のサブセット化」
- RE$NV_LIST 型の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 名前 / 値ペアを追加および変更する例については、6-23 ページの「ルールベースの変換」を参照してください。

ルール・セットの評価

ルール・エンジンでは、イベントに基づいてルール・セットが評価されます。**イベント**とは、ルール・エンジンのクライアントによって定義された状態変化です。クライアントは、DBMS_RULE.EVALUATE プロシージャをコールして、イベントの評価を開始します。DBMS_RULE.EVALUATE プロシージャのコール時にクライアントによって指定される情報には、次の項目が含まれます。

- イベントの評価に使用するルールを含むルール・セットの名前。
- 評価に使用する評価コンテキスト。指定した評価コンテキストを使用するルールのみが評価されます。
- 表の値および変数値。表の値にはその表の行のデータを参照する ROWID が含まれ、変数値には明示的変数のデータが含まれます。暗黙的変数に指定された値は、変数値の評価ファンクションを使用して取得される値をオーバーライドします。指定された変数に属性がある場合、クライアントはその変数全体に対する 1 つの値を送信するか、またはその変数の任意の数の属性の値を送信できます。ただし、変数全体に対する値が指定されている場合は、クライアントは属性の値を指定できません。
- オプションの**イベント・コンテキスト**。イベント・コンテキストは、名前 / 値ペアを含む SYS.RE$NV_LIST 型の配列で、イベントに関する情報を含みます。このオプションの情報がルール・エンジンによって直接使用または解析されることはありません。かわりに、評価ファンクション、変数値の評価ファンクション（暗黙的変数用）、および変数メソッド・ファンクションなど、クライアントのコールバックに渡されます。

また、クライアントは、イベントに関する情報、および DBMS_RULE.EVALUATE プロシージャを使用したイベントの評価方法に関する情報を送信することもできます。たとえば、最初の TRUE ルールまたは最初の MAYBE ルール（TRUE ルールがない場合）が検出された時点で評価を停止する必要があるかどうかを、コール元が指定する場合があります。

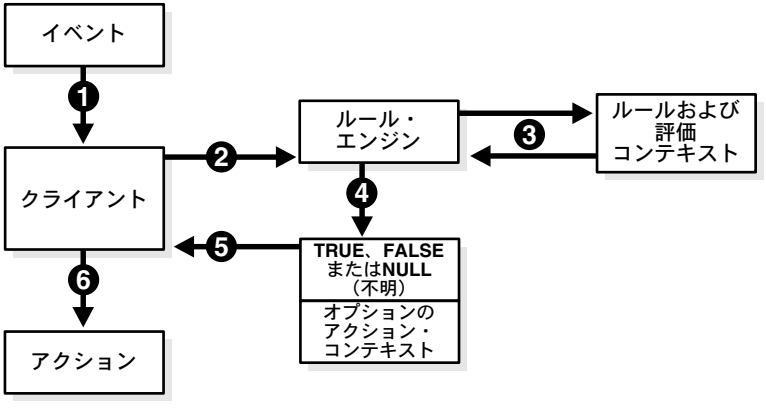
ルール・エンジンは、指定されたルール・セット内のルールを使用してイベントを評価します。次に、ルール・エンジンからクライアントに結果が戻されます。ルールが戻されるときには、EVALUATE プロシージャの 2 つの OUT パラメータ true_rules および maybe_rules が使用されます。true_rules パラメータでは TRUE と評価されるルールが戻され、オプションで、maybe_rules パラメータではさらに情報を与えると TRUE と評価される可能性があるルールが戻されます。

ルール・セットの評価プロセス

図 5-1 に、ルール・セットの評価プロセスを示します。

1. クライアント定義のイベントが発生します。
2. クライアントが DBMS_RULE.EVALUATE プロシージャを実行し、イベントをルール・エンジンに送信します。
3. ルール・エンジンが、ルール・セット内のルールおよび関連する評価コンテキストに基づいてイベントを評価します。クライアントが、DBMS_RULE.EVALUATE プロシージャのコールでルール・セットと評価コンテキストの両方を指定します。評価に使用されるのは、指定されたルール・セット内に存在し、指定された評価コンテキストを使用するルールのみです。
4. ルール・エンジンが評価の結果を取得します。各ルールは、TRUE、FALSE または NULL（不明）と評価されます。
5. ルール・エンジンが、TRUE と評価されたルールをクライアントに戻します。各ルールとともにアクション・コンテキスト全体が戻されます。アクション・コンテキストには、情報が含まれている場合と NULL の場合があります。
6. クライアントが、ルール・エンジンから戻された結果に基づいてアクションを実行します。ルール・エンジンでは、ルール評価に基づくアクションは実行されません。

図 5-1 ルール・セットの評価



関連項目：

- DBMS_RULE.EVALUATE プロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- Streams クライアントおよび maybe_rules の詳細は、6-20 ページの「Streams クライアントについて NULL と評価される、ルール条件内の未定義変数」を参照してください。

部分評価

部分評価は、指定された評価コンテキストのすべての表および変数にデータがない状態で DBMS_RULE.EVALUATE プロシージャが実行される場合に発生します。部分評価の実行中に、使用不可能な列、変数または属性を参照するルールもあれば、使用可能なデータのみを参照するルールもあります。

例として、評価中に使用可能なデータが次のものにかぎられる場合を考えます。

- 列 tab1.col=7
- 属性 v1.a1='ABC'

次のルールが評価に使用されます。

- ルール R1 には、次の条件があります。

```
(tab1.col = 5)
```

- ルール R2 には、次の条件があります。

```
(:v1.a2 > 'aaa')
```

- ルール R3 には、次の条件があります。

```
(:v1.a1 = 'ABC') OR (:v2 = 5)
```

- ルール R4 には、次の条件があります。

```
(:v1.a1 = UPPER('abc'))
```

この使用例では、R1 および R4 は使用可能なデータを、R2 は使用不可能なデータを、R3 は使用可能なデータおよび使用不可能なデータを参照します。

部分評価は常に、ルール内の単純な条件のみを評価します。ルール条件に単純でない部分が含まれている場合は、データがどの程度使用可能であるかに応じて、ルールが完全に評価される場合と評価されない場合があります。ルールが完全に評価されない場合、それを MAYBE ルールとして戻すことができます。

たとえば、前述の使用例では、R1 と、R3 の最初の部分が評価されますが、R2 および R4 は評価されません。クライアントには次の結果が戻されます。

- R1 は FALSE と評価されるため、戻されません。
- R2 は、属性 v1.a2 に関する情報が使用可能でないため、MAYBE として戻されます。
- R3 は単純なルールであり、v1.a1 の値がルール条件の最初の部分と一致するため、R3 は TRUE として戻されます。
- R4 は、ルール条件が単純ではないため、MAYBE として戻されます。このルールを TRUE または FALSE として評価するには、クライアントが変数 v1 の値を指定する必要があります。

関連項目： 5-3 ページ「単純ルール条件」

ルールに関連するデータベース・オブジェクトと権限

DBMS_RULE_ADM パッケージを使用して、次の型のデータベース・オブジェクトを直接作成できます。

- 評価コンテキスト
- ルール
- ルール・セット

DBMS_STREAMS_ADM パッケージを使用すると、ルールとルール・セットを間接的に作成できます。これらのデータベース・オブジェクトに対する権限を制御するには、DBMS_RULE_ADM パッケージの次のプロシージャを使用します。

- GRANT_OBJECT_PRIVILEGE
- GRANT_SYSTEM_PRIVILEGE
- REVOKE_OBJECT_PRIVILEGE
- REVOKE_SYSTEM_PRIVILEGE

ユーザーに対して独自スキーマ内でのルール・セット、ルールおよび評価コンテキストの作成を許可するには、そのユーザーに次のシステム権限を付与します。

- CREATE_RULE_SET_OBJ
- CREATE_RULE_OBJ
- CREATE_EVALUATION_CONTEXT_OBJ

これらの権限と後述する権限は、ユーザーに直接付与するか、ロールを介して付与できます。

注意： ANY オブジェクト（ALTER_ANY_RULE など）に対する権限を付与する場合に、初期化パラメータ O7_DICTIONARY_ACCESSIBILITY を FALSE に設定すると、SYS スキーマを除く全スキーマにある指定した型のオブジェクトへのアクセス権をユーザーに付与できます。デフォルトでは、初期化パラメータ O7_DICTIONARY_ACCESSIBILITY は FALSE に設定されます。

SYS スキーマ内のオブジェクトへのアクセス権を付与する必要がある場合は、オブジェクトに対するオブジェクト権限を明示的に付与できます。または、O7_DICTIONARY_ACCESSIBILITY 初期化パラメータを TRUE に設定することもできます。この場合、ANY オブジェクトについて付与される権限では、SYS を含むすべてのスキーマへのアクセスが許可されます。

関連項目：

- これらのデータベース・オブジェクトの詳細は、5-2 ページの「ルールの構成要素」を参照してください。
- これらのデータベース・オブジェクトに対するシステム権限とオブジェクト権限の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- ユーザー権限の概要は、『Oracle9*i* データベース概要』および『Oracle9*i* データベース管理者ガイド』を参照してください。
- DBMS_STREAMS_ADM パッケージを使用してルールとルール・セットを間接的に作成する方法の詳細は、第 6 章「Streams でのルールの使用方法」を参照してください。

ルール関連のデータベース・オブジェクトを作成するための権限

ユーザーがスキーマ内で評価コンテキスト、ルールまたはルール・セットを作成するには、次の条件を少なくとも 1 つは満たしている必要があります。

- スキーマがユーザー独自のスキーマであり、作成するデータベース・オブジェクトの型に対する CREATE システム権限を付与されていること。たとえば、ユーザーが独自スキーマ内でルール・セットを作成するには、CREATE_RULE_SET_OBJ システム権限を付与されている必要があります。
- 作成するデータベース・オブジェクトの型に対する CREATE ANY システム権限を付与されていること。たとえば、ユーザーが任意のスキーマ内で評価コンテキストを作成するには、CREATE_ANY_EVALUATION_CONTEXT システム権限を付与されている必要があります。

注意： 評価コンテキストを伴うルールを作成する場合は、ルール所有者に評価コンテキストでアクセスするすべてのオブジェクトに対する権限が付与されていること。

ルール関連のデータベース・オブジェクトを変更するための権限

ユーザーが評価コンテキスト、ルールまたはルール・セットを変更するには、次の条件を少なくとも 1 つは満たしている必要があります。

- データベース・オブジェクトの所有者であること。
- データベース・オブジェクトが他のユーザーのスキーマにある場合は、データベース・オブジェクトに対する ALTER OBJECT 権限を付与されていること。たとえば、ユーザーが他のユーザーのスキーマにあるルール・セットを変更するには、そのルール・セットに対する ALTER_ON_RULE_SET オブジェクト権限を付与されている必要があります。
- データベース・オブジェクトに対する ALTER ANY システム権限を付与されていること。たとえば、ユーザーが任意のスキーマにあるルールを変更するには、ALTER_ANY_RULE システム権限を付与されている必要があります。

ルール関連のデータベース・オブジェクトを削除するための権限

ユーザーが評価コンテキスト、ルールまたはルール・セットを削除するには、次の条件を少なくとも 1 つは満たしている必要があります。

- データベース・オブジェクトの所有者であること。
- データベース・オブジェクトに対する DROP ANY システム権限を付与されていること。たとえば、ユーザーが任意のスキーマにあるルール・セットを削除するには、DROP_ANY_RULE_SET システム権限を付与されている必要があります。

ルール・セットにルールを含めるための権限

この項では、ルール・セットにルールを含めるために必要な権限について説明します。

ユーザーは、ルールについて次の条件を少なくとも 1 つは満たしている必要があります。

- ルールの所有者であること。
- ルールが他のユーザーのスキーマにある場合は、そのルールに対する EXECUTE OBJECT 権限を付与されていること。たとえば、ユーザーが hr スキーマ内でルール・セットにルール depts を含めるには、hr.depts ルールに対する EXECUTE_ON_RULE 権限を付与されている必要があります。
- ルールに対する EXECUTE ANY システム権限を付与されていること。たとえば、ルール・セットに任意のルールを含めるには、EXECUTE_ANY_RULE システム権限を付与されている必要があります。

また、ユーザーはルール・セットについて次の条件を少なくとも 1 つは満たしている必要があります。

- ルール・セットの所有者であること。
- ルール・セットが他のユーザーのスキーマにある場合は、そのルール・セットに対する ALTER OBJECT 権限を付与されていること。たとえば、ユーザーが hr スキーマ内で human_resources ルール・セットにルールを含めるには、hr.human_resources ルール・セットに対する ALTER_ON_RULE_SET 権限を付与されている必要があります。
- ルール・セットに対する ALTER ANY システム権限を付与されていること。たとえば、任意のルール・セットにルールを含めるには、ALTER_ANY_RULE_SET システム権限を付与されている必要があります。

ルール・セットを評価するための権限

ルール・セットを評価するには、ユーザーは次の条件を少なくとも 1 つは満たしている必要があります。

- ルール・セットの所有者であること。
- ルール・セットが他のユーザーのスキーマにある場合は、そのルール・セットに対する EXECUTE OBJECT 権限を付与されていること。たとえば、ユーザーが hr スキーマ内で human_resources ルール・セットを評価するには、hr.human_resources ルール・セットに対する EXECUTE_ON_RULE_SET 権限を付与されている必要があります。
- ルール・セットに対する EXECUTE ANY システム権限を付与されていること。たとえば、ユーザーが任意のルール・セットを評価するには、EXECUTE_ANY_RULE_SET システム権限を付与されている必要があります。

ルール・セットに対する EXECUTE オブジェクト権限を付与するには、権限付与者の EXECUTE 権限で、現在ルール・セットに含まれているすべてのルールについて WITH GRANT OPTION を指定する必要があります。

評価コンテキストを使用するための権限

評価コンテキストを使用するには、ユーザーはその評価コンテキストについて次の条件を少なくとも 1 つは満たしている必要があります。

- 評価コンテキストの所有者であること。
- 評価コンテキストが他のユーザーのスキーマにある場合は、その評価コンテキストに対する EXECUTE_ON_EVALUATION_CONTEXT 権限を付与されていること。
- 評価コンテキストに対する EXECUTE_ANY_EVALUATION_CONTEXT システム権限を付与されていること。

6

Streams でのルールの使用方法

この章では、Streams でのルールの使用方法について説明します。

この章の内容は次のとおりです。

- Streams でのルールの使用方法の概要
- システム作成ルール
- Streams の評価コンテキスト
- Streams とイベント・コンテキスト
- Streams およびアクション・コンテキスト
- ユーザー作成ルール、ルール・セットおよび評価コンテキスト
- ルールベースの変換

関連項目：

- ルールの詳細は、第 5 章「ルール」を参照してください。
- 第 15 章「ルールおよびルールベースの変換の管理」

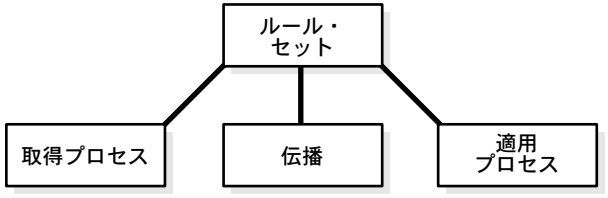
Streams でのルールの使用方法の概要

Streams では、次の各メカニズムがルール・セットに関連付けられている場合にルール・エンジンのクライアントとなります。

- 取得プロセス
- 伝播
- 適用プロセス

これらの各メカニズムは、1 つのルール・セットにのみ関連付けることができます。ただし、1 つのルール・セットを同じデータベース内の複数の取得プロセス、伝播および適用プロセスで使用できます。図 6-1 に、1 つのルール・セットをルール・エンジンの複数のクライアントで使用する方法を示します。

図 6-1 1 つのルール・セットをルール・エンジンの複数のクライアントで使用可能



特に、Streams でルール・セットを使用すると次のことができます。

- 取得プロセスで REDO ログから取得する変更を指定します。つまり、REDO ログ内で検出された変更が原因で、取得プロセスに関連付けられているルール・セット内のルールが 1 つでも TRUE と評価されると、その変更が取得プロセスによって取得されます。
- あるキューから別のキューに伝播で伝播させるイベントを指定します。つまり、キュー内のイベントが原因で、伝播に関連付けられているルール・セット内のルールが 1 つでも TRUE と評価されると、そのイベントが伝播によって伝播させられます。
- 適用プロセスでキューから取り出されるイベントを指定します。つまり、キュー内のイベントが原因で、適用プロセスに関連付けられているルール・セット内のルールが 1 つでも TRUE と評価されると、そのイベントが適用プロセスによって取り出され、処理されます。

伝播または適用プロセスの場合、ルール・セットに対して評価されるイベントは、取得イベントまたはユーザー・エンキュー・イベントのいずれかです。

競合するルールがメカニズムに関連付けられている場合は、どちらかのルールが TRUE と評価されると、メカニズムによってタスクが実行されます。たとえば、取得プロセスに関連付けられたルール・セットに、hr.employees 表に対する DML 変更を取得するように取得プロセスに指示する 1 つのルールが含まれている場合、同じルール・セットに hr.employees 表に対する DML 変更を取得しないように取得プロセスに指示する別のルールが含まれていても、取得プロセスでは hr.employees 表に対する DML 変更が取得されます。

システム作成ルール

Streams では、ルールに基づいて次の 3 つのタスクが実行されます。

- 取得プロセスを使用した変更の取得
- 伝播を使用した変更の伝播
- 適用プロセスを使用した変更の適用

システム作成ルールでは、タスクの粒度レベルとして表、スキーマまたはグローバルのいずれかが指定されます。この項では、各レベルについて説明します。特定のタスクに複数のレベルを指定できます。たとえば、1 つの適用プロセスに対して、oe スキーマ内で特定の表に表レベルの適用を実行し、hr スキーマ全体にスキーマ・レベルの適用を実行するように指示できます。

表 6-1 に、各 Streams タスクに対して各レベルのルールが持つ意味について説明します。

表 6-1 タスクのタイプとルールのレベル

タスク	表ルール	スキーマ・ルール	グローバル・ルール
取得	指定した表について REDO ログ内の変更を取得し、論理変更レコード（LCR）に変換してエンキューします。	指定したスキーマ内のデータベース・オブジェクトについて REDO ログ内の変更を取得し、LCR に変換してエンキューします。	データベース内のすべてのデータベース・オブジェクトに対する変更を取得し、LCR に変換してエンキューします。
伝播	ソース・キュー内の指定した表に関連する LCR を、宛先キューに伝播します。	ソース・キュー内の指定したスキーマにあるデータベース・オブジェクト関連の LCR を、宛先キューに伝播します。	ソース・キューにあるすべての変更を宛先キューに伝播します。
適用	指定した表に関連する、キュー内のすべての LCR またはサブセットを適用します。	指定したスキーマ内のデータベース・オブジェクトに関連する、キュー内の LCR を適用します。	キュー内のすべての LCR を適用します。

DBMS_STREAMS_ADM パッケージのプロシージャを使用すると、これらの各レベルでルールを作成できます。表 6-2 に、DBMS_STREAMS_ADM パッケージによって作成されるルール内に指定するシステム作成ルールの条件のタイプを示します。

表 6-2 DBMS_STREAMS_ADM パッケージによって作成されるシステム作成ルールの条件

ルールの条件が TRUE に評価される対象	Streams のメカニズム	作成に使用するプロシージャ
特定の表に対するすべての DML 変更	取得	ADD_TABLE_RULES
特定の表に対するすべての DDL 変更	取得	ADD_TABLE_RULES
特定のスキーマにあるすべての表に対するすべての DML 変更	取得	ADD_SCHEMA_RULES
特定のスキーマにあるすべてのデータベース・オブジェクトに対するすべての DDL 変更	取得	ADD_SCHEMA_RULES
特定のデータベースにあるすべての表に対するすべての DML 変更	取得	ADD_GLOBAL_RULES
特定のデータベースにあるすべてのデータベース・オブジェクトに対するすべての DDL 変更	取得	ADD_GLOBAL_RULES
特定の表に対する DML 変更を含むすべての LCR	伝播	ADD_TABLE_PROPAGATION_RULES
特定の表に対する DDL 変更を含むすべての LCR	伝播	ADD_TABLE_PROPAGATION_RULES
特定のスキーマにある表に対する DML 変更を含むすべての LCR	伝播	ADD_SCHEMA_PROPAGATION_RULES
特定のスキーマにあるデータベース・オブジェクトに対する DDL 変更を含むすべての LCR	伝播	ADD_SCHEMA_PROPAGATION_RULES
特定のキュー内の DML 変更を含むすべての LCR	伝播	ADD_GLOBAL_PROPAGATION_RULES
特定のキュー内の DDL 変更を含むすべての LCR	伝播	ADD_GLOBAL_PROPAGATION_RULES
特定の表の行のサブセットに対する DML 変更を含むすべての LCR	適用	ADD_SUBSET_RULES
特定の表に対する DML 変更を含むすべての LCR	適用	ADD_TABLE_RULES
特定の表に対する DDL 変更を含むすべての LCR	適用	ADD_TABLE_RULES
特定のスキーマにある表に対する DML 変更を含むすべての LCR	適用	ADD_SCHEMA_RULES

表 6-2 DBMS_STREAMS_ADM パッケージによって作成されるシステム作成ルールの条件（続き）

ルールの条件が TRUE に評価される対象	Streams のメカニズム	作成に使用するプロシージャ
特定のスキーマにあるデータベース・オブジェクトに対する DDL 変更を含むすべての LCR	適用	ADD_SCHEMA_RULES
特定のキュー内の DML 変更を含むすべての LCR	適用	ADD_GLOBAL_RULES
特定のキュー内の DDL 変更を含むすべての LCR	適用	ADD_GLOBAL_RULES

表 6-2 に示した各プロシージャでは、次の操作が実行されます。

- 取得プロセス、伝播または適用プロセスは、存在しない場合には作成されます。
- 指定した取得プロセス、伝播または適用プロセスのルール・セットは、存在しない場合には作成されます。
- 0 以上のルールが作成され、指定の取得プロセス、伝播または適用プロセスのルール・セットに追加されます。

前述のプロシージャによって作成されたすべてのルール・セットとルールでは、Streams 環境用に Oracle が提供する評価コンテキスト SYS.STREAMS$_EVALUATION_CONTEXT が使用されます。

ADD_SUBSET_RULES を除き、前述のプロシージャではルールが作成されないか、1 つまたは 2 つのルールが作成されます。Streams タスクを DML 変更または DDL 変更についてのみ実行する必要がある場合は、ルールを 1 つのみ作成します。ただし、Streams タスクを DML 変更と DDL 変更の両方について実行する必要がある場合は、ルールを個別に作成します。ここで表の DML ルールを作成すると、前に作成した DML ルールを変更せずに、後で同じ表の DDL ルールを作成できます。これは、先に表の DDL ルールを作成してから、後で同じ表の DML ルールを作成する場合も同じです。

ADD_SUBSET_RULES プロシージャでは、表に対する 3 種類の DML 操作である INSERT、UPDATE および DELETE について、常に 3 つのルールが作成されます。ADD_SUBSET_RULES プロシージャでは、表に対する DDL 変更のルールは作成されません。ADD_TABLE_RULES プロシージャを使用すると、表の DDL ルールを作成できます。

取得イベントの伝播ルールを作成する場合は、変更のソース・データベースを指定することをお薦めします。適用プロセスでは、トランザクション制御イベントを使用して、取得イベントがコミット済みトランザクションにアセンブルされます。COMMIT や ROLLBACK などのトランザクション制御イベントには、イベントが発生したソース・データベースの名前が含まれています。これらのイベントが意図に反して循環してしまうことを避けるには、伝播ルールにソース・データベースを指定して条件を含める必要があります。そのためには、伝播ルールを作成するときにソース・データベースを指定します。

次の項では、表ルール、スキーマ・ルールおよびグローバル・ルールについて詳しく説明します。

注意： NOT または OR 条件演算子を使用するルールなど、より複雑なルール条件を指定してルールを作成するには、DBMS_RULE_ADM パッケージを使用します。

関連項目：

- DBMS_STREAMS_ADM パッケージと DBMS_RULE_ADM パッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 6-14 ページ「Streams の評価コンテキスト」
- 2-2 ページ「論理変更レコード（LCR）」
- 6-18 ページ「複合ルール条件」

表ルールとサブセット・ルール

ルールを使用して個々の表にのみ関連する Streams タスクを指定するときには、表レベルのルールを指定します。DML 変更に関する表レベルのルール、DDL 変更に関する表レベルのルールまたは特定の表への両方のタイプの変更に関する 2 つのルールを指定できます。

サブセット・ルールは DML 変更に関する表レベルの特殊なルールであり、ADD_SUBSET_RULES プロシージャを使用して作成します。ADD_SUBSET_RULES プロシージャを使用すると、SELECT 文の WHERE 句に似た条件に基づいて、適用プロセスで特定の表に関連する行の論理変更レコード（LCR）のサブセットのみを適用するように指定できます。そのため、ADD_SUBSET_RULES プロシージャでは、適用プロセスに対して表のうち特定の行のみをメンテナンスするように指定できます。

注意： 1 つ以上の LOB 列を含む表に対するサブセット・ルールの作成は、サポートされていません。

関連項目： サブセット・ルールの詳細は、4-12 ページの「Streams ルールを使用した行のサブセット化」を参照してください。

表レベルのルールの例

DBMS_STREAMS_ADM パッケージのプロシージャを使用し、Streams の適用プロセスの動作を次のように指示する場合を考えます。

- hr.locations 表に対するすべての DML 変更の適用
- hr.countries 表に対するすべての DDL 変更の適用
- hr.regions 表に対する DML 変更のサブセットの適用

hr.locations 表に対するすべての DML 変更の適用 これらの変更が dbs1.net ソース・データベースで発生したとします。

ADD_TABLE_RULES プロシージャを実行して次のルールを作成します。

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name               =>  'hr.locations',
    streams_type             =>  'apply',
    streams_name             =>  'apply',
    queue_name               =>  'streams_queue',
    include_dml              =>  true,
    include_ddl              =>  false,
    include_tagged_lcr       =>  false,
    source_database          =>  'dbs1.net');
END;
/
```

ADD_TABLE_RULES プロシージャでは、次のようなルール条件を持つルールが作成されます。

```
(((:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'LOCATIONS'))
and :dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET' )
```

:dml で始まる各条件は変数です。値は、評価される行 LCR 用に指定したメンバー関数のコールによって決定されます。したがって、前述の例の :dml.get_object_owner() は、評価される行 LCR 用の GET_OBJECT_OWNER メンバー関数のコールです。

また、DBMS_STREAMS_ADM パッケージのプロシージャによって作成されるすべての DML ルールには、デフォルトで次の条件が含まれます。

```
:dml.is_null_tag() = 'Y'
```

DDL ルールの条件は、次のとおりです。

```
:ddl.is_null_tag() = 'Y'
```

これらの条件は、取得プロセスの場合、取得プロセスが変更を取得するには、REDO レコード内のタグが NULL である必要があることを示します。伝播の場合は、伝播が LCR を伝播するには、LCR 内のタグが NULL である必要があることを示します。適用プロセスの場合は、適用プロセスが LCR を適用するには、LCR 内のタグが NULL である必要があることを示します。DBMS_STREAMS_ADM パッケージのプロシージャを実行するときに、include_tagged_lcr パラメータに true を指定すると、この条件をシステム作成ルール内で省略できます。

関連項目：

- 条件内の変数の詳細は、第 5 章「ルール」を参照してください。
- タグの詳細は、第 8 章「Streams のタグ」を参照してください。

hr.countries 表に対するすべての DDL 変更の適用 これらの変更が dbs1.net ソース・データベースで発生したとします。

ADD_TABLE_RULES プロシージャを実行して次のルールを作成します。

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name               =>  'hr.countries',
    streams_type             =>  'apply',
    streams_name             =>  'apply',
    queue_name               =>  'streams_queue',
    include_dml              =>  false,
    include_ddl              =>  true,
    include_tagged_lcr       =>  false,
    source_database          =>  'dbs1.net');
END;
/
```

ADD_TABLE_RULES プロシージャでは、次のようなルール条件を持つルールが作成されます。

```
(((:ddl.get_object_owner() = 'HR' and :ddl.get_object_name() = 'COUNTRIES')
or (:ddl.get_base_table_owner() = 'HR'
and :ddl.get_base_table_name() = 'COUNTRIES'))
and :ddl.is_null_tag() = 'Y' and :ddl.get_source_database_name() = 'DBS1.NET' )
```

:ddl で始まる各条件は変数です。値は、評価される DDL LCR 用に指定したメンバー関数のコールによって決定されます。したがって、前述の例の :ddl.get_object_owner() は、評価される DDL LCR 用の GET_OBJECT_OWNER メンバー関数のコールです。

hr.regions 表に対する DML 変更のサブセットの適用 次の例では、Streams の適用プロセスに対して、region_id が 2 になっている hr.regions 表に対する DML 変更のサブセットを適用するように指示します。これらの変更が dbs1.net ソース・データベースで発生したとします。

ADD_SUBSET_RULES プロシージャを実行して 3 つのルールを作成します。

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SUBSET_RULES(
    table_name               =>  'hr.regions',
    dml_condition            =>  'region_id=2',
    streams_type             =>  'apply',
    streams_name             =>  'apply',
    queue_name               =>  'streams_queue',
    include_tagged_lcr       =>  false,
    source_database          =>  'dbs1.net');
END;
/
```

ADD_SUBSET_RULES プロシージャによって、INSERT 操作用、UPDATE 操作用および DELETE 操作用に 1 つずつ、合計 3 つのルールが作成されます。

挿入ルールで使用されるルール条件を次に示します。

```
:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'
AND :dml.get_command_type() IN ('UPDATE','INSERT')
AND (:dml.get_value('NEW','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2)
AND (:dml.get_command_type()='INSERT'
OR ((:dml.get_value('OLD','"REGION_ID"') IS NOT NULL)
AND NOT EXISTS (SELECT 1 FROM SYS.DUAL
WHERE (:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2))))
```

このルール条件に基づいて、LCR が次のように評価されます。

- 挿入の場合は、LCR 内で region_id の新規の値が 2 であれば、挿入が適用されます。
- 挿入の場合は、LCR 内で region_id の新規の値が 2 でないか、または NULL であれば、挿入がフィルタで除外されます。
- 更新の場合は、LCR 内で region_id の古い値が 2 でないか、または NULL であり、新しい値が 2 であれば、更新は挿入に変換されてから適用されます。

更新ルールで使用されるルール条件を次に示します。

```
:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'
AND :dml.get_command_type()='UPDATE'
AND (:dml.get_value('NEW','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2)
AND (:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2)
```

このルール条件に基づいて、LCR が次のように評価されます。

- 更新の場合は、LCR 内で region_id の古い値と新規の値の両方が 2 であれば、更新がそのまま適用されます。
- 更新の場合は、LCR 内で region_id の古い値または新規の値のどちらかが 2 でないか、または NULL であれば、その更新は更新ルールを満たしていないことになります。LCR は挿入ルールを満たす場合、削除ルールを満たす場合またはどちらのルールも満たさない場合があります。

削除ルールで使用されるルール条件を次に示します。

```
:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'
AND :dml.get_command_type() IN ('UPDATE','DELETE')
AND (:dml.get_value('OLD','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2)
AND (:dml.get_command_type()='DELETE'
OR ((:dml.get_value('NEW','"REGION_ID"') IS NOT NULL)
AND NOT EXISTS (SELECT 1 FROM SYS.DUAL
WHERE (:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2))))
```

このルール条件に基づいて、LCR が次のように評価されます。

- 削除の場合は、LCR 内で region_id の古い値が 2 であれば、削除が適用されます。
- 削除の場合は、LCR 内で region_id の古い値が 2 でないか、または NULL であれば、削除はフィルタで除外されます。
- 更新の場合は、LCR 内で region_id の古い値が 2 で、かつ、新規の値が 2 以外の値または NULL であれば、更新は削除に変換されてから適用されます。

ルールのまとめ この例では、次の表とサブセット・ルールを定義しました。

- hr.locations 表に対する DML 操作が実行されると TRUE と評価される表ルール。
- hr.countries 表に対する DDL 操作が実行されると TRUE と評価される表ルール。
- INSERT 操作によって region_id が 2 の 1 行が挿入されるか、更新操作によって行の region_id が 2 以外の値または NULL から値 2 に変更されると、TRUE と評価されるサブセット・ルール。
- UPDATE 操作によって 1 行が更新され、更新の前後両方の region_id が 2 である場合に TRUE と評価されるサブセット・ルール。
- DELETE 操作によって region_id が 2 の 1 行が削除されるか、更新操作によって行の region_id が 2 から 2 以外の値または NULL に変更されると、TRUE と評価されるサブセット・ルール。

次のリストに、これらのルールを使用した場合に適用プロセスによって適用される変更の例を示します。

- hr.locations 表への 1 行の挿入。
- hr.locations 表からの 5 行の削除。
- hr.countries 表への 1 列の追加。
- hr.regions 表の 1 行の更新。この場合、region_id は 2 で、region_id の新規の値は 1 です。この更新は削除に変換されます。

適用プロセスでは、これらの変更が関連付けられたキューからデキューされ、接続先データベースのデータベース・オブジェクトに適用されます。

次のリストに、前述と同じルールを使用した場合に適用プロセスによって無視される変更の例を示します。

- hr.employees 表への 1 行の挿入。hr.employees 表はどのルールも満たしていないため、この変更は適用されません。
- hr.countries 表の 1 行の更新。これは、DDL 変更ではなく DML 変更です。hr.countries 表のルールは DDL 変更のみを対象としているため、この変更は適用されません。
- hr.locations 表への 1 列の追加。これは、DML 変更ではなく DDL 変更です。hr.locations 表のルールは DML 変更のみを対象としているため、この変更は適用されません。
- hr.regions 表の 1 行の更新。この場合、更新前の region_id は 1 で、更新後も 1 のままです。hr.regions 表のサブセット・ルールが TRUE と評価されるのは、region_id の新旧いずれか（またはその両方）の値が 2 の場合のみであるため、この変更は適用されません。

スキーマ・ルール

ルールを使用してスキーマ関連の Streams タスクを指定する場合は、スキーマ・レベルのルールを指定します。また、Streams タスクは、現在スキーマにあるデータベース・オブジェクトのいずれかが変更されるか、将来スキーマに追加されるデータベース・オブジェクトが変更されると実行されます。DML 変更に関するスキーマ・レベルのルール、DDL 変更に関するスキーマ・レベルのルールまたはスキーマ内のオブジェクトへの両方のタイプの変更に関する 2 つのルールを指定できます。

スキーマ・レベルのルールの例

DBMS_STREAMS_ADM パッケージの ADD_SCHEMA_PROPAGATION_RULES プロシージャを使用し、Streams 伝播に対して、hr スキーマでの DML 変更と DDL 変更を含む LCR を、dbs1.net データベースのキューから dbs2.net データベースのキューに伝播させるように指示する場合を考えます。

ADD_SCHEMA_PROPAGATION_RULES プロシージャを実行して各ルールを作成します。

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name              =>  'hr',
    streams_name             =>  'dbs1_to_dbs2',
    source_queue_name        =>  'streams_queue',
    destination_queue_name   =>  'streams_queue@dbs2.net',
    include_dml              =>  true,
    include_ddl              =>  true,
    include_tagged_lcr       =>  false,
    source_database          =>  'dbs1.net');
END;
/
```

ADD_SCHEMA_PROPAGATION_RULES プロシージャによって、行 LCR（DML 変更を含む）用に 1 つと DDL LCR 用に 1 つ、合計 2 つのルールが作成されます。

行 LCR のルールで使用されるルール条件を次に示します。

```
(:dml.get_object_owner() = 'HR' and :dml.is_null_tag() = 'Y'
and :dml.get_source_database_name() = 'DBS1.NET' )
```

DDL LCR のルールで使用されるルール条件を次に示します。

```
(:ddl.get_object_owner() = 'HR' and :ddl.is_null_tag() = 'Y'
and :ddl.get_source_database_name() = 'DBS1.NET' )
```

次のリストに、これらのルールを使用した場合に伝播によって伝播される変更の例を示します。

- hr.countries 表への 1 行の挿入
- hr.loc_city_ix 索引の変更
- hr.employees 表の切捨て
- hr.countries 表への 1 列の追加
- hr.update_job_history トリガーの変更
- hr スキーマ内での新規表 candidates の作成
- hr.candidates 表への 20 行の挿入

伝播では、前述のすべての変更を含む LCR が、ソース・キューから宛先キューに伝播されます。

ここで、同じルールを使用して、oe.inventories 表に 1 行が挿入される場合を考えます。oe スキーマはスキーマ・レベルのルールで指定されておらず、oe.inventories 表は表レベルのルールで指定されていないため、この変更は無視されます。

グローバル・ルール

ルールを使用してデータベース全体またはキュー全体に関連する Streams タスクを指定するときには、グローバル・レベルのルールを指定します。DML 変更に関するグローバル・ルール、DDL 変更に関するグローバル・ルールまたは両方のタイプに関する 2 つのルールを指定できます。

取得プロセスの場合、1 つのグローバル・ルールは、ソース・データベースに対するすべての DML 変更またはすべての DDL 変更を取得することを意味します。伝播の場合は、ソース・キュー内のすべての行 LCR またはすべての DDL LCR を宛先キューに伝播させることを意味します。適用プロセスの場合は、指定したソース・データベースについて、キュー内にあるすべての行 LCR またはすべての DDL LCR を適用することを意味します。

グローバル・レベルのルールの例

DBMS_STREAMS_ADM パッケージの ADD_GLOBAL_RULES プロシージャを使用し、Streams の取得プロセスに対してデータベース内の DML 変更と DDL 変更をすべて取得するように指示する場合を考えます。

ADD_GLOBAL_RULES プロシージャを実行して各ルールを作成します。

```
BEGIN DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
   streams_type             =>  'capture',
   streams_name             =>  'capture',
   queue_name               =>  'streams_queue',
   include_dml              =>  true,
   include_ddl              =>  true,
   include_tagged_lcr       =>  false,
   source_database          =>  NULL);
END;
/
```

取得ルールが作成中であるため、source_database パラメータに NULL を指定できます。ADD_GLOBAL_RULES を使用して適用ルールを作成する場合は、ソース・データベース名を指定します。

ADD_GLOBAL_RULES プロシージャによって、行 LCR（DML 変更を含む）用に 1 つと DDL LCR 用に 1 つ、合計 2 つのルールが作成されます。

行 LCR のルールで使用されるルール条件を次に示します。

```
(( :dml.get_source_database_name()>=' ' OR :dml.get_source_database_name()<=' ')
and :dml.is_null_tag() = 'Y' )
```

DDL LCR のルールで使用されるルール条件を次に示します。

```
(( :ddl.get_source_database_name()>=' ' OR :ddl.get_source_database_name()<=' ')
and :ddl.is_null_tag() = 'Y' )
```

これらのルールを使用すると、取得プロセスではデータベースに対するサポート対象の DML 変更と DDL 変更がすべて取得されます。グローバル・システム生成ルール内のソース・データベース名に関連する条件は、ルール評価のパフォーマンスを改善するために示されます。

注意： 取得プロセスは、一部の DML 変更や DDL 変更は取得せず、SYS または SYSTEM スキーマ内で行われた変更も取得しません。

関連項目： 取得プロセスと、取得プロセスによって取得される DML 文および DDL 文の詳細は、第 2 章「Streams の取得プロセス」を参照してください。

Streams の評価コンテキスト

システム作成のルール・セットおよびルールでは、SYS スキーマ内の STREAMS$_EVALUATION_CONTEXT という組込み評価コンテキストが使用されます。PUBLIC には、この評価コンテキストの EXECUTE 権限が付与されます。

Oracle のインストール中に、次の文によって Streams の評価コンテキストが作成されます。

```
DECLARE
  vt  SYS.RE$VARIABLE_TYPE_LIST;
BEGIN
  vt := SYS.RE$VARIABLE_TYPE_LIST(
    SYS.RE$VARIABLE_TYPE('DML', 'SYS.LCR$_ROW_RECORD',
       'SYS.DBMS_STREAMS_INTERNAL.ROW_VARIABLE_VALUE_FUNCTION',
       'SYS.DBMS_STREAMS_INTERNAL.ROW_FAST_EVALUATION_FUNCTION'),
    SYS.RE$VARIABLE_TYPE('DDL', 'SYS.LCR$_DDL_RECORD',
       'SYS.DBMS_STREAMS_INTERNAL.DDL_VARIABLE_VALUE_FUNCTION',
       'SYS.DBMS_STREAMS_INTERNAL.DDL_FAST_EVALUATION_FUNCTION'));
  DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
    evaluation_context_name => 'SYS.STREAMS$_EVALUATION_CONTEXT',
    variable_types          => vt,
    evaluation_function     =>
                      'SYS.DBMS_STREAMS_INTERNAL.EVALUATION_CONTEXT_FUNCTION');
END;
/
```

この文には、SYS.DBMS_STREAM_INTERNAL パッケージ内の次の内部ファンクションの参照が含まれています。

- ROW_VARIABLE_VALUE_FUNCTION
- DDL_VARIABLE_VALUE_FUNCTION
- EVALUATION_CONTEXT_FUNCTION

- ROW_FAST_EVALUATION_FUNCTION
- DDL_FAST_EVALUATION_FUNCTION

ROW_VARIABLE_VALUE_FUNCTION では、SYS.LCR$_ROW_RECORD インスタンスをカプセル化する SYS.AnyData ペイロードが SYS.LCR$_ROW_RECORD インスタンスに変換されてから、データに対するルールが評価されます。

DDL_VARIABLE_VALUE_FUNCTION では、SYS.LCR$_DDL_RECORD インスタンスをカプセル化する SYS.AnyData ペイロードが SYS.LCR$_DDL_RECORD インスタンスに変換されてから、データに対するルールが評価されます。

EVALUATION_CONTEXT_FUNCTION は、CREATE_EVALUATION_CONTEXT プロシージャのコールで evaluation_function として指定されます。このファンクションによって、取得イベントの通常のルール評価が補足されます。取得プロセスは行 LCR と DDL LCR をキューにエンキューし、このファンクションによって、コミット、ロールバックおよびデータ・ディクショナリ変更などの他の内部イベントのエンキューが可能になります。この情報は、伝播や適用プロセスのルール評価中にも使用されます。

ROW_FAST_EVALUATION_FUNCTION では、ルール評価中に次の LCR$_ROW_RECORD メンバー関数へのアクセスを最適化することによってパフォーマンスが改善されます。

- GET_OBJECT_OWNER
- GET_OBJECT_NAME
- IS_NULL_TAG
- GET_SOURCE_DATABASE_NAME
- GET_COMMAND_TYPE

DDL_FAST_EVALUATION_FUNCTION では、演算子が <、<=、=、>= または > で、他方のオペランドが定数の場合は、ルール評価中に次の LCR$_DDL_RECORD メンバー関数へのアクセスを最適化することによってパフォーマンスが改善されます。

- GET_OBJECT_OWNER
- GET_OBJECT_NAME
- IS_NULL_TAG
- GET_SOURCE_DATABASE_NAME
- GET_COMMAND_TYPE
- GET_BASE_TABLE_NAME
- GET_BASE_TABLE_OWNER

DBMS_STREAMS_ADM パッケージを使用して作成したルールでは、ADD_SUBSET_RULES プロシージャを使用して作成したサブセット・ルールの場合を除き、ROW_FAST_EVALUATION_FUNCTION または DDL_FAST_EVALUATION_FUNCTION が使用されます。

注意： 前述の内部ファンクションに関する情報は、あくまでも参考用です。これらのファンクションは直接実行しないでください。

関連項目： LCR とそのメンバー関数の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

Streams とイベント・コンテキスト

Streams において、取得プロセスではイベント・コンテキストは使用されませんが、伝播と適用プロセスでは使用されます。取得イベントとユーザー・エンキュー・イベントの両方がキュー内でステージングされます。イベントがキュー内でステージングされると、伝播または適用プロセスでは、イベントとイベント・コンテキストを評価のためにルール・エンジンへ送信可能となります。伝播と適用プロセスでは、名前 / 値ペアのイベント・コンテキストが常に送信されます。この場合、AQ$_MESSAGE が名前で、Streams イベント自体が値です。

カスタム評価コンテキストを作成すると、暗黙的変数を使用して、Streams イベントを参照する伝播と適用ルールを作成できます。その場合、各暗黙的変数に対する変数値の評価ファンクションでは、AQ$_MESSAGE という名前のイベント・コンテキストをチェックできます。この名前と一致するイベント・コンテキストが検出されると、変数値の評価ファンクションからイベント自体に基づいて値が戻されます。イベント・コンテキストは、評価ファンクションと変数メソッド・ファンクションにも渡すことができます。

関連項目：

- イベント・コンテキストの詳細は、5-11 ページの「ルール・セットの評価」を参照してください。
- 変数値の評価ファンクションの詳細は、5-6 ページの「明示的変数と暗黙的変数」を参照してください。
- 5-8 ページ「評価ファンクション」

Streams およびアクション・コンテキスト

Streams では、アクション・コンテキストには 2 つの用途があります。一方はサブセット・ルール内部の LCR の変換で、他方はユーザー定義のルールベースの変換です。ルールのアクション・コンテキストにサブセットの変換とユーザー定義のルールベースの変換の両方が含まれている場合は、サブセットの変換が実行されてから、ユーザー定義のルールベースの変換が実行されます。

サブセット・ルールまたはルールベースの変換を指定して、ルール・セットに含まれる 1 つ以上のルールに非 NULL のアクション・コンテキストを使用する場合は、特定の条件について TRUE と評価できるルールが 1 つのみであることを確認してください。特定の条件について複数のルールが TRUE と評価される場合も、戻されるルールは 1 つのみであり、これは予測できない結果を生じる可能性があります。

たとえば、2 つのルールがあり、どちらも hr.employees 表に対する DML 変更が LCR に含まれている場合に TRUE と評価されるとします。最初のルールには NULL アクション・コンテキストがあります。第 2 のルールには、変換を指定するアクション・コンテキストがあります。hr.employees 表に対する DML 変更があると、どちらのルールも変更について TRUE と評価されますが、戻されるルールは 1 つのみです。この場合、どちらのルールが戻されるかに応じて、変換が発生する場合と発生しない場合があります。

いずれかのルールに非 NULL のアクション・コンテキストがあるかどうかに関係なく、TRUE と評価できるルールがどの条件についてもルール・セット内で 1 つのみになるようにする必要があります。このガイドラインに従うと、非 NULL のアクション・コンテキストが将来ルールに追加される場合なども、予測できない結果を回避できます。

関連項目：

- サブセット・ルールの詳細は、4-12 ページの「Streams ルールを使用した行のサブセット化」および 6-6 ページの「表ルールとサブセット・ルール」を参照してください。
- 6-23 ページ「ルールベースの変換」

ユーザー作成ルール、ルール・セットおよび評価コンテキスト

DBMS_STREAMS_ADM パッケージで作成されるよりも複雑なルールが必要な場合は、DBMS_RULE_ADM パッケージを使用して作成できます。次のような場合に、DBMS_RULE_ADM パッケージの使用が必要になることがあります。

- NOT 条件演算子を使用するルール条件や、特定の操作にのみ関連するルール条件など、複合ルール条件を指定する必要がある場合。
- Streams 環境でルールのカスタム評価コンテキストを作成する必要がある場合。

DBMS_RULE_ADM パッケージを使用してルール・セットを作成し、それを取得プロセス、伝播または適用プロセスに関連付けることができます。

関連項目：

- 12-5 ページ「取得プロセスのルール・セットの指定」
- 13-13 ページ「伝播のルール・セットの指定」
- 14-8 ページ「適用プロセスのルール・セットの指定」

複合ルール条件

Streams 環境では、複合ルール条件とは DBMS_STREAMS_ADM パッケージを使用して作成できないルール条件です。DBMS_STREAMS_ADM パッケージで作成できるシステム作成ルール条件のタイプについては、6-4 ページの表 6-2 を参照してください。より複雑な条件を伴うルールを作成する必要がある場合は、DBMS_RULE_ADM パッケージを使用します。

様々な複合条件があります。ここでは、複合ルール条件の例をいくつか示します。

注意：

- ルール条件の場合、表やユーザーなどのデータベース・オブジェクトの名前は、大 / 小文字の区別を含めてデータベース内の名前と正確に一致させる必要があります。また、名前を二重引用符で囲むことはできません。
- ルール条件でデータベース名を指定する場合は、ドメイン名を含む完全データベース名を指定する必要があります。

NOT 条件演算子を使用してオブジェクトを除外するルール条件

NOT 条件演算子を使用すると、Streams 環境で特定の変更を取得、伝播または適用対象から除外できます。

たとえば、hr.regions 表の変更を除き、hr スキーマ内のすべてのデータベース・オブジェクトに対するすべての DML および DDL 変更について、TRUE と評価されるルール条件を指定する必要があるとします。NOT 条件演算子を使用すると、DML 変更用に 1 つと DDL 変更用に 1 つ、合計 2 つのルールを使用して、このルール条件を指定できます。これらのルールのルール条件を次に示します。

```
(:dml.get_object_owner() = 'HR' AND NOT :dml.get_object_name() = 'REGIONS')
AND :dml.is_null_tag() = 'Y'

(:ddl.get_object_owner() = 'HR' AND NOT :ddl.get_object_name() = 'REGIONS')
AND :ddl.is_null_tag() = 'Y'
```

この例では、HR や REGIONS などのオブジェクト名がすべて大文字で指定されていることに注意してください。ルールが正しく評価されるには、オブジェクト名の大 / 小文字表記がデータ・ディクショナリ内の大 / 小文字表記と一致する必要があります。したがって、オブジェクトの作成時に大 / 小文字表記を指定していない場合、ルール条件ではオブジェクト名をすべて大文字で指定します。ただし、オブジェクトの作成時に二重引用符を使用して大 / 小文字表記を指定した場合は、ルール条件でも同じようにオブジェクト名を大 / 小文字表記で指定する必要があります。たとえば、HR スキーマ内の REGIONS 表を実際には "Regions" として作成した場合、この表に関係するルール条件では Regions を次の例のように指定します。

```
:dml.get_object_name() = 'Regions'
```

DBMS_RULE_ADM パッケージを使用してこれらのルールを作成すると、Streams の評価コンテキストを使用できます。次の例では、複合ルールを保持するルール・セットを作成し、前述の条件を伴うルールを作成して、各ルールをルール・セットに追加します。

```
BEGIN
  -- Create the rule set
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name       => 'strmadmin.complex_rules',
    evaluation_context  => 'SYS.STREAMS$_EVALUATION_CONTEXT');
  -- Create the complex rules
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name  => 'strmadmin.hr_not_regions_dml',
    condition  => ' (:dml.get_object_owner() = ''HR'' AND NOT ' ||
                  ' :dml.get_object_name() = ''REGIONS'') AND ' ||
                  ' :dml.is_null_tag() = ''Y'' ');
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name  => 'strmadmin.hr_not_regions_ddl',
    condition  => ' (:ddl.get_object_owner() = ''HR'' AND NOT ' ||
                  ' :ddl.get_object_name() = ''REGIONS'') AND ' ||
```

```
                ' :ddl.is_null_tag() = ''Y'' ');
  --  Add the rules to the rule set
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.hr_not_regions_dml',
    rule_set_name  => 'strmadmin.complex_rules');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.hr_not_regions_ddl',
    rule_set_name  => 'strmadmin.complex_rules');
END;
/
```

この場合、ルールはルール・セットから Streams の評価コンテキストを継承します。

特定タイプの操作に関するルール条件

特定タイプの操作のみを含む変更の取得、伝播または適用が必要になることがあります。たとえば、特定の表に対する挿入操作のみを含み、更新や削除などの他の操作を含まない変更を適用する場合などです。

hr.employees 表に対する INSERT 操作についてのみ TRUE と評価されるルール条件を指定する場合を考えます。そのためには、ルール条件で INSERT コマンド・タイプを指定します。

```
:dml.get_command_type() = 'INSERT' AND :dml.get_object_owner() = 'HR'
AND :dml.get_object_name() = 'EMPLOYEES' AND :dml.is_null_tag() = 'Y'
```

同様に、DELETE 操作を除き、hr.departments 表に対するすべての DML 操作について TRUE と評価されるルール条件を指定する場合を考えます。

```
:dml.get_command_type() != 'DELETE' AND :dml.get_object_owner() = 'HR'
AND :dml.get_object_name() = 'DEPARTMENTS' AND :dml.is_null_tag() = 'Y'
```

Streams クライアントについて NULL と評価される、ルール条件内の未定義変数

評価中に、変数に対する変数値の評価ファンクションから NULL が返される場合は、ルール条件内の暗黙的変数は未定義となります。ルール・エンジンで DBMS_RULE.EVALUATE プロシージャの実行時にクライアントから変数の値が送信されないと、ルール条件内に属性を持たない明示的変数は未定義となります。

属性を持つ変数に関しては、DBMS_RULE.EVALUATE プロシージャの実行時にクライアントからルール・エンジンへ、変数の値または属性の少なくとも 1 つが送信されないと、変数は未定義となります。たとえば、変数 x が属性 a と b を持つ場合、クライアントから x の値が送信されず、a の値も b の値も送信されない場合には、変数は未定義となります。ただし、クライアントから少なくとも 1 つの属性の値が送信される場合、変数は定義されます。この場合では、クライアントから a の値が送信され、b の値が送信されない場合、変数は定義されます。

ルール条件内の未定義変数は、取得プロセス、伝播と適用プロセスを含む、ルール・エンジンの Streams クライアントに対して NULL と評価されます。これに対して、ルール・エンジンの Streams 以外のクライアントでは、ルール条件内の未定義変数が原因で、ルール・エンジンから maybe_rules がクライアントに戻される場合があります。ルール・セットの評価時に、詳細を示すと TRUE と評価される可能性があるルールが maybe_rules です。

Streams クライアントに戻される maybe_rules の数は、未定義変数をそれぞれ NULL として扱うと減少します。maybe_rules の数が減少すると、イベント発生時のルール・セットの評価が効率的に実行される場合にパフォーマンスを改善できます。Streams 以外のクライアントに maybe_rules を戻すルールは、次の例で示されるように、Streams クライアントでは TRUE または FALSE ルールとなる可能性があります。

未定義変数が Streams クライアントに対する TRUE ルールとなる例 次のユーザー定義のルール条件を考えます。

```
:m IS NULL
```

評価中に変数 m の値が定義されない場合、ルール・エンジンの Streams 以外のクライアントに maybe ルールが発生します。ただし、Streams クライアントについては、未定義変数 m が NULL として扱われるため、この条件は TRUE と評価されます。このようなルールが Streams クライアントのルール・セットに追加されされないように注意する必要があります。このようなルールは、すべてのイベントに対して TRUE と評価されるためです。そのため、取得プロセスのルール・セットにそのようなルールが含まれていると、取得の対象外とされているイベントが取得プロセスによって取得される場合があります。

次に、Streams の :dml 変数を使用している別のユーザー指定のルール条件を示します。

```
:dml.get_object_owner() = 'HR' AND :m IS NULL
```

Streams クライアントでは、イベントが hr スキーマ内の表に対する行の変更で構成されており、かつ、変数 m の値が評価時に不明の場合、未定義変数 m が NULL として扱われるため、この条件は TRUE と評価されます。

未定義変数が Streams クライアントに FALSE ルールを発生させる例 次のユーザー定義のルール条件を考えます。

```
:m = 5
```

評価中に変数 m の値が定義されない場合、ルール・エンジンの Streams 以外のクライアントに maybe ルールが発生します。ただし、Streams については、未定義変数 m が NULL として扱われるため、この条件は FALSE と評価されます。

Streams の :dml 変数を使用する別のユーザー指定のルール条件を考えます。

```
:dml.get_object_owner() = 'HR' AND :m = 5
```

Streams クライアントでは、イベントが hr スキーマに存在する表への行の変更で構成されており、かつ、変数 m の値が評価時に不明である場合、未定義変数 m が NULL として扱われるため、この条件は FALSE と評価されます。

関連項目： 5-11 ページ「ルール・セットの評価」

:dml および :ddl 変数をルール条件のファンクション・パラメータとして使用することの回避

ルール条件のファンクション・パラメータとして :dml と :ddl 変数を使用しないことをお薦めします。次の例では、ファンクション my_function のパラメータとして :dml 変数が使用されています。

```
my_function(:dml) = 'Y'
```

このようなルール条件は、ルール評価のパフォーマンスを低下させ、誤った Streams データ・デクショナリ情報が取得または伝播される原因となります。

関連項目： 2-20 ページ「取得プロセス作成中のデータ・ディクショナリの複製」

カスタム評価コンテキスト

Streams 環境では、カスタム評価コンテキストを使用できます。LCR が関係するユーザー定義の評価コンテキストには、SYS.STREAMS$_EVALUATION_CONTEXT 内のすべての変数を含める必要があります。各変数の型とその変数値の評価ファンクションは、SYS.STREAMS$_EVALUATION_CONTEXT 内で変数ごとに定義されている型および評価ファンクションと同じである必要があります。また、DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT を使用して評価コンテキストを作成する場合は、evaluation_function パラメータに SYS.DBMS_STREAMS_INTERNAL.EVALUATION_CONTEXT_FUNCTION を指定してください。

評価コンテキストに関する情報は、次のデータ・ディクショナリ・ビューで検索できます。

- ALL_EVALUATION_CONTEXT_TABLES
- ALL_EVALUATION_CONTEXT_VARS
- ALL_EVALUATION_CONTEXTS

必要な場合は、これらのデータ・ディクショナリ・ビューの情報を使用し、SYS.STREAMS$_EVALUATION_CONTEXT に基づいて新規の評価コンテキストを作成できます。

注意： Oracle が提供する評価コンテキストの変数との競合を回避するために、$ や # などの特殊文字を含む変数名は使用しないでください。

関連項目： これらのデータ・ディクショナリ・ビューの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

ルールベースの変換

Streams では、**ルールベースの変換**は、ルールが TRUE に評価されるときに発生する LCR を含むイベントの変更です。たとえば、表の特定の列のデータ型が 2 つのデータベース間で異なる場合は、ルールベースの変換を使用できます。この種の列は、ソース・データベースでは NUMBER 列、接続先データベースでは VARCHAR2 列の場合があります。この場合の変換は、入力として列の NUMBER データ型の行 LCR を含む SYS.AnyData オブジェクトを取り、同じ列の VARCHAR2 データ型の行 LCR を含む SYS.AnyData オブジェクトを戻します。

変換は、入力として SYS.AnyData オブジェクトを取って SYS.AnyData オブジェクトを戻す PL/SQL ファンクションとして定義する必要があります。ルールベースの変換でサポートされるのは、1 対 1 の変換のみです。また、ファンクションから戻される LCR は、ファンクションに渡された LCR と同じである必要があります。ルールベースの変換を伴う LCR は変更できますが、新規 LCR を構成してそれを戻すことは許されません。

LCR のその他の変換例は次のとおりです。

- データベース・オブジェクトの所有者名の変更
- データベース・オブジェクト名の変更
- 列名の変更または列の削除
- 複数列への列の分割
- 複数列から 1 列への結合
- 列の内容の変更

Streams では、ルール・アクション・コンテキストを使用してルールベースの変換を指定します。ルール・アクション・コンテキストはルールに関連付けられたオプションの情報であり、イベントのルールが TRUE に評価された後でルール・エンジンのクライアントによって解析されます。ルール・エンジンのクライアントは、ユーザー作成アプリケーション、または Streams などの Oracle の内部機能です。アクション・コンテキスト内の情報は、名前 / 値ペアのリストで構成される SYS.RE$NV_LIST 型のオブジェクトです。

Streams では、ルールベースの変換は常に、アクション・コンテキスト内の次の名前 / 値ペアで構成されます。

- 名前は STREAMS$_TRANSFORM_FUNCTION です。
- 値は、VARCHAR2 として指定された PL/SQL ファンクションの名前を含む SYS.AnyData インスタンスです。このファンクションによって変換が実行されます。

変換ファンクションをコールするユーザーは、そのファンクションの EXECUTE 権限を持っている必要があります。次のリストに、変換ファンクションをコールするユーザーを示します。

- 取得プロセスまたは伝播で使用されるルール用の変換が指定されている場合、変換ファンクションをコールするユーザーはその取得プロセスまたは伝播を作成したユーザーです。
- 適用プロセスで使用されるルール用の変換が指定されている場合、変換ファンクションをコールするユーザーは、その適用プロセスの適用ユーザーです。

Streams 環境で LCR を含むイベントのルールが TRUE と評価され、STREAMS$_TRANSFORM_FUNCTION 名を使用して名前 / 値ペアを含むアクション・コンテキストが戻されると、入力パラメータにイベントを使用して PL/SQL ファンクションが実行されます。アクション・コンテキスト内で STREAMS$_ で始まるその他の名前は Oracle によって内部的に使用され、直接追加、変更または削除することはできません。Streams では、STREAMS$_ 以外で始まる名前 / 値ペアは無視されます。

Streams 環境でイベントのルールが FALSE と評価されると、そのルールはクライアントに戻されず、アクション・コンテキスト内の名前 / 値ペアに表示されている PL/SQL ファンクションは実行されません。様々なルールに同じ変換または異なる変換を使用できます。たとえば、変更の取得、伝播または適用対象となる様々な操作タイプ、表またはスキーマに、異なる変換を関連付けることができます。次の項では、取得プロセス、伝播および適用プロセスでのルールベース変換の動作を説明します。

注意：

- ルールベースの変換は、DBMS_TRANSFORM パッケージを使用して実行する変換とは異なります。この項では、DBMS_TRANSFORM パッケージを使用して実行する変換については説明しません。
- 適用の並列性が 2 以上であれば DML ハンドラをパラレルで実行できるため、多数の変換または高コストの変換がある場合は、かわりに DML ハンドラ内でイベントを変更できます。
- DDL LCR に対してルールベースの変換を実行する場合は、他の変更と一致するように DDL LCR 内で DDL テキストの変更が必要になることがあります。たとえば、ルールベースの変換によって DDL LCR 内の表名が変更になる場合は、DDL テキスト内の表名も同様に変更する必要があります。
- ルールベースの変換を使用して LCR イベントを非 LCR イベントには変換できません。この制限は、取得 LCR とユーザー・エンキュー LCR に適用されます。

関連項目：

- 15-11 ページ「ルールベースの変換の管理」
- 5-9 ページ「ルール・アクション・コンテキスト」
- DBMS_TRANSFORM パッケージの使用方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。
- DBMS_TRANSFORM パッケージの参照情報は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 伝播中に変換を実行するための DBMS_TRANSFORM パッケージの使用例は、3-15 ページの「メッセージの伝播と SYS.AnyData キュー」を参照してください。
- DML ハンドラの詳細は、4-3 ページの「適用プロセスによるイベント処理」を参照してください。

ルールベースの変換と取得プロセス

取得プロセスでルール・セットを使用する場合、取得中に変換を実行するには次の両方の条件を満たす必要があります。

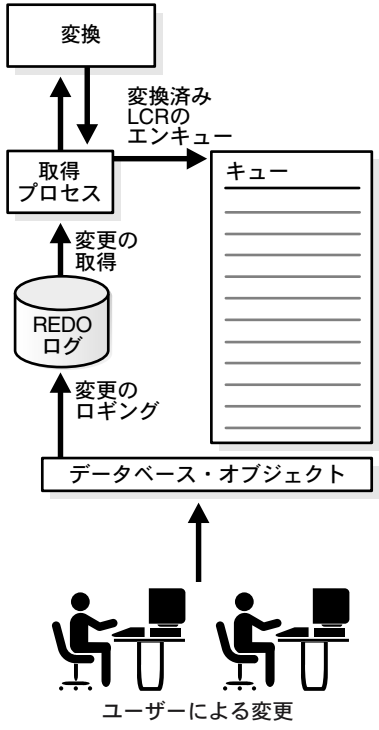
- REDO ログ内で検出された特定の変更について、ルールが TRUE と評価されること。
- ルールの評価時に、STREAMS$_TRANSFORM_FUNCTION 名とともに名前 / 値ペアを含むアクション・コンテキストが取得プロセスに戻されること。

これらの条件を満たしている場合は、取得プロセスで次の手順が実行されます。

1. REDO ログ内の変更が LCR 形式でフォーマットされます。
2. LCR が SYS.AnyData オブジェクトに変換されます。
3. 名前 / 値ペアに含まれる PL/SQL ファンクションが実行され、SYS.AnyData オブジェクトが変換されます。
4. 変換済みの SYS.AnyData オブジェクトが、取得プロセスに関連付けられているキューにエンキューされます。

図 6-2 に、取得中の変換を示します。

図 6-2 取得中の変換



たとえば、LCR イベントが取得中に変換されると、変換済み LCR イベントはソース・キューにエンキューされます。したがって、この種の取得 LCR イベントが dbs1.net データベースから dbs2.net および dbs3.net データベースへと伝播すると、伝播後には dbs2.net および dbs3.net にあるキューに変換済み LCR イベントが含まれることになります。

取得中に変換を実行するメリットは、次のとおりです。

- 変換によってプライベート情報が削除または変更されても、このプライベート情報はソース・キューに表示されず、宛先キューにも伝播しないため、セキュリティを向上できます。
- 実行される変換のタイプによっては、領域消費が減少する場合があります。たとえば、データ量を減少させる変換では、エンキュー、伝播および適用されるデータが減少します。
- 変換は複数の宛先で実行されるのではなくソース側で一度だけ実行されるため、変換済み LCR イベントに複数の宛先がある場合に変更によるオーバーヘッドが減少します。

取得中に変換を実行する場合に考えられるデメリットは、次のとおりです。

- すべてのサイトで変換済み LCR イベントが受信されます。
- ソース・データベースに変換によるオーバーヘッドが発生します。

取得中のルールベースの変換のエラー

取得中に変換ファンクションの実行エラーが発生すると、変更は取得されず、取得プロセスにエラーが戻され、取得プロセスは無効化されます。エラーを回避するために、取得プロセスを有効化する前に、ルールベースの変換を変更または削除する必要があります。

ルールベースの変換と伝播

伝播でルール・セットを使用する場合、伝播中に変換を実行するには次の両方の条件を満たす必要があります。

- 伝播のソース・キュー内の LCR イベントについて、ルールが TRUE と評価されること。この LCR イベントは、取得イベントの場合とユーザー・エンキュー・イベントの場合があります。
- ルールの評価時に、STREAMS$_TRANSFORM_FUNCTION 名とともに名前 / 値ペアを含むアクション・コンテキストが伝播に戻されること。

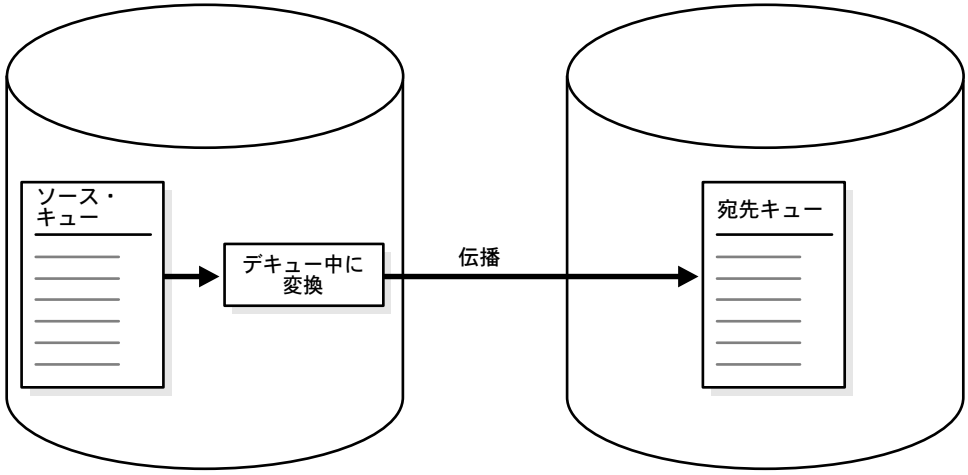
> **関連項目：** 3-3 ページ「取得イベントとユーザー・エンキュー・イベント」

これらの条件を満たしている場合は、伝播で次の手順が実行されます。

1. ソース・キューからの LCR イベントのデキューが開始されます。
2. 名前 / 値ペアに含まれる PL/SQL ファンクションが実行され、LCR イベントが変換されます。
3. 変換済み LCR イベントのデキューが完了します。
4. 変換済み LCR イベントが宛先キューに伝播します。

図 6-3 に、伝播中の変換を示します。

図 6-3 伝播中の変換



たとえば、dbs1.net データベースから dbs2.net データベースへの伝播にルールベースの変換を使用し、dbs1.net データベースから dbs3.net データベースへの伝播にはルールベースの変換を使用しない場合を考えます。

この場合、dbs1.net のキューにある LCR イベントは dbs2.net に伝播する前に変換できますが、それと同じ LCR イベントを dbs3.net に伝播するときにはオリジナルのままで伝播できます。この場合、伝播後は、dbs2.net のキューには変換済み LCR イベントが含まれ、dbs3.net のキューにはオリジナルの LCR イベントが含まれます。

伝播中に変換を実行するメリットは、次のとおりです。

- LCR イベントが伝播される前に変換によってプライベート情報が削除または変更されると、セキュリティを向上できます。
- 一部の宛先キューは変換済み LCR イベントを受信でき、他の宛先キューはオリジナルの LCR イベントを受信できます。
- 様々な宛先が同じ LCR イベントの異なるバリエーションを受信できます。

伝播中に変換を実行する場合に考えられるデメリットは、次のとおりです。

- LCR イベントが変換されると、最初の伝播後に伝播先となるデータベースは変換済み LCR イベントを受信します。たとえば、LCR イベントが dbs2.net から dbs4.net に伝播する場合、dbs4.net は変換済み LCR イベントを受信します。
- 有向ネットワークでは、最初の伝播によって変換が実行されると、ソース・データベース上で変換によるオーバーヘッドが発生します。
- 複数の接続先データベースが同じ変換を必要とする場合は、同じ変換が 2 回以上実行される可能性があります。

伝播中のルールベースの変換のエラー

伝播中に変換ファンクションの実行エラーが発生すると、エラーの原因となった LCR はデキューされず、伝播されず、伝播にエラーが戻されます。エラーを回避するために、LCR を伝播させる前に、ルールベースの変換を変更または削除する必要があります。

ルールベースの変換と適用プロセス

適用プロセスでルール・セットを使用する場合、適用中に変換を実行するには次の両方の条件を満たす必要があります。

- 適用プロセスに関連付けられたキュー内の LCR イベントについて、ルールが TRUE と評価されること。この LCR イベントは、取得イベントの場合とユーザー・エンキュー・イベントの場合があります。
- ルールの評価時に、STREAMS$_TRANSFORM_FUNCTION 名とともに名前 / 値ペアを含むアクション・コンテキストが適用プロセスに戻されること。

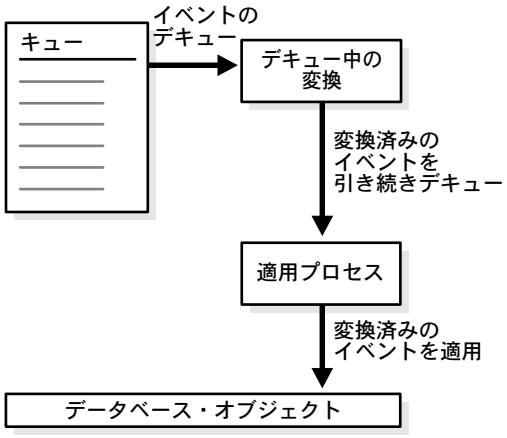
関連項目： 3-3 ページ「取得イベントとユーザー・エンキュー・イベント」

これらの条件を満たしている場合は、適用プロセスで次の手順が実行されます。

1. キューからの LCR イベントのデキューが開始されます。
2. 名前 / 値ペアに含まれる PL/SQL ファンクションが実行され、デキュー中に LCR イベントが変換されます。
3. 変換済み LCR イベントのデキューが完了します。
4. 変換済み LCR イベントが適用されます。

図 6-4 に、適用中の変換を示します。

図 6-4 適用中の変換



たとえば、dbs1.net データベースから dbs2.net データベースに LCR イベントがオリジナルの形式で伝播する場合を考えます。適用プロセスによって dbs2.net のキューからデキューされるときに、LCR イベントが変換されます。

適用中に変換を実行する場合に考えられるメリットは、次のとおりです。

- 最初の伝播後に LCR イベントの伝播先となるデータベースは、LCR イベントをオリジナルの形式で受信できます。たとえば、LCR イベントが dbs2.net から dbs4.net に伝播する場合、dbs4.net はオリジナルの LCR イベントを受信できます。
- ソース・データベースと接続先データベースが異なる場合、ソース・データベース上では変換によるオーバーヘッドは発生しません。

適用中に変換を実行する場合に考えられるデメリットは、次のとおりです。

- LCR イベントの伝播先となるデータベースすべてがオリジナルの LCR イベントを受信するため、LCR イベントにプライベート情報が含まれている場合はセキュリティが問題になる可能性があります。
- 複数の接続先データベースが同じ変換を必要とする場合は、同じ変換が 2 回以上実行される可能性があります。

適用プロセスによるデキュー中のルールベースの変換のエラー

適用中に変換ファンクションの実行エラーが発生すると、エラーの原因となった LCR はデキューされず、その LCR を含むトランザクションは適用されず、適用プロセスにエラーが戻されます。また、適用プロセスは無効化されます。エラーを回避するために、適用プロセスを有効化する前に、ルールベースの変換を変更または削除する必要があります。

変換済み LCR の適用エラー

トランザクションに適用エラーが発生し、そのトランザクションで一部の LCR がルールベースの変換によって変換済みの場合、変換済み LCR はトランザクション内の他のすべての LCR とともに例外キューに移動されます。DBMS_APPLY_ADM パッケージの EXECUTE_ERROR プロシージャを使用して、変換済み LCR を含む例外キュー内のトランザクションを再実行すると、適用プロセスのルール・セットは再評価されないため、LCR の変換は再実行されません。

ルールベースの複数の変換

取得、伝播、適用またはその任意の組合せの実行中に、LCR を変換できます。たとえば、機密データをすべての受信者に対して非表示にする必要がある場合は、取得中に LCR を変換できます。一部の受信者には追加のカスタム変換が必要な場合は、以前に変換済みの LCR を伝播または適用中に変換できます。

7

Streams 競合解消

一部の Streams 環境では、複数のデータベース間にデータ共有が原因で発生する可能性のあるデータの競合を解消するために、競合ハンドラを使用する必要があります。

この章の内容は次のとおりです。

関連項目： 14-28 ページ「Streams の競合解消の管理」

Streams 環境内の DML の競合

複数データベース上で同じデータに対して同時のデータ操作言語（DML）操作を許可している Streams 環境では、競合が発生する可能性があります。Streams 環境で DML の競合が発生する可能性があるのは、適用プロセスが DML 操作によって生じる変更を含むイベントを適用している場合のみです。この種のイベントは、行の論理変更レコードまたは行 LCR と呼ばれます。適用プロセスでは、行 LCR が原因で発生する競合が自動的に検出されます。

たとえば、異なるデータベースからの 2 つのトランザクションによって同じ行がほぼ同時に更新されると、競合が発生することがあります。Streams 環境の構成時には、競合の発生を許可するかどうかを考慮する必要があります。システム設計で競合が許可されている場合は、競合を自動的に解消するように競合解消を構成できます。

通常は、競合の可能性を回避する Streams 環境を設計する必要があります。この章で後述する競合防止技法を使用すると、ほとんどのシステム設計で共有データ全体または大部分における競合を防止できます。ただし、多くのアプリケーションでは、複数のデータベースで共有データをある程度の割合まで随時更新可能にする必要があります。この場合は、競合の可能性への対処が必要となります。

注意： 適用プロセスでは、DDL の競合やユーザー・エンキュー・イベントによって生じる競合は検出されません。この種の競合が環境で防止されることを確認してください。

関連項目： 2-3 ページ「行 LCR」

Streams 環境内の競合のタイプ

複数のデータベースでデータを共有する場合は、次のタイプの競合が発生する可能性があります。

- Streams 環境内の更新の競合
- Streams 環境内の一意性競合
- Streams 環境内の削除の競合
- Streams 環境内の外部キーの競合

Streams 環境内の更新の競合

更新の競合が発生するのは、適用プロセスで適用される行 LCR に含まれている行の更新が、同じ行の他の更新と競合する場合です。また、異なるデータベースからの 2 つのトランザクションによって、同じ行がほぼ同時に更新される場合にも、更新の競合が発生する可能性があります。

Streams 環境内の一意性競合

一意性競合が発生するのは、適用プロセスで適用される行 LCR に含まれている行の変更が、主キー制約または一意制約など、一意性の整合性制約に違反している場合です。たとえば、2 つの異なるデータベースからの 2 つのトランザクションがあり、それぞれのトランザクションが同じ主キー値を持つ表に 1 行を挿入する場合を考えます。この場合は、この 2 つのトランザクションによって一意性競合が発生します。

Streams 環境内の削除の競合

削除の競合が発生するのは、異なるデータベースから発生した 2 つのトランザクションがあり、1 つのトランザクションで 1 行を削除し、もう 1 つのトランザクションで同じ行を更新または削除する場合です。この場合、行 LCR 内で参照された行は存在せず、更新も削除もできません。

Streams 環境内の外部キーの競合

外部キーの競合が発生するのは、適用プロセスで適用される行 LCR に含まれている行の変更が、外部キー制約に違反している場合です。たとえば、hr スキーマ内で、employees 表の department_id 列は、departments 表の department_id 列の外部キーです。次の変更が 2 つの異なるデータベース（A と B）から発生し、3 つ目のデータベース（C）に伝播される場合を考えます。

- データベース A で、department_id が 271 の 1 行が departments 表に挿入されます。この変更はデータベース B に伝播され、そこで適用されます。
- データベース B で、employee_id が 206、department_id が 271 の 1 行が employees 表に挿入されます。

データベース B で発生した変更がデータベース A で発生した変更よりも先にデータベース C で適用されると、データベース C の departments 表にはまだ department_id が 271 の部門の行が存在しないため、外部キーの競合が発生します。

Streams 環境内の競合とトランザクションの順序付け

Streams 環境では、3 つ以上のデータベースがデータを共有し、そのうち 2 つ以上のデータベースでデータが更新されると、順序付けの競合が発生する可能性があります。たとえば、3 つのデータベースが hr.departments 表の情報を共有している使用例を考えます。データベースの名前は mult1.net、mult2.net および mult3.net です。mult1.net で hr.departments 表の 1 行が変更され、その変更が mult2.net と mult3.net の両方に伝播されると想定します。次の一連のアクションが発生する可能性があります。

1. 変更が mult2.net に伝播します。
2. mult2.net の適用プロセスが mult1.net からの変更を適用します。
3. mult2.net で、同じ行に対して別の変更が行われます。
4. mult2.net での変更が mult3.net に伝播します。
5. mult3.net で、1 つの適用プロセスは、別の適用プロセスが mult3.net で mult1.net から変更の適用を実施する前に、mult2.net から変更の適用を試みます。

この場合、mult3.net での行の列値は mult2.net から伝播した行 LCR 内の対応する元の値と一致しないため、競合が発生します。

トランザクションが順不同で適用されると、データ競合の原因となるのみでなく、データのサポートがリモート・データベースに正常に伝播されていないと、そのデータベースで参照整合性の問題が発生する場合があります。新規の顧客が受注部門に連絡する場合の使用例を考えます。顧客レコードが作成され、受注が処理されます。リモート・データベースで受注データが顧客データより前に適用されると、受注で参照されている顧客はリモート・データベースに存在しないため、参照整合性エラーが発生します。

順序付けの競合が発生した場合は、必要なデータがリモート・データベースに伝播して適用されてから、例外キューにあるトランザクションを再実行すれば、競合を解消できます。

Streams 環境内の競合検出

適用プロセスでは、更新、一意性、削除および外部キーの競合が次のように検出されます。

- 行 LCR 内の行の元の値と、接続先データベースにある同じ行の現行の値が一致しない場合は、適用プロセスによって更新の競合が検出されます。
- 挿入または更新操作を含む LCR の適用時に一意制約違反が発生すると、適用プロセスによって一意性競合が検出されます。
- 更新または削除操作を含む LCR の適用時に、行の主キーが存在しないためにその行が見つからないと、適用プロセスによって削除の競合が検出されます。
- LCR の適用時に外部キー制約違反が発生すると、適用プロセスによって外部キーの競合が検出されます。

適用プロセスで LCR を直接適用しようとするか、DML ハンドラなどの適用プロセス・ハンドラで LCR に対して `EXECUTE` メンバー・プロシージャを実行すると、競合が検出される場合があります。また、`DBMS_APPLY_ADM` パッケージの `EXECUTE_ERROR` または `EXECUTE_ALL_ERRORS` プロシージャの実行時にも、競合が検出される場合があります。

注意： 更新 LCR、削除 LCR および LOB 列に対するピース単位更新を取り扱う LCR 内の古い LOB 値は、競合検出には使用されません。

Streams 環境内の競合検出中の行の識別

競合を正確に検出するためには、Oracle は異なるデータベースで対応する行を一意に識別して合致させることができる必要があります。デフォルトでは、Oracle は表の主キーを使用してその表の各行を一意に識別します。表に主キーが存在しない場合は、代替キーを指定する必要があります。代替キーとは、Oracle が表の各行を一意に識別するために使用できる列または列セットです。

関連項目： 4-11 ページ「代替キー列」

Streams 環境内の競合防止

この項では、データ競合を防止する方法について説明します。

プライマリ・データベース所有権モデルの使用

システム内で、共有データを含む表への同時更新アクセスを行うデータベースの数を制限することによって、競合の可能性を回避できます。プライマリ所有権を使用すると、共有データ・セットへの更新が 1 つのデータベースにしか許可されないため、すべての競合が防止されます。アプリケーションでは、行と列のサブセットを使用して、データの所有権を表レベルよりも細かく設定できます。たとえば、アプリケーションでは、共有表の特定の列や行への更新アクセス権を、データベースごとに設定できます。

特定タイプの競合の防止

プライマリ・データベース所有権モデルではアプリケーション要件にとって限定的すぎる場合は、共有所有権データ・モデルを使用できます。このデータ・モデルは、競合が発生する可能性があることを意味します。それでも、通常は単純な方針をいくつか使用して特定タイプの競合を防止できます。

Streams 環境内の一意性競合の防止

各データベースで共有データに一意識別子を使用させることで、一意性競合を回避できます。Streams 環境ですべてのデータベースに一意識別子を確実に使用させるには、3 つの方法があります。

その 1 つは、次の SELECT 文を実行して一意識別子を構成することです。

```
SELECT SYS_GUID() OID FROM DUAL;
```

この SQL 演算子は、16 バイトのグローバル一意識別子を戻します。この値は、グローバル一意識別子を生成するために時刻、日付およびコンピュータ識別子を使用するアルゴリズムに基づいています。グローバル一意識別子は、次のような書式で表示されます。

```
A741C791252B3EA0E034080020AE3E0A
```

また、データを共有する各データベースで順序を作成し、データベース名（または他のグローバル一意値）をローカルの順序と連結して一意性競合を防止する方法もあります。このアプローチを使用すると、重複する順序値を回避し、一意性競合を防止できます。

最後に、2 つのデータベースで同じ値を生成できないように、データを共有する各データベースでカスタマイズされた順序を作成する方法があります。そのためには、CREATE SEQUENCE 文で開始値、増分値および最大値の組合せを使用します。たとえば、次の順序を構成できます。

パラメータ	データベース A	データベース B	データベース C
START WITH	1	3	5
INCREMENT BY	10	10	10
範囲の例	1, 11, 21, 31, 41,...	3, 13, 23, 33, 43,...	5, 15, 25, 35, 45,...

同様のアプローチを使用すると、データベースごとに一意の範囲を生成する START WITH および MAXVALUE を指定して、各データベースに異なる範囲を定義できます。

Streams 環境内の削除の競合の防止

共有データ環境では、削除の競合を必ず回避してください。通常、共有所有権データ・モデル内で動作するアプリケーションでは、DELETE 文を使用して行を削除することはありません。かわりに、アプリケーションで行に削除マークを付けておき、論理的に削除された行を定期的にパージするようにシステムを構成する必要があります。

Streams 環境内の更新の競合の防止

一意性競合と削除の競合の可能性を排除した後、更新の競合の発生可能数も制限する必要があります。ただし、共有所有権データ・モデルでは、更新の競合を全面的に防止することはできません。更新の競合を全面的に防止できない場合は、可能な競合のタイプを把握した上で、それが発生した場合は解消するようにシステムを構成する必要があります。

Streams 環境内の競合解消

更新の競合が検出された場合、競合ハンドラではその解消を試みることができます。Streams には、更新の競合を解消するためのビルトイン競合ハンドラが用意されていますが、一意性、削除、外部キーまたは順序付けの競合については用意されていません。ただし、ビジネス・ルールに固有のデータ競合を解消するために、独自のカスタム競合ハンドラを作成できます。この種の競合ハンドラは、DML ハンドラまたはエラー・ハンドラに付属させることができます。

ビルトイン競合ハンドラとカスタム競合ハンドラのどちらを使用する場合も、競合が検出されるとただちに使用されます。指定した競合ハンドラでも関連する適用ハンドラでも競合を解消できない場合、その競合は例外キューに記録されます。競合が発生した場合は、関連する適用ハンドラを使用してデータベース管理者に通知する必要があります。

競合のためにトランザクションが例外キューに移動される場合は、競合の原因となった条件を訂正できる場合があります。このような場合は、DBMS_APPLY_ADM パッケージの EXECUTE_ERROR プロシージャを使用してトランザクションを再実行できます。

関連項目：

- DML ハンドラおよびエラー・ハンドラの詳細は、4-4 ページの「イベント処理オプション」を参照してください。
- 更新の競合ハンドラと DML ハンドラおよびエラー・ハンドラの相互作用の詳細は、4-16 ページの「ハンドラと行 LCR の処理」を参照してください。
- 4-34 ページ「例外キュー」
- DBMS_APPLY_ADM パッケージの EXECUTE_ERROR プロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ビルトインの更新の競合ハンドラ

この項では、使用可能なビルトインの更新の競合ハンドラのタイプと、この種のハンドラでの列リストと解消列の使用方法について説明します。列リストは、更新の競合がある場合に更新の競合ハンドラがコールされる列のリストです。解消列は、更新の競合ハンドラの識別に使用される列です。ビルトインの更新の競合ハンドラ MAXIMUM または MINIMUM を使用する場合は、解消列も競合の解消に使用されます。解消列は、ハンドラの列リスト内の列の 1 つである必要があります。

特定の表について 1 つ以上の更新の競合ハンドラを指定するには、DBMS_APPLY_ADM パッケージの SET_UPDATE_CONFLICT_HANDLER プロシージャを使用します。一意性競合や削除または外部キーの競合には、ビルトインの競合ハンドラはありません。

関連項目：

- 更新の競合ハンドラの追加、変更および削除の各手順については、14-28 ページ「Streams の競合解消の管理」を参照してください。
- SET_UPDATE_CONFLICT_HANDLER プロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 7-11 ページ「列リスト」
- 7-13 ページ「解消列」

ビルトインの更新の競合ハンドラのタイプ

Oracle には、Streams 環境用に OVERWRITE、DISCARD、MAXIMUM および MINIMUM タイプのビルトインの更新の競合ハンドラが用意されています。

後述の各タイプのハンドラの説明では、次の競合例を参照しています。

1. dbs1.net ソース・データベースで次の更新が行われます。

   ```
   UPDATE hr.employees SET salary = 4900 WHERE employee_id = 200;
   COMMIT;
   ```

   この更新によって、従業員 200 の給与が 4400 から 4900 に変更されます。

2. ほぼ同時に、dbs2.net 接続先データベースで次の更新が行われます。

   ```
   UPDATE hr.employees SET salary = 5000 WHERE employee_id = 200;
   COMMIT;
   ```

3. 取得プロセスは、dbs1.net ソース・データベースで更新を取得し、結果の行 LCR をキューに入れます。
4. 伝播は dbs1.net のキューから dbs2.net のキューに行 LCR を伝播させます。
5. dbs2.net の適用プロセスは、行 LCR を hr.employees 表に適用しようとしますが、dbs2.net の給与値は 5000 であり、これは行 LCR 内の給与の元の値（4400）と一致しないため、競合が発生します。

ここでは、各ビルトイン競合ハンドラと、それぞれがこの競合をどのように解消するかについて説明します。

OVERWRITE 競合が発生すると、OVERWRITE ハンドラは、接続先データベースの現行の値をソース・データベースからの LCR 内の新規の値で置換します。

OVERWRITE ハンドラを競合例の dbs2.net 接続先データベースの hr.employees 表に使用すると、行 LCR 内の新規の値によって dbs2.net の値が上書きされます。したがって、競合解消後は、従業員 200 の給与は 4900 となります。

DISCARD 競合が発生すると、DISCARD ハンドラはソース・データベースからの LCR 内の値を無視し、接続先データベースの値を保持します。

DISCARD ハンドラを競合例の dbs2.net 接続先データベースの hr.employees 表に使用すると、行 LCR 内の新規の値が廃棄されます。したがって、競合解消後は、dbs2.net での従業員 200 の給与は 5000 となります。

MAXIMUM 競合が発生すると、MAXIMUM 競合ハンドラは、指定された解消列について、ソース・データベースからの LCR にある新規の値を、接続先データベース内の現行の値と比較します。LCR にある解消列の新規の値が接続先データベースの列の現行値より大きければ、適用プロセスでは LCR を使用して競合が解消されます。LCR にある解消列の新規の値が接続先データベースの列の現行値より小さければ、適用プロセスでは接続先データベースを使用して競合が解消されます。

MAXIMUM ハンドラを競合例の dbs2.net 接続先データベースにある hr.employees 表の salary 列に使用すると、行 LCR にある給与は表にある現行の給与より小さいため、適用プロセスでは行 LCR は適用されません。したがって、競合解消後は、dbs2.net での従業員 200 の給与は 5000 となります。

関係するトランザクションの時間ベースで競合を解消する場合は、トリガーを使用してトランザクションの時刻が自動的に記録されるように、共有表に列を追加する方法があります。この場合は、この列を MAXIMUM 競合ハンドラ用の解消列として指定でき、最新（または最大）時刻を伴うトランザクションが自動的に使用されます。

ここでは、hr.employees 表に対するトランザクションの時刻を記録するトリガーの例を示します。job_id、salary および commission_pct 列は、競合解消ハンドラの列リストに含まれているとします。トリガーは、列リスト内の列に対して UPDATE が実行されるか、INSERT が実行された場合にのみ起動します。

```
CONNECT hr/hr

ALTER TABLE hr.employees ADD (time TIMESTAMP WITH TIME ZONE);

CREATE OR REPLACE TRIGGER hr.insert_time_employees
BEFORE
  INSERT OR UPDATE OF job_id, salary, commission_pct ON hr.employees
FOR EACH ROW
BEGIN
   -- Consider time synchronization problems. The previous update to this
   -- row may have originated from a site with a clock time ahead of the
   -- local clock time.
   IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
     :NEW.TIME := SYSTIMESTAMP;
   ELSE
     :NEW.TIME := :OLD.TIME + 1 / 86400;
   END IF;
END;
/
```

この種のトリガーを競合解消に使用する場合は、トリガーの起動プロパティがデフォルトの 1 回起動に設定されていることを確認してください。設定が異なると、適用プロセスによってトランザクションが適用されるときに新規時刻がマークされ、トランザクションの実際時刻が失われる可能性があります。

関連項目： 4-23 ページ「トリガー起動プロパティ」

MINIMUM 競合が発生すると、MINIMUM 競合ハンドラは、指定された解消列について、ソース・データベースからの LCR にある新規の値を、接続先データベース内の現行の値と比較します。LCR にある解消列の新規の値が接続先データベースの列の現行値より小さければ、適用プロセスでは LCR を使用して競合が解消されます。LCR にある解消列の新規の値が接続先データベースの列の現行値より大きければ、適用プロセスでは接続先データベースを使用して競合が解消されます。

MINIMUM ハンドラを競合例の dbs2.net 接続先データベースにある hr.employees 表の salary 列に使用すると、行 LCR にある給与は表にある現行の給与より小さいため、適用プロセスでは行 LCR を使用して競合が解消されます。したがって、競合解消後は、従業員 200 の給与は 4900 となります。

列リスト

表に対してビルトインの更新の競合ハンドラを指定するたびに、**列リスト**を指定する必要があります。列リストは、更新の競合ハンドラがコールされる列のリストです。適用プロセスが行 LCR を適用するときに、リスト内の 1 つ以上の列に更新の競合が発生すると、競合を解消するために更新の競合ハンドラがコールされます。リストにない列にのみ競合が発生した場合、更新の競合ハンドラはコールされません。競合解消の有効範囲は、1 つの行 LCR の 1 つの列リストです。

特定の表に対して更新の競合ハンドラを複数指定することはできますが、同じ列を複数の列リストに含めることはできません。たとえば、hr.employees 表にビルトインの更新の競合ハンドラを 2 つ指定する場合を考えます。

- 最初の更新の競合ハンドラの列リストには、列 salary および commission_pct を指定します。
- 2 番目の更新の競合ハンドラの列リストには、列 job_id および department_id を指定します。

また、この表に他の競合ハンドラは存在しないものと想定します。この場合、適用プロセスが行 LCR を適用するときに salary 列に競合が発生すると、それを解消するために最初の更新の競合ハンドラがコールされます。ただし、department_id 列に競合が発生すると、それを解消するために 2 番目の更新の競合ハンドラがコールされます。どの競合ハンドラの列リストにもない列に競合が発生すると、競合ハンドラはコールされず、エラーになります。この例では、manager_id 列に競合が発生すると、エラーになります。行 LCR の適用時に複数の列リストに競合が発生し、列リストにない列での競合がなければ、競合が発生した列リストごとに適切な更新の競合ハンドラが起動されます。

列リストを使用すると、データ型ごとに異なるハンドラを使用して競合を解消できます。たとえば、通常、数値データには MAXIMUM または MINIMUM 競合ハンドラが適しており、文字データには OVERWRITE または DISCARD 競合ハンドラが適しています。

列リストにない列に競合が発生すると、表に対する特定の操作用のエラー・ハンドラがそれを解消しようとします。エラー・ハンドラが競合を解消できない場合や、この種のエラー・ハンドラがない場合は、競合の原因となったトランザクションが例外キューに移動されます。

また、OVERWRITE、MAXIMUM または MINIMUM ビルトイン・ハンドラのいずれかを使用する列リスト内の列に競合が発生した場合に、この列リスト内のすべての列が行 LCR に含まれていないと、使用可能でない値があるため、競合は解消できません。この場合は、競合の原因となったトランザクションが例外キューに移動されます。列リストで DISCARD ビルトイン方法が使用されている場合は、行 LCR にこの列リストのすべての列が含まれていない場合にも、行 LCR が廃棄され、エラーは発生しません。

ソース・データベースの複数の列が接続先データベースの列リストに影響する場合は、列リストに指定された列に条件付きのサプリメンタル・ログ・グループを指定する必要があります。サプリメンタル・ロギングをソース・データベースで指定し、競合を正しく解消するために必要な情報を LCR に追加します。通常、列リスト内の列に条件付きのサプリメンタル・ログ・グループを指定する必要があるのは、列リストに列が複数存在する場合であり、列リストに存在する列が 1 つのみの場合は指定する必要ありません。

ただし、列リストに存在する列が 1 つのみの場合でも、条件付きのサプリメンタル・ログ・グループが必要になることがあります。つまり、適用ハンドラまたはルールベースの変換で、ソース・データベースからの複数列を接続先データベースの列リストの単一行に結合する場合です。たとえば、ルールベースの変換で、通り、州および郵便番号を格納するソース・データベースからの 3 行を使用し、そのデータを接続先データベースで単一の住所列に結合する場合です。

また、列リストに存在する列が複数の場合でも、条件付きのサプリメンタル・ログ・グループが不要なことがあります。たとえば、適用ハンドラまたはルールベースの変換で、ソース・データベースからの単一の住所列を接続先データベースの列リストの複数行に分割する場合です。ルールベースの変換で、ソース・データベースの 1 つの住所列（通り、州および郵便番号を含む）を取得し、そのデータを接続先データベースの 3 列に分割する場合です。

注意： ビルトインの更新の競合ハンドラでは、LOB 列はサポートされません。したがって、SET_UPDATE_CONFLICT_HANDLER プロシージャを実行するときには、column_list パラメータに LOB 列を含めないでください。

関連項目： 2-10 ページ「Streams 環境内のサプリメンタル・ロギング」

解消列

解消列は、更新の競合ハンドラの識別に使用される列です。ビルトインの更新の競合ハンドラ MAXIMUM または MINIMUM を使用する場合は、**解消列**も競合の解消に使用されます。解消列は、ハンドラの列リスト内の列の 1 つである必要があります。

たとえば、hr.employees 表の salary 列を MAXIMUM または MINIMUM 競合ハンドラの解消列として指定すると、行 LCR 内の列リストの値を適用するか、接続先データベースにおける列リストの値を保持するかを判断するために salary 列が評価されます。

競合に解消列が関係する次のどちらの状況でも、エラー・ハンドラで問題を解決できなければ、適用プロセスは競合の原因となった行 LCR を含むトランザクションが例外キューに移動されます。これらの場合、競合は解消できず、接続先データベースの列の値がそのまま保持されます。

- 解消列について、新規の LCR 値と接続先データベースの行の値が同一の場合（解消列が競合の原因となった列でない場合など）。
- 解消列の新規の LCR 値、または接続先データベースの解消列の現行の値が NULL の場合。

注意： 解消列は、OVERWRITE および DISCARD 競合ハンドラには使用されませんが、これらの競合ハンドラ用に指定する必要があります。

データ収束

複数のデータベース間でデータを共有しており、そのすべてでデータを同一にする必要がある場合は、すべてのデータベースでデータを収束させる競合解消ハンドラを使用してください。すべてのデータベースで共有データの変更を許可する場合、表のデータ収束が可能になるのは、データを共有する全データベースが共有データに対する変更を取得し、データを共有する他のすべてのデータベースにその変更を伝播する場合のみです。

このような環境では、MAXIMUM 競合解消方法で収束を保証できるのは、解消列の値が常に増加する場合のみです。時間ベースの解消列は、1 行の連続するタイムスタンプが個別であるかぎり、この要件を満たします。このような環境で MINIMUM 競合解消方法によって収束を保証できるのは、解消列の値が常に減少する場合のみです。

カスタム競合ハンドラ

PL/SQL プロシージャを作成し、カスタム競合ハンドラとして使用できます。特定の表に 1 つ以上のカスタム競合ハンドラを指定するには、DBMS_APPLY_ADM パッケージの SET_DML_HANDLER プロシージャを使用します。特に、このプロシージャを実行してカスタム競合ハンドラを指定するときに、次のパラメータを設定します。

- object_name パラメータを、競合解消の対象となる表の完全修飾された名前に設定します。
- object_type パラメータを TABLE に設定します。
- operation_name パラメータを、カスタム競合ハンドラのコール対象となる操作のタイプに設定します。可能な操作は、INSERT、UPDATE、DELETE および LOB_UPDATE です。
- エラー・ハンドラでエラー発生時に競合解消を実行する場合は、error_handler パラメータを true に設定します。また、競合解消を DML ハンドラに組み込む場合は、error_handler パラメータを false に設定します。

  このパラメータを false に設定した場合は、LCR 用の EXECUTE メンバー・プロシージャを使用して行 LCR を実行すると、指定したオブジェクトと操作に対して DML ハンドラ内で競合解消が実行されます。
- user_procedure パラメータを設定して、競合解消用のプロシージャを指定します。このユーザー・プロシージャは、指定したタイプの操作によって指定した表に生じた競合を解消するためにコールされます。

カスタム競合ハンドラが競合を解消できない場合、適用プロセスは、競合を含むトランザクションを例外キューに移動し、そのトランザクションを適用しません。

特定のオブジェクトにビルトインの更新の競合ハンドラとカスタム競合ハンドラの両方が存在する場合は、次の両方の条件が満たされる場合にのみビルトインの更新の競合ハンドラが起動されます。

- カスタム競合ハンドラで、LCR 用の EXECUTE メンバー・プロシージャを使用して行 LCR が実行される場合。
- 行 LCR 用の EXECUTE メンバー・プロシージャの conflict_resolution パラメータが true に設定されている場合。

関連項目：

- 更新の競合ハンドラと DML ハンドラおよびエラー・ハンドラの相互作用の詳細は、4-16 ページの「ハンドラと行 LCR の処理」を参照してください。
- 14-13 ページ「DML ハンドラの管理」
- 14-20 ページ「エラー・ハンドラの管理」
- `SET_DML_HANDLER` プロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

8

Streams のタグ

この章では、Streams のタグに関連する概念について説明します。

この章の内容は次のとおりです。

- タグの概要
- DBMS_STREAMS_ADM パッケージによって作成されるタグとルール
- タグと適用プロセス
- タグを使用した変更の循環の回避

関連項目： 16-24 ページ「Streams タグの管理」

タグの概要

REDO ログの各 REDO エントリには、**タグ**が関連付けられています。タグのデータ型は RAW です。デフォルトでは、ユーザーまたはアプリケーションが REDO エントリを生成する時点では、各 REDO エントリのタグの値は NULL であり、NULL タグは REDO エントリの領域をコンシュームしません。タグ値のサイズの上限は 2000 バイトです。

タグ値の解析方法を構成できます。たとえば、タグを使用すると、LCR に含まれている変更の発生場所が、ローカル・データベースであるか他のデータベースであるかを判断できるため、変更の循環（発生場所となったデータベースへの LCR の送信）を回避できます。タグは他の LCR の追跡にも使用できます。また、タグを使用して、LCR ごとに接続先データベースのセットを指定することもできます。

REDO ログ内で生成されるタグの値を制御するには、次の方法があります。

- DBMS_STREAMS.SET_TAG プロシージャを使用して、現行のセッションで生成される REDO タグの値を指定します。セッション中にデータベースに変更があると、タグはその変更を記録する REDO エントリの一部となります。様々なセッションで同じタグ設定または異なるタグ設定を使用できます。
- DBMS_APPLY_ADM パッケージの CREATE_APPLY または ALTER_APPLY プロシージャを使用して、適用プロセスの実行時に生成される REDO タグの値を制御します。このタグ設定は、適用プロセス・コーディネータによって調整されるすべてのセッションで使用されます。デフォルトでは、適用プロセスで生成される REDO エントリのタグ値は、'00'（2 つのゼロ）と等価の 16 進数です。

これらのタグは、REDO ログから変更を取り出す取得プロセスによって取得される LCR の一部となります。取得プロセスのルール・セットに含まれるルールに基づき、変更に関する REDO エントリ内のタグ値によって、変更が取得されるかどうかを判断できます。

同様に、タグが LCR の一部になっている場合は、そのタグの値によって、伝播で LCR が伝播されるかどうかと、適用プロセスで LCR が適用されるかどうかを判別できます。変換、DML ハンドラまたはエラー・ハンドラの動作も、タグの値に依存させることが可能です。また、LCR 用の SET_TAG メンバー・プロシージャを使用すると、既存の LCR のタグ値を設定できます。たとえば、変換中に LCR 内のタグを設定できます。

> **関連項目：** LCR 用の SET_TAG メンバー・プロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

DBMS_STREAMS_ADM パッケージによって作成されるタグとルール

DBMS_STREAMS_ADM パッケージのプロシージャを使用してルールを作成すると、デフォルトで各ルールにはタグが NULL の場合にのみ TRUE と評価される条件が含まれます。DML ルールの条件は、次のとおりです。

```
:dml.is_null_tag()='Y'
```

DDL ルールの条件は、次のとおりです。

```
:ddl.is_null_tag()='Y'
```

ルール・セットに 1 つのルールが含まれており、そのルールに前述の条件が含まれている場合を考えます。この場合、Streams 取得プロセス、伝播および適用プロセスの動作は次のようになります。

- 取得プロセスが変更を取得するのは、変更に関する REDO ログ内のタグが NULL で、かつ、他のルール条件が変更について TRUE と評価される場合のみです。
- 伝播によって LCR を含むイベントが伝播されるのは、LCR 内のタグが NULL で、かつ、他のルール条件が LCR について TRUE と評価される場合のみです。
- 適用プロセスによって LCR を含むイベントが適用されるのは、LCR 内のタグが NULL で、かつ、他のルール条件が LCR について TRUE と評価される場合のみです。

具体的には、DBMS_STREAMS_ADM パッケージの次のプロシージャでは、デフォルトでこれらの条件のいずれかを含むルールが作成されます。

- ADD_GLOBAL_PROPAGATION_RULES
- ADD_GLOBAL_RULES
- ADD_SCHEMA_PROPAGATION_RULES
- ADD_SCHEMA_RULES
- ADD_SUBSET_RULES
- ADD_TABLE_PROPAGATION_RULES
- ADD_TABLE_RULES

前述の条件を含むルールを作成しない場合は、これらのプロシージャを実行するときに include_tagged_lcr パラメータを true に設定します。この設定を使用すると、タグに関する条件はルールに含まれません。したがって、LCR のルール評価はタグの値に依存しないことになります。

たとえば、dbs1.net ソース・データベースで実行された hr.locations 表に対するすべての DML 変更について、TRUE と評価される表レベルのルールを考えます。

ADD_TABLE_RULES プロシージャを実行してこのルールを生成する場合を考えます。

```
BEGIN DBMS_STREAMS_ADM.ADD_TABLE_RULES(
   table_name               =>  'hr.locations',
   streams_type             =>  'capture',
   streams_name             =>  'capture',
   queue_name               =>  'streams_queue',
   include_tagged_lcr       =>  false,  -- Note parameter setting
   source_database          =>  'dbs1.net',
   include_dml              =>  true,
   include_ddl              =>  false);
END;
/
```

include_tagged_lcr パラメータがデフォルトの false に設定されていることに注意してください。ADD_TABLE_RULES プロシージャでは、次のようなルール条件を持つルールが生成されます。

```
(((:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'LOCATIONS'))
and :dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET' )
```

取得プロセスでこのルールを含むルール・セットを使用すると、REDO エントリ内の変更に関するタグの値が '0' や '1' などの非 NULL 値の場合に、ルールが FALSE と評価されます。そのため、REDO エントリに hr.locations 表の 1 行の変更が含まれていると、変更が取得されるのは REDO エントリのタグが NULL の場合のみになります。

ただし、ここでは ADD_TABLE_RULES の実行時に include_tagged_lcr パラメータが true に設定されているとします。

```
BEGIN DBMS_STREAMS_ADM.ADD_TABLE_RULES(
   table_name               =>  'hr.locations',
   streams_type             =>  'capture',
   streams_name             =>  'capture',
   queue_name               =>  'streams_queue',
   include_tagged_lcr       =>  true,   -- Note parameter setting
   source_database          =>  'dbs1.net',
   include_dml              =>  true,
   include_ddl              =>  false);
END;
/
```

この場合、ADD_TABLE_RULES プロシージャでは、次のようなルール条件を持つルールが生成されます。

```
(((:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'LOCATIONS'))
and :dml.get_source_database_name() = 'DBS1.NET' )
```

タグに関連する条件がないことに注意してください。取得プロセスでこのルールを含むルール・セットを使用すると、hr.locations 表に対する DML 変更について REDO エントリ内のタグの値が '0' や '1' などの非 NULL 値の場合に、ルールが TRUE と評価されます。また、このルールは、タグが NULL の場合にも TRUE と評価されます。そのため、REDO エントリに hr.locations 表の DML 変更が含まれている場合は、タグの値に関係なく変更が取得されます。

システム作成ルールの is_null_tag 条件を変更する場合は、DBMS_STREAMS_ADM パッケージの適切なプロシージャを使用して新規ルールを作成します。このルールは、is_null_tag 条件を除き、変更するルールと同じである必要があります。次に、DBMS_STREAMS_ADM パッケージの REMOVE_RULE プロシージャを使用して、適切なルール・セットから古いルールを削除します。

DBMS_RULE_ADM パッケージでルールを作成した場合は、このパッケージの ALTER_RULE プロシージャを使用して、is_null_tag 条件を追加、削除または変更できます。

グローバル・ルールを使用していて、データベース全体に対する DDL の変更を取得および適用する場合は、デフォルトでオンライン・バックアップ文が取得され、伝播および適用されます。通常は、データベース管理者はオンライン・バックアップ文のレプリケートを行いません。かわりに、元々それらが実行されるデータベースで実行することのみを考慮します。オンライン・バックアップ文のレプリケートの回避策として、次の方針のいずれかを使用できます。

- オンライン・バックアップ・プロシージャに DBMS_STREAMS.SET_TAG プロシージャへのコールを 1 つ以上含め、セッション・タグにオンライン・バックアップ文が取得プロセスで無視されるような値を設定します。
- 適用プロセスに DDL ハンドラを使用し、オンライン・バックアップ文の適用を回避します。

関連項目：

- DBMS_STREAMS_ADM パッケージのプロシージャで生成されるルールの例については、第 6 章「Streams でのルールの使用方法」を参照してください。
- DBMS_STREAMS_ADM パッケージと DBMS_RULE_ADM.ALTER_RULE プロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- SET_TAG プロシージャの詳細は、16-24 ページの「現行セッションで生成されるタグ値の設定」を参照してください。

タグと適用プロセス

適用プロセスでは、DML 変更または DDL 変更の適用時に、接続先データベースの REDO ログにエントリが生成されます。たとえば、適用プロセスで表の 1 行を更新する変更が適用されると、その変更は接続先データベースの REDO ログに記録されます。これらの REDO エントリ内のタグは、DBMS_APPLY_ADM パッケージの CREATE_APPLY または ALTER_APPLY プロシージャの apply_tag パラメータを設定することで制御できます。たとえば、適用プロセスでは、'0'（ゼロ）または '1' の 16 進値と等価の REDO タグを生成できます。

適用プロセスによって REDO ログ内で生成されるデフォルトのタグ値は '00'（2 つのゼロ）です。この値は、DBMS_STREAMS_ADM パッケージのプロシージャまたは DBMS_APPLY_ADM パッケージの CREATE_APPLY プロシージャを使用して適用プロセスを作成した場合の、適用プロセス用のデフォルトのタグ値です。この値は、単に非 NULL 値であることを示します。非 NULL 値であることが重要なのは、DBMS_STREAMS_ADM パッケージによって作成されたルールには、デフォルトで REDO エントリまたは LCR 内でタグが NULL の場合にのみ TRUE と評価される条件が含まれているためです。既存の適用プロセスのタグ値は、DBMS_APPLY_ADM パッケージの ALTER_APPLY プロシージャを使用して変更できます。

DML ハンドラ、DDL ハンドラまたはメッセージ・ハンドラが DBMS_STREAMS パッケージの SET_TAG プロシージャをコールすると、ハンドラによって生成される後続の REDO エントリには、適用プロセスのタグが異なる場合にも、SET_TAG コールで指定されたタグが含まれます。ハンドラが終了すると、適用プロセスで生成される後続の REDO エントリには、適用プロセス用に指定されたタグが含まれます。

関連項目：

- 適用プロセスの詳細は、第 4 章「Streams 適用プロセス」を参照してください。
- Streams のルールに含まれるデフォルトのタグ条件の詳細は、8-3 ページの「DBMS_STREAMS_ADM パッケージによって作成されるタグとルール」を参照してください。
- 16-26 ページ「適用プロセスで生成されるタグ値の設定」
- DML ハンドラ、DDL ハンドラおよびメッセージ・ハンドラの詳細は、4-4 ページの「イベント処理オプション」を参照してください。
- SET_TAG プロシージャの詳細は、16-24 ページの「現行セッションで生成されるタグ値の設定」を参照してください。
- DBMS_STREAMS_ADM パッケージと DBMS_APPLY_ADM パッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

タグを使用した変更の循環の回避

複数のデータベースがデータを両方向で共有している Streams 環境では、タグを使用して**変更の循環**を回避できます。変更の循環とは、変更が発生場所であるデータベースに再送されることです。通常は、それぞれの変更が無限ループを介して発生場所であるデータベースに送信される可能性があるため、変更の循環を回避する必要があります。このようなループがあると、データベースに意図しないデータが格納され、環境のネットワーキング・リソースとコンピュータ・リソースが過剰に使用される可能性があります。Streams は、デフォルトで変更の循環を回避するように設計されています。

Streams の取得プロセス、伝播および適用プロセスにタグと適切なルールを使用すると、このような変更の循環を回避できます。ここでは、各種 Streams 環境と、これらの環境でタグとルールを使用して変更の循環を回避する方法について説明します。

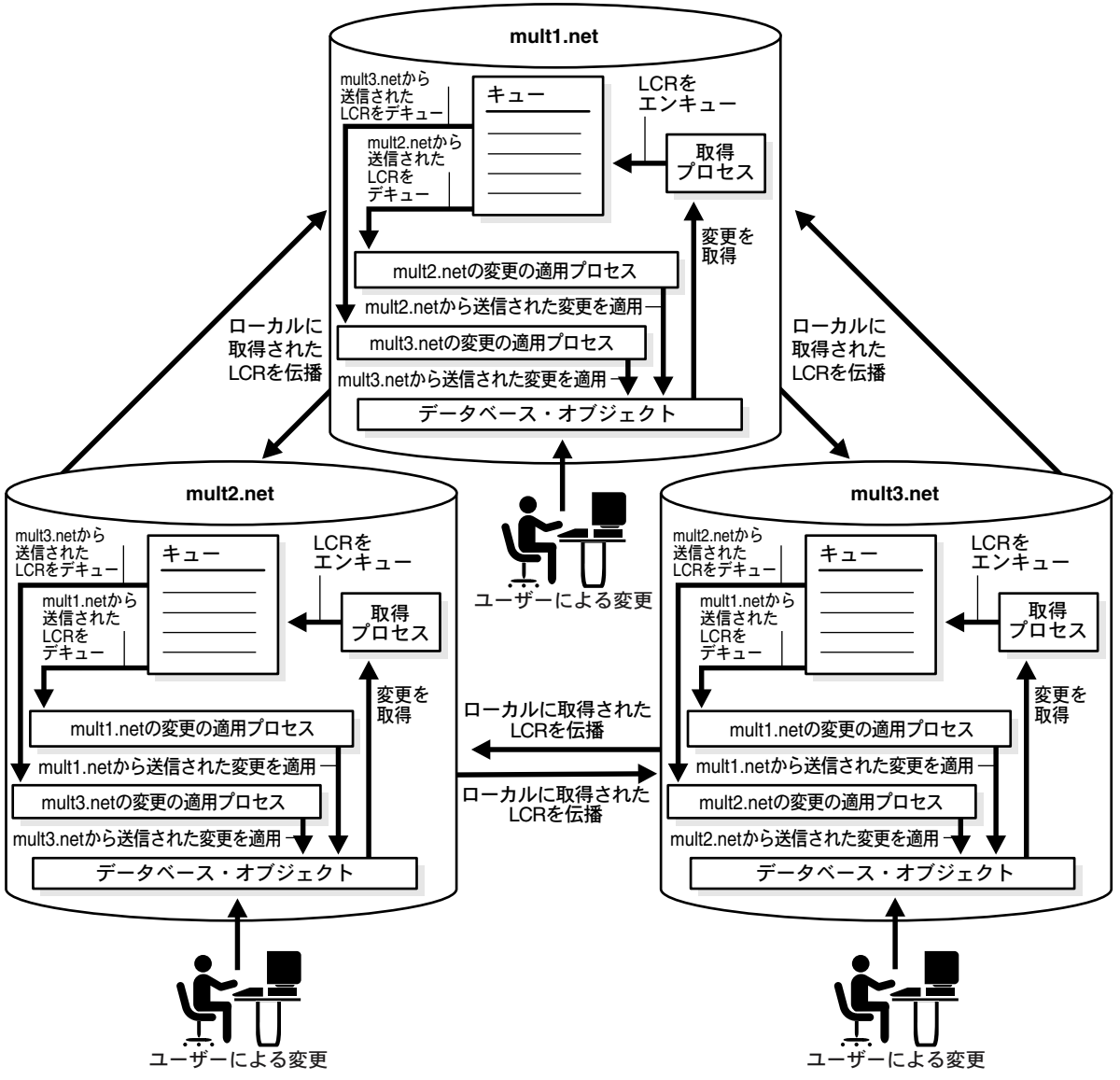
- 各データベースが共有データのソース・データベースと接続先データベースを兼ねている場合
- プライマリ・データベースが複数のセカンダリ・データベースとデータを共有している場合
- プライマリ・データベースが複数の拡張セカンダリ・データベースとデータを共有している場合

各データベースが共有データのソース・データベースと接続先データベースを兼ねている場合

この使用例は、各データベースが他のすべてのデータベースのソース・データベースと接続先データベースを兼ねている Streams 環境を示します。各データベースは、他の各データベースと直接通信します。

