Oracle® Database

管理者ガイド

11*g* リリース 1（11.1）

部品番号 : E05760-03

2008 年 10 月

ORACLE®

Oracle Database 管理者ガイド , 11*g* リリース 1（11.1）

部品番号 : E05760-03

Oracle Database Administrator's Guide, 11*g* Release 1 (11.1)

原本部品番号 : B28310-04

原本著者 : Steve Fogel

原本協力者: Tony Morales, Padmaja Potineni, Sheila Moore, David Austin, Bharat Baddepudi, Prasad Bagal, Cathy Baird, Mark Bauer, Eric Belden, Allen Brumm, Atif Chaudhry, Sudip Datta, Mark Dilman, Jacco Draaijer, Harvey Eneman, Marcus Fallen, Amit Ganesh, GP Gongloor, Vira Goorah, Carolyn Gray, Joan Gregoire, Shivani Gupta, Daniela Hansell, Lilian Hobbs, Bill Hodak, Wei Huang, Pat Huey, Robert Jenkins, Bhushan Khaladkar, Balaji Krishnan, Srinath Krishnaswamy, Vasudha Krishnaswamy, Bala Kuchibhotla, Sushil Kumar, Vikram Kumar, Paul Lane, Adam Lee, Bill Lee, Sue K. Lee, Chon Lei, Yunrui Li, Ilya Listvinsky, Bryn Llewellyn, Catherine Luu, Scott Lynn, Raghu Mani, Vineet Marwah, Colin McGregor, Mughees Minhas, Krishna Mohan, Sheila Moore, Valarie Moore, Niloy Mukherjee, Sujatha Muthulingam, Gary Ngai, Waleed Ojeil, Rod Payne, Hanlin Qian, Ananth Raghavan, Mark Ramacher, Ravi Ramkissoon, Ann Rhee, Yair Sarig, Vikram Shukla, Bipul Sinha, Anupam Singh, Wayne Smith, Jags Srinivasan, Deborah Steiner, Janet Stern, Michael Stewart, Mahesh Subramaniam, Nick Taylor, Anh-Tuan Tran, Alex Tsukerman, Kothanda Umamageswaran, Guhan Viswanathan, Eric Voss, Daniel M. Wong, Wanli Yang, Paul Youn, Wei Zhang

制限付権利の説明

このプログラム（ソフトウェアおよびドキュメントを含む）には、オラクル社およびその関連会社に所有権のある情報が含まれています。このプログラムの使用または開示は、オラクル社およびその関連会社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権と工業所有権に関する法律により保護されています。

独立して作成された他のソフトウェアとの互換性を得るために必要な場合、もしくは法律によって規定される場合を除き、このプログラムのリバース・エンジニアリング、逆アセンブル、逆コンパイル等は禁止されています。

このドキュメントの情報は、予告なしに変更される場合があります。オラクル社およびその関連会社は、このドキュメントに誤りが無いことの保証は致し兼ねます。これらのプログラムのライセンス契約で許諾されている場合を除き、プログラムを形式、手段（電子的または機械的）、目的に関係なく、複製または転用することはできません。

このプログラムが米国政府機関、もしくは米国政府機関に代わってこのプログラムをライセンスまたは使用する者に提供される場合は、次の注意が適用されます。

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

このプログラムは、核、航空、大量輸送、医療あるいはその他の本質的に危険を伴うアプリケーションで使用されることを意図しておりません。このプログラムをかかる目的で使用する際、上述のアプリケーションを安全に使用するために、適切な安全装置、バックアップ、冗長性（redundancy）、その他の対策を講じることは使用者の責任となります。万一かかるプログラムの使用に起因して損害が発生いたしましても、オラクル社およびその関連会社は一切責任を負いかねます。

Oracle、JD Edwards、PeopleSoft、Siebel は米国 Oracle Corporation およびその子会社、関連会社の登録商標です。その他の名称は、他社の商標の可能性があります。

このプログラムは、第三者の Web サイトへリンクし、第三者のコンテンツ、製品、サービスへアクセスすることがあります。オラクル社およびその関連会社は第三者の Web サイトで提供されるコンテンツについては、一切の責任を負いかねます。当該コンテンツの利用は、お客様の責任になります。第三者の製品またはサービスを購入する場合は、第三者と直接の取引となります。オラクル社およびその関連会社は、第三者の製品およびサービスの品質、契約の履行（製品またはサービスの提供、保証義務を含む）に関しては責任を負いかねます。また、第三者との取引により損失や損害が発生いたしましても、オラクル社およびその関連会社は一切の責任を負いかねます。

目次

2 Oracle Database の作成および構成

3 起動と停止

4 プロセスの管理

5 メモリーの管理

6 ユーザーの管理とデータベースのセキュリティ保護

7 データベースの動作の監視

8 診断データの管理

10 REDO ログの管理

11 アーカイブ REDO ログの管理

12 表領域の管理

13 データファイルおよび一時ファイルの管理

14 UNDO の管理

15 Oracle Managed Files の使用

第 III 部 スキーマ・オブジェクト

16 スキーマ・オブジェクトの管理

17 スキーマ・オブジェクトの領域の管理

18 表の管理

19 索引の管理

20 クラスタの管理

21 ハッシュ・クラスタの管理

22 ビュー、順序およびシノニムの管理

23 破損データの修復

第 IV 部　データベース・リソースの管理とタスクのスケジューリング

24　自動データベース・メンテナンス・タスクの管理

25　Oracle Database Resource Manager を使用したリソース割当ての管理

26 Oracle Scheduler の概要

27 Oracle Scheduler を使用したジョブのスケジューリング

28 Oracle Scheduler の管理

30 分散データベースの管理

31 分散データベース・システムのアプリケーション開発

32 分散トランザクションの概念

33 分散トランザクションの管理

第 VI 部　付録

A　リリース 11*g* における DBMS_JOB のサポート

索引

はじめに

このマニュアルでは、Oracle データベースを作成、構成および管理する方法について説明します。

対象読者

このマニュアルは、次のタスクを実行するデータベース管理者を対象にしています。

- Oracle データベースの作成
- Oracle データベースの円滑な運用
- Oracle データベース使用状況の監視

このマニュアルを使用するにあたって、リレーショナル・データベースの概念を理解しておく必要があります。また、Oracle Database を実行するオペレーティング・システム環境についても知っておく必要があります。

ドキュメントのアクセシビリティについて

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

ドキュメント内のサンプル・コードのアクセシビリティについて

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

外部 Web サイトのドキュメントのアクセシビリティについて

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

Oracle サポート・サービスへの TTY アクセス

アメリカ国内では、Oracle サポート・サービスへ 24 時間年中無休でテキスト電話（TTY）アクセスが提供されています。TTY サポートについては、(800)446-2398 にお電話ください。アメリカ国外からの場合は、+1-407-458-2479 にお電話ください。

関連ドキュメント

詳細は、次の Oracle ドキュメントを参照してください。

- 『Oracle Database 2 日でデータベース管理者』
- 『Oracle Database 概要』
- 『Oracle Database SQL リファレンス』
- 『Oracle Database リファレンス』
- 『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』
- 『Oracle Database ストレージ管理者ガイド』
- 『Oracle Database VLDB およびパーティショニング・ガイド』
- 『Oracle Database エラー・メッセージ』
- 『Oracle Database Net Services 管理者ガイド』
- 『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』
- 『Oracle Database パフォーマンス・チューニング・ガイド』
- 『Oracle Database アドバンスト・アプリケーション開発者ガイド』
- 『Oracle Database PL/SQL 言語リファレンス』
- 『SQL*Plus ユーザーズ・ガイドおよびリファレンス』

このマニュアルの例の多くは、Oracle Database のインストール時に基本インストール・オプションを選択した場合にデフォルトでインストールされるサンプル・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たか、およびその使用方法の詳細は、『Oracle Database サンプル・スキーマ』を参照してください。

表記規則

このマニュアルでは、次の表記規則を使用します。

規則	意味
太字	太字は、操作に関連する Graphical User Interface 要素、または本文中で定義されている用語および用語集に記載されている用語を示します。
イタリック体	イタリックは、ユーザーが特定の値を指定するプレースホルダ変数を示します。
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、サンプル内のコード、画面に表示されるテキスト、または入力するテキストを示します。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

Oracle サポート・サービス

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

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

製品マニュアル

製品のマニュアルは、次の URL にあります。

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

研修およびトレーニング

研修に関する情報とスケジュールは、次の URL で入手できます。

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

その他の情報

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

```
http://www.oracle.com/lang/jp/index.html
http://www.oracle.com/technology/global/jp/index.html
```

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

このマニュアルに記載されている新機能

ここでは、このマニュアルに記載されている Oracle Database 11*g* リリース 1（11.1）の新機能について説明し、追加情報の参照先を示します。

管理者ガイドに記載されている Oracle Database 11*g* リリース 1（11.1）の新機能

- 自動メモリー管理の簡素化と改善

  単一の初期化パラメータ（MEMORY_TARGET）を設定して、データベース（SGA およびインスタンス PGA）に割り当てられるメモリーの合計量を指定できます。システムでは、最適のパフォーマンスが得られるように、すべての SGA および PGA コンポーネントを自動的かつ動的にチューニングします。SGA およびインスタンス PGA の最小サイズを個別に指定することもできます。

  5-4 ページの「自動メモリー管理の使用」を参照してください。

- クリティカル・データベース・エラーを防止、検出、診断および解決するための新しい障害診断インフラストラクチャ

  障害診断インフラストラクチャの目的は、問題（クリティカル・エラー）の事前の防止および検出、問題検出後の破損や割込みの抑制、問題診断時間の短縮、問題解決時間の短縮、および顧客と Oracle サポート・サービスとの間の対話の簡素化です。このフレームワークには、クリティカル・エラーの発生時に実行するヘルス・チェック、初期障害データを取得するための多数のデータベース・コンポーネントに対する事前のメモリー内トレース、問題に関するすべての診断データを ZIP ファイルにパッケージ化して Oracle サポート・サービスに送信するインシデント・パッケージング・サービス、問題を調査、レポートおよび解決するためのグラフィカルな環境を提供する Enterprise Manager サポート・ワークベンチなどのテクノロジが含まれています。また、SQL 関連の問題を診断および修復するための新しい SQL 修復アドバイザ、SQL 問題を別の Oracle データベース上で再現するのに必要なすべてのスキーマおよび環境情報を収集する SQL テスト・ケース・ビルダー、データ破損およびその他のデータ障害の診断、影響の評価および修復に役立つデータ・リカバリ・アドバイザと統合されています。

  8-1 ページの第 8 章「診断データの管理」を参照してください。

- 不可視索引

  索引を削除することで全体的なパフォーマンスが向上するかどうかをテストする必要がある場合は、索引を使用禁止にしたり削除するかわりに、索引を不可視にできます。不可視索引は、デフォルトではオプティマイザで無視されますが、使用禁止状態の索引とは異なり、DML 文が終了するまで保持されます。オプティマイザで不可視索引を使用できるように、初期化パラメータをシステム・レベルまたはセッション・レベルで変更できます。

  19-12 ページの「不可視索引の作成」を参照してください。

- 仮想列

  表に仮想列を追加できるようになりました。行にある仮想列の値は、式を評価して導出されます。式に使用できるのは、同じ表の列、定数、SQL ファンクションおよびユーザー定義の PL/SQL ファンクションです。仮想列を使用すると、別にビューを作成する必要がなくなる場合があります。仮想列に索引を作成でき、仮想列をパーティション・キーまたはサブパーティション・キーとして使用できます。

  18-2 ページの「表の概要」を参照してください。仮想列の詳細は、『Oracle Database 概要』を参照してください。

- パスワードでの大 / 小文字使用によるパスワード・ベース認証のセキュリティ強化

  1-16 ページの「DBA の認証」を参照してください。

- データベース常駐接続プーリング

  データベース常駐接続プーリング（DRCP）は、データベース接続を取得し、比較的短時間の処理を実行した後、データベース接続を解放するような一般的な Web アプリケーション使用に対して、データベース・サーバーの接続プールを提供します。DRCP は専用サーバーをプールします。これは、サーバー・フォアグラウンド・プロセスとデータベース・セッションの組合せに相当します。DRCP を使用すると、同じ中間層ホスト上の中間層プロセス間、および異なる中間層ホスト上の中間層プロセス間でデータベース接続を共有できます。この結果、大量のクライアント接続をサポートするために必要なデータベース・リソースが大幅に減少するため、中間層とデータベース層の両方のスケーラビリティが向上します。

  4-5 ページの「データベース常駐接続プーリングの概要」を参照してください。

- 表領域レベルの暗号化

  永続表領域を暗号化して機密データを保護できます。表領域の暗号化は、アプリケーションに対して完全に透過的です。表領域を暗号化すると、すべての表領域ブロックが暗号化されます。暗号化は、表、クラスタ、索引、LOB、表パーティション、索引パーティションなどを含むすべてのセグメント・タイプに対してサポートされています。

  12-8 ページの「暗号化された表領域」を参照してください。

- ファイングレイン・スキーマ・オブジェクト依存性による可用性の向上

  依存するオブジェクトの変更に伴う依存スキーマ・オブジェクトの無効化が Oracle Database 11*g* では大幅に減少したため、メンテナンス、アップグレードおよび表のオンライン再定義時のアプリケーション可用性が向上しました。参照オブジェクトとその依存オブジェクトの間で、データベースは、依存関係に含まれる参照オブジェクトの要素を追跡します。たとえば、単一表のビューで表内の列のサブセットのみが選択された場合、それらの列のみが依存関係に含まれます。オブジェクトの各依存関係について、依存関係に含まれる要素の定義が変更されると（要素の削除も含む）、依存オブジェクトは無効になります。逆に、依存関係に含まれない要素の定義のみが変更された場合、依存オブジェクトは有効なままです。

- 自動化メンテナンス・タスク・インフラストラクチャの拡張

  自動化メンテナンス・タスクのスケジューリングをきめ細かく制御できるようになりました。Oracle Database の新規インストールでは、自動化メンテナンス・タスクとそれを実行するメンテナンス・ウィンドウについて、次のデフォルト構成が用意されています。

  – オプティマイザ統計の収集、自動セグメント・アドバイザおよび自動 SQL チューニング・アドバイザの 3 つの自動化メンテナンス・タスクがあります。

    自動 SQL チューニング・アドバイザは、負荷が高い SQL 文を検証して、問合せパフォーマンスを向上させるための推奨事項を示します。このアドバイザは、SQL プロファイルの推奨事項を自動的に実装するように構成できます。

  – 曜日ごとに個別のメンテナンス・ウィンドウがあります。デフォルトでは、すべてのメンテナンス・タスクがすべてのメンテナンス・ウィンドウで実行されるようにスケジュールされます。

  – デフォルトのリソース・マネージャ・プランが有効化されています。このプランには、自動化メンテナンス・タスクが使用するリソース量を制限するサブプランが含まれます。

  Enterprise Manager または PL/SQL パッケージ・プロシージャを使用すると、すべてのメンテナンス・ウィンドウの起動時間と継続時間を変更したり、メンテナンス・ウィンドウを削除または追加できます。さらに、特定のメンテナンス・タスクを特定のメンテナンス・ウィンドウで実行しないようにすることもできます。また、メンテナンス・タスクに対するリソース割当てを、他のメンテナンス・タスクやアプリケーションと関連させて調整できます。

  第 24 章「自動データベース・メンテナンス・タスクの管理」を参照してください。

- Oracle Database の透過的データ暗号化機能を使用する表列の暗号化で、SECUREFILE LOB がサポートされるようになりました。

  詳細は、『Oracle Database SecureFiles およびラージ・オブジェクト開発者ガイド』を参照してください。

- OLTP 環境での表圧縮のサポート

  圧縮表で次の操作がサポートされるようになりました。

  - DML 文
  - 列の追加および削除

  18-5 ページの「表圧縮の使用」を参照してください。

- システム・グローバル領域での結果キャッシュ

  問合せおよび問合せ断片の結果が、メモリー内の結果キャッシュにキャッシュできるようになりました。データベースでは、後で問合せおよび問合せ断片を実行する際に、キャッシュされた結果を使用できます。問合せを再実行するよりも結果キャッシュから結果を取得する方が処理が速いため、頻繁に実行する問合せでは、結果をキャッシュするとパフォーマンスが大幅に向上します。

  結果キャッシュでは、共有プールのメモリーを使用します。

  5-18 ページの「結果キャッシュの最大サイズの指定」を参照してください。

- Oracle Scheduler の拡張機能

  Oracle Scheduler の拡張機能は次のとおりです。

  - リモート外部ジョブ : リモート外部ジョブは、データベースの外部で実行するオペレーティング・システム実行可能ファイルで、Oracle Scheduler によってスケジュールされます。また、リモート外部ジョブがスケジュールされている Oracle データベースを実行しているコンピュータとは別のホスト・コンピュータで実行されます。リモート・ホストでは Oracle データベースは必要ありません。かわりに、別個にインストールされたスケジューラ・エージェントがあり、スケジューリング・データベースと通信して外部ジョブを起動します。また、このエージェントは、実行結果をスケジューリング・データベースに戻す処理にも関係しています。

    26-8 ページの「外部ジョブ」を参照してください。

  - デタッチ済ジョブ : デタッチ済ジョブは、データベースを再起動する間も引き続き実行できます。デタッチ済ジョブを使用することで、Oracle Scheduler を使用してコールド・バックアップなどのタスクをスケジュールできるようになりました。

    26-10 ページの「デタッチ済ジョブ」を参照してください。

  - 軽量ジョブ : 軽量ジョブは、標準のスケジューラ・ジョブのようなスキーマ・オブジェクトではありません。軽量ジョブは、権限と（場合によっては）ジョブ・メタデータが継承されるジョブ・テンプレートに基づいています。スキーマ・オブジェクトを作成する際のオーバーヘッドを伴わないため、作成および削除に必要な時間が標準ジョブよりも大幅に短縮されます。1 秒間に数百、数千のジョブを作成したり削除する必要がある場合は、軽量ジョブを使用します。

    26-11 ページの「軽量ジョブ」を参照してください。

  - Oracle Data Guard 環境のサポートの拡張 : スケジューラ・ジョブは、データベースのロールがプライマリ・データベースの場合のみ、またはデータベースのロールがロジカル・スタンバイ・データベースの場合のみ実行するように指定できます。

    26-20 ページの「スケジューラによる Oracle Data Guard のサポート」を参照してください。

- Oracle Database Resource Manager の拡張機能

  Oracle Database Resource Manager の拡張機能は次のとおりです。

  - セッションごとの I/O 制限 : I/O リソースの使用制限を超えたセッションを、別のコンシューマ・グループに自動的に切り替えることができます。

    25-23 ページの「リソース・コンシューマ・グループの自動切替えの指定」を参照してください。

  - 開梱時の新しい複合ワークロード・リソース・プラン : Oracle Database 11*g* には、事前定義のリソース・プラン `MIXED_WORKLOAD_PLAN` が用意されています。このプランでは、バッチ操作よりも対話型操作が優先されます。必要なサブプランと推奨するコンシューマ・グループも含まれています。

    25-35 ページの「オラクル社が提供する複合ワークロード・プラン」を参照してください。

  - リソース・マネージャの動的パフォーマンス・ビューの新しい自動ワークロード・リポジトリ（AWR）スナップショット : リソース・プランのアクティブ化に関する履歴統計データ、コンシューマ・グループで使用された CPU リソース、およびコンシューマ・グループ別の CPU 待ち時間を示します。

    25-39 ページの「Oracle Database Resource Manager の監視」を参照してください。

- デフォルトの自動 UNDO 管理モード

  新規にインストールした 11*g* インスタンスは自動 UNDO 管理モードにデフォルト設定され、Database Configuration Assistant を使用してデータベースが作成された場合は、UNDO 表領域が自動的に作成されます。`UNDO_MANAGEMENT` 初期化パラメータが NULL 値の場合は、自動 UNDO 管理にデフォルト設定されます。

  14-2 ページの「自動 UNDO 管理の概要」を参照してください。

- オンラインでの索引の作成および再作成の拡張

  以前のリリースでは、オンラインで索引を作成および再作成する際、再作成の開始時と終了時に短時間の DML ブロッキング・ロックが必要でした。このロックによって他の DML 文が遅延するため、パフォーマンス・スパイクが発生しました。このロックは不要になり、オンラインでの索引操作は完全に透過的になりました。

  19-10 ページの「索引のオンラインでの作成」を参照してください。

- マテリアライズド・ビュー・ログが含まれる表のオンライン再定義機能

  マテリアライズド・ビュー・ログが含まれる表を、オンライン再定義できるようになりました。マテリアライズド・ビュー・ログは、`DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS` パッケージ・プロシージャを使用して仮表にコピーできる依存オブジェクトの 1 つになりました。

  18-26 ページの「表のオンライン再定義」を参照してください。

- 読取り専用表

  `ALTER TABLE` 文を使用して、任意の表を読取り専用モードに設定できます。これは、表を含む表領域を読取り専用モードにする代替手段として使用できます。

  18-25 ページの「表を読取り専用モードにする方法」を参照してください。

- トランスポータブル表領域の拡張機能

  データ・ポンプで、XMLType 表を含む表領域、および XMLTypes のスキーマ・オブジェクトを含む表領域に対して、トランスポータブル表領域機能がサポートされるようになりました。

  12-30 ページの「データベース間での表領域のトランスポート」を参照してください。

- ALTER TABLE...ADD COLUMN の最適化

  特定のタイプの表に対して、NOT NULL 制約とデフォルト値がある列を追加する場合、データベースでは操作に必要なリソース使用量と記憶域要件を最適化できます。新規列のデフォルト値を表メタデータとして格納することにより、既存のすべてのレコードに値を格納する必要がなくなります。

  さらに、次の ADD COLUMN 操作は、DML 操作と同時に実行できるようになりました。

  - デフォルト値がある NOT NULL 列の追加
  - デフォルト値がない NULL 値可能列の追加
  - 仮想列の追加

- 初期化パラメータ管理の拡張機能

  初期化パラメータの処理に次の拡張機能が追加されました。

  - サーバー・パラメータ・ファイル（SPFILE）が、Oracle の HARD イニシアティブに準拠した形式になりました。このイニシアティブは、破損したデータがディスクに書き込まれるのを防止するのに役立ち、ソフトウェアおよび記憶域ハードウェア・レベルで実装されます。
  - 新規のコマンドを使用して、メモリー内の初期化パラメータの現行値からテキスト形式の初期化パラメータ・ファイル（PFILE）またはサーバー・パラメータ・ファイル（SPFILE）を作成できます。
  - 起動時に、初期化パラメータの値がアラート・ログに書き込まれます。書き込まれた値を簡単にコピー・アンド・ペーストして新規の PFILE を作成できます。
  - インスタンスの起動に使用する PFILE または SPFILE の名前とパスがアラート・ログに書き込まれます。
  - Oracle Database では、必要に応じて、SPFILE のミラー化コピーを自動的にミラー復元します。

- データ定義言語（DDL）コマンドでのロックの待機

  単一の初期化パラメータ DDL_LOCK_TIMEOUT を設定して、DDL コマンドが内部構造で必要な排他ロックを待機する時間を指定できるようになりました。この時間を超えると DDL コマンドは失敗します。

  2-29 ページの「DDL ロック・タイムアウトの指定」を参照してください。

第 I 部

基本データベース管理

第 I 部では、データベース管理者が担当する作業の概要を示し、基本的なデータベース管理タスクを実行する方法を説明します。この部の構成は、次のとおりです。

1

データベース管理スタート・ガイド

この章の内容は次のとおりです。

Oracle Database ユーザーのタイプ

ユーザーのタイプとその役割および責任は、データベース・サイトによって異なります。小規模のサイトでは、1 名のデータベース管理者を配置して、アプリケーション開発者およびユーザー向けのデータベースを管理できます。大規模なサイトでは、データベース管理者の役割を複数の人および複数の専門グループに分割する必要があります。

データベース管理者

各データベースには、少なくとも 1 名の DBA が必要です。Oracle Database システムは規模が大きく、多数のユーザーによって使用される可能性があります。したがって、1 名の担当者ではデータベースを管理できないため、複数の DBA でグループを編成して役割を分担します。

DBA が担当するタスクは次のとおりです。

- Oracle Database サーバーとアプリケーション・ツールをインストールおよびアップグレードします。
- データベース・システムにシステム記憶域を割り当て、将来の記憶域要件を計画します。
- アプリケーション開発者がアプリケーションを設計した後、プライマリ・データベースの記憶域構造（表領域）を作成します。
- アプリケーション開発者がアプリケーションを設計した後、プライマリ・オブジェクト（表、ビュー、索引）を作成します。
- アプリケーション開発者から得た情報に基づき、必要に応じてデータベース構造を修正します。
- ユーザーを登録し、システム・セキュリティをメンテナンスします。
- Oracle のライセンス契約に従っていることを確認します。
- データベースに対するユーザー・アクセスを制御し、監視します。
- データベースのパフォーマンスを監視し、最適化します。
- データベース情報のバックアップおよびリカバリの計画を立てます。
- テープ上のアーカイブ済データをメンテナンスします。
- データベースをバックアップおよびリストアします。
- 技術サポートについて Oracle サポート・サービスに連絡します。

セキュリティ管理者

サイトによっては、データベースに 1 名以上のセキュリティ管理者が必要です。セキュリティ管理者は、ユーザーの登録、データベースに対するユーザー・アクセスの制御と監視およびシステム・セキュリティのメンテナンスを実施します。したがって、サイトにセキュリティ管理者が別にいる場合、DBA はこれらの業務に対する役割を持ちません。セキュリティ管理者の業務の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

ネットワーク管理者

サイトによっては、1 名以上のネットワーク管理者がいる場合があります。たとえば、ネットワーク管理者は、Oracle Net Services などの Oracle ネットワーク製品を管理します。ネットワーク管理者の業務の詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

> **関連項目：** 分散環境でのネットワーク管理の詳細は、第 V 部「分散データベースの管理」を参照してください。

アプリケーション開発者

アプリケーション開発者は、データベース・アプリケーションを設計し、実装します。アプリケーション開発者が担当するタスクは、次のとおりです。

- データベース・アプリケーションを設計、開発します。
- アプリケーションのためのデータベース構造を設計します。
- アプリケーションに対する記憶域要件を見積ります。
- アプリケーションのためのデータベース構造の変更を指定します。
- データベース管理者にこのような情報を伝えます。
- 開発中にアプリケーションをチューニングします。
- 開発中にアプリケーションに対するセキュリティ手段を確立します。

アプリケーション開発者は、これらのタスクの一部を DBA と協力して実施する場合があります。アプリケーション開発タスクの詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

アプリケーション管理者

Oracle Database のサイトでは、特定のアプリケーションを管理するために 1 名以上のアプリケーション管理者が必要な場合があります。アプリケーションごとに専門の管理者を置く場合があります。

データベース・ユーザー

データベース・ユーザーは、アプリケーションまたはユーティリティを介してデータベースと対話します。一般ユーザーが担当するタスクは、次のとおりです。

- 許された範囲でデータを入力、修正および削除します。
- データのレポートを作成します。

DBA のタスク

次のタスクは、Oracle Database を設計、実装およびメンテナンスするためのアプローチの優先度を表しています。

タスク 1: データベース・サーバー・ハードウェアの評価

タスク 2: Oracle Database ソフトウェアのインストール

タスク 3: データベースの計画

タスク 4: データベースの作成とオープン

タスク 5: データベースのバックアップ

タスク 6: システム・ユーザーの登録

タスク 7: データベース設計の実装

タスク 8: 実行データベースのバックアップ

タスク 9: データベースのパフォーマンス・チューニング

タスク 10: パッチのダウンロードとインストール

タスク 11: 追加ホストへのロール・アウト

次の項で、これらのタスクについて説明します。

注意： 新しいリリースにアップグレードする場合は、既存の本番環境（ソフトウェアとデータベースの両方）のバックアップを作成してからインストールしてください。既存の本番データベースの保存方法については、『Oracle Database アップグレード・ガイド』を参照してください。

タスク 1: データベース・サーバー・ハードウェアの評価

使用可能なコンピュータ・リソースを、Oracle Database とそのアプリケーションで最大限に活用するための評価を行います。この評価では、次のような情報を明らかにする必要があります。

- Oracle 製品に使用可能なディスク・ドライブ数
- Oracle 製品に使用可能な専用テープ・ドライブ数（テープ・ドライブがある場合）
- Oracle Database インスタンスで使用可能なメモリーの量（システムの構成マニュアルを参照）

タスク 2: Oracle Database ソフトウェアのインストール

データベース管理者は、Oracle Database サーバーのソフトウェアとすべてのフロントエンド・ツール、およびデータベースにアクセスするデータベース・アプリケーションをインストールします。分散処理環境インストールでは、データベースが中核となるコンピュータ（データベース）によって制御され、データベース・ツールとアプリケーションがリモート・マシン（クライアント）で実行される場合があります。このような場合は、Oracle Database を実行するコンピュータに、リモート・マシンを接続するために必要な Oracle Net コンポーネントもインストールする必要があります。

インストールするソフトウェアの詳細は、1-13 ページの「Oracle Database ソフトウェアのリリースの識別」を参照してください。

関連項目： インストールの特定の要件や指示の詳細は、次のマニュアルを参照してください。

- 使用しているオペレーティング・システム固有の Oracle マニュアル
- フロントエンド・ツールおよび Oracle Net ドライバのインストレーション・ガイド

タスク 3: データベースの計画

DBA は、次のことを計画する必要があります。

- データベースの論理記憶域構造
- データベース全体の設計
- データベースのバックアップ計画

重要なことは、データベースの論理記憶域構造が、システムのパフォーマンスと様々なデータベース管理操作にどのように影響を及ぼすかについて計画することです。たとえば、表領域を構成するデータファイルの数、各表領域に格納される情報のタイプ、およびデータファイルの物理的な格納先ディスク・ドライブについて、表領域を作成する前に把握しておく必要があります。データベース構造の論理記憶域全体を計画する際は、実際にデータベースを作成し、稼働したときに、この構造によって生じる影響を考慮します。次の項目について、データベースの論理記憶域構造が及ぼす影響を検討してください。

- Oracle Database を実行しているコンピュータのパフォーマンス
- データ・アクセス操作中のデータベースのパフォーマンス
- データベースのバックアップおよびリカバリの手順の効率

データベース・オブジェクトのリレーショナル設計と、各オブジェクトの記憶特性について計画します。オブジェクトを作成する前に、オブジェクトと各オブジェクトの物理記憶域間の関連について計画を立てることによって、1 つの単位としてのデータベースのパフォーマンスを直接制御できます。データベースの拡張計画についても必ず検討してください。

分散データベース環境では、この計画段階が非常に重要です。頻繁にアクセスされるデータの物理的な位置が、アプリケーションのパフォーマンスに大きな影響を及ぼします。

計画段階中に、データベースのバックアップ計画を作成します。バックアップの効率を改善するために、必要に応じて論理記憶域やデータベースの設計を変更します。

リレーショナル・データベース設計および分散データベース設計については、このマニュアルでは取り扱っていません。このような設計上の問題は、業界標準として認められている資料を参照してください。

データベースの論理記憶域構造、オブジェクトおよび整合性制約の作成方法は、第 II 部「Oracle Database の構造と記憶域」および第 III 部「スキーマ・オブジェクト」を参照してください。

タスク 4: データベースの作成とオープン

データベース設計が完了すると、データベースを作成し、オープンできます。データベースを作成するには、Database Configuration Assistant（DBCA）を使用してインストール時に作成する方法と、データベースを作成するためのスクリプトを用意する方法があります。

データベースの作成方法については第 2 章「Oracle Database の作成および構成」を、データベース起動時のガイドラインについては第 3 章「起動と停止」を参照してください。

タスク 5: データベースのバックアップ

データベース構造の作成後、データベースに対して予定していたバックアップ計画を実行します。追加の REDO ログ・ファイルを作成し、データベース全体の最初のバックアップ（オンラインまたはオフライン）を作成して、その後の定期的なデータベース・バックアップをスケジュールします。

> **関連項目：**『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』

タスク 6: システム・ユーザーの登録

データベース構造のバックアップが完了した後、Oracle のライセンス契約に従ってデータベースのユーザーを登録し、登録したユーザーに対して適切な権限とロールを付与できます。このタスクのガイドラインについては、第 6 章「ユーザーの管理とデータベースのセキュリティ保護」を参照してください。

タスク 7: データベース設計の実装

データベースを作成して起動し、システム・ユーザーを登録した後、必要なすべての表領域を作成して、計画したデータベースの論理構造を実装できます。表領域の作成を完了すると、データベース・オブジェクトを作成できます。

データベースの論理記憶域構造とオブジェクトの作成方法は、第 II 部「Oracle Database の構造と記憶域」および第 III 部「スキーマ・オブジェクト」を参照してください。

タスク 8: 実行データベースのバックアップ

データベースがすべて実装されたときに、再度バックアップを作成します。定期的にバックアップを作成するようにスケジュールし、また、データベース構造の変更を実装した直後にも必ずデータベースのバックアップを作成するようにします。

タスク 9: データベースのパフォーマンス・チューニング

データベースのパフォーマンスの最適化は、DBA の日常的な作業の 1 つです。Oracle Database では、DBA が各種ユーザー・グループへのリソースの割当てを制御できるように、データベース・リソース管理機能が提供されています。データベース・リソース・マネージャについては、第 25 章「Oracle Database Resource Manager を使用したリソース割当ての管理」を参照してください。

> **関連項目：** データベースとアプリケーションのチューニング方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

タスク 10: パッチのダウンロードとインストール

インストール後および定期的に、パッチをダウンロードしてインストールします。パッチは単一の暫定パッチおよびパッチ・セット（または**パッチ・リリース**）として使用できます。暫定パッチは個々のソフトウェアの不具合を修正するもので、使用中のインストールに必要な場合と必要ではない場合があります。パッチ・リリースは、すべての顧客に適用される不具合の修正の集合です。パッチ・リリースにはリリース番号が付けられています。たとえば、Oracle Database 10.2.0.0 をインストールした場合、最初のパッチ・リリースのリリース番号は 10.2.0.1 です。

> **関連項目：** パッチのダウンロードおよびインストールの手順は、使用中のプラットフォームの『Oracle Database インストレーション・ガイド』を参照してください。

タスク 11: 追加ホストへのロール・アウト

Oracle Database のインストールを正しく構成し、チューニングし、パッチを適用してテストした後、そのインストールを他のホストにロール・アウトする必要がある場合があります。ロール・アウトは、次のような場合に行います。

- 複数の本番データベース・システムがある場合
- 本番システムと同じ開発システムやテスト・システムを作成する場合

追加の各ホストでインストール、チューニングおよびパッチ適用を実行するかわりに、テスト済の Oracle Database インストールを他のホストに**クローニング**すると、時間を節約でき、不整合をなくすことができます。次の 2 種類のクローニングを使用できます。

- Oracle ホームのクローニング : Oracle ホーム・ディレクトリとサブディレクトリの構成済およびパッチ適用済バイナリのみが宛先のホストにコピーされ、新しい環境にあわせて調整されます。その後、このクローニングしたホームを使用してインスタンスを起動し、データベースを作成できます。

  Enterprise Manager の Oracle ホームのクローニング・ツールを使用すると、Oracle ホームを 1 つ以上の宛先ホストにクローニングできます。また、提供されているスクリプトのセットと Oracle Universal Installer を使用して、Oracle ホームを手動でクローニングすることもできます。

- データベースのクローニング : データベース・ファイルや初期化パラメータなどのチューニング済のデータベースが、既存の Oracle ホーム（おそらくクローニングしたホーム）にクローニングされます。

  Enterprise Manager のデータベースのクローニング・ツールを使用すると、Oracle データベース・インスタンスを既存の Oracle ホームにクローニングできます。

  関連項目：

  - Oracle ホームのクローニング方法の詳細は、『Oracle Universal Installer および Opatch ユーザーズ・ガイド』および Enterprise Manager のオンライン・ヘルプを参照してください。
  - データベースのクローニングの手順は、Enterprise Manager のオンライン・ヘルプを参照してください。

データベースに対するコマンドと SQL の発行

Oracle Database とのやり取りは主に SQL 文を発行して行います。Oracle Database では、データベースの起動と停止およびデータベース設定の変更などのコマンドを含む、SQL のスーパーセットもサポートされています。これらの SQL 文やコマンドを Oracle Database に発行する方法は 3 つあります。

- SQL*Plus のコマンドライン・インタフェースを使用する直接的な方法
- Oracle Enterprise Manager の GUI を使用する間接的な方法

  Oracle Enterprise Manager（Enterprise Manager）を使用すると、使いやすいグラフィカル・インタフェースを使用してデータベースを管理できます。SQL 文やコマンドは、Enterprise Manager が内部的に発行します。

  詳細は、『Oracle Database 2 日でデータベース管理者』を参照してください。
- SQL Developer を使用する直接的な方法

  開発者は SQL Developer を使用してデータベース・スキーマとアプリケーションの作成およびテストをしますが、SQL Developer はデータベースの管理作業にも使用できます。

  詳細は、『Oracle Database 2 日で開発者ガイド』を参照してください。

この項では、SQL*Plus を使用して SQL 文とコマンドをデータベースに発行する方法を中心に説明します。この項の内容は、次のとおりです。

SQL*Plus の概要

SQL*Plus は Oracle データベースへの主要なコマンドライン・インタフェースです。SQL*Plus は、データベースの起動と停止、データベース初期化パラメータの設定、ユーザーの作成と管理、データベース・オブジェクト（表や索引など）の作成と変更、データの挿入と更新、SQL 問合せの実行などを行うために使用します。

SQL 文とコマンドを発行する前に、データベースに接続する必要があります。SQL*Plus では、ローカル接続またはリモート接続が可能です。**ローカル接続**とは、SQL*Plus を実行しているのと同じコンピュータで動作している Oracle データベースに接続することです。**リモート接続**とは、リモート・コンピュータで動作している Oracle データベースにネットワークを介して接続することです。そのようなデータベースは**リモート・データベース**と呼ばれます。Oracle Database の完全インストール、Oracle Client のインストールまたは Instant Client のインストールを行うと、ローカル・コンピュータに SQL*Plus 実行可能ファイルが提供されます。

関連項目：『SQL*Plus ユーザーズ・ガイドおよびリファレンス』

SQL*Plus を使用したデータベースへの接続

Oracle Database のコンポーネントは、次のとおりです。

- プロセスとメモリーの集合体である Oracle Database インスタンス
- ユーザー・データとシステム・データが格納された一連のディスク・ファイル

SQL*Plus を使用して接続するときには、Oracle インスタンスに接続します。各インスタンスには、システム ID（SID）とも呼ばれるインスタンス ID があります。ホスト・コンピュータには、それぞれに独自のデータ・ファイル・セットを持つ複数の Oracle インスタンスが存在する場合があるため、接続するインスタンスを識別する必要があります。ローカル接続の場合は、オペレーティング・システムの環境変数を設定してインスタンスを識別します。リモート接続の場合は、ネットワーク・アドレスとデータ・サービス名を指定してインスタンスを識別します。ローカル接続とリモート接続のいずれの場合も、環境変数を設定して、SQL*Plus の実行可能ファイルをオペレーティング・システムが検出できるようにし、実行可能ファイルにサポート・ファイルとスクリプトへのパスを設定する必要があります。このため、SQL*Plus を使用して Oracle インスタンスに接続するには、次の手順を完了する必要があります。

関連項目： Oracle インスタンスに関するバックグラウンド情報は、『Oracle Database 概要』を参照してください。

手順 1: コマンド・ウィンドウのオープン

プラットフォームで必要な処理を実行して、オペレーティング・システムのコマンドを入力できるウィンドウを開きます。

プラットフォーム	処理
UNIX と Linux	ターミナル・セッションのオープン
Windows	「コマンド プロンプト」ウィンドウのオープン

手順 2: オペレーティング・システムの環境変数の設定

プラットフォームによっては、SQL*Plus を起動する前に、環境変数を設定するか少なくとも正しく設定されていることを確認することが必要な場合があります。

たとえば、ほとんどのプラットフォームでは、ORACLE_SID と ORACLE_HOME を設定する必要があります。さらに、PATH 環境変数に *ORACLE_HOME*/bin ディレクトリを含めるように設定することもお薦めします。一部のプラットフォームでは、さらに環境変数の設定が必要な場合があります。UNIX および Linux プラットフォームでは、オペレーティング・システムのコマンドを入力して環境変数を設定する必要があります。Windows プラットフォームでは、Windows レジストリ内の ORACLE_HOME と ORACLE_SID に Oracle Universal Installer（OUI）によって値が自動的に割り当てられます。インストール時にデータベースを作成しなかった場合は、レジストリ内の ORACLE_SID が OUI によって設定されないため、後でデータベースを作成するときに、ORACLE_SID 環境変数をコマンド・ウィンドウから設定する必要があります。

UNIX および Linux のインストールには、oraenv と coraenv の 2 つのスクリプトが含まれています。このスクリプトを使用すると、これらの環境変数を容易に設定できます。詳細は、『Oracle Database 管理者リファレンス for UNIX Systems』を参照してください。

いずれのプラットフォームでも、異なる Oracle ホームを使用するインスタンス間で切り替える場合は、ORACLE_HOME 環境変数を変更する必要があります。同じ Oracle ホームを複数のインスタンスで共有している場合は、インスタンスを切り替えるときに ORACLE_SID のみを変更する必要があります。

環境変数およびインスタンスの切替え方法の詳細は、『Oracle Database インストレーション・ガイド』または使用しているオペレーティング・システムの管理ガイドを参照してください。

例 1-1 UNIX の環境変数の設定（C シェル）

```
setenv ORACLE_SID orcl
setenv ORACLE_HOME /u01/app/oracle/product/11.1.0/db_1
setenv LD_LIBRARY_PATH $ORACLE_HOME/lib:/usr/lib:/usr/dt/lib:/usr/openwin/lib:/usr/ccs/lib
```

例 1-2 Windows の環境変数の設定

```
SET ORACLE_SID=orcl
```

例 1-2 では、ORACLE_HOME がレジストリに設定されており、ORACLE_SID は設定されていない（または別のインスタンスに接続するために ORACLE_SID のレジストリ値を上書きする必要がある）と仮定しています。

Windows では、コマンド・プロンプト・ウィンドウで設定した環境変数値でレジストリの値が上書きされます。

手順 3: SQL*Plus の起動

SQL*Plus を起動する手順は、次のとおりです。

1. 次のいずれかを実行します。
   - PATH 環境変数に *ORACLE_HOME*/bin が含まれていることを確認します。
   - *ORACLE_HOME*/bin ディレクトリに移動します。
2. 次のコマンドを入力します（UNIX と Linux では大 / 小文字が区別されます）。

```
sqlplus /nolog
```

手順 4: SQL*Plus の CONNECT 文の発行

最初に Oracle インスタンスに接続するか別のユーザーとして再接続するには、SQL*Plus の CONNECT 文を発行します。CONNECT 文の構文は、次のとおりです。

```
CONN[ECT] [logon] [AS {SYSOPER | SYSDBA}]
```

logon の構文は、次のとおりです。

```
{username | /}[@connect_identifier]
```

username を入力すると、パスワードの入力を求めるプロンプトが SQL*Plus によって表示されます。パスワードは入力しても画面には表示されません。

次の表で、CONNECT 文の構文の構成要素を説明します。

構文の構成要素	説明
/	接続要求の外部認証を要請します。このタイプの認証では、データベース・パスワードは使用されません。最も一般的な形式の外部認証はオペレーティング・システム認証です。この場合、特定のホスト・ユーザー・アカウントでホスト・オペレーティング・システムにログインしたことによって、データベース・ユーザーが認証されます。外部認証は、Oracle ウォレットやネットワーク・サービスで実行することもできます。詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。1-19 ページの「オペレーティング・システム認証の使用」も参照してください。
AS {SYSOPER | SYSDBA}	SYSOPER または SYSDBA のいずれかのシステム権限でデータベース・ユーザーが接続されていることを示します。この権限で接続できるのは、事前定義された特定の管理ユーザーまたはパスワード・ファイルに追加されているユーザーのみです。詳細は、1-16 ページの「管理権限」を参照してください。
username	有効なデータベース・ユーザー名。*username* をデータ・ディクショナリと一致させて、ユーザー・パスワードの入力を求めることによって、データベースが接続要求を認証します。
connect_identifier（1）	リモート接続用の Oracle Net 接続識別子。正確な構文は Oracle Net の構成によって異なります。省略すると、SQL*Plus はローカル・インスタンスへの接続を試みます。 一般的な接続識別子は *net service name* です。この識別子は、Oracle Net 接続記述子（ネットワーク・アドレスとデータベース・サービス名）の別名です。通常、この別名はローカル・コンピュータの tnsnames.ora ファイルで解決されますが、他の方法でも解決できます。 接続識別子の詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

構文の構成要素	説明
connect_identifier（2）	代替方法として、簡易接続の構文を接続識別子で使用できます。簡易接続では、クライアント（ローカル）コンピュータで Oracle Net Services を設定しなくても、すぐに TCP/IP 接続が可能になります。 接続識別子の簡易接続の構文は、次のとおりです。 *host*[:*port*] [/*service_name*] 各項目の意味は次のとおりです。 ■ *host* は、リモート・データベースがあるコンピュータのホスト名または IP アドレスです。 ■ *port* は *host* の Oracle Net リスナーがデータベース接続をリスニングする TCP ポートです。省略すると 1521 になります。 ■ *service_name* は、データベース・サービス名です。リモート・ホストの Net Services リスナーの構成にデフォルト・サービスが指定されている場合は省略できます。デフォルト・サービスが設定されていない場合は、*service_name* を指定する必要があります。通常、各データベースは、グローバル・データベース名と同じ名前のサービスを提供しています。たとえば、orcl.mycompany.com はその一例です。詳細は、2-41 ページの「データベース・サービスの定義」を参照してください。 簡易接続の詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

例 1-3

次の簡単な例では、ユーザー SYSTEM でローカル・データベースに接続します。SQL*Plus によって、ユーザー SYSTEM のパスワードの入力が求められます。

```
connect system
```

例 1-4

次の例では、SYSDBA 権限のユーザー SYS でローカル・データベースに接続します。SQL*Plus によって、ユーザー SYS のパスワードの入力が求められます。

```
connect sys as sysdba
```

ユーザー SYS として接続するときには、AS SYSDBA を指定して接続する必要があります。

例 1-5

次の例では、オペレーティング・システム認証を使用してローカルに接続します。

```
connect /
```

例 1-6

次の例では、オペレーティング・システム認証を使用して SYSDBA 権限でローカルに接続します。

```
connect / as sysdba
```

例 1-7

次の例では、簡易接続の構文を使用し、ユーザー salesadmin として、ホスト db1.mycompany.com で動作するリモート・データベースに接続します。Oracle Net リスナー（リスナー）は、デフォルト・ポート（1521）でリスニングしています。データベース・サービスは、sales.mycompany.com です。SQL*Plus によって、ユーザー salesadmin のパスワードの入力が求められます。

```
connect salesadmin@db1.mycompany.com/sales.mycompany.com
```

例 1-8

次の例は、デフォルト・ポート番号ではない 1522 でリスナーがリスニングしていること以外は例 1-7 と同じです。

```
connect salesadmin@db1.mycompany.com:1522/sales.mycompany.com
```

例 1-9

次の例では、ユーザー salesadmin として、ネット・サービス名 sales1 によって指定されたデータベース・サービスにリモート接続します。SQL*Plus によって、ユーザー salesadmin のパスワードの入力が求められます。

```
connect salesadmin@sales1
```

例 1-10

次の例では、外部認証を使用して、ネット・サービス名 sales1 によって指定されたデータベース・サービスにリモート接続します。

```
connect /@sales1
```

例 1-11

次の例では、外部認証を使用して SYSDBA 権限で、ネット・サービス名 sales1 によって指定されたデータベース・サービスにリモート接続します。

```
connect /@sales1 as sysdba
```

関連項目：

- 1-19 ページ「オペレーティング・システム認証の使用」
- CONNECT 文の詳細は、『SQL*Plus ユーザーズ・ガイドおよびリファレンス』を参照してください。
- ネット・サービス名の詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

Oracle Database ソフトウェアのリリースの識別

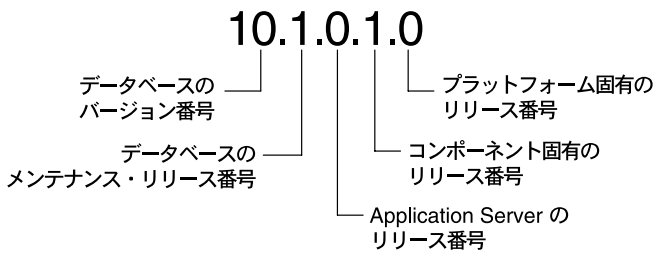
Oracle Database は継続的に開発が進められ、メンテナンスが必要なため、定期的に新リリースが作成されています。すべての顧客が、新リリースを最初に利用したり、既存のリリースに対する特定のメンテナンスを必要とするわけではありません。この結果、複数の製品リリースが同時に存在することになります。

特定のリリースを完全に識別するには、5 つの番号が必要です。以降の項では、各番号の意味について説明します。

リリース番号の形式

Oracle で使用されているリリースの命名体系を理解するために、Oracle Database のリリース番号「リリース 10.1.0.1.0」を例として説明します。

図 1-1 Oracle Database のリリース番号の例



注意： リリース 9.2 からは、Oracle Database のメンテナンス・リリースはリリース番号の 2 桁目の変更で示されます。以前のリリースでは、3 桁目が特定のメンテナンス・リリースを示していました。

データベースのバージョン番号

1 桁目が最も一般的な識別子です。この番号は、重要な新機能が含まれる、ソフトウェアの主な新規バージョンを表します。

データベースのメンテナンス・リリース番号

2 桁目は、メンテナンス・リリース・レベルを表します。新機能がいくつか含まれている場合もあります。

Application Server のリリース番号

3 桁目には、Oracle Application Server（OracleAS）のリリース・レベルが反映されます。

コンポーネント固有のリリース番号

4 桁目では、コンポーネント固有のリリース・レベルが識別されます。この桁の番号は、パッチ・セットや暫定リリースなどに応じてコンポーネントごとに異なる場合があります。

プラットフォーム固有のリリース番号

5 桁目では、プラットフォーム固有のリリースが識別されます。通常、これはパッチ・セットです。異なるプラットフォームに同等のパッチ・セットが必要な場合、この数字は影響を受けるプラットフォーム間で同じになります。

現行のリリース番号のチェック

現在インストールされている Oracle Database のリリースを識別し、使用している他のデータベース・コンポーネントのリリース・レベルを確認するには、データ・ディクショナリ・ビュー PRODUCT_COMPONENT_VERSION を問い合せます。問合せの例は、次のとおりです（コンポーネント・レベルの情報は、V$VERSION ビューを問い合せて確認することもできます）。他の製品のリリース・レベルは、データベース・サーバーと関係なく増加する場合があります。

```
COL PRODUCT FORMAT A35
COL VERSION FORMAT A15
COL STATUS FORMAT A15
SELECT * FROM PRODUCT_COMPONENT_VERSION;

PRODUCT                                    VERSION     STATUS
------------------------------------------ ----------- -----------
NLSRTL                                     10.2.0.1.0  Production
Oracle Database 10g Enterprise Edition     10.2.0.1.0  Prod
PL/SQL                                     10.2.0.1.0  Production
...
```

オラクル社にソフトウェアの問題を報告する際は、この問合せの結果を伝えることが重要です。

DBA のセキュリティと権限の概要

Oracle Database 管理者の管理タスクを実行するには、データベースとそのデータベースが稼働するサーバーのオペレーティング・システムで特定の権限が必要です。DBA アカウントへのアクセスは厳しく管理する必要があります。

この項の内容は、次のとおりです。

- DBA のオペレーティング・システム・アカウント
- 管理ユーザー・アカウント

DBA のオペレーティング・システム・アカウント

データベースに対する管理業務の多くを実行するには、オペレーティング・システム・コマンドを実行できる必要があります。Oracle Database が稼働するオペレーティング・システムによって異なりますが、オペレーティング・システムにアクセスするには、オペレーティング・システム・アカウント（ID）が必要になります。その場合、そのオペレーティング・システム・アカウントに対しては、他のデータベース・ユーザーには不要なオペレーティング・システム権限やアクセス権（Oracle Database ソフトウェアのインストールの実行など）が必要になります。Oracle Database ファイルを自分のアカウント内に格納する必要はありませんが、それらに対するアクセス権は必要です。

> **関連項目：** 使用しているオペレーティング・システム固有の Oracle マニュアルを参照してください。DBA のアカウントの作成方法は、オペレーティング・システムによって異なります。

管理ユーザー・アカウント

Oracle Database のインストール時には、次の 2 つの管理ユーザー・アカウントが自動的に作成されます。

- SYS（デフォルトのパスワード : CHANGE_ON_INSTALL）
- SYSTEM（デフォルトのパスワード : MANAGER）

注意： Oracle Universal Installer（OUI）および Database Configuration Assistant（DBCA）では、SYS および SYSTEM のパスワードの入力が要求され、それぞれデフォルトのパスワードである "change_on_install" または "manager" は受け入れられません。

データベースを手動で作成する場合は、これらのデフォルト・パスワードを使用しないで、SYS と SYSTEM のパスワードをデータベースの作成時に指定することをお薦めします。詳細は、2-16 ページの「データベースの保護 : ユーザー SYS および SYSTEM のパスワードの指定」を参照してください。

日常的な管理タスクを実施するときに使用する管理者ユーザーを最低 1 つ追加作成し、そのユーザーに適切な管理ロールを付与してください。これらの目的のために SYS および SYSTEM を使用しないでください。

セキュリティ拡張に関する注意： このリリースの Oracle Database と以降のリリースで、デフォルトのデータベース・ユーザー・アカウントのセキュリティを確保するために、セキュリティ機能が拡張されています。このリリースのセキュリティ・チェックリストは、『Oracle Database セキュリティ・ガイド』にあります。このチェックリストを参照し、その内容に従ってデータベースを構成してください。

SYS

Oracle Database を作成すると、ユーザー SYS が自動的に作成され、DBA ロールが付与されます。

データベースのデータ・ディクショナリの実表とビューはすべて、SYS スキーマに格納されます。この実表とビューは、Oracle Database の操作にとって非常に重要です。データ・ディクショナリの整合性を維持するために、SYS スキーマ内の表はデータベースによってのみ操作されます。ユーザーや DBA はそれらの表を修正したり、ユーザー SYS のスキーマ内に表を作成しないでください。（ただし、必要に応じてデータ・ディクショナリ設定の記憶域パラメータを変更することは可能です）。

ほとんどのデータベース・ユーザーに対して、SYS アカウントによる Oracle Database への接続を禁止する必要があります。

SYSTEM

Oracle Database を作成すると、ユーザー SYSTEM が自動的に作成され、DBA ロールが付与されます。

このユーザー名 SYSTEM を使用して、管理情報を表示する追加表とビュー、および各種の Oracle Database のオプションとツール製品で使用される内部表とビューが作成されます。管理ユーザー以外のユーザーに関係する表の格納に、SYSTEM スキーマを使用しないでください。

DBA ロール

事前に定義された DBA ロールは、Oracle Database のインストールで自動的に作成されます。このロールには、ほとんどのデータベース・システム権限が含まれています。このため、DBA ロールは実際の DBA にのみ付与してください。

注意： DBA ロールには、SYSDBA または SYSOPER システム権限が含まれていません。これらは、データベースとインスタンスの起動や停止など、基本的なデータベース管理タスクの実行を管理者に許可する特別な管理権限です。これらのシステム権限については、1-16 ページの「管理権限」を参照してください。

DBA の認証

DBA は、データベースの起動や停止などの特別な操作を実行します。これらの操作は DBA のみが実行する必要があるため、DBA のユーザー名には安全性の高い認証方式が必要です。

この項の内容は、次のとおりです。

管理権限

管理者が基本的なデータベース操作を実行するために必要な管理権限は、SYSDBA および SYSOPER という 2 つの特別なシステム権限によって付与されます。必要な認可レベルに応じて、どちらか一方の権限を DBA に付与する必要があります。

注意： SYSDBA および SYSOPER システム権限を使用すると、データベースがオープンしていなくてもデータベース・インスタンスにアクセスできます。これらの権限の制御は、完全にデータベース機能の範囲外にあります。

SYSDBA と SYSOPER 権限も、他の方法では権限を付与できない特定のデータベース操作を実行するための接続の一種とみなすことができます。たとえば、SYSDBA 権限がある場合は、CONNECT AS SYSDBA を指定することでデータベースに接続できます。

SYSDBA と SYSOPER

SYSDBA および SYSOPER システム権限で許可されている操作は、次のとおりです。

システム権限	許可されている操作
SYSDBA	■ STARTUP および SHUTDOWN 操作の実行 ■ ALTER DATABASE OPEN/MOUNT/BACKUP/CHARACTER SET ■ CREATE DATABASE ■ DROP DATABASE ■ CREATE SPFILE ■ ALTER DATABASE ARCHIVELOG ■ ALTER DATABASE RECOVER ■ RESTRICTED SESSION 権限を含む 実際は、このシステム権限を使用することにより、ユーザー SYS として接続できます。

システム権限	許可されている操作
SYSOPER	■ STARTUP および SHUTDOWN 操作の実行 ■ CREATE SPFILE ■ ALTER DATABASE OPEN/MOUNT/BACKUP ■ ALTER DATABASE ARCHIVELOG ■ ALTER DATABASE RECOVER（完全リカバリのみ。UNTIL TIME|CHANGE|CANCEL|CONTROLFILE など、不完全リカバリの形式では、SYSDBA で接続する必要があります。） ■ RESTRICTED SESSION 権限を含む この権限を使用すると、ユーザー・データの表示を除く基本操作を実行できます。

これらの権限の使用を許可する方法は、使用する認証方式によって異なります。

SYSDBA または SYSOPER 権限で接続した場合は、一般的にユーザー名に対応付けられているスキーマではなく、デフォルトのスキーマで接続します。デフォルトのスキーマは、SYSDBA の場合は SYS、SYSOPER の場合は PUBLIC です。

管理権限での接続：例

この例では、ユーザーが SYSDBA システム権限で接続したときに、他のスキーマ（SYS）に割り当てられることを説明します。SYSDBA システム権限が付与されているサンプル・ユーザー oe が、次の文を発行したとします。

```
CONNECT oe
CREATE TABLE admin_test(name VARCHAR2(20));
```

後で、ユーザー oe が次の文を発行します。

```
CONNECT oe AS SYSDBA
SELECT * FROM admin_test;
```

ユーザー oe は次のエラー・メッセージを受け取ります。

```
ORA-00942: 表またはビューが存在しません。
```

SYSDBA で接続しているため、ユーザー oe は現在 SYS スキーマを参照していますが、表は oe スキーマに作成されています。

関連項目：

- 1-19 ページ「オペレーティング・システム認証の使用」
- 1-21 ページ「パスワード・ファイル認証の使用」

DBA の認証方式の選択

DBA は、他のユーザーと同様に（アカウント・パスワードを使用して）、データベースのデータ・ディクショナリを介して認証できます。Oracle Database 11*g* リリース 1 からは、データベース・パスワードは大 / 小文字が区別されることに注意してください（SEC_CASE_SENSITIVE_LOGON 初期化パラメータを FALSE に設定すると、大 / 小文字の区別を無効にして、リリース 11*g* より前の動作に戻すことができます）。

通常のデータ・ディクショナリ認証に加えて、SYSDBA または SYSOPER 権限のある DBA の認証には、次の方法を使用できます。

- オペレーティング・システム（OS）認証
- パスワード・ファイル
- Oracle Internet Directory などのネットワーク・ベースの認証サービスを使用した厳密認証

これらの方法は、データベースが起動していない場合、または使用可能でない場合に、DBA を認証するために必要です（データベースが使用可能な場合も使用できます）。

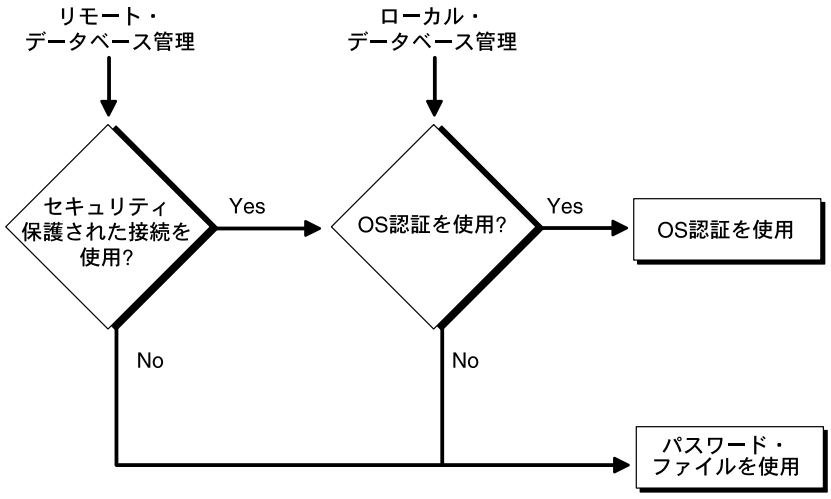
ここからは、オペレーティング・システム認証とパスワード・ファイル認証について説明します。ネットワーク・ベースの認証を使用してデータベース管理者を認証する方法の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

注意：

- これらの認証方式は、旧バージョンの Oracle Database で提供されていた CONNECT INTERNAL 構文にかわるものです。CONNECT INTERNAL は現在サポートされていません。
- オペレーティング・システム認証は、パスワード・ファイル認証より優先されます。オペレーティング・システム認証の要件を満たしている場合は、パスワード・ファイルを使用している場合でもオペレーティング・システム認証によって認証されます。

データベースと同じマシン上でデータベースをローカルで管理するか、または単一のリモート・クライアントから複数の異なるデータベースを管理するかによって、どちらの認証方式を選択するかが決まります。図 1-2 は、DBA 用に選択できる認証方式を示しています。

図 1-2 DBA の認証方式



リモート・データベース管理の場合は、Oracle Net 関連マニュアルを参照して、セキュリティで保護された接続を使用しているかどうかを判断してください。TCP/IP や DECnet などの最も一般的な接続プロトコルは、セキュリティによって保護されていません。

関連項目：

- ネットワーク・ベースの認証を使用してデータベース管理者を認証する方法の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。
- 『Oracle Database Net Services 管理者ガイド』

セキュリティで保護されていないリモート接続

セキュリティで保護されていない接続で、権限を持つユーザーとして Oracle Database に接続するには、パスワード・ファイルによる認証を受ける必要があります。パスワード・ファイル認証を使用すると、SYSDBA または SYSOPER システム権限を付与されたデータベース・ユーザー名を追跡管理するために、パスワード・ファイルが使用されます。この認証方式については、1-21 ページの「パスワード・ファイル認証の使用」を参照してください。

ローカル接続およびセキュリティで保護されたリモート接続

ローカル接続またはセキュリティで保護されたリモート接続で、権限を持つユーザーとして Oracle Database に接続するには、次の 2 通りの方法があります。

- データベースにパスワード・ファイルがあり、ユーザーが SYSDBA または SYSOPER システム権限を付与されている場合は、パスワード・ファイルを使用して接続し、認証を受けることができます。
- データベースがパスワード・ファイルを使用していない場合、あるいはユーザーが SYSDBA または SYSOPER 権限を付与されていないため、パスワード・ファイルに登録されていない場合は、オペレーティング・システム認証を使用できます。ほとんどのオペレーティング・システムでは、データベース管理者のオペレーティング・システム認証のために、データベース管理者のオペレーティング・システム・ユーザー名を特別なグループに登録します。このグループは一般に OSDBA と呼ばれます。そのグループのユーザーには、SYSDBA 権限が付与されます。ユーザーに SYSOPER 権限を付与するには、同様のグループ OSOPER が使用されます。

オペレーティング・システム認証の使用

ここでは、オペレーティング・システムを使用した管理者の認証方法について説明します。

OSDBA と OSOPER

オペレーティング・システム認証を使用する場合は、2 つの特別なオペレーティング・システム・グループによってデータベース管理者の接続が制御されます。これらのグループは一般に OSDBA および OSOPER と呼ばれます。各グループはデータベースのインストール・プロセスで作成され、特定の名前が割り当てられます。グループの名前は、次の表に示すようにオペレーティング・システムによって異なります。

オペレーティング・システム・グループ	UNIX ユーザー・グループ	Windows ユーザー・グループ
OSDBA	dba	ORA_DBA
OSOPER	oper	ORA_OPER

Oracle Universal Installer で提示されるデフォルト名は変更できます。OSDBA および OSOPER グループは、オペレーティング・システム固有の方法で作成されます。

OSDBA または OSOPER グループでのメンバーシップは、データベースへの接続に次のように影響します。

- データベースへの接続時に、OSDBA グループに属するユーザーが AS SYSDBA を指定した場合、そのユーザーは、SYSDBA システム権限でデータベースに接続します。
- データベースへの接続時に、OSOPER グループに属するユーザーが AS SYSOPER を指定した場合、そのユーザーは、SYSOPER システム権限でデータベースに接続します。

- これらのオペレーティング・システム・グループに属していないユーザーが SYSDBA または SYSOPER として接続しようとした場合、CONNECT コマンドを発行するとエラーが発生します。

関連項目： OSDBA および OSOPER グループの作成方法の詳細は、使用しているオペレーティング・システム固有の Oracle マニュアルを参照してください。

オペレーティング・システム認証を使用するための準備

オペレーティング・システムを使用して管理ユーザーを認証するには、次の手順を実行する必要があります。

1. ユーザーのオペレーティング・システム・アカウントを作成します。
2. オペレーティング・システムで定義された OSDBA または OSOPER グループにアカウントを追加します。

オペレーティング・システム認証を使用した接続

次のどちらかの SQL*Plus コマンドを入力すると、ユーザーが管理ユーザーとして認証され、ローカル・データベースに接続できます。

```
CONNECT / AS SYSDBA
CONNECT / AS SYSOPER
```

セキュリティで保護された接続を介したリモート・オペレーティング・システム認証がサポートされているのは、Windows プラットフォームのみです。リモート・データベースのネット・サービス名を指定する必要があります。

```
CONNECT /@net_service_name AS SYSDBA
CONNECT /@net_service_name AS SYSOPER
```

クライアント・コンピュータとデータベース・ホスト・コンピュータの両方が Windows ドメイン上にあることが必要です。

関連項目：

- 1-8 ページ「SQL*Plus を使用したデータベースへの接続」
- CONNECT コマンドの構文は、『SQL*Plus ユーザーズ・ガイドおよびリファレンス』を参照してください。

パスワード・ファイル認証の使用

ここでは、パスワード・ファイル認証を使用した管理ユーザーの認証方法について説明します。

パスワード・ファイル認証を使用するための準備

パスワード・ファイル認証を使用して管理ユーザーを認証するには、次の手順を実行する必要があります。

1. 作成していない場合は、次のように ORAPWD ユーティリティを使用してパスワード・ファイルを作成します。

   ```
   ORAPWD FILE=filename ENTRIES=max_users
   ```

   詳細は、1-23 ページの「パスワード・ファイルの作成とメンテナンス」を参照してください。

   注意：

   - Oracle Database のインストール・プロセスで Database Configuration Assistant（DBCA）を起動した場合は、DBCA によってパスワード・ファイルが作成されます。
   - Oracle Database 11*g* リリース 1 からは、IGNORECASE = Y というコマンドライン引数を組み込まないかぎり、パスワード・ファイル内のパスワードは大 / 小文字が区別されます。

2. REMOTE_LOGIN_PASSWORDFILE 初期化パラメータを EXCLUSIVE に設定します（これはデフォルトです）。

   注意： REMOTE_LOGIN_PASSWORDFILE は静的な初期化パラメータであるため、変更するにはデータベースを再起動する必要があります。

3. ユーザー SYS（または管理権限を持つ他のユーザー）としてデータベースに接続します。
4. データベース内にユーザーが存在しない場合は、ユーザーを作成し、パスワードを割り当てます。

   Oracle Database 11*g* リリース 1 からは、データベース・パスワードは大 / 小文字が区別されることに注意してください（SEC_CASE_SENSITIVE_LOGON 初期化パラメータを FALSE に設定すると、大 / 小文字の区別を無効にして、リリース 11*g* より前の動作に戻すことができます）。

5. 次のコマンドを入力して、ユーザーに SYSDBA または SYSOPER システム権限を付与します。

   ```
   GRANT SYSDBA to oe;
   ```

   この文を実行すると、パスワード・ファイルにユーザーが追加され、AS SYSDBA として接続できるようになります。

   関連項目： パスワード・ファイルの作成とメンテナンスの方法は、1-23 ページの「パスワード・ファイルの作成とメンテナンス」を参照してください。

パスワード・ファイル認証を使用した接続

SQL*Plus の CONNECT コマンドを使用すると、管理ユーザーが認証され、ローカルまたはリモート・データベースに接続できます。接続時には、ユーザー名とパスワード、および AS SYSDBA または AS SYSOPER 句を指定します。Oracle Database 11*g* リリース 1 からは、IGNORECASE = Y オプションを指定してパスワード・ファイルを作成しないかぎり、パスワードは大 / 小文字が区別されます。

たとえば、ユーザー oe は SYSDBA 権限を付与されているため、次のコマンドで接続できます。

```
CONNECT oe AS SYSDBA
```

しかし、ユーザー oe は SYSOPER 権限を付与されていないため、次のコマンドは失敗します。

```
CONNECT oe AS SYSOPER
```

注意： オペレーティング・システム認証は、パスワード・ファイル認証より優先されます。特に、オペレーティング・システムの OSDBA または OSOPER グループに属しているユーザーが、SYSDBA または SYSOPER で接続すると、指定したユーザー名とパスワードに関係なく、関連付けられた管理権限を使用して接続されます。

ユーザーが OSDBA グループにも OSOPER グループにも属しておらず、パスワード・ファイルにも指定されていない場合、SYSDBA または SYSOPER として接続しようとすると失敗します。

関連項目：

- 1-8 ページ「SQL*Plus を使用したデータベースへの接続」
- CONNECT コマンドの構文は、『SQL*Plus ユーザーズ・ガイドおよびリファレンス』を参照してください。

パスワード・ファイルの作成とメンテナンス

パスワード・ファイル作成ユーティリティ ORAPWD を使用して、パスワード・ファイルを作成できます。一部のオペレーティング・システムでは、標準インストール時にこのファイルを作成できます。

この項の内容は、次のとおりです。

関連項目：

- 1-21 ページ「パスワード・ファイル認証の使用」
- 1-18 ページ「DBA の認証方式の選択」

ORAPWD の使用方法

ORAPWD コマンドの構文は次のとおりです。

```
ORAPWD FILE=filename [ENTRIES=numusers]
   [FORCE={Y|N}] [IGNORECASE={Y|N}] [NOSYSDBA={Y|N}]
```

次の表にコマンドの引数がまとめられています。

引数	説明
FILE	パスワード・ファイルに割り当てる名前。名前要件については、使用しているオペレーティング・システムのマニュアルを参照してください。完全なパスを指定する必要があります。ファイル名のみを指定した場合、ファイルはカレント・ディレクトリに書き込まれます。
ENTRIES	（オプション）ファイルに指定できる最大エントリ（ユーザー・アカウント）数。
FORCE	（オプション）y の場合は、既存のパスワード・ファイルが上書きされます。
IGNORECASE	（オプション）y の場合、パスワードは大 / 小文字が区別されません。
NOSYSDBA	（オプション）Data Vault インストール用。詳細は、使用中のプラットフォームの Data Vault インストール・ガイドを参照してください。

等号（=）文字の前後にスペースは使用できません。

SYS のパスワードの入力を求めるコマンド・プロンプトが表示され、作成されたパスワード・ファイルにそのパスワードが保存されます。

例

次のコマンドを使用すると、orapworcl という名前のパスワード・ファイルが作成されます。このパスワード・ファイルでは、30 人までの権限を持つユーザーと各ユーザー固有のパスワードを設定できます。

```
orapwd FILE=orapworcl ENTRIES=30
```

ORAPWD コマンドライン引数の説明

次の項では、ORAPWD コマンドライン引数について説明します。

FILE

この引数には、作成するパスワード・ファイルの名前を設定します。ファイルにはフルパス名を指定する必要があります。ファイル名のみを指定した場合、ファイルはカレント・ディレクトリに書き込まれます。このファイルの内容は暗号化されているため、ユーザーは直接読むことができません。これは必須の引数です。

パスワード・ファイルで許可されているファイル名のタイプは、オペレーティング・システムによって異なります。一部のオペレーティング・システムでは、パスワード・ファイルを特定の形式で作成して特定のディレクトリに格納する必要があります。また、環境変数を使用してパスワード・ファイルの名前と位置を指定できるオペレーティング・システムもあります。Unix および Linux オペレーティング・システムの場合の名前と位置に関する情報は、『Oracle Database 管理者リファレンス for UNIX Systems』を参照してください。Windows の場合は、『Oracle Database プラットフォーム・ガイド for Microsoft Windows』を参照してください。その他のオペレーティング・システムについては、使用しているオペレーティング・システムのマニュアルを参照してください。

Oracle Real Application Clusters を使用して複数の Oracle Database インスタンスを稼働している場合は、各インスタンスの環境変数が同じパスワード・ファイルを指している必要があります。

注意： パスワード・ファイルやパスワード・ファイルの位置を特定する環境変数の保護は、システムのセキュリティにとって非常に重要です。パスワード・ファイルや環境変数にアクセスできるユーザーは、接続に対するセキュリティを脅かす潜在的な可能性を持っています。

ENTRIES

この引数には、パスワード・ファイルで受け入れるエントリの数を指定します。この数は、データベースに SYSDBA または SYSOPER として接続できるユーザーの数に対応します。ORAPWD ユーティリティでは、オペレーティング・システム・ブロックがいっぱいになるまでパスワード・エントリの割当てが継続されるため、実際にはユーザー数より多くのエントリを入力できます。たとえば、オペレーティング・システムのブロック・サイズが 512 バイトの場合は、4 つのパスワード・エントリが格納されます。そのため、割り当てられたパスワード・エントリの数は常に 4 の倍数になります。

パスワード・ファイルにユーザーを追加したり、パスワード・ファイルからユーザーを削除したりすると、エントリは再利用されます。REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE を指定し、ユーザーに SYSDBA および SYSOPER 権限を付与する場合、この引数は必須です。

注意： 割り当てられたパスワード・エントリ数を超える場合は、新しいパスワード・ファイルを作成する必要があります。新しいパスワード・ファイルを作成しなくて済むように、必要と思われる数よりも大きいエントリ数を指定してください。

FORCE

この引数を Y に設定すると、既存のパスワード・ファイルを上書きできます。同じ名前のパスワード・ファイルがすでに存在している場合に、この引数を省略するかまたは N に設定すると、エラーが返されます。

IGNORECASE

この引数を y に設定すると、パスワードは大 / 小文字が区別されません。つまり、ログイン時にユーザーが指定するパスワードとパスワード・ファイル内のパスワードを比較するときに、大 / 小文字は無視されます。

関連項目： パスワードの大 / 小文字区別の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

REMOTE_LOGIN_PASSWORDFILE の設定

パスワード・ファイルを作成する他に、初期化パラメータ REMOTE_LOGIN_PASSWORDFILE を適切な値に設定する必要があります。認識される値は、次のとおりです。

- NONE: このパラメータを NONE に設定すると、Oracle Database はパスワード・ファイルが存在しない場合と同じように動作します。つまり、セキュリティで保護されていない接続では、権限付きの接続は許可されません。
- EXCLUSIVE: （デフォルト）EXCLUSIVE パスワード・ファイルは、1 つのデータベースの 1 つのインスタンスでのみ使用できます。EXCLUSIVE ファイルのみ更新可能です。EXCLUSIVE パスワード・ファイルでは、ユーザーを追加、変更および削除できます。また、ALTER USER コマンドを使用して SYS パスワードを変更できます。
- SHARED: SHARED パスワード・ファイルは、同じサーバーで稼働している複数のデータベース、または Oracle Real Application Clusters（RAC）データベースの複数のインスタンスで使用できます。SHARED パスワード・ファイルは変更できません。つまり、SHARED パスワード・ファイルにはユーザーを追加できません。ユーザーを追加しようとしたり、SYS のパスワードあるいは SYSDBA または SYSOPER 権限のあるユーザーのパスワードを変更しようとすると、エラーが生成されます。REMOTE_LOGIN_PASSWORDFILE が EXCLUSIVE に設定されている間に、SYSDBA または SYSOPER システム権限を必要とするすべてのユーザーをパスワード・ファイルに追加する必要があります。すべてのユーザーを追加した後で、REMOTE_LOGIN_PASSWORDFILE を SHARED に変更すると、ファイルを共有できます。

  このオプションは、複数のデータベースまたは RAC データベースを管理する場合に役立ちます。

REMOTE_LOGIN_PASSWORDFILE が EXCLUSIVE または SHARED に設定されているときに、パスワード・ファイルが欠落している場合は、REMOTE_LOGIN_PASSWORDFILE を NONE に設定した場合と同じです。

注意： REMOTE_LOGIN_PASSWORDFILE が SHARED に設定されている場合、SYS のパスワードは変更できません。パスワードを変更しようとすると、エラー・メッセージが発行されます。

パスワード・ファイルへのユーザーの追加

ユーザーに SYSDBA または SYSOPER 権限を付与すると、そのユーザーの名前と権限情報がパスワード・ファイルに追加されます。データベースに EXCLUSIVE パスワード・ファイルがない場合、つまり、初期化パラメータ REMOTE_LOGIN_PASSWORDFILE が NONE または SHARED であるか、パスワード・ファイルが欠落している場合は、これらの権限を付与しようとするとエラーが発行されます。

ユーザーがこの 2 つの権限のうち 1 つでも持っている間は、そのユーザーの名前がパスワード・ファイルに残っています。これらの権限を両方とも取り消すと、Oracle Database によって、そのユーザーはパスワード・ファイルから削除されます。

パスワード・ファイルを作成した新規ユーザーの追加方法

パスワード・ファイルを作成して新規ユーザーを追加するには、次の手順を実行します。

1. 1-23 ページの「ORAPWD の使用方法」に記載されている指示に従って、パスワード・ファイルを作成します。
2. REMOTE_LOGIN_PASSWORDFILE 初期化パラメータを EXCLUSIVE に設定します（これはデフォルトです）。

注意： REMOTE_LOGIN_PASSWORDFILE は静的な初期化パラメータであるため、変更するにはデータベースを再起動する必要があります。

---

3. 次の例に示すように、SYSDBA 権限で接続し、プロンプトが表示されたら SYS のパスワードを入力します。

```
CONNECT SYS AS SYSDBA
```

4. 必要であれば、インスタンスを起動してデータベースを作成します。または、既存のデータベースをマウントしてオープンします。
5. 必要に応じて、ユーザーを登録します。DBA 自身、および必要に応じて他のユーザーに SYSDBA または SYSOPER 権限を付与します。この後の「SYSDBA および SYSOPER 権限の付与と取消し」を参照してください。

SYSDBA および SYSOPER 権限の付与と取消し

データベースで EXCLUSIVE パスワード・ファイルを使用している場合は、次の例に示すように、GRANT 文を使用して、ユーザーに SYSDBA または SYSOPER システム権限を付与します。

```
GRANT SYSDBA TO oe;
```

ユーザーの SYSDBA または SYSOPER システム権限を取り消すには、次のように REVOKE 文を使用します。

```
REVOKE SYSDBA FROM oe;
```

SYSDBA と SYSOPER は最も強力なデータベース権限であるため、WITH ADMIN OPTION は、GRANT 文で使用されません。つまり、権限受領者が別のユーザーに SYSDBA または SYSOPER 権限を付与することはできません。現在 SYSDBA として接続しているユーザーのみが、別のユーザーに SYSDBA または SYSOPER システム権限を付与できます。また、別のユーザーのシステム権限を取り消すこともできます。この 2 つの権限は、ロールには付与できません。ロールはデータベースを起動しないかぎり使用できないためです。SYSDBA と SYSOPER のデータベース権限とオペレーティング・システム・ロールを混同しないでください。

> **関連項目：** システム権限の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

パスワード・ファイル・メンバーの表示

データベースの SYSDBA または SYSOPER システム権限を付与されているユーザーを確認するには、V$PWFILE_USERS ビューを使用します。このビューで表示される列は、次のとおりです。

列	説明
USERNAME	パスワード・ファイルで認識されるユーザー名。
SYSDBA	この列の値が TRUE の場合、そのユーザーは SYSDBA システム権限でログインできます。
SYSOPER	この列の値が TRUE の場合、そのユーザーは SYSOPER システム権限でログインできます。

パスワード・ファイルのメンテナンス

この項の内容は、次のとおりです。

- パスワード・ファイルがいっぱいになった場合の、パスワード・ファイルのユーザー数の追加
- パスワード・ファイルの削除

パスワード・ファイルのユーザー数の追加

ユーザーに SYSDBA または SYSOPER システム権限を付与しようとしたときに、パスワード・ファイルが満杯（ORA-01996）というエラーが出力された場合は、よりサイズの大きいパスワード・ファイルを作成し、ユーザーに再度権限を付与する必要があります。

パスワード・ファイルの置換

パスワード・ファイルを置換するには、次の手順を実行します。

1. V$PWFILE_USERS ビューを問い合せて、どのユーザーが SYSDBA または SYSOPER 権限を持っているかを確認します。
2. 既存のパスワード・ファイルを削除します。
3. 1-23 ページの「ORAPWD の使用方法」の指示に従って、ORAPWD ユーティリティを使用してパスワード・ファイルを新しく作成します。ENTRIES パラメータには、必要と思われる数よりも大きい数を指定してください。
4. 1-25 ページの「パスワード・ファイルへのユーザーの追加」の指示に従います。

パスワード・ファイルの削除

ユーザー認証にパスワード・ファイルを使用する必要がなくなったと判断した場合は、パスワード・ファイルを削除して、REMOTE_LOGIN_PASSWORDFILE 初期化パラメータを必要に応じて NONE にリセットできます。このファイルを削除した後は、オペレーティング・システムによって認証されたユーザーのみが SYSDBA または SYSOPER データベース管理の操作を実行できます。

データ・ユーティリティ

Oracle Database のデータのメンテナンスに役立つ Oracle ユーティリティが用意されています。

SQL*Loader

SQL*Loader は、データベース管理者と Oracle Database の他のユーザーの双方によって使用されます。それは、オペレーティング・システムの標準ファイル（テキストや C データ形式のファイルなど）からデータベース表にデータをロードします。

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

データ・ポンプ・ユーティリティを使用すると、データをアーカイブしたり、Oracle Database の既存のデータを他へ移動することができます。従来のインポート・ユーティリティを使用して以前のリリースからデータをインポートすることも可能です。従来のエクスポート・ユーティリティは、バージョン 11*g* から通常の使用ではサポートされなくなりました。特殊な状況での使用のみにとどめることをお薦めします。

> **関連項目：** これらのユーティリティの詳細は、『Oracle Database ユーティリティ』を参照してください。

2

Oracle Database の作成および構成

この章の内容は次のとおりです。

関連項目：

- 基礎となるオペレーティング・システム・ファイルが Oracle Database によって自動的に作成および管理されるデータベースの作成方法の詳細は、第 15 章「Oracle Managed Files の使用」を参照してください。
- Oracle Real Application Clusters（RAC）環境でのデータベース作成に関する情報は、使用しているプラットフォーム固有の Oracle Real Application Clusters インストレーション・ガイドを参照してください。

Oracle Database の作成の概要

この項で説明されているガイドラインの一部を使用してデータベースを計画した後、グラフィカル・ツールまたは SQL コマンドを使用してデータベースを作成できます。通常は、Oracle Database ソフトウェアのインストール時にデータベースを作成します。ただし、インストール後にもデータベースを作成できます。インストール後にデータベースを作成する理由には、次のものがあります。

- Oracle Universal Installer（OUI）を使用してソフトウェアのみをインストールして、データベースを作成しなかったため。
- 既存の Oracle データベースと同じホスト・コンピュータに別のデータベース（およびデータベース・インスタンス）を作成するため。この場合、この章では、新しいデータベースが既存のデータベースと同じ Oracle ホームを使用すると仮定します。OUI を再度実行すれば、新しい Oracle ホームにデータベースを作成することもできます。
- データベースのコピー（クローン）を作成するため。

データベースを作成する具体的な方法には、次の方法があります。

- Database Configuration Assistant（DBCA）グラフィカル・ツールを使用

  2-5 ページの「DBCA を使用したデータベースの作成」を参照してください。

- CREATE DATABASE SQL 文を使用

  2-6 ページの「CREATE DATABASE 文を使用したデータベースの作成」を参照してください。

データベースを作成する前の考慮点

データベースの作成では、いくつかのオペレーティング・システム・ファイルを準備することで、それらのファイルを Oracle Database としてまとめて動作できるようにします。データベースは、データファイルの数やアクセスするインスタンスの数にかかわらず、一度のみ作成します。データベースの作成では、既存のデータベース内の情報を消去し、同じ名前と物理構造を持つ新規データベースを作成することもできます。

この項の内容は、次のとおりです。

データベース作成計画

データベースの作成を準備するには、調査と綿密な計画が必要です。表 2-1 に、推奨する処理を示します。

表 2-1 データベース計画タスク

処理	追加情報
データベース表と索引を計画し、それらが必要とする領域を見積ります。	第 II 部「Oracle Database の構造と記憶域」 第 III 部「スキーマ・オブジェクト」

表 2-1 データベース計画タスク（続き）

処理	追加情報
データベースを構成する基礎となるオペレーティング・システム・ファイルのレイアウトを計画します。ファイルを適切に分散すると、ファイル・アクセス時の I/O が分散されるため、データベースのパフォーマンスを大幅に向上できます。Oracle ソフトウェアをインストールしてデータベースを作成するときに I/O を分散させるには、いくつかの方法があります。たとえば、REDO ログ・ファイルを異なるディスクに配置したり、ストライプ化を使用できます。競合が少なくなるようにデータファイルを配置できます。また、データの密度（データ・ブロック当たりの行数）を制御できます。フラッシュ・リカバリ領域を作成する場合は、データファイルとは異なる記憶デバイスに配置することをお薦めします。	『Oracle Database パフォーマンス・チューニング・ガイド』 『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』 該当の Oracle Database インストレーション・ガイドなど、使用しているオペレーティング・システム固有の Oracle マニュアル
データベース記憶域を構成するオペレーティング・システム・ファイルの作成および管理に、Oracle Managed Files および自動ストレージ管理の使用を検討します。	第 15 章「Oracle Managed Files の使用」 『Oracle Database ストレージ管理者ガイド』
グローバル・データベース名を選択します。グローバル・データベース名とは、ネットワーク構造内におけるデータベースの名前と位置です。グローバル・データベース名を作成するには、DB_NAME と DB_DOMAIN の 2 つの初期化パラメータを設定します。	2-26 ページ「グローバル・データベース名の決定」
初期化パラメータ・ファイルに含まれる初期化パラメータについて理解します。特に**サーバー・パラメータ・ファイル**の概要と操作について理解します。サーバー・パラメータ・ファイルを使用すると、初期化パラメータをサーバー側のディスク・ファイルに永続的に格納して管理できます。	2-24 ページ「初期化パラメータと初期化パラメータ・ファイルの概要」 2-32 ページ「サーバー・パラメータ・ファイルの概要」 『Oracle Database リファレンス』

表 2-1 データベース計画タスク（続き）

処理	追加情報
データベース・キャラクタ・セットを選択します。 データ・ディクショナリ内のデータも含め、すべての文字データは、そのデータベースのキャラクタ・セットに格納されます。データベースの作成時には、データベース・キャラクタ・セットを指定する必要があります。 異なるキャラクタ・セットを使用したクライアントがデータベースにアクセスする場合は、クライアント・キャラクタ・セットをすべて含むスーパーセットを選択します。そうしないと、キャラクタの変換が必要な場合、オーバーヘッドが増大し、データが消失する危険性が大きくなります。 代替キャラクタ・セットを指定することもできます。 **注意：** AL32UTF8 は、XMLType データに適した Oracle Database キャラクタ・セットです。これは、有効なすべての XML 文字をサポートする IANA 登録済の標準 UTF-8 エンコーディングと同等です。 Oracle Database のデータベース・キャラクタ・セット UTF8（ハイフンなし）と、データベース・キャラクタ・セット AL32UTF8 またはキャラクタ・エンコーディング UTF-8 を混同しないでください。データベース・キャラクタ・セット UTF8 は、AL32UTF8 に置き換わっています。XML データに UTF8 を使用しないでください。UTF8 でサポートされるのは Unicode バージョン 3.1 以下のみで、有効な XML 文字がすべてサポートされるわけではありません。AL32UTF8 にはこのような制限はありません。 XML データにデータベース・キャラクタ・セット UTF8 を使用すると、致命的なエラーが発生したり、セキュリティに悪影響を与える可能性があります。データベース・キャラクタ・セットでサポートされていない文字が入力文書要素名で使用されている場合は、置換文字（通常は「?」）で置き換えられます。この結果、解析が終了して例外が発生します。	『Oracle Database グローバリゼーション・サポート・ガイド』
データベースでサポートする必要のあるタイム・ゾーンを検討します。 Oracle Database では、2 つのタイム・ゾーン・ファイルのいずれかが、有効なタイム・ゾーンのソースとして使用されます。デフォルトのタイム・ゾーン・ファイルは timezonelrg.dat です。このファイルには、別のタイム・ゾーン・ファイル timezone.dat より多くのタイム・ゾーンが含まれています。	2-22 ページ「データベースのタイム・ゾーン・ファイルの指定」
データベースの標準ブロック・サイズを選択します。このサイズはデータベースの作成時に DB_BLOCK_SIZE 初期化パラメータによって指定しますが、データベースの作成後は変更できません。 標準ブロック・サイズは、SYSTEM 表領域およびその他の大部分の表領域で使用されます。また、表領域の作成時に、最大 4 つの非標準ブロック・サイズを指定できます。	2-28 ページ「データベース・ブロック・サイズの指定」
SYSAUX 表領域の適切な初期サイズを決定します。	2-17 ページ「SYSAUX 表領域の概要」
SYSTEM 以外のユーザーに対してはデフォルトの表領域を使用し、データベース・オブジェクトが SYSTEM 表領域に誤って保存されないようにします。	2-19 ページ「デフォルト永続表領域の作成」
UNDO データを管理するために、UNDO 表領域を使用します。	第 14 章「UNDO の管理」

表 2-1 データベース計画タスク（続き）

処理	追加情報
バックアップおよびリカバリ計画を作成し、データベースを障害から保護します。重要な点として、多重化による制御ファイルの保護、適切なバックアップ・モードの選択、およびオンライン REDO ログとアーカイブ REDO ログの管理があげられます。	第 10 章「REDO ログの管理」 第 11 章「アーカイブ REDO ログの管理」 第 9 章「制御ファイルの管理」 『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』
インスタンスの起動と停止、データベースのマウントとオープンの原理およびオプションについて理解します。	第 3 章「起動と停止」

作成の前提条件

新しいデータベースを作成するには、次の前提条件を満たす必要があります。

- 必要な Oracle ソフトウェアがインストールされていること。これには、オペレーティング・システム固有の各環境変数の設定、およびソフトウェアとデータベース・ファイルのディレクトリ構造の設定が含まれます。
- Oracle Database インスタンスを起動するために十分なメモリーが使用可能であること。
- Oracle Database を実行するコンピュータ上に、設計したデータベースのための十分なディスク記憶域が使用可能であること。

各前提条件については、オペレーティング・システム固有の『Oracle Database インストレーション・ガイド』を参照してください。Oracle Universal Installer を使用すると、表示される手順に従ってインストールでき、環境変数、ディレクトリ構造および認可の設定に関するヘルプが表示されます。

DBCA を使用したデータベースの作成

Database Configuration Assistant（DBCA）はほとんど自動化されており、DBCA が完了するとデータベースが使用可能状態になるので、この方法でデータベースを作成することをお薦めします。選択したインストール・タイプによっては、Oracle Universal Installer（OUI）から DBCA を起動できます。また、Oracle Database をインストール後は、スタンドアロン・ツールとして、いつでも DBCA を起動できます。

DBCA は、対話型モードまたは非対話型（サイレント）モードで実行できます。対話型モードには、データベースを作成して構成するためのグラフィカル・インタフェースおよびガイド付きワークフローが用意されています。非対話型（サイレント）モードでは、データベース作成スクリプトを記述できます。DBCA を非対話型（サイレント）モードで実行するには、コマンドライン引数またはレスポンス・ファイル、あるいはその両方を指定します。

対話型 DBCA を使用したデータベースの作成

対話型 DBCA を使用したデータベースの作成方法の詳細は、『Oracle Database 2 日でデータベース管理者』を参照してください。

非対話型（サイレント）DBCA を使用したデータベースの作成

非対話型（サイレント）モードの DBCA の使用方法の詳細は、使用しているプラットフォームのインストレーション・ガイドの付録 A を参照してください。

次の例では、DBCA にコマンドライン引数を渡してデータベースを作成しています。

```
dbca -silent -createDatabase -templateName General_Purpose.dbc
  -gdbname ora11g -sid ora11g -responseFile NO_VALUE -characterSet AL32UTF8
  -memoryPercentage 30 -emConfiguration LOCAL

Enter SYSTEM user password:
password
Enter SYS user password:
password
Copying database files
1% complete
3% complete
...
```

完全なサイレント操作が実行されるように、stdout をファイルにリダイレクトできます。ただし、その場合はコマンドライン引数またはレスポンス・ファイルに管理アカウントのパスワードを入力する必要があります。

DBCA コマンドライン引数の概要のヘルプを表示するには、次のコマンドを入力します。

```
dbca -help
```

デフォルトなど、引数の詳細は、配布媒体に同梱されているレスポンス・ファイル・テンプレートを参照してください。このファイルの名前と場所は、使用しているプラットフォームのインストレーション・ガイドの付録 A に記載されています。

CREATE DATABASE 文を使用したデータベースの作成

手動でデータベースを作成する場合は、CREATE DATBASE SQL 文を使用します。CREATE DATABASE 文を使用する場合は、実行可能なデータベースを作成する前に他の処理が必要です。この処理には、データ・ディクショナリ表のビューの作成、標準 PL/SQL パッケージのインストールなどがあります。これらの処理は、準備されているスクリプトを実行して行います。

データベース作成用の既存のスクリプトがある場合は、Oracle Database の新機能を利用するために、これらのスクリプトの編集を検討してください。

この項の手順が適用できるのは、単一インスタンスのインストールの場合のみです。Oracle Real Application Clusters（RAC）データベースの作成手順の詳細は、使用しているプラットフォーム固有の Oracle Real Application Clusters インストレーション・ガイドを参照してください。

注意： 単一インスタンスとは、単一のホスト・コンピュータに Oracle インスタンスが 1 つしか存在できないという意味ではありません。実際には、複数の Oracle インスタンス（およびそれらに関連するデータベース）を単一のホスト・コンピュータで実行できます。**単一インスタンス・データベース**とは、1 つの Oracle インスタンスのみがアクセスするデータベースのことです。これに対して、Oracle RAC データベースには、複数のノード上の複数の Oracle インスタンスが同時にアクセスします。Oracle RAC の詳細は、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。

CREATE DATABASE 文を使用してデータベースを作成するには、次の手順を実行します。次の例では、mynewdb というデータベースを作成します。

> **ヒント：** Oracle の自動ストレージ管理（ASM）を使用してディスク記憶域を管理している場合は、これらの手順を実行する前に、ASM インスタンスを起動し、ディスク・グループを構成する必要があります。自動ストレージ管理の詳細は、『Oracle Database ストレージ管理者ガイド』を参照してください。

手順 1: インスタンス識別子（SID）の指定

インスタンスに対して一意の Oracle システム識別子（SID）を決定し、コマンド・ウィンドウを開いて、環境変数 ORACLE_SID を設定します。後続の手順では、このコマンド・ウィンドウを使用します。

ORACLE_SID は、後で作成して同じホスト・コンピュータで同時に実行する他の Oracle Database インスタンスと区別するために使用されます。ORACLE_SID は最大 12 文字で、文字と数字のみを使用できます。一部のプラットフォームでは、SID の大 / 小文字が区別されます。

> **注意：** SID はデータベース名と同じ設定にするのが一般的です。データベース名に指定できるのは最大 8 文字までです。詳細は、『Oracle Database リファレンス』の DB_NAME 初期化パラメータの説明を参照してください。

次の UNIX および Linux オペレーティング・システムの例では、手順 6: インスタンスへの接続で接続するインスタンスの SID を設定しています。

- Bourne、Bash または Korn シェルの場合

```
ORACLE_SID=mynewdb
export ORACLE_SID
```

- C シェルの場合

```
setenv ORACLE_SID mynewdb
```

次の例では、Windows オペレーティング・システム用の SID を設定しています。

```
set ORACLE_SID=mynewdb
```

> **関連項目：** Oracle インスタンスに関するバックグラウンド情報は、『Oracle Database 概要』を参照してください。

手順 2: 必要な環境変数が設定されていることの確認

プラットフォームによっては、環境変数を設定するか少なくとも正しく設定されていることを確認するまで、（手順 6: インスタンスへの接続で必要な）SQL*Plus を起動できない場合があります。

たとえば、ほとんどのプラットフォームでは、ORACLE_SID と ORACLE_HOME を設定する必要があります。さらに、PATH 変数に *ORACLE_HOME*/bin ディレクトリを含めるように設定することもお薦めします。UNIX および Linux プラットフォームでは、これらの環境変数を手動で設定する必要があります。Windows プラットフォームでは、Windows レジストリ内の ORACLE_HOME と ORACLE_SID に OUI が値を自動的に割り当てます。インストール時にデータベースを作成しなかった場合は、レジストリ内の ORACLE_SID を OUI が設定しないため、後でデータベースを作成するときに ORACLE_SID 環境変数を設定する必要があります。

手順 3: データベース管理者の認証方式の選択

データベースを作成するには、作成するユーザーが認証を受け、適切なシステム権限が付与されている必要があります。必要な権限を付与された管理者として認証されるには、次の方法があります。

- パスワード・ファイルによる方法
- オペレーティング・システム認証による方法

この手順では、認証方式を決定します。

パスワード・ファイルによる認証方法を使用する場合は、1-23 ページの「パスワード・ファイルの作成とメンテナンス」の説明に従ってパスワード・ファイルを作成します。オペレーティング・システム認証による方法を使用する場合は、必ずオペレーティング・システムの適切なユーザー・グループのメンバーになっているユーザー・アカウントでホスト・コンピュータにログインします。たとえば、UNIX および Linux プラットフォームでは、通常 dba ユーザー・グループを使用します。Windows プラットフォームでは、Oracle ソフトウェアをインストールするユーザーが、必要なユーザー・グループに自動的に配置されます。

関連項目：

- 1-14 ページ「DBA のセキュリティと権限の概要」
- パスワード・ファイルとオペレーティング・システム認証の詳細は、1-16 ページの「DBA の認証」を参照してください。

手順 4: 初期化パラメータ・ファイルの作成

Oracle インスタンスの起動時に、初期化パラメータ・ファイルが読み込まれます。このファイルにはテキスト・ファイルかバイナリ・ファイルを使用できます。テキスト・ファイルの場合は、テキスト・エディタで作成して変更できます。バイナリ・ファイルの場合は、データベースによって作成されて動的に変更されます。バイナリ・ファイル（こちらの使用を推奨）は、**サーバー・パラメータ・ファイル**と呼ばれます。ここで示す手順では、テキスト形式の初期化パラメータ・ファイルを作成します。後述の手順で、テキスト・ファイルからサーバー・パラメータ・ファイルを作成します。

テキスト形式の初期化パラメータ・ファイルを作成する方法の 1 つは、2-26 ページの「初期化パラメータ・ファイルのサンプル」に記載されているサンプルを編集する方法です。

操作を容易にするために、デフォルトのファイル名を使用して、Oracle Database のデフォルトの場所に初期化パラメータ・ファイルを配置します。こうすることにより、Oracle Database が初期化パラメータ・ファイルのデフォルトの場所を自動的に参照するため、データベースの起動時に STARTUP コマンドの PFILE 句を指定する必要がなくなります。

使用しているプラットフォームの初期化パラメータ・ファイルのデフォルトのファイル名や場所など、初期化パラメータおよび初期化パラメータ・ファイルの詳細は、2-24 ページの「初期化パラメータと初期化パラメータ・ファイルの概要」を参照してください。

関連項目： すべての初期化パラメータの詳細は、『Oracle Database リファレンス』を参照してください。

手順 5: （Windows の場合のみ）インスタンスの作成

Windows プラットフォームでは、インスタンスが存在しない場合には、インスタンスに接続する前に手動で作成する必要があります。ORADIM コマンドを実行すると、新しい Windows サービスが作成されて、Oracle インスタンスが作成されます。

インスタンスを作成する手順は、次のとおりです。

- Windows のコマンド・プロンプトで、次のコマンドを入力します。

  ```
  oradim -NEW -SID sid -STARTMODE MANUAL -PFILE pfile
  ```

  sid は対象の SID（mynewdb など）、*pfile* はテキスト形式の初期化パラメータ・ファイルへの完全パスです。このコマンドでは、インスタンスは作成されますが起動されません。

  警告： この時点では、-STARTMODE 引数を AUTO に設定しないでください。そのように設定すると新しいインスタンスが起動され、まだ存在しないデータベースをマウントしようとします。必要な場合は、手順 14 でこのパラメータを AUTO に変更できます。

ORADIM コマンドの詳細は、『Oracle Database プラットフォーム・ガイド for Microsoft Windows』の Oracle データベース・インスタンスの管理に ORADIM を使用する方法に関する項を参照してください。

手順 6: インスタンスへの接続

SQL*Plus を起動して、SYSDBA システム権限で Oracle Database インスタンスに接続します。

- パスワード・ファイルによる認証を使用するには、次のコマンドを入力してプロンプトが表示された後、SYS パスワードを入力します。

  ```
  $ sqlplus /nolog
  SQL> CONNECT SYS AS SYSDBA
  ```

- オペレーティング・システム認証を使用するには、次のコマンドを入力します。

  ```
  $ sqlplus /nolog
  SQL> CONNECT / AS SYSDBA
  ```

SQL*Plus が次のメッセージを表示します。

```
Connected to an idle instance.
```

注意： SQL*Plus が次のようなメッセージを表示する場合があります。

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options
```

その場合には、インスタンスがすでに起動されています。間違ったインスタンスに接続している可能性があります。EXIT コマンドを使用して SQL*Plus を終了し、ORACLE_SID が正しく設定されていることを確認して、この手順を繰り返してください。

手順 7: サーバー・パラメータ・ファイルの作成

サーバー・パラメータ・ファイルを使用すると、ALTER SYSTEM コマンドを使用して初期化パラメータを変更でき、変更内容はデータベースを停止して起動した後も持続します。サーバー・パラメータ・ファイルは、編集済のテキスト形式の初期化ファイルから作成します。

次の SQL*Plus コマンドでは、テキスト形式の初期化パラメータ・ファイル（PFILE）がデフォルトの場所のデフォルトのファイル名で読み込まれ、テキスト形式の初期化パラメータ・ファイルからサーバー・パラメータ・ファイル（SPFILE）が作成されて、SPFILE がデフォルトの SPFILE 名でデフォルトの場所に書き込まれます。

```
CREATE SPFILE FROM PFILE;
```

デフォルトのファイル名と場所を使用していない場合は、PFILE と SPFILE の両方のファイル名とパスワードを指定することもできます。

ヒント： サーバー・パラメータ・ファイルを有効にするには、データベースを再起動する必要があります。

注意： この時点でのサーバー・パラメータ・ファイルの作成はオプションですが、作成することをお薦めします。サーバー・パラメータ・ファイルを作成しないと、インスタンスが起動されるたびにテキスト形式の初期化パラメータ・ファイルが引き続き読み込まれます。

重要: Oracle Managed Files を使用しており、初期化パラメータ・ファイルに CONTROL_FILES パラメータが指定されていない場合には、CREATE DATABASE 文でデータベースの制御ファイルが作成されるときに制御ファイルの場所と名前を保存できるように、この時点でサーバー・パラメータ・ファイルを作成しておく必要があります。詳細は、2-20 ページの「データベース作成時の Oracle Managed Files の作成」を参照してください。

関連項目：

- 2-31 ページ「サーバー・パラメータ・ファイルを使用した初期化パラメータの管理」
- CREATE SPFILE コマンドの詳細は、『Oracle Database SQL リファレンス』を参照してください。

手順 8: インスタンスの起動

データベースをマウントせずにインスタンスを起動します。通常、この方法で起動するのはデータベースの作成時またはメンテナンス時のみです。STARTUP コマンドで、NOMOUNT 句を指定します。この例では、初期化パラメータ・ファイルまたはサーバー・パラメータ・ファイルがデフォルトの場所に配置されているため、PFILE 句の指定は不要です。

```
STARTUP NOMOUNT
```

この時点で、インスタンス・メモリーが割り当てられて、インスタンスのプロセスが起動されます。データベース自体はまだ存在しません。

関連項目：

- Oracle インスタンスの概要については、『Oracle Database 概要』を参照してください。
- 2-31 ページ「サーバー・パラメータ・ファイルを使用した初期化パラメータの管理」
- STARTUP コマンドの使用方法は、第 3 章「起動と停止」を参照してください。

手順 9: CREATE DATABASE 文の発行

新しいデータベースを作成するには、CREATE DATABASE 文を使用します。

例 1

次の文では、データベース mynewdb を作成します。このデータベース名は、初期化パラメータ・ファイルの DB_NAME パラメータと同じにする必要があります。たとえば、次のような場合を考えてみます。

- 初期化パラメータ・ファイルの CONTROL_FILES パラメータには、制御ファイルの数と場所が指定されています。
- ディレクトリ /u01/app/oracle/oradata/mynewdb が存在します。

```
CREATE DATABASE mynewdb
   USER SYS IDENTIFIED BY sys_password
   USER SYSTEM IDENTIFIED BY system_password
   LOGFILE GROUP 1 ('/u01/app/oracle/oradata/mynewdb/redo01.log') SIZE 100M,
           GROUP 2 ('/u01/app/oracle/oradata/mynewdb/redo02.log') SIZE 100M,
           GROUP 3 ('/u01/app/oracle/oradata/mynewdb/redo03.log') SIZE 100M
   MAXLOGFILES 5
   MAXLOGMEMBERS 5
   MAXLOGHISTORY 1
   MAXDATAFILES 100
   CHARACTER SET US7ASCII
   NATIONAL CHARACTER SET AL16UTF16
   EXTENT MANAGEMENT LOCAL
   DATAFILE '/u01/app/oracle/oradata/mynewdb/system01.dbf' SIZE 325M REUSE
   SYSAUX DATAFILE '/u01/app/oracle/oradata/mynewdb/sysaux01.dbf' SIZE 325M REUSE
   DEFAULT TABLESPACE users
      DATAFILE '/u01/app/oracle/oradata/mynewdb/users01.dbf'
      SIZE 500M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED
   DEFAULT TEMPORARY TABLESPACE tempts1
      TEMPFILE '/u01/app/oracle/oradata/mynewdb/temp01.dbf'
      SIZE 20M REUSE
   UNDO TABLESPACE undotbs
      DATAFILE '/u01/app/oracle/oradata/mynewdb/undotbs01.dbf'
      SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

次の特性を持つデータベースが作成されます。

- データベースには mynewdb という名前が付けられます。グローバル・データベース名は mynewdb.us.oracle.com になります。ドメイン部分（us.oracle.com）は初期化ファイルから取得されます。2-26 ページの「グローバル・データベース名の決定」を参照してください。
- CONTROL_FILES 初期化パラメータ（データベース作成前に初期化パラメータ・ファイルに設定）で指定された 3 つの制御ファイルが作成されます。2-26 ページの「初期化パラメータ・ファイルのサンプル」および 2-28 ページの「制御ファイルの指定」を参照してください。
- ユーザー・アカウント SYS と SYSTEM のパスワードが指定した値に設定されます。リリース 11*g* からは、パスワードの大 / 小文字が区別されます。SYS および SYSTEM のパスワードを指定する 2 つの句は、このリリースの Oracle Database ではオプションです。ただし、指定する場合は、両方の句を指定する必要があります。これらの句の使用方法の詳細は、2-16 ページの「データベースの保護 : ユーザー SYS および SYSTEM のパスワードの指定」を参照してください。
- 新しいデータベースには、LOGFILE 句で指定された 3 つの REDO ログ・ファイルがあります。MAXLOGFILES、MAXLOGMEMBERS および MAXLOGHISTORY は、REDO ログの制限を定義します。第 10 章「REDO ログの管理」を参照してください。
- MAXDATAFILES によって、このデータベースでオープンできるデータファイルの最大数が指定されます。この数は、制御ファイルの初期サイズに影響を及ぼします。

注意： データベースの作成時に、制限をいくつか設定できます。これらの制限には、オペレーティング・システムの制限によって制限され、その影響を受けるものがあります。たとえば、MAXDATAFILES を設定すると、Oracle Database は、初期のデータベースにデータファイルが 1 つしかなくても、制御ファイルに MAXDATAFILES 個のファイル名を格納できるだけの領域を割り当てます。ただし、制御ファイルの最大サイズは制限されており、オペレーティング・システムによって異なるので、CREATE DATABASE 文のパラメータをすべて理論的な最大値で設定できるとはかぎりません。

データベース作成時に制限を設定する方法の詳細は、『Oracle Database SQL リファレンス』および使用しているオペレーティング・システム固有の Oracle マニュアルを参照してください。

- このデータベースにデータを格納する際は、US7ASCII キャラクタ・セットが使用されます。
- NATIONAL CHARACTER SET として AL16UTF16 キャラクタ・セットが指定されます。このキャラクタ・セットは、特に NCHAR、NCLOB または NVARCHAR2 として定義された列にデータを格納する際に使用されます。
- DATAFILE 句で指定したとおりに、SYSTEM 表領域（オペレーティング・システム・ファイル /u01/app/oracle/oradata/mynewdb/system01.dbf からなる）が作成されます。指定した名前のファイルがすでに存在する場合は上書きされます。
- SYSTEM 表領域はローカル管理の表領域として作成されます。2-17 ページの「ローカル管理の SYSTEM 表領域の作成」を参照してください。
- SYSAUX DATAFILE 句で指定したとおりに、SYSAUX 表領域（オペレーティング・システム・ファイル /u01/oracle/oradata/mynewdb/sysaux01.dbf からなる）が作成されます。2-17 ページの「SYSAUX 表領域の概要」を参照してください。
- DEFAULT TABLESPACE 句により、指定した名前で、このデータベース用のデフォルト永続表領域が作成されます。
- DEFAULT TEMPORARY TABLESPACE 句により、指定した名前で、このデータベース用のデフォルト一時表領域が作成されます。2-19 ページの「デフォルト一時表領域の作成」を参照してください。
- UNDO TABLESPACE 句によって、UNDO 表領域が指定の名前で作成されます。初期化パラメータ・ファイルで UNDO_MANAGEMENT=AUTO を指定した場合は、このデータベース用の UNDO データがこの UNDO 表領域に格納されます。このパラメータを省略すると、デフォルトでは AUTO になります。2-19 ページの「自動 UNDO 管理の使用 : UNDO 表領域の作成」を参照してください。
- この CREATE DATABASE 文では ARCHIVELOG 句が指定されていないため、初期状態では REDO ログ・ファイルはアーカイブされません。これはデータベース作成時の慣例です。後で ALTER DATABASE 文を使用して、ARCHIVELOG モードに切り替えることができます。mynewdb のアーカイブに関連する初期化パラメータ・ファイル内の初期化パラメータは、LOG_ARCHIVE_DEST_1 および LOG_ARCHIVE_FORMAT です。第 11 章「アーカイブ REDO ログの管理」を参照してください。

ヒント：

- CREATE DATABASE 文で使用しているディレクトリがすべて存在することを確認します。CREATE DATABASE 文ではディレクトリは作成されません。
- Oracle Managed Files を使用していない場合は、すべての TABLESPACE 句に DATAFILE 句または TEMPFILE 句を指定する必要があります。
- データベースの作成に失敗した場合は、アラート・ログを参照すると、失敗の原因とその対処措置を判別できます。8-19 ページの「アラート・ログの表示」を参照してください。プロセス番号を含むエラー・メッセージが表示される場合は、そのプロセスのトレース・ファイルを調べます。そのプロセス番号がトレース・ファイル名に含まれているトレース・ファイルを探します。詳細は、8-20 ページの「トレース・ファイルの検索」を参照してください。
- CREATE DATABASE 文を失敗後に再発行する場合は、最初にインスタンスを停止してから、前の CREATE DATABASE 文で作成されたファイルを削除する必要があります。

例 2

この例では、Oracle Managed Files を使用したデータベースの作成例を示します。この方法では、CREATE DATABASE 文をより簡単にできます。Oracle Managed Files を使用するには、初期化パラメータ DB_CREATE_FILE_DEST を設定する必要があります。このパラメータには、データベースで作成されて自動的に名前が付けられる様々なデータベース・ファイル用のベース・ディレクトリを定義します。次の文は、初期化パラメータ・ファイルにこのパラメータを設定する例を示しています。

```
DB_CREATE_FILE_DEST='/u01/app/oracle/oradata'
```

Oracle Managed Files および次の CREATE DATABASE 文を使用すると、SYSTEM 表領域と SYSAUX 表領域および文に指定されている追加の表領域がデータベースによって作成され、すべてのデータファイル、制御ファイルおよび REDO ログ・ファイルのデフォルト・サイズとプロパティが選択されます。この方法で設定されるこれらのプロパティおよび他のデフォルト・データベース・プロパティは本番環境には適さない場合があるため、生成された設定を調べて必要に応じて変更することをお薦めします。

```
CREATE DATABASE mynewdb
USER SYS IDENTIFIED BY sys_password
USER SYSTEM IDENTIFIED BY system_password
EXTENT MANAGEMENT LOCAL
DEFAULT TEMPORARY TABLESPACE temp
UNDO TABLESPACE undotbs1
DEFAULT TABLESPACE users;
```

ヒント： CREATE DATABASE 文が失敗する場合で手順 7 を完了していない場合は、このインスタンス用の既存のサーバー・パラメータ・ファイル（SPFILE）が予期しない方法で初期化パラメータを設定していないことを確認します。たとえば、すべての制御ファイルへの完全パスが SPFILE に設定されている場合は、それらの制御ファイルが存在しないと、CREATE DATABASE 文が失敗します。不必要な SPFILE を削除した後は、必ずインスタンスを停止して（STARTUP NOMOUNT を指定して）再起動します。詳細は、2-31 ページの「サーバー・パラメータ・ファイルを使用した初期化パラメータの管理」を参照してください。

関連項目：

- 2-16 ページ「CREATE DATABASE 文の句の指定」
- 2-20 ページ「データベース作成時の Oracle Managed Files の作成」
- 第 15 章「Oracle Managed Files の使用」
- CREATE DATABASE 文で指定できる句およびパラメータ値の詳細は、『Oracle Database SQL リファレンス』を参照してください。

手順 10: 追加の表領域の作成

データベースを稼働させるには、アプリケーション・データ用に表領域を追加作成する必要があります。次のサンプル・スクリプトは、追加の表領域を作成します。

```
CREATE TABLESPACE apps_tbs LOGGING
     DATAFILE '/u01/app/oracle/oradata/mynewdb/apps01.dbf'
     SIZE 500M REUSE AUTOEXTEND ON NEXT  1280K MAXSIZE UNLIMITED
     EXTENT MANAGEMENT LOCAL;
-- create a tablespace for indexes, separate from user tablespace (optional)
CREATE TABLESPACE indx_tbs LOGGING
     DATAFILE '/u01/app/oracle/oradata/mynewdb/indx01.dbf'
     SIZE 100M REUSE AUTOEXTEND ON NEXT  1280K MAXSIZE UNLIMITED
     EXTENT MANAGEMENT LOCAL;
```

表領域の作成方法の詳細は、第 12 章「表領域の管理」を参照してください。

手順 11: スクリプトの実行によるデータ・ディクショナリ・ビューの作成

データ・ディクショナリ・ビュー、シノニムおよび PL/SQL パッケージの作成および SQL*Plus の適切な稼働に必要なスクリプトを実行します。

```
@?/rdbms/admin/catalog.sql
@?/rdbms/admin/catproc.sql
@?/sqlplus/admin/pupbld.sql
EXIT
```

アットマーク（@）は SQL*Plus スクリプトを実行するコマンドの省略記述です。疑問符（?）は Oracle ホーム・ディレクトリを表す SQL*Plus の変数です。次の表に、スクリプトの説明を示します。

スクリプト	説明
CATALOG.SQL	データ・ディクショナリ表のビュー、動的パフォーマンス・ビューおよび多くのビューに対するパブリック・シノニムを作成します。シノニムに PUBLIC アクセスを付与します。
CATPROC.SQL	PL/SQL に必要なスクリプトや PL/SQL とともに使用されるスクリプトをすべて実行します。
PUPBLD.SQL	SQL*Plus に必要です。SQL*Plus がユーザーのコマンドを使用禁止にできるようにします。

手順 12: スクリプトの実行による追加オプションのインストール（オプション）

必要に応じて、他のスクリプトも実行できます。実行するスクリプトは、使用またはインストール対象として選択する機能とオプションによって異なります。使用可能なスクリプトは、『Oracle Database リファレンス』を参照してください。

このデータベースとともに稼働する他の Oracle 製品をインストールする場合は、それらの製品のインストール方法を参照してください。製品によっては、追加のデータ・ディクショナリ表を作成する必要があります。通常は、これらの表を作成して、データベースのデータ・ディクショナリにロードするためのコマンド・ファイルが提供されます。

インストールを計画している製品のインストールと管理の方法は、製品固有の Oracle マニュアルを参照してください。

手順 13: データベースのバックアップ

メディア障害が発生した場合にリカバリするための完全なファイル・セットが確実に存在するように、データベースの全体バックアップを作成してください。データベースのバックアップの詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

手順 14: （オプション）インスタンスの自動起動の有効化

ホスト・コンピュータの再起動時に、Oracle インスタンスが自動的に起動されるように構成する必要がある場合があります。手順については、使用しているオペレーティング・システムのマニュアルを参照してください。たとえば Windows では、次のコマンドを使用して、コンピュータの再起動時にデータベース・サービスがインスタンスを起動するように構成します。

```
ORADIM -EDIT -SID sid -STARTMODE AUTO -SRVCSTART SYSTEM [-SPFILE]
```

自動再起動時にインスタンスが SPFILE を読み込むようにする場合は、`-SPFILE` 引数を使用する必要があります。

`ORADIM` コマンドの詳細は、『Oracle Database プラットフォーム・ガイド for Microsoft Windows』の Oracle データベース・インスタンスの管理に ORADIM を使用する方法に関する項を参照してください。

CREATE DATABASE 文の句の指定

CREATE DATABASE 文を実行すると、Oracle Database はいくつかの操作を実行します。実際の操作は、CREATE DATABASE 文で指定した句、および初期化パラメータの設定に応じて実行されます。Oracle Database が最低限実行する操作は、次のとおりです。

- データベース用のデータファイルの作成
- データベース用の制御ファイルの作成
- データベース用の REDO ログ・ファイルの作成と ARCHIVELOG モードの設定
- SYSTEM 表領域の作成
- SYSAUX 表領域の作成
- データ・ディクショナリの作成
- データベースへのデータの格納に使用するキャラクタ・セットの設定
- データベース・タイム・ゾーンの設定
- データベースのマウントとオープン

この項では、CREATE DATABASE 文の複数の句について説明します。これには、2-11 ページの「手順 9: CREATE DATABASE 文の発行」で説明した句と、その他の句が含まれます。ここで説明する CREATE DATABASE の句の多くは、データベースの作成と管理を簡素化するために使用できます。

この項の内容は、次のとおりです。

- データベースの保護 : ユーザー SYS および SYSTEM のパスワードの指定
- ローカル管理の SYSTEM 表領域の作成
- SYSAUX 表領域の概要
- 自動 UNDO 管理の使用 : UNDO 表領域の作成
- デフォルト一時表領域の作成
- データベース作成時の Oracle Managed Files の作成
- データベース作成時の大型ファイル表領域のサポート
- データベースのタイム・ゾーンとタイム・ゾーン・ファイルの指定
- FORCE LOGGING モードの指定

データベースの保護 : ユーザー SYS および SYSTEM のパスワードの指定

CREATE DATABASE 文でユーザー SYS および SYSTEM のパスワード指定に使用する句は、次のとおりです。

- USER SYS IDENTIFIED BY *password*
- USER SYSTEM IDENTIFIED BY *password*

これらの句を省略すると、これらのユーザーにはそれぞれデフォルトのパスワード change_on_install および manager が割り当てられます。デフォルトのパスワードが使用された場合は、そのことを示すレコードがアラート・ログに書き込まれます。データベースを保護するには、データベース作成直後に ALTER USER 文を使用して、これらのパスワードを変更する必要があります。

これらの句は、このリリースの Oracle Database ではオプションですが、指定することをお薦めします。デフォルトのパスワードは一般に知られており、後で変更するのを怠ると、データベースを悪意のユーザーによる攻撃にさらすことになります。

パスワードを選択するときには、リリース 11*g* 以降ではパスワードの大 / 小文字が区別されることに注意してくださいまた、使用中のデータベースにパスワード形式の要件が設定されてい

る場合もあります。詳細は、『Oracle Database セキュリティ・ガイド』の Oracle データベースによるパスワードの複雑度の検証方法に関する項を参照してください。

関連項目： 2-44 ページ「セキュリティに関する考慮点」

ローカル管理の SYSTEM 表領域の作成

CREATE DATABASE 文に EXTENT MANAGEMENT LOCAL 句を指定すると、ローカル管理の SYSTEM 表領域を作成できます。この文を正常に実行するには、COMPATIBLE 初期化パラメータを 10.0.0 以上に設定する必要があります。EXTENT MANAGEMENT LOCAL 句を指定しない場合は、デフォルトでディクショナリ管理の SYSTEM 表領域が作成されます。ディクショナリ管理表領域は非推奨です。

ローカル管理の SYSTEM 表領域を指定してデータベースを作成する場合に Oracle Managed Files を使用していない場合は、次の条件が満たされているかどうかを確認してください。

- DEFAULT TEMPORARY TABLESPACE 句を CREATE DATABASE 文に指定
- UNDO TABLESPACE 句を CREATE DATABASE 文に指定

関連項目：

- SYSTEM 表領域に対して EXTENT MANAGEMENT LOCAL を指定する場合の DEFAULT TEMPORARY TABLESPACE 句および UNDO TABLESPACE 句の使用に関する詳細は、『Oracle Database SQL リファレンス』を参照してください。
- 12-4 ページ「ローカル管理表領域」
- 12-29 ページ「ローカル管理表領域への SYSTEM 表領域の移行」

SYSAUX 表領域の概要

SYSAUX 表領域は、データベース作成時に必ず作成されます。この SYSAUX 表領域は、SYSTEM 表領域の補助的な表領域として機能します。この表領域は、以前に固有の表領域を必要としていた Oracle Database の多くの機能と製品に対するデフォルト表領域であるため、データベースで必要な表領域の数が削減されます。また、SYSTEM 表領域の負荷も軽減されます。

SYSAUX 表領域に対して指定できるのは、データファイルの属性のみで、CREATE DATABASE 文で SYSAUX DATAFILE 句を使用して指定します。SYSAUX 表領域の必須属性は Oracle Database によって設定されます。次の必須属性があります。

- PERMANENT
- READ WRITE
- EXTENT MANAGMENT LOCAL
- SEGMENT SPACE MANAGMENT AUTO

これらの属性は、ALTER TABLESPACE 文で変更できません。変更しようとするとエラーが発生します。また、SYSAUX 表領域の削除や名前の変更はできません。

SYSAUX 表領域のサイズは、SYSAUX を使用するデータベース・コンポーネントのサイズにより決定します。SYSAUX 占有データを使用する全コンポーネントのリストは、表 2-2 を参照してください。これらのコンポーネントの初期サイズに基づいて、SYSAUX 表領域には、データベース作成時に最低限 240MB 必要です。SYSAUX 表領域の領域要件は、データベースが完全にデプロイされた後、その使用状況やワークロードによって増加します。変動する基準に基づいて SYSAUX 表領域の領域消費を管理する方法の詳細は、12-25 ページの「SYSAUX 表領域の管理」を参照してください。

SYSTEM 表領域に対して DATAFILE 句を指定している場合は、SYSAUX DATAFILE 句も指定する必要があります。指定しない場合は、CREATE DATABASE 文が失敗します。この要件は、Oracle Managed Files 機能が使用可能な場合は適用されません（2-20 ページの「データベース作成時の Oracle Managed Files の作成」を参照してください）。

SYSAUX 表領域のセキュリティ属性は SYSTEM 表領域と同じです。

注意： このマニュアルでは、データベース作成時の SYSAUX データベースの作成について説明しています。SYSAUX 表領域を必要としない Oracle Database のリリースからアップグレードするときは、アップグレード・プロセス時に SYSAUX 表領域を作成する必要があります。この操作については、『Oracle Database アップグレード・ガイド』を参照してください。

表 2-2 に、インストール時のデフォルト表領域として SYSAUX 表領域を使用するコンポーネント、および以前のリリースでそのコンポーネントが格納されていた表領域を示します。

表 2-2 データベース・コンポーネントと SYSAUX 表領域

SYSAUX を使用するコンポーネント	以前のリリースでの格納先表領域
アナリティック・ワークスペース・オブジェクト表	SYSTEM
Enterprise Manager リポジトリ	OEM_REPOSITORY
LogMiner	SYSTEM
ロジカル・スタンバイ	SYSTEM
OLAP API 履歴表	CWMLITE
Oracle Data Mining	ODM
Oracle Spatial	SYSTEM
Oracle Streams	SYSTEM
Oracle Text	DRSYS
Oracle Ultra Search	DRSYS
Oracle *inter*Media ORDPLUGINS のコンポーネント	SYSTEM
Oracle *inter*Media ORDSYS のコンポーネント	SYSTEM
Oracle *inter*Media SI_INFORMTN_SCHEMA のコンポーネント	SYSTEM
サーバーの管理性のコンポーネント	
Statspack リポジトリ	ユーザー定義
Oracle Scheduler	
Workspace Manager	SYSTEM

関連項目： SYSAUX 表領域の管理方法の詳細は、12-25 ページの「SYSAUX 表領域の管理」を参照してください。

自動 UNDO 管理の使用 : UNDO 表領域の作成

自動 UNDO 管理では UNDO 表領域が使用されます。自動 UNDO 管理を使用可能にするには、初期化パラメータ・ファイルで UNDO_MANAGEMENT 初期化パラメータを AUTO に設定します。このパラメータを省略すると、デフォルトでデータベースが自動 UNDO 管理になります。このモードでは、UNDO データが UNDO 表領域に格納され、Oracle Database によって管理されます。UNDO 表領域を定義して名前を付ける必要がある場合は、データベース作成時に CREATE DATABASE 文に UNDO TABLESPACE 句を指定する必要があります。この句を省略して自動 UNDO 管理を使用可能にすると、デフォルトの UNDO 表領域 SYS_UNDOTBS が作成されます。

関連項目：

- 2-30 ページ「UNDO 領域管理方法の指定」
- UNDO 表領域の作成と使用の詳細は、第 14 章「UNDO の管理」を参照してください。

デフォルト永続表領域の作成

CREATE DATABASE 文で DEFAULT TABLESPACE 句を指定すると、データベースのデフォルト永続表領域が作成されます。この表領域は、Oracle Database によって、別の永続表領域が明示的に指定されていない SYSTEM 以外のユーザーに対して割り当てられます。この句を指定しない場合、SYSTEM 以外のユーザーに対するデフォルトの永続表領域は SYSTEM 表領域です。デフォルトの永続表領域を作成することをお薦めします。

関連項目： CREATE DATABASE および ALTER DATABASE の DEFAULT TABLESPACE 句の構文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

デフォルト一時表領域の作成

CREATE DATABASE 文で DEFAULT TEMPORARY TABLESPACE 句を指定すると、データベースにデフォルトの一時表領域が作成されます。Oracle Database は、この表領域を、一時表領域が明示的に割り当てられていないユーザーに対して一時表領域として割り当てます。

一時表領域または表領域グループは、CREATE USER 文で明示的にユーザーに割り当てることができます。ただし、明示的に割り当てない場合は、データベースにデフォルトの一時表領域が指定されていないと、これらのユーザーにはデフォルトで一時表領域として SYSTEM 表領域が割り当てられます。一時データを SYSTEM 表領域に格納するのはよい方法ではなく、各ユーザーを個別に一時表領域に割り当てる方法は効率的ではありません。したがって、CREATE DATABASE の DEFAULT TEMPORARY TABLESPACE 句の使用をお薦めします。

注意： ローカル管理の SYSTEM 表領域を指定すると、その SYSTEM 表領域は一時表領域として使用できません。この場合は、デフォルトの一時表領域を作成する必要があります。この動作については、2-17 ページの「ローカル管理の SYSTEM 表領域の作成」を参照してください。

関連項目：

- CREATE DATABASE および ALTER DATABASE の DEFAULT TEMPORARY TABLESPACE 句の構文の詳細は、『Oracle Database SQL リファレンス』を参照してください。
- 一時表領域の作成と使用については、12-10 ページの「一時表領域」を参照してください。
- 一時表領域グループの作成と使用については、12-12 ページの「複数の一時表領域 : 表領域グループの使用」を参照してください。

データベース作成時の Oracle Managed Files の作成

Oracle Managed Files 機能を使用すると、CREATE DATABASE 文に指定する句とパラメータの数を最小限に抑えることができます。この機能を使用するには、Oracle Database によってファイルが作成および管理されるディレクトリまたは Automatic Storage Management（ASM）ディスク・グループのいずれかを指定します。

初期化パラメータ・ファイルに DB_CREATE_FILE_DEST、DB_CREATE_ONLINE_LOG_DEST_*n* または DB_RECOVERY_FILE_DEST 初期化パラメータを設定すると、データベースの基礎となるオペレーティング・システム・ファイルを Oracle Database で作成および管理できます。指定した初期化パラメータおよび CREATE DATABASE 文で指定した句に応じて、次のデータベース構造のオペレーティング・システム・ファイルが Oracle Database によって自動的に作成および管理されます。

- 表領域とそのデータファイル
- 一時表領域とその一時ファイル
- 制御ファイル
- REDO ログ・ファイル
- アーカイブ REDO ログ・ファイル
- フラッシュバック・ログ
- ブロック・チェンジ・トラッキング・ファイル
- Recovery Manager によるバックアップ

> **関連項目：** フラッシュ・リカバリ領域を作成する初期化パラメータの設定方法については、2-27 ページの「フラッシュ・リカバリ領域の指定」を参照してください。

次の CREATE DATABASE 文で、Oracle Managed Files 機能の動作を示します。必要な初期化パラメータは指定されているとします。

```
CREATE DATABASE mynewdb
     USER SYS IDENTIFIED BY sys_password
     USER SYSTEM IDENTIFIED BY system_password
     EXTENT MANAGEMENT LOCAL
     UNDO TABLESPACE undotbs
     DEFAULT TEMPORARY TABLESPACE tempts1
     DEFAULT TABLESPACE users;
```

- SYSTEM 表領域はローカル管理の表領域として作成されます。EXTENT MANAGEMENT LOCAL 句を指定しないと、SYSTEM 表領域はディクショナリ管理として作成されます。この方法はお薦めしません。
- DATAFILE 句が指定されていないため、Oracle Managed Files の SYSTEM 表領域用のデータファイルが作成されます。
- LOGFILE 句が指定されていないため、Oracle が管理する REDO ログ・ファイルのグループが 2 つ作成されます。
- SYSAUX DATAFILE が指定されていないため、Oracle Managed Files の SYSAUX 表領域用のデータファイルが作成されます。
- UNDO TABLESPACE 句および DEFAULT TABLESPACE 句に DATAFILE 副次句が指定されていないため、これらの各表領域用に Oracle Managed Files のデータファイルが作成されます。
- DEFAULT TEMPORARY TABLESPACE 句に TEMPFILE 副次句が指定されていないため、Oracle Managed Files の一時ファイルが作成されます。
- 初期化パラメータ・ファイルに CONTROL_FILES 初期化パラメータが指定されていない場合は、Oracle Managed Files の制御ファイルも作成されます。

- サーバー・パラメータ・ファイル（2-31 ページの「サーバー・パラメータ・ファイルを使用した初期化パラメータの管理」を参照）を使用している場合は、適切な初期化パラメータが自動的に設定されます。

関連項目：

- Oracle Managed Files 機能と使用方法の詳細は、第 15 章「Oracle Managed Files の使用」を参照してください。
- 自動ストレージ管理の詳細は、『Oracle Database ストレージ管理者ガイド』を参照してください。

データベース作成時の大型ファイル表領域のサポート

Oracle Database では、**大型ファイル表領域**を作成できるようにすることで、表領域の管理を簡素化し、大規模なデータベースのサポートを可能にしています。大型ファイル表領域に含めることができるファイルは 1 つのみですが、そのファイルには最大 40 億ブロックまで設定できます。Oracle Database では、データファイルの最大数が制限されています（通常は 64000 ファイル）。したがって、大型ファイル表領域によって、Oracle Database の記憶域容量が大幅に増加します。

この項では、大型ファイル表領域のサポートを可能にする CREATE DATABASE 文の句について説明します。

関連項目： 大型ファイル表領域の詳細は、12-6 ページの「大型ファイル表領域」を参照してください。

デフォルトの表領域タイプの指定

CREATE DATABASE 文の SET DEFAULT...TABLESPACE 句によって、後続の CREATE TABLESPACE 文でこのデータベースに使用される表領域のデフォルト・タイプが決定します。SET DEFAULT BIGFILE TABLESPACE または SET DEFAULT SMALLFILE TABLESPACE を指定します。この句を省略すると、デフォルトでは、従来のタイプの Oracle Database 表領域である**小型ファイル表領域**が作成されます。小型ファイル表領域には最大 1022 ファイルを含めることができ、それぞれ 400 万ブロックまで設定できます。

大型ファイル表領域を使用すると、Oracle Managed Files の機能がさらに強化されます。これは、大型ファイル表領域によって、データファイルがユーザーに対して完全に透過的になるためです。ALTER TABLESPACE 文の SQL 構文は、基礎となるデータファイルではなく表領域で操作を実行できるように拡張されています。

表領域のデフォルト・タイプが大型ファイル表領域であることを指定するには、2-20 ページの「データベース作成時の Oracle Managed Files の作成」に示した CREATE DATABASE 文を次のように変更します。

```
CREATE DATABASE mynewdb
     USER SYS IDENTIFIED BY sys_password
     USER SYSTEM IDENTIFIED BY system_password
     SET DEFAULT BIGFILE TABLESPACE
     UNDO TABLESPACE undotbs
     DEFAULT TEMPORARY TABLESPACE tempts1;
```

デフォルトの表領域タイプをデータベース作成後に動的に変更するには、ALTER DATABASE 文の SET DEFAULT TABLESPACE 句を使用します。

```
ALTER DATABASE SET DEFAULT BIGFILE TABLESPACE;
```

データベースの現行のデフォルト一時表領域タイプを判断するには、DATABASE_PROPERTIES データ・ディクショナリ・ビューを次のように問い合せます。

```
SELECT PROPERTY_VALUE FROM DATABASE_PROPERTIES
   WHERE PROPERTY_NAME = 'DEFAULT_TBS_TYPE';
```

デフォルトの表領域タイプの上書き

SYSTEM および SYSAUX 表領域は、常にデフォルトの表領域タイプで作成されます。ただし、UNDO および DEFAULT TEMPORARY 表領域のデフォルトの表領域タイプは、CREATE DATABASE 操作時に、明示的に上書きできます。

たとえば、次のように指定すると、デフォルトの表領域タイプが小型ファイルのデータベースで、大型ファイルの UNDO 表領域を作成できます。

```
CREATE DATABASE mynewdb
...
     BIGFILE UNDO TABLESPACE undotbs
        DATAFILE '/u01/oracle/oradata/mynewdb/undotbs01.dbf'
        SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

次のように指定すると、デフォルトの表領域タイプが大型ファイルのデータベースで、小型ファイルの DEFAULT TEMPORARY 表領域を作成できます。

```
CREATE DATABASE mynewdb
   SET DEFAULT BIGFILE TABLSPACE
...
     SMALLFILE DEFAULT TEMPORARY TABLESPACE tempts1
        TEMPFILE '/u01/oracle/oradata/mynewdb/temp01.dbf'
        SIZE 20M REUSE
...
```

データベースのタイム・ゾーンとタイム・ゾーン・ファイルの指定

データベースのタイム・ゾーンおよびサポートするタイム・ゾーン・ファイルを指定できます。

データベースのタイム・ゾーンの設定

データベースの作成時にデータベースのタイム・ゾーンを設定するには、CREATE DATABASE 文で SET TIME_ZONE 句を使用します。データベースのタイム・ゾーンを設定しない場合は、デフォルトでサーバーのオペレーティング・システムのタイム・ゾーンに設定されます。

セッションのデータベース・タイム・ゾーンを変更するには、ALTER SESSION 文で SET TIME_ZONE 句を使用します。

> **関連項目：** データベース・タイム・ゾーンの設定の詳細は、『Oracle Database グローバリゼーション・サポート・ガイド』を参照してください。

データベースのタイム・ゾーン・ファイルの指定

Oracle ホーム・ディレクトリには、次の 2 つのタイム・ゾーン・ファイルがあります。デフォルトのタイム・ゾーン・ファイルは $ORACLE_HOME/oracore/zoneinfo/timezonelrg.dat です。定義数の少ないタイム・ゾーン・ファイルは $ORACLE_HOME/oracore/zoneinfo/timezone.dat にあります。

定義数の少ないタイム・ゾーン・ファイルをすでに使用していて、Oracle Database 11*g* 環境でも引き続き使用する場合、またはデフォルトのタイム・ゾーン・ファイルのかわりに定義数の少ないタイム・ゾーン・ファイルを使用する場合は、次のタスクを実行します。

1. データベースを停止します。
2. ORA_TZFILE 環境変数に、timezone.dat ファイルのフルパス名を設定します。
3. データベースを再起動します。

デフォルトのタイム・ゾーン・ファイルをすでに使用している場合、定義数の少ないタイム・ゾーン・ファイルに変更するのは実用的ではありません。データベースに、定義数の少ないタイム・ゾーン・ファイルには含まれていないタイム・ゾーンのデータが含まれている可能性があるためです。

情報を共有しているすべてのデータベースが同じタイム・ゾーン・データファイルを使用する必要があります。

タイム・ゾーン・ファイルには有効なタイム・ゾーン名が含まれています。タイム・ゾーンごとに次の情報が含まれています。

- 協定世界時（UTC）からのオフセット
- 夏時間への移行期間
- 標準時間および夏時間の略称

ファイル内で、データベースで使用されているタイム・ゾーン名を表示するには、次の問合せを実行します。

```
SELECT * FROM V$TIMEZONE_NAMES;
```

FORCE LOGGING モードの指定

一部のデータ定義言語文（CREATE TABLE など）では、NOLOGGING 句を使用できますが、ある種のデータベース操作ではデータベース REDO ログに REDO レコードが生成されません。NOLOGGING 句を設定すると、データベース・リカバリ・メカニズム外で容易にリカバリできる操作は高速化されますが、メディア・リカバリとスタンバイ・データベースに悪影響を与える可能性があります。

Oracle Database では、DDL 文に NOLOGGING が指定されていても、REDO レコードを強制的に書き込ませることができます。データベースでは、一時表領域と一時セグメントの REDO レコードは生成されないため、FORCE LOGGING はオブジェクトに影響を与えません。

> **関連項目：** NOLOGGING モードで実行できる操作の詳細は、『Oracle Database SQL リファレンス』を参照してください。

FORCE LOGGING 句の使用

データベースを FORCE LOGGING モードにするには、CREATE DATABASE 文で FORCE LOGGING 句を使用します。この句を指定しない場合、データベースは FORCE LOGGING モードになりません。

データベースの作成後に、ALTER DATABASE 文を使用してデータベースを FORCE LOGGING モードにします。この文は、ロギングなしの直接書込みがすべて完了するまで待機するため、完了までに長時間かかる可能性があります。

次の SQL 文を使用すると、FORCE LOGGING モードを取り消しできます。

```
ALTER DATABASE NO FORCE LOGGING;
```

データベースに対して FORCE LOGGING を指定するかどうかに関係なく、表領域レベルで FORCE LOGGING または NO FORCE LOGGING を選択的に指定できます。ただし、データベースの FORCE LOGGING モードが有効になっている場合は、表領域の設定より優先されます。データベースに対して有効になっていない場合は、個々の表領域の設定が実行されます。データベース全体を FORCE LOGGING モードにするか、または個々の表領域を FORCE LOGGING モードにすることをお薦めします。一度に両方の設定を行わないでください。

FORCE LOGGING モードは、データベースの永続属性です。つまり、データベースが停止され、再起動されても、同じロギング・モードのままです。ただし、制御ファイルを再作成すると、CREATE CONTROL FILE 文で FORCE LOGGING 句を指定しないかぎり、データベースは FORCE LOGGING モードで再起動しません。

> **関連項目：** 表領域の作成に FORCE LOGGING 句を使用する方法の詳細は、12-14 ページの「REDO レコードの書込みの制御」を参照してください。

FORCE LOGGING モードのパフォーマンスに関する考慮点

FORCE LOGGING モードは、ある程度のパフォーマンスの低下を伴います。主として完全メディア・リカバリを確実に行うために FORCE LOGGING を指定し、アクティブになっているスタンバイ・データベースがない場合は、次の点を考慮する必要があります。

- 発生すると思われるメディア障害の数
- ロギングなしの直接書込みをリカバリできない場合のダメージの重大度
- FORCE LOGGING モードによるパフォーマンスの低下は許容範囲内かどうか

データベースが NOARCHIVELOG モードで稼働している場合、通常は FORCE LOGGING モードにする利点はありません。NOARCHIVELOG モードではメディア・リカバリが不可能であるため、FORCE LOGGING と組み合せて使用する場合は、ほとんど利点がなく、パフォーマンスが低下する可能性があります。

初期化パラメータの指定

ここでは、新しいデータベースを作成する前に追加または編集できるいくつかの基本的な初期化パラメータについて説明します。この項の内容は、次のとおりです。

- 初期化パラメータと初期化パラメータ・ファイルの概要
- グローバル・データベース名の決定
- フラッシュ・リカバリ領域の指定
- 制御ファイルの指定
- データベース・ブロック・サイズの指定
- 最大プロセス数の指定
- DDL ロック・タイムアウトの指定
- UNDO 領域管理方法の指定
- COMPATIBLE 初期化パラメータの概要
- ライセンスに関するパラメータの設定

関連項目：

- デフォルト設定など、すべての初期化パラメータに関する説明は、『Oracle Database リファレンス』を参照してください。
- メモリー管理に関係する初期化パラメータの説明は、第 5 章「メモリーの管理」を参照してください。

初期化パラメータと初期化パラメータ・ファイルの概要

Oracle インスタンスの起動時に、初期化パラメータ・ファイルから初期化パラメータが読み込まれます。初期化パラメータのうち、初期化パラメータ・ファイル内に特に設定されていないものには、デフォルト値が適用されます。

初期化パラメータ・ファイルは、読取り専用のテキスト・ファイルまたは読取り / 書込みのバイナリ・ファイルです。バイナリ・ファイルは、**サーバー・パラメータ・ファイル**と呼ばれます。サーバー・パラメータ・ファイルを使用すると、ALTER SYSTEM コマンドを使用して初期化パラメータを変更でき、変更内容は停止して起動した後も持続します。また、Oracle Database による自己チューニングの基礎ともなります。このため、サーバー・パラメータ・ファイルを使用することをお薦めします。サーバー・パラメータ・ファイルは、編集済のテキスト形式の初期化ファイルから手動で作成するか、またはデータベースを作成するための Database Configuration Assistant（DBCA）を使用して自動的に作成できます。

サーバー・パラメータ・ファイルを手動で作成する前に、テキスト形式の初期化パラメータ・ファイルを使用してインスタンスを起動できます。起動時に、Oracle インスタンスは最初にデフォルトの場所でサーバー・パラメータ・ファイルを検索し、見つからない場合は、テキスト

形式の初期化パラメータ・ファイルを検索します。また、STARTUP コマンドの引数としてテキスト形式の初期化パラメータ・ファイルを指定すると、既存のサーバー・パラメータ・ファイルを上書きすることもできます。

テキスト形式の初期化パラメータ・ファイルのデフォルトのファイル名と場所は、次の表のとおりです。

プラットフォーム	デフォルト名	デフォルトの場所
UNIX と Linux	init*ORACLE_SID*.ora たとえば、mynewdb データベースの初期化パラメータ・ファイル名は次のようになります。 initmynewdb.ora	*ORACLE_HOME*/dbs
Windows	init*ORACLE_SID*.ora	*ORACLE_HOME*¥database

Oracle Database を初めて作成する場合は、変更するパラメータ値の数を最小限にとどめておくことをお薦めします。データベースと環境に慣れてから、多数の初期化パラメータを ALTER SYSTEM 文で動的にチューニングしてください。テキスト形式の初期化パラメータ・ファイルを使用している場合、現行のインスタンスについてのみ変更できます。永続的に変更するには、初期化パラメータ・ファイル内で手動で更新する必要があります。それ以外の場合は、次回データベースを停止して起動すると、変更内容が失われます。サーバー・パラメータ・ファイルを使用している場合、ALTER SYSTEM 文で行った初期化パラメータ・ファイルの変更内容は、停止して起動した後も持続します。

サーバー・パラメータ・ファイルの詳細は、2-31 ページの「サーバー・パラメータ・ファイルを使用した初期化パラメータの管理」を参照してください。STARTUP コマンドの詳細は、3-3 ページの「初期化パラメータおよび起動の概要」を参照してください。

テキスト形式の初期化パラメータ・ファイル

テキスト形式の初期化パラメータ・ファイル（PFILE）には、次のいずれかの書式の名前 / 値ペアを含める必要があります。

- 値を 1 つのみ受け入れるパラメータの場合

  ```
  parameter_name=value
  ```

- 1 つ以上の値を受け入れるパラメータの場合（CONTROL_FILES パラメータなど）

  ```
  parameter_name=(value[,value] ...)
  ```

文字列型のパラメータ値は、一重引用符（'）で囲む必要があります。ファイル名の大 / 小文字区別が有効となるのは、ホスト・オペレーティング・システムで有効な場合のみです。

複数の値を受け入れるパラメータの場合、アラート・ログから名前 / 値ペアを簡単にコピーして貼り付けられるように、複数行でパラメータを繰り返すことができます。この場合、各行には異なる値が含まれます。

```
control_files='/u01/app/oracle/oradata/orcl/control01.ctl'
control_files='/u01/app/oracle/oradata/orcl/control02.ctl'
control_files='/u01/app/oracle/oradata/orcl/control03.ctl'
```

複数の値を受け入れないパラメータを繰り返した場合、最後に指定した値のみが有効です。

関連項目：

- テキスト形式の初期化パラメータ・ファイルの内容と構文の詳細は、『Oracle Database リファレンス』を参照してください。
- 8-5 ページの「アラート・ログ」

初期化パラメータ・ファイルのサンプル

Oracle Database には通常、適切な値が設定されたテキスト形式の初期化パラメータのサンプルが用意されています。構成やオプション、およびデータベースのチューニング計画によっては、オラクル社が提供するこれらの初期化パラメータを編集し、他の初期化パラメータを追加することが可能です。

テキスト形式の初期化パラメータ・ファイルのサンプルは init.ora という名前で、ほとんどのプラットフォームの次の場所にあります。

ORACLE_HOME/dbs

サンプル・ファイルの内容は次のとおりです。

```
###########################################################################
# Example INIT.ORA file
#
# This file is provided by Oracle Corporation to help you start by providing
# a starting point to customize your RDBMS installation for your site.
#
# NOTE: The values that are used in this file are only intended to be used
# as a starting point. You may want to adjust/tune those values to your
# specific hardware and needs. You may also consider using Database
# Configuration Assistant tool (DBCA) to create INIT file and to size your
# initial set of tablespaces based on the user input.
###########################################################################

# Change '<ORACLE_BASE>' to point to the oracle base (the one you specify at
# install time)

db_name='ORCL'
memory_target=1G
processes = 150
audit_file_dest='<ORACLE_BASE>/admin/orcl/adump'
audit_trail ='db'
db_block_size=8192
db_domain=''
db_recovery_file_dest='<ORACLE_BASE>/flash_recovery_area'
db_recovery_file_dest_size=2G
diagnostic_dest='<ORACLE_BASE>'
dispatchers='(PROTOCOL=TCP) (SERVICE=ORCLXDB)'
open_cursors=300
remote_login_passwordfile='EXCLUSIVE'
undo_tablespace='UNDOTBS1'
# You may want to ensure that control files are created on separate physical
# devices
control_files = (ora_control1, ora_control2)
compatible ='11.1.0'
```

グローバル・データベース名の決定

グローバル・データベース名は、ユーザー指定のローカル・データベース名と、ネットワーク構造内でのデータベースの位置で構成されます。データベース名のローカル名コンポーネントは DB_NAME 初期化パラメータによって決定し、ネットワーク構造内のドメイン（論理的な位置）はオプションで指定できる DB_DOMAIN パラメータによって決定します。これら 2 つのパラメータの設定を組み合せて、ネットワーク内で一意となるデータベース名を形成する必要があります。

たとえば、test.us.acme.com というグローバル・データベース名を持つデータベースを作成するには、新しいパラメータ・ファイルのパラメータを次のように編集します。

```
DB_NAME = test
DB_DOMAIN = us.acme.com
```

ALTER DATABASE RENAME GLOBAL_NAME 文を使用すると、データベースの GLOBAL_NAME を変更できます。ただし、最初に DB_NAME および DB_DOMAIN 初期化パラメータを変更し、制御ファイルを再作成した後に、データベースを停止して再起動する必要があります。制御ファイルの再作成は、ALTER DATABASE BACKUP CONTROLFILE TO TRACE コマンドで簡単に行えます。詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

> **関連項目：** データベース名の別の変更手段である DBNEWID ユーティリティの使用方法は、『Oracle Database ユーティリティ』を参照してください。

DB_NAME 初期化パラメータ

DB_NAME には、8 文字以内のテキスト文字列を設定する必要があります。DB_NAME に指定した名前は、データベースの作成時に、データベースのデータファイル、REDO ログ・ファイルおよび制御ファイルに記録されます。データベース・インスタンスの起動時に、パラメータ・ファイル内の DB_NAME パラメータの値と制御ファイル内のデータベース名が一致しないと、データベースは起動しません。

DB_DOMAIN 初期化パラメータ

DB_DOMAIN は、データベースが作成されるネットワーク・ドメインを指定するテキスト文字列です。作成しようとしているデータベースが分散データベース・システムの一部である場合は、データベースを作成する前に、この初期化パラメータに特に注意してください。これはオプション・パラメータです。

> **関連項目：** 分散データベースの詳細は、第 V 部「分散データベースの管理」を参照してください。

フラッシュ・リカバリ領域の指定

フラッシュ・リカバリ領域は、Oracle Database でバックアップおよびリカバリに関連するファイルを格納および管理できる位置です。この領域は、現行のデータベース・ファイル（データファイル、制御ファイルおよびオンライン REDO ログ）用の位置であるデータベース領域とは異なります。

フラッシュ・リカバリ領域を指定するには、次の初期化パラメータを使用します。

- DB_RECOVERY_FILE_DEST: フラッシュ・リカバリ領域の位置。ディレクトリ、ファイル・システムまたは自動ストレージ管理（ASM）のディスク・グループです。RAW ファイル・システムにはできません。

  Oracle Real Application Clusters（RAC）環境では、この位置はクラスタ・ファイル・システム、ASM ディスク・グループ、または NFS を介して構成された共有ディレクトリであることが必要です。

- DB_RECOVERY_FILE_DEST_SIZE: フラッシュ・リカバリ領域で使用される最大総バイト数を指定します。この初期化パラメータは、DB_RECOVERY_FILE_DEST を使用可能にする前に指定する必要があります。

Oracle RAC 環境では、これら 2 つのパラメータの設定がすべてのインスタンスで同じであることが必要です。

これらのパラメータは、LOG_ARCHIVE_DEST および LOG_ARCHIVE_DUPLEX_DEST パラメータの値を設定している場合は使用可能にできません。フラッシュ・リカバリ領域を設定する前に、これらのパラメータを使用禁止にする必要があります。かわりに、LOG_ARCHIVE_DEST_*n* パラメータの値を設定できます。ローカルの LOG_ARCHIVE_DEST_*n* に値を設定しない場合は、フラッシュ・リカバリ領域を設定すると、そのフラッシュ・リカバリ領域に LOG_ARCHIVE_DEST_10 が暗黙的に設定されます。

フラッシュ・リカバリ領域によってデータベースのバックアップ操作およびリカバリ操作が簡素化されるため、この領域を使用することをお薦めします。

関連項目： フラッシュ・リカバリ領域の作成方法と使用方法については、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

制御ファイルの指定

CONTROL_FILES 初期化パラメータでは、データベースに対して 1 つ以上の制御ファイル名を指定します。CREATE DATABASE 文を実行すると、CONTROL_FILES パラメータに記述した制御ファイルが作成されます。

初期化パラメータ・ファイルに CONTROL_FILES を指定しないと、オペレーティング・システム固有のデフォルト・ファイル名を使用して、初期化パラメータ・ファイルと同じディレクトリに Oracle Database が制御ファイルを作成します。Oracle Managed Files が使用可能な場合は、Oracle Managed Files の制御ファイルが作成されます。

データベース用の制御ファイルを作成するときに新しいオペレーティング・システム・ファイルを作成する場合は、CONTROL_FILES パラメータに記述されているファイル名が現在のシステム上に存在するいずれのファイル名とも一致しないことを確認してください。データベース用の制御ファイルを作成するときに既存のファイルを再利用または上書きする場合は、CONTROL_FILES パラメータに記述されているファイル名が、現在のシステム上に存在するファイル名と一致することを確認し、CREATE DATABASE 文に CONTROLFILE REUSE 句を指定します。

データベースごとに、少なくとも 2 つの制御ファイルを別々の物理ディスク・ドライブに格納して使用することをお薦めします。

関連項目：

- 第 9 章「制御ファイルの管理」
- 2-20 ページ「データベース作成時の Oracle Managed Files の作成」

データベース・ブロック・サイズの指定

データベースの標準ブロック・サイズは、DB_BLOCK_SIZE 初期化パラメータで指定します。標準ブロック・サイズは SYSTEM 表領域で使用され、その他の表領域ではデフォルトとして使用されます。Oracle Database は、最大 4 つの非標準ブロック・サイズをサポートします。

DB_BLOCK_SIZE 初期化パラメータ

標準ブロック・サイズには、最も一般的に使用するブロック・サイズを選択します。多くの場合、設定が必要なブロック・サイズは標準ブロック・サイズのみです。通常、DB_BLOCK_SIZE は 4K または 8K に設定します。このパラメータの値を指定しないと、オペレーティング・システム固有のデフォルト・データ・ブロック・サイズが使用されますが、大抵の場合、このデフォルト・ブロック・サイズで十分です。

データベースの作成後は、データベースを再作成する以外にブロック・サイズを変更する方法はありません。データベースのブロック・サイズがオペレーティング・システムのブロック・サイズと異なる場合は、データベースのブロック・サイズをオペレーティング・システムのブロック・サイズの倍数にする必要があります。たとえば、使用しているオペレーティング・システムのブロック・サイズが 2KB（2048 バイト）の場合、次の DB_BLOCK_SIZE 初期化パラメータの設定は有効です。

```
DB_BLOCK_SIZE=4096
```

データ・ブロック・サイズを大きくすると、ディスクとメモリーの I/O（データのアクセスと格納）の効率が向上します。したがって、次の条件に該当する場合は、オペレーティング・システムのブロック・サイズより大きいブロック・サイズの指定を考慮してください。

- Oracle Database が大容量メモリーと高速ディスク・ドライブを装備した大型コンピュータ・システム上にある場合。たとえば、莫大なハードウェア資源を有するメインフレーム・コンピュータによって制御されるデータベースは、通常 4KB 以上のデータ・ブロック・サイズを使用します。

- Oracle Database が動作するオペレーティング・システムのブロック・サイズが小さい場合。たとえば、オペレーティング・システムのブロック・サイズが 1KB で、この値がデフォルトのデータ・ブロック・サイズと一致する場合、データベースは通常の処理で過度のディスク I/O を実行している可能性があります。この場合にパフォーマンスを最高にするには、データベース・ブロックを複数のオペレーティング・システム・ブロックから構成する必要があります。

  関連項目： デフォルトのブロック・サイズの詳細は、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。

非標準ブロック・サイズ

CREATE TABLESPACE 文で BLOCKSIZE 句を指定すると、非標準のブロック・サイズを持つ表領域を作成できます。これらの非標準ブロック・サイズには、2 の累乗である 2KB、4KB、8KB、16KB、32KB のいずれかを指定します。最大ブロック・サイズに関するプラットフォーム固有の制限が適用されるので、プラットフォームによっては、これらのサイズの一部は指定できない場合があります。

非標準ブロック・サイズを使用する場合は、使用するすべての非標準ブロック・サイズについて、SGA メモリーのバッファ・キャッシュ領域内にサブキャッシュを構成する必要があります。これらのサブキャッシュの構成に使用する初期化パラメータについては、5-8 ページの「自動共有メモリー管理の使用」を参照してください。

データベースに複数のブロック・サイズを指定できる機能は、特にデータベース間で表領域をトランスポートする場合に役立ちます。たとえば、OLTP 環境から、8KB の標準ブロック・サイズを使用するデータ・ウェアハウス環境に、4KB のブロック・サイズを使用する表領域をトランスポートできます。

関連項目：

- 12-3 ページ「表領域の作成」
- 12-30 ページ「データベース間での表領域のトランスポート」

最大プロセス数の指定

Oracle Database に同時に接続できるオペレーティング・システム・プロセスの最大数は、PROCESSES 初期化パラメータによって決定します。このパラメータの値は、最低でも各バックグラウンド・プロセスごとに 1 つ、および各ユーザー・プロセスごとに 1 つです。バックグラウンド・プロセスの数は、使用しているデータベースの機能によって異なります。たとえば、アドバンスト・キューイングまたはファイル・マッピング機能を使用している場合は、追加のバックグラウンド・プロセスが必要です。自動ストレージ管理を使用している場合は、データベース・インスタンス用に追加プロセスを 3 つ追加します。

50 のユーザー・プロセスを実行する予定の場合は、PROCESSES 初期化パラメータの値を 70 に見積もって設定することをお薦めします。

DDL ロック・タイムアウトの指定

データ定義言語（DDL）の文には、内部構造の排他ロックが必要です。DDL 文の実行時にこれらのロックが使用できない場合、そのすぐ後に実行していれば成功するような場合でも、DDL 文は失敗します。

DDL 文でロックを待機できるようにするには、**DDL ロック・タイムアウト**を指定します。これは、DDL コマンドが必要なロックを待機する秒数であり、この秒数を超えると DDL 文は失敗します。

DDL ロック・タイムアウトを指定するには、DDL_LOCK_TIMEOUT パラメータを使用します。設定可能な DDL_LOCK_TIMEOUT の値の範囲は、0 ～ 100,000 です。デフォルトは 0（ゼロ）です。

DDL_LOCK_TIMEOUT は、ALTER SESSION 文を使用して、システム・レベルまたはセッション・レベルで設定できます。

UNDO 領域管理方法の指定

すべての Oracle Database には、データベースの変更を取り消すために使用する情報の管理方法が必要です。これらの情報は、主にコミットされる前のトランザクションの処理レコードから構成されます。これらのレコードを総称して **UNDO データ**と呼びます。この項では、UNDO 表領域を使用する自動 UNDO 管理用の環境を設定する方法について説明します。

関連項目： 第 14 章「UNDO の管理」

UNDO_MANAGEMENT 初期化パラメータ

UNDO_MANAGEMENT 初期化パラメータは、インスタンスを自動 UNDO 管理モード（UNDO が UNDO 表領域に格納されます）で起動するかどうかを指定します。自動 UNDO 管理モードを使用可能にするには、このパラメータを AUTO に設定します。リリース 11*g* からは、パラメータを省略するか NULL にすると、デフォルトで AUTO になります。

UNDO_TABLESPACE 初期化パラメータ

インスタンスを自動 UNDO 管理モードで起動すると、そのインスタンスは、UNDO データを格納するための UNDO 表領域を選択しようとします。自動 UNDO 管理モードでデータベースが作成されている場合は、デフォルトの UNDO 表領域（システム生成の SYS_UNDOTBS 表領域またはユーザー指定の UNDO 表領域）が、インスタンス起動時に使用される UNDO 表領域となります。インスタンスに対するこのデフォルトは、UNDO_TABLESPACE 初期化パラメータに値を指定することで上書きできます。このパラメータは特に、Oracle Real Application Clusters 環境のインスタンスに特定の UNDO 表領域を割り当てる際に役立ちます。

UNDO 表領域が UNDO_TABLESPACE 初期化パラメータによって指定されていない場合は、データベース内で最初に使用可能な UNDO 表領域が選択されます。使用可能な UNDO 表領域がない場合、インスタンスは UNDO 表領域のない状態で起動し、UNDO データは SYSTEM 表領域に書き込まれます。このモードでは実行しないようにしてください。

注意： CREATE DATABASE 文を使用してデータベースを作成するときには、UNDO_TABLESPACE パラメータを初期化パラメータ・ファイルに指定しないでください。かわりに、UNDO TABLESPACE 句を CREATE DATABASE 文に指定します。

COMPATIBLE 初期化パラメータの概要

COMPATIBLE 初期化パラメータによって、ディスクのファイル形式に影響を与える機能の使用をデータベースで使用可能または使用不可にできます。たとえば、Oracle Database 11*g* リリース 1（11.1）データベースを作成した場合でも、初期化パラメータ・ファイルに COMPATIBLE = 10.0.0 を指定すると、11.1 との互換性が必要な機能の使用を試行することによりエラーが発生します。これは、データベースが 10.0.0 互換性レベルにあるとみなされているためです。

データベースの互換性レベルを上げることができます。COMPATIBLE 初期化パラメータを使用してデータベースの互換性を上げた場合、下位の互換性レベルの設定を使用してデータベースを起動する方法はありません。ただし、互換性を上げる前の時点に Point-in-Time リカバリすることはできます。

COMPATIBLE パラメータのデフォルト値は、主要な最新リリースの番号です。

注意： Oracle Database 11*g* リリース 1（11.1）の場合、COMPATIBLE パラメータのデフォルト値は 11.1.0 です。最小値は 10.0.0 です。デフォルト値を使用して Oracle Database を作成した場合は、このリリースのすべての新機能をすぐに使用でき、データベースをダウングレードすることはできません。

関連項目：

- データベースの互換性と COMPATIBLE 初期化パラメータの詳細は、『Oracle Database アップグレード・ガイド』を参照してください。
- データベースの Point-in-Time リカバリの詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

ライセンスに関するパラメータの設定

注意： Oracle では、同時セッションの数によるライセンス提供がなくなりました。したがって、LICENSE_MAX_SESSIONS および LICENSE_SESSIONS_WARNING 初期化パラメータは不要のため、非推奨になりました。

指名ユーザー・ライセンスを使用している場合、Oracle Database ではこの形式のライセンスを施行できます。データベース内に作成するユーザーの数に対して、制限を設定できます。この制限に達すると、それ以上のユーザーは作成できません。

注意： このメカニズムでは、データベースにアクセスする人がそれぞれ一意のユーザー名を持ち、ユーザー名を共有していないものと想定しています。したがって、指名ユーザー・ライセンスによって Oracle のライセンス契約に従っていることを保証できるように、複数のユーザーが同じ名前を使用してログインすることを許可しないでください。

データベースに作成するユーザー数を制限するには、次の例に示すように、そのデータベースの初期化パラメータ・ファイルに LICENSE_MAX_USERS 初期化パラメータを設定します。

```
LICENSE_MAX_USERS = 200
```

サーバー・パラメータ・ファイルを使用した初期化パラメータの管理

Oracle Database の初期化パラメータは、テキスト形式の初期化パラメータ・ファイルに格納されていました。管理性を向上させるために、データベースを起動および停止している間も持続するバイナリ形式のサーバー・パラメータ・ファイルによる初期化パラメータのメンテナンスを選択できます。ここでは、サーバー・パラメータ・ファイルの概要を示し、各パラメータ格納方式を使用した初期化パラメータの管理方法について説明します。この項の内容は、次のとおりです。

サーバー・パラメータ・ファイルの概要

サーバー・パラメータ・ファイルは、Oracle Database サーバーが稼働するマシンで管理される初期化パラメータのリポジトリと考えられます。これは、サーバー側初期化パラメータ・ファイルとして設計されています。サーバー・パラメータ・ファイルに格納された初期化パラメータは永続的で、インスタンスの実行中に行ったパラメータの変更は、インスタンスを停止し、起動しても有効です。この配置によって、ALTER SYSTEM 文による変更を持続させるために、初期化パラメータを手動で更新する必要がなくなります。また、Oracle Database サーバーによる自己チューニングの基礎ともなります。

最初のサーバー・パラメータ・ファイルは、CREATE SPFILE 文を使用して、テキスト形式の初期化パラメータ・ファイルから作成します（Database Configuration Assistant で直接作成することもできます）。サーバー・パラメータ・ファイルは、テキスト・エディタで編集できないバイナリ・ファイルです。Oracle Database には、サーバー・パラメータ・ファイル内のパラメータの設定を表示および変更するために、他のインタフェースが用意されています。

注意： テキスト・エディタでバイナリ形式のサーバー・パラメータ・ファイルをオープンし、そのテキストを表示することは可能ですが、手動で編集しないでください。手動で編集すると、ファイルが破損します。また、インスタンスが起動できなくなり、たとえインスタンスが起動しても失敗します。

PFILE 句を指定せずに STARTUP コマンドを発行すると、Oracle インスタンスは、オペレーティング・システム固有のデフォルトの位置でサーバー・パラメータ・ファイルを検索し、そのファイルから初期化パラメータの設定を読み込みます。サーバー・パラメータ・ファイルが見つからない場合は、テキスト形式の初期化パラメータ・ファイルを検索します。サーバー・パラメータ・ファイルがあってもテキスト形式の初期化パラメータ・ファイルの設定を優先する場合は、STARTUP コマンドの発行時に PFILE 句を指定する必要があります。サーバー・パラメータ・ファイルを使用してインスタンスを起動する方法の詳細は、3-2 ページの「データベースの起動」を参照してください。

サーバー・パラメータ・ファイルへの移行

現在、テキスト形式の初期化パラメータ・ファイルを使用している場合は、次の手順に従ってサーバー・パラメータ・ファイルに移行します。

1. 初期化パラメータ・ファイルがクライアント・マシン上にある場合は、FTP などを使用して、クライアント・マシンからサーバー・マシンにファイルをトランスポートしてください。

   注意： Oracle Real Application Clusters 環境でサーバー・パラメータ・ファイルに移行する場合は、インスタンス固有のすべての初期化パラメータ・ファイルを、1 つの初期化パラメータ・ファイルに結合する必要があります。この操作の方法、および Oracle Real Application Clusters のインストールの一部であるインスタンスでサーバー・パラメータ・ファイルを使用する際の固有の処理については、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』および使用しているプラットフォーム固有の Oracle Real Application Clusters インストレーション・ガイドを参照してください。

   ---

2. CREATE SPFILE FROM PFILE 文を使用して、デフォルト位置にサーバー・パラメータ・ファイルを作成します。手順については、2-33 ページの「サーバー・パラメータ・ファイルの作成」を参照してください。

   この文を実行すると、テキスト形式の初期化パラメータ・ファイルが読み込まれて、サーバー・パラメータ・ファイルが作成されます。CREATE SPFILE 文を発行するために、データベースを起動する必要はありません。

3. インスタンスを起動または再起動します。

   インスタンスはデフォルト位置にある新規 SPFILE を検出し、そのファイルを使用して起動します。

サーバー・パラメータ・ファイルの作成

サーバー・パラメータ・ファイルを作成するには、CREATE SPFILE 文を使用します。この文を実行するには、SYSDBA または SYSOPER システム権限が必要です。

> **注意：** DBCA を使用してデータベースを作成した場合は、サーバー・パラメータ・ファイルが自動的に作成されます。

CREATE SPFILE 文は、インスタンスの起動前後に実行できます。ただし、すでにサーバー・パラメータ・ファイルを使用してインスタンスを起動している場合に、インスタンスで現在使用されているのと同じサーバー・パラメータ・ファイルを再作成しようとすると、エラーが発生します。

サーバー・パラメータ・ファイル（SPFILE）は、既存のテキスト形式の初期化パラメータ・ファイルまたはメモリーから作成できます。メモリーからの SPFILE の作成では、実行中のインスタンス内の初期化パラメータの現行値を SPFILE にコピーします。

次の例では、テキスト形式の初期化パラメータ・ファイル /u01/oracle/dbs/init.ora からサーバー・パラメータ・ファイルを作成しています。この例では SPFILE の名前を指定していないため、ファイルは、2-34 ページの表 2-3 に示したプラットフォーム固有のデフォルトの名前と場所で作成されます。

```
CREATE SPFILE FROM PFILE='/u01/oracle/dbs/init.ora';
```

次の例では、名前と場所を指定してサーバー・パラメータ・ファイルを作成しています。

```
CREATE SPFILE='/u01/oracle/dbs/test_spfile.ora'
       FROM PFILE='/u01/oracle/dbs/test_init.ora';
```

次の例では、メモリー内の初期化パラメータの現行値から、デフォルトの場所にサーバー・パラメータ・ファイルを作成しています。

```
CREATE SPFILE FROM MEMORY;
```

デフォルトの SPFILE 名と場所を使用するか、SPFILE 名と場所を指定するかどうかに関係なく、同じ名前の SPFILE がその場所にすでに存在する場合は、警告メッセージなしに上書きされます。

テキスト形式の初期化パラメータ・ファイルから SPFILE を作成すると、初期化パラメータ・ファイル内のパラメータ設定と同じ行に記述されているコメントも SPFILE で管理されます。他のコメントはすべて無視されます。

SPFILE の名前と格納場所には、データベースによるデフォルト設定を使用することをお薦めします。これにより、データベースの管理が容易になります。たとえば、STARTUP コマンドはこのデフォルトの場所を想定して、SPFILE を読み込みます。

表 2-3 は、UNIX、Linux および Windows プラットフォームごとに、テキスト形式の初期化パラメータ・ファイル（PFILE）とサーバー・パラメータ・ファイル（SPFILE）のデフォルトの名前と場所を示しています。この表では、SPFILE をファイルと想定しています。RAW デバイスの場合、デフォルトの名前は論理ボリュームまたはパーティション・デバイスの名前となり、デフォルトの場所も異なります。

表 2-3 UNIX、LInux、Windows における PFILE および SPFILE のデフォルトの名前と場所

プラットフォーム	PFILE のデフォルト名	SPFILE のデフォルト名	デフォルトの場所（PFILE および SPFILE）
UNIX と Linux	init*ORACLE_SID*.ora	spfile*ORACLE_SID*.ora	*ORACLE_HOME*/dbs、またはデータファイルと同じ場所
Windows	init*ORACLE_SID*.ora	spfile*ORACLE_SID*.ora	*ORACLE_HOME*¥database

注意： 起動時に、インスタンスは最初に spfile*ORACLE_SID*.ora という名前の SPFILE を検索し、見つからない場合は spfile.ora を検索します。spfile.ora を使用すると、すべての Real Application Cluster（RAC）インスタンスで同じサーバー・パラメータ・ファイルを使用できます。

いずれの SPFILE も見つからない場合、インスタンスはテキスト形式の初期化パラメータ・ファイル init*ORACLE_SID*.ora を検索します。

デフォルトの場所以外の場所に SPFILE を作成する場合は、そのサーバー・パラメータ・ファイルを指し示すテキスト形式の初期化パラメータ・ファイルを作成する必要があります。詳細は、3-2 ページの「データベースの起動」を参照してください。

HARD 対応記憶域へのサーバー・パラメータ・ファイルの格納

リリース 11*g* からは、サーバー・パラメータ・ファイル（SPFILE）は、Oracle Hardware Assisted Resilient Data（HARD）イニシアティブに準拠した新規形式です。HARD では、破損データが永続記憶域に書き込まれないように、ソフトウェアと記憶域ハードウェアの両方のレベルで実装されるデータ検証アルゴリズムの包括的なセットが定義されます。SPFILE 内のデータについて HARD 保護を完全に使用可能にするには、SPFILE が HARD 対応の記憶域に存在し、データベース・インスタンスの互換性が少なくとも 11.0.0 に上げられている必要があります。

HARD 準拠の SPFILE は、HARD 対応以外の記憶域に格納できます。この場合、新規 SPFILE 形式では、破損 SPFILE データの検出のみがサポートされます。SPFILE を HARD 対応の記憶域に格納すると、破損データが記憶域に書き込まれることはありません。

HARD の詳細、および HARD 対応のストレージ・システムを提供するストレージ・ベンダーのリストは、次の URL を参照してください。
http://www.oracle.com/technology/deploy/availability/htdocs/HARD.html

Oracle データベースのインストールまたはアップグレード時には、完全な HARD 保護のために次のガイドラインに従ってください。

リリース 11*g* のデータベースをインストールまたは最初に作成する場合

リリース 11*g* のデータベースを最初にインストールまたは作成する場合、COMPATIBLE 初期化パラメータはデフォルトで 11.1.0 に設定されるため、HARD 準拠のサーバー・パラメータ・ファイル（SPFILE）に対するこの要件は満たされます。次に、SPFILE が HARD 対応の記憶域に格納されていることを確認する必要があります。この要件を満たすには、次のいずれかを実行します。

- 共有記憶域のない Oracle Real Application Clusters 環境の場合は、DBCA から SPFILE の場所の指定を要求されたときに、HARD 対応の記憶域上の場所を指定します。
- 単一インスタンスのインストールの場合、または共有記憶域のある Oracle Real Application Clusters 環境の場合は、次の手順を実行します。

  1. Database Configuration Assistant（DBCA）を使用してデータベースのインストールを完了します。

     SPFILE がデフォルトの場所に作成されます。デフォルトの場所の詳細は、2-34 ページの表 2-3 を参照してください。

2. 次のいずれかを実行します。
   - オペレーティング・システムのコマンドを使用して、SPFILE を HARD 対応の記憶域にコピーします。
   - SQL*Plus または SQL Developer などの別の対話型環境で、ユーザー SYS としてデータベースに接続し、次のコマンドを発行します。

     ```
     CREATE SPFILE = 'spfile_name' FROM MEMORY;
     ```

     spfile_name は、HARD 対応の記憶域を指し示す、ファイル名までの完全なパス名です。
3. 次のいずれかを実行します。
   - 次の単一のエントリで、デフォルトの場所にテキスト形式の初期化パラメータ・ファイル（PFILE）を作成します。

     ```
     SPFILE = spfile_name
     ```

     spfile_name は、HARD 対応の記憶域上の SPFILE までの完全なパスです。
   - UNIX および Linux プラットフォームで、デフォルトの SPFILE の場所に、HARD 対応の記憶域上の SPFILE へのシンボリック・リンクを作成します。

   PFILE および SPFILE のデフォルトの名前と場所に関する情報は、表 2-3 を参照してください。
4. データベース・インスタンスを停止します。
5. デフォルトの場所にある SPFILE を削除します。
6. データベース・インスタンスを起動します。

以前のデータベース・リリースからリリース 11*g* にアップグレードする場合

以前のデータベース・リリースからリリース 11*g* にアップグレードする場合は、次の手順を完了して SPFILE を HARD 準拠の形式に移行し、HARD 対応の記憶域に格納します。

1. SQL*Plus または別の対話型問合せアプリケーションを起動し、ユーザー SYS または SYSTEM としてデータベースにログインし、次のコマンドを入力します。

   ```
   ALTER SYSTEM SET COMPATIBLE = '11.1.0' SCOPE = SPFILE;
   ```

   警告： 互換性レベルを 11.1.0 に上げると、11*g* の機能とファイル形式が使用可能になり、様々な影響が発生します。実行する前に、『Oracle Database アップグレード・ガイド』を参照してください。

2. データベース・インスタンスを再起動します。

   データベースの互換性レベルは現在 11.1.0 です。
3. SPFILE がまだ HARD 対応の記憶域に存在していない場合は、次の手順を実行します。
   a. SQL*Plus または別の対話型環境で、ユーザー SYS としてデータベースに接続し、次のコマンドを発行します。

      ```
      CREATE SPFILE = 'spfile_name' FROM MEMORY;
      ```

      spfile_name は、HARD 対応の記憶域を指し示す、ファイル名までの完全なパス名です。
   b. 次のいずれかを実行します。
      - 次の単一のエントリで、デフォルトの場所にテキスト形式の初期化パラメータ・ファイル（PFILE）を作成します。

        ```
        SPFILE = spfile_name
        ```

        spfile_name は、HARD 対応の記憶域上の SPFILE までの完全なパスです。

– UNIX および Linux プラットフォームで、デフォルトの SPFILE の場所に、HARD 対応の記憶域上の SPFILE へのシンボリック・リンクを作成します。

PFILE および SPFILE のデフォルトの名前と場所に関する情報は、表 2-3 を参照してください。

c. データベース・インスタンスを停止します。

d. デフォルトの場所にある SPFILE を削除します。

e. データベース・インスタンスを起動します。

SPFILE 初期化パラメータ

SPFILE 初期化パラメータには、現在のサーバー・パラメータ・ファイルの名前を指定します。データベースでデフォルトのサーバー・パラメータ・ファイルが使用されている場合（つまり、PFILE パラメータを指定せずに STARTUP コマンドを発行した場合）、SPFILE の値はサーバーによって内部的に設定されます。SQL*Plus コマンドの SHOW PARAMETERS SPFILE（またはパラメータの値を問い合せる他の方法）を使用すると、現在使用中のサーバー・パラメータ・ファイルの名前が表示されます。

初期化パラメータ値の変更

ALTER SYSTEM 文を使用すると、初期化パラメータ値を設定、変更またはデフォルトにリストアできます。テキスト形式の初期化パラメータ・ファイルを使用している場合、現行のインスタンスに対するパラメータの値のみ ALTER SYSTEM 文で変更されます。これは、ディスク上のテキスト形式の初期化パラメータを自動的に更新するメカニズムがないためです。以後のインスタンスに渡すためには、ディスク上の初期化パラメータを手動で更新する必要があります。サーバー・パラメータ・ファイルを使用している場合は、このような制限はありません。

初期化パラメータには、次の 2 種類があります。

- **動的な初期化パラメータ** : 現行の Oracle Database インスタンスに対して変更できます。変更は即時に有効になります。
- **静的な初期化パラメータ** : 現行のインスタンスに対して変更できません。これらのパラメータはテキスト形式の初期化パラメータまたはサーバー・パラメータ・ファイルで変更し、変更を有効にするにはデータベースを再起動する必要があります。

初期化パラメータ値の設定または変更

初期化パラメータ値を設定または変更するには、ALTER SYSTEM 文で SET 句を指定します。また、次の表のように、変更の適用範囲を指定する場合は、オプションの SCOPE 句を指定します。

SCOPE 句	説明
SCOPE = SPFILE	サーバー・パラメータ・ファイルのみに変更が適用されます。次のような効果があります。 ■ 変更は現行インスタンスには適用されません。 ■ 動的パラメータと静的パラメータの両方について、次の起動時に変更が有効になり、以後持続します。 静的パラメータでは、SCOPE 指定のみ使用できます。
SCOPE = MEMORY	メモリーのみに変更が適用されます。次のような効果があります。 ■ 変更は現行インスタンスに適用され、即時に有効になります。 ■ 動的パラメータの場合は、変更が即時に有効になりますが、サーバー・パラメータ・ファイルは更新されないので、変更は持続しません。 静的パラメータでは、この指定は使用できません。

SCOPE 句	説明
SCOPE = BOTH	サーバー・パラメータ・ファイルとメモリーの両方に変更が適用されます。次のような効果があります。 ■ 変更は現行インスタンスに適用され、即時に有効になります。 ■ 動的パラメータの場合は、サーバー・パラメータ・ファイルが更新されるので、変更が持続します。 静的パラメータでは、この指定は使用できません。

インスタンスがサーバー・パラメータ・ファイルを使用して起動していない場合に SCOPE=SPFILE または SCOPE=BOTH を指定すると、エラーが発生します。デフォルトは、インスタンス起動時にサーバー・パラメータ・ファイルを使用した場合は SCOPE=BOTH、テキスト形式の初期化パラメータ・ファイルを使用した場合は MEMORY になります。

動的パラメータの場合は、DEFERRED キーワードを指定することもできます。このキーワードを指定すると、これから確立するセッションでのみ変更が有効になります。

SCOPE を SPFILE または BOTH に指定した場合は、オプションの COMMENT 句を使用すると、パラメータの更新にテキスト文字列を関連付けることができます。サーバー・パラメータ・ファイルにコメントが記述されます。

次の文は、接続を削除するまでのログイン試行の最大失敗回数を変更します。この文にはコメントが含まれており、変更をサーバー・パラメータ・ファイルにのみ適用することを明示しています。

```
ALTER SYSTEM SET SEC_MAX_FAILED_LOGIN_ATTEMPTS=3
                 COMMENT='Reduce from 10 for tighter security.'
                 SCOPE=SPFILE;
```

次の例では、属性のリストをとる複雑な初期化パラメータを設定しています。値を設定するパラメータは、LOG_ARCHIVE_DEST_n 初期化パラメータです。この文によって、このパラメータの既存の設定を変更するか、または新しいアーカイブ先を作成できます。

```
ALTER SYSTEM
     SET LOG_ARCHIVE_DEST_4='LOCATION=/u02/oracle/rbdb1/',MANDATORY,'REOPEN=2'
         COMMENT='Add new destimation on Nov 29'
         SCOPE=SPFILE;
```

値がパラメータのリストで構成されている場合は、個々の属性を位置または序数によって編集することはできません。パラメータを更新するたびに、完全な値のリストを指定する必要があります。これによって、新しいリストで古いリストが完全に置換されます。

初期化パラメータ値のクリア

ALTER SYSTEM RESET コマンドを使用すると、インスタンスの起動に使用した SPFILE の初期化パラメータの設定をクリア（削除）できます。SCOPE=MEMORY および SCOPE=BOTH はいずれも指定できません。SCOPE = SPFILE 句は必要ありませんが、指定してもかまいません。

次回のデータベース起動時にデフォルト値が使用されるように、SPFILE のパラメータをクリアできます。

> **関連項目：** ALTER SYSTEM コマンドの詳細は、『Oracle Database SQL リファレンス』を参照してください。

サーバー・パラメータ・ファイルのエクスポート

CREATE PFILE 文を使用すると、サーバー・パラメータ・ファイル（SPFILE）をテキスト形式の初期化パラメータ・ファイルにエクスポートできます。このエクスポートは、次のような場合に必要になる場合があります。

- 診断のために、現在インスタンスで使用しているすべてのパラメータのリストを作成する場合。これは、SQL*Plus の SHOW PARAMETERS コマンド、あるいは V$PARAMETER または V$PARAMETER2 ビューの選択と同じです。
- 最初にエクスポートし、結果のテキスト・ファイルを編集してから、CREATE SPFILE 文を使用して再作成するという手順で、サーバー・パラメータ・ファイルを変更する場合。

PFILE 句にエクスポート・ファイルを指定して、インスタンスを起動することもできます。

CREATE PFILE 文を実行するには、SYSDBA または SYSOPER システム権限が必要です。エクスポート・ファイルは、データベース・サーバー・マシン上に作成されます。パラメータ設定と同じ行に記述されている、パラメータに関するコメントがすべて含まれます。

次の例では、SPFILE からテキスト形式の初期化パラメータ・ファイルを作成しています。

```
CREATE PFILE FROM SPFILE;
```

この例ではファイル名を指定していないため、プラットフォーム固有のデフォルト・サーバー・パラメータ・ファイルから、プラットフォーム固有の名前で初期化パラメータ・ファイルが作成されます。

次の例では、サーバー・パラメータ・ファイルからテキスト形式の初期化パラメータ・ファイルを作成していますが、複数のファイル名が指定されています。

```
CREATE PFILE='/u01/oracle/dbs/test_init.ora'
       FROM SPFILE='/u01/oracle/dbs/test_spfile.ora';
```

注意： メモリー内の初期化パラメータの現行値から PFILE を作成する方法もあります。次は、必要なコマンドの例です。

```
CREATE PFILE='/u01/oracle/dbs/test_init.ora' FROM MEMORY;
```

サーバー・パラメータ・ファイルのバックアップの作成

サーバー・パラメータ・ファイル（SPFILE）をエクスポートして、そのバックアップを作成できます。詳細は、「サーバー・パラメータ・ファイルのエクスポート」を参照してください。データベースのバックアップおよびリカバリ計画が Recovery Manager を使用して実装されている場合は、Recovery Manager を使用して SPFILE のバックアップを作成できます。SPFILE のバックアップは、データベースのバックアップ作成時に自動的に作成されますが、Recovery Manager を使用すると現在アクティブになっている SPFILE のバックアップを作成できます。

関連項目：『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』

失われたまたは破損したサーバー・パラメータ・ファイルのリカバリ

サーバー・パラメータ・ファイル（SPFILE）が失われるかまたは破損した場合は、現行インスタンスが失敗するか、またはデータベース・インスタンス起動時の次回の試行が失敗する可能性があります。SPFILE をリカバリする方法は複数あります。

- インスタンスが実行中の場合は、次のコマンドを発行して、メモリー内の初期化パラメータの現行値から SPFILE を再作成します。

  ```
  CREATE SPFILE FROM MEMORY;
  ```

  このコマンドでは、SPFILE はデフォルトの場所にデフォルトの名前で作成されます。新しい名前を使用して、または指定した場所に SPFILE を作成することもできます。例については、2-33 ページの「サーバー・パラメータ・ファイルの作成」を参照してください。

- 有効なテキスト形式の初期化パラメータ・ファイル（PFILE）がある場合は、次のコマンドを使用して PFILE から SPFILE を再作成します。

  ```
  CREATE SPFILE FROM PFILE;
  ```

  このコマンドは、PFILE がデフォルトの場所にあり、デフォルトの名前であると想定しています。PFILE がデフォルト以外の場所にあるか、デフォルト以外の名前の場合に使用するコマンド構文については、2-33 ページの「サーバー・パラメータ・ファイルの作成」を参照してください。

- バックアップから SPFILE をリストアします。

  詳細は、2-38 ページの「サーバー・パラメータ・ファイルのバックアップの作成」を参照してください。

- これらのいずれの方法も使用できない状況の場合は、次の手順を実行します。

  1. アラート・ログのパラメータ値のリストからテキスト形式の初期化パラメータ・ファイル（PFILE）を作成します。

     インスタンスの起動時に、起動に使用された初期化パラメータがアラート・ログに書き込まれます。このセクションを（XML タグが含まれていない）テキスト・バージョンのアラート・ログからコピーして、新規 PFILE に貼り付けることができます。

     詳細は、8-19 ページの「アラート・ログの表示」を参照してください。

  2. PFILE から SPFILE を作成します。

     手順については、2-33 ページの「サーバー・パラメータ・ファイルの作成」を参照してください。

パラメータ更新時の読取り / 書込みエラー

パラメータの更新時にサーバー・パラメータ・ファイルの読込みまたは書込みでエラーが発生した場合は、アラート・ログにエラーが記録されて、サーバー・パラメータ・ファイルに対する後続のパラメータ更新がすべて無視されます。この時点では、次の処理のいずれかを実行できます。

- インスタンスを停止し、この項の前述の説明に従ってサーバー・パラメータ・ファイルをリカバリした後、インスタンスを再起動します。
- 後続のパラメータ更新を持続的にするための処置を講じない場合は、データベースの実行を継続します。

パラメータ設定の表示

パラメータの設定は、次の表に示すように、いくつかの方法で表示できます。

方法	説明
SHOW PARAMETERS	この SQL*Plus コマンドを使用すると、現行セッションで有効な初期化パラメータの値が表示されます。
SHOW SPPARAMETERS	この SQL*Plus コマンドを使用すると、サーバー・パラメータ・ファイル（SPFILE）の初期化パラメータの値が表示されます。
CREATE PFILE	この SQL 文は、SPFILE または現在のメモリー内設定からテキスト形式の初期化パラメータ・ファイル（PFILE）を作成します。PFILE はテキスト・エディタで表示できます。
V$PARAMETER	このビューを使用すると、現行セッションで有効な初期化パラメータの値が表示されます。
V$PARAMETER2	このビューを使用すると、現行セッションで有効な初期化パラメータの値が表示されます。各リスト・パラメータ値が別々の行に出力されるので、このビューのほうがリスト・パラメータ値を容易に識別できます。
V$SYSTEM_PARAMETER	このビューを使用すると、インスタンスで有効な初期化パラメータの値が表示されます。新規セッションは、インスタンス全体の値からパラメータ値を継承します。
V$SYSTEM_PARAMETER2	このビューを使用すると、インスタンスで有効な初期化パラメータの値が表示されます。新規セッションは、インスタンス全体の値からパラメータ値を継承します。各リスト・パラメータ値が別々の行に出力されるので、このビューのほうがリスト・パラメータ値を容易に識別できます。
V$SPPARAMETER	このビューを使用すると、SPFILE の現在の内容が表示されます。インスタンスで SPFILE が使用されていない場合は、ISSPECIFIED 列に FALSE が返されます。

関連項目： ビューの詳細は、『Oracle Database リファレンス』を参照してください。

データベース・サービスの定義

ここでは、データベース・サービスについて説明します。この項の内容は、次のとおりです。

データベース・サービス（サービス）とは、Oracle Database のワークロードを管理するための論理的な抽象概念です。サービスによって、ワークロードは相互に独立したグループに分割されます。各サービスは、共通の属性、サービスレベルのしきい値および優先度を持つワークロードを表します。グループ化は作業の属性に基づいて行われます。これらの属性には、使用するアプリケーション機能、アプリケーション機能を実行する場合の優先度、管理の対象となるジョブ・クラス、アプリケーション機能またはジョブ・クラスで使用するデータの範囲などが含まれる場合があります。たとえば、Oracle E-Business Suite では、一般会計、売掛 / 未収金、受注などの各役割に対するサービスが定義されています。各データベース・サービスには一意の名前が付いています。

接続要求には、データベース・サービス名を指定できます。サービス名が指定されておらず、Net Services ファイル listener.ora にデフォルト・サービスが指定されている場合は、デフォルト・サービスを使用して接続されます。

複数のサービスが Oracle Database に組み込まれており、ワークロードに関する単一のシステム・イメージ、ワークロードの優先度、現実のトランザクションに対するパフォーマンス測定、およびパフォーマンス目標に違反した場合のアラートと処理を提供しています。サービスを使用すると、ワークロードの構成、管理、有効化と無効化を実行でき、さらに単一エンティティとして測定できます。これらの作業は、Database Configuration Assistant（DBCA）、Net Configuration Assistant（NetCA）、Enterprise Manager（EM）などの標準的なツールを使用して実行できます。Enterprise Manager は、表示と操作に関するサービスを全体としてサポートし、必要な場合はインスタンス・レベルへのドリルダウンをサポートしています。

Real Application Clusters（RAC）は 1 つ以上のインスタンスにまたがるサービスで、実際のトランザクション・パフォーマンスに基づいた現実的なワークロードの均衡化に役立ちます。これによって、エンド・トゥ・エンドの無人リカバリ、ワークロードによるロール変更、位置の完全な透過性が可能となります。また、RAC を使用すると、Enterprise Manager、DBCA および Server Control Utility（SRVCTL）で多数のサービス機能を管理できます。

サービスは、パフォーマンス・チューニングに追加のディメンションも提供します。すべてのセッションを匿名で共有している大部分のシステムでは、「サービスと SQL」によるチューニングで「セッションと SQL」によるチューニングを置換できます。サービスを使用することで、ワークロードが表示可能および測定可能となります。リソースの使用と待機は、アプリケーションがその起因となっています。また、サービスに割り当てたリソースは、ロードの増減にあわせて調整できます。この動的なリソース割当てによって、要求の発生に対応した費用効率の高いソリューションが可能となります。たとえば、サービスを自動的に測定し、そのパフォーマンスをサービス・レベルのしきい値と比較できます。パフォーマンス違反は Enterprise Manager にレポートされるため、自動ソリューションまたはスケジュールされたソリューションを実行できます。

関連項目：『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』

サービスのデプロイ

データベース・サービスを構成するとき、一意のグローバル名、関連するパフォーマンス目標および関連する重要性を各サービスに指定します。これらのサービスは、Oracle Database と緊密に統合され、データ・ディクショナリに保持されます。サービス情報は、次のサービス固有のビューで参照できます。

- DBA_SERVICES
- ALL_SERVICES または V$SERVICES
- V$ACTIVE_SERVICES
- V$SERVICE_STATS
- V$SERVICE_EVENTS
- V$SERVICE_WAIT_CLASSES
- V$SERV_MOD_ACT_STATS
- V$SERVICE_METRICS
- V$SERVICE_METRICS_HISTORY

次の追加ビューにも、サービスに関する情報が表示されます。

- V$SESSION
- V$ACTIVE_SESSION_HISTORY
- DBA_RSRC_GROUP_MAPPINGS
- DBA_SCHEDULER_JOB_CLASSES
- DBA_THRESHOLDS

> **関連項目：** これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。

Oracle Database 機能のいくつかは、サービスをサポートしています。自動ワークロード・リポジトリ（AWR）は、サービスのパフォーマンスを管理します。AWR には、実行時間、待機クラスおよびサービスで使用されたリソースも含めて、サービスのパフォーマンスが記録されます。AWR は、サービス応答時間がしきい値を超えた場合、警告をアラートします。動的なビューには、現在のサービス・パフォーマンスのメトリックが時間の履歴単位でレポートされます。各サービスには、応答時間と CPU 使用に関するサービス品質のしきい値があります。

さらに、データベース・リソース・マネージャによって、サービスがコンシューマ・グループにマップされます。このマッピングによって、サービスの優先度を他のサービスと関連させて自動的に管理できます。コンシューマ・グループを使用すると、比率またはリソース使用の観点から相対的な優先順序を定義できます。この点に関する詳細は、『Oracle Real Application Clusters 配置およびパフォーマンス』を参照してください。

サービスの構成

サービスには、アプリケーション、アプリケーション機能およびデータの範囲が、機能サービスまたはデータ依存サービスとして記述されています。機能サービスは最も一般的なワークロードのマッピングです。特定の機能を使用する複数のセッションはまとめてグループ化されます。Oracle*Applications、ERP、CRM および *i*Support の機能については、作業の機能的な分割が作成されます。SAP、ダイアログおよび更新の機能についても、作業の機能的な分割が作成されます。

これに対して、データ依存ルーティングは、データ・キーに基づいてセッションをサービスにルーティングします。作業要求のサービスへのマッピングは、アプリケーション・サーバーと TP モニターのオブジェクト関連マッピング・レイヤーで発生します。たとえば、RAC では、データベースが共有されているため、これらの範囲は要求に基づいて完全に動的にできます。

また、RAC データベースには、事前接続のアプリケーション・サービスを定義できます。事前接続のサービスは、複数のインスタンスにまたがって、障害時にサービスをサポートします。この事前接続のサービスは、TAF 事前接続モードをサポートし、RAC 使用時には透過的に管理されます。

アプリケーション・サービスに加えて、Oracle Database では、2 つの内部サービスもサポートされています。`SYS$BACKGROUND` はバックグラウンド・プロセスのみで使用され、`SYS$USERS` はサービスに関連していないユーザー・セッションに対するデフォルトのサービスです。

`DBMS_SERVICE` パッケージを使用するか、または `SERVICE_NAMES` パラメータを設定して、単一インスタンスの Oracle Database にアプリケーション・サービスを作成します。応答時間の目標や各サービスの重要性は、後で、EM を介して個別に定義するか、または Enterprise Manager の機能である「Copy Thresholds From a Baseline」を使用して「Manage Metrics」ページまたは「Edit Threshold」ページで定義できます。PL/SQL を使用して定義することもできます。

サービスの使用

サービスを使用するために、アプリケーション・コードを変更する必要はありません。クライアント側の作業はサービスへの接続です。サーバー側の作業は、ジョブ・スケジューラに対するジョブ・クラスの作成時にサービスを指定し、分散データベースにデータベース・リンクを指定することです。サービス下で実行される作業要求は、そのサービスのパフォーマンスしきい値を継承し、サービスの一部として測定されます。

クライアント側での使用

中間層アプリケーションおよびクライアントとサーバーのアプリケーションでは、サービスを TNS 接続データ内の接続の一部として指定することで、サービスを利用します。この接続データは、Thick Net ドライバ用の `TNSnames` ファイル内、Thin ドライバ用の URL 指定ファイル内にあるか、または Oracle Internet Directory 内に保持されている可能性があります。たとえば、Oracle Application Server のデータソースは、サービスにルーティングされるように設定されています。簡易接続ネーミングを使用している場合、この接続には、`hr@myDBhost/myservice` のようなホスト名とサービス名のみが必要です。また、Oracle E-Business Suite の場合、サービスはアプリケーション・データベース識別子で保持され、ICX パラメータの Cookie に保持されます。

サーバー側での使用

Oracle Scheduler、パラレル実行、Oracle Streams アドバンスト・キューイングなどのサーバー側での作業では、ワークロード定義の一部としてサービス名を設定します。

Oracle Scheduler の場合は、ジョブ・クラスの作成時にそのジョブ・クラスで使用するサービスをオプションで定義します。実行中に複数のジョブがジョブ・クラスに割り当てられ、サービス内で複数のジョブ・クラスを実行できます。ジョブ・クラスとともにサービスを使用することで、ジョブ・スケジューラによって実行される作業が、ワークロード管理とパフォーマンス・チューニングに対して示されます。

パラレル問合せとパラレル DML の場合、問合せコーディネータは他のクライアントと同じようにサービスに接続します。パラレル問合せプロセスは、実行中そのサービスを継承します。問合せが終了した時点で、パラレル実行プロセスはデフォルトのサービスに戻ります。

関連項目： Oracle Scheduler の詳細は、第 27 章「Oracle Scheduler を使用したジョブのスケジューリング」を参照してください。

データベースを作成した後の考慮点

2-5 ページの「DBCA を使用したデータベースの作成」または 2-6 ページの「CREATE DATABASE 文を使用したデータベースの作成」で説明されているように、データベースの作成後はインスタンスが稼働しており、データベースがオープンしているので、通常どおりにデータベースを使用できます。必要に応じて、他の処理を実行できます。ここでは、その処理の一部について説明します。

セキュリティに関する考慮点

このリリースの Oracle Database では、データベースのセキュリティを確保するためにいくつかの機能が拡張されました。このリリースのセキュリティ・ガイドラインは、『Oracle Database セキュリティ・ガイド』にあります。これらのガイドラインを参照し、その内容に従ってデータベースを構成してください。

データベースを作成した後は、Oracle Identity Management が活用できるように構成できます。構成方法の詳細は、『Oracle Database エンタープライズ・ユーザー・セキュリティ管理者ガイド』を参照してください。

新しく作成したデータベースには、データベースの管理にとって重要なユーザー・アカウントが少なくとも 3 つあります。SYS、SYSTEM および SYSMAN です。認可されたユーザーのみが使用する追加の管理アカウントが提供されています。これらのアカウントを、オラクル社が提供するパスワードを理解している未認可のユーザーによる使用から保護するには、それぞれのパスワードを期限切れにして最初にロックします。データベース管理者には、これらのアカウントのロック解除とリセットに関する役割があります。

新規の各 Oracle Database インストールで作成される事前定義のユーザー・アカウントの全リストは、『Oracle Database 2 日でセキュリティ・ガイド』を参照してください。

注意： データベースへの不当なアクセスを防ぎ、データベースの整合性を保護するには、データベースの作成時に、ユーザー・アカウント SYS と SYSTEM に対して新しいパスワードを指定することが重要です。パスワードを指定するには、手動でデータベースを作成するときに次の CREATE DATABASE 句を指定するか、または DBCA を使用してデータベースを作成します。

- `USER SYS IDENTIFIED BY`
- `USER SYSTEM IDENTIFIED BY`

関連項目：

- ユーザー SYS および SYSTEM の詳細は、1-14 ページの「管理ユーザー・アカウント」を参照してください。
- 新しいユーザーの追加方法とパスワードの変更方法は、『Oracle Database セキュリティ・ガイド』を参照してください。
- ユーザー・アカウントのロック解除に使用する ALTER USER 文の構文は、『Oracle Database SQL リファレンス』を参照してください。

透過的データ暗号化の有効化

透過的データ暗号化は、個々のデータベース列をデータファイルに格納する前に暗号化するか、表領域全体を暗号化する機能です。ユーザーが、オペレーティング・システムのツールを使用してデータファイルの内容を直接参照することによって、データベース・アクセス制御メカニズムを迂回しようとした場合でも、透過的データ暗号化によって、このようなユーザーが機密情報を参照できないようにします。

CREATE TABLE 権限のあるユーザーは、暗号化対象の表の列を 1 つ以上選択できます。データは、データファイルおよび監査ログ（監査がオンの場合）内で暗号化されます。適切な権限のあるデータベース・ユーザーは、データを暗号化されていない形式で表示できます。透過的データ暗号化の有効化および無効化の詳細は、『Oracle Database Advanced Security 管理者ガイド』を参照してください。

関連項目：

- 18-7 ページ「機密データを格納する列の暗号化」
- 12-8 ページ「暗号化された表領域」

安全性の高い外部パスワード・ストアの作成

アプリケーションでデータベースに接続するためにパスワード資格証明が使用される大規模なデプロイメントでは、この資格証明をクライアント側の Oracle ウォレットに格納できます。Oracle ウォレットは、認証および署名の資格証明を格納するための安全性の高いソフトウェア・コンテナです。

クライアント側の Oracle ウォレットにデータベース・パスワード資格証明を格納することで、ユーザー名とパスワードをアプリケーション・コード、バッチ・ジョブまたはスクリプトに埋め込む必要がなくなります。この結果、スクリプトやアプリケーション・コードに記述したパスワードが外部にさらされる危険性が低くなり、ユーザー名とパスワードを変更するたびにコードを変更する必要がないため、メンテナンスが簡素化されます。また、アプリケーション・コードを変更する必要がないため、これらのユーザー・アカウントのパスワード管理ポリシーをさらに簡単に規定できるようになります。

外部パスワード・ストアを使用するようにクライアントを構成すると、アプリケーションでは、次の構文を使用してパスワード認証を使用しているデータベースに接続できます。

```
CONNECT /@database_alias
```

この CONNECT 文では、データベース・ログイン資格証明を指定する必要はありません。かわりに、データベース・ログイン資格証明はクライアントのウォレットで検索されます。

関連項目： 安全性の高い外部パスワード・ストアを使用するようにクライアントを構成する方法、および外部パスワード・ストア内の資格証明を管理する方法については、『Oracle Database Advanced Security 管理者ガイド』を参照してください。

Oracle Database のサンプル・スキーマのインストール

Oracle Database の配布媒体には、各種の SQL ファイルが格納されています。この SQL ファイルでは、システムの試用、SQL の学習、追加の表、ビュー、シノニムの作成などが可能です。

Oracle Database には、Oracle Database 機能の理解に役立つサンプル・スキーマが含まれています。Oracle Database に関するすべてのマニュアルと研修資料は、更新時にサンプル・スキーマ環境に変換されます。

サンプル・スキーマは Database Configuration Assistant によって自動的にインストールするか、手動でインストールできます。スキーマとインストール手順の詳細は、『Oracle Database サンプル・スキーマ』を参照してください。

データベースの削除

データベースの削除には、データベースのデータファイル、REDO ログ・ファイル、制御ファイルおよび初期化パラメータ・ファイルの削除も含まれます。DROP DATABASE 文では、すべての制御ファイルと制御ファイルに記述されている他のすべてのデータベース・ファイルが削除されます。DROP DATABASE 文を正しく使用するには、次の条件をすべて満たす必要があります。

- データベースをマウントし、クローズしておくこと。
- データベースを共有モードではなく排他的にマウントしておくこと。
- データベースを RESTRICTED でマウントしておくこと。

この文の例を次に示します。

```
DROP DATABASE;
```

DROP DATABASE 文は、アーカイブ・ログ・ファイル、およびデータベースのコピーまたはバックアップには影響を与えません。これらのファイルを削除する場合は、Recovery Manager を使用することをお薦めします。データベースが RAW ディスク上にある場合、その RAW ディスクにある実際の特殊ファイルは削除されません。

Database Configuration Assistant を使用してデータベースを作成した場合は、データベースおよびファイルの削除にこのツールを使用できます。

データベースのデータ・ディクショナリ・ビュー

「パラメータ設定の表示」にリストしたビューの他に、次のビューを使用してデータベースの内容や構造に関する情報を参照できます。

ビュー	説明
DATABASE_PROPERTIES	永続的なデータベース・プロパティが表示されます。
GLOBAL_NAME	グローバル・データベース名が表示されます。
V$DATABASE	制御ファイル内のデータベース情報が表示されます。

3

起動と停止

この章の内容は次のとおりです。

関連項目： Oracle Real Application Clusters 環境に固有の追加情報は、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。

データベースの起動

データベースの起動時には、そのデータベースのインスタンスを作成し、データベースの状態を確認します。インスタンスは通常、データベースをマウントおよびオープンすることで起動します。これによって、有効なユーザーはデータベースに接続して通常のデータ・アクセス操作を実行できます。ここでは、他の起動方法についても説明します。

この項の内容は、次のとおりです。

データベースの起動方法

データベース・インスタンスは、SQL*Plus、Recovery Manager または Enterprise Manager を使用して起動できます。

SQL*Plus を使用したデータベースの起動

SQL*Plus セッションを起動し、管理者権限で Oracle Database に接続すると、`STARTUP` コマンドを発行できます。このマニュアルでは、SQL*Plus を使用するこの方法について詳細に説明します。

Recovery Manager を使用したデータベースの起動

Recovery Manager（RMAN）を使用して、`STARTUP` および `SHUTDOWN` コマンドを実行する方法もあります。この方法を選択するのは、Recovery Manager 環境で SQL*Plus を起動しない場合です。

> **関連項目：** Recovery Manager を使用したデータベースの起動方法の詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

Oracle Enterprise Manager を使用したデータベースの起動

Oracle Enterprise Manager（EM）は、起動や停止も含めたデータベースの管理に使用できます。EM は、GUI コンソール、エージェント、共有サービスおよび Oracle のツール製品を組み合せて、Oracle 製品の管理のために統合された包括的なシステム管理プラットフォームを提供します。EM の Oracle データベース管理専用の部分である EM Database Control では、このマニュアルで説明されている機能を、コマンドライン操作ではなく GUI を使用して実行できます。

> **関連項目：**
>
> - 『Oracle Enterprise Manager 概要』
> - 『Oracle Enterprise Manager 基本インストレーションおよび構成』
> - 『Oracle Database 2 日でデータベース管理者』

ここからは、SQL*Plus を使用してデータベース・インスタンスを起動する方法について説明します。

初期化パラメータおよび起動の概要

データベースは、インスタンスを起動するために、サーバー・パラメータ・ファイル（SPFILE）またはテキスト形式の初期化パラメータ・ファイルからインスタンス構成パラメータ（初期化パラメータ）を読み込む必要があります。

SQL*Plus の STARTUP コマンドを発行すると、データベースは、プラットフォーム固有のデフォルトの場所にある SPFILE から初期化パラメータを読み込もうとします。SPFILE が見つからない場合は、テキスト形式の初期化パラメータ・ファイルを検索します。

注意： UNIX または Linux の場合、SPFILE およびテキスト形式の初期化パラメータ・ファイルが配置されるプラットフォーム固有のデフォルトの場所（ディレクトリ）は、次のとおりです。

```
$ORACLE_HOME/dbs
```

Windows NT および Windows 2000 の場合は、次のとおりです。

```
%ORACLE_HOME%¥database
```

Oracle Database は、プラットフォーム固有のデフォルトの位置にあるファイル名を次の順序で検査し、初期化パラメータ・ファイルを特定します。

1. spfile*$ORACLE_SID*.ora
2. spfile.ora
3. init*$ORACLE_SID*.ora

最初の 2 つのファイル名は SPFILE を表し、3 番目のファイル名はテキスト形式の初期化パラメータ・ファイルを表します。

注意： spfile.ora ファイルがこの検索パスに含まれているのは、Oracle Real Application Clusters 環境では、すべてのインスタンスの初期化パラメータ設定が 1 つのサーバー・パラメータ・ファイルに格納されるためです。サーバー・パラメータ・ファイルにはインスタンス固有の格納場所はありません。

Oracle Real Application Clusters 環境におけるサーバー・パラメータ・ファイルの詳細は、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。

ユーザー（または Database Configuration Assistant）が作成したサーバー・パラメータ・ファイルをテキスト形式の初期化パラメータ・ファイルで上書きする場合は、STARTUP コマンドの PFILE 句を指定すると、その初期化パラメータ・ファイルを識別できます。

```
STARTUP PFILE = /u01/oracle/dbs/init.ora
```

デフォルト以外のサーバー・パラメータ・ファイルを使用した起動

デフォルト以外のサーバー・パラメータ・ファイル（SPFILE）とは、デフォルト以外の場所にある SPFILE のことです。通常は、デフォルト以外の SPFILE を使用してインスタンスを起動する必要はありません。ただし、必要な場合は、この PFILE 句を使用して、次のようにデフォルト以外のサーバー・パラメータ・ファイルでインスタンスを起動できます。

1. SPFILE パラメータのみを記述した 1 行のテキスト形式の初期化パラメータ・ファイルを作成します。パラメータの値には、デフォルト以外のサーバー・パラメータ・ファイルの場所を指定します。

   たとえば、次のパラメータのみを記述したテキスト形式の初期化パラメータ・ファイル /u01/oracle/dbs/spf_init.ora を作成します。

   ```
   SPFILE = /u01/oracle/dbs/test_spfile.ora
   ```

注意： 従来のテキスト形式の初期化パラメータ・ファイルで、サーバー・パラメータ・ファイルを設定するために IFILE 初期化パラメータを使用することはできません。この場合は、必ず SPFILE 初期化パラメータを使用してください。

---

2. この初期化パラメータ・ファイルを指定して、インスタンスを起動します。

```
STARTUP PFILE = /u01/oracle/dbs/spf_init.ora
```

この SPFILE は、データベース・サーバーが稼働しているマシン上に存在している必要があります。したがって、前述の方法は、SPFILE を使用してデータベースを起動する手段をクライアント・マシンにも提供します。また、クライアント・マシンで、クライアント側の初期化パラメータ・ファイルをメンテナンスする必要はありません。クライアント・マシンが SPFILE パラメータを含む初期化パラメータ・ファイルを読み込むと、サーバーにその値が渡され、指定された SPFILE が読み込まれます。

UNIX および Linux プラットフォームでは、SPFILE がデフォルトの場所にない場合、SPFILE へのシンボリック・リンクを作成して、そのシンボリック・リンクをデフォルトの場所に配置することもできます。

PFILE および SPFILE のデフォルトの名前と場所の詳細は、2-34 ページの表 2-3 を参照してください。

初期化ファイルと自動ストレージ管理

通常、自動ストレージ管理（ASM）を使用しているデータベースには、デフォルト以外の SPFILE があります。Database Configuration Assistant（DBCA）を使用して ASM を使用するようにデータベースを構成すると、DBCA によって ASM ディスク・グループ内にデータベース・インスタンスの SPFILE が作成され、ローカル・ファイル・システム内のデフォルトの場所に、その SPFILE を指し示すテキスト形式の初期化パラメータが作成されます。

> **関連項目：** 初期化パラメータ、初期化パラメータ・ファイルおよびサーバー・パラメータ・ファイルの詳細は、第 2 章「Oracle Database の作成および構成」を参照してください。

インスタンス起動の準備

SQL*Plus を使用してデータベース・インスタンスを起動する前に、次の準備手順を実行する必要があります。

1. 環境変数が目的の Oracle インスタンスに接続するように設定されていることを確認します。詳細は、1-8 ページの「データベースに対するコマンドと SQL の発行」を参照してください。
2. データベースに接続せずに、SQL*Plus を起動します。

```
SQLPLUS /NOLOG
```

3. SYSDBA として Oracle Database に接続します。

```
CONNECT username AS SYSDBA
```

データベースに接続され、データベース・インスタンスを起動する準備が完了します。

> **関連項目：** CONNECT、STARTUP および SHUTDOWN コマンドの説明と構文は、『SQL*Plus ユーザーズ・ガイドおよびリファレンス』を参照してください。

インスタンスの起動

Oracle Database インスタンスを起動するには、SQL*Plus の STARTUP コマンドを使用します。インスタンスは、次のような様々なモードで起動できます。

- インスタンスを起動するが、データベースはマウントしないモード。このモードで起動すると、データベースにはアクセスできません。通常、このモードで起動するのは、データベースの作成時または制御ファイルの再作成時のみです。
- インスタンスを起動し、データベースをマウントするが、クローズしたままにするモード。この状態では、一部の DBA アクティビティは可能ですが、データベースへの汎用アクセスはできません。
- インスタンスを起動し、データベースをマウントしてオープンするモード。この操作を非制限モードで実行してユーザー全員にアクセスを許可するか、制限モードで実行して DBA のみにアクセスを許可できます。
- 起動時または停止時に問題が発生した後にインスタンスを強制的に起動するモード、またはインスタンスの起動直後に完全メディア・リカバリを開始するモード。

注意： 共有サーバー・プロセスを介してデータベースに接続してデータベースのインスタンスを起動することはできません。

次の例では、DBA がインスタンスを起動できるいくつかの状態を具体的に説明します。STARTUP コマンドの句を組み合せるときは、いくつかの制限が適用されます。

注意： 制御ファイル、データベース・ファイルまたは REDO ログ・ファイルが使用できない場合は、インスタンスの起動で問題が発生することがあります。データベースをマウントするときに CONTROL_FILES 初期化パラメータで指定された 1 つ以上のファイルが存在しない場合、またはオープンできない場合は、Oracle Database によって警告メッセージが返され、データベースはマウントされません。データベースをオープンするときに、1 つ以上のデータファイルまたは REDO ログ・ファイルを使用できない場合、またはオープンできない場合、データベースは警告メッセージを返し、データベースをオープンしません。

関連項目： STARTUP コマンドの句を組み合せるときに適用される制限の詳細は、『SQL*Plus ユーザーズ・ガイドおよびリファレンス』を参照してください。

インスタンスを起動し、データベースをマウントしてオープンする方法

通常のデータベース操作とは、インスタンスを起動し、データベースをマウントおよびオープンすることを意味します。このモードでは、有効なユーザーがデータベースに接続して、データ・アクセス操作を実行できます。

次のコマンドは、インスタンスを起動し、デフォルトの場所から初期化パラメータを読み込んで、データベースをマウントおよびオープンします（必要に応じて、PFILE 句を指定できます）。

```
STARTUP
```

インスタンスを起動するが、データベースをマウントしない方法

インスタンスは、データベースをマウントしなくても起動できます。通常、この方法で起動するのはデータベースの作成時のみです。STARTUP コマンドで、NOMOUNT 句を指定します。

```
STARTUP NOMOUNT
```

インスタンスを起動し、データベースをマウントする方法

インスタンスを起動し、データベースをオープンしないでマウントして、特定のメンテナンス操作を実行できます。たとえば、次のようなタスクの実行時は、データベースのマウントは必要ですが、オープンしてはなりません。

- REDO ログ・アーカイブ・オプションの有効化および無効化。詳細は、第 11 章「アーカイブ REDO ログの管理」を参照してください。
- 全データベース・リカバリの実行。詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

次のコマンドは、インスタンスを起動してデータベースをマウントしますが、データベースはクローズしたままです。

```
STARTUP MOUNT
```

起動時にインスタンスへのアクセスを制限する方法

インスタンスの使用を管理担当者にのみ許可し、一般データベース・ユーザーの使用を禁止するには、制限モードでインスタンスを起動して、オプションでデータベースをマウントおよびオープンします。次のいずれかのタスクを実行するときは、このインスタンス起動モードを使用してください。

- データのエクスポートまたはインポートを実行する場合
- SQL*Loader を使用してデータをロードする場合
- 一時的に一般ユーザーがデータを使用できないようにする場合
- 特定の移行またはアップグレード操作を実行する場合

通常、CREATE SESSION システム権限を持つすべてのユーザーは、オープンしているデータベースに接続できます。制限モードでデータベースをオープンすると、CREATE SESSION システム権限と RESTRICTED SESSION システム権限の両方を持つユーザーのみがデータベースにアクセスできます。したがって、DBA のみが RESTRICTED SESSION システム権限を持つようにしてください。また、インスタンスが制限モードのとき、DBA は Oracle Net リスナーを介してリモートでインスタンスにアクセスすることはできません。インスタンスが実行されているマシンからローカルでアクセスすることのみ可能です。

次のコマンドは、制限モードでインスタンスを起動します（データベースをマウントしてオープンします）。

```
STARTUP RESTRICT
```

RESTRICT 句は、MOUNT、NOMOUNT および OPEN の各句と組み合せて使用できます。

RESTRICTED SESSION 機能を無効にするには、ALTER SYSTEM 文を使用します。

```
ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

データベースを非制限モードでオープンし、後でアクセス制限が必要であると判明した場合は、ALTER SYSTEM 文を使用して制限できます。3-9 ページの「オープンしているデータベースへのアクセスを制限する方法」を参照してください。

> **関連項目：** ALTER SYSTEM 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

インスタンスを強制的に起動する方法

通常と異なる状況では、データベース・インスタンスを起動しようとしたときに、問題が発生することがあります。次の問題が発生している場合以外は、データベースを強制的に起動しないでください。

- 現行インスタンスを SHUTDOWN NORMAL、SHUTDOWN IMMEDIATE または SHUTDOWN TRANSACTIONAL コマンドで停止できない場合
- インスタンス起動時に問題が発生した場合

このような問題が発生した場合、STARTUP コマンドで FORCE 句を指定して新しいインスタンスを起動（必要に応じて、データベースをマウントおよびオープン）すると、通常は問題を解決できます。

```
STARTUP FORCE
```

インスタンスの実行中に STARTUP FORCE を使用すると、ABORT モードで停止した後に再起動します。この場合、Oracle Database 10*g* リリース 2 からは、メッセージ「Shutting down instance (abort)」に続いて「Starting ORACLE instance (normal)」がアラート・ログに表示されます（以前のバージョンのデータベースでは、アラート・ログに表示されるメッセージは、「Starting ORACLE instance (force)」のみでした）。

> **関連項目：** 現行インスタンスの強制終了による影響の詳細は、3-11 ページの「ABORT 句による停止」を参照してください。

インスタンスを起動し、データベースをマウントして、完全メディア・リカバリを開始する方法

メディア・リカバリが必要な場合は、STARTUP コマンドで RECOVER 句を指定すると、インスタンスを起動し、データベースをインスタンスにマウントして、リカバリ処理を自動的に開始できます。

```
STARTUP OPEN RECOVER
```

必要ない場合にリカバリを実行しようとすると、Oracle Database によってエラー・メッセージが表示されます。

オペレーティング・システム起動時にデータベースを自動的に起動する方法

多くのサイトでは、システムの起動直後に 1 つ以上の Oracle Database インスタンスとデータベースを自動的に起動する手順を使用しています。そのための手順は、オペレーティング・システムによって異なります。自動起動の詳細は、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。

リモート・インスタンスを起動する方法

ローカルの Oracle Database サーバーが分散データベースの一部を構成している場合は、リモート・インスタンスとデータベースを起動できます。リモート・インスタンスの起動と停止の手順は、通信プロトコルとオペレーティング・システムによって大きく異なります。

データベースの可用性の変更

データベースの可用性を変更できます。可用性の変更は、メンテナンス上の理由やデータベースを読取り専用にするために行う場合があります。次の各項では、データベースの可用性を変更する方法について説明します。

- インスタンスにデータベースをマウントする方法
- クローズしているデータベースをオープンする方法
- データベースを読取り専用モードでオープンする方法
- オープンしているデータベースへのアクセスを制限する方法

インスタンスにデータベースをマウントする方法

特定の管理操作を実行する場合は、データベースを起動してインスタンスにマウントし、クローズしたままにする必要があります。そのためには、インスタンスを起動してデータベースをマウントします。

あらかじめ起動したインスタンスにデータベースをマウントするには、次のように、SQL 文 ALTER DATABASE で MOUNT 句を指定します。

```
ALTER DATABASE MOUNT;
```

> **関連項目：** データベースをマウントし、クローズしておくことが必要な操作（およびインスタンスの起動とデータベースのマウントを一度に実行する手順）の詳細は、3-6 ページの「インスタンスを起動し、データベースをマウントする方法」を参照してください。

クローズしているデータベースをオープンする方法

データベースをオープンすることによって、マウントされ、クローズしているデータベースを一般的な用途のために使用可能にできます。マウントされたデータベースをオープンするには、ALTER DATABASE 文で OPEN 句を使用します。

```
ALTER DATABASE OPEN;
```

この文の実行後は、CREATE SESSION システム権限を持つ有効な Oracle Database ユーザーであれば、誰でもデータベースに接続できます。

データベースを読取り専用モードでオープンする方法

データベースを読取り専用モードでオープンすると、オープンしたデータベースを問い合せることができますが、その間にデータの内容がオンラインで変更されることはありません。これにより、データファイルと REDO ログ・ファイルにデータが書き込まれないことが保証されますが、データベース・リカバリや、REDO を生成せずにデータベースの状態を変更する操作が制限されることはありません。たとえば、データファイルをオフラインとオンラインの間で切り替えてもデータの内容には影響しないため、このような切替えは可能です。

ディスク・ソートの実行時など、読取り専用モードでデータベースを問い合せるときに一時表領域を使用する場合、問合せの発行者には、デフォルト一時表領域としてローカル管理表領域が割り当てられている必要があります。割り当てられていない場合は、問合せが失敗します。この操作については、12-11 ページの「ローカル管理の一時表領域の作成」を参照してください。

次の文では、データベースが読取り専用モードでオープンします。

```
ALTER DATABASE OPEN READ ONLY;
```

また、次のように読取り / 書込みモードでデータベースをオープンすることもできます。

```
ALTER DATABASE OPEN READ WRITE;
```

ただし、読取り / 書込みはデフォルトのモードです。

注意： RESETLOGS 句と READ ONLY 句は併用できません。

関連項目： ALTER DATABASE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

オープンしているデータベースへのアクセスを制限する方法

インスタンスを制限モードにするには、SQL 文の ALTER SYSTEM で ENABLE RESTRICTED SESSION 句を指定します。制限モードの場合、インスタンスにアクセスできるのは、管理権限を持つユーザーのみです。インスタンスを制限モードにした場合は、管理タスクを実行する前に現行のユーザー・セッションをすべて停止する必要があります。

インスタンスの制限モードを解除するには、ALTER SYSTEM で DISABLE RESTRICTED SESSION 句を指定します。

関連項目：

- ユーザー・セッションを停止する方法は、4-23 ページの「セッションの停止」を参照してください。
- インスタンスを制限モードにする理由は、3-6 ページの「起動時にインスタンスへのアクセスを制限する方法」を参照してください。

データベースの停止

データベースを停止するには、SQL*Plus の SHUTDOWN コマンドを使用します。停止が完了するまで、データベース停止を開始したセッションに制御が戻りません。停止処理の進行中に接続しようとするユーザーは、次のようなメッセージを受け取ります。

```
ORA-01090: シャットダウン処理中 - 接続はできません
```

注意： 共有サーバー・プロセスを介してデータベースに接続している場合は、データベースを停止できません。

データベースとインスタンスを停止するには、最初に SYSOPER または SYSDBA として接続する必要があります。データベースにはいくつかの停止モードがあります。次の項では、各モードについて説明します。

- NORMAL 句による停止
- IMMEDIATE 句による停止
- TRANSACTIONAL 句による停止
- ABORT 句による停止

一部の停止モードでは、データベースを実際に停止する前に、特定のイベント（トランザクションの完了またはユーザーによる切断など）の発生を待機します。これらのイベントに対するタイムアウト間隔は 1 時間です。このタイムアウト動作については、この後の項で説明します。

- 停止タイムアウトおよび強制終了

NORMAL 句による停止

データベースを通常の状況で停止するには、次のように SHUTDOWN コマンドで NORMAL 句を指定します。

```
SHUTDOWN NORMAL
```

NORMAL 句はオプションです。句が指定されていない場合、これがデフォルトの停止方法になるためです。

通常のデータベース停止では、次のように処理が進みます。

- 文が発行された後は、新しい接続は許可されません。
- データベースは、停止する前に、現在データベースに接続しているすべてのユーザーによるデータベースからの切断を待機します。

次にデータベースを起動するときに、インスタンス・リカバリ手順は必要ありません。

IMMEDIATE 句による停止

データベースの即時停止は、次のような状況のときにのみ使用します。

- 自動バックアップおよび無人バックアップを開始する場合
- 電源が間もなく停止する場合
- データベースやデータベース・アプリケーションの一部が異常に動作している場合で、ユーザーにログオフを依頼できない場合、またはユーザーがログオフできない場合

データベースを即時に停止するには、SHUTDOWN コマンドで IMMEDIATE 句を指定します。

```
SHUTDOWN IMMEDIATE
```

このデータベース停止では、次のように処理が進みます。

- 文が発行された後は、新しい接続や新しいトランザクションの開始は許可されません。
- コミットされていないトランザクションはすべてロールバックされます。(コミットされていない大規模なトランザクションが存在する場合、この停止方法では、その名前に反してただちに完了しないことがあります。)
- Oracle Database は、現在データベースに接続しているユーザーが切断されるのを待機しません。アクティブなトランザクションは暗黙的にロールバックされ、接続中のユーザーはすべて切断されます。

次にデータベースを起動するときに、インスタンス・リカバリ手順は必要ありません。

TRANSACTIONAL 句による停止

アクティブなトランザクションを完了してから、予定どおりにインスタンスを停止する場合は、SHUTDOWN コマンドで TRANSACTIONAL 句を指定します。

```
SHUTDOWN TRANSACTIONAL
```

このデータベース停止では、次のように処理が進みます。

- 文が発行された後は、新しい接続や新しいトランザクションの開始は許可されません。
- すべてのトランザクションが完了すると、まだインスタンスに接続されているすべてのクライアントが切断されます。
- この時点で、インスタンスは SHUTDOWN IMMEDIATE 文を発行した場合と同じように停止します。

次にデータベースを起動するときに、インスタンス・リカバリ手順は必要ありません。

TRANSACTIONAL オプションによる停止では、クライアントの作業内容が失われずに済み、すべてのユーザーがログオフする必要がなくなります。

ABORT 句による停止

データベース・インスタンスを強制終了することによって、データベースをただちに停止できます。このタイプの停止は、次のような状況でのみ実行してください。

データベースまたはデータベース・アプリケーションの一部が異常に動作している場合で、他のタイプの停止がどれも機能しないとき

- データベースを即時停止する必要がある場合（たとえば、電源停止が 1 分以内に起こることがわかっている場合）
- データベース・インスタンスを起動するときに問題が発生した場合

トランザクションとユーザーの接続を強制終了してデータベースを停止する場合は、次のように SHUTDOWN コマンドで ABORT 句を指定します。

```
SHUTDOWN ABORT
```

このデータベース停止では、次のように処理が進みます。

- 文が発行された後は、新しい接続や新しいトランザクションの開始は許可されません。
- Oracle Database によって処理されている現行のクライアント SQL 文はただちに終了します。
- コミットされていないトランザクションはロールバックされません。
- Oracle Database は、現在データベースに接続しているユーザーが切断されるのを待機しません。接続中のユーザーはすべて暗黙的に切断されます。

次にデータベースを起動するときには、インスタンス・リカバリの手順が必要になります。

停止タイムアウトおよび強制終了

ユーザーによる切断またはトランザクションの完了を待機する停止モードには、待機時間に制限があります。停止をブロックしているすべてのイベントが 1 時間以内に発生しない場合、停止コマンドは「ORA-01013: ユーザーによって現行の操作の取消しがリクエストされました」というメッセージを表示して強制終了されます。たとえば、[CTRL] キーを押しながら [C] キーを押して停止プロセスを中断した場合にも、このメッセージが表示されます。インスタンスの停止を中断しようとしないことをお薦めします。停止プロセスが完了してから、インスタンスを再起動してください。

ORA-01013 が発生した後は、インスタンスが予測できない状態になると考える必要があります。このため、SHUTDOWN コマンドを再発行して、停止プロセスを継続する必要があります。後続の SHUTDOWN コマンドが引き続き失敗する場合は、SHUTDOWN ABORT コマンドを発行してインスタンスを停止する必要があります。その後でインスタンスを再起動できます。

データベースの静止

データベースを、DBA によるトランザクション、問合せ、フェッチまたは PL/SQL 文の実行のみ許可された状態にすることが必要な場合があります。このような状態は、システム上で DBA 以外によるトランザクション、問合せ、フェッチまたは PL/SQL 文が実行されていないという意味で、**静止状態**と呼びます。

注意： 静止中のデータベースに関するこの項の説明では、DBA がユーザー SYS または SYSTEM として定義されています。DBA ロールを持つユーザーなどの他のユーザーには、ALTER SYSTEM QUIESCE DATABASE 文の発行や、データベース静止後の処理の継続は許可されていません。

データベースを静止状態にすることで、管理者は、それ以外の状態では安全に実行できない処理を実行できます。次の処理を実行できます。

- 同時ユーザー・トランザクションが同じオブジェクトにアクセスした場合に失敗する処理。たとえば、データベース表のスキーマの変更や、NOWAIT ロックが必要な既存表への列の追加などがこれに該当します。
- 途中で、同時ユーザー・トランザクションに望ましくない影響を与えるおそれがある処理。たとえば、最初に表をエクスポートし、削除してから、最後にインポートするというように、複数の手順で表を再編成する場合などがこれに該当します。表を削除してからインポートするまでの間に、同時ユーザーが表にアクセスしようとすると、状態が正確に表示されなくなります。

データベースの静止機能がない場合は、データベースを停止し、制限モードで再度オープンする必要があります。これは、特に 24 時間、365 日の可用性が必要なシステムにとっては重大な制限です。データベースの静止によって、ユーザーに対する遮断とデータベースの停止と再起動に伴う停止時間を最小限にできるため、制限が大幅に緩和されます。

データベースが静止中の場合、DBA 以外のセッションがアクティブにならないようにするために、データベース・リソース・マネージャの機能が使用されています。したがって、この文が有効な間に現行のリソース・プランを変更しようとすると、その処理はシステムが静止解除されるまでキューに待機します。データベース・リソース・マネージャの詳細は、第 25 章「Oracle Database Resource Manager を使用したリソース割当ての管理」を参照してください。

データベースの静止状態への変更

データベースを静止状態にするには、次の文を発行します。

```
ALTER SYSTEM QUIESCE RESTRICTED;
```

DBA 以外のアクティブなセッションは、非アクティブになるまで処理が継続されます。アクティブなセッションとは、トランザクション、問合せ、フェッチまたは PL/SQL 文を現在実行しているセッション、または現在なんらかの共有リソース（エンキューなど）を保持しているセッションです。非アクティブなセッションがアクティブになることはできません。たとえば、ユーザーが非アクティブなセッションを強制的にアクティブにしようとして SQL 問合せを発行すると、その問合せは停止したようになります。後でデータベースが静止解除されると、セッションが再開され、ブロックされていた処理が実行されます。

DBA 以外のセッションがすべて非アクティブになると、ALTER SYSTEM QUIESCE RESTRICTED 文が完了し、データベースは静止状態となります。Oracle Real Application Clusters 環境でこの文を発行すると、文を発行したインスタンスだけでなく、すべてのインスタンスが影響を受けます。

ALTER SYSTEM QUIESCE RESTRICTED 文は、アクティブなセッションが非アクティブになるまで、長時間待機する場合があります。V$BLOCKING_QUIESCE ビューを問い合せると、静止操作をブロックしているセッションを判別できます。このビューでは、1 つの列 SID（セッション ID）のみが返されます。これを V$SESSION と結合すると、次の例のように、セッションに関する詳細を取得できます。

```
select bl.sid, user, osuser, type, program
from v$blocking_quiesce bl, v$session se
where bl.sid = se.sid;
```

これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。

データベース静止の要求を中断した場合、またはアクティブなセッションがすべて静止する前にセッションが異常終了した場合は、Oracle Database によって、この文による部分的な影響がすべて自動的に取り消されます。

複数の Oracle Call Interface（OCI）の連続したフェッチによって問合せが実行されている場合、ALTER SYSTEM QUIESCE RESTRICTED 文はすべてのフェッチが完了するまで待機しません。現行のフェッチの完了のみを待機します。

専用サーバー接続の場合も、共有サーバー接続の場合も、この文の発行後、DBA 以外のユーザーがログインしようとすると、その処理はデータベース・リソース・マネージャによってすべてキューに送られ、進行しません。ユーザーにはログインが停止したように見えます。データベースが静止解除されると、ログインは再開されます。

文を発行したセッションが終了しても、データベースは静止状態のままです。DBA は、データベースを静止解除される文を明示的に発行するために、データベースにログインする必要があります。

注意： データベースが静止状態の場合でも、Oracle Database のバックグラウンド・プロセスが内部の目的のために更新を実行している可能性があるため、データベースが静止状態の場合はコールド・バックアップを実行できません。また、オンライン・データファイルのヘッダーは引き続きアクセス可能に見えます。このファイル・ヘッダーの状態は、データベースが正しく停止された場合とは異なります。ただし、データベースが静止状態の間でも、オンライン・バックアップ操作は実行できます。

通常操作へのシステムのリストア

次の文は、データベースを通常の操作にリストアさせます。

```
ALTER SYSTEM UNQUIESCE;
```

DBA 以外のアクティビティの処理がすべて許可されます。Oracle Real Application Clusters 環境では、データベースを静止した場合と同じセッションまたは同じインスタンスでこの文を発行する必要はありません。ALTER SYSTEM UNQUIESCE 文を発行したセッションが異常終了した場合、Oracle Database サーバーは静止解除を確実に完了します。

インスタンスの静止状態の表示

V$INSTANCE ビューの ACTIVE_STATE 列を問い合せると、インスタンスの現在の状態を表示できます。列の値は、次のいずれかになります。

- NORMAL: 通常の静止していない状態。
- QUIESCING: 静止途中の状態。DBA 以外の一部のセッションがアクティブな状態です。
- QUIESCED: 静止状態。DBA 以外のセッションは非アクティブで、許可されていない状態です。

データベースの一時停止と再開

ALTER SYSTEM SUSPEND 文は、データファイル（ファイル・ヘッダーとファイル・データ）および制御ファイルへの入出力（I/O）をすべて停止します。一時停止状態によって、I/O に干渉されずにデータベースのバックアップを作成できます。データベースを一時停止すると、実行中のすべての I/O 操作の完了が許可され、新しいデータベース・アクセスはキューに待機した状態になります。

SUSPEND コマンドは、インスタンスに固有ではありません。Oracle Real Application Clusters 環境では、あるシステムで SUSPEND コマンドを発行すると、内部ロッキング・メカニズムを通じてインスタンス間で停止要求が伝播し、特定のクラスタのアクティブ・インスタンスがすべて停止します。ただし、あるインスタンスの一時停止中に新しいインスタンスを起動すると、新しいインスタンスは一時停止されません。

通常のデータベース操作を再開するには、ALTER SYSTEM RESUME 文を使用します。SUSPEND コマンドと RESUME コマンドは、異なるインスタンスから発行できます。たとえば、インスタンス 1、2 および 3 の実行中に、インスタンス 1 から ALTER SYSTEM SUSPEND 文を発行した場合は、インスタンス 1、2 または 3 から同様に RESUME 文を発行できます。

一時停止 / 再開機能は、ディスクやファイルをミラー化してそのミラーを分割できるシステムで役立ち、バックアップとリストアの代替ソリューションを提供します。書込み中に既存のデータベースからミラー化されたディスクを分割できないシステムを使用している場合は、この一時停止 / 再開機能を使用すると容易に分割できます。

一時停止したデータベースのコピーにはコミット前の更新が含まれるため、一時停止 / 再開機能は通常の停止操作の簡易版ではありません。

注意： 表領域をホット・バックアップ・モードに設定する代替手段として ALTER SYSTEM SUSPEND 文を使用しないでください。データベースの一時停止操作ではなく、ALTER TABLESPACE BEGIN BACKUP 文を使用してください。

次の文は、ALTER SYSTEM SUSPEND/RESUME の使用方法を示しています。データベースの状態を確認するために、V$INSTANCE ビューを問い合せています。

```
SQL> ALTER SYSTEM SUSPEND;
System altered
SQL> SELECT DATABASE_STATUS FROM V$INSTANCE;
DATABASE_STATUS
---------
SUSPENDED

SQL> ALTER SYSTEM RESUME;
System altered
SQL> SELECT DATABASE_STATUS FROM V$INSTANCE;
DATABASE_STATUS
---------
ACTIVE
```

関連項目： データベースの一時停止 / 再開機能を使用してデータベースをバックアップする方法の詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

4

プロセスの管理

この章の内容は次のとおりです。

専用サーバー・プロセスと共有サーバー・プロセスの概要

Oracle Database では、インスタンスに接続されているユーザー・プロセスの要求を処理するために、サーバー・プロセスが作成されます。サーバー・プロセスには、次の 2 つのプロセスがあります。

- **専用サーバー・プロセス**。単一のユーザー・プロセスのみを処理します。
- **共有サーバー・プロセス**。複数のユーザー・プロセスを処理できます。

データベースでは、専用サーバー・プロセスは常に使用可能な状態ですが、**共有サーバー**は、1 つ以上の初期化パラメータを特別に設定して、構成および使用可能にする必要があります。

専用サーバー・プロセス

図 4-1「Oracle Database の専用サーバー・プロセス」は、専用サーバー・プロセスの動作の仕組みを示しています。この図では、専用サーバー・プロセスを介して、2 つのユーザー・プロセスがデータベースに接続されています。

一般的には、共有サーバーを使用し、**ディスパッチャ**を介して接続することをお薦めします。これは、図 4-2「Oracle Database の共有サーバー・プロセス」に図示されています。共有サーバー・プロセスは、実行中のインスタンスに必要なプロセスの数を少なくすることができるため、効率が向上します。

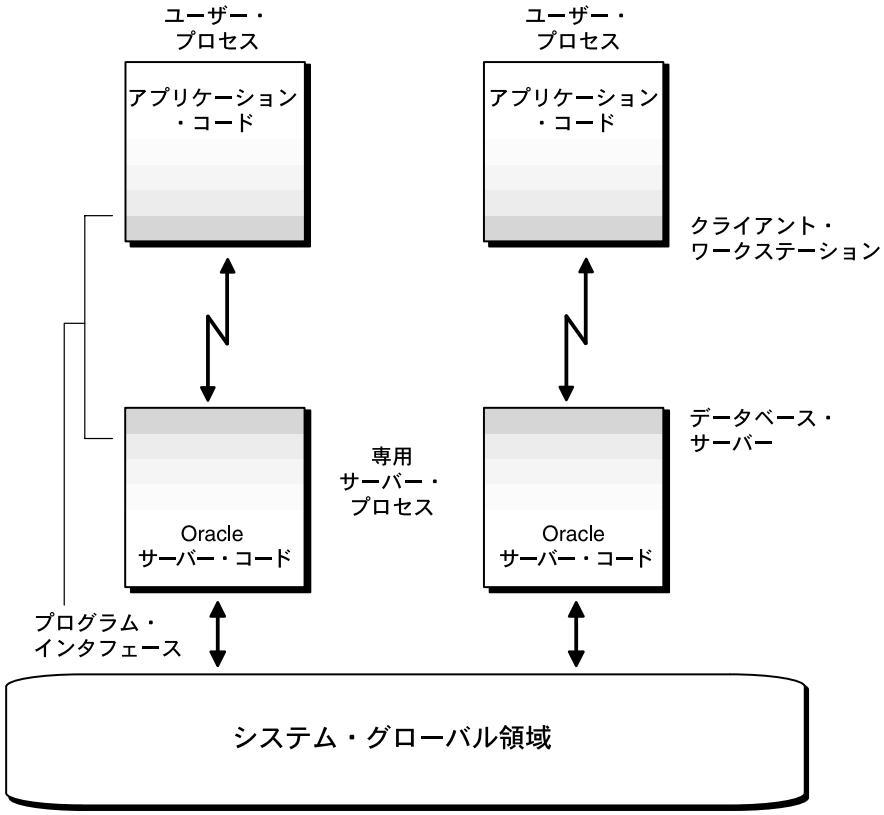
ただし、次の状況では、ユーザーと管理者は、専用サーバー・プロセスを使用して明示的にインスタンスに接続する必要があります。

- バッチ・ジョブを実行する場合（たとえば、ジョブがサーバー・プロセスに対して持つアイドル時間がほとんどないか、まったくない場合）
- Recovery Manager（RMAN）を使用して、データベースをバックアップ、リストアまたはリカバリする場合

Oracle Database が共有サーバー用に構成されている場合に専用サーバー接続を要求するには、専用サーバーを使用するように構成されているネット・サービス名を使用して接続する必要があります。具体的に言うと、ネット・サービス名の接続記述子に `SERVER=DEDICATED` 句を含めます。

> **関連項目：** 専用サーバー接続を要求する方法の詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

図 4-1 Oracle Database の専用サーバー・プロセス

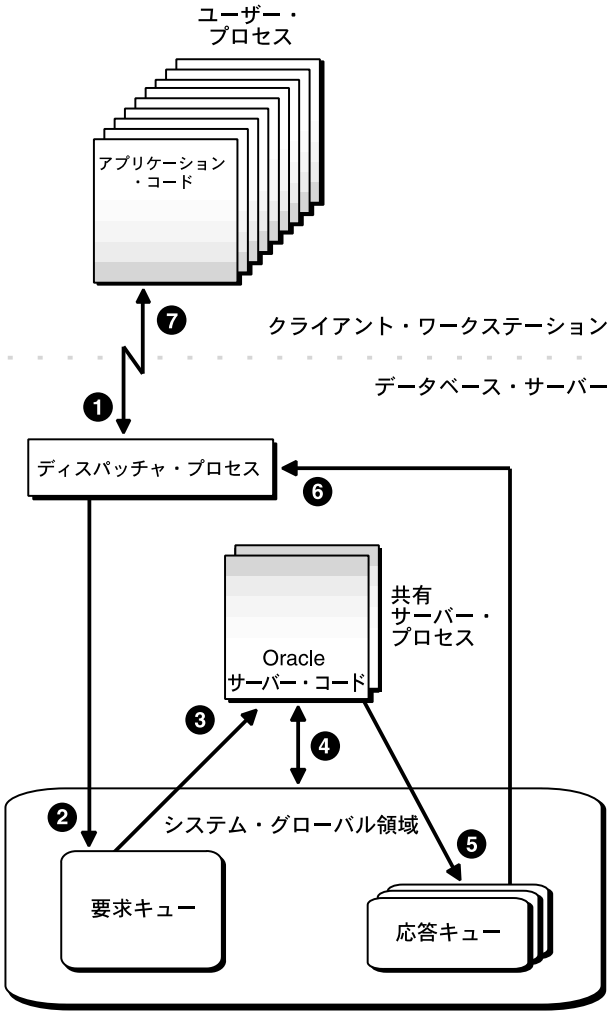


共有サーバー・プロセス

専用サーバー・プロセスを使用した注文入力システムを例にとって考えてみます。顧客が注文受付に電話で商品を注文すると、担当者はその注文をデータベースに入力します。取引のほどんどの間、担当者は顧客と電話で話しています。この間、サーバー・プロセスは必要ないため、担当者のユーザー・プロセス専用のサーバー・プロセスはアイドル状態のままです。アイドル状態のサーバー・プロセスはシステム・リソースを保持しているため、他の担当者が注文を入力する際にシステムのパフォーマンスが低下します。

共有サーバー・アーキテクチャでは、接続ごとに専用サーバー・プロセスは必要ありません(図 4-2 を参照)。

図 4-2 Oracle Database の共有サーバー・プロセス



共有サーバー構成では、クライアントのユーザー・プロセスはディスパッチャに接続します。このディスパッチャには、同時に複数のクライアント接続をサポートする機能があります。各クライアント接続は**バーチャル・サーキット**にバインドされます。バーチャル・サーキットとは、ディスパッチャがクライアントのデータベース接続要求と応答に使用する共有メモリーの一部です。ディスパッチャは、要求が到着すると、バーチャル・サーキットを共通キューに入れます。

アイドル状態の共有サーバー・プロセスは、共通キューからバーチャル・サーキットを選択して要求を処理し、そのバーチャル・サーキットを解放して、共通キューから別のバーチャル・サーキットを取り出します。このアプローチでは、小さいサーバー・プロセス・プールで大量のクライアントを処理することが可能です。専用サーバー・モデルと比較した共有サーバー・

アーキテクチャの大きな利点は、システム・リソースが少なくて済むため、ユーザー数の増加に対応できることです。

リソース管理を容易にするために、共有サーバーを**接続プーリング**に対応するように構成できます。接続プーリングを使用すると、データベース・サーバーでプロトコル接続のタイムアウトを実行し、これらの接続をアクティブ・セッションの処理に使用できるようにすることで、1つのディスパッチャでサポートするユーザー数が増加します。さらに、共有サーバーは**セッションの多重化**に対応するよう構成できます。セッションの多重化では、オペレーティング・システムのリソースを節約するために、複数のセッションが単一のネットワーク接続を介して転送されるように結合されます。

共有サーバー・アーキテクチャには、Oracle Net Services が必要です。共有サーバーを使用するユーザー・プロセスは、Oracle Database インスタンスと同じマシン上にある場合でも、必ず Oracle Net Services を介して接続してください。

> **関連項目：** 共有サーバーの詳細は、接続プーリングやセッションの多重化などの機能も含めて、『Oracle Database Net Services 管理者ガイド』を参照してください。

データベース常駐接続プーリングの概要

データベース常駐接続プーリング（DRCP）は、データベース接続を取得し、比較的短時間の処理を実行した後、データベース接続を解放するような一般的な Web アプリケーション使用に対して、データベース・サーバーの接続プールを提供します。DRCP は専用サーバーをプールします。**プール・サーバー**は、サーバー・フォアグラウンド・プロセスとデータベース・セッションの組合せに相当します。

DRCP は、中間層プロセス内のスレッド間で接続を共有する中間層接続プールを補完します。また、DRCP を使用すると、同じ中間層ホスト上の中間層プロセス間、および異なる中間層ホスト上の中間層プロセス間でもデータベース接続を共有できます。この結果、大量のクライアント接続をサポートするために必要となる基本データベース・リソースが大幅に減少するため、データベース層のメモリー・フットプリントが縮小し、中間層とデータベース層の両方のスケーラビリティが向上します。すぐに使用できるサーバーのプールを保持することで、クライアント接続の作成と切断のコストが削減されるという利点もあります。

DRCP は、中間層接続プーリングを実行できない（PHP/Apache などの）マルチプロセスのシングル・スレッド・アプリケーション・サーバーを含むアーキテクチャに特に適しています。データベースは、DRCP を使用すると同時接続の数を数万まで増やすことができます。