たとえば、hr スキーマ内のデータベース・オブジェクトとデータを、3 つの Oracle データベース mult1.net、mult2.net および mult3.net 間でレプリケートする環境を考えます。hr スキーマ内の表に対する DML 変更と DDL 変更は、この環境の 3 つのデータベースすべてで取得され、環境内の他の各データベースに伝播されて、そこで適用されます。図 8-1 に、各データベースがソース・データベースである環境の例を示します。

図 8-1 各データベースがソース・データベースと接続先データベースを兼ねている場合



前述の環境を次のように構成すると、変更の循環を回避できます。

- 各データベースで、1 つの適用プロセスを各ソース・データベースからの変更について非 NULL の REDO タグを生成するように構成します。DBMS_STREAMS_ADM パッケージのプロシージャを使用して適用プロセスを作成すると、その適用プロセスでは、デフォルトで REDO ログに値 '00' を持つ非 NULL のタグが生成されます。この場合は、それ以上の操作をしなくても、適用プロセスでは非 NULL のタグが生成されます。

  DBMS_APPLY_ADM パッケージの CREATE_APPLY プロシージャを使用して適用プロセスを作成する場合は、apply_tag パラメータを設定しないでください。この場合も、適用プロセスでは、デフォルトで REDO ログに値 '00' を持つ非 NULL のタグが生成され、それ以上の操作は不要です。

- 各データベースの取得プロセスを、変更に関する REDO エントリ内のタグが NULL の場合にのみ、その変更を取得するように構成します。そのためには、取得プロセスで使用されるルール・セットの各 DML ルールに、次の条件が含まれていることを確認します。

  ```
  :dml.is_null_tag()='Y'
  ```

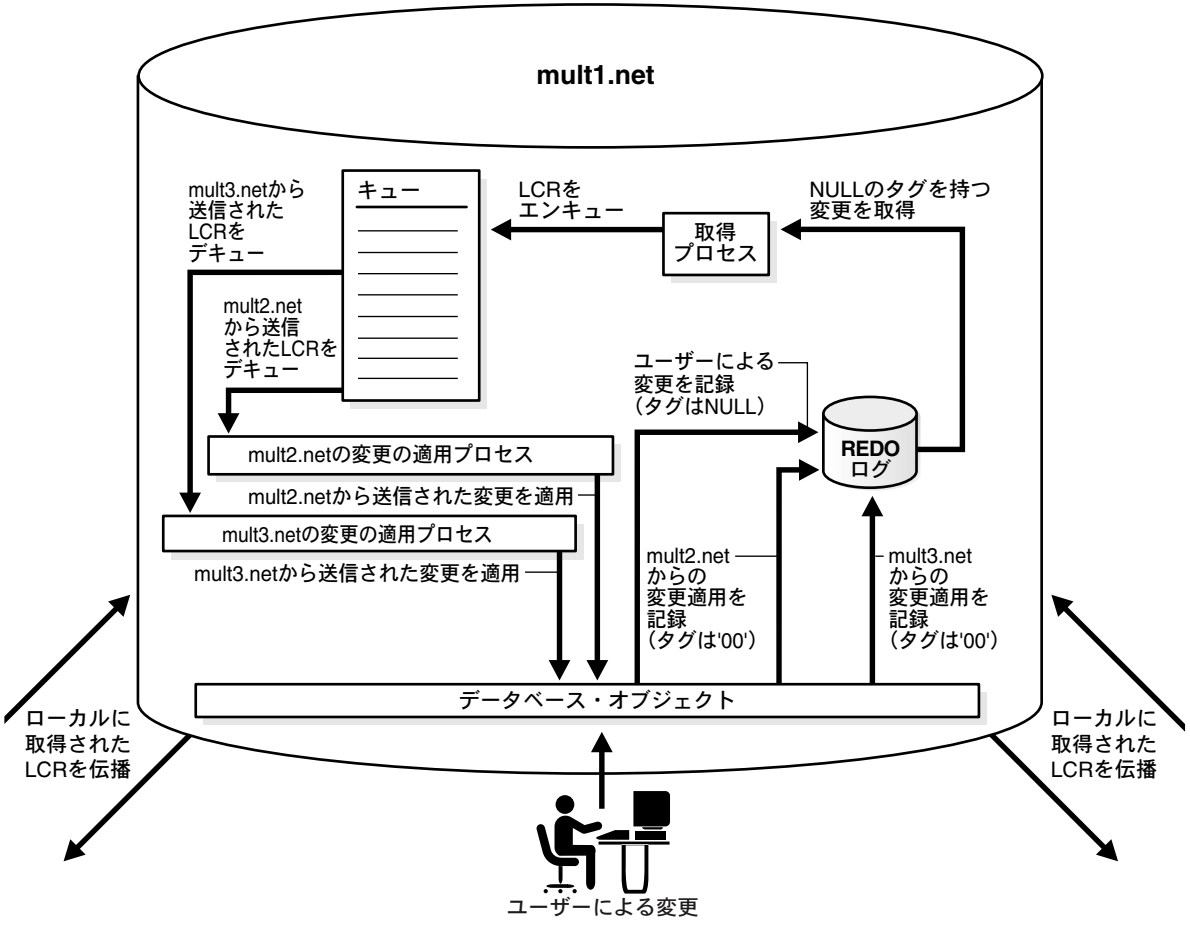
  各 DDL ルールには、次の条件が必要です。

  ```
  :ddl.is_null_tag()='Y'
  ```

  これらのルール条件は、変更のタグが NULL の場合にのみ取得プロセスで変更が取得されることを示します。DBMS_STREAMS_ADM パッケージを使用してルールを生成すると、各ルールにはデフォルトで前述の条件の 1 つが含まれます。

この構成では、適用プロセスによって適用されたすべての変更は、再取得されることがないため（最初にソース・データベースで取得済みであるため）、変更の循環が防止されます。各データベースからは、hr スキーマに対するすべての変更が他の各データベースに送信されます。そのため、この環境では、変更が失われることはなく、すべてのデータベースが同期化されます。図 8-2 に、複数のソースを含む環境でデータベース内でタグを使用する方法を示します。

図 8-2 各データベースがソース・データベースと接続先データベースを兼ねている場合のタグの使用



関連項目： この例は、第 23 章「複数のソース・レプリケーションの例」を参照してください。

プライマリ・データベースが複数のセカンダリ・データベースとデータを共有している場合

この使用例は、1 つのデータベースがプライマリ・データベースで、このプライマリ・データベースが複数のセカンダリ・データベースとデータを共有している Streams 環境を示します。セカンダリ・データベースは、プライマリ・データベースとのみデータを共有します。セカンダリ・データベースは、データを相互に直接共有するかわりに、プライマリ・データベースを介して間接的に共有します。この種の環境は、ハブ・アンド・スポーク環境と呼ばれることがあり、プライマリ・データベースがハブ、セカンダリ・データベースがスポークの役割を果たします。

この種の環境では、変更は次のように取得、伝播および適用されます。

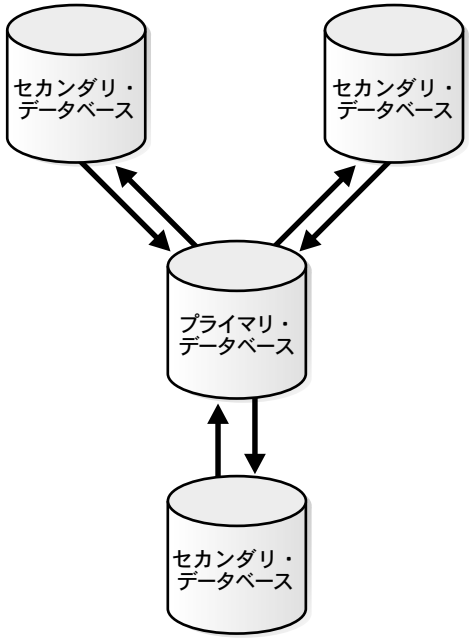
- プライマリ・データベースでは、共有データに対するローカルの変更が取得され、それがすべてのセカンダリ・データベースに伝播され、それぞれのセカンダリ・データベースでローカルに適用されます。
- それぞれのセカンダリ・データベースでは、共有データに対するローカルの変更が取得され、それがプライマリ・データベースにのみ伝播され、プライマリ・データベースでローカルに適用されます。
- プライマリ・データベースでは、それぞれのセカンダリ・データベースからの変更がローカルに適用されます。次に、これらの変更がプライマリ・データベースで取得され、変更の発生場所となったデータベースを除くすべてのセカンダリ・データベースに伝播されます。それぞれのセカンダリ・データベースでは、他のセカンダリ・データベースからの変更がプライマリ・データベースを経由してからローカルに適用されます。この構成は、適用による転送の一例です。

  代替使用例では、キューによる転送を使用できます。この環境でキューによる転送を使用した場合、セカンダリ・データベースからの変更がプライマリ・データベースで適用されても、プライマリ・データベースでは取得されません。かわりに、これらの変更は、プライマリ・データベースのキューから、変更の発生場所となったデータベースを除くすべてのセカンダリ・データベースに転送されます。

  関連項目： 適用による転送とキューによる転送の詳細は、3-6 ページの「有向ネットワーク」を参照してください。

たとえば、1 つのプライマリ・データベース `ps1.net` と 3 つのセカンダリ・データベース `ps2.net`、`ps3.net` および `ps4.net` 間で、`hr` スキーマ内のデータベース・オブジェクトとデータをレプリケートする環境を考えます。この環境では、`hr` スキーマ内の表に対する DML 変更と DDL 変更は、プライマリ・データベースと 3 つのセカンダリ・データベースで取得されます。次に、これらの変更が前述のとおり伝播され、適用されます。この環境では、プライマリ・データベースを介してセカンダリ・データベース間でデータを共有するために、キューによる転送ではなく適用による転送が使用されます。図 8-3 に、1 つのプライマリ・データベースと複数のセカンダリ・データベースを含む環境の例を示します。

図 8-3 プライマリ・データベースが複数のセカンダリ・データベースとデータを共有している場合



この環境を次のように構成すると、変更の循環を回避できます。

- プライマリ・データベース ps1.net の各適用プロセスを、変更の送信元サイトを示す非 NULL の REDO タグを生成するように構成します。この環境の場合、プライマリ・データベースには、変更の送信元となるセカンダリ・データベースごとに少なくとも 1 つは適用プロセスがあります。たとえば、プライマリ・データベースの適用プロセスは、ps2.net 2 次サイトから変更を受信すると、適用するすべての変更について 16 進値 '2' と等価の RAW 値を生成できます。そのためには、DBMS_APPLY_ADM パッケージの CREATE_APPLY または ALTER_APPLY プロシージャで、apply_tag パラメータを非 NULL 値に設定します。

たとえば、次のプロシージャを実行し、16 進値 '2' と等価のタグを持つ REDO エントリを生成する適用プロセスを作成します。

```
BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name      => 'strmadmin.streams_queue',
    apply_name      => 'apply_ps2',
    rule_set_name   => 'strmadmin.apply_rules_ps2',
    apply_tag       => HEXTORAW('2'),
    apply_captured  => true);
END;
/
```

- それぞれのセカンダリ・データベースで、非 NULL の REDO タグを生成するように適用プロセスを構成します。タグの正確な値は、非 NULL であるかぎり無関係です。この環境の場合、それぞれのセカンダリ・データベースには、プライマリ・データベースからの変更を適用する適用プロセスが 1 つあります。

  DBMS_STREAMS_ADM パッケージのプロシージャを使用して適用プロセスを作成すると、その適用プロセスでは、デフォルトで REDO ログに値 '00' を持つ非 NULL のタグが生成されます。この場合は、それ以上の操作をしなくても、適用プロセスでは非 NULL のタグが生成されます。

  たとえば、セカンダリ・データベースに適用プロセスが存在せず、各セカンダリ・データベースで DBMS_STREAMS_ADM パッケージの ADD_SCHEMA_RULES プロシージャを実行して、16 進値 '00' と等価のタグを持つ非 NULL の REDO エントリを生成する適用プロセスを作成する場合を考えます。

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name     => 'hr',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'ps1.net');
END;
/
```

- プライマリ・データベースで、タグに関係なく共有データの変更を取得するように取得プロセスを構成します。そのためには、DBMS_STREAMS_ADM パッケージ内の取得ルールを生成するプロシージャの 1 つを実行するときに、include_tagged_lcr パラメータを true に設定します。DBMS_RULE_ADM パッケージを使用してプライマリ・データベースの取得プロセスのルールを作成する場合は、そのルールに is_null_tag 条件が含まれないことを確認してください。これらの条件には、REDO ログ内のタグが関係するためです。

たとえば、プライマリ・データベースで次のプロシージャを実行し、DML 取得プロセスのルールを 1 つと DDL 取得プロセスのルールを 1 つ生成する場合を考えます。どちらのルールにも、hr スキーマ内の変更について、そのタグに関係なく TRUE と評価される条件が含まれているとします。

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name         => 'hr',
    streams_type        => 'capture',
    streams_name        => 'capture',
    queue_name          => 'strmadmin.streams_queue',
    include_tagged_lcr  =>  true, -- Note parameter setting
    include_dml         =>  true,
    include_ddl         =>  true);
END;
/
```

- 各セカンダリ・データベースの取得プロセスを、変更に関する REDO エントリ内のタグが NULL の場合にのみ、その変更を取得するように構成します。そのためには、セカンダリ・データベースの取得プロセスで使用されるルール・セットの各 DML ルールに、次の条件が含まれていることを確認します。

  ```
  :dml.is_null_tag()='Y'
  ```

  各 DDL ルールには、次の条件が必要です。

  ```
  :ddl.is_null_tag()='Y'
  ```

  これらのルールは、変更のタグが NULL の場合にのみ取得プロセスで変更が取得されることを示します。DBMS_STREAMS_ADM パッケージを使用してルールを生成すると、各ルールにはデフォルトで前述の条件の 1 つが含まれます。DBMS_RULE_ADM パッケージを使用してセカンダリ・データベースの取得プロセスのルールを作成する場合は、各ルールに前述の条件の 1 つが含まれることを確認してください。

- プライマリ・データベースのキューからそれぞれのセカンダリ・データベースのキューへの伝播を 1 つ構成します。各伝播で使用するルール・セットには、セカンダリ・データベースで発生した変更を除き、プライマリ・データベースのキューにあるすべての LCR をセカンダリ・データベースのキューに伝播するように指示するルールを含める必要があります。

  たとえば、伝播がセカンダリ・データベース ps2.net に変更を伝播する場合に、そのタグが 16 進値 '2' と等価であれば、伝播のルールでは、タグが '2' の LCR を除き、hr スキーマに関連するすべての LCR をセカンダリ・データベースに伝播する必要があります。行 LCR の場合は、この種のルールに次の条件を含める必要があります。

  ```
  :dml.get_tag()!=HEXTORAW('2')
  ```

DDL LCR の場合は、この種のルールに次の条件を含める必要があります。

```
:ddl.get_tag()!=HEXTORAW('2')
```

DBMS_RULE_ADM パッケージの CREATE_RULE プロシージャを使用すると、これらの条件を持つルールを作成できます。

- それぞれのセカンダリ・データベースのキューからプライマリ・データベースのキューへの伝播を 1 つ構成します。セカンダリ・データベースの 1 つにあるキューには、適用プロセスによって行われた変更ではなく、そのデータベースでユーザー・セッションおよびアプリケーションによってローカルに行われた変更のみが含まれます。したがって、これらの伝播については、それ以上の構成は不要です。

この構成では、次の方法で変更の循環が防止されます。

- セカンダリ・データベースで発生した変更が、そのセカンダリ・データベースに再び伝播されることはありません。
- プライマリ・データベースで発生した変更が、そのプライマリ・データベースに再び伝播されることはありません。
- 環境内のどのデータベースで共有データに対して行われた変更も、すべて環境内の他の各データベースに伝播します。

そのため、この環境では変更が失われることはなく、すべてのデータベースが同期化されます。

図 8-4 に、プライマリ・データベース ps1.net でのタグの使用方法を示します。

図 8-4 プライマリ・データベースで使用されるタグ

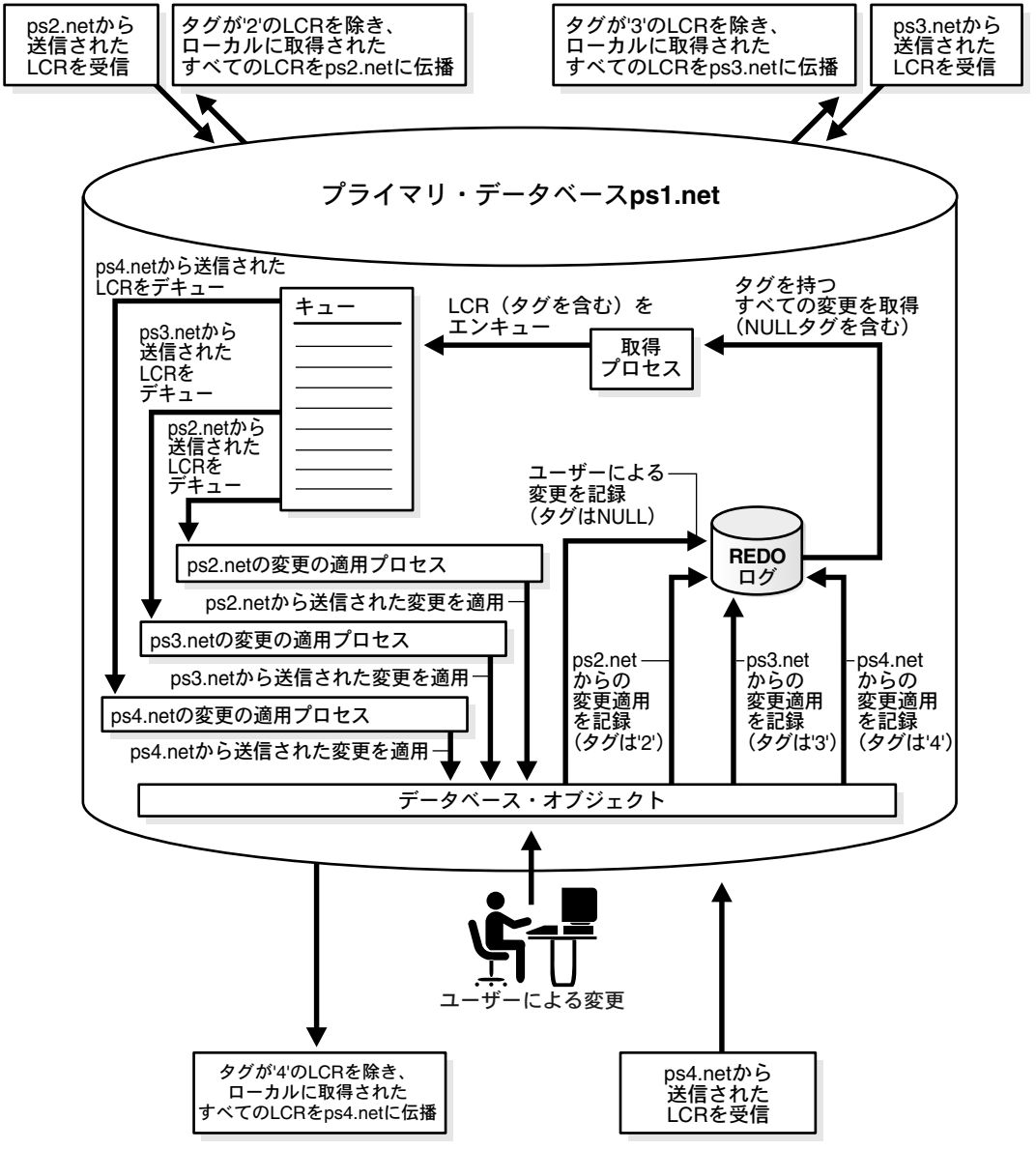
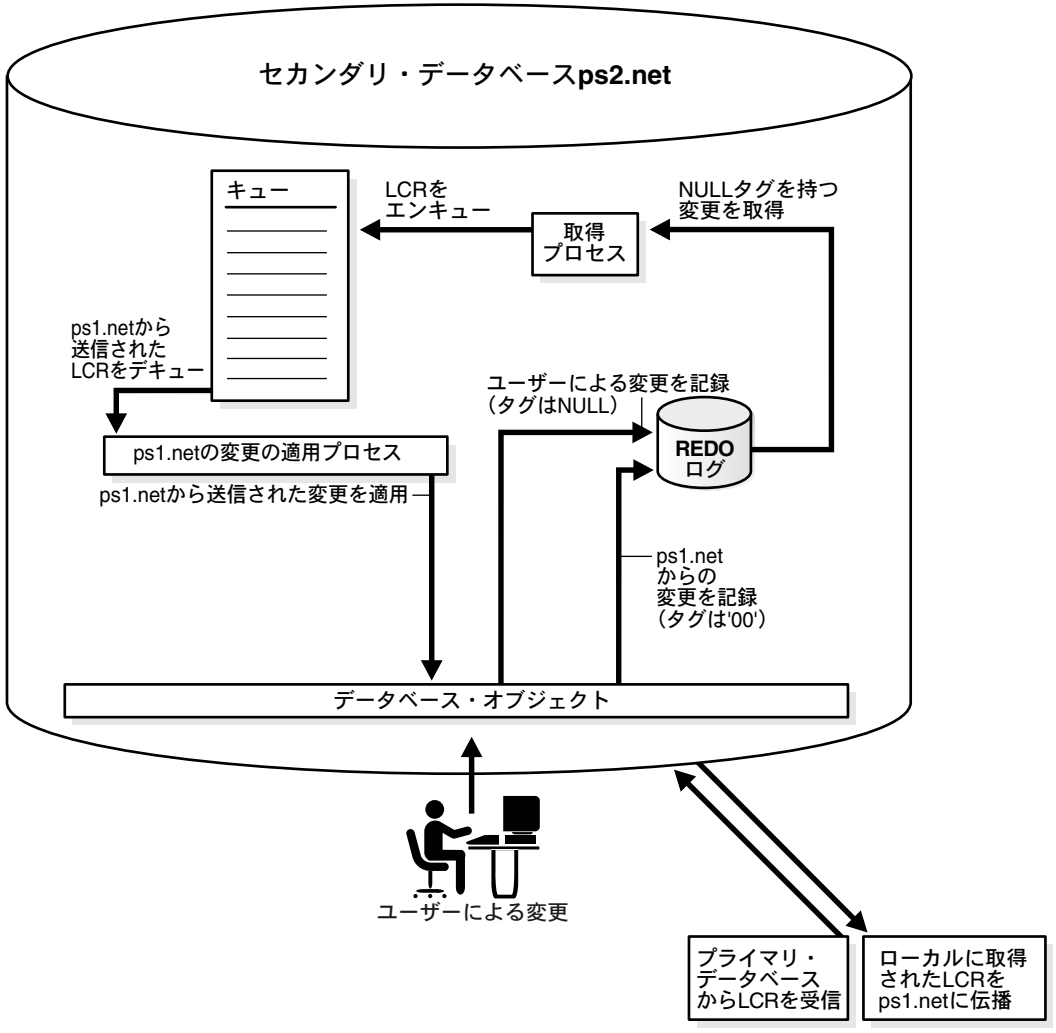


図 8-5 に、セカンダリ・データベースの 1 つ（ps2.net）でのタグの使用方法を示します。

図 8-5 セカンダリ・データベースで使用されるタグ

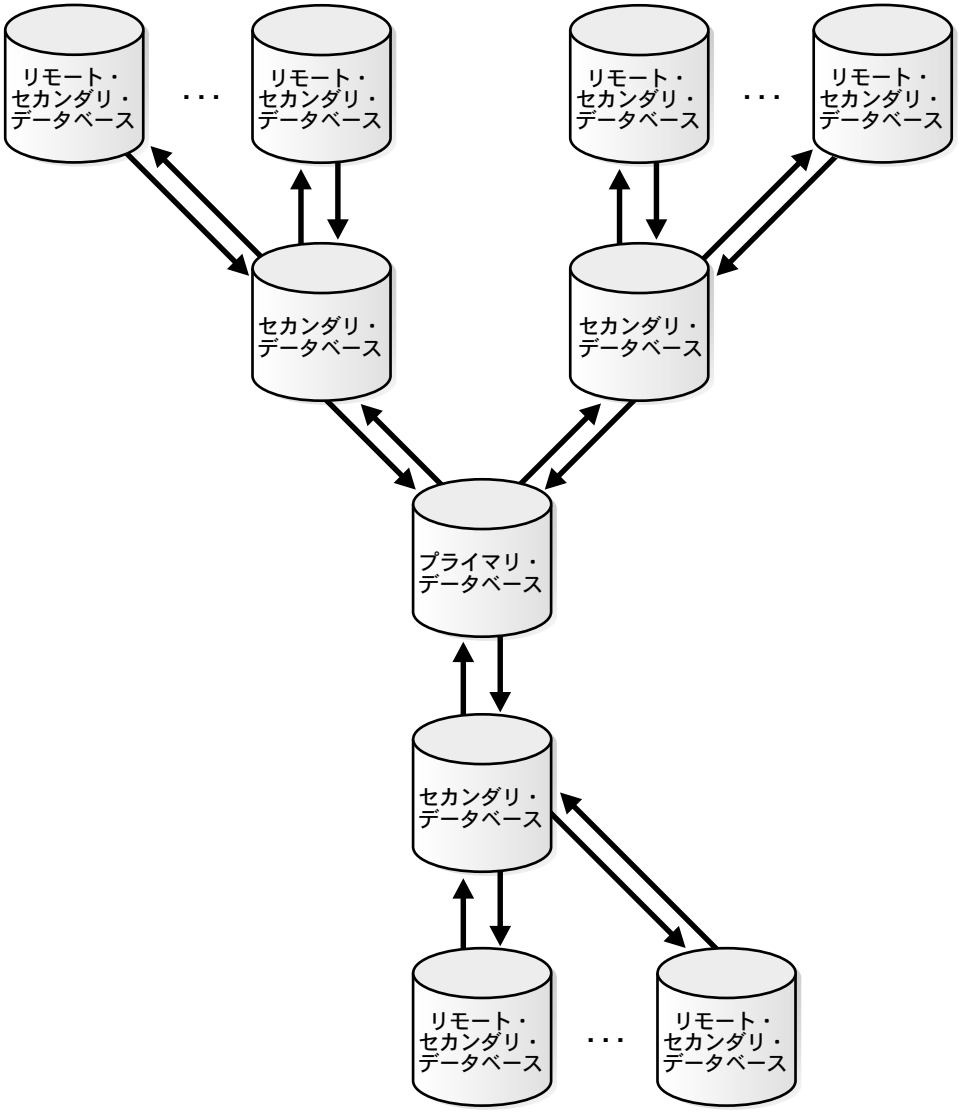


プライマリ・データベースが複数の拡張セカンダリ・データベースとデータを共有している場合

この環境では、1 つのプライマリ・データベースが複数のセカンダリ・データベースとデータを共有しますが、セカンダリ・データベースにはリモート・セカンダリ・データベースと呼ばれる他のセカンダリ・データベースが接続されています。この環境は、8-11 ページの「プライマリ・データベースが複数のセカンダリ・データベースとデータを共有している場合」で説明した環境が拡張されたものです。

リモート・セカンダリ・データベースはデータをプライマリ・データベースと直接共有するのではなく、セカンダリ・データベースを介して間接的に共有します。そのため、共有データはプライマリ・データベース、それぞれのセカンダリ・データベースおよびそれぞれのリモート・セカンダリ・データベースに存在します。これらのデータベースのいずれかで行われた変更は、他のすべてのデータベースで取得され、伝播されます。図 8-6 に、1 つのプライマリ・データベースと複数の拡張セカンダリ・データベースを含む環境を示します。

図 8-6 プライマリ・データベースと複数の拡張セカンダリ・データベース



この種の環境では、次の方法で変更の循環を回避できます。

- 8-11 ページの「プライマリ・データベースが複数のセカンダリ・データベースとデータを共有している場合」で説明した例の場合と同じ方法で、プライマリ・データベースを構成します。
- 8-11 ページの「プライマリ・データベースが複数のセカンダリ・データベースとデータを共有している場合」で説明した例でそれぞれのセカンダリ・データベースを構成したのと同様の方法で、それぞれのリモート・セカンダリ・データベースを構成します。唯一の違いは、リモート・セカンダリ・データベースは、プライマリ・データベースではなくセカンダリ・データベースとデータを直接共有することです。
- それぞれのセカンダリ・データベースで、プライマリ・データベースからの変更のうち、16 進値 '00' と等価の REDO タグ値を持つ変更を適用するように、適用プロセスを 1 つ構成します。この値は、適用プロセスのデフォルトのタグ値です。
- それぞれのセカンダリ・データベースで、各リモート・セカンダリ・データベースからの変更のうち、そのリモート・セカンダリ・データベースに一意の REDO タグ値を持つ変更を適用するように、適用プロセスを 1 つ構成します。
- それぞれのセカンダリ・データベースの取得プロセスを、REDO ログ内の共有データに対する変更を、そのタグ値に関係なくすべて取得するように構成します。
- それぞれのセカンダリ・データベースのキューからプライマリ・データベースのキューへの伝播を 1 つ構成します。伝播で使用するルール・セットには、プライマリ・データベースで発生した変更を除き、セカンダリ・データベースのキューにあるすべての LCR をプライマリ・データベースのキューに伝播するように指示するルールを含める必要があります。そのためには、LCR 内のタグが '00' でない場合にのみ TRUE と評価される条件をルールに追加します。たとえば、行 LCR について次のような条件を入力します。

  ```
  :dml.get_tag()!=HEXTORAW('00')
  ```

- それぞれのセカンダリ・データベースのキューからそれぞれのリモート・セカンダリ・データベースのキューへの伝播を 1 つ構成します。各伝播で使用するルール・セットには、リモート・セカンダリ・データベースで発生した変更を除き、セカンダリ・データベースのキューにあるすべての LCR をリモート・セカンダリ・データベースのキューに伝播するように指示するルールを含める必要があります。そのためには、LCR 内のタグがリモート・セカンダリ・データベースのタグ値と異なる場合にのみ TRUE と評価される条件をルールに追加します。たとえば、リモート・セカンダリ・データベースのタグ値が 16 進値 '19' と等価の場合は、行 LCR について次のような条件を入力します。

  ```
  :dml.get_tag()!=HEXTORAW('19')
  ```

この方法で環境を構成すると、変更の循環が防止され、どのデータベースで発生した変更も失われません。

9

Streams 異機種間での情報の共有

この章では、Oracle データベースと Oracle 以外のデータベースの間で情報を共有するための、Streams のサポートに関連する概念について説明します。

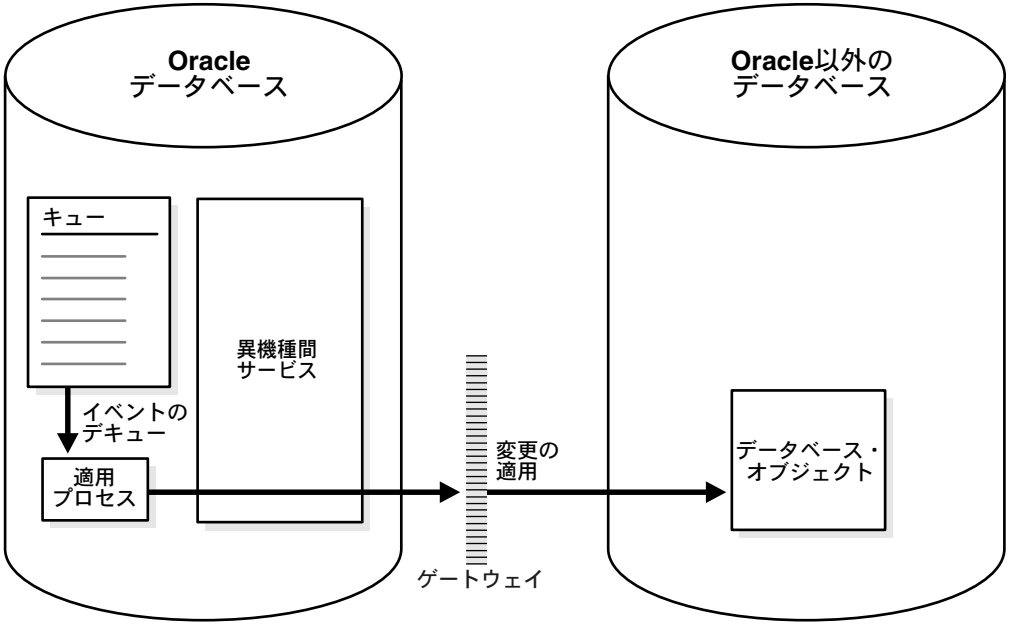
この章の内容は次のとおりです。

- Streams を使用した Oracle データベースから Oracle 以外のデータベースへのデータの共有
- Streams を使用した Oracle 以外のデータベースから Oracle データベースへのデータの共有
- Streams を使用した Oracle 以外のデータベース間でのデータの共有

Streams を使用した Oracle データベースから Oracle 以外のデータベースへのデータの共有

Oracle ソース・データベースから Oracle 以外の接続先データベースへと DML 変更を共有するために、Oracle データベースはプロキシとして機能し、通常は接続先データベースで実行される手順の一部を実行します。つまり、Oracle 以外の接続先データベースを対象とするイベントは、Oracle データベース自体でデキューされ、Oracle データベースの適用プロセスはゲートウェイを介してネットワーク接続経由で変更を Oracle 以外のデータベースに適用します。図 9-1 に、Oracle 以外のデータベースとデータを共有している Oracle データベースを示します。

図 9-1 Oracle データベースから Oracle 以外のデータベースへの異機種間データ共有



Oracle から Oracle 以外への環境での変更の取得とステージング

Oracle から Oracle 以外への環境では、取得プロセスは Oracle のみの環境の場合と同様に動作します。つまり、REDO ログ内で変更を検索し、それを取得プロセスのルールに基づいて取得し、取得された変更を SYS.AnyData キューに論理変更レコード（LCR）としてエンキューします。また、1 つの取得プロセスで、Oracle データベースと Oracle 以外のデータベースの両方で適用される変更を取得できます。

同様に、取得された LCR をステージングする SYS.AnyData キューは、Oracle のみの環境の場合と同様に動作します。Oracle 以外のデータベースで適用される前に、Oracle データベース内の任意の数の中間キューに LCR を伝播させることができます。

関連項目：

- 第 2 章「Streams の取得プロセス」
- 第 3 章「Streams のステージングと伝播」

Oracle から Oracle 以外への環境での変更の適用

Oracle データベース内で実行される適用プロセスは、異機種間サービスとゲートウェイを使用して、LCR にカプセル化された変更を Oracle 以外のデータベース内のデータベース・オブジェクトに直接適用します。LCR は、Oracle のみの Streams 環境の場合のようには、Oracle 以外のデータベース内のキューに伝播しません。かわりに、変更は適用プロセスによってデータベース・リンクを介して Oracle 以外のデータベースに直接適用されます。

関連項目： 適用プロセスの詳細は、第 4 章「Streams 適用プロセス」を参照してください。

Oracle から Oracle 以外への環境での適用プロセスの構成

この項では、変更を Oracle 以外のデータベースに適用する適用プロセスの構成について説明します。

Oracle 以外のデータベースへのデータベース・リンク 変更を Oracle 以外のデータベースに適用する適用プロセスを作成する場合は、そのために適用プロセスで使用される異機種間サービス、ゲートウェイおよびデータベース・リンクをあらかじめ構成する必要があります。データベース・リンクの作成には、明示的な CONNECT TO 句を使用してください。

データベース・リンクを作成し、正常に動作している場合は、DBMS_APPLY_ADM パッケージの CREATE_APPLY プロシージャを使用して適用プロセスを作成し、apply_database_link パラメータにデータベース・リンクを指定します。適用プロセスの作成後は、そのルールを使用して Oracle 以外のデータベースで適用する変更を指定できます。

関連項目：

- 異機種間サービスとゲートウェイの詳細は、『Oracle9*i* Heterogeneous Connectivity Administrator's Guide』を参照してください。
- DBMS_APPLY_ADM パッケージのプロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 適用プロセスのルールを指定する方法は、第 6 章「Streams でのルールの使用方法」を参照してください。

Oracle から Oracle 以外への異機種間環境での代替キー列 Oracle 以外のデータベースでいずれかの表に代替キー列を使用する場合は、DBMS_APPLY_ADM パッケージの SET_KEY_COLUMNS プロシージャを実行するときに、そのデータベースへのデータベース・リンクを指定します。

関連項目：

- 4-11 ページ「代替キー列」
- 14-26 ページ「表の代替キー列の管理」

Oracle から Oracle 以外への異機種間環境での並列性 適用プロセスで Oracle 以外のデータベースに変更を適用する場合は、parallelism 適用プロセス・パラメータをデフォルト設定の 1 に設定する必要があります。現在、Oracle 以外のデータベースへのパラレル適用はサポートされていません。ただし、複数の適用プロセスを使用すると、Oracle 以外のデータベースに変更を適用できます。

Oracle から Oracle 以外への異機種間環境での DML ハンドラ DML ハンドラを使用して Oracle 以外のデータベースでいずれかの表の行 LCR を処理する場合は、DBMS_APPLY_ADM パッケージの SET_DML_HANDLER プロシージャを実行するときに、そのデータベースへのデータベース・リンクを指定します。

関連項目：

- 4-4 ページ「イベント処理オプション」
- 14-13 ページ「DML ハンドラの管理」

Oracle から Oracle 以外への異機種間環境でのメッセージ・ハンドラ メッセージ・ハンドラを使用して Oracle 以外のデータベース用のユーザー・エンキュー・メッセージを処理する場合は、DBMS_APPLY_ADM パッケージの CREATE_APPLY プロシージャを実行するときに、apply_database_link パラメータを使用してそのデータベースへのデータベース・リンクを指定し、message_handler パラメータを使用してメッセージ・ハンドラ・プロシージャを指定します。

関連項目：

- 4-4 ページ「イベント処理オプション」
- 14-12 ページ「適用プロセスのメッセージ・ハンドラの管理」

Oracle から Oracle 以外への異機種間環境でのエラーおよび競合ハンドラ 現在、Oracle データベースから Oracle 以外のデータベースへとデータを共有する場合、エラー・ハンドラと競合ハンドラはサポートされていません。適用エラーが発生すると、エラーの原因となった LCR を含むトランザクションは、Oracle データベースの例外キューに移動します。

Oracle 以外のデータベースで適用されるデータ型

変更を Oracle 以外のデータベースに適用する場合、適用プロセスでは次のデータ型の列に対する変更のみが適用されます。

- CHAR
- VARCHAR2
- NCHAR
- NVARCHAR2
- NUMBER
- DATE
- RAW
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

適用プロセスでは、データ型のうち CLOB、NCLOB、BLOB、BFILE、LONG、LONG RAW、ROWID、UROWID およびユーザー定義型（オブジェクト型、REF、VARRAY および NESTED TABLE など）の列における変更は、Oracle 以外のデータベースには適用されません。これ以外のデータ型が LCR に含まれている場合、適用プロセスにエラーが発生し、エラーの原因となった LCR を含むトランザクションが Oracle データベースの例外キューに移動します。

各 Transparent Gateway は、データ型に関してさらに制限を伴う場合があります。あるデータ型が Oracle から Oracle 以外への環境でサポートされるには、そのデータ型が Streams と使用ゲートウェイの両方でサポートされる必要があります。

関連項目：

- これらのデータ型の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- Transparent Gateway の詳細は、オラクル社が提供するゲートウェイ固有のマニュアルを参照してください。

Oracle 以外のデータベースで適用される DML 変更のタイプ

特定の表に対する DML 変更を Oracle 以外のデータベースで適用するように指定する場合、適用プロセスで適用できるのは次のタイプの DML 変更のみです。

- `INSERT`
- `UPDATE`
- `DELETE`

注意： 適用プロセスで DDL 変更を Oracle 以外のデータベースで適用することはできません。

Oracle から Oracle 以外への環境でのインスタンス化

変更を Oracle 以外のデータベースに適用する適用プロセスを起動する前に、次の手順に従って、そのデータベースで各表をインスタンス化します。

1. `DBMS_HS_PASSTHROUGH` パッケージまたは Oracle 以外のデータベースに用意されているツールを使用して、Oracle 以外のデータベースで表を作成します。

   関連項目： 異機種間サービスと Transparent Gateway の詳細は、『Oracle9*i* Heterogeneous Connectivity Administrator's Guide』とオラクル社が提供するゲートウェイ固有のマニュアルを参照してください。

2. Oracle データベースと Oracle 以外のデータベース間で共有される変更が Oracle データベース側の取得プロセスによって取得される場合は、データを共有予定の表すべてをインスタンス化のために準備します。

   関連項目： 12-10 ページ「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」

3. 次のアクションを実行する PL/SQL プロシージャ（または C プログラム）を作成します。
   - `DBMS_FLASHBACK` パッケージの `GET_SYSTEM_CHANGE_NUMBER` ファンクションを使用し、現行の SCN を取得する。

- DBMS_FLASHBACK パッケージの ENABLE_AT_SYSTEM_CHANGE_NUMBER プロシージャを起動し、取得された SCN を現行セッションに設定する。このアクションによって、すべてのフェッチが同じ SCN を使用して実行されることが確実になります。
- Oracle データベースの表から行を 1 行ずつフェッチして Oracle 以外のサイトの表に移入することにより、Oracle 以外のデータベースの表に対する挿入を行う。すべてのフェッチは、GET_SYSTEM_CHANGE_NUMBER ファンクションを使用して取得された SCN で実行する必要があります。

たとえば、次の PL/SQL プロシージャでは、フラッシュバック SCN を取得し、現行の Oracle データベース内の hr.regions 表の各行をフェッチして、Oracle 以外のデータベース het.net 内の hr.regions 表に挿入しています。Oracle 以外のデータベースに行が挿入される前に、フラッシュバックが無効化されていることに注意してください。

```
SET SERVEROUTPUT ON
CREATE OR REPLACE PROCEDURE insert_reg IS
  CURSOR c1 IS
    SELECT region_id, region_name FROM hr.regions;
  c1_rec c1 % ROWTYPE;
  scn NUMBER;
BEGIN
  scn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER(
     query_scn  =>  scn);
  /* Open c1 in flashback mode */
  OPEN c1;
  /* Disable Flashback */
  DBMS_FLASHBACK.DISABLE;
  LOOP
    FETCH c1 INTO c1_rec;
    EXIT WHEN c1%NOTFOUND;
   /*
     Note that all the DML operations inside the loop are performed
     with Flashback disabled
   */
   INSERT INTO hr.regions@het.net VALUES (
     c1_rec.region_id,
     c1_rec.region_name);
  END LOOP;
  COMMIT;
  DBMS_OUTPUT.PUT_LINE('SCN = ' || scn);
  EXCEPTION WHEN OTHERS THEN
    DBMS_FLASHBACK.DISABLE;
    RAISE;
END;
/
```

戻された SCN をメモします。

注意： このプロシージャを作成および実行するユーザーには、DBMS_FLASHBACK パッケージについての EXECUTE 権限と、関連する表についての全権限が必要です。

---

4. DBMS_APPLY_ADM パッケージの SET_TABLE_INSTANTIATION_SCN プロシージャに対して手順 3 で取得された SCN を指定し、手順 3 で取得された SCN より前に発生した変更を含む LCR をすべてスキップするように、適用プロセスへ指示します。apply_database_link パラメータを、Oracle 以外のリモート・データベース用のデータベース・リンクに設定してください。

   関連項目： DBMS_APPLY_ADM パッケージの SET_TABLE_INSTANTIATION_SCN プロシージャの詳細は、14-33 ページの「接続先データベースでのインスタンス化 SCN の設定」および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

Oracle から Oracle 以外への環境での変換

Oracle から Oracle 以外への環境での変換では、Oracle のみの環境の場合と同様に、取得または適用中のルールベースの変換を指定できます。また、LCR を Oracle 以外のデータベースに適用する前に、1 つ以上の Oracle 中間データベースに伝播させる場合は、Oracle データベースのキューから Oracle データベースの他のキューへの伝播中に、ルールベースの変換を指定できます。

関連項目： 6-23 ページ「ルールベースの変換」

メッセージ・ゲートウェイと Streams

メッセージ・ゲートウェイは Oracle データベースの機能であり、Oracle キューと Oracle 以外のメッセージ・キューイング・システムの間の伝播を提供します。Oracle キューにエンキューされたメッセージは Oracle 以外のキューに自動的に伝播し、Oracle 以外のキューにエンキューされたメッセージは Oracle キューに自動的に伝播します。この機能によって、Oracle 以外のメッセージ・システムへの保証付きメッセージ配信が提供され、Oracle 以外のメッセージ・システムについてはシステム固有のメッセージ・フォーマットがサポートされます。また、Oracle キューと Oracle 以外のメッセージ・システムの間での伝播中に起動される、ユーザー定義変換の仕様もサポートされます。

関連項目： メッセージ・ゲートウェイの詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

Oracle から Oracle 以外への環境でのエラー処理

Oracle 以外のデータベースで LCR を適用するときに適用プロセスに未処理エラーが発生すると、その LCR を含むトランザクションは、適用プロセスを実行中の Oracle データベースにある例外キューに置かれます。適用プロセスは、Oracle のみの環境の場合と同様にデータ競合を検出しますが、現在、Oracle から Oracle 以外への環境では、自動競合解消はサポートされていません。したがって、発生したデータ競合は適用エラーとして処理されます。

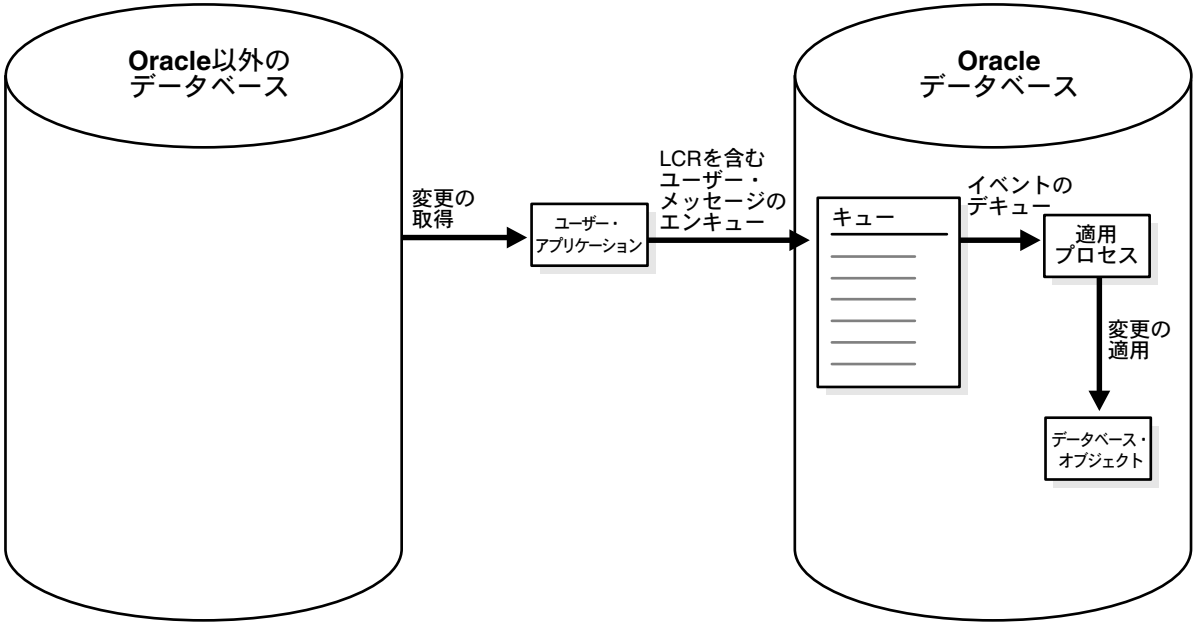
Oracle から Oracle 以外への Streams 環境の例

Oracle から Oracle 以外への Streams 環境内のデータ共有を含む詳細例は、第 22 章「単一ソースの異機種間レプリケーションの例」を参照してください。

Streams を使用した Oracle 以外のデータベースから Oracle データベースへのデータの共有

Oracle 以外のデータベースから変更を取得して Oracle データベースに伝播させるには、カスタム・アプリケーションが必要です。このアプリケーションは、トランザクション・ログを読み込むか、トリガーを使用するか、または他のなんらかの方法で、Oracle 以外のデータベースに対する変更を取得します。このアプリケーションではトランザクションをアセンブルし、順序付けし、それぞれの変更を論理変更レコード（LCR）に変換する必要があります。次に、アプリケーションで DBMS_AQ パッケージを使用して、LCR を Oracle データベースのキューにエンキューします。また、アプリケーションでは、各トランザクション内のすべての LCR をエンキューした後にコミットする必要があります。図 9-2 に、Oracle データベースとデータを共有している Oracle 以外のデータベースを示します。

図 9-2 Oracle 以外のデータベースから Oracle データベースへの異機種間データ共有



Oracle 以外から Oracle への環境での変更の取得とステージング

カスタム・ユーザー・アプリケーションでは、Oracle 以外のデータベースでの変更を LCR にアセンブルし、LCR を Oracle データベースのキューにエンキューする必要があるため、変更の取得全体を受け持つことになります。これは、アプリケーションでは Oracle 以外のデータベースで変更を表す LCR を構成し、その LCR を Oracle データベースのキューにエンキューする必要があることを意味します。アプリケーションでは、Oracle 以外のソース・データベースでコミットされたのと同じ順序で、トランザクションをシリアルにエンキューする必要があります。

変更が適用される Oracle データベースと変更が発生する Oracle 以外のデータベースの両方で、同じトランザクション一貫性を確保する必要がある場合は、トランザクション型のキューを使用して Oracle データベースで LCR をステージングしてください。たとえば、1 つのトランザクションに 3 行の変更が含まれており、カスタム・アプリケーションでは変更ごとに 1 つずつ合計 3 つの行 LCR をエンキューしてからコミットする場合を考えます。トランザクション型のキューを使用すると、3 番目の行 LCR の後で適用プロセスによってコミットが実行され、トランザクションの一貫性が保持されます。非トランザクション型のキューを使用すると、コミットは適用プロセスによって行 LCR ごとに実行されます。`DBMS_STREAMS_ADM` パッケージの `SET_UP_QUEUE` プロシージャでは、自動的にトランザクション型のキューが作成されます。

関連項目：

- 16-2 ページ「LCR の構成とエンキュー」
- 3-10 ページ「SYS.AnyData 型のキューとユーザー・メッセージ」

Oracle 以外から Oracle への環境での変更の適用

Oracle 以外から Oracle への環境では、適用プロセスは Oracle のみの環境の場合と同様に動作します。つまり、各イベントは適用プロセスのルールに基づいて関連付けられたキューからデキューされ、ルールベースの変換が実行され、イベントがハンドラに送信されるか、または直接適用されます。エラー処理と競合解消の動作も、Oracle のみの環境の場合と同じです。そのため、ビルトインの更新の競合ハンドラを指定するか、カスタム競合ハンドラを作成して競合を解消できます。

関連項目：

- 第 4 章「Streams 適用プロセス」
- 第 5 章「ルール」
- 第 7 章「Streams 競合解消」
- 6-23 ページ「ルールベースの変換」

Oracle 以外のデータベースから Oracle データベースへのインスタンス化

Oracle 以外のデータベースに存在する表を、Oracle データベースで自動的にインスタンス化する方法はありません。ただし、次の一般的な手順に従って表を手動でインスタンス化できます。

1. Oracle 以外のデータベースで、Oracle 以外のユーティリティを使用して表をフラット・ファイルにエクスポートします。
2. Oracle データベースで、Oracle 以外のデータベースにある表と一致する空の表を作成します。
3. Oracle データベースで、SQL*Loader を使用してフラット・ファイルの内容を表にロードします。

   関連項目： SQL*Loader の使用については、『Oracle9*i* データベース・ユーティリティ』を参照してください。

Streams を使用した Oracle 以外のデータベース間でのデータの共有

Streams では、Oracle 以外の 2 つのデータベース間でのデータ共有が、Oracle 以外から Oracle へのデータ共有と Oracle から Oracle 以外へのデータ共有の組合せを介してサポートされます。この種の環境では、Oracle 以外の 2 つのデータベース間に位置する中間データベースとして、Oracle データベース内で Streams が使用されます。

たとえば、Oracle 以外から Oracle 以外への環境は、次のデータベースで構成されている場合があります。

- Oracle 以外のデータベース `het1.net`
- Oracle データベース `dbs1.net`
- Oracle 以外のデータベース `het2.net`

ユーザー・アプリケーションでは、`het1.net` での変更をアセンブルし、それを `dbs1.net` のキューにエンキューします。次に、`dbs1.net` の適用プロセスが、異機種間サービスとゲートウェイを使用して変更を `het2.net` に適用します。`dbs1.net` の別の適用プロセスは、`dbs1.net` でキューにある一部またはすべての変更をローカルに適用できます。`dbs1.net` の 1 つ以上の伝播は、キューにある一部またはすべての変更を他の Oracle データベースに伝播できます。

10

Streams 高可用性環境

この章では、Streams 高可用性環境に関連する概念を説明します。

この章の内容は次のとおりです。

Streams 高可用性環境の概要

高可用性ソリューションの構成には、念入りな計画と障害発生時の使用例の分析が必要です。データベース・バックアップとフィジカル・スタンバイ・データベースでは、フェイルオーバー保護用にソース・データベースの物理コピーが提供されます。Data Guard には、SQL 適用モードの場合、高可用性環境にロジカル・スタンバイ・データベースが実装されます。Data Guard は高可用性環境用に設計されているため、ほとんどの障害発生例に対応します。ただし、一部の環境で、Oracle Streams により実現された柔軟性が必要になる場合には、Streams によって提供される拡張機能セットが使用可能です。

この章では、Streams ベースのソリューションが適している使用例と、高可用性環境で発生する Streams 固有の問題を説明します。また、クラスタ内のハードウェア・フェイルオーバー、Oracle Real Application Clusters クラスタ内のインスタンス・フェイルオーバー、およびレプリカ間のフェイルオーバーとスイッチオーバーを含め、Streams を高可用性環境に配置する最良の実施例も説明します。

関連項目：

- Data Guard の詳細は、『Oracle9*i* Data Guard 概要および管理』を参照してください。
- 『Oracle9*i* Real Application Clusters 概要』

障害からの保護

インスタンスまたはシステム障害からの保護策には、Oracle Real Application Clusters が適しています。障害が発生すると、クラスタ内にある障害の発生していない方のノードによってサービスが提供されます。ただし、クラスタ化では、ユーザー・エラー、メディア障害および障害からは保護されません。この種の障害には、データベースの重複コピーが必要です。データベースの物理コピーと論理コピーの両方を作成できます。

物理コピーはブロック単位でソース・データベースと同一になるように行われるもので、データ保護に適した方法です。物理コピーにはデータベース・バックアップ、ミラー化または多重データベース・ファイル、フィジカル・スタンバイ・データベースの 3 タイプがあります。

論理コピーにはソース・データベースと同じ情報が含まれますが、情報はデータベース内に異なる形式で格納される場合があります。データベースの論理コピーを作成することには、多くの利点があります。ただし、論理コピーの作成は、常に物理コピーも作成した上で行う必要があります。物理コピーの代替として作成しないでください。

ロジカル・スタンバイには、次のような利点があります。

- 更新中に論理コピーを開くことができます。このため、論理コピーはほぼリアル・タイムのレポートに役立ちます。
- 論理コピーに様々な物理レイアウトを指定し、目的に応じて最適化できます。たとえば、追加の索引を含めることができ、その結果として論理コピーを利用したレポート・アプリケーションのパフォーマンスが改善されます。
- 論理コピーによって、破損からの保護が強化されます。データは論理的に取得および適用されるため、物理的な破損がデータベースの論理コピーに伝播することはほとんど考えられません。

データベースの論理コピーには 3 タイプあります。

- ロジカル・スタンバイ・データベース
- Streams レプリカ・データベース
- アプリケーションによるコピーのメンテナンス

ロジカル・スタンバイ・データベースは、SQL 適用モードの Oracle Data Guard を使用した場合に最適な状態でメンテナンスされます。この章の後半では、Streams レプリカ・データベースとアプリケーションによるコピーのメンテナンスを説明します。

関連項目：

- データベース・バックアップとミラー化または多重データベース・ファイルの詳細は、『Oracle9*i* バックアップおよびリカバリ概要』を参照してください。
- フィジカル・スタンバイ・データベースとロジカル・スタンバイ・データベースの詳細は、『Oracle9*i* Data Guard 概要および管理』を参照してください。

Streams レプリカ・データベース

SQL 適用モードの Oracle Data Guard と同様に、Oracle Streams を使用して、データベース変更を取得して接続先に伝播し、変更をこれらの接続先に適用できます。Streams は、データのレプリケートのために最適化されます。Streams では、記録中のオンライン REDO ログの変更がローカルに取得され、取得された変更がレプリカ・データベースへ非同期に伝播できます。この最適化により待機時間を減少させ、レプリカのプライマリ・データベースからの遅れを数秒以内に抑えられます。突発的な障害が発生した場合は少量のデータ消失が発生する可能性がありますが、フィジカル・スタンバイ・データベースを併用することで回避できます。

それでもなお、本番データベースの論理コピーの構成およびメンテナンスに Streams の使用を選択する場合があります。Streams を使用すると追加の作業が必要になりますが、固有のビジネス要件を満たすのに必要な優れた柔軟性が提供されます。Streams を使用して構成およびメンテナンスされる論理コピーでは、通常定義のスタンバイ・データベースの有効範囲を超える多くの機能が提供されるため、ロジカル・スタンバイではなくレプリカと呼ばれます。Oracle Streams レプリカの使用が適した要件の一部を次の項に示します。

レプリカ・データベースの更新

レプリカ・データベースとスタンバイ・データベースの最大の違いは、レプリカ・データベースが更新可能であるのに対し、スタンバイ・データベースは更新不可である点です。ジョブ・キューや、レポート・アクティビティをログに記録するレポート・アプリケーションなど、データの更新が必要なアプリケーションをレプリカに対して実行できます。また、レプリカ・データベースでは、ローカル・アプリケーションを Wide Area Network（WAN）の障害から保護し、データベース操作の待機時間を減少させながら、自律的に実行できます。

異機種間プラットフォームのサポート

本番データベースとレプリカ・データベースは、同じプラットフォーム上で稼働している必要はありません。このことにより、コンピュータ資産の使用が柔軟になり、プラットフォーム間の移行が容易になります。

複数キャラクタ・セット

Streams レプリカでは、本番データベースとは異なるキャラクタ・セットを使用できます。データは、適用される前に、1 つのキャラクタ・セットから別のキャラクタ・セットに自動的に変換されます。グローバル操作を使用していて、データを複数の国に配布する必要がある場合に、この機能はきわめて重要です。

オンライン REDO ログのマイニングによる待機時間の最小化

レプリカをほぼリアル・タイムのレポートに使用すると、Streams の本番データベースからの遅れが数秒以内に抑えられ、最新で正確な問合せが提供できます。変更は、アーカイブ後の REDO ログからではなく、記録中のオンライン REDO ログから読取り可能です。

11 以上のデータ・コピー

Streams では、無制限の数のレプリカがサポートされます。その柔軟なルーティング・アーキテクチャではハブ・アンド・スポーク構成が考慮されおり、データを何百ものレプリカに効率的に伝播できます。組織の多くのローカル・オフィスで自律型運用を行う必要がある場合、この機能が重要になることがあります。これに対して、Data Guard によって構成されるスタンバイ・データベースでは LOG_ARCHIVE_DEST_*n* 初期化パラメータを使用して接続先を指定するため、Data Guard を使用するとコピーの数が 10 に制限されます。

高速フェイルオーバー

Streams レプリカは、常に読取り / 書込み操作用にオープンされています。プライマリ・データベースに障害が発生した場合、Streams レプリカですぐに処理を再開できます。少量のデータがプライマリ・データベースに残る場合がありますが、このデータはプライマリ・データベースがリカバリされると自動的に適用されます。データの消失を防ぐよりもリカバリ時間の短縮を重視する場合、この機能が重要になることがあります。プライマリ・データベースが最終的にリカバリされると想定すると、データの消失は一時的なものに過ぎません。

複数接続先への単一の取得

複合環境では、変更の取得は 1 回のみ必要です。これらの変更は複数の接続先に送信できます。この機能により、REDO ログで変更をマイニングするのに必要なリソースをさらに効率的に使用できます。

Streams との共存

ロジカル・スタンバイ・データベースを使用し、さらにプライマリ・データベースで Streams も使用する場合、Streams API を使用してデータベースの論理コピーを作成およびメンテナンスする必要があります。Data Guard の SQL 適用モードと Streams を同じデータベースで使用する機能は、サポートされません。

Streams を使用しない場合

前述のように、一部の高可用性の要件を満たすために Streams の使用を選択する使用例があります。高可用性のルールの 1 つは、単純さの保持です。Oracle Data Guard は高可用性のために設計されており、Streams ベースの高可用性ソリューションよりも実装が容易です。Streams によって提供される柔軟性を利用する場合は、Streams ベースのソリューションを強固にするために必要な専門技術と計画に投資する準備が必要です。つまり、Oracle Data Guard であればデフォルトで提供される自動化および管理ツールのほとんどを、スクリプトを作成して実装する必要があります。

また、Streams は高可用性ではなくデータ統合を主眼に設計されたため、Oracle Data Guard とは異なりゼロ・データ損害モードでの操作は提供されていません。障害発生時にトランザクションを損失すると支障が出る場合は、Streams ではなく Data Guard を使用するか、Streams ベースのソリューションを Data Guard によってメンテナンスされるゼロ・データ損害フィジカル・スタンバイで補う必要があります。Data Guard では、人為エラーから保護する遅延適用オプションも提供されます。フィジカル・スタンバイで補うことによっても、同様の保護が提供可能です。

アプリケーションによるコピーのメンテナンス

データの論理コピーのメンテナンスをアプリケーションで直接実行するように設計すると、可用性を最大限にできます。重要なデータやデータ消失を防ぐためオフサイトに即時移動する必要のあるデータは、アプリケーションで認識されています。また、重要なデータは同期式にレプリケートでき、それほど重要でないデータは非同期式にレプリケートできます。アプリケーションでは、別のデータの論理コピーを管理する他のアプリケーションにデータを同期式または非同期式に送信することによって、データのコピーがメンテナンスされます。同期操作は、分散 SQL またはデータベースのリモート・プロシージャ機能を使用して実行されます。非同期操作は、アドバンスト・キューイングを使用して実行されます。アドバンスト・キューイングは、Oracle Streams のインフラストラクチャの最上部に組み込まれた、データベースに統合されたメッセージ・キューイング機能です。

アプリケーションによるデータ・コピーのメンテナンスでは、最高レベルの可用性が実現可能ですが、最大限の注意が必要です。通常、多くのカスタム開発が必要になります。Data Guard や Streams レプリケーションなどのソリューションによって分析および解決されてきた難しい境界条件の多くを、カスタム・アプリケーションの開発者が再分析し解決する必要があります。また、Data Guard や Streams レプリケーションのような標準ソリューションは、オラクル社と顧客の両方による厳密なテストを通過しています。カスタム開発のソリューションが同じ程度の完成度を示すまでには、大変な労力が必要です。これらの理由から、アプリケーションによるコピーのメンテナンス機能を持つ高可用性ソリューションの構築は、時間と専門技術が十分にある組織のみが行うようにします。

> **関連項目：** アドバンスト・キューイングを使用したアプリケーション開発の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

Streams 高可用性環境の最良の実施例

高可用性環境に Streams を実装するには、発生する可能性のある障害とリカバリでの使用例を考慮し、障害発生後にも Streams で引き続き変更を取得、伝播および適用できるプロシージャを実装する必要があります。次の項目を確認します。

- 高可用性のための Streams の構成
  - データベースとその他の各データベースとの直接接続
  - ハブ・アンド・スポーク構成の作成
  - Streams を使用した Oracle Real Application Clusters の構成
- 障害からのリカバリ
  - フェイルオーバー後のデータベース・リンクの再確立
  - フェイルオーバー後の取得の再起動
  - フェイルオーバー後の伝播の再起動
  - フェイルオーバー後の適用の再起動

ここでは、これらの項目を説明します。

高可用性のための Streams の構成

Streams を使用したソリューションを構成する場合、障害を予想し、アーキテクチャ内に可用性を組み込んで設計することが重要です。分散システム内の各データベースを調べ、そのデータベースに障害が発生した場合のリカバリ・プランを設計する必要があります。データベース障害の影響する対象が、そのデータベースのデータにアクセスするサービスのみの場合もあれば、他のデータベースにも影響して、障害が拡大する場合もあります。

データベースとその他の各データベースとの直接接続

分散システム内の各データベースがその他すべてのデータベースと個々に直接接続した構成では、1 つのデータベースに障害が発生しても他のデータベースでの操作または通信を妨げないため、障害に対して最も耐性があります。すべてのデータがレプリケートされていれば、障害の発生したデータベースを使用していたサービスは、障害の発生していないレプリカに接続できます。

関連項目：

- このような環境の詳細な例は、8-7 ページの「各データベースが共有データのソース・データベースと接続先データベースを兼ねている場合」および第 23 章「複数のソース・レプリケーションの例」を参照してください。
- 3-7 ページ「キューによる転送と適用による転送」

ハブ・アンド・スポーク構成の作成

各データベースがその他すべてのデータベースと個々に直接接続された構成では、最高の高可用性特性が提供されますが、データベースの数が増加すると管理が困難になります。この管理上の問題は、ハブ・アンド・スポーク構成で、多くのデータベースからの変更をハブ・データベースを介してから他のハブ・データベースまたは他のスポーク・データベースに伝播することによって解決されます。新規ソースまたは接続先の追加は、データベースごとに接続を確立するのではなく、ハブ・データベースへの接続を行うのみです。

ただし、ハブは分散環境において非常に重要なノードになります。それに障害が発生すると、ハブを介して行われるすべての通信に障害が発生します。ハブを介して伝播するイベントは非同期な性質を持つため、あるハブから別のハブにストリームをリダイレクトすることは非常に困難です。適切なアプローチとは、ハブを障害に対して耐性がある状態にすることです。

単一データベースを障害に対して耐性がある状態にするために使用した手法は、分散ハブ・データベースにも適用されます。インスタンスおよびノードの障害からの保護には、Oracle Real Application Clusters を使用することをお薦めします。障害とデータ・エラーから保護するためには、この構成を損失のないフィジカル・スタンバイ・データベースと結合する必要があります。障害とデータ・エラーからの保護方法として Streams レプリカのみを使用することは、お薦めしません。

> **関連項目：** このような環境の詳細な例は、8-11 ページの「プライマリ・データベースが複数のセカンダリ・データベースとデータを共有している場合」を参照してください。

Streams を使用した Oracle Real Application Clusters の構成

Oracle Real Application Clusters を Streams と併用する場合、いくつかの重要な考慮事項があります。記録中のオンライン REDO ログからの変更取得は、Oracle Real Application Clusters でサポートされません。変更はアーカイブ REDO ログから取得されます。アーカイブ REDO ログからの取得の場合、本番データベースで変更が行われる時間とそれがレプリカに反映される時間との間に追加の待機時間が発生します。

待機時間の短縮を重視する場合は、システム障害からの保護には、Oracle Real Application Clusters ではなくコールド・フェイルオーバー・クラスタを使用する必要があります。コールド・フェイルオーバー・クラスタは Oracle Real Application Clusters ではありません。かわりに、コールド・フェイルオーバー・クラスタでは、最初のノードに障害が発生すると、2 番目のノードを使用してデータベースがマウントおよびリカバリされます。

取得プロセスは、Oracle Real Application Clusters クラスタ内での実行時に、取得された論理変更レコード（LCR）を受信中のキューを所有しているインスタンス上で実行されます。伝播が有効化されているすべてのインスタンス上でジョブ・キューが実行されている必要があります。あるインスタンスに対して伝播が有効化されていると想定すると、そのインスタンス上で実行中の伝播ジョブにより、そのインスタンスで所有されているキューの LCR が宛先キューに伝播されます。適用プロセスは、適用プロセスによりイベントがデキューされるキューを所有しているインスタンス上で実行されます。取得が実行されるキューと同じ場合と、同じでない場合があります。

Oracle Real Application Clusters を実行中のデータベースへの伝播は、データベース・リンクを介して行われます。データベース・リンクは、イベントを受信予定のキューを所有している宛先インスタンスに接続するように構成する必要があります。

関連項目：

- 2-15 ページ「Streams の取得プロセスと Oracle Real Application Clusters」
- 3-17 ページ「Streams キューと Oracle Real Application Clusters」
- 4-27 ページ「Streams の適用プロセスと Oracle Real Application Clusters」

障害からのリカバリ

ここでは、障害からのリカバリの最良の実施例を説明します。

フェイルオーバー後のデータベース・リンクの再確立

接続先データベースのインスタンスで障害が発生した後も、伝播が引き続き機能することは重要です。障害発生後、伝播ジョブでは接続が再確立されるまでデータベース・リンクが継続的に再試行されます（再試行と次の再試行との間隔は徐々に長くなります）。データベースが同じノード、またはコールド・フェイルオーバー・クラスタ内の別のノードで再起動された場合、データベース・インスタンスが再起動されると接続の再確立が必要になります。状況によってはデータベース・リンクが読取りまたは書込みを待機している場合があり、`TCP_KEEPALIVE_INTERVAL` TCP/IP パラメータによって制御される冗長なタイムアウトが期限切れになるまで、障害は検出されません。このような場合は、データベース・リンクを一度削除してから再作成し、通信が迅速に再確立されるようにする必要があります。

Oracle Real Application Clusters クラスタ内の 1 つのインスタンスに障害が発生した場合、そのインスタンスはクラスタ内の別のノードによってリカバリされます。障害の発生したインスタンスで以前所有されていた各キューは、新規インスタンスに割り当てられます。インバウンド・データベース・リンクは削除して、宛先キューを所有している新規インスタンスを指すように再確立する必要があります。高可用性環境では、必要なデータベース・リンクすべてを削除および再作成するスクリプトを準備することが考えられます。障害の発生したインスタンスがリカバリした後、これらのスクリプトを実行し、Streams で伝播を再開できます。

関連項目： Streams 環境内でデータベース・リンクを作成する方法は、11-13 ページの「ネットワーク接続性とデータベース・リンクの構成」を参照してください。

フェイルオーバー後の取得の再起動

障害発生後に単一ノードのデータベースが再起動されるか、または障害発生後にコールド・フェイルオーバー・クラスタ内の別のノード上にあるデータベースが再起動されると、取得プロセスは障害発生時の状態に自動的に戻ります。つまり、障害発生時に取得プロセスが実行中であった場合は、再起動の必要はありません。

Oracle Real Application Clusters 環境で実行している取得プロセスについては、取得プロセスを実行しているインスタンスに障害が発生した場合、取得された LCR を受信するキューはクラスタ内の別のノードに割り当てられます。DBA_QUEUE_TABLES データ・ディクショナリ・ビューを問い合せ、取得プロセスで使用されていたキューが現在所有されているインスタンスを判別し、そのノードで取得プロセスを再起動する必要があります。障害発生インスタンスが障害後にオンラインに戻った場合、そのインスタンスが障害時に取得プロセスを実行していたとしても、もはや取得プロセスで使用されていたキューの所有者ではないため、取得プロセスは再起動されません。

関連項目：

- 2-15 ページ「Streams の取得プロセスと Oracle Real Application Clusters」
- 12-4 ページ「取得プロセスの起動」

フェイルオーバー後の伝播の再起動

イベントをソース・キューから宛先キューに伝播させるには、ソース・キューを所有しているインスタンス上で伝播ジョブを実行する必要があります。単一ノードのデータベースまたはコールド・フェイルオーバー・クラスタでは、単一データベース・インスタンスが再起動されると伝播が再開されます。

Real Application Clusters 環境で実行するときは、伝播ジョブが正しいインスタンス上で有効化されていることを確認する必要があります。伝播ジョブは、伝播ジョブによりイベントが宛先キューへ送信される送信元のソース・キューを所有するインスタンス上で、実行される必要があります。ジョブを特定のインスタンス上で強制実行させる伝播ジョブに、インスタンス・アフィニティ・パラメータを指定できます。インスタンスに障害が発生し、キューの所有権が別のインスタンスに移行される場合、伝播ジョブのアフィニティをこの別のインスタンスに再設定する必要があります。また、インスタンス上で実行されるジョブについて、そのインスタンスの動的初期化パラメータ JOB_QUEUE_PROCESSES がゼロより大きいことが必要です。

関連項目： 3-17 ページ「Streams キューと Oracle Real Application Clusters」

フェイルオーバー後の適用の再起動

障害発生後に、単一ノードのデータベースが再起動されるか、またはコールド・フェイルオーバー・クラスタ内の別のノード上にあるデータベースが再起動されると、適用プロセスは障害発生時の状態に自動的に戻ります。つまり、障害発生時に適用プロセスが実行中であった場合は、適用プロセスを再起動する必要はありません。

Oracle Real Application Clusters クラスタ内では、適用プロセスをホスティングするインスタンスに障害が発生した場合、適用プロセスによりイベントがデキューされるキューは、クラスタ内の別のノードに割り当てられます。DBA_QUEUE_TABLES データ・ディクショナリ・ビューを問い合せ、適用プロセスで使用していたキューが現在所有されているインスタンスを判別し、そのノードで適用プロセスを再起動する必要があります。障害の発生したインスタンスがその後オンラインに戻っても、もはや適用プロセスで使用していたキューの所有者ではないため、障害時に適用プロセスを実行していたとしても適用プロセスは再起動されません。

関連項目：

- 4-27 ページ「Streams の適用プロセスと Oracle Real Application Clusters」
- 14-7 ページ「適用プロセスの起動」

第 II 部

Streams の管理

第 II 部では、構成、管理、監視およびトランザクションの手順など、Streams 環境の管理について説明します。第 II 部の構成は、次のとおりです。

- 第 11 章「Streams 環境の構成」
- 第 12 章「取得プロセスの管理」
- 第 13 章「ステージングと伝播の管理」
- 第 14 章「適用プロセスの管理」
- 第 15 章「ルールおよびルールベースの変換の管理」
- 第 16 章「その他の Streams 管理タスク」
- 第 17 章「Streams 環境の監視」
- 第 18 章「Streams 環境のトラブルシューティング」

11

Streams 環境の構成

この章では、Streams を使用するためにデータベースまたは分散データベース環境を準備する手順と、Streams 環境を構成する手順について説明します。

この章の内容は次のとおりです。

Streams 管理者の構成

Streams 環境を管理するには、適切な権限を持つ新規ユーザーを作成するか、これらの権限を既存のユーザーに付与します。SYS または SYSTEM ユーザーを Streams 管理者として使用しないでください。また、Streams 管理者のデフォルト表領域として SYSTEM 表領域を使用しないでください。

次の手順に従って、Streams を使用する環境内の各データベースで Streams 管理者を構成します。

1. ユーザーの作成、権限の付与、表領域の作成およびユーザーの変更を行うことができる管理ユーザーとして接続します。
2. Streams 管理者の役割を果たす新規ユーザーを作成するか、既存のユーザーを使用します。たとえば、新規ユーザー strmadmin を作成するには、次の文を実行します。

```
CREATE USER strmadmin IDENTIFIED BY strmadminpw;
```

> **注意：** セキュリティを確保するために、Streams 管理者には strmadminpw 以外のパスワードを使用してください。

3. Streams 管理者に少なくとも次の権限を付与します。

```
GRANT CONNECT, RESOURCE TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM TO strmadmin;

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/
```

4. 必要な場合は、Streams 管理者に次の権限を付与します。

   - Streams 管理者がデータベース上で 1 つ以上の適用プロセスを管理する場合は、DBMS_APPLY_ADM パッケージの EXECUTE 権限。また、Streams 管理者には、DBMS_APPLY_ADM パッケージのサブプログラムを使用して構成された適用ハンドラとエラー・ハンドラに対する EXECUTE 権限も必要です。
   - Streams 管理者がデータベース上で 1 つ以上の取得プロセスを管理する場合は、DBMS_CAPTURE_ADM パッケージの EXECUTE 権限。
   - Streams 管理者がデータベース上で 1 つ以上の伝播を管理する場合は、DBMS_PROPAGATION_ADM パッケージの EXECUTE 権限。
   - Streams 管理者がデータベースの現行の SCN を取得する必要がある場合は、DBMS_FLASHBACK パッケージの EXECUTE 権限。通常、Streams 管理者は現行の SCN を判断し、DBMS_APPLY_ADM パッケージの SET_TABLE_INSTANTIATION_SCN、SET_SCHEMA_INSTANTIATION_SCN または SET_GLOBAL_INSTANTIATION_SCN プロシージャを使用して、インスタンス化 SCN を設定する必要があります。
   - Streams 管理者が環境を容易に監視できるようにする必要がある場合は、SELECT_CATALOG_ROLE。
   - Oracle Enterprise Manager の Streams ツールを使用する予定の場合は、SELECT ANY DICTIONARY 権限。
   - Streams 管理者が PL/SQL サブプログラム内で DBA_APPLY_ERROR データ・ディクショナリ・ビューから選択できるようにする必要がある場合は、このビューの SELECT 権限。この種の PL/SQL サブプログラムの例については、17-35 ページの「適用エラーの詳細情報の表示」を参照してください。
   - 適用プロセスの適用ユーザーを指定しない場合は、他のユーザーが所有する適用オブジェクトに対して DML 変更と DDL 変更を実行するために必要な権限。適用ユーザーを指定する場合、適用ユーザーにはこれらの権限が必要です。
   - 適用プロセスの適用ユーザーを指定しない場合は、Streams の適用プロセスで実行される他のユーザー所有の PL/SQL プロシージャの EXECUTE 権限。これらのプロシージャは、適用ハンドラまたはエラー・ハンドラで使用できます。適用ユーザーを指定する場合、適用ユーザーにはこれらの権限が必要です。
   - Streams の取得プロセス、伝播または適用プロセスで使用されるルールについてルールベースの変換で指定されている、他のユーザー所有の PL/SQL ファンクションの EXECUTE 権限。適用プロセスの適用ユーザーを指定する場合、その適用ユーザーにはこれらの権限が必要です。
   - Streams 管理者が Streams の取得プロセス、伝播または適用プロセスで使用されるキューを所有しておらず、キューの作成時にそのキューのキュー・ユーザーとして指定されていない場合、Streams 管理者がキューのイベントをエンキューまたはデキューできるようにするには、Streams 管理者をキューの保護キュー・ユーザーとして構成する必要があります。また、Streams 管理者には、キューの ENQUEUE 権

限または DEQUEUE 権限、あるいはその両方が必要になる場合があります。手順については、13-3 ページの「保護キューでのユーザー操作の有効化」を参照してください。

5. Streams 管理者用の表領域を作成するか、既存の表領域を使用します。たとえば、次の文では、Streams 管理者用の新規表領域が作成されます。

```
CREATE TABLESPACE streams_tbs DATAFILE '/usr/oracle/dbs/streams_tbs.dbf'
  SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

6. Streams 管理者用の表領域を指定します。

```
ALTER USER strmadmin DEFAULT TABLESPACE streams_tbs
                     QUOTA UNLIMITED ON streams_tbs;
```

7. 前述のすべての手順を、Streams を使用する環境内のデータベースごとに繰り返します。

Streams に関連する初期化パラメータの設定

表 11-1 に、Streams 環境の操作、信頼性およびパフォーマンスに重要な初期化パラメータを示します。これらのパラメータを、Streams 環境に合せて適切に設定してください。

関連項目： これらの初期化パラメータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

表 11-1 Streams に関連する初期化パラメータ

パラメータ	値	説明
AQ_TM_PROCESSES	**デフォルト：**0 **範囲：**0 ～ 10	キュー・モニター・プロセスを設定します。このパラメータを 1 以上に設定すると、指定した数のキュー・モニター・プロセスが起動します。これらのキュー・モニター・プロセスは、遅延や期限切れなど、時間ベースのメッセージ操作の管理、指定した保存期間を過ぎて保存されているメッセージのクリーン・アップ、および保存期間が 0（ゼロ）の場合のコンシューム済みメッセージのクリーン・アップを受け持ちます。 ユーザー・イベントを Streams キューにエンキューする必要がある場合は、このパラメータを 1 以上に設定します。ユーザー・イベントとは、Streams の取得プロセスではなくユーザーやアプリケーションによって作成されるイベントです。

表 11-1 Streams に関連する初期化パラメータ（続き）

パラメータ	値	説明
ARCHIVE_LAG_TARGET	**デフォルト：**0 **範囲：**0 または [60, 7200] の任意の整数	消失する可能性のあるデータ量を制限し、ユーザー指定の期間が経過した後にログ・スイッチを強制実行することで、実際上のスタンバイ・データベースの可用性を高めます。 Streams を Real Application Clusters 環境で使用している場合は、このパラメータを 0（ゼロ）より大きい値に設定してログ・ファイルを自動的に切り替えてください。 **関連項目：** 2-15 ページ「Streams の取得プロセスと Oracle Real Application Clusters」
COMPATIBLE	**デフォルト：**8.1.0 **範囲：**8.1.0 ～現行リリース番号	このパラメータでは、Oracle サーバーが互換性を維持する必要のあるリリースを指定します。互換レベルの異なる Oracle サーバーを相互運用できます。 Streams を使用するには、このパラメータを 9.2.0 以上に設定する必要があります。
GLOBAL_NAMES	**デフォルト：**false **範囲：**true または false	データベース・リンクに、接続先データベースと同じ名前を使用する必要があるかどうかを指定します。 Streams を使用してデータベース間で情報を共有する必要がある場合は、Streams 環境の各データベースでこのパラメータを true に設定します。
JOB_QUEUE_PROCESSES	**デフォルト：**0 **範囲：**0 ～ 1000	各インスタンス（J000 ... J999）用の *Jn* ジョブ・キュー・プロセスの数を指定します。ジョブ・キュー・プロセスでは、DBMS_JOB によって作成された要求が処理されます。 ALTER SYSTEM 文を使用すると、JOB_QUEUE_PROCESSES の設定を動的に変更できます。 このパラメータは、Streams 環境でイベントを伝播する各データベースで 2 以上に設定し、同時に実行できる最大ジョブ数に 2 を加えた値に設定する必要があります。

表 11-1 Streams に関連する初期化パラメータ（続き）

パラメータ	値	説明
LOG_PARALLELISM	**デフォルト :** 1 **範囲 :** 1 ～ 255	Oracle 内での REDO 割当ての並行性レベルを指定します。 データベース上で 1 つ以上の取得プロセスを実行する予定の場合は、このパラメータを 1 に設定する必要があります。 このパラメータを 1 に設定しても、取得の並列性には影響しません。取得プロセスの並列性は、DBMS_CAPTURE_ADM パッケージの SET_PARAMETER プロシージャを使用して設定できます。
LOGMNR_MAX_PERSISTENT_SESSIONS	**デフォルト :** 1 **範囲 :** 1 ～ LICENSE_MAX_SESSIONS	すべてのセッションでインスタンスによって生成された REDO ログをマイニングする場合に、同時にアクティブになっている LogMiner 永続マイニング・セッションの最大数を指定します。 1 つのデータベース上で複数の Streams 取得プロセスを実行する予定の場合は、このパラメータをその取得プロセス数以上の値に設定します。
OPEN_LINKS	**デフォルト :** 4 **範囲 :** 0 ～ 255	1 つのセッション中のリモート・データベースへの同時オープン接続の最大数を指定します。これらの接続には、データベース・リンク、外部プロシージャおよび外部カートリッジが含まれ、それぞれが別個のプロセスを使用します。 Streams 環境では、このパラメータがデフォルト値の 4 以上に設定されていることを確認してください。
PARALLEL_MAX_SERVERS	**デフォルト :** 次のパラメータの値から導出 : CPU_COUNT PARALLEL_ADAPTIVE_MULTI_USER PARALLEL_AUTOMATIC_TUNING **範囲 :** 0 ～ 3599	1 インスタンスのパラレル実行プロセスとパラレル・リカバリ・プロセスの最大数を指定します。Oracle では、需要が増加するにつれて、プロセス数がインスタンス起動時に作成された数からこの値まで増やされます。 Streams 環境では、各取得プロセスと適用プロセスが複数のパラレル実行サーバーを使用できます。十分な数のパラレル実行サーバーを使用できるように、この初期化パラメータを適切な値に設定してください。

表 11-1 Streams に関連する初期化パラメータ（続き）

パラメータ	値	説明
PROCESSES	**デフォルト :** PARALLEL_MAX_SERVERS から導出 **範囲 :** 6 ～オペレーティング・システム依存の上限	Oracle に同時に接続できるオペレーティング・システムのユーザー・プロセスの最大数を指定します。 このパラメータの値が、ロック、ジョブ・キュー・プロセスおよびパラレル実行プロセスなど、すべてのバックグラウンド・プロセスを見込んでいることを確認してください。Streams では、取得プロセスと適用プロセスはバックグラウンド・プロセスとパラレル実行プロセスを使用し、伝播ジョブはジョブ・キュー・プロセスを使用します。
SESSIONS	**デフォルト :** (1.1 * PROCESSES) + 5 から導出 **範囲 :** 1 ～ 2^{31}	システム内に作成できるセッションの最大数を指定します。 1 つのデータベース上で 1 つ以上の取得プロセスまたは適用プロセスを実行する予定の場合は、このパラメータの値の増加が必要になることがあります。データベースの各バックグラウンド・プロセスにはそれぞれセッションが必要です。
SGA_MAX_SIZE	**デフォルト :** 起動時の SGA の初期サイズ **範囲 :** 0 ～オペレーティング・システム依存の上限	データベース・インスタンスの存続期間中における SGA の最大サイズを指定します。 1 つのデータベース上で複数の取得プロセスを実行する予定の場合は、このパラメータの値の増加が必要になることがあります。
SHARED_POOL_SIZE	**デフォルト :** 32 ビット・プラットフォームの場合 : 8MB、最近似のグラニュル・サイズに切上げ 64 ビット・プラットフォームの場合 : 64MB、最近似のグラニュル・サイズに切上げ **範囲 :** 最小値 : グラニュル・サイズ 最大値 : オペレーティング・システム依存	共有プールのサイズ（バイト単位）を指定します。共有プールには、共有カーソル、ストアド・プロシージャ、制御構造およびその他の構造が格納されます。 データベース上の取得プロセスごとに、共有プールのサイズを 10MB ずつ増やす必要があります。

表 11-1 Streams に関連する初期化パラメータ（続き）

パラメータ	値	説明
TIMED_STATISTICS	**デフォルト：** STATISTICS_LEVEL の設定が TYPICAL または ALL の場合は true STATISTICS_LEVEL の設定が BASIC の場合は false STATISTICS_LEVEL のデフォルトは TYPICAL **範囲：** true または false	時間に関連する統計を収集するかどうかを指定します。 Streams に関連するデータ・ディクショナリ・ビューの経過時間の統計を収集する場合は、true に設定します。経過時間の統計を含むビューには V$STREAMS_CAPTURE、V$STREAMS_APPLY_COORDINATOR、V$STREAMS_APPLY_READER および V$STREAMS_APPLY_SERVER が含まれます。

Streams に関連するエクスポート・ユーティリティとインポート・ユーティリティのパラメータ設定

この項では、Streams に関連するエクスポート・ユーティリティとインポート・ユーティリティのパラメータについて説明します。

関連項目：

- 2-12 ページ「インスタンス化」
- 16-32 ページ「Streams を使用したデータベースにおける全データベースのエクスポート / インポートの実行」
- エクスポートとインポートの実行方法は、『Oracle9*i* データベース・ユーティリティ』を参照してください。

Streams に関連するエクスポート・ユーティリティのパラメータ

次のエクスポート・ユーティリティのパラメータが Streams に関連しています。

OBJECT_CONSISTENT エクスポート・ユーティリティ・パラメータと Streams

OBJECT_CONSISTENT エクスポート・ユーティリティ・パラメータでは、オブジェクトごとにエクスポートされたデータおよびプロシージャ型アクションが特定の時点で一貫性を持つように、SET TRANSACTION READ ONLY 文を繰り返し使用するかどうかを指定します。OBJECT_CONSISTENT を y に設定すると、各オブジェクトはパーティション化されている場合にも、独自の読取り専用トランザクションでエクスポートされます。これに対して、CONSISTENT エクスポート・ユーティリティ・パラメータを使用すると、読取り専用トランザクションは 1 つのみになります。

Streams 環境でインスタンス化を実行する場合は、エクスポート・ダンプ・ファイルにある程度の一貫性が必要です。OBJECT_CONSISTENT エクスポート・ユーティリティ・パラメータを使用すれば、Streams のインスタンス化についてこの一貫性を確保できます。Streams のインスタンス化のみでなく、他の用途にもエクスポート・ダンプ・ファイルを使用しており、その用途が OBJECT_CONSISTENT で得られるよりも厳密な一貫性要件を伴う場合は、Streams のインスタンス化にエクスポート・ユーティリティ・パラメータ CONSISTENT、FLASHBACK_SCN または FLASHBACK_TIME を使用できます。

デフォルトでは、OBJECT_CONSISTENT エクスポート・ユーティリティ・パラメータは n に設定されます。Streams のインスタンス化の一部としてエクスポートを実行し、Streams のインスタンス化で使用する以上に厳密なエクスポート・ユーティリティ・パラメータを必要としない場合は、y を指定します。

注意：

- Streams をインスタンス化するためのエクスポート中は、エクスポート対象のオブジェクトに対して DDL 変更が行われないように注意してください。
- 非 NULL アクション・コンテキストを持つルールを含むデータベースまたはスキーマをエクスポートする場合は、そのルールを所有するスキーマのデータベースまたはデフォルト表領域が書込み可能である必要があります。データベースまたは表領域が読取り専用であると、エクスポート・エラーが発生します。

Streams に関連するインポート・ユーティリティのパラメータ

次のインポート・ユーティリティのパラメータが Streams に関連しています。

STREAMS_INSTANTIATION インポート・ユーティリティ・パラメータと Streams

STREAMS_INSTANTIATION インポート・ユーティリティ・パラメータでは、エクスポート・ダンプ・ファイルに表示される場合がある Streams のインスタンス化メタデータをインポートするかどうかを指定します。このパラメータを y に設定すると、インポートによって行われる変更の循環を回避するために、インポート・セッションでは Streams タグが '00' と等価の 16 進値に設定されます。インポートによって生成される REDO エントリには、このタグ値が含まれます。デフォルトでは、STREAMS_INSTANTIATION インポート・ユーティリティ・パラメータは n に設定されます。Streams のインスタンス化の一部としてインポートを実行する場合は、y を指定します。

関連項目： 第 8 章「Streams のタグ」

STREAMS_CONFIGURATION インポート・ユーティリティ・パラメータと Streams

STREAMS_CONFIGURATION インポート・ユーティリティ・パラメータでは、エクスポート・ダンプ・ファイルに表示される場合がある一般的な Streams メタデータをインポートするかどうかを指定します。このインポート・パラメータは、全データベースのインポートを実行する場合にのみ関連します。デフォルトでは、STREAMS_CONFIGURATION インポート・ユーティリティ・パラメータは y に設定されます。通常、インポートがバックアップまたはリストア操作の一部である場合は、y を指定します。

次のオブジェクトは、STREAMS_CONFIGURATION の設定に関係なくインポートされます。

- Streams キューおよびキュー表（STREAMS_CONFIGURATION を n に設定すると、これらのキューがインポート中に開始されることはありません）。
- キュー・サブスクライバ。
- アドバンスト・キューイング・エージェント。
- Streams の伝播に関連するジョブ・キュー・プロセス。
- ルール・セットおよび評価コンテキストを含めたルール Streams ルールおよび Streams 以外のルールを含め、すべてのルールがインポートされます。Streams ルールとは、DBMS_STREAMS_ADM パッケージの特定のプロシージャが実行されるときにシステムによって生成されるルールで、Streams 以外のルールとは、DBMS_RULE_ADM パッケージを使用して作成されるルールです。

  STREAMS_CONFIGURATION パラメータの設定が n の場合、Streams ルールに関する情報は、データ・ディクショナリ・ビュー ALL_STREAMS_GLOBAL_RULES、ALL_STREAMS_SCHEMA_RULES、ALL_STREAMS_TABLE_RULES、DBA_STREAMS_GLOBAL_RULES、DBA_STREAMS_SCHEMA_RULES および DBA_STREAMS_TABLE_RULES にはインポートされません。ただし、これらのルールに関する情報は、STREAMS_CONFIGURATION パラメータの設定に関係なく、データ・ディクショナリ・ビュー ALL_RULES、ALL_RULE_SETS、ALL_RULE_SET_RULES、DBA_RULES、DBA_RULE_SETS、DBA_RULE_SET_RULES、USER_RULES、USER_RULE_SETS および USER_RULE_SET_RULES にインポートされます。

STREAMS_CONFIGURATION インポート・ユーティリティ・パラメータの設定が y の場合、次の情報がインポートに含まれます。STREAMS_CONFIGURATION インポート・ユーティリティ・パラメータの設定が n の場合は、次の情報はインポートに含まれません。

- 取得プロセス。各取得プロセスには次の情報が含まれます。
  - 取得プロセスの名前
  - 取得プロセスの状態
  - 取得プロセスのパラメータの設定

- 取得プロセスで使用されるキューのキュー所有者およびキュー名
- 取得プロセスで使用されるルール・セットのルール・セット所有者およびルール・セット名

- 表がエクスポート・データベースでインスタンス化の準備を完了している場合、これらの表がインポート・データベースでインスタンス化用に準備されます。
- スキーマがエクスポート・データベースでインスタンス化の準備を完了している場合、これらのスキーマがインポート・データベースでインスタンス化用に準備されます。
- エクスポート・データベースがインスタンス化の準備を完了している場合、インポート・データベースがインスタンス化用に準備されます。
- Streams キューの状態。開始または停止。(Streams キュー自体はパラメータの設定に関係なくインポートされます。)
- 伝播。各伝播には次の情報が含まれます。
  - 伝播の名前
  - ソース・キューのキュー所有者およびキュー名
  - 宛先キューのキュー所有者およびキュー名
  - 接続先データベース・リンク
  - 伝播で使用されるルール・セットのルール・セット所有者およびルール・セット名
- 適用プロセス。各適用プロセスには次の情報が含まれます。
  - 適用プロセスの名前
  - 適用プロセスの状態
  - 適用プロセスのパラメータの設定
  - 適用プロセスで使用されるキューのキュー所有者およびキュー名
  - 適用プロセスで使用されるルール・セットのルール・セット所有者およびルール・セット名
  - 取得したイベントまたはユーザー・エンキュー・イベントが適用プロセスにより適用されるかどうか
  - 存在する場合は、適用プロセスの適用ユーザー
  - 存在する場合は、適用プロセスで使用されるメッセージ・ハンドラ
  - 存在する場合は、適用プロセスで使用される DDL ハンドラ
  - 適用プロセスによって行われた変更用に REDO ログ内で生成されたタグ
  - 存在する場合は、適用データベース・リンク
  - 適用プロセスのソース・データベース

  - 適用されたメッセージ番号、最も古いメッセージ番号、適用時間および適用されたメッセージの作成時間を含む、DBA_APPLY_PROGRESS データ・ディクショナリ・ビュー内の適用の進捗に関する情報
  - 適用エラー
- DML ハンドラ。
- エラー・ハンドラ。
- 更新の競合ハンドラ。
- 適用表の代替キー列。
- 各適用オブジェクトのインスタンス化 SCN。
- 各適用オブジェクトの非処理 SCN。
- Streams ルールに関する一部のデータ・ディクショナリ情報。ルール自体は、STREAMS_CONFIGURATION パラメータの設定に関係なくインポートされます。

Streams の取得プロセスを実行するためのデータベースの構成

ここでは、Streams の取得プロセスを実行するためのデータベース要件について説明します。

- ARCHIVELOG モードで実行するためのデータベースの構成
- LogMiner 用の代替表領域の指定

これらのタスクのみでなく、取得プロセスを実行するデータベース上で初期化パラメータが適切に設定されていることを確認してください。

関連項目： 11-4 ページ「Streams に関連する初期化パラメータの設定」

ARCHIVELOG モードで実行するためのデータベースの構成

取得プロセスによって変更が取得されるデータベースは、ARCHIVELOG モードで実行している必要があります。

関連項目：

- 2-22 ページ「ARCHIVELOG モードと取得プロセス」
- ARCHIVELOG モードでデータベースを実行する方法については、『Oracle9*i* データベース管理者ガイド』を参照してください。

LogMiner 用の代替表領域の指定

デフォルトでは、LogMiner の表は SYSTEM 表領域にありますが、変更を取得するために取得プロセスが起動すると、SYSTEM 表領域にこれらの表に使用する領域が足りなくなる場合があります。したがって、LogMiner の表に使用する代替表領域を作成する必要があります。

次の例では、LogMiner 用に表領域 logmnrts を作成します。

1. 表領域を作成するための権限と、DBMS_LOGMNR_D パッケージ内のサブプログラムを実行するための権限を持つ、管理ユーザーとして接続します。
2. LogMiner の表に使用する代替表領域を作成するか、既存の表領域を使用します。たとえば、次の文では、LogMiner の表に使用する代替表領域が作成されます。

```
CREATE TABLESPACE logmnrts DATAFILE '/usr/oracle/dbs/logmnrts.dbf'
  SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

3. DBMS_LOGMNR_D パッケージの SET_TABLESPACE プロシージャを実行して、LogMiner 用の代替表領域を設定します。たとえば、表領域 logmnrts を指定するには、次のプロシージャを実行します。

```
EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('logmnrts');
```

ネットワーク接続性とデータベース・リンクの構成

Streams を使用してデータベース間で情報を共有する予定の場合は、これらのデータベース間のネットワーク接続性とデータベース・リンクを構成します。

- Oracle データベースの場合は、データベースが相互に通信できるようにネットワークと Oracle Net を構成します。

  関連項目：『Oracle9*i* Net Services 管理者ガイド』

- Oracle 以外のデータベースの場合は、Oracle ゲートウェイを Oracle データベースと Oracle 以外のデータベースとの通信用に構成します。

  関連項目：『Oracle9*i* Heterogeneous Connectivity Administrator's Guide』

- あるデータベースのソース・キューから別のデータベースの宛先キューにイベントを伝播させる予定の場合は、ソース・キューを含むデータベースと宛先キューを含むデータベースの間に、プライベート・データベース・リンクを作成します。各データベース・リンクでは、データベース間でユーザーがイベントを伝播できるように CONNECT TO 句を使用する必要があります。

たとえば、Streams 管理者 strmadmin として接続するデータベース dbs2.net へのデータベース・リンクを作成するには、次の文を実行します。

```
CREATE DATABASE LINK dbs2.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'dbs2.net';
```

関連項目： データベース・リンクの作成方法の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

取得ベースの Streams 環境の構成

この項では、次のタスクを実行するための一般的な手順について説明します。

- 新規の単一ソース Streams 環境の作成
- 既存の単一ソース環境への共有オブジェクトの追加
- 既存の単一ソース環境への新規接続先データベースの追加
- 新規の複数ソース環境の作成
- 既存の複数ソース環境への共有オブジェクトの追加
- 既存の複数ソース環境への新規データベースの追加

注意： ここで説明する手順は、DBMS_STREAMS_ADM パッケージを使用して Streams 環境を構成する場合を想定しています。他のパッケージを使用する場合は、タスクごとに追加の手順が必要になることがあります。

関連項目： 取得ベースの Streams 環境を構成する詳細な例は、次の章を参照してください。

- 第 20 章「単一データベースの取得および適用の例」
- 第 21 章「簡単な単一のソース・レプリケーションの例」
- 第 22 章「単一ソースの異機種間レプリケーションの例」
- 第 23 章「複数のソース・レプリケーションの例」

新規の単一ソース Streams 環境の作成

この項では、新規の単一ソース Streams 環境を作成する場合の一般的な手順について説明します。単一ソース環境とは、共有データのソース・データベースが 1 つのみの環境です。単一ソース環境に複数のソース・データベースが存在する場合もありますが、この場合、2 つのソース・データベースが同じデータを取得することはありません。

新規の Streams 環境で取得プロセスを起動して伝播を構成する前に、イベントを受信する伝播または適用プロセスが、これらのイベントを処理するように構成されていることを確認してください。つまり、伝播または適用プロセスが存在し、それぞれがイベントを適切に処理するルール・セットに関連付けられている必要があります。これらの伝播と適用プロセスがこれらのイベントを処理するように適切に構成されていないと、イベントが失われる可能性があります。

通常、あるデータベースで共有オブジェクトに対する変更が取得されてから、リモート・データベースに伝播され、そこで適用される新規 Streams 環境を構成する場合は、環境を次の順序で構成する必要があります。

1. 前述した必要なタスクを完了し、Streams 用の環境内で各データベースを準備します。
   - 11-2 ページ「Streams 管理者の構成」
   - 11-4 ページ「Streams に関連する初期化パラメータの設定」
   - 11-12 ページ「Streams の取得プロセスを実行するためのデータベースの構成」
   - 11-13 ページ「ネットワーク接続性とデータベース・リンクの構成」

   これらのタスクの一部は、一部のデータベースでは不要な場合があります。
2. 存在しない場合は、必要な Streams キューを作成します。取得プロセスまたは適用プロセスを作成するときに、そのプロセスを特定の Streams キューに関連付けます。伝播を作成するときには、その伝播を特定のソース・キューおよび宛先キューに関連付けます。手順については、13-2 ページの「Streams のキューの作成」を参照してください。
3. 任意の共有オブジェクトについて、各ソース・データベースでサプリメンタル・ロギングを指定します。手順については、12-8 ページの「ソース・データベースでのサプリメンタル・ロギングの指定」を参照してください。
4. 各データベースで、環境に必要な取得プロセス、伝播および適用プロセスを作成します。これらは任意の順序で作成できます。
   - 変更を取得する各データベースで、1 つ以上の取得プロセスを作成します。各取得プロセスで、変更の取得に適切なルール・セットが使用されることを確認します。作成した取得プロセスは起動しないでください。手順については、12-2 ページの「取得プロセスの作成」を参照してください。

     DBMS_STREAMS_ADM パッケージを使用して取得ルールを追加すると自動的に、指定した表、指定したスキーマまたはデータベース全体に対して、それぞれ DBMS_CAPTURE_ADM パッケージの PREPARE_TABLE_INSTANTIATION、

PREPARE_SCHEMA_INSTANTIATION または PREPARE_GLOBAL_INSTANTIATION プロシージャが実行されます。

次のいずれかの条件に該当する場合は、適切なプロシージャを実行してインスタンス化の準備を手動で行う必要があります。

- DBMS_RULE_ADM パッケージを使用して取得ルールを追加または変更する場合。
- 既存の取得プロセスを使用し、共有オブジェクトの取得ルールは追加しない場合。

インスタンス化の準備を手動で行う必要がある場合の手順については、12-10 ページの「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」を参照してください。

- 取得イベントをソース・キューから宛先キューに伝播する伝播をすべて作成します。各伝播で、変更の伝播に適切なルール・セットが使用されることを確認します。手順については、13-7 ページの「伝播の作成」を参照してください。
- 変更を適用する各データベースで、1 つ以上の適用プロセスを作成します。各適用プロセスで、変更の適用に適切なルール・セットが使用されることを確認します。作成した適用プロセスは起動しないでください。手順については、14-2 ページの「適用プロセスの作成」を参照してください。

5. 適用プロセスによって変更が適用される各データベース・オブジェクトをインスタンス化するか、そのインスタンス化 SCN を設定します。接続先データベースにデータベース・オブジェクトが存在しない場合は、エクスポート / インポートを使用してインスタンス化します。接続先データベースにデータベース・オブジェクトが存在する場合は、そのインスタンス化 SCN を手動で設定します。

   - エクスポート / インポートを使用してデータベース・オブジェクトをインスタンス化するには、最初に OBJECT_CONSISTENT エクスポート・パラメータを y に設定してソース・データベースでエクスポートするか、より厳密な一貫性レベルを使用します。次に、接続先データベースで、STREAMS_INSTANTIATION インポート・パラメータを y に設定してインポートします。詳細は、14-34 ページの「エクスポート / インポートを使用したインスタンス化 SCN の設定」を参照してください。
   - 表、スキーマまたはデータベースのインスタンス化 SCN を手動で設定するには、接続先データベースで DBMS_APPLY_ADM パッケージの適切なプロシージャを 1 つ以上実行します。
     - SET_TABLE_INSTANTIATION_SCN
     - SET_SCHEMA_INSTANTIATION_SCN
     - SET_GLOBAL_INSTANTIATION_SCN

     これらのプロシージャのいずれかを実行する場合は、接続先データベースの共有オブジェクトに、インスタンス化 SCN の時点でソース・データベースとの一貫性があることを確認する必要があります。

接続先データベースで SET_GLOBAL_INSTANTIATION_SCN を実行する場合は、適用する DDL 変更を含むソース・データベース内の既存の各スキーマに対して SET_SCHEMA_INSTANTIATION_SCN も実行し、適用する DML 変更または DDL 変更を含むソース・データベース内の既存の各表に対して SET_TABLE_INSTANTIATION_SCN を実行する必要があります。

接続先データベースで SET_SCHEMA_INSTANTIATION_SCN を実行する場合は、適用する DML 変更または DDL 変更を含むスキーマ内の既存の各ソース・データベース表に対して、SET_TABLE_INSTANTIATION_SCN も実行する必要があります。

手順については、14-35 ページの「DBMS_APPLY_ADM パッケージを使用したインスタンス化 SCN の設定」を参照してください。

または、メタデータのエクスポート / インポートを実行して、既存のデータベース・オブジェクトについてインスタンス化 SCN を設定できます。このオプションを選択した場合は、行がインポートされないことを確認してください。また、すべての接続先データベースの共有オブジェクトに、エクスポートを実行したソース・データベースとのエクスポート時点での一貫性があることを確認してください。DML 変更のみを共有する場合は、表レベルのエクスポート / インポートで十分です。DDL 変更も共有する場合は、追加の考慮事項があります。メタデータのエクスポート / インポートを実行する方法の詳細は、14-34 ページの「エクスポート / インポートを使用したインスタンス化 SCN の設定」を参照してください。

6. 手順 4 で作成した各適用プロセスを起動します。手順については、14-7 ページの「適用プロセスの起動」を参照してください。
7. 手順 4 で作成した各取得プロセスを起動します。手順については、12-4 ページの「取得プロセスの起動」を参照してください。

環境を構成するときには、取得プロセスと適用プロセスは作成時には停止していますが、伝播は作成すると即時にイベントを伝播するようにスケジュールされることに注意してください。リモートの接続先データベースで関連オブジェクトをインスタンス化する前に、取得プロセスを作成する必要があります。伝播と適用プロセスは取得プロセスを起動する前に作成し、ストリーム全体を実行する前にオブジェクトをインスタンス化してください。

関連項目： 単一ソース環境の設定方法を示す詳細な例は、第 21 章「簡単な単一のソース・レプリケーションの例」および第 22 章「単一ソースの異機種間レプリケーションの例」を参照してください。

既存の単一ソース環境への共有オブジェクトの追加

既存の単一ソース環境に既存のデータベース・オブジェクトを追加するには、適切な取得プロセス、伝播および適用プロセスに必要なルールを追加します。稼働中の Streams 環境で取得ルールまたは伝播ルールを作成または変更する前に、新規のルールまたは変更後のルールの結果としてイベントを受信する伝播や適用プロセスが、これらのイベントを処理するように構成されていることを確認してください。つまり、伝播または適用プロセスが存在し、それぞれがイベントを適切に処理するルール・セットに関連付けられている必要があります。これらの伝播と適用プロセスがこれらのイベントを処理するように適切に構成されていないと、イベントが失われる可能性があります。

たとえば、すでに変更が取得、伝播されて他の表に適用されている Streams 環境に、表を追加する必要がある場合を考えます。この表に対する変更を取得する取得プロセスは 1 つのみで、この表に変更を適用する適用プロセスも 1 つのみであるとします。この場合は、次のルール・セットに表レベルの 1 つ以上のルールを追加する必要があります。

- 表に対する変更を取得する取得プロセスのルール・セット
- 表に対する変更を伝播する各伝播のルール・セット
- 表に変更を適用する適用プロセスのルール・セット

管理手順を正しい順序で実行しないと、イベントが失われる可能性があります。たとえば、取得プロセスを停止せず、先に取得ルール・セットにルールを追加した場合、変更を伝播するように指示するルールがなければ伝播は変更を伝播せず、変更が失われる可能性があります。

イベントの消失を回避するために、構成手順を次の順序で完了する必要があります。

1. 取得プロセスを停止するか、伝播ジョブの 1 つを無効化するか、適用プロセスを停止します。手順については、次の該当する項を参照してください。
   - 12-13 ページ「取得プロセスの停止」
   - 13-16 ページ「伝播ジョブの無効化」
   - 14-7 ページ「適用プロセスの停止」
2. 伝播と適用プロセスのルール・セットに、関連ルールを追加します。手順については、次の該当する項を参照してください。
   - 13-14 ページ「伝播のルール・セットへのルールの追加」
   - 14-8 ページ「適用プロセスのルール・セットへのルールの追加」

3. 取得プロセスで使用するルール・セットに関連ルールを追加します。手順については、12-5 ページの「取得プロセスのルール・セットへのルールの追加」を参照してください。

   DBMS_STREAMS_ADM パッケージを使用して取得ルールを追加すると自動的に、指定した表、指定したスキーマまたはデータベース全体に対して、それぞれ DBMS_CAPTURE_ADM パッケージの PREPARE_TABLE_INSTANTIATION、PREPARE_SCHEMA_INSTANTIATION または PREPARE_GLOBAL_INSTANTIATION プロシージャが実行されます。

   次のいずれかの条件に該当する場合は、適切なプロシージャを実行してインスタンス化の準備を手動で行う必要があります。

   - DBMS_RULE_ADM を使用して、取得プロセスのルール・セット内でルールを作成または変更する場合。
   - 取得プロセスは追加されるオブジェクトに対する変更をすでに取得しているため、取得プロセスのルール・セットにはこれらのオブジェクトのルールを追加しない場合。この場合、オブジェクトのルールを環境内の伝播と適用プロセスには追加できますが、取得プロセスには追加できません。

   インスタンス化の準備を手動で行う必要がある場合の手順については、12-10 ページの「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」を参照してください。

4. 各接続先データベースで、Streams 環境に追加する各データベース・オブジェクトをインスタンス化するか、そのインスタンス化 SCN を設定します。接続先データベースにデータベース・オブジェクトが存在しない場合は、エクスポート / インポートを使用してインスタンス化します。接続先データベースにデータベース・オブジェクトが存在する場合は、そのインスタンス化 SCN を設定します。

   - エクスポート / インポートを使用してデータベース・オブジェクトをインスタンス化するには、最初に OBJECT_CONSISTENT エクスポート・パラメータを y に設定してソース・データベースでエクスポートするか、より厳密な一貫性レベルを使用します。次に、接続先データベースで、STREAMS_INSTANTIATION インポート・パラメータを y に設定してインポートします。詳細は、14-34 ページの「エクスポート / インポートを使用したインスタンス化 SCN の設定」を参照してください。
   - 表、スキーマまたはデータベースのインスタンス化 SCN を手動で設定するには、接続先データベースで DBMS_APPLY_ADM パッケージの適切なプロシージャを 1 つ以上実行します。
     - SET_TABLE_INSTANTIATION_SCN
     - SET_SCHEMA_INSTANTIATION_SCN
     - SET_GLOBAL_INSTANTIATION_SCN

これらのプロシージャのいずれかを接続先データベースで実行する場合は、接続先データベースに追加された各オブジェクトに、インスタンス化 SCN の時点でソース・データベースとの一貫性があることを確認する必要があります。

接続先データベースで SET_GLOBAL_INSTANTIATION_SCN を実行する場合は、適用する変更を含むソース・データベース内の既存の各スキーマに対して SET_SCHEMA_INSTANTIATION_SCN も実行し、適用する変更を含むソース・データベース内の既存の各表に対して SET_TABLE_INSTANTIATION_SCN を実行する必要があります。

接続先データベースで SET_SCHEMA_INSTANTIATION_SCN を実行する場合は、適用する DML 変更または DDL 変更を含むスキーマ内の既存の各ソース・データベース表に対して、SET_TABLE_INSTANTIATION_SCN も実行する必要があります。

手順については、14-35 ページの「DBMS_APPLY_ADM パッケージを使用したインスタンス化 SCN の設定」を参照してください。

または、メタデータのエクスポート / インポートを実行して、既存のデータベース・オブジェクトについてインスタンス化 SCN を設定できます。このオプションを選択した場合は、行がインポートされないことを確認してください。また、インポートする接続先データベースに追加された各オブジェクトに、エクスポートを実行したソース・データベースとのエクスポート時点での一貫性があることを確認してください。DML 変更のみを共有する場合は、表レベルのエクスポート / インポートで十分です。DDL 変更も共有する場合は、追加の考慮事項があります。メタデータのエクスポート / インポートを実行する方法の詳細は、14-34 ページの「エクスポート / インポートを使用したインスタンス化 SCN の設定」を参照してください。

5. 手順 1 で停止したプロセスを起動するか、手順 1 で無効化した伝播ジョブを有効化します。手順については、次の該当する項を参照してください。
   - 12-4 ページ「取得プロセスの起動」
   - 13-10 ページ「伝播ジョブの有効化」
   - 14-7 ページ「適用プロセスの起動」

追加したルールによって生じる最初の LCR が適用プロセスに到達する前に、表またはスキーマがインスタンス化されるように、手順 1 で取得プロセスを停止するか、伝播ジョブの 1 つを無効化するか、適用プロセスを停止する必要があります。この操作をしないと、適用ルールを追加したかどうかに応じてイベントが失われたり、適用エラーが発生する可能性があります。

追加した表がこの手順の実行中にソース・データベースで変更されないことと、表の LCR がストリーム内にないことまたは取得を待機していないことが確実な場合は、手順 4 の前に手順 5 を実行して、プロセスや伝播ジョブの停止期間を短縮できます。

関連項目： 既存の単一ソース環境にオブジェクトを追加する場合の詳細な例は、22-57 ページの「既存の Streams レプリケーション環境へのオブジェクトの追加」を参照してください。

既存の単一ソース環境への新規接続先データベースの追加

既存の単一ソース環境に接続先データベースを追加するには、新規の接続先データベースで 1 つ以上の新規適用プロセスを作成し、必要な場合は、そのデータベースに変更を伝播する 1 つ以上の伝播を構成します。また、新規接続先データベースに伝播するストリーム内の既存の伝播に、ルールの追加が必要になる場合があります。

11-18 ページの「既存の単一ソース環境への共有オブジェクトの追加」で説明した例と同様に、稼働中の Streams 環境で伝播ルールを作成または変更する前に、新規のルールまたは変更後のルールの結果としてイベントを受信する伝播や適用プロセスが、これらのイベントを処理するように構成されていることを確認してください。このように構成されていないと、イベントが失われる可能性があります。

イベントの消失を回避するために、構成手順を次の順序で完了する必要があります。

1. 前述した必要なタスクを完了し、Streams 用に新規の接続先データベースを準備します。
   - 11-2 ページ「Streams 管理者の構成」
   - 11-4 ページ「Streams に関連する初期化パラメータの設定」
   - 11-13 ページ「ネットワーク接続性とデータベース・リンクの構成」

   これらのタスクの一部は、新規データベースでは不要な場合があります。
2. 接続先データベースに存在しない場合は、必要な Streams キューを作成します。適用プロセスを作成するときに、その適用プロセスを特定の Streams キューに関連付けます。手順については、13-2 ページの「Streams のキューの作成」を参照してください。
3. 新規の接続先データベースで、ソース・データベースからの変更を適用できるように 1 つ以上の適用プロセスを作成します。各適用プロセスで、変更の適用に適切なルール・セットが使用されることを確認します。新規データベースでは適用プロセスを起動しないでください。手順については、14-2 ページの「適用プロセスの作成」を参照してください。

   適用プロセスを停止したままにすると、ソース・データベースで行われた変更は新規データベースのインスタンス化が完了するまで適用されません。それ以外の場合は、データが不正になったりエラーが発生します。
4. 必要な伝播を、ソース・データベースからの変更を新規の接続先データベースに伝播するように構成します。各伝播で、変更の伝播に適切なルール・セットが使用されることを確認します。13-7 ページの「伝播の作成」を参照してください。

5. ソース・データベースで、新規の接続先データベースの適用プロセスによって変更が適用される各データベース・オブジェクトについて、インスタンス化の準備を行います。指定した表、指定したスキーマまたはデータベース全体に対して、それぞれ DBMS_CAPTURE_ADM パッケージの PREPARE_TABLE_INSTANTIATION、PREPARE_SCHEMA_INSTANTIATION または PREPARE_GLOBAL_INSTANTIATION プロシージャを実行します。手順については、12-10 ページの「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」を参照してください。

6. 新規の接続先データベースで、適用プロセスによって変更が適用される各データベース・オブジェクトをインスタンス化するか、そのインスタンス化 SCN を設定します。新規の接続先データベースにデータベース・オブジェクトが存在しない場合は、エクスポート / インポートを使用してインスタンス化します。新規の接続先データベースにデータベース・オブジェクトが存在する場合は、そのインスタンス化 SCN を設定します。

   - エクスポート / インポートを使用してデータベース・オブジェクトをインスタンス化するには、最初に OBJECT_CONSISTENT エクスポート・パラメータを y に設定してソース・データベースでエクスポートするか、より厳密な一貫性レベルを使用します。次に、新規の接続先データベースで、STREAMS_INSTANTIATION インポート・パラメータを y に設定してインポートします。詳細は、14-34 ページの「エクスポート / インポートを使用したインスタンス化 SCN の設定」を参照してください。

   - 表、スキーマまたはデータベースのインスタンス化 SCN を手動で設定するには、新規の接続先データベースで DBMS_APPLY_ADM パッケージの適切なプロシージャを 1 つ以上実行します。

     - SET_TABLE_INSTANTIATION_SCN
     - SET_SCHEMA_INSTANTIATION_SCN
     - SET_GLOBAL_INSTANTIATION_SCN

     これらのプロシージャのいずれかを実行する場合は、新規の接続先データベースの共有オブジェクトに、インスタンス化 SCN の時点でソース・データベースとの一貫性があることを確認する必要があります。

     接続先データベースで SET_GLOBAL_INSTANTIATION_SCN を実行する場合は、適用する DDL 変更を含むソース・データベース内の既存の各スキーマに対して SET_SCHEMA_INSTANTIATION_SCN も実行し、適用する DML 変更または DDL 変更を含むソース・データベース内の既存の各表に対して SET_TABLE_INSTANTIATION_SCN を実行する必要があります。

     接続先データベースで SET_SCHEMA_INSTANTIATION_SCN を実行する場合は、適用する DML 変更または DDL 変更を含むスキーマ内の既存の各ソース・データベース表に対して、SET_TABLE_INSTANTIATION_SCN も実行する必要があります。

     手順については、14-35 ページの「DBMS_APPLY_ADM パッケージを使用したインスタンス化 SCN の設定」を参照してください。

または、メタデータのエクスポート / インポートを実行して、既存のデータベース・オブジェクトについてインスタンス化 SCN を設定できます。このオプションを選択した場合は、行がインポートされないことを確認してください。また、インポートする接続先データベースの共有オブジェクトに、エクスポートを実行したソース・データベースとのエクスポート時点での一貫性があることを確認してください。DML 変更のみを共有する場合は、表レベルのエクスポート / インポートで十分です。DDL 変更も共有する場合は、追加の考慮事項があります。メタデータのエクスポート / インポートを実行する方法の詳細は、14-34 ページの「エクスポート / インポートを使用したインスタンス化 SCN の設定」を参照してください。

7. 手順 3 で作成した適用プロセスを起動します。手順については、14-7 ページの「適用プロセスの起動」を参照してください。

関連項目： 既存の単一ソース環境にデータベースを追加する場合の詳細な例は、22-68 ページの「既存の Streams レプリケーション環境へのデータベースの追加」を参照してください。

新規の複数ソース環境の作成

この項では、新規の複数ソース Streams 環境を作成する場合の一般的な手順について説明します。複数ソース環境とは、共有データに複数のソース・データベースが存在する環境です。

この例では、次の用語を使用しています。

- **実装済みデータベース :** 新規の複数ソース環境を作成する前に、すでに共有データベース・オブジェクトが含まれているデータベース。新規の Streams 環境を作成するには、少なくとも 1 つの実装済みデータベースが必要です。
- **エクスポート・データベース :** 共有データベース・オブジェクトのエクスポートを実行する実装済みデータベース。このエクスポートを使用して、インポート・データベースで共有データベース・オブジェクトがインスタンス化されます。すべてのデータベースが実装済みデータベースである環境では、エクスポート・データベースが存在しないことがあります。
- **インポート・データベース :** 新規の複数ソース環境を作成する前には、共有データベース・オブジェクトが含まれていないデータベース。インポート・データベースでは、エクスポート・データベースからのエクスポート・ダンプ・ファイルを使用して、共有データベース・オブジェクトをインスタンス化します。すべてのデータベースが実装済みデータベースである環境では、インポート・データベースが存在しないことがあります。

次の手順に従って新規の複数ソース環境を作成します。

> **注意：** Streams 環境に追加するデータベースでは、そのデータベースでのインスタンス化が完了するまでは、共有されるオブジェクトに対して変更が行われないことを確認してください。

1. 前述した必要なタスクを完了し、Streams 用の環境内で各データベースを準備します。
   - 11-2 ページ「Streams 管理者の構成」
   - 11-4 ページ「Streams に関連する初期化パラメータの設定」
   - 11-12 ページ「Streams の取得プロセスを実行するためのデータベースの構成」
   - 11-13 ページ「ネットワーク接続性とデータベース・リンクの構成」

   これらのタスクの一部は、一部のデータベースでは不要な場合があります。
2. 各実装済みデータベースで、共有オブジェクトに必要なサプリメンタル・ロギングを指定します。手順については、12-8 ページの「ソース・データベースでのサプリメンタル・ロギングの指定」を参照してください。
3. 存在しない場合は、必要な Streams キューを作成します。取得プロセスまたは適用プロセスを作成するときに、そのプロセスを特定の Streams キューに関連付けます。伝播を作成するときには、その伝播を特定のソース・キューおよび宛先キューに関連付けます。手順については、13-2 ページの「Streams のキューの作成」を参照してください。
4. 各データベースで、環境に必要な取得プロセス、伝播および適用プロセスを作成します。これらは任意の順序で作成できます。
   - 変更を取得する各データベースで、1 つ以上の取得プロセスを作成します。各取得プロセスで、変更の取得に適切なルール・セットが使用されることを確認します。作成した取得プロセスは起動しないでください。手順については、12-2 ページの「取得プロセスの作成」を参照してください。

     DBMS_STREAMS_ADM パッケージを使用して取得ルールを追加すると自動的に、指定した表、指定したスキーマまたはデータベース全体に対して、それぞれ DBMS_CAPTURE_ADM パッケージの PREPARE_TABLE_INSTANTIATION、PREPARE_SCHEMA_INSTANTIATION または PREPARE_GLOBAL_INSTANTIATION プロシージャが実行されます。

     次のいずれかの条件に該当する場合は、適切なプロシージャを実行してインスタンス化の準備を手動で行う必要があります。

     - DBMS_RULE_ADM パッケージを使用して取得ルールを追加または変更する場合。
     - 既存の取得プロセスを使用し、共有オブジェクトの取得ルールは追加しない場合。

インスタンス化の準備を手動で行う必要がある場合の手順については、12-10 ページの「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」を参照してください。

- 取得イベントをソース・キューから宛先キューに伝播する伝播をすべて作成します。各伝播で、変更の伝播に適切なルール・セットが使用されることを確認します。手順については、13-7 ページの「伝播の作成」を参照してください。
- 変更を適用する各データベースで、1 つ以上の適用プロセスを作成します。各適用プロセスで、変更の適用に適切なルール・セットが使用されることを確認します。作成した適用プロセスは起動しないでください。手順については、14-2 ページの「適用プロセスの作成」を参照してください。

前述の手順を完了してから、環境に応じて次の各項の手順を完了します。各項の一方の手順のみを完了すればよい場合と、両方の手順を完了する必要がある場合があります。

- 各実装済みデータベースには、11-25 ページの「複数ソース環境の作成時の実装済みデータベースの構成」の手順を完了します。
- 各インポート・データベースには、11-26 ページの「新規環境作成時のインポート・データベースへの共有オブジェクトの追加」の手順を完了します。

複数ソース環境の作成時の実装済みデータベースの構成

環境に複数の実装済みデータベースが存在する場合は、11-23 ページの「新規の複数ソース環境の作成」の手順を完了してから、実装済みデータベースについて次の手順を完了します。

1. 実装済みデータベースごとに、実装済みソース・データベースの接続先データベースとなる環境内の他の各実装済みデータベースで、インスタンス化 SCN を設定します。これらのインスタンス化 SCN の設定は必須です。また、特定の実装済みデータベースで行われ、そのデータベースの対応する SCN より後にコミットされる変更のみが、他の実装済みデータベースで適用されます。

   実装済みデータベースごとに、これらのインスタンス化 SCN を次のいずれかの方法で設定できます。

   a. 実装済みデータベースで共有オブジェクトのメタデータのみのエクスポートを実行し、そのメタデータを他のそれぞれの実装済みデータベースでインポートします。このインポートによって、実装済みデータベースに必須のインスタンス化 SCN が、他の実装済みデータベースで設定されます。行がインポートされないことを確認してください。また、メタデータのインポートを実行する各実装済みデータベースの共有オブジェクトに、メタデータのエクスポートを実行した実装済みデータベースとのエクスポート時点での一貫性があることを確認してください。

      DML 変更のみを共有する場合は、表レベルのエクスポート / インポートで十分です。DDL 変更も共有する場合は、追加の考慮事項があります。メタデータのエクスポート / インポートを実行する方法の詳細は、14-34 ページの「エクスポート / インポートを使用したインスタンス化 SCN の設定」を参照してください。

b. 他の各実装済みデータベースで、インスタンス化 SCN を手動で設定します。この操作は共有オブジェクトごとに行います。各実装済みデータベースの共有オブジェクトに、そのデータベースで設定したインスタンス化 SCN との一貫性があることを確認してください。手順については、14-35 ページの「DBMS_APPLY_ADM パッケージを使用したインスタンス化 SCN の設定」を参照してください。

新規環境作成時のインポート・データベースへの共有オブジェクトの追加

11-23 ページの「新規の複数ソース環境の作成」の手順を完了した後に、インポート・データベースについて次の手順を完了します。

1. エクスポート・データベースとして使用する実装済みデータベースを選択します。まだインスタンス化は実行しないでください。
2. インポート・データベースごとに、インポート・データベースの接続先データベースとなる環境内の他のすべてのデータベースで、インスタンス化 SCN を設定します。インスタンス化 SCN を設定するデータベースには、実装済みデータベースと他のインポート・データベースが含まれることがあります。

   a. インスタンス化中に、またはその後に共有 DDL の変更によって、インポート・データベースで 1 つ以上のスキーマが作成される場合は、環境内の他のすべてのデータベースで、このインポート・データベースについて DBMS_APPLY_ADM パッケージの SET_GLOBAL_INSTANTIATION_SCN プロシージャを実行します。

   b. インポート・データベースにスキーマが存在し、インスタンス化中に、またはその後に共有 DDL の変更によって、そのスキーマに 1 つ以上の表が作成される場合は、環境内の他のすべてのデータベースで、このインポート・データベースのスキーマについて DBMS_APPLY_ADM パッケージの SET_SCHEMA_INSTANTIATION_SCN プロシージャを実行します。この操作は、この種のスキーマごとに行います。

   手順については、14-33 ページの「接続先データベースでのインスタンス化 SCN の設定」を参照してください。

   これらのプロシージャは、インポート・データベースで表がインスタンス化される前に実行し、これらのインポート・データベースについてはローカルの取得プロセスがすでに構成されているため、インスタンス化中に作成される表ごとに SET_TABLE_INSTANTIATION_SCN を実行する必要はありません。
3. 手順 1 で選択したエクスポート・データベースで、OBJECT_CONSISTENT エクスポート・パラメータを y に設定して共有データのエクスポートを実行するか、より厳密な一貫性レベルを使用します。次に、各インポート・データベースで、STREAMS_INSTANTIATION インポート・パラメータを y に設定してインポートを実行します。エクスポート / インポートの使用方法は、11-8 ページの「Streams に関連するエクスポート・ユーティリティとインポート・ユーティリティのパラメータ設定」および『Oracle9*i* データベース・ユーティリティ』を参照してください。

4. エクスポート・データベース以外の実装済みデータベースごとに、実装済みソース・データベースの宛先データベースとなる各インポート・データベースにインスタンス化 SCN を設定します。これらのインスタンス化 SCN の設定は必須です。また、実装済みデータベースで行われ、そのデータベースの対応する SCN より後にコミットされる変更のみが、インポート・データベースで適用されます。

   これらのインスタンス化 SCN を設定するには、次の方法があります。

   a. 各実装済みデータベースでメタデータのみのエクスポートを実行し、そのメタデータを各インポート・データベースでインポートします。各インポートによって、実装済みデータベースに必須のインスタンス化 SCN が、インポート・データベースで設定されます。この場合は、インポート・データベースの共有オブジェクトに、エクスポート時点で実装済みデータベースとの一貫性があることを確認してください。

      DML 変更のみを共有する場合は、表レベルのエクスポート / インポートで十分です。DDL 変更も共有する場合は、追加の考慮事項があります。メタデータのエクスポート / インポートを実行する方法の詳細は、14-34 ページの「エクスポート / インポートを使用したインスタンス化 SCN の設定」を参照してください。

   b. 実装済みデータベースごとに、各インポート・データベースで各共有オブジェクトのインスタンス化 SCN を手動で設定します。各インポート・データベースの共有オブジェクトに、対応するインスタンス化 SCN の時点で実装済みデータベースとの一貫性があることを確認してください。手順については、14-35 ページの「DBMS_APPLY_ADM パッケージを使用したインスタンス化 SCN の設定」を参照してください。

複数ソース環境の構成完了

この項の手順を完了する前に、次のタスクを完了している必要があります。

- 11-23 ページ「新規の複数ソース環境の作成」
- 環境に複数の実装済みデータベースがある場合は、11-25 ページの「複数ソース環境の作成時の実装済みデータベースの構成」。
- 環境に 1 つ以上のインポート・データベースがある場合は、11-26 ページの「新規環境作成時のインポート・データベースへの共有オブジェクトの追加」。

前述の構成手順をすべて完了してから、次の手順を行います。

1. 環境内の各適用プロセスを起動します。手順については、14-7 ページの「適用プロセスの起動」を参照してください。
2. 環境内の各取得プロセスを起動します。手順については、12-4 ページの「取得プロセスの起動」を参照してください。

関連項目： 複数ソース環境の作成方法を示す詳細な例は、第 23 章「複数のソース・レプリケーションの例」を参照してください。

既存の複数ソース環境への共有オブジェクトの追加

既存の複数ソース環境に既存のデータベース・オブジェクトを追加するには、適切な取得プロセス、伝播および適用プロセスに必要なルールを追加します。

この例では、次の用語を使用しています。

- **実装済みデータベース :** 複数ソース環境に追加する共有データベース・オブジェクトがすでに含まれているデータベース。環境にオブジェクトを追加するには、少なくとも 1 つは実装済みデータベースが必要です。
- **エクスポート・データベース :** 環境に追加するデータベース・オブジェクトのエクスポートを実行する実装済みデータベース。このエクスポートを使用して、インポート・データベースで追加したデータベース・オブジェクトがインスタンス化されます。すべてのデータベースが実装済みデータベースである環境では、エクスポート・データベースが存在しないことがあります。
- **インポート・データベース :** 複数ソース環境に追加する前には、共有データベース・オブジェクトが含まれていないデータベース。インポート・データベースでは、エクスポート・データベースからのエクスポート・ダンプ・ファイルを使用して、追加したデータベース・オブジェクトをインスタンス化します。すべてのデータベースが実装済みデータベースである環境では、インポート・データベースが存在しないことがあります。

稼働中の Streams 環境で取得ルールまたは伝播ルールを作成または変更する前に、新規のルールまたは変更後のルールの結果としてイベントを受信する伝播や適用プロセスが、これらのイベントを処理するように構成されていることを確認してください。つまり、伝播または適用プロセスが存在し、それぞれがイベントを適切に処理するルール・セットに関連付けられている必要があります。これらの伝播と適用プロセスがこれらのイベントを処理するように適切に構成されていないと、イベントが失われる可能性があります。

たとえば、すでに変更が取得、伝播されて他の表に適用されている Streams 環境に、新規の表を追加する必要がある場合を考えます。環境内の複数の取得プロセスがこの表に対する変更を取得し、複数の適用プロセスがこの表に変更を適用するとします。この場合は、次のルール・セットに表レベルの 1 つ以上のルールを追加する必要があります。

- 表に対する変更を取得する各取得プロセスのルール・セット
- 表に対する変更を伝播する各伝播のルール・セット
- 表に変更を適用する各適用プロセスのルール・セット

管理手順を正しい順序で実行しないと、イベントが失われる可能性があります。たとえば、取得プロセスを停止せず、先に取得ルール・セットにルールを追加した場合、変更を伝播するように指示するルールがなければ伝播は変更を伝播せず、変更が失われる可能性があります。

イベントの消失を回避するために、構成手順を次の順序で完了する必要があります。

1. 各実装済みデータベースで、環境に追加するオブジェクトに必要なサプリメンタル・ロギングを指定します。手順については、12-8 ページの「ソース・データベースでのサプリメンタル・ロギングの指定」を参照してください。
2. 追加したオブジェクトに対する変更を取得する取得プロセスをすべて停止するか、それを伝播する伝播ジョブをすべて無効化するか、追加したオブジェクトに変更を適用する適用プロセスをすべて停止します。手順については、次の該当する項を参照してください。
   - 12-13 ページ「取得プロセスの停止」
   - 13-16 ページ「伝播ジョブの無効化」
   - 14-7 ページ「適用プロセスの停止」
3. 追加したオブジェクトに変更を伝播または適用する伝播と適用プロセスについて、ルール・セットに関連ルールを追加します。手順については、次の該当する項を参照してください。
   - 13-14 ページ「伝播のルール・セットへのルールの追加」
   - 14-8 ページ「適用プロセスのルール・セットへのルールの追加」
4. 追加したオブジェクトに対する変更を取得する各取得プロセスで使用されるルール・セットに、関連ルールを追加します。手順については、12-5 ページの「取得プロセスのルール・セットへのルールの追加」を参照してください。

   DBMS_STREAMS_ADM パッケージを使用して取得ルールを追加すると自動的に、指定した表、指定したスキーマまたはデータベース全体に対して、それぞれ DBMS_CAPTURE_ADM パッケージの PREPARE_TABLE_INSTANTIATION、PREPARE_SCHEMA_INSTANTIATION または PREPARE_GLOBAL_INSTANTIATION プロシージャが実行されます。

   次のいずれかの条件に該当する場合は、適切なプロシージャを実行してインスタンス化の準備を手動で行う必要があります。

   - DBMS_RULE_ADM を使用して、取得プロセスのルール・セット内でルールを作成または変更する場合。
   - 取得プロセスは追加されるオブジェクトに対する変更をすでに取得しているため、取得プロセスのルール・セットにはこれらのオブジェクトのルールを追加しない場合。この場合、オブジェクトのルールを環境内の伝播と適用プロセスには追加できますが、取得プロセスには追加できません。

   インスタンス化の準備を手動で行う必要がある場合の手順については、12-10 ページの「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」を参照してください。

前述の手順を完了してから、環境に応じて次の各項の手順を完了します。各項の一方の手順のみを完了すればよい場合と、両方の手順を完了する必要がある場合があります。

- 各実装済みデータベースには、11-30 ページの「共有オブジェクト追加時の実装済みデータベースの構成」の手順を完了します。
- 各インポート・データベースには、11-31 ページの「既存の環境におけるインポート・データベースへの共有オブジェクトの追加」の手順を完了します。

共有オブジェクト追加時の実装済みデータベースの構成

11-28 ページの「既存の複数ソース環境への共有オブジェクトの追加」の手順を完了した後に、各実装済みデータベースについて次の手順を完了します。

1. 実装済みデータベースごとに、環境内の他の各実装済みデータベースで、追加した各オブジェクトにインスタンス化 SCN を設定します。これらのインスタンス化 SCN の設定は必須です。また、特定の実装済みデータベースで行われ、そのデータベースの対応する SCN より後にコミットされる変更のみが、他の実装済みデータベースで適用されます。

   実装済みデータベースごとに、追加した各オブジェクトにこれらのインスタンス化 SCN を次のいずれかの方法で設定できます。

   a. 実装済みデータベースで追加したオブジェクトのメタデータのみのエクスポートを実行し、そのメタデータを他のそれぞれの実装済みデータベースでインポートします。このインポートによって、データベースに必須のインスタンス化 SCN が、他のデータベースで設定されます。行がインポートされないことを確認してください。また、他の各実装済みデータベースの共有オブジェクトに、エクスポートを実行した実装済みデータベースとのエクスポート時点での一貫性があることを確認してください。

      DML 変更のみを共有する場合は、表レベルのエクスポート / インポートで十分です。DDL 変更も共有する場合は、追加の考慮事項があります。メタデータのエクスポート / インポートを実行する方法の詳細は、14-34 ページの「エクスポート / インポートを使用したインスタンス化 SCN の設定」を参照してください。

   b. 他の各実装済みデータベースで、追加したオブジェクトについてインスタンス化 SCN を手動で設定します。各実装済みデータベースに追加した各オブジェクトに、そのデータベースで設定したインスタンス化 SCN との一貫性があることを確認してください。手順については、14-35 ページの「DBMS_APPLY_ADM パッケージを使用したインスタンス化 SCN の設定」を参照してください。

既存の環境におけるインポート・データベースへの共有オブジェクトの追加

11-28 ページの「既存の複数ソース環境への共有オブジェクトの追加」の手順を完了した後に、インポート・データベースについて次の手順を完了します。

1. エクスポート・データベースとして使用する実装済みデータベースを選択します。まだインスタンス化は実行しないでください。
2. インポート・データベースごとに、インポート・データベースの接続先データベースとなる環境内の他のすべてのデータベースで、追加したオブジェクトにインスタンス化 SCN を設定します。インスタンス化 SCN を設定するデータベースは、実装済みデータベースの場合もあれば、他のインポート・データベースの場合もあります。

   a. インスタンス化中に、またはその後に共有 DDL の変更によって、インポート・データベースで 1 つ以上のスキーマが作成される場合は、環境内の他のすべてのデータベースで、このインポート・データベースについて DBMS_APPLY_ADM パッケージの SET_GLOBAL_INSTANTIATION_SCN プロシージャを実行します。

   b. インポート・データベースにスキーマが存在し、インスタンス化中に、またはその後に共有 DDL の変更によって、そのスキーマに 1 つ以上の表が作成される場合は、環境内の他の各データベースで、このインポート・データベースのスキーマについて DBMS_APPLY_ADM パッケージの SET_SCHEMA_INSTANTIATION_SCN プロシージャを実行します。この操作は、この種のスキーマごとに行います。

   手順については、14-33 ページの「接続先データベースでのインスタンス化 SCN の設定」を参照してください。

   これらのプロシージャは、インポート・データベースで表がインスタンス化される前に実行し、これらのインポート・データベースについてはローカルの取得プロセスがすでに構成されているため、インスタンス化中に作成される表ごとに SET_TABLE_INSTANTIATION_SCN を実行する必要はありません。
3. 手順 1 で選択したエクスポート・データベースで、OBJECT_CONSISTENT エクスポート・パラメータを y に設定して追加したオブジェクトのエクスポートを実行するか、より厳密な一貫性レベルを使用します。次に、各インポート・データベースで、STREAMS_INSTANTIATION インポート・パラメータを y に設定して、追加したオブジェクトのインポートを実行します。エクスポート / インポートの使用方法は、11-8 ページの「Streams に関連するエクスポート・ユーティリティとインポート・ユーティリティのパラメータ設定」および『Oracle9*i* データベース・ユーティリティ』を参照してください。
4. エクスポート・データベース以外の実装済みデータベースごとに、実装済みソース・データベースの宛先データベースとなる各インポート・データベースで、追加したオブジェクトにインスタンス化 SCN を設定します。これらのインスタンス化 SCN の設定は必須です。また、実装済みデータベースで行われ、そのデータベースの対応する SCN より後にコミットされる変更のみが、インポート・データベースで適用されます。

実装済みデータベースごとに、追加したオブジェクトにこれらのインスタンス化 SCN を次のいずれかの方法で設定できます。

a. 実装済みデータベースで追加したオブジェクトのメタデータのみのエクスポートを実行し、そのメタデータを各インポート・データベースでインポートします。各インポートによって、実装済みデータベースに必須のインスタンス化 SCN が、インポート・データベースで設定されます。この場合は、インポート・データベースで追加した各オブジェクトに、エクスポート時点で実装済みデータベースとの一貫性があることを確認してください。

DML 変更のみを共有する場合は、表レベルのエクスポート / インポートで十分です。DDL 変更も共有する場合は、追加の考慮事項があります。メタデータのエクスポート / インポートを実行する方法の詳細は、14-34 ページの「エクスポート / インポートを使用したインスタンス化 SCN の設定」を参照してください。

b. 各インポート・データベースで、追加したオブジェクトについてインスタンス化 SCN を手動で設定します。各インポート・データベースで追加した各オブジェクトに、対応するインスタンス化 SCN の時点で実装済みデータベースとの一貫性があることを確認してください。手順については、14-35 ページの「DBMS_APPLY_ADM パッケージを使用したインスタンス化 SCN の設定」を参照してください。

複数ソース環境構成へのオブジェクトの追加の完了

構成を完了する前に、次のタスクを完了する必要があります。

- 11-28 ページ「既存の複数ソース環境への共有オブジェクトの追加」
- 環境に実装済みデータベースがある場合は、11-30 ページの「共有オブジェクト追加時の実装済みデータベースの構成」
- 環境にインポート・データベースがある場合は、11-31 ページの「既存の環境におけるインポート・データベースへの共有オブジェクトの追加」

前述の構成手順をすべて完了してから、「既存の複数ソース環境への共有オブジェクトの追加」の 11-29 ページの手順 2 で停止した各プロセスを起動し、無効化した各伝播ジョブを有効化します。手順については、次の該当する項を参照してください。

- 12-4 ページ「取得プロセスの起動」
- 13-10 ページ「伝播ジョブの有効化」
- 14-7 ページ「適用プロセスの起動」

既存の複数ソース環境への新規データベースの追加

次の手順に従って、既存の複数ソース Streams 環境に新規データベースを追加します。

> **注意：** Streams 環境に追加するデータベースでは、そのデータベースでのインスタンス化が完了するまでは、共有されるオブジェクトに対して変更が行われないことを確認してください。

1. 前述した必要なタスクを完了し、Streams 用に新規データベースを準備します。
   - 11-2 ページ「Streams 管理者の構成」
   - 11-4 ページ「Streams に関連する初期化パラメータの設定」
   - 11-12 ページ「Streams の取得プロセスを実行するためのデータベースの構成」
   - 11-13 ページ「ネットワーク接続性とデータベース・リンクの構成」

   これらのタスクの一部は、新規データベースでは不要な場合があります。
2. 存在しない場合は、必要な Streams キューを作成します。取得プロセスまたは適用プロセスを作成するときに、そのプロセスを特定の Streams キューに関連付けます。伝播を作成するときには、その伝播を特定のソース・キューおよび宛先キューに関連付けます。手順については、13-2 ページの「Streams のキューの作成」を参照してください。
3. 新規データベースで、ソース・データベースからの変更を適用できるように 1 つ以上の適用プロセスを作成します。各適用プロセスで、変更の適用に適切なルール・セットが使用されることを確認します。新規データベースでは適用プロセスを起動しないでください。手順については、14-2 ページの「適用プロセスの作成」を参照してください。

   適用プロセスを停止したままにすると、ソース・データベースで行われた変更は新規データベースのインスタンス化が完了するまで適用されません。それ以外の場合は、データが不正になったりエラーが発生します。
4. 新規データベースがソース・データベースとなる場合は、新規データベースで行われる変更の接続先データベースとなる全データベースで、新規データベースからの変更を適用する 1 つ以上の適用プロセスを作成します。各適用プロセスで、変更の適用に適切なルール・セットが使用されることを確認します。これらの新規適用プロセスは起動しないでください。手順については、14-2 ページの「適用プロセスの作成」を参照してください。
5. 新規データベースのソース・データベースとなるデータベースで、変更を新規データベースに送信するように伝播を構成します。各伝播で、変更の伝播に適切なルール・セットが使用されることを確認します。13-7 ページの「伝播の作成」を参照してください。

6. 新規データベースがソース・データベースとなる場合は、そこでの変更を各接続先データベースに送信するように、新規データベースで伝播を構成します。各伝播で、変更の伝播に適切なルール・セットが使用されることを確認します。13-7 ページの「伝播の作成」を参照してください。

7. 新規データベースがソース・データベースとなり、そこに既存の共有オブジェクトがある場合は、その共有オブジェクトに必要なサプリメンタル・ロギングを指定します。手順については、12-8 ページの「ソース・データベースでのサプリメンタル・ロギングの指定」を参照してください。

8. 新規データベースの各ソース・データベースで、新規データベースの適用プロセスによって変更が適用される各データベース・オブジェクトについて、インスタンス化の準備を行います。指定した表、指定したスキーマまたはデータベース全体に対して、それぞれ DBMS_CAPTURE_ADM パッケージの PREPARE_TABLE_INSTANTIATION、PREPARE_SCHEMA_INSTANTIATION または PREPARE_GLOBAL_INSTANTIATION プロシージャを実行します。手順については、12-10 ページの「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」を参照してください。

9. 新規データベースがソース・データベースとなる場合は、関連する変更を取得するために 1 つ以上の取得プロセスを作成します。手順については、12-2 ページの「取得プロセスの作成」を参照してください。

   DBMS_STREAMS_ADM パッケージを使用して取得ルールを追加すると自動的に、指定した表、指定したスキーマまたはデータベース全体に対して、それぞれ DBMS_CAPTURE_ADM パッケージの PREPARE_TABLE_INSTANTIATION、PREPARE_SCHEMA_INSTANTIATION または PREPARE_GLOBAL_INSTANTIATION プロシージャが実行されます。

   次のいずれかの条件に該当する場合は、適切なプロシージャを実行してインスタンス化の準備を手動で行う必要があります。

   - DBMS_RULE_ADM パッケージを使用して取得ルールを追加または変更する場合。
   - 既存の取得プロセスを使用し、共有オブジェクトの取得ルールは追加しない場合。

   インスタンス化の準備を手動で行う必要がある場合の手順については、12-10 ページの「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」を参照してください。

10. 新規データベースがソース・データベースとなる場合は、手順 9 で作成した取得プロセスを起動します。手順については、12-4 ページの「取得プロセスの起動」を参照してください。

前述の手順を完了してから、次の該当する項の手順を完了します。

- 新規データベースと共有するオブジェクトが新規データベースに存在する場合は、11-35 ページの「共有オブジェクトが新規データベースに存在する場合のデータベースの構成」の手順を完了します。
- 新規データベースと共有するオブジェクトが新規データベースに存在しない場合は、11-36 ページの「新規データベースへの共有オブジェクトの追加」の手順を完了します。

共有オブジェクトが新規データベースに存在する場合のデータベースの構成

新規データベースと共有するオブジェクトがすでにそのデータベースに存在する場合は、11-33 ページの「既存の複数ソース環境への新規データベースの追加」の手順を完了してから次の手順を完了します。

1. 新規データベースのソース・データベースごとに、新規データベースでインスタンス化 SCN を設定します。これらのインスタンス化 SCN の設定は必須です。また、ソース・データベースで行われ、そのデータベースの対応する SCN より後にコミットされる変更のみが、新規データベースで適用されます。

   新規データベースのソース・データベースごとに、これらのインスタンス化 SCN を次のいずれかの方法で設定できます。

   a. ソース・データベースで共有オブジェクトのメタデータのみのエクスポートを実行し、そのメタデータを新規データベースでインポートします。このインポートによって、ソース・データベースに必須のインスタンス化 SCN が、新規データベースで設定されます。行がインポートされないことを確認してください。この場合は、新規データベースの共有オブジェクトに、エクスポート時点でソース・データベースとの一貫性があることを確認してください。

      DML 変更のみを共有する場合は、表レベルのエクスポート / インポートで十分です。DDL 変更も共有する場合は、追加の考慮事項があります。メタデータのエクスポート / インポートを実行する方法の詳細は、14-34 ページの「エクスポート / インポートを使用したインスタンス化 SCN の設定」を参照してください。

   b. 新規データベースで、共有オブジェクトについてインスタンス化 SCN を手動で設定します。新規データベースの共有オブジェクトに、対応するインスタンス化 SCN の時点でソース・データベースとの一貫性があることを確認してください。手順については、14-35 ページの「DBMS_APPLY_ADM パッケージを使用したインスタンス化 SCN の設定」を参照してください。

2. 新規データベースについて、それぞれの接続先データベースでインスタンス化 SCN を設定します。これらのインスタンス化 SCN の設定は必須です。また、新規のソース・データベースで行われ、それに対応する SCN より後にコミットされる変更のみが、接続先データベースで適用されます。新規データベースがソース・データベースでない場合は、この手順を実行しないでください。

新規データベースのインスタンス化 SCN を設定するには、次の方法があります。

a. 新規データベースでメタデータのみのエクスポートを実行し、そのメタデータを各接続先データベースでインポートします。行がインポートされないことを確認してください。このインポートによって、新規データベースに必須のインスタンス化 SCN が、各接続先データベースで設定されます。この場合は、各接続先データベースの共有オブジェクトに、エクスポート時点で新規データベースとの一貫性があることを確認してください。

DML 変更のみを共有する場合は、表レベルのエクスポート / インポートで十分です。DDL 変更も共有する場合は、追加の考慮事項があります。メタデータのエクスポート / インポートを実行する方法の詳細は、14-34 ページの「エクスポート / インポートを使用したインスタンス化 SCN の設定」を参照してください。

b. 接続先データベースで、共有オブジェクトについてインスタンス化 SCN を手動で設定します。各接続先データベースの共有オブジェクトに、対応するインスタンス化 SCN の時点で新規データベースとの一貫性があることを確認してください。手順については、14-35 ページの「DBMS_APPLY_ADM パッケージを使用したインスタンス化 SCN の設定」を参照してください。

3. 11-33 ページの手順 3 で新規データベースに作成した適用プロセスを起動します。手順については、14-7 ページの「適用プロセスの起動」を参照してください。

4. 11-33 ページの手順 4 で他の各接続先データベースに作成した適用プロセスを起動します。手順については、14-7 ページの「適用プロセスの起動」を参照してください。新規データベースがソース・データベースでない場合は、この手順を実行しないでください。

新規データベースへの共有オブジェクトの追加

新規データベースと共有するオブジェクトがそのデータベースに存在しない場合は、11-33 ページの「既存の複数ソース環境への新規データベースの追加」の手順を完了してから次の手順を完了します。

1. 新規データベースが他のデータベースのソース・データベースである場合は、新規ソース・データベースの各接続先データベースで、新規データベースのインスタンス化 SCN を設定します。

a. インスタンス化中に、またはその後に共有 DDL の変更によって、新規データベースで 1 つ以上のスキーマが作成される場合は、新規データベースの各接続先データベースで、新規データベースについて DBMS_APPLY_ADM パッケージの SET_GLOBAL_INSTANTIATION_SCN プロシージャを実行します。

b. 新規データベースにスキーマが存在し、インスタンス化中に、またはその後に共有 DDL の変更によって、そのスキーマに 1 つ以上の表が作成される場合は、新規データベースの各接続先データベースで、そのスキーマについて DBMS_APPLY_ADM パッケージの SET_SCHEMA_INSTANTIATION_SCN プロシージャを実行します。この操作は、この種のスキーマごとに行います。

手順については、14-33 ページの「接続先データベースでのインスタンス化 SCN の設定」を参照してください。

これらのプロシージャは、新規データベースで表がインスタンス化される前に実行し、新規データベースではローカルの取得プロセスがすでに構成されているため、インスタンス化中に作成される表ごとに SET_TABLE_INSTANTIATION_SCN を実行する必要はありません。

新規データベースがソース・データベースでない場合は、この手順を実行せずに次の手順に進んでください。

2. 新規データベースでエクスポート / インポートを使用して共有オブジェクトをインスタンス化する場合の、ソース・データベースを 1 つ選択します。最初に OBJECT_CONSISTENT エクスポート・パラメータを y に設定してソース・データベースでエクスポートを実行するか、より厳密な一貫性レベルを使用します。次に、新規データベースで、STREAMS_INSTANTIATION インポート・パラメータを y に設定してインポートを実行します。エクスポート / インポートの使用方法は、11-8 ページの「Streams に関連するエクスポート・ユーティリティとインポート・ユーティリティのパラメータ設定」および『Oracle9*i* データベース・ユーティリティ』を参照してください。

3. 手順 2 でインスタンス化のためのエクスポートを実行したソース・データベースを除き、新規データベースのソース・データベースごとに、新規データベースでインスタンス化 SCN を設定します。これらのインスタンス化 SCN の設定は必須です。また、ソース・データベースで行われ、そのデータベースの対応する SCN より後にコミットされる変更のみが、新規データベースで適用されます。

   ソース・データベースごとに、これらのインスタンス化 SCN を次のいずれかの方法で設定できます。

   a. ソース・データベースでメタデータのみのエクスポートを実行し、そのメタデータを新規データベースでインポートします。このインポートによって、ソース・データベースに必須のインスタンス化 SCN が、新規データベースで設定されます。この場合は、新規データベースの共有オブジェクトに、エクスポート時点でソース・データベースとの一貫性があることを確認してください。

      DML 変更のみを共有する場合は、表レベルのエクスポート / インポートで十分です。DDL 変更も共有する場合は、追加の考慮事項があります。メタデータのエクスポート / インポートを実行する方法の詳細は、14-34 ページの「エクスポート / インポートを使用したインスタンス化 SCN の設定」を参照してください。

   b. 新規データベースで、共有オブジェクトについてインスタンス化 SCN を手動で設定します。新規データベースの共有オブジェクトに、対応するインスタンス化 SCN の時点でソース・データベースとの一貫性があることを確認してください。手順については、14-35 ページの「DBMS_APPLY_ADM パッケージを使用したインスタンス化 SCN の設定」を参照してください。

4. 11-33 ページの手順 3 で新規データベースに作成した適用プロセスを起動します。手順については、14-7 ページの「適用プロセスの起動」を参照してください。
5. 11-33 ページの手順 4 で他の各接続先データベースに作成した適用プロセスを起動します。手順については、14-7 ページの「適用プロセスの起動」を参照してください。新規データベースがソース・データベースでない場合は、この手順を実行しないでください。

12

取得プロセスの管理

取得プロセスは、REDO ログ内の変更を取得し、取得した変更を論理変更レコード（LCR）形式で再フォーマットして、LCR を Streams キューにエンキューします。

この章の内容は次のとおりです。

この項で説明する各タスクは、特に明記されていないかぎり、適切な権限を付与されている Streams 管理者が完了する必要があります。

関連項目：

- 第 2 章「Streams の取得プロセス」
- 11-2 ページ「Streams 管理者の構成」

取得プロセスの作成

次のどのプロシージャでも、取得プロセスを作成できます。

- DBMS_STREAMS_ADM.ADD_TABLE_RULES
- DBMS_STREAMS_ADM.ADD_SCHEMA_RULES
- DBMS_STREAMS_ADM.ADD_GLOBAL_RULES
- DBMS_CAPTURE_ADM.CREATE_CAPTURE

DBMS_STREAMS_ADM パッケージの各プロシージャでは、指定した名前の取得プロセスが存在しない場合は取得プロセスが作成され、取得プロセスにルール・セットがない場合はルール・セットが作成されます。また、ルール・セットに表ルール、スキーマ・ルールまたはグローバル・ルールを追加できます。

CREATE_CAPTURE プロシージャでは、取得プロセスは作成されますが、そのルール・セットやルールは作成されません。ただし、CREATE_CAPTURE プロシージャを使用すると、取得プロセスに関連付ける既存のルール・セットと取得プロセスの開始 SCN を指定できます。

取得プロセスを作成する前に、次のタスクを完了する必要があります。

- 11-12 ページの「Streams の取得プロセスを実行するためのデータベースの構成」で説明したタスクを完了します。
- 存在しない場合は、取得プロセスに関連付ける Streams キューを作成します。手順については、13-2 ページの「Streams のキューの作成」を参照してください。

注意： データベース内で最初の取得プロセスを作成する場合は、この作成中にデータ・ディクショナリが複製されるため、時間がかかることがあります。

DBMS_STREAMS_ADM を使用した取得プロセスの作成例

ここでは、DBMS_STREAMS_ADM パッケージの ADD_TABLE_RULES プロシージャを実行して取得プロセスを作成する例について説明します。

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name         => 'hr.employees',
    streams_type       => 'capture',
    streams_name       => 'strm01_capture',
    queue_name         => 'strm01_queue',
    include_dml        => true,
    include_ddl        => true,
    include_tagged_lcr => false);
END;
/
```

このプロシージャを実行すると、次のアクションが実行されます。

- 取得プロセス strm01_capture が作成されます。この取得プロセスが作成されるのは、それが存在しない場合のみです。新規の取得プロセスが作成される場合、このプロシージャでは開始 SCN も作成時点に設定されます。
- 取得プロセスが、既存のキュー strm01_queue に関連付けられます。
- 取得プロセスにルール・セットがない場合は、ルール・セットが作成され、取得プロセスに関連付けられます。このルール・セットでは、SYS.STREAMS$_EVALUATION_CONTEXT 評価コンテキストが使用されます。ルール・セット名は、システムによって指定されます。
- 2 つのルールが作成されます。一方のルールでは取得プロセスで hr.employees 表に対する DML 変更を取得するように指定され、他方のルールでは取得プロセスで hr.employees 表に対する DDL 変更を取得するように指定されます。ルール名は、システムによって指定されます。
- この 2 つのルールが、取得プロセスに関連付けられたルール・セットに追加されます。
- include_tagged_lcr パラメータが false に設定されているため、変更に NULL タグが付いている場合にのみ、取得プロセスで REDO ログ内の変更を取得するように指定されます。この動作は、取得プロセスのシステム作成ルールを介して実行されます。

関連項目：

- 2-19 ページ「取得プロセスの作成」
- 6-3 ページ「システム作成ルール」
- 8-3 ページ「DBMS_STREAMS_ADM パッケージによって作成されるタグとルール」

DBMS_CAPTURE_ADM を使用した取得プロセスの作成例

ここでは、DBMS_CAPTURE_ADM パッケージの CREATE_CAPTURE プロシージャを実行して取得プロセスを作成する例について説明します。

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name     => 'strm01_queue',
    capture_name   => 'strm02_capture',
    rule_set_name => 'strmadmin.strm01_rule_set',
    start_scn      => 829381993);
END;
/
```

このプロシージャを実行すると、次のアクションが実行されます。

- 取得プロセス strm02_capture が作成されます。同じ名前の取得プロセスが存在することは許されません。
- 取得プロセスが、既存のキュー strm01_queue に関連付けられます。
- 取得プロセスが、既存のルール・セット strm01_rule_set に関連付けられます。
- 取得プロセスの開始 SCN として 829381993 が指定されます。

関連項目：

- 2-19 ページ「取得プロセスの作成」
- 2-14 ページ「取得プロセスの開始 SCN、取得済 SCN および適用済 SCN」

取得プロセスの起動

既存の取得プロセスを起動するには、DBMS_CAPTURE_ADM パッケージの START_CAPTURE プロシージャを実行します。たとえば、次のプロシージャでは、取得プロセス strm01_capture が起動されます。

```
BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name => 'strm01_capture');
END;
/
```

取得プロセスのルール・セットの指定

DBMS_CAPTURE_ADM パッケージの ALTER_CAPTURE プロシージャで rule_set_name パラメータを使用すると、既存の取得プロセスに関連付ける既存のルール・セットを指定できます。たとえば、次のプロシージャでは、取得プロセス strm01_capture のルール・セットが strm02_rule_set に設定されます。

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name  => 'strm01_capture',
    rule_set_name => 'strmadmin.strm02_rule_set');
END;
/
```

関連項目：

- 第 5 章「ルール」
- 第 6 章「Streams でのルールの使用方法」

取得プロセスのルール・セットへのルールの追加

既存の取得プロセスのルール・セットにルールを追加するには、次のいずれかのプロシージャを使用して既存の取得プロセスを指定できます。

- DBMS_STREAMS_ADM.ADD_TABLE_RULES
- DBMS_STREAMS_ADM.ADD_SCHEMA_RULES
- DBMS_STREAMS_ADM.ADD_GLOBAL_RULES

ここでは、DBMS_STREAMS_ADM パッケージの ADD_TABLE_RULES プロシージャを実行して取得プロセス strm01_capture のルール・セットにルールを追加する例について説明します。

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name         => 'hr.departments',
    streams_type       => 'capture',
    streams_name       => 'strm01_capture',
    queue_name         => 'strm01_queue',
    include_dml        => true,
    include_ddl        => true);
END;
/
```

このプロシージャを実行すると、次のアクションが実行されます。

- 2 つのルールが作成されます。一方のルールでは取得プロセスで hr.departments 表に対する DML 変更を取得するように指定され、他方のルールでは取得プロセスで hr.departments 表に対する DDL 変更を取得するように指定されます。ルール名は、システムによって指定されます。
- この 2 つのルールが、取得プロセスに関連付けられたルール・セットに追加されます。

関連項目： 6-3 ページ「システム作成ルール」

取得プロセスのルール・セットからのルールの削除

DBMS_STREAMS_ADM パッケージの REMOVE_RULE プロシージャを実行すると、既存の取得プロセスのルール・セットからルールを削除するように指定できます。たとえば、次のプロシージャでは、取得プロセス strm01_capture のルール・セットからルール DEPARTMENTS3 が削除されます。

```
BEGIN
  DBMS_STREAMS_ADM.REMOVE_RULE(
    rule_name        => 'DEPARTMENTS3',
    streams_type     => 'capture',
    streams_name     => 'strm01_capture',
    drop_unused_rule => true);
END;
/
```

この例では、REMOVE_RULE プロシージャの drop_unused_rule パラメータが true に設定されています。これはデフォルト設定です。したがって、削除するルールがどのルール・セットにもなければ、そのルールはデータベースから削除されます。drop_unused_rule パラメータが false に設定されている場合、ルールはルール・セットから削除されますが、データベースからは削除されません。

また、取得プロセスのルール・セットからすべてのルールを削除する必要がある場合は、REMOVE_RULE プロシージャの実行時に rule_name パラメータに NULL を指定します。

注意： 取得プロセスのルール・セットからルールをすべて削除すると、取得プロセスではイベントが取得されません。

取得プロセスのルール・セットの削除

既存の取得プロセスからルール・セットを削除するように指定するには、DBMS_CAPTURE_ADM パッケージの ALTER_CAPTURE プロシージャで remove_rule_set パラメータを true に設定します。たとえば、次のプロシージャでは、取得プロセス strm01_capture からルール・セットが削除されます。

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name    => 'strm01_capture',
    remove_rule_set => true);
END;
/
```

注意： 取得プロセスのルール・セットを削除すると、取得プロセスでは、SYS および SYSTEM スキーマ内のデータベース・オブジェクトを除き、データベース内の全オブジェクトに対するすべてのサポートされる変更が取得されます。

取得プロセスのパラメータの設定

取得プロセスのパラメータを設定するには、DBMS_CAPTURE_ADM パッケージの SET_PARAMETER プロシージャを使用します。取得プロセス・パラメータでは、取得プロセスの動作を制御します。

たとえば、次のプロシージャでは、取得プロセス strm01_capture の parallelism パラメータが 3 に設定されます。

```
BEGIN
  DBMS_CAPTURE_ADM.SET_PARAMETER(
    capture_name => 'strm01_capture',
    parameter    => 'parallelism',
    value        => '3');
END;
/
```

注意：

- parallelism パラメータを設定すると、取得プロセスが自動的に停止され、再起動されます。
- value パラメータは、パラメータ値が数値の場合にも、常に VARCHAR2 として入力されます。

関連項目：

- 2-23 ページ「取得プロセスのパラメータ」
- 取得プロセス・パラメータの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_CAPTURE_ADM.SET_PARAMETER プロシージャの項を参照してください。

ソース・データベースでのサプリメンタル・ロギングの指定

列に対する変更が接続先データベースで正常に適用されるように、ソース・データベースで特定の列のサプリメンタル・ロギングを指定する必要があります。この項では、ソース・データベースでサプリメンタル・ロギングを指定する方法について説明します。

注意： LOB、LONG およびユーザー定義型は、サプリメンタル・ログ・グループに含めることはできません。

関連項目： サプリメンタル・ロギングが必要な場合については、2-10 ページの「Streams 環境内のサプリメンタル・ロギング」を参照してください。

無条件ログ・グループを使用した表サプリメンタル・ロギングの指定

無条件のサプリメンタル・ログ・グループを指定するには、ALTER TABLE 文に ADD SUPPLEMENTAL LOG GROUP 句と ALWAYS 指定を使用して、必要な列を含む REDO ログ・グループを作成する必要があります。必要な場合は、これらの REDO ログ・グループにキー列を含めることができます。

たとえば、次の文では、無条件のログ・グループ log_group_dep_pk に hr.departments 表の主キー列が追加されます。

```
ALTER TABLE hr.departments ADD SUPPLEMENTAL LOG GROUP log_group_dep_pk
  (department_id) ALWAYS;
```

ALWAYS 指定があるため、このログ・グループは無条件のログ・グループになります。

条件付きログ・グループを使用した表サプリメンタル・ロギングの指定

条件付きのサプリメンタル・ログ・グループを指定するには、ALTER TABLE 文に ADD SUPPLEMENTAL LOG GROUP 句を使用して、必要な列を含む REDO ログ・グループを作成する必要があります。ログ・グループを条件付きにするには、ALWAYS 指定を含めないでください。

たとえば、hr.jobs 表の min_salary および max_salary 列が、接続先データベースで競合解消用の列リストに含まれているとします。次の文では、min_salary および max_salary 列が条件付きのログ・グループ log_group_jobs_cr に追加されます。

```
ALTER TABLE hr.jobs ADD SUPPLEMENTAL LOG GROUP log_group_jobs_cr
  (min_salary, max_salary);
```

サプリメンタル・ログ・グループの削除

条件付きまたは無条件のサプリメンタル・ログ・グループを削除するには、ALTER TABLE 文に DROP SUPPLEMENTAL LOG GROUP 句を使用します。たとえば、サプリメンタル・ログ・グループ log_group_jobs_cr を削除するには、次の文を実行します。

```
ALTER TABLE hr.jobs DROP SUPPLEMENTAL LOG GROUP log_group_jobs_cr;
```

キー列のデータベース・サプリメンタル・ロギングの指定

ソース・データベース内のすべての主キー列と一意キー列について、サプリメンタル・ロギングを指定するオプションもあります。データベース全体に対する変更を取得するように取得プロセスを構成する場合は、このオプションを選択できます。ソース・データベース内のすべての主キー列と一意キー列のサプリメンタル・ロギングを指定するには、次の SQL 文を発行します。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY, UNIQUE INDEX) COLUMNS;
```

主キー列と一意キー列がすべてのソース・データベースと接続先データベースで同一の場合は、このコマンドをソース・データベースで実行すると、すべての接続先データベースで主キー列と一意キー列に必要なサプリメンタル・ロギングが提供されます。

キー列のデータベース・サプリメンタル・ロギングの削除

ソース・データベース内のすべての主キー列と一意キー列のサプリメンタル・ロギングを削除するには、次の SQL 文を発行します。

```
ALTER DATABASE DROP SUPPLEMENTAL LOG DATA;
```

注意： キー列のデータベース・サプリメンタル・ロギングを削除しても、既存の表レベルのサプリメンタル・ログ・グループには影響しません。

取得プロセスの開始 SCN の設定

既存の取得プロセスの開始 SCN を指定するには、DBMS_CAPTURE_ADM パッケージの ALTER_CAPTURE プロシージャで start_scn パラメータを使用します。SCN には、データベースの最初の取得プロセスを作成した時点以後の値を指定する必要があります。データベースの最初の取得プロセスは、変更対象でも変更対象外でもかまいません。無効な SCN を指定するとエラーが戻されます。通常、取得プロセスの開始 SCN をリセットするのは、取得プロセスからの変更について接続先データベースの 1 つで Point-in-Time リカバリを実行する必要がある場合です。

たとえば、次のプロシージャでは、取得プロセス strm01_capture の開始 SCN が 750338948 に設定されます。

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name => 'strm01_capture',
    start_scn    => 750338948);
END;
/
```

関連項目：

- 2-14 ページ「取得プロセスの開始 SCN、取得済 SCN および適用済 SCN」
- 16-27 ページ「接続先データベースにおけるデータベースの Point-in-Time リカバリの実行」

ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備

DBMS_CAPTURE_ADM パッケージの次のプロシージャでは、インスタンス化のためにデータベース・オブジェクトの準備が行われます。

- PREPARE_TABLE_INSTANTIATION では、1 つの表がインスタンス化のために準備されます。
- PREPARE_SCHEMA_INSTANTIATION では、スキーマ内の全データベース・オブジェクトと、将来そのスキーマに追加される全データベース・オブジェクトが、インスタンス化のために準備されます。
- PREPARE_GLOBAL_INSTANTIATION では、データベース内の全オブジェクトと、将来そのデータベースに追加される全オブジェクトが、インスタンス化のために準備されます。

インスタンス化の準備対象となる 1 つ以上のデータベース・オブジェクトを長時間実行トランザクションで変更中に、前述のプロシージャの 1 つを実行すると、そのプロシージャは長時間実行トランザクションが完了するまで待機してから、最小 SCN を記録します。

たとえば、インスタンス化のために hr.regions 表を準備するには、次のプロシージャを実行します。

```
BEGIN
  DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
    table_name  => 'hr.regions');
END;
/
```

関連項目：

- 2-12 ページ「インスタンス化」
- 3-24 ページ「伝播用の Streams データ・ディクショナリ」
- 4-31 ページ「適用プロセス用の Streams データ・ディクショナリ」
- 11-14 ページ「取得ベースの Streams 環境の構成」

ソース・データベースでのインスタンス化の準備の強制終了

DBMS_CAPTURE_ADM パッケージの次のプロシージャでは、インスタンス化の準備が強制終了されます。

- ABORT_TABLE_INSTANTIATION では、PREPARE_TABLE_INSTANTIATION の効果が逆転します。
- ABORT_SCHEMA_INSTANTIATION では、PREPARE_SCHEMA_INSTANTIATION の効果が逆転します。
- ABORT_GLOBAL_INSTANTIATION では、PREPARE_GLOBAL_INSTANTIATION の効果が逆転します。

これらのプロシージャでは、関連データベース・オブジェクトの潜在的なインスタンス化に関係するデータ・ディクショナリ情報が削除されます。

たとえば、hr.regions 表のインスタンス化の準備を強制終了するには、次のプロシージャを実行します。

```
BEGIN
  DBMS_CAPTURE_ADM.ABORT_TABLE_INSTANTIATION(
    table_name  => 'hr.regions');
END;
/
```

変更が取得されるデータベースの DBID の変更

通常、あるデータベースが別のデータベースのクローンである場合、データベース管理者はそのデータベースの DBID を変更します。V$DATABASE 動的パフォーマンス・ビューで DBID 列を問い合せると、データベースの DBID を確認できます。

取得プロセスが DBID を変更したデータベースによって生成される変更を取得している場合は、次の手順を完了します。

1. データベースを停止します。
2. STARTUP RESTRICT を使用し、RESTRICTED SESSION を有効化してデータベースを再起動します。
3. 取得プロセスを削除します。
4. データベースで ALTER SYSTEM SWITCH LOGFILE 文を実行します。
5. データベースですでに変更が取得されている場合は、このソース・データベースで発生した変更を適用するすべての接続先データベースで、データを手動で再同期化します。まだデータベースで変更が取得されていない場合は、この手順は不要です。
6. 必要に応じて取得プロセスを再作成します。
7. ALTER SYSTEM DISABLE RESTRICTED SESSION 文を使用し、RESTRICTED SESSION を無効化します。

関連項目： DBNEWID ユーティリティを使用してデータベースの DBID を変更する方法の詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。

変更が取得されるログ順序番号のリセット

通常、データベース管理者は、Point-in-Time リカバリ中にデータベースのログ順序番号をリセットします。ALTER DATABASE OPEN RESETLOGS 文は、ログ順序番号をリセットする文の一例です。データベースのログ順序番号をリセットすると、既存のローカルの取得プロセスは使用禁止になります。

取得プロセスがログ順序番号をリセットしたデータベースによって生成される変更を取得している場合は、次の手順を完了します。

1. 取得プロセスを削除します。
2. このソース・データベースで発生した変更を適用するすべての接続先データベースで、データを手動で再同期化します。
3. 必要に応じて取得プロセスを再作成します。

関連項目： Point-in-Time リカバリの詳細は、『Oracle9*i* バックアップおよびリカバリ概要』を参照してください。

取得プロセスの停止

既存の取得プロセスを停止するには、DBMS_CAPTURE_ADM パッケージの STOP_CAPTURE プロシージャを実行します。たとえば、次のプロシージャでは、取得プロセス strm01_capture が停止されます。

```
BEGIN
  DBMS_CAPTURE_ADM.STOP_CAPTURE(
    capture_name => 'strm01_capture');
END;
/
```

取得プロセスの削除

既存の取得プロセスを削除するには、DBMS_CAPTURE_ADM パッケージの DROP_CAPTURE プロシージャを実行します。たとえば、次のプロシージャでは、取得プロセス strm01_capture が削除されます。

```
BEGIN
  DBMS_CAPTURE_ADM.DROP_CAPTURE(
    capture_name => 'strm01_capture');
END;
/
```

取得プロセスは、削除する前に停止する必要があります。

13

ステージングと伝播の管理

この章では、Streams のキュー、伝播およびメッセージ環境の管理手順について説明します。

この章の内容は次のとおりです。

- Streams のキューの管理
- Streams の伝播および伝播ジョブの管理
- Streams メッセージ環境の管理

この項で説明する各タスクは、特に明記されていないかぎり、適切な権限を付与されている Streams 管理者が完了する必要があります。

関連項目：

- 第 3 章「Streams のステージングと伝播」
- 11-2 ページ「Streams 管理者の構成」

Streams のキューの管理

Streams のキューでは、SYS.AnyData 型のペイロードを伴うイベントがステージングされます。したがって、Streams のキューでは、SYS.AnyData ラッパーにラップされていれば、ほぼすべての型のペイロードを伴うイベントをステージングできます。Streams の取得プロセスと適用プロセスはそれぞれ 1 つの Streams キューに関連付けられ、Streams の伝播はそれぞれ 1 つの Streams ソース・キューと 1 つの Streams 宛先キューに関連付けられます。

この項では、Streams のキューに関連する次のタスクの手順について説明します。

- Streams のキューの作成
- 保護キューでのユーザー操作の有効化
- 保護キューでのユーザー操作の無効化
- Streams キューの削除

Streams のキューの作成

Streams のキューを作成するには、DBMS_STREAMS_ADM パッケージの SET_UP_QUEUE プロシージャを使用します。このプロシージャを使用すると、作成する Streams キューについて次の要素を指定できます。

- キューのキュー表
- キュー表の STORAGE 句
- キュー名
- キューの保護キュー・ユーザーとして構成され、キューに対する ENQUEUE および DEQUEUE 権限が付与されるキュー・ユーザー
- キューに関するコメント

このプロシージャでは、保護キューとトランザクション型のキューを兼ねるキューが作成され、そのキューが起動されます。

たとえば、キュー表 strm01_queue_table を持つ Streams キュー strm01_queue を作成し、このキューとの間でイベントのエンキューとデキューを行うために必要な権限を hr ユーザーに付与するには、次のプロシージャを実行します。

```
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table  => 'strm01_queue_table',
    queue_name   => 'strm01_queue',
    queue_user   => 'hr');
END;
/
```

また、DBMS_AQADM パッケージのプロシージャを使用すると、SYS.AnyData キューを作成できます。

関連項目：

- DBMS_AQADM パッケージのプロシージャを使用して SYS.AnyData キューを作成する例については、13-18 ページの「SYS.AnyData ラッパーでのユーザー・メッセージ・ペイロードのラップ」を参照してください。
- 3-21 ページ「保護キュー」
- 3-23 ページ「トランザクション型のキューと非トランザクション型のキュー」

保護キューでのユーザー操作の有効化

ユーザーが保護キューについてエンキューやデキューなどのキュー操作を実行できるように、そのユーザーをキューの保護キュー・ユーザーとして構成する必要があります。DBMS_STREAMS_ADM パッケージの SET_UP_QUEUE プロシージャを使用して保護キューを作成すると、キュー所有者と queue_user パラメータで指定したユーザーが自動的にキューの保護ユーザーとして構成されます。他のユーザーに対してキュー操作の実行を許可する必要がある場合は、そのユーザーを次のいずれかの方法で構成できます。

- SET_UP_QUEUE を実行して queue_user を指定します。キューが存在する場合、その作成はスキップされますが、新規キュー・ユーザーを指定した場合はそのユーザーが構成されます。
- ユーザーをエージェントに手動で関連付けます。

次の例に、ユーザーをエージェントに手動で関連付ける手順を示します。oe ユーザーに対して、13-2 ページの「Streams のキューの作成」で作成した strm01_queue に対するキュー操作の実行を許可する必要があるとします。次の手順に従って、oe ユーザーを strm01_queue の保護キュー・ユーザーとして構成します。

1. エージェントを作成してユーザーを変更できる管理ユーザーとして接続します。
2. 次のように入力してエージェントを作成します。

```
EXEC DBMS_AQADM.CREATE_AQ_AGENT(agent_name => 'strm01_queue_agent');
```

3. ユーザーにキューからイベントをデキューする操作を許可する必要がある場合は、エージェントを保護キューのサブスクライバにします。

```
DECLARE
  subscriber SYS.AQ$_AGENT;
BEGIN
  subscriber :=  SYS.AQ$_AGENT('strm01_queue_agent', NULL, NULL);
  SYS.DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name          =>  'strmadmin.strm01_queue',
    subscriber          =>  subscriber,
    rule                =>  NULL,
    transformation      =>  NULL);
END;
/
```

4. ユーザーをエージェントに関連付けます。

```
BEGIN
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name  => 'strm01_queue_agent',
    db_username => 'oe');
END;
/
```

5. ユーザーに付与されていない場合は、DBMS_AQ パッケージの EXECUTE 権限を付与します。

```
GRANT EXECUTE ON DBMS_AQ TO oe;
```

前述の手順を完了すると、oe ユーザーは strm01_queue キューの保護ユーザーとなり、このキューの操作を実行できます。ただし、エンキュー権限やデキュー権限など、キュー操作を実行するための特定の権限もユーザーに付与する必要があります。

関連項目：

- 3-21 ページ「保護キュー」
- AQ エージェントと DBMS_AQADM パッケージの使用方法の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

保護キューでのユーザー操作の無効化

ユーザーに対して、次のような理由で保護キューのキュー操作を禁止する必要が生じる場合があります。

- 取得プロセスを削除したが、その取得プロセスで使用されていたキューを削除しておらず、取得ユーザーであったユーザーには残りの保護キューの操作を禁止する必要がある場合。
- 適用プロセスを削除したが、その適用プロセスで使用されていたキューを削除しておらず、適用ユーザーであったユーザーには残りの保護キューの操作を禁止する必要がある場合。
- DBMS_APPLY_ADM パッケージの ALTER_APPLY プロシージャを使用して適用プロセスの apply_user を変更しており、変更前の apply_user には適用プロセスのキューの操作を禁止する必要がある場合。
- 13-3 ページの「保護キューでのユーザー操作の有効化」で説明した手順に従って、ユーザーに保護キューの操作を許可したが、このユーザーに保護キューの操作を禁止する必要がある場合。

保護キュー・ユーザーを無効化するには、ユーザーからキューに対する ENQUEUE および DEQUEUE 権限を取り消す方法と、DBMS_AQADM パッケージの DISABLE_DB_ACCESS プロシージャを実行する方法があります。たとえば、oe ユーザーに対して、13-2 ページの「Streams のキューの作成」で作成した strm01_queue のキュー操作を禁止する必要があるとします。

> **注意：** 複数の保護キューにエージェントを使用している場合は、そのエージェントに対して DISABLE_DB_ACCESS を実行すると、ユーザーはこれらのキューすべての操作を実行できなくなります。

1. 次のプロシージャを実行し、oe ユーザーに対して保護キュー strm01_queue のキュー操作を禁止します。

```
BEGIN
  DBMS_AQADM.DISABLE_DB_ACCESS(
    agent_name  => 'strm01_queue_agent',
    db_username => 'oe');
END;
/
```

2. 不要になったエージェントは削除できます。

```
BEGIN
  DBMS_AQADM.DROP_AQ_AGENT(
    agent_name  => 'strm01_queue_agent');
END;
/
```

3. キューに対するユーザーの権限が不要になった場合は、その権限を取り消します。

```
BEGIN
  DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE (
   privilege   => 'ALL',
   queue_name  => 'strmadmin.strm01_queue',
   grantee     => 'oe');
END;
/
```

関連項目：

- 3-21 ページ「保護キュー」
- AQ エージェントと DBMS_AQADM パッケージの使用方法の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

Streams キューの削除

既存の Streams キューを削除するには、型付きのキューを削除する場合と同じ操作を実行します。Streams キューは、次のような方法で削除できます。

- DBMS_AQADM パッケージの STOP_QUEUE、DROP_QUEUE および DROP_QUEUE_TABLE プロシージャを使用するか、Streams キューを含むキュー表で force パラメータを TRUE に設定した DROP_QUEUE_TABLE プロシージャのみを使用して明示的に削除します。
- CASCADE オプションを使用して、キューを所有するユーザーを削除します。

Streams キューを削除すると、Streams キューから例外キューに移動されたすべてのエラーとなったトランザクションは自動的に削除されます。

関連項目： キューを削除する方法の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

Streams の伝播および伝播ジョブの管理

伝播によって、Streams のソース・キューから Streams の宛先キューにイベントが伝播されます。この項では、次のタスクの手順について説明します。

- 伝播の作成
- 伝播ジョブの有効化
- 伝播ジョブのスケジューリング
- 伝播ジョブのスケジュールの変更
- 伝播ジョブのスケジュール解除
- 伝播のルール・セットの指定
- 伝播のルール・セットへのルールの追加
- 伝播のルール・セットからのルールの削除
- 伝播のルール・セットの削除
- 伝播ジョブの無効化
- 伝播の削除

また、Streams の伝播は、Oracle Advanced Queuing（AQ）の機能を使用して管理することもできます。

> **関連項目：** AQ の機能を使用して伝播を管理する方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

伝播の作成

次のどのプロシージャでも、伝播を作成できます。

- `DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES`
- `DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES`
- `DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES`
- `DBMS_PROPAGATION_ADM.CREATE_PROPAGATION`

`DBMS_STREAMS_ADM` パッケージの各プロシージャでは、指定した名前の伝播が存在しない場合は伝播が作成され、伝播にルール・セットがない場合はルール・セットが作成されます。また、ルール・セットに表ルール、スキーマ・ルールまたはグローバル・ルールを追加できます。`CREATE_PROPAGATION` プロシージャでは、伝播は作成されますが、そのルール・セットまたはルールは作成されません。すべての伝播は、作成すると自動的に起動します。

伝播を作成する前に、次のタスクを完了する必要があります。

- 存在しない場合は、伝播のソース・キューと宛先キューを作成します。手順については、13-2 ページの「Streams のキューの作成」を参照してください。
- ソース・キューを含むデータベースと宛先キューを含むデータベースの間のデータベース・リンクを作成します。詳細は、11-13 ページの「ネットワーク接続性とデータベース・リンクの構成」を参照してください。

DBMS_STREAMS_ADM を使用した伝播の作成例

ここでは、DBMS_STREAMS_ADM パッケージの ADD_TABLE_RULES プロシージャを実行して伝播を作成する例について説明します。

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              => 'hr.departments',
    streams_name            => 'strm01_propagation',
    source_queue_name       => 'strmadmin.strm01_queue',
    destination_queue_name  => 'strmadmin.strm02_queue@dbs2.net',
    include_dml             => true,
    include_ddl             => true,
    include_tagged_lcr      => false,
    source_database         => 'dbs1.net' );
END;
/
```

このプロシージャを実行すると、次のアクションが実行されます。

- 伝播 strm01_propagation が作成されます。この伝播が作成されるのは、それが存在しない場合のみです。
- 伝播で現行のデータベース内の strm01_queue から dbs2.net データベース内の strm02_queue に LCR を伝播するように指定されます。
- destination_queue_name パラメータに @dbs2.net が指定されているため、伝播で dbs2.net データベース・リンクを使用して LCR を伝播させるように指定されます。
- 伝播にルール・セットがない場合は、ルール・セットが作成され、伝播に関連付けられます。このルール・セットでは、評価コンテキスト SYS.STREAMS$_EVALUATION_CONTEXT が使用されます。ルール・セット名は、システムによって指定されます。
- 2 つのルールが作成されます。一方のルールでは伝播で hr.departments 表に対する DML 変更の結果を含む行 LCR を伝播させるように指定され、他方のルールでは伝播で hr.departments 表に対する変更を含む DDL LCR を伝播させるように指定されます。ルール名は、システムによって指定されます。
- この 2 つのルールが、伝播に関連付けられたルール・セットに追加されます。

- include_tagged_lcr パラメータが false に設定されているため、NULL タグがある場合にのみ伝播で LCR を伝播させるように指定されます。 この動作は、伝播のシステム作成ルールを介して実行されます。
- 伝播させる LCR のソース・データベースとして dbs1.net が指定されます。このデータベースは、現行のデータベースでなくてもかまいません。
- 指定したデータベース・リンクに伝播ジョブが存在しない場合は、伝播ジョブが作成されます。

関連項目：

- 3-4 ページ「キュー間でのイベントの伝播」
- 6-3 ページ「システム作成ルール」
- 8-3 ページ「DBMS_STREAMS_ADM パッケージによって作成されるタグとルール」

DBMS_PROPAGATION_ADM を使用した伝播の作成例

ここでは、DBMS_PROPAGATION_ADM パッケージの CREATE_PROPAGATION プロシージャを実行して伝播を作成する例について説明します。

```
BEGIN
  DBMS_PROPAGATION_ADM.CREATE_PROPAGATION(
    propagation_name   => 'strm02_propagation',
    source_queue       => 'strmadmin.strm03_queue',
    destination_queue  => 'strmadmin.strm04_queue',
    destination_dblink => 'dbs2.net',
    rule_set_name      => 'strmadmin.strm01_rule_set');
END;
/
```

このプロシージャを実行すると、次のアクションが実行されます。

- 伝播 strm02_propagation が作成されます。同じ名前の伝播が存在することはできません。
- 伝播で現行のデータベース内の strm03_queue から dbs2.net データベース内の strm04_queue にイベントを伝播するように指定されます。伝播されるイベントは、ルール・セット内のルールに応じて取得イベントまたはユーザー・エンキュー・イベント、あるいはその両方です。
- 伝播で dbs2.net データベース・リンクを使用してイベントを伝播させるように指定されます。

- 伝播が、既存のルール・セット strm01_rule_set に関連付けられます。
- 指定したデータベース・リンクに伝播ジョブが存在しない場合は、伝播ジョブが作成されます。

関連項目：

- 3-3 ページ「取得イベントとユーザー・エンキュー・イベント」
- 3-4 ページ「キュー間でのイベントの伝播」

伝播ジョブの有効化

デフォルトでは、伝播ジョブは作成時に有効化されます。伝播ジョブを無効化した後に有効化する場合は、DBMS_AQADM パッケージの ENABLE_PROPAGATION_SCHEDULE プロシージャを使用します。

たとえば、dbs2.net データベース・リンクを使用して strmadmin.strm01_queue ソース・キューからイベントを伝播させる伝播ジョブを有効化するには、次のプロシージャを実行します。

```
BEGIN
  DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
    queue_name  => 'strmadmin.strm01_queue',
    destination => 'dbs2.net');
END;
/
```

注意： このタスクを完了すると、ソース・キューから dbs2.net データベース・リンクを使用するすべての宛先キューにイベントを伝播するすべての伝播に影響します。

関連項目：

- ENABLE_PROPAGATION_SCHEDULE プロシージャの使用方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。
- 3-19 ページ「伝播ジョブ」

伝播ジョブのスケジューリング

DBMS_AQADM パッケージの SCHEDULE_PROPAGATION プロシージャを使用すると、伝播ジョブをスケジュールできます。伝播ジョブに問題がある場合は、そのジョブをスケジュール解除してから再スケジュールすると、問題を解決できることがあります。

たとえば、次のプロシージャでは、dbs2.net データベース・リンクを使用して strmadmin.strm01_queue ソース・キューからイベントを伝播させる伝播ジョブがスケジュールされます。

```
BEGIN
  DBMS_AQADM.SCHEDULE_PROPAGATION(
   queue_name  => 'strmadmin.strm01_queue',
   destination => 'dbs2.net');
END;
/
```

注意： このタスクを完了すると、ソース・キューから dbs2.net データベース・リンクを使用するすべての宛先キューにイベントを伝播するすべての伝播に影響します。

関連項目：

- 13-13 ページ「伝播ジョブのスケジュール解除」
- SCHEDULE_PROPAGATION プロシージャの使用方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。
- 3-19 ページ「伝播ジョブ」

伝播ジョブのスケジュールの変更

DBMS_AQADM パッケージの ALTER_PROPAGATION_SCHEDULE プロシージャを使用すると、既存の伝播ジョブのスケジュールを変更できます。

たとえば、dbs2.net データベース・リンクを使用して strmadmin.strm01_queue ソース・キューからイベントを伝播させる伝播ジョブについて、スケジュールを変更する必要があるとします。次のプロシージャでは、イベントを 15 分（900 秒）ごとに 300 秒の継続期間で伝播させ、25 秒待機してから、完全に伝播されたキュー内の新規イベントを伝播させるように、伝播ジョブが設定されます。

```
BEGIN
  DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
   queue_name  => 'strmadmin.strm01_queue',
   destination => 'dbs2.net',
   duration    => 300,
   next_time   => 'SYSDATE + 900/86400',
   latency     => 25);
END;
/
```

注意： このタスクを完了すると、ソース・キューから dbs2.net データベース・リンクを使用するすべての宛先キューにイベントを伝播するすべての伝播に影響します。

関連項目：

- ALTER_PROPAGATION_SCHEDULE プロシージャの使用方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。
- 3-19 ページ「伝播ジョブ」

伝播ジョブのスケジュール解除

DBMS_AQADM パッケージの UNSCHEDULE_PROPAGATION プロシージャを使用すると、伝播ジョブをスケジュール解除できます。伝播ジョブに問題がある場合は、そのジョブをスケジュール解除してから再スケジュールすると、問題を解決できることがあります。

たとえば、次のプロシージャでは、dbs2.net データベース・リンクを使用して strmadmin.strm01_queue ソース・キューからイベントを伝播させる伝播ジョブがスケジュール解除されます。

```
BEGIN
  DBMS_AQADM.UNSCHEDULE_PROPAGATION(
   queue_name  => 'strmadmin.strm01_queue',
   destination => 'dbs2.net');
END;
/
```

注意： このタスクを完了すると、ソース・キューから dbs2.net データベース・リンクを使用するすべての宛先キューにイベントを伝播するすべての伝播に影響します。

関連項目：

- 13-11 ページ「伝播ジョブのスケジューリング」
- SCHEDULE_PROPAGATION プロシージャの使用方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。
- 3-19 ページ「伝播ジョブ」

伝播のルール・セットの指定

伝播に関連付けるルール・セットを指定するには、DBMS_PROPAGATION_ADM パッケージの ALTER_PROPAGATION プロシージャで rule_set_name パラメータを使用します。たとえば、次のプロシージャでは、伝播 strm01_propagation のルール・セットが strm02_rule_set に設定されます。

```
BEGIN
  DBMS_PROPAGATION_ADM.ALTER_PROPAGATION(
    propagation_name  => 'strm01_propagation',
    rule_set_name     => 'strmadmin.strm02_rule_set');
END;
/
```

関連項目：

- 第 5 章「ルール」
- 第 6 章「Streams でのルールの使用方法」

伝播のルール・セットへのルールの追加

伝播のルール・セットにルールを追加する場合は、次のいずれかのプロシージャを実行できます。

- DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES
- DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES
- DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES

ここでは、DBMS_STREAMS_ADM パッケージの ADD_TABLE_RULES プロシージャを実行して、伝播 strm01_propagation のルール・セットにルールを追加する例について説明します。

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              => 'hr.locations',
    streams_name            => 'strm01_propagation',
    source_queue_name       => 'strmadmin.strm01_queue',
    destination_queue_name  => 'strmadmin.strm02_queue@dbs2.net',
    include_dml             => true,
    include_ddl             => true,
    source_database         => 'dbs1.net' );
END;
/
```

このプロシージャを実行すると、次のアクションが実行されます。

- 伝播 strm01_propagation が作成されます。この伝播が作成されるのは、それが存在しない場合のみです。
- 伝播で現行のデータベース内の strm01_queue から dbs2.net データベース内の strm02_queue に LCR を伝播するように指定されます。
- destination_queue_name パラメータに @dbs2.net が指定されているため、伝播で dbs2.net データベース・リンクを使用して LCR を伝播させるように指定されます。
- 2 つのルールが作成されます。一方のルールでは伝播で hr.locations 表に対する DML 変更の結果を含む行 LCR を伝播させるように指定され、他方のルールでは伝播で hr.locations 表に対する変更を含む DDL LCR を伝播させるように指定されます。ルール名は、システムによって指定されます。

- この 2 つのルールが、伝播に関連付けられたルール・セットに追加されます。
- 伝播させる LCR のソース・データベースとして dbs1.net が指定されます。このデータベースは、現行のデータベースでなくてもかまいません。

関連項目：

- 3-4 ページ「キュー間でのイベントの伝播」
- 6-3 ページ「システム作成ルール」

伝播のルール・セットからのルールの削除

既存の伝播のルール・セットからルールを削除するように指定するには、DBMS_STREAMS_ADM パッケージの REMOVE_RULE プロシージャを実行します。たとえば、次のプロシージャでは、伝播 strm01_propagation のルール・セットからルール DEPARTMENTS3 が削除されます。

```
BEGIN
  DBMS_STREAMS_ADM.REMOVE_RULE(
    rule_name        => 'DEPARTMENTS3',
    streams_type     => 'propagation',
    streams_name     => 'strm01_propagation',
    drop_unused_rule => true);
END;
/
```

この例では、REMOVE_RULE プロシージャの drop_unused_rule パラメータが true に設定されています。これはデフォルト設定です。したがって、削除するルールがどのルール・セットにもなければ、そのルールはデータベースから削除されます。drop_unused_rule パラメータが false に設定されている場合、ルールはルール・セットから削除されますが、データベースからは削除されません。

また、伝播のルール・セットからすべてのルールを削除する必要がある場合は、REMOVE_RULE プロシージャの実行時に rule_name パラメータに NULL を指定します。

注意： 伝播のルール・セットからルールをすべて削除すると、その伝播ではソース・キューから宛先キューにイベントが伝播されなくなります。

伝播のルール・セットの削除

伝播からルール・セットを削除するように指定するには、DBMS_PROPAGATION_ADM パッケージの ALTER_PROPAGATION プロシージャで rule_set_name パラメータを NULL に設定します。たとえば、次のプロシージャでは、伝播 strm01_propagation からルール・セットが削除されます。

```
BEGIN
  DBMS_PROPAGATION_ADM.ALTER_PROPAGATION(
    propagation_name => 'strm01_propagation',
    rule_set_name    => NULL);
END;
/
```

注意： 伝播のルール・セットを削除すると、その伝播ではソース・キュー内のすべてのイベントが宛先キューに伝播されます。

伝播ジョブの無効化

伝播ジョブを停止するには、DBMS_AQADM パッケージの DISABLE_PROPAGATION_SCHEDULE プロシージャを使用します。

たとえば、dbs2.net データベース・リンクを使用して strmadmin.strm01_queue ソース・キューからイベントを伝播させる伝播ジョブを停止するには、次のプロシージャを実行します。

```
BEGIN
  DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
    queue_name  => 'strmadmin.strm01_queue',
    destination => 'dbs2.net');
END;
/
```

注意：

- このタスクを完了すると、ソース・キューから dbs2.net データベース・リンクを使用するすべての宛先キューにイベントを伝播するすべての伝播に影響します。
- DISABLE_PROPAGATION_SCHEDULE を実行すると、伝播ジョブが即時に無効化されます。現行の継続時間の終了は待機されません。

関連項目： DISABLE_PROPAGATION_SCHEDULE プロシージャの使用方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

伝播の削除

既存の伝播を削除するには、DBMS_PROPAGATION_ADM パッケージの DROP_PROPAGATION プロシージャを実行します。たとえば、次のプロシージャでは、伝播 strm01_propagation が削除されます。

```
BEGIN
  DBMS_PROPAGATION_ADM.DROP_PROPAGATION(
    propagation_name => 'strm01_propagation');
END;
/
```

注意： 伝播を削除すると、その伝播で使用されていた伝播ジョブを使用する伝播が他にない場合、その伝播ジョブは自動的に削除されます。

Streams メッセージ環境の管理

Streams では、SYS.AnyData 型のキューを使用したメッセージングが可能です。これらのキューでは、SYS.AnyData 型のペイロードを伴うユーザー・メッセージがステージングされます。また、SYS.AnyData ペイロードは、様々なデータ型のペイロードのラッパーとして使用できます。

この項では、次のタスクの手順について説明します。

- SYS.AnyData ラッパーでのユーザー・メッセージ・ペイロードのラップ
- SYS.AnyData キューと型付きのキューの間でのメッセージの伝播

注意： この項の例は、各データベースで Streams 管理者を構成している場合を想定しています。

関連項目：

- Streams でのメッセージングの概要は、3-10 ページの「SYS.AnyData 型のキューとユーザー・メッセージ」を参照してください。
- 11-2 ページ「Streams 管理者の構成」
- 第 19 章「Streams メッセージングの例」
- AQ の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。
- SYS.AnyData 型の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

SYS.AnyData ラッパーでのユーザー・メッセージ・ペイロードのラップ

ほぼすべての型のペイロードは、SYS.AnyData ペイロードでラップできます。ここでは、SYS.AnyData キューとの間でメッセージのエンキューとデキューを行う例について説明します。

ペイロードを SYS.AnyData ペイロードにラップしてエンキューする例

次の手順に、SYS.AnyData ペイロードに様々な型のペイロードをラップする方法を示します。

1. dbs1.net データベースでユーザーの作成、権限の付与、表領域の作成およびユーザーの変更を行うことができる管理ユーザーとして接続します。
2. oe ユーザーが DBMS_AQ パッケージの ENQUEUE および DEQUEUE プロシージャを実行できるように、DBMS_AQ パッケージの EXECUTE 権限を付与します。

   ```
   GRANT EXECUTE ON DBMS_AQ TO oe;
   ```

3. 次の例のように、Streams 管理者として接続します。

   ```
   CONNECT strmadmin/strmadminpw@dbs1.net
   ```

4. 存在しない場合は、SYS.AnyData キューを作成します。

   ```
   BEGIN
     DBMS_STREAMS_ADM.SET_UP_QUEUE(
       queue_table  => 'oe_q_table_any',
       queue_name   => 'oe_q_any',
       queue_user   => 'oe');
   END;
   /
   ```

   oe ユーザーは、oe_q_any キューの保護キュー・ユーザーとして自動的に構成され、このキューに対する ENQUEUE および DEQUEUE 権限が付与されます。

5. 次のように入力してエージェントを作成します。

   ```
   EXEC DBMS_AQADM.CREATE_AQ_AGENT(agent_name => 'local_agent');
   ```

6. oe_q_any キューにサブスクライバを追加します。このサブスクライバによって、イベントの明示的なデキューが実行されます。

```
DECLARE
  subscriber SYS.AQ$_AGENT;
BEGIN
  subscriber :=  SYS.AQ$_AGENT('LOCAL_AGENT', NULL, NULL);
  SYS.DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name  =>  'strmadmin.oe_q_any',
    subscriber  =>  subscriber);
END;
/
```

7. oe ユーザーを local_agent エージェントに関連付けます。

```
BEGIN
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name  => 'local_agent',
    db_username => 'oe');
END;
/
```

8. oe ユーザーとして接続します。

```
CONNECT oe/oe@dbs1.net
```

9. 入力パラメータとして SYS.AnyData 型のオブジェクトを取り、ペイロードを含むメッセージを既存の SYS.AnyData キューにエンキューするプロシージャを作成します。

```
CREATE OR REPLACE PROCEDURE oe.enq_proc (payload SYS.AnyData)
IS
  enqopt     DBMS_AQ.ENQUEUE_OPTIONS_T;
  mprop      DBMS_AQ.MESSAGE_PROPERTIES_T;
  enq_msgid  RAW(16);
BEGIN
  mprop.SENDER_ID := SYS.AQ$_AGENT('LOCAL_AGENT', NULL, NULL);
  DBMS_AQ.ENQUEUE(
    queue_name          =>  'strmadmin.oe_q_any',
    enqueue_options     =>  enqopt,
    message_properties  =>  mprop,
    payload             =>  payload,
    msgid               =>  enq_msgid);
END;
/
```

10. 適切な Convert*data_type* ファンクションを指定して、手順 9 で作成したプロシージャを実行します。次のコマンドでは、様々な型のメッセージがエンキューされます。

    VARCHAR2 型 :

```
EXEC oe.enq_proc(SYS.AnyData.ConvertVarchar2('Chemicals - SW'));
COMMIT;
```

    NUMBER 型 :

```
EXEC oe.enq_proc(SYS.AnyData.ConvertNumber('16'));
COMMIT;
```

    ユーザー定義型 :

```
BEGIN
  oe.enq_proc(SYS.AnyData.ConvertObject(oe.cust_address_typ(
    '1646 Brazil Blvd','361168','Chennai','Tam', 'IN')));
END;
/
COMMIT;
```

    関連項目 : これらのエンキューされたメッセージの内容を表示する方法については、17-13 ページの「キュー内のユーザー・エンキュー・イベントの内容の表示」を参照してください。

SYS.AnyData ペイロードにラップされているペイロードをデキューする例

次の手順に、SYS.AnyData ペイロードにラップされているペイロードをデキューする方法を示します。この例では、13-18 ページの「ペイロードを SYS.AnyData ペイロードにラップしてエンキューする例」の手順を完了していることを想定しています。

メッセージをデキューするには、メッセージのコンシューマを知る必要があります。キュー内のメッセージのコンシューマを検索するには、そのキューの所有者として接続し、AQ$*queue_table_name* を問い合せます。この場合、*queue_table_name* はキュー表名です。たとえば、oe_q_any キュー内のメッセージのコンシューマを検索するには、次の問合せを実行します。

```
CONNECT strmadmin/strmadminpw@dbs1.net

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_Q_TABLE_ANY;
```

1. oe ユーザーとして接続します。

```
CONNECT oe/oe@dbs1.net
```

2. デキューするメッセージのコンシューマを入力として取るプロシージャを作成します。次の例のプロシージャでは、oe.cust_address_typ のメッセージがデキューされ、その内容が出力されます。

```
CREATE OR REPLACE PROCEDURE oe.get_cust_address (
consumer IN VARCHAR2) AS
  address         OE.CUST_ADDRESS_TYP;
  deq_address     SYS.AnyData;
  msgid           RAW(16);
  deqopt          DBMS_AQ.DEQUEUE_OPTIONS_T;
  mprop           DBMS_AQ.MESSAGE_PROPERTIES_T;
  new_addresses   BOOLEAN := TRUE;
  next_trans      EXCEPTION;
  no_messages     EXCEPTION;
  pragma exception_init (next_trans, -25235);
  pragma exception_init (no_messages, -25228);
  num_var         pls_integer;
BEGIN
     deqopt.consumer_name := consumer;
     deqopt.wait := 1;
     WHILE (new_addresses) LOOP
     BEGIN
      DBMS_AQ.DEQUEUE(
         queue_name          =>  'strmadmin.oe_q_any',
         dequeue_options     =>  deqopt,
         message_properties  =>  mprop,
         payload             =>  deq_address,
         msgid               =>  msgid);
      deqopt.navigation := DBMS_AQ.NEXT;
         DBMS_OUTPUT.PUT_LINE('****');
         IF (deq_address.GetTypeName() = 'OE.CUST_ADDRESS_TYP') THEN
             DBMS_OUTPUT.PUT_LINE('Message TYPE is: ' ||
                                  deq_address.GetTypeName());
             num_var := deq_address.GetObject(address);
             DBMS_OUTPUT.PUT_LINE(' **** CUSTOMER ADDRESS **** ');
             DBMS_OUTPUT.PUT_LINE(address.street_address);
             DBMS_OUTPUT.PUT_LINE(address.postal_code);
             DBMS_OUTPUT.PUT_LINE(address.city);
             DBMS_OUTPUT.PUT_LINE(address.state_province);
             DBMS_OUTPUT.PUT_LINE(address.country_id);
         ELSE
            DBMS_OUTPUT.PUT_LINE('Message TYPE is: ' ||
                                 deq_address.GetTypeName());
```

```
        END IF;
      COMMIT;
    EXCEPTION
      WHEN next_trans THEN
      deqopt.navigation := DBMS_AQ.NEXT_TRANSACTION;
      WHEN no_messages THEN
        new_addresses := FALSE;
        DBMS_OUTPUT.PUT_LINE('No more messages');
    END;
  END LOOP;
END;
/
```

3. 手順 1 で作成したプロシージャを実行し、次の例のようにデキューするメッセージのコンシューマを指定します。

```
SET SERVEROUTPUT ON SIZE 100000
EXEC oe.get_cust_address('LOCAL_AGENT');
```

SYS.AnyData キューと型付きのキューの間でのメッセージの伝播

Streams 環境では、SYS.AnyData キューと型付きのキューを相互運用できます。型付きのキューとは、特定の型のメッセージしかステージングできないキューです。メッセージを SYS.AnyData キューから型付きのキューに伝播させるには、メッセージを型付きのキューの型と一致するように変換する必要があります。ここでは、LCR 以外のユーザー・メッセージと LCR を、SYS.AnyData キューと型付きのキューの間で伝播させる例を示します。

注意： この項の例では、13-18 ページの「SYS.AnyData ラッパーでのユーザー・メッセージ・ペイロードのラップ」の例を完了していることを想定しています。

関連項目： SYS.AnyData と型付きのキューとの間の伝播の詳細は、3-15 ページの「メッセージの伝播と SYS.AnyData キュー」を参照してください。

LCR 以外のユーザー・メッセージを型付きのキューに伝播させる例

次の手順では、SYS.AnyData キュー oe_q_any から oe.cust_address_typ 型の型付きのキュー oe_q_address への伝播を設定します。ソース・キュー oe_q_any は dbs1.net データベースにあり、宛先キュー oe_q_address は dbs2.net データベースにあります。どちらのキューも strmadmin の所有です。

1. dbs1.net で、権限を付与できる管理ユーザーとして接続します。
2. まだ付与されていない場合は、次の権限を strmadmin に付与します。

```
GRANT EXECUTE ON DBMS_TRANSFORM TO strmadmin;
```

3. dbs1.net と dbs2.net の oe.cust_address_typ で、strmadmin に EXECUTE 権限を付与します。

```
CONNECT oe/oe@dbs1.net

GRANT EXECUTE ON oe.cust_address_typ TO strmadmin;

CONNECT oe/oe@dbs2.net

GRANT EXECUTE ON oe.cust_address_typ TO strmadmin;
```

4. 存在しない場合は、dbs2.net で型付きのキューを作成します。

```
CONNECT strmadmin/strmadminpw@dbs2.net

BEGIN
  DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table         => 'strmadmin.oe_q_table_address',
    queue_payload_type  => 'oe.cust_address_typ',
    multiple_consumers  => true);
  DBMS_AQADM.CREATE_QUEUE(
    queue_name   => 'strmadmin.oe_q_address',
    queue_table  => 'strmadmin.oe_q_table_address');
  DBMS_AQADM.START_QUEUE(
    queue_name   => 'strmadmin.oe_q_address');
END;
/
```

5. 存在しない場合は、dbs1.net と dbs2.net の間のデータベース・リンクを作成します。

```
CONNECT strmadmin/strmadminpw@dbs1.net

CREATE DATABASE LINK dbs2.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
  USING 'DBS2.NET';
```

6. strmadmin スキーマ内の dbs1.net に、oe.cust_address_typ オブジェクトを含む SYS.AnyData ペイロードを取り、oe.cust_address_typ オブジェクトを戻すファンクション any_to_cust_address_typ を作成します。

```
CREATE OR REPLACE FUNCTION strmadmin.any_to_cust_address_typ(
  in_any IN SYS.AnyData)
RETURN OE.CUST_ADDRESS_TYP
AS
  address       OE.CUST_ADDRESS_TYP;
  num_var       NUMBER;
  type_name     VARCHAR2(100);
BEGIN
  -- Get the type of object
  type_name := in_any.GetTypeName();
  -- Check if the object type is OE.CUST_ADDRESS_TYP
  IF (type_name = 'OE.CUST_ADDRESS_TYP') THEN
    -- Put the address in the message into the address variable
    num_var := in_any.GetObject(address);
    RETURN address;
  ELSE
    raise_application_error(-20101, 'Conversion failed - ' || type_name);
  END IF;
END;
/
```

7. DBMS_TRANSFORM パッケージを使用して dbs1.net で変換を作成します。

```
BEGIN
  DBMS_TRANSFORM.CREATE_TRANSFORMATION(
   schema         => 'strmadmin',
   name           => 'anytoaddress',
   from_schema    => 'SYS',
   from_type      => 'ANYDATA',
   to_schema      => 'oe',
   to_type        => 'cust_address_typ',
   transformation => 'strmadmin.any_to_cust_address_typ(source.user_data)');
END;
/
```

8. 存在しない場合は、型付きのキューのサブスクライバを作成します。このサブスクライバには、適切な型のメッセージのみが宛先キューに確実に伝播されるルールを含める必要があります。

```
DECLARE
  subscriber  SYS.AQ$_AGENT;
BEGIN
  subscriber := SYS.AQ$_AGENT ('ADDRESS_AGENT_REMOTE',
                               'STRMADMIN.OE_Q_ADDRESS@DBS2.NET',
                               0);
  DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name     => 'strmadmin.oe_q_any',
    subscriber     => subscriber,
    rule           =>
                'TAB.USER_DATA.GetTypeName()=''OE.CUST_ADDRESS_TYP''',
    transformation => 'strmadmin.anytoaddress');
END;
/
```

9. dbs1.net の SYS.AnyData キューと dbs2.net の型付きのキューの間の伝播をスケジュールします。

```
BEGIN
  DBMS_AQADM.SCHEDULE_PROPAGATION(
    queue_name   => 'strmadmin.oe_q_any',
    destination  => 'dbs2.net');
END;
/
```

10. SYS.AnyData ラッパーにラップされている oe.cust_address_typ 型のメッセージをエンキューします。

```
CONNECT oe/oe@dbs1.net

BEGIN
  oe.enq_proc(SYS.AnyData.ConvertObject(oe.cust_address_typ(
    '1668 Chong Tao','111181','Beijing',NULL, 'CN')));
END;
/
COMMIT;
```

11. 伝播が完了するまでしばらく待機し、dbs2.net でキュー表を問い合せて伝播されたメッセージを確認します。

```
CONNECT strmadmin/strmadminpw@dbs2.net

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_Q_TABLE_ADDRESS;
```

関連項目： 伝播中の変換の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

型付きのキューに LCR を伝播させる例

SYS.AnyData キューから型付きのキューに LCR を伝播させるには、非 LCR イベントの場合と同じ手順を実行します。ただし、Oracle には変換ファンクションが用意されています。DBMS_STREAMS パッケージの次のファンクションを使用すると、SYS.AnyData キュー内の LCR を型付きのキュー内のメッセージに変換できます。

- CONVERT_ANYDATA_TO_LCR_ROW ファンクションでは、行 LCR を含む SYS.AnyData ペイロードが SYS.LCR$_ROW_RECORD ペイロードに変換されます。
- CONVERT_ANYDATA_TO_LCR_DDL ファンクションでは、DDL LCR を含む SYS.AnyData ペイロードが SYS.LCR$_DDL_RECORD ペイロードに変換されます。

ユーザー・エンキュー LCR を適切な型付きのキューに伝播することはできますが、取得した LCR の型付きのキューへの伝播はサポートされません。

次の例では、SYS.AnyData キュー oe_q_any から SYS.LCR$_ROW_RECORD 型の型付きのキュー oe_q_lcr への、行 LCR の伝播が設定されます。ソース・キュー oe_q_any は dbs1.net データベースにあり、宛先キュー oe_q_lcr は dbs3.net データベースにあります。

1. dbs1.net で、権限を付与できる管理ユーザーとして接続します。
2. まだ付与されていない場合は、次の権限を strmadmin に付与します。

```
GRANT EXECUTE ON DBMS_TRANSFORM TO strmadmin;
```

3. 存在しない場合は、LCR 型のキューを作成します。

```
CONNECT strmadmin/strmadminpw@dbs3.net

BEGIN
  DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table         => 'strmadmin.oe_q_table_lcr',
    queue_payload_type  => 'SYS.LCR$_ROW_RECORD',
    multiple_consumers  => true);
  DBMS_AQADM.CREATE_QUEUE(
    queue_name   => 'strmadmin.oe_q_lcr',
    queue_table  => 'strmadmin.oe_q_table_lcr');
  DBMS_AQADM.START_QUEUE(
    queue_name   => 'strmadmin.oe_q_lcr');
END;
/
```

4. 存在しない場合は、dbs1.net と dbs3.net の間のデータベース・リンクを作成します。

```
CONNECT strmadmin/strmadminpw@dbs1.net

CREATE DATABASE LINK dbs3.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
  USING 'DBS3.NET';
```

5. DBMS_TRANSFORM パッケージを使用して dbs1.net で変換を作成します。

```
BEGIN
  DBMS_TRANSFORM.CREATE_TRANSFORMATION(
    schema         => 'strmadmin',
    name           => 'anytolcr',
    from_schema    => 'SYS',
    from_type      => 'ANYDATA',
    to_schema      => 'SYS',
    to_type        => 'LCR$_ROW_RECORD',
    transformation =>
          'SYS.DBMS_STREAMS.CONVERT_ANYDATA_TO_LCR_ROW(source.user_data)');
END;
/
```

6. 存在しない場合は、型付きのキューのサブスクライバを作成します。このサブスクライバでは、変換パラメータ用に CONVERT_ANYDATA_TO_LCR_ROW ファンクションを指定します。

```
DECLARE
  subscriber  SYS.AQ$_AGENT;
BEGIN
  subscriber := SYS.AQ$_AGENT (
    'ROW_LCR_AGENT_REMOTE',
    'STRMADMIN.OE_Q_LCR@DBS3.NET',
    0);
  DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name     => 'strmadmin.oe_q_any',
    subscriber     => subscriber,
    rule           => 'TAB.USER_DATA.GetTypeName()=''SYS.LCR$_ROW_RECORD''',
    transformation => 'strmadmin.anytolcr');
END;
/
```

7. dbs1.net の SYS.AnyData キューと dbs3.net の LCR キューの間の伝播をスケジュールします。

```
BEGIN
  DBMS_AQADM.SCHEDULE_PROPAGATION(
    queue_name   => 'strmadmin.oe_q_any',
    destination  => 'dbs3.net');
END;
/
```

8. プロシージャを作成し、行 LCR を構成して strmadmin.oe_q_any キューにエンキューします。

```
CONNECT oe/oe@dbs1.net

CREATE OR REPLACE PROCEDURE oe.enq_row_lcr_proc(
                 source_dbname  VARCHAR2,
                 cmd_type       VARCHAR2,
                 obj_owner      VARCHAR2,
                 obj_name       VARCHAR2,
                 old_vals       SYS.LCR$_ROW_LIST,
                 new_vals       SYS.LCR$_ROW_LIST) AS
  eopt           DBMS_AQ.ENQUEUE_OPTIONS_T;
  mprop          DBMS_AQ.MESSAGE_PROPERTIES_T;
  enq_msgid      RAW(16);
  row_lcr        SYS.LCR$_ROW_RECORD;
BEGIN
  mprop.SENDER_ID := SYS.AQ$_AGENT('LOCAL_AGENT', NULL, NULL);
  -- Construct the LCR based on information passed to procedure
  row_lcr := SYS.LCR$_ROW_RECORD.CONSTRUCT(
    source_database_name  =>  source_dbname,
    command_type          =>  cmd_type,
    object_owner          =>  obj_owner,
    object_name           =>  obj_name,
    old_values            =>  old_vals,
    new_values            =>  new_vals);
  -- Enqueue the created row LCR
  DBMS_AQ.ENQUEUE(
    queue_name         =>  'strmadmin.oe_q_any',
    enqueue_options    =>  eopt,
    message_properties =>  mprop,
    payload            =>  SYS.AnyData.ConvertObject(row_lcr),
    msgid              =>  enq_msgid);
END enq_row_lcr_proc;
/
```

9. oe.inventories 表に 1 行を挿入する行 LCR を作成し、その行 LCR を strmadmin.oe_q_any キューにエンキューします。

```
DECLARE
  newunit1  SYS.LCR$_ROW_UNIT;
  newunit2  SYS.LCR$_ROW_UNIT;
  newunit3  SYS.LCR$_ROW_UNIT;
  newvals   SYS.LCR$_ROW_LIST;
BEGIN
  newunit1 := SYS.LCR$_ROW_UNIT(
    'PRODUCT_ID',
    SYS.AnyData.ConvertNumber(3503),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit2 := SYS.LCR$_ROW_UNIT(
    'WAREHOUSE_ID',
    SYS.AnyData.ConvertNumber(1),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit3 := SYS.LCR$_ROW_UNIT(
    'QUANTITY_ON_HAND',
    SYS.AnyData.ConvertNumber(157),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1,newunit2,newunit3);
oe.enq_row_lcr_proc(
  source_dbname  =>  'DBS1.NET',
  cmd_type       =>  'INSERT',
  obj_owner      =>  'OE',
  obj_name       =>  'INVENTORIES',
  old_vals       =>  NULL,
  new_vals       =>  newvals);
END;
/
COMMIT;
```

10. 伝播が完了するまでしばらく待機し、dbs3.net でキュー表を問い合せて伝播されたメッセージを確認します。

```
CONNECT strmadmin/strmadminpw@dbs3.net

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_Q_TABLE_LCR;
```

関連項目： 行 LCR と DDL LCR の変換ファンクションの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_STREAMS パッケージを参照してください。

14

適用プロセスの管理

Streams の適用プロセスでは、特定のキューから論理変更レコード（LCR）とユーザー・メッセージがデキューされ、それが直接適用されるか、ユーザー定義プロシージャにパラメータとして渡されます。

この章の内容は次のとおりです。

- 適用プロセスの作成、起動、停止および削除
- 適用プロセスのルール・セットの管理
- 適用プロセス・パラメータの設定
- 適用プロセスの適用ユーザーの設定
- 適用プロセスのメッセージ・ハンドラの管理
- DML ハンドラの管理
- 適用プロセスの DDL ハンドラの管理
- エラー・ハンドラの管理
- 表の代替キー列の管理
- Streams の競合解消の管理
- 適用エラーの管理
- 接続先データベースでのインスタンス化 SCN の設定

この項で説明する各タスクは、特に明記されていないかぎり、適切な権限を付与されている Streams 管理者が完了する必要があります。

関連項目：

- 第 4 章「Streams 適用プロセス」
- 11-2 ページ「Streams 管理者の構成」
- 16-25 ページ「適用プロセス用の Streams タグの管理」

適用プロセスの作成、起動、停止および削除

この項では、適用プロセスを作成、起動、停止および削除する手順について説明します。

適用プロセスの作成

次のどのプロシージャでも、適用プロセスを作成できます。

- DBMS_STREAMS_ADM.ADD_TABLE_RULES
- DBMS_STREAMS_ADM.ADD_SUBSET_RULES
- DBMS_STREAMS_ADM.ADD_SCHEMA_RULES
- DBMS_STREAMS_ADM.ADD_GLOBAL_RULES
- DBMS_APPLY_ADM.CREATE_APPLY

DBMS_STREAMS_ADM パッケージの各プロシージャでは、指定した名前の適用プロセスが存在しない場合は適用プロセスが作成され、適用プロセスにルール・セットがない場合はルール・セットが作成されます。また、ルール・セットに表ルール、スキーマ・ルールまたはグローバル・ルールを追加できます。

CREATE_APPLY プロシージャでは、適用プロセスは作成されますが、そのルール・セットやルールは作成されません。ただし、CREATE_APPLY プロシージャを使用すると、適用プロセスに関連付ける既存のルール・セットと、イベント・ハンドラ、適用ユーザー、適用タグ、取得イベントとユーザー・エンキュー・イベントのどちらを適用するかなど、他の多数のオプションを指定できます。

適用プロセスに関連付ける Streams キューが存在しない場合は、先にそのキューを作成してから、適用プロセスを作成します。

注意： 作成する適用プロセスの構成によっては、適用プロセスで変更を適用する表の列について、ソース・データベースでサプリメンタル・ロギングが必要になる場合があります。

関連項目：

- 13-2 ページ「Streams のキューの作成」
- サプリメンタル・ロギングが必要な場合については、2-10 ページの「Streams 環境内のサプリメンタル・ロギング」を参照してください。
- 12-8 ページ「ソース・データベースでのサプリメンタル・ロギングの指定」

DBMS_STREAMS_ADM を使用した適用プロセスの作成例

ここでは、DBMS_STREAMS_ADM パッケージの ADD_SCHEMA_RULES プロシージャを実行して、適用プロセスを作成する例について説明します。

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name         => 'hr',
    streams_type        => 'apply',
    streams_name        => 'strm01_apply',
    queue_name          => 'strm01_queue',
    include_dml         => true,
    include_ddl         => false,
    include_tagged_lcr  => false,
    source_database     => 'dbs1.net');
END;
/
```

このプロシージャを実行すると、次のアクションが実行されます。

- 取得イベントをローカル・データベースに適用する適用プロセス strm01_apply が作成されます。この適用プロセスが作成されるのは、それが存在しない場合のみです。
- 適用プロセスが、既存のキュー strm01_queue に関連付けられます。
- 適用プロセスにルール・セットがない場合は、ルール・セットが作成され、適用プロセスに関連付けられます。このルール・セットでは、SYS.STREAMS$_EVALUATION_CONTEXT 評価コンテキストが使用されます。ルール・セット名は、システムによって指定されます。
- ルールが 1 つ作成されます。このルールでは、適用プロセスで、hr スキーマ内のデータベース・オブジェクトに対する DML 変更の結果を含む行 LCR を適用するように指定されます。ルール名は、システムによって指定されます。
- このルールが、適用プロセスに関連付けられたルール・セットに追加されます。

- 適用プロセスの apply_tag が、'00'（2 つのゼロ）と等価の 16 進値に設定されます。適用プロセスで生成される REDO エントリのタグは、この値になります。
- include_tagged_lcr パラメータが false に設定されているため、NULL タグがある場合にのみ適用プロセスで行 LCR を適用するように指定されます。この動作は、適用プロセスのシステム作成ルールを介して実行されます。

関連項目：

- 4-30 ページ「適用プロセスの作成」
- 6-3 ページ「システム作成ルール」
- 8-3 ページ「DBMS_STREAMS_ADM パッケージによって作成されるタグとルール」

DBMS_APPLY_ADM を使用した適用プロセスの作成例

この項の最初の例では、取得イベントを適用する適用プロセスが作成され、2 番目の例ではユーザー・エンキュー・イベントを適用する適用プロセスが作成されます。1 つの適用プロセスで取得イベントとユーザー・エンキュー・イベントの両方を適用することはできません。

関連項目：

- 4-30 ページ「適用プロセスの作成」
- イベント・ハンドラの詳細は、4-4 ページの「イベント処理オプション」を参照してください。
- 8-6 ページ「タグと適用プロセス」
- apply_database_link パラメータを使用して、Oracle 以外のデータベースにイベントを適用するように適用プロセスを構成する方法については、9-2 ページの「Streams を使用した Oracle データベースから Oracle 以外のデータベースへのデータの共有」を参照してください。

取得イベントを適用する適用プロセスの作成例 ここでは、DBMS_APPLY_ADM パッケージの CREATE_APPLY プロシージャを実行して、取得イベントを適用する適用プロセスを作成する例について説明します。

```
BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name          => 'strm02_queue',
    apply_name          => 'strm02_apply',
    rule_set_name       => 'strmadmin.strm01_rule_set',
    message_handler     => NULL,
    ddl_handler         => 'hr.ddl_handler',
    apply_user          => 'hr',
    apply_database_link => NULL,
    apply_tag           => HEXTORAW('5'),
    apply_captured      => true);
END;
/
```

このプロシージャを実行すると、次のアクションが実行されます。

- 適用プロセス strm02_apply が作成されます。同じ名前の適用プロセスが存在することは許されません。
- 適用プロセスが、既存のキュー strm02_queue に関連付けられます。
- 適用プロセスが、既存のルール・セット strm01_rule_set に関連付けられます。
- 適用プロセスではメッセージ・ハンドラを使用しないように指定されます。
- DDL ハンドラとして hr スキーマ内の ddl_handler PL/SQL プロシージャが指定されます。CREATE_APPLY プロシージャを実行するユーザーは、ddl_handler PL/SQL プロシージャの EXECUTE 権限を持っている必要があります。
- 変更を適用するユーザーとして、CREATE_APPLY プロシージャを実行中のユーザー（Streams 管理者）ではなく hr が指定されます。
- apply_database_link パラメータが NULL に設定されているため、適用プロセスではローカル・データベースに変更を適用するように指定されます。
- 適用プロセスで生成される各 REDO エントリのタグとして、'5' と等価の 16 進値が指定されます。
- 適用プロセスではユーザー・エンキュー・イベントではなく取得された LCR を適用するように指定されます。したがって、取得プロセスではなくユーザー・アプリケーションによって構成された LCR が適用プロセス用のキュー内でステージングされても、この適用プロセスでは LCR は適用されません。

ユーザー・エンキュー・イベントを適用する適用プロセスの作成例 ここでは、DBMS_APPLY_ADM パッケージの CREATE_APPLY プロシージャを実行して、ユーザー・エンキュー・イベントを適用する適用プロセスを作成する例について説明します。

```
BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name          => 'strm01_queue',
    apply_name          => 'strm03_apply',
    rule_set_name       => 'strmadmin.strm02_rule_set',
    message_handler     => 'strmadmin.mes_handler',
    ddl_handler         => NULL,
    apply_user          => NULL,
    apply_database_link => NULL,
    apply_tag           => NULL,
    apply_captured      => false);
END;
/
```

このプロシージャを実行すると、次のアクションが実行されます。

- 適用プロセス strm03_apply が作成されます。同じ名前の適用プロセスが存在することは許されません。
- 適用プロセスが、既存のキュー strm01_queue に関連付けられます。
- 適用プロセスが、既存のルール・セット strm02_rule_set に関連付けられます。
- メッセージ・ハンドラとして strmadmin スキーマ内の mes_handler PL/SQL プロシージャが指定されます。CREATE_APPLY プロシージャを実行するユーザーは、mes_handler PL/SQL プロシージャの EXECUTE 権限を持っている必要があります。
- 適用プロセスでは DDL ハンドラを使用しないように指定されます。
- apply_user パラメータが NULL に設定されているため、CREATE_APPLY プロシージャを実行するユーザーが変更を適用するユーザーとして指定されます。
- apply_database_link パラメータが NULL に設定されているため、適用プロセスではローカル・データベースに変更を適用するように指定されます。
- 適用プロセスで生成される各 REDO エントリが NULL タグを持つように指定されます。
- 適用プロセスでは取得イベントではなくユーザー・エンキュー・イベントを適用するように指定されます。

適用プロセスの起動

既存の適用プロセスを起動するには、DBMS_APPLY_ADM パッケージの START_APPLY プロシージャを実行します。たとえば、次のプロシージャでは、適用プロセス strm01_apply が起動されます。

```
BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'strm01_apply');
END;
/
```

適用プロセスの停止

既存の適用プロセスを停止するには、DBMS_APPLY_ADM パッケージの STOP_APPLY プロシージャを実行します。たとえば、次のプロシージャでは、適用プロセス strm01_apply が停止されます。

```
BEGIN
  DBMS_APPLY_ADM.STOP_APPLY(
    apply_name => 'strm01_apply');
END;
/
```

適用プロセスの削除

既存の適用プロセスを削除するには、DBMS_APPLY_ADM パッケージの DROP_APPLY プロシージャを実行します。たとえば、次のプロシージャでは、適用プロセス strm02_apply が削除されます。

```
BEGIN
  DBMS_APPLY_ADM.DROP_APPLY(
    apply_name => 'strm02_apply');
END;
/
```

適用プロセスを削除しようとして、指定された適用プロセスの例外キューにエラーが存在する場合、エラーが発生します。したがって、適用プロセスの例外キューにエラーが存在する場合は、適用プロセスを削除する前にエラーを削除します。

関連項目： 14-31 ページ「適用エラーの管理」

適用プロセスのルール・セットの管理

この項では、次のタスクの手順について説明します。

- 適用プロセスのルール・セットの指定
- 適用プロセスのルール・セットへのルールの追加
- 適用プロセスのルール・セットからのルールの削除
- 適用プロセスのルール・セットの削除

関連項目：

- 第 5 章「ルール」
- 第 6 章「Streams でのルールの使用方法」

適用プロセスのルール・セットの指定

適用プロセスに関連付けるルール・セットを指定するには、DBMS_APPLY_ADM パッケージの ALTER_APPLY プロシージャで rule_set_name パラメータを使用します。たとえば、次のプロシージャでは、適用プロセス strm01_apply のルール・セットが strm02_rule_set に設定されます。

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name     => 'strm01_apply',
    rule_set_name => 'strmadmin.strm02_rule_set');
END;
/
```

適用プロセスのルール・セットへのルールの追加

適用プロセスのルール・セットにルールを追加するには、次のいずれかの手順を実行します。

- DBMS_STREAMS_ADM.ADD_TABLE_RULES
- DBMS_STREAMS_ADM.ADD_SUBSET_RULES
- DBMS_STREAMS_ADM.ADD_SCHEMA_RULES
- DBMS_STREAMS_ADM.ADD_GLOBAL_RULES

ここでは、DBMS_STREAMS_ADM パッケージの ADD_TABLE_RULES プロシージャを実行して、適用プロセス strm01_apply のルール・セットにルールを追加する例について説明します。

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name         => 'hr.departments',
    streams_type       => 'apply',
    streams_name       => 'strm01_apply',
    queue_name         => 'strm01_queue',
    include_dml        => true,
    include_ddl        => true,
    source_database    => 'dbs1.net');
END;
/
```

このプロシージャを実行すると、次のアクションが実行されます。

- ルールが 1 つ作成されます。このルールでは、適用プロセスで hr.departments 表に対する DML 変更の結果を含む行 LCR を適用するように指定されます。ルール名は、システムによって指定されます。
- ルールが 1 つ作成されます。このルールでは、適用プロセスで hr.departments 表に対する DDL 変更の結果を含む DDL LCR を適用するように指定されます。ルール名は、システムによって指定されます。
- このルールが、適用プロセスに関連付けられたルール・セットに追加されます。
- 適用プロセスで dbs1.net ソース・データベースからの LCR のみを適用するように指定されます。

関連項目： 6-3 ページ「システム作成ルール」

適用プロセスのルール・セットからのルールの削除

既存の適用プロセスのルール・セットからルールを削除するように指定するには、DBMS_STREAMS_ADM パッケージの REMOVE_RULE プロシージャを実行します。たとえば、次のプロシージャでは、適用プロセス strm01_apply のルール・セットからルール DEPARTMENTS3 が削除されます。

```
BEGIN
  DBMS_STREAMS_ADM.REMOVE_RULE(
    rule_name         => 'DEPARTMENTS3',
    streams_type      => 'apply',
    streams_name      => 'strm01_apply',
    drop_unused_rule => true);
END;
/
```

この例では、REMOVE_RULE プロシージャの drop_unused_rule パラメータが true に設定されています。これはデフォルト設定です。したがって、削除するルールがどのルール・セットにもなければ、そのルールはデータベースから削除されます。drop_unused_rule パラメータが false に設定されている場合、ルールはルール・セットから削除されますが、データベースからは削除されません。

また、適用プロセスのルール・セットからすべてのルールを削除する必要がある場合は、REMOVE_RULE プロシージャの実行時に rule_name パラメータに NULL を指定します。

注意： 取得イベントを適用する適用プロセスのルール・セットからルールをすべて削除すると、その適用プロセスではキューにある取得イベントが適用されなくなります。同様に、ユーザー・エンキュー・イベントを適用する適用プロセスのルール・セットからルールをすべて削除すると、その適用プロセスではキューにあるユーザー・エンキュー・イベントが適用されなくなります。