> **関連項目：**
>
> - 『Oracle Call Interface プログラマーズ・ガイド』
> - 『Oracle Database 概要』

データベース常駐接続プーリングを使用する場合

データベース常駐接続プーリングは、複数のクライアントがデータベースにアクセスする場合で、次のいずれかに該当する場合に有効です。

- 大量のクライアント接続を最小限のメモリー使用でサポートする必要がある場合
- 複数の類似クライアント・アプリケーションがあり、セッションの共有または再利用が可能な場合

  類似アプリケーションとは、同じデータベース資格証明で接続し、同じスキーマを使用するアプリケーションです。
- クライアント・アプリケーションでデータベース接続を取得し、比較的短時間の処理を実行した後、データベース接続を解放する場合
- クライアント要求にまたがるセッション・アフィニティが必要ない場合
- クライアント側に複数のプロセスおよび複数のホストが存在する場合

データベース常駐接続プーリングの利点

データベース常駐接続プーリングの使用には次の利点があります。

- 複数の中間層クライアント・アプリケーション間でリソースを共有できます。
- リソース使用量が減少するため、データベースおよびアプリケーションのスケーラビリティが向上します。

専用サーバー、共有サーバーおよびデータベース常駐接続プーリングの相違点

表 4-1 に、専用サーバー、共有サーバーおよびデータベース常駐接続プーリングの相違点を示します。

表 4-1 専用サーバー、共有サーバーおよびデータベース常駐接続プーリングの相違点

専用サーバー	共有サーバー	データベース常駐接続プーリング
クライアント要求を受け取ると、クライアント用に新規サーバー・プロセスとセッションが作成されます。	クライアントから最初の要求を受け取ると、ディスパッチャ・プロセスによって要求が共通キューに入れられます。要求は使用可能な共有サーバー・プロセスによって取り出されます。その後のクライアントと共有サーバー・プロセス間の通信はディスパッチャ・プロセスによって管理されます。	クライアントから最初の要求を受け取ると、接続ブローカによって使用可能なプール・サーバーが選択され、そのプール・サーバーにクライアント接続が渡されます。 使用可能なプール・サーバーがない場合は、接続ブローカによって作成されます。プールが最大サイズに達した場合、クライアント要求は、プール・サーバーが使用可能になるまで待機キューに入れられます。
データベース・リソースを解放すると、セッションおよびサーバー・プロセスが終了します。	データベース・リソースを解放すると、セッションが終了します。	データベース・リソースを解放すると、プール・サーバーがプールに解放されます。
メモリー要件は、サーバー・プロセスとセッションの数に比例します。クライアントごとに 1 つのサーバーと 1 つのセッションが存在します。	メモリー要件は、共有サーバーとセッションの合計に比例します。クライアントごとに 1 つのセッションが存在します。	メモリー要件は、プール・サーバーとそのセッションの数に比例します。プール・サーバーごとに 1 つのセッションが存在します。
セッション・メモリーは PGA から割り当てられます。	セッション・メモリーは SGA から割り当てられます。	セッション・メモリーは PGA から割り当てられます。

専用サーバー、共有サーバーおよびデータベース常駐接続プーリングのメモリー使用量の例

各セッションに必要なメモリーが 400KB、各サーバー・プロセスに必要なメモリーが 4MB であるアプリケーションについて考えます。プール・サイズは 100 で、使用される共有サーバーの数は 100 です。

5000 のクライアント接続がある場合、各構成で使用されるメモリーは次のようになります。

- 専用サーバー

  使用メモリー = 5000 X (400KB + 4MB) = 22GB

- 共有サーバー

  使用メモリー = 5000 X 400KB + 100 X 4MB = 2.5GB

  2.5GB のうち 2GB は SGA から割り当てられます。

- データベース常駐接続プーリング

  使用メモリー = 100 X (400KB + 4MB) + (5000 X 35KB)= 615MB

データベース常駐接続プーリングの使用に関する制限事項

プール・サーバーを使用して次の処理を実行することはできません。

- データベースの停止
- データベース常駐接続プールの停止
- 接続済ユーザーのパスワードの変更
- データベース常駐接続プールへの共有データベース・リンクを使用した接続
- 暗号化、証明書など、アドバンスト・セキュリティ・オプション（ASO）のオプションの使用
- OCI_MIGRATE オプションを使用した直接的な、または OCIConnectionPool を介した間接的なサーバー側での移行可能セッションの使用

プール内のデータベース・ユーザーに関係する DDL 文の実行には注意が必要です。これは、プール内の DDL 前のセッションを DDL 後にクライアントに渡すことができるためです。たとえば、ユーザーを削除する場合は、プール内にそのユーザーのセッションが存在していないこと、およびそのユーザーとして認証されたブローカへの接続が存在していないことを確認してください。

明示的なロールが使用可能であり、プールに解放されるセッションは、後でデフォルトのログイン・ロールを必要とする（同じユーザーの）接続に渡される可能性があります。明示的なロールのあるセッションは解放せずに、かわりにセッションを終了してください。

Oracle Database の共有サーバー構成

共有メモリー・リソースは、実行時に共有サーバーを使用可能にできるように事前に構成されています。初期化パラメータ・ファイルにパラメータを指定して共有メモリー・リソースを構成する必要はありませんが、さらに環境にあわせて構成できます。ディスパッチャおよび共有サーバー・プロセス（共有サーバー）は、ALTER SYSTEM 文を使用して動的に起動できます。

この項では、共有サーバーを使用可能にする方法、および共有サーバーの初期化パラメータを設定または変更する方法を説明します。この章の内容は、次のとおりです。

- 共有サーバー用初期化パラメータ
- 共有サーバーの使用可能化
- ディスパッチャの構成
- 共有サーバーのデータ・ディクショナリ・ビュー

> **関連項目：** ALTER SYSTEM 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

共有サーバー用初期化パラメータ

共有サーバー操作を制御する初期化パラメータは、次のとおりです。

- SHARED_SERVERS: 起動する初期共有サーバー数および最低限保持する共有サーバー数を指定します。共有サーバーを使用するための必須パラメータはこのパラメータのみです。
- MAX_SHARED_SERVERS: 同時に実行可能な共有サーバーの最大数を指定します。
- SHARED_SERVER_SESSIONS: 同時に実行可能な共有サーバー・ユーザー・セッションの合計数を指定します。このパラメータを設定すると、専用サーバー用のユーザー・セッションを予約できます。
- DISPATCHERS: 共有サーバー・アーキテクチャのディスパッチャ・プロセスを構成します。
- MAX_DISPATCHERS: 同時に実行可能なディスパッチャ・プロセスの最大数を指定します。このパラメータは現在は無視してかまいません。将来のリリースで、同時接続数に基づいてディスパッチャ数が自動調整されるときに使用されます。

- CIRCUITS: 受信および発信用のネットワーク・セッションに使用可能なバーチャル・サーキットの合計数を指定します。

  関連項目： これらの初期化パラメータの詳細は、『Oracle Database リファレンス』を参照してください。

共有サーバーの使用可能化

共有サーバーを使用可能にするには、SHARED_SERVERS 初期化パラメータに 0（ゼロ）より大きい値を設定します。他の共有サーバー初期化パラメータの設定は不要です。共有サーバーが動作するには最低 1 つのディスパッチャが必要であるため、構成されていない場合でも、ディスパッチャは起動されます。ディスパッチャについては、4-10 ページの「ディスパッチャの構成」を参照してください。

共有サーバーを動的に起動するには、ALTER SYSTEM 文を使用して SHARED_SERVERS パラメータに 0（ゼロ）以外の値を設定するか、またはデータベースの起動時に SHARED_SERVERS を初期化パラメータ・ファイルに組み込みます。SHARED_SERVERS が初期化パラメータ・ファイルに組み込まれていない場合、または値が 0（ゼロ）に設定されている場合、共有サーバーはデータベースの起動時に使用可能になりません。

注意： 下位互換性のために、SHARED_SERVERS がデータベースの起動時に初期化パラメータ・ファイルに組み込まれていない場合でも、DISPATCHERS が組み込まれ、少なくとも 1 つのディスパッチャが指定されていると、共有サーバーは使用可能になります。この場合、SHARED_SERVERS のデフォルトは 1 です。

ただし、SHARED_SERVERS と DISPATCHERS のいずれも初期化ファイルに組み込まれていない場合は、インスタンスが起動された後に DISPATCHERS パラメータを単に変更しても共有サーバーを起動することはできません。共有サーバーを起動するには、SHARED_SERVERS を 0（ゼロ）以外の値に変更する必要があります。

SHARED_SERVERS の値の決定

SHARED_SERVERS 初期化パラメータは、インスタンスの起動時に作成する共有サーバーの最小数を指定します。インスタンスの起動後は、既存の共有サーバーがビジーになる頻度と要求キューの長さに基づいて、Oracle Database が共有サーバーの数を動的に調整します。

標準的なシステムでは、共有サーバーの数は、10 接続ごとに共有サーバー 1 つという割合で安定します。OLTP アプリケーションでは、要求率が低い場合、または要求に対してサーバー使用の比率が低い場合は、接続数 / サーバー数の割合が高くなります。対照的に、要求率が高いか、要求に対してサーバー使用の比率が大きいアプリケーションの場合は、接続数 / サーバー数の割合が低くなります。

PMON（プロセス・モニター）バックグラウンド・プロセスは、SHARED_SERVERS で指定された値を下回る状態まで共有サーバーを終了させることはできません。したがって、このパラメータを使用すると、偶発的な負荷の変動のために PMON が共有サーバーを終了および再起動することがなくなるため、負荷が安定し、システムへの過負荷を最小にできます。

システムの平均的な負荷がわかっている場合は、SHARED_SERVERS を最適な値に設定できます。次に、このパラメータの使用方法の例を示します。

1000 人の従業員が勤務するテレマーケティング・センターでデータベースが使用されているとします。従業員は、平均して勤務時間の 90% を顧客との電話応対に費やし、レコードの検索と更新には 10% のみ費やしています。従業員が顧客と電話をしている間に共有サーバーが終了し、データベースにアクセスしたときに再び起動することがないように、DBA は最適な共有サーバー数として 100 を指定しています。

ただし、すべての勤務時間帯で同じ数の従業員が勤務しているわけではありません。夜間の時間帯は 200 人の従業員で十分です。SHARED_SERVERS は動的なパラメータであるため、DBA は、夜間の共有サーバーの数を 20 に減らし、バッチ・ジョブなどの他のタスクにリソースが解放されるようにします。

共有サーバー・プロセス数の削減

最低限アクティブに保つ必要がある共有サーバーの数を削減するには、SHARED_SERVERS パラメータを小さい値に動的に設定します。設定後は、共有サーバー数が SHARED_SERVERS パラメータの値に減少するまで、非アクティブになる共有サーバーには PMON によって終了のマークが付けられます。

次の文は、共有サーバーの数を減らす例です。

```
ALTER SYSTEM SET SHARED_SERVERS = 5;
```

SHARED_SERVERS を 0（ゼロ）に設定すると、共有サーバーは使用禁止になります。詳細は、4-14 ページの「共有サーバーの使用禁止」を参照してください。

共有サーバー・プロセス数の制限

MAX_SHARED_SERVERS パラメータは、PMON によって自動的に作成可能な共有サーバーの最大数を指定します。デフォルト値はありません。値が指定されていない場合、PMON は次の制限に従い、負荷に基づいて必要な数の共有サーバーを起動します。

- プロセスの制限（PROCESSES 初期化パラメータによって設定）
- 最小空きプロセス・スロット数（少なくとも総プロセス・スロット数の 1/8、または PROCESSES が 24 未満に設定されている場合は 2 スロット）
- システム・リソース

> **注意：** Windows NT の場合、各サーバーは共通プロセス内のスレッドであるため、MAX_SHARED_SERVERS を大きい値に設定する場合は注意が必要です。

SHARED_SERVERS の値が、MAX_SHARED_SERVERS の値より優先されます。したがって、SHARED_SERVERS を MAX_SHARED_SERVERS より大きい値に設定すると、MAX_SHARED_SERVERS の値を超える数の共有サーバーを PMON に起動させることができます。その後は、MAX_SHARED_SERVERS の値を SHARED_SERVERS より大きい値に動的に変更することで、共有サーバー数に新しい上限を設定できます。

共有サーバー数を制限する主な理由は、メモリーや CPU タイムなどのリソースを他のプロセス用に確保しておくためです。たとえば、前述のテレマーケティング・センターの例について考えてみます。

DBA は、夜間のバッチ・ジョブ用にリソースの 2/3 を確保しようとします。DBA は、MAX_SHARED_SERVERS の値を最大プロセス数（PROCESSES）の 1/3 より小さい値に設定します。この設定によって、DBA は、すべての従業員がデータベースに同時にアクセスした場合でも、バッチ・ジョブは専用サーバーに接続でき、従業員の要求処理後に共有サーバーが停止するまで待機する必要がないことを確認します。

共有サーバー数を制限するもう 1 つの理由は、多数のサーバー・プロセスが同時に実行されることによって、大量のスワッピングが発生してシステムの処理速度が低下しないようにするためです。ただし、この場合は MAX_SHARED_SERVERS より PROCESSES が上限として機能します。

この他にも、共有サーバー数を制限する理由として、テスト、デバッグ、パフォーマンス分析およびチューニングがあります。たとえば、特定のユーザー・コミュニティを効率的にサポートするために必要な共有サーバーの数を調べるために、MAX_SHARED_SERVERS の値を非常に小さい数に設定しておき、ユーザーが応答時間の遅延を認識しない数まで増やすことができます。

共有サーバー・セッション数の制限

SHARED_SERVER_SESSIONS 初期化パラメータは、同時共有サーバー・ユーザー・セッションの最大数を指定します。動的パラメータであるこのパラメータを設定すると、データベース・セッションを専用サーバー用に確保できます。この結果、データベースのバックアップやリカバリなど、専用サーバーを必要とする管理タスクが共有サーバー・セッションによって専有されることはありません。

このパラメータにデフォルト値はありません。値が指定されていない場合は、SESSIONS 初期化パラメータの制限範囲内で共有サーバー・セッションが必要に応じて作成されます。

共有メモリーの保護

CIRCUITS パラメータによって、共有メモリーに作成可能なバーチャル・サーキットの許容最大数が設定されます。このパラメータにデフォルトはありません。値が指定されていない場合は、DISPATCHERS 初期化パラメータおよびシステム・リソースの制限範囲内でサーキットが必要に応じて作成されます。

ディスパッチャの構成

DISPATCHERS 初期化パラメータは、共有サーバー・アーキテクチャのディスパッチャ・プロセスを構成します。共有サーバーが動作するには、少なくとも 1 つのディスパッチャ・プロセスが必要です。ディスパッチャを指定しない場合でも、SHARED_SERVER に 0（ゼロ）以外の値を設定して共有サーバーを使用可能にすると、Oracle Database によって、TCP プロトコルに対して 1 つのディスパッチャがデフォルトで作成されます。この構成に相当する DISPATCHERS 初期化パラメータの明示的な設定は次のようになります

```
dispatchers="(PROTOCOL=tcp)"
```

次のいずれかの条件に適合する場合は、DISPATCHERS 初期化パラメータを使用して、追加のディスパッチャを構成できます。

- TCP/IP 以外のプロトコルを構成する必要がある場合。DISPATCHERS パラメータの次のいずれかの属性でプロトコル・アドレスを構成します。
  - ADDRESS
  - DESCRIPTION
  - PROTOCOL
- 次のオプションのディスパッチャ属性を 1 つ以上構成する場合。
  - DISPATCHERS
  - CONNECTIONS
  - SESSIONS
  - TICKS
  - LISTENER
  - MULTIPLEX
  - POOL
  - SERVICE

注意： このパラメータの構成には、Database Configuration Assistant が役立ちます。

DISPATCHERS 初期化パラメータの属性

この項では、DISPATCHERS 初期化パラメータで指定可能な属性について簡単に説明します。

プロトコル・アドレスは必須です。プロトコル・アドレスは次の属性を 1 つ以上使用して指定します。

属性	説明
ADDRESS	ディスパッチャがリスニングするエンドポイントのネットワーク・プロトコル・アドレスを指定します。
DESCRIPTION	ネットワーク・プロトコル・アドレスなど、ディスパッチャがリスニングするエンドポイントのネットワークの説明を指定します。構文は次のとおりです。 (DESCRIPTION=(ADDRESS=...))
PROTOCOL	ディスパッチャによってリスニング・エンドポイントが生成されるネットワーク・プロトコルを指定します。次に例を示します。 (PROTOCOL=tcp) プロトコル・アドレスの構文の詳細は、『Oracle Database Net Services リファレンス』を参照してください。

次の属性は、構成に必要なディスパッチャの数を指定します。この属性はオプションで、デフォルトは 1 です。

属性	説明
DISPATCHERS	起動する初期ディスパッチャの数を指定します。

次の属性は、構成の各ディスパッチャのネットワーク属性に関する情報をインスタンスに通知します。これらの属性はすべてオプションです。

属性	説明
CONNECTIONS	各ディスパッチャに対して許容されるネットワーク接続の最大数を指定します。
SESSIONS	各ディスパッチャに対して許容されるネットワーク・セッションの最大数を指定します。
TICKS	ティックの継続時間（秒数）を指定します。ティックは、接続プールのタイムアウトを指定できる時間単位です。接続プーリングに使用されます。
LISTENER	リスナーの別名を指定します。PMON プロセスはこの名前でディスパッチャ情報を登録します。別名には、ネーミング・メソッドを使用して解決される名前を設定してください。
MULTIPLEX	Oracle Connection Manager のセッションの多重化機能を使用可能にする場合に使用します。
POOL	接続プーリングを使用可能にする場合に使用します。
SERVICE	ディスパッチャがリスナーに登録するサービス名を指定します。

完全な属性名または先頭の 3 文字以上で構成されたサブストリングを指定できます。たとえば、SESSIONS=3、SES=3、SESS=3 または SESSI=3 のように指定できます。

関連項目： DISPATCHERS 初期化パラメータの属性の詳細は、『Oracle Database リファレンス』を参照してください。

ディスパッチャ数の決定

オペレーティング・システムに対して各プロセスごとに可能な接続数を把握した後、インスタンス起動時に各ネットワーク・プロトコルに対して作成する初期ディスパッチャの数を、次の式を使用して計算します。

```
Number of dispatchers =
   CEIL ( max. concurrent sessions / connections for each dispatcher )
```

CEIL は、最も近い整数に切り上げた結果を返します。

たとえば、各プロセスごとに 970 の接続数をサポートできるシステムがある場合に、次の最大セッション数を仮定します。

- TCP/IP を介して最大 4,000 のセッションが同時に接続します。
- SSL 付き TCP/IP を介して最大 2,500 のセッションが同時に接続します。

この場合は、次のように、TCP/IP 用の DISPATCHERS 属性には最低で 5 ディスパッチャ（4000 ÷ 970）を、SSL 付き TCP/IP 用には最低で 3 ディスパッチャ（2500 ÷ 970）を設定する必要があります。

```
DISPATCHERS='(PROT=tcp)(DISP=5)', '(PROT-tcps)(DISP=3)'
```

パフォーマンスによっては、ディスパッチャ数の調整が必要となる場合があります。

初期ディスパッチャ数の設定

複数のディスパッチャ構成を指定するには、DISPATCHERS に文字列のカンマ区切りのリストを設定するか、初期化ファイルに複数の DISPATCHERS パラメータを指定します。DISPATCHERS を複数回指定する場合、各行は初期化パラメータ・ファイル内で互いに隣接している必要があります。内部では、Oracle Database によって、INDEX 値（0（ゼロ）から開始）が各 DISPATCHERS パラメータに割り当てられます。この DISPATCHERS パラメータは、後で、ALTER SYSTEM 文の中で索引番号を使用して参照できます。

次に、DISPATCHERS 初期化パラメータの設定例をいくつか示します。

例 : 標準　これは、DISPATCHERS 初期化パラメータの標準的な設定例です。

```
DISPATCHERS="(PROTOCOL=TCP)(DISPATCHERS=2)"
```

例 : ディスパッチャが使用する IP アドレスの設定　次の仮説例では、指定した IP アドレスでリスニングする 2 つのディスパッチャが作成されます。このアドレスは、インスタンスが稼働しているホストの有効な IP アドレスであることが必要です（ホストは複数の IP アドレスで構成されている場合があります）。

```
DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(HOST=144.25.16.201))(DISPATCHERS=2)"
```

例 : ディスパッチャが使用するポートの設定　ディスパッチャで特定のポートをリスニング・エンドポイントとして使用するには、次のように PORT 属性を追加します。

```
DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(PORT=5000))"
DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(PORT=5001))"
```

ディスパッチャ数の変更

特定のインスタンスに対するディスパッチャ・プロセス数を制御できます。共有サーバー数と異なり、ディスパッチャ数は自動的には変更されません。ディスパッチャ数は、ALTER SYSTEM 文を使用して明示的に変更します。Oracle Database の今回のリリースでは、MAX_DISPATCHERS パラメータで指定されている制限より多いディスパッチャ数を設定できます。将来のリリースでは、MAX_DISPATCHERS が考慮される予定です。

次のビューを監視し、ディスパッチャ・プロセスの負荷を判別します。

- V$QUEUE
- V$DISPATCHER
- V$DISPATCHER_RATE

> **関連項目：** これらのビューを監視してディスパッチャの負荷とパフォーマンスを判別する方法については、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

これらのビューによって、ディスパッチャ・プロセスに対する負荷が一貫して高いことが判明した場合は、追加のディスパッチャ・プロセスを起動してユーザー要求をルーティングすることで、パフォーマンスを改善できます。逆に、ディスパッチャの負荷が一貫して低い場合は、ディスパッチャの数を少なくすることにより、パフォーマンスを改善できます。

インスタンスの実行中にディスパッチャの数を動的に変更するには、ALTER SYSTEM 文を使用して、既存のディスパッチャ構成に対する DISPATCHERS 属性の設定を変更します。新規のディスパッチャ構成を追加して、異なるネットワーク属性でディスパッチャを起動することもできます。

特定のディスパッチャ構成に対するディスパッチャ数を少なくしても、ディスパッチャが即時に削除されるわけではありません。Oracle Database は、ユーザーの切断にあわせて DISPATCHERS で指定した制限数に達するまでディスパッチャを停止します。

たとえば、初期化パラメータ・ファイルの次の DISPATCHERS 設定でインスタンスが起動されたとします。

```
DISPATCHERS='(PROT=tcp)(DISP=2)', '(PROT=tcps)(DISP=2)'
```

TCP/IP プロトコル用のディスパッチャ数を 2 から 3 に増やし、SSL 付き TCP/IP プロトコル用のディスパッチャ数を 2 から 1 に減らすには、次の文を発行します。

```
ALTER SYSTEM SET DISPATCHERS = '(INDEX=0)(DISP=3)', '(INDEX=1)(DISP=1)';
```

または

```
ALTER SYSTEM SET DISPATCHERS = '(PROT=tcp)(DISP=3)', '(PROT-tcps)(DISP=1)';
```

> **注意：** （DISP=1）を指定する必要はありません。DISPATCHERS パラメータのデフォルト値は 1 なので、この指定はオプションです。

現在起動している TCP/IP 用のディスパッチャ・プロセスが 2 以下の場合は、新しいプロセスが作成されます。複数の SSL 付き TCP/IP 用のディスパッチャ・プロセスが現在 1 つ以上起動している場合は、接続ユーザーの切断にあわせて余分なプロセスが停止されます。

TCP/IP プロトコル用のディスパッチャ数は変更しないで、接続プーリングをサポートする別の TCP/IP ディスパッチャを追加すると仮定します。この場合は次の文を入力します。

```
ALTER SYSTEM SET DISPATCHERS = '(INDEX=2)(PROT=tcp)(POOL=on)';
```

INDEX 属性は、新規ディスパッチャ構成を追加するために必要です。前述の文で（INDEX=2）を省略すると、INDEX 0 の TCP/IP ディスパッチャ構成が接続プーリングをサポートするように変更され、その構成に対するディスパッチャ数が、ディスパッチャ数（属性 DISPATCHERS）が指定されていない場合のデフォルトである 1 に減らされます。

ディスパッチャ数変更時の注意

- INDEX キーワードを使用すると、変更するディスパッチャ構成を識別できます。この INDEX を指定しないと、指定した DESCRIPTION、ADDRESS または PROTOCOL と一致する最初のディスパッチャ構成が変更されます。既存のディスパッチャ構成の中で一致するものが見つからなかった場合は、新しいディスパッチャが追加されます。
- INDEX 値の範囲は 0 ～ *n*-1 で、*n* は現行のディスパッチャ構成の数です。ALTER SYSTEM 文で INDEX 値に *n* を指定すると（*n* は現行のディスパッチャ構成の数）、新しいディスパッチャ構成が追加されます。
- 現行のディスパッチャ構成の値、つまり、ディスパッチャの数や接続プーリングを使用できるかなどを確認するには、V$DISPATCHER_CONFIG 動的パフォーマンス・ビューを問い合せます。ディスパッチャが関連付けられているディスパッチャ構成を確認するには、V$DISPATCHER ビューの CONF_INDX 列を問い合せます。
- ディスパッチャ構成の DESCRIPTION、ADDRESS、PROTOCOL、CONNECTIONS、TICKS、MULTIPLEX および POOL 属性を変更しても、その変更は既存のディスパッチャには反映されず、新しいディスパッチャにのみ反映されます。したがって、構成に関連付けられているすべてのディスパッチャに対して変更を反映するには、DISPATCHERS パラメータを変更した後で、既存のディスパッチャを強制的に停止し、新しく指定したプロパティに配置された新しいディスパッチャをデータベースで起動する必要があります。

  LISTENER 属性と SERVICES 属性は、同じ制約の対象となりません。これらの属性は、変更した構成に関連付けられている既存のディスパッチャに適用されます。SESSIONS 属性は、その値が減らされた場合のみ、既存のディスパッチャに適用されます。反対に値が増加された場合は、新しく起動されるディスパッチャにのみ適用されます。

特定のディスパッチャ・プロセスの停止

ALTER SYSTEM 文では、ディスパッチャを停止してディスパッチャの数を少なくする判断は、データベースが行います。別の方法として、特定のディスパッチャ・プロセスを停止できます。停止するディスパッチャ・プロセスの名前を識別するには、V$DISPATCHER 動的パフォーマンス・ビューを使用します。

```
SELECT NAME, NETWORK FROM V$DISPATCHER;
```

各ディスパッチャは、D*nnn* 形式の名前で一意に識別されます。

ディスパッチャ D002 を停止するには、次の文を発行します。

```
ALTER SYSTEM SHUTDOWN IMMEDIATE 'D002';
```

IMMEDIATE キーワードを指定すると、ディスパッチャによる新規接続の受入れが停止し、データベースはそのディスパッチャを介した既存のすべての接続を即時に終了します。すべてのセッションがクリーン・アップされてから、ディスパッチャ・プロセスが停止します。IMMEDIATE を指定しなかった場合、ディスパッチャは、接続ユーザーがすべて切断され、接続がすべて終了するまで待ってから停止します。

共有サーバーの使用禁止

共有サーバーは、SHARED_SERVERS を 0（ゼロ）に設定して使用禁止にします。新しいクライアントは共有モードでは接続できなくなります。ただし、SHARED_SERVERS を 0（ゼロ）に設定しても、Oracle Database では、共有サーバーの接続がすべてクローズされるまで一部の共有サーバーは保持されます。保持される共有サーバーの数は、SHARED_SERVERS の変更前の設定で指定されていた数、または MAX_SHARED_SERVERS パラメータの値のいずれか小さい値です。SHARED_SERVERS と MAX_SHARED_SERVERS の両方が 0（ゼロ）に設定されている場合は、すべての共有サーバーが停止し、残りの共有サーバー・クライアントからの要求は、SHARED_SERVERS または MAX_SHARED_SERVERS の値が再度引き上げられるまでキューで待機します。

ディスパッチャを一度停止して、すべての共有サーバー・クライアントを切断するには、次の文を入力します。

```
ALTER SYSTEM SET DISPATCHERS = '';
```

共有サーバーのデータ・ディクショナリ・ビュー

次のビューは、共有サーバー構成情報の取得やパフォーマンスの監視に使用できます。

ビュー	説明
V$DISPATCHER	名前、ネットワーク・アドレス、状態、各種使用統計、索引番号などのディスパッチャ・プロセス情報を提供します。
V$DISPATCHER_CONFIG	ディスパッチャに関する構成情報を提供します。
V$DISPATCHER_RATE	ディスパッチャ・プロセスのレート統計が含まれています。
V$QUEUE	共有サーバー・メッセージ・キューについての情報が含まれています。
V$SHARED_SERVER	共有サーバーについての情報が含まれています。
V$CIRCUIT	バーチャル・サーキットについての情報が含まれています。バーチャル・サーキットとは、ディスパッチャおよびサーバーを介したデータベースへのユーザー接続です。
V$SHARED_SERVER_MONITOR	共有サーバーのチューニング情報が含まれています。
V$SGA	各種のシステム・グローバル領域（SGA）グループに関するサイズ情報が含まれています。この情報は、共有サーバーのチューニング時に役立ちます。
V$SGASTAT	チューニングに役立つ SGA 関連の詳細な統計情報が含まれています。
V$SHARED_POOL_RESERVED	共有プール内で予約済のプールと領域をチューニングする際に役立つ統計リストが含まれています。

関連項目：

- これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。
- 共有サーバーの監視とチューニングの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

データベース常駐接続プーリングの構成

データベース・サーバーは、データベース常駐接続プーリングを使用できるように事前に構成されています。ただし、この機能は、接続プールを起動して明示的に使用可能にする必要があります。

この項の内容は、次のとおりです。

データベース常駐接続プーリングを使用可能にする方法

Oracle Database には、SYS_DEFAULT_CONNECTION_POOL と呼ばれるデフォルトの接続プールがあります。デフォルトでは、このプールは作成されますが起動はされません。データベース常駐接続プーリングを使用可能にするには、接続プールを明示的に起動する必要があります。

データベース常駐接続プーリングを使用可能にする手順は、次のとおりです。

1. 4-16 ページの「データベース常駐接続プールの起動」の説明に従って、データベース常駐接続プールを起動します。
2. 4-16 ページの「接続プールへのクライアント接続要求のルーティング」の説明に従って、クライアント接続要求を接続プールにルーティングします。

データベース常駐接続プールの起動

接続プールを起動する手順は、次のとおりです。

1. SQL*Plus を起動して、SYS ユーザーとしてデータベースに接続します。
2. 次のコマンドを発行します。

```
SQL> EXECUTE DBMS_CONNECTION_POOL.START_POOL();
```

起動されると、接続プールは明示的に停止されるまでこの状態のままです。接続プールは、データベース・インスタンスが再起動されると、インスタンスの停止時点でアクティブだった場合は自動的に再起動されます。

Oracle Real Application Clusters（RAC）環境では、インスタンスを使用して接続プールを管理できます。プール構成に対する変更は、すべての Oracle RAC インスタンスで適用されます。

接続プールへのクライアント接続要求のルーティング

クライアント・アプリケーションでは、接続文字列で接続タイプを POOLED に指定する必要があります。

次の例は、クライアントをデータベース常駐接続プールに接続する簡単な接続文字列を示しています。

```
oraclehost.company.com:1521/books.company.com:POOLED
```

次の例は、クライアントをデータベース常駐接続プールに接続する TNS 接続記述子を示しています。

```
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp) (HOST=myhost)
       (PORT=1521))(CONNECT_DATA=(SERVICE_NAME=sales)
       (SERVER=POOLED)))
```

データベース常駐接続プーリングを使用禁止にする方法

データベース常駐接続プーリングを使用禁止にするには、接続プールを明示的に停止する必要があります。次の手順を実行します。

1. SQL*Plus を起動して、SYS ユーザーとしてデータベースに接続します。
2. 次のコマンドを発行します。

```
SQL> EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();
```

関連項目： DBMS_CONNECTION_POOL パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

注意： データベース常駐接続プールを使用禁止にする操作は、サーバーに渡されたすべてのクライアント要求が完了しないと完了できません。

データベース常駐接続プーリングの接続プールの構成

接続プールはデフォルトのパラメータ値を使用して構成されます。DBMS_CONNECTION_POOL パッケージ内のプロシージャを使用すると、使用方法に応じて接続プールを構成できます。Oracle Real Application Clusters（RAC）環境では、構成パラメータは各 Oracle RAC インスタンスに適用されます。

表 4-2 に、接続プール用に構成可能なパラメータを示します。

表 4-2 データベース常駐接続プーリングの構成パラメータ

パラメータ名	説明
MINSIZE	プール内のプール・サーバーの最小数。デフォルト値は 4 です。
MAXSIZE	プール内のプール・サーバーの最大数。デフォルト値は 40 です。
INCRSIZE	クライアント・アプリケーション要求の受取り時にサーバーが使用可能でない場合にプールが増分されるプール・サーバーの数。デフォルト値は 3 です。
SESSION_CACHED_CURSORS	各プール・サーバー・セッションでキャッシュするセッション・カーソルの数。デフォルト値は 20 です。
INACTIVITY_TIMEOUT	プール・サーバーがプールでアイドル状態のままで待機する最大時間（秒数）。この時間が経過すると、サーバーは終了します。デフォルト値は 300 です。 このパラメータはプールが MINSIZE の場合は適用されません。
MAX_THINK_TIME	クライアントがプールからプール・サーバーを取得した後で非アクティブ状態でいる最大時間（秒数）。クライアント・アプリケーションが、プールからプール・サーバーを取得した後、MAX_THINK_TIME で指定した時間内にデータベース・コールを発行しない場合、プール・サーバーは解放されてクライアント接続が終了します。デフォルト値は 30 です。
MAX_USE_SESSION	プール・サーバーがプールから取得されプールに解放される回数。デフォルト値は 5000 です。
MAX_LIFETIME_SESSION	プール・サーバーがプールに存在している時間（秒数）。デフォルト値は 3600 です。
NUM_CBROK	クライアント要求を処理するために作成される接続ブローカの数。デフォルト値は 1 です。 複数の接続ブローカ・プロセスを作成すると、大量のクライアント・アプリケーションがある場合に、クライアント接続要求の負荷を分散できます。

表 4-2 データベース常駐接続プーリングの構成パラメータ（続き）

パラメータ名	説明
MAXCONN_CBROK	各接続ブローカで処理できる最大接続数。 デフォルト値は 40000 です。ただし、データベースがインストールされているプラットフォームで許可される最大接続がデフォルト値より少ない場合は、MAXCONN_CBROK を使用して設定した値はその値で上書きされます。 MAXCONN_CBROK で指定した接続数がサポートされるように、オペレーティング・システムのプロセスごとのファイル記述子に関する制限は十分大きい値に設定してください。

CONFIGURE_POOL プロシージャの使用

DBMS_CONNECTION_POOL パッケージの CONFIGURE_POOL プロシージャを使用すると、拡張オプションを指定して接続プールを構成できます。このプロシージャは通常、接続プールのすべてのパラメータを変更する必要がある場合に使用します。

ALTER_PARAM プロシージャ

DBMS_CONNECTION_POOL パッケージの ALTER_PARAM プロシージャを使用すると、特定の構成パラメータを他のパラメータに影響を与えることなく変更できます。

たとえば、次のコマンドでは、使用されるプール・サーバーの最小数が変更されます。

```
SQL> EXECUTE DBMS_CONNECTION_POOL.ALTER_PARAM ('','MINSIZE','10');
```

次の例では、各接続ブローカで処理できる最大接続数が 50000 に変更されます。

```
SQL> EXECUTE DBMS_CONNECTION_POOL.ALTER_PARAM ('','MAXCONN_CBROK','50000');
```

このコマンドを実行する前に、データベースがインストールされているプラットフォームで許可される最大接続数が MAXCONN_CBROK に設定した値未満でないことを確認します。

たとえば、Linux では、/etc/security/limits.conf ファイルに次のエントリがあると、ユーザー test_user に許可される最大接続数は 30000 となります。

```
test_user HARD NOFILE 30000
```

各接続ブローカで処理できる最大接続数を 50000 に設定するには、最初に limits.conf ファイルの値を 50000 以上の値に変更します。

接続プールのデフォルト設定のリストア

接続プールのパラメータを変更した後でデフォルトのプール設定に戻す場合は、DBMS_CONNECTION_POOL パッケージの RESTORE_DEFAULT プロシージャを使用します。接続プールの設定をデフォルトに戻すコマンドは、次のとおりです。

```
SQL> EXECUTE DBMS_CONNECTION_POOL.RESTORE_DEFAULTS();
```

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

データベース常駐接続プーリングのデータ・ディクショナリ・ビュー

表 4-3 に、データベース常駐接続プーリングに関する情報を提供するデータ・ディクショナリ・ビューを示します。これらのビューを使用すると、接続プールに関する情報を取得し、データベース常駐接続プーリングのパフォーマンスを監視できます。

表 4-3 データベース常駐接続プーリングのデータ・ディクショナリ・ビュー

ビュー	説明
DBA_CPOOL_INFO	プール・ステータス、接続の最大数と最小数、アイドル・セッションのタイムアウトなど、接続プールに関する情報が含まれます。
V$CPOOL_STATS	セッション要求の数、要求と一致するセッションがプールで検出された回数、セッション要求の合計待機時間など、プール統計が含まれます。
V$CPOOL_CC_STATS	プールの接続クラス・レベルの統計が含まれます。

関連項目： これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。

Oracle Database バックグラウンド・プロセスの概要

マルチプロセスの Oracle Database システムでは、最高のパフォーマンスを提供し、多くのユーザーが同時に使用できるように、**バックグラウンド・プロセス**が使用されています。バックグラウンド・プロセスでは、ユーザー・プロセスごとに実行される複数のデータベース・プログラムによって処理される機能が統合されます。バックグラウンド・プロセスは、非同期的に I/O を実行して他の Oracle Database プロセスを監視することによって、並列性を高め、パフォーマンスと信頼性を向上させます。

表 4-4 は、基本的な Oracle Database バックグラウンド・プロセスを示しています。これらの多くは、このマニュアルの別の箇所に詳細な解説があります。データベース・サーバーの機能またはオプションを追加すると、バックグラウンド・プロセスの数が増える場合があります。たとえば、アドバンスト・キューイングを使用している場合は、キュー・モニター（QMN*n*）バックグラウンド・プロセスが存在します。また、データファイルをストレージ・サブシステムの物理デバイスにマッピングするために FILE_MAPPING 初期化パラメータを指定している場合は、FMON プロセスが存在します。

表 4-4 Oracle Database バックグラウンド・プロセス

プロセス名	説明
データベース・ライター（DBW*n*）	データベース・ライターは、変更があったブロックをデータベース・バッファ・キャッシュからデータファイルに書き込みます。Oracle Database では、最大 20 のデータベース・ライター・プロセス（DBW0 ～ DBW9 および DBWa ～ DBWj）を使用できます。DBW*n* プロセスの数は、DB_WRITER_PROCESSES 初期化パラメータで指定します。データベースは、CPU 数とプロセッサ・グループ数に基づいて、この初期化パラメータに適切なデフォルト設定を選択またはユーザー指定の設定を調整します。 DB_WRITER_PROCESSES 初期化パラメータの設定の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
ログ・ライター（LGWR）	ログ・ライター・プロセスは、REDO ログ・エントリをディスクに書き込みます。REDO ログ・エントリは、SGA の REDO ログ・バッファ内で生成されます。LGWR によって、REDO ログ・エントリは REDO ログ・ファイルに順次書き込まれます。データベースの REDO ログが多重化されている場合、LGWR は REDO ログ・エントリを REDO ログ・ファイルのグループに書き込みます。ログ・ライター・プロセスの詳細は、第 10 章「REDO ログの管理」を参照してください。

表 4-4 Oracle Database バックグラウンド・プロセス（続き）

プロセス名	説明
チェックポイント（CKPT）	SGA 内で変更があったすべてのデータベース・バッファは、特定の時点で DBW*n* によってデータファイルに書き込まれます。このイベントはチェックポイントと呼ばれます。チェックポイント・プロセスは、最新のチェックポイントを示すために、チェックポイントで DBW*n* にシグナルを出し、データベースのすべてのデータファイルと制御ファイルを更新する役割を持ちます。
システム・モニター（SMON）	システム・モニターは、障害の発生したインスタンスの再起動時にリカバリを実行します。Oracle Real Application Clusters データベースでは、1 つのインスタンスの SMON プロセスが、障害を起こした他のインスタンスのインスタンス・リカバリを実行できます。また、SMON により、不要になった一時セグメントがクリーン・アップされ、ファイル読込みエラーやオフライン・エラーのためにシステム障害時やインスタンス・リカバリ時にスキップされたデッド・トランザクションがリカバリされます。これらのトランザクションは、表領域またはファイルがオンラインに戻るときに、SMON によって最後にリカバリされます。
プロセス・モニター（PMON）	プロセス・モニターは、ユーザー・プロセスが失敗したときにプロセス・リカバリを実行します。PMON は、キャッシュをクリーン・アップし、プロセスで使用されていたリソースを解放する役割を持ちます。また、PMON は、ディスパッチャ・プロセス（この表で後述）とサーバー・プロセスをチェックし、障害がある場合は再起動します。
アーカイバ（ARC*n*）	REDO ログ・ファイルは、ログ・ファイルがいっぱいになるか、ログ・スイッチが発生すると、1 つ以上のアーカイバ・プロセスによってアーカイブ記憶域にコピーされます。アーカイバ・プロセスについては、第 11 章「アーカイブ REDO ログの管理」を参照してください。
リカバラ（RECO）	リカバラ・プロセスは、ネットワーク障害やシステム障害が原因で分散データベース内で保留されている分散トランザクションを解決するために使用されます。ローカル RECO は、一定の間隔でリモート・データベースに接続し、保留されている分散トランザクションのローカル部分のコミットまたはロールバックを自動的に完了しようとします。このプロセスの詳細と開始方法は、第 33 章「分散トランザクションの管理」を参照してください。
ディスパッチャ（D*nnn*）	ディスパッチャは、オプションのバックグラウンド・プロセスです。これが存在するのは、共有サーバー構成を使用している場合のみです。共有サーバーについては、4-7 ページの「Oracle Database の共有サーバー構成」を参照してください。
グローバル・キャッシュ・サービス（LMS）	グローバル・キャッシュ・サービスは、Oracle Real Application Clusters 環境でリソースを管理し、インスタンス間のリソース制御機能を提供します。

関連項目： Oracle Database バックグラウンド・プロセスの詳細は、『Oracle Database 概要』を参照してください。

SQL のパラレル実行用プロセスの管理

注意： この項に記載されているパラレル実行機能は、Oracle Database Enterprise Edition で利用できます。

この項では、SQL 文のパラレル実行の管理方法について説明します。この構成では、Oracle Database によって、SQL 文の処理作業を複数のパラレル・プロセスに分割できます。

多数の SQL 文の実行をパラレル化できます。**並列度**は、単一の処理に対応付け可能なパラレル実行サーバーの数で表されます。並列度は、次の要素によって決まります。

- 文の PARALLEL 句
- 問合せ内で参照されているオブジェクトの場合は、オブジェクトが作成または変更されたときに使用された PARALLEL 句
- 文に挿入されたパラレル・ヒント
- データベースによって決定されたデフォルト

SQL のパラレル実行の使用例は、18-11 ページの「表作成のパラレル化」を参照してください。

この項の内容は、次のとおりです。

関連項目：

- パラレル・ヒントの使用方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

パラレル実行サーバーの概要

インスタンスが起動すると、Oracle Database によって、パラレル操作に使用可能なパラレル実行サーバーのプールが作成されます。**パラレル実行コーディネータ**と呼ばれるプロセスが**パラレル実行サーバー**のプールの実行をディスパッチし、これらすべてのパラレル実行サーバーからユーザーへの結果の送信を調整します。

PARALLEL_MAX_SERVERS 初期化パラメータのデフォルト値は 0（ゼロ）より大きい値に設定されているため、パラレル実行サーバーはデフォルトで使用可能です。これらのプロセスは、並列化を利用できる様々な Oracle Database の機能によって使用されます。関連する初期化パラメータは、データベースによって大半のユーザー用にチューニングされますが、必要に応じて環境にあわせて変更できます。簡単にチューニングできるように、一部のパラメータは動的に変更できます。

並列化は、トランザクション・リカバリ、レプリケーション、SQL 実行など、多くの機能で使用できます。ここで説明している SQL のパラレル実行の場合、1 つの文の実行フェーズ全体を通して、複数のパラレル・サーバー・プロセスが対応付けられます。文の処理が完了すると、その文に対応付けられていたプロセスが他の文を処理できるようになります。

関連項目： SQL のパラレル実行など、パラレル実行の使用方法およびチューニング方法の詳細は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。

セッションのパラレル実行の変更

セッションの SQL のパラレル実行は、ALTER SESSION 文を使用して制御できます。

SQL のパラレル実行を使用禁止にする方法

SQL のパラレル実行は、ALTER SESSION DISABLE PARALLEL DML|DDL|QUERY 文で使用禁止にできます。この文を発行した後は、後続のすべての DML（INSERT、UPDATE、DELETE）、DDL（CREATE、ALTER）または問合せ（SELECT）操作がシリアルに実行されます。これらの文に PARALLEL 句が指定されている場合や、関係する表または索引にパラレル属性が指定されている場合でも、それとは無関係に文はシリアルに実行されます。

次の文は、パラレル DDL 操作を使用禁止にします。

```
ALTER SESSION DISABLE PARALLEL DDL;
```

SQL のパラレル実行を使用可能にする方法

SQL のパラレル実行は、ALTER SESSION ENABLE PARALLEL DML|DDL|QUERY 文で使用可能にできます。その後、PARALLEL 句またはパラレル・ヒントを文に対応付けると、その DML 文、DDL 文または問合せ文はパラレルで実行されます。DDL 文と問合せ文のパラレル実行はデフォルトで使用可能です。

DML 文は、ALTER SESSION 文を明示的に発行して、パラレル DML を使用可能にした場合のみパラレル化できます。

```
ALTER SESSION ENABLE PARALLEL DML;
```

SQL のパラレル実行の強制

ALTER SESSION FORCE PARALLEL DML|DDL|QUERY 文を発行すると、後続のすべての DML、DDL または問合せ文をパラレルで実行するように強制できます。また、特定の並列度を強制的に適用して、後続の文に対応付けられた PARALLEL 句を無効にできます。この文で並列度を指定しなかった場合は、デフォルトの並列度が使用されます。ただし、ヒントによって文に指定された並列度は、強制された並列度を無効にします。

次の文は、後続の文のパラレル実行を強制し、優先する並列度を 5 に設定します。

```
ALTER SESSION FORCE PARALLEL DDL PARALLEL 5;
```

外部プロシージャのプロセスの管理

外部プロシージャは、異なる言語で記述されている別のプログラムからコールされるプロシージャです。例として、PL/SQL プログラムから、特別な用途の処理に必要な 1 つ以上の C ルーチンをコールする場合があげられます。

これらのコール可能ルーチンは、Dynamic Link Library（DLL）または Java クラス・メソッドの場合はライブラリ・ユニットに格納されており、ベース言語で登録されています。Oracle Database には**コール仕様**という特別な用途のインタフェースが用意されており、ユーザーはこのインタフェースを使用して他の言語から外部プロシージャをコールできます。

外部プロシージャをコールするには、アプリケーションがネットワーク・リスナー・プロセスにアラートを送り、続いて、そのプロセスが外部プロシージャ・エージェントを起動します。エージェントのデフォルト名は extproc です。このエージェントは、データベース・サーバーと同じコンピュータ上に存在している必要があります。アプリケーションは、リスナーによって確立されたネットワーク接続を使用して、外部プロシージャ・エージェントに DLL またはライブラリ・ユニットの名前、外部プロシージャの名前および関連するパラメータを渡します。次に、外部プロシージャ・エージェントは DLL またはライブラリ・ユニットをロードして外部プロシージャを実行し、外部プロシージャから返された値をアプリケーションに返送します。

DLL へのアクセスを制御するには、データベース管理者がアプリケーション開発者に適切な DLL の実行権限を付与します。アプリケーション開発者は外部プロシージャを作成し、他のユーザーに特定の外部プロシージャの実行権限を付与します。

注意： 外部ライブラリ（DLL ファイル）は、静的にリンクする必要があります。つまり、他の外部ライブラリ（DLL ファイル）の外部シンボルを参照しないようにします。Oracle Database ではこれらのシンボルを解決しないため、外部プロシージャが失敗する原因となる場合があります。

外部プロシージャをコールするための環境は tnsnames.ora および listener.ora エントリで構成され、データベースのインストール中にデフォルトで構成されます。セキュリティ・レベルを上げるには、追加のネットワーク構成手順の実行が必要になることがあります。これらの手順の詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

関連項目： 外部プロシージャの詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

セッションの停止

現行のユーザー・セッションを停止することが必要になる場合があります。たとえば、管理操作を実行するために、管理に関係のないセッションをすべて停止する必要がある場合などです。ここでは、セッションの停止について説明します。この項の内容は、次のとおりです。

- 停止するセッションの識別
- アクティブ・セッションの停止
- 非アクティブ・セッションの停止

セッションが停止すると、そのセッションのアクティブ・トランザクションがロールバックされ、そのセッションが保持していたリソース（ロックやメモリー領域など）がただちに解放されて、他のセッションで使用可能になります。

SQL 文 ALTER SYSTEM KILL SESSION を使用して、現行のセッションを停止します。次の文は、システム識別子が 7 でシリアル番号が 15 のセッションを停止します。

```
ALTER SYSTEM KILL SESSION '7,15';
```

停止するセッションの識別

停止するセッションを識別するには、セッションの索引番号とシリアル番号を指定します。セッションのシステム識別子（SID）とシリアル番号を識別するには、V$SESSION 動的パフォーマンス・ビューを問い合せます。たとえば、次の問合せはユーザー jward のすべてのセッションを識別します。

```
SELECT SID, SERIAL#, STATUS
  FROM V$SESSION
  WHERE USERNAME = 'JWARD';

SID    SERIAL#    STATUS
-----  ---------  --------
    7         15  ACTIVE
   12         63  INACTIVE
```

Oracle Database に対して SQL コールを実行しているとき、セッションは ACTIVE です。データベースに対して SQL コールを実行していないとき、セッションは INACTIVE です。

関連項目： セッションの状態値については、『Oracle Database リファレンス』を参照してください。

アクティブ・セッションの停止

セッションの停止時にユーザー・セッションがトランザクションを処理している場合（STATUS が ACTIVE）、トランザクションはロールバックされ、ユーザーはただちに次のメッセージを受け取ります。

```
ORA-00028: セッションは強制終了されました。
```

ユーザーが、ORA-00028 のメッセージを受け取った後、データベースに再接続する前に追加の文を実行すると、Oracle Database は次のメッセージを返します。

```
ORA-01012: ログオンされていません。
```

ネットワーク I/O やトランザクションのロールバックを実行している場合は、アクティブ・セッションを中断できません。そのセッションは、操作が完了するまで停止できません。この場合、セッションは、停止するまですべてのリソースを保持します。また、セッションを停止させるために ALTER SYSTEM 文を発行したセッションは、対象のセッションが停止するまで最大 60 秒待機します。中断できなかった操作が 1 分たっても終了しない場合、ALTER SYSTEM 文の発行者は、セッションが停止されることを示すマークが設定されたというメッセージを受け取ります。停止マークが付けられたセッションは、V$SESSION での状態（STATUS）が KILLED になり、サーバー（SERVER）が PSEUDO 以外の値になります。

非アクティブ・セッションの停止

停止時にセッションが Oracle Database に対して SQL コールを実行していない場合（STATUS が INACTIVE）、ORA-00028 のメッセージはただちには返されません。このメッセージは、その後ユーザーが停止したセッションを使用しようとしたときに返されます。

非アクティブ・セッションを停止すると、V$SESSION ビューのセッションの STATUS が KILLED になります。停止したセッションの行は、ユーザーが再びそのセッションを使用しようとして ORA-00028 のメッセージを受け取った後で、V$SESSION から削除されます。

次の例では、非アクティブ・セッションを停止しています。最初に V$SESSION を問い合せてセッションの SID と SERIAL# を識別してから、セッションを停止しています。

```
SELECT SID,SERIAL#,STATUS,SERVER
   FROM V$SESSION
   WHERE USERNAME = 'JWARD';

SID    SERIAL#   STATUS     SERVER
-----  --------  ---------  ---------
    7        15  INACTIVE   DEDICATED
   12        63  INACTIVE   DEDICATED
2 rows selected.

ALTER SYSTEM KILL SESSION '7,15';
Statement processed.

SELECT SID, SERIAL#, STATUS, SERVER
   FROM V$SESSION
   WHERE USERNAME = 'JWARD';

SID    SERIAL#   STATUS     SERVER
-----  --------  ---------  ---------
    7        15  KILLED     PSEUDO
   12        63  INACTIVE   DEDICATED
2 rows selected.
```

プロセスおよびセッションのデータ・ディクショナリ・ビュー

次の表に、プロセスとセッションの管理に役立つデータ・ディクショナリ・ビューを示します。

ビュー	説明
V$PROCESS	現在アクティブになっているプロセスの情報が含まれています。
V$SESSION	現行のセッションごとにセッション情報がリストされます。
V$SESS_IO	各ユーザー・セッションのI/O統計情報が含まれています。
V$SESSION_LONGOPS	6秒（絶対時間）以上実行されていた各種操作の状態が表示されます。現在、状態が表示される操作は、多数のバックアップおよびリカバリ機能、統計収集および問合せの実行などです。Oracle Databaseのリリースごとにさらに操作が追加されます。
V$SESSION_WAIT	各セッションの現在または最後の待機が表示されます。
V$SESSION_WAIT_HISTORY	各アクティブ・セッションの最後の10個の待機イベントがリストされます。
V$WAIT_CHAINS	ブロックされたセッションに関する情報が表示されます。
V$SYSSTAT	システム統計情報が含まれています。
V$RESOURCE_LIMIT	一部のシステム・リソースについて、現行および最大のグローバル・リソース使用率が表示されます。
V$SQLAREA	共有SQL領域に関する統計情報が含まれています。SQL文字列ごとに1行ずつ含まれます。メモリー内にあり、解析済で、実行準備のできているSQL文に関する統計情報も提供します。

5

メモリーの管理

この章の内容は次のとおりです。

メモリー管理の概要

メモリー管理には、データベースの変更に応じた Oracle Database インスタンス・メモリー構造の最適なサイズのメンテナンスが含まれます。管理する必要があるメモリー構造は、システム・グローバル領域（SGA）とインスタンス・プログラム・グローバル領域（インスタンス PGA）です。

Oracle Database では様々なメモリー管理方法がサポートされており、これらは初期化パラメータの設定で選択されます。自動メモリー管理と呼ばれる方法を使用可能にすることをお薦めします。

自動メモリー管理

リリース 11*g* からは、Oracle Database で SGA メモリーとインスタンス PGA メモリーを完全に自動的に管理できます。ユーザーが指定するのはインスタンスで使用する合計メモリー・サイズのみです。Oracle Database によって、処理要求にあわせて SGA とインスタンス PGA の間でメモリーが動的に交換されます。この機能は自動メモリー管理と呼ばれます。このメモリー管理方法を使用すると、個々の SGA コンポーネントのサイズおよび個々の PGA のサイズも動的にチューニングされます。

手動メモリー管理

個々のメモリー・コンポーネントのサイズをより直接的に制御する場合は、自動メモリー管理を使用禁止にして、手動メモリー管理用にデータベースを構成できます。手動メモリー管理として使用できる方法はいくつかあります。これらの方法には、自動の部分が、ある度合で残っているものがあります。それらの方法では、DBA に要求される作業量および知識量が異なります。次の方法があります。

- 自動共有メモリー管理 : SGA 用
- 手動共有メモリー管理 : SGA 用
- 自動 PGA メモリー管理 : インスタンス PGA 用
- 手動 PGA メモリー管理 : インスタンス PGA 用

これらのメモリー管理方法については、この章で後述します。

注意： メモリーを管理する最も簡単な方法は、Oracle Enterprise Manager の GUI を使用することです。

Enterprise Manager を使用してメモリーを管理する手順は、次のとおりです。

1. 次のいずれかを実行します。

   - Oracle Enterprise Manager Database Control を使用している場合は、データベース・ホームページにアクセスします。手順については、『Oracle Database 2 日でデータベース管理者』を参照してください。

   - Oracle Enterprise Manager Grid Control を使用している場合は、対象のデータベース・ターゲットに進みます。データベース・ホームページが表示されます。

2. ページの上部にある**「サーバー」**をクリックして「サーバー」ページを表示します。
3. 「データベース構成」セクションで、**「メモリー・アドバイザ」**をクリックします。

関連項目： メモリーの自動管理および手動管理の様々な方法については、『Oracle Database 概要』を参照してください。

メモリー・アーキテクチャの概要

Oracle Database に関連する基本的なメモリー構造は、次のとおりです。

- システム・グローバル領域（SGA）

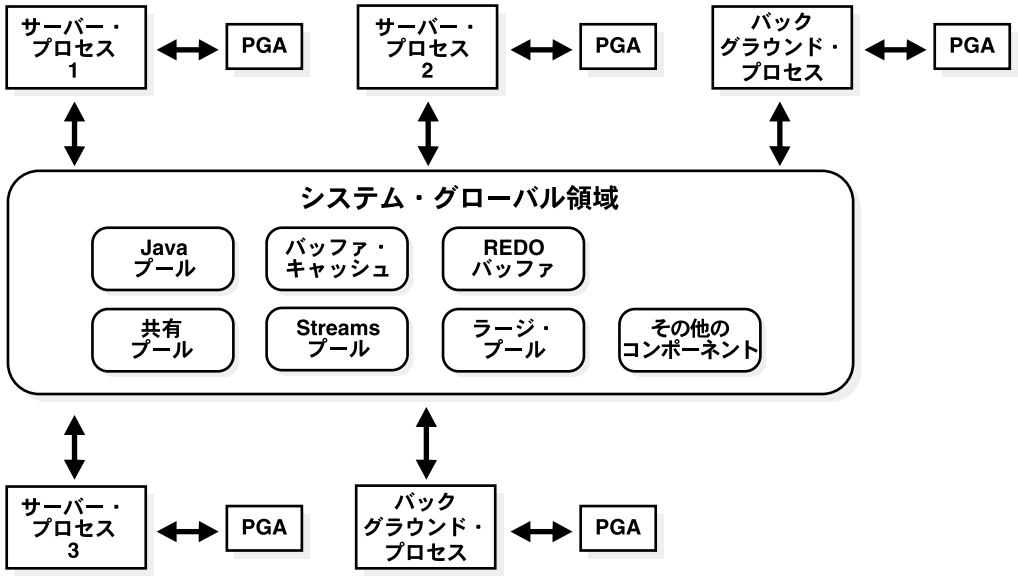
  SGA は、SGA コンポーネントと呼ばれる共有メモリー構造のグループで、1 つの Oracle Database インスタンスに対するデータと制御情報が含まれています。SGA は、すべてのサーバー・プロセスおよびバックグラウンド・プロセスで共有されます。SGA に格納されるデータの例には、キャッシュ・データ・ブロックや共有 SQL 領域があります。

- プログラム・グローバル領域（PGA）

  PGA は、1 つのサーバー・プロセスに対するデータと制御情報を含むメモリー・リージョンです。サーバー・プロセスの起動時に、Oracle Database によって作成される非共有メモリーです。PGA へのアクセスはそのサーバー・プロセスに限定されます。サーバー・プロセスごとに 1 つの PGA があります。また、バックグラウンド・プロセスでも独自の PGA が割り当てられます。Oracle Database インスタンスに連結されたすべてのバックグラウンド・プロセスおよびサーバー・プロセスに対して割り当てられた PGA メモリーの合計は、**合計インスタンス PGA メモリー**と呼ばれます。また、個々の PGA すべての集合は、**合計インスタンス PGA** または単に**インスタンス PGA** と呼ばれます。

図 5-1 に、これらのメモリー構造の関係を示します。

図 5-1 Oracle Database のメモリー構造



関連項目： Oracle Database インスタンスのメモリー・アーキテクチャの詳細は、『Oracle Database 概要』を参照してください。

自動メモリー管理の使用

ここでは、Oracle Database の自動メモリー管理機能のバックグラウンド情報を提供し、この機能を使用可能にする方法を示します。この項の内容は、次のとおりです。

- 自動メモリー管理の概要
- 自動メモリー管理を使用可能にする方法
- 自動メモリー管理の監視およびチューニング

自動メモリー管理の概要

インスタンス・メモリーを管理する最も単純な方法は、Oracle Database インスタンスで自動的に管理およびチューニングされるようにすることです。（ほとんどのプラットフォームで）このために必要な処理は、ターゲット・メモリー・サイズ初期化パラメータ（MEMORY_TARGET）と、オプションで最大メモリー・サイズ初期化パラメータ（MEMORY_MAX_TARGET）を設定することのみです。この結果、インスタンスによってターゲット・メモリー・サイズにチューニングされ、システム・グローバル領域（SGA）とインスタンス・プログラム・グローバル領域（インスタンス PGA）の間で、必要に応じてメモリーが再配分されます。ターゲット・メモリー初期化パラメータは動的であるため、ターゲット・メモリー・サイズは、データベースを再起動することなくいつでも変更できます。最大メモリー・サイズは、ターゲット・メモリー・サイズを誤って大きく設定しすぎないように、また、合計インスタンス・メモリーを今後増やす場合に、Oracle Database インスタンス用に十分なメモリーを確保しておくために、上限の役割を果たします。特定の SGA コンポーネントは容易に縮小できないため、または最小サイズで残しておく必要があるため、インスタンスによって、ユーザーがターゲット・メモリー・サイズを小さく設定しすぎないように制御されます。

Database Configuration Assistant（DBCA）を使用してデータベースを作成する際に基本インストール・オプションを選択した場合は、自動メモリー管理が使用可能になります。拡張インストールを選択した場合は、Database Configuration Assistant（DBCA）を使用して自動メモリー管理を選択できます。

注意： LOCK_SGA 初期化パラメータが TRUE に設定されている場合は、自動メモリー管理を使用可能にできません。このパラメータの詳細は、『Oracle Database リファレンス』を参照してください。

関連項目：

- 5-21 ページ「自動メモリー管理をサポートするプラットフォーム」

自動メモリー管理を使用可能にする方法

データベースの作成時に（DBCA で適切なオプションを選択して、または CREATE DATABASE SQL 文に適切な初期化パラメータを設定して）自動メモリー管理を使用可能にしなかった場合は、後で自動メモリー管理を使用可能にできます。自動メモリー管理を使用可能にする場合は、データベースを停止して再起動する必要があります。

自動メモリー管理を使用可能にする手順は、次のとおりです。

1. SQL*Plus を起動して、SYSDBA ユーザーとしてデータベースに接続します。

   手順については、1-14 ページの「DBA のセキュリティと権限の概要」および 1-16 ページの「DBA の認証」を参照してください。

2. MEMORY_TARGET の最小値を次の方法で計算します。

   a. 次の SQL*Plus コマンドを入力して、SGA_TARGET および PGA_AGGREGATE_TARGET の現行サイズを確認します。

   ```
   SHOW PARAMETER TARGET
   ```

   SQL*Plus によって、すべての初期化パラメータの値が、パラメータ名に TARGET が付加されて表示されます。

   ```
   NAME                           TYPE        VALUE
   ------------------------------ ----------- ----------------
   archive_lag_target             integer     0
   db_flashback_retention_target  integer     1440
   fast_start_io_target           integer     0
   fast_start_mttr_target         integer     0
   memory_max_target              big integer 0
   memory_target                  big integer 0
   pga_aggregate_target           big integer 90M
   sga_target                     big integer 272M
   ```

   b. 次の問合せを実行して、データベースの起動以降に割り当てられた最大インスタンス PGA を確認します。

   ```
   select value from v$pgastat where name='maximum PGA allocated';
   ```

   c. 手順 2b の問合せ結果と PGA_AGGREGATE_TARGET から最大値を算出します。この値に SGA_TARGET を加算します。

   ```
   memory_target = sga_target + max(pga_aggregate_target, maximum PGA allocated)
   ```

   たとえば、前述のように SGA_TARGET が 272M、PGA_AGGREGATE_TARGET が 90M で、割当て済の最大 PGA が 120M と確認された場合、MEMORY_TARGET は 392M (272M + 120M) 以上にする必要があります。

3. 使用する MEMORY_TARGET の値を選択します。

   この値は、手順 2 で計算した最小値にするか、または使用可能な物理メモリーが十分ある場合はこれより大きい値を使用できます。

4. MEMORY_MAX_TARGET 初期化パラメータについては、予測可能な範囲で、データベースに割り当てる予定の最大メモリー量に決定します。つまり、SGA とインスタンス PGA サイズの合計に対する最大値を決定します。この値は、前述の手順で選択した MEMORY_TARGET の値以上に設定できます。

5. 次のいずれかを実行します。

   - Oracle Database インスタンスをサーバー・パラメータ・ファイルを使用して起動（Database Configuration Assistant（DBCA）を使用してデータベースを作成した場合のデフォルト）した場合は、次のコマンドを入力します。

     ```
     ALTER SYSTEM SET MEMORY_MAX_TARGET = nM SCOPE = SPFILE;
     ```

     n は手順 4 で計算した値です。

SCOPE = SPFILE 句を指定すると、サーバー・パラメータ・ファイル内の値のみが設定され、実行中のインスタンスに対する値は設定されません。MEMORY_MAX_TARGET は動的な初期化パラメータではないため、この SCOPE 句を組み込む必要があります。

- インスタンスをテキスト形式の初期化パラメータ・ファイルを使用して起動した場合は、ファイルを手動で編集して次の文を組み込みます。

```
memory_max_target = nM
memory_target = mM
```

n は手順 4 で決定した値、*m* は手順 3 で決定した値です。

注意： テキスト形式の初期化パラメータ・ファイルでは、MEMORY_MAX_TARGET の行を省略して MEMORY_TARGET の値を指定した場合、データベースによって、MEMORY_MAX_TARGET は MEMORY_TARGET の値に自動的に設定されます。MEMORY_TARGET の行を省略して MEMORY_MAX_TARGET を指定した場合、MEMORY_TARGET パラメータはデフォルトで 0（ゼロ）に設定されます。起動後に、MEMORY_TARGET は、MEMORY_MAX_TARGET の値を超えない範囲で、0（ゼロ）以外の値に動的に変更できます。

---

6. データベースを停止して再起動します。

   手順については、3-1 ページの第 3 章「起動と停止」を参照してください。

7. Oracle Database インスタンスをサーバー・パラメータ・ファイルを使用して起動した場合は、次のコマンドを入力します。

```
ALTER SYSTEM SET MEMORY_TARGET = nM;
ALTER SYSTEM SET SGA_TARGET = 0;
ALTER SYSTEM SET PGA_AGGREGATE_TARGET = 0;
```

n は手順 3 で決定した値です。

注意： この手順では、SGA とインスタンス PGA のサイズが必要に応じて制限なくチューニングされるように、SGA_TARGET と PGA_AGGREGATE_TARGET を 0（ゼロ）に設定するように指示しています。これらのパラメータ値を 0（ゼロ）に設定する文を削除し、いずれかまたは両方の値を正数にしておくことができます。この場合、値は SGA またはインスタンス PGA のサイズの最小値として機能します。

関連項目：

- 5-4 ページ「自動メモリー管理の概要」
- 5-3 ページ「メモリー・アーキテクチャの概要」
- ALTER SYSTEM SQL 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

自動メモリー管理の監視およびチューニング

動的なパフォーマンス・ビュー V$MEMORY_DYNAMIC_COMPONENTS に、SGA とインスタンス PGA の合計サイズなど、動的にチューニングされたすべてのメモリー・コンポーネントの現行サイズが示されます。

ビュー V$MEMORY_TARGET_ADVICE では、MEMORY_TARGET 初期化パラメータに対するチューニング・アドバイスが提供されます。

```
SQL>  select * from v$memory_target_advice order by memory_size;

MEMORY_SIZE MEMORY_SIZE_FACTOR ESTD_DB_TIME ESTD_DB_TIME_FACTOR    VERSION
----------- ------------------ ------------ ------------------- ----------
        180                 .5          458               1.344          0
        270                .75          367              1.0761          0
        360                  1          341                   1          0
        450               1.25          335               .9817          0
        540                1.5          335               .9817          0
        630               1.75          335               .9817          0
        720                  2          335               .9817          0
```

MEMORY_SIZE_FACTOR が 1 の行には、MEMORY_TARGET 初期化パラメータで設定されたメモリーの現行サイズ、および現行のワークロードを完了するために必要な DB 時間の量が示されています。上下の行の結果には、MEMORY_TARGET の代替サイズの数値が示されています。各代替サイズについて、サイズ・ファクタ（現行サイズの乗数）、MEMORY_TARGET パラメータが代替サイズに変更された場合に現行のワークロードを完了するために必要な見積 DB 時間が示されます。合計メモリー・サイズが MEMORY_TARGET の現行サイズより小さい場合は、見積 DB 時間が増加していることに注意してください。また、この例では、合計メモリー・サイズを 450MB より大きくしても効果がないことに注意してください。ただし、ワークロード全体がまだ実行されていない場合、この状況は変わる可能性があります。

Enterprise Manager には、MEMORY_TARGET の最適なサイズを選択できる使いやすいグラフィカル・メモリー・アドバイザが用意されています。詳細は、『Oracle Database 2 日でデータベース管理者』を参照してください。

関連項目：

- これらの動的パフォーマンス・ビューの詳細は、『Oracle Database リファレンス』を参照してください。
- DB 時間の定義については、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

メモリーの手動構成

個々のメモリー・コンポーネントのサイズをより直接的に制御する場合は、自動メモリー管理を使用禁止にして、手動メモリー管理用にデータベースを構成できます。SGA およびインスタンス PGA について、それぞれ 2 つの異なる手動メモリー管理方法があります。

SGA の 2 つの手動メモリー管理方法では、DBA に要求される作業量および知識量が異なります。自動共有メモリー管理では、DBA が SGA のターゲット・サイズと最大サイズを設定します。設定後、データベースによって、SGA の合計サイズが指定したターゲットにチューニングされ、多数の SGA コンポーネントのサイズが動的にチューニングされます。手動共有メモリー管理では、DBA が個々のいくつかの SGA コンポーネントのサイズを設定し、その結果、SGA サイズ全体を決定します。その後は、変動する基準に基づいて、これらの個々の SGA コンポーネントを手動でチューニングします。

インスタンス PGA の場合は自動 PGA メモリー管理があり、この方法では DBA がインスタンス PGA のターゲット・サイズを設定します。設定後、データベースによって、インスタンス PGA のサイズが指定したターゲットにチューニングされ、個々の PGA のサイズが動的にチューニングされます。また、手動 PGA メモリー管理もあり、この方法では、DBA が各 SQL 演算子タイプ（sort または hash-join など）の最大作業領域サイズを設定します。このメモリー管理方法は、サポートされていますがお薦めしません。

次の各項では、これらの手動メモリー管理方法すべてについて詳細に説明します。

- 自動共有メモリー管理の使用
- 手動共有メモリー管理の使用
- 自動 PGA メモリー管理の使用
- 手動 PGA メモリー管理の使用

関連項目： Oracle Database のメモリー管理方法の概要は、『Oracle Database 概要』を参照してください。

自動共有メモリー管理の使用

この項の内容は、次のとおりです。

- 自動共有メモリー管理の概要
- SGA のコンポーネントおよびグラニュル
- 最大 SGA サイズの設定
- SGA ターゲット・サイズの設定
- 自動共有メモリー管理を使用可能にする方法
- 自動共有メモリー管理の詳細説明

関連項目：

- SGA のコンポーネントのチューニング方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

自動共有メモリー管理の概要

自動共有メモリー管理によって SGA メモリー管理が簡素化されます。`SGA_TARGET` 初期化パラメータを使用してインスタンスに使用可能な SGA メモリーの合計量を指定すると、Oracle Database によって、メモリーが最も効率的に使用されるように、このメモリーが様々な SGA コンポーネントに自動的に配分されます。

自動共有メモリー管理が使用可能な場合、様々な SGA コンポーネントのサイズは自由に変更され、ワークロードのニーズにあわせてチューニングされるため、追加構成の必要はありません。データベースによって、必要に応じて使用可能なメモリーが様々なコンポーネントに自動的に配分されるため、システムでは使用可能なすべての SGA メモリーを最大限に使用できます。

サーバー・パラメータ・ファイル（SPFILE）を使用している場合、Oracle Database では、自動チューニングされたコンポーネントのサイズがインスタンス停止後も保持されます。このため、システムでは、インスタンスが起動するたびにワークロードの特性を再度認識する必要があります。過去のインスタンスの情報に基づいて、前回インスタンスが停止した時点でのワークロードを継続して評価できます。

SGA のコンポーネントおよびグラニュル

SGA は、特定のクラスのメモリー割当て要求を満たすために使用されるメモリーのプールである多数のメモリー・**コンポーネント**で構成されます。メモリー・コンポーネントの例として、共有プール（SQL および PL/SQL 実行のメモリー割当てに使用）、Java プール（Java オブジェクトおよびその他の Java 実行メモリーに使用）、およびバッファ・キャッシュ（キャッシュ・ディスク・ブロックに使用）などがあります。すべての SGA コンポーネントが**グラニュル**という単位で領域を割当ておよび割当て解除します。Oracle Database は、各 SGA コンポーネントに対するグラニュルの内部数で SGA メモリーの使用状況を追跡します。

SGA の動的コンポーネント用のメモリーは、グラニュル単位で割り当てられます。グラニュル・サイズは、合計 SGA サイズにより決定されます。一般に、ほとんどのプラットフォームでは、合計 SGA サイズが 1GB 以下の場合、グラニュル・サイズは 4MB です。1GB より大きい SGA のグラニュル・サイズは 16MB です。一部にプラットフォーム依存性が存在する場合があります。たとえば、32 ビットの Windows NT の場合、1GB より大きい SGA のグラニュル・サイズは 8MB です。詳細は、使用しているオペレーティング・システム固有のマニュアルを参照してください。

V$SGAINFO ビューを問い合せると、インスタンスで使用中のグラニュル・サイズを確認できます。SGA のすべてのコンポーネントに、同じグラニュル・サイズが使用されます。

コンポーネントに対してグラニュル・サイズの倍数でないサイズを指定すると、そのサイズは最も近い倍数に繰り上げられます。たとえば、グラニュル・サイズが 4MB の場合に DB_CACHE_SIZE を 10MB として指定すると、実際には 12MB が割り当てられます。

最大 SGA サイズの設定

インスタンスの存続期間中の SGA の最大サイズは、SGA_MAX_SIZE 初期化パラメータによって決まります。バッファ・キャッシュ、共有プール、ラージ・プール、Java プールおよび Streams プールのサイズに影響を与える初期化パラメータは動的に変更できますが、これらのサイズと SGA の他のコンポーネント（固定 SGA、可変 SGA および REDO ログ・バッファ）のサイズとの合計が、SGA_MAX_SIZE で指定された値を超えるような変更はできません。

SGA_MAX_SIZE が指定されていない場合は、初期化時に指定またはデフォルト設定されたコンポーネントすべての合計がデフォルト値として選択されます。SGA_MAX_SIZE に指定した値が、データベースの初期化時に、すべてのコンポーネントに対してパラメータ・ファイルで明示的にまたはデフォルトで割り当てたメモリーの合計より少ない場合、SGA_MAX_SIZE の設定は無視され、このパラメータに対する適切な値が選択されます。

SGA ターゲット・サイズの設定

自動共有メモリー管理機能を使用可能にするには、SGA_TARGET パラメータを 0（ゼロ）以外の値に設定します。このパラメータは実際には、個別コンポーネントの特定のセットに対して割り当てられるメモリーを制御するパラメータを置換します。この割当ては現在、必要に応じて自動的かつ動的にサイズ変更（チューニング）されています。

注意： 自動共有メモリー管理が機能するには、STATISTICS_LEVEL 初期化パラメータが TYPICAL（デフォルト）または ALL に設定されている必要があります。

SGA_TARGET 初期化パラメータには、SGA の合計サイズが反映されます。表 5-1 に、SGA_TARGET にメモリーが含まれるコンポーネント（**自動サイズ設定 SGA コンポーネント**）と、コンポーネントに対応する初期化パラメータを示します。

表 5-1 自動サイズ設定 SGA コンポーネントと対応するパラメータ

SGA コンポーネント	初期化パラメータ
固定 SGA および Oracle Database インスタンスに必要なその他の内部割当て	N/A
共有プール	SHARED_POOL_SIZE
ラージ・プール	LARGE_POOL_SIZE
Java プール	JAVA_POOL_SIZE
バッファ・キャッシュ	DB_CACHE_SIZE
Streams プール	STREAMS_POOL_SIZE

表 5-2 にリストされているパラメータは、設定された場合、SGA_TARGET からメモリーを取得しますが、表 5-1 にリストされたコンポーネントに使用されるメモリーは残されます。

表 5-2 SGA_TARGET の領域を使用する手動サイズ設定 SGA コンポーネント

SGA コンポーネント	初期化パラメータ
ログ・バッファ	LOG_BUFFER
KEEP バッファ・キャッシュおよび RECYCLE バッファ・キャッシュ	DB_KEEP_CACHE_SIZE DB_RECYCLE_CACHE_SIZE
非標準ブロック・サイズ・バッファ・キャッシュ	DB_*n*K_CACHE_SIZE

SGA_TARGET を 0（ゼロ）以外の値に設定する他に、すべての自動サイズ設定 SGA コンポーネントの値を 0（ゼロ）に設定して、これらのコンポーネントの完全な自動チューニングを使用可能にする必要があります。

また、1 つ以上の自動サイズ設定 SGA コンポーネントの値を 0（ゼロ）以外に設定することもできます。この結果、SGA のチューニング時にこの値がこのコンポーネントの最小設定として使用されます。この操作については、この項の後半を参照してください。

注意： より簡単に自動共有メモリー管理を使用可能にするには、Oracle Enterprise Manager（EM）を使用します。自動共有メモリー管理を使用可能にし、合計 SGA サイズを設定すると、EM によって ALTER SYSTEM 文が自動的に生成され、SGA_TARGET が指定したサイズに設定され、自動サイズ設定 SGA コンポーネントがすべて 0（ゼロ）に設定されます。

SQL*Plus を使用して SGA_TARGET を設定する場合は、自動サイズ設定 SGA コンポーネントを 0（ゼロ）または最小値に設定する必要があります。

SGA および仮想メモリー ほとんどのシステムで最適なパフォーマンスを実現するには、SGA 全体が実メモリーに収まる必要があります。実メモリーに収まらず、その一部を格納するために仮想メモリーが使用される場合は、データベース・システム全体のパフォーマンスが大幅に低下する可能性があります。これは、オペレーティング・システムによって SGA の一部でページング（ディスクの読取りおよび書込み）が実行されるためです。

ページング・アクティビティを監視する方法は、使用しているオペレーティング・システムのマニュアルを参照してください。また、Enterprise Manager の「ホスト」ページの「パフォーマンス」プロパティ・ページからページング・アクティビティを表示することもできます。

SGA ターゲット・サイズの監視およびチューニング V$SGAINFO ビューでは、様々な SGA コンポーネントの現行のチューニング・サイズに関する情報が提供されます。

V$SGA_TARGET_ADVICE ビューでは、SGA_TARGET の値の決定に役立つ情報が提供されます。

```
SQL> select * from v$sga_target_advice order by sga_size;

  SGA_SIZE SGA_SIZE_FACTOR ESTD_DB_TIME ESTD_DB_TIME_FACTOR ESTD_PHYSICAL_READS
---------- -------------- ------------ ------------------ ------------------
       290             .5       448176             1.6578            1636103
       435            .75       339336             1.2552            1636103
       580              1       270344                  1            1201780
       725           1.25       239038              .8842             907584
       870            1.5       211517              .7824             513881
      1015           1.75       201866              .7467             513881
      1160              2       200703              .7424             513881
```

このビューの情報は、自動メモリー管理用の V$MEMORY_TARGET_ADVICE ビューで提供される情報と同じです。このビューについては、5-7 ページの「自動メモリー管理の監視およびチューニング」を参照してください。

Enterprise Manager には、SGA_TARGET の最適なサイズを選択できる使いやすいグラフィカル・メモリー・アドバイザが用意されています。詳細は、『Oracle Database 2 日でデータベース管理者』を参照してください。

> **関連項目：** これらの動的パフォーマンス・ビューの詳細は、『Oracle Database リファレンス』を参照してください。

自動共有メモリー管理を使用可能にする方法

自動共有メモリー管理（ASMM）を使用可能にする手順は、手動共有メモリー管理から ASMM に変更するか、自動メモリー管理から ASMM に変更するかによって異なります。

手動共有メモリー管理から ASMM に変更する手順は、次のとおりです。

1. 次の問合せを実行して SGA_TARGET の値を取得します。

   ```
   SELECT (
      (SELECT SUM(value) FROM V$SGA) -
      (SELECT CURRENT_SIZE FROM V$SGA_DYNAMIC_FREE_MEMORY)
      ) "SGA_TARGET"
   FROM DUAL;
   ```

2. テキスト形式の初期化パラメータ・ファイルを編集してデータベースを再起動するか、または次の文を発行して SGA_TARGET の値を設定します。

   ```
   ALTER SYSTEM SET SGA_TARGET=value [SCOPE={SPFILE|MEMORY|BOTH}]
   ```

   value は手順 1 で計算した値、またはすべての SGA コンポーネント・サイズの合計と SGA_MAX_SIZE の間の値です。ALTER SYSTEM 文とその SCOPE 句の詳細は、『Oracle Database SQL リファレンス』を参照してください。

3. 次のいずれかを実行します。

   - より完全な自動チューニングを行うには、表 5-1 に記載されている自動サイズ設定 SGA コンポーネントの値を 0（ゼロ）に設定します。このためには、テキスト形式の初期化パラメータ・ファイルを編集するか、または ALTER SYSTEM 文を発行します。
   - 1 つ以上の自動サイズ設定 SGA コンポーネントの最小サイズを制御するには、これらのコンポーネントのサイズを目的の値に設定します（詳細は、次の項を参照してください）。その他の自動サイズ設定 SGA コンポーネントの値を 0（ゼロ）に設定します。このためには、テキスト形式の初期化パラメータ・ファイルを編集するか、または ALTER SYSTEM 文を発行します。

自動メモリー管理から ASMM に変更する手順は、次のとおりです。

1. MEMORY_TARGET 初期化パラメータを 0（ゼロ）に設定します。

   ```
   ALTER SYSTEM SET MEMORY_TARGET = 0;
   ```

   データベースによって、現在の SGA メモリー割当てに基づいて SGA_TARGET が設定されます。

2. 次のいずれかを実行します。

   - より完全な自動チューニングを行うには、表 5-1 に記載されている自動サイズ設定 SGA コンポーネントの値を 0（ゼロ）に設定します。このためには、テキスト形式の初期化パラメータ・ファイルを編集するか、または ALTER SYSTEM 文を発行します。
   - 1 つ以上の自動サイズ設定 SGA コンポーネントの最小サイズを制御するには、これらのコンポーネントのサイズを目的の値に設定します（詳細は、次の項を参照してください）。その他の自動サイズ設定 SGA コンポーネントの値を 0（ゼロ）に設定します。このためには、テキスト形式の初期化パラメータ・ファイルを編集するか、または ALTER SYSTEM 文を発行します。

例 たとえば、手動共有メモリー管理用にインスタンスのパラメータが現在次のように構成されていて、SGA_MAX_SIZE が 1200M に設定されているとします。

- SHARED_POOL_SIZE = 200M
- DB_CACHE_SIZE = 500M
- LARGE_POOL_SIZE = 200M

また、問合せが次のような結果であるとします。

問合せ	結果
SELECT SUM(value) FROM V$SGA	1200M
SELECT CURRENT_SIZE FROM V$SGA_DYNAMIC_FREE_MEMORY	208M

Oracle Enterprise Manager で合計 SGA サイズを 992M に設定するか、次の文を発行することによって、自動共有メモリー管理を利用できます。

```
ALTER SYSTEM SET SGA_TARGET = 992M;
ALTER SYSTEM SET SHARED_POOL_SIZE = 0;
ALTER SYSTEM SET LARGE_POOL_SIZE = 0;
ALTER SYSTEM SET JAVA_POOL_SIZE = 0;
ALTER SYSTEM SET DB_CACHE_SIZE = 0;
ALTER SYSTEM SET STREAMS_POOL_SIZE = 0;
```

992M は、1200M - 208M から算出された値です。

自動共有メモリー管理の詳細説明

ここでは、自動共有メモリー管理についてさらに詳細に説明します。この項の内容は、次のとおりです。

- 自動サイズ設定 SGA コンポーネントの最小値の設定
- 自動チューニングと共有プール
- SGA_TARGET の動的変更
- 自動サイズ設定コンポーネントのパラメータの変更
- 手動サイズ設定コンポーネントのパラメータの変更

自動サイズ設定 SGA コンポーネントの最小値の設定 自動サイズ設定 SGA コンポーネントに対応するパラメータに最小値を指定して、これらのコンポーネントのサイズをある程度制御できます。この設定は、特定のコンポーネントで最低限のメモリーが確保されていないとアプリケーションが正しく動作しないことがわかっている場合に便利です。コンポーネントに対して SGA 領域の最小量を指定するには、対応する初期化パラメータの値を設定します。

1 つ以上の自動サイズ設定コンポーネントの最小サイズを手動で制限すると、動的な調整に使用されるメモリーの合計量が減少します。この量が減少すると、システムがワークロードの変動に適応するシステムの機能が次々に制限されます。したがって、この方法は特別な場合を除いてお薦めしません。デフォルトの自動管理は、システム・パフォーマンスおよび使用可能なリソースの使用が最大になるように動作します。

自動チューニングと共有プール 自動共有メモリー管理機能を使用可能にすると、内部チューニング・アルゴリズムにより、ワークロードに基づいて共有プールの最適サイズが決定されます。通常、値を少しずつ徐々に増加することにより、この値に収束していきます。ただし、内部チューニング・アルゴリズムでは通常、共有プールは縮小されません。これは、共有プール内にオープン・カーソル、確保された PL/SQL パッケージおよびその他の SQL 実行状態が存在することによって、解放可能なグラニュルを検出できないためです。このため、内部チューニング・アルゴリズムでは、共有プールの控えめな増分のみが行われます。控えめなサイズから開始し、パフォーマンスが最適になったサイズで共有プールを安定させます。

SGA_TARGET の動的変更 SGA_TARGET パラメータは、SGA_MAX_SIZE パラメータに指定された値まで動的に増やすことができ、減らすこともできます。SGA_TARGET の値を減らした場合は、メモリーを解放する 1 つ以上の自動調整コンポーネントがシステムによって識別されます。SGA_TARGET は、1 つ以上の自動調整コンポーネントがその最小サイズに達するまで減らすことができます。Oracle Database では、SGA_TARGET の最小許容値を決定するときに、自動サイズ設定コンポーネントに設定された値、SGA_TARGET 領域を使用する手動サイズ設定コンポーネント、および CPU の数など、いくつかの要因が考慮されます。

SGA_TARGET が変更されたときに、消費される物理メモリーの量の変更は、オペレーティング・システムによって異なります。動的共有メモリーをサポートしない一部の UNIX プラットフォームでは、SGA で使用される物理メモリーは SGA_MAX_SIZE パラメータの値と同じです。SGA_TARGET を SGA_MAX_SIZE より小さい値に設定した場合、このようなプラットフォームでは実際の利点はありません。したがって、これらのプラットフォームでは、SGA_MAX_SIZE を設定しないことをお薦めします。

Solaris や Windows などの他のプラットフォームでは、SGA で消費される物理メモリーは SGA_TARGET と同じです。

たとえば、次のような構成環境とします。

- SGA_MAX_SIZE = 1024M
- SGA_TARGET = 512M
- DB_8K_CACHE_SIZE = 128M

この例では、SGA_TARGET は 1024M までサイズを増やすことができ、自動サイズ設定コンポーネントの 1 つ以上がその最小サイズに達するまでサイズを減らすこともできます。正確な値は、システムの CPU 数などの環境要因によって決定します。ただし、DB_8K_CACHE_SIZE の値は常に 128M で固定です。

注意： 自動共有メモリー管理機能を使用可能にする場合、データベースを起動する前に、SGA_TARGET を 0（ゼロ）以外の目的の値に設定しておくことをお薦めします。SGA_TARGET を 0（ゼロ）から 0（ゼロ）以外の値に動的に変更しても、共有プールを縮小できない場合があるため、目的の結果が得られないことがあります。起動後も、必要に応じて SGA_TARGET の値を動的に増減できます。

自動サイズ設定コンポーネントのパラメータの変更 SGA_TARGET が設定されていない場合、自動共有メモリー管理機能は使用できません。したがって、すべてのコンポーネント・パラメータのサイズ変更に関するルールは以前のリリースの場合と同じです。ただし、自動共有メモリー管理が有効な場合は、自動的にサイズ変更したコンポーネントについて手動で指定したサイズが、そのコンポーネントの下限のサイズとして機能します。この制限は、対応するパラメータの値を変更することで動的に変更できます。

特定の SGA コンポーネントに指定したサイズの下限が、現在のサイズより小さい場合、そのコンポーネントのサイズはすぐには変更されません。新しい設定は自動チューニング・アルゴリズムを将来において減少した最小サイズに制限するのみです。たとえば、次のような構成を考えてみます。

- SGA_TARGET = 512M
- LARGE_POOL_SIZE = 256M
- 現在の実際のラージ・プール・サイズ = 284M

この例では、LARGE_POOL_SIZE の値をコンポーネントの現行の実際のサイズを超える値に増加した場合は、増加した最小サイズを格納するためにコンポーネントが拡張されます。たとえば、LARGE_POOL_SIZE の値を 300M に増やした場合、ラージ・プールは 300M に達するまで段階的に拡張されます。このサイズ変更は、1 つ以上の自動チューニング・コンポーネントに影響を与えます。

LARGE_POOL_SIZE の値を 200 まで減らしても、コンポーネントのサイズが即時に変更されることはありません。この新しい設定は、将来ラージ・プールが減らされる限度を 200M までに制限するのみです。

手動サイズ設定コンポーネントのパラメータの変更 手動サイズ設定コンポーネントのパラメータも動的に変更できます。ただし、最小サイズは設定しないで、パラメータの値によって、対応するコンポーネントの正確なサイズを指定します。手動サイズ設定コンポーネントのサイズを増やすと、余分なメモリーが 1 つ以上の自動サイズ設定コンポーネントから削除されます。手動サイズ設定コンポーネントのサイズを減らすと、解放されたメモリーが複数の自動サイズ設定コンポーネントに提供されます。

たとえば、次のような構成を考えてみます。

- SGA_TARGET = 512M
- DB_8K_CACHE_SIZE = 128M

この例の DB_8K_CACHE_SIZE を 16M 増やして 144M にすると、その 16M が、自動サイズ設定コンポーネントから削除されます。同様に、DB_8K_CACHE_SIZE を 16M 減らして 112M にすると、その 16M が、自動サイズ設定コンポーネントに提供されます。

手動共有メモリー管理の使用

自動メモリー管理または自動共有メモリー管理を使用しない場合は、いくつかの SGA コンポーネントのサイズを手動で構成し、データベース・ワークロードの変更に基づいてこれらのサイズを監視およびチューニングする必要があります。ここでは、これらの SGA コンポーネントのサイズを制御するパラメータの設定ガイドラインを提供します。

DBCA を使用してデータベースを作成し、手動共有メモリー管理を選択している場合は、DBCA によって、バッファ・キャッシュ、共有プール、ラージ・プールおよび Java プールのサイズを入力する必要があるフィールドが提供されます。また、作成されるサーバー・パラメータ・ファイル（SPFILE）に、対応する初期化パラメータが設定されます。この方法ではなく CREATE DATABASE SQL 文とテキスト形式のパラメータ・ファイルを使用してデータベースを作成する場合は、次のいずれかを実行できます。

- SGA コンポーネント・サイズを設定する初期化パラメータの値を指定します。
- テキスト形式の初期化ファイルから SGA コンポーネント・サイズのパラメータを削除します。サイズを設定していないコンポーネントに対しては、Oracle Database によって適切なデフォルトが選択されます。

この項の内容は、次のとおりです。

手動共有メモリー管理を使用可能にする方法

手動共有メモリー管理自体を使用可能にする初期化パラメータはありません。自動メモリー管理と自動共有メモリー管理の両方を使用禁止にすることによって、手動共有メモリー管理が事実上使用可能になります。

手動共有メモリー管理を使用可能にする手順は、次のとおりです。

1. MEMORY_TARGET 初期化パラメータを 0（ゼロ）に設定します。
2. SGA_TARGET 初期化パラメータを 0（ゼロ）に設定します。

次に、以降の項の説明に従って、様々な SGA コンポーネントの値を設定する必要があります。

バッファ・キャッシュ初期化パラメータの設定

SGA コンポーネントであるバッファ・キャッシュのサイズは、バッファ・キャッシュ初期化パラメータによって決まります。これらのパラメータを使用して、データベースで使用される各ブロック・サイズのキャッシュ・サイズを指定します。これらの初期化パラメータはすべて動的です。

バッファ・キャッシュのサイズはパフォーマンスに影響を及ぼします。一般に、キャッシュ・サイズを大きくすると、ディスクの読取りと書込みの回数が少なくなります。ただし、キャッシュを大きくすると、メモリーを過度に消費してページングやスワッピングが発生する可能性があります。

Oracle Database では、データベース内で複数のブロック・サイズがサポートされます。非標準のブロック・サイズを設定した表領域を作成する場合は、これらの表領域を格納するための非標準のブロック・サイズ・バッファを構成する必要があります。SYSTEM 表領域には標準のブロック・サイズが使用されます。標準のブロック・サイズを指定するには、初期化パラメータ DB_BLOCK_SIZE を設定します。指定できる値は 2K から 32K です。

データベースで複数のブロック・サイズを使用する場合は、DB_CACHE_SIZE と、少なくとも 1 つの DB_*n*K_CACHE_SIZE パラメータを設定する必要があります。Oracle Database は、DB_CACHE_SIZE パラメータに適切なデフォルト値を割り当てますが、DB_*n*K_CACHE_SIZE パラメータはデフォルトで 0（ゼロ）に設定され、追加のブロック・サイズ・キャッシュは構成されません。

非標準のブロック・サイズ・バッファのサイズおよび数は、次のパラメータによって指定します。

```
DB_2K_CACHE_SIZE
DB_4K_CACHE_SIZE
DB_8K_CACHE_SIZE
DB_16K_CACHE_SIZE
DB_32K_CACHE_SIZE
```

各パラメータは、対応するブロック・サイズのキャッシュ・サイズを指定します。

注意： 最大ブロック・サイズに関するプラットフォーム固有の制限が適用されるため、プラットフォームによっては、これらのサイズの一部は指定できない場合があります。

関連項目： 12-14 ページ「表領域の非標準のブロック・サイズの指定」

ブロック・サイズおよびキャッシュ・サイズの設定例

```
DB_BLOCK_SIZE=4096
DB_CACHE_SIZE=1024M
DB_2K_CACHE_SIZE=256M
DB_8K_CACHE_SIZE=512M
```

この例では、パラメータ DB_BLOCK_SIZE は、データベースの標準ブロック・サイズを 4K に設定しています。標準ブロック・サイズ・バッファのキャッシュ・サイズは 1024MB です。また、2KB および 8KB のキャッシュ・サイズがそれぞれ 256MB と 512MB で構成されます。

注意： DB_*n*K_CACHE_SIZE パラメータを使用して、標準ブロック・サイズのキャッシュ・サイズを指定することはできません。DB_BLOCK_SIZE の値が *n*KB の場合に DB_*n*K_CACHE_SIZE を設定しても無効です。標準ブロック・サイズのキャッシュ・サイズは、常に DB_CACHE_SIZE の値によって決まります。

キャッシュのサイズには制限があるため、ディスク上のすべてのデータをキャッシュに入れられるとはかぎりません。キャッシュがいっぱいになると、その後キャッシュ・ミスが発生し、Oracle Database は新しいデータ用の領域を確保するために、キャッシュ内の使用済データをディスクに書き込みます（バッファが使用済でなければ、新しいブロックをバッファに読み取るためのディスクへの書込みは行われません）。ディスクへの書込みにより上書きされたデータにアクセスすると、さらにキャッシュ・ミスが発生します。

キャッシュのサイズは、データを要求したときにキャッシュ・ヒットになる確率に影響します。キャッシュが大きい場合は、要求されたデータがキャッシュに入っている可能性が高くなります。キャッシュのサイズを大きくすると、データ要求がキャッシュ・ヒットになる確率が高くなります。

インスタンスの実行中も、データベースを停止せずにバッファ・キャッシュのサイズを変更できます。この操作には ALTER SYSTEM 文を使用します。

個々のキャッシュ・コンポーネントのサイズおよび保留中のサイズ変更操作を追跡するには、固定ビュー V$BUFFER_POOL を使用します。

複数バッファ・プール 異なるバッファ・プールを持つデータベース・バッファ・キャッシュを構成して、バッファ・キャッシュ内にデータを保持するか、またはデータ・ブロックの使用直後に新しいデータがバッファを使用できるようにするかを指定できます。その後、特定のスキーマ・オブジェクト（表、クラスタ、索引およびパーティション）を適切なバッファ・プールに割り当て、キャッシュからデータ・ブロックをエージ・アウトする方法を制御できます。

- KEEP バッファ・プールでは、スキーマ・オブジェクトのデータ・ブロックがメモリーに保持されます。
- RECYCLE バッファ・プールでは、データ・ブロックが不要になるとすぐにメモリーから除去されます。
- DEFAULT バッファ・プールには、いずれのバッファ・プールにも割り当てられていないスキーマ・オブジェクトのデータ・ブロックと、明示的に DEFAULT プールに割り当てられたスキーマ・オブジェクトのデータ・ブロックが含まれます。

KEEP バッファ・プールと RECYCLE バッファ・プールを構成する初期化パラメータは、DB_KEEP_CACHE_SIZE と DB_RECYCLE_CACHE_SIZE です。

注意： 複数バッファ・プールは、標準ブロック・サイズに対してのみ使用可能です。非標準ブロック・サイズのキャッシュには、1 つの DEFAULT プールのみ使用できます。

関連項目：

- バッファ・キャッシュのチューニング方法、および複数バッファ・プールの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

共有プール・サイズの指定

SHARED_POOL_SIZE 初期化パラメータは、SGA のコンポーネントである共有プールのサイズを指定または調整する動的なパラメータです。Oracle Database によって、適切なデフォルト値が選択されます。

Oracle Database 10*g* リリース 1 より前のリリースでは、割り当てられる共有プール・メモリーの量は、SHARED_POOL_SIZE 初期化パラメータの値に、インスタンスの起動時に計算された内部 SGA オーバーヘッドの量を加算した値と等しい値でした。内部 SGA オーバーヘッドとは、他の複数の初期化パラメータの値に基づいて起動時に Oracle Database によって割り当てられるメモリーです。このメモリーは、SGA の様々なサーバー・コンポーネントの状態を維持するために使用されます。たとえば、SHARED_POOL_SIZE パラメータが 64MB に設定されていて、計算された内部 SGA オーバーヘッドの値が 12MB である場合、共有プールの実際のサイズは 64+12=76MB ですが、SHARED_POOL_SIZE パラメータの値は 64MB と表示されます。

Oracle Database 10*g* リリース 1 以降では、内部 SGA オーバーヘッドのサイズは、ユーザー指定の SHARED_POOL_SIZE の値に含まれます。つまり、自動メモリー管理または自動共有メモリー管理を使用していない場合、起動時に割り当てられる共有プール・メモリーの量は SHARED_POOL_SIZE 初期化パラメータの値と等しくなり、グラニュル・サイズの倍数に丸められます。したがって、このパラメータには、必要な共有プール・サイズに内部 SGA オーバーヘッドを加えた値を設定する必要があります。前述の例の場合、SHARED_POOL_SIZE パラメータが起動時に 64MB に設定されているとすると、内部 SGA オーバーヘッドの値が変わっていなければ、起動後に使用可能な共有プールの値は、64-12=52MB になります。起動後に共有プール・メモリーとして有効な値を 64MB 維持するには、SHARED_POOL_SIZE パラメータを 64+12=76MB に設定する必要があります。

Oracle Database 10*g* リリース 1 より前のリリースから移行する場合、Oracle Database 11*g* の移行ユーティリティでは、アップグレード前の環境の内部 SGA オーバーヘッドの値とこのパラメータの古い値に基づいて、このパラメータの新しい値が推奨されます。Oracle Database 10*g* からは、内部 SGA オーバーヘッド（共有プールの起動オーバーヘッドとも呼ばれます）の正確な値は、V$SGAINFO ビューから問合せできます。また、手動共有メモリー管理モードでは、ユーザー指定の SHARED_POOL_SIZE の値が小さすぎるために、内部 SGA オーバーヘッドの必要量にも満たない場合は、起動時に ORA-371 エラーが生成され、SHARED_POOL_SIZE パラメータに使用する推奨値が示されます。

Oracle Database 11*g* で自動共有メモリー管理を使用する場合、共有プールは自動的にチューニングされ、ORA-371 エラーは生成されません。

結果キャッシュと共有プール・サイズ 結果キャッシュのメモリーは共有プールから取得されます。したがって、結果キャッシュの最大サイズを大きくする場合は、共有プールのサイズ設定時にこのサイズを考慮する必要があります。

関連項目： 5-18 ページ「結果キャッシュの最大サイズの指定」

ラージ・プール・サイズの指定

LARGE_POOL_SIZE 初期化パラメータは、SGA のコンポーネントであるラージ・プールのサイズを指定または調整する動的なパラメータです。ラージ・プールは、SGA のオプションのコンポーネントです。ラージ・プールを作成する場合は、LARGE_POOL_SIZE パラメータを設定する必要があります。ラージ・プールの構成方法は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

Java プール・サイズの指定

JAVA_POOL_SIZE 初期化パラメータは、SGA のコンポーネントである Java プールのサイズを指定または調整する動的なパラメータです。Oracle Database によって、適切なデフォルト値が選択されます。Java プールの構成方法は、『Oracle Database Java 開発者ガイド』を参照してください。

Streams プール・サイズの指定

STREAMS_POOL_SIZE 初期化パラメータは、SGA のコンポーネントである Streams プールのサイズを指定または調整する動的なパラメータです。STREAMS_POOL_SIZE が 0（ゼロ）に設定されている場合、Oracle Streams 製品では、メモリーが必要時にバッファ・キャッシュから Streams プールに転送されます。詳細は、『Oracle Streams 概要および管理』の Streams プールの説明を参照してください。

結果キャッシュの最大サイズの指定

RESULT_CACHE_MAX_SIZE 初期化パラメータは、SGA のコンポーネントである結果キャッシュの最大サイズを指定する動的なパラメータです。通常このパラメータを指定する必要はありません。これは、データベースによって、SGA に使用可能な合計メモリーと現在使用中のメモリー管理方法に基づいてデフォルトの最大サイズが選択されるためです。現在のデフォルトの最大サイズを参照するには、RESULT_CACHE_MAX_SIZE パラメータの値を表示します。この最大サイズを変更するには、ALTER SYSTEM 文を使用して RESULT_CACHE_MAX_SIZE を設定するか、またはテキスト形式の初期化パラメータ・ファイルでこのパラメータを指定します。いずれの場合も、値は最も近い 32K の倍数に切り上げられます。

インスタンスの起動時に RESULT_CACHE_MAX_SIZE が 0（ゼロ）の場合、結果キャッシュは使用禁止です。使用可能にするには、RESULT_CACHE_MAX_SIZE を 0（ゼロ）以外の値に設定して（またはテキスト形式の初期化パラメータ・ファイルからこのパラメータを削除してデフォルトの最大サイズを取得して）、データベースを再起動する必要があります。

結果キャッシュが使用禁止の状態でデータベースを起動した後、ALTER SYSTEM 文を使用して RESULT_CACHE_MAX_SIZE を 0（ゼロ）以外の値に設定し、その後データベースを再起動していない場合、RESULT_CACHE_MAX_SIZE パラメータの値を問い合せると、結果キャッシュが使用禁止状態のままでも 0（ゼロ）以外の値が返されます。したがって、RESULT_CACHE_MAX_SIZE の値は、結果キャッシュが使用可能かどうかを判断する最も信頼できる方法ではありません。かわりに、次の問合せを使用できます。

```
SELECT dbms_result_cache.status() FROM dual;

DBMS_RESULT_CACHE.STATUS()
------------------------------------------
ENABLED
```

結果キャッシュのメモリーは共有プールから取得されるため、結果キャッシュの最大サイズを大きくする場合は、共有プールのサイズを大きくすることも考慮する必要があります。

ビュー V$RESULT_CACHE_STATISTICS および PL/SQL パッケージのプロシージャ DBMS_RESULT_CACHE.MEMORY_REPORT では、結果キャッシュに現在割り当てられているメモリー量の判断に役立つ情報が表示されます。

PL/SQL パッケージのファンクション DBMS_RESULT_CACHE.FLUSH では、結果キャッシュがクリアされ、すべてのメモリーが共有プールに解放されます。

関連項目：

- 結果キャッシュの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- DBMS_RESULT_CACHE パッケージのプロシージャとファンクションの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- V$RESULT_CACHE_STATISTICS ビューの詳細は、『Oracle Database リファレンス』を参照してください。
- クラスタ・データベースの RESULT_CACHE_MAX_SIZE の設定方法は、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。

その他の SGA 初期化パラメータの指定

いくつかの追加の初期化パラメータを設定して、SGA のメモリー使用方法を制御できます。

物理メモリー LOCK_SGA パラメータを TRUE に設定すると、SGA 全体が物理メモリーにロックされます。このパラメータは、自動メモリー管理または自動共有メモリー管理と組み合せて使用することはできません。

SGA 開始アドレス SHARED_MEMORY_ADDRESS および HI_SHARED_MEMORY_ADDRESS パラメータでは、実行時の SGA の開始アドレスが指定されます。これらのパラメータはほとんど使用されません。64 ビットのプラットフォームの場合、HI_SHARED_MEMORY_ADDRESS では、64 ビット・アドレスの上位 32 ビットが指定されます。

拡張バッファ・キャッシュ・メカニズム USE_INDIRECT_DATA_BUFFERS パラメータは、4GB を超える物理メモリーをサポートする 32 ビット・プラットフォームで、拡張バッファ・キャッシュ・メカニズムを使用可能にします。この大容量の物理メモリーをサポートしていないプラットフォームでは、このパラメータは無視されます。このパラメータは、自動メモリー管理または自動共有メモリー管理と組み合せて使用することはできません。

関連項目：

- これらの初期化パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 5-4 ページ「自動メモリー管理の使用」
- 5-8 ページ「自動共有メモリー管理の使用」

自動 PGA メモリー管理の使用

デフォルトで、Oracle Database では、インスタンス PGA 専用のメモリーの合計量が自動的かつグローバルに管理されます。この量は、初期化パラメータ PGA_AGGREGATE_TARGET を設定することによって制御できます。Oracle Database では、すべてのデータベース・サーバー・プロセスとバックグラウンド・プロセスに割り当てられる PGA メモリーの合計量がこのターゲット値を超えないようにします。

DBCA を使用してデータベースを作成する場合は、合計インスタンス PGA の値を指定できます。その後、DBCA によって、作成されるサーバー・パラメータ・ファイル（SPFILE）に、PGA_AGGREGATE_TARGET 初期化パラメータが設定されます。合計インスタンス PGA を指定しない場合は、DBCA によって適切なデフォルト値が選択されます。

CREATE DATABASE SQL 文とテキスト形式のパラメータ・ファイルを使用してデータベースを作成する場合は、PGA_AGGREGATE_TARGET の値を指定できます。このパラメータを省略すると、データベースによってデフォルト値が選択されます。

自動 PGA メモリー管理を使用すると、すべての専用サーバー・セッションに対する SQL 作業領域のサイズが自動で設定され、これらのセッションについてはすべての *_AREA_SIZE 初期化パラメータが無視されます。特定の時期に、インスタンス上のアクティブな作業領域で使用可能な PGA メモリーの合計量がパラメータ PGA_AGGREGATE_TARGET から自動的に導出されます。この量は、PGA_AGGREGATE_TARGET の値から、他の目的で割り当てられた PGA メモリー（例 : セッション・メモリー）を減算した値に設定されます。結果の PGA メモリーは、その特定のメモリー要件に基づいて個々のアクティブな作業領域に割り当てられます。

PGA メモリーの使用統計を提供する動的なパフォーマンス・ビューが用意されています。これらの統計のほとんどは、PGA_AGGREGATE_TARGET が設定されると使用可能になります。

- 作業領域メモリーの割当ておよび使用に関する統計は、次の動的パフォーマンス・ビューで表示できます。

  ```
  V$SYSSTAT
  V$SESSTAT
  V$PGASTAT
  V$SQL_WORKAREA
  V$SQL_WORKAREA_ACTIVE
  ```

- V$PROCESS ビューの次の 3 つの列では、Oracle Database プロセスによって割り当てられ使用されている PGA メモリーがレポートされます。

  ```
  PGA_USED_MEM
  PGA_ALLOCATED_MEM
  PGA_MAX_MEM
  ```

注意 : 自動 PGA メモリー管理方法は、専用および共有のサーバー・プロセスによって割り当てられた作業領域に適用されます。専用および共有サーバー・モードでの PGA メモリー割当ての詳細は、『Oracle Database 概要』を参照してください。

関連項目 :

- この項に記載したビューの詳細は、『Oracle Database リファレンス』を参照してください。
- これらのビューの使用方法は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

手動 PGA メモリー管理の使用

Oracle Database では、SQL 作業領域を手動でチューニングする手動 PGA メモリー管理がサポートされています。

Oracle Database 10*g* より前のリリースでは、データベース管理者がパラメータ SORT_AREA_SIZE、HASH_AREA_SIZE、BITMAP_MERGE_AREA_SIZE および CREATE_BITMAP_AREA_SIZE を設定することで、SQL 作業領域の最大サイズを制御していました。しかし、最大作業領域サイズは、データ入力サイズとシステム内でアクティブな作業領域の合計数から選択するのが理想であるため、これらのパラメータの設定は困難です。これら 2 つの要因は、作業領域および時間により大幅に変動します。このため、各種 *_AREA_SIZE パラメータを最適な状況でチューニングすることは困難です。

このような理由から、自動 PGA メモリー管理を使用可能にしておくことをお薦めします。

SQL 作業領域を手動でチューニングする場合は、WORKAREA_SIZE_POLICY 初期化パラメータを MANUAL に設定する必要があります。

注意： 初期化パラメータ WORKAREA_SIZE_POLICY は、セッション・レベルおよびシステム・レベルのパラメータで、設定できる値は MANUAL または AUTO の 2 つのみです。デフォルトは AUTO です。PGA_AGGREGATE_TARGET を設定した後、メモリー管理モードを自動と手動の間で切り替えることができます。WORKAREA_SIZE_POLICY が AUTO に設定されている場合、*_AREA_SIZE パラメータの設定は無視されます。

メモリー管理の参考情報

ここでは、メモリー管理に関する次の参考情報を提供します。

- 自動メモリー管理をサポートするプラットフォーム
- メモリー管理のデータ・ディクショナリ・ビュー

自動メモリー管理をサポートするプラットフォーム

次のプラットフォームでは自動メモリー管理がサポートされます。自動メモリー管理は、SGA および PGA のサイズを自動的にチューニングする Oracle Database の機能で、パフォーマンスを最適化するために、要求に応じてメモリーが領域間で再配分されます。

- Linux
- Solaris
- Windows
- HP-UX
- AIX

メモリー管理のデータ・ディクショナリ・ビュー

次の動的パフォーマンス・ビューには、メモリー管理に関する情報が表示されます。

ビュー	説明
V$SGA	システム・グローバル領域（SGA）に関する要約情報が表示されます。
V$SGAINFO	様々な SGA コンポーネントのサイズ、グラニュル・サイズおよび空きメモリーなど、SGA に関するサイズ情報が表示されます。
V$SGASTAT	共有プール、ラージ・プール、Java プールおよび Streams プール内に割り当てられているメモリーの量に関する詳細情報が表示されます。
V$PGASTAT	PGA メモリー使用統計と、自動 PGA メモリー・マネージャが使用可能な場合（PGA_AGGREGATE_TARGET が設定されている場合）はその統計が表示されます。V$PGASTAT の累積値は、インスタンスの起動以降に蓄積されます。
V$MEMORY_DYNAMIC_COMPONENTS	自動調整された静的なすべてのメモリー・コンポーネントの現在のサイズに関する情報と、各コンポーネントで発生した最後の操作（例：拡大または縮小）が表示されます。
V$SGA_DYNAMIC_COMPONENTS	すべての SGA コンポーネントの現在のサイズと各コンポーネントの最後の操作が表示されます。
V$SGA_DYNAMIC_FREE_MEMORY	将来の動的な SGA サイズ変更操作に使用可能な SGA メモリーの容量に関する情報が表示されます。
V$MEMORY_CURRENT_RESIZE_OPS	現在進行中のサイズ変更操作に関する情報が表示されます。サイズ変更の操作は、SGA、インスタンス PGA または動的な SGA コンポーネントの拡大または縮小です。
V$SGA_CURRENT_RESIZE_OPS	現在進行中の動的 SGA コンポーネントのサイズ変更操作に関する情報が表示されます。
V$MEMORY_RESIZE_OPS	SGA_TARGET および PGA_AGGREGATE_TARGET に対する自動拡張および縮小操作も含めて、最後に完了した 800 件のメモリー・コンポーネント・サイズ変更操作に関する情報が表示されます。
V$SGA_RESIZE_OPS	最後に完了した 800 件の SGA コンポーネント・サイズ変更操作に関する情報が表示されます。
V$MEMORY_TARGET_ADVICE	自動メモリー管理を使用可能にした場合、MEMORY_TARGET のチューニングに役立つ情報が表示されます。
V$SGA_TARGET_ADVICE	SGA_TARGET のチューニングに役立つ情報が表示されます。
V$PGA_TARGET_ADVICE	PGA_AGGREGATE_TARGET のチューニングに役立つ情報が表示されます。

関連項目： メモリー管理ビューの詳細は、『Oracle Database リファレンス』を参照してください。

6

ユーザーの管理とデータベースのセキュリティ保護

この章の内容は次のとおりです。

データベースに対するセキュリティ・ポリシー設定の重要性

すべてのデータベースに対してセキュリティ・ポリシーを設定することが重要です。セキュリティ・ポリシーによって、不慮または不正によるデータの破壊またはデータベース・インフラストラクチャの損傷からデータベースを保護する方法が設定されます。

すべてのデータベースにはセキュリティ管理者と呼ばれる管理者が割り当てられ、データベースのセキュリティ・ポリシーの実装およびメンテナンスを担当します。小規模なデータベース・システムの場合は、データベース管理者がセキュリティ管理者を兼務できます。ただし、大規模なデータベース・システムの場合は、専任者または専任グループがセキュリティ管理者の責務を担当するようにしてください。

データベースに対するセキュリティ・ポリシー設定の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

ユーザーとリソースの管理

データベースに接続するには、各ユーザーが、データベースに事前に定義された有効なユーザー名を指定する必要があります。ユーザーにはアカウントが設定され、ユーザーに関する情報がデータベース・ディクショナリに格納されている必要があります。

データベース・ユーザー（アカウント）を作成するときは、ユーザーに関する次の属性を指定します。

- ユーザー名
- 認証方式
- デフォルト表領域
- 一時表領域
- その他の表領域および割当て制限
- ユーザー・プロファイル

ユーザーの作成および管理方法の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

ユーザー権限とロールの管理

権限とロールは、ユーザーのデータへのアクセスおよび実行可能な SQL 文のタイプを制御するために使用します。次の表は、権限とロールの、3 つのタイプを示しています。

タイプ	説明
システム権限	システムによって定義された権限。常に管理者によって付与されます。特定のデータベース操作の実行をユーザーに許可します。
オブジェクト権限	システムによって定義された権限。特定のオブジェクトへのアクセスを制御します。
ロール	権限や他のロールの集合。システムによって定義されたロールも存在しますが、大部分は管理者によって作成されます。権限や他のロールをグループ化するロールによって、複数の権限またはロールをユーザーに容易に付与できます。

権限とロールを付与する権限を所有しているユーザーは、権限とロールを他のユーザーに付与できます。権限とロールの付与は、管理者レベルで開始します。データベースの作成時に、管理ユーザー SYS が作成され、システム権限および Oracle Database で事前定義されたロールがすべて付与されます。次に、ユーザー SYS は、権限とロールを他のユーザーに付与し、そのユーザーが別のユーザーに対して特定の権限を付与できる権限を与えることもできます。

ユーザーの権限とロールの管理方法の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

データベース使用の監査

選択したユーザーのデータベース操作は、管理者が実行した操作も含めて、監視および記録できます。データベース監査を実装する理由はいくつかあります。データベース監査の基本情報と方法の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

事前定義のユーザー・アカウント

Oracle Database には、多数の事前定義のユーザー・アカウントが用意されています。次の 3 種類の事前定義のアカウントがあります。

- 管理アカウント（SYS、SYSTEM、SYSMAN および DBSNMP）

  SYS および SYSTEM については 1-14 ページの「DBA のセキュリティと権限の概要」を参照してください。SYSMAN は、Oracle Enterprise Manager の管理タスクの実行に使用されます。DBSNMP アカウントは、データベースを監視および管理するために Enterprise Manager の管理エージェントで使用されます。これらのアカウントは削除しないでください。

- サンプル・スキーマのアカウント

  これらのアカウントは、Oracle Database のマニュアルや説明書の例で使用されます。たとえば、HR、SH、OE などがあります。これらのアカウントは、使用する前にロックを解除して、パスワードを再設定する必要があります。

- 内部アカウント

  これらのアカウントは、Oracle Database の個々の機能またはコンポーネントで独自のスキーマを使用できるように作成されています。内部アカウントは削除しないでください。また、これらのアカウントを使用してログインしようとしないでください。

  関連項目： 事前定義のアカウントの表は、『Oracle Database 2 日でセキュリティ・ガイド』を参照してください。

7

データベースの動作の監視

データベースの動作を定期的に監視することは重要です。監視することによって、気付いていないエラーの情報を取得できるのみでなく、データベースの正常な動作について理解を深めることもできます。正常な動作を理解しておくことで、なんらかの誤りがある場合に状況を容易に認識できるようになります。

この章の内容は次のとおりです。

エラーおよびアラートの監視

次の各項では、データベースのエラーおよびアラートを監視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。

- トレース・ファイルおよびアラート・ログを使用したエラーの監視
- サーバー生成アラートを使用したデータベースの動作の監視

> **注意：** データベースのエラーおよびアラートを監視する最も簡単で最適な方法は、Enterprise Manager のデータベース・ホームページを使用することです。この項で説明するのは、データ・ディクショナリ・ビュー、PL/SQL パッケージおよびその他のコマンドライン機能を使用する別の監視方法です。

トレース・ファイルおよびアラート・ログを使用したエラーの監視

各サーバー・プロセスとバックグラウンド・プロセスは、対応する**トレース・ファイル**に情報を書き込むことができます。プロセスによって内部エラーが検出されると、エラー情報が関連トレース・ファイルにダンプされます。トレース・ファイルに書き込まれる情報の一部はデータベース管理者用であり、その他の情報は Oracle サポート・サービス用です。また、トレース・ファイルの情報は、アプリケーションとインスタンスのチューニングにも使用されます。

> **注意：** クリティカル・エラーの場合は、自動診断リポジトリにインシデントおよびインシデント・ダンプも作成されます。詳細は、8-1 ページの第 8 章「診断データの管理」を参照してください。

アラート・ログはメッセージとエラーの履歴ログであり、次の項目が含まれます。

- 発生したすべての内部エラー（ORA-600）、ブロック破損エラー（ORA-1578）およびデッドロック・エラー（ORA-60）
- たとえば、CREATE/ALTER/DROP 文、STARTUP/SHUTDOWN 文、ARCHIVELOG 文などの管理操作
- 共有サーバーとディスパッチャ・プロセスの機能に関係するメッセージとエラー
- マテリアライズド・ビューの自動リフレッシュ中に発生したエラー
- データベースとインスタンスの起動時点でデフォルト以外の値があったすべての初期化パラメータの値

Oracle Database は、オペレータのコンソール上に情報を表示するかわりに（一部のシステムではコンソール上に情報を表示する）、アラート・ログを使用して、これらの操作を記録します。操作が成功すると、タイムスタンプとともに「completed」というメッセージがアラート・ログに書き込まれます。

アラート・ログは、XML 形式のファイルとテキスト形式のファイルの両方で保持されます。いずれの形式のアラート・ログもテキスト・エディタで表示できます。または ADRCI ユーティリティを使用すると、XML タグが削除された XML 形式のファイルを表示できます。

バックグラウンド・プロセスでエラーが発生していないかどうかを確認するために、インスタンスのアラート・ログとトレース・ファイルを定期的にチェックします。たとえば、ログ・ライター・プロセス（LGWR）でログ・グループのメンバーに書き込むことができないと、LGWR トレース・ファイルとアラート・ログに、問題の内容を示すエラー・メッセージが書き込まれます。このようなエラー・メッセージが見つかった場合は、メディアまたは I/O の問題が発生しているため、ただちに解決する必要があります。

Oracle Database は、他の重要な統計に加えて、初期化パラメータの値もアラート・ログに書き込みます。

アラート・ログおよびバックグラウンド・プロセスとサーバー・プロセスに対応するすべてのトレース・ファイルは、自動診断リポジトリに書き込まれます。この場所は、DIAGNOSTIC_DEST 初期化パラメータで指定されます。トレース・ファイルの名前はオペレー

ティング・システムによって異なりますが、通常は、ファイルに情報を書き込んでいるプロセスの名前（LGWR、RECO など）が含まれます。

関連項目：

- 自動診断リポジトリの詳細は、8-1 ページの第 8 章「診断データの管理」を参照してください。
- アラート・ログの詳細は、8-5 ページの「アラート・ログ」を参照してください。
- 8-19 ページ「アラート・ログの表示」
- ADRCI ユーティリティの詳細は、『Oracle Database ユーティリティ』を参照してください。
- トレース・ファイルの名前の詳細は、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。

トレース・ファイル・サイズの制御

すべてのトレース・ファイル（アラート・ログを除く）の最大サイズは、ファイルを指定のオペレーティング・システム・ブロック数に制限する初期化パラメータ MAX_DUMP_FILE_SIZE を使用して制御できます。アラート・ログのサイズを制御するには、不要になったファイルを手動で削除する必要があります。削除しないと、ファイルが引き続き追加されます。

インスタンスの実行中にアラート・ログを削除しても問題はありませんが、削除する前にアラート・ログのアーカイブ・コピーを作成してください。このアーカイブ・コピーは、インスタンス履歴の調査を必要とする問題が発生したときに役立ちます。

Oracle Database がトレース・ファイルに書き込む時期の制御

バックグラウンド・プロセスは適宜、トレース・ファイルに情報を書き込みます。ARC*n* バックグラウンド・プロセスの場合は、生成されるトレース情報の量とタイプを初期化パラメータで制御できます。この操作については、11-14 ページの「ARCHIVELOG プロセスによって生成されるトレース出力の制御」を参照してください。他のバックグラウンド・プロセスには、このような柔軟性はありません。

クリティカル・エラーが発生したときは必ず、サーバー・プロセスのためにトレース・ファイルに情報が書き込まれます。また、初期化パラメータ SQL_TRACE = TRUE を設定すると、SQL トレース機能が有効になり、インスタンスに対するすべての SQL 文の処理についてパフォーマンス統計が生成され、自動診断リポジトリに書き込まれます。

必要に応じて、サーバー・プロセスに対するトレース・ファイルの生成を要求できます。SQL 文 ALTER SESSION SET SQL_TRACE を使用すると、SQL_TRACE 初期化パラメータの現行の値にかかわらず、対応するサーバー・プロセスのために各セッションのトレース・ロギングを使用可能または使用禁止にできます。次の例は、特定のセッションに対して SQL トレース機能を使用可能にします。

```
ALTER SESSION SET SQL_TRACE TRUE;
```

セッションの SQL トレースを制御する場合は、DBMS_SESSION または DBMS_MONITOR パッケージを使用します。

注意： サーバー・プロセスの SQL トレース機能は、著しいシステム・オーバーヘッドを引き起こし、パフォーマンスに重大な影響を及ぼします。このため、統計を収集するときにのみ、この機能を使用可能にしてください。

関連項目：

- クリティカル・エラー（インシデントと呼ばれます）のデータベースによる処理方法の詳細は、8-1 ページの第 8 章「診断データの管理」を参照してください。

共有サーバー・セッション用トレース・ファイルの読込み

共有サーバーが使用可能な場合、ディスパッチャを使用している各セッションは共有サーバー・プロセスに転送され、セッションでトレースが使用可能な場合（または、エラーが発生した場合）のみ、トレース情報がサーバーのトレース・ファイルに書き込まれます。ディスパッチャを使用して接続する特定のセッションのトレースを追跡するには、いくつかの共有サーバーのトレース・ファイルを調べることが必要な場合があります。このために、コマンドライン・ユーティリティ・プログラム `trcsess` が用意されています。このプログラムでは、ユーザー・セッションに関係するすべてのトレース情報が 1 つの場所に統合され、情報が時間で順序付けされます。

> **関連項目：** SQL トレース機能の使用方法、および `TKPROF` と `trcsess` を使用して、生成されたトレース・ファイルを解析する方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

サーバー生成アラートを使用したデータベースの動作の監視

サーバー生成アラートとは、切迫した問題に関する Oracle Database サーバーからの通知です。この通知には、問題を修正するための提案が含まれていることがあります。問題の状況が解決した場合も通知が発行されます。

アラートは、問題が発生した場合や次のようなメトリックに予期した値に対してデータが一致しない場合に自動的に生成されます。

- 秒当たりの物理読込み数
- 秒当たりのユーザー・コミット数
- SQL サービス応答時間

サーバー生成アラートは、しきい値レベルに基づいて発行するように指定できます。あるいは、イベントの発生に従って単純に発行することもできます。しきい値ベースのアラートは、警告レベルおよびクリティカル・レベルのしきい値でトリガーできます。これらのレベルの値には、ユーザー定義の値または内部的な値を指定できます。また、一部のアラートにはデフォルトのしきい値レベルがあり、必要に応じてレベルを変更できます。たとえば、デフォルトでは、領域使用率が 85% の警告レベルまたは 97% のクリティカル・レベルのしきい値を超えると、表領域使用に関するサーバー生成アラートが生成されます。次に、しきい値レベルに基づくアラートの例を示します。

- 古すぎるスナップショット
- 一時停止された再開可能セッション
- リカバリ領域の領域使用

アラート・メッセージは、ユーザー `SYS` が所有する事前定義の永続キュー `ALERT_QUE` に送信されます。Oracle Enterprise Manager がこのキューを読み取り、未処理のサーバー・アラートに関する通知を提供します。この通知には、問題を修正するための処置が示されている場合があります。アラートは Enterprise Manager のデータベース・ホームページに表示され、選択した管理者に電子メールまたはページャ通知を送信するように構成できます。アラートをアラート・キューに書き込めない場合、アラートに関するメッセージは Oracle Database のアラート・ログに書き込まれます。

バックグラウンド・プロセスは、自動ワークロード・リポジトリにデータを定期的にフラッシュして、メトリック値の履歴を取得します。アラート履歴表と `ALERT_QUE` は、システムによって一定の間隔で自動的にパージされます。

サーバー生成アラートのしきい値の設定と取得

サーバー・アラート・メトリックのしきい値設定は、DBMS_SERVER_ALERTS PL/SQL パッケージの SET_THRESHOLD および GET_THRESHOLD プロシージャを使用して表示および変更できます。次の各項では、これらのプロシージャの使用例を示します。

- しきい値レベルの設定
- しきい値情報の取得

注意： しきい値を設定および取得する最も便利な方法は、Enterprise Manager のグラフィカル・インタフェースを使用することです。手順については、『Oracle Database 2 日でデータベース管理者』を参照してください。

関連項目： DBMS_SERVER_ALERTS パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

しきい値レベルの設定 次の例は、SET_THRESHOLD プロシージャを使用して、インスタンスに対する各ユーザー・コールの CPU 時間にしきい値を設定する方法を示しています。

```
DBMS_SERVER_ALERT.SET_THRESHOLD(
 DBMS_SERVER_ALERT.CPU_TIME_PER_CALL, DBMS_SERVER_ALERT.OPERATOR_GE, '8000',
 DBMS_SERVER_ALERT.OPERATOR_GE, '10000', 1, 2, 'inst1',
 DBMS_SERVER_ALERT.OBJECT_TYPE_SERVICE, 'main.regress.rdbms.dev.us.oracle.com');
```

この例では、各ユーザー・コールの CPU 時間が 8,000 マイクロ秒を超えた場合は警告アラートが、各ユーザー・コールの CPU 時間が 10,000 マイクロ秒を超えた場合はクリティカル・アラートが発行されます。次の引数が指定されています。

- CPU_TIME_PER_CALL は、メトリック識別子を指定しています。サポートされるメトリックのリストは、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 観察期間は 1 分に設定しています。条件がしきい値を逸脱し、ここに指定した期間（分単位）にわたって逸脱状態が継続すると、アラートが発行されます。
- 連続発生回数は 2 回に設定しています。ここに指定した回数分メトリック値がしきい値に違反すると、アラートが生成されます。
- インスタンスの名前は inst1 に設定しています。
- 定数 DBMS_ALERT.OBJECT_TYPE_SERVICE は、しきい値が設定されるオブジェクト型を指定します。この例では、サービス名は main.regress.rdbms.dev.us.oracle.com です。

しきい値情報の取得 しきい値を取得するには、GET_THRESHOLD プロシージャを使用します。次に例を示します。

```
DECLARE
 warning_operator         BINARY_INTEGER;
 warning_value            VARCHAR2(60);
 critical_operator        BINARY_INTEGER;
 critical_value           VARCHAR2(60);
 observation_period       BINARY_INTEGER;
 consecutive_occurrences  BINARY_INTEGER;
BEGIN
 DBMS_SERVER_ALERT.GET_THRESHOLD(
 DBMS_SERVER_ALERT.CPU_TIME_PER_CALL, warning_operator, warning_value,
    critical_operator, critical_value, observation_period,
    consecutive_occurrences, 'inst1',
 DBMS_SERVER_ALERT.OBJECT_TYPE_SERVICE, 'main.regress.rdbms.dev.us.oracle.com');
 DBMS_OUTPUT.PUT_LINE('Warning operator:       ' || warning_operator);
 DBMS_OUTPUT.PUT_LINE('Warning value:          ' || warning_value);
```

```
 DBMS_OUTPUT.PUT_LINE('Critical operator:      ' || critical_operator);
 DBMS_OUTPUT.PUT_LINE('Critical value:         ' || critical_value);
 DBMS_OUTPUT.PUT_LINE('Observation_period:     ' || observation_period);
 DBMS_OUTPUT.PUT_LINE('Consecutive occurrences:' || consecutive_occurrences);
END;
/
```

DBA_THRESHOLDS ビューを使用して、特定のしきい値設定をチェックすることもできます。次に例を示します。

```
SELECT metrics_name, warning_value, critical_value, consecutive_occurrences
   FROM DBA_THRESHOLDS
   WHERE metrics_name LIKE '%CPU Time%';
```

サーバー生成アラートの表示

サーバー生成アラートを表示する最も簡単な方法は、Enterprise Manager のデータベース・ホームページにアクセスすることです。次の説明では、これらのアラートを表示する別の方法を示します。

Enterprise Manager 以外の独自のツールを使用してアラートを表示する場合は、ALERT_QUE をサブスクライブして、ALERT_QUE を読み取り、アラートにしきい値レベルを設定した後にアラート通知を表示する必要があります。エージェントを作成して、ALERT_QUE にエージェントをサブスクライブするには、DBMS_AQADM パッケージの CREATE_AQ_AGENT および ADD_SUBSCRIBER プロシージャを使用します。

保護されている ALERT_QUE にキューされているメッセージには、サブスクライブするエージェントに関連付けられているユーザーのみアクセスできます。したがって、サブスクライブするエージェントにデータベース・ユーザーを関連付ける必要があります。また、エンキュー権限をユーザーに割り当てることも必要です。そのためには、DBMS_AQADM パッケージの ENABLE_DB_ACCESS および GRANT_QUEUE_PRIVILEGE プロシージャを使用します。

必要な場合は、DBMS_AQ.REGISTER プロシージャに登録して、アラートが ALERT_QUE にエンキューされたときに非同期通知を受信できます。通知は、電子メール、HTTP ポストまたは PL/SQL プロシージャの形式にできます。

アラート・メッセージの読込みには、DBMS_AQ.DEQUEUE プロシージャまたは OCIAQDeq コールを使用できます。メッセージをデキューした後、DBMS_SERVER_ALERT.EXPAND_MESSAGE プロシージャを使用して、メッセージのテキストをオープンします。

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

サーバー生成アラートのデータ・ディクショナリ・ビュー

次のデータ・ディクショナリ・ビューは、サーバー生成アラートに関する情報を提供します。

ビュー	説明
DBA_THRESHOLDS	インスタンスに定義されているしきい値設定をリストします。
DBA_OUTSTANDING_ALERTS	データベースで未処理のアラートを示します。
DBA_ALERT_HISTORY	消去されたアラートの履歴をリストします。
V$ALERT_TYPES	各アラートのグループやタイプなどの情報を提供します。
V$METRICNAME	メトリックの名前、識別子およびシステム・メトリックに関する他の情報が含まれています。
V$METRIC	システム・レベルのメトリック値が含まれています。
V$METRIC_HISTORY	システム・レベルのメトリック値の履歴が含まれています。

> **関連項目：** 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・ビューの詳細は、『Oracle Database リファレンス』を参照してください。

パフォーマンスの監視

データベース・パフォーマンスの監視の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。ここでは、このガイドに記載されていない詳細を含む追加の内容について説明します。

- ロックの監視
- 待機イベントの監視
- パフォーマンス監視のデータ・ディクショナリ・ビュー

ロックの監視

ロックは、同一のリソースにアクセスするトランザクション間での有害な相互作用を防止するメカニズムです。リソースは、表や行などのユーザー・オブジェクトである場合、またはメモリー内の共有データ構造やデータ・ディクショナリ行など、ユーザーには表示されないシステム・オブジェクトの場合があります。SQL 文を実行する場合、Oracle Database は、必要なロックを自動的に取得および管理します。したがって、管理者はこのような詳細を考慮する必要はありません。ただし、データベースでは手動によるデータのロックも可能です。

互いにロックしたデータを 2 人以上のユーザーが待機している場合は、デッドロックが発生する可能性があります。デッドロックによって、一部のトランザクションが続行できなくなります。Oracle Database は、デッドロック状況を自動的に検出し、そのデッドロックに関係している文の 1 つをロールバックしてデッドロック状況を解決し、競合している行ロックの一方を解放します。

Oracle Database は、デッドロックを回避するように設計されていますが、デッドロックは頻繁に発生しません。最も多いデッドロックは、トランザクションがデータベースのデフォルトのロックを明示的に無視する場合です。デッドロックはデータベースのパフォーマンスに影響を与える可能性があるため、ロックを監視できるスクリプトとビューが用意されています。

`utllockt.sql` スクリプトは、ロックを待機しているシステム上のセッションとそのロックをツリー形式で表示します。このスクリプト・ファイルの位置は、オペレーティング・システムによって異なります。

第 2 のスクリプト `catblock.sql` は `utllockt.sql` に必要なロック・ビューを作成するスクリプトであるため、`utllockt.sql` を実行する前に実行する必要があります。

関連項目：

- 7-8 ページ「パフォーマンス監視のデータ・ディクショナリ・ビュー」
- ロックの詳細は、『Oracle Database 概要』を参照してください。

待機イベントの監視

待機イベントとは、イベントの完了を待機してから処理を続行する必要があることを示す統計で、サーバー・プロセスによって増分されます。セッションは、入力の待機、オペレーティング・システムによるディスクへの書込みなどのサービス完了の待機、ロックまたはラッチの待機など、様々な理由で待機することがあります。

セッションがリソースを待機している場合、そのセッションでは効率的な作業が実行されていません。待機の多くはソースに関係しています。待機イベントのデータによって、ラッチの競合、バッファの競合、I/O の競合など、パフォーマンスに影響を与える可能性がある様々な問題の兆候が明らかになります。

このために、待機イベントの統計を表示するビューが用意されています。これらのビューの説明およびインスタンスをチューニングする際のビューの役割については、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

パフォーマンス監視のデータ・ディクショナリ・ビュー

ここでは、Oracle Database インスタンスの監視に使用できるいくつかのデータ・ディクショナリ・ビューについて説明します。これらのビューは一般的なものです。プロセス固有の他のビューについては、そのプロセスに関する項を参照してください。

ビュー	説明
V$LOCK	現在 Oracle Database によって保持されているロックと、未処理のロック要求またはラッチ要求が表示されます。
DBA_BLOCKERS	別のセッションが待機しているオブジェクトのロックを保持しているセッションが表示されます。
DBA_WAITERS	ロックされているオブジェクトを待機しているセッションが表示されます。
DBA_DDL_LOCKS	データベース内のすべての DDL ロックおよび DDL ロックに対する未処理の要求すべてが表示されます。
DBA_DML_LOCKS	データベース内のすべての DML ロックおよび DML ロックに対する未処理の要求すべてが表示されます。
DBA_LOCK	データベース内のすべてのロックまたはラッチ、およびロックまたはラッチに対する未処理の要求すべてが表示されます。
DBA_LOCK_INTERNAL	保持しているロックまたはラッチごとに 1 行、ロックまたはラッチの未処理要求ごとに 1 行の情報が表示されます。
V$LOCKED_OBJECT	システム上のすべてのトランザクションが獲得したすべてのロックがリストされます。
V$SESSION_WAIT	アクティブ・セッションが待機しているリソースまたはイベントが表示されます。
V$SYSSTAT	システム統計情報が含まれています。
V$RESOURCE_LIMIT	一部のシステム・リソースについて、現行および最大のグローバル・リソース使用率が表示されます。
V$SQLAREA	共有 SQL 領域に関する統計情報が、SQL 文字列ごとに 1 行ずつ含まれています。また、メモリー内にあり、解析済で、実行準備のできている SQL 文に関する統計情報も提供します。
V$LATCH	非親ラッチの統計情報と、親ラッチのサマリー統計情報が含まれています。

関連項目： これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。

8

診断データの管理

リリース 11*g* 以降の Oracle Database は、診断データを収集して管理するための高度な障害診断インフラストラクチャを備えています。**診断データ**には、以前のリリースにも含まれていたトレース・ファイル、ダンプおよびコア・ファイルに加えて、顧客や Oracle サポート・サービスが問題を迅速かつ効率的に識別、調査、追跡、解決できる新しいタイプの診断データが含まれています。

この章の内容は次のとおりです。

Oracle Database の障害診断インフラストラクチャの概要

ここでは、Oracle Database の障害診断インフラストラクチャに関するバックグラウンド情報を説明します。この章の内容は、次のとおりです。

障害診断インフラストラクチャの概要

障害診断インフラストラクチャは、問題の防止、検出、診断および解決に役立ちます。特に対象とする問題はクリティカル・エラーで、これには、データベース・コードの不具合、メタデータの破損、顧客データの破損によって生じるエラーなどがあります。

クリティカル・エラーが発生すると、インシデント番号が割り当てられ、エラーに関する診断データ（トレース・ファイルなど）が即時に取得されて、この番号にタグ付けされます。診断データは自動診断リポジトリ（ADR）に格納されます。ADR はデータベースの外部にあるファイルベースのリポジトリで、診断データはインシデント番号によって後で検索して分析できます。

障害診断インフラストラクチャの目的は次のとおりです。

- 初期障害の診断
- 問題の防止
- 問題検出後の破損や割込みの抑制
- 問題の診断時間の短縮
- 問題の解決時間の短縮
- 顧客と Oracle サポート・サービス間の対話の簡素化

これらの目的を達成するための主要なテクノロジは、次のとおりです。

- **初期障害発生時の診断データの自動取得**: クリティカル・エラーの場合は、初期障害時にエラー情報を取得することによって、問題の早期解決と停止時間の短縮の可能性が飛躍的に増大します。常時機能しているメモリーベースのトレース・システムは、多くのデータベース・コンポーネントから診断データを事前に収集するため、問題の根本原因を特定するのに役立ちます。このような事前の診断データは、飛行機の「ブラック・ボックス」フライト・レコーダで収集されるデータに類似しています。問題が検出されると、アラートが生成され、障害診断インフラストラクチャがアクティブ化されて診断データが取得および格納されます。診断データは、データベースの外部にあるリポジトリに格納され（したがって、データベースが停止した場合でもデータを使用できます）、コマンドライン・ユーティリティおよび Enterprise Manager を使用して簡単にアクセスできます。
- **標準化されたトレース形式**: すべてのデータベース・コンポーネント間でトレース形式を標準化することによって、DBA および Oracle サポート・サービス担当者は単一のツール・セットを使用して問題を分析できます。問題をより簡単に診断できるため、停止時間を短縮できます。
- **ヘルス・チェック**: クリティカル・エラーが検出されると、障害診断インフラストラクチャでは 1 つ以上のヘルス・チェックが実行され、クリティカル・エラーの詳細な分析が実行されます。ヘルス・チェックの結果は、エラーについて収集された他の診断データに追加されます。各ヘルス・チェックでは、データ・ブロックの破損、UNDO および REDO の破損、データ・ディクショナリの破損などが検出されます。DBA は、定期的にまたは必要に応じてこれらのヘルス・チェックを手動で起動できます。
- **インシデント・パッケージング・サービス（IPS）とインシデント・パッケージ**: IPS を使用すると、クリティカル・エラーに関する診断データ（トレース、ダンプ、ヘルス・チェック・レポートなど）を自動的かつ容易に収集でき、データを ZIP ファイルにパッケージ化して Oracle サポート・サービスに転送できます。クリティカル・エラーに関する

すべての診断データはそのエラーのインシデント番号にタグ付けされているため、分析に必要なファイルを判別するためにトレース・ファイルなどのファイル全体を検索する必要はありません。インシデント・パッケージング・サービスによって、必要なファイルが自動的に識別され、ZIP ファイルに追加されます。ZIP ファイルが作成される前に、IPS では最初に診断データをインシデント・パッケージ（パッケージ）と呼ばれる中間論理構造に収集します。パッケージは、自動診断リポジトリに格納されます。必要な場合は、この中間論理構造にアクセスして、コンテンツを表示して変更し、診断データの追加や削除をいつでも実行できます。また、準備が完了した時点でパッケージから ZIP ファイルを作成して Oracle サポート・サービスにアップロードできます。

- **データ・リカバリ・アドバイザ**: データ・リカバリ・アドバイザはデータベースのヘルス・チェックおよび Recovery Manager に統合され、データ破損の問題の表示、各問題の程度の評価（クリティカル、高優先度、低優先度）、問題の影響の説明、修復オプションの推奨、顧客が選択したオプションの実行可能性チェック、および修復プロセスの自動化を実行します。
- **SQL テスト・ケース・ビルダー**: 多くの SQL 関連の問題では、再現可能なテスト・ケースの取得が、問題の迅速な解決にとって重要な要因となります。問題とそれが発生した環境に関する可能なかぎりの情報を収集することは困難で時間がかかる場合がありますが、SQL テスト・ケース・ビルダーは、この収集作業を自動化します。迅速に収集された情報を Oracle サポート・サービスにアップロードすると、サポート担当者は問題を簡単かつ正確に再現できます。

関連項目：

- SQL テスト・ケース・ビルダーの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

インシデントおよび問題の概要

クリティカル・エラーの診断と解決を容易にするために、障害診断インフラストラクチャには、問題とインシデントという Oracle Database の概念が導入されています。

問題とは、データベース内のクリティカル・エラーです。クリティカル・エラーは、`ORA-00600` などの内部エラー、または `ORA-07445`（オペレーティング・システム例外）や `ORA-04031`（共有プールのメモリー不足）などの重大なエラーを示します。問題は ADR で追跡されます。各問題には、問題を説明するテキスト文字列の問題キーがあります。この問題キーには、エラー・コード（例 : `ORA 600`）が含まれており、1 つ以上のエラー・パラメータが含まれる場合もあります。

インシデントとは、問題の 1 回の発生です。問題（クリティカル・エラー）が複数回発生した場合は、発生ごとに 1 つのインシデントが作成されます。インシデントには日時が記録され、自動診断リポジトリ（ADR）で追跡されます。各インシデントは、ADR 内で一意の数値であるインシデント ID によって識別されます。インシデントが発生すると、データベースでは次の処理が実行されます。

- アラート・ログにエントリを作成します。
- インシデント・アラートを Oracle Enterprise Manager（Enterprise Manager）に送信します。
- インシデントに関する初期障害診断データをダンプ・ファイル形式（インシデント・ダンプ）で収集します。
- インシデント・ダンプをインシデント ID にタグ付けします。
- インシデントに対して作成された ADR サブディレクトリにインシデント・ダンプを格納します。

通常、クリティカル・エラーの診断と解決は、インシデント・アラートから開始されます。インシデント・アラートは、Enterprise Manager のデータベース・ホームページに表示されます。問題および関連するインシデントは、Enterprise Manager または ADRCI コマンドライン・ユーティリティを使用して表示できます。

インシデントのフラッド制御

1 つの問題に対して、短期間に数十または数百のインシデントが生成される場合があります。その結果、大量の診断データが生成され、ADR の領域を大きく消費し、問題の診断と解決の効率が低下することになります。このため、障害診断インフラストラクチャでは、特定のしきい値に達した後は、フラッド制御がインシデント生成に適用されます。**フラッド制御されたインシデント**とは、アラート・ログ・エントリは作成されて ADR に記録されるが、インシデント・ダンプは生成されないインシデントです。フラッド制御されたインシデントによって、診断データによるシステムのオーバーロードを回避しながら、クリティカル・エラーの発生を管理者に通知できます。Enterprise Manager または ADRCI を使用してインシデントを表示するときは、フラッド制御されたインシデントを表示するか非表示にするかを選択できます。

インシデントのフラッド制御のしきい値レベルは事前定義されており、変更できません。しきい値レベルは次のように定義されています。

- 1 時間に同じ問題キーに対して 5 つのインシデントが発生した場合、その問題キーに対する後続のインシデントはフラッド制御されます。その問題キーに対するインシデントの通常（非フラッド制御）の記録は、次の 1 時間から再開されます。
- 1 日に同じ問題キーに対して 25 のインシデントが発生した場合、その問題キーに対する後続のインシデントはフラッド制御されます。その問題キーに対するインシデントの通常の記録は、翌日から再開されます。

さらに、1 時間に同じ問題キーに対して 50 のインシデントが発生した場合、または 1 日に同じ問題キーに対して 250 のインシデントが発生した場合、その問題キーに対する後続のインシデントは ADR に記録されません。この場合は、後続のインシデントが記録されないことを示すメッセージがアラート・ログに書き込まれます。問題キーに対してインシデントが継続して生成されている間は、1 時間または 1 日が経過するまで、10 分ごとにこのメッセージがアラート・ログに追加されます。1 時間または 1 日が経過すると、その問題キーに対するインシデントの通常の記録が再開されます。

関連項目：

- 8-16 ページ「Enterprise Manager サポート・ワークベンチを使用した問題の表示」
- 8-9 ページ「問題の調査、レポートおよび解決」
- 8-6 ページ「ADRCI コマンドライン・ユーティリティ」

障害診断インフラストラクチャのコンポーネント

障害診断インフラストラクチャの主要なコンポーネントは次のとおりです。

- 自動診断リポジトリ（ADR）
- アラート・ログ
- トレース・ファイル、ダンプおよびコア・ファイル
- ADR のその他の内容
- Enterprise Manager サポート・ワークベンチ
- ADRCI コマンドライン・ユーティリティ

自動診断リポジトリ（ADR）

ADR は、データベース診断データ（トレース、ダンプ、アラート・ログ、状態モニター・レポートなど）用のファイルベースのリポジトリです。ADR は、複数のインスタンスや製品間で統合されたディレクトリ構造を備えています。リリース 11*g* 以降、データベース、自動ストレージ管理（ASM）、その他の Oracle 製品やコンポーネントでは、すべての診断データが ADR に格納されるようになりました。各製品のインスタンスごとに、ADR 内の独自のホーム・ディレクトリ下に診断データが格納されます。たとえば、共有記憶域と ASM を備えた Oracle Real Application Clusters 環境では、各データベース・インスタンスおよび各 ASM インスタンスに 1 つの ADR ホーム・ディレクトリがあります。ADR の統合されたディレクトリ構造、製品とインスタンスを通して一貫性のある診断データ形式、統合されたツール・セットによって、顧

客と Oracle サポート・サービスは、複数のインスタンス間で診断データを関連付けて分析できます。

注意： リリース 11*g* 以降の Oracle Database では、アラート・ログを含むすべての診断データが ADR に格納されるため、初期化パラメータの BACKGROUND_DUMP_DEST と USER_DUMP_DEST が非推奨になりました。これらのパラメータは、ADR の場所を指定する初期化パラメータ DIAGNOSTIC_DEST に置き換えられています。

関連項目： DIAGNOSTIC_DEST パラメータと ADR ホームの詳細は、8-7 ページの「自動診断リポジトリの構造、内容および場所」を参照してください。

アラート・ログ

アラート・ログは XML ファイルで、データベース・メッセージとエラーの履歴ログです。アラート・ログは ADR に格納され、次の内容に関するメッセージが記録されます。

- クリティカル・エラー（インシデント）
- 管理操作（データベースの起動と停止、データベースのリカバリ、表領域の作成や削除など）
- マテリアライズド・ビューの自動リフレッシュ中に発生したエラー
- その他のデータベース・イベント

アラート・ログは、Enterprise Manager および ADRCI ユーティリティを使用して、テキスト形式（XML タグは削除されます）で表示できます。また、下位互換性のために ADR に格納されたテキスト形式のアラート・ログもあります。ただし、テキスト形式は構造化されておらず、リリースごとに変更される場合があるため、アラート・ログの内容の解析は XML 形式で行うことをお薦めします。

関連項目：

- 8-6 ページ「ADRCI コマンドライン・ユーティリティ」
- 8-19 ページ「アラート・ログの表示」

トレース・ファイル、ダンプおよびコア・ファイル

トレース・ファイル、ダンプおよびコア・ファイルには、問題の調査に使用する診断データが含まれています。これらは ADR に格納されます。

トレース・ファイル 各サーバー・プロセスとバックグラウンド・プロセスは、対応するトレース・ファイルに情報を書き込むことができます。トレース・ファイルはプロセスの存続期間にわたって定期的に更新され、プロセスの環境、ステータス、アクティビティおよびエラーに関する情報を格納できます。さらに、プロセスでクリティカル・エラーが検出されると、そのエラーに関する情報が関連のトレース・ファイルに書き込まれます。トレース・ファイルは SQL トレース機能によっても作成され、各 SQL 文のパフォーマンス情報が提供されます。SQL トレース機能は、セッションまたはインスタンスに対して使用可能にできます。

トレース・ファイル名はプラットフォームによって異なります。通常、データベース・バックグラウンド・プロセスのトレース・ファイル名は、Oracle SID、バックグラウンド・プロセス名およびオペレーティング・システム・プロセス番号で構成され、サーバー・プロセスのトレース・ファイル名は、Oracle SID、文字列「ora」およびオペレーティング・システム・プロセス番号で構成されます。ファイル拡張子は .trc です。たとえば、orcl_ora_344.trc は、サーバー・プロセスのトレース・ファイル名です。トレース・ファイルは、対応するトレース・マップ・ファイル（.trm）を伴う場合があります。このファイルにはトレース・ファイルの構造情報が記載されており、検索とナビゲーションに使用されます。

Oracle Database には、トレース・ファイルの分析に役立つツールが用意されています。アプリケーション・トレース機能、SQL トレース機能およびトレース・ツールの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

関連項目： 8-20 ページ「トレース・ファイルの検索」

ダンプ ダンプは、特殊なタイプのトレース・ファイルです。通常、ダンプはイベント（インシデントなど）に関する診断データの 1 回かぎりの出力です。これに対して、トレースは診断データの継続的な出力です。インシデントが発生すると、そのインシデントに対して作成されたインシデント・ディレクトリに 1 つ以上のダンプが書き込まれます。インシデント・ダンプのファイル名には、インシデント番号が含まれます。

コア・ファイル コア・ファイルには、メモリー・ダンプがポート固有の形式ですべてバイナリで格納されます。コア・ファイル名は、文字列「core」とオペレーティング・システム・プロセス ID で構成されます。コア・ファイルを使用するのは Oracle サポート・サービスのエンジニアのみです。プラットフォームによっては、コア・ファイルがない場合があります。

ADR のその他の内容

ADR には、前の各項で説明したファイルに加えて、状態モニター・レポート、データ修復レコード、SQL テスト・ケース、インシデント・パッケージなどが格納されます。これらのコンポーネントについては、この章で後述します。

Enterprise Manager サポート・ワークベンチ

Enterprise Manager サポート・ワークベンチ（サポート・ワークベンチ）は、問題（クリティカル・エラー）を調査およびレポートし、場合によっては修復できる機能で、使用しやすいグラフィカル・インタフェースを備えています。このサポート・ワークベンチは、初期障害診断データの収集、サポート・リクエスト番号の取得、診断データの Oracle サポート・サービスへのアップロードを短時間かつ最小限の作業で実行できるセルフ・サービス機能を提供します。これによって、問題の解決時間を短縮できます。また、サポート・ワークベンチでは、SQL 関連の問題やデータ破損の問題などの修復に役立つ Oracle アドバイザへのアクセスが推奨され、アドバイザに簡単にアクセスできます。

ADRCI コマンドライン・ユーティリティ

ADR コマンド・インタプリタ（ADRCI）は、問題の調査、ヘルス・チェック・レポートの表示、および初期障害診断データのパッケージ化と Oracle サポート・サービスへのアップロードを、すべてコマンドライン環境内で実行できるユーティリティです。また、ADRCI を使用すると、ADR に格納されているトレース・ファイルの名前の表示、およびアラート・ログの表示（XML タグは削除されます）を、フィルタ処理の有無を選択して実行できます。

ADRCI の詳細は、『Oracle Database ユーティリティ』を参照してください。

自動診断リポジトリの構造、内容および場所

自動診断リポジトリ（ADR）は、データベースの外部に格納されるディレクトリ構造です。したがって、このリポジトリは、データベースが停止しても問題の診断に使用できます。

ADR のルート・ディレクトリは **ADR ベース**と呼ばれます。その場所は DIAGNOSTIC_DEST 初期化パラメータによって設定されます。このパラメータを省略するか NULL のままにすると、データベースでは起動時に DIAGNOSTIC_DEST を次のように設定します。

- 環境変数 ORACLE_BASE が設定されている場合、DIAGNOSTIC_DEST は ORACLE_BASE で指定されたディレクトリに設定されます。
- 環境変数 ORACLE_BASE が設定されていない場合、DIAGNOSTIC_DEST は *ORACLE_HOME*/log に設定されます。

ADR ベース内には、複数の ADR ホームが存在する場合があります。この場合、各 ADR ホームは、特定の Oracle 製品またはコンポーネントの特定のインスタンスに関するすべての診断データ（トレース、ダンプ、アラート・ログなど）のルート・ディレクトリです。たとえば、ASM を備えた Oracle Real Application Clusters 環境では、各データベース・インスタンスおよび各 ASM インスタンスに 1 つの ADR ホームがあります。すべての ADR ホームは、同じ階層ディレクトリ構造を共有しています。

ADR ホームの場所は、ADR ベース・ディレクトリの後に続く次のパスで指定されます。

```
diag/product_type/product_id/instance_id
```

表 8-1 に、Oracle Database インスタンスの各パス・コンポーネントの値を示します。

表 8-1 Oracle Database の ADR ホームのパス・コンポーネント

パス・コンポーネント	Oracle Database の値
product_type	rdbms
product_id	データベース名
instance_id	*SID*

たとえば、SID とデータベース名が両方とも orclbi のデータベースの場合、ADR ホームの場所は次のようになります。

```
ADR_base/diag/rdbms/orclbi/orclbi/
```

ADR ホーム・サブディレクトリ

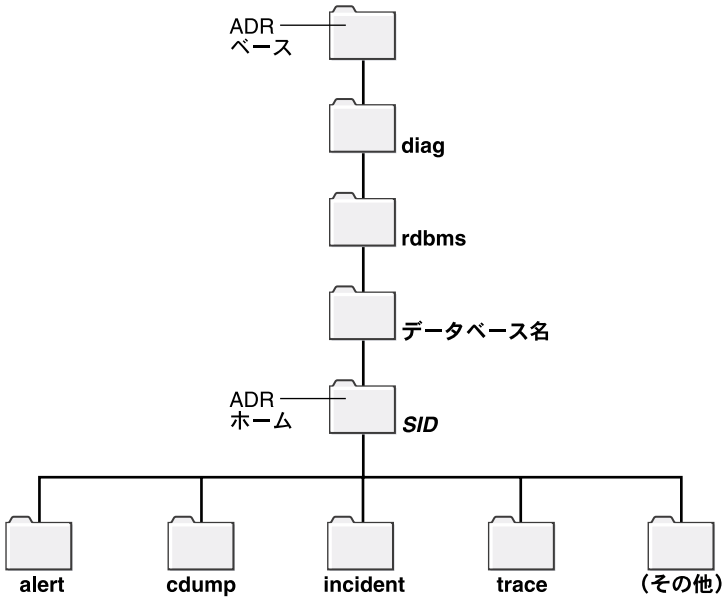
ADR ホーム・ディレクトリ内には、データベース・インスタンスが診断データを格納するサブディレクトリがあります。表 8-2 に、いくつかのサブディレクトリとその内容を示します。

表 8-2 ADR ホーム・サブディレクトリ

サブディレクトリ名	内容
alert	XML 形式のアラート・ログ。
cdump	コア・ファイル。
incident	複数のサブディレクトリがあり、各サブディレクトリには特定のインシデントの名前が付けられ、そのインシデントのみに関するダンプが含まれます。
trace	バックグラウンド・プロセスとサーバー・プロセスのトレース・ファイル、および SQL トレース・ファイル。
（その他）	ADR ホームのその他のサブディレクトリには、インシデント・パッケージ、状態モニター・レポートなどの情報が格納されます。

図 8-1 に、Oracle Database インスタンスの ADR のディレクトリ階層を示します。他の Oracle 製品やコンポーネント（ASM、Oracle Net Services など）用の ADR ホームも、この階層内の同じ ADR ベース下に存在する場合があります。

図 8-1 Oracle Database インスタンスの ADR ディレクトリ構造



Oracle Real Application Clusters 環境での ADR

Oracle Real Application Clusters（RAC）環境では、ADR ベースは、ノードごとに独自のローカル記憶域に設定するか、共有記憶域に設定できます。共有記憶域に設定する利点は次のとおりです。

- ADRCI を使用して、すべてのインスタンスから集計された診断データを 1 つのレポートに表示できます。
- データ・リカバリ・アドバイザを使用して、破損したデータ・ブロック、破損または欠落したファイル、その他のデータ障害を診断して修復できます（Oracle RAC の場合、データ・リカバリ・アドバイザを使用するには共有記憶域が必要です）。

  データ・リカバリ・アドバイザの詳細は、『Oracle Database 2 日でデータベース管理者』を参照してください。

V$DIAG_INFO ビューを使用した ADR の場所の表示

V$DIAG_INFO ビューには、現在の Oracle Database インスタンスにとって重要な ADR の場所がすべてリストされます。

```
SELECT * FROM V$DIAG_INFO;

INST_ID NAME                 VALUE
------- -------------------- ----------------------------------------------------------
      1 Diag Enabled         TRUE
      1 ADR Base             /u01/oracle
      1 ADR Home             /u01/oracle/diag/rdbms/orclbi/orclbi
      1 Diag Trace           /u01/oracle/diag/rdbms/orclbi/orclbi/trace
      1 Diag Alert           /u01/oracle/diag/rdbms/orclbi/orclbi/alert
      1 Diag Incident        /u01/oracle/diag/rdbms/orclbi/orclbi/incident
      1 Diag Cdump           /u01/oracle/diag/rdbms/orclbi/orclbi/cdump
      1 Health Monitor       /u01/oracle/diag/rdbms/orclbi/orclbi/hm
      1 Default Trace File   /u01/oracle/diag/rdbms/orclbi/orclbi/trace/orcl_ora_22769.trc
      1 Active Problem Count 8
      1 Active Incident Count 20
```

次の表で、このビューに表示されるいくつかの情報を説明します。

表 8-3 V$DIAG_INFO ビューのデータ

名前	説明
ADR Base	ADR ベースのパス
ADR Home	現行データベース・インスタンスの ADR ホームのパス
Diag Trace	バックグラウンド・プロセスのトレース・ファイル、サーバー・プロセスのトレース・ファイル、SQL トレース・ファイル、およびテキスト形式のアラート・ログの場所
Diag Alert	XML 形式のアラート・ログの場所
Default Trace File	現行セッションのトレース・ファイルへのパス

問題の調査、レポートおよび解決

ここでは、Enterprise Manager サポート・ワークベンチ（サポート・ワークベンチ）を使用して、問題（クリティカル・エラー）を調査およびレポートし、場合によってはその問題を修復する方法について説明します。最初に、実行する必要がある一般的なタスクを要約した「ロードマップ」を示します。

注意： この項で説明するタスクは、すべて Enterprise Manager ベースです。すべてのタスク（または同等のタスク）は、`ADRCI` コマンドライン・ユーティリティ、PL/SQL パッケージ（`DBMS_HM`、`DBMS_SQLDIAG` など）、およびその他のソフトウェア・ツールを使用して実行できます。`ADRCI` ユーティリティの詳細は『Oracle Database ユーティリティ』を、PL/SQL パッケージの詳細は『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

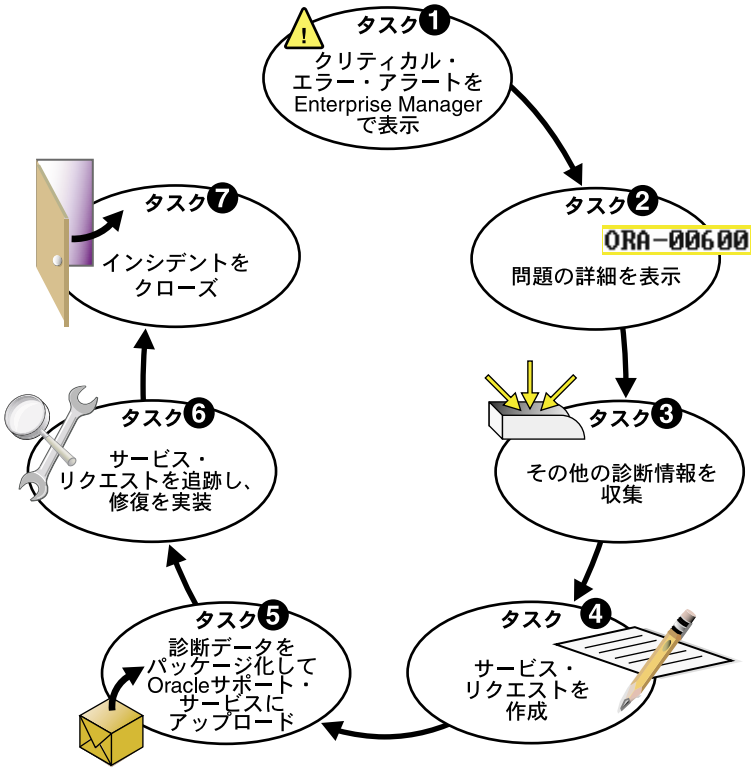
関連項目： 問題とその診断データの詳細は、8-2 ページの「Oracle Database の障害診断インフラストラクチャの概要」を参照してください。

問題の調査、レポートおよび解決のロードマップ

問題の調査は、Enterprise Manager の「サポート・ワークベンチ」ホームページから開始できます。ただし、データベース・ホームページでクリティカル・エラー・アラートの検討から開始するワークフローが一般的です。ここでは、このワークフローの概要について説明します。

図 8-2 に、問題を調査およびレポートし、場合によってはその問題を修復するためのタスクを示します。

図 8-2 問題の調査、レポートおよび解決のためのワークフロー



次に、各タスクについて説明します。各タスクの詳細は、後続の各項で説明します。

- **タスク 1: Enterprise Manager でのクリティカル・エラー・アラートの表示**（8-11 ページ）

  最初に、Enterprise Manager のデータベース・ホームページにアクセスし、クリティカル・エラー・アラートを検討します。詳細を表示するアラートを選択し、「問題の詳細」ページに進みます。

- **タスク 2: 問題の詳細の表示**（8-12 ページ）

  問題の詳細を検証し、その問題に対して記録されたすべてのインシデントのリストを表示します。自動的に実行されていたヘルス・チェックの結果を表示します。

- **タスク 3: （オプション）追加の診断情報の収集**（8-12 ページ）

  必要に応じて、追加のヘルス・チェックまたは他の診断を実行します。SQL 関連エラーの場合は、必要に応じて SQL テスト・ケース・ビルダーを起動して、SQL 問題に関連する必要なすべてのデータを収集し、Oracle サポート・サービスで問題を再現できる方法で情報をパッケージ化します。

- **タスク 4: （オプション）サービス・リクエストの作成**（8-13 ページ）

  必要に応じて、Oracle MetaLink を使用してサービス・リクエストを作成し、サービス・リクエスト番号を問題情報とともに記録します。このステップをスキップした場合は、後でサービス・リクエストを作成できます。または、サポート・ワークベンチでサービス・リクエストを作成できます。

- **タスク 5: 診断データのパッケージ化と Oracle サポート・サービスへのアップロード**（8-13 ページ）

  ガイド付きのワークフロー（ウィザード）を起動して、問題に対して収集した診断データを自動的にパッケージ化し、Oracle サポート・サービスにアップロードします。

- **タスク 6: サービス・リクエストの追跡と修復の実装**（8-15 ページ）

  必要に応じて、サポート・ワークベンチでサービス・リクエストのアクティビティ・ログを保持します。Oracle アドバイザを実行して、SQL エラーまたは破損データを修復します。

- **タスク 7: インシデントのクローズ**（8-16 ページ）

  問題に対する 1 つ、複数またはすべてのインシデントのステータスを「クローズ」に設定します。

関連項目： 8-16 ページ「Enterprise Manager サポート・ワークベンチを使用した問題の表示」

タスク 1: Enterprise Manager でのクリティカル・エラー・アラートの表示

問題（クリティカル・エラー）を調査するプロセスでは、最初にデータベース・ホームページでクリティカル・エラー・アラートを検討します。

クリティカル・エラー・アラートを表示する手順は、次のとおりです。

1. Enterprise Manager のデータベース・ホームページにアクセスします。

   Oracle Enterprise Manager Database Control の場合、手順については『Oracle Database 2 日でデータベース管理者』を参照してください。Oracle Enterprise Manager Grid Control の場合は、対象のデータベース・ターゲットに進みます。

2. 「アラート」セクションで、アラートの表を検証します。

   クリティカル・エラー・アラートは「重大度」列に赤の×印で表示され、「カテゴリ」列に「インシデント」と表示されます。

注意： 「アラート」表を表示するには、「アラート」ヘッダーの横にある表示 / 非表示アイコンをクリックします。

図 8-3 データベース・ホームページにある「アラート」表

3. 「メッセージ」列で、調査するクリティカル・エラー・アラートのメッセージをクリックします。

   「インシデント」ページまたは「データ障害」ページが表示されます。このページには次の内容が表示されます。

   - 問題に対するインシデントの数も含めた問題情報
   - 過去 24 時間に発生したクリティカル・エラーに関する「パフォーマンスとクリティカル・エラー」時間グラフ
   - 重大度、タイムスタンプおよびメッセージを含むアラート詳細
   - アラートをクリアしたり、アラートに関するコメントを記録するためのコントロール

4. 「パフォーマンスとクリティカル・エラー」時間グラフを検討し、パフォーマンス問題とクリティカル・エラーの間の関連をメモします。必要に応じて、アラートをクリアしたり、アラートに関するコメントを記録します。

5. 次のいずれかを実行します。
   - 調査しているクリティカル・エラー・アラートに関連する問題の詳細を表示する場合は、8-12 ページの「タスク 2: 問題の詳細の表示」に進みます。
   - 時間グラフに過去 24 時間にわたって多数の異なる問題が表示されているときに、すべての問題の要約を表示する場合は、次の手順を実行します。
     a. **「すべての問題の表示」**をクリックします。

        「サポート・ワークベンチ」ホームページが表示されます。
     b. 8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」で説明するように、問題とインシデントを表示します。
     c. 8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」で説明するように、1 つの問題を選択して問題の詳細を表示します。
     d. 8-12 ページの「タスク 3: （オプション）追加の診断情報の収集」、または 8-13 ページの「タスク 4: （オプション）サービス・リクエストの作成」に進みます。

タスク 2: 問題の詳細の表示

「問題の詳細」ページで調査を続行します。

問題の詳細を表示する手順は、次のとおりです。

1. 「インシデント」ページまたは「データ障害」ページで、**「問題の詳細の表示」**をクリックします。

   「問題の詳細」ページに「インシデント」サブページが表示されます。
2. （オプション）インシデントの詳細を表示するには、インシデントを選択して**「表示」**をクリックします。

   「インシデントの詳細」ページが表示されます。
3. （オプション）「インシデントの詳細」ページで、**「チェッカ結果」**をクリックして「チェッカ結果」サブページを表示します。

   このページには、クリティカル・エラーの検出時に自動的に実行されていたヘルス・チェックの結果が表示されます。

関連項目： 8-20 ページ「状態モニターを使用したヘルス・チェックの実行」

タスク 3: （オプション）追加の診断情報の収集

次のアクティビティを実行して、問題に関する追加の診断情報を収集します。この追加情報は、診断データに自動的に追加され、Oracle サポート・サービスにアップロードされます。これらのアクティビティを実行するかどうかを判断できない場合は、Oracle サポート・サービスの担当者に確認してください。

- 追加のヘルス・チェックを手動で起動します。

  8-20 ページの「状態モニターを使用したヘルス・チェックの実行」を参照してください。
- SQL テスト・ケース・ビルダーを起動します。

  手順については、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

タスク 4: （オプション）サービス・リクエストの作成

ここでは、Oracle サポート・サービスへのサービス・リクエストを作成し、サービス・リクエスト番号を問題情報とともに記録できます。このタスクをスキップした場合は、タスク 5 で、サポート・ワークベンチでドラフトのサービス・リクエストが自動的に作成されます。

サービス・リクエストを作成する手順は、次のとおりです。

1. 「問題の詳細」ページの「調査と解決」セクションで、**「Metalink にアクセス」**をクリックします。

   新規のブラウザ・ウィンドウに、Oracle MetaLink の「ログイン」と「登録」ページが表示されます。

   注意： 「問題の詳細」ページに戻る場合の手順については、8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」を参照してください。

2. Oracle MetaLink にログインし、通常の方法でサービス・リクエストを作成します。

   （オプション）サービス・リクエスト番号（SR#）は次のステップで使用するため、記録しておきます。

3. （オプション）「問題の詳細」ページに戻り、次の作業を実行します。

   a. 「サマリー」セクションで、SR# ラベルに隣接する**「編集」**ボタンをクリックします。

   b. オープンしたウィンドウで、SR# を入力して**「OK」**をクリックします。

   「問題の詳細」ページに SR# が記録されます。この情報は参照のみです。

タスク 5: 診断データのパッケージ化と Oracle サポート・サービスへのアップロード

このタスクでは、サポート・ワークベンチのクイック・パッケージング・プロセスを使用して、問題に関する診断情報をパッケージ化し、Oracle サポート・サービスにアップロードします。クイック・パッケージングには最小限のステップがあり、ガイド付きワークフロー（ウィザード）に編成されています。ウィザードを使用して、1 つの問題に対してインシデント・パッケージ（パッケージ）を作成し、そのパッケージから ZIP ファイルを作成してアップロードします。クイック・パッケージングでは、アップロードする診断情報を編集したりカスタマイズすることはできません。ただし、クイック・パッケージングを使用すると、直接的かつ簡単な方法で診断データをパッケージ化してアップロードできます。

アップロードする前に、機密データを編集したり診断情報から削除する場合、追加のユーザー・ファイル（アプリケーション構成ファイルやスクリプトなど）を添付する場合、あるいは他のカスタマイズを実行する場合は、カスタム・パッケージング・プロセスを使用する必要があります。このプロセスは手動操作が多いため、ステップ数も多くなります。手順については、8-32 ページの「カスタム・インシデント・パッケージの作成、編集およびアップロード」を参照してください。このタスク 5 の手順ではなく、カスタム・パッケージング・プロセスの手順に従う場合は、終了後に 8-15 ページの「タスク 6: サービス・リクエストの追跡と修復の実装」に進んでください。

注意： サポート・ワークベンチでは、Oracle Configuration Manager を使用して診断データをアップロードします。Oracle Configuration Manager がインストールされていなかったり、適切に構成されていないと、アップロードに失敗する場合があります。失敗した場合は、ファイルを Oracle サポート・サービスに手動でアップロードするように要求するメッセージが表示されます。アップロードは、Oracle MetaLink を使用して手動で実行できます。

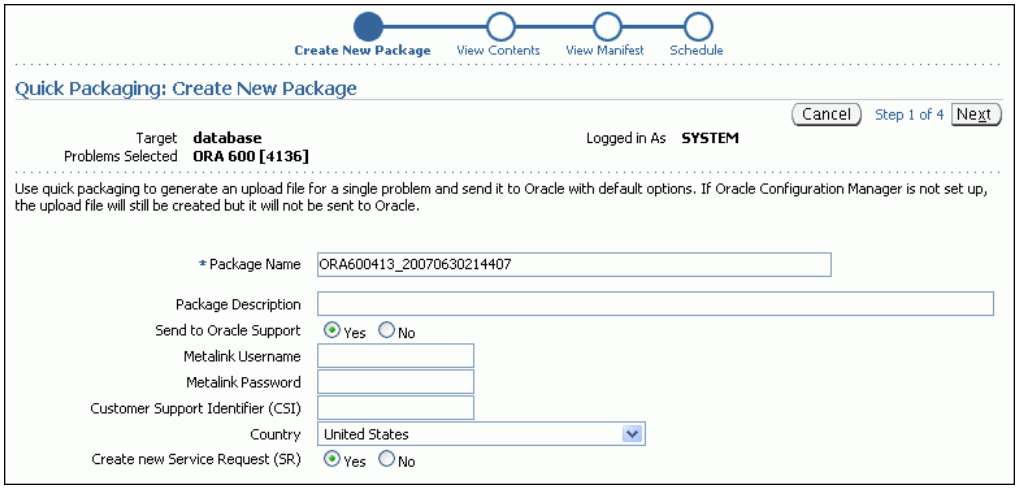
Oracle Configuration Manager の詳細は、『Oracle Configuration Manager インストレーションおよび管理ガイド』を参照してください。

診断データをパッケージ化して Oracle サポート・サービスにアップロードする手順は、次のとおりです。

1. 「問題の詳細」ページの「調査と解決」セクションで、**「クイック・パッケージ」**をクリックします。

   「クイック・パッケージ」ウィザードの「新規パッケージの作成」ページが表示されます。

   注意： 「問題の詳細」ページに戻る場合の手順については、8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」を参照してください。



2. 必要に応じて、パッケージ名と説明を入力します。

3. ページの残りのフィールドを入力します。問題に対してサービス・リクエストをすでに作成している場合は、「新規サービス・リクエスト (SR) の作成」の横にある**「いいえ」**を選択します。

   「はい」を選択すると、「クイック・パッケージング」ウィザードでドラフトのサービス・リクエストが作成されます。この場合は、後で Oracle MetaLink にログインして、サービス・リクエストの詳細を入力する必要があります。

4. **「次」**をクリックし、「クイック・パッケージング」ウィザードの残りのページに進みます。

   「クイック・パッケージング」ウィザードが完了した後も、作成されたパッケージはサポート・ワークベンチで使用できます。カスタム・パッケージング操作を使用してパッケージを変更し（新規インシデントの追加など）、後でパッケージを再アップロードできます。8-38 ページの「インシデント・パッケージの表示と変更」を参照してください。

タスク 6: サービス・リクエストの追跡と修復の実装

診断情報を Oracle サポート・サービスにアップロードした後は、様々なアクティビティを実行してサービス・リクエストを追跡し、追加の診断情報を収集して、修復を実装できます。次のようなアクティビティがあります。

- 問題情報への Oracle バグ番号の追加

  「問題の詳細」ページで、Bug# ラベルに隣接する**「編集」**ボタンをクリックします。この情報は参照のみです。

- 問題のアクティビティ・ログへのコメントの追加

  このアクティビティは、組織内の他の DBA と問題ステータスや履歴情報を共有する場合に実行します。たとえば、Oracle サポート・サービスとの対話の結果を記録できます。コメントを追加する手順は、次のとおりです。

  1. 8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」で説明するように、該当する問題の「問題の詳細」ページにアクセスします。
  2. **「アクティビティ・ログ」**をクリックして、「アクティビティ・ログ」サブページを表示します。
  3. 「コメント」フィールドにコメントを入力し、**「コメントの追加」**をクリックします。

     これで、コメントがアクティビティ・ログに記録されます。

- 新規に発生したインシデントのパッケージ化と再アップロード

  このアクティビティには、8-32 ページの「カスタム・インシデント・パッケージの作成、編集およびアップロード」で説明するように、カスタム・パッケージング方法を使用する必要があります。

- ヘルス・チェックの実行

  8-20 ページの「状態モニターを使用したヘルス・チェックの実行」を参照してください。

- 推奨された Oracle アドバイザの実行と修復の実装

  推奨されたアドバイザには、次のいずれかの方法でアクセスします。

  - **「問題の詳細」ページ**:「調査と解決」セクションの「セルフ・サービス」タブ
  - **「サポート・ワークベンチ」ホームページ**:「チェッカ結果」サブページ
  - **「インシデントの詳細」ページ**:「チェッカ結果」サブページ

  表 8-4 に、クリティカル・エラーの修復に役立つアドバイザを示します。

表 8-4 クリティカル・エラーの修復に役立つ Oracle アドバイザ

アドバイザ	対象となるクリティカル・エラー	参照先
データ・リカバリ・アドバイザ	破損ブロック、破損または欠落したファイル、その他のデータ障害	8-30 ページ「データ・リカバリ・アドバイザを使用したデータ破損の修復」
SQL 修復アドバイザ	SQL 文エラー	8-28 ページ「SQL 修復アドバイザを使用した SQL エラーの修復」

関連項目: 「インシデントの詳細」ページの「チェッカ結果」サブページを表示する手順については、8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」を参照してください。

タスク 7: インシデントのクローズ

特定のインシデントの処理が終了した場合は、そのインシデントをクローズできます。デフォルトでは、クローズしたインシデントは「問題の詳細」ページに表示されません。

すべてのインシデントは、クローズされているかどうかに関係なく、30 日後にパージされます。インシデントのパージは、「インシデントの詳細」ページで無効にできます。

インシデントをクローズする手順は、次のとおりです。

1. 「サポート・ワークベンチ」ホームページにアクセスします。

   手順については、8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」を参照してください。

2. 対象の問題を選択して**「表示」**をクリックします。

   「問題の詳細」ページが表示されます。

3. クローズするインシデントを選択して**「閉じる」**をクリックします。

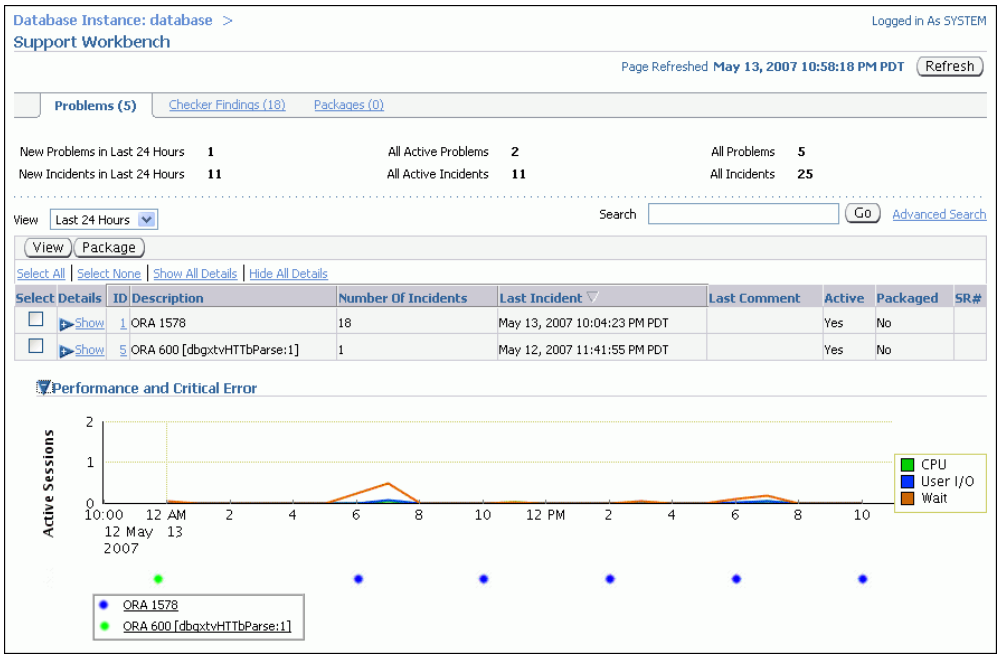
   確認ページが表示されます。

4. 必要に応じてコメントを入力し、**「OK」**をクリックします。

Enterprise Manager サポート・ワークベンチを使用した問題の表示

すべての問題を表示したり、特定の期間内に発生した問題のみを表示するには、Enterprise Manager の「サポート・ワークベンチ」ホームページ（8-16 ページの図 8-4）を使用します。

図 8-4 Enterprise Manager の「サポート・ワークベンチ」ホームページ



「サポート・ワークベンチ」ホームページにアクセスする手順は、次のとおりです。

1. Enterprise Manager のデータベース・ホームページにアクセスします。

   Oracle Enterprise Manager Database Control の場合、手順については『Oracle Database 2 日でデータベース管理者』を参照してください。Oracle Enterprise Manager Grid Control の場合は、対象のデータベース・ターゲットに進みます。

2. **「ソフトウェアとサポート」**をクリックして、「ソフトウェアとサポート」ページを表示します。

3. 「サポート」セクションで、**「サポート・ワークベンチ」**をクリックします。

   「サポート・ワークベンチ」ホームページに「問題」サブページが表示されます。デフォルトでは、過去 24 時間の問題が表示されます。

4. すべての問題を表示するには、「表示」リストから**「すべて」**を選択します。

5. （オプション）「パフォーマンスとクリティカル・エラー」セクションが非表示の場合は、セクション・ヘッダーに隣接する**表示 / 非表示**アイコンをクリックするとセクションが表示されます。

   このセクションでは、パフォーマンスの変化とインシデントの発生の関連を表示できます。

6. （オプション）「詳細」列の**「表示」**をクリックすると、問題に対するすべてのインシデントのリストが表示されます。リスト内のインシデント ID をクリックすると、「インシデントの詳細」ページが表示されます。

特定の問題の詳細を表示する手順は、次のとおりです。

1. 「サポート・ワークベンチ」ホームページで問題を選択し、**「表示」**をクリックします。

   「問題の詳細」ページに「インシデント」サブページが表示されます。「インシデント」サブページには、オープンしてダンプが生成された（つまり、フラッド制御されていない）すべてのインシデントが表示されます。

2. （オプション）オープン状態のインシデントとクローズしたインシデントの両方を表示するには、「ステータス」リストで**「すべてのインシデント」**を選択します。通常のインシデントとフラッド制御されているインシデントの両方を表示するには、「ダンプされたデータ」リストで**「すべてのインシデント」**を選択します。

3. （オプション）インシデントの詳細を表示するには、インシデントを選択して**「表示」**をクリックします。

   「インシデントの詳細」ページが表示されます。

4. （オプション）「インシデントの詳細」ページで、インシデントのチェッカ結果を表示するには、**「チェッカ結果」**をクリックします。

5. （オプション）「インシデントの詳細」ページで、インシデントに対して使用可能なユーザー操作を表示するには、**「追加の診断」**をクリックします。各ユーザー操作を実行して、インシデントまたはその問題に対する追加の診断を収集できます。

関連項目： 8-4 ページ「インシデントのフラッド制御」

ユーザー報告の問題の作成

システム生成の問題（データベースに内部的に生成されたクリティカル・エラー）は、自動診断リポジトリ（ADR）に自動的に追加されて、サポート・ワークベンチで追跡されます。サポート・ワークベンチでは、その問題に関する追加の診断データを収集して、診断データを Oracle サポート・サービスにアップロードし、場合によっては問題を解決できます。これらの作業はすべて、8-9 ページの「問題の調査、レポートおよび解決」で説明したように、使用しやすいワークフローで実行できます。

気づいた問題を同じワークフローで処理できるように、その問題を手動で ADR に追加する場合があります。このような問題の例には、自動データベース診断モニター（ADDM）で診断されなかったグローバル・データベースのパフォーマンスに関する問題があります。サポート・ワークベンチは、このようなユーザー報告の問題を作成して処理するメカニズムを備えています。

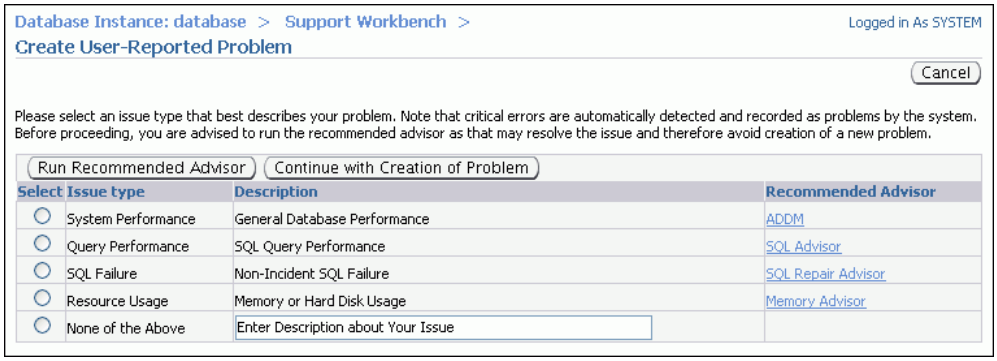
ユーザー報告の問題を作成する手順は、次のとおりです。

1. 「サポート・ワークベンチ」ホームページにアクセスします。

   手順については、8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」を参照してください。

2. 「関連リンク」で**「ユーザー報告の問題の作成」**をクリックします。

   「ユーザー報告の問題の作成」ページが表示されます。



3. リストされている問題のタイプのいずれかと問題が一致する場合は、その問題のタイプを選択して**「推奨アドバイザの実行」**をクリックし、Oracle アドバイザを実行して問題の解決を試みます。

4. 推奨アドバイザで問題が解決しない場合、またはアドバイザを実行しなかった場合は、次のいずれかを実行します。

   - リストされている問題のタイプのいずれかと問題が一致する場合は、その問題のタイプを選択して**「問題の作成の続行」**をクリックします。
   - リストされている問題のタイプのいずれかと問題が一致しない場合は、問題のタイプ**「上記のいずれでもない」**を選択して説明を入力し、**「問題の作成の続行」**をクリックします。

   「問題の詳細」ページが表示されます。

5. 「問題の詳細」ページの手順に従います。

   詳細は、8-9 ページの「問題の調査、レポートおよび解決」を参照してください。

関連項目： 問題と ADR の詳細は、8-2 ページの「Oracle Database の障害診断インフラストラクチャの概要」を参照してください。

アラート・ログの表示

アラート・ログは、テキスト・エディタ、Enterprise Manager または ADRCI ユーティリティを使用して表示できます。

Enterprise Manager を使用してアラート・ログを表示する手順は、次のとおりです。

1. Enterprise Manager のデータベース・ホームページにアクセスします。

   Oracle Enterprise Manager Database Control の場合、手順については『Oracle Database 2 日でデータベース管理者』を参照してください。Oracle Enterprise Manager Grid Control の場合は、対象のデータベース・ターゲットに進みます。

2. 「関連リンク」で**「アラート・ログの内容」**をクリックします。

   「アラート・ログの内容の表示」ページが表示されます。

3. 表示するエントリの数を選択して**「実行」**をクリックします。

テキスト・エディタを使用してアラート・ログを表示する手順は、次のとおりです。

1. SQL*Plus または SQL Developer などの問合せツールを使用して、データベースに接続します。
2. 8-8 ページの「V$DIAG_INFO ビューを使用した ADR の場所の表示」に示す `V$DIAG_INFO` ビューを問い合せます。
3. XML タグがないテキストのみのアラート・ログを表示する手順は、次のとおりです。
   a. `V$DIAG_INFO` 問合せ結果から `Diag Trace` エントリに対応するパスをメモし、ディレクトリをそのパスに変更します。
   b. テキスト・エディタを使用して、alert_*SID*.log ファイルをオープンします。
4. XML 形式のアラート・ログを表示する手順は、次のとおりです。
   a. `V$DIAG_INFO` 問合せ結果から `Diag Alert` エントリに対応するパスをメモし、ディレクトリをそのパスに変更します。
   b. テキスト・エディタを使用して、log.xml ファイルをオープンします。

関連項目： ADRCI ユーティリティを使用してテキスト形式（XML タグは削除されます）のアラート・ログを表示し、そのアラート・ログに対して問合せを実行する方法については、『Oracle Database ユーティリティ』を参照してください。

トレース・ファイルの検索

トレース・ファイルは、自動診断リポジトリ（ADR）内の各 ADR ホーム下にある trace ディレクトリに格納されます。このディレクトリ内の個々のトレース・ファイルを検索するには、データ・ディクショナリ・ビューを使用します。たとえば、現行セッションのトレース・ファイルへのパス、または各 Oracle Database プロセスのトレース・ファイルへのパスを検索できます。

現行セッションのトレース・ファイルを検索する手順は、次のとおりです。

- 次の問合せを発行します。

```
SELECT VALUE FROM V$DIAG_INFO WHERE NAME = 'Default Trace File';
```

トレース・ファイルへのフルパスが返されます。

現行インスタンスのすべてのトレース・ファイルを検索する手順は、次のとおりです。

- 次の問合せを発行します。

```
SELECT VALUE FROM V$DIAG_INFO WHERE NAME = 'Diag Trace';
```

現在のインスタンスの ADR トレース・ディレクトリへのパスが返されます。

各 Oracle Database プロセスのトレース・ファイルを特定する手順は、次のとおりです。

- 次の問合せを発行します。

```
SELECT PID, PROGRAM, TRACEFILE FROM V$PROCESS;
```

関連項目：

- 8-7 ページ「自動診断リポジトリの構造、内容および場所」
- 『Oracle Database ユーティリティ』の ADRCI SHOW TRACEFILE コマンドに関する項

状態モニターを使用したヘルス・チェックの実行

ここでは、状態モニターとその使用方法を説明します。この項の内容は、次のとおりです。

状態モニターの概要

リリース 11*g* 以降の Oracle Database は、データベースに対して診断チェックを実行する、状態モニターと呼ばれるフレームワークを備えています。

状態モニター・チェックの概要

状態モニター・チェック（チェッカ、ヘルス・チェックまたはチェックとも呼ばれます）では、データベースの様々なレイヤーとコンポーネントが検証されます。ヘルス・チェックでは、ファイルの破損、物理ブロックおよび論理ブロックの破損、UNDO および REDO の破損、データ・ディクショナリの破損などが検出されます。また、ヘルス・チェックの結果のレポートが生成され、多くの場合、問題を解決するための推奨事項が示されます。ヘルス・チェックは次の 2 つの方法で実行できます。

- **リアクティブ**: 障害診断インフラストラクチャでは、クリティカル・エラーの内容に応じてヘルス・チェックを自動的に実行できます。
- **手動**: DBA は、DBMS_HM PL/SQL パッケージまたは Enterprise Manager インタフェースのいずれかを使用して、ヘルス・チェックを手動で実行できます。チェッカは必要に応じて定期的に実行できます。また、サービス・リクエストの作業中に、Oracle サポート・サービスからチェッカの実行を要請される場合もあります。

状態モニター・チェックでは、結果、推奨事項およびその他の情報が自動診断リポジトリ（ADR）に格納されます。

ヘルス・チェックは次の 2 つのモードで実行できます。

- **DB オンライン**・モードは、データベースがオープン状態（つまり、OPEN モードまたは MOUNT モード）のときは、チェックを実行できることを意味します。
- **DB オフライン**・モードは、インスタンスは使用可能だがデータベース自体がクローズ（つまり、NOMOUNT モード）している場合にチェックを実行できることを意味します。

DB オンライン・モードでは、すべてのヘルス・チェックを実行できます。DB オフライン・モードで使用できるのは、REDO の整合性チェックと DB 構造の整合性チェックのみです。

関連項目： 8-4 ページ「自動診断リポジトリ（ADR）」

ヘルス・チェックのタイプ

状態モニターでは次のチェックが実行されます。

- **DB 構造の整合性チェック**: このチェックではデータベース・ファイルの整合性が検証され、これらのファイルがアクセス不可、破損状態または不整合の場合はエラーが報告されます。データベースがマウント・モードまたはオープン・モードの場合は、制御ファイルに列記されたログ・ファイルとデータファイルがチェックされます。データベースが NOMOUNT モードの場合は、制御ファイルのみがチェックされます。
- **データ・ブロックの整合性チェック**: このチェックではチェックサム・エラー、先頭 / 末尾の不一致、ブロック内の論理的不整合などのディスク・イメージ・ブロック破損が検出されます。ほとんどの破損はブロック・メディア・リカバリを使用して修復できます。破損ブロック情報は V$DATABASE_BLOCK_CORRUPTION ビューでも取得されます。このチェックではブロック間またはセグメント間の破損は検出されません。
- **REDO の整合性チェック**: このチェックではアクセシビリティと破損について REDO ログの内容がスキャンされ、可能な場合はアーカイブ・ログもスキャンされます。REDO の整合性チェックではアーカイブ・ログや REDO の破損などのエラーが報告されます。
- **UNDO セグメントの整合性チェック**: このチェックでは、論理的な UNDO 破損が検出されます。UNDO 破損の場所を特定した後、このチェックでは PMON および SMON を使用して、破損したトランザクションのリカバリを試みます。このリカバリに失敗した場合、状態モニターは破損の情報を V$CORRUPT_XID_LIST に格納します。ほとんどの UNDO 破損は、コミットを強制実行することで解決できます。
- **トランザクションの整合性チェック**: このチェックは、1 つの特定トランザクションのみがチェックされることを除いて、UNDO セグメントの整合性チェックと同じです。
- **ディクショナリの整合性チェック**: このチェックでは、tab$、col$ などのコア・ディクショナリ・オブジェクトの整合性が検証されます。次のチェックが実行されます。
  - 各ディクショナリ・オブジェクトに対するディクショナリ・エントリの内容が確認されます。
  - 行間レベルのチェックが実行され、ディクショナリの行で論理的制約が適用されていることが確認されます。
  - オブジェクト関係のチェックが実行され、ディクショナリ・オブジェクト間で親子関係が規定されていることが確認されます。

ディクショナリの整合性チェックは、次のディクショナリ・オブジェクトに対して実行されます。

tab$、clu$、fet$、uet$、seg$、undo$、ts$、file$、obj$、ind$、icol$、col$、user$、con$、cdef$、ccol$、bootstrap$、objauth$、ugroup$、tsq$、syn$、view$、typed_view$、superobj$、seq$、lob$、coltype$、subcoltype$、ntab$、refcon$、opqtype$、dependency$、access$、viewcon$、icoldep$、dual$、sysauth$、objpriv$、defrole$ および ecol$

ヘルス・チェックの手動実行

状態モニターでは、次の 2 つの方法でヘルス・チェックを手動で実行できます。

- DBMS_HM PL/SQL パッケージを使用する方法
- 「アドバイザ・セントラル」ページの「チェッカ」サブページにある Enterprise Manager インタフェースを使用する方法

DBMS_HM PL/SQL パッケージを使用したヘルス・チェックの実行

ヘルス・チェックを実行するための DBMS_HM プロシージャは RUN_CHECK です。RUN_CHECK をコールするには、次のようにチェック名と実行名を指定します。

```
BEGIN
    DBMS_HM.RUN_CHECK('Dictionary Integrity Check', 'my_run');
END;
```

ヘルス・チェック名のリストを取得するには、次の問合せを実行します。

```
SELECT name FROM v$hm_check WHERE internal_check='N';

NAME
----------------------------------------------------------------
DB Structure Integrity Check
Data Block Integrity Check
Redo Integrity Check
Transaction Integrity Check
Undo Segment Integrity Check
Dictionary Integrity Check
```

ほとんどのヘルス・チェックに入力パラメータを指定できます。パラメータ名と説明は V$HM_CHECK_PARAM ビューを使用して表示できます。一部のパラメータは必須ですが、オプションのパラメータもあります。オプションのパラメータを省略すると、デフォルトが使用されます。次の問合せでは、すべてのヘルス・チェックのパラメータ情報が表示されます。

```
SELECT c.name check_name, p.name parameter_name, p.type,
p.default_value, p.description
FROM v$hm_check_param p, v$hm_check c
WHERE p.check_id = c.id and c.internal_check = 'N'
ORDER BY c.name;
```

入力パラメータは、セミコロン（;）で区切られた名前 / 値のペアとして input_params 引数で渡されます。次の例では、トランザクション ID をパラメータとして「トランザクションの整合性チェック」に渡す方法を示します。

```
BEGIN
DBMS_HM.RUN_CHECK (
   check_name   => 'Transaction Integrity Check',
   run_name     => 'my_run',
   input_params => 'TXN_ID=7.33.2');
END;
```

関連項目：

- 8-27 ページ「ヘルス・チェック・パラメータの参考情報」
- DBMS_HM の使用例の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

Enterprise Manager を使用したヘルス・チェックの実行

Enterprise Manager は、状態モニター・チェッカを実行するためのインタフェースを備えています。

Enterprise Manager を使用して状態モニター・チェッカを実行する手順は、次のとおりです。

1. データベース・ホームページにある「関連リンク」セクションで、**「アドバイザ・セントラル」**をクリックします。
2. **「チェッカ」**をクリックして「チェッカ」サブページを表示します。
3. 「チェッカ」セクションで、実行するチェッカをクリックします。
4. 入力パラメータの値を入力するか、オプション・パラメータの場合は空白のままにしてデフォルトを使用します。
5. **「実行」**をクリックし、パラメータを確認して**「実行」**を再度クリックします。

チェッカ・レポートの表示

チェッカを実行した後は、その実行内容のレポートを表示できます。レポートには、結果、推奨事項およびその他の情報が含まれます。このレポートは、Enterprise Manager、ADRCI ユーティリティまたは DBMS_HM PL/SQL パッケージを使用して表示できます。次の表に、各表示方法で使用可能なレポート形式を示します。

レポートの表示方法	使用可能なレポート形式
Enterprise Manager	HTML
DBMS_HM PL/SQL パッケージ	HTML、XML およびテキスト
ADRCI ユーティリティ	XML

チェッカの実行結果（結果、推奨事項およびその他の情報）は ADR に格納されますが、レポートはすぐに生成されません。DBMS_HM PL/SQL パッケージまたは Enterprise Manager を使用してレポートを要求したときに、レポートがまだ生成されていない場合は、最初に ADR 内のチェッカ実行データからレポートが生成され、現行インスタンス用の ADR ホームの HM サブディレクトリに XML 形式でレポート・ファイルとして格納された後に、レポートが表示されます。レポートがすでに生成されている場合は、そのレポートが表示されます。ADRCI ユーティリティを使用する場合、レポートがまだ生成されていないときは、最初にレポート・ファイルを生成するためのコマンドを実行し、次にその内容を表示するためのコマンドを実行する必要があります。

チェッカ・レポートは、Enterprise Manager を使用して表示することをお薦めします。次の各項では、各表示方法の手順を説明します。

- Enterprise Manager を使用したレポートの表示
- DBMS_HM を使用したレポートの表示
- ADRCI ユーティリティを使用したレポートの表示

関連項目： 8-4 ページ「自動診断リポジトリ（ADR）」

Enterprise Manager を使用したレポートの表示

Enterprise Manager を使用して、特定のチェッカ実行の状態モニター・レポートと結果を表示できます。

Enterprise Manager を使用して実行結果を表示する手順は、次のとおりです。

1. データベース・ホームページにアクセスします。

   Oracle Enterprise Manager Database Control の場合、手順については『Oracle Database 2日でデータベース管理者』を参照してください。Oracle Enterprise Manager Grid Control の場合は、対象のデータベース・ターゲットに進みます。

2. 「関連リンク」セクションで、**「アドバイザ・セントラル」**をクリックします。
3. **「チェッカ」**をクリックして「チェッカ」サブページを表示します。
4. 表示するチェッカ実行の実行名をクリックします。

   「実行の詳細」ページにチェッカ実行の結果が表示されます。

5. **「実行」**をクリックして「実行」サブページを表示します。

   チェッカ実行に関する詳細な情報が表示されます。

6. **「レポートの表示」**をクリックして、チェッカ実行のレポートを表示します。

   レポートが新規のブラウザ・ウィンドウに表示されます。

DBMS_HM を使用したレポートの表示

状態モニター・チェッカ・レポートは、DBMS_HM パッケージ・ファンクション GET_RUN_REPORT を使用して表示できます。このファンクションを使用すると、HTML、XML またはテキスト形式のレポートを要求できます。デフォルトはテキスト形式で、次に SQL*Plus の例を示します。

```
SET LONG 100000
SET LONGCHUNKSIZE 1000
SET PAGESIZE 1000
SET LINESIZE 512
SELECT DBMS_HM.GET_RUN_REPORT('HM_RUN_1061') FROM DUAL;

DBMS_HM.GET_RUN_REPORT('HM_RUN_1061')
------------------------------------------------------------------------------

 Run Name                      : HM_RUN_1061
 Run Id                        : 1061
 Check Name                    : Data Block Integrity Check
 Mode                          : REACTIVE
 Status                        : COMPLETED
 Start Time                    : 2007-05-12 22:11:02.032292 -07:00
 End Time                      : 2007-05-12 22:11:20.835135 -07:00
 Error Encountered             : 0
 Source Incident Id            : 7418
 Number of Incidents Created   : 0

Input Paramters for the Run
 BLC_DF_NUM=1
 BLC_BL_NUM=64349

Run Findings And Recommendations
 Finding
 Finding Name  : Media Block Corruption
 Finding ID    : 1065
 Type          : FAILURE
 Status        : OPEN
 Priority      : HIGH
 Message       : Block 64349 in datafile 1:
```

```
                '/ade/sfogel_emdb/oracle/dbs/t_db1.f' is media corrupt
Message         : Object BMRTEST1 owned by SYS might be unavailable
Finding
Finding Name    : Media Block Corruption
Finding ID      : 1071
Type            : FAILURE
Status          : OPEN
Priority        : HIGH
Message         : Block 64351 in datafile 1:
                '/ade/sfogel_emdb/oracle/dbs/t_db1.f' is media corrupt
Message         : Object BMRTEST2 owned by SYS might be unavailable
```

関連項目： DBMS_HM パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ADRCI ユーティリティを使用したレポートの表示

ADRCI ユーティリティを使用して、状態モニター・チェッカ・レポートを作成および表示できます。

ADRCI を使用してチェッカ・レポートを作成および表示する手順は、次のとおりです。

1. オペレーティング・システム環境変数（ORACLE_HOME など）が適切に設定されていることを確認し、次のコマンドをオペレーティング・システムのコマンド・プロンプトに入力します。

   ```
   ADRCI
   ```

   ユーティリティが起動し、次のプロンプトが表示されます。

   ```
   adrci>>
   ```

   必要に応じて、現行の ADR ホームを変更できます。すべての ADR ホームをリストするには SHOW HOMES コマンドを、現行の ADR ホームを変更するには SET HOMEPATH コマンドを使用します。詳細は、『Oracle Database ユーティリティ』を参照してください。

2. 次のコマンドを入力します。

   ```
   show hm_run
   ```

   このコマンドによって、ADR リポジトリに登録されている（V$HM_RUN に格納された）すべてのチェッカ実行がリストされます。

3. レポートを作成するチェッカ実行を検索し、チェッカ実行名をメモします。このチェッカ実行のレポートがすでに生成されている場合は、REPORT_FILE フィールドにファイル名が表示されます。レポートが生成されていない場合は、次のコマンドを実行してレポートを生成します。

   ```
   create report hm_run run_name
   ```

4. レポートを表示するには、次のコマンドを入力します。

   ```
   show report hm_run run_name
   ```

関連項目： 8-4 ページ「自動診断リポジトリ（ADR）」

状態モニターのビュー

チェッカ・レポートを要求するかわりに、レポートが作成された ADR データを直接問い合せると、特定のチェッカ実行の結果を表示できます。このデータは、V$HM_RUN、V$HM_FINDING および V$HM_RECOMMENDATION の各ビューを使用して表示できます。

次の例では、V$HM_RUN ビューを問い合せて、チェッカ実行の履歴を表示します。

```
SELECT run_id, name, check_name, run_mode, src_incident FROM v$hm_run;

    RUN_ID NAME         CHECK_NAME                        RUN_MODE SRC_INCIDENT
---------- ------------ --------------------------------- -------- ------------
         1 HM_RUN_1     DB Structure Integrity Check      REACTIVE            0
       101 HM_RUN_101   Transaction Integrity Check       REACTIVE         6073
       121 TXNCHK       Transaction Integrity Check       MANUAL              0
       181 HMR_tab$     Dictionary Integrity Check        MANUAL              0
           .
           .
           .
       981 Proct_ts$    Dictionary Integrity Check        MANUAL              0
      1041 HM_RUN_1041  DB Structure Integrity Check      REACTIVE            0
      1061 HM_RUN_1061  Data Block Integrity Check        REACTIVE         7418
```

次の例では、RUN_ID 1061 を指定して V$HM_FINDING ビューを問い合せ、リアクティブなデータ・ブロック・チェックの結果詳細を取得します。

```
SELECT type, description FROM v$hm_finding WHERE run_id = 1061;

TYPE          DESCRIPTION
------------- ----------------------------------------
FAILURE       Block 64349 in datafile 1: '/ade/sfogel_e
              mdb/oracle/dbs/t_db1.f' is media corrupt

FAILURE       Block 64351 in datafile 1: '/ade/sfogel_e
              mdb/oracle/dbs/t_db1.f' is media corrupt
```

関連項目：

- 8-21 ページ「ヘルス・チェックのタイプ」
- V$HM_* ビューの詳細は、『Oracle Database リファレンス』を参照してください。

ヘルス・チェック・パラメータの参考情報

次の表では、パラメータが必要なヘルス・チェック用のパラメータを説明します。デフォルト値が（なし）のパラメータは必須です。

表 8-5「データ・ブロックの整合性チェック」のパラメータ

パラメータ名	タイプ	デフォルト値	説明
BLC_DF_NUM	数値	（なし）	ブロック・データ・ファイル番号
BLC_BL_NUM	数値	（なし）	データ・ブロック番号

表 8-6「REDO の整合性チェック」のパラメータ

パラメータ名	タイプ	デフォルト値	説明
SCN_TEXT	テキスト	0	最新の良好な REDO の SCN（既知の場合）

表 8-7「UNDO セグメントの整合性チェック」のパラメータ

パラメータ名	タイプ	デフォルト値	説明
USN_NUMBER	テキスト	（なし）	UNDO セグメント番号

表 8-8「トランザクションの整合性チェック」のパラメータ

パラメータ名	タイプ	デフォルト値	説明
TXN_ID	テキスト	（なし）	トランザクション ID

表 8-9「ディクショナリの整合性チェック」のパラメータ

パラメータ名	タイプ	デフォルト値	説明
CHECK_MASK	テキスト	ALL	使用可能な値は、次のとおりです。 ■ COLUMN_CHECKS: 列のチェックのみを実行します。コア・テーブル内の列レベルの制約を検証します。 ■ ROW_CHECKS: 行のチェックのみを実行します。コア・テーブル内の行レベルの制約を検証します。 ■ REFERENTIAL_CHECKS: 参照チェックのみを実行します。コア・テーブル内の参照の制約を検証します。 ■ ALL: すべてのチェックを実行します。
TABLE_NAME	テキスト	ALL_CORE_TABLES	チェックする単一のコア・テーブルの名前。省略すると、すべてのコア・テーブルがチェックされます。

SQL 修復アドバイザを使用した SQL エラーの修復

ほとんど発生しませんが、SQL 文がクリティカル・エラーで失敗した場合は、SQL 修復アドバイザを実行して失敗した文を修復できます。

この項の内容は、次のとおりです。

SQL 修復アドバイザの概要

SQL 修復アドバイザは、SQL 文がクリティカル・エラーで失敗した後に実行します。このアドバイザは文を分析し、多くの場合、文を修復するためのパッチを推奨します。推奨事項を実装すると、適用した SQL パッチによってエラーが回避されます。これは、問合せオプティマイザによって、今後の実行に対して別の実行計画が選択されるためです。

SQL 修復アドバイザの実行

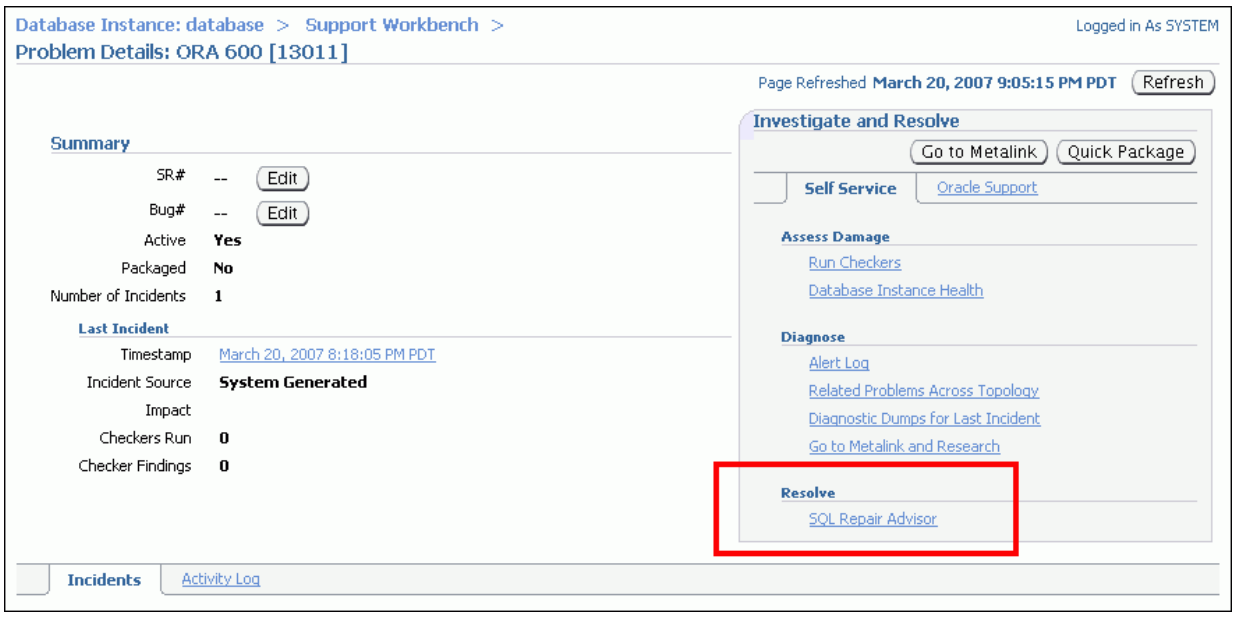
SQL 修復アドバイザは、サポート・ワークベンチの「問題の詳細」ページから実行します。この項の説明は、SQL 文によるクリティカル・エラーの通知をすでに受け取り、8-9 ページの「問題の調査、レポートおよび解決」で説明したワークフローに従っていることを前提としています。

SQL 修復アドバイザを実行する手順は、次のとおりです。

1. 失敗した SQL 文に関連する問題の「問題の詳細」ページにアクセスします。

   手順については、8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」を参照してください。

2. 「調査と解決」セクションの「セルフ・サービス」タブにある「解決」ヘッダーで、**「SQL 修復アドバイザ」**をクリックします。

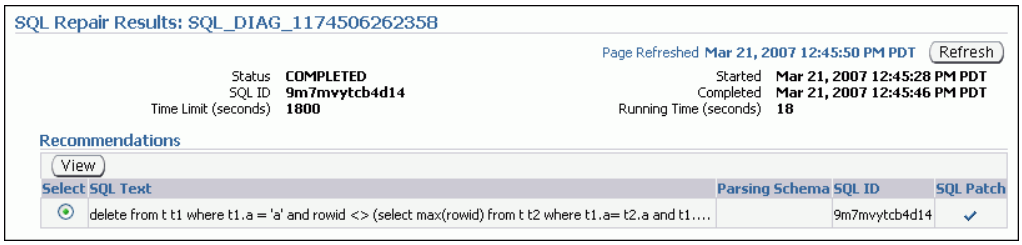


「SQL 修復アドバイザ」ページが表示されます。

3. タスク名（オプション）を入力してアドバイザ・タスクの時間制限（オプション）を設定し、設定を調整してアドバイザを即時に実行するか、後の日時に実行するかをスケジュールします。

4. **「発行」**をクリックします。

   「処理中」ページが表示されます。その少し後に「SQL 修復結果」ページが表示されます。



   「SQL パッチ」列のチェック・マークは、推奨事項があることを示します。この列にチェック・マークがない場合は、SQL 修復アドバイザが SQL 文に対してパッチを提示できなかったことを意味します。

5. 推奨事項がある場合は、**「表示」**をクリックして推奨事項を表示します。

   「修復の推奨」ページに、文に対する推奨パッチが表示されます。

6. **「実装」**をクリックします。

   「SQL 修復結果」ページに戻り、確認メッセージが表示されます。

7. （オプション）**「SQL ワークシートを使用して検証」**をクリックして、SQL ワークシートの文を実行し、パッチによって文が正常に修復したことを検証します。

SQL パッチの表示、無効化または削除

SQL 修復アドバイザを使用して SQL パッチを適用した後は、その SQL パッチを表示してパッチが適用されたことを確認したり、パッチを無効化または削除できます。パッチを削除する理由の 1 つに、後続リリースの Oracle Database をインストールし、パッチを適用した SQL 文でエラーが発生した不具合が修正されている場合があります。

SQL パッチを表示、無効化または削除する手順は、次のとおりです。

1. Enterprise Manager のデータベース・ホームページにアクセスします。

   Oracle Enterprise Manager Database Control の場合、手順については『Oracle Database 2 日でデータベース管理者』を参照してください。Oracle Enterprise Manager Grid Control の場合は、対象のデータベース・ターゲットに進みます。

2. ページの上部にある**「サーバー」**をクリックして「サーバー」ページを表示します。

3. 「問合せオプティマイザ」セクションで、**「SQL 計画管理」**をクリックします。

   「SQL 計画管理」ページが表示されます。このページの詳細は、オンライン・ヘルプを参照してください。

4. ページの上部にある**「SQL パッチ」**をクリックして「SQL パッチ」サブページを表示します。

   「SQL パッチ」サブページに、データベース内のすべての SQL パッチが表示されます。

5. 関連する SQL テキストを検証して、特定のパッチを検索します。

   SQL テキストをクリックして、その文の完全なテキストを表示します。

6. パッチを無効化するには、パッチを選択して**「無効化」**をクリックします。

   確認メッセージが表示され、パッチのステータスが `DISABLED` に変更されます。後でパッチを再び有効にするには、パッチを選択して**「有効化」**をクリックします。

7. パッチを削除するには、パッチを選択して**「削除」**をクリックします。

   確認メッセージが表示されます。

関連項目： 8-28 ページ「SQL 修復アドバイザの概要」

データ・リカバリ・アドバイザを使用したデータ破損の修復

データ・ブロックの破損、UNDO 破損、データ・ディクショナリの破損などを修復するには、データ・リカバリ・アドバイザを使用します。データ・リカバリ・アドバイザは Enterprise Manager サポート・ワークベンチ（サポート・ワークベンチ）、状態モニターおよび RMAN ユーティリティに統合され、データ破損の問題の表示、各問題の程度の評価（クリティカル、高優先度、低優先度）、問題の影響の説明、修復オプションの推奨、顧客が選択したオプションの実行可能性チェック、および修復プロセスの自動化を実行します。

データ・リカバリ・アドバイザの使用方法の詳細は、『Oracle Database 2 日でデータベース管理者』を参照してください。ここでは、サポート・ワークベンチからアドバイザにアクセスする方法をいくつか説明します。

データ・リカバリ・アドバイザは、次の情報を表示したときにサポート・ワークベンチによって自動的に推奨され、サポート・ワークベンチからアクセスできます。

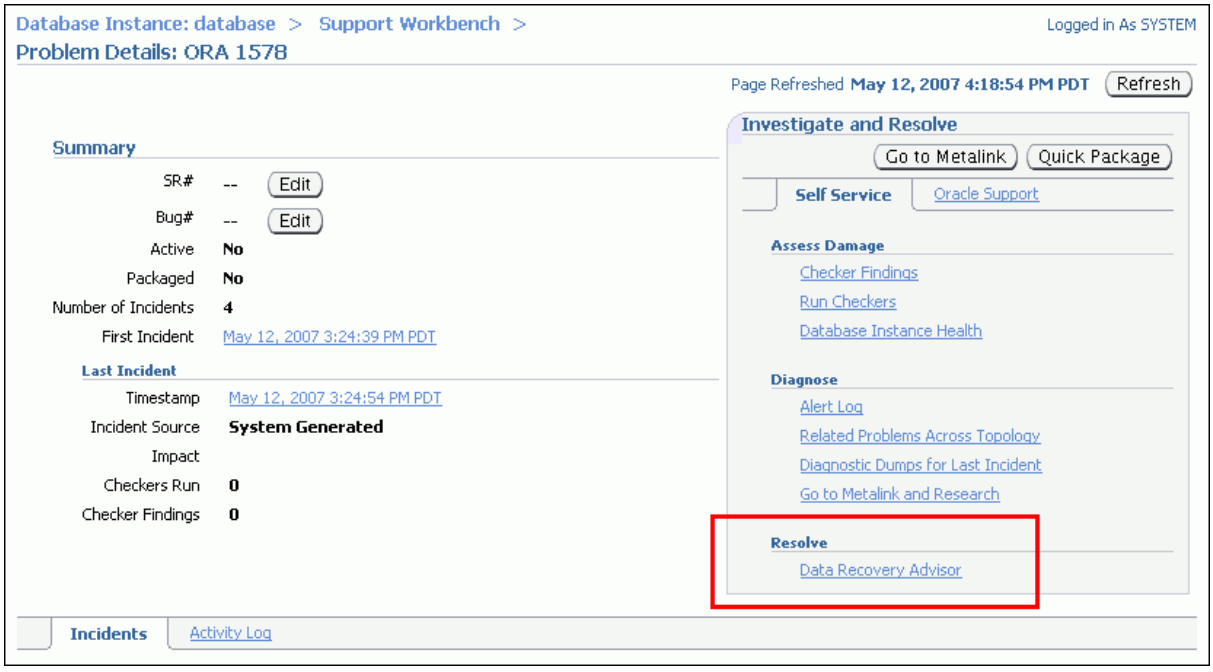
- データ破損またはその他のデータ障害に関連する問題の詳細
- データ破損またはその他のデータ障害に関連するヘルス・チェック結果

データ・リカバリ・アドバイザは「アドバイザ・セントラル」ページからも使用できます。このページへのリンクは、データベース・ホームページおよび「パフォーマンス」ページの「関連リンク」セクションにあります。

注意： データ・リカバリ・アドバイザを使用できるのは、SYSDBA として接続している場合のみです。

サポート・ワークベンチからデータ・リカバリ・アドバイザにアクセスする方法は次のとおりです。

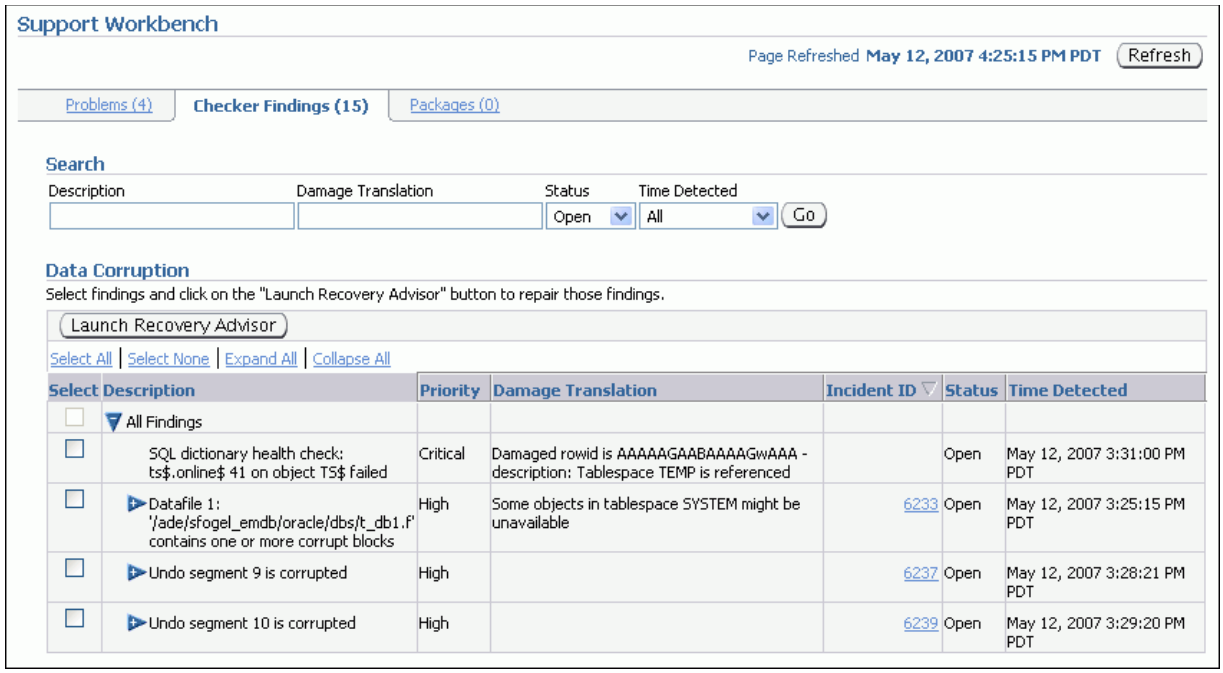
- 「問題の詳細」ページからのアクセス



「調査と解決」セクションの**「データ・リカバリ・アドバイザ」**リンクをクリックします。

このページへのアクセス手順については、8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」を参照してください。

- 「サポート・ワークベンチ」ホームページの「チェッカ結果」サブページからのアクセス



1 つ以上のデータ破損結果を選択して、**「リカバリ・アドバイザの起動」**をクリックします。

「サポート・ワークベンチ」ホームページへのアクセス手順については、8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」を参照してください。

関連項目： データ・リカバリ・アドバイザの実行手順については、『Oracle Database 2 日でデータベース管理者』を参照してください。

カスタム・インシデント・パッケージの作成、編集およびアップロード

ここでは、インシデント・パッケージに関するバックグラウンド情報を提供し、Enterprise Manager サポート・ワークベンチ（サポート・ワークベンチ）のカスタム・パッケージング・プロセスを使用してカスタマイズしたパッケージを作成、変更およびアップロードする方法を説明します。この項の内容は、次のとおりです。

- インシデント・パッケージの概要
- カスタム・パッケージングを使用した問題のパッケージ化とアップロード
- インシデント・パッケージの表示と変更
- インシデント・パッケージのプリファレンスの設定

関連項目： 8-2 ページ「Oracle Database の障害診断インフラストラクチャの概要」

インシデント・パッケージの概要

診断データをカスタマイズして Oracle サポート・サービスにアップロードするには、最初に、インシデント・パッケージ（パッケージ）と呼ばれる中間論理構造にデータを収集します。**パッケージ**は自動診断リポジトリ（ADR）に格納されているメタデータの集合で、ADR 内外の診断データファイルやその他のファイルを指し示します。パッケージを作成するときは、そのパッケージに追加する問題を 1 つ以上選択します。次に、サポート・ワークベンチによって、選択した問題に関連する問題情報、インシデント情報および診断データ（トレース・ファイル、ダンプなど）がパッケージに自動的に追加されます。1 つの問題に対して多数のインシデント（同じ問題の多数の発生）が存在する場合があるため、デフォルトでは、各問題の最初の 3 つと最後の 3 つのインシデントのみがパッケージに追加され、発生から 90 日を超えるインシデントは除外されます。このデフォルトの数は、サポート・ワークベンチの「インシデント・パッケージング構成」ページで変更できます。

パッケージが作成された後は、任意のタイプの外部ファイルを追加したり、選択したファイルをパッケージから削除できます。また、パッケージ内の選択したファイルを編集して機密データを削除することもできます。パッケージ・コンテンツを追加または削除すると、パッケージのメタデータのみが変更されます。

診断データを Oracle サポート・サービスにアップロードする準備ができたら、最初に、パッケージのメタデータで参照されるすべてのファイルを含めて ZIP ファイルを作成します。次に、Oracle Configuration Manager を使用してその ZIP ファイルをアップロードします。

注意： Oracle Configuration Manager をインストールしていないか、適切に構成していない場合は、Oracle MetaLink を使用して ZIP ファイルを手動でアップロードする必要があります。

Oracle Configuration Manager の詳細は、『Oracle Configuration Manager Installation and Administration Guide』を参照してください。

次の各項では、パッケージの詳細を説明します。

- インシデント・パッケージ内の関係付けられた診断データの概要
- クイック・パッケージングとカスタム・パッケージングの概要

関連項目：

- 8-34 ページ「カスタム・パッケージングを使用した問題のパッケージ化とアップロード」
- 8-38 ページ「インシデント・パッケージの表示と変更」

インシデント・パッケージ内の関係付けられた診断データの概要

問題の診断能力を向上させるには、問題に直接関連する診断データに加えて、その診断データに関連したデータの検証が必要になる場合があります。診断データは、時間、プロセス ID またはその他の基準によって関係付けることができます。たとえば、インシデントの検証では、そのインシデントの 5 分後に発生したインシデントの検証も役立つ場合があります。同様に、インシデントの診断データには、インシデント発生時に実行されていた Oracle Database プロセスのトレース・ファイルが含まれていることは明確ですが、元のプロセスに関連する他のプロセスのトレース・ファイルを含めることが役に立つ場合もあります。

このため、問題とそれに関連するインシデントがパッケージに追加されるときは、同時に、関連するトレース・ファイルとともに関係付けられたインシデントが追加されます。

パッケージの物理ファイルを作成する過程で、サポート・ワークベンチはインシデント・パッケージング・サービスを要求してパッケージをファイナライズします。**ファイナライズ**とは、パッケージ内のインシデントに時間で関係付けられた追加のトレース・ファイル、および他の診断情報（アラート・ログ、ヘルス・チェッカ・レポート、SQL テスト・ケース、構成情報など）をパッケージに追加することを意味します。このため、ZIP ファイル内のファイル数が、サポート・ワークベンチでパッケージ・コンテンツとして表示されたファイルの数より多くなる場合があります。

インシデント・パッケージング・サービスは一連のルールに従って、既存のパッケージ・データに関係付ける ADR 内のトレース・ファイルを決定します。一部のルールは、Enterprise Manager の「インシデント・パッケージング構成」ページで変更できます。

当初のパッケージ・データと関係付けられた追加データには機密情報が含まれている可能性があるため、Oracle サポート・サービスにアップロードする前に、このような機密情報を含むファイルを削除または編集することが重要です。このため、サポート・ワークベンチでは、パッケージをファイナライズするコマンドを独立した操作として実行できます。パッケージを手動でファイナライズした後は、パッケージ・コンテンツを検証し、ファイルを削除または編集してから、ZIP ファイルを生成してアップロードできます。

注意： パッケージのファイナライズは、パッケージがクローズして変更不可になることではありません。ファイナライズしたパッケージには、引続き診断データを追加できます。また、同じパッケージを複数回ファイナライズすることもできます。ファイナライズするたびに、関係付けられた新しいデータが追加されます。

関連項目： 8-44 ページ「インシデント・パッケージのプリファレンスの設定」

クイック・パッケージングとカスタム・パッケージングの概要

Enterprise Manager サポート・ワークベンチには、インシデント・パッケージを作成してアップロードする方法として、クイック・パッケージング方法とカスタム・パッケージング方法があります。

クイック・パッケージング: 最小限のステップがある自動化された方法で、ガイド付きワークフロー（ウィザード）に編成されています。1 つの問題を選択してパッケージ名と説明を入力し、パッケージ・コンテンツのアップロードを即時に実行するか、指定の日時に実行するかをスケジュールします。サポート・ワークベンチでは、問題に関連する診断データを自動的にパッケージに追加してファイナライズし、ZIP ファイルを作成してアップロードします。この方法では、パッケージ・ファイルを追加、編集または削除したり、SQL テスト・ケースなどの他の診断データを追加することはできません。ただし、この方法は、初期障害診断データを Oracle サポート・サービスにアップロードする最も簡単かつ迅速な方法です。

クイック・パッケージングが完了した後、ウィザードで作成されたパッケージはそのまま残ることに注意してください。このパッケージは、カスタム・パッケージング操作を使用して後で変更し、手動で再アップロードできます。

カスタム・パッケージング: 手動による方法で、ステップが多くなります。この方法は、詳細なパッケージング作業を行うサポート・ワークベンチの上級ユーザーを対象にしています。カスタム・パッケージングを使用すると、1 つ以上の問題から新規パッケージを作成したり、1 つ

以上の問題を既存のパッケージに追加できます。新規または更新したパッケージに対して、次のような操作を実行できます。

- 問題やインシデントを追加または削除できます。
- パッケージに対してトレース・ファイルを追加、編集または削除できます。
- 任意のタイプの外部ファイルを追加または削除できます。
- その他の診断データ（SQL テスト・ケースなど）を追加できます。
- パッケージを手動でファイナライズしてからパッケージ・コンテンツを表示して、機密データを編集または削除する必要があるか、ファイルを削除してパッケージ・サイズを縮小する必要があるかを決定できます。

これらの操作には、Oracle サポート・サービスに送信する診断情報が十分であることを確認するまで、数日間かかる場合があります。

カスタム・パッケージングでは、ZIP ファイルの作成と Oracle サポート・サービスへのアップロードを、2 つの独立したステップとして実行できます。各ステップは、即時に実行するか、日時をスケジュールして実行できます。

> **関連項目：** クイック・パッケージング方法の手順については、8-13 ページの「タスク 5: 診断データのパッケージ化と Oracle サポート・サービスへのアップロード」を参照してください。

カスタム・パッケージングを使用した問題のパッケージ化とアップロード

ここでは、1 つ以上の問題を表示してパッケージ化し、Oracle サポート・サービスにアップロードする高度なワークフローについて説明します。このワークフローでは、サポート・ワークベンチのカスタム・パッケージング機能を使用します。この機能によって、アップロードする前にインシデント・パッケージ（パッケージ）に対してファイルを追加、編集および削除できます。

カスタム・パッケージングを使用して問題をパッケージ化およびアップロードする手順は、次のとおりです。

1. 「サポート・ワークベンチ」ホームページにアクセスします。

   手順については、8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」を参照してください。

2. （オプション）パッケージに含める各問題について、問題に関連付けられているサービス・リクエスト番号（SR#）を指定します（ある場合）。指定するには、各問題について次の手順を実行します。

   a. 「サポート・ワークベンチ」ホームページの下部にある「問題」サブページで、問題を選択して**「表示」**をクリックします。

   > **注意：** 問題のリストに対象の問題がみつからない場合、または問題数が多くてスクロールできない場合は、「表示」リストから期間を選択して**「実行」**をクリックします。対象の問題を選択して**「表示」**をクリックします。

   「問題の詳細」ページが表示されます。

   b. SR# ラベルの横にある**「編集」**をクリックし、サービス・リクエスト番号を入力して、**「OK」**をクリックします。

   「問題の詳細」ページにサービス・リクエスト番号が表示されます。

   c. ページの上部にあるロケータ・リンクで**「サポート・ワークベンチ」**をクリックし、「サポート・ワークベンチ」ホームページに戻ります。

3. 「サポート・ワークベンチ」ホームページで、パッケージ化する問題を選択して**「パッケージ」**をクリックします。

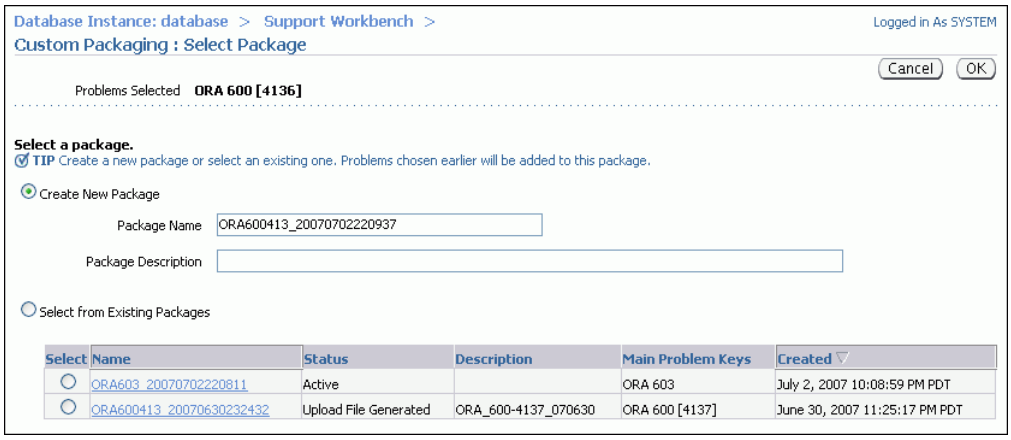
   「パッケージング・モードの選択」ページが表示されます。

   > **注意：** パッケージング・プロセスでは、関係付けられた追加の問題を自動的に選択してパッケージに追加できます。関係付けられた問題の例には、選択した問題の数分後に発生した問題があります。詳細は、8-33 ページの「インシデント・パッケージ内の関係付けられた診断データの概要」を参照してください。

4. **「カスタム・パッケージング」**オプションを選択して**「続行」**をクリックします。

   「カスタム・パッケージング : パッケージの選択」ページが表示されます。

図 8-5「カスタム・パッケージング : パッケージの選択」ページ

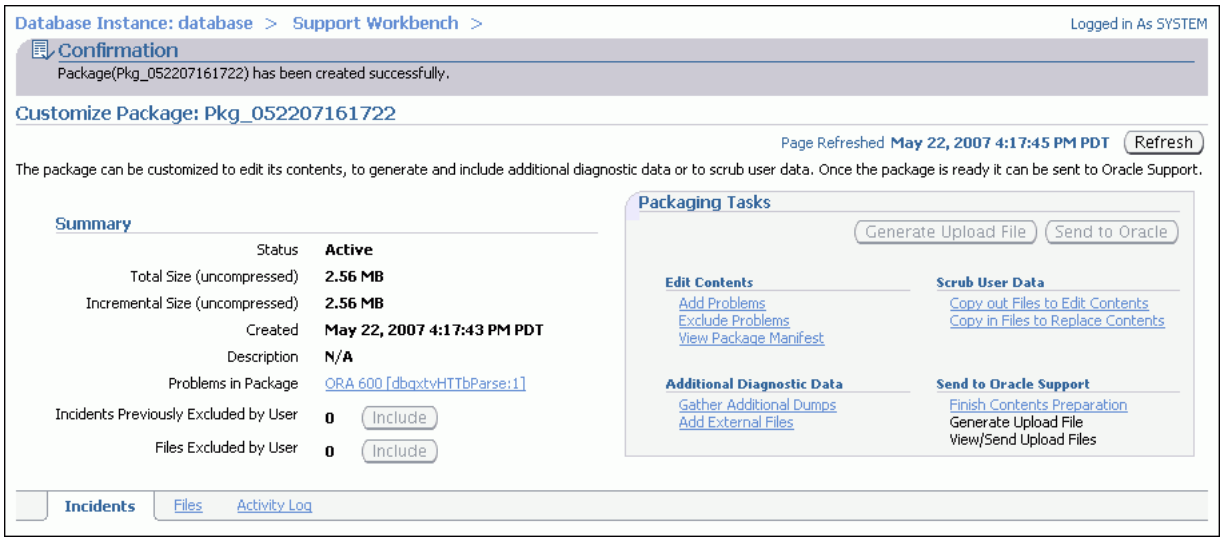


5. 次のいずれかを実行します。

   - 新規パッケージを作成するには、**「新規パッケージの作成」**オプションを選択し、パッケージ名と説明を入力して**「OK」**をクリックします。
   - 選択した問題を既存のパッケージに追加するには、**「既存パッケージからの選択」**オプションを選択し、更新するパッケージを選択して**「OK」**をクリックします。

   「パッケージのカスタマイズ」ページが表示されます。このページには、パッケージに含まれている問題とインシデントに加えて、選択可能なパッケージング・タスクが表示されます。これらのタスクは、新規パッケージまたは更新された既存のパッケージに対して実行します。

図 8-6「パッケージのカスタマイズ」ページ



6. （オプション）「パッケージング・タスク」セクションで、リンクをクリックして 1 つ以上のパッケージング・タスクを実行します。または、「パッケージのカスタマイズ」ページとそのサブページにある他のコントロールを使用して、パッケージを操作します。完了後に「パッケージのカスタマイズ」ページに戻ります。

   いくつかの一般的なパッケージング・タスクの手順については、8-38 ページの「インシデント・パッケージの表示と変更」を参照してください。

7. 「パッケージのカスタマイズ」ページの「パッケージング・タスク」セクションで、「Oracle サポートに送信」ヘッダー下にある**「コンテンツ準備の終了」**をクリックしてパッケージをファイナライズします。

   パッケージに含まれているファイルのリスト（またはリストの一部）が表示されます（表示されるまでしばらく時間がかかる場合があります）。このリストには、関係付けられた診断情報を挿入することが決定され、ファイナライズ・プロセスでその情報が追加されたファイルが表示されます。

   パッケージのファイナライズの定義については、8-33 ページの「インシデント・パッケージ内の関係付けられた診断データの概要」を参照してください。

8. **「ファイル」**リンクをクリックして、パッケージ内のすべてのファイルを表示します。リストを検証して、公開できない機密データのファイルがあるかどうかを確認します。該当するファイルがみつかった場合は、そのファイルを除外（削除）または編集します。

   ファイルを編集および削除する手順については、8-39 ページの「インシデント・パッケージ・ファイルの編集（コピー・アウトとコピー・イン）」および 8-43 ページの「インシデント・パッケージ・ファイルの削除」を参照してください。

   ファイルの内容を表示するには、ファイルの表の右側の列にある眼鏡アイコンをクリックします。ホスト資格証明を入力します（要求された場合）。

   > **注意：** 通常、トレース・ファイルは Oracle 内部でのみ使用します。

9. **「アップロード・ファイルの生成」**をクリックします。

   「アップロード・ファイルの生成」ページが表示されます。

10. **「全体」**または**「増分」**オプションを選択して、全パッケージの ZIP ファイルまたは増加分パッケージの ZIP ファイルを生成します。

    全パッケージの ZIP ファイルの場合は、常にパッケージのすべてのコンテンツ（元のコンテンツおよび関係付けられたすべてのデータ）が ZIP ファイルに追加されます。

    増加分パッケージの ZIP ファイルの場合は、同じパッケージの ZIP ファイルが最後に作成された時点以降に新規追加または変更された診断情報のみが ZIP ファイルに追加されます。たとえば、パッケージに対して生成された物理ファイルにトレース・ファイルが最後に追加された時点以降にトレース情報がそのトレース・ファイルに追加された場合は、そのトレース・ファイルが増加分パッケージの ZIP ファイルに追加されます。逆に、パッケージに対して最後にアップロードした時点以降にトレース・ファイルを変更していない場合、そのトレース・ファイルは増加分パッケージの ZIP ファイルに追加されません。

    注意： パッケージに対してアップロード・ファイルが作成されていない場合、「増分」オプションは使用不可になります。

11. ファイル作成を即時実行するか、後の日時に実行するかをスケジュールして（**「即時」**または**「後で」**を選択）、**「発行」**をクリックします。

    ファイル作成ではシステム・リソースを大量に使用する場合があるため、ファイルの作成は、システム使用量が少ない時期にスケジュールすることをお薦めします。

    「処理中」ページが表示され、ZIP ファイルが作成されます。処理が完了すると、確認ページが表示されます。

    注意： ZIP ファイルが作成されると、パッケージは自動的にファイナライズされます。

12. **「OK」**をクリックします。

    「パッケージのカスタマイズ」ページに戻ります。

13. **「Oracle に送信」**をクリックします。

    「アップロード・ファイルの表示 / 送信」ページが表示されます。

14. アップロードする ZIP ファイルを選択して**「Oracle に送信」**をクリックします。

    「Oracle に送信」ページが表示されます。選択した ZIP ファイルが表にリストされます。

15. 要求された Oracle MetaLink 情報を入力します。「新規サービス・リクエスト (SR) の作成」の横にある**「はい」**または**「いいえ」**を選択します。「はい」を選択すると、ドラフトのサービス・リクエストが作成されます。この場合は、後で Oracle MetaLink にログインして、サービス・リクエストの詳細を入力する必要があります。「いいえ」を選択した場合は、既存のサービス・リクエスト番号を入力します。

16. アップロードを即時または後の日時にスケジュールして、**「発行」**をクリックします。

    「処理中」ページが表示されます。アップロードが正常に完了した場合は、確認ページが表示されます。アップロードが完了できなかった場合は、エラー・ページが表示されます。エラー・ページには、ZIP ファイルを Oracle に手動でアップロードするように要求するメッセージが表示される場合があります。その場合の手順については、Oracle サポート・サービス担当者に問い合せてください。

17. **「OK」**をクリックします。

    「アップロード・ファイルの表示 / 送信」ページに戻ります。「送信時刻」列で、アップロードしたファイルのステータスをチェックします。

注意： サポート・ワークベンチでは、Oracle Configuration Manager を使用して物理ファイルをアップロードします。Oracle Configuration Manager がインストールされていなかったり、適切に構成されていないと、アップロードに失敗する場合があります。失敗した場合は、ファイルを Oracle サポート・サービスに手動でアップロードするように要求するメッセージが、パッケージの ZIP ファイルへのパスとともに表示されます。アップロードは、Oracle MetaLink を使用して手動で実行できます。

Oracle Configuration Manager の詳細は、『Oracle Configuration Manager インストレーションおよび管理ガイド』を参照してください。

関連項目：

- 8-3 ページ「インシデントおよび問題の概要」
- 8-32 ページ「インシデント・パッケージの概要」
- 8-33 ページ「クイック・パッケージングとカスタム・パッケージングの概要」

インシデント・パッケージの表示と変更

カスタム・パッケージング方法を使用してインシデント・パッケージを作成した後は、Oracle サポート・サービスへのアップロード前に、そのパッケージのコンテンツを表示または変更できます。さらに、クイック・パッケージング方法を使用して診断データをパッケージ化してアップロードした後は、サポート・ワークベンチで作成したパッケージのコンテンツを表示または変更して、パッケージを再アップロードできます。パッケージを変更するには、パッケージング・タスクの選択肢からタスクを選択します。ほとんどのタスクは「パッケージのカスタマイズ」ページから選択できます。

ここでは、いくつかの一般的なパッケージング・タスクの手順について説明します。この項の内容は、次のとおりです。

- インシデント・パッケージ・ファイルの編集（コピー・アウトとコピー・イン）
- インシデント・パッケージへの外部ファイルの追加
- インシデント・パッケージ・ファイルの削除
- インシデント・パッケージのアクティビティ・ログの表示と更新

また、次の各項では、パッケージの詳細の表示方法、特定のパッケージの「パッケージのカスタマイズ」ページにアクセスする方法について説明します。

- パッケージの詳細の表示
- 「パッケージのカスタマイズ」ページへのアクセス

関連項目：

- 8-32 ページ「インシデント・パッケージの概要」
- 8-34 ページ「カスタム・パッケージングを使用した問題のパッケージ化とアップロード」

パッケージの詳細の表示

「パッケージの詳細」ページには、パッケージ内のインシデント、トレース・ファイルおよびその他のファイルに関する情報が表示され、アクティビティ・ログを表示してパッケージに追加できます。

パッケージの詳細を表示する手順は、次のとおりです。

1. 「サポート・ワークベンチ」ホームページにアクセスします。

   手順については、8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」を参照してください。

2. **「パッケージ」**リンクをクリックして「パッケージ」サブページを表示します。

   自動診断リポジトリ（ADR）に現時点で格納されているパッケージのリストが表示されます。

3. （オプション）表示するパッケージの数を減らすには、リストの上部にある**「検索」**フィールドにテキストを入力して**「実行」**をクリックします。

   パッケージ名に検索テキストが含まれているパッケージがすべて表示されます。全パッケージのリストを表示するには、**「パッケージ」**リンクを再度クリックします。

4. 「パッケージ名」列で、対象のパッケージのリンクをクリックします。

   「パッケージの詳細」ページが表示されます。

「パッケージのカスタマイズ」ページへのアクセス

「パッケージのカスタマイズ」ページでは、各種のパッケージング・タスク（問題の追加および削除、パッケージ・ファイルの追加、削除および修正（編集）、パッケージの ZIP ファイルの生成およびアップロードなど）を実行します。

「パッケージのカスタマイズ」ページにアクセスする手順は、次のとおりです。

1. 8-39 ページの「パッケージの詳細の表示」で説明するように、特定のパッケージの「パッケージの詳細」ページにアクセスします。

2. **「パッケージのカスタマイズ」**をクリックします。

   「パッケージのカスタマイズ」ページが表示されます。

インシデント・パッケージ・ファイルの編集（コピー・アウトとコピー・イン）

サポート・ワークベンチを使用すると、インシデント・パッケージ内の 1 つ以上のファイルを編集できます。この作業は、ファイル内の機密データを削除または上書きする場合に必要です。パッケージ・ファイルを編集するには、最初にパッケージから宛先ディレクトリにファイルをコピー・アウトし、テキスト・エディタまたは他のユーティリティを使用して編集してから、そのファイルをパッケージにコピーし直して、元のパッケージ・ファイルを上書きします。

次の手順では、パッケージがすでに作成されて診断データが格納されていることを前提としています。

インシデント・パッケージ・ファイルを編集する手順は、次のとおりです。

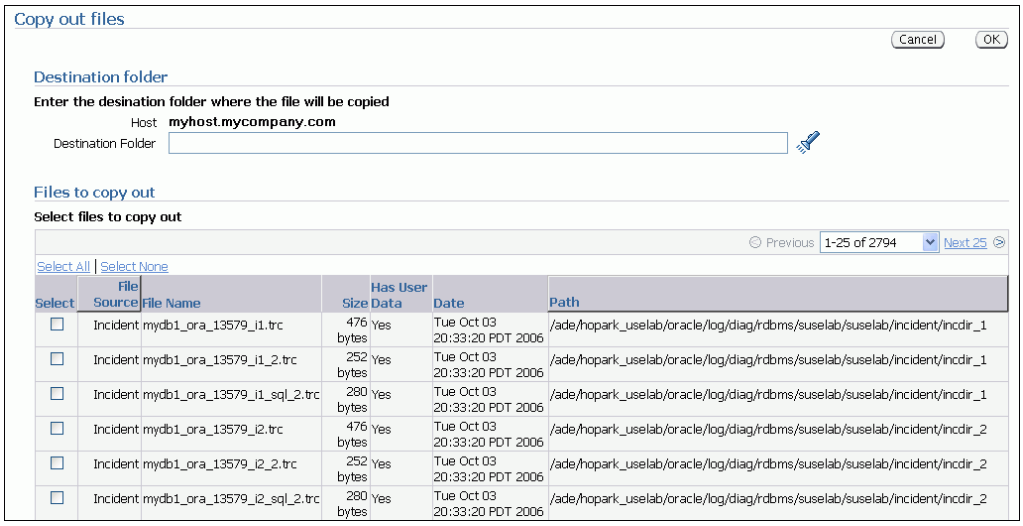
1. 対象のインシデント・パッケージの「パッケージのカスタマイズ」ページにアクセスします。