適用プロセスのルール・セットの削除

適用プロセスからルール・セットを削除するように指定するには、DBMS_APPLY_ADM パッケージの ALTER_APPLY プロシージャで remove_rule_set パラメータを true に設定します。たとえば、次のプロシージャでは、適用プロセス strm01_apply からルール・セットが削除されます。

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name      => 'strm01_apply',
    remove_rule_set => true);
END;
/
```

注意： 取得イベントを適用する適用プロセスのルール・セットを削除すると、その適用プロセスではキューにある取得イベントがすべて適用されます。同様に、ユーザー・エンキュー・イベントを適用する適用プロセスのルール・セットを削除すると、その適用プロセスではキューにあるユーザー・エンキュー・イベントがすべて適用されます。

適用プロセス・パラメータの設定

適用プロセスのパラメータを設定するには、DBMS_APPLY_ADM パッケージの SET_PARAMETER プロシージャを使用します。適用プロセス・パラメータでは、適用プロセスの動作を制御します。

たとえば、次のプロシージャでは、適用プロセス strm01_apply の commit_serialization パラメータが none に設定されます。commit_serialization パラメータをこの値に設定すると、適用プロセスではトランザクションを任意の順序でコミットできます。

```
BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name   => 'strm01_apply',
    parameter    => 'commit_serialization',
    value        => 'none');
END;
/
```

注意：

- value パラメータは、パラメータ値が数値の場合にも、常に VARCHAR2 として入力されます。
- parallelism 適用プロセス・パラメータを 1 よりも大きい値に設定する場合は、適用プロセスで変更が適用される表のすべての一意キー列と外部キー列について、ソース・データベースで条件付きのサプリメンタル・ログ・グループを指定する必要があります。構成によっては、これらの表の他の列についてもサプリメンタル・ロギングが必要になる場合があります。

関連項目：

- 4-31 ページ「適用プロセスのパラメータ」
- 適用プロセス・パラメータの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_APPLY_ADM.SET_PARAMETER プロシージャの項を参照してください。
- 12-8 ページ「ソース・データベースでのサプリメンタル・ロギングの指定」

適用プロセスの適用ユーザーの設定

適用ユーザーとは、すべての DML 文と DDL 文を適用し、ユーザー定義の適用ハンドラを実行するユーザーです。適用プロセスの適用ユーザーを設定するには、DBMS_APPLY_ADM パッケージの ALTER_APPLY プロシージャで apply_user パラメータを使用します。たとえば、次のプロシージャでは、適用プロセス strm03_apply の適用ユーザーが hr に設定されます。

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'strm03_apply',
    apply_user => 'hr');
END;
/
```

apply_user パラメータでは、適用オブジェクトの DML 変更と DDL 変更を実行するために必要な権限と、任意の適用ハンドラを実行するために必要な権限を持ったユーザーを指定する必要があります。また、このユーザーには、適用プロセスで使用するキューのデキュー権限と、適用プロセスで使用されるルール・セットと変換のファンクションを実行する権限も必要です。これらの権限は、適用ユーザーに直接付与してください。ロールを介しては付与できません。

適用プロセスのメッセージ・ハンドラの管理

この項では、適用プロセスのメッセージ・ハンドラを設定する手順と削除する手順について説明します。

関連項目：

- 4-3 ページ「適用プロセスによるイベント処理」
- メッセージ・ハンドラの作成例については、第 19 章「Streams メッセージングの例」を参照してください。

適用プロセスのメッセージ・ハンドラの設定

適用プロセスのメッセージ・ハンドラを設定するには、DBMS_APPLY_ADM パッケージの ALTER_APPLY プロシージャで message_handler パラメータを使用します。たとえば、次のプロシージャでは、適用プロセス strm03_apply のメッセージ・ハンドラに、hr スキーマ内の mes_proc プロシージャが設定されます。

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name      => 'strm03_apply',
    message_handler => 'hr.mes_proc');
END;
/
```

ALTER_APPLY プロシージャを実行するユーザーは、指定したメッセージ・ハンドラの EXECUTE 権限を持っている必要があります。

適用プロセスのメッセージ・ハンドラの削除

適用プロセスのメッセージ・ハンドラを削除するには、DBMS_APPLY_ADM パッケージの ALTER_APPLY プロシージャで remove_message_handler パラメータを true に設定します。たとえば、次のプロシージャでは、適用プロセス strm03_apply からメッセージ・ハンドラが削除されます。

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name             => 'strm03_apply',
    remove_message_handler => true);
END;
/
```

DML ハンドラの管理

この項では、DML ハンドラを作成、設定および削除する手順について説明します。

関連項目：

- 4-3 ページ「適用プロセスによるイベント処理」
- DML ハンドラの使用方法を示す詳細な例は、第 20 章「単一データベースの取得および適用の例」を参照してください。

DML ハンドラの作成

DML ハンドラには、次のシグネチャが必要です。

```
PROCEDURE user_procedure (
   parameter_name   IN  SYS.AnyData);
```

この場合、*user_procedure* はプロシージャ名で、*parameter_name* はプロシージャに渡されるパラメータの名前です。プロシージャに渡されるパラメータは、行 LCR が SYS.AnyData にカプセル化されたものです。

ユーザー・プロシージャには、次の制限が適用されます。

- COMMIT 文または ROLLBACK 文は実行しないでください。これらの文を実行すると、LCR を含むトランザクションの一貫性が損なわれる危険性があります。
- 行 LCR 用の EXECUTE メンバー・プロシージャを使用して行を操作する場合は、1 回の行操作で複数行を操作しないでください。複数行を操作する DML 文は、手動で構成して実行する必要があります。
- コマンド・タイプが UPDATE または DELETE の場合、LCR 用の EXECUTE メンバー・プロシージャを使用して送り直す行操作では、古い値のリストにキー全体を含める必要があります。キーは、SET_KEY_COLUMNS プロシージャで代替キーを指定しないかぎり主キーです。
- コマンド・タイプが INSERT の場合、LCR 用の EXECUTE メンバー・プロシージャを使用して送り直す行操作では、新規の値のリストにキー全体を含める必要があります。それ以外の場合は、行が重複する可能性があります。キーは、SET_KEY_COLUMNS プロシージャで代替キーを指定しないかぎり主キーです。

DML ハンドラは、行 LCR のカスタマイズされた処理に使用できます。たとえば、ハンドラで LCR を変更し、LCR 用の EXECUTE メンバー・プロシージャを使用して実行できます。DML ハンドラ内で行 LCR を実行すると、適用プロセスでは行 LCR の DML ハンドラもエラー・ハンドラもコールされずに、行 LCR が適用されます。

また、DML ハンドラを使用して DML 変更の履歴を記録することもできます。たとえば、DML ハンドラでは、処理する LCR に関する情報を表に挿入し、EXECUTE メンバー・プロシージャを使用してその LCR を適用できます。この種の DML ハンドラを作成するには、まず履歴情報を保持する表を作成します。

```
CREATE TABLE strmadmin.history_row_lcrs(
  timestamp             DATE,
  source_database_name  VARCHAR2(128),
  command_type          VARCHAR2(30),
  object_owner          VARCHAR2(32),
  object_name           VARCHAR2(32),
  tag                   RAW(10),
  transaction_id        VARCHAR2(10),
  scn                   NUMBER,
  old_values            SYS.LCR$_ROW_LIST,
  new_values            SYS.LCR$_ROW_LIST)
    NESTED TABLE old_values STORE AS old_values_ntab
    NESTED TABLE new_values STORE AS new_values_ntab;
```

次に、行 LCR 内の情報を history_row_lcrs 表に挿入して行 LCR を実行するプロシージャを作成します。

```
CREATE OR REPLACE PROCEDURE history_dml(in_any IN SYS.ANYDATA)
 IS
  lcr   SYS.LCR$_ROW_RECORD;
  rc    PLS_INTEGER;
 BEGIN
  -- Access the LCR
  rc := in_any.GETOBJECT(lcr);
  -- Insert information in the LCR into the history_row_lcrs table
  INSERT INTO strmadmin.history_row_lcrs VALUES
   (SYSDATE, lcr.GET_SOURCE_DATABASE_NAME(), lcr.GET_COMMAND_TYPE(),
    lcr.GET_OBJECT_OWNER(), lcr.GET_OBJECT_NAME(),
    lcr.GET_TAG(), lcr.GET_TRANSACTION_ID(), lcr.GET_SCN(),
    lcr.GET_VALUES('old'), lcr.GET_VALUES('new', 'n'));
  --  Apply row LCR
  lcr.EXECUTE(true);
END;
/
```

注意： 接続先データベースで DML ハンドラに必要なすべての列について、ソース・データベースで無条件のサプリメンタル・ロギングを指定する必要があります。この例の DML ハンドラの場合は、行 LCR に関する情報を記録するのみで、他の方法による行 LCR の操作は行わないため、追加のサプリメンタル・ロギングは不要です。

関連項目： 12-8 ページ「ソース・データベースでのサプリメンタル・ロギングの指定」

DML ハンドラの設定

DML ハンドラでは、適用プロセスによってデキューされた、特定の表に対する特定の操作を含む各行 LCR が処理されます。同じ表に複数の DML ハンドラを指定して、その表に対する異なる操作を処理できます。ローカル・データベース内の指定した表に変更を適用するすべての適用プロセスで、指定した DML ハンドラが使用されます。

DML ハンドラを設定するには、DBMS_APPLY_ADM パッケージの SET_DML_HANDLER プロシージャを使用します。たとえば、次のプロシージャでは、hr.locations 表に対する UPDATE 操作用の DML ハンドラが設定されます。したがって、変更をローカルに適用する適用プロセスでは、hr.locations 表に対する UPDATE 操作を含む行 LCR をデキューするときに、その行 LCR が処理のために strmadmin スキーマの history_dml PL/SQL プロシージャに送信されます。適用プロセスでは、この種の変更を含む行 LCR が直接適用されることはありません。

```
BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name         => 'hr.locations',
    object_type         => 'TABLE',
    operation_name      => 'UPDATE',
    error_handler       => false,
    user_procedure      => 'strmadmin.history_dml',
    apply_database_link => NULL);
END;
/
```

注意：

- 適用プロセスで Oracle 以外のリモート・データベースに変更を適用する場合は、同じ表に異なる DML ハンドラを使用できます。apply_database_link パラメータを非 NULL 値に設定して DBMS_APPLY_ADM パッケージの SET_DML_HANDLER プロシージャを実行すると、Oracle 以外のリモート・データベースに適用される変更の DML ハンドラを指定できます。
- SET_DML_HANDLER プロシージャを実行する場合、ハンドラが使用されるオブジェクトを指定すると、指定オブジェクトがローカルの接続先データベースに存在するかどうかが Oracle によりチェックされます。オブジェクトが存在しない場合、エラーが発生します。したがって、ソース・データベースと接続先データベースでオブジェクト名が異なる場合、行 LCR が適用される前に、ルールベースの変換を使用して行 LCR のオブジェクト名を変換してください。

関連項目：

- 9-3 ページ「Oracle から Oracle 以外への環境での適用プロセスの構成」
- 6-23 ページ「ルールベースの変換」

DML ハンドラの削除

DML ハンドラを削除するには、DBMS_APPLY_ADM パッケージの SET_DML_HANDLER プロシージャを使用します。このプロシージャを実行するときに、特定の表に対する特定の操作について、user_procedure パラメータを NULL に設定します。たとえば、次のプロシージャでは、hr.locations 表に対する UPDATE 操作用の DML ハンドラが削除されます。DML ハンドラを削除すると、変更をローカルに適用する適用プロセスでは、この種の変更を含む行 LCR が直接適用されます。

```
BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name    => 'hr.locations',
    object_type    => 'TABLE',
    operation_name => 'UPDATE',
    error_handler  => false,
    user_procedure => NULL);
END;
/
```

適用プロセスの DDL ハンドラの管理

この項では、適用プロセスの DDL ハンドラを作成、指定および削除する手順について説明します。

注意： 適用された DDL LCR は、すべて自動的にコミットされます。したがって、DDL ハンドラが DDL LCR の EXECUTE メンバー・プロシージャをコールすると、自動的にコミットが実行されます。

関連項目：

- 4-3 ページ「適用プロセスによるイベント処理」
- LCR 型用の EXECUTE メンバー・プロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

適用プロセスの DDL ハンドラの作成

DDL ハンドラには、次のシグネチャが必要です。

```
PROCEDURE handler_procedure (
   parameter_name   IN  SYS.AnyData);
```

handler_procedure はプロシージャ名で、*parameter_name* はプロシージャに渡されるパラメータの名前です。プロシージャに渡されるパラメータは、DDL LCR が SYS.AnyData にカプセル化されたものです。

DDL ハンドラは、DDL LCR のカスタマイズされた処理に使用できます。たとえば、ハンドラで LCR を変更し、LCR 用の EXECUTE メンバー・プロシージャを使用して実行できます。DDL ハンドラ内で DDL LCR を実行すると、適用プロセスでは DDL ハンドラを再度コールせずに LCR が適用されます。

また、DDL ハンドラを使用して DDL 変更の履歴を記録することもできます。たとえば、DDL ハンドラでは、処理する LCR に関する情報を表に挿入し、EXECUTE メンバー・プロシージャを使用してその LCR を適用できます。

この種の DDL ハンドラを作成するには、まず履歴情報を保持する表を作成します。

```
CREATE TABLE strmadmin.history_ddl_lcrs(
  timestamp             DATE,
  source_database_name  VARCHAR2(128),
  command_type          VARCHAR2(30),
  object_owner          VARCHAR2(32),
  object_name           VARCHAR2(32),
  object_type           VARCHAR2(18),
  ddl_text              CLOB,
  logon_user            VARCHAR2(32),
  current_schema        VARCHAR2(32),
  base_table_owner      VARCHAR2(32),
  base_table_name       VARCHAR2(32),
  tag                   RAW(10),
  transaction_id        VARCHAR2(10),
  scn                   NUMBER);
```

次に、DDL LCR 内の情報を history_ddl_lcrs 表に挿入して DDL LCR を実行するプロシージャを作成します。

```
CREATE OR REPLACE procedure history_ddl(in_any IN SYS.ANYDATA)
 IS
   lcr       SYS.LCR$_DDL_RECORD;
   rc        PLS_INTEGER;
   ddl_text  CLOB;
 BEGIN
   -- Access the LCR
   rc := in_any.GETOBJECT(lcr);
   DBMS_LOB.CREATETEMPORARY(ddl_text, TRUE);
   lcr.GET_DDL_TEXT(ddl_text);
   --  Insert DDL LCR information into history_ddl_lcrs table
   INSERT INTO strmadmin.history_ddl_lcrs VALUES(
     SYSDATE, lcr.GET_SOURCE_DATABASE_NAME(), lcr.GET_COMMAND_TYPE(),
     lcr.GET_OBJECT_OWNER(), lcr.GET_OBJECT_NAME(), lcr.GET_OBJECT_TYPE(),
     ddl_text, lcr.GET_LOGON_USER(), lcr.GET_CURRENT_SCHEMA(),
     lcr.GET_BASE_TABLE_OWNER(), lcr.GET_BASE_TABLE_NAME(), lcr.GET_TAG(),
     lcr.GET_TRANSACTION_ID(), lcr.GET_SCN());
   --  Apply DDL LCR
   lcr.EXECUTE();
   -- Free temporary LOB space
   DBMS_LOB.FREETEMPORARY(ddl_text);
END;
/
```

適用プロセスの DDL ハンドラの設定

DDL ハンドラでは、適用プロセスによってデキューされた DDL LCR がすべて処理されます。適用プロセスの DDL ハンドラを設定するには、DBMS_APPLY_ADM パッケージの ALTER_APPLY プロシージャで ddl_handler パラメータを使用します。たとえば、次のプロシージャでは、適用プロセス strm01_apply の DDL ハンドラに strmadmin スキーマ内の history_ddl プロシージャが設定されます。

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name  => 'strm01_apply',
    ddl_handler => 'strmadmin.history_ddl');
END;
/
```

適用プロセスの DDL ハンドラの削除

DDL ハンドラでは、適用プロセスによってデキューされた DDL LCR がすべて処理されます。適用プロセスの DDL ハンドラを削除するには、DBMS_APPLY_ADM パッケージの ALTER_APPLY プロシージャで remove_ddl_handler パラメータを true に設定します。たとえば、次のプロシージャでは、適用プロセス strm01_apply から DDL ハンドラが削除されます。

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name         => 'strm01_apply',
    remove_ddl_handler => true);
END;
/
```

エラー・ハンドラの管理

この項では、エラー・ハンドラを作成、設定および削除する手順について説明します。

関連項目： 4-3 ページ「適用プロセスによるイベント処理」

エラー・ハンドラの作成

エラー・ハンドラを作成するには、DBMS_APPLY_ADM パッケージの SET_DML_HANDLER プロシージャを実行して、error_handler パラメータを true に設定します。

エラー・ハンドラには、次のシグネチャが必要です。

```
PROCEDURE user_procedure (
     message             IN SYS.AnyData,
     error_stack_depth   IN NUMBER,
     error_numbers       IN DBMS_UTILITY.NUMBER_ARRAY,
     error_messages      IN emsg_array);
```

user_procedure はプロシージャ名です。各パラメータは必須であり、指定されたデータ型の値を指定する必要があります。ただし、パラメータ名は変更できます。emsg_array パラメータには、VARCHAR2 型の PL/SQL 表である 76 文字以上のユーザー定義配列を指定する必要があります。

注意： エラー・ハンドラについては、SET_DML_HANDLER で指定されたユーザー・プロシージャに関する特定の制限に従う必要があります。これらの制限の詳細は、14-13 ページの「DML ハンドラの作成」を参照してください。

エラー・ハンドラの実行結果は、次のいずれかになります。

- エラー・ハンドラによってエラーが正常に解決され、該当する場合は行 LCR が適用され、制御が適用プロセスに戻されます。
- エラー・ハンドラがエラーの解決に失敗し、エラーが呼び出されます。エラーが呼び出されると、トランザクションがロールバックされ、例外キューに置かれます。

DML 操作を再試行する必要がある場合は、エラー・ハンドラ・プロシージャで LCR 用の EXECUTE メンバー・プロシージャを実行します。

次の例では、hr.regions 表の主キー違反を解決するエラー・ハンドラ regions_pk_error を作成します。接続先データベースで、ユーザーが hr.regions 表に行を挿入し、リモート・ソース・データベースの取得プロセスによって発生した hr.regions 表に対する変更が、適用プロセスによって適用されるとします。この環境では、接続先データベースのユーザーが、ソース・データベースから適用される挿入行 LCR と同じ主キー値を持つ行を挿入すると、エラーが発生する可能性があります。

この例では、hr.regions 表の主キー違反エラーごとに次の情報を記録するために、strmadmin スキーマに表 errorlog を作成します。

- エラー発生時のタイムスタンプ。
- エラーの原因となったユーザー（送信者）。これは、取得された LCR の場合は取得プロセス名、ユーザーがエンキューした LCR の場合は AQ エージェント名です。
- 将来、他のオブジェクトのエラーがログに記録される可能性があるため、DML 操作が実行されたオブジェクトの名前。
- DML 操作に使用されたコマンドのタイプ。
- 違反となった制約の名前。
- エラー・メッセージ。
- エラーの原因となった LCR。

このエラー・ハンドラでは、hr.regions 表の主キー違反によるエラーのみが解決されます。このタイプのエラーを解決するために、エラー・ハンドラでは順序を使用して行 LCR 内の region_id 値が変更されてから、行 LCR が実行されて適用されます。他のタイプのエラーが発生した場合は、errorlog 表に格納した行 LCR を使用して手動で解決できます。

たとえば、エラー・ハンドラでは次のエラーが解決されます。

1. 接続先データベースで、ユーザーが hr.regions 表に region_id の値が 6、region_name の値が 'LILLIPUT' の 1 行を挿入します。
2. ソース・データベースで、ユーザーが hr.regions 表に region_id の値が 6、region_name の値が 'BROBDINGNAG' の 1 行を挿入します。
3. ソース・データベースの取得プロセスが、手順 2 で説明した変更を取得します。

4. 伝播が、ソース・データベースのキューから接続先データベースの適用プロセスで使用されるキューに、変更を含む LCR を伝播させます。
5. 適用プロセスが LCR を適用すると、主キー違反のためエラーになります。
6. 適用プロセスが、エラー・ハンドラを起動してエラーを処理します。
7. エラー・ハンドラは、エラーを strmadmin.errorlog 表に記録します。
8. エラー・ハンドラが、順序を使用して LCR 内で region_id 値を変更し、その LCR を実行して適用します。

regions_pk_error エラー・ハンドラを作成する手順は、次のとおりです。

1. hr ユーザーとして接続して次の文を実行し、エラー・ハンドラで新規の主キー値を割り当てるために使用される順序を作成します。

```
CONNECT hr/hr

CREATE SEQUENCE hr.reg_exception_s START WITH 9000;
```

この例では、接続先データベースのユーザーは region_id が 9000 以上の hr.regions 表には行を挿入しないと仮定します。

2. Streams 管理者に、この順序に対する ALL 権限を付与します。

```
GRANT ALL ON reg_exception_s TO strmadmin;
```

3. Streams 管理者として接続して次の文を実行し、errorlog 表を作成します。

```
CONNECT strmadmin/strmadminpw

CREATE TABLE strmadmin.errorlog(
  logdate       DATE,
  sender        VARCHAR2(100),
  object_name   VARCHAR2(32),
  command_type  VARCHAR2(30),
  errnum        NUMBER,
  errmsg        VARCHAR2(2000),
  text          VARCHAR2(2000),
  lcr           SYS.LCR$_ROW_RECORD);
```

4. regions_pk_error プロシージャを含めるパッケージを作成します。

```
CREATE OR REPLACE PACKAGE errors_pkg
AS
 TYPE emsg_array IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;
 PROCEDURE regions_pk_error(
   message            IN SYS.ANYDATA ,
   error_stack_depth  IN NUMBER ,
   error_numbers      IN DBMS_UTILITY.NUMBER_ARRAY,
   error_messages     IN EMSG_ARRAY);
END errors_pkg ;
/
```

5. regions_pk_error プロシージャを含むパッケージ本体を作成します。

```
CREATE OR REPLACE PACKAGE BODY errors_pkg AS
 PROCEDURE regions_pk_error (
   message            IN SYS.ANYDATA,
   error_stack_depth  IN NUMBER,
   error_numbers      IN DBMS_UTILITY.NUMBER_ARRAY,
   error_messages     IN EMSG_ARRAY )
 IS
  reg_id     NUMBER;
  ad         SYS.ANYDATA;
  lcr        SYS.LCR$_ROW_RECORD;
  ret        PLS_INTEGER;
  vc         VARCHAR2(30) ;
  errlog_rec errorlog%ROWTYPE ;
  ov2        SYS.LCR$_ROW_LIST;
 BEGIN
  -- Access the error number from the top of the stack.
  -- In case of check constraint violation,
  -- get the name of the constraint violated
  IF error_numbers(1) IN ( 1 , 2290 ) THEN
   ad  := DBMS_STREAMS.GET_INFORMATION('CONSTRAINT_NAME');
   ret := ad.GetVarchar2(errlog_rec.text);
  ELSE
   errlog_rec.text := NULL ;
  END IF ;
  ad  := DBMS_STREAMS.GET_INFORMATION('SENDER');
  ret := ad.GETVARCHAR2(errlog_rec.sender);
  -- Try to access the LCR
  ret := message.GETOBJECT(lcr);
  errlog_rec.object_name  := lcr.GET_OBJECT_NAME() ;
  errlog_rec.command_type := lcr.GET_COMMAND_TYPE() ;
  errlog_rec.errnum := error_numbers(1) ;
  errlog_rec.errmsg := error_messages(1) ;
  INSERT INTO strmadmin.errorlog VALUES (SYSDATE, errlog_rec.sender,
```

```
      errlog_rec.object_name, errlog_rec.command_type,
      errlog_rec.errnum, errlog_rec.errmsg, errlog_rec.text, lcr);
  -- Add the logic to change the contents of LCR with correct values
  -- In this example, get a new region_id number
  -- from the hr.reg_exception_s sequence
  ov2 := lcr.GET_VALUES('new', 'n');
  FOR i IN 1 .. ov2.count
  LOOP
    IF ov2(i).column_name = 'REGION_ID' THEN
     SELECT hr.reg_exception_s.NEXTVAL INTO reg_id FROM DUAL;
     ov2(i).data := Sys.AnyData.ConvertNumber(reg_id) ;
    END IF ;
  END LOOP ;
  -- Set the NEW values in the LCR
  lcr.SET_VALUES(value_type => 'NEW', value_list => ov2);
  -- Execute the modified LCR to apply it
  lcr.EXECUTE(true);
 END regions_pk_error;
END errors_pkg;
/
```

注意：

- 変更があった行に対する以降の変更が正常に適用されるように、2 つのデータベースでできるだけ迅速に行を収束させる必要があります。つまり、ソース・データベースと接続先データベースで、行の region_id を一致させる必要があります。この手動変更がデータベースで再取得されないようにする必要がある場合は、DBMS_STREAMS パッケージの SET_TAG プロシージャを使用して、値の変更が取得されないようにするセッション用のタグを設定します。
- この例のエラー・ハンドラは、LCR 用の GET_VALUES メンバー関数と SET_VALUES メンバー・プロシージャの使用を示しています。ただし、LCR 内で変更する値が 1 つのみの場合は、GET_VALUE メンバー関数と SET_VALUE メンバー・プロシージャを使用する方が簡単で効率的な場合があります。

関連項目： 16-24 ページ「現行セッションで生成されるタグ値の設定」

エラー・ハンドラの設定

エラー・ハンドラでは、適用プロセスによってデキューされた、特定の表に対する特定の操作を含む行 LCR によって発生するエラーが処理されます。同じ表に複数のエラー・ハンドラを指定して、その表に対する異なる操作によって発生するエラーを処理できます。ローカル・データベース内の指定した表に変更を適用するすべての適用プロセスで、指定したエラー・ハンドラが使用されます。

エラー・ハンドラは、DBMS_APPLY_ADM パッケージの SET_DML_HANDLER プロシージャを使用して設定できます。このプロシージャを実行してエラー・ハンドラを設定する場合は、error_handler パラメータを true に設定します。

たとえば、次のプロシージャでは、hr.regions 表に対する INSERT 操作用のエラー・ハンドラが設定されます。したがって、ローカルの hr.regions 表に対する INSERT 操作を含む行 LCR が適用プロセスによってデキューされ、その行 LCR がエラーになると、適用プロセスでは行 LCR が処理のために strmadmin.errors_pkg.regions_pk_error PL/SQL プロシージャに送信されます。エラー・ハンドラでエラーを解決できない場合は、その行 LCR および同じトランザクション内の他のすべての行 LCR が例外キューに移動されます。

次のプロシージャを実行してエラー・ハンドラを設定します。

```
BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name         => 'hr.regions',
    object_type         => 'TABLE',
    operation_name      => 'INSERT',
    error_handler       => true,
    user_procedure      => 'strmadmin.errors_pkg.regions_pk_error',
    apply_database_link => NULL);
END;
/
```

エラー・ハンドラの削除

エラー・ハンドラを削除するには、DBMS_APPLY_ADM パッケージの SET_DML_HANDLER プロシージャを使用します。このプロシージャを実行するときに、特定の表に対する特定の操作について、user_procedure パラメータを NULL に設定します。

たとえば、次のプロシージャでは、hr.regions 表に対する INSERT 操作用のエラー・ハンドラが削除されます。

```
BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name     => 'hr.regions',
    object_type     => 'TABLE',
    operation_name => 'INSERT',
    user_procedure => NULL);
END;
/
```

注意： error_handler パラメータを指定する必要はありません。

表の代替キー列の管理

この項では、表の代替キー列を設定および削除する手順について説明します。

関連項目： 4-11 ページ「代替キー列」

表の代替キー列の設定

適用プロセスで表に変更が適用される場合に、代替キー列を使用すると、主キーを持つ表の主キー列を置換するか、主キーを持たない表の主キー列として使用できます。表の代替キー列を設定するには、DBMS_APPLY_ADM パッケージの SET_KEY_COLUMNS プロシージャを使用します。この設定は、ローカルの変更をデータベースに適用するすべての適用プロセスに適用されます。

たとえば、hr.employees 表の代替キー列を first_name 列、last_name 列および hire_date 列に設定し、employee_id 列を置換するには、次のプロシージャを実行します。

```
BEGIN
  DBMS_APPLY_ADM.SET_KEY_COLUMNS(
    object_name         => 'hr.employees',
    column_list         => 'first_name,last_name,hire_date');
END;
/
```

注意：

- 接続先データベースで column_list または column_table パラメータで代替キー列として指定したすべての列について、ソース・データベースで無条件のサプリメンタル・ログ・グループを指定する必要があります。この例では、hr.employees 表の first_name、last_name および hire_date 列を含む無条件のサプリメンタル・ログ・グループを指定します。
- 適用プロセスで Oracle 以外のリモート・データベースに変更を適用する場合は、同じ表に異なる代替キー列を使用できます。apply_database_link パラメータを非 NULL 値に設定して DBMS_APPLY_ADM パッケージの SET_KEY_COLUMNS プロシージャを実行すると、Oracle 以外のリモート・データベースに適用される変更用の代替キー列を指定できます。

関連項目：

- 12-8 ページ「ソース・データベースでのサプリメンタル・ロギングの指定」
- 9-3 ページ「Oracle から Oracle 以外への環境での適用プロセスの構成」

表の代替キー列の削除

表の代替キー列を削除するには、DBMS_APPLY_ADM パッケージの SET_KEY_COLUMNS プロシージャで、column_list または column_table パラメータに NULL を指定します。表に主キーがある場合は、代入主キーを削除すると、適用プロセスではデータベースに対するローカルの変更にその表の主キーが使用されます。

たとえば、hr.employees 表の代替キー列を削除するには、次のプロシージャを実行します。

```
BEGIN
  DBMS_APPLY_ADM.SET_KEY_COLUMNS(
    object_name  => 'hr.employees',
    column_list  => NULL);
END;
/
```

Streams の競合解消の管理

この項では、表の更新の競合ハンドラを作成、指定および削除する手順について説明します。データベース上で実行中の、指定の表に変更を適用するすべての適用プロセスでは、指定した更新の競合ハンドラがローカルに使用されます。

関連項目： 第 7 章「Streams 競合解消」

更新の競合ハンドラの設定

更新の競合ハンドラを設定するには、DBMS_APPLY_ADM パッケージの SET_UPDATE_CONFLICT_HANDLER プロシージャを使用します。更新の競合解消ハンドラを作成する場合は、次のいずれかの事前作成方法を使用できます。

- OVERWRITE
- DISCARD
- MAXIMUM
- MINIMUM

たとえば、Streams 環境の dbs1.net で hr.jobs 表に対する変更を取得して、dbs2.net 接続先データベースに伝播し、そこで適用する場合を考えます。この環境の場合、アプリケーションでは両方のデータベースの hr.jobs 表に対して DML 変更を実行できますが、特定の DML 変更に競合がある場合は、dbs1.net データベースでの変更で dbs2.net データベースでの変更を常に上書きする必要があります。この環境では、dbs2.net データベースで OVERWRITE ハンドラを指定して、この目標を達成できます。

dbs2.net データベースで hr スキーマ内の hr.jobs 表について更新の競合ハンドラを指定するには、dbs2.net で次のプロシージャを実行します。

```
DECLARE
  cols DBMS_UTILITY.NAME_ARRAY;
  BEGIN
    cols(1) := 'job_title';
    cols(2) := 'min_salary';
    cols(3) := 'max_salary';
    DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
      object_name       => 'hr.jobs',
      method_name       => 'OVERWRITE',
      resolution_column => 'job_title',
      column_list       => cols);
END;
/
```

注意：

- OVERWRITE および DISCARD メソッドには resolution_column は使用されませんが、column_list 内の列の 1 つは指定する必要があります。
- 接続先データベースの column_list にあるすべての列について、ソース・データベースで条件付きのサプリメンタル・ログ・グループを指定する必要があります。この例では、dbs1.net データベースの hr.jobs 表の job_title、min_salary および max_salary 列を含む条件付きのサプリメンタル・ログ・グループを指定します。
- 競合解消では、LOB 列はサポートされません。したがって、SET_UPDATE_CONFLICT_HANDLER を実行するときには、LOB 列を column_list パラメータに含めないでください。

関連項目：

- 12-8 ページ「ソース・データベースでのサプリメンタル・ロギングの指定」
- 時間ベースの競合解消に MAXIMUM 事前作成方法を使用する Streams 環境の例については、第 23 章「複数のソース・レプリケーションの例」を参照してください。

既存の更新の競合ハンドラの変更

DBMS_APPLY_ADM パッケージの SET_UPDATE_CONFLICT_HANDLER プロシージャを実行すると、既存の更新の競合ハンドラを変更できます。既存の競合ハンドラを更新するには、そのハンドラと同じ表および解消列を指定します。

14-28 ページの「更新の競合ハンドラの設定」で作成した更新の競合ハンドラを変更するには、hr.jobs 表および解消列としての job_title 列を指定します。この更新の競合ハンドラを変更するには、異なるタイプの事前作成方法または異なる列リスト、あるいはその両方を指定できます。ただし、更新の競合ハンドラの解消列を変更する場合は、ハンドラを削除して再作成する必要があります。

たとえば、環境に変更があり、競合が発生した場合に dbs1.net からの変更を廃棄する必要があるが、dbs1.net からの以前の変更によって dbs2.net の変更が上書きされている場合を考えます。dbs2.net データベースで DISCARD ハンドラを指定すると、この目標を達成できます。

dbs2.net データベースで hr スキーマ内の hr.jobs 表について既存の更新の競合ハンドラを変更するには、次のプロシージャを実行します。

```
DECLARE
  cols DBMS_UTILITY.NAME_ARRAY;
  BEGIN
    cols(1) := 'job_title';
    cols(2) := 'min_salary';
    cols(3) := 'max_salary';
    DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
      object_name       => 'hr.jobs',
      method_name       => 'DISCARD',
      resolution_column => 'job_title',
      column_list       => cols);
END;
/
```

既存の更新の競合ハンドラの削除

DBMS_APPLY_ADM パッケージの SET_UPDATE_CONFLICT_HANDLER プロシージャを実行すると、既存の更新の競合ハンドラを削除できます。既存の競合ハンドラを削除するには、メソッドに NULL を指定し、その競合ハンドラと同じ表、列リストおよび解消列を指定します。

たとえば、14-28 ページの「更新の競合ハンドラの設定」で作成し、14-29 ページの「既存の更新の競合ハンドラの変更」で変更した更新の競合ハンドラを削除する必要があるとします。この更新の競合ハンドラを削除するには、次のプロシージャを実行します。

```
DECLARE
  cols DBMS_UTILITY.NAME_ARRAY;
  BEGIN
    cols(1) := 'job_title';
    cols(2) := 'min_salary';
    cols(3) := 'max_salary';
    DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
      object_name       => 'hr.jobs',
      method_name       => NULL,
      resolution_column => 'job_title',
      column_list       => cols);
END;
/
```

適用エラーの管理

この項では、適用エラーを再試行する手順と削除する手順について説明します。

関連項目：

- 4-34 ページ「例外キュー」
- 17-34 ページ「適用エラーのチェック」
- 17-35 ページ「適用エラーの詳細情報の表示」
- 適用エラーに考えられる原因については、4-10 ページの「表に DML 変更を適用する際の考慮事項」を参照してください。

適用エラー・トランザクションの再試行

ここでは、特定のエラー・トランザクションを再試行する方法と、適用プロセスのすべてのエラー・トランザクションを再試行する方法について説明します。適用エラー・トランザクションを再試行する前に、データベース・オブジェクトに DML または DDL 変更を加えて、1 つ以上の適用エラーの原因となった条件を訂正する必要がある場合があります。また、同じデータベース・オブジェクトに対する変更を取得するように、1 つ以上の取得プロセスを構成できます。ただし、場合によっては変更を取得しないようにすることもできます。この場合は、変更を加えるセッションのタグを取得されない値に設定できます。

関連項目： 16-24 ページ「現行セッションで生成されるタグ値の設定」

特定の適用エラー・トランザクションの再試行

適用エラーの原因となった条件を訂正した後に、DBMS_APPLY_ADM パッケージの EXECUTE_ERROR プロシージャを実行してトランザクションを再試行できます。たとえば、トランザクション識別子 5.4.312 を持つトランザクションを再試行するには、次のプロシージャを実行します。

```
BEGIN
  DBMS_APPLY_ADM.EXECUTE_ERROR(
    local_transaction_id => '5.4.312',
    execute_as_user      => false);
END;
/
```

execute_as_user が true の場合、適用プロセスでは現行ユーザーのセキュリティ・コンテキスト内でトランザクションが再実行されます。execute_as_user が false の場合、適用プロセスではトランザクションの元の受信者のセキュリティ・コンテキスト内でトランザクションが再実行されます。元の受信者とは、エラーが呼び出された時点でトランザクションを処理していたユーザーです。

どちらの場合も、トランザクションを実行するユーザーには、適用オブジェクトに対する DML 変更および DDL 変更を実行するための権限と、任意の適用ハンドラを実行するための権限が必要です。また、ユーザーには適用プロセスで使用されるキューに対するデキュー権限も必要です。

適用プロセスのすべてのエラー・トランザクションの再試行

適用プロセスについて、すべての適用エラーの原因となった条件を訂正した後に、DBMS_APPLY_ADM パッケージの EXECUTE_ALL_ERRORS プロシージャを実行して、すべてのエラー・トランザクションを再試行できます。たとえば、適用プロセス strm01_apply のすべてのエラー・トランザクションを再試行するには、次のプロシージャを実行します。

```
BEGIN
  DBMS_APPLY_ADM.EXECUTE_ALL_ERRORS(
    apply_name       => 'strm01_apply',
    execute_as_user  => false);
END;
/
```

注意： apply_name パラメータに NULL を指定した場合に、複数の適用プロセスがあると、すべての適用プロセスの全適用エラーが再試行されます。

適用エラー・トランザクションの削除

ここでは、特定のエラー・トランザクションを削除する方法と、適用プロセスのすべてのエラー・トランザクションを削除する方法について説明します。

特定の適用エラー・トランザクションの削除

エラー・トランザクションを適用しない場合は、DBMS_APPLY_ADM パッケージの DELETE_ERROR プロシージャを使用して、そのトランザクションを例外キューから削除できます。たとえば、トランザクション識別子 5.4.312 を持つトランザクションを削除するには、次のプロシージャを実行します。

```
EXEC DBMS_APPLY_ADM.DELETE_ERROR(local_transaction_id => '5.4.312');
```

適用プロセスのすべてのエラー・トランザクションの削除

エラー・トランザクションをまったく適用しない場合は、DBMS_APPLY_ADM パッケージの DELETE_ALL_ERRORS プロシージャを実行して、エラー・トランザクションをすべて削除できます。たとえば、適用プロセス strm01_apply のすべてのエラー・トランザクションを削除するには、次のプロシージャを実行します。

```
EXEC DBMS_APPLY_ADM.DELETE_ALL_ERRORS(apply_name => 'strm01_apply');
```

注意： apply_name パラメータに NULL を指定した場合に、複数の適用プロセスがあると、すべての適用プロセスの全適用エラーが削除されます。

接続先データベースでのインスタンス化 SCN の設定

インスタンス化 SCN は、接続先データベースの適用プロセスに対して、ソース・データベースで特定の SCN より後にコミットされたデータベース・オブジェクトに変更を適用するように指示します。インスタンス化 SCN を設定するには、次の方法があります。

- ソース・データベースでエクスポートし、接続先データベースでインポートして、関連するデータベース・オブジェクトのインスタンス化を実行します。この場合は、インスタンス化によって接続先データベースでデータベース・オブジェクトが作成され、そこにソース・データベースからデータが移入され、関連するインスタンス化 SCN が設定されます。
- ソース・データベースでのエクスポート中または接続先データベースでのインポート中、あるいはその両方の時点で ROWS パラメータを n に設定し、メタデータのみのエクスポート / インポートを実行します。この場合、データベース・オブジェクトはインスタンス化されますが、データはインポートされません。
- DBMS_APPLY_ADM パッケージの SET_TABLE_INSTANTIATION_SCN、SET_SCHEMA_INSTANATIATION_SCN および SET_GLOBAL_INSTANTIATION_SCN プロシージャを使用して、インスタンス化 SCN を設定します。

関連項目：

- 2-12 ページ「インスタンス化」
- 4-25 ページ「インスタンス化 SCN および非処理 SCN」

エクスポート / インポートを使用したインスタンス化 SCN の設定

この項では、エクスポート / インポートを実行してインスタンス化 SCN を設定する方法について説明します。この項の情報は、メタデータのエクスポート / インポート操作と、行をインポートするエクスポート / インポート操作の両方に適用されます。

エクスポート / インポートを使用してデータベース・オブジェクトのインスタンス化 SCN を設定するには、最初に OBJECT_CONSISTENT エクスポート・パラメータを Y に設定するか、より厳密な一貫性レベルを使用して、ソース・データベースでエクスポートします。次に、接続先データベースで、STREAMS_INSTANTIATION インポート・パラメータを Y に設定してインポートします。

注意 :

- インポートを実行する接続先データベースに、データベース・オブジェクトについて非 NULL のインスタンス化 SCN がすでに存在する場合は、インポートを実行しても、そのデータベース・オブジェクトのインスタンス化 SCN は更新されません。
- Streams をインスタンス化するためのエクスポート中は、エクスポート対象のオブジェクトに対して DDL 変更が行われないように注意してください。
- エクスポート・データベースからエクスポートされた表の表サプリメンタル・ロギング仕様は、インポート・データベースで表がインポートされる際に保持されます。

ここでは、各種のエクスポート / インポート操作用のインスタンス化 SCN 設定について説明します。これらの項の説明は、準備済みの表を対象としています。準備済みの表とは、DBMS_CAPTURE_ADM パッケージの PREPARE_TABLE_INSTANTIATION、PREPARE_SCHEMA_INSTANTIATION または PREPARE_GLOBAL_INSTANTIATION プロシージャを使用して、インスタンス化の準備を完了している表です。インポート中にインスタンス化 SCN を設定するには、エクスポート前に表を準備済みにする必要があります。ただし、データベースとスキーマの場合は、インポート中にインスタンス化 SCN が設定されるようにエクスポート前に準備する必要はありません。

全データベースのエクスポートと全データベースのインポート

全データベースのエクスポートと全データベースのインポートでは、インポート・データベースで次のインスタンス化 SCN が設定されます。

- データベースの、またはグローバルなインスタンス化 SCN
- インポートされるユーザーごとの、スキーマのインスタンス化 SCN
- インポートされる準備済みの表ごとの、表のインスタンス化 SCN

全データベースまたはユーザー・エクスポートおよびユーザー・インポート

全データベースまたはユーザー・エクスポートとユーザー・インポートでは、インポート・データベースで次のインスタンス化 SCN が設定されます。

- インポートされるユーザーごとの、スキーマのインスタンス化 SCN
- インポートされる準備済みの表ごとの、表のインスタンス化 SCN

全データベース、ユーザーまたは表エクスポートおよび表インポート

1 つ以上の表を含むエクスポートと表のインポートでは、インポート・データベースでインポートされる準備済みの表ごとに、表のインスタンス化 SCN が設定されます。

関連項目：

- エクスポート / インポートの使用方法は、11-8 ページの「Streams に関連するエクスポート・ユーティリティとインポート・ユーティリティのパラメータ設定」および『Oracle9*i* データベース・ユーティリティ』を参照してください。
- Streams 環境の構成時に、エクスポート / インポート操作を実行してインスタンス化 SCN を設定する方法の詳細は、11-14 ページの「取得ベースの Streams 環境の構成」を参照してください。
- 12-10 ページ「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」

DBMS_APPLY_ADM パッケージを使用したインスタンス化 SCN の設定

DBMS_APPLY_ADM パッケージの次のいずれかのプロシージャを使用すると、指定した表、指定したスキーマまたはデータベース全体について、接続先データベースでインスタンス化 SCN を設定できます。

- SET_TABLE_INSTANTIATION_SCN
- SET_SCHEMA_INSTANTIATION_SCN
- SET_GLOBAL_INSTANTIATION_SCN

SET_SCHEMA_INSTANTIATION_SCN を使用してスキーマのインスタンス化 SCN を設定する場合は、SET_TABLE_INSTANTIATION_SCN を使用して、スキーマ内の表ごとにインスタンス化 SCN を設定する必要があります。同様に、SET_GLOBAL_INSTANTIATION_SCN を使用してデータベースのインスタンス化 SCN を設定する場合は、SET_SCHEMA_INSTANTIATION_SCN を使用して、データベース内のスキーマごとにインスタンス化 SCN を設定する必要があります。

表 14-1 に、各プロシージャとインスタンス化 SCN を設定する文のタイプを示します。

表 14-1 インスタンス化 SCN を設定するプロシージャと、その対象となる文

プロシージャ	インスタンス化 SCN の設定対象	例
SET_TABLE_INSTANTIATION_SCN	CREATE TABLE を除く、表に対する DML 文と DDL 文 表の索引と表のトリガーに対する DDL 文	UPDATE ALTER TABLE DROP TABLE 表に対する CREATE、ALTER または DROP INDEX 表に対する CREATE、ALTER または DROP TRIGGER
SET_SCHEMA_INSTANTIATION_SCN	CREATE USER を除く、ユーザーに対する DDL 文 表レベルのインスタンス化 SCN によって処理される DDL 文を除き、PUBLIC 以外の所有者を持つ全データベース・オブジェクトに対する DDL 文	CREATE TABLE ALTER USER DROP USER CREATE PROCEDURE
SET_GLOBAL_INSTANTIATION_SCN	所有者を持たない、ユーザー以外のデータベース・オブジェクトに対する DDL 文 PUBLIC に所有されるデータベース・オブジェクトに対する DDL 文 CREATE USER 文	CREATE USER CREATE TABLESPACE

次の例では、ソース・データベース hrdb1.net で次のプロシージャを実行することで、hrdb2.net データベースで hr.departments 表のインスタンス化 SCN が現行の SCN に設定されます。

```
DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@HRDB2.NET(
    source_object_name    => 'hr.departments',
    source_database_name  => 'hrdb1.net',
    instantiation_scn     => iscn);
END;
/
```

注意：

- 関連するインスタンス化 SCN が存在しないと、適用中にエラーが呼び出されます。
- SET_SCHEMA_INSTANTIATION_SCN プロシージャでは、スキーマ内の表についてはインスタンス化 SCN が設定されません。
- SET_GLOBAL_INSTANTIATION_SCN プロシージャでは、データベース内のスキーマについてはインスタンス化 SCN が設定されません。
- 適用プロセスで Oracle 以外のリモート・データベースに変更が適用される場合は、インスタンス化 SCN を設定するときに、apply_database_link パラメータをリモートの適用に使用するデータベース・リンクに設定します。

関連項目：

- Streams 環境の構成時にインスタンス化 SCN を設定する場合の詳細は、第 11 章「Streams 環境の構成」を参照してください。
- SET_TABLE_INSTANTIATION_SCN プロシージャの詳細な例は、第 22 章「単一ソースの異機種間レプリケーションの例」を参照してください。
- DDL LCR に使用できるインスタンス化 SCN の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_APPLY_ADM パッケージの説明を参照してください。
- 9-6 ページ「Oracle から Oracle 以外への環境でのインスタンス化」

15

ルールおよびルールベースの変換の管理

Streams 環境では、取得プロセス、伝播および適用プロセスの動作がルールを使用して制御されます。また、Streams 環境では、ルールベースの変換を使用して、ルールが TRUE に評価される場合に発生するイベントが変更されます。変換は、イベントの取得中、伝播中または適用中に発生させることができます。この章では、ルール・セット、ルールおよびルールベースの変換の管理手順について説明します。

この章の内容は次のとおりです。

- ルール・セットとルールの管理
- 評価コンテキスト、ルール・セットおよびルールに対する権限の管理
- ルールベースの変換の管理

この項で説明する各タスクは、特に明記されていないかぎり、適切な権限を付与されている Streams 管理者が完了する必要があります。

関連項目：

- 第 5 章「ルール」
- 第 6 章「Streams でのルールの使用方法」
- 6-23 ページ「ルールベースの変換」
- 第 24 章「ルールベースのアプリケーションの例」
- 11-2 ページ「Streams 管理者の構成」

ルール・セットとルールの管理

ルールまたはルール・セットの変更は、そのルールまたはルール・セットを使用する Streams の取得プロセス、伝播および適用プロセスを停止せずに行うことができます。Streams はコミットされた直後の変更を検出します。新規バージョンのルールまたはルール・セットが使用されるイベントについて厳密に制御する必要がある場合は、関連する取得プロセスおよび適用プロセスを停止し、関連する伝播ジョブを無効化したうえで、ルールまたはルール・セットを変更し、停止したプロセスおよび伝播ジョブを再起動する必要があります。

この項では、次のタスクの手順について説明します。

- ルール・セットの作成
- ルールの作成
- ルール・セットへのルールの追加
- ルールの変更
- システム作成ルールの変更
- ルール・セットからのルールの削除
- ルールの削除
- ルール・セットの削除

関連項目：

- 12-13 ページ「取得プロセスの停止」
- 13-16 ページ「伝播ジョブの無効化」
- 14-7 ページ「適用プロセスの停止」

ルール・セットの作成

ここでは、DBMS_RULE_ADM パッケージの CREATE_RULE_SET プロシージャを実行してルール・セットを作成する例について説明します。

```
BEGIN
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name       => 'strmadmin.hr_capture_rules',
    evaluation_context  => 'SYS.STREAMS$_EVALUATION_CONTEXT');
END;
/
```

このプロシージャを実行すると、次のアクションが実行されます。

- strmadmin スキーマにルール・セット hr_capture_rules が作成されます。同じ名前と所有者を持つルール・セットが存在することは許されません。
- ルール・セットが、SYS.STREAMS$_EVALUATION_CONTEXT 評価コンテキストに関連付けられます。これは、Oracle が提供する Streams 用の評価コンテキストです。

また、DBMS_STREAMS_ADM パッケージの次のプロシージャを使用すると、Streams の取得プロセス、伝播または適用プロセスにルール・セットが存在しない場合に自動的にルール・セットを作成できます。

- ADD_GLOBAL_PROPAGATION_RULES
- ADD_GLOBAL_RULES
- ADD_SCHEMA_PROPAGATION_RULES
- ADD_SCHEMA_RULES
- ADD_SUBSET_RULES
- ADD_TABLE_PROPAGATION_RULES
- ADD_TABLE_RULES

関連項目：

- 12-3 ページ「DBMS_STREAMS_ADM を使用した取得プロセスの作成例」
- 13-8 ページ「DBMS_STREAMS_ADM を使用した伝播の作成例」
- 14-3 ページ「DBMS_STREAMS_ADM を使用した適用プロセスの作成例」

ルールの作成

ここでは、DBMS_RULE_ADM パッケージの CREATE_RULE プロシージャを実行してルールを作成する例について説明します。

```
BEGIN
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name          => 'strmadmin.hr_dml',
    condition          => ' :dml.get_object_owner() = ''HR'' ',
    evaluation_context => NULL);
END;
/
```

このプロシージャを実行すると、次のアクションが実行されます。

- strmadmin スキーマにルール hr_dml が作成されます。同じ名前と所有者を持つルールが存在することは許されません。
- hr スキーマ内の表に対する DML 変更について、TRUE と評価される条件が作成されます。

この例では、ルールの評価コンテキストは指定されていません。したがって、このルールは追加先となるルール・セットの評価コンテキストを継承します。または、DBMS_RULE_ADM.ADD_RULE プロシージャを実行してルール・セットに追加するときに、明示的に評価コンテキストを割り当てます。この時点では、どのルール・セットにも属していないため、このルールは評価できません。

また、DBMS_STREAMS_ADM パッケージの次のプロシージャを使用すると、自動的にルールを作成してルール・セットに追加できます。

- ADD_GLOBAL_PROPAGATION_RULES
- ADD_GLOBAL_RULES
- ADD_SCHEMA_PROPAGATION_RULES
- ADD_SCHEMA_RULES
- ADD_SUBSET_RULES
- ADD_TABLE_PROPAGATION_RULES
- ADD_TABLE_RULES

関連項目：

- 12-3 ページ「DBMS_STREAMS_ADM を使用した取得プロセスの作成例」
- 13-8 ページ「DBMS_STREAMS_ADM を使用した伝播の作成例」
- 14-3 ページ「DBMS_STREAMS_ADM を使用した適用プロセスの作成例」

ルール・セットへのルールの追加

ここでは、DBMS_RULE_ADM パッケージの ADD_RULE プロシージャを実行して hr_capture_rules ルール・セットに hr_dml ルールを追加する例について説明します。

```
BEGIN
  DBMS_RULE_ADM.ADD_RULE(
    rule_name          => 'strmadmin.hr_dml',
    rule_set_name      => 'strmadmin.hr_capture_rules',
    evaluation_context => NULL);
END;
/
```

この例では、ADD_RULE プロシージャの実行時には評価コンテキストは指定されていません。したがって、ルールに独自の評価コンテキストがなければ、hr_capture_rules ルール・セットの評価コンテキストを継承します。ルール・セットに指定した以外の評価コンテキストをルールに使用する場合は、ADD_RULE プロシージャの実行時に evaluation_context パラメータをその評価コンテキストに設定できます。

ルールの変更

DBMS_RULE_ADM パッケージの ALTER_RULE プロシージャを使用すると、既存のルールを変更できます。特に、このプロシージャを使用すると次のことができます。

- ルールの条件の変更
- ルールの評価コンテキストの変更
- ルールの評価コンテキストの削除
- ルールのアクション・コンテキストの変更
- ルールのアクション・コンテキストの削除
- ルールに関するコメントの変更
- ルールに関するコメントの削除

たとえば、15-3 ページの「ルールの作成」で作成したルールの条件を変更する必要がある場合を考えます。既存の hr_dml ルールの条件は、hr スキーマ内の任意のオブジェクトに対する任意の DML 変更について TRUE と評価されます。このスキーマ内の employees 表に対する変更を除外する必要がある場合は、hr.employees 表に対する DML 変更については FALSE と評価されるが、このスキーマの他の表に対する DML 変更については引き続き TRUE と評価されるように、このルールを変更できます。次のプロシージャを実行すると、ルールが前述のとおり変更されます。

```
BEGIN
  DBMS_RULE_ADM.ALTER_RULE(
    rule_name          => 'strmadmin.hr_dml',
    condition          => ' :dml.get_object_owner() = ''HR'' AND NOT ' ||
                          ' :dml.get_object_name() = ''EMPLOYEES'' ',
    evaluation_context => NULL);
END;
/
```

注意：

- ルールの条件を変更すると、そのルールを含むすべてのルール・セットに影響します。
- ルールのアクション・コンテキストを保持しながらルールを変更するには、ALTER_RULE プロシージャの action_context パラメータに NULL を指定します。NULL は、action_context パラメータのデフォルト値です。

システム作成ルールの変更

システム作成ルールとは、DBMS_STREAMS_ADM パッケージのプロシージャを実行して作成するルールです。システム作成ルールにルールベースの変換を使用する必要がある場合は、そのルールのアクション・コンテキストを変更してルールベースの変換を追加できます。

また、DBMS_STREAMS_ADM パッケージでは必要なルール条件を持つルールを作成できない場合は、次の一般的な手順に従って、システム作成ルールに基づく条件を持つ新規ルールを作成できます。

1. システム作成ルールのルール条件をコピーします。DBA_STREAMS_TABLE_RULES、DBA_STREAMS_SCHEMA_RULES または DBA_STREAMS_GLOBAL_RULES データ・ディクショナリ・ビューを問い合せると、システム作成ルールのルール条件を表示できます。
2. コピーしたルール条件を変更して、新規ルールを作成します。
3. 新規ルールを Streams の取得プロセス、伝播または適用プロセスのルール・セットに追加します。
4. 元のルールが不要になった場合は、DBMS_STREAMS_ADM パッケージの REMOVE_RULE プロシージャを使用して削除します。

関連項目：

- 6-23 ページ「ルールベースの変換」
- Streams に関連するデータ・ディクショナリ・ビューの詳細は、第 17 章「Streams 環境の監視」を参照してください。

ルール・セットからのルールの削除

ここでは、DBMS_RULE_ADM パッケージの REMOVE_RULE プロシージャを実行して hr_capture_rules ルール・セットから hr_dml ルールを削除する例について説明します。

```
BEGIN
  DBMS_RULE_ADM.REMOVE_RULE(
    rule_name     => 'strmadmin.hr_dml',
    rule_set_name => 'strmadmin.hr_capture_rules');
END;
/
```

REMOVE_RULE プロシージャの実行後も、ルールはデータベース内に存在し、他のルール・セットに含まれていた場合は、そのルール・セットに残ります。

ルールの削除

ここでは、DBMS_RULE_ADM パッケージの DROP_RULE プロシージャを実行してデータベースから hr_dml ルールを削除する例について説明します。

```
BEGIN
  DBMS_RULE_ADM.DROP_RULE(
    rule_name => 'strmadmin.hr_dml',
    force     => false);
END;
/
```

この例では、DROP_RULE プロシージャの force パラメータが false に設定されています。これはデフォルト設定です。したがって、このルールが 1 つ以上のルール・セットに含まれている場合は削除できません。force パラメータが true に設定されている場合は、ルールがデータベースから削除され、それを含むルール・セットから自動的に削除されます。

ルール・セットの削除

ここでは、DBMS_RULE_ADM パッケージの DROP_RULE_SET プロシージャを実行してデータベースから hr_capture_rules ルール・セットを削除する例について説明します。

```
BEGIN
  DBMS_RULE_ADM.DROP_RULE_SET(
    rule_set_name => 'strmadmin.hr_capture_rules',
    delete_rules  => false);
END;
/
```

この例では、DROP_RULE_SET プロシージャの delete_rules パラメータが false に設定されています。これはデフォルト設定です。したがって、このルール・セットにルールが含まれている場合、そのルールは削除されません。delete_rules パラメータを true に設定すると、他のルール・セットには含まれておらず、指定したルール・セットに含まれているルールが、データベースから自動的に削除されます。ルール・セットに含まれているルールが、他の 1 つ以上のルール・セットにも含まれている場合、そのルールは削除されません。

評価コンテキスト、ルール・セットおよびルールに対する権限の管理

この項では、次のタスクの手順について説明します。

- 評価コンテキスト、ルール・セットおよびルールに対するシステム権限の付与
- 評価コンテキスト、ルール・セットまたはルールに対するオブジェクト権限の付与
- 評価コンテキスト、ルール・セットおよびルールに対するシステム権限の取消し
- 評価コンテキスト、ルール・セットまたはルールに対するオブジェクト権限の取消し

関連項目：

- 5-14 ページ「ルールに関連するデータベース・オブジェクトと権限」
- 『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の、DBMS_RULE_ADM パッケージの GRANT_SYSTEM_PRIVILEGE および GRANT_OBJECT_PRIVILEGE プロシージャの項を参照してください。

評価コンテキスト、ルール・セットおよびルールに対するシステム権限の付与

DBMS_RULE_ADM パッケージの GRANT_SYSTEM_PRIVILEGE プロシージャを使用すると、評価コンテキスト、ルール・セットおよびルールに対するシステム権限をユーザーとロールに付与できます。これらの権限により、ユーザーは独自のスキーマ内で、または権限の ANY バージョンが付与されている場合は任意のスキーマ内で、これらのオブジェクトを作成、変更、実行または削除できます。

たとえば、strmadmin ユーザーに独自スキーマ内で評価コンテキストを作成するための権限を付与するには、権限の付与およびユーザーの変更を行うことができるユーザーとして接続し、次のように入力します。

```
BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => SYS.DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee      => 'strmadmin',
    grant_option => false);
END;
/
```

この例では、GRANT_SYSTEM_PRIVILEGE プロシージャの grant_option パラメータが false に設定されています。これはデフォルト設定です。したがって、strmadmin ユーザーは、他のユーザーやロールには CREATE_EVALUATION_CONTEXT_OBJ システム権限を付与できません。grant_option パラメータを true に設定した場合、strmadmin ユーザーはこのシステム権限を他のユーザーに付与できます。

評価コンテキスト、ルール・セットまたはルールに対するオブジェクト権限の付与

DBMS_RULE_ADM パッケージの GRANT_OBJECT_PRIVILEGE プロシージャを使用すると、特定の評価コンテキスト、ルール・セットまたはルールに対するオブジェクト権限を付与できます。これらの権限を付与されたユーザーは、指定されたオブジェクトを変更または実行できます。

たとえば、hr ユーザーに、strmadmin スキーマ内のルール・セット hr_capture_rules を変更するための権限と実行するための権限の両方を付与するには、次のように入力します。

```
BEGIN
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege    => SYS.DBMS_RULE_ADM.ALL_ON_RULE_SET,
    object_name  => 'strmadmin.hr_capture_rules',
    grantee      => 'hr',
    grant_option => false);
END;
/
```

この例では、GRANT_OBJECT_PRIVILEGE プロシージャの grant_option パラメータが false に設定されています。これはデフォルト設定です。したがって、hr ユーザーは、指定されたルール・セットに対する ALL_ON_RULE_SET オブジェクト権限を他のユーザーやロールに付与できません。grant_option パラメータを true に設定した場合、hr ユーザーはこのオブジェクト権限を他のユーザーに付与できます。

評価コンテキスト、ルール・セットおよびルールに対するシステム権限の取消し

DBMS_RULE_ADM パッケージの REVOKE_SYSTEM_PRIVILEGE プロシージャを使用すると、評価コンテキスト、ルール・セットおよびルールに対するシステム権限を取り消すことができます。

たとえば、strmadmin ユーザーが独自スキーマ内で評価コンテキストを作成するための権限を取り消すには、権限の付与およびユーザーの変更を行うことができるユーザーとして接続し、次のように入力します。

```
BEGIN
  DBMS_RULE_ADM.REVOKE_SYSTEM_PRIVILEGE(
    privilege    => SYS.DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    revokee      => 'strmadmin');
END;
/
```

評価コンテキスト、ルール・セットまたはルールに対するオブジェクト権限の取消し

DBMS_RULE_ADM パッケージの REVOKE_OBJECT_PRIVILEGE プロシージャを使用すると、特定の評価コンテキスト、ルール・セットまたはルールに対するオブジェクト権限を取り消すことができます。

たとえば、hr ユーザーから、strmadmin スキーマ内のルール・セット hr_capture_rules を変更するための権限と実行するための権限の両方を取り消すには、次のように入力します。

```
BEGIN
  DBMS_RULE_ADM.REVOKE_OBJECT_PRIVILEGE(
    privilege    => SYS.DBMS_RULE_ADM.ALL_ON_RULE_SET,
    object_name  => 'strmadmin.hr_capture_rules',
    revokee      => 'hr');
END;
/
```

ルールベースの変換の管理

Streams では、ルールベースの変換は、ルールが TRUE に評価される場合に発生するイベントの変更です。ルールベースの変換を指定するには、ルール・アクション・コンテキストを使用します。アクション・コンテキスト内でルールベースの変換を指定する名前 / 値ペアでは、名前は STREAMS$_TRANSFORM_FUNCTION で、値は変換を実行する PL/SQL ファンクションの名前を含む SYS.AnyData インスタンスです。

この項では、次のタスクの手順について説明します。

- ルールベースの変換の作成
- ルールベースの変換の変更
- ルールベースの変換の削除

注意：

- ルールのアクション・コンテキストには、自動ロック・メカニズムはありません。したがって、アクション・コンテキストが同時に複数のセッションで更新されないことを確認してください。
- DDL LCR に対してルールベースの変換を実行する場合は、他の変更と一致するように DDL LCR 内で DDL テキストの変更が必要になることがあります。たとえば、ルールベースの変換によって DDL LCR 内の表名が変更になる場合は、DDL テキスト内の表名も同様に変更する必要があります。

関連項目： 6-23 ページ「ルールベースの変換」

ルールベースの変換の作成

ルールベースの変換に含まれるファンクションには、次のシグネチャが必要です。

```
FUNCTION user_function (
   parameter_name   IN  SYS.AnyData)
RETURN SYS.AnyData;
```

user_function はファンクション名で、*parameter_name* はファンクションに渡されるパラメータの名前です。ファンクションに渡されるパラメータは LCR を SYS.AnyData にカプセル化したもので、ファンクションではそれを戻す必要があります。

次の手順では、ルールベースの変換を作成する一般的な手順について説明します。

1. 変換を実行する PL/SQL ファンクションを作成します。

   注意： 変換ファンクションで例外が呼び出されないことを確認してください。例外が呼び出されると、取得プロセス、伝播または適用プロセスが無効化される場合があり、それらを進行させるために変換ファンクションの訂正が必要になります。

   次の例では、hr スキーマ内に、departments 表の department_name 列の値を Executive から Management に変更するファンクション executive_to_management が作成されます。会社の事業所の 1 つがこの部門に異なる名称を使用している場合に、このような変換が必要になることがあります。

```
CONNECT hr/hr

CREATE OR REPLACE FUNCTION hr.executive_to_management(in_any IN SYS.AnyData)
RETURN SYS.AnyData
IS
  lcr SYS.LCR$_ROW_RECORD;
  rc  NUMBER;
  ob_owner VARCHAR2(30);
  ob_name VARCHAR2(30);
  dep_value_anydata SYS.AnyData;
  dep_value_varchar2 VARCHAR2(30);
BEGIN
  -- Get the type of object
  -- Check if the object type is SYS.LCR$_ROW_RECORD
  IF in_any.GETTYPENAME='SYS.LCR$_ROW_RECORD' THEN
    -- Put the row LCR into lcr
    rc := in_any.GETOBJECT(lcr);
    -- Get the object owner and name
    ob_owner := lcr.GET_OBJECT_OWNER();
    ob_name := lcr.GET_OBJECT_NAME();
    -- Check for the hr.departments table
    IF ob_owner = 'HR' AND ob_name = 'DEPARTMENTS' THEN
      -- Get the old value of the department_name column in the LCR
      dep_value_anydata := lcr.GET_VALUE('old','DEPARTMENT_NAME');
      IF dep_value_anydata IS NOT NULL THEN
        -- Put the column value into dep_value_varchar2
        rc := dep_value_anydata.GETVARCHAR2(dep_value_varchar2);
        -- Change a value of Executive in the column to Management
        IF (dep_value_varchar2 = 'Executive') THEN
          lcr.SET_VALUE('OLD','DEPARTMENT_NAME',
            SYS.ANYDATA.CONVERTVARCHAR2('Management'));
        END IF;
```

```
      END IF;
      -- Get the new value of the department_name column in the LCR
      dep_value_anydata := lcr.GET_VALUE('new','DEPARTMENT_NAME');
      IF dep_value_anydata IS NOT NULL THEN
        -- Put the column value into dep_value_varchar2
        rc := dep_value_anydata.GETVARCHAR2(dep_value_varchar2);
        -- Change a value of Executive in the column to Management
        IF (dep_value_varchar2 = 'Executive') THEN
          lcr.SET_VALUE('new','DEPARTMENT_NAME',
            SYS.ANYDATA.CONVERTVARCHAR2('Management'));
        END IF;
      END IF;
    RETURN SYS.ANYDATA.CONVERTOBJECT(lcr);
    END IF;
  END IF;
RETURN in_any;
END;
/
```

2. Streams 管理者に hr.executive_to_management ファンクションの EXECUTE 権限を付与します。

```
GRANT EXECUTE ON hr.executive_to_management TO strmadmin;
```

3. hr.departments 表の DML 操作についてサブセット・ルールを作成します。このサブセット・ルールでは、手順 1 で作成した変換を使用します。

   ルールベースの変換を使用する場合、サブセット・ルールは不要です。この例では、複数の名前 / 値ペアを持つアクション・コンテキストの具体例を示すために、サブセット・ルールを使用しています。複数の名前 / 値ペアを持つアクション・コンテキストを変更する場合には、注意する必要があります。15-18 ページの「ルールベースの変換の変更」を参照してください。

   この例では、データベース dbs1.net に適用プロセスのサブセット・ルールが作成されます。これらのルールが TRUE と評価されるのは、LCR に hr.departments 表のうち location_id が 1700 の 1 行に対する DML 変更が含まれている場合です。この例では、SYS.AnyData キュー strm01_queue がデータベースに存在することを想定しています。

これらのルールを作成するには、Streams 管理者として接続して次の ADD_SUBSET_RULES プロシージャを実行します。

```
CONNECT strmadmin/strmadminpw

BEGIN
  DBMS_STREAMS_ADM.ADD_SUBSET_RULES(
    table_name               =>  'hr.departments',
    dml_condition            =>  'location_id=1700',
    streams_type             =>  'apply',
    streams_name             =>  'strm01_apply',
    queue_name               =>  'strm01_queue',
    include_tagged_lcr       =>  false,
    source_database          =>  'dbs1.net');
END;
/
```

注意：

- Streams 管理者がルールとルール・セットを作成するには、CREATE_RULE_SET_OBJ（または CREATE_ANYRULE_SET_OBJ）および CREATE_RULE_OBJ（または CREATE_ANY_RULE_OBJ）システム権限が必要です。これらの権限を付与するには、DBMS_RULE_ADM パッケージの GRANT_SYSTEM_PRIVILEGE プロシージャを使用します。
- この例では、DBMS_STREAMS_ADM パッケージを使用してルールが作成されます。また、DBMS_RULE_ADM パッケージを使用してルールを作成し、ルール・セットに追加し、ルールベースの変換を指定することもできます。この方法の例については、22-36 ページの「単一データベースからのデータを共有する柔軟な構成」を参照してください。

4. 次の問合せを実行して、システム作成ルールの名前を判断します。

```
SELECT RULE_NAME, SUBSETTING_OPERATION FROM DBA_STREAMS_TABLE_RULES
  WHERE TABLE_NAME='DEPARTMENTS' AND DML_CONDITION='location_id=1700';
```

この問合せでは、次のような出力が表示されます。

```
RULE_NAME                      SUBSET
------------------------------ ------
DEPARTMENTS5                   INSERT
DEPARTMENTS6                   UPDATE
DEPARTMENTS7                   DELETE
```

注意： この情報を取得するには、ADD_SUBSET_RULES を実行するときに OUT パラメータを使用する方法もあります。

これらはサブセット・ルールであるため、そのうちの 2 つには内部変換を実行する非 NULL のアクション・コンテキストが含まれています。

- サブセット化条件 INSERT を持つルールには、更新によって location_id 列の値が他の値から 1700 に変更される場合に、更新を挿入に変換する内部変換が含まれています。内部変換は、挿入には影響しません。
- サブセット化条件 DELETE を持つルールには、更新によって location_id 列の値が 1700 から他の値に変更される場合に、更新を削除に変換する内部変換が含まれています。内部変換は、削除には影響しません。

この例では、次の問合せを実行すると、ルール DEPARTMENTS5 および DEPARTMENTS7 が非 NULL のアクション・コンテキストを持ち、ルール DEPARTMENTS6 が NULL のアクション・コンテキストを持つことを確認できます。

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A13
COLUMN ACTION_CONTEXT_NAME HEADING 'Action Context Name' FORMAT A27
COLUMN ACTION_CONTEXT_VALUE HEADING 'Action Context Value' FORMAT A26

SELECT
    RULE_NAME,
    AC.NVN_NAME ACTION_CONTEXT_NAME,
    AC.NVN_VALUE.ACCESSVARCHAR2() ACTION_CONTEXT_VALUE
  FROM DBA_RULES R, TABLE(R.RULE_ACTION_CONTEXT.ACTX_LIST) AC
  WHERE RULE_NAME IN ('DEPARTMENTS5','DEPARTMENTS6','DEPARTMENTS7');
```

この問合せでは、次のような出力が表示されます。

```
Rule Name     Action Context Name         Action Context Value
------------- --------------------------- -------------------------
DEPARTMENTS5  STREAMS$_ROW_SUBSET         INSERT
DEPARTMENTS7  STREAMS$_ROW_SUBSET         DELETE
```

DEPARTMENTS6 ルールは、アクション・コンテキストが NULL であるため出力には表示されません。

5. 各サブセット・ルールのアクション・コンテキストを変更し、ルールベースの変換用の名前 / 値ペアを追加します。他のユーザーが同時にアクション・コンテキストを変更中でないことを確認してください。名前 / 値ペアには、STREAMS$_TRANSFORM_FUNCTION 名を使用する必要があります。

   DEPARTMENTS5 ルールにルールベースの変換を追加します。次の文では、新規のペアを追加する前に、アクション・コンテキストが変数に選択されて、アクション・コンテキスト内の既存の名前 / 値ペアが保たれます。

```
DECLARE
  action_ctx       SYS.RE$NV_LIST;
  ac_name          VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
BEGIN
  action_ctx := SYS.RE$NV_LIST(SYS.RE$NV_ARRAY());
  SELECT RULE_ACTION_CONTEXT
    INTO action_ctx
    FROM DBA_RULES R
    WHERE RULE_OWNER='STRMADMIN' AND RULE_NAME='DEPARTMENTS5';
  action_ctx.ADD_PAIR(ac_name,
                SYS.ANYDATA.CONVERTVARCHAR2('hr.executive_to_management'));
  DBMS_RULE_ADM.ALTER_RULE(
    rule_name       =>  'strmadmin.departments5',
    action_context  => action_ctx);
END;
/
```

   DEPARTMENTS6 ルールにルールベースの変換を追加します。この文では、DEPARTMENTS6 ルールのアクション・コンテキストが NULL のため、問い合せる必要はありません。

```
DECLARE
  action_ctx       SYS.RE$NV_LIST;
  ac_name          VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
BEGIN
  action_ctx := SYS.RE$NV_LIST(SYS.RE$NV_ARRAY());
  action_ctx.ADD_PAIR(ac_name,
                SYS.ANYDATA.CONVERTVARCHAR2('hr.executive_to_management'));
  DBMS_RULE_ADM.ALTER_RULE(
    rule_name       =>  'strmadmin.departments6',
    action_context  => action_ctx);
END;
/
```

   DEPARTMENTS7 ルールにルールベースの変換を追加します。この文では、新規の名前 / 値ペアを追加する前に、既存のアクション・コンテキストを問い合せて、それを変数に挿入します。

```
DECLARE
  action_ctx       SYS.RE$NV_LIST;
  ac_name          VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
BEGIN
  action_ctx := SYS.RE$NV_LIST(SYS.RE$NV_ARRAY());
  SELECT RULE_ACTION_CONTEXT
    INTO action_ctx
    FROM DBA_RULES R
    WHERE RULE_OWNER='STRMADMIN' AND RULE_NAME='DEPARTMENTS7';
  action_ctx.ADD_PAIR(ac_name,
                SYS.ANYDATA.CONVERTVARCHAR2('hr.executive_to_management'));
  DBMS_RULE_ADM.ALTER_RULE(
    rule_name       =>  'strmadmin.departments7',
    action_context  => action_ctx);
END;
/
```

これらのルールのアクション・コンテキスト内の名前 / 値ペアを表示する問合せを実行すると、DEPARTMENTS6 ルールを含め、ルールごとにルールベースの変換用の名前 / 値ペアが表示されます。

```
SELECT
    RULE_NAME,
    AC.NVN_NAME ACTION_CONTEXT_NAME,
    AC.NVN_VALUE.ACCESSVARCHAR2() ACTION_CONTEXT_VALUE
  FROM DBA_RULES R, TABLE(R.RULE_ACTION_CONTEXT.ACTX_LIST) AC
  WHERE RULE_NAME IN ('DEPARTMENTS5','DEPARTMENTS6','DEPARTMENTS7');
```

この問合せでは、次のような出力が表示されます。

```
Rule Name    Action Context Name         Action Context Value
------------ --------------------------- --------------------------
DEPARTMENTS5 STREAMS$_ROW_SUBSET         INSERT
DEPARTMENTS5 STREAMS$_TRANSFORM_FUNCTION hr.executive_to_management
DEPARTMENTS6 STREAMS$_TRANSFORM_FUNCTION hr.executive_to_management
DEPARTMENTS7 STREAMS$_ROW_SUBSET         DELETE
DEPARTMENTS7 STREAMS$_TRANSFORM_FUNCTION hr.executive_to_management
```

関連項目： この例に使用したルールのタイプの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ルールベースの変換の変更

ルールベースの変換を変更するには、変換ファンクションを編集する方法と、異なる変換ファンクションを実行するようにアクション・コンテキストを編集する方法があります。この例では、異なるファンクションを実行するようにアクション・コンテキストを編集します。ファンクション自体を編集する場合、アクション・コンテキストを変更する必要はありません。

この例では、DEPARTMENTS5 ルールについてルールベースの変換を変更します。そのために、最初にこのルールのアクション・コンテキストから STREAMS$_TRANSFORM_FUNCTION 名を持つ名前 / 値ペアを削除し、次に異なる名前 / 値ペアを追加します。このルールベースの変換は、15-11 ページの「ルールベースの変換の作成」の例で DEPARTMENTS5 ルールに追加したものです。

アクション・コンテキストに、変換を指定する名前 / 値ペアの他にも名前 / 値ペアが含まれている場合、アクション・コンテキストを変更するときには、変換に無関係な名前 / 値ペアを変更または削除しないように注意してください。

Streams の場合、サブセット・ルールではアクション・コンテキスト内の名前 / 値ペアを使用して、特定の状況で UPDATE 操作を INSERT および DELETE 操作に変換する内部変換が実行されます。この種の変換は、行の移行と呼ばれます。サブセット・ルール用に新規の変換を指定するか、既存の変換を変更する場合は、行の移行を実行する名前 / 値ペアを保つように注意してください。

> **関連項目：** 4-12 ページ「行の移行」

次の手順に従って、ルールベースの変換を変更します。

1. 次の問合せを実行すると、ルールのアクション・コンテキスト内の名前 / 値ペアをすべて表示できます。

```
COLUMN ACTION_CONTEXT_NAME HEADING 'Action Context Name' FORMAT A30
COLUMN ACTION_CONTEXT_VALUE HEADING 'Action Context Value' FORMAT A26

SELECT
    AC.NVN_NAME ACTION_CONTEXT_NAME,
    AC.NVN_VALUE.ACCESSVARCHAR2() ACTION_CONTEXT_VALUE
  FROM DBA_RULES R, TABLE(R.RULE_ACTION_CONTEXT.ACTX_LIST) AC
  WHERE RULE_NAME = 'DEPARTMENTS5';
```

この問合せでは、次のような出力が表示されます。

```
Action Context Name            Action Context Value
------------------------------ --------------------------
STREAMS$_ROW_SUBSET            INSERT
STREAMS$_TRANSFORM_FUNCTION    hr.executive_to_management
```

2. DEPARTMENTS5 ルールの場合、変換ファンクションは executive_to_management です。この変換ファンクションを変更するには、この手順の最初に、このファンクション名を含む名前 / 値ペアを DEPARTMENTS5 ルールのアクション・コンテキストから削除します。次に、この手順で、新規のファンクション名を含む名前 / 値ペアをこのルールのアクション・コンテキストに追加します。この例では、新規の変換ファンクションが hr.executive_to_lead で、strmadmin ユーザーにはその EXECUTE 権限が付与されていると想定しています。

ルールのアクション・コンテキストに既存の名前 / 値ペアを保つために、この例では変更前にルールのアクション・コンテキストを選択して変数に挿入しています。

```
DECLARE
  action_ctx       SYS.RE$NV_LIST;
  ac_name          VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
BEGIN
  SELECT RULE_ACTION_CONTEXT
    INTO action_ctx
    FROM DBA_RULES R
    WHERE RULE_OWNER='STRMADMIN' AND RULE_NAME='DEPARTMENTS5';
  action_ctx.REMOVE_PAIR(ac_name);
  action_ctx.ADD_PAIR(ac_name,
                SYS.ANYDATA.CONVERTVARCHAR2('hr.executive_to_lead'));
  DBMS_RULE_ADM.ALTER_RULE(
    rule_name       =>  'strmadmin.departments5',
    action_context  => action_ctx);
END;
/
```

変換ファンクションが正しく変更されたかどうか確認するには、手順 1 で問合せを再実行できます。3 つのサブセット・ルールの一貫性を保つために、DEPARTMENTS6 および DEPARTMENTS7 ルールのアクション・コンテキストも同様に変更する必要があります。

ルールベースの変換の削除

ルールからルールベースの変換を削除するには、そのルールのアクションから STREAMS$_TRANSFORM_FUNCTION 名を持つ名前 / 値ペアを削除します。この例では、ルール DEPARTMENTS5 からルールベースの変換を削除します。このルールベースの変換は、15-11 ページの「ルールベースの変換の作成」の例で DEPARTMENTS5 ルールに追加したものです。

ルールベースの変換を削除すると、ルールのアクション・コンテキストが変更されます。アクション・コンテキストに、変換を指定する名前 / 値ペアの他にも名前 / 値ペアが含まれている場合、アクション・コンテキストを変更するときには、変換に無関係な名前 / 値ペアを変更または削除しないように注意してください。

Streams の場合、サブセット・ルールではアクション・コンテキスト内の名前 / 値ペアを使用して、特定の状況で UPDATE 操作を INSERT および DELETE 操作に変換する内部変換が実行されます。この種の変換は、行の移行と呼ばれます。サブセット・ルール用に新規の変換を指定するか、既存の変換を変更する場合は、行の移行を実行する名前 / 値ペアを保つように注意してください。

この例では、ルールのアクション・コンテキストを問い合せて変数に挿入してから、ルールベースの変換に関する名前 / 値ペアを削除します。

```
DECLARE
  action_ctx       SYS.RE$NV_LIST;
  ac_name          VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
BEGIN
  SELECT RULE_ACTION_CONTEXT
    INTO action_ctx
    FROM DBA_RULES R
    WHERE RULE_OWNER='STRMADMIN' AND RULE_NAME='DEPARTMENTS5';
  action_ctx.REMOVE_PAIR(ac_name);
  DBMS_RULE_ADM.ALTER_RULE(
    rule_name       =>  'strmadmin.departments5',
    action_context  =>  action_ctx);
END;
/
```

変換ファンクションが削除されたかどうか確認するには、15-18 ページの手順 1 で問合せを実行できます。3 つのサブセット・ルールの一貫性を保つために、DEPARTMENTS6 および DEPARTMENTS7 ルールのアクション・コンテキストも同様に変更する必要があります。

関連項目： 4-12 ページ「行の移行」

16

その他の Streams 管理タスク

この章では、論理変更レコード（LCR）および Streams タグを管理する手順、ならびに Streams 環境で全データベースのエクスポート / インポートを実行する手順について説明します。

この章の内容は次のとおりです。

- 論理変更レコード（LCR）の管理
- Streams タグの管理
- 接続先データベースにおけるデータベースの Point-in-Time リカバリの実行
- Streams を使用したデータベースにおける全データベースのエクスポート / インポートの実行

この章で説明する各タスクは、特に明記されていないかぎり、適切な権限を付与されている Streams 管理者が完了する必要があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

論理変更レコード（LCR）の管理

この項では、論理変更レコード（LCR）の管理について説明します。LCR を作成または変更するときには、次の要件を満たしていることを確認してください。

- 行 LCR を作成または変更する場合は、command_type 属性に古い列値の有無および新しい列値の有無との一貫性があることを確認してください。
- DDL LCR を作成または変更する場合は、ddl_text が base_table_name、base_table_owner、object_type、object_owner、object_name および command_type 属性と一貫していることを確認してください。

LCR の構成とエンキュー

次の LCR コンストラクタを使用して LCR を作成します。

- データ操作言語（DML）文によって生じた行に対する変更を含む行 LCR を作成するには、SYS.LCR$_ROW_RECORD コンストラクタを使用します。
- データ定義言語（DDL）変更を含む DDL LCR を作成するには、SYS.LCR$_DDL_RECORD コンストラクタを使用します。各 DDL LCR の ddl_text 属性に指定する DDL テキストが、Oracle SQL 構文に準拠していることを確認してください。

次の例では、Oracle データベース内のキューと、そのキューに関連付けられた適用プロセスを作成します。次に、渡された情報に基づいて行 LCR を構成し、それをキューにエンキューする PL/SQL プロシージャを作成します。

1. Oracle データベースに Streams キューを作成します。この例では、Streams 管理者が strmadmin ユーザーであることを想定しています。

```
CONNECT strmadmin/strmadminpw

BEGIN DBMS_STREAMS_ADM.SET_UP_QUEUE(
   queue_table          =>  'strm04_queue_table',
   storage_clause       =>  NULL,
   queue_name           =>  'strm04_queue');
END;
/
```

2. キュー内のメッセージを受信する Oracle データベースで、適用プロセスを作成します。適用プロセスでは、取得プロセスによって取得されたイベントではなくユーザー・エンキュー・イベントが適用されるため、この適用プロセスの作成時には、apply_captured パラメータを false に設定してください。また、hr.regions 表に変更が適用され、適用ユーザーにはこの表に DML 変更を加える権限が必要であるため、apply_user パラメータが hr に設定されていることを確認してください。

```
BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
     queue_name      => 'strm04_queue',
     apply_name      => 'strm04_apply',
     apply_captured  => false,
     apply_user      => 'hr');
END;
/
```

3. 適用プロセスのルール・セットを作成し、DML 変更を dbs1.net ソース・データベースに作成された hr.regions 表に適用するルールを追加します。

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name               =>  'hr.regions',
    streams_type             =>  'apply',
    streams_name             =>  'strm04_apply',
    queue_name               =>  'strm04_queue',
    include_dml              =>  true,
    include_ddl              =>  false,
    include_tagged_lcr       =>  false,
    source_database          =>  'dbs1.net');
END;
/
```

4. 適用プロセスの disable_on_error パラメータを n に設定します。

```
BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'strm04_apply',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/
```

5. 適用プロセスを起動します。

```
EXEC DBMS_APPLY_ADM.START_APPLY('strm04_apply');
```

6. 行 LCR を構成して手順 1 で作成したキューにエンキューするプロシージャ construct_row_lcr を作成します。

```
CREATE OR REPLACE PROCEDURE construct_row_lcr(
                  source_dbname  VARCHAR2,
                  cmd_type       VARCHAR2,
                  obj_owner      VARCHAR2,
                  obj_name       VARCHAR2,
                  old_vals       SYS.LCR$_ROW_LIST,
                  new_vals       SYS.LCR$_ROW_LIST) AS
  eopt           DBMS_AQ.ENQUEUE_OPTIONS_T;
  mprop          DBMS_AQ.MESSAGE_PROPERTIES_T;
  enq_msgid      RAW(16);
  row_lcr        SYS.LCR$_ROW_RECORD;
BEGIN
  mprop.SENDER_ID := SYS.AQ$_AGENT('strmadmin', NULL, NULL);
  -- Construct the LCR based on information passed to procedure
  row_lcr := SYS.LCR$_ROW_RECORD.CONSTRUCT(
    source_database_name  =>  source_dbname,
    command_type          =>  cmd_type,
    object_owner          =>  obj_owner,
    object_name           =>  obj_name,
    old_values            =>  old_vals,
    new_values            =>  new_vals);
  -- Enqueue the created row LCR
  DBMS_AQ.ENQUEUE(
    queue_name         =>  'strm04_queue',
    enqueue_options    =>  eopt,
    message_properties =>  mprop,
    payload            =>  SYS.AnyData.ConvertObject(row_lcr),
    msgid              =>  enq_msgid);
END construct_row_lcr;
/
```

注意： トランザクション識別子や SCN は適用プロセスで生成されてメモリーに格納されるため、これらの値を LCR の作成時にアプリケーションで指定する必要はありません。LCR でトランザクション識別子または SCN を指定しても、適用プロセスでは無視され、新規の値が割り当てられます。

関連項目： LCR のコンストラクタの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

7. 手順 2 で作成した construct_row_lcr プロシージャを使用して LCR を作成し、エンキューします。

a. hr.regions 表に 1 行を挿入する行 LCR を作成します。

```
CONNECT strmadmin/strmadminpw

DECLARE
  newunit1  SYS.LCR$_ROW_UNIT;
  newunit2  SYS.LCR$_ROW_UNIT;
  newvals   SYS.LCR$_ROW_LIST;
BEGIN
  newunit1 := SYS.LCR$_ROW_UNIT(
    'region_id',
    SYS.AnyData.ConvertNumber(5),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit2 := SYS.LCR$_ROW_UNIT(
    'region_name',
    SYS.AnyData.ConvertVarchar2('Moon'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1,newunit2);
construct_row_lcr(
  source_dbname  =>  'dbs1.net',
  cmd_type       =>  'INSERT',
  obj_owner      =>  'hr',
  obj_name       =>  'regions',
  old_vals       =>  NULL,
  new_vals       =>  newvals);
END;
/
COMMIT;
```

hr ユーザーとして接続して hr.regions 表を問い合せ、適用済の行の変更を表示できます。

```
CONNECT hr/hr

SELECT * FROM hr.regions;
```

region_id が 5 である行は、region_name が Moon となっているはずです。

b. hr.regions 表から 1 行を更新する行 LCR を作成します。

```
CONNECT strmadmin/strmadminpw

DECLARE
  oldunit1  SYS.LCR$_ROW_UNIT;
  oldunit2  SYS.LCR$_ROW_UNIT;
  oldvals   SYS.LCR$_ROW_LIST;
  newunit1  SYS.LCR$_ROW_UNIT;
  newvals   SYS.LCR$_ROW_LIST;
BEGIN
  oldunit1 := SYS.LCR$_ROW_UNIT(
    'region_id',
    SYS.AnyData.ConvertNumber(5),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldunit2 := SYS.LCR$_ROW_UNIT(
    'region_name',
    SYS.AnyData.ConvertVarchar2('Moon'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldvals := SYS.LCR$_ROW_LIST(oldunit1,oldunit2);
  newunit1 := SYS.LCR$_ROW_UNIT(
    'region_name',
    SYS.AnyData.ConvertVarchar2('Mars'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1);
construct_row_lcr(
  source_dbname  =>  'dbs1.net',
  cmd_type       =>  'UPDATE',
  obj_owner      =>  'hr',
  obj_name       =>  'regions',
  old_vals       =>  oldvals,
  new_vals       =>  newvals);
END;
/
COMMIT;
```

hr ユーザーとして接続して hr.regions 表を問い合せ、適用済の行の変更を表示できます。

```
CONNECT hr/hr

SELECT * FROM hr.regions;
```

region_id が 5 である行は、region_name が Mars となっているはずです。

c. hr.regions 表から 1 行を削除する行 LCR を作成します。

```
CONNECT strmadmin/strmadminpw

DECLARE
  oldunit1  SYS.LCR$_ROW_UNIT;
  oldunit2  SYS.LCR$_ROW_UNIT;
  oldvals   SYS.LCR$_ROW_LIST;
BEGIN
  oldunit1 := SYS.LCR$_ROW_UNIT(
    'region_id',
    SYS.AnyData.ConvertNumber(5),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldunit2 := SYS.LCR$_ROW_UNIT(
    'region_name',
    SYS.AnyData.ConvertVarchar2('Mars'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldvals := SYS.LCR$_ROW_LIST(oldunit1,oldunit2);
construct_row_lcr(
  source_dbname  =>  'dbs1.net',
  cmd_type       =>  'DELETE',
  obj_owner      =>  'hr',
  obj_name       =>  'regions',
  old_vals       =>  oldvals,
  new_vals       =>  NULL);
END;
/
COMMIT;
```

hr ユーザーとして接続して hr.regions 表を問い合せ、適用済の行の変更を表示できます。

```
CONNECT hr/hr

SELECT * FROM hr.regions;
```

region_id が 5 である行は、削除されているはずです。

一部の行 LCR メンバー関数の use_old パラメータ

リリース 9.2.0.2 では、SYS.LCR$_ROW_RECORD 型の次のメンバー関数に、新規パラメータ use_old が導入されます。

- GET_LOB_INFORMATION メンバー関数
- GET_VALUE メンバー関数
- GET_VALUES メンバー関数

現在、use_old パラメータについては『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』に記載されていません。『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』では、これらのメンバー関数の項が次の項に置き換えられています。

GET_LOB_INFORMATION メンバー関数

列の LOB 情報を取得します。

戻り値は次のいずれかになります。

```
DBMS_LCR.NOT_A_LOB        CONSTANT NUMBER := 1;
DBMS_LCR.NULL_LOB         CONSTANT NUMBER := 2;
DBMS_LCR.INLINE_LOB       CONSTANT NUMBER := 3;
DBMS_LCR.EMPTY_LOB        CONSTANT NUMBER := 4;
DBMS_LCR.LOB_CHUNK        CONSTANT NUMBER := 5;
DBMS_LCR.LAST_LOB_CHUNK   CONSTANT NUMBER := 6;
```

指定した列が存在しない場合、NULL が戻されます。

行 LCR のコマンド・タイプが UPDATE である場合、列の値を取得するには、use_old パラメータに 'Y' を指定すると便利です。

構文

```
MEMBER FUNCTION GET_LOB_INFORMATION(
  value_type    IN VARCHAR2,
  column_name   IN VARCHAR2,
  use_old       IN VARCHAR2   DEFAULT 'Y')
RETURN NUMBER;
```

パラメータ

表 16-1 GET_LOB_INFORMATION 関数パラメータ

パラメータ	説明
value_type	列について戻す値のタイプ（old または new）。
column_name	列名。
use_old	Y であり、value_type が new で、新規の値が存在しない場合、対応する古い値が戻されます。N であり、value_type が new の場合、新規の値が存在しなくても古い値は戻されません。 value_type が old であるか、または行 LCR の command_type が UPDATE でない場合、use_old パラメータの値は無視されます。 use_old パラメータでは、NULL は有効な指定ではありません。

GET_VALUE メンバー関数

指定した値タイプに応じて、指定した列の古い値または新規の値が戻されます。

行 LCR のコマンド・タイプが UPDATE である場合、列の値を取得するには、use_old パラメータに 'Y' を指定すると便利です。

構文

```
MEMBER FUNCTION GET_VALUE(
   value_type          IN VARCHAR2,
   column_name         IN VARCHAR2,
   use_old             IN VARCHAR2   DEFAULT 'Y')
RETURN SYS.AnyData;
```

パラメータ

表 16-2 GET_VALUE プロシージャ・パラメータ

パラメータ	説明
value_type	列について戻す値のタイプ。列の古い値を取得するには old を指定します。列の新規の値を取得するには new を指定します。
column_name	列名。この列が存在し、値が NULL である場合、NULL 値を含む SYS.AnyData インスタンスが戻されます。列値が存在しない場合、NULL が戻されます。

表 16-2 GET_VALUE プロシージャ・パラメータ（続き）

パラメータ	説明
use_old	Y であり、value_type が new で、新規の値が存在しない場合、対応する古い値が戻されます。 N であり、value_type が new の場合、新規の値が存在しないときは NULL が戻されます。 value_type が old であるか、または行 LCR の command_type が UPDATE でない場合、use_old パラメータの値は無視されます。 use_old パラメータでは、NULL は有効な指定ではありません。

GET_VALUES メンバー関数

指定した値タイプに応じて、古い値または新規の値のリストが戻されます。

行 LCR のコマンド・タイプが UPDATE である場合、すべての列の値を取得するには、use_old パラメータに 'Y' を指定すると便利です。

構文

```
MEMBER FUNCTION GET_VALUES(
   value_type   IN VARCHAR2,
   use_old      IN VARCHAR2   DEFAULT 'Y')
RETURN SYS.LCR$_ROW_LIST;
```

パラメータ

表 16-3 GET_VALUES プロシージャ・パラメータ

パラメータ	説明
value_type	戻す値のタイプ。古い値のリストを戻すには old を指定します。新規の値のリストを戻すには new を指定します。
use_old	Y であり、value_type が new の場合、LCR の新規の値がすべて含まれるリストが戻されます。新規の値がリストに存在しない場合、対応する古い値が戻されます。したがって、戻されるリストには、既存のすべての新規の値と、存在しない新規の値に対応する古い値が含まれます。 N であり、value_type が new の場合、古い値は戻されず、LCR の新規の値がすべて含まれるリストが戻されます。 value_type が old であるか、または行 LCR の command_type が UPDATE でない場合、use_old パラメータの値は無視されます。 use_old パラメータでは、NULL は有効な指定ではありません。

LOB 列を含む LCR の構成と処理

Streams 環境での LOB に関係する行変更の一般的な考慮事項は、次のとおりです。

- LOB に関係する行変更は、複数の LCR として取得、伝播および適用される場合があります。
- これらの LCR の評価に使用されるルールは、ルール・セット内のルールが行変更に対応するすべての LCR を TRUE と評価するか、またはどの LCR も TRUE と評価しないように、決定的である必要があります。
- 行変更に対応するすべての LCR が同じ方法で変換されるよう、これらの LCR の変換は決定的である必要があります。
- ユーザー変換で、LOB 挿入に対応する LCR 内の LOB チャンクのサイズが、LCR 内のオフセットを変更せずに縮小される場合、結果の LOB には、DBMS_LOB.WRITE プロシージャに記述されているように、0 バイトの充填文字または空白が含まれます。

ここでは、LOB を構成または処理する場合に満たす必要のある要件と、LOB を含む LCR に対する適用プロセスの動作について説明します。また、この項には、LOB を含む LCR を構成してエンキューする例も記載されています。

LOB を含む LCR の構成と処理の要件

環境で LOB 列を含む LCR を使用する場合は、その LCR を構成するか、適用ハンドラまたはルールベースの変換で処理するときに、次の要件を満たす必要があります。

- LCR の LOB 列のデータ部分が、VARCHAR2 型または RAW 型である必要があります。VARCHAR2 は CLOB、RAW は BLOB として解析されます。
- 表外の LOB に有効なコマンド・タイプは、LOB WRITE、LOB ERASE および LOB TRIM のみです。
- LOB WRITE、LOB ERASE および LOB TRIM の LCR の場合は、old_values コレクションを空または NULL、new_values を空以外にする必要があります。
- LOB WRITE LCR と LOB ERASE LCR の場合は、lob_offset を有効な値にする必要があります。他のすべてのコマンド・タイプの場合は、その列の LOB チャンクが続くものとみなして、lob_offset を NULL にする必要があります。
- LOB ERASE LCR と LOB TRIM LCR の場合は、lob_operation_size を有効な値にする必要があります。他のすべてのコマンド・タイプの場合は、lob_operation_size を NULL にする必要があります。
- LOB TRIM および LOB ERASE は、LCR に含まれている LOB 列の lob_information が LAST_LOB_CHUNK に設定されている場合にのみ有効なコマンド・タイプです。
- LOB WRITE は、LCR に含まれている LOB 列の lob_information が LAST_LOB_CHUNK または LOB_CHUNK に設定されている場合にのみ有効なコマンド・タイプです。

- lob_information が NULL_LOB に設定されている LOB の場合は、列のデータ部分を VARCHAR2 型の NULL（CLOB の場合）または RAW 型の NULL（BLOB の場合）にする必要があります。それ以外の場合は、非 NULL のインライン LOB 列として解析されます。
- 各 LOB WRITE LCR、LOB ERASE LCR および LOB TRIM LCR に許容されるのは、1 つの新規チャンクを持つ 1 つの LOB 列参照のみです。
- LOB ERASE LCR および LOB TRIM LCR 用の新規 LOB チャンクは、SYS.AnyData にカプセル化された NULL 値にする必要があります。

前述の要件の妥当性チェックは、すべて適用プロセスによって実行されます。前述の要件が満たされない場合、LOB 列を含む LCR は適用プロセスでは適用できず、適用ハンドラでは処理できません。この場合、LCR は同じトランザクション内の他の LCR とともに例外キューに移動します。

関連項目：

- 16-2 ページ「LCR の構成とエンキュー」
- 適用ハンドラの詳細は、4-3 ページの「適用プロセスによるイベント処理」を参照してください。
- LOB の詳細は、『Oracle9*i* アプリケーション開発者ガイド - ラージ・オブジェクト』を参照してください。

LOB を含む LCR に対する適用プロセスの動作

LOB を含む LCR を検出すると、適用プロセスは次のように動作します。

- コマンド・タイプが INSERT または UPDATE の LCR にデータを含む新規の LOB があり、lob_information が DBMS_LCR.LOB_CHUNK または DBMS_LCR.LAST_LOB_CHUNK でない場合は、そのデータが適用されます。
- コマンド・タイプが INSERT または UPDATE の LCR にデータを含まない新規の LOB があり、lob_information が DBMS_LCR.EMPTY_LOB の場合は、空の LOB として適用されます。
- コマンド・タイプが INSERT または UPDATE の LCR にデータを含まない新規の LOB があり、lob_information が DBMS_LCR.NULL_LOB または DBMS_LCR.INLINE_LOB の場合は、NULL として適用されます。
- コマンド・タイプが INSERT または UPDATE の LCR に新規の LOB があり、lob_information が DBMS_LCR.LOB_CHUNK または DBMS_LCR.LAST_LOB_CHUNK の場合は、すべての LOB 値が無視されます。コマンド・タイプが INSERT の場合は、LOB チャンクが続くものとみなされ、列に空の LOB が挿入されます。コマンド・タイプが UPDATE の場合は、LOB チャンクが続くものとみなされ、列の値が無視されます。

- コマンド・タイプが UPDATE の LCR 内の新規列がすべて LOB で、その lob_information が DBMS_LCR.LOB_CHUNK または DBMS_LCR.LAST_LOB_CHUNK の場合は、LOB チャンクが続くものとみなされ、更新がスキップされます。
- コマンド・タイプが UPDATE または DELETE の LCR の場合は、古い LOB 値が無視されます。
- LCR 内の LOB 列のデータを変更しないでください。これには、DML ハンドラ、エラー・ハンドラおよびルールベースの変換ファンクションが含まれています。

LOB を含む LCR を構成およびエンキューするスクリプトの例

次の例に、LOB を含む LCR を構成およびエンキューする PL/SQL プロシージャの作成手順を示します。この例は、第 11 章「Streams 環境の構成」で説明した必要な操作を完了し、Streams 用にデータベースの準備を完了したものと想定しています。このスクリプトを起動する Streams 管理者が、DBMS_AQ および DBMS_APPLY_ADM パッケージの EXECUTE 権限を持っていることを確認してください。

1. 出力の表示と結果のスプーリング
2. Streams 管理者として接続
3. Streams キューを作成
4. 適用プロセスの作成と起動
5. LOB 列を含む表を持つスキーマの作成
6. Streams 管理者への表に対して必要な権限を付与
7. LOB を含む LCR をエンキューする PL/SQL プロシージャの作成
8. CLOB をエンキューする do_enq_clob ファンクションの作成
9. do_enq_clob ファンクションを使用した CLOB のエンキュー
10. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 16-23 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************ BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL lob_construct.out

/*
```

手順 2 Streams 管理者として接続

```
*/

SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET SERVEROUTPUT ON SIZE 100000

CONNECT strmadmin/strmadminpw

/*
```

手順 3 Streams キューを作成

```
*/

BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table => 'lobex_queue_table',
    queue_name  => 'lobex_queue');
END;
/

/*
```

手順 4 適用プロセスの作成と起動

```
*/

BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name      => 'strmadmin.lobex_queue',
    apply_name      => 'apply_lob',
    apply_captured  => false);
END;
/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name => 'apply_lob',
    parameter  => 'disable_on_error',
    value      => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    'apply_lob');
END;
/

/*
```

手順 5 LOB 列を含む表を持つスキーマの作成

```
*/

CONNECT sys/change_on_install AS SYSDBA

CREATE USER lob_user IDENTIFIED BY Lob_user_pw;
GRANT CONNECT,RESOURCE TO lob_user;

CONNECT lob_user/lob_user_pw

CREATE TABLE with_clob (a  NUMBER PRIMARY KEY,
                        c1 CLOB,
                        c2 CLOB,
                        c3 CLOB);

CREATE TABLE with_blob (a NUMBER PRIMARY KEY,
                        b BLOB);

/*
```

手順 6 Streams 管理者への表に対して必要な権限を付与

これらの権限を付与すると、Streams 管理者はオフセットの LOB の長さを取得し、表に対して DML 操作を実行できます。

```
*/

GRANT ALL ON with_clob TO strmadmin;
GRANT ALL ON with_blob TO strmadmin;
COMMIT;

/*
```

手順 7 LOB を含む LCR をエンキューする PL/SQL プロシージャの作成

```
*/

CONNECT strmadmin/strmadminpw

CREATE OR REPLACE PROCEDURE enq_row_lcr(source_dbname  VARCHAR2,
                                        cmd_type       VARCHAR2,
                                        obj_owner      VARCHAR2,
                                        obj_name       VARCHAR2,
                                        old_vals       SYS.LCR$_ROW_LIST,
                                        new_vals       SYS.LCR$_ROW_LIST) AS
  eopt           DBMS_AQ.ENQUEUE_OPTIONS_T;
  mprop          DBMS_AQ.MESSAGE_PROPERTIES_T;
  enq_msgid      RAW(16);
  xr_lcr         SYS.LCR$_ROW_RECORD;
BEGIN
  mprop.SENDER_ID := SYS.AQ$_AGENT('strmadmin', NULL, NULL);
  xr_lcr := SYS.LCR$_ROW_RECORD.CONSTRUCT(
              source_database_name => source_dbname,
              command_type         => cmd_type,
              object_owner         => obj_owner,
              object_name          => obj_name,
              old_values           => old_vals,
              new_values           => new_vals);
  -- Enqueue a row lcr
  DBMS_AQ.ENQUEUE(
        queue_name         => 'lobex_queue',
        enqueue_options    => eopt,
        message_properties => mprop,
        payload            => SYS.AnyData.ConvertObject(xr_lcr),
        msgid              => enq_msgid);
END enq_row_lcr;
/
SHOW ERRORS

/*
```

手順 8 CLOB をエンキューする do_enq_clob ファンクションの作成

```
*/

-- Description of each variable:
-- src_dbname  : Source database name
-- tab_owner   : Table owner
-- tab_name    : Table name
-- col_name    : Name of the CLOB column
-- new_vals    : SYS.LCR$_ROW_LIST containing primary key and supplementally
--               logged colums
-- clob_data   : CLOB that contains data to be sent
-- offset      : Offset from which data should be sent, default is 1
-- lsize       : Size of data to be sent, default is 0
-- chunk_size  : Size used for creating LOB chunks, default is 2048

CREATE OR REPLACE FUNCTION do_enq_clob(src_dbname     VARCHAR2,
                                       tab_owner      VARCHAR2,
                                       tab_name       VARCHAR2,
                                       col_name       VARCHAR2,
                                       new_vals       SYS.LCR$_ROW_LIST,
                                       clob_data      CLOB,
                                       offset         NUMBER default 1,
                                       lsize          NUMBER default 0,
                                       chunk_size     NUMBER default 2048)
RETURN NUMBER IS
  lob_offset NUMBER; -- maintain lob offset
  newunit    SYS.LCR$_ROW_UNIT;
  tnewvals   SYS.LCR$_ROW_LIST;
  lob_flag   NUMBER;
  lob_data   VARCHAR2(32767);
  lob_size   NUMBER;
  unit_pos   NUMBER;
  final_size NUMBER;
  exit_flg   BOOLEAN;
  c_size     NUMBER;
  i          NUMBER;
BEGIN
  lob_size := DBMS_LOB.GETLENGTH(clob_data);
  unit_pos := new_vals.count + 1;
  tnewvals := new_vals;
  c_size   := chunk_size;
  i := 0;
  -- validate parameters
  IF (unit_pos <= 1) THEN
    DBMS_OUTPUT.PUT_LINE('Invalid new_vals list');
    RETURN 1;
```

```
END IF;

IF (c_size < 1) THEN
  DBMS_OUTPUT.PUT_LINE('Invalid LOB chunk size');
  RETURN 1;
END IF;

IF (lsize < 0 OR lsize > lob_size) THEN
  DBMS_OUTPUT.PUT_LINE('Invalid LOB size');
  RETURN 1;
END IF;

IF (offset < 1 OR offset >= lob_size) THEN
  DBMS_OUTPUT.PUT_LINE('Invalid lob offset');
  RETURN 1;
ELSE
  lob_offset := offset;
END IF;

-- calculate final size
IF (lsize = 0) THEN
  final_size := lob_size;
ELSE
  final_size := lob_offset + lsize;
END IF;

--  The following output lines are for debugging purposes only.
-- DBMS_OUTPUT.PUT_LINE('Final size: ' || final_size);
-- DBMS_OUTPUT.PUT_LINE('Lob size: ' || lob_size);

IF (final_size < 1 OR final_size > lob_size) THEN
  DBMS_OUTPUT.PUT_LINE('Invalid lob size');
  RETURN 1;
END IF;

-- expand new_vals list for LOB column
tnewvals.extend();

exit_flg := FALSE;

-- Enqueue all LOB chunks
LOOP
  --  The following output line is for debugging purposes only.
  DBMS_OUTPUT.PUT_LINE('About to write chunk#' || i);
  i := i + 1;
```

```
-- check if last LOB chunk
IF ((lob_offset + c_size) < final_size) THEN
  lob_flag := DBMS_LCR.LOB_CHUNK;
ELSE
  lob_flag := DBMS_LCR.LAST_LOB_CHUNK;
  exit_flg := TRUE;
  --  The following output line is for debugging purposes only.
  DBMS_OUTPUT.PUT_LINE('Last LOB chunk');
END IF;

--  The following output lines are for debugging purposes only.
DBMS_OUTPUT.PUT_LINE('lob offset: ' || lob_offset);
DBMS_OUTPUT.PUT_LINE('Chunk size: ' || to_char(c_size));

lob_data := DBMS_LOB.SUBSTR(clob_data, c_size, lob_offset);

-- create row unit for clob
newunit := SYS.LCR$_ROW_UNIT(col_name,
                             SYS.AnyData.ConvertVarChar2(lob_data),
                             lob_flag,
                             lob_offset,
                             NULL);

-- insert new LCR$_ROW_UNIT
tnewvals(unit_pos) := newunit;

-- enqueue lcr
enq_row_lcr(
      source_dbname => src_dbname,
      cmd_type      => 'LOB WRITE',
      obj_owner     => tab_owner,
      obj_name      => tab_name,
      old_vals      => NULL,
      new_vals      => tnewvals);

-- calculate next chunk size
lob_offset := lob_offset + c_size;

IF ((final_size - lob_offset) < c_size) THEN
  c_size := final_size - lob_offset + 1;
END IF;

--  The following output line is for debugging purposes only.
DBMS_OUTPUT.PUT_LINE('Next chunk size : ' || TO_CHAR(c_size));
```

```
    IF (c_size < 1) THEN
      exit_flg := TRUE;
    END IF;

    EXIT WHEN exit_flg;

  END LOOP;

  RETURN 0;
END do_enq_clob;
/

SHOW ERRORS

/*
```

手順 9 do_enq_clob ファンクションを使用した CLOB のエンキュー

次の例の DBMS_OUTPUT 行は、必要に応じてデバッグに使用できます。これらの行が必要ない場合、コメント行にするか、または削除できます。

```
*/

SET SERVEROUTPUT ON SIZE 100000
DECLARE
  c1_data CLOB;
  c2_data CLOB;
  c3_data CLOB;
  newunit1 SYS.LCR$_ROW_UNIT;
  newunit2 SYS.LCR$_ROW_UNIT;
  newunit3 SYS.LCR$_ROW_UNIT;
  newunit4 SYS.LCR$_ROW_UNIT;
  newvals  SYS.LCR$_ROW_LIST;
  big_data VARCHAR(22000);
  n        NUMBER;
BEGIN
  -- Create primary key for LCR$_ROW_UNIT
  newunit1 := SYS.LCR$_ROW_UNIT('A',
                                Sys.AnyData.ConvertNumber(3),
                                NULL,
                                NULL,
                                NULL);
  -- Create empty CLOBs
  newunit2 := sys.lcr$_row_unit('C1',
                                Sys.AnyData.ConvertVarChar2(NULL),
                                DBMS_LCR.EMPTY_LOB,
                                NULL,
                                NULL);
```

```
newunit3 := SYS.LCR$_ROW_UNIT('C2',
                              Sys.AnyData.ConvertVarChar2(NULL),
                              DBMS_LCR.EMPTY_LOB,
                              NULL,
                              NULL);
newunit4 := SYS.LCR$_ROW_UNIT('C3',
                              Sys.AnyData.ConvertVarChar2(NULL),
                              DBMS_LCR.EMPTY_LOB,
                              NULL,
                              NULL);
newvals := SYS.LCR$_ROW_LIST(newunit1,newunit2,newunit3,newunit4);

-- Perform an insert
enq_row_lcr(
  source_dbname => 'MYDB.NET',
  cmd_type      => 'INSERT',
  obj_owner     => 'LOB_USER',
  obj_name      => 'WITH_CLOB',
  old_vals      => NULL,
  new_vals      => newvals);

-- construct clobs
big_data := RPAD('Hello World', 1000, '_');
big_data := big_data || '#';
big_data := big_data || big_data || big_data || big_data || big_data;
DBMS_LOB.CREATETEMPORARY(
  lob_loc => c1_data,
  cache   => TRUE);
DBMS_LOB.WRITEAPPEND(
  lob_loc => c1_data,
  amount  => length(big_data),
  buffer  => big_data);

big_data := RPAD('1234567890#', 1000, '_');
big_data := big_data || big_data || big_data || big_data;
DBMS_LOB.CREATETEMPORARY(
  lob_loc => c2_data,
  cache   => TRUE);
DBMS_LOB.WRITEAPPEND(
  lob_loc => c2_data,
  amount  => length(big_data),
  buffer  => big_data);

big_data := RPAD('ASDFGHJKLQW', 2000, '_');
big_data := big_data || '#';
big_data := big_data || big_data || big_data || big_data || big_data;
DBMS_LOB.CREATETEMPORARY(
```

```
  lob_loc => c3_data,
  cache   => TRUE);
DBMS_LOB.WRITEAPPEND(
  lob_loc => c3_data,
  amount  => length(big_data),
  buffer  => big_data);

-- pk info
newunit1 := SYS.LCR$_ROW_UNIT('A',
                              SYS.AnyData.ConvertNumber(3),
                              NULL,
                              NULL,
                              NULL);
newvals  := SYS.LCR$_ROW_LIST(newunit1);

-- write c1 clob
n := do_enq_clob(
       src_dbname => 'MYDB.NET',
       tab_owner  => 'LOB_USER',
       tab_name   => 'WITH_CLOB',
       col_name   => 'C1',
       new_vals   => newvals,
       clob_data  => c1_data,
       offset     => 1,
       chunk_size => 1024);
DBMS_OUTPUT.PUT_LINE('n=' || n);

-- write c2 clob
newvals  := SYS.LCR$_ROW_LIST(newunit1);
n := do_enq_clob(
       src_dbname => 'MYDB.NET',
       tab_owner  => 'LOB_USER',
       tab_name   => 'WITH_CLOB',
       col_name   => 'C2',
       new_vals   => newvals,
       clob_data  => c2_data,
       offset     => 1,
       chunk_size => 2000);
DBMS_OUTPUT.PUT_LINE('n=' || n);

-- write c3 clob
newvals  := SYS.LCR$_ROW_LIST(newunit1);
n := do_enq_clob(src_dbname=>'MYDB.NET',
       tab_owner  => 'LOB_USER',
       tab_name   => 'WITH_CLOB',
       col_name   => 'C3',
       new_vals   => newvals,
```

```
        clob_data  => c3_data,
        offset     => 1,
        chunk_size => 500);
  DBMS_OUTPUT.PUT_LINE('n=' || n);

  COMMIT;

END;
/

/*
```

手順 10 スプール結果のチェック

このスクリプトの完了後に lob_construct.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT *****************************/
```

このスクリプトの実行後に lob_user.with_clob 表をチェックして、適用プロセスによって適用された行をリストできます。適用プロセスにエンキューされた行を適用する時間を与えるには、DBMS_LOCK.SLEEP 文を使用します。

```
CONNECT lob_user/lob_user_pw

EXECUTE DBMS_LOCK.SLEEP(10);

SELECT a, c1, c2, c3 FROM with_clob ORDER BY a;

SELECT a, LENGTH(c1), LENGTH(c2), LENGTH(c3) FROM with_clob ORDER BY a;
```

Streams タグの管理

現行セッションまたは適用プロセスによって生成されるタグの値を設定または取得できます。ここでは、タグ値の設定方法と取得方法について説明します。

- 現行セッションの Streams タグの管理
- 適用プロセス用の Streams タグの管理

関連項目：

- 第 8 章「Streams のタグ」
- 17-48 ページ「Streams タグの監視」

現行セッションの Streams タグの管理

この項では、現行セッション用のタグを設定する手順と取得する手順について説明します。

現行セッションで生成されるタグ値の設定

DBMS_STREAMS パッケージの SET_TAG プロシージャを使用すると、現行セッションで生成されるすべての REDO エントリ用のタグを設定できます。たとえば、現行セッションでタグを '1D' の 16 進値に設定するには、次のプロシージャを実行します。

```
BEGIN
   DBMS_STREAMS.SET_TAG(
      tag  =>  HEXTORAW('1D'));
END;
/
```