   手順については、8-39 ページの「「パッケージのカスタマイズ」ページへのアクセス」を参照してください。

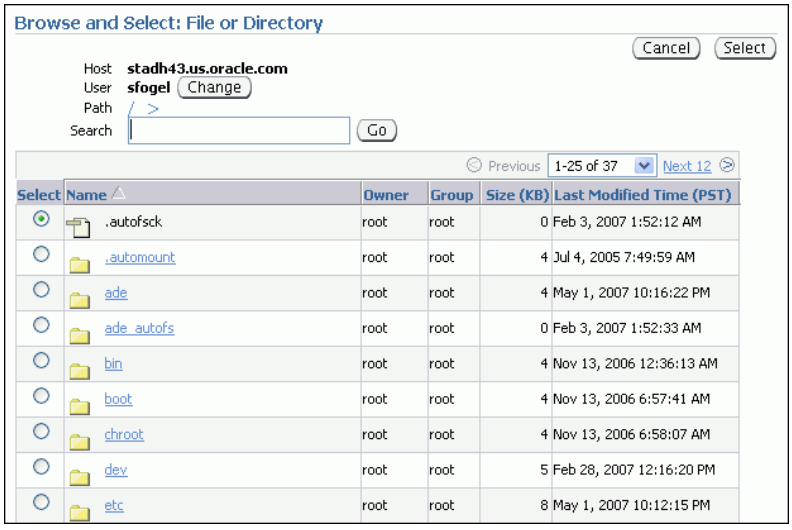
2. 「パッケージング・タスク」セクションで、「ユーザー・データの修正」ヘッダー下にある**「コンテンツ編集のためのファイルのコピー・アウト」**をクリックします。

   「ファイルのコピー・アウト」ページが表示されます。このページには、ファイルをコピーできるホストの名前が表示されます。

図 8-7「ファイルのコピー・アウト」ページ



3. 次のいずれかを実行して、ファイルの宛先ディレクトリを指定します。
   - **「宛先フォルダ」**フィールドにディレクトリ・パスを入力します。
   - **「宛先フォルダ」**フィールドの横にある懐中電灯アイコンをクリックして、次の手順を実行します。
     - ホスト資格証明を要求された場合は、ファイルをコピー・アウトするホストの資格証明を入力して**「OK」**をクリックします（次回に資格証明を要求されないようにするには、**「優先資格証明として保存」**を選択します）。
     - 「参照および選択：ファイルまたはディレクトリ」ウィンドウで、ディレクトリ・リンクをクリックしてディレクトリ階層の下位に移動するか、**「パス」**ラベルの横にあるディレクトリ名をクリックしてディレクトリ階層の上位に移動して、対象の宛先ディレクトリを検索します。



リストに表示するディレクトリの数を減らすには、**「検索」**フィールドに検索テキストを入力して**「実行」**をクリックします。ディレクトリ名に検索テキストが含まれているディレクトリがすべて表示されます。

– 対象の宛先ディレクトリを選択して**「選択」**をクリックします。

「参照および選択 : ファイルまたはディレクトリ」ウィンドウがクローズし、選択したディレクトリへのパスが「ファイルのコピー・アウト」ページの「宛先フォルダ」フィールドに表示されます。

4. 「コピー・アウトするファイル」で、対象のファイルを選択して**「OK」**をクリックします。

注意 : 対象のファイルがみつからない場合は、別のページにある場合があります。**「次」**リンクをクリックして、次のページを表示します。対象のファイルがみつかるまで、**「次」**をクリックし続けるか、ファイル番号のリスト（「次」リンクの左側）から番号を選択します。ファイルを選択して**「OK」**をクリックします。

「パッケージのカスタマイズ」ページに戻ると、コピー・アウトされたファイルをリストした確認メッセージが表示されます。

5. テキスト・エディタまたは他のユーティリティを使用して、ファイルを編集します。
6. 「パッケージのカスタマイズ」ページの「パッケージング・タスク」セクションで、「ユーザー・データの修正」ヘッダー下にある**「コンテンツ置換えのためのファイルのコピー・イン」**をクリックします。

「ファイルのコピー・イン」ページが表示されます。このページには、コピー・アウトしたファイルが表示されます。

7. コピー・インするファイルを選択して**「OK」**をクリックします。

ファイルがパッケージにコピー・インされて、既存のファイルが上書きされます。「パッケージのカスタマイズ」ページに戻ると、コピー・インされたファイルをリストした確認メッセージが表示されます。

インシデント・パッケージへの外部ファイルの追加

インシデント・パッケージには、任意のタイプの外部ファイルを追加できます。

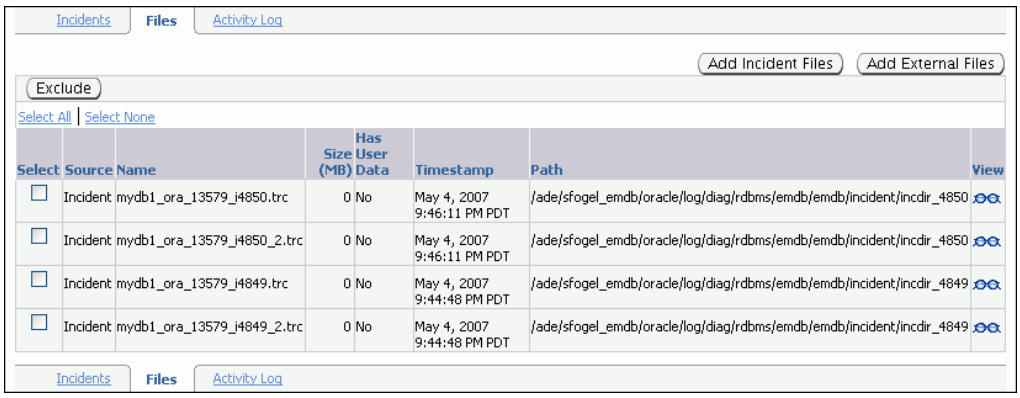
外部ファイルをインシデント・パッケージに追加する手順は、次のとおりです。

1. 対象のインシデント・パッケージの「パッケージのカスタマイズ」ページにアクセスします。

手順については、8-39 ページの「「パッケージのカスタマイズ」ページへのアクセス」を参照してください。

2. **「ファイル」**リンクをクリックして「ファイル」サブページを表示します。

図 8-8「パッケージのカスタマイズ」ページの「ファイル」サブページ



このページでは、パッケージに対してファイルを追加または削除できます。

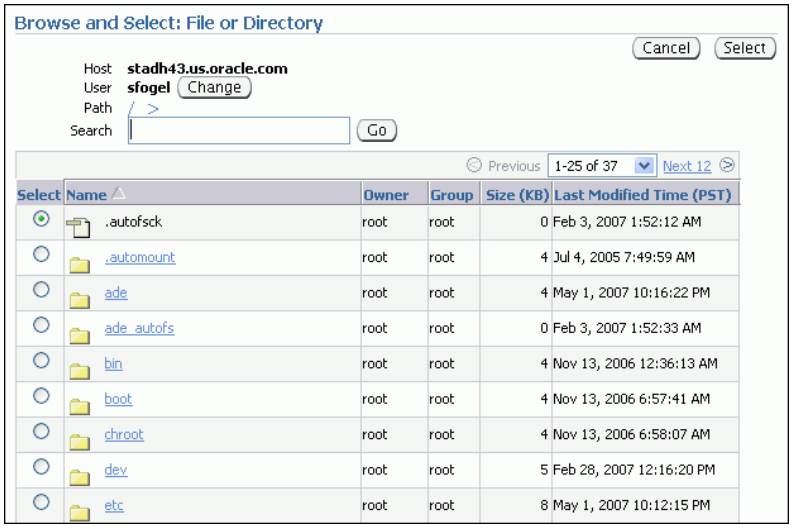
注意：「表示」リストに表示されるのは、物理ファイルをすでに作成したパッケージのみです。このリストには、増加分パッケージ・コンテンツまたは全パッケージ・コンテンツのいずれかを表示できます。デフォルトでは増加分パッケージ・コンテンツが選択されています。デフォルトの選択で表示されるのは、パッケージに対して物理ファイルが最後に生成された時点以降に作成または変更されたパッケージ・ファイルのみです。

3. **「外部ファイルの追加」**をクリックします。

   「外部ファイルの追加」ページが表示されます。このページには、ファイルを選択可能なホストの名前が表示されます。

4. 次のいずれかを実行して、追加するファイルを指定します。

   - **「ファイル名」**フィールドに、ファイルへのフルパスを入力します。
   - **「ファイル名」**フィールドの横にある懐中電灯アイコンをクリックして、次の手順を実行します。
     - ホスト資格証明を要求された場合は、外部ファイルが常駐するホストの資格証明を入力して**「OK」**をクリックします（次回に資格証明を要求されないようにするには、**「優先資格証明として保存」**を選択します）。
     - 「参照および選択：ファイルまたはディレクトリ」ウィンドウで、ディレクトリ・リンクをクリックしてディレクトリ階層の下位に移動するか、**「パス」**ラベルの横にあるディレクトリ名をクリックしてディレクトリ階層の上位に移動して、対象のファイルを検索します。



     リストに表示するファイルまたはディレクトリの数を減らすには、**「検索」**フィールドに検索テキストを入力して**「実行」**をクリックします。ファイル名またはディレクトリ名に検索テキストが含まれているファイルまたはディレクトリがすべて表示されます。

     - 「選択」列で、対象のファイルをクリックして選択し、**「選択」**をクリックします。

       「参照および選択」ウィンドウがクローズし、選択したファイルへのパスが「外部ファイルの追加」ページの「ファイル名」フィールドに表示されます。

5. **「OK」**をクリックします。

   「パッケージのカスタマイズ」ページに戻ると、「ファイル」サブページが表示されます。これで、選択したファイルがファイル・リストに表示されます。

インシデント・パッケージ・ファイルの削除

インシデント・パッケージから任意のタイプのファイルを 1 つ以上削除できます。

インシデント・パッケージ・ファイルを削除する手順は、次のとおりです。

1. 対象のインシデント・パッケージの「パッケージのカスタマイズ」ページにアクセスします。

   手順については、8-39 ページの「「パッケージのカスタマイズ」ページへのアクセス」を参照してください。

2. **「ファイル」**リンクをクリックして「ファイル」サブページを表示します。

   パッケージに含まれているファイルのリストが表示されます

   このパッケージ対して物理ファイルをまだ生成していない場合は、すべてのパッケージ・ファイルがリストに表示されます。物理ファイルをすでに生成している場合は、ファイル・リストの上部に「表示」リストが表示されます。このリストでは、増加分パッケージ・コンテンツのみを表示するか、全パッケージ・コンテンツを表示するかを選択できます。デフォルトでは増加分パッケージ・コンテンツが選択されています。デフォルトの選択で表示されるのは、パッケージに対して物理ファイルが最後に生成された時点以降に作成または変更されたパッケージ・ファイルのみです。すべてのパッケージ・ファイルを表示するには、「表示」リストから**「全パッケージ・コンテンツ」**を選択します。

3. 削除するファイルを選択して**「除外」**をクリックします。

注意： 対象のファイルがみつからない場合は、別のページにある場合があります。**「次」**リンクをクリックして、次のページを表示します。対象のファイルがみつかるまで、**「次」**をクリックし続けるか、ファイル番号のリスト（「次」リンクの左側）から番号を選択します。ファイルを選択して**「削除」**をクリックします。

インシデント・パッケージのアクティビティ・ログの表示と更新

サポート・ワークベンチでは、インシデント・パッケージごとにアクティビティ・ログが保持されます。パッケージに対して実行したほとんどのアクティビティ（ファイルの追加や削除、パッケージの ZIP ファイルの作成など）はログに記録されます。独自のノートをログに追加することもできます。この機能は、特に、複数のデータベース管理者がパッケージで作業をする場合に便利です。

インシデント・パッケージのアクティビティ・ログを表示および更新する手順は、次のとおりです。

1. 対象のインシデント・パッケージの「パッケージの詳細」ページにアクセスします。

   手順については、8-39 ページの「「パッケージのカスタマイズ」ページへのアクセス」を参照してください。

2. **「アクティビティ・ログ」**リンクをクリックして、「アクティビティ・ログ」サブページを表示します。

   アクティビティ・ログが表示されます。

3. 独自のノートをアクティビティ・ログに追加するには、**「ノート」**フィールドにテキストを入力して**「ノートの追加」**をクリックします。

   ノートの日時が記録され、リストに追加されます。

インシデント・パッケージのプリファレンスの設定

ここでは、インシデント・パッケージのプリファレンスの設定手順について説明します。インシデント・パッケージのプリファレンスの例には、インシデント情報を保持する日数、および各問題についてパッケージに含める最初と最後のインシデントの数があります（デフォルトでは、問題に多数のインシデントがある場合、最初の 3 つと最後の 3 つのインシデントのみがパッケージに追加されます）。この設定およびインシデント・パッケージの他のプリファレンスは、Enterprise Manager または ADRCI ユーティリティを使用して変更できます。

Enterprise Manager を使用してインシデント・パッケージのプリファレンスを設定する手順は、次のとおりです。

1. 「サポート・ワークベンチ」ホームページにアクセスします。

   手順については、8-16 ページの「Enterprise Manager サポート・ワークベンチを使用した問題の表示」を参照してください。

2. ページの下部にある「関連リンク」セクションで、**「インシデント・パッケージング構成」**をクリックします。

   「インシデント・パッケージング構成の表示」ページが表示されます。**「ヘルプ」**をクリックして、このページの設定の説明を表示します。

3. **「編集」**をクリックします。

   「インシデント・パッケージング構成の編集」ページが表示されます。

4. 設定を編集し、**「OK」**をクリックして変更を適用します。

関連項目：

- 8-32 ページ「インシデント・パッケージの概要」
- 8-3 ページ「インシデントおよび問題の概要」
- 8-13 ページ「タスク 5: 診断データのパッケージ化と Oracle サポート・サービスへのアップロード」
- ADRCI については、『Oracle Database ユーティリティ』を参照してください。

第 II 部

Oracle Database の構造と記憶域

第 II 部では、データベース構造を記憶域コンポーネントの観点から説明し、各コンポーネントの作成方法と管理方法について説明します。この部の構成は、次のとおりです。

9

制御ファイルの管理

この章の内容は次のとおりです。

> **関連項目：** Oracle Database サーバーによって作成および管理される制御ファイルの作成方法は、第 15 章「Oracle Managed Files の使用」を参照してください。

制御ファイルの概要

制御ファイルはデータベースの物理構造を記録した小さなバイナリ・ファイルであり、すべての Oracle Database に含まれています。制御ファイルには、次の情報が格納されています。

- データベース名
- 対応するデータファイルと REDO ログ・ファイルの名前と位置
- データベース作成のタイムスタンプ
- 現行のログ順序番号
- チェックポイント情報

制御ファイルは、データベースがオープンしているときに必ず Oracle Database サーバーが書き込めるように、使用可能にしておく必要があります。制御ファイルがないと、データベースがマウントできず、リカバリが困難になります。

Oracle Database の制御ファイルはデータベースとともに作成されます。デフォルトでは、データベースの作成時に、制御ファイルのコピーが少なくとも 1 つ作成されます。デフォルトで複数のコピーが作成されるオペレーティング・システムもあります。データベース作成時に、制御ファイルのコピーを 2 つ以上作成することをお薦めします。その後も、制御ファイルを失ったり、制御ファイル内の設定を変更する場合には、制御ファイルを作成できます。

制御ファイルのガイドライン

ここでは、データベースの制御ファイルを管理するためのガイドラインについて説明します。この項の内容は、次のとおりです。

- 制御ファイルのファイル名の指定
- 異なるディスク上での制御ファイルの多重化
- 制御ファイルのバックアップ
- 制御ファイルのサイズ管理

制御ファイルのファイル名の指定

データベース初期化パラメータ・ファイルの CONTROL_FILES 初期化パラメータを使用して、制御ファイルの名前を指定します（9-4 ページの「初期制御ファイルの作成」を参照してください）。インスタンスは、起動時に、指定されたすべてのファイルを認識してオープンします。さらに、データベースの稼働中、指定されたすべての制御ファイルに情報を書き込み、メンテナンスします。

データベースの作成前に CONTROL_FILES に対してファイルを指定しない場合は、次のように処理されます。

- Oracle Managed Files を使用していない場合、データベースではデフォルトのファイル名で制御ファイルが作成されます。デフォルトのファイル名はオペレーティング・システムによって異なります。
- Oracle Managed Files を使用している場合は、この機能を使用可能にするために設定した初期化パラメータによって制御ファイルの名前と位置が決定します（第 15 章「Oracle Managed Files の使用」を参照）。
- 自動ストレージ管理を使用している場合は、不完全 ASM ファイル名を DB_CREATE_FILE_DEST および DB_RECOVERY_FILE_DEST 初期化パラメータに設定できます。ASM によって、制御ファイルが適切な場所に自動的に作成されます。詳細は、『Oracle Database ストレージ管理者ガイド』の ASM ファイル名の概要および ASM を使用するデータベースの作成に関する項を参照してください。

異なるディスク上での制御ファイルの多重化

Oracle Database には少なくとも 2 つの制御ファイルを作成し、各ファイルを異なる物理ディスクに配置するようにします。ディスク障害によって制御ファイルが破損した場合は、対応するインスタンスを必ず停止します。ディスク・ドライブを修復後、他のディスク上にある制御ファイルの正常なコピーを使用して破損した制御ファイルをリストアすると、インスタンスを再起動できます。この場合は、メディア・リカバリは不要です。

多重制御ファイルは、次のように動作します。

- データベースは、データベース初期化パラメータ・ファイルの初期化パラメータ CONTROL_FILES にリストされているすべてのファイル名に対して、情報を書き込みます。
- データベースの稼働中に、CONTROL_FILES パラメータにリストされている最初のファイルのみデータベースによって読み込まれます。
- データベースの稼働中に制御ファイルのいずれかが使用できなくなった場合、インスタンスは動作不能になり、異常終了します。

注意： データベースには最低 2 つ以上の制御ファイルを作成し、それらを異なる物理ディスク上に配置することをお薦めします。

制御ファイルを多重化する方法の 1 つは、REDO ログが多重化されている場合、REDO ログ・グループのメンバーが格納されているすべてのディスク・ドライブに制御ファイルのコピーを格納することです。このようにファイルを配置することによって、単一のディスク障害のために制御ファイルと REDO ログ・グループがすべて失われる危険が少なくなります。

制御ファイルのバックアップ

制御ファイルのバックアップは非常に重要です。初期設定時およびデータベースの物理構造を変更したときは、必ずバックアップを作成してください。たとえば、次のような構造上の変更を行った場合は、バックアップを作成する必要があります。

- データファイルの追加、削除または名前変更
- 表領域の追加または削除、表領域の読取り / 書込み状態の変更
- REDO ログ・ファイルまたは REDO ログ・グループの追加と削除

制御ファイルのバックアップ方法については、9-8 ページの「制御ファイルのバックアップ」を参照してください。

制御ファイルのサイズ管理

制御ファイルのサイズを決定する主な要因は、対応するデータベースを作成した CREATE DATABASE 文の MAXDATAFILES、MAXLOGFILES、MAXLOGMEMBERS、MAXLOGHISTORY および MAXINSTANCES パラメータに設定された値です。これらのパラメータの値を大きくすると、対応するデータベースの制御ファイルのサイズも大きくなります。

関連項目：

- 制御ファイルの最大サイズの詳細は、使用しているオペレーティング・システム固有の Oracle マニュアルを参照してください。
- CREATE DATABASE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

制御ファイルの作成

ここでは、制御ファイルを作成する方法について説明します。この項の内容は、次のとおりです。

- 初期制御ファイルの作成
- 制御ファイルの追加コピーの作成、名前変更および再配置
- 新しい制御ファイルの作成

初期制御ファイルの作成

Oracle Database の初期制御ファイルは、CREATE DATABASE 文を発行したときに作成されます。制御ファイルの名前は、データベースの作成時に使用される初期化パラメータ・ファイルの CONTROL_FILES パラメータで指定します。CONTROL_FILES で指定するファイル名はパスを含めて完全に指定する必要があり、オペレーティング・システムによって異なります。CONTROL_FILES 初期化パラメータの例を次に示します。

```
CONTROL_FILES = (/u01/oracle/prod/control01.ctl,
                 /u02/oracle/prod/control02.ctl,
                 /u03/oracle/prod/control03.ctl)
```

指定した名前のファイルがデータベース作成時にすでに存在する場合は、CREATE DATABASE 文で CONTROLFILE REUSE 句を指定してください。そうしないとエラーが発生します。なお、古い制御ファイルのサイズと新しい制御ファイルの SIZE パラメータが異なる場合、REUSE 句は使用できません。

制御ファイルのサイズは、Oracle Database の一部のリリース間で異なることがあります。また、制御ファイル内に指定されたファイルの数が変わると、制御ファイルのサイズも変わります。制御ファイルのサイズには、MAXLOGFILES、MAXLOGMEMBERS、MAXLOGHISTORY、MAXDATAFILES、MAXINSTANCES などの構成パラメータが影響します。

CONTROL_FILES 初期化パラメータの値を後で変更して、制御ファイルを追加したり、既存の制御ファイルの名前や位置を変更できます。

> **関連項目：** 制御ファイルの指定方法の詳細は、使用しているオペレーティング・システム固有の Oracle マニュアルを参照してください。

制御ファイルの追加コピーの作成、名前変更および再配置

多重化のために制御ファイルのコピーを追加作成するには、既存の制御ファイルを新しい位置にコピーし、そのファイル名を制御ファイルのリストに追加します。同様に、制御ファイルの名前を変更するには、既存の制御ファイルを新しい名前や位置にコピーし、制御ファイル・リストのファイル名を変更します。どちらの場合も、作業中に制御ファイルが変更されないように、制御ファイルをコピーする前にデータベースを停止してください。

現行の制御ファイルの多重化コピーを追加、または制御ファイル名を変更する手順は、次のとおりです。

1. データベースを停止します。
2. オペレーティング・システムのコマンドを使用して、既存の制御ファイルを新しい位置にコピーします。
3. データベース初期化パラメータ・ファイルの CONTROL_FILES パラメータを編集して、新しい制御ファイル名を追加するか、または既存の制御ファイル名を変更します。
4. データベースを再起動します。

新しい制御ファイルの作成

ここでは、新しい制御ファイルを作成する場合とその方法について説明します。

新しい制御ファイルを作成する場合

次の状況の場合に、新しい制御ファイルを作成する必要があります。

- データベースの制御ファイルがすべて破損し、制御ファイルのバックアップがない場合。
- データベース名を変更する場合。

  たとえば、分散環境でデータベース名が別のデータベース名と競合する場合は、データベース名を変更します。

  > **注意：** データベース名と DBID（内部データベース識別子）は、DBNEWID ユーティリティを使用して変更できます。このユーティリティの詳細は、『Oracle Database ユーティリティ』を参照してください。

- 互換性レベルが 10.2.0 より前の値に設定されているときに、CREATE DATABASE コマンドまたは CREATE CONTROLFILE コマンドの MAXLOGFILES、MAXLOGMEMBERS、MAXLOGHISTORY および MAXINSTANCES のいずれかのパラメータに関連するデータベース構成の領域を変更する必要がある場合。互換性が 10.2.0 以上の場合は、このような変更を加えるときに新しい制御ファイルを作成する必要はありません。制御ファイルは、新しい構成情報を格納できるように、必要に応じて自動的に拡張されます。

  たとえば、データベースを作成または制御ファイルを再作成したときに、MAXLOGFILES を 3 に設定したとします。次に、4 つ目の REDO ログ・ファイル・グループを、ALTER DATABASE コマンドを使用してデータベースに追加するとします。互換性が 10.2.0 以上に設定されている場合、この作業は可能です。制御ファイルは、新しいログ・ファイル情報を格納できるように、自動的に拡張されます。ただし、互換性が 10.2.0 より前に設定されている場合は、ALTER DATABASE コマンドを使用するとエラーが発生します。この場合は、最初に新しい制御ファイルを作成する必要があります。

  互換性レベルの詳細は、2-30 ページの「COMPATIBLE 初期化パラメータの概要」を参照してください。

CREATE CONTROLFILE 文

CREATE CONTROLFILE 文を使用して、データベースの新しい制御ファイルを作成できます。次の文は、prod データベース（これまで別のデータベース名を使用していたデータベース）に対する新しい制御ファイルを作成します。

```
CREATE CONTROLFILE
   SET DATABASE prod
   LOGFILE GROUP 1 ('/u01/oracle/prod/redo01_01.log',
                    '/u01/oracle/prod/redo01_02.log'),
           GROUP 2 ('/u01/oracle/prod/redo02_01.log',
                    '/u01/oracle/prod/redo02_02.log'),
           GROUP 3 ('/u01/oracle/prod/redo03_01.log',
                    '/u01/oracle/prod/redo03_02.log')
   RESETLOGS
   DATAFILE '/u01/oracle/prod/system01.dbf' SIZE 3M,
            '/u01/oracle/prod/rbs01.dbs' SIZE 5M,
            '/u01/oracle/prod/users01.dbs' SIZE 5M,
            '/u01/oracle/prod/temp01.dbs' SIZE 5M
   MAXLOGFILES 50
   MAXLOGMEMBERS 3
   MAXLOGHISTORY 400
   MAXDATAFILES 200
   MAXINSTANCES 6
   ARCHIVELOG;
```

注意：

- CREATE CONTROLFILE 文は、指定したデータファイルと REDO ログ・ファイルを破損する可能性があります。ファイル名を省略すると、そのファイルのデータが失われたり、データベース全体にアクセスできなくなる場合があります。この文を発行するときには十分に注意し、必ず「新しい制御ファイルの作成手順」の手順に従ってください。
- 新しい制御ファイルを作成する前にデータベースの FORCE LOGGING を使用可能にしていて、この設定を引き続き有効にする場合は、CREATE CONTROLFILE 文で FORCE LOGGING 句を指定する必要があります。2-23 ページの「FORCE LOGGING モードの指定」を参照してください。

関連項目： CREATE CONTROLFILE 文の詳細な構文は、『Oracle Database SQL リファレンス』を参照してください。

新しい制御ファイルの作成手順

新しい制御ファイルを作成するには、次の手順を実行します。

1. データベースのデータファイルと REDO ログ・ファイルすべてのリストを作成します。

   9-8 ページの「制御ファイルのバックアップ」に記載されている制御ファイルのバックアップの推奨事項に従っている場合は、現在のデータベース構造を反映するデータファイルと REDO ログ・ファイルのリストがすでに作成されています。しかし、そのようなリストがない場合は、次の文を実行することでリストが生成されます。

   ```
   SELECT MEMBER FROM V$LOGFILE;
   SELECT NAME FROM V$DATAFILE;
   SELECT VALUE FROM V$PARAMETER WHERE NAME = 'control_files';
   ```

   このようなリストが手元になく、制御ファイルが破損してデータベースをオープンできない場合は、データベースを構成するデータファイルと REDO ログ・ファイルをすべて特定してください。新しい制御ファイルを作成すると、手順 5 で指定したファイル以外はリカバリできなくなります。さらに、SYSTEM 表領域を構成するファイルを 1 つでも省略すると、データベースをリカバリできないことがあります。

2. データベースを停止します。

   データベースがオープンしている場合は、できるかぎり通常モードでデータベースを停止します。IMMEDIATE 句または ABORT 句は、他に方法がない場合にのみ使用してください。

3. データベースのデータファイルと REDO ログ・ファイルすべてのバックアップを作成します。

4. 新しいインスタンスを起動します。ただし、データベースのマウントとオープンは行いません。

   ```
   STARTUP NOMOUNT
   ```

5. CREATE CONTROLFILE 文を使用して、データベースの新しい制御ファイルを作成します。

   制御ファイルに加えて、REDO ログ・グループも失ってしまった場合は、新しい制御ファイルの作成時に RESETLOGS 句を指定します。この場合は、失った REDO ログをリカバリする必要があります（手順 8）。データベースの名前を変更した場合は、必ず RESETLOGS 句を指定してください。それ以外の場合は、NORESETLOGS 句を選択してください。

6. 新しい制御ファイルのバックアップをオフラインの記憶デバイスに格納します。バックアップ作成の手順については、9-8 ページの「制御ファイルのバックアップ」を参照してください。

7. データベースの CONTROL_FILES 初期化パラメータを編集し、手順 5 で作成したすべての制御ファイルがデータベースの新しい一部となるように指定します。ただし、バックアップ制御ファイルは含めません。データベースの名前を変更する場合は、インスタンス・パラメータ・ファイルの DB_NAME パラメータを編集して新しい名前を指定します。
8. 必要に応じて、データベースをリカバリします。データベースをリカバリしない場合は、手順 9 にスキップしてください。

   リカバリの一部として制御ファイルを作成している場合は、データベースをリカバリしてください。NORESETLOGS 句を使用して新しい制御ファイルを作成した場合（手順 5）は、クローズ状態の完全なデータベース・リカバリ操作によってデータベースをリカバリできます。

   RESETLOGS 句を使用して新しい制御ファイルを作成した場合は、USING BACKUP CONTROL FILE を指定する必要があります。オンラインまたはアーカイブ REDO ログあるいはデータファイルが失われた場合は、これらのファイルのリカバリ手順に従ってください。

   関連項目： データベースのリカバリ、および失われた制御ファイルのリカバリ方法については、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

9. 次のいずれかの方法で、データベースをオープンします。
   - リカバリを実行しなかった場合、または手順 8 でクローズ状態の完全なデータベース・リカバリを実行した場合は、通常の手順でデータベースをオープンします。

     ```
     ALTER DATABASE OPEN;
     ```

   - 制御ファイルの作成時に RESETLOGS を指定した場合は、ALTER DATABASE 文で RESETLOGS を指定します。

     ```
     ALTER DATABASE OPEN RESETLOGS;
     ```

これでデータベースはオープンされ、使用可能になります。

制御ファイル作成後のトラブルシューティング

CREATE CONTROLFILE 文を実行した後で、エラーが発生する場合があります。ここでは、制御ファイルに関連する最も一般的なエラーについて説明します。

- 欠落したファイルや余分なファイルのチェック
- CREATE CONTROLFILE でのエラー処理

欠落したファイルや余分なファイルのチェック

新しい制御ファイルを作成し、それを使用してデータベースをオープンした後、アラート・ログをチェックして、データ・ディクショナリと制御ファイルの間で不整合（データ・ディクショナリにはデータファイルがあるが、制御ファイルには含まれていないなど）が検出されていないかを確認してください。

データ・ディクショナリ内にはデータファイルが存在していて、新しい制御ファイルには含まれていない場合、データベースは制御ファイル内に MISSING*nnnn*（*nnnn* は 10 進数のファイル番号）という名前のプレースホルダ・エントリを作成します。制御ファイル内の MISSING*nnnn* には、オフラインであること、およびメディア・リカバリを必要とすることを示すフラグが設定されます。

MISSING*nnnn* に対応する実際のデータファイルが読取り専用または通常のオフラインであった場合は、MISSING*nnnn* の名前を実際のデータファイルの名前に変更することで、データファイルをアクセス可能にできます。MISSING*nnnn* が、読取り専用や通常のオフラインのどちらでもないデータファイルに対応している場合は、名前変更操作を行ってもデータファイルはアクセス可能になりません。これは、RESETLOGS の結果によって使用できなくなったデータ

ファイルには、メディア・リカバリが必要なためです。この場合は、データファイルを含む表領域を削除する必要があります。

反対に、制御ファイルに指定されているデータファイルがデータ・ディクショナリに存在しない場合、データベースは新しい制御ファイルからそのファイルへの参照を削除します。どちらの場合も、検出された状態を通知するメッセージがアラート・ログに書き込まれます。

CREATE CONTROLFILE でのエラー処理

新しい制御ファイルの作成後、データベースをマウントしてオープンしようとしたときに Oracle Database でエラー（通常、ORA-01173、ORA-01176、ORA-01177、ORA-01215、ORA-01216 のいずれか）が出力された場合、そのエラーの最も可能性の高い原因は、CREATE CONTROLFILE 文でファイルを省略したか、またはリストに示されていないファイルを指定したかのいずれかです。この場合は、9-6 ページの手順 3 で作成したバックアップのファイルをリストアし、正しいファイル名を使用して手順 4 以降を再実行してください。

制御ファイルのバックアップ

制御ファイルをバックアップするには、ALTER DATABASE BACKUP CONTROLFILE 文を使用します。次の 2 つの方法があります。

- 次の文を使用して、制御ファイルをバイナリ・ファイル（既存の制御ファイルの複製）にバックアップを作成します。

  ```
  ALTER DATABASE BACKUP CONTROLFILE TO '/oracle/backup/control.bkp';
  ```

- 後で制御ファイルの再作成に使用できる SQL 文を生成します。

  ```
  ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
  ```

  このコマンドは、トレース・ファイルに SQL スクリプトを書き込みます。書き込まれた SQL スクリプトは、制御ファイルを再作成するために、トレース・ファイルから取得して編集できます。トレース・ファイルの名前と場所を確認するには、アラート・ログを表示します。

  関連項目：

  - 制御ファイルのバックアップの詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。
  - 8-19 ページ「アラート・ログの表示」

現行のコピーを使用した制御ファイルのリカバリ

ここでは、現行のバックアップまたは多重化コピーから制御ファイルをリカバリする方法について説明します。

制御ファイルのコピーを使用した制御ファイル破損からのリカバリ

この手順では、CONTROL_FILES パラメータで指定されている制御ファイルの 1 つが破損し、制御ファイルのディレクトリにはまだアクセス可能で、制御ファイルの多重化コピーがある場合を想定しています。

1. インスタンスを停止してから、オペレーティング・システム・コマンドを使用して、破損した制御ファイルに正常なコピーを上書きします。

```
% cp /u03/oracle/prod/control03.ctl  /u02/oracle/prod/control02.ctl
```

2. SQL*Plus を起動してデータベースをオープンします。

```
SQL> STARTUP
```

制御ファイルのコピーを使用した永続的なメディア障害からのリカバリ

この手順では、永続的なメディア障害のために、CONTROL_FILES パラメータで指定されている制御ファイルの 1 つにアクセスできず、制御ファイルの多重化コピーがある場合を想定しています。

1. インスタンスを停止してから、オペレーティング・システム・コマンドを使用して、制御ファイルの現行のコピーをアクセス可能な新しい位置にコピーします。

```
% cp /u01/oracle/prod/control01.ctl  /u04/oracle/prod/control03.ctl
```

2. 初期化パラメータ・ファイルの CONTROL_FILES パラメータを編集し、破損したファイルの位置を新しい位置に置き換えます。

```
CONTROL_FILES = (/u01/oracle/prod/control01.ctl,
                 /u02/oracle/prod/control02.ctl,
                 /u04/oracle/prod/control03.ctl)
```

3. SQL*Plus を起動してデータベースをオープンします。

```
SQL> STARTUP
```

多重制御ファイルがある場合は、CONTROL_FILES 初期化パラメータを編集することで、データベースを迅速に起動できます。破損した制御ファイルを CONTROL_FILES 設定から削除すると、即時にデータベースを再起動できます。次に、破損した制御ファイルを再作成し、CONTROL_FILES 初期化パラメータにリカバリした制御ファイルを設定してから、データベースを停止し、再起動します。

制御ファイルの削除

たとえば、制御ファイルの位置が不適切な場合は、データベースからその制御ファイルを削除できます。ただし、データベースには常に少なくとも 2 つの制御ファイルが存在する必要があります。

1. データベースを停止します。
2. データベース初期化パラメータ・ファイルの CONTROL_FILES パラメータを編集して、古い制御ファイル名を削除します。
3. データベースを再起動します。

注意： この操作では、不要な制御ファイルをディスクから物理的に削除することはできません。データベースから制御ファイルを削除した後、オペレーティング・システムのコマンドを使用して不要なファイルを削除してください。

制御ファイルのデータ・ディクショナリ・ビュー

次のビューには、制御ファイルに関する情報が表示されます。

ビュー	説明
V$DATABASE	制御ファイル内のデータベース情報が表示されます。
V$CONTROLFILE	制御ファイル名が一覧表示されます。
V$CONTROLFILE_RECORD_SECTION	制御ファイルのレコード・セクションに関する情報が表示されます。
V$PARAMETER	CONTROL_FILES 初期化パラメータで指定されている制御ファイルの名前が表示されます。

この例では、制御ファイル名が一覧表示されます。

```
SQL> SELECT NAME FROM V$CONTROLFILE;

NAME
------------------------------------
/u01/oracle/prod/control01.ctl
/u02/oracle/prod/control02.ctl
/u03/oracle/prod/control03.ctl
```

10

REDO ログの管理

この章の内容は次のとおりです。

関連項目： Oracle Database サーバーによって作成および管理される REDO ログ・ファイルの詳細は、第 15 章「Oracle Managed Files の使用」を参照してください。

REDO ログの概要

REDO ログは、リカバリ操作にとって最も重要な構造です。これは、データベースに加えられたすべての変更を発生時に格納する、2 つ以上の事前割当てファイルから構成されます。Oracle Database の各インスタンスには、インスタンスの障害時にデータベースを保護するための REDO ログが 1 つずつ対応付けられています。

REDO スレッド

複数のデータベース・インスタンスのコンテキストでは、各データベース・インスタンスの REDO ログは、REDO スレッドとも呼ばれます。標準的な構成では、Oracle Database にアクセスするデータベース・インスタンスは 1 つのみであるため、スレッドは 1 つしか存在しません。ただし、Oracle Real Application Clusters 環境では、複数のインスタンスが単一のデータベースに同時にアクセスし、各インスタンスが専用の REDO スレッドを持ちます。各インスタンスが別々の REDO スレッドを使用するため、1 つのセットの REDO ログ・ファイルで競合が回避され、その結果、潜在的なパフォーマンスのボトルネックを解消できます。

この章では、標準的な単一インスタンスの Oracle Database で、REDO ログを構成して管理する方法について説明します。文のすべての説明と例では、スレッド番号を 1 と想定します。Oracle Real Application Clusters 環境における REDO ログ・グループについては、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。

REDO ログの内容

REDO ログ・ファイルには、**REDO レコード**が書き込まれます。REDO レコードは **REDO エントリ**とも呼ばれ、**変更ベクトル**（データベース内の単一ブロックに加えられた変更の記述）のグループからなっています。たとえば、従業員表の給与値を変更する場合は、その表のデータ・セグメント・ブロック、UNDO セグメント・データ・ブロックおよび UNDO セグメントのトランザクション表の変更内容を記述する変更ベクトルを含む REDO レコードが生成されます。

REDO エントリには、UNDO セグメントなど、データベースに対するすべての変更の再構築に使用できるデータが記録されます。したがって、REDO ログによってロールバック・データも保護されます。REDO データベースを使用してデータベースをリカバリすると、データベースは REDO レコード内の変更ベクトルを読み込んで変更内容を関連ブロックに適用します。

REDO レコードは、循環方式でシステム・グローバル領域（SGA）の REDO ログ・バッファに入れられ（10-3 ページの「Oracle Database による REDO ログへの書込み」を参照）、データベースのバックグラウンド・プロセスであるログ・ライター（LGWR）によって REDO ログ・ファイルの 1 つに書き込まれます。トランザクションがコミットされると、LGWR によってそのトランザクションの REDO レコードが SGA の REDO ログ・バッファから REDO ログ・ファイルに書き込まれ、コミットされた各トランザクションの REDO レコードを識別するために**システム変更番号**（SCN）が割り当てられます。特定のトランザクションに対応付けられたすべての REDO ログ・レコードがディスク上のオンライン・ログに安全に書き込まれた場合にのみ、ユーザー・プロセスはトランザクションがコミットされたことを示す通知を受け取ります。

また、REDO レコードは、対応するトランザクションがコミットされる前に REDO ログ・ファイルに書き込むこともできます。REDO ログ・バッファがいっぱいになるか、別のトランザクションがコミットされると、一部の REDO レコードがコミットされていない可能性があっても、LGWR は REDO ログ・バッファ内のすべての REDO ログ・エントリを REDO ログ・ファイルにフラッシュします。データベースでは、必要に応じて、これらの変更をロールバックできます。

Oracle Database による REDO ログへの書込み

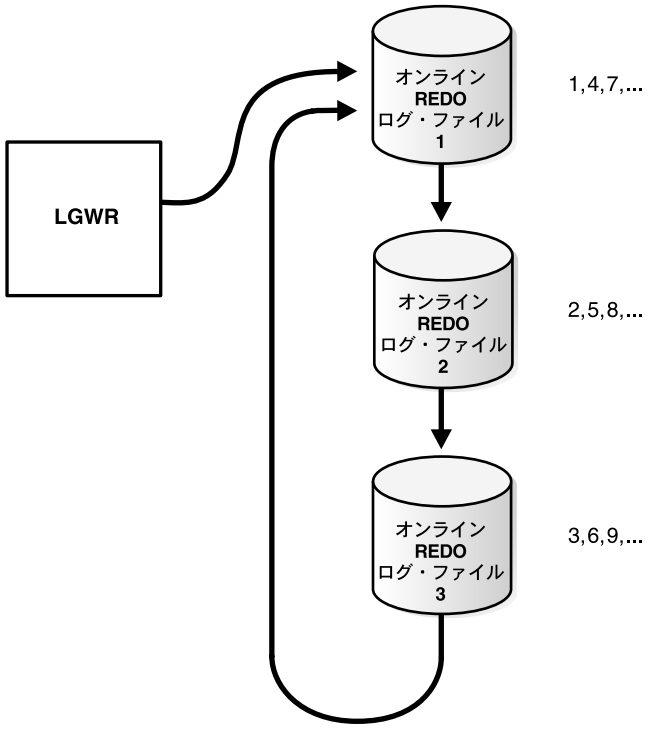
データベースの REDO ログは、2 つ以上の REDO ログ・ファイルから構成されます。データベースでは、一方のファイルのアーカイブ中にも（データベースが ARCHIVELOG モードのとき）、他方のファイルが常に書込み可能であることを保証するために、最低 2 つのファイルを必要とします。詳細は、11-1 ページの「アーカイブ REDO ログの管理」を参照してください。

LGWR は、REDO ログ・ファイルに循環方式で書き込みます。つまり、現行の REDO ログ・ファイルがいっぱいになると、LGWR は次に使用可能な REDO ログ・ファイルへの書込みを開始します。使用可能な最後の REDO ログ・ファイルがいっぱいになると、LGWR は最初の REDO ログ・ファイルに戻って書込みを行い、再び循環を開始します。図 10-1 は、REDO ログ・ファイルへの循環方式の書込みを示しています。各ラインの隣の番号は、LGWR が各 REDO ログ・ファイルに書き込む順序を示しています。

いっぱいになった REDO ログ・ファイルは、アーカイブが使用可能になっているかどうかに応じて、LGWR で再利用できます。

- アーカイブが使用禁止になっている場合（データベースが NOARCHIVELOG モードのとき）、いっぱいになった REDO ログ・ファイルは、そこに記録された変更がデータファイルに書き込まれた後に使用可能になります。
- アーカイブが使用可能になっている場合（データベースが ARCHIVELOG モードのとき）、いっぱいになった REDO ログ・ファイルは、そこに記録された変更がデータファイルに書き込まれ、かつ、そのファイルがアーカイブされた後に、LGWR で使用可能になります。

図 10-1 LGWR による REDO ログ・ファイルの再利用



アクティブ（カレント）および非アクティブな REDO ログ・ファイル

Oracle Database は、REDO ログ・ファイルを一度に 1 つのみ使用して、REDO ログ・バッファから書き込まれた REDO レコードを格納します。LGWR が書込み中の REDO ログ・ファイルを**現行**の REDO ログ・ファイルと呼びます。

インスタンス・リカバリに必要な REDO ログ・ファイルを**アクティブ**な REDO ログ・ファイルと呼びます。また、インスタンス・リカバリには不要な REDO ログ・ファイルを**非アクティブ**な REDO ログ・ファイルと呼びます。

アーカイブを使用可能にしている場合は（データベースが ARCHIVELOG モードのとき）、いずれかのアーカイバ・バックグラウンド・プロセス（ARC*n*）によって内容がアーカイブされるまで、データベースはアクティブなオンライン・ログ・ファイルの再利用または上書きができません。アーカイブが使用禁止になっている場合は（データベースが NOARCHIVELOG モードのとき）、最後の REDO ログ・ファイルがいっぱいになると、LGWR は使用可能な最初のアクティブ・ファイルを上書きして書込みを継続します。

ログ・スイッチとログ順序番号

ログ・スイッチは、データベースが、ある REDO ログ・ファイルへの書込みを終了して他のファイルへの書込みを開始するポイントです。現行の REDO ログ・ファイルが完全にいっぱいになり、引き続き次の REDO ログ・ファイルへの書込みが必要になると、通常はログ・スイッチが発生します。ただし、ログ・スイッチは、現行の REDO ログ・ファイルが完全にいっぱいになっているかどうかに関係なく、定期的に発生するように構成できます。ログ・スイッチは、手動で強制的に発生させることもできます。

Oracle Database は、ログ・スイッチが発生して LGWR が書込みを開始するたびに、各 REDO ログ・ファイルに新しい**ログ順序番号**を割り当てます。データベースが REDO ログ・ファイルをアーカイブしても、そのファイルのログ順序番号は変わりません。一巡して再び使用可能になった REDO ログ・ファイルには、次に使用可能なログ順序番号が割り当てられます。

各オンライン REDO ログ・ファイルまたはアーカイブ REDO ログ・ファイルは、そのログ順序番号で一意に識別されます。クラッシュ、インスタンスまたはメディア・リカバリ中に、データベースは必要なアーカイブおよび REDO ログ・ファイルのログ順序番号を使用して、REDO ログ・ファイルを昇順に正しく適用します。

REDO ログの計画

ここでは、データベース・インスタンスの REDO ログを構成するときに考慮する必要があるガイドラインについて説明します。この項の内容は、次のとおりです。

- REDO ログ・ファイルの多重化
- 異なるディスクへの REDO ログ・メンバーの配置
- REDO ログ・メンバーのサイズの設定
- 適切な REDO ログ・ファイル数の選択
- アーカイブ・タイムラグの制御

REDO ログ・ファイルの多重化

REDO ログ自体が影響を受ける障害に備え、Oracle Database では REDO ログを**多重化**できます。つまり、REDO ログの 2 つ以上の同一コピーを自動的に別の場所に保持できます。最大の効果を得るためには、この保存場所を別々のディスクにする必要があります。ただし、REDO ログのすべてのコピーが同一ディスク上にある場合でも、冗長性によって、I/O エラー、ファイルの破損などから保護できます。REDO ログ・ファイルを多重化すると、LGWR は同じ REDO ログ情報を複数の REDO ログ・ファイルに書き込むため、REDO ログの単一の障害箇所は問題になりません。

多重化は、REDO ログ・ファイルのグループを作成することによって実装されます。**グループ**は、REDO ログ・ファイルとその多重化されたコピーで構成されます。それぞれの同一コピーはグループの**メンバー**と呼ばれます。各 REDO ログ・グループは、グループ 1、グループ 2 のように数値で定義されます。

図 10-2 多重 REDO ログ・ファイル

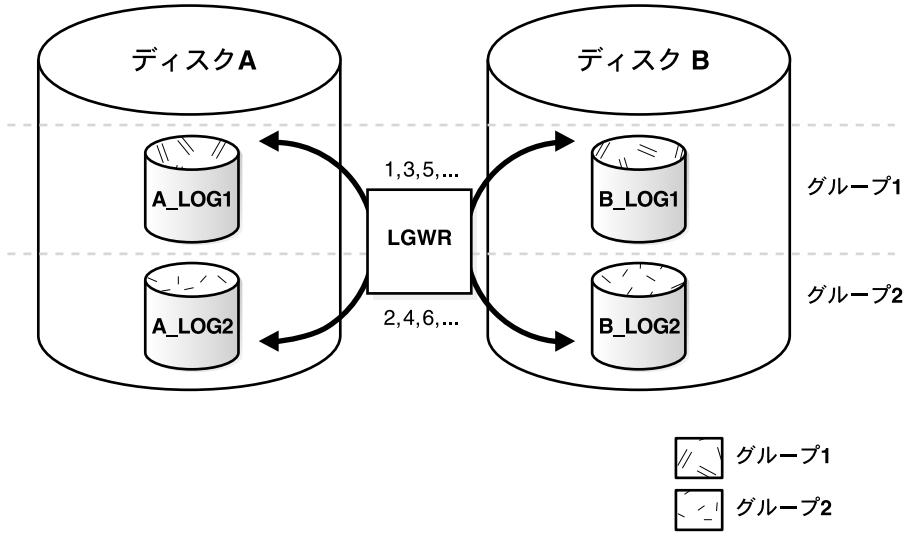


図 10-2 の A_LOG1 と B_LOG1 はどちらもグループ 1 のメンバーで、A_LOG2 と B_LOG2 はどちらもグループ 2 のメンバーです。グループ内の各メンバーのサイズは、完全に一致させてください。

同一のログ順序番号が LGWR により割り当てられることが示すように、ログ・ファイル・グループの各メンバーは同時にアクティブになり、LGWR によって同時に書き込まれます。図 10-2 では、LGWR は最初に A_LOG1 と B_LOG1 の双方に同時に書き込みます。次に A_LOG2 と B_LOG2 の双方に同時に書き込みます。以降も同様です。LGWR が異なるグループのメンバー（A_LOG1 と B_LOG2 など）に同時に書き込むことはありません。

注意： REDO ログ・ファイルは多重化することをお薦めします。これは、リカバリが必要になったときにログ・ファイルのデータが失われていると、致命的な事態を招くおそれがあるためです。REDO ログを多重化すると、データベースの実行時の I/O の量が増加するため注意してください。構成によっては、これによってデータベース・パフォーマンス全体が影響を受けます。

REDO ログの障害への対処

LGWR がグループのメンバーに書き込めない場合、データベースはそのメンバーに INVALID を示すマークを付け、LGWR トレース・ファイルとデータベース・アラート・ログに、アクセス不可能ファイルの問題を示すエラー・メッセージを書き込みます。REDO ログ・メンバーが使用不可能な場合の LGWR 固有の処理は、次の表に示すように、使用不可能になった理由によって決定します。

条件	LGWR の処理
LGWR がグループ内の最低 1 つのメンバーに正常に書き込める場合	通常どおり書込みが行われます。LGWR はグループのうち使用可能なメンバーにのみ書き込みし、使用不可のメンバーは無視します。
グループをアーカイブする必要があるため、LGWR がログ・スイッチの発生時に次のグループにアクセスできない場合	データベース操作は、グループが使用可能になるか、グループがアーカイブされるまで一時的に停止します。

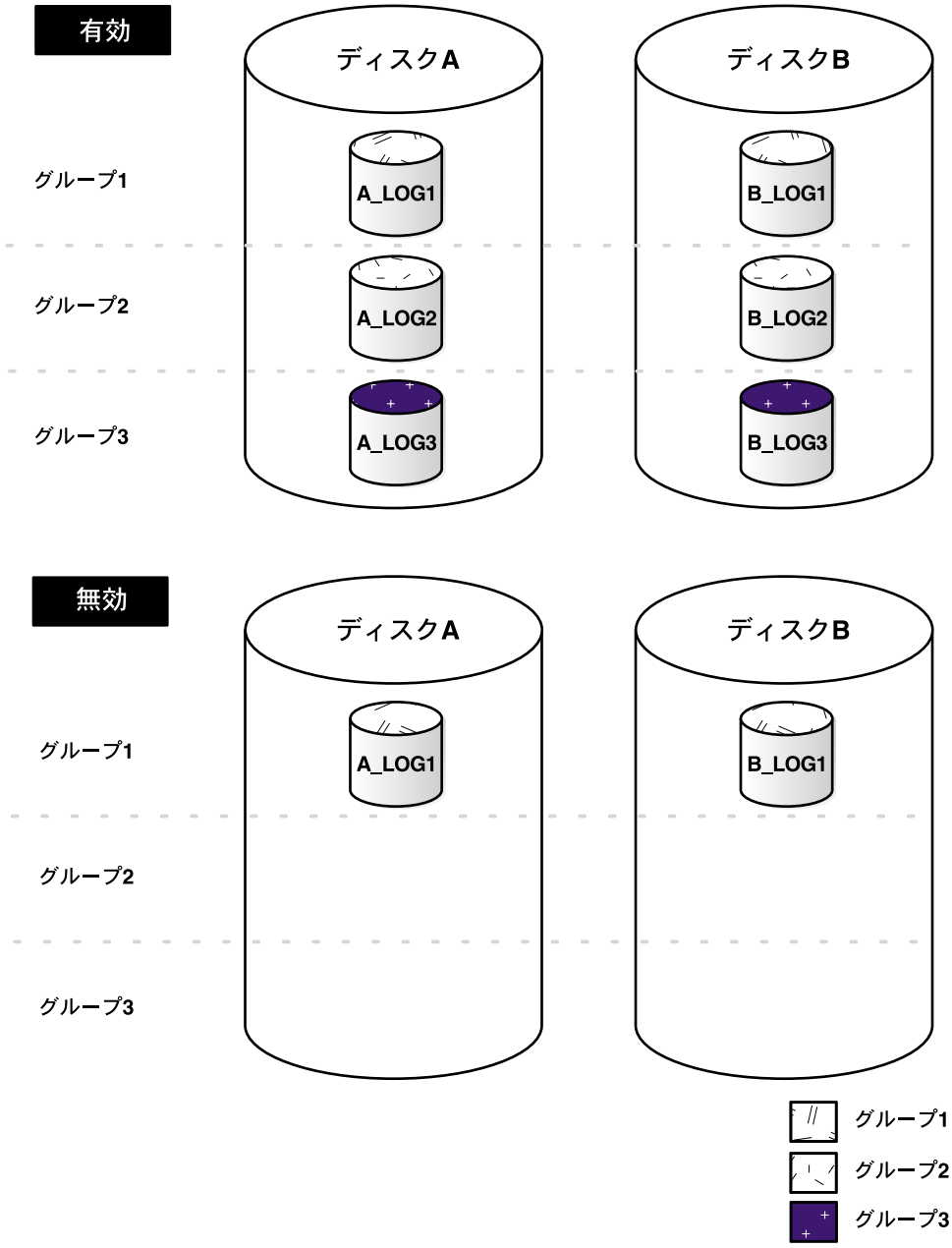
条件	LGWR の処理
メディア障害のため、ログ・スイッチの発生時に LGWR が次のグループのどのメンバーにもアクセスできない場合	Oracle Database はエラーを返し、データベース・インスタンスは停止します。この場合は、データベース上で REDO ログ・ファイルの損失からのメディア・リカバリを実行する必要があります。 データベースのチェックポイントが実行済で、損失した REDO ログがそのチェックポイント以前の場合、その REDO ログに記録されたデータは、データベースによってデータファイルに保存されているため、メディア・リカバリの必要はありません。アクセスできない REDO ログ・グループのみ削除してください。データベースによって不良ログがアーカイブされていない場合は、`ALTER DATABASE CLEAR UNARCHIVED LOG` を使用してアーカイブを使用禁止にしてから、ログを削除してください。
LGWR が書込み中に、グループのすべてのメンバーに突然アクセスできなくなった場合	Oracle Database はエラーを返し、データベース・インスタンスは即時に停止します。この場合は、メディア・リカバリを実行する必要があります。ログのドライブを不注意からオフにした場合など、ログを含むメディアが実際には失われていなければ、メディア・リカバリは不要です。この場合は、ドライブをオンに戻して、データベースで自動インスタンス・リカバリを実行します。

有効な構成と無効な構成

ほとんどの場合、多重 REDO ログ・ファイルを対称型にする必要があります。つまり、REDO ログのどのグループも、メンバーが同数になるようにします。ただし、データベースでは、必ずしも多重 REDO ログ・ファイルを対称型にする必要はありません。たとえば、あるグループのメンバーは 1 つ、他のグループのメンバーは 2 つでもかまいません。この構成は、一部の REDO ログ・メンバーには一時的に影響しても、他のメンバーには影響しないディスク障害からデータベースを保護します。

インスタンスの REDO ログに関する唯一の要件は、最低 2 つのグループを持つことです。図 10-3 は、多重 REDO ログに有効な構成と無効な構成を示しています。2 番目の構成は、グループが 1 つしかないため無効です。

図 10-3 多重 REDO ログ・ファイルの有効な構成と無効な構成



異なるディスクへの REDO ログ・メンバーの配置

多重 REDO ログ・ファイルを設定する場合は、グループのメンバーを異なる物理ディスク上に配置します。このようにすると、1 つのディスクで障害が発生しても、LGWR が使用できなくなるのはグループの 1 つのメンバーのみで、それ以外のメンバーには引き続きアクセスできるので、インスタンスは継続して機能します。

REDO ログをアーカイブする場合は、バックグラウンド・プロセス LGWR と ARC*n* 間の競合をなくすために、REDO ログ・メンバーを複数のディスクに分散します。たとえば、多重化した REDO ログ・メンバーのグループが 2 つある場合（二重化された REDO ログ）は、各メンバーを異なるディスク上に配置し、アーカイブ先を 5 つ目のディスクに設定します。このようにすると、LGWR（メンバーへの書込み）と ARC*n*（メンバーの読込み）間の競合は発生しません。

データ・ブロックや REDO レコードの書込み時の競合を減らすために、データファイルも REDO ログ・ファイルとは異なるディスク上に配置することをお薦めします。

REDO ログ・メンバーのサイズの設定

REDO ログ・ファイルのサイズを設定するときは、REDO ログをアーカイブするかどうかを考慮してください。REDO ログ・ファイルのサイズは、いっぱいになったグループを 1 つのオフライン記憶メディア（テープやディスクなど）にアーカイブできるとともに、そのメディア上の未使用領域が最小になるように設定します。たとえば、いっぱいになった 1 つの REDO ログ・グループを 1 本のテープにアーカイブし、そのテープの記憶容量の 49% が未使用のまま残っているとします。この場合は、REDO ログ・ファイルのサイズを小さくして、テープごとに 2 つずつ REDO ログ・グループがアーカイブされるように構成することをお薦めします。

同一の多重 REDO ログ・グループのメンバーは、すべて同じサイズにする必要があります。異なるグループのメンバーは異なるサイズにすることができます。ただし、グループ間でファイル・サイズを変えても特に利点はありません。ログ・スイッチ間にチェックポイントが発生するように設定していない場合は、グループのサイズを同一に設定して、チェックポイントが一定の間隔で発生することを保証してください。

REDO ログ・ファイルの最小サイズは 4MB です。

> **関連項目：** 使用しているオペレーティング・システム固有の Oracle マニュアルを参照してください。REDO ログ・ファイルのデフォルトのサイズは、オペレーティング・システムによって異なります。

適切な REDO ログ・ファイル数の選択

データベース・インスタンスに対する REDO ログ・ファイルの適切な数を決定する最良の方法は、様々な構成をテストすることです。最適な構成では、LGWR が REDO ログ情報を書き込むのを妨げない最小の数がグループ数になります。

データベース・インスタンスに必要なグループが 2 つのみの場合もあります。また、LGWR が使用可能な再利用グループが常に存在するようにするため、データベース・インスタンスにグループを追加する必要がある場合もあります。テスト中に、現行の REDO ログ構成に問題がないかどうかを確認するには、LGWR トレース・ファイルおよびデータベース・アラート・ログの内容を調べるのが最も容易です。チェックポイントが未完了か、またはグループがアーカイブされていないため、LGWR が頻繁にグループを待機することをメッセージが示す場合は、グループを追加してください。

インスタンスの REDO ログ構成を設定または変更する前に、REDO ログ・ファイル数を制限するパラメータを検討してください。次のパラメータは、データベースに追加できる REDO ログ・ファイルの数を制限します。

- `CREATE DATABASE` 文の `MAXLOGFILES` パラメータは、データベース当たりの REDO ログ・ファイルの最大グループ数を決定します。グループの値は 1 ～ `MAXLOGFILES` です。互換性レベルが 10.2.0 より前に設定されている場合、この上限値を変更する唯一の方法はデータベースまたは制御ファイルを再作成することです。したがって、データベースを作成する前にこの上限値を十分検討してください。互換性が 10.2.0 以上に設定されている場合は、`MAXLOGFILES` 制限を超えることが可能で、制御ファイルは必要に応じて拡張されます。

CREATE DATABASE 文に MAXLOGFILES パラメータが指定されていない場合は、オペレーティング・システム固有のデフォルト値が使用されます。

- CREATE DATABASE 文の MAXLOGMEMBERS パラメータは、グループに含まれるメンバーの最大値を決定します。この上限値を変更する唯一の方法は、MAXLOGFILES の場合と同様、データベースまたは制御ファイルを再作成することです。したがって、データベースを作成する前にこの上限値を十分検討してください。CREATE DATABASE 文に MAXLOGMEMBERS パラメータが指定されていない場合は、オペレーティング・システムのデフォルト値が使用されます。

関連項目：

- MAXLOGFILES パラメータおよび MAXLOGMEMBERS パラメータのデフォルト値と有効な値については、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。

アーカイブ・タイムラグの制御

すべての有効な REDO ログ・スレッドについて、そのカレント・ログを一定の間隔で強制的に切り替えることができます。プライマリ / スタンバイ・データベース構成では、プライマリ・サイトの REDO ログをアーカイブしてスタンバイ・データベースにトランスポートすることによって、スタンバイ・データベースを変更できます。プライマリ・データベースで変更が発生してから、スタンバイ・データベースでその変更が適用されるまで、タイムラグが発生します。これは、プライマリ・データベースで REDO ログがアーカイブ REDO ログにアーカイブされてから、スタンバイ・データベースにトランスポートされるまで、スタンバイ・データベースは変更を待機する必要があるためです。ARCHIVE_LAG_TARGET 初期化パラメータを設定すると、このタイムラグを制限できます。このパラメータを設定することによって、許容するタイムラグの長さを秒単位で指定できます。

ARCHIVE_LAG_TARGET 初期化パラメータの設定

ARCHIVE_LAG_TARGET 初期化パラメータを設定すると、データベースはインスタンスの現行の REDO ログを定期的に調べます。次の条件が満たされると、インスタンスはログを切り替えます。

- カレント・ログが *n* 秒前に作成され、カレント・ログのアーカイブ見積り時間が *m* 秒（この時間はカレント・ログで使用されている REDO ブロック数に比例する）の場合に、*n*+*m* が ARCHIVE_LAG_TARGET 初期化パラメータの値を超えている。
- カレント・ログに REDO レコードが含まれている。

Oracle Real Application Clusters 環境では、他のスレッドが遅れている場合、前述の条件を検出したインスタンスによって他のスレッドも切り替えられ、ログがアーカイブされます。これは、Oracle Real Application Clusters で 2 つのノードを持つプライマリ / セカンダリ構成を実行しているときのように、クラスタ内に他のインスタンスよりも稼働率の低いインスタンスがある場合に特に有効です。

ARCHIVE_LAG_TARGET 初期化パラメータは、Oracle Data Guard 環境が非データ消失モードで構成されていない場合に、プライマリ側で停止や障害が起こったとき、スタンバイ側で失ってもかまわない REDO 時間（秒）のターゲットを指定します。また、このパラメータは、プライマリ・データベースのカレント・ログが使用される時間の上限（秒単位）も指定します。アーカイブ見積り時間も考慮されるため、これはログ・スイッチ時間そのものではありません。

次の初期化パラメータ設定では、ログ・スイッチ間隔を 30 分（標準的な値）に設定します。

```
ARCHIVE_LAG_TARGET = 1800
```

0（ゼロ）を指定すると、時間ベースのログ・スイッチ機能は使用禁止になります。これはデフォルトの設定です。

スタンバイ・データベースが存在していなくても、ARCHIVE_LAG_TARGET 初期化パラメータを設定できます。たとえば、強制的にログ・スイッチを発生させてアーカイブを行うために、ARCHIVE_LAG_TARGET 初期化パラメータを設定できます。

ARCHIVE_LAG_TARGET は動的パラメータで、ALTER SYSTEM SET 文を使用して設定できます。

> **注意：** Oracle Real Application Clusters 環境では、すべてのインスタンスの ARCHIVE_LAG_TARGET パラメータに同じ値を指定する必要があります。異なる値を設定すると、予測できない動作が発生します。

ARCHIVE_LAG_TARGET の設定に影響する要因

ARCHIVE_LAG_TARGET パラメータを設定するかどうか、またはその設定値を決定するには、次の要因を考慮してください。

- 切替え（およびアーカイブ）に伴うオーバーヘッド
- ログがいっぱいになったときに発生する通常のログ・スイッチの頻度
- スタンバイ・データベースで許容できる REDO 損失の量

指定した間隔より短い頻度ですでにログ・スイッチが自然に発生している状況では、ARCHIVE_LAG_TARGET を設定しても特に利点はありません。ただし、REDO の生成速度が一定でない場合は、時間間隔を指定することで、各カレント・ログが使用される時間範囲の上限を設定できます。

ARCHIVE_LAG_TARGET 初期化パラメータに極端に小さい値を指定すると、パフォーマンスに悪影響を及ぼすことがあります。これは、小さい値によって頻繁にログ・スイッチが発生するためです。このパラメータには、プライマリ・データベースのパフォーマンスを低下させないように適切な値を設定してください。

REDO ログ・グループおよびメンバーの作成

データベースの REDO ログを事前に計画し、データベースの作成時に必要な REDO ログ・ファイルのグループとメンバーをすべて作成してください。しかし、グループやメンバーの追加作成が必要な状況もあります。たとえば、REDO ログにグループを追加することによって、REDO ログ・グループの可用性の問題を解決できます。

新たに REDO ログ・グループおよびメンバーを作成するには、ALTER DATABASE システム権限が必要です。データベースには、最大 MAXLOGFILES 個までグループを作成できます。

> **関連項目：** ALTER DATABASE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

REDO ログ・グループの作成

REDO ログ・ファイルの新しいグループを作成するには、SQL 文 ALTER DATABASE で ADD LOGFILE 句を指定します。

次の文は、データベースに新しい REDO ログ・グループを追加します。

```
ALTER DATABASE
  ADD LOGFILE ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo') SIZE 4M;
```

> **注意：** オペレーティング・システム上のファイルを作成する位置を指定するには、新しいログ・メンバーのファイル名を絶対パスで指定してください。そうしないと、ファイルはオペレーティング・システムごとに異なるデータベースのデフォルトのディレクトリまたはカレント・ディレクトリに作成されます。

また、次のように GROUP 句を使用して、グループの識別番号を指定することもできます。

```
ALTER DATABASE
  ADD LOGFILE GROUP 10 ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo')
      SIZE 4M;
```

グループ番号を使用することによって、REDO ログ・グループの管理がより簡単になります。ただし、グループ番号には 1 から MAXLOGFILES の範囲の値を使用する必要があります。REDO ログ・ファイルのグループ番号をスキップする（つまり、グループに 10、20、30 などの番号を付ける）と、データベースの制御ファイル内で不要な領域が消費されてしまうため、グループ番号はスキップしないでください。

REDO ログ・メンバーの作成

完全な REDO ログ・ファイルのグループを作成する必要がない場合もあります。つまり、グループはすでに存在しているが、そのグループの 1 つ以上のメンバーが（ディスク障害などにより）削除されたために完全ではない場合です。このような場合は、既存のグループに新しいメンバーを追加できます。

既存のグループ用に新しい REDO ログ・メンバーを作成するには、SQL 文 ALTER DATABASE で ADD LOGFILE MEMBER 句を指定します。次の文は、REDO ログ・グループ 2 に新しい REDO ログ・メンバーを追加します。

```
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/dbs/log2b.rdo' TO GROUP 2;
```

REDO ログ・メンバーを追加する際、ファイル名は指定する必要がありますが、サイズを指定する必要はありません。新しいメンバーのサイズは、既存グループのメンバーのサイズによって決まります。

ALTER DATABASE 文を使用するときは、次の例のように、TO 句にグループの他のメンバーをすべて指定することによって、目的のグループを選択的に識別できます。

```
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/dbs/log2c.rdo'
    TO ('/oracle/dbs/log2a.rdo', '/oracle/dbs/log2b.rdo');
```

注意： オペレーティング・システム上のファイルを作成する位置を指定するには、新しいログ・メンバーのファイル名を絶対パスで指定してください。そうしないと、ファイルはオペレーティング・システムごとに異なるデータベースのデフォルトのディレクトリまたはカレント・ディレクトリに作成されます。また、新しいログ・メンバーの状態は INVALID として表示されるので注意してください。これは正常であり、最初に使用するときにアクティブ（空白）に変更されます。

REDO ログ・メンバーの再配置および名前変更

オペレーティング・システムのコマンドを使用して REDO ログを再配置し、ALTER DATABASE 文を使用してデータベースにその REDO ログの新しい名前（位置）を通知できます。たとえば、REDO ログ・ファイルが現在保存されているディスクを削除する場合や、複数のデータファイルや REDO ログ・ファイルが同一のディスク上に格納されていて、競合を少なくするためにそれらを分離する場合に、この手順が必要となります。

REDO ログ・メンバーの名前を変更するには、ALTER DATABASE システム権限が必要です。さらに、ファイルを目的の位置にコピーするためのオペレーティング・システム権限と、データベースをオープンしてバックアップするための権限が必要となることもあります。

REDO ログを再配置したり、データベースにその他の構造上の変更を加えたりする前には、各操作の実行中に発生する問題に備えて、データベース全体のバックアップを作成してください。また、今後発生する可能性のある問題に備えて、一連の REDO ログ・ファイルを名前変更または再配置した後、ただちにデータベースの制御ファイルのバックアップを作成してください。

REDO ログを再配置する手順は、次のとおりです。これらの手順では、次の状況を想定しています。

- ログ・ファイルは、diska および diskb という 2 つのディスク上にあります。
- REDO ログは多重化されています。最初のグループはメンバー /diska/logs/log1a.rdo と /diskb/logs/log1b.rdo からなり、2 番目のグループはメンバー /diska/logs/log2a.rdo と /diskb/logs/log2b.rdo からなっています。
- diska にある REDO ログ・ファイルを diskc に再配置する必要があります。新しいファイル名には新しい位置が反映され、/diskc/logs/log1c.rdo および /diskc/logs/log2c.rdo となります。

REDO ログ・メンバーの名前変更の手順

1. データベースを停止します。

```
SHUTDOWN
```

2. REDO ログ・ファイルを新しい位置にコピーします。

REDO ログ・メンバーなどのオペレーティング・システム・ファイルは、適切なオペレーティング・システム・コマンドを使用してコピーする必要があります。ファイルのコピーに関する説明は、オペレーティング・システム固有のマニュアルを参照してください。

> **注意：** SQL*Plus の HOST コマンドを使用すると、既存の SQL*Plus を終了せずにオペレーティング・システム・コマンドを使用してファイルをコピーできます（その他のオペレーティング・システムのコマンドも実行できます）。HOST という語のかわりに 1 文字を使用するオペレーティング・システムもあります。たとえば、UNIX では感嘆符（!）が使用できます。

次の例では、オペレーティング・システム・コマンド（UNIX）を使用して、REDO ログ・メンバーを新しい位置に移動しています。

```
mv /diska/logs/log1a.rdo /diskc/logs/log1c.rdo
mv /diska/logs/log2a.rdo /diskc/logs/log2c.rdo
```

3. データベースを起動して、マウントします。ただし、オープンはしません。

```
CONNECT / as SYSDBA
STARTUP MOUNT
```

4. REDO ログ・メンバーの名前を変更します。

   ALTER DATABASE 文で RENAME FILE 句を使用して、データベースの REDO ログ・ファイルの名前を変更します。

```
ALTER DATABASE
  RENAME FILE '/diska/logs/log1a.rdo', '/diska/logs/log2a.rdo'
           TO '/diskc/logs/log1c.rdo', '/diskc/logs/log2c.rdo';
```

5. 通常の操作を実行するためにデータベースをオープンします。

   REDO ログの変更は、データベースがオープンされたときに有効となります。

```
ALTER DATABASE OPEN;
```

REDO ログ・グループおよびメンバーの削除

場合によっては、REDO ログ・メンバーを含むグループ全体を削除できます。たとえば、インスタンスの REDO ログのグループ数を少なくする場合などです。また、1 つ以上の特定の REDO ログ・メンバーの削除が必要になる場合もあります。たとえば、ディスク障害が発生した場合は、アクセスできないファイルに書き込まれないように、障害のあったディスク上の REDO ログ・ファイルをすべて削除します。これ以外にも、特定の REDO ログ・ファイルが不要になることがあります。たとえば、適切ではない位置にファイルを配置した場合です。

ログ・グループの削除

REDO ログ・グループを削除するには、ALTER DATABASE システム権限が必要です。REDO ログ・グループを削除する前に、次の制限と注意点について検討してください。

- グループ内のメンバー数にかかわらず、インスタンスには少なくとも 2 つの REDO ログ・ファイルのグループが必要です（1 つのグループは 1 つ以上のメンバーから構成されます）。
- REDO ログ・グループは、非アクティブである場合にのみ削除できます。カレントのグループを削除する必要がある場合は、最初にログ・スイッチを発生させる必要があります。
- 削除する前に、REDO ログ・グループがアーカイブされていることを確認します（アーカイブが使用可能になっている場合）。アーカイブされているかどうかを確認するには、V$LOG ビューを使用します。

```
SELECT GROUP#, ARCHIVED, STATUS FROM V$LOG;

   GROUP# ARC STATUS
--------- --- ----------------
        1 YES ACTIVE
        2 NO  CURRENT
        3 YES INACTIVE
        4 YES INACTIVE
```

SQL 文 ALTER DATABASE に DROP LOGFILE 句を指定して、REDO ログ・グループを削除します。

次の文は、REDO ログ・グループ 3 を削除します。

```
ALTER DATABASE DROP LOGFILE GROUP 3;
```

データベースから REDO ログ・グループを削除するときは、Oracle Managed Files 機能を使用していないかぎり、オペレーティング・システム・ファイルはディスクから削除されません。より正確に言えば、対応するデータベースの制御ファイルが更新されて、そのグループのメンバーがデータベース構造から削除されます。REDO ログ・グループを削除した後は、この処理が正常に終了したことを確認してから、適切なオペレーティング・システム・コマンドを使用して、削除した REDO ログ・ファイルを実際に削除します。

Oracle Managed Files 機能を使用している場合は、オペレーティング・システム・ファイルのクリーン・アップが自動的に実行されます。

REDO ログ・メンバーの削除

REDO ログ・メンバーを削除するには、ALTER DATABASE システム権限が必要です。各 REDO ログ・メンバーを削除する前に、次の制限と注意点について検討してください。

- REDO ログ・ファイルを削除することにより、多重 REDO ログ・ファイルを一時的に非対称にしても問題はありません。たとえば、多重化した REDO ログ・ファイルのグループを使用している場合、他のすべてのグループにメンバーが 2 つずつ残っていても、あるグループのメンバーを 1 つ削除できます。ただし、すべてのグループに少なくともメンバーが 2 つ存在するように、この状態をただちに訂正し、REDO ログの単一の障害箇所が発生する可能性を取り除いてください。
- グループ内のメンバー数にかかわらず、インスタンスには常に、少なくとも 2 つの有効な REDO ログ・ファイル・グループが必要です（1 つのグループは 1 つ以上のメンバーから構成されます）。削除するメンバーがグループの最後の有効なメンバーである場合は、他のメンバーが有効にならないかぎり、そのメンバーを削除できません。REDO ログ・ファイルの状態を確認するには、V$LOGFILE ビューを使用します。REDO ログ・ファイルは、データベースがアクセスできないと INVALID になります。データベースがそのログ・ファイルを完全でない、または正しくないと判断すると、そのログ・ファイルは STALE になります。この失効したログ・ファイルは、次にそのグループがアクティブ・グループになったときに、再び有効になります。
- REDO ログ・メンバーは、アクティブ・グループまたはカレント・グループの一部でない場合にのみ削除できます。アクティブ・グループのメンバーを削除する場合は、最初にログ・スイッチを発生させます。
- メンバーを削除する前に、その REDO ログ・メンバーが属するグループがアーカイブされていることを確認します（アーカイブが使用可能になっている場合）。アーカイブされているかどうかを確認するには、V$LOG ビューを使用します。

非アクティブである特定の REDO ログ・メンバーを削除するには、ALTER DATABASE 文で DROP LOGFILE MEMBER 句を指定します。

次の文は、REDO ログ /oracle/dbs/log3c.rdo を削除します。

```
ALTER DATABASE DROP LOGFILE MEMBER '/oracle/dbs/log3c.rdo';
```

データベースから REDO ログ・メンバーを削除しても、オペレーティング・システム・ファイルはディスクから削除されません。より正確に言えば、対応するデータベースの制御ファイルが更新されて、データベース構造からメンバーが削除されます。REDO ログ・ファイルを削除した後は、処理が正常終了したことを確認してから、適切なオペレーティング・システム・コマンドを使用して、削除した REDO ログ・ファイルを実際に削除します。

アクティブ・グループのメンバーを削除するには、最初にログ・スイッチを発生させる必要があります。

ログ・スイッチの強制

ログ・スイッチは、LGWR がある REDO ログ・グループへの書込みを中止して、別のログ・グループへの書込みを開始するときに発生します。デフォルトでは、現行の REDO ログ・ファイル・グループがいっぱいになると、ログ・スイッチが自動的に発生します。

REDO ログのメンテナンス操作を実行するために、ログ・スイッチを強制的に発生させて、現在アクティブなグループを非アクティブの状態に変更できます。たとえば、現在アクティブなグループを削除する場合は、アクティブでない状態になるまでそのグループを削除できません。また、現在アクティブなグループのメンバーが完全にいっぱいになる前に、特定の時点でそのグループをアーカイブする必要がある場合にも、ログ・スイッチの強制的な実行が必要です。このオプションは、いっぱいになるまで長い時間を必要とする大きな REDO ログ・ファイルが含まれた構成で有効です。

ログ・スイッチを強制するには、ALTER SYSTEM 権限が必要です。ALTER SYSTEM 文で SWITCH LOGFILE 句を指定します。

次の文は、ログ・スイッチを強制します。

```
ALTER SYSTEM SWITCH LOGFILE;
```

REDO ログ・ファイル内のブロックの検証

データベースは、チェックサムを使用して REDO ログ・ファイル内のブロックを検証するように構成できます。初期化パラメータ DB_BLOCK_CHECKSUM を TYPICAL（デフォルト）に設定すると、ディスクに書き込まれる各データベース・ブロック（カレント・ログに書き込まれている各 REDO ログ・ブロックを含む）に対するチェックサムが計算されます。チェックサムは、ブロックのヘッダーに格納されます。

Oracle Database は、チェックサムを使用して REDO ログ・ブロック内の破損を検出します。リカバリ処理中に REDO ログ・ブロックがアーカイブ・ログから読み込まれるとき、およびブロックがアーカイブ・ログ・ファイルに書き込まれるとき、そのブロックの検証が行われます。破損が検出されると、エラーが発生しアラート・ログに書き込まれます。

REDO ログ・ブロックのアーカイブ時に破損が検出されると、システムは、グループ内の別のメンバーからそのブロックを読み込もうとします。REDO ログ・グループ内のすべてのメンバーでブロックが破損していると、アーカイブ処理は継続できません。

DB_BLOCK_CHECKSUM パラメータの値は、ALTER SYSTEM 文で動的に変更できます。

注意： DB_BLOCK_CHECKSUM を使用可能にすると、わずかなオーバーヘッドとデータベースのパフォーマンスの低下が発生します。データベースのパフォーマンスを監視して、パフォーマンスを犠牲にしてでもデータ・ブロックのチェックサムを使用して破損を検出する利点があるかを判断してください。

関連項目： DB_BLOCK_CHECKSUM 初期化パラメータの説明は、『Oracle Database リファレンス』を参照してください。

REDO ログ・ファイルの初期化

データベースがオープンしている間に REDO ログ・ファイルが破損し、その結果アーカイブが継続できなくなり、データベース・アクティビティが停止することがあります。このような状況では、ALTER DATABASE CLEAR LOGFILE 文を使用して、データベースを停止せずにファイルを再初期化できます。

次の文は、REDO ログ・グループ 3 のログ・ファイルを初期化します。

```
ALTER DATABASE CLEAR LOGFILE GROUP 3;
```

この文は、REDO ログの削除が不可能な次の 2 つの状況に対応できます。

- ログ・グループが 2 つのみの場合
- 破損した REDO ログ・ファイルがカレント・グループに属する場合

破損した REDO ログ・ファイルがアーカイブされていない場合は、この文に UNARCHIVED キーワードを使用します。

```
ALTER DATABASE CLEAR UNARCHIVED LOGFILE GROUP 3;
```

この文によって、破損した REDO ログは初期化され、アーカイブを回避できます。初期化された REDO ログは、アーカイブされていなくても使用できます。

バックアップのリカバリに必要なログ・ファイルを初期化すると、そのバックアップからのリカバリ処理ができなくなります。データベースは、そのバックアップからのリカバリ処理ができないことを示すメッセージをアラート・ログに書き込みます。

注意： アーカイブされていない REDO ログ・ファイルを初期化する場合は、データベースのバックアップをもう 1 つ作成する必要があります。

オフライン表領域をオンラインにするために必要な、アーカイブされていない REDO ログを初期化するには、ALTER DATABASE CLEAR LOGFILE 文で UNRECOVERABLE DATAFILE 句を指定します。

オフライン表領域をオンラインにするために必要な REDO ログを初期化すると、その表領域は二度とオンラインにはできません。表領域を削除するか、不完全リカバリを実行する必要があります。正常にオフライン化された表領域には、リカバリは必要ありません。

REDO ログのデータ・ディクショナリ・ビュー

次のビューには、REDO ログに関する情報が表示されます。

ビュー	説明
V$LOG	制御ファイルの REDO ログ・ファイル情報が表示されます。
V$LOGFILE	REDO ログ・グループとメンバーおよびメンバーの状態を識別します。
V$LOG_HISTORY	ログの履歴情報が含まれます。

次の問合せは、データベースの REDO ログに関する制御ファイル情報を返します。

```
SELECT * FROM V$LOG;

GROUP# THREAD#   SEQ   BYTES  MEMBERS  ARC STATUS     FIRST_CHANGE# FIRST_TIM
------ ------- ----- -------  -------  --- ---------  ------------- ---------
     1       1 10605 1048576        1  YES ACTIVE          11515628 16-APR-00
     2       1 10606 1048576        1  NO  CURRENT         11517595 16-APR-00
     3       1 10603 1048576        1  YES INACTIVE        11511666 16-APR-00
     4       1 10604 1048576        1  YES INACTIVE        11513647 16-APR-00
```

グループのすべてのメンバーの名前を表示するには、次の問合せを使用します。

```
SELECT * FROM V$LOGFILE;

GROUP#   STATUS  MEMBER
------  -------  ----------------------------------
     1           D:¥ORANT¥ORADATA¥IDDB2¥REDO04.LOG
     2           D:¥ORANT¥ORADATA¥IDDB2¥REDO03.LOG
     3           D:¥ORANT¥ORADATA¥IDDB2¥REDO02.LOG
     4           D:¥ORANT¥ORADATA¥IDDB2¥REDO01.LOG
```

メンバーの STATUS が空白の場合、そのファイルは使用中です。

関連項目： これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。

11

アーカイブ REDO ログの管理

この章の内容は次のとおりです。

関連項目：

- Oracle Database サーバーによって作成および管理されるアーカイブ REDO ログの作成の詳細は、第 15 章「Oracle Managed Files の使用」を参照してください。
- Oracle Real Application Clusters 環境におけるアーカイブに固有の情報は、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。

アーカイブ REDO ログの概要

Oracle Database では、いっぱいになった REDO ログ・ファイルのグループを 1 つ以上のオフライン・アーカイブ先に保存できます。これを総称して、**アーカイブ REDO ログ**と呼びます。REDO ログをアーカイブ REDO ログに変更するプロセスを**アーカイブ**と呼びます。このプロセスを実行できるのは、データベースが **ARCHIVELOG モード**で稼働しているときのみです。自動アーカイブと手動アーカイブのいずれかを選択できます。

アーカイブ REDO ログ・ファイルは、REDO ログ・グループのいっぱいになったメンバーのうちのコピーの 1 つです。このファイルには、REDO ログ・グループの同一メンバーの REDO エントリと一意のログ順序番号が格納されています。たとえば、REDO ログを多重化しており、グループ 1 に同一メンバー・ファイル `a_log1` および `b_log1` が含まれている場合は、アーカイバ・プロセス（ARC*n*）によってこれらのメンバー・ファイルの 1 つがアーカイブされます。万一 `a_log1` が破損した場合でも、ARC*n* によってそれと同一の `b_log1` をアーカイブできます。このアーカイブ REDO ログには、アーカイブを使用可能にした後に作成された各グループのコピーが含まれます。

データベースが `ARCHIVELOG` モードで稼働しているときは、REDO ログ・グループがアーカイブされないかぎり、ログ・ライター・プロセス（LGWR）は REDO ログ・グループを再利用（上書き）できません。自動アーカイブが使用可能な場合は、バックグラウンド・プロセス ARC*n* によってアーカイブ操作が自動的に実行されます。データベースは必要に応じて複数のアーカイバ・プロセスを起動して、いっぱいになった REDO ログのアーカイブが遅れないようにします。

アーカイブ REDO ログは、次の操作に使用できます。

- データベースのリカバリ
- スタンバイ・データベースの更新
- LogMiner ユーティリティを使用したデータベースの履歴情報の取得

関連項目： アーカイブ REDO ログの使用方法は、次のマニュアルを参照してください。

- 『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』
- スタンバイ・データベースの設定とメンテナンスは、『Oracle Data Guard 概要および管理』を参照してください。
- LogMiner の PL/SQL パッケージの使用方法は、『Oracle Database ユーティリティ』を参照してください。

NOARCHIVELOG モードと ARCHIVELOG モードの選択

ここでは、データベースを NOARCHIVELOG モードまたは ARCHIVELOG モードで稼働する際の考慮点について説明します。この項の内容は、次のとおりです。

- NOARCHIVELOG モードによるデータベースの実行
- ARCHIVELOG モードによるデータベースの実行

いっぱいになった REDO ログ・ファイル・グループをアーカイブ可能にするかどうかは、データベース上で実行されているアプリケーションの可用性と信頼性の要件によって決まります。ディスク障害の発生時にもデータベース内のデータが失われないようにする場合は、ARCHIVELOG モードを使用します。いっぱいになった REDO ログ・ファイルをアーカイブすると、管理作業が増えます。

NOARCHIVELOG モードによるデータベースの実行

データベースを NOARCHIVELOG モードで実行すると、REDO ログはアーカイブされません。データベースの制御ファイルは、グループがいっぱいになってもアーカイブする必要がないことを示します。したがって、ログ・スイッチが発生して、いっぱいになったグループがアクティブでなくなると、そのグループは LGWR で再利用できるようになります。

NOARCHIVELOG モードでは、データベースはインスタンス障害からは保護されますが、メディア障害からは保護されません。オンライン REDO ログ・グループに格納されているデータベースへの最新の変更のみをインスタンスのリカバリに使用できます。データベースが NOARCHIVELOG モードのときにメディア障害が発生した場合、最後にデータベース全体のバックアップを行った時点までのデータベースをリストアできます。そのバックアップ以降のトランザクションはリカバリできません。

NOARCHIVELOG モードでは、オンライン表領域のバックアップを実行できません。さらに、データベースが ARCHIVELOG モードのときに作成したオンライン表領域のバックアップも使用できません。NOARCHIVELOG モードで操作しているデータベースのリストアには、データベースがクローズされているときに作成されたデータベース全体のバックアップのみを使用できます。したがって、NOARCHIVELOG モードでデータベースを操作する場合は、データベース全体のバックアップを短い間隔で定期的に作成してください。

ARCHIVELOG モードによるデータベースの実行

データベースを ARCHIVELOG モードで実行する場合は、REDO ログのアーカイブを使用可能にします。データベースの制御ファイルは、いっぱいになった REDO ログ・ファイルのグループがアーカイブされるまでは、LGWR でこのグループを再使用できないことを示します。いっぱいになったグループは、ログ・スイッチの発生直後からアーカイブに使用できます。

いっぱいになったグループのアーカイブには、次のような利点があります。

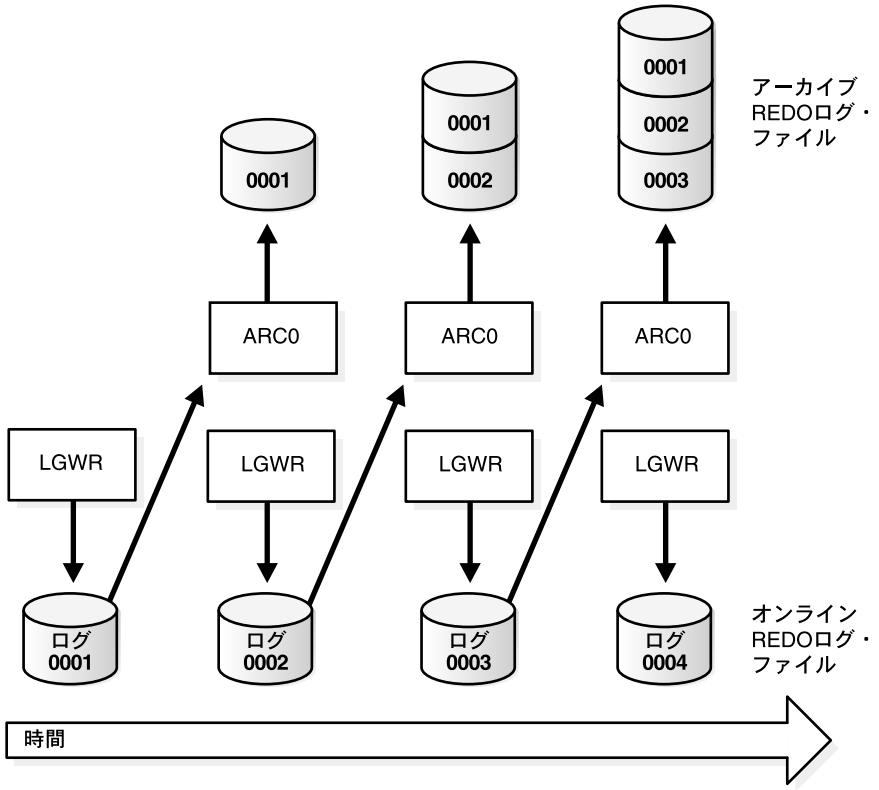
- データベースのバックアップ、オンライン REDO ログおよびアーカイブ REDO ログ・ファイルが揃っていると、オペレーティング・システムやディスクに障害が発生しても、コミットされたすべてのトランザクションをリカバリできることが保証されます。
- アーカイブ・ログを保管していれば、オープンしているデータベースを通常どおり使用している状態で作成したバックアップを使用できます。
- オリジナル・データベースのアーカイブ REDO ログを絶えずスタンバイ・データベースに適用することで、スタンバイをオリジナルとともに最新の状態に保つことができます。

いっぱいになった REDO ログ・ファイルを自動的にアーカイブするようにインスタンスを構成する方法と、手動でアーカイブする方法があります。通常は、自動アーカイブのほうが便利で効率的です。図 11-1 は、アーカイバ・プロセス（この図では ARC0）によって、いっぱいになった REDO ログ・ファイルがデータベースのアーカイブ REDO ログに書き込まれる過程を示しています。

分散データベース内のデータベースをすべて ARCHIVELOG モードで操作している場合は、調整式分散データベース・リカバリを実行できます。ただし、分散データベース内のデータベースのいずれかが NOARCHIVELOG モードで操作されている場合、（すべてのデータベースの整合性

を維持するために）グローバルな分散データベースのリカバリは、NOARCHIVELOG モードで操作しているデータベース全体の最新のバックアップによって制限されます。

図 11-1　ARCHIVELOG モードでの REDO ログ・ファイルの使用



アーカイブの制御

この項では、データベースのアーカイブ・モードを設定する方法と、アーカイブ・プロセスを制御する方法について説明します。この項の内容は、次のとおりです。

- 初期データベース・アーカイブ・モードの設定
- データベース・アーカイブ・モードの変更
- 手動アーカイブの実行
- アーカイバ・プロセス数の調整

> **関連項目：** アーカイブ・モードの制御に関する追加情報は、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。

初期データベース・アーカイブ・モードの設定

初期のアーカイブ・モードは、CREATE DATABASE 文でデータベース作成の一部として設定します。多くの場合、このプロセスで生成される REDO 情報はアーカイブする必要がないため、データベース作成時には NOARCHIVELOG モード（デフォルト）を使用できます。初期のアーカイブ・モードを変更するかどうかは、データベースの作成後に決定します。

ARCHIVELOG モードを指定した場合は、アーカイブ REDO ログ・ファイルのアーカイブ先（11-7 ページの「アーカイブ先の指定」を参照）を初期化パラメータで指定する必要があります。

データベース・アーカイブ・モードの変更

データベース・アーカイブ・モードを変更するには、ALTER DATABASE 文で ARCHIVELOG 句または NOARCHIVELOG 句を使用します。また、データベース・アーカイブ・モードを変更するには、管理者権限（AS SYSDBA）でデータベースに接続する必要があります。

次の手順は、データベース・アーカイブ・モードを NOARCHIVELOG から ARCHIVELOG に切り替えます。

1. データベース・インスタンスを停止します。

   ```
   SHUTDOWN
   ```

   データベースがオープンされている場合は、アーカイブ・モードを切り替える前にクローズし、対応するインスタンスを停止する必要があります。メディア・リカバリを必要とするデータファイルがある場合は、モードを ARCHIVELOG から NOARCHIVELOG に変更できません。

2. データベースのバックアップを作成します。

   データベースに重要な変更をする前に、データベースのデータを保護するため必ずバックアップを作成してください。このバックアップは、NOARCHIVELOG モードでのデータベースの最終バックアップとなり、ARCHIVELOG モードへの切替え中に問題が生じた場合に使用できます。データベース・バックアップの作成については、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

3. 初期化パラメータ・ファイルを編集して、アーカイブ REDO ログ・ファイルのアーカイブ先（11-7 ページの「アーカイブ先の指定」を参照）を初期化パラメータで指定します。

4. 新しいインスタンスを起動し、データベースをマウントします。オープンはしません。

   ```
   STARTUP MOUNT
   ```

   アーカイブを使用可能または使用禁止にするには、データベースをマウントして、オープンしないようにする必要があります。

5. データベース・アーカイブ・モードの変更。通常の操作を実行するためにデータベースをオープンします。

   ```
   ALTER DATABASE ARCHIVELOG;
   ALTER DATABASE OPEN;
   ```

6. データベースを停止します。

   ```
   SHUTDOWN IMMEDIATE
   ```

7. データベースのバックアップを作成します。

   データベースのアーカイブ・モードを変更すると、制御ファイルが更新されます。変更後は、すべてのデータベース・ファイルと制御ファイルのバックアップを作成する必要があります。以前のバックアップは NOARCHIVELOG モードで作成されているため、使用できなくなります。

> **関連項目：** Real Application Clusters 使用時のアーカイブ・モード切替えの詳細は、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。

手動アーカイブの実行

データベースを手動アーカイブ・モードで操作するには、11-5 ページの「データベース・アーカイブ・モードの変更」で説明した手順に従います。ただし、手順 5 で新しいモードを指定する場合は、次の文を使用します。

```
ALTER DATABASE ARCHIVELOG MANUAL;
```

データベースを手動 ARCHIVELOG モードで操作している場合は、いっぱいになった REDO ログ・ファイルの非アクティブ・グループをアーカイブしないと、データベース操作が一時的に停止する可能性があります。いっぱいになった REDO ログ・グループを手動でアーカイブするには、管理者権限を使用して接続してください。データベースがマウントされ、オープンしていないことを確認します。手動アーカイブを実行するには、ALTER SYSTEM 文で ARCHIVE LOG 句を指定します。次の文は、アーカイブされていないログ・ファイルをすべてアーカイブします。

```
ALTER SYSTEM ARCHIVE LOG ALL;
```

手動アーカイブ・モードを使用している場合、アーカイブ先にスタンバイ・データベースは指定できません。

また、自動アーカイブが使用可能な場合でも、いっぱいになった REDO ログ・メンバーの非アクティブ・グループを別の位置に再度アーカイブする場合などに手動アーカイブを使用できます。この場合は、手動アーカイブが完了していなくてもインスタンスでは REDO ログ・グループを再利用できるため、ファイルが上書きされる場合があります。このような場合は、アラート・ログにエラー・メッセージが書き込まれます。

アーカイバ・プロセス数の調整

LOG_ARCHIVE_MAX_PROCESSES 初期化パラメータを使用して、データベースが最初に起動する ARC*n* プロセスの数を指定します。デフォルトのプロセス数は 2 です。通常は、この初期化パラメータを指定したり、デフォルト値を変更する必要はありません。その理由は、データベースでは、必要に応じて追加のアーカイバ・プロセス（ARC*n*）を起動して、いっぱいになった REDO ログの自動処理が遅れないようにするためです。

ただし、追加の ARC*n* プロセスの起動に伴う実行時のオーバーヘッドを回避するには、LOG_ARCHIVE_MAX_PROCESSES 初期化パラメータを設定して、インスタンス起動時に開始する ARC*n* プロセスの数（最大 10）を指定できます。LOG_ARCHIVE_MAX_PROCESSES パラメータは動的で、ALTER SYSTEM 文を使用して変更できます。データベースはマウントされ、オープンしていないことが必要です。次の文は、現在実行されている ARC*n* プロセスの数を増加（または減少）させます。

```
ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESSES=3;
```

アーカイブ先の指定

REDO ログをアーカイブする前に、アーカイブ先を指定し、アーカイブ先の様々な状態を理解する必要があります。11-15 ページの「アーカイブ REDO ログに関する情報の表示」に示す動的パフォーマンス・ビュー（V$）を使用して、必要なすべてのアーカイブ情報を参照できます。

この項の内容は、次のとおりです。

アーカイブ先の指定

REDO ログのアーカイブ先を単一にするか、または**多重化**するかを選択できます。アーカイブ先を単一にする場合は、そのアーカイブ先を LOG_ARCHIVE_DEST 初期化パラメータで指定します。アーカイブ・ログを多重化する場合は、LOG_ARCHIVE_DEST_n パラメータを使用して最大 10 のアーカイブ先を指定するか、または LOG_ARCHIVE_DEST と LOG_ARCHIVE_DUPLEX_DEST パラメータを使用して 1 次アーカイブ先と 2 次アーカイブ先のみを指定できます。次の表に多重化の方法の要約を示します。この後の各項で、詳細を説明します。

方法	初期化パラメータ	ホスト	例
1	LOG_ARCHIVE_DEST_*n* 各項目の意味は次のとおりです。 *n* は 1 ～ 10 の整数	ローカルまたはリモート	LOG_ARCHIVE_DEST_1 = 'LOCATION = /disk1/arc' LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1'
2	LOG_ARCHIVE_DEST および LOG_ARCHIVE_DUPLEX_DEST	ローカルのみ	LOG_ARCHIVE_DEST = '/disk1/arc' LOG_ARCHIVE_DUPLEX_DEST ='/disk2/arc'

関連項目：

- REDO ログのアーカイブ制御に使用される初期化パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- スタンバイ・アーカイブ先の指定に使用する LOG_ARCHIVE_DEST_*n* 初期化パラメータの使用方法は、『Oracle Data Guard 概要および管理』を参照してください。この初期化パラメータには他にも指定できるキーワードがありますが、このマニュアルでは説明されていません。

方法 1: LOG_ARCHIVE_DEST_*n* パラメータの使用

LOG_ARCHIVE_DEST_*n* パラメータ（*n* は 1 ～ 10 の整数）を使用して、1 ～ 10 の異なるアーカイブ先を指定します。末尾に番号が付いた各パラメータによって、特定のアーカイブ先を一意に識別します。

LOG_ARCHIVE_DEST_*n* の位置は、次の表に示すキーワードを使用して指定します。

キーワード	指定内容	例
LOCATION	ローカル・ファイル・システムの位置	LOG_ARCHIVE_DEST_1 = 'LOCATION = /disk1/arc'
SERVICE	Oracle Net のサービス名を介したリモート・アーカイブ	LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1'

LOCATION キーワードを使用する場合は、オペレーティング・システムに有効なパス名を指定します。SERVICE を指定すると、tnsnames.ora ファイルを介してネット・サービス名が接続記述子に変換されます。この記述子には、リモート・データベースへの接続に必要な情報が

含まれています。データベースがスタンバイ・データベースの制御ファイルのログ履歴を正しく更新できるように、サービス名には対応するデータベース SID が必要です。

LOG_ARCHIVE_DEST_*n* 初期化パラメータを使用してアーカイブ REDO ログのアーカイブ先を設定する手順は、次のとおりです。

1. SQL*Plus を使用してデータベースを停止します。

   ```
   SHUTDOWN
   ```

2. LOG_ARCHIVE_DEST_*n* 初期化パラメータを設定し、1 ～ 10 のアーカイブ先を指定します。LOCATION キーワードには、オペレーティング・システム固有のパス名を指定します。たとえば、次のように入力します。

   ```
   LOG_ARCHIVE_DEST_1 = 'LOCATION = /disk1/archive'
   LOG_ARCHIVE_DEST_2 = 'LOCATION = /disk2/archive'
   LOG_ARCHIVE_DEST_3 = 'LOCATION = /disk3/archive'
   ```

   スタンバイ・データベースにアーカイブする場合は、SERVICE キーワードを使用して、tnsnames.ora ファイルに含まれる有効なネット・サービス名を指定します。たとえば、次のように入力します。

   ```
   LOG_ARCHIVE_DEST_4 = 'SERVICE = standby1'
   ```

3. 必要に応じて、LOG_ARCHIVE_FORMAT 初期化パラメータを設定します。ファイル名にスレッド番号を含めるには %t を、ログ順序番号を含めるには %s を、リセットログ ID（ub4 で表されるタイムスタンプ値）を含めるには %r をそれぞれ使用します。番号の左をゼロで埋めるには、大文字（%T、%S および %R）を使用します。

   > **注意：** COMPATIBLE 初期化パラメータが 10.0.0 以上に設定されている場合は、LOG_ARCHIVE_FORMAT パラメータを指定するときに、リセットログ ID（%r）の指定が必要です。このパラメータのデフォルトは、オペレーティング・システムによって異なります。たとえば、UNIX の場合のデフォルト書式は次のとおりです。
   >
   > ```
   > LOG_ARCHIVE_FORMAT=%t_%s_%r.dbf
   > ```
   >
   > データベースのインカネーションは、RESETLOGS オプションを指定してデータベースをオープンすると変更されます。%r を指定すると、アーカイブ REDO ログ・ファイル名からリセットログ ID が取得されます。このリカバリ方法の詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

   次に、LOG_ARCHIVE_FORMAT の設定例を示します。

   ```
   LOG_ARCHIVE_FORMAT = arch_%t_%s_%r.arc
   ```

   この設定では、スレッド 1、ログ順序番号 100、101 および 102、リセットログ ID 509210197 について次のようなアーカイブ・ログが生成されます。リセットログ ID が同一の場合は、すべてのファイルが同じデータベース・インカネーションに含まれることを示します。

   ```
   /disk1/archive/arch_1_100_509210197.arc,
   /disk1/archive/arch_1_101_509210197.arc,
   /disk1/archive/arch_1_102_509210197.arc

   /disk2/archive/arch_1_100_509210197.arc,
   /disk2/archive/arch_1_101_509210197.arc,
   /disk2/archive/arch_1_102_509210197.arc

   /disk3/archive/arch_1_100_509210197.arc,
   /disk3/archive/arch_1_101_509210197.arc,
   /disk3/archive/arch_1_102_509210197.arc
   ```

方法 2: LOG_ARCHIVE_DEST および LOG_ARCHIVE_DUPLEX_DEST の使用

最大 2 つのアーカイブ先ディレクトリを指定するには、LOG_ARCHIVE_DEST パラメータを使用して 1 次アーカイブ先を指定し、必要に応じて LOG_ARCHIVE_DUPLEX_DEST で 2 次アーカイブ先を指定します。アーカイブ先は、ローカルである必要があります。データベースでは、REDO ログはどちらかのパラメータで指定したすべてのアーカイブ先ディレクトリにアーカイブされます。

方法 2 を使用する手順は、次のとおりです。

1. SQL*Plus を使用してデータベースを停止します。

   ```
   SHUTDOWN
   ```

2. LOG_ARCHIVE_DEST および LOG_ARCHIVE_DUPLEX_DEST パラメータにアーカイブ先を指定します。ALTER SYSTEM 文を使用して、LOG_ARCHIVE_DUPLEX_DEST を動的に指定することもできます。たとえば、次のように入力します。

   ```
   LOG_ARCHIVE_DEST = '/disk1/archive'
   LOG_ARCHIVE_DUPLEX_DEST = '/disk2/archive'
   ```

3. 方法 1 の手順 3 で説明したように、LOG_ARCHIVE_FORMAT 初期化パラメータを設定します。

アーカイブ先の状態の理解

各アーカイブ先は次のような可変特性を持っており、これらの特性によってその状態が決まります。

- **Valid/Invalid**: ディスクの位置またはサービス名情報が指定されているかどうか、およびそれらが有効かどうかを示します。
- **Enabled/Disabled**: 位置の使用可能状態と、データベースがアーカイブ先を使用できるかどうかを示します。
- **Active/Inactive**: アーカイブ先へのアクセスに問題があったかどうかを示します。

これらの特性は、何通りかの組合せが可能です。インスタンスの各アーカイブ先について現在の状態などの情報を取得するには、V$ARCHIVE_DEST ビューを問い合せます。

ビューで表示される位置の状態は、表 11-1 に示す特性によって決まります。アーカイブを使用する際は、その特性が Valid、Enabled および Active である必要があります。

表 11-1 アーカイブ先の状態

	特性			
状態	Valid	使用可能	Active	意味
VALID	○	○	○	ユーザーがアーカイブ先を適切に初期化しているため、アーカイブ操作に使用できます。
INACTIVE	×	N/A	N/A	ユーザーがアーカイブ先情報を指定していないか、または削除しました。
ERROR	○	○	×	アーカイブ先ファイルの作成または書込み中にエラーが発生しました。エラー・データを参照してください。
FULL	○	○	×	アーカイブ先がいっぱいです（ディスク領域が残っていません）。
DEFERRED	○	×	○	ユーザーがアーカイブ先を手動で一時的に使用禁止にしています。
DISABLED	○	×	×	ユーザーがエラーの発生後にアーカイブ先を手動で一時的に使用禁止にしています。エラー・データを参照してください。
BAD PARAM	N/A	N/A	N/A	パラメータ・エラーが発生しました。エラー・データを参照してください。

LOG_ARCHIVE_DEST_STATE_*n*（*n* は 1 ～ 10 の整数）初期化パラメータを使用して、指定したアーカイブ先（*n*）の使用可能状態を制御できます。

- ENABLE は、アーカイブ先としてデータベースが使用できることを示します。
- DEFER は、その位置が一時的に使用禁止になっていることを示します。
- ALTERNATE は、代替アーカイブ先を示します。

アーカイブ先の使用可能状態は DEFER です。親アーカイブ先に障害が発生すると ENABLE になります。

ログ転送モードの指定

アーカイブ・ログをアーカイブ先に転送する場合のモードには、**ノーマル・アーカイブ転送**および**スタンバイ転送**という 2 つのモードがあります。ノーマル転送では、ファイルはローカル・ディスクに転送されます。スタンバイ転送では、ファイルはネットワークを介してローカルまたはリモートのスタンバイ・データベースに転送されます。

ノーマル転送モード

ノーマル転送モードでは、アーカイブ先はデータベースの別のディスク・ドライブです。この構成では、アーカイブがインスタンスに必要な他のファイルと競合せず、短時間で完了します。アーカイブ先は、LOG_ARCHIVE_DEST_*n* または LOG_ARCHIVE_DEST パラメータで指定します。

アーカイブ REDO ログ・ファイルとそれに対応するデータベース・バックアップは、ローカル・ディスクからテープなどの永続的で安価なオフライン記憶メディアに移動しておくことをお薦めします。アーカイブ・ログは主としてデータベース・リカバリに使用されるため、プライマリ・データベースに障害が発生した場合でも、これらのログが安全であることを保証する必要があります。

スタンバイ転送モード

スタンバイ転送モードでは、アーカイブ先はローカルまたはリモートのスタンバイ・データベースです。

注意： ローカル・ディスク上でスタンバイ・データベースをメンテナンスすることも可能ですが、スタンバイ・データベースはリモート・サイトでメンテナンスし、最大限の障害対策を講じることをお薦めします。

スタンバイ・データベースを**管理リカバリ・モード**で操作している場合は、転送されたアーカイブ REDO ログを自動的に適用して、スタンバイ・データベースとソース・データベースの同期状態を維持できます。

ファイルをスタンバイ・データベースに正常に転送するには、ARC*n* またはサーバー・プロセスが次の処理を実行する必要があります。

- リモートの位置の認識
- リモート・サーバー上にある**リモート・ファイル・サーバー**（RFS）プロセスの併用によるアーカイブ・ログの転送

各 ARC*n* プロセスには、スタンバイ・アーカイブ先ごとに対応する RFS があります。たとえば、3 つの ARC*n* プロセスを 2 つのスタンバイ・データベースにアーカイブする場合、Oracle Database は 6 つの RFS 接続を確立します。

Oracle Net Services を使用して、ネットワークを介してアーカイブ・ログをリモートの位置に転送します。リモート・アーカイブを指定するには、アーカイブ先の属性としてネット・サービス名を指定します。次に、Oracle Database は、このサービス名を tnsnames.ora ファイルを介して接続記述子に変換します。この記述子には、リモート・データベースへの接続に必要

な情報が含まれています。データベースがスタンバイ・データベースの制御ファイルのログ履歴を正しく更新できるように、サービス名には対応するデータベース SID が必要です。

RFS プロセスは接続先ノード上で実行され、ARC*n* クライアントへのネットワーク・サーバーとして機能します。最終的には、ARC*n* は情報を RFS にプッシュし、RFS がそれをスタンバイ・データベースに転送します。

RFS プロセスは、リモート接続先へのアーカイブ時に必要であり、次のタスクを受け持ちます。

- ARC*n* プロセスからのネットワーク I/O の消費
- STANDBY_ARCHIVE_DEST パラメータを使用した、スタンバイ・データベース上でのファイル名の作成
- リモート・サイトでのログ・ファイルの移入
- スタンバイ・データベースの制御ファイルの更新（Recovery Manager によるリカバリで使用可能にするため）

アーカイブ REDO ログは、オリジナルのデータベースの完全なレプリカであるスタンバイ・データベースをメンテナンスするうえで重要です。データベースは、スタンバイ・アーカイブ・モードで操作できます。このモードでは、スタンバイ・データベースがオリジナル・データベースからのアーカイブ REDO ログで自動的に更新されます。

関連項目：

- 『Oracle Data Guard 概要および管理』
- サービス名を使用してリモート・データベースに接続する方法の詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

アーカイブ先の障害管理

アーカイブ先で発生した障害が、自動アーカイブ・モードで操作している場合のエラー原因となることがあります。Oracle Database には、アーカイブ先の障害に関連する問題を最小限に抑えるためのプロシージャが用意されています。次の各項では、それらのプロシージャについて説明します。

- 正常なアーカイブ先の最小数の指定
- 障害アーカイブ先への再アーカイブ

正常なアーカイブ先の最小数の指定

オプションの初期化パラメータ LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* によって、データベースがオンライン・ログ・ファイルを再利用できるようになるまでに REDO ログ・グループを正常にアーカイブすることが必要なアーカイブ先の最小数が決定します。デフォルト値は 1 です。*n* の有効値は、二重化を使用する場合は 1 ～ 2、多重化を使用する場合は 1 ～ 10 です。

必須およびオプションのアーカイブ先の指定

LOG_ARCHIVE_DEST_*n* パラメータを使用すると、アーカイブ先として OPTIONAL（デフォルト）または MANDATORY を指定できます。LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* パラメータでは、すべての MANDATORY アーカイブ先と、非スタンバイの OPTIONAL アーカイブ先をいくつか使用して、LGWR がオンライン・ログを上書きできるかどうかが判断されます。次の規則が適用されます。

- アーカイブ先として MANDATORY 属性を指定しないと、OPTIONAL が指定されます。
- ローカル・アーカイブ先を少なくとも 1 つは指定する必要があります。この場合は、OPTIONAL または MANDATORY を宣言できます。
- LOG_ARCHIVE_MIN_SUCCEED_DEST の最小値は 1 なので、LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* の値を指定すると、Oracle Database では、少なくとも 1 つのローカル・アーカイブ先が MANDATORY として扱われます。

- MANDATORY のスタンバイ・アーカイブ先も含めて、MANDATORY のアーカイブ先のいずれかで障害が発生すると、Oracle Database では LOG_ARCHIVE_MIN_SUCCEED_DEST パラメータを無視します。
- LOG_ARCHIVE_MIN_SUCCEED_DEST には、アーカイブ先の数を超える値や、MANDATORY のアーカイブ先の数と OPTIONAL のローカル・アーカイブ先の数との合計を超える値は指定できません。
- MANDATORY のアーカイブ先に DEFER を指定した場合で、アーカイブ・ログがスタンバイ・サイトに転送されないままオンライン・ログが上書きされるときは、ログを手動でスタンバイ・サイトに転送する必要があります。

アーカイブ・ログを二重化する場合は、LOG_ARCHIVE_DEST および LOG_ARCHIVE_DUPLEX_DEST パラメータを使用して、アーカイブ先が必須かオプションかを指定できます。次の規則が適用されます。

- LOG_ARCHIVE_DEST によって宣言されたアーカイブ先は必須です。
- LOG_ARCHIVE_DUPLEX_DEST によって宣言されたアーカイブ先は、LOG_ARCHIVE_MIN_SUCCEED_DEST = 1 であればオプション、LOG_ARCHIVE_MIN_SUCCEED_DEST = 2 であれば必須です。

正常なアーカイブ先の数の指定 : 使用例

LOG_ARCHIVE_DEST_*n* および LOG_ARCHIVE_MIN_SUCCEED_DEST パラメータの関係は、使用例を見ると理解しやすくなります。

オプションのローカル・アーカイブ先へのアーカイブ例 この例では、それぞれ OPTIONAL として宣言している 3 つのローカル・アーカイブ先にアーカイブします。表 11-2 に、この場合の LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* に考えられる値を示します。

表 11-2 使用例 1 の LOG_ARCHIVE_MIN_SUCCEED_DEST の値

値	意味
1	データベースは、最低 1 つの OPTIONAL のアーカイブ先へのアーカイブに成功した場合のみログ・ファイルを再利用できます。
2	データベースは、最低 2 つの OPTIONAL のアーカイブ先へのアーカイブに成功した場合のみログ・ファイルを再利用できます。
3	データベースは、OPTIONAL のすべてのアーカイブ先へのアーカイブに成功した場合のみログ・ファイルを再利用できます。
4 以上	エラーです。値がアーカイブ先数を超えています。

この例は、LOG_ARCHIVE_DEST_*n* パラメータを使用してアーカイブ先を明示的に MANDATORY に設定していない場合でも、LOG_ARCHIVE_MIN_SUCCEED_DEST が 1、2 または 3 に設定されていれば、データベースは必ずこれらの位置の 1 つ以上に正常にアーカイブすることを示しています。

必須およびオプションのアーカイブ先へのアーカイブ例 次のような状況を考えてみます。

- MANDATORY のアーカイブ先は 2 つ指定されている。
- OPTIONAL のアーカイブ先は 2 つ指定されている。
- アーカイブ先は、いずれもスタンバイ・データベースではない。

表 11-3 に、LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* に考えられる値を示します。

表 11-3 使用例 2 の LOG_ARCHIVE_MIN_SUCCEED_DEST の値

値	意味
1	データベースは、この値を無視して MANDATORY のアーカイブ先の数（この例では 2）を使用します。
2	データベースは、OPTIONAL のアーカイブ先へのアーカイブに失敗しても、ログ・ファイルを再利用できます。
3	データベースは、最低 1 つの OPTIONAL のアーカイブ先へのアーカイブに成功した場合のみログを再利用できます。
4	データベースは、OPTIONAL のアーカイブ先へのアーカイブに両方とも成功した場合のみログを再利用できます。
5 以上	エラーです。値がアーカイブ先数を超えています。

この例は、アーカイブ先が少なくなるように LOG_ARCHIVE_MIN_SUCCEED_DEST を設定した場合でも、データベースはその設定とは無関係に、MANDATORY として指定されているアーカイブ先に必ずアーカイブすることを示しています。

障害アーカイブ先への再アーカイブ

LOG_ARCHIVE_DEST_*n* パラメータの REOPEN 属性を使用して、エラーの発生後に ARC*n* が障害アーカイブ先への再アーカイブを試行するかどうかと、その時期を指定します。REOPEN は、OPEN エラーのみでなく、すべてのエラーに適用されます。

REOPEN=*n* では、ARC*n* が障害アーカイブ先の再オープンを試行するまでの最小秒数を設定します。*n* のデフォルト値は 300 秒です。値に 0（ゼロ）を指定すると、REOPEN 属性はオフになり、ARC*n* は障害発生後にアーカイブを試行しません。REOPEN キーワードを指定しない場合、ARC*n* はエラー発生後にアーカイブ先を再オープンしません。

REOPEN を使用して、ARC*n* が再接続とアーカイブ・ログ転送を試行する回数を指定することはできません。REOPEN は成功または失敗で終了します。

OPTIONAL アーカイブ先に REOPEN を指定すると、データベースはエラーがある場合にオンライン・ログを上書きできます。MANDATORY のアーカイブ先に REOPEN を指定すると、正常にアーカイブできない場合に本番データベースの機能が停止します。この状況では、次の方法を検討してください。

- 障害アーカイブ先に手動でアーカイブする。
- アーカイブ先を遅延させる、アーカイブ先をオプションとして指定する、サービスを変更するのいずれかの方法によってアーカイブ先を変更する。
- アーカイブ先を削除する。

REOPEN キーワードを使用する場合は、次の点に注意してください。

- ARC*n* は、アーカイブ操作をログ・ファイルの先頭から開始する場合にのみアーカイブ先を再オープンします。現行のアーカイブ操作中に再オープンすることはありません。ARC*n* は、常に先頭からログ・コピーを再試行します。
- 特定の時間またはデフォルト設定で REOPEN を指定した場合、ARC*n* は記録されたエラー発生時刻から REOPEN 間隔が経過した時刻が現在時刻より前かどうかをチェックします。現在時刻より前であれば、ARC*n* はログ・コピーを再試行します。
- REOPEN 句は、ACTIVE=TRUE のアーカイブ先状態に影響を及ぼします。VALID および ENABLED 状態は変化しません。

ARCHIVELOG プロセスによって生成されるトレース出力の制御

バックグラウンド・プロセスは適宜、トレース・ファイルに情報を書き込みます。（詳細は、7-2 ページの「トレース・ファイルおよびアラート・ログを使用したエラーの監視」を参照してください。）ARCHIVELOG プロセスの場合は、トレース・ファイルに書き込む出力を制御できます。制御するには、LOG_ARCHIVE_TRACE 初期化パラメータを設定して**トレース・レベル**を指定します。設定可能な値は、次のとおりです。

トレース・レベル	意味
0	アーカイブ・ログのトレースは出力されません。これはデフォルトです。
1	REDO ログ・ファイルのアーカイブを追跡します。
2	アーカイブ・ログのアーカイブ先ごとにアーカイブ状態を追跡します。
4	アーカイブ操作のフェーズを追跡します。
8	アーカイブ・ログのアーカイブ先のアクティビティを追跡します。
16	アーカイブ・ログのアーカイブ先の詳細アクティビティを追跡します。
32	アーカイブ・ログのアーカイブ先パラメータの変更を追跡します。
64	ARC*n* プロセスの状態のアクティビティを追跡します。
128	FAL（フェッチ・アーカイブ・ログ）サーバー関連のアクティビティを追跡します。
256	将来のリリースでサポートされる予定です。
512	非同期の LGWR アクティビティを追跡します。
1024	RFS 物理クライアントを追跡します。
2048	ARC*n*/RFS ハートビートを追跡します。
4096	リアルタイム適用を追跡します。
8192	REDO 適用アクティビティ（メディア・リカバリまたはフィジカル・スタンバイ）を追跡します。

パラメータ値として、必要な各トレース・レベルの合計を設定することにより、トレース・レベルを組み合せることができます。たとえば、LOG_ARCHIVE_TRACE=12 に設定すると、トレース・レベル 8 および 4 の出力が生成されます。また、プライマリ・データベースとスタンバイ・データベースには、異なる値を設定できます。

LOG_ARCHIVE_TRACE パラメータのデフォルト値は 0（ゼロ）です。このレベルでは、エラー条件が発生すると、ARCHIVELOG プロセスによって適切なアラートおよびトレース・エントリが生成されます。

このパラメータの値は、ALTER SYSTEM 文で動的に変更できます。データベースはマウントされ、オープンしていないことが必要です。次に例を示します。

```
ALTER SYSTEM SET LOG_ARCHIVE_TRACE=12;
```

この方法で行った変更は、次回のアーカイブ操作の開始時に有効になります。

> **関連項目：** このパラメータをスタンバイ・データベースで使用する方法については、『Oracle Data Guard 概要および管理』を参照してください。

アーカイブ REDO ログに関する情報の表示

アーカイブ REDO ログに関する情報を表示するには、動的パフォーマンス・ビューまたは ARCHIVE LOG LIST コマンドを使用します。

この項の内容は、次のとおりです。

アーカイブ REDO ログ・ビュー

アーカイブ REDO ログに関して役立つ情報を含む動的パフォーマンス・ビューがいくつかあります。次の表に要約を示します。

動的パフォーマンス・ビュー	説明
V$DATABASE	データベースが ARCHIVELOG モードと NOARCHIVELOG モードのいずれであるか、および MANUAL（アーカイブ・モード）が指定されているかどうかが表示されます。
V$ARCHIVED_LOG	制御ファイルに格納されたアーカイブ・ログ履歴情報が表示されます。リカバリ・カタログを使用している場合は、RC_ARCHIVED_LOG ビューにも同様の情報が含まれます。
V$ARCHIVE_DEST	現行インスタンス、すべてのアーカイブ先、各アーカイブ先の現行の値、モードおよび状態が表示されます。
V$ARCHIVE_PROCESSES	インスタンスの各アーカイブ・プロセスの状態情報が表示されます。
V$BACKUP_REDOLOG	アーカイブ・ログのバックアップ情報が含まれます。リカバリ・カタログを使用している場合は、RC_BACKUP_REDOLOG ビューにも同様の情報が含まれます。
V$LOG	データベースの REDO ログ・グループをすべて表示し、その中でアーカイブする必要があるグループを示します。
V$LOG_HISTORY	アーカイブ済ログや各アーカイブ・ログの SCN 範囲などのログ履歴情報が含まれます。

たとえば、次の問合せでは、アーカイブする必要がある REDO ログ・グループが表示されます。

```
SELECT GROUP#, ARCHIVED
   FROM SYS.V$LOG;

GROUP#     ARC
--------   ---
       1   YES
       2   NO
```

現行のアーカイブ・モードを確認するには、V$DATABASE ビューを問い合せます。

```
SELECT LOG_MODE FROM SYS.V$DATABASE;

LOG_MODE
------------
NOARCHIVELOG
```

関連項目： 動的パフォーマンス・ビューの詳細は、『Oracle Database リファレンス』を参照してください。

ARCHIVE LOG LIST コマンド

SQL*Plus コマンドの ARCHIVE LOG LIST を使用して、接続されているインスタンスのアーカイブ情報を表示します。次に例を示します。

```
SQL> ARCHIVE LOG LIST

Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            D:¥oracle¥oradata¥IDDB2¥archive
Oldest online log sequence     11160
Next log sequence to archive   11163
Current log sequence           11163
```

この表示は、現行インスタンスのアーカイブ REDO ログの設定に関して必要なすべての情報を示します。

- このデータベースは現在 ARCHIVELOG モードで操作されています。
- 自動アーカイブは使用可能です。
- アーカイブ REDO ログのアーカイブ先は、D:¥oracle¥oradata¥IDDB2¥archive です。
- いっぱいになった REDO ログ・グループのうち、最も古いものの順序番号は 11160 です。
- いっぱいになった REDO ログ・グループのうち、次にアーカイブされるものの順序番号は 11163 です。
- 現行の REDO ログ・ファイルの順序番号は 11163 です。

> **関連項目：** ARCHIVE LOG LIST コマンドの詳細は、『SQL*Plus ユーザーズ・ガイドおよびリファレンス』を参照してください。

12

表領域の管理

この章の内容は次のとおりです。

関連項目：

- データベース構造、領域管理、表領域およびデータファイルの詳細は、『Oracle Database 概要』を参照してください。
- Oracle Database サーバーによって作成および管理されるデータファイルと一時ファイルの作成方法は、第 15 章「Oracle Managed Files の使用」を参照してください。

表領域を管理するためのガイドライン

Oracle Database の表領域を使用して作業する前に、次の各項で説明するガイドラインについて理解してください。

複数の表領域の使用

データベース操作を実行する際に複数の表領域を使用すると、システムの柔軟性が向上します。データベースに複数の表領域があるときには、次のことが可能です。

- ユーザー・データをデータ・ディクショナリ・データから分離し、I/O の競合を減らす。
- あるアプリケーションのデータを別のアプリケーションのデータから分離し、表領域をオフライン化する必要が生じた場合に、複数のアプリケーションが影響を受けないようにする。
- I/O の競合を低減するために、異なるディスク・ドライブ上に異なる表領域のデータファイルを配置する。
- 別の表領域をオンライン状態に維持しながら、個々の表領域をオフライン化して、全体の可用性を高める。
- 高い更新アクティビティ、読取り専用アクティビティ、一時セグメント記憶域など、異なるタイプのデータベース利用のために異なる表領域を確保することによって、表領域利用を最適化する。
- 表領域のバックアップを個別に作成する。

一部のオペレーティング・システムでは、同時にオープン可能なファイルの数に制限が設けられています。このような制限によって、同時にオンライン化可能な表領域の数に影響が出ることがあります。そのため、使用しているオペレーティング・システムの制限を超えないように、表領域を効率よく計画する必要があります。表領域はデータベースの要件を満たすために必要な数だけ作成し、構成するファイル数もできるかぎり少なくなるようにしてください。表領域のサイズを大きくする必要がある場合は、小さいデータファイルを多数作成するのではなく、1 つまたは 2 つの大きなデータファイルを追加するか、または自動拡張を使用可能にしてデータファイルを作成します。

これらの要素を考慮に入れてデータを再検討し、データベース設計に必要となる表領域の数を決定してください。

ユーザーに対する表領域割当て制限の割当て

表、クラスタ、マテリアライズド・ビュー、索引およびその他のオブジェクトを作成しようとするユーザーには、そのオブジェクトを作成するための権限と、そのオブジェクトのセグメントを格納する表領域の**割当て制限**（領域の許容または制限）を付与します。

> **関連項目：** ユーザーの作成方法、および表領域割当て制限の割当て方法については、『Oracle Database セキュリティ・ガイド』を参照してください。

表領域の作成

表領域を作成する前に、それを格納するデータベースを作成する必要があります。どのデータベースでも、重要な表領域は SYSTEM 表領域です。この表領域には、データ・ディクショナリやシステム・ロールバック・セグメントなど、データベース・サーバー機能の基本となる情報が格納されます。SYSTEM 表領域は、データベース作成時に最初に作成される表領域です。他の表領域と同様に管理されますが、より高いレベルの権限が必要で、一部の機能は制限されます。たとえば、SYSTEM 表領域の名前変更、削除、オフライン化は実行できません。

SYSAUX 表領域は、SYSTEM 表領域の補助表領域として機能し、データベース作成時に常に作成されます。この補助表領域には、様々な Oracle 製品および機能で使用される情報とスキーマが格納されるため、各製品で独自の表領域を持つ必要がなくなります。SYSTEM 表領域と同様に、SYSAUX 表領域を管理するためには他の表領域より高いレベルのセキュリティが必要で、この表領域の名前変更や削除はできません。SYSAUX 表領域の管理については、12-25 ページの「SYSAUX 表領域の管理」を参照してください。

表領域を作成する手順はオペレーティング・システムによって異なりますが、必ず最初に、オペレーティング・システムを使用して、データファイルが割り当てられるディレクトリ構造を作成する必要があります。ほとんどのオペレーティング・システムでは、新しい表領域を作成するとき、またはデータファイルを加えて既存の表領域を変更するときに、データファイルのサイズと完全なファイル名を指定します。新しい表領域を作成するとき、または既存の表領域を変更するときは、いずれの場合も、データベースは、指定されたとおりにデータファイルを自動的に割り当てて、フォーマットします。

新しい表領域を作成するには、SQL 文 CREATE TABLESPACE または CREATE TEMPORARY TABLESPACE を使用します。表領域を作成するには、CREATE TABLESPACE システム権限が必要です。後で、ALTER TABLESPACE または ALTER DATABASE 文を使用して、この表領域を変更できます。そのためには、ALTER TABLESPACE または ALTER DATABASE システム権限が必要です。

また、CREATE UNDO TABLESPACE 文を使用して、**UNDO 表領域**と呼ばれる特別なタイプの表領域も作成できます。この表領域は、UNDO レコードを格納するために特別に設計されています。UNDO レコードとはデータベースが生成するレコードで、リカバリや読込み一貫性のために、または ROLLBACK 文の要求によって、データベースの変更をロールバックまたは取り消す際に使用されます。UNDO 表領域の作成と管理については、第 14 章「UNDO の管理」を参照してください。

永続表領域および一時表領域の作成とメンテナンスについては、次の各項で説明します。

- ローカル管理表領域
- 大型ファイル表領域
- 暗号化された表領域
- 一時表領域
- 複数の一時表領域 : 表領域グループの使用

関連項目：

- データベース作成時に作成される表領域については、第 2 章「Oracle Database の作成および構成」およびオペレーティング・システム固有の Oracle Database インストレーション・ガイドを参照してください。
- CREATE TABLESPACE、CREATE TEMPORARY TABLESPACE、ALTER TABLESPACE および ALTER DATABASE 文の構文とセマンティクスの詳細は、『Oracle Database SQL リファレンス』を参照してください。
- 非標準のブロック・サイズを持つ表領域の作成に必要な初期化パラメータの詳細は、2-28 ページの「データベース・ブロック・サイズの指定」を参照してください。

ローカル管理表領域

ローカル管理表領域では、その表領域内のすべてのエクステント情報がビットマップを使用して追跡されるため、次のような利点があります。

- 領域操作が高速かつ同時に実行されます。領域の割当てと割当て解除によって、ローカル管理のリソース（ヘッダー・ファイルに格納されているビットマップ）が変更されます。
- パフォーマンスが向上します。
- ローカル管理の一時表領域では UNDO や REDO が生成されないため、読取り可能なスタンバイ・データベースを使用できます。
- AUTOALLOCATE 句を指定すると、データベースで適切なエクステント・サイズが自動的に選択されるため、領域割当てが簡素化されます。
- 必要な情報はファイル・ヘッダーとビットマップ・ブロックに格納されるため、データ・ディクショナリに対するユーザーの依存性が低下します。
- ローカル管理表領域では、使用可能エクステントを結合する必要はありません。

SYSTEM 表領域も含めて、すべての表領域をローカルに管理できます。

DBMS_SPACE_ADMIN パッケージによって、ローカル管理表領域のメンテナンス手順が提供されます。

関連項目：

- 2-17 ページ「ローカル管理の SYSTEM 表領域の作成」、12-29 ページ「ローカル管理表領域への SYSTEM 表領域の移行」および 12-26 ページ「ローカル管理表領域の問題の診断と修復」
- 単一データファイルまたは一時ファイルのみ格納される、別のタイプのローカル管理表領域の作成方法については、12-6 ページの「大型ファイル表領域」を参照してください。
- DBMS_SPACE_ADMIN パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ローカル管理表領域の作成

ローカル管理表領域を作成するには、CREATE TABLESPACE 文の EXTENT MANAGEMENT 句に LOCAL を指定します。これは新しい永続表領域に対するデフォルトですが、AUTOALLOCATE 句または UNIFORM 句を指定する場合は、EXTENT MANAGEMENT LOCAL 句を指定する必要があります。AUTOALLOCATE 句（デフォルト）を指定してデータベースでエクステントを自動的に管理するか、または表領域を特定サイズ（UNIFORM）の均一エクステントで管理するかを指定できます。

様々なサイズのオブジェクトが表領域に含まれ、異なるエクステント・サイズの多数のエクステントが必要と予測される場合は、AUTOALLOCATE を選択してください。領域の割当てと割当て解除を厳密に制御しなくてもよい場合は、AUTOALLOCATE を選択すると表領域の管理作業が簡素化されます。この設定では、ある程度の領域が無駄になるという短所もありますが、ほとんどの場合、Oracle Database によって領域が管理されるという利点のほうが重要です。

未使用領域の厳密な制御が必要で、オブジェクトに割り当てられる領域、エクステントの数とサイズを正確に予測できる場合は、UNIFORM を選択してください。この設定によって、表領域から使用できない領域がなくなります。

エクステント管理のタイプを明示的に指定しない場合は、Oracle Database によってエクステント管理が次のように判断されます。

- CREATE TABLESPACE 文に DEFAULT 記憶域句を指定しない場合、データベースでは自動割当てのローカル管理表領域が作成されます。

- CREATE TABLESPACE 文に DEFAULT 記憶域句を指定した場合、データベースでは次のことが考慮されます。
  - MINIMUM EXTENT 句を指定した場合は、MINIMUM EXTENT、INITIAL および NEXT の値が等しいかどうかと、PCTINCREASE の値が 0（ゼロ）かどうかが評価されます。3 つの値が等しく、PCTINCREASE の値が 0（ゼロ）の場合は、エクステント・サイズ = INITIAL で均一なローカル管理表領域が作成されます。MINIMUM EXTENT、INITIAL および NEXT パラメータの値が等しくない場合、または PCTINCREASE が 0（ゼロ）でない場合は、指定したエクステント記憶域パラメータが無視され、自動割当てのローカル管理表領域が作成されます。
  - MINIMUM EXTENT 句を指定しない場合は、INITIAL および NEXT の記憶域パラメータの値が等しいかどうかと、PCTINCREASE が 0（ゼロ）かどうかのみが評価されます。2 つの値が等しく、PCTINCREASE が 0（ゼロ）の場合は、均一なローカル管理表領域が作成されます。それ以外の場合は、ローカル管理の表領域が作成され、自動的に割り当てられます。

次の文は、ローカル管理表領域 lmtbsb を作成し、AUTOALLOCATE を指定しています。

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

AUTOALLOCATE を指定すると、表領域はシステム管理になり、最小エクステント・サイズは 64KB となります。

AUTOALLOCATE のかわりに UNIFORM を指定すると、表領域は均一サイズのエクステントで管理されます。このサイズは、UNIFORM の SIZE 句で指定できます。SIZE を指定しないと、デフォルト・サイズは 1MB になります。

次の例では、128KB の均一エクステントの表領域を作成しています（ブロック・サイズが 2KB のデータベースでは、各エクステントは 64 個のデータベース・ブロックに相当します）。128KB の各エクステントは、このファイルのエクステント・ビットマップでは 1 ビットで表されます。

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K;
```

EXTENT MANAGEMENT LOCAL を明示的に指定する場合は、DEFAULT の記憶域句、MINIMUM EXTENT または TEMPORARY を指定できません。ローカル管理の一時表領域を作成するには、CREATE TEMPORARY TABLESPACE 文を使用します。

注意： ローカル管理表領域にデータファイルを割り当てる場合は、領域管理に使用されるメタデータ（エクステント・ビットマップまたは領域ヘッダー・セグメント）用の領域を考慮する必要があります。これはユーザー領域の一部です。たとえば、エクステント管理句に UNIFORM を指定するときに、SIZE パラメータを指定しないと、デフォルトのエクステント・サイズは 1MB になります。この場合、データファイルには 1MB より大きいサイズ（少なくとも 1 ブロックとビットマップ用の領域分は大きいサイズ）を指定する必要があります。

ローカル管理表領域のセグメント領域管理の指定

ローカル管理表領域では、Oracle Database でセグメント領域の管理に使用できる方法は、自動と手動の 2 つです。手動セグメント領域管理では、「空きリスト」と呼ばれるリンクされたリストを使用してセグメント内の空き領域を管理します。一方、自動セグメント領域管理では、ビットマップを使用します。自動セグメント領域管理の方がより効率的な方法で、新しい永続ローカル管理表領域すべてのデフォルトです。

自動セグメント領域管理では、手動セグメント領域管理に比べて領域の使用効率が向上します。また、ユーザー数やインスタンス数の増加につれて拡張されるという点で自己チューニング型です。Oracle Real Application Clusters 環境の場合、自動セグメント領域管理ではインスタンスに対する領域の動的アフィニティに対処できます。さらに、多くの標準的な処理負荷の場合、

自動セグメント領域管理を使用したアプリケーションのパフォーマンスは、手動セグメント領域管理を使用して適切にチューニングされたアプリケーションよりも向上します。

自動セグメント領域管理は新しい永続ローカル管理表領域のデフォルトですが、SEGMENT SPACE MANAGEMENT AUTO 句を使用して明示的に使用可能にできます。

次の文は、自動セグメント領域管理を行う lmtbsb 表領域を作成します。

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL
    SEGMENT SPACE MANAGEMENT AUTO;
```

SEGMENT SPACE MANAGEMENT MANUAL 句を指定すると、自動セグメント領域管理を使用禁止にできます。

表領域の作成時に指定するセグメント領域管理方法は、その後表領域に作成されたすべてのセグメントに対して適用されます。表領域のセグメント領域管理モードは変更できません。

注意：

- エクステント管理を LOCAL UNIFORM に設定した場合は、各エクステントに最低 5 個のデータベース・ブロックがあることを確認してください。
- エクステント管理を LOCAL AUTOALLOCATE に設定し、データベース・ブロックのサイズが 16KB 以上の場合、Oracle は、5 個のブロックの最小サイズを 64KB に切り上げた複数のエクステントを作成してセグメント領域を管理します。

自動セグメント領域管理を使用するローカル管理表領域は、単一ファイル表領域、つまり、大型ファイル表領域（12-6 ページの「大型ファイル表領域」を参照）としても作成できます。

大型ファイル表領域

大型ファイル表領域は、単一で非常に大きい（最大 40 億ブロック）データファイルを持つ表領域です。これに対して、従来の小型ファイル表領域には複数のデータファイルを格納できますが、各データファイルは大きくありません。大型ファイル表領域の利点は、次のとおりです。

- 8000 ブロックを持つ大型ファイル表領域には、32TB のデータファイルを格納できます。32000 ブロックを持つ大型ファイル表領域には、128TB のデータファイルを格納できます。Oracle Database では、データファイルの最大数が制限されています（通常は 64000 ファイル）。したがって、大型ファイル表領域によって、Oracle Database の記憶域容量が大幅に増加します。
- 大型ファイル表領域を使用すると、データベースに必要なデータファイルの数を減らすことができます。また、別の利点として、CREATE DATABASE 文と CREATE CONTROLFILE 文の DB_FILES 初期化パラメータと MAXDATAFILES パラメータを調整すると、データファイル情報に必要な SGA 領域の量と制御ファイルのサイズを削減できます。
- 大型ファイル表領域によるデータファイルの透過性によって、データベース管理が簡素化されます。ALTER TABLESPACE 文の SQL 構文を使用すると、基礎となる各データファイルではなく表領域で操作を実行できます。

大型ファイル表領域は、自動セグメント領域管理を指定したローカル管理表領域でのみサポートされます。ただし、ローカル管理の UNDO 表領域、一時表領域および SYSTEM 表領域の 3 つは例外です。

注意：

- 大型ファイル表領域は、ストライプ化や RAID、および動的に拡張可能な論理ボリュームをサポートする、自動ストレージ管理（ASM）などの論理ボリューム・マネージャとともに使用することを目的にしています。
- パラレル問合せ実行および Recovery Manager のバックアップ・パラレル化で問題が生じる可能性があるため、ストライプ化をサポートしていないシステムには大型ファイル表領域を作成しないでください。
- 大規模なファイル・サイズをサポートしていないプラットフォームで大型ファイル表領域を使用することはお薦めしません。表領域の容量が制限される場合があります。サポートされている最大ファイル・サイズについては、使用しているオペレーティング・システム固有のマニュアルを参照してください。

大型ファイル表領域の作成

大型ファイル表領域を作成するには、CREATE TABLESPACE 文の BIGFILE キーワード（CREATE BIGFILE TABLESPACE ...）を指定します。Oracle Database は、自動セグメント領域管理を指定したローカル管理表領域を自動的に作成します。この文には、EXTENT MANAGEMENT LOCAL および SEGMENT SPACE MANAGEMENT AUTO を必要に応じて指定できます。ただし、EXTENT MANAGEMENT DICTIONARY または SEGMENT SPACE MANAGEMENT MANUAL を指定すると、データベースはエラーを返します。この文の残りの構文は CREATE TABLESPACE 文と同じですが、指定できるのは 1 つのデータファイルのみです。次に例を示します。

```
CREATE BIGFILE TABLESPACE bigtbs
    DATAFILE '/u02/oracle/data/bigtbs01.dbf' SIZE 50G
...
```

SIZE は、キロバイト（KB）、メガバイト（MB）、ギガバイト（GB）またはテラバイト（TB）で指定できます。

データベース作成時にデフォルトの表領域タイプを BIGFILE に設定している場合は、CREATE TABLESPACE 文に BIGFILE キーワードを指定する必要はありません。この場合、大型ファイル表領域はデフォルトで作成されます。

データベース作成時にデフォルトの表領域タイプを BIGFILE に設定しておきながら、従来の表領域（小型ファイル）を作成する場合は、CREATE SMALLFILE TABLESPACE 文を指定すると、デフォルトの表領域タイプよりも作成する表領域が優先されます。

関連項目： 2-21 ページ「データベース作成時の大型ファイル表領域のサポート」

大型ファイル表領域の識別

次のビューには、表領域を大型ファイル表領域として識別する BIGFILE 列が含まれています。

- DBA_TABLESPACES
- USER_TABLESPACES
- V$TABLESPACE

大型ファイル表領域は、その単一データファイルの相対ファイル番号によっても識別できます。その番号は、ほとんどのプラットフォームで 1024 ですが、OS/390 では 4096 です。

暗号化された表領域

永続表領域を暗号化して機密データを保護できます。表領域の暗号化はアプリケーションに対して完全に透過的であるため、アプリケーションの変更は不要です。暗号化された表領域は、主に、データベースを介さない手段による不当なアクセスからデータを保護します。たとえば、暗号化された表領域を、Oracle データベース間で移動したり、保管のために外部の設備に移動するなどの目的で、バックアップ・メディアに書き込む場合、それらの表領域は暗号化されたままです。また、暗号化された表領域は、ユーザーがデータベースのセキュリティ機能を迂回してオペレーティング・システムのファイル・システムから直接データベース・ファイルにアクセスしようとした場合に、そのユーザーからデータを保護します。

表領域の暗号化によって、すべてのセキュリティ問題に対処できるわけではありません。たとえば、データベース内からのアクセスは制御できません。暗号化された表領域に格納されているオブジェクトに対する権限が付与されたユーザーは、追加のパスワードやキーを指定せずにそれらのオブジェクトにアクセスできます。

表領域を暗号化すると、すべての表領域ブロックが暗号化されます。暗号化は、表、クラスタ、索引、LOB（BASICFILE と SECUREFILE）、表パーティション、索引パーティションなどを含むすべてのセグメント・タイプに対してサポートされています。

注意： 暗号化された表領域に格納されている SECUREFILE LOB に対して LOB 暗号化を使用する必要はありません。

最大限のセキュリティを確保するために、暗号化された表領域のデータは、UNDO 表領域、REDO ログおよび一時表領域に書き込まれる場合は自動的に暗号化されます。暗号化された UNDO 表領域や一時表領域を明示的に作成する必要はありません。実際、それらの表領域タイプに暗号化を指定することはできません。

異なる表領域に別のパーティションがあるパーティション表およびパーティション索引表の場合は、同じ表または索引で、暗号化された表領域と暗号化されていない表領域の両方を使用できます。

表領域の暗号化では Oracle Database の透過的データ暗号化が使用されます。この機能を使用するには、データベースのマスター暗号化キーを格納するための Oracle ウォレットを作成する必要があります。暗号化された表領域を作成する場合、および暗号化データを格納または取得する場合は、ウォレットがオープンしている必要があります。ウォレットは、オープンするとすべてのセッションで使用可能になり、明示的にクローズするか、データベースが停止されるまではオープンしたままになります。

表領域を暗号化するには、COMPATIBLE 初期化パラメータを 11.1.0 以上に設定した状態でデータベースをオープンする必要があります。Oracle Database11*g* リリース 1 の新規インストールでは、COMPATIBLE のデフォルト設定は 11.1.0 です。表領域を作成できるユーザーは、暗号化された表領域も作成できます。

透過的データ暗号化では、次に示す Advanced Encryption Standard（AES）アルゴリズムや Triple Data Encryption Standard（3DES）アルゴリズムなど、業界標準の暗号化アルゴリズムがサポートされています。

- 3DES168
- AES128
- AES192
- AES256

暗号化キーの長さはアルゴリズム名で示されています。たとえば、AES128 アルゴリズムでは 128 ビットのキーが使用されます。使用するアルゴリズムは表領域の作成時に指定します。表領域ごとに異なるアルゴリズムを使用できます。理論上は、キーの長さが長くなるほどセキュリティが強化されますが、その分、CPU オーバーヘッドがかかります。CREATE TABLESPACE 文でアルゴリズムを指定しない場合は、AES128 がデフォルトとなります。表領域の暗号化では、ディスク領域のオーバーヘッドは発生しません。

例

次の文では、デフォルトの暗号化アルゴリズムを使用して暗号化された表領域を作成しています。

```
CREATE TABLESPACE securespace
DATAFILE '/u01/app/oracle/oradata/orcl/secure01.dbf' SIZE 100M
ENCRYPTION
DEFAULT STORAGE(ENCRYPT);
```

次の文では、AES256 アルゴリズムを使用して同じ表領域を作成しています。

```
CREATE TABLESPACE securespace
DATAFILE '/u01/app/oracle/oradata/orcl/secure01.dbf' SIZE 100M
ENCRYPTION USING 'AES256'
DEFAULT STORAGE(ENCRYPT);
```

制限事項

暗号化された表領域の制限事項は、次のとおりです。

- 既存の表領域は、ALTER TABLESPACE 文で暗号化できません。ただし、データ・ポンプまたは SQL 文（CREATE TABLE AS SELECT や ALTER TABLE MOVE など）を使用して、既存の表データを暗号化された表領域に移動できます。
- 暗号化された表領域は、別のデータベースにトランスポートする際に制限を受けます。12-32 ページの「トランスポータブル表領域の使用に関する制限事項」を参照してください。
- 暗号化された表領域が含まれたデータベースをリカバリする場合（SHUTDOWN ABORT の後、またはデータベース・インスタンスが停止する致命的なエラーの後など）は、リカバリ・プロセスがデータ・ブロックと REDO を復号化できるように、データベースのマウントの後に Oracle ウォレットをオープンしてからデータベースをオープンする必要があります。

また、透過的データ暗号化に関する一般的な制限事項については、『Oracle Database Advanced Security 管理者ガイド』を参照してください。

表領域の暗号化情報の問合せ

DBA_TABLESPACES と USER_TABLESPACES のデータ・ディクショナリ・ビューには、ENCRYPTED という列が含まれています。表領域が暗号化されている場合は、この列に YES と表示されます。

V$ENCRYPTED_TABLESPACES ビューには、現在暗号化されているすべての表領域がリストされます。次の問合せでは、暗号化された表領域の名前と暗号化アルゴリズムが表示されます。

```
SELECT t.name, e.encryptionalg algorithm
FROM  v$tablespace t, v$encrypted_tablespaces e
WHERE t.ts# = e.ts#;

NAME                           ALGORITHM
------------------------------ ---------
SECURESPACE                    AES128
```

関連項目：

- 透過的データ暗号化の詳細、およびウォレットの作成とオープンの手順については、『Oracle Database 2 日でセキュリティ・ガイド』を参照してください。
- 表領域全体を暗号化する代替方法については、18-7 ページの「機密データを格納する列の暗号化」を参照してください。
- Oracle Real Application Clusters 環境で Oracle ウォレットを使用する方法については、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。
- CREATE TABLESPACE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

一時表領域

一時表領域には、セッションの間のみ存続する一時データが格納されます。一時表領域を使用すると、メモリーに格納できない複数のソート操作の同時実行性を改善し、ソート時の領域管理操作の効率を改善できます。

一時表領域は、次の情報を格納するために使用します。

- 中間ソート結果
- 一時表と一時索引
- 一時 LOB
- 一時 B ツリー

特定のインスタンスのソート操作はすべて、一時表領域内の 1 つのソート・セグメントを共有し、ソート・セグメントは、一時領域を必要とするソート操作を実行するすべてのインスタンスに存在します。ソート・セグメントは、一時表領域を使用してソート処理を実行する最初の文によってインスタンスの起動後に作成され、停止時にのみ解放されます。

デフォルトでは、新規に Oracle Database をインストールするたびに、TEMP という単一の一時表領域が作成されます。追加の一時表領域は、CREATE TABLESPACE 文で作成できます。一時表領域を各データベース・ユーザーに割り当てるには、CREATE USER 文または ALTER USER 文を使用します。複数のユーザーが単一の一時表領域を共有できます。

一時表領域には、オブジェクトを明示的に作成できません。

注意： 前述の説明の例外は、一時表です。一時表を作成すると、その行はデフォルト一時表領域に格納されます。ただし、一時表を新規の一時表領域内に作成する場合を除きます。詳細は、18-10 ページの「一時表の作成」を参照してください。

デフォルト一時表領域

一時表領域が明示的に割り当てられていないユーザーは、データベースのデフォルト一時表領域（新規インストールでは TEMP）を使用します。データベースのデフォルト一時表領域は次のコマンドで変更できます。

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE tablespace_name;
```

データベースの現行のデフォルト一時表領域を判断するには、次の問合せを実行します。

```
SELECT PROPERTY_NAME, PROPERTY_VALUE FROM DATABASE_PROPERTIES WHERE
   PROPERTY_NAME='DEFAULT_TEMP_TABLESPACE';

PROPERTY_NAME              PROPERTY_VALUE
-------------------------- ------------------------------
DEFAULT_TEMP_TABLESPACE    TEMP
```

一時表領域の領域割当て

一時表領域のソート・セグメントの領域割当てと割当て解除は、V$SORT_SEGMENT ビューを使用して表示できます。V$SORT_USAGE ビューでは、そのセグメント内の現行のソート・ユーザーが識別されます。

一時表領域を使用するソート操作が完了しても、ソート・セグメントに割り当てられたエクステントの割当ては解除されず、エクステントには、使用可能マークおよび再利用可能マークが付けられます。DBA_TEMP_FREE_SPACE ビューには、各一時表領域の割当済領域の合計と空き領域が表示されます。詳細は、12-12 ページの「一時表領域の領域使用情報の表示」を参照してください。大量の未使用領域があるローカル管理の一時表領域は手動で縮小できます。詳細は、12-23 ページの「ローカル管理の一時表領域の縮小」を参照してください。

関連項目：

- ユーザーの作成方法、および一時表領域の割当て方法については、『Oracle Database セキュリティ・ガイド』を参照してください。
- デフォルト一時表領域の詳細は、『Oracle Database 概要』を参照してください。
- V$SORT_SEGMENT、V$SORT_USAGE および DBA_TEMP_FREE_SPACE の各ビューの詳細は、『Oracle Database リファレンス』を参照してください。
- ソートのチューニングの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

ローカル管理の一時表領域の作成

ローカル管理表領域では、領域の管理がより簡単で効率的であるため、一時表領域には理想的です。ローカル管理の一時表領域では、**一時ファイル**が使用されます。一時表領域の外側のデータは変更されず、一時表領域データの REDO は発生しません。このため、読取り専用データベースまたはスタンバイ・データベースでディスク上ソート操作を実行できます。

一時ファイルの情報を表示するには、データファイルの場合とは異なるビューを使用します。V$TEMPFILE および DBA_TEMP_FILES ビューは、V$DATAFILE および DBA_DATA_FILES ビューに相当します。

ローカル管理の一時表領域を作成するには、CREATE TEMPORARY TABLESPACE 文を使用します。この文を発行するには、CREATE TABLESPACE システム権限が必要です。

次の文では、各エクステントが 16MB の一時表領域が作成されます。16MB の各エクステント（標準ブロック・サイズが 2KB のときは 8000 個のブロックに相当）は、このファイルのビットマップに 1 ビットで表されます。

```
CREATE TEMPORARY TABLESPACE lmtemp TEMPFILE '/u02/oracle/data/lmtemp01.dbf'
     SIZE 20M REUSE
     EXTENT MANAGEMENT LOCAL UNIFORM SIZE 16M;
```

すべての一時表領域は均一サイズのローカル管理エクステントを使用して作成されるため、一時表領域の場合、エクステント管理句はオプションです。SIZE のデフォルトは 1MB です。ただし、SIZE に別の値を指定する必要がある場合は、前述の文を使用します。

注意： 一部のオペレーティング・システムでは、一時ファイルのブロックが実際にアクセスされるまで、データベースは一時ファイル用の領域を割り当てません。このような領域割当ての遅延により、一時ファイルの作成やサイズ変更が短時間で済みますが、一時ファイルを後で使用するときに十分なディスク領域が使用可能である必要があります。使用しているシステムでデータベースがこのように一時ファイルを割り当てているかどうかは、オペレーティング・システムのマニュアルを参照して判断してください。

大型ファイル一時表領域の作成

通常の表領域と同様に、単一ファイル（大型ファイル）の一時表領域を作成できます。単一ファイルの一時表領域を作成するには、CREATE BIGFILE TEMPORARY TABLESPACE 文を使用します。大型ファイル表領域については、12-7 ページの「大型ファイル表領域の作成」および 12-21 ページの「大型ファイル表領域の変更」を参照してください。ただし、データファイルではなく一時ファイルを使用する一時表領域の作成を検討してください。

一時表領域の領域使用情報の表示

DBA_TEMP_FREE_SPACE ディクショナリ・ビューには、各一時表領域の領域使用に関する情報が表示されます。この情報には、割当て済領域と空き領域が含まれます。これらの統計を表示するには、次のコマンドを使用してこのビューを問い合せます。

```
SELECT * from DBA_TEMP_FREE_SPACE;

TABLESPACE_NAME                TABLESPACE_SIZE ALLOCATED_SPACE FREE_SPACE
------------------------------ --------------- --------------- ----------
TEMP                           250609664       250609664       249561088
```

複数の一時表領域：表領域グループの使用

ユーザーは、**一時表領域グループ**を使用して複数の表領域から一時領域を消費できます。単一の一時表領域ではなく表領域グループを使用することによって、ソート（特に多数のパーティションがある表でのソート）の結果を保持するのに 1 つの表領域では不十分な場合に発生する問題を回避できます。パラレル実行のサーバーで表領域グループを使用すると、1 回のパラレル操作で複数の一時表領域を使用できます。

表領域グループには、次の特性があります。

- 表領域グループには、1 つ以上の表領域が含まれます。1 つのグループに含まれる表領域の数に明示的な制限はありません。
- 表領域のネームスペースを共有するため、表領域グループにはグループ内の表領域と同じ名前は付けられません。
- 表領域グループ名は、データベースにデフォルト一時表領域を割り当てるとき、またはユーザーに一時表領域を割り当てるときに表領域名が表示される場所に指定できます。

表領域グループは明示的に作成しません。表領域グループは、最初の一時表領域がグループに割り当てられると暗黙的に作成されます。また、表領域グループに含まれる最後の一時表領域がグループから削除されると、その表領域グループが削除されます。

DBA_TABLESPACE_GROUPS ビューには、表領域グループとそのメンバーの表領域がリスト表示されます。

> **関連項目：** 一時表領域または表領域グループのユーザーへの割当てについては、『Oracle Database セキュリティ・ガイド』を参照してください。

表領域グループの作成

表領域グループは、CREATE TEMPORARY TABLESPACE 文または ALTER TABLESPACE 文に TABLESPACE GROUP 句を指定したときに、指定した表領域グループが存在していない場合に暗黙的に作成されます。

たとえば、group1 および group2 が両方とも存在しない場合は、次の文によってこの 2 つのグループが作成され、それぞれのグループには指定の表領域のみがメンバーとして含まれます。

```
CREATE TEMPORARY TABLESPACE lmtemp2 TEMPFILE '/u02/oracle/data/lmtemp201.dbf'
     SIZE 50M
     TABLESPACE GROUP group1;

ALTER TABLESPACE lmtemp TABLESPACE GROUP group2;
```

表領域グループのメンバーの変更

表領域を既存の表領域グループに追加するには、CREATE TEMPORARY TABLESPACE 文または ALTER TABLESPACE 文の TABLESPACE GROUP 句で既存の表領域グループ名を指定します。

次の文は、表領域を既存のグループに追加します。この文では、表領域 lmtemp3 を作成して group1 に追加します。その結果、group1 には表領域 lmtemp2 と lmtemp3 が含まれます。

```
CREATE TEMPORARY TABLESPACE lmtemp3 TEMPFILE '/u02/oracle/data/lmtemp301.dbf'
     SIZE 25M
     TABLESPACE GROUP group1;
```

次の文も表領域を既存のグループに追加しますが、この場合、表領域 lmtemp2 はすでに group1 に属しているため、実質的には group1 から group2 への移動となります。

```
ALTER TABLESPACE lmtemp2 TABLESPACE GROUP group2;
```

この結果、group2 には lmtemp と lmtemp2 が含まれ、group1 には tmtemp3 のみが含まれます。

次の文を使用すると、表領域をグループから削除できます。

```
ALTER TABLESPACE lmtemp3 TABLESPACE GROUP '';
```

これによって、表領域 lmtemp3 はどのグループにも属さなくなります。さらに、group1 に属するメンバーがなくなるため、group1 は暗黙的に削除されます。

デフォルト一時表領域としての表領域グループの割当て

ALTER DATABASE...DEFAULT TEMPORARY TABLESPACE 文を使用して、表領域グループをデータベースのデフォルト一時表領域として割り当てます。次に例を示します。

```
ALTER DATABASE sample DEFAULT TEMPORARY TABLESPACE group2;
```

これで、明示的に一時表領域が割り当てられていないユーザーは表領域 lmtemp と lmtemp2 を使用することになります。

表領域グループがデフォルト一時表領域に指定されている場合、そのグループのメンバーの表領域は削除できません。メンバーの表領域を削除するには、最初にその表領域を表領域グループから削除する必要があります。同様に、単一の一時表領域がデフォルト一時表領域に指定されている場合は、その一時表領域は削除できません。

表領域の非標準のブロック・サイズの指定

DB_BLOCK_SIZE 初期化パラメータで指定された標準のデータベース・ブロック・サイズとは異なるブロック・サイズの表領域を作成できます。この機能によって、ブロック・サイズの異なる表領域をデータベース間でトランスポートできます。

CREATE TABLESPACE 文の BLOCKSIZE 句を使用して、データベースの標準とは異なるブロック・サイズを指定して表領域を作成できます。BLOCKSIZE 句を正しく実行するためには、DB_CACHE_SIZE と、少なくとも 1 つの DB_nK_CACHE_SIZE 初期化パラメータをすでに設定している必要があります。さらに、BLOCKSIZE 句で指定する整数を 1 つの DB_nK_CACHE_SIZE パラメータの設定に対応付ける必要があります。冗長な指定になりますが、BLOCKSIZE を DB_BLOCK_SIZE 初期化パラメータで指定されている標準のブロック・サイズと同じに指定することも可能です。

次の文は表領域 lmtbsb を作成しますが、ブロック・サイズを DB_BLOCK_SIZE 初期化パラメータで指定されている標準のデータベース・ブロック・サイズとは異なるサイズにします。

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K
    BLOCKSIZE 8K;
```

関連項目：

- 2-28 ページ「データベース・ブロック・サイズの指定」
- DB_CACHE_SIZE および DB_nK_CACHE_SIZE パラメータ設定の詳細は、5-15 ページの「バッファ・キャッシュ初期化パラメータの設定」を参照してください。
- 12-30 ページ「データベース間での表領域のトランスポート」

REDO レコードの書込みの制御

一部のデータベース操作については、データベースで REDO レコードを生成するかどうかを制御できます。REDO を使用しないと、メディア・リカバリはできません。ただし、REDO の生成を抑制するとパフォーマンスが改善されるため、簡単にリカバリできる操作に適している場合があります。たとえば、CREATE TABLE...AS SELECT 文は、データベース障害やインスタンス障害が発生した場合に操作を繰り返すことができます。

これらの操作を表領域内のオブジェクトに対して実行するときに REDO を抑制する必要がある場合は、CREATE TABLESPACE 文に NOLOGGING 句を指定します。この句を指定しないか、かわりに LOGGING を指定した場合は、表領域内のオブジェクトに変更が行われると REDO が生成されます。一時セグメントや一時表領域の場合は、ロギング属性に関係なく REDO は生成されません。

表領域レベルで指定するロギング属性は、その表領域内で作成されるオブジェクトのデフォルト属性になります。このデフォルトのロギング属性は、CREATE TABLE 文を使用するなど、スキーマ・オブジェクト・レベルで LOGGING や NOLOGGING を指定することで上書きできます。

スタンバイ・データベースがある場合は、NOLOGGING 句を指定すると、そのスタンバイ・データベースの可用性と精度に問題が生じます。この問題を克服するために、FORCE LOGGING モードを指定できます。CREATE TABLESPACE 文に FORCE LOGGING 句を指定すると、表領域内のオブジェクトを変更するすべての操作について、REDO レコードを強制的に生成させることができます。これにより、オブジェクト・レベルでの指定が上書きされます。

FORCE LOGGING モードの表領域を別のデータベースにトランスポートすると、新しい表領域では FORCE LOGGING モードは維持されません。

関連項目：

- NOLOGGING モードで実行できる操作の詳細は、『Oracle Database SQL リファレンス』を参照してください。
- FORCE LOGGING モードの詳細と、CREATE DATABASE 文で FORCE LOGGING 句を使用する効果の詳細は、2-23 ページの「FORCE LOGGING モードの指定」を参照してください。

表領域の可用性の変更

オンラインの表領域をオフライン化すると、一般的な使用を一時的に禁止にできます。データベースの残りの部分はオープンしていて使用可能であり、ユーザーはデータにアクセスできます。逆に、オフライン状態の表領域をオンライン化して、データベース・ユーザーがその表領域内のスキーマ・オブジェクトを使用できるようにすることもできます。表領域の可用性を変更するには、データベースをオープンする必要があります。

表領域の可用性を変更するには、ALTER TABLESPACE 文を使用します。そのためには、ALTER TABLESPACE または MANAGE TABLESPACE システム権限が必要です。

> **関連項目：** 表領域内の各データファイルの可用性を変更する方法については、13-6 ページの「データファイルの可用性の変更」を参照してください。

表領域のオフライン化

表領域は、次のような場合にオフライン化できます。

- データベースの一部のみを使用できないようにし、残りの部分には正常にアクセスできるようにする場合
- オフライン表領域のバックアップを実行する場合（ただし、表領域はオンラインで使用中の場合でもバックアップは可能）
- アプリケーションの更新時またはメンテナンス時に、アプリケーションとその表のグループを一時的に使用できないようにする場合
- 表領域のデータファイルを名前変更または再配置する場合

  詳細は、13-8 ページの「データファイルの名前変更と再配置」を参照してください。

表領域をオフライン化すると、その関連ファイルがすべてオフライン化されます。

次の表領域はオフライン化できません。

- SYSTEM
- UNDO 表領域
- 一時表領域

表領域をオフライン化する前に、その表領域がデフォルト表領域としてすでに割り当てられているユーザーの表領域割当ての変更を考慮してください。このようなユーザーは表領域がオフラインの間その中のオブジェクトにアクセスできないため、表領域割当ての変更をお薦めします。

ALTER TABLESPACE...OFFLINE 文では、次のパラメータを指定できます。

句	説明
NORMAL	表領域のどのデータファイルにもエラー条件が存在していない場合は、この表領域を通常の方法でオフライン化できます。書込みエラーが発生していると、現時点では表領域のデータファイルをオフライン化することはできません。OFFLINE NORMAL を指定すると、データベースは表領域のデータファイルすべてのチェックポイントを取ってから、それらのファイルをオフライン化します。NORMAL はデフォルトです。

句	説明
TEMPORARY	表領域の 1 つまたは複数のデータファイルについてエラー条件が存在している場合でも、表領域を一時的にオフライン化できます。OFFLINE TEMPORARY を指定すると、データベースはまだオフライン化されていないデータファイルのチェックポイントを取ってから、これらのファイルをオフライン化します。 オフラインになっているファイルがないときに表領域を一時的にオフライン化する場合は、表領域をオンラインに戻す前にメディア・リカバリを実行する必要はありません。しかし、表領域の 1 つまたは複数のファイルが書込みエラーのためにオフラインになっており、この表領域を一時的にオフライン化する場合は、表領域をオンラインに戻す前にリカバリする必要があります。
IMMEDIATE	表領域が即時にオフライン化されます。データベースはデータファイルのチェックポイントを取りません。OFFLINE IMMEDIATE を指定すると、表領域をオンライン化する前に表領域のメディア・リカバリが必要です。データベースを NOARCHIVELOG モードで運用している場合は、表領域を即時にオフライン化することはできません。

注意： 表領域をオフライン化する必要がある場合は、可能なかぎり NORMAL 句（デフォルト）を使用してください。この設定によって、不完全リカバリの後に ALTER DATABASE OPEN RESETLOGS 文を使用して REDO ログ順序をリセットした場合でも、表領域をオンラインに戻すためのリカバリが不要になることが保証されます。

TEMPORARY は、表領域を通常の方法でオフライン化できないときのみ指定してください。この場合、エラーのためにオフライン化されたファイルのみをリカバリする必要があり、その後に表領域をオンライン化できます。IMMEDIATE は、NORMAL 設定と TEMPORARY 設定を試した後にのみ指定してください。

次の例では、users 表領域を通常の方法でオフライン化しています。

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

表領域のオンライン化

データベースがオープンされている場合は、いつでも Oracle Database 内の任意の表領域をオンライン化できます。通常、表領域は、データベース・ユーザーがその中のデータを使用できるようにオンラインになっています。

オンライン化しようとする表領域が、正常に（ALTER TABLESPACE OFFLINE 文の NORMAL 句を使用して）オフライン化されていない場合は、最初にメディア・リカバリをしないかぎりオンライン化できません。メディア・リカバリを実行しないと、エラーが返されて表領域はオフラインのままになります。

関連項目： メディア・リカバリの実行方法については、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

次の文は、users 表領域をオンライン化します。

```
ALTER TABLESPACE users ONLINE;
```

読取り専用表領域の使用

表領域を読取り専用にすると、表領域のデータファイルに対して書込み操作ができなくなります。読取り専用表領域の主な目的は、データベース内の大規模かつ静的部分のバックアップおよびリカバリを実行しなくて済むようにすることです。また、読取り専用表領域は、ユーザーが履歴データを変更できないように履歴データを完全に保護する手段でもあります。表領域を読取り専用にすると、その表領域内のすべての表はユーザーの更新権限レベルに関係なく更新できません。

注意： 表領域は、それが作成されたデータベース内でしかオンライン化できないため、読取り専用にすること自体でアーカイブ要件やデータ公開要件を満たすことはできません。ただし、12-30 ページの「データベース間での表領域のトランスポート」で説明するように、トランスポータブル表領域機能を使用すると、これらの要件を満たすことができます。

表や索引などの項目は読取り専用表領域から削除できますが、読取り専用表領域内のオブジェクトは作成または変更できません。`ALTER TABLE...ADD` または `ALTER TABLE...MODIFY` など、データ・ディクショナリ内のファイル記述を更新する文は実行できますが、新しい記述は表領域を読取り / 書込み用にするまでは使用できません。

読取り専用表領域は、他のデータベースにトランスポートすることもできます。読取り専用表領域は更新できないため、CD-ROM または Write Once-Read Many（WORM）デバイスに格納できます。

この項の内容は、次のとおりです。

- 表領域を読取り専用にする方法
- 読取り専用表領域を書込み可能にする方法
- WORM デバイスでの読取り専用表領域の作成
- 読取り専用表領域内にあるデータファイルのオープンの遅延

関連項目： 12-30 ページ「データベース間での表領域のトランスポート」

表領域を読取り専用にする方法

すべての表領域は、最初は読取り / 書込み用として作成されます。表領域を読取り専用に変更するには、`ALTER TABLESPACE` 文で `READ ONLY` キーワードを使用します。そのためには、`ALTER TABLESPACE` または `MANAGE TABLESPACE` システム権限が必要です。

表領域を読取り専用にするには、あらかじめ次の条件を満たす必要があります。

- 表領域は必ずオンラインにする。これにより、表領域に適用する必要のあるロールバック情報がないことが保証されます。
- 表領域をアクティブな UNDO 表領域または `SYSTEM` 表領域にしない。
- 表領域を現行のオンライン・バックアップに含めない（オンライン・バックアップは、終了時に表領域内にあるすべてのデータファイルのヘッダー・ファイルを更新するためです）。

読取り専用表領域のデータにアクセスする際のパフォーマンスを向上させるため、表領域を読取り専用にする直前に、表領域内の表のブロックすべてにアクセスする問合せを発行することをお薦めします。各表に対して `SELECT COUNT (*)` などの単純な問合せを実行しておくと、それ以降、表領域のデータ・ブロックに最も効率的にアクセスできるようになります。これによって、最後にブロックを変更したトランザクションの状態をデータベースが確認する必要がなくなるからです。

次の文は、`flights` 表領域を読取り専用にします。

```
ALTER TABLESPACE flights READ ONLY;
```

データベースのトランザクション処理中に、ALTER TABLESPACE...READ ONLY 文を発行できます。この文が発行されると、表領域は推移読取り専用状態になります。トランザクションでは、その表領域に対して DML 文を使用した変更ができなくなります。変更を試行すると、そのトランザクションは終了してロールバックされます。ただし、すでに変更を実行し、追加変更を試行しないトランザクションは、コミットまたはロールバックできます。

ALTER TABLESPACE...READ ONLY 文は、戻る前に、表領域に対する変更が保留中またはコミット解除されたトランザクション、およびこの文の発行前に開始されたトランザクションがコミットまたはロールバックされるのを待機します。文の発行前に開始されたトランザクションがアクティブなままであっても、表領域に対する変更をロールバックしてセーブポイントまでロールバックすると、文はこのアクティブ・トランザクションを待機しなくなります。

注意： この推移読取り専用状態になるのは、初期化パラメータ COMPATIBLE の値が 8.1.0 以上の場合のみです。このパラメータが 8.1.0 より小さい値に設定されている場合は、アクティブなトランザクションが存在していると、ALTER TABLESPACE...READ ONLY 文は失敗します。

ALTER TABLESPACE 文の完了までに長時間かかる場合は、読取り専用状態になるのを妨げているトランザクションを識別できます。次に、それらのトランザクションの所有者に通知し、必要に応じてトランザクションを終了させるかどうかを決定できます。

次の例では、ALTER TABLESPACE...READ ONLY 文に対応するトランザクション・エントリが識別され、そのセッション・アドレス（saddr）が表示されます。

```
SELECT SQL_TEXT, SADDR
    FROM V$SQLAREA,V$SESSION
    WHERE V$SQLAREA.ADDRESS = V$SESSION.SQL_ADDRESS
        AND SQL_TEXT LIKE 'alter tablespace%';

SQL_TEXT                                 SADDR
---------------------------------------- --------
alter tablespace tbs1 read only          80034AF0
```

各アクティブ・トランザクションの開始システム変更番号（SCN）は、V$TRANSACTION ビューに格納されています。このビューを開始 SCN の昇順でソートして表示すると、トランザクションが実行順にリストされます。前述の例の場合は、読取り専用文のトランザクション・エントリのセッション・アドレスがわかっているので、V$TRANSACTION ビューで特定できます。開始 SCN よりも小さい番号を持つトランザクション（以前に実行されたトランザクションを示します）はすべて、表領域の停止とその後の読取り専用状態になるのを妨げている可能性があります。

```
SELECT SES_ADDR, START_SCNB
    FROM V$TRANSACTION
    ORDER BY START_SCNB;

SES_ADDR START_SCNB
-------- ----------
800352A0       3621   --> waiting on this txn
80035A50       3623   --> waiting on this txn
80034AF0       3628   --> this is the ALTER TABLESPACE statement
80037910       3629   --> don't care about this txn
```

この時点で、ブロックしているトランザクションの所有者を見つけることができます。

```
SELECT T.SES_ADDR, S.USERNAME, S.MACHINE
   FROM V$SESSION S, V$TRANSACTION T
   WHERE T.SES_ADDR = S.SADDR
   ORDER BY T.SES_ADDR

SES_ADDR USERNAME             MACHINE
-------- -------------------- --------------------
800352A0 DAVIDB               DAVIDBLAP             --> Contact this user
```

```
80035A50 MIKEL               LAB61                --> Contact this user
80034AF0 DBA01               STEVEFLAP
80037910 NICKD               NICKDLAP
```

表領域を読取り専用にした後は、その表領域のバックアップをただちに作成することをお薦めします。表領域は読取り専用になっているかぎり変更できないため、それ以後のバックアップは不要です。

> **関連項目：**『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』

読取り専用表領域を書込み可能にする方法

表領域を書込み可能に変更するには、ALTER TABLESPACE 文で READ WRITE キーワードを指定します。そのためには、ALTER TABLESPACE または MANAGE TABLESPACE システム権限が必要です。

表領域を読取り / 書込み用にするには、前提条件として、表領域のみでなく、そのすべてのデータファイルをオンライン化する必要があります。データファイルをオンライン化するには、ALTER DATABASE 文の DATAFILE...ONLINE 句を使用します。データファイルの現行の状態を確認するには、V$DATAFILE ビューを使用します。

次の文は、flights 表領域を書込み可能にします。

```
ALTER TABLESPACE flights READ WRITE;
```

読取り専用表領域を書込み可能に変更すると、データファイルの制御ファイル・エントリが更新されるため、読取り専用バージョンのデータファイルをリカバリの開始点として使用できます。

WORM デバイスでの読取り専用表領域の作成

CD-ROM または WORM デバイスに読取り専用表領域を作成する手順は、次のとおりです。

1. 別のデバイスに書込み可能表領域を作成します。その表領域に属するオブジェクトを作成して、データを挿入します。
2. 表領域を読取り専用に変更します。
3. 表領域のデータファイルを WORM デバイスにコピーします。ファイルをコピーするには、オペレーティング・システムのコマンドを使用します。
4. 表領域をオフライン化します。
5. データファイルの名前を、WORM デバイスにコピーしたファイルと一致するように名前変更します。これには、RENAME DATAFILE 句を指定した ALTER TABLESPACE 文を使用します。データファイルの名前を変更すると、制御ファイルに記述されているこれらのファイルの名前も変更されます。
6. 表領域をオンライン化します。

読取り専用表領域内にあるデータファイルのオープンの遅延

大規模データベースのほとんどが、アクセス速度の遅いデバイスや階層形式の記憶デバイス上にある読取り専用表領域に格納されている場合は、READ_ONLY_OPEN_DELAYED 初期化パラメータを TRUE に設定することを検討する必要があります。これにより、読取り専用表領域内のデータファイルは、そこに格納されたデータの読取り試行時に初めてアクセスされるため、データベースのオープンなど、特定の操作が高速になります。

READ_ONLY_OPEN_DELAYED=TRUE に設定すると、次のような副次的な影響があります。

- オープン時に、読取り専用の欠落ファイルや不良ファイルが検出されません。これらのファイルは、アクセス試行時にのみ検出されます。
- ALTER SYSTEM CHECK DATAFILES では、読取り専用ファイルはチェックされません。
- ALTER TABLESPACE...ONLINE および ALTER DATABASE DATAFILE...ONLINE では、読取り専用ファイルはチェックされません。最初のアクセス時にのみチェックされます。
- V$RECOVER_FILE、V$BACKUP および V$DATAFILE_HEADER は、読取り専用ファイルにアクセスしません。読取り専用ファイルは結果リスト上に「DELAYED OPEN」というエラーで示され、他の列の値は 0（ゼロ）になります。
- V$DATAFILE は読取り専用ファイルにアクセスしません。読取り専用ファイルにはサイズ「0」がリストされます。
- V$RECOVER_LOG は読取り専用ファイルにアクセスしません。リカバリに必要な可能性があるログは、リストに追加されません。
- ALTER DATABASE NOARCHIVELOG は読取り専用ファイルにアクセスしません。リカバリが必要な読取り専用ファイルがある場合でも、処理が継続します。

注意：

- RECOVER DATABASE および ALTER DATABASE OPEN RESETLOGS は、パラメータ値に関係なく、すべての読取り専用データファイルに引き続きアクセスします。これらの操作で読取り専用ファイルへのアクセスを回避する場合は、該当ファイルをオフライン化する必要があります。
- バックアップ制御ファイルを使用すると、一部のファイルの読取り専用状態が不正確になる場合があります。これにより、これらの操作の一部で予期しない結果が返されることがあります。この状況には注意が必要です。

表領域の変更とメンテナンス

ここでは、表領域の変更とメンテナンスに関する事項について説明します。この項の内容は、次のとおりです。

ローカル管理表領域の変更

ローカル管理表領域をローカル管理の一時表領域に変更できません。また、セグメント領域の管理方法を変更することもできません。ローカル管理表領域では、使用可能エクステントを結合する必要はありません。ただし、次のような操作の場合は、ALTER TABLESPACE 文をローカル管理表領域に対して使用できます。

- データファイルを追加する場合。次に例を示します。

```
ALTER TABLESPACE lmtbsb
   ADD DATAFILE '/u02/oracle/data/lmtbsb02.dbf' SIZE 1M;
```

- 表領域の可用性（ONLINE/OFFLINE）を変更する場合。12-15 ページの「表領域の可用性の変更」を参照してください。
- 表領域を読取り専用または読取り / 書込み用にする場合。12-17 ページの「読取り専用表領域の使用」を参照してください。
- データファイルの名前を変更したり、表領域内のデータファイルのサイズの自動拡張を使用可能または使用禁止にする場合。第 13 章「データファイルおよび一時ファイルの管理」を参照してください。

大型ファイル表領域の変更

ALTER TABLESPACE 文の次の 2 つの句は、大型ファイル表領域使用時におけるデータファイルの透過性をサポートします。

- RESIZE: RESIZE 句を使用すると、大型ファイル表領域内の単一データファイルを参照せずに、そのデータファイルのサイズを絶対サイズに変更できます。次に例を示します。

```
ALTER TABLESPACE bigtbs RESIZE 80G;
```

- AUTOEXTEND（ADD DATAFILE 句の範囲外で使用）:

  大型ファイル表領域では、ADD DATAFILE 句の範囲外で AUTOEXTEND 句を使用できます。次に例を示します。

```
ALTER TABLESPACE bigtbs AUTOEXTEND ON NEXT 20G;
```

大型ファイル表領域に対して ADD DATAFILE 句を指定すると、エラーが発生します。

ローカル管理の一時表領域の変更

> **注意：** ALTER TABLESPACE 文に TEMPORARY キーワードを指定して、ローカル管理の永続表領域をローカル管理の一時表領域に変更することはできません。ローカル管理の一時表領域を作成するには、CREATE TEMPORARY TABLESPACE 文を使用する必要があります。

次の例のように、ALTER TABLESPACE を使用すると、一時ファイルを追加、オフライン化またはオンライン化できます。

```
ALTER TABLESPACE lmtemp
   ADD TEMPFILE '/u02/oracle/data/lmtemp02.dbf' SIZE 18M REUSE;

ALTER TABLESPACE lmtemp TEMPFILE OFFLINE;
ALTER TABLESPACE lmtemp TEMPFILE ONLINE;
```

> **注意：** 一時表領域はオフライン化できません。かわりに、一時ファイルをオフライン化します。V$TEMPFILE ビューには、一時ファイルのオンライン化の状態が表示されます。

ALTER DATABASE 文を使用すると、一時ファイルを変更できます。

次の文では、一時ファイルをオフラインにしてから、オンラインに戻しています。前述の例の最後の 2 つの ALTER TABLESPACE 文と同様に動作します。

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' OFFLINE;
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' ONLINE;
```

次の文では、一時ファイルのサイズが変更されます。

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' RESIZE 18M;
```

次の文では、一時ファイルが削除され、オペレーティング・システム・ファイルが削除されます。

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' DROP
    INCLUDING DATAFILES;
```

この一時ファイルが属していた表領域は残ります。アラート・ログには、一時ファイルが削除されたことを示すメッセージが書き込まれます。オペレーティング・システム・エラーによってファイルが削除されなかった場合でも文は正常終了しますが、エラーを示すメッセージがアラート・ログに書き込まれます。

また、ALTER DATABASE 文を使用して、既存の一時ファイルの自動拡張を使用可能または使用禁止にしたり、一時ファイル名を変更（RENAME FILE）できます。必要な構文については、『Oracle Database SQL リファレンス』を参照してください。

> **注意：** 一時ファイルの名前を変更するには、一時ファイルをオフライン化し、オペレーティング・システムのコマンドを使用して、その一時ファイルを名前変更または再配置します。次に、ALTER DATABASE RENAME FILE コマンドを使用してデータベースの制御ファイルを更新します。

ローカル管理の一時表領域の縮小

データベースで大規模なソート操作を実行すると、一時表領域が増大し、ディスク領域の容量が大幅に占有される場合があります。ソート操作が完了しても余分になった領域は解放されず、使用可能マークと再利用可能マークが付けられるのみです。したがって、単一の大規模なソート操作を実行すると、ソート操作の完了後に大量の割当て済一時領域が未使用のままになります。このため、データベースでは、ローカル管理の一時表領域を縮小して未使用領域を解放できます。

一時表領域を縮小する場合は、ALTER TABLESPACE 文の SHRINK SPACE 句を使用します。一時表領域の特定の一時ファイルを縮小する場合は、ALTER TABLESPACE 文の SHRINK TEMPFILE 句を使用します。表領域または一時ファイルの他の属性を維持しながら可能なかぎり空き領域を縮小します。オプションの KEEP 句は、表領域または一時ファイルの最小サイズを定義します。

縮小はオンライン操作です。これは、ユーザー・セッションは必要に応じてソート・エクステントの割当てを継続でき、すでに実行中の問合せは影響を受けないことを意味します。

次の例では、ローカル管理の一時表領域の lmtmp1 を 20MB のサイズに縮小しています。

```
ALTER TABLESPACE lmtemp1 SHRINK SPACE KEEP 20M;
```

次の例では、ローカル管理の一時表領域 lmtmp2 の一時ファイル lmtemp02.dbf を縮小しています。KEEP 句を省略しているため、データベースでは、一時ファイルを最小可能サイズまで縮小することを試みます。

```
ALTER TABLESPACE lmtemp2 SHRINK TEMPFILE '/u02/oracle/data/lmtemp02.dbf';
```

表領域の名前変更

永続表領域または一時表領域の名前は、ALTER TABLESPACE 文の RENAME TO 句を使用して変更できます。たとえば、次の文は users 表領域の名前を変更します。

```
ALTER TABLESPACE users RENAME TO usersts;
```

表領域の名前を変更すると、データ・ディクショナリ、制御ファイルおよび（オンライン）データファイル・ヘッダー内でその表領域名への参照がすべて更新されます。表領域 ID は変更されないため、たとえば、その表領域がユーザーのデフォルト表領域の場合、DBA_USERS ビューには名前が変更された表領域がユーザーのデフォルト表領域として表示されます。

この文の操作では、次の点に注意してください。

- COMPATIBLE パラメータは、10.0.0 以上に設定する必要があります。
- 名前を変更する表領域が SYSTEM 表領域または SYSAUX 表領域の場合、名前は変更されず、エラーが発生します。
- 表領域内のデータファイルがオフラインの場合、または表領域がオフラインの場合、その表領域の名前は変更されず、エラーが発生します。
- 表領域が読取り専用の場合、データファイル・ヘッダーは更新されません。これは破損とはみなされませんが、データファイル・ヘッダーの名前が変更されなかったことを示すメッセージがアラート・ログに書き込まれます。データ・ディクショナリと制御ファイルは更新されます。
- 表領域がデフォルト一時表領域の場合は、データベース・プロパティ表内の対応するエントリが更新され、DATABASE_PROPERTIES ビューに新しい名前が表示されます。
- 表領域が UNDO 表領域で、次の条件が満たされると、サーバー・パラメータ・ファイル（SPFILE）の表領域名は新しい表領域名に変更されます。
  - サーバー・パラメータ・ファイルを使用して、インスタンスを起動した場合。
  - 表領域名がインスタンスに対して UNDO_TABLESPACE で指定されている場合。

  従来の初期化パラメータ・ファイル（PFILE）を使用している場合は、初期化パラメータ・ファイルを手動で変更する必要があることを示すメッセージがアラート・ログに書き込まれます。

表領域の削除

表領域とその内容が不要になった場合は、その表領域と内容（表領域に含まれるセグメント）をデータベースから削除できます。表領域を削除するには、DROP TABLESPACE システム権限が必要です。

注意： 削除された表領域のデータはリカバリできません。そのため、削除しようとしている表領域に含まれているデータはすべて、将来的に必要のないことを確かめてください。また、表領域をデータベースから削除する直前および直後に、データベースの完全バックアップを作成する必要があります。表領域を誤って削除した場合、または表領域を削除した後にデータベースで問題が発生した場合は、データベースをリカバリできるように、必ずバックアップを作成することをお薦めします。

表領域を削除すると、対応付けられたデータベースの制御ファイル中のファイル・ポインタのみが削除されます。必要に応じて、削除された表領域を構成していたオペレーティング・システム・ファイル（データファイル）を削除するように Oracle Database に指示することもできます。表領域の削除と同時にデータファイルを削除するようにデータベースに指示しない場合は、後でオペレーティング・システムの適切なコマンドを使用して削除する必要があります。

アクティブなセグメントを含む表領域は削除できません。たとえば、表領域内の表が現在使用されている場合、またはコミットされていないトランザクションをロールバックする必要がある UNDO データが表領域に含まれている場合、その表領域は削除できません。表領域はオンラインでもオフラインでもかまいませんが、削除する前にオフラインにすることをお薦めします。

表領域を削除するには、DROP TABLESPACE 文を使用します。次の文は、users 表領域を、その中のセグメントも含めて削除します。

```
DROP TABLESPACE users INCLUDING CONTENTS;
```

表領域が空の場合（表、ビューまたは他の構造が格納されていない場合）は、INCLUDING CONTENTS 句を指定する必要はありません。CASCADE CONSTRAINTS 句を使用すると、表領域内の表の主キーと一意キーを参照する別の表領域の表から、すべての参照整合性制約を削除できます。

表領域の削除と同時に表領域に対応付けられたデータファイルを削除するには、INCLUDING CONTENTS AND DATAFILES 句を使用します。次の文は、users 表領域とそれに対応付けられているデータファイルを削除します。

```
DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;
```

アラート・ログには、削除された各データファイルのメッセージが書き込まれます。オペレーティング・システム・エラーによってファイルが削除されなかった場合でも DROP TABLESPACE 文は正常終了しますが、エラーを示すメッセージがアラート・ログに書き込まれます。

関連項目： 13-11 ページ「データファイルの削除」

SYSAUX 表領域の管理

SYSAUX 表領域は、データベースの作成時に、SYSTEM 表領域の補助表領域としてインストールされます。これまで個別に表領域を作成して使用していた一部のデータベースのコンポーネントは、SYSAUX 表領域に含まれるようになりました。

SYSAUX 表領域が使用不可能になった場合でも、データベースのコア機能は実行可能です。SYSAUX 表領域を使用しているデータベース機能はエラーとなるか、機能が制限される可能性があります。

SYSAUX 表領域に含まれる占有データの監視

SYSAUX 表領域の登録済占有データのリストは、2-17 ページの「SYSAUX 表領域の概要」を参照してください。これらのコンポーネントは SYSAUX 表領域を使用し、SYSAUX 表領域を占有する方法がインストール時に提供されます。

SYSAUX 表領域の占有データは、V$SYSAUX_OCCUPANTS ビューを使用して監視できます。このビューには、SYSAUX 表領域の占有データに関する次の情報がリスト表示されます。

- 占有データの名前
- 占有データの説明
- スキーマ名
- 移動プロシージャ
- 現行の領域使用

ビュー情報は、占有データ別にメンテナンスされます。

> **関連項目：** V$SYSAUX_OCCUPANTS ビューの詳細は、『Oracle Database リファレンス』を参照してください。

SYSAUX 表領域内外への占有データの移動

コンポーネントのインストール時には、コンポーネントを SYSAUX に常駐させないように指定することもできます。また、後でコンポーネントを指定の表領域に再配置する必要が生じた場合、そのコンポーネントについては、V$SYSAUX_OCCUPANTS ビューで指定した移動プロシージャを使用して移動を実行できます。

たとえば、デフォルトの表領域である SYSAUX に Oracle Ultra Search をインストールしたとします。後で、Ultra Search がかなりの領域を使用していることがわかりました。SYSAUX への領域負担を軽減するために、V$SYSAUX_OCCUPANTS ビューで指定した PL/SQL 移動プロシージャをコールして、Ultra Search を別の表領域に再配置できます。

移動プロシージャを使用すると、コンポーネントを他の表領域から SYSAUX 表領域に移動することもできます。

SYSAUX 表領域のサイズの制御

SYSAUX 表領域は、多数のデータベース・コンポーネントによって占有され（表 2-2 を参照）、その合計サイズはそれらのコンポーネントが消費する領域によって決定します。同様に、コンポーネントが消費する領域は、使用される機能およびデータベース・ワークロードの性質によって決定します。

SYSAUX 表領域を最も大きく占有するのは自動ワークロード・リポジトリ（AWR）です。AWR が消費する領域は、特定の時間におけるシステム内でのアクティブなセッションの数、スナップショット間隔、履歴データ保存期間など、いくつかの要因によって決定します。同時にアクティブなセッションが平均 10 ある標準的なシステムでは、AWR データ用として約 200 ～ 300MB の領域が必要になる場合があります。

次の表に、システム構成と予測される負荷に基づいて SYSAUX 表領域のサイズを設定するためのガイドラインを示します。

パラメータ / 推奨設定	小型	中型	大型
CPU 数	2	8	32
同時アクティブ・セッション数	10	20	100
ユーザー・オブジェクト数 : 表および索引	500	5,000	50,000
デフォルト構成で安定した状態での予測 SYSAUX のサイズ	500 MB	2 GB	5 GB

AWR のサイズを制御するには、スナップショット間隔と履歴データ保存期間を変更します。AWR のスナップショット間隔と保存期間の管理の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

SYSAUX 表領域のもう 1 つの主要な占有データは、埋込みの Enterprise Manager（EM）リポジトリです。このリポジトリは Oracle Enterprise Manager Database Control で使用され、そのメタデータが格納されます。このリポジトリのサイズは、データベース・アクティビティ、およびリポジトリに格納された構成関連情報によって異なります。

他のデータベース・コンポーネントが消費する SYSAUX 表領域の領域サイズは、関連する機能（Oracle UltraSearch、Oracle Text、Oracle Streams など）を使用中の場合のみ大きくなります。このような機能を使用していない場合、これらのコンポーネントは SYSAUX 表領域のサイズに大きく影響しません。

ローカル管理表領域の問題の診断と修復

Oracle Database には、DBMS_SPACE_ADMIN パッケージが組み込まれています。このパッケージは、ローカル管理表領域の問題の診断と修復に使用するサポートの集まりです。

DBMS_SPACE_ADMIN パッケージのプロシージャ

次の表に、DBMS_SPACE_ADMIN パッケージに含まれるプロシージャを示します。各プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

プロシージャ	説明
ASSM_SEGMENT_VERIFY	自動セグメント領域管理が使用可能な表領域内に作成されたセグメントの整合性を検証します。*sid*_ora_*process_id*.trc という名前のダンプ・ファイルを、V$DIAG_INFO ビューの Diag Trace エントリに対応する位置に出力します。 手動セグメント領域管理が指定されている表領域には、SEGMENT_VERIFY を使用します。
ASSM_TABLESPACE_VERIFY	自動セグメント領域管理が使用可能な表領域の整合性を検証します。*sid*_ora_*process_id*.trc という名前のダンプ・ファイルを、V$DIAG_INFO ビューの Diag Trace エントリに対応する位置に出力します。 手動セグメント領域管理が指定されている表領域には、TABLESPACE_VERIFY を使用します。
SEGMENT_CORRUPT	適切なエラー・リカバリを実行できるように、セグメントに破損または有効マークを付けます。
SEGMENT_DROP_CORRUPT	現在、破損マークが設定されているセグメントを削除します（領域は再生しません）。
SEGMENT_DUMP	特定のセグメントのセグメント・ヘッダーおよびビットマップ・ブロックを、V$DIAG_INFO ビューの Diag Trace エントリに対応する位置にある *sid*_ora_*process_id*.trc という名前のダンプ・ファイルにダンプします。オプションで、セグメント・ヘッダーおよびビットマップ・ブロック・サマリーを含む簡略化したダンプ（各ブロックの空き領域率が含まれていない）を選択できます。

プロシージャ	説明
SEGMENT_VERIFY	セグメントのエクステント・マップの整合性を検証します。
TABLESPACE_FIX_BITMAPS	適切な DBA 範囲（エクステント）にビットマップ内で使用可能マークまたは使用済マークを付けます。
TABLESPACE_FIX_SEGMENT_STATES	移行が停止した表領域内のセグメントの状態を修正します。
TABLESPACE_MIGRATE_FROM_LOCAL	ローカル管理表領域をディクショナリ管理表領域に移行します。
TABLESPACE_MIGRATE_TO_LOCAL	ディクショナリ管理表領域をローカル管理表領域に移行します。
TABLESPACE_REBUILD_BITMAPS	適切なビットマップを再作成します。
TABLESPACE_REBUILD_QUOTAS	特定の表領域の割当て制限を再作成します。
TABLESPACE_RELOCATE_BITMAPS	ビットマップを指定の保存先に再配置します。
TABLESPACE_VERIFY	表領域のセグメントのビットマップとエクステント・マップが同期しているかどうかを検証します。

次の使用例では、DBMS_SPACE_ADMIN パッケージを使用して問題を診断し、解決できる代表的な状況について説明します。

注意： 前述の一部のプロシージャは、正しく使用しないとデータが消失してリカバリ不能になる場合があります。これらのプロシージャに不明な点がある場合は、Oracle サポート・サービスと共同で作業を行ってください。

関連項目：

- DBMS_SPACE_ADMIN パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 8-8 ページ「V$DIAG_INFO ビューを使用した ADR の場所の表示」

使用例 1: 割当て済ブロックが空き（オーバーラップなし）とマークされているときのビットマップの修復

TABLESPACE_VERIFY プロシージャの使用時に、ビットマップ内で「空き」マークが付いているブロックがセグメントに割り当てられたにもかかわらず、セグメント間のオーバーラップがレポートされていないことが検出された場合。

この使用例では、次のタスクを実行してください。

1. SEGMENT_DUMP プロシージャをコールして、管理者がそのセグメントに割り当てた範囲をダンプします。
2. 範囲ごとに、TABLESPACE_EXTENT_MAKE_USED オプションを指定して TABLESPACE_FIX_BITMAPS プロシージャをコールし、領域に使用済のマークを付けます。
3. TABLESPACE_REBUILD_QUOTAS をコールして割当て制限を再作成します。

使用例 2: 破損したセグメントの削除

ビットマップに「空き」マークが付いたセグメント・ブロックがあるため、セグメントを削除できない場合。このセグメントには、自動的に「破損」マークが付けられます。

この使用例では、次のタスクを実行してください。

1. SEGMENT_VERIFY_EXTENTS_GLOBAL オプションを指定して SEGMENT_VERIFY プロシージャをコールします。オーバーラップがレポートされない場合は、手順 2 から 5 までを実行します。
2. SEGMENT_DUMP プロシージャをコールして、そのセグメントに割り当てられたデータ・ブロック・アドレス範囲をダンプします。
3. 範囲ごとに、TABLESPACE_EXTENT_MAKE_FREE オプションを指定して TABLESPACE_FIX_BITMAPS プロシージャをコールし、領域に「空き」のマークを付けます。
4. SEGMENT_DROP_CORRUPT をコールして SEG$ エントリを削除します。
5. TABLESPACE_REBUILD_QUOTAS をコールして割当て制限を再作成します。

使用例 3: オーバーラップがレポートされたビットマップの修復

TABLESPACE_VERIFY プロシージャで、いくつかオーバーラップがレポートされる場合。前の内部エラーに基づいて、一部の実データを削除する必要があります。

この場合、表 t1 などの削除するオブジェクトを選択してから、次のタスクを実行します。

1. t1 がオーバーラップしているすべてのオブジェクトのリストを作成します。
2. 表 t1 を削除します。必要に応じて、SEGMENT_DROP_CORRUPT プロシージャをコールしてフォローアップします。
3. t1 がオーバーラップしていたすべてのオブジェクトに対して、SEGMENT_VERIFY プロシージャをコールします。必要に応じて、TABLESPACE_FIX_BITMAPS プロシージャをコールして該当するビットマップに使用済を示すマークを付けます。
4. TABLESPACE_VERIFY プロシージャを再度実行し、問題が解決したかどうかを検証します。

使用例 4: ビットマップ・ブロックのメディア破損の訂正

ビットマップ・ブロックの集合にメディア破損がある場合。

この使用例では、次のタスクを実行してください。

1. すべてのビットマップ・ブロック、または 1 つしか破損していない場合はそのブロックに対して、TABLESPACE_REBUILD_BITMAPS プロシージャをコールします。
2. TABLESPACE_REBUILD_QUOTAS をコールして割当て制限を再作成します。
3. TABLESPACE_VERIFY プロシージャをコールして、ビットマップの整合性を検証します。

使用例 5: ディクショナリ管理表領域からローカル管理表領域への移行

TABLESPACE_MIGRATE_TO_LOCAL プロシージャを使用して、ディクショナリ管理表領域をローカル管理表領域に移行する場合。この操作はオンラインで実行されますが、領域管理操作は移行が完了するまでブロックされます。つまり、移行処理中にデータの読取りや変更はできますが、大量のデータをロードする場合は追加のエクステントの割当てが必要になるため、操作がブロックされる場合があります。

データベースのブロック・サイズは 2KB、表領域 tbs_1 の既存のエクステント・サイズは 10、50 および 10,000 ブロック（それぞれ使用済、使用済および使用可能）とします。MINIMUM EXTENT 値は 20KB（10 ブロック）です。システムにビットマップの割当て単位を選択させることができます。MINIMUM EXTENT を超えない範囲の最大公分母であることから、10 ブロックの値が選択されます。

tbs_1 をローカル管理表領域に変換する文は、次のとおりです。

```
EXEC DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL ('tbs_1');
```

割当て単位のサイズを指定する場合は、必ずシステムによって計算される単位サイズの因数にします。

ローカル管理表領域への SYSTEM 表領域の移行

DBMS_SPACE_ADMIN パッケージを使用して、SYSTEM 表領域をディクショナリ管理からローカル管理に移行します。次の文は、この移行を実行します。

```
SQL> EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL('SYSTEM');
```

移行を実行する前に、次の条件を満たす必要があります。

- データベースのデフォルト一時表領域が SYSTEM ではないこと。
- ディクショナリ管理表領域にロールバック・セグメントがないこと。
- ローカル管理表領域に 1 つ以上のオンライン・ロールバック・セグメントがあるか、自動 UNDO 管理を使用している場合は、UNDO 表領域がオンラインになっていること。
- UNDO 領域を含む表領域（つまり、ロールバック・セグメントを含む表領域または UNDO 表領域）を除き、すべての表領域が読取り専用モードになっていること。
- システムが制限モードになっていること。
- データベースのコールド・バックアップがあること。

コールド・バックアップを除き、前述のすべての条件は TABLESPACE_MIGRATE_TO_LOCAL プロシージャにより施行されます。

注意： SYSTEM 表領域をローカル管理に移行すると、データベース内のディクショナリ管理表領域を読取り / 書込み用にできなくなります。ディクショナリ管理表領域を読取り / 書込みモードで使用できるようにする必要がある場合は、これらの表領域をローカル管理に移行してから、SYSTEM 表領域を移行することをお薦めします。

データベース間での表領域のトランスポート

ここでは、データベース間で表領域をトランスポートする方法について説明します。この項の内容は、次のとおりです。

注意： トランスポータブル表領域セットを生成するには、Oracle8*i* 以上の Enterprise Edition を使用する必要があります。ただし、Oracle8*i* 以上であれば、どのエディションでもトランスポータブル表領域セットを同じプラットフォーム上の Oracle Database にインポートできます。トランスポータブル表領域セットを別のプラットフォーム上の Oracle Database にインポートするには、両方のデータベースの互換性が 10.0 以上に設定されている必要があります。表領域の移動に伴うリリース・レベルでのデータベースの互換性の詳細は、12-34 ページの「トランスポータブル表領域の互換性に関する注意事項」を参照してください。

トランスポータブル表領域の概要

トランスポータブル表領域機能を使用すると、表領域セットを別の Oracle Database にコピーできます。

注意： この方法で表領域をトランスポートする場合は、トランスポート処理が完了するまで、トランスポート対象の表領域を読取り専用モードにする必要があります。これが望ましくない場合は、トランスポータブル表領域をバックアップ機能から使用できます。詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

ディクショナリ管理またはローカル管理のどちらの表領域でもトランスポートできます。Oracle9*i* から、トランスポートする表領域をターゲット・データベースの標準ブロック・サイズと同じブロック・サイズにする必要がなくなりました。

同じデータのエクスポート / インポートやアンロード / ロードを使用するよりも、トランスポータブル表領域を使用してデータを移動するほうが高速です。これは、実際のすべてのデータを含むデータファイルが単に移動先にコピーされるためです。表領域オブジェクトのメタデータのみを新規データベースにトランスポートするには、データ・ポンプを使用します。

注意： Oracle Database 11*g* リリース 1 からは、トランスポータブル表領域に対してデータ・ポンプを使用する必要があります。XMLType データをデータベース・バージョン 10*g* リリース 2 以前に下位移行できるのは、元のインポート・ユーティリティ（IMP）とエクスポート・ユーティリティ（EXP）を使用できる環境がある場合のみです。これらのユーティリティの詳細は『Oracle Database ユーティリティ』を、XMLType の詳細は『Oracle XML DB 開発者ガイド』を参照してください。

トランスポータブル表領域機能は、次のような場合に役立ちます。

- データ・ウェアハウス表のパーティションをエクスポートおよびインポートする場合
- 構造化データを CD で公開する場合
- 読取り専用バージョンの複数の表領域を複数のデータベースにコピーする場合
- 履歴データをアーカイブする場合
- 表領域の Point-in-Time リカバリ（TSPITR）を実行する場合

これらの使用例については、12-41 ページの「トランスポータブル表領域の使用 : 使用例」を参照してください。

表領域をトランスポートする方法は、次の 2 通りあります。

- 手動。この項で説明する手順に従います。SQL*Plus、Recovery Manager およびデータ・ポンプに対するコマンドの発行が含まれます。
- Enterprise Manager の「表領域のトランスポート」ウィザードの使用。

  「表領域のトランスポート」ウィザードを実行する手順は、次のとおりです。

  1. `EXP_FULL_DATABASE` ロールを持つユーザーで Enterprise Manager にログインします。
  2. **「メンテナンス」**リンクをクリックして、「メンテナンス」タブを表示します。
  3. ヘッダー「データベース・ファイルの移動」の下の**「表領域のトランスポート」**をクリックします。

     関連項目： データ・ウェアハウス環境でのトランスポータブル表領域の使用方法は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。

プラットフォーム間での表領域のトランスポート

Oracle Database 11*g* から、プラットフォーム間で表領域のトランスポートが可能になりました。この機能を使用すると、次のことが可能になります。

- データベースをプラットフォーム間で移行できます。
- コンテンツ・プロバイダは、簡単にかつ効率的に構造化データを公開し、別のプラットフォームで Oracle Database を実行している顧客に配布できます。
- データ・ウェアハウス環境からデータ・マート（多くの場合、小規模プラットフォームで実行されている）へのデータの配布を簡素化できます。
- 異なるオペレーティング・システムまたはプラットフォーム上の Oracle Database インストール間で読取り専用表領域を共有できます。この場合、次の各項で説明するように、それらのプラットフォームおよび endianness が同じプラットフォームからストレージ・システムにアクセスできることが前提となります。

多くのプラットフォーム（すべてではありません）では、クロス・プラットフォームでの表領域のトランスポートがサポートされます。`V$TRANSPORTABLE_PLATFORM` ビューを問い合せると、サポートされているプラットフォームを参照して、各プラットフォームの endian 形式（バイトの並び順）を確認できます。次の問合せを実行すると、プラットフォーム間での表領域のトランスポートがサポートされているプラットフォームが表示されます。

```
SQL> COLUMN PLATFORM_NAME FORMAT A32
SQL> SELECT * FROM V$TRANSPORTABLE_PLATFORM;

PLATFORM_ID PLATFORM_NAME                    ENDIAN_FORMAT
----------- -------------------------------- --------------
          1 Solaris[tm] OE (32-bit)          Big
          2 Solaris[tm] OE (64-bit)          Big
          7 Microsoft Windows IA (32-bit)    Little
         10 Linux IA (32-bit)                Little
          6 AIX-Based Systems (64-bit)       Big
          3 HP-UX (64-bit)                   Big
          5 HP Tru64 UNIX                    Little
          4 HP-UX IA (64-bit)                Big
         11 Linux IA (64-bit)                Little
         15 HP Open VMS                      Little
          8 Microsoft Windows IA (64-bit)    Little
          9 IBM zSeries Based Linux          Big
         13 Linux 64-bit for AMD             Little
         16 Apple Mac OS                     Big
         12 Microsoft Windows 64-bit for AMD Little
         17 Solaris Operating System (x86)   Little

16 rows selected.
```

ソースおよびターゲットのプラットフォームで endianness が異なる場合は、ソースまたはターゲットのプラットフォームで、トランスポートする表領域をターゲットの形式に変換する必要があります。両方のプラットフォームで endianness が同じ場合は変換する必要はなく、表領域は同じプラットフォーム上にある場合と同様にトランスポートできます。

表領域を別のプラットフォームにトランスポートする前に、その表領域が属するプラットフォームをデータファイル・ヘッダーで識別する必要があります。互換性が 10.0.0 以上に設定されている Oracle Database では、データファイルの読取りまたは書込みを 1 回以上実行することによってプラットフォームを識別できます。

トランスポータブル表領域の使用に関する制限事項

表領域をトランスポートする場合は、次の制限事項に注意してください。

- ソース・データベースとターゲット・データベースでは、同じキャラクタ・セットおよび各国語キャラクタ・セットを使用する必要があります。
- 同じ名前を持つ表領域がすでに存在しているターゲット・データベースには、表領域をトランスポートできません。ただし、トランスポート操作を実行する前に、トランスポートする表領域またはトランスポート先の表領域のいずれかの名前を変更できます。
- 基礎となるオブジェクトを持つオブジェクト（マテリアライズド・ビューなど）またはオブジェクトを含むオブジェクト（パーティション表など）は、それらのオブジェクトがすべて表領域セットに含まれている場合のみトランスポートできます。
- 暗号化された表領域には、次の制限事項があります。
  - 暗号化された表領域をトランスポートする前に、Oracle ウォレットを宛先データベースに手動でコピーする必要があります。ただし、マスター暗号化キーが Oracle ウォレットではなく、ハードウェア・セキュリティ・モジュール（HSM）デバイスに格納されている場合を除きます。コピーしたウォレットのパスワードは、宛先データベースでも同じになります。ただし、宛先データベースでパスワードを変更して、各データベースに独自のウォレット・パスワードを設定することをお薦めします。HSM デバイス、Oracle ウォレットの位置の決定、および Oracle Wallet Manager を使用したウォレット・パスワードの変更の詳細は、『Oracle Database Advanced Security 管理者ガイド』を参照してください。
  - 透過的データ暗号化用の Oracle ウォレットがすでに存在するデータベースには、暗号化された表領域をトランスポートできません。この場合は、Oracle Data Pump を使用して表領域のスキーマ・オブジェクトをエクスポートし、宛先データベースにイン

ポートする必要があります。必要に応じて、Oracle Data Pump 機能を利用してデータのエクスポートおよびインポート時にデータの暗号化を保持できます。詳細は、『Oracle Database ユーティリティ』を参照してください。

  - 暗号化された表領域は、endianness が異なるプラットフォームにはトランスポートできません。

- ブロックの暗号化を使用していない表領域であっても、暗号化列のある表が含まれている場合はトランスポートできません。Oracle Data Pump を使用して、表領域のスキーマ・オブジェクトをエクスポートおよびインポートする必要があります。Oracle Data Pump 機能を利用すると、データのエクスポートおよびインポート時にデータの暗号化を保持できます。詳細は、『Oracle Database ユーティリティ』を参照してください。
- Oracle Database 10*g* リリース 2 からは、XMLType を含む表領域をトランスポートできます。Oracle Database 11*g* リリース 1 からは、XMLType を含む表領域の表領域メタデータをエクスポートおよびインポートする場合は、データ・ポンプを使用する必要があります。

  次の問合せでは、XMLType を含む表領域のリストが返されます。

```
select distinct p.tablespace_name from dba_tablespaces p,
  dba_xml_tables x, dba_users u, all_all_tables t where
  t.table_name=x.table_name and t.tablespace_name=p.tablespace_name
  and x.owner=u.username
```

  XMLType の詳細は、『Oracle XML DB 開発者ガイド』を参照してください。

  XMLType を含む表領域のトランスポートには、次の制限事項があります。

  - ターゲット・データベースに XML DB がインストールされている必要があります。
  - XMLType 表が参照するスキーマを XML DB 標準スキーマにすることはできません。
  - XMLType 表が参照するスキーマには循環依存性を設定できません。
  - 行レベルでセキュリティが設定された XMLType 表は、エクスポートまたはインポートできないためサポートされていません。
  - トランスポートされた XMLType 表のスキーマがターゲット・データベース内に存在しない場合は、スキーマがインポートおよび登録されます。ターゲット・データベースにスキーマが存在する場合は、`ignore=y` オプションを設定しないと、エラーが返されます。
  - XMLType 表が別のスキーマに依存するスキーマを使用している場合、依存しているスキーマはエクスポートされません。そのスキーマがターゲット・データベースにすでに存在する場合のみ、インポートが成功します。

他の制限事項は次のとおりです。

アドバンスト・キュー　トランスポータブル表領域は、複数受信者を持つ 8.0 互換のアドバンスト・キューをサポートしません。

SYSTEM 表領域オブジェクト　SYSTEM 表領域またはユーザー SYS が所有するオブジェクトはトランスポートできません。これに該当するオブジェクトは、PL/SQL、Java クラス、コールアウト、ビュー、シノニム、ユーザー、権限、ディメンション、ディレクトリ、順序などです。

OPAQUE 型　解釈がアプリケーション固有で、データベースに対して不透明なタイプ（`RAW`、`BFILE` など）は、トランスポートできますが、クロス・プラットフォームのトランスポート操作では変換されません。このタイプの実際の構造はアプリケーションのみが認識するため、このタイプが新規プラットフォームに移動した後、アプリケーションでは endianness の問題に対処する必要があります。OPAQUE 型を使用するタイプとオブジェクトも、直接的または間接的にこの制限の影響を受けます。

浮動小数点数　`BINARY_FLOAT` 型および `BINARY_DOUBLE` 型は、データ・ポンプを使用してトランスポートできます。

トランスポータブル表領域の互換性に関する注意事項

Oracle Database では、トランスポータブル表領域セットを作成するときに、ターゲット・データベースで実行する必要のある最も低い互換性レベルが計算されます。このレベルのことを、トランスポータブル・セットの互換性レベルと呼びます。Oracle Database 11*g* からは、ターゲット・データベースが同じプラットフォーム上にあるか別のプラットフォーム上にあるかに関係なく、表領域は同等以上の互換性が設定されたデータベースに常にトランスポートできます。トランスポータブル・セットの互換性レベルがターゲット・データベースの互換性レベルよりも高い場合は、エラーが通知されます。

次の表に、様々な使用例でのソース表領域とターゲット表領域の互換性の最低要件を示します。ソース・データベースとターゲット・データベースの互換性設定は同一である必要はありません。

表 12-1 互換性の最低要件

	互換性の最低設定	
トランスポートの使用例	**ソース・データベース**	**ターゲット・データベース**
同じプラットフォーム上のデータベース間	8.0	8.0
データベース・ブロック・サイズがターゲット・データベースと異なる表領域	9.0	9.0
異なるプラットフォーム上のデータベース間	10.0	10.0

データベース間で表領域をトランスポートする手順および例

次に、表領域のトランスポート処理の手順を示します。各手順の詳細は、後続の例で示します。

1. V$TRANSPORTABLE_PLATFORM ビューを問い合せて両方のプラットフォームの endian 形式をチェック（クロス・プラットフォームでのトランスポートの場合）

   同じプラットフォームに表領域セットをトランスポートする場合は、この手順をスキップしてください。

2. 自己完結型の表領域セットの選択

3. トランスポータブル表領域セットの生成

   トランスポータブル表領域セット（**トランスポータブル・セット**）は、トランスポートされる表領域セットのデータファイルと、そのセットの構造情報（メタデータ）を含むエクスポート・ファイルから構成されます。データ・ポンプを使用してエクスポートを実行します。

   endianness がソース・プラットフォームとは異なるプラットフォームに表領域セットをトランスポートする場合は、表領域セットをターゲット・プラットフォームの endianness に変換する必要があります。ソース側で変換する場合はこの手順の中で実行し、ターゲット側で変換する場合は手順 4 で実行できます。

   > **注意：** この方法でトランスポータブル表領域を生成する場合は、表領域を一時的に読取り専用にする必要があります。この方法が望ましくない場合は、代替方法として、トランスポータブル表領域をバックアップ機能から使用できます。詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

4. 表領域セットのトランスポート

   データファイルとエクスポート・ファイルをターゲット・データベースからアクセス可能な場所にコピーします。

   endianness がソース・プラットフォームとは異なるプラットフォームに表領域セットをトランスポートし、ソース側で表領域セットをターゲット・プラットフォームの endianness に変換していない場合は、ここでターゲット側での変換を実行する必要があります。

5. 表領域セットのインポート

   データ・ポンプ・ユーティリティを起動して、表領域セットのメタデータをターゲット・データベースにインポートします。

例

表領域をトランスポートする手順については、次のデータファイルと表領域を想定した例で詳細に説明します。

表領域	データファイル
sales_1	/u01/oracle/oradata/salesdb/sales_101.dbf
sales_2	/u01/oracle/oradata/salesdb/sales_201.dbf

手順 1: プラットフォームのサポートと endianness の確認

この手順は、ソース・プラットフォームとは異なるプラットフォームに表領域セットをトランスポートする場合のみ実行します。

ソース・プラットフォームとは異なるプラットフォームに表領域セットをトランスポートする場合は、ソースおよびターゲットのプラットフォームの両方でプラットフォーム間の表領域トランスポートがサポートされているかどうかと、各プラットフォームの endianness を確認します。両方のプラットフォームの endianness が同じ場合、変換は必要ありません。endianness が異なる場合は、ソース・データベースまたはターゲット・データベースのいずれかで、表領域セットを変換する必要があります。

sales_1 および sales_2 を異なるプラットフォームにトランスポートする場合は、各プラットフォームで次の問合せを実行できます。問合せで行が返される場合、そのプラットフォームではプラットフォーム間の表領域トランスポートがサポートされています。

```
SELECT d.PLATFORM_NAME, ENDIAN_FORMAT
     FROM V$TRANSPORTABLE_PLATFORM tp, V$DATABASE d
     WHERE tp.PLATFORM_NAME = d.PLATFORM_NAME;
```

ソース・プラットフォームでの問合せの結果は次のとおりです。

```
PLATFORM_NAME              ENDIAN_FORMAT
-------------------------- --------------
Solaris[tm] OE (32-bit)    Big
```

ターゲット・プラットフォームでの問合せの結果は次のとおりです。

```
PLATFORM_NAME              ENDIAN_FORMAT
-------------------------- --------------
Microsoft Windows NT       Little
```

この問合せ結果によって、endian 形式が異なっているため、表領域セットをトランスポートする場合に変換が必要であることがわかります。

手順 2: 自己完結した表領域セットの選択

トランスポータブル・セット内のオブジェクトとセット外のオブジェクトとの間に、論理的または物理的な依存関係が存在することがあります。トランスポートできるのは、自己完結した表領域セットのみです。この場合、自己完結とは、表領域セット内から外部への参照がないことを意味します。次に、自己完結した表領域に違反する例を示します。

- 表領域セット内に、そのセットに含まれない表に関する索引が含まれている場合

> **注意：** 表に対応する索引が表領域セットの外部にある場合は、違反になりません。

- パーティション表の一部が表領域セットに含まれている場合

  コピーする表領域セットは、パーティション化した表のすべてのパーティションが含まれている状態、またはまったく含まれていない状態にしてください。パーティション表のサブセットをトランスポートする場合は、パーティションを表に変換する必要があります。

- 参照整合性制約がセット境界を越えて別の表を指している場合

  表領域セットをトランスポートするときには、参照整合性制約を含めるかどうかを選択できます。ただし、参照整合性制約を含めることによって、表領域セットの自己完結性に影響を与える場合があります。制約をトランスポートしなければ、その制約はポインタとは見なされません。

- 表領域セット内の表に、そのセットに含まれない LOB を指す LOB 列が含まれている場合

- ユーザー A が登録された XML DB スキーマ（*.xsd）にユーザー B が登録されたグローバル・スキーマをインポートする際、ユーザー A のデフォルト表領域が表領域 A、ユーザー B のデフォルト表領域が表領域 B で、表領域 A のみが表領域セットに含まれている場合

表領域セットが自己完結型かどうかを判断するには、オラクル社が提供するパッケージ DBMS_TTS の TRANSPORT_SET_CHECK プロシージャをコールします。このプロシージャを実行するには、EXECUTE_CATALOG_ROLE ロール（最初は SYS に付与されている）を付与されている必要があります。

DBMS_TTS パッケージをコールするときは、自己完結かどうかを調べるトランスポータブル・セットの表領域のリストを指定します。制約を含むかどうかを指定することもできます。厳密または完全な完結であるかを調べる場合は、TTS_FULL_CHECK パラメータを TRUE に設定する必要があります。

厳密または完全な完結のチェックは、トランスポータブル・セットから外部への参照のみではなく、外部からトランスポータブル・セットへの参照も捕捉する必要がある場合に実行します。依存オブジェクトがトランスポータブル・セットに完全に含まれているか、またはトランスポータブル・セットの外部にのみ存在することが必要な場合は、TSPITR を実行します。

たとえば、表 t を含んでいるが、その索引 i を含んでいない表領域に対して TSPITR を実行すると、トランスポート後に索引とデータの整合性がなくなるため、これは違反になります。完全完結チェックを実行することにより、トランスポータブル・セットからの依存関係またはトランスポータブル・セットへの依存関係がないことが保証されます。詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』の TSPITR の例を参照してください。

注意： デフォルトでは、トランスポータブル表領域は、完全完結しているかどうかではなく自己完結しているかどうかがチェックされます。

次の文を使用して、表領域 sales_1 および sales_2 が自己完結しているかどうかを、参照整合性制約を考慮して（TRUE を指定して）調べます。

```
EXECUTE DBMS_TTS.TRANSPORT_SET_CHECK('sales_1,sales_2', TRUE);
```

この PL/SQL パッケージをコールした後に、TRANSPORT_SET_VIOLATIONS ビューからすべての違反を選択して表示できます。表領域セットが自己完結している場合、このビューは空になります。次の問合せ例は、2 つの違反がある場合を示しています。1 つ目は表領域セットの境界を超えている外部キー定数 dept_fk で、2 つ目は表領域セットに部分的に含まれているパーティション表 jim.sales です。

```
SQL> SELECT * FROM TRANSPORT_SET_VIOLATIONS;

VIOLATIONS
--------------------------------------------------------------------------------
Constraint DEPT_FK between table JIM.EMP in tablespace SALES_1 and table
JIM.DEPT in tablespace OTHER
Partitioned table JIM.SALES is partially contained in the transportable set
```

これらの違反は、sales_1 および sales_2 をトランスポータブルにする前に解決する必要があります。次の手順で説明するように、整合性制約違反を回避するための選択肢の 1 つとして、整合性制約をエクスポートしない方法があります。

関連項目：

- DBMS_TTS パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- TSPITR における DBMS_TTS パッケージの使用の詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

手順 3: トランスポータブル表領域セットの生成

この手順は、権限を持つユーザーが実行できます。ただし、トランスポータブル表領域のエクスポート操作を実行するためには、EXP_FULL_DATABASE ロールが割り当てられている必要があります。

注意： この方法でトランスポータブル表領域を生成する場合は、表領域を一時的に読取り専用にする必要があります。この方法が望ましくない場合は、代替方法として、トランスポータブル表領域をバックアップ機能から使用できます。詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

トランスポートする表領域セットが自己完結であることを確認した後に、次の処理を実行してトランスポータブル表領域セットを生成します。

1. コピーするセット内のすべての表領域を読取り専用にします。

```
SQL> ALTER TABLESPACE sales_1 READ ONLY;

Tablespace altered.