このプロシージャを実行すると、現行セッションで DML 文または DDL 文によって生成される各 REDO エントリのタグ値は 1D になります。このプロシージャを実行すると、現行セッションにのみ影響します。

現行セッションのタグ値の取得

DBMS_STREAMS パッケージの GET_TAG プロシージャを使用すると、現行セッションで生成されるすべての REDO エントリ用のタグを取得できます。たとえば、現行セッションの REDO エントリに生成されるタグの 16 進値を取得するには、次のプロシージャを実行します。

```
SET SERVEROUTPUT ON
DECLARE
   raw_tag RAW(2048);
BEGIN
   raw_tag := DBMS_STREAMS.GET_TAG();
   DBMS_OUTPUT.PUT_LINE('Tag Value = ' || RAWTOHEX(raw_tag));
END;
/
```

また、次のように DUAL ビューを問い合せると、現行セッションのタグ値を表示できます。

```
SELECT DBMS_STREAMS.GET_TAG FROM DUAL;
```

適用プロセス用の Streams タグの管理

この項では、適用プロセス用のタグを設定する手順と削除する手順について説明します。

関連項目：

- 適用プロセスと適用ハンドラによるタグの使用方法の概要は、8-6 ページの「タグと適用プロセス」を参照してください。
- 第 4 章「Streams 適用プロセス」
- 第 14 章「適用プロセスの管理」

適用プロセスで生成されるタグ値の設定

適用プロセスでは、変更をデータベースに適用するとき、またはハンドラを起動するときに、REDO エントリが生成されます。適用プロセスで生成される全 REDO エントリ用のデフォルト・タグは、DBMS_APPLY_ADM パッケージの CREATE_APPLY プロシージャを使用して適用プロセスを作成するとき、または DBMS_APPLY_ADM パッケージの ALTER_APPLY プロシージャを使用して既存の適用プロセスを変更するときに設定できます。どちらのプロシージャでも、apply_tag パラメータを、適用プロセスで生成されるタグ用に指定する値に設定します。

たとえば、既存の適用プロセス strm01_apply で REDO ログに生成されるタグの値を '7' の 16 進値に設定するには、次のプロシージャを実行します。

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
     apply_name  =>  'strm01_apply',
     apply_tag   =>  HEXTORAW('7'));
END;
/
```

このプロシージャを実行すると、適用プロセスで生成される各 REDO エントリのタグ値が 7 になります。

適用プロセス用の適用タグの削除

適用プロセス用の適用タグを削除するには、DBMS_APPLY_ADM パッケージの ALTER_APPLY プロシージャで、remove_apply_tag パラメータを true に設定します。適用タグを削除すると、適用プロセスで生成される各 REDO エントリのタグは NULL になります。たとえば、次のプロシージャでは、適用プロセス strm02_apply から適用タグが削除されます。

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name       => 'strm02_apply',
    remove_apply_tag => true);
END;
/
```

接続先データベースにおけるデータベースの Point-in-Time リカバリの実行

Point-in-Time リカバリとは、指定した現在以外の時刻、SCN またはログ順序番号までデータベースをリカバリすることです。Streams 環境の接続先データベースで Point-in-Time リカバリが必要な場合は、リカバリ時点以後に適用された取得済みの変更を再適用する必要があります。

関連する取得プロセスごとに、次のいずれかの方法を選択して、Streams 環境の接続先データベースで Point-in-Time リカバリを実行できます。

- 接続先データベースで適用される変更を取得する既存の取得プロセスについて、開始 SCN をリセットします。
- 接続先データベースで再適用する必要のある変更を取得するように、新規の取得プロセスを作成します。

新規の取得プロセスを作成するよりも、既存の取得プロセスの開始 SCN をリセットする方が簡単です。ただし、複数の接続先データベースで適用される変更が取得プロセスによって取得されると、Point-in-Time リカバリを実行していないデータベースも含め、すべての接続先データベースに変更が再送信されます。変更がすでに接続先データベースで適用されている場合、その変更は適用プロセスで廃棄されますが、複数の接続先データベースに変更を再送信するために必要なネットワーク・リソースとコンピュータ・リソースを、使用しないようにすることが必要な場合があります。この場合は、リカバリ済みの接続先データベースにのみ変更を伝播するように、新規の取得プロセスと新規の伝播を作成して一時的に使用できます。

ここでは、次の各タスクの手順について説明します。

- リカバリを実行するための既存の取得プロセスの開始 SCN のリセット
- リカバリを実行するための新規の取得プロセスの作成

Point-in-Time リカバリを実行した接続先データベースに複数の適用プロセスがある場合は、この項で説明するタスクの 1 つを適用プロセスごとに実行してください。

リカバリ中の接続先データベースに関して、次のいずれかの条件が TRUE である場合、どちらの方法も使用できません。

- 伝播では、ユーザー・エンキュー・メッセージが接続先データベースに伝播されます。どちらの方法でも、接続先データベースでは、ユーザー・エンキュー・イベントではなく取得イベントのみが再適用されます。
- 接続先データベースはソース・データベースでもあります。
- 有向ネットワーク構成では、接続先データベースは取得プロセスからのイベントをその他のデータベースに伝播させる目的で使用されますが、接続先データベースにはこの取得プロセスからのイベントは適用されません。

環境でこれらのいずれかの条件が TRUE である場合、この項で説明している方法は使用できません。かわりに、すべての接続先データベースでデータを手動で再同期化する必要があります。

関連項目：

- Point-in-Time リカバリの詳細は、『Oracle9*i* バックアップおよびリカバリ概要』を参照してください。
- 2-14 ページ「取得プロセスの開始 SCN、取得済 SCN および適用済 SCN」
- 12-12 ページ「変更が取得されるログ順序番号のリセット」
- 3-6 ページ「有向ネットワーク」

リカバリを実行するための既存の取得プロセスの開始 SCN のリセット

Point-in-Time リカバリを実行するために、既存の取得プロセスの開始 SCN をリセットするように決定した場合は、次の手順を実行します。

1. ソース・データベースと接続先データベースの間で有向ネットワークを使用していない場合は、ソース・データベースのソース・キューから接続先データベースの宛先キューに変更を伝播させる伝播を削除します。伝播を削除するには、`DBMS_PROPAGATION_ADM` パッケージの `DROP_PROPAGATION` プロシージャを使用します。

   有向ネットワークを使用しており、ソース・データベースと接続先データベースの間に中間データベースがある場合は、ソース・データベースの伝播を含め、接続先データベースへのパスに含まれる各中間データベースで伝播を削除します。

   注意： 適切な伝播を削除する必要があります。伝播を無効化するのみでは不十分です。手順 6 で伝播を再作成します。この時点で削除すると、取得プロセスの開始 SCN をリセットした後に作成されたイベントのみが確実に伝播されます。

   関連項目： 3-6 ページ「有向ネットワーク」

2. 接続先データベースで Point-in-Time リカバリを実行します。

3. 接続先データベースで、適用プロセスについてソース・データベースからの最も古いメッセージ番号を問い合せます。次に、問合せ結果をメモします。最も古いメッセージ番号とは、適用する必要があると思われる最も古いシステム変更番号（SCN）です。

   次の文に、実行する問合せの例を示します。

   ```
   SELECT APPLY_NAME, OLDEST_MESSAGE_NUMBER FROM DBA_APPLY_PROGRESS;
   ```

4. DBMS_CAPTURE_ADM パッケージの STOP_CAPTURE プロシージャを使用して、既存の取得プロセスを停止します。

5. 既存の取得プロセスの開始 SCN をリセットします。

   そのためには、DBMS_CAPTURE_ADM パッケージの ALTER_CAPTURE プロシージャを実行し、start_scn パラメータを手順 3 の問合せでメモした値に設定します。たとえば、取得プロセス strm01_capture の開始 SCN を値 829381993 にリセットするには、次の ALTER_CAPTURE プロシージャを実行します。

   ```
   BEGIN
     DBMS_CAPTURE_ADM.ALTER_CAPTURE(
       capture_name  =>  'strm01_capture',
       start_scn     =>  829381993);
   END;
   /
   ```

6. ソース・データベースと接続先データベースの間で有向ネットワークを使用していない場合は、DBMS_PROPAGATION_ADM パッケージの CREATE_PROPAGATION プロシージャを使用して、変更をソース・キューから宛先キューに伝播させる新規の伝播を作成します。伝播の作成時には、元の伝播で使用していたルール・セットを rule_set_name パラメータに指定します。

   有向ネットワークを使用しており、ソース・データベースと接続先データベースの間に中間データベースがある場合は、ソース・データベースの伝播を含め、接続先データベースへのパスに含まれる各中間データベースで、新規の伝播を作成します。

7. DBMS_CAPTURE_ADM パッケージの START_CAPTURE プロシージャを使用して、既存の取得プロセスを起動します。

リカバリを実行するための新規の取得プロセスの作成

Point-in-Time リカバリを実行するために新規の取得プロセスを作成するように決定した場合は、次の手順を実行します。

1. ソース・データベースと接続先データベースの間で有向ネットワークを使用していない場合は、ソース・データベースのソース・キューから接続先データベースの宛先キューに変更を伝播させる伝播を削除します。伝播を削除するには、DBMS_PROPAGATION_ADM パッケージの DROP_PROPAGATION プロシージャを使用します。

   有向ネットワークを使用しており、ソース・データベースと接続先データベースの間に中間データベースがある場合は、最後の中間データベースと接続先データベースの間でイベントを伝播する伝播を削除します。他の中間データベースやソース・データベースでは、伝播を削除する必要はありません。

   注意： 適切な伝播を削除する必要があります。伝播を無効化するのみでは不十分です。

   関連項目： 3-6 ページ「有向ネットワーク」

2. 接続先データベースで Point-in-Time リカバリを実行します。

3. 接続先データベースで、適用プロセスについてソース・データベースからの最も古いメッセージ番号を問い合せます。次に、問合せ結果をメモします。最も古いメッセージ番号とは、適用する必要があると思われる最も古いシステム変更番号（SCN）です。

   次の文に、実行する問合せの例を示します。

   ```
   SELECT APPLY_NAME, OLDEST_MESSAGE_NUMBER FROM DBA_APPLY_PROGRESS;
   ```

4. DBMS_STREAMS_ADM パッケージの SET_UP_QUEUE プロシージャを使用して、取得プロセスで使用するキューをソース・データベースに作成します。

   有向ネットワークを使用しており、ソース・データベースと接続先データベースの間に中間データベースがある場合は、ソース・データベースの新規キューを含め、接続先データベースへのパスに含まれる各中間データベースでキューを作成します。接続先データベースでは新規キューを作成しないでください。

5. ソース・データベースと接続先データベースの間で有向ネットワークを使用していない場合は、DBMS_PROPAGATION_ADM パッケージの CREATE_PROPAGATION プロシージャを使用して、変更を手順 4 で作成したソース・キューから宛先キューに伝播させる新規の伝播を作成します。伝播の作成時には、元の伝播で使用していたルール・セットを rule_set_name パラメータに指定します。

   有向ネットワークを使用しており、ソース・データベースと接続先データベースの間に中間データベースがある場合は、ソース・データベースから最初の中間データベースへ

の伝播を含め、接続先データベースへのパスに含まれる各中間データベースで伝播を作成します。これらの伝播では、手順 6 で作成する取得プロセスによって取得された変更が、手順 4 で作成したキュー間で伝播されます。

6. DBMS_CAPTURE_ADM パッケージの CREATE_CAPTURE プロシージャを使用して、ソース・データベースで新規の取得プロセスを作成します。source_queue パラメータを手順 4 で作成したローカル・キューに設定し、rule_set_name パラメータを元の取得プロセスで使用していたルール・セットに設定し、start_scn パラメータを手順 3 の問合せでメモした値に設定します。元の取得プロセスで使用していたルール・セットによって、リカバリした接続先データベースに送信する必要のないイベントが取得される場合は、一部のルールを元のルール・セットと共有するようにカスタマイズした小型のルール・セットを作成して使用できます。

7. DBMS_CAPTURE_ADM パッケージの START_CAPTURE プロシージャを使用して、手順 6 で作成した取得プロセスを起動します。

8. リカバリ後のデータベースの適用プロセスの最も古いメッセージ番号が、ソース・データベースの元の取得プロセスの取得番号に近づいた時点で、DBMS_CAPTURE_ADM パッケージの STOP_CAPTURE プロシージャを使用して元の取得プロセスを停止します。

   接続先データベースでは、次の問合せを使用すると、適用プロセスについてソース・データベースからの最も古いメッセージ番号を判断できます。

   ```
   SELECT APPLY_NAME, OLDEST_MESSAGE_NUMBER FROM DBA_APPLY_PROGRESS;
   ```

   ソース・データベースでは、次の問合せを使用すると、元の取得プロセスの取得番号を判断できます。

   ```
   SELECT CAPTURE_NAME, CAPTURE_MESSAGE_NUMBER FROM V$STREAMS_CAPTURE;
   ```

9. リカバリ後のデータベースの適用プロセスの最も古いメッセージ番号が、ソース・データベースの元の取得プロセスの取得番号を超えた時点で、手順 6 で作成した新規の取得プロセスを削除します。

10. ソース・データベースと接続先データベースの間で有向ネットワークを使用していない場合は、手順 5 で作成した新規の伝播を削除します。

    有向ネットワークを使用しており、ソース・データベースと接続先データベースの間に中間データベースがある場合は、ソース・データベースの新規の伝播を含め、接続先データベースへのパスに含まれる各中間データベースで新規の伝播を削除します。

11. ソース・データベースと接続先データベースの間で有向ネットワークを使用していない場合は、手順 4 で作成したキューを削除します。

    有向ネットワークを使用しており、ソース・データベースと接続先データベースの間に中間データベースがある場合は、ソース・データベースの新規キューを含め、接続先データベースへのパスに含まれる各中間データベースで新規キューを削除します。接続先データベースではキューを削除しないでください。

12. ソース・データベースと接続先データベースの間で有向ネットワークを使用していない場合は、ソース・データベースの元のソース・キューから接続先データベースの宛先キューに変更を伝播させる伝播を作成します。伝播を作成するには、DBMS_PROPAGATION_ADM パッケージの CREATE_PROPAGATION プロシージャを使用します。伝播の作成時には、元の伝播で使用していたルール・セットを rule_set_name パラメータに指定します。

    有向ネットワークを使用しており、ソース・データベースと接続先データベースの間に中間データベースがある場合は、最後の中間データベースから接続先データベースへの伝播を再作成します。これは、手順 1 で削除した伝播です。

13. 手順 8 で停止した取得プロセスを起動します。

手順 7 より後の手順は、すべて後から実行できます。また、手順 8 で説明した条件が満たされた時点で実行することもできます。

Streams を使用したデータベースにおける全データベースのエクスポート / インポートの実行

この項では、1 つ以上の Streams 取得プロセス、伝播または適用プロセスが動作するデータベースで、全データベースのエクスポート / インポートを実行する方法について説明します。ここでは、インポート・データベースおよびエクスポート・データベースが異なるコンピュータ上で動作しており、エクスポート・データベースがインポート・データベースで置き換えられる全データベースのエクスポート / インポートに関して説明しています。インポート・データベースのグローバル名とエクスポート・データベースのグローバル名は一致する必要があります。

注意： 既存の Streams 環境にデータベースを追加する場合、この項の指示には従わないでください。かわりに、11-14 ページの「取得ベースの Streams 環境の構成」を参照してください。

関連項目：

- 11-8 ページ「Streams に関連するエクスポート・ユーティリティとインポート・ユーティリティのパラメータ設定」
- 全データベースのエクスポート / インポートを実行する方法の詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。

次の手順に従って、Streams を使用したデータベースで全データベースのエクスポート / インポートを実行します。

1. エクスポート・データベースに、他のデータベースからの伝播に関する宛先キューが含まれる場合、エクスポート・データベースにイベントを伝播させる各伝播ジョブを無効化します。DBMS_AQADM パッケージの DISABLE_PROPAGATION_SCHEDULE プロシージャを使用すると、伝播ジョブを無効化できます。

2. 手順 1 で無効化した伝播ジョブで使用されていたデータベース・リンクが、インポート・データベースが動作するコンピュータを指し示すように、ネットワーク構成に必要な変更を加えます。

   この手順を実行するには、これらの伝播ジョブで使用されていたデータベース・リンクを再作成するか、またはソース・キューが含まれるデータベースで Oracle ネットワーキング・ファイルを変更する必要がある場合があります。

3. エクスポート・データベースに対するデータ操作言語（DML）およびデータ定義言語（DDL）変更を中止するようすべてのユーザーに通知し、これらの変更が中止されるまで待機します。

4. エクスポート・データベースの現行のシステム変更番号（SCN）をメモします。DBMS_FLASHBACK パッケージの GET_SYSTEM_CHANGE_NUMBER ファンクションを使用して、現行の SCN を決定できます。

   この手順を実行した後は、エクスポート・データベースで実行中の取得プロセスを停止しないでください。手順 10 では、V$STREAMS_CAPTURE 動的パフォーマンス・ビューを使用して、手順 3 の後にデータベースに対し DML または DDL 変更が加えられていないことを確認するよう指示しています。取得プロセスが停止および再起動された場合、このビューの取得プロセスに関する情報がリセットされます。

   手順 10 のチェックを有効にするには、取得プロセスでこの情報がリセットされないことが必要です。 取得プロセスの自動停止を防止するには、取得プロセスの message_limit および time_limit パラメータが infinite 以外の値に設定されている場合、これらのパラメータを infinite に設定する必要があります。

5. エクスポート・データベースで適用プロセスが実行されておらず、ユーザー・エンキュー・イベントが伝播されていない場合、この時点で全データベースのエクスポートを開始します。必要な Streams メタデータがエクスポートされるよう、FULL エクスポート・パラメータが y に設定されていることを確認します。

   エクスポート・データベースで 1 つ以上の適用プロセスが実行されているか、またはユーザー・エンキュー・イベントが伝播されている場合、エクスポートを開始せずに次の手順に進みます。

6. エクスポート・データベースで 1 つ以上の取得プロセスが実行中である場合、各取得プロセスの適用済 SCN が手順 4 で決定した SCN に達するか、これを超えるまで待機します。

   各取得プロセスの適用済 SCN を表示するには、DBA_CAPTURE データ・ディクショナリ・ビューの APPLIED_SCN 列を問い合せます。

7. エクスポート・データベースに、ユーザー・エンキュー・イベントを伝播させている伝播ジョブが存在する場合、DBMS_AQADM パッケージの DISABLE_PROPAGATION_SCHEDULE プロシージャを使用して、これらの伝播ジョブを無効化します。

8. エクスポート・データベースで 1 つ以上の適用プロセスが実行されているか、またはユーザー・エンキュー・イベントが伝播されている場合、この時点で全データベースのエクスポートを開始します。必要な Streams メタデータがエクスポートされるよう、FULL エクスポート・パラメータが y に設定されていることを確認します。手順 5 ですでにエクスポートを開始した場合、手順 9 に進みます。

9. エクスポートが完了した後、インポート・データベースが動作しているコンピュータにエクスポート・ダンプ・ファイルを送信します。

10. エクスポート・データベースで 1 つ以上の取得プロセスが実行されている場合、次の手順に従って、手順 4 で決定した SCN より前に、エクスポート・データベースに対するすべての DML および DDL 変更が停止されたことを確認します。

    a. DBMS_FLASHBACK パッケージの GET_SYSTEM_CHANGE_NUMBER 関数を使用して、現行の SCN を取得します。この SCN は新規 SCN と呼ばれます。

    b. 各取得プロセスの取得メッセージ番号が手順 a で決定した新規 SCN に達するか、これを超えるまで待機します。各取得プロセスの取得メッセージ番号を表示するには、V$STREAMS_CAPTURE 動的パフォーマンス・ビューの CAPTURE_MESSAGE_NUMBER 列を問い合せます。

    c. 各取得プロセスのエンキュー・メッセージ番号が、手順 4 で決定した SCN 以下であるかどうか検証します。各取得プロセスのエンキュー・メッセージ番号を表示するには、V$STREAMS_CAPTURE 動的パフォーマンス・ビューの ENQUEUE_MESSAGE_NUMBER 列を問い合せます。

       各取得プロセスのエンキュー・メッセージ番号が、手順 4 で決定した SCN 以下である場合、手順 11 に進みます。

       ただし、いずれかの取得プロセスのエンキュー・メッセージ番号が手順 4 で決定した SCN を超えている場合、手順 4 で決定した SCN より後に 1 つ以上の DML または DDL 変更が加えられており、これらの変更は取得プロセスにより取得およびエンキューされています。この場合、16-33 ページの手順 1 から、この項のすべての手順を再度実行します。

注意： この検証を有効にするため、手順 4 以降は各取得プロセスが途切れずに実行されている必要があります。

---

11. 全データベースのインポートを実行します。必要な Streams メタデータがインスポートされるよう、STREAMS_CONFIGURATION および FULL インポート・パラメータがどちらも y に設定されていることを確認します。STREAMS_CONFIGURATION インポート・パラメータのデフォルト設定は y です。また、インポート中は、インポート・データベースに対する DML または DDL 変更が行われないことを確認してください。
12. ユーザーに対しインポート・データベースへのアクセスを許可し、エクスポート・データベースをシャットダウンします。
13. 手順 1 および 7 で無効化した伝播ジョブを有効化します。

手順 4 で取得プロセスの message_limit または time_limit パラメータの値をリセットした場合は、これらのパラメータを元の設定にリセットしてください。

17

Streams 環境の監視

この章では、Streams に関連する静的データ・ディクショナリ・ビューと動的パフォーマンス・ビューについて説明します。これらのビューを使用して、Streams 環境を監視できます。また、この章には、Streams 環境の監視に使用できる問合せの例も記載されています。

この章の内容は次のとおりです。

注意： Oracle Enterprise Manager の Streams ツールも、Streams 環境を監視するための優れた手段です。詳細は、Streams ツールのオンライン・ヘルプを参照してください。

関連項目： この章で説明するデータ・ディクショナリ・ビューの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

Streams の静的データ・ディクショナリ・ビューの概要

次の表に、Streams の静的データ・ディクショナリ・ビューを示します。

表 17-1 Streams の静的データ・ディクショナリ・ビュー

ALL_ ビュー	DBA_ ビュー	USER_ ビュー
ALL_APPLY	DBA_APPLY	なし
ALL_APPLY_CONFLICT_COLUMNS	DBA_APPLY_CONFLICT_COLUMNS	なし
ALL_APPLY_DML_HANDLERS	DBA_APPLY_DML_HANDLERS	なし
ALL_APPLY_ERROR	DBA_APPLY_ERROR	なし
なし	DBA_APPLY_INSTANTIATED_OBJECTS	なし
ALL_APPLY_KEY_COLUMNS	DBA_APPLY_KEY_COLUMNS	なし
ALL_APPLY_PARAMETERS	DBA_APPLY_PARAMETERS	なし
ALL_APPLY_PROGRESS	DBA_APPLY_PROGRESS	なし
ALL_CAPTURE	DBA_CAPTURE	なし
ALL_CAPTURE_PARAMETERS	DBA_CAPTURE_PARAMETERS	なし
ALL_CAPTURE_PREPARED_DATABASE	DBA_CAPTURE_PREPARED_DATABASE	なし
ALL_CAPTURE_PREPARED_SCHEMAS	DBA_CAPTURE_PREPARED_SCHEMAS	なし
ALL_CAPTURE_PREPARED_TABLES	DBA_CAPTURE_PREPARED_TABLES	なし
ALL_EVALUATION_CONTEXT_TABLES	DBA_EVALUATION_CONTEXT_TABLES	USER_EVALUATION_CONTEXT_TABLES
ALL_EVALUATION_CONTEXT_VARS	DBA_EVALUATION_CONTEXT_VARS	USER_EVALUATION_CONTEXT_VARS
ALL_EVALUATION_CONTEXTS	DBA_EVALUATION_CONTEXTS	USER_EVALUATION_CONTEXTS
ALL_PROPAGATION	DBA_PROPAGATION	なし
ALL_RULE_SET_RULES	DBA_RULE_SET_RULES	USER_RULE_SET_RULES
ALL_RULE_SETS	DBA_RULE_SETS	USER_RULE_SETS
ALL_RULES	DBA_RULES	USER_RULES
ALL_STREAMS_GLOBAL_RULES	DBA_STREAMS_GLOBAL_RULES	なし
ALL_STREAMS_SCHEMA_RULES	DBA_STREAMS_SCHEMA_RULES	なし
ALL_STREAMS_TABLE_RULES	DBA_STREAMS_TABLE_RULES	なし

Streams の動的パフォーマンス・ビューの概要

次のリストに、Streams の動的パフォーマンス・ビューを示します。

- V$STREAMS_APPLY_COORDINATOR
- V$STREAMS_APPLY_READER
- V$STREAMS_APPLY_SERVER
- V$STREAMS_CAPTURE

Streams の取得プロセスの監視

ここでは、取得プロセス情報を表示するために実行できる問合せについて説明します。

- 各取得プロセスのキュー、ルール・セットおよび状態の表示
- 単一の取得プロセスに関する一般情報の表示
- 単一の取得プロセスのパラメータ設定のリスト表示
- データベースにおける全取得プロセスの適用済 SCN の判断
- 単一の取得プロセスに関する REDO ログのスキャン待機時間の判断
- 単一の取得プロセスに関するイベントのエンキュー待機時間の判断
- どのデータベース・オブジェクトがインスタンス化のために準備済みであるかの判断
- ソース・データベースのサプリメンタル・ログ・グループの表示

関連項目：

- 第 2 章「Streams の取得プロセス」
- 第 12 章「取得プロセスの管理」

各取得プロセスのキュー、ルール・セットおよび状態の表示

この項で説明する問合せを実行すると、データベース内の各取得プロセスに関する次の一般情報を表示できます。

- 取得プロセス名
- 取得プロセスで使用されるキューの名前
- 取得プロセスで使用されるルール・セットの名前
- 取得プロセスの状態（ENABLE、DISABLED または ABORTED）

この一般情報をデータベース内の取得プロセスごとに表示するには、次の問合せを実行します。

```
COLUMN CAPTURE_NAME HEADING 'Capture|Process|Name' FORMAT A15
COLUMN QUEUE_NAME HEADING 'Capture|Process|Queue' FORMAT A20
COLUMN RULE_SET_NAME HEADING 'Capture|Process|Rule Set' FORMAT A15
COLUMN STATUS HEADING 'Capture|Process|Status' FORMAT A15

SELECT CAPTURE_NAME, QUEUE_NAME, RULE_SET_NAME, STATUS FROM DBA_CAPTURE;
```

出力は次のようになります。

```
Capture         Capture              Capture         Capture
Process         Process              Process         Process
Name            Queue                Rule Set        Status
--------------- -------------------- --------------- ---------------
CAPTURE         STREAMS_QUEUE        RULESET$_6      ENABLED
```

単一の取得プロセスに関する一般情報の表示

この項で説明する問合せを実行すると、特定の取得プロセスに関する次の一般情報が表示されます。

- プロセス番号（cp*nn*）
- セッション識別子
- セッションのシリアル番号
- 取得プロセスの現在の状態（INITIALIZING、CAPTURING CHANGES、EVALUATING RULE、ENQUEUING MESSAGE、SHUTTING DOWN または CREATING LCR）
- スキャンされた REDO エントリの合計数
- エンキューされた LCR の合計数

たとえば、取得プロセス capture に関して前述の情報を表示するには、次の問合せを実行します。

```
COLUMN PROCESS_NAME HEADING 'Capture|Process|Number' FORMAT A7
COLUMN SID HEADING 'Session|ID' FORMAT 9999
COLUMN SERIAL# HEADING 'Session|Serial|Number' FORMAT 9999
COLUMN STATE HEADING 'State' FORMAT A17
COLUMN TOTAL_MESSAGES_CAPTURED HEADING 'Redo Entries|Scanned' FORMAT 9999999
COLUMN TOTAL_MESSAGES_ENQUEUED HEADING 'Total|LCRs|Enqueued' FORMAT 999999

SELECT SUBSTR(s.PROGRAM,INSTR(S.PROGRAM,'(')+1,4) PROCESS_NAME,
       c.SID,
       c.SERIAL#,
       c.STATE,
       c.TOTAL_MESSAGES_CAPTURED,
       c.TOTAL_MESSAGES_ENQUEUED
  FROM V$STREAMS_CAPTURE c, V$SESSION s
  WHERE c.CAPTURE_NAME = 'CAPTURE' AND
        c.SID = s.SID AND
        c.SERIAL# = s.SERIAL#;
```

出力は次のようになります。

```
Capture         Session                                      Total
Process Session  Serial                    Redo Entries     LCRs
Number       ID  Number State                    Scanned Enqueued
------- ------- ------- ----------------- ------------ --------
CP01         18     150 CAPTURING CHANGES        56900        7
```

スキャンされた REDO エントリの数が、取得プロセスのルール・セットについて TRUE と評価される DML と DDL の REDO エントリ数よりも多い場合があります。取得プロセスのキューには、取得プロセスのルール・セットについて TRUE と評価される DML と DDL の REDO エントリのみがエンキューされます。また、エンキューされた LCR の合計数には、トランザクション制御文を含む LCR が算入されます。これらの行 LCR には、COMMIT や ROLLBACK などのディレクティブが含まれています。したがって、エンキューされた LCR の合計数は、取得プロセスによってエンキューされた行変更と DDL 変更の数より多くなります。

> **関連項目：** トランザクション制御文の詳細は、2-3 ページの「行 LCR」を参照してください。

単一の取得プロセスのパラメータ設定のリスト表示

この項で説明する問合せを実行すると、特定の取得プロセスの各取得プロセス・パラメータについて現行の設定が表示されます。

たとえば、取得プロセス capture の取得プロセス・パラメータの設定を表示するには、次の問合せを実行します。

```
COLUMN PARAMETER HEADING 'Parameter' FORMAT A20
COLUMN VALUE HEADING 'Value' FORMAT A20
COLUMN SET_BY_USER HEADING 'Set by User?' FORMAT A20

SELECT PARAMETER,
       VALUE,
       SET_BY_USER
  FROM DBA_CAPTURE_PARAMETERS
  WHERE CAPTURE_NAME = 'CAPTURE';
```

出力は次のようになります。

```
Parameter            Value                Set by User?
-------------------- -------------------- --------------------
DISABLE_ON_LIMIT     N                    NO
MAXIMUM_SCN          INFINITE             NO
MESSAGE_LIMIT        INFINITE             NO
PARALLELISM          3                    YES
STARTUP_SECONDS      0                    NO
TIME_LIMIT           INFINITE             NO
TRACE_LEVEL          0                    NO
WRITE_ALERT_LOG      Y                    NO
```

注意： パラメータの Set by User? 列が NO の場合、そのパラメータはデフォルト値に設定されています。パラメータの Set by User? 列が YES の場合は、そのパラメータはデフォルト値に設定されている場合と、設定されていない場合があります。

関連項目：

- 2-23 ページ「取得プロセスのパラメータ」
- 12-7 ページ「取得プロセスのパラメータの設定」

データベースにおける全取得プロセスの適用済 SCN の判断

取得プロセスの適用済システム変更番号（SCN）は、関連適用プロセスによりデキューされた最新イベントの SCN です。この適用済 SCN より小さい番号のすべての変更は、取得プロセスで取得された変更を適用するすべての適用プロセスによりデキューされています。REDO ログ内のすべてのトランザクションがすべてのダウンストリーム・データベースで適用されるまで、すべての REDO ログを取得プロセスで使用できるようにしておく必要があるため、この SCN は重要です。

データベース内のすべての取得プロセスの適用済 SCN を表示するには、次の問合せを実行します。

```
COLUMN CAPTURE_NAME HEADING 'Capture Process Name' FORMAT A30
COLUMN APPLIED_SCN HEADING 'Applied SCN' FORMAT 999999

SELECT CAPTURE_NAME, APPLIED_SCN FROM DBA_CAPTURE;
```

出力は次のようになります。

```
Capture Process Name           Applied SCN
------------------------------ -----------
CAPTURE_EMP                         177154
```

単一の取得プロセスに関する REDO ログのスキャン待機時間の判断

この項で説明する問合せを実行すると、取得プロセスに関する次の情報を検索できます。

- REDO ログのスキャン待機時間。この時間は、取得プロセスでスキャンされた最新 REDO ログ・イベントの作成時刻から現在の時刻までの秒数を示します。取得プロセスを起動した直後は、この秒数が比較的大きい場合があります。
- 最後に記録された状態以降の秒数。これは、取得プロセスの状態が最後に記録されてからの秒数です。
- 現行の取得プロセス時刻。これは、取得プロセスの状態が最後に記録された時刻です。
- イベント作成時刻。これは、データ操作言語（DML）またはデータ定義言語（DDL）の変更によって、最新の取得イベントに関する REDO 情報が生成された時刻です。

この問合せで表示される情報は、有効化されている取得プロセスにのみ有効です。

次の問合せを実行すると、取得プロセス capture に関する REDO のスキャン待機時間を判断できます。

```
COLUMN LATENCY_SECONDS HEADING 'Latency|in|Seconds' FORMAT 999999
COLUMN LAST_STATUS HEADING 'Seconds Since|Last Status' FORMAT 999999
COLUMN CAPTURE_TIME HEADING 'Current|Process|Time'
COLUMN CREATE_TIME HEADING 'Event|Creation Time' FORMAT 999999

SELECT ((SYSDATE - CAPTURE_MESSAGE_CREATE_TIME)*86400) LATENCY_SECONDS,
       ((SYSDATE - CAPTURE_TIME)*86400) LAST_STATUS,
       TO_CHAR(CAPTURE_TIME, 'HH24:MI:SS MM/DD/YY') CAPTURE_TIME,
       TO_CHAR(CAPTURE_MESSAGE_CREATE_TIME, 'HH24:MI:SS MM/DD/YY') CREATE_TIME
  FROM V$STREAMS_CAPTURE
  WHERE CAPTURE_NAME = 'CAPTURE';
```

出力は次のようになります。

```
Latency               Current
     in Seconds Since Process           Event
Seconds   Last Status Time              Creation Time
------- ------------ ----------------- -----------------
      4            4 12:04:13 03/01/02 12:04:13 03/01/02
```

この問合せで戻される Latency in Seconds は、現在の時刻（SYSDATE）と Event Creation Time の差です。この問合せで戻される Seconds Since Last Status は、現在の時刻（SYSDATE）と Current Process Time の差です。

単一の取得プロセスに関するイベントのエンキュー待機時間の判断

この項で説明する問合せを実行すると、取得プロセスに関する次の情報を検索できます。

- イベントのエンキュー待機時間。この時間は、イベントが REDO ログに記録されてから取得プロセスでエンキューされるまでの秒数を示します。
- イベント作成時刻。これは、データ操作言語（DML）またはデータ定義言語（DDL）の変更によって、最新のエンキュー済みイベントに関する REDO 情報が生成された時刻です。
- エンキュー時刻。これは、取得プロセスによってイベントがキューにエンキューされた時刻です。
- エンキューされたイベントのメッセージ番号。

この問合せで表示される情報は、有効化されている取得プロセスにのみ有効です。

次の問合せを実行すると、取得プロセス capture に関するイベントの取得待機時間を判断できます。

```
COLUMN LATENCY_SECONDS HEADING 'Latency|in|Seconds' FORMAT 999999
COLUMN CREATE_TIME HEADING 'Event Creation|Time' FORMAT A20
COLUMN ENQUEUE_TIME HEADING 'Enqueue Time' FORMAT A20
COLUMN ENQUEUE_MESSAGE_NUMBER HEADING 'Message|Number' FORMAT 999999

SELECT (ENQUEUE_TIME-ENQUEUE_MESSAGE_CREATE_TIME)*86400 LATENCY_SECONDS,
       TO_CHAR(ENQUEUE_MESSAGE_CREATE_TIME, 'HH24:MI:SS MM/DD/YY') CREATE_TIME,
       TO_CHAR(ENQUEUE_TIME, 'HH24:MI:SS MM/DD/YY') ENQUEUE_TIME,
       ENQUEUE_MESSAGE_NUMBER
  FROM V$STREAMS_CAPTURE
  WHERE CAPTURE_NAME = 'CAPTURE';
```

出力は次のようになります。

```
Latency
     in Event Creation                          Message
Seconds Time                 Enqueue Time         Number
------- -------------------- -------------------- -------
      0 10:56:51 03/01/02    10:56:51 03/01/02     253962
```

この問合せで戻される Latency in Seconds は、Enqueue Time と Event Creation Time の差です。

どのデータベース・オブジェクトがインスタンス化のために準備済みであるかの判断

データベース・オブジェクトをインスタンス化のために準備するには、DBMS_CAPTURE_ADM パッケージの次のいずれかのプロシージャを使用します。

- PREPARE_TABLE_INSTANTIATION では、1 つの表がインスタンス化のために準備されます。
- PREPARE_SCHEMA_INSTANTIATION では、スキーマ内のすべてのデータベース・オブジェクトがインスタンス化のために準備されます。
- PREPARE_GLOBAL_INSTANTIATION では、データベース内のすべてのデータベース・オブジェクトがインスタンス化のために準備されます。

どのデータベース・オブジェクトのインスタンス化の準備が完了しているかを判断するには、対応する次のデータ・ディクショナリ・ビューを問い合せます。

- DBA_CAPTURE_PREPARED_TABLES
- DBA_CAPTURE_PREPARED_SCHEMAS
- DBA_CAPTURE_PREPARED_DATABASE

たとえば、インスタンス化の準備が完了しているすべての表、各表が準備された時刻およびその SCN を表示するには、次の問合せを実行します。

```
COLUMN TABLE_OWNER HEADING 'Table Owner' FORMAT A15
COLUMN TABLE_NAME HEADING 'Table Name' FORMAT A15
COLUMN SCN HEADING 'Instantiation SCN' FORMAT 999999
COLUMN TIMESTAMP HEADING 'Time Ready for|Instantiation'

SELECT TABLE_OWNER,
       TABLE_NAME,
       SCN,
       TO_CHAR(TIMESTAMP, 'HH24:MI:SS MM/DD/YY') TIMESTAMP
  FROM DBA_CAPTURE_PREPARED_TABLES;
```

出力は次のようになります。

```
                                                 Time Ready for
Table Owner     Table Name      Instantiation SCN Instantiation
--------------- --------------- ----------------- -----------------
HR              COUNTRIES                  196655 12:59:30 02/28/02
HR              DEPARTMENTS                196658 12:59:30 02/28/02
HR              EMPLOYEES                  196659 12:59:30 02/28/02
HR              JOBS                       196660 12:59:30 02/28/02
HR              JOB_HISTORY                196661 12:59:30 02/28/02
HR              LOCATIONS                  196662 12:59:30 02/28/02
HR              REGIONS                    196664 12:59:30 02/28/02
```

関連項目： 12-10 ページ「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」

ソース・データベースのサプリメンタル・ログ・グループの表示

サプリメンタル・ロギングでは、UPDATE 操作が実行されるたびに REDO ログに列データが追加されます。取得プロセスは、この追加情報を取得して LCR に含めます。取得された LCR を適用する適用プロセスには、変更を適切にスケジュールまたは適用するために、この追加情報が必要となる場合があります。

ソース・データベースで表に対して 1 つ以上のログ・グループが指定されているかどうかをチェックするには、次の問合せを実行します。

```
COLUMN LOG_GROUP_NAME HEADING 'Log Group' FORMAT A20
COLUMN TABLE_NAME HEADING 'Table' FORMAT A20
COLUMN ALWAYS HEADING 'Type of Log Group' FORMAT A30

SELECT
    LOG_GROUP_NAME,
    TABLE_NAME,
    DECODE(ALWAYS,
             'ALWAYS', 'Unconditional',
             NULL,     'Conditional') ALWAYS
  FROM DBA_LOG_GROUPS;
```

出力は次のようになります。

```
Log Group            Table                Type of Log Group
-------------------- -------------------- ------------------------------
LOG_GROUP_DEP_PK     DEPARTMENTS          Unconditional
LOG_GROUP_JOBS_CR    JOBS                 Conditional
```

特定のログ・グループの列を表示するには、DBA_LOG_GROUP_COLUMNS データ・ディクショナリ・ビューを問い合せます。また、V$DATABASE 動的パフォーマンス・ビューを問い合せると、データベース・レベルで指定されたサプリメンタル・ロギングを表示できます。

関連項目：

- 2-10 ページ「Streams 環境内のサプリメンタル・ロギング」
- 12-8 ページ「ソース・データベースでのサプリメンタル・ロギングの指定」

Streams キューの監視

ここでは、Streams キューに関する情報を表示するために実行できる問合せについて説明します。

- データベース内の Streams キューの表示
- キュー内の各ユーザー・エンキュー・イベントのコンシューマの判断
- キュー内のユーザー・エンキュー・イベントの内容の表示

関連項目：

- 第 3 章「Streams のステージングと伝播」
- 第 13 章「ステージングと伝播の管理」

データベース内の Streams キューの表示

Streams キューは、SYS.AnyData オブジェクト型です。データベース内のすべての Streams キューを表示するには、次の問合せを実行します。

```
COLUMN OWNER HEADING 'Owner' FORMAT A10
COLUMN NAME HEADING 'Queue Name' FORMAT A25
COLUMN QUEUE_TABLE HEADING 'Queue Table' FORMAT A20
COLUMN USER_COMMENT HEADING 'Comment' FORMAT A20

SELECT q.OWNER, q.NAME, t.QUEUE_TABLE, q.USER_COMMENT
  FROM DBA_QUEUES q, DBA_QUEUE_TABLES t
  WHERE t.OBJECT_TYPE = 'SYS.ANYDATA' AND
        q.QUEUE_TABLE = t.QUEUE_TABLE AND
        q.OWNER       = t.OWNER;
```

出力は次のようになります。

```
Owner      Queue Name                Queue Table          Comment
---------- ------------------------- -------------------- --------------------
STRMADMIN  AQ$_STREAMS_QUEUE_TABLE_E STREAMS_QUEUE_TABLE  exception queue
STRMADMIN  STREAMS_QUEUE             STREAMS_QUEUE_TABLE
```

Streams キューを作成すると、例外キューが自動的に作成されます。

関連項目： 13-2 ページ「Streams のキューの管理」

キュー内の各ユーザー・エンキュー・イベントのコンシューマの判断

キュー内の各ユーザー・エンキュー・イベントのコンシューマを判断するには、キュー所有者のスキーマ内で AQ$*queue_table_name* を問い合せます。この場合、*queue_table_name* はキュー表名です。たとえば、oe_queue_table キュー表内のユーザー・エンキュー・イベントのコンシューマを検索するには、次の問合せを実行します。

```
COLUMN MSG_ID HEADING 'Message ID' FORMAT 9999
COLUMN MSG_STATE HEADING 'Message State' FORMAT A13
COLUMN CONSUMER_NAME HEADING 'Consumer' FORMAT A30

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_QUEUE_TABLE;
```

出力は次のようになります。

```
Message ID                       Message State Consumer
-------------------------------- ------------- ------------------------------
99315B276CFA1872E034080020AE3E0A PROCESSED     APPLY_OE
99315B276CFB1872E034080020AE3E0A PROCESSED     APPLY_OE
99315B276CFA1872E034080020AE3E0A READY         EXPLICIT_DQ
99315B276CFB1872E034080020AE3E0A READY         EXPLICIT_DQ
```

注意： この問合せを実行すると、取得イベントではなくユーザー・エンキュー・イベントのみが表示されます。

関連項目： この例に表示されるイベントを Streams キューにエンキューする例については、第 19 章「Streams メッセージングの例」を参照してください。

キュー内のユーザー・エンキュー・イベントの内容の表示

Streams キュー内で、SYS.AnyData ペイロードにカプセル化されているペイロードの内容を表示するには、SYS.AnyData 型の Access*data_type* 静的ファンクションを使用してキュー表を問い合せます。この場合、*data_type* は表示するペイロードの型です。

関連項目： この項の問合せに表示されるイベントを Streams キューにエンキューする例については、13-18 ページの「SYS.AnyData ラッパーでのユーザー・メッセージ・ペイロードのラップ」を参照してください。

たとえば、キュー表 oe_queue_table を持つキューにある NUMBER 型のペイロードの内容を表示するには、キュー所有者として次の問合せを実行します。

```
SELECT qt.user_data.AccessNumber() "Numbers in Queue"
  FROM strmadmin.oe_q_table_any qt;
```

出力は次のようになります。

```
Numbers in Queue
----------------
              16
```

同様に、キュー表 oe_q_table_any を持つキューにある VARCHAR2 型のペイロードの内容を表示するには、次の問合せを実行します。

```
SELECT qt.user_data.AccessVarchar2() "Varchar2s in Queue"
   FROM strmadmin.oe_q_table_any qt;
```

出力は次のようになります。

```
Varchar2s in Queue
--------------------------------------------------------------------------------
Chemicals - SW
```

ユーザー定義のデータ型の内容を表示するには、独自に作成したカスタム・ファンクションを使用してキュー表を問い合せます。たとえば、oe.cust_address_typ のペイロードの内容を表示するには、Streams 管理者として接続し、次のようなファンクションを作成します。

```
CONNECT oe/oe

CREATE OR REPLACE FUNCTION oe.view_cust_address_typ(
in_any IN SYS.AnyData)
RETURN oe.cust_address_typ
IS
  address   oe.cust_address_typ;
  num_var   NUMBER;
BEGIN
  IF (in_any.GetTypeName() = 'OE.CUST_ADDRESS_TYP') THEN
    num_var := in_any.GetObject(address);
    RETURN address;
  ELSE RETURN NULL;
  END IF;
END;
/

GRANT EXECUTE ON oe.view_cust_address_typ TO STRMADMIN;

GRANT EXECUTE ON oe.cust_address_typ TO STRMADMIN;
```

次に、このファンクションを使用して、キュー表を次のように問い合せます。

```
CONNECT strmadmin/strmadminpw

SELECT oe.view_cust_address_typ(qt.user_data) "Customer Addresses"
  FROM strmadmin.oe_q_table_any qt
  WHERE qt.user_data.GetTypeName() = 'OE.CUST_ADDRESS_TYP';
```

出力は次のようになります。

```
Customer Addresses(STREET_ADDRESS, POSTAL_CODE, CITY, STATE_PROVINCE, COUNTRY_ID
--------------------------------------------------------------------------------
CUST_ADDRESS_TYP('1646 Brazil Blvd', '361168', 'Chennai', 'Tam', 'IN')
```

Streams 伝播および伝播ジョブの監視

ここでは、伝播および伝播ジョブの情報を表示するために実行できる問合せについて説明します。

- 伝播のソース・キューと宛先キューの判断
- 伝播のルール・セットの判断
- 伝播ジョブのスケジュールの表示
- 伝播されたイベントの合計数とバイト数の判断

関連項目：

- 第 3 章「Streams のステージングと伝播」
- 13-7 ページ「Streams の伝播および伝播ジョブの管理」

伝播のソース・キューと宛先キューの判断

ソース・キューを含むデータベースで DBA_PROPAGATION データ・ディクショナリ・ビューを問い合せると、伝播のソース・キューと宛先キューを判断できます。

たとえば、次の問合せを実行すると、伝播 dbs1_to_dbs2 に関して次の情報が表示されます。

- ソース・キューの所有者
- ソース・キュー名
- ソース・キューを含むデータベース
- 宛先キューの所有者

- 宛先キュー名
- 伝播で使用されるデータベース・リンク

```
COLUMN 'Source Queue' FORMAT A35
COLUMN 'Destination Queue' FORMAT A35

SELECT p.SOURCE_QUEUE_OWNER ||'.'||
       p.SOURCE_QUEUE_NAME ||'@'||
       g.GLOBAL_NAME "Source Queue",
       p.DESTINATION_QUEUE_OWNER ||'.'||
       p.DESTINATION_QUEUE_NAME ||'@'||
       p.DESTINATION_DBLINK "Destination Queue"
  FROM DBA_PROPAGATION p, GLOBAL_NAME g
  WHERE PROPAGATION_NAME = 'DBS1_TO_DBS2';
```

出力は次のようになります。

```
Source Queue                       Destination Queue
---------------------------------- ----------------------------------
STRMADMIN.STREAMS_QUEUE@DBS1.NET    STRMADMIN.STREAMS_QUEUE@DBS2.NET
```

伝播のルール・セットの判断

次の問合せを実行すると、伝播 dbs1_to_dbs2 で使用されるルール・セットの所有者と名前が表示されます。

```
COLUMN RULE_SET_OWNER HEADING 'Rule Set Owner' FORMAT A35
COLUMN RULE_SET_NAME HEADING 'Rule Set Name' FORMAT A35

SELECT RULE_SET_OWNER, RULE_SET_NAME
  FROM DBA_PROPAGATION
  WHERE PROPAGATION_NAME = 'DBS1_TO_DBS2';
```

出力は次のようになります。

```
Rule Set Owner                     Rule Set Name
---------------------------------- ----------------------------------
STRMADMIN                          RULESET$_3
```

伝播ジョブのスケジュールの表示

この項で説明する問合せを実行すると、伝播 dbs1_to_dbs2 で使用される伝播ジョブの伝播スケジュールに関して次の情報が表示されます。

- 伝播スケジュールの開始日時。
- 伝播ジョブの継続時間。これは、再起動前にジョブによってイベントが伝播される期間です。
- 伝播ジョブの待機時間。これは、継続時間中にキューにある他のすべてのメッセージが宛先に伝播された場合に、次のメッセージが伝播されるまでの最大待機時間です。
- 伝播ジョブが有効化されているかどうか。
- 最後にスケジュールを実行したプロセスの名前。
- スケジュールの実行に失敗した場合の連続失敗回数。連続して 16 回失敗すると、伝播ジョブは自動的に無効化されます。

ソース・キューを含むデータベースで、次の問合せを実行します。

```
COLUMN START_DATE HEADING 'Start Date'
COLUMN PROPAGATION_WINDOW HEADING 'Duration|in Seconds' FORMAT 99999
COLUMN NEXT_TIME HEADING 'Next|Time' FORMAT A8
COLUMN LATENCY HEADING 'Latency|in Seconds' FORMAT 99999
COLUMN SCHEDULE_DISABLED HEADING 'Status' FORMAT A8
COLUMN PROCESS_NAME HEADING 'Process' FORMAT A8
COLUMN FAILURES HEADING 'Number of|Failures' FORMAT 99

SELECT TO_CHAR(s.START_DATE, 'HH24:MI:SS MM/DD/YY') START_DATE,
       s.PROPAGATION_WINDOW,
       s.NEXT_TIME,
       s.LATENCY,
       DECODE(s.SCHEDULE_DISABLED,
                'Y', 'Disabled',
                'N', 'Enabled') SCHEDULE_DISABLED,
       s.PROCESS_NAME,
       s.FAILURES
  FROM DBA_QUEUE_SCHEDULES s, DBA_PROPAGATION p
  WHERE p.PROPAGATION_NAME = 'DBS1_TO_DBS2'
  AND p.DESTINATION_DBLINK = s.DESTINATION
  AND s.SCHEMA = p.SOURCE_QUEUE_OWNER
  AND s.QNAME = p.SOURCE_QUEUE_NAME;
```

出力は次のようになります。

```
                    Duration Next        Latency                    Number of
Start Date        in Seconds Time     in Seconds Status   Process   Failures
----------------- ---------- -------- ---------- -------- -------- ---------
15:23:40 03/02/02                              5 Enabled  J002             0
```

この伝播ジョブでは、Streams の伝播ジョブのデフォルト・スケジュールが使用されています。つまり、継続時間と次回実行時刻はどちらも NULL で、待機時間は 5 秒です。継続時間が NULL の場合、ジョブでは変更が伝播されますが、ジョブが自動的に再起動されることはありません。次回実行時刻が NULL の場合、伝播ジョブは現在実行中です。

関連項目：

- Streams の伝播ジョブのデフォルト伝播スケジュールの詳細は、3-20 ページの「伝播のスケジューリングと Streams 伝播」を参照してください。
- 伝播ジョブが無効化されている場合は、18-5 ページの「伝播で使用される伝播ジョブが有効化されているか」を参照してください。
- DBA_QUEUE_SCHEDULES データ・ディクショナリ・ビューの詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』および『Oracle9*i* データベース・リファレンス』を参照してください。

伝播されたイベントの合計数とバイト数の判断

同じデータベース・リンクを共有するソース・キューからのすべての伝播ジョブには、1 つの伝播スケジュールが使用されます。この項で説明する問合せを実行すると、特定の伝播ジョブに関連付けられている伝播スケジュールに関して次の情報が表示されます。

- 伝播スケジュールを実行するシステムによって消費された合計時間
- 伝播スケジュールに従って伝播されたイベントの合計数
- 伝播スケジュールに従って伝播された合計バイト数

たとえば、伝播 dbs1_to_dbs2 で使用される伝播ジョブに関してこの情報を表示するには、ソース・キューを含むデータベースで次の問合せを実行します。

```
COLUMN TOTAL_TIME HEADING 'Total Time Executing|in Seconds' FORMAT 999999
COLUMN TOTAL_NUMBER HEADING 'Total Events Propagated' FORMAT 999999999
COLUMN TOTAL_BYTES HEADING 'Total Bytes Propagated' FORMAT 9999999999999
```

```
SELECT s.TOTAL_TIME, s.TOTAL_NUMBER, s.TOTAL_BYTES
  FROM DBA_QUEUE_SCHEDULES s, DBA_PROPAGATION p
  WHERE p.PROPAGATION_NAME = 'DBS1_TO_DBS2'
  AND p.DESTINATION_DBLINK = s.DESTINATION
  AND s.SCHEMA = p.SOURCE_QUEUE_OWNER
  AND s.QNAME = p.SOURCE_QUEUE_NAME;
```

出力は次のようになります。

```
Total Time Executing
          in Seconds Total Events Propagated Total Bytes Propagated
-------------------- ---------------------- ----------------------
                  65                     71                  46536
```

関連項目： DBA_QUEUE_SCHEDULES データ・ディクショナリ・ビューの詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』および『Oracle9*i* データベース・リファレンス』を参照してください。

Streams の適用プロセスの監視

ここでは、適用プロセス情報を表示するために実行できる問合せについて説明します。

- 各適用プロセスに関する一般情報の表示
- 単一の適用プロセスのパラメータ設定のリスト表示
- 適用ハンドラに関する情報の表示
- 接続先データベースで指定されている代替キー列の表示
- 接続先データベース用の更新の競合ハンドラに関する情報の表示
- インスタンス化 SCN が設定されている表の判断
- 適用プロセス用のリーダー・サーバーに関する情報の表示
- イベントが取得されてからデキューされるまでの待機時間の判断
- コーディネータ・プロセスに関する情報の表示
- イベントが取得されてから適用されるまでの待機時間の判断
- 適用プロセス用の適用サーバーに関する情報の表示
- 適用プロセスに有効な適用並列性の表示
- 適用エラーのチェック
- 適用エラーの詳細情報の表示

関連項目：

- 第 4 章「Streams 適用プロセス」
- 第 14 章「適用プロセスの管理」

各適用プロセスに関する一般情報の表示

この項で説明する問合せを実行すると、データベース内の各適用プロセスに関する次の一般情報を表示できます。

- 適用プロセス名。
- 適用プロセスで使用されるキューの名前。
- 適用プロセスで使用されるルール・セットの名前。
- 適用プロセスによって適用されるイベントのタイプ。適用プロセスでは、取得プロセスで取得されたイベント、またはユーザーやアプリケーションによってエンキューされたイベントを適用できます。
- 適用プロセスの状態（ENABLED、DISABLED または ABORTED）。

この一般情報をデータベース内の適用プロセスごとに表示するには、次の問合せを実行します。

```
COLUMN APPLY_NAME HEADING 'Apply|Process|Name' FORMAT A15
COLUMN QUEUE_NAME HEADING 'Apply|Process|Queue' FORMAT A15
COLUMN RULE_SET_NAME HEADING 'Apply|Process|Rule Set' FORMAT A15
COLUMN APPLY_CAPTURED HEADING 'Type of|Events|Applied' FORMAT A15
COLUMN STATUS HEADING 'Apply|Process|Status' FORMAT A8

SELECT APPLY_NAME,
       QUEUE_NAME,
       RULE_SET_NAME,
       DECODE(APPLY_CAPTURED,
              'YES', 'Captured',
              'NO',  'User-Enqueued') APPLY_CAPTURED,
       STATUS
FROM DBA_APPLY;
```

出力は次のようになります。

```
Apply           Apply           Apply           Type of         Apply
Process         Process         Process         Events          Process
Name            Queue           Rule Set        Applied         Status
--------------- --------------- --------------- --------------- --------
APPLY_OE        OE_QUEUE        APPLY_OE_RS     User-Enqueued   ENABLED
APPLY           OE_QUEUE        RULESET$_4      Captured        DISABLED
```

単一の適用プロセスのパラメータ設定のリスト表示

この項で説明する問合せを実行すると、特定の適用プロセスの各適用プロセス・パラメータについて現行の設定が表示されます。

たとえば、適用プロセス strm01_apply の適用プロセス・パラメータの設定を表示するには、次の問合せを実行します。

```
COLUMN PARAMETER HEADING 'Parameter' FORMAT A20
COLUMN VALUE HEADING 'Value' FORMAT A20
COLUMN SET_BY_USER HEADING 'Set by User?' FORMAT A20

SELECT PARAMETER,
       VALUE,
       SET_BY_USER
  FROM DBA_APPLY_PARAMETERS
  WHERE APPLY_NAME = 'STRM01_APPLY';
```

出力は次のようになります。

```
Parameter            Value                Set by User?
-------------------- -------------------- --------------------
COMMIT_SERIALIZATION FULL                 NO
DISABLE_ON_ERROR     Y                    YES
DISABLE_ON_LIMIT     N                    NO
MAXIMUM_SCN          INFINITE             NO
PARALLELISM          1                    NO
STARTUP_SECONDS      0                    NO
TIME_LIMIT           INFINITE             NO
TRACE_LEVEL          0                    NO
TRANSACTION_LIMIT    INFINITE             NO
WRITE_ALERT_LOG      Y                    NO
```

注意： パラメータの Set by User? 列が NO の場合、そのパラメータはデフォルト値に設定されています。パラメータの Set by User? 列が YES の場合は、そのパラメータはデフォルト値に設定されている場合と、設定されていない場合があります。

関連項目：

- 4-31 ページ「適用プロセスのパラメータ」
- 14-11 ページ「適用プロセス・パラメータの設定」

適用ハンドラに関する情報の表示

この項では、適用プロセスの DML ハンドラ、DDL ハンドラおよびエラー・ハンドラに関する情報を表示する問合せについて説明します。

関連項目： 4-3 ページ「適用プロセスによるイベント処理」

ローカルの適用に関するすべての DML ハンドラとエラー・ハンドラの表示

接続先データベースで DBMS_APPLY_ADM パッケージの SET_DML_HANDLER プロシージャを使用して、ローカルの DML ハンドラまたはエラー・ハンドラを指定すると、変更をローカルに適用するデータベース内のすべての適用プロセスについて、そのハンドラが必要に応じて実行されます。DML ハンドラとエラー・ハンドラは、特定の表に対する指定した操作について実行されます。

データベース内で変更をローカルに適用する各適用プロセスについて、DML ハンドラまたはエラー・ハンドラを表示するには、次の問合せを実行します。

```
COLUMN OBJECT_OWNER HEADING 'Table|Owner' FORMAT A5
COLUMN OBJECT_NAME HEADING 'Table Name' FORMAT A10
COLUMN OPERATION_NAME HEADING 'Operation' FORMAT A9
COLUMN USER_PROCEDURE HEADING 'Handler Procedure' FORMAT A40
COLUMN ERROR_HANDLER HEADING 'Type of|Handler' FORMAT A10

SELECT OBJECT_OWNER,
       OBJECT_NAME,
       OPERATION_NAME,
       USER_PROCEDURE,
       DECODE(ERROR_HANDLER,
              'Y', 'Error',
              'N', 'DML') ERROR_HANDLER
  FROM DBA_APPLY_DML_HANDLERS
  WHERE APPLY_DATABASE_LINK IS NULL
  ORDER BY OBJECT_OWNER, OBJECT_NAME, ERROR_HANDLER;
```

出力は次のようになります。

```
Table                                                                Type of
Owner Table Name Operation Handler Procedure                         Handler
----- ---------- --------- ---------------------------------------- ----------
HR    LOCATIONS  UPDATE    STRMADMIN.HISTORY_DML                    DML
HR    REGIONS    INSERT    STRMADMIN.ERRORS_PKG.REGIONS_PK_ERROR    Error
```

注意： Oracle 以外のリモート・データベースに関して、変更を処理する DML ハンドラを指定することもできます。この問合せでは、この種の DML ハンドラは表示されません。これは、DML ハンドラが表示されるのは、APPLY_DATABASE_LINK 列が NULL の場合のみであるためです。

関連項目：

- 14-13 ページ「DML ハンドラの管理」
- 14-20 ページ「エラー・ハンドラの管理」

各適用プロセス用の DDL ハンドラとメッセージ・ハンドラの表示

データベース内の各適用プロセスについて、DDL ハンドラとメッセージ・ハンドラを表示するには、次の問合せを実行します。

```
COLUMN APPLY_NAME HEADING 'Apply Process Name' FORMAT A20
COLUMN DDL_HANDLER HEADING 'DDL Handler' FORMAT A20
COLUMN MESSAGE_HANDLER HEADING 'Message Handler' FORMAT A20

SELECT APPLY_NAME, DDL_HANDLER, MESSAGE_HANDLER FROM DBA_APPLY;
```

出力は次のようになります。

```
Apply Process Name   DDL Handler          Message Handler
-------------------- -------------------- --------------------
APPLY                oe.ddl_handler
APPLY_OE                                  oe.msg_handler
```

関連項目：

- 14-17 ページ「適用プロセスの DDL ハンドラの管理」
- 14-12 ページ「適用プロセスのメッセージ・ハンドラの管理」

接続先データベースで指定されている代替キー列の表示

接続先データベースで代替キーを指定できます。代替キーとは、Oracle が適用中に表の各行を識別するために使用できる列または列セットです。代替キー列を使用すると、主キーのない表のキー列を指定できます。また、接続先データベースで表が適用プロセスによって処理されるときには、その表の主キーのかわりに使用できます。

接続先データベースで指定されている代替キー列をすべて表示するには、次の問合せを実行します。

```
COLUMN OBJECT_OWNER HEADING 'Table Owner' FORMAT A20
COLUMN OBJECT_NAME HEADING 'Table Name' FORMAT A20
COLUMN COLUMN_NAME HEADING 'Substitute Key Name' FORMAT A20
COLUMN APPLY_DATABASE_LINK HEADING 'Database Link|for Remote|Apply' FORMAT A15

SELECT OBJECT_OWNER, OBJECT_NAME, COLUMN_NAME, APPLY_DATABASE_LINK
  FROM DBA_APPLY_KEY_COLUMNS
  ORDER BY APPLY_DATABASE_LINK, OBJECT_OWNER, OBJECT_NAME;
```

出力は次のようになります。

```
                                                                Database Link
                                                                for Remote
Table Owner          Table Name           Substitute Key Name   Apply
-------------------- -------------------- -------------------- ---------------
HR                   DEPARTMENTS          DEPARTMENT_NAME
HR                   DEPARTMENTS          LOCATION_ID
HR                   EMPLOYEES            FIRST_NAME
HR                   EMPLOYEES            LAST_NAME
HR                   EMPLOYEES            HIRE_DATE
```

注意： Oracle 以外のリモート・データベースの代替キー列の場合は、この問合せを実行すると、最終列にデータベース・リンクが表示されます。ローカルの接続先データベースの代替キー列が指定されている場合は、最終列が NULL になります。

関連項目：

- 4-11 ページ「代替キー列」
- 14-26 ページ「表の代替キー列の管理」

接続先データベース用の更新の競合ハンドラに関する情報の表示

DBMS_APPLY_ADM パッケージの SET_UPDATE_CONFLICT_HANDLER プロシージャを使用して更新の競合ハンドラを指定した場合は、関連する競合が発生すると、そのハンドラがデータベース内のすべての適用プロセスに対して実行されます。

この項で説明する問合せを実行すると、ビルトインの更新の競合ハンドラを使用して競合解消が指定されている列がすべて表示されます。つまり、データベース内で指定されているすべての列リストの列が表示されます。また、この問合せでは、指定されているビルトインの競合ハンドラの型と、列リストに指定されている解消列も表示されます。

データベース内のすべての更新の競合ハンドラに関する情報を表示するには、次の問合せを実行します。

```
COLUMN OBJECT_OWNER HEADING 'Table|Owner' FORMAT A5
COLUMN OBJECT_NAME HEADING 'Table Name' FORMAT A12
COLUMN METHOD_NAME HEADING 'Method' FORMAT A12
COLUMN RESOLUTION_COLUMN HEADING 'Resolution|Column' FORMAT A13
COLUMN COLUMN_NAME HEADING 'Column Name' FORMAT A30

SELECT OBJECT_OWNER,
       OBJECT_NAME,
       METHOD_NAME,
       RESOLUTION_COLUMN,
       COLUMN_NAME
  FROM DBA_APPLY_CONFLICT_COLUMNS
  ORDER BY OBJECT_OWNER, OBJECT_NAME, RESOLUTION_COLUMN;
```

出力は次のようになります。

```
Table                          Resolution
Owner Table Name   Method      Column       Column Name
----- ------------ ----------- ------------ ------------------------------
HR    COUNTRIES    MAXIMUM     TIME         COUNTRY_NAME
HR    COUNTRIES    MAXIMUM     TIME         REGION_ID
HR    COUNTRIES    MAXIMUM     TIME         TIME
HR    DEPARTMENTS  MAXIMUM     TIME         DEPARTMENT_NAME
HR    DEPARTMENTS  MAXIMUM     TIME         LOCATION_ID
HR    DEPARTMENTS  MAXIMUM     TIME         MANAGER_ID
HR    DEPARTMENTS  MAXIMUM     TIME         TIME
```

関連項目：

- 第 7 章「Streams 競合解消」
- 14-28 ページ「Streams の競合解消の管理」

インスタンス化 SCN が設定されている表の判断

インスタンス化 SCN は、接続先データベースで設定されます。この SCN によって、表について取得されても適用プロセスで無視される LCR と、データベース・オブジェクトについて取得されて適用プロセスで適用される LCR が制御されます。ソース・データベースからの表に関する LCR のコミット SCN が、接続先データベースでその表のインスタンス化 SCN 以下であれば、接続先データベースの適用プロセスでは LCR が廃棄されます。それ以外の場合は、適用プロセスによって LCR が適用されます。

次の問合せを実行すると、接続先データベースでインスタンス化 SCN が設定されている各表と、それぞれの表のインスタンス化 SCN が表示されます。

```
COLUMN SOURCE_DATABASE HEADING 'Source Database' FORMAT A15
COLUMN SOURCE_OBJECT_OWNER HEADING 'Object Owner' FORMAT A15
COLUMN SOURCE_OBJECT_NAME HEADING 'Object Name' FORMAT A15
COLUMN INSTANTIATION_SCN HEADING 'Instantiation SCN' FORMAT 999999

SELECT SOURCE_DATABASE,
       SOURCE_OBJECT_OWNER,
       SOURCE_OBJECT_NAME,
       INSTANTIATION_SCN
  FROM DBA_APPLY_INSTANTIATED_OBJECTS;
```

出力は次のようになります。

```
Source Database Object Owner    Object Name     Instantiation SCN
--------------- --------------- --------------- -----------------
DBS1.NET        HR              REGIONS                    196660
DBS1.NET        HR              COUNTRIES                  196660
DBS1.NET        HR              LOCATIONS                  196660
```

関連項目： 14-33 ページ「接続先データベースでのインスタンス化 SCN の設定」

適用プロセス用のリーダー・サーバーに関する情報の表示

適用プロセス用のリーダー・サーバーによって、キューからイベントがデキューされます。リーダー・サーバーはパラレル実行サーバーであり、LCR 間の依存性を計算してイベントをトランザクションにアセンブルします。次に、アセンブルしたトランザクションをコーディネータに戻すと、コーディネータはそれをアイドル状態の適用サーバーに割り当てます。

この項で説明する問合せを実行すると、特定の適用プロセス用のリーダー・サーバーに関して次の情報が表示されます。

- リーダー・サーバーによってデキューされるイベントのタイプ（取得された LCR またはユーザーがエンキューしたメッセージ）
- リーダー・サーバーで使用されるパラレル実行サーバーの名前
- リーダー・サーバーの現在の状態（IDLE、DEQUEUE MESSAGES または SCHEDULE MESSAGES）
- 適用プロセスの前回の起動時刻以降にリーダー・サーバーによってデキューされたイベントの合計数

この問合せで表示される情報は、有効化されている適用プロセスにのみ有効です。

たとえば、適用プロセス apply に関して前述の情報を表示するには、次の問合せを実行します。

```
COLUMN APPLY_CAPTURED HEADING 'Apply Type' FORMAT A22
COLUMN PROCESS_NAME HEADING 'Process Name' FORMAT A12
COLUMN STATE HEADING 'State' FORMAT A17
COLUMN TOTAL_MESSAGES_DEQUEUED HEADING 'Total Events Dequeued' FORMAT 99999999

SELECT DECODE(ap.APPLY_CAPTURED,
                'YES','Captured LCRS',
                'NO','User-enqueued messages','UNKNOWN') APPLY_CAPTURED,
       SUBSTR(s.PROGRAM,INSTR(S.PROGRAM,'(')+1,4) PROCESS_NAME,
       r.STATE,
       r.TOTAL_MESSAGES_DEQUEUED
       FROM V$STREAMS_APPLY_READER r, V$SESSION s, DBA_APPLY ap
       WHERE r.APPLY_NAME = 'APPLY' AND
             r.SID = s.SID AND
             r.SERIAL# = s.SERIAL# AND
             r.APPLY_NAME = ap.APPLY_NAME;
```

出力は次のようになります。

```
Apply Type             Process Name State             Total Events Dequeued
---------------------- ------------ ----------------- ---------------------
Captured LCRS          P000         DEQUEUE MESSAGES                   3803
```

イベントが取得されてからデキューされるまでの待機時間の判断

この項で説明する問合せを実行すると、特定の適用プロセスによってデキューされた最後のイベントに関して次の情報が表示されます。

- 待機時間。取得イベントにおける待機時間とは、ソース・データベースでイベントが作成されてから、適用プロセスでデキューされるまでの時間です。ユーザー・エンキュー・イベントにおける待機時間とは、ローカル・データベースでイベントがエンキューされてから、適用プロセスでデキューされるまでの時間です。
- イベント作成時刻。取得イベントの場合、イベント作成時刻は、データ操作言語（DML）またはデータ定義言語（DDL）の変更によって、ソース・データベースでイベントに関する REDO 情報が生成された時刻です。ユーザー・エンキュー・イベントの場合、イベント作成時刻はイベントが最後にエンキューされた時刻です。ユーザー・エンキュー・イベントは、適用プロセスに到達する前に、伝播によってさらに 1 回以上エンキューされる場合があります。
- イベントが適用プロセスによってデキューされた時刻。
- 適用プロセスによって最後にデキューされたイベントのメッセージ番号。

この問合せで表示される情報は、有効化されている適用プロセスにのみ有効です。

たとえば、最後に取得されて適用プロセス `apply` でデキューされたイベントの取得および伝播の待機時間を表示するには、次の問合せを実行します。

```
COLUMN LATENCY HEADING 'Latency|in|Seconds' FORMAT 9999
COLUMN CREATION HEADING 'Event Creation' FORMAT A17
COLUMN LAST_DEQUEUE HEADING 'Last Dequeue Time' FORMAT A20
COLUMN DEQUEUED_MESSAGE_NUMBER HEADING 'Dequeued|Message Number' FORMAT 999999

SELECT (DEQUEUE_TIME-DEQUEUED_MESSAGE_CREATE_TIME)*86400 LATENCY,
     TO_CHAR(DEQUEUED_MESSAGE_CREATE_TIME,'HH24:MI:SS MM/DD/YY') CREATION,
     TO_CHAR(DEQUEUE_TIME,'HH24:MI:SS MM/DD/YY') LAST_DEQUEUE,
     DEQUEUED_MESSAGE_NUMBER
  FROM V$STREAMS_APPLY_READER
  WHERE APPLY_NAME = 'APPLY';
```

出力は次のようになります。

```
Latency
     in                                              Dequeued
Seconds Event Creation    Last Dequeue Time    Message Number
------- ----------------- -------------------- --------------
     36 10:56:51 03/01/02 10:57:27 03/01/02            253962
```

コーディネータ・プロセスに関する情報の表示

コーディネータ・プロセスは、リーダー・サーバーからトランザクションを取得して適用サーバーに渡します。コーディネータ・プロセス名は ap*nn* で、*nn* はコーディネータ・プロセス番号です。

この項で説明する問合せを実行すると、特定の適用プロセス用のコーディネータ・プロセスに関して次の情報が表示されます。

- プロセス名のコーディネータ番号（ap*nn*）
- コーディネータ・セッションのセッション識別子
- コーディネータ・セッションのシリアル番号
- コーディネータの現在の状態（INITIALIZING、APPLYING、SHUTTING DOWN CLEANLY または ABORTING）
- 適用プロセスが最後に起動されてからコーディネータ・プロセスが受信したトランザクションの合計数
- 適用プロセスが最後に起動されてから、その適用プロセスで正常に適用されたトランザクションの合計数
- 適用プロセスが最後に起動されてから、適用プロセスで適用されて適用エラーとなったトランザクションの数

この問合せで表示される情報は、有効化されている適用プロセスにのみ有効です。

たとえば、適用プロセス apply に関して前述の情報を表示するには、次の問合せを実行します。

```
COLUMN PROCESS_NAME HEADING 'Coordinator|Process|Name' FORMAT A11
COLUMN SID HEADING 'Session|ID' FORMAT 9999
COLUMN SERIAL# HEADING 'Session|Serial|Number' FORMAT 9999
COLUMN STATE HEADING 'State' FORMAT A21
COLUMN TOTAL_RECEIVED HEADING 'Total|Trans|Received' FORMAT 99999999
COLUMN TOTAL_APPLIED HEADING 'Total|Trans|Applied' FORMAT 99999999
COLUMN TOTAL_ERRORS HEADING 'Total|Apply|Errors' FORMAT 9999

SELECT SUBSTR(s.PROGRAM,INSTR(S.PROGRAM,'(')+1,4) PROCESS_NAME,
       c.SID,
       c.SERIAL#,
       c.STATE,
       c.TOTAL_RECEIVED,
       c.TOTAL_APPLIED,
       c.TOTAL_ERRORS
       FROM V$STREAMS_APPLY_COORDINATOR c, V$SESSION s
       WHERE c.APPLY_NAME = 'APPLY' AND
             c.SID = s.SID AND
             c.SERIAL# = s.SERIAL#;
```

出力は次のようになります。

```
Coordinator         Session                              Total     Total  Total
Process     Session  Serial                              Trans     Trans  Apply
Name             ID  Number State                    Received   Applied Errors
----------- ------- ------- --------------------- --------- --------- ------
AP01             11      40 APPLYING                      78        73      2
```

イベントが取得されてから適用されるまでの待機時間の判断

この項では、特定のイベントについて取得から適用までの待機時間を表示する、2 つの異なる問合せについて説明します。つまり、取得イベントの場合、この 2 つの問合せでは、ソース・データベースでイベントが作成されてから、適用プロセスで適用されるまでの時間が表示されます。一方の問合せでは V$STREAMS_APPLY_COORDINATOR 動的パフォーマンス・ビューが使用され、他方の問合せでは DBA_APPLY_PROGRESS 静的データ・ディクショナリ・ビューが使用されます。

> **注意：** これらの問合せでは、適用プロセスでユーザー・エンキュー・イベントではなく取得イベントが適用されることを想定しています。

この 2 つの問合せには、次の重要な違いがあります。

- V$STREAMS_APPLY_COORDINATOR ビューの問合せを実行する場合は、適用プロセスが有効化されている必要がありますが、DBA_APPLY_PROGRESS ビューの問合せを実行する場合は、適用プロセスが無効化されていてもかまいません。
- V$STREAMS_APPLY_COORDINATOR ビューの問合せでは、DBA_APPLY_PROGRESS ビューの問合せよりも新しいトランザクションの待機時間を表示できます。

どちらの問合せを実行した場合も、特定の適用プロセスで適用されたイベントに関して次の情報が表示されます。

- イベントの取得から適用までの待機時間。
- イベント作成時刻。取得イベントの場合、イベント作成時刻は、データ操作言語（DML）またはデータ定義言語（DDL）の変更によって、ソース・データベースでイベントに関する REDO 情報が生成された時刻です。
- イベントが適用プロセスによって適用された時刻。
- イベントのメッセージ番号。

待機時間に関する V$STREAMS_APPLY_COORDINATOR の問合せ例

適用プロセス apply で適用されたイベントについて、V$STREAMS_APPLY_COORDINATOR ビューを使用して取得から適用までの待機時間を表示するには、次の問合せを実行します。

```
COLUMN 'Latency in Seconds' FORMAT 999999
COLUMN 'Event Creation' FORMAT A17
COLUMN 'Apply Time' FORMAT A17
COLUMN 'Applied Message Number' FORMAT 999999

SELECT (HWM_TIME-HWM_MESSAGE_CREATE_TIME)*86400 "Latency in Seconds",
     TO_CHAR(HWM_MESSAGE_CREATE_TIME,'HH24:MI:SS MM/DD/YY')
        "Event Creation",
     TO_CHAR(HWM_TIME,'HH24:MI:SS MM/DD/YY') "Apply Time",
     HWM_MESSAGE_NUMBER "Applied Message Number"
  FROM V$STREAMS_APPLY_COORDINATOR
  WHERE APPLY_NAME = 'APPLY';
```

出力は次のようになります。

```
Latency in Seconds Event Creation    Apply Time        Applied Message Number
------------------ ----------------- ----------------- ----------------------
                36 10:56:51 03/01/02 10:57:27 03/01/02                 253962
```

待機時間に関する DBA_APPLY_PROGRESS の問合せ例

適用プロセス apply で適用されたイベントについて、DBA_APPLY_PROGRESS ビューを使用して取得から適用までの待機時間を表示するには、次の問合せを実行します。

```
COLUMN 'Latency in Seconds' FORMAT 999999
COLUMN 'Event Creation' FORMAT A17
COLUMN 'Apply Time' FORMAT A17
COLUMN 'Applied Message Number' FORMAT 999999

SELECT (APPLY_TIME-APPLIED_MESSAGE_CREATE_TIME)*86400 "Latency in Seconds",
     TO_CHAR(APPLIED_MESSAGE_CREATE_TIME,'HH24:MI:SS MM/DD/YY')
        "Event Creation",
     TO_CHAR(APPLY_TIME,'HH24:MI:SS MM/DD/YY') "Apply Time",
     APPLIED_MESSAGE_NUMBER "Applied Message Number"
  FROM DBA_APPLY_PROGRESS
  WHERE APPLY_NAME = 'APPLY';
```

出力は次のようになります。

```
Latency in Seconds Event Creation    Apply Time        Applied Message Number
------------------ ----------------- ----------------- ----------------------
                38 10:50:09 03/01/02 10:50:47 03/01/02                 253678
```

適用プロセス用の適用サーバーに関する情報の表示

適用プロセスでは、1 つ以上の適用サーバーを使用できます。適用サーバーによって、LCR が DML 文または DDL 文としてデータベース・オブジェクトに適用されるか、適切なハンドラに渡されます。LCR 以外のメッセージの場合、適用サーバーはイベントをメッセージ・ハンドラに渡します。各適用サーバーは、パラレル実行サーバーです。

この項で説明する問合せを実行すると、特定の適用プロセス用の適用サーバーに関して次の情報が表示されます。

- 各適用サーバーによって適用されるイベントのタイプ（取得された LCR またはユーザーがエンキューしたメッセージ）。
- パラレル実行サーバーのプロセス名（順番に表示）。
- 各適用サーバーの現在の状態（IDLE、RECORD LOW-WATERMARK、ADD PARTITION、DROP PARTITION、EXECUTE TRANSACTION、WAIT COMMIT、WAIT DEPENDENCY または WAIT FOR NEXT CHUNK）。これらの各状態の詳細は、『Oracle9*i* データベース・リファレンス』の「V$STREAMS_APPLY_SERVER」を参照してください。
- 適用プロセスが最後に起動されてから、各適用サーバーに割り当てられたトランザクションの合計数。トランザクションには、複数のイベントが含まれている場合があります。
- 適用プロセスが最後に起動されてから、各適用サーバーによって適用されたイベントの合計数。

この問合せで表示される情報は、有効化されている適用プロセスにのみ有効です。

たとえば、適用プロセス apply に関して前述の情報を表示するには、次の問合せを実行します。

```
COLUMN APPLY_CAPTURED HEADING 'Apply Type' FORMAT A22
COLUMN PROCESS_NAME HEADING 'Process Name' FORMAT A12
COLUMN STATE HEADING 'State' FORMAT A17
COLUMN TOTAL_ASSIGNED HEADING 'Total|Transactions|Assigned' FORMAT 99999999
COLUMN TOTAL_MESSAGES_APPLIED HEADING 'Total|Events|Applied' FORMAT 99999999

SELECT DECODE(ap.APPLY_CAPTURED,
                'YES','Captured LCRS',
                'NO','User-enqueued messages','UNKNOWN') APPLY_CAPTURED,
       SUBSTR(s.PROGRAM,INSTR(S.PROGRAM,'(')+1,4) PROCESS_NAME,
       r.STATE,
       r.TOTAL_ASSIGNED,
       r.TOTAL_MESSAGES_APPLIED
  FROM V$STREAMS_APPLY_SERVER R, V$SESSION S, DBA_APPLY AP
```

```
  WHERE r.APPLY_NAME = 'APPLY' AND
        r.SID = s.SID AND
        r.SERIAL# = s.SERIAL# AND
        r.APPLY_NAME = ap.APPLY_NAME
  ORDER BY r.SERVER_ID;
```

出力は次のようになります。

```
                                                          Total     Total
                                                   Transactions    Events
Apply Type             Process Name State               Assigned   Applied
---------------------- ------------ ---------------- ------------ ---------
Captured LCRs          P001         IDLE                       94      2141
Captured LCRs          P002         IDLE                       12       276
Captured LCRs          P003         IDLE                        0         0
```

適用プロセスに有効な適用並列性の表示

環境によっては、適用プロセスで使用可能な適用サーバーすべてを使用できない場合があります。たとえば、適用プロセスの並列性は 5 に設定されていても、その適用プロセスでは適用サーバーが 3 つしか使用されたことがない場合があります。このような場合、有効な適用並列性は 3 です。

次の問合せを実行すると、適用プロセス apply に有効な適用並列性が表示されます。

```
SELECT COUNT(SERVER_ID) "Effective Parallelism"
  FROM V$STREAMS_APPLY_SERVER
  WHERE APPLY_NAME = 'APPLY' AND
        TOTAL_MESSAGES_APPLIED > 0;
```

出力は次のようになります。

```
Effective Parallelism
---------------------
                    2
```

この問合せでは、有効な並列性として 2 が戻されています。適用プロセス apply の並列性が 3 に設定されている場合は、この適用プロセスが最後に起動されてから使用されていない適用サーバーが 1 つあることになります。

次の問合せを実行すると、各適用サーバーで適用されたイベントの合計数を表示できます。

```
COLUMN SERVER_ID HEADING 'Apply Server ID' FORMAT 99
COLUMN TOTAL_MESSAGES_APPLIED HEADING 'Total Events Applied' FORMAT 999999

SELECT SERVER_ID, TOTAL_MESSAGES_APPLIED
  FROM V$STREAMS_APPLY_SERVER
  WHERE APPLY_NAME = 'APPLY'
  ORDER BY SERVER_ID;
```

出力は次のようになります。

```
Apply Server ID Total Events Applied
--------------- --------------------
              1                 2141
              2                  276
              3                    0
```

この場合、適用プロセスでは、再起動後は適用サーバー 3 が使用されていません。適用プロセスの parallelism がその有効な並列性より大きい値に設定されている場合は、parallelism を小さい値に設定することを考慮してください。

適用エラーのチェック

適用エラーの有無をチェックするには、次の問合せを実行します。

```
COLUMN APPLY_NAME HEADING 'Apply|Process|Name' FORMAT A8
COLUMN SOURCE_DATABASE HEADING 'Source|Database' FORMAT A8
COLUMN LOCAL_TRANSACTION_ID HEADING 'Local|Transaction|ID' FORMAT A11
COLUMN ERROR_MESSAGE HEADING 'Error Message' FORMAT A50

SELECT APPLY_NAME, SOURCE_DATABASE, LOCAL_TRANSACTION_ID, ERROR_MESSAGE
  FROM DBA_APPLY_ERROR;
```

適用エラーがあると、出力は次のようになります。

```
Apply             Local
Process  Source   Transaction
Name     Database ID          Error Message
-------- -------- ----------- --------------------------------------------------
APPLY    DBS1.NET 5.4.312     ORA-00001: 一意制約 (HR.JOB_ID_PK) に反しています
```

適用エラーがある場合は、エラーが発生したトランザクションを再実行するか、そのトランザクションを削除できます。エラーが発生したトランザクションを再実行する場合は、まずトランザクションでエラーが呼び出される原因となった条件を訂正します。

エラーが発生したトランザクションを削除する場合に、複数のデータベース間でデータを共有していると、手動によるデータの再同期化が必要になることがあります。データを手動で再同期化する場合は、必要に応じて適切なセッション・タグを設定してください。

関連項目：

- 4-34 ページ「例外キュー」
- 14-31 ページ「適用エラーの管理」
- 適用エラーに考えられる原因については、4-10 ページの「表に DML 変更を適用する際の考慮事項」を参照してください。
- 16-24 ページ「現行セッションの Streams タグの管理」

適用エラーの詳細情報の表示

この項では、データベースの例外キューにあるエラー・トランザクションの詳細情報を表示するために使用できる SQL スクリプトについて説明します。これらのスクリプトは LCR イベント情報を表示するように設計されていますが、環境で使用される非 LCR イベント情報を表示するように拡張できます。

これらのスクリプトを使用するには、次の手順を実行します。

1. DBA_APPLY_ERROR ビューの明示的な SELECT 権限の付与
2. SYS.AnyData オブジェクト内の値を出力するプロシージャの作成
3. 指定した LCR を出力するプロシージャの作成
4. 全例外キューにある全 LCR を出力するプロシージャの作成
5. 特定のトランザクションに関するすべてのエラー LCR を出力するプロシージャの作成

注意： これらのスクリプトでは、LCR イベントにおける VARCHAR2 値の最初の 255 文字のみ表示されます。

手順 1 DBA_APPLY_ERROR ビューの明示的な SELECT 権限の付与

ここで説明する print_errors および print_transaction プロシージャを作成して実行するユーザーには、DBA_APPLY_ERROR データ・ディクショナリ・ビューに対する明示的な SELECT 権限を付与する必要があります。この権限は、ロールを介しては付与できません。

1. 権限を付与できる管理ユーザーとして接続します。
2. DBA_APPLY_ERROR データ・ディクショナリ・ビューの SELECT 権限を適切なユーザーに付与します。たとえば、この権限を strmadmin ユーザーに付与するには、次の文を実行します。

   ```
   GRANT SELECT ON DBA_APPLY_ERROR TO strmadmin;
   ```

3. 手順 2 で権限を付与したユーザーとしてデータベースに接続します。

手順 2 SYS.AnyData オブジェクト内の値を出力するプロシージャの作成

次のプロシージャを実行すると、選択した値の型について指定した SYS.AnyData オブジェクト内の値が出力されます。

```
CREATE OR REPLACE PROCEDURE print_any(data IN SYS.AnyData) IS
  tn  VARCHAR2(61);
  str VARCHAR2(255);
  chr CHAR(255);
  num NUMBER;
  dat DATE;
  rw  RAW(4000);
  res NUMBER;
BEGIN
  IF data IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('NULL value');
    RETURN;
  END IF;
  tn := data.GETTYPENAME();
  IF tn = 'SYS.VARCHAR2' THEN
    res := data.GETVARCHAR2(str);
    DBMS_OUTPUT.PUT_LINE(str);
  ELSIF tn = 'SYS.CHAR' then
    res := data.GETCHAR(chr);
    DBMS_OUTPUT.PUT_LINE(chr);
  ELSIF tn = 'SYS.VARCHAR' THEN
    res := data.GETVARCHAR(chr);
    DBMS_OUTPUT.PUT_LINE(chr);
  ELSIF tn = 'SYS.NUMBER' THEN
    res := data.GETNUMBER(num);
    DBMS_OUTPUT.PUT_LINE(num);
  ELSIF tn = 'SYS.DATE' THEN
    res := data.GETDATE(dat);
    DBMS_OUTPUT.PUT_LINE(dat);
  ELSIF tn = 'SYS.RAW' THEN
    res := data.GETRAW(rw);
    DBMS_OUTPUT.PUT_LINE(RAWTOHEX(rw));
  ELSE
    DBMS_OUTPUT.PUT_LINE('typename is ' || tn);
  END IF;
END print_any;
/
```

手順 3 指定した LCR を出力するプロシージャの作成

次のプロシージャを実行すると、指定した LCR が出力されます。このプロシージャでは、17-36 ページの「SYS.AnyData オブジェクト内の値を出力するプロシージャの作成」で作成した print_any プロシージャがコールされます。

```
CREATE OR REPLACE PROCEDURE print_lcr(lcr IN SYS.ANYDATA) IS
  typenm   VARCHAR2(61);
  ddllcr   SYS.LCR$_DDL_RECORD;
  proclcr  SYS.LCR$_PROCEDURE_RECORD;
  rowlcr   SYS.LCR$_ROW_RECORD;
  res      NUMBER;
  newlist  SYS.LCR$_ROW_LIST;
  oldlist  SYS.LCR$_ROW_LIST;
  ddl_text CLOB;
BEGIN
  typenm := lcr.GETTYPENAME();
  DBMS_OUTPUT.PUT_LINE('type name: ' || typenm);
  IF (typenm = 'SYS.LCR$_DDL_RECORD') THEN
    res := lcr.GETOBJECT(ddllcr);
    DBMS_OUTPUT.PUT_LINE('source database: ' ||
                         ddllcr.GET_SOURCE_DATABASE_NAME);
    DBMS_OUTPUT.PUT_LINE('owner: ' || ddllcr.GET_OBJECT_OWNER);
    DBMS_OUTPUT.PUT_LINE('object: ' || ddllcr.GET_OBJECT_NAME);
    DBMS_OUTPUT.PUT_LINE('is tag null: ' || ddllcr.IS_NULL_TAG);
    DBMS_LOB.CREATETEMPORARY(ddl_text, TRUE);
    ddllcr.GET_DDL_TEXT(ddl_text);
    DBMS_OUTPUT.PUT_LINE('ddl: ' || ddl_text);
    DBMS_LOB.FREETEMPORARY(ddl_text);
  ELSIF (typenm = 'SYS.LCR$_ROW_RECORD') THEN
    res := lcr.GETOBJECT(rowlcr);
    DBMS_OUTPUT.PUT_LINE('source database: ' ||
                         rowlcr.GET_SOURCE_DATABASE_NAME);
    DBMS_OUTPUT.PUT_LINE('owner: ' || rowlcr.GET_OBJECT_OWNER);
    DBMS_OUTPUT.PUT_LINE('object: ' || rowlcr.GET_OBJECT_NAME);
    DBMS_OUTPUT.PUT_LINE('is tag null: ' || rowlcr.IS_NULL_TAG);
    DBMS_OUTPUT.PUT_LINE('command_type: ' || rowlcr.GET_COMMAND_TYPE);
    oldlist := rowlcr.GET_VALUES('old');
    FOR i IN 1..oldlist.COUNT LOOP
      IF oldlist(i) IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('old(' || i || '): ' || oldlist(i).column_name);
        print_any(oldlist(i).data);
      END IF;
    END LOOP;
    newlist := rowlcr.GET_VALUES('new', 'n');
    FOR i in 1..newlist.count LOOP
      IF newlist(i) IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('new(' || i || '): ' || newlist(i).column_name);
```

```
        print_any(newlist(i).data);
      END IF;
    END LOOP;
    ELSE
      DBMS_OUTPUT.PUT_LINE('Non-LCR Message with type ' || typenm);
  END IF;
END print_lcr;
/
```

手順 4 全例外キューにある全 LCR を出力するプロシージャの作成

次のプロシージャを実行すると、データベースの全例外キューにある LCR がすべて出力されます。このプロシージャでは、17-37 ページの「指定した LCR を出力するプロシージャの作成」で作成した print_lcr プロシージャがコールされます。

```
CREATE OR REPLACE PROCEDURE print_errors IS
  CURSOR c IS
    SELECT LOCAL_TRANSACTION_ID,
           SOURCE_DATABASE,
           MESSAGE_COUNT,
           ERROR_NUMBER,
           ERROR_MESSAGE
      FROM DBA_APPLY_ERROR
      ORDER BY SOURCE_DATABASE, SOURCE_COMMIT_SCN;
  i      NUMBER;
  txnid  VARCHAR2(30);
  source VARCHAR2(128);
  msgcnt NUMBER;
  errnum NUMBER := 0;
  errno  NUMBER;
  errmsg VARCHAR2(128);
  lcr    SYS.AnyData;
  r      NUMBER;
BEGIN
  FOR r IN c LOOP
    errnum := errnum + 1;
    msgcnt := r.MESSAGE_COUNT;
    txnid  := r.LOCAL_TRANSACTION_ID;
    source := r.SOURCE_DATABASE;
    errmsg := r.ERROR_MESSAGE;
    errno  := r.ERROR_NUMBER;
    DBMS_OUTPUT.PUT_LINE('************************************************');
    DBMS_OUTPUT.PUT_LINE('----- ERROR #' || errnum);
    DBMS_OUTPUT.PUT_LINE('----- Local Transaction ID: ' || txnid);
    DBMS_OUTPUT.PUT_LINE('----- Source Database: ' || source);
    DBMS_OUTPUT.PUT_LINE('----Error Number: '||errno);
    DBMS_OUTPUT.PUT_LINE('----Message Text: '||errmsg);
    FOR i IN 1..msgcnt LOOP
```

```
      DBMS_OUTPUT.PUT_LINE('--message: ' || i);
        lcr := DBMS_APPLY_ADM.GET_ERROR_MESSAGE(i, txnid);
        print_lcr(lcr);
    END LOOP;
  END LOOP;
END print_errors;
/
```

このプロシージャを作成後に実行するには、次のように入力します。

```
SET SERVEROUTPUT ON SIZE 1000000

EXEC print_errors
```

手順 5 特定のトランザクションに関するすべてのエラー LCR を出力するプロシージャの作成

次のプロシージャを実行すると、特定のトランザクションについて例外キューにある LCR がすべて出力されます。このプロシージャでは、17-37 ページの「指定した LCR を出力するプロシージャの作成」で作成した print_lcr プロシージャがコールされます。

```
CREATE OR REPLACE PROCEDURE print_transaction(ltxnid IN VARCHAR2) IS
  i      NUMBER;
  txnid  VARCHAR2(30);
  source VARCHAR2(128);
  msgcnt NUMBER;
  errno  NUMBER;
  errmsg VARCHAR2(128);
  lcr    SYS.ANYDATA;
BEGIN
  SELECT LOCAL_TRANSACTION_ID,
         SOURCE_DATABASE,
         MESSAGE_COUNT,
         ERROR_NUMBER,
         ERROR_MESSAGE
      INTO txnid, source, msgcnt, errno, errmsg
      FROM DBA_APPLY_ERROR
      WHERE LOCAL_TRANSACTION_ID =  ltxnid;
  DBMS_OUTPUT.PUT_LINE('----- Local Transaction ID: ' || txnid);
  DBMS_OUTPUT.PUT_LINE('----- Source Database: ' || source);
  DBMS_OUTPUT.PUT_LINE('----Error Number: '||errno);
  DBMS_OUTPUT.PUT_LINE('----Message Text: '||errmsg);
  FOR i IN 1..msgcnt LOOP
  DBMS_OUTPUT.PUT_LINE('--message: ' || i);
    lcr := DBMS_APPLY_ADM.GET_ERROR_MESSAGE(i, txnid); -- gets the LCR
    print_lcr(lcr);
  END LOOP;
END print_transaction;
/
```

このプロシージャを作成後に実行するには、エラー・トランザクションのローカル・トランザクション識別子を渡します。たとえば、ローカル・トランザクション識別子が 1.17.2485 の場合は、次のように入力します。

```
SET SERVEROUTPUT ON SIZE 1000000

EXEC print_transaction('1.17.2485')
```

ルールおよびルールベースの変換の監視

ここでは、ルールおよびルールベースの変換に関する情報を表示するために実行できる問合せについて説明します。

- Streams のプロセスまたは伝播で使用される Streams ルールの表示
- Streams のルールの条件の表示
- 各ルール・セットの評価コンテキストの表示
- 評価コンテキストで使用される表に関する情報の表示
- 評価コンテキストで使用される変数に関する情報の表示
- ルール・セットの全ルールの表示
- ルール・セット内の各ルールの条件の表示
- 条件に指定パターンが含まれる各ルールの列挙
- ルール・セット内のルールベースの変換の表示

関連項目：

- 第 5 章「ルール」
- 第 6 章「Streams でのルールの使用方法」
- 第 15 章「ルールおよびルールベースの変換の管理」

Streams のプロセスまたは伝播で使用される Streams ルールの表示

Streams のルールとは、取得プロセス、伝播または適用プロセス用に DBMS_STREAMS_ADM パッケージを使用して作成するルールです。これらのルールによって、取得プロセス、伝播または適用プロセスの動作が決定されます。たとえば、hr.employees 表に対する DML 変更について取得ルールが TRUE と評価されると、取得プロセスでこの表に対する DML 変更が取得されます。

Streams のルールを表示するには、次のデータ・ディクショナリ・ビューを問い合せます。

- ALL_STREAMS_GLOBAL_RULES
- DBA_STREAMS_GLOBAL_RULES
- ALL_STREAMS_SCHEMA_RULES
- DBA_STREAMS_SCHEMA_RULES
- ALL_STREAMS_TABLE_RULES
- DBA_STREAMS_TABLE_RULES

> **注意：** これらのビューに表示されるのは、DBMS_STREAMS_ADM パッケージまたは Oracle Enterprise Manager の Streams ツールを使用して作成されたルールのみです。DBMS_RULE_ADM パッケージによってこれらのルールに対して手動で行われた変更や、DBMS_RULE_ADM パッケージを使用して作成されたルールは、表示されません。

たとえば、次の問合せを実行すると、適用プロセス strm01_apply のスキーマ・ルールがすべて表示されます。

```
COLUMN SCHEMA_NAME HEADING 'Schema|Name' FORMAT A10
COLUMN SOURCE_DATABASE HEADING 'Source' FORMAT A10
COLUMN RULE_TYPE HEADING 'Rule Type' FORMAT A10
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A10
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A10
COLUMN INCLUDE_TAGGED_LCR HEADING 'Apply|Tagged|LCRs?' FORMAT A15

SELECT SCHEMA_NAME, SOURCE_DATABASE, RULE_TYPE,
       RULE_NAME, RULE_OWNER, INCLUDE_TAGGED_LCR
  FROM DBA_STREAMS_SCHEMA_RULES
  WHERE STREAMS_NAME = 'STRM01_APPLY' AND STREAMS_TYPE = 'APPLY';
```

出力は次のようになります。

```
                                                        Apply
Schema                                                  Tagged
Name       Source     Rule Type  Rule Name  Rule Owner LCRs?
---------- ---------- ---------- ---------- ---------- ---------------
HR         DBS1.NET   DML        HR1        STRMADMIN  NO
```

これらの結果は、dbs1.net データベースで発生した hr スキーマに対する DML 変更を含む LCR が、適用プロセスによって適用されることを示します。適用プロセスのルール・セットには、適用プロセスに対してこれらの変更を適用するように指定するルール hr1 があり、その所有者は strmadmin ユーザーです。また、この適用プロセスでは、LCR 内のタグが NULL の場合にのみ、これらの変更が適用されます。

関連項目： 6-3 ページ「システム作成ルール」

Streams のルールの条件の表示

Streams のルール名とレベルがわかっている場合は、そのルール条件を表示できます。レベルは、グローバル、スキーマまたは表のいずれかです。

たとえば、17-41 ページの「Streams のプロセスまたは伝播で使用される Streams ルールの表示」の問合せで戻されるルールを考えます。Streams のスキーマ・ルールの名前は hr1 で、次の問合せを実行するとその条件を表示できます。

```
SELECT RULE_CONDITION "Schema Rule Condition"
  FROM DBA_STREAMS_SCHEMA_RULES
  WHERE RULE_NAME = 'HR1' AND
        RULE_OWNER = 'STRMADMIN';
```

出力は次のようになります。

```
Schema Rule Condition
--------------------------------------------------------------------
(:dml.get_object_owner() = 'HR' and :dml.is_null_tag() = 'Y' and
:dml.get_source_database_name() = 'DBS1.NET' )
```

関連項目：

- 5-2 ページ「ルール条件」
- 6-3 ページ「システム作成ルール」

各ルール・セットの評価コンテキストの表示

次の問合せを実行すると、データベース内のルール・セットごとにデフォルトの評価コンテキストが表示されます。

```
COLUMN RULE_SET_OWNER HEADING 'Rule Set|Owner' FORMAT A15
COLUMN RULE_SET_NAME HEADING 'Rule Set Name' FORMAT A15
COLUMN RULE_SET_EVAL_CONTEXT_OWNER HEADING 'Eval Context|Owner' FORMAT A12
COLUMN RULE_SET_EVAL_CONTEXT_NAME HEADING 'Eval Context Name' FORMAT A27

SELECT RULE_SET_OWNER,
       RULE_SET_NAME,
       RULE_SET_EVAL_CONTEXT_OWNER,
       RULE_SET_EVAL_CONTEXT_NAME
  FROM DBA_RULE_SETS;
```

出力は次のようになります。

```
Rule Set                        Eval Context
Owner           Rule Set Name   Owner        Eval Context Name
--------------- --------------- ------------ --------------------------
STRMADMIN       RULESET$_2      SYS          STREAMS$_EVALUATION_CONTEXT
STRMADMIN       STRM02_QUEUE_R  STRMADMIN    AQ$_STRM02_QUEUE_TABLE_V
STRMADMIN       APPLY_OE_RS     STRMADMIN    OE_EVAL_CONTEXT
STRMADMIN       OE_QUEUE_R      STRMADMIN    AQ$_OE_QUEUE_TABLE_V
STRMADMIN       AQ$_1_RE        STRMADMIN    AQ$_OE_QUEUE_TABLE_V
SUPPORT         RS              SUPPORT      EVALCTX
```

関連項目： 5-5 ページ「ルール評価コンテキスト」

評価コンテキストで使用される表に関する情報の表示

次の問合せを実行すると、support ユーザーが所有する評価コンテキスト evalctx で使用される表に関する情報が表示されます。

```
COLUMN TABLE_ALIAS HEADING 'Table Alias' FORMAT A20
COLUMN TABLE_NAME HEADING 'Table Name' FORMAT A40

SELECT TABLE_ALIAS,
       TABLE_NAME
  FROM DBA_EVALUATION_CONTEXT_TABLES
  WHERE EVALUATION_CONTEXT_OWNER = 'SUPPORT' AND
        EVALUATION_CONTEXT_NAME = 'EVALCTX';
```

出力は次のようになります。

```
Table Alias          Table Name
-------------------- ----------------------------------------
PROB                 problems
```

関連項目： 5-5 ページ「ルール評価コンテキスト」

評価コンテキストで使用される変数に関する情報の表示

次の問合せを実行すると、support ユーザーが所有する評価コンテキスト evalctx で使用される変数に関する情報が表示されます。

```
COLUMN VARIABLE_NAME HEADING 'Variable Name' FORMAT A15
COLUMN VARIABLE_TYPE HEADING 'Variable Type' FORMAT A15
COLUMN VARIABLE_VALUE_FUNCTION HEADING 'Variable Value|Function' FORMAT A20
COLUMN VARIABLE_METHOD_FUNCTION HEADING 'Variable Method|Function' FORMAT A20

SELECT VARIABLE_NAME,
       VARIABLE_TYPE,
       VARIABLE_VALUE_FUNCTION,
       VARIABLE_METHOD_FUNCTION
  FROM DBA_EVALUATION_CONTEXT_VARS
  WHERE EVALUATION_CONTEXT_OWNER = 'SUPPORT' AND
        EVALUATION_CONTEXT_NAME = 'EVALCTX';
```

出力は次のようになります。

```
                                 Variable Value       Variable Method
Variable Name   Variable Type    Function             Function
--------------- ---------------- -------------------- --------------------
CURRENT_TIME    DATE             timefunc
```

関連項目： 5-5 ページ「ルール評価コンテキスト」

ルール・セットの全ルールの表示

この項で説明する問合せを実行すると、ルール・セット内の全ルールに関する情報が表示されます。

- ルールの所有者。
- ルール名。
- 存在する場合は、ルールの評価コンテキスト。ルールに評価コンテキストがなく、そのルールをルール・セットに追加するときに ADD_RULE プロシージャで評価コンテキストが指定されていない場合は、ルール・セットの評価コンテキストが継承されます。
- ルールに評価コンテキストがある場合は、評価コンテキストの所有者。

たとえば、ユーザー strmadmin が所有するルール・セット oe_queue_r 内の各ルールについて前述の情報を表示するには、次の問合せを実行します。

```
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A10
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A20
COLUMN RULE_EVALUATION_CONTEXT_NAME HEADING 'Eval Context Name' FORMAT A27
COLUMN RULE_EVALUATION_CONTEXT_OWNER HEADING 'Eval Context|Owner' FORMAT A11

SELECT R.RULE_OWNER,
       R.RULE_NAME,
       R.RULE_EVALUATION_CONTEXT_NAME,
       R.RULE_EVALUATION_CONTEXT_OWNER
  FROM DBA_RULES R, DBA_RULE_SET_RULES RS
  WHERE RS.RULE_SET_OWNER = 'STRMADMIN' AND
  RS.RULE_SET_NAME = 'OE_QUEUE_R' AND
  RS.RULE_NAME = R.RULE_NAME AND
  RS.RULE_OWNER = R.RULE_OWNER;
```

出力は次のようになります。

```
                                                          Eval Contex
Rule Owner Rule Name            Eval Context Name         Owner
---------- -------------------- ------------------------- -----------
STRMADMIN  HR1                  STREAMS$_EVALUATION_CONTEXT SYS
STRMADMIN  APPLY_LCRS           STREAMS$_EVALUATION_CONTEXT SYS
STRMADMIN  OE_QUEUE$3
STRMADMIN  APPLY_ACTION
```

ルール・セット内の各ルールの条件の表示

次の問合せを実行すると、ユーザー strmadmin が所有するルール・セット hr_queue_r 内の各ルールの条件が表示されます。

```
SET LONGCHUNKSIZE 4000
SET LONG 4000
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A15
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A15
COLUMN RULE_CONDITION HEADING 'Rule Condition' FORMAT A45

SELECT R.RULE_OWNER,
       R.RULE_NAME,
       R.RULE_CONDITION
  FROM DBA_RULES R, DBA_RULE_SET_RULES RS
  WHERE RS.RULE_SET_OWNER = 'STRMADMIN' AND
  RS.RULE_SET_NAME = 'HR_QUEUE_R' AND
  RS.RULE_NAME = R.RULE_NAME AND
  RS.RULE_OWNER = R.RULE_OWNER;
```

出力は次のようになります。

```
Rule Owner      Rule Name       Rule Condition
--------------- --------------- --------------------------------------------
STRMADMIN       APPLY_ACTION     hr.get_hr_action(tab.user_data) = 'APPLY'
STRMADMIN       APPLY_LCRS      :dml.get_object_owner() = 'HR' AND  (:dml.get
                                _object_name() = 'DEPARTMENTS' OR
                                :dml.get_object_name() = 'EMPLOYEES')

STRMADMIN       HR_QUEUE$3      hr.get_hr_action(tab.user_data) != 'APPLY'
```

関連項目：

- 5-2 ページ「ルール条件」
- 6-3 ページ「システム作成ルール」

条件に指定パターンが含まれる各ルールの列挙

条件に指定パターンが含まれるデータベース内の各ルールを列挙するには、DBMS_RULES データ・ディクショナリ・ビューを問い合せ、DBMS_LOB.INSTR ファンクションを使用してルール条件のパターンを検索できます。たとえば、次の問合せでは、条件にパターン 'HR' が含まれる各ルールが列挙されます。

```
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A30
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A30

SELECT RULE_OWNER, RULE_NAME FROM DBA_RULES
  WHERE DBMS_LOB.INSTR(RULE_CONDITION, 'HR', 1, 1) > 0;
```

出力は次のようになります。

```
Rule Owner                     Rule Name
------------------------------ ------------------------------
STRMADMIN                      DEPARTMENTS4
STRMADMIN                      DEPARTMENTS5
STRMADMIN                      DEPARTMENTS6
```

ルール・セット内のルールベースの変換の表示

Streams では、ルールベースの変換は、名前 / 値ペアに STREAMS$_TRANSFORM_FUNCTION 名を含むルールのアクション・コンテキスト内で指定されます。名前 / 値ペアに含まれる値は、変換を実行する PL/SQL プロシージャの名前です。

次の問合せを実行すると、ルール・セット RULESET$_4 内のルールに指定されているルールベースの変換がすべて表示されます。

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A20
COLUMN ACTION_CONTEXT_VALUE HEADING 'Transformation Procedure' FORMAT A40

SELECT
    r.RULE_NAME,
    ac.NVN_VALUE.ACCESSVARCHAR2() ACTION_CONTEXT_VALUE
  FROM DBA_RULES r,
       TABLE(R.RULE_ACTION_CONTEXT.ACTX_LIST) ac,
       DBA_RULE_SET_RULES s
  WHERE ac.NVN_NAME      = 'STREAMS$_TRANSFORM_FUNCTION' AND
        s.RULE_SET_NAME  = 'RULESET$_4' AND
        s.RULE_SET_OWNER = 'STRMADMIN' AND
        r.RULE_NAME      = s.RULE_NAME AND
        r.RULE_OWNER     = s.RULE_OWNER;
```

このルール・セット内のルールにルールベースの変換が指定されている場合、出力は次のようになります。

```
Rule Name            Transformation Procedure
-------------------- ----------------------------------------
DEPARTMENTS7         hr.executive_to_management
DEPARTMENTS6         hr.executive_to_management
DEPARTMENTS5         hr.executive_to_management
```

関連項目：

- 6-23 ページ「ルールベースの変換」
- 15-11 ページ「ルールベースの変換の管理」

Streams タグの監視

ここでは、現行セッション用と適用プロセス用の Streams タグを表示するために実行できる問合せについて説明します。

- 現行セッション用のタグ値の表示
- 適用プロセス用のタグ値の表示

関連項目：

- 第 8 章「Streams のタグ」
- 16-24 ページ「Streams タグの管理」
- DBMS_STREAMS パッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

現行セッション用のタグ値の表示

次のように DUAL ビューを問い合せると、現行セッションについてすべての REDO エントリ内で生成されたタグ値を表示できます。

```
SELECT DBMS_STREAMS.GET_TAG FROM DUAL;
```

出力は次のようになります。

```
GET_TAG
--------------------------------------------------------------------------------
1D
```

また、DBMS_STREAMS.GET_TAG ファンクションをコールすると、セッション用のタグを判断できます。

適用プロセス用のタグ値の表示

DBA_APPLY データ・ディクショナリ・ビューで APPLY_TAG の値を問い合せると、適用プロセスによって生成されるすべての REDO エントリのデフォルト・タグを取得できます。たとえば、適用プロセス strm01_apply によって REDO エントリ内で生成されるタグの 16 進値を取得するには、次の問合せを実行します。

```
SELECT APPLY_TAG "Tag Value for strm01_apply"
  FROM DBA_APPLY WHERE APPLY_NAME = 'STRM01_APPLY';
```

出力は次のようになります。

```
Tag Value for strm01_apply
--------------------------------------------------------------------------------
00
```

適用プロセスに関連付けられているハンドラまたは変換ファンクションでは、DBMS_STREAMS.GET_TAG ファンクションをコールしてタグを取得できます。

18

Streams 環境のトラブルシューティング

この章では、Streams 環境で一般的な問題を識別して解決する方法について説明します。

この章の内容は次のとおりです。

- 取得に関する問題のトラブルシューティング
- 伝播に関する問題のトラブルシューティング
- 適用に関する問題のトラブルシューティング
- ルールおよびルールベースの変換に関する問題のトラブルシューティング
- トレース・ファイルとアラート・ログでの問題のチェック

取得に関する問題のトラブルシューティング

取得プロセスで予期したとおりに変更が取得されない場合や、取得プロセスに他の問題が発生する場合は、次のチェックリストを使用して問題を識別し、解決します。

- 取得プロセスが有効化されているかどうか
- 現行の取得プロセスかどうか
- LOG_PARALLELISM は 1 に設定されているか
- LOGMNR_MAX_PERSISTENT_SESSIONS は十分に大きい値に設定されているか

関連項目：

- 第 2 章「Streams の取得プロセス」
- 第 12 章「取得プロセスの管理」
- 17-3 ページ「Streams の取得プロセスの監視」

取得プロセスが有効化されているかどうか

変更が取得されるのは、取得プロセスが有効化されている場合のみです。取得プロセスが有効化されているか、無効化されているか、異常終了しているかは、DBA_CAPTURE データ・ディクショナリ・ビューを問い合せるとチェックできます。

たとえば、取得プロセス CAPTURE が有効化されているかどうかをチェックするには、次の問合せを実行します。

```
SELECT STATUS FROM DBA_CAPTURE WHERE CAPTURE_NAME = 'CAPTURE';
```

取得プロセスが無効化されている場合、出力は次のようになります。

```
STATUS
--------
DISABLED
```

取得プロセスが無効化されているか、異常終了している場合は、再起動します。取得プロセスが無効化または異常終了された原因が不明な場合は、そのトレース・ファイルをチェックしてください。

関連項目：

- 12-4 ページ「取得プロセスの起動」
- 18-23 ページ「トレース・ファイルとアラート・ログでの問題のチェック」
- Oracle Real Application Clusters 環境で取得プロセスを再起動する方法については、2-15 ページの「Streams の取得プロセスと Oracle Real Application Clusters」を参照してください。

現行の取得プロセスかどうか

取得プロセスで最近の変更が取得されていない場合は、遅延が原因となっていることがあります。遅延の有無をチェックするには、V$STREAMS_CAPTURE 動的パフォーマンス・ビューを問い合せます。取得プロセスの待機時間が長い場合は、parallelism 取得プロセス・パラメータの設定を調整すると、パフォーマンスを改善できることがあります。

関連項目：

- 17-7 ページ「単一の取得プロセスに関する REDO ログのスキャン待機時間の判断」
- 17-8 ページ「単一の取得プロセスに関するイベントのエンキュー待機時間の判断」
- 2-23 ページ「取得プロセスの並列性」
- 12-7 ページ「取得プロセスのパラメータの設定」

LOG_PARALLELISM は 1 に設定されているか

LOG_PARALLELISM 初期化パラメータでは、Oracle 内での REDO 割当ての並行性レベルを指定します。データベース上で 1 つ以上の取得プロセスを実行する予定の場合は、このパラメータを 1 に設定する必要があります。この初期化パラメータが 2 以上に設定されていると、エラー ORA-01374 が発生することがあります。

このパラメータを 1 に設定しても、取得の並列性には影響しません。取得プロセスの並列性は、DBMS_CAPTURE_ADM パッケージの SET_PARAMETER プロシージャを使用して設定できます。

LOGMNR_MAX_PERSISTENT_SESSIONS は十分に大きい値に設定されているか

LOGMNR_MAX_PERSISTENT_SESSIONS 初期化パラメータでは、すべてのセッションでインスタンスによって生成された REDO ログをマイニング中に、同時にアクティブになっている LogMiner 永続マイニング・セッションの最大数を指定します。1 つのデータベース上で複数の Streams 取得プロセスを実行する予定の場合は、このパラメータをその取得プロセス数以上の値に設定します。

取得プロセスを削除できず、複数の取得プロセスを使用している場合は、LOGMNR_MAX_PERSISTENT_SESSIONS 初期化パラメータの設定値が小さすぎることが原因となっている可能性があります。この初期化パラメータの設定値を大きくしてから、取得プロセスの削除操作を再試行してください。

または、この初期化パラメータのサイズを大きくしない場合は、実行中の取得プロセスを 1 つ以上停止してから、取得プロセスの削除操作を再試行してください。削除操作に成功した場合は、停止した取得プロセスを再起動します。

伝播に関する問題のトラブルシューティング

伝播で予期したとおりに変更が伝播されない場合は、次のチェックリストを使用して伝播の問題を識別して解決します。

- 伝播には適切なソース・キューと宛先キューが使用されているか
- 伝播で使用される伝播ジョブが有効化されているか
- 十分な数のジョブ・キュー・プロセスがあるか
- Streams キューのセキュリティは適切に構成されているか

関連項目：

- 第 3 章「Streams のステージングと伝播」
- 第 13 章「ステージングと伝播の管理」
- 17-15 ページ「Streams 伝播および伝播ジョブの監視」

伝播には適切なソース・キューと宛先キューが使用されているか

伝播の宛先キューにイベントが予期したとおりに表示されない場合、その伝播は適切なソース・キューから適切な宛先キューにイベントを伝播させるように構成されていない可能性があります。

たとえば、伝播 dbs1_to_dbs2 のソース・キューと宛先キューをチェックするには、次の問合せを実行します。

```
COLUMN SOURCE_QUEUE HEADING 'Source Queue' FORMAT A35
COLUMN DESTINATION_QUEUE HEADING 'Destination Queue' FORMAT A35

SELECT
  p.SOURCE_QUEUE_OWNER||'.'||
    p.SOURCE_QUEUE_NAME||'@'||
    g.GLOBAL_NAME SOURCE_QUEUE,
  p.DESTINATION_QUEUE_OWNER||'.'||
    p.DESTINATION_QUEUE_NAME||'@'||
    p.DESTINATION_DBLINK DESTINATION_QUEUE
  FROM DBA_PROPAGATION p, GLOBAL_NAME g
  WHERE p.PROPAGATION_NAME = 'DBS1_TO_DBS2';
```

出力は次のようになります。

```
Source Queue                          Destination Queue
------------------------------------ ------------------------------------
STRMADMIN.STREAMS_QUEUE@DBS1.NET    STRMADMIN.STREAMS_QUEUE@DBS2.NET
```

伝播に適切なキューが使用されていない場合は、新規の伝播を作成してください。既存の伝播が環境に適切でない場合は、その削除が必要になることがあります。

伝播で使用される伝播ジョブが有効化されているか

伝播ジョブでイベントを伝播させるには、その伝播ジョブの伝播スケジュールを有効化する必要があります。伝播でイベントが予期したとおりに伝播されていない場合は、この伝播の伝播ジョブ・スケジュールが有効化されていない可能性があります。

この項で説明する問合せを実行すると、伝播ジョブのスケジュールに関する次の情報を検索できます。

- ソース・キューから宛先キューへのイベントの伝播に使用されたデータベース・リンク。
- 伝播スケジュールが有効化されているか、無効化されているか。
- 最終イベントの伝播に使用されたジョブ・キュー・プロセス。
- 伝播スケジュールの実行が試行されたときの連続失敗回数。この数が 16 に達すると、スケジュールは自動的に無効化されます。

- 伝播エラーがある場合は、最後のエラーの発生時刻。
- 伝播エラーがある場合は、最後のエラーのエラー・メッセージ。

たとえば、伝播 dbs1_to_dbs2 で使用される伝播ジョブが有効化されているかどうかをチェックするには、次の問合せを実行します。

```
COLUMN DESTINATION_DBLINK HEADING 'Destination|DB Link' FORMAT A15
COLUMN SCHEDULE_DISABLED HEADING 'Schedule' FORMAT A8
COLUMN PROCESS_NAME HEADING 'Process' FORMAT A7
COLUMN FAILURES HEADING 'Number of|Failures' FORMAT 9999
COLUMN LAST_ERROR_TIME HEADING 'Last Error Time' FORMAT A15
COLUMN LAST_ERROR_MSG HEADING 'Last Error Message' FORMAT A18

SELECT p.DESTINATION_DBLINK,
       DECODE(s.SCHEDULE_DISABLED,
                'Y', 'Disabled',
                'N', 'Enabled') SCHEDULE_DISABLED,
       s.PROCESS_NAME,
       s.FAILURES,
       s.LAST_ERROR_TIME,
       s.LAST_ERROR_MSG
  FROM DBA_QUEUE_SCHEDULES s, DBA_PROPAGATION p
  WHERE p.PROPAGATION_NAME = 'DBS1_TO_DBS2'
  AND p.DESTINATION_DBLINK = s.DESTINATION
  AND s.SCHEMA = p.SOURCE_QUEUE_OWNER
  AND s.QNAME = p.SOURCE_QUEUE_NAME;
```

この伝播ジョブのスケジュールが現在は有効化されている場合、出力は次のようになります。

```
Destination                      Number of
DB Link         Schedule Process  Failures Last Error Time Last Error Message
--------------- -------- ------- --------- --------------- ------------------
DBS2.NET        Enabled  J001            0
```

問題を解決するために次の操作を試行してください。

- 伝播ジョブが無効化されていて、まだ DBMS_AQADM パッケージの ENABLE_PROPAGATION_SCHEDULE プロシージャを実行していない場合は、このプロシージャを使用して有効化できます。
- 伝播ジョブが無効化されており、その原因が不明の場合は、イベントを最後に伝播したプロセスのトレース・ファイルをチェックします。前述の出力では、このプロセスは J001 です。
- 伝播ジョブは有効化されているが、イベントを伝播していない場合は、そのスケジュールを解除してから再設定してください。

関連項目：

- 13-10 ページ「伝播ジョブの有効化」
- 18-23 ページ「トレース・ファイルとアラート・ログでの問題のチェック」
- 13-13 ページ「伝播ジョブのスケジュール解除」
- 13-11 ページ「伝播ジョブのスケジューリング」
- 17-17 ページ「伝播ジョブのスケジュールの表示」

十分な数のジョブ・キュー・プロセスがあるか

伝播ジョブでは、ジョブ・キュー・プロセスを使用してイベントが伝播されます。伝播を実行する各データベース・インスタンス内で、JOB_QUEUE_PROCESSES 初期化パラメータが 2 以上に設定されていることを確認してください。同時に実行されるすべてのジョブに十分に対応できる値に設定する必要があります。

関連項目：

- 11-4 ページ「Streams に関連する初期化パラメータの設定」
- 伝播ジョブを使用する場合の JOB_QUEUE_PROCESSES 初期化パラメータ設定の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』の伝播機能の説明を参照してください。
- JOB_QUEUE_PROCESSES 初期化パラメータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。
- ジョブ・キューの詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

Streams キューのセキュリティは適切に構成されているか

Streams のキューは保護キューであり、ユーザーがその操作を実行できるようにセキュリティを適切に構成する必要があります。Streams のキューのセキュリティが適切に構成されていないと、次のいずれかのエラーが発生する可能性があります。

- ORA-24093: AQ エージェントはデータベース・ユーザーの権限を付与されていません。
- ORA-25224: 保護キューにエンキューするために送信者名を指定する必要があります。

関連項目： 3-21 ページ「保護キュー」

ORA-24093: AQ エージェントはデータベース・ユーザーの権限を付与されていません。

エージェントには、エンキュー操作とデキュー操作の両方について、保護キューへのアクセス権を明示的に付与する必要があります。これらの権限をエージェントに付与するには、DBMS_AQADM パッケージの ENABLE_DB_ACCESS プロシージャを使用します。

たとえば、エージェント explicit_dq にデータベース・ユーザー oe の権限を付与するには、次のプロシージャを実行します。

```
BEGIN
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name  => 'explicit_dq',
    db_username => 'oe');
END;
/
```

データベース内のエージェントの権限をチェックするには、次の問合せを実行します。

```
SELECT AGENT_NAME "Agent", DB_USERNAME "User" FROM DBA_AQ_AGENT_PRIVS;
```

出力は次のようになります。

```
Agent                          User
------------------------------ ------------------------------
EXPLICIT_ENQ                   OE
APPLY_OE                       OE
EXPLICIT_DQ                    OE
```

関連項目： エージェントに対する権限付与の詳細例については、13-3 ページの「保護キューでのユーザー操作の有効化」を参照してください。

ORA-25224: 保護キューにエンキューするために送信者名を指定する必要があります。

保護キューにエンキューするには、メッセージ・プロパティで SENDER_ID をそのキューに対する保護キュー権限を持つエージェントに設定する必要があります。

関連項目： エンキュー用の SENDER_ID の設定例については、19-12 ページの「非 LCR イベントをエンキューするプロシージャの作成」を参照してください。

適用に関する問題のトラブルシューティング

適用プロセスで予期したとおりに変更が適用されない場合は、次のチェックリストを使用して適用の問題を識別して解決します。

- 適用プロセスが有効化されているかどうか
- 現行の適用プロセスかどうか
- 適用プロセスで取得イベントとユーザー・エンキュー・イベントのどちらが適用されるか
- カスタム適用ハンドラが指定されているかどうか
- 適用プロセスが依存トランザクションを待機しているかどうか
- 例外キューに適用エラーがあるか

関連項目：

- 第 4 章「Streams 適用プロセス」
- 第 14 章「適用プロセスの管理」
- 17-19 ページ「Streams の適用プロセスの監視」

適用プロセスが有効化されているかどうか

変更が適用されるのは、適用プロセスが有効化されている場合のみです。適用プロセスが有効化されているか、無効化されているか、異常終了しているかは、DBA_APPLY データ・ディクショナリ・ビューを問い合せるとチェックできます。

たとえば、適用プロセス APPLY が有効化されているかどうかをチェックするには、次の問合せを実行します。

```
SELECT STATUS FROM DBA_APPLY WHERE APPLY_NAME = 'APPLY';
```

適用プロセスが無効化されている場合、出力は次のようになります。

```
STATUS
--------
DISABLED
```

適用プロセスが無効化されているか、異常終了している場合は、再起動します。適用プロセスが無効化または異常終了された原因が不明な場合は、そのトレース・ファイルをチェックしてください。

関連項目：

- 14-7 ページ「適用プロセスの起動」
- 18-23 ページ「トレース・ファイルとアラート・ログでの問題のチェック」
- Oracle Real Application Clusters 環境で適用プロセスを再起動する方法については、4-27 ページの「Streams の適用プロセスと Oracle Real Application Clusters」を参照してください。

現行の適用プロセスかどうか

適用プロセスで最近の変更が適用されていない場合は、遅延が原因となっていることがあります。適用プロセスの待機時間は、V$STREAMS_APPLY_COORDINATOR 動的パフォーマンス・ビューを問い合せてチェックできます。適用プロセスの待機時間が長い場合は、parallelism 適用プロセス・パラメータの設定を調整すると、パフォーマンスを改善できることがあります。

関連項目：

- 17-30 ページ「イベントが取得されてから適用されるまでの待機時間の判断」
- 4-32 ページ「適用プロセスの並列性」
- 14-11 ページ「適用プロセス・パラメータの設定」

適用プロセスで取得イベントとユーザー・エンキュー・イベントのどちらが適用されるか

適用プロセスで適用できるのは、取得イベントまたはユーザー・エンキュー・イベントのどちらか一方のみです。両方は適用できません。適用プロセスが特定の型のイベントを適用していない場合は、他の型のイベントを適用するように構成されていることが原因の可能性があります。適用プロセスで適用されるイベントの型は、DBA_APPLY データ・ディクショナリ・ビューを問い合せてチェックできます。

たとえば、適用プロセス APPLY で取得イベントとユーザー・エンキュー・イベントのどちらが適用されるかをチェックするには、次の問合せを実行します。

```
COLUMN APPLY_CAPTURED HEADING 'Type of Events Applied' FORMAT A25

SELECT DECODE(APPLY_CAPTURED,
                'YES', 'Captured',
                'NO',  'User-Enqueued') APPLY_CAPTURED
  FROM DBA_APPLY
  WHERE APPLY_NAME = 'APPLY';
```

適用プロセスで取得イベントが適用される場合、出力は次のようになります。

```
Type of Events Applied
-------------------------
Captured
```

適用プロセスで予期した型のイベントが適用されていない場合は、その型のイベントを適用する新規適用プロセスの作成が必要になることがあります。

関連項目：

- 3-3 ページ「取得イベントとユーザー・エンキュー・イベント」
- 12-2 ページ「取得プロセスの作成」

カスタム適用ハンドラが指定されているかどうか

PL/SQL プロシージャを使用すると、適用プロセスでデキューされるイベントをカスタマイズされた方法で処理できます。この種のハンドラには、DML ハンドラ、DDL ハンドラおよびメッセージ・ハンドラがあります。適用プロセスが予期したとおりに動作していない場合は、その適用プロセスで使用されるハンドラ・プロシージャをチェックして、不備を訂正します。これらのプロシージャの名前は、DBA_APPLY_DML_HANDLERS および DBA_APPLY データ・ディクショナリ・ビューを問い合せて検索できます。適用に関する問題を解決するには、ハンドラ・プロシージャの変更または削除が必要になることがあります。

関連項目：

- 適用ハンドラの概要は、4-4 ページの「イベント処理オプション」を参照してください。
- 適用ハンドラの管理については、第 14 章「適用プロセスの管理」を参照してください。
- 適用ハンドラに関する情報が表示される問合せについては、17-22 ページの「適用ハンドラに関する情報の表示」を参照してください。

適用プロセスが依存トランザクションを待機しているかどうか

適用プロセスについて、parallelism パラメータを 1 よりも大きい値に設定し、commit_serialization パラメータを full に設定した場合は、上位 SCN を持つ他のトランザクションに依存するトランザクションがあると、その適用プロセスは対象トランザクション・リスト（ITL）の競合を検出する可能性があります。ITL の競合が発生するのは、トランザクションを作成したセッションがブロック内の ITL スロットを待機していた場合です。この状況が発生するのは、セッションがブロック内で 1 行をロックする必要があるが、他の 1 つ以上のセッションによって同一ブロック内で数行がロックされており、ブロックに空き ITL スロットがない場合です。

ITL の競合は、共有ビットマップ索引の断片が原因でセッションが待機中でも発生する可能性があります。ビットマップ索引は、キー値と ROWID の範囲に索引を付けます。ビットマップ索引の各エントリで実際の表の多数の行に対処できます。同じビットマップ索引の断片で扱われる複数行を 2 つのセッションで更新する必要がある場合、2 番目のセッションは最初のトランザクションが COMMIT または ROLLBACK されるまで待機します。

適用プロセスでこの種の依存性が検出されると、ITL の競合が自動的に解決され、これに関する情報がデータベース用のアラート・ログと適用プロセスのトレース・ファイルに記録されます。デッドロックが検出されている間は適用プロセスが進まないため、ITL の競合は、適用プロセスのパフォーマンスに悪影響を及ぼす可能性があります。

将来の問題を回避するには、次のいずれかの操作を実行します。

- 使用可能な ITL の数を増加させます。そのためには、ALTER TABLE 文を使用して表の INITRANS または MAXTRANS 設定を変更します。
- 適用プロセスの commit_serialization パラメータを none に設定します。
- 適用プロセスの parallelism 適用プロセス・パラメータを 1 に設定します。

関連項目：

- 4-31 ページ「適用プロセスのパラメータ」
- 18-23 ページ「トレース・ファイルとアラート・ログでの問題のチェック」
- INITRANS および MAXTRANS の詳細は、『Oracle9*i* データベース管理者ガイド』および『Oracle9*i* SQL リファレンス』を参照してください。

例外キューに適用エラーがあるか

適用プロセスがイベントを適用できない場合は、そのイベントおよび同じトランザクション内の他のすべてのイベントが、そのキューの例外キューに移動されます。適用エラーを定期的にチェックして、適用できなかったトランザクションの有無を確認する必要があります。適用エラーの有無は、DBA_APPLY_ERROR データ・ディクショナリ・ビューを問い合せてチェックできます。

関連項目：

- 17-34 ページ「適用エラーのチェック」
- 14-31 ページ「適用エラーの管理」

LCR イベントの場合は、次のタイプの適用プロセス・エラーが発生することがあります。

- ORA-01403: データが見つかりません。
- ORA-26687: インスタンス化 SCN が設定されていません。
- ORA-26688: LCR にキーがありません。
- ORA-26689: LCR で列データ型が一致していません。

ORA-01403: データが見つかりません。

通常、このエラーが発生するのは、既存の行の更新が試行され、その行 LCR 内の OLD_VALUES がターゲット・サイト側の現行の値と一致しない場合です。

この問題を解決するには、行 LCR を正常に適用できるように、行の現行の値を更新します。行の変更が接続先データベースの取得プロセスで取得されている場合、接続先サイトでこの手動変更を適用する必要はありません。この場合は、次の手順で操作します。

1. 行を訂正するセッションで適用タグを設定します。このタグは、一部の接続先データベースでの手動変更の適用を防止する値に設定してください。

   ```
   EXEC DBMS_STREAMS.SET_TAG(tag => HEXTORAW('17'));
   ```

2. 行を更新して古い値を訂正します。

3. このエラーまたはすべてのエラーを再実行します。単一のエラーを再実行するには、DBMS_APPLY_ADM パッケージの EXECUTE_ERROR プロシージャを実行し、エラーの原因となったトランザクションのトランザクション識別子を指定します。

   ```
   EXEC DBMS_APPLY_ADM.EXECUTE_ERROR(local_transaction_id => '5.4.312');
   ```

   または、EXECUTE_ALL_ERRORS プロシージャを実行して、適用プロセスのエラーをすべて実行します。

   ```
   EXEC DBMS_APPLY_ADM.EXECUTE_ALL_ERRORS(apply_name => 'APPLY');
   ```

4. 接続先データベースで適用する他の変更を現行セッションで行う場合は、セッションのタグを次のように適切な値にリセットします。

   ```
   EXEC DBMS_STREAMS.SET_TAG(tag => NULL);
   ```

ORA-26687: インスタンス化 SCN が設定されていません。

通常、このエラーが発生する原因は、適用プロセスで変更を適用するオブジェクトにインスタンス化 SCN が設定されていないことです。DBA_APPLY_INSTANTIATED_OBJECTS データ・ディクショナリ・ビューを問い合せると、インスタンス化 SCN を持つオブジェクトをリスト表示できます。

ソース・データベースでオブジェクトをエクスポートし、それを接続先データベースでインポートすると、1 つ以上のオブジェクトのインスタンス化 SCN を設定できます。エクスポート / インポートを使用しない場合は、DBMS_APPLY_ADM パッケージの次の 1 つ以上のプロシージャを実行できます。

- SET_TABLE_INSTANTIATION_SCN
- SET_SCHEMA_INSTANTIATION_SCN
- SET_GLOBAL_INSTANTIATION_SCN

> **関連項目：** 14-33 ページ「接続先データベースでのインスタンス化 SCN の設定」

ORA-26688: LCR にキーがありません。

通常、このエラーは次のいずれかの条件が原因で発生します。

- LCR で変更が適用されるオブジェクトが、接続先データベースに存在しない場合。この場合は、オブジェクトの有無をチェックします。また、ルール条件と適用ハンドラに適切な大 / 小文字を使用していることを確認してください。たとえば、列名がデータ・ディクショナリではすべて大文字になっている場合は、ルール条件と適用ハンドラにもすべて大文字で指定する必要があります。
- LCR で変更が適用される表の主キーに問題がある場合。この場合は、DBA_CONSTRAINTS データ・ディクショナリ・ビューを問い合せて、主キーが使用可能になっているかどうかを確認します。表の主キーが存在しない場合は、DBMS_APPLY_ADM パッケージの SET_KEY_COLUMNS プロシージャを使用して代替キー列を指定します。また、適用対象となる表に主キーがない場合は、エラー ORA-23416 も発生する可能性があります。

  関連項目：

  - 4-10 ページ「表に DML 変更を適用する際の考慮事項」
  - 14-26 ページ「表の代替キー列の管理」

- ソース・データベースでサプリメンタル・ロギングが必要な列に、それが指定されていない場合。この場合、ソース・データベースからの LCR には、キー列の値が含まれない可能性があります。

  関連項目：

  - 17-11 ページ「ソース・データベースのサプリメンタル・ログ・グループの表示」
  - 2-10 ページ「Streams 環境内のサプリメンタル・ロギング」
  - 12-8 ページ「ソース・データベースでのサプリメンタル・ロギングの指定」

ORA-26689: LCR で列データ型が一致していません。

通常、このエラーが発生するのは、ソース・データベースの表にある 1 つ以上の列が、接続先データベースの対応する列と一致しない場合です。ソース・データベースからの LCR には、接続先データベースの表より多数の列が含まれている場合や、1 つ以上の列の型が一致しない場合があります。データベース間で列が異なる場合は、ルールベースの変換を使用してエラーを回避できます。

また、このエラーの発生原因には、ソース・データベースで非キー列に必要なサプリメンタル・ロギングが指定されていないことも考えられます。この場合、ソース・データベースからの LCR には、これらの非キー列に必要な値が含まれない可能性があります。

関連項目：

- 4-10 ページ「表に DML 変更を適用する際の考慮事項」
- 6-23 ページ「ルールベースの変換」
- 15-11 ページ「ルールベースの変換の管理」
- 2-10 ページ「Streams 環境内のサプリメンタル・ロギング」

ルールおよびルールベースの変換に関する問題のトラブルシューティング

取得プロセス、伝播または適用プロセスが予期したとおりに動作しない場合は、その取得プロセス、伝播または適用プロセスのルールまたはルールベースの変換が適切に構成されていないことが問題となっている可能性があります。次のチェックリストを使用し、ルールおよびルールベースの変換の問題を識別して解決します。

- ルールは Streams のプロセスまたは伝播用に適切に構成されているか
- ルールベースの変換が適切に構成されているか

関連項目：

- 第 5 章「ルール」
- 第 6 章「Streams でのルールの使用方法」
- 第 15 章「ルールおよびルールベースの変換の管理」

ルールは Streams のプロセスまたは伝播用に適切に構成されているか

Streams の取得プロセス、伝播または適用プロセスが予期したとおりに動作しない場合は、取得プロセス、伝播または適用プロセスのルール・セット内でルールが適切に構成されていないことが問題の可能性があります。たとえば、取得プロセスで特定の表に対する変更が取得されることを予期していたのに、これらの変更が取得されない場合は、その取得プロセスのルール・セットに、オブジェクトに変更がある場合に TRUE と評価されるルールが含まれていないことが原因の可能性があります。

次のデータ・ディクショナリ・ビューを問い合せると、Streams の特定の取得プロセス、伝播または適用プロセスのルールをチェックできます。

- DBA_STREAMS_TABLE_RULES
- DBA_STREAMS_SCHEMA_RULES
- DBA_STREAMS_GLOBAL_RULES

注意： これらのデータ・ディクショナリ・ビューには、DBMS_STREAMS_ADM パッケージまたは Oracle Enterprise Manager の Streams ツールを使用して作成されたルールに関する情報が表示されます。DBMS_RULE_ADM パッケージを使用して作成されたルールに関する情報は表示されません。DBMS_RULE_ADM パッケージで作成されたルールに関する情報を表示するには、DBA_RULES データ・ディクショナリ・ビューを問い合せます。

関連項目： 17-40 ページ「ルールおよびルールベースの変換の監視」

ルール・セットにルールが割り当てられていない場合と、ルール・セットがない場合とは異なります。たとえば、伝播のルールがないルール・セットを使用すると、伝播では何も伝播されません。伝播に対しまったくルール・セットを使用しない場合、伝播ではソース・キュー内のすべてが伝播されます。

この項の内容は、次のとおりです。

- 表の取得プロセス・ルールをチェックする例
- 表の適用プロセス・ルールをチェックする例
- スキーマ・ルールとグローバル・ルールのチェック
- ルールに関する問題の解決

表の取得プロセス・ルールをチェックする例

たとえば、データベースで取得プロセス CAPTURE を実行しており、hr.departments 表に変更があるときに、この取得プロセスについて TRUE と評価される表ルールをチェックする必要がある場合を考えます。このルールが取得プロセスのルール・セットに含まれているかどうかを判断するには、次の問合せを実行します。

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A30
COLUMN RULE_TYPE HEADING 'Rule Type' FORMAT A30

SELECT RULE_NAME, RULE_TYPE
  FROM DBA_STREAMS_TABLE_RULES
  WHERE STREAMS_NAME   = 'CAPTURE' AND
        STREAMS_TYPE   = 'CAPTURE' AND
        TABLE_OWNER    = 'HR' AND
        TABLE_NAME = 'DEPARTMENTS';
```

出力は次のようになります。

```
Rule Name                      Rule Type
------------------------------ ------------------------------
DEPARTMENTS1                   DML
```

これらの結果によれば、取得プロセス CAPTURE では hr.departments 表に対する DML 変更が取得される必要があります。つまり、取得プロセスのルール・セットには、取得プロセスによって REDO ログ内で hr.departments 表に対する DML 変更が検出されたときに TRUE と評価されるルールが存在します。

問合せ結果には表に対する DDL 型のルールが含まれているため、取得プロセスでは表に対する DDL 変更が取得される必要があります。取得プロセスで表に対する DML 変更と DDL 変更の両方を取得する必要がある場合は、表に対するそれぞれの型のルールが問合せ結果に表示されます。

表の適用プロセス・ルールをチェックする例

適用プロセスで特定の表に変更が適用されることを予期していたのに、これらの変更が適用されない場合は、適用プロセスのキューに LCR がある場合に TRUE と評価されるルールが、その適用プロセスのルール・セットに含まれていないことが原因の可能性があります。取得ルールや伝播ルールの場合と同様に、DBMS_STREAMS_ADM パッケージのプロシージャを使用して作成されたルールは、次のデータ・ディクショナリ・ビューを問い合せてチェックできます。

- DBA_STREAMS_TABLE_RULES
- DBA_STREAMS_SCHEMA_RULES
- DBA_STREAMS_GLOBAL_RULES

また、適用プロセスでは、キュー内のイベントを受信しなければ、そのイベントを適用できません。したがって、適用プロセスで取得イベントを適用している場合は、そのイベントを取得する取得プロセスのルール・セットを適切に構成する必要があります。同様に、イベントが適用プロセスに到達する前に 1 つ以上のデータベースから伝播される場合は、各伝播のルールを適切に構成する必要があります。適用プロセスが依存する取得プロセスまたは伝播のルールが適切に構成されていないと、イベントが適用プロセスのキューに到達しない可能性があります。

取得プロセスが複数のデータベースに伝播されて適用される変更を取得する環境では、次のガイドラインに従って、問題の原因は適用プロセスが依存する取得プロセスや伝播にあるのか、または適用プロセス自体にあるのかを判断できます。

- 取得プロセスの他の接続先データベースがその取得プロセスからの変更を適用していない場合、問題の原因はおそらく取得プロセスまたはその取得プロセス付近の伝播にあると思われます。この場合は、まず取得プロセスのルールが適切に構成されていることを確認し、次にその取得プロセスに最も近い伝播のルールをチェックしてください。
- 取得プロセスの他の接続先データベースがその取得プロセスからの変更を適用している場合、おそらく問題の原因は適用プロセス自体にあるか、その適用プロセス付近の伝播にあると思われます。この場合は、まず適用プロセスのルールが適切に構成されていることを確認し、次にそれに最も近い伝播のルールをチェックしてください。

また、適用ルールをチェックすると、1 つ以上の表にサブセット・ルールが存在する可能性があります。サブセット・ルールが TRUE と評価されるのは、表の特定の行サブセットに対する変更が LCR に含まれている場合のみです。たとえば、hr.departments 表に対する変更があるときに、適用プロセス APPLY について TRUE と評価される表ルールをチェックするには、次の問合せを実行します。

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A20
COLUMN RULE_TYPE HEADING 'Rule Type' FORMAT A20
COLUMN DML_CONDITION HEADING 'Subset Condition' FORMAT A30

SELECT RULE_NAME, RULE_TYPE, DML_CONDITION
  FROM DBA_STREAMS_TABLE_RULES
  WHERE STREAMS_NAME   = 'APPLY' AND
        STREAMS_TYPE   = 'APPLY' AND
        TABLE_OWNER    = 'HR' AND
        TABLE_NAME     = 'DEPARTMENTS';


Rule Name            Rule Type            Subset Condition
-------------------- -------------------- ------------------------------
DEPARTMENTS5         DML                  location_id=1700
DEPARTMENTS6         DML                  location_id=1700
DEPARTMENTS7         DML                  location_id=1700
```

この問合せでは、DML_CONDITION 列に表のサブセット条件が戻されることに注意してください。この例では、hr.departments 表のサブセット・ルールが指定されています。これらのサブセット・ルールは、location_id が 1700 の 1 行が関係する変更が LCR に含まれている場合にのみ、TRUE と評価されます。そのため、適用プロセスですべての変更を表に適用することを予期していても、このサブセット・ルールが原因で、適用プロセスでは location_id が 1700 でない行が関係する変更は廃棄されます。

関連項目：

- サブセット・ルールの概念情報は、4-12 ページの「Streams ルールを使用した行のサブセット化」を参照してください。
- サブセット・ルールを指定する方法の詳細は、6-7 ページの「表レベルのルールの例」を参照してください。

スキーマ・ルールとグローバル・ルールのチェック

スキーマ・ルールまたはグローバル・ルールを使用して、それぞれ特定のスキーマまたはデータベース内の全データベース・オブジェクトに対する変更を取得することもできます。たとえば、hr スキーマに対する変更があるときに、取得プロセス CAPTURE について TRUE と評価されるスキーマ・ルールをチェックするには、次の問合せを実行します。

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A20
COLUMN RULE_TYPE HEADING 'Rule Type' FORMAT A20

SELECT RULE_NAME, RULE_TYPE
  FROM DBA_STREAMS_SCHEMA_RULES
  WHERE STREAMS_NAME   = 'CAPTURE' AND
        SCHEMA_NAME    = 'HR';
```

出力は次のようになります。

```
Rule Name                      Rule Type
------------------------------ ------------------------------
HR7                            DML
HR8                            DDL
```

これらの結果によれば、取得プロセス CAPTURE では、hr スキーマ内の全オブジェクトに対する DML 変更と DDL 変更が取得される必要があります。

取得プロセスについて問い合せたときに、DBA_STREAMS_GLOBAL_RULES データ・ディクショナリ・ビューで行が戻される場合、その取得プロセスでは、サポートされていない変更と SYS および SYSTEM スキーマに対する変更を除き、データベース内でのすべての変更が取得されます。

関連項目： 取得プロセスで取得される変更と取得されない変更のタイプについては、2-6 ページの「取得されるデータ型」および 2-7 ページの「取得される変更のタイプ」を参照してください。

ルールに関する問題の解決

Streams の取得プロセス、伝播または適用プロセスが予期したとおりに動作していないのは、そのルール・セットに 1 つ以上のルールが欠落しているためであると判断した場合は、DBMS_STREAMS_ADM パッケージの次のいずれかのプロシージャを使用して適切なルールを追加できます。

- ADD_GLOBAL_PROPAGATION_RULES
- ADD_GLOBAL_RULES
- ADD_SCHEMA_PROPAGATION_RULES
- ADD_SCHEMA_RULES
- ADD_SUBSET_RULES
- ADD_TABLE_PROPAGATION_RULES
- ADD_TABLE_RULES

DBMS_RULE_ADM パッケージを使用すると、必要に応じてカスタマイズされたルールを追加できます。

また、Streams の取得プロセス、伝播または適用プロセスが予期したとおりに動作していないのは、1 つ以上のルールの変更やルール・セットからの削除が必要なためである可能性もあります。

適切なルールがあっても、関連するオブジェクトの変更が Streams の取得プロセス、伝播または適用プロセスによってフィルタで除外される場合は、トレース・ファイルとアラート・ログをチェックして、Streams のデータ・ディクショナリであるマルチバージョン・データ・ディクショナリの欠落に関する警告の有無を確認します。この種のメッセージには、次の情報が含まれている場合があります。

- gdbnm: 欠落しているオブジェクトのソース・データベースのグローバル名
- scn: 欠落しているトランザクションの SCN

この種のメッセージが見つかった場合は、新規の接続先データベース用にカスタム取得ルールを使用するか、既存の取得ルールを再利用して、インスタンス化の準備のために適切なプロシージャを実行してください。

- PREPARE_TABLE_INSTANTIATION
- PREPARE_SCHEMA_INSTANTIATION
- PREPARE_GLOBAL_INSTANTIATION

関連項目：

- 15-5 ページ「ルールの変更」
- 15-7 ページ「ルール・セットからのルールの削除」
- 12-10 ページ「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」
- Streams のデータ・ディクショナリの詳細は、2-20 ページの「取得プロセス作成中のデータ・ディクショナリの複製」を参照してください。

ルールベースの変換が適切に構成されているか

ルールベースの変換は、ルールが TRUE に評価される場合に発生するイベントの変更です。ルールベースの変換はルールのアクション・コンテキスト内で指定されます。また、これらのアクション・コンテキストに含まれる名前 / 値ペアでは、名前に STREAMS$_TRANSFORM_FUNCTION、値にユーザー作成プロシージャが指定されます。このユーザー作成プロシージャによって変換が実行されます。ユーザー定義プロシージャに不備があると、予期しない動作が発生する可能性があります。

Streams の取得プロセス、伝播または適用プロセスが予期したとおりに動作しない場合は、その取得プロセス、伝播または適用プロセスに指定したルールベースの変換プロシージャをチェックして不備を訂正します。これらのプロシージャの名前は、Streams の取得プロセス、伝播または適用プロセスで使用するルール・セットにあるルールのアクション・コンテキストを問い合せると確認できます。問題を解決するには、変換プロシージャの変更やルールベースの変換の削除が必要になる場合があります。アクション・コンテキスト内の名前 / 値ペアのうち、名前部分のスペルが正しいことを確認してください。

ルールベースの変換が実行される前に、ルールが評価されます。たとえば、表名を emps から employees に変更する変換がある場合は、その変換を使用する各ルールのルール条件で、表名 employees ではなく emps が指定されていることを確認してください。

関連項目：

- 取得プロセスで使用されるルール・セットを表示する問合せについては、17-4 ページの「各取得プロセスのキュー、ルール・セットおよび状態の表示」を参照してください。
- ルール・セット内のルールに指定されているルールベースの変換プロシージャを表示する問合せについては、17-47 ページの「ルール・セット内のルールベースの変換の表示」を参照してください。
- ルールベースの変換を変更または削除する方法については、15-11 ページの「ルールベースの変換の管理」を参照してください。

トレース・ファイルとアラート・ログでの問題のチェック

各取得プロセス、伝播ジョブおよび適用プロセスに関するメッセージは、そのプロセスまたは伝播ジョブを実行しているデータベースのトレース・ファイルに記録されます。取得プロセスはソース・データベース上で実行され、伝播ジョブは伝播のソース・キューを含むデータベース上で実行され、適用プロセスは接続先データベース上で実行されます。これらのトレース・ファイル・メッセージは、Streams 環境での問題を識別して解決する上で役立ちます。

バックグラウンド・プロセスのトレース・ファイルはすべて、初期化パラメータ BACKGROUND_DUMP_DEST で指定した接続先ディレクトリに書き込まれます。トレース・ファイル名はオペレーティング・システム固有ですが、通常、各ファイルにはそれを書き込むプロセスの名前が含まれています。

たとえば、一部のオペレーティング・システムでは、プロセスのトレース・ファイル名は *sid_xxxxx_iiiii*.trc となり、意味は次のようになります。

- *sid* はデータベースのシステム識別子です。
- *xxxxx* はプロセス名です。
- *iiiii* はオペレーティング・システムのプロセス番号です。

また、write_alert_log パラメータを、取得プロセスと適用プロセスの両方について y に設定できます。このパラメータをにデフォルト設定である y に設定すると、データベースのアラート・ログには取得プロセスまたは適用プロセスの停止理由を示すメッセージが書き込まれます。

トレース・ファイル内の情報を制御するには、DBMS_CAPTURE_ADM および DBMS_APPLY_ADM パッケージの SET_PARAMETER プロシージャを使用して、trace_file 取得プロセスまたは適用プロセス・パラメータを設定します。

次のチェックリストを使用して、Streams に関連するトレース・ファイルをチェックします。

- 取得プロセスのトレース・ファイルに取得の問題に関するメッセージが含まれているか
- 伝播ジョブに関連するトレース・ファイルに問題に関するメッセージが含まれているか
- 適用プロセスのトレース・ファイルに適用の問題に関するメッセージが含まれているか

関連項目：

- トレース・ファイルとアラート・ログおよびこれらの名前と位置の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。
- `trace_file` 取得プロセス・パラメータおよび `trace_file` 適用プロセス・パラメータの設定方法の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- トレース・ファイルの名前と位置の詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

取得プロセスのトレース・ファイルに取得の問題に関するメッセージが含まれているか

取得プロセスは、Oracle バックグラウンド・プロセス `cp`*nn* です。この場合、*nn* は取得プロセス番号です。たとえば、一部のオペレーティング・システムでは、取得プロセスを実行しているデータベースのシステム識別子が `hqdb` で、取得プロセス番号が `01` であれば、その取得プロセスのトレース・ファイル名は `hqdb_cp01` で始まります。

関連項目： 取得プロセスの取得プロセス番号を表示する問合せについては、17-4 ページの「単一の取得プロセスに関する一般情報の表示」を参照してください。

伝播ジョブに関連するトレース・ファイルに問題に関するメッセージが含まれているか

それぞれの伝播には、ジョブ・キュー・コーディネータ・プロセスとジョブ・キュー・プロセスに依存する伝播ジョブが使用されます。ジョブ・キュー・コーディネータ・プロセス名は `cjq`*nn* で、*nn* はジョブ・キュー・コーディネータ・プロセス番号です。ジョブ・キュー・プロセス名は `j`*nnn* で、*nnn* はジョブ・キュー・プロセス番号です。

たとえば、一部のオペレーティング・システムでは、伝播ジョブを実行しているデータベースのシステム識別子が `hqdb` で、ジョブ・キュー・コーディネータ・プロセスが `01` であれば、そのジョブ・キュー・コーディネータ・プロセスのトレース・ファイル名は `hqdb_cjq01` で始まります。同様に、前述のデータベース上で、ジョブ・キュー・プロセスが `001` であれば、そのトレース・ファイル名は `hqdb_j001` で始まります。プロセス名は、`DBA_QUEUE_SCHEDULES` データ・ディクショナリ・ビューの `PROCESS_NAME` 列を問い合せるとチェックできます。

関連項目： 伝播ジョブで使用されるジョブ・キュー・プロセスを表示する問合せについては、18-5 ページの「伝播で使用される伝播ジョブが有効化されているか」を参照してください。

適用プロセスのトレース・ファイルに適用の問題に関するメッセージが含まれているか

適用プロセスは、Oracle バックグラウンド・プロセス ap*nn* です。この場合、*nn* は適用プロセス番号です。たとえば、一部のオペレーティング・システムでは、適用プロセスを実行しているデータベースのシステム識別子が hqdb で、適用プロセス番号が 01 であれば、その適用プロセスのトレース・ファイル名は hqdb_ap01 で始まります。

また、適用プロセスではパラレル実行サーバーも使用されます。適用プロセスに関する情報は、1 つ以上のパラレル実行サーバーのトレース・ファイルに記録される場合があります。パラレル実行サーバーのプロセス名は p*nnn* で、*nnn* はプロセス番号です。そのため、一部のオペレーティング・システムでは、適用プロセスを実行しているデータベースのシステム識別子が hqdb で、プロセス番号が 001 であれば、その適用プロセスに関する情報を含むトレース・ファイルの名前は hqdb_p001 で始まります。

関連項目：

- 適用プロセスの適用プロセス番号を表示する問合せについては、17-29 ページの「コーディネータ・プロセスに関する情報の表示」を参照してください。
- 適用プロセスのリーダー・サーバーで使用されるパラレル実行サーバーを表示する問合せについては、17-27 ページの「適用プロセス用のリーダー・サーバーに関する情報の表示」を参照してください。
- 適用プロセスの適用サーバーで使用されるパラレル実行サーバーを表示する問合せについては、17-29 ページの「適用プロセス用の適用サーバーに関する情報の表示」を参照してください。

第 III 部

環境とアプリケーションの例

第 III 部には、次の詳細例が記載されています。

- 第 19 章「Streams メッセージングの例」
- 第 20 章「単一データベースの取得および適用の例」
- 第 21 章「簡単な単一のソース・レプリケーションの例」
- 第 22 章「単一ソースの異機種間レプリケーションの例」
- 第 23 章「複数のソース・レプリケーションの例」
- 第 24 章「ルールベースのアプリケーションの例」

19

Streams メッセージングの例

この章では、Streams を使用して構成できるメッセージ環境について説明します。

この章の内容は次のとおりです。

関連項目：

- 第 3 章「Streams のステージングと伝播」
- 第 13 章「ステージングと伝播の管理」
- 17-12 ページ「Streams キューの監視」

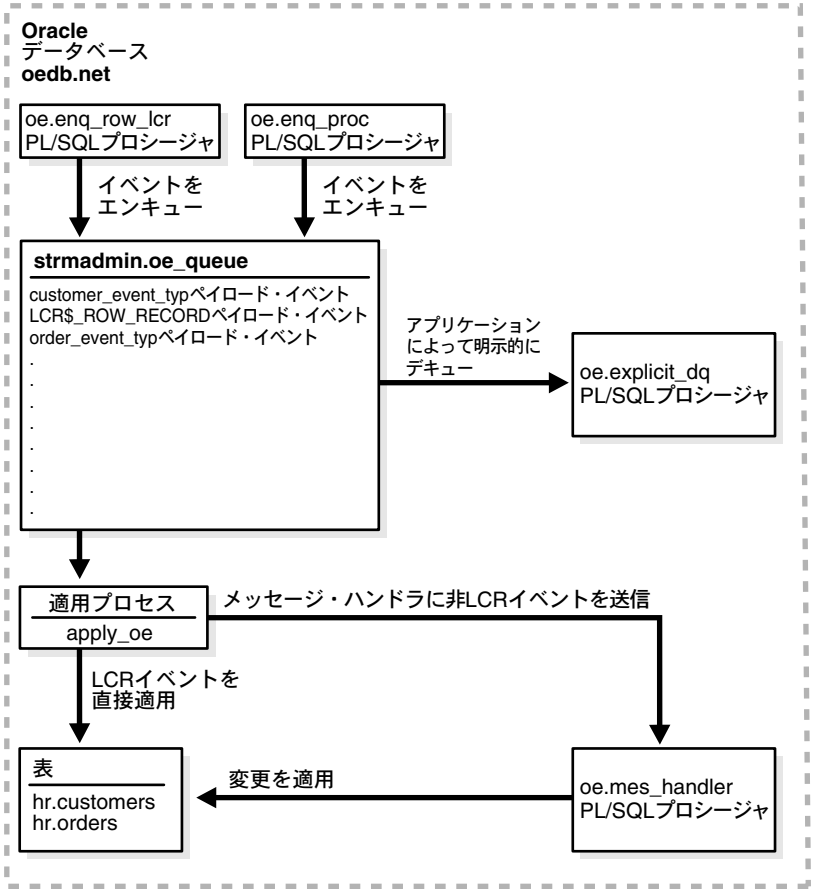
メッセージングの例の概要

この例では、データベース oedb.net にある単一の SYS.AnyData キューを使用して、様々な型のメッセージ・ペイロードを含むイベントが同じキューに格納される Streams メッセージ環境を作成する方法について説明します。特に、この例には、Streams の次のメッセージ機能を示します。

- 注文のペイロードと顧客のペイロードを含むメッセージを、SYS.Anydata イベントとしてキューにエンキューする機能
- 行 LCR のペイロードを含むメッセージを、SYS.Anydata イベントとしてキューにエンキューする機能
- イベントの適用に関するルール・セットを作成する機能
- ルール・セットで使用される評価コンテキストを作成する機能
- ルールに基づいてイベントをデキューして処理する Streams 適用プロセスを作成する機能
- メッセージ・ハンドラを作成して適用プロセスに関連付ける機能
- 適用プロセスを使用せず、ルールに基づいてイベントを明示的にデキューして処理する機能

図 19-1 に、この環境の概要を示します。

図 19-1 Streams メッセージ環境の例



前提条件

この項の例を開始する前に、前提条件となる次の作業を完了する必要があります。

- 環境内のすべてのデータベースについて、次の初期化パラメータを指定の値に設定します。
  - AQ_TM_PROCESSES: このパラメータでは、キュー・モニター・プロセスを設定します。値 1 ～ 10 では、メッセージの監視用に作成するキュー・モニター・プロセスの数を指定します。AQ_TM_PROCESSES を指定しないか、0 に設定すると、キュー・モニター・プロセスは作成されません。この例では、AQ_TM_PROCESSES を 1 以上の値に設定する必要があります。

    このパラメータを 1 以上に設定すると、指定した数のキュー・モニター・プロセスが起動します。これらのキュー・モニター・プロセスは、遅延や期限切れなど、時間ベースのメッセージ操作の管理、指定した保存期間を過ぎて保存されているメッセージのクリーン・アップ、および保存期間が 0（ゼロ）の場合のコンシューム済みメッセージのクリーン・アップを受け持ちます。
  - COMPATIBLE: このパラメータは 9.2.0 以上に設定する必要があります。
- これらのスクリプトを実行するクライアントから oedb.net データベースにアクセスできるように、ネットワークと Oracle Net を構成します。

  関連項目：『Oracle9*i* Net Services 管理者ガイド』
- この例では、新規ユーザーが Streams 管理者（strmadmin）として作成され、このユーザーのデータに使用する表領域の指定を求めるプロンプトが表示されます。この例を開始する前に、新規の表領域を作成するか、既存の表領域を Streams 管理者用に識別してください。SYSTEM 表領域は、Streams 管理者用にしないでください。

ユーザーの設定と Streams キューの作成

次の手順に従って、Streams メッセージ環境用のユーザーを設定し、Streams キューを作成します。

1. 出力の表示と結果のスプーリング
2. ユーザーの設定
3. Streams キューの作成
4. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 19-9 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、データベースに接続できるコンピュータ上で SQL*Plus を使用して実行してください。

```
/************************** BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_setup_message.out

/*
```

手順 2 ユーザーの設定

oedb.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@oedb.net AS SYSDBA

/*
```

この例では、oe サンプル・スキーマを使用します。この例を正常に機能させるには、oe ユーザーには DBMS_AQ パッケージ内のサブプログラムを実行するための権限が必要です。oe ユーザーは、手順 3 で Streams キューを作成するときにキュー・ユーザーとして指定します。SET_UP_QUEUE プロシージャを実行すると、oe ユーザーにキューに対する ENQUEUE

および DEQUEUE 権限が付与されますが、キューとの間でイベントのエンキューとデキューを行うために、oe ユーザーには DBMS_AQ パッケージの EXECUTE 権限も必要です。

また、この例に示す構成操作と管理操作のほとんどは、Streams 管理者が実行します。この手順では、Streams 管理者 strmadmin を作成し、このユーザーに必要な権限を付与します。これらの権限を付与すると、ユーザーは Streams に関連するパッケージ内のサブプログラムを実行し、ルール・セットおよびルールを作成し、データ・ディクショナリ・ビューを問い合せて Streams 環境を監視できます。このユーザー用に、異なる名前を選択できます。

注意：

- セキュリティを確保するために、Streams 管理者には strmadminpw 以外のパスワードを使用してください。
- Streams 管理者には、SELECT_CATALOG_ROLE は不要です。この例で付与されているのは、Streams 管理者が環境を簡単に監視できるようにするためです。
- Oracle Enterprise Manager の Streams ツールを使用する予定の場合は、Streams 管理者にこの手順で示す権限の他に SELECT ANY DICTIONARY 権限を付与します。
- ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

```
*/

GRANT EXECUTE ON DBMS_AQ TO oe;

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter the tablespace for the Streams administrator: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
                     QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM   TO strmadmin;
GRANT EXECUTE ON DBMS_AQ          TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM       TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM TO strmadmin;

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
```

```
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

/*
```

手順 3 Streams キューの作成

Streams 管理者として接続します。

```
*/

CONNECT strmadmin/strmadminpw@oedb.net

/*
```

SET_UP_QUEUE プロシージャを実行して、oedb.net でキュー oe_queue を作成します。このキューは、メッセージ環境で使用されるイベントを保持することで Streams キューとして機能します。

SET_UP_QUEUE プロシージャを実行すると、次のアクションが実行されます。

- キュー表 oe_queue_table が作成されます。このキュー表の所有者は Streams 管理者（strmadmin）であり、このユーザーのデフォルト記憶域が使用されます。
- Streams 管理者（strmadmin）が所有するキュー oe_queue が作成されます。
- キューが起動されます。

```
*/

BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table => 'oe_queue_table',
    queue_name  => 'oe_queue');
END;
/

/*
```

手順 4 oe ユーザーへのキューに対する権限の付与

```
*/

BEGIN
  SYS.DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(
    privilege  => 'ALL',
    queue_name => 'strmadmin.oe_queue',
    grantee    => 'oe');
END;
/

/*
```

手順 5 明示的エンキュー用のエージェントの作成

oe_queue キューで明示的なエンキュー操作を実行するためのエージェントを作成します。

```
*/

BEGIN
  SYS.DBMS_AQADM.CREATE_AQ_AGENT(
    agent_name  => 'explicit_enq');
END;
/

/*
```

手順 6 oe ユーザーと explicit_enq エージェントの関連付け

ユーザーが保護キューについてエンキューやデキューなどのキュー操作を実行できるように、そのユーザーをキューの保護キュー・ユーザーとして構成する必要があります。oe_queue キューは、SET_UP_QUEUE を使用して作成されたため、保護キューです。この手順を実行すると、oe ユーザーはこのキューでのエンキュー操作が実行できるようになります。

```
*/

BEGIN
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name  => 'explicit_enq',
    db_username => 'oe');
END;
/

/*
```

手順 7 スプール結果のチェック

このスクリプトの完了後に streams_setup_message.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT *****************************/
```

エンキュー・プロシージャの作成

次の手順に従って、非 LCR イベントを Streams キューにエンキューする PL/SQL プロシージャ 1 つと、行 LCR イベントを Streams キューにエンキューする PL/SQL プロシージャを 1 つ作成します。

1. 出力の表示と結果のスプーリング
2. 注文を表す型の作成
3. 顧客を表す型の作成
4. 非 LCR イベントをエンキューするプロシージャの作成
5. 行 LCR イベントを構成してエンキューするプロシージャの作成
6. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 19-13 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、データベースに接続できるコンピュータ上で SQL*Plus を使用して実行してください。

```
/************************ BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_enqprocs_message.out

/*
```

手順 2 注文を表す型の作成

oe として接続します。

```
*/

CONNECT oe/oe@oedb.net

/*
```

oe.orders 表の列に基づいて注文を表す型を作成します。型属性には、oe.orders 表の列と、追加の属性 action が 1 つ含まれます。この型のインスタンスの action 属性の値を使用して、インスタンス上で実行される適切なアクション（適用プロセスによるデキューまたは明示的なデキュー）が判断されます。この型は、Streams キューにエンキューされるイベントに使用されます。

```
*/

CREATE OR REPLACE TYPE order_event_typ AS OBJECT (
  order_id       NUMBER(12),
  order_date     TIMESTAMP(6) WITH LOCAL TIME ZONE,
  order_mode     VARCHAR2(8),
  customer_id    NUMBER(6),
  order_status   NUMBER(2),
  order_total    NUMBER(8,2),
  sales_rep_id   NUMBER(6),
  promotion_id   NUMBER(6),
  action         VARCHAR(7));
/

/*
```

手順 3 顧客を表す型の作成

oe.customers 表の列に基づいて顧客を表す型を作成します。型属性には、oe.customers 表の列と、追加の属性 action が 1 つ含まれます。この型のインスタンスの action 属性の値を使用して、インスタンス上で実行される適切なアクション（適用プロセスによるデキューまたは明示的なデキュー）が判断されます。この型は、Streams キューにエンキューされるイベントに使用されます。

```
*/

CREATE OR REPLACE TYPE customer_event_typ AS OBJECT (
  customer_id         NUMBER(6),
  cust_first_name     VARCHAR2(20),
  cust_last_name      VARCHAR2(20),
  cust_address        CUST_ADDRESS_TYP,
  phone_numbers       PHONE_LIST_TYP,
  nls_language        VARCHAR2(3),
  nls_territory       VARCHAR2(30),
  credit_limit        NUMBER(9,2),
  cust_email          VARCHAR2(30),
  account_mgr_id      NUMBER(6),
  cust_geo_location   MDSYS.SDO_GEOMETRY,
  action              VARCHAR(7));
/

/*
```

手順 4 非 LCR イベントをエンキューするプロシージャの作成

イベントを Streams キューにエンキューする PL/SQL プロシージャ enq_proc を作成します。

> **注意：** エンキューされた 1 つのメッセージを適用プロセスと明示的なデキューでデキューできますが、この例はその機能を示していません。

```
*/

CREATE OR REPLACE PROCEDURE oe.enq_proc (event IN SYS.Anydata) IS
    enqopt       DBMS_AQ.ENQUEUE_OPTIONS_T;
    mprop        DBMS_AQ.MESSAGE_PROPERTIES_T;
    enq_eventid  RAW(16);
  BEGIN
    mprop.SENDER_ID := SYS.AQ$_AGENT('explicit_enq', NULL, NULL);
    DBMS_AQ.ENQUEUE(
      queue_name          =>  'strmadmin.oe_queue',
      enqueue_options     =>  enqopt,
      message_properties  =>  mprop,
      payload             =>  event,
      msgid               =>  enq_eventid);
END;
/

/*
```

手順 5 行 LCR イベントを構成してエンキューするプロシージャの作成

行 LCR を構成し、それをキューにエンキューするプロシージャ enq_row_lcr を作成します。

> **関連項目：** 『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』で LCR のコンストラクタの詳細を参照してください。

```
*/

CREATE OR REPLACE PROCEDURE oe.enq_row_lcr(
                 source_dbname  VARCHAR2,
                 cmd_type       VARCHAR2,
                 obj_owner      VARCHAR2,
                 obj_name       VARCHAR2,
                 old_vals       SYS.LCR$_ROW_LIST,
                 new_vals       SYS.LCR$_ROW_LIST) AS
  eopt           DBMS_AQ.ENQUEUE_OPTIONS_T;
  mprop          DBMS_AQ.MESSAGE_PROPERTIES_T;
  enq_msgid      RAW(16);
  row_lcr        SYS.LCR$_ROW_RECORD;
BEGIN
  mprop.SENDER_ID := SYS.AQ$_AGENT('explicit_enq', NULL, NULL);
  -- Construct the LCR based on information passed to procedure
  row_lcr := SYS.LCR$_ROW_RECORD.CONSTRUCT(
    source_database_name  =>  source_dbname,
    command_type          =>  cmd_type,
    object_owner          =>  obj_owner,
    object_name           =>  obj_name,
    old_values            =>  old_vals,
    new_values            =>  new_vals);
  -- Enqueue the created row LCR
  DBMS_AQ.ENQUEUE(
    queue_name         =>  'strmadmin.oe_queue',
    enqueue_options    =>  eopt,
    message_properties =>  mprop,
    payload            =>  SYS.AnyData.ConvertObject(row_lcr),
    msgid              =>  enq_msgid);
END enq_row_lcr;
/

/*
```

手順 6 スプール結果のチェック

このスクリプトの完了後に `streams_enqprocs_message.out` スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT *****************************/
```

適用プロセスの構成

次の手順に従って、Streams キューにあるユーザー・エンキュー・イベントを適用するように適用プロセスを構成します。

1. 出力の表示と結果のスプーリング
2. action 属性の値を判断するファンクションの作成
3. メッセージ・ハンドラの作成
4. strmadmin へのプロシージャの EXECUTE 権限の付与
5. ルール・セットの評価コンテキストの作成
6. 適用プロセスのルール・セットの作成
7. event の action が apply の場合に TRUE と評価されるルールの作成
8. 行 LCR イベントについて TRUE と評価されるルールの作成
9. ルール・セットへのルールの追加
10. 適用プロセスの作成
11. oe ユーザーに対するルール・セットの EXECUTE 権限の付与
12. 適用プロセスの起動
13. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 19-21 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、データベースに接続できるコンピュータ上で SQL*Plus を使用して実行してください。

```
/************************* BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_apply_message.out

/*
```

手順 2 action 属性の値を判断するファンクションの作成

oe として接続します。

```
*/

CONNECT oe/oe@oedb.net

/*
```

キューにあるイベントの action 属性の値を判断するファンクション get_oe_action を作成します。このファンクションを後でこの例のルールに使用して、イベントの action 属性の値を判断します。ルール・エンジンのクライアントでは、イベントの適切なアクション（適用プロセスによるデキューまたは明示的なデキュー）が実行されます。この例では、ルール・エンジンのクライアントは、適用プロセスと oe.explicit_dq PL/SQL プロシージャです。

```
*/

CREATE OR REPLACE FUNCTION oe.get_oe_action (event IN SYS.Anydata)
RETURN VARCHAR2
IS
  ord         oe.order_event_typ;
  cust        oe.customer_event_typ;
  num           NUMBER;
  type_name     VARCHAR2(61);
BEGIN
  type_name := event.GETTYPENAME;
  IF type_name = 'OE.ORDER_EVENT_TYP' THEN
    num := event.GETOBJECT(ord);
    RETURN ord.action;
  ELSIF type_name = 'OE.CUSTOMER_EVENT_TYP' THEN
    num := event.GETOBJECT(cust);
    RETURN cust.action;
  ELSE
    RETURN NULL;
  END IF;
END;
/

/*
```

手順 3 メッセージ・ハンドラの作成

適用プロセスで使用されるメッセージ・ハンドラ mes_handler を作成します。このプロシージャは、oe.order_event_typ 型または oe.customer_event_typ 型のユーザー・エンキュー・イベントのペイロードを取り、それぞれ oe.orders 表および oe.customers 表に 1 行として挿入します。

```
*/

CREATE OR REPLACE PROCEDURE oe.mes_handler (event SYS.AnyData)
IS
  ord           oe.order_event_typ;
  cust          oe.customer_event_typ;
  num           NUMBER;
  type_name     VARCHAR2(61);
BEGIN
  type_name := event.GETTYPENAME;
  IF type_name = 'OE.ORDER_EVENT_TYP' THEN
    num := event.GETOBJECT(ord);
    INSERT INTO oe.orders VALUES (ord.order_id, ord.order_date,
      ord.order_mode, ord.customer_id, ord.order_status, ord.order_total,
      ord.sales_rep_id, ord.promotion_id);
  ELSIF type_name = 'OE.CUSTOMER_EVENT_TYP' THEN
    num := event.GETOBJECT(cust);
    INSERT INTO oe.customers VALUES (cust.customer_id, cust.cust_first_name,
      cust.cust_last_name, cust.cust_address, cust.phone_numbers,
      cust.nls_language, cust.nls_territory, cust.credit_limit, cust.cust_email,
      cust.account_mgr_id, cust.cust_geo_location);
  END IF;
END;
/

/*
```

手順 4 strmadmin へのプロシージャの EXECUTE 権限の付与

```
*/

GRANT EXECUTE ON get_oe_action TO strmadmin;

GRANT EXECUTE ON mes_handler TO strmadmin;

/*
```

手順 5 ルール・セットの評価コンテキストの作成

Streams 管理者として接続します。

```
*/

CONNECT strmadmin/strmadminpw@oedb.net

/*
```

ルール・セットの評価コンテキストを作成します。この例では、表の別名は tab ですが、必要な場合は異なる別名を使用できます。

```
*/

DECLARE
    table_alias     SYS.RE$TABLE_ALIAS_LIST;
  BEGIN
    table_alias := SYS.RE$TABLE_ALIAS_LIST(SYS.RE$TABLE_ALIAS(
                                           'tab',
                                           'strmadmin.oe_queue_table'));
    DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
      evaluation_context_name  =>  'oe_eval_context',
      table_aliases            =>  table_alias);
END;
/

/*
```

手順 6 適用プロセスのルール・セットの作成

適用プロセスのルール・セットを作成します。

```
*/

BEGIN
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name       =>  'apply_oe_rs',
    evaluation_context  =>  'strmadmin.oe_eval_context');
END;
/

/*
```

手順 7 event の action が apply の場合に TRUE と評価されるルールの作成

event の action 値が apply の場合に TRUE と評価されるルールを作成します。oe.get_oe_action ファンクションに tab.user_data が渡されることに注意してください。tab.user_data 列には、キュー表内のイベント・ペイロードが保持されます。キュー表の表の別名は、19-17 ページの手順 5 で tab として指定しています。

```
*/

BEGIN
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name   => 'strmadmin.apply_action',
    condition   => ' oe.get_oe_action(tab.user_data) = ''APPLY'' ');
END;
/

/*
```

手順 8 行 LCR イベントについて TRUE と評価されるルールの作成

キュー内のイベントが oe.orders 表または oe.customers 表を変更する行 LCR の場合に TRUE と評価されるルールを作成します。このルールにより、適用プロセスは表にユーザーによってエンキューされた変更を直接適用できます。このルールは LCR の評価に使用するため、便宜上、このルールでは Oracle が提供する評価コンテキスト SYS.STREAMS$_EVALUATION_CONTEXT を使用しています。このルールをルール・セットに追加すると、評価中には、ルール・セットの評価コンテキストのかわりに、この評価コンテキストが使用されます。

```
*/

BEGIN
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name           =>  'apply_lcrs',
    condition           =>  ':dml.GET_OBJECT_OWNER() = ''OE'' AND ' ||
                            ' (:dml.GET_OBJECT_NAME() = ''ORDERS'' OR ' ||
                            ':dml.GET_OBJECT_NAME() = ''CUSTOMERS'') ',
    evaluation_context  =>  'SYS.STREAMS$_EVALUATION_CONTEXT');
END;
/

/*
```

手順 9 ルール・セットへのルールの追加

手順 7 と手順 8 で作成したルールを、19-17 ページの手順 6 で作成したルール・セットに追加します。

```
*/

BEGIN
  DBMS_RULE_ADM.ADD_RULE(
    rule_name          =>  'apply_action',
    rule_set_name      =>  'apply_oe_rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name          =>  'apply_lcrs',
    rule_set_name      =>  'apply_oe_rs');
END;
/

/*
```

手順 10 適用プロセスの作成

oe_queue に関連付けられた適用プロセスを作成します。この適用プロセスでは、apply_oe_rs ルール・セットを使用し、メッセージ・ハンドラとして mes_handler プロシージャを使用します。

```
*/

BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name       =>  'strmadmin.oe_queue',
    apply_name       =>  'apply_oe',
    rule_set_name    =>  'strmadmin.apply_oe_rs',
    message_handler  =>  'oe.mes_handler',
    apply_user       =>  'oe',
    apply_captured   =>  false);
END;
/

/*
```

手順 11 oe ユーザーに対するルール・セットの EXECUTE 権限の付与

strmadmin.apply_oe_rs ルール・セットの EXECUTE 権限を付与します。手順 10 で適用プロセスを作成するときに、oe を適用ユーザーとして指定したため、oe には適用プロセスで使用されるルール・セットの実行権限が必要です。

```
*/

BEGIN
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name  => 'strmadmin.apply_oe_rs',
    grantee      => 'oe',
    grant_option => FALSE);
END;
/

/*
```

手順 12 適用プロセスの起動

エラーが発生しても適用プロセスが無効化されないように、disable_on_error パラメータを n に設定して、oedb.net で適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_oe',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  =>  'apply_oe');
END;
/

/*
```

手順 13 スプール結果のチェック

このスクリプトの完了後に streams_apply_message.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT *****************************/
```

明示的なデキューの構成

次の手順に従い、メッセージの内容に基づいてメッセージの明示的なデキューを構成します。

1. 出力の表示と結果のスプーリング
2. 明示的デキュー用のエージェントの作成
3. oe ユーザーと explicit_dq エージェントの関連付け
4. oe_queue キューへのサブスクライバの追加
5. イベントを明示的にデキューするプロシージャの作成
6. スプール結果のチェック

> **注意：** このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 19-25 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、データベースに接続できるコンピュータ上で SQL*Plus を使用して実行してください。

```
/************************ BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_explicit_dq.out

/*
```

手順 2 明示的デキュー用のエージェントの作成

Streams 管理者として接続します。

```
*/

CONNECT strmadmin/strmadminpw@oedb.net

/*
```

oe_queue キューで明示的なデキュー操作を実行するためのエージェントを作成します。

```
*/

BEGIN
  SYS.DBMS_AQADM.CREATE_AQ_AGENT(
    agent_name  => 'explicit_dq');
END;
/

/*
```

手順 3 oe ユーザーと explicit_dq エージェントの関連付け

ユーザーが保護キューについてエンキューやデキューなどのキュー操作を実行できるように、そのユーザーをキューの保護キュー・ユーザーとして構成する必要があります。oe_queue キューは、SET_UP_QUEUE を使用して作成されたため、保護キューです。次の手順でエージェントを使用してキューへのサブスクライバが作成されると、oe ユーザーはこのキューに対してデキュー操作を実行できるようになります。

```
*/

BEGIN
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name  => 'explicit_dq',
    db_username => 'oe');
END;
/

/*
```

手順 4 oe_queue キューへのサブスクライバの追加

oe_queue キューにサブスクライバを追加します。このサブスクライバによって、イベントの明示的なデキューが実行されます。サブスクライバ・ルールを使用して、action 値が apply でないイベントがデキューされます。イベントの action 値が apply の場合、そのイベントはサブスクライバに無視されます。この種のイベントは、適用プロセスによってデキューされて処理されます。

```
*/

DECLARE
  subscriber SYS.AQ$_AGENT;
BEGIN
  subscriber :=  SYS.AQ$_AGENT('explicit_dq', NULL, NULL);
  SYS.DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name  =>  'strmadmin.oe_queue',
    subscriber  =>  subscriber,
    rule        =>  'oe.get_oe_action(tab.user_data) != ''APPLY''');
END;
/

/*
```

手順 5 イベントを明示的にデキューするプロシージャの作成

oe として接続します。

```
*/

CONNECT oe/oe@oedb.net

/*
```

19-23 ページの手順 4 で作成したサブスクライバを使用してイベントを明示的にデキューする、PL/SQL プロシージャ explicit_dq を作成します。

注意：

- このプロシージャでは、イベントがデキュー後にコミットされます。コミットによって、デキューされたメッセージがこのサブスクライバで正常にコンシュームされたことがキューに通知されます。
- このプロシージャでは、複数のトランザクションを処理し、2 つの例外ハンドラを使用できます。それ以上メッセージがなくなって第 2 の例外ハンドラ no_messages がループを出ると、最初の例外ハンドラ next_trans が次のトランザクションに移動します。

```
*/

CREATE OR REPLACE PROCEDURE oe.explicit_dq (consumer IN VARCHAR2) AS
  deqopt       DBMS_AQ.DEQUEUE_OPTIONS_T;
  mprop        DBMS_AQ.MESSAGE_PROPERTIES_T;
  msgid        RAW(16);
  payload      SYS.AnyData;
  new_messages BOOLEAN := TRUE;
  ord          oe.order_event_typ;
  cust         oe.customer_event_typ;
  tc           pls_integer;
  next_trans   EXCEPTION;
  no_messages  EXCEPTION;
  pragma exception_init (next_trans, -25235);
  pragma exception_init (no_messages, -25228);
BEGIN
  deqopt.consumer_name := consumer;
  deqopt.wait := 1;
  WHILE (new_messages) LOOP
    BEGIN
    DBMS_AQ.DEQUEUE(
      queue_name          =>  'strmadmin.oe_queue',
      dequeue_options     =>  deqopt,
      message_properties  =>  mprop,
      payload             =>  payload,
      msgid               =>  msgid);
    COMMIT;
    deqopt.navigation := DBMS_AQ.NEXT;
    DBMS_OUTPUT.PUT_LINE('Event Dequeued');
    DBMS_OUTPUT.PUT_LINE('Type Name := ' || payload.GetTypeName);
    IF (payload.GetTypeName = 'OE.ORDER_EVENT_TYP') THEN
      tc := payload.GetObject(ord);
      DBMS_OUTPUT.PUT_LINE('order_id     - ' || ord.order_id);
      DBMS_OUTPUT.PUT_LINE('order_date   - ' || ord.order_date);
      DBMS_OUTPUT.PUT_LINE('order_mode   - ' || ord.order_mode);
      DBMS_OUTPUT.PUT_LINE('customer_id  - ' || ord.customer_id);
      DBMS_OUTPUT.PUT_LINE('order_status - ' || ord.order_status);
      DBMS_OUTPUT.PUT_LINE('order_total  - ' || ord.order_total);
      DBMS_OUTPUT.PUT_LINE('sales_rep_id - ' || ord.sales_rep_id);
      DBMS_OUTPUT.PUT_LINE('promotion_id - ' || ord.promotion_id);
    END IF;
    IF (payload.GetTypeName = 'OE.CUSTOMER_EVENT_TYP') THEN
      tc := payload.GetObject(cust);
      DBMS_OUTPUT.PUT_LINE('customer_id     - ' || cust.customer_id);
      DBMS_OUTPUT.PUT_LINE('cust_first_name - ' || cust.cust_first_name);
      DBMS_OUTPUT.PUT_LINE('cust_last_name  - ' || cust.cust_last_name);
      DBMS_OUTPUT.PUT_LINE('street_address  - ' ||
```

```
                            cust.cust_address.street_address);
      DBMS_OUTPUT.PUT_LINE('postal_code     - ' ||
                            cust.cust_address.postal_code);
      DBMS_OUTPUT.PUT_LINE('city            - ' || cust.cust_address.city);
      DBMS_OUTPUT.PUT_LINE('state_province  - ' ||
                            cust.cust_address.state_province);
      DBMS_OUTPUT.PUT_LINE('country_id      - ' ||
                            cust.cust_address.country_id);
      DBMS_OUTPUT.PUT_LINE('phone_number1   - ' || cust.phone_numbers(1));
      DBMS_OUTPUT.PUT_LINE('phone_number2   - ' || cust.phone_numbers(2));
      DBMS_OUTPUT.PUT_LINE('phone_number3   - ' || cust.phone_numbers(3));
      DBMS_OUTPUT.PUT_LINE('nls_language    - ' || cust.nls_language);
      DBMS_OUTPUT.PUT_LINE('nls_territory   - ' || cust.nls_territory);
      DBMS_OUTPUT.PUT_LINE('credit_limit    - ' || cust.credit_limit);
      DBMS_OUTPUT.PUT_LINE('cust_email      - ' || cust.cust_email);
      DBMS_OUTPUT.PUT_LINE('account_mgr_id  - ' || cust.account_mgr_id);
    END IF;
    EXCEPTION
      WHEN next_trans THEN
      deqopt.navigation := DBMS_AQ.NEXT_TRANSACTION;
      WHEN no_messages THEN
        new_messages  := FALSE;
        DBMS_OUTPUT.PUT_LINE('No more events');
     END;
  END LOOP;
END;
/

/*
```

手順 6 スプール結果のチェック

このスクリプトの完了後に streams_explicit_dq.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT *****************************/
```

イベントのエンキュー

次の手順に従って、非 LCR イベントと行 LCR イベントをキューにエンキューします。

1. 出力の表示と結果のスプーリング
2. 適用プロセスによってデキューされる非 LCR イベントのエンキュー
3. 明示的にデキューされる非 LCR イベントのエンキュー
4. 適用プロセスによってデキューされる行 LCR イベントのエンキュー
5. スプール結果のチェック

注意 :

- ユーザー・エンキュー LCR は明示的にデキューできますが、この例はその機能を示していません。
- このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 19-31 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、データベースに接続できるコンピュータ上で SQL*Plus を使用して実行してください。

```
/************************ BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_enq_deq.out

/*
```

手順 2 適用プロセスによってデキューされる非 LCR イベントのエンキュー

oe として接続します。

```
*/

CONNECT oe/oe@oedb.net

/*
```

action の値が apply のイベントをエンキューします。適用プロセスでは、適用プロセスのルールに基づき、19-16 ページの「メッセージ・ハンドラの作成」で作成したメッセージ・ハンドラ・プロシージャ oe.mes_handler を使用して、これらのイベントがデキューされて処理されます。エンキュー後の COMMIT によって、この 2 つのエンキューが同じトランザクションの一部になります。エンキューされたメッセージは、それをエンキューしたセッションでエンキューがコミットされるまで参照できません。

```
*/

BEGIN
  oe.enq_proc(SYS.AnyData.convertobject(oe.order_event_typ(
    2500,'05-MAY-01','online',117,3,44699,161,NULL,'APPLY')));
END;
/

BEGIN
  oe.enq_proc(SYS.AnyData.convertobject(oe.customer_event_typ(
    990,'Hester','Prynne',oe.cust_address_typ('555 Beacon Street','Boston',
    'MA',02109,'US'),oe.phone_list_typ('+1 617 123 4104', '+1 617 083 4381',
    '+1 617 742 5813'),'i','AMERICA',5000,'a@scarlet_letter.com',145,
    NULL,'APPLY')));
END;
/

COMMIT;

/*
```

手順 3 明示的にデキューされる非 LCR イベントのエンキュー

action の値が dequeue のイベントをエンキューします。これらのイベントは action が apply でないため、19-23 ページの「イベントを明示的にデキューするプロシージャの作成」で作成した oe.explicit_dq プロシージャによってデキューされます。適用プロセスでは、適用プロセス・ルールに基づいてこれらのイベントが無視されます。エンキュー後の COMMIT によって、この 2 つのエンキューが同じトランザクションの一部になります。

```
*/

BEGIN
  oe.enq_proc(SYS.AnyData.convertobject(oe.order_event_typ(
    2501,'22-JAN-00','direct',117,3,22788,161,NULL,'DEQUEUE')));
END;
/

BEGIN
  oe.enq_proc(SYS.AnyData.convertobject(oe.customer_event_typ(
    991,'Nick','Carraway',oe.cust_address_typ('10th Street',
    11101,'Long Island','NY','US'),oe.phone_list_typ('+1 718 786 2287',
    '+1 718 511 9114', '+1 718 888 4832'),'i','AMERICA',3000,
    'nick@great_gatsby.com',149,NULL,'DEQUEUE')));
END;
/

COMMIT;

/*
```

手順 4 適用プロセスによってデキューされる行 LCR イベントのエンキュー

行 LCR イベントをエンキューします。適用プロセスでは、これらのイベントが直接適用されます。エンキューされた LCR は、トランザクション境界でコミットされる必要があります。この手順では、各エンキューの後に COMMIT 文が実行され、各エンキューが別個のトランザクションとなります。ただし、1 つのトランザクションに複数の LCR がある場合は、コミット前に複数の LCR のエンキューを実行できます。

oe.orders 表に 1 行を挿入する行 LCR を作成します。

```
*/

DECLARE
  newunit1  SYS.LCR$_ROW_UNIT;
  newunit2  SYS.LCR$_ROW_UNIT;
  newunit3  SYS.LCR$_ROW_UNIT;
  newunit4  SYS.LCR$_ROW_UNIT;
  newunit5  SYS.LCR$_ROW_UNIT;
  newunit6  SYS.LCR$_ROW_UNIT;
  newunit7  SYS.LCR$_ROW_UNIT;
```

```
  newunit8  SYS.LCR$_ROW_UNIT;
  newvals   SYS.LCR$_ROW_LIST;
BEGIN
  newunit1 := SYS.LCR$_ROW_UNIT(
    'ORDER_ID',
    SYS.AnyData.ConvertNumber(2502),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit2 := SYS.LCR$_ROW_UNIT(
    'ORDER_DATE',
    SYS.AnyData.ConvertTimestampLTZ('04-NOV-00'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit3 := SYS.LCR$_ROW_UNIT(
    'ORDER_MODE',
    SYS.AnyData.ConvertVarchar2('online'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit4 := SYS.LCR$_ROW_UNIT(
    'CUSTOMER_ID',
    SYS.AnyData.ConvertNumber(145),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit5 := SYS.LCR$_ROW_UNIT(
    'ORDER_STATUS',
    SYS.AnyData.ConvertNumber(3),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit6 := SYS.LCR$_ROW_UNIT(
    'ORDER_TOTAL',
    SYS.AnyData.ConvertNumber(35199),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit7 := SYS.LCR$_ROW_UNIT(
    'SALES_REP_ID',
    SYS.AnyData.ConvertNumber(160),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
```

```
  newunit8 := SYS.LCR$_ROW_UNIT(
    'PROMOTION_ID',
    SYS.AnyData.ConvertNumber(1),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1,newunit2,newunit3,newunit4,
                               newunit5,newunit6,newunit7,newunit8);
oe.enq_row_lcr(
  source_dbname  =>  'OEDB.NET',
  cmd_type       =>  'INSERT',
  obj_owner      =>  'OE',
  obj_name       =>  'ORDERS',
  old_vals       =>  NULL,
  new_vals       =>  newvals);
END;
/
COMMIT;

/*
```

oe.orders 表に挿入した 1 行を更新する行 LCR を作成します。

```
*/

DECLARE
  oldunit1  SYS.LCR$_ROW_UNIT;
  oldunit2  SYS.LCR$_ROW_UNIT;
  oldvals   SYS.LCR$_ROW_LIST;
  newunit1  SYS.LCR$_ROW_UNIT;
  newvals   SYS.LCR$_ROW_LIST;
BEGIN
  oldunit1 := SYS.LCR$_ROW_UNIT(
    'ORDER_ID',
    SYS.AnyData.ConvertNumber(2502),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldunit2 := SYS.LCR$_ROW_UNIT(
    'ORDER_TOTAL',
    SYS.AnyData.ConvertNumber(35199),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldvals := SYS.LCR$_ROW_LIST(oldunit1,oldunit2);
```

```
  newunit1 := SYS.LCR$_ROW_UNIT(
    'ORDER_TOTAL',
    SYS.AnyData.ConvertNumber(5235),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1);
oe.enq_row_lcr(
  source_dbname  =>  'OEDB.NET',
  cmd_type       =>  'UPDATE',
  obj_owner      =>  'OE',
  obj_name       =>  'ORDERS',
  old_vals       =>  oldvals,
  new_vals       =>  newvals);
END;
/
COMMIT;

/*
```

手順 5 スプール結果のチェック

このスクリプトの完了後に streams_enq_deq.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT ******************************/
```

イベントの明示的なデキューと適用済みイベントの問合せ

次の手順に従って、イベントを明示的にデキューして、適用プロセスで適用済みのイベントを問い合せます。これらのイベントは、19-26 ページの「イベントのエンキュー」でエンキューされています。

手順 1 イベントを明示的にデキューするプロシージャの実行

19-23 ページの「イベントを明示的にデキューするプロシージャの作成」で作成したプロシージャを実行し、デキューするイベントのコンシューマを指定します。この場合、コンシューマは 19-23 ページの「oe_queue キューへのサブスクライバの追加」で追加したサブスクライバです。この例では、このプロシージャで明示的にデキューされないイベントは、適用プロセスでデキューされます。

```
CONNECT oe/oe@oedb.net

SET SERVEROUTPUT ON SIZE 100000

EXEC oe.explicit_dq('explicit_dq');
```

19-28 ページの「明示的にデキューされる非 LCR イベントのエンキュー」でエンキューされた非 LCR イベントを確認する必要があります。

手順 2 適用済みイベントの問合せ

`oe.orders` および `oe.customers` 表を問い合せて、適用プロセスによって適用されたイベントに対応する行を表示します。

```
SELECT * FROM oe.orders WHERE order_id = 2500;

SELECT cust_first_name, cust_last_name, cust_email
  FROM oe.customers WHERE customer_id = 990;

SELECT * FROM oe.orders WHERE order_id = 2502;
```

19-27 ページの「適用プロセスによってデキューされる非 LCR イベントのエンキュー」でエンキューされた非 LCR イベント、および 19-28 ページの「適用プロセスによってデキューされる行 LCR イベントのエンキュー」でエンキューされた行 LCR を確認する必要があります。

JMS を使用したイベントのエンキューとデキュー

この例では、JMS を使用して非 LCR イベントと行 LCR イベントをキューにエンキューします。次に、JMS を使用して、これらのイベントをキューからデキューします。

この操作の手順は次のとおりです。

1. catxlcr.sql スクリプトの実行
2. ユーザー・イベントの型の作成
3. CLASSPATH の設定
4. Oracle のオブジェクト型にマップする Java クラスの作成
5. メッセージをエンキューするための Java コードの作成
6. メッセージをデキューするための Java コードの作成
7. スクリプトのコンパイル
8. エンキュー・プログラムの実行
9. デキュー・プログラムの実行

手順 1 catxlcr.sql スクリプトの実行

この例を正常に完了するには、Oracle ホームの `rdbms/admin/` ディレクトリにある `catxlcr.sql` スクリプトを使用して、LCR スキーマを `SYS` スキーマにロードする必要があります。まだ実行していない場合は、このスクリプトを実行してください。

たとえば、Oracle ホーム・ディレクトリが `/usr/oracle` の場合は、次のように入力してスクリプトを実行します。

```
CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA

@/usr/oracle/rdbms/admin/catxlcr.sql
```

手順 2 ユーザー・イベントの型の作成

```
CONNECT oe/oe

CREATE TYPE address AS OBJECT (street VARCHAR (30), num NUMBER)
/

CREATE TYPE person AS OBJECT (name VARCHAR (30), home ADDRESS)
/
```

手順 3 CLASSPATH の設定

使用している JDK のリリースに基づいて、次の jar ファイルと zip ファイルを CLASSPATH に含める必要があります。

また、LD_LIBRARY_PATH（Solaris）または PATH（Windows NT）の $ORACLE_HOME/lib が設定されていることを確認してください。

```
-- For JDK1.3.x
$ORACLE_HOME/jdbc/lib/classes12.zip
$ORACLE_HOME/rdbms/jlib/aqapi13.jar
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/xdb.jar
$ORACLE_HOME/xdk/lib/xmlparserv2.jar
$ORACLE_HOME/jlib/jndi.jar

-- For JDK1.2.x
$ORACLE_HOME/jdbc/lib/classes12.zip
$ORACLE_HOME/rdbms/jlib/aqapi12.jar
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/xdb.jar
$ORACLE_HOME/xdk/lib/xmlparserv2.jar
$ORACLE_HOME/jlib/jndi.jar

-- For JDK1.1.x
$ORACLE_HOME/jdbc/lib/classes111.zip
$ORACLE_HOME/rdbms/jlib/aqapi11.jar
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/xdb.jar
$ORACLE_HOME/xdk/lib/xmlparserv2.jar
$ORACLE_HOME/jlib/jndi.jar
```

手順 4 Oracle のオブジェクト型にマップする Java クラスの作成

最初に、次の 2 行を含むファイル input.typ を作成します。

```
SQL PERSON AS JPerson
SQL ADDRESS AS JAddress
```

次に、JPublisher を実行します。

```
jpub -input=input.typ -user=OE/OE
```

ここまでの操作を完了すると、person 型と address 型用にそれぞれ JPerson および JAddress という 2 つの Java クラスが生成されます。

手順 5 メッセージをエンキューするための Java コードの作成

このプログラムでは、Oracle JMS API を使用してメッセージを Streams トピックに公開します。

このプログラムの処理内容は、次のとおりです。

- 非 LCR ベースの ADT メッセージをトピックに公開します。
- JMS テキスト・メッセージをトピックに公開します。
- LCR ベースのメッセージをトピックに公開します。

```
import oracle.AQ.*;
import oracle.jms.*;
import javax.jms.*;
import java.lang.*;
import oracle.xdb.*;

public class StreamsEnq
{
  public static void main (String args [])
       throws java.sql.SQLException, ClassNotFoundException, JMSException
  {
     TopicConnectionFactory tc_fact= null;
     TopicConnection        t_conn = null;
     TopicSession           t_sess = null;

     try
     {
       if (args.length < 3 )
         System.out.println("Usage:java filename [SID] [HOST] [PORT]");
       else
       {
         /* Create the TopicConnectionFactory
          * Only the JDBC OCI driver can be used to access Streams through JMS
          */
         tc_fact = AQjmsFactory.getTopicConnectionFactory(
                     args[1], args[0], Integer.parseInt(args[2]), "oci8");

         t_conn = tc_fact.createTopicConnection( "OE","OE");

         /* Create a Topic Session */
         t_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

         /* Start the connection */
         t_conn.start() ;
```

```
        /* Publish non-LCR based messages */
        publishUserMessages(t_sess);

        /* Publish LCR based messages */
        publishLcrMessages(t_sess);

        t_sess.close() ;
        t_conn.close() ;
        System.out.println("End of StreamsEnq Demo") ;
      }
    }
    catch (Exception ex)
    {
      System.out.println("Exception-1: " + ex);
      ex.printStackTrace();
    }
}

/*
 * publishUserMessages - this method publishes an ADT message and a
 * JMS text message to a streams topic
 */
public static void publishUserMessages(TopicSession t_sess) throws Exception
{
  Topic           topic     = null;
  TopicPublisher  t_pub     = null;
  JPerson         pers      = null;
  JAddress        addr      = null;
  TextMessage     t_msg     = null;
  AdtMessage      adt_msg   = null;
  AQjmsAgent      agent     = null;
  AQjmsAgent[]    recipList = null;

  try
  {
    /* Get the topic */
    topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");

    /* Create a publisher */
    t_pub = t_sess.createPublisher(topic);


    /* Agent to access oe_queue */
    agent = new AQjmsAgent("explicit_enq", null);
```

```
  /* Create a PERSON adt message */
  adt_msg = ((AQjmsSession)t_sess).createAdtMessage();

  pers = new JPerson();
  addr = new JAddress();

  addr.setNum(new java.math.BigDecimal(500));
  addr.setStreet("Oracle Pkwy");

  pers.setName("Mark");
  pers.setHome(addr);

  /* Set the payload in the message */
  adt_msg.setAdtPayload(pers);

  ((AQjmsMessage)adt_msg).setSenderID(agent);

  System.out.println("Publish message 1 -type  PERSON\n");

  /* Create the recipient list */
  recipList = new AQjmsAgent[1];
  recipList[0] = new AQjmsAgent("explicit_dq", null);

  /* Publish the message */
  ((AQjmsTopicPublisher)t_pub).publish(topic, adt_msg, recipList);

  t_sess.commit();

  t_msg = t_sess.createTextMessage();

  t_msg.setText("Test message");
  t_msg.setStringProperty("color", "BLUE");
  t_msg.setIntProperty("year", 1999);

  ((AQjmsMessage)t_msg).setSenderID(agent);

  System.out.println("Publish message 2 -type  JMS TextMessage\n");

  /* Publish the message */
  ((AQjmsTopicPublisher)t_pub).publish(topic, t_msg, recipList);

  t_sess.commit();

}
catch (JMSException jms_ex)
```

```
  {
    System.out.println("JMS Exception: " + jms_ex);

    if(jms_ex.getLinkedException() != null)
      System.out.println("Linked Exception: " + jms_ex.getLinkedException());
  }
}

/*
 * publishLcrMessages - this method publishes an XML LCR message to a
 * streams topic
 */
public static void publishLcrMessages(TopicSession t_sess) throws Exception
{
  Topic                topic     = null;
  TopicPublisher       t_pub     = null;
  XMLType              xml_lcr   = null;
  AdtMessage           adt_msg   = null;
  AQjmsAgent           agent     = null;
  StringBuffer         lcr_data  = null;
  AQjmsAgent[]         recipList = null;
  java.sql.Connection  db_conn   = null;

  try
  {
    /* Get the topic */
    topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");

    /* Create a publisher */
    t_pub = t_sess.createPublisher(topic);

    /* Get the JDBC connection */
    db_conn = ((AQjmsSession)t_sess).getDBConnection();

    /* Agent to access oe_queue */
    agent = new AQjmsAgent("explicit_enq", null);


    /* Create a adt message */
    adt_msg = ((AQjmsSession)t_sess).createAdtMessage();

  /* Create the LCR representation in XML */
  lcr_data = new StringBuffer();

  lcr_data.append("<ROW_LCR ");
  lcr_data.append("xmlns='http://xmlns.oracle.com/streams/schemas/lcr' \n");
  lcr_data.append("xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' \n");
```

```
      lcr_data.append("xsi:schemaLocation='http://xmlns.oracle.com/streams/schemas/lcr
http://xmlns.oracle.com/streams/schemas/lcr/streamslcr.xsd'");
      lcr_data.append("> \n");

lcr_data.append("<source_database_name>source_dbname</source_database_name> \n");
      lcr_data.append("<command_type>INSERT</command_type> \n");
      lcr_data.append("<object_owner>Ram</object_owner> \n");
      lcr_data.append("<object_name>Emp</object_name> \n");
      lcr_data.append("<tag>0ABC</tag> \n");
      lcr_data.append("<transaction_id>0.0.0</transaction_id> \n");
      lcr_data.append("<scn>0</scn> \n");
      lcr_data.append("<old_values> \n");
      lcr_data.append("<old_value> \n");
      lcr_data.append("<column_name>C01</column_name> \n");
      lcr_data.append("<data><varchar2>Clob old</varchar2></data> \n");
      lcr_data.append("</old_value> \n");
      lcr_data.append("<old_value> \n");
      lcr_data.append("<column_name>C02</column_name> \n");
      lcr_data.append("<data><varchar2>A123FF</varchar2></data> \n");
      lcr_data.append("</old_value> \n");
      lcr_data.append("<old_value> \n");
      lcr_data.append("<column_name>C03</column_name> \n");
      lcr_data.append("<data> \n");

lcr_data.append("<date><value>1997-11-24</value><format>SYYYY-MM-DD</format></date>
\n");
      lcr_data.append("</data> \n");
      lcr_data.append("</old_value> \n");
      lcr_data.append("<old_value> \n");
      lcr_data.append("<column_name>C04</column_name> \n");
      lcr_data.append("<data> \n");

lcr_data.append("<timestamp><value>1999-05-31T13:20:00.000</value><format>SYYYY-MM-D
D'T'HH24:MI:SS.FF</format></timestamp> \n");
      lcr_data.append("</data> \n");
      lcr_data.append("</old_value> \n");
      lcr_data.append("<old_value> \n");
      lcr_data.append("<column_name>C05</column_name> \n");
      lcr_data.append("<data><raw>ABCDE</raw></data> \n");
      lcr_data.append("</old_value> \n");
      lcr_data.append("</old_values> \n");
      lcr_data.append("<new_values> \n");
      lcr_data.append("<new_value> \n");
      lcr_data.append("<column_name>C01</column_name> \n");
      lcr_data.append("<data><varchar2>A123FF</varchar2></data> \n");
      lcr_data.append("</new_value> \n");
      lcr_data.append("<new_value> \n");
```

```
      lcr_data.append("<column_name>C02</column_name> \n");
      lcr_data.append("<data><number>35.23</number></data> \n");
      lcr_data.append("</new_value> \n");
      lcr_data.append("<new_value> \n");
      lcr_data.append("<column_name>C03</column_name> \n");
      lcr_data.append("<data><number>-100000</number></data> \n");
      lcr_data.append("</new_value> \n");
      lcr_data.append("<new_value> \n");
      lcr_data.append("<column_name>C04</column_name> \n");
      lcr_data.append("<data><varchar>Hel lo</varchar></data> \n");
      lcr_data.append("</new_value> \n");
      lcr_data.append("<new_value> \n");
      lcr_data.append("<column_name>C05</column_name> \n");
      lcr_data.append("<data><char>wor ld</char></data> \n");
      lcr_data.append("</new_value> \n");
      lcr_data.append("</new_values> \n");
      lcr_data.append("</ROW_LCR>");

      /* Create the XMLType containing the LCR */
      xml_lcr = oracle.xdb.XMLType.createXML(db_conn, lcr_data.toString());

      /* Set the payload in the message */
      adt_msg.setAdtPayload(xml_lcr);

      ((AQjmsMessage)adt_msg).setSenderID(agent);

      System.out.println("Publish message 3 - XMLType containing LCR ROW\n");

      /* Create the recipient list */
      recipList = new AQjmsAgent[1];
      recipList[0] = new AQjmsAgent("explicit_dq", null);

      /* Publish the message */
      ((AQjmsTopicPublisher)t_pub).publish(topic, adt_msg, recipList);

      t_sess.commit();

    }
    catch (JMSException jms_ex)
    {
      System.out.println("JMS Exception: " + jms_ex);

      if(jms_ex.getLinkedException() != null)
        System.out.println("Linked Exception: " + jms_ex.getLinkedException());
    }
  }

}
```

手順 6 メッセージをデキューするための Java コードの作成

このプログラムでは、Oracle JMS API を使用してメッセージを Streams トピックから受信します。

このプログラムの処理内容は、次のとおりです。

- person、address および XMLType のマッピングを JMS typemap に登録します。
- Streams トピックから LCR メッセージを受信します。
- Streams トピックからユーザー ADT メッセージを受信します。

```
import oracle.AQ.*;
import oracle.jms.*;
import javax.jms.*;
import java.lang.*;
import oracle.xdb.*;
import java.sql.SQLException;

public class StreamsDeq
{
  public static void main (String args [])
       throws java.sql.SQLException, ClassNotFoundException, JMSException
  {
     TopicConnectionFactory tc_fact= null;
     TopicConnection        t_conn = null;
     TopicSession           t_sess = null;

     try
     {
       if (args.length < 3 )
         System.out.println("Usage:java filename [SID] [HOST] [PORT]");
       else
       {
         /* Create the TopicConnectionFactory
          * Only the JDBC OCI driver can be used to access Streams through JMS
          */
         tc_fact = AQjmsFactory.getTopicConnectionFactory(
                     args[1], args[0], Integer.parseInt(args[2]), "oci8");

         t_conn = tc_fact.createTopicConnection( "OE","OE");

         /* Create a Topic Session */
         t_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

         /* Start the connection */
         t_conn.start() ;
```

```
        receiveMessages(t_sess);

        t_sess.close() ;
        t_conn.close() ;
        System.out.println("\nEnd of StreamsDeq Demo") ;
      }
    }
    catch (Exception ex)
    {
      System.out.println("Exception-1: " + ex);
      ex.printStackTrace();
    }
}

/*
 * receiveMessages -This method receives messages from the Streams queue
 */
public static void receiveMessages(TopicSession t_sess) throws Exception
{
  Topic           topic   = null;
  JPerson         pers    = null;
  JAddress        addr    = null;
  XMLType         xtype   = null;
  TextMessage     t_msg   = null;
  AdtMessage      adt_msg = null;
  Message         jms_msg = null;
  TopicReceiver   t_recv  = null;
  int             i       = 0;
  java.util.Dictionary map= null;

  try
  {
    /* Get the topic */
    topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");

    /* Create a TopicReceiver to receive messages for consumer "jms_recv */
    t_recv = ((AQjmsSession)t_sess).createTopicReceiver(topic,
                                                        "jms_recv", null);

    map = ((AQjmsSession)t_sess).getTypeMap();

    /* Register mappings for ADDRESS and PERSON in the JMS typemap */
    map.put("OE.PERSON", Class.forName("JPerson"));
    map.put("OE.ADDRESS", Class.forName("JAddress"));

    /* Register mapping for XMLType in the TypeMap - required for LCRs */
    map.put("SYS.XMLTYPE", Class.forName("oracle.xdb.XMLTypeFactory"));
```

```
      System.out.println("Receive messages ...\n");

      do
      {
        try
        {
          jms_msg = (t_recv.receive(10));

          i++;

          /* Set navigation mode to NEXT_MESSAGE */

((AQjmsTopicReceiver)t_recv).setNavigationMode(AQjmsConstants.NAVIGATION_NEXT_MESSAG
E);
        }
        catch (JMSException jms_ex2)
        {
          if((jms_ex2.getLinkedException() != null) &&
             (jms_ex2.getLinkedException() instanceof SQLException))
          {
            SQLException sql_ex2 =(SQLException)(jms_ex2.getLinkedException());

            /* End of current transaction group
             * Use NEXT_TRANSACTION navigation mode
             */
            if(sql_ex2.getErrorCode() == 25235)
            {

((AQjmsTopicReceiver)t_recv).setNavigationMode(AQjmsConstants.NAVIGATION_NEXT_TRANSA
CTION);

              continue;
            }
            else
              throw jms_ex2;
          }
          else
            throw jms_ex2;
        }

        if(jms_msg == null)
        {
          System.out.println("\nNo more messages");
        }
        else
        {
          if(jms_msg instanceof AdtMessage)
```

```
      {
        adt_msg = (AdtMessage)jms_msg;

        System.out.println("Retrieved message " + i + ": " +
                           adt_msg.getAdtPayload());

        if(adt_msg.getAdtPayload() instanceof JPerson)
        {
          pers =(JPerson)( adt_msg.getAdtPayload());

          System.out.println("PERSON: Name: " + pers.getName());
        }
        else if(adt_msg.getAdtPayload() instanceof JAddress)
        {
          addr =(JAddress)( adt_msg.getAdtPayload());

          System.out.println("ADDRESS: Street" + addr.getStreet());
        }
        else if(adt_msg.getAdtPayload() instanceof oracle.xdb.XMLType)
        {
          xtype = (XMLType)adt_msg.getAdtPayload();

          System.out.println("XMLType: Data: \n" + xtype.getStringVal());

        }
        System.out.println("Msg id: " + adt_msg.getJMSMessageID());
        System.out.println();

      }
      else if(jms_msg instanceof TextMessage)
      {
        t_msg = (TextMessage)jms_msg;

        System.out.println("Retrieved message " + i + ": " +
                           t_msg.getText());

        System.out.println("Msg id: " + t_msg.getJMSMessageID());
        System.out.println();
      }
      else
        System.out.println("Invalid message type");
    }
  } while (jms_msg != null);

  t_sess.commit();
}
catch (JMSException jms_ex)
```

```
    {
      System.out.println("JMS Exception: " + jms_ex);

      if(jms_ex.getLinkedException() != null)
        System.out.println("Linked Exception: " + jms_ex.getLinkedException());

      t_sess.rollback();
    }
    catch (java.sql.SQLException sql_ex)
    {
      System.out.println("SQL Exception: " + sql_ex);
      sql_ex.printStackTrace();

      t_sess.rollback();
    }
  }
}
```

手順 7 スクリプトのコンパイル

```
javac StreamsEnq.java StreamsDeq.java JPerson.java JAddress.java
```

手順 8 エンキュー・プログラムの実行

```
java StreamsEnq  ORACLE_SID HOST PORT
```

たとえば、Oracle SID が orc182、ホストが hq_server、ポートが 1521 の場合は、次のように入力します。

```
java StreamsEnq orcl82 hq_server 1521
```

手順 9 デキュー・プログラムの実行

```
java StreamsDeq  ORACLE_SID HOST PORT
```

たとえば、Oracle SID が orc182、ホストが hq_server、ポートが 1520 の場合は、次のように入力します。

```
java StreamsDeq orcl82 hq_server 1521
```

20

単一データベースの取得および適用の例

この章では、単一データベースの例を説明します。例では、表に対する変更を取得し、取得された変更をキューに再エンキューする際に DML ハンドラを使用します。その後、変更のサブセットを異なる表へ適用します。

この章の内容は次のとおりです。

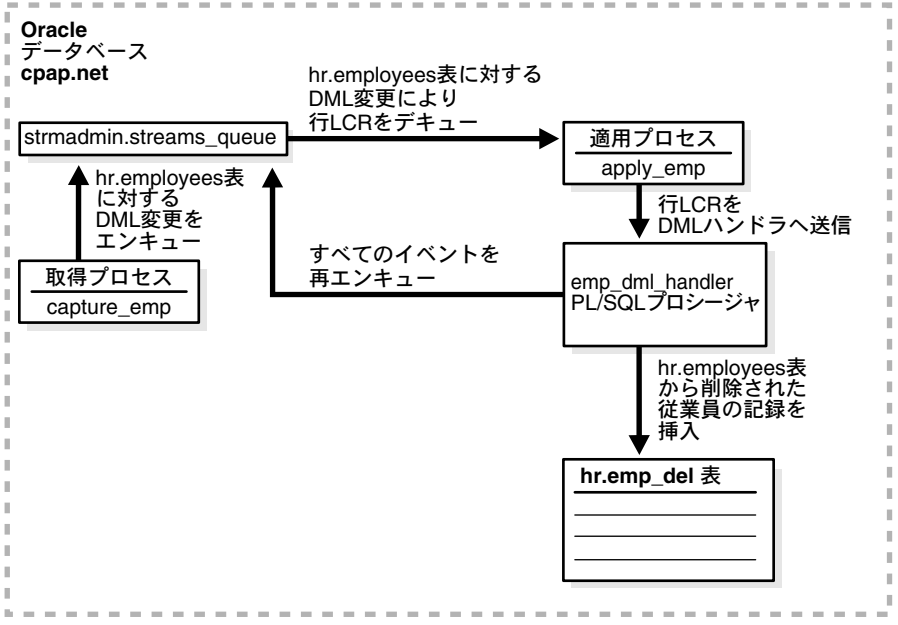
単一データベースの取得および適用の例の概要

この章の例では、Streams を使用して、単一データベース cpap.net でデータ操作言語（DML）の変更を取得および適用する方法を示します。具体的には、hr スキーマでの employees 表に対する DML 変更の例が取得されます。ここでは、行の論理変更レコード（LCR）がキュー streams_queue に含められます。その場合、適用プロセスによってこれらの行 LCR は同一のキューからデキューされ、DML ハンドラに送られます。DML ハンドラでは、取得した行 LCR に対し、次のアクションが実行されます。

- 取得したすべての行 LCR がキューに再エンキューされます。行 LCR は、取得されるとバッファ・キューに存在することになり、明示的にはデキューできません。行 LCR が DML ハンドラにより再エンキューされると、アプリケーションで明示的にデキュー可能になります。この例では、これらの行 LCR をデキューするアプリケーションは作成されません。
- 削除された従業員のレコードを hr の emp_del 表に挿入します。この例では、すべての削除された従業員のレコードを保持するのに、emp_del 表が使用されることを想定しています。DML ハンドラは、各行 LCR に DELETE 文が含まれるかどうかを判別するために使用されます。行 LCR に DELETE 文が含まれることが DML ハンドラで検出されると、DML ハンドラにより、emp_del 表に対する DELETE 文が INSERT 文に変換されます。

図 20-1 に、この環境の概要を示します。

図 20-1 単一データベースの取得および適用の例



関連項目：

- 第 2 章「Streams の取得プロセス」
- DML ハンドラの詳細は、4-4 ページの「LCR イベントの処理」を参照してください。

前提条件

この章の例を開始する前に、前提条件となる次の作業を完了する必要があります。

- データベースについて、次の初期化パラメータを指定の値に設定します。
  - AQ_TM_PROCESSES: このパラメータでは、キュー・モニター・プロセスを設定します。値 1 ～ 10 では、メッセージの監視用に作成するキュー・モニター・プロセスの数を指定します。AQ_TM_PROCESSES を指定しないか、0 に設定すると、キュー・モニター・プロセスは作成されません。この例では、AQ_TM_PROCESSES を 1 以上の値に設定する必要があります。

    このパラメータを 1 以上に設定すると、指定した数のキュー・モニター・プロセスが起動します。これらのキュー・モニター・プロセスは、遅延や期限切れなど、時間ベースのメッセージ操作の管理、指定した保存期間を過ぎて保存されているメッセージのクリーン・アップ、および保存期間が 0（ゼロ）の場合のコンシューム済みメッセージのクリーン・アップを受け持ちます。
  - COMPATIBLE: このパラメータは 9.2.0 以上に設定する必要があります。
  - LOG_PARALLELISM: データベースによってイベントが取得されるため、このパラメータは、1 に設定する必要があります。

    関連項目： Streams 環境で重要な他の初期化パラメータについては、11-4 ページの「Streams に関連する初期化パラメータの設定」を参照してください。
- ARCHIVELOG モードで実行されるようにデータベースを設定します。取得される変更を生成するデータベースは、ARCHIVELOG モードで実行されている必要があります。

  関連項目： ARCHIVELOG モードでデータベースを実行する方法については、『Oracle9*i* データベース管理者ガイド』を参照してください。
- この例では、新規ユーザーが Streams 管理者（strmadmin）として作成され、このユーザーのデータに使用する表領域の指定を求めるプロンプトが表示されます。この例を開始する前に、新規の表領域を作成するか、既存の表領域を Streams 管理者用に識別してください。SYSTEM 表領域は、Streams 管理者用にしないでください。

環境の設定

次の手順に従って、hr.emp_del 表を作成し、Streams 管理者を設定して、キューを作成します。

1. 出力の表示と結果のスプーリング
2. hr.emp_del 表の作成
3. cpap.net でのユーザーの設定
4. cpap.net での Streams キューの作成
5. スプール結果のチェック

> **注意：** このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 20-7 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************* BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_setup_catapp.out

/*
```

手順 2 hr.emp_del 表の作成

cpap.net に hr ユーザーとして接続します。

```
*/

CONNECT hr/hr@cpap.net

/*
```

hr.emp_del 表を作成します。emp_del 表の shape は、emp_del 表への行の挿入日付が記録される timestamp 列が追加されていることを除き、employees 表と同一です。

```
*/

CREATE TABLE emp_del(
  employee_id    NUMBER(6),
  first_name     VARCHAR2(20),
  last_name      VARCHAR2(25),
  email          VARCHAR2(25),
  phone_number   VARCHAR2(20),
  hire_date      DATE,
  job_id         VARCHAR2(10),
  salary         NUMBER(8,2),
  commission_pct NUMBER(2,2),
  manager_id     NUMBER(6),
  department_id  NUMBER(4),
  timestamp      DATE);

CREATE UNIQUE INDEX emp_del_id_pk ON emp_del (employee_id);

ALTER TABLE emp_del ADD (CONSTRAINT emp_del_id_pk PRIMARY KEY (employee_id));

/*
```

手順 3 cpap.net でのユーザーの設定

cpap.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@cpap.net AS SYSDBA

/*
```

Streams 管理者 strmadmin を作成し、このユーザーに必要な権限を付与します。これらの権限を付与されたユーザーは、キューの管理、Streams に関連するパッケージ内のサブプログラムの実行、ルール・セットおよびルールの作成、データ・ディクショナリ・ビューとキュー表の問合せによる Streams 環境の監視を行うことができます。このユーザー用に、異なる名前を選択できます。

この例では、Streams 管理者は適用プロセスの適用ユーザーとなり、変更を hr.emp_del 表に適用できる必要があります。したがって、Streams 管理者には、この表に対する ALL 権限が付与されます。

注意：

- セキュリティを確保するために、Streams 管理者には strmadminpw 以外のパスワードを使用してください。
- Streams 管理者には、SELECT_CATALOG_ROLE は不要です。この例で付与されているのは、Streams 管理者が環境を簡単に監視できるようにするためです。
- Oracle Enterprise Manager の Streams ツールを使用する予定の場合は、Streams 管理者にこの手順で示す権限の他に SELECT ANY DICTIONARY 権限を付与します。
- ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on cpap.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
                     QUOTA UNLIMITED ON &streams_tbs;

GRANT ALL ON hr.emp_del TO strmadmin;

GRANT EXECUTE ON DBMS_APPLY_ADM        TO strmadmin;
GRANT EXECUTE ON DBMS_AQ               TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM            TO strmadmin;
GRANT EXECUTE ON DBMS_CAPTURE_ADM      TO strmadmin;
GRANT EXECUTE ON DBMS_FLASHBACK        TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM      TO strmadmin;

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/
```

```
BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

/*
```

手順 4 cpap.net での Streams キューの作成

cpap.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@cpap.net

/*
```

SET_UP_QUEUE プロシージャを実行して、cpap.net でキュー streams_queue を作成します。このキューは、適用プロセスからデキュー対象となる取得された変更を保持することにより、Streams キューとして機能します。

SET_UP_QUEUE プロシージャを実行すると、次のアクションが実行されます。

- キュー表 streams_queue_table が作成されます。このキュー表の所有者は Streams 管理者（strmadmin）であり、このユーザーのデフォルト記憶域が使用されます。
- Streams 管理者（strmadmin）が所有するキュー streams_queue が作成されます。
- キューが起動されます。

```
*/

EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();

/*
```

手順 5 スプール結果のチェック

このスクリプトの完了後に streams_setup_catapp.out スプール・ファイルをチェックして、すべてのアクションが正常終了したことを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT *****************************/
```

取得および適用の構成

次の手順に従って、hr.employees 表への変更を取得し、DML ハンドラを使用してカスタマイズされた方法で単一データベースに変更を適用します。

1. 出力の表示と結果のスプーリング
2. cpap.net で LogMiner の表に使用する代替表領域の作成
3. cpap.net でのサプリメンタル・ロギングの指定
4. cpap.net での取得プロセスの構成
5. hr.employees 表用のインスタンス化 SCN の設定
6. エージェント emp_agent の作成
7. キュー・サブスクライバの作成
8. 行 LCR をエンキューするプロシージャの作成
9. DML ハンドラ・プロシージャの作成
10. hr.employees 表用の DML ハンドラの設定
11. 再エンキューされたイベントをデキューするプロシージャの作成
12. cpap.net での適用プロセスの構成
13. cpap.net での適用プロセスの起動
14. cpap.net での取得プロセスの起動
15. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 20-18 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************* BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_config_capapp.out

/*
```

手順 2 cpap.net で LogMiner の表に使用する代替表領域の作成

デフォルトでは、LogMiner の表は SYSTEM 表領域にありますが、変更を取得するために取得プロセスが起動すると、SYSTEM 表領域にこれらの表に使用する領域が足りなくなる場合があります。したがって、LogMiner の表に使用する代替表領域を作成する必要があります。

関連項目： 2-18 ページ「LogMiner の表のための代替表領域」

cpap.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@cpap.net AS SYSDBA

/*
```

LogMiner の表に使用する代替表領域を作成します。

注意： 各 ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

```
*/

ACCEPT tspace_name DEFAULT 'logmnrts' PROMPT 'Enter tablespace name (for example,
logmnrts): '

ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the datafile
directory (for example, /usr/oracle/dbs): '

ACCEPT db_file_name DEFAULT 'logmnrts.dbf' PROMPT 'Enter the name of the datafile
(for example, logmnrts.dbf): '
```

```
CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'
  SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');

/*
```

手順 3 cpap.net でのサプリメンタル・ロギングの指定

サプリメンタル・ロギングによって、表に対して行われた変更に関する追加情報が REDO ログに書き込まれます。適用プロセスでは、一意の行の識別など特定の操作を実行するために、この追加情報が必要になります。

次の文では、hr.employees 表内の主キー列について、無条件のサプリメンタル・ログ・グループを指定します。

関連項目：

- 2-10 ページ「Streams 環境内のサプリメンタル・ロギング」
- 12-8 ページ「ソース・データベースでのサプリメンタル・ロギングの指定」

```
*/

ALTER TABLE hr.employees ADD SUPPLEMENTAL LOG GROUP log_group_employees_pk
  (employee_id) ALWAYS;

/*
```

手順 4 cpap.net での取得プロセスの構成

cpap.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@cpap.net

/*
```

cpap.net で hr.employees 表に対する DML 変更を取得するために、取得プロセスを構成します。この手順では、この表に対する DML 変更を、取得プロセスで取得して指定のキューにエンキューするように指定します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name     => 'hr.employees',
    streams_type   => 'capture',
    streams_name   => 'capture_emp',
    queue_name     => 'strmadmin.streams_queue',
    include_dml    =>  true,
    include_ddl    =>  false);
END;
/

/*
```

手順 5 hr.employees 表用のインスタンス化 SCN の設定

この例では、1 つの単一データベースにおける変更を取得および適用するため、インスタンス化は必要ありません。ただし、cpap.net データベースの適用プロセスに対して、特定のシステム変更番号（SCN）より後に hr.employees 表に対して行われた変更を適用するように指示する必要があります。

この例では、DBMS_FLASHBACK パッケージの GET_SYSTEM_CHANGE_NUMBER ファンクションを使用して、データベースの現行の SCN を取得します。この SCN は、DBMS_APPLY_ADM パッケージで SET_TABLE_INSTANTIATION_SCN プロシージャを実行するために使用されます。

SET_TABLE_INSTANTIATION_SCN プロシージャでは、ある表の LCR のうち、適用プロセスによって無視される LCR と適用される LCR が制御されます。ソース・データベースからの表に関する LCR のコミット SCN が、接続先データベースでその表のインスタンス化 SCN 以下であれば、接続先データベースの適用プロセスでは LCR が廃棄されます。それ以外の場合は、適用プロセスによって LCR が適用されます。この例では、cpap.net データベースは、ソース・データベースと接続先データベースの両方を兼ねています。

適用プロセスにより、この手順で取得した SCN より後にコミットされた SCN を持つ hr.employees 表にトランザクションが適用されます。

注意： hr.employees 表では、インスタンス化の準備も完了済であることが必要です。この準備は、手順 4 にある hr.employees 表への DML 変更を取得するためのルールを使用して取得プロセスが構成されるとき、自動的に行われていました。

```
*/

DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN(
    source_object_name    => 'hr.employees',
    source_database_name  => 'cpap.net',
    instantiation_scn     => iscn);
END;
/

/*
```

手順 6 エージェント emp_agent の作成

この例では、エージェント emp_agent を使用して、streams_queue に対して明示的にエンキューおよびデキューが行われます。strmadmin ユーザーがこのキュー用のキュー表を所有するため、strmadmin ユーザーはキューの保護ユーザーとなります。この手順によって、エージェント emp_agent が作成され、このエージェントが strmadmin ユーザーと関連付けられます。これにより、エージェントは保護キューへのエンキューや保護キューからのデキューに使用可能になります。

```
*/

BEGIN
  DBMS_AQADM.CREATE_AQ_AGENT(
     agent_name => 'emp_agent');
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name  => 'emp_agent',
    db_username => 'strmadmin');
END;
/

/*
```

手順 7 キュー・サブスクライバの作成

再エンキューされたイベントをデキューするためにアプリケーションで使用されるサブスクライバを作成します。イベントがキューに再エンキューされる前に、サブスクライバを 1 つ以上指定する必要があります。

```
*/

DECLARE
  subscriber SYS.AQ$_AGENT;
BEGIN
  subscriber :=  SYS.AQ$_AGENT('emp_agent', NULL, NULL);
  SYS.DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name          =>  'strmadmin.streams_queue',
    subscriber          =>  subscriber,
    rule                =>  NULL,
    transformation      =>  NULL);
END;
/

/*
```

手順 8 行 LCR をエンキューするプロシージャの作成

この手順によって、enq_row_lcr プロシージャが作成されます。このプロシージャは、hr.employees 表に対する変更を含む行 LCR をエンキューするために、手順 9 において作成された DML ハンドラ・プロシージャで使用します。

```
*/

CREATE OR REPLACE PROCEDURE enq_row_lcr(in_any IN SYS.ANYDATA) IS
  enqopt       DBMS_AQ.ENQUEUE_OPTIONS_T;
  mprop        DBMS_AQ.MESSAGE_PROPERTIES_T;
  recipients   DBMS_AQ.AQ$_RECIPIENT_LIST_T;
  enq_eventid  RAW(16);
BEGIN
  mprop.SENDER_ID := SYS.AQ$_AGENT(
    name     => 'emp_agent',
    address  => NULL,
    protocol => NULL);
  recipients(1) := SYS.AQ$_AGENT(
    name     => 'emp_agent',
    address  => NULL,
    protocol => NULL);
  mprop.RECIPIENT_LIST := recipients;
```

```
  DBMS_AQ.ENQUEUE(
    queue_name         => 'strmadmin.streams_queue',
    enqueue_options    => enqopt,
    message_properties => mprop,
    payload            => in_any,
    msgid              => enq_eventid);
END;
/

/*
```

手順 9 DML ハンドラ・プロシージャの作成

この手順では、emp_dml_handler プロシージャを作成します。このプロシージャが hr.employees 表に対する DML 変更の DML ハンドラになります。このプロシージャでは、次のアクションが実行されます。

- 手順 8 で作成した enq_row_lcr プロシージャを使用して、すべての行 LCR を streams_queue に再エンキューします。
- DELETE コマンド・タイプを含む行 LCR を INSERT 行 LCR に変換し、次に行 LCR を実行して、変換された行 LCR を hr.emp_del 表に挿入します。

```
*/

CREATE OR REPLACE PROCEDURE emp_dml_handler(in_any IN SYS.ANYDATA) IS
  lcr          SYS.LCR$_ROW_RECORD;
  rc           PLS_INTEGER;
  command      VARCHAR2(10);
  old_values   SYS.LCR$_ROW_LIST;
BEGIN
  -- Re-enqueue the row LCR for explicit dequeue by another application
  enq_row_lcr(in_any);
  -- Access the LCR
  rc := in_any.GETOBJECT(lcr);
  -- Get the object command type
  command := lcr.GET_COMMAND_TYPE();
  -- Check for DELETE command on the hr.employees table
  IF command = 'DELETE' THEN
    -- Set the command_type in the row LCR to INSERT
    lcr.SET_COMMAND_TYPE('INSERT');
    -- Set the object_name in the row LCR to EMP_DEL
    lcr.SET_OBJECT_NAME('EMP_DEL');
    -- Get the old values in the row LCR
    old_values := lcr.GET_VALUES('old');
    -- Set the old values in the row LCR to the new values in the row LCR
    lcr.SET_VALUES('new', old_values);
    -- Set the old values in the row LCR to NULL
```

```
    lcr.SET_VALUES('old', NULL);
    -- Add a SYSDATE value for the timestamp column
    lcr.ADD_COLUMN('new', 'TIMESTAMP', SYS.AnyData.ConvertDate(SYSDATE));
    --  Apply the row LCR as an INSERT into the EMP_DEL table
    lcr.EXECUTE(true);
  END IF;
END;
/

/*
```