SQL> ALTER TABLESPACE sales_2 READ ONLY;

Tablespace altered.
```

2. ホスト・システムでデータ・ポンプ・エクスポート・ユーティリティを起動して、トランスポータブル・セットに含める表領域を指定します。

```
SQL> HOST

$ EXPDP system/password DUMPFILE=expdat.dmp DIRECTORY=dpump_dir
       TRANSPORT_TABLESPACES = sales_1,sales_2
```

エクスポート操作のモードを決定する TRANSPORT_TABLESPACES は、常に指定する必要があります。この例では、次のように指定しています。

- DUMPFILE パラメータでは、作成する構造情報エクスポート・ファイルの名前を expdat.dmp と指定します。
- DIRECTORY パラメータでは、オペレーティング・システムまたは自動ストレージ管理のダンプ・ファイルの場所を示すデフォルトのディレクトリ・オブジェクトを指定します。DIRECTORY オブジェクトはデータ・ポンプを起動する前に作成し、ディレクトリに対する READ および WRITE オブジェクト権限を PUBLIC に付与する必要があります。CREATE DIRECTORY コマンドの詳細は、『Oracle Database SQL リファレンス』を参照してください。
- トリガーと索引は、デフォルトでエクスポート操作に含まれています。

表領域のトランスポート操作を厳密完結チェック付きで実行する場合は、次の例に示すように、TRANSPORT_FULL_CHECK パラメータを使用します。

```
EXPDP system/password DUMPFILE=expdat.dmp DIRECTORY = dpump_dir
      TRANSPORT_TABLESPACES=sales_1,sales_2 TRANSPORT_FULL_CHECK=Y
```

この例では、データ・ポンプ・エクスポート・ユーティリティによって、トランスポータブル・セット内のオブジェクトとトランスポータブル・セット外のオブジェクトとの間に依存性がないことを検証します。トランスポートする表領域セットが自己完結していない場合、エクスポートは失敗し、トランスポータブル・セットが自己完結していないことがわかります。その場合は、手順 1 に戻ってすべての違反を解決する必要があります。

注意： データ・ポンプ・ユーティリティを使用してエクスポートするのは、表領域のデータ・ディクショナリの構造情報（メタデータ）のみです。実際のデータはアンロードされないため、この操作は大規模な表領域セットの場合でも比較的早く完了します。

---

3. 完了した後、終了して SQL*Plus に戻ります。

```
$ EXIT
```

関連項目： データ・ポンプ・ユーティリティの使用方法は、『Oracle Database ユーティリティ』を参照してください。

sales_1 および sales_2 表領域を異なるプラットフォームにトランスポートする場合に、各プラットフォームの endianness が異なるため、表領域セットをトランスポートする前に変換を行う場合は、sales_1 および sales_2 を構成するデータファイルを変換します。

4. SQL*Plus からホスト・システムに戻ります。

```
SQL> HOST
```

5. Recovery Manager の CONVERT コマンドを使用して変換を実行します。Recovery Manager を起動し、ターゲット・データベースに接続します。

```
$ RMAN TARGET /

Recovery Manager: Release 10.1.0.0.0

Copyright (c) 1995, 2003, Oracle Corporation.  All rights reserved.

connected to target database: salesdb (DBID=3295731590)
```

6. データファイルを変換して、ソース・プラットフォーム上の一時的な場所に格納します。この例では、一時的な場所はディレクトリ /temp で、このディレクトリはすでに作成されています。変換されたデータファイルの名前は、システムによって割り当てられます。

```
RMAN> CONVERT TABLESPACE sales_1,sales_2
2> TO PLATFORM 'Microsoft Windows NT'
3> FORMAT '/temp/%U';

Starting backup at 08-APR-03
using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=11 devtype=DISK
channel ORA_DISK_1: starting datafile conversion
input datafile fno=00005 name=/u01/oracle/oradata/salesdb/sales_101.dbf
converted datafile=/temp/data_D-10_I-3295731590_TS-ADMIN_TBS_FNO-5_05ek24v5
channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:00:15
channel ORA_DISK_1: starting datafile conversion
input datafile fno=00004 name=/u01/oracle/oradata/salesdb/sales_101.dbf
converted datafile=/temp/data_D-10_I-3295731590_TS-EXAMPLE_FNO-4_06ek24vl
```

```
channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:00:45
Finished backup at 08-APR-03
```

関連項目： Recovery Manager の CONVERT コマンドの詳細は、『Oracle Database バックアップおよびリカバリ・リファレンス』を参照してください。

7. Recovery Manager を終了します。

```
RMAN> exit
Recovery Manager complete.
```

手順 4: 表領域セットのトランスポート

表領域のデータファイルとエクスポート・ファイルの両方を、ターゲット・データベースからアクセスできる場所にトランスポートします。

ソースとターゲットの両方がファイル・システムの場合は、次の機能を使用できます。

- フラット・ファイルをコピーする機能（オペレーティング・システムのコピー・ユーティリティや FTP など）
- DBMS_FILE_TRANSFER パッケージ
- Recovery Manager
- CD で配布する機能

ソースとターゲットのいずれかが自動ストレージ管理（ASM）ディスク・グループの場合は、次の機能を使用できます。

- XML DB リポジトリの /sys/asm 仮想フォルダ間での FTP

  詳細は、『Oracle Database ストレージ管理者ガイド』を参照してください。
- DBMS_FILE_TRANSFER パッケージ
- Recovery Manager

注意： UNIX dd ユーティリティを使用してデータベース間で RAW デバイス・ファイルをコピーする場合は、注意が必要です。dd ユーティリティは、ソース RAW デバイス・ファイル全体をコピーするために使用するか、またはソース RAW デバイス・ファイルの特定の範囲のみをコピーするようにオプションを指定して起動できます。

ただし、データファイルには非表示の制御情報も格納されているため、RAW デバイス・ファイルの実際のデータファイル・サイズを確認するのは困難です。したがって、dd ユーティリティを使用する場合は、ソース RAW デバイス・ファイルの内容全体をコピーすることをお薦めします。

endianness がソース・プラットフォームとは異なるプラットフォームに表領域セットをトランスポートし、表領域セットをまだ変換していない場合は、この手順を実行します。この例では、トランスポートする前に次の手順を完了していると想定しています。

1. トランスポートするソース表領域を読取り専用に設定する。
2. エクスポート・ユーティリティを使用して、エクスポート・ファイル（後述の例では expdat.dmp）を作成する。

ターゲット・プラットフォーム上で変換するデータファイルは、ターゲット・プラットフォーム上の一時的な場所に移動できます。ただし、変換前か変換後かに関係なく、すべてのデータファイルをターゲット・プラットフォーム上の指定の場所に移動する必要があります。

ここで、次の例に示すように、Recovery Manager を使用してトランスポート済の必要なデータファイルを移動先ホストの endian 形式に変換し、その結果を /orahome/dbs に格納します。

```
RMAN> CONVERT DATAFILE
2> '/hq/finance/work/tru/tbs_31.f',
3> '/hq/finance/work/tru/tbs_32.f',
4> '/hq/finance/work/tru/tbs_41.f'
5> TO PLATFORM="Solaris[tm] OE (32-bit)"
6> FROM PLATFORM="HP TRu64 UNIX"
7> DB_FILE_NAME_CONVERT=
8> "/hq/finance/work/tru/", "/hq/finance/dbs/tru"
9> PARALLELISM=5;
```

データファイルは、表領域名ではなくファイル名で識別します。表領域のメタデータがインポートされるまで、ローカル・インスタンスでは対象の表領域名を認識できません。ソース・プラットフォームとターゲット・プラットフォームはオプションです。Recovery Manager では、データファイルを調べてソース・プラットフォームを判別します。また、デフォルトのターゲット・プラットフォームは、変換を実行するホストのプラットフォームになります。

> **関連項目：** DBMS_FILE_TRANSFER パッケージを使用して、トランスポートされるファイルとそのメタデータをコピーする方法は、13-12 ページの「データベース・サーバーを使用したファイルのコピー」を参照してください。

手順 5: 表領域セットのインポート

> **注意：** 表領域セットを受け取るデータベースの標準ブロック・サイズと異なるブロック・サイズの表領域をトランスポートする場合は、最初に DB_*n*K_CACHE_SIZE 初期化パラメータ・エントリを受取り側データベースのパラメータ・ファイル内に設定する必要があります。
>
> たとえば、ブロック・サイズが 8KB の表領域を標準ブロック・サイズが 4KB のデータベースにトランスポートする場合は、DB_8K_CACHE_SIZE 初期化パラメータ・エントリをパラメータ・ファイルに含める必要があります。このエントリがすでにパラメータ・ファイルに含まれている場合は、ALTER SYSTEM SET 文を使用してこのパラメータを設定できます。
>
> DB_*n*K_CACHE_SIZE 初期化パラメータの値の指定方法は、『Oracle Database リファレンス』を参照してください。

この手順は、権限を持つユーザーが実行できます。表領域セットをインポートするには、次のタスクを実行します。

1. データ・ポンプ・インポート・ユーティリティの impdp を使用して、次のように表領域のメタデータをインポートします。

   ```
   IMPDP system/password DUMPFILE=expdat.dmp DIRECTORY=dpump_dir
      TRANSPORT_DATAFILES=
      /salesdb/sales_101.dbf,
      /salesdb/sales_201.dbf
      REMAP_SCHEMA=(dcranney:smith) REMAP_SCHEMA=(jfee:williams)
   ```

   この例では、次の指定を行います。

   - DUMPFILE パラメータでは、インポートされる表領域のメタデータが含まれるエクスポート・ファイルを指定します。
   - DIRECTORY パラメータでは、ダンプ・ファイルの場所を識別するディレクトリ・オブジェクトを指定します。
   - TRANSPORT_DATAFILES パラメータによって、インポートする表領域が含まれるすべてのデータファイルを識別します。
   - REMAP_SCHEMA パラメータによって、データベース・オブジェクトの所有権を変更します。REMAP_SCHEMA を指定しない場合、すべてのデータベース・オブジェクト（表や索引など）は同じユーザー・スキーマ内でソース・データベースとして作成され、

そのユーザーはターゲット・データベース内にすでに存在している必要があります。ユーザーが存在しない場合は、インポート・ユーティリティによってエラーが返されます。この例では、ソース・データベース内で dcranney が所有している表領域セット内のオブジェクトは、表領域セットをインポートした後のターゲット・データベース内では smith の所有となります。同様に、ソース・データベース内で jfee が所有しているオブジェクトは、ターゲット・データベース内では williams の所有となります。この場合、ターゲット・データベースにユーザー dcranney および jfee は存在しなくてもかまいませんが、ユーザー smith および williams は存在する必要があります。

この文が正常に実行されると、コピーするセット内のすべての表領域は読取り専用モードのままになります。インポート・ログをチェックして、エラーが発生しなかったかどうかを確認してください。

多数のデータファイルを扱う場合、データファイル名のリストを文の行で指定することは煩雑です。また、文の行制限を超える場合もあります。このような場合には、インポート・パラメータ・ファイルを使用できます。たとえば、次のようにしてデータ・ポンプ・インポート・ユーティリティを起動できます。

```
IMPDP system/password PARFILE='par.f'
```

パラメータ・ファイル par.f の内容は、次のとおりです。

```
DIRECTORY=dpump_dir
DUMPFILE=expdat.dmp
TRANSPORT_DATAFILES="'/db/sales_jan','/db/sales_feb'"
REMAP_SCHEMA=dcranney:smith
REMAP_SCHEMA=jfee:williams
```

関連項目： インポート・ユーティリティの使用方法は、『Oracle Database ユーティリティ』を参照してください。

2. 必要に応じて、次のように表領域を読取り / 書込みモードに戻します。

```
ALTER TABLESPACE sales_1 READ WRITE;
ALTER TABLESPACE sales_2 READ WRITE;
```

トランスポータブル表領域の使用 : 使用例

次の各項では、トランスポータブル表領域の用途について説明します。

- データ・ウェアハウスのためのパーティションのトランスポートと連結
- 構造化データの CD での公開
- 複数データベースで同じ表領域を読取り専用でマウントする方法
- トランスポータブル表領域を使用した履歴データのアーカイブ
- トランスポータブル表領域を使用した TSPITR の実行

データ・ウェアハウスのためのパーティションのトランスポートと連結

標準的な企業のデータ・ウェアハウスには、1 つ以上の大きいファクト表が含まれています。これらのファクト表は、企業データ・ウェアハウスを履歴データベースにするために、日付別にパーティション化されていることがあります。この場合、索引を作成すると、スター・クエリーを高速化できます。履歴データベースから最も古いパーティションを削除するたびにグローバル索引を再作成しなくても済むように、この種の履歴パーティション表についてローカル索引を作成することをお薦めします。

たとえば、1 か月分のデータをデータ・ウェアハウスに毎月ロードする場合を考えます。データ・ウェアハウスには、sales という大型のファクト表があり、次の列が含まれています。

```
CREATE TABLE sales (invoice_no NUMBER,
   sale_year  INT NOT NULL,
   sale_month INT NOT NULL,
```

```
   sale_day   INT NOT NULL)
   PARTITION BY RANGE (sale_year, sale_month, sale_day)
     (partition jan98 VALUES LESS THAN (1998, 2, 1),
      partition feb98 VALUES LESS THAN (1998, 3, 1),
      partition mar98 VALUES LESS THAN (1998, 4, 1),
      partition apr98 VALUES LESS THAN (1998, 5, 1),
      partition may98 VALUES LESS THAN (1998, 6, 1),
      partition jun98 VALUES LESS THAN (1998, 7, 1));
```

次のようにして、ローカルの非同一キー索引を作成します。

```
CREATE INDEX sales_index ON sales(invoice_no) LOCAL;
```

最初は、すべてのパーティションは空で、同じデフォルト表領域にあります。パーティションを毎月 1 つ作成し、パーティション表 sales に連結する必要があります。

現在が 1998 年 7 月で、7 月分の売上データをパーティション表にロードするとします。ステージング・データベース内で、新しい表領域 ts_jul を作成します。また、その表領域内で、sales 表と正確に同じ列タイプを持つ表 jul_sales も作成します。表 jul_sales は、CREATE TABLE ... AS SELECT 文を使用して作成できます。jul_sales を作成して移入した後に、sales 表内のローカル索引と同じ列に索引を付けて、この表の索引 jul_sale_index を作成することもできます。索引の作成後に、表領域 ts_jul をデータ・ウェアハウスにトランスポートします。

データ・ウェアハウスでは、7 月分の売上データ用の sales 表にパーティションを追加します。これにより、ローカルの非同一キー索引用にも 1 つのパーティションが作成されます。

```
ALTER TABLE sales ADD PARTITION jul98 VALUES LESS THAN (1998, 8, 1);
```

トランスポートされた表 jul_sales を新しいパーティションに変換して、表 sales に連結します。

```
ALTER TABLE sales EXCHANGE PARTITION jul98 WITH TABLE jul_sales
   INCLUDING INDEXES
   WITHOUT VALIDATION;
```

この文により、新しいデータがパーティション表に連結され、7 月分の売上データが新しいパーティション jul98 に格納されます。また、索引 jul_sale_index が sales 表のローカル索引のパーティションに変換されます。この文では、構造情報を操作するだけであり、データベースのポインタを切り替えれば済むので、結果は即時に返されます。新しいパーティション内のデータが旧パーティション内のデータとオーバーラップしないことがわかっている場合は、WITHOUT VALIDATION 句を指定してください。この句を指定しないと、新しいパーティションの範囲を検証するために、そこに含まれる新しいデータがすべて検査されます。

sales 表のすべてのパーティションが同じステージング・データベースから取り込まれる場合（ステージング・データベースが破壊されることはありません）、変換文は常に成功します。ただし、一般に、パーティション表のデータが異なるデータベースから取り込まれる場合は、変換操作が失敗する可能性があります。たとえば、sales の jan98 パーティションが同じステージング・データベースから取り込まれていない場合は、前述の変換操作が失敗して次のエラーが返されることがあります。

```
ORA-19728: 表 JUL_SALES とパーティション JAN98( 表 SALES) 間で、データ・オブジェクト番号が競合しています。
```

この競合を解決するには、次の文を発行して、競合しているパーティションを移動します。

```
ALTER TABLE sales MOVE PARTITION jan98;
```

次に、変換操作を再試行してください。

交換が成功した後は、jul_sales と jul_sale_index を削除しても安全です（どちらも空になっています）。これで、7 月分の売上データはデータ・ウェアハウスに正常にロードされたことになります。

構造化データの CD での公開

トランスポータブル表領域を使用すると、構造化データを CD で公開できます。データ・プロバイダは、公開するデータを含む表領域をロードし、トランスポータブル・セットを生成し、トランスポータブル・セットを CD にコピーできます。これにより、この CD を配布できます。

顧客は、この CD を受け取って既存のデータベースに CD の内容を追加できます。CD からディスク記憶域にデータファイルをコピーする必要はありません。たとえば、Windows NT マシンの D ドライブが CD ドライブであるとします。次のようにして、データファイル catalog.f とエクスポート・ファイル expdat.dmp を含むトランスポータブル・セットをインポートできます。

```
IMPDP system/password DUMPFILE=expdat.dmp DIRECTORY=dpump_dir
   TRANSPORT_DATAFILES='D:¥catalog.f'
```

CD は、データベースの稼働中に取り出すことができます。この場合、その表領域への後続の問合せでは、CD 上のデータファイルをオープンできないことを示すエラーが返されます。ただし、このデータベースの他の部分への操作は影響を受けません。CD をドライブに戻すと、表領域は再び読込み可能になります。

CD を取り出すのは、読取り専用表領域のデータファイルを削除するのと同じことです。データベースを停止して再起動すると、削除されたデータファイルが見つからず、データベースをオープンできないことを示すメッセージが表示されます（初期化パラメータ READ_ONLY_OPEN_DELAYED を TRUE に設定していない場合）。READ_ONLY_OPEN_DELAYED が TRUE に設定されている場合は、トランスポートされた表領域を問い合せるときにのみ、このファイルが読み込まれます。したがって、CD から表領域をトランスポートする場合は、その CD がデータベースに永続的に連結されないかぎり、常に READ_ONLY_OPEN_DELAYED 初期化パラメータを TRUE に設定しておく必要があります。

複数データベースで同じ表領域を読取り専用でマウントする方法

トランスポータブル表領域を使用すると、複数のデータベースで 1 つの表領域を読取り専用でマウントできます。これにより、データを別々のディスクに複製しなくても、異なるデータベースで同じデータを共有できます。表領域のデータファイルは、どのデータベースからもアクセス可能にする必要があります。データベースの破損を回避するために、表領域はマウント先のすべてのデータベース内で読取り専用のままにしてください。

次に、複数のデータベースで同じ表領域を読取り専用でマウントする方法を 2 つ示します。

- 表領域の元のデータベースが表領域を共有するデータベースと異なる場合。

  ソース・データベースでトランスポータブル・セットを生成し、すべてのデータベースからアクセス可能なディスクにそのトランスポータブル・セットを格納し、その後、表領域をマウントする各データベースにメタデータをインポートします。

- 表領域が、その表領域を共有するデータベースの 1 つに属する場合。

  データファイルがすでに共有ディスクに格納されているとします。表領域がすでに格納されているデータベース上で、この表領域を読取り専用にしてトランスポータブル・セットを生成し、データファイルは共有ディスク上の同じ位置に残したまま、表領域を他のデータベースにインポートします。

ディスクを複数のコンピュータからアクセス可能にするには、いくつかの方法があります。クラスタ・ファイル・システムまたは RAW ディスクのいずれかを使用できます。ネットワーク・ファイル・システム（NFS）を使用することも可能です。ただし、NFS の停止中にユーザーが共有表領域を問い合せると、NFS 操作がタイムアウトになるまでデータベースが停止することがあります。

後で、一部のデータベースから読取り専用表領域を削除できます。読取り専用表領域を削除しても、表領域のデータファイルは変更されません。したがって、削除操作によって表領域が破損することはありません。表領域をマウントしているデータベースが 1 つしかない場合を除き、表領域は読取り / 書込み可能にしないでください。

トランスポータブル表領域を使用した履歴データのアーカイブ

トランスポータブル表領域セットは、任意の Oracle Database にインポートできる自己完結したファイル・セットであるため、この章で説明するトランスポータブル表領域の手順を使用して、企業データ・ウェアハウスに旧データまたは履歴データをアーカイブできます。

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

トランスポータブル表領域を使用した TSPITR の実行

トランスポータブル表領域を使用して、表領域の TSPITR を実行できます。

関連項目： トランスポータブル表領域を使用して TSPITR を実行する方法は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

データベースのトランスポータブル表領域を使用したプラットフォーム間の移動

トランスポータブル表領域機能を使用して異なるプラットフォーム間でデータベースを移動するには、移動先プラットフォームに新規データベースを作成して、すべてのユーザー表領域をトランスポートします。詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

SYSTEM 表領域はトランスポートできません。したがって、SYSTEM 表領域に依存する順序、PL/SQL パッケージなどのオブジェクトはトランスポートできません。このため、移動先のデータベースにこれらのオブジェクトを手動で作成するか、またはデータ・ポンプを使用して、トランスポータブル表領域で移動できないオブジェクトをトランスポートする必要があります。

表領域のデータ・ディクショナリ・ビュー

次のデータ・ディクショナリ・ビューおよび動的パフォーマンス・ビューは、データベースの表領域に関して役立つ情報を提供します。

ビュー	説明
V$TABLESPACE	制御ファイルに記述されているすべての表領域の名前と番号
V$ENCRYPTED_TABLESPACES	暗号化されたすべての表領域の名前と暗号化アルゴリズム
DBA_TABLESPACES、USER_TABLESPACES	すべての（またはユーザーがアクセス可能な）表領域の説明
DBA_TABLESPACE_GROUPS	表領域グループとそのグループに属する表領域
DBA_SEGMENTS、USER_SEGMENTS	すべての（またはユーザーがアクセス可能な）表領域内のセグメントに関する情報
DBA_EXTENTS、USER_EXTENTS	すべての（またはユーザーがアクセス可能な）表領域内のデータ・エクステントに関する情報
DBA_FREE_SPACE、USER_FREE_SPACE	すべての（またはユーザーがアクセス可能な）表領域内の使用可能エクステントに関する情報
DBA_TEMP_FREE_SPACE	各一時表領域の割当て済領域の合計と空き領域
V$DATAFILE	所有する表領域の表領域番号など、すべてのデータファイルに関する情報
V$TEMPFILE	所有する表領域の表領域番号など、すべての一時ファイルに関する情報
DBA_DATA_FILES	表領域に属するファイル（データファイル）
DBA_TEMP_FILES	一時表領域に属するファイル（一時ファイル）
V$TEMP_EXTENT_MAP	ローカル管理の一時表領域すべての全エクステントに関する情報

ビュー	説明
V$TEMP_EXTENT_POOL	ローカル管理の一時表領域の場合、キャッシュされ、各インスタンスで使用されている一時領域の状態
V$TEMP_SPACE_HEADER	各一時ファイルの使用済領域 / 空き領域
DBA_USERS	すべてのユーザーのデフォルト表領域と一時表領域
DBA_TS_QUOTAS	すべてのユーザーの表領域割当て制限
V$SORT_SEGMENT	特定インスタンス内のすべてのソート・セグメントに関する情報。このビューは、表領域が TEMPORARY タイプの場合にのみ更新されます。
V$TEMPSEG_USAGE	一時または永続表領域のユーザーに使用される一時（ソート）セグメントの説明

次に示すのは、これらのビューのごく一部の使用例です。

関連項目： これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。

例 1: 表領域とデフォルト記憶域パラメータの表示

データベースに含まれるすべての表領域の名前とデフォルト記憶域パラメータをすべて表示するには、DBA_TABLESPACES ビューに対して次の問合せを使用します。

```
SELECT TABLESPACE_NAME "TABLESPACE",
   INITIAL_EXTENT "INITIAL_EXT",
   NEXT_EXTENT "NEXT_EXT",
   MIN_EXTENTS "MIN_EXT",
   MAX_EXTENTS "MAX_EXT",
   PCT_INCREASE
   FROM DBA_TABLESPACES;

TABLESPACE  INITIAL_EXT  NEXT_EXT  MIN_EXT   MAX_EXT    PCT_INCREASE
----------  -----------  --------  -------   -------    ------------
RBS             1048576   1048576        2        40               0
SYSTEM           106496    106496        1        99               1
TEMP             106496    106496        1        99               0
TESTTBS           57344     16384        2        10               1
USERS             57344     57344        1        99               1
```

例 2: データファイルとデータベースの対応する表領域の表示

データファイルの名前、サイズおよびデータベースの対応する表領域を表示するには、DBA_DATA_FILES ビューに対して次の問合せを入力します。

```
SELECT  FILE_NAME, BLOCKS, TABLESPACE_NAME
   FROM DBA_DATA_FILES;

FILE_NAME                                    BLOCKS  TABLESPACE_NAME
------------                               ----------  ------------------
/U02/ORACLE/IDDB3/DBF/RBS01.DBF                  1536  RBS
/U02/ORACLE/IDDB3/DBF/SYSTEM01.DBF               6586  SYSTEM
/U02/ORACLE/IDDB3/DBF/TEMP01.DBF                 6400  TEMP
/U02/ORACLE/IDDB3/DBF/TESTTBS01.DBF              6400  TESTTBS
/U02/ORACLE/IDDB3/DBF/USERS01.DBF                 384  USERS
```

例 3: 各表領域の空き領域（エクステント）の統計の表示

データベース内の各表領域について、使用可能エクステントと結合アクティビティの統計を生成するには、次の問合せを入力します。

```
SELECT TABLESPACE_NAME "TABLESPACE", FILE_ID,
   COUNT(*)    "PIECES",
   MAX(blocks) "MAXIMUM",
   MIN(blocks) "MINIMUM",
   AVG(blocks) "AVERAGE",
   SUM(blocks) "TOTAL"
   FROM DBA_FREE_SPACE
GROUP BY TABLESPACE_NAME, FILE_ID;

TABLESPACE    FILE_ID  PIECES   MAXIMUM    MINIMUM  AVERAGE    TOTAL
----------    -------  ------   -------    -------  -------   ------
RBS                 2       1       955        955      955       955
SYSTEM              1       1       119        119      119       119
TEMP                4       1      6399       6399     6399      6399
TESTTBS             5       5      6364          3     1278      6390
USERS               3       1       363        363      363       363
```

PIECES は表領域ファイル内の空き領域エクステント数を示し、MAXIMUM および MINIMUM はデータベース・ブロック内の領域で連続する最大領域と最小領域を示します。また、AVERAGE は空き領域があるエクステントの平均ブロック・サイズを示し、TOTAL は各表領域ファイル内の空き領域のブロック数を示します。新しいオブジェクトを作成しようとしているとき、またはセグメントを拡張予定であり、表領域に十分な領域があることを確かめるときに、この問合せを使用します。

13

データファイルおよび一時ファイルの管理

この章の内容は次のとおりです。

関連項目： Oracle Database サーバーによって作成および管理されるデータファイルと一時ファイルの作成方法は、第 15 章「Oracle Managed Files の使用」を参照してください。

データファイルを管理するためのガイドライン

データファイルは、データベース内に存在するすべての論理構造のデータを格納する、オペレーティング・システムの物理ファイルです。データファイルは、表領域ごとに明示的に作成する必要があります。

> **注意：** 一時ファイルはデータファイルの特殊なクラスで、一時表領域にのみ関連付けられます。この章で説明する内容は、相違点が示されている場合を除き、データファイルと一時ファイルの両方に適用されます。一時ファイルの詳細は、12-11 ページの「ローカル管理の一時表領域の作成」を参照してください。

Oracle Database は、絶対ファイル番号および相対ファイル番号という 2 つの関連するファイル番号を各データファイルに割り当てます。これらは、データファイルを一意に識別するために使用されます。これらの番号について、次の表で説明します。

ファイル番号のタイプ	説明
絶対	データベース内のデータファイルを一意に識別します。このファイル番号は、ファイル名を使用するかわりにデータファイルを参照する多くの SQL 文で使用できます。絶対ファイル番号は、V$DATAFILE ビューまたは V$TEMPFILE ビューの FILE# 列、あるいは DBA_DATA_FILES ビューまたは DBA_TEMP_FILES ビューの FILE_ID 列で確認できます。
相対	表領域内のデータファイルを一意に識別します。小規模および中規模サイズのデータベースでは、多くの場合、相対ファイル番号と絶対ファイル番号は同じです。ただし、データベース内のデータファイル数が一定のしきい値（通常は 1023）を超えている場合は、相対ファイル番号と絶対ファイル番号が異なります。大型ファイル表領域では、相対ファイル番号は常に 1024（OS/390 プラットフォームでは 4096）です。

ここでは、データファイルの管理について説明します。この項の内容は、次のとおりです。

- データファイル数の決定
- データファイルのサイズ設定
- 適切なデータファイルの配置
- REDO ログ・ファイルから分離したデータファイルの格納

データファイル数の決定

データベースの SYSTEM 表領域および SYSAUX 表領域には、少なくとも 1 つのデータファイルが必要です。データベースにはこれ以外に、複数の表領域とそれに関連するデータファイルまたは一時ファイルが含まれている必要があります。データベースで作成するデータファイルの数に応じて、初期化パラメータの設定値および CREATE DATABASE 文の句の指定が決定します。

オペレーティング・システムによっては、Oracle Database に格納できるデータファイルの数が制限される場合があります。また、データファイル数およびその割当ての方法と場所によって、データベースのパフォーマンスが影響を受ける可能性があることを考慮してください。

> **注意：** データベース内のデータファイルの数を制御してその管理を簡素化する方法の 1 つが、大型ファイル表領域の使用です。大型ファイル表領域は 1 つの大型データファイルのみで構成され、大規模データベースを使用する場合、および論理ボリューム・マネージャを使用してオペレーティング・システム・ファイルを管理する場合に特に役立ちます。大型ファイル表領域については、12-6 ページの「大型ファイル表領域」を参照してください。

データベースのデータファイルの数を決める際は、次のガイドラインを考慮してください。

DB_FILES 初期化パラメータの値の決定

DB_FILES 初期化パラメータは、Oracle Database インスタンスを起動するときに、データファイル情報用に予約するシステム・グローバル領域（SGA）の容量を指定します。これによって、インスタンスで作成可能なデータファイルの最大数が決定します。この制限が適用されるのは、インスタンスが存続する期間内です。DB_FILES の値は（初期化パラメータ設定を変更することで）変更できますが、新しい値はインスタンスをいったん停止して再起動するまでは有効になりません。

DB_FILES の値を決めるときは、次の点を考慮してください。

- DB_FILES の値が小さすぎる場合は、最初にデータベースを停止しないと、DB_FILES 制限を超えてデータファイルを追加できません。
- DB_FILES の値が大きすぎると、メモリーが不必要に消費されます。

データファイルを表領域に追加するときの制限事項の考慮

データファイルを従来の小型ファイル表領域に追加するときには、次の制限事項があります。

- ほとんどのオペレーティング・システムでは、1 つのプロセスで同時にオープンできるファイルの数に制限があります。オープン・ファイル数がオペレーティング・システム制限に達すると、それ以上データファイルを作成できなくなります。
- オペレーティング・システムでは、データファイルの数とサイズに制限があります。
- データベースでは、インスタンスによってオープンされる Oracle Database のデータファイルの最大数が制限されます。この制限はオペレーティング・システムによって異なります。
- DB_FILES 初期化パラメータで指定したデータファイルの数を超えることはできません。
- CREATE DATABASE 文または CREATE CONTROLFILE 文を発行するときに、MAXDATAFILES パラメータによって制御ファイルのデータファイル部分の初期サイズを指定します。ただし、新しく追加するファイルの番号が MAXDATAFILES より大きく DB_FILES 以下であれば、データファイルのセクションにより多数のファイルを格納できるように、制御ファイルが自動的に拡張されます。

パフォーマンスへの影響の考慮

表領域、そして最終的にはデータベースに格納されるデータファイルの数は、パフォーマンスに影響を与える可能性があります。

Oracle Database では、オペレーティング・システムで定義されている制限よりも多くの数のデータファイルをデータベース内に作成できます。データベースの DBW*n* プロセスは、すべてのオンライン・データファイルをオープンできます。Oracle Database には、オープン・ファイル記述子をキャッシュとして処理し、オープン・ファイル記述子の数がオペレーティング・システムで定義されている制限に達したときに自動的にファイルをクローズする機能があります。この機能は、パフォーマンスに悪影響を与えるおそれがあります。できれば、オープン・ファイル記述子に関するオペレーティング・システムの制限を調整して、それがデータベース内のオンライン・データファイルの数より大きくなるようにしてください。

関連項目：

- オペレーティング・システムの制限の詳細は、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。
- CREATE DATABASE 文または CREATE CONTROLFILE 文の MAXDATAFILES パラメータの詳細は、『Oracle Database SQL リファレンス』を参照してください。

データファイルのサイズ設定

表領域を作成するときは、データベース・オブジェクトの将来的なサイズを見積り、十分な数のデータファイルを作成する必要があります。必要に応じて、後で表領域に割り当てられたディスク領域のすべての容量（その結果としてデータベースの容量）を大きくするために、データファイルを作成して表領域に追加できます。可能であれば、データがすべてのデバイスに均等に配分されるように、データファイルを複数のデバイスに作成してください。

適切なデータファイルの配置

表領域の位置は、その表領域を構成するデータファイルの物理的な位置によって決まります。コンピュータのハードウェア資源を適切に使用してください。

たとえば、データベースを格納するためのディスク・ドライブが複数使用可能な場合は、競合の可能性のあるデータファイルを別のディスクに配置することを検討してください。このようにすると、ユーザーが情報を問い合せるときに両方のディスク・ドライブが同時に動作し、同時にデータを取得できます。

関連項目： I/O およびデータファイルの配置の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

REDO ログ・ファイルから分離したデータファイルの格納

データファイルは、データベースの REDO ログ・ファイルが格納されているディスク・ドライブに格納しないでください。データファイルと REDO ログ・ファイルが同じディスク・ドライブに格納されていて、このディスク・ドライブで障害が発生すると、これらのファイルをデータベースのリカバリ手順で使用できなくなります。

REDO ログ・ファイルを多重化すると、全 REDO ログ・ファイルが失われる可能性が低くなるので、データファイルを一部の REDO ログ・ファイルと同じドライブに格納できます。

データファイルの作成および表領域への追加

データファイルを作成して表領域を対応付けるには、次の表にリストされている文のいずれかを使用します。どの文でも、作成するデータファイルのファイル仕様を指定するか、または Oracle Managed Files 機能を使用してデータベース・サーバーが作成し管理するファイルを作成できます。表には、データファイルの作成に使用する文の簡単な説明と、このマニュアル内に記載されている文の詳細説明への参照が示されています。

SQL 文	説明	追加情報
`CREATE TABLESPACE`	表領域とそれを構成するデータファイルを作成します。	12-3 ページ「表領域の作成」
`CREATE TEMPORARY TABLESPACE`	ローカル管理の一時表領域とそれを構成する一時ファイルを作成します。一時ファイルは特殊なデータファイルです。	12-11 ページ「ローカル管理の一時表領域の作成」
`ALTER TABLESPACE ...ADD DATAFILE`	データファイルを作成して表領域に追加します。	12-22 ページ「ローカル管理の一時表領域の変更」
`ALTER TABLESPACE ...ADD TEMPFILE`	一時ファイルを作成して一時表領域に追加します。	12-11 ページ「ローカル管理の一時表領域の作成」
`CREATE DATABASE`	データベースとそれに対応付けられたデータファイルを作成します。	2-6 ページ「CREATE DATABASE 文を使用したデータベースの作成」
`ALTER DATABASE ...CREATE DATAFILE`	古いデータファイルにかわる空のデータファイルを作成します。この文は、バックアップがない状態で失われたデータファイルを再作成する場合に利用します。	『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

表領域に新しいデータファイルを追加するときにファイル名を完全に指定しないと、データファイルは、オペレーティング・システムに応じてデフォルトのデータベース・ディレクトリまたはカレント・ディレクトリに作成されます。データファイルには、常に完全修飾名を使用することをお薦めします。既存のファイルを再利用する場合以外は、新しいファイル名が他のファイルと競合しないことを確認してください。すでに削除済の旧ファイルは上書きされます。

データファイルを作成する文が失敗した場合は、作成されたオペレーティング・システム・ファイルがすべて削除されます。ただし、ファイル・システムやストレージ・サブシステムで発生する多数の潜在的なエラーが原因で、オペレーティング・システムのコマンドを使用した手動でのファイル削除が必要になる場合があります。

データファイルのサイズ変更

ここでは、データファイルのサイズを変更する様々な方法について説明します。この項の内容は、次のとおりです。

データファイルの自動拡張機能の使用可能および使用禁止

データベースに追加の領域が必要になった場合に、自動的にサイズを拡張するデータファイルを作成できます。また、既存のデータファイルを自動拡張するように変更することも可能です。ファイルのサイズは、指定されている最大値に達するまで、指定の増分値ずつ大きくなります。

データファイルを自動的に拡張するように設定しておくと、次のような利点があります。

- 表領域で領域が足りなくなった場合に、管理者が即時に介入する必要性が減ります。
- エクステントの割当て失敗が原因でアプリケーションが停止または一時停止することがなくなります。

データファイルが自動的に拡張できるかどうか確認するには、DBA_DATA_FILES ビューを問い合せ、AUTOEXTENSIBLE 列を調べます。

自動ファイル拡張を指定するには、次の SQL 文を使用してデータファイルを作成するときに AUTOEXTEND ON 句を指定します。

- CREATE DATABASE
- ALTER DATABASE
- CREATE TABLESPACE
- ALTER TABLESPACE

既存のデータファイルの自動ファイル拡張機能を使用可能または使用禁止にしたり、データファイルのサイズを手動で変更するには、ALTER DATABASE 文を使用します。大型ファイル表領域では、ALTER TABLESPACE 文を使用してこれらの操作を実行できます。

次の例は、users 表領域に追加するデータファイルの自動拡張機能を使用可能にします。

```
ALTER TABLESPACE users
    ADD DATAFILE '/u02/oracle/rbdb1/users03.dbf' SIZE 10M
      AUTOEXTEND ON
      NEXT 512K
      MAXSIZE 250M;
```

NEXT の値は、データファイルの拡張時にこのファイルに追加される増分値の最小サイズです。MAXSIZE の値は、自動拡張可能なファイルの最大サイズです。

次の例は、データファイルの自動拡張機能を使用禁止にします。

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/users03.dbf'
    AUTOEXTEND OFF;
```

関連項目： データファイルを作成または変更するための SQL 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

手動によるデータファイルのサイズ変更

手動でデータファイルのサイズを増減させるには、ALTER DATABASE 文を使用します。データファイルのサイズを変更できるため、データファイルを追加しなくてもデータベースに領域を追加できます。この機能は、データベースで許容されているデータファイルの最大数に達することが懸念される場合に有効です。

大型ファイル表領域では、ALTER TABLESPACE 文を使用して、データファイルのサイズを変更できます。大型ファイル表領域にはデータファイルを追加できません。

また、データファイルのサイズを手動で縮小することで、データベース内の未使用領域を再生できます。これは、領域要件の見積りの誤りを訂正する際に有効です。

次の例では、データファイル /u02/oracle/rbdb1/stuff01.dbf が 250MB まで拡張されていることを想定しています。ただし、その表領域には現在小さなオブジェクトが格納されているので、データファイルのサイズを縮小できます。

次の文は、データファイル /u02/oracle/rbdb1/stuff01.dbf のサイズを縮小します。

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf'
   RESIZE 100M;
```

> **注意：** 必ずしもファイルのサイズを指定した値まで縮小できるわけではありません。ファイルに格納されているデータ量が指定の縮小サイズよりも大きい場合は、エラーが戻されます。

データファイルの可用性の変更

個々のデータファイルまたは一時ファイルは、オフライン化またはオンライン化することでその可用性を変更できます。オフラインのデータファイルはデータベースに使用できず、オンライン化されるまでアクセスできません。

データファイルの可用性の変更には、次のような理由があります。

- データファイルのオフライン・バックアップを実行するため。
- データファイルの名前変更または再配置を行うため。最初にデータファイルをオフライン化するか、あるいは表領域をオフライン化する必要があります。
- データベースのデータファイルへの書込みに問題があり、データファイルが自動的にオフライン化された場合。この場合は、後で問題を解決してから、データファイルを手動でオンライン化できます。
- データファイルが破損または欠落している場合。データベースをオープンするには、その前にデータファイルをオフライン化する必要があります。

読取り専用表領域のデータファイルはオフライン化またはオンライン化ができますが、ファイルをオンライン化しても表領域の読取り専用状態に影響を与えることはありません。表領域が読取り / 書込み可能な状態に戻るまで、このデータファイルには書き込めません。

> **注意：** 表領域自体をオフライン化することによって、表領域のすべてのデータファイルを一時的に使用禁止にできます。表領域をオンラインに戻すためには、表領域内のデータファイルはすべてそのままにしておく必要があります。ただし、13-8 ページの「データファイルの名前変更と再配置」で説明する手順に従ってデータファイルを再配置または名前を変更することは可能です。
>
> 詳細は、12-15 ページの「表領域のオフライン化」を参照してください。

データファイルをオンライン化またはオフライン化するには、ALTER DATABASE システム権限が必要です。ALTER TABLESPACE 文を使用してすべてのデータファイルまたは一時ファイルをオフライン化するには、ALTER TABLESPACE または MANAGE TABLESPACE システム権限が

必要です。Oracle Real Application Clusters 環境では、データベースを排他モードでオープンする必要があります。

ここでは、データファイルの可用性を変更する様々な方法について説明します。この項の内容は、次のとおりです。

- ARCHIVELOG モードでデータファイルをオンライン化またはオフライン化する方法
- NOARCHIVELOG モードでデータファイルをオフライン化する方法
- 表領域内のすべてのデータファイルおよび一時ファイルの可用性の変更

ARCHIVELOG モードでデータファイルをオンライン化またはオフライン化する方法

個々のデータファイルをオンライン化するには、DATAFILE 句を指定して ALTER DATABASE 文を発行します。次の文では、指定したデータファイルがオンライン化されます。

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' ONLINE;
```

これと同じファイルをオフライン化するには、次の文を発行します。

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' OFFLINE;
```

注意： この形式の ALTER DATABASE 文を使用するには、データベースを ARCHIVELOG モードにしてください。NOARCHIVELOG モードでデータファイルをオフライン化すると、ファイルが失われる可能性があります。ARCHIVELOG モードを使用することで、データファイルの不慮の損失を防止できます。

NOARCHIVELOG モードでデータファイルをオフライン化する方法

データベースが NOARCHIVELOG モードのときにデータファイルをオフライン化するには、DATAFILE 句および OFFLINE FOR DROP 句を指定して ALTER DATABASE 文を使用します。

- OFFLINE キーワードを指定すると、破損しているかどうかに関係なくデータファイルに OFFLINE のマークが付くため、データベースをオープンできます。
- FOR DROP キーワードによって、データファイルが後で削除されるようにマークが付けられます。このようなデータファイルは、オンラインに戻すことはできません。

注意： この操作では、実際にデータファイルを削除しません。データファイルはデータ・ディクショナリに残っているため、次のいずれかの方法で、データファイルを削除する必要があります。

- ALTER TABLESPACE ...DROP DATAFILE 文

  OFFLINE FOR DROP の後、この方法はディクショナリ管理表領域のみに機能します。
- DROP TABLESPACE ...INCLUDING CONTENTS AND DATAFILES 文
- 前述の方法で失敗した場合は、オペレーティング・システム・コマンドを使用してデータファイルを削除します。この方法は、データ・ディクショナリのデータファイルおよび制御ファイルへの参照が残るため、望ましい方法ではありません。

次の文は、指定したデータファイルをオフライン化し、削除対象としてマークを付けます。

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/users03.dbf' OFFLINE FOR DROP;
```

表領域内のすべてのデータファイルおよび一時ファイルの可用性の変更

ALTER TABLESPACE 文で句を指定することにより、表領域内にあるすべてのデータファイルまたは一時ファイルのオンラインまたはオフラインの状態を変更できます。具体的には、オンライン / オフラインの状態に影響を与える文として次のものがあります。

- ALTER TABLESPACE ...DATAFILE {ONLINE|OFFLINE}
- ALTER TABLESPACE ...TEMPFILE {ONLINE|OFFLINE}

入力が必要なのは表領域名のみであり、個々のデータファイルや一時ファイルを入力する必要はありません。すべてのデータファイルまたは一時ファイルが影響を受けますが、表領域そのもののオンライン / オフラインの状態は変わりません。

ほとんどの場合、データベースがマウントされていれば、オープンしていなくても、前述の ALTER TABLESPACE 文を発行できます。ただし、表領域が SYSTEM 表領域、UNDO 表領域またはデフォルト一時表領域である場合は、データベースをオープンしないでください。ALTER DATABASE DATAFILE 文および ALTER DATABASE TEMPFILE 文にも ONLINE/OFFLINE 句がありますが、これらの文では表領域のファイル名をすべて入力する必要があります。

この操作は表領域の可用性を変更する ALTER TABLESPACE...ONLINE|OFFLINE 文とは操作が異なるため、構文も異なります。ALTER TABLESPACE 文は表領域だけでなくデータファイルもオフラインにしますが、一時表領域または一時ファイルの状態を変更するためには使用できません。

データファイルの名前変更と再配置

データファイルの名前を変更して、それらの名前や位置を変更できます。次の項では、その手順について説明します。

- 単一の表領域のデータファイルを名前変更および再配置する手順
- 複数の表領域のデータファイルを名前変更および再配置する手順

これらの手順を使用してデータファイルを名前変更または再配置すると、データベースの制御ファイルに記録されている、データファイルへのポインタのみが変わります。これらの手順では、オペレーティング・システム・ファイルを物理的に名前変更したり、ファイルをオペレーティング・システム・レベルでコピーすることはありません。データファイルの名前変更と再配置には、複数の手順が必要です。手順と例題をよく理解してから実行してください。

単一の表領域のデータファイルを名前変更および再配置する手順

ここでは、単一の表領域のデータファイルを名前変更および再配置するための手順をいくつか示します。この手順を実行するには、ALTER TABLESPACE システム権限が必要です。

> **関連項目：** データファイルの名前変更または再配置に備えて表領域をオフライン化する方法については、12-15 ページの「表領域のオフライン化」を参照してください。

単一の表領域のデータファイルの名前を変更する手順

単一の表領域のデータファイルの名前を変更する手順は、次のとおりです。

1. データファイルを含む表領域をオフライン化します。データベースはオープンしている必要があります。

   次に例を示します。

   ```
   ALTER TABLESPACE users OFFLINE NORMAL;
   ```

2. オペレーティング・システムを使用してデータファイルの名前を変更します。

3. ALTER TABLESPACE 文に RENAME DATAFILE 句を指定して、データベース内のファイル名を変更します。

   たとえば、次の文はデータファイル /u02/oracle/rbdb1/user1.dbf および /u02/oracle/rbdb1/user2.dbf をそれぞれ /u02/oracle/rbdb1/users01.dbf および /u02/oracle/rbdb1/users02.dbf に名前変更します。

```
ALTER TABLESPACE users
    RENAME DATAFILE '/u02/oracle/rbdb1/user1.dbf',
                    '/u02/oracle/rbdb1/user2.dbf'
                 TO '/u02/oracle/rbdb1/users01.dbf',
                    '/u02/oracle/rbdb1/users02.dbf';
```

   古いデータファイルと新しいデータファイルを正しく識別するために、必ず完全なファイル名（パスを含む）を指定してください。特に、古いデータファイル名は、データ・ディクショナリの DBA_DATA_FILES ビューに表示されるとおり、正確に指定してください。

4. データベースのバックアップを作成します。データベースの構造を変更した後は、即時にデータベースの完全バックアップを実行してください。

単一の表領域のデータファイルを再配置する手順

ここでは、データファイルを再配置する手順の例を示します。

想定する条件は、次のとおりです。

- オープンしているデータベースに users という表領域が存在し、すべて同じディスク上に配置されたデータファイルによって構成されています。
- users 表領域のデータファイルを、別の分離されたディスク・ドライブに再配置します。
- 現在、オープンしているデータベースに管理者権限で接続しています。
- データベースの現行のバックアップは取得済です。

次の手順を実行します。

1. 特定のファイルの名前やサイズが不明な場合は、データ・ディクショナリ・ビュー DBA_DATA_FILES を問い合せて情報を取得できます。

```
SQL> SELECT FILE_NAME, BYTES FROM DBA_DATA_FILES
  2> WHERE TABLESPACE_NAME = 'USERS';

FILE_NAME                                BYTES
---------------------------------------- ----------------
/u02/oracle/rbdb1/users01.dbf            102400000
/u02/oracle/rbdb1/users02.dbf            102400000
```

2. データファイルを含む表領域をオフライン化します。

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

3. オペレーティング・システムを使用し、データファイルを新しい位置にコピーして名前変更します。13-12 ページの「データベース・サーバーを使用したファイルのコピー」で説明する DBMS_FILE_TRANSFER パッケージを使用して、ファイルをコピーできます。

> **注意：** SQL*Plus の HOST コマンドを使用すると、SQL*Plus を一時的に終了し、オペレーティング・システムのコマンドを実行してファイルをコピーできます。

4. データベース内のデータファイルの名前を変更します。

   users 表領域を構成するファイルのデータファイル・ポインタは、対応付けられているデータベースの制御ファイルに記録されていますが、これらのポインタをこの時点で旧ファイル名から新ファイル名に変更する必要があります。

   ALTER TABLESPACE...RENAME DATAFILE 文を使用します。

```
ALTER TABLESPACE users
    RENAME DATAFILE '/u02/oracle/rbdb1/users01.dbf',
                    '/u02/oracle/rbdb1/users02.dbf'
                 TO '/u03/oracle/rbdb1/users01.dbf',
                    '/u04/oracle/rbdb1/users02.dbf';
```

5. データベースのバックアップを作成します。データベースの構造を変更した後は、即時にデータベースの完全バックアップを実行してください。

複数の表領域のデータファイルを名前変更および再配置する手順

1 つ以上の表領域のデータファイルは、ALTER DATABASE RENAME FILE 文を使用して、名前変更および再配置できます。1 回の操作で複数の表領域のデータファイルを名前変更または再配置する方法は、これ以外にありません。この手順を実行するには、ALTER DATABASE システム権限が必要です。

注意： SYSTEM 表領域、デフォルト一時表領域、またはアクティブな UNDO 表領域のデータファイルを名前変更または再配置する場合、これらの表領域はオフライン化できないため、この ALTER DATABASE 文を使用する必要があります。

複数の表領域のデータファイルの名前を変更する手順は、次のとおりです。

1. データベースがマウントされ、クローズされていることを確認します。

   注意： データベースは必要な場合クローズする必要はありませんが、データファイル（または一時ファイル）はオフラインにする必要があります。

   ---

2. オペレーティング・システムで新しい位置と名前を指定して、名前変更するデータファイルをコピーします。13-12 ページの「データベース・サーバーを使用したファイルのコピー」で説明する DBMS_FILE_TRANSFER パッケージを使用して、ファイルをコピーできます。

3. ALTER DATABASE を使用して、データベースの制御ファイル内のファイル・ポインタの名前を変更します。

   たとえば、次の文はデータファイル /u02/oracle/rbdb1/sort01.dbf および /u02/oracle/rbdb1/user3.dbf をそれぞれ /u02/oracle/rbdb1/temp01.dbf および /u02/oracle/rbdb1/users03.dbf に名前変更します。

```
ALTER DATABASE
    RENAME FILE '/u02/oracle/rbdb1/sort01.dbf',
                '/u02/oracle/rbdb1/user3.dbf'
             TO '/u02/oracle/rbdb1/temp01.dbf',
                '/u02/oracle/rbdb1/users03.dbf;
```

   古いデータファイルと新しいデータファイルを正しく識別するために、必ず完全なファイル名（パスを含む）を指定してください。特に、古いデータファイル名は、DBA_DATA_FILES ビューに表示されるとおり、正確に指定してください。

4. データベースのバックアップを作成します。データベースの構造を変更した後は、即時にデータベースの完全バックアップを実行してください。

データファイルの削除

単一のデータファイルまたは一時ファイルを削除するには、ALTER TABLESPACE コマンドの DROP DATAFILE および DROP TEMPFILE 句を使用します。データファイルは空である必要があります（データファイルから割り当てられたエクステントが残っていない場合、データファイルは空とみなされます）。データファイルまたは一時ファイルを削除すると、データファイルまたは一時ファイルへの参照はデータ・ディクショナリおよび制御ファイルから削除され、物理ファイルがファイル・システムまたは自動ストレージ管理（ASM）ディスク・グループから削除されます。

次の例では、ASM ディスク・グループ DGROUP1 の別名 example_df3.f で識別されたデータファイルを削除します。データファイルは表領域 example に属します。

```
ALTER TABLESPACE example DROP DATAFILE '+DGROUP1/example_df3.f';
```

次の例では、表領域 lmtemp に属する一時ファイル lmtemp02.dbf を削除します。

```
ALTER TABLESPACE lmtemp DROP TEMPFILE '/u02/oracle/data/lmtemp02.dbf';
```

これは、次の文と同じです。

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' DROP
     INCLUDING DATAFILES;
```

ALTER TABLESPACE 構文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

データファイルの削除に関する制限事項

データファイルおよび一時ファイルの削除に関する制限事項は、次のとおりです。

- データベースはオープンしている必要があります。
- データファイルが空でない場合、そのファイルは削除できません。

  空でなく、スキーマ・オブジェクトを削除しても空にできないデータファイルを削除する場合は、そのデータファイルを含む表領域を削除する必要があります。
- 表領域の最初のデータファイルまたは 1 つのみのデータファイルは削除できません。

  これは、大型ファイル表領域には DROP DATAFILE を使用できないことを意味します。
- 読取り専用表領域のデータファイルは削除できません。
- SYSTEM 表領域のデータファイルは削除できません。
- ローカル管理表領域のデータファイルがオフラインの場合、そのファイルは削除できません。

関連項目： 12-24 ページ「表領域の削除」

データファイル内のデータ・ブロックの検証

チェックサムを使用してデータ・ブロックを検証するようにデータベースを構成する場合は、初期化パラメータ DB_BLOCK_CHECKSUM を TYPICAL（デフォルト）に設定します。これによって、DBW*n* プロセスおよびダイレクト・ローダーが各ブロックのチェックサムを計算し、ブロックをディスクに書き込むときにこのチェックサムをブロック・ヘッダーに格納します。

ブロックが読み込まれるときにチェックサムが検証されるのは、DB_BLOCK_CHECKSUM が TRUE に設定され、前回ブロックが書き込まれたときにチェックサムが格納されている場合のみです。破損が検出されると、メッセージ ORA-01578 が返され、破損に関する情報がアラート・ログに書き込まれます。

DB_BLOCK_CHECKSUM パラメータの値は、ALTER SYSTEM 文で動的に変更できます。このパラメータの設定に関係なく、SYSTEM 表領域のデータ・ブロックは常にチェックサムを使用して検証されます。

関連項目： DB_BLOCK_CHECKSUM 初期化パラメータの詳細は、『Oracle Database リファレンス』を参照してください。

データベース・サーバーを使用したファイルのコピー

データベース内のファイルをコピーする場合、またはデータベース間でファイルを転送する場合（トランスポータブル表領域機能を使用して転送する場合など）は、必ずしもオペレーティング・システムを使用する必要はありません。これらの処理は、DBMS_FILE_TRANSFER パッケージ、または Streams の伝播を使用して行うこともできます。このマニュアルでは Streams の使用方法については説明していませんが、DBMS_FILE_TRANSFER パッケージの使用例は 13-13 ページの「ファイルのローカル・ファイル・システムへのコピー」に示しています。

DBMS_FILE_TRANSFER パッケージでは、ローカル・ファイル・システムまたは自動ストレージ管理（ASM）ディスク・グループをファイル転送の移動元または移動先として使用できます。ASM への転送または ASM からの転送に含めることができるのは、Oracle データベース・ファイル（データファイル、一時ファイル、制御ファイルなど）のみです。

注意： データベースによって変更されているファイルのコピーまたは転送に DBMS_FILE_TRANSFER パッケージを使用しないでください。コピーや転送に使用すると、ファイルの整合性がなくなる可能性があります。

UNIX システムでは、DBMS_FILE_TRANSFER パッケージを使用して作成したファイルの所有者は、インスタンスを実行するシャドウ・プロセスの所有者です。通常、この所有者は ORACLE です。DBMS_FILE_TRANSFER を使用して作成されたファイルは、常に、データベース内のすべてのプロセスで書込みと読取りが可能です。ただし、権限を持たないユーザーがこのようなファイルを直接読取りまたは書込みする必要がある場合は、システム管理者によるアクセスが必要になる場合があります。

この項の内容は、次のとおりです。

関連項目：

- 『Oracle Streams 概要および管理』
- 12-30 ページ「データベース間での表領域のトランスポート」
- DBMS_FILE_TRANSFER パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ファイルのローカル・ファイル・システムへのコピー

この項では、DBMS_FILE_TRANSFER パッケージの COPY_FILE プロシージャを使用してファイルをローカル・ファイル・システムにコピーする例を示します。次の例では、/usr/admin/source ディレクトリ内の db1.dat という名前のバイナリ・ファイルを、ローカル・ファイル・システムの /usr/admin/destination ディレクトリに db1_copy.dat という名前でコピーします。

1. 権限を付与でき、SQL を使用してディレクトリ・オブジェクトを作成できる管理ユーザーとして SQL*Plus で接続します。
2. SQL コマンド CREATE DIRECTORY を使用して、コピー元ファイルが格納されるディレクトリのディレクトリ・オブジェクトを作成します。ディレクトリ・オブジェクトは、ディレクトリの別名に類似しています。たとえば、SOURCE_DIR というディレクトリ・オブジェクトを、使用しているコンピュータ・システムの /usr/admin/source ディレクトリに対して作成するには、次の文を実行します。

   ```
   CREATE DIRECTORY SOURCE_DIR AS '/usr/admin/source';
   ```

3. SQL コマンド CREATE DIRECTORY を使用して、バイナリ・ファイルがコピーされるディレクトリのディレクトリ・オブジェクトを作成します。たとえば、DEST_DIR というディレクトリ・オブジェクトを、使用しているコンピュータ・システムの /usr/admin/destination ディレクトリに対して作成するには、次の文を実行します。

   ```
   CREATE DIRECTORY DEST_DIR AS '/usr/admin/destination';
   ```

4. COPY_FILE プロシージャを実行するユーザーに対して、必要な権限を付与します。この例では、strmadmin ユーザーがプロシージャを実行します。

   ```
   GRANT EXECUTE ON DBMS_FILE_TRANSFER TO strmadmin;

   GRANT READ ON DIRECTORY source_dir TO strmadmin;

   GRANT WRITE ON DIRECTORY dest_dir TO strmadmin;
   ```

5. strmadmin ユーザーとして接続し、プロンプトが表示されたらユーザー・パスワードを入力します。

   ```
   CONNECT strmadmin
   ```

6. COPY_FILE プロシージャを実行して、ファイルをコピーします。

   ```
   BEGIN
     DBMS_FILE_TRANSFER.COPY_FILE(
           source_directory_object       =>  'SOURCE_DIR',
           source_file_name              =>  'db1.dat',
           destination_directory_object  =>  'DEST_DIR',
           destination_file_name         =>  'db1_copy.dat');
   END;
   /
   ```

注意： データベースによって変更されているファイルのコピーまたは転送に DBMS_FILE_TRANSFER パッケージを使用しないでください。コピーや転送に使用すると、ファイルの整合性がなくなる可能性があります。

サード・パーティ・ファイル転送

DBMS_FILE_TRANSFER パッケージのプロシージャは、通常、ローカル・プロシージャ・コールとして起動しますが、リモート・プロシージャ・コールとして起動することもできます。リモート・プロシージャ・コールとして起動すると、別のデータベースに接続している場合でも、データベース内のファイルをコピーできます。たとえば、次のリモート・プロシージャ・コールを実行すると、別のデータベースに接続していても、データベース DB でファイルをコピーできます。

```
DBMS_FILE_TRANSFER.COPY_FILE@DB(...)
```

また、リモート・プロシージャ・コールを使用すると、いずれのデータベースにも接続していなくても、2 つのデータベース間でファイルをコピーできます。たとえば、データベース A に接続し、ファイルをデータベース B からデータベース C に転送できます。この場合、データベース A は、転送するファイルの転送元でも転送先でもないため、サード・パーティになります。

サード・パーティ・ファイル転送では、ファイルのプッシュとプルの両方が可能です。前述の例では、A から B または C へのデータベース・リンクがあり、そのデータベースから別のデータベースへのデータベース・リンクがある場合は、サード・パーティ・ファイル転送を実行できます。データベース A から B および C 両方へのデータベース・リンクは必要ありません。

たとえば、A から B へのデータベース・リンクがあり、B から C への別のデータベース・リンクがある場合は、データベース A で次のプロシージャを実行してファイルを B から C に転送できます。

```
DBMS_FILE_TRANSFER.PUT_FILE@B(...)
```

この構成では、ファイルをプッシュします。

または、A から C へのデータベース・リンクがあり、C から B への別のデータベース・リンクがある場合は、データベース A で次のプロシージャを実行してファイルを B から C に転送できます。

```
DBMS_FILE_TRANSFER.GET_FILE@C(...)
```

この構成では、ファイルをプルします。

ファイル転送と DBMS_SCHEDULER パッケージ

DBMS_SCHEDULER パッケージを使用して、単一のデータベース内およびデータベース間でファイルを自動的に転送できます。DBMS_SCHEDULER パッケージではサード・パーティ・ファイル転送もサポートされています。スケジューラによって実行されるファイル転送が長時間実行される場合は、ファイルの読取りまたは書込みを行うデータベースで V$SESSION_LONGOPS 動的パフォーマンス・ビューを使用してファイル転送を監視できます。スケジューラ・ジョブで使用するデータベース・リンクは、必ず固定ユーザー・データベース・リンクです。

再開可能なスケジューラ・ジョブを使用すると、特に断続的に障害が発生する場合に、ファイル転送の信頼性を自動的に改善できます。転送先ファイルがクローズする前にファイル転送が失敗した場合は、部分的に書き込まれた転送先ファイルがデータベースによって削除された後、ファイル転送を最初から再開できます。したがって、ジョブの残りの部分が再開可能な場合は、再開可能なスケジューラ・ジョブを使用してファイルを転送することを検討してください。スケジューラ・ジョブの詳細は、第 27 章「Oracle Scheduler を使用したジョブのスケジューリング」を参照してください。

注意： 再開可能な 1 つのジョブで複数のファイルを転送する場合は、すでに転送済のファイルと転送されていないファイルがある状態でジョブを再開する方法を検討する必要があります。

拡張ファイル転送メカニズム

DBMS_FILE_TRANSFER パッケージと DBMS_SCHEDULER パッケージの両方を使用すると、複雑なファイル転送メカニズムを作成できます。たとえば、転送するファイルのコピーが複数のデータベースに存在する場合は、ソースの可用性、ソースのロード、宛先データベースへの通信帯域幅などの要因を検討して、最初にアクセスするソース・データベース、および障害発生時にアクセスを試みるソース・データベースを決定します。この場合、それらの要因に関する情報を入手する必要があるため、要因を検討するメカニズムを構築する必要があります。

別の例として、ロードよりも完了時間が短いことが重要である場合は、複数のスケジューラ・ジョブを発行してファイル転送をパラレルで実行できます。また、ソース・データベースと転送先データベースのファイル・レイアウトに関する知識があれば、使用する I/O デバイスが異なる場合のみ同時転送を実行またはスケジュールすることで、ディスクの競合を最小限にできます。

ファイルと物理デバイスのマッピング

データファイルが単なるファイル・システム・ファイルである環境や、RAW デバイス上で直接作成される環境では、表領域と基礎となるデバイスとの関連付けを調べるのは比較的容易です。Oracle Database には、ファイルとデバイスとのマッピングを提供する DBA_TABLESPACES、DBA_DATA_FILES および V$DATAFILE などのビューが用意されています。これらのマッピングをデバイス統計と併用して、I/O パフォーマンスを評価できます。

ただし、ホスト・ベースの論理ボリューム・マネージャ（LVM）と、Redundant Array of Inexpensive Disks（RAID）機能を提供する洗練されたストレージ・サブシステムの導入によって、ファイルからデバイスへのマッピングを判別するのが難しくなっています。最新ファイルがブラック・ボックスに隠れていると、そのファイルを判断するのが難くなるため、問題が発生します。この項では、この問題を解決するための Oracle Database のアプローチについて説明します。

この項の内容は、次のとおりです。

- Oracle Database のファイル・マッピング・インタフェースの概要
- Oracle Database のファイル・マッピング・インタフェースの動作
- Oracle Database のファイル・マッピング・インタフェースの使用方法
- ファイル・マッピングの例

注意： この項では、Oracle Database のファイル・マッピング・インタフェースの概要と、DBMS_STORAGE_MAP パッケージおよび動的パフォーマンス・ビューを使用してファイルから物理デバイスへのマッピングを公開する方法について説明します。この機能にアクセスするには、Oracle Enterprise Manager（EM）を使用する方が簡単です。EM には、ファイルから物理デバイスへのマッピング用に、使用しやすいグラフィカル・インタフェースが用意されています。

Oracle Database のファイル・マッピング・インタフェースの概要

I/O パフォーマンスを把握するには、ファイルが格納されている記憶域の階層の詳細を知る必要があります。Oracle Database には、ファイル、論理ビューの中間レイヤーおよび実際の物理デバイスのマッピング全体を表示するメカニズムが用意されています。この表示には、動的パフォーマンス・ビュー（V$ ビュー）のセットが使用されます。これらのビューを使用すると、ファイル・ブロックがあるディスクを正確に特定できます。

これらのビューを作成するために、ストレージ・ベンダーは特定の I/O スタック要素のマッピングを受け持つマッピング・ライブラリを提供する必要があります。データベースは、バックグラウンド・プロセス FMON によって起動される外部の非 Oracle Database プロセスを介して、これらのライブラリと通信します。FMON は、マッピング情報の管理を受け持ちます。Oracle には PL/SQL パッケージ DBMS_STORAGE_MAP が用意されており、このパッケージを使用して、マッピング・ビューを移入するマッピング操作を起動します。

注意： ファイル・マッピング・インタフェースは、Windows プラットフォームでは使用できません。

Oracle Database のファイル・マッピング・インタフェースの動作

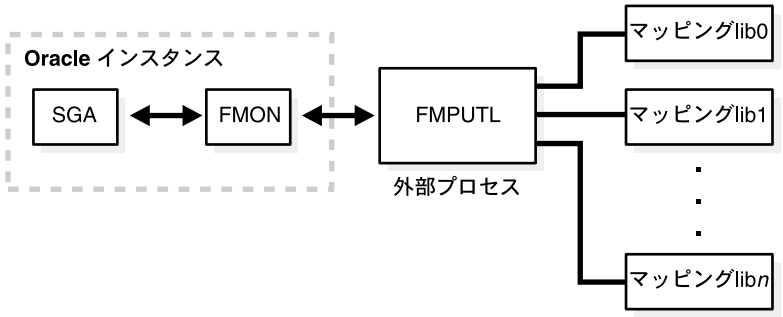
この項では、Oracle Database のファイル・マッピング・インタフェースの構成要素と、インタフェースの動作について説明します。この章の内容は、次のとおりです。

- ファイル・マッピングの構成要素
- マッピング構造
- マッピング構造の例
- 構成 ID

ファイル・マッピングの構成要素

次の図は、ファイル・マッピング・メカニズムの構成要素を示しています。

図 13-1 ファイル・マッピングの構成要素



ここでは、これらの構成要素と、各構成要素が連動してマッピング・ビューを移入する動作について説明します。

- FMON
- 外部プロセス（FMPUTL）
- マッピング・ライブラリ

FMON FMON は、FILE_MAPPING 初期化パラメータが TRUE に設定されている場合に、データベースにより起動されるバックグラウンド・プロセスです。FMON の役割は、次のとおりです。

- SGA に格納されるマッピング情報を作成します。この情報は、次の構造で構成されます。
  - ファイル
  - ファイル・システムのエクステント
  - 要素
  - 副要素

  これらの構造については、13-18 ページの「マッピング構造」を参照してください。
- 次の原因で変更が発生した場合にマッピング情報をリフレッシュします。
  - データファイル（サイズ）の変更
  - データファイルの追加または削除
  - 記憶域の構成変更（低頻度）
- マッピング情報をデータ・ディクショナリに保存して、起動操作と停止操作の間も持続する情報のビューを保持します。
- インスタンスの起動時にマッピング情報を SGA にリストアします。これにより、インスタンスを起動するたびにマッピング情報全体を再作成するという、高コストの操作が不要になります。

DBMS_STORAGE_MAP パッケージで起動されるプロシージャを使用すると、このマッピングを制御しやすくなります。

外部プロセス（FMPUTL） FMON は外部の非 Oracle Database プロセス FMPUTL を起動し、このプロセスはベンダーが提供するマッピング・ライブラリと直接通信します。このプロセスは、I/O スタックのすべてのレベルにマッピング・ライブラリが存在していれば、すべてのレベルを通じてマッピング情報を取得します。一部のプラットフォームでは、I/O マッピング・スタックの全レベルを通じてマッピングするにはルート権限が必要であるため、外部プロセスの SETUID ビットを ON に設定する必要があります。

この外部プロセスの役割は、マッピング・ライブラリを検出してアドレス空間に動的にロードすることです。

マッピング・ライブラリ Oracle Database はマッピング・ライブラリを使用して、特定のマッピング・ライブラリが所有する要素のマッピング情報を検出します。これらのマッピング・ライブラリを通じて、個々の I/O スタック要素に関する情報が伝達されます。この情報を使用して、ユーザーが問合せできる動的パフォーマンス・ビューが移入されます。

マッピングを完成するには、すべてのスタック・レベルにマッピング・ライブラリが存在する必要があり、各ライブラリが I/O マッピング・スタックの独自部分を所有できます。たとえば、VERITAS VxVM ライブラリは VERITAS ボリューム・マネージャに関連するスタック要素を所有し、EMC ライブラリは I/O マッピング・スタックのうちすべての EMC ストレージ固有レイヤーを所有します。

マッピング・ライブラリはベンダーから提供されます。ただし、現在、Oracle には EMC ストレージ用のマッピング・ライブラリが用意されています。データベース・サーバーに使用可能なマッピング・ライブラリは、特殊ファイル filemap.ora 内で識別されます。

マッピング構造

この項では、マッピング構造とその Oracle Database 表現について説明します。マッピング・ビューに表示される情報を解析するには、この情報を理解する必要があります。

マッピング情報を構成する基本構造は、次のとおりです。

- ファイル

  すべてのマッピング構造は、ファイル・サイズ、ファイルを構成するファイル・システムのエクステント数およびファイル・タイプなど、ファイルの属性セットを提供します。

- ファイル・システムのエクステント

  ファイル・システムのエクステントのマッピング構造では、1 つの要素にあるブロックの連続するチャンクが記述されます。これには、デバイス・オフセット、エクステント・サイズ、ファイル・オフセット、タイプ（データまたはパリティ）およびエクステントがある要素の名前が含まれます。

  注意： ファイル・システムのエクステントは、Oracle Database のエクステントとは異なります。ファイル・システムのエクステントは、そのファイル・システムで管理されるデバイスに書き込まれる連続する物理データ・ブロックです。Oracle Database のエクステントは、表領域エクステントなど、データベースで管理される論理構造です。

- 要素

  要素のマッピング構造は、I/O スタック内の記憶域コンポーネントを記述する抽象マッピング構造です。要素には、ミラー、ストライプ、パーティション、RAID5、連結要素およびディスクがあります。これらの構造は、マッピングのビルディング・ブロックです。

- 副要素

  副要素のマッピング構造では、I/O マッピング・スタック内のある要素と次の要素のリンクが記述されます。この構造には、副要素番号、サイズ、副要素が存在する要素の名前および要素のオフセットが含まれます。

次の例は、これらのマッピング構造すべてを示しています。

マッピング構造の例

2 つのデータファイル X および Y で構成される Oracle Database を考えてみます。ファイル X と Y はどちらもボリューム A にマウントされたファイル・システム上に存在し、ファイル X は 2 つのエクステントで構成され、ファイル Y は 1 つのエクステントのみで構成されているとします。

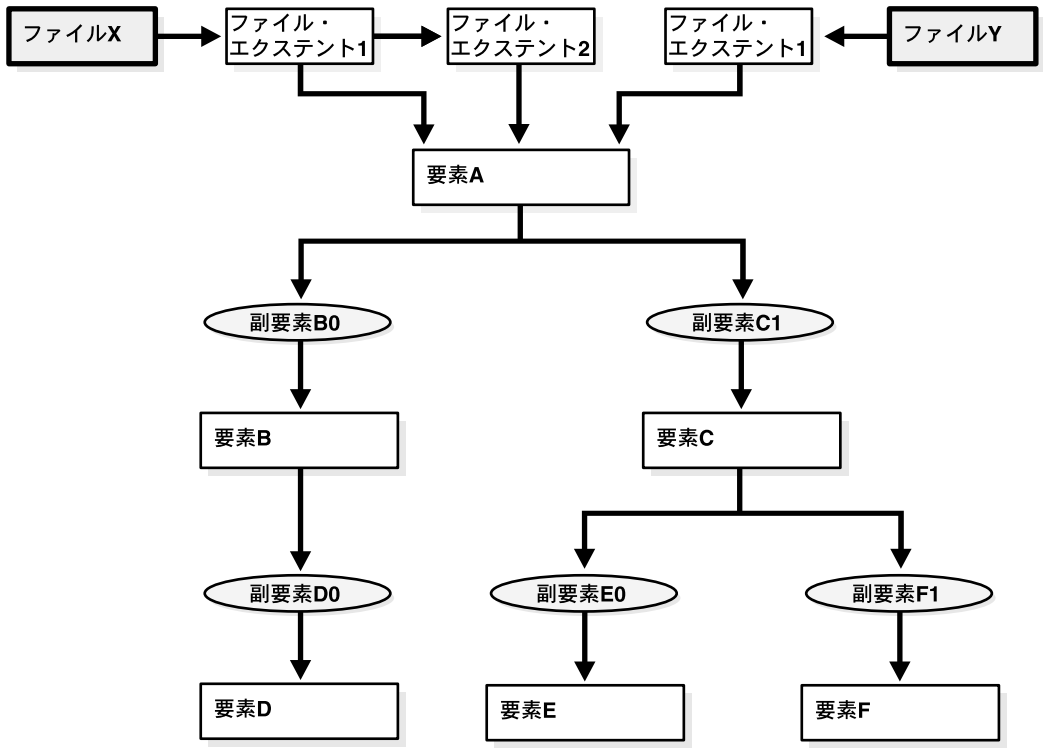
ファイル X の 2 つのエクステントとファイル Y の 1 つのエクステントは、いずれも要素 A にマップされます。要素 A は、要素 B および C にストライプ化されています。要素 A は、副要素 B0 と C1 を介してそれぞれ要素 B と C にマップされます。

要素 B は、要素 D（物理ディスク）のパーティションで、副要素 D0 を介して要素 D にマップされます。

要素 C は、副要素 E0 と F1 を介してそれぞれ要素 E と F（両方とも物理ディスク）にまたがってミラー化されています。

図 13-2 は、すべてのマッピング構造を示しています。

図 13-2 マッピング構造の図



この図が示すマッピング構造は、Oracle Database インスタンスのマッピング情報全体を記述するには十分であり、ファイル内の各論理ブロックを I/O スタック内の各レベルで 1 つ（またはミラー化の場合は 1 つ以上）の（要素名、要素オフセットの）タプルにマップしていることに注意してください。

構成 ID

構成 ID では、要素またはファイルに関連付けられたバージョン情報を取得します。ベンダーのライブラリには構成 ID が用意されており、変更があるたびに更新されます。構成 ID がなければ、データベースではマッピングに変更があったかどうかを指示できません。

構成 ID には、次の 2 種類があります。

- 永続

  この種の構成 ID は、インスタンスが停止されても持続します。

- 非永続

  この種の構成 ID は、インスタンスが停止すると持続しません。データベースでは、インスタンスが稼働している間にのみマッピング情報をリフレッシュできます。

Oracle Database のファイル・マッピング・インタフェースの使用方法

この項では、Oracle Database のファイル・マッピング・インタフェース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。

- ファイル・マッピングの有効化
- DBMS_STORAGE_MAP パッケージの使用
- ファイル・マッピング・ビューからの情報の取得

ファイル・マッピングの有効化

ファイル・マッピング機能を使用可能にする手順は、次のとおりです。

1. 32 ビット・プラットフォームの場合は /opt/ORCLfmap/prot1_32/etc ディレクトリ、64 ビット・プラットフォームの場合は /opt/ORCLfmap/prot1_64/etc ディレクトリに、有効な filemap.ora ファイルが存在することを確認します。

   > **注意：** filemap.ora ファイルの形式と内容についてはこの項で説明しますが、これはあくまでも参考情報です。filemap.ora ファイルは、システムのインストール時にデータベースによって作成されます。ベンダーから独自ライブラリが提供されるまで、filemap.ora ファイルのエントリは 1 つのみで、オラクル社が提供する EMC ライブラリに関するものです。このエントリをコメント解除して手動で変更する必要があるのは、EMC Symmetrix 配列が使用可能な場合のみです。

   filemap.ora ファイルは、使用可能なすべてのマッピング・ライブラリが記述されている構成ファイルです。FMON を使用するには、filemap.ora ファイルが存在し、マッピング・ライブラリへの有効なパスを指している必要があります。それ以外の場合は、正常に起動しません。

   ライブラリごとに、次の行を filemap.ora に含める必要があります。

   ```
   lib=vendor_name:mapping_library_path
   ```

   各項目の意味は次のとおりです。

   - *vendor_name* には、EMC Symmetric ライブラリの場合は `Oracle` を指定します。
   - *mapping_library_path* には、マッピング・ライブラリのフルパスを指定します。

   このファイル内のライブラリの順序がきわめて重要であることに注意してください。各ライブラリは、構成ファイル内での順序に基づいて問合せされます。

   ファイル・マッピング・サービスは、使用可能なマッピング・ライブラリがなくても起動できます。filemap.ora ファイルは、空であっても存在する必要があります。この場合、マッピング・サービスは、新しいマッピング情報を検出できないという制約を伴います。この種の構成で許可されるのは、リストア操作と削除操作のみです。

2. `FILE_MAPPING` 初期化パラメータを `TRUE` に設定します。

   このパラメータを設定するためにインスタンスを停止する必要はありません。次の `ALTER SYSTEM` 文を使用して設定できます。

   ```
   ALTER SYSTEM SET FILE_MAPPING=TRUE;
   ```

3. 適切な `DBMS_STORAGE_MAP` マッピング・プロシージャを起動します。次の 2 つの方法があります。

   - コールド・スタートの使用例では、Oracle Database が起動するのみで、まだマッピング操作は起動されていません。`DBMS_STORAGE_MAP.MAP_ALL` プロシージャを実行して、データベースに関連する I/O サブシステム全体のマッピング情報を作成します。
   - ウォーム・スタートの使用例では、マッピング情報はすでに作成されており、`DBMS_STORAGE_MAP.MAP_SAVE` プロシージャを起動してマッピング情報をデータ・ディクショナリに保存するかどうかをオプションで選択できます。（このプロシージャ

は、デフォルトで DBMS_STORAGE_MAP.MAP_ALL() 内で起動します。）これにより、SGA 内のすべてのマッピング情報がディスクに強制的にフラッシュされます。

データベースの再起動後に、DBMS_STORAGE_MAP.RESTORE() を使用してマッピング情報を SGA にリストアします。必要な場合は、DBMS_STORAGE_MAP.MAP_ALL() をコールしてマッピング情報をリフレッシュできます。

DBMS_STORAGE_MAP パッケージの使用

DBMS_STORAGE_MAP パッケージを使用すると、マッピング操作を制御できます。次の表に、使用可能な各種プロシージャを示します。

プロシージャ	用途
MAP_OBJECT	オブジェクト名、所有者およびタイプで識別されるデータベース・オブジェクトのマッピング情報を作成します。
MAP_ELEMENT	指定した要素のマッピング情報を作成します。
MAP_FILE	指定したファイル名のマッピング情報を作成します。
MAP_ALL	すべてのタイプのデータベース・ファイル（アーカイブ・ログ以外）のマッピング情報全体を作成します。
DROP_ELEMENT	指定した要素のマッピング情報を削除します。
DROP_FILE	指定したファイル名のファイル・マッピング情報を削除します。
DROP_ALL	このインスタンスの SGA からすべてのマッピング情報を削除します。
SAVE	マッピング全体の再生成に必要な情報をデータ・ディクショナリに保存します。
RESTORE	マッピング情報全体をデータ・ディクショナリからインスタンスの共有メモリーにロードします。
LOCK_MAP	このインスタンスの SGA 内でマッピング情報をロックします。
UNLOCK_MAP	このインスタンスの SGA 内でマッピング情報のロックを解除します。

関連項目：

- DBMS_STORAGE_MAP パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- DBMS_STORAGE_MAP パッケージの使用例は、13-23 ページの「ファイル・マッピングの例」を参照してください。

ファイル・マッピング・ビューからの情報の取得

DBMS_STORAGE_MAP パッケージにより生成されたマッピング情報は、動的パフォーマンス・ビューで取得されます。次の表に、これらのビューの概要を示します。

ビュー	説明
V$MAP_LIBRARY	外部プロセスにより動的にロードされたすべてのマッピング・ライブラリのリストが含まれています。
V$MAP_FILE	インスタンスの共有メモリーにあるすべてのファイル・マッピング構造のリストが含まれています。
V$MAP_FILE_EXTENT	インスタンスの共有メモリーにあるすべてのファイル・システム・エクステントのマッピング構造のリストが含まれています。
V$MAP_ELEMENT	インスタンスの SGA にあるすべての要素マッピング構造のリストが含まれています。

ビュー	説明
V$MAP_EXT_ELEMENT	すべての要素マッピングの補足情報が含まれています。
V$MAP_SUBELEMENT	インスタンスの共有メモリーにあるすべての副要素マッピング構造のリストが含まれています。
V$MAP_COMP_LIST	すべての要素マッピング構造の補足情報が含まれています。
V$MAP_FILE_IO_STACK	ファイルの記憶域コンテナの階層配列が一連の行として表示されます。各行は 1 つの階層レベルを表します。

関連項目： 動的パフォーマンス・ビューの詳細は、『Oracle Database リファレンス』を参照してください。

ただし、DBMS_STORAGE_MAP.MAP_OBJECT プロシージャにより生成された情報は、グローバルな一時表 MAP_OBJECT に取得されます。この表には、オブジェクトの記憶域コンテナの階層配置が表示されます。表の各行は 1 つの階層レベルを表します。MAP_OBJECT 表の内容は、次のとおりです。

列	データ型	説明
OBJECT_NAME	VARCHAR2(2000)	オブジェクトの名前。
OBJECT_OWNER	VARCHAR2(2000)	オブジェクトの所有者。
OBJECT_TYPE	VARCHAR2(2000)	オブジェクト型。
FILE_MAP_IDX	NUMBER	ファイル索引（V$MAP_FILE 内の FILE_MAP_IDX に対応）。
DEPTH	NUMBER	I/O スタック内の要素の深さ。
ELEM_IDX	NUMBER	要素に対応する索引。
CU_SIZE	NUMBER	要素上に連続して存在する、ファイルの論理ブロックの連続するセット（HKB 単位）。
STRIDE	NUMBER	この要素上で連続しているファイルの連続単位（CU）間の HKB 数。RAID5 ファイルとストライプ化されたファイルに使用されます。
NUM_CU	NUMBER	ファイル内で STRIDE HKB で区切られた、この要素上で相互に隣接する連続単位の数。RAID5 では、パリティ・ストライプも連続単位数に含まれます。
ELEM_OFFSET	NUMBER	HKB 単位による要素オフセット。
FILE_OFFSET	NUMBER	ファイルの先頭から連続単位の先頭バイトまでのオフセット（HKB 単位）。
DATA_TYPE	VARCHAR2(2000)	データ型（DATA、PARITY または DATA AND PARITY）。
PARITY_POS	NUMBER	パリティの桁。RAID5 のみ。このフィールドは、データ部分とパリティを区別するために必要です。
PARITY_PERIOD	NUMBER	パリティ間隔。RAID5 のみ。

ファイル・マッピングの例

次の例では、Oracle Database のファイル・マッピング機能の強力な機能について説明します。次のような機能があります。

- 特定のデバイスにまたがるすべてのデータベース・ファイルをマップする機能
- 特定のファイルを対応するデバイスにマップする機能
- I/O スタックのすべてのレベルでブロックを配分するなど、特定のデータベース・オブジェクトをマップする機能

次の 2 つのデータファイルで構成される Oracle Database インスタンスを考えてみます。

- `t_db1.f`
- `t_db2.f`

この 2 つのファイルは、VERITAS VxVM ホスト・ベースのストライプ化ボリューム `/dev/vx/dsk/ipfdg/ipf-vol1` にマウントされた Solaris UFS ファイル・システム上で作成されており、このボリュームは EMC Symmetrix 配列から外部化された次のホスト・デバイスで構成されているとします。

- `/dev/vx/rdmp/c2t1d0s2`
- `/dev/vx/rdmp/c2t1d1s2`

次の例では、`MAP_ALL()` 操作を実行する必要があることに注意してください。

例 1: 1 つのデバイスにまたがるすべてのデータベース・ファイルのマッピング

次の問合せでは、`/dev/vx/rdmp/c2t1d1s2` ホスト・デバイスに関連付けられたすべての Oracle Database ファイルが戻されます。

```
SELECT UNIQUE me.ELEM_NAME, mf.FILE_NAME
   FROM V$MAP_FILE_IO_STACK fs, V$MAP_FILE mf, V$MAP_ELEMENT me
   WHERE mf.FILE_MAP_IDX = fs.FILE_MAP_IDX
   AND me.ELEM_IDX = fs.ELEM_IDX
   AND me.ELEM_NAME = '/dev/vx/rdmp/c2t1d1s2';
```

問合せ結果は次のとおりです。

```
ELEM_NAME                 FILE_NAME
------------------------  --------------------------------
/dev/vx/rdmp/c2t1d1s2     /oracle/dbs/t_db1.f
/dev/vx/rdmp/c2t1d1s2     /oracle/dbs/t_db2.f
```

例 2: ファイルから対応するデバイスへのマッピング

次の問合せでは、`/oracle/dbs/t_db1.f` データファイルのトポロジ・グラフが表示されます。

```
WITH fv AS
  (SELECT FILE_MAP_IDX, FILE_NAME FROM V$MAP_FILE
   WHERE FILE_NAME = '/oracle/dbs/t_db1.f')
SELECT fv.FILE_NAME, LPAD(' ', 4 * (LEVEL - 1)) || el.ELEM_NAME ELEM_NAME
   FROM V$MAP_SUBELEMENT sb, V$MAP_ELEMENT el, fv,
     (SELECT UNIQUE ELEM_IDX FROM V$MAP_FILE_IO_STACK io, fv
      WHERE io.FILE_MAP_IDX = fv.FILE_MAP_IDX) fs
   WHERE el.ELEM_IDX = sb.CHILD_IDX
   AND fs.ELEM_IDX = el.ELEM_IDX
   START WITH sb.PARENT_IDX IN
     (SELECT DISTINCT ELEM_IDX
      FROM V$MAP_FILE_EXTENT fe, fv
      WHERE fv.FILE_MAP_IDX = fe.FILE_MAP_IDX)
   CONNECT BY PRIOR sb.CHILD_IDX = sb.PARENT_IDX;
```

表示されるトポロジ・グラフは次のとおりです。

```
FILE_NAME                 ELEM_NAME
------------------------  ------------------------------------------------
/oracle/dbs/t_db1.f       _sym_plex_/dev/vx/rdsk/ipfdg/ipf-vol1_-1_-1
/oracle/dbs/t_db1.f           _sym_subdisk_/dev/vx/rdsk/ipfdg/ipf-vol1_0_0_0
/oracle/dbs/t_db1.f               /dev/vx/rdmp/c2t1d0s2
/oracle/dbs/t_db1.f                   _sym_symdev_000183600407_00C
/oracle/dbs/t_db1.f                       _sym_hyper_000183600407_00C_0
/oracle/dbs/t_db1.f                       _sym_hyper_000183600407_00C_1
/oracle/dbs/t_db1.f           _sym_subdisk_/dev/vx/rdsk/ipfdg/ipf-vol1_0_1_0
/oracle/dbs/t_db1.f               /dev/vx/rdmp/c2t1d1s2
/oracle/dbs/t_db1.f                   _sym_symdev_000183600407_00D
/oracle/dbs/t_db1.f                       _sym_hyper_000183600407_00D_0
/oracle/dbs/t_db1.f                       _sym_hyper_000183600407_00D_1
```

例 3: データベース・オブジェクトのマッピング

この例では、scott.bonus 表について I/O スタックの全レベルにおけるブロックの分散を表示します。

次のように、最初に MAP_OBJECT() 操作を実行する必要があります。

```
EXECUTE DBMS_STORAGE_MAP.MAP_OBJECT('BONUS','SCOTT','TABLE');
```

問合せは次のとおりです。

```
SELECT io.OBJECT_NAME o_name, io.OBJECT_OWNER o_owner, io.OBJECT_TYPE o_type,
       mf.FILE_NAME, me.ELEM_NAME, io.DEPTH,
      (SUM(io.CU_SIZE * (io.NUM_CU - DECODE(io.PARITY_PERIOD, 0, 0,
                         TRUNC(io.NUM_CU / io.PARITY_PERIOD)))) / 2) o_size
   FROM MAP_OBJECT io, V$MAP_ELEMENT me, V$MAP_FILE mf
   WHERE io.OBJECT_NAME =  'BONUS'
   AND   io.OBJECT_OWNER = 'SCOTT'
   AND   io.OBJECT_TYPE =  'TABLE'
   AND   me.ELEM_IDX = io.ELEM_IDX
   AND   mf.FILE_MAP_IDX = io.FILE_MAP_IDX
   GROUP BY io.ELEM_IDX, io.FILE_MAP_IDX, me.ELEM_NAME, mf.FILE_NAME, io.DEPTH,
         io.OBJECT_NAME, io.OBJECT_OWNER, io.OBJECT_TYPE
   ORDER BY io.DEPTH;
```

問合せの結果は次のとおりです。o_size 列が KB 単位で表されていることに注意してください。

```
O_NAME O_OWNER O_TYPE  FILE_NAME            ELEM_NAME                      DEPTH   O_SIZE
------ ------- ------  -------------------  -----------------------------  ------  ------
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  /dev/vx/dsk/ipfdg/ipf-vol1          0      20
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_plex_/dev/vx/rdsk/ipf          1      20
                                            pdg/if-vol1_-1_-1
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_subdisk_/dev/vx/rdsk/          2      12
                                            ipfdg/ipf-vol1_0_1_0
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_subdisk_/dev/vx/rdsk/ipf       2       8
                                            dg/ipf-vol1_0_2_0
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  /dev/vx/rdmp/c2t1d1s2               3      12
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  /dev/vx/rdmp/c2t1d2s2               3       8
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_symdev_000183600407_00D        4      12
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_symdev_000183600407_00E        4       8
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_hyper_000183600407_00D_0       5      12
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_hyper_000183600407_00D_1       5      12
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_hyper_000183600407_00E_0       6       8
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_hyper_000183600407_00E_1       6       8
```

データファイルのデータ・ディクショナリ・ビュー

次のデータ・ディクショナリ・ビューは、データベースのデータファイルに関して役立つ情報を提供します。

ビュー	説明
DBA_DATA_FILES	属している表領域やファイル ID など、各データファイルに関する記述情報が表示されます。ファイル ID を使用すると、他のビューと結合して詳細情報を得ることができます。
DBA_EXTENTS USER_EXTENTS	DBA ビューには、データベース内のすべてのセグメントを構成するエクステントが表示されます。エクステントを含むデータファイルのファイル ID が含まれます。USER ビューには、現行ユーザーの所有するオブジェクトに属するセグメントのエクステントが表示されます。
DBA_FREE_SPACE USER_FREE_SPACE	DBA ビューには、すべての表領域の使用可能エクステントが表示されます。エクステントを含むデータファイルのファイル ID が含まれます。USER ビューには、現行ユーザーからアクセス可能な表領域の使用可能エクステントが表示されます。
V$DATAFILE	制御ファイル内のデータファイル情報が含まれます。
V$DATAFILE_HEADER	データファイル・ヘッダーからの情報が含まれます。

ここでは、これらのビューの 1 つである V$DATAFILE の使用例を示します。

```
SELECT NAME,
    FILE#,
    STATUS,
    CHECKPOINT_CHANGE# "CHECKPOINT"
  FROM   V$DATAFILE;

NAME                                FILE#     STATUS       CHECKPOINT
---------------------------------   -----     -------       ----------
/u01/oracle/rbdb1/system01.dbf          1     SYSTEM              3839
/u02/oracle/rbdb1/temp01.dbf            2     ONLINE              3782
/u02/oracle/rbdb1/users03.dbf           3     OFFLINE             3782
```

FILE# は各データファイルのファイル番号を示します。データベースとともに作成される SYSTEM 表領域内の最初のデータファイルは、常にファイル 1 になります。STATUS は、データファイルに関する他の情報を示します。データファイルが SYSTEM 表領域の一部である場合、このファイルの STATUS は SYSTEM になります（ただし、ファイルがリカバリを必要とする場合を除きます）。SYSTEM 表領域以外の表領域内のデータファイルがオンラインの場合、このファイルの STATUS は ONLINE になります。SYSTEM 表領域以外の表領域内のデータファイルがオフラインの場合、このファイルの STATUS は OFFLINE または RECOVER になります。CHECKPOINT は、データファイルの最新のチェックポイントに対して書き込まれた最後の SCN（システム変更番号）を示します。

関連項目： これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。

14

UNDO の管理

リリース 11*g* からは、デフォルトのインストールで、Oracle Database によって UNDO が自動的に管理されます。通常、DBA による操作は不要です。ただし、インストールで Oracle Flashback 操作を使用する場合は、それらの操作が正常に終了するように、UNDO 管理タスクをいくつか実行することが必要な場合があります。

この章の内容は次のとおりです。

> **関連項目：** Oracle Database によってデータファイルが作成および管理される UNDO 表領域の作成方法は、第 15 章「Oracle Managed Files の使用」を参照してください。

UNDO の概要

Oracle Database では、データベースの変更をロールバックまたは取り消すために使用する情報を作成して管理します。これらの情報は、主にコミットされる前のトランザクションの処理レコードから構成されます。これらのレコードを総称して **UNDO** と呼びます。

UNDO レコードは次の処理に使用されます。

- ROLLBACK 文を発行したときのトランザクションのロールバック
- データベースのリカバリ
- 読込み一貫性の提供
- Oracle Flashback Query を使用した過去のある時点のデータの分析
- Oracle Flashback 機能を使用した論理的な破損のリカバリ

ROLLBACK 文を発行すると、コミットされていないトランザクションによってデータベースに加えられた変更が、UNDO レコードを使用して取り消されます。データベース・リカバリ時は、REDO ログからデータファイルに適用されたコミットされていない変更が、UNDO レコードを使用してすべて取り消されます。UNDO レコードは、あるユーザーがデータを変更しているときに同じデータに同時にアクセスしようとしている別のユーザーのために、そのデータの変更前のイメージを維持することによって読込み一貫性を提供します。

自動 UNDO 管理の概念

この項では、自動 UNDO 管理の概念について説明します。この項の内容は、次のとおりです。

自動 UNDO 管理の概要

Oracle には、ロールバック情報と領域を管理するための、自動 UNDO 管理と呼ばれる完全に自動化されたメカニズムが用意されています。自動 UNDO 管理では、データベースによって UNDO セグメントが UNDO 表領域で管理されます。リリース 11*g* からは、新しくインストールされたデータベースに対して自動 UNDO 管理がデフォルト・モードになります。Database Configuration Assistant（DBCA）を使用してデータベースを作成すると、UNDOTBS1 という自動拡張可能な UNDO 表領域が自動的に作成されます。

UNDO 表領域は明示的に作成することもできます。UNDO 表領域の作成方法については 14-8 ページの「UNDO 表領域の作成」を参照してください。

インスタンスが起動すると、データベースは最初に使用可能になった UNDO 表領域を自動的に選択します。使用可能な UNDO 表領域がない場合、インスタンスは UNDO 表領域のない状態で起動し、UNDO レコードは SYSTEM 表領域に格納されます。これはお薦めできません。アラート・ログ・ファイルには、システムが UNDO 表領域のない状態で稼働していることを伝える警告メッセージが書き込まれます。

データベースに複数の UNDO 表領域があるときは、必要に応じて、起動時に特定の UNDO 表領域を使用するように指定することもできます。これには、次の例のように、UNDO_TABLESPACE 初期化パラメータを設定します。

```
UNDO_TABLESPACE = undotbs_01
```

初期化パラメータで指定された表領域が存在しない場合、STARTUP コマンドは失敗します。Oracle Real Application Clusters 環境で UNDO_TABLESPACE パラメータを使用すると、インスタンスに特定の UNDO 表領域を割り当てることができます。

データベースは、手動 UNDO 管理モードで実行することもできます。このモードでは、UNDO 領域がロールバック・セグメントを介して管理され、UNDO 表領域は使用されません。

注意： ロールバック・セグメントの領域管理は複雑です。データベースを自動 UNDO 管理モードのままにすることをお薦めします。

次に、UNDO 管理用の初期化パラメータの概要を示します。

初期化パラメータ	説明
UNDO_MANAGEMENT	AUTO または NULL の場合は、自動 UNDO 管理を使用可能にします。MANUAL の場合は、手動 UNDO 管理モードを設定します。デフォルトは AUTO です。
UNDO_TABLESPACE	自動 UNDO 管理モードの場合のみ有効です（オプション）。UNDO 表領域の名前を指定します。データベースに複数の UNDO 表領域があり、データベース・インスタンスで特定の UNDO 表領域を使用するように指定する場合にのみ使用します。

自動 UNDO 管理が使用可能な場合は、初期化パラメータ・ファイルに手動 UNDO 管理に関するパラメータが含まれていても、それらは無視されます。

注意： Oracle Database の以前のリリースでは、手動 UNDO 管理モードがデフォルトです。自動 UNDO 管理モードに変更するには、UNDO 表領域を作成してから UNDO_MANAGEMENT 初期化パラメータを AUTO に変更する必要があります。Oracle Database がリリース 9*i* 以上で、自動 UNDO 管理に変更する場合は、手順について『Oracle Database アップグレード・ガイド』を参照してください。

UNDO_MANAGEMENT 初期化パラメータが NULL の場合、リリース 11*g* 以上では自動 UNDO 管理モードにデフォルト設定されますが、以前のリリースでは手動 UNDO 管理モードにデフォルト設定されます。したがって、以前のリリースをリリース 11*g* にアップグレードする場合は注意が必要です。UNDO 表領域のサイズの設定方法に関する情報など、自動 UNDO 管理モードに移行するための適切な方法について、『Oracle Database アップグレード・ガイド』を参照してください。

関連項目： UNDO 管理で使用する初期化パラメータの詳細は、『Oracle Database リファレンス』を参照してください。

UNDO の保存期間

トランザクションがコミットされると、ロールバックまたはトランザクション・リカバリの実行に UNDO データは不要になります。しかし、長時間実行の問合せ中にデータ・ブロックの変更前のイメージを生成する場合は、読込み一貫性を保証するために古いロールバック情報が必要になることがあります。また、いくつかの Oracle Flashback 機能を正常に終了させるには、古いロールバック情報が必要になる場合があります。このため、古いロールバック情報をできるかぎり長い期間保存することをお薦めします。

自動 UNDO 管理が使用可能な場合は、常に現在の **UNDO 保存期間**が存在します。これは、Oracle Database が古いロールバック情報を上書きするまでの最小保存期間です。現在の UNDO 保存期間よりも古い（コミット済の）ロールバック情報は、「期限切れ」と呼ばれ、その領域は新しいトランザクションで上書きできます。現在の UNDO 保存期間内の古いロールバック情報は、「期限切れでない」と呼ばれ、読取り一貫性と Oracle Flashback 操作のために保存されます。

Oracle Database では、UNDO 表領域サイズとシステム・アクティビティに基づいて、UNDO 保存期間を自動的にチューニングします。必要に応じて UNDO_RETENTION 初期化パラメータを設定することで、最小 UNDO 保存期間（秒単位）を指定できます。次に、UNDO 保存期間にこのパラメータが具体的にどのような影響を与えるかを説明します。

- 固定サイズの UNDO 表領域の場合、UNDO_RETENTION パラメータは無視されます。データベースでは、システム・アクティビティと UNDO 表領域サイズに基づいて、常に最適な保存期間を確保するように UNDO 保存期間をチューニングします。詳細は、14-4 ページの「UNDO の保存期間の自動チューニング」を参照してください。

- AUTOEXTEND オプションが有効な UNDO 表領域の場合、データベースでは、UNDO_RETENTION で指定された最小保存期間を維持しようとします。空き領域が少なくなると、期限切れでないロールバック情報を上書きするかわりに、表領域が自動的に拡張されます。自動拡張可能な UNDO 表領域に対して MAXSIZE 句が指定されている場合は、最大サイズに到達すると、データベースは期限切れでないロールバック情報の上書きを開始する場合があります。DBCA で自動的に作成された UNDOTBS1 表領域は、自動的に拡張します。

UNDO の保存期間の自動チューニング

Oracle Database では、UNDO 表領域の構成方法に基づいて、UNDO 保存期間を自動的にチューニングします。

- UNDO 表領域が AUTOEXTEND オプションで構成されている場合、データベースでは、UNDO の保存期間を、システムでアクティブな最長実行問合せより若干長くなるように動的にチューニングします。ただし、この保存期間は、Oracle Flashback 操作に対応するには不十分な場合があります。Oracle Flashback 操作で「スナップショットが古すぎます」エラーが発生する場合は、これらの操作をサポートするのに十分な UNDO データが保存されるように、ユーザーが介入する必要があることを示しています。Oracle Flashback 機能にさらに対応するには、UNDO_RETENTION パラメータを予想される最長の Oracle Flashback 操作時間と同じ値に設定したり、UNDO 表領域を固定サイズに変更することもできます。
- UNDO 表領域が固定サイズの場合、データベースでは、表領域のサイズと現行のシステムの負荷に対して最適な保存期間を確保するように、UNDO 保存期間を動的にチューニングします。通常、この最適な保存期間は、アクティブな最長実行問合せの期間よりもかなり長くなります。

  UNDO 表領域を固定サイズに変更する場合は、十分な大きさの表領域サイズを選択する必要があります。小さすぎる UNDO 表領域サイズを選択すると、次の 2 つのエラーが発生する可能性があります。

  - DML が失敗する可能性。これは、新しいトランザクションに対する UNDO を格納するための十分な領域がないためです。
  - 長時間実行される問合せが「スナップショットが古すぎます」エラーで失敗する可能性。これは、読取り一貫性に対して不十分な UNDO データがあることを示しています。

  詳細は、14-6 ページの「固定サイズの UNDO 表領域のサイズ変更」を参照してください。

注意： UNDO の保存期間の自動チューニングは、LOB に対してサポートされていません。これは、LOB のロールバック情報が UNDO 表領域ではなく、セグメント自体に格納されるためです。LOB の場合、データベースでは、UNDO_RETENTION で指定された最小保存期間を維持しようとします。ただし、空き領域が少なくなると、期限切れでない LOB のロールバック情報が上書きされる場合があります。

関連項目： 14-6 ページ「最小 UNDO 保存期間の設定」

保存期間の保証

長時間実行される問合せや Oracle Flashback 操作を正常に終了させるために、保存期間の保証を有効化できます。保存期間の保証を有効にすると、指定した UNDO の最小保存期間が保証されます。つまり、UNDO 表領域の領域不足によってトランザクションが失敗した場合でも、期限切れでない UNDO データは上書きされません保存期間の保証を有効にしないと、領域が十分でない場合、期限切れでない UNDO が上書きされるため、システムの UNDO 保存期間が短くなります。デフォルトでは、このオプションは無効になっています。

警告： 保存期間の保証を有効にすると、複数の DML 操作が失敗する可能性があります。この機能は注意して使用してください。

保存期間の保証を有効にするには、CREATE DATABASE 文または CREATE UNDO TABLESPACE 文を使用して UNDO 表領域を作成するときに、その UNDO 表領域に対して RETENTION GUARANTEE 句を指定します。または、後でこの句を ALTER TABLESPACE 文で指定することもできます。保存期間の保証を無効にするには、RETENTION NOGUARANTEE 句を使用します。

DBA_TABLESPACES ビューを使用して、UNDO 表領域の保存期間の保証の設定を確認できます。RETENTION 列には、GUARANTEE、NOGUARANTEE または NOT APPLY（UNDO 表領域以外の表領域で使用）の値が表示されます。

UNDO の保存期間のチューニングとアラートしきい値

固定サイズの UNDO 表領域の場合、データベースでは、データベース統計と UNDO 表領域のサイズに基づいて最適な保存期間を計算します。最適な UNDO 管理を実現するために、データベースでは、UNDO の保存期間を、表領域サイズの 100% ではなく、85% を基にチューニングするか、または使用済領域に対する警告アラートしきい値の率、あるいはいずれか低い方に基づいてチューニングします（警告アラートしきい値のデフォルトは 85% ですが、変更できます）。したがって、UNDO 表領域の警告アラートしきい値を 85% 未満に設定すると、チューニングされる UNDO 保存期間のサイズが小さくなることがあります。表領域のアラートしきい値の詳細は、17-2 ページの「表領域のアラートの管理」を参照してください。

チューニング済 UNDO 保存期間の追跡

現在の保存期間を判別するには、V$UNDOSTAT ビューの TUNED_UNDORETENTION 列を問い合せます。このビューには、10 分単位の統計収集間隔ごとに 1 行のデータが過去 4 日間まで表示されます（過去 4 日間より前のデータは、DBA_HIST_UNDOSTAT ビューに表示されます）。TUNED_UNDORETENTION は秒数で表示されます。

```
select to_char(begin_time, 'DD-MON-RR HH24:MI') begin_time,
to_char(end_time, 'DD-MON-RR HH24:MI') end_time, tuned_undoretention
from v$undostat order by end_time;

BEGIN_TIME      END_TIME        TUNED_UNDORETENTION
--------------- --------------- -------------------
04-FEB-05 00:01 04-FEB-05 00:11               12100
      ...
07-FEB-05 23:21 07-FEB-05 23:31               86700
07-FEB-05 23:31 07-FEB-05 23:41               86700
07-FEB-05 23:41 07-FEB-05 23:51               86700
07-FEB-05 23:51 07-FEB-05 23:52               86700

576 rows selected.
```

V$UNDOSTAT の詳細は、『Oracle Database リファレンス』を参照してください。

最小 UNDO 保存期間の設定

最小 UNDO 保存期間（秒単位）を指定するには、UNDO_RETENTION 初期化パラメータを設定します。14-3 ページの「UNDO の保存期間」で説明されているように、現行の UNDO 保存期間は UNDO_RETENTION よりも大きくなるように、または保存期間の保証が有効な場合を除き、空き領域が少ない場合は UNDO_RETENTION よりも少なくなるように自動的にチューニングされます。

最小 UNDO 保存期間の設定方法

- 次のいずれかを実行します。
  - 初期化パラメータ・ファイルで UNDO_RETENTION を設定します。

    ```
    UNDO_RETENTION = 1800
    ```

  - UNDO_RETENTION は、ALTER SYSTEM 文を使用していつでも変更できます。

    ```
    ALTER SYSTEM SET UNDO_RETENTION = 2400;
    ```

UNDO_RETENTION パラメータの変更は即時に反映されますが、その効果は、現行の UNDO 表領域に十分な領域がある場合のみ表れます。

固定サイズの UNDO 表領域のサイズ変更

UNDO 保存期間の動的チューニングは、通常、固定サイズの UNDO 表領域に有効です。固定サイズの表領域を使用する場合は、UNDO アドバイザを使用すると、必要な容量を見積るのに役立ちます。UNDO アドバイザには、Enterprise Manager または DBMS_ADVISOR PL/SQL パッケージを介してアクセスできます。このアドバイザには、Enterprise Manager を介してアクセスすることをお薦めします。Enterprise Manager を介して UNDO アドバイザを使用する方法の詳細は、『Oracle Database 2 日でデータベース管理者』を参照してください。

UNDO アドバイザは、自動ワークロード・リポジトリ（AWR）に収集されたデータに基づいて分析します。したがって、AWR で適切なワークロード統計を使用可能にし、UNDO アドバイザが正確な推奨事項を作成できるようにすることが重要です。新規作成したデータベースでは、適切な統計が即時に使用できない場合があります。このような場合は、ワークロード・サイクルが 1 回以上完了するまで、引き続きデフォルトの自動拡張可能な UNDO 表領域を使用します。

AWR 統計の収集間隔と保存期間の調整は、アドバイザが作成する推奨事項の精度とタイプに影響します。詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

UNDO アドバイザを使用する場合は、最初に次の 2 つの値を見積ります。

- 最長実行問合せの予想される長さ

  データベースによるワークロード・サイクルが完了すると、「自動 UNDO 管理」ページの「システム・アクティビティ」サブページにある「最長実行問合せ」フィールドを表示できます。

- Oracle Flashback 操作に必要な最長間隔

  たとえば、過去に最大 48 時間の Oracle Flashback 問合せを実行したと予想される場合、Oracle Flashback 要件は 48 時間です。

次に、これら 2 つの値のうち最大値を選択し、その値を UNDO アドバイザへの入力として使用します。

UNDO アドバイザの実行によって、UNDO 表領域のサイズは変更されません。アドバイザは推奨事項を単に返します。表領域のデータファイルを固定サイズに変更するには、ALTER DATABASE 文を使用する必要があります。

次の例では、UNDO 表領域に undotbs.dbf という自動拡張可能なデータファイルあると仮定します。この例では、表領域を 300MB の固定サイズに変更します。

```
ALTER DATABASE DATAFILE '/oracle/dbs/undotbs.dbf' RESIZE 300M;
ALTER DATABASE DATAFILE '/oracle/dbs/undotbs.dbf' AUTOEXTEND OFF;
```

注意： UNDO 表領域を固定サイズに変更する場合は、データベースの作成後、最初に、全ワークロードが実行されるように十分な時間をとることをお薦めします。これによって、UNDO 表領域をワークロードの処理に必要な最小サイズに設定できます。その後、必要に応じて UNDO アドバイザを使用すると、長時間実行される問合せや Oracle Flashback 操作に対応するために、UNDO 表領域のサイズをどのくらい大きく設定するかを決定できます。

関連項目： UNDO アドバイザを使用して UNDO 表領域の最小サイズを計算する手順は、『Oracle Database 2 日でデータベース管理者』を参照してください。

UNDO アドバイザの PL/SQL インタフェース

UNDO アドバイザをアクティブにするには、アドバイザ・フレームワークを使用して UNDO アドバイザ・タスクを作成します。次の例では、UNDO 表領域を評価するための UNDO アドバイザ・タスクを作成します。アドバイザの名前は 'Undo Advisor' です。分析は自動ワークロード・リポジトリのスナップショットに基づいて実行されます。このスナップショットは、START_SNAPSHOT パラメータと END_SNAPSHOT パラメータを設定して指定する必要があります。次の例では、START_SNAPSHOT が 1 で、END_SNAPSHOT は 2 です。

```
DECLARE
   tid    NUMBER;
   tname  VARCHAR2(30);
   oid    NUMBER;
BEGIN
   DBMS_ADVISOR.CREATE_TASK('Undo Advisor', tid, tname, 'Undo Advisor Task');
   DBMS_ADVISOR.CREATE_OBJECT(tname, 'UNDO_TBS', null, null, null, 'null', oid);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'TARGET_OBJECTS', oid);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'START_SNAPSHOT', 1);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'END_SNAPSHOT', 2);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'INSTANCE', 1);
   DBMS_ADVISOR.execute_task(tname);
END;
/
```

アドバイザ・タスクを作成した後は、Enterprise Manager の自動データベース診断モニターに出力および推奨事項を表示できます。この情報は、DBA_ADVISOR_* データ・ディクショナリ・ビュー（DBA_ADVISOR_TASKS、DBA_ADVISOR_OBJECTS、DBA_ADVISOR_FINDINGS、DBA_ADVISOR_RECOMMENDATIONS など）にも表示されます。

関連項目：

- 様々なアドバイザにアドバイザ・タスクを作成する例は、17-13 ページの「セグメント・アドバイザの使用」を参照してください。
- DBA_ADVISOR_* データ・ディクショナリ・ビューについては、『Oracle Database リファレンス』を参照してください。

UNDO 表領域の管理

この項では、UNDO 表領域を管理する際に必要な手順について説明します。この項の内容は、次のとおりです。

- UNDO 表領域の作成
- UNDO 表領域の変更
- UNDO 表領域の削除
- UNDO 表領域の切替え
- UNDO 領域に対するユーザー割当ての確立
- UNDO 領域のデータ・ディクショナリ・ビュー

UNDO 表領域の作成

Oracle Database リリース 11*g* の新規インストールでは、Database Configuration Assistant（DBCA）によって UNDO 表領域が自動的に作成されますが、UNDO 表領域を手動で作成することが必要な場合があります。

UNDO 表領域の作成には 2 つの方法があります。1 つは、CREATE DATABASE 文の発行時に UNDO 表領域を作成する方法です。これは、データベースを新規作成中にインスタンスが自動 UNDO 管理モードで起動したとき（UNDO_MANAGEMENT = AUTO）に実行されます。もう 1 つは、既存のデータベースで使用する方法です。この場合は CREATE UNDO TABLESPACE 文を使用します。

UNDO 表領域にはデータベース・オブジェクトは作成できません。UNDO 表領域は、システムが管理している UNDO データ用に予約されています。

Oracle Database では、単一ファイル UNDO 表領域を作成できます。単一ファイル（大型ファイル）表領域については、12-6 ページの「大型ファイル表領域」を参照してください。

CREATE DATABASE を使用した UNDO 表領域の作成

CREATE DATABASE 文で UNDO TABLESPACE 句を使用すると、特定の UNDO 表領域を作成できます。

次の文は、CREATE DATABASE 文での UNDO TABLESPACE 句の使用例を示しています。ここでは、UNDO 表領域に undotbs_01 という名前を付け、/u01/oracle/rbdb1/undo0101.dbf という 1 つのデータファイルを割り当てています。

```
CREATE DATABASE rbdb1
     CONTROLFILE REUSE
     .
     .
     .
     UNDO TABLESPACE undotbs_01 DATAFILE '/u01/oracle/rbdb1/undo0101.dbf';
```

CREATE DATABASE の実行中に UNDO 表領域を正常に作成できない場合は、CREATE DATABASE 操作全体が失敗します。データベース・ファイルをクリーン・アップし、エラーを訂正して、再度 CREATE DATABASE 操作を実行する必要があります。

また、データベース作成時に、CREATE DATABASE 文を使用して単一ファイル UNDO 表領域を作成できます。この操作については、2-21 ページの「データベース作成時の大型ファイル表領域のサポート」を参照してください。

> **関連項目：** CREATE DATABASE 文を使用して UNDO 表領域を作成する構文については、『Oracle Database SQL リファレンス』を参照してください。

CREATE UNDO TABLESPACE 文の使用

CREATE UNDO TABLESPACE 文は CREATE TABLESPACE 文とほぼ同じですが、UNDO キーワードを指定します。UNDO 表領域の属性のほとんどはデータベースが決定しますが、データベース管理者は DATAFILE 句を指定できます。

この例では、AUTOEXTEND オプションを使用して undotbs_02 UNDO 表領域を作成しています。

```
CREATE UNDO TABLESPACE undotbs_02
     DATAFILE '/u01/oracle/rbdb1/undo0201.dbf' SIZE 2M REUSE AUTOEXTEND ON;
```

複数の UNDO 表領域を作成できますが、UNDO 表領域は 1 つのみアクティブにできます。

> **関連項目：** CREATE UNDO TABLESPACE 文を使用して UNDO 表領域を作成する構文については、『Oracle Database SQL リファレンス』を参照してください。

UNDO 表領域の変更

UNDO 表領域を変更するには、ALTER TABLESPACE 文を使用します。ただし、UNDO 表領域のほとんどはシステムが管理しているため、考慮が必要になるのは次の操作のみです。

- データファイルの追加
- データファイルの名前変更
- データファイルのオンライン化またはオフライン化
- データファイルのオープン状態のバックアップの開始または終了
- UNDO の保存期間の保証の有効化または無効化

DBA が変更可能な属性もこれらの属性のみです。

UNDO 表領域が領域不足の場合、または領域不足の発生を防止する場合は、さらにファイルを追加したり、既存のデータファイルのサイズを変更できます。

次の例では、UNDO 表領域 undotbs_01 にデータファイルを 1 つ追加しています。

```
ALTER TABLESPACE undotbs_01
     ADD DATAFILE '/u01/oracle/rbdb1/undo0102.dbf' AUTOEXTEND ON NEXT 1M
         MAXSIZE UNLIMITED;
```

ALTER DATABASE...DATAFILE 文を使用すると、データファイルのサイズを変更または拡張できます。

> **関連項目：**
>
> - 13-5 ページ「データファイルのサイズ変更」
> - ALTER TABLESPACE の構文については、『Oracle Database SQL リファレンス』を参照してください。

UNDO 表領域の削除

UNDO 表領域を削除するには、DROP TABLESPACE 文を使用します。次の例では、UNDO 表領域 undotbs_01 を削除しています。

```
DROP TABLESPACE undotbs_01;
```

UNDO 表領域は、現在どのインスタンスでも使用されていない場合にのみ削除できます。UNDO 表領域に処理中のトランザクションが含まれている場合（トランザクションが失敗してまだリカバリされていない場合など）、DROP TABLESPACE 文は失敗します。しかし、DROP TABLESPACE は、UNDO 表領域に期限切れでない（保存期間内である）ロールバック情報が含まれている場合でも UNDO 表領域を削除するため、既存の問合せでロールバック情報を必要とする場合は、UNDO 表領域を削除しないように注意する必要があります。

UNDO 表領域に対する DROP TABLESPACE は、DROP TABLESPACE...INCLUDING CONTENTS と同じように動作します。つまり、UNDO 表領域の内容はすべて削除されます。

> **関連項目：** DROP TABLESPACE の構文については、『Oracle Database SQL リファレンス』を参照してください。

UNDO 表領域の切替え

ある UNDO 表領域から別の UNDO 表領域に切り替えることができます。UNDO_TABLESPACE 初期化パラメータは動的パラメータであるため、ALTER SYSTEM SET 文を使用して新しい UNDO 表領域を割り当てることができます。

次の文は、新しい UNDO 表領域に切り替えます。

```
ALTER SYSTEM SET UNDO_TABLESPACE = undotbs_02;
```

undotbs_01 が現行の UNDO 表領域であるとすると、このコマンドが正常に実行された後、インスタンスは undotbs_01 のかわりに undotbs_02 を UNDO 表領域として使用します。

切替え先の表領域が次のいずれかの条件を満たす場合はエラーがレポートされ、切替えは行われません。

- 表領域が存在しない場合
- 表領域が UNDO 表領域ではない場合
- 表領域が別のインスタンスによってすでに使用されている場合（RAC 環境のみ）

切替え操作が実行されている間、データベースはオンラインであり、このコマンドの実行中でもユーザー・トランザクションを実行できます。切替え操作が正常に完了すると、切替え操作開始後に開始されたすべてのトランザクションが新しい UNDO 表領域内のトランザクション表に割り当てられます。

切替え操作は、古い UNDO 表領域内のトランザクションがコミットされるまで待機しません。古い UNDO 表領域内に未処理のトランザクションがある場合、古い UNDO 表領域は PENDING OFFLINE モード（状態）になります。このモードでは、既存のトランザクションは引き続き実行できますが、新しいユーザー・トランザクションの UNDO レコードをこの UNDO 表領域に格納することはできません。

UNDO 表領域は、切替え操作が正常に完了した後も、この PENDING OFFLINE モードのまま存在できます。PENDING OFFLINE の UNDO 表領域は、別のインスタンスが使用することも、削除することもできません。最終的に、すべてのアクティブなトランザクションがコミットされた後、UNDO 表領域は自動的に PENDING OFFLINE モードから OFFLINE モードに移行します。それ以降は、他のインスタンスが（Oracle Real Application Clusters 環境で）その UNDO 表領域を使用できます。

UNDO TABLESPACE のパラメータ値を「''」（2 つの一重引用符）に設定した場合は、現行の UNDO 表領域が次の使用可能な UNDO 表領域に切り替えられます。使用可能な UNDO 表領域がない場合もあるため、この文の使用には注意が必要です。

次の例では、現行の UNDO 表領域の割当てを解除しています。

```
ALTER SYSTEM SET UNDO_TABLESPACE = '';
```

UNDO 領域に対するユーザー割当ての確立

Oracle Database Resource Manager を使用すると、UNDO 領域に対するユーザー割当てを確立できます。DBA は、データベース・リソース・マネージャのディレクティブ UNDO_POOL を使用して、ユーザーのグループ（リソース・コンシューマ・グループ）が消費する UNDO 表領域の量を制限できます。

UNDO プールは、コンシューマ・グループごとに指定できます。UNDO プールによって、コンシューマ・グループが生成できる UNDO の合計量が制御されます。コンシューマ・グループが生成する UNDO の合計量がその UNDO 制限を超えると、UNDO を生成している現行の UPDATE トランザクションが終了します。コンシューマ・グループの他のメンバーは、UNDO 領域がプールから解放されるまで、新たに更新を実行できなくなります。

UNDO_POOL ディレクティブが明示的に定義されていないときは、ユーザーは無制限に UNDO 領域を使用できます。

関連項目： 第 25 章「Oracle Database Resource Manager を使用したリソース割当ての管理」

UNDO 表領域に対する領域のアラートしきい値の管理

Oracle Database では、表領域に使用可能な領域が少なくなると事前にアラートが生成されるため、表領域のディスク領域の使用状況を管理するために役立ちます。UNDO 表領域のアラートのしきい値の設定方法は、17-2 ページの「表領域のアラートの管理」を参照してください。

Oracle Database では、UNDO 領域の事前アラート以外に、「スナップショットが古すぎます」エラーが発生するような長時間実行される問合せがシステムにある場合にもアラートが生成されます。過剰にアラートが生成されるのを防ぐため、長時間実行される問合せのアラートは 24 時間以上の間隔をおいて発行されます。アラートが生成された場合は、Enterprise Manager の「Undo Advisor」ページをチェックして UNDO 表領域に関する詳細情報を参照できます。

自動 UNDO 管理への移行

現在、ロールバック・セグメントを使用して UNDO 領域を管理している場合は、データベースを自動 UNDO 管理に移行することをお薦めします。

詳細は、『Oracle Database アップグレード・ガイド』を参照してください。

UNDO 領域のデータ・ディクショナリ・ビュー

ここでは、自動 UNDO 管理モードにおいて UNDO 領域に関する情報を表示する際に役立つビューについて説明し、いくつかの例を示します。ここで紹介したビュー以外にも、表領域やデータファイルの情報を表示するビューを使用して、情報を取得できます。これらのビューの詳細は、13-25 ページの「データファイルのデータ・ディクショナリ・ビュー」を参照してください。

UNDO 表領域に関する領域情報を取得するために、次の動的パフォーマンス・ビューが役立ちます。

ビュー	説明
V$UNDOSTAT	UNDO 領域の監視とチューニングのための統計情報が含まれます。このビューは、現行の作業負荷に必要な UNDO 領域の量を見積る際に利用できます。また、データベースはこの情報を使用して、システム内の UNDO の使用方法をチューニングします。このビューの情報は、自動 UNDO 管理モードでのみ意味があります。
V$ROLLSTAT	自動 UNDO 管理モードの場合、このビューの情報は、UNDO 表領域内の UNDO セグメントの動作を反映します。
V$TRANSACTION	UNDO セグメント情報が含まれます。

ビュー	説明
DBA_UNDO_EXTENTS	UNDO 表領域内の各エクステントの状態およびサイズを示します。
DBA_HIST_UNDOSTAT	V$UNDOSTAT 情報の統計スナップショットが含まれます。詳細は、『Oracle Database 2 日でデータベース管理者』を参照してください。

関連項目： 自動 UNDO 管理モードで使用するビューの詳細は、『Oracle Database リファレンス』を参照してください。

V$UNDOSTAT ビューは、現行インスタンス内の UNDO 領域におけるトランザクションの実行の効果を監視する際に役立ちます。UNDO 領域の消費、トランザクションの同時実行性、UNDO 保存期間のチューニング、インスタンス内の長時間実行される問合せの長さおよび SQL ID に関する統計が使用できます。

ビュー内の各行には、インスタンス内で 10 分ごとに収集された統計が表示されます。行は、BEGIN_TIME 列の値の降順に並びます。各行は、BEGIN_TIME と END_TIME によってマーク付けされた時間間隔に基づいています。各列は、その時間間隔で収集された特定の統計データを表します。ビューの最初の行には、（部分的な）現在の時間間隔に対応する統計が含まれます。ビューには、4 日周期にわたる合計 576 の行があります。

次の例は、V$UNDOSTAT ビューに対する問合せの結果を示したものです。

```
SELECT TO_CHAR(BEGIN_TIME, 'MM/DD/YYYY HH24:MI:SS') BEGIN_TIME,
       TO_CHAR(END_TIME, 'MM/DD/YYYY HH24:MI:SS') END_TIME,
       UNDOTSN, UNDOBLKS, TXNCOUNT, MAXCONCURRENCY AS "MAXCON"
       FROM v$UNDOSTAT WHERE rownum <= 144;

BEGIN_TIME          END_TIME              UNDOTSN   UNDOBLKS   TXNCOUNT     MAXCON
------------------- ------------------- ---------- ---------- ---------- ----------
10/28/2004 14:25:12 10/28/2004 14:32:17          8         74   12071108          3
10/28/2004 14:15:12 10/28/2004 14:25:12          8         49   12070698          2
10/28/2004 14:05:12 10/28/2004 14:15:12          8        125   12070220          1
10/28/2004 13:55:12 10/28/2004 14:05:12          8         99   12066511          3
...
10/27/2004 14:45:12 10/27/2004 14:55:12          8         15   11831676          1
10/27/2004 14:35:12 10/27/2004 14:45:12          8        154   11831165          2

144 rows selected.
```

この例は、10/27/2004 の 14:35:12 から 24 時間前までの間に、システムで UNDO 領域がどのように消費されたのかを示しています。

15

Oracle Managed Files の使用

この章の内容は次のとおりです。

Oracle Managed Files の概要

Oracle Managed Files を使用すると、Oracle Database の管理が容易になります。Oracle Database を構成するオペレーティング・システム・ファイルをデータベース管理者（DBA）が直接管理する必要はありません。Oracle Managed Files を使用して、ファイル・システム・ディレクトリを指定します。これらのディレクトリでは、データベースによって、ファイルがデータベース・オブジェクト・レベルで自動的に作成、命名および管理されます。たとえば、指定する必要があるのは表領域の作成のみで、DATAFILE 句で表領域のデータファイルの名前とパスを指定する必要はありません。この機能では、論理ボリューム・マネージャ（LVM）が効果的に使用されます。

データベースでは内部的に標準ファイル・システム・インタフェースを使用し、必要に応じて、次のデータベース構造のファイルを作成および削除します。

- 表領域
- REDO ログ・ファイル
- 制御ファイル
- アーカイブ・ログ
- ブロック・チェンジ・トラッキング・ファイル
- フラッシュバック・ログ
- Recovery Manager によるバックアップ

初期化パラメータによって、特定のタイプのファイルに使用するファイル・システム・ディレクトリを指定します。これにより、一意の Oracle Managed Files が作成され、不要になると削除されます。

この機能は、トレース・ファイル、監査ファイル、アラート・ログおよびコア・ファイルなどの管理ファイルの作成および命名には影響を与えません。

> **関連項目：** Oracle Managed Files の機能を拡張する自動ストレージ管理（ASM）、Oracle Database の統合ファイル・システムおよびボリューム・マネージャについては、『Oracle Database ストレージ管理者ガイド』を参照してください。Oracle Managed Files によって、ファイルは自動的に作成および管理されますが、ASM では、ストライプ化、ソフトウェアのミラー化、動的な記憶域の構成などの追加機能を利用でき、サード・パーティ製の論理ボリューム・マネージャを購入する必要がありません。

Oracle Managed Files の使用対象

Oracle Managed Files は、次のタイプのデータベースに適しています。

- 次のものによってサポートされるデータベース
  - ストライプ化 /RAID および動的に拡張可能な論理ボリュームをサポートする論理ボリューム・マネージャ
  - サイズが大きく拡張可能なファイルを提供するファイル・システム
- ローエンド・データベースまたはテスト・データベース

Oracle Managed Files 機能の目的は、RAW ディスクを使用したシステムの管理を容易にすることではありません。この機能は、ディスク領域割当てについて、オペレーティング・システム機能との統合を実現します。オペレーティング・システムは RAW ディスクの割当てをサポートしていない（割当ては手動で行う）ので、この機能は利用できません。その一方で、Oracle Managed Files では（RAW ディスクではなく）必ずオペレーティング・システムのファイル・システムを使用する必要があるため、ディスク上のファイルの配置方法を DBA が管理できず、一部の I/O をチューニングできなくなります。

論理ボリューム・マネージャ（LVM）の概要

LVM は、ほとんどのオペレーティング・システムで利用できるソフトウェア・パッケージです。論理ディスク・マネージャ（LDM）と呼ばれることもあります。LVM を使用すると、複数の物理ディスクの断片を、ソフトウェアの高層部で 1 つのディスクとして表される、連続した単一のアドレス空間に結合できます。LVM では、基盤となるどの物理ディスクよりも、容量、パフォーマンス、信頼性および可用性の特性に優れた論理ボリュームを作成できます。LVM は、ミラー化、ストライプ化、連結および RAID5 などの手法を使用して、これらの特性を実装します。

LVM の中には、論理ボリュームを作成後、そのボリュームの使用中に特性を変更できるものがあります。ボリュームはサイズ変更やミラー化できるだけでなく、別の物理ディスクに再配置することもできます。

ファイル・システムの概要

ファイル・システムとは、連続するディスク・アドレス空間内に構築されたデータ構造です。ファイル・マネージャ（FM）は、ファイル・システムを操作するソフトウェア・パッケージですが、これがファイル・システムと呼ばれる場合もあります。オペレーティング・システムには必ず FM が組み込まれています。FM の主要なタスクは、ファイル・システム内のファイルにディスク領域への割当てまたは割当て解除です。

ファイル・システムを使用すると、多数のファイルにディスク領域を割り当てることができます。各ファイルは、Oracle Database などのアプリケーションに連続するアドレス空間を提供するために作成されます。実際には、ファイル・システムのディスク領域の中でファイルは連続していない場合があります。ファイルは、作成、読取り、書込み、サイズ変更および削除ができます。各ファイルには対応する名前があり、ファイルを参照する際に使用します。

ファイル・システムは通常、LVM が作成する論理ボリュームの最上部に構築されます。したがって、特定のファイル・システム内にあるファイルはすべて、基盤となる論理ボリュームから継承された同じパフォーマンス、信頼性および可用性の特性を持ちます。ファイル・システムは、その中のすべてのファイルによって共有される、単一の記憶域のプールです。ファイル・システムの領域がなくなると、そのファイル・システム内にあるファイルを増やすことはできません。1 つのファイル・システムで使用可能な領域が、他のファイル・システムの領域に影響を及ぼすことはありません。ただし、LVM と FM の組合せによっては、ファイル・システムの領域を追加または削除できます。

オペレーティング・システムは、複数のファイル・システムをサポートできます。別々のファイルに異なる記憶特性を与える場合や、使用可能なディスク領域を分割して互いに影響を及ぼさないプールを作成する場合に、複数のファイル・システムが作成されます。

Oracle Managed Files の使用上の利点

Oracle Managed Files を使用すると、次のような利点があります。

- データベースの管理が容易になります。

  ファイル名を考えて、特定の記憶域要件を定義する必要はありません。一貫性のある一連のルールに基づいて、すべての関連ファイルが命名されます。記憶域の特性と記憶域を割り当てるプールは、ファイル・システムによって定義されます。

- 管理者による誤ったファイルの指定が原因で破損することが少なくなります。

  Oracle Managed Files とファイル名はすべて一意です。一般的によくある間違いは、2 つの異なるデータベースで同じファイルを使用することであり、これが長時間にわたるシステム・ダウンを引き起こし、コミット済トランザクションが失われる原因となります。1 つのファイルを参照するために 2 つの異なる名前を使用することは、重大な破損の原因となるもう 1 つの間違いです。

- 不要なファイルの存在によるディスク領域の浪費が減少します。

  Oracle Database では、Oracle Managed Files が不要になったとき、古いファイルが自動的に削除されます。大規模なシステムでは、特定のファイルがまだ必要かどうか誰も確信できないという理由だけで、大量のディスク領域が浪費されています。Oracle Managed Files

の削除機能は、ディスク上の不要ファイルの削除という管理タスクを容易にし、ファイルを誤って削除することを防止します。

- テスト・データベースおよび開発データベースを容易に作成できます。

  ファイル構造と命名について検討する時間を最小限にとどめることができ、実行するファイル管理タスクも従来より少なくて済みます。これにより、テスト・データベースまたは開発データベースの実際の要件を満たす作業に集中できます。

- 移植可能なサード・パーティ製ツールの開発が容易になります。

  Oracle Managed Files では、SQL スクリプト内でオペレーティング・システム固有のファイル名を指定する必要がありません。

Oracle Managed Files と既存の機能

Oracle Managed Files を使用しても、既存の機能が不要になるわけではありません。既存データベースは、常に従来どおり操作できます。古いファイルはそれまでの方法で管理し、その一方で新しいファイルは管理ファイルとして作成できます。したがって、データベースには Oracle Managed Files とそれ以外のファイルがともに存在する状態になります。

Oracle Managed Files の作成および使用の有効化

次の表に、Oracle Managed Files の使用を有効化する初期化パラメータを示します。

初期化パラメータ	説明
DB_CREATE_FILE_DEST	作成操作でファイル仕様を指定しなかった場合に、データベースによってデータファイルまたは一時ファイルが作成されるデフォルトのファイル・システム・ディレクトリまたは ASM ディスク・グループの場所を定義します。 DB_CREATE_ONLINE_LOG_DEST_*n* を指定していない場合は、REDO ログ・ファイルおよび制御ファイルのデフォルトの場所としても使用されます。
DB_CREATE_ONLINE_LOG_DEST_*n*	作成操作でファイル仕様を指定しなかった場合に、REDO ログ・ファイルおよび制御ファイルが作成される、デフォルトのファイル・システム・ディレクトリまたは ASM ディスク・グループの場所を定義します。この初期化パラメータは、*n* を変更することで複数回使用できます。その際は、*n* に REDO ログ・ファイルまたは制御ファイルの多重コピーを指定します。多重コピーは最大 5 つ指定できます。
DB_RECOVERY_FILE_DEST	データベースによって Recovery Manager によるバックアップ（フォーマット・オプションが使用されていない場合）、アーカイブ・ログ（他のローカル・アーカイブ先が構成されていない場合）およびフラッシュバック・ログが作成されるデフォルトのファイル・システム・ディレクトリまたは ASM ディスク・グループであるフラッシュ・リカバリ領域の場所を定義します。DB_CREATE_ONLINE_LOG_DEST_*n* を指定していない場合は、REDO ログ・ファイルおよび制御ファイルまたはその多重コピーのデフォルトの場所としても使用されます。

これらのパラメータで指定したファイル・システム・ディレクトリは、すでに存在している必要があります。データベースはディレクトリを作成しません。ディレクトリには、データベースによるファイル作成を可能にする権限が必要です。

ファイル作成操作で場所を明示的に指定しなかった場合は、必ずデフォルトの場所が使用されます。ファイル名はデータベースが作成するため、作成されたファイルは Oracle Managed Files になります。

これら 2 つの初期化パラメータはどちらも動的であり、`ALTER SYSTEM` または `ALTER SESSION` 文を使用して設定できます。

関連項目：

- 初期化パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 15-7 ページ「Oracle Managed Files の命名方法」

DB_CREATE_FILE_DEST 初期化パラメータの設定

初期化パラメータ・ファイルに `DB_CREATE_FILE_DEST` 初期化パラメータを設定して、データベースが次のファイルを作成するデフォルトの場所を識別できるようにします。

- データファイル
- 一時ファイル
- REDO ログ・ファイル
- 制御ファイル
- ブロック・チェンジ・トラッキング・ファイル

ファイル・システム・ディレクトリの名前を指定して、これらに対するオペレーティング・システム・ファイルを作成するためのデフォルトの場所にします。次の例では、Oracle Managed Files を作成する際に使用するデフォルトのディレクトリとして、`/u01/app/oracle/oradata` を設定しています。

```
DB_CREATE_FILE_DEST = '/u01/app/oracle/oradata'
```

DB_RECOVERY_FILE_DEST パラメータの設定

初期化パラメータ・ファイルに `DB_RECOVERY_FILE_DEST` および `DB_RECOVERY_FILE_DEST_SIZE` パラメータを設定して、フラッシュ・リカバリ領域のデフォルトの場所を識別できるようにします。フラッシュ・リカバリ領域には、次が格納されます。

- REDO ログ・ファイルまたはその多重コピー
- 制御ファイルまたはその多重コピー
- Recovery Manager によるバックアップ（データファイルのコピー、制御ファイルのコピー、バックアップ・ピース、制御ファイルの自動バックアップ）
- アーカイブ・ログ
- フラッシュバック・ログ

ファイル・システム・ディレクトリの名前を指定して、これらに対するオペレーティング・システム・ファイルを作成するためのデフォルトの場所にします。次に例を示します。

```
DB_RECOVERY_FILE_DEST      = '/u01/app/oracle/flash_recovery_area'
DB_RECOVERY_FILE_DEST_SIZE = 20G
```

DB_CREATE_ONLINE_LOG_DEST_n 初期化パラメータの設定

初期化パラメータ・ファイルに DB_CREATE_ONLINE_LOG_DEST_*n* 初期化パラメータを設定して、データベースが次のファイルを作成するデフォルトの場所を識別できるようにします。

- REDO ログ・ファイル
- 制御ファイル

ファイル・システム・ディレクトリまたは ASM ディスク・グループの名前を指定して、これらに対するファイルを作成するためのデフォルトの場所にします。多重コピーを配置する場所は最大 5 つまで指定できます。

REDO ログ・ファイルおよび制御ファイルを作成する場合のみ、このパラメータは、DB_CREATE_FILE_DEST および DB_RECOVERY_FILE_DEST 初期化パラメータで指定されているデフォルトの場所を上書きします。DB_CREATE_FILE_DEST パラメータを指定せず、DB_CREATE_ONLINE_LOG_DEST_*n* パラメータのみを指定している場合は、REDO ログ・ファイルと制御ファイルのみが Oracle Managed Files として作成されます。

少なくとも 2 つのパラメータを指定することをお薦めします。次に例を示します。

```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
```

これによって、ファイルを多重化できるため、REDO ログ・ファイルまたは制御ファイルの保存先のどちらかで障害が発生した場合のフォルト・トレランスが向上します。

Oracle Managed Files の作成

次の条件のいずれかを満たしている場合で、作成操作でファイル仕様を指定しなかったときに、必要に応じて Oracle Database によって Oracle Managed Files が作成されます。

- 初期化パラメータ・ファイルに、DB_CREATE_FILE_DEST、DB_REDOVERY_FILE_DEST または DB_CREATE_ONLINE_LOG_DEST_*n* 初期化パラメータのいずれかを指定している場合
- DB_RECOVERY_FILE_DEST、DB_CREATE_FILE_DEST または DB_CREATE_ONLINE_LOG_DEST_*n* 初期化パラメータのいずれかを動的に設定するために、ALTER SYSTEM 文を発行した場合
- DB_CREATE_FILE_DEST、DB_RECOVERY_FILE_DEST または DB_CREATE_ONLINE_LOG_DEST_*n* 初期化パラメータのいずれかを動的に設定するために、ALTER SESSION 文を発行した場合

Oracle Managed Files を作成する文がエラーを検出した場合、またはなんらかの障害のために完了しなかった場合は、その文によって作成された Oracle Managed Files はすべて、エラーまたは障害のリカバリの一部として自動的に削除されます。ただし、ファイル・システムやストレージ・サブシステムで発生する多数の潜在的なエラーが原因で、オペレーティング・システムのコマンドを使用した手動でのファイル削除が必要になる場合があります。

この項の内容は、次のとおりです。

Oracle Managed Files の命名方法

> **注意：** この項で説明する命名方法は、オペレーティング・システムのファイル・システムに作成されるファイルにのみ適用されます。自動ストレージ管理（ASM）ディスク・グループに作成されるファイルの命名方法については、『Oracle Database ストレージ管理者ガイド』を参照してください。

Oracle Managed Files のファイル名は、ファイル命名に関する Optimal Flexible Architecture（OFA）標準に準拠しています。割り当てられた名前は、次の要件を満たしています。

- データベース・ファイルが他のすべてのファイルと容易に区別できます。
- 1 つのデータベース・タイプのファイルが、他のデータベース・タイプのファイルと容易に区別できます。
- ファイルはファイル・タイプ固有の重要な属性に明確に関連付けられます。たとえば、データファイルの表領域への関連付けを容易にするためにデータファイル名に表領域名を含めたり、アーカイブ・ログ名にスレッド、順序および作成日を含めることが可能です。

同じ名前を持つ Oracle Managed Files は 1 つもありません。Oracle Managed Files の作成に使用される名前は、次の 3 つのソースから構成されます。

- デフォルトの作成場所。
- ファイルのタイプに基づいて選択されたファイル名テンプレート。テンプレートは、オペレーティング・システムのプラットフォームや、自動ストレージ管理を使用しているかどうかによって異なります。
- Oracle Database またはオペレーティング・システムによって作成された一意の文字列。これにより、ファイル作成によって既存のファイルが破損しないこと、および誤って他のファイルが選択されないことが保証されます。

特定の例として、Solaris ファイル・システムにおける Oracle Managed Files のファイル名の書式を次に示します。

```
<destination_prefix>/o1_mf_%t_%u_.dbf
```

各項目の意味は次のとおりです。

- *<destination_prefix>* は *<destination_location>*/*<db_unique_name>*/*<datafile>* です。

  各項目の意味は次のとおりです。

  - *<destination_location>* は、DB_CREATE_FILE_DEST に指定した場所です。
  - *<db_unique_name>* は、ターゲット・データベースのグローバルな一意の名前（DB_UNIQUE_NAME 初期化パラメータ）です。DB_UNIQUE_NAME パラメータがない場合は、DB_NAME 初期化パラメータ値が使用されます。
- %t は表領域の名前を表します。
- %u は一意性を保証する 8 文字の文字列を表します。

たとえば、次のようなパラメータ設定を考えてみます。

```
DB_CREATE_FILE_DEST   = /u01/app/oracle/oradata
DB_UNIQUE_NAME = PAYROLL
```

この例のデータファイル名は次のようになります。

```
/u01/app/oracle/oradata/PAYROLL/datafile/o1_mf_tbs1_2ixh90q_.dbf
```

他のファイル・タイプの名前もほぼ同じです。他のプラットフォームでもファイル名はほぼ同じですが、各プラットフォームのネーミング規則の制約を受けます。

後続の例では、Solaris ファイル・システムで OMF アーカイブ先として表示される Oracle Managed Files の名前を使用します。

注意： Oracle Managed Files の名前は変更しないでください。Oracle Managed Files は名前に基づいて識別されます。ファイル名を変更すると、データベースでは Oracle Managed Files として認識できなくなり、ファイルは適切に管理されません。

データベース作成時の Oracle Managed Files の作成

ここでは、Oracle Managed Files 使用時の CREATE DATABASE 文の処理について説明します。

注意： この項で説明するルールとデフォルトは、Database Configuration Assistant（DBCA）によるデータベースの作成にも適用されます。DBCA では、グラフィカル・インタフェースを使用して Oracle Managed Files を有効化し、この項で説明する初期化パラメータに対応するファイルの場所を指定できます。

関連項目： CREATE DATABASE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

データベース作成時の制御ファイルの指定

データベース作成時には、CONTROL_FILES 初期化パラメータによって指定されたファイルで、制御ファイルが作成されます。CONTROL_FILES パラメータが設定されておらず、Oracle Managed Files の作成に必要な初期化パラメータが少なくとも 1 つ設定されている場合は、制御ファイルのデフォルトの保存先に Oracle Managed Files の制御ファイルが作成されます。デフォルトの保存先は、次の優先度に従って定義されます。

- DB_CREATE_ONLINE_LOG_DEST_*n* 初期化パラメータで指定されている 1 つ以上の制御ファイル。最初のディレクトリに作成されたファイルが主制御ファイルになります。DB_CREATE_ONLINE_LOG_DEST_*n* が指定されている場合、DB_CREATE_FILE_DEST または DB_RECOVERY_FILE_DEST（フラッシュ・リカバリ領域）に制御ファイルは作成されません。
- DB_CREATE_ONLINE_LOG_DEST_*n* に値が指定されていない場合で、DB_CREATE_FILE_DEST および DB_RECOVERY_FILE_DEST の両方に値が設定されている場合は、それぞれの場所に 1 つの制御ファイルが作成されます。DB_CREATE_FILE_DEST に指定された場所が主制御ファイルの場所になります。
- DB_CREATE_FILE_DEST に対してのみ値が指定されている場合は、その場所に 1 つの制御ファイルが作成されます。
- DB_RECOVERY_FILE_DEST に対してのみ値が指定されている場合は、その場所に 1 つの制御ファイルが作成されます。

CONTROL_FILES パラメータが設定されておらず、前述の初期化パラメータがいずれも設定されていない場合、Oracle Database のデフォルトの処理はオペレーティング・システムによって異なります。少なくとも 1 つの制御ファイルのコピーが、オペレーティング・システム固有のデフォルトの場所に作成されます。この方法で作成された制御ファイルのコピーは Oracle Managed Files ではありません。そのため、初期化パラメータ・ファイルに CONTROL_FILES 初期化パラメータを追加する必要があります。

Oracle Managed Files の制御ファイルが作成された場合で、サーバー・パラメータ・ファイルが存在するときは、サーバー・パラメータ・ファイルに CONTROL_FILES 初期化パラメータのエントリが追加されます。サーバー・パラメータ・ファイルが存在しない場合は、CONTROL_FILES 初期化パラメータのエントリをテキスト形式の初期化パラメータ・ファイルに手動で追加する必要があります。

関連項目： 第 9 章「制御ファイルの管理」

データベース作成時の REDO ログ・ファイルの指定

CREATE DATABASE 文で LOGFILE 句は必須でなく、単純にこれを省略すると Oracle Managed Files の REDO ログ・ファイルが作成されます。LOGFILE 句を省略すると、デフォルトの REDO ログ・ファイルの保存先に REDO ログ・ファイルが作成されます。デフォルトの保存先は、次の優先度に従って定義されます。

- DB_CREATE_ONLINE_LOG_DEST_*n* が設定されている場合は、指定された各ディレクトリにログ・ファイルのメンバーが作成されます。最大数は MAXLOGMEMBERS 初期化パラメータの値です。
- DB_CREATE_ONLINE_LOG_DEST_*n* パラメータが設定されていない場合で、DB_CREATE_FILE_DEST および DB_RECOVERY_FILE_DEST 初期化パラメータの両方が設定されているときは、それぞれの場所に 1 つの Oracle Managed Files のログ・ファイルのメンバーが作成されます。DB_CREATE_FILE_DEST アーカイブ先のログ・ファイルが最初のメンバーです。
- DB_CREATE_FILE_DEST 初期化パラメータのみが指定されている場合は、その場所にログ・ファイルのメンバーが作成されます。
- DB_RECOVERY_FILE_DEST 初期化パラメータのみが指定されている場合は、その場所にログ・ファイルのメンバーが作成されます。

Oracle Managed Files の REDO ログ・ファイルのデフォルト・サイズは 100MB です。

必要に応じて、ファイル名を省略した LOGFILE 句を指定することにより、デフォルトの属性を変更した Oracle Managed Files の REDO ログ・ファイルを作成できます。REDO ログ・ファイルは同じ方法で作成されますが、次のような例外があります。CREATE DATABASE 文の LOGFILE 句にファイル名を指定せず、Oracle Managed Files の作成に必要な初期化パラメータが 1 つも設定されていない場合は、CREATE DATABASE 文が失敗します。

関連項目： 第 10 章「REDO ログの管理」

データベース作成時の SYSTEM 表領域および SYSAUX 表領域用データファイルの指定

CREATE DATABASE 文で DATAFILE 句または SYSAUX DATAFILE 句は必須でなく、単純にこれを省略すると SYSTEM 表領域および SYSAUX 表領域用の Oracle Managed Files のデータファイルが作成されます。DATAFILE 句を省略すると、次の処理のいずれかが実行されます。

- DB_CREATE_FILE_DEST が設定されている場合は、SYSTEM 表領域用および SYSAUX 表領域用の Oracle Managed Files のデータファイルが DB_CREATE_FILE_DEST ディレクトリに 1 つずつ作成されます。
- DB_CREATE_FILE_DEST が設定されていない場合は、オペレーティング・システム固有の名前とサイズで、SYSTEM 表領域のデータファイルおよび SYSAUX 表領域のデータファイルが 1 つずつ作成されます。この方法で作成された SYSTEM 表領域のデータファイルまたは SYSAUX 表領域のデータファイルは Oracle Managed Files ではありません。

デフォルトでは、SYSTEM および SYSAUX 表領域用を含めて Oracle Managed Files のデータファイルは 100MB で、自動拡張可能です。自動拡張が必要な場合、データファイルは既存サイズまたは 100MB 単位（いずれか小さい方）で拡張されます。データファイルの指定時（CREATE 操作または ALTER TABLESPACE 操作時）に、STORAGE 句の NEXT パラメータを使用して、自動拡張可能単位を明示的に指定することもできます。

必要に応じて、SYSTEM 表領域用または SYSAUX 表領域用の Oracle Managed Files のデータファイルを作成し、デフォルトの属性を上書きできます。そのためには、ファイル名を省略し、上書きする属性を指定して、DATAFILE 句を指定します。ファイル名を指定せずに、DB_CREATE_FILE_DEST パラメータを設定すると、SYSTEM 表領域用または SYSAUX 表領域用の Oracle Managed Files のデータファイルが DB_CREATE_FILE_DEST ディレクトリに作成され、指定した属性が上書きされます。ただし、ファイル名を指定せず、DB_CREATE_FILE_DEST パラメータを設定しないと、CREATE DATABASE 文が失敗します。

Oracle Managed Files のデフォルト属性を上書きするときに、SIZE 値を指定しても AUTOEXTEND 句を指定しない場合、データファイルは自動拡張可能になりません。

データベース作成時の UNDO 表領域データファイルの指定

UNDO TABLESPACE 句の DATAFILE 副次句は必須でなく、ファイル仕様に必ずしもファイル名を指定する必要はありません。ファイル名を指定せず、DB_CREATE_FILE_DEST が設定されている場合は、Oracle Managed Files のデータファイルが DB_CREATE_FILE_DEST ディレクトリに作成されます。DB_CREATE_FILE_DEST が設定されていない場合は、構文エラーで文が失敗します。

UNDO TABLESPACE 句自体は、CREATE DATABASE 文のオプションです。この句を指定せず、自動 UNDO 管理モードが使用可能な場合（デフォルト）は、次のルールに従って、SYS_UNDOTS という名前のデフォルトの UNDO 表領域が作成され、自動拡張可能な 20MB のデータファイルが割り当てられます。

- DB_CREATE_FILE_DEST が設定されている場合は、指定されたディレクトリに Oracle Managed Files のデータファイルが作成されます。
- DB_CREATE_FILE_DEST が設定されていない場合は、オペレーティング・システム固有のデフォルトの場所にデータファイルが作成されます。

関連項目： 第 14 章「UNDO の管理」

データベース作成時のデフォルト一時表領域用一時ファイルの指定

DEFAULT TEMPORARY TABLESPACE 句の TEMPFILE 副次句は必須でなく、ファイル仕様に必ずしもファイル名を指定する必要はありません。ファイル名を指定せず、DB_CREATE_FILE_DEST が設定されている場合は、Oracle Managed Files の一時ファイルが DB_CREATE_FILE_DEST ディレクトリに作成されます。DB_CREATE_FILE_DEST が設定されていない場合は、構文エラーで CREATE DATABASE 文が失敗します。

DEFAULT TEMPORARY TABLESPACE 句自体はオプションです。この句を指定していない場合、デフォルト一時表領域は作成されません。

Oracle Managed Files の一時ファイルのデフォルト・サイズは 100MB です。このファイルは自動的に拡張可能で、最大サイズに制限はありません。

Oracle Managed Files を使用した CREATE DATABASE 文の例

ここでは、Oracle Managed Files 機能を使用した CREATE DATABASE 文の例を示します。

CREATE DATABASE: 例 1　この例では、次の Oracle Managed Files を含むデータベースが作成されます。

- ディレクトリ /u01/app/oracle/oradata の SYSTEM 表領域用データファイル。サイズは無制限に自動拡張可能です。
- ディレクトリ /u01/app/oracle/oradata の SYSAUX 表領域用データファイル。サイズは無制限に自動拡張可能です。この表領域は、自動セグメント領域管理を使用してローカル管理されます。
- それぞれ 100MB のメンバーを 2 つ含む 2 つのオンライン・ログ・グループ。/u02/oradata と /u03/oradata に 1 つずつ作成されます。
- 自動 UNDO 管理モードが使用可能の場合（デフォルト）は、ディレクトリ /u01/app/oracle/oradata の UNDO 表領域用データファイル。サイズは 20MB で、無制限に自動拡張可能です。SYS_UNDOTS という名前の UNDO 表領域が作成されます。
- CONTROL_FILES 初期化パラメータが指定されていない場合は、2 つの制御ファイルが /u02/oradata と /u03/oradata に 1 つずつ作成されます。/u02/oradata の制御ファイルが主制御ファイルになります。

Oracle Managed Files に関連する初期化パラメータ・ファイルで、次のパラメータを設定します。

```
DB_CREATE_FILE_DEST = '/u01/app/oracle/oradata'
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
```

SQL プロンプトから次の文を発行します。

```
SQL> CREATE DATABASE sample;
```

CREATE DATABASE: 例 2 この例では、次の Oracle Managed Files を含むデータベースが作成されます。

- ディレクトリ /u01/app/oracle/oradata の SYSTEM 表領域用データファイル。サイズは無制限に自動拡張可能です。
- ディレクトリ /u01/app/oracle/oradata の SYSAUX 表領域用データファイル。サイズは無制限に自動拡張可能です。この表領域は、自動セグメント領域管理を使用してローカル管理されます。
- ディレクトリ /u01/app/oracle/oradata の 2 つの REDO ログ・ファイル（各 100MB）。これらのファイルは多重化されていません。
- ディレクトリ /u01/app/oracle/oradata の UNDO 表領域用データファイル。サイズは 20MB で、無制限に自動拡張可能です。SYS_UNDOTS という名前の UNDO 表領域が作成されます。
- /u01/app/oracle/oradata の 1 つの制御ファイル。

この例では、次のように想定されています。

- 初期化パラメータ・ファイルに DB_CREATE_ONLINE_LOG_DEST_*n* 初期化パラメータは 1 つも指定されていません。
- 初期化パラメータ・ファイルに CONTROL_FILES 初期化パラメータは指定されていません。
- 自動 UNDO 管理モードは使用可能になっています。

SQL プロンプトから次の文を発行します。

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/app/oracle/oradata';
SQL> CREATE DATABASE sample2;
```

このデータベース構成は、本番データベースにはお薦めしません。この例は、非常にローエンドのデータベースか、単純なテスト・データベースを簡単に作成する方法を示しています。このデータベースの耐障害性を高めるには、制御ファイルを少なくともう 1 つ作成し、REDO ログを多重化する必要があります。

CREATE DATABASE: 例 3 この例では、デフォルト一時表領域および UNDO 表領域用の Oracle Managed Files のファイル・サイズを指定しています。次の Oracle Managed Files を持つデータベースが作成されます。

- ディレクトリ /u01/app/oracle/oradata の SYSTEM 表領域用データファイル（400MB）。SIZE が指定されているため、このファイルは自動拡張可能ではありません。
- ディレクトリ /u01/app/oracle/oradata の SYSAUX 表領域用データファイル（200MB）。SIZE が指定されているため、このファイルは自動拡張可能ではありません。この表領域は、自動セグメント領域管理を使用してローカル管理されます。
- それぞれ 100MB のメンバーを 2 つ含む 2 つの REDO ログ・グループ。ディレクトリ /u02/oradata と /u03/oradata に 1 つずつ作成されます。
- デフォルト一時表領域 dflt_ts に対して、ディレクトリ /u01/app/oracle/oradata の一時ファイル（10MB）。SIZE が指定されているため、このファイルは自動拡張可能ではありません。

- UNDO 表領域 undo_ts に対して、ディレクトリ /u01/app/oracle/oradata のデータファイル（100MB）。SIZE が指定されているため、このファイルは自動拡張可能ではありません。
- CONTROL_FILES 初期化パラメータが指定されていない場合は、2 つの制御ファイルがディレクトリ /u02/oradata と /u03/oradata に 1 つずつ作成されます。/u02/oradata の制御ファイルが主制御ファイルになります。

初期化パラメータ・ファイルで、次のパラメータを設定します。

```
DB_CREATE_FILE_DEST = '/u01/app/oracle/oradata'
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
```

SQL プロンプトから次の文を発行します。

```
SQL> CREATE DATABASE sample3 DATAFILE SIZE 400M
2>   SYSAUX DATAFILE SIZE 200M
3>   DEFAULT TEMPORARY TABLESPACE dflt_ts TEMPFILE SIZE 10M
4>   UNDO TABLESPACE undo_ts DATAFILE SIZE 100M;
```

Oracle Managed Files を使用した表領域用データファイルの作成

ここでは、データファイルを作成する次の文について説明します。

- CREATE TABLESPACE
- CREATE UNDO TABLESPACE
- ALTER TABLESPACE ...ADD DATAFILE

表領域を作成するときは、永続表領域と UNDO 表領域のいずれの場合も、DATAFILE 句はオプションです。DATAFILE 句を指定する場合、ファイル名はオプションです。DATAFILE 句またはファイル名を省略すると、次のルールが適用されます。

- DB_CREATE_FILE_DEST 初期化パラメータが設定されている場合は、パラメータで指定された場所に Oracle Managed Files のデータファイルが作成されます。
- DB_CREATE_FILE_DEST 初期化パラメータが設定されていない場合は、データファイルを作成する文が失敗します。

ALTER TABLESPACE...ADD DATAFILE 文で表領域にデータファイルを追加する場合、ファイル名はオプションです。ファイル名を省略すると、前の段落で説明したのと同じルールが適用されます。

デフォルトでは、永続表領域用の Oracle Managed Files のデータファイルのサイズは 100MB です。このファイルは自動的に拡張可能で、最大サイズに制限はありません。ただし、DATAFILE 句で SIZE 値を指定して（AUTOEXTEND 句を指定せずに）これらのデフォルトを変更すると、データファイルは自動拡張可能になりません。

関連項目：

- 15-9 ページ「データベース作成時の SYSTEM 表領域および SYSAUX 表領域用データファイルの指定」
- 15-10 ページ「データベース作成時の UNDO 表領域データファイルの指定」
- 第 12 章「表領域の管理」

CREATE TABLESPACE: 例

ここでは、Oracle Managed Files を持つ表領域の作成例をいくつか示します。

> **関連項目：** CREATE TABLESPACE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

CREATE TABLESPACE: 例 1 次の例では、データファイルを作成するデフォルトの場所を /u01/oradata に設定してから、そのディレクトリ上のデータファイルを含む表領域 tbs_1 を作成しています。データファイルは 100MB で、無制限に自動拡張可能です。

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';
SQL> CREATE TABLESPACE tbs_1;
```

CREATE TABLESPACE: 例 2 この例では、ディレクトリ /u01/oradata 上のデータファイルを含む表領域 tbs_2 を作成しています。このデータファイルの初期サイズは 400MB で、SIZE 句が指定されているため自動拡張可能ではありません。

初期化パラメータ・ファイルで、次のパラメータを設定します。

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```

SQL プロンプトから次の文を発行します。

```
SQL> CREATE TABLESPACE tbs_2 DATAFILE SIZE 400M;
```

CREATE TABLESPACE: 例 3 この例では、ディレクトリ /u01/oradata 上のデータファイルを含む表領域 tbs_3 を作成しています。作成されるデータファイルは初期サイズが 100MB、最大サイズが 800MB で自動拡張可能です。

初期化パラメータ・ファイルで、次のパラメータを設定します。

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```

SQL プロンプトから次の文を発行します。

```
SQL> CREATE TABLESPACE tbs_3 DATAFILE AUTOEXTEND ON MAXSIZE 800M;
```

CREATE TABLESPACE: 例 4 次の例では、データファイルを作成するデフォルトの場所を /u01/oradata に設定してから、そのディレクトリ上の 2 つのデータファイルを含む表領域 tbs_4 を作成しています。どちらのデータファイルも初期サイズは 200MB で、SIZE 値が指定されているため自動拡張可能ではありません。

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';
SQL> CREATE TABLESPACE tbs_4 DATAFILE SIZE 200M SIZE 200M;
```

CREATE UNDO TABLESPACE: 例

次の例では、ディレクトリ /u01/oradata 上のデータファイルを含む UNDO 表領域 undotbs_1 を作成しています。UNDO 表領域用のデータファイルは 100MB で、無制限に自動拡張可能です。

初期化パラメータ・ファイルで、次のパラメータを設定します。

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```

SQL プロンプトから次の文を発行します。

```
SQL> CREATE UNDO TABLESPACE undotbs_1;
```

> **関連項目：** CREATE UNDO TABLESPACE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

ALTER TABLESPACE: 例

この例では、自動拡張可能な Oracle Managed Files のデータファイルを tbs_1 表領域に追加しています。デフォルトの初期サイズは 100MB で、最大サイズは 800MB です。

初期化パラメータ・ファイルで、次のパラメータを設定します。

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```

SQL プロンプトから次の文を入力します。

```
SQL> ALTER TABLESPACE tbs_1 ADD DATAFILE AUTOEXTEND ON MAXSIZE 800M;
```

> **関連項目：** ALTER TABLESPACE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

Oracle Managed Files を使用した一時表領域用一時ファイルの作成

ここでは、一時ファイルを作成する次の文について説明します。

- CREATE TEMPORARY TABLESPACE
- ALTER TABLESPACE ...ADD TEMPFILE

一時表領域を作成する場合、TEMPFILE 句はオプションです。TEMPFILE 句を指定する場合、ファイル名はオプションです。TEMPFILE 句またはファイル名を省略すると、次のルールが適用されます。

- DB_CREATE_FILE_DEST 初期化パラメータが設定されている場合は、パラメータで指定された場所に Oracle Managed Files の一時ファイルが作成されます。
- DB_CREATE_FILE_DEST 初期化パラメータが設定されていない場合は、一時ファイルを作成する文が失敗します。

ALTER TABLESPACE...ADD TEMPFILE 文で表領域に一時ファイルを追加する場合、ファイル名はオプションです。ファイル名を省略すると、前の段落で説明したのと同じルールが適用されます。

Oracle Managed Files のデフォルト属性を上書きするときに、SIZE 値を指定しても AUTOEXTEND 句を指定しない場合、データファイルは自動拡張可能になりません。

> **関連項目：** 15-10 ページ「データベース作成時のデフォルト一時表領域用一時ファイルの指定」

CREATE TEMPORARY TABLESPACE: 例

次の例では、データファイルを作成するデフォルトの場所を /u01/oradata に設定してから、そのディレクトリ上の一時ファイルを含む表領域 temptbs_1 を作成しています。一時ファイルは 100MB で、無制限に自動拡張可能です。

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';
SQL> CREATE TEMPORARY TABLESPACE temptbs_1;
```

> **関連項目：** CREATE TABLESPACE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

ALTER TABLESPACE... ADD TEMPFILE: 例

次の例では、データファイルを作成するデフォルトの場所を /u03/oradata に設定してから、デフォルトの場所にある一時ファイルを表領域 temptbs_1 に追加しています。一時ファイルの初期サイズは 100MB です。このファイルは自動的に拡張可能で、最大サイズに制限はありません。

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u03/oradata';
SQL> ALTER TABLESPACE TBS_1 ADD TEMPFILE;
```

> **関連項目：** ALTER TABLESPACE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

Oracle Managed Files を使用した制御ファイルの作成

CREATE CONTROLFILE 文を発行すると、CONTROL_FILES 初期化パラメータによって指定されたファイルで、制御ファイルが作成（REUSE を指定した場合は再利用）されます。CONTROL_FILES パラメータが設定されていない場合は、制御ファイルのデフォルトの保存先に制御ファイルが作成されます。デフォルトの保存先は、15-8 ページの「データベース作成時の制御ファイルの指定」に記載されている優先度によって決定します。

Oracle Database によって Oracle Managed Files の制御ファイルが作成された場合で、サーバー・パラメータ・ファイルが存在するときは、サーバー・パラメータ・ファイルに CONTROL_FILES 初期化パラメータが作成されます。サーバー・パラメータ・ファイルが存在しない場合は、CONTROL_FILES 初期化パラメータを手動で作成して、初期化パラメータ・ファイルに追加する必要があります。

データベースのデータファイルが Oracle Managed Files の場合は、文の DATAFILE 句に、そのファイルのデータベース生成ファイル名を指定する必要があります。

REDO ログ・ファイルが Oracle Managed Files の場合は、NORESETLOGS または RESETLOGS キーワードによって、LOGFILE 句に指定できるパラメータが決まります。

- NORESETLOGS キーワードを使用する場合は、Oracle Managed Files の REDO ログ・ファイル用に生成されるファイル名を LOGFILE 句に指定する必要があります。
- RESETLOGS キーワードを使用する場合は、REDO ログ・ファイル名を CREATE DATABASE 文の場合と同様に指定できます。15-9 ページの「データベース作成時の REDO ログ・ファイルの指定」を参照してください。

Oracle Managed Files を使用した CREATE CONTROLFILE 文の使用例については、次の項を参照してください。

関連項目：

- CREATE CONTROLFILE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。
- 15-8 ページ「データベース作成時の制御ファイルの指定」

NORESETLOGS キーワードを使用した CREATE CONTROLFILE: 例

次の CREATE CONTROLFILE 文は、Oracle Managed Files のデータファイルおよび REDO ログ・ファイルを含むデータベースで ALTER DATABASE BACKUP CONTROLFILE TO TRACE 文を発行したときに生成されます。

```
CREATE CONTROLFILE
     DATABASE sample
     LOGFILE
       GROUP 1 ('/u01/oradata/SAMPLE/onlinelog/o1_mf_1_o220rtt9_.log',
                '/u02/oradata/SAMPLE/onlinelog/o1_mf_1_v2o0b2i3_.log')
                 SIZE 100M,
       GROUP 2 ('/u01/oradata/SAMPLE/onlinelog/o1_mf_2_p22056iw_.log',
                '/u02/oradata/SAMPLE/onlinelog/o1_mf_2_p02rcyg3_.log')
                 SIZE 100M
     NORESETLOGS
     DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_xu34ybm2_.dbf'
              SIZE 100M,
              '/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_aawbmz51_.dbf'
              SIZE 100M,
              '/u01/oradata/SAMPLE/datafile/o1_mf_sys_undo_apqbmz51_.dbf'
              SIZE 100M
     MAXLOGFILES 5
     MAXLOGHISTORY 100
     MAXDATAFILES 10
     MAXINSTANCES 2
     ARCHIVELOG;
```

RESETLOGS キーワードを使用した CREATE CONTROLFILE: 例

次の文は、RESETLOGS オプションを指定した CREATE CONTROLFILE 文の例です。DB_CREATE_FILE_DEST、DB_RECOVERY_FILE_DEST および DB_CREATE_ONLINE_LOG_DEST_*n* を組み合せて設定する必要があります。

```
CREATE CONTROLFILE
     DATABASE sample
     RESETLOGS
     DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_aawbmz51_.dbf',
              '/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_axybmz51_.dbf',
              '/u01/oradata/SAMPLE/datafile/o1_mf_sys_undo_azzbmz51_.dbf'
     SIZE 100M
     MAXLOGFILES 5
     MAXLOGHISTORY 100
     MAXDATAFILES 10
     MAXINSTANCES 2
     ARCHIVELOG;
```

後で、ALTER DATABASE OPEN RESETLOGS 文を発行して、REDO ログ・ファイルを再作成する必要があります。この操作については、15-17 ページの「ALTER DATABASE OPEN RESETLOGS 文の使用」を参照してください。使用していたログ・ファイルが Oracle Managed Files である場合、そのファイルは削除されません。

Oracle Managed Files を使用した REDO ログ・ファイルの作成

REDO ログ・ファイルはデータベース作成時に作成されます。また、次の文のどちらかを発行したときにも作成できます。

- ALTER DATABASE ADD LOGFILE
- ALTER DATABASE OPEN RESETLOGS

> **関連項目：** ALTER DATABASE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

ALTER DATABASE ADD LOGFILE 文の使用

ALTER DATABASE ADD LOGFILE 文を使用すると、現行の REDO ログ・ファイルに後から新しいグループを追加できます。Oracle Managed Files を使用している場合、ADD LOGFILE 句のファイル名はオプションです。ファイル名を省略した場合は、ログ・ファイルのデフォルトの保存先に REDO ログ・ファイルが作成されます。デフォルトの保存先は、15-9 ページの「データベース作成時の REDO ログ・ファイルの指定」に記載されている優先度によって決まります。

ファイル名を指定せず、Oracle Managed Files の作成に必要な初期化パラメータが 1 つも指定されていない場合は、文はエラーを戻します。

Oracle Managed Files のログ・ファイルのデフォルト・サイズは 100MB です。

完全ファイル名を指定すると、REDO ログ・ファイルのメンバーを引き続き追加および削除できます。

> **関連項目：**
>
> - 15-9 ページ「データベース作成時の REDO ログ・ファイルの指定」
> - 15-15 ページ「Oracle Managed Files を使用した制御ファイルの作成」

新しい REDO ログ・ファイルの追加：例　次の例では、一方のメンバーがディレクトリ /u01/oradata、もう一方のメンバーが /u02/oradata に存在するログ・グループを作成します。各ログ・ファイルのサイズは 100MB です。

初期化パラメータ・ファイルで、次のパラメータを設定します。

```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u01/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u02/oradata'
```

SQL プロンプトから次の文を発行します。

```
SQL> ALTER DATABASE ADD LOGFILE;
```

ALTER DATABASE OPEN RESETLOGS 文の使用

前に RESETLOGS を指定して制御ファイルを作成しており、その際、ファイル名を指定しなかった場合、または存在しないファイル名を指定した場合は、ALTER DATABASE OPEN RESETLOGS 文を発行したときに、REDO ログ・ファイルが作成されます。制御ファイル内に何も指定されていない場合に、REDO ログ・ファイルの格納ディレクトリを決めるルールは、15-9 ページの「データベース作成時の REDO ログ・ファイルの指定」に記載されているルールと同じです。

Oracle Managed Files を使用したアーカイブ・ログの作成

アーカイブ・ログは、次の場合に DB_RECOVERY_FILE_DEST の場所に作成されます。

- ARC バックグラウンド・プロセスまたは LGWR バックグラウンド・プロセスがオンライン REDO ログをアーカイブする場合。または、
- ALTER SYSTEM ARHIVE LOG CURRENT 文が発行された場合。

たとえば、次のパラメータ設定が初期化パラメータ・ファイルに含まれているとします。

```
DB_RECOVERY_FILE_DEST_SIZE = 20G
DB_RECOVERY_FILE_DEST      = '/u01/oradata'
LOG_ARCHIVE_DEST_1         = 'LOCATION=USE_DB_RECOVERY_FILE_DEST'
```

Oracle Managed Files の動作

ファイル名を使用して既存ファイルを識別する SQL 文では、Oracle Managed Files のファイル名が受け入れられます。これらのファイル名は他のファイル名と同様に、制御ファイルに格納されています。また、バックアップとリカバリ用に Recovery Manager を使用している場合は、Recovery Manager カタログに格納されています。これらのファイル名を表示するには、データファイルと一時ファイルの監視に使用できる通常の固定パフォーマンス・ビューまたは動的パフォーマンス・ビュー（たとえば、V$DATAFILE や DBA_DATA_FILES など）のいずれかを使用します。

次に、データベース生成ファイル名を使用した文の例を示します。

```
SQL> ALTER DATABASE
  2> RENAME FILE '/u01/oradata/mydb/datafile/o1_mf_tbs01_ziw3bopb_.dbf'
  3> TO '/u01/oradata/mydb/tbs0101.dbf';

SQL> ALTER DATABASE
  2> DROP LOGFILE '/u01/oradata/mydb/onlinelog/o1_mf_1_wo94n2xi_.log';

SQL> ALTER TABLE emp
  2> ALLOCATE EXTENT
  3> (DATAFILE '/u01/oradata/mydb/datafile/o1_mf_tbs1_2ixfh90q_.dbf');
```

Oracle Managed Files のデータファイル、一時ファイルおよび制御ファイルは、Oracle Managed Files 以外の対応するファイルと同様にバックアップおよびリストアを実行できます。データベース生成ファイル名を使用しても、エクスポート・ファイルなどの論理バックアッ

プ・ファイルの使用には影響しません。これは特に、表領域の Point-in-Time リカバリ（TSPITR）およびトランスポータブル表領域のエクスポート・ファイルにとって重要です。

Oracle Managed Files の動作が Oracle Managed Files 以外のファイルと異なる場合があります。次の項では、それらの場合について説明します。

データファイルおよび一時ファイルの削除

データベース管理ではないファイルとは異なり、Oracle Managed Files のデータファイルまたは一時ファイルを削除すると、制御ファイルからファイル名が削除されて、ファイル・システムからファイルが自動的に削除されます。Oracle Managed Files を削除する文は、次のとおりです。

- `DROP TABLESPACE`
- `ALTER DATABASE TEMPFILE ...DROP`

これらの文を使用して、いつでも Oracle Managed Files あるいはそれ以外のファイルを削除することもできます。

- `ALTER TABLESPACE ...DROP DATAFILE`
- `ALTER TABLESPACE ...DROP TEMPFILE`

REDO ログ・ファイルの削除

Oracle Managed Files の REDO ログ・ファイルを削除すると、その Oracle Managed Files が削除されます。削除するグループまたはメンバーを指定します。次の文は、REDO ログ・ファイルを削除します。

- `ALTER DATABASE DROP LOGFILE`
- `ALTER DATABASE DROP LOGFILE MEMBER`

ファイルの名前変更

ファイルの名前を変更するには、次の文が使用されます。

- `ALTER DATABASE RENAME FILE`
- `ALTER TABLESPACE ...RENAME DATAFILE`

これらの文は、実際にはオペレーティング・システム上のファイルの名前を変更しませんが、そのかわりに制御ファイル内の名前が変更されます。変更前のファイルが Oracle Managed Files で、そのファイルが存在している場合は削除されます。この文を発行するときは、オペレーティング・システムのファイル名の規則を使用して各ファイルを指定する必要があります。

スタンバイ・データベースの管理

スタンバイ・データベースのデータファイル、制御ファイルおよび REDO ログ・ファイルは、データベースで管理できます。プライマリ・データベースで Oracle Managed Files が使用されているかどうかは関係ありません。

スタンバイ・データベースのリカバリでデータファイルを作成する REDO を検出したとき、そのデータファイルが Oracle Managed Files の場合は、リカバリ・プロセスによって、ローカル・ファイル・システムのデフォルトの場所に空のファイルが作成されます。これにより、管理者が操作することなく、新しいファイルの REDO が即時に適用されます。

スタンバイ・データベースのリカバリで表領域を削除する REDO を検出した場合は、ローカル・ファイル・システム内にある Oracle Managed Files のデータファイルがすべて削除されます。プライマリ・データベースで `INCLUDING DATAFILES` オプションを発行したかどうかは関係ありません。

Oracle Managed Files の使用例

ここでは、使用例を示して、Oracle Managed Files の使用方法をさらに詳しく説明します。

使用例 1: 多重 REDO ログを含むデータベースの作成および管理

この使用例では、DBA が、データファイルと REDO ログ・ファイルが異なるディレクトリに存在するデータベースを作成します。REDO ログ・ファイルと制御ファイルは多重化されています。データベースは UNDO 表領域を使用し、デフォルト一時表領域を持っています。このデータベースの作成とメンテナンスに関するタスクは、次のとおりです。

1. 初期化パラメータの設定

   DBA は、データベースを作成する前に、初期化パラメータ・ファイルに 3 つの汎用的なファイル作成デフォルトを設定します。自動 UNDO 管理モード（デフォルト）も使用可能にします。

   ```
   DB_CREATE_FILE_DEST = '/u01/oradata'
   DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
   DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
   UNDO_MANAGEMENT = AUTO
   ```

   DB_CREATE_FILE_DEST パラメータは、データファイルと一時ファイルのデフォルトのファイル・システム・ディレクトリを設定します。

   DB_CREATE_ONLINE_LOG_DEST_1 および DB_CREATE_ONLINE_LOG_DEST_2 パラメータは、REDO ログ・ファイルと制御ファイルを作成するためのデフォルトのファイル・システム・ディレクトリを設定します。REDO ログ・ファイルと制御ファイルは、2 つのディレクトリの間で多重化されます。

2. データベースの作成

   初期化パラメータの設定が完了すると、次の文でデータベースを作成できます。

   ```
   SQL> CREATE DATABASE sample
   2>   DEFAULT TEMPORARY TABLESPACE dflttmp;
   ```

   DATAFILE 句が指定されておらず、DB_CREATE_FILE_DEST 初期化パラメータが設定されているため、SYSTEM 表領域のデータファイルはデフォルトのファイル・システム（この使用例では /u01/oradata）に作成されます。ファイル名は、データベースによって一意に生成されます。データファイルは初期サイズが 100MB で、無制限に自動拡張可能です。このファイルは Oracle Managed Files です。SYSAUX 表領域についても同様のデータファイルが作成されます。

   LOGFILE 句が指定されていないため、2 つの REDO ログ・グループが作成されます。各グループにはそれぞれ 2 つのメンバーがあり、一方のメンバーは DB_CREATE_ONLINE_LOG_DEST_1、もう一方のメンバーは DB_CREATE_ONLINE_LOG_DEST_2 に作成されます。ファイル名は、データベースによって一意に生成されます。ログ・ファイルのサイズは 100MB です。ログ・ファイルのメンバーは Oracle Managed Files です。

   同様に、CONTROL_FILES 初期化パラメータが設定されておらず、2 つの DB_CREATE_ONLINE_LOG_DEST_*n* 初期化パラメータが設定されているので、2 つの制御ファイルが作成されます。DB_CREATE_ONLINE_LOG_DEST_1 に配置された制御ファイルが主制御ファイルになります。DB_CREATE_ONLINE_LOG_DEST_2 に配置された制御ファイルは多重コピーです。ファイル名は、データベースによって一意に生成されます。これらのファイルは Oracle Managed Files です。サーバー・パラメータ・ファイルが存在する場合は、CONTROL_FILES 初期化パラメータが生成されます。

   自動 UNDO 管理モードが設定されていますが、UNDO 表領域が指定されておらず、DB_CREATE_FILE_DEST 初期化パラメータが設定されているため、DB_CREATE_FILE_DEST で指定されたディレクトリに、UNDOTBS という名前のデフォルト UNDO 表領域が作成されます。データファイルは 20MB で、自動拡張可能です。このファイルは Oracle Managed Files です。

最後に、dflttmp という名前のデフォルト一時表領域が指定されています。パラメータ・ファイルに DB_CREATE_FILE_DEST が設定されているため、このパラメータで指定されたディレクトリに dflttmp の一時ファイルが作成されます。一時ファイルは 100MB で、無制限に自動拡張可能です。このファイルは Oracle Managed Files です。

作成されたファイルを、生成ファイル名によるファイル・ツリーで表現すると次のようになります。

```
/u01
     /oradata
          /SAMPLE
               /datafile
                    /o1_mf_system_cmr7t30p_.dbf
                    /o1_mf_sysaux_cmr7t88p_.dbf
                    /o1_mf_sys_undo_2ixfh90q_.dbf
                    /o1_mf_dflttmp_157se6ff_.tmp
/u02
     /oradata
          /SAMPLE
               /onlinelog
                    /o1_mf_1_0orrm31z_.log
                    /o1_mf_2_2xyz16am_.log
               /controlfile
                    /o1_mf_cmr7t30p_.ctl
/u03
     /oradata
          /SAMPLE
               /onlinelog
                    /o1_mf_1_ixfvm8w9_.log
                    /o1_mf_2_q89tmp28_.log
               /controlfile
                    /o1_mf_x1sr8t36_.ctl
```

内部的に生成されたファイル名は、通常のビューを選択して表示できます。次に例を示します。

```
SQL> SELECT NAME FROM V$DATAFILE;

NAME
--------------------------------------------------
/u01/oradata/SAMPLE/datafile/o1_mf_system_cmr7t30p_.dbf
/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_cmr7t88p_.dbf
/u01/oradata/SAMPLE/datafile/o1_mf_sys_undo_2ixfh90q_.dbf

3 rows selected
```

3. 制御ファイルの管理

データベースの作成時に制御ファイルが作成され、パラメータ・ファイルに CONTROL_FILES 初期化パラメータが追加されました。必要に応じて、DBA は CREATE CONTROLFILE 文を使用して、データベース用の制御ファイルを再作成したり、新しい制御ファイルを作成できます。

DATAFILE 句および LOGFILE 句には、正しい Oracle Managed Files のファイル名を指定する必要があります。ALTER DATABASE BACKUP CONTROLFILE TO TRACE 文は、正しいファイル名を含むスクリプトを生成します。また、ファイル名は、V$DATAFILE、V$TEMPFILE および V$LOGFILE ビューを選択して確認することもできます。次の例では、サンプル・データベースの制御ファイルを再作成しています。

```
CREATE CONTROLFILE REUSE
  DATABASE sample
  LOGFILE
    GROUP 1('/u02/oradata/SAMPLE/onlinelog/o1_mf_1_0orrm31z_.log',
            '/u03/oradata/SAMPLE/onlinelog/o1_mf_1_ixfvm8w9_.log'),
    GROUP 2('/u02/oradata/SAMPLE/onlinelog/o1_mf_2_2xyz16am_.log',
            '/u03/oradata/SAMPLE/onlinelog/o1_mf_2_q89tmp28_.log')
```

```
NORESETLOGS
DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_cmr7t30p_.dbf',
         '/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_cmr7t88p_.dbf',
         '/u01/oradata/SAMPLE/datafile/o1_mf_sys_undo_2ixfh90q_.dbf',
         '/u01/oradata/SAMPLE/datafile/o1_mf_dflttmp_157se6ff_.tmp'
MAXLOGFILES 5
MAXLOGHISTORY 100
MAXDATAFILES 10
MAXINSTANCES 2
ARCHIVELOG;
```

この文で作成される制御ファイルは、データベースを作成したときに生成された CONTROL_FILES 初期化パラメータの指定どおりに配置されます。REUSE 句が指定されているので、既存のファイルがすべて上書きされます。

4. REDO ログの管理

   REDO ログ・ファイルの新しいグループを作成するには、DBA が ALTER DATABASE ADD LOGFILE 文を使用します。次の文は、DB_CREATE_ONLINE_LOG_DEST_1 と DB_CREATE_ONLINE_LOG_DEST_2 にメンバーを持つログ・ファイルを追加します。これらのファイルは Oracle Managed Files です。

   ```
   SQL> ALTER DATABASE ADD LOGFILE;
   ```

   オンライン REDO ログ・ファイルのメンバーは、完全なファイル名を指定することにより、追加および削除できます。

   GROUP 句を使用して、ログ・グループを削除できます。次の例では、Oracle Managed Files のログ・ファイルの各メンバーに対応するオペレーティング・システム・ファイルが自動的に削除されます。

   ```
   SQL> ALTER DATABASE DROP LOGFILE GROUP 3;
   ```

5. 表領域の管理

   sample データベースで今後表領域を作成する際、すべてのデータファイルがデフォルトで配置される記憶域は、DB_CREATE_FILE_DEST 初期化パラメータで指定された場所（この使用例では /u01/oradata）です。ファイル名を指定せずにデータファイルを作成すると、そのファイルは初期化パラメータ DB_CREATE_FILE_DEST で指定されたファイル・システムに配置されます。次に例を示します。

   ```
   SQL> CREATE TABLESPACE tbs_1;
   ```

   この文は、/u01/oradata を記憶域とする表領域を作成します。作成されるデータファイルは初期サイズが 100MB で、無制限に自動拡張可能です。このデータファイルは Oracle Managed Files です。

   表領域を削除すると、その表領域に対応する Oracle Managed Files も自動的に削除されます。次の文は、表領域とその格納に使用されているすべての Oracle Managed Files を削除します。

   ```
   SQL> DROP TABLESPACE tbs_1;
   ```

   最初のデータファイルがいっぱいになっても、新しいデータファイルは自動的に作成されません。別の Oracle Managed Files のデータファイルを追加することによって、表領域を拡張できます。次の文は、DB_CREATE_FILE_DEST で指定された場所に別のデータファイルを追加します。

   ```
   SQL> ALTER TABLESPACE tbs_1 ADD DATAFILE;
   ```

   デフォルトのファイル・システムは、初期化パラメータを変更することによって変更できます。これを行っても、既存のデータベースは変更されません。今後の作成にのみ影響を与えます。次の文を使用すると、初期化パラメータを動的に変更できます。

   ```
   SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST='/u04/oradata';
   ```

6. REDO 情報のアーカイブ

   REDO ログ・ファイルのアーカイブは、Oracle Managed Files と Oracle Managed Files 以外のファイルの間で違いはありません。アーカイブするログ・ファイルのファイル・システム上のアーカイブ先は、LOG_ARCHIVE_DEST_*n* 初期化パラメータで指定できます。ファイル名は、LOG_ARCHIVE_FORMAT パラメータまたはそのデフォルトに基づいて生成されます。アーカイブ・ログは Oracle Managed Files ではありません。

7. バックアップ、リストアおよびリカバリ

   Oracle Managed Files は標準オペレーティング・システム・ファイルと互換性があるため、オペレーティング・システム・ユーティリティを使用してバックアップまたはリストアを実行できます。データベースのバックアップ、リストアおよびリカバリを実行する既存の方法はすべて、Oracle Managed Files に対しても機能します。

使用例 2: データベース領域とフラッシュ・リカバリ領域を含むデータベースの作成と管理

この使用例では、DBA が、制御ファイルおよび REDO ログ・ファイルを多重化するデータベースを作成します。アーカイブ・ログと Recovery Manager によるバックアップは、フラッシュ・リカバリ領域に作成されます。このデータベースの作成とメンテナンスに関するタスクは、次のとおりです。

1. 初期化パラメータの設定

   DBA は、次の汎用的なファイル作成デフォルトを設定します。

   ```
   DB_CREATE_FILE_DEST = '/u01/oradata'
   DB_RECOVERY_FILE_DEST_SIZE = 10G
   DB_RECOVERY_FILE_DEST = '/u02/oradata'
   LOG_ARCHIVE_DEST_1 = 'LOCATION = USE_DB_RECOVERY_FILE_DEST'
   ```

   DB_CREATE_FILE_DEST パラメータは、データファイル、一時ファイル、制御ファイルおよび REDO ログのデフォルトのファイル・システム・ディレクトリを設定します。

   DB_RECOVERY_FILE_DEST パラメータは、制御ファイル、REDO ログおよび Recovery Manager によるバックアップのデフォルトのファイル・システム・ディレクトリを設定します。

   LOG_ARCHIVE_DEST_1 構成の 'LOCATION=USE_DB_RECOVERY_FILE_DEST' は、アーカイブ・ログを DB_RECOVERY_FILE_DEST の場所にリダイレクトします。

   DB_CREATE_FILE_DEST パラメータと DB_RECOVERY_FILE_DEST パラメータは、ログ・ファイルおよび制御ファイル作成用のデフォルトのディレクトリを設定します。REDO ログ・ファイルと制御ファイルは、2 つのディレクトリの間で多重化されます。

2. データベースの作成

3. 制御ファイルの管理

4. REDO ログの管理

5. 表領域の管理

   タスク 2、3、4 と 5 は使用例 1 と同じです。ただし、制御ファイルと REDO ログは、DB_CREATE_FILE_DEST と DB_RECOVERY_FILE_DEST の場所の間で多重化されます。

6. REDO ログ情報のアーカイブ

   オンライン・ログのアーカイブは、Oracle Managed Files と Oracle Managed Files 以外のファイルの間には違いはありません。DB_RECOVERY_FILE_DEST で作成されるアーカイブ・ログは、Oracle Managed Files です。

7. バックアップ、リストアおよびリカバリ

   Oracle Managed Files は標準オペレーティング・システム・ファイルと互換性があるため、オペレーティング・システム・ユーティリティを使用してバックアップまたはリストアを実行できます。データベースのバックアップ、リストアおよびリカバリを実行する既存の

方法はすべて、Oracle Managed Files に対しても機能します。フォーマット・オプションが指定されていない場合、Recovery Manager によるすべてのディスクのバックアップは、DB_RECOVERY_FILE_DEST の場所に作成されます。バックアップは Oracle Managed Files です。

使用例 3: 既存のデータベースへの Oracle Managed Files の追加

この例では、Oracle Managed Files が含まれていない既存のデータベースに対して、DBA が Oracle Managed Files を含む新しい表領域を作成し、/u03/oradata ディレクトリにその表領域を配置しようとしていると想定しています。

1. 初期化パラメータの設定

   データファイルの自動作成を可能にするために、DB_CREATE_FILE_DEST 初期化パラメータを、データファイルを作成するファイル・システム・ディレクトリに設定します。これは、次のように動的に実行できます。

   ```
   SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u03/oradata';
   ```

2. 表領域の作成

   DB_CREATE_FILE_DEST の設定が完了すると、CREATE TABLESPACE 文から DATAFILE 句を省略できます。データファイルは、DB_CREATE_FILE_DEST で指定された場所にデフォルトで作成されます。次に例を示します。

   ```
   SQL> CREATE TABLESPACE tbs_2;
   ```

   tbs_2 表領域を削除すると、データファイルが自動的に削除されます。

第 III 部

スキーマ・オブジェクト

第 III 部では、Oracle Database 内のスキーマ・オブジェクトの作成方法と管理方法について説明します。この部の構成は、次のとおりです。

16

スキーマ・オブジェクトの管理

この章の内容は次のとおりです。

一度の操作で複数の表やビューを作成する方法

CREATE SCHEMA 文を使用すると、一度の操作で複数の表やビューを作成し、権限を付与できます。CREATE SCHEMA 文は、複数の表とビューの作成、および権限の付与を一度の操作で確実に行う必要がある場合に便利です。個々の表やビューの作成が失敗したり、権限の付与が失敗したりすると、文全体がロールバックされます。オブジェクトは作成されず、権限も付与されません。

CREATE SCHEMA 文に指定できるのは、CREATE TABLE、CREATE VIEW および GRANT 文のみです。指定した文を発行するための権限を持っている必要があります。この文を実行しても実際にスキーマが作成されるわけではありません。スキーマが作成されるのは、CREATE USER 文でユーザーを作成したときです。そのかわりに、この文はスキーマを移入します。

次の文は、2 つの表とそれらのデータを結合するビューを作成します。

```
CREATE SCHEMA AUTHORIZATION scott
    CREATE TABLE dept (
        deptno NUMBER(3,0) PRIMARY KEY,
        dname VARCHAR2(15),
        loc VARCHAR2(25))
    CREATE TABLE emp (
        empno NUMBER(5,0) PRIMARY KEY,
        ename VARCHAR2(15) NOT NULL,
        job VARCHAR2(10),
        mgr NUMBER(5,0),
        hiredate DATE DEFAULT (sysdate),
        sal NUMBER(7,2),
        comm NUMBER(7,2),
        deptno NUMBER(3,0) NOT NULL
        CONSTRAINT dept_fkey REFERENCES dept)
   CREATE VIEW sales_staff AS
        SELECT empno, ename, sal, comm
        FROM emp
        WHERE deptno = 30
        WITH CHECK OPTION CONSTRAINT sales_staff_cnst
        GRANT SELECT ON sales_staff TO human_resources;
```

CREATE SCHEMA 文は、STORAGE 句など、ANSI の CREATE TABLE 文と CREATE VIEW 文を拡張した Oracle Database 独自の機能をサポートしていません。

> **関連項目：** CREATE SCHEMA 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

表、索引およびクラスタの分析

次の目的でスキーマ・オブジェクト（表、索引またはクラスタ）を分析します。

- 統計の収集と管理
- 記憶形式の妥当性の検証
- 表またはクラスタの移行行と連鎖行の識別

注意： オプティマイザ統計の収集には、ANALYZE の COMPUTE 句および ESTIMATE 句を使用しないでください。これらの句は、下位互換性を維持するためにサポートされています。そのかわりに、DBMS_STATS パッケージを使用します。このパッケージを使用すると、統計をパラレルに収集したり、パーティション・オブジェクトのグローバル統計を収集でき、統計収集を他の方法で細かくチューニングできます。統計に依存するコストベース・オプティマイザでは、最終的には DBMS_STATS で収集された統計のみが使用されます。DBMS_STATS パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

コストベース・オプティマイザに関連しない統計収集には、ANALYZE 文（DBMS_STATS ではなく）を使用する必要があります。たとえば、次のような場合です。

- VALIDATE または LIST CHAINED ROWS 句を使用する場合
- 空きリスト・ブロックの情報を収集する場合

この項の内容は、次のとおりです。

DBMS_STATS を使用した表および索引統計の収集

DBMS_STATS パッケージまたは ANALYZE 文を使用して、表、索引またはクラスタの物理記憶特性の統計を収集できます。これらの統計はデータ・ディクショナリに格納され、オプティマイザで使用して、分析対象オブジェクトにアクセスする SQL 文に最も効率的な実行計画を選択できます。

オプティマイザ統計の収集には、より多様性のある DBMS_STATS パッケージを使用することをお薦めしますが、空きブロックや平均容量など、オプティマイザに関連付けられていない統計の収集には ANALYZE 文を使用する必要があります。

DBMS_STATS パッケージを使用すると、パラレル実行を利用した統計収集と統計の外部操作ができます。統計をデータ・ディクショナリ以外の表に格納し、オプティマイザに影響を与えずにその統計を操作できます。統計をデータベース間でコピーしたり、バックアップ・コピーを作成できます。

次の DBMS_STATS プロシージャにより、オプティマイザ統計を収集できます。

- GATHER_INDEX_STATS
- GATHER_TABLE_STATS
- GATHER_SCHEMA_STATS
- GATHER_DATABASE_STATS

関連項目：

- DBMS_STATS を使用してオプティマイザ統計を収集する方法は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- DBMS_STATS パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

表、索引、クラスタおよびマテリアライズド・ビューの妥当性チェック

表、索引、クラスタまたはマテリアライズド・ビューの構造の整合性を検証するには、VALIDATE STRUCTURE オプションを指定した ANALYZE 文を使用します。構造が有効な場合、エラーは返されません。しかし、構造が破損していると、エラー・メッセージが出力されます。

たとえば、ハードウェアやその他のシステムに障害が発生した場合、索引が破損し、正しく機能しなくなる可能性があります。索引の妥当性をチェックすると、索引内のすべてのエントリが、対応付けられた表の正しい行を示しているかを確認できます。索引が破損した場合は、その索引を削除して再作成できます。

表、索引またはクラスタが破損している場合は、削除して再作成する必要があります。マテリアライズド・ビューが破損している場合は、完全リフレッシュを実行し、問題が修正されたことを確認します。問題が修正されない場合は、マテリアライズド・ビューを削除して再作成します。

次の文は、emp 表を分析します。

```
ANALYZE TABLE emp VALIDATE STRUCTURE;
```

CASCADE オプションを含めると、オブジェクトとすべての依存オブジェクト（索引など）の妥当性をチェックできます。次の文は、emp 表と、それに対応付けられているすべての索引の妥当性をチェックします。

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE;
```

デフォルトでは、CASCADE オプションによって、完全な妥当性チェックが実行されます。この操作はリソースを消費する可能性があるため、FAST 句を使用してより高速なバージョンの妥当性チェックを実行できます。このバージョンでは、最適化されたチェック・アルゴリズムを使用して破損の有無をチェックしますが、破損の詳細はレポートしません。FAST チェックで破損が検出された場合は、FAST 句を指定せずに CASCADE オプションを使用すると、破損箇所を特定できます。次の文では、emp 表と、それに対応付けられているすべての索引の妥当性チェックを高速に実行します。

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE FAST;
```

妥当性をチェックするオブジェクトに対して DML を実行している間でも、オンラインで構造の妥当性をチェックするように指定できます。オブジェクトに影響を与える DML 文と並行して妥当性チェックを実行すると、パフォーマンスがわずかに低下しますが、これはオンラインで ANALYZE 文を実行できる柔軟性によって相殺されます。次の文は、emp 表と、それに対応付けられているすべての索引の妥当性をオンラインでチェックします。

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE ONLINE;
```

関連項目： ANALYZE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

表とクラスタの連鎖行のリスト

表またはクラスタの連鎖行と移行行は、LIST CHAINED ROWS 句を指定した ANALYZE 文を使用して検出できます。この文の結果は、LIST CHAINED ROWS 句によって返される情報を受け入れるために明示的に作成した指定の表に格納されます。この結果は、行を更新するための領域が十分であるかどうかを判断する上で役立ちます。

CHAINED_ROWS 表の作成

ANALYZE...LIST CHAINED ROWS 文によって返されるデータを格納する表を作成するには、UTLCHAIN.SQL または UTLCHN1.SQL スクリプトを実行します。これらのスクリプトは、データベースに付属しています。これらのスクリプトは、スクリプトを実行するユーザーのスキーマ内に CHAINED_ROWS という名前の表を作成します。

注意： CHAINED_ROWS 表を作成するためにどちらのスクリプトを実行するかは、データベースの互換性レベルと分析する表のタイプによって決まります。詳細は、『Oracle Database SQL リファレンス』を参照してください。

CHAINED_ROWS 表を作成した後、ANALYZE 文の INTO 句にその表を指定します。たとえば、次の文は、CHAINED_ROWS 表に、emp_dept クラスタ内の連鎖行に関する情報を含む行を挿入します。

```
ANALYZE CLUSTER emp_dept LIST CHAINED ROWS INTO CHAINED_ROWS;
```

関連項目：

- CHAINED_ROWS 表の詳細は、『Oracle Database リファレンス』を参照してください。
- 過剰な行連鎖のある表についてのセグメント・アドバイザのレポート方法の詳細は、17-13 ページの「セグメント・アドバイザの使用」を参照してください。

表内の移行行または連鎖行の解消

CHAINED_ROWS 表の情報を使用すると、既存表内にある移行行と連鎖行を低減または解消できます。これには、次の手順を使用します。

1. ANALYZE 文を使用して、移行行と連鎖行に関する情報を収集します。

   ```
   ANALYZE TABLE order_hist LIST CHAINED ROWS;
   ```

2. 出力表を問い合せます。

   ```
   SELECT *
   FROM CHAINED_ROWS
   WHERE TABLE_NAME = 'ORDER_HIST';

   OWNER_NAME  TABLE_NAME  CLUST... HEAD_ROWID          TIMESTAMP
   ----------  ----------  -----... ------------------  ---------
   SCOTT       ORDER_HIST       ... AAAAluAAHAAAAA1AAA  04-MAR-96
   SCOTT       ORDER_HIST       ... AAAAluAAHAAAAA1AAB  04-MAR-96
   SCOTT       ORDER_HIST       ... AAAAluAAHAAAAA1AAC  04-MAR-96
   ```

   移行行または連鎖行がすべてリストされます。

3. 出力表の問合せによって、移行行または連鎖行が多数存在することがわかれば、以降の手順を実行して移行行を解消します。

4. 既存表と同じ列を持つ中間表を作成して、移行行と連鎖行を格納します。

```
CREATE TABLE int_order_hist
   AS SELECT *
      FROM order_hist
      WHERE ROWID IN
         (SELECT HEAD_ROWID
            FROM CHAINED_ROWS
            WHERE TABLE_NAME = 'ORDER_HIST');
```

5. 既存表から移行行と連鎖行を削除します。

```
DELETE FROM order_hist
   WHERE ROWID IN
      (SELECT HEAD_ROWID
         FROM CHAINED_ROWS
         WHERE TABLE_NAME = 'ORDER_HIST');
```

6. 中間表の行を既存表に挿入します。

```
INSERT INTO order_hist
   SELECT *
   FROM int_order_hist;
```

7. 中間表を削除します。

```
DROP TABLE int_order_history;
```

8. 手順 1 で収集した情報を出力表から削除します。

```
DELETE FROM CHAINED_ROWS
   WHERE TABLE_NAME = 'ORDER_HIST';
```

9. 再度 ANALYZE 文を使用してから、出力表を問い合せます。

出力表に表示された行は連鎖しています。連鎖行を解消するには、データ・ブロックのサイズを大きくする以外にありません。すべての状況において連鎖を回避することはほぼ不可能です。LONG 列や大きい CHAR 列または VARCHAR2 列を持つ表では、ほとんどの場合、連鎖の発生は避けられません。

表とクラスタの切捨て

表（またはクラスタ）は残したままで、内容が完全に空になるように、表のすべての行またはクラスタ化表のグループ内のすべての行を削除できます。たとえば、月ごとのデータが含まれている表では、各月の終わりにそのデータをアーカイブした後で、表を空にする（すべての行を削除する）必要があります。

表からすべての行を削除するには、次の 3 通りの方法があります。

- DELETE 文を使用する。
- DROP 文と CREATE 文を使用する。
- TRUNCATE 文を使用する。

次の項では、これらの方法について説明します。

DELETE の使用

DELETE 文を使用して表の行を削除できます。たとえば、次の文は emp 表からすべての行を削除します。

```
DELETE FROM emp;
```

DELETE 文を使用するときに、表またはクラスタに多数の行が存在していると、それらの行を削除する際に相当のシステム・リソースが使用されます。たとえば、CPU 時間、その表と対応付けられた索引の REDO ログ領域、UNDO セグメント領域などのリソースが必要です。また、各行が削除されるときに、トリガーが起動される場合があります。結果的に空になる表またはクラスタに事前に割り当てられた領域は、行を削除してもそのオブジェクトに対応付けられたままです。DELETE を使用すると削除する行を選択できますが、TRUNCATE と DROP の場合はオブジェクト全体が削除されます。

> **関連項目：** DELETE 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

DROP と CREATE の使用

表を削除してから再作成します。たとえば、次の例では、emp 表を削除してから再作成しています。

```
DROP TABLE emp;
CREATE TABLE emp ( ... );
```

表やクラスタを削除してから再作成すると、対応付けられた索引、整合性制約およびトリガーもすべて削除され、削除された表またはクラスタ化表に依存するオブジェクトはすべて無効になります。また、削除された表またはクラスタ化表に対する権限付与もすべて削除されます。

TRUNCATE の使用

TRUNCATE 文を使用して表のすべての行を削除できます。たとえば、次の文は emp 表を切り捨てます。

```
TRUNCATE TABLE emp;
```

TRUNCATE 文は、表またはクラスタからすべての行を削除するための高速で効率的な方法を提供します。TRUNCATE 文はロールバック情報を生成せず、即時にコミットします。この文はデータ定義言語（DDL）であり、ロールバックできません。TRUNCATE 文を実行しても、切り捨てられる表に対応付けられている構造（制約およびトリガー）または認可は影響を受けません。また、TRUNCATE 文では、表を切り捨てた後で、表に現在割り当てられている領域を、その表を含む表領域に戻すかどうかも指定できます。

自分のスキーマにある表またはクラスタは切り捨てることができます。DROP ANY TABLE システム権限を持っているユーザーは、どのスキーマ内の表またはクラスタでも切り捨てることができます。

親キーを含む表またはクラスタ化表を切り捨てる際は、別の表で定義されているすべての参照外部キーを事前に使用禁止にする必要があります。自己参照制約を使用禁止にする必要はありません。

TRUNCATE 文によって表から行を削除する場合、表に対応付けられているトリガーは起動されません。また、TRUNCATE 文は、監査が使用可能の場合でも、DELETE 文に対応するどのような監査情報も生成しません。そのかわりに、発行された TRUNCATE 文に対して、単一の監査レコードが生成されます。監査の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

ハッシュ・クラスタや、ハッシュ・クラスタまたは索引クラスタ内の表を個別に切り捨てることはできません。索引クラスタを切り捨てると、そのクラスタ内のすべての表からすべての行が削除されます。個々のクラスタ化表からすべての行を削除する必要がある場合は、DELETE 文を使用するか、または表を削除してから再作成してください。

TRUNCATE 文の REUSE STORAGE または DROP STORAGE オプションは、切り捨てた後に、表またはクラスタに現在割り当てられている領域を、その表を含む表領域に戻すかどうかを制御します。デフォルトのオプション DROP STORAGE は、文実行後の表に割り当てられたエクステントの数を MINEXTENTS の元の設定まで減らします。解放されたエクステントはシステムに戻され、他のオブジェクトによって使用できます。

一方、REUSE STORAGE オプションを指定すると、表またはクラスタに対して現在割り当てられているすべての領域は割り当てられたままになります。たとえば、次の文は emp_dept クラスタを切り捨てて、クラスタに対してそれまでに割り当てられているすべてのエクステントを、今後の挿入と削除のためにそのまま残します。

```
TRUNCATE CLUSTER emp_dept REUSE STORAGE;
```

REUSE または DROP STORAGE オプションは、対応付けられたすべての索引に適用されます。表またはクラスタを切り捨てると、対応付けられた索引もすべて切り捨てられます。切り捨てられた表、クラスタまたは対応付けられた索引の記憶域パラメータは、切捨て後も変わりません。

> **関連項目：** TRUNCATE TABLE および TRUNCATE CLUSTER 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

トリガーの使用可能および使用禁止

データベース・トリガーとは、データベースに格納されており、表に行を追加するなどの特定の条件が発生したときにアクティブ化（起動）されるプロシージャです。トリガーを使用してデータベースの標準機能を補完することにより、データベース管理システムを高度にカスタマイズできます。たとえば、表に対する DML 操作を制限するトリガーを作成して、通常の営業時間中に発行された文のみ許可できます。

データベース・トリガーは、表、スキーマまたはデータベースに対応付けることができます。データベース・トリガーは、次の場合に暗黙的に起動されます。

- 対応付けられている表に対して DML 文（INSERT、UPDATE、DELETE）が実行されたとき
- データベースまたはスキーマ内のオブジェクトに対して、特定の DDL 文（ALTER、CREATE、DROP など）が実行されたとき
- 指定したデータベース・イベントが発生したとき（STARTUP、SHUTDOWN、SERVERERROR など）

このリストがすべてではありません。トリガーを起動する文とデータベース・イベントの詳細は、『Oracle Database SQL リファレンス』を参照してください。

トリガーを作成するには、CREATE TRIGGER 文を使用します。トリガーは、トリガー・イベントの前（BEFORE）、後（AFTER）またはトリガー・イベントのかわりに（INSTEAD OF）起動するように定義できます。次の文は、表 scott.emp に対してトリガー scott.emp_permit_changes を作成します。このトリガーは、指定されたいずれかの文が実行される前に起動します。

```
CREATE TRIGGER scott.emp_permit_changes
     BEFORE
     DELETE OR INSERT OR UPDATE
     ON scott.emp
     .
     .
     .
pl/sql block
     .
     .
     .
```

後で DROP TRIGGER 文を発行し、トリガーをデータベースから削除できます。

トリガーには、次の 2 つのモードがあります。

- 使用可能

  トリガーが起動される文を発行したときに、トリガー制限（存在する場合）が TRUE と評価された場合は、使用可能トリガーによってトリガー本体が実行されます。デフォルトでは、トリガーを最初に作成したときに使用可能に設定されます。

- 使用禁止

  トリガーが起動される文を発行したときに、トリガー制限（存在する場合）が TRUE と評価された場合でも、使用禁止トリガーはトリガー本体を実行しません。

ALTER TABLE 文を使用してトリガーを使用可能または使用禁止にするには、表を所有しているか、表に対する ALTER オブジェクト権限があるか、または ALTER ANY TABLE システム権限があることが必要です。また、ALTER TRIGGER 文を使用してトリガーを個別に使用可能または使用禁止にするには、トリガーを所有しているか、または ALTER ANY TRIGGER システム権限を持っている必要があります。

関連項目：

- トリガーの詳細は、『Oracle Database 概要』を参照してください。
- CREATE TRIGGER 文の構文は、『Oracle Database SQL リファレンス』を参照してください。
- トリガーの作成と使用の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

トリガーを使用可能にする方法

使用禁止のトリガーを使用可能にするには、ENABLE オプションを指定した ALTER TRIGGER 文を使用します。たとえば、inventory 表に定義されている reorder という使用禁止のトリガーを使用可能にするには、次の文を入力します。

```
ALTER TRIGGER reorder ENABLE;
```

ENABLE ALL TRIGGERS オプションを指定した ALTER TABLE 文を使用すれば、特定の表に定義されているトリガーをすべて使用可能にできます。たとえば、inventory 表に定義されているトリガーをすべて使用可能にするには、次の文を入力します。

```
ALTER TABLE inventory
    ENABLE ALL TRIGGERS;
```

関連項目： ALTER TRIGGER 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

トリガーを使用禁止にする方法

次の条件のいずれか 1 つが成り立つ場合は、一時的にトリガーを使用禁止にすることを検討してください。

- トリガーの参照するオブジェクトが使用可能でない場合
- 大規模なデータ・ロードを実行する際に、トリガーを起動せずに迅速にデータをロードする場合
- トリガーが適用される表にデータをロードする場合

トリガーを使用禁止にするには、DISABLE オプションを指定した ALTER TRIGGER 文を使用します。たとえば、inventory 表に定義されているトリガー reorder を使用禁止にするには、次の文を入力します。

```
ALTER TRIGGER reorder DISABLE;
```

DISABLE ALL TRIGGERS オプションを指定した ALTER TABLE 文を使用すれば、表に関連するトリガーをすべて同時に使用禁止にできます。たとえば、inventory 表に定義されているトリガーをすべて使用禁止にするには、次の文を入力します。

```
ALTER TABLE inventory
    DISABLE ALL TRIGGERS;
```

整合性制約の管理

整合性制約とは、表の 1 つ以上の列に格納される値を制限するルールです。CREATE TABLE 文または ALTER TABLE 文に制約句を指定することにより、その制約の影響を受ける列と、制約の条件を識別できます。

ここでは、制約の概念と、整合性制約の定義および管理に使用する SQL 文について説明します。この項の内容は、次のとおりです。

- 整合性制約の状態
- 定義時の整合性制約の設定
- 既存の整合性制約の変更、名前の変更または削除
- 制約チェックの遅延
- 制約例外のレポート
- 制約情報の表示

関連項目：

- 整合性制約の詳細は、『Oracle Database 概要』を参照してください。
- アプリケーションで整合性制約を使用する際の詳細と使用例については、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

整合性制約の状態

制約は、使用可能（ENABLE）と使用禁止（DISABLE）のいずれの状態にするかを指定できます。制約が使用可能になっている場合は、データベース内でデータが入力または更新されるときにチェックが行われ、制約に従っていないデータは入力されません。制約が使用禁止になっている場合は、ルールに従っていないデータでもデータベースに入力できます。

また、表の既存データが必ず制約に従うように指定できます（VALIDATE）。逆に NOVALIDATE を指定すると、既存データが制約に従っていることは保証されません。

表に定義されている整合性制約は、次のいずれかの状態にあります。

- ENABLE、VALIDATE
- ENABLE、NOVALIDATE
- DISABLE、VALIDATE
- DISABLE、NOVALIDATE

これらの状態の意味と組合せの結果の詳細は、『Oracle Database SQL リファレンス』を参照してください。ここでは、これらの結果のいくつかについて説明します。

制約を使用禁止にする方法

整合性制約によって定義したルールを施行するには、その制約を常に使用可能にしておく必要があります。しかし、次のような場合は、パフォーマンス上の理由から、表の整合性制約を一時的に使用禁止にすることを検討してください。

- 表に大量のデータをロードする場合
- 表に大規模な変更を加えるバッチ操作を実行する場合（たとえば、既存の番号に 1000 を加えてすべての従業員番号を変更する場合）
- 1 つの表を一度にインポートまたはエクスポートする場合

これら 3 つの場合には、整合性制約を一時的に使用禁止にすることにより、操作のパフォーマンスを改善できます。これは、特にデータ・ウェアハウス構成に当てはまります。

制約が使用禁止である間は、その制約に違反するデータを入力できます。したがって、前述の操作を終了した後に、制約を必ず使用可能にする必要があります。

制約を使用可能にする方法

制約が使用可能になっている場合、制約に違反する行は表に挿入されません。しかし、制約が使用禁止の場合は、制約に違反する行でも表に挿入できます。このような行を制約の例外と呼びます。制約が妥当性チェックなしで使用可能な状態にある場合、制約が使用禁止になっていた間に入力された違反データはそのまま残っています。制約を妥当性チェック済みの状態にするためには、制約に違反する行を更新または削除する必要があります。

制約を使用可能にするときに、特定の整合性制約に対する例外を指定できます。16-15 ページの「制約例外のレポート」を参照してください。制約に違反している行はすべて EXCEPTIONS 表に格納され、検証できます。

妥当性チェックなしで使用可能な制約の状態

制約が妥当性チェックなしで使用可能な状態にある場合、それ以後の文はすべて、制約に従っているかどうかがチェックされます。ただし、表の既存データはチェックされません。妥当性チェックなしで使用可能な状態の制約を持つ表には、無効なデータが含まれる可能性がありますが、無効なデータを新たに追加することはできません。妥当性チェックなしで使用可能な制約は、有効なオンライン・トランザクション処理（OLTP）データをアップロードしているデータ・ウェアハウス構成で役立ちます。

制約を使用可能にする場合に、妥当性チェックは必ずしも必要ではありません。妥当性チェックなしで制約を使用可能にする方が、妥当性チェックありで制約を使用可能にするよりはるかに高速です。また、すでに使用可能になっている制約の妥当性をチェックする場合、妥当性チェック中の DML ロックは必要ありません（すでに使用禁止にした制約の妥当性をチェックする場合とは異なります）。これは、制約の規定により、妥当性チェック中に違反データが挿入

されないことが保証されているためです。したがって、妥当性チェックなしで使用可能にすれば、制約を使用可能にすることによって一般に生じる停止時間を短縮できます。

整合性制約の効率的な使用 : 手順

整合性制約の状態を次の順序で使用したときに、最も大きな利点が得られます。

1. 使用禁止状態
2. 操作（ロード、エクスポート、インポート）の実行
3. 妥当性チェックなしで使用可能な状態
4. 使用可能状態

制約をこの順序で使用する際の利点は、次のとおりです。

- ロックが保持されません。
- すべての制約を同時に使用可能状態にすることができます。
- 制約を使用可能にする処理がパラレルで行われます。
- 表での同時アクティビティを実行できます。

定義時の整合性制約の設定

CREATE TABLE 文または ALTER TABLE 文で整合性制約を定義するときに ENABLE/DISABLE 句を指定して、その制約を使用可能 / 使用禁止、妥当性チェックあり / 妥当性チェックなしの状態にすることができます。制約の定義時に ENABLE/DISABLE 句を指定しなければ、自動的にその制約は妥当性チェックありで使用可能な状態になります。

定義時に制約を使用禁止にする方法

次の CREATE TABLE 文と ALTER TABLE 文は、整合性制約を定義して、使用禁止にします。

```
CREATE TABLE emp (
    empno NUMBER(5) PRIMARY KEY DISABLE,   . . . ;

ALTER TABLE emp
   ADD PRIMARY KEY (empno) DISABLE;
```

整合性制約を定義して使用禁止にする ALTER TABLE 文は、表の行がその整合性制約に違反しているために失敗することはありません。制約のルールが施行されていないので、制約の定義が許可されます。

定義時に制約を使用可能にする方法

次の CREATE TABLE 文と ALTER TABLE 文は、整合性制約を定義して、使用可能にします。

```
CREATE TABLE emp (
    empno NUMBER(5) CONSTRAINT emp.pk PRIMARY KEY,   . . . ;

ALTER TABLE emp
    ADD CONSTRAINT emp.pk PRIMARY KEY (empno);
```

整合性制約を定義して使用可能にする ALTER TABLE 文は、表の行が整合性制約に違反しているために失敗する場合があります。この場合、その文はロールバックされ、制約定義は格納されず、使用可能にもなりません。

UNIQUE または PRIMARY KEY 制約を使用可能にすると、対応する索引が作成されます。

注意 : 並列性を利用できるように制約を使用可能にする効率的な手順は、16-12 ページの「整合性制約の効率的な使用 : 手順」を参照してください。

関連項目 : 19-8 ページ「制約に対応付けられた索引の作成」

既存の整合性制約の変更、名前の変更または削除

ALTER TABLE 文では、制約を使用可能または使用禁止にする他、制約を変更または削除することもできます。制約を規定するために UNIQUE または PRIMARY KEY 索引が使用されている場合、その索引に対応する制約を削除または使用禁止にすると、明示的に指定しないかぎり、索引は削除されます。

使用可能な外部キーが PRIMARY キーまたは UNIQUE キーを参照している場合、PRIMARY キーまたは UNIQUE キーの制約またはその索引を削除したり使用禁止にしたりすることはできません。

使用可能状態の制約を使用禁止にする方法

次の文は、使用可能状態の整合性制約を使用禁止にします。2 番目の文では、対応する索引を保持するように指定しています。

```
ALTER TABLE dept
    DISABLE CONSTRAINT dname_ukey;

ALTER TABLE dept
    DISABLE PRIMARY KEY KEEP INDEX,
    DISABLE UNIQUE (dname, loc) KEEP INDEX;
```

次の文は、使用禁止状態の整合性制約を妥当性チェックなしで使用可能な状態にします。

```
ALTER TABLE dept
    ENABLE NOVALIDATE CONSTRAINT dname_ukey;

ALTER TABLE dept
    ENABLE NOVALIDATE PRIMARY KEY,
    ENABLE NOVALIDATE UNIQUE (dname, loc);
```

次の文は、使用禁止状態の整合性制約を使用可能にするか、または妥当性チェックありの状態にします。

```
ALTER TABLE dept
    MODIFY CONSTRAINT dname_key VALIDATE;

ALTER TABLE dept
    MODIFY PRIMARY KEY ENABLE NOVALIDATE;
```

次の文は、使用禁止状態の整合性制約を使用可能にします。

```
ALTER TABLE dept
    ENABLE CONSTRAINT dname_ukey;

ALTER TABLE dept
    ENABLE PRIMARY KEY,
    ENABLE UNIQUE (dname, loc);
```

UNIQUE キーまたは PRIMARY KEY 制約、およびすべての依存する FOREIGN KEY 制約を一度に使用禁止または削除するには、DISABLE 句または DROP 句の CASCADE オプションを使用します。たとえば、次の文は PRIMARY KEY 制約とこれに依存する FOREIGN KEY 制約を使用禁止にします。

```
ALTER TABLE dept
    DISABLE PRIMARY KEY CASCADE;
```

制約名の変更

ALTER TABLE...RENAME CONSTRAINT 文を使用すると、表に対する既存の制約の名前を変更できます。新しい制約名には、ユーザーの既存の制約名と競合しない名前を指定する必要があります。

次の文は、表 dept に対する dname_ukey 制約の名前を変更します。

```
ALTER TABLE dept
    RENAME CONSTRAINT dname_ukey TO dname_unikey;
```

制約名を変更しても、実表に対するすべての依存性は引き続き有効です。

RENAME CONSTRAINT 句を使用すると、制約のシステム生成名を変更できます。

制約の削除

整合性制約は、規定するルールが成立しなくなった場合、またはその制約が不要になった場合に削除できます。制約を削除するには、ALTER TABLE 文で次のいずれかの句を指定します。

- DROP PRIMARY KEY
- DROP UNIQUE
- DROP CONSTRAINT

次の 2 つの文は、整合性制約を削除します。2 番目の文は、PRIMARY KEY 制約に対応する索引を保持します。

```
ALTER TABLE dept
    DROP UNIQUE (dname, loc);

ALTER TABLE emp
    DROP PRIMARY KEY KEEP INDEX,
    DROP CONSTRAINT dept_fkey;
```

FOREIGN KEY が UNIQUE または PRIMARY KEY を参照している場合は、DROP 文に CASCADE CONSTRAINTS 句を指定しないかぎり、制約を削除できません。

制約チェックの遅延

データベースが制約をチェックしたときに制約が満たされていない場合は、エラーが通知されます。制約の妥当性チェックは、トランザクションが終わるまで遅延できます。

SET CONSTRAINTS 文を発行すると、トランザクションの実行中、または別の SET CONSTRAINTS 文によってモードが再設定されるまで、SET CONSTRAINTS モードが継続します。

注意 :

- SET CONSTRAINT 文は、トリガーの内部では発行できません。
- 遅延可能な一意キーと主キーは、必ず非一意索引を使用する必要があります。

すべての制約を遅延に設定する方法

データ操作に使用するアプリケーションでは、実際にデータの処理を始める前にすべての制約を遅延に設定する必要があります。遅延可能制約をすべて遅延に設定するには、次の DML 文を使用します。

```
SET CONSTRAINTS ALL DEFERRED;
```

> **注意：** SET CONSTRAINTS 文は、現行のトランザクションにのみ適用されます。制約を作成したときに指定したデフォルトは、その制約が存在するかぎり保持されています。ALTER SESSION SET CONSTRAINTS 文は、現行のセッションにしか適用されません。

コミットのチェック（オプション）

COMMIT の発行直前に SET CONSTRAINTS ALL IMMEDIATE 文を発行することにより、制約違反をチェックできます。制約になんらかの問題があると、この文は失敗し、エラーの原因となっている制約が識別されます。制約違反のままコミットすると、トランザクションはロールバックされ、エラー・メッセージが返されます。

制約例外のレポート

制約の妥当性チェック時に例外が存在すると、エラーが返され、整合性制約は妥当性チェックなしの状態のままになります。整合性制約の例外が存在しているために文が正常に実行されない場合、文はロールバックされます。例外が存在している場合は、制約の例外をすべて更新または削除するまで、制約の妥当性はチェックできません。

整合性制約に違反している行を判断するには、ENABLE 句に EXCEPTIONS オプションを指定して ALTER TABLE 文を発行します。EXCEPTIONS オプションにより、例外を含むすべての行の行 ID、表所有者、表名および制約名が指定した表に格納されます。

制約を使用可能にする前に、ENABLE 句の EXCEPTIONS オプションからの情報を格納する適切な例外レポート表を作成する必要があります。例外表を作成するには、UTLEXCPT.SQL スクリプトまたは UTLEXPT1.SQL スクリプトを実行します。

> **注意：** EXCEPTIONS 表を作成するためにどちらのスクリプトを実行するかは、データベースの互換性レベルと分析する表のタイプによって決まります。詳細は、『Oracle Database SQL リファレンス』を参照してください。

これらのスクリプトのどちらを使用しても、EXCEPTIONS という名前の表が作成されます。また、スクリプトを変更して再実行すると、新たに別の名前の例外表を作成できます。

次の文は、dept 表の PRIMARY KEY を検証します。例外が存在すると、EXCEPTIONS 表に情報が挿入されます。

```
ALTER TABLE dept ENABLE PRIMARY KEY EXCEPTIONS INTO EXCEPTIONS;
```

dept 表に重複する主キー値が存在し、dept の PRIMARY KEY 制約の名前が sys_c00610 である場合は、次の問合せによって例外が表示されます。

```
SELECT * FROM EXCEPTIONS;
```

次の例外が表示されます。

```
fROWID               OWNER      TABLE_NAME      CONSTRAINT
------------------  ---------  --------------  -----------
AAAAZ9AABAAABvqAAB  SCOTT      DEPT            SYS_C00610
AAAAZ9AABAAABvqAAG  SCOTT      DEPT            SYS_C00610
```

次の文および結果のように、例外レポート表およびマスター表の行を結合した詳細な問合せの実行により、特定の制約に違反している実際の行を表示できます。

```
SELECT deptno, dname, loc FROM dept, EXCEPTIONS
    WHERE EXCEPTIONS.constraint = 'SYS_C00610'
    AND dept.rowid = EXCEPTIONS.row_id;

DEPTNO     DNAME             LOC
---------- --------------    -----------
10         ACCOUNTING        NEW YORK
10         RESEARCH          DALLAS
```

制約に違反している行はすべて更新するか、または制約を含む表から削除する必要があります。例外を更新する場合は、制約に違反する値を、制約を満たす値または NULL に変更します。マスター表の行を更新または削除した後、以後取得する例外レポートとの混同を避けるために、例外レポート表の例外に対応する行は削除します。マスター表と例外レポート表を更新する文は、トランザクションの一貫性を保証するために、同じトランザクション内で実行してください。

前述の例の例外を訂正するために、次のトランザクションを発行できます。

```
UPDATE dept SET deptno = 20 WHERE dname = 'RESEARCH';
DELETE FROM EXCEPTIONS WHERE constraint = 'SYS_C00610';
COMMIT;
```

例外管理の最終的な目的は、例外レポート表の例外をすべて取り除くことにあります。

注意： 制約が使用禁止になっている表の現在の例外を訂正している間に、他のユーザーが新しい例外を作成する文を発行する可能性があります。これを避けるには、例外を取り除く前に、制約に ENABLE NOVALIDATE のマークを付けます。

関連項目： EXCEPTIONS 表の詳細は、『Oracle Database リファレンス』を参照してください。

制約情報の表示

表の制約定義を表示し、制約で指定されている列を識別できるように、次のビューが用意されています。

ビュー	説明
DBA_CONSTRAINTS ALL_CONSTRAINTS USER_CONSTRAINTS	DBA ビューには、データベース内のすべての制約定義が表示されます。ALL ビューには、現行ユーザーがアクセス可能な制約定義が表示されます。USER ビューには、現行ユーザーが所有している制約定義が表示されます。
DBA_CONS_COLUMNS ALL_CONS_COLUMNS USER_CONS_COLUMNS	DBA ビューには、制約で指定されているデータベース内のすべての列が表示されます。ALL ビューには、制約で指定されていて、現行ユーザーがアクセス可能な列のみが表示されます。USER ビューには、制約で指定されていて、現行ユーザーが所有している列のみが表示されます。

関連項目： これらのビューの列の詳細は、『Oracle Database リファレンス』を参照してください。

スキーマ・オブジェクトの名前変更

オブジェクト名を変更するには、そのオブジェクトが自分のスキーマ内に存在する必要があります。スキーマ・オブジェクトは、次のいずれかの方法で名前を変更できます。

- オブジェクトを削除して再作成する。
- RENAME 文を使用してオブジェクトの名前を変更する。
- ALTER ... RENAME 文を使用してオブジェクトの名前を変更する（索引およびトリガーの場合）。

オブジェクトを削除して再作成する場合、そのオブジェクトに付与された権限はすべて失われます。オブジェクトを再作成するときに、再度権限を付与してください。

RENAME 文を使用して、表、ビュー、順序またはそれらのプライベート・シノニムの名前を変更することもできます。RENAME 文を使用すると、そのオブジェクトの整合性制約、索引および権限付与は新しい名前に引き継がれます。たとえば、次の文は sales_staff ビューの名前を変更します。

```
RENAME sales_staff TO dept_30;
```

注意： ストアド PL/SQL プログラム・ユニット、パブリック・シノニムまたはクラスタに対しては、RENAME を使用できません。これらのオブジェクトの名前を変更するには、削除してから再作成してください。

スキーマ・オブジェクト名を変更する前に、次のような影響について検討する必要があります。

- 名前を変更されたオブジェクトに依存しているビューと PL/SQL プログラム・ユニットはすべて無効になるため、次に使用する前に再コンパイルする必要があります。
- 名前を変更されたオブジェクトのシノニムを使用すると、必ずエラーが返されます。

関連項目： RENAME 文の構文は、『Oracle Database SQL リファレンス』を参照してください。

オブジェクト依存性の管理

ここでは、オブジェクト依存性とオブジェクトの無効化に関するバックグラウンド情報を提供し、無効なオブジェクトを再検証する方法について説明します。この項の内容は、次のとおりです。

- オブジェクト依存性とオブジェクトの無効化の概要
- DDL を使用した手動による無効なオブジェクトの再コンパイル
- PL/SQL パッケージのプロシージャを使用した手動による無効なオブジェクトの再コンパイル

オブジェクト依存性とオブジェクトの無効化の概要

スキーマ・オブジェクトには、他のオブジェクトを参照するタイプのものがあります。たとえば、ビューには表または他のビューを参照する問合せが含まれ、PL/SQL サブプログラムは他のサブプログラムを起動し、静的 SQL を使用して表やビューを参照します。他のオブジェクトを参照するオブジェクトは**依存オブジェクト**と呼ばれ、参照されるオブジェクトは**参照オブジェクト**と呼ばれます。これらの参照はコンパイル時に確立され、コンパイラが参照を解決できない場合は、コンパイル対象の依存オブジェクトに無効のマークが付けられます。

Oracle Database には、依存オブジェクトが参照オブジェクトに関して常に最新であることを確認する自動メカニズムが用意されています。依存オブジェクトが作成されると、データベースによって、依存オブジェクトとその参照オブジェクト間の依存性が追跡されます。参照オブジェクトが依存オブジェクトに影響を与えるような方法で変更されると、依存オブジェクトには無効のマークが付けられます。無効になった依存オブジェクトは、参照オブジェクトの新し

い定義で再コンパイルして使用できるようにする必要があります。再コンパイルは、無効な依存オブジェクトが参照されると自動的に実行されます。

スキーマ・オブジェクトを無効にする可能性のある変更に注意することは重要です。無効化により、データベース上で実行されているアプリケーションが影響を受けるためです。ここでは、オブジェクトがどのように無効化されるか、および無効になったオブジェクトをどのように識別し検証するかについて説明します。

オブジェクトの無効化

アプリケーションの通常の実行では、ビューやストアド・プロシージャが無効になることはありません。アプリケーションでは普通、実行中に表の構造を変更したり、ビューやストアド・プロシージャの定義を変更することはないためです。表やビュー、PL/SQL ユニットが変更されるのは、通常、パッチ・スクリプトや非定型の DDL 文を使用して、アプリケーションにパッチを適用したり、アプリケーションをアップグレードする場合です。一連の参照オブジェクトを変更するパッチを適用した後は、依存オブジェクトが無効のままになっている可能性があります。

データベース内の無効な一連のオブジェクトを表示するには、次の問合せを使用します。

```
SELECT object_name, object_type FROM dba_objects
WHERE status = 'INVALID';
```

スキーマ・オブジェクトが無効になると、Enterprise Manager のデータベース・ホームページにアラートが表示されます。

オブジェクトの無効化によって、アプリケーションは次の 2 つの影響を受けます。第 1 に、無効なオブジェクトは、再検証されるまでアプリケーションで使用できません。再検証によって、アプリケーション実行の待機時間が長くなります。無効なオブジェクトが多数ある場合は、初回実行時の待機時間が長時間になる可能性があります。第 2 に、プロシージャ、ファンクションまたはパッケージの無効化によって、そのプロシージャ、ファンクションまたはパッケージを同時に実行している他のセッションで例外が発生する可能性があります。アプリケーションを別のセッションで使用しているときにパッチを適用すると、アプリケーションを実行しているセッションによって、使用中のオブジェクトが無効化されたことが通知され、ORA-4061、ORA-4064、ORA-4065 または ORA-4068 の 4 つの例外のうちのいずれか 1 つが発生します。これらの例外は、パッチ適用後にアプリケーション・セッションを再起動して修正する必要があります。

適切な SQL 文に COMPILE 句を指定して、スキーマ・オブジェクトを強制的に再コンパイルできます。詳細は、16-19 ページの「DDL を使用した手動による無効なオブジェクトの再コンパイル」を参照してください。

無効なオブジェクトが多数存在することが判明している場合は、UTL_RECOMP PL/SQL パッケージを使用して一括再コンパイルを実行します。詳細は、16-19 ページの「PL/SQL パッケージのプロシージャを使用した手動による無効なオブジェクトの再コンパイル」を参照してください。

次に、スキーマ・オブジェクトの無効化に関する一般的な規則をいくつか示します。

- 参照オブジェクトとその依存オブジェクトの間で、データベースは、依存関係に含まれる参照オブジェクトの要素を追跡します。たとえば、単一表のビューで表内の列のサブセットのみが選択された場合、それらの列のみが依存関係に含まれます。オブジェクトの各依存関係について、依存関係に含まれる要素の定義が変更されると（要素の削除も含む）、依存オブジェクトは無効になります。逆に、依存関係に含まれない要素の定義のみが変更された場合、依存オブジェクトは有効なままです。

  したがって、開発者がスキーマ・オブジェクトの変更時に注意することにより、多くの場合、依存オブジェクトの無効化とそれによるデータベースへの不要な追加作業の発生を回避することができます。

- 依存オブジェクトは連鎖的に無効になります。オブジェクトがなんらかの理由で無効になると、そのオブジェクトのすべての依存オブジェクトがただちに無効になります。

- スキーマ・オブジェクトに対するオブジェクト権限を取り消すと、依存オブジェクトは連鎖的に無効になります。

関連項目： スキーマ・オブジェクトの依存性の詳細は、『Oracle Database 概要』を参照してください。

DDL を使用した手動による無効なオブジェクトの再コンパイル

単一のスキーマ・オブジェクトを手動で再コンパイルするには、ALTER 文を使用します。たとえば、パッケージ本体の Pkg1 を再コンパイルするには、次の DDL 文を実行します。

```
ALTER PACKAGE pkg1 COMPILE REUSE SETTINGS;
```

関連項目： 様々な ALTER 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

PL/SQL パッケージのプロシージャを使用した手動による無効なオブジェクトの再コンパイル

アプリケーションのアップグレードまたはパッチ適用後に無効なオブジェクトを再検証しておくと、オブジェクトの必要時に再検証が行われることによるアプリケーションの待機時間の発生を回避できます。Oracle には、オブジェクトの再検証を支援する UTL_RECOMP パッケージが用意されています。RECOMP_SERIAL プロシージャは、特定のスキーマの無効なオブジェクトすべてを再コンパイルします。スキーマ名の引数が指定されていない場合は、データベース内の無効なオブジェクトすべてを再コンパイルします。RECOMP_PARALLEL プロシージャも同様に機能しますが、複数の CPU を利用してパラレルに処理する点が異なります。

例

次の PL/SQL ブロックを実行して、データベース内の無効なオブジェクトすべてをパラレルに、依存順序に従って再検証します。

```
begin
   utl_recomp.recomp_parallel();
end;
```

DBMS_UTILITY パッケージを使用して、無効なオブジェクトを個別に再検証することもできます。次のスクリプトは、HR スキーマの UPDATE_SALARY プロシージャを再検証する PL/SQL ブロックです。

```
begin
   dbms_utility.validate('HR', 'UPDATE_SALARY', namespace=>1);
end;
```

次のスクリプトは、パッケージ本体の HR.ACCT_MGMT を再検証する PL/SQL ブロックです。

```
begin
   dbms_utility.validate('HR', 'ACCT_MGMT', namespace=>2);
end;
```

関連項目： UTL_RECOMP パッケージおよび DBMS_UTILITY パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

オブジェクトの名前解決の管理

SQL 文で参照されるオブジェクト名は、ピリオドで区切られた複数の断片から構成できます。ここでは、データベースでオブジェクト名を解決する方法を説明します。

1. Oracle Database は、SQL 文で参照される名前の最初の断片を識別しようとします。たとえば、scott.emp の最初の断片は scott です。断片が 1 つしか存在しない場合、その断片は最初の断片とみなされます。

   a. 現行スキーマ内で、オブジェクト名の最初の断片に一致するオブジェクトが検索されます。そのようなオブジェクトが見つからない場合は、手順 b に進みます。

   b. オブジェクト名の最初の断片に一致するパブリック・シノニムが検索されます。そのようなシノニムが見つからない場合は、手順 c に進みます。

   c. オブジェクト名の最初の断片に一致するスキーマが検索されます。スキーマが検出された場合は手順 b に戻り、次はオブジェクト名の 2 番目の断片を使用して、識別されたスキーマ内が検索されます。2 番目の断片が、識別されたスキーマ内のオブジェクトに一致しない場合、または 2 番目の断片がない場合は、エラーが返されます。

   手順 c でスキーマが検出されない場合、そのオブジェクトは識別できず、エラーが返されます。

2. スキーマ・オブジェクトが識別されました。SQL 文で与えられた名前の残りの断片は、見つかったオブジェクトの有効な部分に一致する必要があります。たとえば、名前が scott.emp.deptno で、scott がスキーマとして識別され、emp が表として識別された場合は、（emp が表であるため）deptno は列に対応する必要があります。また、emp がパッケージとして識別された場合、deptno はそのパッケージのパブリック定数、変数、プロシージャまたはファンクションに対応する必要があります。

分散データベースにおいて、グローバル・オブジェクト名が明示的またはシノニム内で間接的に使用されている場合、ローカル・データベースはローカルで参照を解決します。たとえば、シノニムをリモート表のグローバル・オブジェクト名として解決します。部分的に解決された文はリモート・データベースに転送され、前述の手順に従って、リモート・データベースでオブジェクトの解決が行われます。

データベースによる参照の解決方法の関係で、あるオブジェクトが、他のオブジェクトが存在しないことに依存している可能性があります。この状況が発生するのは、依存するオブジェクトが使用している参照の解析方法が、他のオブジェクトが存在しているときには異なる場合です。たとえば、次のような場合を考えてみます。

- 現時点では、company スキーマに表 emp が含まれています。
- company.emp に対してパブリック・シノニム emp が作成され、company.emp に対する SELECT 権限が PUBLIC ロールに付与されます。
- jward スキーマには、表またはプライベート・シノニム emp は含まれていません。
- ユーザー jward が、次の文を使用して自分のスキーマにビューを作成します。

```
CREATE VIEW dept_salaries AS
     SELECT deptno, MIN(sal), AVG(sal), MAX(sal) FROM emp
     GROUP BY deptno
     ORDER BY deptno;
```

jward が dept_salaries ビューを作成すると、emp への参照は、jward.emp を表、ビューまたはプライベート・シノニムとして検索し、いずれも見つからない場合はパブリック・シノニム emp として検索して見つけることで解決されます。その結果、jward.dept_salaries は、jward.emp が存在しないことと、public.emp が存在することに依存していることがわかります。

ここで、jward が次の文を使用して自分のスキーマに新しいビュー emp を作成するとします。

```
CREATE VIEW emp AS
     SELECT empno, ename, mgr, deptno
     FROM company.emp;
```

jward.emp の構造が company.emp とは異なることに注意してください。

データベースは、オブジェクト定義内で参照を解決するときに、新しい依存オブジェクトの、存在しないオブジェクト（スキーマ・オブジェクト）への依存性に内部的に注目します。このスキーマ・オブジェクトが存在する場合は、オブジェクトの定義の解析が変化します。存在しないオブジェクトを後で作成する場合は、この種の依存性に注意する必要があります。存在しないオブジェクトを作成する場合は、依存オブジェクトを再コンパイルして検証できるように、すべての依存オブジェクトを無効にする必要があります。また、依存するすべてのファンクション索引を使用禁止としてマークする必要があります。

したがって、前述の例では、jward.emp が作成されると、jward.dept_salaries は jward.emp に依存するため無効になります。その後、jward.dept_salaries が使用されると、データベースはビューの再コンパイルを試みます。emp への参照を解決するときに、jward.emp が見つかります（public.emp は参照先のオブジェクトではなくなっています）。jward.emp には sal 列がないため、ビューを置換するときにエラーが見つかり、ビューは無効のままになります。

要約すると、存在しないオブジェクトを後で作成する場合は、オブジェクトの解決中にチェックされる存在しないオブジェクトへの依存性を管理する必要があります。

> **関連項目：** 分散データベースにおける名前解決の詳細は、29-16 ページの「スキーマ・オブジェクトとデータベース・リンク」を参照してください。

異なるスキーマへの切替え

次の文は、現行セッションのスキーマを、この文で指定するスキーマ名に設定します。

```
ALTER SESSION SET CURRENT_SCHEMA = <schema name>
```

その後の SQL 文では、修飾子が省略されている場合に、Oracle Database によって、このスキーマ名がスキーマ修飾子として使用されます。また、データベースでは、指定したスキーマの一時表領域が、一時データベース・オブジェクトのソート、結合および格納に使用されます。セッションには元の権限が保持され、前述の ALTER SESSION 文によって余分な権限は取得されません。

次の例では、プロンプトが表示されたら tiger というパスワードを入力します。

```
CONNECT scott
ALTER SESSION SET CURRENT_SCHEMA = joe;
SELECT * FROM emp;
```

emp は識別されたスキーマではないため、表名は joe スキーマのもとで解決されます。ただし、scott が joe.emp 表の選択権限を持たない場合、scott は SELECT 文を実行できません。

スキーマ・オブジェクト情報の表示

Oracle Database には PL/SQL パッケージが用意されており、スキーマ・オブジェクト情報の表示に使用できるオブジェクトおよびデータ・ディクショナリ・ビューを作成した DDL を判断できます。特定のタイプのスキーマ・オブジェクトに固有のビューとパッケージは、関連する章に記載されています。ここでは、汎用的な性質を持ち、複数のスキーマ・オブジェクトに適用されるビューとパッケージについて説明します。

PL/SQL パッケージを使用したスキーマ・オブジェクト情報の表示

オラクル社が提供する PL/SQL パッケージ DBMS_METADATA.GET_DDL を使用すると、スキーマ・オブジェクトに関するメタデータを（オブジェクトの作成に使用する DDL の形式で）取得できます。

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

例 : DBMS_METADATA パッケージの使用

DBMS_METADATA パッケージは、スキーマ・オブジェクトの完全な定義を取得できる強力なツールです。このパッケージを使用すると、あるオブジェクトのすべての属性を 1 回のパスで取得できます。オブジェクトは、その作成（再作成）に使用できる DDL で表されます。

次の文では、GET_DDL ファンクションを使用して、現行スキーマ内にあるすべての表の DDL をフェッチし、ネストした表とオーバーフロー・セグメントを除外しています。また、DDL で記憶域句が返されないようにするため、SET_TRANSFORM_PARAM（ハンドル値として「現行セッション用」を意味する DBMS_METADATA.SESSION_TRANSFORM をとる）を使用してそれを指定しています。セッション・レベルの変換パラメータは、最後にデフォルトにリセットされています。変換パラメータ値は、いったん設定すると、明示的にデフォルトにリセットされるまで有効です。

```
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(
     DBMS_METADATA.SESSION_TRANSFORM,'STORAGE',false);
SELECT DBMS_METADATA.GET_DDL('TABLE',u.table_name)
     FROM USER_ALL_TABLES u
     WHERE u.nested='NO'
     AND (u.iot_type is null or u.iot_type='IOT');
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(
     DBMS_METADATA.SESSION_TRANSFORM,'DEFAULT');
```

DBMS_METADATA.GET_DDL の出力は、LONG データ型です。SQL*Plus を使用している場合は、出力がデフォルトで切り捨てられる場合があります。出力が切り捨てられないようにするには、DBMS_METADATA.GET_DDL 文を発行する前に、次の SQL*Plus コマンドを発行してください。

```
SQL> SET LONG 9999
```

> **関連項目：** DBMS_METADATA パッケージの使用に関する詳細とその他の使用例については、『Oracle XML Developer's Kit プログラマーズ・ガイド』を参照してください。

スキーマ・オブジェクトのデータ・ディクショナリ・ビュー

次のビューには、スキーマ・オブジェクトに関する一般的な情報が表示されます。

ビュー	説明
DBA_OBJECTS ALL_OBJECTS USER_OBJECTS	DBA ビューには、データベース内のすべてのスキーマ・オブジェクトが表示されます。ALL ビューには、現行ユーザーがアクセス可能なオブジェクトが表示されます。USER ビューには、現行ユーザーが所有しているオブジェクトが表示されます。
DBA_CATALOG ALL_CATALOG USER_CATALOG	データベース内にあるすべての表、ビュー、シノニムおよび順序の名前、タイプおよび所有者（USER ビューでは所有者は表示されない）がリストされます。
DBA_DEPENDENCIES ALL_DEPENDENCIES USER_DEPENDENCIES	プロシージャ、パッケージ、ファンクション、パッケージ本体およびトリガーの間の依存性（データベース・リンクを持たないビューへの依存性など）がすべてリストされます。

関連項目： データ・ディクショナリ・ビューの詳細は、『Oracle Database リファレンス』を参照してください。

次に、これらのビューの使用例を示します。

- 例 1: スキーマ・オブジェクトのタイプ別表示
- 例 2: ビューとシノニムの依存性の表示

例 1: スキーマ・オブジェクトのタイプ別表示

次の問合せは、問合せを発行しているユーザーが所有しているオブジェクトをすべてリストします。

```
SELECT OBJECT_NAME, OBJECT_TYPE
    FROM USER_OBJECTS;
```

問合せの出力は次のとおりです。

```
OBJECT_NAME                    OBJECT_TYPE
-----------------------------  -------------------
EMP_DEPT                       CLUSTER
EMP                            TABLE
DEPT                           TABLE
EMP_DEPT_INDEX                 INDEX
PUBLIC_EMP                     SYNONYM
EMP_MGR                        VIEW
```

例 2: ビューとシノニムの依存性の表示

ビューまたはシノニムを作成するとき、ビューやシノニムはその基礎になるベース・オブジェクトに基づきます。ビューの依存性を明確にするには、ALL_DEPENDENCIES、USER_DEPENDENCIES および DBA_DEPENDENCIES データ・ディクショナリ・ビューを使用します。シノニムのベース・オブジェクトのリストを表示するには、ALL_SYNONYMS、USER_SYNONYMS および DBA_SYNONYMS データ・ディクショナリ・ビューを使用します。たとえば、次の問合せは、ユーザー jward によって作成されたシノニムのベース・オブジェクトをリストします。

```
SELECT TABLE_OWNER, TABLE_NAME, SYNONYM_NAME
    FROM DBA_SYNONYMS
    WHERE OWNER = 'JWARD';
```

問合せの出力は次のとおりです。

```
TABLE_OWNER                    TABLE_NAME   SYNONYM_NAME
------------------------------ -----------  -----------------
SCOTT                          DEPT         DEPT
SCOTT                          EMP          EMP
```

17

スキーマ・オブジェクトの領域の管理

この章の内容は次のとおりです。

表領域のアラートの管理

Oracle Database では、使用可能な領域が少なくなると事前にアラートで通知されるため、表領域のディスク領域を管理するのに役立ちます。デフォルトで、**警告**および**クリティカル**の 2 つのアラートしきい値が定義されています。警告のしきい値は、領域が残り少なくなり始める境界値です。クリティカルのしきい値は、即時に注意を喚起する必要のある深刻な境界値です。データベースは、両方のしきい値でアラートを発行します。

ローカル管理表領域とディクショナリ管理表領域の両方に対してアラートしきい値を指定するには、次の 2 つの方法があります。

- パーセント・フルによる方法

  警告のしきい値とクリティカルのしきい値の両方について、使用済領域が合計領域の一定割合以上になるとアラートが発行されます。

- 空き領域（KB 単位）による方法

  警告のしきい値とクリティカルのしきい値の両方について、空き領域が一定容量（KB）未満になるとアラートが発行されます。空き領域のしきい値は、表領域が大規模な場合に便利です。

ローカル管理表領域のアラートはサーバーで生成されます。ディクショナリ管理表領域の場合は、Enterprise Manager がこの機能を提供します。詳細は、7-4 ページの「サーバー生成アラートを使用したデータベースの動作の監視」を参照してください。

新しい表領域には、次のようにアラートしきい値が割り当てられます。

- **ローカル管理表領域**: ローカル管理表領域を新規作成すると、データベースに定義されているデフォルトのしきい値がその表領域に割り当てられます。新しく作成されたデータベースには、警告のしきい値に 85% 使用済、クリティカルのしきい値に 97% 使用済のデフォルトが割り当てられます。新しいデータベースに対する空き領域のしきい値のデフォルトは、両方ともゼロ（無効）です。これらのデータベースのデフォルトは変更可能で、その手順については後で説明します。

- **ディクショナリ管理表領域**: ディクショナリ管理表領域を新規作成すると、Enterprise Manager のメトリック・カテゴリ「表領域の空き領域 (MB)(ディクショナリ管理)」および「表領域使用率 (%)(ディクショナリ管理)」の「その他すべて」にリストされているしきい値が割り当てられます。これらの値は、「メトリックとポリシー設定」ページで変更できます。

注意： バージョン 9.x 以前から 10.x にアップグレードしたデータベースでは、すべてのローカル管理表領域のアラートしきい値はデフォルトで 0（ゼロ）に設定されます。この設定は、アラート・メカニズムを事実上使用禁止にして、新しく移行されたデータベースへの過剰なアラートを回避しています。

アラートしきい値の設定

各表領域には、パーセント・フルのしきい値のみ、空き領域のしきい値のみ、または同時に両方のしきい値タイプを設定できます。どちらのタイプのしきい値も、0（ゼロ）に設定すると無効になります。

理想的な警告のしきい値は、クリティカルのしきい値が発行される前に問題を解決できる時間を考慮して、早めにアラートを発行する設定です。クリティカルのしきい値は、ただちに処理してサービスの損失を回避できるよう十分早めにアラートを発行するように設定します。

アラートしきい値の設定方法

- ローカル管理表領域の設定には、Enterprise Manager（手順については『Oracle Database 2 日でデータベース管理者』を参照）または DBMS_SERVER_ALERT.SET_THRESHOLD パッケージ・プロシージャ（使用方法の詳細は『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照）を使用します。
- ディクショナリ管理表領域の設定には、Enterprise Manager を使用します。手順については、『Oracle Database 2 日でデータベース管理者』を参照してください。

例：ローカル管理表領域

次の例は、USERS 表領域について空き領域のしきい値を 10MB（警告）および 2MB（クリティカル）に設定し、パーセント・フルのしきい値を無効にします。

```
BEGIN
DBMS_SERVER_ALERT.SET_THRESHOLD(
   metrics_id              => DBMS_SERVER_ALERT.TABLESPACE_BYT_FREE,
   warning_operator        => DBMS_SERVER_ALERT.OPERATOR_LE,
   warning_value           => '10240',
   critical_operator       => DBMS_SERVER_ALERT.OPERATOR_LE,
   critical_value          => '2048',
   observation_period      => 1,
   consecutive_occurrences => 1,
   instance_name           => NULL,
   object_type             => DBMS_SERVER_ALERT.OBJECT_TYPE_TABLESPACE,
   object_name             => 'USERS');

DBMS_SERVER_ALERT.SET_THRESHOLD(
   metrics_id              => DBMS_SERVER_ALERT.TABLESPACE_PCT_FULL,
   warning_operator        => DBMS_SERVER_ALERT.OPERATOR_GT,
   warning_value           => '0',
   critical_operator       => DBMS_SERVER_ALERT.OPERATOR_GT,
   critical_value          => '0',
   observation_period      => 1,
   consecutive_occurrences => 1,
   instance_name           => NULL,
   object_type             => DBMS_SERVER_ALERT.OBJECT_TYPE_TABLESPACE,
   object_name             => 'USERS');
END;
/
```

注意： パーセント・フルのしきい値に 0（ゼロ）以外の値を設定する場合は、「以上」演算子 OPERATOR_GE を使用します。

データベースのデフォルトしきい値への表領域のリストア

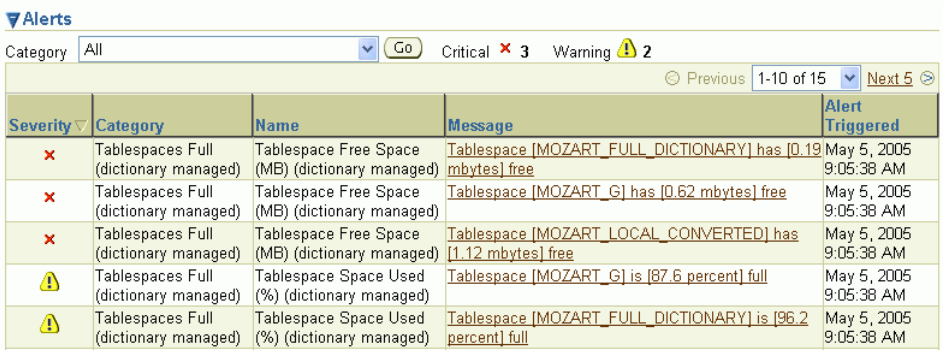
ローカル管理表領域のアラートしきい値に明示的に値を設定した後は、その値を DBMS_SERVER_ALERT.SET_THRESHOLD を使用して NULL に設定することで、データベースのデフォルト値に戻すことができます。

データベースのデフォルトしきい値の変更

ローカル管理表領域のデータベース・デフォルトしきい値を変更するには、前述の例のように DBMS_SERVER_ALERT.SET_THRESHOLD を起動します。ただし、この場合は、object_name を NULL に設定します。データベース・デフォルトを使用する表領域はすべて新しいデフォルトに切り替わります。

アラートの表示

アラートを表示するには、Enterprise Manager Database Control のホームページにアクセスします。



ローカル管理表領域のアラートは、DBA_OUTSTANDING_ALERTS ビューを使用して表示することもできます。詳細は、7-6 ページの「サーバー生成アラートのデータ・ディクショナリ・ビュー」を参照してください。

制限事項

しきい値ベースのアラートには、次の制限があります。

- アラートは、オフラインまたは読取り専用モードのローカル管理表領域については発行されません。ただし、それらの表領域が読取り / 書込みモードまたは使用可能になると、そのアラート・システムは再アクティブ化されます。
- 表領域をオフライン化または読取り専用モードにする場合は、しきい値を 0（ゼロ）に設定して、表領域に対するアラートを使用禁止にする必要があります。後で表領域を再度オンライン化または読取り / 書込みモードにする場合は、しきい値を再設定してアラートを再び使用可能にできます。

関連項目：

- サーバー生成アラートに関する一般的な詳細は、7-4 ページの「サーバー生成アラートを使用したデータベースの動作の監視」を参照してください。
- DBMS_SERVER_ALERT パッケージのプロシージャとその使用方法の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 自動ワークロード・リポジトリを使用して領域使用の統計を収集する方法は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- 表領域内で使用されていない領域を再生する方法は、17-12 ページの「使用できない領域の再生」を参照してください。
- リサイクル・ビン領域を再生する方法は、18-45 ページの「リサイクル・ビン内のオブジェクトのパージ」を参照してください。

再開可能領域割当ての管理

Oracle Database では、領域割当てが失敗した場合に大規模なデータベース処理を一時停止して、後で再開するための方法が提供されています。これにより、Oracle Database サーバーがユーザーにエラーを返すかわりに対処措置を講じることができます。エラー条件が訂正されると、一時停止していた処理が自動的に再開します。この機能のことを、**再開可能領域割当て**と呼びます。また、影響を受ける文のことを、再開可能文と呼びます。

この項の内容は、次のとおりです。

再開可能領域割当ての概要

ここでは、再開可能領域割当ての概要について説明します。再開可能領域割当ての動作について説明し、修飾文とエラー条件を具体的に定義します。

再開可能領域割当ての動作

再開可能領域割当ての動作の概要は、次のとおりです。詳細はこの後の各項で説明します。

1. 文が再開可能モードで実行されるのは、その文のセッションが、次のいずれかの処理によって再開可能領域割当てに対応している場合のみです。
   - `RESUMABLE_TIMEOUT` 初期化パラメータが 0（ゼロ）以外の値に設定された。
   - `ALTER SESSION ENABLE RESUMABLE` 文が発行された。
2. 次のいずれかの条件が成立すると、再開可能文が一時停止します（非再開可能文では、これらの条件に対応するエラーが通知されます）。
   - 領域不足条件
   - 最大エクステント数到達条件
   - スペース割当制限超過条件
3. 再開可能文の実行が一時停止すると、ユーザー指定の操作の実行、エラーの記録および文の実行状態の問合せを行うメカニズムがただちに動作します。再開可能文が一時停止すると、次の処理が実行されます。
   - エラーがアラート・ログに記録されます。
   - 一時停止された再開可能セッションのアラートが発行されます。
   - ユーザーが `AFTER SUSPEND` システム・イベントに対してトリガーを登録していた場合は、そのユーザー・トリガーが実行されます。ユーザー指定の PL/SQL プロシージャは、`DBMS_RESUMABLE` パッケージと `DBA_RESUMABLE` または `USER_RESUMABLE` ビューを使用して、エラー・メッセージ・データにアクセスできます。
4. 文の一時停止によって自動的にトランザクションが一時停止します。その結果、すべてのトランザクション・リソースが文の一時停止から再開までの間保持されます。
5. ユーザーの介入や他の問合せによってソート領域が解放されるなどの結果としてエラー条件が解決されると、一時停止していた文が自動的に実行を再開し、一時停止された再開可能セッションのアラートがクリアされます。
6. 一時停止した文は、`DBMS_RESUMABLE.ABORT()` プロシージャを使用して、強制的に例外を発生できます。このプロシージャは、DBA または文を発行したユーザーがコールできます。

7. 一時停止のタイムアウト間隔は、再開可能文に対応付けられています。タイムアウト間隔（デフォルトは 2 時間）の間一時停止していた再開可能文がリストアすると、ユーザーに例外が返されます。
8. 再開可能文は、実行中に一時停止と再開を複数回繰り返すことができます。

再開可能な操作

再開可能な操作は、次のとおりです。

- 問合せ

  一時領域（ソート領域）を使い果たした SELECT 文は、再開可能な実行の候補になります。Oracle Call Interface（OCI）の使用時は、OCIStmtExecute() および OCIStmtFetch() の各コールが候補になります。

- DML

  INSERT、UPDATE および DELETE の各文が候補になります。各文の実行に使用されているインタフェースには関係ありません。OCI、SQLJ、PL/SQL やその他のインタフェースでも有効です。外部表からの INSERT INTO...SELECT も再開可能にできます。

- インポート / エクスポート

  SQL*Loader については、リカバリ可能なエラーの後に文が再開可能かどうかをコマンドライン・パラメータで制御します。

- DDL

  次の文が、再開可能な実行の候補になります。

  - CREATE TABLE ...AS SELECT
  - CREATE INDEX
  - ALTER INDEX ...REBUILD
  - ALTER TABLE ...MOVE PARTITION
  - ALTER TABLE ...SPLIT PARTITION
  - ALTER INDEX ...REBUILD PARTITION
  - ALTER INDEX ...SPLIT PARTITION
  - CREATE MATERIALIZED VIEW
  - CREATE MATERIALIZED VIEW LOG

訂正可能なエラー

訂正可能なエラーには、次の 3 つのクラスがあります。

- 領域不足条件

  データベースの操作によって、表領域内の表、索引、一時セグメント、UNDO セグメント、クラスタ、LOB、表パーティションまたは索引パーティション用のエクステントをこれ以上取得できません。たとえば、次のエラーはこのカテゴリに属します。

  ```
  ORA-1653 unable to extend table ... in tablespace ...
  ORA-1654 unable to extend index ... in tablespace ...
  ```

- 最大エクステント数到達条件

  表、索引、一時セグメント、UNDO セグメント、クラスタ、LOB、表パーティションまたは索引パーティション内のエクステント数が、オブジェクトで定義されている最大エクステント数に等しくなりました。たとえば、次のエラーはこのカテゴリに属します。

  ```
  ORA-1631 max # extents ... reached in table ...
  ORA-1654 max # extents ... reached in index ...
  ```

- スペース割当制限超過条件

  ユーザーが自分に割り当てられている表領域内のスペース割当制限を超過しました。具体的には、次のエラーによって示されます。

  ```
  ORA-1536 space quote exceeded for tablespace string
  ```

再開可能領域割当てと分散処理

分散環境で、ユーザーが再開可能領域割当てを有効または無効にした場合、または DBA として RESUMABLE_TIMEOUT 初期化パラメータを変更した場合、その影響を受けるのはローカルのインスタンスのみです。分散トランザクションで、セッションまたはリモート・インスタンスが一時停止されるのは、RESUMABLE がリモート・インスタンスで有効な場合のみです。

パラレル実行と再開可能領域割当て

パラレル実行では、パラレル実行サーバー・プロセスの 1 つで訂正可能なエラーが発生した場合、そのサーバー・プロセスの実行が一時停止します。他のパラレル実行サーバー・プロセスでは、エラーが発生するまで、または一時停止したサーバー・プロセスに（直接または間接に）ブロックされるまで、個々のタスクの実行が継続されます。訂正可能なエラーが解決すると、一時停止したプロセスが実行を再開し、パラレル操作の実行は継続されます。一時停止したプロセスが終了した場合、パラレル操作は中断し、ユーザーに対してエラーが返されます。

異なるパラレル実行プロセスで、1 つ以上の訂正可能なエラーが発生することがあります。この結果、AFTER SUSPEND トリガーが複数回、パラレルに発行される場合があります。また、あるパラレル実行サーバー・プロセスが一時停止中に他のパラレル実行サーバー・プロセスで訂正不可能なエラーが発生すると、一時停止していた文はただちに強制終了します。

パラレル実行については、すべてのパラレル実行コーディネータとサーバー・プロセスが DBA_RESUMABLE または USER_RESUMABLE ビューに独自のエントリを持っています。

再開可能領域割当ての有効化および無効化

再開可能領域割当ては、再開可能モードを有効にしたセッションの中で文を実行するときのみ可能です。再開可能領域割当ての有効化と無効化には 2 通りの方法があります。再開可能領域割当ては、RESUMABLE_TIMEOUT 初期化パラメータを使用してシステム・レベルで制御するか、または ALTER SESSION 文の句を使用してセッション・レベルで有効化できます。

> **注意：** 一時停止した文はなんらかのシステム・リソースを保持している可能性があるため、ユーザーが再開可能領域割当てを有効にして、再開可能文を実行するには、RESUMABLE システム権限が付与されている必要があります。

RESUMABLE_TIMEOUT 初期化パラメータの設定

再開可能領域割当てをシステム全体で有効化し、RESUMABLE_TIMEOUT 初期化パラメータを設定してタイムアウト間隔を指定できます。たとえば、初期化パラメータ・ファイルの RESUMABLE_TIMEOUT パラメータを次のように設定することによって、すべてのセッションは、初期状態で再開可能領域割当てが有効になり、タイムアウト間隔が 1 時間に設定されます。

```
RESUMABLE_TIMEOUT  = 3600
```

このパラメータが 0（ゼロ）に設定された場合、すべてのセッションに対する再開可能領域割当ては、初期状態で無効になります。これはデフォルトです。

ALTER SYSTEM SET 文を使用すると、このパラメータの値をシステム・レベルで変更できます。たとえば、次の文では、すべてのセッションに対して再開可能領域割当てが無効になります。

```
ALTER SYSTEM SET RESUMABLE_TIMEOUT=0;
```

セッション内で、ユーザーは ALTER SESSION SET 文を発行して RESUMABLE_TIMEOUT 初期化パラメータを設定し、再開可能領域割当てを有効にしてタイムアウト値を変更するか、または再開可能モードを無効にできます。

ALTER SESSION を使用した再開可能領域割当ての有効化と無効化

次の SQL 文を使用して、セッションの再開可能モードを有効化できます。

```
ALTER SESSION ENABLE RESUMABLE;
```

再開可能モードを無効化するには、次の文を使用します。

```
ALTER SESSION DISABLE RESUMABLE;
```

RESUMABLE_TIMEOUT 初期化パラメータが 0（ゼロ）の値に設定されている場合、新しいセッションのデフォルトの再開可能モードは無効です。

タイムアウト間隔や、再開可能文の識別に使用する名前も指定できます。次の項では、各操作について説明します。

> **関連項目：** 17-8 ページ「LOGON トリガーを使用したデフォルト再開可能モードの設定」

タイムアウト間隔の指定 介入操作が発生しなかった場合に一時停止した文がエラーとなるタイムアウト間隔は、再開可能モードを有効化するときに指定できます。次の文は、再開可能トランザクションが 3600 秒後にタイムアウトし、エラーになることを指定します。

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600;
```

TIMEOUT の値は、別の ALTER SESSION ENABLE RESUMABLE 文や他の手段によって変更されるまで、またはセッションが終了するまで有効です。ENABLE RESUMABLE TIMEOUT 句を使用して再開可能モードを有効化する場合、デフォルトのタイムアウト間隔は 7200 秒です。

> **関連項目：** 再開可能領域割当てのタイムアウト間隔を変更するその他の方法の詳細は、17-7 ページの「RESUMABLE_TIMEOUT 初期化パラメータの設定」を参照してください。

再開可能文の命名 再開可能文は、名前で識別するように設定できます。次の文は、再開可能文に名前を割り当てます。

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600 NAME 'insert into table';
```

NAME の値は、別の ALTER SESSION ENABLE RESUMABLE 文によって変更されるまで、またはセッションが終了するまで有効です。NAME のデフォルト値は、'User *username*(*userid*), Session *sessionid*, Instance *instanceid*' です。

文の名前は、DBA_RESUMABLE ビューおよび USER_RESUMABLE ビューで再開可能文を識別する際に使用します。

LOGON トリガーを使用したデフォルト再開可能モードの設定

RESUMABLE_TIMEOUT 初期化パラメータを設定する以外に、デフォルトの再開可能モードを設定するもう 1 つの方法は、データベース・レベルの LOGON トリガーを登録して、再開可能を有効化してタイムアウト間隔を設定するように、ユーザーのセッションを変更する方法です。

注意： 再開可能文のデフォルトのモードとタイムアウトを変更する登録済トリガーが複数ある場合、その結果は予測できません。これは、トリガーの起動順序が Oracle Database で保証されていないためです。

一時停止文の検出

再開可能文が一時停止するとき、クライアントにはエラーは通知されません。訂正処理を実行するため、Oracle Database ではユーザーにエラーを通知して状況に関する情報を提供するかわりの手段が提供されています。

ユーザーへの通知 : AFTER SUSPEND システム・イベントおよびトリガー

再開可能文で訂正可能なエラーが発生すると、システムは内部的に AFTER SUSPEND システム・イベントを生成します。ユーザーは、このイベントに対するトリガーをデータベース・レベルとスキーマ・レベルの両方で登録できます。ユーザーがこのシステム・イベントを処理するトリガーを登録した場合、トリガーは SQL 文が一時停止した後に実行されます。

AFTER SUSPEND トリガー内部で実行された SQL 文は、再開不可能であり、かつ自律型です。トリガー内部で開始されたトランザクションは、SYSTEM ロールバック・セグメントを使用します。これらの条件が課されるのは、デッドロックを回避し、文と同じエラー条件に直面する可能性を少なくするためです。

ユーザーは USER_RESUMABLE ビュー、DBA_RESUMABLE ビューまたは DBMS_RESUMABLE.SPACE_ERROR_INFO ファンクションをトリガー内部で使用して、再開可能文に関する情報を取得できます。

また、トリガーでは DBMS_RESUMABLE パッケージをコールして、一時停止した文の終了や再開可能タイムアウト値の変更を実行できます。次の例では、DBMS_RESUMABLE をコールしてタイムアウトを 3 時間に設定する、システム全体で有効な AFTER SUSPEND トリガーを作成することによって、デフォルトのシステム・タイムアウトを変更します。

```
CREATE OR REPLACE TRIGGER resumable_default_timeout
AFTER SUSPEND
ON DATABASE
BEGIN
   DBMS_RESUMABLE.SET_TIMEOUT(10800);
END;
```

関連項目： トリガーおよびシステム・イベントに関する情報は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

ビューを使用した一時停止文情報の取得

次のビューを問い合せることにより、再開可能文の状態に関する情報を取得できます。

ビュー	説明
DBA_RESUMABLE USER_RESUMABLE	これらのビューには、現在実行中または一時停止しているすべての再開可能文に関する行が含まれます。これらは、再開可能文の実行状態を監視したり、再開可能文に関する特定の情報を取得するために、DBA、AFTER SUSPEND トリガーまたは別のセッションが使用できます。
V$SESSION_WAIT	ある文が一時停止すると、その文を起動したセッションは待機状態になります。このビューには、「文が一時停止され、エラーのクリアを待機しています」という内容の EVENT 列を持つセッションに対して 1 行が挿入されます。

関連項目： これらのビューに含まれる列の詳細は、『Oracle Database リファレンス』を参照してください。

DBMS_RESUMABLE パッケージの使用方法

DBMS_RESUMABLE パッケージは、再開可能領域割当ての管理に役立ちます。次のプロシージャを起動できます。

プロシージャ	説明
ABORT(sessionID)	このプロシージャは、一時停止した再開可能文を強制終了します。パラメータ sessionID は、文が実行されているセッション ID です。パラレル DML/DDL の場合、sessionID はパラレル DML/DDL に参加している任意のセッション ID です。 Oracle Database では、ABORT 操作が常に正常終了することが保証されています。ABORT は、AFTER SUSPEND トリガーの内部または外部のどちらでもコールできます。 ABORT のコール元は、sessionID を持つセッションの所有者、ALTER SYSTEM 権限を持っているユーザー、DBA 権限を持っているユーザーのいずれかである必要があります。
GET_SESSION_ TIMEOUT(sessionID)	このファンクションは、sessionID を持つセッションで設定されている再開可能領域割当ての現行のタイムアウト値を返します。タイムアウトは秒数で返されます。セッションが存在しない場合、このファンクションは -1 を返します。
SET_SESSION_ TIMEOUT(sessionID, timeout)	このプロシージャは、sessionID を持つセッションに対して再開可能領域割当てのタイムアウト間隔を設定します。パラメータ timeout の単位は秒数です。新しい timeout 設定は、即時にセッションに適用されます。セッションが存在しない場合は何も起こりません。
GET_TIMEOUT()	このファンクションは、現行セッションで設定されている再開可能領域割当ての現行の timeout 値を返します。値は秒数で返されます。
SET_ TIMEOUT(timeout)	このプロシージャは、現行セッションに対して再開可能領域割当ての timeout 値を設定します。パラメータ timeout の単位は秒数です。新しいタイムアウト設定は、即時にセッションに適用されます。

関連項目：『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』

操作一時停止アラート

再開可能セッションが一時停止されると、操作を完了するためにリソースを割り当てる必要があるオブジェクトに対して、操作一時停止アラートが発行されます。リソースが割り当てられて操作が完了すると、操作一時停止アラートはクリアされます。システム生成アラートの詳細は、17-2 ページの「表領域のアラートの管理」を参照してください。

再開可能領域割当ての例 : AFTER SUSPEND トリガーの登録

次の例では、システム全体で有効な AFTER SUSPEND トリガーを作成し、ユーザー SYS としてデータベース・レベルで登録します。任意のセッションで再開可能文が一時停止すると、このトリガーは次の 2 つのうちどちらかの処理を実行します。

- UNDO セグメントが領域上限に達した場合は、メッセージが DBA に送られ、文が強制終了します。
- 他のリカバリ可能なエラーが発生した場合には、タイムアウト間隔が 8 時間にリセットされます。

この例で使用する文を次に示します。

```
CREATE OR REPLACE TRIGGER resumable_default
AFTER SUSPEND
ON DATABASE
DECLARE
   /* declare transaction in this trigger is autonomous */
   /* this is not required because transactions within a trigger
      are always autonomous */
   PRAGMA AUTONOMOUS_TRANSACTION;
   cur_sid           NUMBER;
   cur_inst          NUMBER;
   errno             NUMBER;
   err_type          VARCHAR2;
   object_owner      VARCHAR2;
   object_type       VARCHAR2;
   table_space_name  VARCHAR2;
   object_name       VARCHAR2;
   sub_object_name   VARCHAR2;
   error_txt         VARCHAR2;
   msg_body          VARCHAR2;
   ret_value         BOOLEAN;
   mail_conn         UTL_SMTP.CONNECTION;
BEGIN
   -- Get session ID
   SELECT DISTINCT(SID) INTO cur_SID FROM V$MYSTAT;

   -- Get instance number
   cur_inst := userenv('instance');

   -- Get space error information
   ret_value :=
   DBMS_RESUMABLE.SPACE_ERROR_INFO(err_type,object_type,object_owner,
        table_space_name,object_name, sub_object_name);
   /*
   -- If the error is related to undo segments, log error, send email
   -- to DBA, and abort the statement. Otherwise, set timeout to 8 hours.
   --
   -- sys.rbs_error is a table which is to be
   -- created by a DBA manually and defined as
   -- (sql_text VARCHAR2(1000), error_msg VARCHAR2(4000),
   -- suspend_time DATE)
   */

   IF OBJECT_TYPE = 'UNDO SEGMENT' THEN
       /* LOG ERROR */
       INSERT INTO sys.rbs_error (
           SELECT SQL_TEXT, ERROR_MSG, SUSPEND_TIME
           FROM DBMS_RESUMABLE
           WHERE SESSION_ID = cur_sid AND INSTANCE_ID = cur_inst
        );
       SELECT ERROR_MSG INTO error_txt FROM DBMS_RESUMABLE
           WHERE SESSION_ID = cur_sid and INSTANCE_ID = cur_inst;
```

```
        -- Send email to receipient via UTL_SMTP package
        msg_body:='Subject: Space Error Occurred

                   Space limit reached for undo segment ' || object_name ||
                   on ' || TO_CHAR(SYSDATE, 'Month dd, YYYY, HH:MIam') ||
                   '. Error message was ' || error_txt;

        mail_conn := UTL_SMTP.OPEN_CONNECTION('localhost', 25);
        UTL_SMTP.HELO(mail_conn, 'localhost');
        UTL_SMTP.MAIL(mail_conn, 'sender@localhost');
        UTL_SMTP.RCPT(mail_conn, 'recipient@localhost');
        UTL_SMTP.DATA(mail_conn, msg_body);
        UTL_SMTP.QUIT(mail_conn);

        -- Abort the statement
        DBMS_RESUMABLE.ABORT(cur_sid);
    ELSE
        -- Set timeout to 8 hours
        DBMS_RESUMABLE.SET_TIMEOUT(28800);
    END IF;

    /* commit autonomous transaction */
    COMMIT;
END;
/
```

使用できない領域の再生

ここでは、使用できなくなっている領域の再生方法について説明します。また、再生可能な領域が存在しているセグメントを特定する Oracle Database のコンポーネントであるセグメント・アドバイザについても紹介します。この項の内容は、次のとおりです。

- 再生可能な未使用領域の理解
- セグメント・アドバイザの使用
- オンラインによるデータベース・セグメントの縮小
- 未使用領域の割当て解除

再生可能な未使用領域の理解

時間の経過とともに、表領域内のオブジェクトを更新および削除すると、新しいデータに対して個別に再利用するには不十分な小さい空き領域が作成されます。このような空き領域は、断片化された空き領域と呼ばれます。

オブジェクトに断片化された空き領域があると、結果的に多くの無駄な領域が生じ、データベースのパフォーマンスに影響を与える場合があります。断片化を解消して領域を再生するには、**オンラインによるセグメントの縮小**を実行することをお薦めします。このプロセスは、最高水位標の下の断片化された空き領域を統合し、セグメントを圧縮します。圧縮すると最高水位標が移動し、その結果、最高水位標の上に新しい空き領域ができます。その後、この最高水位標よりも上の領域は、割当て解除されます。セグメントは、この操作の開始から終了までの大半で、問合せおよび DML を使用でき、特別なディスク領域の割当ては不要です。

オンラインによるセグメントの縮小で利点を得られるセグメントを特定するには、**セグメント・アドバイザ**を使用します。セグメント・アドバイザは、自動セグメント領域管理（ASSM）を備えたローカル管理表領域のセグメントに対してのみ使用できます。調査可能なセグメントの種類については、他にも制限があります。詳細は、17-25 ページの「オンラインによるデータベース・セグメントの縮小」を参照してください。

再生可能な領域を含む表がオンラインによるセグメントの縮小に適していない場合、または領域の再生中に表の論理属性または物理属性を変更する場合は、セグメントの縮小にかわる手段として、**表のオンライン再定義**を使用できます。オンライン再定義は**再編成**とも呼ばれます。

オンライン再定義は、オンラインによるセグメントの縮小とは異なり、別のディスク領域を割り当てる必要があります。詳細は、18-26 ページの「表のオンライン再定義」を参照してください。

セグメント・アドバイザの使用

セグメント・アドバイザは、再生可能な領域があるセグメントを特定します。セグメント・アドバイザは、自動ワークロード・リポジトリ（AWR）内の使用統計や増加統計を調査したり、セグメントのデータをサンプリングすることで分析を実行します。セグメント・アドバイザは、メンテナンス・ウィンドウの実行時に自動化メンテナンス・タスクとして実行するように構成されていますが、必要時に（手動で）実行することもできます。セグメント・アドバイザの自動化メンテナンス・タスクを自動セグメント・アドバイザと呼びます。

セグメント・アドバイザは、次の種類のアドバイスを生成します。

- セグメント・アドバイザにより、オブジェクトにかなりの量の空き領域があることが判断されると、オンラインによるセグメントの縮小が提案されます。自動セグメント領域管理のない表領域内の表の場合など、オブジェクトがセグメントの縮小に適さない表である場合は、オンラインによる表の再定義が提案されます。
- セグメント・アドバイザにより、特定のしきい値を超える行連鎖のある表が検出されると、表に過剰な連鎖行があることが記録されます。

> **注意：** セグメント・アドバイザによるフラグ設定の対象は、行を長くする更新によって生じた行連鎖のタイプのみです。

領域管理のアラートを受け取った場合、または領域の再生を決定した場合は、セグメント・アドバイザを開始してください。

セグメント・アドバイザを使用する手順

1. 自動セグメント・アドバイザの結果を確認します。

   自動セグメント・アドバイザを理解するには、この後の「自動セグメント・アドバイザ」を参照してください。結果の表示方法の詳細は、17-18 ページの「セグメント・アドバイザの結果の表示」を参照してください。

2. （オプション）セグメント・アドバイザを手動で再実行し、個々のセグメントの更新結果を取得します。

   この後の「手動によるセグメント・アドバイザの実行」を参照してください。

自動セグメント・アドバイザ

自動セグメント・アドバイザは、すべてのメンテナンス・ウィンドウ内で実行するように構成されている自動化メンテナンス・タスクです。

自動セグメント・アドバイザは、すべてのデータベース・オブジェクトを分析するわけではありません。かわりに、データベース統計を調査し、セグメント・データをサンプリングした後、次のような分析対象オブジェクトを選択します。

- 領域のクリティカルまたは警告のしきい値を超えた表領域
- アクティビティが最も多いセグメント
- 増加率が最も高いセグメント

分析対象オブジェクトが選択されても、セグメント・アドバイザがオブジェクトを処理する前にメンテナンス・ウィンドウが終了する場合、そのオブジェクトは、自動セグメント・アドバイザの次回の実行に組み込まれます。

自動セグメント・アドバイザによって分析対象として選択された表領域とセグメントのセットは変更できません。ただし、自動セグメント・アドバイザのタスクの有効化または無効化、自動セグメント・アドバイザの実行予定回数、または自動化メンテナンス・タスクのシステム・

リソース使用率は変更できます。詳細は、17-24 ページの「自動セグメント・アドバイザの構成」を参照してください。

関連項目：

- 17-18 ページ「セグメント・アドバイザの結果の表示」
- 第 24 章「自動データベース・メンテナンス・タスクの管理」

手動によるセグメント・アドバイザの実行

セグメント・アドバイザは、Enterprise Manager または PL/SQL パッケージのプロシージャ・コールを使用して、いつでも手動で実行できます。セグメント・アドバイザを手動で実行する理由には、次のものがあります。

- 自動セグメント・アドバイザが選択しなかった表領域またはセグメントを分析するため。
- 個々の表領域またはセグメントを再度分析して、最新の推奨事項を入手するため。

セグメント・アドバイザには、3 種類のレベルでアドバイスを要求できます。

- **セグメント・レベル :** 非パーティション表、パーティション表のパーティションまたはサブパーティション、索引、または LOB 列など、単一のセグメントに対してアドバイスが生成されます。
- **オブジェクト・レベル** : 表や索引など、オブジェクト全体についてアドバイスが生成されます。オブジェクトがパーティション化されている場合は、オブジェクトのすべてのパーティションに対してアドバイスが生成されます。また、Enterprise Manager からセグメント・アドバイザを手動で実行する場合は、表の索引や LOB セグメントなど、オブジェクトの依存オブジェクトについてアドバイスを要求できます。
- **表領域レベル** : 表領域のすべてのセグメントに対してアドバイスが生成されます。

17-17 ページの表 17-2 にある `OBJECT_TYPE` 列は、アドバイスを要求できるオブジェクトのタイプを示しています。

Enterprise Manager を使用した手動によるセグメント・アドバイザの実行 Enterprise Manager を使用してセグメント・アドバイザを手動で実行するには、`OEM_ADVISOR` ロールが付与されている必要があります。セグメント・アドバイザを実行するには、次の 2 つの方法があります。

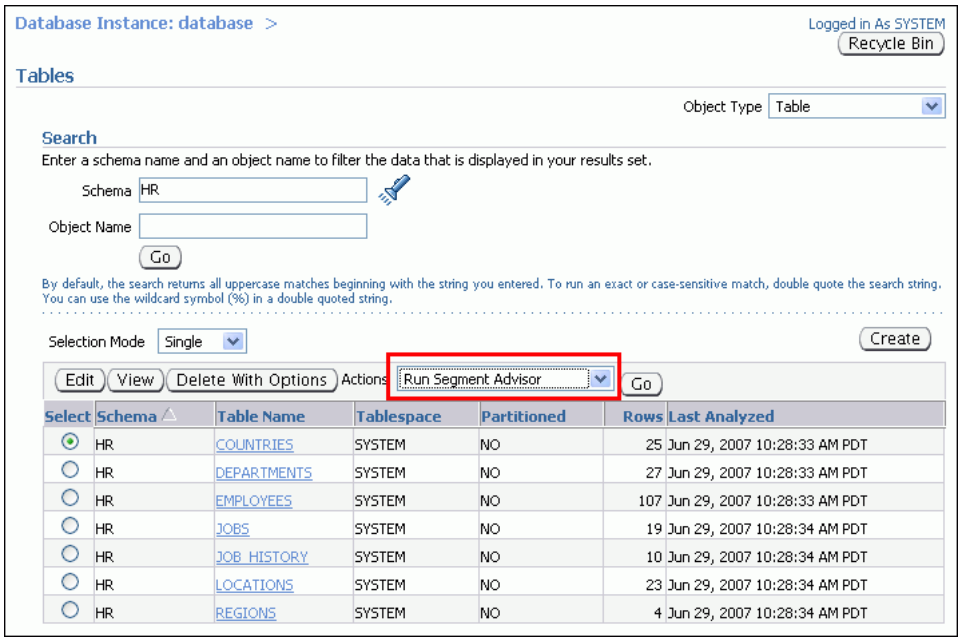
- セグメント・アドバイザ・ウィザードの使用

  この方法を使用すると、表領域レベルまたはオブジェクト・レベルでアドバイスを要求できます。オブジェクト・レベルでは、表、索引、表パーティションおよび索引パーティションについてアドバイスを要求できます。LOB セグメントなどの依存オブジェクトは、分析の対象にはなりません。

- スキーマ・オブジェクトを表示するページでの「セグメント・アドバイザの実行」コマンドの使用

  たとえば、(「スキーマ」ページからアクセス可能な)「表」ページに表のリストを表示する場合は、表を選択して「アクション」メニューから**「セグメント・アドバイザの実行」**コマンドを選択します。

図 17-1「表」ページ



この方法では、スキーマ・オブジェクトの依存オブジェクトをセグメント・アドバイザの実行対象に含めることができます。たとえば、表を選択して**「セグメント・アドバイザの実行」**コマンドを選択すると、パーティション、索引セグメント、LOB セグメントなど、表の依存オブジェクトが表示されます。依存オブジェクトを選択することで、そのオブジェクトを実行対象に指定できます。

いずれの場合も、セグメント・アドバイザのタスクが Oracle Database のスケジューラのジョブとして作成されます。ジョブは、ただちに実行するようにスケジュールできます。あるいは、スケジューラが提供する高度なスケジューリング機能を活用できます。

セグメント・アドバイザ・ウィザードを使用してセグメント・アドバイザを手動で実行する手順

1. データベース・ホームページから、「関連リンク」の下の**「アドバイザ・セントラル」**をクリックします。

   「アドバイザ・セントラル」ページが表示されます（図 17-2 を参照してください）。

2. 「アドバイザ」の下の**「セグメント・アドバイザ」**をクリックします。

   セグメント・アドバイザ・ウィザードの最初のページが表示されます。

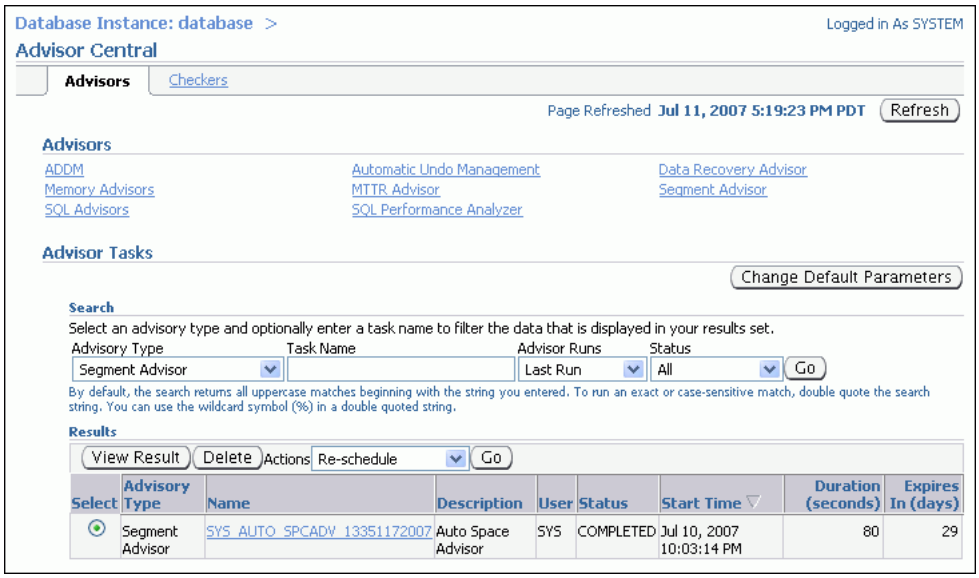
3. セグメント・アドバイザのジョブをスケジュールするウィザードの各手順に従い、ウィザードの最終ページで**「発行」**をクリックします。

   「アドバイザ・セントラル」ページが再度表示され、「結果」ヘッダー下のリスト上部にセグメント・アドバイザの新しいジョブが表示されます。ジョブ・ステータスは SCHEDULED または RUNNING となります（ジョブが表示されない場合は、リスト上部の検索フィールドを使用して表示します）。

4. ジョブのステータスを確認します。ステータスが COMPLETED でない場合は、ページの上部にある**「リフレッシュ」**ボタンを繰り返しクリックします（使用しているブラウザのリフレッシュ・アイコンは使用しないでください）。

   ジョブ・ステータスが COMPLETED に変わった場合は、**「選択」**列をクリックしてジョブを選択し、**「結果の表示」**をクリックします。

図 17-2「アドバイザ・セントラル」ページ



関連項目： スケジューラの高度なスケジュール機能の詳細は、第 27 章「Oracle Scheduler を使用したジョブのスケジューリング」を参照してください。

PL/SQL を使用した手動によるセグメント・アドバイザの実行 セグメント・アドバイザは、DBMS_ADVISOR パッケージを使用して実行することもできます。パッケージ・プロシージャを使用してセグメント・アドバイザのタスクを作成し、タスクの引数を設定してからタスクを実行します。ADVISOR 権限が付与されている必要があります。表 17-1 に、セグメント・アドバイザに関連するプロシージャを示します。これらのプロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

表 17-1 セグメント・アドバイザに関連する DBMS_ADVISOR パッケージ・プロシージャ

パッケージ・プロシージャ名	説明
CREATE_TASK	このプロシージャを使用して、セグメント・アドバイザのタスクを作成します。ADVISOR_NAME パラメータの値に、'Segment Advisor' を指定します。
CREATE_OBJECT	このプロシージャを使用して、セグメント領域のアドバイス対象オブジェクトを識別します。このプロシージャのパラメータ値は、オブジェクトの種類によって異なります。表 17-2 に、パラメータ値をオブジェクトの種類別に示します。 **注意：**IOT オーバーフロー・セグメントについてアドバイスを要求するには、TABLE、TABLE PARTITION または TABLE SUBPARTITION のオブジェクト型を使用します。次の問合せを使用して、IOT のオーバーフロー・セグメントを検索し、CREATE_OBJECT で使用するオーバーフロー・セグメントの表名を判断します。 `select table_name, iot_name, iot_type from dba_tables;`
SET_TASK_PARAMETER	このプロシージャを使用して、必要なセグメント・アドバイスを指定します。表 17-3 に、このプロシージャに関係する入力パラメータを示します。リストされていないパラメータは、セグメント・アドバイザでは使用されません。
EXECUTE_TASK	このプロシージャを使用して、セグメント・アドバイザのタスクを実行します。

表 17-2 DBMS_ADVISOR.CREATE_OBJECT の入力

入力パラメータ				
OBJECT_TYPE	**ATTR1**	**ATTR2**	**ATTR3**	**ATTR4**
TABLESPACE	*tablespace name*	NULL	NULL	未使用。NULL を指定します。
TABLE	*schema name*	*table name*	NULL	未使用。NULL を指定します。
INDEX	*schema name*	*index name*	NULL	未使用。NULL を指定します。
TABLE PARTITION	*schema name*	*table name*	*table partition name*	未使用。NULL を指定します。
INDEX PARTITION	*schema name*	*index name*	*index partition name*	未使用。NULL を指定します。
TABLE SUBPARTITION	*schema name*	*table name*	*table subpartition name*	未使用。NULL を指定します。
INDEX SUBPARTITION	*schema name*	*index name*	*index subpartition name*	未使用。NULL を指定します。
LOB	*schema name*	*segment name*	NULL	未使用。NULL を指定します。
LOB PARTITION	*schema name*	*segment name*	*lob partition name*	未使用。NULL を指定します。
LOB SUBPARTITION	*schema name*	*segment name*	*lob subpartition name*	未使用。NULL を指定します。

表 17-3 DBMS_ADVISOR.SET_TASK_PARAMETER の入力

入力パラメータ	説明	可能な値	デフォルト値
time_limit	セグメント・アドバイザを実行する時間制限を秒単位で指定します。	任意の秒数。	UNLIMITED
recommend_all	セグメント・アドバイザですべてのセグメントについて結果を生成するかどうかを指定します。	TRUE: 領域再生の推奨事項に関係なく、指定されたすべてのセグメントに対して結果が生成されます。 FALSE: 領域再生の推奨事項を生成するオブジェクトに対してのみ結果が生成されます。	TRUE

例 次の例は、DBMS_ADVISOR プロシージャを使用して、サンプル表 hr.employees に対してセグメント・アドバイザを実行する方法を示しています。これらのパッケージ・プロシージャを実行するユーザーには、そのパッケージに対するオブジェクトの EXECUTE 権限、または ADVISOR システム権限が必要です。

オブジェクト型 TABLE を DBMS_ADVISOR.CREATE_OBJECT に渡すと、オブジェクト・レベルを要求したことになります。表がパーティション化されていない場合は、表セグメントが分析されます（索引または LOB セグメントなどの依存セグメントは対象外です）。表がパーティション化されている場合は、すべての表パーティションが分析され、個別に結果と推奨事項が生成されます。

```
variable id number;
begin
  declare
  name varchar2(100);
  descr varchar2(500);
  obj_id number;
  begin
```

```
  name:='Manual_Employees';
  descr:='Segment Advisor Example';

  dbms_advisor.create_task (
    advisor_name     => 'Segment Advisor',
    task_id          => :id,
    task_name        => name,
    task_desc        => descr);

  dbms_advisor.create_object (
    task_name        => name,
    object_type      => 'TABLE',
    attr1            => 'HR',
    attr2            => 'EMPLOYEES',
    attr3            => NULL,
    attr4            => NULL,
    attr5            => NULL,
    object_id        => obj_id);

  dbms_advisor.set_task_parameter(
    task_name        => name,
    parameter        => 'recommend_all',
    value            => 'TRUE');

  dbms_advisor.execute_task(name);
  end;
end;
/
```

セグメント・アドバイザの結果の表示

セグメント・アドバイザでは、様々な種類の結果（推奨事項、結果、処置、オブジェクト）が作成されます。次の方法で結果を表示できます。

- Enterprise Manager の使用
- DBA_ADVISOR_* ビューの問合せ
- DBMS_SPACE.ASA_RECOMMENDATIONS プロシージャのコール

表 17-4 に、様々な種類の結果および関連する DBA_ADVISOR_* ビューを示します。

表 17-4 セグメント・アドバイザの結果の種類

結果の種類	関連するビュー	説明
推奨事項	DBA_ADVISOR_RECOMMENDATIONS	セグメントの縮小または再編成が効果的な場合は、そのセグメントに対して推奨事項が生成されます。表 17-5 に、生成される結果および推奨事項の例を示します。
結果	DBA_ADVISOR_FINDINGS	これは、分析したセグメントに関するレポートです。結果には、分析対象の各セグメントの使用済領域と空き領域の統計が含まれます。すべての結果が推奨事項になるわけではありません（推奨事項はわずかでも生成される結果は多い場合があります）。PL/SQL を使用してセグメント・アドバイザを手動で実行する場合は、SET_TASK_PARAMETER プロシージャの recommend_all に 'TRUE' を指定すると、そのセグメントに対する推奨事項が作成されるかどうかに関係なく、分析に適した各セグメントに対して結果が生成されます。行連鎖のアドバイスについては、自動セグメント・アドバイザでは結果のみが生成され、推奨事項は生成されません。推奨する領域再生がない場合、自動セグメント・アドバイザでは結果は生成されません。
処置	DBA_ADVISOR_ACTIONS	すべての推奨事項は、セグメントの縮小またはオンライン再定義（再編成）の実施を提案する処置に関連付けられます。DBA_ADVISOR_ACTIONS ビューには、セグメントの縮小を実行するために必要な SQL、またはオブジェクトを再編成するための提案が表示されます。
オブジェクト	DBA_ADVISOR_OBJECTS	結果、推奨事項および処置はすべて 1 つのオブジェクトに関連付けられます。セグメント・アドバイザが複数のセグメントを分析した場合は、表領域またはパーティション表と同様に、DBA_ADVISOR_OBJECTS ビューに、分析した各セグメントごとに 1 つのエントリが作成されます。表 17-2 は、このビューの列を示しています。これらの列を使用して、分析されたセグメントの情報を問い合せます。このビューのオブジェクトは、結果、推奨および処置の各ビューにあるオブジェクトに関連付けることができます。

関連項目：

- DBA_ADVISOR_* ビューの詳細は、『Oracle Database リファレンス』を参照してください。
- DBMS_SPACE.ASA_RECOMMENDATIONS ファンクションの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

Enterprise Manager を使用したセグメント・アドバイザの結果の表示

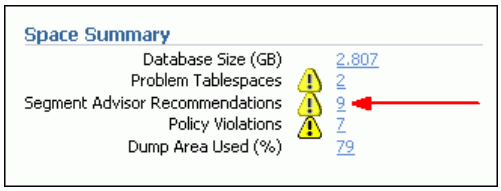
Enterprise Manager（EM）では、自動セグメント・アドバイザの実行とセグメント・アドバイザの手動実行の両方についてセグメント・アドバイザの結果を表示できます。次の種類の結果を表示できます。

- すべての推奨事項（自動および手動によるセグメント・アドバイザの複数の実行）
- 自動セグメント・アドバイザの最新の実行での推奨事項
- 特定の実行での推奨事項
- 行連鎖の結果

自動セグメント・アドバイザの最新の実行で分析されたセグメントを一覧に表示することもできます。

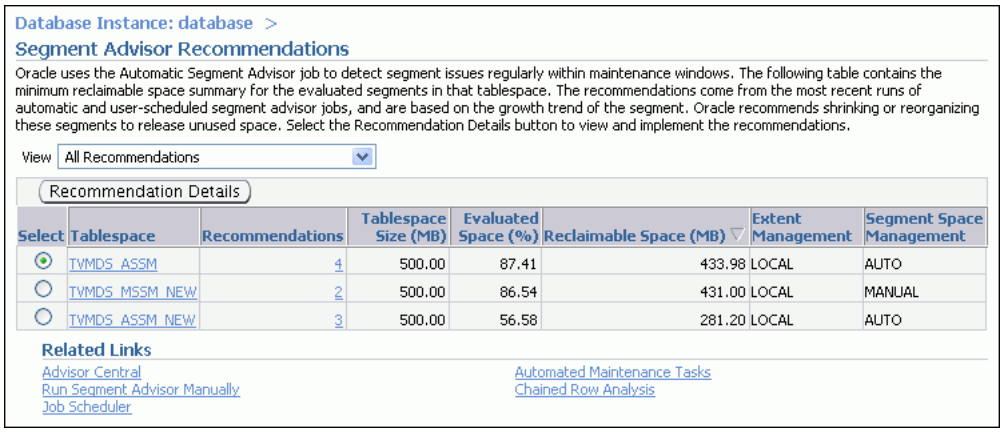
EM を使用してセグメント・アドバイザの結果を表示する手順（すべての実行）

1. データベース・ホームページで、「領域サマリー」ヘッダーの下にある**「セグメント・アドバイザ推奨」**タイトルの横の数値リンクをクリックします。



「セグメント・アドバイザ推奨」ページが表示されます。推奨事項は、表領域別に編成されます。

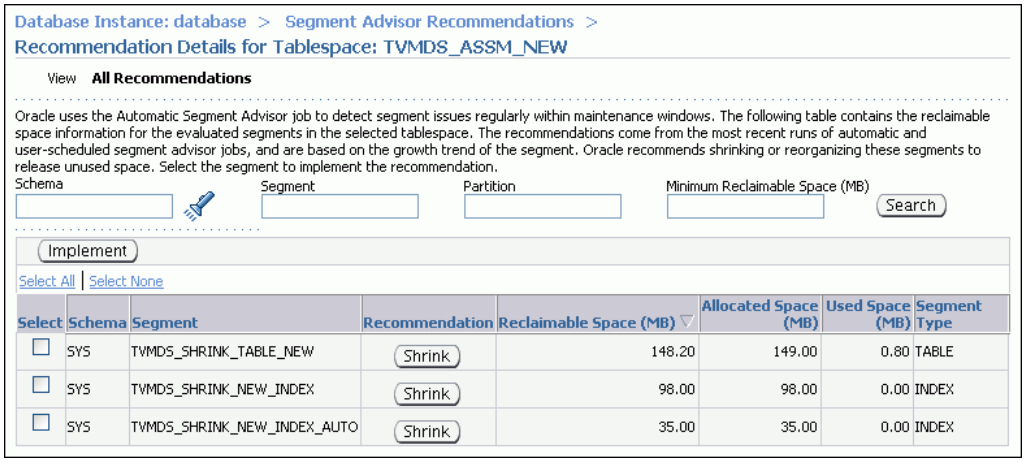
図 17-3「セグメント・アドバイザ推奨」ページ



2. 推奨事項が提示されている場合は、表領域を選択し、次に**「推奨事項の詳細」**をクリックします。

「推奨事項の詳細」ページが表示されます。このページから推奨事項アクティビティ（縮小または再編成）を開始できます。

図 17-4「推奨事項の詳細」ページ



ヒント： リストのエントリは、再生可能領域の大きい順にソートされています。列ヘッダーをクリックすると、ソート順を変更したり、昇順から降順に変更できます。

EM を使用してセグメント・アドバイザの結果を表示する手順（自動セグメント・アドバイザの最新の実行）

1. データベース・ホームページで、「領域サマリー」ヘッダーの下にある**「セグメント・アドバイザ推奨」**タイトルの横の数値リンクをクリックします。

   「セグメント・アドバイザ推奨」ページが表示されます。（図 17-3 を参照してください）。

2. 「表示」ドロップダウン・リストで、**「最後の自動実行からの推奨事項」**を選択します。

3. 推奨事項が提示されている場合は、**「選択」**列をクリックして表領域を選択し、次に**「推奨事項の詳細」**をクリックします。

   「推奨事項の詳細」ページが表示されます。（図 17-4 を参照してください）。このページから推奨事項アクティビティ（縮小または再編成）を開始できます。

EM を使用してセグメント・アドバイザの結果を表示する手順（特定の実行）

1. 「アドバイザ・セントラル」ページから開始します。

   Enterprise Manager ウィザードでセグメント・アドバイザを実行した場合は、セグメント・アドバイザのタスクを発行した後に「アドバイザ・セントラル」ページが表示されます。それ以外の場合は、データベース・ホームページの「関連リンク」の下にある**「アドバイザ・セントラル」**をクリックして、このページを表示します。

2. タスクが「結果」ヘッダーの下のリストに表示されていることを確認します。タスクがない場合は、手順を完了します（17-16 ページの図 17-2 を参照）。

   a. ページの「検索」セクションの「アドバイザ・タスク」の下にある「アドバイザ・タイプ」リストから**「セグメント・アドバイザ」**を選択します。

   b. 「アドバイザ実行」リストで、**「すべて」**または該当する期間を選択します。

   c. （オプション）タスク名を入力します。

   d. **「実行」**をクリックします。

      「結果」セクションにセグメント・アドバイザのタスクが表示されます。

3. ジョブのステータスを確認します。ステータスが COMPLETED でない場合は、タスク・ステータスに COMPLETED が表示されるまで、ページの上部にある**「リフレッシュ」**ボタンをクリックします（使用しているブラウザのリフレッシュ・アイコンは使用しないでください）。

4. タスク名をクリックします。

   セグメント・アドバイザのタスクのページに、表領域別に編成された推奨事項が表示されます。

5. リストで表領域を選択して、**「推奨事項の詳細」**をクリックします。

   「推奨事項の詳細」ページが表示されます。（図 17-4 を参照してください）。このページから推奨事項アクティビティ（縮小または再編成）を開始できます。

行連鎖の結果を表示する手順

1. データベース・ホームページで、「領域サマリー」ヘッダーの下にある**「セグメント・アドバイザ推奨」**タイトルの横の数値リンクをクリックします。

   「セグメント・アドバイザ推奨」ページが表示されます。（図 17-3 を参照してください）。

2. 「関連リンク」ヘッダーの下にある**「行チェーン分析」**をクリックします。

   「行チェーン分析」ページには、連鎖行があるすべてのセグメントと各連鎖行の割合が表示されます。

DBA_ADVISOR_* ビューの問合せによるセグメント・アドバイザの結果の表示

表 17-5 のヘッダーには、セグメント・アドバイザからの出力が記載された DBA_ADVISOR_* ビューの列が表示されます。これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。この表に、考えられる出力の要約を示します。また、17-17 ページの表 17-2 に、分析したセグメントの情報が格納される DBA_ADVISOR_OBJECTS ビューの列を示します。

DBA_ADVISOR_* ビューを問い合せる前に、DBA_ADVISOR_TASKS の STATUS 列を問い合せることで、セグメント・アドバイザのタスクが完了していることを確認できます。

```
select task_name, status from dba_advisor_tasks
   where owner = 'STEVE' and advisor_name = 'Segment Advisor';

TASK_NAME                      STATUS
------------------------------ -----------
Manual Employees               COMPLETED
```

次の例は、ユーザー STEVE が発行したセグメント・アドバイザのすべての実行から結果を取得するために、DBA_ADVISOR_* ビューを問い合せる方法を示しています。

```
select af.task_name, ao.attr2 segname, ao.attr3 partition, ao.type, af.message
  from dba_advisor_findings af, dba_advisor_objects ao
  where ao.task_id = af.task_id
  and ao.object_id = af.object_id
  and ao.owner = 'STEVE';

TASK_NAME          SEGNAME      PARTITION       TYPE            MESSAGE
------------------ ------------ --------------- --------------- -------------------------
Manual_Employees   EMPLOYEES                    TABLE           The free space in the obje
                                                                 ct is less than 10MB.

Manual_Salestable4 SALESTABLE4  SALESTABLE4_P1  TABLE PARTITION Perform shrink, estimated
                                                                 savings is 74444154 bytes.

Manual_Salestable4 SALESTABLE4  SALESTABLE4_P2  TABLE PARTITION The free space in the obje
                                                                 ct is less than 10MB.
```

表 17-5 セグメント・アドバイザの結果 : 要約

DBA_ADVISOR_FINDINGS の MESSAGE 列	DBA_ADVISOR_FINDINGS の MORE_INFO 列	DBA_ADVISOR_RECOMMENDATIONS の BENEFIT_TYPE 列	DBA_ADVISOR_ACTIONS の ATTR1 列
情報が不十分なため、推奨事項を作成できません。	なし	なし	なし
オブジェクト内の空き領域が 10MB 以下です。	割当領域 : *xxx*: 使用領域 : *xxx*: 再利用可能領域 : *xxx*	なし	なし
オブジェクト内には空き領域がありますが、... のため縮小できません。	割当領域 : *xxx*: 使用領域 : *xxx*: 再利用可能領域 : *xxx*	なし	なし
オブジェクト内の空き領域は前回のエクステントのサイズ未満です。	割当領域 : *xxx*: 使用領域 : *xxx*: 再利用可能領域 : *xxx*	なし	なし
xxx バイトの節約が予測されるため、縮小を実行してください。	割当領域 : *xxx*: 使用領域 : *xxx*: 再利用可能領域 : *xxx*	*xxx* バイトの節約が予測されるため、縮小を実行してください。	実行するコマンド (例 : ALTER *object* SHRINK SPACE;)

表 17-5 セグメント・アドバイザの結果 : 要約（続き）

DBA_ADVISOR_FINDINGS の MESSAGE 列	DBA_ADVISOR_FINDINGS の MORE_INFO 列	DBA_ADVISOR_ RECOMMENDATIONS の BENEFIT_TYPE 列	DBA_ADVISOR_ACTIONS の ATTR1 列
表 *schema.table* の行移動を有効にして縮小を実行してください。*xxx* バイトの節約が予測されるためです。	割当領域 : *xxx*: 使用領域 : *xxx*: 再利用可能領域 : *xxx*	表 *schema.table* の行移動を有効にして縮小を実行してください。*xxx* バイトの節約が予測されるためです。	実行するコマンド（例 : ALTER *object* SHRINK SPACE;）
オブジェクト *object* の再編成を実行してください。*xxx* バイトの節約が予測されるためです。 （注意 : これは、オンラインによるセグメントの縮小に適していない再生可能な領域を持つオブジェクトに対する結果です。）	割当領域 : *xxx*: 使用領域 : *xxx*: 再利用可能領域 : *xxx*	オブジェクト *object* の再編成を実行してください。*xxx* バイトの節約が予測されるためです。	再編成の実行
オブジェクトには、再編成によって削除できる連鎖行があります。	*xx* パーセントの連鎖行を再編成で削除できます。	なし	なし

DBMS_SPACE.ASA_RECOMMENDATIONS を使用したセグメント・アドバイザの結果の表示

DBMS_SPACE パッケージの ASA_RECOMMENDATIONS プロシージャは、ネストした表オブジェクトを戻します。この表オブジェクトには、自動セグメント・アドバイザの実行および手動によるセグメント・アドバイザの実行（オプション）に対する結果または推奨事項が格納されています。このプロシージャをコールすると、必要なすべての結合が実行され、使いやすい形式で情報が戻るため、DBA_ADVISOR_* ビューを使用するよりも簡単です。

次の問合せは、自動セグメント・アドバイザの最新の実行による推奨事項と、推奨事項に従うための実行コマンドの例を戻します。

```
select tablespace_name, segment_name, segment_type, partition_name,
recommendations, c1 from
table(dbms_space.asa_recommendations('FALSE', 'FALSE', 'FALSE'));


TABLESPACE_NAME                SEGMENT_NAME                   SEGMENT_TYPE
------------------------------ ------------------------------ --------------
PARTITION_NAME
------------------------------
RECOMMENDATIONS
--------------------------------------------------------------------------------
C1
--------------------------------------------------------------------------------
TVMDS_ASSM                     ORDERS1                        TABLE PARTITION
ORDERS1_P2
Perform shrink, estimated savings is 57666422 bytes.
alter table "STEVE"."ORDERS1" modify partition "ORDERS1_P2" shrink space

TVMDS_ASSM                     ORDERS1                        TABLE PARTITION
ORDERS1_P1
Perform shrink, estimated savings is 45083514 bytes.
alter table "STEVE"."ORDERS1" modify partition "ORDERS1_P1" shrink space

TVMDS_ASSM_NEW                 ORDERS_NEW                     TABLE

Perform shrink, estimated savings is 155398992 bytes.
alter table "STEVE"."ORDERS_NEW" shrink space

TVMDS_ASSM_NEW                 ORDERS_NEW_INDEX               INDEX
```

```
Perform shrink, estimated savings is 102759445 bytes.
alter index "STEVE"."ORDERS_NEW_INDEX" shrink space
```

DBMS_SPACE.ASA_RECOMMENDATIONS の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

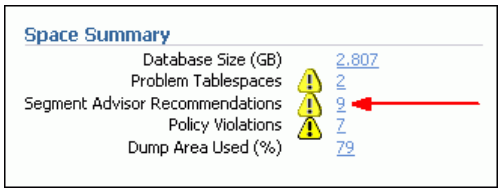
自動セグメント・アドバイザの構成

自動セグメント・アドバイザは自動化メンテナンス・タスクの 1 つです。そのため、このタスクの実行時に変更を加える場合は、Enterprise Manager または PL/SQL パッケージ・プロシージャ・コールを使用できます。適切なリソース・プランを変更することで、タスクに割り当てられているリソースを制御することもできます。

これらは PL/SQL パッケージ・プロシージャをコールして変更できますが、Enterprise Manager を使用するほうがさらに簡単です。

Enterprise Manager を使用して自動セグメント・アドバイザのタスクを構成する手順

1. ユーザー SYSTEM で Enterprise Manager にログインします。
2. データベース・ホームページで、「領域サマリー」ヘッダーの下にある**「セグメント・アドバイザ推奨」**ラベルの横の数値リンクをクリックします。



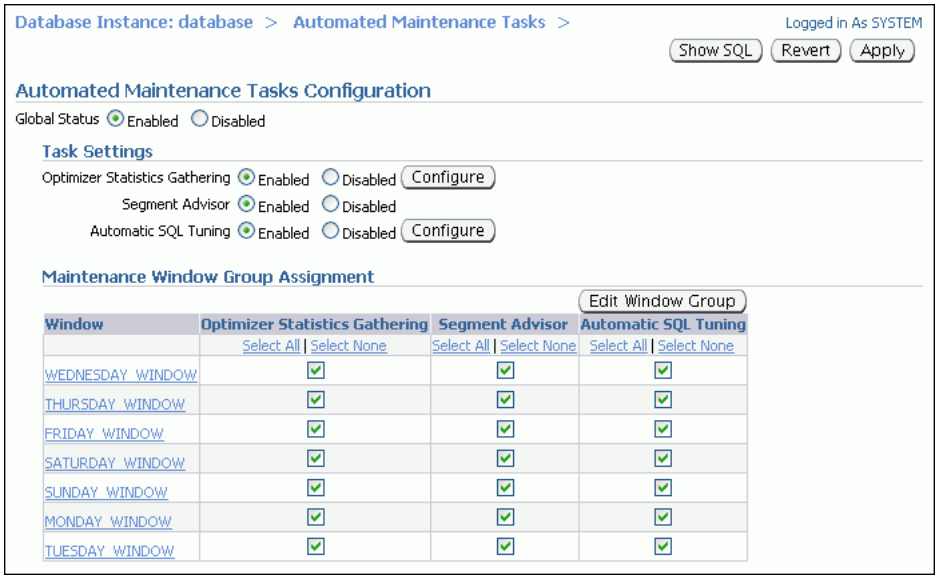
「セグメント・アドバイザ推奨」ページが表示されます。

3. 「関連リンク」ヘッダーの下にある**「自動化メンテナンス・タスク」**リンクをクリックします。

「自動化メンテナンス・タスク」ページが表示されます

4. **「構成」**をクリックします。

「自動化メンテナンス・タスク構成」ページが表示されます。



5. 自動セグメント・アドバイザを完全に無効化するには、「タスク設定」の下にある「セグメント・アドバイザ」ラベルの横の**「無効」**を選択して、**「適用」**をクリックします。

6. 特定のメンテナンス・ウィンドウの自動セグメント・アドバイザを無効化するには、「セグメント・アドバイザ」列の下の該当するチェック・ボックスの選択を解除し、**「適用」**をクリックします。
7. メンテナンス・ウィンドウの開始時間、終了時間および継続時間を変更するには、**「ウィンドウ・グループの編集」**をクリックします。

   「ウィンドウ・グループの編集」ページが表示されます。メンテナンス・ウィンドウの名前をクリックして**「編集」**をクリックし、ウィンドウのスケジュールを変更します。

関連項目：

- 第 24 章「自動データベース・メンテナンス・タスクの管理」

自動セグメント・アドバイザ情報の表示

次のビューには、自動セグメント・アドバイザ固有の情報が表示されます。詳細は、『Oracle Database リファレンス』を参照してください。

ビュー	説明
DBA_AUTO_SEGADV_SUMMARY	このビューには、各行に 1 件ずつ自動セグメント・アドバイザの実行が要約されます。フィールドには、処理された表領域やセグメントの数、および作成された推奨事項の件数が表示されます。
DBA_AUTO_SEGADV_CTL	自動セグメント・アドバイザがセグメントの選択および処理に使用する制御情報が表示されます。各行には、単一のオブジェクト（表領域またはセグメント）に関する情報（オブジェクトが処理されたかどうか、処理された場合はオブジェクトを処理したタスク ID やその選択理由など）が格納されます。

オンラインによるデータベース・セグメントの縮小

Oracle Database セグメントの最高水位標の下の断片化された空き領域を再生するには、オンラインによるセグメントの縮小を使用します。セグメントの縮小の利点は、次のとおりです。

- データの縮小はキャッシュ利用の向上につながります。つまり、オンライン・トランザクション処理（OLTP）のパフォーマンスが改善されます。
- データが縮小されることで、全表スキャンを実行する際に必要なスキャン・ブロック数が少なくなります。つまり、意思決定支援システム（DSS）のパフォーマンスが改善されます。

セグメントの縮小は、オンラインで適切に機能する操作です。セグメント縮小のデータ移動フェーズでは、DML 操作および問合せを発行できます。縮小操作の最後に領域が割当て解除される際は、短い時間ですが同時 DML 操作がブロックされます。索引は、縮小操作中も保持され、操作が完了した後も使用できます。セグメントの縮小では、余分なディスク領域の割当ては不要です。

セグメントの縮小では、最高水位標の上下両方の未使用領域を再生します。これに対して、領域の割当て解除では、最高水位標よりも上の未使用領域のみを再生します。縮小操作では、デフォルトの場合、セグメントを縮小して最高水位標を調整し、回復した領域が解放されます。

セグメントの縮小には、新しい位置への行の移動が必要です。したがって、最初に、縮小するオブジェクトの行を移動可能にし、オブジェクトに定義した ROWID ベースのトリガーを無効にする必要があります。表内の行を移動可能にするには、`ALTER TABLE ...ENABLE ROW MOVEMENT` コマンドを使用します。

縮小操作は、自動セグメント領域管理（ASSM）を使用しているローカル管理表領域内のセグメントに対してのみ実行できます。ASSM 表領域内では、次の表を除くすべてのセグメント・タイプがオンラインによるセグメントの縮小に適しています。

- IOT マッピング表
- ROWID ベースのマテリアライズド・ビューを備えた表

- ファンクション索引がある表
- SECUREFILE LOB

> **関連項目：** ALTER TABLE コマンドの詳細は、『Oracle Database SQL リファレンス』を参照してください。

オンラインによるセグメントの縮小の起動

オンラインによるセグメントの縮小を起動する前に、セグメント・アドバイザの結果および推奨事項を表示します。詳細は、17-13 ページの「セグメント・アドバイザの使用」を参照してください。

Enterprise Manager（EM）または SQL*Plus の SQL コマンドを使用して、オンラインによるセグメントの縮小を起動します。ここからは、コマンドラインによる方法について説明します。

> **注意：** セグメントの縮小は、EM の「推奨事項の詳細」ページから直接起動できます（図 17-4 を参照してください）。または、EM で個別の表に対してセグメントの縮小を起動するには、「表」ページに表を表示して該当する表を選択し、「アクション」リストで**「セグメントの縮小」**をクリックします（図 17-1 を参照してください）。索引、マテリアライズド・ビューなどを縮小するには、EM で同様の操作を実行します。

表、索引構成表、索引、パーティション、サブパーティション、マテリアライズド・ビューまたはマテリアライズド・ビュー・ログの領域を縮小できます。この縮小には、SHRINK SPACE 句を指定した ALTER TABLE、ALTER INDEX、ALTER MATERIALIZED VIEW 文または ALTER MATERIALIZED VIEW LOG 文を使用します。データベース・オブジェクトを縮小する構文、追加情報および制約は、『Oracle Database SQL リファレンス』を参照してください。

次の 2 つの句を必要に応じて使用することで、縮小操作の処理方法を制御できます。

- COMPACT 句を指定すると、セグメントの縮小操作を 2 つのフェーズに分割できます。COMPACT を指定すると、Oracle Database はセグメント領域の断片化を解消して表の行を縮小しますが、将来のある時点まで最高水位標のリセットおよび領域の割当て解除を延期します。このオプションは、操作に影響される可能性がある長時間実行される問合せがあり、再生されたブロックから読取りが試行される場合に役立ちます。断片化解消および縮小の結果は、第 2 のフェーズでデータの移動をやり直す必要がないように、ディスクに保存されます。第 2 のフェーズを完了するには、オフピーク時に COMPACT 句を指定せずに SHRINK SPACE 句を再度発行します。
- CASCADE 句は、セグメントの縮小操作の範囲をオブジェクトのすべての依存セグメントまで拡張します。たとえば、表のセグメントを縮小する場合に CASCADE を指定すると、表のすべての索引も縮小されます（パーティション表のパーティションの縮小に CASCADE を指定する必要はありません）。指定されたオブジェクトの依存セグメントを一覧表示するには、DBMS_SPACE パッケージの OBJECT_DEPENDENT_SEGMENTS プロシージャを実行します。

DDL 操作と同様に、セグメントの縮小によって、後続の SQL 文が再度解析されることになります。これは、COMPACT 句を指定しないかぎり、カーソルが無効になるためです。

例

表およびその依存セグメント（BASICFILE LOB セグメントを含む）のすべてを縮小します。

```
ALTER TABLE employees SHRINK SPACE CASCADE;
```

BASICFILE LOB セグメントのみを縮小します。

```
ALTER TABLE employees MODIFY LOB (perf_review) (SHRINK SPACE);
```

パーティション表の単一パーティションを縮小します。

```
ALTER TABLE customers MODIFY PARTITION cust_P1 SHRINK SPACE;
```

IOT 索引セグメントとオーバーフロー・セグメントを縮小します。

```
ALTER TABLE cities SHRINK SPACE CASCADE;
```

IOT オーバーフロー・セグメントのみを縮小します。

```
ALTER TABLE cities OVERFLOW SHRINK SPACE;
```

関連項目： LOB セグメントの詳細は、『Oracle Database SecureFiles およびラージ・オブジェクト開発者ガイド』を参照してください。

未使用領域の割当て解除

未使用領域の割当てを解除する場合は、未使用の（最高水位標）データベース・セグメントの最後で未使用領域を解放して、表領域の他のセグメントで領域を使用できるようにします。

割当てを解除する前に、DBMS_SPACE パッケージの UNUSED_SPACE プロシージャを実行できます。このプロシージャは、最高水位標の位置とセグメント内の未使用領域の量についての情報を返します。自動セグメント領域管理を使用しているローカル管理表領域のセグメントでは、SPACE_USAGE プロシージャを使用すると、未使用領域に関する正確な情報を取得できます。

関連項目： DBMS_SPACE パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

次の文は、セグメント（表、索引またはクラスタ）内の未使用領域の割当てを解除します。

```
ALTER TABLE table DEALLOCATE UNUSED KEEP integer;
ALTER INDEX index DEALLOCATE UNUSED KEEP integer;
ALTER CLUSTER cluster DEALLOCATE UNUSED KEEP integer;
```

KEEP 句はオプションです。セグメント内に保持する領域の大きさを指定できます。DBA_FREE_SPACE ビューを調べることにより、割当て解除によって解放された領域を確認できます。

関連項目：

- 未使用領域の割当て解除に関連する構文とセマンティクスの詳細は、『Oracle Database SQL リファレンス』を参照してください。
- DBA_FREE_SPACE ビューの詳細は、『Oracle Database リファレンス』を参照してください。

データ型の領域使用の理解

表またはその他のデータ構造を作成する際には、必要となる領域の大きさを把握しておく必要があります。領域要件は、データ型ごとに異なります。データ型とその領域要件の詳細は、『Oracle Database PL/SQL 言語リファレンス』および『Oracle Database SQL リファレンス』を参照してください。

スキーマ・オブジェクトの領域使用情報の表示

Oracle Database には、スキーマ・オブジェクトの領域使用に関する情報を表示するためのデータ・ディクショナリ・ビューと PL/SQL パッケージが用意されています。特定のスキーマ・オブジェクトに固有のビューとパッケージは、このマニュアルの各オブジェクトに関連した章に記載されています。ここでは、汎用的な性質を持ち、複数のスキーマ・オブジェクトに適用されるビューとパッケージについて説明します。

PL/SQL パッケージを使用したスキーマ・オブジェクトの領域使用情報の表示

オラクル社が提供する PL/SQL パッケージを使用すると、スキーマ・オブジェクトに関する次の情報を取得できます。

パッケージとプロシージャ / ファンクション	説明
DBMS_SPACE.UNUSED_SPACE	オブジェクト（表、索引またはクラスタ）の未使用領域に関する情報を返します。
DBMS_SPACE.FREE_BLOCKS	セグメントの空き領域が空きリストで管理されている（つまり、セグメント領域管理が MANUAL である）オブジェクト（表、索引またはクラスタ）の空きデータ・ブロックに関する情報を返します。
DBMS_SPACE.SPACE_USAGE	セグメント領域管理が AUTO であるオブジェクト（表、索引またはクラスタ）の空きデータ・ブロックに関する情報を返します。

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

例 : DBMS_SPACE.UNUSED_SPACE の使用

次の SQL*Plus の例では、DBMS_SPACE パッケージを使用して、未使用領域情報を取得しています。

```
SQL> VARIABLE total_blocks NUMBER
SQL> VARIABLE total_bytes NUMBER
SQL> VARIABLE unused_blocks NUMBER
SQL> VARIABLE unused_bytes NUMBER
SQL> VARIABLE lastextf NUMBER
SQL> VARIABLE last_extb NUMBER
SQL> VARIABLE lastusedblock NUMBER
SQL> exec DBMS_SPACE.UNUSED_SPACE('SCOTT', 'EMP', 'TABLE', :total_blocks, -
>    :total_bytes,:unused_blocks, :unused_bytes, :lastextf, -
>    :last_extb, :lastusedblock);

PL/SQL procedure successfully completed.

SQL> PRINT

TOTAL_BLOCKS
------------
           5

TOTAL_BYTES
-----------
      10240

...

LASTUSEDBLOCK
-------------
            3
```

スキーマ・オブジェクトの領域使用のデータ・ディクショナリ・ビュー

次のビューには、スキーマ・オブジェクトの領域使用に関する情報が表示されます。

ビュー	説明
DBA_SEGMENTS USER_SEGMENTS	DBA ビューには、すべてのデータベース・セグメントに割り当てられている記憶域が表示されます。ユーザー・ビューには、現行のユーザーのセグメントに割り当てられている記憶域が表示されます。
DBA_EXTENTS USER_EXTENTS	DBA ビューには、データベース内のすべてのセグメントを構成するエクステントが表示されます。ユーザー・ビューには、現行のユーザーのセグメントを構成するエクステントが表示されます。
DBA_FREE_SPACE USER_FREE_SPACE	DBA ビューには、すべての表領域の使用可能エクステントが表示されます。ユーザー・ビューには、ユーザーが割当て制限を持つ表領域の空き領域情報が表示されます。

次に、これらのビューの使用例を示します。

関連項目： データ・ディクショナリ・ビューの詳細は、『Oracle Database リファレンス』を参照してください。

例 1: セグメント情報の表示

次の問合せは、スキーマ hr の各索引セグメントの名前とサイズを返します。

```
SELECT SEGMENT_NAME, TABLESPACE_NAME, BYTES, BLOCKS, EXTENTS
    FROM DBA_SEGMENTS
    WHERE SEGMENT_TYPE = 'INDEX'
    AND OWNER='HR'
    ORDER BY SEGMENT_NAME;
```

問合せ出力は、次のとおりです。

```
SEGMENT_NAME              TABLESPACE_NAME    BYTES BLOCKS EXTENTS
------------------------- --------------- -------- ------ -------
COUNTRY_C_ID_PK           EXAMPLE            65536     32       1
DEPT_ID_PK                EXAMPLE            65536     32       1
DEPT_LOCATION_IX          EXAMPLE            65536     32       1
EMP_DEPARTMENT_IX         EXAMPLE            65536     32       1
EMP_EMAIL_UK              EXAMPLE            65536     32       1
EMP_EMP_ID_PK             EXAMPLE            65536     32       1
EMP_JOB_IX                EXAMPLE            65536     32       1
EMP_MANAGER_IX            EXAMPLE            65536     32       1
EMP_NAME_IX               EXAMPLE            65536     32       1
JHIST_DEPARTMENT_IX       EXAMPLE            65536     32       1
JHIST_EMPLOYEE_IX         EXAMPLE            65536     32       1
JHIST_EMP_ID_ST_DATE_PK   EXAMPLE            65536     32       1
JHIST_JOB_IX              EXAMPLE            65536     32       1
JOB_ID_PK                 EXAMPLE            65536     32       1
LOC_CITY_IX               EXAMPLE            65536     32       1
LOC_COUNTRY_IX            EXAMPLE            65536     32       1
LOC_ID_PK                 EXAMPLE            65536     32       1
LOC_STATE_PROVINCE_IX     EXAMPLE            65536     32       1
REG_ID_PK                 EXAMPLE            65536     32       1

19 rows selected.
```

例 2: エクステント情報の表示

データベース内に現在割り当てられているエクステントに関する情報は、DBA_EXTENTS データ・ディクショナリ・ビューに格納されています。たとえば、次の問合せによって、hr スキーマの各索引セグメントに割り当てられているエクステントとそれらのエクステントのサイズが識別されます。

```
SELECT SEGMENT_NAME, SEGMENT_TYPE, TABLESPACE_NAME, EXTENT_ID, BYTES, BLOCKS
    FROM DBA_EXTENTS
    WHERE SEGMENT_TYPE = 'INDEX'
    AND OWNER='HR'
    ORDER BY SEGMENT_NAME;
```

問合せ出力は、次のとおりです。

```
SEGMENT_NAME              SEGMENT_TYPE TABLESPACE_NAME EXTENT_ID    BYTES BLOCKS
------------------------- ------------ --------------- --------- -------- ------
COUNTRY_C_ID_PK           INDEX        EXAMPLE                 0    65536     32
DEPT_ID_PK                INDEX        EXAMPLE                 0    65536     32
DEPT_LOCATION_IX          INDEX        EXAMPLE                 0    65536     32
EMP_DEPARTMENT_IX         INDEX        EXAMPLE                 0    65536     32
EMP_EMAIL_UK              INDEX        EXAMPLE                 0    65536     32
EMP_EMP_ID_PK             INDEX        EXAMPLE                 0    65536     32
EMP_JOB_IX                INDEX        EXAMPLE                 0    65536     32
EMP_MANAGER_IX            INDEX        EXAMPLE                 0    65536     32
EMP_NAME_IX               INDEX        EXAMPLE                 0    65536     32
JHIST_DEPARTMENT_IX       INDEX        EXAMPLE                 0    65536     32
JHIST_EMPLOYEE_IX         INDEX        EXAMPLE                 0    65536     32
JHIST_EMP_ID_ST_DATE_PK   INDEX        EXAMPLE                 0    65536     32
JHIST_JOB_IX              INDEX        EXAMPLE                 0    65536     32
JOB_ID_PK                 INDEX        EXAMPLE                 0    65536     32
LOC_CITY_IX               INDEX        EXAMPLE                 0    65536     32
LOC_COUNTRY_IX            INDEX        EXAMPLE                 0    65536     32
LOC_ID_PK                 INDEX        EXAMPLE                 0    65536     32
LOC_STATE_PROVINCE_IX     INDEX        EXAMPLE                 0    65536     32
REG_ID_PK                 INDEX        EXAMPLE                 0    65536     32

19 rows selected.
```

hr スキーマには、複数のエクステントが割り当てられているセグメントはありません。

例 3: 表領域内の空き領域（エクステント）の表示

データベース内の使用可能エクステント（どのセグメントにも割り当てられていないエクステント）に関する情報は、DBA_FREE_SPACE データ・ディクショナリ・ビューに格納されています。たとえば、次の問合せは、SMUNDO 表領域内の使用可能エクステントとして使用可能な空き領域を示します。

```
SELECT TABLESPACE_NAME, FILE_ID, BYTES, BLOCKS
    FROM DBA_FREE_SPACE
    WHERE TABLESPACE_NAME='SMUNDO';
```

問合せ出力は、次のとおりです。

```
TABLESPACE_NAME  FILE_ID    BYTES BLOCKS
--------------- -------- -------- ------
SMUNDO                 3    65536     32
SMUNDO                 3    65536     32
SMUNDO                 3    65536     32
SMUNDO                 3    65536     32
SMUNDO                 3    65536     32
SMUNDO                 3    65536     32
SMUNDO                 3   131072     64
SMUNDO                 3   131072     64
```

```
SMUNDO                  3    65536     32
SMUNDO                  3  3407872   1664

10 rows selected.
```

例 4: 追加のエクステントを割り当てることができないセグメントの表示

セグメントは、次のいずれかの理由でエクステントを割り当てることができない場合があります。

- セグメントを含む表領域に次のエクステントのための十分な領域がない場合
- セグメント内のエクステント数が最大エクステント数に到達した場合
- セグメント内のエクステント数が、データ・ブロック・サイズ（オペレーティング・システム固有）によって許可される最大のエクステント数に到達した場合

次の問合せは、これらの基準のいずれかを満たすすべてのセグメントの名前、所有者および表領域を返します。

```
SELECT a.SEGMENT_NAME, a.SEGMENT_TYPE, a.TABLESPACE_NAME, a.OWNER
    FROM DBA_SEGMENTS a
    WHERE a.NEXT_EXTENT >= (SELECT MAX(b.BYTES)
        FROM DBA_FREE_SPACE b
        WHERE b.TABLESPACE_NAME = a.TABLESPACE_NAME)
    OR a.EXTENTS = a.MAX_EXTENTS
    OR a.EXTENTS = 'data_block_size' ;
```

注意： この問合せを使用するときは、*data_block_size* をシステムのデータ・ブロック・サイズに置き換えてください。

追加のエクステントを割り当てることのできないセグメントを識別した後、その原因に応じて、次のどちらかの方法で問題を解決できます。

- 表領域がいっぱいの場合は、表領域にデータファイルを追加するか、既存のデータファイルを拡張します。
- セグメント内に存在するエクステントが多すぎるものの、セグメントの MAXEXTENTS を増やせない場合は、次の手順を実行します。

  1. セグメント内のデータをエクスポートします。
  2. セグメントを削除して、再作成します。その際、多数のエクステントを割り当てる必要がないように、INITIAL 記憶域パラメータに大きい値を設定します。または、PCTINCREASE および NEXT 記憶域パラメータを調整して、セグメント内により多くの領域を確保できます。
  3. セグメントにデータをインポートします。

データベース・オブジェクトの容量計画

Oracle Database には、データベース・オブジェクトの容量を計画する方法が 2 つあります。

- Enterprise Manager の使用
- DBMS_SPACE PL/SQL パッケージの使用

ここでは、PL/SQL による方法について説明します。Enterprise Manager を使用した容量計画の詳細は、Enterprise Manager のオンライン・ヘルプおよび『Oracle Database 2 日でデータベース管理者』を参照してください。

DBMS_SPACE パッケージには、新しいオブジェクトのサイズを予測したり、既存のデータベース・オブジェクトのサイズを監視できる 3 つのプロシージャがあります。ここでは、これらのプロシージャについて説明します。この項の内容は、次のとおりです。

表の領域使用の見積り

データベース表のサイズは、表領域の記憶域属性、表領域のブロック・サイズ、他の様々な要因によって大きく変化する可能性があります。DBMS_SPACE パッケージの CREATE_TABLE_COST プロシージャを使用すると、表を作成する際の領域使用コストを予測できます。このプロシージャのパラメータの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

プロシージャには、2 つの異形があります。第 1 の異形は、行の平均サイズを使用してサイズを見積ります。第 2 の異形は、列情報を使用して表のサイズを見積ります。両方の異形ともに入力として次の値が必要です。

- TABLESPACE_NAME: オブジェクトが作成される表領域。デフォルトは SYSTEM 表領域です。
- ROW_COUNT: 表の予測行数。
- PCT_FREE: 更新の結果、既存の行を将来拡張するように各ブロックに確保する空き領域の割合（パーセンテージ）。

さらに、第 1 の異形には、AVG_ROW_SIZE の値も入力する必要があります。値は、予測された行サイズの平均をバイト単位で指定します。

また、第 2 の異形には、予想された各列値を COLINFOS に指定する必要があります。この値は、属性 COL_TYPE（列のデータ型）と COL_SIZE（列の文字数またはバイト数）からなるオブジェクト型です。

このプロシージャでは、2 つの値が戻ります。

- USED_BYTES: ブロック・メタデータ、PCT_FREE 領域などのオーバーヘッドを含め、データとして使用される実際のバイト数。
- ALLOC_BYTES: 表領域のエクステントの特性を考慮してオブジェクトに割り当てられる予測した領域の大きさ。

索引の領域使用の見積り

DBMS_SPACE パッケージの CREATE_INDEX_COST プロシージャを使用すると、既存の表に索引を作成する際の領域使用コストを予測できます。

プロシージャには次の値を入力する必要があります。

- DDL: 索引を作成する CREATE INDEX 文。この DDL 文に指定する表は、既存の表であることが必要です。
- PLAN_TABLE（オプション）: 使用する PLAN TABLE の名前。デフォルトは NULL です。

このプロシージャから戻る結果は、セグメントに対して収集された統計によって異なります。したがって、このプロシージャを実行する直前に必ず統計を取得してください。最近の統計がない状態でもエラーになりませんが、不適切な結果が戻る可能性があります。このプロシージャでは、次の値が戻ります。

- USED_BYTES: 実際の索引データを表すバイト数。
- ALLOC_BYTES: 表領域の索引に割り当てられる領域の大きさ。

オブジェクト増加傾向の取得

DBMS_SPACE パッケージの OBJECT_GROWTH_TREND プロシージャを使用すると、1 行以上の表が作成されます。各行には、特定の時点でのオブジェクトの領域使用が表示されます。このプロシージャは、自動ワークロード・リポジトリから領域使用合計を取得、または現在の領域使用を計算し、その値を自動ワークロード・リポジトリから取得したこれまでの領域使用の変化と結び付けます。このプロシージャのパラメータの詳細は、『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。

プロシージャには次の値を入力する必要があります。

- OBJECT_OWNER: オブジェクトの所有者。
- OBJECT_NAME: オブジェクトの名前。
- PARTITION_NAME: 適切な場合は、表または索引パーティションの名前を指定します。それ以外の場合は、NULL を指定します。
- OBJECT_TYPE: オブジェクトの種類。
- START_TIME: 増加傾向分析の開始を示す TIMESTAMP 値。
- END_TIME: 増加傾向分析の終了を示す TIMESTAMP 値。デフォルトは NOW です。
- INTERVAL: プロシージャが領域使用情報を取得するレポート間隔の長さ（分単位）。
- SKIP_INTERPOLATED: INTERVAL 前後の記録統計に基づく値を省略する（'YES'）か、省略しない（'NO'）かを指定します。この設定は、結果表がチャートではなく表で示される場合に役立ちます。表形式で表示すると、実際の記録間隔が要求されたレポート間隔にどのように関係しているかを明瞭に理解できます。

このプロシージャでは表が戻ります。表の各行には 1 間隔ごとにオブジェクトの領域使用情報が表示されます。戻される表が非常に大きい場合は、別のアプリケーションで情報を使用できるように、結果が作成される際にパイプライン化されます。出力される表には、次の列があります。

- TIMEPOINT: レポート間隔の時間を示す TIMESTAMP 値。

  最も古いオブジェクトの記録統計よりも前の TIME 値の記録は作成されません。
- SPACE_USAGE: オブジェクト・データとして実際に使用されているバイト数。
- SPACE_ALLOC: その時点で表領域のオブジェクトに割り当てられていたバイト数。
- QUALITY: 要求されたレポート間隔と実際の統計記録が一致している程度を示す値。オブジェクト・サイズ使用統計には保証されているレポート間隔はなく、実際のレポート間隔は時間の経過およびオブジェクトによって変わるため、この情報が役立ちます。

QUALITY 列には、次の値が設定されます。

- GOOD: 記録されたタイムスタンプが入力パラメータに指定した INTERVAL の 10% 以内で、TIME の値が記録統計に常に基づいている場合。
- INTERPOLATED: 値は GOOD の基準を満たしていないが、TIME 値の前後の記録統計に基づいている場合。現在のメモリー内統計は、クラスタ内のすべてのインスタンスで収集でき、現時点の記録値として取り扱うことができます。
- PROJECTION: 表が作成された時点では TIME 値が未来であった場合。Oracle Real Application Clusters 環境では、統計の記録ルールによって、どのオブジェクトが選択されるかを各インスタンスが独自に選択できるようになっています。

このプロシージャから戻る出力は、RAC 環境のすべてのインスタンスについて記録した値の集合です。各値は、GOOD および INTERPOLATED 値の組合せから計算できます。その値の少なくとも 80% が GOOD のインスタンス値から導出された場合、戻される集計値は GOOD にマークされます。

18

表の管理

この章の内容は次のとおりです。

表の概要

表は、Oracle Database のデータ記憶域の基本単位です。データは、行および列に格納されます。表は、employees などの表名と一連の列で定義します。各列には列名（employee_id、last_name、job_id など）、データ型（VARCHAR2、DATE、NUMBER など）および幅を指定します。幅は、データ型（DATE など）によって事前に決まる場合があります。NUMBER データ型の列の場合は、幅ではなく、精度および位取りを定義します。行は、単一のレコードに対応する列情報の集合です。

表の各列にはルールを指定できます。これらのルールは整合性制約と呼ばれています。NOT NULL は、整合性制約の 1 つです。これは、各行の列に値を指定することを強制する制約です。

透過的データ暗号化を起動して、データを暗号化してから格納できます。ユーザーが、オペレーティング・システムのツールを使用して Oracle データファイルの内容を直接参照することによって、データベース・アクセス制御メカニズムを迂回しようとした場合でも、暗号化によって、このようなユーザーが機密データを参照できないようにします。

表には仮想列を含めることもできます。**仮想列**は表の他の列とほぼ同じですが、値が式を評価して導出される点が異なります。式に使用できるのは、同じ表の列、定数、SQL ファンクションおよびユーザー定義の PL/SQL ファンクションです。仮想列に明示的に書き込むことはできません。

列の型には、LOB、VARRAY およびネストした表のように専用セグメントに格納されるものがあります。LOB と VARRAY は LOB セグメントに格納されますが、ネストした表は記憶表に格納されます。これらのセグメントに対して STORAGE 句を指定し、表レベルで指定した記憶域パラメータを上書きできます。

表を作成した後は、SQL 文または Oracle のバルク・ロード・ユーティリティを使用してデータ行を挿入します。表データは、SQL を使用して問合せ、削除または更新できます。

関連項目：

- 表の概要については、『Oracle Database 概要』を参照してください。
- Oracle Database のデータ型の説明は、『Oracle Database SQL リファレンス』を参照してください。
- 表の領域を管理するためのガイドラインは、第 17 章「スキーマ・オブジェクトの領域の管理」を参照してください。
- 整合性制約の指定や表の分析など、表の管理に関するその他の情報については、第 16 章「スキーマ・オブジェクトの管理」を参照してください。
- 透過的データ暗号化については、『Oracle Database Advanced Security 管理者ガイド』を参照してください。

表を管理するためのガイドライン

ここでは、表を管理するときに従うべきガイドラインについて説明します。これらのガイドラインに従うことで、表の作成や表データのロード、更新および問合せを行うときに、表の管理が容易になり、パフォーマンスの向上にもつながります。

この項の内容は、次のとおりです。

作成前の表の設計

通常、アプリケーション開発者は、表などのアプリケーションの要素を設計する必要があります。データベース管理者は、アプリケーション表を保持する、基礎となる表領域に対する属性の設定を担当します。DBA またはアプリケーション開発者は（あるいは双方が協力して）、サイトの業務に基づいて実際の表の作成を担当します。

表を設計する場合は、アプリケーション開発者と協力し、次のガイドラインを考慮してください。

- 表、列、索引およびクラスタには、内容を表現する名前を使用します。
- 表名および列に略語を使用する場合や、単数形と複数形の使用には、一貫性を持たせます。
- COMMENT コマンドを使用して、各表とその列の意味を記載します。
- 各表は正規化します。
- 各列に対して適切なデータ型を選択します。
- 一部の表に仮想列を 1 つ以上追加した場合、アプリケーションに利点があるかどうかを検討します。
- 記憶域を節約するために、NULL を許可する列を最後に定義します。
- 記憶域を節約し、SQL 文のパフォーマンスを最適化するために、適切な場合は必ず表をクラスタ化します。

表を作成する前に、整合性制約の使用についても判断します。表の列に整合性制約を定義することによって、データベースのビジネス・ルールを自動的に徹底できます。

作成する表のタイプに関するオプションの考慮

作成する表のタイプを決定します。次のタイプがあります。

表のタイプ	説明
通常の（ヒープ構成）表	この章の主な説明の対象でもある基本的で多目的な表です。この表のデータは、順序付けされていないコレクション（ヒープ）として格納されます。
クラスタ化表	クラスタ化表は、クラスタの一部となっている表です。クラスタとは、同じデータ・ブロックを共有する表のグループです。グループ化されるのは、これらの表が共通の列を共有し、多くの場合まとめて使用されるためです。 クラスタとクラスタ化表については、第 20 章「クラスタの管理」を参照してください。
索引構成表	通常の（ヒープ構成）表とは異なり、索引構成表のデータは B ツリーの索引構造に主キー・ソート方式で格納されます。B ツリーの各索引エントリには、索引構成表の行の主キー列値以外に、非キー列値も格納されます。 索引構成表については、18-48 ページの「索引構成表の管理」を参照してください。
パーティション表	パーティション表では、データをパーティションと呼ばれる管理が容易な単位に分割し、さらにそれをサブパーティションに分割できます。各パーティションは個々に管理でき、他のパーティションとは無関係に操作できます。これによって、可用性やパフォーマンスを考慮して適切にチューニング可能な構造を用意できます。 パーティション表については、『Oracle Database VLDB およびパーティショニング・ガイド』を参照してください。

各表の位置の指定

新しい表を格納する表領域を識別するには、CREATE TABLE 文に TABLESPACE 句を指定します。使用する表領域に対する適切なシステム権限と割当て権限があることを確認してください。CREATE TABLE 文で表領域を指定しない場合は、作成したユーザーのデフォルト表領域内に表が作成されます。

新しい表を含む表領域を指定するときは、その選択が意味することを確実に理解しておいてください。各表の作成時に表領域を適切に指定することによって、データベース・システムのパフォーマンスが向上し、データベース管理に必要な時間を短縮できます。

次のように、表領域を指定しない場合や不適切な表領域を指定した場合は、パフォーマンスに影響を与えます。

- ユーザーのオブジェクトを SYSTEM 表領域に作成すると、データ・ディクショナリ・オブジェクトとユーザー・オブジェクトの両方が同じデータファイルを求めて競合し、データベースのパフォーマンスが低下するおそれがあります。ユーザーのオブジェクトは SYSTEM 表領域に格納しないでください。これを回避するには、データベースに表領域が作成される際に、すべてのユーザーにデフォルトの表領域が割り当てられていることを確認します。
- アプリケーションに関係する表をいろいろな表領域に無計画に格納すると、DBA がアプリケーションのデータ管理操作（バックアップやリカバリなど）に要する時間が増大する可能性があります。

表作成のパラレル化

CREATE TABLE 文で副問合せ（AS SELECT）を使用して表を作成する際は、パラレル実行を利用できます。複数のプロセスが同時に動作して表を作成するため、表を作成するときのパフォーマンスが向上します。

表作成のパラレル化については、18-11 ページの「表作成のパラレル化」を参照してください。

表作成時の NOLOGGING の使用

表を最も効率よく作成するには、CREATE TABLE...AS SELECT 文で NOLOGGING 句を使用します。NOLOGGING 句を指定すると、表の作成中に最小限の REDO 情報しか生成されません。これには、次のような利点があります。

- REDO ログ・ファイルの領域を節約できます。
- 表の作成に要する時間が削減できます。
- 大規模な表のパラレル作成のパフォーマンスが向上します。

また、NOLOGGING 句を指定することで、SQL*Loader を使用した後続のダイレクト・ロードおよびダイレクト・ロード INSERT 操作がロギングされなくなります。後続のデータ操作文（DML）文（UPDATE、DELETE および従来型パスの挿入）は、表の NOLOGGING 属性の影響を受けず、REDO を生成します。

表の作成後にその表の損失（たとえば、表の作成に使用したデータにアクセスできなくなるなど）を避ける必要がある場合は、作成直後に表のバックアップを取得してください。一時的に使用するために作成する表など、そのような予防策が不要な場合もあります。

一般に、NOLOGGING を指定して表を作成するときは、小規模な表より大規模な表のほうが相対的にパフォーマンスの向上が大きくなります。小規模な表の場合は、NOLOGGING を指定しても、表作成に要する時間にほとんど影響はありません。一方、大規模な表では、特に表作成をパラレル化したときにパフォーマンスが著しく向上します。

表圧縮の使用

データベースが GB または TB 以上に大きくなる場合は、表圧縮の使用を検討してください。表圧縮を使用すると、ディスク領域が節約され、バッファ・キャッシュのメモリー使用が削減されます。また、読込み中の問合せ実行速度も向上します。ただし、データのロードや DML については CPU オーバーヘッドがかかります。表圧縮はアプリケーションに対して完全に透過的です。長い読取り専用操作が実行されるオンライン分析処理（OLAP）システムで特に有効ですが、オンライン・トランザクション処理（OLTP）システムでも使用できます。

表圧縮の指定には、CREATE TABLE 文の COMPRESS 句を使用します。既存の表に対して圧縮を使用可能にするには、この句を ALTER TABLE 文で使用します。この場合、圧縮されるデータは、圧縮を使用可能にした後で挿入または更新されたデータのみです。同様に、ALTER TABLE...NOCOMPRESS 文を使用すると、既存の圧縮表に対する表圧縮を使用禁止にできます。この場合、圧縮済のデータはすべて圧縮されたままになり、新規データは圧縮されずに挿入されます。

圧縮はすべての表操作に対して使用可能にするか、またはダイレクト・パス・インサートに対してのみ使用可能にできます。すべての操作に対して暗号化が使用可能な場合、圧縮は、すべての DML 文中、およびバルク（ダイレクト・パス）挿入操作でデータが挿入される場合に実行されます。従来の DML に対して圧縮を使用可能にするには、COMPATIBLE 初期化パラメータを 11.1.0 以上に設定する必要があります。

すべての操作に対して圧縮を使用可能にするには、COMPRESS FOR ALL OPERATIONS 句を使用する必要があります。ダイレクト・パス・インサートに対してのみ圧縮を使用可能にするには、COMPRESS FOR DIRECT_LOAD OPERATIONS 句を使用します。キーワード COMPRESS は、単独で COMPRESS FOR DIRECT_LOAD OPERATIONS 句と同じであり、以前のデータベース・リリースと同じ圧縮動作が起動されます。

圧縮表の列の追加と削除

表のすべての操作で圧縮が使用可能なときには、表の列の追加および削除ができます。ダイレクト・パス・インサートに対してのみ圧縮が使用可能な場合は、列を削除できず、デフォルト値を指定しない場合にのみ列を追加できます。

例

次の例では、OLTP アプリケーションで使用される表 transaction のすべての操作に対して圧縮が使用可能になります。

```
CREATE TABLE transaction ( ... ) COMPRESS FOR ALL OPERATIONS;
```

次の 2 つの例では、データ・ウェアハウスのファクト表である sales_history 表のダイレクト・パス・インサートに対してのみ圧縮が使用可能になります。

```
CREATE TABLE sales_history ( ... ) COMPRESS FOR DIRECT_LOAD OPERATIONS;

CREATE TABLE sales_history ( ... ) COMPRESS;
```

圧縮とパーティション表

圧縮はパーティション・レベルで使用可能または使用禁止にできます。したがって、圧縮パーティションと非圧縮パーティションの両方を含む表を作成できます。表に対する圧縮の設定とそのパーティションに対する設定が一致しない場合、パーティションについてはパーティションの設定が優先されます。次の例では、northeast パーティション以外のすべてのパーティションが圧縮されます。

```
CREATE TABLE sales
      (saleskey number,
       quarter number,
       product number,
       salesperson number,
       amount number(12, 2),
       region varchar2(10)) COMPRESS
   PARTITION BY LIST (region)
      (PARTITION northwest VALUES ('NORTHWEST'),
       PARTITION southwest VALUES ('SOUTHWEST'),
       PARTITION northeast VALUES  ('NORTHEAST') NOCOMPRESS,
       PARTITION southeast VALUES ('SOUTHEAST'),
       PARTITION western VALUES ('WESTERN'));
```

表が圧縮されているかどうかの判別

圧縮表の場合は、*_TABLES データ・ディクショナリ・ビューの COMPRESSION 列に ENABLED と表示されます。パーティション表の場合はこの列が NULL で、*_TAB_PARTITIONS データ・ディクショナリ・ビューの COMPRESSION 列に、圧縮されているパーティションが示されます。さらに、COMPRESS_FOR 列に、表の圧縮対象が FOR ALL OPERATIONS（すべての操作）か DIRECT LOAD ONLY（ダイレクト・ロードのみ）かが表示されます。

```
SQL> SELECT table_name, compression, compress_for FROM user_tables;

TABLE_NAME       COMPRESS COMPRESS_FOR
---------------- -------- ------------------
T1               DISABLED
T2               ENABLED  DIRECT LOAD ONLY
T3               ENABLED  FOR ALL OPERATIONS
```

関連項目：

- CREATE TABLE...COMPRESS 文および ALTER TABLE...COMPRESS 文の制限事項などの詳細は、『Oracle Database SQL リファレンス』を参照してください。
- 表パーティション化の詳細は、『Oracle Database VLDB およびパーティショニング・ガイド』を参照してください。
- 18-16 ページ「ダイレクト・パス・インサートを使用したデータの表への挿入」

機密データを格納する列の暗号化

機密データを格納する個々の表の列を暗号化できます。機密データには、社会保障番号、クレジット・カード番号、医療記録などがあります。列の暗号化は、アプリケーションに対して完全に透過的ですが、いくつか制限事項があります。

暗号化は、セキュリティの問題をすべて解決するわけではありませんが、ユーザーがデータベースのセキュリティ機能を迂回して、オペレーティング・システムのファイル・システムから直接データベース・ファイルにアクセスしようとした場合に、そのユーザーからデータを保護します。

列の暗号化では Oracle Database の透過的データ暗号化が使用されます。この機能を使用するには、データベースのマスター暗号化キーを格納するための Oracle ウォレットを作成する必要があります。暗号化列を含む表を作成する場合、および暗号化データを格納または取得する場合は、ウォレットがオープンしている必要があります。ウォレットは、オープンするとすべてのセッションで使用可能になり、明示的にクローズするか、データベースが停止されるまではオープンしたままになります。

透過的データ暗号化では、次に示す Advanced Encryption Standard（AES）アルゴリズムや Triple Data Encryption Standard（3DES）アルゴリズムなど、業界標準の暗号化アルゴリズムがサポートされています。

- 3DES168
- AES128
- AES192
- AES256

使用するアルゴリズムは表の作成時に選択します。表のすべての暗号化列で同じアルゴリズムが使用されます。デフォルトは AES192 です。暗号化キーの長さはアルゴリズム名で示されています。たとえば、AES128 アルゴリズムでは 128 ビットのキーが使用されます。

1 つ以上の表にある多数の列を暗号化する場合は、かわりに表領域全体を暗号化してその表領域にこれらの表を格納することも考慮できます。表領域の暗号化でも同様に透過的データ暗号化機能が使用されますが、物理的なブロック・レベルで暗号化されるため、多数の列を暗号化するよりパフォーマンスが向上します。表領域レベルで暗号化する別の理由は、列暗号化の次の制限事項に対処するためです。

- COMPATIBLE 初期化パラメータが 10.2.0（透過的データ暗号化を使用可能にするための最小設定値）に設定されている場合、ソートまたはハッシュ結合に関与していて一時表領域に書き込まれる暗号化列のデータは、平文で書き込まれるため、攻撃にさらされます。一時表領域に書き込まれる暗号化データを暗号化されたままにするには、COMPATIBLE を 11.1.0 以上に設定する必要があります。なお、UNDO 表領域または REDO ログに書き込まれる場合は、COMPATIBLE が 10.2.0 以上に設定されていれば、暗号化列のデータは暗号化されたままです。
- オブジェクト・データ型などの特定のデータ型は、列暗号化ではサポートされていません。
- 暗号化列がある表が含まれた表領域に対しては、トランスポータブル表領域機能を使用できません。
- その他の制限事項については、『Oracle Database Advanced Security 管理者ガイド』を参照してください。

関連項目：

- 12-8 ページ「暗号化された表領域」
- 18-9 ページ「例 : 表の作成」
- 透過的データ暗号化の詳細、およびウォレットの作成とオープンの手順については、『Oracle Database Advanced Security 管理者ガイド』を参照してください。
- CREATE TABLE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。
- Oracle Real Application Clusters 環境で Oracle ウォレットを使用する方法については、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。
- 12-30 ページ「データベース間での表領域のトランスポート」

表サイズの見積りと見積りに応じた計画

表を作成する前に表のサイズを見積ります。見積りは、なるべくデータベース計画の一部として実行します。データベース表のサイズと用途を確認することは、データベース計画の重要な部分です。

表の見積りサイズの合計と、索引、UNDO 領域および REDO ログ・ファイルの見積りを使用して、作成するデータベースを格納するために必要なディスク容量を決定できます。この見積りによって、適切なハードウェアを購入できます。

見積ったサイズと個々の表サイズの増加率を使用すると、作成する表に最適な表領域の属性とその基礎となるデータファイルを適格に判断できます。これによって、表のディスク領域の管理が容易になり、表を使用するアプリケーションの I/O パフォーマンスが向上します。

表作成時の制限事項

表の計画と使用に影響を与える可能性のある制限事項がいくつかあります。

- オブジェクト型を含む表は、Oracle8 より古いバージョンのデータベースにインポートできません。
- エクスポートされた表は、異なるスキーマで同じ名前の付いた既存の表にマージできません。
- オリジナルのデータがデータベースにまだ存在するときは、型とエクステント表を異なるスキーマには移動できません。
- Oracle Database には、表が持つ列（またはオブジェクト型の属性）の合計数に制限があります。この制限については、『Oracle Database リファレンス』を参照してください。

  ユーザー定義型のデータを含む表を作成すると、ユーザー定義型の列はその型データを格納するリレーショナル列にマップされます。これにより、追加のリレーショナル列が作成されます。これらのリレーショナル列は「非表示」で、DESCRIBE 表の文では表示されず、SELECT * 文でも返されません。したがって、オブジェクト表、REF の列を持つリレーショナル表、VARRAY、ネストした表またはオブジェクト型を作成するときは、データベースが表に対して実際に作成した列の合計数が、指定した数よりも多くなることがあるので注意してください。

  関連項目： ユーザー定義型の詳細は、『Oracle Database オブジェクト・リレーショナル開発者ガイド』を参照してください。

表の作成

自分のスキーマに新しい表を作成するには、CREATE TABLE システム権限が必要です。別のユーザーのスキーマに表を作成するには、CREATE ANY TABLE システム権限が必要です。また、表の所有者には、その表を含む表領域に対する割当て制限または UNLIMITED TABLESPACE システム権限が必要です。

表は SQL 文 CREATE TABLE を使用して作成します。

この項の内容は、次のとおりです。

> **関連項目 :** この章で説明している CREATE TABLE などの SQL 文の正確な構文については、『Oracle Database SQL リファレンス』を参照してください。

例 : 表の作成

次の文を発行すると、表 admin_emp が hr スキーマに作成され、admin_tbs 表領域に格納されます。

```
CREATE TABLE hr.admin_emp (
         empno      NUMBER(5) PRIMARY KEY,
         ename      VARCHAR2(15) NOT NULL,
         ssn        NUMBER(9) ENCRYPT,
         job        VARCHAR2(10),
         mgr        NUMBER(5),
         hiredate   DATE DEFAULT (sysdate),
         photo      BLOB,
         sal        NUMBER(7,2),
         hrly_rate  NUMBER(7,2) GENERATED ALWAYS AS (sal/2080),
         comm       NUMBER(7,2),
         deptno     NUMBER(3) NOT NULL
                     CONSTRAINT admin_dept_fkey REFERENCES hr.departments
                     (department_id))
   TABLESPACE admin_tbs
   STORAGE ( INITIAL 50K);

COMMENT ON TABLE hr.admin_emp IS 'Enhanced employee table';
```

次に、この例について説明します。

- 表の複数の列で整合性制約が定義されています。
- STORAGE 句では、第 1 エクステントのサイズが指定されています。この句の詳細は、『Oracle Database SQL リファレンス』を参照してください。
- 1 つの列（ssn）で、Oracle Database の透過的データ暗号化機能を使用した暗号化が定義されています。したがって、この CREATE TABLE 文を正常に実行するためには、Oracle ウォレットがオープンしている必要があります。
- photo 列のデータ型は BLOB です。このデータ型は、ラージ・オブジェクト（LOB）と呼ばれるデータ型セットのメンバーです。LOB は、半構造化データ（例 : XML ツリー）および非構造化データ（例 : 色イメージのビット・ストリーム）の格納に使用されます。
- 1 つの列（hrly_rate）が仮想列として定義されています。この列は、年収を 2,080 で除算して従業員の時給を計算しています。仮想列に関するルールの説明は、『Oracle Database SQL リファレンス』を参照してください。
- COMMENT 文を使用して、表に関するコメントが格納されています。*_TAB_COMMENTS データ・ディクショナリ・ビューを問い合せると、このようなコメントを取得できます。詳細は、『Oracle Database SQL リファレンス』を参照してください。

関連項目：

- 表列に指定できるデータ型の詳細は、『Oracle Database SQL リファレンス』を参照してください。
- 16-10 ページ「整合性制約の管理」
- 透過的データ暗号化および Oracle ウォレットの詳細は、『Oracle Database Advanced Security 管理者ガイド』を参照してください。
- LOB の詳細は、『Oracle Database SecureFiles およびラージ・オブジェクト開発者ガイド』を参照してください。

一時表の作成

一時表は、複数の DML 操作の実行によって作成されるため、結果セットがバッファリング（一時的に保存）されるアプリケーションに有用です。たとえば、次のような場合を考えてみます。

Web ベースの航空予約アプリケーションでは、顧客がオプションの旅程を複数作成できます。各旅程は一時表の行で表されます。アプリケーションは、旅程への変更を反映するように行を更新します。使用する旅程を顧客が決定すると、アプリケーションは、該当する旅程の行を永続表に移動します。

セッションの開始時から終了時まで旅程データはプライベートです。セッションの終了時に、オプションの旅程は削除されます。

一時表の定義はすべてのセッションで参照できますが、一時表内のデータを参照できるのは、そのデータを表に挿入するセッションのみです。

一時表を作成するには、CREATE GLOBAL TEMPORARY TABLE 文を使用します。ON COMMIT 句は、表のデータが**トランザクション固有**（デフォルト）であるか、**セッション固有**であるかを示します。各オプションが表す意味は、次のとおりです。

ON COMMIT 設定	意味
DELETE ROWS	トランザクション固有の一時表を作成します。セッションは、表に最初に挿入するトランザクションを持つ一時表に対するバインドになります。バインドは、トランザクション終了時に消失します。表は、各コミット後に切捨て（すべての行を削除）が行われます。
PRESERVE ROWS	セッション固有の一時表を作成します。セッションは、セッション内で表に最初に挿入される一時表に対するバインドになります。このバインドは、セッションの最後で、またはセッション内で表に対する TRUNCATE が発行されることによって消去されます。表は、セッション終了時に切り捨てられます。

次の文では、トランザクション固有の一時表を作成しています。

```
CREATE GLOBAL TEMPORARY TABLE admin_work_area
        (startdate DATE,
         enddate DATE,
         class CHAR(20))
      ON COMMIT DELETE ROWS;
```

一時表には索引を作成できます。この索引も一時索引であり、索引内のデータのセッションまたはトランザクションの有効範囲は、基礎となる表のデータと同じです。

デフォルトで、一時表の行は、作成したユーザーのデフォルトの一時表領域に格納されます。ただし、一時表の作成時に CREATE GLOBAL TEMPORARY TABLE の TABLESPACE 句を使用すると、一時表を別の表領域に割り当てることができます。この機能を使用すると、一時表で使用される領域を節約できます。たとえば、小規模な一時表の操作を多数実行する必要があるとします。このとき、デフォルトの一時表領域はソート操作用に構成されているためにエクステント・サイズが大きい場合、これらの小規模な操作では不要なディスク領域が大量に消費されます。この場合は、エクステント・サイズの小さい第 2 の一時表領域を割り当てることをお薦めします。

次の 2 つの文では、エクステント・サイズが 64KB で一時表領域が作成され、その表領域に新規の一時表が作成されます。

```
CREATE TEMPORARY TABLESPACE tbs_t1
    TEMPFILE 'tbs_t1.f' SIZE 50m REUSE AUTOEXTEND ON
    MAXSIZE UNLIMITED
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 64K;

CREATE GLOBAL TEMPORARY TABLE admin_work_area
        (startdate DATE,
         enddate DATE,
         class CHAR(20))
      ON COMMIT DELETE ROWS
      TABLESPACE tbs_t1;
```

関連項目： 12-10 ページ「一時表領域」

永続表とは異なり、一時表とその索引には、作成時にセグメントが自動的に割り当てられません。かわりに、最初に INSERT（または CREATE TABLE AS SELECT）が実行されると、セグメントが割り当てられます。これは、最初の INSERT の前に、SELECT、UPDATE または DELETE が実行されると、表が空に見えることを意味します。

既存の一時表で DDL 操作（TRUNCATE を除く）が許可されるのは、その一時表にバインドされているセッションがない場合のみです。

トランザクションをロールバックすると、入力したデータは消失しますが、表定義はそのまま残ります。

トランザクション固有の一時表では、1 回に 1 トランザクションのみが許可されます。単一のトランザクションに複数の自律型トランザクションがある場合、各自律型トランザクションは、直前のトランザクションのコミット直後にのみ表を使用できます。

一時表のデータは、その定義どおり一時的なため、一時表データのバックアップとリカバリはシステム障害のイベントでは使用できません。このような障害に備えて、一時表データを保存する代替方法を用意してください。

表作成のパラレル化

表の作成に AS SELECT 句を指定して、別の表からデータを移入すると、パラレル実行を利用できます。CREATE TABLE...AS SELECT 文には、CREATE 部分（DDL）と SELECT 部分（問合せ）の 2 つの部分があります。Oracle Database では、この文の両方の部分をパラレル化できます。次の条件が 1 つでも成り立つ場合は、CREATE 部分がパラレル化されます。

- PARALLEL 句が CREATE TABLE...AS SELECT 文に含まれている。
- ALTER SESSION FORCE PARALLEL DDL 文が指定されている。

次の条件がすべて成り立つ場合は、問合せ部分がパラレル化されます。

- 問合せにパラレル・ヒント指定（PARALLEL または PARALLEL_INDEX）が含まれている、または CREATE 部分に PARALLEL 句が含まれている、または問合せの中で参照されるスキーマ・オブジェクトに対応付けられた PARALLEL 宣言がある。
- 問合せで指定した表のうち少なくとも 1 つで、全表スキャンまたは複数のパーティションにまたがる索引レンジ・スキャンが必要である。

表の作成をパラレル化した場合、その表には対応付けられたパラレル宣言（PARALLEL 句）が付きます。表に対するその後のすべての DML または問合せでは、パラレル化が可能な場合、パラレル実行の使用が試みられます。

表の作成をパラレル化し、表圧縮を使用して圧縮形式で結果を格納する簡単な文を次に示します。

```
CREATE TABLE hr.admin_emp_dept
     PARALLEL COMPRESS
     AS SELECT * FROM hr.employees
     WHERE department_id = 10;
```

この場合の PARALLEL 句は、表の作成時に最適な数のパラレル実行サーバーを選択することをデータベースに指示しています。

関連項目：

- パラレル実行の使用に関する詳細は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。
- 4-21 ページ「SQL のパラレル実行用プロセスの管理」

表のロード

表にデータを挿入または初期ロードするには、いくつかの方法があります。最も一般的に使用される方法は、次のとおりです。

方法	説明
SQL*Loader	これは、外部ファイルから Oracle Database の表にデータをロードする Oracle のユーティリティ・プログラムです。 SQL*Loader の詳細は、『Oracle Database ユーティリティ』を参照してください。
CREATE TABLE ...AS SELECT 文（CTAS）	この SQL 文を使用すると、表を作成し、別の既存の表から選択したデータを移入できます。
INSERT 文	INSERT 文を使用すると、列値を指定するか、または別の既存の表からデータを選択する副問合せを指定することによって、行を表に追加できます。
MERGE 文	MERGE 文を使用すると、別の既存の表から行を選択することによって、行を表に挿入するか、または表の行を更新できます。新しいデータの行が、表にすでに存在している項目に対応している場合は UPDATE が実行され、対応する項目がない場合は INSERT が実行されます。

CREATE TABLE ... AS SELECT、INSERT および MERGE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

DML エラー・ロギングを使用したデータの挿入

副問合せで INSERT 文を使用して表をロードすると、エラーが発生した場合は文が終了して文全体がロールバックされます。これは、時間とシステム・リソースを無駄に消費することになります。このような INSERT 文の場合は、DML エラー・ロギング機能を使用することで、この状況を回避できます。

DML エラー・ロギングを使用するには、エラー・ロギング表の名前を指定する句を文に追加します。データベースは、このエラー・ロギング表に DML 操作の過程で発生したエラーを記録します。このエラー・ロギング句を INSERT 文に追加すると、特定の種類のエラーでは、文が終了してロールバックされることがなくなります。かわりに、各エラーが記録され、文は続行されます。エラーが発生した行については、後で訂正処理を実行します。

DML エラー・ロギングは、INSERT、UPDATE、MERGE および DELETE 文で機能します。ここでは、特に INSERT 文について説明します。

DML エラー・ロギングを使用してデータを挿入する手順は、次のとおりです。

1. エラー・ロギング表を作成します（オプション）。

   表は、手動で作成するか、または DBMS_ERRLOG パッケージを使用して自動的に作成できます。詳細は、18-15 ページの「エラー・ロギング表の作成」を参照してください。

2. エラー・ロギング句を指定して INSERT 文を実行します。この句は、次のように動作します。

   - 必要に応じて、作成したエラー・ロギング表を参照します。エラー・ロギング表名を指定しない場合、データベースは、デフォルトの名前のエラー・ロギング表に記録します。デフォルトのエラー・ロギング表名は、ERR$_ の後に、挿入対象となる表名の最初の 25 文字を付加した名前です。
   - 必要に応じて、***タグ***（カッコで囲まれた数値または文字列リテラル）を指定します。このタグはエラー・ログに追加され、エラーの原因となった文の識別に役立ちます。タグを省略した場合は、NULL 値が使用されます。
   - 必要に応じて、REJECT LIMIT 副次句を指定します。

     この副次句は、許容可能なエラーの最大発生数を示します。この最大数を超えると、INSERT 文が終了してロールバックされます。UNLIMITED を指定することもできます。デフォルトの拒否の上限は 0（ゼロ）です。これは、最初のエラーが発生した時点でエラーが記録され、文がロールバックされることを意味します。パラレル DML 操作では、この拒否の上限が各パラレル・サーバーに対して適用されます。

     注意： 拒否の上限を超えて文がロールバックされた場合、エラー・ロギング表には、その時点までに記録されたログ・エントリが保持されます。

   エラー・ロギング句の構文については、『Oracle Database SQL リファレンス』を参照してください。

3. エラー・ロギング表を問い合せ、エラーの原因となった行に対する訂正処理を実行します。

   エラー・ロギング表の構造については、後述の「エラー・ロギング表の書式」を参照してください。

例 次の文は、DW_EMPL 表に行を挿入し、ERR_EMPL 表にエラーを記録します。タグ 'daily_load' は、各ログ・エントリにコピーされます。エラー数が 25 を超えると、文が終了してロールバックされます。

```
INSERT INTO dw_empl
  SELECT employee_id, first_name, last_name, hire_date, salary, department_id
  FROM employees
  WHERE hire_date > sysdate - 7
  LOG ERRORS INTO err_empl ('daily_load') REJECT LIMIT 25
```

エラー・ロギングのその他の例については、『Oracle Database SQL リファレンス』および『Oracle Database データ・ウェアハウス・ガイド』を参照してください。

エラー・ロギング表の書式

エラー・ロギング表は、次の 2 つの部分で構成されます。

- エラーを説明する一連の必須列。Oracle エラー番号の列は、この一例です。

  表 18-1 に、エラーを説明するための必須列を示します。

- エラーの原因となった行のデータが格納される一連のオプション列。列名は、挿入対象の表（DML 表）の列名に対応しています。

  エラー・ロギング表のこの部分の列数は、0（ゼロ）、1 または複数（最大で DML 表の列数）の場合があります。DML 表の列と同じ名前の列がエラー・ロギング表に存在する場合は、対応するデータが、障害のある挿入予定の行から、このエラー・ロギング表の列に書き込まれます。DML 表の列に対応する列がエラー・ロギング表にない場合、その列は記録されません。エラー・ロギング表に、DML 表の列と一致しない名前の列がある場合、その列は無視されます。

  エラーが発生する原因の 1 つは型変換エラーであるため、エラー・ロギング表のオプション列のデータ型は、データの消失または変換エラーなしで値を取得できる型であることが必要です（ロギング表のオプション列の型が DML 表の列と同じ型の場合は、問題のあるデータをロギング表に取得すると、エラーの原因となった同じデータ変換の問題が発生する可能性があります）。データベースでは、変換エラーの原因となるデータの値を適切に記録するように最善の努力が行われます。値を導出できない場合、列には NULL が記録されます。エラー・ロギング表への挿入でエラーが発生した場合は、文が終了します。

  表 18-2 に、エラー・ロギング表の列のデータ型を示します。DML 表の各データ型に対しては、ここに記載されているデータ型を使用することをお薦めします。DBMS_ERRLOG パッケージを使用してエラー・ロギング表を自動的に作成する場合は、これらのデータ型が使用されます。

表 18-1 エラーを説明する必須列

列名	データ型	説明
ORA_ERR_NUMBER$	NUMBER	Oracle エラー番号
ORA_ERR_MESG$	VARCHAR2(2000)	Oracle エラー・メッセージのテキスト
ORA_ERR_ROWID$	ROWID	エラーとなった行の ROWID（更新および削除の場合）
ORA_ERR_OPTYP$	VARCHAR2(2)	操作の種類 : 挿入（I）、更新（U）、削除（D） **注意 :** MERGE 操作の UPDATE 句および INSERT 句のエラーは、U および I の値で区別されます。
ORA_ERR_TAG$	VARCHAR2(2000)	ユーザーがエラー・ロギング句に指定したタグの値

表 18-2 エラー・ロギング表の列のデータ型

DML 表の列の型	エラー・ロギング表の列の型	注意
NUMBER	VARCHAR2(4000)	変換エラーを記録できます。
CHAR/VARCHAR2(n)	VARCHAR2(4000)	情報の消失なしで値を記録します。
NCHAR/NVARCHAR2(n)	NVARCHAR2(4000)	情報の消失なしで値を記録します。
DATE/TIMESTAMP	VARCHAR2(4000)	情報の消失なしで値を記録します。デフォルトの日時書式マスクを使用して文字書式に変換します。
RAW	RAW(2000)	情報の消失なしで値を記録します。
ROWID	UROWID	ROWID 型を記録します。
LONG/LOB		サポートしていません。
ユーザー定義型		サポートしていません。

エラー・ロギング表の作成

エラー・ロギング表は手動で作成できます。または、PL/SQL パッケージを使用して自動的に作成できます。

エラー・ロギング表の自動作成 エラー・ロギング表を自動作成するには、DBMS_ERRLOG パッケージを使用します。CREATE_ERROR_LOG プロシージャは、エラーを説明するための必須列および指定された DML 表の列をすべて備えたエラー・ロギング表を作成し、表 18-2 に示したデータ型マッピングを実行します。

次の文は、前述の例で使用したエラー・ロギング表を作成します。

```
EXECUTE DBMS_ERRLOG.CREATE_ERROR_LOG('DW_EMPL', 'ERR_EMPL');
```

DBMS_ERRLOG の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

手動によるエラー・ロギング表の作成 エラー・ロギング表を手動で作成するには、標準 DDL を使用します。表の構造に関する要件は、18-14 ページの「エラー・ロギング表の書式」を参照してください。エラーを説明するための必須列はすべて挿入する必要があります。列は順不同にできますが、必須列は表の最初の方の列に指定する必要があります。

エラー・ロギングの制限事項と注意

Oracle Database は、DML 操作中に次のエラーを記録します。

- 列の値が大きすぎる場合
- 制約（NOT NULL 制約、一意制約、参照制約、CHECK 制約）違反の場合
- トリガー実行時にエラーが発生した場合
- 副問合せの列と表内の対応する列との間の型変換でエラーが発生した場合
- パーティション・マッピング・エラーの場合
- 特定の MERGE 操作エラー（ソース表の安定したセット行を取得できません）の場合

一部のエラーは記録されずに、DML 操作の終了およびロールバックが実施されます。これらのエラーの一覧と DML ロギングの他の制約については、『Oracle Database SQL リファレンス』の INSERT に関する項で error_logging_clause の説明を参照してください。

領域に関する考慮事項 DML エラー・ロギングを使用するには、その前に領域の要件について考慮する必要があります。挿入する表の領域のみでなく、エラー・ロギング表の領域も必要です。

セキュリティ DML エラー・ロギングを指定した INSERT 文を発行するユーザーには、エラー・ロギング表に対する INSERT 権限が必要です。

> **関連項目：** DML エラー・ロギングの例は、『Oracle Database SQL リファレンス』および『Oracle Database データ・ウェアハウス・ガイド』を参照してください。

ダイレクト・パス・インサートを使用したデータの表への挿入

Oracle Database では、次の 2 つのいずれかの方法でデータが挿入されます。

- **従来型のインサート処理**では、表内の空き領域が再利用され、新規に挿入するデータと既存のデータが相互に配置されます。このような操作では、参照整合性制約も維持されます。
- **ダイレクト・パス・インサート処理**では、表内の既存データの後ろに挿入データが追加されます。データは、バッファ・キャッシュをバイパスしてデータファイルに直接書き込まれます。既存のデータの空き領域は再利用されず、参照整合性制約は無視されます。これらのプロシージャを組み合せることで、パフォーマンスを高めることができます。

さらに、シリアルまたはパラレルのいずれかのモードでデータを挿入できます。シリアル・モードでは、単一のプロセスが文を実行し、パラレル・モードでは、複数のプロセスが 1 つの SQL 文を実行するために協調して同時に動作します。後者は、パラレル実行と呼ばれます。

ここでは、表へのデータの挿入について、1 つの方法を中心に説明します。具体的には、INSERT 文のダイレクト・パス形式を使用します。この章の内容は、次のとおりです。

注意： このマニュアルに記載されている、表にデータを挿入する詳細と例は少数です。データ・ウェアハウスおよびアプリケーション開発に関するオラクル社のマニュアルには、表へのデータの挿入および操作に関する広範囲にわたる情報が記載されています。次に例を示します。

- 『Oracle Database データ・ウェアハウス・ガイド』
- 『Oracle Database アドバンスト・アプリケーション開発者ガイド』
- 『Oracle Database SecureFiles およびラージ・オブジェクト開発者ガイド』

ダイレクト・パス・インサートを使用する利点

ダイレクト・パス INSERT のパフォーマンス上の利点は、次のとおりです。

- ダイレクト・パス INSERT では、REDO および UNDO エントリのロギングを使用禁止にできます。これに対して、従来型のインサート処理では空き領域を再利用し、参照整合性を維持するため、これらのエントリを常にロギングする必要があります。
- ダイレクト・パス INSERT 処理は、パラレル・モードで実行する場合にも、トランザクションの原子性が保証されます。原子性は、パラレル・ダイレクト・パス・ロード（SQL*Loader を使用）では保証されません。
- パラレル・ダイレクト・パス・ロード中にエラーが発生すると、ロード終了時に一部の索引に UNUSABLE のマークが付けられることがあります。対照的に、パラレル・ダイレクト・パス INSERT の場合は、索引更新時にエラーが発生すると、文がロールバックされます。

ダイレクト・パス・インサートの使用可能化

ダイレクト・パス INSERT 文を使用してパラレル・モードでデータを挿入することによって、または Oracle の SQL*Loader ユーティリティをダイレクト・パス・モードで使用することによって、ダイレクト・パス INSERT 処理を実装できます。ダイレクト・パス・インサートは、シリアル・モードまたはパラレル・モードで実行できます。

シリアル・モードでダイレクト・パス INSERT をアクティブにするには、INSERT キーワードの直後または INSERT 文の副問合せの SELECT キーワードの直後にある各 INSERT 文に APPEND ヒントを指定する必要があります。

パラレル DML モードで挿入する場合は、ダイレクト・パス INSERT がデフォルトです。パラレル DML モードで実行するには、次の要件を満たす必要があります。

- Oracle Enterprise Edition がインストールされていること。
- セッションでパラレル DML が使用可能であること。そのためには、次の文を実行します。

  ```
  ALTER SESSION { ENABLE | FORCE } PARALLEL DML;
  ```

- 作成時または作成後に目的の表に対してパラレル属性を指定すること。または、各インサート処理に PARALLEL ヒントを指定すること。

ダイレクト・パス INSERT を使用禁止にするには、各 INSERT 文に NOAPPEND ヒントを指定します。この指定によって、パラレル DML モードが無視されます。

注意：

- ダイレクト・パス INSERT がサポートするのは、INSERT 文の副問合せ構文でのみで、VALUES 句はサポートされません。INSERT 文の副問合せ構文の詳細は、『Oracle Database SQL リファレンス』を参照してください。
- ダイレクト・パス INSERT の使用については、他の制限事項があります。詳細は、『Oracle Database SQL リファレンス』を参照してください。

関連項目： ヒントの使用の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

ダイレクト・パス・インサートの動作

ダイレクト・パス INSERT は、パーティション表と非パーティション表の両方で使用できます。

パーティション表または非パーティション表へのシリアル・ダイレクト・パス・インサート シングル・プロセスでは、表セグメントまたは各パーティション・セグメントの現在の最高水位標の上にデータが挿入されます（**最高水位標**とは、データを受け取るためにブロックがフォーマットされないレベルです）。COMMIT を実行すると、最高水位標が新しい値に更新され、ユーザーにデータが表示されます。

パーティション表へのパラレル・ダイレクト・パス・インサート この状況は、シリアル・ダイレクト・パス INSERT と類似しています。各パラレル実行サーバーには、1 つ以上のパーティションが割り当てられます。単一のパーティションで実行するプロセスは 1 つのみです。各パラレル実行サーバーでは、割り当てられたパーティション・セグメントの現在の最高水位標の上にデータを挿入します。COMMIT を実行すると、各パーティション・セグメントの最高水位標が新しい値に更新され、ユーザーにデータが表示されます。

非パーティション表へのパラレル・ダイレクト・パス・インサート 各パラレル実行サーバーは、新しい一時セグメントを割り当て、その一時セグメントにデータを挿入します。COMMIT を実行すると、パラレル実行コーディネータが新しい一時セグメントをプライマリ表セグメントにマージし、ユーザーにデータが表示されます。

ダイレクト・パス・インサートのロギング・モードの指定

ダイレクト・パス INSERT では、インサート処理の REDO およびロールバック情報を記録するかどうかを選択できます。

- 作成時（CREATE 文で）または作成後（ALTER 文で）に、表、パーティション、索引または LOB 記憶域について、ロギング・モードを指定できます。
- 作成時または作成後に LOGGING または NOLOGGING を指定しないと、次のようにデフォルト設定されます。
  - パーティションのロギング属性は、その表のロギング属性にデフォルト設定されます。
  - 表または索引のロギング属性は、常駐している表領域のロギング属性にデフォルト設定されます。
  - LOB 記憶域のロギング属性は、LOB 記憶域に CACHE を指定した場合は LOGGING にデフォルト設定されます。CACHE を指定しなかった場合、ロギング属性は LOB 値が常駐している表領域の属性にデフォルト設定されます。
- CREATE TABLESPACE または ALTER TABLESPACE 文で、表領域のロギング属性を設定します。

> **注意：** データベースまたは表領域が FORCE LOGGING モードの場合、ダイレクト・パス INSERT は、ロギング設定に関係なく常にロギングされます。

ロギング付きダイレクト・パス・インサート このモードでは、Oracle Database によってインスタンスの完全な REDO ロギングおよびメディア・リカバリが実行されます。データベースが ARCHIVELOG モードの場合は、REDO ログをテープにアーカイブできます。データベースが NOARCHIVELOG モードの場合、インスタンスのクラッシュはリカバリできますが、ディスク障害はリカバリできません。

ロギングなしダイレクト・パス・インサート このモードでは、Oracle Database によって REDO または UNDO ロギングなしでデータが挿入されます（新規エクステントに無効のマークを付けるために最小限のロギングが行われ、データ・ディクショナリの変更は常にロギングされます）。このモードによって、パフォーマンスが改善します。ただし、後でメディア・リカバリを実行する必要がある場合は、REDO データがロギングされていないため、エクステント無効化レコードによって一連のブロックに論理的破損のマークが付きます。したがって、このようなインサート処理の後にはデータをバックアップすることが重要です。

ダイレクト・パス・インサートのその他の考慮事項

ダイレクト・パス INSERT を使用する際には、さらに次の考慮事項があります。

圧縮表 表の作成に COMPRESS または COMPRESS FOR DIRECT_LOAD OPERATIONS 句を使用する場合、ダイレクト・パス INSERT を使用して表のデータをロード時に圧縮できます。表の作成に COMPRESS FOR ALL OPERATIONS 句を使用する場合、従来型 INSERT またはダイレクト・パス INSERT を使用して表のデータをロード時に圧縮できます。

詳細は、18-5 ページの「表圧縮の使用」を参照してください。

ダイレクト・パス・インサートでの索引メンテナンス 索引がある（パーティションまたは非パーティション）表では、ダイレクト・パス INSERT 処理の終了時に、Oracle Database が索引メンテナンスを実行します。この索引メンテナンスは、パラレル・ダイレクト・パス INSERT に対してはパラレル実行サーバーで、シリアル・ダイレクト・パス INSERT に対してはシングル・プロセスで実行されます。INSERT 処理の前に索引を削除し、後で再作成することによって、索引メンテナンスでのパフォーマンスへの影響を回避できます。

ダイレクト・パス・インサートでの領域に関する考慮事項 ダイレクト・パス INSERT は、従来型パス INSERT よりも多くの領域を必要とします。

すべてのシリアル・ダイレクト・パス INSERT 処理では、パーティション表へのパラレル・ダイレクト・パス INSERT と同様に、影響を受けるセグメントの最高水位標の上にデータが挿入されます。このため、追加の領域が必要となります。

非パーティション表へのパラレル・ダイレクト・パス INSERT は、各並列度ごとに一時セグメントを作成するため、より多くの領域を必要とします。非パーティション表が自動セグメント領域管理モードのローカル管理表領域にない場合は、NEXT および PCTINCREASE 記憶域パラメータ、および MINIMUM EXTENT 表領域パラメータの値を変更して、一時セグメントに十分な（かつ過剰ではない）記憶域を用意してください。次の事項を考慮に入れ、これらのパラメータに値を選択します。

- 各エクステントのサイズは、さほど小さくありません（1MB 以上）。この設定は、オブジェクト内のエクステント総数に影響を与えます。
- 各エクステントのサイズが小さいと、パラレル INSERT では、必要以上に大きいセグメントで領域を無駄にすることになります。

これらのパラメータは、ダイレクト・パス INSERT 処理の完了後に、シリアル処理に適した設定に再設定できます。

ダイレクト・パス・インサートでのロックに関する考慮事項 ダイレクト・パス INSERT では、表（またはパーティション表のすべてのパーティション）の排他ロックが取得されます。その結果、ユーザーは、表に対する挿入、更新または削除の同時操作すべてを実行できません。同時索引作成および作成操作も許可されません。同時問合せはサポートされますが、問合せではインサート処理以前の情報のみが返されます。

表に関する統計の自動収集

PL/SQL パッケージ DBMS_STATS を使用すると、コストベースの最適化に関する統計を生成および管理できます。このパッケージを使用して、統計の収集、変更、表示、エクスポート、インポートおよび削除ができます。また、すでに収集した統計を識別または命名する際も、このパッケージを使用できます。

以前は、DBMS_STATS を使用可能にし、CREATE（または ALTER）TABLE 文で MONITORING キーワードを指定して、表の統計を自動的に収集していました。Oracle Database 11*g* からは、MONITORING および NOMONITORING キーワードは非推奨になり、統計は自動的に収集されます。これらのキーワードを指定しても無視されます。

監視では、統計が最後に収集された時点以降表に対して実行された INSERT、UPDATE および DELETE の概数が追跡されます。影響を受ける行数に関する情報は、SMON が周期的に（およそ 3 時間ごとに）データをデータ・ディクショナリに取り込むまで、システム・グローバル領域（SGA）に保持されます。このデータ・ディクショナリ情報は、DBA_TAB_MODIFICATIONS、ALL_TAB_MODIFICATIONS または USER_TAB_MODIFICATIONS を通じて参照できます。データベースはこれらのビューを使用して、失効した統計を持つ表を識別します。

表の監視を使用禁止にするには、STATISTICS_LEVEL 初期化パラメータを BASIC に設定します。デフォルトは TYPICAL で、自動統計収集が使用可能です。自動統計収集と DBMS_STATS パッケージによって、オプティマイザは正確な実行計画を生成できます。

関連項目：

- STATISTICS_LEVEL 初期化パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- オプティマイザ統計の管理の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- DBMS_STATS パッケージを使用する詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- スケジューラを使用して統計を自動的に収集する方法は、24-2 ページの「自動化メンテナンス・タスクの概要」を参照してください。

表の変更

表を変更するには ALTER TABLE 文を使用します。表を変更するには、その表が自分のスキーマに含まれているか、その表の ALTER オブジェクト権限または ALTER ANY TABLE システム権限のいずれかを持っている必要があります。

ALTER TABLE 文の使用方法については、次の各項を参照してください。

- ALTER TABLE 文を使用する理由
- 表の物理属性の変更
- 新規セグメントまたは表領域への表の移動
- 表の記憶域の手動割当て
- 既存の列定義の変更
- 表の列の追加
- 表の列名の変更
- 表の列の削除
- 表を読取り専用モードにする方法

注意： 表を変更する前に、表を変更した結果についてよく理解しておいてください。これらの結果については、『Oracle Database SQL リファレンス』の ALTER TABLE 句の説明を参照してください。

パッケージのビュー、マテリアライズド・ビュー、トリガー、ドメイン索引、ファンクション索引、CHECK 制約、ファンクション、プロシージャが実表に依存する場合は、その実表または列を変更すると依存するオブジェクトに影響する可能性があります。データベースによる依存性管理の詳細は、16-17 ページの「オブジェクト依存性の管理」を参照してください。

ALTER TABLE 文を使用する理由

ALTER TABLE 文は、表に影響を与える次の処理を実行するために使用できます。

- 物理的な特性（INITRANS または記憶域パラメータ）を変更する場合
- 表を新しいセグメントまたは表領域に移動する場合
- 明示的にエクステントを割り当てるか、未使用領域の割当てを解除する場合
- 列を追加、削除または名前変更する場合、あるいは既存の列定義（データ型、長さ、デフォルト値、NOT NULL 整合性制約、列の式（仮想列の場合）および暗号化プロパティ）を変更する場合
- 表のロギング属性を変更する場合
- CACHE/NOCACHE 属性を変更する場合
- 表に関連付けられている整合性制約を追加、変更または削除する場合
- 表に関連付けられている整合性制約またはトリガーを使用可能にするか、使用禁止にする場合
- 表の並列度を変更する場合
- 表の名前を変更する場合
- 表を読取り専用モードにする場合および読取り / 書込みモードに戻す場合
- 索引構成表の特性を追加または変更する場合
- 外部表の特性を変更する場合
- LOB 列を追加または変更する場合
- オブジェクト型、ネストした表または VARRAY の列を追加または変更する場合

これらの多くのタスクについて、次の各項で説明します。

表の物理属性の変更

表のトランザクション・エントリ設定 INITRANS を変更する場合、INITRANS の新しい設定は、その後表に割り当てられるデータ・ブロックにのみ適用されます。

記憶域パラメータ INITIAL と MINEXTENTS は変更できません。他の記憶域パラメータ（たとえば NEXT や PCTINCREASE）の新しい設定はすべて、その後に表に割り当てられるエクステントにのみ影響します。割り当てられる次のエクステントのサイズは、NEXT と PCTINCREASE の現行値によって決まります。前の値に基づいて決まるわけではありません。

> **関連項目：** 物理属性句と記憶域句の詳細は、『Oracle Database SQL リファレンス』を参照してください。

新規セグメントまたは表領域への表の移動

ALTER TABLE...MOVE 文を使用すると、非パーティション表のデータまたはパーティション表のパーティションのデータを新しいセグメントに再配置できます。必要に応じて、割当て制限がある別の表領域に再配置することもできます。また、この文を使用して ALTER TABLE では変更できないデータを含んでいる表またはパーティションの記憶域属性も変更できます。ALTER TABLE...MOVE 文に COMPRESS 句を指定すると、表圧縮を使用して新しいセグメントを格納できます。

表を新規データファイルを含む新しい表領域に移動する重要な理由の 1 つは、列データの古いバージョン（セグメントの縮小、再編成または以前の表移動によってディスクの未使用部分に現在も残されているバージョン）が、オペレーティング・システム・ユーティリティなどを使用してデータベースのアクセス制御を迂回することによって参照される可能性をなくすためです。これは、透過的データ暗号化を追加して変更しようとしている列の場合は特に重要です。

注意： ALTER TABLE...MOVE 文では、文の実行中は表に対する DML 操作が許可されません。表の移動中にも表に対して DML を使用できるようにする場合は、18-26 ページの「表のオンライン再定義」を参照してください。

次の文は、新しい記憶域パラメータを指定して、hr.admin_emp 表を新しいセグメントに移動します。

```
ALTER TABLE hr.admin_emp MOVE
      STORAGE ( INITIAL 20K
                NEXT 40K
                MINEXTENTS 2
                MAXEXTENTS 20
                PCTINCREASE 0 );
```

表を移動すると、表の行の ROWID が変わります。これによって、表の索引に UNUSABLE のマークが付き、これらの索引を使用して表にアクセスする DML に対しては、ORA-01502 エラーが返されます。表の索引を削除または再作成する必要があります。同様に、表の統計は無効になるため、表を移動した後に新しい統計を収集する必要があります。

表に LOB 列が含まれている場合は、この文を使用して、ユーザーが明示的に指定できる LOB データと、表に関連した LOB 索引セグメントを、表とともに移動できます。特に指定しない場合、デフォルトでは LOB データと LOB 索引セグメントは移動されません。

関連項目： 透過的データ暗号化の詳細は、18-7 ページの「機密データを格納する列の暗号化」を参照してください。

表の記憶域の手動割当て

Oracle Database は、必要に応じて表のデータ・セグメントに追加のエクステントを動的に割り当てます。ただし、表に追加のエクステントを明示的に割り当てることもできます。たとえば、Oracle Real Application Clusters 環境で、表のエクステントを特定のインスタンスに対して明示的に割り当てることが可能です。

新しいエクステントは、ALTER TABLE...ALLOCATE EXTENT 句を使用して表に割り当てることができます。

また、ALTER TABLE 文の DEALLOCATE UNUSED 句を使用して、未使用領域の割当てを明示的に解除することもできます。この操作については、17-12 ページの「使用できない領域の再生」を参照してください。

関連項目： Oracle Real Application Clusters 環境での ALLOCATE EXTENT 句の使用方法は、『Oracle Real Application Clusters 管理およびデプロイメント・ガイド』を参照してください。

既存の列定義の変更

既存の列定義を変更するには、ALTER TABLE...MODIFY 文を使用します。列のデータ型、デフォルト値、列制約、列の式（仮想列の場合）または列の暗号化は変更できます。

既存のデータがすべて新しい長さを満たしている場合は、既存の列の長さを拡張または縮小できます。列は、バイト・セマンティクスから CHAR セマンティクスに、あるいはその逆に変更できます。空でない CHAR 列の長さを縮小するには、初期化パラメータ BLANK_TRIMMING=TRUE を設定する必要があります。

データ型 CHAR の列長を拡張するために表を変更している場合、特に表の行数が多い場合は、この操作は時間がかかり、さらに相当な追加記憶域を必要とする可能性があります。これは、各行の CHAR 値に空白を埋めて、新しい列長に合わせる必要があるためです。

> **関連項目：** 表の列の変更とその他の制限事項の詳細は、『Oracle Database SQL リファレンス』を参照してください。

表の列の追加

既存の表に列を追加するには、ALTER TABLE...ADD 文を使用します。

次の文は、hr.admin_emp 表を変更して新しい列 bonus を追加します。

```
ALTER TABLE hr.admin_emp
      ADD (bonus NUMBER (7,2));
```

表に新しい列を追加すると、DEFAULT 句を指定しないかぎり、その列は最初は NULL です。デフォルト値を指定すると、各行がデフォルト値で即時に更新されます。この処理に多少時間を要すること、および更新時には表に排他 DML ロックがかかることに注意してください。表のタイプ（例 : LOB 列のない表）によっては、NOT NULL 制約とデフォルト値の両方を指定すると、データベースによって列の追加操作が最適化され、表が DML 用にロックされる時間が大幅に短縮されます。

NOT NULL 制約付きの列を追加できるのは、表に行がまったく含まれていない場合、またはデフォルト値を指定する場合のみです。

> **関連項目：** 表の列の追加に関するその他のルールおよび制限事項は、『Oracle Database SQL リファレンス』を参照してください。

圧縮表への列の追加

表のすべての操作で圧縮が使用可能な場合は、デフォルト値を指定してもしなくても、その表に列を追加できます。ダイレクト・パス・インサートに対してのみ圧縮が使用可能な場合は、デフォルト値を指定しない場合にのみ、列を追加できます。

> **関連項目：** 18-5 ページ「表圧縮の使用」

仮想列の追加

新しい列が仮想列の場合、その値は列式によって決定されます（仮想列の値は、問合せ実行時にのみ計算されることに注意してください）。

表の列名の変更

Oracle Database では、表の既存の列の名前を変更できます。列名を変更するには、ALTER TABLE 文の RENAME COLUMN 句を使用します。新しい名前には、表の既存の列名と競合しない名前を指定する必要があります。RENAME COLUMN 句とともに他の句は使用できません。

次の文は、hr.admin_emp 表の comm 列の名前を変更します。

```
ALTER TABLE hr.admin_emp
      RENAME COLUMN comm TO commission;
```

前述のように、表の列を変更すると、依存するオブジェクトが無効になる可能性があります。ただし、列名を変更すると、ファンクション索引と CHECK 制約が引き続き有効になるように、関連するデータ・ディクショナリ表が更新されます。

また、Oracle Database では列制約の名前も変更できます。この操作については、16-14 ページの「制約名の変更」を参照してください。

注意： ALTER TABLE の RENAME TO 句の構文は RENAME COLUMN 句に似ていますが、表自体の名前の変更に使用します。

表の列の削除

索引構成表などの表から、不要になった列を削除できます。これにより、データベースの領域を解放でき、データをエクスポート / インポートしてから索引と制約を再作成する必要がなくなります。

表からすべての列を削除することはできません。また、SYS が所有している表の列も削除できません。削除しようとするとエラーが発生します。

> **関連項目：** 表からの列の削除に関するその他の制限事項およびオプションの詳細は、『Oracle Database SQL リファレンス』を参照してください。

表から列を削除する方法

ALTER TABLE...DROP COLUMN 文を発行すると、列記述子およびターゲット列に関連付けられているデータが表の各行から削除されます。1 つの文で複数の列を削除できます。

次の文は、hr.admin_emp 表から列を削除する操作の例を示しています。最初の文は、sal 列のみを削除します。

```
ALTER TABLE hr.admin_emp DROP COLUMN sal;
```

次の文は、bonus 列と comm 列を両方とも削除します。

```
ALTER TABLE hr.admin_emp DROP (bonus, commission);
```

列に未使用マークを付ける方法

大きい表のすべての行から列データを削除する際に所要時間が重要な場合は、ALTER TABLE...SET UNUSED 文を使用できます。この文は 1 つ以上の列に未使用マークを付けますが、実際にターゲット列を削除したり該当列が占めるディスク領域をリストアすることはありません。ただし、未使用マークが付けられた列は、問合せやデータ・ディクショナリ・ビューに表示されなくなり、その名前が削除されて新しい列に再利用できるようになります。その列に定義されている制約、索引および統計も、すべて削除されます。

hiredate 列と mgr 列に未使用マークを付けるには、次の文を実行します。

```
ALTER TABLE hr.admin_emp SET UNUSED (hiredate, mgr);
```

後で ALTER TABLE...DROP UNUSED COLUMNS 文を発行し、未使用マークが付いている列を削除できます。表の特定列の明示的な削除文を発行すると、未使用列もターゲット表から削除されます。

データ・ディクショナリ・ビュー USER_UNUSED_COL_TABS、ALL_UNUSED_COL_TABS または DBA_UNUSED_COL_TABS を使用すると、未使用の列を含むすべての表を表示できます。COUNT フィールドには、表の未使用の列数が表示されます。

```
SELECT * FROM DBA_UNUSED_COL_TABS;

OWNER                          TABLE_NAME                     COUNT
------------------------------ ------------------------------ -----
HR                             ADMIN_EMP                          2
```

外部表の場合は、SET UNUSED 文が ALTER TABLE DROP COLUMN 文に透過的に変換されます。外部表はデータベース内でメタデータのみで構成されているため、DROP COLUMN 文は SET UNUSED 文の実行と同じことになります。

未使用列の削除

未使用列に対して実行できるのは、ALTER TABLE...DROP UNUSED COLUMNS 文のみです。この文では、表から未使用の列が物理的に削除され、ディスク領域が再生されます。

次の ALTER TABLE 文では、オプションの句 CHECKPOINT が指定されています。この句を指定すると、指定した行数（この場合は 250 行）が処理された後に、チェックポイントが適用されます。チェックポイントによって、列削除操作中に累積される UNDO ログの量が減少し、UNDO 領域が使い果たされるおそれがなくなります。

```
ALTER TABLE hr.admin_emp DROP UNUSED COLUMNS CHECKPOINT 250;
```

圧縮表の列の削除

表のすべての操作で圧縮が使用可能な場合は、表の列を削除できます。ダイレクト・パス・インサートのみで圧縮が使用可能な場合は、表の列を削除できません。

関連項目： 18-5 ページ「表圧縮の使用」

表を読取り専用モードにする方法

表を読取り専用モードにするには、ALTER TABLE...READ ONLY 文を使用し、表を読取り / 書込みモードに戻すには、ALTER TABLE...READ WRITE 文を使用します。読取り専用モードが有効な表の例に、構成表があります。アプリケーションに含まれている構成表が、インストール後変更されず、ユーザーによる変更を禁止する必要がある場合は、アプリケーションのインストール・スクリプトによって、これらの表を読取り専用モードにできます。

表を読取り専用モードにするには、その表に対する ALTER TABLE 権限、または ALTER ANY TABLE 権限が必要です。また、COMPATIBILE 初期化パラメータが 11.1.0 以上に設定されている必要があります。

次の例は、SALES 表を読取り専用モードにします。

```
ALTER TABLE SALES READ ONLY;
```

次の例は、表を読取り / 書込みモードに戻します。

```
ALTER TABLE SALES READ WRITE;
```

表が読取り専用モードの場合、表データの変更操作は許可されません。読取り専用表で許可されない操作は、次のとおりです。

- 表またはそのパーティションに対するすべての DML 操作
- TRUNCATE TABLE
- SELECT FOR UPDATE
- ALTER TABLE ADD/MODIFY/RENAME/DROP COLUMN
- ALTER TABLE SET COLUMN UNUSED
- ALTER TABLE DROP/TRUNCATE/EXCHANGE (SUB) PARTITION

- 読取り専用表が関係している型に対する ALTER TABLE UPGRADE INCLUDING DATA または ALTER TYPE CASCADE INCLUDING TABLE DATA
- オンライン再定義
- FLASHBACK TABLE

読取り専用表で許可される操作は、次のとおりです。

- SELECT
- CREATE/ALTER/DROP INDEX
- ALTER TABLE ADD/MODIFY/DROP/ENABLE/DISABLE CONSTRAINT
- 物理的なプロパティ変更のための ALTER TABLE
- ALTER TABLE DROP UNUSED COLUMNS
- ALTER TABLE ADD/COALESCE/MERGE/MODIFY/MOVE/RENAME/SPLIT (SUB)PARTITION
- ALTER TABLE MOVE
- ALTER TABLE ENABLE ROW MOVEMENT および ALTER TABLE SHRINK
- RENAME TABLE および ALTER TABLE RENAME TO
- DROP TABLE
- ALTER TABLE DEALLOCATE UNUSED
- ALTER TABLE ADD/DROP SUPPLEMENTAL LOG

> **関連項目：** ALTER TABLE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

表のオンライン再定義

データベース・システムでは、次のような理由で表の構造を論理的または物理的に変更する必要が生じます。

- 問合せまたは DML のパフォーマンスを改善するため
- アプリケーションの変更に対応するため
- 記憶域を管理するため

Oracle Database には、表の可用性に大きな影響を与えずに表の構造を変更できるメカニズムが用意されています。このメカニズムは、**表のオンライン再定義**と呼ばれます。表のオンライン再定義では、表を再定義する従来の方法に比べて、可用性が大幅に向上します。

オンラインで表を再定義している間も、その再定義プロセスの大部分で、問合せおよび DML を使用してその表にアクセスできます。表が排他モードでロックされるのは、そのサイズや再定義の複雑さに関係なくわずかな間のみで、ユーザーに対しては完全に透過的です。

表のオンライン再定義には、再定義の対象となる表が使用している領域とほぼ同等の空き領域が必要です。新しい列を追加する場合は、より多くの領域が必要になります。

表のオンライン再定義を実行するには、Enterprise Manager のオブジェクトの再編成ウィザードまたは DBMS_REDEFINITION パッケージを使用します。

注意： オブジェクトの再編成ウィザードを起動する手順

1. Enterprise Manager の「表」ページで**「選択」**列をクリックし、再定義する表を選択します。
2. 「アクション」リストで、**「再編成」**を選択します。
3. **「実行」**をクリックします。

ここでは、DBMS_REDEFINITION パッケージを使用したオンライン再定義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：** DBMS_REDEFINITION パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

表のオンライン再定義の機能

表のオンライン再定義では、次のことが可能です。

- 表またはクラスタの記憶域パラメータの変更
- 異なる表領域への表またはクラスタの移動

> **注意：** 表を別の表領域に移動する際に、DML でその表を使用する必要性がない場合は、より簡単な ALTER TABLE MOVE コマンドを使用できます。18-22 ページの「新規セグメントまたは表領域への表の移動」を参照してください。

- 表またはクラスタ内の 1 つ以上の列の追加、変更または削除
- パーティション化サポートの追加または削除（非クラスタ化表のみ）
- パーティション構造の変更
- 同じスキーマ内の別の表領域へのパーティションの移動を含む、単一の表パーティションの物理的なプロパティの変更
- マテリアライズド・ビュー・ログまたは Oracle Streams アドバンスト・キューイングのキュー表の物理的なプロパティの変更
- パラレル問合せのサポートの追加
- 表またはクラスタの再作成による断片化の低減

> **注意：** 多くの場合、オンラインによるセグメントの縮小が断片化を削減する簡単な方法です。17-12 ページの「使用できない領域の再生」を参照してください。

- 通常の表（ヒープ構成表）から索引構成表へ、または索引構成表から通常の表への編成の変更
- リレーショナル表からオブジェクト列を持つ表へ、またはオブジェクト列を持つ表からリレーショナル表への変換
- オブジェクト表からリレーショナル表またはオブジェクト列を持つ表へ、あるいはリレーショナル表またはオブジェクト列を持つ表からオブジェクト表への変換

DBMS_REDEFINITION を使用したオンライン再定義の実行

表のオンライン再定義を実行するには、DBMS_REDEFINITION パッケージを使用します。パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

オンラインで表を再定義する手順は、次のとおりです。

1. 再定義方法（キー別または ROWID 別）を選択します。

   キー別 : 再定義に使用する主キーまたは疑似主キーを選択します。疑似主キーは、NOT NULL 制約が指定されているすべての構成要素の列を備えた一意のキーです。この方法の場合、表の再定義前のバージョンと再定義後のバージョンの主キー列は同じになります。これはデフォルトの再定義方法であり、この方法を使用することをお薦めします。

   ROWID 別 : この方法は、キーを使用できない場合に使用します。この方法では、表の再定義後のバージョンに M_ROW$$ という非表示列が追加されます。再定義の完了後は、この列を未使用としてマークするか、削除することをお薦めします。COMPATIBLE が 10.2.0 以上に設定されている場合は、再定義の最終フェーズでこの列が自動的に未使用に設定されます。次に、ALTER TABLE ... DROP UNUSED COLUMNS 文を使用してその列を削除できます。

   この方法は、索引構成表に対しては使用できません。

2. CAN_REDEF_TABLE プロシージャを起動して、表をオンラインで再定義できることを確認します。表がオンライン再定義の候補でない場合、このプロシージャは表をオンライン再定義できない理由を示すエラーを出力します。

3. 必要な論理属性と物理属性のすべてを備えた空の仮表を（再定義する表と同じスキーマ内に）作成します。削除される列の場合は、仮表の定義に含めないでください。列を追加する場合は、その列の定義を仮表に追加します。列を変更する場合は、必要なプロパティを備えた仮表にその列を作成します。

   再定義する表の索引、制約、権限付与およびトリガーすべてを備えた仮表を作成する必要はありません。これらは、依存オブジェクトをコピーするときに手順 6 で定義します。

4. （オプション）次の手順のパフォーマンスを改善するために、大きい表の再定義をパラレルで実行する場合は、次の文を発行します。

   ```
   alter session force parallel dml parallel degree-of-parallelism;
   alter session force parallel query parallel degree-of-parallelism;
   ```

5. 次の情報を指定して START_REDEF_TABLE をコールし、再定義プロセスを開始します。

   - 再定義する表のスキーマと表名
   - 仮表名
   - 再定義される表の列を仮表の列にマップする列マッピング文字列

     詳細は、18-30 ページの「列マッピング文字列の作成」を参照してください。

   - 再定義方法

     再定義方法を指定するために、パッケージ定数が用意されています。
     DBMS_REDEFINITION.CONS_USE_PK は、主キーまたは擬似主キーを使用して再定義が実行されるように指定するために使用します。
     DBMS_REDEFINITION.CONS_USE_ROWID は、ROWID を使用して再定義が実行されるように指定するために使用します。この引数を指定しない場合は、デフォルトの再定義方法（CONS_USE_PK）が使用されます。

   - 行の順序に使用する列（オプション）
   - パーティション名（パーティション表の単一のパーティションのみを再定義する場合）

   このプロセスにはデータのコピー操作が含まれるため、多少の時間を要する可能性があります。再定義する表は、プロセスの開始から終了まで問合せおよび DML で使用できます。

注意： なんらかの理由で START_REDEF_TABLE に失敗した場合は、ABORT_REDEF_TABLE をコールする必要があります。コールしないと、表を再定義する後続の試行でエラーが発生します。

6. 次の 2 つの方法のいずれかを使用して、再定義する表から仮表に依存オブジェクト（トリガー、索引、マテリアライズド・ビュー・ログ、権限付与、制約など）と統計をコピーします。第 1 の方法は、ほとんど自動化されているため、この方法を使用することをお薦めします。ただし、場合によっては第 2 の方法の使用を選択することも考えられます。第 1 の方法では、表の統計を仮表にコピーすることもできます。

   - 方法 1: 依存オブジェクトの自動作成

     COPY_TABLE_DEPENDENTS プロシージャを使用して、仮表に対する依存オブジェクトを自動的に作成します。このプロシージャは、依存オブジェクトの**登録**も実施します。依存オブジェクトを登録することで、これらのオブジェクトの個別情報とコピーされた複製を、再定義完了プロセスの一部として後で自動的にスワップできます。その結果、再定義が完了すると、依存オブジェクトの名前がオリジナルの依存オブジェクトと同じ名前になります。

     詳細は、18-30 ページの「依存オブジェクトの自動作成」を参照してください。

   - 方法 2: 依存オブジェクトの手動による作成

     仮表に対する依存オブジェクトは、手動で作成して登録できます。詳細は、18-31 ページの「依存オブジェクトの手動による作成」を参照してください。

     注意： Oracle Database リリース 9*i* では、トリガー、索引、権限付与および制約を仮表に作成する際は、手動で行う必要がありました。現在でも、手動による作成が必要な場合や望ましい場合があります。このような場合、仮表に関与する参照制約（つまり、仮表は参照制約の親表または子表のどちらかです）は、すべて使用禁止で作成する必要があります。オンライン再定義が完了すると、参照制約が自動的に使用可能になります。また、再定義プロセスが完了または強制終了するまでは、仮表で定義されたトリガーは実行されません。

7. FINISH_REDEF_TABLE プロシージャを実行して、表の再定義を完了します。このプロシージャの実行中、元の表はそのデータ量とは無関係に、わずかな時間ですが排他モードでロックされます。ただし、FINISH_REDEF_TABLE 部分は、保留中の DML すべてがコミットされるのを待機してから、再定義を完了します。

8. 再定義に ROWID を使用したときに、COMPATIBLE 初期化パラメータが 10.1.0 以下に設定されている場合は、再定義後の表に追加された非表示列（M_ROW$$）を削除するか UNUSED に設定してください。

   ```
   ALTER TABLE table_name SET UNUSED (M_ROW$$);
   ```

   COMPATIBLE が 10.2.0 以上の場合は、再定義の完了時に非表示列が自動的に UNUSED に設定されます。次に、ALTER TABLE ... DROP UNUSED COLUMNS 文を使用して列を削除できます。

9. 仮表に対する長時間実行の問合せがある場合は、完了するのを待ってから、仮表を削除します。

   仮表に対するアクティブな問合せの実行中に仮表を削除すると、ORA-08103 エラー（「現在、指定したオブジェクトは存在しません。」）が発生する場合があります。

関連項目： 18-35 ページ「表のオンライン再定義の例」

列マッピング文字列の作成

引数として START_REDEF_TABLE に渡す列マッピング文字列には、カンマで区切られた列マッピングのペアのリストが含まれています。各ペアの構文は、次のとおりです。

```
[expression]  column_name
```

column_name は、仮表の列を意味します。オプションの *expression* には、SQL（SELECT）文の式のルールに従って、再定義する表の列、定数、演算子、関数またはメソッド・コールなどを指定できます。ただし、使用できるのは、値がすぐに決定される単純な副次式、つまり、ある評価と次の評価で結果が変化しない副次式と、順序および SYSDATE のみです。副問合せは使用できません。最も簡単な場合、式は再定義する表の列名のみで構成されます。

式を指定すると、その値は再定義の過程で仮表内の指定の列に配置されます。式を省略した場合は、再定義する表と仮表の両方に *column_name* という列が存在し、再定義する表にあるその列の値が仮表の同じ列に配置されていると想定されます。

たとえば、再定義する表の override 列を override_commission という名前に変更し、すべてのオーバーライド・コミッションを 2% 増加する場合、正しい列マッピングのペアは次のとおりです。

```
override*1.02  override_commission
```

列マッピング文字列に '*' または NULL を指定すると、すべての列（名前は変更されない）が仮表に配置されることになります。それ以外の場合は、文字列で明示的に指定した列のみが仮表に配置されます。列マッピングのペアの順序は重要ではありません。

列マッピング文字列の例は、18-35 ページの「表のオンライン再定義の例」を参照してください。

データの変換　列をマッピングする際は、いくつかの制限はありますが、データ型を変換できます。

'*' または NULL を列マッピング文字列として指定した場合は、SQL で許可される暗黙的な変換のみがサポートされます。たとえば、CHAR から VARCHAR2 に、INTEGER から NUMBER に変換できます。

あるオブジェクト型から別のオブジェクト型への変換や、あるコレクション型から別のコレクション型への変換など、その他のデータ型変換を実行する場合は、変換を実行する式とともに列マッピングのペアを指定する必要があります。式には、CAST 関数、TO_NUMBER などの組込み関数、作成した変換関数などを指定できます。

依存オブジェクトの自動作成

仮表に対する依存オブジェクトを自動的に作成するには、COPY_TABLE_DEPENDENTS プロシージャを使用します。

num_errors 出力引数をチェックすることで、依存オブジェクトのコピー中にエラーが発生したかどうかを検出できます。ignore_errors 引数を TRUE に設定すると、COPY_TABLE_DEPENDENTS プロシージャは、オブジェクト作成時にエラーを検出しても、依存オブジェクトのコピーを続行します。DBA_REDEFINITION_ERRORS ビューを問い合せることで、これらのエラーを確認できます。

エラーには、次のような理由があります。

- システム・リソースの不足
- 依存オブジェクトの再コーディングが必要となるような表の論理構造の変更

  この種のエラーについては、18-35 ページの「表のオンライン再定義の例」の例 3 を参照してください。

ignore_errors を FALSE に設定すると、COPY_TABLE_DEPENDENTS プロシージャは、エラーを検出すると、オブジェクトのコピーをただちに停止します。

エラーを修正してから COPY_TABLE_DEPENDENTS プロシージャを再実行することで、依存オブジェクトのコピーを再試行できます。「依存オブジェクトの手動による作成」に説明されているように、オブジェクトを手動で作成し、それらを登録することもできます。

COPY_TABLE_DEPENDENTS プロシージャは、必要に応じて何回でも使用できます。オブジェクトがすでに正常にコピーされている場合は、再度コピーされません。

依存オブジェクトの手動による作成

SQL*Plus または Enterprise Manager で、仮表に対する依存オブジェクトを手動で作成する場合は、REGISTER_DEPENDENT_OBJECT プロシージャを使用して依存オブジェクトを登録する必要があります。依存オブジェクトを登録すると、再定義の完了プロセスで、依存オブジェクト名を再定義前の名前にリストアできます。

COPY_TABLE_DEPENDENTS プロシージャによる依存オブジェクトのコピーがエラーとなり、手動による介入が必要な場合は、REGISTER_DEPENDENT_OBJECT プロシージャを使用します。

DBA_REDEFINITION_OBJECTS ビューを問い合せることによって、登録されている依存オブジェクトを判断できます。このビューには、REGISTER_DEPENDENT_OBJECT プロシージャで明示的に登録、または COPY_TABLE_DEPENDENTS プロシージャで暗黙的に登録された依存オブジェクトが表示されます。このビューには、現在の情報のみが表示されます。

UNREGISTER_DEPENDENT_OBJECT プロシージャを使用すると、再定義している表および仮表に対する依存オブジェクトの登録を解除できます。

注意： 手動で作成する依存オブジェクトは、対応する元の依存オブジェクトと同一である必要はありません。たとえば、マテリアライズド・ビュー・ログを仮表に手動で作成する場合は、別の列を記録できます。また、仮表の依存オブジェクトが増減してもかまいません。

再定義プロセスの結果

再定義プロセスの最終的な結果は、次のようになります。

- 元の表は、仮表の列、索引、制約、権限付与、トリガーおよび統計を使用して再定義されます。
- REGISTER_DEPENDENT_OBJECT を明示的に使用するか、または COPY_TABLE_DEPENDENTS を暗黙的に使用して登録された依存オブジェクトは、自動的に名前が変更されるため、再定義した表の依存オブジェクト名は再定義の前と同じになります。

  注意： 登録または自動コピーが行われていない依存オブジェクトの名前は、手動で変更する必要があります。

- 仮表に関与する参照制約は、再定義した表に関与して使用可能になります。
- （再定義前に）元の表に定義されていた索引、トリガー、マテリアライズド・ビュー・ログ、権限付与および制約がすべて仮表に移動し、ユーザーが仮表を削除したときに同時に削除されます。再定義する前に元の表に関与していた参照制約がすべて仮表に関与し、使用禁止になります。
- 一部の PL/SQL オブジェクト、ビュー、シノニムおよびその他の表依存オブジェクトが、無効になる場合があります。変更された表の要素に依存するオブジェクトのみが無効になります。たとえば、再定義で変更されなかった再定義表の列のみを問い合せる PL/SQL プロシージャは有効のままです。スキーマ・オブジェクトの依存性の詳細は、『Oracle Database 概要』を参照してください。

中間での同期化の実行

START_REDEF_TABLE をコールして再定義プロセスを開始してから FINISH_REDEF_TABLE コールが完了するまでの間に、元の表に対して多数の DML 文が実行される可能性があります。これが問題になることがわかっている場合は、定期的に仮表を元の表と同期化することをお薦めします。同期化には、SYNC_INTERIM_TABLE プロシージャをコールします。このプロシージャをコールすると、FINISH_REDEF_TABLE で再定義プロセスを完了するための時間が短縮されます。SYNC_INTERIM_TABLE をコールできる回数に制限はありません。

FINISH_REDEF_TABLE の実行中に元の表がロックされるわずかな時間は、SYNC_INTERIM_TABLE のコールの有無とは関係ありません。

エラー後の表のオンライン再定義の強制終了およびクリーン・アップ

再定義プロセス中にエラーが発生した場合、または再定義プロセスの終了を選択した場合は、ABORT_REDEF_TABLE をコールしてください。このプロシージャは、再定義プロセスに対応付けられた一時ログおよび一時表を削除します。このプロシージャをコールした後は、仮表とその依存オブジェクトを削除できます。

オンライン再定義プロセスの再起動が必要な場合は、最初に ABORT_REDEF_TABLE をコールしないと、表を再定義する後続の試みでエラーが発生します。

表のオンライン再定義に関する制限事項

表のオンライン再定義には、次の制限が適用されます。

- 表の再定義に主キーまたは疑似主キー（NOT NULL 制約が指定されているすべての構成要素の列を備えた一意のキー）を使用する場合は、再定義した表に同じ主キー列または疑似主キーが必要です。ROWID を使用して再定義する表に、索引構成表を含めないでください。
- マテリアライズド・ビュー・ログが含まれている表を再定義した後に依存マテリアライズド・ビューをリフレッシュする場合は、完全リフレッシュを実行する必要があります。
- n-way マスター構成でレプリケートされた表の再定義は可能ですが、水平サブセット化（表内の行のサブセット）、垂直サブセット化（表内の列のサブセット）または列変換は使用できません。
- 索引構成表のオーバーフロー表は、個別にオンライン再定義できません。
- ファイングレイン・アクセス・コントロール（行レベルのセキュリティ）が設定された表はオンライン再定義できません。
- BFILE 列を持つ表は、オンライン再定義できません。
- 複数の LONG 列を保持している表は、オンラインで再定義できますが、これらの列は、CLOB に変換する必要があります。また、LONG RAW 列は、BLOB に変換する必要があります。LOB 列を持つ表は、オンライン再定義可能です。
- パラレル実行のためのリソースが十分なシステムで、仮表がパーティション化されていない環境では、次の場合にのみ、LONG 列から LOB 列への再定義をパラレルで実行できます。
  - 仮表への LOB 列の格納に使用するセグメントが、自動セグメント領域管理（ASSM）を使用できるローカル管理表領域に属している場合。
  - 単一の LONG 列から単一の LOB 列への簡単なマッピングで、仮表に存在する LOB 列が 1 つのみの場合。

  仮表がパーティション化されている場合は、パラレル実行でパーティション化する通常の方法が適用されます。
- SYS および SYSTEM スキーマ内の表は、オンライン再定義できません。
- 一時表は再定義できません。
- 表内の行のサブセットは再定義できません。

- 仮表の列を元の表の列にマッピングするときに使用できるのは、値がすぐに決定される単純な式、順序および SYSDATE のみです。たとえば、副問合せは使用できません。
- 新しい列を再定義の一部として追加しようとして、それらの列に列マッピングがない場合は、その再定義が完了するまで NOT NULL を宣言しないでください。
- 再定義しようとする表と仮表の間では参照制約を作成できません。
- 表の再定義は、NOLOGGING モードでは実行できません。
- マテリアライズド・ビュー・ログおよびキュー表の場合、オンライン再定義は物理的なプロパティの変更に制限されます。水平サブセット化または垂直サブセット化が使用できず、列の変換もできません。列マッピング文字列に唯一有効な値は NULL です。
- VARRAY は、列マッピングに CAST 演算子を使用してネストした表に変換できます。ただし、ネストした表を VARRAY に変換することはできません。

単一パーティションのオンライン再定義

Oracle Database 10*g* リリース 2 からは、表の単一パーティションをオンラインで再定義できます。これは、異なる表領域にパーティションを移動する際に、移動中でもパーティションに対して DML を使用できるようにする場合などに便利です。

この機能の別の用途は、表全体を再定義する際に、リソース要件を低減するために 1 度に 1 つのパーティションずつオンラインで再定義することです。たとえば、異なる表領域に非常に大きな表を移動する場合は、表を 1 度に 1 つのパーティションずつ移動することで、移動を完了するために必要な空き領域と UNDO 領域を最小化できます。ただし、単一パーティションを再定義するときに、グローバル索引がある場合は、再定義が完了したときに UNUSABLE のマークが設定されることに注意してください。

単一パーティションの再定義は、次の点で表の再定義とは異なります。

- 依存オブジェクトをコピーする必要はありません。COPY_TABLE_DEPENDENTS プロシージャは、単一パーティションの再定義には使用できません。
- 仮表に対してローカル索引を手動で作成する必要があります。
- START_REDEF_TABLE には、NULL の列マッピング文字列が必要です。
- ROWID による方法を使用するときには、再定義の最終フェーズで、非表示列 M_ROW$$ が未使用に設定されるかわりに削除されます。

注意： パーティションを別の表領域に移動する際に、DML でそのパーティションを使用する必要性がない場合は、より簡単な ALTER TABLE MOVE PARTITION コマンドを使用できます。

関連項目：

- 『Oracle Database VLDB およびパーティショニング・ガイド』のパーティションの移動に関する項
- 『Oracle Database SQL リファレンス』

単一パーティションのオンライン再定義のルール

単一パーティションを再定義するための基本的な仕組みは、データベースの**パーティション交換**機能（ALTER TABLE...EXCHANGE PARTITION）です。したがって、単一パーティションのオンライン定義のルールと制限事項は、この仕組みに基づいて決まります。一般的には、次の制限事項があります。

- 論理的な変更（列の追加や削除など）は許可されません。
- パーティション化する方法の変更（レンジ・パーティション化からハッシュ・パーティション化への変更など）は許可されません。
- グローバル索引がある場合は、表のパーティションの再定義が完了したときに UNUSABLE のマークが設定されます。

仮表を定義する際のルールは、次のとおりです。

- 再定義するパーティションが、レンジ、ハッシュまたはリスト・パーティションである場合は、非パーティションの仮表が必要です。
- 再定義するパーティションがレンジ - ハッシュ・コンポジット・パーティション表のレンジ・パーティションである場合は、ハッシュ・パーティション表の仮表が必要です。また、仮表のパーティション化キーは、レンジ - ハッシュ・パーティション表のサブパーティション化キーと同一であり、仮表のパーティション数は、再定義するレンジ・パーティションのサブパーティション数と同一であることが必要です。
- 再定義するパーティションがレンジ - リスト・コンポジット・パーティション表のレンジ・パーティションである場合は、リスト・パーティション表の仮表が必要です。また、仮表のパーティション化キーは、レンジ - リスト・パーティション表のサブパーティション化キーと同一であり、仮表のリスト・パーティションの値リストは、再定義するレンジ・パーティションのリスト・サブパーティションの値リストと正確に一致している必要があります。
- 仮表を圧縮表として定義する場合は、次のいずれかを実行します。
  - ROWID による再定義ではなく、キーによる再定義を使用します。
  - 行 ID による方法を使用する必要がある場合は、COMPRESS FOR ALL OPERATIONS として仮表を定義します。

次の補足ルールは、再定義する表がパーティション化された索引構成表である場合に適用されます。

- 仮表も索引構成されている必要があります。
- 元の表と仮表では、同じ列に対する主キーが同じ順序で保持されている必要があります。
- キー圧縮が使用可能な場合は、元の表と仮表の両方で、同じ接頭辞の長さでキー圧縮が使用可能になっている必要があります。
- オーバーフロー・セグメントがある場合は、元の表と仮表の両方にあるか、または両方にないことが必要です。マッピング表についても同様です。
- 元の表と仮表の両方で、LOB 列に対して同じ記憶域属性が必要です。

> **関連項目：**『Oracle Database VLDB およびパーティショニング・ガイド』のパーティションの交換に関する項

表のオンライン再定義の例

次の例で使用されているすべての DBMS_REDEFINITION サブプログラムに関する詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

例	説明
例 1	新しい列を追加しパーティションを追加することで、表を再定義します。
例 2	オブジェクト・データ型を使用して表を再定義します。
例 3	手動で登録した依存オブジェクトを使用して表を再定義します。
例 4	単一の表パーティションを異なる表領域に移動して表を再定義します。

例 1

この例は、以前に作成した表 hr.admin_emp のオンライン再定義を示しています。この表の列は、この時点では empno、ename、job、deptno のみです。表を次のように再定義します。

- 新しい列 mgr、hiredate、sal および bonus を追加します（これらの列は元の表に存在していましたが、前述の例で削除されました）。
- 新しい列 bonus を 0 に初期化します。
- 列 deptno の値を 10 増やしています。
- 再定義された表を empno の範囲でパーティション化します。

この再定義の手順は、次のとおりです。

1. 表がオンライン再定義の候補であることを確認します。この場合は、主キーまたは疑似主キーを使用して再定義が実行されるように指定します。

```
BEGIN
DBMS_REDEFINITION.CAN_REDEF_TABLE('hr','admin_emp',
      DBMS_REDEFINITION.CONS_USE_PK);
END;
/
```

2. 仮表 hr.int_admin_emp を作成します。

```
CREATE TABLE hr.int_admin_emp
        (empno      NUMBER(5) PRIMARY KEY,
         ename      VARCHAR2(15) NOT NULL,
         job        VARCHAR2(10),
         mgr        NUMBER(5),
         hiredate   DATE DEFAULT (sysdate),
         sal        NUMBER(7,2),
         deptno     NUMBER(3) NOT NULL,
         bonus      NUMBER (7,2) DEFAULT(1000))
     PARTITION BY RANGE(empno)
       (PARTITION emp1000 VALUES LESS THAN (1000) TABLESPACE admin_tbs,
        PARTITION emp2000 VALUES LESS THAN (2000) TABLESPACE admin_tbs2);
```

3. 再定義プロセスを開始します。

```
BEGIN
DBMS_REDEFINITION.START_REDEF_TABLE('hr', 'admin_emp','int_admin_emp',
       'empno empno, ename ename, job job, deptno+10 deptno, 0 bonus',
        dbms_redefinition.cons_use_pk);
END;
/
```

4. 依存オブジェクトをコピーします（hr.int_admin_emp に対するトリガー、索引、マテリアライズド・ビュー・ログ、権限付与および制約がある場合、それらは自動的に作成されます）。

```
DECLARE
num_errors PLS_INTEGER;
BEGIN
DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS('hr', 'admin_emp','int_admin_emp',
   DBMS_REDEFINITION.CONS_ORIG_PARAMS, TRUE, TRUE, TRUE, TRUE, num_errors);
END;
```

このコールでは、ignore_errors 引数が TRUE に設定されていることに注意してください。これは、仮表が主キー制約付きで作成されており、COPY_TABLE_DEPENDENTS によって、主キー制約と索引が元の表からコピーされる際にエラーが発生するためです。これらのエラーは無視できますが、後続の手順に記載されている問合せを実行して、他のエラーの存在を確認する必要があります。

5. DBA_REDEFINITION_ERRORS ビューを問い合せて、エラーをチェックします。

```
SQL> select object_name, base_table_name, ddl_txt from
        DBA_REDEFINITION_ERRORS;

OBJECT_NAME   BASE_TABLE_NAME  DDL_TXT
------------- ---------------- ------------------------------
SYS_C005836   ADMIN_EMP        CREATE UNIQUE INDEX "HR"."TMP$
                               $_SYS_C0058360" ON "HR"."INT_A
                               DMIN_EMP" ("EMPNO")

SYS_C005836   ADMIN_EMP        ALTER TABLE "HR"."INT_ADMIN_EM
                               P" ADD CONSTRAINT "TMP$$_SYS_C
                               0058360" PRIMARY KEY
```

これらのエラーは、仮表にある既存の主キー制約に起因しているため、無視できます。このアプローチでは、再定義後の表の主キー制約名と索引名が変更されていることに注意してください。別のアプローチを使用すると、エラーの発生と名前の変更を回避できますが、仮表は主キー制約なしで定義されることになります。この例の場合、主キー制約と索引は元の表からコピーされます。

注意： 最良のアプローチは、主キー制約付きで仮表を定義し、REGISTER_DEPENDENT_OBJECT を使用して主キー制約と索引を登録してから、COPY_TABLE_DEPENDENTS で残りの依存オブジェクトをコピーすることです。このアプローチでは、エラーが回避され、再定義した表には常に主キーがあり、依存オブジェクト名も変わりません。

---

6. 必要に応じて、仮表 hr.int_admin_emp を同期化します。

```
BEGIN
DBMS_REDEFINITION.SYNC_INTERIM_TABLE('hr', 'admin_emp', 'int_admin_emp');
END;
/
```

7. 再定義を完了します。

```
BEGIN
DBMS_REDEFINITION.FINISH_REDEF_TABLE('hr', 'admin_emp', 'int_admin_emp');
END;
/
```

この手順が終了するまでに、わずかな間のみ、表 hr.admin_emp が排他モードでロックされます。このコールの後、表 hr.admin_emp は hr.int_admin_emp 表のすべての属性を持つように再定義されます。

8. 仮表に対する長時間実行の問合せがある場合は、完了するのを待ってから、仮表を削除します。

例 2

この例では、列をオブジェクト属性に変更するために表を再定義します。再定義した表にオブジェクト型の新しい列を確保します。

元の表（CUSTOMER）の定義は、次のとおりです。

```
Name         Type
------------ -------------
CID          NUMBER            <- Primary key
NAME         VARCHAR2(30)
STREET       VARCHAR2(100)
CITY         VARCHAR2(30)
STATE        VARCHAR2(2)
ZIP          NUMBER(5)
```

新しいオブジェクトの型定義は、次のとおりです。

```
CREATE TYPE ADDR_T AS OBJECT (
   street VARCHAR2(100),
   city VARCHAR2(30),
   state VARCHAR2(2),
   zip NUMBER(5, 0) );
```

再定義の手順は、次のとおりです。

1. 表がオンライン再定義の候補であることを確認します。主キーまたは疑似主キーを使用して再定義が実行されるように指定します。

```
BEGIN
DBMS_REDEFINITION.CAN_REDEF_TABLE('STEVE','CUSTOMER',
       DBMS_REDEFINITION.CONS_USE_PK);
END;
/
```

2. 仮表 int_customer を作成します。

```
CREATE TABLE INT_CUSTOMER(
  CID NUMBER,
  NAME  VARCHAR2(30),
  ADDR  ADDR_T);
```

仮表には主キーが定義されていないことに注意してください。手順 5 で依存オブジェクトがコピーされると、主キー制約と索引がコピーされます。

3. CUSTOMER は大きい表であるため、後続の手順のためにパラレル操作を指定します。

```
alter session force parallel dml parallel 4;
alter session force parallel query parallel 4;
```

4. 主キーを使用して再定義プロセスを開始します。

```
BEGIN
DBMS_REDEFINITION.START_REDEF_TABLE(
   uname       => 'STEVE',
   orig_table  => 'CUSTOMER',
   int_table   => 'INT_CUSTOMER',
   col_mapping => 'cid cid,  name name,
      addr_t(street, city, state, zip) addr');
END;
/
```

addr_t(street, city, state, zip) は、オブジェクト・コンストラクタへのコールです。

5. 依存オブジェクトをコピーします。

```
DECLARE
num_errors PLS_INTEGER;
BEGIN
DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
   'STEVE','CUSTOMER','INT_CUSTOMER',DBMS_REDEFINITION.CONS_ORIG_PARAMS,
    TRUE, TRUE, TRUE, FALSE, num_errors, TRUE);
END;
/
```

このコールの最後の引数は、表の統計が仮表にコピーされることを意味します。

6. 必要に応じて、仮表を同期化します。

```
BEGIN
DBMS_REDEFINITION.SYNC_INTERIM_TABLE('STEVE', 'CUSTOMER', 'INT_CUSTOMER');
END;
/
```

7. 再定義を完了します。

```
BEGIN
DBMS_REDEFINITION.FINISH_REDEF_TABLE('STEVE', 'CUSTOMER', 'INT_CUSTOMER');
END;
/
```

8. 仮表に対する長時間実行の問合せがある場合は、完了するのを待ってから、仮表を削除します。

例 3

この例では、依存オブジェクトを手動で作成および登録する必要がある場合を考えてみます。

表 T1 には C1 という列があり、再定義した後、この列を C2 にするとします。C1 には索引 Index1 があると想定します。この場合、COPY_TABLE_DEPENDENTS は、Index1 に対応して、仮表に対する索引の作成を試行し、仮表には存在しない列 C1 に対して索引の作成を試行します。これは結果的にエラーとなります。したがって、列 C2 に対しては、索引を手動で作成して登録する必要があります。手順は、次のとおりです。

1. 仮表 INT_T1 を作成し、列 C2 に対して索引 Int_Index1 を作成します。
2. CAN_REDEF_TABLE を使用して T1 がオンライン定義の候補であることを確認し、次に START_REDEF_TABLE を使用して再定義プロセスを開始します。
3. 元（Index1）と仮（Int_Index1）の依存オブジェクトを登録します。

```
BEGIN
DBMS_REDEFINITION.REGISTER_DEPENDENT_OBJECT(
   uname         => 'STEVE',
   orig_table    => 'T1',
   int_table     => 'INT_T1',
   dep_type      => DBMS_REDEFINITION.CONS_INDEX,
   dep_owner     => 'STEVE',
   dep_orig_name => 'Index1',
   dep_int_name  => 'Int_Index1');
END;
/
```

4. COPY_TABLE_DEPENDENTS を使用して、残りの依存オブジェクトをコピーします。
5. 必要に応じて、仮表を同期化します。
6. 再定義を完了し、仮表を削除します。

例 4

この例では、単一のパーティションを再定義します。販売表というレンジ・パーティションの最も古いパーティションを、表領域 TBS_LOW_FREQ に移動します。再定義するパーティションが格納されている表の定義は、次のとおりです。

```
CREATE TABLE salestable
(s_productid NUMBER,
s_saledate DATE,
s_custid NUMBER,
s_totalprice NUMBER)
TABLESPACE users
PARTITION BY RANGE(s_saledate)
(PARTITION sal03q1 VALUES LESS THAN (TO_DATE('01-APR-2003', 'DD-MON-YYYY')),
PARTITION sal03q2 VALUES LESS THAN (TO_DATE('01-JUL-2003', 'DD-MON-YYYY')),
PARTITION sal03q3 VALUES LESS THAN (TO_DATE('01-OCT-2003', 'DD-MON-YYYY')),
PARTITION sal03q4 VALUES LESS THAN (TO_DATE('01-JAN-2004', 'DD-MON-YYYY')));
```

表には、次のように定義されたローカル・パーティション索引があります。

```
CREATE INDEX sales_index ON salestable
   (s_saledate, s_productid, s_custid) LOCAL;
```

手順は、次のとおりです。次のプロシージャ・コールでは、パーティション名（part_name）という特別な引数に注目してください。

1. salestable が再定義の候補であることを確認します。

```
BEGIN
DBMS_REDEFINITION.CAN_REDEF_TABLE(
   uname        => 'STEVE',
   tname        => 'SALESTABLE',
   options_flag => DBMS_REDEFINITION.CONS_USE_ROWID,
   part_name    => 'sal03q1');
END;
/
```

2. TBS_LOW_FREQ 表領域に仮表を作成します。これはレンジ・パーティションの再定義であるため、仮表は非パーティション表です。

```
CREATE TABLE int_salestable
(s_productid NUMBER,
s_saledate DATE,
s_custid NUMBER,
s_totalprice NUMBER)
TABLESPACE tbs_low_freq;
```

3. ROWID を使用して再定義プロセスを開始します。

```
BEGIN
DBMS_REDEFINITION.START_REDEF_TABLE(
   uname        => 'STEVE',
   orig_table   => 'salestable',
   int_table    => 'int_salestable',
   col_mapping  => NULL,
   options_flag => DBMS_REDEFINITION.CONS_USE_ROWID,
   part_name    => 'sal03q1');
END;
/
```

4. 仮表に対してローカル索引を手動で作成します。

```
CREATE INDEX int_sales_index ON int_salestable
(s_saledate, s_productid, s_custid)
TABLESPACE tbs_low_freq;
```

5. 必要に応じて、仮表を同期化します。

```
BEGIN
DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
   uname      => 'STEVE',
   orig_table => 'salestable',
   int_table  => 'int_salestable',
   part_name  => 'sal03q1');
END;
/
```

6. 再定義を完了します。

```
BEGIN
DBMS_REDEFINITION.FINISH_REDEF_TABLE(
   uname      => 'STEVE',
   orig_table => 'salestable',
   int_table  => 'int_salestable',
   part_name  => 'sal03q1');
END;
/
```

7. 仮表に対する長時間実行の問合せがある場合は、完了するのを待ってから、仮表を削除します。

次の問合せは、最も古いパーティションが新しい表領域に移動したことを示します。

```
select partition_name, tablespace_name from user_tab_partitions
 where table_name = 'SALESTABLE';

PARTITION_NAME                 TABLESPACE_NAME
------------------------------ ------------------------------
SAL03Q1                        TBS_LOW_FREQ
SAL03Q2                        USERS
SAL03Q3                        USERS
SAL03Q4                        USERS

4 rows selected.
```

DBMS_REDEFINITION パッケージに必要な権限

DBMS_REDEFINITION パッケージの実行権限は、EXECUTE_CATALOG_ROLE に付与されます。実行ユーザーは、このパッケージの実行権限以外に、次の権限が付与されていることが必要です。

- CREATE ANY TABLE
- ALTER ANY TABLE
- DROP ANY TABLE
- LOCK ANY TABLE
- SELECT ANY TABLE

COPY_TABLE_DEPENDENTS を実行するには、次の追加権限が必要です。

- CREATE ANY TRIGGER
- CREATE ANY INDEX

エラーが発生した表の変更の調査と取消し

表に対してエラーが発生する変更を調査して取り消せるようにするために、Oracle Database には、データベース・オブジェクトの過去の状態を表示したり、Point-in-Time メディア・リカバリを使用せずにデータベース・オブジェクトを以前の状態に戻すために使用できる一連の機能が用意されています。これらの機能は **Oracle Flashback 機能**と呼ばれており、『Oracle Database アドバンスト・アプリケーション開発者ガイド』で説明されています。

エラーが発生する変更を調査するために、複数の Oracle Flashback 問合せを使用して、特定の時点における行データを表示できます。さらに効率的な方法として、Oracle Flashback Version Query を使用して、ある期間にわたる行への変更すべてを表示できます。この機能では、`SELECT` 文に `VERSIONS` 句を追加できるため、行の値への変更を表示するシステム変更番号（SCN）またはタイムスタンプの範囲を指定できます。この問合せでは、変更の原因となったトランザクションなど、関連するメタデータを返すこともできます。

エラーが発生するトランザクションを特定した後、Oracle Flashback Transaction Query を使用して、そのトランザクションで実行された他の変更を特定できます。次に、Oracle Flashback Transaction を使用して、エラーが発生するトランザクションを取り消すことができます。（Oracle Flashback Transaction では、依存するすべてのトランザクション、つまりエラーが発生するトランザクションと同じ行が関係する後続のトランザクションも取り消す必要があることに注意してください。）18-41 ページの「Oracle Flashback Table を使用した表のリカバリ」に説明されている Oracle Flashback Table も使用できます。

注意： Oracle Flashback 機能を使用するには、自動 UNDO 管理を使用している必要があります。14-2 ページの「自動 UNDO 管理の概念」を参照してください。

関連項目： Oracle Flashback 機能の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

Oracle Flashback Table を使用した表のリカバリ

Oracle Flashback Table では、以前の時点の状態に表をリストアできます。この文は、ユーザーやアプリケーションによって偶発的に変更または削除された表を迅速にリカバリするためのオンライン・ソリューションです。多くの場合、Oracle Flashback Table によって、複雑な Point-in-Time リカバリ操作を行う必要がなくなります。

Oracle Flashback Table:

- 指定された表のすべてのデータが、タイムスタンプまたは SCN で表された以前の時点にリストアされます。
- リストア操作はオンラインで実行されます。
- アプリケーションがフラッシュバックされた表を使用して機能するために必要な索引、トリガー、制約など、表の属性すべてが自動的に維持されます。
- 分散環境におけるすべてのリモート状態が維持されます。たとえば、レプリケート表がフラッシュバックされる場合は、レプリケーションに必要な表の変更すべてが維持されます。
- 制約によって指定されているデータの整合性が維持されます。表は、表の制約すべてに違反していないことを条件にしてフラッシュバックされます。この制約には、`FLASHBACK TABLE` 文の対象になっている表と `FLASHBACK TABLE` 文の対象になっていない表との間に指定されている参照整合性制約も含まれます。
- フラッシュバック操作後も、元の表のデータは消失しません。後で、元の状態に戻すことができます。

注意： Oracle Flashback Table を使用するには、自動 UNDO 管理を使用している必要があります。14-2 ページの「自動 UNDO 管理の概念」を参照してください。

関連項目： FLASHBACK TABLE 文の詳細は、『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』を参照してください。

表の削除

不要になった表を削除するには、DROP TABLE 文を使用します。削除する表は、自分のスキーマに含まれているか、または DROP ANY TABLE システム権限を持っている必要があります。

注意： 表を削除する前に、表を削除した結果についてよく理解しておいてください。

- 表を削除すると、その表定義はデータ・ディクショナリから削除されます。その結果、表のすべての行はアクセスできなくなります。
- 表に対応付けられている索引とトリガーは、すべて削除されます。
- 削除した表に依存しているビューと PL/SQL プログラム・ユニットはすべてそのまま残りますが、無効になります（使用できません）。データベースによる依存性管理の詳細は、16-17 ページの「オブジェクト依存性の管理」を参照してください。
- 削除する表のシノニムはすべてそのまま残りますが、使用するとエラーが返されます。
- 削除した表に割り当てられていたエクステントは表領域の空き領域にすべて戻され、新しいエクステントまたは新しいオブジェクトを必要とするその他のオブジェクトによって再利用されます。クラスタ化表に対応する行はすべて、そのクラスタのブロックから削除されます。クラスタ化表については、第 20 章「クラスタの管理」を参照してください。

次の文は、hr.int_admin_emp 表を削除します。

```
DROP TABLE hr.int_admin_emp;
```

削除する表に、他の表の外部キーが参照している主キーまたは一意キーが含まれていて、その子表の FOREIGN KEY 制約を削除する場合は、次のように DROP TABLE 文に CASCADE 句を指定します。

```
DROP TABLE hr.admin_emp CASCADE CONSTRAINTS;
```

表を削除した場合、通常、その表に関連付けられている領域はすぐには解放されません。正確には、表の名前が変更されてリサイクル・ビンに配置されます。これによって、誤って表を削除したことが後で判明した場合は、FLASHBACK TABLE 文でリカバリできます。DROP TABLE 文の発行時点で、表に関連付けられている領域をただちに解放する場合は、次のように PURGE 句を指定します。

```
DROP TABLE hr.admin_emp PURGE;
```

表は、削除するかわりに切り捨てることができます。TRUNCATE 文を使用すると、表からすべての行を効率よく高速に削除できます。この操作は、切り捨てる表に対応付けられた構造（列定義、制約、トリガーなど）や認可には影響しません。TRUNCATE 文については、16-6 ページの「表とクラスタの切捨て」を参照してください。

フラッシュバック・ドロップの使用とリサイクル・ビンの管理

表を削除した場合、その表に関連付けられている領域はすぐには削除されません。表の名前が変更され、表および関連するオブジェクトがリサイクル・ビンに配置されます。これによって、誤って表が削除された場合に、後でその表をリカバリできます。この機能はフラッシュバック・ドロップと呼ばれます。表のリカバリには、FLASHBACK TABLE 文が使用されます。この目的のために FLASHBACK TABLE 文の使用方法を説明する前に、リサイクル・ビンの機能とその内容の管理方法を理解することが重要です。

この項の内容は、次のとおりです。

リサイクル・ビンの概要

リサイクル・ビンとは、実際には、削除されたオブジェクトに関する情報を含んでいるデータ・ディクショナリ表です。削除された表および関連するオブジェクト（索引、制約、ネストした表など）は、削除されずにそのまま領域を使用します。この領域は、リサイクル・ビンから明確にパージされるまで、または、あまり可能性はありませんが、表領域の制約のためにデータベースによるパージが必要となるまでは、ユーザー領域の割当てにとって不利です。

ユーザーに SYSDBA 権限がない場合、リサイクル・ビンの中でユーザーが所有するオブジェクトは、アクセス権があるオブジェクトのみであるため、各ユーザーには各自のリサイクル・ビンがあるとみなすことができます。リサイクル・ビンにある各自のオブジェクトは、次の文を使用して表示できます。

```
SELECT * FROM RECYCLEBIN;
```

表領域をその内容も含めて削除すると、表領域内のオブジェクトはリサイクル・ビンに配置されず、その表領域に配置されていたオブジェクトに対するリサイクル・ビン内のエントリはすべてパージされます。内容を含まない表領域を削除した場合、つまり空の表領域を削除した場合も、表領域内のオブジェクトに対するリサイクル・ビン内のエントリがすべてパージされます。同様に、それぞれの削除操作は次のように処理されます。

- ユーザーを削除すると、そのユーザーが所有しているオブジェクトはリサイクル・ビンには配置されず、リサイクル・ビン内のオブジェクトがすべてパージされます。
- クラスタを削除すると、そのメンバー表はリサイクル・ビンには配置されず、リサイクル・ビン内の古いメンバー表がすべてパージされます。
- タイプを削除すると、サブタイプなどの依存オブジェクトはリサイクル・ビンには配置されず、リサイクル・ビン内の古い依存オブジェクトがすべてパージされます。

リサイクル・ビン内のオブジェクト名の変更

削除された表をリサイクル・ビンに移動すると、その表とその表に関連するオブジェクトには、システムで生成された名前が割り当てられます。名前の変更は、複数の表が同じ名前の場合に発生する可能性がある、名前の競合を回避するために必要です。名前の変更は、次の状況で発生します。

- ユーザーが表を削除し、同じ名前で表を作成し、その後、作成した表を再度削除した場合。
- 2 人のユーザーが同じ名前の表を持ち、両方のユーザーが各自の表を削除した場合。

名前変更の表記規則は、次のとおりです。

```
BIN$unique_id$version
```

各項目の意味は次のとおりです。

- *unique_id* は、このオブジェクトに対する、26 文字からなるグローバルに一意の識別子です。これによって、リサイクル・ビンの名前がすべてのデータベース全体で一意に識別されます。
- *version* は、データベースによって割り当てられるバージョン番号です。

リサイクル・ビンの有効化と無効化

リサイクル・ビンは、recyclebin 初期化パラメータを使用して有効化および無効化できます。リサイクル・ビンが有効化されていると、削除した表とその依存オブジェクトはリサイクル・ビンに配置されます。リサイクル・ビンが無効化されていると、削除した表とその依存オブジェクトはリサイクル・ビンには配置されず、そのまま削除されるため、それらをリカバリするには他の手段を使用する必要があります（バックアップからのリカバリなど）。

リサイクル・ビンは、デフォルトで有効化されています。

リサイクル・ビンの無効化：

- 次のいずれかの文を発行します。

```
ALTER SESSION SET recyclebin = OFF;

ALTER SYSTEM SET recyclebin = OFF;
```

リサイクル・ビンの有効化：

- 次のいずれかの文を発行します。

```
ALTER SESSION SET recyclebin = ON;

ALTER SYSTEM SET recyclebin = ON;
```

ALTER SYSTEM または ALTER SESSION 文でリサイクル・ビンを有効化および無効化すると、ただちにその設定が有効になります。リサイクル・ビンの無効化はパージではありません。パージの場合は、すでにリサイクル・ビンにあるオブジェクトに影響を与えます。

他の初期化パラメータと同様に、テキスト形式の初期化ファイル init*SID*.ora の recyclebin パラメータには初期値を設定できます。

```
recyclebin=on
```

> **関連項目：** 初期化パラメータの詳細は、2-24 ページの「初期化パラメータと初期化パラメータ・ファイルの概要」を参照してください。

リサイクル・ビン内のオブジェクトの表示と問合せ

Oracle Database では、リサイクル・ビンのオブジェクトに関する情報を取得する 2 種類の方法を提供しています。

ビュー	説明
USER_RECYCLEBIN	ユーザーは、このビューを使用して、リサイクル・ビンにある削除した各自のオブジェクトを表示できます。このビューには、使いやすいシノニム RECYCLEBIN があります。
DBA_RECYCLEBIN	管理者はこのビューを使用して、リサイクル・ビンにある削除されたすべてのオブジェクトを表示できます。

これらのビューの使用目的の 1 つは、次の例のように、削除したオブジェクトに対してデータベースが割り当てた名前を識別することにあります。

```
SELECT object_name, original_name FROM dba_recyclebin
   WHERE owner = 'HR';

OBJECT_NAME                    ORIGINAL_NAME
------------------------------ --------------------------------
BIN$yrMKlZaLMhfgNAgAIMenRA==$0 EMPLOYEES
```

リサイクル・ビンの内容は、SQL*Plus の SHOW RECYCLEBIN コマンドを使用して表示することもできます。

```
SQL> show recyclebin

ORIGINAL NAME    RECYCLEBIN NAME                OBJECT TYPE  DROP TIME
---------------- ------------------------------ ------------ -------------------
EMPLOYEES        BIN$yrMKlZaVMhfgNAgAIMenRA==$0 TABLE        2003-10-27:14:00:19
```

リサイクル・ビンにあるオブジェクトは、他のオブジェクトと同じ要領で問い合せることができます。ただし、オブジェクトの名前は、リサイクル・ビンの中で識別されているとおりに指定する必要があります。次に例を示します。

```
SELECT * FROM "BIN$yrMKlZaVMhfgNAgAIMenRA==$0";
```

リサイクル・ビン内のオブジェクトのパージ

リサイクル・ビンから項目をリストアすることはないと判断した場合は、PURGE 文を使用して、項目および関連するオブジェクトをリサイクル・ビンから削除し、記憶域を解放できます。実行するには、項目を削除する場合と同じ権限が必要です。

PURGE 文を使用して表をパージする場合は、リサイクル・ビンの中で表に割り当てられている名前または元の表の名前を使用できます。リサイクル・ビンの名前は、DBA_RECYCLEBIN または USER_RECYCLEBIN ビューから取得できます。詳細は、18-45 ページの「リサイクル・ビン内のオブジェクトの表示と問合せ」を参照してください。次の例では、
表 hr.int_admin_emp をパージします。BIN$jsleilx392mk2=293$0 は、リサイクル・ビンに配置される際に変更された名前です。

```
PURGE TABLE BIN$jsleilx392mk2=293$0;
```

次の文を使用しても同様の結果となります。

```
PURGE TABLE int_admin_emp;
```

PURGE 文を使用すると、指定の表領域からリサイクル・ビンのすべてのオブジェクトをパージ、または指定のユーザーに属する表領域オブジェクトのみをパージできます。次に例を示します。

```
PURGE TABLESPACE example;
PURGE TABLESPACE example USER oe;
```

次の文を使用することで、ユーザーは独自のオブジェクトのリサイクル・ビンをパージして、オブジェクトの領域を解放できます。

```
PURGE RECYCLEBIN;
```

SYSDBA 権限がある場合は、前述の文の RECYCLEBIN のかわりに、DBA_RECYCLEBIN を指定することによって、リサイクル・ビン全体をパージできます。

また、PURGE 文を使用して、リサイクル・ビンから索引をパージ、またはリサイクル・ビンから指定の表領域にあるすべてのオブジェクトをパージすることもできます。

> **関連項目：** PURGE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

リサイクル・ビンからの表のリストア

FLASHBACK TABLE ...TO BEFORE DROP 文を使用して、リサイクル・ビンからオブジェクトをリカバリできます。リサイクル・ビンでの表の名前、または元の表の名前を指定できます。オプションの RENAME TO 句を使用すると、リカバリ時に表の名前を変更できます。リサイクル・ビンの名前は、DBA_RECYCLEBIN または USER_RECYCLEBIN ビューから取得できます。詳細は、18-45 ページの「リサイクル・ビン内のオブジェクトの表示と問合せ」を参照してください。FLASHBACK TABLE ...TO BEFORE DROP 文を使用するには、表を削除する場合と同じ権限が必要です。

次の例は、int_admin_emp 表をリストアし、その表に新しい名前を割り当てます。

```
FLASHBACK TABLE int_admin_emp TO BEFORE DROP
   RENAME TO int2_admin_emp;
```

システム生成のリサイクル・ビン名は、表を複数回削除した場合に大変便利です。たとえば、リサイクル・ビン内の int2_admin_emp 表に 3 つのバージョンがあり、2 つ目のバージョンをリカバリするとします。次の例のように、2 つの FLASHBACK TABLE 文を発行することでリカバリできます。あるいは、リサイクル・ビンを問い合せて、適切なシステム生成の名前にフラッシュバックできます。問合せに作成時間を含めると、正しい表をリストアしていることを確認できます。

```
SELECT object_name, original_name, createtime FROM recyclebin;

OBJECT_NAME                    ORIGINAL_NAME   CREATETIME
------------------------------ --------------- -------------------
BIN$yrMKlZaLMhfgNAgAIMenRA==$0 INT2_ADMIN_EMP  2006-02-05:21:05:52
BIN$yrMKlZaVMhfgNAgAIMenRA==$0 INT2_ADMIN_EMP  2006-02-05:21:25:13
BIN$yrMKlZaQMhfgNAgAIMenRA==$0 INT2_ADMIN_EMP  2006-02-05:22:05:53

FLASHBACK TABLE BIN$yrMKlZaVMhfgNAgAIMenRA==$0 TO BEFORE DROP;
```

依存オブジェクトのリストア

リサイクル・ビンから表をリストアすると、索引などの依存オブジェクトは元の名前が復元されず、システム生成のリサイクル・ビンの名前のままになります。元の名前をリストアする場合は、依存オブジェクトの名前を手動で変更する必要があります。依存オブジェクトの元の名前を手動でリストアする場合は、表をリストアする前に、各依存オブジェクトのシステム生成のリサイクル・ビン名を記録する必要があります。

次の例では、HR サンプル・スキーマから、削除した表 JOB_HISTORY の索引の一部の元の名前をリストアします。この例では、HR ユーザーとしてログインしていることを想定しています。

1. JOB_HISTORY の削除後、リサイクル・ビンからリストアする前に、次の問合せを実行します。

```
SELECT OBJECT_NAME, ORIGINAL_NAME, TYPE FROM RECYCLEBIN;

OBJECT_NAME                    ORIGINAL_NAME             TYPE
------------------------------ ------------------------- ---------
BIN$DBo9UChtZSbgQFeMiAdCcQ==$0 JHIST_JOB_IX              INDEX
BIN$DBo9UChuZSbgQFeMiAdCcQ==$0 JHIST_EMPLOYEE_IX         INDEX
BIN$DBo9UChvZSbgQFeMiAdCcQ==$0 JHIST_DEPARTMENT_IX       INDEX
BIN$DBo9UChwZSbgQFeMiAdCcQ==$0 JHIST_EMP_ID_ST_DATE_PK   INDEX
BIN$DBo9UChxZSbgQFeMiAdCcQ==$0 JOB_HISTORY               TABLE
```

2. 次のコマンドを実行して表をリストアします。

```
FLASHBACK TABLE JOB_HISTORY TO BEFORE DROP;
```

3. 次の問合せを実行して、すべての JOB_HISTORY 索引がシステム生成のリサイクル・ビン名を保持していることを確認します。

```
SELECT INDEX_NAME FROM USER_INDEXES WHERE TABLE_NAME = 'JOB_HISTORY';

INDEX_NAME
------------------------------
BIN$DBo9UChwZSbgQFeMiAdCcQ==$0
BIN$DBo9UChtZSbgQFeMiAdCcQ==$0
BIN$DBo9UChuZSbgQFeMiAdCcQ==$0
BIN$DBo9UChvZSbgQFeMiAdCcQ==$0
```

4. 次のようにして、最初の 2 つの索引の元の名前をリストアします。

```
ALTER INDEX "BIN$DBo9UChtZSbgQFeMiAdCcQ==$0" RENAME TO JHIST_JOB_IX;
ALTER INDEX "BIN$DBo9UChuZSbgQFeMiAdCcQ==$0" RENAME TO JHIST_EMPLOYEE_IX;
```

システム生成の名前は、二重引用符で囲む必要があります。

索引構成表の管理

ここでは、索引構成表の管理について説明します。この項の内容は、次のとおりです。

索引構成表の概要

索引構成表は、プライマリ B ツリーの異形である記憶域編成を持っています。順序付けされていないコレクション（ヒープ）としてデータを格納する通常の（ヒープ構成）表とは異なり、索引構成表のデータは B ツリーの索引構造に主キー・ソート方式で格納されます。索引構造の各リーフ・ブロックには、キー列と非キー列の両方が格納されます。

索引構成表の構造には、次の利点があります。

- 索引のみのスキャンで十分なため、主キーに対して高速にランダム・アクセスできます。また、索引構造以外に表記憶域がないため、新しい行の追加、行の更新、行の削除などにより表データを変更すると、索引構造の更新のみが実行されます。
- 行が主キー順にクラスタ化されているため、主キーに対して高速にレンジ・アクセスできます。
- 主キーの複製が回避されるため、記憶域の所要量を低く抑えられます。ヒープ構成表の場合、主キーは索引と基礎となる表の両方には格納されません。

索引構成表は、すべての表機能を備えています。制約、トリガー、LOB 列とオブジェクト列、パーティション化、パラレル操作、オンライン再編成、およびレプリケーションなどの機能をサポートします。さらに、次の機能も提供します。

- キー圧縮
- オーバーフロー記憶域と固有の列配置
- ビットマップ索引を含めた 2 次索引

高速な主キー・アクセスと高可用性を必要とする OLTP アプリケーションには、索引構成表が理想的です。たとえば、電子注文処理に使用される注文表の問合せおよび DML は大部分が主キー・ベースであるため、大量のボリュームが断片化の原因となり、再編成が頻繁に必要となります。索引構成表は、2 次索引を無効化せずにオンラインで再編成できるため、ウィンドウの使用を制限される時間が大幅に短縮または排除されます。

索引構成表は、アプリケーション固有の索引構造をモデル化するのに適しています。たとえば、テキスト、イメージおよびオーディオ・データを含むコンテンツ・ベースの情報検索アプリケーションには、索引構成表を使用して有効にモデル化できる逆索引が必要です。インターネット検索エンジンの基本の構成要素は、索引構成表を使用してモデル化できる逆向きの索引です。

これらは、索引構成表のアプリケーションのほんの数例です。

関連項目：

- 索引構成表に関する詳細は、『Oracle Database 概要』を参照してください。
- 索引構成表のパーティション化の詳細は、『Oracle Database VLDB およびパーティショニング・ガイド』を参照してください。

索引構成表の作成

索引構成表を作成するには、CREATE TABLE 文を使用します。ただし、追加情報を指定する必要があります。

- ORGANIZATION INDEX 修飾子。これによって、索引構成表であることを示します。
- 主キー。主キーは、単一列主キーの場合は列制約句、複数列主キーの場合は表制約句によって指定します。

必要に応じて、次の情報を指定できます。

- OVERFLOW 句。この句は、非キー列の一部を別のオーバーフロー・データ・セグメントに格納できるようにすることにより、B ツリー索引の稠密なクラスタを保ちます。
- PCTTHRESHOLD 値。この値には、オーバーフロー・セグメントの使用時に、索引ブロックに格納される部分の行の最大サイズをブロック・サイズの比率として定義します。行の最大サイズを超える行や列は、オーバーフロー・セグメントに格納されます。行は、列の境界で先頭と後尾の 2 つの断片に分けられます。先頭の断片は指定したしきい値に収まり、索引のリーフ・ブロック内にキーとともに格納されます。後尾の断片は、1 つ以上の行断片としてオーバーフロー領域に格納されます。したがって、索引エントリには、キー値、指定したしきい値に収まる非キー列値および行の残りの部分へのポインタが含まれています。
- INCLUDING 句。この句は、主キーとともに索引ブロックに格納される非キー列を指定するために使用できます。

例 : 索引構成表の作成

次の文によって、索引構成表が作成されます。

```
CREATE TABLE admin_docindex(
        token char(20),
        doc_id NUMBER,
        token_frequency NUMBER,
        token_offsets VARCHAR2(2000),
        CONSTRAINT pk_admin_docindex PRIMARY KEY (token, doc_id))
    ORGANIZATION INDEX
    TABLESPACE admin_tbs
    PCTTHRESHOLD 20
    OVERFLOW TABLESPACE admin_tbs2;
```

この例では、token 列と doc_id 列で構成される主キーを使用して、admin_docindex という索引構成表を作成します。OVERFLOW 句と PCTTHRESHOLD 句では、行の長さが索引ブロック・サイズの 20% を超えた場合に、そのしきい値を超えた列とその後のすべての列がオーバーフロー・セグメントに移動されるように指定しています。オーバーフロー・セグメントは、admin_tbs2 表領域に格納されます。

> **関連項目 :** 索引構成表を作成する構文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

索引構成表に対する制限

索引構成表の作成には、次の制限があります。

- 列の最大数は 1000 です。
- キー列と非キー列を含めて、行の索引部分の最大列数は 255 です。255 個を超える列が必要な場合は、オーバーフロー・セグメントを使用する必要があります。
- 主キーに含めることができる最大列数は 32 です。
- PCTTHRESHOLD は 1 ～ 50 の範囲にする必要があります。デフォルトは 50 です。
- すべてのキー列は、指定したしきい値内に収まる必要があります。
- 行の最大サイズが索引ブロック・サイズの 50% を超える場合に、オーバーフロー・セグメントを指定しないと、CREATE TABLE 文が失敗します。
- 索引構成表に仮想列は設定できません。

オブジェクト型を含む索引構成表の作成

索引構成表は、オブジェクト型を格納できます。次の例は、オブジェクト型 admin_typ を作成し、オブジェクト型 admin_typ の列を含む索引構成表を作成しています。

```
CREATE OR REPLACE TYPE admin_typ AS OBJECT
    (col1 NUMBER, col2 VARCHAR2(6));
CREATE TABLE admin_iot (c1 NUMBER primary key, c2 admin_typ)
    ORGANIZATION INDEX;
```

オブジェクト型の索引構成表を作成することもできます。次に例を示します。

```
CREATE TABLE admin_iot2 OF admin_typ (col1 PRIMARY KEY)
    ORGANIZATION INDEX;
```

次に、索引構成表がネストした表を効率的に格納する例を示します。ネストした表の列ごとに、ネストした表のすべての行を保持する記憶表が内部的に作成されます。

```
CREATE TYPE project_t AS OBJECT(pno NUMBER, pname VARCHAR2(80));
/
CREATE TYPE project_set AS TABLE OF project_t;
/
CREATE TABLE proj_tab (eno NUMBER, projects PROJECT_SET)
    NESTED TABLE projects STORE AS emp_project_tab
                ((PRIMARY KEY(nested_table_id, pno))
    ORGANIZATION INDEX)
    RETURN AS LOCATOR;
```

ネストした表のシングル・インスタンスに属する行は、nested_table_id 列で識別されます。ネストした表の列を格納するために通常の表が使用される場合、ネストした表の行は、一般的にクラスタ化が解除されます。ただし、索引構成表を使用する場合、ネストした表は nested_table_id 列に基づいてクラスタ化できます。

関連項目：

- 索引構成表を作成する構文の詳細は、『Oracle Database SQL リファレンス』を参照してください。
- パーティション化された索引構成表の作成方法は、『Oracle Database VLDB およびパーティショニング・ガイド』を参照してください。
- オブジェクト型の詳細は、『Oracle Database オブジェクト・リレーショナル開発者ガイド』を参照してください。

しきい値の選択と監視

キー列とともに最初のいくつかの非キー列が頻繁にアクセスされる場合は、その非キー列を取り込めるしきい値を選択してください。

しきい値を選択した後、指定した値が適切な値であることを確認するために、表を監視できます。ANALYZE TABLE ...LIST CHAINED ROWS 文を使用して、しきい値を超える行の数と、どの行がしきい値を超えているかを判断できます。

関連項目：

- 連鎖行の詳細は、16-5 ページの「表とクラスタの連鎖行のリスト」を参照してください。
- ANALYZE 文の構文は、『Oracle Database SQL リファレンス』を参照してください。

INCLUDING 句の使用

PCTTHRESHOLD を指定する以外に、INCLUDING 句を使用して、キー列とともに格納する非キー列を制御できます。データベースでは、索引リーフ・ブロック内に INCLUDING 句で指定した列を含めてその列までのすべての非キー列を取り込むことができます。ただし、その列が指定したしきい値を超えない場合にかぎります。INCLUDING 句で指定した列より後のすべての非キー列は、オーバーフロー・セグメントに格納されます。INCLUDING 句と PCTTHRESHOLD 句が競合する場合は、PCTTHRESHOLD 句が優先されます。

注意： Oracle Database では、主キー・ベースのアクセス効率を高めるために、索引構成表のすべての主キー列が、表の先頭に（キー順に）移動されます。次に例を示します。

```
CREATE TABLE admin_iot4(a INT, b INT, c INT, d INT,
                primary key(c,b))
    ORGANIZATION INDEX;
```

格納後の列順は、a b c d ではなく、c b a d となります。格納された列順に基づき、最後の主キー列は b になります。INCLUDING 列には、最後の主キー列（この例では b）と非キー列（つまり、格納後の列順で b の後の任意の列）のどちらでも指定できます。

次の CREATE TABLE 文は、18-49 ページの「例：索引構成表の作成」で示した文と類似していますが、token_offsets 列の値が常にオーバーフロー領域に格納される索引構成表を作成するように変更されています。

```
CREATE TABLE admin_docindex2(
        token CHAR(20),
        doc_id NUMBER,
        token_frequency NUMBER,
        token_offsets VARCHAR2(2000),
        CONSTRAINT pk_admin_docindex2 PRIMARY KEY (token, doc_id))
    ORGANIZATION INDEX
    TABLESPACE admin_tbs
    PCTTHRESHOLD 20
    INCLUDING token_frequency
    OVERFLOW TABLESPACE admin_tbs2;
```

この例では、索引リーフ・ブロック内のキー列値とともに、token_offsets までの非キー列のみ（この場合は 1 つの列のみ）が格納されます。

索引構成表作成のパラレル化

CREATE TABLE...AS SELECT 文を使用すると、索引構成表を作成して、既存の表からその索引構成表にデータをロードできます。PARALLEL 句を指定することによって、ロードをパラレルで実行できます。

次の文は、従来型の表 hr.jobs から行を選択し、索引構成表をパラレルに作成します。

```
CREATE TABLE admin_iot3(i PRIMARY KEY, j, k, l)
     ORGANIZATION INDEX
     PARALLEL
     AS SELECT * FROM hr.jobs;
```

この文によって、SQL*Loader を使用するパラレル・バルク・ロードの代替手段が提供されます。

キー圧縮の使用

キー圧縮を使用して索引構成表を作成すると、キー列の接頭辞が同じ値で繰り返し格納されるのを避けることができます。

キー圧縮によって、索引キーは接頭辞および接尾辞エントリに分割されます。圧縮するために、接頭辞エントリは索引ブロック内のすべての接尾辞エントリ間で共有されます。このような共有によって、領域が大幅に節約され、各索引ブロックに格納できるキー数が増え、パフォーマンスが向上します。

キー圧縮を使用可能にするには、次の操作を行う際に COMPRESS 句を使用します。

- 索引構成表の作成
- 索引構成表の移動

また、接頭辞の長さをキー列の数で指定できます。これにより、キー列が接頭辞および接尾辞エントリにどのように分割されるかが決まります。

```
CREATE TABLE admin_iot5(i INT, j INT, k INT, l INT, PRIMARY KEY (i, j, k))
    ORGANIZATION INDEX COMPRESS;
```

この文は、次の文と等価です。

```
CREATE TABLE admin_iot6(i INT, j INT, k INT, l INT, PRIMARY KEY(i, j, k))
    ORGANIZATION INDEX COMPRESS 2;
```

値リスト (1,2,3)、(1,2,4)、(1,2,7)、(1,3,5)、(1,3,4)、(1,4,4) では、(1,2)、(1,3) の反復的な発生が圧縮されます。

また、次のように、圧縮に使用されるデフォルトの接頭辞の長さを変更することもできます。

```
CREATE TABLE admin_iot7(i INT, j INT, k INT, l INT, PRIMARY KEY (i, j, k))
    ORGANIZATION INDEX COMPRESS 1;
```

値リスト (1,2,3)、(1,2,4)、(1,2,7)、(1,3,5)、(1,3,4)、(1,4,4) では、1 の反復的な発生が圧縮されます。

圧縮は、次のように使用禁止にすることができます。

```
ALTER TABLE admin_iot5 MOVE NOCOMPRESS;
```

キー圧縮のアプリケーションは、株価など、単一の項目に属して一連のタイムスタンプを表す行を使用する時系列のアプリケーションで使用されます。索引構成表には、主キーに従って行をクラスタ化する機能があるため、このようなアプリケーションには効果的です。索引構成表を主キー（株式銘柄、タイムスタンプ）で定義することによって、時系列データを効率的に格納および操作できます。キー圧縮を採用した索引構成表を使用することによって、項目識別子（株式銘柄など）の反復的な発生を圧縮して、記憶域を大幅に節約できます。

> **関連項目：** キー圧縮の詳細は、『Oracle Database 概要』を参照してください。

索引構成表のメンテナンス

索引構成表と通常の表の相違点は、物理的な構成のみです。論理的には、通常の表と同じように操作されます。INSERT、SELECT、DELETE および UPDATE の各文では、通常の表を指定する場合と同じように、索引構成表を指定できます。

索引構成表の変更

通常の表に使用可能な変更オプションはすべて索引構成表にも使用できます。使用可能なオプションには、ADD、MODIFY、DROP COLUMNS および CONSTRAINTS があります。ただし、索引構成表の主キー制約は、削除、遅延または使用禁止にできません。

ALTER TABLE 文を使用すると、主キー索引セグメントとオーバーフロー・データ・セグメントの物理属性と記憶域属性を変更できます。OVERFLOW キーワードより前に指定したすべての属性は、主キー索引セグメントに適用できます。OVERFLOW キーワードより後に指定したすべての属性は、オーバーフロー・データ・セグメントに適用できます。たとえば、次のようにして、主キー索引セグメントの INITRANS を 4 に、オーバーフロー・データ・セグメントの INITRANS を 6 に設定できます。

```
ALTER TABLE admin_docindex INITRANS 4 OVERFLOW INITRANS 6;
```

また、PCTTHRESHOLD および INCLUDING 列の値も変更できます。後続の操作では、新しい設定を使用して、先頭部分とオーバーフローの後尾の部分に行が分割されます。たとえば、admin_docindex 表の PCTHRESHOLD および INCLUDING 列の値を次のように変更できます。

```
ALTER TABLE admin_docindex PCTTHRESHOLD 15 INCLUDING doc_id;
```

INCLUDING 列を doc_id に設定すると、その後のすべての列、つまり token_frequency および token_offsets はオーバーフロー・データ・セグメントに格納されます。

オーバーフロー・データ・セグメントなしで作成された索引構成表の場合は、ADD OVERFLOW 句を使用してオーバーフロー・データ・セグメントを追加できます。たとえば、次のように表 admin_iot3 にオーバーフロー・セグメントを追加できます。

```
ALTER TABLE admin_iot3 ADD OVERFLOW TABLESPACE admin_tbs2;
```

索引構成表の移動（再作成）

索引構成表は主として B ツリー索引に格納されるため、増分更新の結果として断片化が生じることがあります。ただし、このような断片化は、ALTER TABLE...MOVE 文を使用して索引を再作成することで低減できます。

次の文は、索引構成表 admin_docindex を再作成します。

```
ALTER TABLE admin_docindex MOVE;
```

ONLINE キーワードを使用して、索引構成表をオンラインで再作成できます。OVERFLOW キーワードを指定すると、オーバーフロー・データ・セグメントが存在する場合はそれが再作成されます。たとえば、admin_docindex 表を再作成し、オーバーフロー・データ・セグメントを再作成しない場合は、次のようにオンラインで移動します。

```
ALTER TABLE admin_docindex MOVE ONLINE;
```

admin_docindex 表とオーバーフロー・データ・セグメントを再作成するには、次の文のように移動操作を実行します。この文は、表とオーバーフロー・データ・セグメントを新しい表領域に移動する方法も示しています。

```
ALTER TABLE admin_docindex MOVE TABLESPACE admin_tbs2
    OVERFLOW TABLESPACE admin_tbs3;
```

次の最後の文で、LOB 列（CLOB）を持つ索引構成表が作成されます。その後、この表は LOB 索引とともに移動し、データ・セグメントが再作成され新しい表領域に移動します。

```
CREATE TABLE admin_iot_lob
   (c1 number (6) primary key,
    admin_lob CLOB)
   ORGANIZATION INDEX
   LOB (admin_lob) STORE AS (TABLESPACE admin_tbs2);
.
.
.
ALTER TABLE admin_iot_lob MOVE LOB (admin_lob) STORE AS (TABLESPACE admin_tbs3);
```

関連項目： 索引構成表の LOB の詳細は、『Oracle Database SecureFiles およびラージ・オブジェクト開発者ガイド』を参照してください。

索引構成表に対する 2 次索引の作成

索引構成表に 2 次索引を作成することで、複数のアクセス・パスを提供できます。索引構成表の 2 次索引は、2 つの点で通常の表の索引とは異なります。

- 索引構成表の 2 次索引には、物理的な ROWID ではなく、論理的な ROWID が格納されます。論理的な ROWID の格納が必要な理由は、B ツリー索引の行にある本来の可動性のために、その行に永続的な物理アドレスがないためです。列の物理的な位置が変化しても、その論理的な ROWID は有効です。この効果の 1 つは、ALTER TABLE ...MOVE などの表のメンテナンス操作によって 2 次索引が使用禁止状態にならないことです。
- 論理 ROWID にも、列が見つかる可能性があるデータベースのブロック・アドレスを識別する物理的な不確定要素が含まれています。物理的な不確定要素が正しい場合、2 次キーが見つかると、2 次索引スキャンによって単一の追加 I/O が生じます。パフォーマンスは、通常の表での 2 次索引スキャンの場合と同様です。

一意の 2 次索引、一意でない 2 次索引、機能ベースの 2 次索引およびビットマップ索引が、索引構成表の 2 次索引としてサポートされます。

1 つの索引構成表に対する 1 つの 2 次索引の作成

次の文は、索引構成表 docindex に 2 次索引を作成します、doc_id と token はキー列です。

```
CREATE INDEX Doc_id_index on Docindex(Doc_id, Token);
```

この 2 次索引によって、問合せ（次の文にある doc_id の述語に関係する問合せ）が効率的に処理されます。

```
SELECT Token FROM Docindex WHERE Doc_id = 1;
```

論理 ROWID の物理的不確定要素のメンテナンス

論理 ROWID には、不確定要素が作成される際に行のブロック位置を識別する不確定要素を含めることができます。完全なキー検索を実行するかわりに、不確定要素を使用してブロックが直接検索されます。ただし、新しい行が挿入されると、不確定要素は失効となる可能性があります。索引は論理 ROWID の主キー構成要素を介してそのまま使用できますが、行へのアクセスは遅くなります。

不確定要素の失効を監視するには、DBMS_STATS パッケージを使用して索引統計を収集します。既存の不確定要素が有効かどうかがチェックされ、有効な不確定要素を保持している行の割合がデータ・ディクショナリに記録されます。この統計は、DBA_INDEXES ビュー（および関連するビュー）の PCT_DIRECT_ACCESS 列に格納されます。

新しい不確定要素を取得するために、2 次索引を再作成できます。索引構成表に対する 2 次索引の再作成には、通常の表に対する索引の再作成とは異なり、実表の読込みが必要です。不確定要素を修正する迅速で手軽な方法は、ALTER INDEX ...UPDATE BLOCK REFERENCES 文を使用する方法です。この文はオンラインで実行されますが、DML は基礎となる索引構成表でそのまま実行できます。

2 次索引を再作成した後、あるいは不確定要素のブロック参照を更新した後は、索引統計を再度収集してください。

ビットマップ索引

索引構成表とともにマッピング表が作成される場合は、索引構成表でのビットマップ索引がサポートされます。ビットマップ索引を作成するには、索引構成表の作成に使用する CREATE TABLE 文、または後でマッピング表を追加する ALTER TABLE 文に、MAPPING TABLE 句を指定します。

> **関連項目：** マッピング表の説明は、『Oracle Database 概要』を参照してください。

索引構成表の分析

通常の表と同様に、索引構成表の分析には DBMS_STATS パッケージ、または ANALYZE 文を使用します。

索引構成表のオプティマイザ統計の収集

オプティマイザ統計を収集するには、DBMS_STATS パッケージを使用します。

たとえば、次の文は hr スキーマの索引構成表 countries について統計を収集します。

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS ('HR','COUNTRIES');
```

DBMS_STATS パッケージでは、主キー索引セグメントとオーバーフロー・データ・セグメントの両方が分析され、表の論理統計と物理統計が算出されます。

- 論理統計は、USER_TABLES、ALL_TABLES または DBA_TABLES を使用して問合せできます。
- 主キー索引セグメントの物理統計を問い合せるには、USER_INDEXES、ALL_INDEXES または DBA_INDEXES（および主キー索引名）を使用します。たとえば、表 admin_docindex の主キー索引セグメントの物理統計は、次のようにして取得できます。

  ```
  SELECT LAST_ANALYZED, BLEVEL,LEAF_BLOCKS, DISTINCT_KEYS
     FROM DBA_INDEXES WHERE INDEX_NAME= 'PK_ADMIN_DOCINDEX';
  ```

- オーバーフロー・データ・セグメントの物理統計を問い合せるには、USER_TABLES、ALL_TABLES または DBA_TABLES を使用します。IOT_TYPE = 'IOT_OVERFLOW' で検索すると、オーバーフロー・エントリを識別できます。たとえば、admin_docindex 表に対応付けられたオーバーフロー・データ・セグメントの物理属性は、次のようにして取得できます。

  ```
  SELECT LAST_ANALYZED, NUM_ROWS, BLOCKS, EMPTY_BLOCKS
     FROM DBA_TABLES WHERE IOT_TYPE='IOT_OVERFLOW'
           and IOT_NAME= 'ADMIN_DOCINDEX';
  ```

  関連項目：

  - オプティマイザ統計の収集の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
  - DBMS_STATS パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

索引構成表の構造の検証

索引構成表の構造を検証、または連鎖行をリストする場合は、ANALYZE 文を使用します。これらの操作は、このマニュアルの該当する項で説明されています。

- 16-4 ページ「表、索引、クラスタおよびマテリアライズド・ビューの妥当性チェック」
- 16-5 ページ「表とクラスタの連鎖行のリスト」

注意： 索引構成表の連鎖行をリストする場合は、特別な注意が必要です。詳細は、『Oracle Database SQL リファレンス』を参照してください。

索引構成表での ORDER BY 句の使用

ORDER BY 句が主キー列またはその接頭辞のみを参照する場合、行は主キー列でソートされた状態で返されるため、オプティマイザはソートのオーバーヘッドを回避します。

データはすでに主キーでソートされているので、次の 2 つの問合せはソートのオーバーヘッドを回避します。

```
SELECT * FROM admin_docindex2 ORDER BY token, doc_id;
SELECT * FROM admin_docindex2 ORDER BY token;
```

ただし、主キー列の接尾辞または非主キー列に ORDER BY 句がある場合は、別のソートが必要になります（他の 2 次索引が定義されていない場合）。

```
SELECT * FROM admin_docindex2 ORDER BY doc_id;
SELECT * FROM admin_docindex2 ORDER BY token_frequency;
```

索引構成表の標準的な表への変換

索引構成表を標準的な表に変換するには、Oracle のインポート / エクスポート・ユーティリティ、あるいは CREATE TABLE...AS SELECT 文を使用します。

索引構成表を標準的な表に変換する手順

- 従来型パスを使用して、索引構成表のデータをエクスポートします。
- 同じ定義で、標準的な表の定義を作成します。
- IGNORE=y（オブジェクト存在エラーを無視する）を指定して、索引構成表のデータをインポートします。

注意： 索引構成表を標準的な表に変換する前に、Oracle8 より古いバージョンのエクスポート・ユーティリティでは索引構成表をエクスポートできないことに注意してください。

関連項目： IMPORT および EXPORT ユーティリティの使用方法は、『Oracle Database ユーティリティ』を参照してください。

外部表の管理

Oracle Database では、外部表内のデータへの読取り専用アクセスが可能です。**外部表**はデータベース内に存在しない表として定義されており、アクセス・ドライバが提供されていればどのようなフォーマットにすることもできます。外部表を記述するメタデータを提供することで、外部表内のデータをあたかも標準的なデータベース表内に存在しているデータのように公開できます。外部データは、SQL を使用して直接およびパラレルに問合せできます。

外部表のデータは、選択、結合、ソートなどが行えます。外部表のビューやシノニムも作成できます。ただし、外部表に対して DML 操作（UPDATE、INSERT または DELETE）は実行できず、索引も作成できません。

外部表は、任意の SELECT 文の結果を Oracle Data Pump で使用できるように、オラクル社が開発したプラットフォームに依存しないフォーマットにアンロードするフレームワークも提供しています。

注意： 外部表の統計収集には、DBMS_STATS パッケージを使用できます。ANALYZE 文による外部表の統計収集はサポートされていません。

DBMS_STATS パッケージの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

外部表のメタデータは、CREATE TABLE...ORGANIZATION EXTERNAL 文を使用して定義します。外部表の定義は、外部データを最初にデータベースにロードしなくても外部データに対して任意の SQL 問合せを実行できるビューとみなすことができます。表内の外部データを読み込むために実際に使用されているメカニズムが、アクセス・ドライバです。外部表を使用してデータをアンロードすると、SELECT 文のデータ型（問合せの形式とも呼ばれます）に基づいてメタデータが自動的に作成されます。

Oracle Database では、外部表のための 2 種類のアクセス・ドライバを提供しています。デフォルトのアクセス・ドライバは ORACLE_LOADER で、Oracle のローダー・テクノロジを使用して外部ファイルからデータを読み込むことができます。ORACLE_LOADER アクセス・ドライバは、SQL*Loader ユーティリティの制御ファイル構文のサブセットであるデータ・マッピング機能を提供します。第 2 のアクセス・ドライバ ORACLE_DATAPUMP は、データをアンロード（つまり、データベースからデータを読み取り、1 つ以上の外部ファイルで表された外部表にそのデータを挿入）してから、データを Oracle Database に再ロードします。

Oracle Database の外部表機能は、データ・ウェアハウスで一般的な、抽出、変換およびロード（ETL）の基本タスクを実行する際に役立つ手段を提供します。

次の項では、外部表のためにサポートされているデータ定義言語（DDL）文について説明します。サポートされている DDL 文はここで説明しているもののみですが、これらの文の句がすべてサポートされているわけではありません。

- 外部表の作成
- 外部表の変更
- 外部表の削除
- 外部表のシステム権限およびオブジェクト権限

関連項目：

- 外部表、アクセス・ドライバおよびそのアクセス・パラメータの詳細は、『Oracle Database ユーティリティ』を参照してください。
- データ・ウェアハウス環境での外部表の使用方法は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。

外部表の作成

外部表は、CREATE TABLE 文の ORGANIZATION EXTERNAL 句を使用して作成します。実際には表が作成されるわけではなく、外部表にはエクステントが対応付けられません。そのかわりに、外部データへのアクセスを可能にするメタデータをデータ・ディクショナリに作成します。

注意： 外部表に仮想列は設定できません。

次の例では、外部表を作成してから、データをデータベース表にアップロードしています。あるいは、CREATE TABLE 文の AS *subquery* 句を指定し、外部表フレームワークを介してデータをアンロードできます。外部表のデータ・ポンプ・アンロードは、ORACLE_DATAPUMP アクセス・ドライバのみを使用できます。

例 : 外部表の作成とデータのロード

ファイル empxt1.dat には、次のサンプル・データが収められています。

```
360,Jane,Janus,ST_CLERK,121,17-MAY-2001,3000,0,50,jjanus
361,Mark,Jasper,SA_REP,145,17-MAY-2001,8000,.1,80,mjasper
362,Brenda,Starr,AD_ASST,200,17-MAY-2001,5500,0,10,bstarr
363,Alex,Alda,AC_MGR,145,17-MAY-2001,9000,.15,80,aalda
```

ファイル empxt2.dat には、次のサンプル・データが収められています。

```
401,Jesse,Cromwell,HR_REP,203,17-MAY-2001,7000,0,40,jcromwel
402,Abby,Applegate,IT_PROG,103,17-MAY-2001,9000,.2,60,aapplega
403,Carol,Cousins,AD_VP,100,17-MAY-2001,27000,.3,90,ccousins
404,John,Richardson,AC_ACCOUNT,205,17-MAY-2001,5000,0,110,jrichard
```

次の SQL 文の例は、スキーマ hr に外部表 admin_ext_employees を作成し、そのデータを hr.employees 表にロードします。

```
CONNECT  /  AS SYSDBA;
-- Set up directories and grant access to hr
CREATE OR REPLACE DIRECTORY admin_dat_dir
    AS '/flatfiles/data';
CREATE OR REPLACE DIRECTORY admin_log_dir
    AS '/flatfiles/log';
CREATE OR REPLACE DIRECTORY admin_bad_dir
    AS '/flatfiles/bad';
GRANT READ ON DIRECTORY admin_dat_dir TO hr;
GRANT WRITE ON DIRECTORY admin_log_dir TO hr;
GRANT WRITE ON DIRECTORY admin_bad_dir TO hr;
-- hr connects. Provide the user password (hr) when prompted.
CONNECT hr
-- create the external table
CREATE TABLE admin_ext_employees
                  (employee_id       NUMBER(4),
                   first_name        VARCHAR2(20),
                   last_name         VARCHAR2(25),
                   job_id            VARCHAR2(10),
                   manager_id        NUMBER(4),
                   hire_date         DATE,
                   salary            NUMBER(8,2),
                   commission_pct    NUMBER(2,2),
                   department_id     NUMBER(4),
                   email             VARCHAR2(25)
                  )
     ORGANIZATION EXTERNAL
     (
       TYPE ORACLE_LOADER
       DEFAULT DIRECTORY admin_dat_dir
       ACCESS PARAMETERS
```

```
       (
         records delimited by newline
         badfile admin_bad_dir:'empxt%a_%p.bad'
         logfile admin_log_dir:'empxt%a_%p.log'
         fields terminated by ','
         missing field values are null
         ( employee_id, first_name, last_name, job_id, manager_id,
           hire_date char date_format date mask "dd-mon-yyyy",
           salary, commission_pct, department_id, email
         )
       )
       LOCATION ('empxt1.dat', 'empxt2.dat')
     )
     PARALLEL
     REJECT LIMIT UNLIMITED;
-- enable parallel for loading (good if lots of data to load)
ALTER SESSION ENABLE PARALLEL DML;
-- load the data in hr employees table
INSERT INTO employees (employee_id, first_name, last_name, job_id, manager_id,
                       hire_date, salary, commission_pct, department_id, email)
            SELECT * FROM admin_ext_employees;
```

この例について、次の各段落で説明します。

この例で、最初の数行の文は、データソースを保存するオペレーティング・システム・ディレクトリ用のディレクトリ・オブジェクトと、アクセス・パラメータで指定される不良レコードやログ・ファイル用のディレクトリ・オブジェクトを作成します。また、必要に応じて READ または WRITE のディレクトリ・オブジェクト権限を付与する必要があります。

注意： ディレクトリ・オブジェクトまたは BFILE を作成する場合は、次の条件が満たされているかどうかを確認してください。

- オペレーティング・システム・ファイルが、シンボリック・リンクまたはハード・リンクでないこと。
- Oracle Database のディレクトリ・オブジェクトに指定されているオペレーティング・システムのディレクトリ・パスが、既存のオペレーティング・システムのディレクトリ・パスであること。
- ディレクトリ・オブジェクトに指定されているオペレーティング・システムのディレクトリ・パスの構成要素に、シンボリック・リンクが含まれていないこと。

TYPE 指定は、外部表のアクセス・ドライバを示します。アクセス・ドライバはデータベースの外部データを解析する API です。Oracle Database では、ORACLE_LOADER と ORACLE_DATAPUMP の 2 種類のアクセス・ドライバを提供しています。TYPE 指定を省略した場合は、ORACLE_LOADER がデフォルトのアクセス・ドライバになります。AS *subquery* 句を指定して、ある Oracle Database からデータをアンロードし、同一または異なる Oracle Database に再ロードする場合は、ORACLE_DATAPUMP アクセス・ドライバを指定する必要があります。

ACCESS PARAMETERS 句で指定するアクセス・パラメータは、データベースには不透明です。これらのアクセス・パラメータはアクセス・ドライバによって定義されるもので、データベースが外部表にアクセスするときにアクセス・ドライバに提供されます。ORACLE_LOADER アクセス・パラメータの詳細は、『Oracle Database ユーティリティ』を参照してください。

PARALLEL 句は、データソースに対するパラレル問合せを可能にします。パラレル化の最小単位はデフォルトではデータソースですが、データソース内部でのパラレル・アクセスは可能なかぎり実装されます。たとえば、PARALLEL=3 と指定すると、データソースに対して複数のパラレル実行サーバーを稼働しておくことができます。しかし、データソース内部でのパラレル・アクセスは、次の条件がすべて成り立つ場合にのみ、アクセス・ドライバによって提供されます。

- メディアが、データソース内部でのランダムな位置指定をサポートしている。
- レコード境界をランダムな位置から検索できる。
- データファイルが、複数のチャンクに分割することが適切なほど十分大きい。

注意： PARALLEL 句の指定は、大量のデータを扱うときのみ意味があります。データが大量でない場合には、PARALLEL 句を指定すると悪影響を及ぼす可能性が高いので、お薦めできません。

REJECT LIMIT 句は、外部データの問合せ中に発生する可能性のあるエラーの数に上限を設けないことを指定します。パラレル・アクセスの場合、この上限は各パラレル実行サーバーに個別に適用されます。たとえば、REJECT LIMIT を指定すると、各パラレル問合せプロセスで 10 個の拒否が許可されます。したがって、パラレル問合せに関して正確に規定される REJECT LIMIT の値は、0（ゼロ）および UNLIMITED のみです。

この例では、INSERT INTO TABLE 文によって外部データソースから Oracle Database SQL エンジンへのデータフローが生成され、そこでデータが処理されます。外部表ソースからのデータがアクセス・ドライバで解析されて外部表インタフェースに提供されると、外部データがその外部表現から Oracle Database の内部データ型に変換されます。

関連項目： 外部表を作成するための CREATE TABLE 文の構文の詳細、および句の使用に関する制限については、『Oracle Database SQL リファレンス』を参照してください。

外部表の変更

外部表の特性を変更するには、表 18-3 のいずれかの ALTER TABLE 句を使用します。これ以外の句は使用できません。

表 18-3 外部表の ALTER TABLE 句

ALTER TABLE 句	説明	例
REJECT LIMIT	拒否の上限を変更します。	ALTER TABLE admin_ext_employees REJECT LIMIT 100;
PROJECT COLUMN	アクセス・ドライバが後続の問合せで行の妥当性をチェックする方法を決定します。 ■ PROJECT COLUMN REFERENCED: アクセス・ドライバは、問合せの選択リストのみを処理します。この設定では、同一の外部表とは異なる列リストを問い合せたときに、一貫した行のセットが提供されない可能性があります。これはデフォルトです。 ■ PROJECT COLUMN ALL: アクセス・ドライバは、外部表に定義されているすべての列を処理します。この設定では、外部表を問い合せたときに、常に一貫した行のセットが提供されます。	ALTER TABLE admin_ext_employees PROJECT COLUMN REFERNCED; ALTER TABLE admin_ext_employees PROJECT COLUMN ALL;
DEFAULT DIRECTORY	デフォルトのディレクトリ指定を変更します。	ALTER TABLE admin_ext_employees DEFAULT DIRECTORY admin_dat2_dir;

表 18-3 外部表の ALTER TABLE 句（続き）

ALTER TABLE 句	説明	例
ACCESS PARAMETERS	外部表のメタデータの削除と再作成を行わずにアクセス・パラメータを変更できます。	`ALTER TABLE admin_ext_employees ACCESS PARAMETERS (FIELDS TERMINATED BY ';');`
LOCATION	外部表のメタデータの削除と再作成を行わずにデータソースを変更できます。	`ALTER TABLE admin_ext_employees LOCATION ('empxt3.txt', 'empxt4.txt');`
PARALLEL	標準的な表の場合と同じです。並列度を変更できます。	新しい構文はありません。
ADD COLUMN	標準的な表の場合と同じです。外部表に列を追加できます。仮想列は使用できません。	新しい構文はありません。
MODIFY COLUMN	標準的な表の場合と同じです。外部表の列を変更できます。仮想列は使用できません。	新しい構文はありません。
SET UNUSED	ALTER TABLE DROP COLUMN コマンドに透過的に変換されます。外部表はデータベース内でメタデータのみで構成されているため、DROP COLUMN コマンドは SET UNUSED コマンドの実行と同じことになります。	新しい構文はありません。
DROP COLUMN	標準的な表の場合と同じです。外部表の列を削除できます。	新しい構文はありません。
RENAME TO	標準的な表の場合と同じです。外部表の名前を変更できます。	新しい構文はありません。

外部表の削除

外部表では、DROP TABLE 文によってデータベース内の表メタデータのみ削除されます。実際のデータはデータベースの外側に存在しているため、影響はありません。

外部表のシステム権限およびオブジェクト権限

外部表のシステム権限およびオブジェクト権限は、標準的な表のサブセットになります。外部表に適用できるシステム権限は、次のものにかぎられます。

- CREATE ANY TABLE
- ALTER ANY TABLE
- DROP ANY TABLE
- SELECT ANY TABLE

外部表に適用できるオブジェクト権限は、次のものにかぎられます

- ALTER
- SELECT

ただし、ディレクトリには次のオブジェクト権限が対応付けられています。

- READ
- WRITE

外部表では、データソースのあるディレクトリ・オブジェクトに対して READ 権限が必要であり、同時に、不良ファイル、ログ・ファイルまたは廃棄ファイルのあるディレクトリ・オブジェクトに対して WRITE 権限が必要です。

表のデータ・ディクショナリ・ビュー

次のビューを使用して、表に関する情報にアクセスできます。

ビュー	説明
DBA_TABLES ALL_TABLES USER_TABLES	DBA ビューには、データベース内のすべてのリレーショナル表が表示されます。ALL ビューには、ユーザーがアクセス可能なすべての表が表示されます。USER ビューは、ユーザーが所有する表のみに制限されます。これらのビューの一部の列には、DBMS_STATS パッケージまたは ANALYZE 文によって生成される統計が含まれます。
DBA_TAB_COLUMNS ALL_TAB_COLUMNS USER_TAB_COLUMNS	これらのビューには、データベース内の表の列、ビューおよびクラスタが表示されます。これらのビューの一部の列には、DBMS_STATS パッケージまたは ANALYZE 文によって生成される統計が含まれます。
DBA_ALL_TABLES ALL_ALL_TABLES USER_ALL_TABLES	これらのビューには、データベース内のすべてのリレーショナル表およびオブジェクト表が表示されます。オブジェクト表については、このマニュアルでは詳しく説明していません。
DBA_TAB_COMMENTS ALL_TAB_COMMENTS USER_TAB_COMMENTS	これらのビューには、表およびビューのコメントが表示されます。コメントは、COMMENT 文を使用して入力します。
DBA_COL_COMMENTS ALL_COL_COMMENTS USER_COL_COMMENTS	これらのビューには、表およびビューの列のコメントが表示されます。コメントは、COMMENT 文を使用して入力します。
DBA_EXTERNAL_TABLES ALL_EXTERNAL_TABLES USER_EXTERNAL_TABLES	これらのビューには、データベースで定義されている外部表の特定の属性がリストされます。
DBA_EXTERNAL_LOCATIONS ALL_EXTERNAL_LOCATIONS USER_EXTERNAL_LOCATIONS	これらのビューには、外部表のデータソースがリストされます。
DBA_TAB_HISTOGRAMS ALL_TAB_HISTOGRAMS USER_TAB_HISTOGRAMS	これらのビューには、表およびビューに関するヒストグラムが表示されます。
DBA_TAB_STATISTICS ALL_TAB_STATISTICS USER_TAB_STATISTICS	これらのビューには、表のオプティマイザ統計が格納されます。
DBA_TAB_COL_STATISTICS ALL_TAB_COL_STATISTICS USER_TAB_COL_STATISTICS	これらのビューは、関連する TAB_COLUMNS ビューから抽出された列の統計およびヒストグラム情報を提供します。
DBA_TAB_MODIFICATIONS ALL_TAB_MODIFICATIONS USER_TAB_MODIFICATIONS	これらのビューには、表統計が最後に収集された時点以降変更された表が表示されます。これらのビューは即時には移入されず、ある程度の時間（通常は 3 時間）が経過した後に移入されます。
DBA_ENCRYPTED_COLUMNS USER_ENCRYPTED_COLUMNS ALL_ENCRYPTED_COLUMNS	これらのビューには、暗号化された表の列がリストされ、各列に使用している暗号化アルゴリズムがリストされます。

ビュー	説明
DBA_UNUSED_COL_TABS ALL_UNUSED_COL_TABS USER_UNUSED_COL_TABS	これらのビューには、ALTER TABLE ...SET UNUSED 文によって未使用のマークが付けられた列を持つ表がリストされます。
DBA_PARTIAL_DROP_TABS ALL_PARTIAL_DROP_TABS USER_PARTIAL_DROP_TABS	これらのビューには、DROP COLUMN 操作が一部完了している表がリストされます。これらの操作は、ユーザーによる中断やシステム障害が原因で不完全になることがあります。

例：列情報の表示

_COLUMNS 接尾辞で終わるビューのいずれかを使用すれば、名前、データ型、長さ、精度、位取り、デフォルト・データ値などの列情報を表示できます。たとえば、次の問合せは、emp 表と dept 表のデフォルトの列値をすべてリストします。

```
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE, DATA_LENGTH, LAST_ANALYZED
    FROM DBA_TAB_COLUMNS
    WHERE OWNER = 'HR'
    ORDER BY TABLE_NAME;
```

問合せの出力は次のとおりです。

```
TABLE_NAME           COLUMN_NAME          DATA_TYPE   DATA_LENGTH LAST_ANALYZED
-------------------- -------------------- ---------- ------------ -------------
COUNTRIES            COUNTRY_ID           CHAR                2 05-FEB-03
COUNTRIES            COUNTRY_NAME         VARCHAR2           40 05-FEB-03
COUNTRIES            REGION_ID            NUMBER             22 05-FEB-03
DEPARTMENTS          DEPARTMENT_ID        NUMBER             22 05-FEB-03
DEPARTMENTS          DEPARTMENT_NAME      VARCHAR2           30 05-FEB-03
DEPARTMENTS          MANAGER_ID           NUMBER             22 05-FEB-03
DEPARTMENTS          LOCATION_ID          NUMBER             22 05-FEB-03
EMPLOYEES            EMPLOYEE_ID          NUMBER             22 05-FEB-03
EMPLOYEES            FIRST_NAME           VARCHAR2           20 05-FEB-03
EMPLOYEES            LAST_NAME            VARCHAR2           25 05-FEB-03
EMPLOYEES            EMAIL                VARCHAR2           25 05-FEB-03
.
.
.
LOCATIONS            COUNTRY_ID           CHAR                2 05-FEB-03
REGIONS              REGION_ID            NUMBER             22 05-FEB-03
REGIONS              REGION_NAME          VARCHAR2           25 05-FEB-03

51 rows selected.
```

関連項目：

- これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。
- オブジェクト表の詳細は、『Oracle Database オブジェクト・リレーショナル開発者ガイド』を参照してください。
- ヒストグラムおよび表の統計生成の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- 16-3 ページ「表、索引およびクラスタの分析」

19

索引の管理

この章の内容は次のとおりです。

索引の概要

索引とは、表とクラスタに対応付けられるオプションの構造です。索引を使用することで、表に対して SQL 問合せを高速に実行できます。このマニュアルの索引によって情報を素早く検索できるのと同じように、Oracle Database の索引は表データへの高速なアクセス・パスを提供します。索引を使用するために、問合せを書きなおす必要はありません。索引を使用しなくても処理結果は同じですが、索引を使用するとより短時間で結果が表示されます。

Oracle Database は、補完的なパフォーマンス機能を持つ複数の索引付け方法を提供します。次の方法があります。

- B ツリー索引 : デフォルトの設定で、最も一般的です。
- B ツリー・クラスタ索引 : 特にクラスタ用に定義します。
- ハッシュ・クラスタ索引 : 特にハッシュ・クラスタ用に定義します。
- グローバル索引とローカル索引 : パーティション表とパーティション索引に関連します。
- 逆キー索引 : Oracle Real Application Clusters アプリケーションに最も役立ちます。
- ビットマップ索引 : サイズが小さいので、小さい値の集合を持つ列に効果的です。
- ファンクション索引 : 事前計算された関数や式の値を含みます。
- ドメイン索引 : アプリケーションまたはカートリッジに固有の索引です。

索引は、対応付けられた表内のデータから論理的にも物理的にも独立しています。索引は独立した構造体であり、記憶域を必要とします。索引は、実表、データベース・アプリケーションまたはその他の索引に影響を与えることなく、作成または削除できます。索引に対応する表に対して行の挿入、更新および削除が発生すると、データベースは索引を自動的にメンテナンスします。索引を削除しても、すべてのアプリケーションは引き続き動作可能です。ただし、それまで索引が付けられていたデータへのアクセスが遅くなります。

関連項目 :

- 索引の概要については、『Oracle Database 概要』を参照してください。
- 第 17 章「スキーマ・オブジェクトの領域の管理」

索引を管理するためのガイドライン

ここでは、索引管理のガイドラインについて説明します。この項の内容は、次のとおりです。

- 表データ挿入後の索引の作成
- 正しい表および列への索引付け
- パフォーマンスのための索引列の順序付け
- 表当たりの索引数の制限
- 不必要な索引の削除
- 索引サイズの見積りと記憶域パラメータの設定
- 各索引の表領域の指定
- 索引作成のパラレル化
- 索引作成時の NOLOGGING の使用
- 索引の結合と再作成に関するコストと利点の検討
- 制約を使用禁止または削除する前のコストの検討

関連項目：

- Oracle が提供する各種索引付け方法の説明など、索引と索引付けの概念に関する情報は、『Oracle Database 概要』を参照してください。
- ビットマップ索引の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』および『Oracle Database データ・ウェアハウス・ガイド』を参照してください。
- ドメイン固有のオペレータと索引付け方法の定義方法および Oracle Database サーバーへの統合方法については、『Oracle Database データ・カートリッジ開発者ガイド』を参照してください。

表データ挿入後の索引の作成

SQL*Loader またはインポート・ユーティリティを使用して、表にデータを挿入またはロードすることがあります。表の索引作成は、データを挿入またはロードした後のほうが効率的です。データをロードする前に索引を 1 つ以上作成すると、行が挿入されるたびにすべての索引の更新が必要になります。

すでにデータが格納されている表に索引を作成するには、ソート領域が必要です。索引の作成ユーザーに割り当てられているメモリーから確保されるソート領域もあります。各ユーザーのソート領域の大きさは、初期化パラメータ SORT_AREA_SIZE によって決まります。また、データベースでは、索引作成のときにのみユーザーの一時表領域に割り当てられる一時セグメントとの間でソート情報のスワップが行われます。

特定の条件下では、SQL*Loader のダイレクト・パス・ロードを使用してデータを表にロードし、データがロードされたときに索引が作成されるようにすることも可能です。

関連項目： SQL*Loader を使用したダイレクト・パス・ロードの詳細は、『Oracle Database ユーティリティ』を参照してください。

正しい表および列への索引付け

索引を作成するかどうか判断する際は、次のガイドラインを参考にしてください。

- 大きな表で頻繁に検索する行数が総行数の 15% 未満の場合に索引を作成します。この割合は、表をスキャンする相対速度と、索引キーに関して行データがどのように分散されているかによって大きく異なります。表のスキャンが速いほどこの割合は低くなります。また、行データがクラスタ化されているほど、この割合は高くなります。
- 複数の表を結合するパフォーマンスを改善するには、結合に使用する列に索引を付けます。

注意： 主キーおよび一意キーには自動的に索引が作成されますが、外部キーには必要に応じて索引を作成できます。

- 小さい表に索引は不要です。問合せに時間がかかる場合は、表のサイズが大きくなっていることが考えられます。

索引付けに適した列

列のタイプによっては、索引を付けるほうが望ましいものがあります。次のような特性を 1 つ以上持つ列には、索引の作成を検討してください。

- 一意の値が比較的多い。
- 値の範囲が広い（通常の索引が適している）。
- 値の範囲が狭い（ビットマップ索引が適している）。
- 列に多くの NULL が含まれているが、通常の問合せでは必ず値を持つ列を選択する。この場合は次の句を使用します。

```
WHERE COL_X > -9.99 * power(10,125)
```

この句は、次の句よりも適しています。

```
WHERE COL_X IS NOT NULL
```

これは、最初の句では COL_X の索引を使用しているためです（COL_X は数値列とします）。

索引付けに適さない列

次のような特性を持つ列は、索引付けには適していません。

- 列に NULL が多く含まれていて、NULL 以外の値を検索することがない。

LONG 列および LONG RAW 列には索引を作成できません。

仮想列

仮想列には、一意索引または非一意索引を作成できます。

パフォーマンスのための索引列の順序付け

CREATE INDEX 文の列の順序は、問合せのパフォーマンスに影響を与えます。一般的には、最も頻繁に使用する列を最初に指定します。

たとえば、col1、col2 および col3 の各列にアクセスする問合せを高速にするために、列にまたがる単一の索引を作成すると、col1 のみにアクセスする問合せ、または col1 と col2 にアクセスする問合せが速くなります。しかし、col2 のみにアクセスする問合せ、col3 のみにアクセスする問合せ、および col2 と col3 にアクセスする問合せは速くなりません。

表当たりの索引数の制限

表は、多数の索引を持つことができます。ただし、索引の数が多いほど、表を変更するときに発生するオーバーヘッドが増加します。特に、行を挿入したり削除したりするときは、その表の索引もすべて更新する必要があります。また、列を更新するときには、その列を含む索引もすべて更新する必要があります。

このように、表からデータを検索する速度とその表を更新する速度は二律背反的です。たとえば、表が主に読取り専用である場合、索引を増やすと有効ですが、表が頻繁に更新される場合は、索引を少なくすることをお薦めします。

不必要な索引の削除

次のような状況では、索引の削除を検討してください。

- 索引を使用しても問合せが速くならない場合。これには、たとえば表が非常に小さい場合や、表の行数は多いものの、索引エントリが非常に少ない場合などがあります。
- アプリケーションの問合せが索引を使用しない場合。
- 索引を再作成する前にいったん削除する必要がある場合。

関連項目： 19-15 ページ「索引の使用状況の監視」

索引サイズの見積りと記憶域パラメータの設定

索引を作成する前にそのサイズを見積っておくと、ディスク領域の計画と管理がいっそう容易になります。索引の見積りサイズの合計と、表、UNDO 表領域および REDO ログ・ファイルの見積りを使用して、作成するデータベースを格納するために必要なディスク容量を決定できます。この見積りを利用して適切なハードウェアを購入できます。

個々の索引の見積りサイズを使用することで、索引が使用するディスク領域をより適切に管理できます。索引を作成するときに、適切な記憶域パラメータを設定し、その索引を使用するアプリケーションの I/O パフォーマンスを改善できます。たとえば、索引を作成する前に索引の最大サイズを見積る場合を想定します。索引を作成するときに記憶域パラメータを設定すると、その表のデータ・セグメントに割り当てるエクステントを少なくできます。そのため、索引データすべてが比較的ディスク領域の連続した部分に格納されます。これによって、この索引に関係したディスク I/O 操作に要する時間が短くなります。

単一の索引エントリの最大サイズは、データ・ブロック・サイズのおよそ 2 分の 1 です。

索引を主キー制約または一意キー制約を規定するために使用する場合、その索引のために作成する索引セグメントの記憶域パラメータは、次のどちらかの方法で設定できます。

- `CREATE TABLE` 文または `ALTER TABLE` 文の `ENABLE ... USING INDEX` 句
- `ALTER INDEX` 文の `STORAGE` 句

各索引の表領域の指定

索引はどの表領域にも作成できます。索引は、その索引を付けた表と同じ表領域にも、異なる表領域にも作成できます。表とその索引に対して同じ表領域を使用すると、（表領域やファイルのバックアップなどの）データベースのメンテナンスやアプリケーションの可用性確保の面で便利です。これは、すべての関連するデータが常にまとまってオンラインになっているためです。

表とその索引に対して（異なるディスク上にある）異なる表領域を使用すると、表と索引を同じ表領域に格納するよりも、パフォーマンスが向上します。これは、ディスクの競合が解消されるためです。表とその索引に対して異なる表領域を使用し、一方の（データまたは索引のいずれかを含む）表領域がオフラインになっている場合には、その表を参照している文が動作する保証はありません。

索引作成のパラレル化

表作成をパラレル化できるのと同様に、索引の作成もパラレル化できます。複数のプロセスが同時に動作して索引を作成するため、1 つのサーバー・プロセスが順に索引を作成する場合よりも高速に索引を作成できます。

索引を並行して作成する場合、問合せサーバー・プロセスごとに別々の記憶域パラメータが使用されます。したがって、`INITIAL` 値を 5MB、並行度を 12 で索引を作成する場合は、作成時に 60MB 以上の記憶域を使用します。

関連項目：

- データ・ウェアハウス環境でのパラレル実行の使用方法は、『Oracle Database データ・ウェアハウス・ガイド』を参照してください。

索引作成時の NOLOGGING の使用

CREATE INDEX 文で NOLOGGING を指定すると、索引の作成時に最小限の REDO ログ・レコードしか生成されません。

> **注意：** NOLOGGING を使用して作成された索引はアーカイブされないため、索引作成後にバックアップを実行してください。

NOLOGGING を使用して索引を作成すると、次のような利点があります。

- REDO ログ・ファイルの領域を節約できます。
- 索引の作成に要する時間が削減できます。
- 大規模な索引のパラレル作成のパフォーマンスが向上します。

一般に、LOGGING を指定しないで索引を作成した場合、小規模な索引より大規模な索引のほうが相対的にパフォーマンスの向上が大きくなります。小規模な索引を LOGGING を指定しないで作成しても、索引作成に要する時間にはほとんど影響しません。一方、大規模な索引では、特に索引作成のパラレル化もあわせて指定したときに、パフォーマンスが著しく向上します。

索引の結合と再作成に関するコストと利点の検討

不適切な索引サイズの設定やサイズの拡大によって、索引の断片化が生じることがあります。断片化を解消または低減するには、索引を再作成するか、索引を結合します。ただし、どちらの作業を行う場合も、事前に各選択肢のコストと利点を分析し、状況に最も有効な方法を選択してください。表 19-1 は、索引を再作成する場合と結合する場合のコストと利点を示しています。

表 19-1 索引の結合と再作成に関するコストと利点

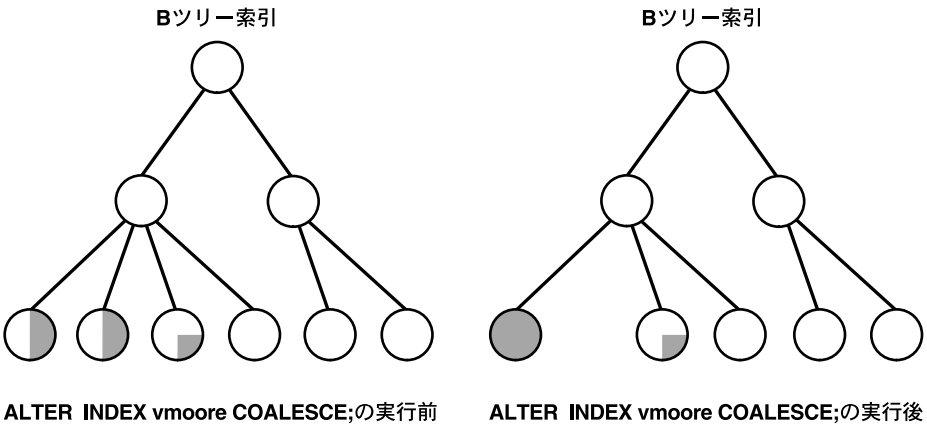
索引の再作成	索引の結合
索引を別の表領域に迅速に移動できる。	索引を別の表領域に移動することはできない。
多くのディスク領域を必要とし、コストが高い。	必要なディスク領域が少ないため、コストが低い。
新しいツリーを作成して、可能であればその高さを縮小する。	ツリーの同じブランチ内のリーフ・ブロックを結合する。
オリジナルの索引を削除せずに、記憶域パラメータと表領域パラメータを迅速に変更できる。	索引のリーフ・ブロックを迅速に解放できる。

再利用のために解放できる B ツリー索引のリーフ・ブロックがある場合は、次の文を使用してそのようなリーフ・ブロックをマージできます。

```
ALTER INDEX vmoore COALESCE;
```

図 19-1 は、索引 vmoore に対する ALTER INDEX COALESCE の効果を示しています。処理を実行する前は、最初の 2 つのリーフ・ブロックが半分満たされています。これは、断片化を低減できる余地があることを示しています。つまり、1 番目のブロックをいっぱいにして、2 番目のブロックを空にすることができます。

図 19-1　索引の結合



制約を使用禁止または削除する前のコストの検討

一意キーと主キーには対応する索引があるため、一意キー制約や主キー制約を使用禁止または削除するかどうかを検討するときには、索引の削除と作成にかかわるコストを分析してください。また、一意キー制約や主キー制約に対する索引が大きい場合には、その索引を削除して再作成するよりも、その制約を使用可能な状態のままにするほうが時間を節約できます。一意キー制約や主キー制約を削除または使用禁止にするときには、索引を保持するか削除するかを明示的に指定することもできます。

関連項目： 16-10 ページ「整合性制約の管理」

索引の作成

ここでは、索引の作成方法について説明します。自分のスキーマに索引を作成するには、次の条件の少なくとも 1 つが成り立つことが必要です。

- 索引を付ける表またはクラスタが、自分のスキーマに格納されている。
- 索引を付ける表に対する INDEX 権限を持っている。
- CREATE ANY INDEX システム権限を持っている。

自分のスキーマ以外のスキーマに索引を作成するには、次の条件がすべて成り立つことが必要です。

- CREATE ANY INDEX システム権限を持っている。
- 目的のスキーマの所有者が、索引または索引パーティションを作成する表領域への割当て制限を持っているか、UNLIMITED TABLESPACE システム権限を持っている。

この項の内容は、次のとおりです。

索引の明示的な作成

SQL 文 CREATE INDEX を使用して、索引を明示的に（整合性制約の他に）作成できます。次の文は、emp 表の ename 列に対して emp_ename という名前の索引を作成します。

```
CREATE INDEX emp_ename ON emp(ename)
      TABLESPACE users
      STORAGE (INITIAL 20K
      NEXT 20k
      PCTINCREASE 75);
```

索引に対して、複数の記憶域設定および 1 つの表領域が明示的に指定されています。索引に記憶域オプション（INITIAL や NEXT など）を指定しない場合、デフォルトの表領域または指定された表領域のデフォルトの記憶域オプションが自動的に使用されます。

> **関連項目：** CREATE INDEX 文の構文と制限事項については、『Oracle Database SQL リファレンス』を参照してください。

一意索引の明示的な作成

索引は、一意にすることも、重複を許可することもできます。一意索引を使用すると、表の複数行のキー列に重複した値が入らないことが保証されます。非一意索引では、列の値にこのような制限はありません。

一意索引を作成するには、CREATE UNIQUE INDEX 文を使用します。次の例では、一意索引を作成しています。

```
CREATE UNIQUE INDEX dept_unique_index ON dept (dname)
      TABLESPACE indx;
```

別の方法として、目的の列に一意整合性制約を定義することもできます。データベースでは、一意のキーに対して自動的に一意索引を定義することによって、一意整合性制約を規定します。この操作については、次の項を参照してください。ただし、問合せのパフォーマンス向上のために必要な索引は、一意索引も含め、明示的に作成することをお薦めします。

> **関連項目：** パフォーマンス向上のための索引作成の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

制約に対応付けられた索引の作成

Oracle Database は、表に一意キー整合性制約または主キー整合性制約を規定するために、一意キーまたは主キーの一意索引を作成します。この索引は、制約を使用可能にしたときに、データベースによって自動的に作成されます。CREATE TABLE 文または ALTER TABLE 文を発行して索引を作成する場合は、それ以外に必要なアクションはありませんが、必要に応じて、USING INDEX 句を指定して索引作成を制御することができます。これは、制約を定義して使用可能にする場合、および定義したが使用禁止にしていた制約を使用可能にする場合のどちらでも可能です。

一意キー制約または主キー制約を使用可能にし、対応する索引を作成するには、表の所有者が索引を格納する表領域の割当て制限または UNLIMITED TABLESPACE システム権限を持っている必要があります。特に指定しないかぎり、制約に対応付けられた索引には、常に制約と同じ名前が付けられます。

注意： 並列性を利用できるように制約を使用可能にする効率的な手順は、16-12 ページの「整合性制約の効率的な使用：手順」を参照してください。

制約に対応付けられた索引に対する記憶域オプションの指定

USING INDEX 句を使用すると、一意キー制約または主キー制約に対応する索引の記憶域オプションを設定できます。次の CREATE TABLE 文は、主キー制約を使用可能にして、対応する索引の記憶域オプションを指定します。

```
CREATE TABLE emp (
     empno NUMBER(5) PRIMARY KEY, age INTEGER)
     ENABLE PRIMARY KEY USING INDEX
     TABLESPACE users;
```