手順 10 hr.employees 表用の DML ハンドラの設定

hr.employees 表用の DML ハンドラを手順 9 で作成したプロシージャに設定します。DML ハンドラは、INSERT、UPDATE および DELETE 表の使用可能な操作ごとに個別に設定する必要があることに注意してください。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name         => 'hr.employees',
    object_type         => 'TABLE',
    operation_name      => 'INSERT',
    error_handler       => false,
    user_procedure      => 'strmadmin.emp_dml_handler',
    apply_database_link => NULL);
END;
/

BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name         => 'hr.employees',
    object_type         => 'TABLE',
    operation_name      => 'UPDATE',
    error_handler       => false,
    user_procedure      => 'strmadmin.emp_dml_handler',
    apply_database_link => NULL);
END;
/
```

```
BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name         => 'hr.employees',
    object_type         => 'TABLE',
    operation_name      => 'DELETE',
    error_handler       => false,
    user_procedure      => 'strmadmin.emp_dml_handler',
    apply_database_link => NULL);
END;
/

/*
```

手順 11 再エンキューされたイベントをデキューするプロシージャの作成

この手順で作成する emp_dq プロシージャは、手順 9 で作成した DML ハンドラによって再エンキューされたイベントをデキューするのに使用できます。emp_dq プロシージャを実行すると、キュー内の各行 LCR がデキューされ、行 LCR のコマンドのタイプ、INSERT、UPDATE または DELETE のいずれかが表示されます。コマンド・タイプのみでなく、行 LCR 内のあらゆる情報をアクセスおよび表示できます。

> **関連項目：** LCR の情報の表示に関する詳細は、17-35 ページの「適用エラーの詳細情報の表示」を参照してください。

```
*/

CREATE OR REPLACE PROCEDURE emp_dq (consumer IN VARCHAR2) AS
  deqopt       DBMS_AQ.DEQUEUE_OPTIONS_T;
  mprop        DBMS_AQ.MESSAGE_PROPERTIES_T;
  msgid        RAW(16);
  payload      SYS.AnyData;
  new_messages BOOLEAN := TRUE;
  row_lcr      SYS.LCR$_ROW_RECORD;
  tc           pls_integer;
  next_trans   EXCEPTION;
  no_messages  EXCEPTION;
  pragma exception_init (next_trans, -25235);
  pragma exception_init (no_messages, -25228);
BEGIN
  deqopt.consumer_name := consumer;
  deqopt.wait := 1;
  WHILE (new_messages) LOOP
    BEGIN
      DBMS_AQ.DEQUEUE(
        queue_name          =>  'strmadmin.streams_queue',
        dequeue_options     =>  deqopt,
        message_properties  =>  mprop,
```

```
        payload             =>  payload,
        msgid               =>  msgid);
      COMMIT;
      deqopt.navigation := DBMS_AQ.NEXT;
      IF (payload.GetTypeName = 'SYS.LCR$_ROW_RECORD') THEN
        tc := payload.GetObject(row_lcr);
        DBMS_OUTPUT.PUT_LINE(row_lcr.GET_COMMAND_TYPE || ' row LCR dequeued');
      END IF;
      EXCEPTION
        WHEN next_trans THEN
        deqopt.navigation := DBMS_AQ.NEXT_TRANSACTION;
        WHEN no_messages THEN
          new_messages  := FALSE;
          DBMS_OUTPUT.PUT_LINE('No more events');
     END;
  END LOOP;
END;
/

/*
```

手順 12 cpap.net での適用プロセスの構成

DML 変更を hr.employees 表に適用する適用プロセスを構成します。適用プロセスの DML ハンドラにより、削除された従業員が emp_del 表に挿入されますが、キュー内の行 LCR には、emp_del 表ではなく employees 表への変更が含まれるため、このルールで employees 表が指定されます。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.employees',
    streams_type    => 'apply',
    streams_name    => 'apply_emp',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  false,
    source_database => 'cpap.net');
END;
/

/*
```

手順 13 cpap.net での適用プロセスの起動

エラーが発生しても適用プロセスが無効化されないように、disable_on_error パラメータを n に設定して、cpap.net で適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_emp',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply_emp');
END;
/

/*
```

手順 14 cpap.net での取得プロセスの起動

cpap.net で取得プロセスを起動します。

```
*/

BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name  => 'capture_emp');
END;
/

/*
```

手順 15 スプール結果のチェック

このスクリプトの完了後に streams_config_capapp.out スプール・ファイルをチェックして、すべてのアクションが正常終了したことを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT ******************************/
```

DML 変更、結果の問合せおよびイベントのデキュー

次の手順に従って、hr.employees 表に対する DML 変更を行い、hr.emp_del 表への挿入結果および streams_queue_table で再エンキューされたイベントの問合せを行い、DML ハンドラにより再エンキューされたイベントをデキューします。

手順 1 hr.employees 表での INSERT、UPDATE および DELETE の実行

hr.employees 表に対して、次の DML 変更を行います。

```
CONNECT hr/hr@cpap.net

INSERT INTO hr.employees values(207, 'JOHN', 'SMITH', 'JSMITH@MYCOMPANY.COM',
  NULL, '07-JUN-94', 'AC_ACCOUNT', 777, NULL, NULL, 110);
COMMIT;

UPDATE hr.employees SET salary=5999 WHERE employee_id=206;
COMMIT;

DELETE FROM hr.employees WHERE employee_id=207;
COMMIT;
```

手順 2 hr.emp_del 表および streams_queue_table の問合せ

前述の手順で実行した変更を取得し、適用するのに必要な時間が経過した後、次の問合せを実行して結果を確認します。

```
CONNECT strmadmin/strmadminpw@cpap.net

SELECT * FROM hr.emp_del;

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$STREAMS_QUEUE_TABLE;
```

最初の問合せを実行する際、employee_id が 207 の従業員のレコードを確認します。この従業員は前述の手順で削除されています。2 度目の問合せを実行する場合には、前述の手順で行われた変更すべてで発生する、再エンキューされたイベントを確認する必要があります。これらのイベントに対して MSG_STATE は READY であることが必要です。

手順 3 DML ハンドラにより再エンキューされたイベントのデキュー

emp_dq プロシージャを使用して、DML ハンドラにより再エンキューされたイベントをデキューします。

```
SET SERVEROUTPUT ON SIZE 100000

EXEC emp_dq('emp_agent');
```

DML 文により変更された各行に対し、戻される行は 1 行で、各行には変更のコマンド・タイプ（INSERT、UPDATE あるいは DELETE のいずれか）が表示されます。手順 2 にあるキュー表の問合せをイベントのデキュー後に繰り返す場合、デキューされたイベントがコンシューム済みとなる必要があります。つまり、これらのイベントに対する MSG_STATE は、PROCESSED であることが必要です。

```
SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$STREAMS_QUEUE_TABLE;
```

21

簡単な単一のソース・レプリケーションの例

この章では、Streams を使用して構成できる簡単な単一のソース・レプリケーション環境の例を説明します。

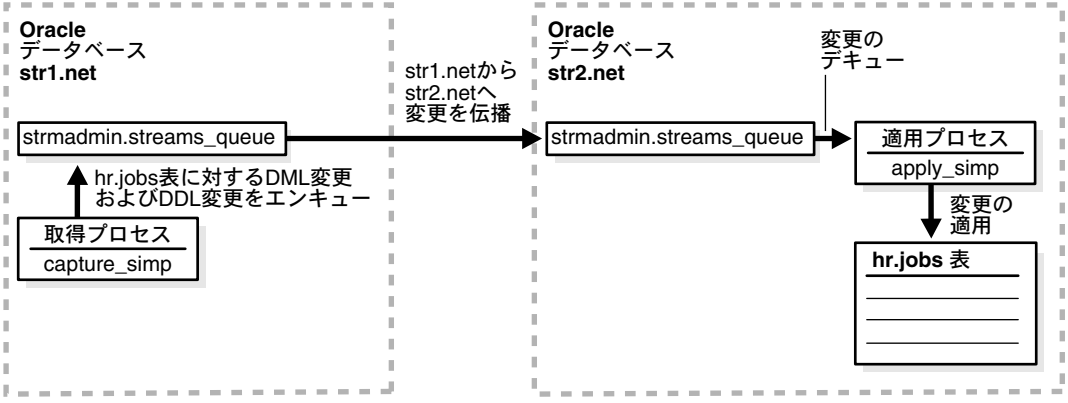
この章の内容は次のとおりです。

簡単な単一のソース・レプリケーションの例の概要

この章の例では、Streams を使用して 2 つのデータベース間の 1 つの表でデータをレプリケートする方法を示します。取得プロセスによって、str1.net Oracle データベースで hr スキーマ内の jobs 表に対して行われたデータ操作言語（DML）変更およびデータ定義言語（DDL）変更が取得され、これらの変更が伝播によって str2.net Oracle データベースへ伝播されます。次に、適用プロセスによりこれらの変更が str2.net データベースで適用されます。この例では、hr.jobs 表が str2.net データベースで読取り専用であることを想定しています。

図 21-1 に、この環境の概要を示します。

図 21-1 単一のソース・データベースからのデータ共有の簡単な例



前提条件

この章の例を開始する前に、前提条件となる次の作業を完了する必要があります。

- 次の初期化パラメータを指定の値に設定します。
  - AQ_TM_PROCESSES: このパラメータでは、キュー・モニター・プロセスを設定します。値 1 ～ 10 では、メッセージの監視用に作成するキュー・モニター・プロセスの数を指定します。AQ_TM_PROCESSES を指定しないか、0 に設定すると、キュー・モニター・プロセスは作成されません。この例では、各データベースで AQ_TM_PROCESSES を 1 以上の値に設定する必要があります。

    このパラメータを 1 以上に設定すると、指定した数のキュー・モニター・プロセスが起動します。これらのキュー・モニター・プロセスは、遅延や期限切れなど、時間ベースのメッセージ操作の管理、指定した保存期間を過ぎて保存されているメッセージのクリーン・アップ、および保存期間が 0（ゼロ）の場合のコンシューム済みメッセージのクリーン・アップを受け持ちます。
  - GLOBAL_NAMES: Streams 環境の各データベースで、このパラメータを true に設定する必要があります。
  - JOB_QUEUE_PROCESSES: Streams 環境でイベントを伝播する各データベースで、このパラメータを 2 以上の値に設定する必要があります。同時に実行できるジョブの最大数に 1 を加えた値に設定してください。この例では、str1.net でイベントが伝播されます。そのため、str1.net で JOB_QUEUE_PROCESSES を 2 以上の値に設定する必要があります。
  - COMPATIBLE: Streams 環境の各データベースで、このパラメータを 9.2.0 以上に設定する必要があります。
  - LOG_PARALLELISM: このパラメータは、イベントを取得する各データベースで 1 に設定する必要があります。この例では、このパラメータを str1.net で 1 に設定する必要があります。

    関連項目： Streams 環境で重要な他の初期化パラメータについては、11-4 ページの「Streams に関連する初期化パラメータの設定」を参照してください。

- 取得される変更を生成するデータベースは、ARCHIVELOG モードで実行している必要があります。この例では、str1.net で変更が生成されるため、str1.net は ARCHIVELOG モードで実行する必要があります。

  関連項目： ARCHIVELOG モードでデータベースを実行する方法については、『Oracle9*i* データベース管理者ガイド』を参照してください。

- ネットワークと Oracle Net を、str1.net データベースが str2.net データベースと通信できるように構成します。

  関連項目：『Oracle9*i* Net Services 管理者ガイド』

- この例では、各データベースで新規ユーザーが Streams 管理者（strmadmin）として作成され、このユーザーのデータに使用する表領域の指定を求めるプロンプトが表示されます。この例を開始する前に、各データベースで Streams 管理者用に新規の表領域を作成するか、既存の表領域を識別してください。SYSTEM 表領域は、Streams 管理者用にしないでください。

ユーザーの設定とキューおよびデータベース・リンクの作成

次の手順に従って、2 つの Oracle データベースを含む Streams レプリケーション環境用にユーザーを設定して、キューとデータベース・リンクを設定します。

1. 出力の表示と結果のスプーリング
2. str1.net でのユーザーの設定
3. str1.net での Streams キューの作成
4. str1.net でのデータベース・リンクの作成
5. str2.net でのユーザーの設定
6. str2.net での Streams キューの設定
7. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 21-9 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************* BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_setup_simple.out

/*
```

手順 2 str1.net でのユーザーの設定

str1.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@str1.net AS SYSDBA

/*
```

Streams 管理者 strmadmin を作成し、このユーザーに必要な権限を付与します。これらの権限を付与されたユーザーは、キューの管理、Streams に関連するパッケージ内のサブプログラムの実行、ルール・セットおよびルールの作成、データ・ディクショナリ・ビューとキュー表の問合せによる Streams 環境の監視を行うことができます。このユーザー用に、異なる名前を選択できます。

注意：

- セキュリティを確保するために、Streams 管理者には strmadminpw 以外のパスワードを使用してください。
- Streams 管理者には、SELECT_CATALOG_ROLE は不要です。この例で付与されているのは、Streams 管理者が環境を簡単に監視できるようにするためです。
- Oracle Enterprise Manager の Streams ツールを使用する予定の場合は、Streams 管理者にこの手順で示す権限の他に SELECT ANY DICTIONARY 権限を付与します。
- ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on str1.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
                     QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_AQADM            TO strmadmin;
GRANT EXECUTE ON DBMS_CAPTURE_ADM      TO strmadmin;
GRANT EXECUTE ON DBMS_PROPAGATION_ADM  TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM      TO strmadmin;

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

/*
```

手順 3 str1.net での Streams キューの作成

変更を取得するデータベースで Streams 管理者として接続します。この例では、データベース `str1.net` に接続します。

```
*/

CONNECT strmadmin/strmadminpw@str1.net

/*
```

`SET_UP_QUEUE` プロシージャを実行して、`str1.net` でキュー `streams_queue` を作成します。このキューは、取得され他のデータベースに伝播される変更を保持することで、Streams キューとして機能します。

SET_UP_QUEUE プロシージャを実行すると、次のアクションが実行されます。

- キュー表 streams_queue_table が作成されます。このキュー表の所有者は Streams 管理者（strmadmin）であり、このユーザーのデフォルト記憶域が使用されます。
- Streams 管理者（strmadmin）が所有するキュー streams_queue が作成されます。
- キューが起動されます。

```
*/

EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();

/*
```

手順 4 str1.net でのデータベース・リンクの作成

変更が取得されるデータベースから変更の伝播先となるデータベースへの、データベース・リンクを作成します。この例では、変更が取得されるデータベースは str1.net、これらの変更の伝播先となるデータベースは str2.net です。

```
*/

CREATE DATABASE LINK str2.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'str2.net';

/*
```

手順 5 str2.net でのユーザーの設定

str2.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@str2.net AS SYSDBA

/*
```

Streams 管理者 strmadmin を作成し、このユーザーに必要な権限を付与します。これらの権限を付与されたユーザーは、キューの管理、Streams に関連するパッケージ内のサブプログラムの実行、ルール・セットおよびルールの作成、データ・ディクショナリ・ビューとキュー表の問合せによる Streams 環境の監視を行うことができます。この例では、Streams 管理者は適用プロセスの適用ユーザーになり、str2.net で変更を hr.jobs 表に適用できる必要があります。したがって、Streams 管理者には、この表に対する ALL 権限が付与されます。Streams 管理者には、異なる名前を選択できます。

注意：

- セキュリティを確保するために、Streams 管理者には strmadminpw 以外のパスワードを使用してください。
- Streams 管理者には、SELECT_CATALOG_ROLE は不要です。この例で付与されているのは、Streams 管理者が環境を簡単に監視できるようにするためです。
- Oracle Enterprise Manager の Streams ツールを使用する予定の場合は、Streams 管理者にこの手順で示す権限の他に SELECT ANY DICTIONARY 権限を付与します。
- ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

GRANT ALL ON hr.jobs TO strmadmin;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on str2.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
                     QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM        TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM            TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM      TO strmadmin;

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
```

```
END;
/

/*
```

手順 6 str2.net での Streams キューの設定

str2.net で Streams 管理者として接続します。

```
*/

CONNECT strmadmin/strmadminpw@str2.net

/*
```

SET_UP_QUEUE プロシージャを実行して、str2.net でキュー streams_queue を作成します。このキューは、このデータベースで適用される変更を保持することで Streams キューとして機能します。

SET_UP_QUEUE プロシージャを実行すると、次のアクションが実行されます。

- キュー表 streams_queue_table が作成されます。このキュー表の所有者は Streams 管理者（strmadmin）であり、このユーザーのデフォルト記憶域が使用されます。
- Streams 管理者（strmadmin）が所有するキュー streams_queue が作成されます。
- キューが起動されます。

```
*/

EXEC  DBMS_STREAMS_ADM.SET_UP_QUEUE();

/*
```

手順 7 スプール結果のチェック

このスクリプトの完了後に streams_setup_simple.out スプール・ファイルをチェックして、すべてのアクションが正常終了したことを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT *****************************/
```

1 つの表に対する変更の取得、伝播および適用の構成

次の手順に従って、DBMS_STEAMS_ADM パッケージを使用して、hr.jobs 表に対する取得、伝播および適用の定義を指定します。

1. 出力の表示と結果のスプーリング
2. str1.net での LogMiner 表に使用する代替表領域の作成
3. str1.net でのサプリメンタル・ロギングの指定
4. str1.net での伝播の構成
5. str1.net での取得プロセスの構成
6. str2.net での hr.jobs 表のインスタンス化
7. str2.net での hr.jobs 表のサプリメンタル・ログ・グループの削除
8. str2.net での適用プロセスの構成
9. str2.net での適用プロセスの起動
10. str1.net での取得プロセスの起動
11. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 21-17 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************* BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_share_jobs.out

/*
```

手順 2 str1.net での LogMiner 表に使用する代替表領域の作成

デフォルトでは、LogMiner の表は SYSTEM 表領域にありますが、変更を取得するために取得プロセスが起動すると、SYSTEM 表領域にこれらの表に使用する領域が足りなくなる場合があります。したがって、LogMiner の表に使用する代替表領域を作成する必要があります。

関連項目： 2-18 ページ「LogMiner の表のための代替表領域」

str1.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@str1.net AS SYSDBA

/*
```

LogMiner の表に使用する代替表領域を作成します。

注意： 各 ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

```
*/

ACCEPT tspace_name DEFAULT 'logmnrts' PROMPT 'Enter tablespace name (for example,
logmnrts): '

ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the datafile
directory (for example, /usr/oracle/dbs): '

ACCEPT db_file_name DEFAULT 'logmnrts.dbf' PROMPT 'Enter the name of the datafile
(for example, logmnrts.dbf): '

CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'
  SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');

/*
```

手順 3 str1.net でのサプリメンタル・ロギングの指定

サプリメンタル・ロギングによって、表に対して行われた変更に関する追加情報が REDO ログに書き込まれます。適用プロセスでは、一意の行の識別や競合解消など、特定の操作を実行するために、この追加情報が必要になります。この環境で変更が取得されるのは `str1.net` データベースのみであるため、これが `hr.jobs` 表に対するサプリメンタル・ロギングを指定する必要のある唯一のデータベースです。

次の文では、`hr.jobs` 表内の主キー列について、無条件のサプリメンタル・ログ・グループを指定します。

関連項目：

- 2-10 ページ「Streams 環境内のサプリメンタル・ロギング」
- 12-8 ページ「ソース・データベースでのサプリメンタル・ロギングの指定」

```
*/

ALTER TABLE hr.jobs ADD SUPPLEMENTAL LOG GROUP log_group_jobs_pk
  (job_id) ALWAYS;

/*
```

手順 4 str1.net での伝播の構成

str1.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@str1.net

/*
```

str1.net のキューから str2.net のキューへの、hr.jobs 表に対する DML 変更および DDL 変更の伝播を構成してスケジュールします。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              => 'hr.jobs',
    streams_name            => 'str1_to_str2',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@str2.net',
    include_dml             =>  true,
    include_ddl             =>  true,
    source_database         => 'str1.net');
END;
/

/*
```

手順 5 str1.net での取得プロセスの構成

取得プロセスを構成して、str1.net での hr.jobs 表に対する変更を取得します。この手順では、この表に対する変更を、取得プロセスで取得し、指定のキューにエンキューするように指定します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name     => 'hr.jobs',
    streams_type   => 'capture',
    streams_name   => 'capture_simp',
    queue_name     => 'strmadmin.streams_queue',
    include_dml    =>  true,
    include_ddl    =>  true);
END;
/

/*
```

手順 6 str2.net での hr.jobs 表のインスタンス化

この例では、hr.jobs 表が str1.net データベースと str2.net データベースの両方に存在し、この表が両方のデータベースで同一であることを想定しています。この場合、str1.net で表のメタデータのみのエクスポートを実行して、作成されたエクスポート・ダンプ・ファイルを str2.net でインポートすることによって、str2.net データベースで表をインスタンス化できます。このメタデータのみのエクスポート / インポートを実行することで、str2.net で hr.jobs 表に対するインスタンス化 SCN が記録されます。インスタンス化 SCN は、適用プロセスにより str2.net で表への変更が適用される前に必要です。

異なるウィンドウを開き、str2.net でインスタンス化する hr.jobs 表を str1.net でエクスポートします。エクスポート・コマンドの実行時に、OBJECT_CONSISTENT エクスポート・パラメータが y に設定されていることと、ROWS エクスポート・パラメータが n に設定されていることを確認します。また、エクスポート中は、hr.jobs 表に対する DML 変更または DDL 変更が行われないことを確認してください。

次にエクスポート・コマンドの例を示します。

```
exp userid=hr/hr FILE=jobs_instant.dmp TABLES=jobs OBJECT_CONSISTENT=y ROWS=n
```

> **関連項目：**『Oracle9*i* データベース・ユーティリティ』でエクスポートの実行方法を参照してください。

```
*/

PAUSE Press <RETURN> to continue when the export is complete in the other window
that you opened.

/*
```

エクスポート・ダンプ・ファイル jobs_instant.dmp を接続先データベースに送信します。この例では、接続先データベースは str2.net です。

バイナリ FTP または他のなんらかの方法を使用して、エクスポート・ダンプ・ファイルを接続先データベースに送信できます。ファイルを送信するには、異なるウィンドウを開く必要がある場合があります。

```
*/

PAUSE Press <RETURN> to continue after transferring the dump file.

/*
```

別のウィンドウで、str2.net データベースが稼働しているコンピュータに接続し、エクスポート・ダンプ・ファイル jobs_instant.dmp をインポートして、str2.net データベースで jobs 表をインスタンス化します。str2.net が稼働しているコンピュータには、telnet またはリモート・ログインを使用して接続できます。

インポート・コマンドの実行時には、STREAMS_INSTANTIATION インポート・パラメータが y に設定されていることを確認してください。このパラメータによって、インポートされる各オブジェクトのエクスポート SCN 情報が、インポート時に確実に記録されます。

次にインポート・コマンドの例を示します。

```
imp userid=hr/hr FILE=jobs_instant.dmp IGNORE=y COMMIT=y LOG=import.log
STREAMS_INSTANTIATION=y
```

> **関連項目：**『Oracle9*i* データベース・ユーティリティ』でインポートの実行方法を参照してください。

```
*/

PAUSE Press <RETURN> to continue after the import is complete at str2.net.

/*
```

手順 7 str2.net での hr.jobs 表のサプリメンタル・ログ・グループの削除

hr.jobs 表を str2.net でインスタンス化した場合、表に対する str1.net からのサプリメンタル・ログ・グループは保持されています。str2.net では、この表に対する変更を取得する取得プロセスが存在しないため、このログ・グループは str2.net で必要ありません。ログ・グループを削除して、str2.net の REDO ログ内に余分な情報が書き込まれることを回避できます。str2.net に hr ユーザーとして接続します。

```
*/

CONNECT hr/hr@str2.net

/*
```

str2.net でサプリメンタル・ログ・グループを削除します。

```
*/

ALTER TABLE hr.jobs DROP SUPPLEMENTAL LOG GROUP log_group_jobs_pk;

/*
```

手順 8 str2.net での適用プロセスの構成

str2.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@str2.net

/*
```

str2.net を構成して、変更を hr.jobs 表に適用します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.jobs',
    streams_type    => 'apply',
    streams_name    => 'apply_simp',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'str1.net');
END;
/

/*
```

手順 9 str2.net での適用プロセスの起動

エラーが発生しても適用プロセスが無効化されないように、disable_on_error パラメータを n に設定して、str2.net で適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_simp',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply_simp');
END;
/

/*
```

手順 10 str1.net での取得プロセスの起動

str1.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@str1.net

/*
```

str1.net で取得プロセスを起動します。

```
*/

BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name  => 'capture_simp');
END;
/

/*
```

手順 11 スプール結果のチェック

このスクリプトの完了後に streams_share_jobs.out スプール・ファイルをチェックして、すべてのアクションが正常終了したことを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT *****************************/
```

hr.jobs 表に対する変更と結果の表示

次の手順に従って、str1.net で hr.jobs 表に対する DML 変更および DDL 変更を行い、変更が str1.net で取得され、str1.net から str2.net へ伝播され、str2.net で hr.jobs 表に適用されていることを確認します。

手順 1 str1.net での hr.jobs に対する変更

hr.jobs 表に対する次の変更を行います。

```
CONNECT hr/hr@str1.net

UPDATE hr.jobs SET max_salary=9545 WHERE job_id='PR_REP';
COMMIT;

ALTER TABLE hr.jobs ADD(duties VARCHAR2(4000));
```

手順 2 str2.net での hr.jobs 表の問合せおよび記述

前述の手順で実行した変更を取得、伝播し、適用するのに必要な時間が経過した後、次の問合せを実行して UPDATE 変更が str2.net に伝播されて適用されたことを確認します。

```
CONNECT hr/hr@str2.net

SELECT * FROM hr.jobs WHERE job_id='PR_REP';
```

max_salary 列の値は、9545 になっている必要があります。

次に、hr.jobs 表を記述して、ALTER 変更および TABLE 変更が str2.net に伝播され、適用されたことを確認します。

```
DESC hr.jobs
```

duties 列が追加されている必要があります。

22

単一ソースの異機種間レプリケーションの例

この章では、Streams を使用して構成できる単一ソースの異機種間レプリケーション環境、およびこの環境に新たなオブジェクトおよびデータベースを追加する場合に必要となるタスクの例を説明します。

この章の内容は次のとおりです。

簡単な単一のソース・レプリケーションの例の概要

この例では、Streams を使用して 4 つのデータベース間でデータをレプリケートする方法を説明します。データベースのうち 3 つが Oracle データベースで 1 つが Sybase データベースであるため、異機種間環境です。`dbs1.net` Oracle データベースで、`hr` スキーマ内の表に対して行われた DML 変更と DDL 変更が取得され、他の 2 つの Oracle データベースに伝播します。適用プロセスでは Oracle 以外のデータベースには DDL 変更を適用できないため、DML 変更のみが取得されて `dbs4.net` データベースに伝播します。`hr` スキーマに対する変更は、`dbs1.net` でのみ発生します。`hr` スキーマは、環境内の他のデータベースでは読取り専用です。

図 22-1 に、この環境の概要を示します。

図 22-1 単一のソース・データベースからのデータを共有する環境の例

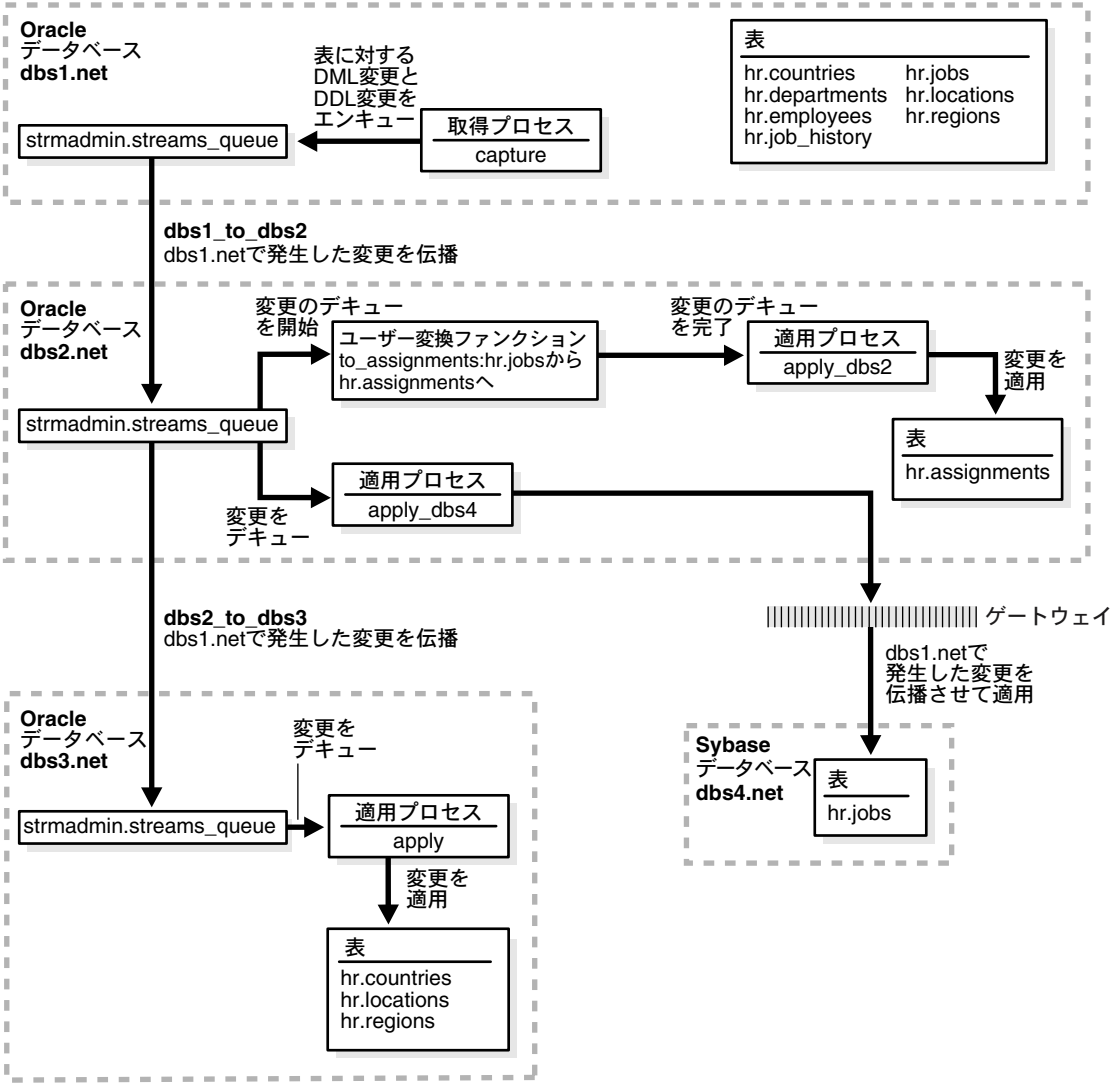


図 22-1 に示すように、dbs1.net の hr スキーマには次の表が含まれています。

- countries
- departments
- employees
- job_history
- jobs
- locations
- regions

この例では有向ネットワークを使用しています。これは、ソース・データベースで取得された変更が、1 つ以上の中間データベースを介して他のデータベースに伝播されることを意味します。ここでは、dbs1.net データベースから中間データベース dbs2.net を介して、dbs3.net データベースに変更が伝播します。また、dbs1.net データベースから dbs2.net データベースに変更が伝播され、そこで変更がゲートウェイを介して dbs4.net データベースに直接適用されます。

この環境内の一部のデータベースには、特定の表がありません。データベースが表の中間データベースではなく、その表が含まれていない場合、そのデータベースには表に対する変更を伝播する必要はありません。たとえば、departments、employees、job_history および jobs の各表は、dbs3.net には存在しません。したがって、これらの表に対する変更は dbs2.net から dbs3.net には伝播しません。

この例では、Streams を使用して次の一連のアクションが実行されます。

1. 取得プロセスは、dbs1.net データベースで hr スキーマ内のすべての表に対する DML 変更と DDL 変更を取得し、このデータベースのキューにエンキューします。この例では、7 つの表のうち 4 つに対する変更のみが接続先データベースに伝播しますが、22-57 ページの「既存の Streams レプリケーション環境へのオブジェクトの追加」を示す例では、hr スキーマ内の残りの表が接続先データベースに追加されます。
2. dbs1.net データベースからは、これらの変更がメッセージの形式で dbs2.net のキューに伝播します。
3. dbs2.net では、jobs 表に対する DML 変更が assignments 表（jobs の直接のマッピング）に対する DML 変更に変換されてから適用されます。hr スキーマ内の他の表に対する変更は、dbs2.net では適用されません。
4. dbs3.net のキューは、dbs1.net の countries、locations および regions 表で発生した変更を dbs2.net のキューから受信するため、これらの変更は dbs2.net から dbs3.net に伝播します。この構成は、有向ネットワークの一例です。
5. dbs3.net の適用プロセスは、countries、locations および regions の各表に変更を適用します。

6. dbs4.net は Sybase データベースであり、dbs1.net で発生した jobs 表に対する変更を dbs2.net のキューから受信するため、これらの変更はゲートウェイ経由で dbs4.net のデータベース・リンクを使用して dbs2.net からリモートで適用されます。この構成は、異機種間サポートの一例です。

前提条件

この章の例を開始する前に、前提条件となる次の作業を完了する必要があります。

- 環境内のすべてのデータベースについて、次の初期化パラメータを指定の値に設定します。
  - AQ_TM_PROCESSES: このパラメータでは、キュー・モニター・プロセスを設定します。値 1 ～ 10 では、メッセージの監視用に作成するキュー・モニター・プロセスの数を指定します。AQ_TM_PROCESSES を指定しないか、0 に設定すると、キュー・モニター・プロセスは作成されません。この例では、各データベースで AQ_TM_PROCESSES を 1 以上の値に設定する必要があります。

    このパラメータを 1 以上に設定すると、指定した数のキュー・モニター・プロセスが起動します。これらのキュー・モニター・プロセスは、遅延や期限切れなど、時間ベースのメッセージ操作の管理、指定した保存期間を過ぎて保存されているメッセージのクリーン・アップ、および保存期間が 0（ゼロ）の場合のコンシューム済みメッセージのクリーン・アップを受け持ちます。
  - GLOBAL_NAMES: Streams 環境の各データベースで、このパラメータを true に設定する必要があります。
  - JOB_QUEUE_PROCESSES: Streams 環境でイベントを伝播する各データベースで、このパラメータを 2 以上の値に設定する必要があります。同時に実行できるジョブの最大数に 1 を加えた値に設定してください。この例では、dbs1.net と dbs2.net がイベントを伝播します。そのため、この 2 つのデータベースでは JOB_QUEUE_PROCESSES を 2 以上の値に設定する必要があります。
  - COMPATIBLE: このパラメータは 9.2.0 以上に設定する必要があります。
  - LOG_PARALLELISM: このパラメータは、イベントを取得する各データベースで 1 に設定する必要があります。この例では、このパラメータを dbs1.net で 1 に設定します。

    関連項目： Streams 環境で重要な他の初期化パラメータについては、11-4 ページの「Streams に関連する初期化パラメータの設定」を参照してください。

- 取得される変更を生成するデータベースは、ARCHIVELOG モードで実行している必要があります。この例では、dbs1.net で変更が生成されるため、dbs1.net は ARCHIVELOG モードで実行する必要があります。

  関連項目： ARCHIVELOG モードでデータベースを実行する方法については、『Oracle9*i* データベース管理者ガイド』を参照してください。

- dbs2.net 上の Oracle ゲートウェイを、Sybase データベース dbs4.net と通信するように構成します。

  関連項目：『Oracle9*i* Heterogeneous Connectivity Administrator's Guide』

- Sybase データベース dbs4.net で hr ユーザーを設定します。

  関連項目： Sybase データベース内でユーザーと表を作成する方法については、Sybase マニュアルを参照してください。

- dbs4.net Sybase データベースで、dbs1.net Oracle データベースから hr.jobs 表をインスタンス化します。

  関連項目： 9-6 ページ「Oracle から Oracle 以外への環境でのインスタンス化」

- ネットワークと Oracle Net を、次のデータベースが相互に通信できるように構成します。
  - dbs1.net と dbs2.net
  - dbs2.net と dbs3.net
  - dbs2.net と dbs4.net

  関連項目：『Oracle9*i* Net Services 管理者ガイド』

- この例では、各データベースで新規ユーザーが Streams 管理者（strmadmin）として作成され、このユーザーのデータに使用する表領域の指定を求めるプロンプトが表示されます。この例を開始する前に、各データベースで Streams 管理者用に新規の表領域を作成するか、既存の表領域を識別してください。SYSTEM 表領域は、Streams 管理者用にしないでください。

ユーザーの設定とキューおよびデータベース・リンクの作成

次の手順に従って、Oracle データベース 3 つと Sybase データベース 1 つを含む Streams レプリケーション環境用に、ユーザーを設定してキューとデータベース・リンクを作成します。

1. 出力の表示と結果のスプーリング
2. dbs1.net での hr.countries 表の変更
3. dbs1.net でのユーザーの設定
4. dbs1.net での Streams キューの作成
5. dbs1.net でのデータベース・リンクの作成
6. dbs2.net でのユーザーの設定
7. dbs2.net での Streams キューの作成
8. dbs2.net でのデータベース・リンクの作成
9. dbs2.net での hr.assignments 表の作成
10. dbs3.net でのユーザーの設定
11. dbs3.net での Streams キューの作成
12. dbs3.net の hr スキーマにあるすべての表の削除
13. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 22-17 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************ BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_setup_single.out

/*
```

手順 2 dbs1.net での hr.countries 表の変更

dbs1.net に hr ユーザーとして接続します。

```
*/

CONNECT hr/hr@dbs1.net

/*
```

hr.countries 表を索引構成表から通常の表に変換します。現在、取得プロセスでは索引構成表に対する変更は取得できません。

```
*/

ALTER TABLE countries RENAME TO countries_orig;

CREATE TABLE hr.countries(
  country_id      CHAR(2) CONSTRAINT  country_id_nn_noiot NOT NULL,
  country_name    VARCHAR2(40),
  region_id       NUMBER,
  CONSTRAINT      country_c_id_pk_noiot  PRIMARY KEY (country_id));

ALTER TABLE hr.countries
ADD (CONSTRAINT countr_reg_fk_noiot
         FOREIGN KEY (region_id)
         REFERENCES regions(region_id)) ;

INSERT INTO hr.countries (SELECT * FROM hr.countries_orig);

DROP TABLE hr.countries_orig CASCADE CONSTRAINTS;
```

```
ALTER TABLE locations
  ADD (CONSTRAINT loc_c_id_fk
    FOREIGN KEY (country_id)
    REFERENCES countries(country_id));

/*
```

手順 3 dbs1.net でのユーザーの設定

dbs1.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@dbs1.net AS SYSDBA

/*
```

Streams 管理者 strmadmin を作成し、このユーザーに必要な権限を付与します。これらの権限を付与されたユーザーは、キューの管理、Streams に関連するパッケージ内のサブプログラムの実行、ルール・セットおよびルールの作成、データ・ディクショナリ・ビューとキュー表の問合せによる Streams 環境の監視を行うことができます。このユーザー用に、異なる名前を選択できます。

注意：

- セキュリティを確保するために、Streams 管理者には strmadminpw 以外のパスワードを使用してください。
- Streams 管理者には、SELECT_CATALOG_ROLE は不要です。この例で付与されているのは、Streams 管理者が環境を簡単に監視できるようにするためです。
- Oracle Enterprise Manager の Streams ツールを使用する予定の場合は、Streams 管理者にこの手順で示す権限の他に SELECT ANY DICTIONARY 権限を付与します。
- ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on dbs1.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
                     QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_AQADM            TO strmadmin;
GRANT EXECUTE ON DBMS_CAPTURE_ADM      TO strmadmin;
GRANT EXECUTE ON DBMS_FLASHBACK        TO strmadmin;
GRANT EXECUTE ON DBMS_PROPAGATION_ADM  TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM      TO strmadmin;

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

/*
```

手順 4 dbs1.net での Streams キューの作成

変更を取得するデータベースで Streams 管理者として接続します。この例では、データベース dbs1.net に接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs1.net

/*
```

SET_UP_QUEUE プロシージャを実行して、dbs1.net でキュー streams_queue を作成します。このキューは、取得され他のデータベースに伝播される変更を保持することで、Streams キューとして機能します。

SET_UP_QUEUE プロシージャを実行すると、次のアクションが実行されます。

- キュー表 streams_queue_table が作成されます。このキュー表の所有者は Streams 管理者（strmadmin）であり、このユーザーのデフォルト記憶域が使用されます。
- Streams 管理者（strmadmin）が所有するキュー streams_queue が作成されます。
- キューが起動されます。

```
*/

EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();

/*
```

手順 5 dbs1.net でのデータベース・リンクの作成

変更が取得されるデータベースから変更の伝播先となるデータベースへの、データベース・リンクを作成します。この例では、変更が取得されるデータベースは dbs1.net、これらの変更の伝播先となるデータベースは dbs2.net です。

```
*/

CREATE DATABASE LINK dbs2.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'dbs2.net';

/*
```

手順 6 dbs2.net でのユーザーの設定

dbs2.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@dbs2.net AS SYSDBA

/*
```

Streams 管理者 strmadmin を作成し、このユーザーに必要な権限を付与します。これらの権限を付与されたユーザーは、キューの管理、Streams に関連するパッケージ内のサブプログラムの実行、ルール・セットおよびルールの作成、データ・ディクショナリ・ビューとキュー表の問合せによる Streams 環境の監視を行うことができます。このユーザー用に、異なる名前を選択できます。

注意：

- セキュリティを確保するために、Streams 管理者には strmadminpw 以外のパスワードを使用してください。
- Streams 管理者には、SELECT_CATALOG_ROLE は不要です。この例で付与されているのは、Streams 管理者が環境を簡単に監視できるようにするためです。
- Oracle Enterprise Manager の Streams ツールを使用する予定の場合は、Streams 管理者にこの手順で示す権限の他に SELECT ANY DICTIONARY 権限を付与します。
- ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on dbs2.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
                     QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM        TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM            TO strmadmin;
```

```
GRANT EXECUTE ON DBMS_PROPAGATION_ADM  TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM      TO strmadmin;

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

/*
```

手順 7 dbs2.net での Streams キューの作成

dbs2.net で Streams 管理者として接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs2.net

/*
```

SET_UP_QUEUE プロシージャを実行して、dbs2.net でキュー streams_queue を作成します。このキューは、このデータベースで適用される変更と、他のデータベースに伝播される変更を保持することで、Streams キューとして機能します。

SET_UP_QUEUE プロシージャを実行すると、次のアクションが実行されます。

- キュー表 streams_queue_table が作成されます。このキュー表の所有者は Streams 管理者（strmadmin）であり、このユーザーのデフォルト記憶域が使用されます。
- Streams 管理者（strmadmin）が所有するキュー streams_queue が作成されます。
- キューが起動されます。

```
*/

EXEC  DBMS_STREAMS_ADM.SET_UP_QUEUE();

/*
```

手順 8 dbs2.net でのデータベース・リンクの作成

変更の伝播先となるデータベースへのデータベース・リンクを作成します。この例では、変更はデータベース dbs2.net から別の Oracle データベースである dbs3.net と、Sybase データベースである dbs4.net に伝播します。Sybase データベースへのデータベース・リンクは、Streams 管理者ではなく表の所有者に接続することに注意してください。このデータベース・リンクでは、dbs4.net データベースで、hr.jobs 表を変更する権限を付与されているすべてのユーザーに接続できます。

> **注意：** Sybase など、Oracle 以外の一部のデータベースでは、ユーザー名とパスワードの大 / 小文字が正しく指定されていることを確認する必要があります。したがって、Sybase データベースでは、ユーザー名とパスワードを二重引用符で囲んで指定します。

```
*/

CREATE DATABASE LINK dbs3.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'dbs3.net';

CREATE DATABASE LINK dbs4.net CONNECT TO "hr" IDENTIFIED BY "hrpass"
   USING 'dbs4.net';

/*
```

手順 9 dbs2.net での hr.assignments 表の作成

この例では、dbs1.net で hr.jobs 表に対する変更が、dbs2.net で hr.assignments 表に対する変更に変換される、ルールベースの変換を説明します。この例の変換部分が正常に機能するように、dbs2.net 上で hr.assignments 表を作成する必要があります。

dbs2.net に hr として接続します。

```
*/

CONNECT hr/hr@dbs2.net

/*
```

dbs2.net データベース上で hr.assignments 表を作成します。

```
*/

CREATE TABLE hr.assignments AS SELECT * FROM hr.jobs;

ALTER TABLE hr.assignments ADD PRIMARY KEY (job_id);

/*
```

手順 10 dbs3.net でのユーザーの設定

dbs3.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@dbs3.net AS SYSDBA

/*
```

Streams 管理者 strmadmin を作成し、このユーザーに必要な権限を付与します。これらの権限を付与されたユーザーは、キューの管理、Streams に関連するパッケージ内のサブプログラムの実行、ルール・セットおよびルールの作成、データ・ディクショナリ・ビューとキュー表の問合せによる Streams 環境の監視を行うことができます。このユーザー用に、異なる名前を選択できます。

注意：

- セキュリティを確保するために、Streams 管理者には strmadminpw 以外のパスワードを使用してください。
- Streams 管理者には、SELECT_CATALOG_ROLE は不要です。この例で付与されているのは、Streams 管理者が環境を簡単に監視できるようにするためです。
- Oracle Enterprise Manager の Streams ツールを使用する予定の場合は、Streams 管理者にこの手順で示す権限の他に SELECT ANY DICTIONARY 権限を付与します。
- ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on dbs3.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
                     QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM        TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM            TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM      TO strmadmin;
```

```
BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

/*
```

手順 11 dbs3.net での Streams キューの作成

dbs3.net で Streams 管理者として接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs3.net

/*
```

SET_UP_QUEUE プロシージャを実行して、dbs3.net でキュー streams_queue を作成します。このキューは、このデータベースで適用される変更を保持することで Streams キューとして機能します。

SET_UP_QUEUE プロシージャを実行すると、次のアクションが実行されます。

- キュー表 streams_queue_table が作成されます。このキュー表の所有者は Streams 管理者（strmadmin）であり、このユーザーのデフォルト記憶域が使用されます。
- Streams 管理者（strmadmin）が所有するキュー streams_queue が作成されます。
- キューが起動されます。

```
*/

EXEC  DBMS_STREAMS_ADM.SET_UP_QUEUE();

/*
```

手順 12 dbs3.net の hr スキーマにあるすべての表の削除

この例では、dbs1.net からエクスポートして dbs3.net にインポートすることで、hr スキーマ内の表をインスタンス化する方法を示します。この例のインスタンス化部分を正常に機能させるには、dbs3.net でこれらの表を削除する必要があります。

dbs3.net に hr として接続します。

```
*/

CONNECT hr/hr@dbs3.net

/*
```

dbs3.net データベースで hr スキーマ内の表をすべて削除します。

> **注意：** この手順に従って hr スキーマ内の表をすべて削除した後に、この例の残りの項の説明に従って dbs3.net で hr スキーマを再インスタンス化する必要があります。hr スキーマが Oracle データベースに存在しない場合は、Oracle マニュアル・セットに記載されている一部の例が失敗する可能性があります。

```
*/

DROP TABLE hr.countries CASCADE CONSTRAINTS;
DROP TABLE hr.departments CASCADE CONSTRAINTS;
DROP TABLE hr.employees CASCADE CONSTRAINTS;
DROP TABLE hr.job_history CASCADE CONSTRAINTS;
DROP TABLE hr.jobs CASCADE CONSTRAINTS;
DROP TABLE hr.locations CASCADE CONSTRAINTS;
DROP TABLE hr.regions CASCADE CONSTRAINTS;

/*
```

手順 13 スプール結果のチェック

このスクリプトの完了後に streams_setup_single.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT ******************************/
```

単一データベースからのデータを共有するためのスクリプトの例

この例では、Streams を使用して hr スキーマ内の表のレプリケーションを行う 2 つの方法について説明します。

- 環境を構成する簡単な方法の例については、22-18 ページの「単一データベースからのデータを共有する単純な構成」を参照してください。この例では、DBMS_STREAMS_ADM パッケージを使用して、取得プロセス、伝播、および適用プロセスと、それぞれに関連付けられたルール・セットを作成します。Streams 環境を構成するには、DBMS_STREAMS_ADM パッケージを使用するのが最も簡単な方法です。
- この環境のより柔軟な構成方法の例については、22-36 ページの「単一データベースからのデータを共有する柔軟な構成」を参照してください。この例では、DBMS_CAPTURE_ADM パッケージを使用して取得プロセスを作成し、DBMS_PROPAGATION_ADM パッケージを使用して伝播を作成し、DBMS_APPLY_ADM パッケージを作成して適用プロセスを作成します。また、この例では、DBMS_RULES_ADM パッケージを使用して、これらの取得プロセス、伝播および適用プロセスに関連付けられたルール・セットを作成して移入します。DBMS_STREAMS_ADM パッケージのかわりにこれらのパッケージを使用すると、より多数の構成オプションを使用して柔軟に構成できます。

注意： これらの例では、同じ Streams 環境について 2 つの構成方法を示します。したがって、特定の分散データベース・システムに対して、一方の例のみを実行してください。両方を実行すると、オブジェクトが存在することを示すエラーが発生します。

単一データベースからのデータを共有する単純な構成

次の手順に従い、主に DBMS_STREAMS_ADM パッケージを使用して、取得、伝播および適用の定義を指定します。

1. 出力の表示と結果のスプーリング
2. dbs1.net で LogMiner の表に使用する代替表領域の作成
3. dbs1.net でのサプリメンタル・ロギングの指定
4. dbs1.net での伝播の構成
5. dbs1.net での取得プロセスの構成
6. 他のデータベースでの既存の表のインスタンス化 SCN の設定
7. dbs3.net での dbs1.net 表のインスタンス化
8. dbs3.net でのサプリメンタル・ログ・グループの削除
9. dbs3.net での適用プロセスの構成

10. dbs3.net での適用プロセスの適用ユーザー hr の指定
11. hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与
12. dbs3.net での適用プロセスの起動
13. dbs2.net での伝播の構成
14. dbs2.net での行 LCR 用ルールベースの変換の作成
15. dbs2.net でローカルの適用に使用する適用プロセスの構成
16. dbs2.net での適用プロセスの適用ユーザー hr の指定
17. hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与
18. dbs2.net でローカルの適用に使用する適用プロセスの起動
19. dbs4.net での適用に使用する dbs2.net の適用プロセスの構成
20. dbs4.net での適用に使用する dbs2.net の適用プロセスの起動
21. dbs1.net での取得プロセスの起動
22. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 22-35 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************* BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_share_schema1.out

/*
```

手順 2 dbs1.net で LogMiner の表に使用する代替表領域の作成

デフォルトでは、LogMiner の表は SYSTEM 表領域にありますが、変更を取得するために取得プロセスを起動すると、SYSTEM 表領域にこれらの表に使用する領域が足りなくなる場合があります。したがって、LogMiner の表に使用する代替表領域を作成する必要があります。

関連項目： 2-18 ページ「LogMiner の表のための代替表領域」

dbs1.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@dbs1.net AS SYSDBA

/*
```

LogMiner の表に使用する代替表領域を作成します。

注意： 各 ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

```
*/

ACCEPT tspace_name DEFAULT 'logmnrts' PROMPT 'Enter the name of the tablespace (for
example, logmnrts): '

ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the datafile
directory (for example, /usr/oracle/dbs): '

ACCEPT db_file_name DEFAULT 'logmnrts.dbf' PROMPT 'Enter the name of the datafile
(for example, logmnrts.dbf): '

CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'
  SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');

/*
```

手順 3 dbs1.net でのサプリメンタル・ロギングの指定

サプリメンタル・ロギングによって、表に対して行われた変更に関する追加情報が REDO ログに書き込まれます。適用プロセスでは、一意の行の識別や競合解消など、特定の操作を実行するために、この追加情報が必要になります。この環境で変更が取得されるのは dbs1.net データベースのみのため、hr スキーマ内の表のサプリメンタル・ロギングを指定する必要があるのは、このデータベースのみです。

hr スキーマ内のすべての主キー列について、無条件のサプリメンタル・ログ・グループを指定します。

関連項目：

- 2-10 ページ「Streams 環境内のサプリメンタル・ロギング」
- 12-8 ページ「ソース・データベースでのサプリメンタル・ロギングの指定」

```
*/

ALTER TABLE hr.countries ADD SUPPLEMENTAL LOG GROUP log_group_countries_pk
  (country_id) ALWAYS;

ALTER TABLE hr.departments ADD SUPPLEMENTAL LOG GROUP log_group_departments_pk
  (department_id) ALWAYS;

ALTER TABLE hr.employees ADD SUPPLEMENTAL LOG GROUP log_group_employees_pk
  (employee_id) ALWAYS;

ALTER TABLE hr.jobs ADD SUPPLEMENTAL LOG GROUP log_group_jobs_pk
  (job_id) ALWAYS;

ALTER TABLE hr.job_history ADD SUPPLEMENTAL LOG GROUP log_group_job_history_pk
  (employee_id, start_date) ALWAYS;

ALTER TABLE hr.locations ADD SUPPLEMENTAL LOG GROUP log_group_locations_pk
  (location_id) ALWAYS;

ALTER TABLE hr.regions ADD SUPPLEMENTAL LOG GROUP log_group_regions_pk
  (region_id) ALWAYS;

/*
```

手順 4 dbs1.net での伝播の構成

dbs1.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs1.net

/*
```

hr スキーマ内の DML 変更と DDL 変更について、dbs1.net のキューから dbs2.net のキューへの伝播を構成してスケジュールします。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name             => 'hr',
    streams_name            => 'dbs1_to_dbs2',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@dbs2.net',
    include_dml             =>  true,
    include_ddl             =>  true,
    source_database         => 'dbs1.net');
END;
/

/*
```

手順 5 dbs1.net での取得プロセスの構成

dbs1.net で、hr スキーマ全体に対する変更を取得する取得プロセスを構成します。この手順では、指定したスキーマ内の表に対する変更を、取得プロセスで取得して指定のキューにエンキューするように指定します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name     => 'hr',
    streams_type    => 'capture',
    streams_name    => 'capture',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true);
END;
/

/*
```

手順 6 他のデータベースでの既存の表のインスタンス化 SCN の設定

この例では、hr.jobs 表がすでに dbs2.net および dbs4.net に存在します。dbs2.net では、この表の名前は assignments ですが、dbs1.net の jobs 表と同じ構成およびデータを持っています。また、この例では、dbs4.net は Sybase データベースです。Streams 環境内の他のすべての表は、エクスポート / インポートを使用して他のデータベースでインスタンス化されます。

hr.jobs 表はすでに dbs2.net と dbs4.net に存在するため、この例では dbs1.net で DBMS_FLASHBACK パッケージの GET_SYSTEM_CHANGE_NUMBER ファンクションを使用して、このデータベース用の現行の SCN を取得します。この SCN は、dbs2.net で DBMS_APPLY_ADM パッケージの SET_TABLE_INSTANTIATION_SCN プロシージャを実行するために使用されます。このプロシージャを、dbs2.net と dbs4.net で実行し、hr.jobs 表のインスタンス化 SCN を設定します。

SET_TABLE_INSTANTIATION_SCN プロシージャでは、ある表の LCR のうち、適用プロセスによって無視される LCR と適用される LCR が制御されます。ソース・データベースからの表に関する LCR のコミット SCN が、接続先データベースでその表のインスタンス化 SCN 以下であれば、接続先データベースの適用プロセスでは LCR が廃棄されます。それ以外の場合は、適用プロセスによって LCR が適用されます。

この例では、dbs2.net の両方の適用プロセスによって、この手順で取得した SCN より後にコミットされた SCN を持つ hr.jobs 表にトランザクションが適用されます。

注意： この例では、hr.jobs 表の内容が、この手順の完了時に dbs1.net、dbs2.net（表名は hr.assignments）および dbs4.net で一貫した状態になっていることを想定しています。一貫性を確保するために、この手順の実行中は各データベースで表のロックが必要になる場合があります。

```
*/

DECLARE
  iscn  NUMBER;          -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@DBS2.NET(
    source_object_name    => 'hr.jobs',
    source_database_name  => 'dbs1.net',
    instantiation_scn     => iscn);
```

```
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@DBS2.NET(
    source_object_name    => 'hr.jobs',
    source_database_name  => 'dbs1.net',
    instantiation_scn     => iscn,
    apply_database_link   => 'dbs4.net');
END;
/

/*
```

手順 7 dbs3.net での dbs1.net 表のインスタンス化

dbs1.net で異なるウィンドウを開き、dbs3.net でインスタンス化する表をエクスポートします。エクスポート・コマンドの実行時に、OBJECT_CONSISTENT エクスポート・パラメータが y に設定されていることを確認します。また、エクスポート中は、エクスポート対象のオブジェクトに対する DDL 変更が行われないことを確認してください。

次にエクスポート・コマンドの例を示します。

```
exp userid=hr/hr FILE=hr_instant1.dmp TABLES=countries,locations,regions
OBJECT_CONSISTENT=y
```

> **関連項目：**『Oracle9*i* データベース・ユーティリティ』でエクスポートの実行方法を参照してください。

```
*/

PAUSE Press <RETURN> to continue when the export is complete in the other window
that you opened.

/*
```

エクスポート・ダンプ・ファイル hr_instant1.dmp を接続先データベースに送信します。この例では、接続先データベースは dbs3.net です。

バイナリ FTP または他のなんらかの方法を使用して、エクスポート・ダンプ・ファイルを接続先データベースに送信できます。ファイルを送信するには、異なるウィンドウを開く必要がある場合があります。

```
*/

PAUSE Press <RETURN> to continue after transferring the dump file.

/*
```

別のウィンドウで、dbs3.net データベースが稼働しているコンピュータに接続し、エクスポート・ダンプ・ファイル hr_instant1.dmp をインポートして、dbs3.net データベース内の countries、locations および regions 表をインスタンス化します。dbs3.net が稼働しているコンピュータには、telnet またはリモート・ログインを使用して接続できます。

インポート・コマンドの実行時には、STREAMS_INSTANTIATION インポート・パラメータが y に設定されていることを確認してください。このパラメータによって、インポートされる各オブジェクトのエクスポート SCN 情報が、インポート時に確実に記録されます。

次にインポート・コマンドの例を示します。

```
imp userid=hr/hr FILE=hr_instant1.dmp IGNORE=y FULL=y COMMIT=y LOG=import.log
STREAMS_INSTANTIATION=y
```

> **関連項目：**『Oracle9*i* データベース・ユーティリティ』でインポートの実行方法を参照してください。

```
*/

PAUSE Press <RETURN> to continue after the import is complete at dbs3.net.

/*
```

手順 8 dbs3.net でのサプリメンタル・ログ・グループの削除

hr スキーマを dbs3.net でインスタンス化した場合、dbs1.net からのサプリメンタル・ログ・グループは保持されています。dbs3.net では、hr スキーマの表に対する変更を取得する取得プロセスが存在しないため、これらのログ・グループは dbs3.net で必要ありません。ログ・グループを削除して、dbs3.net の REDO ログ内に余分な情報が書き込まれることを回避できます。

dbs3.net に hr ユーザーとして接続します。

```
*/

CONNECT hr/hr@dbs3.net

/*
```

dbs3.net でサプリメンタル・ログ・グループを削除します。

```
*/

ALTER TABLE hr.countries DROP SUPPLEMENTAL LOG GROUP log_group_countries_pk;

ALTER TABLE hr.locations DROP SUPPLEMENTAL LOG GROUP log_group_locations_pk;

ALTER TABLE hr.regions DROP SUPPLEMENTAL LOG GROUP log_group_regions_pk;

/*
```

手順 9 dbs3.net での適用プロセスの構成

dbs3.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs3.net

/*
```

dbs3.net を、countries 表、locations 表および regions 表に変更を適用するように構成します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.countries',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'dbs1.net');
END;
/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.locations',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'dbs1.net');
END;
/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.regions',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
```

```
    include_ddl     =>  true,
    source_database => 'dbs1.net');
END;
/

/*
```

手順 10 dbs3.net での適用プロセスの適用ユーザー hr の指定

この例では、このデータベースで適用プロセスによって変更が適用される全データベース・オブジェクトを、hr ユーザーが所有しているものとします。したがって、hr はすでにこれらのデータベース・オブジェクトを変更するための権限を持っており、hr を適用ユーザーにすると便利です。

前述の手順で適用プロセスを作成するときには、Streams 管理者 strmadmin が適用プロセスの作成プロシージャを実行したため、デフォルトで strmadmin が適用ユーザーとして指定されました。hr を適用ユーザーとして指定するかわりに、strmadmin を適用ユーザーのままにすることもできます。ただし、その場合は strmadmin に、変更が適用される全データベース・オブジェクトに対する権限と、適用プロセスで使用される全ユーザー・プロシージャを実行する権限を付与する必要があります。適用プロセスによって複数のスキーマ内のデータベース・オブジェクトに変更が適用される環境では、Streams 管理者を適用ユーザーとして使用する方が便利な場合があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'apply',
    apply_user => 'hr');
END;
/

/*
```

手順 11 hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与

前述の手順で hr ユーザーを適用ユーザーとして指定したため、hr ユーザーには適用プロセスで使用されるルール・セットの実行権限が必要です。

```
*/

DECLARE
   rs_name  VARCHAR2(64);   -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
    INTO rs_name
    FROM DBA_APPLY
    WHERE APPLY_NAME='APPLY';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee     => 'hr');
END;
/

/*
```

手順 12 dbs3.net での適用プロセスの起動

エラーが発生しても適用プロセスが無効化されないように、disable_on_error パラメータを n に設定して、dbs3.net で適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply');
END;
/

/*
```

手順 13 dbs2.net での伝播の構成

dbs2.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs2.net

/*
```

dbs2.net のキューから dbs3.net のキューへの伝播を構成してスケジュールします。dbs3.net で変更を適用する表ごとに、この伝播を指定する必要があります。この構成は、dbs2.net での変更が dbs1.net で発生しているため、有向ネットワークの一例です。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name               => 'hr.countries',
    streams_name             => 'dbs2_to_dbs3',
    source_queue_name        => 'strmadmin.streams_queue',
    destination_queue_name   => 'strmadmin.streams_queue@dbs3.net',
    include_dml              =>  true,
    include_ddl              =>  true,
    source_database          => 'dbs1.net');
END;
/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              => 'hr.locations',
    streams_name            => 'dbs2_to_dbs3',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@dbs3.net',
    include_dml             =>  true,
    include_ddl             =>  true,
    source_database         => 'dbs1.net');
END;
/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              => 'hr.regions',
    streams_name            => 'dbs2_to_dbs3',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@dbs3.net',
    include_dml             =>  true,
    include_ddl             =>  true,
    source_database         => 'dbs1.net');
```

```
END;
/

/*
```

手順 14 dbs2.net での行 LCR 用ルールベースの変換の作成

dbs2.net に hr ユーザーとして接続します。

```
*/

CONNECT hr/hr@dbs2.net

/*
```

dbs1.net で jobs 表に対する DML 文によって発生した行の変更が、dbs2.net の assignments 表に対する行の変更に変換されるように、ルールベースの変換ファンクションを作成します。

次のファンクションでは、jobs 表の各行 LCR が assignments 表の行 LCR に変換されます。

> **注意：** assignments 表に DDL 変更も適用されている場合は、DDL LCR 用に別の変換が必要になります。この変換では、オブジェクト名と DDL テキストを変更する必要があります。

```
*/

CREATE OR REPLACE FUNCTION hr.to_assignments_trans_dml(
  p_in_data in SYS.AnyData)
  RETURN SYS.AnyData IS out_data SYS.LCR$_ROW_RECORD;
  tc   pls_integer;
BEGIN
  -- Typecast AnyData to LCR$_ROW_RECORD
     tc := p_in_data.GetObject(out_data);
     IF out_data.GET_OBJECT_NAME() = 'JOBS'
     THEN
  -- Transform the in_data into the out_data
     out_data.SET_OBJECT_NAME('ASSIGNMENTS');
     END IF;
  -- Convert to AnyData
     RETURN SYS.AnyData.ConvertObject(out_data);
END;
/

/*
```

手順 15 dbs2.net でローカルの適用に使用する適用プロセスの構成

dbs2.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs2.net

/*
```

dbs2.net を、assignments 表に変更を適用するように構成します。assignments 表は、dbs1.net の jobs 表から変更を受信することに注意してください。

```
*/

DECLARE
  to_assignments_rulename_dml   VARCHAR2(30);
  dummy_rule                    VARCHAR2(30);
  action_ctx_dml                SYS.RE$NV_LIST;
  ac_name                       VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
BEGIN
--  DML changes to the jobs table from dbs1.net are applied to the assignments
--  table. The to_assignments_rulename_dml variable is an out parameter
--  in this call.
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.jobs', -- jobs, not assignments, specified
    streams_type    => 'apply',
    streams_name    => 'apply_dbs2',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     => true,
    include_ddl     => false,
    source_database => 'dbs1.net',
    dml_rule_name   => to_assignments_rulename_dml,
    ddl_rule_name   => dummy_rule);
-- Specify the name-value pair in the action context
  action_ctx_dml := SYS.RE$NV_LIST(SYS.RE$NV_ARRAY());
  action_ctx_dml.ADD_PAIR(
    ac_name,
    SYS.ANYDATA.CONVERTVARCHAR2('hr.to_assignments_trans_dml'));
--  Modify the rule for jobs to use the transformation.
  DBMS_RULE_ADM.ALTER_RULE(
    rule_name        => to_assignments_rulename_dml,
    action_context   => action_ctx_dml);
END;
/

/*
```

手順 16 dbs2.net での適用プロセスの適用ユーザー hr の指定

この例では、このデータベースで適用プロセスによって変更が適用される全データベース・オブジェクトを、hr ユーザーが所有しているものとします。したがって、hr はすでにこれらのデータベース・オブジェクトを変更するための権限を持っており、hr を適用ユーザーにすると便利です。

前述の手順で適用プロセスを作成するときには、Streams 管理者 strmadmin が適用プロセスの作成プロシージャを実行したため、デフォルトで strmadmin が適用ユーザーとして指定されました。hr を適用ユーザーとして指定するかわりに、strmadmin を適用ユーザーのままにすることもできます。ただし、その場合は strmadmin に、変更が適用される全データベース・オブジェクトに対する権限と、適用プロセスで使用される全ユーザー・プロシージャを実行する権限を付与する必要があります。適用プロセスによって複数のスキーマ内のデータベース・オブジェクトに変更が適用される環境では、Streams 管理者を適用ユーザーとして使用する方が便利な場合があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'apply_dbs2',
    apply_user => 'hr');
END;
/

/*
```

手順 17 hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与

前述の手順で hr ユーザーを適用ユーザーとして指定したため、hr ユーザーには適用プロセスで使用されるルール・セットの実行権限が必要です。

```
*/

DECLARE
   rs_name  VARCHAR2(64);   -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
    INTO rs_name
    FROM DBA_APPLY
    WHERE APPLY_NAME='APPLY_DBS2';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee     => 'hr');
```

```
END;
/

/*
```

手順 18 dbs2.net でローカルの適用に使用する適用プロセスの起動

エラーが発生しても適用プロセスが無効化されないように、disable_on_error パラメータを n に設定して、dbs2.net でローカル適用のための適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_dbs2',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply_dbs2');
END;
/

/*
```

手順 19 dbs4.net での適用に使用する dbs2.net の適用プロセスの構成

Sybase データベース dbs4.net 用の適用プロセスを構成します。dbs2.net データベースは、dbs4.net へのゲートウェイとして機能します。したがって、dbs4.net 用の適用プロセスは dbs2.net で構成する必要があります。適用プロセスで DDL 変更を Oracle 以外のデータベースに適用することはできません。そのため、ADD_TABLE_RULES プロシージャの実行時には、include_ddl パラメータが false に設定されます。

```
*/

BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name          => 'strmadmin.streams_queue',
    apply_name          => 'apply_dbs4',
    apply_database_link => 'dbs4.net',
    apply_captured      => true);
END;
/
```

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.jobs',
    streams_type    => 'apply',
    streams_name    => 'apply_dbs4',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  false,
    source_database => 'dbs1.net');
END;
/

/*
```

手順 20 dbs4.net での適用に使用する dbs2.net の適用プロセスの起動

エラーが発生しても適用プロセスが無効化されないように、disable_on_error パラメータを n に設定し、データベース・リンク dbs4.net を使用して Sybase 用のリモート適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_dbs4',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply_dbs4');
END;
/

/*
```

手順 21 dbs1.net での取得プロセスの起動

dbs1.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs1.net

/*
```

dbs1.net で取得プロセスを起動します。

```
*/

BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name  => 'capture');
END;
/

/*
```

手順 22 スプール結果のチェック

このスクリプトの完了後に streams_share_schema1.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/*
```

ここで、dbs1.net で特定の表に対する DML 変更および DDL 変更を行い、これらの変更が、この環境での Streams プロセスおよび伝播用に構成したルールに基づいて環境内の他のデータベースにレプリケートされていることを確認できます。

> **関連項目：** この環境でレプリケートされた変更の例については、22-56 ページの「hr スキーマでの表に対する DML 変更および DDL 変更」を参照してください。

```
/************************** END OF SCRIPT *****************************/
```

単一データベースからのデータを共有する柔軟な構成

次の手順に従い、より柔軟なアプローチを使用して取得、伝播および適用の定義を指定します。このアプローチでは、DBMS_STREAMS_ADM パッケージは使用しません。かわりに次のパッケージを使用します。

- DBMS_CAPTURE_ADM パッケージ。取得プロセスを構成します。
- DBMS_PROPAGATION_ADM パッケージ。伝播を構成します。
- DBMS_APPLY_ADM パッケージ。適用プロセスを構成します。
- DBMS_RULES_ADM パッケージ。取得、伝播および適用のルールとルール・セットを指定します。

> **注意：** この例で作成するルールでは、ALL_STREAMS_TABLE_RULES および DBA_STREAMS_TABLE_RULES データ・ディクショナリ・ビューには移入されません。この例で作成するルールを表示するには、ALL_RULES、DBA_RULES または USER_RULES データ・ディクショナリ・ビューを問い合せる必要があります。

この例に含まれる手順は、次のとおりです。

1. 出力の表示と結果のスプーリング
2. dbs1.net で LogMiner の表に使用する代替表領域の作成
3. dbs1.net でのサプリメンタル・ロギングの指定
4. dbs1.net での伝播の構成
5. dbs1.net での取得プロセスの構成
6. dbs1.net の hr スキーマのインスタンス化の準備
7. 他のデータベースでの既存の表のインスタンス化 SCN の設定
8. dbs3.net での dbs1.net の表のインスタンス化
9. dbs3.net でのサプリメンタル・ログ・グループの削除
10. dbs3.net での適用プロセスの構成
11. hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与
12. dbs3.net での適用プロセスの起動
13. dbs2.net での伝播の構成
14. dbs2.net での行 LCR 用ルールベースの変換の作成
15. dbs2.net でローカルの適用に使用する適用プロセスの構成

16. hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与
17. dbs2.net でローカルの適用に使用する適用プロセスの起動
18. dbs4.net での適用に使用する dbs2.net の適用プロセスの構成
19. dbs4.net での適用に使用する dbs2.net の適用プロセスの起動
20. dbs1.net での取得プロセスの起動
21. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 22-55 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************* BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_share_schema2.out

/*
```

手順 2 dbs1.net で LogMiner の表に使用する代替表領域の作成

デフォルトでは、LogMiner の表は SYSTEM 表領域にありますが、変更を取得するために取得プロセスが起動すると、SYSTEM 表領域にこれらの表に使用する領域が足りなくなる場合があります。したがって、LogMiner の表に使用する代替表領域を作成する必要があります。

関連項目： 2-18 ページ「LogMiner の表のための代替表領域」

dbs1.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@dbs1.net AS SYSDBA

/*
```

LogMiner の表に使用する代替表領域を作成します。

> **注意：** 各 ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

```
*/

ACCEPT tspace_name DEFAULT 'logmnrts' PROMPT 'Enter the name of the tablespace (for
example, logmnrts): '

ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the datafile
directory (for example, /usr/oracle/dbs): '

ACCEPT db_file_name DEFAULT 'logmnrts.dbf' PROMPT 'Enter the name of the datafile
(for example, logmnrts.dbf): '

CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'
  SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');

/*
```

手順 3 dbs1.net でのサプリメンタル・ロギングの指定

サプリメンタル・ロギングによって、表に対して行われた変更に関する追加情報が REDO ログに書き込まれます。適用プロセスでは、一意の行の識別や競合解消など、特定の操作を実行するために、この追加情報が必要になります。この環境で変更が取得されるのは dbs1.net データベースのみのため、hr スキーマ内の表のサプリメンタル・ロギングを指定する必要があるのは、このデータベースのみです。

hr スキーマ内のすべての主キー列について、無条件のサプリメンタル・ログ・グループを指定します。

関連項目：

- 2-10 ページ「Streams 環境内のサプリメンタル・ロギング」
- 12-8 ページ「ソース・データベースでのサプリメンタル・ロギングの指定」

```
*/

ALTER TABLE hr.countries ADD SUPPLEMENTAL LOG GROUP log_group_countries_pk
  (country_id) ALWAYS;

ALTER TABLE hr.departments ADD SUPPLEMENTAL LOG GROUP log_group_departments_pk
  (department_id) ALWAYS;

ALTER TABLE hr.employees ADD SUPPLEMENTAL LOG GROUP log_group_employees_pk
  (employee_id) ALWAYS;

ALTER TABLE hr.jobs ADD SUPPLEMENTAL LOG GROUP log_group_jobs_pk
  (job_id) ALWAYS;

ALTER TABLE hr.job_history ADD SUPPLEMENTAL LOG GROUP log_group_job_history_pk
  (employee_id, start_date) ALWAYS;

ALTER TABLE hr.locations ADD SUPPLEMENTAL LOG GROUP log_group_locations_pk
  (location_id) ALWAYS;

ALTER TABLE hr.regions ADD SUPPLEMENTAL LOG GROUP log_group_regions_pk
  (region_id) ALWAYS;

/*
```

手順 4 dbs1.net での伝播の構成

dbs1.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs1.net

/*
```

dbs1.net のキューから dbs2.net のキューへの伝播を構成してスケジュールします。この構成では、hr スキーマに対するすべての変更を伝播するように指定します。ルール・セットの指定を省略するオプションがありますが、その場合はキューにあるすべてが伝播されます。これは、後で複数の取得プロセスによって streams_queue が使用される場合には、望ましくないことがあります。

```
*/

BEGIN
  -- Create the rule set
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name       => 'strmadmin.propagation_dbs1_rules',
    evaluation_context  => 'SYS.STREAMS$_EVALUATION_CONTEXT');
  -- Create rules for all modifications to the hr schema
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name  => 'strmadmin.all_hr_dml',
    condition  => ' :dml.get_object_owner() = ''HR'' AND ' ||
                  ' :dml.is_null_tag() = ''Y'' AND ' ||
                  ' :dml.get_source_database_name() = ''DBS1.NET'' ');
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name  => 'strmadmin.all_hr_ddl',
    condition  => ' :ddl.get_object_owner() = ''HR''  AND ' ||
                  ' :ddl.is_null_tag() = ''Y'' AND ' ||
                  ' :ddl.get_source_database_name() = ''DBS1.NET'' ');
  -- Add rules to rule set
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_hr_dml',
    rule_set_name  => 'strmadmin.propagation_dbs1_rules');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_hr_ddl',
    rule_set_name  => 'strmadmin.propagation_dbs1_rules');
  -- Create the propagation
  DBMS_PROPAGATION_ADM.CREATE_PROPAGATION(
    propagation_name    => 'dbs1_to_dbs2',
    source_queue        => 'strmadmin.streams_queue',
    destination_queue   => 'strmadmin.streams_queue',
    destination_dblink  => 'dbs2.net',
    rule_set_name       => 'strmadmin.propagation_dbs1_rules');
END;
/

/*
```

手順 5 dbs1.net での取得プロセスの構成

dbs1.net で hr スキーマ全体を取得するように、取得プロセスとルールを作成します。

```
*/

BEGIN
  -- Create the rule set
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name       => 'strmadmin.demo_rules',
    evaluation_context  => 'SYS.STREAMS$_EVALUATION_CONTEXT');
  --   Create rules that specify the entire hr schema
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name  => 'strmadmin.schema_hr_dml',
    condition  => ' :dml.get_object_owner() = ''HR''  AND ' ||
                  ' :dml.is_null_tag() = ''Y'' AND ' ||
                  ' :dml.get_source_database_name() = ''DBS1.NET'' ');
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name  => 'strmadmin.schema_hr_ddl',
    condition  => ' :ddl.get_object_owner() = ''HR''  AND ' ||
                  ' :ddl.is_null_tag() = ''Y'' AND ' ||
                  ' :ddl.get_source_database_name() = ''DBS1.NET'' ');
  --  Add the rules to the rule set
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.schema_hr_dml',
    rule_set_name  => 'strmadmin.demo_rules');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.schema_hr_ddl',
    rule_set_name  => 'strmadmin.demo_rules');
  --  Create a capture process that uses the rule set
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name    => 'strmadmin.streams_queue',
    capture_name  => 'capture',
    rule_set_name => 'strmadmin.demo_rules');
END;
/

/*
```

手順 6 dbs1.net の hr スキーマのインスタンス化の準備

dbs1.net に Streams 管理者として接続したままで、dbs1.net の hr スキーマを dbs3.net でインスタンス化できるように準備します。この手順では、スキーマにある表の最小 SCN にインスタンス化のマークを付けます。最小 SCN 以降の SCN は、インスタンス化に使用できます。

> **注意：** この手順は、22-18 ページの「単一データベースからのデータを共有する単純な構成」では不要です。前述の例では、手順 5 で DBMS_STREAMS_ADM パッケージの ADD_SCHEMA_RULES プロシージャを実行すると、hr スキーマに対して DBMS_CAPTURE_ADM パッケージの PREPARE_SCHEMA_INSTANTIATION プロシージャが自動的に実行されました。

```
*/

BEGIN
  DBMS_CAPTURE_ADM.PREPARE_SCHEMA_INSTANTIATION(
    schema_name    => 'hr');
END;
/

/*
```

手順 7 他のデータベースでの既存の表のインスタンス化 SCN の設定

この例では、hr.jobs 表がすでに dbs2.net および dbs4.net に存在します。dbs2.net では、この表の名前は assignments ですが、dbs1.net の jobs 表と同じ shape およびデータを持っています。また、この例では、dbs4.net は Sybase データベースです。Streams 環境内の他のすべての表は、エクスポート / インポートを使用して他のデータベースでインスタンス化されます。

hr.jobs 表はすでに dbs2.net と dbs4.net に存在するため、この例では dbs1.net で DBMS_FLASHBACK パッケージの GET_SYSTEM_CHANGE_NUMBER ファンクションを使用して、このデータベース用の現行の SCN を取得します。この SCN は、dbs2.net で DBMS_APPLY_ADM パッケージの SET_TABLE_INSTANTIATION_SCN プロシージャを実行するために使用されます。このプロシージャを、dbs2.net と dbs4.net で実行し、hr.jobs 表のインスタンス化 SCN を設定します。

SET_TABLE_INSTANTIATION_SCN プロシージャでは、ある表の LCR のうち、適用プロセスによって無視される LCR と適用される LCR が制御されます。ソース・データベースからの表に関する LCR のコミット SCN が、接続先データベースでその表のインスタンス化 SCN 以下であれば、接続先データベースの適用プロセスでは LCR が廃棄されます。それ以外の場合は、適用プロセスによって LCR が適用されます。

この例では、dbs2.net の両方の適用プロセスによって、この手順で取得した SCN より後にコミットされた SCN を持つ hr.jobs 表にトランザクションが適用されます。

注意： この例では、hr.jobs 表の内容が、この手順の完了時に dbs1.net、dbs2.net（表名は hr.assignments）および dbs4.net で一貫した状態になっていることを想定しています。一貫性を確保するために、この手順の実行中は各データベースで表のロックが必要になる場合があります。

```
*/

DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@DBS2.NET(
    source_object_name    => 'hr.jobs',
    source_database_name  => 'dbs1.net',
    instantiation_scn     => iscn);
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@DBS2.NET(
    source_object_name    => 'hr.jobs',
    source_database_name  => 'dbs1.net',
    instantiation_scn     => iscn,
    apply_database_link   => 'dbs4.net');
END;
/

/*
```

手順 8 dbs3.net での dbs1.net の表のインスタンス化

dbs1.net で異なるウィンドウを開き、dbs3.net でインスタンス化する表をエクスポートします。エクスポート・コマンドの実行時に、OBJECT_CONSISTENT エクスポート・パラメータが y に設定されていることを確認します。また、エクスポート中は、エクスポート対象のオブジェクトに対する DDL 変更が行われないように注意してください。

次にエクスポート・コマンドの例を示します。

```
exp userid=hr/hr FILE=hr_instant1.dmp TABLES=countries,locations,regions
OBJECT_CONSISTENT=y
```

関連項目： 『Oracle9*i* データベース・ユーティリティ』でエクスポートの実行方法を参照してください。

```
*/

PAUSE Press <RETURN> to continue when the export is complete in the other window
that you opened.

/*
```

エクスポート・ダンプ・ファイル hr_instant1.dmp を接続先データベースに送信します。この例では、接続先データベースは dbs3.net です。

バイナリ FTP または他のなんらかの方法を使用して、エクスポート・ダンプ・ファイルを接続先データベースに送信できます。ファイルを送信するには、異なるウィンドウを開く必要がある場合があります。

```
*/

PAUSE Press <RETURN> to continue after transferring the dump file.

/*
```

別のウィンドウで、dbs3.net データベースが稼働しているコンピュータに接続し、エクスポート・ダンプ・ファイル hr_instant1.dmp をインポートして、dbs3.net データベース内の countries、locations および regions 表をインスタンス化します。dbs3.net が稼働しているコンピュータには、telnet またはリモート・ログインを使用して接続できます。

インポート・コマンドの実行時には、STREAMS_INSTANTIATION インポート・パラメータが y に設定されていることを確認してください。このパラメータによって、インポートされる各オブジェクトのエクスポート SCN 情報が、インポート時に確実に記録されます。

次にインポート・コマンドの例を示します。

```
imp userid=hr/hr FILE=hr_instant1.dmp IGNORE=y FULL=y COMMIT=y LOG=import.log
STREAMS_INSTANTIATION=y
```

> **関連項目：**『Oracle9*i* データベース・ユーティリティ』でインポートの実行方法を参照してください。

```
*/

PAUSE Press <RETURN> to continue after the import is complete at dbs3.net.

/*
```

手順 9 dbs3.net でのサプリメンタル・ログ・グループの削除

hr スキーマを dbs3.net でインスタンス化した場合、dbs1.net からのサプリメンタル・ログ・グループは保持されています。dbs3.net では、hr スキーマの表に対する変更を取得する取得プロセスが存在しないため、これらのログ・グループは dbs3.net で必要ありません。ログ・グループを削除して、dbs3.net の REDO ログ内の誤った情報を回避できます。

dbs3.net に hr ユーザーとして接続します。

```
*/

CONNECT hr/hr@dbs3.net

/*
```

dbs3.net でサプリメンタル・ログ・グループを削除します。

```
*/

ALTER TABLE hr.countries DROP SUPPLEMENTAL LOG GROUP log_group_countries_pk;

ALTER TABLE hr.locations DROP SUPPLEMENTAL LOG GROUP log_group_locations_pk;

ALTER TABLE hr.regions DROP SUPPLEMENTAL LOG GROUP log_group_regions_pk;

/*
```

手順 10 dbs3.net での適用プロセスの構成

dbs3.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs3.net

/*
```

dbs3.net を、countries 表、locations 表および regions 表に DML 変更と DDL 変更を適用するように構成します。

```
*/

BEGIN
  -- Create the rule set
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name       => 'strmadmin.apply_rules',
    evaluation_context  => 'SYS.STREAMS$_EVALUATION_CONTEXT');
  -- Rules for hr.countries
```

```
DBMS_RULE_ADM.CREATE_RULE(
  rule_name    => 'strmadmin.all_countries_dml',
  condition    => ' :dml.get_object_owner() = ''HR'' AND ' ||
                  ' :dml.get_object_name() = ''COUNTRIES''  AND ' ||
                  ' :dml.is_null_tag() = ''Y'' AND ' ||
                  ' :dml.get_source_database_name() = ''DBS1.NET'' ');
DBMS_RULE_ADM.CREATE_RULE(
  rule_name    => 'strmadmin.all_countries_ddl',
  condition    => ' :ddl.get_object_owner() = ''HR'' AND ' ||
                  ' :ddl.get_object_name() = ''COUNTRIES'' AND ' ||
                  ' :ddl.is_null_tag() = ''Y'' AND ' ||
                  ' :ddl.get_source_database_name() = ''DBS1.NET'' ');
-- Rules for hr.locations
DBMS_RULE_ADM.CREATE_RULE(
  rule_name    => 'strmadmin.all_locations_dml',
  condition    => ' :dml.get_object_owner() = ''HR'' AND ' ||
                  ' :dml.get_object_name() = ''LOCATIONS'' AND ' ||
                  ' :dml.is_null_tag() = ''Y'' AND ' ||
                  ' :dml.get_source_database_name() = ''DBS1.NET'' ');
DBMS_RULE_ADM.CREATE_RULE(
  rule_name    => 'strmadmin.all_locations_ddl',
  condition    => ' :ddl.get_object_owner() = ''HR'' AND ' ||
                  ' :ddl.get_object_name() = ''LOCATIONS'' AND ' ||
                  ' :ddl.is_null_tag() = ''Y'' AND ' ||
                  ' :ddl.get_source_database_name() = ''DBS1.NET'' ');
-- Rules for hr.regions
DBMS_RULE_ADM.CREATE_RULE(
  rule_name    => 'strmadmin.all_regions_dml',
  condition    => ' :dml.get_object_owner() = ''HR'' AND ' ||
                  ' :dml.get_object_name() = ''REGIONS'' AND ' ||
                  ' :dml.is_null_tag() = ''Y'' AND ' ||
                  ' :dml.get_source_database_name() = ''DBS1.NET'' ');
DBMS_RULE_ADM.CREATE_RULE(
  rule_name    => 'strmadmin.all_regions_ddl',
  condition    => ' :ddl.get_object_owner() = ''HR'' AND ' ||
                  ' :ddl.get_object_name() = ''REGIONS'' AND ' ||
                  ' :ddl.is_null_tag() = ''Y'' AND ' ||
                  ' :ddl.get_source_database_name() = ''DBS1.NET'' ');
-- Add rules to rule set
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_countries_dml',
  rule_set_name  => 'strmadmin.apply_rules');
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_countries_ddl',
  rule_set_name  => 'strmadmin.apply_rules');
```

```
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_locations_dml',
    rule_set_name  => 'strmadmin.apply_rules');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_locations_ddl',
    rule_set_name  => 'strmadmin.apply_rules');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_regions_dml',
    rule_set_name  => 'strmadmin.apply_rules');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_regions_ddl',
    rule_set_name  => 'strmadmin.apply_rules');
  -- Create the apply process
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name     => 'strmadmin.streams_queue',
    apply_name     => 'apply',
    rule_set_name  => 'strmadmin.apply_rules',
    apply_user     => 'hr',
    apply_captured => true);
END;
/

/*
```

手順 11 hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与

前述の手順で hr ユーザーを適用ユーザーとして指定したため、hr ユーザーには適用プロセスで使用されるルール・セットの実行権限が必要です。

```
*/

BEGIN
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => 'strmadmin.apply_rules',
    grantee     => 'hr');
END;
/

/*
```

手順 12 dbs3.net での適用プロセスの起動

エラーが発生しても適用プロセスが無効化されないように、disable_on_error パラメータを n に設定して、dbs3.net で適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply');
END;
/

/*
```

手順 13 dbs2.net での伝播の構成

dbs2.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs2.net

/*
```

dbs2.net のキューから dbs3.net のキューへの伝播を構成してスケジュールします。この構成は、dbs2.net での変更が dbs1.net で発生しているため、有向ネットワークの一例です。

```
*/

BEGIN
  -- Create the rule set
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name       => 'strmadmin.propagation_dbs3_rules',
    evaluation_context  => 'SYS.STREAMS$_EVALUATION_CONTEXT');
  -- Create rules for all modifications to the countries table
```

```
DBMS_RULE_ADM.CREATE_RULE(
  rule_name  => 'strmadmin.all_countries_dml',
  condition  => ' :dml.get_object_owner() = ''HR'' AND ' ||
                ' :dml.get_object_name() = ''COUNTRIES'' AND ' ||
                ' :dml.is_null_tag() = ''Y'' AND ' ||
                ' :dml.get_source_database_name() = ''DBS1.NET'' ');
DBMS_RULE_ADM.CREATE_RULE(
  rule_name  => 'strmadmin.all_countries_ddl',
  condition  => ' :ddl.get_object_owner() = ''HR'' AND ' ||
                ' :ddl.get_object_name() = ''COUNTRIES'' AND ' ||
                ' :ddl.is_null_tag() = ''Y'' AND ' ||
                ' :ddl.get_source_database_name() = ''DBS1.NET'' ');
-- Create rules for all modifications to the locations table
DBMS_RULE_ADM.CREATE_RULE(
  rule_name   => 'strmadmin.all_locations_dml',
  condition  => ' :dml.get_object_owner() = ''HR'' AND ' ||
                ' :dml.get_object_name() = ''LOCATIONS'' AND ' ||
                ' :dml.is_null_tag() = ''Y'' AND ' ||
                ' :dml.get_source_database_name() = ''DBS1.NET'' ');
DBMS_RULE_ADM.CREATE_RULE(
  rule_name   => 'strmadmin.all_locations_ddl',
  condition  => ' :ddl.get_object_owner() = ''HR'' AND ' ||
                ' :ddl.get_object_name() = ''LOCATIONS'' AND ' ||
                ' :ddl.is_null_tag() = ''Y'' AND ' ||
                ' :ddl.get_source_database_name() = ''DBS1.NET'' ');
-- Create rules for all modifications to the regions table
DBMS_RULE_ADM.CREATE_RULE(
  rule_name   => 'strmadmin.all_regions_dml',
  condition  => ' :dml.get_object_owner() = ''HR'' AND ' ||
                ' :dml.get_object_name() = ''REGIONS'' AND ' ||
                ' :dml.is_null_tag() = ''Y'' AND ' ||
                ' :dml.get_source_database_name() = ''DBS1.NET'' ');
DBMS_RULE_ADM.CREATE_RULE(
  rule_name   => 'strmadmin.all_regions_ddl',
  condition  => ' :ddl.get_object_owner() = ''HR'' AND ' ||
                ' :ddl.get_object_name() = ''REGIONS'' AND ' ||
                ' :ddl.is_null_tag() = ''Y'' AND ' ||
                ' :ddl.get_source_database_name() = ''DBS1.NET'' ');
-- Add rules to rule set
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_countries_dml',
  rule_set_name  => 'strmadmin.propagation_dbs3_rules');
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_countries_ddl',
  rule_set_name  => 'strmadmin.propagation_dbs3_rules');
```

```
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_locations_dml',
    rule_set_name  => 'strmadmin.propagation_dbs3_rules');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_locations_ddl',
    rule_set_name  => 'strmadmin.propagation_dbs3_rules');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_regions_dml',
    rule_set_name  => 'strmadmin.propagation_dbs3_rules');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_regions_ddl',
    rule_set_name  => 'strmadmin.propagation_dbs3_rules');
  -- Create the propagation
  DBMS_PROPAGATION_ADM.CREATE_PROPAGATION(
    propagation_name    => 'dbs2_to_dbs3',
    source_queue        => 'strmadmin.streams_queue',
    destination_queue   => 'strmadmin.streams_queue',
    destination_dblink  => 'dbs3.net',
    rule_set_name       => 'strmadmin.propagation_dbs3_rules');
END;
/

/*
```

手順 14 dbs2.net での行 LCR 用ルールベースの変換の作成

dbs2.net に hr ユーザーとして接続します。

```
*/

CONNECT hr/hr@dbs2.net

/*
```

dbs1.net で jobs 表に対する DML 文によって発生した行の変更が、dbs2.net の assignments 表に対する行の変更に変換されるように、ルールベースの変換ファンクションを作成します。

次のファンクションでは、jobs 表の各行 LCR が assignments 表の行 LCR に変換されます。

注意： assignments 表に DDL 変更も適用されている場合は、DDL LCR 用に別の変換が必要になります。この変換では、オブジェクト名と DDL テキストを変更する必要があります。

```
*/

CREATE OR REPLACE FUNCTION hr.to_assignments_trans_dml(
  p_in_data in SYS.AnyData)
  RETURN SYS.AnyData IS out_data SYS.LCR$_ROW_RECORD;
  tc   pls_integer;
BEGIN
  -- Typecast AnyData to LCR$_ROW_RECORD
     tc := p_in_data.GetObject(out_data);
     IF out_data.GET_OBJECT_NAME() = 'JOBS'
     THEN
  -- Transform the in_data into the out_data
     out_data.SET_OBJECT_NAME('ASSIGNMENTS');
     END IF;
  -- Convert to AnyData
     RETURN SYS.AnyData.ConvertObject(out_data);
END;
/

/*
```

手順 15 dbs2.net でローカルの適用に使用する適用プロセスの構成

dbs2.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs2.net

/*
```

dbs2.net を、ローカルの assignments 表に変更を適用するように構成します。assignments 表は、dbs1.net の jobs 表から変更を受信することに注意してください。

```
*/

DECLARE
  action_ctx_dml       SYS.RE$NV_LIST;
  action_ctx_ddl       SYS.RE$NV_LIST;
  ac_name              VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
BEGIN
  -- Specify the name-value pair in the action context
  action_ctx_dml := SYS.RE$NV_LIST(SYS.RE$NV_ARRAY());
  action_ctx_dml.ADD_PAIR(
    ac_name,
    SYS.ANYDATA.CONVERTVARCHAR2('hr.to_assignments_trans_dml'));
  --  Create the rule set strmadmin.apply_rules
```

```
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name       => 'strmadmin.apply_rules',
    evaluation_context  => 'SYS.STREAMS$_EVALUATION_CONTEXT');
  --  Create a rule that transforms all DML changes to the jobs table into
  --  DML changes for assignments table
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name       => 'strmadmin.all_jobs_dml',
    condition       => ' :dml.get_object_owner() = ''HR'' AND ' ||
                       ' :dml.get_object_name() = ''JOBS'' AND ' ||
                       ' :dml.is_null_tag() = ''Y'' AND ' ||
                       ' :dml.get_source_database_name() = ''DBS1.NET'' ',
    action_context  => action_ctx_dml);
  --  Add the rule to the rule set
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_jobs_dml',
    rule_set_name  => 'strmadmin.apply_rules');
  -- Create an apply process that uses the rule set
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name     => 'strmadmin.streams_queue',
    apply_name     => 'apply_dbs2',
    rule_set_name  => 'strmadmin.apply_rules',
    apply_user     => 'hr',
    apply_captured => true);
END;
/

/*
```

手順 16 hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与

前述の手順で hr ユーザーを適用ユーザーとして指定したため、hr ユーザーには適用プロセスで使用されるルール・セットの実行権限が必要です。

```
*/

BEGIN
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => 'strmadmin.apply_rules',
    grantee     => 'hr');
END;
/

/*
```

手順 17 dbs2.net でローカルの適用に使用する適用プロセスの起動

エラーが発生しても適用プロセスが無効化されないように、disable_on_error パラメータを n に設定して、dbs2.net でローカル適用のための適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_dbs2',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply_dbs2');
END;
/

/*
```

手順 18 dbs4.net での適用に使用する dbs2.net の適用プロセスの構成

dbs2.net を、Sybase データベース dbs4.net の jobs 表に DML 変更を適用するように構成します。これらの変更が dbs1.net で発生したことに注意してください。

```
*/

BEGIN
  -- Create the rule set
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name      => 'strmadmin.apply_dbs4_rules',
    evaluation_context => 'SYS.STREAMS$_EVALUATION_CONTEXT');
  -- Create rule strmadmin.all_jobs_remote for all modifications
  -- to the jobs table
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name    => 'strmadmin.all_jobs_remote',
    condition    => ' :dml.get_object_owner() = ''HR'' AND ' ||
                    ' :dml.get_object_name() = ''JOBS'' AND ' ||
                    ' :dml.is_null_tag() = ''Y'' AND ' ||
                    ' :dml.get_source_database_name() = ''DBS1.NET'' ');
  -- Add the rule to the rule set
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_jobs_remote',
    rule_set_name  => 'strmadmin.apply_dbs4_rules');
  -- Create an apply process that uses the rule set
```

```
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name          => 'strmadmin.streams_queue',
    apply_name          => 'apply_dbs4',
    rule_set_name       => 'strmadmin.apply_dbs4_rules',
    apply_database_link => 'dbs4.net',
    apply_captured      => true);
END;
/

/*
```

手順 19 dbs4.net での適用に使用する dbs2.net の適用プロセスの起動

エラーが発生しても適用プロセスが無効化されないように、disable_on_error パラメータを n に設定し、データベース・リンク dbs4.net を使用して Sybase 用のリモート適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_dbs4',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply_dbs4');
END;
/

/*
```

手順 20 dbs1.net での取得プロセスの起動

dbs1.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs1.net

/*
```

dbs1.net で取得プロセスを起動します。

```
*/

BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name  => 'capture');
END;
/

/*
```

手順 21 スプール結果のチェック

このスクリプトの完了後に streams_share_schema2.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/*
```

ここで、dbs1.net で特定の表に対する DML 変更および DDL 変更を行い、これらの変更が、この環境での Streams プロセスおよび伝播用に構成したルールに基づいて環境内の他のデータベースにレプリケートされていることを確認できます。

> **関連項目：** この環境でレプリケートされた変更の例については、22-56 ページの「hr スキーマでの表に対する DML 変更および DDL 変更」を参照してください。

```
/************************** END OF SCRIPT ******************************/
```

hr スキーマでの表に対する DML 変更および DDL 変更

22-18 ページの「単一データベースからのデータを共有するためのスクリプトの例」の項で説明されているいずれかの例を完了した後に、dbs1.net データベースで hr スキーマの表に対する DML 変更および DDL 変更を行うことができます。これらの変更は、Streams プロセスおよび伝播用に構成したルールに基づいて、環境内で他のデータベースにレプリケートされます。変更がレプリケートされていることを他のデータベースで確認できます。

たとえば、次の手順に従って、dbs1.net で、hr.jobs および hr.locations 表に対する DML 変更を行います。また、dbs1.net で hr.locations 表に対する DDL 変更も行うことができます。

これらの変更を行った後、dbs2.net で hr.assignments 表を問い合せて、dbs1.net でこの表に対して行った DML 変更がレプリケートされていることを確認できます。dbs2.net で適用プロセス用に構成したルールベースの変換によって、hr.jobs 表に対する DML 変更が、hr.assignments 表に対する DML 変更に変換されることに注意してください。dbs3.net で hr.locations 表を問い合せて、dbs1.net でこの表に対して行った DML 変更および DDL 変更がレプリケートされていることを確認できます。

手順 1 hr スキーマでの表に対する DML 変更および DDL 変更

次の変更を行います。

```
CONNECT hr/hr@dbs1.net

UPDATE hr.jobs SET max_salary=10000 WHERE job_id='MK_REP';
COMMIT;

INSERT INTO hr.locations VALUES(
  3300, '521 Ralston Avenue', '94002', 'Belmont', 'CA', 'US');
COMMIT;

ALTER TABLE hr.locations RENAME COLUMN state_province TO state_or_province;
```

手順 2 dbs2.net での hr.assignments 表の問合せ

前述の手順で実行した変更を取得、伝播し、適用するのに必要な時間が経過した後、次の問合せを実行して、dbs1.net で hr.jobs 表に対して行われた UPDATE 変更が、dbs2.net で hr.assignments 表に適用されていることを確認します。

```
CONNECT hr/hr@dbs2.net

SELECT max_salary FROM hr.assignments WHERE job_id='MK_REP';
```

max_salary の値が 10000 になっているはずです。

手順 3 dbs3.net での hr.locations 表の問合せおよび記述

次の問合せを実行して、dbs1.net で hr.locations 表に対して行われた INSERT 変更が、dbs3.net で適用されていることを確認します。

```
CONNECT hr/hr@dbs3.net

SELECT * FROM hr.locations WHERE location_id=3300;
```

前述の手順で dbs1.net で hr.locations 表に挿入された行を確認します。

次に、hr.locations 表で記述して、ALTER 変更および TABLE 変更が正しく伝播され、適用されたことを確認します。

```
DESC hr.locations
```

表の 5 番目の列が state_or_province になっているはずです。

既存の Streams レプリケーション環境へのオブジェクトの追加

この例では、レプリケート・オブジェクトを既存のデータベースに追加して、前項で構成した Streams 環境を拡張します。この例を実行するには、前述の例のいずれか一方のタスクを完了している必要があります。

この例では、dbs3.net データベースの hr スキーマに次の表を追加します。

- departments
- employees
- job_history
- jobs

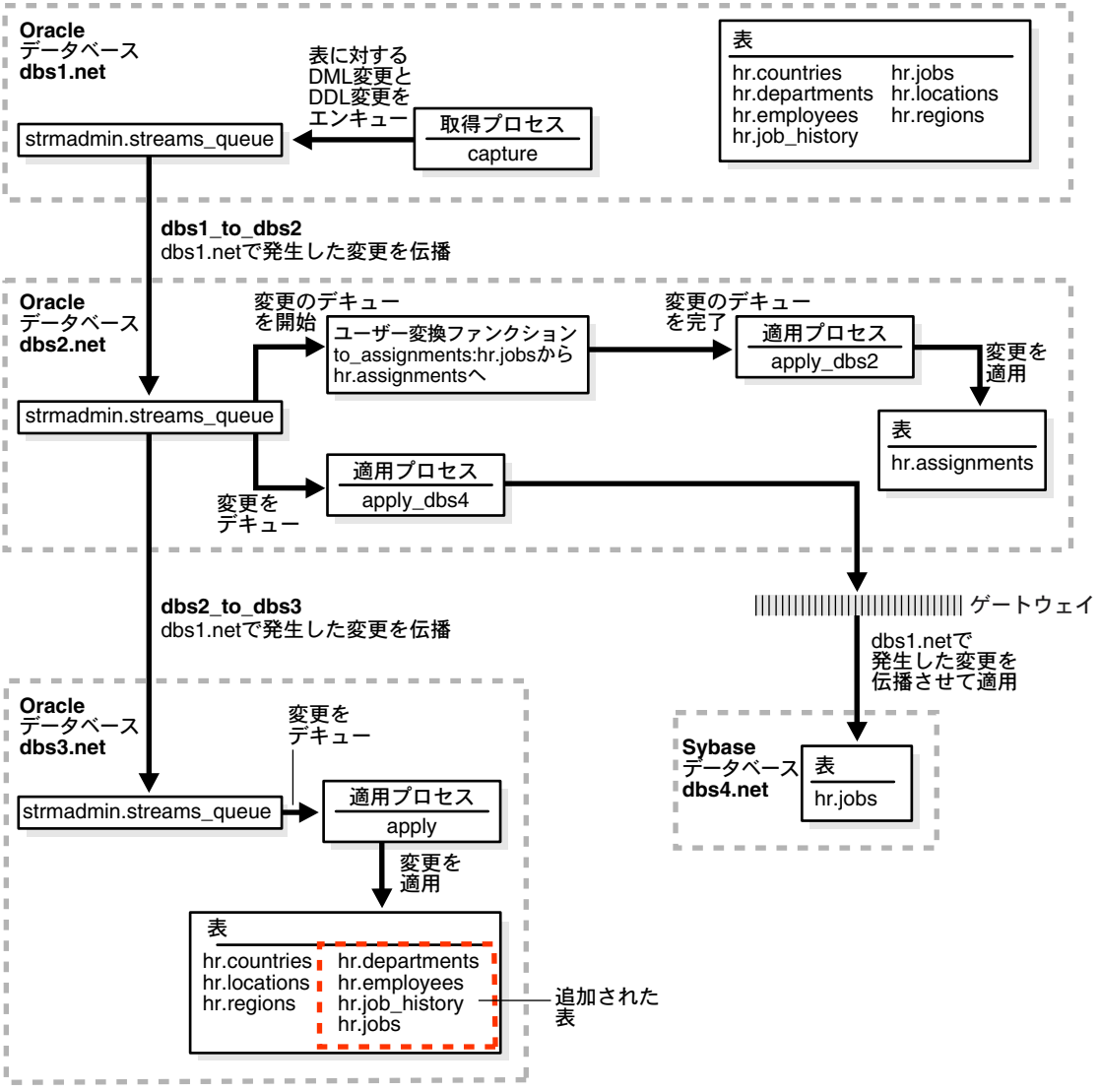
この例を完了すると、Streams では、これらの表に対する変更が次の一連のアクションで処理されます。

1. 取得プロセスが dbs1.net で変更を取得してエンキューします。
2. 伝播によって、dbs1.net のキューから dbs2.net のキューに変更が伝播されます。
3. 伝播によって、dbs2.net のキューから dbs3.net のキューに変更が伝播されます。
4. dbs3.net の適用プロセスが、変更を dbs3.net で適用します。

この例を完了すると、countries 表、locations 表および regions 表は前項で dbs3.net でインスタンス化されているため、dbs3.net データベースの hr スキーマにはオリジナルの表がすべて含まれることになります。

図 22-2 に、表が追加された環境の概要を示します。

図 22-2 環境内の dbs3.net へのオブジェクトの追加



次の手順に従って、前述の表を dbs3.net データベースにレプリケートします。

1. 出力の表示と結果のスプーリング
2. dbs3.net での適用プロセスの停止
3. dbs3.net で追加の表に使用する適用プロセスの構成
4. dbs2.net で追加の表に使用する表伝播ルールの指定
5. 追加した 4 つの表を dbs1.net でインスタンス化するための準備
6. dbs3.net での dbs1.net の表のインスタンス化
7. dbs3.net でのサプリメンタル・ログ・グループの削除
8. dbs3.net での適用プロセスの起動
9. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 22-67 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************* BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_addobjs.out

/*
```

手順 2 dbs3.net での適用プロセスの停止

オブジェクトを dbs3.net に追加し終わるまでは、追加したオブジェクトに対する変更を適用する適用プロセスによって、これらのオブジェクトに対する変更が適用されないようにする必要があります。そのためには、ソース・データベースで取得プロセスを停止します。または、dbs2.net から dbs3.net への変更の伝播を停止する方法もあります。さらに、dbs3.net で適用プロセスを停止する方法もあります。この例では、dbs3.net で適用プロセスを停止します。

dbs3.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs3.net

/*
```

dbs3.net で適用プロセスを停止します。

```
*/

BEGIN
  DBMS_APPLY_ADM.STOP_APPLY(
    apply_name  => 'apply');
END;
/

/*
```

手順 3 dbs3.net で追加の表に使用する適用プロセスの構成

dbs3.net の適用プロセスを、追加する表に変更を適用するように構成します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.departments',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'dbs1.net');
END;
/
```

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.employees',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'dbs1.net');
END;
/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.job_history',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'dbs1.net');
END;
/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.jobs',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'dbs1.net');
END;
/

/*
```

手順 4 dbs2.net で追加の表に使用する表伝播ルールの指定

dbs2.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs2.net

/*
```

dbs2.net のキューから dbs3.net のキューへの伝播に使用するルールに、表を追加します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name               => 'hr.departments',
    streams_name             => 'dbs2_to_dbs3',
    source_queue_name        => 'strmadmin.streams_queue',
    destination_queue_name   => 'strmadmin.streams_queue@dbs3.net',
    include_dml              =>  true,
    include_ddl              =>  true,
    source_database          => 'dbs1.net');
END;
/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              => 'hr.employees',
    streams_name            => 'dbs2_to_dbs3',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@dbs3.net',
    include_dml             =>  true,
    include_ddl             =>  true,
    source_database         => 'dbs1.net');
END;
/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              => 'hr.job_history',
    streams_name            => 'dbs2_to_dbs3',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@dbs3.net',
    include_dml             =>  true,
    include_ddl             =>  true,
    source_database         => 'dbs1.net');
```

```
END;
/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              => 'hr.jobs',
    streams_name            => 'dbs2_to_dbs3',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@dbs3.net',
    include_dml             =>  true,
    include_ddl             =>  true,
    source_database         => 'dbs1.net');
END;
/

/*
```

手順 5 追加した 4 つの表を dbs1.net でインスタンス化するための準備

dbs1.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs1.net

/*
```

表をインスタンス化できるように準備します。これらの表は、dbs3.net でインスタンス化されます。この手順では、表の最小 SCN にインスタンス化のマークを付けます。最小 SCN 以降の SCN は、インスタンス化に使用できます。また、dbs3.net の関連する伝播と適用プロセスに関する Streams データ・ディクショナリに、これらの表に関する情報が含まれるようにするためにも、この準備が必要です。

関連項目：

- 12-10 ページ「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」
- 3-24 ページ「伝播用の Streams データ・ディクショナリ」
- 4-31 ページ「適用プロセス用の Streams データ・ディクショナリ」

```
*/

BEGIN
  DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
    table_name  => 'hr.departments');
END;
/

BEGIN
  DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
    table_name  => 'hr.employees');
END;
/

BEGIN
  DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
    table_name  => 'hr.job_history');
END;
/

BEGIN
  DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
    table_name  => 'hr.jobs');
END;
/

/*
```

手順 6 dbs3.net での dbs1.net の表のインスタンス化

dbs1.net で異なるウィンドウを開き、dbs3.net でインスタンス化する表をエクスポートします。エクスポート・コマンドの実行時に、OBJECT_CONSISTENT エクスポート・パラメータが y に設定されていることを確認します。また、エクスポート中は、エクスポート対象のオブジェクトに対する DDL 変更が行われないことを確認してください。

次にエクスポート・コマンドの例を示します。

```
exp userid=hr/hr FILE=hr_instant2.dmp TABLES=departments,employees,job_history,jobs
OBJECT_CONSISTENT=y
```

> **関連項目：**『Oracle9*i* データベース・ユーティリティ』でエクスポートの実行方法を参照してください。

```
*/

PAUSE Press <RETURN> to continue when the export is complete in the other window
that you opened.

/*
```

エクスポート・ダンプ・ファイル hr_instant2.dmp を接続先データベースに送信します。この例では、接続先データベースは dbs3.net です。

バイナリ FTP または他のなんらかの方法を使用して、エクスポート・ダンプ・ファイルを接続先データベースに送信できます。ファイルを送信するには、異なるウィンドウを開く必要がある場合があります。

```
*/

PAUSE Press <RETURN> to continue after transferring the dump file.

/*
```

別のウィンドウで、dbs3.net データベースが稼働しているコンピュータに接続し、エクスポート・ダンプ・ファイル hr_instant2.dmp をインポートして、dbs3.net データベース内の表をインスタンス化します。dbs3.net が稼働しているコンピュータには、telnet またはリモート・ログインを使用して接続できます。

インポート・コマンドの実行時には、STREAMS_INSTANTIATION インポート・パラメータが y に設定されていることを確認してください。このパラメータによって、インポートされる各オブジェクトのエクスポート SCN 情報が、インポート時に確実に記録されます。

次にインポート・コマンドの例を示します。

```
imp userid=hr/hr FILE=hr_instant2.dmp IGNORE=y FULL=y COMMIT=y LOG=import.log
STREAMS_INSTANTIATION=y
```

> **関連項目：**『Oracle9*i* データベース・ユーティリティ』でインポートの実行方法を参照してください。

```
*/

PAUSE Press <RETURN> to continue after the import is complete at dbs3.net.