制約に対応付けられた索引の指定

一意キー制約および主キー制約に対応付けられた索引をより明示的に制御する場合、データベースでは次のことが可能です。

- 制約を規定するために使用する既存の索引の指定
- 索引の作成と制約の規定に使用する CREATE INDEX 文の指定

これらのオプションは、USING INDEX 句を使用して指定します。次にいくつかの例を示します。

例 1:

```
CREATE TABLE a (
     a1 INT PRIMARY KEY USING INDEX (create index ai on a (a1)));
```

例 2:

```
CREATE TABLE b(
     b1 INT,
     b2 INT,
     CONSTRAINT bu1 UNIQUE (b1, b2)
                    USING INDEX (create unique index bi on b(b1, b2)),
     CONSTRAINT bu2 UNIQUE (b2, b1) USING INDEX bi);
```

例 3:

```
CREATE TABLE c(c1 INT, c2 INT);
CREATE INDEX ci ON c (c1, c2);
ALTER TABLE c ADD CONSTRAINT cpk PRIMARY KEY (c1) USING INDEX ci;
```

単一の文で制約とともに索引を作成し、同じ文でその索引を別の制約にも使用する場合、Oracle では、索引を再使用する前に索引を作成するための句の再編成を試みます。

関連項目： 16-10 ページ「整合性制約の管理」

索引作成時の付随的統計の収集

Oracle Database では、索引の作成または再作成の際、リソース・コストをほとんど使用せずに統計を収集できます。これらの統計はデータ・ディクショナリに格納され、SQL 文の実行計画を選択する際にオプティマイザによって常時使用されます。次の文では、表 emp の列 ename に対して索引 emp_ename を作成すると同時に、索引、表および列の各統計を計算します。

```
CREATE INDEX emp_ename ON emp(ename)
     COMPUTE STATISTICS;
```

関連項目：

- 統計の収集とオプティマイザによるその使用の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- 16-3 ページ「表、索引およびクラスタの分析」

大きな索引の作成

極端に大きい索引を作成するときは、次の手順を使用して、索引の作成に大きな一時表領域を割り当てることを検討してください。

1. CREATE TABLESPACE 文または CREATE TEMPORARY TABLESPACE 文を使用して、新しい一時表領域を作成します。
2. ALTER USER 文で TEMPORARY TABLESPACE オプションを指定して、この一時表領域を自分の新しい一時表領域として割り当てます。
3. CREATE INDEX 文を使用して、索引を作成します。
4. DROP TABLESPACE 文を使用して、この表領域を削除します。次に ALTER USER 文を使用して、自分の一時表領域を元の一時表領域に再設定します。

この手順により、通常使用している一時表領域（ほとんどの場合、共有されている）が極度に肥大化して今後のパフォーマンスに影響を及ぼす問題を回避できます。

索引のオンラインでの作成

索引はオンラインで作成または再作成できます。これにより、実表に索引を作成または再作成しながら、同じ実表を更新できます。索引の作成中でもデータ操作言語（DML）操作が実行できます。しかし、データ定義言語（DDL）操作は実行できません。また、索引の作成中または再作成中のパラレル実行はサポートされていません。

次の文は、オンラインでの索引作成操作を示しています。

```
CREATE INDEX emp_name ON emp (mgr, emp1, emp2, emp3) ONLINE;
```

注意： オンライン索引ビルドが完了するまでの時間は、表のサイズおよび同時に実行している DML 文の数に比例することに注意してください。したがって、オンライン索引ビルドは DML アクティビティが低いときに開始することをお薦めします。

関連項目： 19-14 ページ「既存の索引の再作成」

ファンクション索引の作成

ファンクション索引を使用すると、関数や式から返される値を修飾する問合せが可能です。関数や式の値は、事前に計算されて索引に格納されます。

ユーザー定義のファンクションに基づく索引の場合は、従来型の索引を作成するための前提条件に加えて、これらのファンクションに DETERMINISTIC のマークが設定されている必要があります。さらに、これらのファンクションを別のユーザーが所有している場合は、ファンクション索引で使用するすべてのユーザー定義ファンクションに対する EXECUTE オブジェクト権限を持っている必要があります。

また、ファンクション索引を使用するには、次のことを行います。

- 索引の作成後に表を分析します。
- NULL 値は索引に格納されないため、問合せには索引を作成した式からの NULL 値を必要としないことを保証します。

注意： CREATE INDEX では、ファンクション索引で最後に使用された関数のタイムスタンプが格納されます。このタイムスタンプは、索引の妥当性チェック時に更新されます。ファンクション索引について表領域の Point-in-Time リカバリを実行する場合、索引で最後に使用された関数のタイムスタンプが索引に格納されたタイムスタンプより新しい場合は、その索引には無効を示すマークが設定されます。ANALYZE INDEX...VALIDATE STRUCTURE 文を使用して、この索引の妥当性をチェックする必要があります。

ファンクション索引の具体例として、ファンクション area(geo) に定義されているファンクション索引（area_index）を定義する次の文を示します。

```
CREATE INDEX area_index ON rivers (area(geo));
```

次の SQL 文では、area(geo) が WHERE 句で参照されているため、オプティマイザは索引 area_index の使用を考慮します。

```
SELECT id, geo, area(geo), desc
     FROM rivers
     WHERE Area(geo) >5000;
```

表の所有者は、ファンクション索引で使用されるファンクションに対して EXECUTE 権限を持つ必要があります。

ファンクション索引は使用するファンクションに依存するため、ファンクションの変更時に無効にできます。ファンクションが有効な場合は、ALTER INDEX...ENABLE 文を使用して、使用禁止になっているファンクション索引を使用可能にすることができます。ALTER INDEX...DISABLE 文では、ファンクション索引を使用禁止にすることができます。ファンクションの本体で作業をする場合は、ファンクション索引を使用禁止にするか検討してください。

注意： ファンクション索引を作成するかわりに、仮想列をターゲット表に追加して仮想列に索引を付けることができます。詳細は、18-2 ページの「表の概要」を参照してください。

関連項目：

- ファンクション索引の詳細は、『Oracle Database 概要』を参照してください。
- アプリケーションでファンクション索引を使用する際の詳細と使用例については、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

キー圧縮型索引の作成

キー圧縮を使用して索引を作成すると、キー列の接頭辞が同じ値で繰り返し格納されることを回避できます。

キー圧縮によって、索引キーは接頭辞および接尾辞エントリに分割されます。圧縮するために、接頭辞エントリは索引ブロック内のすべての接尾辞エントリ間で共有されます。このような共有によって、領域が大幅に節約され、各索引ブロックに格納できるキー数が増え、パフォーマンスが向上します。

キー圧縮は、次のような状況で役立ちます。

- ROWID を追加してキーを一意にしている非一意索引がある場合。このような状況でキー圧縮を使用すると、重複キーは接頭辞エントリとして索引ブロックに ROWID なしで格納されます。残りの行は、ROWID のみからなる接尾辞エントリとなります。
- 一意の複数列索引がある場合。

キー圧縮を使用可能にするには、COMPRESS 句を使用します。また、接頭辞の長さをキー列の数で指定できます。これにより、キー列が接頭辞および接尾辞エントリにどのように分割されるかが決まります。たとえば、次の文は、索引リーフ・ブロック内のキーの重複発生を圧縮します。

```
CREATE INDEX  emp_ename ON emp(ename)
   TABLESPACE users
   COMPRESS 1;
```

索引の再作成中に COMPRESS 句を指定することもできます。たとえば、次のようにして、再作成中に圧縮を使用禁止にすることができます。

```
ALTER INDEX emp_ename REBUILD NOCOMPRESS;
```

関連項目： キー圧縮の詳細は、『Oracle Database 概要』を参照してください。

不可視索引の作成

リリース 11*g* からは、不可視索引を作成できます。**不可視索引**とは、セッションまたはシステム・レベルで OPTIMIZER_USE_INVISIBLE_INDEXES 初期化パラメータを明示的に TRUE に設定しないかぎり、オプティマイザで無視される索引です。索引を使用禁止にしたり削除するかわりに、索引を不可視にできます。不可視索引を使用すると、次の操作を実行できます。

- 索引を削除する前に、削除した状態をテストできます。
- アプリケーション全体に影響を与えずに、アプリケーションの特定の操作またはモジュールに対して一時的な索引構造を使用できます。

使用禁止状態の索引とは異なり、不可視索引は DML 文が終了するまで保持されます。

不可視索引を作成するには、SQL 文 CREATE INDEX で INVISIBLE 句を指定します。次の文は、emp 表の ename 列に対して emp_ename という名前の不可視索引を作成します。

```
CREATE INDEX emp_ename ON emp(ename)
      TABLESPACE users
      STORAGE (INITIAL 20K
      NEXT 20k
      PCTINCREASE 75)
      INVISIBLE;
```

関連項目：

- 19-14 ページ「索引の不可視化」
- SELECT 文のコメントを使用して Oracle Database オプティマイザにヒントを渡す方法については、『Oracle Database SQL リファレンス』を参照してください。

索引の変更

索引を変更するには、その索引が自分のスキーマに含まれているか、または ALTER ANY INDEX システム権限を持っている必要があります。ALTER INDEX 文では、次の操作を実行できます。

- 既存の索引の再作成または結合
- 未使用領域の割当て解除または新規エクステントの割当て
- パラレル実行の指定（または指定解除）およびその並列度の変更
- 記憶域パラメータまたは物理属性の変更
- LOGGING または NOLOGGING の指定
- キー圧縮の使用可能または使用禁止の設定
- 索引への UNUSABLE マークの設定
- 索引の不可視化
- 索引の名前変更
- 索引使用状況の監視の開始または停止

索引の列構造は変更できません。

これらの操作のいくつかについて、次の項で詳しく説明します。

- 索引の記憶域特性の変更
- 既存の索引の再作成
- 索引の不可視化
- 索引の使用状況の監視

関連項目：

- ALTER INDEX 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

索引の記憶域特性の変更

主キーと一意キーの整合性制約を規定するためにデータベースによって作成される索引も含めて、どの索引の記憶域パラメータも、ALTER INDEX 文を使用して変更できます。たとえば、次の文は emp_ename 索引を変更します。

```
ALTER INDEX emp_ename
     STORAGE (PCTINCREASE 50);
```

記憶域パラメータ INITIAL と MINEXTENTS は変更できません。他の記憶域パラメータの新しい設定はすべて、その後に索引に割り当てられるエクステントにのみ影響します。

整合性制約を実装する索引では、ENABLE 句の USING INDEX 副次句を指定した ALTER TABLE 文を発行することによって、記憶域パラメータを調整できます。たとえば、次の文は、表 emp に作成された索引の記憶域オプションを変更して、主キー制約を規定します。

```
ALTER TABLE emp
     ENABLE PRIMARY KEY USING INDEX;
```

関連項目： ALTER INDEX 文の構文と制限事項については、『Oracle Database SQL リファレンス』を参照してください。

既存の索引の再作成

既存の索引を再作成する前に、19-6 ページの表 19-1 の説明に従って、索引を再作成する方法と結合する方法のコストと利点を比較してください。

索引を再作成するときは、既存の索引をデータソースとして使用できます。このようにして索引を作成すると、記憶特性の変更や新しい表領域への移動ができます。既存のデータソースに基づいて索引を再作成すると、ブロック内の断片化も解消されます。索引を削除して CREATE INDEX 文を使用するよりも、既存の索引を再作成した方がパフォーマンスは優れています。

次の文は、既存の索引 emp_name を再作成します。

```
ALTER INDEX emp_name REBUILD;
```

REBUILD 句は、索引名の直後で他のオプションより前に指定する必要があります。この句を DEALLOCATE UNUSED 句と組み合せて使用することはできません。

索引はオンラインで再作成できます。オンラインで再作成することにより、再作成と同時に実表を更新できます。次の文は、emp_name 索引をオンラインで再作成します。

```
ALTER INDEX emp_name REBUILD ONLINE;
```

注意： オンラインでの索引の再作成には、別の方式による索引の再作成に比べ、処理できるキーの最大長に関して、より厳しい制限があります。オンラインでの再作成時に ORA-01450（キーの最大長超過）エラーが発生した場合は、オフラインで再作成し結合を試みるか、索引を削除し再作成します。

索引の再作成に必要な大きさの領域がない場合、かわりに索引を結合する方法が使用できます。索引の結合はオンライン操作です。

関連項目：

- 19-10 ページ「索引のオンラインでの作成」
- 19-15 ページ「索引の領域使用の監視」

索引の不可視化

可視索引を不可視にするには、次の文を発行します。

```
ALTER INDEX index INVISIBLE;
```

不可視索引を可視にするには、次の文を発行します。

```
ALTER INDEX index VISIBLE;
```

索引が可視か不可視かを調べるには、ディクショナリ・ビュー USER_INDEXES、ALL_INDEXES または DBA_INDEXES を問い合せます。たとえば、索引 IND1 が不可視かどうかを調べるには、次の問合せを発行します。

```
SELECT INDEX_NAME, VISIBILITY FROM USER_INDEXES
   WHERE INDEX_NAME = 'IND1';

INDEX_NAME   VISIBILITY
----------   ----------
IND1         VISIBLE
```

関連項目： 不可視索引の定義については、19-12 ページの「不可視索引の作成」を参照してください。

索引の名前変更

索引の名前を変更するには、次の文を発行します。

```
ALTER INDEX index_name RENAME TO new_name;
```

索引の使用状況の監視

Oracle Database には、索引を監視し、それらが使用されているかどうかを判断する手段が用意されています。未使用の索引がある場合は、それを削除して不要な文によるオーバーヘッドを解消できます。

索引の使用状況の監視を開始するには、次の文を発行します。

```
ALTER INDEX index MONITORING USAGE;
```

その後、監視を停止するには、次の文を発行します。

```
ALTER INDEX index NOMONITORING USAGE;
```

監視中の索引について、それが使用されているかどうか確認するには、ビュー V$OBJECT_USAGE を問い合せます。このビューには USED 列があり、監視期間中に索引が使用されたかどうかによって YES または NO の値を持ちます。また、このビューには監視の開始時間と停止時間も記録され、MONITORING 列（YES/NO）には使用状況の監視が現在アクティブであるかどうかが示されます。

MONITORING USAGE を指定するたびに、指定した索引の V$OBJECT_USAGE ビューはリセットされます。前回の使用方法の情報はクリアまたはリセットされ、新しい開始時間が記録されます。NOMONITORING USAGE を指定すると、これ以上の監視は実行されず、監視期間の終了時間が記録されます。次に ALTER INDEX...MONITORING USAGE 文が発行されるまで、ビューの情報は変更されずに保持されます。

索引の領域使用の監視

索引のキー値を頻繁に挿入、更新および削除していると、索引がその取得した領域を効果的に使用しなくなる可能性があります。そこで、定期的な間隔で索引の領域使用の効率を監視します。まず、ANALYZE INDEX...VALIDATE STRUCTURE 文を使用して索引の構造を分析した後、次のように INDEX_STATS ビューを問い合せます。

```
SELECT PCT_USED FROM INDEX_STATS WHERE NAME = 'index';
```

索引の領域使用の割合は、どれくらい頻繁に索引キーが挿入、更新または削除されるかによって変化します。次の一連の操作を何度か実行し、索引の平均的な領域使用効率の履歴を作成してください。

- 統計分析
- 索引の検証
- PCT_USED のチェック
- 索引の削除および再作成（または結合）

索引の領域使用がその平均を下回っているときは、索引を削除してから再作成または結合することによって、索引の領域を圧縮できます。

関連項目： 16-3 ページ「表、索引およびクラスタの分析」

索引の削除

索引を削除するには、その索引が自分のスキーマに含まれているか、または DROP ANY INDEX システム権限を持っている必要があります。

索引を削除するのは、次のような場合です。

- 索引が不要になった場合。
- 対応する表に対して発行した問合せで、索引が予想されたパフォーマンスの改善を達成していない場合。たとえば、表が非常に小さい、または表には多くの行があるものの、索引エントリが非常に少ないなどがこれに該当します。
- アプリケーションに索引を使用するデータ問合せが含まれない場合。
- 索引が無効になり、再作成する前に削除する必要がある場合。
- 索引がかなり断片化し、再作成する前に削除する必要がある場合。

索引が削除されると、その索引のセグメントのエクステントはすべて、索引を含んでいる表領域に戻され、表領域内の他のオブジェクトで利用できます。

索引を削除する方法は、索引の作成方法、つまり CREATE INDEX 文によって索引を明示的に作成したか、または表にキー制約を定義することによって索引を暗黙的に作成したかによって異なります。CREATE INDEX 文を使用して明示的に作成した索引は、DROP INDEX 文で削除できます。次の文は、emp_ename 索引を削除します。

```
DROP INDEX emp_ename;
```

使用可能になっている一意キー制約や主キー制約に対応付けられた索引のみを削除することはできません。制約に対応付けられた索引を削除するには、制約自体を使用禁止にするかまたは削除します。

注意： 表を削除すると、対応する索引はすべて自動的に削除されます。

関連項目：

- DROP INDEX 文の構文と制限事項については、『Oracle Database SQL リファレンス』を参照してください。
- 16-10 ページ「整合性制約の管理」
- 索引の削除にかわる方法については、19-14 ページの「索引の不可視化」を参照してください。

索引のデータ・ディクショナリ・ビュー

次のビューには、索引に関する情報が表示されます。

ビュー	説明
DBA_INDEXES ALL_INDEXES USER_INDEXES	DBA ビューには、データベース内にあるすべての表の索引が表示されます。ALL ビューには、ユーザーがアクセス可能なすべての表の索引が表示されます。USER ビューは、ユーザーが所有する索引のみに制限されます。これらのビューの一部の列には、DBMS_STATS パッケージまたは ANALYZE 文によって生成される統計が含まれます。
DBA_IND_COLUMNS ALL_IND_COLUMNS USER_IND_COLUMNS	これらのビューには、表の索引の列が表示されます。これらのビューの一部の列には、DBMS_STATS パッケージまたは ANALYZE 文によって生成される統計が含まれます。
DBA_IND_EXPRESSIONS ALL_IND_EXPRESSIONS USER_IND_EXPRESSIONS	これらのビューには、表のファンクション索引の式が表示されます。
DBA_IND_STATISTICS ALL_IND_STATISTICS USER_IND_STATISTICS	これらのビューには、索引のオプティマイザ統計が含まれています。
INDEX_STATS	最後に発行された ANALYZE INDEX...VALIDATE STRUCTURE 文の情報が格納されています。
INDEX_HISTOGRAM	最後に発行された ANALYZE INDEX...VALIDATE STRUCTURE 文の情報が格納されています。
V$OBJECT_USAGE	ALTER INDEX...MONITORING USAGE 機能で生成された索引使用状況の情報が含まれます。

関連項目： これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。

20

クラスタの管理

この章の内容は次のとおりです。

クラスタの概要

クラスタは、表データを格納するために選択可能なオプションの方法を提供します。クラスタは、同じデータ・ブロックを共有する表のグループで構成されています。表をグループ化する理由は、各表が共通の列を共有しており、一緒に使用されるケースが多いためです。たとえば、emp 表と dept 表が deptno 列を共有しているとします。emp 表および dept 表をクラスタ化する場合（図 20-1 を参照）、Oracle Database では、emp 表および dept 表の各部門の行はすべて、物理的に同じデータ・ブロックに格納されます。

クラスタは、異なる表の関連する行を同じデータ・ブロックに格納します。そのため、クラスタを正しく使用することには、主に次のような利点があります。

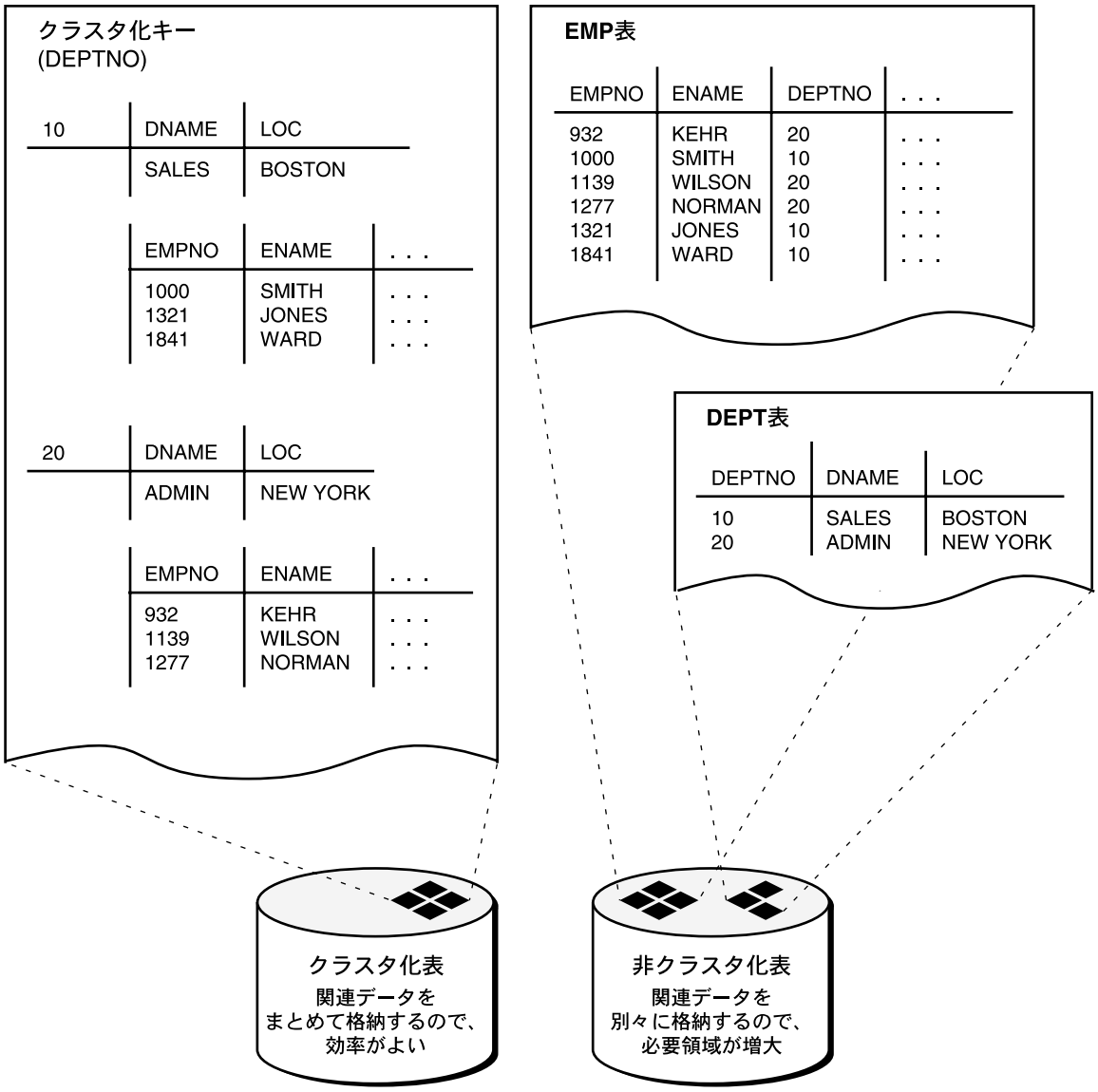
- クラスタ化表の結合によって、ディスク I/O が減少し、アクセス時間が改善されます。
- **クラスタ・キー**は、クラスタ化表が共有する列または列のグループです。最初にクラスタを作成するときに、クラスタ・キー列を指定します。その後、そのクラスタに追加する表を作成するたびに同じ列を指定します。各クラスタ・キー値は、その値が含まれている異なる表行数に関係なく、クラスタとクラスタ索引のそれぞれに 1 度しか格納されません。

  そのため、関連する表および索引データをクラスタに格納するために必要な記憶域は、非クラスタ化表形式の場合に比べて少なくて済みます。図 20-1 に、クラスタ・キーの格納方法を示します。各クラスタ・キー（各 deptno）は、emp 表と dept 表の両方に同じ値が含まれている多数の行について 1 度しか格納されていません。

クラスタを作成した後、そのクラスタ内に表を作成できます。ただし、クラスタ化表に行を挿入する前に、クラスタ索引を作成する必要があります。クラスタを使用しても、クラスタ化表に対して索引を追加することには影響しません。索引は通常どおり作成および削除できます。

別々にアクセスされることの多い表には、クラスタを使用しないでください。

図 20-1　クラスタ化表データ



関連項目：

- もう 1 つのタイプのクラスタ（ハッシュ・クラスタ）の詳細は、第 21 章「ハッシュ・クラスタの管理」を参照してください。
- この章のタスクを実行する前に、第 17 章「スキーマ・オブジェクトの領域の管理」を一読されることをお薦めします。

クラスタを管理するためのガイドライン

次の項では、クラスタを管理する際に考慮すべきガイドラインについて説明します。この項の内容は、次のとおりです。

- クラスタに適した表の選択
- クラスタ・キーに適した列の選択
- 平均クラスタ・キーとその対応行が必要とする領域の指定
- 各クラスタとクラスタ索引の行の位置の指定
- クラスタ・サイズの見積りと記憶域パラメータの設定

関連項目：

- クラスタの詳細は、『Oracle Database 概要』を参照してください。
- どのようなときにクラスタを使用するかについてのガイドラインは、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

クラスタに適した表の選択

次のような条件が成り立つ場合は、表にクラスタを使用します。

- 表に対する主な操作が挿入や更新ではなく、問合せである場合
- 複数の表のレコードを頻繁に問い合せたり、結合する場合

クラスタ・キーに適した列の選択

クラスタ・キー列の選択には注意が必要です。表を結合する問合せで複数列を使用する場合、クラスタ・キーはコンポジット・キーにします。一般に、適切なクラスタ索引を表す特性は、適切な索引を表す特性と同じです。適切な索引を表す特性の詳細は、19-2 ページの「索引を管理するためのガイドライン」を参照してください。

適切なクラスタ・キーの条件は、1 つのキー値に対応している行のグループで 1 つのデータ・ブロックがほぼいっぱいになるような、一意の値を持つことです。クラスタ・キー値ごとの行が少なすぎると、領域を浪費し、結果的にパフォーマンスが低下します。共通の値を共有する行がわずかしかないクラスタ・キーは、クラスタを作成するときに SIZE に小さい値を指定しないかぎり、ブロック内の領域を浪費する可能性があります（20-5 ページの「平均クラスタ・キーとその対応行が必要とする領域の指定」を参照してください）。

クラスタ・キー値ごとの行が多すぎると、そのキーを持つ行を見つけるために余分な検索が起こる可能性があります。クラスタ・キーの値があまりにも一般的な場合（たとえば、male と female といった性別など）は、過度の検索が発生し、クラスタ化していない場合よりパフォーマンスが低下するおそれがあります。

クラスタ索引は一意にできません。また、LONG 型として定義されている列を含むことはできません。

平均クラスタ・キーとその対応行が必要とする領域の指定

CREATE CLUSTER 文にはオプションの句 SIZE があります。これは、平均クラスタ・キーとそれに対応する行で必要なバイト数の見積りです。データベースでは、次の処理を実行するときに、SIZE パラメータが使用されます。

- クラスタ化したデータ・ブロック内に収めることのできるクラスタ・キー（および対応する行）の数を見積る場合。
- クラスタ化したデータ・ブロックに配置するクラスタ・キーの数を制限する場合。これにより、クラスタ内のキーの格納効率が最大になります。

SIZE は、クラスタ・キーが使用できる領域を制限しません。たとえば、2 つのクラスタ・キーが 1 つのデータ・ブロックに収まるように SIZE が設定された場合、どちらのクラスタ・キーでも、その利用可能なデータ・ブロック領域をいくらでも使用できます。

デフォルトでは、データベースは 1 つのクラスタ・キーとそれに対応する行をそのクラスタのデータ・セグメントの 1 データ・ブロックに格納します。ブロック・サイズはオペレーティング・システムによって異なることがありますが、クラスタ化表が異なるマシン上にある他のデータベースにインポートされるときにも、ブロック当たり 1 つのキーというルールは守られます。

クラスタ・キー値に対応するすべての行を 1 つのブロック内に収めることができない場合は、ブロックをまとめて連鎖することにより、特定のキーを持つすべての値へのアクセスの高速化が図られます。クラスタ索引は、クラスタ・キー値と対応する行をそれぞれに含むブロックの連鎖の始点を示します。クラスタの SIZE が、複数のキーが 1 つのブロックに収まる大きさに設定されている場合は、ブロックが複数の連鎖に属する可能性があります。

各クラスタとクラスタ索引の行の位置の指定

適切な権限と表領域割当て制限を持つユーザーであれば、現在オンライン状態の表領域内に新しいクラスタおよび関連するクラスタ索引を作成できます。新しいクラスタまたは索引を格納する表領域を識別するには、CREATE CLUSTER/INDEX 文に必ず TABLESPACE 句を指定します。

クラスタとそのクラスタ索引は、異なる表領域内に作成できます。実際、クラスタとその索引を、それぞれ異なる記憶デバイス上に格納された異なる表領域内に作成すると、ディスク競合を最小限に抑えて、表データと索引データを同時に検索できます。

クラスタ・サイズの見積りと記憶域パラメータの設定

クラスタを作成する前にクラスタ・サイズを見積る利点は、次のとおりです。

- クラスタの見積りサイズの合計と、索引および REDO ログ・ファイルの見積りを使用して、作成するデータベースを格納するために必要なディスク容量を決定できます。この見積りを利用して適切なハードウェアを購入できます。
- 個々のクラスタの見積りサイズを使用することで、クラスタが使用するディスク領域をより適切に管理できます。クラスタを作成するときに、適切な記憶域パラメータを設定し、そのクラスタを使用するアプリケーションの I/O パフォーマンスを改善できます。

クラスタのデータ・セグメントに対する記憶域パラメータは、CREATE CLUSTER 文または ALTER CLUSTER 文の STORAGE 句を使用して設定します。クラスタ内に表を設定する個々の CREATE 文または ALTER 文には STORAGE 句は指定しません。クラスタ化された表の作成時または変更時に指定した記憶域パラメータは無視されます。クラスタに設定された記憶域パラメータは、表の記憶域パラメータを上書きします。

クラスタの作成

自分のスキーマにクラスタを作成するには、CREATE CLUSTER システム権限とそのクラスタを格納する表領域に対する割当て制限を持っているか、または UNLIMITED TABLESPACE システム権限を持っている必要があります。

別のユーザーのスキーマにクラスタを作成するには、CREATE ANY CLUSTER システム権限が必要です。さらに、所有者はそのクラスタを格納する表領域に対する割当て制限を持っているか、または UNLIMITED TABLESPACE システム権限を持っている必要があります。

クラスタを作成するには、CREATE CLUSTER 文を使用します。次の文は、deptno 列によってクラスタ化された、emp 表と dept 表を格納するクラスタ emp_dept を作成します。

```
CREATE CLUSTER emp_dept (deptno NUMBER(3))
   SIZE 600
   TABLESPACE users
   STORAGE (INITIAL 200K
      NEXT 300K
      MINEXTENTS 2
      PCTINCREASE 33);
```

この例のように INDEX キーワードを指定しない場合は、デフォルトで索引クラスタが作成されます。また、ハッシュ・パラメータ（HASHKEYS、HASH IS または SINGLE TABLE HASHKEYS）を指定すると、ハッシュ・クラスタを作成できます。ハッシュ・クラスタについては、第 21 章「ハッシュ・クラスタの管理」を参照してください。

クラスタ化表の作成

クラスタに表を作成するには、CREATE TABLE システム権限または CREATE ANY TABLE システム権限のどちらかが必要です。なお、クラスタ内に表を作成するために、表領域割当て制限または UNLIMITED TABLESPACE システム権限は必要ありません。

クラスタに表を作成するには、CLUSTER 句を指定した CREATE TABLE 文を使用します。次の文を使用すると、emp_dept クラスタ内に emp 表と dept 表を作成できます。

```
CREATE TABLE emp (
   empno NUMBER(5) PRIMARY KEY,
   ename VARCHAR2(15) NOT NULL,
   . . .
   deptno NUMBER(3) REFERENCES dept)
   CLUSTER emp_dept (deptno);

CREATE TABLE dept (
   deptno NUMBER(3) PRIMARY KEY, . . . )
   CLUSTER emp_dept (deptno);
```

注意： クラスタ化表のスキーマは、CREATE TABLE 文で指定できます。クラスタ化表は、そのクラスタを含むスキーマとは異なるスキーマに配置できます。また、列名は一致しなくてもかまいませんが、その構造は同じである必要があります。

関連項目： クラスタ表を作成する CREATE TABLE 文の構文は、『Oracle Database SQL リファレンス』を参照してください。

クラスタ索引の作成

クラスタ索引を作成するには、次の条件のいずれかが成り立つ必要があります。

- スキーマにクラスタが含まれている。
- CREATE ANY INDEX システム権限を持っている。

どちらの場合も、クラスタ索引を格納する表領域に対する割当て制限または UNLIMITED TABLESPACE システム権限が必要です。

クラスタ索引は、クラスタ化表に行を挿入する前に作成する必要があります。次の文は、emp_dept クラスタに対するクラスタ索引を作成します。

```
CREATE INDEX emp_dept_index
   ON CLUSTER emp_dept
   TABLESPACE users
   STORAGE (INITIAL 50K
      NEXT 50K
      MINEXTENTS 2
      MAXEXTENTS 10
      PCTINCREASE 33);
```

クラスタ索引句（ON CLUSTER）では、クラスタ索引を作成するクラスタ emp_dept を識別します。この文では、クラスタとクラスタ索引の記憶域設定も明示的に指定しています。

> **関連項目：** クラスタ索引を作成する CREATE INDEX 文の構文は、『Oracle Database SQL リファレンス』を参照してください。

クラスタの変更

クラスタを変更するには、そのクラスタが自分のスキーマに含まれているか、または ALTER ANY CLUSTER システム権限を持っている必要があります。既存のクラスタを変更することにより、次の設定を変更できます。

- 物理属性（INITRANS および記憶域特性）
- クラスタ・キー値のすべての行を格納するために必要な平均使用可能空き領域（SIZE）
- デフォルトの並列度

また、クラスタに新しいエクステントを明示的に割り当てたり、クラスタの最後にある未使用のエクステントの割当てを解除できます。データベースは、必要に応じてクラスタのデータ・セグメントに追加のエクステントを動的に割り当てます。ただし、状況によっては、クラスタに追加のエクステントを明示的に割り当てることもできます。たとえば、Real Application Clusters を使用している場合、クラスタのエクステントを特定のインスタンスに明示的に割り当てることができます。クラスタに新しいエクステントを割り当てるには、ALLOCATE EXTENT 句を指定した ALTER CLUSTER 文を使用します。

クラスタ・サイズ・パラメータ（SIZE）を変更すると、そのクラスタにすでに割り当てられているブロックと今後割り当てられるブロックを含め、そのクラスタが使用するすべてのデータ・ブロックに対して新しい設定が適用されます。すでに表に割り当てられているブロックは、即時ではなく、必要なときに再編成されます。

クラスタのトランザクション・エントリ設定 INITRANS を変更すると、INITRANS の新しい設定は、その後クラスタに割り当てられるブロックにのみ適用されます。

記憶域パラメータ INITIAL と MINEXTENTS は変更できません。他の記憶域パラメータの新しい設定はすべて、その後にクラスタに割り当てられるエクステントにのみ影響します。

クラスタを変更するには、ALTER CLUSTER 文を使用します。

> **関連項目：** ALTER CLUSTER 文の構文は、『Oracle Database SQL リファレンス』を参照してください。

クラスタ化表の変更

クラスタ化表を変更するには、ALTER TABLE 文を使用します。ただし、クラスタ化表に対して ALTER TABLE 文でデータ・ブロック領域パラメータ、トランザクション・エントリ・パラメータまたは記憶域パラメータを設定しても、エラー・メッセージ（ORA-01771「クラスタ表に対するオプションが無効です。」）が出力されます。これは、クラスタ化表では必ずそのクラスタのパラメータが使用されるためです。そのため、クラスタ化表に対して ALTER TABLE 文を使用できるのは、列の追加や変更、非クラスタ化キー列の削除、整合性制約やトリガーの追加、削除、使用可能または使用禁止のみに限定されます。表の変更方法については、18-20 ページの「表の変更」を参照してください。

関連項目： ALTER TABLE 文の構文は、『Oracle Database SQL リファレンス』を参照してください。

クラスタ索引の変更

クラスタ索引は、他の索引と同じように変更できます。19-13 ページの「索引の変更」を参照してください。

注意： クラスタ索引のサイズを見積るときには、索引が実際の行ではなく各クラスタ・キーに付いていることに注意してください。したがって、各キーは索引内に 1 度しか現れません。

クラスタの削除

クラスタ内の表が不要になった場合は、そのクラスタを削除できます。クラスタを削除すると、そのクラスタ内の表および対応するクラスタ索引も削除されます。クラスタのデータ・セグメントとクラスタ索引の索引セグメントの両方に属するすべてのエクステントは、それらを含んでいる表領域に戻され、その表領域内の他のセグメントで使用可能になります。

表を含まないクラスタとそのクラスタ索引を削除するには、DROP CLUSTER 文を使用します。たとえば、次の文は空のクラスタ emp_dept を削除します。

```
DROP CLUSTER emp_dept;
```

クラスタに 1 つ以上のクラスタ化表が含まれており、その表も同様に削除する場合は、次のように、DROP CLUSTER 文に INCLUDING TABLES 句を追加します。

```
DROP CLUSTER emp_dept INCLUDING TABLES;
```

INCLUDING TABLES 句を指定していない場合に、クラスタに表が含まれていると、エラーが返されます。

クラスタ内の 1 つ以上の表が、そのクラスタ外の表の FOREIGN KEY 制約によって参照される主キーまたは一意キーを含んでいる場合、依存する FOREIGN KEY 制約が削除されないかぎり、そのクラスタを削除できません。この削除は、DROP CLUSTER 文の CASCADE CONSTRAINTS 句を使用すると簡単に実行できます。次に例を示します。

```
DROP CLUSTER emp_dept INCLUDING TABLES CASCADE CONSTRAINTS;
```

制約が存在している場合に、CASCADE CONSTRAINTS 句を使用しないと、データベースはエラーを返します。

関連項目： DROP CLUSTER 文の構文は、『Oracle Database SQL リファレンス』を参照してください。

クラスタ化表の削除

クラスタを削除するには、そのクラスタが自分のスキーマに含まれているか、または DROP ANY CLUSTER システム権限を持っている必要があります。クラスタ化表をそのクラスタの所有者が所有していなくても、その表を含むクラスタを削除するために特別な権限は必要ありません。

クラスタ化表は、そのクラスタ、他のクラスタ化表またはクラスタ索引に影響を及ぼすことなく、個別に削除できます。クラスタ化表は、非クラスタ化表を削除する場合と同じように、DROP TABLE 文を使用して削除します。18-24 ページの「表の列の削除」を参照してください。

注意： 単一の表をクラスタから削除するときは、表の各行が個別に削除されます。クラスタ全体を最も効率よく削除するには、INCLUDING TABLES 句を指定した DROP CLUSTER 文を使用して、そのクラスタを表も含めて削除します。クラスタの残りの部分をそのままにしておく場合のみ、（DROP TABLE 文を使用して）クラスタから表を個別に削除してください。

クラスタ索引の削除

クラスタ索引は、クラスタまたはそのクラスタ化表に影響を及ぼすことなく削除できます。ただし、クラスタ索引が存在しないと、クラスタ化表を使用できません。クラスタへのアクセスを可能にするには、クラスタ索引の再作成が必要です。断片化したクラスタ索引を再作成する手順の一部として、クラスタ索引を削除することがあります。索引を削除する方法の詳細は、19-16 ページの「索引の削除」を参照してください。

クラスタのデータ・ディクショナリ・ビュー

次のビューには、クラスタに関する情報が表示されます。

ビュー	説明
DBA_CLUSTERS ALL_CLUSTERS USER_CLUSTERS	DBA ビューには、データベース内のすべてのクラスタが表示されます。ALL ビューには、ユーザーがアクセス可能なすべてのクラスタが表示されます。USER ビューは、ユーザーが所有するクラスタのみに制限されます。これらのビューの一部の列には、DBMS_STATS パッケージまたは ANALYZE 文によって生成される統計が含まれます。
DBA_CLU_COLUMNS USER_CLU_COLUMNS	これらのビューでは、表の列とクラスタの列がマップされています。

関連項目： これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。

21

ハッシュ・クラスタの管理

この章の内容は次のとおりです。

ハッシュ・クラスタの概要

ハッシュ・クラスタに表を格納することは、データ検索のパフォーマンスを改善するための 1 つの選択肢です。ハッシュ・クラスタは、索引付きの非クラスタ化表または索引クラスタの代替手段を提供します。索引付きの表または索引クラスタでは、Oracle Database は、表内の行の位置を特定するために、別個の索引に格納しているキー値を使用します。ハッシングを使用するには、ハッシュ・クラスタを作成し、そこに表をロードします。表の行は物理的にはハッシュ・クラスタに格納され、**ハッシュ関数**の結果に従って検索されます。

Oracle Database はハッシュ関数を使用して、特定のクラスタ・キー値に基づく、**ハッシュ値**と呼ばれる数値の分布を生成します。ハッシュ・クラスタのキーは、索引クラスタのキーと同じように単一列キーでもコンポジット・キー（複数列キー）でもかまいません。ハッシュ・クラスタ内で行を検索または格納する場合、データベースは行のクラスタ・キー値にハッシュ関数を適用します。結果として生成されるハッシュ値はクラスタ内のデータ・ブロックに対応しており、データベースは、発行された文のかわりにそのデータ・ブロックに対して読取りまたは書込みを行います。

索引付きの表または索引クラスタ内の行を検索または格納するためには、少なくとも 2 回（通常はそれ以上）の I/O を実行する必要があります。

- 1 回以上の I/O による、索引内でのキー値の検索または格納
- 別の I/O による、表またはクラスタ内での行の読取りまたは書込み

一方、ハッシュ・クラスタでは、ハッシュ関数を使用して行の位置が特定されるので、I/O は必要ありません。結果として、ハッシュ・クラスタ内の行の読込みや書込みに必要とされるのは最小限の I/O 操作のみになります。

> **関連項目：** この章のタスクを実行する前に、第 17 章「スキーマ・オブジェクトの領域の管理」を一読されることをお薦めします。

ハッシュ・クラスタを使用する場合

ここでは、ハッシングが最も有効な場合と有効でない場合を対比することによって、ハッシュ・クラスタを使用すべき状況を判断する際に役立つ情報を提供します。ハッシングではなく索引を使用する場合は、表を個別に格納するのか、またはクラスタの一部として格納するのかを考慮してください。

注意： ハッシングを使用する場合でも、クラスタ・キーを含む表のどの列にも異なる索引を設定できます。

ハッシングが有効な状況

ハッシングは、次のような状況で有効です。

- ほとんどの問合せが次のようなクラスタ・キーとの等式を含んでいる場合。

  ```
  SELECT ... WHERE cluster_key = ...;
  ```

  このような場合、等価条件内のクラスタ・キーがハッシュされ、対応するハッシュ・キーが通常 1 回の読込みで検索されます。それに対して、索引付きの表では、最初にキー値を索引内で検索する必要があります（通常複数回の読込み）。その後で行が表から読み込まれます（別の読込み）。

- ハッシュ・クラスタ内の表のサイズが最初から固定されていて、行数とそのクラスタ内の表が必要とする領域を決定できる場合。ハッシュ・クラスタ内の表でそのクラスタの初期割当てより多くの領域が必要な場合、オーバーフロー・ブロックが必要となるためにパフォーマンスがかなり低下するおそれがあります。

ハッシングが不利な状況

ハッシングは次のような状況では有効ではありません。

- ほとんどの問合せがクラスタ・キー値全体にわたって行を検索する場合。たとえば、全表スキャンや次のような問合せでは、ハッシュ関数は特定のハッシュ・キーの位置を決定するために使用できません。そのかわりに、全表スキャンと同等の機能を実行して問合せの行をフェッチする必要があります。

  ```
  SELECT . . . WHERE cluster_key < . . . ;
  ```

  索引では、キー値はその索引内で順序付けられており、問合せの WHERE 句を満たすクラスタ・キー値を、比較的少ない I/O で見つけることができます。

- 表が固定でなく、継続的に拡大する場合。表が無制限に拡大する場合、表（そのクラスタ）の存続期間にわたって必要な領域を事前に定義することはできません。
- アプリケーションが頻繁に表の全表スキャンを実行し、その表内でデータが散在している場合。この状況でハッシングを使用すると、全表スキャンの処理時間が長くなります。
- 最終的にハッシュ・クラスタが必要とする領域を事前に割り当てることができない場合。

ハッシュ・クラスタの作成

ハッシュ・クラスタを作成するには、HASHKEYS 句を指定した CREATE CLUSTER 文を使用します。次の例では、trial 表を格納するクラスタ trial_cluster を作成し、trialno 列（クラスタ・キー）によってクラスタ化する文と、クラスタ内に表を作成する文を示しています。

```
CREATE CLUSTER trial_cluster (trialno NUMBER(5,0))
    TABLESPACE users
    STORAGE (INITIAL 250K     NEXT 50K
      MINEXTENTS 1     MAXEXTENTS 3
      PCTINCREASE 0)
    HASH IS trialno HASHKEYS 150;

CREATE TABLE trial (
    trialno NUMBER(5,0) PRIMARY KEY,
    ...)
    CLUSTER trial_cluster (trialno);
```

索引クラスタの場合と同様に、ハッシュ・クラスタのキーは、単一列でもコンポジット・キー（複数列のキー）でもかまいません。この例では、単一列が使用されています。

HASHKEYS 値（この例では 150）は、クラスタに使用されるハッシュ関数で生成できる一意のハッシュ値の数を指定し、制限します。指定した値は最も近い素数に丸められます。

HASH IS 句を指定しない場合は、内部ハッシュ関数が使用されます。すでにクラスタ・キーがその範囲に均一に分布する一意識別子の場合は、内部ハッシュ関数を無視し、前述の例のようにクラスタ・キーをハッシュ値として指定できます。また、HASH IS 句を使用して、ユーザー定義のハッシュ関数を指定することもできます。

ハッシュ・クラスタのクラスタ索引は作成できませんが、ハッシュ・クラスタ・キーに索引を作成する必要はありません。

クラスタ内に表を作成する方法、索引とハッシュ・クラスタに共通する CREATE CLUSTER 文のパラメータ設定のガイドライン、およびクラスタを作成するために必要な権限の追加情報は、第 20 章「クラスタの管理」を参照してください。次の項では、ハッシュ・クラスタ固有の CREATE CLUSTER 文のパラメータの設定とそのガイドラインについて説明します。

- ソートされたハッシュ・クラスタの作成
- 単一表ハッシュ・クラスタの作成
- ハッシュ・クラスタ内の領域使用の制御
- ハッシュ・クラスタに必要なサイズの見積り

ソートされたハッシュ・クラスタの作成

ソートされたハッシュ・クラスタでは、ハッシュ関数の各値に対応する行が、指定された列のセットの昇順でソートされます。これにより、クラスタ化したデータの後続の操作では応答時間が改善されます。

たとえば、電話会社では、1 つの交換機を介した固定数の発信電話番号について、詳細な通話記録を保存する必要があります。各発信電話番号から、無制限の通話件数が発生する可能性があります。

通話時に保存された通話記録は、後で請求書を作成するときに、発信電話番号ごとに先入れ先出し（FIFO）で処理されます。各通話には、タイムスタンプによって識別される詳細な通話記録があります。次のようなデータが収集されます。

発信電話番号	タイムスタンプによって識別される通話記録
650-555-1212	t0, t1, t2, t3, t4, ...
650-555-1213	t0, t1, t2, t3, t4, ...
650-555-1214	t0, t1, t2, t3, t4, ...
...	...

次の SQL 文では、telephone_number 列がハッシュ・キーです。ハッシュ・クラスタは、call_timestamp および call_duration 列でソートされます。ハッシュ・キーの数は、10 桁の電話番号に基づいています。

```
CREATE CLUSTER call_detail_cluster (
   telephone_number NUMBER,
   call_timestamp NUMBER SORT,
   call_duration NUMBER SORT )
  HASHKEYS 10000 HASH IS telephone_number
  SIZE 256;

CREATE TABLE call_detail (
   telephone_number     NUMBER,
   call_timestamp       NUMBER   SORT,
   call_duration        NUMBER   SORT,
   other_info           VARCHAR2(30) )
  CLUSTER call_detail_cluster (
   telephone_number, call_timestamp, call_duration );
```

データのソート順を指定すると、次の問合せにより、指定のハッシュ・キーの通話記録が最も古いレコードから順に戻されます。

```
SELECT * WHERE telephone_number = 6505551212;
```

単一表ハッシュ・クラスタの作成

表中の行への高速なアクセスを提供する**単一表ハッシュ・クラスタ**を作成できます。ただし、この表はハッシュ・クラスタ内の唯一の表にする必要があります。ハッシュ・キーとデータ行の間に 1 対 1 のマッピングが必要となるためです。次の文は、クラスタ・キー variety を持つ単一表ハッシュ・クラスタ peanut を作成します。

```
CREATE CLUSTER peanut (variety NUMBER)
   SIZE 512 SINGLE TABLE HASHKEYS 500;
```

HASHKEYS 値は最も近い素数に丸められるため、このクラスタはそれぞれサイズ 512 バイトのハッシュ・キー値を最大 503 個持ちます。SINGLE TABLE 句は、ハッシュ・クラスタにのみ有効です。また、必ず HASHKEYS も指定する必要があります。

> **関連項目：** CREATE CLUSTER 文の構文は、『Oracle Database SQL リファレンス』を参照してください。

ハッシュ・クラスタ内の領域使用の制御

ハッシュ・クラスタを作成するときは、パフォーマンスと使用領域が最適になるように、クラスタ・キーを正しく選択し、HASH IS、SIZE および HASHKEYS の各パラメータを設定することが重要です。次に示すガイドラインでは、これらのパラメータを設定する方法について説明します。

キーの選択

正しいクラスタ・キーの選択は、クラスタ化表に対して最もよく発行される問合せのタイプによって決まります。たとえば、ハッシュ・クラスタ内の emp 表について検討します。問合せが従業員番号によって行を頻繁に選択する場合、最適なクラスタ・キーは empno 列です。問合せが部門によって行を頻繁に選択する場合、最適なクラスタ・キーは deptno 列です。単一の表を含むハッシュ・クラスタの場合は、通常、含まれる表の主キー全体をクラスタ・キーにします。

ハッシュ・クラスタのキーは、索引クラスタのキーと同じように単一列でもコンポジット・キー（複数列キー）でもかまいません。コンポジット・キーによるハッシュ・クラスタは、データベースの内部ハッシュ関数を使用する必要があります。

HASH IS パラメータの設定

HASH IS パラメータを指定するのは、クラスタ・キーが NUMBER データ型の単一の列であり、均一に分布した整数を含む場合のみです。この条件を満たしていれば、それぞれの一意のクラスタ・キー値が衝突（同じハッシュ値を持つクラスタ・キーが 2 つ発生すること）せずに一意のハッシュ値にハッシュするように、クラスタ内に行を分散させることができます。この条件が当てはまらない場合は、この句を指定せずに内部ハッシュ関数を使用してください。

SIZE パラメータの設定

SIZE は、ハッシュ・キーに対応するすべての行を保持するために必要な領域の平均サイズに設定します。そのため、SIZE を適切に決定するには、格納するデータの特性をよく理解する必要があります。

- ハッシュ・クラスタに含まれている表が 1 つで、その表の行のハッシュ・キー値が一意（1 つの値につき 1 行）である場合、SIZE はクラスタ内の平均の行サイズに設定できます。
- ハッシュ・クラスタが複数の表を含む場合、SIZE は代表的なハッシュ値に対応するすべての行を保持するために必要な領域の平均サイズに設定できます。

また、SIZE の見積り値を決定した後で、次の点を考慮してください。SIZE の値が小さい場合（データ・ブロックごとに 5 つ以上のハッシュ・キーを割り当てることができる場合）は、その値を CREATE CLUSTER 文の SIZE に使用できます。しかし、SIZE の値が大きい場合（データ・ブロックごとに割当て可能なハッシュ・キーが 4 つ以下の場合）は、衝突の発生頻度を予測し、データ検索パフォーマンスと領域使用効率のどちらを重視するか検討する必要があります。

- ハッシュ・クラスタで内部ハッシュ関数を使用せず（HASH IS を指定した場合）、衝突がわずかであるか、または衝突のないことが予想される場合は、見積り値をそのまま SIZE に設定できます。この場合、衝突は発生せず、領域は可能なかぎり効率的に使用されます。
- 挿入時に頻繁な衝突が予想される場合、行を格納するためにオーバーフロー・ブロックが割り当てられる可能性は高くなります。衝突が頻繁に起こる場合にブロックのオーバーフローの可能性を軽減し、最大のパフォーマンスを引き出すには、次のように SIZE を調整する必要があります。

ブロック当たりの使用可能領域 / 算出された SIZE	SIZE の設定
1	SIZE
2	SIZE+15%
3	SIZE+12%
4	SIZE+8%
>4	SIZE

ただし、SIZE の値を過大に見積ると、クラスタ内の未使用領域を増やすことになります。領域効率がデータ検索のパフォーマンスよりも重要な場合は、前述の表で示した調整を無視して、SIZE に元の値を使用してください。

HASHKEYS パラメータの設定

ハッシュ・クラスタ内の行を最大限まで分散させるために、データベースは HASHKEYS 値を最も近い素数に丸めます。

ハッシュ・クラスタ内の使用領域の制御

次に示す例では、クラスタ・キーを正しく選択し、HASH IS、SIZE および HASHKEYS の各パラメータを設定する方法を示します。すべての例において、データ・ブロック・サイズは 2KB で、利用可能なデータ領域（ブロック・サイズからオーバーヘッドを差し引いた領域）の平均は各ブロックの 1,950 バイトとします。

ハッシュ・クラスタ内の使用領域の制御 : 例 1 ハッシュ・クラスタに emp 表をロードすることになりました。ほとんどの問合せは、従業員番号によって従業員レコードを検索します。emp 表の最大行数は常に 10,000、平均の行サイズは 55 バイトと見積られています。

この場合は、empno をクラスタ・キーにします。この列には一意の整数が格納されているので、内部ハッシュ関数は無視できます。SIZE は平均の行サイズ（55 バイト）に設定できます。これにより、各データ・ブロックに 34 のハッシュ・キーが割り当てられます。HASHKEYS は、表の行数である 10,000 に設定できます。この値は、10,000 より大きい最初の素数である 10,007 に切り上げられます。

```
CREATE CLUSTER emp_cluster (empno
NUMBER)
. . .
SIZE 55
HASH IS empno HASHKEYS 10000;
```

ハッシュ・クラスタ内の使用領域の制御 : 例 2 前述の例と同様の条件を考えます。ただし、この例では、ほとんどの場合、行が部門番号によって検索されるとします。平均 10 名の従業員を持つ部門が最大で 1,000 部門存在します。部門番号は 10 ずつ増加します（0、10、20、30、...）。

この場合は、deptno をクラスタ・キーにします。この列には一様に分布する整数が格納されているので、内部ハッシュ関数は無視できます。事前に見積った SIZE（各部門のすべての行を保持するために必要な領域の平均サイズ）は 55 × 10 バイト、つまり 550 バイトです。SIZE にこの値を使用して、各データ・ブロックに 3 つのハッシュ・キーのみを割り当てることができます。いくらかの衝突が予想される状況で、最大限のデータ検索パフォーマンスが必要な場合、オーバーフロー・ブロックを必要とする衝突を防ぐために、見積りの SIZE を少し変更してください。ここでは、SIZE を 12% 調整して 620 バイトにします（21-5 ページの「SIZE パラメータの設定」を参照）。これにより、予想される衝突を考慮した上で行の領域をより多く確保できます。

HASHKEYS は、一意の部門番号の数である 1,000 に設定できます。この値は、1,000 より大きい最初の素数である 1,009 に切り上げられます。

```
CREATE CLUSTER emp_cluster (deptno NUMBER)
. . .
SIZE 620
HASH IS deptno HASHKEYS 1000;
```

ハッシュ・クラスタに必要なサイズの見積り

索引クラスタの場合と同じように、ハッシュ・クラスタ内のデータに必要な記憶域を見積ることは重要です。

Oracle Database は、SIZE および HASHKEYS の設定に従って、ハッシュ表の格納に十分な領域の初期割当てを保証します。ハッシュ表サイズを考慮せずに記憶域パラメータ INITIAL、NEXT および MINEXTENTS を設定した場合は、少なくとも SIZE*HASHKEYS に達するまで増分（追加の）エクステントが割り当てられます。たとえば、データ・ブロック・サイズが 2KB、各ブロックの使用可能なデータ領域（ブロック・サイズからオーバーヘッドを差し引いた領域）が約 1,900 バイトの場合に、CREATE CLUSTER 文で STORAGE パラメータと HASH パラメータを次のように指定したとします。

```
STORAGE (INITIAL 100K
    NEXT 150K
    MINEXTENTS 1
    PCTINCREASE 0)
SIZE 1500
HASHKEYS 100
```

この例では、各データ・ブロックにハッシュ・キーを 1 つのみ割り当てることができます。そのため、ハッシュ・クラスタが必要とする初期領域は少なくとも 200KB（100 × 2KB）です。しかし、記憶域パラメータの設定にはこの要件が考慮されていません。そのため、100KB の初期のエクステントと 150KB の増分のエクステントがハッシュ・クラスタに割り当てられます。

一方、HASH パラメータが次のように指定されたとします。

```
SIZE 500 HASHKEYS 100
```

この場合、各データ・ブロックに 3 つのハッシュ・キーが割り当てられます。そのため、ハッシュ・クラスタが必要とする初期領域は少なくとも 68KB（34 × 2KB）です。記憶域パラメータの初期設定はこの要件を満たしているため、100KB の初期エクステントがハッシュ・クラスタに割り当てられます。

ハッシュ・クラスタの変更

ハッシュ・クラスタは、ALTER CLUSTER 文を使用して変更できます。

```
ALTER CLUSTER emp_dept . . . ;
```

ハッシュ・クラスタの変更に関係する問題は、20-7 ページの「クラスタの変更」で説明されている索引クラスタを変更する場合と同じです。ただし、SIZE、HASHKEYS および HASH IS の各パラメータは ALTER CLUSTER 文に指定できません。これらのパラメータを変更するには、クラスタを再作成して、元のクラスタからデータをコピーする必要があります。

ハッシュ・クラスタの削除

ハッシュ・クラスタは、DROP CLUSTER 文を使用して削除できます。

```
DROP CLUSTER emp_dept;
```

ハッシュ・クラスタ内の表は、DROP TABLE 文を使用して削除されます。ハッシュ・クラスタとハッシュ・クラスタ内の表の削除についての問題は、索引クラスタの場合と同じです。

関連項目： 20-8 ページ「クラスタの削除」

ハッシュ・クラスタのデータ・ディクショナリ・ビュー

次のビューには、ハッシュ・クラスタに関する情報が表示されます。

ビュー	説明
DBA_CLUSTERS ALL_CLUSTERS USER_CLUSTERS	DBA ビューには、データベース内のすべてのクラスタ（ハッシュ・クラスタを含む）が表示されます。ALL ビューには、ユーザーがアクセス可能なすべてのクラスタが表示されます。USER ビューは、ユーザーが所有するクラスタのみに制限されます。これらのビューの一部の列には、DBMS_STATS パッケージまたは ANALYZE 文によって生成される統計が含まれます。
DBA_CLU_COLUMNS USER_CLU_COLUMNS	これらのビューでは、表の列とクラスタの列がマップされています。
DBA_CLUSTER_HASH_EXPRESSIONS ALL_CLUSTER_HASH_EXPRESSIONS USER_CLUSTER_HASH_EXPRESSIONS	これらのビューには、ハッシュ・クラスタのハッシュ関数がリストされます。

関連項目： これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。

22

ビュー、順序およびシノニムの管理

この章の内容は次のとおりです。

ビューの管理

ここでは、ビューを管理する方法について説明します。この項の内容は、次のとおりです。

ビューの概要

ビューとは、別の表または表の組合せの論理表現です。ビューのデータは、そのビューの基礎となる表から生成されます。この表を**実表**と呼びます。実表は、実際の表の場合もビュー自体の場合もあります。ビューに対して実行するすべての操作は、実際にはビューの実表に影響します。ビューの使用方法は、表の場合とほぼ同じです。通常の表と同様に、ビューに対する問合せ、更新、挿入および削除ができます。

ビューによって、他の表およびビューに常駐するデータの様々な表現（サブセットまたはスーパーセットなど）が可能です。ビューを使用すると、ユーザーの必要にあわせて調整したデータ表現ができます。

> **関連項目：** ビューの概要については、『Oracle Database 概要』を参照してください。

ビューの作成

ビューを作成するには、次に示す要件を満たす必要があります。

- 自分のスキーマに新しいビューを作成するには、CREATE VIEW システム権限が必要です。別のユーザーのスキーマ内にビューを作成するには、CREATE ANY VIEW システム権限が必要です。これらの権限は明示的に取得するか、またはロールを介して取得できます。
- どのスキーマのビューであっても、その所有者は、ビュー定義で参照されるすべてのオブジェクトにアクセスする権限を明示的に付与されている必要があります。ロールを介してこれらの権限を取得することはできません。また、ビューの機能は、ビューの所有者の権限によって決まります。たとえば、ビューの所有者に Scott の emp 表の INSERT 権限しかない場合、emp 表に新しい行を挿入するためにはビューを使用できますが、このビューの行を選択（SELECT）、更新（UPDATE）または削除（DELETE）するためには使用できません。
- ビューの所有者がビューにアクセスする権限を他のユーザーに付与しようとする場合は、ベース・オブジェクトに対する GRANT OPTION 付きのオブジェクト権限、または ADMIN OPTION 付きのシステム権限が必要です。

ビューを作成するには、CREATE VIEW 文を使用します。ビューはそれぞれ、表、マテリアライズド・ビューまたは他のビューを参照する問合せによって定義されます。すべての副問合せと同様に、ビューを定義する問合せには、FOR UPDATE 句を指定できません。

次の文は、emp 表のデータのサブセットに対してビューを作成します。

```
CREATE VIEW sales_staff AS
      SELECT empno, ename, deptno
      FROM emp
      WHERE deptno = 10
    WITH CHECK OPTION CONSTRAINT sales_staff_cnst;
```

sales_staff ビューを定義する問合せは、部門番号 10 の行のみを参照します。また、CHECK OPTION は、そのビューが選択できない行に対して INSERT および UPDATE 文を発行できないという制約（sales_staff_cnst）付きでビューを作成します。たとえば、次の INSERT 文では、sales_staff ビュー（部門番号が 10 の行のみを含む）によって emp 表に行が正常に挿入されます。

```
INSERT INTO sales_staff VALUES (7584, 'OSTER', 10);
```

しかし、次の INSERT 文は、sales_staff ビューを使用しても選択できない部門番号 30 の行を挿入しようとしているため、エラーが返されます。

```
INSERT INTO sales_staff VALUES (7591, 'WILLIAMS', 30);
```

WITH READ ONLY 句を指定してビューを作成できます。これにより、このビューからは、実表の更新、挿入または削除ができなくなります。WITH 句を指定しない場合、一部の制限を伴いますが、ビューは従来どおり更新可能です。

> **関連項目：** CREATE VIEW 文の構文およびセマンティクスは、『Oracle Database SQL リファレンス』を参照してください。

結合ビュー

FROM 句で複数の実表またはビューを指定するビューを作成することもできます。この種のビューを**結合ビュー**と呼びます。次の文は、emp 表と dept 表のデータを結合する division1_staff ビューを作成します。

```
CREATE VIEW division1_staff AS
      SELECT ename, empno, job, dname
      FROM emp, dept
      WHERE emp.deptno IN (10, 30)
         AND emp.deptno = dept.deptno;
```

更新可能な結合ビューは、UPDATE、INSERT および DELETE 操作が可能な結合ビューです。詳細は、22-6 ページの「結合ビューの更新」を参照してください。

ビュー作成時の問合せ定義の展開

ビューが作成されるときに、Oracle Database は最上位のビュー問合せのワイルドカード（*）を列リストに展開します。結果の問合せはデータ・ディクショナリに格納され、副問合せはそのまま残されます。展開された列リスト中の列名は、引用符で囲まれています。これは、ベース・オブジェクトの列がもともと引用符付きで入力された可能性があり、問合せの構文を正しいものにするためには引用符が必要であることを示しています。

たとえば、dept ビューが次のように作成される場合を想定します。

```
CREATE VIEW dept AS SELECT * FROM scott.dept;
```

データベースは、dept ビューを定義している問合せを次のように格納します。

```
SELECT "DEPTNO", "DNAME", "LOC" FROM scott.dept;
```

エラー付きで作成されたビューでは、ワイルドカードは展開されません。エラーなしでビューがコンパイルされると、定義された問合せのワイルドカードが展開されます。

エラー付きビューの作成

CREATE VIEW 文に構文エラーがない場合は、そのビューを定義している問合せを実行できなくても、データベースはビューを作成できます。この場合、ビューは、エラー付きで作成されたと見なされます。たとえば、存在しない表や既存の表の無効な列を参照するビューを作成するとき、またはビューの所有者が必要な権限を持っていないときでも、ビューを作成し、データ・ディクショナリに登録できます。ただし、そのビューは使用できません。

エラー付きのビューを作成するには、CREATE VIEW 文の FORCE 句を指定する必要があります。

```
CREATE FORCE VIEW AS ...;
```

デフォルトでは、エラー付きのビューは INVALID として作成されます。このようなビューを作成しようとすると、ビューがエラー付きで作成されたことを示すメッセージが返されます。状況が変化して無効なビューの問合せが実行可能になると、ビューは再コンパイルされ、有効（使用可能）になります。条件の変更とビューに及ぼす影響の詳細は、16-17 ページの「オブジェクト依存性の管理」を参照してください。

ビューの置換

ビューを置換するには、ビューの削除および作成に必要なすべての権限が必要です。ビューの定義を変更する場合は、そのビューを置換する必要があり、ビューの定義の変更に ALTER VIEW 文は使用できません。次の方法でビューを置換できます。

- ビューを削除してから再作成します。

  注意： ビューを削除するときに、ロールおよびユーザーに付与された対応するオブジェクト権限はすべて取り消されます。ビューを再作成してから、再度権限を付与してください。

- OR REPLACE 句を含む CREATE VIEW 文によって、ビューを再定義します。OR REPLACE 句は、ビューの現行の定義を置換し、現行のセキュリティ認可を保存します。たとえば、前述のように、sales_staff ビューを作成し、いくつかのオブジェクト権限をロールと他のユーザーに付与した場合を想定します。ただし、ここでは sales_staff ビューを再定義して、WHERE 句に指定されている部門番号を変更するものとします。この場合は、次の文によって、sales_staff ビューの現行バージョンを置換できます。

  ```
  CREATE OR REPLACE VIEW sales_staff AS
       SELECT empno, ename, deptno
       FROM emp
       WHERE deptno = 30
       WITH CHECK OPTION CONSTRAINT sales_staff_cnst;
  ```

ビューを置換する前に、次の影響を検討してください。

- ビューを置換することによって、データ・ディクショナリ内のビュー定義が置換されます。ビューによって参照される基礎となるオブジェクトは影響を受けません。
- 前のビューには CHECK OPTION で制約を定義していたが、新しいビュー定義には指定しない場合、その制約は削除されます。
- 置換されたビューに依存しているビューはすべて無効（使用不可）になります。また、ビューの新しいバージョンでの変更内容によっては、依存している PL/SQL プログラム・ユニットも無効になる場合があります。たとえば、ビューの WHERE 句のみが変更された場合、依存している PL/SQL プログラム・ユニットは有効のままです。ただし、ビューの列数、列名またはデータ型が変更された場合は、依存している PL/SQL プログラム・ユニットも無効になります。データベースでオブジェクトの依存性を管理する方法の詳細は、16-17 ページの「オブジェクト依存性の管理」を参照してください。

問合せでのビューの使用

ビューに対して問合せを発行したり、INSERT、UPDATE または DELETE 文を発行するためには、そのビューに対する SELECT、INSERT、UPDATE または DELETE の各オブジェクト権限を、明示的にまたはロールを介して持っている必要があります。

ビューは、表と同じ方法で問い合せることができます。たとえば、Division1_staff ビューを問い合せるには、そのビューを参照する有効な SELECT 文を入力します。

```
SELECT * FROM Division1_staff;

ENAME        EMPNO       JOB             DNAME
-----------------------------------------------------
CLARK         7782       MANAGER         ACCOUNTING
KING          7839       PRESIDENT       ACCOUNTING
MILLER        7934       CLERK           ACCOUNTING
ALLEN         7499       SALESMAN        SALES
WARD          7521       SALESMAN        SALES
JAMES         7900       CLERK           SALES
TURNER        7844       SALESMAN        SALES
MARTIN        7654       SALESMAN        SALES
BLAKE         7698       MANAGER         SALES
```

一部の制限を伴いますが、ビューを使用して、実表に対する行の挿入、更新または削除ができます。次の文は、sales_staff ビューを使用して emp 表に新しい行を挿入します。

```
INSERT INTO sales_staff
    VALUES (7954, 'OSTER', 30);
```

ビューに対する DML 操作の制限では、次の基準がリストされている順に適用されます。

1. ビューの定義に SET または DISTINCT 演算子、GROUP BY 句またはグループ関数を含む問合せが使用されている場合、そのビューによる実表への行の挿入、更新または削除はできません。
2. ビューの定義に WITH CHECK OPTION が使用されていて、行を実表から選択できない場合、そのビューによる実表への行の挿入または更新はできません。
3. DEFAULT 句を持たない NOT NULL 列がビューから省略されている場合、そのビューによる実表への行の挿入はできません。
4. ビューの作成に DECODE(deptno, 10, "SALES", ...) のような式が使用されている場合、そのビューによる実表への行の挿入または更新はできません。

sales_staff ビューの WITH CHECK OPTION で作成された制約によって許可されるのは、部門番号 30 を持つ行の emp 表への挿入または更新のみです。一方、次の文（つまり、deptno 列を除外する）で sales_staff ビューが定義されるとします。

```
CREATE VIEW sales_staff AS
    SELECT empno, ename
    FROM emp
    WHERE deptno = 10
    WITH CHECK OPTION CONSTRAINT sales_staff_cnst;
```

このビュー定義を検討すると、既存のレコードの empno または ename フィールドは更新できますが、ビューでは deptno フィールドを変更できないため、sales_staff ビューを介して emp 表に行を挿入することはできません。deptno フィールドに DEFAULT 値 10 を定義していた場合は、挿入を実行できます。

ユーザーが無効なビューを参照しようとすると、次のエラー・メッセージが返されます。

```
ORA-04063: ビュー 'view_name' にエラーがあります。
```

このエラー・メッセージが返されるのは、ビューは存在しているが、問合せのエラーが原因で使用禁止状態の場合です（このビューが最初に作成されたときにあったエラーか、またはビューは正常に作成されたが、基礎となるオブジェクトが変更または削除されたために、後で使用禁止状態になったかは関係ありません）。

結合ビューの更新

更新可能な結合ビュー（**変更可能な結合ビュー**と呼ぶこともあります）とは、SELECT 文の最上位の FROM 句に複数の表を含むビューで、WITH READ ONLY 句の制限を受けないものです。

更新可能な結合ビューに関する規則を次の表に示します。これらの基準を満たすビューは、従来どおり更新可能とみなされます。

規則	説明
一般規則	結合ビューに対する INSERT、UPDATE または DELETE 操作は、基礎となる実表を一度に 1 つしか変更できません。
UPDATE 規則	結合ビューの更新可能な列はすべて、**キー保存表**の列にマップする必要があります。キー保存表については、22-7 ページの「キー保存表」を参照してください。ビューの定義に WITH CHECK OPTION 句が使用されている場合は、すべての結合列および繰返し表の列はいずれも更新できません。
DELETE 規則	結合の中にキー保存表が 1 つしかない場合には、結合ビューから行を削除できます。このキー保存表は、FROM 句で繰り返すことができます。ビューの定義に WITH CHECK OPTION 句が使用されていて、キー保存表が繰り返される場合は、そのビューから行を削除できません。
INSERT 規則	INSERT 文では、明示的にも暗黙的にも**非キー保存表**の列を参照しないでください。結合ビューの定義に WITH CHECK OPTION 句が使用されている場合は、INSERT 文を使用できません。

結合ビュー内の列が従来どおり更新可能かどうかを示すデータ・ディクショナリ・ビューがあります。このようなビューについては、22-11 ページの「UPDATABLE_COLUMNS ビューの使用」を参照してください。

注意： 結合ビューが従来どおり更新可能かどうかについては、いくつかのその他の制限と条件が影響します。これらの制限と条件については、『Oracle Database SQL リファレンス』の CREATE VIEW 文の説明を参照してください。

ビューが従来どおり更新可能でない場合は、そのビューに INSTEAD OF トリガーを作成して更新可能にできます。トリガーの詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

また、ビューが別のネストされたビュー上の結合である場合は、そのネストされたビューをトップレベル・ビューにマージ可能である必要があります。マージ可能なビューとマージ不可能なビューの詳細、およびオプティマイザによってビューを参照する文が最適化される方法の概要は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

ここでは、従来どおり更新可能な結合ビューの規則の例を示し、キー保存表について説明します。それぞれの例は、表に主キーと外部キーを明示的に定義した場合、または一意索引を定義した場合にのみ動作します。次の文は、emp および dept の制約が適切に設定された表定義を作成します。

```
CREATE TABLE dept (
      deptno        NUMBER(4) PRIMARY KEY,
      dname         VARCHAR2(14),
      loc           VARCHAR2(13));

CREATE TABLE emp (
      empno        NUMBER(4) PRIMARY KEY,
      ename        VARCHAR2(10),
      job          VARCHAR2(9),
      mgr          NUMBER(4),
      sal          NUMBER(7,2),
      comm         NUMBER(7,2),
      deptno       NUMBER(2),
      FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO));
```

また、前述の例の主キーと外部キーの制約を省略し、dept（deptno）に UNIQUE INDEX を作成した場合でも、後続の例は動作可能です。

次の文は、例で参照される emp_dept 結合ビューを作成します。

```
CREATE VIEW emp_dept AS
      SELECT emp.empno, emp.ename, emp.deptno, emp.sal, dept.dname, dept.loc
      FROM emp, dept
      WHERE emp.deptno = dept.deptno
         AND dept.loc IN ('DALLAS', 'NEW YORK', 'BOSTON');
```

キー保存表

キー保存表の概念は、結合ビューを更新する上での制限を理解するために重要です。表のすべてのキーが結合の結果のキーでもある場合、その表はキー保存になります。つまり、キー保存表とは、結合後もそのキーを保存している表のことです。

> **注意：** 表をキー保存にするために，表の 1 つ以上のキーを選択する必要はありません。1 つ以上のキーを選択した場合に、そのキーが結合の結果のキーであれば十分です。

表のキー保存特性は、表内の実際のデータには依存しません。これは、そのスキーマの特性です。たとえば、emp 表で各部門に多くても 1 人の従業員しか含まれていない場合、emp と dept の結合の結果では deptno は一意ですが、dept はキー保存表ではありません。

emp_dept からすべての行を選択すると、結果は次のようになります。

```
EMPNO      ENAME      DEPTNO  DNAME          LOC
---------- ---------- ------- -------------- -----------
      7782 CLARK           10 ACCOUNTING     NEW YORK
      7839 KING            10 ACCOUNTING     NEW YORK
      7934 MILLER          10 ACCOUNTING     NEW YORK
      7369 SMITH           20 RESEARCH       DALLAS
      7876 ADAMS           20 RESEARCH       DALLAS
      7902 FORD            20 RESEARCH       DALLAS
      7788 SCOTT           20 RESEARCH       DALLAS
      7566 JONES           20 RESEARCH       DALLAS
8 rows selected.
```

このビューでは、empno が emp 表のキー、および結合の結果のキーでもあるため、emp がキー保存表です。deptno は、dept 表のキーですが、結合のキーではないため、dept はキー保存表ではありません。

DML 文と結合ビュー

一般規則では、結合ビューにおいて、UPDATE、INSERT または DELETE 文では、基礎となる実表を 1 つしか変更できません。以降の例は、UPDATE、DELETE および INSERT 文に固有の規則を示しています。

UPDATE 文 次の例は、emp_dept ビューを正常に変更する UPDATE 文を示したものです。

```
UPDATE emp_dept
     SET sal = sal * 1.10
     WHERE deptno = 10;
```

次に示す UPDATE 文は、emp_dept ビューには使用できません。

```
UPDATE emp_dept
     SET loc = 'BOSTON'
     WHERE ename = 'SMITH';
```

この文は dept 実表を変更しようとしますが、dept 表は emp_dept ビューのキー保存表でないため、エラー番号 ORA-01779「キー保存されていない表にマップする列は変更できません」が出力され、文を実行できません。

一般に、結合ビューの更新可能な列はすべて、キー保存表の列にマップする必要があります。ビューの定義に WITH CHECK OPTION 句が使用されている場合は、すべての結合列およびビューで 2 回以上参照されている表から取得したすべての列はいずれも更新できません。

したがって、たとえば emp_dept ビューの定義に WITH CHECK OPTION が使用されている場合、次に示す UPDATE 文は失敗します。

```
UPDATE emp_dept
     SET deptno = 10
     WHERE ename = 'SMITH';
```

結合列を更新しようとするため、この文は失敗となります。

> **関連項目：** UPDATE 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

DELETE 文 結合の中にキー保存表が 1 つしかない場合には、結合ビューから削除できます。このキー保存表は、FROM 句で繰り返すことができます。

次の DELETE 文は、emp_dept ビューに対して動作します。

```
DELETE FROM emp_dept
     WHERE ename = 'SMITH';
```

emp_dept ビューに対するこの DELETE 文は、実表 emp に対する DELETE 操作に変換でき、表 emp は結合ビュー内の唯一のキー保存表であるため、この文は有効です。

次のビューでは、2 つのキー保存表がありますが、同じ表であるため、DELETE 操作を実行できます。つまり、キー保存表が繰り返されています。この場合、この削除文は、FROM リストの最初の表（この例では、e1）で操作されます。

```
CREATE VIEW emp_emp AS
     SELECT e1.ename, e2.empno, e2.deptno
     FROM emp e1, emp e2
     WHERE e1.empno = e2.empno;
```

ビューの定義に WITH CHECK OPTION 句が使用されていて、キー保存表が繰り返される場合は、そのビューから行を削除できません。

```
CREATE VIEW emp_mgr AS
     SELECT e1.ename, e2.ename mname
     FROM emp e1, emp e2
     WHERE e1.mgr = e2.empno
     WITH CHECK OPTION;
```

関連項目： DELETE 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

INSERT 文 emp_dept ビューに対する次の INSERT 文は正常に動作します。

```
INSERT INTO emp_dept (ename, empno, deptno)
   VALUES ('KURODA', 9010, 40);
```

変更されるキー保存実表は 1 つのみであり（emp）、40 は dept 表の有効な deptno であるため（つまり、emp 表に対する FOREIGN KEY 整合性制約を満たすため）、この文は動作します。

次の INSERT 文は、実表 emp に対する UPDATE が失敗するのと同じ理由で失敗します。つまり、77 という deptno が存在しないため、emp 表に対する FOREIGN KEY 整合性制約に違反します。

```
INSERT INTO emp_dept (ename, empno, deptno)
   VALUES ('KURODA', 9010, 77);
```

次の INSERT 文はエラー番号 ORA-01776「結合ビューを介して複数の実表を変更できません。」が出力されて失敗します。

```
INSERT INTO emp_dept (empno, ename, loc)
   VALUES (9010, 'KURODA', 'BOSTON');
```

INSERT は、暗黙的にも明示的にも、非キー保存表の列を参照できません。結合ビューの定義に WITH CHECK OPTION 句が使用されている場合は、その結合ビューに対して INSERT を実行できません。

関連項目： INSERT 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

外部結合が含まれるビューの更新

外部結合が含まれるビューは、変更可能な場合もあります。次に例を示します。

```
CREATE VIEW emp_dept_oj1 AS
    SELECT empno, ename, e.deptno, dname, loc
    FROM emp e, dept d
    WHERE e.deptno = d.deptno (+);
```

文：

```
SELECT * FROM emp_dept_oj1;
```

結果：

```
EMPNO   ENAME      DEPTNO  DNAME          LOC
------- ---------- ------- -------------- -------------
7369    SMITH      40      OPERATIONS     BOSTON
7499    ALLEN      30      SALES          CHICAGO
7566    JONES      20      RESEARCH       DALLAS
7654    MARTIN     30      SALES          CHICAGO
7698    BLAKE      30      SALES          CHICAGO
7782    CLARK      10      ACCOUNTING     NEW YORK
7788    SCOTT      20      RESEARCH       DALLAS
7839    KING       10      ACCOUNTING     NEW YORK
7844    TURNER     30      SALES          CHICAGO
7876    ADAMS      20      RESEARCH       DALLAS
7900    JAMES      30      SALES          CHICAGO
7902    FORD       20      RESEARCH       DALLAS
7934    MILLER     10      ACCOUNTING     NEW YORK
7521    WARD       30      SALES          CHICAGO
14 rows selected.
```

emp_dept_oj1 の emp 実表の列は、emp が結合の中のキー保存表であるため、ビューを介して変更可能です。

次のビューにも外部結合が含まれています。

```
CREATE VIEW emp_dept_oj2 AS
SELECT e.empno, e.ename, e.deptno, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno (+) = d.deptno;
```

文:

```
SELECT * FROM emp_dept_oj2;
```

結果:

```
EMPNO      ENAME      DEPTNO    DNAME          LOC
---------- ---------- --------- -------------- ----
7782       CLARK      10        ACCOUNTING     NEW YORK
7839       KING       10        ACCOUNTING     NEW YORK
7934       MILLER     10        ACCOUNTING     NEW YORK
7369       SMITH      20        RESEARCH       DALLAS
7876       ADAMS      20        RESEARCH       DALLAS
7902       FORD       20        RESEARCH       DALLAS
7788       SCOTT      20        RESEARCH       DALLAS
7566       JONES      20        RESEARCH       DALLAS
7499       ALLEN      30        SALES          CHICAGO
7698       BLAKE      30        SALES          CHICAGO
7654       MARTIN     30        SALES          CHICAGO
7900       JAMES      30        SALES          CHICAGO
7844       TURNER     30        SALES          CHICAGO
7521       WARD       30        SALES          CHICAGO
                                 OPERATIONS     BOSTON
15 rows selected.
```

このビューでは、結合の結果の empno 列が NULL（前述の SELECT 文の最終行）になる可能性があるため、emp はキー保存表ではなくなります。したがって、このビューでは、UPDATE、DELETE および INSERT 操作を実行できません。

別のネストされたビュー上の外部結合を含むビューの場合は、表を含むビューがその外部ビューの最上位まですべてマージされる場合に、その表はキー保存になります。外部結合されているビューは現在、単純な場合にのみマージされます。次に例を示します。

```
SELECT col1, col2, ... FROM T;
```

ビューの選択リストには式がなく、WHERE 句が存在しません。

次のビューのセットを考えてみます。

```
CREATE VIEW emp_v AS
    SELECT empno, ename, deptno
        FROM emp;
CREATE VIEW emp_dept_oj1 AS
    SELECT e.*, Loc, d.dname
        FROM emp_v e, dept d
            WHERE e.deptno = d.deptno (+);
```

これらの例では、emp_v が単純なビューであるため、emp_v は emp_dept_oj1 にマージされ、その結果 emp がキー保存表となります。ただし、次のように emp_v が変更される場合もあります。

```
CREATE VIEW emp_v_2 AS
    SELECT empno, ename, deptno
        FROM emp
            WHERE sal > 1000;
```

この場合は、WHERE 句の存在により、emp_v_2 が emp_dept_oj1 にマージされず、その結果 emp はキー保存表ではなくなります。

ビューが更新可能かどうか不確かな場合は、USER_UPDATABLE_COLUMNS ビューから選択することで確認できます。次に例を示します。

```
SELECT owner, table_name, column_name, updatable FROM USER_UPDATABLE_COLUMNS
     WHERE TABLE_NAME = 'EMP_DEPT_VIEW';
```

ここでは、次のような出力が返されます。

```
OWNER       TABLE_NAME      COLUMN_NAM      UPD
----------  ----------      ----------      ---
SCOTT       EMP_DEPT_V      EMPNO           NO
SCOTT       EMP_DEPT_V      ENAME           NO
SCOTT       EMP_DEPT_V      DEPTNO          NO
SCOTT       EMP_DEPT_V      DNAME           NO
SCOTT       EMP_DEPT_V      LOC             NO
5 rows selected.
```

UPDATABLE_COLUMNS ビューの使用

次の表に示すビューは、従来どおり更新可能な結合ビューを識別する際に利用できます。

ビュー	説明
DBA_UPDATABLE_ COLUMNS	変更可能なすべての表とビューのすべての列が表示されます。
ALL_UPDATABLE_ COLUMNS	ユーザーがアクセス可能で変更可能なすべての表とビューのすべての列が表示されます。
USER_UPDATABLE_ COLUMNS	ユーザーのスキーマ内で変更可能なすべての表とビューのすべての列が表示されます。

次に、emp_dept ビュー内の更新可能な列を示します。

```
SELECT COLUMN_NAME, UPDATABLE
      FROM USER_UPDATABLE_COLUMNS
      WHERE TABLE_NAME = 'EMP_DEPT';

COLUMN_NAME                    UPD
------------------------------ ---
EMPNO                          YES
ENAME                          YES
DEPTNO                         YES
SAL                            YES
DNAME                          NO
LOC                            NO

6 rows selected.
```

関連項目： 更新可能な列のビューの詳細は、『Oracle Database リファレンス』を参照してください。

ビューの変更

ALTER VIEW 文は、無効なビューを明示的に再コンパイルする場合にのみ使用します。ビューの定義を変更する場合は、22-4 ページの「ビューの置換」を参照してください。

ALTER VIEW 文を使用すると、実際に使用する前に再コンパイル・エラーを調べることができます。ビューの実表の 1 つを変更したとき、変更がビューまたはそれに依存する他のオブジェクトに影響しないようにするには、そのビューを明示的に再コンパイルします。

ALTER VIEW 文を使用するには、そのビューが自分のスキーマに含まれているか、または ALTER ANY TABLE システム権限を持っている必要があります。

関連項目： ALTER VIEW 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

ビューの削除

自分のスキーマにあるビューはすべて削除できます。別のユーザーのスキーマ内にあるビューを削除するには、DROP ANY VIEW システム権限が必要です。ビューを削除するには、DROP VIEW 文を使用します。たとえば、次の文は emp_dept ビューを削除します。

```
DROP VIEW emp_dept;
```

関連項目： DROP VIEW 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

順序の管理

ここでは、順序の管理について説明します。この項の内容は、次のとおりです。

順序の概要

順序とは、複数のユーザーが一意の整数を生成するために使用できるデータベース・オブジェクトです。シーケンス・ジェネレータは順序番号を生成し、一意主キーの自動的な生成、および複数の行または表の間のキーの調整に役立ちます。

順序がないと、シーケンシャル値はプログラムによって生成されるのみです。新しい主キー値は、最後に生成された値を選択し、その値を増分することによって取得できます。この方法では、トランザクション中のロックが必要となるため、複数のユーザーが主キーの次の値を待機することになります。この待機を**シリアライズ**と呼びます。開発者がこのような構成メンバーをアプリケーションに設定している場合は、順序へのアクセスに置き換えるように薦めてください。順序によってシリアライズが解消され、アプリケーションの同時実行性が改善されます。

関連項目： 順序の概要については、『Oracle Database 概要』を参照してください。

順序の作成

自分のスキーマに順序を作成するには、CREATE SEQUENCE システム権限が必要です。別のユーザーのスキーマ内に順序を作成するには、CREATE ANY SEQUENCE 権限が必要です。

順序を作成するには、CREATE SEQUENCE 文を使用します。たとえば、次の文は、emp 表の empno 列に対して従業員番号を生成するために使用する順序を作成します。

```
CREATE SEQUENCE emp_sequence
      INCREMENT BY 1
      START WITH 1
      NOMAXVALUE
      NOCYCLE
      CACHE 10;
```

複数のパラメータを指定して、順序の機能を制御できます。これらのパラメータを使用して、順序の昇順または降順、順序の開始点、最大値と最小値、および順序値の間隔を指定できます。NOCYCLE オプションは、順序が最大値または最小値に達すると、それ以上の値を生成できなくなることを示します。

CACHE 句は順序番号により高速にアクセスできるように、順序番号の集合をメモリーに事前割当てし、維持します。キャッシュ内の最後の順序番号が使用されると、別の順序の集合がキャッシュ内に読み込まれます。

順序番号の集合をキャッシュする場合に、順序番号がスキップされることがあります。たとえば、インスタンスが異常停止すると（たとえばインスタンス障害が発生したり、SHUTDOWN ABORT 文が発行されたりすると）、キャッシュされているが使用されていない順序番号は失われます。また、使用されても保存されなかった順序番号も失われます。さらに、エクスポートとインポートの後、データベースがキャッシュされた順序番号をスキップすることもあります。詳細は、『Oracle Database ユーティリティ』を参照してください。

関連項目：

- CREATE SEQUENCE 文の構文については、『Oracle Database SQL リファレンス』を参照してください。
- Oracle Real Application Clusters 環境での順序番号のキャッシュによるパフォーマンス向上の詳細は、『Oracle Real Application Clusters 配置およびパフォーマンス』を参照してください。

順序の変更

順序を変更するには、その順序が自分のスキーマに含まれているか、または ALTER ANY SEQUENCE システム権限を持っている必要があります。順序を変更することにより、順序番号の生成方法を定義するパラメータを変更できます。ただし、順序の開始番号は変更できません。順序の開始点を変更するには、順序を削除してから、再作成します。

順序を変更するには、ALTER SEQUENCE 文を使用します。たとえば、次の文は、emp_sequence を変更します。

```
ALTER SEQUENCE emp_sequence
    INCREMENT BY 10
    MAXVALUE 10000
    CYCLE
    CACHE 20;
```

関連項目： ALTER SEQUENCE 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

順序の使用

順序を使用するには、順序が自分のスキーマに含まれているか、または別のユーザーの順序に対する SELECT オブジェクト権限が付与されている必要があります。定義済の順序は、複数のユーザー（該当する順序を含む順序に対する SELECT オブジェクト権限を持つユーザー）が待機せずにアクセスおよび増分できます。順序を増分したトランザクションの完了を待機せずに、その順序は再び増分されます。

次の各項の例では、マスター / ディテール表の関係における順序の使用方法について説明します。顧客からの注文の情報を格納する 2 つの表 orders_tab（マスター表）および line_items_tab（ディテール表）で部分的に構成される注文入力システムを想定します。order_seq という名前の順序が、次の文で定義されます。

```
CREATE SEQUENCE Order_seq
    START WITH 1
    INCREMENT BY 1
    NOMAXVALUE
    NOCYCLE
    CACHE 20;
```

順序の参照

順序は、NEXTVAL および CURRVAL 疑似列を使用して、SQL 文内で参照できます。それぞれの新しい順序番号は、順序の疑似列 NEXTVAL への参照によって生成され、現行の順序番号は、疑似列 CURRVAL を使用して繰り返し参照できます。

NEXTVAL および CURRVAL は、予約語またはキーワードではなく、SELECT、INSERT または UPDATE のような SQL 文の中で疑似列名として使用できます。

NEXTVAL を使用した順序番号の生成 順序番号を生成して使用するには、*seq_name*.NEXTVAL を参照します。たとえば、顧客からの発注を受けると想定します。順序番号は、値のリストの中で参照できます。次に例を示します。

```
INSERT INTO Orders_tab (Orderno, Custno)
    VALUES (Order_seq.NEXTVAL, 1032);
```

または、順序番号は、UPDATE 文の SET 句の中で参照できます。次に例を示します。

```
UPDATE Orders_tab
    SET Orderno = Order_seq.NEXTVAL
    WHERE Orderno = 10112;
```

順序番号は、問合せまたは副問合せの SELECT の最も外側でも参照できます。次に例を示します。

```
SELECT Order_seq.NEXTVAL FROM dual;
```

定義されているように、order_seq.NEXTVAL への最初の参照が値 1 を返します。order_seq.NEXTVAL を参照する後続の各文は、次の順序番号（2、3、4、...）を生成します。疑似列 NEXTVAL は、必要に応じた数の新しい順序番号を生成するために使用されます。ただし、各行に生成される順序番号は 1 つのみです。つまり、1 つの文で NEXTVAL が複数回参照される場合、最初の参照によって次の番号が生成され、文の中のすべての後続の参照によって同じ番号が返されます。

生成された順序番号は、その番号を生成したセッションに対してのみ使用可能です。トランザクションのコミットまたはロールバックには関係なく、order_seq.NEXTVAL を参照する他のユーザーは一意の値を取得します。2 人のユーザーが同時に同じ順序にアクセスしている場合、各ユーザーが受け取る順序番号に食い違いが生じる可能性があります。これは、他のユーザーも順序番号を生成しているためです。

CURRVAL を使用した順序番号の使用 自分のセッションの現在の順序値を使用または参照するには、*seq_name*.CURRVAL を参照します。現行のユーザー・セッション（現行または既存のトランザクション内）で *seq_name*.NEXTVAL が参照された場合のみ、CURRVAL が使用されます。CURRVAL は、同一文内で複数回の場合を含め、必要な回数だけ参照できます。次の順序番号は、NEXTVAL が参照されると生成されます。前述の例を続けると、注文の明細項目を挿入することで、顧客からの発注を完了します。

```
INSERT INTO Line_items_tab (Orderno, Partno, Quantity)
    VALUES (Order_seq.CURRVAL, 20321, 3);

INSERT INTO Line_items_tab (Orderno, Partno, Quantity)
    VALUES (Order_seq.CURRVAL, 29374, 1);
```

前項で指定した INSERT 文が、新しい順序番号 347 を生成した場合、この項の文で挿入された両方の行は注文番号 347 の行を挿入します。

NEXTVAL および CURRVAL の使用と制限事項 CURRVAL および NEXTVAL は、次の場所で使用できます。

- INSERT 文の VALUES 句
- SELECT 文の SELECT リスト
- UPDATE 文の SET 句

CURRVAL および NEXTVAL は、次の場所では使用できません。

- 副問合せ
- ビューの問合せまたはマテリアライズド・ビューの問合せ
- DISTINCT 演算子を指定した SELECT 文
- GROUP BY または ORDER BY 句を指定した SELECT 文
- UNION、INTERSECT または MINUS 集合演算子を指定した別の SELECT 文と組み合せた SELECT 文
- SELECT 文の WHERE 句
- CREATE TABLE または ALTER TABLE 文の中の列の DEFAULT 値
- CHECK 制約の条件

順序番号のキャッシュ

順序番号は、システム・グローバル領域（SGA）内の順序キャッシュに保持できます。順序キャッシュ内の順序番号へのアクセスは、順序番号をディスクから読み込むより高速です。

順序キャッシュは、複数のエントリで構成されています。各エントリには、単一の順序の順序番号を多数保持できます。

すべての順序番号に高速にアクセスするには、次のガイドラインに従ってください。

- 順序キャッシュが、アプリケーションによって同時に使用されるすべての順序を保持できるようにしてください。
- 順序キャッシュ内に保持された各順序の値の数を増やしてください。

順序キャッシュ内のエントリの数 アプリケーションが順序キャッシュ内の順序にアクセスする場合、順序番号は高速に読み込まれます。ただし、アプリケーションがキャッシュ内にない順序にアクセスする場合は、順序番号が使用される前に、順序がディスクからキャッシュに読み込まれる必要があります。

アプリケーションが多数の順序を同時に使用する場合は、順序キャッシュの大きさが不足して、すべての順序を保持できないことがあります。このような場合、順序番号へのアクセスにはディスク読取りが頻繁に必要になります。すべての順序に高速にアクセスするには、キャッシュに十分なエントリを用意し、アプリケーションが同時に使用するすべての順序を保持してください。

各順序キャッシュ・エントリ内の値の数 順序が順序キャッシュに読み込まれるとき、キャッシュ・エントリに順序値が生成および格納されます。その後、これらの値に高速にアクセスできます。キャッシュに格納される順序値の数は、CREATE SEQUENCE 文の CACHE パラメータによって決まります。このパラメータのデフォルト値は 20 です。

次の CREATE SEQUENCE 文は seq2 順序を作成し、SEQUENCE キャッシュ内に 50 個の順序値を格納します。

```
CREATE SEQUENCE seq2
    CACHE 50;
```

次に、seq2 の最初の 50 個の値がキャッシュから読み込まれます。51 番目の値がアクセスされると、次の 50 個の値がディスクから読み込まれます。

CACHE に上限を選択することにより、ディスクから順序キャッシュへの読取りを減らし、より長く連続する順序番号にアクセスできます。ただし、インスタンス障害が発生した場合は、キャッシュ内のすべての順序値が失われます。エクスポートの実行中にトランザクションが順序番号へのアクセスを続ける場合、エクスポートとインポートの後で、キャッシュされた順序番号がスキップされることもあります。

CREATE SEQUENCE 文の中で NOCACHE オプションを使用する場合、順序値は順序キャッシュに格納されません。この場合は、順序にアクセスするたびにディスク読取りが必要となります。このようなディスク読取りにより、順序へのアクセスが遅くなります。次の CREATE SEQUENCE 文は SEQ3 順序を作成しますが、その値をキャッシュ内に格納しません。

```
CREATE SEQUENCE seq3
    NOCACHE;
```

順序の削除

自分のスキーマ内の順序はどれでも削除できます。別のスキーマ内の順序を削除するには、DROP ANY SEQUENCE システム権限が必要です。不要になった順序は、DROP SEQUENCE 文を使用して削除できます。たとえば、次の文は order_seq 順序を削除します。

```
DROP SEQUENCE order_seq;
```

順序を削除すると、その定義がデータ・ディクショナリから削除されます。順序のシノニムはそのまま残りますが、参照時にエラーが返されます。

> **関連項目：** DROP SEQUENCE 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

シノニムの管理

ここでは、シノニムの管理について説明します。この項の内容は、次のとおりです。

シノニムの概要

シノニムは、スキーマ・オブジェクトの別名です。シノニムは、オブジェクトの名前と所有者をマスクし、分散データベースのリモート・オブジェクトに位置透過性を提供することによって、あるレベルのセキュリティを提供します。また、SQL 文で使用できるため、データベース・ユーザーにとって SQL 文の複雑さが軽減されます。

シノニムを使用することで、基礎となるオブジェクトの名前変更または移動が可能になります。その場合、シノニムの再定義のみで、そのシノニムに基づくアプリケーションは変更しなくてもそのまま機能します。

パブリック・シノニムとプライベート・シノニムの両方を作成できます。**パブリック・シノニム**は PUBLIC という名前の特別なユーザー・グループによって所有され、データベース内のすべてのユーザーがアクセスできます。**プライベート・シノニム**は、特定のユーザーのスキーマ内に含まれており、そのユーザーおよび基礎となるオブジェクトの権限受領者のみが使用できます。

シノニム自体は安全ではありません。シノニムのオブジェクト権限を付与した場合、実際には基礎となるオブジェクトの権限を付与することになり、そのシノニムは GRANT 文でオブジェクトの別名としてのみ機能します。

> **関連項目：** シノニムの詳細は、『Oracle Database 概要』を参照してください。

シノニムの作成

自分のスキーマに新しいプライベート・シノニムを作成するには、CREATE SYNONYM システム権限が必要です。別のユーザーのスキーマ内にプライベート・シノニムを作成するには、CREATE ANY SYNONYM 権限が必要です。パブリック・シノニムを作成するには、CREATE PUBLIC SYNONYM システム権限が必要です。

シノニムを作成するには、CREATE SYNONYM 文を使用します。CREATE SYNONYM 文の正常な実行に、基礎となるスキーマ・オブジェクトは必要ありません。また、そのオブジェクトにアクセスする権限も不要です。次の文は、jward のスキーマに含まれる emp 表のパブリック・シノニム PUBLIC_EMP を作成します。

```
CREATE PUBLIC SYNONYM public_emp FOR jward.emp
```

リモート・プロシージャまたはファンクションのシノニムを作成する場合は、そのリモート・プロシージャまたはファンクションのスキーマ名でリモート・オブジェクトを修飾する必要があります。また、リモート・オブジェクトが存在するデータベースに、ローカル・パブリック・シノニムを作成できます。この場合は、プロシージャまたはファンクションへの後続のすべてのコールに、そのデータベース・リンクを含める必要があります。

> **関連項目：** CREATE SYNONYM 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

DML 文でのシノニムの使用

基礎となるオブジェクトへのアクセスに必要な権限が、明示的に、使用可能なロールから、または PUBLIC から付与されている場合は、自分のスキーマまたはパブリック・シノニムに含まれるプライベート・シノニムを正常に使用できます。また、基礎となるオブジェクトについて必要なオブジェクト権限が付与されている場合は、別のスキーマ内のプライベート・シノニムを参照することもできます。

付与されているオブジェクト権限のみを使用して、別のユーザーのシノニムを参照できます。たとえば、jward.emp 表に対する SELECT 権限のみがあるときに、シノニム jward.employee が jward.emp に対して作成された場合は、jward.employee シノニムの問合せはできますが、jward.employee シノニムを使用した行の挿入はできません。

シノニムは、そのシノニムの基礎となるオブジェクトを参照するのと同じ方法で、DML 文で参照できます。たとえば、employee というシノニムが表またはビューを参照する場合は、次の文が有効になります。

```
INSERT INTO employee (empno, ename, job)
    VALUES (emp_sequence.NEXTVAL, 'SMITH', 'CLERK');
```

fire_emp というシノニムがスタンドアロン・プロシージャまたはパッケージ・プロシージャを参照する場合は、それをコマンドで実行できます。

```
EXECUTE Fire_emp(7344);
```

シノニムの削除

自分のスキーマ内のプライベート・シノニムはどれでも削除できます。別のユーザーのスキーマ内にあるプライベート・シノニムを削除するには、DROP ANY SYNONYM システム権限が必要です。パブリック・シノニムを削除するには、DROP PUBLIC SYNONYM システム権限が必要です。

不要になったシノニムを削除するには、DROP SYNONYM 文を使用します。プライベート・シノニムを削除する場合は、PUBLIC キーワードを省略します。パブリック・シノニムを削除する場合は、PUBLIC キーワードを指定します。

たとえば、次の文はプライベート・シノニム emp を削除します。

```
DROP SYNONYM emp;
```

次の文は、パブリック・シノニム public_emp を削除します。

```
DROP PUBLIC SYNONYM public_emp;
```

シノニムを削除すると、その定義がデータ・ディクショナリから削除されます。削除したシノニムを参照するオブジェクトはすべて残ります。ただし、それらのオブジェクトは無効（使用不可）になります。シノニムの削除が他のスキーマ・オブジェクトに与える影響の詳細は、「オブジェクト依存性の管理」を参照してください。

> **関連項目：** DROP SYNONYM 文の構文と詳細は、『Oracle Database SQL リファレンス』を参照してください。

ビュー、順序およびシノニムのデータ・ディクショナリ・ビュー

次のビューには、ビュー、シノニムおよび順序に関する情報が表示されます。

ビュー	説明
DBA_VIEWS ALL_VIEWS USER_VIEWS	DBA ビューには、データベース内のすべてのビューが表示されます。ALL ビューは、現行ユーザーがアクセスできるビューのみに制限されます。USER ビューは、現行ユーザーが所有するビューのみに制限されます。
DBA_SYNONYMS ALL_SYNONYMS USER_SYNONYMS	これらのビューには、シノニムが表示されます。
DBA_SEQUENCES ALL_SEQUENCES USER_SEQUENCES	これらのビューには、順序が表示されます。
DBA_UPDATABLE_COLUMNS ALL_UPDATABLE_COLUMNS USER_UPDATABLE_COLUMNS	これらのビューには、更新可能な結合ビューの列がすべて表示されます。

関連項目： これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。

23

破損データの修復

この章の内容は次のとおりです。

> **注意：** DBMS_REPAIR パッケージについて詳しくない場合は、このパッケージに含まれる修復プロシージャを実行する際に、Oracle サポート・サービスのアナリストと共同で作業することをお薦めします。

データ・ブロック破損を修復するオプション

Oracle Database には、データ・ブロックの破損を検出して修正するために、複数の方法が用意されています。その 1 つは、破損の検出後にオブジェクトを削除して再作成することです。しかし、この方法が必ずしも可能とはかぎらず、またそれが望ましくない場合もあります。データ・ブロックの破損が行のサブセットにかぎられている場合は、破損した行を除くすべてのデータを選択して表を再作成する方法があります。

また、DBMS_REPAIR パッケージを使用してデータ・ブロック破損を管理する方法もあります。DBMS_REPAIR を使用すると、表と索引の破損ブロックを検出して修復できます。オブジェクトは、再作成または修復の試行中でも続けて使用できます。

注意： データの損失を伴う破損の場合は、そのデータがデータベース・システム全体にどのように格納されているかを分析して理解する必要があります。修復の内容によっては、データを失ったり、論理的一貫性が損なわれる場合があります。このパッケージで提供される修復アプローチが特定の破損に対して適切かどうかを個々に判断する必要があります。

DBMS_REPAIR パッケージの内容

ここでは、DBMS_REPAIR パッケージに含まれているプロシージャと、その使用に伴う制約および制限事項について説明します。

関連項目： DBMS_REPAIR プロシージャの構文、制限事項および例外の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

DBMS_REPAIR プロシージャ

次の表は、DBMS_REPAIR パッケージに含まれているプロシージャの一覧を示します。

プロシージャ名	説明
ADMIN_TABLES	修復表および孤立キー表の管理機能（作成、削除、パージ）を提供します。 **注意：**これらの表は常に SYS スキーマに作成されます。
CHECK_OBJECT	表または索引内の破損を検出してレポートします。
DUMP_ORPHAN_KEYS	破損データ・ブロック内の行を指す索引エントリをレポートします。
FIX_CORRUPT_BLOCKS	すでに CHECK_OBJECT プロシージャで破損ブロックとして識別されているブロックにソフトウェア破損を示すマークを付けます。
REBUILD_FREELISTS	オブジェクトの空きリストを再作成します。
SEGMENT_FIX_STATUS	セグメント領域管理が AUTO の場合に、ビットマップ・エントリの破損状態を修正する機能を提供します。
SKIP_CORRUPT_BLOCKS	このプロシージャを使用すると、表と索引のスキャン時に、破損マークが付いたブロックが無視されます。使用しない場合は、破損マークが付いたブロックが検出されたときにエラー ORA-1578 が返されます。

これらのプロシージャの詳細と使用例は、23-6 ページの「DBMS_REPAIR の例」を参照してください。

制約と制限事項

DBMS_REPAIR プロシージャには、次の制約があります。

- LOB データ型、ネストした表および VARRAY を含む表はサポートされますが、表外格納の列は無視されます。
- クラスタは、SKIP_CORRUPT_BLOCKS および REBUILD_FREELISTS プロシージャではサポートされますが、CHECK_OBJECT プロシージャではサポートされません。
- 索引構成表および LOB 索引はサポートされません。
- DUMP_ORPHAN_KEYS プロシージャは、ビットマップ索引またはファンクション索引には機能しません。
- DUMP_ORPHAN_KEYS プロシージャで処理される最大キー長は、3,950 バイトです。

DBMS_REPAIR パッケージの使用方法

データ・ブロック破損に対処する手段として DBMS_REPAIR を検討する場合は、次のアプローチに従うことをお薦めします。

- タスク 1: 破損の検出とレポート
- タスク 2: DBMS_REPAIR の使用に伴うコストと利点の評価
- タスク 3: オブジェクトの使用可能化
- タスク 4: 破損の修復および失われたデータの再作成

タスク 1: 破損の検出とレポート

最初のタスクでは、破損を検出しレポートします。レポートでは、ブロックに関する問題が明らかになるだけでなく、それに対応する修復ディレクティブも識別されます。破損を検出する方法はいくつかあります。表 23-1 に様々な検出方法を示します。

表 23-1 破損検出方法の比較

検出方法	説明
DBMS_REPAIR PL/SQL パッケージ	指定した表、パーティションまたは索引のブロック・チェックが実行されます。修復表に結果が移入されます。
DB_VERIFY ユーティリティ	オフライン・データベースでブロック・チェックを実行します。
ANALYZE TABLESQL 文	VALIDATE STRUCTURE オプションを指定すると、索引、表またはクラスタの構造の整合性が ANALYZE TABLE 文によって検証され、表と索引が同期しているかどうかがチェックまたは検証されます。
DB_BLOCK_CHECKING 初期化パラメータ	DB_BLOCK_CHECKING=TRUE の場合は、実際に破損マークを付ける前に、破損ブロックが識別されます。チェックは、ブロックの変更時に実行されます。

DBMS_REPAIR: CHECK_OBJECT および ADMIN_TABLES プロシージャの使用

CHECK_OBJECT プロシージャは、指定されたオブジェクトのブロック破損をチェックしてレポートします。索引と表に対する ANALYZE...VALIDATE STRUCTURE 文と同様に、索引とデータ・ブロックに対してブロック・チェックが実行されます。

CHECK_OBJECT では、破損がレポートされるだけでなく、そのオブジェクトに対して後で FIX_CORRUPT_BLOCKS を実行した場合に行われる修正も識別されます。この情報は修復表への移入によって使用可能になるため、最初に ADMIN_TABLES プロシージャで修復表を作成しておく必要があります。

CHECK_OBJECT プロシージャを実行した後は、修復表の簡単な問合せによってそのオブジェクトの破損および修復ディレクティブが表示されます。この情報に基づいて、レポートされた問題に最も適切な対処方法を評価できます。

DB_VERIFY: オフライン・データベース・チェックの実行

データ破損が発生した場合は、オフライン診断ユーティリティとして DB_VERIFY を使用します。

> **関連項目：** DB_VERIFY の詳細は、『Oracle Database ユーティリティ』を参照してください。

ANALYZE: 破損のレポート

ANALYZE TABLE...VALIDATE STRUCTURE 文は、分析するオブジェクトの構造の妥当性をチェックします。オブジェクトの構造内で破損が検出されると、エラー・メッセージが表示されます。この場合は、オブジェクトを削除して再作成する必要があります。

ANALYZE TABLE 文の CASCADE 句を使用すると、1 回の操作で、表とすべての索引の構造をチェックできます。この操作ではリソースを大量に消費する可能性があるため、軽量なチェックを実行する FAST オプションを使用できます。詳細は、16-4 ページの「表、索引、クラスタおよびマテリアライズド・ビューの妥当性チェック」を参照してください。

> **関連項目：**
>
> - ANALYZE 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

DB_BLOCK_CHECKING 初期化パラメータ

DB_BLOCK_CHECKING 初期化パラメータを TRUE に設定すると、データベースのブロック・チェックを使用可能にすることができます。これにより、データ・ブロックおよび索引ブロックが変更された際には、必ずそのブロックの内部一貫性がチェックされます。
DB_BLOCK_CHECKING は、ALTER SYSTEM SET 文で変更可能な動的パラメータです。システム表領域では、ブロック・チェックは常に使用可能になっています。

> **関連項目：** DB_BLOCK_CHECKING 初期化パラメータの詳細は、『Oracle Database リファレンス』を参照してください。

タスク 2: DBMS_REPAIR の使用に伴うコストと利点の評価

DBMS_REPAIR を使用する前に、その利害得失を検討する必要があります。また、破損オブジェクトの対応手段として使用可能な他のオプションも検討してください。次の質問に答えることから開始してください。

- 破損の範囲はどの程度ですか。

  破損の有無と修復アクションの要不要を判断するには、CHECK_OBJECT プロシージャを実行して修復表を問い合せます。

- ブロック破損の対応手段として使用可能な他のオプションがありますか。この質問については、次の対応が可能かどうかを検討します。
  - 他のソースからのデータが使用可能な場合は、そのオブジェクトを削除し、再作成して再移入する。
  - CREATE TABLE...AS SELECT 文を発行して、破損表から新しい表を作成する。
  - SELECT 文から破損行を除外して、破損を無視する。
  - メディア・リカバリを実行する。
- DBMS_REPAIR を使用してオブジェクトを使用可能にした場合に、どのような論理的な破損や副作用が生じますか。それらの問題に対処できますか。そのためにはどんな作業が必要ですか。

  破損マークが付いたブロックの行にアクセスできない場合があります。また、正常にアクセスできる行が含まれているブロックでも、破損マークが付いている場合があります。

ブロックに破損マークが付いている場合は、参照整合性制約が壊れていることがあります。この場合は、制約を使用禁止にし、再び使用可能にすると、不整合がレポートされます。すべての問題を解決すれば、再び制約を使用できるようになります。

表にトリガーが定義されている場合は、論理的な破損が生じることがあります。たとえば、行を再度挿入したときに、挿入トリガーが起動されるかどうかを確認します。これらの問題に対処するには、インストレーションでどのトリガーがどのように使用されているかを理解する必要があります。

索引と表が同期化されていない場合は、DUMP_ORPHAN_KEYS プロシージャを実行して、破損データの再作成に役立つ情報をキーから取得します。次に、ALTER INDEX...REBUILD ONLINE 文を発行し、表と索引を同期化します。

- 修復によってデータが失われる場合に、このデータを取り出すことができますか。

  データ・ブロックに破損マークが付いている場合は、索引からデータを取り出すことができます。この情報を取り出すには、DUMP_ORPHAN_KEYS プロシージャを利用します。

タスク 3: オブジェクトの使用可能化

DBMS_REPAIR を使用して表と索引のスキャン時に破損を無視することにより、オブジェクトを使用可能にします。

破損の修復 : FIX_CORRUPT_BLOCKS および SKIP_CORRUPT_BLOCKS プロシージャの使用

DBMS_REPAIR の機能の適用範囲外にある破損をスキップする環境を設定し、それによって破損オブジェクトを使用可能にできます。

データ・ブロック内の不良行のように、破損によってデータが失われている場合は、FIX_CORRUPT_BLOCKS プロシージャを使用して、この種のすべてのブロックに破損マークを設定します。次に、SKIP_CORRUPT_BLOCKS プロシージャを実行します。このプロシージャは、破損マークが付いているブロックをスキップするように設定します。プロシージャの SKIP_FLAG パラメータを設定すると、表および索引のスキャン時に、破損マークが付いたすべてのブロックがスキップされます。これは、メディア破損ブロックとソフトウェア破損ブロックのどちらにも適用されます。

破損ブロックをスキップする操作の意味

索引と表が同期化されていない場合、ある問合せで索引のみをプローブし、後続の問合せで索引と表の両方をプローブするような状況下では、SET TRANSACTION READ ONLY トランザクションの一貫性が保たれないことがあります。表ブロックに破損マークが付いている場合、この 2 つの問合せは異なる結果を返すので、読取り専用トランザクションのルールに違反します。この場合の対処方法の 1 つとして、SET TRANSACTION READ ONLY トランザクション内で破損をスキップしないようにします。

これと同様の問題は、連鎖している行の選択時にも発生します。同じ行を問い合せても、破損にアクセスできる場合とできない場合があるため、異なった結果が生じます。

タスク 4: 破損の修復および失われたデータの再作成

オブジェクトを使用可能にした後で、次の修復アクティビティを実行します。

DUMP_ORPHAN_KEYS プロシージャを使用したデータのリカバリ

DUMP_ORPHAN_KEYS プロシージャは、破損データ・ブロック内の行を指す索引エントリをレポートします。この種の索引エントリがすべて、破損のキーと ROWID を格納する孤立キー表に挿入されます。

索引エントリ情報を取り出した後、ALTER INDEX...REBUILD ONLINE 文を使用して索引を再作成できます。

SEGMENT_FIX_STATUS プロシージャを使用したセグメント・ビットマップの修正

セグメントの空き領域がビットマップで管理されている場合（SEGMENT SPACE MANAGEMENT AUTO）は、このプロシージャを使用します。

このプロシージャは、対応するブロックの現在の内容に基づいてビットマップ・エントリの状態を再計算します。また、ビットマップ・エントリを特定の値に設定するように指定することもできます。通常は、状態が適切に再計算されるので、値を強制的に設定する必要はありません。

DBMS_REPAIR の例

この項の内容は、次のとおりです。

例 : 修復表または孤立キー表の作成

ADMIN_TABLE プロシージャは、修復表または孤立キー表の作成、パージまたは削除に使用します。

修復表は、CHECK_OBJECT プロシージャによって検出された破損の内容と、FIX_CORRUPT_BLOCKS プロシージャを実行した場合にこれらの破損がどのように処理されるかを示す情報を提供します。また、FIX_CORRUPT_BLOCKS プロシージャの実行が必要かを判断する際にも使用されます。

孤立キー表は、DUMP_ORPHAN_KEYS プロシージャの実行時に使用され、破損行を指す索引エントリが格納されます。DUMP_ORPHAN_KEYS プロシージャは、そのアクティビティをロギングし、索引情報を使用可能な形にして、孤立キー表に移入します。

例 : 修復表の作成

次の例では、users 表領域の修復表を作成しています。

```
BEGIN
DBMS_REPAIR.ADMIN_TABLES (
     TABLE_NAME => 'REPAIR_TABLE',
     TABLE_TYPE => dbms_repair.repair_table,
     ACTION     => dbms_repair.create_action,
     TABLESPACE => 'USERS');
END;
/
```

修復表または孤立キー表それぞれについて、存在しなくなったオブジェクトに関連する行を除外するビューも作成されます。ビュー名は、修復表または孤立キー表の名前に対応しており、接頭辞 DBA_ が付いています（例：DBA_REPAIR_TABLE、DBA_ORPHAN_KEY_TABLE）。

次の問合せでは、users 表領域に作成された修復表が表示されます。

```
DESC REPAIR_TABLE

 Name                           Null?    Type
 ------------------------------ -------- --------------
 OBJECT_ID                      NOT NULL NUMBER
 TABLESPACE_ID                  NOT NULL NUMBER
 RELATIVE_FILE_ID               NOT NULL NUMBER
 BLOCK_ID                       NOT NULL NUMBER
 CORRUPT_TYPE                   NOT NULL NUMBER
 SCHEMA_NAME                    NOT NULL VARCHAR2(30)
 OBJECT_NAME                    NOT NULL VARCHAR2(30)
 BASEOBJECT_NAME                         VARCHAR2(30)
 PARTITION_NAME                          VARCHAR2(30)
 CORRUPT_DESCRIPTION                     VARCHAR2(2000)
 REPAIR_DESCRIPTION                      VARCHAR2(200)
 MARKED_CORRUPT                 NOT NULL VARCHAR2(10)
 CHECK_TIMESTAMP                NOT NULL DATE
 FIX_TIMESTAMP                           DATE
 REFORMAT_TIMESTAMP                      DATE
```

例：孤立キー表の作成

次の例は、users 表領域の孤立キー表の作成方法を示しています。

```
BEGIN
DBMS_REPAIR.ADMIN_TABLES (
     TABLE_NAME => 'ORPHAN_KEY_TABLE',
     TABLE_TYPE => dbms_repair.orphan_table,
     ACTION     => dbms_repair.create_action,
     TABLESPACE => 'USERS');
END;
/
```

次の問合せでは、孤立キー表の定義を表示しています。

```
DESC ORPHAN_KEY_TABLE

 Name                           Null?    Type
 ------------------------------ -------- -----------------
 SCHEMA_NAME                    NOT NULL VARCHAR2(30)
 INDEX_NAME                     NOT NULL VARCHAR2(30)
 IPART_NAME                              VARCHAR2(30)
 INDEX_ID                       NOT NULL NUMBER
 TABLE_NAME                     NOT NULL VARCHAR2(30)
 PART_NAME                               VARCHAR2(30)
 TABLE_ID                       NOT NULL NUMBER
 KEYROWID                       NOT NULL ROWID
 KEY                            NOT NULL ROWID
 DUMP_TIMESTAMP                 NOT NULL DATE
```

例 : 破損の検出

CHECK_OBJECT プロシージャは、指定されたオブジェクトをチェックし、破損および修復ディレクティブに関する情報を修復表に移入します。オブジェクトの一部をチェックする場合は、必要に応じて、範囲、パーティション名またはサブパーティション名を指定できます。

妥当性チェックでは、オブジェクト内部でそれまでに破損マークが付けられていないブロックがすべてチェックされます。ブロックごとに、トランザクションおよびデータ・レイヤー部分の自己整合性がチェックされます。CHECK_OBJECT の実行中に、破損バッファ・キャッシュ・ヘッダーを持つブロックが検出されると、そのブロックはスキップされます。

scott.dept 表に対する CHECK_OBJECT プロシージャの実行例を次に示します。

```
SET SERVEROUTPUT ON
DECLARE num_corrupt INT;
BEGIN
num_corrupt := 0;
DBMS_REPAIR.CHECK_OBJECT (
     SCHEMA_NAME => 'SCOTT',
     OBJECT_NAME => 'DEPT',
     REPAIR_TABLE_NAME => 'REPAIR_TABLE',
     CORRUPT_COUNT =>  num_corrupt);
DBMS_OUTPUT.PUT_LINE('number corrupt: ' || TO_CHAR (num_corrupt));
END;
/
```

SQL*Plus には、1 つの破損を示す次の行が出力されます。

```
number corrupt: 1
```

修復表を問い合せると、破損の説明および修復アクションに関する提案を含む情報が表示されます。

```
SELECT OBJECT_NAME, BLOCK_ID, CORRUPT_TYPE, MARKED_CORRUPT,
       CORRUPT_DESCRIPTION, REPAIR_DESCRIPTION
     FROM REPAIR_TABLE;

OBJECT_NAME                     BLOCK_ID CORRUPT_TYPE MARKED_COR
------------------------------ ---------- ------------ ----------
CORRUPT_DESCRIPTION
--------------------------------------------------------------------------------
REPAIR_DESCRIPTION
--------------------------------------------------------------------------------
DEPT                                    3            1 FALSE
kdbchk: row locked by non-existent transaction
        table=0   slot=0
        lockid=32   ktbbhitc=1
mark block software corrupt
```

まだ破損ブロックに破損マークが付いていないため、ここで重要なデータを抽出します。ブロックに破損マークが付けられた後は、そのブロック全体がスキップされます。

例 : 破損ブロックの修正

CHECK_OBJECT プロシージャによって生成した修復表内の情報に基づいて、FIX_CORRUPT_BLOCKS プロシージャを使用し、指定したオブジェクトの破損ブロックを修正します。ブロックの変更前には、そのブロックがまだ破損状態にあるかどうかが確認されます。破損ブロックは、ソフトウェア破損マークを付けることによって修復されます。修復が実行されると、修復表内の対応する行がタイムスタンプ付きで更新されます。

次の例では、CHECK_OBJECT プロシージャによってレポートされた表 scott.dept の破損ブロックを修正しています。

```
SET SERVEROUTPUT ON
DECLARE num_fix INT;
BEGIN
num_fix := 0;
DBMS_REPAIR.FIX_CORRUPT_BLOCKS (
     SCHEMA_NAME => 'SCOTT',
     OBJECT_NAME=> 'DEPT',
     OBJECT_TYPE => dbms_repair.table_object,
     REPAIR_TABLE_NAME => 'REPAIR_TABLE',
     FIX_COUNT=> num_fix);
DBMS_OUTPUT.PUT_LINE('num fix: ' || TO_CHAR(num_fix));
END;
/
```

SQL*Plus では次の行が出力されます。

```
num fix: 1
```

次の問合せによって、修復が完了していることが確認されます。

```
SELECT OBJECT_NAME, BLOCK_ID, MARKED_CORRUPT
     FROM REPAIR_TABLE;

OBJECT_NAME                     BLOCK_ID MARKED_COR
------------------------------ ---------- ----------
DEPT                                    3 TRUE
```

例 : 破損データ・ブロックを指す索引エントリの検索

DUMP_ORPHAN_KEYS プロシージャは、破損データ・ブロック内の行を指す索引エントリをレポートします。索引エントリごとに、指定した孤立キー表に 1 行ずつ挿入されます。この孤立キー表は、事前に作成しておく必要があります。

この情報は、表内の失われた行を再作成する場合や診断に使用します。

> **注意 :** このプロシージャは、修復表で識別された表に対応付けられている索引ごとに実行する必要があります。

この例で、pk_dept は scott.dept 表の索引です。この索引をスキャンして、破損データ・ブロック内の行を指す索引エントリの有無を判別しています。

```
SET SERVEROUTPUT ON
DECLARE num_orphans INT;
BEGIN
num_orphans := 0;
DBMS_REPAIR.DUMP_ORPHAN_KEYS (
     SCHEMA_NAME => 'SCOTT',
     OBJECT_NAME => 'PK_DEPT',
     OBJECT_TYPE => dbms_repair.index_object,
     REPAIR_TABLE_NAME => 'REPAIR_TABLE',
     ORPHAN_TABLE_NAME=> 'ORPHAN_KEY_TABLE',
     KEY_COUNT => num_orphans);
```

```
DBMS_OUTPUT.PUT_LINE('orphan key count: ' || TO_CHAR(num_orphans));
END;
/
```

次の出力は、孤立キーが 3 つあることを示しています。

```
orphan key count: 3
```

孤立キー表の索引エントリは、索引の再作成が必要であることを示しています。索引の再作成により、表プローブと索引プローブが同じ結果セットを返すことが保証されます。

例 : 破損ブロックのスキップ

SKIP_CORRUPT_BLOCKS プロシージャは、指定されたオブジェクトの索引および表のスキャン時に破損ブロックをスキップするかしないかを指定します。オブジェクトが表であれば、スキップは表とその索引に適用されます。オブジェクトがクラスタであれば、スキップはクラスタ内のすべての表とそれぞれの索引に適用されます。

次の例では、scott.dept 表のソフトウェア破損ブロックのスキップを可能に設定しています。

```
BEGIN
DBMS_REPAIR.SKIP_CORRUPT_BLOCKS (
     SCHEMA_NAME => 'SCOTT',
     OBJECT_NAME => 'DEPT',
     OBJECT_TYPE => dbms_repair.table_object,
     FLAGS => dbms_repair.skip_flag);
END;
/
```

DBA_TABLES ビューを使用して scott の表を問い合せると、表 scott.dept の SKIP_CORRUPT が使用可能になっていることが示されます。

```
SELECT OWNER, TABLE_NAME, SKIP_CORRUPT FROM DBA_TABLES
    WHERE OWNER = 'SCOTT';

OWNER                          TABLE_NAME                     SKIP_COR
------------------------------ ------------------------------ --------
SCOTT                          ACCOUNT                        DISABLED
SCOTT                          BONUS                          DISABLED
SCOTT                          DEPT                           ENABLED
SCOTT                          DOCINDEX                       DISABLED
SCOTT                          EMP                            DISABLED
SCOTT                          RECEIPT                        DISABLED
SCOTT                          SALGRADE                       DISABLED
SCOTT                          SCOTT_EMP                      DISABLED
SCOTT                          SYS_IOT_OVER_12255             DISABLED
SCOTT                          WORK_AREA                      DISABLED

10 rows selected.
```

第IV部

データベース・リソースの管理とタスクのスケジューリング

第 IV 部では、自動データベース・メンテナンス・タスク、データベース・リソースの管理およびタスクのスケジューリングについて説明します。この部の構成は、次のとおりです。

24

自動データベース・メンテナンス・タスクの管理

Oracle Database では、データベース管理者が通常実行する一般的なメンテナンス・タスクの多くが自動化されました。これらの自動化メンテナンス・タスクは、システムの負荷が軽いと予測される時間帯に実行されます。管理者は、個々のメンテナンス・タスクを有効または無効にしたり、これらのタスクの実行時期やタスクに対するリソース割当てを構成できます。

この章の内容は次のとおりです。

注意： この章では、PL/SQL パッケージを使用して、自動化メンテナンス・タスクを管理する方法について説明します。簡単な方法として、Enterprise Manager のグラフィカル・インタフェースを使用する方法があります。

Enterprise Manager を使用して自動メンテナンス・タスクを管理する手順は、次のとおりです。

1. データベース・ホームページにアクセスします。

   手順の詳細は、『Oracle Database 2 日でデータベース管理者』または Oracle Enterprise Manager Grid Control のオンライン・ヘルプを参照してください。

2. データベース・ホームページで、**「サーバー」**をクリックして「サーバー」ページを表示します。

3. 「Oracle Scheduler」セクションで、メンテナンス・タスクを構成する場合は**「自動化メンテナンス・タスク」**をクリックし、メンテナンス・ウィンドウを構成する場合は**「ウィンドウ」**をクリックします。

自動化メンテナンス・タスクの概要

自動化メンテナンス・タスクとは、データベースのメンテナンス操作を実行するために、一定の間隔をおいて自動的に開始されるタスクです。問合せオプティマイザのスキーマ・オブジェクトに関する統計を収集するタスクなどはその一例です。自動化メンテナンス・タスクはメンテナンス・ウィンドウ内で実行されます。このウィンドウは、システムの負荷が低い時間帯にタスクが発生するように事前定義した時間間隔です。管理者は、データベースのリソース使用パターンに基づいてメンテナンス・ウィンドウをカスタマイズしたり、特定のデフォルト・ウィンドウでの実行を禁止できます。独自のメンテナンス・ウィンドウを作成することもできます。

Oracle Database では、3 種類の自動化メンテナンス・タスクが事前定義されています。

- **自動オプティマイザ統計収集 :** データベース内に統計がないか、古い統計のみがあるすべてのスキーマ・オブジェクトに関するオプティマイザ統計を収集します。このタスクで収集された統計は、SQL の実行パフォーマンスを改善するために SQL 問合せオプティマイザによって使用されます。

  関連項目 : 自動統計収集の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

- **自動セグメント・アドバイザ :** 再生可能な領域が存在しているセグメントを識別し、それらのセグメントの断片化を解消する方法について推奨事項を生成します。

  セグメント・アドバイザを手動で実行すると、最新の推奨事項を取得したり、自動セグメント・アドバイザで再生可能な領域を調査しなかったセグメントについて推奨事項を取得することもできます。

  関連項目 : 詳細は、17-13 ページの「セグメント・アドバイザの使用」を参照してください。

- **自動 SQL チューニング・アドバイザ :** 高負荷の SQL 文のパフォーマンスを調査し、それらの文のチューニング方法について推奨事項を生成します。このアドバイザは、SQL プロファイルの推奨事項を自動的に実装するように構成できます。

  関連項目 : SQL チューニング・アドバイザの詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

デフォルトでは、これら 3 種類の自動化メンテナンス・タスクは、すべてのメンテナンス・ウィンドウで実行するように構成されています。

メンテナンス・ウィンドウの概要

メンテナンス・ウィンドウは、自動化メンテナンス・タスクが実行される連続した時間間隔です。メンテナンス・ウィンドウは、MAINTENANCE_WINDOW_GROUP というウィンドウ・グループに属する Oracle Scheduler のウィンドウです。スケジューラのウィンドウは、単純な繰返し間隔（毎週土曜の午前 0 時から午前 6 時の間など）の場合や複雑な間隔（会社の休業日を除いて、毎月最後の稼働日の午前 0 時から午前 6 時の間など）の場合があります。

メンテナンス・ウィンドウがオープンすると、Oracle Database では、そのウィンドウでの実行がスケジュールされている各メンテナンス・タスクに対して Oracle Scheduler のジョブが作成されます。各ジョブには、実行時に生成されるジョブ名が割り当てられます。自動化メンテナンス・タスクのジョブ名はすべて ORA$AT で始まります。たとえば、自動セグメント・アドバイザのジョブには、ORA$AT_SA_SPC_SY_26 という名前が指定されます。自動化メンテナンス・タスクのジョブが終了すると、そのジョブは Oracle Scheduler のジョブ・システムから削除されます。ただし、スケジューラのジョブ履歴には記録が残ります。

注意： ジョブ履歴を表示するには、SYS ユーザーでログインする必要があります。

非常に長いメンテナンス・ウィンドウの場合、自動 SQL チューニング・アドバイザを除くすべての自動化メンテナンス・タスクは 4 時間ごとに再起動されます。この機能によって、ウィンドウ・サイズに関係なく、メンテナンス・タスクが定期的に実行されます。

自動化メンテナンス・タスクのフレームワークは、データベースに定義されているメンテナンス・ウィンドウに依存しています。24-7 ページの表 24-1 に、新規 Oracle Database の各インストールで自動的に定義されるメンテナンス・ウィンドウのリストを示します。

関連項目：

- ウィンドウとウィンドウ・グループの詳細は、第 27 章「Oracle Scheduler を使用したジョブのスケジューリング」を参照してください。

自動化メンテナンス・タスクの構成

メンテナンス・ウィンドウのサブセットで特定のメンテナンス・タスクを有効または無効にするには、DBMS_AUTO_TASK_ADMIN PL/SQL パッケージを使用します。

この項の内容は、次のとおりです。

- すべてのメンテナンス・ウィンドウに対するメンテナンス・タスクの有効化と無効化
- 特定のメンテナンス・ウィンドウに対するメンテナンス・タスクの有効化と無効化

すべてのメンテナンス・ウィンドウに対するメンテナンス・タスクの有効化と無効化

管理者は、1 回の操作ですべてのメンテナンス・ウィンドウに対して特定の自動化メンテナンス・タスクを無効にできます。無効にするには、DBMS_AUTO_TASK_ADMIN PL/SQL パッケージの DISABLE プロシージャを window_name 引数なしでコールします。たとえば、次のプロシージャを使用して自動 SQL チューニング・アドバイザ・タスクを完全に無効にできます。

```
BEGIN
dbms_auto_task_admin.disable(
    client_name => 'sql tuning advisor',
    operation   => NULL,
    window_name => NULL);
END;
```

このメンテナンス・タスクを再度有効にするには、次のように ENABLE プロシージャを使用します。

```
BEGIN
dbms_auto_task_admin.enable(
    client_name => 'sql tuning advisor',
    operation   => NULL,
    window_name => NULL);
END;
```

client_name 引数に使用するタスク名は、DBA_AUTOTASK_CLIENT データベース・ディクショナリ・ビューにリストされています。

すべてのウィンドウに対して自動化メンテナンス・タスクすべてを有効または無効にするには、引数を指定せずに ENABLE または DISABLE プロシージャをコールします。

```
EXECUTE DBMS_AUTO_TASK_ADMIN.DISABLE;
```

関連項目：

- 24-8 ページ「自動化メンテナンス・タスクのデータベース・ディクショナリ・ビュー」
- DBMS_AUTO_TASK_ADMIN PL/SQL パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

特定のメンテナンス・ウィンドウに対するメンテナンス・タスクの有効化と無効化

デフォルトでは、すべてのメンテナンス・タスクが事前定義のすべてのメンテナンス・ウィンドウで実行されます。管理者は、特定のウィンドウに対してメンテナンス・タスクを無効にできます。次の例は、ウィンドウ MONDAY_WINDOW での自動 SQL チューニング・アドバイザの実行を無効にしています。

```
BEGIN
dbms_auto_task_admin.disable(
    client_name => 'sql tuning advisor',
    operation   => NULL,
    window_name => 'MONDAY_WINDOW');
END;
```

メンテナンス・ウィンドウの構成

事前定義のメンテナンス・ウィンドウをデータベース環境に適した時間に調整したり、新しいメンテナンス・ウィンドウを作成できます。メンテナンス・ウィンドウは、DBMS_SCHEDULER PL/SQL パッケージを使用してカスタマイズできます。

この項の内容は、次のとおりです。

- メンテナンス・ウィンドウの変更
- 新規メンテナンス・ウィンドウの作成
- メンテナンス・ウィンドウの削除

メンテナンス・ウィンドウの変更

DBMS_SCHEDULER PL/SQL パッケージには、ウィンドウの属性を変更するための SET_ATTRIBUTE プロシージャが含まれています。たとえば、次のスクリプトは、メンテナンス・ウィンドウ SATURDAY_WINDOW の期間を 4 時間に変更します。

```
BEGIN
dbms_scheduler.disable(
    name  => 'SATURDAY_WINDOW');
dbms_scheduler.set_attribute(
    name      => 'SATURDAY_WINDOW',
    attribute => 'DURATION',
    value     => numtodsinterval(4, 'hour'));
dbms_scheduler.enable(
    name => 'SATURDAY_WINDOW');
END;
```

ウィンドウを変更する際は、事前に DBMS_SCHEDULER.DISABLE サブプログラムを使用してそのウィンドウを無効にし、終了した後に DBMS_SCHEDULER.ENABLE を使用して再度有効にする必要があります。現在オープン中のウィンドウを変更した場合は、そのウィンドウの次回オープンまで変更内容が反映されません。

> **関連項目：** ウィンドウの変更の詳細は、27-30 ページの「ウィンドウの使用」を参照してください。

新規メンテナンス・ウィンドウの作成

DBMS_SCHEDULER PL/SQL パッケージには、ADD_WINDOW_GROUP_MEMBER サブプログラムが用意されています。このサブプログラムはウィンドウをウィンドウ・グループに追加します。メンテナンス・ウィンドウを作成するには、スケジューラのウィンドウを作成して、ウィンドウ・グループ MAINTENANCE_WINDOW_GROUP に追加する必要があります。

次の例は、DBMS_SCHEDULER パッケージを使用して EARLY_MORNING_WINDOW というメンテナンス・ウィンドウを作成しています。このウィンドウは、午前 5 時から午前 6 時まで毎日 1 時間実行されます。

```
BEGIN
dbms_scheduler.create_window(
    window_name     => 'EARLY_MORNING_WINDOW',
    duration        =>  numtodsinterval(1, 'hour'),
    resource_plan   => 'DEFAULT_MAINTENANCE_PLAN',
    repeat_interval => 'FREQ=DAILY;BYHOUR=5;BYMINUTE=0;BYSECOND=0');
dbms_scheduler.add_window_group_member(
    group_name  => 'MAINTENANCE_WINDOW_GROUP',
    window_list => 'EARLY_MORNING_WINDOW');
END;
```

関連項目：

- 27-31 ページ「ウィンドウの作成」
- DBMS_SCHEDULER パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

メンテナンス・ウィンドウの削除

既存のメンテナンス・ウィンドウを削除するには、MAINTENANCE_WINDOW_GROUP ウィンドウ・グループから該当するウィンドウを削除します。ウィンドウは存在し続けますが、自動化メンテナンス・タスクは実行されません。このウィンドウに割り当てられているスケジューラの他のジョブは、引き続き通常どおりに実行されます。

次の例は、ウィンドウ・グループから EARLY_MORNING_WINDOW を削除しています。

```
BEGIN
DBMS_SCHEDULER.REMOVE_WINDOW_GROUP_MEMBER(
    group_name  => 'MAINTENANCE_WINDOW_GROUP',
    window_list => 'EARLY_MORNING_WINDOW');
END;
```

関連項目：

- 27-40 ページ「ウィンドウ・グループからのメンバーの削除」
- 27-34 ページ「ウィンドウの削除」
- DBMS_SCHEDULER パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

自動化メンテナンス・タスクに対するリソース割当ての構成

ここでは、メンテナンス・ウィンドウに対するリソース割当てについて説明します。この項の内容は、次のとおりです。

- 自動化メンテナンス・タスクに対するリソース割当ての概要
- 自動化メンテナンス・タスクに対するリソース割当ての変更

関連項目： 第 25 章「Oracle Database Resource Manager を使用したリソース割当ての管理」

自動化メンテナンス・タスクに対するリソース割当ての概要

デフォルトでは、事前定義のすべてのメンテナンス・ウィンドウで、リソース・プラン DEFAULT_MAINTENANCE_PLAN が使用されます。自動化メンテナンス・タスクは、そのサブプラン ORA$AUTOTASK_SUB_PLAN のもとで実行されます。このサブプランによって、リソース割当て合計に対する配分が複数のメンテナンス・タスク間に均一に分割されます。

DEFAULT_MAINTENANCE_PLAN には、次のリソース割当てが定義されています。

コンシューマ・グループ / サブプラン	レベル 1	レベル 2
ORA$AUTOTASK_SUB_PLAN	-	25%
ORA$DIAGNOSTICS	-	5%
OTHER_GROUPS	-	70%
SYS_GROUP	100%	-

このプランでは、SYS_GROUP コンシューマ・グループのセッションが優先されます（このグループのセッションは、SYS および SYSTEM のユーザー・アカウントで作成されたセッションです）。SYS_GROUP のセッションで使用されないリソース割当ては、プランの他のコンシューマ・グループやサブプランに属するセッションによって共有されます。この割当てでは、25% がメンテナンス・タスクに、5% が診断操作を実行するバックグラウンド・プロセスに、70% がユーザー・セッションに割り当てられます。自動化メンテナンス・タスクに対するリソース割当てを増減するには、DEFAULT_MAINTENANCE_PLAN を調整します。詳細は、24-7 ページの「自動化メンテナンス・タスクに対するリソース割当ての変更」を参照してください。

リソース・プランと同様に、コンシューマ・グループまたはサブプランによって使用されていない割当て部分は、他のコンシューマ・グループまたはサブプランで使用できます。データベース・リソース・マネージャは、CPU の 100% が使用されるまで、リソース・プランによるリソース割当ての制限を開始しません。

注意： DEFAULT_MAINTENANCE_PLAN はデフォルトですが、任意のリソース・プランを任意のメンテナンス・ウィンドウに割り当てることができます。

関連項目： リソース・プランの詳細は、第 25 章「Oracle Database Resource Manager を使用したリソース割当ての管理」を参照してください。

自動化メンテナンス・タスクに対するリソース割当ての変更

メンテナンス・ウィンドウ内の自動化メンテナンス・タスクに対するリソース割当てを変更するには、そのウィンドウのリソース・プランのサブプラン ORA$AUTOTASK_SUB_PLAN に割り当てられているリソースの割合を変更する必要があります（デフォルトでは、事前定義の各メンテナンス・ウィンドウのリソース・プランは DEFAULT_MAINTENANCE_PLAN です）。プランのトップレベルでのリソース割当てが 100% になるように、ウィンドウのリソース・プランの 1 つ以上のサブプランまたはコンシューマ・グループに対するリソース割当てを調整する必要があります。リソース割当ての変更の詳細は、第 25 章「Oracle Database Resource Manager を使用したリソース割当ての管理」を参照してください。

自動化メンテナンス・タスクの参照情報

ここでは、自動化メンテナンス・タスクについて説明します。この項の内容は、次のとおりです。

- 事前定義のメンテナンス・ウィンドウ
- 自動化メンテナンス・タスクのデータベース・ディクショナリ・ビュー

事前定義のメンテナンス・ウィンドウ

デフォルトでは、7 種類のメンテナンス・ウィンドウが事前定義されており、それぞれが曜日を表します。週末のメンテナンス・ウィンドウである SATURDAY_WINDOW と SUNDAY_WINDOW は、平日のメンテナンス・ウィンドウよりも期間が長くなっています。ウィンドウ・グループ MAINTENANCE_WINDOW_GROUP は、この 7 種類のウィンドウで構成されています。表 24-1 に、事前定義のメンテナンス・ウィンドウのリストを示します。

表 24-1 事前定義のメンテナンス・ウィンドウ

ウィンドウ名	説明
MONDAY_WINDOW	月曜の午後 10 時に開始して午前 2 時に終了します。
TUESDAY_WINDOW	火曜の午後 10 時に開始して午前 2 時に終了します。
WEDNESDAY_WINDOW	水曜の午後 10 時に開始して午前 2 時に終了します。
THURSDAY_WINDOW	木曜の午後 10 時に開始して午前 2 時に終了します。

表 24-1 事前定義のメンテナンス・ウィンドウ（続き）

ウィンドウ名	説明
FRIDAY_WINDOW	金曜の午後 10 時に開始して午前 2 時に終了します。
SATURDAY_WINDOW	土曜の午前 6 時に開始して 20 時間後に終了します。
SUNDAY_WINDOW	日曜の午前 6 時に開始して 20 時間後に終了します。

自動化メンテナンス・タスクのデータベース・ディクショナリ・ビュー

表 24-2 に、自動化メンテナンス・タスクのデータベース・ディクショナリ・ビューに関する情報を示します。

表 24-2 自動化メンテナンス・タスクのデータベース・ディクショナリ・ビュー

ビュー名	説明
DBA_AUTOTASK_CLIENT_JOB	自動化メンテナンス・タスク用に作成され、現在実行中のスケジューラのジョブに関する情報が表示されます。このビューでは、ジョブの対象となるオブジェクトの情報と、同じタスクについて前回のインスタンス化から追加された統計情報が提供されます。この追加データの一部は、スケジューラの一般的なビューから取得されます。
DBA_AUTOTASK_CLIENT	7 日間および 30 日間の各自動化メンテナンス・タスクに関する統計データが表示されます。
DBA_AUTOTASK_JOB_HISTORY	自動化メンテナンス・タスクのジョブ実行履歴がリストされます。ジョブは、実行終了後に、このビューに追加されます。
DBA_AUTOTASK_WINDOW_CLIENTS	MAINTENANCE_WINDOW_GROUP に属するウィンドウが、各メンテナンス・タスクのウィンドウの「有効」または「無効」ステータスとともにリストされます。このビューは主に Enterprise Manager によって使用されます。
DBA_AUTOTASK_CLIENT_HISTORY	各自動化メンテナンス・タスクのジョブ実行回数について、ウィンドウ別に履歴が表示されます。この情報は、Enterprise Manager の「ジョブ履歴」ページに表示されます。

関連項目： ビューの列の詳細は、25-46 ページの「リソース・マネージャのデータ・ディクショナリ・ビュー」を参照してください。

25

Oracle Database Resource Manager を使用したリソース割当ての管理

この章の内容は次のとおりです。

注意： この章では、PL/SQL パッケージ・プロシージャを使用してリソース・マネージャを管理する方法について説明します。リソース・マネージャを管理する簡単な方法は、Enterprise Manager の GUI を使用することです。

Enterprise Manager でリソース・マネージャを管理する手順は、次のとおりです。

1. データベース・ホームページにアクセスします。

   手順については、『Oracle Database 2 日でデータベース管理者』を参照してください。
2. ページの上部にある**「サーバー」**をクリックして「サーバー」ページを表示します。
3. 「リソース・マネージャ」セクションで、**「はじめに」**をクリックします。

Oracle Database Resource Manager の概要

Oracle Database Resource Manager（リソース・マネージャ）を使用すると、多数のコンカレント・データベース・セッションにリソースを最適に割り当てられます。次の各項では、リソース・マネージャの概要を説明します。

リソース・マネージャが対処する問題

データベース・リソースの割当てがオペレーティング・システムによって決定される場合は、次のような問題が生じることがあります。

- 過剰なオーバーヘッドの発生

  過剰なオーバーヘッドは、サーバー・プロセスの数が多いときに、Oracle Database サーバー・プロセス間でオペレーティング・システムのコンテキストが切り替わることによって発生します。

- 非効率的なスケジューリング

  オペレーティング・システムは、データベース・サーバーがラッチを保持している間にデータベース・サーバーのスケジュールを解除しますが、これは非効率的です。

- 不適当なリソースの割当て

  オペレーティング・システムはタスク間の優先度付けができないため、すべてのアクティブなプロセスの間で均等にリソースを分配します。

- パラレル実行サーバーやアクティブ・セッションなど、データベース固有のリソースを管理できない

リソース・マネージャによるこれらの問題の対処方法

リソース・マネージャは、ハードウェア・リソースの割当て方法をデータベースで厳密に制御することによって、これらの問題を克服します。複数のコンカレント・ユーザー・セッションがあり、各セッションでは異なる優先度のジョブが実行される環境では、すべてのセッションが同等に処理されるわけではありません。リソース・マネージャを使用すると、セッション属性に基づいてセッションを複数のグループに分類し、それらのグループに対して、アプリケーション環境のハードウェア利用が最適化されるようにリソースを配分できます。

リソース・マネージャを使用すると、次のことが可能になります。

- システムにかかる負荷やユーザー数に関係なく、特定のセッションの処理リソースが最小量になることを保証します。
- 様々なユーザーおよびアプリケーションに、比率を指定して CPU タイムを割り当てて、使用可能な処理リソースを分配します。データ・ウェアハウスの場合は、バッチ・ジョブよりもリレーショナル・オンライン分析処理（ROLAP）アプリケーションへの割当て率を高くすることができます。
- ユーザー・グループのメンバーが実行する処理の並列度を制限します。
- アクティブ・セッション・プールを作成します。**アクティブ・セッション・プール**は、ユーザー・グループ内で同時にアクティブにできる指定された最大数のユーザー・セッションで構成されています。最大数を超える追加セッションは実行待ちのキューに送られますが、キューで待機しているジョブが終了するまでのタイムアウト周期を指定できます。アクティブ・セッション・プールによって、リソースについて実際に競合するセッションの総数が制限されるため、アクティブなセッションの進行を早めることができます。

- CPU または I/O が指定量を超えて使用されている状況を検出することで、リソース集中型のセッションまたはコールを管理します。このようなセッションは、自動的に終了させるか、別の（優先度の低い）グループに自動的に切り替えることができます。
- ある操作に対するオプティマイザの見積り実行時間が指定された制限を超える場合は、その操作が実行されないようにします。
- セッションのアイドル時間を制限します。この制限は、他のセッションをブロックしているセッションのみに限定することもできます。
- 特定のリソース割当て方式を使用するように、インスタンスを構成します。インスタンスを停止して再起動しなくても、日中の方式から夜間の方式へというように、リソース割当て方式を動的に変更できます。方式の変更は、Oracle Scheduler を使用してスケジュールすることもできます。詳細は、第 26 章「Oracle Scheduler の概要」を参照してください。

リソース・マネージャの要素

次の表に、リソース・マネージャの要素を示します。

要素	説明
リソース・コンシューマ・グループ	リソースの要件に基づいてグループ化されたセッションのグループ。リソース・マネージャは、個々のセッションではなくリソース・コンシューマ・グループにリソースを割り当てます。
リソース・プラン	リソース・コンシューマ・グループへのリソースの割当て方法を指定するディレクティブのコンテナ。特定のリソース・プランをアクティブ化することで、データベースによるリソースの割当て方法を指定します。
リソース・プラン・ディレクティブ	リソース・コンシューマ・グループを特定のプランに関連付け、そのリソース・コンシューマ・グループに対するリソースの割当て方法を指定します。

これらの要素の作成とメンテナンスには、DBMS_RESOURCE_MANAGER PL/SQL パッケージを使用します。要素は、データ・ディクショナリ内の表に格納されます。要素に関する情報は、データ・ディクショナリ・ビューを使用して表示できます。

関連項目： 25-46 ページ「リソース・マネージャのデータ・ディクショナリ・ビュー」

リソース・コンシューマ・グループの概要

リソース・コンシューマ・グループ（コンシューマ・グループ）とは、処理要件に基づいてグループ化されたユーザー・セッションの集合です。セッションが作成されると、そのセッションは、管理者が設定したマッピング・ルールに基づいてコンシューマ・グループに自動的にマップされます。データベース管理者（DBA）は、セッションを、異なるコンシューマ・グループに手動で切り替えることができます。同様に、アプリケーションでは、そのセッションを特定のコンシューマ・グループに切り替える PL/SQL パッケージ・プロシージャを実行できます。

リソース・マネージャによるリソース（CPU など）の割当て先は、コンシューマ・グループのみであるため、セッションがコンシューマ・グループのメンバーになると、そのリソース割当ては、コンシューマ・グループの割当てによって決定されます。デフォルトでは、コンシューマ・グループの各セッションは、そのグループに割り当てられているリソースをグループ内の他のセッションとラウンド・ロビン方式で共有します。

常にデータ・ディクショナリに存在する 3 種類の特別なコンシューマ・グループがあります。これらのグループは、変更したり削除することはできません。次に、この 3 種類のコンシューマ・グループを示します。

- SYS_GROUP

  これは、ユーザー・アカウント SYS または SYSTEM によって作成されたすべてのセッションの初期コンシューマ・グループです。この初期コンシューマ・グループは、コンシューマ・グループへのセッションのマッピング・ルールで上書きできます。

- DEFAULT_CONSUMER_GROUP

  これは、SYS および SYSTEM 以外のユーザー・アカウントによって開始されたすべてのセッションの初期コンシューマ・グループです。この初期コンシューマ・グループは、コンシューマ・グループへのセッションのマッピング・ルールで上書きできます。DEFAULT_CONSUMER_GROUP には、リソース・プラン・ディレクティブで名前を指定できません。

- OTHER_GROUPS

  このグループは、DEFAULT_CONSUMER_GROUP に属しているセッションも含めて、現在はアクティブなプランの一部でないコンシューマ・グループに属しているすべてのセッションに適用されます。OTHER_GROUPS には、すべてのプランに指定されているリソース・プラン・ディレクティブを設定する必要があります。このグループは、マッピング・ルールを介してセッションに明示的に割り当てることはできません。

関連項目：

- 25-44 ページ表 25-2「事前定義のリソース・コンシューマ・グループ」
- 25-25 ページ「コンシューマ・グループへのセッションのマッピング・ルールの指定」

リソース・プラン・ディレクティブの概要

リソース・マネージャは、現在アクティブなリソース・プランに属している一連のリソース・プラン・ディレクティブ（ディレクティブ）に従って、リソースをコンシューマ・グループに割り当てます。リソース・プランとそのリソース・プラン・ディレクティブには親子の関係があります。各ディレクティブは 1 つのコンシューマ・グループを参照します。現在アクティブなプランの 2 つのディレクティブが同じコンシューマ・グループを参照することはできません。

ディレクティブには、コンシューマ・グループへのリソース割当てを制限できる多くの方法があります。たとえば、コンシューマ・グループが確保できる CPU 全体の割合を制御したり、コンシューマ・グループ内でアクティブにできるセッションの総数を制限できます。詳細は、25-7 ページの「リソース割当て方法の概要」を参照してください。

リソース・プランの概要

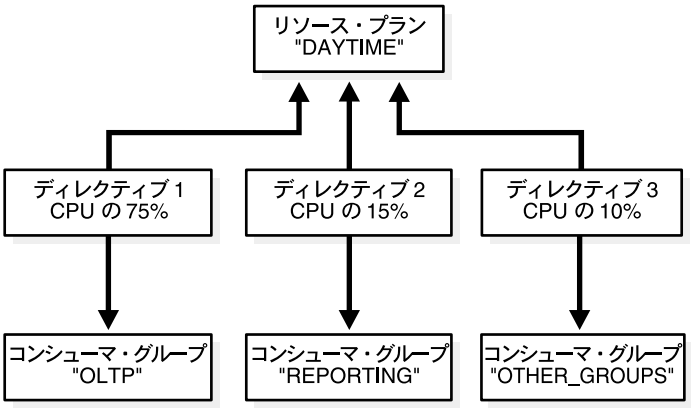
各 Oracle データベースに事前定義されているリソース・プランの他に、リソース・プランは必要な数だけ作成できます。ただし、1 度にアクティブにできるリソース・プランは 1 つのみです。リソース・プランがアクティブな場合は、その子の各リソース・プラン・ディレクティブが異なるコンシューマ・グループへのリソース割当てを制御します。各プランには、OTHER_GROUPS というコンシューマ・グループにリソースを割り当てるディレクティブが必要です。OTHER_GROUPS は、現在アクティブなプランの一部でないコンシューマ・グループに属しているすべてのセッションに適用されます。

注意： リソース・プラン（または単にプラン）という用語は、リソース・マネージャの 1 つの要素を意味しますが、この章では、リソース・プランの要素自体や、そのリソース・プラン・ディレクティブ、ディレクティブが参照するコンシューマ・グループを含めた完全なリソース・プラン・スキーマを示す場合にも使用しています。たとえば、この章で DAYTIME リソース・プランに触れたときは、DAYTIME というリソース・プラン要素を意味する場合も、DAYTIME リソース・プランとそのディレクティブが定義する特定のリソース割当てスキーマを意味する場合もあります。したがって、簡潔に言えば、「DAYTIME プランは、バッチ・アプリケーションよりも対話型アプリケーションに近い」といえます。

例 : 単純なリソース・プラン

図 25-1 は、日中にオンライン・トランザクション処理（OLTP）アプリケーションとレポート・アプリケーションを同時に実行する組織の単純なリソース・プランを示しています。現在アクティブなプラン DAYTIME によって、3 つのリソース・コンシューマ・グループに CPU リソースが割り当てられます。具体的には、CPU タイムの 75% が OLTP に、15% が REPORTS に、残りの 10% が OTHER_GROUPS に割り当てられています。

図 25-1 単純なリソース・プラン



Oracle Database には、単純なリソース・プランを迅速に作成できるプロシージャ（CREATE_SIMPLE_PLAN）が用意されています。このプロシージャについては、25-10 ページの「単純なリソース・プランの作成」を参照してください。

注意 : 現在アクティブなリソース・プランは、CPU 使用率が 100% になるまで割当て制限を規定しません。CPU 使用率が 100% 未満の場合、データベースは CPU にバインドされないため、すべてのセッションが必要なリソース割当てを獲得できるように制限を規定する必要はありません。

さらに、制限が規定されている場合は、あるコンシューマ・グループが使用していない割当てを、他のコンシューマ・グループが使用できます。前述の例では、OLTP グループが割当てをすべて使用していない場合、リソース・マネージャは、REPORTS グループまたは OTHER_GROUPS グループが、未使用の割当てを使用することを許可します。

サブプランの概要

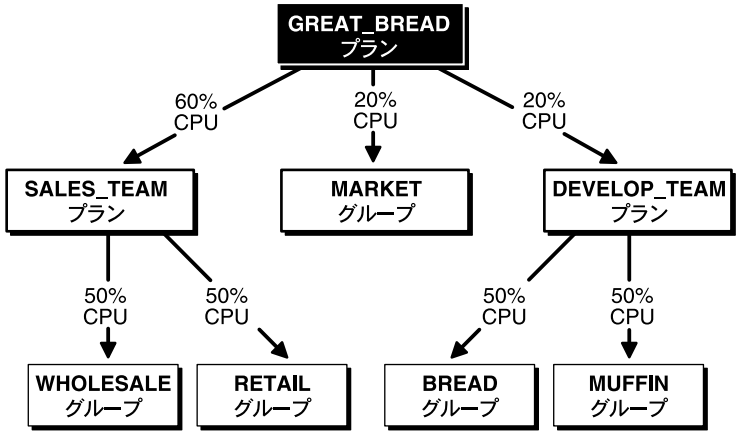
リソース・プラン・ディレクティブ（ディレクティブ）は、コンシューマ・グループを参照するかわりに、別のリソース・プランを参照できます。この場合、そのプランはサブプランと呼ばれます。サブプラン自体に、リソースをコンシューマ・グループと他のサブプランに割り当てるディレクティブがあります。この場合、リソースの割当ては次のように段階的な方式で機能します。最初に、トップレベルのリソース・プラン（現在アクティブなプラン）によって、リソースがコンシューマ・グループとサブプランに分割されます。次に、各サブプランによって、リソース割当て合計の一部がそのコンシューマ・グループとサブプランに割り当てられます。階層的なプランは、必要な数のサブプランを使用して作成できます。

リソース・サブプランは、リソース・プランを作成する方法と同じ方法で作成します。プランとサブプランの作成方法は同じです。プランは、サブプランとして使用することのみでサブプランになります。

例 : サブプランを含むリソース・プラン

この例では、Great Bread Company によって、CPU リソースが図 25-2 のように割り当てられています。この図は、トップレベルのプラン（GREAT_BREAD）とその子すべてを示しています。わかりやすくするために、OTHER_GROUPS コンシューマ・グループを指定する要件は無視し、リソース・プラン・ディレクティブは（プランの一部であっても）省略されています。かわりに、ディレクティブが割り当てる CPU の比率を、プラン、サブプランおよびコンシューマ・グループ間を結ぶ線に沿って表示しています。

図 25-2　サブプランを含むリソース・プラン



この GREAT_BREAD プランでは、次のようにリソースを割り当てます。

- CPU リソースの 20% をコンシューマ・グループ MARKET に割り当てます。
- CPU リソースの 60% をサブプラン SALES_TEAM に割り当てます。このサブプランでは、リソースが WHOLESALE コンシューマ・グループと RETAIL コンシューマ・グループ間で均等に分割されます。
- CPU リソースの 20% をサブプラン DEVELOP_TEAM に割り当てます。このサブプランでは、リソースが BREAD コンシューマ・グループと MUFFIN コンシューマ・グループ間で均等に分割されます。

サブプランまたはコンシューマ・グループは、複数の親を持つ場合があります。たとえば、MARKET グループが SALES_TEAM サブプランに含まれていた場合です。ただし、プラン内でループすることはできません。たとえば、SALES_TEAM サブプランには、GREAT_BREAD プランを参照するディレクティブを設定できません。

> **関連項目：** 複雑なリソース・プランの例は、25-32 ページの「各種の方法を組み合せた Oracle Database Resource Manager の例」を参照してください。

リソース割当て方法の概要

リソース・プラン・ディレクティブは、リソース・コンシューマ・グループまたはサブプランへのリソースの割当て方法を指定します。各ディレクティブでは、リソースをそのコンシューマ・グループまたはサブプランに割り当てるための複数の異なる方法を指定できます。次の各項では、これらのリソース割当て方法について簡単に説明します。

- CPU
- キューイングを備えたアクティブ・セッション・プール
- 並列度制限
- コンシューマ・グループの自動切替え
- SQL の取消しとセッションの終了
- 実行時間制限
- UNDO プール
- アイドル時間制限

CPU

この方法では、コンシューマ・グループおよびサブプランの間での CPU リソースの割当て方法を指定します。複数レベル（最大 8 レベル）の CPU リソース割当てによって、プラン内で CPU 使用の優先度を設定できます。レベル 2 のコンシューマ・グループとサブプランは、レベル 1 に割り当てられなかったリソース、つまりレベル 1 のコンシューマ・グループまたはサブプランで使用されなかったリソースを取得します。同様に、レベル 3 のリソース・コンシューマには、レベル 1 と 2 の割当ての一部が残っている場合のみ、リソースが割り当てられます。同じ規則がレベル 4 から 8 にも適用されます。複数のレベルを使用すると、優先度を設定できるのみでなく、すべての主リソースと残りのリソースの使用方法を明示的に指定できます。

複数レベルのプラン・スキーマの例は、25-33 ページの図 25-3 を参照してください。

注意： レベルが 1 つのみの場合、あるコンシューマ・グループまたはサブプランが使用していない割当ては、同じレベルの別のコンシューマ・グループまたはサブプランで使用できます。

キューイングを備えたアクティブ・セッション・プール

コンシューマ・グループ内で同時にアクティブにできるセッションの最大数を制御できます。この最大数によって、アクティブ・セッション・プールが定義されます。**アクティブ・セッション**とは、コール中のセッションです。セッションがブロックされている場合（I/O リクエスト完了の待機中など）もアクティブとみなされます。アクティブ・セッション・プールがいっぱいの場合、コールを処理しようとしているセッションはキューに送られます。アクティブなセッションが終了すると、キュー内の最初のセッションがキューから削除され、実行に向けてスケジュールされます。実行キュー内のセッションがタイムアウトするまでの期間も指定できます。タイムアウトすると、エラーで終了します。

パラレル実行セッションは全体で 1 つのアクティブ・セッションとして数えられます。

この機能は、リソースについて競合する、コンシューマ・グループ内のセッション数を制限する場合に便利です。たとえば、コンシューマ・グループがレポートのための長時間実行のパラレル問合せ処理に使用されている場合、1 つのレポートをできるだけ早く完了させるためにアクティブ・セッション数を 1 つに制限して、CPU またはパラレル問合せスレーブを他のレポートと競合しないようにできます。

並列度制限

管理者は、1 つのコンシューマ・グループ内での操作に対して最大並列度を制限できます。この制限は、1 つのコンシューマ・グループ内の 1 つの操作に適用されます。コンシューマ・グループ内のすべての操作にわたる総合的な並列度が制限されるわけではありません。ただし、必要な制御を実現するために、並列度制限とアクティブ・セッション・プールの機能を組み合せることはできます。

コンシューマ・グループの自動切替え

この方法を使用すると、指定のコンシューマ・グループにセッションを自動的に切り替えるための基準を指定して、リソース割当てを制御できます。通常、この方法は、グループ内の典型的なセッションに対するリソース使用予測を上回ったセッションを、優先度の高いコンシューマ・グループ（システム・リソースを高い比率で使用するグループ）から優先度の低いコンシューマ・グループに切り替えるために使用します。

詳細は、25-23 ページの「リソース制限の設定による自動切替えの指定」を参照してください。

SQL の取消しとセッションの終了

ディレクティブを指定すると、システム・リソースの使用量に基づいて、長時間実行 SQL 問合せを取り消したり、長時間実行セッションを終了できます。詳細は、25-23 ページの「リソース制限の設定による自動切替えの指定」を参照してください。

実行時間制限

ある操作に許可される最大の実行時間を指定できます。ある操作についてデータベースが見積った実行時間が、指定された最大実行時間より長い場合、その操作はエラーを出力して終了します。このエラーはトラップ可能であり、操作は再スケジュールできます。

UNDO プール

UNDO プールは、コンシューマ・グループごとに指定できます。UNDO プールは、コンシューマ・グループが生成できる、コミットされていないトランザクションに対する UNDO 合計量を制御します。コンシューマ・グループが生成する UNDO の合計量がその UNDO 制限を超えると、その UNDO を生成している現行のデータ操作言語（DML）文が終了します。コンシューマ・グループの他のメンバーは、UNDO 領域がプールから解放されるまで、新たにデータ操作を実行できません。

アイドル時間制限

セッションのアイドル時間を指定できます。指定した時間を経過するとセッションは終了します。厳しいアイドル時間制限を指定して、他のセッションをブロックしているアイドル状態のセッションに適用することもできます。

リソース・マネージャの管理権限の概要

リソース・マネージャを管理するには、システム権限 ADMINISTER_RESOURCE_MANAGER が必要です。この権限（ADMIN オプション付き）は、DBA ロールを介してデータベース管理者に付与されます。

リソース・マネージャの管理者は、DBMS_RESOURCE_MANAGER PL/SQL パッケージ内のすべてのプロシージャを実行できます。

ADMIN オプションを持つ管理者は、必要に応じて管理権限を他のユーザーまたはロールに付与できます。そのためには、DBMS_RESOURCE_MANAGER_PRIVS PL/SQL パッケージを使用します。次の表に、関連するパッケージ・プロシージャの一覧を示します。

プロシージャ	説明
GRANT_SYSTEM_PRIVILEGE	ユーザーまたはロールに、ADMINISTER_RESOURCE_MANAGER システム権限を付与します。
REVOKE_SYSTEM_PRIVILEGE	ユーザーまたはロールから、ADMINISTER_RESOURCE_MANAGER システム権限を取り消します。

次のスクリプトは、ユーザー SCOTT に ADMIN オプションなしで管理権限を付与する PL/SQL ブロックです。したがって、SCOTT は DBMS_RESOURCE_MANAGER パッケージ内のすべてのプロシージャを実行できますが、管理権限を他のユーザーに付与するための GRANT_SYSTEM_PRIVILEGE プロシージャは使用できません。

```
BEGIN
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE(
   GRANTEE_NAME   => 'SCOTT',
   PRIVILEGE_NAME => 'ADMINISTER_RESOURCE_MANAGER',
   ADMIN_OPTION   => FALSE);
END;
```

この権限は、REVOKE_SYSTEM_PRIVILEGE プロシージャを使用して取り消すことができます。

注意： ADMINISTER_RESOURCE_MANAGER システム権限を付与または取り消す方法は、DBMS_RESOURCE_MANAGER_PRIVS パッケージを使用する以外にありません。SQL の GRANT または REVOKE 文を使用して付与または取り消すことはできません。

関連項目： リソース・マネージャの次のパッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

- DBMS_RESOURCE_MANAGER
- DBMS_RESOURCE_MANAGER_PRIVS

単純なリソース・プランの作成

CREATE_SIMPLE_PLAN プロシージャを使用すると、多くの状況に対応できる単純なリソース・プランを迅速に作成できます。このプロシージャでは、1 回のプロシージャ・コールの実行で、コンシューマ・グループを作成し、そのグループにリソースを割り当てることができます。このプロシージャを使用する際は、ペンディング・エリアの作成、各コンシューマ・グループの個別作成、リソース・プラン・ディレクティブの指定のために、後述のプロシージャをコールする必要はありません。

CREATE_SIMPLE_PLAN プロシージャには、次の引数を指定します。

パラメータ	説明
SIMPLE_PLAN	プランの名前
CONSUMER_GROUP1	1 番目のグループのコンシューマ・グループ名
GROUP1_PERCENT	このグループに割り当てる CPU リソース
CONSUMER_GROUP2	2 番目のグループのコンシューマ・グループ名
GROUP2_PERCENT	このグループに割り当てる CPU リソース
CONSUMER_GROUP3	3 番目のグループのコンシューマ・グループ名
GROUP3_PERCENT	このグループに割り当てる CPU リソース
CONSUMER_GROUP4	4 番目のグループのコンシューマ・グループ名
GROUP4_PERCENT	このグループに割り当てる CPU リソース
CONSUMER_GROUP5	5 番目のグループのコンシューマ・グループ名
GROUP5_PERCENT	このグループに割り当てる CPU リソース
CONSUMER_GROUP6	6 番目のグループのコンシューマ・グループ名
GROUP6_PERCENT	このグループに割り当てる CPU リソース
CONSUMER_GROUP7	7 番目のグループのコンシューマ・グループ名
GROUP7_PERCENT	このグループに割り当てる CPU リソース
CONSUMER_GROUP8	8 番目のグループのコンシューマ・グループ名
GROUP8_PERCENT	このグループに割り当てる CPU リソース

このプロシージャでは、最大 8 個のコンシューマ・グループを指定できます。サポート対象のリソース割当て方法は、CPU による方法のみです。プランでは、EMPHASIS CPU 割当てポリシー（デフォルト）が使用され、各コンシューマ・グループでは、ROUND_ROBIN スケジューリング・ポリシー（これもデフォルト）が使用されます。プラン内の指定された各コンシューマ・グループには、レベル 2 の CPU パーセンテージが割り当てられます。このプランには、SYS_GROUP（ユーザー SYS および SYSTEM 用にシステムによって定義された初期コンシューマ・グループ）と OTHER_GROUPS も暗黙的に含まれています。SYS_GROUP コンシューマ・グループには、レベル 1 で CPU の 100% が割り当てられ、OTHER_GROUPS には、レベル 3 で CPU の 100% が割り当てられます。

例 : CREATE_SIMPLE_PLAN プロシージャを使用した単純なプランの作成

次のスクリプトは、ユーザー指定の 2 つのコンシューマ・グループを備えた単純なリソース・プランを作成する PL/SQL ブロックです。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN(SIMPLE_PLAN => 'SIMPLE_PLAN1',
   CONSUMER_GROUP1 => 'MYGROUP1', GROUP1_PERCENT => 80,
   CONSUMER_GROUP2 => 'MYGROUP2', GROUP2_PERCENT => 20);
END;
```

この文を実行すると、次のプランが作成されます。

コンシューマ・グループ	レベル 1	レベル 2	レベル 3
SYS_GROUP	100%	-	-
MYGROUP1	-	80%	-
MYGROUP2	-	20%	-
OTHER_GROUPS	-	-	100%

関連項目 :

- EMPHASIS CPU 割当てポリシーの詳細は、25-14 ページの「リソース・プランの作成」を参照してください。
- ROUND_ROBIN スケジューリング・ポリシーの詳細は、25-13 ページの「リソース・コンシューマ・グループの作成」を参照してください。
- 25-3 ページ「リソース・マネージャの要素」

複雑なリソース・プランの作成

複雑なリソース・プランが必要な場合は、ペンディング・エリアと呼ばれるステージング領域で、プランとそのディレクティブやコンシューマ・グループを作成し、そのプランの妥当性をチェックしてから、データ・ディクショナリに保存する必要があります。

複雑なリソース・プランの作成に必要な手順の概要は、次のとおりです。

注意 : 複雑なリソース・プランとは、DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN プロシージャでは作成できないリソース・プランを指します。

手順 1: ペンディング・エリアを作成します。

手順 2: コンシューマ・グループを作成、変更または削除します。

手順 3: リソース・プランを作成します。

手順 4: リソース・プラン・ディレクティブを作成します。

手順 5: ペンディング・エリアの妥当性をチェックします。

手順 6: ペンディング・エリアを発行します。

これらの手順は、DBMS_RESOURCE_MANAGER PL/SQL パッケージのプロシージャを使用して完了します。次の各項では、詳細について説明します。

- ペンディング・エリアの概要
- ペンディング・エリアの作成
- リソース・コンシューマ・グループの作成
- リソース・プランの作成

- リソース・プラン・ディレクティブの作成
- ペンディング・エリアの妥当性チェック
- ペンディング・エリアの発行
- ペンディング・エリアのクリア

関連項目：

- 25-45 ページ「DBMS_RESOURCE_MANAGER パッケージ・プロシージャの要約」
- DBMS_RESOURCE_MANAGER PL/SQL パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 25-3 ページ「リソース・マネージャの要素」

ペンディング・エリアの概要

ペンディング・エリアとは、現在実行中のアプリケーションに影響を与えずに、新規リソース・プランの作成、既存のプランの更新、またはプランの削除を実行できるステージング領域です。ペンディング・エリアを作成すると、そのペンディング・エリアはデータベースによって初期化されて既存のプランがコピーされます。これによって、既存のプランが更新可能となります。

ヒント： ペンディング・エリアを作成した後、DBA_RSRC_PLANS データ・ディクショナリ・ビューを問い合せてすべてのプランをリストすると、各プランについて 2 つのコピー（PENDING ステータスが付いているコピーと付いていないコピー）が表示されます。PENDING ステータスのプランには、ペンディング・エリアの作成後に実行した変更内容が反映されています。保留中の変更は、コンシューマ・グループの場合は DBA_RSRC_CONSUMER_GROUPS を使用して、リソース・プラン・ディレクティブの場合は DBA_RSRC_PLAN_DIRECTIVES を使用して表示できます。詳細は、25-46 ページの「リソース・マネージャのデータ・ディクショナリ・ビュー」を参照してください。

変更を追加した後のペンディング・エリアは、発行する前に妥当性をチェックします。発行すると、保留中の変更内容がすべてデータ・ディクショナリに適用され、ペンディング・エリアはクリアされ、解除されます。

最初にペンディング・エリアを作成せずにプランを作成、更新または削除しようとすると（つまり、コンシューマ・グループまたはリソース・プラン・ディレクティブを作成、更新または削除しようとすると）、エラー・メッセージが表示されます。

ペンディング・エリアの発行によって、作成した新しいプランがアクティブ化されることはありません。単に、プランに関する新規または更新情報がデータ・ディクショナリに格納されるのみです。ただし、現在アクティブなプランを変更した場合、そのプランは新しいプラン定義で再度アクティブ化されます。リソース・プランのアクティブ化の詳細は、25-30 ページの「Oracle Database Resource Manager の有効化とプランの切替え」を参照してください。

ペンディング・エリアを作成した後は、そのペンディング・エリアを発行またはクリアするか、ログアウトするまで、他のユーザーはペンディング・エリアを作成できません。

ペンディング・エリアの作成

ペンディング・エリアは CREATE_PENDING_AREA プロシージャを使用して作成します。

例 : ペンディング・エリアの作成

次のスクリプトは、ペンディング・エリアを作成して初期化する PL/SQL ブロックです。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
END;
```

リソース・コンシューマ・グループの作成

リソース・コンシューマ・グループを作成するには、CREATE_CONSUMER_GROUP プロシージャを使用します。次のパラメータを指定できます。

パラメータ	説明
CONSUMER_GROUP	コンシューマ・グループに割り当てる名前。
COMMENT	任意のコメント。
CPU_MTH	非推奨。MGMT_MTH を使用してください。
MGMT_MTH	コンシューマ・グループのセッション間で CPU を分配するためのリソース割当て方法。デフォルトは 'ROUND-ROBIN' で、ラウンドロビン・スケジューラを使用して、セッションの適切な実行を保証します。'RUN-TO-COMPLETION' では、長時間実行セッションを他のセッションより先にスケジュールすることを指定します。この設定によって、長時間実行セッション（バッチ処理など）をより早く完了できるようになります。

例 : リソース・コンシューマ・グループの作成

次のスクリプトは、グループ内のセッションにリソースを割り当てるデフォルトの方法（ROUND-ROBIN）を使用して、OLTP というコンシューマ・グループを作成する PL/SQL ブロックです。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
   CONSUMER_GROUP => 'OLTP',
   COMMENT        => 'OLTP applications');
END;
```

関連項目 :

- 25-36 ページ「コンシューマ・グループの更新」
- 25-36 ページ「コンシューマ・グループの削除」

リソース・プランの作成

リソース・プランを作成するには、CREATE_PLAN プロシージャを使用します。次の表のパラメータを指定できます。最初の 2 つのパラメータは必須です。他のパラメータはオプションです。

パラメータ	説明
PLAN	グループに割り当てる名前。
COMMENT	任意の内容を表現するコメント。
CPU_MTH	非推奨。MGMT_MTH を使用してください。
ACTIVE_SESS_POOL_MTH	アクティブ・セッション・プールのリソース割当て方法。デフォルトは ACTIVE_SESS_POOL_ABSOLUTE で、これは使用可能な唯一の方法です。
PARALLEL_DEGREE_LIMIT_MTH	任意の操作について並列度の制限を指定するためのリソース割当て方法。デフォルトは PARALLEL_DEGREE_LIMIT_ABSOLUTE で、これは使用可能な唯一の方法です。
QUEUEING_MTH	キューイングのリソース割当て方法。キュー内の非アクティブ・セッションをキューから削除し、アクティブ・セッション・プールに追加する順序を制御します。デフォルトは FIFO_TIMEOUT で、これは使用可能な唯一の方法です。
MGMT_MTH	各コンシューマ・グループまたは各サブプランが取得する CPU の量を指定するためのリソース割当て方法。デフォルトの方法 'EMPHASIS' は単一レベルまたは複数レベルのプラン用の方法で、コンシューマ・グループ間での CPU の分配方法の指定にパーセンテージを使用します。'RATIO' は単一レベルのプラン用の方法で、CPU の分配方法の指定に割合を使用します。
SUB_PLAN	TRUE の場合は、プランをトップレベルのプランとして使用できません。サブプランとしてのみ使用できます。デフォルトは FALSE です。

例 : リソース・プランの作成

次のスクリプトは、DAYTIME というリソース・プランを作成する PL/SQL ブロックです。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PLAN(
   PLAN    => 'DAYTIME',
   COMMENT => 'More resources for OLTP applications');
END;
```

RATIO CPU 割当て方法の概要

RATIO 方法は、単一レベルの CPU 割当てのみを使用する単純なプランを対象にした、代替の CPU 割当て方法です。パーセンテージのかわりに、各コンシューマ・グループに与える CPU の割合に対応する数を指定します。RATIO 方法を使用するには、CREATE_PLAN プロシージャの MGMT_MTH 引数に 'RATIO' を設定します。この方法を使用するプランの例は、25-15 ページの「リソース・プラン・ディレクティブの作成」を参照してください。

関連項目 :

- 25-36 ページ「プランの更新」
- 25-37 ページ「プランの削除」

リソース・プラン・ディレクティブの作成

リソース・プラン・ディレクティブを作成するには、CREATE_PLAN_DIRECTIVE プロシージャを使用します。次のパラメータを指定できます。

パラメータ	説明
PLAN	ディレクティブが属しているリソース・プランの名前。
GROUP_OR_SUBPLAN	リソースを割り当てるコンシューマ・グループまたはサブプランの名前。
COMMENT	任意のコメント。
CPU_P1	非推奨。MGMT_P1 を使用してください。
CPU_P2	非推奨。MGMT_P2 を使用してください。
CPU_P3	非推奨。MGMT_P3 を使用してください。
CPU_P4	非推奨。MGMT_P4 を使用してください。
CPU_P5	非推奨。MGMT_P5 を使用してください。
CPU_P6	非推奨。MGMT_P6 を使用してください。
CPU_P7	非推奨。MGMT_P7 を使用してください。
CPU_P8	非推奨。MGMT_P8 を使用してください。
ACTIVE_SESS_POOL_P1	コンシューマ・グループ内で同時にアクティブにできるセッションの最大数を指定します。他のセッションは、非アクティブ・セッション・キューで実行を待機します。デフォルトは UNLIMITED です。
QUEUEING_P1	非アクティブ・セッション・キュー内で実行を待機しているセッションがタイムアウトしてコールが中止されるまでの時間を秒数で指定します。デフォルトは UNLIMITED です。
PARALLEL_DEGREE_LIMIT_P1	任意の操作について並列度の制限を指定します。デフォルトは UNLIMITED です。
SWITCH_GROUP	切替え基準が満たされた場合に、セッションの切替え先となるコンシューマ・グループを指定します。グループ名が 'CANCEL_SQL' の場合は、切替え基準が満たされると、現在のコールは取り消されます。グループ名が 'KILL_SESSION' の場合は、切替え基準が満たされると、そのセッションは強制終了します。デフォルトは NULL です。
SWITCH_TIME	処理が実行されるまでにコールが実行可能な時間を CPU 秒数で指定します。デフォルトは UNLIMITED です。処理は SWITCH_GROUP で指定されます。
SWITCH_ESTIMATE	TRUE の場合は、各コールの実行時間が見積られ、実行時間の見積りが SWITCH_TIME を超える場合は、コールを開始する前にセッションを SWITCH_GROUP に切り替えます。デフォルトは FALSE です。 実行時間の見積りはオプティマイザから取得されます。見積りの正確さは、様々な要因（特にオプティマイザ統計の品質）によって異なります。一般に、統計は± 10 分未満の誤差と考えてください。

パラメータ	説明
MAX_EST_EXEC_TIME	コールに許可される最大の実行時間を CPU 秒数で指定します。あるコールに対するオプティマイザの見積り実行時間が MAX_EST_EXEC_TIME を超える場合、そのコールは実行されず、ORA-07455 が発行されます。オプティマイザによる見積りが示されていない場合、このディレクティブは効果がありません。デフォルトは UNLIMITED です。 見積りの正確さは、様々な要因（特にオプティマイザ統計の品質）によって異なります。一般に、統計は± 10 分未満の誤差と考えてください。
UNDO_POOL	コンシューマ・グループで生成される、コミットされていないトランザクションの UNDO 合計量の最大値を KB で設定します。デフォルトは UNLIMITED です。
MAX_IDLE_TIME	セッションの最大アイドル時間を秒数で示します。デフォルトは NULL で、無制限を意味します。
MAX_IDLE_BLOCKER_TIME	ブロックしているセッションの最大アイドル時間を秒数で示します。デフォルトは NULL で、無制限を意味します。
SWITCH_TIME_IN_CALL	非推奨。SWITCH_FOR_CALL を使用してください。
MGMT_P1	MGMT_MTH パラメータが EMPHASIS に設定されているプランの場合は、レベル 1 の CPU パーセンテージを指定します。MGMT_MTH パラメータが RATIO に設定されている場合は、CPU 使用の比重を指定します。MGMT_P*n* パラメータのデフォルトはすべて NULL です。
MGMT_P2	EMPHASIS の場合は、レベル 2 の CPU パーセンテージを指定します。RATIO には適用しません。
MGMT_P3	EMPHASIS の場合は、レベル 3 の CPU パーセンテージを指定します。RATIO には適用しません。
MGMT_P4	EMPHASIS の場合は、レベル 4 の CPU パーセンテージを指定します。RATIO には適用しません。
MGMT_P5	EMPHASIS の場合は、レベル 5 の CPU パーセンテージを指定します。RATIO には適用しません。
MGMT_P6	EMPHASIS の場合は、レベル 6 の CPU パーセンテージを指定します。RATIO には適用しません。
MGMT_P7	EMPHASIS の場合は、レベル 7 の CPU パーセンテージを指定します。RATIO には適用しません。
MGMT_P8	EMPHASIS の場合は、レベル 8 の CPU パーセンテージを指定します。RATIO には適用しません。
SWITCH_IO_MEGABYTES	処理が実行される前に、セッションで移動（読取りおよび書込み）できる I/O のサイズ（MB）を指定します。デフォルトは UNLIMITED です。処理は SWITCH_GROUP で指定されます。
SWITCH_IO_REQS	処理が実行されるまでにセッションが実行可能な I/O のリクエスト件数を指定します。デフォルトは UNLIMITED です。処理は SWITCH_GROUP で指定されます。
SWITCH_FOR_CALL	TRUE の場合は、SWITCH_TIME、SWITCH_IO_MEGABYTES または SWITCH_IO_REQS に基づいて別のコンシューマ・グループに自動的に切り替えたセッションが、トップレベルのコールが完了した時に当初のコンシューマ・グループに戻されます。デフォルトは NULL です。

例 1:
次のスクリプトは、プラン DAYTIME のリソース・プラン・ディレクティブを作成する PL/SQL ブロックです（ペンディング・エリアには、DAYTIME プランと OLTP コンシューマ・グループがすでに作成されていると仮定します）。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN             => 'DAYTIME',
   GROUP_OR_SUBPLAN => 'OLTP',
   COMMENT          => 'OLTP group',
   MGMT_P1          => 75);
END;
```

このディレクティブは、レベル 1 での CPU リソースの 75% を OLTP コンシューマ・グループに割り当てます。

25-5 ページの図 25-1 にあるプランを完了するには、REPORTING コンシューマ・グループを作成してから、次の PL/SQL ブロックを実行します。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                    => 'DAYTIME',
   GROUP_OR_SUBPLAN        => 'REPORTING',
   COMMENT                 => 'Reporting group',
   MGMT_P1                 => 15,
   PARALLEL_DEGREE_LIMIT_P1 => 8,
   ACTIVE_SESS_POOL_P1     => 4);

DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                    => 'DAYTIME',
   GROUP_OR_SUBPLAN        => 'OTHER_GROUPS',
   COMMENT                 => 'This one is required',
   MGMT_P1                 => 10);
END;
```

このプランでは、コンシューマ・グループ REPORTING に対して操作の最大並列度が 8 に設定されていますが、他のコンシューマ・グループの並列度に制限はありません。また、この REPORTING グループには、同時にアクティブにできるセッションの最大数が 4 に設定されています。

例 2:
この例では、CPU の割当てに RATIO 方法を使用します。この方法は、パーセンテージのかわりに割合を使用します。アプリケーション・パッケージでは、クライアントに対して Gold、Silver および Bronze の 3 種類のサービス・レベルが提供されていると仮定します。GOLD_CG、SILVER_CG および BRONZE_CG という 3 つのコンシューマ・グループを作成し、次のリソース・プランを作成します。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PLAN
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    MGMT_MTH         => 'RATIO',
    COMMENT          => 'Plan that supports three service levels');

DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'GOLD_CG',
    COMMENT          => 'Gold service level customers',
    MGMT_P1          => 10);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'SILVER_CG',
    COMMENT          => 'Silver service level customers',
    MGMT_P1          => 5);
```

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'BRONZE_CG',
    COMMENT          => 'Bronze service level customers',
    MGMT_P1          => 2);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
    (PLAN             => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
    COMMENT          => 'Lowest priority sessions',
    MGMT_P1          => 1);
END;
```

CPU 割当ての割合は、GOLD_CG、SILVER_CG、BRONZE_CG および OTHER_GROUPS コンシューマ・グループに対して、それぞれ 10:5:2:1 の割合です。

セッションが GOLD_CG および SILVER_CG コンシューマ・グループにのみ存在する場合、CPU 割当ての割合は、この 2 つのグループ間で 10:5 になります。

リソース・プラン・ディレクティブの相互作用

トップレベルのプランと複数のサブプランから同じコンシューマ・グループを参照する場合があります。この場合は、同じコンシューマ・グループを参照する複数のリソース・プラン・ディレクティブがあるため、次の規則が適用されます。

- コンシューマ・グループの並列度制限は、すべての入力値の最小値になります。
- コンシューマ・グループのアクティブ・セッション・プールはすべての入力値の合計になり、キューのタイムアウトはすべての入力タイムアウト値の最小値になります。
- コンシューマ・グループの UNDO プールは、すべての入力値の合計になります。
- 複数の SWITCH_TIME、SWITCH_IO_MEGABYTES または SWITCH_IO_REQS がある場合は、Oracle Database Resource Manager（リソース・マネージャ）によって、すべての入力値の中で最も制限が大きい値が選択されます。具体的には、次のようになります。
  - SWITCH_TIME = *min*（入力されるすべての SWITCH_TIME 値）
  - SWITCH_IO_MEGABYTES = *min*（入力されるすべての SWITCH_IO_MEGABYTES 値）
  - SWITCH_IO_REQS = *min*（入力されるすべての SWITCH_IO_REQS 値）
  - SWITCH_ESTIMATE = FALSE よりも SWITCH_ESTIMATE = TRUE が優先する

  注意： 両方のプラン・ディレクティブの切替え時間が同じで、切替えグループが異なる場合、切替え先のグループは、リソース・マネージャによって任意に決められた一方に固定されます。

- いずれかの入力値が TRUE の場合、SWITCH_FOR_CALL は TRUE です。
- 見積り実行時間の最大値には、すべての入力値の中で最も制限の大きい値が選択されます。具体的には、次のようになります。

  MAX_EST_EXEC_TIME = *min*（入力されるすべての MAX_EST_EXEC_TIME 値）
- 最大アイドル時間はすべての入力値の最小値になります。
- 最大アイドル・ブロッカ時間はすべての入力値の最小値になります。

関連項目：

- 25-37 ページ「リソース・プラン・ディレクティブの更新」
- 25-37 ページ「リソース・プラン・ディレクティブの削除」

ペンディング・エリアの妥当性チェック

ペンディング・エリアで変更を追加しているときは、いつでも VALIDATE_PENDING_AREA をコールして、そのペンディング・エリアの妥当性を確認できます。

従う必要がある規則は、次のとおりです。これらの規則が妥当性チェック・プロシージャによってチェックされます。

- プランにループを含めることはできません。ループが発生するのは、サブプランに、プラン階層ではそのサブプランの上位となるプランを参照するディレクティブが含まれている場合です。たとえば、サブプランはトップレベルのプランを参照できません。
- プラン・ディレクティブで参照されるプランやリソース・コンシューマ・グループは、すべて存在している必要があります。
- すべてのプランに、プランまたはリソース・コンシューマ・グループを指すプラン・ディレクティブが必要です。
- 特定のレベルの全パーセンテージの合計は、100 以下であることが必要です。
- アクティブなインスタンスで現在トップレベルのプランとして使用されているプランは、削除できません。
- 次のパラメータは、リソース・コンシューマ・グループを参照するプラン・ディレクティブでのみ使用できます。他のリソース・プランを参照するプラン・ディレクティブでは使用できません。
  - PARALLEL_DEGREE_LIMIT_P1
  - ACTIVE_SESS_POOL_P1
  - QUEUEING_P1
  - SWITCH_GROUP
  - SWITCH_TIME
  - SWITCH_ESTIMATE
  - SWITCH_IO_REQS
  - SWITCH_IO_MEGABYTES
  - MAX_EST_EXEC_TIME
  - UNDO_POOL
  - MAX_IDLE_TIME
  - MAX_IDLE_BLOCKER_TIME
  - SWITCH_FOR_CALL
- アクティブなプランに含まれるリソース・コンシューマ・グループ数は、31 以内にする必要があります。また、プランは最大 31 の子を持つことができます。
- プランとリソース・コンシューマ・グループには、異なる名前を使用する必要があります。
- アクティブなプラン内のどこかに、OTHER_GROUPS のプラン・ディレクティブが存在する必要があります。これによって、現在のアクティブ・プラン内に含まれるコンシューマ・グループのいずれにも属していないセッションに、OTHER_GROUPS のディレクティブで指定したリソースが割り当てられます。

これらの規則のいずれかに違反すると、VALIDATE_PENDING_AREA からエラー・メッセージが返されます。その場合は、変更を加えて問題を修正し、プロシージャを再度コールできます。

プラン・ディレクティブによって参照されない孤立したコンシューマ・グループを作成できます。これによって、当面は使用しないものの、将来的に実装するプランの一部となる、コンシューマ・グループを作成できます。

例 : ペンディング・エリアの妥当性チェック

次のスクリプトは、ペンディング・エリアの妥当性をチェックする PL/SQL ブロックです。

```
BEGIN
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
END;
```

関連項目 : 25-12 ページ「ペンディング・エリアの概要」

ペンディング・エリアの発行

変更の妥当性チェックを完了した後は、SUBMIT_PENDING_AREA プロシージャをコールして変更内容をアクティブにします。

SUBMIT プロシージャでは妥当性チェックも実行されるため、妥当性チェック・プロシージャを別個にコールする必要はありません。ただし、プランを大幅に変更している場合は、通常、変更の妥当性チェックを段階的に実施するほうが問題のデバッグ作業が容易になります。ペンディング・エリア内のすべての変更について妥当性チェックが成功するまで、変更は発行されません（つまり、アクティブになりません）。

SUBMIT_PENDING_AREA プロシージャは、変更の妥当性チェックとコミットに成功すると、ペンディング・エリアをクリア（解除）します。

注意 : VALIDATE_PENDING_AREA が正常に終了しても、SUBMIT_PENDING_AREA のコールが失敗する場合があります。これが起こるのは、VALIDATE_PENDING_AREA をコールしてから SUBMIT_PENDING_AREA をコールするまでの間に、削除しようとしているプランがインスタンスによってロードされた場合などです。

例 : ペンディング・エリアの発行

次のスクリプトは、ペンディング・エリアを発行する PL/SQL ブロックです。

```
BEGIN
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
```

関連項目 : 25-12 ページ「ペンディング・エリアの概要」

ペンディング・エリアのクリア

ペンディング・エリアを随時クリアするためのプロシージャも用意されています。次のスクリプトは、変更のすべてをペンディング・エリアからクリアし、ペンディング・エリアを解除する PL/SQL ブロックです。

```
BEGIN
DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA();
END;
```

CLEAR_PENDING_AREA をコールした後、再び変更を実施するには、その前に CREATE_PENDING_AREA プロシージャをコールする必要があります。

関連項目 : 25-12 ページ「ペンディング・エリアの概要」

リソース・コンシューマ・グループへのセッションの割当て

ここでは、データベース管理者、ユーザーおよびアプリケーションがセッションをリソース・コンシューマ・グループに割り当てる際に使用できる自動および手動の方法について説明します。セッションがリソース・コンシューマ・グループに割り当てられると、Oracle Database Resource Manager（リソース・マネージャ）では、そのセッションに対するリソース割当てが管理の対象となります。

> **注意：** 初期コンシューマ・グループが明示的に割り当てられていないセッションは、コンシューマ・グループ DEFAULT_CONSUMER_GROUP に分類されます。

この項の内容は、次のとおりです。

リソース・コンシューマ・グループへのセッション割当ての概要

リソース・マネージャを有効にする前に、ユーザー・セッションをリソース・コンシューマ・グループに割り当てる方法を指定する必要があります。これには、マッピング・ルールを作成します。このルールにより、リソース・マネージャは、セッションの起動時にセッション属性に基づいて各セッションをコンシューマ・グループに自動的に割り当てることができます。セッションがその初期コンシューマ・グループに割り当てられ実行された後は、プロシージャをコールして、セッションを別のコンシューマ・グループに手動で切り替えることができます。この操作は、リソースを過剰に使用しているセッションを、よりリソース割当ての制限されたコンシューマ・グループに移動する必要がある場合などに実行します。あるコンシューマ・グループから別のコンシューマ・グループにそれぞれのセッションを切り替えることができるように、ユーザーやアプリケーションにスイッチ特権を付与することもできます。

セッション属性に変更がある場合やセッションが指定のリソース使用制限を超える場合は、あるコンシューマ・グループから別の（通常は優先度の低い）コンシューマ・グループにセッションを自動的に切り替えることもできます。

初期リソース・コンシューマ・グループの割当て

セッションの初期コンシューマ・グループは、管理者が構成するマッピング・ルールによって決まります。マッピング・ルールの構成方法は、25-25 ページの「コンシューマ・グループへのセッションのマッピング・ルールの指定」を参照してください。新しいセッションに適用するマッピング・ルールがない場合、セッションのデフォルトのコンシューマ・グループは、そのセッションを起動したユーザーの INITIAL_RSRC_CONSUMER_GROUP 属性を使用して指定されます。この属性の値は、*_USER ビューの INITIAL_RSRC_CONSUMER_GROUP 列を表示することで確認できます。

手動によるリソース・コンシューマ・グループの切替え

DBMS_RESOURCE_MANAGER PL/SQL パッケージには、管理者が実行中のセッションのリソース・コンシューマ・グループを変更できるように、2 つのプロシージャが用意されています。この 2 つのプロシージャでは、コーディネータのセッションに対応付けられたパラレル実行サーバー・セッションのコンシューマ・グループも変更できます。これらのプロシージャで行った変更は永続的ではなく、現行セッションのみに影響します。ユーザーの初期コンシューマ・グループも変更されません。

あるユーザーが CPU を過剰に使用している場合は、そのユーザーのセッションを停止（終了）するかわりに、ユーザーのコンシューマ・グループをリソースの割当てが低いグループに変更できます。

単一のセッションの切替え

SWITCH_CONSUMER_GROUP_FOR_SESS プロシージャは、指定したセッションを、指定したリソース・コンシューマ・グループに即時に移動します。このプロシージャを使用すると、セッションの優先度を実質的に変更できます。

次のスクリプトは、特定のセッションを新しいコンシューマ・グループに切り替える PL/SQL ブロックです。セッション識別子（SID）が 17、セッション・シリアル番号（SERIAL#）が 12345 のセッションを移動します。新しいコンシューマ・グループは HIGH_PRIORITY コンシューマ・グループです。

```
BEGIN
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS ('17', '12345',
   'HIGH_PRIORITY');
END;
```

SID、セッション・シリアル番号、およびセッションの現行リソース・コンシューマ・グループは、V$SESSION ビューを使用して確認できます。

> **関連項目：** V$SESSION ビューの詳細は、『Oracle Database リファレンス』を参照してください。

ユーザーの全セッションの切替え

SWITCH_CONSUMER_GROUP_FOR_USER プロシージャは、指定したユーザー名に関連するすべてのセッションのリソース・コンシューマ・グループを変更します。次のスクリプトは、ユーザー SCOTT に属しているすべてのセッションを LOW_GROUP コンシューマ・グループに切り替える PL/SQL ブロックです。

```
BEGIN
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER ('SCOTT',
    'LOW_GROUP');
END;
```

リソース・コンシューマ・グループの自動切替えの指定

特定の条件を満たしたときに、セッションを別のコンシューマ・グループに自動的に切り替えるように、リソース・マネージャを構成できます。自動切替えは、次の場合に実行されます。

- セッション属性が変更され、新しいマッピング・ルールが有効になった場合。
- セッションが、そのコンシューマ・グループに設定されている CPU または I/O リソースの使用制限を超えた場合。

次の各項では、詳細について説明します。

- マッピング・ルールを使用した自動切替えの指定
- リソース制限の設定による自動切替えの指定

マッピング・ルールを使用した自動切替えの指定

セッションの実行中にセッション属性が変更されると、コンシューマ・グループへのセッションのマッピング・ルールが再評価されます。新しいルールが有効な場合、そのセッションは別のコンシューマ・グループに移動する可能性があります。詳細は、25-25 ページの「コンシューマ・グループへのセッションのマッピング・ルールの指定」を参照してください。

リソース制限の設定による自動切替えの指定

コンシューマ・グループに対してリソース・プラン・ディレクティブを作成する場合は、そのグループのセッションに対して CPU および I/O リソースの使用制限を指定できます。そのためには、セッション内の単一のコールが制限のいずれかを超過した場合に実行する処理を指定します。次のような処理が考えられます。

- セッションを指定のコンシューマ・グループに動的に切り替えます。

  通常、ターゲット・コンシューマ・グループは、より低いリソース割当てのコンシューマ・グループです。セッションのユーザーには、新しいコンシューマ・グループに対するスイッチ特権が必要です。スイッチ特権がない場合は切り替えることができません。詳細は、25-29 ページの「スイッチ特権の付与と取消し」を参照してください。
- セッションを停止（終了）します。
- セッションの現行の SQL 文を中止します。

このタイプのセッション自動切替えに関連するリソース・プラン・ディレクティブの属性は、次のとおりです。

- `SWITCH_GROUP`
- `SWITCH_TIME`
- `SWITCH_ESTIMATE`
- `SWITCH_IO_MEGABYTES`
- `SWITCH_IO_REQS`
- `SWITCH_FOR_CALL`

これらの属性の詳細は、25-15 ページの「リソース・プラン・ディレクティブの作成」を参照してください。

切替えは、実行中のセッションがリソースを使用している場合に発生します。ユーザー入力の待機時や CPU サイクルの待機中には発生しません。切替え後のグループのアクティブ・セッション・プールがいっぱいの場合も、切り替えたセッションは引き続き実行することを許可されます。このような状況では、コンシューマ・グループは、アクティブ・セッション・プールで指定された数より多いセッションを実行できます。

`SWITCH_FOR_CALL` は、中間層のサーバーがセッション・プーリングを使用している 3 層アプリケーションの場合に有効です。PL/SQL の**最上位コール**は、1 つのコールとして処理される PL/SQL ブロック全体です。SQL の最上位コールは個々の SQL 文です。

次に、リソース制限に基づいた自動切替えの例を示します。

例 1:

次のスクリプトは、OLTP グループのリソース・プラン・ディレクティブを作成する PL/SQL ブロックです。このディレクティブによって、セッション内のコールが CPU 時間の 5 秒を超過した場合は、そのグループ内のセッションが LOW_GROUP コンシューマ・グループに切り替わります。この例は、予想外に長い問合せに起因するリソースの過剰使用を防止します。通常、切替え先のコンシューマ・グループは、より低いリソース割当てのコンシューマ・グループです。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN             => 'DAYTIME',
   GROUP_OR_SUBPLAN => 'OLTP',
   COMMENT          => 'OLTP group',
   MGMT_P1          => 75,
   SWITCH_GROUP     => 'LOW_GROUP',
   SWITCH_TIME      => 5);
END;
```

例 2

次のスクリプトは、OLTP グループのリソース・プラン・ディレクティブを作成する PL/SQL ブロックです。このディレクティブによって、セッションが 10,000 回の I/O リクエストまたは 2,500MB のデータ転送を超過した場合は、そのグループ内のセッションが LOW_GROUP コンシューマ・グループに一時的に切り替わります。切り替えられたセッションは、問題となった最上位コールが完了した後、元のグループに戻ります。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                => 'DAYTIME',
   GROUP_OR_SUBPLAN    => 'OLTP',
   COMMENT             => 'OLTP group',
   MGMT_P1             => 75,
   SWITCH_GROUP        => 'LOW_GROUP',
   SWITCH_IO_REQS      => 10000,
   SWITCH_IO_MEGABYTES => 2500,
   SWITCH_FOR_CALL     => TRUE);
END;
```

例 3:

次のスクリプトは、OLTP グループのリソース・プラン・ディレクティブを作成する PL/SQL ブロックです。このディレクティブによって、60 秒の CPU 時間を超過したセッションは停止（終了）します。この例は、リソース集中型の問合せに起因するリソースの過剰使用を防止します。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN             => 'DAYTIME',
   GROUP_OR_SUBPLAN => 'OLTP',
   COMMENT          => 'OLTP group',
   MGMT_P1          => 75,
   SWITCH_GROUP     => 'KILL_SESSION',
   SWITCH_TIME      => 60);
END;
```

関連項目： 25-15 ページ「リソース・プラン・ディレクティブの作成」

コンシューマ・グループへのセッションのマッピング・ルールの指定

ここでは、コンシューマ・グループへのセッションのマッピング・ルールに関するバックグラウンド情報を提供し、ルールの作成方法と優先度の設定方法について説明します。この項の内容は、次のとおりです。

コンシューマ・グループへのセッションのマッピング・ルールの概要

コンシューマ・グループへのセッションのマッピング・ルールを作成すると、次のことができます。

- セッションの初期コンシューマ・グループをセッション属性に基づいて指定できます。
- リソース・マネージャを使用して、実行中のセッションをセッション属性の変更に基づいて別のコンシューマ・グループに動的に切り替えることができます。

このマッピング・ルールは、セッション属性（ユーザー名、セッションがデータベースへの接続に使用したサービス、クライアント・プログラムの名前など）に基づいています。

マッピング・ルール間の競合を解決するために、リソース・マネージャでは、各ルールが優先度別に順序付けされます。たとえば、ユーザー SCOTT が SALES サービスを使用してデータベースに接続すると仮定します。あるマッピング・ルールには、ユーザー SCOTT が MED_PRIORITY コンシューマ・グループで起動することが記載され、別のルールには、SALES サービスに接続するセッションは SALES コンシューマ・グループで起動することが記載されている場合、この競合は、マッピング・ルールの優先度によって解決されます。

マッピング・ルールの基本となるセッション属性には、ログイン属性とランタイム属性の 2 つのタイプがあります。ログイン属性が有効なのは、セッションのログイン時、つまり、リソース・マネージャによってセッションの初期コンシューマ・グループが決定されるときのみです。一方、すでにログインしているセッションは、ランタイム属性の変更に基づいて、その後、別のコンシューマ・グループに再割当てできます。

コンシューマ・グループへのセッションの自動割当てを構成するには、SET_CONSUMER_GROUP_MAPPING および SET_CONSUMER_GROUP_MAPPING_PRI プロシージャを使用します。これらのプロシージャに対してはペンディング・エリアを使用する必要があります。（ペンディング・エリアを作成してプロシージャを実行し、必要に応じてペンディング・エリアの妥当性をチェックしてから、そのペンディング・エリアを発行する必要があります。ペンディング・エリアの使用例は、25-11 ページの「複雑なリソース・プランの作成」を参照してください。）

セッションは、マッピング・ルールを介して様々な時点で特定のコンシューマ・グループに自動的に切り替えられます。

- 最初のセッションのログイン時には、マッピング・ルールが評価されてそのセッションの初期グループが判別されます。
- セッション属性が新しい値に動的に変更されると（可能性があるのはランタイム属性のみ）、マッピング・ルールが再評価され、セッションが別のコンシューマ・グループに切り替わる場合があります。

これらのルールについては、次の 2 つの点に注意してください。

- マッピング・ルールが提供されているランタイム属性がそれまでと同じ値に設定されている場合、切替えは発生しません。
- セッションは、すでに配置されているグループと同じコンシューマ・グループに切り替えることができます。この場合は、切替えが実行されることで、別のグループに切り替わる際に一般的に初期化されるセッション統計がゼロに初期化されます。このような統計の例には、セッションの ACTIVE_TIME_IN_GROUP 値があります。

関連項目：

- 25-21 ページ「初期リソース・コンシューマ・グループの割当て」
- 25-23 ページ「マッピング・ルールを使用した自動切替えの指定」

コンシューマ・グループのマッピング・ルールの作成

セッションのマッピング・ルールは、一連の属性とコンシューマ・グループのペアで構成され、このペアによって、セッションとコンシューマ・グループとの組合せ方法が決定します。1 つのセッション属性をコンシューマ・グループにマップするには、SET_CONSUMER_GROUP_MAPPING プロシージャを使用します。このプロシージャのパラメータは、次のとおりです。

パラメータ	説明
ATTRIBUTE	ログインまたはランタイムのセッション属性タイプ。パッケージ定数として指定されています。
VALUE	マッピングされる属性の値。
CONSUMER_GROUP	マッピング先のコンシューマ・グループ。

ATTRIBUTE には、次のいずれかを指定できます。

属性	タイプ	説明
ORACLE_USER	ログイン	Oracle Database ユーザー名。
SERVICE_NAME	ログイン	クライアントが接続の確立に使用したサービス名。
CLIENT_OS_USER	ログイン	ログインしているクライアントのオペレーティング・システム・ユーザー名。
CLIENT_PROGRAM	ログイン	サーバーへのログインに使用されたクライアント・プログラム名。
CLIENT_MACHINE	ログイン	クライアントが接続に使用しているコンピュータ名。
MODULE_NAME	ランタイム	DBMS_APPLICATION_INFO.SET_MODULE プロシージャによる設定、またはそれに相当する OCI 属性の設定に従って現在実行されているアプリケーション内のモジュール名。
MODULE_NAME_ACTION	ランタイム	次のプロシージャのいずれかによる設定、またはそれに相当する OCI 属性の設定に従って実行されている現行モジュールと処理の組合せ。 ■ DBMS_APPLICATION_INFO.SET_MODULE ■ DBMS_APPLICATION_INFO.SET_ACTION この属性にはモジュール名を指定し、その後ろにピリオド（.）および処理名を順に続けます（*module_name.action_name*）。
SERVICE_MODULE	ランタイム	*service_name.module_name* の書式によるサービス名およびモジュール名の組合せ。
SERVICE_MODULE_ACTION	ランタイム	*service_name.module_name.action_name* の書式によるサービス名、モジュール名および処理名の組合せ。

たとえば、次の PL/SQL ブロックでは、ログインするたびに、ユーザー SCOTT が DEV_GROUP コンシューマ・グループにマップされます。

```
BEGIN
DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING
     (DBMS_RESOURCE_MANAGER.ORACLE_USER, 'SCOTT', 'DEV_GROUP');
END;
```

この場合も、SET_CONSUMER_GROUP_MAPPING プロシージャを実行する前にペンディング・エリアを作成する必要があります。

マッピング・ルールの優先度の作成

競合するマッピング・ルールを解決するために、最も高い重要度のセッション属性から最も低いセッション属性まで優先度を設定できます。各属性に、1（最も高い重要度）～ 10（最も低い重要度）の一意の整数を設定するには、SET_CONSUMER_GROUP_MAPPING_PRI プロシージャを使用します。次の例は、この優先度の設定方法を示しています。

```
BEGIN
DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING_PRI(
    EXPLICIT => 1,
    SERVICE_MODULE_ACTION => 2,
    SERVICE_MODULE => 3,
    MODULE_NAME_ACTION => 4,
    MODULE_NAME => 5,
    SERVICE_NAME => 6,
    ORACLE_USER => 7,
    CLIENT_PROGRAM => 8,
    CLIENT_OS_USER => 9,
    CLIENT_MACHINE => 10);
END;
```

この例では、データベース・ユーザー名の優先度は 7（重要度が低い）に設定され、モジュール名の優先度は 5（重要度が高い）に設定されています。

注意： SET_CONSUMER_GROUP_MAPPING_PRI には、疑似属性 EXPLICIT を引数として指定する必要があります。この疑似属性には 1 を設定します。これは、明示的なコンシューマ・グループの切替えの優先度が最も高いことを示します。コンシューマ・グループの明示的な切替えには、次のパッケージ・プロシージャを使用します。これらのプロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

- DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP
- DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS
- DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER

マッピング・ルールの優先度がどのように機能するかを示すために、DEV_GROUP コンシューマ・グループへのユーザー SCOTT のマッピングの他に、次のようなモジュール名のマッピング・ルールがあると仮定します。

```
BEGIN
DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING
     (DBMS_RESOURCE_MANAGER.MODULE_NAME, 'BACKUP', 'LOW_PRIORITY');
END;
```

ユーザー SCOTT のセッションによって、モジュール名が BACKUP に設定された場合、そのセッションは LOW_PRIORITY コンシューマ・グループに再割当てされます。これは、モジュール名マッピングのほうが、データベース・ユーザー・マッピングより優先度が高いためです。

セッション属性の現在の優先度を確認するには、ビュー DBA_RSRC_MAPPING_PRIORITY を問い合せます。

認可されていないクライアントがそのセッション属性を設定して優先度の高いコンシューマ・グループにクライアント自身をマッピングしないように、コンシューマ・グループに対してユーザー・スイッチ特権が適用されます。したがって、特定のセッションの属性がマッピング・ペアと一致した場合でも、指定したコンシューマ・グループに対するスイッチ特権がそのセッションに付与されていない場合は、そのマッピング・ルールは無視されます。

ユーザーまたはアプリケーションに対する手動によるコンシューマ・グループの切替えの有効化

管理者は、ユーザーが DBMS_SESSION パッケージの SWITCH_CURRENT_CONSUMER_GROUP プロシージャを使用して、現在のコンシューマ・グループを切り替えられるように、スイッチ特権をユーザーに付与できます。ユーザーが対話型セッション（SQL*Plus など）でこのプロシージャを実行するか、アプリケーションでこのプロシージャをコールすることで、セッションを切り替え、実質的にその優先度を動的に変更できます。

ユーザーは、SWITCH_CURRENT_CONSUMER_GROUP プロシージャを使用して、スイッチ特権があるコンシューマ・グループにのみ切り替えることができます。他のプロシージャからこのプロシージャがコールされた場合、ユーザーはコール側のプロシージャの所有者がスイッチ特権を持っているコンシューマ・グループに切り替えることができます。

このプロシージャのパラメータは、次のとおりです。

パラメータ	説明
NEW_CONSUMER_GROUP	切替え先のコンシューマ・グループ。
OLD_CONSUMER_GROUP	切替え元のコンシューマ・グループの名前が戻ります。このパラメータは、後で元のコンシューマ・グループに再び切り替えるときに使用できます。
INITIAL_GROUP_ON_ERROR	切替えエラー発生時の動作を制御します。 TRUE に設定すると、エラーが発生した場合にユーザーは初期コンシューマ・グループに切り替わります。 FALSE の場合は、そのままエラーが出力されます。

次の SQL*Plus セッションは、新しいコンシューマ・グループへの切替え方法を示しています。出力パラメータ old_group の値が出力されているため、変更前のコンシューマ・グループ名がどのように保存されているかがわかります。

```
SET serveroutput on
DECLARE
    old_group varchar2(30);
BEGIN
DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP('OLTP', old_group, FALSE);
DBMS_OUTPUT.PUT_LINE('OLD GROUP = ' || old_group);
END;
/
```

次の行が出力されます。

```
OLD GROUP = DEFAULT_CONSUMER_GROUP
```

リソース・マネージャでは、セッションをすでに配置されているコンシューマ・グループに切り替えるために SWITCH_CURRENT_CONSUMER_GROUP プロシージャがコールされている場合でも、切替えが発生したとみなされます。

注意： リソース・マネージャは、一般的なデータベース・ユーザー名を使用してアプリケーションにログインしている環境でも機能します。DBMS_SESSION パッケージは、セッションの開始時、または特定のモジュールがコールされたときにセッションのコンシューマ・グループ割当てを切り替えるために使用できます。

関連項目： DBMS_SESSION パッケージのその他の例と詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

スイッチ特権の付与と取消し

DBMS_RESOURCE_MANAGER_PRIVS PL/SQL パッケージを使用すると、ユーザー、ロールまたは PUBLIC にスイッチ特権を付与したり、取り消すことができます。ユーザーまたはアプリケーションは、スイッチ特権を使用してセッションを指定のリソース・コンシューマ・グループに切り替えることができます。コンシューマ・グループへのセッションのマッピング・ルールに指定されているコンシューマ・グループ、またはリソース・プラン・ディレクティブの SWITCH_GROUP パラメータに指定されているコンシューマ・グループに、セッションを自動的に切り替えることもできます。このパッケージでは、スイッチ特権を取り消すこともできます。次の表に、関連するパッケージ・プロシージャの一覧を示します。

プロシージャ	説明
GRANT_SWITCH_CONSUMER_GROUP	ユーザー、ロールまたは PUBLIC に、指定したリソース・コンシューマ・グループに切り替える許可を付与します。
REVOKE_SWITCH_CONSUMER_GROUP	ユーザー、ロールまたは PUBLIC から、指定したリソース・コンシューマ・グループに切り替える許可を取り消します。

DEFAULT_CONSUMER_GROUP では、スイッチ特権が PUBLIC に付与されています。したがって、すべてのユーザーに、このコンシューマ・グループに対するスイッチ特権が自動的に付与されます。

関連項目：

- 25-28 ページ「ユーザーまたはアプリケーションに対する手動によるコンシューマ・グループの切替えの有効化」
- 25-23 ページ「リソース・コンシューマ・グループの自動切替えの指定」

スイッチ特権の付与

次の例では、コンシューマ・グループ OLTP に切り替える権限をユーザー SCOTT に付与しています。

```
BEGIN
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP (
   GRANTEE_NAME   => 'SCOTT',
   CONSUMER_GROUP => 'OLTP',
   GRANT_OPTION   =>  TRUE);
END;
```

ユーザー SCOTT には、OLTP のスイッチ特権を他のユーザーに付与する許可も付与されます。

特定のリソース・コンシューマ・グループに切り替えるロールに権限を付与すると、そのロールを付与されたユーザーおよびそのロールを有効にしているユーザーは、各自のセッションをそのコンシューマ・グループに切り替えることができます。

特定のコンシューマ・グループに切り替える許可を PUBLIC に付与すると、だれでもそのグループに切り替えることができます。

GRANT_OPTION 引数が TRUE の場合、コンシューマ・グループのスイッチ特権が付与されたユーザーは、そのコンシューマ・グループのスイッチ特権を他のユーザーに付与することもできます。

スイッチ特権の取消し

次の例では、コンシューマ・グループ OLTP に切り替えるユーザー SCOTT の権限を取り消しています。

```
BEGIN
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (
   REVOKEE_NAME   => 'SCOTT',
   CONSUMER_GROUP => 'OLTP');
END;
```

特定のコンシューマ・グループに対するユーザーのスイッチ特権を取り消すと、そのユーザーがそのコンシューマ・グループに切り替えようとすると失敗します。ユーザーから初期コンシューマ・グループのスイッチ特権を取り消すと、そのユーザーはログイン時に自動的に DEFAULT_CONSUMER_GROUP に割り当てられます。

コンシューマ・グループに対するスイッチ特権をロールから取り消すと、そのロールを介してのみコンシューマ・グループのスイッチ特権を与えられているユーザーは、そのコンシューマ・グループに切り替えることができなくなります。

コンシューマ・グループに対するスイッチ特権を PUBLIC から取り消すと、直接またはロールを介してスイッチ特権を明示的に割り当てられているユーザー以外は、そのコンシューマ・グループに切り替えることができなくなります。

Oracle Database Resource Manager の有効化とプランの切替え

Oracle Database Resource Manager（リソース・マネージャ）を有効にするには、RESOURCE_MANAGER_PLAN 初期化パラメータを設定します。このパラメータは、トップレベルのプランを指定し、現行のインスタンスで使用するプランを識別します。このパラメータでプランを指定しない場合、リソース・マネージャはアクティブになりません。

デフォルトでは、リソース・マネージャは有効化されていません。

テキスト形式の初期化パラメータ・ファイルに次のように記述すると、データベースの起動時にリソース・マネージャがアクティブ化され、トップレベルのプランが mydb_plan として設定されます。

```
RESOURCE_MANAGER_PLAN = mydb_plan
```

DBMS_RESOURCE_MANAGER.SWITCH_PLAN パッケージ・プロシージャまたは ALTER SYSTEM 文を使用して、リソース・マネージャをアクティブ化（または解除）したり、現行のトップレベル・プランを変更することもできます。

次の SQL 文は、トップレベルのプランを mydb_plan に設定し、リソース・マネージャをアクティブ化します（アクティブ化されていない場合）。

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'mydb_plan';
```

指定したプランがデータ・ディクショナリに存在しない場合は、エラー・メッセージが返されます。

Oracle Scheduler のウィンドウでのリソース・マネージャの自動有効化

リソース・プランを指定する Oracle Scheduler のウィンドウがオープン状態の場合、リソース・マネージャは自動的にアクティブ化されます。スケジューラのウィンドウがクローズ状態の場合、そのウィンドウに関連付けられているリソース・プランは無効で、スケジューラのウィンドウがオープンする前に実行されていたリソース・プランが再度有効になります（ウィンドウのオープン前に有効なリソース・プランがなかった場合、リソース・マネージャはウィンドウのクローズ時に再度無効になります）。Oracle Real Application Clusters 環境では、スケジューラのウィンドウはすべてのインスタンスに適用されます。したがって、ウィンドウのリソース・プランはあらゆるインスタンスで有効になります。

デフォルトでは、一連の自動化メンテナンス・タスクは**メンテナンス・ウィンドウ**内で実行されます。このメンテナンス・ウィンドウは、事前定義のスケジューラのウィンドウで、MAINTENANCE_WINDOW_GROUP ウィンドウ・グループのメンバーであり、

DEFAULT_MAINTENANCE_PLAN リソース・プランを指定するウィンドウです。したがって、デフォルトでは、リソース・マネージャはメンテナンス・ウィンドウの期間中にアクティブ化されます。

関連項目：

- 26-15 ページ「ウィンドウ」
- 第 24 章「自動データベース・メンテナンス・タスクの管理」

Oracle Scheduler のウィンドウでのプラン切替えの無効化

場合によっては、スケジューラのウィンドウ境界でのリソース・マネージャ・プランの自動変更は、望ましくない可能性があります。たとえば、終了する必要のある重要なタスクがあり、リソース・マネージャ・プランにタスクの優先度を設定してある場合、設定を変更するまでは同じプランのままであることを想定しています。しかし、スケジューラのウィンドウはプランの設定後にアクティブになるため、タスクの実行中にリソース・マネージャ・プランが変更される可能性があります。

この状況を回避するには、次の SQL 文に示すように、RESOURCE_MANAGER_PLAN 初期化パラメータをシステムに必要なプランの名前に設定し、その名前の前に「FORCE:」を付加します。

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'FORCE:mydb_plan';
```

接頭辞 FORCE を使用することで、現行のリソース・プランの変更は、データベース管理者が RESOURCE_MANAGER_PLAN 初期化パラメータの値を変更した場合のみ可能であることを示します。この制限を解除するには、プラン名の前に「FORCE:」を付加せずに、そのコマンドを再実行します。

DBMS_RESOURCE_MANAGER.SWITCH_PLAN パッケージ・プロシージャには同様の機能があります。

関連項目： DBMS_RESOURCE_MANAGER.SWITCH_PLAN の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

リソース・マネージャの無効化

リソース・マネージャを無効にするには、次の手順を完了します。

1. 次の SQL 文を発行します。

   ```
   ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
   ```

2. すべての Oracle Scheduler のウィンドウからリソース・マネージャの関連を削除します。

   そのためには、resource_plan 属性のリソース・プランを参照するすべてのスケジューラのウィンドウについて、DBMS_SCHEDULER.SET_ATTRIBUTE プロシージャを使用して、resource_plan を空の文字列（''）に設定します。SYS ユーザーでログインしていない場合は、SYS スキーマ名を使用してウィンドウ名を修飾します。スケジューラのウィンドウは、DBA_SCHEDULER_WINDOWS データ・ディクショナリ・ビューで表示できます。詳細は、27-32 ページの「ウィンドウの変更」および『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

注意： デフォルトでは、すべてのメンテナンス・ウィンドウが DEFAULT_MAINTENANCE_PLAN リソース・プランを参照します。リソース・マネージャを完全に無効にするには、すべてのメンテナンス・ウィンドウを変更してこのプランを削除する必要があります。ただし、自動化メンテナンス・タスクによるリソース使用が規制されなくなるため、他のセッションのパフォーマンスに悪影響を与える可能性があることに注意してください。メンテナンス・ウィンドウの詳細は、第 24 章「自動データベース・メンテナンス・タスクの管理」を参照してください。

各種の方法を組み合せた Oracle Database Resource Manager の例

ここでは、リソース・プランの例をいくつか示します。ここで取り上げる例は、次のとおりです。

- 複数レベルのプランの例
- 各種のリソース割当て方法を使用した例
- オラクル社が提供する複合ワークロード・プラン

複数レベルのプランの例

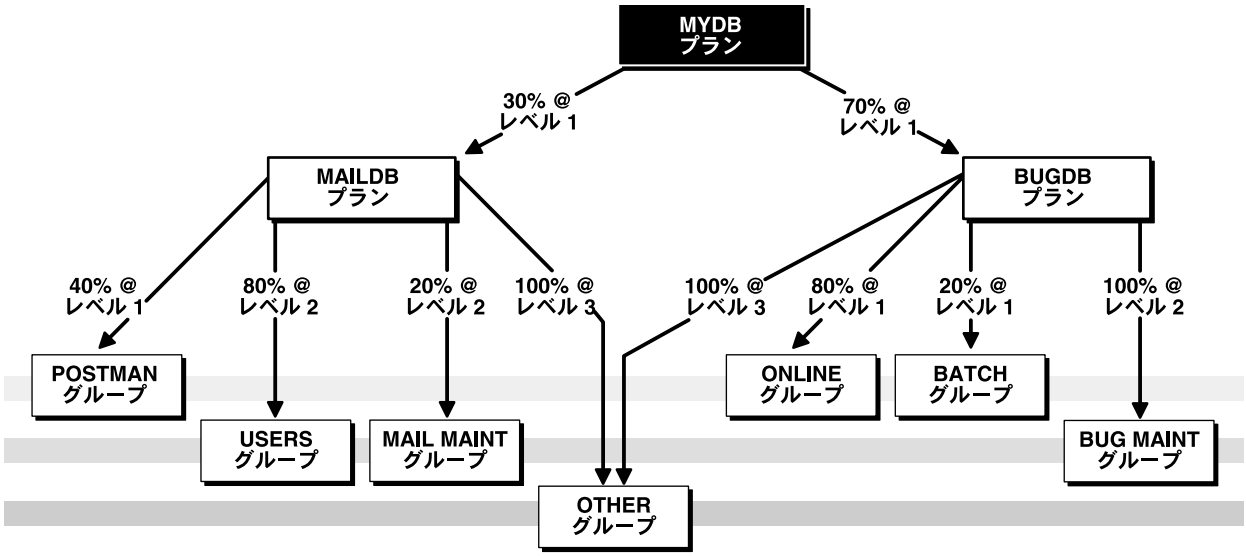
次のスクリプトは、25-33 ページの図 25-3 に示されている複数レベルのプランを作成する PL/SQL ブロックです。デフォルトのリソース割当て方法の設定は、すべてのプランとリソース・コンシューマ・グループに使用されます。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'bugdb_plan',
   COMMENT => 'Resource plan/method for bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'maildb_plan',
   COMMENT => 'Resource plan/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'mydb_plan',
   COMMENT => 'Resource plan/method for bug and mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Online_group',
   COMMENT => 'Resource consumer group/method for online bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Batch_group',
   COMMENT => 'Resource consumer group/method for batch job bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Maint_group',
   COMMENT => 'Resource consumer group/method for users sessions for bug db maint');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Users_group',
   COMMENT => 'Resource consumer group/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Postman_group',
   COMMENT => 'Resource consumer group/method for mail postman');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_Maint_group',
   COMMENT => 'Resource consumer group/method for users sessions for mail db maint');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Online_group',
   COMMENT => 'online bug users sessions at level 1', MGMT_P1 => 80, MGMT_P2=> 0);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Batch_group',
   COMMENT => 'batch bug users sessions at level 1', MGMT_P1 => 20, MGMT_P2 => 0,
   PARALLEL_DEGREE_LIMIT_P1 => 8);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Bug_Maint_group',
   COMMENT => 'bug maintenance users sessions at level 2', MGMT_P1 => 0, MGMT_P2 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', MGMT_P1 => 0, MGMT_P2 => 0,
   MGMT_P3 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Postman_group',
   COMMENT => 'mail postman at level 1', MGMT_P1 => 40, MGMT_P2 => 0);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Users_group',
   COMMENT => 'mail users sessions at level 2', MGMT_P1 => 0, MGMT_P2 => 80);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Mail_Maint_group',
   COMMENT => 'mail maintenance users sessions at level 2', MGMT_P1 => 0, MGMT_P2 => 20);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', MGMT_P1 => 0, MGMT_P2 => 0,
   MGMT_P3 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
   GROUP_OR_SUBPLAN => 'maildb_plan',
```

```
   COMMENT=> 'all mail users sessions at level 1', MGMT_P1 => 30);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
   GROUP_OR_SUBPLAN => 'bugdb_plan',
   COMMENT => 'all bug users sessions at level 1', MGMT_P1 => 70);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
```

妥当性チェックは SUBMIT_PENDING_AREA によって暗黙的に実行されるので、直前の VALIDATE_PENDING_AREA のコールはオプションです。

図 25-3 複数レベルのプラン・スキーマ



このプラン・スキーマでは、次のように CPU リソースが割り当てられています。

- mydb_plan では、CPU の 30% が maildb_plan サブプランに割り当てられ、70% が bugdb_plan サブプランに割り当てられます。サブプランは両方ともレベル 1 です。mydb_plan 自体にはレベル 1 の下にレベルがないため、レベル 1 で一方のサブプランが使用していないリソース割当ては、その兄弟サブプランが使用できます。したがって、maildb_plan が CPU の 20% のみを使用する場合、CPU の 80% は bugdb_plan が使用できます。
- maildb_plan ではレベル 1、2 および 3 で割当てを定義し、bugdb_plan ではレベル 1 および 2 で割当てを定義します。これらのサブプランの各レベルは、その親プラン mydb_plan のレベルから独立しています。つまり、プラン・スキーマのすべてのプランとサブプランには、独自のレベル 1、レベル 2、レベル 3 などがあります。
- maildb_plan に割り当てられた CPU の 30% について、その 40%（実質的には CPU 合計の 12%）はレベル 1 の Postman_group に割り当てられます。Postman_group にはレベル 1 で兄弟がないため、レベル 1 では暗黙的に 60% が残っています。この 60% は、レベル 2 の Users_group と Mail_Maint_group がそれぞれ 80% と 20% の割合で共有します。この 60% の他に、Users_group と Mail_Maint_group は、レベル 1 の Postman_group が使用していないリソース（40% の一部）も使用できます。
- レベル 2 の Users_group または Mail_Maint_group が使用していない CPU リソースは OTHER_GROUPS に割り当てられます。これは、複数レベルのプランの場合、使用されていないリソースは、同じレベルの兄弟ではなく、次の下位レベルのコンシューマ・グループまたはサブプランに再配分されるためです。したがって、Users_group が 80% ではなく 70% のみを使用している場合は、残りの 10% を Mail_Maint_group で使用することはできません。この 10% を使用できるのは、レベル 3 の OTHER_GROUPS のみです。

各種のリソース割当て方法を使用した例

ここで示す例は、パッケージ化された Enterprise Resource Planning（ERP）または Customer Relationship Management（CRM）アプリケーションをサポートするデータベース用のプランを表します。このような環境で必要な処理は多種多様です。大規模なパラレル問合せを含む長時間実行のバッチ・ジョブとともに、短いトランザクションや短い問合せが混在している場合があります。その目的は、バッチ・ジョブをパラレルに実行しながら、オンライン・トランザクション処理（OLTP）の適切な応答時間を得ることにあります。

次の表にプランがまとめられています。

グループ	CPU リソース割当て %	アクティブ・セッション・プール・パラメータ	コンシューマ・グループの自動切替え	最大見積り実行時間	UNDO プール
oltp	レベル 1: 80%		切替え先グループ : batch 切替え時間 : 3 秒		200K
batch	レベル 2: 100%	プール・サイズ : 5 タイムアウト : 600 秒	--	3600 秒	--
OTHER_GROUPS	レベル 3: 100%	--	--		--

レベル 1 の oltp に CPU の 80% のみを割り当てることで、batch には最低 20% が保証され、さらに、oltp が使用しない CPU リソースを使用できます。ただし、OTHER_GROUPS に対する CPU リソースの保証はありません。OTHER_GROUPS が CPU リソースを取得するのは、batch がその割当てを使用できない場合のみです。類似するプランでは、レベル 1 の oltp に 80%、batch に 20% が付与され、レベル 2 の OTHER_GROUPS には 100% が付与されます。このプランでは、oltp が使用していない CPU 割当てが、batch ではなく、OTHER_GROUPS に付与されます。

次の文は、この表のプランを erp_plan という名前で作成します。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'erp_plan',
  COMMENT => 'Resource plan/method for ERP Database');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'oltp',
  COMMENT => 'Resource consumer group/method for OLTP jobs');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'batch',
  COMMENT => 'Resource consumer group/method for BATCH jobs');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'oltp', COMMENT => 'OLTP sessions', MGMT_P1 => 80,
  SWITCH_GROUP => 'batch', SWITCH_TIME => 3, UNDO_POOL => 200);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'batch', COMMENT => 'BATCH sessions', MGMT_P2 => 100,
  ACTIVE_SESS_POOL_P1 => 5, QUEUEING_P1 => 600, MAX_EST_EXEC_TIME => 3600);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'OTHER_GROUPS', COMMENT => 'mandatory', MGMT_P3 => 100);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
```

オラクル社が提供する複合ワークロード・プラン

Oracle Database には、事前定義のリソース・プラン MIXED_WORKLOAD_PLAN が用意されています。このプランでは、バッチ操作よりも対話型操作が優先されます。必要なサブプランと推奨するコンシューマ・グループも含まれています。MIXED_WORKLOAD_PLAN は、次のように定義されています。

	CPU リソース割当て				
グループまたはサブプラン	レベル 1	レベル 2	レベル 3	コンシューマ・グループの自動切替え	最大並列度
BATCH_GROUP			100%		
INTERACTIVE_GROUP		85%		切替え先グループ : BATCH_GROUP 切替え時間 : 60 秒 コールに対する切替え : TRUE	1
ORA$AUTOTASK_SUB_PLAN		5%			
ORA$DIAGNOSTICS		5%			
OTHER_GROUPS		5%			
SYS_GROUP	100%				

このプランの INTERACTIVE_GROUP は短いトランザクションを対象にしているため、60 秒を超えるコールは、長いバッチ操作を対象にした BATCH_GROUP に自動的に切り替わります。

この事前定義のプランは、自社の環境に対して適切な場合に使用できます。（このプランは変更可能です。使用する予定がない場合は削除できます。）BATCH_GROUP および INTERACTIVE_GROUP という名前に特別な意味はありません。これら名前には各グループに関する意図した目的のみが反映されています。対話型アプリケーションとバッチ・アプリケーションで適切なリソース管理を遂行できるように、アプリケーション・セッションをこれらのグループにマップして、CPU リソース割当て率を適切に調整する作業は管理者に任されます。たとえば、INTERACTIVE_GROUP コンシューマ・グループで対話型アプリケーションを確実に実行するには、25-25 ページの「コンシューマ・グループへのセッションのマッピング・ルールの指定」の説明に従って、ユーザー名、サービス名、プログラム名、モジュール名または処理に基づいて、その対話型アプリケーションのユーザー・セッションをこのコンシューマ・グループにマップする必要があります。同様の方法で、バッチ・アプリケーションを BATCH_GROUP にマップすることも必要です。最後に、25-30 ページの「Oracle Database Resource Manager の有効化とプランの切替え」の説明に従って、このプランを有効にしてください。

このプランの他のリソース・コンシューマ・グループとサブプランについては、25-43 ページの表 25-1 および 25-44 ページの表 25-2 を参照してください。

コンシューマ・グループ、プランおよびディレクティブのメンテナンス

ここでは、Oracle Database Resource Manager（リソース・マネージャ）のコンシューマ・グループ、リソース・プランおよびリソース・プラン・ディレクティブをメンテナンスする方法について説明します。メンテナンス・タスクは、DBMS_RESOURCE_MANAGER PL/SQL パッケージを使用して実行します。この項の内容は、次のとおりです。

- コンシューマ・グループの更新
- コンシューマ・グループの削除
- プランの更新
- プランの削除
- リソース・プラン・ディレクティブの更新
- リソース・プラン・ディレクティブの削除

関連項目：

- 25-45 ページ「DBMS_RESOURCE_MANAGER パッケージ・プロシージャの要約」
- DBMS_RESOURCE_MANAGER PL/SQL パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

コンシューマ・グループの更新

コンシューマ・グループ情報を更新するには、UPDATE_CONSUMER_GROUP プロシージャを使用します。最初にペンディング・エリアを作成し、発行する前に、コンシューマ・グループを更新する必要があります。引数を指定せずに UPDATE_CONSUMER_GROUP プロシージャを実行すると、データ・ディクショナリ内のコンシューマ・グループ情報は変更されません。

コンシューマ・グループの削除

DELETE_CONSUMER_GROUP プロシージャは、指定されたコンシューマ・グループを削除します。最初にペンディング・エリアを作成し、発行する前に、コンシューマ・グループを削除する必要があります。コンシューマ・グループを削除すると、そのグループを初期コンシューマ・グループとして割り当てられているすべてのユーザーには、初期コンシューマ・グループとして DEFAULT_CONSUMER_GROUP が割り当てられます。削除したコンシューマ・グループに属している現在実行中のセッションはすべて、コンシューマ・グループ・マッピング・ルールに基づいて新しいコンシューマ・グループに割り当てられます。マッピングでセッションのコンシューマ・グループが見つからなかった場合、そのセッションは DEFAULT_CONSUMER_GROUP に切り替わります。

リソース・プラン・ディレクティブが参照しているコンシューマ・グループは削除できません。

プランの更新

プラン情報を更新するには、UPDATE_PLAN プロシージャを使用します。最初にペンディング・エリアを作成し、発行する前に、プランを更新する必要があります。引数を指定せずに UPDATE_PLAN プロシージャを実行すると、データ・ディクショナリ内のプラン情報は変更されません。次のスクリプトは、COMMENT パラメータを更新する PL/SQL ブロックです。

```
BEGIN
DBMS_RESOURCE_MANAGER.UPDATE_PLAN(
   PLAN => 'DAYTIME',
   NEW_COMMENT => '50% more resources for OLTP applications');
END;
```

プランの削除

DELETE_PLAN プロシージャは、指定したプランと、それに対応付けられているすべてのプラン・ディレクティブを削除します。最初にペンディング・エリアを作成し、発行する前に、プランを削除する必要があります。

次のスクリプトは、great_bread プランとそのディレクティブを削除する PL/SQL ブロックです。

```
BEGIN
DBMS_RESOURCE_MANAGER.DELETE_PLAN(PLAN => 'great_bread');
END;
```

削除したディレクティブが参照していたリソース・コンシューマ・グループは削除されませんが、great_bread プランとの関連は解除されます。

DELETE_PLAN_CASCADE プロシージャは、指定のプランとその子孫すべて（プラン・ディレクティブ、および必須のマークが付けられていないサブプランとリソース・コンシューマ・グループ）を削除します。エラーが発生した DELETE_PLAN_CASCADE はロールバックされ、プランは変更されません。

現在アクティブなプランは削除できません。

リソース・プラン・ディレクティブの更新

プラン・ディレクティブを更新するには、UPDATE_PLAN_DIRECTIVE プロシージャを使用します。最初にペンディング・エリアを作成し、発行する前に、リソース・プラン・ディレクティブを更新する必要があります。引数を指定せずに UPDATE_PLAN_DIRECTIVE プロシージャを実行すると、データ・ディクショナリ内のプラン・ディレクティブは変更されません。

リソース・プラン・ディレクティブの削除

リソース・プラン・ディレクティブを削除するには、DELETE_PLAN_DIRECTIVE プロシージャを使用します。最初にペンディング・エリアを作成し、発行する前に、リソース・プラン・ディレクティブを削除する必要があります。

データベース・リソース・マネージャの構成とステータスの表示

Oracle Database Resource Manager（リソース・マネージャ）の現在の構成とステータスを表示するには、様々な静的データ・ディクショナリ・ビューと動的パフォーマンス・ビューを使用できます。ここでは、次の例を示します。

- ユーザーまたはロールに権限付与されたコンシューマ・グループの表示
- プラン情報の表示
- セッションの現行コンシューマ・グループの表示
- 現在アクティブなプランの表示

> **関連項目：** すべての静的データ・ディクショナリ・ビューと動的パフォーマンス・ビューの詳細は、『Oracle Database リファレンス』を参照してください。

ユーザーまたはロールに権限付与されたコンシューマ・グループの表示

DBA_RSRC_CONSUMER_GROUP_PRIVS ビューには、ユーザーまたはロールに付与されたコンシューマ・グループが表示されます。具体的には、ユーザーまたはロールが属しているグループや、切替え先のグループが表示されます。たとえば、次に示すビューでは、ユーザー SCOTT は常に SALES コンシューマ・グループで開始し、特定の付与を介して MARKETING グループに切り替えることができます。さらに、DEFAULT_CONSUMER_GROUP グループと LOW_GROUP グループにも切り替えることができます。これは、これらのグループに PUBLIC が付与されているためです。また、SCOTT には、他のユーザーに SALES グループを付与する機能がありますが、MARKETING グループを付与する機能はありません。

```
SQL> SELECT * FROM DBA_RSRC_CONSUMER_GROUP_PRIVS;

GRANTEE            GRANTED_GROUP                  GRANT_OPTION INITIAL_GROUP
------------------ ------------------------------ ------------ -------------
PUBLIC             DEFAULT_CONSUMER_GROUP         YES          YES
PUBLIC             LOW_GROUP                      NO           NO
SCOTT              MARKETING                      NO           NO
SCOTT              SALES                          YES          YES
SYSTEM             SYS_GROUP                      NO           YES
```

SCOTT には、DBMS_RESOURCE_MANAGER_PRIVS パッケージを使用してこれらのグループに切り替える機能がすでに付与されています。

プラン情報の表示

この例では、DBA_RSRC_PLANS ビューを使用して、データベース内に定義されているすべてのリソース・プランを表示しています。プランはすべて NULL ステータスです。これは、ペンディング・エリア内のプランでないことを意味します。

> **注意：** ペンディング・エリア内のプランのステータスは PENDING です。

```
SQL> SELECT PLAN,COMMENTS,STATUS FROM DBA_RSRC_PLANS;

PLAN                       COMMENTS                                          STATUS
-------------------------- ------------------------------------------------  ------
MIXED_WORKLOAD_PLAN        Example plan for a mixed workload that prio
ORA$AUTOTASK_SUB_PLAN      Default sub-plan for automated maintenance
ORA$AUTOTASK_HIGH_SUB_PLAN Default sub-plan for high-priority, automat
ERP_PLAN                   Resource plan/method for ERP Database
DEFAULT_PLAN               Default, basic, pre-defined plan that prior
INTERNAL_QUIESCE           Plan for quiescing the database.  This plan
INTERNAL_PLAN              Internally-used plan for disabling the reso
DEFAULT_MAINTENANCE_PLAN   Default plan for maintenance windows that p
```

セッションの現行コンシューマ・グループの表示

V$SESSION ビューを使用すれば、セッションに現在割り当てられているコンシューマ・グループを表示できます。

```
SQL> SELECT SID,SERIAL#,USERNAME,RESOURCE_CONSUMER_GROUP FROM V$SESSION;

SID    SERIAL#  USERNAME                  RESOURCE_CONSUMER_GROUP
-----  -------  ------------------------  --------------------------------
.
.
.
   11       136 SYS                       SYS_GROUP
   13     16570 SCOTT                     SALES
```

現在アクティブなプランの表示

この例では、mydb_plan（25-32 ページの「複数レベルのプランの例」の例で作成したプラン）をトップレベルのプランとして設定しています。次に、V$RSRC_PLAN ビューを問い合せて、現在アクティブなプランを表示します。このビューには、現行のトップレベルのプランとその子孫のサブプランすべてが表示されます。

```
SQL> ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = mydb_plan;

System altered.

SQL> SELECT NAME, IS_TOP_PLAN FROM V$RSRC_PLAN;

NAME             IS_TOP_PLAN
---------------------------
MYDB_PLAN        TRUE
MAILDB_PLAN      FALSE
BUGDB_PLAN       FALSE
```

Oracle Database Resource Manager の監視

次の動的パフォーマンス・ビューは、Oracle Database Resource Manager の設定結果の監視に役立ちます。

- V$RSRC_PLAN
- V$RSRC_CONSUMER_GROUP
- V$RSRC_SESSION_INFO
- V$RSRC_PLAN_HISTORY
- V$RSRC_CONS_GROUP_HISTORY

これらのビューには、次の情報が表示されます。

- 現在のステータス情報
- リソース・プランのアクティブ化の履歴
- リソース・コンシューマ・グループとセッションによるリソース使用と CPU 待ちに関する現在の統計と履歴の統計

さらに、履歴統計は、DBA_HIST_RSRC_PLAN および DBA_HIST_RSRC_CONSUMER_GROUP の各ビューを介して表示できます。これらのビューには、それぞれ V$RSRC_PLAN_HISTORY と V$RSRC_CONS_GROUP_HISTORY の自動ワークロード・リポジトリ（AWR）スナップショットが含まれています。

チューニングに役立つように、V$RSRCMGRMETRIC および V$RSRCMGRMETRIC_HISTORY の各ビューには、過去 1 時間に、CPU 待ちに費やした時間と CPU の使用量が、コンシューマ・グループごとに分単位で表示されます。Enterprise Manager では、これらのメトリックを「リソース・マネージャ統計」ページでグラフィカルに表示できます。

V$RSRC_PLAN このビューには、現在アクティブなリソース・プランとそのサブプランが表示されます。

```
SELECT name, is_top_plan FROM v$rsrc_plan;

NAME                           IS_TOP_PLAN
------------------------------ -----------
DEFAULT_PLAN                   TRUE
ORA$AUTOTASK_SUB_PLAN          FALSE
ORA$AUTOTASK_HIGH_SUB_PLAN     FALSE
```

IS_TOP_PLAN が TRUE のプランは、現在アクティブな（トップレベルの）プランです。他のプランは、トップレベルのプランのサブプランか、リスト内の他のサブプランのサブプランです。

V$RSRC_CONSUMER_GROUP CPU 使用と CPU 待ちを監視するには、V$RSRC_CONSUMER_GROUP ビューを使用します。このビューには、各コンシューマ・グループの全セッションについて、CPU 使用累積時間、CPU 待ち累積時間および CPU 待ち累積回数が表示されます。その他にも、チューニングに役立つ測定値が含まれています。

```
SQL> SELECT name, active_sessions, queue_length,
  consumed_cpu_time, cpu_waits, cpu_wait_time
  FROM v$rsrc_consumer_group;

NAME               ACTIVE_SESSIONS QUEUE_LENGTH CONSUMED_CPU_TIME  CPU_WAITS CPU_WAIT_TIME
------------------ --------------- ------------ ----------------- ---------- -------------
OLTP_ORDER_ENTRY                 1            0             29690        467          6709
OTHER_GROUPS                     0            0           5982366       4089         60425
SYS_GROUP                        1            0           2420704        914         19540
DSS_QUERIES                      4            2           4594660       3004         55700
```

前述の問合せ結果では、DSS_QUERIES コンシューマ・グループには、アクティブ・セッション・プールに 4 つのセッションがあり、2 つのセッションがキューでアクティブ化を待機しています。

このビューの主な測定値は CPU_WAIT_TIME です。この測定値は、リソース管理のために、コンシューマ・グループのセッションが CPU の割当てを待機していた合計時間を示します。この測定値には、ラッチまたはエンキューの競合、I/O 待機などに起因する待機は含まれません。

V$RSRC_SESSION_INFO 1 つ以上のセッションを監視するには、このビューを使用します。このビューには、リソース・マネージャによるセッションへの影響が表示されます。次のような情報が表示されます。

- セッションが現在属しているコンシューマ・グループ。
- セッションが当初属していたコンシューマ・グループ。
- コンシューマ・グループへのセッションのマップに使用されたセッション属性。
- セッションの状態（RUNNING、WAIT_FOR_CPU、QUEUED など）。
- メトリックに関する現在の統計と累積の統計（CPU 使用、待機時間、待ち行列時間など）。現在の統計には、セッションが現在のコンシューマ・グループに所属してからの統計が反映されています。累積の統計には、セッションが作成された以降に所属したすべてのコンシューマ・グループでの統計が反映されています。

```
SQL> SELECT se.sid sess_id, co.name consumer_group,
 se.state, se.consumed_cpu_time cpu_time, se.cpu_wait_time, se.queued_time
 FROM v$rsrc_session_info se, v$rsrc_consumer_group co
 WHERE se.current_consumer_group_id = co.id;

SESS_ID CONSUMER_GROUP     STATE     CPU_TIME CPU_WAIT_TIME QUEUED_TIME
------- ------------------ -------- --------- ------------- -----------
    113 OLTP_ORDER_ENTRY   WAITING     137947         28846           0
    135 OTHER_GROUPS       IDLE        785669         11126           0
    124 OTHER_GROUPS       WAITING      50401         14326           0
```

```
    114 SYS_GROUP          RUNNING        495             0           0
    102 SYS_GROUP          IDLE         88054            80           0
    147 DSS_QUERIES        WAITING     460910        512154           0
```

このビューの CPU_WAIT_TIME には、V$RSRC_CONSUMER_GROUP ビューと同様の意味がありますが、個々のセッションに適用されます。

V$RSRC_PLAN_HISTORY このビューには、インスタンスでリソース・プランが有効または無効になった日時が表示されます。各リソース・プランのアクティブ化または解除には連番が割り当てられます。このビューの各エントリについては、プランの各コンシューマ・グループに対応するエントリが V$RSRC_CONS_GROUP_HISTORY ビューにあり、コンシューマ・グループの累積統計が表示されます。この 2 つのビューは、それぞれの SEQUENCE# 列で結合しています。

```
SQL> SELECT sequence# seq, name plan_name,
to_char(start_time, 'DD-MON-YY HH24:MM') start_time,
to_char(end_time, 'DD-MON-YY HH24:MM') end_time, window_name
FROM v$rsrc_plan_history;

 SEQ PLAN_NAME                    START_TIME      END_TIME        WINDOW_NAME
---- ------------------------- --------------- --------------- ----------------
   1                           29-MAY-07 23:05 29-MAY-07 23:05
   2 DEFAULT_MAINTENANCE_PLAN  29-MAY-07 23:05 30-MAY-07 02:05 TUESDAY_WINDOW
   3                           30-MAY-07 02:05 30-MAY-07 22:05
   4 DEFAULT_MAINTENANCE_PLAN  30-MAY-07 22:05 31-MAY-07 02:05 WEDNESDAY_WINDOW
   5                           31-MAY-07 02:05 31-MAY-07 22:05
   6 DEFAULT_MAINTENANCE_PLAN  31-MAY-07 22:05                 THURSDAY_WINDOW
```

PLAN_NAME の下の NULL 値は、プランがアクティブにならなかったことを示します。

このビューの AWR スナップショットは DBA_HIST_RSRC_PLAN ビューに格納されます。

V$RSRC_CONS_GROUP_HISTORY このビューは、時間の経過に従ってコンシューマ・グループ内でリソースがどのように共有されたかを理解するのに役立ちます。sequence# 列は、V$RSRC_PLAN_HISTORY ビューの同名の列に対応しています。これによって、コンシューマ・グループ統計の各行について、アクティブであったプランを判別できます。

```
SQL> select sequence# seq, name, cpu_wait_time, cpu_waits,
consumed_cpu_time from V$RSRC_CONS_GROUP_HISTORY;

 SEQ NAME                     CPU_WAIT_TIME  CPU_WAITS CONSUMED_CPU_TIME
---- ------------------------ ------------- ---------- ------------------
   2 SYS_GROUP                        18133        691           33364431
   2 OTHER_GROUPS                     51252        825          181058333
   2 ORA$AUTOTASK_MEDIUM_GROUP           21          5            4019709
   2 ORA$AUTOTASK_URGENT_GROUP           35          1             198760
   2 ORA$AUTOTASK_STATS_GROUP             0          0                  0
   2 ORA$AUTOTASK_SPACE_GROUP             0          0                  0
   2 ORA$AUTOTASK_SQL_GROUP               0          0                  0
   2 ORA$AUTOTASK_HEALTH_GROUP            0          0                  0
   2 ORA$DIAGNOSTICS                      0          0            1072678
   4 SYS_GROUP                        40344         85           42519265
   4 OTHER_GROUPS                    123295       1040          371481422
   4 ORA$AUTOTASK_MEDIUM_GROUP            1          4            7433002
   4 ORA$AUTOTASK_URGENT_GROUP        22959        158           19964703
   4 ORA$AUTOTASK_STATS_GROUP             0          0                  0
     .
     .
   6 ORA$DIAGNOSTICS                      0          0                  0
```

このビューの AWR スナップショットは DBA_HIST_RSRC_CONSUMER_GROUP ビューに格納されます。コンシューマ・グループ統計の各履歴セットについて、アクティブであったプランを判別するには、DBA_HIST_RSRC_CONSUMER_GROUP と DBA_HIST_RSRC_PLAN を併用します。

関連項目：

- これらのビューおよびリソース・マネージャのその他のビューの詳細は、『Oracle Database リファレンス』を参照してください。
- AWR の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

オペレーティング・システムのリソース制御との相互作用

多くのオペレーティング・システムではリソース管理ツールを提供しています。これらのツールの名前には、「workload manager」や「resource manager」などの語句が含まれており、管理者が定義したポリシーを使用して、単一のサーバー・リソースを複数のアプリケーションで共有できることを意図しています。Oracle Database は、静的な構成を想定し、データベースの起動時に検出された使用可能なリソースから、ラッチなどの内部リソースを割り当てます。オペレーティング・システムのリソース構成を頻繁に変更すると、データベースが最適に実行されず、不安定になる場合があります。

オペレーティング・システムのリソース制御を使用するためのガイドライン

オペレーティング・システムによるリソース制御を Oracle Database と併用する場合は、次のガイドラインに従って慎重に使用する必要があります。

1. 通常、オペレーティング・システムのリソース制御は Oracle Database Resource Manager（リソース・マネージャ）と同時に使用しないでください。これは、いずれも互いの存在を認識しないためです。同時使用の結果、オペレーティング・システムとリソース・マネージャの両方がリソース割当てを制御しようとするため、予測できない動作が発生し、Oracle Database が不安定になります。
   - インスタンス内のリソース分配を制御する場合は、リソース・マネージャを使用し、オペレーティング・システムのリソース制御はオフにしてください。
   - ノード上に複数のインスタンスがあり、その間でリソースを分配する場合は、リソース・マネージャではなくオペレーティング・システムのリソース制御を使用してください。

   注意： 現在、Oracle Database では両方のツールの同時使用はサポートされていません。将来のリリースでは、ある程度の制限付きで相互作用が可能になる予定です。

2. オペレーティング・システムのリソース・マネージャ（HP 社の Process Resource Manager、Sun 社の Solaris Resource Manager など）を Oracle Database Resource Manager（リソース・マネージャ）と同時に使用する場合は、次の条件のすべてを満たす必要があります。
   - 各データベース・インスタンスが、オペレーティング・システムの専用のリソース・マネージャ・グループまたは管理エンティティに割り当てられている必要があります。
   - インスタンスのすべてのプロセスを実行している専用エンティティが、1 つの優先度（リソース使用）レベルで実行されている必要があります。
   - 専用エンティティに割り当てた CPU リソースは、数分に 1 回の割合を超える頻度では変更できません。
   - プロセスの優先度管理が使用禁止になっている必要があります。

注意： 個々のデータベース・プロセスを異なる優先度レベルで管理する機能（例：UNIX プラットフォームでの nice コマンドの使用）は、サポートされていません。インスタンスがクラッシュするなど、重大な結果になる可能性があります。オペレーティング・システムのリソース制御を使用して、Oracle Database インスタンスが固定されているメモリーを管理できる場合も、同様に望ましくない結果となることが予想されます。

3. オペレーティング・システムのリソース制御のみを使用する場合は、リソース・マネージャを必ずオフにしてください。手順については、25-31 ページの「リソース・マネージャの無効化」を参照してください。

Oracle Database Resource Manager の参照情報

ここでは、Oracle Database Resource Manager（リソース・マネージャ）の参照情報を提供します。

- 事前定義のリソース・プランおよびコンシューマ・グループ
- DBMS_RESOURCE_MANAGER パッケージ・プロシージャの要約
- リソース・マネージャのデータ・ディクショナリ・ビュー

事前定義のリソース・プランおよびコンシューマ・グループ

各 Oracle データベースに事前定義されているリソース・プランを表 25-1 にリストし、リソース・コンシューマ・グループを表 25-2 にリストします。

表 25-1 事前定義のリソース・プラン

リソース・プラン	説明
DEFAULT_MAINTENANCE_PLAN	メンテナンス・ウィンドウのデフォルト・プラン。このプランの詳細は、24-6 ページの「自動化メンテナンス・タスクに対するリソース割当ての概要」を参照してください。
DEFAULT_PLAN	SYS_GROUP 操作の優先度を設定し、自動化メンテナンス操作と診断操作に最小限のリソースを割り当てる基本のデフォルト・プラン。
INTERNAL_PLAN	リソース・マネージャの無効化用。内部処理でのみ使用されます。
INTERNAL_QUIESCE	データベースの静止用。このプランは直接アクティブ化できません。アクティブ化するには QUIESCE コマンドを使用します。
MIXED_WORKLOAD_PLAN	バッチ操作よりも対話型操作を優先させる複合ワークロード・プランの例。詳細は、25-35 ページの「オラクル社が提供する複合ワークロード・プラン」を参照してください。
ORA$AUTOTASK_HIGH_SUB_PLAN	優先度の高い自動化メンテナンス・タスクのデフォルト・サブプラン。このサブプランは、ORA$AUTOTASK_SUB_PLAN によって参照されます。このサブプランは直接参照しないでください。このプランは、DEFAULT_MAINTENANCE_PLAN のサブプランです。
ORA$AUTOTASK_SUB_PLAN	自動化メンテナンス・タスクのデフォルト・サブプラン。このサブプランに対するディレクティブは、自動化メンテナンス・タスクによって使用されるリソースを管理するために、すべてのトップレベルのプランに含まれている必要があります。このプランは、DEFAULT_MAINTENANCE_PLAN のサブプランです。

表 25-2 事前定義のリソース・コンシューマ・グループ

リソース・コンシューマ・グループ	説明
BATCH_GROUP	バッチ操作用のコンシューマ・グループ。
DEFAULT_CONSUMER_GROUP	SYS および SYSTEM 以外のユーザー・アカウントよって開始されるすべてのセッションの初期コンシューマ・グループ。この初期コンシューマ・グループは、コンシューマ・グループへのセッションのマッピング・ルールで上書きできます。DEFAULT_CONSUMER_GROUP には、リソース・プラン・ディレクティブで名前を指定できません。
INTERACTIVE_GROUP	対話型 OLTP 操作用のコンシューマ・グループ。
LOW_GROUP	優先度が低いセッション用のコンシューマ・グループ。
ORA$DIAGNOSTICS	クリティカル・エラーの発生時に診断ダンプを作成するデータベース・プロセスによって使用されるコンシューマ・グループ。
ORA$AUTOTASK_HEALTH_GROUP	今後使用する目的で確保されています。ORA$AUTOTASK_HIGH_SUB_PLAN に組み込まれています。
ORA$AUTOTASK_MEDIUM_GROUP	優先度が中程度のメンテナンス・タスク用のコンシューマ・グループ。
ORA$AUTOTASK_SPACE_GROUP	自動セグメント・アドバイザのメンテナンス・タスク用のコンシューマ・グループ。ORA$AUTOTASK_HIGH_SUB_PLAN に組み込まれています。
ORA$AUTOTASK_SQL_GROUP	自動 SQL チューニング・アドバイザのメンテナンス・タスク用のコンシューマ・グループ。ORA$AUTOTASK_HIGH_SUB_PLAN に組み込まれています。
ORA$AUTOTASK_STATS_GROUP	オプティマイザ統計収集のメンテナンス・タスク用のコンシューマ・グループ。ORA$AUTOTASK_HIGH_SUB_PLAN に組み込まれています。
ORA$AUTOTASK_URGENT_GROUP	緊急のメンテナンス・タスク用のコンシューマ・グループ。
OTHER_GROUPS	DEFAULT_CONSUMER_GROUP に属しているセッションを含め、現在アクティブなプランの一部でないコンシューマ・グループに属するすべてのセッションに選択的に適用されるコンシューマ・グループ。OTHER_GROUPS には、すべてのプランに指定されているリソース・プラン・ディレクティブを設定する必要があります。このグループは、マッピング・ルールを介してセッションに明示的に割り当てることはできません。
SYS_GROUP	システム管理者用のコンシューマ・グループ。これは、ユーザー・アカウント SYS または SYSTEM によって作成されたすべてのセッションの初期コンシューマ・グループです。この初期コンシューマ・グループは、コンシューマ・グループへのセッションのマッピング・ルールで上書きできます。

DBMS_RESOURCE_MANAGER パッケージ・プロシージャの要約

表 25-3 に、DBMS_RESOURCE_MANAGER パッケージのプロシージャの要約を示します。これらのプロシージャを実行するには、ADMINISTER_RESOURCE_MANAGER 権限が必要です。

表 25-3 DBMS_RESOURCE_MANAGER パッケージのプロシージャ

プロシージャ	説明
CREATE_SIMPLE_PLAN	最大 8 個のコンシューマ・グループを含む単純なリソース・プランを 1 ステップで作成します。これは、Oracle Database Resource Manager を初めて使用する際の最も迅速な方法です。
CREATE_PLAN	リソース・プランを作成し、その割当て方法を指定します。
UPDATE_PLAN	リソース・プランを更新します。
DELETE_PLAN	リソース・プランとそのディレクティブを削除します。
DELETE_PLAN_CASCADE	リソース・プランとそのすべての子を削除します。
CREATE_CONSUMER_GROUP	リソース・コンシューマ・グループを作成します。
UPDATE_CONSUMER_GROUP	コンシューマ・グループを更新します。
DELETE_CONSUMER_GROUP	コンシューマ・グループを削除します。
CREATE_PLAN_DIRECTIVE	プランのリソース・コンシューマ・グループまたはサブプランにリソースを割り当てるリソース・プラン・ディレクティブを指定します。
UPDATE_PLAN_DIRECTIVE	プラン・ディレクティブを更新します。
DELETE_PLAN_DIRECTIVE	プラン・ディレクティブを削除します。
CREATE_PENDING_AREA	プラン・スキーマを変更できるペンディング・エリア（スクラッチ領域）を作成します。
VALIDATE_PENDING_AREA	プラン・スキーマに対する保留中の変更の妥当性をチェックします。
CLEAR_PENDING_AREA	保留中のすべての変更をペンディング・エリアからクリアします。
SUBMIT_PENDING_AREA	プラン・スキーマに対してすべての変更を発行します。
SET_INITIAL_CONSUMER_GROUP	ユーザーの初期コンシューマ・グループを設定します。このプロシージャは非推奨です。ユーザーまたはセッションの初期コンシューマ・グループを指定する際は、SET_CONSUMER_GROUP_MAPPING プロシージャを使用することをお薦めします。
SWITCH_CONSUMER_GROUP_FOR_SESS	特定セッションのコンシューマ・グループを切り替えます。
SWITCH_CONSUMER_GROUP_FOR_USER	特定ユーザーに属しているすべてのセッションのコンシューマ・グループを切り替えます。
SWITCH_PLAN	現行のリソース・マネージャ・プランを設定します。
SET_CONSUMER_GROUP_MAPPING	セッションをコンシューマ・グループにマッピングします。
SET_CONSUMER_GROUP_MAPPING_PRI	セッション属性のマッピングの優先度を設定します。

関連項目： DBMS_RESOURCE_MANAGER PL/SQL パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

リソース・マネージャのデータ・ディクショナリ・ビュー

表 25-4 に、リソース・マネージャに関連付けられているビューをリストします。

表 25-4 リソース・マネージャのデータ・ディクショナリ・ビュー

ビュー	説明
DBA_RSRC_CONSUMER_GROUP_PRIVS USER_RSRC_CONSUMER_GROUP_PRIVS	DBA ビューには、すべてのリソース・コンシューマ・グループと、それが付与されているユーザーおよびロールがリストされます。USER ビューには、ユーザーに付与されたすべてのリソース・コンシューマ・グループがリストされます。
DBA_RSRC_CONSUMER_GROUPS	データベースに存在するすべてのリソース・コンシューマ・グループがリストされます。
DBA_RSRC_MANAGER_SYSTEM_PRIVS USER_RSRC_MANAGER_SYSTEM_PRIVS	DBA ビューには、リソース・マネージャのシステム権限が付与されているユーザーとロールがすべてリストされます。USER ビューには、DBMS_RESOURCE_MANAGER パッケージのシステム権限が付与されているユーザーがすべてリストされます。
DBA_RSRC_PLAN_DIRECTIVES	データベースに存在するすべてのリソース・プラン・ディレクティブがリストされます。
DBA_RSRC_PLANS	データベースに存在するすべてのリソース・プランがリストされます。
DBA_RSRC_GROUP_MAPPINGS	すべてのセッション属性に対する様々なマッピングのペアがすべてリストされます。
DBA_RSRC_MAPPING_PRIORITY	各属性の現在のマッピング優先度がリストされます。
DBA_HIST_RSRC_PLAN	リソース・プランのアクティブ化に関する履歴情報が表示されます。このビューには、V$RSRC_PLAN_HISTORY の AWR スナップショットが含まれています。
DBA_HIST_RSRC_CONSUMER_GROUP	コンシューマ・グループに関する履歴統計情報が表示されます。このビューには、V$RSRC_CONS_GROUP_HISTORY の AWR スナップショットが含まれています。
DBA_USERS USERS_USERS	DBA ビューには、データベース内のすべてのユーザーに関する情報が含まれます。各ユーザーの初期リソース・コンシューマ・グループが表示されます。USER ビューには、現行ユーザーに関する情報が表示されます。現行ユーザーの初期リソース・コンシューマ・グループが表示されます。
V$ACTIVE_SESS_POOL_MTH	使用可能なアクティブ・セッション・プールによるリソース割当て方法がすべて表示されます。
V$PARALLEL_DEGREE_LIMIT_MTH	使用可能な並列度制限によるリソース割当て方法がすべて表示されます。
V$QUEUEING_MTH	使用可能なキューイングによるリソース割当て方法がすべて表示されます。
V$RSRC_CONS_GROUP_HISTORY	V$RSRC_PLAN_HISTORY ビューの各エントリについて、プランの各コンシューマ・グループに対応するエントリが含まれており、コンシューマ・グループの累積統計が表示されます。
V$RSRC_CONSUMER_GROUP	アクティブなリソース・コンシューマ・グループに関する情報が表示されます。このビューは、チューニングに使用できます。
V$RSRC_CONSUMER_GROUP_CPU_MTH	リソース・コンシューマ・グループで使用可能な CPU リソース割当て方法がすべて表示されます。
V$RSRCMGRMETRIC	過去 1 分間のリソース使用履歴と累積 CPU 待ち時間（リソース管理に起因するもの）がコンシューマ・グループごとに表示されます。
V$RSRCMGRMETRIC_HISTORY	過去 1 時間のリソース使用履歴と累積 CPU 待ち時間（リソース管理に起因するもの）がコンシューマ・グループごとに分単位で表示されます。新しいリソース・プランが有効な場合、履歴はクリアされます。
V$RSRC_PLAN	現在のアクティブ・リソース・プランの名前がすべて表示されます。
V$RSRC_PLAN_CPU_MTH	リソース・プランで使用可能な CPU リソース割当て方法がすべて表示されます。

表 25-4 リソース・マネージャのデータ・ディクショナリ・ビュー（続き）

ビュー	説明
V$RSRC_PLAN_HISTORY	インスタンスでリソース・マネージャのプランが有効または無効になった日時が表示されます。このビューは、時間の経過に従ってコンシューマ・グループ内でリソースがどのように共有されたかを理解するのに役立ちます。
V$RSRC_SESSION_INFO	各セッションについてリソース・マネージャの統計が表示されます。リソース・マネージャによるセッションへの影響が表示されます。チューニングに使用できます。
V$SESSION	現行セッションごとにセッション情報が表示されます。その中には、各現行セッションのリソース・コンシューマ・グループの名前が含まれます。

関連項目： これらの各ビューの内容の詳細は、『Oracle Database リファレンス』を参照してください。

26

Oracle Scheduler の概要

この章の内容は次のとおりです。

Oracle Scheduler の概要

数百または数千ものタスクのスケジューリングを簡素化するために、Oracle Database にはエンタープライズ・ジョブ・スケジューラである Oracle Scheduler が組み込まれています。Oracle Scheduler（スケジューラ）は、DBMS_SCHEDULER PL/SQL パッケージのプロシージャおよびファンクションにより実装されます。

スケジューラを使用すると、企業環境において様々なコンピューティング・タスクをいつどこで実行するかを制御できます。また、これらのタスクの効率的な管理および計画に役立ちます。多数の日常的なコンピューティング・タスクが手動で操作することなく確実に実行されるため、操作コストの削減、信頼性の高いルーチンの実現、人為的なエラーの最小化および必要期間の短縮が可能です。

この項の内容は次のとおりです。

スケジューラの機能

スケジューラでは、洗練された柔軟なエンタープライズ・スケジューリング機能が提供されます。次のことを実行できます。

- PL/SQL 無名ブロック、PL/SQL ストアド・プロシージャおよび Java ストアド・プロシージャの実行。
- アプリケーション、シェル・スクリプトおよびバッチ・ファイルなど、データベース外部の実行可能ファイル（**外部実行可能ファイル**）の実行。外部実行可能ファイルをローカル・システムまたはリモート・システムで実行できます。リモート・システムに Oracle Database がインストールされている必要はありません。必要となるのはスケジューラ・エージェントのみです。スケジューラ・エージェントは、Oracle Database でサポートされている全プラットフォームおよび他の一部のプラットフォームで使用できます。
- 次の方法によるジョブ実行のスケジュール。
  - 時間ベースのスケジューリング

    特定の日時に 1 回または繰り返し実行するようにジョブをスケジュールできます。「特定の休日を除き毎週月曜日と木曜日の午前 3 時」または「業務上の各四半期の最終水曜日」のように、複雑な繰返し間隔を定義できます。
  - イベントベースのスケジューリング

    スケジューラでは、システム・イベントまたはビジネス・イベントに対応してジョブを開始できます。アプリケーションは、イベントを検出するとスケジューラに通知します。送信された通知のタイプに応じて、スケジューラは特定のジョブを開始します。イベントベースのスケジューリングの例として、ファイルがシステムに着信した時点、在庫が事前に指定したレベルを下回った時点、またはトランザクションに失敗した時点でジョブを開始させる場合があります。
  - 依存性スケジューリング

    スケジューラでは、前の 1 つ以上のタスクの結果に基づいてタスクを実行できます。ブランチやネストしたチェーンを含んだ複雑な依存性チェーンを定義できます。

- ビジネス要件に基づくジョブの優先度付け。

  スケジューラを使用すると、競合するジョブ間のリソース割当てを制御でき、ビジネス・ニーズに基づいてジョブ処理を調整できます。これは次の方法で実現されます。

  - 共通の特性や動作を共有するジョブは、ジョブ・クラスと呼ばれるさらに大きいエンティティにグループ化できます。クラス間に優先度を付けるには、各クラスに割り当てられるリソースを制御します。この結果、重要なジョブが優先され、ジョブの完了に必要なリソースの確保が保証されます。たとえば、データ・ウェアハウスをロードする重要なプロジェクトがある場合は、複数のデータ・ウェアハウス・ジョブをすべて 1 つのクラスにまとめ、使用可能なリソースの割当て比率を高くすることによって、その他のジョブより優先的に実行できます。
  - スケジューラは、スケジュールに基づいて優先度付けを変更できる機能を提供することによって、ジョブ間でのさらに詳細な優先度付けを行います。重要なジョブの定義は時間の経過とともに変化する場合があるため、スケジューラでは、期間ごとにジョブ間での優先度を変更することもできます。たとえば、オフピーク時にはデータ・ウェアハウスのロードに使用される抽出、転送およびロード（ETL）ジョブを重要なジョブとみなし、ピーク時には重要なジョブとみなさない場合があります。ただし、業務上の四半期の終了時期に実行する必要があるジョブは、ETL ジョブよりも優先させることが必要となる場合があります。このような場合は、各ジョブ・クラスに割り当てるリソースを変更して、クラス間での優先度を変更できます。詳細は、27-29 ページの「ジョブ・クラスの作成」および 27-31 ページの「ウィンドウの作成」を参照してください。

- ジョブの管理と監視

  ジョブの作成から完了までの間には様々な状態があります。スケジューラのアクティビティはログに記録されるため、ジョブのステータスや最終の実行時間などの情報を簡単に追跡できます。この情報はビューに格納され、Enterprise Manager または SQL で簡単に問い合せることができます。このビューからジョブとその実行内容に関する有益な情報が得られるため、これを使用してジョブをより適切にスケジュールし管理できます。たとえば、DBA は特定のユーザーの失敗ジョブをすべて簡単に追跡できます。28-7 ページの「スケジューラの監視と管理」を参照してください。

- クラスタ化された環境におけるジョブの実行と管理

  クラスタは、同じタスクを実行するために連携して動作するデータベース・インスタンスのセットです。Oracle Real Application Clusters（RAC）は、アプリケーションを変更せずにスケーラビリティと信頼性を提供します。スケジューラは、このようなクラスタ化された環境でのジョブの実行を完全にサポートします。システムの負荷を均等にし、パフォーマンスを向上させるために、ジョブを実行するデータベース・サービスを指定することもできます。詳細は、26-18 ページの「Real Application Clusters 環境におけるスケジューラの使用」を参照してください。

スケジューラ・オブジェクト

スケジューラを使用するには、スケジューラ・オブジェクトを作成します。これは、ジョブ・スケジューリングの内容、時期および方法を定義するスキーマ・オブジェクトです。スケジューラ・オブジェクトにより、モジュール化された方法でタスクを管理できます。この方法のメリットの 1 つは、既存のタスクに類似する新規タスクを作成する際にオブジェクトを再利用できることです。

すべてのスケジューラ・オブジェクトには属性があります。これらの属性には、オブジェクトの作成時または変更時に値を割り当てます。

スケジューラ・オブジェクトは次のとおりです。

- プログラム
- スケジュール
- ジョブ
- チェーン
- ジョブ・クラス
- ウィンドウ
- ウィンドウ・グループ

これらの各オブジェクトの詳細は、他の項を参照してください。重要なスケジューラ・オブジェクトはジョブです。ジョブでは、実行する処理とその実行に使用するスケジュールを定義します。スケジュールを定義するには、スタンドアロンで行う方法と他のスケジューラ・オブジェクトを参照する方法があります。

スケジューラ・オブジェクトはスキーマに属しているため、オブジェクト権限を付与できます。ジョブ・クラス、ウィンドウおよびウィンドウ・グループのような一部のスケジューラ・オブジェクトは、作成ユーザーがユーザー SYS でない場合も常に SYS スキーマに作成されます。他のオブジェクトはすべて、作成ユーザーのスキーマまたは指定のスキーマに作成されます。

関連項目：

- 28-31 ページの「スケジューラ権限」

プログラム

プログラム・オブジェクト（プログラム）では、スケジューラによって実行される内容が記述されます。プログラムの要素は次のとおりです。

- 処理 : ストアド・プロシージャ名、オペレーティング・システムのファイル・システムにある実行可能ファイル（外部実行可能ファイル）の名前、PL/SQL 無名ブロックのテキストなど
- タイプ : `'STORED_PROCEDURE'` または `'PLSQL_BLOCK'` など
- ストアド・プロシージャまたは外部実行可能ファイルが受け入れる引数の数

プログラムは、ジョブとは別個のエンティティです。ジョブは特定の時間または特定のイベントが発生した場合に実行され、特定のプログラムを起動します。ジョブは、既存のプログラム・オブジェクトを指し示すように作成できます。これは、様々なジョブで同じプログラムを使用でき、そのプログラムを様々な時間に様々な設定で実行できることを意味します。したがって、適切な権限があれば、様々なユーザーが同じプログラムを再定義せずに使用できます。このため、ユーザーが既存プログラムのリストから選択できるプログラム・ライブラリの作成が可能です。

プログラムで参照されるストアド・プロシージャまたは外部実行可能ファイルが引数を受け入れる場合は、これらの引数をプログラムの作成後に個別ステップで定義します。各引数のデフォルト値を定義することもできます。

プログラムの詳細は、27-20 ページの「プログラムの作成」を、ジョブの概要は、26-5 ページの「ジョブ」を参照してください。

スケジュール

スケジュール・オブジェクト（スケジュール）は、ジョブの実行時期と実行回数を指定します。スケジュールは複数のジョブで共有できます。たとえば、業務上の四半期の終了時期は、多くのジョブにとって共通の期間である可能性があります。ジョブの作成者は、新規のジョブを定義するたびに四半期の終了時期のスケジュールを定義するかわりに、名前付きのスケジュールを指し示すことができます。

スケジュールには、時間スケジュールとイベント・スケジュールの 2 種類があります。

時間スケジュールでは、ジョブを後で実行するか即時に実行するようにスケジュールできます。時間スケジュールには、開始日時、オプションの終了日時およびオプションの繰返し間隔が含まれています。

イベント・スケジュールでは、ファイルがシステムに着信した時点など、特定のイベントが発生した時点でジョブを実行するように指定できます。イベントの詳細は、27-41 ページの「イベントの使用」を参照してください。

詳細は、27-24 ページの「スケジュールの作成」を参照してください。

ジョブ

ジョブ・オブジェクト（ジョブ）は、1 回以上実行するようにスケジュールされるユーザー定義タスクを記述したメタデータの集合です。ジョブは、実行する必要がある内容（処理）と実行時期（スケジュール）の組合せです。

ジョブの処理は、次のいずれかの方法で指定します。

- 実行するデータベース・プログラム・ユニットまたは外部実行可能ファイルをジョブ属性として指定する方法。この方法をジョブの処理の**インライン**指定と呼びます。
- 既存のプログラム・オブジェクト（プログラム）名をジョブ属性として指定し、実行するデータベース・プログラム・ユニットまたは外部実行可能ファイルをプログラムで指定する方法。

  ジョブ所有者には、プログラムに対する EXECUTE 権限または EXECUTE ANY PROGRAM システム権限が必要です。

ジョブ・スケジュールは、次のいずれかの方法で指定します。

- ジョブ・オブジェクトの属性を設定して開始日時、終了日時および繰返し間隔を定義する方法。この方法をスケジュールの**インライン**指定と呼びます。
- 既存のスケジュール・オブジェクト（スケジュール）の名前をジョブ属性として指定し、開始日時、終了日時および繰返し間隔をスケジュールで定義する方法。

ジョブを作成して有効化した後は、スケジューラにより自動的にジョブがスケジュールに従って実行されます。ジョブの実行ステータスとジョブ・ログは、データ・ディクショナリ・ビューを問い合せることで確認できます。

この項の内容は次のとおりです。

- ジョブ・インスタンス
- ジョブ引数
- プログラム、ジョブおよびスケジュールの関連
- ジョブ・カテゴリ

関連項目：

- 27-3 ページの「ジョブの作成」
- 27-16 ページの「ジョブ・ログの表示」

ジョブ・インスタンス

ジョブ・インスタンスは、ジョブの特定の実行を表します。1 回のみ実行するようにスケジュールされたジョブのインスタンスは 1 つのみです。繰返しスケジュールされたジョブには複数のインスタンスがあり、ジョブの各実行が 1 つのインスタンスを表します。たとえば、2002 年 10 月 8 日、火曜日に実行されるようにスケジュールされたジョブのインスタンスは 1 つです。1 週間毎日正午に実行されるジョブには 7 つのインスタンスがあり、ジョブの実行ごとに 1 つのインスタンスが対応します。

ジョブの作成時に、ジョブを表すためにスケジューラのジョブ表に追加されるエントリは 1 つのみです。ロギング・レベルの設定に応じて、ジョブが実行されるたびにエントリが 1 つジョブ・ログに追加されます。したがって、繰返しスケジュールのジョブを作成した場合、ジョブのビューには 1 つのエントリが、ジョブ・ログには複数のエントリが存在します。各ジョブ・インスタンスのログ・エントリでは、特定の実行に関して、ジョブの完了ステータス、開始時間と終了時間などの情報が提供されます。ジョブの各実行には一意のログ ID が割り当てられます。この ID は、ジョブ・ログのビューとジョブ実行の詳細ビューの両方に使用されます。

詳細は、28-32 ページの「スケジューラのデータ・ディクショナリ・ビュー」を参照してください。

ジョブ引数

ジョブでプログラム・オブジェクト（プログラム）を参照する場合、ジョブ引数を指定してデフォルトのプログラム引数値を上書きするか、デフォルト値のないプログラム引数の値を提供できます。また、ジョブが指定するインライン処理（例 : ストアド・プロシージャ）に対しても引数値を提供できます。

必要なプログラム引数値のすべてが、参照先のプログラム・オブジェクトにデフォルトとして定義されているか、またはジョブ引数として定義されるまで、ジョブは使用できません。

ジョブの一般的な例には、一連のレポートを夜間に実行するジョブがあります。様々な部門で様々なレポートが必要な場合は、このタスクのプログラムを、様々な部門の様々なユーザー間で共有できるように作成できます。このプログラム処理では、レポート・スクリプトが実行され、プログラムには 1 つの引数として、部門番号を設定します。各ユーザーはこのプログラムを指し示すジョブを作成し、部門番号をジョブ引数として指定できます。

関連項目 :

- 27-5 ページ「ジョブ引数の設定」
- 27-20 ページ「プログラム引数の定義」
- 27-3 ページ「ジョブの作成」

プログラム、ジョブおよびスケジュールの関連

実行内容と実行時期を定義するには、プログラム、ジョブおよびスケジュール間に関連を割り当てます。図 26-1 に、これらの関連の例を示します。

図 26-1 プログラム、ジョブおよびスケジュール間の関連

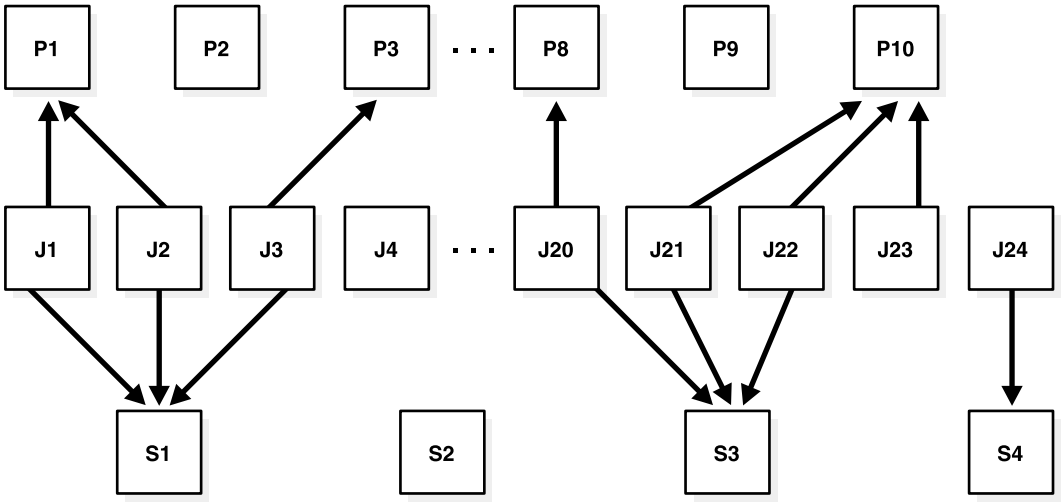


図 26-1 を理解するために、表を分析する場合を考えてみます。この例では、P1 が、DBMS_STATS パッケージを使用して表を分析するプログラムです。このプログラムには、表名に対する入力パラメータがあります。2 つのジョブ J1 と J2 は、両方とも同じプログラムを指し示していますが、別々の表名が指定されています。さらに、スケジュール S1 には、毎日午前 2 時の実行時間を指定できます。最終的な結果として、J1 と J2 で名前が指定された 2 つの表は、毎日午前 2 時に分析されます。

J4 は、すべての関連情報がそのジョブ自体に定義された自己完結のジョブで、他のエンティティを指し示していないことに注意してください。P2、P9 および S2 は、必要に応じて、プログラムまたはスケジュールに関連を割り当てないままにできることを示しています。たとえば、年度末の在庫を計算するプログラムを作成し、一時的に、そのプログラムをどのジョブにも割り当てないままにできます。

ジョブ・カテゴリ

スケジューラは、次のタイプのジョブをサポートしています。

- データベース・ジョブ
- チェーン・ジョブ
- 外部ジョブ
- デタッチ済ジョブ
- 軽量ジョブ

データベース・ジョブ

データベース・ジョブでは、PL/SQL 無名ブロック、PL/SQL ストアド・プロシージャおよび Java ストアド・プロシージャなどの Oracle Database プログラム・ユニットが実行されます。処理がインラインで指定されているデータベース・ジョブの場合、job_type は 'PLSQL_BLOCK' または 'STORED_PROCEDURE' に設定され、job_action には PL/SQL 無名ブロックのテキストまたはストアド・プロシージャ名が含まれています。（処理がインライン指定されるかわりにプログラム・オブジェクト名が指定されている場合は、それに応じて対応する program_type および program_action が設定されます。）

チェーン・ジョブ

チェーンは、依存性ベースのスケジューリングを可能にするスケジューラ・メカニズムです。最も単純な形式では、プログラム・オブジェクトのグループとプログラム・オブジェクト間の依存性が定義されます。ジョブは、単一のプログラム・オブジェクトを指し示すかわりにチェーンを指し示すことができます。その場合、ジョブはチェーンを開始する役割を持ちます。チェーン・ジョブの場合、job_type は 'CHAIN' として指定されます。

詳細は、26-12 ページの「チェーン」を参照してください。

外部ジョブ

外部ジョブでは、外部実行可能ファイルが実行されます。**外部実行可能ファイル**は、オペレーティング・システムの実行可能ファイルで、データベースの外部で実行されます。外部ジョブの場合、job_type は 'EXECUTABLE' として指定されます （名前付きプログラムを使用している場合は、対応する program_type が 'EXECUTABLE' になります）。job_action（名前付きプログラムを使用している場合は対応する program_action）には、必要な外部実行可能ファイルのパス（コマンドライン引数を除く）が、オペレーティング・システムに依存したフルパスの形式で入ります。たとえば、/usr/local/bin/perl や C:¥perl¥bin¥perl のようになります。タイプ 'EXECUTABLE' のプログラムまたはジョブの引数は、CHAR または VARCHAR2 などの文字列型であることが必要です。

Windows のバッチ・ファイルは直接実行できないため、cmd.exe を使用して実行する必要があることに注意してください。

スケジューラのジョブと同様に、ジョブの作成時にはスキーマを割り当てることができます。割り当てたスキーマはジョブの所有者になります。外部ジョブは SYS スキーマに作成できますが、この方法はお薦めしません。

外部ジョブを実行するスキーマには、CREATE JOB と CREATE EXTERNAL JOB の両方の権限が必要です。

外部実行可能ファイルは、オペレーティング・システム・ユーザーとして実行する必要があります。したがって、スケジューラでは、作成する外部ジョブにオペレーティング・システムの資格証明を割り当てることができます。そのためには、Oracle Database 11*g* リリース 1 で導入された資格証明と呼ばれるデータベース・オブジェクトを使用します。

外部ジョブには、ローカル外部ジョブとリモート外部ジョブの 2 つのタイプがあります。**ローカル外部ジョブ**では、そのジョブがスケジュールされているデータベースと同じコンピュータ上で外部実行可能ファイルが実行されます。**リモート外部ジョブ**では、リモート・ホスト（つまり、そのジョブがスケジュールされているデータベースを実行しているコンピュータとは別のホスト・コンピュータ）で実行可能ファイルが実行されます。リモート・ホストでは Oracle データベースは必要ありません。

次の各項では、資格証明、ローカル外部ジョブおよびリモート外部ジョブについて説明します。

- 資格証明の概要
- ローカル外部ジョブの概要
- リモート外部ジョブの概要

関連項目： 27-6 ページ「外部ジョブの作成」

資格証明の概要 **資格証明**は、専用のデータベース・オブジェクトに格納されているホスト・ユーザー名とパスワードのペアです。外部ジョブの資格証明を指定するには、そのジョブの credential_name 属性を設定します。これにより、ジョブの外部実行可能ファイルは、その資格証明に指定されているユーザー名とパスワードで実行されます。

資格証明を作成するには、DBMS_SCHEDULER.CREATE_CREDENTIAL プロシージャを使用します。自分のスキーマに資格証明を作成するには CREATE JOB 権限が、SYS 以外のスキーマに資格証明を作成するには CREATE ANY JOB 権限が必要です。資格証明を使用できるのは、資格証明の EXECUTE 権限がジョブの所有者に付与されているジョブ、またはジョブの所有者が資格証明の所有者を兼ねているジョブのみです。資格証明は他のスキーマ・オブジェクトのようにスキーマに属しているため、資格証明に対する権限は GRANT SQL 文を使用して付与します。

```
BEGIN
 DBMS_SCHEDULER.CREATE_CREDENTIAL('HRCREDENTIAL', 'hruser', 'hr001515');
END;

GRANT EXECUTE ON HRCREDENTIAL to HR;
```

データベース内の資格証明のリストを表示するには、*_SCHEDULER_CREDENTIALS ビューを問い合せます。資格証明パスワードは不明瞭な状態で格納されるため、*_SCHEDULER_CREDENTIALS ビューには表示されません。

ローカル外部ジョブの概要 ローカル外部ジョブでは、そのジョブがスケジュールされている Oracle データベースと同じコンピュータ上で外部実行可能ファイルが実行されます。このようなジョブの場合、destination ジョブ属性は NULL か、または localhost の値が格納されます。

ローカル外部ジョブでは、stdout および stderr の出力がディレクトリ *ORACLE_HOME*/scheduler/log 内のログ・ファイルに書き込まれます。これらのファイルの内容は、DBMS_SCHEDULER.GET_FILE で取得できます。

ローカル外部ジョブには資格証明を割り当てる必要はありません。ただし、セキュリティ向上のために割り当てることをお薦めします。資格証明を割り当てなかった場合、ジョブはデフォルト資格証明を使用して実行されます。表 26-1 に、各プラットフォームおよびジョブ所有者のデフォルトの資格証明を示します。

表 26-1 ローカル外部ジョブに対するデフォルトの資格証明

SYS スキーマのジョブか	プラットフォーム	デフォルトの資格証明
はい	すべて	Oracle Database をインストールしたユーザー
いいえ	UNIX と Linux	ファイル *ORACLE_HOME*/rdbms/admin/externaljob.ora に指定されている run-user 属性および run-group 属性の値
いいえ	Windows	OracleJobScheduler*SID* Windows サービスの実行ユーザー（ローカル・システム・アカウント、指名ローカル・ユーザーまたはドメイン・ユーザー） **注意：** このサービスは手動で有効化して開始する必要があります。セキュリティ向上のため、ローカル・システム・アカウントのかわりに指名ユーザーを使用することをお薦めします。

注意： デフォルトの資格証明は前のリリースの Oracle Database との互換性を保つために用意されており、将来のリリースで非推奨になる可能性があります。したがって、1 つの資格証明をすべてのローカル外部ジョブに割り当てることが最善の方法です。

資格証明が割り当てられていないローカル外部ジョブの実行を禁止するには、*ORACLE_HOME*/rdbms/admin/externaljob.ora ファイルから run_user 属性を削除する（UNIX と Linux の場合）か、OracleJobScheduler サービスを停止します（Windows の場合）。SYS スキーマ内のローカル外部ジョブは、これらのステップでは実行不可になりません。

関連項目：

- ローカル外部ジョブをサポートするためのインストール後の構成ステップについては、使用しているオペレーティング・システム固有のマニュアルを参照してください。
- 27-9 ページ例 27-6「ローカル外部ジョブの作成と stdout の取得」

リモート外部ジョブの概要 リモート外部ジョブでは、そのジョブがスケジュールされているOracleデータベースを実行しているコンピュータとは別のホスト・コンピュータで外部実行可能ファイルが実行されます。説明上、ここでは、リモート実行可能ファイルが実行されるコンピュータを**リモート・ホスト**と呼びます。リモート・ホストには、Oracle Databaseがインストールされている場合も、されていない場合もあります。ただし、いずれの場合も、リモート・ホストには、データベースがリモート・ホストで外部実行可能ファイルを起動する際に通信するスケジューラ・エージェントがあります。このエージェントは、実行結果をデータベースに戻す処理にも関係しています。エージェントは、個別にインストールされるリモート・ホスト上の実行可能ファイルです。エージェントは、着信ジョブ・リクエストをネットワーク・ポートでリスニングして実行します。

リモート外部ジョブを作成する場合は、そのジョブの`destination`属性としてリモート・ホストとポート番号を指定します。また、リモート外部ジョブの資格証明も指定する必要があります。

リモート外部ジョブでは、stdoutおよびstderrの出力がディレクトリ*AGENT_HOME*/data/log内のログ・ファイルに書き込まれます。これらのファイルの内容は、`DBMS_SCHEDULER.GET_FILE`で取得できます。stdoutの出力の取得方法は、27-9ページの例27-6「ローカル外部ジョブの作成とstdoutの取得」を参照してください。この例はローカル外部ジョブに関するものですが、メソッドはリモート外部ジョブの場合も同じです。

関連項目：

- 28-12ページ「リモート外部ジョブの有効化と無効化」

デタッチ済ジョブ

デタッチ済ジョブを使用して、スケジューラから独立して非同期的に個別プロセスで実行されるスクリプトまたはアプリケーションを起動します。通常、デタッチ済ジョブは別のプロセスを起動してから終了します。デタッチ済ジョブは、終了時（ジョブの処理の完了時）も実行中の状態のままです。この状態は、ジョブにより起動された非同期プロセスがまだアクティブであることを示します。非同期プロセスは、処理が終了した時点でデータベースに接続し、`DBMS_SCHEDULER.END_DETACHED_JOB_RUN`をコールする必要があります。これによりジョブが終了します。

ジョブは、`detached`属性が`TRUE`に設定されているプログラム・オブジェクト（プログラム）（**デタッチ済プログラム**）を指している場合、デタッチ済です。

デタッチ済ジョブは、次の2つの場合に使用します。

- 必要以上にリソースを保持するため、起動した非同期プロセスが完了するまで待機することが実際的でない場合

  たとえば、非同期Webサービスにリクエストを送信する場合です。Webサービスが応答するまでに数時間から数日かかる可能性があり、応答を待機する間スケジューラ・ジョブ・スレーブを保持することは望ましくありません。（ジョブ・スレーブの詳細は、26-16ページの「スケジューラのアーキテクチャ」を参照してください。）

- プロセスによりデータベースが停止されるため、起動した非同期プロセスが完了するまで待機できない場合

  たとえば、スケジューラ・ジョブを使用して、データベースの停止、コールド・バックアップの作成およびデータベースの再起動を実行するRMANスクリプトを起動する場合です。27-10ページの例 27-7を参照してください。

デタッチ済ジョブの動作は次のとおりです。

1. ジョブの開始時に、ジョブ・コーディネータによりジョブにジョブ・スレーブが割り当てられ、デタッチ済プログラムに定義されているプログラム処理がジョブ・スレーブにより実行されます。プログラム処理は、PL/SQLブロック、ストアド・プロシージャまたは外部実行可能ファイルのいずれかです。
2. プログラム処理は、別のスクリプトまたは実行可能ファイル（この例ではプロセスA）の即時リターン・コールを実行してから終了します。プログラム処理の動作は完了しているため、ジョブ・スレーブは終了しますが、ジョブは実行中の状態のまま残ります。

3. プロセス A により処理が実行されます。データベースに対する DML を実行する場合は、その処理をコミットする必要があります。処理が完了すると、プロセス A によりデータベースにログが記録され、END_DETACHED_JOB_RUN がコールされます。
4. デタッチ済ジョブが完了として記録されます。

実行中のデタッチ済ジョブを終了するには、STOP_JOB をコールする方法もあります。

> **関連項目：** デタッチ済ジョブを使用してデータベースのコールド・バックアップを実行する例は、27-10 ページの「デタッチ済ジョブの作成」を参照してください。

軽量ジョブ

頻繁に実行する短時間のジョブが多数ある場合は、軽量ジョブを使用します。特定の状況では、軽量ジョブを使用することで、わずかながらパフォーマンスが向上する場合があります。

軽量ジョブには、次の特性があります。

- 標準ジョブとは異なり、スキーマ・オブジェクトではありません。
- スキーマ・オブジェクトを作成する際のオーバーヘッドを伴わないため、作成および削除に必要な時間が標準ジョブよりも大幅に短縮されます。
- セッションの平均作成時間が標準ジョブよりも短縮されます。
- ジョブのメタデータと実行時データについては、ディスク上に小さいフットプリントが保持されます。

軽量ジョブを指定するには、job_style ジョブ属性を 'LIGHTWEIGHT' に設定します。もう 1 つのジョブ・スタイルは、デフォルトの 'REGULAR' です。

プログラムやスケジュールと同様に、標準ジョブはスキーマ・オブジェクトです。Oracle Database 11*g* リリース 1 より前のリリースでは、標準ジョブがスケジューラでサポートされる唯一のジョブ・スタイルでした。

標準ジョブは最大の柔軟性を提供しますが、作成または削除する際に多少のオーバーヘッドを伴います。ユーザーは、ジョブの権限をきめ細かく制御できます。ジョブには、別のユーザーが所有するプログラムやストアド・プロシージャを処理として指定できます。

実行頻度が低い比較的少数のジョブを作成する必要がある場合は、軽量ジョブよりも標準ジョブが優先されます。

軽量ジョブでは、プログラム・オブジェクト（プログラム）を参照してジョブの処理を指定する必要があります。プログラムは軽量ジョブの作成時に有効化されている必要があり、プログラム・タイプは 'PLSQL_BLOCK' または 'STORED_PROCEDURE' であることが必要です。軽量ジョブはスキーマ・オブジェクトではないため、権限は付与できません。軽量ジョブは指定のプログラムから権限を継承します。したがって、プログラムに対して特定の権限セットを持つユーザーは、軽量ジョブに対して対応する権限を持つことになります。

> **関連項目：** 軽量ジョブの作成例は、27-3 ページの「ジョブの作成」および 28-19 ページの「スケジューラの使用例」を参照してください。

チェーン

チェーンは、前の 1 つ以上のジョブの結果に応じて異なるジョブが開始される依存性スケジューリングを実装できる手段です。チェーンは、依存性ルールを使用して結合された複数のステップで構成されています。依存性ルールでは、タスクまたはチェーン自体の開始または停止に使用できる条件を定義します。条件には、前のタスクの成功、失敗、完了コードまたは終了コードを使用できます。AND/OR などの論理式を条件に使用することもできます。チェーンは、実行するタスクと、タスクの実行時期の選択に関して多くの可能なパスを持ち、ある意味で Decision Tree に似ています。

最も単純な形式のチェーンは、1 つの目的のために互いにリンクされた複数のスケジューラ・プログラム・オブジェクト（プログラム）で構成されています。チェーンの例には、「プログラム A を実行してからプログラム B を実行するが、プログラム A とプログラム B の両方が正常に完了した場合にのみプログラム C を実行し、正常に完了しない場合は 1 時間待機してからプログラム D を実行する」などがあります。

チェーンの作成が必要な典型的な状況として、融資申請を検証して承認した後に融資するなど、正常な取引のために様々なプログラムの結合を必要とする金融取引などがあります。

スケジューラ・ジョブは、単一のプログラム・オブジェクトを指し示すかわりにチェーンを指し示すことができます。その場合、ジョブはチェーンを開始する役割を持ちます。このジョブは**チェーン・ジョブ**と呼ばれます。複数のチェーン・ジョブで同じチェーンを指し示すことができ、そのうちの複数のジョブを同時に実行することで、チェーンの様々な進捗ポイントで同じチェーンの複数インスタンスをそれぞれ作成できます。

チェーン内の各位置は**ステップ**と呼ばれます。通常は、最初の一連のチェーン・ステップを開始し、後続のステップは、1 つ以上前のステップの完了に従って実行されます。各ステップは、次のいずれかを指し示します。

- プログラム・オブジェクト（プログラム）

  プログラムでは、データベース・プログラム・ユニット（ストアド・プロシージャまたは PL/SQL 無名ブロックなど）を実行するか、ローカルまたはリモートの外部実行可能ファイルを実行できます。リモート外部実行可能ファイルの場合は、宛先ホストを指定します。

- 別のチェーン（ネストしたチェーン）

  チェーンをネストするレベル数に制限はありません。

- イベント・スケジュールまたはインライン・イベント

  イベント・スケジュールを指し示すステップ、またはインライン・イベント指定を持つステップは、特定のイベントが呼び出されるまで待機します。イベントが発生すると、このステップは完了し、イベント・ステップに依存するステップが実行可能になります。チェーン内のイベントの一般例は、承認や拒否のようなユーザーの介入です。

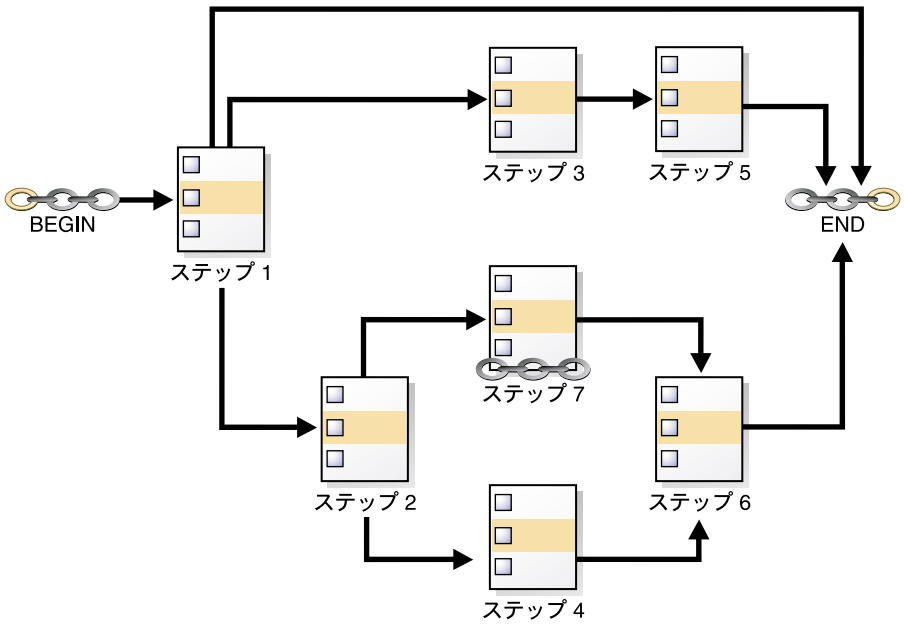
チェーン内の複数のステップは、同じプログラムまたはネストしたチェーンを起動できます。

26-13 ページの図 26-2 に、複数のブランチを持つチェーンを示します。このチェーンでは、ルールが次のように定義されているとします。

- ステップ 1 が正常に完了した場合は、ステップ 2 を開始します。
- ステップ 1 が失敗してエラー・コード 20100 が戻された場合は、ステップ 3 を開始します。
- ステップ 1 が失敗して他のエラー・コードが戻された場合は、チェーンを終了します。

ステップ 4、5、6 および 7 の実行は、他のルールにより制御されます。

図 26-2 複数のブランチを持つチェーン



チェーンを指し示しているジョブの実行中は、実行しているチェーンのすべてのステップについて現在の状態を監視できます。各ステップには、スケジューラによって、チェーン・ジョブと同じジョブ名と所有者を設定した**ステップ・ジョブ**が作成されます。各ステップ・ジョブにはさらに、そのジョブを一意に識別するためのサブ名があります。ステップ・ジョブ・サブ名は、ビュー *_SCHEDULER_RUNNING_JOBS、*_SCHEDULER_JOB_LOG および *_SCHEDULER_JOB_RUN_DETAILS に JOB_SUBNAME 列として、*_SCHEDULER_RUNNING_CHAINS ビューに STEP_JOB_SUBNAME 列として含まれています。

詳細は、27-49 ページの「チェーンの使用」を参照してください。

ジョブ・クラス

通常、ジョブ・クラスを作成するのは、スケジューラ管理者のロールを持っている場合のみです。

ジョブ・クラスは、次の内容を実行します。

- メンバーのジョブへの同じ属性値セットの割当て

  各ジョブ・クラスでは、ロギング・レベルなどの属性セットを指定します。ジョブをジョブ・クラスに割り当てると、そのジョブはこれらの属性を継承します。たとえば、全給与ジョブに対するログ・エントリのパージに関して、同じポリシーを指定できます。

- メンバーのジョブに対するサービス・アフィニティの設定

  ジョブ・クラスの service 属性を必要なデータベース・サービス名に設定できます。これによって、Real Application Clusters 環境のインスタンスが決まります。このインスタンスは、メンバーのジョブを実行し、必要に応じて、メンバーのジョブに割り当てられたシステム・リソースを実行します。詳細は、26-19 ページの「スケジューラ使用時のサービス・アフィニティ」を参照してください。

- メンバーのジョブに対するリソース割当ての設定

  各ジョブ・クラスでは、リソース・コンシューマ・グループを属性として指定できるため、ジョブ・クラスでは、データベース・リソース・マネージャとスケジューラ間のリンクが提供されます。このため、指定したコンシューマ・グループに属するメンバーのジョブには、現行のリソース・プランの設定に応じて、リソースが割り当てられます。

  また、resource_consumer_group 属性を NULL のままにし、ジョブ・クラスの service 属性を必要なデータベース・サービス名に設定できます。このサービスは、リ

ソース・コンシューマ・グループにマップできます。resource_consumer_group 属性と service 属性が設定されているときに、指定のサービスがリソース・コンシューマ・グループにマップされた場合は、resource_consumer_group 属性に指定されているリソース・コンシューマ・グループが優先されます。

コンシューマ・グループへのサービスのマッピングの詳細は、第 25 章「Oracle Database Resource Manager を使用したリソース割当ての管理」を参照してください。

- ジョブの優先度によるグループ化

  同じジョブ・クラス内では、個々のジョブに 1 ～ 5 の優先度値を割り当てることができます。このため、クラス内の 2 つのジョブが同時に開始するようにスケジュールされている場合は、優先度の高いジョブが優先されます。この機能によって、重要度の高いジョブが時間どおりに完了するのを重要度の低いジョブが妨げることはありません。

  2 つのジョブに同じ優先値が割り当てられている場合は、開始日の早いジョブが優先されます。ジョブに優先度が割り当てられていない場合、そのジョブの優先度は 3 にデフォルト設定されます。

注意： ジョブの優先度は、同じクラス内のジョブの間で優先度を付ける場合にのみ使用されます。

同じスケジュールを共有している場合でも、クラス A 内の優先度の高いジョブが、クラス B 内の優先度の低いジョブより先に開始されることは保証されません。異なるクラスのジョブ間の優先度付けは、現行のリソース・プランと、指定されたリソース・コンシューマ・グループまたは各ジョブ・クラスのサービス名に応じて異なります。

ジョブ・クラスの定義するときは、ジョブを機能別に分類してください。マーケティング、生産、販売、財務および人事など、同様のデータにアクセスするジョブをグループに分けることを考慮してください。

次の制限事項に注意してください。

- ジョブは必ず 1 つのクラスに属している必要があります。ジョブを作成するときは、そのジョブが属するクラスを指定できます。クラスを指定しない場合、ジョブは自動的に DEFAULT_JOB_CLASS クラスのメンバーになります。
- クラス内にジョブが存在している状態でクラスを削除するとエラーになります。クラスのメンバーであるジョブがまだ存在している場合でもクラスを強制的に削除することはできますが、そのクラスを参照しているジョブはすべて自動的に使用禁止になり、DEFAULT_JOB_CLASS クラスに割り当てられます。削除したクラスに属しているジョブの中ですでに実行中のジョブは、ジョブの開始時に判別されたクラスの設定のまま実行されます。

ウィンドウ

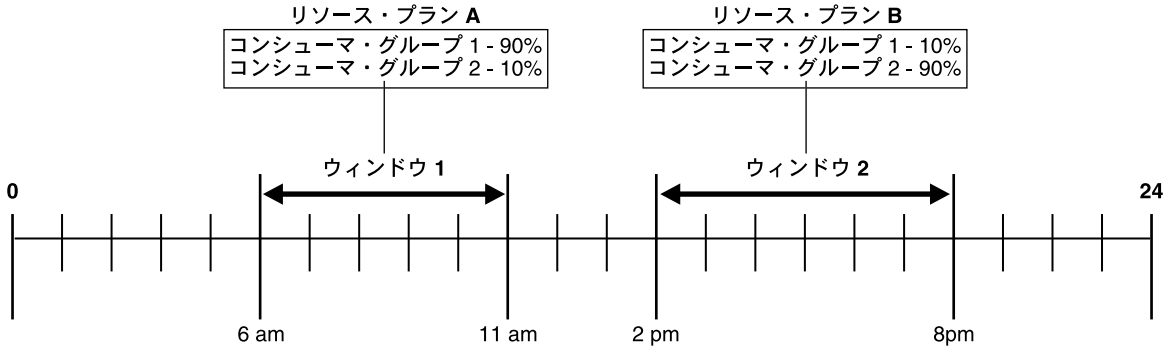
通常、ウィンドウを作成するのは、スケジューラ管理者のロールを持っている場合のみです。

日または週などの様々な期間中に、ジョブを自動的に開始したり、複数のジョブ間のリソース割当てを変更するには、ウィンドウを作成します。ウィンドウは、午前 12 時～午前 6 時など、開始と終了が明確に定義された期間で表されます。

ウィンドウは、ジョブ・クラスを使用して、リソース割当てを制御します。各ウィンドウは、ウィンドウが**オープン**したとき（アクティブになったとき）に、アクティブにするリソース・プランを指定し、各ジョブ・クラスは、リソース・コンシューマ・グループを指定するか、コンシューマ・グループにマップできるデータベース・サービスを指定します。したがって、ウィンドウ内で実行するジョブには、そのジョブ・クラスのコンシューマ・グループとウィンドウのリソース・プランに応じてリソースが割り当てられます。

図 26-3 に、2 つのウィンドウが設定された稼働日を示します。この構成では、コンシューマ・グループ 1 にリンクしているジョブ・クラスに属するジョブが、午後よりも午前にリソースを多く使用しています。コンシューマ・グループ 2 にリンクしているジョブ・クラスのジョブは、この逆です。

図 26-3 ジョブに割り当てるリソースの定義に役立つウィンドウ



リソース・プランおよびコンシューマ・グループの詳細は、第 25 章「Oracle Database Resource Manager を使用したリソース割当ての管理」を参照してください。

各ウィンドウには、優先度を割り当てることができます。ウィンドウが重複する場合は、優先度の高いウィンドウが優先度の低いウィンドウより優先して選択されます。スケジューラは、ウィンドウの開始時間と終了時間に従って、各ウィンドウを自動的にオープンしたり、クローズします。

ジョブは、その `schedule_name` 属性でウィンドウを指定できます。スケジューラは、このウィンドウがオープンするとジョブを開始します。ウィンドウがすでにオープンしている場合にそのウィンドウを指し示す新規ジョブが作成されると、そのジョブはウィンドウが次回オープンするまで開始されません。

ウィンドウの作成と使用については、27-31 ページの「ウィンドウの作成」を参照してください。

注意： 必要な場合は、現行のリソース・プランが切り替わらないように一時的にウィンドウをブロックできます。詳細は、25-30 ページの「Oracle Database Resource Manager の有効化とプランの切替え」、または『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` パッケージ・プロシージャに関する説明を参照してください。

ウィンドウ・グループ

通常、ウィンドウ・グループを作成するのは、スケジューラ管理者のロールを持っている場合のみです。

ジョブのスケジュールで、ウィンドウを使いやすいようにグループ化できます。ウィンドウ・グループを使用すると、1 日あるいは 1 週間などの間に複数期間実行するジョブを簡単にスケジュールできます。ジョブの `schedule_name` 属性をこのウィンドウ・グループの名前に設定すると、ウィンドウ・グループで指定したすべての期間で、このジョブを実行できます。

たとえば、「週末」という名前のウィンドウと「平日夜間」という名前のウィンドウがあり、この 2 つのウィンドウを「停止時間」という名前のウィンドウ・グループに追加するとします。データ・ウェアハウスの従業員は、この「停止時間」ウィンドウ・グループに従って、問合せを実行するジョブを作成できます。このウィンドウ・グループ（「平日夜間」と「週末」の時間帯）では、問合せに対して使用可能なリソースが高い比率で割り当てられる可能性があります。

ウィンドウ・グループのウィンドウがすでにオープンしているときに、そのウィンドウ・グループを指し示す新規ジョブが作成された場合、そのジョブはウィンドウ・グループの次のウィンドウがオープンするまで開始されません。

ウィンドウ・グループの作成例は、27-39 ページの「ウィンドウ・グループの作成」を参照してください。

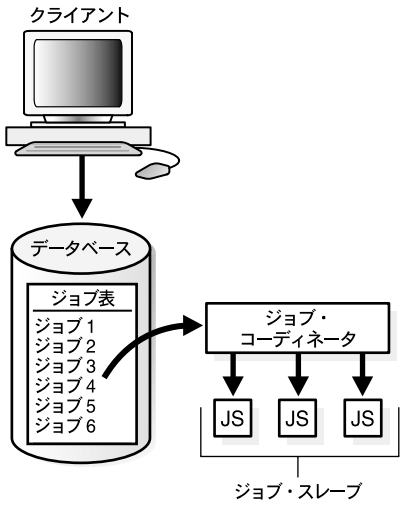
スケジューラのアーキテクチャ

この項では、スケジューラのアーキテクチャについて説明します。この項の内容は、次のとおりです。

- ジョブ表
- ジョブ・コーディネータ
- ジョブの実行方法
- ジョブ・スレーブ
- Real Application Clusters 環境におけるスケジューラの使用

図 26-4 に、データベースによるジョブの処理方法を示します。

図 26-4 スケジューラの構成要素



ジョブ表

ジョブ表は、すべてのジョブに対するコンテナです。データベースごとに 1 つの表があります。ジョブ表には、すべてのジョブに関して、ロギング・レベルや所有者名などの情報が格納されます。この情報は *_SCHEDULER_JOBS ビューで参照できます。

ジョブはデータベース・オブジェクトであるため、累積されて多くの領域を使用する場合があります。これを回避するために、ジョブ・オブジェクトは、ジョブの完了後、デフォルトで自動的に削除されます。この動作は、auto_drop ジョブ属性によって制御されます。

使用可能なジョブのビューと管理については、28-32 ページの「スケジューラのデータ・ディクショナリ・ビュー」を参照してください。

ジョブ・コーディネータ

ジョブ・コーディネータ・バックグラウンド・プロセス（cjqNNN）は、必要になると自動的に起動し、停止します。データベースの起動時に、ジョブ・コーディネータは起動されません。しかし、近いうちに実行されるジョブまたはオープンされるウィンドウがあるかどうかは、データベースが監視しています。コーディネータは、このようなジョブやウィンドウがある場合に起動されます。

実行中のジョブまたはウィンドウがある間は、コーディネータは継続的に実行されます。スケジューラが一定期間非アクティブな状態にあり、近い将来にスケジュールされたジョブまたはウィンドウがない場合、コーディネータは自動的に停止します。

ジョブ・コーディネータを起動するかどうかをデータベースが判断する際は、ジョブのサービス・アフィニティが考慮されます。たとえば、近い将来スケジュールされるジョブが 1 つのみで、このジョブが属するジョブ・クラスで、4 つの RAC インスタンスのうち 2 つのみにサービス・アフィニティがある場合は、この 2 つのインスタンスのジョブ・コーディネータのみが起動されます。詳細は、26-19 ページの「スケジューラ使用時のサービス・アフィニティ」を参照してください。

ジョブ・コーディネータの機能は次のとおりです。

- ジョブ・スレーブを制御および起動します。
- ジョブ表を問い合せます。
- ジョブ表からジョブを定期的に取り出し、メモリー・キャッシュに格納します。この結果、ディスクに格納されないため、パフォーマンスが向上します。
- メモリー・キャッシュからジョブを取り出し、実行するためにジョブ・スレーブに渡します。
- スレーブが不要になると、ジョブ・スレーブ・プールをクリーン・アップします。
- スケジュールされているジョブがないときは休止状態になります。
- 新規ジョブが実行されるとき、または CREATE_JOB プロシージャを使用してジョブが作成されたときに起動します。
- データベースの異常停止後の起動時に、実行していたジョブをリカバリします。

ジョブ・コーディネータによるジョブ表のチェックの時期は、設定する必要はありません。期間はシステムによって自動的に選択されます。起動するジョブ・スレーブ数は、CPU の負荷と未処理のジョブ数に基づいて、コーディネータで自動的に決定されます。特別な場合は、データベース管理者が DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE プロシージャに MAX_JOB_SLAVE_PROCESSES パラメータを設定することで、起動するスレーブの最大数を制限できます。

インスタンスごとに 1 つのジョブ・コーディネータが使用されます。これは、RAC 環境の場合も同じです。

> **関連項目：** ジョブ・コーディネータの管理については、28-32 ページの「スケジューラのデータ・ディクショナリ・ビュー」を参照してください。
> RAC の情報については、26-18 ページの「Real Application Clusters 環境におけるスケジューラの使用」を参照してください。

ジョブの実行方法

処理のためにジョブが取り出されると、ジョブ・スレーブは次のことを実行します。

1. ジョブの実行に必要なすべてのメタデータを収集します。例として、プログラムの引数および権限情報があります。
2. ジョブの所有者としてデータベース・セッションを開始し、トランザクションを開始した後に、ジョブの実行を開始します。
3. ジョブが完了すると、スレーブはトランザクションをコミットし、終了します。
4. セッションをクローズします。

ジョブ・スレーブ

ジョブ・スレーブは、発行されたジョブを実際に実行します。ジョブの実行時期になるとジョブ・コーディネータによって起動されます。ジョブ・スレーブは、ジョブ表からジョブを実行するためのメタデータを収集します。

ジョブが終了すると、スレーブは次のことを実行します。

- 必要に応じて、ジョブを再スケジュールします。
- ジョブ表の状態を更新して、ジョブを完了するか、再スケジュールするかを反映します。
- エントリをジョブ・ログ表に挿入します。
- 実行回数を更新し、必要に応じて、失敗の回数と再試行の回数を更新します。
- クリーン・アップします。
- 新規作業を検索します（検出されない場合は休止状態になります）。

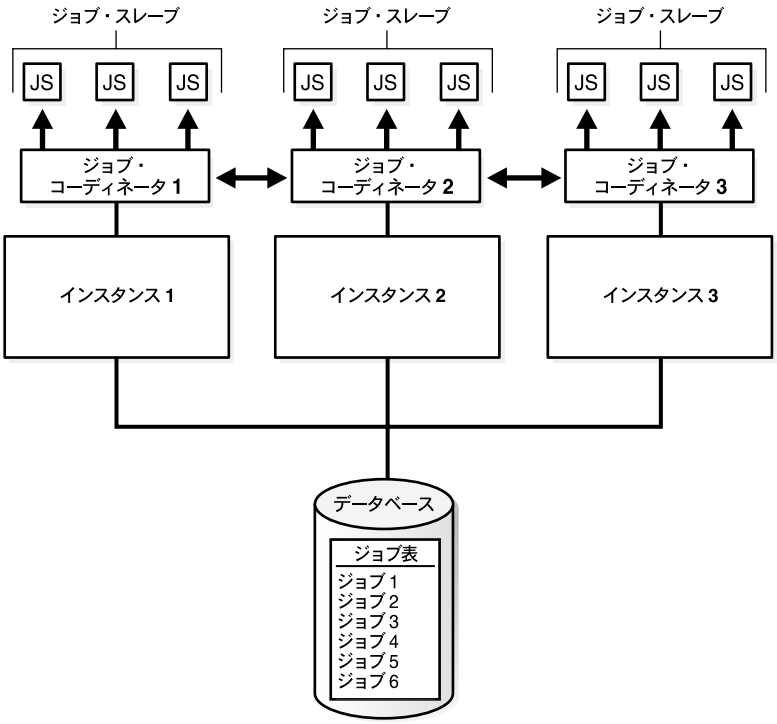
スケジューラは、必要に応じてスレーブ・プールを動的にサイズ変更します。

Real Application Clusters 環境におけるスケジューラの使用

Real Application Clusters（RAC）環境でのスケジューラでは、各データベースに対して 1 つのジョブ表が使用され、各インスタンスに対して 1 つのジョブ・コーディネータが使用されます。ジョブ・コーディネータは相互に通信し、情報を最新に保ちます。スケジューラは、ジョブ・クラスのジョブの負荷を均等にするように試みます。ジョブ・クラスにサービス・アフィニティがない場合は、使用可能なすべてのインスタンス間で負荷を均等にし、ジョブ・クラスにサービス・アフィニティがある場合は、特定のサービスに割り当てられたインスタンス間で負荷を均等にします。

図 26-5 に、典型的な RAC アーキテクチャを示します。各インスタンスのジョブ・コーディネータは他のコーディネータと情報を交換します。

図 26-5 RAC のアーキテクチャとスケジューラ



スケジューラ使用時のサービス・アフィニティ

スケジューラを使用すると、ジョブを実行するデータベース・サービスを指定できます（サービス・アフィニティ）。この結果、インスタンス・アフィニティより可用性が向上します。これは、インスタンスが停止した場合に、そのサービスに他のノードが動的に割り当てられることが保証されるためです。インスタンス・アフィニティにはこの機能がないため、インスタンスが停止した場合、そのインスタンスに対するアフィニティを持つジョブはいずれもインスタンスが回復するまで実行できません。図 26-6 は、サービスとインスタンスの典型的な使用例です。

図 26-6 サービス・アフィニティとスケジューラ

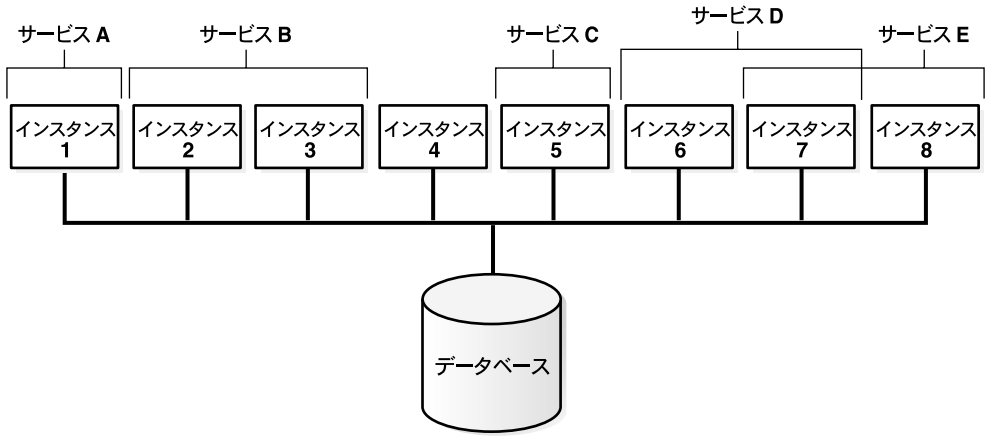


図 26-6 では、サービスのプロパティが変更可能で、その変更はスケジューラによって自動的に認識されます。

各ジョブ・クラスで、データベース・サービスを指定できます。サービスの指定がない場合、このジョブ・クラスは、起動しているすべてのインスタンスへのマッピングが保証されている内部サービスに属します。

スケジューラによる Oracle Data Guard のサポート

Oracle Database 11*g* リリース 1 からは、データベースがプライマリ・データベースかロジカル・スタンバイ・データベースかに基づいて、スケジューラが Oracle Data Guard 環境でジョブを実行できるようになりました。

フィジカル・スタンバイ・データベースの場合、スケジューラのオブジェクトに対する変更またはプライマリ・データベース上のスケジューラのジョブによるデータベースの変更は、他のデータベースの変更と同様にフィジカル・スタンバイに適用されます。

プライマリ・データベースおよびロジカル・スタンバイ・データベースの場合は、データベースのロールがプライマリ・データベースの場合のみ、またはデータベースのロールがロジカル・スタンバイの場合のみジョブを実行できるように指定する追加機能があります。指定するには、`DBMS_SCHEDULER.SET_ATTRIBUTE` プロシージャを使用して `database_role` ジョブ属性を `'PRIMARY'` または `'LOGICAL STANDBY'` のいずれかの値に設定します（両方のロールでジョブを実行するには、ジョブのコピーを作成して、一方のジョブの `database_role` を `'PRIMARY'` に設定し、もう一方のジョブを `'LOGICAL STANDBY'` に設定します）。スイッチオーバーまたはフェイルオーバーの際、スケジューラは、新しいロールに固有のジョブの実行に自動的に切り替わります。フェイルオーバーの際は、プライマリ・データベースでそれまで正常に実行されていた記録を利用できるように、DML がジョブ・イベント・ログに複製されます。

関連項目：

- `database_role` 属性の設定例は、28-25 ページの「属性の設定例」を参照してください。
- 28-30 ページ「Oracle Data Guard 環境でのジョブの作成例」
- 『Oracle Data Guard 概要および管理』

27

Oracle Scheduler を使用したジョブのスケジューリング

この章の内容は次のとおりです。

> **注意：** この章では、DBMS_SCHEDULER パッケージを使用してスケジューラ・オブジェクトを処理する方法について説明します。同じ作業を Oracle Enterprise Manager を使用して実行できます。
>
> DBMS_SCHEDULER については『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を、Oracle Scheduler の各ページについては Oracle Enterprise Manager のオンライン・ヘルプを参照してください。

スケジューラ・オブジェクトとそのネーミング

各スケジューラ・オブジェクトは、[schema.]name という形式の完全なデータベース・スキーマ・オブジェクトです。スケジューラ・オブジェクトは、データベース・オブジェクトのネーミング規則に厳密に従っています。また、他のデータベース・オブジェクトと SQL ネームスペースを共有しています。

スケジューラ・オブジェクトの名前が DBMS_SCHEDULER パッケージで使用されるときも、SQL ネーミング規則に従います。スケジューラ・オブジェクト名は、デフォルトでは大文字です。大文字にしない場合は二重引用符で囲みます。たとえば、ジョブの作成時には、job_name => 'my_job' は、job_name => 'My_Job' および job_name => 'MY_JOB' と同じですが、job_name => '"my_job"' とは同じではありません。DBMS_SCHEDULER パッケージ内でスケジューラ・オブジェクト名のカンマ区切りのリストが使用される場合も、これらのネーミング規則に従います。

オブジェクトのネーミングの詳細は、『Oracle Database SQL リファレンス』を参照してください。

ジョブの使用

ジョブは、スケジュールとプログラムの組合せで、他にプログラムに必要な引数が含まれます。この項では、ジョブの基本的なタスクについて説明します。この項の内容は、次のとおりです。

- ジョブのタスクとそのプロシージャ
- ジョブの作成
- ジョブの変更
- ジョブの実行
- ジョブの停止
- ジョブの削除
- ジョブの無効化
- ジョブの有効化
- ジョブのコピー
- ジョブ・ログの表示
- 外部ジョブの stdout と stderr の表示

> **関連項目：** ジョブの概要については、26-5 ページの「ジョブ」を参照してください。

ジョブのタスクとそのプロシージャ

表 27-1 に、ジョブの一般的なタスクとそれに対応するプロシージャおよび権限を示します。

表 27-1 ジョブのタスクとそのプロシージャ

タスク	プロシージャ	必要な権限
ジョブの作成	CREATE_JOB または CREATE_JOBS	CREATE JOB または CREATE ANY JOB
ジョブの変更	SET_ATTRIBUTE または SET_JOB_ATTRIBUTES	ALTER または CREATE ANY JOB、あるいは所有者
ジョブの実行	RUN_JOB	ALTER または CREATE ANY JOB、あるいは所有者
ジョブのコピー	COPY_JOB	ALTER または CREATE ANY JOB、あるいは所有者
ジョブの削除	DROP_JOB	ALTER または CREATE ANY JOB、あるいは所有者

表 27-1 ジョブのタスクとそのプロシージャ（続き）

タスク	プロシージャ	必要な権限
ジョブの停止	STOP_JOB	ALTER または CREATE ANY JOB、あるいは所有者
ジョブの使用禁止	DISABLE	ALTER または CREATE ANY JOB、あるいは所有者
ジョブの使用可能化	ENABLE	ALTER または CREATE ANY JOB、あるいは所有者

権限の詳細は、28-31 ページの「スケジューラ権限」を参照してください。

ジョブの作成

この項の内容は次のとおりです。

ジョブ作成の概要

1 つ以上のジョブを作成するには、CREATE_JOB または CREATE_JOBS プロシージャ、あるいは Enterprise Manager を使用します。単一のジョブを作成するには、CREATE_JOB プロシージャを使用します。このプロシージャを使用して、異なるオブジェクトに基づく様々なタイプの複数のジョブを作成するとオーバーロードになります。単一トランザクションに複数のジョブを作成する場合は、CREATE_JOBS プロシージャを使用してください。

各ジョブの作成時には、ジョブ・タイプ、処理、スケジュール、オプションのジョブ・クラスおよび他の属性を指定します。ジョブはデフォルトでは使用禁止で作成されるため、実行するには DBMS_SCHEDULER.ENABLE を使用して使用可能にする必要があります。ジョブを使用可能にすると、次にスケジュールされている日時にスケジューラによって自動的に実行されます。CREATE_JOB プロシージャの enabled 引数を TRUE に設定することもできます。この場合、ジョブは作成直後にスケジュールに従って自動的に実行できるようになります。

例 27-1 に、update_sales という単一ジョブを作成する例を示します。このジョブは、売上集計表を更新する、OPS スキーマ内のストアド・プロシージャをコールします。

例 27-1 ジョブの作成

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name           =>  'update_sales',
   job_type           =>  'STORED_PROCEDURE',
   job_action         =>  'OPS.SALES_PKG.UPDATE_SALES_SUMMARY',
   start_date         =>  '28-APR-08 07.00.00 PM Australia/Sydney',
   repeat_interval    =>  'FREQ=DAILY;INTERVAL=2', /* every other day */
   end_date           =>  '20-NOV-08 07.00.00 PM Australia/Sydney',
   job_class          =>  'batch_update_jobs',
   comments           =>  'My new job');
END;
/
```

schema.job_name を指定すると、別のスキーマ内にジョブを作成できます。したがって、ジョブの作成者がジョブの所有者であるとはかぎりません。ジョブの所有者は、ジョブが作成されるスキーマを所有しているユーザーです。ジョブは、そのジョブが作成されるスキーマの

権限で実行されます。実行時のジョブの NLS 環境は、そのジョブが作成された時点に存在していた環境です。

作成したジョブは、*_SCHEDULER_JOBS ビューを使用して問い合せることができます。

ジョブは、デフォルトでは完了後に自動的に削除されるように設定されています。auto_drop 属性を FALSE に設定すると、ジョブは保持されます。繰返しジョブは、ジョブ終了日を過ぎるか、最大実行回数（max_runs）または最大失敗回数（max_failures）に達するまでは自動削除されないことに注意してください。

ジョブの処理とジョブ・スケジュールの指定

CREATE_JOB プロシージャはオーバーロードになるため、複数の異なる使用方法があります。例 27-1 に示すように、ジョブの処理と繰返し間隔をジョブ属性として指定する方法（ジョブの処理とジョブ・スケジュールのインライン指定）に加えて、プログラム・オブジェクト（プログラム）を指し示すジョブを作成してジョブの処理を指定する方法、スケジュール・オブジェクト（スケジュール）を指し示すジョブを作成して繰返し間隔を指定する方法、またはプログラムとスケジュールの両方を指し示すジョブを作成する方法があります。この操作については、次の項を参照してください。

- 名前付きプログラムを使用したジョブの作成
- 名前付きスケジュールを使用したジョブの作成
- 名前付きプログラムとスケジュールを使用したジョブの作成

名前付きプログラムを使用したジョブの作成 ジョブの処理をインラインで記述するかわりに、名前付きプログラムを指し示してジョブを作成することもできます。名前付きプログラムを使用してジョブを作成するには、ジョブの作成時に CREATE_JOB プロシージャで program_name の値を指定し、job_type、job_action および number_of_arguments の値は指定しません。

ジョブの作成時に既存のプログラムを使用するには、ジョブの所有者がプログラムの所有者であるか、またはそのプログラムに対する EXECUTE 権限を持っている必要があります。次の PL/SQL ブロックは、名前付きプログラムを指定した CREATE_JOB プロシージャの使用例です。この PL/SQL ブロックでは、my_new_job1 という標準ジョブが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name          =>  'my_new_job1',
   program_name      =>  'my_saved_program',
   repeat_interval   =>  'FREQ=DAILY;BYHOUR=12',
   comments          =>  'Daily at noon');
END;
/
```

次の PL/SQL ブロックでは、軽量ジョブが作成されます。軽量ジョブは 1 つのプログラムを参照する必要があり、プログラム・タイプは 'PLSQL_BLOCK' または 'STORED_PROCEDURE' であることが必要です。さらに、プログラムはジョブの作成時に使用可能になっている必要があります。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name         =>  'my_lightweight_job1',
   program_name     =>  'polling_prog_n2',
   repeat_interval  =>  'FREQ=SECONDLY;INTERVAL=10',
   end_date         =>  '30-APR-09 04.00.00 AM Australia/Sydney',
   job_style        => 'LIGHTWEIGHT',
   comments         => 'Job that polls device n2 every 10 seconds');
END;
/
```

名前付きスケジュールを使用したジョブの作成 ジョブのスケジュールをインラインで記述するかわりに、名前付きスケジュールを指し示してジョブを作成することもできます。名前付きスケジュールを使用してジョブを作成するには、ジョブの作成時に CREATE_JOB プロシージャで schedule_name の値を指定し、start_date、repeat_interval および end_date の値は指定しません。

ジョブの作成にはすべての名前付きスケジュールを使用できます。これは、スケジュールはすべて PUBLIC に対するアクセス権限付きで作成されているためです。次の文は、名前付きスケジュールを指定した CREATE_JOB プロシージャの使用例です。この文では、my_new_job2 という標準ジョブが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name                =>  'my_new_job2',
   job_type                =>  'PLSQL_BLOCK',
   job_action              =>  'BEGIN SALES_PKG.UPDATE_SALES_SUMMARY; END;',
   schedule_name           =>  'my_saved_schedule');
END;
/
```

名前付きプログラムとスケジュールを使用したジョブの作成 ジョブは、名前付きプログラムとスケジュールの両方を指し示して作成することもできます。次の文は、名前付きプログラムとスケジュールを指定した CREATE_JOB プロシージャの使用例です。この文では、既存のプログラム my_saved_program1 と既存のスケジュール my_saved_schedule1 に基づいて、my_new_job3 という標準ジョブが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name           =>  'my_new_job3',
   program_name       =>  'my_saved_program1',
   schedule_name      =>  'my_saved_schedule1');
END;
/
```

ジョブ引数の設定

ジョブの作成後、次の場合にジョブ引数の設定が必要な場合があります。

- インラインのジョブ処理が、引数を必要とするストアド・プロシージャまたはその他の実行可能ファイルの場合
- ジョブが名前付きプログラム・オブジェクトを参照していて、そのデフォルト・プログラム引数の 1 つ以上を上書きする場合
- ジョブが名前付きプログラム・オブジェクトを参照していて、そのプログラム引数の 1 つ以上にデフォルト値が割り当てられていない場合

ジョブ引数を設定するには、SET_JOB_ARGUMENT_VALUE または SET_JOB_ANYDATA_VALUE プロシージャ、あるいは Enterprise Manager を使用します。SET_JOB_ANYDATA_VALUE は、VARCHAR2 文字列として表現できない複雑なデータ型に使用されます。

引数を必要とする場合があるジョブの例の 1 つに、開始日と終了日が必要なレポート作成プログラムを開始するジョブがあります。次のコード例では、レポート作成プログラムの 2 番目の引数である終了日のジョブ引数が設定されます。

```
BEGIN
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (
   job_name               => 'ops_reports',
   argument_position      => 2,
   argument_value         => '12-DEC-03');
END;
/
```

値がすでに設定されている引数についてこのプロシージャを使用すると、その引数は上書きされます。引数値を設定するには、引数名または引数の位置を使用します。引数名を使用するには、ジョブが名前付きプログラム・オブジェクトを参照し、そのプログラム・オブジェクト内

で引数に名前が割り当てられている必要があります。プログラムがインラインで記述されている場合にサポートされるのは、位置による設定のみです。引数は、タイプが plsql_block のジョブに対してはサポートされません。

設定されている値を削除するには、RESET_JOB_ARGUMENT プロシージャを使用します。このプロシージャは、通常の引数および ANYDATA 引数の両方に使用できます。

SET_JOB_ARGUMENT_VALUE および SET_JOB_ANYDATA_VALUE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ジョブ属性の設定

ジョブの作成後に、SET_ATTRIBUTE プロシージャ、SET_JOB_ATTRIBUTES プロシージャまたは Enterprise Manager を使用してジョブ属性を設定できます。CREATE_JOB のコール中に多数のジョブ属性を設定できますが、ジョブの作成後に SET_ATTRIBUTE または SET_JOB_ATTRIBUTES でのみ設定可能な属性もあります。

SET_ATTRIBUTE プロシージャ、SET_JOB_ATTRIBUTES プロシージャおよび様々なジョブ属性の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

外部ジョブの作成

ローカル外部ジョブまたはリモート外部ジョブを作成するには、CREATE JOB 権限と CREATE EXTERNAL JOB 権限の両方が必要です。

ローカル外部ジョブまたはリモート外部ジョブを作成する手順は、次のとおりです。

1. 外部ジョブに必要な設定タスクがすべて実行済であることを確認します。
   - ローカル外部ジョブの場合は、Oracle Database のプラットフォーム・ガイドを参照してください。
   - リモート外部ジョブの場合は、28-13 ページの「リモート外部ジョブを実行するためのデータベースの設定」の説明に従ってデータベース設定手順を完了してください。
2. DBMS_SCHEDULER パッケージの CREATE_JOB プロシージャを使用してジョブを作成します。

   enabled 属性を省略するか、または FALSE に設定します。
3. CREATE_CREDENTIAL プロシージャを使用して資格証明を作成します。

   詳細は、26-8 ページの「資格証明の概要」を参照してください。

   > **注意：** Windows では、外部実行可能ファイルを実行するホスト・ユーザーに Log on as a batch job ログオン権限を割り当てる必要があります。

4. SET_ATTRIBUTE プロシージャを使用して、ジョブの credential_name 属性を設定します。

   ジョブの所有者は、資格証明の EXECUTE 権限を持っているか、または資格証明の所有者であることが必要です。リモート外部ジョブの場合、credential_name 属性が必須です。ローカル外部ジョブの場合、この属性が設定されていなければデフォルトの資格証明が使用されます。詳細は、26-9 ページの表 26-1 を参照してください。

   > **注意：** セキュリティ向上のために、ローカル外部ジョブに資格証明を割り当てることをお薦めします。

5. リモート外部ジョブの場合のみ、SET_ATTRIBUTE プロシージャを使用して、ジョブの destination 属性を設定します。

   属性は、*host*:*port* の形式で指定する必要があります。*host* にはリモート・ホストのホスト名または IP アドレス、*port* にはホストがリスニングするスケジューラ・エージェントのポートが入ります。このポート番号を判断するには、ファイル schagent.conf を表示します。このファイルは、リモート・ホストのスケジューラ・エージェントのホーム・ディレクトリに格納されています。

6. （オプション）リモート外部ジョブの場合のみ、nslookup などのユーティリティを使用して、リモート・ホスト名が有効であることとホストにアクセス可能であることを確認します。

7. ENABLE プロシージャを使用してジョブを有効化します。

例 27-2 ローカル外部ジョブの作成

次の例では、LOGOWNER という名前の資格証明を使用して CLEANLOGS というローカル外部ジョブが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB(
   job_name             => 'CLEANLOGS',
   job_type             => 'EXECUTABLE',
   job_action           => '/home/logowner/cleanlogs',
   repeat_interval      => 'FREQ=DAILY; BYHOUR=23',
   enabled              => FALSE);
DBMS_SCHEDULER.SET_ATTRIBUTE('CLEANLOGS', 'credential_name', 'LOGOWNER');
DBMS_SCHEDULER.ENABLE('CLEANLOGS');
END;
/
```

ジョブの作成時には資格証明を指定できず、かわりに SET_ATTRIBUTE を使用して指定する必要があるため、資格証明を設定する機会を設けるためにジョブは使用禁止の状態で作成されます。資格証明の設定後、ジョブが使用可能になります。

例 27-3 DOS コマンドを実行するローカル外部ジョブの作成

次の例に、Windows 上で DOS 組込みコマンド（この例では mkdir）を実行するローカル外部ジョブの作成方法を示します。このジョブでは、/c オプションを指定して cmd.exe が実行されます。デフォルトの資格証明が使用されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB(
   job_name             => 'MKDIR_JOB',
   job_type             => 'EXECUTABLE',
   number_of_arguments  => 3,
   job_action           => '¥windows¥system32¥cmd.exe',
   auto_drop            => FALSE);

DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('mkdir_job',1,'/c');
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('mkdir_job',2,'mkdir');
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('mkdir_job',3,'¥temp¥extjob_test_dir');
DBMS_SCHEDULER.SET_ATTRIBUTE('MKDIR_JOB', 'credential_name', 'STEVE');
DBMS_SCHEDULER.ENABLE('MKDIR_JOB');
END;
/
```

例 27-4 複数のリモート・ホストに対するリモート外部ジョブの作成

次の例では、複数のリモート・ホストに対して同じリモート外部ジョブが作成されます。PL/SQL コードには、ホスト名を繰り返すループが組み込まれています。remote_cred はすべてのホストに有効な資格証明の名前です。作成先のリストは、ホスト名とスケジューラ・エージェントのポートのリストです。すべてのホストで実行される実行可能ファイルは、/u01/app/ext_backup というアプリケーションです。

このコードを実行するユーザーには、CREATE JOB 権限と CREATE EXTERNAL JOB 権限の両方が必要です。

```
declare
job_prefix varchar2(30) := 'remote_';
job_name varchar2(30);
destinations dbms_utility.lname_array;
begin

   destinations(1) := 'host1:1234';
   destinations(2) := 'host2:1234';
   destinations(3) := 'host3:1234';
   destinations(4) := 'host4:1234';

  for i in 1..destinations.LAST loop
    job_name := dbms_scheduler.generate_job_name(job_prefix);
    dbms_scheduler.create_job(job_name,
    job_type=>'executable',
    job_action=>'/u01/app/ext_backup',
    number_of_arguments=>0,
    enabled=>false);

    dbms_scheduler.set_attribute(job_name,'destination',destinations(i));
    dbms_scheduler.set_attribute(job_name,'credential_name','remote_cred');
    dbms_scheduler.enable(job_name);
  end loop;
end;
/
```

次に、この例について説明します。

- ジョブには開始日が指定されていないため、即時に実行されます。
- ジョブには繰返し間隔が指定されていないため、完了時に削除されます。
- ジョブは使用禁止の状態で作成され、ジョブの資格証明が設定されるまで使用可能になりません。

例 27-5 SQL 文を発行するリモート外部ジョブの作成

次の例では、リモート外部ジョブで SQL 文をリモートの Oracle データベースに発行する方法を示しています。ジョブの処理では、SQL*Plus を使用して文を発行するシェル・スクリプトが実行されます。このスクリプトは、リモート・ホストに存在している必要があります。次のスクリプトは、Linux 上で SQL*Plus を実行するために必要な環境変数すべてを設定することから開始しています。

データベース・パスワードをスクリプトにハードコードしないように、外部認証が使用されています。

```
#!/bin/sh

export ORACLE_HOME=/u01/app/oracle/product/11.1.0/db_1
export ORACLE_SID=orcl
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib

# The following command assumes external authentication
$ORACLE_HOME/bin/sqlplus / << EOF
set serveroutput on;
```

```
select * from dual;
EXIT;
EOF
```

例 27-6 ローカル外部ジョブの作成と stdout の取得

Linux と UNIX に関する次の例では、ローカル外部ジョブを作成して実行してから、GET_FILE プロシージャを使用してジョブの stdout の出力を取得する方法を示しています。ローカル外部ジョブの場合、stdout の出力は *ORACLE_HOME*/scheduler/log 内のログ・ファイルに格納されます。このパスを GET_FILE に指定する必要はなく、ファイル名のみを指定します。ファイル名を生成するには、ジョブの外部ログ ID についてログ・ビューを問い合せて _stdout を追加します。

```
-- User scott must have CREATE JOB and CREATE EXTERNAL JOB privileges
grant create job, create external job to scott ;

connect scott/tiger
set serveroutput on

-- Create a credential for the job to use
exec dbms_scheduler.create_credential('my_cred','host_username','host_passwd')

-- Create a job that lists a directory. After running, the job is dropped.
begin
DBMS_SCHEDULER.CREATE_JOB(
job_name=>'lsdir',
job_type=>'EXECUTABLE',
job_action=>'/bin/ls',
number_of_arguments => 1,
enabled => false,
auto_drop =>true
);
dbms_scheduler.set_job_argument_value('lsdir',1,'/tmp');
dbms_scheduler.set_attribute('lsdir','credential_name','my_cred');
dbms_scheduler.enable('lsdir');
end;
/

-- Wait a bit for the job to run, and then check the job results.
select job_name, status, error#, actual_start_date, additional_info
 from user_scheduler_job_run_details where job_name='LSDIR';

-- Now use the external log id from the additional_info column to
-- formulate the log file name and retrieve the output
declare
 my_clob clob;
 log_id varchar2(50);
begin
 select regexp_substr(additional_info,'job[_0-9]*') into log_id
   from user_scheduler_job_run_details where job_name='LSDIR';
 dbms_lob.createtemporary(my_clob, false);
 dbms_scheduler.get_file(
   source_file     => log_id ||'_stdout',
   credential_name => 'my_cred',
   file_contents   => my_clob,
   source_host     => null);
 dbms_output.put_line(my_clob);
end;
/
```

注意： リモート外部ジョブの場合、メソッドは同じですが次の違いがあります。

- ジョブの destination 属性を設定します。
- GET_FILE プロシージャにソース・ホストを指定します。

GET_FILE では、ローカル外部ジョブとリモート外部ジョブの両方について、正しいホスト位置でログ・ファイルが自動的に検索されます。

関連項目：

- 外部認証の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。
- 26-8 ページの「外部ジョブ」
- 27-19 ページ「外部ジョブの stdout と stderr の表示」
- 27-14 ページ「外部ジョブの停止」

デタッチ済ジョブの作成

デタッチ済ジョブでは、detached 属性が TRUE に設定されているプログラム・オブジェクト（プログラム）を指し示す必要があります。

例 27-7 コールド・バックアップを実行するデタッチ済ジョブの作成

Linux と UNIX に関する次の例では、データベースのコールド・バックアップを実行する夜間ジョブが作成されます。このジョブには 3 つのステップが含まれています。

ステップ 1: RMAN 起動スクリプトの作成

コールド・バックアップを実行するための RMAN スクリプトをコールするシェル・スクリプトを作成します。このシェル・スクリプトは $ORACLE_HOME/scripts/coldbackup.sh に置かれます。Oracle Database をインストールしたユーザー（通常はユーザー oracle）が実行できるようにする必要があります。

```
#!/bin/sh

export ORACLE_HOME=/u01/app/oracle/product/11.1.0/db_1
export ORACLE_SID=orcl
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib

$ORACLE_HOME/bin/rman TARGET / @$ORACLE_HOME/scripts/coldbackup.rman
  trace /u01/app/oracle/backup/coldbackup.out &
exit 0
```

ステップ 2: RMAN スクリプトの作成

コールド・バックアップを実行してからジョブを終了する、RMAN スクリプトを作成します。このスクリプトは $ORACLE_HOME/scripts/coldbackup.rman に置かれます。

```
run {
# Shut down database for backups and put into MOUNT mode
shutdown immediate
startup mount

# Perform full database backup
backup full format "/u01/app/oracle/backup/%d_FULL_%U" (database) ;

# Open database after backup
alter database open;
```

```
# Call notification routine to indicate job completed successfully
sql " BEGIN  DBMS_SCHEDULER.END_DETACHED_JOB_RUN(''sys.backup_job'', 0,
  null); END; ";
}
```

ステップ 3: ジョブの作成とデタッチ済プログラムの使用

次の PL/SQL ブロックを発行します。

```
BEGIN
  DBMS_SCHEDULER.CREATE_PROGRAM(
    program_name   => 'sys.backup_program',
    program_type   => 'executable',
    program_action => '?/scripts/coldbackup.sh',
    enabled        =>  TRUE);

  DBMS_SCHEDULER.SET_ATTRIBUTE('sys.backup_program', 'detached', TRUE);

  DBMS_SCHEDULER.CREATE_JOB(
    job_name        => 'sys.backup_job',
    program_name    => 'sys.backup_program',
    repeat_interval => 'FREQ=DAILY;BYHOUR=1;BYMINUTE=0');

  DBMS_SCHEDULER.ENABLE('sys.backup_job');
END;
/
```

関連項目： 26-10 ページの「デタッチ済ジョブ」

単一トランザクションでの複数ジョブの作成

多数のジョブを作成する必要がある場合は、CREATE_JOBS プロシージャを使用すると、トランザクションのオーバーヘッドを減らしてパフォーマンスを改善できる可能性があります。例 27-8 に、このプロシージャを使用して単一トランザクションで複数のジョブを作成する方法を示します。詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

例 27-8 単一トランザクションでの複数ジョブの作成

```
DECLARE
 newjob sys.job;
 newjobarr sys.job_array;
BEGIN
 -- Create an array of JOB object types
 newjobarr := sys.job_array();

 -- Allocate sufficient space in the array
 newjobarr.extend(5);

 -- Add definitions for 5 jobs
 FOR i IN 1..5 LOOP
   -- Create a JOB object type
   newjob := sys.job(job_name => 'TESTJOB' || to_char(i),
                     job_style => 'REGULAR',
                     job_template => 'PROG1',
                     repeat_interval => 'FREQ=HOURLY',
                     start_date => systimestamp + interval '600' second,
                     max_runs => 2,
                     auto_drop => FALSE,
                     enabled => TRUE
                    );

   -- Add it to the array
   newjobarr(i) := newjob;
 END LOOP;
```

```
 -- Call CREATE_JOBS to create jobs in one transaction
 DBMS_SCHEDULER.CREATE_JOBS(newjobarr, 'TRANSACTIONAL');
END;
/

PL/SQL procedure successfully completed.

SELECT JOB_NAME FROM USER_SCHEDULER_JOBS;

JOB_NAME
------------------------------
TESTJOB1
TESTJOB2
TESTJOB3
TESTJOB4
TESTJOB5

5 rows selected.
```

ジョブの変更

ジョブを変更するには、SET_ATTRIBUTE または SET_JOB_ATTRIBUTES プロシージャ、あるいは Enterprise Manager を使用します。ジョブはすべて変更できます。また、ジョブ名以外のすべてのジョブ属性を変更できます。変更の際に実行中のジョブ・インスタンスがあった場合、そのジョブはコールの影響を受けません。変更は、次回のジョブ実行時から反映されます。

通常、データベースによって自動的に作成されたジョブは変更しないでください。データベースによって作成されたジョブの場合、ジョブ・ビューで SYSTEM 列が TRUE に設定されます。ジョブの属性は、*_SCHEDULER_JOBS ビューで使用可能です。

実行中のジョブに対してそのジョブ属性を変更することは有効ですが、変更した属性は、スケジュールされている次回のジョブ実行時まで反映されません。

SET_ATTRIBUTE および SET_JOB_ATTRIBUTES プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』と 28-2 ページの「Oracle Scheduler の構成」を参照してください。

ジョブの実行

ジョブを実行するには、次の 3 つの方法があります。

- ジョブ・スケジュールに従って実行する方法 : この場合、ジョブが使用可能であれば、そのジョブがスケジューラのジョブ・コーディネータによって自動的に選択され、ジョブ・スレーブの制御下で実行されます。ジョブは、ジョブ所有者として実行されます。ジョブが成功したかどうかを確認するには、ジョブ・ビュー（*_SCHEDULER_JOBS）またはジョブ・ログを問い合せる必要があります。ジョブ・スレーブとスケジューラのアーキテクチャの詳細は、26-18 ページの「ジョブ・スレーブ」を参照してください。
- イベントの発生時に実行する方法 : イベント・キューで特定のイベントが受信されると、使用可能になっているイベントベースのジョブが開始されます。(27-41 ページの「イベントの使用」を参照してください)。イベントベースのジョブも、ジョブ・スレーブの制御下でジョブを所有するユーザーとして実行されます。ジョブが成功したかどうかを確認するには、ジョブ・ビューまたはジョブ・ログを問い合せる必要があります。
- DBMS_SCHEDULER.RUN_JOB をコールして実行する方法 : RUN_JOB プロシージャを使用すると、ジョブをテストしたり、指定したスケジュール外で実行できます。ジョブは、前の 2 つのジョブ実行方法と同様に非同期で実行するか、同期で実行できます。後者の場合、ジョブは RUN_JOB をコールしたセッションで実行されます。

> **注意 :** スケジュールに従ってジョブを実行する場合、RUN_JOB をコールする必要はありません。ジョブが使用可能であれば、スケジューラによって自動的に実行されます。

DBMS_SCHEDULER.RUN_JOB によるジョブの非同期実行

ジョブを非同期で実行するには、use_current_session 引数を FALSE に設定して RUN_JOB を使用します。この場合、ジョブはスケジュールに従って開始される場合やイベントにより開始される場合と同様に実行されます。つまり、ジョブ・スレーブの制御下でジョブ所有者として実行されます。RUN_JOB をコールするセッションは即時に戻り、ジョブの完了を待機する間にブロックされることはありません。ジョブが成功したかどうかを確認するには、ジョブ・ビューまたはジョブ・ログを問い合せる必要があります。

DBMS_SCHEDULER.RUN_JOB によるジョブの同期実行

use_current_session 引数を TRUE に設定して RUN_JOB を使用すると、ジョブを同期で実行できます。この場合、ジョブはジョブ・スレーブによって実行されるかわりに、RUN_JOB を起動するユーザー・セッション内で実行されます。RUN_JOB をコールするセッションは、ジョブの完了までブロックされます。

RUN_JOB を使用してジョブを同期で実行すると、ジョブの failure_count および run_count は変更されません。ただし、ジョブ・ログにはジョブの実行が反映されます。ジョブによって生成されたランタイム・エラーは、RUN_JOB の実行者に返送されます。

RUN_JOB を使用してリモート外部ジョブまたはチェーンを指し示すジョブを実行する場合は、use_current_session を FALSE に設定する必要があります。

ジョブの実行環境

ジョブは、ジョブの所有者に直接付与された権限、またはデフォルトのログイン・ロールによって間接的に付与された権限で実行されます。外部オペレーティング・システムのロールはサポートされていません。適切な権限が付与されている場合、ユーザーは他のユーザーのスキーマ内にジョブを作成できます。したがって、ジョブの作成者と所有者は異なる場合があります。たとえば、ユーザー jim に CREATE ANY JOB 権限があり、scott のスキーマ内にジョブを作成した場合、そのジョブは scott の権限で実行されます。

ジョブが作成されたセッションの NLS 環境が保存され、ジョブの実行時に使用されます。ジョブが実行される NLS 環境を変更するには、別の NLS 設定のセッション内にジョブを作成する必要があります。

ジョブの停止

実行中の 1 つ以上のジョブを停止するには、STOP_JOB プロシージャまたは Enterprise Manager を使用します。STOP_JOB は、ジョブおよびジョブ・クラスのカンマ区切りのリストを受け入れます。ジョブ・クラスが指定されている場合、そのジョブ・クラス内の実行中のジョブはすべて停止されます。たとえば、次の文では、ジョブ job1 とジョブ・クラス dw_jobs 内のすべてのジョブが停止されます。

```
BEGIN
DBMS_SCHEDULER.STOP_JOB('job1, sys.dw_jobs');
END;
/
```

指定したジョブのインスタンスはすべて停止されます。ジョブの停止後、1 回かぎりのジョブの状態は STOPPED に設定され、繰返しジョブの状態は、（次回のジョブ実行がスケジュールされているため）SCHEDULED に設定されます。また、ジョブ・ログには、OPERATION が 'STOPPED' に設定され、ADDITIONAL_INFO が 'REASON="Stop job called by user: *username*"' に設定されたエントリが作成されます。

デフォルトでは、スケジューラは割込みメカニズムを使用してジョブを正常に停止しようとします。この方法では、制御がスレーブ・プロセスに戻され、スレーブ・プロセスはジョブ実行の統計を収集できます。force オプションが TRUE に設定されている場合、ジョブは即時に停止するため、そのジョブ実行について特定の実行時間の統計が使用できない場合があります。

チェーンを実行中のジョブを停止すると、実行中のすべてのステップが（各ステップで force オプションを TRUE に設定して STOP_JOB をコールすることによって）自動的に停止されます。

STOP_JOB プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

注意： ジョブが停止したときにロールバックされるのは、現行トランザクションのみです。これは、データの一貫性を損なう原因となる場合があります。

外部ジョブの停止

スケジューラを使用している場合、外部ジョブの実装者は、force が FALSE に設定された STOP_JOB がコールされた場合に、その外部ジョブを正常にクリーン・アップするメカニズムを利用できます。次の説明は、すべてのプラットフォーム上で資格証明を使用せずに作成されたローカル外部ジョブと、UNIX および Linux プラットフォーム上のリモート外部ジョブにのみ該当します。

UNIX および Linux では、スケジューラによって起動されたプロセスに SIGTERM シグナルが送信されます。外部ジョブの実装者は、割込みハンドラに SIGTERM をトラップし、ジョブの実装内容をすべてクリーン・アップして終了する必要があります。Windows では、force が FALSE に設定された STOP_JOB は、Windows XP、Windows 2003 およびそれ以降のオペレーティング・システムでのみサポートされます。これらのプラットフォームでは、スケジューラによって起動されるプロセスがコンソール・プロセスです。このプロセスを停止するために、スケジューラは CTRL-BREAK をプロセスに送信します。CTRL_BREAK は、SetConsoleCtrlHandler() ルーチンにハンドラを登録することによって処理できます。

チェーン・ジョブの停止

チェーンを指し示すジョブが停止すると、実行中のチェーンのステップがすべて停止します。

個々のチェーン・ステップの停止の詳細は、27-59 ページの「個々のチェーン・ステップの停止」を参照してください。

ジョブの削除

1 つ以上のジョブを削除するには、DROP_JOB プロシージャまたは Enterprise Manager を使用します。DROP_JOB は、ジョブおよびジョブ・クラスのカンマ区切りのリストを受け入れます。ジョブ・クラスが指定されている場合、そのジョブ・クラス内のジョブはすべて削除されます。ただし、そのジョブ・クラス自体は削除されません。

たとえば、次の文では、ジョブ job1 と job3、およびジョブ・クラス jobclass1 と jobclass2 内のすべてのジョブが削除されます。

```
BEGIN
DBMS_SCHEDULER.DROP_JOB ('job1, job3, sys.jobclass1, sys.jobclass2');
END;
/
```

ジョブを削除すると、そのジョブはジョブ表から削除され、そのメタデータも削除され、*_SCHEDULER_JOBS ビューに表示されなくなります。したがって、これ以降ジョブは実行されません。

DROP_JOB のコール時にジョブのインスタンスが実行中の場合、そのコールはエラーになります。この場合でも、コールで force オプションを TRUE に設定すると、ジョブを削除できます。force オプションを TRUE に設定すると、最初に、実行中のジョブ・インスタンスが割込みメカニズムによって停止（force オプションを FALSE に設定して STOP_JOB をコール）され、次にジョブが削除されます。

または、STOP_JOB をコールして最初にジョブを停止してから、DROP_JOB をコールしてジョブを削除することもできます。MANAGE SCHEDULER 権限が付与されている場合は、通常の STOP_JOB コールがジョブの停止に失敗したときに force 付きで STOP_JOB をコールし、その後で DROP_JOB をコールできます。

デフォルトでは、force は FALSE に設定されます。

commit_semantics が STOP_ON_FIRST_ERROR に設定されている場合は、最初のエラーでコールが戻り、エラー発生前に正常終了した削除操作がディスクにコミットされます。commit_semantics が TRANSACTIONAL に、force が FALSE に設定されている場合は、最初のエラーでコールが戻り、エラー発生前の削除操作がロールバックされます。commit_semantics が ABSORB_ERRORS に設定されている場合は、エラーへの対応が取り組まれ、残りのジョブの削除が試行されて、正常に終了したすべての削除操作がコミットされます。デフォルトでは、commit_semantics は STOP_ON_FIRST_ERROR に設定されています。

ジョブ・クラスを削除するには、DROP_JOB_CLASS プロシージャを使用してください。ジョブ・クラスの削除方法は、27-29 ページの「ジョブ・クラスの削除」を参照してください。

DROP_JOB プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ジョブの無効化

1 つ以上のジョブを無効（使用禁止）にするには、DISABLE プロシージャまたは Enterprise Manager を使用します。ジョブは他の理由で使用禁止になる場合もあります。たとえば、ジョブが属しているジョブ・クラスが削除されると、ジョブは使用禁止になります。また、ジョブが指し示しているプログラムまたはスケジュールのいずれかが削除された場合も使用禁止になります。ジョブが指し示しているプログラムまたはスケジュールが使用禁止の場合は、そのジョブ自体は使用禁止にならないため、スケジューラがジョブを実行しようとしたときに、エラーとなることに注意してください。

ジョブを使用禁止にすると、そのジョブのメタデータはそのまま存在しますが、ジョブ自体は実行対象ではないため、ジョブ・コーディネータがこれらのジョブを取り出して処理することはありません。ジョブが使用禁止になると、ジョブ表内のその state は disabled に変更されます。

force オプションを FALSE に設定してジョブを使用禁止にすると、ジョブが現在実行中の場合はエラーが返されます。force が TRUE に設定されているときは、ジョブは使用禁止になりますが、現在実行中のインスタンスは完了できます。

commit_semantics が STOP_ON_FIRST_ERROR に設定されている場合は、最初のエラーでコールが戻り、エラー発生前に正常終了した使用禁止操作がディスクにコミットされます。commit_semantics が TRANSACTIONAL に、force が FALSE に設定されている場合は、最初のエラーでコールが戻り、エラー発生前の使用禁止操作がロールバックされます。commit_semantics が ABSORB_ERRORS に設定されている場合は、エラーへの対応が取り組まれ、残りのジョブの使用禁止が試行されて、正常に終了したすべての使用禁止操作がコミットされます。デフォルトでは、commit_semantics は STOP_ON_FIRST_ERROR に設定されています。

DISABLE プロシージャ・コールにジョブ名またはジョブ・クラス名のカンマ区切りのリストを指定することで、1 回のコールで複数のジョブを使用禁止にすることもできます。たとえば、次の文では、ジョブとジョブ・クラスを組み合せて指定しています。

```
BEGIN
DBMS_SCHEDULER.DISABLE('job1, job2, job3, sys.jobclass1, sys.jobclass2');
END;
/
```

DISABLE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ジョブの有効化

1 つ以上のジョブを有効（使用可能）にするには、ENABLE プロシージャまたは Enterprise Manager を使用します。このプロシージャを使用すると、ジョブは、ジョブ・コーディネータによって取り出され、処理されるようになります。ジョブは、デフォルトで使用禁止で作成されるため、実行するには使用可能にする必要があります。ジョブを使用可能にすると、妥当性チェックが実行されます。チェックに失敗すると、ジョブは使用可能になりません。

commit_semantics が STOP_ON_FIRST_ERROR に設定されている場合は、最初のエラーでコールが戻り、エラー発生前に正常終了した使用可能にする操作がディスクにコミットされます。commit_semantics が TRANSACTIONAL に設定されている場合は、最初のエラーでコールが戻り、エラー発生前の使用可能にする操作がロールバックされます。commit_semantics が ABSORB_ERRORS に設定されている場合は、エラーへの対応が取り組まれ、残りのジョブを使用可能にする操作が試行されて、正常に終了した使用可能にする操作がすべてコミットされます。デフォルトでは、commit_semantics は STOP_ON_FIRST_ERROR に設定されています。

ENABLE プロシージャ・コールにジョブ名またはジョブ・クラス名のカンマ区切りのリストを指定することで、1 回のコールで複数のジョブを使用可能にすることもできます。たとえば、次の文では、ジョブとジョブ・クラスを組み合せて指定しています。

```
BEGIN
DBMS_SCHEDULER.ENABLE ('job1, job2, job3,
   sys.jobclass1, sys.jobclass2, sys.jobclass3');
END;
/
```

ENABLE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ジョブのコピー

ジョブをコピーするには、COPY_JOB プロシージャまたは Enterprise Manager を使用します。このコールによって、旧ジョブのすべての属性（ジョブ名以外）が新規ジョブにコピーされます。新規ジョブは使用禁止の状態で作成されます。

COPY_JOB プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ジョブ・ログの表示

ジョブの実行、状態変化および失敗について、ジョブ・ログ内の情報を表示できます。ジョブ・ログは、次の 2 つのデータ・ディクショナリ・ビューとして実装されます。

- *_SCHEDULER_JOB_LOG
- *_SCHEDULER_JOB_RUN_DETAILS

スケジューラは、有効なロギング・レベルに応じて、ジョブの実行時、作成時、削除時、有効化時などにジョブ・ログ・エントリを作成できます。繰返しスケジュールを持つジョブの場合、スケジューラではジョブ・ログにジョブ・インスタンスごとに 1 つずつ複数のエントリが作成されます。各ログ・エントリは、ジョブの完了ステータスなど、特定の実行に関する情報を提供します。

次の例では、max_runs 属性の値が 4 に設定されている繰返しジョブのジョブ・ログ・エントリを示しています。

```
SELECT job_name, job_class, operation, status FROM USER_SCHEDULER_JOB_LOG;

JOB_NAME         JOB_CLASS            OPERATION       STATUS
---------------- -------------------- --------------- ----------
JOB1             CLASS1               RUN             SUCCEEDED
JOB1             CLASS1               RUN             SUCCEEDED
JOB1             CLASS1               RUN             SUCCEEDED
JOB1             CLASS1               RUN             SUCCEEDED
JOB1             CLASS1               COMPLETED
```

ジョブまたはジョブ・クラスの logging_level 属性を設定すると、ジョブ・ログに情報が書き込まれる頻度を制御できます。表 27-2 に、logging_level に可能な値を示します。

表 27-2 ジョブのロギング・レベル

ロギング・レベル	説明
DBMS_SCHEDULER.LOGGING_OFF	ロギングは実行されません。
DBMS_SCHEDULER.LOGGING_FAILED_RUNS	ジョブが失敗した場合にのみログ・エントリが作成されます。
DBMS_SCHEDULER.LOGGING_RUNS	ジョブが実行されるたびにログ・エントリが作成されます。
DBMS_SCHEDULER.LOGGING_FULL	ジョブが実行されるたびに、および作成、有効化、無効化、更新（SET_ATTRIBUTE を使用）、停止および削除など、ジョブに対して実行された操作ごとに、ログ・エントリが作成されます。

ジョブの実行に関するログ・エントリは、ジョブの実行が正常に完了するか、失敗するかまたは停止するまで作成されません。

次の例に、ジョブのライフサイクル全体に関するジョブ・ログ・エントリを示します。この例では、ジョブ・クラスのロギング・レベルは LOGGING_FULL で、ジョブは非繰返しジョブです。ジョブは最初に成功した実行の後で再び使用可能になるため、もう 1 回実行されます。その後、停止され、削除されます。

```
SELECT to_char(log_date, 'DD-MON-YY HH24:MM:SS') TIMESTAMP, job_name,
  job_class, operation, status FROM USER_SCHEDULER_JOB_LOG
  WHERE job_name = 'JOB2' ORDER BY log_date;

TIMESTAMP            JOB_NAME  JOB_CLASS  OPERATION  STATUS
-------------------- --------- ---------- ---------- ---------
18-DEC-07 23:10:56   JOB2      CLASS1     CREATE
18-DEC-07 23:12:01   JOB2      CLASS1     UPDATE
18-DEC-07 23:12:31   JOB2      CLASS1     ENABLE
18-DEC-07 23:12:41   JOB2      CLASS1     RUN        SUCCEEDED
18-DEC-07 23:13:12   JOB2      CLASS1     ENABLE
18-DEC-07 23:13:18   JOB2                 RUN        STOPPED
18-DEC-07 23:19:36   JOB2      CLASS1     DROP
```

実行詳細

RUN、RETRY_RUN または RECOVERY_RUN 操作に関する *_SCHEDULER_JOB_LOG 内の行ごとに、*_SCHEDULER_JOB_RUN_DETAILS ビューには対応する行があります。2 つの異なるビューの行は、それぞれの LOG_ID 列で関係付けられています。実行詳細ビューを参照して、ジョブが失敗した理由や停止された理由を判断できます。

```
SELECT to_char(log_date, 'DD-MON-YY HH24:MM:SS') TIMESTAMP, job_name, status,
   SUBSTR(additional_info, 1, 40) ADDITIONAL_INFO
   FROM user_scheduler_job_run_details ORDER BY log_date;

TIMESTAMP            JOB_NAME   STATUS    ADDITIONAL_INFO
-------------------- ---------- --------- ----------------------------------------
18-DEC-07 23:12:41   JOB2       SUCCEEDED
18-DEC-07 23:12:18   JOB2       STOPPED   REASON="Stop job called by user:'SYSTEM'
19-DEC-07 14:12:20   REMOTE_16  FAILED    ORA-29273: HTTP request failed ORA-06512
```

実行詳細ビューには、実際のジョブ開始時刻と継続時間も含まれています。

ジョブおよびジョブ・クラスのロギング・レベルの優先度

ジョブとジョブ・クラスの両方に logging_level 属性があります。この属性に可能な値については、27-17 ページの表 27-2 を参照してください。ジョブ・クラスのデフォルトのロギング・レベルは LOGGING_RUNS で、個別ジョブのデフォルトのレベルは LOGGING_OFF です。ジョブ・クラスのロギング・レベルがクラス内のジョブのロギング・レベルよりも高い場合は、ジョブ・クラスのロギング・レベルが優先されます。したがって、デフォルトでは、すべてのジョブの実行がジョブ・ログに記録されます。

極端に短く頻繁に実行されるジョブが含まれているジョブ・クラスの場合は、各実行ごとの記録のオーバーヘッドが大きすぎるため、ロギングをオフにするか、ジョブが失敗した場合にのみロギングが発生するように設定することもできます。一方で、特定のクラス内のジョブについては発生したすべての操作の完全な監査証跡が必要な場合があります。この場合はそのクラスに対して完全ロギングを有効化する必要があります。

すべてのジョブの監査証跡が確実に作成されるようにする場合は、個々のジョブ作成者がロギングをオフにできないようにする必要があります。これをサポートするために、スケジューラではクラス別のレベルがジョブの情報を記録する最低レベルとなっています。ジョブ作成者は、個々のジョブに対するロギング・レベルを上げることはできますが、下げることはできません。したがって、個々のジョブのロギング・レベルをすべて LOGGING_OFF に設定すると、クラス内のすべてのジョブがクラスでの指定に従って記録されます。

この機能は、デバッグの目的で提供されています。たとえば、クラス別のレベルでジョブ実行を記録するように設定されているときに、ジョブ・レベルでロギングがオフにされた場合でも、スケジューラはジョブの実行を記録します。一方、ジョブ作成者が完全ロギングをオンにしているときに、クラス別のレベルでは実行のみを記録するように設定されている場合は、ジョブの上位のロギング・レベルが優先され、この個別ジョブのすべての操作がログに記録されます。このように、エンド・ユーザーは、完全ロギングをオンにして自分のジョブをテストできます。

個々のジョブのロギング・レベルを設定するには、そのジョブについて SET_ATTRIBUTE プロシージャを使用する必要があります。たとえば、mytestjob というジョブの完全ロギングをオンにするには、次の文を発行します。

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   'mytestjob', 'logging_level', DBMS_SCHEDULER.LOGGING_FULL);
END;
```

ジョブ・クラスのロギング・レベルを設定できるのは、MANAGE SCHEDULER 権限を付与されているユーザーのみです。

> **関連項目：** ジョブ・クラスのロギング・レベルの設定の詳細は、28-9 ページの「ウィンドウ・ログおよびジョブ・ログの監視と管理」を参照してください。

外部ジョブの stdout と stderr の表示

資格証明が割り当てられている外部ジョブでは、stdout と stderr がログ・ファイルに書き込まれます。ローカル外部ジョブでは、ディレクトリ *ORACLE_HOME*/scheduler/log 内のログ・ファイルに書き込まれます。リモート外部ジョブでは、ディレクトリ *AGENT_HOME*/data/log 内のログ・ファイルに書き込まれます。これらのファイルの内容は、`DBMS_SCHEDULER.GET_FILE` で取得できます。ファイル名は、ジョブ・ログ ID とそれに続く文字列 _stdout または _stderr で構成されています。ジョブのジョブ・ログ ID を取得するには、`*_SCHEDULER_JOB_RUN_DETAILS` ビューの `ADDITIONAL_INFO` 列を問い合せて、次のような名前 / 値ペアを解析します。

```
EXTERNAL_LOG_ID="job_71035_3158"
```

たとえば、ファイル名は job_71035_3158_stdout のようになります。stdout の出力の取得方法は、27-9 ページの例 27-6「ローカル外部ジョブの作成と stdout の取得」を参照してください。この例はローカル外部ジョブに関するものですが、メソッドはリモート外部ジョブの場合も同じです。

また、ローカル外部ジョブまたはリモート外部ジョブによって `stderr` に出力が書き込まれる際は、最初の 200 バイトが `*_SCHEDULER_JOB_RUN_DETAILS` ビューの `ADDITIONAL_INFO` 列に記録されます。この情報は、次のような名前 / 値ペアの形式で記録されます。

```
STANDARD_ERROR="text"
```

注意： `ADDITIONAL_INFO` 列には、複数の名前と値のペアを指定できます。順不同のため、`STANDARD_ERROR` の名前と値のペアを検索するには、フィールドを解析する必要があります。

関連項目： `DBMS_SCHEDULER.GET_FILE` の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

プログラムの使用

プログラムは、特定のタスクに関するメタデータの集合です。この項では、プログラムの基本的なタスクについて説明します。この項の内容は、次のとおりです。

- プログラムのタスクとそのプロシージャ
- プログラムの作成
- プログラムの変更
- プログラムの削除
- プログラムの無効化
- プログラムの有効化

関連項目： プログラムの概要については、26-4 ページの「プログラム」を参照してください。

プログラムのタスクとそのプロシージャ

表 27-3 に、プログラムの一般的なタスクとそれに対応するプロシージャおよび権限を示します。

表 27-3 プログラムのタスクとそのプロシージャ

タスク	プロシージャ	必要な権限
プログラムの作成	CREATE_PROGRAM	CREATE JOB または CREATE ANY JOB
プログラムの変更	SET_ATTRIBUTE	ALTER または CREATE ANY JOB、あるいは所有者
プログラムの削除	DROP_PROGRAM	ALTER または CREATE ANY JOB、あるいは所有者
プログラムの使用禁止	DISABLE	ALTER または CREATE ANY JOB、あるいは所有者
プログラムの使用可能化	ENABLE	ALTER または CREATE ANY JOB、あるいは所有者

権限の詳細は、28-31 ページの「スケジューラ権限」を参照してください。

プログラムの作成

プログラムを作成するには、CREATE_PROGRAM プロシージャまたは Enterprise Manager を使用します。デフォルトでは、プログラムは作成者のスキーマ内に作成されます。別のユーザーのスキーマ内にプログラムを作成するには、プログラム名をスキーマ名で修飾する必要があります。他のユーザーがプログラムを使用する場合、そのユーザーにはそのプログラムに対する EXECUTE 権限が必要です。したがって、プログラムを作成した後で、そのプログラムに対する EXECUTE 権限を付与する必要があります。次に、プログラムの作成例を示します。この例では、my_program1 というプログラムが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name           => 'my_program1',
   program_action         => '/usr/local/bin/date',
   program_type           => 'EXECUTABLE',
   comments               => 'My comments here');
END;
/
```

プログラム引数の定義

プログラムを作成した後は、各プログラム引数の名前またはデフォルト値を定義できます。プログラム引数にデフォルト値が定義されていない場合は、プログラムを参照するジョブで引数値を指定する必要があります（ジョブでデフォルト値を上書きすることもできます）。ジョブを使用可能にするには、すべての引数値を定義する必要があります。

プログラム引数値を設定するには、DEFINE_PROGRAM_ARGUMENT または DEFINE_ANYDATA_ARGUMENT プロシージャを使用します。DEFINE_ANYDATA_ARGUMENT は、ANYDATA オブジェクト内にカプセル化する必要がある複雑な型に使用されます。引数を必要とする場合があるプログラムの例の 1 つに、開始日と終了日が必要なレポート作成プログラムを開始するジョブがあります。次のコード例では、レポート作成プログラムの 2 番目の引数である終了日の引数が設定されます。この例では、SET_JOB_ANYDATA_VALUE や SET_JOB_ARGUMENT_VALUE などの他のパッケージ・プロシージャから（位置ではなく）名前で引数を参照できるように、引数に名前が割り当てられています。

```
BEGIN
DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT (
   program_name           => 'operations_reporting',
   argument_position      => 2,
   argument_name          => 'end_date',
   argument_type          => 'VARCHAR2',
   default_value          => '12-DEC-03');
END;
/
```

プログラム引数は、名前または位置で削除できます。次に例を示します。

```
BEGIN
DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (
   program_name            => 'operations_reporting',
   argument_position       => 2);

DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (
   program_name            => 'operations_reporting',
   argument_name           => 'end_date');
END;
/
```

一部の特殊なケースでは、プログラムのロジックがスケジューラの環境によって異なる場合があります。この目的のために、スケジューラには、プログラムに引数として渡すことができる事前定義のメタデータ引数がいくつかあります。たとえば、スケジュールがウィンドウ名である一部のジョブについては、ジョブの開始時にウィンドウがオープンしている時間の長さを認識していると便利です。これは、ウィンドウ終了時間をプログラムに対するメタデータ引数として定義すると可能です。

特定ジョブのメタデータに対するアクセス権限がプログラムに必要な場合は、プログラムの実行時にスケジューラによって値が指定されるように、DEFINE_METADATA_ARGUMENT プロシージャを使用して特別なメタデータ引数を定義できます。

> **関連項目：** DEFINE_PROGRAM_ARGUMENT、DEFINE_ANYDATA_ARGUMENT および DEFINE_METADATA_ARGUMENT プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

プログラムの変更

プログラムを変更するには、Enterprise Manager または DBMS_SCHEDULER.SET_ATTRIBUTE および DBMS_SCHEDULER.SET_ATTRIBUTE_NULL パッケージ・プロシージャを使用します。DBMS_SCHEDULER パッケージ・プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

次に、Enterprise Manager を使用したプログラムの変更方法を示します。

1. データベース・ホームページにアクセスします。
2. ページの上部にある**「サーバー」**をクリックして「サーバー」ページを表示します。
3. 「Oracle Scheduler」セクションで、**「プログラム」**をクリックします。

   「スケジューラのプログラム」ページが表示されます。このページには既存のプログラムが表示されます。
4. プログラムを選択し、**「編集」**をクリックします。

   「プログラムの編集」ページが表示されます。
5. 「有効」ヘッダーの横にある**「はい」**または**「いいえ」**を選択します。
6. 「説明」フィールドで、コメントを変更します。
7. 「タイプ」ドロップダウン・リストから、次のいずれかを選択します。

   - **PLSQL_BLOCK**

     「ソース」フィールドが表示されます。このフィールドで PL/SQL コードを入力または変更します。

   - **STORED_PROCEDURE**

     「プロシージャ名」フィールドが表示されます。フィールドにストアド・プロシージャ名が含まれている場合は、**「プロシージャの表示」**をクリックするとストアド・プロシージャを表示または編集できます。フィールドが空白の場合、またはストアド・プロシージャを変更する場合は、**「プロシージャの選択」**をクリックします。「プロシー

ジャの選択」ページが表示されます。ストアド・プロシージャを選択して**「選択」**をクリックすると、「プログラムの編集」ページに戻ります（「プロシージャの選択」ページの使用方法に関するヘルプは、ページの上部にある**「ヘルプ」**をクリックしてください）。

プロシージャ名を選択すると、「プログラムの編集」ページの「引数」ヘッダーの下に、引数のリストが表示されます。必要に応じて、1 つ以上の引数のデフォルト値を入力します。

- **EXECUTABLE**

  「実行可能な名前」フィールドが表示されます。実行可能ファイルのフルパスを入力します。「引数」ヘッダーの下にある引数を編集または削除するか、**「行の追加」**をクリックして引数を追加します。

8. **「適用」**をクリックして変更内容を保存します。

変更したプログラムが、現在実行中のジョブで使用されている場合、そのジョブは変更操作前に定義されたプログラムで引き続き実行されます。

プログラムの削除

1 つ以上のプログラムを削除するには、DROP_PROGRAM プロシージャまたは Enterprise Manager を使用します。

削除されたプログラムを指し示している実行中のジョブは DROP_PROGRAM コールの影響を受けず、実行を継続できます。プログラムが削除されると、そのプログラムに関係する引数もすべて削除されます。プログラム名のカンマ区切りのリストを指定することで、1 回のコールで複数のプログラムを削除できます。たとえば、次の文では 3 つのプログラムが削除されます。

```
BEGIN
DBMS_SCHEDULER.DROP_PROGRAM('program1, program2, program3');
END;
/
```

DROP_PROGRAM プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

プログラムの無効化

1 つ以上のプログラムを無効（使用禁止）にするには、DISABLE プロシージャまたは Enterprise Manager を使用します。プログラムが使用禁止になると、ステータスが disabled に変更されます。使用禁止プログラムとは、メタデータは存在しているが、このプログラムを指し示すジョブは実行できないことを意味します。

使用禁止のプログラムを指し示している実行中のジョブは DISABLE コールの影響を受けず、実行を継続できます。プログラムが使用禁止になった場合でも、そのプログラムに関係する引数は影響を受けません。

プログラムは他の理由で使用禁止になる場合もあります。たとえば、プログラム引数が削除された場合や number_of_arguments の変更ですべての引数を定義できなくなった場合などです。

DISABLE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

プログラムの有効化

1 つ以上のプログラムを有効（使用可能）にするには、ENABLE プロシージャまたは Enterprise Manager を使用します。プログラムを使用可能になると、使用可能フラグが TRUE に設定されます。プログラムは、デフォルトで使用禁止で作成されるため、そのプログラムを指し示すジョブを使用可能にするには、プログラムを使用可能にする必要があります。プログラムが使用可能になる前に、処理が有効であること、およびすべての引数が定義されていることを確認するために妥当性チェックが実行されます。

ENABLE プロシージャ・コールにプログラム名のカンマ区切りのリストを指定することで、1 回のコールで複数のプログラムを使用可能にできます。たとえば、次の文では 3 つのプログラムが使用可能になります。

```
BEGIN
DBMS_SCHEDULER.ENABLE('program1, program2, program3');
END;
/
```

ENABLE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

スケジュールの使用

スケジュールは、ジョブの実行時期またはウィンドウのオープン時期を定義します。スケジュールは、データベースにオブジェクトとして作成および保存することによってユーザー間で共有できます。

この項では、スケジュールの基本的なタスクについて説明します。この項の内容は、次のとおりです。

- スケジュールのタスクとそのプロシージャ
- スケジュールの作成
- スケジュールの変更
- スケジュールの削除
- 繰返し間隔の設定

> **関連項目：** スケジュールの概要については、26-5 ページの「スケジュール」を参照してください。

スケジュールのタスクとそのプロシージャ

表 27-4 に、スケジュールの一般的なタスクとその処理に使用するプロシージャを示します。

表 27-4 スケジュールのタスクとそのプロシージャ

タスク	プロシージャ	必要な権限
スケジュールの作成	CREATE_SCHEDULE	CREATE JOB または CREATE ANY JOB
スケジュールの変更	SET_ATTRIBUTE	ALTER または CREATE ANY JOB、あるいは所有者
スケジュールの削除	DROP_SCHEDULE	ALTER または CREATE ANY JOB、あるいは所有者

権限の詳細は、28-31 ページの「スケジューラ権限」を参照してください。

スケジュールの作成

スケジュールを作成するには、CREATE_SCHEDULE プロシージャまたは Enterprise Manager を使用します。スケジュールは、そのスケジュールを作成するユーザーのスキーマ内に作成され、最初に作成した時点で使用可能です。また、別のユーザーのスキーマ内に作成できます。作成したスケジュールは、他のユーザーが使用できます。スケジュールは PUBLIC に対するアクセス権限付きで作成されます。したがって、スケジュールに対するアクセス権限を明示的に付与する必要はありません。次に、スケジュールの作成例を示します。

```
BEGIN
DBMS_SCHEDULER.CREATE_SCHEDULE (
  schedule_name     => 'my_stats_schedule',
  start_date        => SYSTIMESTAMP,
  end_date          => SYSTIMESTAMP + INTERVAL '30' day,
  repeat_interval   => 'FREQ=HOURLY; INTERVAL=4',
  comments          => 'Every 4 hours');
END;
/
```

CREATE_SCHEDULE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

スケジュールの変更

スケジュールを変更するには、SET_ATTRIBUTE プロシージャまたは Enterprise Manager を使用します。スケジュールを変更すると、スケジュールの定義が変更されます。スケジュール名以外のすべての属性を変更できます。スケジュールの属性は、*_SCHEDULER_SCHEDULES ビューで使用可能です。

スケジュールを変更しても、このスケジュールを使用している実行中のジョブとオープン中のウィンドウに影響を与えることはありません。変更が有効になるのは、次回のジョブ実行時またはウィンドウ・オープン時からです。

SET_ATTRIBUTE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

スケジュールの削除

スケジュールを削除するには、DROP_SCHEDULE プロシージャまたは Enterprise Manager を使用します。このプロシージャ・コールによって、スケジュール・オブジェクトがデータベースから削除されます。

DROP_SCHEDULE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

繰返し間隔の設定

ジョブの繰返し時期および間隔を制御するには、ジョブ自体またはジョブが参照する名前付きスケジュールの repeat_interval 属性を設定します。repeat_interval は、DBMS_SCHEDULER パッケージまたは Enterprise Manager で設定できます。

repeat_interval の評価結果は一連のタイムスタンプです。スケジューラは各タイムスタンプの日時にジョブを実行します。ジョブやスケジュールからの開始日も、タイムスタンプの結果セットを決定する要因となることに注意してください（repeat_interval の評価の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください）。repeat_interval に値が指定されていない場合、ジョブは指定した開始日に 1 回のみ実行されます。

ジョブの開始直後に repeat_interval が評価され、スケジュールされた次回のジョブ実行時間が判別されます。ジョブがまだ実行されている間に、スケジュールされた次回のジョブ実行時間に達する可能性があります。しかし、ジョブの新規インスタンスは、現在のジョブが完了するまでは開始されません。

繰返し間隔を指定する方法は、次の 2 通りあります。

- スケジューラのカレンダ指定構文の使用方法
- PL/SQL 式の使用方法

スケジューラのカレンダ指定構文の使用方法

ジョブの繰返し間隔を設定する主要な方法は、スケジューラのカレンダ指定式を使用して repeat_interval 属性を設定する方法です。repeat_interval のカレンダ指定構文および CREATE_SCHEDULE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

カレンダ指定式の例

次に、簡単な繰返し間隔の例を示します。わかりやすくするために、開始日は評価結果に影響を与えないと仮定します。

毎週金曜日に実行（3 つの例はすべて等価です）

```
FREQ=DAILY; BYDAY=FRI;
FREQ=WEEKLY; BYDAY=FRI;
FREQ=YEARLY; BYDAY=FRI;
```

隔週金曜日に実行

```
FREQ=WEEKLY; INTERVAL=2; BYDAY=FRI;
```

毎月末に実行

```
FREQ=MONTHLY; BYMONTHDAY=-1;
```

毎月末の翌日に実行

```
FREQ=MONTHLY; BYMONTHDAY=-2;
```

3 月 10 日に実行（2 つの例は等価です）

```
FREQ=YEARLY; BYMONTH=MAR; BYMONTHDAY=10;
FREQ=YEARLY; BYDATE=0310;
```

10 日おきに実行

```
FREQ=DAILY; INTERVAL=10;
```

毎日午後 4 時、5 時および 6 時に実行

```
FREQ=DAILY; BYHOUR=16,17,18;
```

隔月 15 日に実行

```
FREQ=MONTHLY; INTERVAL=2; BYMONTHDAY=15;
```

毎月 29 日に実行

```
FREQ=MONTHLY; BYMONTHDAY=29;
```

毎月第 2 水曜日に実行

```
FREQ=MONTHLY; BYDAY=2WED;
```

毎年最終金曜日に実行

```
FREQ=YEARLY; BYDAY=-1FRI;
```

50 時間おきに実行

```
FREQ=HOURLY; INTERVAL=50;
```

隔月末に実行

```
FREQ=MONTHLY; INTERVAL=2; BYMONTHDAY=-1;
```

毎月初めの 3 日間 1 時間おきに実行

```
FREQ=HOURLY; BYMONTHDAY=1,2,3;
```

以降は、多少複雑な繰返し間隔の例です。

毎月最後の稼働日に実行（稼働日は月曜日～金曜日と仮定）

```
FREQ=MONTHLY; BYDAY=MON,TUE,WED,THU,FRI; BYSETPOS=-1
```

会社の休業日を除いて、毎月最後の稼働日に実行（この例では、既存の名前付きスケジュール Company_Holidays を参照）

```
FREQ=MONTHLY; BYDAY=MON,TUE,WED,THU,FRI; EXCLUDE=Company_Holidays; BYSETPOS=-1
```

毎週金曜日および会社の休業日の正午に実行

```
FREQ=YEARLY;BYDAY=FRI;BYHOUR=12;INCLUDE=Company_Holidays
```

米国独立記念日、戦没将兵記念日、労働者の日の 3 つの休日に実行（この例では、それぞれ休日に対応する日を定義する既存の 3 つの名前付きスケジュール JUL4、MEM および LAB を参照）

```
JUL4,MEM,LAB
```

カレンダ指定式評価の例

繰返し間隔を "FREQ=MINUTELY;INTERVAL=2;BYHOUR=17; BYMINUTE=2,4,5,50,51,7;" に、開始日を 28-FEB-2004 23:00:00 に設定すると、次のスケジュールが生成されます。

```
SUN 29-FEB-2004 17:02:00
SUN 29-FEB-2004 17:04:00
SUN 29-FEB-2004 17:50:00
MON 01-MAR-2004 17:02:00
MON 01-MAR-2004 17:04:00
MON 01-MAR-2004 17:50:00
...
```

繰返し間隔を "FREQ=MONTHLY;BYMONTHDAY=15, -1" に、開始日を 29-DEC-2003 9:00:00 に設定すると、次のスケジュールが生成されます。

```
WED 31-DEC-2003 09:00:00
THU 15-JAN-2004 09:00:00
SAT 31-JAN-2004 09:00:00
SUN 15-FEB-2004 09:00:00
SUN 29-FEB-2004 09:00:00
MON 15-MAR-2004 09:00:00
WED 31-MAR-2004 09:00:00
...
```

繰返し間隔を "FREQ=MONTHLY;" に、開始日を 29-DEC-2003 9:00:00 に設定すると、次のスケジュールが生成されます（BYMONTHDAY 句がないため、日付指定（月）が開始日から取得されることに注意してください）。

```
MON 29-DEC-2003 09:00:00
THU 29-JAN-2004 09:00:00
SUN 29-FEB-2004 09:00:00
MON 29-MAR-2004 09:00:00
...
```

カレンダ指定式の使用例

カレンダ指定構文を使用した次の例について考えてみます。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name             => 'scott.my_job1',
   start_date           => '15-JUL-04 01.00.00 AM Europe/Warsaw',
   repeat_interval      => 'FREQ=MINUTELY; INTERVAL=30;',
   end_date             => '15-SEP-04 01.00.00 AM Europe/Warsaw',
   comments             => 'My comments here');
END;
/
```

この文では、scott 内に my_job1 が作成されます。このジョブの開始日は 7 月 15 日、終了日は 9 月 15 日で、30 分おきに実行されます。

PL/SQL 式の使用方法

カレンダ指定構文より複雑な機能が必要な場合は、PL/SQL 式を使用できます。ただし、PL/SQL 式はウィンドウまたは名前付きスケジュールでは使用できません。また、日付またはタイムスタンプに評価される必要があります。これ以外に制限はないため、適切なプログラミングによって、あらゆる繰返し間隔を作成できます。次の例について考えてみます。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name             => 'scott.my_job2',
   start_date           => '15-JUL-04 01.00.00 AM Europe/Warsaw',
   repeat_interval      => 'SYSTIMESTAMP + INTERVAL '30' MINUTE',
   end_date             => '15-SEP-04 01.00.00 AM Europe/Warsaw',
   comments             => 'My comments here');
END;
/
```

この文では、scott 内に my_job1 が作成されます。このジョブの開始日は 7 月 15 日で、30 分おきに実行され 9 月 15 日に終了します。ジョブが 30 分おきに実行されるのは、repeat_interval が SYSTIMESTAMP + INTERVAL '30' MINUTE に設定されており、この式では 30 分後が戻されるためです。

PL/SQL 式とカレンダ指定構文の動作の相違点

カレンダ指定式と PL/SQL 繰返し間隔の動作には、次のような重要な相違があります。

- 開始日

  カレンダ指定構文を使用した場合、開始日は参照日のみです。つまり、スケジュールはこの日付から有効になります。これは、ジョブが開始日に開始されることを意味しません。

  PL/SQL 式を使用した場合、開始日はジョブが最初に実行を開始した実際の時間を表します。

- 次回の実行時期

  カレンダ指定構文を使用した場合、次回のジョブ実行時期は固定されています。

  PL/SQL 式を使用した場合、次回のジョブ実行時期は現行ジョブ実行の実際の開始時間によって異なります。この違いを示す例を考えてみます。あるジョブを午後 2 時に開始し、2 時間おきに繰り返すスケジュールの場合、繰返し間隔をカレンダ指定構文で指定すると、ジョブは 4 時、6 時というように繰り返され、以降も同様に繰り返されます。PL/SQL を指定してジョブを午後 2 時 10 分に開始した場合、そのジョブは午後 4 時 10 分に繰り返されます。次のジョブが実際には 4 時 11 分に開始された場合、その次のジョブ実行は 6 時 11 分になります。

この 2 つの違いを示すために、次の状況を考えてみます。開始日は 15-July-2003 1:45:00、繰返しは 2 時間おきとします。カレンダ式 "FREQ=HOURLY; INTERVAL=2; BYMINUTE=0;" では、次のスケジュールが生成されます。

```
TUE 15-JUL-2003  03:00:00
TUE 15-JUL-2003  05:00:00
TUE 15-JUL-2003  07:00:00
TUE 15-JUL-2003  09:00:00
TUE 15-JUL-2003  11:00:00
...
```

カレンダ式では、2 時間おきの毎正時に繰返しを実行します。

一方、PL/SQL 式 "SYSTIMESTAMP + interval '2' hour" では、実行時間は次のようになります。

```
TUE 15-JUL-2003  01:45:00
TUE 15-JUL-2003  03:45:05
TUE 15-JUL-2003  05:45:09
TUE 15-JUL-2003  07:45:14
TUE 15-JUL-2003  09:45:20
...
```

繰返し間隔と夏時間

ジョブの繰返しでは、次回のジョブ実行時間のスケジュールは、タイム・ゾーン列のあるタイムスタンプに格納されます。カレンダ指定構文を使用する場合、タイム・ゾーンは start_date から取り出されます。start_date が設定されていない場合の操作の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

PL/SQL 式に基づいた繰返し間隔の場合、タイム・ゾーンは PL/SQL 式が返すタイムスタンプの一部になっています。いずれの場合も、リージョン名を使用することが重要です。たとえば、"+2:00" のように絶対タイム・ゾーン・オフセットを指定するのではなく、"Europe/Istanbul" のように指定します。タイム・ゾーンがリージョン名で指定されている場合のみ、スケジューラはそのリージョンに適用される夏時間調整に従います。

ジョブ・クラスの使用

ジョブ・クラスを使用すると、リソース割当ておよび優先度付けの目的でジョブをグループ化でき、一連の属性値をメンバーのジョブに簡単に割り当てることができます。

データベースとともに作成されるデフォルトのジョブ・クラスがあります。ジョブ・クラスを指定せずにジョブを作成すると、ジョブはこのデフォルトのジョブ・クラス（DEFAULT_JOB_CLASS）に割り当てられます。デフォルトのジョブ・クラスでは、EXECUTE 権限が PUBLIC に付与されているため、ジョブの作成権限を持つすべてのデータベース・ユーザーは、デフォルトのジョブ・クラスにジョブを作成できます。

この項では、ジョブ・クラスの基本的なタスクについて説明します。この項の内容は、次のとおりです。

- ジョブ・クラスのタスクとそのプロシージャ
- ジョブ・クラスの作成
- ジョブ・クラスの変更
- ジョブ・クラスの削除

> **関連項目：** ジョブ・クラスの概要については、26-13 ページの「ジョブ・クラス」を参照してください。

ジョブ・クラスのタスクとそのプロシージャ

表 27-5 に、ジョブ・クラスの一般的なタスクとそれに対応するプロシージャと権限を示します。

表 27-5 ジョブ・クラスのタスクとそのプロシージャ

タスク	プロシージャ	必要な権限
ジョブ・クラスの作成	CREATE_JOB_CLASS	MANAGE SCHEDULER
ジョブ・クラスの変更	SET_ATTRIBUTE	MANAGE SCHEDULER
ジョブ・クラスの削除	DROP_JOB_CLASS	MANAGE SCHEDULER

権限の詳細は、28-31 ページの「スケジューラ権限」を参照してください。

ジョブ・クラスの作成

ジョブ・クラスを作成するには、CREATE_JOB_CLASS プロシージャまたは Enterprise Manager を使用します。たとえば、次の文では、すべての財務ジョブ用のジョブ・クラスが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name             =>  'finance_jobs',
   resource_consumer_group    =>  'finance_group');
END;
/
```

ジョブ・クラスを問い合せるには、*_SCHEDULER_JOB_CLASSES ビューを使用します。

SET_ATTRIBUTE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。また、ジョブ・クラスの作成例は、28-2 ページの「Oracle Scheduler の構成」を参照してください。

ジョブ・クラスの変更

ジョブ・クラスを変更するには、SET_ATTRIBUTE プロシージャまたは Enterprise Manager を使用します。また、ジョブ・クラス名以外のすべての属性を変更できます。ジョブ・クラスの属性は、*_SCHEDULER_JOB_CLASSES ビューで使用可能です。

ジョブ・クラスを変更しても、そのクラスに属する実行中のジョブは影響を受けません。変更は、まだ実行が開始されていないジョブに対してのみ有効です。

SET_ATTRIBUTE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。また、28-2 ページの「Oracle Scheduler の構成」を参照してください。

ジョブ・クラスの削除

1 つ以上のジョブ・クラスを削除するには、DROP_JOB_CLASS プロシージャまたは Enterprise Manager を使用します。ジョブ・クラスを削除すると、そのジョブ・クラスに関するすべてのメタデータがデータベースから削除されます。

DROP_JOB_CLASS プロシージャ・コールにジョブ・クラス名のカンマ区切りのリストを指定することで、1 回のコールで複数のジョブ・クラスを削除できます。たとえば、次の文では 3 つのジョブ・クラスが削除されます。

```
BEGIN
DBMS_SCHEDULER.DROP_JOB_CLASS('jobclass1, jobclass2, jobclass3');
END;
/
```

DROP_JOB_CLASS プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ウィンドウの使用

ウィンドウを使用すると、異なる時間帯で別々のリソース・プランを自動的にアクティブにできます。ジョブを実行すると、そのジョブに割り当てられているリソースが、リソース・プランの変更時に変わることがわかります。

ウィンドウの主な属性は次のとおりです。

- スケジュール

  ウィンドウが有効である時期を制御します。

- 継続時間

  ウィンドウがオープンしている長さを制御します。

- リソース・プラン

  ウィンドウのオープン時にアクティブ化されるリソース・プランの名前を設定します。

特定の時間に有効にできるウィンドウは 1 つのみです。ウィンドウは SYS スキーマに属します。

すべてのウィンドウ・アクティビティは、*_SCHEDULER_WINDOW_LOG ビュー（**ウィンドウ・ログ**と呼ばれます）に記録されます。ウィンドウ・ログの記録例は、28-9 ページの「ウィンドウ・ログ」を参照してください。

この項では、ウィンドウの基本的なタスクについて説明します。この項の内容は、次のとおりです。

> **関連項目：** ウィンドウの概要については、26-15 ページの「ウィンドウ」を参照してください。

ウィンドウのタスクとそのプロシージャ

表 27-6 に、ウィンドウの一般的なタスクとその処理に使用するプロシージャを示します。

表 27-6 ウィンドウのタスクとそのプロシージャ

タスク	プロシージャ	必要な権限
ウィンドウの作成	CREATE_WINDOW	MANAGE SCHEDULER
ウィンドウのオープン	OPEN_WINDOW	MANAGE SCHEDULER
ウィンドウのクローズ	CLOSE_WINDOW	MANAGE SCHEDULER
ウィンドウの変更	SET_ATTRIBUTE	MANAGE SCHEDULER
ウィンドウの削除	DROP_WINDOW	MANAGE SCHEDULER
ウィンドウの使用禁止	DISABLE	MANAGE SCHEDULER
ウィンドウの使用可能化	ENABLE	MANAGE SCHEDULER

権限の詳細は、28-31 ページの「スケジューラ権限」を参照してください。

ウィンドウの作成

ウィンドウを作成するには、Enterprise Manager または DBMS_SCHEDULER.CREATE_WINDOW パッケージ・プロシージャを使用します。これらの作成方法には、一方は PL/SQL を使用し、もう一方は GUI を使用すること以外に、異なる点が 1 つあります。パッケージ・プロシージャを使用する場合は、resource_plan パラメータを NULL のままにできることです。この場合は、ウィンドウのオープン時に現行のプランが有効のままになります。詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』および 28-2 ページの「Oracle Scheduler の構成」を参照してください。

Enterprise Manager を使用してウィンドウを作成する手順は、次のとおりです。

1. データベース・ホームページにアクセスします。
2. ページの上部にある**「サーバー」**をクリックして「サーバー」ページを表示します。
3. 「Oracle Scheduler」セクションで、**「ウィンドウ」**をクリックします。

   「スケジューラのウィンドウ」ページが表示されます。このページには既存のウィンドウが表示されます。
4. **「作成」**をクリックします。

   「ウィンドウの作成」ページが表示されます。
5. ウィンドウ名を入力します。
6. 「リソース・プラン」ドロップダウン・リストからリソース・プランを選択するか、リソース・プランを作成します。

   デフォルトの INTERNAL_PLAN を使用できます。既存のリソース・プランの内容を表示するには、**「リソース・プランの表示」**をクリックします。新規リソース・プランを作成するには、**「リソース・プランの作成」**をクリックしてその作成手順に従います。
7. **「低」**または**「高」**の優先度を選択します。
8. 必要に応じて、**「説明」**フィールドにコメントを入力します。
9. 「スケジュール」ヘッダーの下で、次のいずれかを実行します。
   - スケジュールを作成する場合は、**「カレンダの使用」**を選択します。
   - 既存のスケジュールを使用する場合は、**「既存スケジュールの使用」**を選択します。スケジュールとその情報が表示されます。別のスケジュールを使用する場合は、**「スケジュールの選択」**をクリックします。この場合は「スケジュールの選択」ページが表示されます。「結果」ヘッダーの下に表示されたスケジュールのいずれかを選択するか、スキーマを指定して「スケジュール名」にテキスト文字列を入力し、**「実行」**をクリックします。名前に検索文字列が含まれたスケジュールが返されます。使用するスケジュールを選択し、**「選択」**をクリックします。

     注意： スケジューラでは、スケジュールに対して定義されたウィンドウがすでに存在するかどうかはチェックされません。したがって、複数のウィンドウが重複する場合があります。また、繰返し間隔として PL/SQL 式が設定されている名前付きスケジュールの使用は、ウィンドウに対してはサポートされていません。

     「OK」をクリックし、残りの手順をスキップします。
10. タイム・ゾーンを変更する場合は、「タイムゾーン」フィールドの横にある懐中電灯アイコンをクリックし、その変更手順に従います。
11. 「繰返し」ドロップダウン・リストの下で、ウィンドウの繰返し間隔を選択します。**「繰返しなし」**以外の値を選択すると、間隔および開始時間を入力できるページに変わります。
12. **「開始」**ヘッダーの下で、スケジュールの開始を**「即時」**または**「後で」**から選択します。**「後で」**を選択した場合は日付を入力します。

13. **「期間」**ヘッダーの下で、ウィンドウのオープン期間を入力します。
14. **「OK」**をクリックしてウィンドウを保存します。

ウィンドウの変更

ウィンドウを変更するには、SET_ATTRIBUTE プロシージャまたは Enterprise Manager を使用します。変更するときは、WINDOW_NAME 以外のすべてのウィンドウ属性を変更できます。ウィンドウの属性は、*_SCHEDULER_WINDOWS ビューで使用可能です。

ウィンドウを変更しても、アクティブなウィンドウには影響を与えません。変更は、次回ウィンドウがオープンしたときに有効になります。

すべてのウィンドウを変更できます。使用禁止のウィンドウを変更した場合は、変更した後も使用禁止のままです。使用可能なウィンドウは自動的に使用禁止になり、変更後、使用可能化プロセスで実行される妥当性チェックが成功した場合は再び使用可能になります。

SET_ATTRIBUTE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。また、28-2 ページの「Oracle Scheduler の構成」を参照してください。

ウィンドウのオープン

ウィンドウがオープンすると、スケジューラは、ウィンドウの作成時に関連付けられたリソース・プランに切り替えます。ウィンドウのオープン時に実行中のジョブがある場合、そのジョブに割り当てられているリソースはリソース・プランの切替えによって変更される場合があります。

ウィンドウをオープンする方法は、次の 2 通りあります。

- ウィンドウのスケジュールに基づいたオープン
- OPEN_WINDOW プロシージャを使用した手動オープン

  このプロシージャは、スケジュールとは関係なくウィンドウをオープンします。ウィンドウがオープンし、関連付けられたリソース・プランが即時に有効になります。手動でオープンできるのは、使用可能なウィンドウのみです。

  OPEN_WINDOW プロシージャで、duration 属性を使用して、ウィンドウがオープンしている時間の長さを指定できます。継続時間の型は INTERVAL DAY TO SECOND です。継続時間の指定がない場合、ウィンドウはそのウィンドウに保存されている通常の継続時間の間オープン状態です。

  ウィンドウを手動でオープンしても、スケジュールされた通常のウィンドウの実行には影響を与えません。

  手動でオープンしたウィンドウがクローズすると、その時点でオープン対象のウィンドウが他にある場合は、重複ウィンドウのルールが適用され、オープンするウィンドウが判別されます。

  すでにオープンしているウィンドウがある場合でも、OPEN_WINDOW コールまたは Enterprise Manager で force オプションを TRUE に設定すると、ウィンドウを強制的にオープンできます。

  force オプションが TRUE に設定されているときは、その時点でオープンしているウィンドウの優先度の方が高い場合でも、スケジューラはウィンドウを自動的にクローズします。手動でオープンしたウィンドウの継続時間中は、より優先度の高いウィンドウが他にスケジュールされていても、スケジューラはそのウィンドウをオープンしません。すでにオープンしているウィンドウもオープンできます。この場合、ウィンドウは、OPEN_WINDOW コマンドが発行された時点から、コールで指定された継続時間の間オープンしたままになります。

  この状態を示す例について考えてみます。window1 が継続時間 4 時間で作成されたとします。現在は、オープンから 2 時間が経過しています。この時点で OPEN_WINDOW コールを使用して window1 を再オープンしても、継続時間を指定しない場合、window1 は、さらに 4 時間オープンしたままになります。これは、以前からこの継続時間で作成されているためです。継続時間に 30 分を指定した場合、ウィンドウは 30 分後にクローズします。

ウィンドウがオープンすると、ウィンドウ・ログにエントリが作成されます。

現在のリソース・プランが、ALTER SYSTEM 文 FORCE オプションあるいは DBMS_RESOURCE_MANAGER.SWITCH_PLAN パッケージ・プロシージャの allow_scheduler_plan_switches 引数 FALSE を使用して、手動で切り替えられたものである場合、ウィンドウでリソース・プランの切替えに失敗する可能性があります。この場合、リソース・プランの切替え失敗がウィンドウ・ログに書き込まれます。

OPEN_WINDOW プロシージャおよび DBMS_RESOURCE_MANAGER.SWITCH_PLAN プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ウィンドウのクローズ

ウィンドウをクローズする方法は、次の 2 通りあります。

- スケジュールに基づいたクローズ

  ウィンドウは作成時に定義されたスケジュールに基づいてクローズします。

- CLOSE_WINDOW プロシージャを使用した手動クローズ

  CLOSE_WINDOW プロシージャは、オープン中のウィンドウを期間終了前にクローズします。

クローズしたウィンドウは、そのウィンドウが有効ではないことを意味します。ウィンドウがクローズすると、スケジューラは、リソース・プランをそのウィンドウ期間外で有効だったリソース・プランに切り替えるか、またはウィンドウの重複がある場合は別のウィンドウに切り替えます。存在しないウィンドウやオープンしていないウィンドウをクローズしようとすると、エラーが生成されます。

実行中のジョブは、そのジョブが実行されているウィンドウがクローズした場合でも、stop_on_window_close 属性がジョブの作成時に TRUE に設定されていないかぎりクローズしません。ただし、リソース・プランが変更される場合があるため、ジョブに割り当てられているリソースは変更される可能性があります。

実行中のジョブにスケジュールとしてウィンドウ・グループが設定されていると、そのジョブは、同じウィンドウ・グループのメンバーである別のウィンドウが元のウィンドウのクローズ時にアクティブになった場合、停止しません。これは、stop_on_window_close 属性が TRUE に設定された状態でジョブが作成されている場合でも同じです。

ウィンドウがクローズ状態になると、ウィンドウ・ログ DBA_SCHEDULER_WINDOW_LOG にエントリが追加されます。

CLOSE_WINDOW プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ウィンドウの削除

1 つ以上のウィンドウを削除するには、DROP_WINDOW プロシージャまたは Enterprise Manager を使用します。ウィンドウが削除されると、そのウィンドウに関するすべてのメタデータが *_SCHEDULER_WINDOWS ビューから削除されます。また、ウィンドウに対するすべての参照がウィンドウ・グループから削除されます。

DROP_WINDOW プロシージャにウィンドウ名またはウィンドウ・グループ名のカンマ区切りのリストを指定すると、1 回のコールで複数のウィンドウを削除できます。たとえば、次の文ではウィンドウとウィンドウ・グループの両方が削除されます。

```
BEGIN
DBMS_SCHEDULER.DROP_WINDOW ('window1, window2,
  window3, windowgroup1, windowgroup2');
END;
/
```

ウィンドウ・グループ名を指定した場合、ウィンドウ・グループ内のウィンドウは削除されますが、ウィンドウ・グループは削除されません。ウィンドウ・グループを削除するには、DROP_WINDOW_GROUP プロシージャを使用する必要があります。

DROP_WINDOW プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ウィンドウの無効化

1 つ以上のウィンドウを無効（使用禁止）にするには、DISABLE プロシージャまたは Enterprise Manager を使用します。使用禁止のウィンドウはオープンしませんが、ウィンドウのメタデータは存在しているため、再び使用可能にできます。DISABLE プロシージャは複数のスケジューラ・オブジェクトに対して使用されるため、ウィンドウを使用禁止にするときは、先頭に SYS を付ける必要があります。

ウィンドウは他の理由で使用禁止になる場合もあります。たとえば、ウィンドウはそのスケジュールの終了時に使用禁止になります。また、存在しないスケジュールをウィンドウが指し示している場合も使用禁止になります。

スケジュールとしてウィンドウが設定されているジョブがある場合、そのウィンドウは、プロシージャ・コールで force を TRUE に設定しないかぎり、使用禁止にできません。デフォルトでは、force は FALSE に設定されます。ウィンドウが使用禁止の場合でも、スケジュールとしてそのウィンドウが設定されているジョブは使用禁止になりません。

DISABLE プロシージャ・コールにウィンドウ名またはウィンドウ・グループ名のカンマ区切りのリストを指定することで、1 回のコールで複数のウィンドウを使用禁止にすることもできます。たとえば、次の文ではウィンドウとウィンドウ・グループの両方が使用禁止になります。

```
BEGIN
DBMS_SCHEDULER.DISABLE ('sys.window1, sys.window2,
   sys.window3, sys.windowgroup1, sys.windowgroup2');
END;
/
```

DISABLE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ウィンドウの有効化

1 つ以上のウィンドウを有効（使用可能）にするには、ENABLE プロシージャまたは Enterprise Manager を使用します。使用可能なウィンドウとは、オープン可能なウィンドウのことです。ウィンドウは、デフォルトで enabled で作成されます。ENABLE プロシージャを使用してウィンドウを使用可能にすると、妥当性チェックが実行され、このチェックが成功した場合のみウィンドウは使用可能になります。ウィンドウが使用可能になると、ウィンドウ・ログ表にログが記録されます。ENABLE プロシージャは複数のスケジューラ・オブジェクトに対して使用されるため、ウィンドウを使用可能にするときは、先頭に SYS を付ける必要があります。

ウィンドウ名のカンマ区切りのリストを指定することで、1 回のコールで複数のウィンドウを使用可能にできます。たとえば、次の文では 3 つのウィンドウが使用可能になります。

```
BEGIN
DBMS_SCHEDULER.ENABLE ('sys.window1, sys.window2, sys.window3');
END;
/
```

ENABLE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ウィンドウの重複

お薦めする方法ではありませんが、ウィンドウは重複して設定できます。1 度にアクティブにできるウィンドウは 1 つのみであるため、ウィンドウが重複したときにどのウィンドウをアクティブにするかを判別するために、次のルールが使用されます。

- 同じ優先度のウィンドウが重複している場合は、アクティブなウィンドウがオープンしたままになります。ただし、優先度の高いウィンドウと重複している場合は、優先度の低いウィンドウがクローズし、優先度の高いウィンドウがオープンします。優先度の低いウィンドウを指定しているスケジュールを持つ現在実行中のジョブは、ジョブの作成時に割り当てた動作に従って停止される場合があります。
- ウィンドウの終了時に、定義されているウィンドウが複数ある場合は、優先度の最も高いウィンドウがオープンします。すべてのウィンドウの優先度が同じ場合は、残りの時間の比率の最も高いウィンドウがオープンします。
- 削除対象のオープン中のウィンドウは、自動的にクローズします。この時点で前述のルールが適用されます。

2 つのウィンドウが重複した場合は、スケジューラのログにエントリが必ず記録されます。

ウィンドウの重複例

図 27-1 に、24 時間スケジュールの場合のウィンドウ、リソース・プランおよび優先度の判別方法に関する一般的な例を示します。次の 2 つの例では、ウィンドウ 1 はリソース・プラン 1 に、ウィンドウ 2 はリソース・プラン 2 に関連付けられ、以下も同様に関連付けられているとします。

図 27-1 ウィンドウとリソース・プラン（例 1）

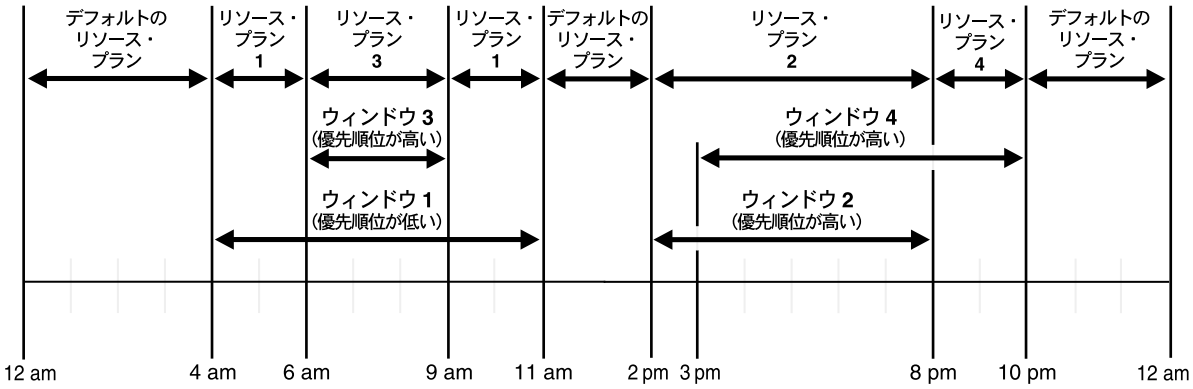


図 27-1 では、次の処理が発生します。

- 午前 12 時～午前 4 時

  いずれのウィンドウもオープンしていないため、デフォルトのリソース・プランが有効です。

- 午前 4 時～午前 6 時

  ウィンドウ 1 には低い優先度が割り当てられていますが、高い優先度のウィンドウが他に存在しないためウィンドウ 1 がオープンします。したがって、リソース・プラン 1 が有効です。

- 午前 6 時～午前 9 時

  ウィンドウ 1 より優先度の高いウィンドウ 3 がオープンします。したがって、リソース・プラン 3 が有効です。

- 午前 9 時～午前 11 時

  ウィンドウ 1 は、より優先度の高いウィンドウがオープンしたために午前 6 時にクローズしましたが、午前 9 時に、この優先度の高いウィンドウがクローズし、ウィンドウ 1 には当初のスケジュールの残り時間がまだ 2 時間あります。ウィンドウ 1 は残りの 2 時間再びオープン状態となり、リソース・プランが有効になります。

- 午前 11 時～午後 2 時

  いずれのウィンドウもオープンしていないため、デフォルトのリソース・プランが有効です。

- 午後 2 時～午後 3 時

  ウィンドウ 2 がオープンするため、リソース・プラン 2 が有効です。

- 午後 3 時～午後 8 時

  ウィンドウ 4 の優先度はウィンドウ 2 と同じであるため、ウィンドウ 2 への割込みはありません。したがって、リソース・プラン 2 が有効です。

- 午後 8 時～午後 10 時

  ウィンドウ 4 がオープンしているため、リソース・プラン 4 が有効です。

- 午後 10 時～午前 12 時

  いずれのウィンドウもオープンしていないため、デフォルトのリソース・プランが有効です。

図 27-2 に、24 時間スケジュールの場合のウィンドウ、リソース・プランおよび優先度の判別方法に関する別の例を示します。

図 27-2 ウィンドウとリソース・プラン（例 2）

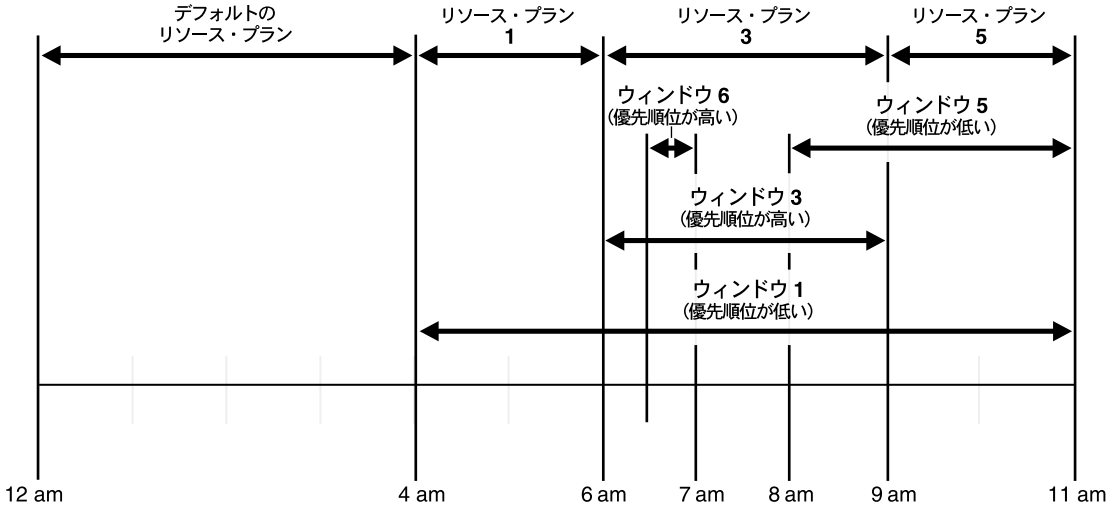


図 27-2 では、次の処理が発生します。

- 午前 12 時～午前 4 時

  デフォルトのリソース・プランが有効です。

- 午前 4 時～午前 6 時

  ウィンドウ 1 には低い優先度が割り当てられていますが、高い優先度のウィンドウが他に存在しないためウィンドウ 1 がオープンします。したがって、リソース・プラン 1 が有効です。

- 午前 6 時～午前 9 時

  ウィンドウ 1 より優先度の高いウィンドウ 3 がオープンします。ウィンドウ 6 は優先度の高い別のウィンドウがすでに有効であるためオープンしません。

- 午前 9 時～午前 11 時

  午前 9 時の時点では、ウィンドウ 5 またはウィンドウ 1 の 2 つが選択対象です。両方とも優先度が低いため、残り継続時間の比率の大きさに基づいて選択されます。ウィンドウ 1 の総継続時間と比較して、ウィンドウ 5 の残り時間の比率の方が大きい状態です。たとえば、ウィンドウ 1 が午前 11 時 30 分まで延長された場合でも、ウィンドウ 5 の残り継続時間に対する比率が 2/3 × 100% であるのに対して、ウィンドウ 1 は 2.5/7 × 100% であるため比率が小さくなります。したがって、リソース・プラン 5 が有効になります。

ウィンドウ・グループの使用

ウィンドウ・グループを使用すると、1 日あるいは 1 週間などの間に複数期間実行するジョブを簡単にスケジュールできます。ウィンドウ・グループを作成してウィンドウをグループに加え、ウィンドウ・グループ名をジョブの schedule_name 属性に設定すると、ジョブがウィンドウ・グループ内のすべてのウィンドウの指定期間に実行されます。

ウィンドウ・グループは SYS スキーマに格納されます。この項では、ウィンドウ・グループの基本的なタスクについて説明します。この項の内容は、次のとおりです。

- ウィンドウ・グループのタスクとそのプロシージャ
- ウィンドウ・グループの作成
- ウィンドウ・グループの削除
- ウィンドウ・グループへのメンバーの追加
- ウィンドウ・グループからのメンバーの削除
- ウィンドウ・グループの有効化
- ウィンドウ・グループの無効化

関連項目： ウィンドウ・グループの概要については、26-16 ページの「ウィンドウ・グループ」を参照してください。

ウィンドウ・グループのタスクとそのプロシージャ

表 27-7 に、ウィンドウ・グループの一般的なタスクとその処理に使用するプロシージャを示します。

表 27-7 ウィンドウ・グループのタスクとそのプロシージャ

タスク	プロシージャ	必要な権限
ウィンドウ・グループの作成	CREATE_WINDOW_GROUP	MANAGE SCHEDULER
ウィンドウ・グループの削除	DROP_WINDOW_GROUP	MANAGE SCHEDULER
ウィンドウ・グループへのメンバーの追加	ADD_WINDOW_GROUP_MEMBER	MANAGE SCHEDULER
ウィンドウ・グループからのメンバーの削除	REMOVE_WINDOW_GROUP_MEMBER	MANAGE SCHEDULER
ウィンドウ・グループの有効化	ENABLE	MANAGE SCHEDULER
ウィンドウ・グループの無効化	DISABLE	MANAGE SCHEDULER

権限の詳細は、28-31 ページの「スケジューラ権限」を参照してください。

ウィンドウ・グループの作成

ウィンドウ・グループを作成するには、CREATE_WINDOW_GROUP プロシージャまたは Enterprise Manager を使用します。グループのメンバー・ウィンドウは、グループの作成時に指定するか、または後で ADD_WINDOW_GROUP_MEMBER プロシージャを使用して追加できます。ウィンドウ・グループは別のウィンドウ・グループのメンバーとして設定できません。ただし、メンバーを設定しないウィンドウ・グループは作成できます。

ウィンドウ・グループを作成し、存在しないメンバー・ウィンドウを指定すると、エラーが生成され、ウィンドウ・グループは作成されません。ウィンドウがすでにウィンドウ・グループのメンバーである場合、再度追加されることはありません。

ウィンドウ・グループは SYS スキーマ内に作成されます。ウィンドウ・グループは、ウィンドウと同様に、PUBLIC に対するアクセス権限付きで作成されるため、ウィンドウ・グループにアクセスするための権限は必要ありません。

次の文では、downtime というウィンドウ・グループが作成され、2 つのウィンドウ（weeknights と weekends）が追加されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW_GROUP (
   group_name   =>  'downtime',
   window_list  =>  'weeknights, weekends');
END;
/
```

CREATE_WINDOW_GROUP プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ウィンドウ・グループの削除

1 つ以上のウィンドウ・グループを削除するには、DROP_WINDOW_GROUP プロシージャまたは Enterprise Manager を使用します。このプロシージャをコールすると、ウィンドウ・グループは削除されますが、ウィンドウ・グループのメンバーであるウィンドウは削除されません。ウィンドウ・グループ自体は削除せずに、ウィンドウ・グループのメンバーであるウィンドウをすべて削除する場合は、DROP_WINDOW プロシージャを使用し、そのコールにウィンドウ・グループ名を指定します。

DROP_WINDOW_GROUP プロシージャ・コールにウィンドウ・グループ名のカンマ区切りのリストを指定することで、1 回のコールで複数のウィンドウ・グループを削除できます。たとえば、次の文では 3 つのウィンドウ・グループが削除されます。

```
BEGIN
DBMS_SCHEDULER.DROP_WINDOW_GROUP('windowgroup1, windowgroup2, windowgroup3');
END;
/
```

ウィンドウ・グループの追加および削除の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ウィンドウ・グループへのメンバーの追加

ウィンドウ・グループにウィンドウを追加するには、ADD_WINDOW_GROUP_MEMBER プロシージャを使用します。

ウィンドウのカンマ区切りのリストを指定すると、1 回のコールで複数のメンバーをウィンドウ・グループに追加できます。たとえば、次の文ではウィンドウ・グループ window_group1 に 3 つのウィンドウが追加されます。

```
BEGIN
DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER ('window_group1',
   'window1, window2, window3');
END;
/
```

すでにオープンしているウィンドウがウィンドウ・グループに追加された場合、スケジューラは、ウィンドウ・グループ内の次のウィンドウがオープンするまで、このウィンドウ・グループを指し示すジョブを開始しません。

ADD_WINDOW_GROUP_MEMBER プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ウィンドウ・グループからのメンバーの削除

ウィンドウ・グループから 1 つ以上のウィンドウを削除するには、REMOVE_WINDOW_GROUP_MEMBER プロシージャまたは Enterprise Manager を使用します。stop_on_window_close フラグが設定されているジョブが停止するのは、ウィンドウがクローズするときのみです。オープン中のウィンドウをウィンドウ・グループから削除しても、この処理への影響はありません。

ウィンドウのカンマ区切りのリストを指定すると、1 回のコールで複数のメンバーをウィンドウ・グループから削除できます。たとえば、次の文では 3 つのウィンドウが削除されます。

```
BEGIN
DBMS_SCHEDULER.REMOVE_WINDOW_GROUP_MEMBER('window_group1', 'window1, window2,
   window3');
END;
/
```

REMOVE_WINDOW_GROUP_MEMBER プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ウィンドウ・グループの有効化

1 つ以上のウィンドウ・グループを有効（使用可能）にするには、ENABLE プロシージャまたは Enterprise Manager を使用します。デフォルトでは、ウィンドウ・グループは ENABLED で作成されます。次に例を示します。

```
BEGIN
DBMS_SCHEDULER.ENABLE('sys.windowgroup1', 'sys.windowgroup2, sys.windowgroup3');
END;
/
```

ENABLE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ウィンドウ・グループの無効化

ウィンドウ・グループを無効（使用禁止）にするには、DISABLE プロシージャまたは Enterprise Manager を使用します。ウィンドウ・グループを使用禁止にした場合、そのウィンドウ・グループをスケジュールとして設定しているジョブは、メンバーのウィンドウがオープンしても実行されません。ただし、ウィンドウ・グループのメタデータはそのままであるため、ウィンドウ・グループを再び使用可能にできます。ウィンドウ・グループのメンバーはオープンすることに注意してください。

DISABLE プロシージャ・コールにウィンドウ・グループ名のカンマ区切りのリストを指定することで、1 回のコールで複数のウィンドウ・グループを使用禁止にすることもできます。たとえば、次の文では 3 つのウィンドウ・グループが使用禁止になります。

```
BEGIN
DBMS_SCHEDULER.DISABLE('sys.windowgroup1, sys.windowgroup2, sys.windowgroup3');
END;
/
```

この例では、ウィンドウ・グループは使用禁止になりますが、ウィンドウ・グループのメンバーであるウィンドウは使用禁止になりません。

DISABLE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

イベントの使用

この項の内容は次のとおりです。

- イベントの概要
- スケジューラによって呼び出されるイベントの使用
- アプリケーションによって呼び出されるイベントの使用

関連項目：

- 28-29 ページ「イベントに基づくジョブとスケジュールの作成例」
- プロセス・フローを正確に制御するためにチェーンでイベントを使用する方法については、27-49 ページの「チェーンの使用」を参照してください。

イベントの概要

イベントは、なんらかの処理または発生が検出されたことを、アプリケーションまたはシステム・プロセスが別のアプリケーションまたはシステム・プロセスに示すために送信するメッセージです。イベントは、1 つのアプリケーションまたはプロセスによって**呼び出され**（送信）、1 つ以上のアプリケーションまたはプロセスによって**使用されます**（受信）。

スケジューラには、次の 2 種類のイベントがあります。

- スケジューラによって呼び出されるイベント

  スケジューラは、スケジューラ自体で発生する状態の変更を示すためにイベントを呼び出します。たとえば、スケジューラは、ジョブの開始時、ジョブの完了時、ジョブがその割当ての実行時間を超えたときなどにイベントを呼び出します。イベントのコンシューマは、そのイベントに対応して処理を実行するアプリケーションです。

  たとえば、システムの負荷が高いために、スケジュールされた開始時間から 30 分をすぎてもジョブが開始されない場合、スケジューラは、ハンドラのアプリケーションがデータベース管理者に電子メール通知を送信するイベントを呼び出すことができます。

- アプリケーションによって呼び出されるイベント

  アプリケーションは、スケジューラが使用するイベントを呼び出すことができます。スケジューラは、ジョブを開始することでこのイベントを処理します。スケジュールは、日付、時間、繰り返し発生する情報を定義するかわりに、イベントを参照するように作成できま

す。このようなスケジュール（**イベント・スケジュール**）が割り当てられたジョブは、イベントが呼び出されると実行されます。

たとえば、在庫が一定のしきい値を下回ったことを感知した在庫追跡システムでは、在庫の補充ジョブを開始するイベントを呼び出すことができます。

スケジューラは、Oracle Streams アドバンスト・キューイングを使用してイベントを呼び出して使用します。ジョブの状態の変更に関するイベントが発生すると、スケジューラは、メッセージをデフォルトのイベント・キューにエンキューします。アプリケーションは、このキューをサブスクライブし、イベント・メッセージをデキューして、適切な処理を行います。ジョブの開始をスケジューラに通知するイベントが発生すると、アプリケーションは、このジョブの設定時に指定したキューにメッセージをエンキューします。

関連項目：

- アドバンスト・キューイングの詳細は、『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイド』を参照してください。

スケジューラによって呼び出されるイベントの使用

ジョブの状態が変化したときに、スケジューラがイベントを呼び出すようにジョブを設定できます。このためには、raise_events ジョブ属性を設定します。この属性は CREATE_JOB プロシージャでは設定できないため、最初にジョブを作成し、SET_ATTRIBUTE プロシージャを使用してそのジョブを変更します。

デフォルトでは、SET_ATTRIBUTE を使用してジョブを変更するまで、ジョブによって状態変化のイベントは呼び出されません。

表 27-8 に、スケジューラによって呼び出されるイベントに関する 1 つの管理タスクの要約を示します。

表 27-8 スケジューラによって呼び出されるイベントに対するイベント・タスクとそのプロシージャ

タスク	プロシージャ	必要な権限
イベントを呼び出すようにジョブを変更する方法	SET_ATTRIBUTE	CREATE ANY JOB、あるいは変更対象のジョブの所有権またはそのジョブに対する ALTER 権限

ジョブに対するジョブの状態変化イベントを有効（使用可能）にすると、スケジューラは、メッセージをスケジューラ・イベント・キュー SYS.SCHEDULER$_EVENT_QUEUE にエンキューすることによってこれらのイベントを呼び出します。このキューはセキュアなキューであるため、アプリケーションによっては、特定のユーザーがキューの操作を実行できるようにキューを構成する必要があります。セキュアなキューの詳細は、『Oracle Streams 概要および管理』を参照してください。

スケジュール・イベント・キューが無制限に大きくならないように、スケジューラが呼び出したイベントは、デフォルトで 24 時間で期限切れになります（期限切れイベントはキューから削除されます）。この有効期限を変更するには、SET_SCHEDULER_ATTRIBUTE プロシージャを使用して event_expiry_time スケジューラ属性を設定します。詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

イベントを呼び出すようにジョブを変更する方法

ジョブに対するジョブの状態変化イベントを使用可能にするには、SET_ATTRIBUTE プロシージャを使用して、raise_events ジョブ属性のビット・フラグをオンにします。各ビット・フラグは、イベントの呼出しの対象となる様々なジョブ状態を表します。たとえば、最下位ビットをオンにすることで、job started イベントを呼び出すことができます。複数の状態変化イベント・タイプを 1 回のコールで使用可能にするには、必要なビット・フラグの値を加算し、その結果を引数として SET_ATTRIBUTE に提供します。

次の例では、ジョブ dw_reports に対する複数の状態変化イベントを使用可能にします。次のイベント・タイプを使用可能にします。両方がなんらかのエラーを示しています。

- JOB_FAILED
- JOB_SCH_LIM_REACHED

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE('dw_reports', 'raise_events',
   DBMS_SCHEDULER.JOB_FAILED + DBMS_SCHEDULER.JOB_SCH_LIM_REACHED);
END;
/
```

> **関連項目：** ジョブ状態ビット・フラグの名前と値の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_SCHEDULER.SET_ATTRIBUTE に関する説明を参照してください。

スケジューラ呼出しイベントのアプリケーションでの使用

スケジューラ・イベントを使用するには、アプリケーションがスケジューラ・イベント・キュー SYS.SCHEDULER$_EVENT_QUEUE をサブスクライブする必要があります。このキューはセキュアなキューであり、所有者は SYS です。あるユーザーについてこのキューのサブスクリプションを作成するには、次の処理を実行します。

1. SYS ユーザーまたは MANAGE ANY QUEUE 権限のあるユーザーでデータベースにログインします。
2. 新規または既存のエージェントを使用してキューをサブスクライブします。
3. パッケージ・プロシージャ DBMS_AQADM.ENABLE_DB_ACCESS を次のように実行します。

   ```
   DBMS_AQADM.ENABLE_DB_ACCESS(agent_name, db_username);
   ```

   agent_name は、イベント・キューのサブスクライブに使用したエージェントを表し、*db_username* は、サブスクリプションを作成する対象のユーザーです。

ユーザーにデキュー権限を付与する必要はありません。スケジューラ・イベント・キューに関するデキュー権限は、PUBLIC に付与されます。

または、次の例に示すように、ユーザーが ADD_EVENT_QUEUE_SUBSCRIBER プロシージャを使用してスケジューラ・イベント・キューをサブスクライブすることもできます。

```
DBMS_SCHEDULER.ADD_EVENT_QUEUE_SUBSCRIBER(subscriber_name);
```

この例の *subscriber_name* は、スケジューラ・イベント・キューのサブスクライブに使用される Oracle Streams アドバンスト・キューイング（AQ）エージェントの名前です。（NULL の場合は、呼出しユーザーのユーザー名でエージェントが作成されます）。このコールによって、スケジューラ・イベント・キューのサブスクリプションが作成され、指定エージェントを使用してデキューする許可がユーザーに付与されます。サブスクリプションはルールベースです。ルールによって、ユーザーが許可されるのは、所有するジョブが呼び出したイベントの参照のみで、その他のメッセージはすべて除外されます。サブスクリプションが使用可能になると、ユーザーは一定の間隔でメッセージをポーリングするか、またはメッセージが通知されるように AQ に登録できます。

詳細は、『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイド』を参照してください。

スケジューラ・イベント・キュー

スケジューラ・イベント・キュー SYS.SCHEDULER$_EVENT_QUEUE のタイプは scheduler$_event_info です。次に、このタイプの詳細を示します。

```
create or replace type sys.scheduler$_event_info as object
(
  event_type         VARCHAR2(4000),
  object_owner       VARCHAR2(4000),
  object_name        VARCHAR2(4000),
  event_timestamp    TIMESTAMP WITH TIME ZONE,
  error_code         NUMBER,
  error_msg          VARCHAR2(4000),
  event_status       NUMBER,
  log_id             NUMBER,
  run_count          NUMBER,
  failure_count      NUMBER,
  retry_count        NUMBER,
  spare1             NUMBER,
  spare2             NUMBER,
  spare3             VARCHAR2(4000),
  spare4             VARCHAR2(4000),
  spare5             TIMESTAMP WITH TIME ZONE,
  spare6             TIMESTAMP WITH TIME ZONE,
  spare7             RAW(2000),
  spare8             RAW(2000),
);
```

属性	説明
event_type	"JOB_STARTED"、"JOB_SUCCEEDED"、"JOB_FAILED"、"JOB_BROKEN"、"JOB_COMPLETED"、"JOB_STOPPED"、"JOB_SCH_LIM_REACHED"、"JOB_DISABLED"、"JOB_CHAIN_STALLED"、"JOB_OVER_MAX_DUR" のいずれか。 これらのイベント・タイプの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_SCHEDULER パッケージの定数に関する項を参照してください。
object_owner	イベントを呼び出したジョブの所有者。
object_name	イベントを呼び出したジョブの名前。
event_timestamp	イベントが発生した時刻。
error_code	ジョブ実行時にエラーが発生した場合にのみ使用可能です。トップレベルのエラー・コードが格納されます。
error_msg	ジョブ実行時にエラーが発生した場合にのみ使用可能です。エラー・スタック全体が格納されます。
event_status	イベント・タイプにさらに規定を追加します。event_type が "JOB_STARTED" の場合、ステータス 1 は通常の開始であり、ステータス 2 は再試行であることを示します。 event_type が "JOB_FAILED" の場合、ステータス 4 はジョブ実行中に発生したエラーによる失敗であり、ステータス 8 はなんからの異常終了であることを示します。 event_type が "JOB_STOPPED" の場合、ステータス 16 は正常な終了であり、ステータス 32 は FORCE オプションが TRUE に設定された終了であることを示します。
log_id	スケジューラ・ジョブ・ログ内の ID を指し示します。このログから追加情報を取得できます。イベントに対応するログ・エントリが必ず存在するとはかぎらないことに注意してください。エントリが存在しない場合、log_id は NULL です。
run_count	イベントが呼び出されたときのジョブの実行回数。

属性	説明
failure_count	イベントが呼び出されたときのジョブの失敗回数。
retry_count	イベントが呼び出されたときのジョブの再試行回数。
spare1 - spare8	現在は実装されていません。

アプリケーションによって呼び出されるイベントの使用

アプリケーションで、ジョブを開始するようにスケジューラに通知するイベントを呼び出すことができます。この方法で開始されるジョブは、イベントベースのジョブと呼ばれます。オプションで、ジョブはイベントのメッセージ内容を取得できます。

イベントベースのジョブを作成するには、次の 2 つの追加属性を設定する必要があります。

- queue_spec

  キュー仕様。アプリケーションがジョブ開始イベントを呼び出すためのメッセージをエンキューするキューの名前が含まれます。または、セキュアなキューの場合はキュー名の後にコンマとエージェント名が記述されます。

- event_condition

  メッセージ・プロパティに基づく条件式。メッセージによってジョブが開始されるには TRUE に評価される必要があります。式には、Oracle Streams アドバンスト・キューイング・ルールの構文を使用する必要があります。したがって、メッセージ・ペイロードがオブジェクト・タイプであり、式内のオブジェクト属性に tab.user_data の接頭辞を付けている場合は、式にユーザー・データ・プロパティを組み込むことができます。

  ルールの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_AQADM.ADD_SUBSCRIBER プロシージャに関する説明を参照してください。

  次の例では、event_condition が、午前 0 時から午前 9 時の間に発生する在庫不足イベントのみを選択するように設定されます。メッセージ・ペイロードは event_type と event_timestamp の 2 つの属性を持つオブジェクトであるとします。

```
event_condition = 'tab.user_data.event_type = ''LOW_INVENTORY'' and
extract hour from tab.user_data.event_timestamp < 9'
```

queue_spec および event_condition は、インラインのジョブ属性として指定するか、またはこれらの属性を指定した**イベント・スケジュール**を作成し、ジョブからこのスケジュールを指し示すことができます。

> **注意：** スケジューラは、event_condition と一致するイベントの発生ごとにイベントベースのジョブを実行します。ただし、ジョブの実行中に発生したイベントは無視されます。イベントは使用されますが、ジョブの別の実行はトリガーされません。

表 27-9 に、アプリケーションによって呼び出される（スケジューラによって使用される）イベントに関する一般的な管理タスクとそれに対応するプロシージャを示します。

表 27-9 アプリケーションによって呼び出されるイベントに対するイベント・タスクとそのプロシージャ

タスク	プロシージャ	必要な権限
イベントベースのジョブの作成	CREATE_JOB	CREATE JOB または CREATE ANY JOB
イベントベースのジョブの変更	SET_ATTRIBUTE	CREATE ANY JOB、あるいは変更対象のジョブの所有権またはそのジョブに対する ALTER 権限

表 27-9 アプリケーションによって呼び出されるイベントに対するイベント・タスクとそのプロシージャ（続き）

タスク	プロシージャ	必要な権限
イベント・スケジュールの作成	CREATE_EVENT_SCHEDULE	CREATE JOB または CREATE ANY JOB
イベント・スケジュールの変更	SET_ATTRIBUTE	CREATE ANY JOB、あるいは変更対象のスケジュールの所有権またはそのスケジュールに対する ALTER 権限

関連項目： キューの作成およびメッセージのエンキュー方法については、『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイド』を参照してください。

イベントベースのジョブの作成

イベントベースのジョブを作成するには、CREATE_JOB プロシージャまたは Enterprise Manager を使用します。ジョブには、ジョブ属性としてイベント情報をインラインで組み込むか、またはイベント・スケジュールを指し示してイベント情報を指定できます。

時間スケジュールに基づくジョブと同様に、イベントベースのジョブは、ジョブ終了日をすぎるか、max_runs または最大失敗回数（max_failures）に達するまでは自動削除されません。

イベント情報をジョブ属性として指定する方法 イベント情報をジョブ属性として指定するには、CREATE_JOB の代替構文を使用して、queue_spec および event_condition 属性を組み込みます。

次の例では、ファイルがサーバーに到着したことを示すシグナルがアプリケーションからスケジューラに送信されるときに開始されるジョブを作成しています。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name            =>  'process_file_j1',
   program_name        =>  'process_file_p1',
   event_condition     =>  'tab.user_data.event_type = ''FILE_ARRIVAL''',
   queue_spec          =>  'file_events_q, file_agent1',
   enabled             =>  TRUE,
   comments            =>  'Start job when a file arrives on the system');
END;
/
```

CREATE_JOB プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

イベント情報をイベント・スケジュールで指定する方法 イベント情報をイベント・スケジュールで指定するには、ジョブの schedule_name 属性にイベント・スケジュールの名前を設定する必要があります。次に例を示します。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name            =>  'process_file_j1',
   program_name        =>  'process_file_p1',
   schedule_name       =>  'file_events_schedule',
   enabled             =>  TRUE,
   comments            =>  'Start job when a file arrives on the system');
END;
/
```

詳細は、27-47 ページの「イベント・スケジュールの作成」を参照してください。

イベントベースのジョブの変更

イベントベースのジョブを変更するには、SET_ATTRIBUTE プロシージャを使用します。イベントをインラインで指定するジョブの場合、SET_ATTRIBUTE を使用して queue_spec 属性と event_condition 属性を別々に設定することはできません。かわりに、event_spec と呼ばれる属性を設定し、イベント条件とキュー仕様を 3 番目と 4 番目の引数として SET_ATTRIBUTE に渡す必要があります。

次に、event_spec 属性の使用例を示します。

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE ('my_job', 'event_spec',
   'tab.user_data.event_type = ''FILE_ARRIVAL''', 'file_events_q, file_agent1');
END;
/
```

SET_ATTRIBUTE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

イベント・スケジュールの作成

イベントに基づいたスケジュールを作成できます。作成したスケジュールは、複数のジョブに再利用できます。そのためには、CREATE_EVENT_SCHEDULE プロシージャまたは Enterprise Manager を使用します。次に、イベント・スケジュールの作成例を示します。

```
BEGIN
DBMS_SCHEDULER.CREATE_EVENT_SCHEDULE (
   schedule_name     =>  'file_events_schedule',
   start_date        =>  SYSTIMESTAMP,
   event_condition   =>  'tab.user_data.event_type = ''FILE_ARRIVAL''',
   queue_spec        =>  'file_events_q, file_agent1');
END;
/
```

イベント・スケジュールを削除するには、DROP_SCHEDULE プロシージャを使用します。CREATE_EVENT_SCHEDULE の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

イベント・スケジュールの変更

イベント・スケジュールのイベント情報を変更する方法は、ジョブのイベント情報を変更する方法と同じです。詳細は、27-47 ページの「イベントベースのジョブの変更」を参照してください。

次の例は、イベント・スケジュールのイベント情報を変更するための SET_ATTRIBUTE プロシージャおよび event_spec 属性の使用方法を示しています。

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE ('file_events_schedule', 'event_spec',
   'tab.user_data.event_type = ''FILE_ARRIVAL''', 'file_events_q, file_agent1');
END;
/
```

SET_ATTRIBUTE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

イベントベースのジョブにイベント・メッセージを渡す方法

メタデータの引数を介して、スケジューラは、イベントベースのジョブに対して、ジョブを開始したイベントのメッセージ内容を渡すことができます。次の規則が適用されます。

- ジョブは、タイプ STORED_PROCEDURE の名前付きプログラムを使用する必要があります。
- 名前付きプログラムの引数の 1 つがメタデータ引数であり、その metadata_attribute が EVENT_MESSAGE に設定されている必要があります。
- プログラムを実装するストアド・プロシージャには、名前付きプログラムのメタデータ引数に対応する位置に引数が必要です。引数の型は、アプリケーションがジョブ開始イベントをエンキューするキューのデータ型であることが必要です。

EVENT_MESSAGE メタデータ引数があるジョブを RUN_JOB プロシージャを使用して手動で実行した場合、その引数に渡される値は NULL です。

次の例は、イベント・メッセージの内容を受け取ることができるイベントベースのジョブを作成する方法を示しています。

```
create or replace procedure my_stored_proc (event_msg IN event_queue_type)
as
begin
  -- retrieve and process message body
end;
/

begin
  dbms_scheduler.create_program (
      program_name => 'my_prog',
      program_action=> 'my_stored_proc',
      program_type => 'STORED_PROCEDURE',
      number_of_arguments => 1,
      enabled => FALSE) ;

  dbms_scheduler.define_metadata_argument (
      program_name => 'my_prog',
      argument_position => 1 ,
      metadata_attribute => 'EVENT_MESSAGE') ;

  dbms_scheduler.enable ('my_prog');
exception
  when others then raise ;
end ;
/

begin
  dbms_scheduler.create_job (
     job_name => 'my_evt_job' ,
     program_name => 'my_prog',
     schedule_name => 'my_evt_sch',
     enabled => true,
     auto_Drop => false) ;
exception
  when others then raise ;
end ;
/
```

チェーンの使用

チェーンは、結合した 1 つの目的のために互いにリンクされた一連の名前付きタスクです。チェーンは、前の 1 つ以上のジョブの結果に応じて異なるジョブが開始される依存性ベースのスケジューリングを実装できる手段です。

チェーンを作成して使用するには、次のタスクを実行します。

タスク	参照先
1. チェーン・オブジェクトを作成します。	チェーンの作成
2. チェーン内のステップを定義します。	チェーン・ステップの定義
3. ルールを追加します。	チェーンへのルールの追加
4. チェーンを使用可能にします。	チェーンの有効化
5. チェーンを指し示すジョブ（チェーン・ジョブ）を作成します。	チェーン用のジョブの作成

この項で説明する他の内容は、次のとおりです。

- チェーンのタスクとそのプロシージャ
- チェーンの削除
- チェーンの実行
- チェーン・ルールの削除
- チェーンの無効化
- チェーン・ステップの削除
- チェーンの停止
- 個々のチェーン・ステップの停止
- チェーンの一時停止
- チェーン・ステップのスキップ
- チェーンの一部実行
- 実行中のチェーンの監視
- 停止状態チェーンの処理

関連項目：

- チェーンの概要については、26-12 ページの「チェーン」を参照してください。
- 28-27 ページ「チェーンの作成例」

チェーンのタスクとそのプロシージャ

表 27-10 に、チェーンに関する一般的なタスクとそれに対応するプロシージャを示します。

表 27-10 チェーンのタスクとそのプロシージャ

タスク	プロシージャ	必要な権限
チェーンの作成	CREATE_CHAIN	CREATE JOB、CREATE EVALUATION CONTEXT および CREATE RULE SET（所有者の場合）。CREATE ANY JOB、CREATE ANY RULE SET および CREATE ANY EVALUATION CONTEXT（所有者以外の場合）。
チェーンの削除	DROP_CHAIN	チェーンの所有権、あるいはチェーンに対する ALTER 権限または CREATE ANY JOB 権限。所有者以外の場合は、DROP ANY EVALUATION CONTEXT および DROP ANY RULE SET も必要です。
チェーンの変更	ALTER_CHAIN	チェーンの所有権、あるいはチェーンに対する ALTER 権限または CREATE ANY JOB。
チェーンの変更	SET_ATTRIBUTE	チェーンの所有権、あるいはチェーンに対する ALTER 権限または CREATE ANY JOB。
実行中チェーンの変更	ALTER_RUNNING_CHAIN	ジョブの所有権、あるいはジョブに対する ALTER 権限または CREATE ANY JOB。
チェーンの実行	RUN_CHAIN	CREATE JOB または CREATE ANY JOB。また、新規ジョブの所有者には、チェーンに対する EXECUTE 権限または EXECUTE ANY PROGRAM が必要です。
チェーンへのルールの追加	DEFINE_CHAIN_RULE	チェーンの所有権、あるいはチェーンに対する ALTER 権限または CREATE ANY JOB 権限。CREATE RULE（チェーンの所有者の場合）、CREATE ANY RULE（チェーンの所有者以外の場合）。
チェーン内のルールの変更	DEFINE_CHAIN_RULE	チェーンの所有権、あるいはチェーンに対する ALTER 権限または CREATE ANY JOB 権限。チェーンの所有者以外の場合は、そのルールに対する ALTER 権限または ALTER ANY RULE が必要です。
チェーンからのルールの削除	DROP_CHAIN_RULE	チェーンの所有権、あるいはチェーンに対する ALTER 権限または CREATE ANY JOB 権限。DROP ANY RULE（チェーンの所有者以外の場合）。
チェーンの有効化	ENABLE	チェーンの所有権、あるいはチェーンに対する ALTER 権限または CREATE ANY JOB。
チェーンの無効化	DISABLE	チェーンの所有権、あるいはチェーンに対する ALTER 権限または CREATE ANY JOB。
ステップの作成	DEFINE_CHAIN_STEP	チェーンの所有権、あるいはチェーンに対する ALTER 権限または CREATE ANY JOB。
ステップの削除	DROP_CHAIN_STEP	チェーンの所有権、あるいはチェーンに対する ALTER 権限または CREATE ANY JOB。
ステップの変更	DEFINE_CHAIN_STEP	チェーンの所有権、あるいはチェーンに対する ALTER 権限または CREATE ANY JOB。

チェーンの作成

チェーンを作成するには、CREATE_CHAIN プロシージャを使用します。CREATE_CHAIN を使用してチェーン・オブジェクトを作成した後、チェーン・ステップおよびチェーン・ルールを別々に定義します。

次に、チェーンの作成例を示します。

```
BEGIN
DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name          => 'my_chain1',
   rule_set_name       => NULL,
   evaluation_interval => NULL,
   comments            => 'My first chain');
END;
/
```

rule_set_name および evaluation_interval 引数は通常 NULL のままです。evaluation_interval では、チェーン・ルールが評価される繰返し間隔を定義できます。rule_set_name は、Oracle Streams 内で定義されたルール・セットです。

関連項目：

- evaluation_interval 属性の詳細は、27-52 ページの「チェーンへのルールの追加」を参照してください。
- CREATE_CHAIN の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- ルールおよびルール・セットの詳細は、『Oracle Streams 概要および管理』を参照してください。

チェーン・ステップの定義

チェーン・オブジェクトの作成後、1 つ以上のチェーン・ステップを定義します。各ステップは、次のいずれかを指し示します。

- スケジューラ・プログラム・オブジェクト（プログラム）
- 別のチェーン（ネストしたチェーン）
- イベント・スケジュールまたはインライン・イベント

プログラムまたはネストしたチェーンを指し示すステップを定義するには、DEFINE_CHAIN_STEP プロシージャを使用します。次に、my_chain1 に 2 つのステップを追加する例を示します。

```
BEGIN
DBMS_SCHEDULER.DEFINE_CHAIN_STEP (
   chain_name      =>  'my_chain1',
   step_name       =>  'my_step1',
   program_name    =>  'my_program1');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP (
   chain_name      =>  'my_chain1',
   step_name       =>  'my_step2',
   program_name    =>  'my_chain2');
END;
/
```

名前付きプログラムまたはチェーンは、ステップを定義する時点では、存在していなくてもかまいません。ただし、チェーンの実行時には存在して使用可能になっている必要があります。そうでない場合は、エラーが生成されます。

外部実行可能ファイルを実行するステップ

外部実行可能ファイルを実行するステップを定義した後、ALTER_CHAIN プロシージャを使用して、そのステップの資格証明を設定する必要があります。リモートの外部実行可能ファイルの場合は、ALTER_CHAIN プロシージャを使用してステップの宛先も設定してください。

イベントを待機するステップ

イベントの発生を待機するステップを定義するには、DEFINE_CHAIN_EVENT_STEP プロシージャを使用します。プロシージャの引数を使用して、イベント・スケジュールを指し示すか、またはインラインのキュー仕様およびイベント条件を組み込むことができます。この例では、名前付きイベント・スケジュール内に指定されているイベントを待機する、3 番目のチェーン・ステップが作成されます。

```
BEGIN
DBMS_SCHEDULER.DEFINE_CHAIN_EVENT_STEP (
   chain_name           =>  'my_chain1',
   step_name            =>  'my_step3',
   event_schedule_name  =>  'my_event_schedule');
END;
/
```

ステップを再開可能にする方法

データベース・リカバリ後、デフォルトでは、実行中だったステップが STOPPED としてマークされ、チェーンが続行されます。ALTER_CHAIN を使用してチェーン・ステップの restart_on_recovery 属性を TRUE に設定すると、これらのステップをデータベース・リカバリ後に自動的に再開するように指定できます。

DEFINE_CHAIN_STEP、DEFINE_CHAIN_EVENT_STEP および ALTER_CHAIN プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

チェーンへのルールの追加

チェーンにルールを追加するには、DEFINE_CHAIN_RULE プロシージャを使用します。このプロシージャは、チェーンに追加する各ルールに対して 1 回ずつコールします。

チェーン・ルールでは、ステップの実行時期、およびステップ間の依存関係が定義されます。各ルールには条件と処理があります。ルールが評価されるときに、ルールの条件が TRUE に評価されると、その処理が実行されます。条件では、スケジューラ・チェーン条件の構文、または SQL の WHERE 句で有効な構文を使用できます。この構文には、ステップの完了ステータスなど、チェーン・ステップの属性への参照を組み込むことができます。標準な処理は、指定したステップを実行すること、またはステップのリストを実行することです。

チェーン・ルールはすべてまとまって機能し、チェーン全体の処理を定義します。チェーン・ジョブの開始時および各ステップの終了時に、次に実行される処理を判断するためにすべてのルールが評価されます。複数のルールに TRUE 条件がある場合は、複数の処理を発生させることができます。チェーンの evaluation_interval 属性を設定して、ルールを一定間隔で評価させることもできます。

条件は通常、1 つ以上の先行するステップの結果に基づきます。たとえば、先行する 2 つのステップが成功した場合はあるステップを実行し、2 つのステップのいずれかが失敗した場合には別のステップを実行する場合があります。

スケジューラのチェーン条件構文には、次の 2 つの書式があります。

stepname [NOT] {SUCCEEDED|FAILED|STOPPED|COMPLETED}
stepname ERROR_CODE {*comparision_operator*|[NOT] IN} {*integer*|*list_of_integers*}

条件をブール演算子 AND、OR および NOT() と組み合せて条件式を作成できます。式にかっこを使用すると、評価順序を指定できます。

ステップに割り当てたプログラム内で RAISE_APPLICATION_ERROR PL/SQL 文を使用して、ERROR_CODE を設定できます。この方法でプログラムにより設定されるエラー・コードは負の

数値ですが、チェーン・ルール内で ERROR_CODE をテストするときには正の数値をテストします。たとえば、プログラムに次の文が含まれている場合を考えます。

```
RAISE_APPLICATION_ERROR(-20100, errmsg);
```

チェーン・ルールの条件を次のように指定する必要があります。

```
stepname ERROR_CODE=20100
```

ステップ属性

次のリストに、SQL WHERE 句の構文を使用する際に条件に含めることのできるステップ属性を示します。

```
completed
state
start_date
end_date
error_code
duration
```

completed 属性はブール値で、state 属性が SUCCEEDED、FAILED または STOPPED のときには TRUE です。

表 27-11 に、state 属性に可能な値を示します。これらの値は、*_SCHEDULER_RUNNING_CHAINS ビューの STATE 列で参照可能です。

表 27-11 チェーン・ステップの state 属性の値

state 属性の値	意味
NOT_STARTED	ステップのチェーンは実行中ですが、ステップはまだ開始されていません。
SCHEDULED	ルールによって AFTER 句を使用してステップが開始されており、指定の待機時間はまだ満了していません。
RUNNING	ステップは実行中です。イベント・ステップの場合、ステップは開始されており、イベントを待機中です。
PAUSED	ステップの PAUSE 属性が TRUE に設定されており、ステップは一時停止されています。依存しているステップを開始する前に、一時停止を解除する必要があります。
SUCCEEDED	ステップは正常に完了しています。ステップの ERROR_CODE は 0（ゼロ）です。
FAILED	ステップはエラーで完了しました。ERROR_CODE は 0（ゼロ）以外の値です。
STOPPED	ステップは STOP_JOB プロシージャで停止されました。
STALLED	ステップはネストしたチェーンであり、停止されています。

SQL WHERE 句の構文に関するルールと例については、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DEFINE_CHAIN_RULE プロシージャに関する項を参照してください。

条件の例

次の例では、スケジューラ・チェーン条件の構文を使用しています。

次の条件を含んだルールによって開始されたステップは、form_validation_step というステップが完了（SUCCEEDED、FAILED または STOPPED）すると開始されます。

```
form_validation_step COMPLETED
```

次の条件も同様ですが、条件が満たされるにはステップが成功する必要があることを示しています。

```
form_validation_step SUCCEEDED
```

次の条件ではエラーがテストされます。ステップ form_validation_step が失敗して 20001 以外のエラー・コードが戻された場合は、TRUE になります。

```
form_validation_step FAILED AND form_validation_step ERROR_CODE != 20001
```

その他の例については、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DEFINE_CHAIN_RULE プロシージャに関する項を参照してください。

チェーンの開始

チェーン・ジョブの開始時にチェーンを開始できるように、少なくとも 1 つのルールに、常に TRUE に評価される条件が必要です。このための最も簡単な方法は、スケジュール・チェーン条件の構文を使用している場合は条件を 'TRUE' に設定すること、または SQL 構文を使用している場合は条件を '1=1' に設定することです。

チェーンの終了

少なくとも 1 つのチェーン・ルールに、'END' の action が含まれている必要があります。チェーン・ジョブは、END 処理が含まれているルールの 1 つが TRUE に評価されるまでは完了しません。通常は、異なる END 処理が設定された別々のルールがあり、エラー・コードを伴う場合と伴わない場合があります。

チェーンに実行するステップがなくなるか、またはイベントの発生の待機中ではなく、END 処理が含まれているルールで TRUE に評価されるルールがない（または END 処理が設定されているルールがない）場合、チェーン・ジョブの状態は CHAIN_STALLED になります。詳細は、27-61 ページの「停止状態チェーンの処理」を参照してください。

ルールの定義例

次の例では、ステップ step1 でチェーンを開始するルール、step1 の完了時にステップ step2 を開始するルールが定義されます。rule_name および comments はオプションであり、デフォルトで NULL に設定されます。rule_name を使用しない場合は、後で DEFINE_CHAIN_RULE のもう 1 つのコールを使用して、そのルールを再定義できます。新しい定義により前の定義が上書きされます。

```
BEGIN
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   chain_name    =>   'my_chain1',
   condition     =>   'TRUE',
   action        =>   'START step1',
   rule_name     =>   'my_rule1',
   comments      =>   'start the chain');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   chain_name    =>   'my_chain1',
   condition     =>   'step1 completed',
   action        =>   'START step2',
   rule_name     =>   'my_rule2');
END;
/
```

関連項目：

- DEFINE_CHAIN_RULE プロシージャおよびスケジューラ・チェーン条件の構文の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 28-27 ページ「チェーンの作成例」

チェーン・ルールの評価間隔の設定

スケジューラでは、チェーン・ジョブの開始時と各チェーン・ステップの終了時に、すべてのチェーン・ルールが評価されます。1 時間に 1 回など、一定の時間間隔でもルールが評価されるようにチェーンを構成できます。この機能は、時間に基づいて、またはチェーン外部での発生に基づいてチェーン・ステップを開始する必要がある場合に役立ちます。次に例を示します。

- チェーン・ステップがリソース集中型であるため、オフピーク時に実行する必要があるとします。ステップの条件として、別のステップの完了と午後 6 時から午前 0 時までの時間の両方を設定できます。スケジューラでは、この条件が TRUE になる時期を判別するために、ルールをその都度評価する必要があります。
- ステップが、チェーン外部にある他のプロセスから表にデータが到着するまで待機する必要があるとします。このステップの条件として、別のステップの完了と行を含む特定の表の両方を設定できます。スケジューラでは、この条件が TRUE になる時期を判別するために、ルールをその都度評価する必要があります。この条件は SQL WHERE 句構文を使用し、次のようになります。

  ```
  ':step1.state=''SUCCEEDED'' AND select count(*) from oe.sync_table > 0'
  ```

チェーンの評価間隔を設定するには、チェーンの作成時に evaluation_interval 属性を設定します。この属性のデータ型は INTERVAL DAY TO SECOND です。

```
BEGIN
DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name          => 'my_chain1',
   rule_set_name       => NULL,
   evaluation_interval => INTERVAL '30' MINUTE,
   comments            => 'Chain with 30 minute evaluation interval');
END;
/
```

チェーンの有効化

チェーンを有効（使用可能）にするには、ENABLE プロシージャを使用します。ジョブでチェーンを実行するには、チェーンを使用可能にする必要があります。すでに使用可能なチェーンを使用可能にしてもエラーは戻りません。

次の例では、チェーン my_chain1 が使用可能になります。

```
BEGIN
DBMS_SCHEDULER.ENABLE ('my_chain1');
END;
/
```

ENABLE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

注意： チェーンは、次の場合にスケジューラによって自動的に使用禁止にされます。

- いずれかのチェーン・ステップが指し示しているプログラムが削除されている場合
- いずれかのチェーン・ステップが指し示しているネストしたチェーンが削除されている場合
- いずれかのチェーン・イベント・ステップが指し示しているイベント・スケジュールが削除されている場合

チェーン用のジョブの作成

チェーンを実行するには、RUN_CHAIN プロシージャを使用するか、タイプ 'CHAIN' のジョブ（**チェーン・ジョブ**）を作成してスケジュールする必要があります。ジョブの処理では、次の例に示すように、チェーン名を参照する必要があります。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name        => 'chain_job_1',
   job_type        => 'CHAIN',
   job_action      => 'my_chain1',
   repeat_interval => 'freq=daily;byhour=13;byminute=0;bysecond=0',
   enabled         => TRUE);
END;
/
```

実行中のチェーン・ジョブの各ステップには、スケジューラによって、チェーン・ジョブと同じジョブ名と所有者を設定した**ステップ・ジョブ**が作成されます。各ステップ・ジョブにはさらに、そのジョブを一意に識別するためのサブ名があります。ジョブのサブ名は、ビュー *_SCHEDULER_RUNNING_JOBS、*_SCHEDULER_JOB_LOG および *_SCHEDULER_JOB_RUN_DETAILS に列として組み込まれます。また、次の場合を除いて、ジョブのサブ名は通常はステップ名と同じです。

- ネストしたチェーンの場合、現行のステップ名がジョブのサブ名としてすでに使用されている可能性があります。この場合、スケジューラは、ステップ名に '*_N*' を追加します。*N* は整数であり、この追加によってジョブ・サブ名が一意になります。
- ステップ・ジョブの作成時にエラーが発生した場合、スケジューラは、ジョブ・サブ名を '*step_name*_0' に設定して、ジョブ・ログ・ビュー（*_SCHEDULER_JOB_LOG および *_SCHEDULER_JOB_RUN_DETAILS）に FAILED エントリを記録します。

関連項目：

- CREATE_JOB プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 事前にチェーン・ジョブを作成せずにチェーンを実行する別の方法は、27-57 ページの「チェーンの実行」を参照してください。

チェーンの削除

チェーンをステップとルールも含めて削除するには、DROP_CHAIN プロシージャを使用します。次に、チェーンを削除する例を示します。この例では、my_chain1 が削除されます。

```
BEGIN
DBMS_SCHEDULER.DROP_CHAIN (
   chain_name   => 'my_chain1',
   force        => TRUE);
END;
/
```

DROP_CHAIN プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

チェーンの実行

次の 2 つのプロシージャを使用して、チェーンを即時に実行できます。

- RUN_JOB
- RUN_CHAIN

すでにチェーン用のチェーン・ジョブを作成している場合は、RUN_JOB プロシージャを使用してそのジョブを実行（したがってチェーンを実行）できますが、RUN_JOB の use_current_session 引数を FALSE に設定する必要があります。

RUN_CHAIN プロシージャを使用すると、チェーンに対するチェーン・ジョブをあらかじめ作成することなく、チェーンを実行できます。また、RUN_CHAIN を使用してチェーンの一部のみを実行することもできます。

RUN_CHAIN では、指定したチェーンを実行する一時ジョブが作成されます。ジョブ名を指定した場合は、その名前でジョブが作成されます。指定しない場合は、デフォルトのジョブ名が割り当てられます。

開始ステップ・リストを指定すると、チェーンの実行開始時にそれらのステップのみが開始されます（通常開始されるステップは、それらがリスト内にない場合は実行されません）。開始ステップのリストが提供されない場合は、チェーンが通常どおり開始されます。つまり、初期評価が行われ、実行を開始するステップが判断されます。次に、チェーン my_chain1 を即時に実行する例を示します。

```
BEGIN
DBMS_SCHEDULER.RUN_CHAIN (
   chain_name    =>  'my_chain1',
   job_name      =>  'partial_chain_job',
   start_steps   =>  'my_step2, my_step4');
END;
/
```

関連項目：

- 27-60 ページの「チェーンの一部実行」
- RUN_CHAIN プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

チェーン・ルールの削除

チェーンからルールを削除するには、DROP_CHAIN_RULE プロシージャを使用します。次に、my_rule1 を削除する例を示します。

```
BEGIN
DBMS_SCHEDULER.DROP_CHAIN_RULE (
   chain_name   =>   'my_chain1',
   rule_name    =>   'my_rule1',
   force        =>   TRUE);
END;
/
```

DROP_CHAIN_RULE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

チェーンの無効化

チェーンを無効（使用禁止）にするには、DISABLE プロシージャを使用します。次に、my_chain1 を使用禁止にする例を示します。

```
BEGIN
DBMS_SCHEDULER.DISABLE ('my_chain1');
END;
/
```

DISABLE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

注意： チェーンは、次の場合にスケジューラによって自動的に使用禁止にされます。

- いずれかのチェーン・ステップが指し示しているプログラムが削除されている場合
- いずれかのチェーン・ステップが指し示しているネストしたチェーンが削除されている場合
- いずれかのチェーン・イベント・ステップが指し示しているイベント・スケジュールが削除されている場合

チェーン・ステップの削除

チェーンからステップを削除するには、DROP_CHAIN_STEP プロシージャを使用します。次に、my_chain2 から my_step2 を削除する例を示します。

```
BEGIN
DBMS_SCHEDULER.DROP_CHAIN_STEP (
   chain_name   =>   'my_chain2',
   step_name    =>   'my_step2',
   force        =>    TRUE);
END;
/
```

DROP_CHAIN_STEP プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

チェーンの停止

実行中のチェーンを停止するには、DBMS_SCHEDULER.STOP_JOB をコールして、チェーン・ジョブ（チェーンを開始したジョブ）の名前を渡します。チェーン・ジョブを停止すると、実行中のチェーンの全ステップが停止され、チェーンが終了します。

STOP_JOB プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

個々のチェーン・ステップの停止

個々のチェーン・ステップを停止するには 2 つの方法があります。

- ルールの条件を満たしたときに 1 つ以上のステップを停止するチェーン・ルールの作成
- STOP_JOB プロシージャのコール

  停止するステップごとに、スキーマ名、チェーン・ジョブ名およびステップ・ジョブ・サブ名を指定する必要があります。

```
BEGIN
  DBMS_SCHEDULER.STOP_JOB('oe.chainrunjob.stepa');
END;
/
```

  この例では、chainrunjob はチェーン・ジョブ名で、stepa はステップ・ジョブ・サブ名です。通常、ステップ・ジョブ・サブ名はステップ名と同じですが、異なる場合もあります。ステップ・ジョブ・サブ名は、*_SCHEDULER_RUNNING_CHAINS ビューの STEP_JOB_SUBNAME 列から取得できます。

チェーン・ステップを停止すると、state が STOPPED に設定され、チェーン・ルールが評価されて次に実行するステップが判別されます。

STOP_JOB プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

チェーンの一時停止

チェーン全体、またはチェーンの個別ブランチを一時停止できます。そのためには、DBMS_SCHEDULER.ALTER_CHAIN または ALTER_RUNNING_CHAIN を使用して、1 つ以上のステップの PAUSE 属性を TRUE に設定します。チェーン・ステップを一時停止すると、チェーンの実行をそのステップの実行後に一時停止できます。

ステップを一時停止すると、そのステップの実行後に state 属性が PAUSED に変わりますが、completed 属性は FALSE のままです。そのため、一時停止したステップの完了に依存しているステップは実行されません。一時停止したステップの PAUSE 属性を FALSE にリセットすると、state 属性が完了状態（SUCCEEDED、FAILED または STOPPED）に設定され、一時停止したステップの完了を待機しているステップを実行できるようになります。

図 27-3 ステップ 3 で一時停止したチェーン

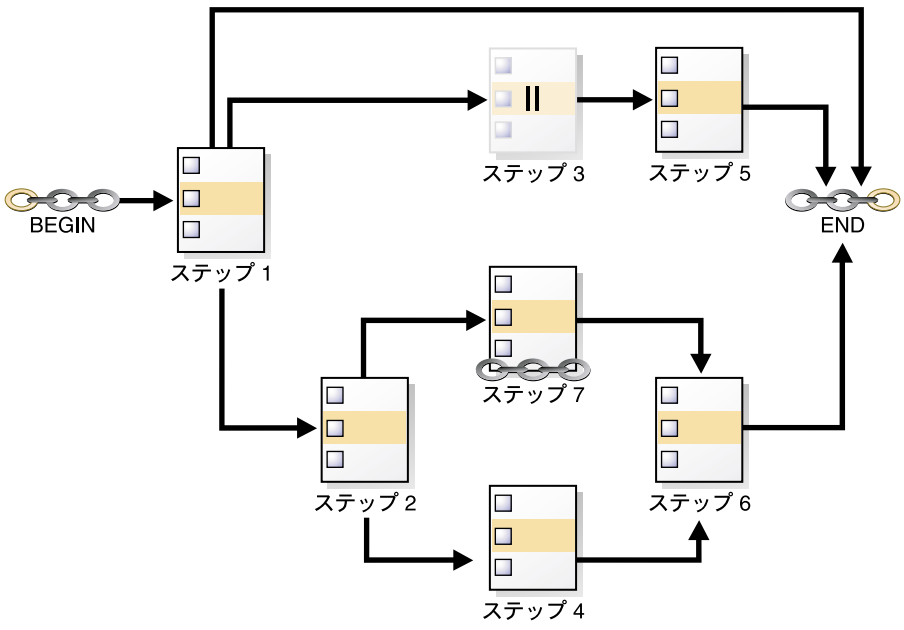


図 27-3 では、ステップ 3 が一時停止されています。ステップ 3 の一時停止が解除されるまで、ステップ 5 は実行されません。ステップ 2 のみを一時停止した場合、ステップ 4、6 および 7 は実行されません。ただし、ステップ 1、3 および 5 は実行できます。いずれの場合も、チェーンのブランチを 1 つのみ一時停止することになります。

チェーン全体を一時停止するには、チェーンのステップをすべて一時停止します。チェーンの一時停止を解除するには、チェーン・ステップの 1 つ、多数またはすべての一時停止を解除します。図 27-3 のチェーンでは、ステップ 1 を一時停止すると、ステップ 1 の実行後にチェーン全体も一時停止します。

関連項目： 『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS_SCHEDULER.ALTER_CHAIN および DBMS_SCHEDULER.ALTER_RUNNING_CHAIN プロシージャに関する項を参照してください。

チェーン・ステップのスキップ

チェーンの 1 つ以上のステップをスキップできます。そのためには、DBMS_SCHEDULER.ALTER_CHAIN または ALTER_RUNNING_CHAIN を使用して、1 つ以上のステップの SKIP 属性を TRUE に設定します。ステップの SKIP 属性が TRUE の場合、そのステップを実行するためのチェーン条件を満たすと、そのステップは実行されるかわりに、即時に成功した場合と同様に処理されます。SKIP を TRUE に設定しても、実行中のステップ、遅延後に実行するようにスケジュールされているステップまたは実行済のステップには影響しません。

ステップのスキップが特に役立つのは、チェーンのテスト時です。たとえば、27-59 ページの図 27-3 に示したチェーンをテストする際に、ステップ 7 をスキップすると、このステップはネストしたチェーンであるため、テスト時間を大幅に短縮できます。

チェーンの一部実行

チェーンの一部のみを実行するには 2 つの方法があります。

- ALTER_CHAIN プロシージャを使用して 1 つ以上のステップの PAUSE 属性を TRUE に設定し、RUN_JOB を使用してチェーン・ジョブを開始するか、または RUN_CHAIN を使用してチェーンを開始します。一時停止したステップに依存するステップは実行されません （ただし、一時停止したステップは実行されます）。

  この方法のデメリットは、チェーンを将来実行する場合に備えて、影響を受けるステップの PAUSE 属性の設定を FALSE に戻す必要があることです。

- RUN_CHAIN プロシージャを使用して、実行しないステップをスキップし、チェーンの特定ステップのみを開始します。

  この方法の方が簡単であり、ステップの開始前に初期状態を設定することもできます。

この 2 つの方法の両方を使用して、チェーンの開始時と終了時の両方でステップをスキップすることが必要な場合もあります。

詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の RUN_CHAIN プロシージャの説明を参照してください。

関連項目： 27-60 ページの「チェーン・ステップのスキップ」

実行中のチェーンの監視

次の 2 つのビューを使用して、実行中のチェーンのステータスを表示できます。

*_SCHEDULER_RUNNING_JOBS
*_SCHEDULER_RUNNING_CHAINS

*_SCHEDULER_RUNNING_JOBS ビューには、チェーン・ジョブに関する 1 行と実行中の各ステップに関する 1 行が含まれています。*_SCHEDULER_RUNNING_CHAINS ビューには、各チェーン・ステップ（ネストしたチェーンを含む）に関する 1 行と、各ステップの実行ステータス（NOT_STARTED、RUNNING、STOPPED、SUCCEEDED など）が含まれています。

これらのビューの詳細は、『Oracle Database リファレンス』を参照してください。

停止状態チェーンの処理

ステップの完了時には、常にチェーン・ルールが評価され、次に実行するステップが判別されます。いずれのルールでも別のステップが開始されず、チェーンが終了せず、チェーンの evaluation_interval が NULL の場合は、チェーンが **stalled** 状態になります。チェーンが停止状態になっている場合、実行中のステップはありません。また、（指定の時間間隔だけ待機した後に）実行するようにスケジュールされているステップや、イベントを待機中のイベント・ステップもありません。このチェーンは、ユーザーが手動で操作しないかぎり、先の処理に進むことができません。この場合、チェーンを実行しているジョブの状態は CHAIN_STALLED に設定されます（ただし、ジョブは *_SCHEDULER_RUNNING_JOBS ビューにリストされたままです）。

停止状態のチェーンの問題を解決するには、ビュー ALL_SCHEDULER_RUNNING_CHAINS（チェーン（ネストしたチェーンを含む）内のすべてのステップの状態が示されます）とビュー ALL_SCHEDULER_CHAIN_RULES（すべてのチェーン・ルールが含まれます）が使用されます。

ALTER_RUNNING_CHAIN プロシージャを使用していずれかのステップの state を変更すると、チェーンの実行を継続できます。たとえば、ステップ 11 がステップ 9 の正常終了まで待機してから開始するようになっており、状態の変更がこれに関係する場合、ステップ 9 の state を 'SUCCEEDED' に設定することもできます。

あるいは、1 つ以上のルールが正しくない場合は、DEFINE_CHAIN_RULE プロシージャを使用して、（同じルール名を使用して）それらのルールを置換するか、または新規ルールを作成できます。新規および更新したルールは、実行中のチェーンと次回以降のすべてのチェーン実行に適用されます。ルールの追加または更新後は、停止状態のチェーン・ジョブで EVALUATE_RUNNING_CHAIN を実行し、必要な処理をトリガーする必要があります。

ジョブ間のリソースの割当て

個々のジョブ・レベルでリソース割当てを管理するのは実用的ではないため、スケジューラでは、ジョブ間のリソース割当ての管理にジョブ・クラスの概念が使用されます。ジョブ・クラスの他に、リソース・マネージャも使用されます。

リソース・マネージャを使用したジョブ間のリソース割当て

データベース・リソース・マネージャは、データベース・セッション間のリソース割当て方法を制御します。ジョブなどの非同期セッションのみでなく、ユーザー・セッションなどの同期セッションも制御します。データベース内のすべての作業単位をリソース・コンシューマ・グループにグループ化し、リソース・プランを使用して、様々なグループ間のリソース割当て方法を指定します。リソース・マネージャで制御されるリソースの種類の詳細は、第 25 章「Oracle Database Resource Manager を使用したリソース割当ての管理」を参照してください。

ジョブに対するリソース割当ては、ジョブ・クラスをコンシューマ・グループと関連付けるか、またはジョブ・クラスをデータベース・サービス名と関連付け、そのデータベース・サービスをコンシューマ・グループにマッピングすることによって指定します。ジョブ・クラスのマッピング先となるコンシューマ・グループは、ジョブ・クラスの作成時に指定できます。ジョブ・クラスの作成時にリソース・コンシューマ・グループまたはデータベース・サービス名が指定されていない場合、ジョブ・クラスはデフォルトのコンシューマ・グループにマッピングされます。ジョブ・クラスの `resource_consumer_group` 属性と `service` 属性が設定されているときに、指定のサービスがリソース・コンシューマ・グループにマップされた場合は、`resource_consumer_group` 属性に指定されているリソース・コンシューマ・グループが優先されます。

スケジューラは、完了に必要なリソースを確保できないまま同時に多数のジョブを実行するかわりに、少なくとも一部のジョブを完了できるように、同時に実行するジョブの数を制限しようとします。

スケジューラとリソース・マネージャは緊密に統合されています。ジョブ・コーディネータは、データベース・リソースの可用性をリソース・マネージャから取得します。この情報に基づいて、コーディネータは開始するジョブの数を判断します。実行するための十分なリソースがあるジョブ・クラスのジョブのみを開始します。コーディネータは、コンシューマ・グループにマッピングされている特定ジョブ・クラス内のジョブを開始し続けます。これは、そのコンシューマ・グループに割り当てられているリソースが最大に達したとリソース・マネージャが判断するまで続きます。つまり、実行準備が整っていても、実行に必要なリソースがないためにジョブ・コーディネータによって取り出されないジョブがジョブ表内に存在する可能性があります。したがって、スケジュールされた正確な時間にジョブが実行されることは保証されません。コーディネータは、どのコンシューマ・グループに使用可能なリソースがまだあるかを基準にして、ジョブ表からジョブを取り出します。

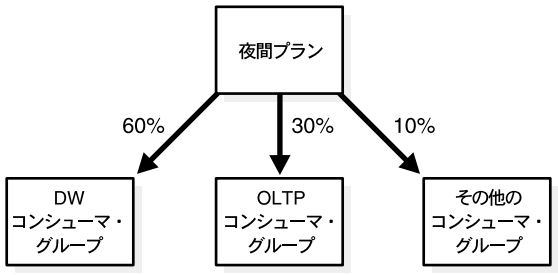
ジョブが実行中の場合も、リソース・マネージャは、指定したリソース・プランに基づいて実行中の各ジョブに割り当てられたリソースの管理を続けます。リソース・マネージャが管理できるのはデータベースのプロセスのみであることに注意してください。リソースのアクティブな管理は、外部ジョブには適用されません。

1 つのデータベースで一度に有効にできるリソース・プランは 1 つのみです。`DBMS_RESOURCE_MANAGER.SWITCH_PLAN` プロシージャを使用すると、システム上のアクティブなリソース・プランを手動で切り替えることができます。特別な場合は、特定のリソース・プランを実行し、ウィンドウのオープンによるリソース・プランの切替えを無効にすることもできます。そのためには、`allow_scheduler_plan_switches` を `FALSE` に設定して `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` プロシージャを使用します。スケジューラのウィンドウには、リソース・プラン属性を指定できることにも注意してください。指定したリソース・プランは、ウィンドウのオープン中はアクティブな状態です。

ジョブに対するリソース割当ての例

次の例では、ジョブに対するリソースの割当て方法について説明します。「夜間プラン」というアクティブなリソース・プランがあり、3 つのジョブ・クラスがあるとします。JC1 はコンシューマ・グループ DW に、JC2 はコンシューマ・グループ OLTP に、JC3 はデフォルトのコンシューマ・グループにそれぞれマッピングされています。図 27-4 に、この例を簡単な図で示します。

図 27-4 リソース・プランの例



このリソース・プランでは、ジョブ・クラス JC1 に属するジョブが明らかに優先されます。コンシューマ・グループ DW はリソースを 60% 取得するため、ジョブ・クラス JC1 に属するジョブはリソースを 60% 取得します。コンシューマ・グループ OLTP はリソースを 30% 取得するため、ジョブ・クラス JC2 内のジョブはリソースを 30% 取得します。コンシューマ・グループ Other では、他のすべてのコンシューマ・グループがリソースを 10% 取得することが指定されます。つまり、ジョブ・クラス JC3 に属するすべてのジョブが、10% のリソースを共有し、リソースを最大 10% 取得できます。

あるコンシューマ・グループで未使用のリソースは、他のコンシューマ・グループで使用できます。したがって、ジョブ・クラス JC1 のジョブが割当ての 60% を完全に使用していない場合、その未使用部分はクラス JC2 と JC3 のジョブが使用できます。リソース・マネージャは、CPU 使用が 100% に達するまで、リソース使用の制限を開始しません。詳細は、第 25 章「Oracle Database Resource Manager を使用したリソース割当ての管理」を参照してください。

28

Oracle Scheduler の管理

この章の内容は次のとおりです。

> **注意：** この章では、DBMS_SCHEDULER パッケージを使用して Oracle Scheduler を管理する方法について説明します。同じ多数の作業を Oracle Enterprise Manager を使用して実行できます。
>
> DBMS_SCHEDULER については『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を、Oracle Scheduler の各ページについては Oracle Enterprise Manager のオンライン・ヘルプを参照してください。

Oracle Scheduler の構成

Oracle Scheduler（スケジューラ）を構成するときは、次のタスクを実行する必要があります。

- タスク 1: スケジューラ権限の設定
- タスク 2: スケジューラ環境の構成

タスク 1: スケジューラ権限の設定

スケジューラを管理するには、SCHEDULER_ADMIN ロールが必要です。通常、データベース管理者（DBA）は、DBA ロール（またはそれと等価のロール）の一部として、ADMIN オプション付きでこのロールをすでに持っています。このロールは、次の文を発行して別の管理者に付与できます。

```
GRANT SCHEDULER_ADMIN TO username;
```

SCHEDULER_ADMIN ロールは、権限を付与されたユーザーがコードを任意のユーザーとして実行できる強力なロールであるため、かわりにスケジューラの個々のシステム権限を付与することを考慮してください。オブジェクト権限とシステム権限は、標準的な SQL 権限付与構文を使用して付与されます。例として、データベース管理者が次の文を発行したとします。

```
GRANT CREATE JOB TO scott;
```

この文が実行されると、scott は、自分のスキーマ内にジョブ、スケジュールまたはプログラムを作成できます。別の例として、データベース管理者が次の文を発行したとします。

```
GRANT MANAGE SCHEDULER TO adam;
```

この文が実行されると、adam は、ウィンドウ、ジョブ・クラスまたはウィンドウ・グループを、作成、変更または削除できます。また、スケジューラ属性の設定と検索、およびスケジューラ・ログのパージもできます。

チェーンの各権限の設定

スケジューラ・チェーンでは、基礎となる Oracle Streams ルール・エンジン・オブジェクトと、それに関連する権限を使用します。ユーザーが自分のスキーマにチェーンを作成するには、自分のスキーマにルール、ルール・セットおよび評価コンテキストを作成するためのルール・エンジン権限に加え、CREATE JOB 権限が必要です。これらの権限は、次の文を発行することで付与できます。

```
BEGIN
DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(DBMS_RULE_ADM.CREATE_RULE_OBJ, 'username'),
DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE (
   DBMS_RULE_ADM.CREATE_RULE_SET_OBJ, 'username'),
DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE (
   DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ, 'username')
END;
/
```

ユーザーが、別のスキーマにチェーンを作成するには、自分のスキーマ以外のスキーマにルール、ルール・セットおよび評価コンテキストを作成するためのルール・エンジン権限に加え、CREATE ANY JOB 権限が必要です。これらの権限は、次の文を発行することで付与できます。

```
BEGIN
DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(DBMS_RULE_ADM.CREATE_ANY_RULE, 'username'),
DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE (
   DBMS_RULE_ADM.CREATE_ANY_RULE_SET, 'username'),
DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE (
   DBMS_RULE_ADM.CREATE_ANY_EVALUATION_CONTEXT, 'username')
END;
/
```

各自のスキーマ以外のスキーマのチェーンを変更または削除するには、ルール、ルール・セットおよび評価コンテキストについて、対応する各システムのルール・エンジン権限が必要です。Streams ルール・エンジン権限の詳細は、DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE の使用方法の説明を参照してください。

関連項目：

- 28-31 ページ「スケジューラ権限」
- チェーンの権限の詳細は、27-50 ページの「チェーンのタスクとそのプロシージャ」を参照してください。

タスク 2: スケジューラ環境の構成

この項では、次のタスクについて説明します。

- タスク 2A: ジョブ・クラスの作成
- タスク 2B: ウィンドウの作成
- タスク 2C: リソース・プランの作成
- タスク 2D: ウィンドウ・グループの作成
- タスク 2E: スケジューラ属性の設定

タスク 2A: ジョブ・クラスの作成

ジョブ・クラスを作成するには、CREATE_JOB_CLASS プロシージャを使用します。次の文は、ジョブ・クラスの作成例です。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name             => 'my_jobclass1',
   resource_consumer_group    => 'my_res_group1',
   comments                   => 'This is my first job class.');
END;
/
```

この文では、属性としてリソース・コンシューマ・グループ my_res_group1 などが設定された my_jobclass1 というジョブ・クラスが作成されます。ジョブ・クラスの内容を確認するには、次の文を発行します。

```
SELECT * FROM DBA_SCHEDULER_JOB_CLASSES;

JOB_CLASS_NAME        RESOURCE_CONSU   SERVICE   LOGGING_LEV  LOG_HISTORY    COMMENTS
----------------      --------------   -------   -----------  -----------    --------
DEFAULT_JOB_CLASS                                      RUNS                 The default
AUTO_TASKS_JOB_CLASS  AUTO_TASK_CON                    RUNS                 System maintenance
FINANCE_JOBS          FINANCE_GROUP                    RUNS
MY_JOBCLASS1          MY_RES_GROUP1                    RUNS                 My first job class
MY_CLASS1                              my_service1     RUNS                 My second job class

5 rows selected.
```

ジョブ・クラスは SYS スキーマ内に作成されることに注意してください。

関連項目： CREATE_JOB_CLASS の構文については『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を、ジョブ・クラスの詳細は 27-29 ページの「ジョブ・クラスの作成」を、およびジョブ・クラスの他の作成例は 28-21 ページの「ジョブ・クラスの作成例」をそれぞれ参照してください。

タスク 2B: ウィンドウの作成

ウィンドウを作成するには、CREATE_WINDOW プロシージャを使用します。次の文は、ウィンドウの作成例です。

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW (
   window_name     =>  'my_window1',
   resource_plan   =>  'my_resourceplan1',
   start_date      =>  '15-APR-03 01.00.00 AM Europe/Lisbon',
   repeat_interval =>  'FREQ=DAILY',
   end_date        =>  '15-SEP-04 01.00.00 AM Europe/Lisbon',
   duration        =>  interval '50' minute,
   window_priority =>  'HIGH',
   comments        =>  'This is my first window.');
END;
/
```

ウィンドウが正しく作成されたことを検証するには、DBA_SCHEDULER_WINDOWS ビューを問い合せます。次に、文の発行例を示します。

```
SELECT WINDOW_NAME, RESOURCE_PLAN, DURATION, REPEAT_INTERVAL
FROM DBA_SCHEDULER_WINDOWS;

WINDOW_NAME    RESOURCE_PLAN     DURATION         REPEAT_INTERVAL
-----------    -------------     -------------    ---------------
MY_WINDOW1     MY_RESOURCEPLAN1  +000 00:50:00    FREQ=DAILY
```

> **関連項目：** CREATE_WINDOW の構文については『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を、ウィンドウの詳細は 27-31 ページの「ウィンドウの作成」を、およびウィンドウの他の作成例は 28-23 ページの「ウィンドウの作成例」をそれぞれ参照してください。

タスク 2C: リソース・プランの作成

リソース・プランを作成するには、CREATE_SIMPLE_PLAN プロシージャを使用します。このプロシージャでは、文を 1 回実行するだけで、コンシューマ・グループを作成し、それらにリソースを割り当てることができます。

次の文は、このプロシージャを使用して、my_simple_plan1 というリソース・プランを作成する例を示しています。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN (
   simple_plan       => 'my_simple_plan1',
   consumer_group1   => 'my_group1',
   group1_cpu        => 80,
   consumer_group2   => 'my_group2',
   group2_cpu        => 20);
END;
/
```

この文では、my_simple_plan1 というリソース・プランが作成されます。リソース・プランの内容を確認するには、DBA_RSRC_PLANS ビューを問い合せます。次に例を示します。

```
SELECT PLAN, STATUS FROM DBA_RSRC_PLANS;

PLAN                           STATUS
------------------------------ -------------------------
SYSTEM_PLAN                    ACTIVE
INTERNAL_QUIESCE               ACTIVE
INTERNAL_PLAN                  ACTIVE
MY_SIMPLE_PLAN1                ACTIVE
```

関連項目： リソース・プランの詳細は、27-62 ページの「ジョブ間のリソースの割当て」を参照してください。

タスク 2D: ウィンドウ・グループの作成

ウィンドウ・グループを作成するには、CREATE_WINDOW_GROUP および ADD_WINDOW_GROUP_MEMBER プロシージャを使用します。次の文は、これらのプロシージャの使用例です。

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW_GROUP (
   group_name        =>  'my_window_group1',
   comments          =>  'This is my first window group.');

DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER (
   group_name        =>  'my_window_group1',
   window_list       =>  'my_window1, my_window2');

DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER (
   group_name        =>  'my_window_group1',
   window_list       =>  'my_window3');
END;
/
```

この文では、my_window2 と my_window3 はすでに作成済であると想定されています。ウィンドウを作成するには、CREATE_WINDOW プロシージャを使用します。

これらの文では、my_window_group1 というウィンドウ・グループが作成された後、そのウィンドウ・グループに my_window1、my_window2 および my_window3 が追加されます。ウィンドウ・グループの内容を確認するには、次の文を発行します。

```
SELECT * FROM DBA_SCHEDULER_WINDOW_GROUPS;

WINDOW_GROUP_NAME   ENABLED  NUMBER_OF_WINDOWS   COMMENTS
-----------------   -------  -----------------   --------------------
MY_WINDOW_GROUP1    TRUE                     3   This is my first window group.

SELECT * FROM DBA_SCHEDULER_WINGROUP_MEMBERS;

WINDOW_GROUP_NAME              WINDOW_NAME
------------------------------ ---------------
MY_WINDOW_GROUP1               MY_WINDOW1
MY_WINDOW_GROUP1               MY_WINDOW2
MY_WINDOW_GROUP1               MY_WINDOW3
```

関連項目： CREATE_WINDOW_GROUP の構文については『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を、ウィンドウ・グループの詳細は 27-38 ページの「ウィンドウ・グループの使用」を、およびウィンドウ・グループの詳細な作成例は、28-24 ページの「ウィンドウ・グループの作成例」をそれぞれ参照してください。

タスク 2E: スケジューラ属性の設定

スケジューラの動作を制御するスケジューラ属性がいくつかあります。それらは、default_timezone、log_history、max_job_slave_processes および event_expiry_time です。これらの属性の値は、SET_SCHEDULER_ATTRIBUTE プロシージャを使用して設定できます。これらの属性を設定するには、MANAGE SCHEDULER 権限が必要です。設定可能な属性は、次のとおりです。

- default_timezone

  カレンダ指定構文を使用したジョブとウィンドウの繰返しの場合は、それぞれの繰返し間隔で使用されるタイム・ゾーンを認識している必要があります。タイム・ゾーンは、通常 start_date から取得されます。ただし、start_date が設定されていない場合（通常の状態）、タイム・ゾーンは default_timezone スケジューラ属性から取得されます。

  スケジューラは、default_timezone の値をオペレーティング・システム環境から導出します。オペレーティング・システムから互換性のある値を検出できない場合、スケジューラは default_timezone を NULL に設定します。

  default_timezone が正しく設定されているかどうかの確認は非常に重要です。正しく設定されていない場合は、設定する必要があります。確認には、次の問合せを実行します。

  ```
  SQL> select dbms_scheduler.stime from dual;

  STIME
  ---------------------------------------------------------------------------
  14-OCT-04 02.56.03.206273000 PM US/PACIFIC
  ```