/*
```

手順 7 dbs3.net でのサプリメンタル・ログ・グループの削除

hr スキーマを dbs3.net でインスタンス化した場合、dbs1.net からのサプリメンタル・ログ・グループは保持されています。dbs3.net では、hr スキーマの表に対する変更を取得する取得プロセスが存在しないため、これらのログ・グループは dbs3.net で必要ありません。ログ・グループを削除して、dbs3.net の REDO ログ内に余分な情報が書き込まれることを回避できます。

dbs3.net に hr ユーザーとして接続します。

```
*/

CONNECT hr/hr@dbs3.net

/*
```

dbs3.net でサプリメンタル・ログ・グループを削除します。

```
*/

ALTER TABLE hr.departments DROP SUPPLEMENTAL LOG GROUP log_group_departments_pk;

ALTER TABLE hr.employees DROP SUPPLEMENTAL LOG GROUP log_group_employees_pk;

ALTER TABLE hr.jobs DROP SUPPLEMENTAL LOG GROUP log_group_jobs_pk;

ALTER TABLE hr.job_history DROP SUPPLEMENTAL LOG GROUP log_group_job_history_pk;

/*
```

手順 8 dbs3.net での適用プロセスの起動

dbs3.net で適用プロセスを起動します。これは、手順 2 で停止した適用プロセスです。

dbs3.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs3.net

/*
```

dbs3.net で適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply');
```

```
END;
/

/*
```

手順 9 スプール結果のチェック

このスクリプトの完了後に streams_addobjs.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT ******************************/
```

hr.employees 表に対する DML 変更

22-57 ページの「既存の Streams レプリケーション環境へのオブジェクトの追加」の項で説明されている例を完了した後に、dbs1.net データベースで hr スキーマの表に対する DML 変更および DDL 変更を行うことができます。これらの変更は、dbs3.net にレプリケートされます。変更がレプリケートされていることを、dbs3.net のこれらの表で確認できます。

たとえば、次の手順に従って、dbs1.net で hr.employees 表に対する DML 変更を行います。次に、dbs3.net で hr.employees 表を問い合せて、変更がレプリケートされていることを確認します。

手順 1 hr.employees 表に対する DML 変更

次の変更を行います。

```
CONNECT hr/hr@dbs1.net

UPDATE hr.employees SET job_id='ST_MAN' WHERE employee_id=143;
COMMIT;
```

手順 2 dbs3.net での hr.employees 表の問合せ

前述の手順で実行した変更を取得、伝播し、適用するのに必要な時間が経過した後、次の問合せを実行して、dbs1.net で hr.employees 表に対して行われた UPDATE 変更が、dbs3.net で hr.employees 表に適用されていることを確認します。

```
CONNECT hr/hr@dbs3.net

SELECT job_id FROM hr.employees WHERE employee_id=143;
```

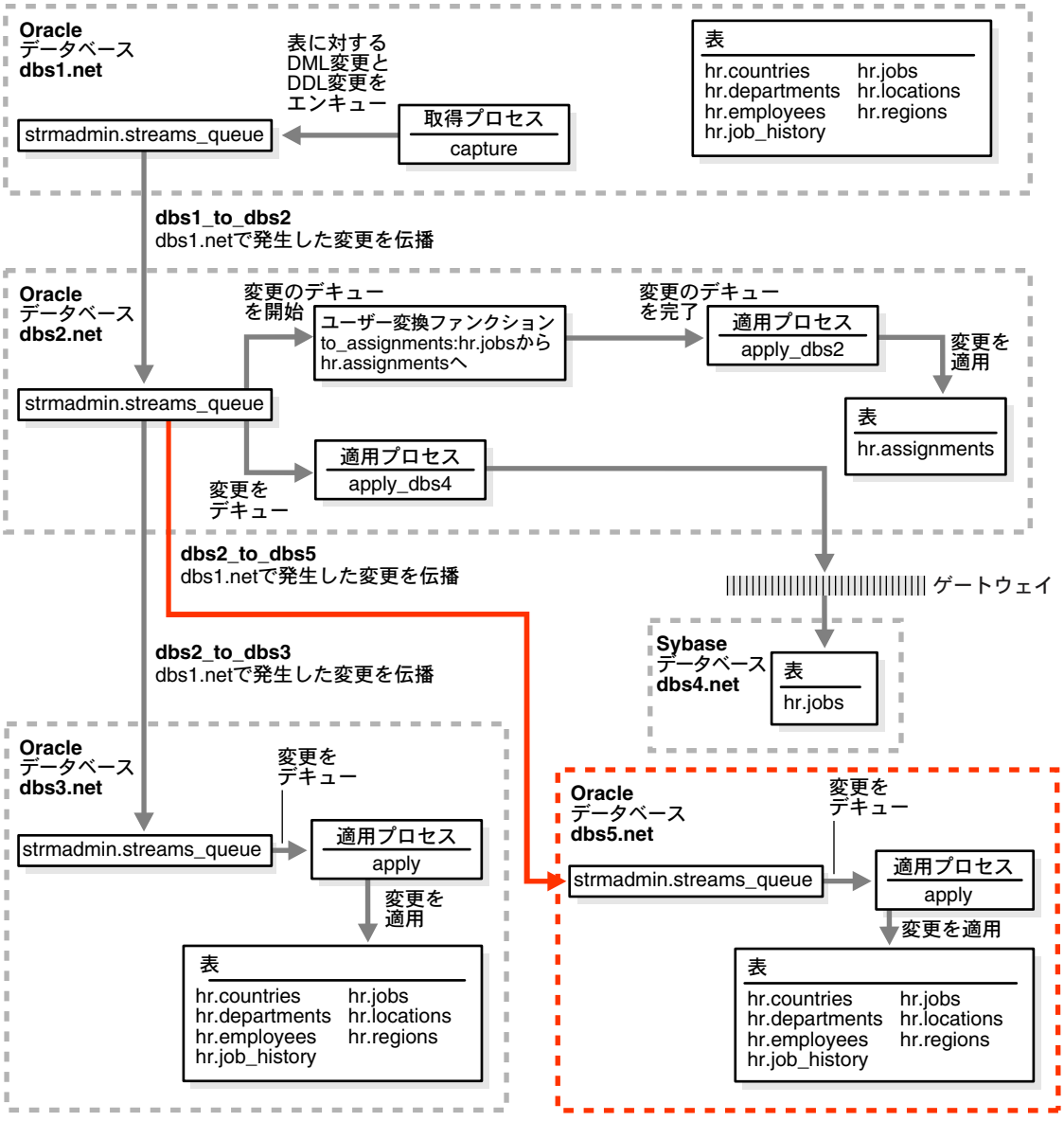
job_id の値が ST_MAN になっているはずです。

既存の Streams レプリケーション環境へのデータベースの追加

この例では、既存の構成にデータベースを追加して、前項で構成した Streams 環境を拡張します。この例では、dbs2.net のキューから hr スキーマ全体に対する変更を受信するために、既存の Oracle データベース dbs5.net を追加します。

図 22-3 に、データベースが追加された環境の概要を示します。

図 22-3 環境への dbs5.net Oracle データベースの追加



この例を完了するには、次の前提条件を満たしている必要があります。

- dbs5.net データベースが存在すること。
- dbs2.net および dbs5.net データベースが Oracle Net を介して相互に通信できること。
- この章の前述の例のタスクを完了していること。
- Streams 環境全体を正常に機能させる場合は、22-5 ページの「前提条件」を満たしていること。
- この例では、各データベースで新規ユーザーが Streams 管理者（strmadmin）として作成され、このユーザーのデータに使用する表領域の指定を求めるプロンプトが表示されます。この例を開始する前に、各データベースで Streams 管理者用に新規の表領域を作成するか、既存の表領域を識別してください。SYSTEM 表領域は、Streams 管理者用にしないでください。

次の手順に従って、Streams 環境に dbs5.net を追加します。

1. 出力の表示と結果のスプーリング
2. dbs5.net の hr スキーマにあるすべての表の削除
3. dbs5.net でのユーザーの設定
4. dbs5.net での Streams キューの作成
5. dbs5.net での適用プロセスの構成
6. dbs5.net での適用プロセスの適用ユーザー hr の指定
7. hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与
8. dbs2.net と dbs5.net の間のデータベース・リンクの作成
9. dbs2.net と dbs5.net の間の伝播の構成
10. dbs1.net の hr スキーマのインスタンス化の準備
11. dbs5.net での dbs1.net の表のインスタンス化
12. dbs5.net でのサプリメンタル・ログ・グループの削除
13. dbs5.net での適用プロセスの起動
14. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 22-80 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************* BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_adddb.out

/*
```

手順 2 dbs5.net の hr スキーマにあるすべての表の削除

この例では、dbs1.net からエクスポートして dbs5.net にインポートすることで、hr スキーマ内の表をインスタンス化する方法を示します。この例のインスタンス化部分を正常に機能させるには、dbs5.net でこれらの表を削除する必要があります。

dbs5.net に hr として接続します。

```
*/

CONNECT hr/hr@dbs5.net

/*
```

dbs5.net データベースで hr スキーマ内の表をすべて削除します。

注意： この手順に従って hr スキーマ内の表をすべて削除した後に、この例の残りの項の説明に従って dbs5.net で hr スキーマを再インスタンス化する必要があります。hr スキーマが Oracle データベースに存在しない場合は、Oracle マニュアル・セットに記載されている一部の例が失敗する可能性があります。

```
*/

DROP TABLE hr.countries CASCADE CONSTRAINTS;
DROP TABLE hr.departments CASCADE CONSTRAINTS;
DROP TABLE hr.employees CASCADE CONSTRAINTS;
DROP TABLE hr.job_history CASCADE CONSTRAINTS;
DROP TABLE hr.jobs CASCADE CONSTRAINTS;
DROP TABLE hr.locations CASCADE CONSTRAINTS;
DROP TABLE hr.regions CASCADE CONSTRAINTS;

/*
```

手順 3 dbs5.net でのユーザーの設定

dbs5.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@dbs5.net AS SYSDBA

/*
```

Streams 管理者 strmadmin を作成し、このユーザーに必要な権限を付与します。これらの権限を付与されたユーザーは、キューの管理、Streams に関連するパッケージ内のサブプログラムの実行、ルール・セットおよびルールの作成、データ・ディクショナリ・ビューとキュー表の問合せによる Streams 環境の監視を行うことができます。このユーザー用に、異なる名前を選択できます。

注意：

- セキュリティを確保するために、Streams 管理者には strmadminpw 以外のパスワードを使用してください。
- Streams 管理者には、SELECT_CATALOG_ROLE は不要です。この例で付与されているのは、Streams 管理者が環境を簡単に監視できるようにするためです。
- Oracle Enterprise Manager の Streams ツールを使用する予定の場合は、Streams 管理者にこの手順で示す権限の他に SELECT ANY DICTIONARY 権限を付与します。
- ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on dbs5.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
                     QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM        TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM            TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM      TO strmadmin;

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

/*
```

手順 4 dbs5.net での Streams キューの作成

追加するデータベースに Streams 管理者として接続します。この例では、データベース dbs5.net に接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs5.net

/*
```

SET_UP_QUEUE プロシージャを実行して、dbs5.net でキュー streams_queue を作成します。このキューは、このデータベースで適用される変更を保持することで Streams キューとして機能します。

SET_UP_QUEUE プロシージャを実行すると、次のアクションが実行されます。

- キュー表 streams_queue_table が作成されます。このキュー表の所有者は Streams 管理者（strmadmin）であり、このユーザーのデフォルト記憶域が使用されます。
- Streams 管理者（strmadmin）が所有するキュー streams_queue が作成されます。
- キューが起動されます。

```
*/

EXEC  DBMS_STREAMS_ADM.SET_UP_QUEUE();

/*
```

手順 5 dbs5.net での適用プロセスの構成

dbs5.net に Streams 管理者として接続したまま、hr スキーマに変更を適用する適用プロセスを構成します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name     => 'hr',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'dbs1.net');
END;
/

/*
```

手順 6 dbs5.net での適用プロセスの適用ユーザー hr の指定

この例では、このデータベースで適用プロセスによって変更が適用される全データベース・オブジェクトを、hr ユーザーが所有しているものとします。したがって、hr はすでにこれらのデータベース・オブジェクトを変更するための権限を持っており、hr を適用ユーザーにすると便利です。

前述の手順で適用プロセスを作成するときには、Streams 管理者 strmadmin が適用プロセスの作成プロシージャを実行したため、デフォルトで strmadmin が適用ユーザーとして指定されました。hr を適用ユーザーとして指定するかわりに、strmadmin を適用ユーザーのままにすることもできます。ただし、その場合は strmadmin に、変更が適用される全データベース・オブジェクトに対する権限と、適用プロセスで使用される全ユーザー・プロシージャを実行する権限を付与する必要があります。適用プロセスによって複数のスキーマ内のデータベース・オブジェクトに変更が適用される環境では、Streams 管理者を適用ユーザーとして使用する方が便利な場合があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'apply',
    apply_user => 'hr');
END;
/

/*
```

手順 7 hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与

前述の手順で hr ユーザーを適用ユーザーとして指定したため、hr ユーザーには適用プロセスで使用されるルール・セットの実行権限が必要です。

```
*/

DECLARE
   rs_name  VARCHAR2(64);   -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
    INTO rs_name
    FROM DBA_APPLY
    WHERE APPLY_NAME='APPLY';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee     => 'hr');
```

```
END;
/

/*
```

手順 8 dbs2.net と dbs5.net の間のデータベース・リンクの作成

dbs2.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs2.net

/*
```

変更の伝播先となるデータベースへのデータベース・リンクを作成します。この例では、変更はデータベース dbs2.net から dbs5.net に伝播します。

```
*/

CREATE DATABASE LINK dbs5.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'dbs5.net';

/*
```

手順 9 dbs2.net と dbs5.net の間の伝播の構成

dbs2.net に Streams 管理者として接続したまま、dbs2.net のキューから dbs5.net のキューへの伝播を構成してスケジュールします。hr スキーマに対する変更が dbs1.net で発生したことに注意してください。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name             => 'hr',
    streams_name            => 'dbs2_to_dbs5',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@dbs5.net',
    include_dml             =>  true,
    include_ddl             =>  true,
    source_database         => 'dbs1.net');
END;
/

/*
```

手順 10 dbs1.net の hr スキーマのインスタンス化の準備

dbs1.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs1.net

/*
```

hr スキーマをインスタンス化できるように準備します。このスキーマ内の表が dbs5.net でインスタンス化されます。この準備は、dbs5.net の関連する伝播と適用プロセスに関する Streams データ・ディクショナリに、hr スキーマとこのスキーマ内のオブジェクトに関する情報が含まれるようにするために必要です。

関連項目：

- 12-10 ページ「ソース・データベースでインスタンス化を行うためのデータベース・オブジェクトの準備」
- 3-24 ページ「伝播用の Streams データ・ディクショナリ」
- 4-31 ページ「適用プロセス用の Streams データ・ディクショナリ」

```
*/

BEGIN
  DBMS_CAPTURE_ADM.PREPARE_SCHEMA_INSTANTIATION(
    schema_name  => 'hr');
END;
/

/*
```

手順 11 dbs5.net での dbs1.net の表のインスタンス化

dbs1.net で異なるウィンドウを開き、dbs5.net でインスタンス化するスキーマをエクスポートします。エクスポート・コマンドの実行時に、OBJECT_CONSISTENT エクスポート・パラメータが y に設定されていることを確認します。また、エクスポート中は、エクスポート対象のオブジェクトに対する DDL 変更が行われないように注意してください。

次にエクスポート・コマンドの例を示します。

```
exp hr/hr FILE=hr_schema.dmp OWNER=hr OBJECT_CONSISTENT=y
```

関連項目：『Oracle9*i* データベース・ユーティリティ』でエクスポートの実行方法を参照してください。

```
*/

PAUSE Press <RETURN> to continue when the export is complete in the other window
that you opened.

/*
```

エクスポート・ダンプ・ファイル hr_schema.dmp を接続先データベースに送信します。この例では、接続先データベースは dbs5.net です。

バイナリ FTP または他のなんらかの方法を使用して、エクスポート・ダンプ・ファイルを接続先データベースに送信できます。ファイルを送信するには、異なるウィンドウを開く必要がある場合があります。

```
*/

PAUSE Press <RETURN> to continue after transferring the dump file.

/*
```

別のウィンドウで、dbs5.net データベースが稼働しているコンピュータに接続し、エクスポート・ダンプ・ファイル hr_schema.dmp をインポートして、dbs5.net データベース内の表をインスタンス化します。dbs5.net が稼働しているコンピュータには、telnet またはリモート・ログインを使用して接続できます。

インポート・コマンドの実行時には、STREAMS_INSTANTIATION インポート・パラメータが y に設定されていることを確認してください。このパラメータによって、インポートされる各オブジェクトのエクスポート SCN 情報が、インポート時に確実に記録されます。

次にインポート・コマンドの例を示します。

```
imp hr/hr FILE=hr_schema.dmp FROMUSER=hr IGNORE=y COMMIT=y LOG=import.log
STREAMS_INSTANTIATION=y
```

> **関連項目：**『Oracle9*i* データベース・ユーティリティ』でインポートの実行方法を参照してください。

```
*/

PAUSE Press <RETURN> to continue after the import is complete at dbs5.net.

/*
```

手順 12 dbs5.net でのサプリメンタル・ログ・グループの削除

hr スキーマを dbs5.net でインスタンス化した場合、dbs1.net からのサプリメンタル・ログ・グループは保持されています。dbs5.net では、hr スキーマの表に対する変更を取得する取得プロセスが存在しないため、これらのログ・グループは dbs5.net で必要ありません。ログ・グループを削除して、dbs5.net の REDO ログ内の誤った情報を回避できます。

dbs5.net に hr ユーザーとして接続します。

```
*/

CONNECT hr/hr@dbs5.net

/*
```

dbs5.net でサプリメンタル・ログ・グループを削除します。

```
*/

ALTER TABLE hr.countries DROP SUPPLEMENTAL LOG GROUP log_group_countries_pk;

ALTER TABLE hr.departments DROP SUPPLEMENTAL LOG GROUP log_group_departments_pk;

ALTER TABLE hr.employees DROP SUPPLEMENTAL LOG GROUP log_group_employees_pk;

ALTER TABLE hr.jobs DROP SUPPLEMENTAL LOG GROUP log_group_jobs_pk;

ALTER TABLE hr.job_history DROP SUPPLEMENTAL LOG GROUP log_group_job_history_pk;

ALTER TABLE hr.locations DROP SUPPLEMENTAL LOG GROUP log_group_locations_pk;

ALTER TABLE hr.regions DROP SUPPLEMENTAL LOG GROUP log_group_regions_pk;

/*
```

手順 13 dbs5.net での適用プロセスの起動

dbs5.net で Streams 管理者として接続します。

```
*/

CONNECT strmadmin/strmadminpw@dbs5.net

/*
```

エラーが発生しても適用プロセスが無効化されないように、disable_on_error パラメータを n に設定して、dbs5.net で適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply');
END;
/

/*
```

手順 14 スプール結果のチェック

このスクリプトの完了後に streams_adddb.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT ******************************/
```

hr.departments 表に対する DML 変更

22-68 ページの「既存の Streams レプリケーション環境へのデータベースの追加」の項で説明されている例を完了した後に、dbs1.net データベースで hr スキーマの表に対する DML 変更および DDL 変更を行うことができます。これらの変更は、dbs5.net にレプリケートされます。変更がレプリケートされていることを、dbs5.net のこれらの表で確認できます。

たとえば、次の手順に従って、dbs1.net で hr.departments 表に対する DML 変更を行います。次に、dbs5.net で hr.departments 表を問い合せて、変更がレプリケートされていることを確認します。

手順 1 hr.departments 表に対する DML 変更

次の変更を行います。

```
CONNECT hr/hr@dbs1.net

UPDATE hr.departments SET location_id=2400 WHERE department_id=270;
COMMIT;
```

手順 2 dbs5.net での hr.departments 表の問合せ

前述の手順で実行した変更を取得、伝播し、適用するのに必要な時間が経過した後、次の問合せを実行して、dbs1.net で hr.departments 表に対して行われた UPDATE 変更が、dbs5.net で hr.departments 表に適用されていることを確認します。

```
CONNECT hr/hr@dbs5.net

SELECT location_id FROM hr.departments WHERE department_id=270;
```

location_id の値が 2400 になっているはずです。

23

複数のソース・レプリケーションの例

この章では、Streams を使用して構成できる複数のソース・レプリケーション環境の例を説明します。

この章の内容は次のとおりです。

複数のソース・データベース例の概要

この例では、Streams を使用して、3 つの Oracle データベース間でスキーマのデータをレプリケートする方法について説明します。hr スキーマ内の表に対する DML 変更と DDL 変更が、この環境の全データベースで取得され、この環境の他の各データベースに伝播します。

図 23-1 に、この環境の概要を示します。

図 23-1 複数のデータベースからのデータが共有される環境の例

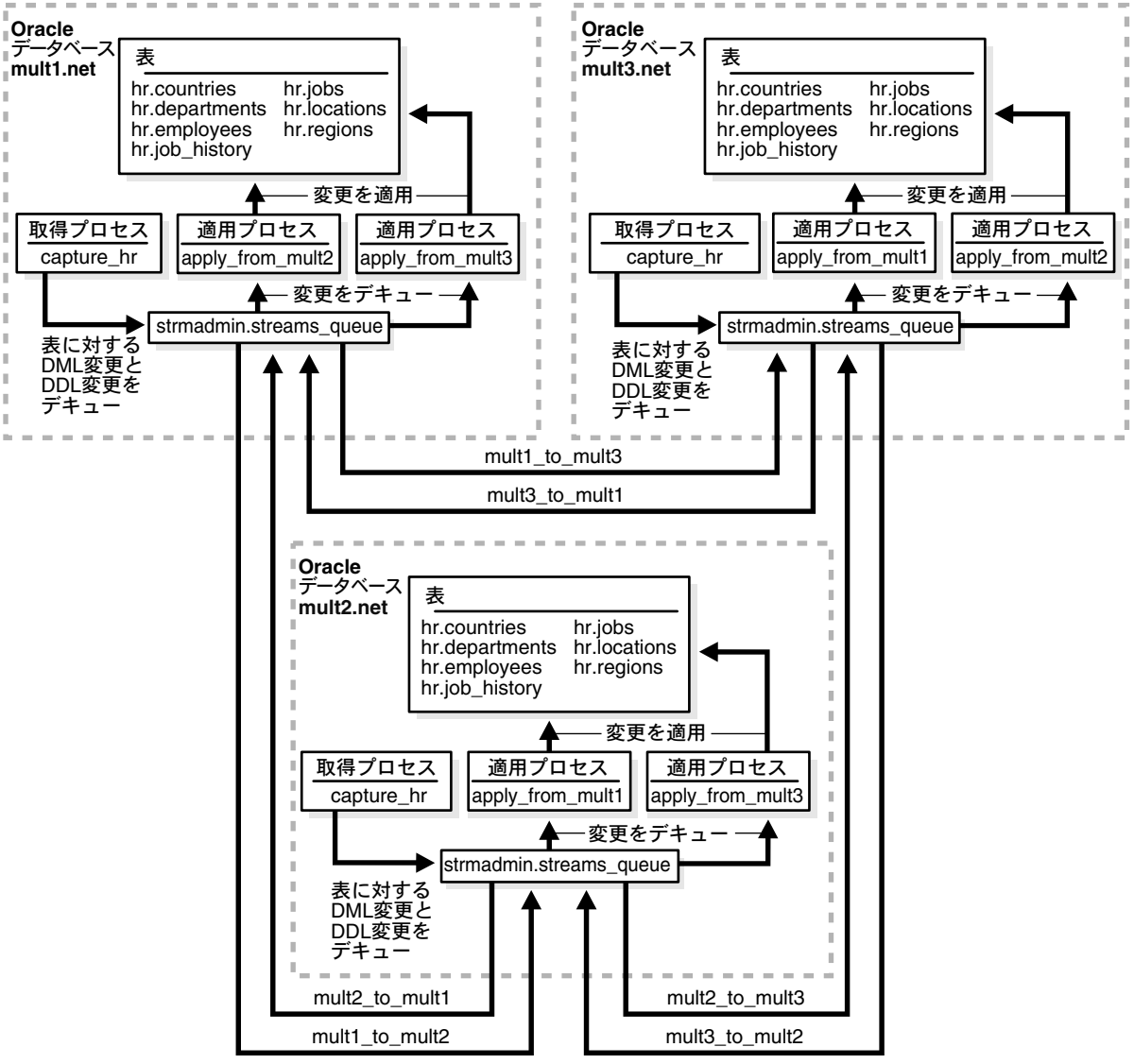


図 23-1 に示すように、この例が完了すると、すべてのデータベースに hr スキーマが含まれることになります。ただし、この例の開始時には、hr スキーマは mult1.net にのみ存在します。この例の実行中に、mult2.net と mult3.net で hr スキーマをインスタンス化します。

この例では、Streams を使用して次の一連のアクションが実行されます。

1. インスタンス化の後に、各データベースの取得プロセスが、hr スキーマ内のすべての表に対する DML 変更と DDL 変更を取得し、それをローカル・キューにエンキューします。
2. これらの変更が、各データベースから環境内の他の全データベースに伝播します。
3. 各データベースの適用プロセスが、環境内の他のデータベースから受信した hr スキーマ内の変更を適用します。

この例では各データベースにキューを 1 つのみ使用しますが、様々なソース・データベースからの変更を分離する必要がある場合は、データベースごとに複数のキューを使用できます。また、この例では、適用プロセスにデフォルトの適用タグを使用して、変更がソース・データベースに送信されるのを回避します。適用プロセスを作成すると、その適用プロセスで適用される変更の REDO エントリには、デフォルトでタグ '00'（2 つのゼロ）が使用されます。デフォルトでは、これらの変更が再取得されることはありません。これは、DBMS_STREAMS_ADM パッケージで作成されたルールにはデフォルトで is_null_tag()='Y' 条件があり、この条件によって、REDO エントリのタグが NULL の場合にのみ各取得プロセスで REDO エントリに対する変更が確実に取得されるためです。

> **関連項目：** タグの詳細は、第 8 章「Streams のタグ」および 16-24 ページの「Streams タグの管理」を参照してください。

前提条件

この章の例を開始する前に、前提条件となる次の作業を完了する必要があります。

- Streams 環境内の各データベースで、次の初期化パラメータを指定の値に設定します。
  - AQ_TM_PROCESSES: このパラメータでは、キュー・モニター・プロセスを設定します。値 1 ～ 10 では、メッセージの監視用に作成するキュー・モニター・プロセスの数を指定します。AQ_TM_PROCESSES を指定しないか、0 に設定すると、キュー・モニター・プロセスは作成されません。この例では、AQ_TM_PROCESSES を 1 以上の値に設定する必要があります。

    このパラメータを 1 以上に設定すると、指定した数のキュー・モニター・プロセスが起動します。これらのキュー・モニター・プロセスは、遅延や期限切れなど、時間ベースのメッセージ操作の管理、指定した保存期間を過ぎて保存されているメッセージのクリーン・アップ、および保存期間が 0（ゼロ）の場合のコンシューム済みメッセージのクリーン・アップを受け持ちます。

- GLOBAL_NAMES: このパラメータは true に設定する必要があります。データベースのグローバル名が mult1.net、mult2.net および mult3.net であることを確認してください。
- JOB_QUEUE_PROCESSES: 各データベースでイベントが伝播されるため、このパラメータを 2 以上の値に設定する必要があります。同時に実行できるジョブの最大数に 1 を加えた値に設定してください。
- COMPATIBLE: このパラメータは 9.2.0 以上に設定する必要があります。
- LOG_PARALLELISM: 各データベースによりイベントが取得されるため、このパラメータは 1 に設定する必要があります。

注意： この例を正しく実行するには、他の初期化パラメータの設定を変更することが必要な場合もあります。

関連項目： Streams 環境で重要な他の初期化パラメータについては、11-4 ページの「Streams に関連する初期化パラメータの設定」を参照してください。

- 取得される変更を生成するデータベースは、ARCHIVELOG モードで実行している必要があります。この例では、すべてのデータベースで変更が取得されるため、すべてのデータベースを ARCHIVELOG モードで実行する必要があります。

  関連項目： ARCHIVELOG モードでデータベースを実行する方法については、『Oracle9*i* データベース管理者ガイド』を参照してください。

- ネットワークと Oracle Net を、3 つのデータベースすべてが相互に通信できるように構成します。

  関連項目： 『Oracle9*i* Net Services 管理者ガイド』

ユーザーの設定とキューおよびデータベース・リンクの作成

この項では、3 つの Oracle データベースを含む Streams レプリケーション環境用に、ユーザーを設定してキューとデータベース・リンクを作成する方法について説明します。この例の残りの部分は、この項で構成するユーザーとキューに応じて異なります。

次の手順に従って、すべてのデータベースでユーザーを設定して `streams_queue` を作成します。

1. 出力の表示と結果のスプーリング
2. mult1.net での hr.countries 表の変更
3. mult1.net で LogMiner の表に使用する代替表領域の作成
4. mult1.net でのユーザーの設定
5. mult1.net での Streams キューの作成
6. mult1.net でのデータベース・リンクの作成
7. 最終時刻による競合解消のための mult1.net の表の準備
8. mult2.net で LogMiner の表に使用する代替表領域の作成
9. mult2.net でのユーザーの設定
10. mult2.net での Streams キューの作成
11. mult2.net でのデータベース・リンクの作成
12. mult2.net の hr スキーマにあるすべての表の削除
13. mult3.net で LogMiner の表に使用する代替表領域の作成
14. mult3.net でのユーザーの設定
15. mult3.net での Streams キューの作成
16. mult3.net でのデータベース・リンクの作成
17. mult3.net の hr スキーマにあるすべての表の削除
18. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 23-22 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************* BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_setup_mult.out

/*
```

手順 2 mult1.net での hr.countries 表の変更

mult1.net に hr ユーザーとして接続します。

```
*/

CONNECT hr/hr@mult1.net

/*
```

hr.countries 表を索引構成表から通常の表に変換します。現在、取得プロセスでは索引構成表に対する変更は取得できません。

```
*/

ALTER TABLE countries RENAME TO countries_orig;

CREATE TABLE hr.countries(
  country_id      CHAR(2) CONSTRAINT  country_id_nn_noiot NOT NULL,
  country_name    VARCHAR2(40),
  region_id       NUMBER,
  CONSTRAINT      country_c_id_pk_noiot  PRIMARY KEY (country_id));

ALTER TABLE hr.countries
ADD (CONSTRAINT countr_reg_fk_noiot
         FOREIGN KEY (region_id)
         REFERENCES regions(region_id));

INSERT INTO COUNTRIES (SELECT * FROM hr.countries_orig);

DROP TABLE hr.countries_orig CASCADE CONSTRAINTS;
```

```
ALTER TABLE locations
  ADD (CONSTRAINT loc_c_id_fk
    FOREIGN KEY (country_id)
    REFERENCES countries(country_id));

/*
```

手順 3 mult1.net で LogMiner の表に使用する代替表領域の作成

デフォルトでは、LogMiner の表は SYSTEM 表領域にありますが、変更を取得するために取得プロセスを起動すると、SYSTEM 表領域にこれらの表に使用する領域が足りなくなる場合があります。したがって、LogMiner の表に使用する代替表領域を作成する必要があります。

関連項目： 2-18 ページ「LogMiner の表のための代替表領域」

mult1.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@mult1.net AS SYSDBA

/*
```

LogMiner の表に使用する代替表領域を作成します。

注意： 各 ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

```
*/

ACCEPT tspace_name DEFAULT 'logmnrts' PROMPT 'Enter the name of the tablespace (for
example, logmnrts): '

ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the datafile
directory (for example, /usr/oracle/dbs): '

ACCEPT db_file_name DEFAULT 'logmnrts.dbf' PROMPT 'Enter the name of the datafile
(for example, logmnrts.dbf): '

CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'
  SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');

/*
```

手順 4 mult1.net でのユーザーの設定

Streams 管理者 strmadmin を作成し、このユーザーに必要な権限を付与します。これらの権限を付与されたユーザーは、キューの管理、Streams に関連するパッケージ内のサブプログラムの実行、ルール・セットおよびルールの作成、データ・ディクショナリ・ビューとキュー表の問合せによる Streams 環境の監視を行うことができます。このユーザー用に、異なる名前を選択できます。

注意：

- セキュリティを確保するために、Streams 管理者には strmadminpw 以外のパスワードを使用してください。
- Streams 管理者には、SELECT_CATALOG_ROLE は不要です。この例で付与されているのは、Streams 管理者が環境を簡単に監視できるようにするためです。
- Oracle Enterprise Manager の Streams ツールを使用する予定の場合は、Streams 管理者にこの手順で示す権限の他に SELECT ANY DICTIONARY 権限を付与します。
- ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on mult1.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
                     QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM        TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM            TO strmadmin;
GRANT EXECUTE ON DBMS_CAPTURE_ADM      TO strmadmin;
GRANT EXECUTE ON DBMS_PROPAGATION_ADM  TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM      TO strmadmin;

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
```

```
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

/*
```

手順 5 mult1.net での Streams キューの作成

mult1.net に Streams 管理者として接続します。

```
*/

CONNECT strmadmin/strmadminpw@mult1.net

/*
```

SET_UP_QUEUE プロシージャを実行して、mult1.net でキュー streams_queue を作成します。このキューは、このデータベースで適用される変更と、他のデータベースに伝播される変更を保持することで、Streams キューとして機能します。

SET_UP_QUEUE プロシージャを実行すると、次のアクションが実行されます。

- キュー表 streams_queue_table が作成されます。このキュー表の所有者は Streams 管理者（strmadmin）であり、このユーザーのデフォルト記憶域が使用されます。
- Streams 管理者（strmadmin）が所有するキュー streams_queue が作成されます。
- キューが起動されます。

```
*/

EXEC  DBMS_STREAMS_ADM.SET_UP_QUEUE();

/*
```

手順 6 mult1.net でのデータベース・リンクの作成

現行のデータベースから環境内の他のデータベースへのデータベース・リンクを作成します。

```
*/

CREATE DATABASE LINK mult2.net CONNECT TO strmadmin
   IDENTIFIED BY strmadminpw USING 'mult2.net';

CREATE DATABASE LINK mult3.net CONNECT TO strmadmin
   IDENTIFIED BY strmadminpw USING 'mult3.net';

/*
```

手順 7 最終時刻による競合解消のための mult1.net の表の準備

この例では、トランザクションの最終時刻に基づいて競合解消を実行できるように、hr スキーマ内の表を構成します。

mult1.net に hr ユーザーとして接続します。

```
*/

CONNECT hr/hr@mult1.net

/*
```

hr スキーマ内の各表に time 列を追加します。

```
*/

ALTER TABLE hr.countries ADD (time TIMESTAMP WITH TIME ZONE);
ALTER TABLE hr.departments ADD (time TIMESTAMP WITH TIME ZONE);
ALTER TABLE hr.employees ADD (time TIMESTAMP WITH TIME ZONE);
ALTER TABLE hr.job_history ADD (time TIMESTAMP WITH TIME ZONE);
ALTER TABLE hr.jobs ADD (time TIMESTAMP WITH TIME ZONE);
ALTER TABLE hr.locations ADD (time TIMESTAMP WITH TIME ZONE);
ALTER TABLE hr.regions ADD (time TIMESTAMP WITH TIME ZONE);

/*
```

hr スキーマ内の表ごとに、トランザクションによって挿入または更新される各行について、トランザクションの時刻を挿入するためのトリガーを作成します。

```
*/

CREATE OR REPLACE TRIGGER hr.insert_time_countries
BEFORE
  INSERT OR UPDATE ON hr.countries FOR EACH ROW
BEGIN
   -- Consider time synchronization problems. The previous update to this
   -- row may have originated from a site with a clock time ahead of the
   -- local clock time.
   IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
     :NEW.TIME := SYSTIMESTAMP;
   ELSE
     :NEW.TIME := :OLD.TIME + 1 / 86400;
   END IF;
END;
/

CREATE OR REPLACE TRIGGER hr.insert_time_departments
BEFORE
  INSERT OR UPDATE ON hr.departments FOR EACH ROW
BEGIN
   IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
     :NEW.TIME := SYSTIMESTAMP;
   ELSE
     :NEW.TIME := :OLD.TIME + 1 / 86400;
   END IF;
END;
/

CREATE OR REPLACE TRIGGER hr.insert_time_employees
BEFORE
  INSERT OR UPDATE ON hr.employees FOR EACH ROW
BEGIN
   IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
     :NEW.TIME := SYSTIMESTAMP;
   ELSE
     :NEW.TIME := :OLD.TIME + 1 / 86400;
   END IF;
END;
/

CREATE OR REPLACE TRIGGER hr.insert_time_job_history
BEFORE
  INSERT OR UPDATE ON hr.job_history FOR EACH ROW
```

```
BEGIN
   IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
     :NEW.TIME := SYSTIMESTAMP;
   ELSE
     :NEW.TIME := :OLD.TIME + 1 / 86400;
   END IF;
END;
/

CREATE OR REPLACE TRIGGER hr.insert_time_jobs
BEFORE
  INSERT OR UPDATE ON hr.jobs FOR EACH ROW
BEGIN
   IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
     :NEW.TIME := SYSTIMESTAMP;
   ELSE
     :NEW.TIME := :OLD.TIME + 1 / 86400;
   END IF;
END;
/

CREATE OR REPLACE TRIGGER hr.insert_time_locations
BEFORE
  INSERT OR UPDATE ON hr.locations FOR EACH ROW
BEGIN
   IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
     :NEW.TIME := SYSTIMESTAMP;
   ELSE
     :NEW.TIME := :OLD.TIME + 1 / 86400;
   END IF;
END;
/

CREATE OR REPLACE TRIGGER hr.insert_time_regions
BEFORE
  INSERT OR UPDATE ON hr.regions FOR EACH ROW
BEGIN
   IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
     :NEW.TIME := SYSTIMESTAMP;
   ELSE
     :NEW.TIME := :OLD.TIME + 1 / 86400;
   END IF;
END;
/

/*
```

手順 8 mult2.net で LogMiner の表に使用する代替表領域の作成

デフォルトでは、LogMiner の表は SYSTEM 表領域にありますが、変更を取得するために取得プロセスを起動すると、SYSTEM 表領域にこれらの表に使用する領域が足りなくなる場合があります。したがって、LogMiner の表に使用する代替表領域を作成する必要があります。

関連項目： 2-18 ページ「LogMiner の表のための代替表領域」

mult2.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@mult2.net AS SYSDBA

/*
```

LogMiner の表に使用する代替表領域を作成します。

注意： 各 ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

```
*/

ACCEPT tspace_name DEFAULT 'logmnrts' PROMPT 'Enter the name of the tablespace (for
example, logmnrts): '

ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the datafile
directory (for example, /usr/oracle/dbs): '

ACCEPT db_file_name DEFAULT 'logmnrts.dbf' PROMPT 'Enter the name of the datafile
(for example, logmnrts.dbf): '

CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'
  SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');

/*
```

手順 9 mult2.net でのユーザーの設定

Streams 管理者 strmadmin を作成し、このユーザーに必要な権限を付与します。これらの権限を付与されたユーザーは、キューの管理、Streams に関連するパッケージ内のサブプログラムの実行、ルール・セットおよびルールの作成、データ・ディクショナリ・ビューとキュー表の問合せによる Streams 環境の監視を行うことができます。このユーザー用に、異なる名前を選択できます。

注意：

- セキュリティを確保するために、Streams 管理者には strmadminpw 以外のパスワードを使用してください。
- Streams 管理者には、SELECT_CATALOG_ROLE は不要です。この例で付与されているのは、Streams 管理者が環境を簡単に監視できるようにするためです。
- Oracle Enterprise Manager の Streams ツールを使用する予定の場合は、Streams 管理者にこの手順で示す権限の他に SELECT ANY DICTIONARY 権限を付与します。
- ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on mult2.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
                     QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM        TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM            TO strmadmin;
GRANT EXECUTE ON DBMS_CAPTURE_ADM      TO strmadmin;
GRANT EXECUTE ON DBMS_FLASHBACK        TO strmadmin;
GRANT EXECUTE ON DBMS_PROPAGATION_ADM  TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM      TO strmadmin;
```

```
BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

/*
```

手順 10 mult2.net での Streams キューの作成

mult2.net に Streams 管理者として接続します。

```
*/

CONNECT strmadmin/strmadminpw@mult2.net

/*
```

SET_UP_QUEUE プロシージャを実行して、mult2.net でキュー streams_queue を作成します。このキューは、このデータベースで適用される変更と、他のデータベースに伝播される変更を保持することで、Streams キューとして機能します。

SET_UP_QUEUE プロシージャを実行すると、次のアクションが実行されます。

- キュー表 streams_queue_table が作成されます。このキュー表の所有者は Streams 管理者（strmadmin）であり、このユーザーのデフォルト記憶域が使用されます。
- Streams 管理者（strmadmin）が所有するキュー streams_queue が作成されます。
- キューが起動されます。

```
*/

EXEC  DBMS_STREAMS_ADM.SET_UP_QUEUE();

/*
```

手順 11 mult2.net でのデータベース・リンクの作成

現行のデータベースから環境内の他のデータベースへのデータベース・リンクを作成します。

```
*/

CREATE DATABASE LINK mult1.net CONNECT TO strmadmin
   IDENTIFIED BY strmadminpw USING 'mult1.net';

CREATE DATABASE LINK mult3.net CONNECT TO strmadmin
   IDENTIFIED BY strmadminpw USING 'mult3.net';

/*
```

手順 12 mult2.net の hr スキーマにあるすべての表の削除

この例では、mult1.net からエクスポートして mult2.net にインポートすることで、mult2.net で hr スキーマ内の表をインスタンス化する方法を示します。この例のインスタンス化部分を正常に機能させるには、mult2.net で hr スキーマ内の表を削除する必要があります。

> **注意：** この手順に従って mult2.net で hr スキーマ内の表をすべて削除した後に、この例の残りの項の説明に従って hr スキーマを再インスタンス化する必要があります。hr スキーマが Oracle データベースに存在しない場合は、Oracle マニュアル・セットに記載されている一部の例が失敗する可能性があります。

mult2.net に hr として接続します。

```
*/

CONNECT hr/hr@mult2.net

/*
```

mult2.net データベースで hr スキーマ内の表をすべて削除します。

```
*/

DROP TABLE hr.countries CASCADE CONSTRAINTS;
DROP TABLE hr.departments CASCADE CONSTRAINTS;
DROP TABLE hr.employees CASCADE CONSTRAINTS;
DROP TABLE hr.job_history CASCADE CONSTRAINTS;
DROP TABLE hr.jobs CASCADE CONSTRAINTS;
```

```
DROP TABLE hr.locations CASCADE CONSTRAINTS;
DROP TABLE hr.regions CASCADE CONSTRAINTS;

/*
```

手順 13 mult3.net で LogMiner の表に使用する代替表領域の作成

デフォルトでは、LogMiner の表は SYSTEM 表領域にありますが、変更を取得するために取得プロセスを起動すると、SYSTEM 表領域にこれらの表に使用する領域が足りなくなる場合があります。したがって、LogMiner の表に使用する代替表領域を作成する必要があります。

関連項目： 2-18 ページ「LogMiner の表のための代替表領域」

mult3.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@mult3.net AS SYSDBA

/*
```

LogMiner の表に使用する代替表領域を作成します。

注意： 各 ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

```
*/

ACCEPT tspace_name DEFAULT 'logmnrts' PROMPT 'Enter the name of the tablespace (for
example, logmnrts): '

ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the datafile
directory (for example, /usr/oracle/dbs): '

ACCEPT db_file_name DEFAULT 'logmnrts.dbf' PROMPT 'Enter the name of the datafile
(for example, logmnrts.dbf): '

CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'
  SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');

/*
```

手順 14 mult3.net でのユーザーの設定

Streams 管理者 strmadmin を作成し、このユーザーに必要な権限を付与します。これらの権限を付与されたユーザーは、キューの管理、Streams に関連するパッケージ内のサブプログラムの実行、ルール・セットおよびルールの作成、データ・ディクショナリ・ビューとキュー表の問合せによる Streams 環境の監視を行うことができます。このユーザー用に、異なる名前を選択できます。

注意：

- セキュリティを確保するために、Streams 管理者には strmadminpw 以外のパスワードを使用してください。
- Streams 管理者には、SELECT_CATALOG_ROLE は不要です。この例で付与されているのは、Streams 管理者が環境を簡単に監視できるようにするためです。
- Oracle Enterprise Manager の Streams ツールを使用する予定の場合は、Streams 管理者にこの手順で示す権限の他に SELECT ANY DICTIONARY 権限を付与します。
- ACCEPT コマンドは、スクリプトに 1 行で指定する必要があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on mult3.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
                     QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM        TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM            TO strmadmin;
GRANT EXECUTE ON DBMS_CAPTURE_ADM      TO strmadmin;
GRANT EXECUTE ON DBMS_FLASHBACK        TO strmadmin;
GRANT EXECUTE ON DBMS_PROPAGATION_ADM  TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM      TO strmadmin;
```

```
BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

/*
```

手順 15 mult3.net での Streams キューの作成

mult3.net に Streams 管理者として接続します。

```
*/

CONNECT strmadmin/strmadminpw@mult3.net

/*
```

SET_UP_QUEUE プロシージャを実行して、mult3.net でキュー streams_queue を作成します。このキューは、このデータベースで適用される変更と、他のデータベースに伝播される変更を保持することで、Streams キューとして機能します。

SET_UP_QUEUE プロシージャを実行すると、次のアクションが実行されます。

- キュー表 streams_queue_table が作成されます。このキュー表の所有者は Streams 管理者（strmadmin）であり、このユーザーのデフォルト記憶域が使用されます。
- Streams 管理者（strmadmin）が所有するキュー streams_queue が作成されます。
- キューが起動されます。

```
*/

EXEC  DBMS_STREAMS_ADM.SET_UP_QUEUE();

/*
```

手順 16 mult3.net でのデータベース・リンクの作成

現行のデータベースから環境内の他のデータベースへのデータベース・リンクを作成します。

```
*/

CREATE DATABASE LINK mult1.net CONNECT TO strmadmin
   IDENTIFIED BY strmadminpw USING 'mult1.net';

CREATE DATABASE LINK mult2.net CONNECT TO strmadmin
   IDENTIFIED BY strmadminpw USING 'mult2.net';

/*
```

手順 17 mult3.net の hr スキーマにあるすべての表の削除

この例では、mult1.net からエクスポートして mult3.net にインポートすることで、mult3.net で hr スキーマ内の表をインスタンス化する方法を示します。この例のインスタンス化部分を正常に機能させるには、mult3.net で hr スキーマ内の表を削除する必要があります。

> **注意：** この手順に従って mult3.net で hr スキーマ内の表をすべて削除した後に、この例の残りの項の説明に従って hr スキーマを再インスタンス化する必要があります。hr スキーマが Oracle データベースに存在しない場合は、Oracle マニュアル・セットに記載されている一部の例が失敗する可能性があります。

mult3.net に hr として接続します。

```
*/

CONNECT hr/hr@mult3.net

/*
```

mult3.net データベースで hr スキーマ内の表をすべて削除します。

```
*/

DROP TABLE hr.countries CASCADE CONSTRAINTS;
DROP TABLE hr.departments CASCADE CONSTRAINTS;
DROP TABLE hr.employees CASCADE CONSTRAINTS;
DROP TABLE hr.job_history CASCADE CONSTRAINTS;
DROP TABLE hr.jobs CASCADE CONSTRAINTS;
```

```
DROP TABLE hr.locations CASCADE CONSTRAINTS;
DROP TABLE hr.regions CASCADE CONSTRAINTS;

/*
```

手順 18 スプール結果のチェック
このスクリプトの完了後に streams_setup_mult.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT ******************************/
```

複数のデータベースからのデータを共有するためのスクリプトの例

次の手順に従って、複数のデータベースからの情報を共有する Streams 環境を構成します。

1. 出力の表示と結果のスプーリング
2. mult1.net でのサプリメンタル・ロギングの指定
3. mult1.net での取得プロセスの作成
4. mult1.net での各ソース・データベース用の適用プロセスの作成
5. mult1.net の各適用プロセスの適用ユーザーとしての hr の指定
6. hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与
7. mult1.net での最終時刻に基づく競合解消の構成
8. mult1.net での伝播の構成
9. mult2.net での取得プロセスの作成
10. 他のデータベースでの mult2.net 用インスタンス化 SCN の設定
11. mult2.net での各ソース・データベース用の適用プロセスの作成
12. mult2.net の各適用プロセスの適用ユーザーとしての hr の指定
13. hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与
14. mult2.net での伝播の構成
15. mult3.net での取得プロセスの作成
16. 他のデータベースでの mult3.net 用インスタンス化 SCN の設定

17. mult3.net での各ソース・データベース用の適用プロセスの作成
18. mult3.net の各適用プロセスの適用ユーザーとしての hr の指定
19. hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与
20. mult3.net での伝播の構成
21. mult2.net と mult3.net での hr スキーマのインスタンス化
22. mult2.net での最終時刻に基づく競合解消の構成
23. mult2.net での適用プロセスの起動
24. mult3.net での最終時刻に基づく競合解消の構成
25. mult3.net での適用プロセスの起動
26. mult1.net での適用プロセスの起動
27. mult1.net での取得プロセスの起動
28. mult2.net での取得プロセスの起動
29. mult3.net での取得プロセスの起動
30. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 23-56 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************* BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL streams_mult.out

/*
```

手順 2 mult1.net でのサプリメンタル・ロギングの指定

mult1.net に SYS ユーザーとして接続します。

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@mult1.net AS SYSDBA

/*
```

23-29 ページの「mult1.net での最終時刻に基づく競合解消の構成」で指定されているとおり、各表の主キーを含む無条件のサプリメンタル・ログ・グループと、各表の列リストを指定します。

注意：

- 便宜上、この例には、各表の主キー列と、1 つの無条件ログ・グループで更新の競合解消に使用される列が含まれています。各表の主キー列を無条件のログ・グループに含めて、更新の競合解消に使用する列を条件付きのログ・グループに含めるように選択することもできます。
- この例の mult2.net および mult3.net でサプリメンタル・ロギングを明示的に指定する必要はありません。この例の後半で示されるように、hr スキーマのエクスポート / インポートを使用してこれらのデータベースで表をインスタンス化する場合、mult1.net でのサプリメンタル・ロギングの指定は mult2.net および mult3.net で保持されます。

関連項目：

- 2-10 ページ「Streams 環境内のサプリメンタル・ロギング」
- 12-8 ページ「ソース・データベースでのサプリメンタル・ロギングの指定」

```
*/

ALTER TABLE hr.countries ADD SUPPLEMENTAL LOG GROUP log_group_countries
  (country_id, country_name, region_id, time) ALWAYS;

ALTER TABLE hr.departments ADD SUPPLEMENTAL LOG GROUP log_group_departments
  (department_id, department_name, manager_id, location_id, time) ALWAYS;

ALTER TABLE hr.employees ADD SUPPLEMENTAL LOG GROUP log_group_employees
  (employee_id, first_name, last_name, email, phone_number, hire_date, job_id,
   salary, commission_pct, manager_id, department_id, time) ALWAYS;
```

```
ALTER TABLE hr.jobs ADD SUPPLEMENTAL LOG GROUP log_group_jobs
  (job_id, job_title, min_salary, max_salary, time) ALWAYS;

ALTER TABLE hr.job_history ADD SUPPLEMENTAL LOG GROUP log_group_job_history
  (employee_id, start_date, end_date, job_id, department_id, time) ALWAYS;

ALTER TABLE hr.locations ADD SUPPLEMENTAL LOG GROUP log_group_locations
  (location_id, street_address, postal_code, city, state_province,
   country_id, time) ALWAYS;

ALTER TABLE hr.regions ADD SUPPLEMENTAL LOG GROUP log_group_regions
  (region_id, region_name, time) ALWAYS;

/*
```

手順 3 mult1.net での取得プロセスの作成

mult1.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@mult1.net

/*
```

mult1.net で、hr スキーマ全体に対する変更を取得する取得プロセスを作成します。この手順を完了すると、ユーザーは mult1.net で hr スキーマ内の表を変更できます。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name   => 'hr',
    streams_type  => 'capture',
    streams_name  => 'capture_hr',
    queue_name    => 'strmadmin.streams_queue',
    include_dml   =>  true,
    include_ddl   =>  true);
END;
/

/*
```

手順 4 mult1.net での各ソース・データベース用の適用プロセスの作成

mult1.net を、mult2.net の hr スキーマに変更を適用するように構成します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name     => 'hr',
    streams_type    => 'apply',
    streams_name    => 'apply_from_mult2',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'mult2.net');
END;
/

/*
```

mult1.net を、mult3.net の hr スキーマに変更を適用するように構成します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name     => 'hr',
    streams_type    => 'apply',
    streams_name    => 'apply_from_mult3',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'mult3.net');
END;
/

/*
```

手順 5 mult1.net の各適用プロセスの適用ユーザーとしての hr の指定

この例では、このデータベースで適用プロセスによって変更が適用される全データベース・オブジェクトを、hr ユーザーが所有しているものとします。したがって、hr はすでにこれらのデータベース・オブジェクトを変更するための権限を持っており、hr を適用ユーザーにすると便利です。

前述の手順で適用プロセスを作成するときには、Streams 管理者 strmadmin が適用プロセスの作成プロシージャを実行したため、デフォルトで strmadmin が適用ユーザーとして指定されました。hr を適用ユーザーとして指定するかわりに、strmadmin を適用ユーザーのままにすることもできます。ただし、その場合は strmadmin に、変更が適用される全データベース・オブジェクトに対する権限と、適用プロセスで使用される全ユーザー・プロシージャを実行する権限を付与する必要があります。適用プロセスによって複数のスキーマ内のデータベース・オブジェクトに変更が適用される環境では、Streams 管理者を適用ユーザーとして使用する方が便利な場合があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'apply_from_mult2',
    apply_user => 'hr');
END;
/

BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'apply_from_mult3',
    apply_user => 'hr');
END;
/

/*
```

手順 6 hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与

前述の手順で hr ユーザーを適用ユーザーとして指定したため、hr ユーザーには各適用プロセスで使用されるルール・セットの実行権限が必要です。

```
*/

DECLARE
   rs_name  VARCHAR2(64);   -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
    INTO rs_name
    FROM DBA_APPLY
    WHERE APPLY_NAME='APPLY_FROM_MULT2';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee     => 'hr');
END;
/

DECLARE
   rs_name  VARCHAR2(64);   -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
    INTO rs_name
    FROM DBA_APPLY
    WHERE APPLY_NAME='APPLY_FROM_MULT3';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee     => 'hr');
END;
/

/*
```

手順 7 mult1.net での最終時刻に基づく競合解消の構成

hr スキーマ内の表ごとに、更新の競合ハンドラを指定します。表ごとに、MAXIMUM 競合ハンドラ用の解消列として time 列を指定します。更新の競合が発生すると、指定した更新の競合ハンドラによって、最終（またはそれ以降の）時刻を持つトランザクションが適用され、それ以前の時刻を持つトランザクションが廃棄されます。この例では主キー値に更新がないことを想定しているため、各表の列リストに主キーは含まれていません。

```
*/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'country_name';
  cols(2) := 'region_id';
  cols(3) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.countries',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'department_name';
  cols(2) := 'manager_id';
  cols(3) := 'location_id';
  cols(4) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.departments',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/
```

```
DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1)  := 'first_name';
  cols(2)  := 'last_name';
  cols(3)  := 'email';
  cols(4)  := 'phone_number';
  cols(5)  := 'hire_date';
  cols(6)  := 'job_id';
  cols(7)  := 'salary';
  cols(8)  := 'commission_pct';
  cols(9)  := 'manager_id';
  cols(10) := 'department_id';
  cols(11) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.employees',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'job_title';
  cols(2) := 'min_salary';
  cols(3) := 'max_salary';
  cols(4) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.jobs',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/
```

```
DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'employee_id';
  cols(2) := 'start_date';
  cols(3) := 'end_date';
  cols(4) := 'job_id';
  cols(5) := 'department_id';
  cols(6) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.job_history',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'street_address';
  cols(2) := 'postal_code';
  cols(3) := 'city';
  cols(4) := 'state_province';
  cols(5) := 'country_id';
  cols(6) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.locations',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/
```

```
DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'region_name';
  cols(2) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.regions',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

/*
```

手順 8 mult1.net での伝播の構成

hr スキーマ内の DML 変更と DDL 変更について、mult1.net のキューから mult2.net のキューへの伝播を構成してスケジュールします。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name                => 'hr',
    streams_name               => 'mult1_to_mult2',
    source_queue_name          => 'strmadmin.streams_queue',
    destination_queue_name     => 'strmadmin.streams_queue@mult2.net',
    include_dml                =>  true,
    include_ddl                =>  true,
    source_database            => 'mult1.net');
END;
/

/*
```

hr スキーマ内の DML 変更と DDL 変更について、mult1.net のキューから mult3.net のキューへの伝播を構成してスケジュールします。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name                => 'hr',
    streams_name               => 'mult1_to_mult3',
    source_queue_name          => 'strmadmin.streams_queue',
    destination_queue_name     => 'strmadmin.streams_queue@mult3.net',
    include_dml                =>  true,
    include_ddl                =>  true,
    source_database            => 'mult1.net');
END;
/

/*
```

手順 9 mult2.net での取得プロセスの作成

mult2.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@mult2.net

/*
```

mult2.net で、hr スキーマ全体に対する変更を取得する取得プロセスを作成します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name   => 'hr',
    streams_type  => 'capture',
    streams_name  => 'capture_hr',
    queue_name    => 'strmadmin.streams_queue',
    include_dml   =>  true,
    include_ddl   =>  true);
END;
/

/*
```

手順 10 他のデータベースでの mult2.net 用インスタンス化 SCN の設定

この例では、hr スキーマはすべてのデータベースにすでに存在します。このスキーマ内の表は、手順 21 で mult2.net および mult3.net でインスタンス化されるまで、mult1.net にしか存在しません。インスタンス化には、mult1.net からの表のエクスポートが使用されます。これらのエクスポート / インポート操作によって、mult2.net および mult3.net で mult1.net 用のスキーマのインスタンス化 SCN が自動的に設定されます。

ただし、mult2.net および mult3.net 用のインスタンス化 SCN は、環境内の他のサイトで自動的に設定されることはありません。この手順では、mult1.net および mult3.net で mult2.net 用のスキーマのインスタンス化 SCN を手動で設定します。mult2.net の現行の SCN は、mult2.net で DBMS_FLASHBACK パッケージの GET_SYSTEM_CHANGE_NUMBER ファンクションを使用して取得されます。mult1.net および mult3.net では、この SCN を使用して DBMS_APPLY_ADM パッケージの SET_SCHEMA_INSTANTIATION_SCN プロシージャが実行されます。

SET_SCHEMA_INSTANTIATION_SCN プロシージャでは、スキーマについて適用プロセスによって無視される DDL LCR と適用される DDL LCR が制御されます。ソース・データベースからのスキーマ内のデータベース・オブジェクトに関する DDL LCR のコミット SCN が、一部の接続先データベースでそのデータベース・オブジェクトのインスタンス化 SCN 以下であれば、接続先データベースの適用プロセスでは DDL LCR が無視されます。それ以外の場合は、適用プロセスによって DDL LCR が適用されます。

mult2.net で表がインスタンス化される前に SET_SCHEMA_INSTANTIATION_SCN プロシージャを実行しており、ローカルの取得プロセスはすでに構成されているため、インスタンス化の後で各表に対して SET_TABLE_INSTANTIATION_SCN を実行する必要はありません。この例では、mult1.net および mult3.net の適用プロセスによって、この手順で取得した SCN より後にコミットされた SCN を持つ hr スキーマ内の表にトランザクションが適用されます。

注意 :

- 存在しないスキーマをインスタンス化する場合は、スキーマのインスタンス化 SCN のかわりにグローバル・インスタンス化 SCN を設定できます。
- インスタンス化 SCN を設定する前に表がインスタンス化される場合は、スキーマのインスタンス化 SCN とスキーマ内の各表のインスタンス化 SCN を設定する必要があります。

```
*/

DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@MULT1.NET(
    source_schema_name    => 'hr',
    source_database_name  => 'mult2.net',
    instantiation_scn     => iscn);
  DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@MULT3.NET(
    source_schema_name    => 'hr',
    source_database_name  => 'mult2.net',
    instantiation_scn     => iscn);
END;
/

/*
```

手順 11 mult2.net での各ソース・データベース用の適用プロセスの作成

mult2.net を、mult1.net の hr スキーマに変更を適用するように構成します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name     => 'hr',
    streams_type    => 'apply',
    streams_name    => 'apply_from_mult1',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'mult1.net');
END;
/

/*
```

mult2.net を、mult3.net の hr スキーマに変更を適用するように構成します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name     => 'hr',
    streams_type    => 'apply',
    streams_name    => 'apply_from_mult3',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'mult3.net');
END;
/

/*
```

手順 12 mult2.net の各適用プロセスの適用ユーザーとしての hr の指定

この例では、このデータベースで適用プロセスによって変更が適用される全データベース・オブジェクトを、hr ユーザーが所有しているものとします。したがって、hr はすでにこれらのデータベース・オブジェクトを変更するための権限を持っており、hr を適用ユーザーにすると便利です。

前述の手順で適用プロセスを作成するときには、Streams 管理者 strmadmin が適用プロセスの作成プロシージャを実行したため、デフォルトで strmadmin が適用ユーザーとして指定されました。hr を適用ユーザーとして指定するかわりに、strmadmin を適用ユーザーのままにすることもできます。ただし、その場合は strmadmin に、変更が適用される全データベース・オブジェクトに対する権限と、適用プロセスで使用される全ユーザー・プロシージャを実行する権限を付与する必要があります。適用プロセスによって複数のスキーマ内のデータベース・オブジェクトに変更が適用される環境では、Streams 管理者を適用ユーザーとして使用する方が便利な場合があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'apply_from_mult1',
    apply_user => 'hr');
END;
/
```

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'apply_from_mult3',
    apply_user => 'hr');
END;
/

/*
```

手順 13 hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与

前述の手順で hr ユーザーを適用ユーザーとして指定したため、hr ユーザーには各適用プロセスで使用されるルール・セットの実行権限が必要です。

```
*/

DECLARE
   rs_name  VARCHAR2(64);   -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
    INTO rs_name
    FROM DBA_APPLY
    WHERE APPLY_NAME='APPLY_FROM_MULT1';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee     => 'hr');
END;
/

DECLARE
   rs_name  VARCHAR2(64);   -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
    INTO rs_name
    FROM DBA_APPLY
    WHERE APPLY_NAME='APPLY_FROM_MULT3';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee     => 'hr');
END;
/

/*
```

手順 14 mult2.net での伝播の構成

hr スキーマ内の DML 変更と DDL 変更について、mult2.net のキューから mult1.net のキューへの伝播を構成してスケジュールします。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name             => 'hr',
    streams_name            => 'mult2_to_mult1',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@mult1.net',
    include_dml             =>  true,
    include_ddl             =>  true,
    source_database         => 'mult2.net');
END;
/

/*
```

hr スキーマ内の DML 変更と DDL 変更について、mult2.net のキューから mult3.net のキューへの伝播を構成してスケジュールします。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name             => 'hr',
    streams_name            => 'mult2_to_mult3',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@mult3.net',
    include_dml             =>  true,
    include_ddl             =>  true,
    source_database         => 'mult2.net');
END;
/

/*
```

手順 15 mult3.net での取得プロセスの作成

mult3.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@mult3.net

/*
```

mult3.net で、hr スキーマ全体に対する変更を取得する取得プロセスを作成します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name   => 'hr',
    streams_type  => 'capture',
    streams_name  => 'capture_hr',
    queue_name    => 'strmadmin.streams_queue',
    include_dml   =>  true,
    include_ddl   =>  true);
END;
/

/*
```

手順 16 他のデータベースでの mult3.net 用インスタンス化 SCN の設定

この例では、hr スキーマはすべてのデータベースにすでに存在します。このスキーマ内の表は、手順 21 で mult2.net および mult3.net でインスタンス化されるまで、mult1.net にしか存在しません。インスタンス化には、mult1.net からの表のエクスポートが使用されます。これらのエクスポート / インポート操作によって、mult2.net および mult3.net で mult1.net 用のスキーマのインスタンス化 SCN が自動的に設定されます。

ただし、mult2.net および mult3.net 用のインスタンス化 SCN は、環境内の他のサイトで自動的に設定されることはありません。この手順では、mult1.net および mult2.net で mult3.net 用のスキーマのインスタンス化 SCN を手動で設定します。mult3.net の現行の SCN は、mult3.net で DBMS_FLASHBACK パッケージの GET_SYSTEM_CHANGE_NUMBER ファンクションを使用して取得されます。mult1.net および mult2.net では、この SCN を使用して DBMS_APPLY_ADM パッケージの SET_SCHEMA_INSTANTIATION_SCN プロシージャが実行されます。

SET_SCHEMA_INSTANTIATION_SCN プロシージャでは、スキーマについて適用プロセスによって無視される DDL LCR と適用される DDL LCR が制御されます。ソース・データベースからのスキーマ内のデータベース・オブジェクトに関する DDL LCR のコミット SCN が、一部の接続先データベースでそのデータベース・オブジェクトのインスタンス化 SCN 以下であれば、接続先データベースの適用プロセスでは DDL LCR が廃棄されます。それ以外の場合は、適用プロセスによって DDL LCR が適用されます。

mult3.net で表がインスタンス化される前に SET_SCHEMA_INSTANTIATION_SCN プロシージャを実行しており、ローカルの取得プロセスはすでに構成されているため、インスタンス化の後で各表に対して SET_TABLE_INSTANTIATION_SCN を実行する必要はありません。この例では、mult1.net および mult2.net の適用プロセスによって、この手順で取得した SCN より後にコミットされた SCN を持つトランザクションが、hr スキーマ内の表に適用されます。

注意：

- 存在しないスキーマをインスタンス化する場合は、スキーマのインスタンス化 SCN のかわりにグローバル・インスタンス化 SCN を設定できます。
- インスタンス化 SCN を設定する前に表がインスタンス化される場合は、スキーマのインスタンス化 SCN とスキーマ内の各表のインスタンス化 SCN を設定する必要があります。

```
*/

DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@MULT1.NET(
    source_schema_name    => 'hr',
    source_database_name  => 'mult3.net',
    instantiation_scn     => iscn);
  DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@MULT2.NET(
    source_schema_name    => 'hr',
    source_database_name  => 'mult3.net',
    instantiation_scn     => iscn);
END;
/

/*
```

手順 17 mult3.net での各ソース・データベース用の適用プロセスの作成

mult3.net を、mult1.net の hr スキーマに変更を適用するように構成します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name     => 'hr',
    streams_type    => 'apply',
    streams_name    => 'apply_from_mult1',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'mult1.net');
END;
/

/*
```

mult3.net を、mult2.net の hr スキーマに変更を適用するように構成します。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name     => 'hr',
    streams_type    => 'apply',
    streams_name    => 'apply_from_mult2',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    source_database => 'mult2.net');
END;
/

/*
```

手順 18 mult3.net の各適用プロセスの適用ユーザーとしての hr の指定

この例では、このデータベースで適用プロセスによって変更が適用される全データベース・オブジェクトを、hr ユーザーが所有しているものとします。したがって、hr はすでにこれらのデータベース・オブジェクトを変更するための権限を持っており、hr を適用ユーザーにすると便利です。

前述の手順で適用プロセスを作成するときには、Streams 管理者 strmadmin が適用プロセスの作成プロシージャを実行したため、デフォルトで strmadmin が適用ユーザーとして指定されました。hr を適用ユーザーとして指定するかわりに、strmadmin を適用ユーザーのままにすることもできます。ただし、その場合は strmadmin に、変更が適用される全データベース・オブジェクトに対する権限と、適用プロセスで使用される全ユーザー・プロシージャを実行する権限を付与する必要があります。適用プロセスによって複数のスキーマ内のデータベース・オブジェクトに変更が適用される環境では、Streams 管理者を適用ユーザーとして使用する方が便利な場合があります。

関連項目： 11-2 ページ「Streams 管理者の構成」

```
*/

BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'apply_from_mult1',
    apply_user => 'hr');
END;
/

BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'apply_from_mult2',
    apply_user => 'hr');
END;
/

/*
```

手順 19 hr ユーザーに対する適用プロセスのルール・セットの実行権限の付与

前述の手順で hr ユーザーを適用ユーザーとして指定したため、hr ユーザーには各適用プロセスで使用されるルール・セットの実行権限が必要です。

```
*/

DECLARE
   rs_name  VARCHAR2(64);   -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
    INTO rs_name
    FROM DBA_APPLY
    WHERE APPLY_NAME='APPLY_FROM_MULT1';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee     => 'hr');
END;
/

DECLARE
   rs_name  VARCHAR2(64);   -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
    INTO rs_name
    FROM DBA_APPLY
    WHERE APPLY_NAME='APPLY_FROM_MULT2';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee     => 'hr');
END;
/

/*
```

手順 20 mult3.net での伝播の構成

hr スキーマ内の DML 変更と DDL 変更について、mult3.net のキューから mult1.net のキューへの伝播を構成してスケジュールします。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name             => 'hr',
    streams_name            => 'mult3_to_mult1',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@mult1.net',
    include_dml             =>  true,
    include_ddl             =>  true,
    source_database         => 'mult3.net');
END;
/

/*
```

hr スキーマ内の DML 変更と DDL 変更について、mult3.net のキューから mult2.net のキューへの伝播を構成してスケジュールします。

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name             => 'hr',
    streams_name            => 'mult3_to_mult2',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@mult2.net',
    include_dml             =>  true,
    include_ddl             =>  true,
    source_database         => 'mult3.net');
END;
/

/*
```

手順 21 mult2.net と mult3.net での hr スキーマのインスタンス化

mult1.net で異なるウィンドウを開き、mult2.net および mult3.net でインスタンス化するスキーマをエクスポートします。エクスポート・コマンドの実行時に、OBJECT_CONSISTENT エクスポート・パラメータが y に設定されていることを確認します。また、エクスポート中は、エクスポート対象のオブジェクトに対する DDL 変更が行われないように注意してください。

次にエクスポート・コマンドの例を示します。

```
exp hr/hr FILE=hr_schema.dmp OWNER=hr OBJECT_CONSISTENT=y
```

> **関連項目：**『Oracle9*i* データベース・ユーティリティ』でエクスポートの実行方法を参照してください。

```
*/

PAUSE Press <RETURN> to continue when the export is complete in the other window
that you opened.

/*
```

エクスポート・ダンプ・ファイル hr_schema.dmp を接続先データベースに送信します。この例では、接続先データベースは mult2.net および mult3.net です。

バイナリ FTP または他のなんらかの方法を使用して、エクスポート・ダンプ・ファイルを接続先データベースに送信できます。ファイルを送信するには、異なるウィンドウを開く必要がある場合があります。

```
*/

PAUSE Press <RETURN> to continue after transferring the dump file to all of the
other databases in the environment.

/*
```

別のウィンドウで、mult2.net データベースが稼働しているコンピュータに接続し、エクスポート・ダンプ・ファイル hr_schema.dmp をインポートして、mult2.net データベース内の表をインスタンス化します。mult2.net が稼働しているコンピュータには、telnet またはリモート・ログインを使用して接続できます。

インポート・コマンドの実行時には、STREAMS_INSTANTIATION インポート・パラメータが y に設定されていることを確認してください。このパラメータによって、インポートされる各オブジェクトのエクスポート SCN 情報が、インポート時に確実に記録されます。

また、インポートが完了して取得プロセスが作成されるまでは、接続先データベース（mult2.net）でインポート対象のスキーマ内の表に対する変更が行われないことを確認してください。

次にインポート・コマンドの例を示します。

```
imp hr/hr FILE=hr_schema.dmp FROMUSER=hr IGNORE=y COMMIT=y LOG=import.log
STREAMS_INSTANTIATION=y
```

> **関連項目：**『Oracle9*i* データベース・ユーティリティ』でインポートの実行方法を参照してください。

```
*/

PAUSE Press <RETURN> to continue after the import is complete at mult2.net.

/*
```

別のウィンドウで、mult3.net データベースが稼働しているコンピュータに接続し、エクスポート・ダンプ・ファイル hr_schema.dmp をインポートして、mult3.net データベース内の表をインスタンス化します。

mult3.net への接続後に、mult2.net の場合と同じ方法でインポートを実行します。

```
*/

PAUSE Press <RETURN> to continue after the import is complete at mult3.net.

/*
```

手順 22 mult2.net での最終時刻に基づく競合解消の構成

mult2.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@mult2.net

/*
```

hr スキーマ内の表ごとに、更新の競合ハンドラを指定します。表ごとに、MAXIMUM 競合ハンドラ用の解消列として time 列を指定します。更新の競合が発生すると、指定した更新の競合ハンドラによって、最終（またはそれ以降の）時刻を持つトランザクションが適用され、それ以前の時刻を持つトランザクションが廃棄されます。

```
*/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'country_name';
  cols(2) := 'region_id';
  cols(3) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.countries',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'department_name';
  cols(2) := 'manager_id';
  cols(3) := 'location_id';
  cols(4) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.departments',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1)  := 'first_name';
  cols(2)  := 'last_name';
  cols(3)  := 'email';
  cols(4)  := 'phone_number';
  cols(5)  := 'hire_date';
  cols(6)  := 'job_id';
  cols(7)  := 'salary';
  cols(8)  := 'commission_pct';
  cols(9)  := 'manager_id';
  cols(10) := 'department_id';
  cols(11) := 'time';
```

```
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.employees',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'job_title';
  cols(2) := 'min_salary';
  cols(3) := 'max_salary';
  cols(4) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.jobs',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'employee_id';
  cols(2) := 'start_date';
  cols(3) := 'end_date';
  cols(4) := 'job_id';
  cols(5) := 'department_id';
  cols(6) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.job_history',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'street_address';
  cols(2) := 'postal_code';
  cols(3) := 'city';
  cols(4) := 'state_province';
```

```
  cols(5) := 'country_id';
  cols(6) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.locations',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'region_name';
  cols(2) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.regions',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

/*
```

手順 23 mult2.net での適用プロセスの起動

エラーが発生しても無効化されないように、両方の適用プロセスの disable_on_error パラメータを n に設定して、mult2.net で両方の適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_from_mult1',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply_from_mult1');
END;
/
```

```
BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_from_mult3',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply_from_mult3');
END;
/

/*
```

手順 24 mult3.net での最終時刻に基づく競合解消の構成

mult3.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@mult3.net

/*
```

hr スキーマ内の表ごとに、更新の競合ハンドラを指定します。表ごとに、MAXIMUM 競合ハンドラ用の解消列として time 列を指定します。更新の競合が発生すると、指定した更新の競合ハンドラによって、最終（またはそれ以降の）時刻を持つトランザクションが適用され、それ以前の時刻を持つトランザクションが廃棄されます。

```
*/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'country_name';
  cols(2) := 'region_id';
  cols(3) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.countries',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/
```

```
DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'department_name';
  cols(2) := 'manager_id';
  cols(3) := 'location_id';
  cols(4) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.departments',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1)  := 'first_name';
  cols(2)  := 'last_name';
  cols(3)  := 'email';
  cols(4)  := 'phone_number';
  cols(5)  := 'hire_date';
  cols(6)  := 'job_id';
  cols(7)  := 'salary';
  cols(8)  := 'commission_pct';
  cols(9)  := 'manager_id';
  cols(10) := 'department_id';
  cols(11) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.employees',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'job_title';
  cols(2) := 'min_salary';
  cols(3) := 'max_salary';
  cols(4) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.jobs',
    method_name           =>  'MAXIMUM',
```

```
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'employee_id';
  cols(2) := 'start_date';
  cols(3) := 'end_date';
  cols(4) := 'job_id';
  cols(5) := 'department_id';
  cols(6) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.job_history',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'street_address';
  cols(2) := 'postal_code';
  cols(3) := 'city';
  cols(4) := 'state_province';
  cols(5) := 'country_id';
  cols(6) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.locations',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'region_name';
  cols(2) := 'time';
```

```
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name           =>  'hr.regions',
    method_name           =>  'MAXIMUM',
    resolution_column     =>  'time',
    column_list           =>  cols);
END;
/

/*
```

手順 25 mult3.net での適用プロセスの起動

エラーが発生しても無効化されないように、両方の適用プロセスの disable_on_error パラメータを n に設定して、mult3.net で両方の適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_from_mult1',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply_from_mult1');
END;
/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_from_mult2',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply_from_mult2');
END;
/

/*
```

手順 26 mult1.net での適用プロセスの起動

mult1.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@mult1.net

/*
```

エラーが発生しても無効化されないように、両方の適用プロセスの disable_on_error パラメータを n に設定して、mult1.net で両方の適用プロセスを起動します。

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_from_mult2',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply_from_mult2');
END;
/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_from_mult3',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply_from_mult3');
END;
/

/*
```

手順 27 mult1.net での取得プロセスの起動

mult1.net で取得プロセスを起動します。

```
*/

BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name  => 'capture_hr');
END;
/

/*
```

手順 28 mult2.net での取得プロセスの起動

mult2.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@mult2.net

/*
```

mult2.net で取得プロセスを起動します。

```
*/

BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name  => 'capture_hr');
END;
/

/*
```

手順 29 mult3.net での取得プロセスの起動

mult3.net に strmadmin ユーザーとして接続します。

```
*/

CONNECT strmadmin/strmadminpw@mult3.net

/*
```

mult3.net で取得プロセスを起動します。

```
*/

BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name  => 'capture_hr');
END;
/

SET ECHO OFF

/*
```

手順 30 スプール結果のチェック

このスクリプトの完了後に streams_mult.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT ******************************/
```

hr スキーマでの表に対する DML 変更および DDL 変更

環境内のどのデータベースでも、hr スキーマ内の表に対する DML 変更および DDL 変更を行うことができます。これらの変更は、環境内の他のデータベースにレプリケートされ、問合せを実行してレプリケートされたデータを表示できます。

たとえば、次の手順に従って、mult1.net および mult2.net で hr.employees 表に対する DML 変更を行います。以前構成した更新の競合ハンドラが更新の競合を解消することを確認するには、2 つのデータベースの同じ行に変更を加え、ほぼ同時に変更をコミットします。次に、環境内の各データベースで hr.employees 表を問い合せて、変更が取得され、伝播され、正しく適用されたことを確認します。

mult3.net で hr.jobs に対する DDL 変更を行うことができます。このとき、変更が mult3.net で取得されたこと、環境内の他のデータベースに伝播されたこと、そしてこれらのデータベースに適用されたことを確認します。

手順 1 mult.net および mult2.net での hr.employees に対する DML 変更

次の変更を行います。DML 変更は、ほぼ同時にコミットするようにしますが、mult1.net で変更をコミットした後に mult2.net で変更をコミットします。

```
CONNECT hr/hr@mult1.net

UPDATE hr.employees SET salary=9000 WHERE employee_id=206;
COMMIT;

CONNECT hr/hr@mult2.net

UPDATE hr.employees SET salary=10000 WHERE employee_id=206;
COMMIT;
```

手順 2 mult3.net での hr.jobs 表の変更

job_title 列から job_name へ名前を変更することで、hr.jobs 表を変更します。

```
CONNECT hr/hr@mult3.net

ALTER TABLE hr.jobs RENAME COLUMN job_title TO job_name;
```

手順 3 各データベースでの hr.employees 表の問合せ

手順 1 で実行した変更を取得、伝播し、適用するのに必要な時間が経過した後、次の問合せを実行して UPDATE の変更が各データベースに適用されたことを確認します。

```
CONNECT hr/hr@mult1.net

SELECT salary FROM hr.employees WHERE employee_id=206;

CONNECT hr/hr@mult2.net

SELECT salary FROM hr.employees WHERE employee_id=206;

CONNECT hr/hr@mult3.net

SELECT salary FROM hr.employees WHERE employee_id=206;
```

すべての問合せで、salary の値が 10000 になるはずです。

手順 4 各データベースでの hr.jobs 表の記述

手順 2 で実行した変更を取得、伝播し、適用するのに必要な時間が経過した後、各データベースで hr.jobs 表を記述して、ALTER TABLE の変更が伝播され、正しく適用されたことを確認します。

```
CONNECT hr/hr@mult1.net

DESC hr.jobs

CONNECT hr/hr@mult2.net

DESC hr.jobs

CONNECT hr/hr@mult3.net

DESC hr.jobs
```

すべてのデータベースで、job_name が表の 2 番目の列として表示されるはずです。

24

ルールベースのアプリケーションの例

この章では、Oracle ルール・エンジンを使用するルールベースのアプリケーションについて説明します。

この章の内容は次のとおりです。

注意： この章の例は、Streams から独立しています。つまりこれらの例では、Streams の取得プロセス、伝播、あるいは適用プロセスはルール・エンジンのクライアントではなく、キューは使用されません。

関連項目：

- 第 5 章「ルール」
- 第 15 章「ルールおよびルールベースの変換の管理」
- 17-40 ページ「ルールおよびルールベースの変換の監視」

ルールベースのアプリケーションの概要

この章の各例では、顧客の問題を処理するルールベースのアプリケーションを作成します。アプリケーションではルールを使用し、新規の問題がレポートされた場合は、その優先度に基づいて、完了する必要のあるアクションを判断します。たとえば、アプリケーションでは、それぞれの問題をその優先度に基づいて社内の特定のセンターに割り当てます。

アプリケーションでは、これらのルールがルール・エンジンを使用して規定されます。サポートの問題に関連する情報を定義するために、評価コンテキスト evalctx が作成されます。ルールは前述の要件に基づいて作成され、ルール・セット rs に追加されます。

問題を割り当てるタスクは、ユーザー定義プロシージャ problem_dispatch によって実行されます。このプロシージャはルール・エンジンをコールしてルール・セット rs 内のルールを評価し、TRUE と評価されるルールに基づいて適切なアクションを実行します。

注意： これらの例を完了するには、COMPATIBLE 初期化パラメータを 9.2.0 以上に設定する必要があります。

明示的変数に格納された表以外のデータに関するルールの使用

この例では、ルールを使用して、明示的変数に格納されたデータを評価する方法について説明します。この例では、優先度に基づいて顧客の問題を処理します。この種の問題を処理するために次のルールが使用されます。

- 優先度が 3 以上のすべての問題をサンノゼ・センターに割り当てます。
- 優先度が 2 以下のすべての問題をニューヨーク・センターに割り当てます。
- 優先度が 1 の問題については、サポート担当副社長に警告を送信します。

評価コンテキストには、明示的変数 priority のみが含まれます。この明示的変数は、ディスパッチされる問題の優先度を参照します。この変数の値は、problem_dispatch プロシージャによって DBMS_RULE.EVALUATE プロシージャに渡されます。

この操作の手順は次のとおりです。

1. 出力の表示と結果のスプーリング
2. support ユーザーの作成
3. support ユーザーへのルールに関して必要なシステム権限の付与
4. evalctx 評価コンテキストの作成
5. 問題の優先度に対応するルールの作成
6. rs ルール・セットの作成
7. ルール・セットへのルールの追加
8. データ・ディクショナリの問合せ

9. problem_dispatch PL/SQL プロシージャの作成
10. サンプル問題のディスパッチ
11. スプール結果のチェック

> **注意：** このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 24-8 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************ BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL rules_stored_variables.out

/*
```

手順 2 support ユーザーの作成

```
*/

CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;

GRANT CONNECT, RESOURCE TO support IDENTIFIED BY support;

/*
```

手順 3 support ユーザーへのルールに関して必要なシステム権限の付与

```
*/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee      => 'support',
    grant_option => FALSE);
END;
/

/*
```

手順 4 evalctx 評価コンテキストの作成

```
*/

CONNECT support/support

SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET SERVEROUTPUT ON;
DECLARE
  vt SYS.RE$VARIABLE_TYPE_LIST;
BEGIN
  vt := SYS.RE$VARIABLE_TYPE_LIST(
    SYS.RE$VARIABLE_TYPE('priority', 'NUMBER', NULL, NULL));
  DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
    evaluation_context_name    => 'evalctx',
    variable_types             => vt,
    evaluation_context_comment => 'support problem definition');
END;
/

/*
```

手順 5 問題の優先度に対応するルールの作成

次のコードは、各ルールにアクション・コンテキストを 1 つ作成し、各アクション・コンテキストに名前 / 値ペアを 1 つ作成します。

```
*/

DECLARE
  ac  SYS.RE$NV_LIST;
BEGIN
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('San Jose'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r1',
    condition      => ':priority > 2',
    action_context => ac,
    rule_comment   => 'Low priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('New York'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r2',
    condition      => ':priority <= 2',
    action_context => ac,
    rule_comment   => 'High priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('ALERT', SYS.AnyData.CONVERTVARCHAR2('John Doe'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r3',
    condition      => ':priority = 1',
    action_context => ac,
    rule_comment   => 'Urgent problems');
END;
/

/*
```

手順 6 rs ルール・セットの作成

```
*/

BEGIN
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name      => 'rs',
    evaluation_context => 'evalctx',
    rule_set_comment   => 'support rules');
END;
/

/*
```

手順 7 ルール・セットへのルールの追加

```
*/

BEGIN
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r1',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r2',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r3',
    rule_set_name => 'rs');
END;
/

/*
```

手順 8 データ・ディクショナリの問合せ

この時点で、前述の手順で作成した評価コンテキスト、ルールおよびルール・セットを表示できます。

```
*/

SELECT * FROM USER_EVALUATION_CONTEXTS;

SELECT * FROM USER_RULES;

SELECT * FROM USER_RULE_SETS;

/*
```

手順 9 problem_dispatch PL/SQL プロシージャの作成

```
*/

CREATE OR REPLACE PROCEDURE problem_dispatch (priority NUMBER)
IS
    vv        SYS.RE$VARIABLE_VALUE;
    vvl       SYS.RE$VARIABLE_VALUE_LIST;
    truehits  SYS.RE$RULE_HIT_LIST;
    maybehits SYS.RE$RULE_HIT_LIST;
    ac        SYS.RE$NV_LIST;
    namearray SYS.RE$NAME_ARRAY;
    name      VARCHAR2(30);
    cval      VARCHAR2(100);
    rnum      INTEGER;
    i         INTEGER;
    status    PLS_INTEGER;
BEGIN
  vv  := SYS.RE$VARIABLE_VALUE('priority',
                               SYS.AnyData.CONVERTNUMBER(priority));
  vvl := SYS.RE$VARIABLE_VALUE_LIST(vv);
  truehits := SYS.RE$RULE_HIT_LIST();
  maybehits := SYS.RE$RULE_HIT_LIST();
  DBMS_RULE.EVALUATE(
      rule_set_name        => 'support.rs',
      evaluation_context   => 'evalctx',
      variable_values      => vvl,
      true_rules           => truehits,
      maybe_rules          => maybehits);
  FOR rnum IN 1..truehits.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE('Using rule '|| truehits(rnum).rule_name);
    ac := truehits(rnum).rule_action_context;
    namearray := ac.GET_ALL_NAMES;
      FOR i IN 1..namearray.count loop
        name := namearray(i);
        status := ac.GET_VALUE(name).GETVARCHAR2(cval);
        IF (name = 'CENTER') then
          DBMS_OUTPUT.PUT_LINE('Assigning problem to ' || cval);
        ELSIF (name = 'ALERT') THEN
          DBMS_OUTPUT.PUT_LINE('Sending alert to: '|| cval);
        END IF;
      END LOOP;
  END LOOP;
END;
/

/*
```

手順 10 サンプル問題のディスパッチ

```
*/

EXECUTE problem_dispatch(1);
EXECUTE problem_dispatch(2);
EXECUTE problem_dispatch(3);
EXECUTE problem_dispatch(5);

/*
```

手順 11 スプール結果のチェック

このスクリプトの完了後に rules_stored_variables.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT ****************************/
```

表に格納されたデータに関するルールの使用

この例では、ルールを使用して、表に格納されたデータを評価する方法について説明します。この例は、24-2 ページの「明示的変数に格納された表以外のデータに関するルールの使用」で説明した例に似ています。どちらの例でも、アプリケーションでは顧客の問題が優先度に基づいてルーティングされます。ただし、この例では、問題が変数のかわりに表に格納されています。

アプリケーションでは、support スキーマ内で顧客の問題が挿入される problems 表を使用します。この例では、顧客の問題を処理するために次のルールが使用されます。

- 優先度が 3 以上のすべての問題をサンノゼ・センターに割り当てます。
- 優先度が 2 以下のすべての問題をニューヨーク・センターに割り当てます。
- 優先度が 1 の問題については、サポート担当副社長に警告を送信します。

評価コンテキストは problems 表で構成されています。この表のうち、ルーティングされる問題に対応する、関連する行が、表の値として DBMS_RULE.EVALUATE プロシージャに渡されます。

この操作の手順は次のとおりです。

1. 出力の表示と結果のスプーリング
2. support ユーザーの削除と再作成
3. support ユーザーへのルールに関して必要なシステム権限の付与
4. problems 表の作成
5. evalctx 評価コンテキストの作成
6. 問題の優先度に対応するルールの作成
7. rs ルール・セットの作成
8. ルール・セットへのルールの追加
9. データ・ディクショナリの問合せ
10. problem_dispatch PL/SQL プロシージャの作成
11. 問題のロギング
12. problems 表内の問題のリスト表示
13. problem_dispatch プロシージャの実行による問題のディスパッチ
14. problems 表内の問題のリスト表示
15. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 24-16 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************ BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL rules_table.out

/*
```

手順 2 support ユーザーの削除と再作成

```
*/

CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;

DROP USER support CASCADE;

GRANT CONNECT, RESOURCE TO support IDENTIFIED BY support;

/*
```

手順 3 support ユーザーへのルールに関して必要なシステム権限の付与

```
*/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee      => 'support',
    grant_option => FALSE);
END;
/

/*
```

手順 4 problems 表の作成

```
*/

CONNECT support/support

SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET SERVEROUTPUT ON;

CREATE TABLE problems(
  probid          NUMBER PRIMARY KEY,
  custid          NUMBER,
  priority        NUMBER,
  description     VARCHAR2(4000),
  center          VARCHAR2(100));

/*
```

手順 5 evalctx 評価コンテキストの作成

```
*/

DECLARE
  ta  SYS.RE$TABLE_ALIAS_LIST;
BEGIN
  ta := SYS.RE$TABLE_ALIAS_LIST(SYS.RE$TABLE_ALIAS('prob', 'problems'));
  DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
    evaluation_context_name    => 'evalctx',
    table_aliases              => ta,
    evaluation_context_comment => 'support problem definition');
END;
/

/*
```

手順 6 問題の優先度に対応するルールの作成

次のコードは、各ルールにアクション・コンテキストを 1 つ作成し、各アクション・コンテキストに名前 / 値ペアを 1 つ作成します。

```
*/

DECLARE
  ac  SYS.RE$NV_LIST;
BEGIN
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('San Jose'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r1',
    condition      => 'prob.priority > 2',
    action_context => ac,
    rule_comment   => 'Low priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('New York'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r2',
    condition      => 'prob.priority <= 2',
    action_context => ac,
    rule_comment   => 'High priority problems');
  ac := sys.RE$NV_LIST(NULL);
  ac.ADD_PAIR('ALERT', SYS.AnyData.CONVERTVARCHAR2('John Doe'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r3',
    condition      => 'prob.priority = 1',
    action_context => ac,
    rule_comment   => 'Urgent problems');
END;
/

/*
```

手順 7 rs ルール・セットの作成

```
*/

BEGIN
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name      => 'rs',
    evaluation_context => 'evalctx',
    rule_set_comment   => 'support rules');
END;
/

/*
```

手順 8 ルール・セットへのルールの追加

```
*/

BEGIN
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r1',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r2',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r3',
    rule_set_name => 'rs');
END;
/

/*
```

手順 9 データ・ディクショナリの問合せ

この時点で、前述の手順で作成した評価コンテキスト、ルールおよびルール・セットを表示できます。

```
*/

SELECT * FROM USER_EVALUATION_CONTEXTS;

SELECT * FROM USER_RULES;

SELECT * FROM USER_RULE_SETS;

/*
```

手順 10 problem_dispatch PL/SQL プロシージャの作成

```
*/

CREATE OR REPLACE PROCEDURE problem_dispatch
IS
    cursor c IS SELECT probid, rowid FROM problems WHERE center IS NULL;
    tv        SYS.RE$TABLE_VALUE;
    tvl       SYS.RE$TABLE_VALUE_LIST;
    truehits  SYS.RE$RULE_HIT_LIST;
    maybehits SYS.RE$RULE_HIT_LIST;
    ac        SYS.RE$NV_LIST;
    namearray SYS.RE$NAME_ARRAY;
    name      VARCHAR2(30);
    cval      VARCHAR2(100);
```

```
    rnum      INTEGER;
    i         INTEGER;
    status    PLS_INTEGER;
BEGIN
  FOR r IN c LOOP
    tv  := SYS.RE$TABLE_VALUE('prob', rowidtochar(r.rowid));
    tvl := SYS.RE$TABLE_VALUE_LIST(tv);
    truehits := SYS.RE$RULE_HIT_LIST();
    maybehits := SYS.RE$RULE_HIT_LIST();
    DBMS_RULE.EVALUATE(
      rule_set_name        => 'support.rs',
      evaluation_context   => 'evalctx',
      table_values         => tvl,
      true_rules           => truehits,
      maybe_rules          => maybehits);
    FOR rnum IN 1..truehits.COUNT LOOP
      DBMS_OUTPUT.PUT_LINE('Using rule '|| truehits(rnum).rule_name);
      ac := truehits(rnum).rule_action_context;
      namearray := ac.GET_ALL_NAMES;
      FOR i IN 1..namearray.COUNT LOOP
        name := namearray(i);
        status := ac.GET_VALUE(name).GETVARCHAR2(cval);
        IF (name = 'CENTER') THEN
          UPDATE PROBLEMS SET center = cval WHERE rowid = r.rowid;
          DBMS_OUTPUT.PUT_LINE('Assigning '|| r.probid || ' to ' || cval);
        ELSIF (name = 'ALERT') THEN
          DBMS_OUTPUT.PUT_LINE('Alert: '|| cval || ' Problem:' || r.probid);
        END IF;
       END LOOP;
    END LOOP;
  END LOOP;
END;
/

/*
```

手順 11 問題のロギング

```
*/

INSERT INTO problems(probid, custid, priority, description)
  VALUES(10101, 11, 1, 'no dial tone');

INSERT INTO problems(probid, custid, priority, description)
  VALUES(10102, 21, 2, 'noise on local calls');

INSERT INTO problems(probid, custid, priority, description)
  VALUES(10103, 31, 3, 'noise on long distance calls');

COMMIT;

/*
```

手順 12 problems 表内の問題のリスト表示

次の SELECT 文を実行すると、手順 11 でログに記録された問題が表示されます。新規に挿入された各行の center 列が NULL であることに注意してください。

```
*/

SELECT * FROM problems;

/*
```

手順 13 problem_dispatch プロシージャの実行による問題のディスパッチ

```
*/

EXECUTE problem_dispatch;

/*
```

手順 14 problems 表内の問題のリスト表示

手順 13 で問題が正常にディスパッチされた場合は、次の SELECT 文を実行すると、各問題のディスパッチ先となったセンターが center 列に表示されます。

```
*/

SELECT * FROM problems;

/*
```

手順 15 スプール結果のチェック

このスクリプトの完了後に rules_table.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT ******************************/
```

明示的変数と表データの両方に関するルールの使用

この例では、ルールを使用して、明示的変数と表に格納されたデータを評価する方法について説明します。アプリケーションでは、support スキーマ内で顧客の問題が挿入される problems 表を使用します。この例では、顧客の問題を処理するために次のルールが使用されます。

- 優先度が 3 以上のすべての問題をサンノゼ・センターに割り当てます。
- 優先度が 2 のすべての問題をニューヨーク・センターに割り当てます。
- 優先度が 1 のすべての問題を、午前 8 時から午後 8 時まではタンパ・センターに割り当てます。
- 優先度が 1 のすべての問題を、午後 8 時から午前 8 時まではバンガロア・センターに割り当てます。
- 優先度が 1 の問題については、サポート担当副社長に警告を送信します。

評価コンテキストは problems 表で構成されています。この表のうち、ルーティングされる問題に対応する、関連する行が、表の値として DBMS_RULE.EVALUATE プロシージャに渡されます。

この例の一部のルールは現在の時刻を参照します。この時刻は、明示的変数 current_time として表されます。現在の時刻は、評価コンテキスト内で追加のデータとして扱われます。現在の時刻は、次の理由で変数として表されます。

- 現在の時刻は頻繁に更新する必要があるため、表に格納するのは実際的ではありません。
- 現在の時刻には、それを必要とする各ルールに SYSDATE のコールを挿入するとアクセスできますが、この場合、同じ SQL 関数 SYSDATE が繰り返し起動されることになり、ルール評価が低速になる可能性があります。現在の時刻の値がルールごとに異なると、不正な動作が発生する可能性があります。

この操作の手順は次のとおりです。

1. 出力の表示と結果のスプーリング
2. support ユーザーの削除と再作成
3. support ユーザーへのルールに関して必要なシステム権限の付与
4. problems 表の作成
5. evalctx 評価コンテキストの作成
6. 問題の優先度に対応するルールの作成
7. rs ルール・セットの作成
8. ルール・セットへのルールの追加
9. データ・ディクショナリの問合せ
10. problem_dispatch PL/SQL プロシージャの作成
11. 問題のロギング
12. problems 表内の問題のリスト表示
13. problem_dispatch プロシージャの実行による問題のディスパッチ
14. problems 表内の問題のリスト表示
15. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 24-24 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************ BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL rules_var_tab.out

/*
```

手順 2 support ユーザーの削除と再作成

```
*/

CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;

DROP USER support CASCADE;

GRANT CONNECT, RESOURCE TO support IDENTIFIED BY support;

/*
```

手順 3 support ユーザーへのルールに関して必要なシステム権限の付与

```
*/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee      => 'support',
    grant_option => FALSE);
END;
/

/*
```

手順 4 problems 表の作成

```
*/

CONNECT support/support

SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET SERVEROUTPUT ON;

CREATE TABLE problems(
  probid          NUMBER PRIMARY KEY,
  custid          NUMBER,
  priority        NUMBER,
  description     VARCHAR2(4000),
  center          VARCHAR2(100));

/*
```

手順 5 evalctx 評価コンテキストの作成

```
*/

DECLARE
  ta SYS.RE$TABLE_ALIAS_LIST;
  vt SYS.RE$VARIABLE_TYPE_LIST;
BEGIN
  ta := SYS.RE$TABLE_ALIAS_LIST(SYS.RE$TABLE_ALIAS('prob', 'problems'));
  vt := SYS.RE$VARIABLE_TYPE_LIST(
          SYS.RE$VARIABLE_TYPE('current_time', 'DATE', NULL, NULL));
  DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
    evaluation_context_name    => 'evalctx',
    table_aliases              => ta,
    variable_types             => vt,
    evaluation_context_comment => 'support problem definition');
END;
/

/*
```

手順 6 問題の優先度に対応するルールの作成

次のコードは、各ルールにアクション・コンテキストを 1 つ作成し、各アクション・コンテキストに名前 / 値ペアを 1 つ作成します。

```
*/

DECLARE
  ac SYS.RE$NV_LIST;
BEGIN
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('San Jose'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r1',
    condition      => 'prob.priority > 2',
    action_context => ac,
    rule_comment   => 'Low priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('New York'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r2',
    condition      => 'prob.priority = 2',
    action_context => ac,
    rule_comment   => 'High priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('ALERT', SYS.AnyData.CONVERTVARCHAR2('John Doe'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r3',
    condition      => 'prob.priority = 1',
    action_context => ac,
    rule_comment   => 'Urgent problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('Tampa'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r4',
    condition => '(prob.priority = 1) and ' ||
                 '(TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) >= 8) and ' ||
                 '(TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) <= 20)',
    action_context => ac,
    rule_comment => 'Urgent daytime problems');
  ac := sys.RE$NV_LIST(NULL);
  ac.add_pair('CENTER', SYS.Anydata.CONVERTVARCHAR2('Bangalore'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r5',
    condition => '(prob.priority = 1) and ' ||
                 '((TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) < 8) or ' ||
                 ' (TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) > 20))',
```

```
    action_context => ac,
    rule_comment => 'Urgent nighttime problems');
END;
/

/*
```

手順 7 rs ルール・セットの作成

```
*/

BEGIN
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name      => 'rs',
    evaluation_context => 'evalctx',
    rule_set_comment   => 'support rules');
END;
/

/*
```

手順 8 ルール・セットへのルールの追加

```
*/

BEGIN
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r1',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r2',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r3',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r4',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r5',
    rule_set_name => 'rs');
END;
/

/*
```

手順 9 データ・ディクショナリの問合せ

この時点で、前述の手順で作成した評価コンテキスト、ルールおよびルール・セットを表示できます。

```
*/

SELECT * FROM USER_EVALUATION_CONTEXTS;

SELECT * FROM USER_RULES;

SELECT * FROM USER_RULE_SETS;

/*
```

手順 10 problem_dispatch PL/SQL プロシージャの作成

```
*/

CREATE OR REPLACE PROCEDURE problem_dispatch
IS
    cursor c  is SELECT probid, rowid FROM PROBLEMS WHERE center IS NULL;
    tv        SYS.RE$TABLE_VALUE;
    tvl       SYS.RE$TABLE_VALUE_LIST;
    vv1       SYS.RE$VARIABLE_VALUE;
    vvl       SYS.RE$VARIABLE_VALUE_LIST;
    truehits  SYS.RE$RULE_HIT_LIST;
    maybehits SYS.RE$RULE_HIT_LIST;
    ac        SYS.RE$NV_LIST;
    namearray SYS.RE$NAME_ARRAY;
    name      VARCHAR2(30);
    cval      VARCHAR2(100);
    rnum      INTEGER;
    i         INTEGER;
    status    PLS_INTEGER;
BEGIN
  FOR r IN c LOOP
    tv  := sYS.RE$TABLE_VALUE('prob', ROWIDTOCHAR(r.rowid));
    tvl := SYS.RE$TABLE_VALUE_LIST(tv);
    vv1 := SYS.RE$VARIABLE_VALUE('current_time',
                                 SYS.AnyData.CONVERTDATE(SYSDATE));
    vvl := SYS.RE$VARIABLE_VALUE_LIST(vv1);
    truehits := SYS.RE$RULE_HIT_LIST();
    maybehits := SYS.RE$RULE_HIT_LIST();
    DBMS_RULE.EVALUATE(
        rule_set_name        => 'support.rs',
        evaluation_context   => 'evalctx',
        table_values         => tvl,
        variable_values      => vvl,
```

```
        true_rules           => truehits,
        maybe_rules          => maybehits);
    FOR rnum IN 1..truehits.COUNT loop
      DBMS_OUTPUT.PUT_LINE('Using rule '|| truehits(rnum).rule_name);
      ac := truehits(rnum).rule_action_context;
      namearray := ac.GET_ALL_NAMES;
      FOR i in 1..namearray.COUNT LOOP
        name := namearray(i);
        status := ac.GET_VALUE(name).GETVARCHAR2(cval);
        IF (name = 'CENTER') THEN
          UPDATE problems SET center = cval
          WHERE rowid = r.rowid;
          DBMS_OUTPUT.PUT_LINE('Assigning '|| r.probid || ' to ' || cval);
        ELSIF (name = 'ALERT') THEN
          DBMS_OUTPUT.PUT_LINE('Alert: '|| cval || ' Problem:' || r.probid);
        END IF;
      END LOOP;
    END LOOP;
  END LOOP;
END;
/

/*
```

手順 11 問題のロギング

```
*/

INSERT INTO problems(probid, custid, priority, description)
  VALUES(10201, 12, 1, 'no dial tone');

INSERT INTO problems(probid, custid, priority, description)
  VALUES(10202, 22, 2, 'noise on local calls');

INSERT INTO PROBLEMS(probid, custid, priority, description)
  VALUES(10203, 32, 3, 'noise on long distance calls');

COMMIT;

/*
```

手順 12 problems 表内の問題のリスト表示

次の SELECT 文を実行すると、手順 11 でログに記録された問題が表示されます。新規に挿入された各行の center 列が NULL であることに注意してください。

```
*/

SELECT * FROM problems;

/*
```

手順 13 problem_dispatch プロシージャの実行による問題のディスパッチ

```
*/

EXECUTE problem_dispatch;

/*
```

手順 14 problems 表内の問題のリスト表示

手順 13 で問題が正常にディスパッチされた場合は、次の SELECT 文を実行すると、各問題のディスパッチ先となったセンターが center 列に表示されます。

```
*/

SELECT * FROM problems;

/*
```

手順 15 スプール結果のチェック

このスクリプトの完了後に rules_var_tab.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT ******************************/
```

暗黙的変数と表データに関するルールの使用

この例では、ルールを使用して、暗黙的変数と表に格納されたデータを評価する方法について説明します。アプリケーションでは、support スキーマ内で顧客の問題が挿入される problems 表を使用します。この例では、顧客の問題を処理するために次のルールが使用されます。

- 優先度が 3 以上のすべての問題をサンノゼ・センターに割り当てます。
- 優先度が 2 のすべての問題をニューヨーク・センターに割り当てます。
- 優先度が 1 のすべての問題を、午前 8 時から午後 8 時まではタンパ・センターに割り当てます。
- 優先度が 1 のすべての問題を、午後 8 時から午前 8 時まではバンガロア・センターに割り当てます。
- 優先度が 1 の問題については、サポート担当副社長に警告を送信します。

評価コンテキストは problems 表で構成されています。この表のうち、ルーティングされる問題に対応する、関連する行が、表の値として DBMS_RULE.EVALUATE プロシージャに渡されます。

24-16 ページの「明示的変数と表データの両方に関するルールの使用」の例と同様に、現在の時刻は変数 current_time として表されます。ただし、コール元は、この変数の値を評価中に指定しません。つまり、この例では current_time は暗黙的変数です。current_time には PL/SQL ファンクション timefunc が指定されており、このファンクションは評価中に 1 回起動されて値を取得します。

その他、暗黙的変数を使用すると、次のいずれかの条件が当てはまる場合にも便利です。

- コール元に変数値へのアクセス権がない場合。
- 変数がルール内で参照される頻度が低い場合。この変数は暗黙的であるため、その値は必要な場合にのみ取得でき、各評価で渡す必要はありません。

この操作の手順は次のとおりです。

1. 出力の表示と結果のスプーリング
2. support ユーザーの削除と再作成
3. support ユーザーへのルールに関して必要なシステム権限の付与
4. problems 表の作成
5. current_time の値を戻す timefunc ファンクションの作成
6. evalctx 評価コンテキストの作成
7. 問題の優先度に対応するルールの作成
8. rs ルール・セットの作成

9. ルール・セットへのルールの追加
10. データ・ディクショナリの問合せ
11. problem_dispatch PL/SQL プロシージャの作成
12. 問題のロギング
13. problems 表内の問題のリスト表示
14. problem_dispatch プロシージャの実行による問題のディスパッチ
15. problems 表内の問題のリスト表示
16. スプール結果のチェック

注意： このマニュアルをオンラインで表示している場合は、次の BEGINNING OF SCRIPT 行から 24-33 ページの END OF SCRIPT 行までのテキストをテキスト・エディタにコピーし、テキストを編集して、環境に即したスクリプトを作成できます。このスクリプトは、環境内のすべてのデータベースに接続できるコンピュータ上で、SQL*Plus を使用して実行してください。

```
/************************ BEGINNING OF SCRIPT ******************************
```

手順 1 出力の表示と結果のスプーリング

SET ECHO ON を実行し、スクリプト用のスプール・ファイルを指定します。このスクリプトを実行した後に、スプール・ファイルでエラーの有無をチェックしてください。

```
*/

SET ECHO ON
SPOOL rules_implicit_var.out

/*
```

手順 2 support ユーザーの削除と再作成

```
*/

CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;

DROP USER support CASCADE;

GRANT CONNECT, RESOURCE TO support IDENTIFIED BY support;

/*
```

手順 3 support ユーザーへのルールに関して必要なシステム権限の付与

```
*/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee      => 'support',
    grant_option => FALSE);
END;
/

/*
```

手順 4 problems 表の作成

```
*/

CONNECT support/support

SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET SERVEROUTPUT ON;

CREATE TABLE problems(
  probid          NUMBER PRIMARY KEY,
  custid          NUMBER,
  priority        NUMBER,
  description     VARCHAR2(4000),
  center          VARCHAR2(100));

/*
```

手順 5 current_time の値を戻す timefunc ファンクションの作成

```
*/

CREATE OR REPLACE FUNCTION timefunc(
  eco    VARCHAR2,
  ecn    VARCHAR2,
  var    VARCHAR2,
  evctx  SYS.RE$NV_LIST)
RETURN SYS.RE$VARIABLE_VALUE
IS
BEGIN
  IF (var = 'CURRENT_TIME') THEN
    RETURN(SYS.RE$VARIABLE_VALUE('CURRENT_TIME',
                                 SYS.AnyData.CONVERTDATE(sysdate)));
  ELSE
    RETURN(NULL);
  END IF;
END;
/

/*
```

手順 6 evalctx 評価コンテキストの作成

```
*/

DECLARE
  ta SYS.RE$TABLE_ALIAS_LIST;
  vt SYS.RE$VARIABLE_TYPE_LIST;
BEGIN
  ta := SYS.RE$TABLE_ALIAS_LIST(SYS.RE$TABLE_ALIAS('prob', 'problems'));
  vt := SYS.RE$VARIABLE_TYPE_LIST(
          SYS.RE$VARIABLE_TYPE('current_time', 'DATE', 'timefunc', NULL));
  DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
    evaluation_context_name    => 'evalctx',
    table_aliases              => ta,
    variable_types             => vt,
    evaluation_context_comment => 'support problem definition');
END;
/

/*
```

手順 7 問題の優先度に対応するルールの作成

次のコードは、各ルールにアクション・コンテキストを 1 つ作成し、各アクション・コンテキストに名前 / 値ペアを 1 つ作成します。

```
*/

DECLARE
  ac SYS.RE$NV_LIST;
BEGIN
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('San Jose'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r1',
    condition      => 'prob.priority > 2',
    action_context => ac,
    rule_comment   => 'Low priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('New York'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r2',
    condition      => 'prob.priority = 2',
    action_context => ac,
    rule_comment   => 'High priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('ALERT', SYS.AnyData.CONVERTVARCHAR2('John Doe'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r3',
    condition      => 'prob.priority = 1',
    action_context => ac,
    rule_comment   => 'Urgent problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('Tampa'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r4',
    condition => '(prob.priority = 1) and ' ||
                 '(TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) >= 8) and ' ||
                 '(TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) <= 20)',
    action_context => ac,
    rule_comment   => 'Urgent daytime problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.add_pair('CENTER', sys.anydata.convertvarchar2('Bangalore'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r5',
    condition => '(prob.priority = 1) and ' ||
                 '((TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) < 8) or ' ||
                 ' (TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) > 20))',
```

```
    action_context => ac,
    rule_comment => 'Urgent nighttime problems');
END;
/

/*
```

手順 8 rs ルール・セットの作成

```
*/

BEGIN
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name      => 'rs',
    evaluation_context => 'evalctx',
    rule_set_comment   => 'support rules');
END;
/

/*
```

手順 9 ルール・セットへのルールの追加

```
*/

BEGIN
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r1',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r2',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r3',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r4',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r5',
    rule_set_name => 'rs');
END;
/

/*
```

手順 10 データ・ディクショナリの問合せ

この時点で、前述の手順で作成した評価コンテキスト、ルールおよびルール・セットを表示できます。

```
*/

SELECT * FROM USER_EVALUATION_CONTEXTS;

SELECT * FROM USER_RULES;

SELECT * FROM USER_RULE_SETS;

/*
```

手順 11 problem_dispatch PL/SQL プロシージャの作成

```
*/

CREATE OR REPLACE PROCEDURE problem_dispatch
IS
    cursor c  IS SELECT probid, rowid FROM problems WHERE center IS NULL;
    tv        SYS.RE$TABLE_VALUE;
    tvl       SYS.RE$TABLE_VALUE_LIST;
    truehits  SYS.RE$RULE_HIT_LIST;
    maybehits SYS.RE$RULE_HIT_LIST;
    ac        SYS.RE$NV_LIST;
    namearray SYS.RE$NAME_ARRAY;
    name      VARCHAR2(30);
    cval      VARCHAR2(100);
    rnum      INTEGER;
    i         INTEGER;
    status    PLS_INTEGER;
BEGIN
  FOR r IN c LOOP
    tv  := SYS.RE$TABLE_VALUE('prob', rowidtochar(r.rowid));
    tvl := SYS.RE$TABLE_VALUE_LIST(tv);
    truehits := SYS.RE$RULE_HIT_LIST();
    maybehits := SYS.RE$RULE_HIT_LIST();
    DBMS_RULE.EVALUATE(
        rule_set_name        => 'support.rs',
        evaluation_context   => 'evalctx',
        table_values         => tvl,
        true_rules           => truehits,
        maybe_rules          => maybehits);
    FOR rnum IN 1..truehits.COUNT LOOP
      DBMS_OUTPUT.PUT_LINE('Using rule '|| truehits(rnum).rule_name);
      ac := truehits(rnum).rule_action_context;
      namearray := ac.GET_ALL_NAMES;
```

```
      FOR i IN 1..namearray.COUNT LOOP
        name := namearray(i);
        status := ac.GET_VALUE(name).GETVARCHAR2(cval);
        IF (name = 'CENTER') THEN
          UPDATE problems SET center = cval
            WHERE rowid = r.rowid;
          DBMS_OUTPUT.PUT_LINE('Assigning '|| r.probid || ' to ' || cval);
        ELSIF (name = 'ALERT') THEN
          DBMS_OUTPUT.PUT_LINE('Alert: '|| cval || ' Problem:' || r.probid);
        END IF;
      END LOOP;
    END LOOP;
  END LOOP;
END;
/

/*
```

手順 12 問題のロギング

```
*/

INSERT INTO problems(probid, custid, priority, description)
  VALUES(10301, 13, 1, 'no dial tone');

INSERT INTO problems(probid, custid, priority, description)
  VALUES(10302, 23, 2, 'noise on local calls');

INSERT INTO problems(probid, custid, priority, description)
  VALUES(10303, 33, 3, 'noise on long distance calls');

COMMIT;

/*
```

手順 13 problems 表内の問題のリスト表示

次の SELECT 文を実行すると、手順 12 でログに記録された問題が表示されます。新規に挿入された各行の center 列が NULL であることに注意してください。

```
*/

SELECT * FROM problems;

/*
```

手順 14 problem_dispatch プロシージャの実行による問題のディスパッチ

```
*/

EXECUTE problem_dispatch;

/*
```

手順 15 problems 表内の問題のリスト表示

手順 13 で問題が正常にディスパッチされた場合は、次の SELECT 文を実行すると、各問題のディスパッチ先となったセンターが center 列に表示されます。

```
*/

SELECT * FROM problems;

/*
```

手順 16 スプール結果のチェック

このスクリプトの完了後に rules_implicit_var.out スプール・ファイルをチェックして、すべてのアクションが正常終了したかどうかを確認します。

```
*/

SET ECHO OFF
SPOOL OFF

/************************** END OF SCRIPT ******************************/
```

第 IV 部

付録

第 IV 部の構成は、次のとおりです。

- 付録 A「LCR 用の XML Schema」

A

LCR 用の XML Schema

この付録で説明する XML Schema では、論理変更レコード（LCR）のフォーマットが定義されます。

この付録の内容は次のとおりです。

このスキーマ用の名前空間は、次の場合に記載されています。

```
http://xmlns.oracle.com/streams/schemas/lcr
```

スキーマは、次の場合に記載されています。

```
http://xmlns.oracle.com/streams/schemas/lcr/streamslcr.xsd
```

このスキーマ定義は、SQL*Plus で SYS として接続して次のファイルを実行すると、データベースにロードできます。

```
rdbms/admin/catxlcr.sql
```

rdbms ディレクトリは Oracle ホームにあります。

LCR 用の XML Schema 定義

LCR 用の XML Schema 定義は、次のとおりです。

```
'<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://xmlns.oracle.com/streams/schemas/lcr"
        xmlns:lcr="http://xmlns.oracle.com/streams/schemas/lcr"
        xmlns:xdb="http://xmlns.oracle.com/xdb"
          version="1.0"
        elementFormDefault="qualified">

  <simpleType name = "short_name">
    <restriction base = "string">
      <maxLength value="30"/>
    </restriction>
  </simpleType>

  <simpleType name = "long_name">
    <restriction base = "string">
      <maxLength value="4000"/>
    </restriction>
  </simpleType>

  <simpleType name = "db_name">
    <restriction base = "string">
      <maxLength value="128"/>
    </restriction>
  </simpleType>

  <!-- Default session parameter is used if format is not specified -->
  <complexType name="datetime_format">
    <sequence>
      <element name = "value" type = "string" nillable="true"/>
      <element name = "format" type = "string" minOccurs="0" nillable="true"/>
    </sequence>
  </complexType>

  <complexType name="anydata">
    <choice>
      <element name="varchar2" type = "string" xdb:SQLType="VARCHAR2"
                                                          nillable="true"/>

      <!-- Represent char as varchar2. xdb:CHAR blank pads upto 2000 bytes! -->
      <element name="char" type = "string" xdb:SQLType="VARCHAR2"
                                                          nillable="true"/>
      <element name="nchar" type = "string" xdb:SQLType="NVARCHAR2"
                                                          nillable="true"/>
```

```
    <element name="nvarchar2" type = "string" xdb:SQLType="NVARCHAR2"
                                                        nillable="true"/>
    <element name="number" type = "double" xdb:SQLType="NUMBER"
                                                        nillable="true"/>
    <element name="raw" type = "hexBinary" xdb:SQLType="RAW"
                                                        nillable="true"/>
    <element name="date" type = "lcr:datetime_format"/>
    <element name="timestamp" type = "lcr:datetime_format"/>
    <element name="timestamp_tz" type = "lcr:datetime_format"/>
    <element name="timestamp_ltz" type = "lcr:datetime_format"/>

    <!-- Interval YM should be as per format allowed by SQL -->
    <element name="interval_ym" type = "string" nillable="true"/>

    <!-- Interval DS should be as per format allowed by SQL -->
    <element name="interval_ds" type = "string" nillable="true"/>
  </choice>
</complexType>

<complexType name="column_value">
  <sequence>
    <element name = "column_name" type = "lcr:long_name" nillable="false"/>
    <element name = "data" type = "lcr:anydata" nillable="false"/>
    <element name = "lob_information" type = "string" minOccurs="0"
                                                          nillable="true"/>
    <element name = "lob_offset" type = "nonNegativeInteger" minOccurs="0"
                                                          nillable="true"/>
    <element name = "lob_operation_size" type = "nonNegativeInteger"
                                          minOccurs="0" nillable="true"/>
  </sequence>
</complexType>

<element name = "ROW_LCR">
  <complexType>
    <sequence>
      <element name = "source_database_name" type = "lcr:db_name"
                                                           nillable="false"/>
      <element name = "command_type" type = "string" nillable="false"/>
      <element name = "object_owner" type = "lcr:short_name"
                                                           nillable="false"/>
      <element name = "object_name" type = "lcr:short_name"
                                                           nillable="false"/>
      <element name = "tag" type = "hexBinary" xdb:SQLType="RAW"
                                          minOccurs="0" nillable="true"/>
      <element name = "transaction_id" type = "string" minOccurs="0"
                                                            nillable="true"/>
```

```
      <element name = "scn" type = "double" xdb:SQLType="NUMBER"
                                              minOccurs="0" nillable="true"/>
      <element name = "old_values" minOccurs = "0">
        <complexType>
          <sequence>
            <element name = "old_value" type="lcr:column_value"
                                                  maxOccurs = "unbounded"/>
          </sequence>
        </complexType>
      </element>
      <element name = "new_values" minOccurs = "0">
        <complexType>
          <sequence>
            <element name = "new_value" type="lcr:column_value"
                                                  maxOccurs = "unbounded"/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>

<element name = "DDL_LCR">
  <complexType>
    <sequence>
      <element name = "source_database_name" type = "lcr:db_name"
                                                    nillable="false"/>
      <element name = "command_type" type = "string" nillable="false"/>
      <element name = "current_schema" type = "lcr:short_name"
                                                    nillable="false"/>
      <element name = "ddl_text" type = "string" nillable="false"/>
      <element name = "object_type" type = "string"
                                      minOccurs = "0" nillable="true"/>
      <element name = "object_owner" type = "lcr:short_name"
                                      minOccurs = "0" nillable="true"/>
      <element name = "object_name" type = "lcr:short_name"
                                      minOccurs = "0" nillable="true"/>
      <element name = "logon_user" type = "lcr:short_name"
                                      minOccurs = "0" nillable="true"/>
      <element name = "base_table_owner" type = "lcr:short_name"
                                      minOccurs = "0" nillable="true"/>
      <element name = "base_table_name" type = "lcr:short_name"
                                      minOccurs = "0" nillable="true"/>
      <element name = "tag" type = "hexBinary" xdb:SQLType="RAW"
                                      minOccurs = "0" nillable="true"/>
      <element name = "transaction_id" type = "string"
                                      minOccurs = "0" nillable="true"/>
```

```
        <element name = "scn" type = "double" xdb:SQLType="NUMBER"
                                              minOccurs = "0" nillable="true"/>
      </sequence>
    </complexType>
  </element>
</schema>';
```

索引

A

C

D

E

G

I

J

L

M

N

O

P

R

S

U

V

X

あ

い

え

か

き

け

こ

さ

し

す

そ

た

て

と

は

ひ

へ

ほ

め

も

ゆ

る

れ

ろ