  確実に夏時間が適用されるようにするには、default_timezone を、絶対タイム・ゾーン・オフセットではなく、リージョン名に設定することをお薦めします。たとえば、データベースが米国フロリダ州のマイアミにある場合は、次の文を発行します。

  ```
  DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('default_timezone','US/Eastern');
  ```

  有効なリージョン名のリストを表示するには、次の問合せを実行します。

  ```
  SELECT DISTINCT TZNAME FROM V$TIMEZONE_NAMES;
  ```

  default_timezone 属性を適切に設定しないと、繰返しジョブや繰返しウィンドウに使用されるデフォルトのタイム・ゾーンは、SYSTIMESTAMP から取り出される絶対オフセット（データベースのオペレーティング・システム環境のタイム・ゾーン）となります。これは、start_date が設定されていない繰返しジョブや繰返しウィンドウでは、夏時間調整が適用されないことになります。

- log_history

  この属性を使用すると、スケジューラが実行するロギングの量を制御できます。ジョブ・ログおよびウィンドウ・ログが無計画に大きくならないように、スケジューラには、保持する履歴の量（日数）を指定する属性があります。1 日に 1 回、スケジューラは、指定した履歴より古いすべてのログ・エントリをジョブ・ログとウィンドウ・ログから自動的にパージします。デフォルト値は 30 日です。

  デフォルト値を変更するには、SET_SCHEDULER_ATTRIBUTE プロシージャを使用します。たとえば、90 日に変更する場合は、次の文を発行します。

  ```
  DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('log_history','90');
  ```

  有効値の範囲は 1 ～ 999 です。

- max_job_slave_processes

  この属性を使用すると、特定のシステム構成と負荷に対する最大スレーブ・プロセス数を設定できます。スケジューラは、所定のシステム構成と負荷に対する最適なスレーブ・プロセス数を自動的に判別しますが、スケジューラに固定の制限を設定することもできます。制限を設定する場合に、この属性を設定できます。デフォルト値は NULL で、有効値の範囲は 1 ～ 999 です。

  max_job_slave_processes によって設定された数が実際の最大数ですが、指定された数のスレーブをスケジューラが開始するという意味ではありません。たとえば、この属性が 10 に設定されている場合でも、スケジューラは 4 個以上のスレーブ・プロセスを開始しない方がよいと判断する場合があります。ただし、スケジューラが 15 個のスレーブ・プロセスの開始を計画している場合に、最大数が 10 に設定されていると、11 個以上のプロセスは開始しません。

- event_expiry_time

  この属性を使用すると、スケジューラが生成したイベントが終了する（つまり、自動的にキューからパージされる）までの時間（秒）を設定できます。

SET_SCHEDULER_ATTRIBUTE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

スケジューラの監視と管理

次の項では、スケジューラを監視および管理する方法について説明します。

- 現在アクティブなウィンドウとリソース・プランの表示
- 現在実行中のジョブに関する情報の検索
- ウィンドウ・ログおよびジョブ・ログの監視と管理
- ジョブ優先度の変更
- スケジューラ・セキュリティの管理

現在アクティブなウィンドウとリソース・プランの表示

現在アクティブなウィンドウとそのウィンドウに関連付けられているプランを表示するには、次の文を発行します。

```
SELECT WINDOW_NAME, RESOURCE_PLAN FROM DBA_SCHEDULER_WINDOWS
WHERE ACTIVE='TRUE';

WINDOW_NAME                    RESOURCE_PLAN
------------------------------ -------------------------
MY_WINDOW10                    MY_RESOURCEPLAN1
```

アクティブなウィンドウがない場合は、次の文を発行するとアクティブなリソース・プランを表示できます。

```
SELECT * FROM V$RSRC_PLAN;
```

現在実行中のジョブに関する情報の検索

ジョブの状態を確認するには、次の文を発行します。

```
SELECT JOB_NAME, STATE FROM DBA_SCHEDULER_JOBS
WHERE JOB_NAME = 'MY_EMP_JOB1';

JOB_NAME                       STATE
------------------------------ ---------
MY_EMP_JOB1                    DISABLED
```

この場合は、ENABLE プロシージャを使用してジョブを使用可能にできます。表 28-1 に、ジョブの状態に関する有効値を示します。

表 28-1 ジョブの状態

ジョブの状態	説明
disabled	ジョブは使用禁止です。
scheduled	ジョブは実行するようにスケジュールされています。
実行	ジョブは現在実行中です。
completed	ジョブは完了しており、次回の実行はスケジュールされていません。
stopped	ジョブは 1 回実行するようにスケジュールされましたが、実行中に停止しました。
broken	ジョブは中断されました。
failed	ジョブは 1 回実行するようにスケジュールされ、失敗しました。
retry scheduled	ジョブは 1 回以上失敗し、再試行の実行がスケジュールされました。
succeeded	ジョブは 1 回実行するようにスケジュールされ、正常に完了しました。
chain_stalled	ジョブのタイプはチェーンであり、実行中のステップ、実行するようにスケジュールされているステップ、およびイベントを待機しているイベント・ステップはありません。また、チェーン evaluation_interval は NULL に設定されています。手動で介入しないかぎり、チェーンの処理は先に進みません。

現在実行中のジョブの進行状況を確認するには、次の文を発行します。

```
SELECT * FROM ALL_SCHEDULER_RUNNING_JOBS;
```

CPU_USED 列に有効なデータを表示するには、RESOURCE_LIMIT 初期化パラメータを true に設定する必要があります。

実行中のチェーンの一部のジョブに関する情報を得るには、次の文を発行します。

```
SELECT * FROM ALL_SCHEDULER_RUNNING_CHAINS WHERE JOB_NAME='MY_JOB1';
```

cjqNNN の形式のプロセスを検索すると、ジョブ・コーディネータが実行中かどうかをチェックできます。

> **関連項目：** *_SCHEDULER_RUNNING_JOBS および DBA_SCHEDULER_JOBS ビューの詳細は、『Oracle Database リファレンス』を参照してください。

ウィンドウ・ログおよびジョブ・ログの監視と管理

スケジューラは、ジョブ・ログとウィンドウ・ログという 2 種類のログをサポートしています。

ジョブ・ログ

ジョブの実行、ジョブ・ステータスの変化およびジョブの失敗について、ジョブ・ログ内の情報を表示できます。ジョブ・ログは、次の 2 つのデータ・ディクショナリ・ビューとして実装されます。

- *_SCHEDULER_JOB_LOG
- *_SCHEDULER_JOB_RUN_DETAILS

スケジューラがジョブに関して実行するロギングの量は、ジョブ・クラス・レベルと個別ジョブ・レベルの両方で制御できます。通常、クラス内のジョブに関するロギングを厳密に制御できるため、ロギングをクラス・レベルで制御します。

各種ロギング・レベルの定義と、ジョブとジョブ・クラスの間のロギング・レベルの優先度については、27-16 ページの「ジョブ・ログの表示」を参照してください。デフォルトでは、ジョブ・クラスのロギング・レベルは LOGGING_RUNS で、すべてのジョブの実行がログに記録されます。

ジョブ・クラスの作成時に logging_level 属性を設定するか、後で SET_ATTRIBUTE プロシージャを使用してロギング・レベルを変更できます。次の例では、myclass1 ジョブ・クラス内のジョブのロギング・レベルを LOGGING_FAILED_RUNS に設定しています。これは、異常終了した実行のみがログに記録されることを意味します。すべてのジョブ・クラスは SYS スキーマ内に作成されることに注意してください。

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   'sys.myclass1', 'logging_level', DBMS_SCHEDULER.LOGGING_FAILED_RUNS);
END;
```

ジョブ・クラスのロギング・レベルを設定するには、MANAGE SCHEDULER 権限が付与されている必要があります。

関連項目：

- ジョブ・ログの詳細と、ジョブ・ログ・ビューに対する問合せ例については、27-16 ページの「ジョブ・ログの表示」を参照してください。
- ジョブ・ログ・ビューの詳細は、『Oracle Database リファレンス』を参照してください。
- CREATE_JOB_CLASS および SET_ATTRIBUTE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- ログ・エントリの保存期間の設定については、28-6 ページの「タスク 2E: スケジューラ属性の設定」を参照してください。

ウィンドウ・ログ

スケジューラでは、次の操作が実行されるたびにウィンドウ・ログにエントリが 1 つ記録されます。

- ウィンドウの作成または削除
- ウィンドウのオープン
- ウィンドウのクローズ
- ウィンドウの重複
- ウィンドウの使用可能化または使用禁止

ウィンドウ・アクティビティのロギングには、ロギング・レベルはありません。

ウィンドウ・ログの内容を調べるには、DBA_SCHEDULER_WINDOW_LOG ビューを問い合せます。次の文は、このビューの出力例を示しています。

```
SELECT log_id, to_char(log_date, 'DD-MON-YY HH24:MM:SS') timestamp,
   window_name, operation FROM DBA_SCHEDULER_WINDOW_LOG;

    LOG_ID TIMESTAMP            WINDOW_NAME       OPERATION
---------- -------------------- ----------------- --------
         4 10/01/2004 15:29:23  WEEKEND_WINDOW    CREATE
         5 10/01/2004 15:33:01  WEEKEND_WINDOW    UPDATE
        22 10/06/2004 22:02:48  WEEKNIGHT_WINDOW  OPEN
        25 10/07/2004 06:59:37  WEEKNIGHT_WINDOW  CLOSE
        26 10/07/2004 22:01:37  WEEKNIGHT_WINDOW  OPEN
        29 10/08/2004 06:59:51  WEEKNIGHT_WINDOW  CLOSE
```

DBA_SCHEDULER_WINDOWS_DETAILS ビューは、以前はアクティブで現在はクローズ（完了）されているすべてのウィンドウに関する情報を提供します。次の文は、このビューの出力例を示しています。

```
SELECT LOG_ID, WINDOW_NAME, ACTUAL_START_DATE, ACTUAL_DURATION
  FROM DBA_SCHEDULER_WINDOW_DETAILS;

    LOG_ID WINDOW_NAME      ACTUAL_START_DATE                    ACTUAL_DURATION
---------- ---------------- ---------------------------------- ---------------
        25 WEEKNIGHT_WINDOW 06-OCT-04 10:02.48.832438 PM PST8PDT +000 01:02:32
        29 WEEKNIGHT_WINDOW 07-OCT-04 10.01.37.025704 PM PST8PDT +000 03:02:00
```

これらのビューの両方でログ ID が対応しており、この例では、DBA_SCHEDULER_WINDOWS_DETAILS ビューの行が DBA_SCHEDULER_WINDOW_LOG ビューの CLOSE 操作に対応しています。

関連項目：

- ウィンドウ・ログ・ビューの詳細は、『Oracle Database リファレンス』を参照してください。

ログのパージ

ジョブ・ログおよびウィンドウ・ログが無計画に大きくならないように、SET_SCHEDULER_ATTRIBUTE プロシージャを使用して、保持する履歴の量（日数）を指定します。1 日に 1 回、スケジューラは、指定した履歴期間より古いすべてのログ・エントリをジョブ・ログとウィンドウ・ログから自動的にパージします。デフォルトの履歴期間は 30 日です。たとえば、履歴期間を 90 日に変更する場合は、次の文を発行します。

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('log_history','90');
```

一部のジョブ・クラスは他のジョブ・クラスより重要です。そのため、このグローバルな履歴設定は、クラス別の設定を使用して上書きできます。例として、3 つのジョブ・クラス（class1、class2 および class3）があるとします。ウィンドウ・ログおよび class1 と class3 については履歴を 10 日間保持し、class2 については 30 日間保持するとします。このように設定するには、次の文を発行します。

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('log_history','10');
DBMS_SCHEDULER.SET_ATTRIBUTE('class2','log_history','30');
```

ジョブ・クラスの作成時には、クラス別の履歴を設定することもできます。

チェーン実行のステップに関連したログ・エントリは、メインのチェーン・ジョブのエントリがパージされるまでパージされません。

手動でのログのパージ

PURGE_LOG プロシージャを使用して、ログを手動でパージできます。例として、ジョブ・ログとウィンドウ・ログの両方からすべてのエントリをパージする文を示します。

```
DBMS_SCHEDULER.PURGE_LOG();
```

別の例として、3 日前より古いすべてのエントリをジョブ・ログからパージする文を示します。ウィンドウ・ログはこの文の影響を受けません。

```
DBMS_SCHEDULER.PURGE_LOG(log_history => 3, which_log => 'JOB_LOG');
```

次の文では、10 日前より古いすべてのウィンドウ・ログ・エントリと、job1 および class2 内のジョブに関連する 10 日前より古いすべてのジョブ・ログ・エントリがパージされます。

```
DBMS_SCHEDULER.PURGE_LOG(log_history => 10, job_name => 'job1, sys.class2');
```

ジョブ優先度の変更

ジョブの優先度を変更するには、SET_ATTRIBUTE プロシージャを使用します。ジョブの優先度は、1 を最も高い優先度として 1 ～ 5 の範囲で設定する必要があります。たとえば、次の文では、my_job1 のジョブ優先度の設定が 1 に変更されます。

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           =>   'my_emp_job1',
   attribute      =>   'job_priority',
   value          =>   1);
END;
/
```

属性が変更されたことを確認するには、次の文を発行します。

```
SELECT JOB_NAME, JOB_PRIORITY FROM DBA_SCHEDULER_JOBS;

JOB_NAME                       JOB_PRIORITY
------------------------------ ------------
MY_EMP_JOB                                3
MY_EMP_JOB1                               1
MY_NEW_JOB1                               3
MY_NEW_JOB2                               3
MY_NEW_JOB3                               3
```

関連項目： SET_ATTRIBUTE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

スケジューラ・セキュリティの管理

ジョブのスケジュールと実行にスケジューラを使用する必要がある通常のユーザーに対しては、CREATE JOB システム権限を付与してください。システム・リソースを管理する必要があるデータベース管理者には、MANAGE SCHEDULER を付与してください。スケジューラに関するその他のシステム権限またはロールを付与するときは、十分に注意してください。特に、CREATE ANY JOB システム権限と、この権限が含まれる SCHEDULER_ADMIN ロールは、コードを任意のユーザーで実行できるため、非常に強力です。これらの権限とロールの付与は、非常に強力な権限を持つロールまたはユーザーに限定してください。

セキュリティに関して特に重要な問題は、外部ジョブの処理です。外部ジョブの処理は、データベース外でジョブを実行する必要があるユーザーにのみ実行を許可してください。それらのユーザーには、CREATE EXTERNAL JOB システム権限を付与する必要があります。詳細は、27-6 ページの「外部ジョブの作成」を参照してください。スケジューラに関するセキュリティには、他に特別な要件はありません。セキュリティの詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

> **注意：** Oracle Database 10g リリース 1 から 10g リリース 2 にアップグレードすると、CREATE EXTERNAL JOB が CREATE JOB 権限を持つすべてのユーザーとロールに対して自動的に付与されます。必要のないユーザーに対してはこの権限を取り消すことをお薦めします。

リモート外部ジョブの有効化と無効化

Oracle Scheduler（スケジューラ）では、リモート・ホストで外部ジョブをスケジュールし、実行できます。リモート・ホストに Oracle データベースがインストールされている必要はありませんが、スケジューラ・エージェントはインストールされている必要があります。これによって、スケジューリングするデータベースは、リモート・ホストでリモート外部ジョブを開始し、ジョブの出力とエラー情報を取得できます。エージェントは、そのエージェントのホスト・コンピュータでリモート外部ジョブを開始する必要のあるすべてのデータベースに登録する必要があります。リモート外部ジョブを実行する必要のある各データベースには、初期設定も必要です。この設定によって、データベースとリモートのスケジューラ・エージェント間でのセキュアな通信が可能になります。

リモート外部ジョブの有効化には、次の手順が含まれます。

1. リモート外部ジョブを実行するためのデータベースの設定
2. スケジューラ・エージェントのインストール、構成および起動

この項では、次の内容も説明します。

- スケジューラ・エージェントの停止
- リモート外部ジョブの無効化

> **関連項目：** 26-10 ページ「リモート外部ジョブの概要」

リモート外部ジョブを実行するためのデータベースの設定

データベースでリモートのスケジューラ・エージェントを使用してジョブを実行するには、その前に、データベースを正しく構成し、そのエージェントをデータベースに登録しておく必要があります。リモートのスケジューラ・エージェントをセキュアに登録するには、エージェントの登録パスワードをデータベースに設定する必要があります。登録できるスケジューラ・エージェントの数を制限して、パスワードの有効期限を設定できます。

リモート外部ジョブを実行する必要のある各データベースに対して、次の手順を 1 回実行します。

リモート外部ジョブを実行するためのデータベースの設定手順

1. SQL*Plus を使用して、SYS ユーザーとしてデータベースに接続します。

   手順については、1-8 ページの「SQL*Plus を使用したデータベースへの接続」を参照してください。

2. 次のコマンドを入力して、XML DB オプションがインストールされていることを確認します。

   ```
   SQL> DESC RESOURCE_VIEW
   ```

   XML DB がインストールされていない場合は、このコマンドで「オブジェクトが存在しません。」エラーが戻ります。

   > **注意：** XML DB がインストールされていない場合は、先に進む前に XML DB をインストールする必要があります。

3. 次の PL/SQL ブロックを発行して、データベースへの HTTP 接続を使用可能にします。

   ```
   BEGIN
     DBMS_XDB.SETHTTPPORT(port);
   END;
   ```

   port は、データベースが HTTP 接続をリスニングする TCP ポート番号です。

   port には 1 ～ 65536 の整数を指定する必要があります。UNIX および Linux の場合は、1023 より大きい値にする必要があります。まだ使用されていないポート番号を選択してください。

4. 次のコマンドを使用して、スクリプト prvtrsch.plb を実行します。

   ```
   SQL> @?/rdbms/admin/prvtrsch.plb
   ```

5. SET_AGENT_REGISTRATION_PASS プロシージャを使用して、スケジューラ・エージェントの登録パスワードを設定します。

   次の例では、エージェント登録パスワードを mypassword に設定します。

   ```
   BEGIN
     DBMS_SCHEDULER.SET_AGENT_REGISTRATION_PASS('mypassword');
   END;
   ```

   > **注意：** エージェント登録パスワードを設定するには、MANAGE SCHEDULER 権限が必要です。SET_AGENT_REGISTRATION_PASS プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

スケジューラ・エージェントのインストール、構成および起動

特定のホストでリモート外部ジョブを実行するには、その前に、そのホスト上にスケジューラ・エージェントをインストールし、構成して起動しておく必要があります。スケジューラ・エージェントは、独自の Oracle ホームにインストールする必要があります。

Windows、Linux および UNIX プラットフォームの場合、スケジューラ・エージェント・ソフトウェアは Oracle Database ゲートウェイのインストール・メディア（Database のインストール・メディア）に収録されており、次のサイトからオンラインで入手することもできます。

```
http://www.oracle.com/technology/software/products/database
```

IBM z/OS および IBM iSeries OS/400 など、他のプラットフォーム用のエージェント・ソフトウェアは、該当するプラットフォーム用の Oracle Scheduler Agent のインストール・メディアに収録されています。これらのプラットフォーム上でエージェントをインストールするには、プラットフォーム固有のマニュアルを参照してください。

リモートの Windows、Linux または UNIX ホストでのスケジューラ・エージェントのインストール、構成および起動の手順は、次のとおりです。

1. リモート・ホストにログインします。
   - Windows の場合は、管理者としてログインします。
   - UNIX および Linux の場合は、スケジューラ・エージェントを実行する予定のユーザーとしてログインします。このユーザーに特別な権限は必要ありません。
2. Oracle Database ゲートウェイ製品のインストール・メディアから Oracle Universal Installer（OUI）を実行します。
   - Windows の場合は、`setup.exe` を実行します。
   - UNIX および Linux の場合は、次のコマンドを使用します。

     /*directory_path*/runInstaller

   directory_path は、Oracle Database ゲートウェイ製品のインストール・メディアへのパスです。
3. OUI の「ようこそ」画面で**「次へ」**をクリックします。
4. 「製品の選択」ページで、**「Oracle Scheduler Agent」**を選択して、**「次へ」**をクリックします。
5. 「ホームの詳細の指定」ページで、エージェントの新しい Oracle ホームの名前とパスを入力して、**「次へ」**をクリックします。
6. 「Oracle Scheduler Agent」ページで、次の手順を実行します。
   a. 「Scheduler Agent のホスト名」フィールドに、スケジューラ・エージェントが実行されるコンピュータのホスト名を入力するか、デフォルトのホスト名をそのまま使用します。
   b. 「Scheduler Agent のポート番号」フィールドに、スケジューラ・エージェントが接続をリスニングする TCP ポート番号を入力して、**「次へ」**をクリックします。

      1 ～ 65535 の整数を選択します。UNIX および Linux の場合は、1023 より大きい値にする必要があります。選択するポート番号がまだ使用されていないことを確認します。
7. 「サマリー」ページで、**「インストール」**をクリックします。
8. （UNIX および Linux のみ）Oracle Universal Installer がスクリプト `root.sh` の実行を求めるプロンプトを表示したら、`root` ユーザーとして次のコマンドを入力します。

   script_path/root.sh

   このスクリプトは、エージェントのインストール用に選択したディレクトリに格納されています。
9. インストールが完了したら、OUI を終了します。

10. テキスト・エディタを使用して、スケジューラ・エージェントのホーム・ディレクトリにあるエージェント構成パラメータ・ファイル `schagent.conf` の `PORT=` ディレクティブのポート番号を確認します。

    このポート番号はリモート外部ジョブの作成時に必要になります。必要に応じて、他のエージェント構成パラメータを変更します。

11. リモート・ホスト上のファイアウォール・ソフトウェア、またはそのホストを保護する他のファイアウォールに、スケジューラ・エージェントに対応するための例外があることを確認します。

12. エージェントのホスト・コンピュータでリモート外部ジョブを実行するデータベースに、スケジューラ・エージェントを登録します。次のコマンドを使用します。

    ```
    AGENT_HOME/bin/schagent -registerdatabase db_host db_http_port
    ```

    各項目の意味は次のとおりです。

    - *db_host* は、データベースがあるホストのホスト名または IP アドレスです。
    - *db_http_port* は、データベースがリスニングするポート番号です（HTTP 接続用）。このパラメータは以前に 28-13 ページの「リモート外部ジョブを実行するためのデータベースの設定」で設定しました。次の SQL 文をデータベースに発行することにより、ポート番号を確認できます。

      ```
      SELECT DBMS_XDB.GETHTTPPORT() FROM DUAL;
      ```

      ポート番号が 0 のときは、HTTP 接続が無効になっています。

    28-13 ページの「リモート外部ジョブを実行するためのデータベースの設定」で設定したエージェント登録パスワードの入力を求めるプロンプトがエージェントによって表示されます。

13. エージェントのホストでリモート外部ジョブを実行する各データベースについて、前述の手順を繰り返します。

14. （UNIX および Linux のみ）次のコマンドを使用してスケジューラ・エージェントを起動します。

    ```
    AGENT_HOME/bin/schagent -start
    ```

注意： Windows では、インストール時にスケジューラ・エージェント・サービスが自動的に作成されて起動されます。サービス名の末尾は、`OracleSchedulerExecutionAgent` になっています。Oracle データベースがインストールされている Windows コンピュータ上で実行され、資格証明なしのローカル外部ジョブの実行を管理する `OracleJobScheduler` サービスと、このサービスを混同しないでください。

スケジューラ・エージェントの停止

スケジューラ・エージェントを停止すると、スケジューラ・エージェントがあるホストでリモート外部ジョブを実行できなくなります。

スケジューラ・エージェントを停止する手順は、次のとおりです。

- 次のいずれかを実行します。
  - UNIX および Linux の場合は、次のコマンドを実行します。

    ```
    AGENT_HOME/bin/schagent -stop
    ```

  - Windows では、OracleSchedulerExecutionAgent で終わる名前のサービスを停止します。

リモート外部ジョブの無効化

データベースでリモート外部ジョブを実行する機能は、REMOTE_SCHEDULER_AGENT ユーザーを削除することで、無効（使用禁止）にできます。

リモート外部ジョブを無効にする手順は、次のとおりです。

- 次の SQL 文を発行します。

  ```
  DROP USER REMOTE_SCHEDULER_AGENT CASCADE;
  ```

新規スケジューラ・エージェントの登録とリモート外部ジョブの実行は、prvtrsch.plb を再度実行するまで無効（使用禁止）です。

スケジューラのインポート / エクスポート

スケジューラ・オブジェクトをエクスポートするには、データ・ポンプ・ユーティリティ（impdp および expdp）を使用します。スケジューラでは、以前のインポートおよびエクスポート・ユーティリティ（IMP および EXP）は使用できません。また、スケジューラ・オブジェクトは、データベースが読取り専用モードの間はエクスポートできません。

エクスポートでは、スケジューラ・オブジェクトの作成に使用された DDL が生成されます。すべての属性がエクスポートされます。インポートが実行されると、すべてのデータベース・オブジェクトが新規データベースに再作成されます。すべてのスケジュールは、それぞれのタイム・ゾーンで格納され、新規データベース内で保持されます。たとえば、「サンフランシスコにあるデータベースの月曜日午後 1 時（太平洋沿岸標準時）」というスケジュールは、エクスポートされドイツでデータベースにインポートされた場合でも同じです。

スケジューラの資格証明はエクスポートされますが、セキュリティ上の理由で、これらの資格証明のパスワードはエクスポートされません。スケジューラの資格証明をインポートした後は、DBMS_SCHEDULER パッケージの SET_ATTRIBUTE プロシージャを使用してパスワードを再設定する必要があります。

> **関連項目：** データ・ポンプの詳細は、『Oracle Database ユーティリティ』を参照してください。

スケジューラのトラブルシューティング

ここでは、トラブルシューティングに関する情報を提供します。この項の内容は、次のとおりです。

- ジョブの実行に失敗する理由
- 障害後のジョブ・リカバリ
- プログラムが使用禁止になる理由
- ウィンドウの有効化に失敗する理由

ジョブの実行に失敗する理由

ジョブはいくつかの理由で実行に失敗する場合があります。実行されなかった可能性のあるジョブのトラブルシューティングを行う前に、次の文を発行し、ジョブが実行されていないことを確認してください。

```
SELECT JOB_NAME, STATE FROM DBA_SCHEDULER_JOBS;
```

標準的な出力は次のようになります。

```
JOB_NAME                       STATE
------------------------------ ---------
MY_EMP_JOB                     DISABLED
MY_EMP_JOB1                    FAILED
MY_NEW_JOB1                    DISABLED
MY_NEW_JOB2                    BROKEN
MY_NEW_JOB3                    COMPLETED
```

実行中でないジョブには次の 4 つのタイプがあります。

- 失敗したジョブ
- 中断されたジョブ
- 使用禁止のジョブ
- 完了したジョブ

関連項目： 28-18 ページ「障害後のジョブ・リカバリ」

失敗したジョブ

ジョブ表内のジョブのステータスが FAILED の場合、そのジョブは実行がスケジュールされたが実行に失敗したことを示します。ジョブが再起動可能として指定されていた場合は、すべての再試行に失敗しています。

実行の途中でジョブが失敗すると、最後のトランザクションのみがロールバックされます。複数のトランザクションを実行するジョブの場合は、restartable を TRUE に設定するときに注意する必要があります。失敗したジョブを問い合せるには、*_SCHEDULER_JOB_RUN_DETAILS ビューを問い合せます。

中断されたジョブ

中断されたジョブとは、特定の失敗数を超えたジョブです。この数は max_failures で設定され、変更できます。中断されたジョブは、ジョブ全体が中断されており、修正されるまで実行されません。デバッグとテストには、RUN_JOB プロシージャを使用できます。

中断されたジョブを問い合せるには、*_SCHEDULER_JOBS および *_SCHEDULER_JOB_LOG ビューを問い合せます。

使用禁止のジョブ

ジョブは次の理由で使用禁止になる場合があります。

- 手動で使用禁止にされた場合
- ジョブが属しているジョブ・クラスが削除された場合
- ジョブが指し示しているプログラム、チェーンまたはスケジュールが削除された場合
- ウィンドウまたはウィンドウ・グループがジョブのスケジュールとなっているときに、そのウィンドウまたはウィンドウ・グループが削除された場合

完了したジョブ

ジョブは、end_date または max_runs に達すると完了します。(最近正常に完了したジョブが再度実行するようにスケジュールされると、ジョブの状態は SCHEDULED になります。)

障害後のジョブ・リカバリ

スケジューラは、次の場合に中断されたジョブのリカバリを試行します。

- データベースが異常停止した場合
- ジョブ・スレーブ・プロセスが強制終了したか、失敗した場合
- 外部ジョブの場合に、実行可能ファイルまたはスクリプトを起動する外部ジョブ・プロセスが強制終了したか、失敗した場合(外部ジョブ・プロセスは、Unix では extjob です。Windows では外部ジョブ・サービスです。)
- 外部ジョブの場合に、エンド・ユーザーの実行可能ファイルまたはスクリプトを起動するプロセスが強制終了したか、失敗した場合

ジョブ・リカバリは次のように実行されます。

- スケジューラにより、障害が発生したときに実行されていたジョブのインスタンスについて、ジョブ・ログにエントリが追加されます。ログ・エントリでは、OPERATION は RUN、STATUS は STOPPED になっており、ADDITIONAL_INFO には次のいずれかが挿入されます。
  - REASON= ジョブ・スレーブ・プロセスは終了しました
  - REASON=ORA-01014: Oracle のシャットダウン処理中です。
- そのジョブの restartable が TRUE に設定されている場合は、ジョブが再開されます。
- restartable が FALSE に設定されている場合は、次のようになります。
  - ジョブが 1 回のみ実行されるジョブであり、auto_drop が TRUE に設定されている場合は、ジョブの実行が完了するとそのジョブは削除されます。
  - ジョブが 1 回のみ実行されるジョブであり、auto_drop が FALSE に設定されている場合は、ジョブが使用禁止になり、そのジョブの状態 state は STOPPED に設定されます。
  - ジョブが繰返しジョブの場合、スケジューラは次回のジョブの実行をスケジュールし、そのジョブの状態 state は SCHEDULED に設定されます。

このリカバリ・プロセスの結果、ジョブが再開されると、新規実行は RECOVERY_RUN 操作としてジョブ・ログに記録されます。

プログラムが使用禁止になる理由

プログラムは、プログラム引数が削除された場合や number_of_arguments の変更ですべての引数を定義できなくなった場合に使用禁止になります。

プログラムの詳細は、27-19 ページの「プログラムの使用」を参照してください。

ウィンドウの有効化に失敗する理由

ウィンドウは、次の場合に有効化できません。

- スケジュールの終了時にウィンドウが使用禁止にされた場合
- すでに存在しないスケジュールを指すウィンドウが使用禁止にされた場合

ウィンドウの詳細は、27-30 ページの「ウィンドウの使用」を参照してください。

スケジューラの使用例

この項の内容は、次のとおりです。

ジョブの作成例

この項では、ジョブの作成例をいくつか示します。ジョブを作成するには、CREATE_JOB または CREATE_JOBS プロシージャを使用します。

例 28-1 単一の標準ジョブの作成

次の文では、oe スキーマ内に my_job1 という単一の標準ジョブが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name             => 'oe.my_job1',
   job_type             => 'PLSQL_BLOCK',
   job_action           => 'BEGIN DBMS_STATS.GATHER_TABLE_STATS(''oe'',
                            ''sales''); END;',
   start_date           => '15-JUL-08 1.00.00AM US/Pacific',
   repeat_interval      => 'FREQ=DAILY',
   end_date             => '15-SEP-08 1.00.00AM US/Pacific',
   enabled              =>  TRUE,
   comments             => 'Gather table statistics');
END;
/
```

このジョブは、sales 表に関して表の統計を収集します。開始日は 7 月 15 日で、1 日に 1 回実行され、終了日は 9 月 15 日です。ジョブが作成されたことを確認するには、次の文を発行します。

```
SELECT JOB_NAME FROM DBA_SCHEDULER_JOBS WHERE JOB_NAME = 'MY_JOB1';

JOB_NAME
------------------------------
MY_JOB1
```

例 28-2 単一トランザクションでの一連の軽量ジョブの作成例

次の例では、1 つのトランザクションで一連の軽量ジョブが作成されます。

```
DECLARE
 newjob sys.job;
 newjobarr sys.job_array;
BEGIN
 -- To create a lightweight job, the program must be enabled.
 -- The program action must be a PL/SQL block or stored procedure.
 DBMS_SCHEDULER.ENABLE('PROG1');

 newjobarr := sys.job_array();
 newjobarr.extend(5);
 FOR i IN 1..5 LOOP
   newjob := sys.job(job_name => 'LWJOB' || to_char(i),
                     job_style => 'LIGHTWEIGHT',
                     job_template => 'PROG1',
                     repeat_interval => 'FREQ=MINUTELY;INTERVAL=3',
                     start_date => systimestamp + interval '10' second,
                     enabled => TRUE
                    );
   newjobarr(i) := newjob;
 end loop;

 DBMS_SCHEDULER.CREATE_JOBS(newjobarr, 'TRANSACTIONAL');
END;
/
```

関連項目： CREATE_JOB プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。また、27-3 ページの「ジョブの作成」を参照してください。

ジョブ・クラスの作成例

この項では、ジョブ・クラスの作成例をいくつか示します。ジョブ・クラスを作成するには、CREATE_JOB_CLASS プロシージャを使用します。

例 28-3 ジョブ・クラスの作成

次の文ではジョブ・クラスが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name             =>  'my_class1',
   service                    =>  'my_service1',
   comments                   =>  'This is my first job class');
END;
/
```

この文では、SYS 内に my_class1 が作成されます。my_service1 というサービスが使用されます。ジョブ・クラスが作成されたことを確認するには、次の文を発行します。

```
SELECT JOB_CLASS_NAME FROM DBA_SCHEDULER_JOB_CLASSES
WHERE JOB_CLASS_NAME = 'MY_CLASS1';

JOB_CLASS_NAME
------------------------------
MY_CLASS1
```

例 28-4 ジョブ・クラスの作成

次の文ではジョブ・クラスが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name             =>  'finance_jobs',
   resource_consumer_group    =>  'finance_group',
   service                    =>  'accounting',
   comments                   =>  'All finance jobs');
END;
/
```

この文では、SYS 内に finance_jobs が作成されます。この文では、finance_group というリソース・コンシューマ・グループが割り当てられ、accounting サービスのサービス・アフィニティが指定されます。accounting サービスが finance_group 以外のリソース・コンシューマ・サービスにマップされた場合、このクラスのジョブは finance_group コンシューマ・グループで実行されることに注意してください。これは、resource_consumer_group 属性が優先されるためです。

> **関連項目：** CREATE_JOB_CLASS プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。また、27-29 ページの「ジョブ・クラスの作成」を参照してください。

プログラムの作成例

この項では、プログラムの作成例をいくつか示します。プログラムを作成するには、CREATE_PROGRAM プロシージャを使用します。

例 28-5 プログラムの作成

次の文では、oe スキーマ内にプログラムが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name          => 'oe.my_program1',
   program_type          => 'PLSQL_BLOCK',
   program_action        => 'BEGIN DBMS_STATS.GATHER_TABLE_STATS(''oe'',
                             ''sales''); END;',
   number_of_arguments   => 0,
   enabled               => TRUE,
   comments              => 'My comments here');
END;
/
```

この文では、sales 表に関する表統計を収集する PL/SQL を使用する my_program1 が作成されます。プログラムが作成されたことを確認するには、次の文を発行します。

```
SELECT PROGRAM_NAME FROM DBA_SCHEDULER_PROGRAMS
WHERE PROGRAM_NAME = 'MY_PROGRAM1';

PROGRAM_NAME
-------------------------
MY_PROGRAM1
```

例 28-6 プログラムの作成

次の文では、oe スキーマ内にプログラムが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name          => 'oe.my_saved_program1',
   program_action        => '/usr/local/bin/date',
   program_type          => 'EXECUTABLE',
   comments              => 'My comments here');
END;
/
```

この文では、実行可能ファイルを使用する my_saved_program1 が作成されます。

> **関連項目：** CREATE_PROGRAM プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。また、27-20 ページの「プログラムの作成」を参照してください。

ウィンドウの作成例

この項では、ウィンドウの作成例をいくつか示します。ウィンドウを作成するには、CREATE_WINDOW プロシージャを使用します。

例 28-7 ウィンドウの作成

次の文では、SYS 内に my_window1 というウィンドウが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW (
   window_name          =>  'my_window1',
   resource_plan        =>  'my_res_plan1',
   start_date           =>  '15-JUL-03 1.00.00AM US/Pacific',
   repeat_interval      =>  'FREQ=DAILY',
   end_date             =>  '15-SEP-03 1.00.00AM US/Pacific',
   duration             =>  interval '80' MINUTE,
   comments             =>  'This is my first window');
END;
/
```

このウィンドウは 5 月 15 日～ 10 月 15 日の間、毎日 1 回、午前 1 時から 80 分間オープンします。ウィンドウが作成されたことを確認するには、次の文を発行します。

```
SELECT WINDOW_NAME FROM DBA_SCHEDULER_WINDOWS WHERE WINDOW_NAME = 'MY_WINDOW1';

WINDOW_NAME
------------------------------
MY_WINDOW1
```

例 28-8 ウィンドウの作成

次の文では、SYS 内に my_window2 というウィンドウが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW (
   window_name       => 'my_window2',
   schedule_name     => 'my_stats_schedule',
   resource_plan     => 'my_resourceplan1',
   duration          => interval '60' minute,
   comments          => 'My window');
END;
/
```

関連項目： CREATE_WINDOW プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。また、27-31 ページの「ウィンドウの作成」を参照してください。

ウィンドウ・グループの作成例

この項では、ウィンドウ・グループの作成例を示します。ウィンドウ・グループを作成するには、CREATE_WINDOW_GROUP プロシージャを使用します。

例 28-9 ウィンドウ・グループの作成

次の文では、my_window_group1 というウィンドウ・グループが作成されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW_GROUP ('my_windowgroup1');
END;
/
```

作成した後は、次の文を発行して、my_window_group1 に 3 つのウィンドウ（my_window1、my_window2 および my_window3）を追加できます。

```
BEGIN
DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER (
   group_name   =>  'my_window_group1',
   window_list  =>  'my_window1, my_window2');

DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER (
   group_name   =>  'my_window_group1',
   window_list  =>  'my_window3');
END;
/
```

ウィンドウ・グループが作成され、ウィンドウが追加されたことを確認するには、次の文を発行します。

```
SELECT * FROM DBA_SCHEDULER_WINDOW_GROUPS;

WINDOW_GROUP_NAME    ENABLED   NUMBER_OF_WINDOWS   COMMENTS
-----------------    -------   -----------------   ---------------
MY_WINDOW_GROUP1     TRUE                      3   This is my first window group
```

関連項目： CREATE_WINDOW_GROUP および ADD_WINDOW_GROUP_MEMBER の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。また、27-39 ページの「ウィンドウ・グループの作成」を参照してください。

属性の設定例

この項では、属性の設定例をいくつか示します。属性を設定するには、SET_ATTRIBUTE および SET_SCHEDULER_ATTRIBUTE プロシージャを使用します。

例 28-10 繰返し間隔属性の設定

次の例では、my_emp_job1 が日次で実行されるように頻度が再設定されます。

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           =>   'my_emp_job1',
   attribute      =>   'repeat_interval',
   value          =>   'FREQ=DAILY');
END;
/
```

変更されたことを確認するには、次の文を発行します。

```
SELECT JOB_NAME, REPEAT_INTERVAL FROM DBA_SCHEDULER_JOBS
WHERE JOB_NAME =  'MY_EMP_JOB1';

JOB_NAME             REPEAT_INTERVAL
----------------     ---------------
MY_EMP_JOB1          FREQ=DAILY
```

例 28-11 コメント属性の設定

次の例では、my_saved_program1 に対するコメントが再設定されます。

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           =>   'my_saved_program1',
   attribute      =>   'comments',
   value          =>   'For nightly table stats');
END;
/
```

変更されたことを確認するには、次の文を発行します。

```
SELECT PROGRAM_NAME, COMMENTS FROM DBA_SCHEDULER_PROGRAMS;

PROGRAM_NAME         COMMENTS
------------         ------------------------
MY_PROGRAM1          My comments here
MY_SAVED_PROGRAM1    For nightly table stats
```

例 28-12 継続時間属性の設定

次の例では、my_window3 の継続時間が 90 分に再設定されます。

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           =>   'my_window3',
   attribute      =>   'duration',
   value          =>   interval '90' minute);
END;
/
```

変更されたことを確認するには、次の文を発行します。

```
SELECT WINDOW_NAME, DURATION FROM DBA_SCHEDULER_WINDOWS
WHERE WINDOW_NAME = 'MY_WINDOW3';

WINDOW_NAME         DURATION
-----------         ---------------
MY_WINDOW3          +000 00:90:00
```

例 28-13 データベース・ロール属性の設定

次の例では、ジョブ my_job のデータベース・ロールが LOGICAL STANDBY に設定されます。

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE (
   name        => 'my_job',
   attribute   => 'database_role',
   value       =>'LOGICAL STANDBY');
END;
/
```

データベース・ロールが変更されたことを確認するには、次のコマンドを発行します。

```
SELECT JOB_NAME, DATABASE_ROLE FROM DBA_SCHEDULER_JOB_ROLES
    WHERE JOB_NAME = 'MY_JOB';

JOB_NAME             DATABASE_ROLE
--------             -----------------
MY_JOB               LOGICAL STANDBY
```

例 28-14 イベント失効属性の設定

次の例では、イベントの失効時間を秒（最大 3600 秒）で設定しています。

```
BEGIN
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE (
   attribute     =>   event_expiry_time,
   value         =>   3600);
END;
/
```

例 28-15 一連のジョブに対する複数のジョブ属性の設定

次の例では、5 つの各ジョブに 4 つの異なる属性が設定されます。

```
DECLARE
 newattr sys.jobattr;
 newattrarr sys.jobattr_array;
 j number;
BEGIN
 -- Create new JOBATTR array
 newattrarr := sys.jobattr_array();

 -- Allocate enough space in the array
 newattrarr.extend(20);
 j := 1;
 FOR i IN 1..5 LOOP
   -- Create and initialize a JOBATTR object type
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'MAX_FAILURES',
                          attr_value => 5);
   -- Add it to the array.
   newattrarr(j) := newattr;
   j := j + 1;
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'COMMENTS',
                          attr_value => 'Test job');
   newattrarr(j) := newattr;
   j := j + 1;
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'END_DATE',
                          attr_value => systimestamp + interval '24' hour);
   newattrarr(j) := newattr;
   j := j + 1;
```

```
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'SCHEDULE_LIMIT',
                          attr_value => interval '1' hour);
   newattrarr(j) := newattr;
   j := j + 1;
 END LOOP;

 -- Call SET_JOB_ATTRIBUTES to set all 20 set attributes in one transaction
 DBMS_SCHEDULER.SET_JOB_ATTRIBUTES(newattrarr, 'TRANSACTIONAL');
END;
/
```

関連項目： SET_SCHEDULER_ATTRIBUTE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。また、28-6 ページの「タスク 2E: スケジューラ属性の設定」を参照してください。

チェーンの作成例

この項では、チェーンの作成例を示します。チェーンを作成するには、CREATE_CHAIN プロシージャを使用します。チェーンを作成した後、DEFINE_CHAIN_STEP または DEFINE_CHAIN_EVENT_STEP プロシージャを使用してチェーンにステップを追加し、DEFINE_CHAIN_RULE プロシージャを使用してルールを定義します。

例 28-16 チェーンの作成

次の例では、my_program2 および my_program3 の前に my_program1 を実行するチェーンが作成されます。my_program1 の完了後、my_program2 および my_program3 は、並行して実行されます。

この例のユーザーには、CREATE EVALUATION CONTEXT、CREATE RULE および CREATE RULE SET 権限が必要です。詳細は、28-2 ページの「チェーンの各権限の設定」を参照してください。

```
BEGIN
DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name           =>  'my_chain1',
   rule_set_name        =>  NULL,
   evaluation_interval  =>  NULL,
   comments             =>  NULL);
END;
/

--- define three steps for this chain. Referenced programs must be enabled.
BEGIN
DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepA', 'my_program1');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepB', 'my_program2');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepC', 'my_program3');
END;
/

--- define corresponding rules for the chain.
BEGIN
DBMS_SCHEDULER.DEFINE_CHAIN_RULE('my_chain1', 'TRUE', 'START stepA');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain1', 'stepA COMPLETED', 'Start stepB, stepC');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain1', 'stepB COMPLETED AND stepC COMPLETED', 'END');
END;
/

--- enable the chain
BEGIN
  DBMS_SCHEDULER.ENABLE('my_chain1');
```

```
END;
/

--- create a chain job to start the chain daily at 1:00 p.m.
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name        => 'chain_job_1',
   job_type        => 'CHAIN',
   job_action      => 'my_chain1',
   repeat_interval => 'freq=daily;byhour=13;byminute=0;bysecond=0',
   enabled         => TRUE);
END;
/
```

例 28-17 チェーンの作成

次の例では、最初に my_program1 を実行してチェーンを作成します。正常に実行された場合は my_program2、失敗した場合は my_program3 が実行されます。

```
BEGIN
DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name           => 'my_chain2',
   rule_set_name        => NULL,
   evaluation_interval  => NULL,
   comments             => NULL);
END;
/

--- define three steps for this chain.
BEGIN
DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain2', 'step1', 'my_program1');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain2', 'step2', 'my_program2');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain2', 'step3', 'my_program3');
END;
/

--- define corresponding rules for the chain.
BEGIN
DBMS_SCHEDULER.DEFINE_CHAIN_RULE ('my_chain2', 'TRUE', 'START step1');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain2', 'step1 SUCCEEDED', 'Start step2');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain2', 'step1 COMPLETED AND step1 NOT SUCCEEDED', 'Start step3');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain2', 'step2 COMPLETED OR step3 COMPLETED', 'END');
END;
/
```

関連項目： CREATE_CHAIN、DEFINE_CHAIN_STEP および DEFINE_CHAIN_RULE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。また、28-6 ページの「タスク 2E: スケジューラ属性の設定」を参照してください。

イベントに基づくジョブとスケジュールの作成例

この項では、イベント・ベースのジョブおよびイベント・スケジュールの作成例を示します。イベント・ベースのジョブを作成するには、CREATE_JOB プロシージャを使用します。イベント・ベースのスケジュールを作成するには、CREATE_EVENT_SCHEDULE プロシージャを使用します。

これらの例では、アプリケーションでシステムにファイルが到着したことが検出されたときに、イベントを my_events_q キューにエンキューするアプリケーションがあることを想定しています。

例 28-18 イベント・ベースのスケジュールの作成

次の例では、午前 9 時前にファイルがシステムに到着したことを示すイベントをスケジューラが受信するたびに、ジョブの開始に使用可能なスケジュールを作成する方法を示しています。

```
BEGIN
DBMS_SCHEDULER.CREATE_EVENT_SCHEDULE (
   schedule_name     =>  'scott.file_arrival',
   start_date        =>  systimestamp,
   event_condition   =>  'tab.user_data.object_owner = ''SCOTT''
      and tab.user_data.event_name = ''FILE_ARRIVAL''
      and extract hour from tab.user_data.event_timestamp < 9',
   queue_spec        =>  'my_events_q');
END;
/
```

例 28-19 イベントベースのジョブの作成

次の例では、ファイルがシステムに到着したことを示すイベントをスケジューラが受信したときに開始されるジョブを作成しています。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name            =>  my_job,
   program_name        =>  my_program,
   start_date          =>  '15-JUL-04 1.00.00AM US/Pacific',
   event_condition     =>  'tab.user_data.event_name = ''FILE_ARRIVAL''',
   queue_spec          =>  'my_events_q'
   enabled             =>  TRUE,
   comments            =>  'my event-based job');
END;
/
```

関連項目： CREATE_JOB および CREATE_EVENT_SCHEDULE プロシージャの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

Oracle Data Guard 環境でのジョブの作成例

Oracle Data Guard 環境では、スケジューラには、2 つのデータベース・ロール（プライマリとロジカル・スタンバイ）に対する追加のサポートが含まれています。データベースのロールがプライマリの場合のみ、またはデータベースのロールがロジカル・スタンバイの場合のみ、ジョブが実行されるように構成できます。そのためには、database_role 属性を設定します。この例では、両方のデータベース・ロールでジョブを実行できるようにする方法を説明します。ジョブのコピーを 2 つ作成し、それぞれに異なる database_role を割り当てる方法を使用します。

デフォルトでは、ジョブが実行されるのは、データベースのロールがそのジョブの作成時と同じロールの場合です。両方のロールで同じジョブを実行する手順は、次のとおりです。

1. ジョブをコピーします。
2. 新しいジョブを使用可能にします。
3. 新しいジョブの database_role 属性を必要なロールに変更します。

この例は、プライマリ・データベースで primary_job というジョブを作成することで起動します。次に、このジョブのコピーを作成し、database_role 属性を LOGICAL STANDBY に設定します。プライマリ・データベースがロジカル・スタンバイになると、ジョブはそのスケジュールに従って引き続き実行されます。

ジョブをコピーした場合、新しいジョブは使用禁止になっているため、使用可能にする必要があります。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
     job_name       => 'primary_job',
     program_name   => 'my_prog',
     schedule_name  => 'my_sched');

DBMS_SCHEDULER.COPY_JOB('primary_job','standby_job');
DBMS_SCHEDULER.ENABLE(name=>'standby_job', commit_semantics=>'ABSORB_ERRORS');
DBMS_SCHEDULER.SET_ATTRIBUTE('standby_job','database_role','LOGICAL STANDBY');
END;
/
```

この例を実行すると、DBA_SCHEDULER_JOB_ROLES ビューのデータは、次のようになります。

```
SELECT JOB_NAME, DATABASE_ROLE FROM DBA_SCHEDULER_JOB_ROLES
   WHERE JOB_NAME IN ('PRIMARY_JOB','STANDBY_JOB');

JOB_NAME               DATABASE_ROLE
--------               ----------------
PRIMARY_JOB            PRIMARY
STABDBY_JOB            LOGICAL STANDBY
```

注意： フィジカル・スタンバイ・データベースの場合、スケジューラのオブジェクトに対する変更またはプライマリ・データベース上のスケジューラのジョブによるデータベースの変更は、他のデータベースの変更と同様にフィジカル・スタンバイに適用されます。

スケジューラの参照情報

ここでは、Oracle Scheduler の参照情報を提供します。この章の内容は、次のとおりです。

- スケジューラ権限
- スケジューラのデータ・ディクショナリ・ビュー

スケジューラ権限

表 28-2 に、スケジューラの様々な権限をリストします。

表 28-2 スケジューラ権限

権限名	許可されている操作
システム権限 :	
CREATE JOB	自分のスキーマ内にジョブ、チェーン、スケジュール、プログラムおよび資格証明を作成できます。CREATE JOB 権限が付与されていない場合でも、自分のスキーマ内で常にジョブ、スケジュールおよびプログラムを変更および削除できます。この場合、使用しているスキーマ内に CREATE ANY JOB 権限を持つ別のユーザーによって、そのジョブがすでに作成されていることが必要です。
CREATE ANY JOB	SYS 以外の任意のスキーマ内でジョブ、チェーン、スケジュール、プログラムおよび資格証明を作成、変更および削除できます。これは非常に強力な権限で、権限受領者は任意の他のデータベース・ユーザーで PL/SQL コードを実行できるため、慎重に使用してください。
CREATE EXTERNAL JOB	この権限はデータベース外で実行するジョブを作成する場合に必要です。EXECUTABLE タイプのジョブの所有者、または EXECUTABLE タイプのプログラムを指し示すジョブの所有者にはこの権限が必要です。EXECUTABLE タイプのジョブを実行する場合は、この権限および CREATE JOB 権限が必要です。リモート・ホストからファイルを取得したり、1 つ以上のリモート・ホストにファイルを保存する場合も、この権限が必要です。
EXECUTE ANY PROGRAM	自分のジョブに任意のスキーマのプログラムまたはチェーンを使用できます。
EXECUTE ANY CLASS	自分のジョブを任意のジョブ・クラスの下で実行できます。
MANAGE SCHEDULER	これはスケジューラを管理する上で最も重要な権限です。ジョブ・クラス、ウィンドウおよびウィンドウ・グループを、作成、変更および削除できます。また、スケジューラ属性の設定と検索、スケジューラ・ログのパージ、パブリック資格証明の削除、データベースのエージェント・パスワードの設定も可能です。

表 28-2 スケジューラ権限（続き）

権限名	許可されている操作
オブジェクト権限 :	
EXECUTE	この権限はプログラム、チェーンおよびジョブ・クラスに対してのみ付与されます。プログラム、チェーン、またはジョブ・クラスを使用して実行するジョブを作成できます。また、オブジェクト属性を表示できます。
ALTER	権限付与されているオブジェクトを変更または削除できます。変更操作には、プログラム引数の使用可能化、使用不能化、定義または削除、ジョブ引数値の設定または再設定、およびジョブの実行などの操作が含まれます。プログラム、ジョブおよびチェーンについては、この権限によってオブジェクト属性を表示できます。この権限はジョブ、チェーン、プログラムおよびスケジュールに対してのみ付与されます。他のタイプのスケジューラ・オブジェクトに対して、MANAGE SCHEDULER システム権限を付与できます。この権限を付与できる対象は、次のとおりです。 ジョブ（DROP_JOB、RUN_JOB、ALTER_RUNNING_CHAIN、SET_JOB_ARGUMENT_VALUE、RESET_JOB_ARGUMENT_VALUE、SET_JOB_ANYDATA_VALUE）および（force オプションなしの STOP_JOB） チェーン（DROP_CHAIN、ALTER_CHAIN、DEFINE_CHAIN_RULE、DEFINE_CHAIN_STEP、DEFINE_CHAIN_EVENT_STEP、DROP_CHAIN_RULE、および DROP_CHAIN_STEP） プログラム（DROP_PROGRAM、DEFINE_PROGRAM_ARGUMENT、DEFINE_ANYDATA_ARGUMENT、DEFINE_METADATA_ARGUMENT、DROP_PROGRAM_ARGUMENT、SET_ATTRIBUTE_NULL） スケジュール（DROP_SCHEDULE）
ALL	特定のオブジェクトに対して、他の使用可能なすべての権限によって許可される操作を許可します。この権限は、ジョブ、プログラム、チェーン、スケジュールおよびジョブ・クラスに対して付与されます。

SCHEDULER_ADMIN ロールは、表 28-2 に記載されているすべてのシステム権限付き（ADMIN オプション付き）で作成されます。SCHEDULER_ADMIN ロールは DBA（ADMIN オプション付き）に付与されます。

SELECT ALL_SCHEDULER_* ビュー、SELECT USER_SCHEDULER_* ビュー、SELECT SYS.SCHEDULER$_JOBSUFFIX_S（ジョブ名生成用）、および EXECUTE SYS.DEFAULT_JOB_CLASS の各オブジェクト権限が、PUBLIC に付与されます。

スケジューラのデータ・ディクショナリ・ビュー

スケジューラの情報は、多数のビューで確認できます。次に、my_job1 の完了したインスタンスに関する情報の表示例を示します。

```
SELECT JOB_NAME, STATUS, ERROR#
FROM DBA_SCHEDULER_JOB_RUN_DETAILS WHERE JOB_NAME = 'MY_JOB1';

JOB_NAME     STATUS           ERROR#
--------     --------------   ------
MY_JOB1      FAILURE           20000
```

表 28-3 に、スケジューラに関連付けられたビューを示します。*_SCHEDULER_JOBS、*_SCHEDULER_SCHEDULES、*_SCHEDULER_PROGRAMS、*_SCHEDULER_RUNNING_JOBS、*_SCHEDULER_JOB_LOG および *_SCHEDULER_JOB_RUN_DETAILS の各ビューは、ジョブの管理に特に役立ちます。スケジューラのビューの詳細は、『Oracle Database リファレンス』を参照してください。

注意 : 次の表のビュー名の先頭にあるアスタリスクは、DBA、ALL または USER で置換されます。

表 28-3 スケジューラのビュー

ビュー	説明
*_SCHEDULER_CHAIN_RULES	すべてのチェーンについて、すべてのルールが表示されます。
*_SCHEDULER_CHAIN_STEPS	すべてのチェーンについて、すべてのステップが表示されます。
*_SCHEDULER_CHAINS	すべてのチェーンが表示されます。
*_SCHEDULER_CREDENTIALS	すべての資格証明が表示されます。
*_SCHEDULER_GLOBAL_ATTRIBUTE	スケジューラ属性の現行値が表示されます。
*_SCHEDULER_JOB_ARGS	すべてのジョブについて、すべての引数の設定値が表示されます。
*_SCHEDULER_JOB_CLASSES	すべてのジョブ・クラスが表示されます。
*_SCHEDULER_JOB_LOG	設定されているロギング・レベルに応じて、ジョブの実行と状態変化が表示されます。
*_SCHEDULER_JOB_ROLES	Oracle Data Guard データベース・ロールによるすべてのジョブが表示されます。
*_SCHEDULER_JOB_RUN_DETAILS	完了（失敗または成功）したすべてのジョブ実行が表示されます。
*_SCHEDULER_JOBS	すべてのジョブ（使用可能なジョブと使用禁止のジョブ）が表示されます。
*_SCHEDULER_PROGRAM_ARGS	すべてのプログラムに定義されているすべての引数が、デフォルト値（存在する場合）とともに表示されます。
*_SCHEDULER_PROGRAMS	すべてのプログラムが表示されます。
*_SCHEDULER_RUNNING_CHAINS	実行中のすべてのチェーンが表示されます。
*_SCHEDULER_RUNNING_JOBS	現在実行中のすべてのジョブに関する状態情報が表示されます。
*_SCHEDULER_SCHEDULES	すべてのスケジュールが表示されます。
*_SCHEDULER_WINDOW_DETAILS	完了したすべてのウィンドウの実行が表示されます。
*_SCHEDULER_WINDOW_GROUPS	すべてのウィンドウ・グループが表示されます。
*_SCHEDULER_WINDOW_LOG	ウィンドウに行われた変更の状態がすべて表示されます。
*_SCHEDULER_WINDOWS	すべてのウィンドウが表示されます。
*_SCHEDULER_WINGROUP_MEMBERS	すべてのウィンドウ・グループのメンバーが、グループ・メンバーごとに 1 行ずつ表示されます。

第 V 部

分散データベースの管理

第 V 部では、分散データベース環境の管理について説明します。この部の構成は、次のとおりです。

29

分散データベースの概念

この章の内容は次のとおりです。

分散データベース・アーキテクチャ

分散データベース・システムを使用すると、アプリケーションはローカル・データベースおよびリモート・データベースのデータにアクセスできます。**同機種間分散データベース・システム**では、個々のデータベースは Oracle Database です。**異機種間分散データベース・システム**では、少なくともデータベースの 1 つが Oracle Database 以外のデータベースです。分散データベースは、**クライアント / サーバー**・アーキテクチャを使用して情報の要求を処理します。

この項の内容は、次のとおりです。

同機種間分散データベース・システム

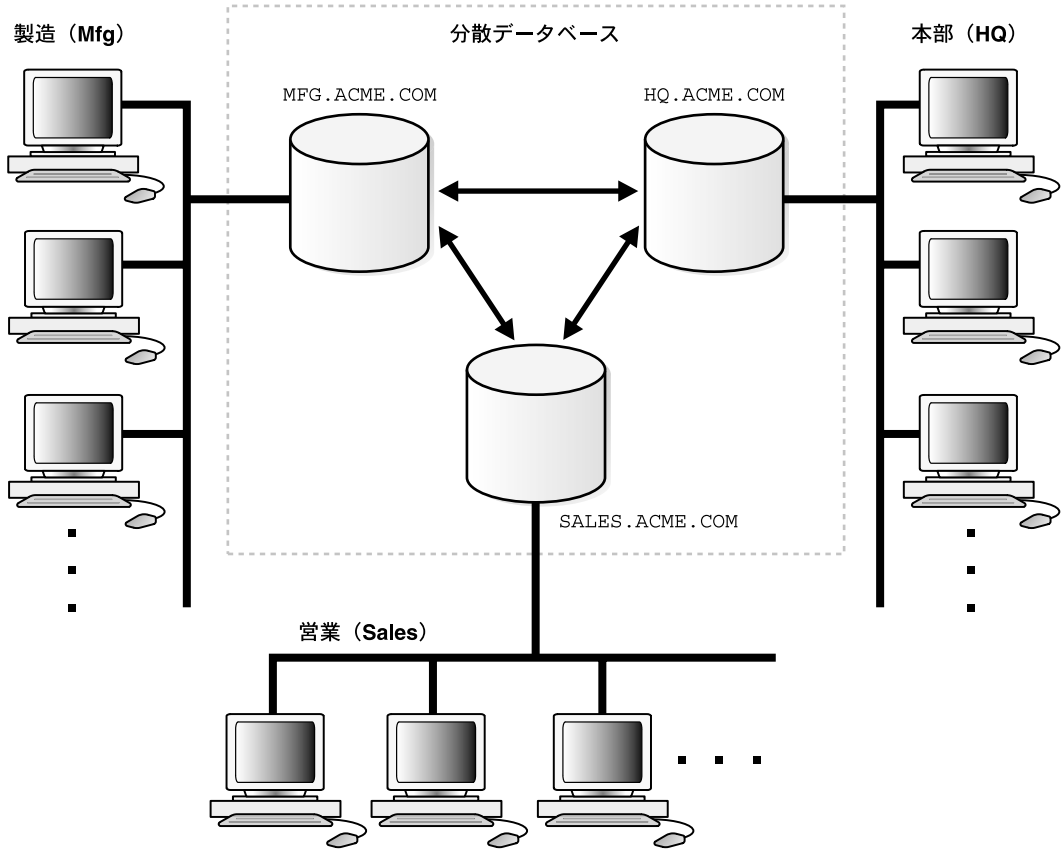
同機種間分散データベース・システムとは、1 台以上のマシンに存在する 2 つ以上の Oracle Database のネットワークを表します。図 29-1 は、hq、mfg および sales の 3 つのデータベースを接続している分散システムを示しています。アプリケーションは、1 つの分散環境内にある複数のデータベースのデータに同時にアクセスしたり、データを同時に変更できます。たとえば、ローカル・データベース mfg の製造（Mfg）クライアントから発行する 1 回の問合せによって、ローカル・データベースの products 表のデータとリモート・データベース hq の dept 表のデータを結合したデータを取得できます。

クライアント・アプリケーションに対して、データベースの位置とプラットフォームは透過的です。また、分散システム内のリモート・オブジェクトに対してユーザーがローカル・オブジェクトと同じ構文を使用してアクセスできるように、リモート・オブジェクトの**シノニム**を作成することもできます。たとえば、mfg データベースに接続しているときにデータベース hq のデータにアクセスする場合は、リモートの dept 表に対するシノニムを mfg 上に作成することで、次の問合せを発行できます。

```
SELECT * FROM dept;
```

このように、分散システムではローカル・データベースと同様のデータ・アクセスが可能です。mfg のユーザーは、アクセスするデータがリモート・データベースに存在していることを知る必要はありません。

図 29-1　同機種間分散データベース



Oracle Database の分散データベース・システムは、異なるバージョンの Oracle Database で構成することが可能です。サポートされている Oracle Database のリリースは、すべて分散データベース・システムに加わることができます。しかし、分散データベースを使用するアプリケーションでは、システムの各ノードで使用可能な機能を理解しておく必要があります。分散データベース・アプリケーションでは、Oracle Database でのみ使用可能な SQL の拡張を Oracle7 で実行できるかのような想定を行ってはなりません。

分散データベースと分散処理

分散データベースおよび**分散処理**という用語は密接に関連していますが、その意味は異なります。この 2 つの用語の定義は、次のとおりです。

- 分散データベース

  分散システムを構成する一連のデータベース。アプリケーションからは単一のデータソースのように見えます。

- 分散処理

  アプリケーションが自身のタスクをネットワーク内の異なるコンピュータ間に分散するときに発生する操作。たとえば、データベース・アプリケーションは通常、フロントエンド側のプレゼンテーション・タスクをクライアント・コンピュータに配置し、バックエンド側のデータベース・サーバーでデータベースへの共有アクセスを管理できるようにします。このことから、分散データベース・アプリケーション処理システムは、一般にクライアント / サーバー・データベース・アプリケーション・システムとも呼ばれます。

分散データベース・システムは、分散処理アーキテクチャを利用しています。たとえば、Oracle Database サーバーは、別の Oracle Database サーバーが管理しているデータを要求するときにクライアントとして動作します。

分散データベースとレプリケート・データベース

分散データベース・システムおよび**データベース・レプリケーション**という用語は関連していますが、その意味は異なります。**純粋な**（つまり、レプリケートされていない）分散データベースでは、すべてのデータとそれをサポートしているデータベース・オブジェクトの単一コピーが管理されています。通常、分散データベース・アプリケーションは、分散トランザクションを使用してローカルおよびリモートのデータにアクセスし、グローバル・データベースをリアルタイムで変更します。

注意： このマニュアルでは、純粋な分散データベースについてのみ説明しています。

レプリケーションという用語は、分散システムに属している複数のデータベースの間でデータベース・オブジェクトをコピーし、管理する操作を指します。レプリケーションは分散データベース・テクノロジに依存していますが、データベース・レプリケーションを使用すると、純粋な分散データベース環境では得られないような利点をアプリケーションが利用できます。

一般に、レプリケーションを使用すると代替のデータ・アクセス手段が提供されるので、ローカル・データベースのパフォーマンスが向上し、アプリケーションの可用性が保護されます。たとえば、アプリケーションは、ネットワークの通信量を最小限に抑えてパフォーマンスの最大化を図るために、リモート・サーバーではなくローカル・データベースにアクセスできます。また、ローカル・サーバーに障害が発生しても、レプリケートされたデータを持つ他のサーバーにアクセスできれば、アプリケーションは引き続き実行できます。

関連項目：

- Oracle Database のレプリケーション機能の詳細は、『Oracle Database アドバンスト・レプリケーション』を参照してください。
- Oracle Streams、つまりデータベース間で情報を共有する別の方法については、『Oracle Streams 概要および管理』を参照してください。

異機種間分散データベース・システム

異機種間分散データベース・システムでは、少なくともデータベースの 1 つが Oracle Database 以外のシステムです。アプリケーションにとって、異機種間分散データベース・システムは 1 つのローカルな Oracle Database のように見えます。ローカルの Oracle Database サーバーでは、データの分散と異機種性は隠されています。

Oracle Database サーバーは、Oracle Database 以外のシステムへのアクセスに、Oracle 異機種間サービスと**エージェント**を組み合せて使用します。Oracle Database 以外のデータ・ストアに Oracle Transparent Gateway を使用してアクセスする場合、エージェントはシステム固有のアプリケーションになります。たとえば、Oracle Database 分散システムに Sybase データベースを含める場合は、そのシステム内の Oracle Database が Sybase データベースとやり取りできるように、Sybase 固有の Transparent Gateway を入手する必要があります。

Oracle Database 以外のシステムが ODBC または OLE DB プロトコルをサポートしている場合は、かわりに **Generic Connectivity** を使用して Oracle Database 以外のデータ・ストアにアクセスできます。

注意： このマニュアルで Oracle 異機種間サービスについて説明しているのは、この章の概要的な内容のみです。異機種間サービスの詳細は、『Oracle Database Heterogeneous Connectivity 管理者ガイド』を参照してください。

異機種間サービス

異機種間サービスは Oracle Database サーバー内部に統合されたコンポーネントであり、現在の Oracle Transparent Gateway 製品群を実現するためのテクノロジです。異機種間サービスは、Oracle Database ゲートウェイ製品とその他の異機種間アクセス機能に対して共通のアーキテクチャと管理のメカニズムを提供します。また、以前にリリースされたほとんどの Oracle Transparent Gateway のユーザーに上位互換の機能を提供します。

Transparent Gateway エージェント

Oracle Database 以外の個々のシステムにアクセスする場合、異機種間サービスは Transparent Gateway エージェントを使用して、Oracle Database 以外の特定のシステムとのインタフェースの役目を果たします。エージェントは Oracle Database 以外のシステムに固有のものであり、システムのタイプごとに異なるエージェントが必要になります。

Transparent Gateway エージェントは、Oracle Database サーバー内の異機種間サービス・コンポーネントを使用して、Oracle Database システムと Oracle Database 以外のシステムとの間のやり取りを容易にします。エージェントは Oracle Database サーバーのかわりに、Oracle Database 以外のシステムで SQL とトランザクション要求を実行します。

> **関連項目：** 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。

Generic Connectivity

Generic Connectivity を使用すると、異機種間サービス ODBC エージェントまたは異機種間サービス OLE DB エージェントのどちらかを使用して Oracle Database 以外のデータ・ストアに接続できます。これらはどちらも Oracle 製品に標準機能として組み込まれています。ODBC または OLE DB の規格と互換性があれば、どのようなデータソースでも Generic Connectivity エージェントを使用してアクセスできます。

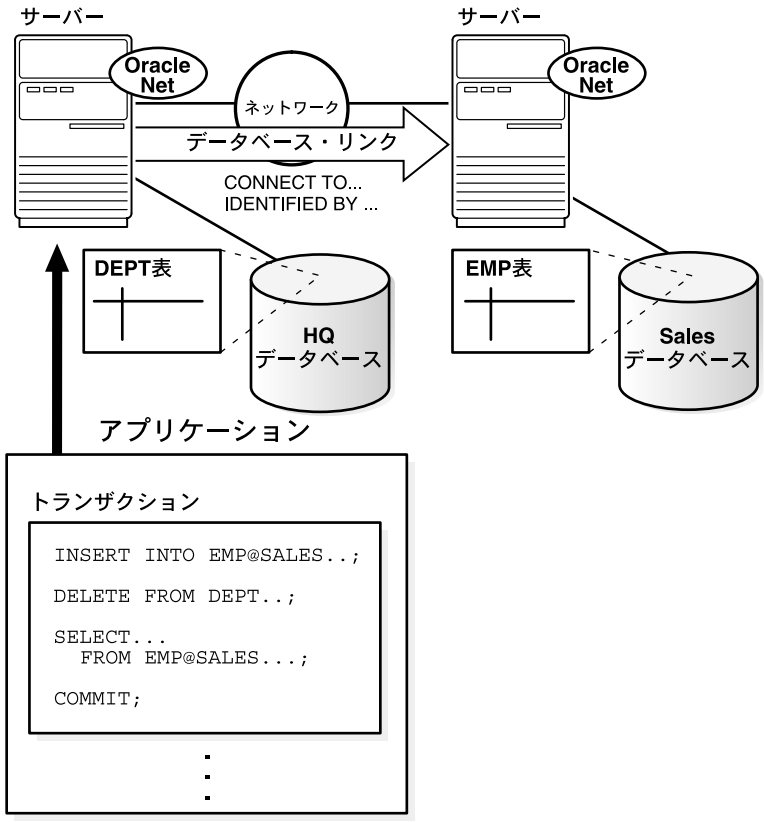
Generic Connectivity の利点は、必ずしもシステム固有のエージェントを別途購入して構成する必要がないことです。ODBC ドライバまたは OLE DB ドライバを使用して、エージェントとインタフェースできます。ただし、一部のデータ・アクセス機能は Transparent Gateway エージェントでしか利用できません。

クライアント / サーバー・データベース・アーキテクチャ

データベース・サーバーとはデータベースを管理する Oracle ソフトウェアのことであり、クライアントとはサーバーの情報を要求するアプリケーションのことです。ネットワーク内の各コンピュータはノードと呼ばれ、1 つ以上のデータベースのホストとなることができます。分散データベース・システム内の各ノードは、状況に応じてクライアント、サーバー、あるいはその両方として機能します。

図 29-2 の `hq` データベースのホストは、そのローカル・データに対して文が発行されたときはデータベース・サーバーとして機能します。たとえば、トランザクション内の 2 番目の文は、ローカルの `dept` 表に対して文を発行しています。しかし、リモート・データに対して文が発行されたときはクライアントとして機能します。たとえば、トランザクション内の最初の文は、`sales` データベース内のリモートの表 `emp` に対して発行されています。

図 29-2 Oracle Database の分散データベース・システム



クライアントは、データベース・サーバーに**直接**または**間接**に接続できます。直接接続となるのは、クライアントがサーバーに接続し、そのサーバー上に存在するデータベースの情報にアクセスするときです。たとえば、図 29-2 のように、hq データベースに接続してこのデータベースの dept 表にアクセスする場合は、次の文を発行できます。

```
SELECT * FROM dept;
```

この場合は、リモート・データベースのオブジェクトにアクセスしていないので、問合せは直接になります。

これに対して、間接接続となるのは、クライアントがサーバーに接続した後、別の異なるサーバー上のデータベースに格納されている情報にアクセスするときです。たとえば、図 29-2 のように、hq データベースに接続した後、リモートの sales データベースの emp 表にアクセスする場合は、次の文を発行できます。

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

この場合は、アクセスしようとしているオブジェクトが直接接続しているデータベース上にないため、問合せは間接になります。

データベース・リンク

分散データベース・システムの中心的な概念となるのが、**データベース・リンク**です。データベース・リンクとは 2 つの物理的なデータベース・サーバー間の接続のことで、これにより、クライアントからそれらのサーバーに 1 つの論理データベースとしてアクセスできるようになります。

この項の内容は、次のとおりです。

データベース・リンクの概要

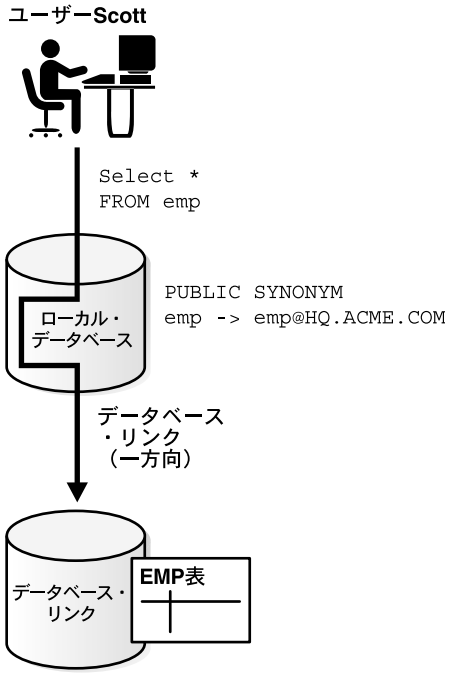
データベース・リンクは、ある Oracle Database サーバーから別のデータベース・サーバーへの一方向の通信経路を定義するポインタです。リンク・ポインタは、実際にはデータ・ディクショナリ表内のエントリとして定義されます。リンクにアクセスするには、データ・ディクショナリ・エントリのあるローカル・データベースに接続する必要があります。

データベース・リンク接続が一方向であるということは、たとえばローカル・データベース A に接続しているクライアントはデータベース A に格納されているリンクを使用してリモート・データベース B の情報にアクセスできるが、データベース B に接続しているユーザーは同じリンクを使用してデータベース A のデータにアクセスできないことを意味します。データベース B のローカル・ユーザーがデータベース A のデータにアクセスする場合には、データベース B のデータ・ディクショナリに格納されるリンクを定義する必要があります。

データベース・リンク接続を使用することで、ローカル・ユーザーはリモート・データベースのデータにアクセスできるようになります。この接続を実現するためには、分散システム内の各データベースがネットワーク・ドメイン内で一意の**グローバル・データベース名**を持っていることが必要です。グローバル・データベース名は、分散システム内のデータベース・サーバーを一意に識別します。

図 29-3 は、ユーザー `scott` がグローバル名 `hq.acme.com` を使用して、リモート・データベースの `emp` 表にアクセスしている例を示します。

図 29-3 データベース・リンク



データベース・リンクには、プライベートとパブリックの 2 種類があります。プライベートの場合は、そのリンクを作成したユーザーのみがアクセスできます。パブリックの場合は、すべてのデータベース・ユーザーがアクセスできます。

データベース・リンクにおける原理的な違いの 1 つに、リモート・データベースにどのように接続するかという点があります。ユーザーは、次のタイプのリンクによってリモート・データベースにアクセスします。

リンクのタイプ	説明
接続ユーザー・リンク	ユーザーはユーザー自身として接続します。つまり、ユーザーにはローカル・データベースのアカウントと同じユーザー名およびパスワードを持つリモート・データベースのアカウントが必要です。
固定ユーザー・リンク	ユーザーは、リンク内で参照されるユーザー名とパスワードを使用して接続します。たとえば、Jane が、ユーザー名とパスワード `scott/tiger` で hq データベースに接続する固定ユーザー・リンクを使用する場合、Jane は `scott` として接続し、`scott` に直接付与されている hq 内でのすべての権限と、hq データベースで `scott` に付与されているすべてのデフォルト・ロールを持ちます。
現行ユーザー・リンク	ユーザーは、グローバル・ユーザーとして接続します。ローカル・ユーザーは、ストアド・プロシージャのコンテキスト内ではグローバル・ユーザーとして接続できます。グローバル・ユーザーのパスワードをリンク定義に格納する必要はありません。たとえば、Jane は Scott が記述したプロシージャにアクセスでき、hq データベース上の Scott のアカウントと Scott のスキーマにアクセスできます。現行ユーザー・リンクは、Oracle Advanced Security の機能の 1 つです。

データベース・リンクを作成するには、`CREATE DATABASE LINK` 文を使用します。リンクを作成した後、SQL 文の中でリンクを使用してスキーマ・オブジェクトを指定できます。

関連項目：

- CREATE DATABASE 文の構文は、『Oracle Database SQL リファレンス』を参照してください。
- Oracle Advanced Security の詳細は、『Oracle Database Advanced Security 管理者ガイド』を参照してください。

共有データベース・リンクの概要

共有データベース・リンクとは、ローカル・サーバー・プロセスとリモート・データベースの間のリンクのことです。複数のクライアント・プロセスが同じリンクを同時に使用できるので、共有データベース・リンクと呼ばれます。

ローカル・データベースがデータベース・リンクを介してリモート・データベースに接続するとき、どちらのデータベースも専用サーバー・モードまたは共有サーバー・モードのいずれかで稼働しています。可能な組合せを次の表に示します。

ローカル・データベースのモード	リモート・データベースのモード
専用	専用
専用	共有サーバー
共有サーバー	専用
共有サーバー	共有サーバー

共有データベース・リンクは、これら 4 つのどの構成でも確立できます。共有リンクは、通常のデータベース・リンクと次の点で異なります。

- データベース・リンクを介して同じスキーマ・オブジェクトにアクセスする異なるユーザーの間で、ネットワーク接続を共有できます。
- ユーザーが特定のサーバー・プロセスからリモート・サーバーに接続を確立するときに、そのプロセスからリモート・サーバーに対してすでに確立されている接続を再利用できます。接続を再利用するには、その接続が同じサーバー・プロセスから同じデータベース・リンクを使用して確立されている必要があります。セッションは異なっていてもかまいません。非共有のデータベース・リンクでは、複数のセッション間で接続が共有されることはありません。
- 共有サーバー構成で共有データベース・リンクを使用すると、ネットワーク接続はローカル・サーバーの共有サーバー・プロセスの外部で直接に確立されます。ローカル共有サーバーでの非共有のデータベース・リンクでは、この接続はローカル・ディスパッチャを介して確立されるため、ローカル・ディスパッチャのためのコンテキスト・スイッチングが発生し、データがディスパッチャを経由することになります。

関連項目： 共有サーバーの詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

データベース・リンクを使用する理由

データベース・リンクの大きな利点として、ユーザーがリモート・データベースにある他のユーザーのオブジェクトに対して、オブジェクトの所有者が持つ権限の制限内でアクセスできるという点があります。つまり、ローカル・ユーザーはリモート・データベースのユーザーにならなくても、リモート・データベースへのリンクにアクセスできます。

たとえば、従業員が経費精算書を買掛管理（A/P）アプリケーションに送信し、さらに A/P アプリケーションを使用するユーザーが hq データベースから従業員に関する情報を取得する必要があるとします。A/P のユーザーは、必要な情報を取得するために、hq データベースに接続してリモートの hq データベース内のストアド・プロシージャを実行できます。A/P のユーザーは、自分のジョブを実行するために hq データベースのユーザーになる必要はありません。プロシージャに記述されている制御された方法で hq の情報にアクセスできればそれで済みます。

関連項目：

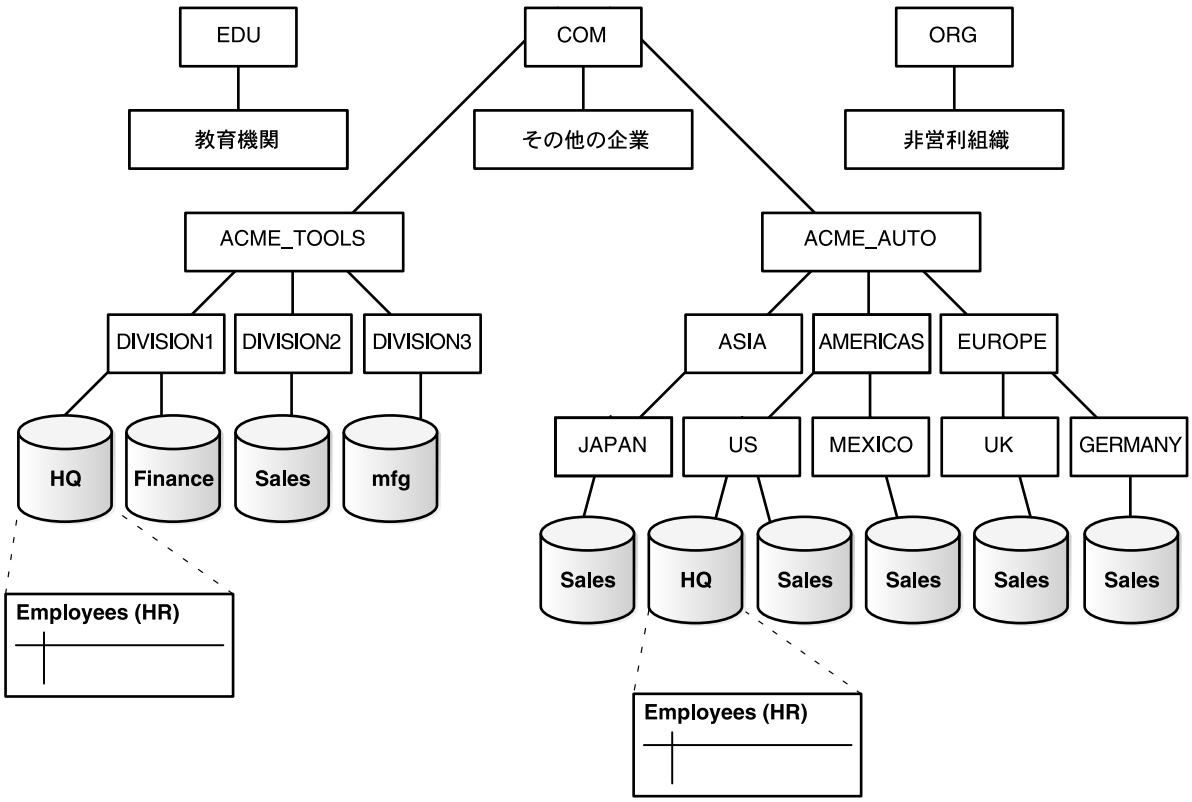
- データベース・リンク・ユーザーの詳細は、29-13 ページの「データベース・リンクのユーザー」を参照してください。
- パスワードを管理者以外のユーザーから隠す方法の詳細は、「データベース・リンク情報の表示」を参照してください。

データベース・リンク内のグローバル・データベース名

データベース・リンクの動作の仕組みを理解するには、まずグローバル・データベース名について理解する必要があります。分散データベース内の各データベースは、それぞれのグローバル・データベース名によって一意に識別されます。データベースのグローバル・データベース名は、データベースのネットワーク・ドメインの前に個々のデータベース名を連結して構成されます。データベースのネットワーク・ドメインは、データベースの作成時に DB_DOMAIN 初期化パラメータによって指定し、個々のデータベース名は DB_NAME 初期化パラメータによって指定します。

たとえば、図 29-4 では、ネットワーク全体のデータベースを階層的な配置で表しています。

図 29-4　ネットワーク化されたデータベースの階層的な配置



データベースの名前は、ツリーのリーフ（葉）から始まってルート（根）に至るまでの経路によって形成されます。たとえば、mfg データベースは、com ドメインの acme_tools ブランチの division3 にあります。mfg のグローバル・データベース名は、ツリー内のノードを次のように連結して作成されます。

- mfg.division3.acme_tools.com

複数のデータベースが 1 つの個別名を共有できますが、各データベースのグローバル・データベース名は一意にする必要があります。たとえば、ネットワーク・ドメイン us.americas.acme_auto.com と uk.europe.acme_auto.com には、どちらにも sales データベースがあります。グローバル・データベース名の命名体系では、americas 部門の sales データベースと europe 部門の sales データベースが次のように区別されます。

- sales.us.americas.acme_auto.com
- sales.uk.europe.acme_auto.com

> **関連項目：** グローバル・データベース名を指定および変更する方法の詳細は、30-2 ページの「分散システムでのグローバル名の管理」を参照してください。

データベース・リンクの名前

通常、データベース・リンクの名前は、参照先のリモート・データベースのグローバル・データベース名と同じです。たとえば、データベースのグローバル・データベース名が sales.us.oracle.com であれば、データベース・リンクの名前も sales.us.oracle.com になります。

初期化パラメータ GLOBAL_NAMES を TRUE に設定すると、データベース・リンクの名前がリモート・データベースのグローバル・データベース名と同じになることが保証されます。たとえば、hq のグローバル・データベース名が hq.acme.com で、GLOBAL_NAMES が TRUE の場合は、リンク名も必ず hq.acme.com になります。データベースは、初期化パラメータ・ファイル内の DB_DOMAIN の設定値ではなく、データ・ディクショナリに格納されているグローバル・データベース名のドメイン部分を調べます（30-4 ページの「グローバル・データベース名のドメインの変更」を参照）。

初期化パラメータ GLOBAL_NAMES を FALSE に設定した場合は、グローバル・ネーミングを使用する必要はありません。データベース・リンクに自由に名前を付けられます。たとえば、hq.acme.com へのデータベース・リンクに foo という名前を付けることができます。

注意： グローバル・ネーミングはレプリケーションなどの多数の機能で必要になるため、グローバル・ネーミングの使用をお薦めします。

グローバル・ネーミングを有効にすると、データベース・リンクの名前がリンクの参照先であるデータベースのグローバル名と同じになるため、データベース・リンクは本質的に分散データベースのユーザーに対して透過的になります。たとえば、次の文は、リモート・データベース sales へのデータベース・リンクをローカル・データベース内に作成します。

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com USING 'sales1';
```

> **関連項目：** 初期化パラメータ GLOBAL_NAMES の指定の詳細は、『Oracle Database リファレンス』を参照してください。

データベース・リンクのタイプ

Oracle Database では、**プライベート**、**パブリック**および**グローバル**のデータベース・リンクを作成できます。これらの基本的なリンク・タイプは、どのユーザーに対してリモート・データベースへのアクセスが許可されているかによって異なります。

タイプ	所有者	説明
プライベート	リンクを作成したユーザー。次のビューによって所有権データを表示できます。 ■ DBA_DB_LINKS ■ ALL_DB_LINKS ■ USER_DB_LINKS	ローカル・データベースの特定のスキーマにリンクを作成します。プライベート・データベース・リンクの所有者またはスキーマ内の PL/SQL サブプログラムのみが、このリンクを使用して、対応するリモート・データベース内のデータベース・オブジェクトにアクセスできます。
パブリック	PUBLIC と呼ばれるユーザー。プライベート・データベース・リンクと同じビューによって所有権データを表示できます。	データベース全体にわたるリンクを作成します。データベース内のすべてのユーザーと PL/SQL サブプログラムが、パブリック・リンクを使用して、対応するリモート・データベース内のデータベース・オブジェクトにアクセスできます。
グローバル	PUBLIC と呼ばれるユーザー。プライベート・データベース・リンクと同じビューによって所有権データを表示できます。	ネットワーク全体にわたるリンクを作成します。Oracle ネットワークでディレクトリ・サーバーを使用すると、そのディレクトリ・サーバーは、ネットワーク内のすべての Oracle Database に対するグローバル・データベース・リンクを（ネット・サービス名として）自動的に作成および管理します。どのデータベースのユーザーと PL/SQL サブプログラムでも、グローバル・リンクを使用して、対応するリモート・データベース内のオブジェクトにアクセスできます。 **注意：**Oracle Database の以前のリリースでは、グローバル・データベース・リンクはデータベース・リンクと呼ばれ、Oracle Names Server に登録されていました。Oracle Names Server の使用は非推奨になりました。このマニュアルでのグローバル・データベース・リンクは、ディレクトリ・サーバーのネット・サービス名の使用を指します。

分散データベースで利用するデータベース・リンクのタイプは、システムを使用しているアプリケーションの仕様要件によって決まります。リンクの選択時には、次の機能を考慮してください。

リンクのタイプ	機能
プライベート・データベース・リンク	このリンクは、パブリック・リンクやグローバル・リンクよりも安全です。その理由は、このリンクを使用してリモート・データベースにアクセスできるのが、プライベート・リンクの所有者と、同じスキーマ内のサブプログラムのみに限定されるためです。
パブリック・データベース・リンク	多数のユーザーがリモートの Oracle Database へのアクセス・パスを必要とするときは、データベース内のすべてのユーザーが使用できる 1 つのパブリック・データベース・リンクを作成できます。
グローバル・データベース・リンク	Oracle ネットワークでディレクトリ・サーバーを使用すると、管理者は、システムに存在するすべてのデータベースに対するグローバル・データベース・リンクを容易に管理できます。データベース・リンクの管理が集中化され、単純になります。

関連項目：

- 各タイプのデータベース・リンクの作成方法の詳細は、30-8 ページの「リンク・タイプの指定」を参照してください。
- リンクに関する情報へのアクセス方法は、30-18 ページの「データベース・リンク情報の表示」を参照してください。

データベース・リンクのユーザー

リンクを作成するときは、どのユーザーがリモート・データベースに接続してデータにアクセスする必要があるかを決めます。次の表は、データベース・リンクに関係するユーザーのカテゴリについて、それらの違いを説明したものです。

ユーザー・タイプ	説明	リンク作成構文のサンプル
接続ユーザー	固定のユーザー名およびパスワードが指定されていないデータベース・リンクにアクセスするローカル・ユーザー。SYSTEM が問合せでパブリック・リンクにアクセスする場合、接続ユーザーは SYSTEM であり、データベースはリモート・データベースの SYSTEM スキーマに接続します。 **注意：**接続ユーザーは、必ずしもリンクを作成したユーザーである必要はありません。リンクにアクセスしようとしているすべてのユーザーが接続ユーザーになります。	`CREATE PUBLIC DATABASE LINK hq USING 'hq';`
現行ユーザー	CURRENT_USER データベース・リンク内のグローバル・ユーザー。グローバル・ユーザーは、X.509 証明書（SSL 認証方式のエンタープライズ・ユーザー）またはパスワード（パスワード認証方式のエンタープライズ・ユーザー）によって認証される必要があり、リンクに関係する両方のデータベースのユーザーであることが必要です。現行ユーザー・リンクは、Oracle Advanced Security オプションの機能の 1 つです。 グローバル・セキュリティの詳細は、『Oracle Database Advanced Security 管理者ガイド』を参照してください。	`CREATE PUBLIC DATABASE LINK hq CONNECT TO CURRENT_USER using 'hq';`
固定ユーザー	ユーザー名 / パスワードがリンク定義の一部になっているユーザー。リンクに固定ユーザーが含まれている場合は、その固定ユーザーのユーザー名とパスワードがリモート・データベースへの接続に使用されます。	`CREATE PUBLIC DATABASE LINK hq CONNECT TO jane IDENTIFIED BY doe USING 'hq';`

関連項目： リンクを作成するユーザーを指定する方法の詳細は、30-10 ページの「リンク・ユーザーの指定」を参照してください。

接続ユーザー・データベース・リンク

接続ユーザー・リンクには、接続文字列が対応付けられていません。接続ユーザー・リンクの利点は、リンクを参照するユーザーが同じユーザーとしてリモート・データベースに接続するため、資格証明がデータ・ディクショナリのリンク定義に格納されている必要がないことです。

接続ユーザー・リンクには、いくつかの欠点があります。これらのリンクでは、ユーザーが接続先のリモート・データベースのアカウントと権限を持っている必要があるため、管理者の権限管理作業が増えます。また、必要以上の権限をユーザーに与えることは、セキュリティの基本概念である最低限の権限、つまり「ユーザーにはユーザー自身のジョブの実行に必要な権限のみを与える」に反することになります。

接続ユーザー・データベース・リンクの使用許可はいくつかの要因によって左右されますが、最も重要な要因は認証方式、つまりユーザーがパスワードを使用してデータベースによって認証されるのか、あるいはオペレーティング・システムまたはネットワークの認証サービスによって外部認証されるのかということです。ユーザーが外部認証される場合、接続ユーザー・データベース・リンクの使用許可は、リモート・データベースがユーザーのリモート認証を承認するかどうかという要因によっても左右されます。このリモート認証は、REMOTE_OS_AUTHENT 初期化パラメータによって設定します。

REMOTE_OS_AUTHENT パラメータは、次のように働きます。

REMOTE_OS_AUTHENT の値	結果
リモート・データベースに対して TRUE	外部認証方式のユーザーは、接続ユーザー・データベース・リンクを使用してリモート・データベースに接続できます。
リモート・データベースに対して FALSE	外部認証方式のユーザーは、Oracle Advanced Security オプションでサポートされているセキュリティ保護されたプロトコルまたはネットワーク認証サービスを使用しないかぎり、接続ユーザー・データベース・リンクを使用してリモート・データベースに接続することはできません。

注意： REMOTE_OS_AUTHENT 初期化パラメータは非推奨です。下位互換性のためにのみ維持されています。

固定ユーザー・データベース・リンク

固定ユーザー・リンクの利点は、接続文字列で指定したユーザーのセキュリティ・コンテキストを使用して、プライマリ・データベースのユーザーをリモート・データベースに接続することです。たとえば、ローカル・ユーザー joe は、固定ユーザー scott とパスワード tiger を指定したパブリック・データベース・リンクを joe のスキーマ内に作成できます。jane がこの固定ユーザー・リンクを使用して問合せを発行すると、ローカル・データベースのユーザーである jane が、scott/tiger としてリモート・データベースに接続します。

固定ユーザー・リンクでは、接続文字列にユーザー名とパスワードが対応付けられています。ユーザー名とパスワードは、データ・ディクショナリ表の別のリンク情報に格納されます。

現行ユーザー・データベース・リンク

現行ユーザー・データベース・リンクは、グローバル・ユーザーを利用します。グローバル・ユーザーは、X.509 証明書またはパスワードによって認証される必要があり、リンクに関係する両方のデータベースのユーザーであることが必要です。

CURRENT_USER リンクを実行するユーザーは、必ずしもグローバル・ユーザーである必要はありません。たとえば、jane が買掛管理データベースへのパスワードによって（グローバル・ユーザーとしてではなく）認証される場合、jane はストアド・プロシージャにアクセスして、hq データベースからデータを取得できます。プロシージャは現行ユーザー・データベース・リンクを使用しますが、このリンクによって jane はグローバル・ユーザー scott として hq に接続します。ユーザー scott はグローバル・ユーザーであり、SSL 上の証明書を介して認証されますが、jane はグローバル・ユーザーではなく、SSL では認証されません。

現行ユーザー・データベース・リンクによって、次のような結果になることに注意してください。

- 現行ユーザー・データベース・リンクに対してストアド・オブジェクトの内部以外からアクセスしている場合、現行ユーザーは、リンクにアクセスしている接続ユーザーと同じになります。たとえば、scott が現行ユーザー・リンクを介して SELECT 文を発行した場合、現行ユーザーは scott になります。
- データベース・リンクにアクセスするプロシージャ、ビュー、トリガーなどのストアド・オブジェクトを実行すると、現行ユーザーは、そのストアド・オブジェクトをコールしたユーザーではなく、オブジェクトを所有しているユーザーになります。たとえば、jane がプロシージャ scott.p（scott によって作成されたプロシージャ）をコールし、コールされたプロシージャの内部から現行ユーザー・リンクが確立される場合、リンクの現行ユーザーは scott になります。
- ストアド・オブジェクトが実行者権限のファンクション、プロシージャまたはパッケージである場合は、実行者の認可 ID を使用してリモート・ユーザーとして接続します。たとえば、ユーザー jane がプロシージャ scott.p（scott によって作成された実行者権限のプロシージャ）をコールし、プロシージャ scott.p の内部からリンクが確立される場合、リンクの現行ユーザーは jane になります。

- データベースにエンタープライズ・ユーザーとして接続して、共有のグローバル・スキーマ内に存在するストアド・プロシージャで現行ユーザー・リンクを使用することはできません。たとえば、ユーザー jane がデータベース hq 上の共有スキーマ guest 内のストアド・プロシージャにアクセスしている場合、jane は、このスキーマ内で現行ユーザー・リンクを使用してリモート・データベースにログインすることはできません。

関連項目：

- データベース・リンクに関連するセキュリティ上の問題の詳細は、29-19 ページの「分散データベースのセキュリティ」を参照してください。
- 『Oracle Database Advanced Security 管理者ガイド』
- 実行者権限のファンクション、プロシージャまたはパッケージの詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

データベース・リンクの作成：例

データベース・リンクを作成するには、CREATE DATABASE LINK 文を使用します。次の表は、ローカル・データベース内でリモートの sales.us.americas.acme_auto.com データベースへのデータベース・リンクを作成する SQL 文の例です。

SQL 文	接続先のデータベース	接続時のユーザー	リンク・タイプ
CREATE DATABASE LINK sales.us.americas.acme_auto.com USING 'sales_us';	sales（ネット・サービス名 sales_us を使用）	接続ユーザー	プライベート 接続ユーザー
CREATE DATABASE LINK foo CONNECT TO CURRENT_USER USING 'am_sls';	sales（サービス名 am_sls を使用）	現行グローバル・ユーザー	プライベート 現行ユーザー
CREATE DATABASE LINK sales.us.americas.acme_auto.com CONNECT TO scott IDENTIFIED BY tiger USING 'sales_us';	sales（ネット・サービス名 sales_us を使用）	scott（パスワード tiger を使用）	プライベート 固定ユーザー
CREATE PUBLIC DATABASE LINK sales CONNECT TO scott IDENTIFIED BY tiger USING 'rev';	sales（ネット・サービス名 rev を使用）	scott（パスワード tiger を使用）	パブリック 固定ユーザー
CREATE SHARED PUBLIC DATABASE LINK sales.us.americas.acme_ auto.com CONNECT TO scott IDENTIFIED BY tiger AUTHENTICATED BY anupam IDENTIFIED BY bhide USING 'sales';	sales（ネット・サービス名 sales を使用）	scott（パスワード tiger を使用。認証用としてユーザー名 anupam とパスワード bhide を使用）	共有パブリック 固定ユーザー

関連項目：

- リンクの作成方法の詳細は、30-7 ページの「データベース・リンクの作成」を参照してください。
- CREATE DATABASE LINK 文の構文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

スキーマ・オブジェクトとデータベース・リンク

データベース・リンクを作成した後、リモート・データベースのオブジェクトにアクセスするSQL 文を実行できます。たとえば、データベース・リンク foo を使用してリモート・オブジェクト emp にアクセスするには、次の文を発行します。

```
SELECT * FROM emp@foo;
```

特定のリモート・オブジェクトにアクセスするには、リモート・データベース内での認可も必要です。

データベース・リンクを使用して適切な構成のオブジェクト名を作成することは、分散システムにおいて欠かせないデータ操作の 1 つです。

データベース・リンクを使用したスキーマ・オブジェクトの命名

Oracle Database は、グローバル・データベース名を使用してスキーマ・オブジェクトにグローバルな名前を付ける際、次のスキームを使用します。

schema.schema_object@global_database_name

各項目の意味は次のとおりです。

- *schema* は、データの論理構造（スキーマ・オブジェクト）の集まりです。スキーマはデータベース・ユーザーによって所有され、そのユーザーと同じ名前を持ちます。ユーザーはそれぞれ 1 つのスキーマを所有します。
- *schema_object* は、表、索引、ビュー、シノニム、プロシージャ、パッケージ、データベース・リンクなどの論理データ構造です。
- *global_database_name* は、リモート・データベースを一意に識別する名前です。この名前は、リモート・データベースの初期化パラメータ DB_NAME および DB_DOMAIN を連結したものと同じである必要があります。ただし、パラメータ GLOBAL_NAMES を FALSE に設定している場合は、任意の名前を使用できます。

たとえば、ユーザーまたはアプリケーションは、データベース sales.division3.acme.com へのデータベース・リンクを次のように使用して、リモート・データを参照できます。

```
SELECT * FROM scott.emp@sales.division3.acme.com;  # emp table in scott's schema
SELECT loc FROM scott.dept@sales.division3.acme.com;
```

GLOBAL_NAMES を FALSE に設定している場合は、sales.division3.acme.com へのリンクに対して任意の名前を使用できます。たとえば、リンク foo をコールできます。次に、リモート・データベースに次のようにアクセスできます。

```
SELECT name FROM scott.emp@foo;  # link name different from global name
```

リモート・スキーマ・オブジェクトへのアクセスに必要な認可

リモート・スキーマ・オブジェクトにアクセスするには、リモート・データベース内でそのオブジェクトへのアクセス権を付与される必要があります。また、リモート・オブジェクトの更新、挿入または削除を実行するには、オブジェクトに対する SELECT 権限と、UPDATE、INSERT または DELETE 権限を付与される必要があります。データベースにはリモート記述機能がないため、ローカル・オブジェクトにアクセスする場合とは異なり、リモート・オブジェクトにアクセスするには SELECT 権限が必要です。データベースでは、構造を判断するためにリモート・オブジェクトの SELECT * を実行する必要があります。

スキーマ・オブジェクトのシノニム

Oracle Database では、シノニムを作成して、データベース・リンク名をユーザーから隠すことができます。シノニムを使用すると、ローカル・データベースの表にアクセスする場合と同じ構文を使用して、リモート・データベースの表にアクセスできます。たとえば、リモート・データベース内の表に対して次の問合せを発行するとします。

```
SELECT * FROM emp@hq.acme.com;
```

この場合、emp@hq.acme.com に対応するシノニム emp を作成すれば、同じデータにアクセスする際に次の問合せを発行できます。

```
SELECT * FROM emp;
```

> **関連項目：** データベース・リンクを使用して指定したオブジェクトのシノニムを作成する方法の詳細は、30-22 ページの「シノニムを使用した位置の透過性の作成」を参照してください。

スキーマ・オブジェクトの名前解決

スキーマ・オブジェクトへのアプリケーション参照を解決する（このプロセスを**名前解決**と呼びます）ために、データベースではオブジェクト名を階層的に構成しています。たとえば、データベースでは、データベース内部の各スキーマは一意の名前を持ち、スキーマ内部では各オブジェクトが一意の名前を持つことが保証されています。そのため、スキーマ・オブジェクトの名前はデータベースの内部で常に一意になります。また、データベースはオブジェクトのローカル名へのアプリケーション参照を解決します。

分散データベースでは、表などのスキーマ・オブジェクトに対してシステム内のすべてのアプリケーションからアクセスが可能です。データベースは、グローバル・データベース名による階層ネーミング・モデルを拡張して**グローバル・オブジェクト名**を効果的に作成することによって、分散データベース・システム内のスキーマ・オブジェクトへの参照を解決します。たとえば、問合せでリモートの表を参照する場合は、その完全修飾名（表が存在しているデータベースを含む）を指定します。

たとえば、ローカル・データベースにユーザー SYSTEM として接続するとします。

```
CONNECT SYSTEM@sales1
```

次に、データベース・リンク hq.acme.com を使用して次の文を発行し、リモート・データベース hq の scott スキーマおよび jane スキーマにあるオブジェクトにアクセスします。

```
SELECT * FROM scott.emp@hq.acme.com;
INSERT INTO jane.accounts@hq.acme.com (acc_no, acc_name, balance)
  VALUES (5001, 'BOWER', 2000);
UPDATE jane.accounts@hq.acme.com
  SET balance = balance + 500;
DELETE FROM jane.accounts@hq.acme.com
  WHERE acc_name = 'BOWER';
```

データベース・リンクの制限事項

データベース・リンクを使用して次の操作を実行することはできません。

- リモート・オブジェクトへの権限の付与。
- 一部のリモート・オブジェクトに対する DESCRIBE の実行。ただし、次のリモート・オブジェクトは DESCRIBE をサポートしています。
  - 表
  - ビュー
  - プロシージャ
  - ファンクション
- リモート・オブジェクトの分析。
- 参照整合性の定義または規定。
- リモート・データベース内のユーザーへのロールの付与。
- リモート・データベースにおけるデフォルト以外のロールの取得。たとえば、jane がローカル・データベースに接続し、scott として接続する固定ユーザー・リンクを使用したストアド・プロシージャを実行する場合、jane はリモート・データベースにおける scott のデフォルト・ロールを受け取ります。jane は、SET ROLE を発行してデフォルト以外のロールを取得することはできません。
- 共有サーバー接続を使用したハッシュ結合による問合せの実行。
- SSL、パスワードまたは Windows NT のシステム固有の認証によって認証されていない現行ユーザー・リンクの使用。

分散データベースの管理

ここでは、Oracle Database の分散データベース・システムにおけるデータベース管理について説明します。この項の内容は、次のとおりです。

関連項目：

- 同機種システムの管理方法の詳細は、第 30 章「分散データベースの管理」を参照してください。
- 異機種間サービスの概念の詳細は、『Oracle Database Heterogeneous Connectivity 管理者ガイド』を参照してください。

サイト自律性

サイト自律性とは、分散データベース内の各サーバーが他のすべてのデータベースから独立して管理されることを意味します。複数のデータベースが協調して動作する場合もありますが、各データベースはそれぞれ別々のデータ・リポジトリであり、個別に管理されます。Oracle Database の分散データベースのサイト自律性には、次のような利点があります。

- 独立性を維持する必要がある個々の企業またはグループの論理的な組織構造を、システムのノード群として反映できます。
- ローカル管理者がそれぞれ対応するローカル・データを管理します。したがって、個々のデータベース管理者（DBA）の責任範囲が小さくなり、管理しやすくなります。
- 障害が単独で発生するため、分散データベースの他のノードに損傷を与える可能性が低くなります。1 つのデータベース障害によってすべての分散操作が停止したり、パフォーマンスのボトルネックが生じることはありません。
- 孤立したシステム障害が発生した際、管理者は、システム内の他のノードとは独立してリカバリできます。
- 各ローカル・データベースにはデータ・ディクショナリが存在します。ローカル・データへのアクセスにグローバル・カタログは不要です。
- 各ノードで独立してソフトウェアをアップグレードできます。

Oracle Database では分散データベース・システム内の各データベースを独立して管理できますが、システムのグローバルな要件を無視することのないようにしてください。たとえば、次のような作業が必要になることがあります。

- サーバー間接続を容易にするために作成したリンクをサポートするため、追加のユーザー・アカウントを各データベースに作成する。
- `COMMIT_POINT_STRENGTH` や `OPEN_LINKS` などの追加の初期化パラメータを設定する。

分散データベースのセキュリティ

データベースでは、非分散データベース環境で使用できる次のセキュリティ機能が、分散データベース・システムでもすべてサポートされています。

- ユーザーおよびロールのパスワード認証。
- ユーザーおよびロールに対する一部の外部認証タイプ。サポートされている外部認証タイプは、次のとおりです。
  - Kerberos バージョン 5（接続ユーザー・リンク用）
  - DCE（接続ユーザー・リンク用）
- クライアント / サーバー間およびサーバー間接続におけるログイン・パケットの暗号化。

次の項では、Oracle Database の分散データベース・システムの構成時に考慮する必要がある内容について説明します。

- データベース・リンクを介した認証
- パスワードなしの認証
- ユーザー・アカウントおよびロールのサポート
- ユーザーと権限の集中管理
- データの暗号化

> **関連項目：** 外部認証の詳細は、『Oracle Database Advanced Security 管理者ガイド』を参照してください。

データベース・リンクを介した認証

データベース・リンクにはプライベートとパブリックの 2 種類があり、それぞれについて**認証ありの**場合と**認証なしの**場合があります。パブリック・リンクを作成するには、リンク作成文で PUBLIC キーワードを指定します。たとえば、次の文を発行できます。

```
CREATE PUBLIC DATABASE LINK foo USING 'sales';
```

認証ありのリンクを作成するには、データベース・リンク作成文で CONNECT TO 句、AUTHENTICATED BY 句、あるいはこれら両方の句を指定します。たとえば、次の文を発行できます。

```
CREATE DATABASE LINK sales CONNECT TO scott IDENTIFIED BY tiger USING 'sales';
CREATE SHARED PUBLIC DATABASE LINK sales CONNECT TO nick IDENTIFIED BY firesign
     AUTHENTICATED BY david IDENTIFIED BY bowie USING 'sales';
```

次の表は、ユーザーがリンクを介してリモート・データベースにアクセスする方法を示しています。

リンク・タイプ	認証	セキュリティ・アクセス
プライベート	いいえ	データベースでは、リモート・データベースに接続するときに、ローカル・セッションから取得したセキュリティ情報（ユーザー ID/ パスワード）を使用します。そのため、このリンクは接続ユーザー・データベース・リンクです。2 つのデータベース間で、パスワードが同期化されている必要があります。
プライベート	はい	ユーザー ID/ パスワードがローカル・セッションのコンテキストからではなく、リンク定義から取得されます。そのため、このリンクは固定ユーザー・データベース・リンクです。 この構成では、2 つのデータベース間で異なるパスワードを使用できますが、ローカル・データベース・リンクのパスワードはリモート・データベースのパスワードと一致している必要があります。
パブリック	いいえ	動作はプライベートの非認証リンクとほぼ同じですが、すべてのユーザーがこのリモート・データベースへのポインタを参照できる点が異なります。
パブリック	はい	ローカル・データベースのすべてのユーザーがリモート・データベースにアクセスでき、すべてのユーザーが同じユーザー ID/ パスワードを使用して接続します。

パスワードなしの認証

接続ユーザーまたは現行ユーザーのデータベース・リンクを使用するときは、Kerberos などの外部認証ソースを使用して**エンド・トゥ・エンドのセキュリティ**を確立できます。エンド・トゥ・エンド認証では、資格証明がサーバー間で渡され、同じドメインに属するデータベース・サーバーによって資格証明を認証できます。たとえば、jane がローカル・データベースで外部認証され、接続ユーザー・リンクを使用して jane としてリモート・データベースに接続しようとする場合、ローカル・サーバーはリモート・データベースにセキュリティ・チケットを渡します。

ユーザー・アカウントおよびロールのサポート

分散データベース・システムでは、システムを使用するアプリケーションのサポートに必要なユーザー・アカウントおよびロールについて慎重に計画する必要があります。特に、次の点に注意してください。

- サーバー間接続の確立に必要なユーザー・アカウントは、分散データベース・システムのすべてのデータベースで使用可能である必要があります。
- 分散データベース・アプリケーションのユーザーに対してアプリケーション権限を使用可能にするために必要なロールは、分散データベース・システムのすべてのデータベースに存在する必要があります。

分散データベース・システム内のノードに対応するデータベース・リンクを作成する際は、それらのリンクを使用するサーバー間接続をサポートするために、各サイトでどのユーザー・アカウントとロールが必要になるかを決めてください。

通常、分散環境では、ユーザーは多数のネットワーク・サービスへのアクセスが必要になります。個々のユーザーが個々のネットワーク・サービスにアクセスするために個別の認証を構成する必要があるときは、特に大規模なシステムの場合にセキュリティの管理が難しくなることがあります。

> **関連項目：** 各種データベース・リンクをシステムでサポートするために使用可能にする必要があるユーザー・アカウントの詳細は、30-7 ページの「データベース・リンクの作成」を参照してください。

ユーザーと権限の集中管理

データベースには、分散システムに関係するユーザーおよび権限を管理するための様々な手段が用意されています。たとえば、次のような方法があります。

- エンタープライズ・ユーザー管理。SSL を介して、またはパスワードを使用して認証されるグローバル・ユーザーを作成し、それらのユーザーと権限を独立したディレクトリ・サービスによってディレクトリ内で管理できます。
- ネットワーク認証サービス。この一般的な方法によって、分散環境のセキュリティ管理が容易になります。Oracle Advanced Security オプションを使用することで、Oracle Net と Oracle Database の分散データベース・システムのセキュリティを強化できます。Oracle 以外の認証ソリューションの例としては、Windows NT のシステム固有の認証があります。

  関連項目：

  - 『Oracle Database Advanced Security 管理者ガイド』
  - 『Oracle Database エンタープライズ・ユーザー・セキュリティ管理者ガイド』

スキーマに依存するグローバル・ユーザー ユーザーと権限を集中管理する方法の 1 つとして、次のユーザーを作成できます。

- 中央ディレクトリ内のグローバル・ユーザー
- グローバル・ユーザーが接続する必要のあるすべてのデータベース内のユーザー

たとえば、次の SQL 文を使用して `fred` というグローバル・ユーザーを作成できます。

```
CREATE USER fred IDENTIFIED GLOBALLY AS 'CN=fred adams,O=Oracle,C=England';
```

この解決方法を使用すると、1 つのグローバル・ユーザーを中央ディレクトリによって認証できます。

スキーマに依存するグローバル・ユーザーによる解決方法では、`fred` というユーザーを、そのユーザーがアクセスする必要のあるすべてのデータベースに作成する必要があります。ユーザーのほとんどはアプリケーション・スキーマにアクセスするための権限は必要とするものの、自分自身のスキーマは必要としないため、すべてのグローバル・ユーザーに対し、各データベースにそれぞれのアカウントを作成するのは大変な手間となります。この問題を回避するため、データベースではスキーマに依存しないユーザーもサポートしています。このユーザーは、すべてのデータベース内にある単一の汎用スキーマにアクセスするグローバル・ユーザーです。

スキーマに依存しないグローバル・ユーザー データベースは、グローバル・ユーザーをエンタープライズ・ディレクトリ・サービスによって集中管理する機能をサポートしています。このディレクトリ内で管理されるユーザーのことを、**エンタープライズ・ユーザー**と呼びます。このディレクトリには、次のことに関する情報が格納されています。

- エンタープライズ・ユーザーが分散システムのどのデータベースにアクセスできるのか。
- エンタープライズ・ユーザーが各データベースのどのロールを使用できるのか。
- エンタープライズ・ユーザーが各データベースのどのスキーマに接続できるのか。

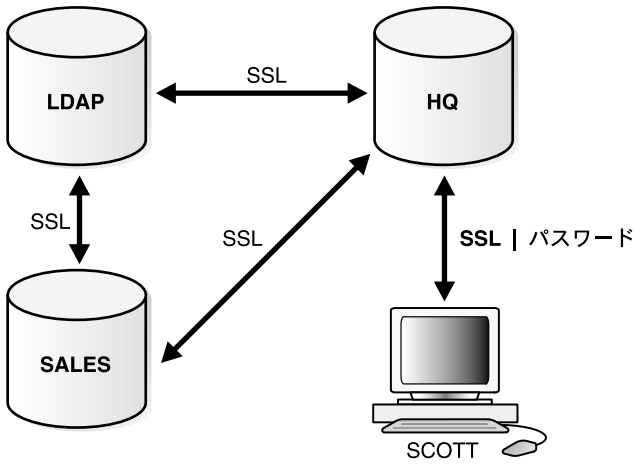
各データベースの管理者は、エンタープライズ・ユーザーが接続する必要のあるデータベースごとに各エンタープライズ・ユーザーのグローバル・ユーザー・アカウントを作成する必要はありません。そのかわりに、複数のエンタープライズ・ユーザーが**共有スキーマ**と呼ばれる同じデータベース・スキーマに接続できます。

注意： 共有スキーマ内の現行ユーザー・データベース・リンクにはアクセスできません。

たとえば、jane、bill および scott が全員、人事管理アプリケーションを使用しているとします。hq アプリケーション・オブジェクトはすべて、hq データベースの guest スキーマに格納されています。この場合、共有スキーマとして使用するローカルのグローバル・ユーザー・アカウントを作成できます。このグローバル・ユーザー名、つまり共有スキーマ名は guest です。jane、bill および scott はすべて、ディレクトリ・サービス内でエンタープライズ・ユーザーとして作成されます。また、3 名はディレクトリ内の guest スキーマにマップされ、hq アプリケーションの異なる認可が割り当てられることが可能です。

図 29-5 は、エンタープライズ・ディレクトリ・サービスを使用したグローバル・ユーザーのセキュリティの例を示しています。

図 29-5 グローバル・ユーザーのセキュリティ



エンタープライズ・ディレクトリ・サービスに、hq および sales のエンタープライズ・ユーザーに関する次の情報が格納されているとします。

データベース	ロール	スキーマ	エンタープライズ・ユーザー
tb	clerk1	guest	bill scott
sales	clerk2	guest	jane scott

また、hq および sales のローカル管理者が次の文を発行したとします。

データベース	CREATE 文
hq	CREATE USER guest IDENTIFIED GLOBALLY AS ''; CREATE ROLE clerk1 GRANT select ON emp; CREATE PUBLIC DATABASE LINK sales_link CONNECT AS CURRENT_USER USING 'sales';
sales	CREATE USER guest IDENTIFIED GLOBALLY AS ''; CREATE ROLE clerk2 GRANT select ON dept;

ここで、sales に関係する分散トランザクションを実行するために、エンタープライズ・ユーザー scott がローカル・データベース hq への接続を要求するとします。このとき、次の手順が実行されます（ただし、必ずしもこの順序どおりとはかぎりません）。

1. エンタープライズ・ユーザー scott が、SSL またはパスワードを使用して認証されます。
2. ユーザー scott が次の文を発行します。

```
SELECT e.ename, d.loc
FROM emp e, dept@sales_link d
WHERE e.deptno=d.deptno;
```

3. データベース hq と sales が、SSL を使用して相互に認証します。
4. データベース hq はエンタープライズ・ディレクトリ・サービスを問い合せて、エンタープライズ・ユーザー scott が hq にアクセスできるかどうかを判断し、scott がロール clerk1 を使用してローカル・スキーマ guest にアクセスできることを確認します。
5. データベース sales はエンタープライズ・ディレクトリ・サービスを問い合せて、エンタープライズ・ユーザー scott が sales にアクセスできるかどうかを判断し、scott がロール clerk2 を使用してローカル・スキーマ guest にアクセスできることを確認します。
6. エンタープライズ・ユーザー scott は sales にログインし、ロール clerk2 を使用して guest スキーマにアクセスします。そこで SELECT を発行し、必要な情報を取得してその情報を hq に送信します。
7. データベース hq は要求されたデータを sales から受信し、それをクライアント scott に返します。

> **関連項目：** エンタープライズ・ユーザーのセキュリティの詳細は、『Oracle Database エンタープライズ・ユーザー・セキュリティ管理者ガイド』を参照してください。

データの暗号化

Oracle Advanced Security オプションでは、データが解読または改ざんされないように、Oracle Net とその関連製品でネットワーク・データの暗号化とチェックサムを使用できます。この機能では、RSA Data Security 社の RC4 またはデータ暗号化規格（DES）の暗号化アルゴリズムを使用することによって、データが不正に読み取られることを防ぎます。

データが転送中に改ざん、削除または再現されなかったことを保証するために、Oracle Advanced Security オプションのセキュリティ・サービスは、暗号的に安全なメッセージ・ダイジェストを生成し、ネットワーク上に送られる各パケットにそのダイジェストを含めることができます。

> **関連項目：** Oracle Advanced Security オプションのこれらの機能およびその他の機能の詳細は、『Oracle Database Advanced Security 管理者ガイド』を参照してください。

データベース・リンクの監査

操作の監査は、常にローカルで実行する必要があります。つまり、適切な監査オプションがそれぞれのデータベースで設定されている場合は、ユーザーがローカル・データベース内で操作を実行し、データベース・リンクを介してリモート・データベースにアクセスすると、ローカルの操作はローカル・データベースで監査され、リモートの操作はリモート・データベースで監査されます。

リモート・データベースでは、正常終了した接続要求とその後の SQL 文が別のサーバーからのものか、またはローカルに接続しているクライアントからのものかを判断することはできません。たとえば、次のような場合を考えてみます。

- 固定ユーザー・リンク `hq.acme.com` により、ローカル・ユーザー `jane` がリモート・ユーザー `scott` としてリモートの `hq` データベースに接続します。
- ユーザー `scott` は、リモート・データベースで監査されます。

リモート・データベース・セッション中に実行される操作は、あたかも `scott` が `hq` にローカルに接続し、そこで同じ操作を実行しているかのように監査されます。`jane` がリモート・データベースで何を実行しているのかを監査する場合は、リモート・データベースで監査オプションを設定し、リンクに埋め込まれているユーザー名（この場合は `hq` データベースの `scott`）の操作を捕捉する必要があります。

注意： グローバル・ユーザーに対応するグローバル・ユーザー名を監査できます。

リモート・オブジェクトに対してローカルの監査オプションを設定することはできません。したがって、リモート・オブジェクトへのアクセスをリモート・データベースで監査することはできますが、データベース・リンクの使用は監査できません。

管理ツール

DBA は、Oracle Database の分散データベース・システムを管理する際にいくつかのツールを選択できます。

- Enterprise Manager
- サード・パーティ製管理ツール
- SNMP サポート

Enterprise Manager

Enterprise Manager は、GUI を備えた Oracle Database の管理ツールです。操作性に優れたインタフェースを介して、分散データベースの管理機能を提供します。Enterprise Manager は、次の操作に使用できます。

- 複数のデータベースの管理。Enterprise Manager を使用して、1 つのデータベースを管理したり、複数のデータベースを同時に管理できます。
- データベース管理作業の集中化。世界中のあらゆる場所のあらゆる Oracle Database プラットフォームで稼働しているローカルおよびリモートの両方のデータベースを管理できます。また、Oracle Net がサポートしている任意のネットワーク・プロトコルでこれらの Oracle Database プラットフォームを接続できます。
- SQL、PL/SQL および Enterprise Manager コマンドの動的な実行。Enterprise Manager を使用して、文を入力、編集および実行できます。実行した文の履歴も保持されます。

  これにより、それらの文を再入力しなくても再実行できるので、分散データベース・システムで非常に長い文を繰り返し実行する必要がある場合に特に便利です。
- グローバル・ユーザー、グローバル・ロール、エンタープライズ・ディレクトリ・サービスなどのセキュリティ機能の管理。

サード・パーティ製管理ツール

現在、Oracle Database およびネットワークの管理に役立つ製品が、60 を超える企業から 150 以上出荷されており、真にオープンな環境を実現しています。

SNMP サポート

ネットワーク管理機能以外にも、Oracle では **Simple Network Management Protocol**（SNMP）をサポートしており、SNMP ベースの任意のネットワーク管理システムによって Oracle Database サーバーを検索および問合せできます。SNMP は、次に示す多数の一般的なネットワーク管理システムの基盤となる公認規格です。

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

分散システムでのトランザクション処理

トランザクションとは、1 人のユーザーが実行する 1 つ以上の SQL 文によって構成された論理作業単位のことです。トランザクションは、ユーザーの最初に実行可能な SQL 文で開始し、そのユーザーによってコミットまたはロールバックされたときに終了します。

リモート・トランザクションは、1 つのリモート・ノードにアクセスする文のみで構成されます。**分散トランザクション**は、複数のノードにアクセスする文で構成されます。

次の項では、トランザクション処理における重要な概念を定義し、トランザクションが分散データベースのデータにアクセスする仕組みについて説明します。

- リモート SQL 文
- 分散 SQL 文
- リモート文と分散型の文の共有 SQL
- リモート・トランザクション
- 分散トランザクション
- 2 フェーズ・コミット・メカニズム
- データベース・リンクの名前解決
- スキーマ・オブジェクトの名前解決

リモート SQL 文

リモート問合せ文とは、そのすべてが同一のリモート・ノードに存在している 1 つ以上のリモート表から情報を選択する問合せのことです。たとえば、次の問合せは、リモートの sales データベースの scott スキーマにある dept 表のデータにアクセスします。

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

リモート更新文とは、そのすべてが同一のリモート・ノードに存在している 1 つ以上の表のデータを変更する更新のことです。たとえば、次の問合せは、リモートの sales データベースの scott スキーマにある dept 表を更新します。

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

> **注意：** リモート更新には 1 つ以上のリモート・ノードからデータを取得する副問合せを含めることができますが、更新は 1 つのリモート・ノードでのみ発生するため、その文はリモート更新として分類されます。

分散 SQL 文

分散問合せ文は、複数のノードから情報を取得します。たとえば、次の問合せは、ローカル・データベースのデータと同時にリモートの sales データベースのデータにもアクセスします。

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

分散更新文は、複数のノードのデータを変更します。分散更新を行うには、プロシージャやトリガーなど、異なるノードのデータにアクセスする複数のリモート更新を含む PL/SQL サブプログラム・ユニットを使用します。たとえば、次の PL/SQL プログラム・ユニットは、ローカル・データベースとリモートの sales データベースにある表を更新します。

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

データベースによってプログラム内の文がリモート・ノードに送られると、それらの文の実行はユニットとして成功または失敗します。

リモート文と分散型の文の共有 SQL

共有 SQL を使用したリモート文または分散型の文の仕組みは、本質的にはローカル文の仕組みと同じです。SQL テキストは必ず一致している必要があり、参照先のオブジェクトも必ず一致している必要があります。可能であれば、任意の文または分解された問合せをローカルまたはリモートで処理するために共有 SQL 領域を使用できます。

> **関連項目：** 共有 SQL の詳細は、『Oracle Database 概要』を参照してください。

リモート・トランザクション

リモート・トランザクションには 1 つ以上のリモート文があり、そのすべてが 1 つのリモート・ノードを参照しています。たとえば、次のトランザクションには 2 つの文があり、それぞれがリモートの sales データベースにアクセスします。

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

分散トランザクション

分散トランザクションとは、1 つ以上の文からなり、それらが個別に、またはグループとして、分散データベースの複数のノードのデータを更新するようなトランザクションのことです。たとえば、このトランザクションは、ローカル・データベースとリモートの sales データベースを更新します。

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

注意： トランザクションのすべての文が 1 つのリモート・ノードのみを参照している場合、そのトランザクションは分散トランザクションではなくリモート・トランザクションです。

2 フェーズ・コミット・メカニズム

データベースは、トランザクションが分散か非分散かにかかわらず、トランザクション内のすべての文がユニットとしてコミットまたはロールバックすることを保証します。実行中のトランザクションの結果は、すべてのノードの全トランザクションに対して不可視であり、このような透過性は、問合せや更新、リモート・プロシージャ・コールなど、あらゆるタイプの操作を含むトランザクションにおいて成り立つことが必要です。

非分散データベースにおけるトランザクション管理の一般的なメカニズムの詳細は、『Oracle Database 概要』を参照してください。分散データベースでは、データベースはネットワーク上で同じ特性を持つトランザクション管理を連携させる必要があり、ネットワークやシステムに障害が発生しても、データ整合性を維持する必要があります。

データベースの **2 フェーズ・コミット**・メカニズムは、分散トランザクションに参加しているすべてのデータベース・サーバーが、トランザクション内の文をすべてコミットするか、またはすべてロールバックすることを保証します。また、2 フェーズ・コミット・メカニズムにより、整合性制約、リモート・プロシージャ・コールおよびトリガーによって実行される暗黙的なデータ操作言語（DML）操作が保護されます。

関連項目： Oracle Database の 2 フェーズ・コミット・メカニズムの詳細は、第 32 章「分散トランザクションの概念」を参照してください。

データベース・リンクの名前解決

グローバル・オブジェクト名とは、データベース・リンクを使用して指定されたオブジェクトのことです。グローバル・オブジェクト名の必須構成要素は、次のとおりです。

- オブジェクト名
- データベース名
- ドメイン

次の表は、明示的に指定されるグローバル・データベース・オブジェクト名の構成要素を示しています。

文	オブジェクト	データベース	ドメイン
SELECT * FROM joan.dept@sales.acme.com	dept	sales	acme.com
SELECT * FROM emp@mktg.us.acme.com	emp	mktg	us.acme.com

SQL 文にグローバル・オブジェクト名への参照が含まれていると、データベースでは、必ずグローバル・オブジェクト名の中で指定されているデータベース名と一致する名前を持つデータベース・リンクを検索します。たとえば、次の文を発行した場合を考えます。

```
SELECT * FROM scott.emp@orders.us.acme.com;
```

この場合は、データベースによって orders.us.acme.com というデータベース・リンクが検索されます。データベースはこの操作を実行して、指定されたリモート・データベースへのパスを判断します。

データベースは、一致するデータベース・リンクを常に次の順序で検索します。

1. SQL 文を発行したユーザーのスキーマ内のプライベート・データベース・リンク
2. ローカル・データベース内のパブリック・データベース・リンク
3. グローバル・データベース・リンク（ディレクトリ・サーバーが使用可能な場合のみ）

グローバル・データベース名が完全なときの名前解決

完全なグローバル・データベース名が指定された次の SQL 文を発行するとします。

```
SELECT * FROM emp@prod1.us.oracle.com;
```

この場合は、データベース名（prod1）とドメイン構成要素（us.oracle.com）の両方を指定しているため、データベースは、プライベート、パブリックおよびグローバルのデータベース・リンクを検索します。データベースは、指定されたグローバル・データベース名に一致するリンクのみを検索します。

グローバル・データベース名が部分的なときの名前解決

ドメインの任意の一部を指定した場合、データベースでは、完全なグローバル・データベース名を指定したものとみなされます。SQL 文で部分的なグローバル・データベース名を指定した場合（つまり、データベース構成要素のみを指定した場合）、データベースは DB_DOMAIN 初期化パラメータ値を DB_NAME 初期化パラメータ値の後に追加し、完全な名前を構成します。たとえば、次の文を発行した場合を考えます。

```
CONNECT scott@locdb
SELECT * FROM scott.emp@orders;
```

locdb のネットワーク・ドメインが us.acme.com の場合、データベースはこのドメインを orders の後に追加し、orders.us.acme.com という完全なグローバル・データベース名を構成します。データベースは、構築されたグローバル名のみに一致するデータベース・リンクを検索します。一致するリンクが見つからない場合、データベースはエラーを返し、SQL 文は実行できません。

グローバル・データベース名をまったく指定しないときの名前解決

グローバル・オブジェクト名がローカル・データベース内のオブジェクトを参照していて、データベース・リンク名がアットマーク（@）を使用して指定されていない場合、データベースは、オブジェクトがローカルであることを自動的に検出して検索を行わないか、またはデータベース・リンクを使用してオブジェクト参照を解決します。たとえば、次の文を発行した場合を考えます。

```
CONNECT scott@locdb
SELECT * from scott.emp;
```

2 番目の文ではデータベース・リンク接続文字列を使用してグローバル・データベース名を指定していないため、データベースは、データベース・リンクを検索しません。

名前解決のための検索の終了

データベースは、最初に一致したデータベース・リンクが見つかったときに、必ずしも一致するデータベース・リンクの検索を停止するとはかぎりません。データベースは、リモート・データベースへの完全なパス（リモート・アカウントとサービス名の両方）がわかるまで、一致するプライベート、パブリックおよびネットワークのデータベース・リンクを検索します。

最初に一致したデータベース・リンクが見つかると、次の表に従ってリモート・スキーマが決定されます。

ユーザー操作	データベースの応答	例
CONNECT 句が指定されていない場合	接続ユーザー・データベース・リンクを使用します。	`CREATE DATABASE LINK k1 USING 'prod'`
CONNECT TO ... IDENTIFIED BY 句が指定されている場合	固定ユーザー・データベース・リンクを使用します。	`CREATE DATABASE LINK k2 CONNECT TO scott IDENTIFIED BY tiger USING 'prod'`
CONNECT TO CURRENT_USER 句が指定されている場合	現行ユーザー・データベース・リンクを使用します。	`CREATE DATABASE LINK k3 CONNECT TO CURRENT_USER USING 'prod'`
USING 句が指定されていない場合	データベース文字列を指定するリンクが見つかるまで検索します。一致するデータベース・リンクが見つかっても文字列がまったく識別されない場合、データベースはエラーを返します。	`CREATE DATABASE LINK k4 CONNECT TO CURRENT_USER`

完全なパスを決定した後、データベースはリモート・セッションを作成します（同じローカル・セッションのために同一の接続がまだオープンになっていない場合）。セッションがすでに存在している場合、データベースはそのセッションを再利用します。

スキーマ・オブジェクトの名前解決

ローカルの Oracle Database が、SQL 文を発行したローカル・ユーザーのために指定のリモート・データベースに接続した後も、あたかもリモート・ユーザーが対応する SQL 文を発行したかのように、オブジェクトの解決処理が続きます。最初に一致したデータベース・リンクが見つかると、次の規則に従ってリモート・スキーマが決定されます。

指定したリンクのタイプ	オブジェクト解決の場所
固定ユーザー・データベース・リンク	リンク作成文で指定されたスキーマ
接続ユーザー・データベース・リンク	接続ユーザーのリモート・スキーマ
現行ユーザー・データベース・リンク	現行ユーザーのスキーマ

オブジェクトが見つからなかった場合、データベースはリモート・データベースのパブリック・オブジェクトをチェックします。オブジェクトを解決できなくても確立されたリモート・セッションは残りますが、SQL 文は実行できず、エラーが返されます。

次に、分散データベース・システムにおけるグローバル・オブジェクトの名前解決の例を示します。以降のすべての例で、示しているようなことを想定します。

グローバル・オブジェクトの名前解決の例 : 完全なオブジェクト名

この例では、データベースが完全なグローバル・オブジェクト名をどのように解決し、プライベートとパブリックの両方のデータベース・リンクを使用して、どのようにリモート・データベースへの適切なパスを決定するのかを示します。この例では、次のことを想定しています。

- リモート・データベースの名前は、sales.division3.acme.com です。
- ローカル・データベースの名前は、hq.division3.acme.com です。
- ディレクトリ・サーバーは使用できません。したがって、グローバル・データベース・リンクも使用できません。
- リモート表 emp は、スキーマ tsmith 内にあります。

次の文が scott によってローカル・データベースで発行された場合を考えます。

```
CONNECT scott@hq

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

その後、JWARD が接続して次の文を発行するとします。

```
CONNECT jward@hq

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

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

データベースは、最後の文を次のように処理します。

1. データベースは、jward の UPDATE 文で完全なグローバル・オブジェクト名が参照されていると判断します。したがって、一致する名前を持つデータベース・リンクの検索がローカル・データベース内で開始されます。
2. データベースは、一致するプライベート・データベース・リンクをスキーマ jward 内で検索します。しかし、プライベート・データベース・リンク jward.sales.division3.acme.com は、リモートの sales データベースへの完全なパスを示しておらず、リモート・アカウントのみを示しています。そのため、データベースは、一致するパブリック・データベース・リンクを続いて検索します。

3. データベースは、パブリック・データベース・リンクを scott のスキーマ内で検索します。データベースは、このパブリック・データベース・リンクからネット・サービス名 dbstring を取得します。
4. データベースは、一致するプライベートの固定ユーザー・データベース・リンクから取得したリモート・アカウントを結合して完全なパスを決定し、次に、ユーザー tsmith/radio としてリモートの sales データベースに接続します。
5. これで、リモート・データベースは、emp 表へのオブジェクト参照を解決できます。データベースは tsmith スキーマ内で検索を行い、参照先の emp 表を検出します。
6. リモート・データベースは文の実行を完了し、その結果をローカル・データベースに返します。

グローバル・オブジェクトの名前解決の例 : 部分的なオブジェクト名

この例では、データベースが部分的なグローバル・オブジェクト名をどのように解決し、プライベートとパブリックの両方のデータベース・リンクを使用してリモート・データベースへの適切なパスをどのように決定するのかを示します。

この例では、次のことを想定しています。

- リモート・データベースの名前は、sales.division3.acme.com です。
- ローカル・データベースの名前は、hq.division3.acme.com です。
- ディレクトリ・サーバーは使用できません。したがって、グローバル・データベース・リンクも使用できません。
- リモート・データベース sales の表 emp はスキーマ tsmith 内にあり、スキーマ scott 内にはありません。
- リモート・データベース sales に emp というパブリック・シノニムが存在します。このシノニムは、リモート・データベース sales の tsmith.emp を指しています。
- 29-30 ページの「グローバル・オブジェクトの名前解決の例 : 完全なオブジェクト名」で示したパブリック・データベース・リンクが、ローカル・データベース hq にすでに作成されています。

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

ローカル・データベース hq で次の文が発行された場合を考えます。

```
CONNECT scott@hq

CREATE DATABASE LINK sales.division3.acme.com;

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

データベースは、最後の DELETE 文を次のように処理します。

1. データベースは、scott の DELETE 文で部分的なグローバル・オブジェクト名が参照されていることを検出します。次のようにローカル・データベースのドメインを使用して、部分的なグローバル・オブジェクト名を完全なグローバル・オブジェクト名に拡張します。

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

2. データベースは、一致する名前を持つデータベース・リンクをローカル・データベース内で検索します。

3. データベースは、スキーマ scott 内で、一致するプライベートの接続ユーザー・リンクを検出しますが、そのプライベート・データベース・リンクにはパスがまったく示されていません。データベースは、接続先のユーザー名 / パスワードをパスのリモート・アカウントとして使用し、一致するパブリックのデータベース・リンクを検索して検出します。

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

4. データベースは、このパブリック・データベース・リンクからサービス名 dbstring を取得します。この時点で、データベースは完全なパスを決定しました。
5. データベースは、リモート・データベースに scott/tiger として接続し、検索を行いますが、スキーマ scott 内で emp というオブジェクトは検出しません。
6. リモート・データベースは、emp というパブリック・シノニムを検索して見つけます。
7. リモート・データベースは文を実行し、その結果をローカル・データベースに返します。

ビュー、シノニムおよびプロシージャでのグローバル名前解決

ビュー、シノニムまたは PL/SQL プログラム・ユニット（プロシージャ、ファンクション、トリガーなど）は、グローバル・オブジェクト名によってリモートのスキーマ・オブジェクトを参照できます。グローバル・オブジェクト名が完全な場合、データベースは、グローバル・オブジェクト名を拡張せずにオブジェクトの定義を格納します。ただし、名前が部分的な場合、データベースは、ローカル・データベース名のドメインを使用してその名前を拡張します。

次の表は、ビュー、シノニムおよびプログラム・ユニットの部分的なグローバル・オブジェクト名を、データベースがいつ拡張するのかを示したものです。

ユーザー操作	データベースの応答
ビューの作成	部分的なグローバル名は拡張されません。定義内の問合せのテキストがそのままデータ・ディクショナリに格納されます。そのかわりに、ビューを使用する文が解析されるたびに、データベースは、部分的なグローバル・オブジェクト名を拡張します。
シノニムの作成	部分的なグローバル名が拡張されます。データ・ディクショナリに格納されるシノニムの定義には、拡張されたグローバル・オブジェクト名が含まれます。
プログラム・ユニットのコンパイル	部分的なグローバル名が拡張されます。

グローバル名を変更したときに起こる動作

グローバル名の変更は、部分的なグローバル・オブジェクト名を使用してリモート・データを参照するビュー、シノニムおよびプロシージャに影響を与える可能性があります。参照先データベースのグローバル名を変更すると、ビューおよびプロシージャは存在しないデータベースまたは正しくないデータベースを参照しようとすることがあります。一方、シノニムは実行時にデータベース・リンク名を拡張しないので、何も変わりません。

グローバル名の変更例

たとえば、sales.uk.acme.com および hq.uk.acme.com という 2 つのデータベースを考えます。また、sales データベースに次のビューとシノニムがあるとします。

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

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

データベースは、employee シノニム定義を拡張して、次のように保存します。

```
scott.emp@hr.uk.acme.com
```

使用例 1: 両方のデータベース名が変更された場合 最初に、営業および人事の両部門が米国に配置替えされるという状況を考えます。その結果、対応するグローバル・データベース名はどちらも次のように変更されます。

- sales.uk.acme.com が sales.us.acme.com になります。
- hq.uk.acme.com が hq.us.acme.com になります。

次の表は、グローバル名の変更前および変更後の問合せの拡張を示しています。

sales への問合せ	変更前の拡張	変更後の拡張
SELECT * FROM employee_names	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.us.acme.com
SELECT * FROM employee	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.uk.acme.com

使用例 2: 一方のデータベース名が変更された場合 この例では、営業部のみが米国に移動し、人事部は英国に留まるとします。その結果、対応するグローバル・データベース名はどちらも次のように変更されます。

- sales.uk.acme.com が sales.us.acme.com になります。
- hq.uk.acme.com は変更されません。

次の表は、グローバル名の変更前および変更後の問合せの拡張を示しています。

sales への問合せ	変更前の拡張	変更後の拡張
SELECT * FROM employee_names	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.us.acme.com
SELECT * FROM employee	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.uk.acme.com

この場合、employee_names ビューを定義している問合せが、存在しないグローバル・データベース名に拡張されます。一方、employee シノニムは、引き続き正しいデータベースである hq.uk.acme.com を参照します。

分散データベース・アプリケーションの開発

分散システムにおけるアプリケーションの開発では、非分散システムには当てはまらない問題が起こります。ここでは、分散アプリケーションの開発について説明します。この項の内容は、次のとおりです。

- 分散データベース・システムにおける透過性
- リモート・プロシージャ・コール（RPC）
- 分散問合せの最適化

> **関連項目：** 分散システム向けアプリケーションの開発方法の詳細は、第31章「分散データベース・システムのアプリケーション開発」を参照してください。

分散データベース・システムにおける透過性

システムを使用するユーザーに対して Oracle Database の分散データベース・システムを透過的にするアプリケーションを最小限の作業で開発できます。透過性の目的は、分散データベース・システムを単一の Oracle Database であるかのように処理することです。その結果、開発者およびシステムのユーザーは、分散データベースの複雑さから解放されます。透過性が備わっていなければ、この複雑さが原因で分散データベース・アプリケーションの開発は困難になり、ユーザーの生産性は大幅に低下することになります。

ここでは、分散データベース・システムにおける透過性についてさらに詳しく説明します。

位置の透過性

Oracle Database の分散データベース・システムには、アプリケーション開発者および管理者がデータベース・オブジェクトの物理的な位置をアプリケーションおよびユーザーから隠す機能があります。ユーザーが、アプリケーションの接続先ノードに関係なく、表などのデータベース・オブジェクトを一様に参照できるのには、**位置の透過性**が関係しています。位置の透過性には、次のようないくつかの利点があります。

- データベースのユーザーはデータベース・オブジェクトの物理的な位置を知る必要がないため、リモート・データへのアクセスが簡単になります。
- 管理者は、エンド・ユーザーや既存のデータベース・アプリケーションに影響を与えることなく、データベース・オブジェクトを移動できます。

通常、管理者および開発者は、アプリケーション・スキーマ内の表およびサポート・オブジェクトに対する位置の透過性を確立するために、シノニムを使用します。たとえば、次の文は、データベース内に別のリモート・データベース内の表に対応するシノニムを作成します。

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

これにより、リモート表にアクセスする際に次のような問合せを使用する必要がなくなります。

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

アプリケーションでは、リモート表の位置を考慮せずに、単純な問合せを発行できます。

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

シノニムの他にも、ビューおよびストアド・プロシージャを使用して、分散データベース・システムで動作するアプリケーションに対する位置の透過性を確立できます。

SQL および COMMIT の透過性

Oracle Database の分散データベース・アーキテクチャでは、問合せ、更新およびトランザクションの透過性も提供されています。たとえば、SELECT、INSERT、UPDATE、DELETE などの標準の SQL 文は、非分散データベース環境の場合とまったく同じように動作します。また、アプリケーションは、標準の SQL 文である COMMIT、SAVEPOINT および ROLLBACK を使用して、トランザクションを管理します。分散トランザクションを制御するために複雑なプログラミングやその他の特別な操作を行う必要はありません。

- 1 つのトランザクション内の文からは任意の数のローカル表またはリモート表を参照できます。
- データベースでは、分散トランザクションに関係するノードがすべて同じ動作をすることが保証されており、それらのノードすべてがトランザクションをコミットまたはロールバックします。
- 分散トランザクションのコミット中にネットワークまたはシステムの障害が発生した場合、トランザクションは自動的かつ透過的、およびグローバルに解決されます。具体的には、ネットワークまたはシステムがリストアされると、すべてのノードがトランザクションをコミットまたはロールバックします。

データベースの内部では、コミットされた各トランザクションに対して、そのトランザクション内の文による変更を一意に識別するための**システム変更番号**（**SCN**）が対応付けられます。分散データベースでは、次のときに通信中のノードの SCN が調整されます。

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

特に、分散データベース・システムのノード間で SCN が調整されることにより、文とトランザクションの両方のレベルでグローバルな分散読込み一貫性が保証されることは大きな利点です。必要であれば、グローバルな時間ベースの分散リカバリを完了することもできます。

レプリケーションの透過性

データベースはまた、システムのノード間でデータを透過的にレプリケートするための機能も数多く備えています。Oracle Database のレプリケーション機能の詳細は、『Oracle Database アドバンスト・レプリケーション』を参照してください。

リモート・プロシージャ・コール（RPC）

開発者は、分散データベースで動作するアプリケーションをサポートするための PL/SQL パッケージおよびプロシージャをコーディングできます。アプリケーションでは、ローカル・データベースでの処理を実行するためにローカル・プロシージャ・コールを実行でき、リモート・データベースでの処理を実行するために **RPC** を実行できます。

プログラムがリモート・プロシージャをコールすると、ローカル・サーバーは、コールされたリモート・サーバーにすべてのプロシージャ・パラメータを渡します。たとえば、次の PL/SQL プログラム・ユニットは、リモートの sales データベースにあるパッケージ化されたプロシージャ del_emp をコールし、それにパラメータ 1257 を渡します。

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

RPC を正常に実行するためには、コール側プロシージャがリモート・サイトに存在し、接続しようとするユーザーがそのプロシージャを実行するための適切な権限を持っている必要があります。

分散データベース・システム対応のパッケージおよびプロシージャを開発する際、開発者は、どのプログラム・ユニットをリモートの位置で実行し、その結果をどのようにコール側アプリケーションに返すのかについて理解した上で、コードを記述する必要があります。

分散問合せの最適化

Oracle Database の機能の 1 つである**分散問合せの最適化**は、トランザクションの分散 SQL 文内で参照されているリモート表からデータを取得するときに、サイト間で必要となるデータ転送の量を減らします。

分散問合せの最適化では、コストベースの最適化を使用して、リモート表から必要なデータのみを抽出する SQL 式が検索または生成されます。抽出されたデータはリモート・サイト（場合によってはローカル・サイト）で処理され、その結果がローカル・サイトに送信されて、最終処理が行われます。このような操作により、表データのすべてをローカル・サイトに転送して処理する場合と比較して、必要なデータ転送の量が低減します。

DRIVING_SITE、NO_MERGE、INDEX など、各種のコストベース・オプティマイザ・ヒントを使用することによって、Oracle Database がデータを処理する場所と、データへのアクセス方法を制御できます。

> **関連項目：** コストベースの最適化の詳細は、31-4 ページの「コストベース最適化の使用」を参照してください。

分散環境でのキャラクタ・セットのサポート

Oracle Database は、クライアント、Oracle Database サーバーおよび Oracle Database 以外のサーバーが異なるキャラクタ・セットを使用する環境をサポートしています。異機種環境のために NCHAR のサポートが提供されています。各国語サポート（NLS）および異機種間サービス（HS）に関連する各種の環境変数および初期化パラメータを設定することにより、異なるキャラクタ・セット間でのデータ変換を制御できます。

キャラクタの設定は、次の NLS および HS パラメータで定義されます。

パラメータ	環境	定義の対象
NLS_LANG（環境変数）	クライアント / サーバー	クライアント
NLS_LANGUAGE NLS_CHARACTERSET NLS_TERRITORY	クライアント / サーバー 非異機種間分散 異機種間分散	Oracle Database サーバー
HS_LANGUAGE	異機種間分散	Oracle Database 以外のサーバー Transparent Gateway
NLS_NCHAR（環境変数） HS_NLS_NCHAR	異機種間分散	Oracle Database サーバー Transparent Gateway

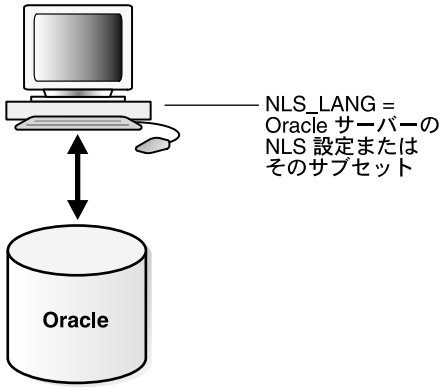
関連項目：

- NLS パラメータの詳細は、『Oracle Database グローバリゼーション・サポート・ガイド』を参照してください。
- HS パラメータの詳細は、『Oracle Database Heterogeneous Connectivity 管理者ガイド』を参照してください。

クライアント / サーバー環境

クライアント / サーバー環境では、図 29-6 のように、クライアント・キャラクタ・セットを Oracle Database サーバーのキャラクタ・セットと同じにするか、またはそのサブセットに設定します。

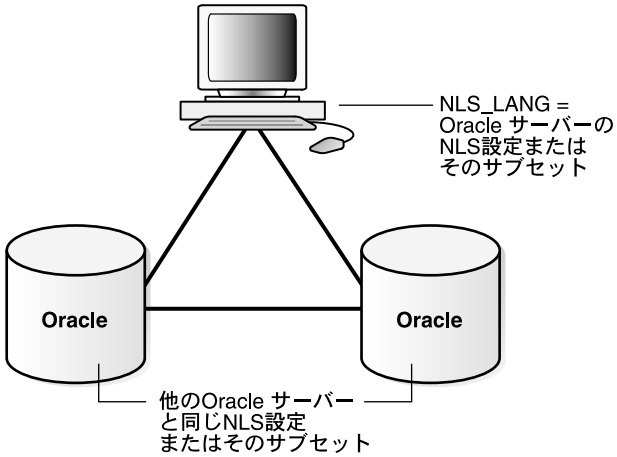
図 29-6 クライアント / サーバー環境における NLS パラメータの設定



同機種間分散環境

同機種環境では、図 29-7 のように、クライアントとサーバーのキャラクタ・セットをメイン・サーバーのキャラクタ・セットと同じにするか、またはそのサブセットにします。

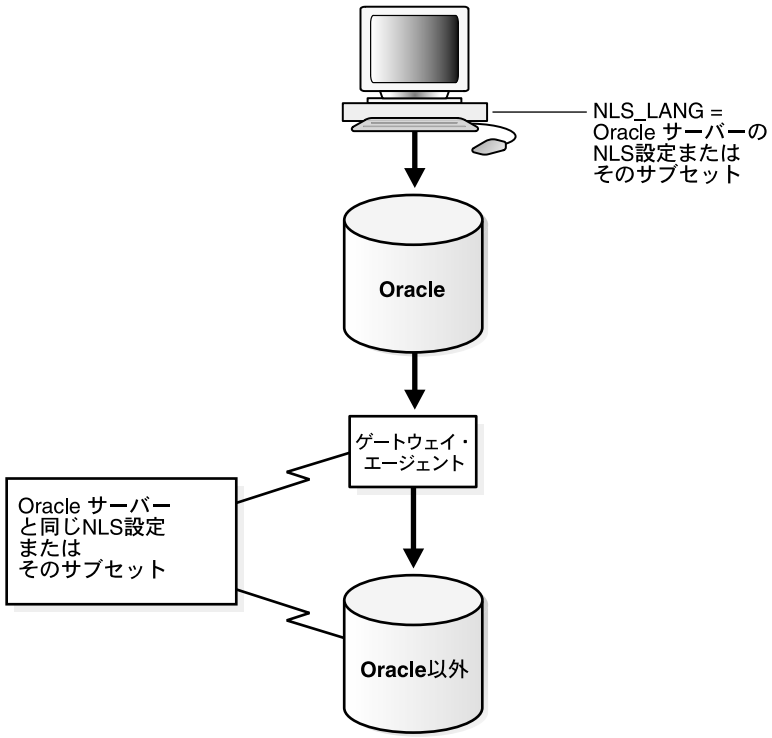
図 29-7 同機種環境における NLS パラメータの設定



異機種間分散環境

異機種環境では、図 29-8 のように、クライアント、Transparent Gateway および Oracle Database 以外のデータソースの、グローバリゼーション・サポートのパラメータ設定をすべて同じにするか、またはデータベース・サーバーのキャラクタ・セットのサブセットにします。Transparent Gateway は、グローバリゼーションを完全にサポートしています。

図 29-8 異機種環境における NLS パラメータの設定



異機種環境では、異機種間サービス・テクノロジによって構築された Transparent Gateway のみが完全な NCHAR 機能をサポートします。特定の Transparent Gateway が NCHAR をサポートしているかどうかは、対象となる Oracle Database 以外のデータソースによって異なります。特定の Transparent Gateway による NCHAR サポートの処理方法の詳細は、システム固有の Transparent Gateway のマニュアルを参照してください。

関連項目： 異機種間サービスの詳細は、『Oracle Database Heterogeneous Connectivity 管理者ガイド』を参照してください。

30

分散データベースの管理

この章の内容は次のとおりです。

分散システムでのグローバル名の管理

分散データベース・システムでは、各データベースが一意の**グローバル・データベース名**を持ちます。グローバル・データベース名は、システム内のデータベースを一意に識別します。分散データベースでの主な管理作業として、グローバル・データベース名の作成と変更の管理があります。

この項の内容は、次のとおりです。

グローバル・データベース名の書式の理解

グローバル・データベース名は、データベース名とドメインの 2 つの構成要素から構成されます。データベース名とドメイン名は、データベースの作成時に次の初期化パラメータによって決まります。

構成要素	パラメータ	要件	例
データベース名	DB_NAME	文字数は必ず 8 文字以内です。	sales
データベースが存在するドメイン	DB_DOMAIN	必ず標準のインターネット表記規則に従います。ドメイン名の各レベルは必ずドットで区切ります。ドメイン名の順序はリーフ（葉）からルート（根）、左から右です。	us.acme.com

次に、有効なグローバル・データベース名の例を示します。

DB_NAME	DB_DOMAIN	グローバル・データベース名
sales	au.oracle.com	sales.au.oracle.com
sales	us.oracle.com	sales.us.oracle.com
mktg	us.oracle.com	mktg.us.oracle.com
payroll	nonprofit.org	payroll.nonprofit.org

DB_DOMAIN 初期化パラメータはデータベースの作成時にのみ重要であり、DB_NAME パラメータとともに使用してデータベースのグローバル名を構成します。データベースのグローバル名は、この時点でデータ・ディクショナリに格納されます。グローバル名を変更する際は、初期化パラメータ・ファイルの DB_DOMAIN パラメータを変更するのではなく、ALTER DATABASE 文を使用する必要があります。しかし、ドメインの変更が反映されるように、次回データベースを起動する前に DB_DOMAIN パラメータを変更しておくことには意味があります。

グローバル・ネーミング施行の判断

ローカル・データベースのリンクに付ける名前は、アクセスするリモート・データベースがグローバル・ネーミングを施行するかどうかによって異なります。リモート・データベースがグローバル・ネーミングを施行する場合は、そのリモート・データベースのグローバル・データベース名をリンクの名前として使用する必要があります。たとえば、ローカルの hq サーバーに接続していて、リモートの mfg データベースへのリンクを作成する場合は、mfg がグローバル・ネーミングを施行していれば、その mfg のグローバル・データベース名をリンク名として使用する必要があります。

データベース・リンク名の一部としてサービス名を使用することもできます。たとえば、サービス名 sn1 と sn2 を使用してデータベース hq.acme.com に接続している場合、hq がグローバル・ネーミングを施行していれば、hq へのリンク名を次のように作成できます。

- HQ.ACME.COM@SN1
- HQ.ACME.COM@SN2

> **関連項目：** リンク名でのサービス名の使用の詳細は、30-11 ページの「リンク名に含まれるサービス名を指定するための接続修飾子の使用」を参照してください。

グローバル・ネーミングがデータベースで施行されているかどうかを判断するには、データベースの初期化パラメータ・ファイルを調べるか、または V$PARAMETER ビューを問い合せます。たとえば、mfg でグローバル・ネーミングが施行されているかどうかを判断するには、mfg のセッションを開始して、次の globalnames.sql スクリプトを作成し、実行できます（出力例も含まれています）。

```
COL NAME FORMAT A12
COL VALUE FORMAT A6
SELECT NAME, VALUE FROM V$PARAMETER
   WHERE NAME = 'global_names'
/

SQL> @globalnames

NAME         VALUE
------------ ------
global_names FALSE
```

グローバル・データベース名の参照

データベースのグローバル名を参照するには、データ・ディクショナリ・ビュー GLOBAL_NAME を使用します。たとえば、次の文を発行します。

```
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
--------------------------------------------------------------------------------
SALES.AU.ORACLE.COM
```

グローバル・データベース名のドメインの変更

データベースのグローバル名のドメインを変更するには、ALTER DATABASE 文を使用します。データベースを作成した後に初期化パラメータ DB_DOMAIN を変更しても、グローバル・データベース名やデータベース・リンク名の解決には効果がありません。

次の例は、名前変更文の構文を示しています。ここで、*database* はデータベース名、*domain* はネットワーク・ドメインを表します。

```
ALTER DATABASE RENAME GLOBAL_NAME TO database.domain;
```

グローバル・データベース名のドメインを変更するには、次の手順を実行します。

1. 現行のグローバル・データベース名を確認します。たとえば、次の文を発行します。

```
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
--------------------------------------------------------------------------------
SALES.AU.ORACLE.COM
```

2. ALTER DATABASE 文を使用して、グローバル・データベース名を変更します。たとえば、次のように入力します。

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.us.oracle.com;
```

3. GLOBAL_NAME 表を問い合せて、新しい名前を確認します。たとえば、次のように入力します。

```
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
--------------------------------------------------------------------------------
SALES.US.ORACLE.COM
```

グローバル・データベース名の変更 : 使用例

この例では、ローカル・データベースのグローバル・データベース名のドメイン部分を変更します。また、部分指定のグローバル名を使用してデータベース・リンクを作成し、Oracle Database がその名前をどのように解決するのかを調べます。データベースは、初期化パラメータ DB_DOMAIN の値ではなく、ローカル・データベースの現行のグローバル・データベース名のドメイン部分を使用して部分名を解決することがわかります。

1. SALES.US.ACME.COM に接続して GLOBAL_NAME データ・ディクショナリ・ビューを問い合せ、現行のデータベース・グローバル名を確認します。

```
CONNECT SYSTEM@sales.us.acme.com
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
--------------------------------------------------------------------------------
SALES.US.ACME.COM
```

2. V$PARAMETER ビューを問い合せ、DB_DOMAIN 初期化パラメータの現在の設定を調べます。

```
SELECT NAME, VALUE FROM V$PARAMETER WHERE NAME = 'db_domain';

NAME       VALUE
---------  -----------
db_domain  US.ACME.COM
```

3. 部分指定のグローバル名のみを使用して、hq 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 hq USING 'sales';
```

データベースは、ローカル・データベースのグローバル・データベース名のドメイン部分を、リンクで指定されたデータベースの名前の後に追加して、このリンクのグローバル・データベース名を拡張します。

4. USER_DB_LINKS を問い合せて、データベースが部分指定のグローバル・データベース名を解決するために使用したドメイン名を確認します。

```
SELECT DB_LINK FROM USER_DB_LINKS;

DB_LINK
------------------
HQ.US.ACME.COM
```

この結果は、ローカル・データベースのグローバル・データベース名のドメイン部分が us.acme.com であることを示しています。データベースは、データベース・リンク作成時の部分的なデータベース・リンク名を解決するために、このドメインを使用します。

5. sales データベースが日本に移動するという通知を受けたので、sales データベースを sales.jp.acme.com に変更します。

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.jp.acme.com;
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
--------------------------------------------------------------------------------
SALES.JP.ACME.COM
```

6. 再び V$PARAMETER を問い合せると、グローバル・データベース名のドメイン部分を変更しても、DB_DOMAIN の値は変更されていないことがわかります。

```
SELECT NAME, VALUE FROM V$PARAMETER
  WHERE NAME = 'db_domain';

NAME       VALUE
---------  -----------
db_domain  US.ACME.COM
```

この結果は、DB_DOMAIN 初期化パラメータの値が ALTER DATABASE RENAME GLOBAL_NAME 文とは関係がないことを示しています。ALTER DATABASE 文は、DB_DOMAIN 初期化パラメータではなく、グローバル・データベース名のドメインを決定します（それでも、新しいドメイン名が反映されるように DB_DOMAIN を変更することには意味があります）。

7. データベース supply への別のデータベース・リンクを作成してから、USER_DB_LINKS を問い合せて、データベースが supply のグローバル・データベース名のドメイン部分をどのように解決するのかを確認します。

```
CREATE DATABASE LINK supply USING 'supply';
SELECT DB_LINK FROM USER_DB_LINKS;

DB_LINK
------------------
HQ.US.ACME.COM
SUPPLY.JP.ACME.COM
```

この結果は、データベースがドメイン jp.acme.com. を使用して、部分指定のリンク名を解決したことを示しています。このドメインは、リンクの作成時に使用されます。これは、このドメインがローカル・データベースのグローバル・データベース名のドメイン部分であるためです。データベースは、部分的なリンク名を解決するときに、DB_DOMAIN 初期化パラメータの設定を使用しません。

8. 次に、前の情報が誤りで、sales は jp.acme.com ドメインではなく asia.jp.acme.com ドメインにあるという通知を受けました。そのため、グローバル・データベース名を次のように変更します。

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.asia.jp.acme.com;
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
--------------------------------------------------------------------------------
SALES.ASIA.JP.ACME.COM
```

9. 再び V$PARAMETER を問い合せて、パラメータ DB_DOMAIN の設定を確認します。

```
SELECT NAME, VALUE FROM V$PARAMETER
  WHERE NAME = 'db_domain';

NAME        VALUE
----------  -----------
db_domain   US.ACME.COM
```

この結果は、パラメータ・ファイルのドメイン設定が、2 つの ALTER DATABASE RENAME 文を発行する前と変わっていないことを示しています。

10. 最後に、warehouse データベースへのリンクを作成し、再度 USER_DB_LINKS を問い合せて、データベースが部分指定のグローバル名をどのように解決するのかを調べます。

```
CREATE DATABASE LINK warehouse USING 'warehouse';
SELECT DB_LINK FROM USER_DB_LINKS;

DB_LINK
------------------
HQ.US.ACME.COM
SUPPLY.JP.ACME.COM
WAREHOUSE.ASIA.JP.ACME.COM
```

ここでもデータベースは、リンクの作成時にローカル・データベースのグローバル・データベース名のドメイン部分を使用して、部分的なリンク名を拡張していることがわかります。

注意： supply データベース・リンクを訂正するには、必ずこのデータベース・リンクを削除してから再作成してください。

関連項目： DB_NAME および DB_DOMAIN 初期化パラメータの指定の詳細は、『Oracle Database リファレンス』を参照してください。

データベース・リンクの作成

アプリケーションによるデータおよびスキーマ・オブジェクトへのアクセスを分散データベース・システム全体にわたってサポートするために、必要なデータベース・リンクをすべて作成する必要があります。この項の内容は、次のとおりです。

- データベース・リンクの作成に必要な権限の取得
- リンク・タイプの指定
- リンク・ユーザーの指定
- リンク名に含まれるサービス名を指定するための接続修飾子の使用

データベース・リンクの作成に必要な権限の取得

データベース・リンクは、リモート・データベースのオブジェクトへのアクセスを可能にするローカル・データベース内のポインタです。プライベート・データベース・リンクを作成するには、あらかじめ適切な権限が付与されている必要があります。次の表は、どのリンク・タイプのデータベースに対してどの権限が必要なのかを示しています。

権限	データベース	この権限を必要とする操作
CREATE DATABASE LINK	ローカル	プライベート・データベース・リンクの作成
CREATE PUBLIC DATABASE LINK	ローカル	パブリック・データベース・リンクの作成
CREATE SESSION	リモート	任意タイプのデータベース・リンクの作成

現在使用可能な権限を確認するには、ROLE_SYS_PRIVS を問い合せます。たとえば、次の privs.sql スクリプトを作成し、実行できます（出力例も含まれています）。

```
SELECT DISTINCT PRIVILEGE AS "Database Link Privileges"
FROM ROLE_SYS_PRIVS
WHERE PRIVILEGE IN ( 'CREATE SESSION','CREATE DATABASE LINK',
                     'CREATE PUBLIC DATABASE LINK')
/

SQL> @privs

Database Link Privileges
----------------------------------------
CREATE DATABASE LINK
CREATE PUBLIC DATABASE LINK
CREATE SESSION
```

リンク・タイプの指定

データベース・リンクを作成するときは、そのデータベース・リンクにアクセスするユーザーを決める必要があります。ここでは、3 種類の基本的なリンク・タイプの作成方法について説明します。

- プライベート・データベース・リンクの作成
- パブリック・データベース・リンクの作成
- グローバル・データベース・リンクの作成

プライベート・データベース・リンクの作成

プライベート・データベース・リンクを作成するには、次の文を指定します。ここで、*link_name* はグローバル・データベース名または任意のリンク名を表します。

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

プライベート・データベース・リンクの例を次に示します。

SQL 文	結果
`CREATE DATABASE LINK supply.us.acme.com;`	リモートの `supply` データベースへの、グローバル・データベース名を使用したプライベート・リンク。 リンクは、接続ユーザーのユーザー ID/ パスワードを使用します。そのため、`scott`（パスワード：`tiger`）が問合せの中でこのリンクを使用すると、リモート・データベースへの接続が `scott/tiger` として確立されます。
`CREATE DATABASE LINK link_2 CONNECT TO jane IDENTIFIED BY doe USING 'us_supply';`	サービス名 us_supply を持つデータベースへの、link_2 という名前のプライベート固定ユーザー・リンク。このリンクは、接続ユーザーとは無関係に、jane/doe というユーザー ID/ パスワードでリモート・データベースに接続します。
`CREATE DATABASE LINK link_1 CONNECT TO CURRENT_USER USING 'us_supply';`	サービス名 `us_supply` を持つデータベースへの、`link_1` という名前のプライベート・リンク。このリンクは、現行ユーザーのユーザー ID/ パスワードを使用してリモート・データベースにログインします。 **注意：**現行ユーザーと接続ユーザーは異なる場合があります。また、現行ユーザーは、リンクに関係している両方のデータベースのグローバル・ユーザーであることが必要です（29-13 ページの「データベース・リンクのユーザー」を参照）。現行ユーザー・リンクは、Oracle Advanced Security オプションの一部です。

関連項目： `CREATE DATABASE LINK` の構文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

パブリック・データベース・リンクの作成

パブリック・データベース・リンクを作成するには、キーワード PUBLIC を使用します。ここで、*link_name* はグローバル・データベース名または任意のリンク名を表します。

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

パブリック・データベース・リンクの例を次に示します。

SQL 文	結果
`CREATE PUBLIC DATABASE LINK supply.us.acme.com;`	リモートの supply データベースへのパブリック・リンク。リンクは、接続ユーザーのユーザー ID/ パスワードを使用します。そのため、scott（パスワード：tiger）が問合せの中でこのリンクを使用すると、リモート・データベースへの接続が scott/tiger として確立されます。
`CREATE PUBLIC DATABASE LINK pu_link CONNECT TO CURRENT_USER USING 'supply';`	サービス名 supply を持つデータベースへの、pu_link という名前のパブリック・リンク。このリンクは、現行ユーザーのユーザー ID/ パスワードを使用してリモート・データベースにログインします。 **注意：**現行ユーザーと接続ユーザーは異なる場合があります。また、現行ユーザーは、リンクに関係している両方のデータベースのグローバル・ユーザーであることが必要です（29-13 ページの「データベース・リンクのユーザー」を参照）。
`CREATE PUBLIC DATABASE LINK sales.us.acme.com CONNECT TO jane IDENTIFIED BY doe;`	リモートの sales データベースへのパブリック固定ユーザー・リンク。このリンクは、jane/doe というユーザー ID/ パスワードでリモート・データベースに接続します。

関連項目： CREATE PUBLIC DATABASE LINK の構文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

グローバル・データベース・リンクの作成

以前のリリースでは、**グローバル・データベース・リンク**は Oracle Names Server で定義していました。Oracle Names Server は非推奨になりました。現在はディレクトリ・サーバーが使用できるようになり、データベースはネット・サービス名によって識別されるようになりました。このマニュアルでは、ディレクトリ・サーバーでのネット・サービス名によるデータベースの識別をグローバル・データベース・リンクと呼びます。

グローバル・データベース・リンクとして動作するディレクトリ・エントリの作成方法の詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

リンク・ユーザーの指定

データベース・リンクは、あるデータベースから別のデータベースへの通信経路を定義します。アプリケーションがデータベース・リンクを使用してリモート・データベースに接続するとき、Oracle Database はローカル・アプリケーションの要求にかわってリモート・データベース内でデータベース・セッションを確立します。

プライベートまたはパブリックのデータベース・リンクを作成する際、固定ユーザー、現行ユーザーおよび接続ユーザーのデータベース・リンクを作成することで、リモート・データベースのどのスキーマにリンクが接続を確立するのかを指定できます。

固定ユーザー・データベース・リンクの作成

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

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

固定ユーザー・データベース・リンクの例を次に示します。

SQL 文	結果
`CREATE PUBLIC DATABASE LINK supply.us.acme.com CONNECT TO scott AS tiger;`	リモートの supply データベースへの、グローバル・データベース名を使用したパブリック・リンク。このリンクは、scott/tiger というユーザー ID/ パスワードでリモート・データベースに接続します。
`CREATE DATABASE LINK foo CONNECT TO jane IDENTIFIED BY doe USING 'finance';`	サービス名 finance を持つデータベースへの、foo という名前のプライベート固定ユーザー・リンク。このリンクは、jane/doe というユーザー ID/ パスワードでリモート・データベースに接続します。

アプリケーションで固定ユーザー・データベース・リンクを使用するとき、ローカル・サーバーは必ずリモート・データベース内の固定リモート・スキーマへの接続を確立します。また、ローカル・サーバーはアプリケーションがリンクを使用してリモート・データベースにアクセスするときに、ネットワークを介して固定ユーザーの資格証明を送信します。

接続ユーザーおよび現行ユーザー・データベース・リンクの作成

接続ユーザーおよび現行ユーザー・データベース・リンクでは、リンク定義に資格証明が含まれません。リモート・データベースへの接続に使用する資格証明は、データベース・リンクを参照するユーザーや、アプリケーションが実行する操作によって異なります。

注意： 多くの分散アプリケーションでは、ユーザーがリモート・データベースでの権限を持つ必要はありません。これを実現する簡単な方法の 1 つは、固定ユーザーまたは現行ユーザーのデータベース・リンクを含むプロシージャを作成することです。この方法では、プロシージャにアクセスするユーザーに対して第三者の権限が一時的に付与されます。

接続ユーザーと現行ユーザーの区別に関する概念の詳細は、29-13 ページの「データベース・リンクのユーザー」を参照してください。

接続ユーザー・データベース・リンクの作成 接続ユーザー・データベース・リンクを作成するには、CONNECT TO 句を省略します。次の構文は、接続ユーザー・データベース・リンクを作成します。ここで、*dblink* はリンクの名前、*net_service_name* はオプションの接続文字列を表します。

```
CREATE [SHARED] [PUBLIC] DATABASE LINK dblink ... [USING 'net_service_name'];
```

たとえば、接続ユーザー・データベース・リンクを作成するために、次の構文を使用します。

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

現行ユーザー・データベース・リンクの作成 現行ユーザー・データベース・リンクを作成するには、リンク作成文で CONNECT TO CURRENT_USER 句を使用します。現行ユーザー・リンクは、Oracle Advanced Security オプションでのみ使用できます。

次の構文は、現行ユーザー・データベース・リンクを作成します。ここで、*dblink* はリンクの名前、*net_service_name* はオプションの接続文字列を表します。

```
CREATE [SHARED] [PUBLIC] DATABASE LINK dblink CONNECT TO CURRENT_USER
[USING 'net_service_name'];
```

たとえば、sales データベースへの現行ユーザー・データベース・リンクを作成するには、次の構文を使用します。

```
CREATE DATABASE LINK sales CONNECT TO CURRENT_USER USING 'sales';
```

注意： 現行ユーザー・データベース・リンクを使用するには、リンクに関係している両方のデータベースで現行ユーザーがグローバル・ユーザーであることが必要です。

関連項目： データベース・リンクの作成に関する構文情報の詳細は、『Oracle Database SQL リファレンス』を参照してください。

リンク名に含まれるサービス名を指定するための接続修飾子の使用

場合によっては、同じリモート・データベースを指すものの、異なる通信経路を使用してリモート・データベースへの接続を確立する同じタイプ（パブリックなど）のデータベース・リンクを複数作成しなければならないことがあります。次のような場合に、この方法が役立ちます。

- リモート・データベースが Oracle Real Application Clusters 構成の一部であり、そのためリモート・データベースの特定のインスタンスに接続を確立できるようにローカル・ノードで複数のパブリック・データベース・リンクを定義する場合
- Oracle Database サーバーへの接続に、TCP/IP を使用するクライアントと、DECNET を使用するクライアントがある場合

このような機能を容易に利用できるように、データベースでは、データベース・リンク名の中でオプションのサービス名を使用してデータベース・リンクを作成できます。データベース・リンクを作成するときは、サービス名をデータベース・リンク名の末尾部分に指定し、アットマーク（@）で区切ります。たとえば、@sales のようにします。この文字列のことを**接続修飾子**と呼びます。

たとえば、リモート・データベース hq.acme.com が Oracle Real Application Clusters 環境で管理されているとします。hq データベースには、hq_1 および hq_2 という名前の 2 つのインスタンスがあります。ローカル・データベースで次のパブリック・データベース・リンクを作成して、hq データベースのリモート・インスタンスへの経路を定義できます。

```
CREATE PUBLIC DATABASE LINK hq.acme.com@hq_1
  USING 'string_to_hq_1';
CREATE PUBLIC DATABASE LINK hq.acme.com@hq_2
  USING 'string_to_hq_2';
CREATE PUBLIC DATABASE LINK hq.acme.com
  USING 'string_to_hq';
```

最初の 2 つの例では、サービス名が単にデータベース・リンク名の一部であることに注意してください。サービス名のテキストは、必ずしも接続の確立方法を示す必要はありません。この情報は、USING 句のサービス名で指定します。また、3 番目の例ではサービス名がリンク名の一部として指定されていません。この場合は、サービス名をリンク名の一部として指定したときと同様に、USING 文字列によってインスタンスが決まります。

サービス名を使用して特定のインスタンスを指定するには、サービス名をグローバル・オブジェクト名の末尾に付けます。

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

この例では、2 つのアットマーク（@）があることに注意してください。

共有データベース・リンクの使用

標準のデータベース・リンクを使用してリモート・サーバーを参照するアプリケーションはすべて、ローカル・データベースとリモート・データベースの間で接続を確立します。多くのユーザーがアプリケーションを同時に実行した場合、ローカル・データベースとリモート・データベースの間で多数の接続が発生する可能性があります。

共有データベース・リンクを使用すると、ローカル・サーバーとリモート・サーバーの間で必要なネットワーク接続の数を制限できます。

この項の内容は、次のとおりです。

- 共有データベース・リンクの使用の判断
- 共有データベース・リンクの作成
- 共有データベース・リンクの構成

> **関連項目：** 共有データベース・リンクの概要の詳細は、29-9 ページの「共有データベース・リンクの概要」を参照してください。

共有データベース・リンクの使用の判断

アプリケーションおよび共有サーバーの構成を綿密に検討して、共有リンクを使用するかどうかを判断してください。簡単なガイドラインとして、データベース・リンクにアクセスするユーザー数がローカル・データベース内のサーバー・プロセス数よりもかなり多いと予測されるときは、共有データベース・リンクを使用します。

データベース・リンクに関して可能な 3 つの構成を次の表に示します。

リンク・タイプ	サーバー・モード	結果
非共有	専用 / 共有サーバー	アプリケーションで標準のパブリック・データベース・リンクを使用していて、100 人のユーザーが同時に接続を要求した場合は、リモート・データベースへの直接ネットワーク接続が 100 必要になります。
共有	共有サーバー	ローカルの共有サーバー・モード・データベースに 10 の共有サーバー・プロセスが存在している場合、同じデータベース・リンクを使用するユーザーが 100 人いるときに必要となるリモート・サーバーへのネットワーク接続数は 10 以下になります。ローカルの各共有サーバー・プロセスが必要とするリモート・サーバーへの接続は、それぞれ 1 つで済む場合があります。
共有	専用	10 のクライアントがローカルの専用サーバーに接続していて、各クライアントが同じ接続のセッションを 10 持っており（したがって全部で 100 のセッションを確立している）、各セッションで同じリモート・データベースを参照している場合、必要な接続は 10 で済みます。非共有データベース・リンクでは、100 の接続が必要になります。

共有データベース・リンクは、必ずしもすべての状況で役立つとはかぎりません。たとえば、リモート・サーバーに接続するユーザーが 1 人のみの場合を考えます。このユーザーが共有データベース・リンクを定義し、ローカル・データベースに 10 個の共有サーバー・プロセスが存在する場合、このユーザーはリモート・サーバーへのネットワーク接続を最大 10 個必要とす

る可能性があります。ユーザーは個々の共有サーバー・プロセスを使用できるため、各プロセスはそれぞれリモート・サーバーへの接続を確立できます。

非共有のデータベース・リンクでは1つのネットワーク接続しか必要としないため、このような状況では明らかに非共有のデータベース・リンクのほうが望ましいといえます。共有データベース・リンクでは、シングル・ユーザーの場合に多くのネットワーク接続が発生します。したがって、共有リンクは、多数のユーザーが同じリンクを使用する必要があるときのみ使用してください。通常、共有リンクはパブリック・データベース・リンクで使用しますが、多数のクライアントが同じローカル・スキーマ（したがって同じプライベート・データベース・リンク）にアクセスするときは、プライベート・データベース・リンクでも使用できます。

注意： 複数階層の環境では、共有データベース・リンクを使用してリモート・データベースに接続した場合、そのリモート・データベースは、移行不可能なデータベース・リンクを使用して別のデータベースにリンクすることができないという制約があります。このリンクには共有サーバーを使用するか、またはリンク自体が別の共有データベース・リンクである必要があります。

共有データベース・リンクの作成

共有データベース・リンクを作成するには、CREATE DATABASE LINK 文でキーワード SHARED を使用します。

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

キーワード SHARED を使用するときは、必ず AUTHENTICATED BY 句が必要です。AUTHENTICATED BY で指定されたスキーマがリモート・データベースに存在し、少なくとも、CREATE SESSION 権限が付与されている必要があります。このスキーマの資格証明は、ローカル・データベースとリモート・データベース間での認証方式と考えることができます。これらの資格証明は、データベース・リンクのユーザーになりすまして情報に不当にアクセスしようとするクライアントから、リモート共有サーバー・プロセスを保護するために必要です。

共有デ―タベース・リンクとの接続後、リモート・データベース上での操作は、AUTHENTICATED BY スキーマではなく、CONNECT TO ユーザー、または CURRENT_USER の権限で実行されます。

次の例では、scott として接続し、linkuser として認証される、sales データベースへの固定ユーザー共有リンクを作成しています。

```
CREATE SHARED DATABASE LINK link2sales
CONNECT TO scott IDENTIFIED BY tiger
AUTHENTICATED BY linkuser IDENTIFIED BY ostrich
USING 'sales';
```

関連項目： CREATE DATABASE LINK 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

共有データベース・リンクの構成

共有データベース・リンクは、次の方法で構成できます。

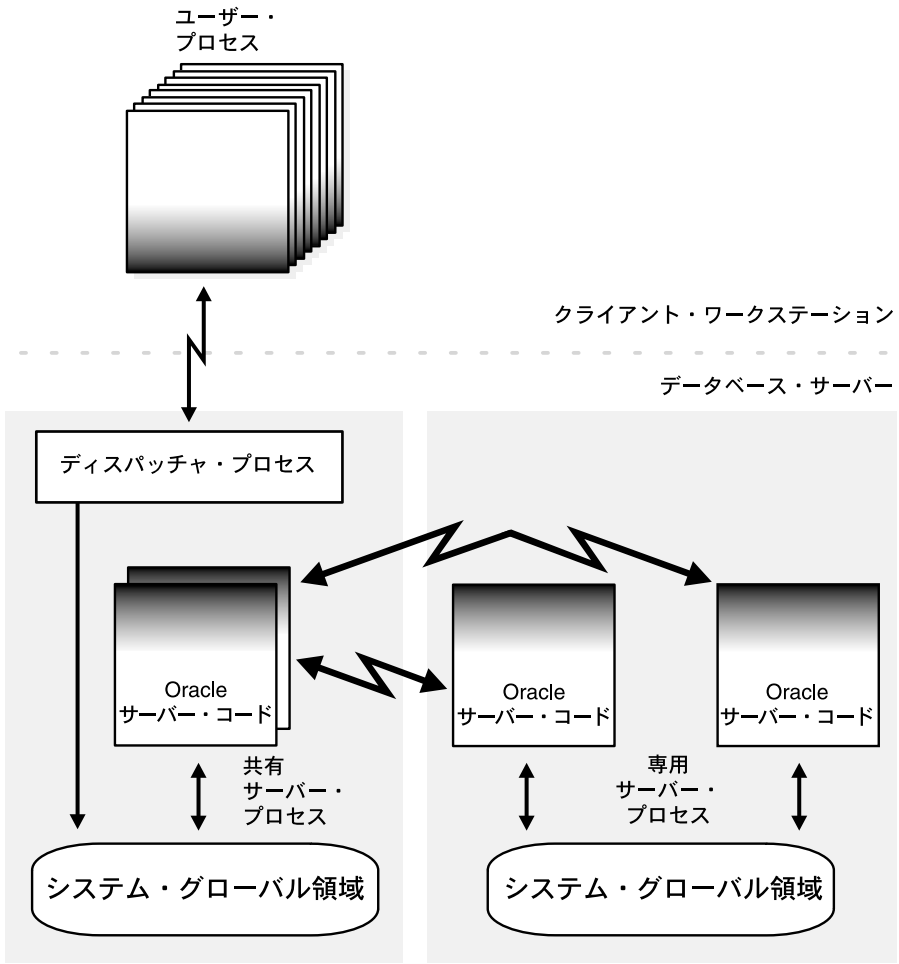
- 専用サーバーへの共有リンクの作成
- 共有サーバーへの共有リンクの作成

専用サーバーへの共有リンクの作成

図 30-1 に示す構成では、ローカル・サーバーの共有サーバー・プロセスは専用のリモート・サーバー・プロセスを所有しています。この構成の利点は、ローカルの共有サーバーとリモートの専用サーバーとの間に直接的なネットワーク・トランスポートが存在することです。ただし、余分なバックエンド・サーバー・プロセスが必要になるという欠点もあります。

注意： リモート・サーバーは、共有サーバーまたは専用サーバーのどちらか一方にできます。ローカル・サーバーとリモート・サーバーの間には専用の接続が存在します。リモート・サーバーが共有サーバーのときは、サービス名の定義に SERVER=DEDICATED 句を使用することで、専用サーバー接続を強制できます。

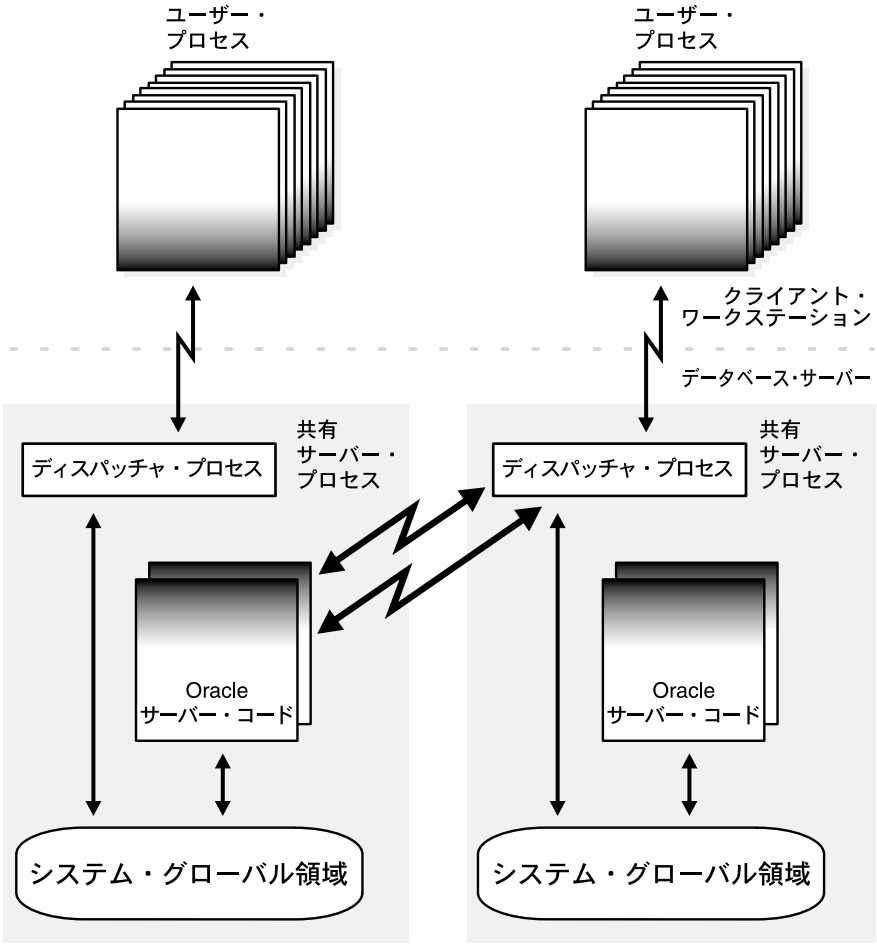
図 30-1 専用サーバー・プロセスへの共有データベース・リンク



共有サーバーへの共有リンクの作成

図 30-2 に示す構成では、リモート・サーバーで共有サーバー・プロセスを使用しています。この構成では、多数の専用サーバー・プロセスは必要ありませんが、リモート・サーバーのディスパッチャを経由する接続が必要になります。この場合、ローカル・サーバーとリモート・サーバーの両方を共有サーバーとして構成する必要があります。

図 30-2 共有サーバーへの共有データベース・リンク



関連項目： 共有サーバー・オプションの詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

データベース・リンクの管理

この項の内容は、次のとおりです。

データベース・リンクのクローズ

あるセッションでデータベース・リンクにアクセスする場合、そのセッションをクローズするまでリンクはオープンしたままです。リンクがオープンしているということは、そのリンクを介してアクセスしている各リモート・データベースのプロセスがアクティブであることを意味します。この状況では、次のような結果が生じます。

- 20 人のユーザーがセッションをオープンしてローカル・データベースの同じパブリック・リンクにアクセスする場合、20 のデータベース・リンク接続がオープンします。
- 20 人のユーザーがセッションをオープンして各ユーザーがプライベート・リンクにアクセスする場合、20 のデータベース・リンク接続がオープンします。
- 1 人のユーザーがセッションを開始して 20 の異なるリンクにアクセスする場合、20 のデータベース・リンク接続がオープンします。

セッションをクローズした後、セッションでアクティブだったリンクは自動的にクローズされます。ユーザーがリンクを手動でクローズする場合もあります。たとえば、次のようなときにリンクをクローズします。

- リンクによって確立されたネットワーク接続がアプリケーションでそれほど頻繁に使用されないとき
- ユーザー・セッションを終了する必要があるとき

リンクをクローズする場合は、次の文を発行します。ここで、*linkname* はリンクの名前を表します。

```
ALTER SESSION CLOSE DATABASE LINK linkname;
```

この文は、現行セッションでアクティブなリンクのみをクローズします。

データベース・リンクの削除

データベース・リンクは、表やビューと同様に削除できます。リンクがプライベートの場合は、自分のスキーマ内に存在する必要があります。リンクがパブリックの場合は、`DROP PUBLIC DATABASE LINK` システム権限が必要です。

構文は次のとおりです。ここで、*dblink* はリンクの名前を表します。

```
DROP [PUBLIC] DATABASE LINK dblink;
```

プライベート・データベース・リンクの削除手順

1. SQL*Plus を使用して、ローカル・データベースに接続します。たとえば、次のように入力します。

   ```
   CONNECT scott@local_db
   ```

2. USER_DB_LINKS を問い合せて、所有しているリンクを確認します。たとえば、次のように入力します。

```
SELECT DB_LINK FROM USER_DB_LINKS;

DB_LINK
---------------------------------
SALES.US.ORACLE.COM
MKTG.US.ORACLE.COM
2 rows selected.
```

3. DROP DATABASE LINK 文を使用して、目的のリンクを削除します。たとえば、次のように入力します。

```
DROP DATABASE LINK sales.us.oracle.com;
```

パブリック・データベース・リンクの削除手順

1. ローカル・データベースに DROP PUBLIC DATABASE LINK 権限を持つユーザーとして接続します。たとえば、次のように入力します。

```
CONNECT SYSTEM@local_db AS SYSDBA
```

2. DBA_DB_LINKS を問い合せて、パブリック・リンクを確認します。たとえば、次のように入力します。

```
SELECT DB_LINK FROM USER_DB_LINKS
  WHERE OWNER = 'PUBLIC';

DB_LINK
---------------------------------
DBL1.US.ORACLE.COM
SALES.US.ORACLE.COM
INST2.US.ORACLE.COM
RMAN2.US.ORACLE.COM
4 rows selected.
```

3. DROP PUBLIC DATABASE LINK 文を使用して、目的のリンクを削除します。たとえば、次のように入力します。

```
DROP PUBLIC DATABASE LINK sales.us.oracle.com;
```

アクティブ・データベース・リンクの接続数の制限

静的な初期化パラメータ OPEN_LINKS を使用すると、ユーザー・プロセスからリモート・データベースへの接続数を制限できます。このパラメータは、分散トランザクションにおいて 1 つのユーザー・セッションが同時に使用できるリモート接続数を制御します。

このパラメータを設定する際は、次の点を考慮してください。

- 設定値は、複数のデータベースを参照する 1 つの SQL 文によって参照されるデータベースの数以上にします。
- 複数の分散データベースに継続してアクセスする場合は、設定値を大きくします。たとえば、3 つのデータベースに定期的にアクセスする場合は、OPEN_LINKS を 3 以上に設定します。
- OPEN_LINKS のデフォルト値は 4 です。OPEN_LINKS を 0（ゼロ）に設定した場合、分散トランザクションは許可されません。

> **関連項目：** OPEN_LINKS 初期化パラメータの詳細は、『Oracle Database リファレンス』を参照してください。

データベース・リンク情報の表示

各データベースのデータ・ディクショナリには、そのデータベース内にあるデータベース・リンクの定義がすべて格納されています。データ・ディクショナリ表およびビューを使用して、リンクに関する情報を取得できます。この項の内容は、次のとおりです。

データベース内のリンクの判断

次のビューには、ローカル・データベースで定義され、データ・ディクショナリに格納されているデータベース・リンクが表示されます。

ビュー	用途
DBA_DB_LINKS	データベース内のデータベース・リンクがすべてリストされます。
ALL_DB_LINKS	接続ユーザーがアクセス可能なデータベース・リンクがすべてリストされます。
USER_DB_LINKS	接続ユーザーが所有しているデータベース・リンクがすべてリストされます。

これらのデータ・ディクショナリ・ビューには、データベース・リンクに関する同じ基本情報が含まれていますが、いくつか例外もあります。

列	対象となるビュー	説明
OWNER	USER_* を除くすべて	データベース・リンクを作成したユーザー。リンクがパブリックの場合、ユーザーは PUBLIC としてリストされます。
DB_LINK	すべて	データベース・リンクの名前
USERNAME	すべて	リンク定義に固定ユーザーが含まれている場合、この列には固定ユーザーのユーザー名が表示されます。固定ユーザーが含まれていない場合、この列は NULL になります。
PASSWORD	USER_* のみ	未使用です。下位互換性のためにのみ維持されています。
HOST	すべて	リモート・データベースへの接続に使用されるネット・サービス名。
CREATED	すべて	データベース・リンクの作成日時。

どのユーザーでも USER_DB_LINKS を問い合せることで、そのユーザーが使用できるデータベース・リンクを判断できます。ALL_DB_LINKS ビューまたは DBA_DB_LINKS ビューを使用できるのは、追加の権限を持つユーザーのみです。

次のスクリプトは、DBA_DB_LINKS ビューを問い合せて、リンク情報にアクセスします。

```
COL OWNER FORMAT a10
COL USERNAME FORMAT A8 HEADING "USER"
COL DB_LINK FORMAT A30
COL HOST FORMAT A7 HEADING "SERVICE"
SELECT * FROM DBA_DB_LINKS
/
```

ここでスクリプトが起動され、その出力結果が表示されます。

```
SQL>@link_script

OWNER      DB_LINK                         USER     SERVICE CREATED
---------- ------------------------------- -------- ------- ----------
SYS        TARGET.US.ACME.COM              SYS      inst1   23-JUN-99
PUBLIC     DBL1.UK.ACME.COM                BLAKE    ora51   23-JUN-99
PUBLIC     RMAN2.US.ACME.COM                        inst2   23-JUN-99
PUBLIC     DEPT.US.ACME.COM                         inst2   23-JUN-99
JANE       DBL.UK.ACME.COM                 BLAKE    ora51   23-JUN-99
SCOTT      EMP.US.ACME.COM                 SCOTT    inst2   23-JUN-99
6 rows selected.
```

オープンしているリンク接続の判断

自分のセッションで現在オープンしているデータベース・リンク接続がわかれば役に立つ場合があります。しかし、SYSDBA として接続している場合には、ビューを問い合せてすべてのセッションでオープンしているすべてのリンクを判断することはできず、現在作業中のセッションのリンク情報にアクセスすることしかできません。

次のビューには、現行セッションで現在オープンしているデータベース・リンク接続が表示されます。

ビュー	用途
V$DBLINK	自分のセッションでオープンしているすべてのデータベース・リンク、つまり、IN_TRANSACTION 列が YES に設定されているすべてのデータベース・リンクがリストされます。
GV$DBLINK	自分のセッションでオープンしているすべてのデータベース・リンクと対応するインスタンスがリストされます。このビューは、Oracle Real Application Clusters 構成の使用時に役立ちます。

これらのデータ・ディクショナリ・ビューには、データベース・リンクに関する同じ基本情報が含まれていますが、1 つ例外があります。

列	対象となるビュー	説明
DB_LINK	すべて	データベース・リンクの名前
OWNER_ID	すべて	データベース・リンクの所有者
LOGGED_ON	すべて	データベース・リンクが現在ログインされているかどうか
HETEROGENEOUS	すべて	データベース・リンクが同機種間（NO）と異機種間（YES）のどちらであるか
PROTOCOL	すべて	データベース・リンクの通信プロトコル
OPEN_CURSORS	すべて	データベース・リンクでカーソルがオープンされているかどうか
IN_TRANSACTION	すべて	コミットまたはロールバックがまだ完了していないトランザクションで、データベース・リンクがアクセスされているかどうか
UPDATE_SENT	すべて	データベース・リンクで更新があったかどうか
COMMIT_POINT_STRENGTH	すべて	データベース・リンクを使用しているトランザクションのコミット・ポイント強度
INST_ID	GV$DBLINK のみ	ビュー情報が取得されたインスタンス

たとえば、次のスクリプトを作成し、実行することで、オープンされているリンクを判断できます（出力例も含まれています）。

```
COL DB_LINK FORMAT A25
COL OWNER_ID FORMAT 99999 HEADING "OWNID"
COL LOGGED_ON FORMAT A5 HEADING "LOGON"
COL HETEROGENEOUS FORMAT A5 HEADING "HETER"
COL PROTOCOL FORMAT A8
COL OPEN_CURSORS FORMAT 999 HEADING "OPN_CUR"
COL IN_TRANSACTION FORMAT A3 HEADING "TXN"
COL UPDATE_SENT FORMAT A6 HEADING "UPDATE"
COL COMMIT_POINT_STRENGTH FORMAT 99999 HEADING "C_P_S"

SELECT * FROM V$DBLINK
/

SQL> @dblink

DB_LINK                   OWNID LOGON HETER PROTOCOL OPN_CUR TXN UPDATE  C_P_S
------------------------- ------ ----- ----- -------- ------- --- ------ ------
INST2.ACME.COM                 0 YES   YES    UNKN           0 YES YES       255
```

位置の透過性の作成

必要なデータベース・リンクの構成が完了した後、各種のツールを使用してデータベース・システムの分散的な性質をユーザーから隠すことができます。言い換えれば、ユーザーはリモート・オブジェクトに対してあたかもローカル・オブジェクトであるかのようにアクセスできるようになります。ここでは、分散機能をユーザーから隠す方法について説明します。

- ビューを使用した位置の透過性の作成
- シノニムを使用した位置の透過性の作成
- プロシージャを使用した位置の透過性の作成

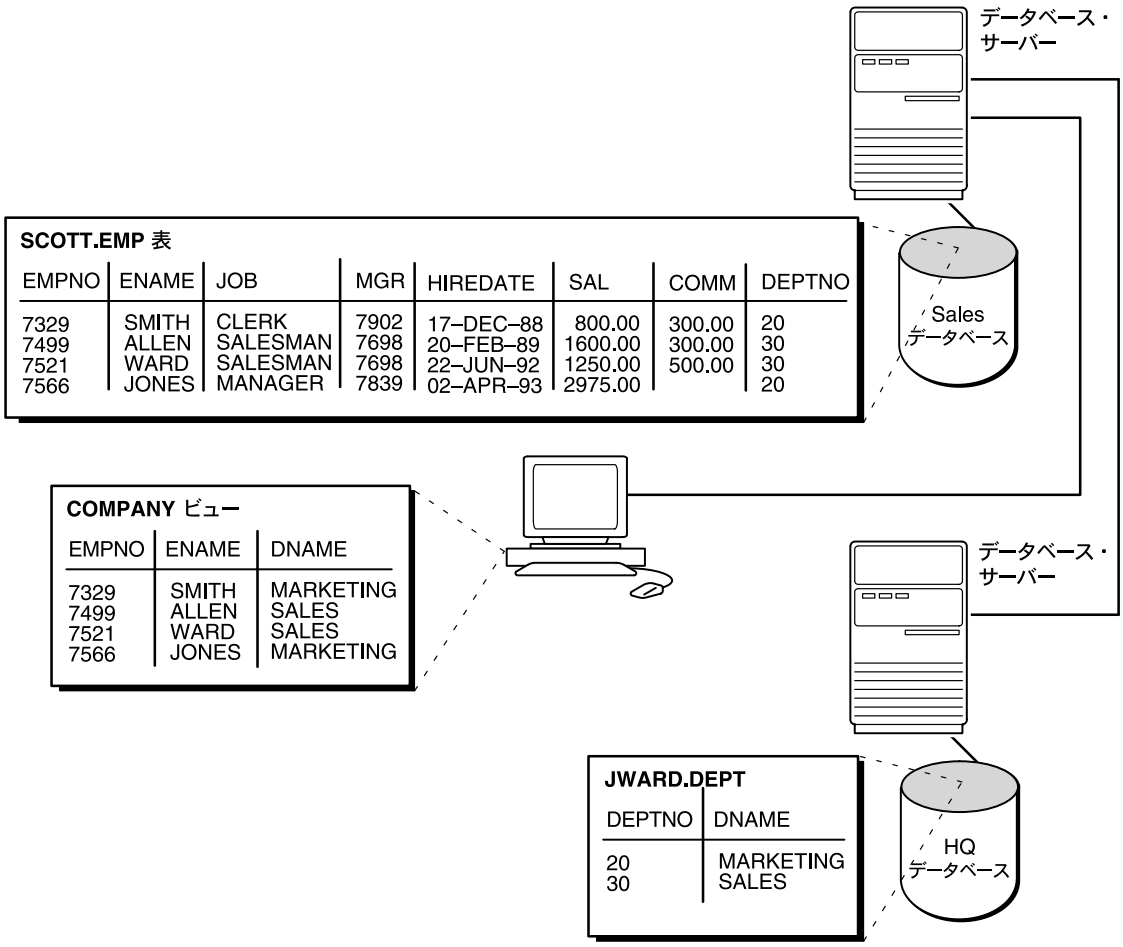
ビューを使用した位置の透過性の作成

ローカル・ビューは、分散データベース・システムにおいてローカルおよびリモートの表に対する位置の透過性を提供できます。

たとえば、emp 表がローカル・データベースに、dept 表がリモート・データベースに、それぞれ格納されているとします。システムのユーザーに対してこれらの表を透過的にするために、ローカル・データとリモート・データを結合するビューをローカル・データベースに作成できます。

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

図 30-3　ビューと位置の透過性



ユーザーはこのビューにアクセスするときに、データが物理的に格納されている場所や、複数の表のデータにアクセスしているかどうかについて知る必要はありません。したがって、ユーザーは必要な情報をより簡単に取得できます。たとえば、次の問合せは、ローカルおよびリモートの両方のデータベース表からのデータを提供します。

```
SELECT * FROM company;
```

ローカル・ビューの所有者は、リモート・ユーザーによってすでに付与されているローカル・ビューのオブジェクト権限のみを付与できます（リモート・ユーザーはデータベース・リンクのタイプによって示されます）。この仕組みは、ローカル・データを参照するビューの権限の管理と似ています。

シノニムを使用した位置の透過性の作成

シノニムは、分散データベース・システム内での位置など、基礎となるオブジェクトの個別情報を隠すので、分散環境と非分散環境の両方で役立ちます。基礎となるオブジェクトを名前変更または移動する必要がある場合でも、シノニムを再定義するだけで済み、シノニムをベースとするアプリケーションは引き続き通常どおり動作します。また、分散データベース・システムのユーザーが使用する SQL 文も、シノニムによって単純化されます。

シノニムの作成

次のもののシノニムを作成できます。

- 表
- 型
- ビュー
- マテリアライズド・ビュー
- 順序
- プロシージャ
- ファンクション
- パッケージ

シノニムはすべて、作成するデータベースのデータ・ディクショナリに格納されるスキーマ・オブジェクトです。シノニムでは、データベース・リンクを介したリモート表へのアクセスを簡単にするために、単一ワードでリモート・データにアクセスでき、それによって特定のオブジェクト名や位置をシノニムのユーザーから隠すことができます。

シノニムを作成するための構文は、次のとおりです。

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

各項目の意味は次のとおりです。

- PUBLIC は、すべてのユーザーがこのシノニムを使用できることを指定するキーワードです。このパラメータを省略するとシノニムがプライベートになり、作成者のみ使用可能になります。パブリック・シノニムは、CREATE PUBLIC SYNONYM システム権限を持つユーザーのみが作成できます。
- *synonym_name* には、ユーザーおよびアプリケーションが参照する代替オブジェクト名を指定します。
- *schema* には、*object_name* で指定されたオブジェクトのスキーマを指定します。このパラメータを省略すると、オブジェクトのスキーマとして作成者のスキーマが使用されます。
- *object_name* には、表、ビュー、順序、マテリアライズド・ビュー、型、プロシージャ、ファンクションまたはパッケージのいずれかを適宜指定します。
- *database_link_name* には、*object_name* で指定されたオブジェクトが存在するリモート・データベースおよびスキーマを識別するデータベース・リンクを指定します。

シノニムには、そのスキーマ内に存在するオブジェクトの中で一意の名前を付ける必要があります。スキーマにスキーマ・オブジェクトがあり、同じ名前のパブリック・シノニムが存在する場合、データベースはスキーマを所有しているユーザーがその名前を参照するときに、常にスキーマ・オブジェクトの方を検索します。

例 : パブリック・シノニムの作成

分散データベース・システム内のすべてのデータベースにおいて、hq データベースに格納されている scott.emp 表のパブリック・シノニムが定義されているとします。

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

表 scott.emp@hq.acme.com の位置はパブリック・シノニムによって隠されているので、使用場所に依存しない従業員管理アプリケーションを設計できます。アプリケーション内の SQL 文は、パブリック・シノニム emp を参照して表にアクセスできます。

また、emp 表を hq データベースから hr データベースに移動する場合も、システムのノードでパブリック・シノニムを変更するだけで済みます。従業員管理アプリケーションは、すべてのノードで引き続き正常に動作します。

権限とシノニムの管理

シノニムは、実際のオブジェクトへの参照です。特定のスキーマ・オブジェクトのシノニムにアクセスするユーザーは、元のスキーマ・オブジェクトそのものに対する権限を持っている必要があります。たとえば、ユーザーがシノニムにアクセスしようとして、そのシノニムが識別する表に対してユーザーが権限を持っていなければ、表またはビューが存在しないというエラーが発生します。

scott がリモート・オブジェクト scott.emp@sales.acme.com の別名としてローカル・シノニム emp を作成したとします。この場合、scott は自分以外のローカル・ユーザーにシノニムのオブジェクト権限を付与することはできません。また、scott はシノニムのローカル権限を付与することもできません。この操作は最終的に sales データベースのリモートの emp 表の権限を付与することになりますが、そのような操作は許可されていないためです。この動作は、ローカルの表またはビューの別名であるシノニムの権限管理とは異なります。

したがって、シノニムを位置の透過性のために使用する場合、ローカル権限は管理できません。ベース・オブジェクトのセキュリティは、リモート・ノードですべて管理されます。たとえば、ユーザー admin は emp_syn シノニムのオブジェクト権限を付与することはできません。

ビューやプロシージャの定義で参照されるデータベース・リンクとは異なり、シノニムで参照されるデータベース・リンクを解決する際は、シノニムへの参照が解析される時点で有効なスキーマが所有しているプライベート・リンクが最初に検索されます。したがって、オブジェクトの適切な解決を保証するには、基礎となるオブジェクトのスキーマをシノニムの定義の中で指定することが特に重要になります。

プロシージャを使用した位置の透過性の作成

プロシージャと呼ばれる PL/SQL プログラム・ユニットは、位置の透過性を提供できます。それには、次の方法があります。

- ローカル・プロシージャを使用したリモート・データの参照
- ローカル・プロシージャを使用したリモート・プロシージャのコール
- ローカル・シノニムを使用したリモート・プロシージャの参照

ローカル・プロシージャを使用したリモート・データの参照

プロシージャおよびファンクションには、スタンドアロンとパッケージのどちらの場合でも、リモート・データを参照する SQL 文を含めることができます。たとえば、次の文で作成されるプロシージャを考えます。

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

ユーザーまたはアプリケーションが fire_emp プロシージャをコールするときに、リモート表が変更されようとしていることは見かけ上わかりません。

プロシージャ内の文でローカルのプロシージャ、ビューまたはシノニムを使用して間接的にリモート・データを参照するときは、2 層目の位置の透過性が可能です。たとえば、次の文はローカル・シノニムを定義します。

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

このシノニムがあれば、次の文を使用して fire_emp プロシージャを作成できます。

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

表 emp@hq.acme.com を名前変更または移動した場合でも、表を参照するローカル・シノニムを変更するだけで済みます。このプロシージャをコールするプロシージャやアプリケーションを変更する必要はありません。

ローカル・プロシージャを使用したリモート・プロシージャのコール

ローカル・プロシージャを使用してリモート・プロシージャをコールできます。リモート・プロシージャでは、要求されたデータ操作言語（DML）を実行できます。たとえば、scott が local_db に接続して次のプロシージャを作成したとします。

```
CONNECT scott@local_db

CREATE PROCEDURE fire_emp (enum NUMBER)
AS
BEGIN
  EXECUTE term_emp@hq.acme.com;
END;
```

ここで、scott は、リモート・データベースに接続して次のリモート・プロシージャを作成するとします。

```
CONNECT scott@hq.acme.com

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

ユーザーまたはアプリケーションが local_db に接続して fire_emp プロシージャをコールすると、このプロシージャはさらに hq.acme.com にあるリモートの term_emp プロシージャをコールします。

ローカル・シノニムを使用したリモート・プロシージャの参照

たとえば、scott がローカルの sales.acme.com データベースに接続して次のプロシージャを作成したとします。

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

この後、ユーザー peggy が supply.acme.com データベースに接続し、scott がリモートの sales データベースで作成したプロシージャに対する次のシノニムを作成します。

```
SQL> CONNECT peggy@supply
SQL> CREATE PUBLIC SYNONYM emp FOR scott.fire_emp@sales.acme.com;
```

supply のローカル・ユーザーは、このシノニムを使用して sales のプロシージャを実行できます。

プロシージャと権限の管理

ローカル・プロシージャに、リモートの表またはビューを参照する文が含まれているとします。ローカル・プロシージャの所有者は、EXECUTE 権限を任意のユーザーに付与することができます。これにより、そのユーザーには、プロシージャを実行する権限とリモート・データに間接的にアクセスする許可が与えられます。

一般に、プロシージャはセキュリティの面で役に立ちます。プロシージャの中で参照されるオブジェクトの権限を明示的にコール側のユーザーに付与する必要はありません。

文の透過性の管理

データベースでは、次に示す標準の DML 文でリモート表を参照することが可能です。

- SELECT（問合せ）
- INSERT
- UPDATE
- DELETE
- SELECT...FOR UPDATE（異機種間システムでは常にサポートされるとはかぎらない）
- LOCK TABLE

結合、集計、副問合せおよび SELECT...FOR UPDATE を含む問合せでは、ローカルおよびリモートの表およびビューをいくつでも参照できます。たとえば、次の問合せは 2 つのリモート表の情報を結合します。

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

同機種環境では、UPDATE、INSERT、DELETE および LOCK TABLE 文は、ローカルおよびリモートの両方の表を参照できます。リモート・データを更新するためのプログラミングは不要です。たとえば、次の文はローカル・データベースの jward スキーマの emp 表から行を選択して、scott.sales スキーマのリモート表 emp に新しい行を挿入します。

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

文の透過性に関する制限事項

文の透過性には、いくつかの制限事項が適用されます。

- リモートの Oracle Database 以外のシステム上のオブジェクトを更新するデータ操作言語（DML）文では、ローカルの Oracle Database のオブジェクトを参照できません。たとえば、次のような文を実行するとエラーになります。

  ```
  INSERT INTO remote_table@link as SELECT * FROM local_table;
  ```

- 1 つの SQL 文において、参照されるすべての LONG および LONG RAW の列、順序、更新済みの表およびロック済みの表は、同じノードに存在する必要があります。

- データベースでは、同機種システムにおけるリモートのデータ定義言語（DDL）文（CREATE、ALTER、DROP など）は使用できません。ただし、次の例のように DBMS_SQL パッケージのプロシージャのリモート実行を使用する場合を除きます。

  ```
  DBMS_SQL.PARSE@link_name(crs, 'drop table emp', v7);
  ```

  異機種間システムでは、パススルー機能によって DDL の実行が可能です。
- ANALYZE 文の LIST CHAINED ROWS 句は、リモート表を参照できません。
- 分散データベース・システムでは、SYSDATE、USER、UID、USERENV などの環境に依存した SQL ファンクションは、データベースによって、その文（または文の一部）が実行される場所にかかわらず、常にローカル・サーバーについて評価されます。

  注意： Oracle Database は、USERENV ファンクションを問合せの用途でのみサポートしています。

- リモート・オブジェクトのアクセスに関連して、次のようなパフォーマンス上の制限事項があります。
  - リモート・ビューは統計データを持ちません。
  - パーティション表に対する問合せは、最適化されない場合があります。
  - リモート表で考慮される索引の数は 20 以下です。
  - コンポジット索引で使用される列の数は 20 以下です。
- Oracle Database の分散読込み一貫性の実装には制限があり、これによってあるノードが別のノードに対して古い状態になる可能性があります。問合せを実行したときに、読込み一貫性に従ってデータが取得されているにもかかわらず、そのデータが古い場合があります。この問題の管理方法の詳細は、33-22 ページの「読込み一貫性の管理」を参照してください。

  関連項目： DBMS_SQL パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

分散データベースの管理 : 例

ここでは、データベース・リンクの管理例を示します。

例 1: パブリック固定ユーザーのデータベース・リンクの作成

次の文では、ローカル・データベースに jane として接続し、データベース sales に対してパブリック固定ユーザー scott のデータベース・リンクを作成しています。データベースには、ネット・サービス名 sldb を介してアクセスします。

```
CONNECT jane@local

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

この文の実行後は、ローカル・データベースに接続する任意のユーザーが、sales.division3.acme.com データベース・リンクを使用してリモート・データベースに接続できます。各ユーザーは、リモート・データベースのスキーマ scott に接続します。

ユーザーは、次の SQL 問合せを発行して、scott のリモート・スキーマの表 emp にアクセスできます。

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

各アプリケーションまたはユーザー・セッションは、サーバーの共通アカウントへの接続をそれぞれ別々に作成します。リモート・データベースへの接続は、アプリケーションの実行中またはユーザー・セッションの継続期間中オープンしたままです。

例 2: パブリック固定ユーザーの共有データベース・リンクの作成

次の例では、ローカル・データベースに dana として接続し、ネット・サービス名 sldb を使用して sales データベースへのパブリック・リンクを作成しています。このリンクによって、リモート・データベースに scott として接続でき、このユーザーが scott として認証されます。

```
CONNECT dana@local

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

ローカルの共有サーバーに接続する任意のユーザーがこのデータベース・リンクを使用し、共有サーバー・プロセスを介してリモートの sales データベースに接続できます。その後ユーザーは、scott スキーマの表を問い合せることができます。

前述の例では、ローカルの共有サーバーはリモート・サーバーへの接続を 1 つずつ確立できます。ローカルの共有サーバー・プロセスで sales.division3.acme.com データベース・リンクを介したリモート・サーバーへのアクセスが必要になると、そのたびにローカルの共有サーバー・プロセスが確立済みのネットワーク接続を再利用します。

例 3: パブリック接続ユーザーのデータベース・リンクの作成

次の例では、ローカル・データベースに larry として接続し、ネット・サービス名 sldb を使用してデータベースへのパブリック・リンクを作成しています。

```
CONNECT larry@local

CREATE PUBLIC DATABASE LINK redwood
  USING 'sldb';
```

ローカル・データベースに接続する任意のユーザーが、redwood データベース・リンクを使用できます。データベース・リンクを使用するローカル・データベースの接続ユーザーによって、リモート・スキーマが決まります。

接続ユーザー scott がデータベース・リンクを使用した場合、データベース・リンクはリモート・スキーマ scott に接続します。接続ユーザー fox がデータベース・リンクを使用した場合、データベース・リンクはリモート・スキーマ fox に接続します。

リモート・スキーマ fox が emp スキーマ・オブジェクトを解決できない場合は、ローカル・データベースでローカル・ユーザー fox が次の文を実行すると失敗します。つまり、sales.division3.acme.com の fox スキーマに emp が表、ビューまたは（パブリック）シノニムとして存在しない場合は、エラーが返されます。

```
CONNECT fox@local

SELECT * FROM emp@redwood;
```

例 4: パブリック接続ユーザーの共有データベース・リンクの作成

次の例では、ローカル・データベースに neil として接続し、ネット・サービス名 sldb を使用して sales データベースへの共有パブリック・リンクを作成しています。ユーザーは、crazy/horse というユーザー ID/ パスワードによって認証されます。次の文は、パブリック接続ユーザーの共有データベース・リンクを作成します。

```
CONNECT neil@local

CREATE SHARED PUBLIC DATABASE LINK sales.division3.acme.com
  AUTHENTICATED BY crazy IDENTIFIED BY horse
  USING 'sldb';
```

ローカル・サーバーに接続する各ユーザーは、この共有データベース・リンクを使用してリモート・データベースに接続し、対応するリモート・スキーマの表を問い合せることができます。

ローカルの共有サーバー・プロセスは、それぞれリモート・サーバーへの接続を 1 つずつ確立します。ローカルのサーバー・プロセスで sales.division3.acme.com データベース・リンクを介したリモート・サーバーへのアクセスが必要になると、たとえ接続ユーザーが異なるユーザーであっても、そのたびにローカルの共有サーバー・プロセスが確立済みのネットワーク接続を再利用します。

このデータベース・リンクが頻繁に使用されると、最終的にローカル・データベースのすべての共有サーバーがリモート接続を持つことになります。この時点で、新しいユーザーがこの共有データベース・リンクを使用しても、リモート・サーバーへの物理的な接続は新たに確立されません。

例 5: パブリック現行ユーザーのデータベース・リンクの作成

次の例では、ローカル・データベースに接続ユーザーとして接続し、ネット・サービス名 sldb を使用して sales データベースへのパブリック・リンクを作成しています。次の文は、パブリック現行ユーザーのデータベース・リンクを作成します。

```
CONNECT bart@local

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

> **注意：** このリンクを使用するには、現行ユーザーがグローバル・ユーザーであることが必要です。

このデータベース・リンクの結果は、次のとおりです。

scott がリモートの emp 表から 1 行を削除するローカル・プロシージャ fire_emp を作成し、fire_emp の実行権限を ford に付与したとします。

```
CONNECT scott@local_db

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

GRANT EXECUTE ON fire_emp TO ford;
```

ここで、ford はローカル・データベースに接続して scott のプロシージャを実行したとします。

```
CONNECT ford@local_db

EXECUTE PROCEDURE scott.fire_emp (enum 10345);
```

ford がプロシージャ scott.fire_emp を実行するとき、プロシージャは scott の権限のもとで実行されます。現行ユーザー・データベース・リンクが使用されているため、接続は ford のリモート・スキーマではなく、scott のリモート・スキーマに確立されます。scott はグローバル・ユーザーである必要がありますが、ford はグローバル・ユーザーでなくてもかまいません。

> **注意：** かわりに接続ユーザー・データベース・リンクが使用された場合、接続は ford のリモート・スキーマに確立されます。実行者権限と権限の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

scott のリモート・スキーマへの固定ユーザー・データベース・リンクを使用しても、同じ結果を得ることができます。

31

分散データベース・システムのアプリケーション開発

この章の内容は次のとおりです。

関連項目： Oracle Database 環境でのアプリケーション開発の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

アプリケーション・データの分散の管理

分散データベース環境では、データベース管理者（DBA）と共同でデータの最適な格納場所を決めます。その際、次の点について考慮します。

- 個々の場所から転送されるトランザクション数
- 各ノードで使用されるデータ（表の部分）の量
- パフォーマンス特性とネットワークの信頼性
- 各種ノードの速度とディスクの容量
- ノードまたはリンクが使用できないときの重要度
- 表間の参照整合性の必要性

データベース・リンクにより確立される接続の制御

SQL 文やリモート・プロシージャ・コールの中でグローバル・オブジェクト名が参照されると、ローカル・ユーザーにかわってデータベース・リンクがリモート・データベース内のセッションへの接続を確立します。リモートの接続とセッションは、それまでにローカルのユーザー・セッションに対して接続が確立されていない場合にのみ作成されます。

リモート・データベースとの間に確立された接続とセッションは、アプリケーションやユーザーによって明示的に終了されないかぎり、ローカル・ユーザーのセッションの間存続します。データベース・リンクを経由して SELECT 文を発行すると、UNDO セグメントにトランザクション・ロックが設定されます。セグメントを再び解放するには、COMMIT 文または ROLLBACK 文を発行する必要があります。

アプリケーションで不要になった高コストの接続を切断するには、データベース・リンクを使用して確立されたリモート接続を終了することが有効です。リモートの接続とセッションを終了するには、CLOSE DATABASE LINK 句を指定した ALTER SESSION 文を使用します。たとえば、次のトランザクションを発行した場合を考えます。

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

次の文は、sales データベース・リンクが指すリモート・データベース内のセッションを終了します。

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

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

注意： データベース・リンクをクローズする前に、まずそのリンクを使用しているカーソルをすべてクローズし、次にそのリンクを使用している現行トランザクションがあればそのトランザクションを終了してください。

関連項目： ALTER SESSION 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

分散システムの参照整合性の維持

たとえば、整合性制約違反など、分散型の文の一部が失敗した場合、データベースはエラー番号 ORA-02055 を返します。以降の文またはプロシージャ・コールは、ROLLBACK または ROLLBACK TO SAVEPOINT が発行されるまで、エラー番号 ORA-02067 を返します。

返されたエラー・メッセージをすべてチェックするようにアプリケーションを設計し、分散更新の一部が失敗したことを示すエラー・メッセージがないかを確認します。失敗を検出した場合は、トランザクション全体をロールバックしてからアプリケーションの処理を進めるようにしてください。

データベースでは、宣言参照整合性の制約を分散システムのノード間で定義することは許可されていません。つまり、1 つの表での宣言参照整合性の制約では、リモート表の主キーまたは一意キーを参照する外部キーを指定することはできません。ただし、トリガーを使用してノード間の親子の表関係を維持することは可能です。

トリガーを使用して分散データベースのノード間で参照整合性を定義する場合は、ネットワークの障害によって親表だけでなく子表へのアクセスも制限される可能性があります。たとえば、子表が sales データベースにあり、親表が hq データベースにあるとします。2 つのデータベース間のネットワーク接続が失敗した場合、子表に対する一部のデータ操作言語（DML）文（子表に行を挿入する文や子表内の外部キーの値を更新する文など）が実行を継続できない場合があります。これは、参照整合性トリガーが hq データベース内の親表にアクセスする必要があるためです。

> **関連項目：** トリガーを使用した参照整合性の規定の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

分散問合せのチューニング

ローカルの Oracle Database サーバーは、分散問合せを対応する数のリモート問合せに分割し、次に、その問合せを実行するためにリモート・ノードに送信します。リモート・ノードは問合せを実行し、その結果をローカル・ノードに返します。次に、ローカル・ノードは必要な後処理を実行し、その結果をユーザーまたはアプリケーションに返します。

問合せの処理が最適化されるようにアプリケーションを設計するには、いくつかの方法があります。この項の内容は、次のとおりです。

- 連結インライン・ビューの使用
- コストベース最適化の使用
- ヒントの使用
- 実行計画の分析

連結インライン・ビューの使用

分散問合せを最適化する最も効果的な方法は、リモート・データベースへのアクセスをできるだけ抑えて必要なデータのみを取得することです。

たとえば、分散問合せで 5 つのリモート表を 2 つの異なるリモート・データベースから参照し、複合フィルタ（WHERE r1.salary + r2.salary > 50000 など）を使用するとします。この場合、リモート・データベースへのアクセスを 1 回にし、フィルタをリモート・サイトで適用するように問合せをリライトすることで、問合せのパフォーマンスを改善できます。このリライトにより、問合せを実行するサイトに転送されるデータの量が少なくなります。

リモート・データベースへのアクセスが 1 回になるように問合せをリライトするには、連結インライン・ビューを使用します。用語の定義は次のとおりです。

- **連結**

  同じデータベースにある 2 つ以上の表。

- **インライン・ビュー**

  親の SELECT 文の表に置き換えられる SELECT 文。次のかっこで囲まれた部分の埋込み SELECT 文がインライン・ビューの例です。

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

- **連結インライン・ビュー**

  複数の表のデータを 1 つのデータベースのみから選択するインライン・ビュー。これにより、リモート・データベースへのアクセス回数が減り、分散問合せのパフォーマンスが向上します。

連結インライン・ビューを使用して分散問合せを構成し、分散問合せのパフォーマンスを改善することをお薦めします。Oracle Database のコストベース最適化を使用すると、分散問合せの多くが透過的にリライトされ、連結インライン・ビューによるパフォーマンスの向上が得られます。

コストベース最適化の使用

連結インライン・ビューによるクエリー・リライトに加えて、コストベース最適化の方法を使用すると、参照先の表から収集される統計とオプティマイザが実行する計算に従って分散問合せが最適化されます。

たとえば、コストベースの最適化によって次の問合せを分析します。この例では、表統計が使用できると仮定しています。CREATE TABLE 文内の問合せが分析されます。

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

この文は、次のようにリライトされます。

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

このリライトにより、別名 v がインライン・ビューに割り当てられます。このインライン・ビューは、前述の SELECT 文内の表として参照できます。連結インライン・ビューを作成することで、リモート・サイトで実行される問合せの量が減り、それによって高コストのネットワーク通信量が減少します。

コストベース最適化の動作の仕組み

オプティマイザの主なタスクは、連結インライン・ビューを使用するように分散問合せをリライトすることです。この最適化は、次の 3 つの手順で実行されます。

1. マージ可能なビューがすべてマージされます。
2. オプティマイザが連結問合せブロックのテストを実行します。
3. オプティマイザが連結インライン・ビューを使用して問合せをリライトします。

問合せがリライトされた後、その問合せが実行され、データ・セットがユーザーに返されます。

コストベース最適化をユーザーに対して透過的に実行する場合は、複数の分散問合せのパフォーマンスを改善することはできません。特に、次のものが分散問合せに含まれている場合、コストベース最適化は効果がありません。

- 集計
- 副問合せ
- 複合 SQL

これらの要素が 1 つでも分散問合せに含まれている場合は、問合せの修正方法と、分散問合せのパフォーマンスを改善するためのヒントの使用について、31-6 ページの「ヒントの使用」を参照してください。

コストベース最適化の設定

分散問合せのパフォーマンス改善のためにコストベース最適化を使用するようにシステムを設定すると、その処理がユーザーに対して透過的になります。つまり、問合せの発行時に自動的に最適化が実行されます。

コストベースの最適化を利用するようにシステムを設定するには、次のタスクを完了する必要があります。

- 環境の設定
- 表の分析

環境の設定 コストベース最適化を使用可能にするには、OPTIMIZER_MODE 初期化パラメータを CHOOSE または COST に設定します。このパラメータは次の方法で設定できます。

- 初期化パラメータ・ファイルの OPTIMIZER_MODE パラメータを変更します。
- ALTER SESSION 文を発行してセッション・レベルで設定します。

OPTIMIZER_MODE 初期化パラメータをセッション・レベルで設定するには、次のどちらかの文を発行します。

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

> **関連項目：** パラメータ・ファイルの OPTIMIZER_MODE 初期化パラメータの設定と、コストベース最適化方法を使用するためのシステムの構成の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

表の分析 コストベース最適化で分散問合せのための最も効率的なパスが選択されるようにするには、問合せに関係する表の正確な統計を提供する必要があります。そのためには、DBMS_STATS パッケージまたは ANALYZE 文を使用します。

注意： DBMS_STATS プロシージャまたは ANALYZE 文を実行するには、表に対してローカルに接続する必要があります。たとえば、次の文は実行できません。

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

この ANALYZE 文または等価の DBMS_STATS プロシージャを実行するには、先にリモート・サイトへの接続が必要になります。

次の DBMS_STATS プロシージャを使用すると、特定のクラスのオプティマイザ統計を収集できます。

- GATHER_INDEX_STATS
- GATHER_TABLE_STATS
- GATHER_SCHEMA_STATS
- GATHER_DATABASE_STATS

たとえば、分散トランザクションが日常的に scott.dept 表にアクセスするとします。コストベースのオプティマイザが引き続き最適な方法を確実に選択するように、次の文を実行します。

```
BEGIN
   DBMS_STATS.GATHER_TABLE_STATS ('scott', 'dept');
END;
```

関連項目：

- 統計生成の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。
- DBMS_STATS パッケージの使用の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ヒントの使用

文が十分に最適化されない場合は、ヒントを使用してコストベース最適化の機能を拡張できます。特に、独自の問合せを記述して連結インライン・ビューを利用する場合は、分散問合せがリライトされないようにコストベース・オプティマイザに指示を与えます。

また、データベース環境に関する特別な情報（統計、負荷、ネットワークおよび CPU の制限事項、分散問合せなど）を持っている場合は、ヒントを指定してコストベース最適化を適切に誘導できます。たとえば、データベース環境の情報に基づく連結インライン・ビューを使用して、独自に最適化した問合せを記述した場合は、NO_MERGE ヒントを指定することにより、オプティマイザが問合せをリライトしないようにできます。

この手法は、分散問合せに集計、副問合せまたは複合 SQL が含まれている場合に特に役立ちます。このタイプの分散問合せはオプティマイザによってリライトできないため、NO_MERGE を指定して、オプティマイザが 31-5 ページの「コストベース最適化の動作の仕組み」で説明されている手順を省略するように指示します。

DRIVING_SITE ヒントを使用すると、リモート・サイトを問合せ実行サイトとして機能するように定義できます。この方法では、問合せがリモート・サイトで実行され、データがローカル・サイトに返されます。リモート・サイトにデータの大部分が格納されているときは、このヒントが特に役立ちます。

関連項目： ヒントの使用の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

NO_MERGE ヒントの使用

NO_MERGE ヒントは、データベースでインライン・ビューと連結されない可能性のある SQL 文とがマージされないようにするために使用します（31-6 ページの「ヒントの使用」を参照）。このヒントは、SELECT 文に埋め込みます。インライン・ビューを使用する SELECT 文の先頭に引数として指定するか、またはインライン・ビューを定義する問合せブロック内に指定します。

```
/* with argument */

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

/* in query block */

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

通常、このヒントは、データベース環境の情報に基づいて最適化した問合せを作成したときに使用します。

DRIVING_SITE ヒントの使用

DRIVING_SITE ヒントを使用すると、問合せを実行するサイトを指定できます。問合せを実行するサイトはコストベース最適化によって決定されるのが最適ですが、オプティマイザの判断を変更する方がよい場合は、実行サイトを手動で指定できます。

DRIVING_SITE ヒントを使用した SELECT 文の例を次に示します。

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

実行計画の分析

分散問合せのチューニングの際に重要なこととして、実行計画の分析があります。分析結果から得られるフィードバックは、データベースのテストと検証を行う上で重要な要素になります。計画を比較するときは、検証が特に重要になります。たとえば、コストベースの最適化によって分散問合せを最適化する実行計画を、ヒント、連結インライン・ビューおよびその他の方法を使用して問合せを手動で最適化する計画と比較します。

> **関連項目：** 実行計画、EXPLAIN PLAN 文およびその結果の解釈方法の詳細は、『Oracle Database パフォーマンス・チューニング・ガイド』を参照してください。

計画を格納するためのデータベースの準備

分散問合せの実行計画を表示できるようにするには、まずデータベース内に実行計画を格納する場所を準備します。そのために、スクリプトを実行します。次のスクリプトを実行し、データベース内に実行計画を格納する場所を準備します。

```
SQL> @UTLXPLAN.SQL
```

> **注意：** utlxplan.sql ファイルは、$ORACLE_HOME/rdbms/admin ディレクトリにあります。

utlxplan.sql を実行すると、現行スキーマ内に、実行計画を一時的に格納する PLAN_TABLE という表が作成されます。

実行計画の生成

データベース内に実行計画を格納する場所を準備すると、指定した問合せの計画を表示するための準備が完了します。SQL 文を直接実行するかわりに、EXPLAIN PLAN FOR 句に文を追加します。たとえば、次のような文を実行できます。

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

実行計画の表示

前述の SQL 文を実行すると、すでに作成した PLAN_TABLE に実行計画が一時的に格納されます。実行計画の結果を表示するには、次のスクリプトを実行します。

```
@UTLXPLS.SQL
```

> **注意：** utlxpls.sql ファイルは $ORACLE_HOME/rdbms/admin ディレクトリにあります。

utlxpls.sql スクリプトを実行すると、指定した SELECT 文の実行計画が表示されます。結果は、次のように書式化されます。

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

独自の連結インライン・ビューを記述するか、またはヒントを使用して、分散問合せを手動で最適化しようとする場合は、手動最適化の前および後に実行計画を生成するのが最適です。これら両方の実行計画を使用することで、手動最適化の効果を比較し、必要に応じて変更を加え、分散問合せのパフォーマンスを改善できます。

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

```
SELECT OTHER
FROM PLAN_TABLE
   WHERE operation = 'REMOTE';
```

出力例を次に示します。

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

> **注意：** OTHER 列の内容全体がうまく表示されない場合は、次の SQL*Plus コマンドを実行してください。
>
> ```
> SET LONG 9999999
> ```

リモート・プロシージャのエラー処理

データベースがプロシージャをローカルまたはリモートの位置で実行するときには、次の 4 種類の例外が発生する可能性があります。

- PL/SQL のユーザー定義例外。この例外は、EXCEPTION キーワードを使用して宣言する必要があります。
- PL/SQL の事前定義例外。NO_DATA_FOUND キーワードなど。
- SQL エラー。ORA-00900 や ORA-02015 など。
- RAISE_APPLICATION_ERROR() プロシージャを使用して生成されるアプリケーション例外。

ローカル・プロシージャを使用するときは、次のような例外ハンドラを記述して、これらのメッセージをトラップできます。

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

WHEN 句には例外名が必要です。RAISE_APPLICATION_ERROR で生成された例外など、例外が名前を持っていない場合は、PRAGMA_EXCEPTION_INIT を使用して名前を割り当てることができます。次に例を示します。

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

リモート・プロシージャをコールするときは、例外をローカル・プロシージャの例外ハンドラで処理できます。リモート・プロシージャは、エラー番号をローカルのコール側プロシージャに返す必要があります。ローカルのコール側プロシージャは、受け取った例外を前の例で示したように処理します。PL/SQL のユーザー定義例外は、常に ORA-06510 をローカル・プロシージャに返します。

したがって、2 つの異なるユーザー定義例外をエラー番号で区別することはできません。その他のリモート例外はすべて、ローカル例外と同じ方法で処理できます。

> **関連項目：** PL/SQL プロシージャの詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

32

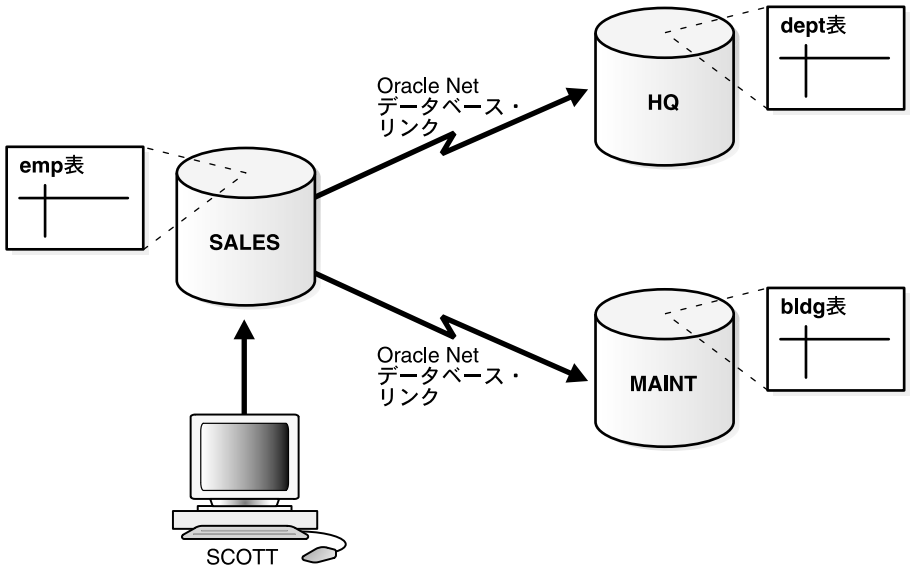
分散トランザクションの概念

この章の内容は次のとおりです。

分散トランザクションの概要

分散トランザクションは 1 つ以上の文からなり、それらが個別に、またはグループとして、分散データベースの複数ノードのデータを更新します。たとえば、図 32-1 に示すデータベース構成を考えます。

図 32-1　分散システム



scott によって実行される次の分散トランザクションは、ローカルの sales データベース、リモートの hq データベース、およびリモートの maint データベースを更新します。

```
UPDATE scott.dept@hq.us.acme.com
  SET loc = 'REDWOOD SHORES'
  WHERE deptno = 10;
UPDATE scott.emp
  SET deptno = 11
  WHERE deptno = 10;
UPDATE scott.bldg@maint.us.acme.com
  SET room = 1225
  WHERE room = 1163;
COMMIT;
```

> **注意：**　トランザクションのすべての文が 1 つのリモート・ノードのみを参照している場合、そのトランザクションは分散トランザクションではなくリモート・トランザクションです。

分散トランザクションでは、次の 2 種類の操作が許可されます。

- DML および DDL トランザクション
- トランザクション制御文

DML および DDL トランザクション

分散トランザクションでサポートされているデータ操作言語（DML）およびデータ定義言語（DDL）操作は、次のとおりです。

- CREATE TABLE AS SELECT
- DELETE
- INSERT（デフォルトおよびダイレクト・ロード）
- LOCK TABLE
- SELECT
- SELECT FOR UPDATE

DML 文および DDL 文はパラレルに実行でき、ダイレクト・ロード・インサート文はシリアルに実行できます。ただし、次の制限に注意してください。

- リモート操作はすべて SELECT 文である必要があります。
- これらの文は、別の分散トランザクション内の句であってはいけません。
- INSERT、UPDATE または DELETE 文の *table_expression_clause* で参照される表がリモートの場合、実行はパラレルではなくシリアルになります。
- パラレル DML/DDL またはダイレクト・ロード・インサートの発行後にリモート操作を実行することはできません。
- XA または OCI を使用してトランザクションを開始した場合、そのトランザクションはシリアルに実行されます。
- パラレル操作の実行元であるトランザクションで、ループバック操作を実行することはできません。たとえば、実際はローカル・オブジェクトのシノニムであるリモート・オブジェクトを参照することはできません。
- トランザクションで SELECT 以外の分散操作を実行する場合、DML はパラレル化されません。

トランザクション制御文

サポートされているトランザクション制御文は、次のとおりです。

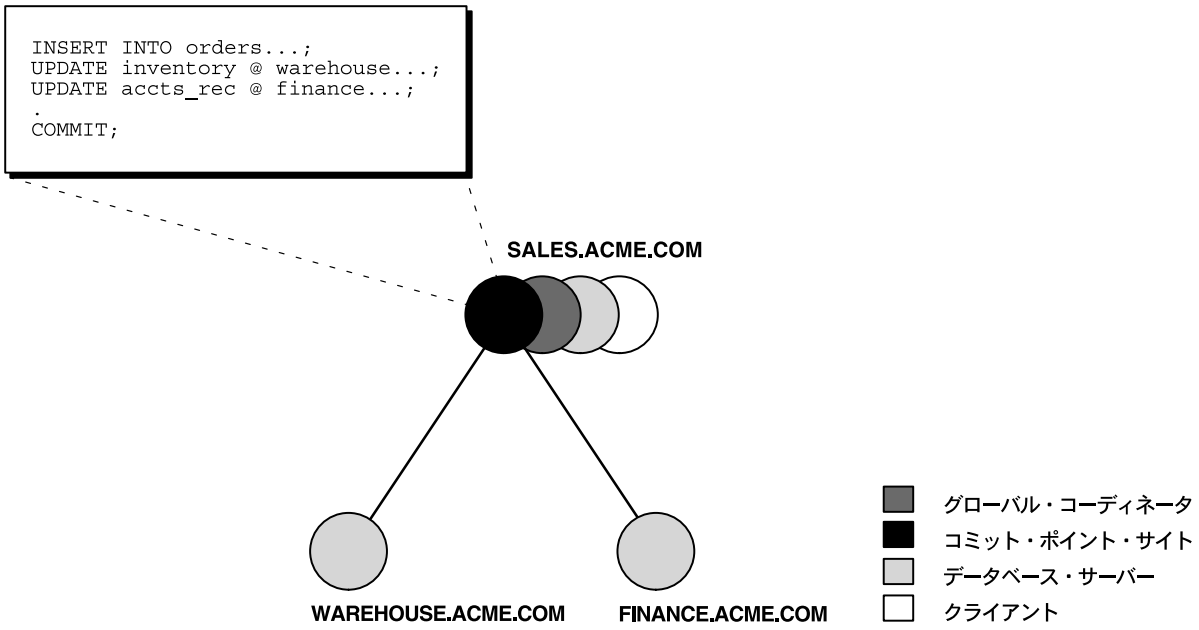
- COMMIT
- ROLLBACK
- SAVEPOINT

> **関連項目：** これらの SQL 文の詳細は、『Oracle Database SQL リファレンス』を参照してください。

分散トランザクションのセッション・ツリー

分散トランザクションで文が発行されると、データベースはトランザクションに参加しているすべてのノードの**セッション・ツリー**を定義します。セッション・ツリーとは、セッション間の関係とセッションのロールを表す階層モデルです。セッション・ツリーの例を図 32-2 に示します。

図 32-2　セッション・ツリーの例



分散トランザクションのセッション・ツリーに参加しているすべてのノードは、次に示すロールを 1 つ以上持ちます。

ロール	説明
クライアント	異なるノードに属するデータベース内の情報を参照するノード。
データベース・サーバー	別のノードからの情報の要求を受け取るノード。
グローバル・コーディネータ	分散トランザクションの実行元ノード。
ローカル・コーディネータ	他のノードのデータを強制的に参照して、自身のトランザクション部分を完了するノード。
コミット・ポイント・サイト	グローバル・コーディネータの指示に従ってトランザクションをコミットまたはロールバックするノード。

分散トランザクションのノードが果たすロールは、次の条件によって決まります。

- トランザクションがローカルとリモートのどちらであるか。
- ノードの**コミット・ポイント強度**（32-6 ページの「コミット・ポイント・サイト」を参照）。
- 要求されたすべてのデータがノードで使用可能か、またはトランザクションを完了するために他のノードを参照する必要があるか。
- ノードが読取り専用かどうか。

クライアント

情報を別のノードのデータベースから参照するとき、ノードはクライアントとして機能します。参照先のノードはデータベース・サーバーです。図 32-2 のノード sales は、warehouse データベースおよび finance データベースが稼働しているノードのクライアントです。

データベース・サーバー

データベース・サーバーは、クライアントがデータを要求する要求先データベースが稼働しているノードです。

図 32-2 では、sales ノードのアプリケーションは、warehouse ノードおよび finance ノードのデータにアクセスする分散トランザクションを開始します。したがって、sales.acme.com はクライアント・ノードのロールを持ち、warehouse および finance はどちらもデータベース・サーバーのロールを持ちます。この例では、sales はデータベース・サーバーとクライアントを兼務しています。これは、アプリケーションが sales データベースのデータも変更するためです。

ローカル・コーディネータ

自身の分散トランザクション部分を完了するために他のノードのデータを参照する必要があるノードのことを、ローカル・コーディネータと呼びます。図 32-2 では、sales は自身が直接参照している warehouse ノードおよび finance ノードを調整するので、ローカル・コーディネータになります。ノード sales はまた、トランザクションに関係するすべてのノードを調整することから、グローバル・コーディネータにもなります。

ローカル・コーディネータは、自身が直接やり取りするノードの間で次のようにトランザクションを調整する役目を果たします。

- これらのノード間でトランザクションのステータス情報を受け渡します。
- これらのノードに問合せを渡します。
- これらのノードから問合せを受け取り、他のノードに渡します。
- 問合せの結果を開始元のノードに返します。

グローバル・コーディネータ

分散トランザクションの実行元であるノードのことを、グローバル・コーディネータと呼びます。分散トランザクションを発行するデータベース・アプリケーションは、グローバル・コーディネータとして機能しているノードに直接接続します。たとえば、図 32-2 では、ノード sales で発行されたトランザクションは、データベース・サーバー warehouse および finance の情報を参照します。したがって、sales.acme.com は、この分散トランザクションのグローバル・コーディネータになります。

グローバル・コーディネータは、セッション・ツリーの親またはルートになります。グローバル・コーディネータは、分散トランザクション処理中に次の操作を実行します。

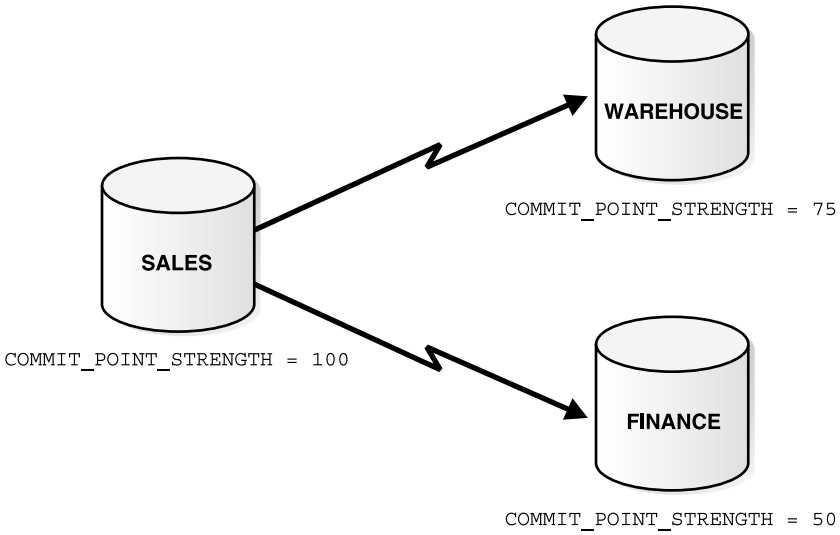
- 分散トランザクションのすべての SQL 文やリモート・プロシージャ・コールなどを参照先のノードに直接送り、それによってセッション・ツリーを構成します。
- コミット・ポイント・サイト以外のすべての直接参照先ノードに対して、トランザクションの準備をするように指示します。
- すべてのノードの準備が正常に完了した場合に、トランザクションのグローバル・コミットを開始するようにコミット・ポイント・サイトに対して指示します。
- ノードから異常終了の応答があった場合に、トランザクションのグローバル・ロールバックを開始するようにすべてのノードに対して指示します。

コミット・ポイント・サイト

コミット・ポイント・サイトの役割は、グローバル・コーディネータの指示に従ってコミットまたはロールバックの操作を開始することです。システム管理者は、すべてのノードにコミット・ポイント強度を割り当てることで、セッション・ツリー内のノードの 1 つをコミット・ポイント・サイトとして必ず指定します。コミット・ポイント・サイトには、最も重要なデータを格納するノードを選択してください。

図 32-3 は、sales がコミット・ポイント・サイトとして機能している分散システムの例です。

図 32-3 コミット・ポイント・サイト



コミット・ポイント・サイトは、分散トランザクションに関係する他のすべてのノードと次の点で区別されます。

- コミット・ポイント・サイトが準備完了状態に入ることはありません。そのため、コミット・ポイント・サイトに最も重要なデータが格納されている場合は、たとえ障害が起きたとしても、データがインダウトのままになることはありません。障害が発生すると、障害を起こしたノードは準備完了状態のままになり、インダウト・トランザクションが解決されるまで必要なロックが保持されます。
- コミット・ポイント・サイトは、トランザクションに関係している他のノードよりも先にコミットします。実際は、コミット・ポイント・サイトでの分散トランザクションの結果によって、すべてのノードでのトランザクションがコミットされるかロールバックされるかが決まり、他のノードはコミット・ポイント・サイトの指示に従います。グローバル・コーディネータは、すべてのノードでコミット・ポイント・サイトと同様にトランザクションが完了することを保証します。

分散トランザクションのコミットの仕組み

分散トランザクションは、コミット・ポイント以外のすべてのサイトで準備が完了した後、コミットされたとみなされますが、実際には、トランザクションはコミット・ポイント・サイトで先にコミットされています。コミット・ポイント・サイトの REDO ログは、このノードで分散トランザクションがコミットされるとただちに更新されます。

コミット・ポイント・ログにはコミットの記録があります。そのため、たとえ参加中のノードの一部がまだ準備完了状態であり、それらのノードで実際にトランザクションがコミットされていない場合であっても、トランザクションはコミットされたとみなされます。同様の意味で、コミット・ポイント・サイトでコミットがまだログに記録されていない場合には、分散トランザクションはコミットされていないとみなされます。

コミット・ポイント強度

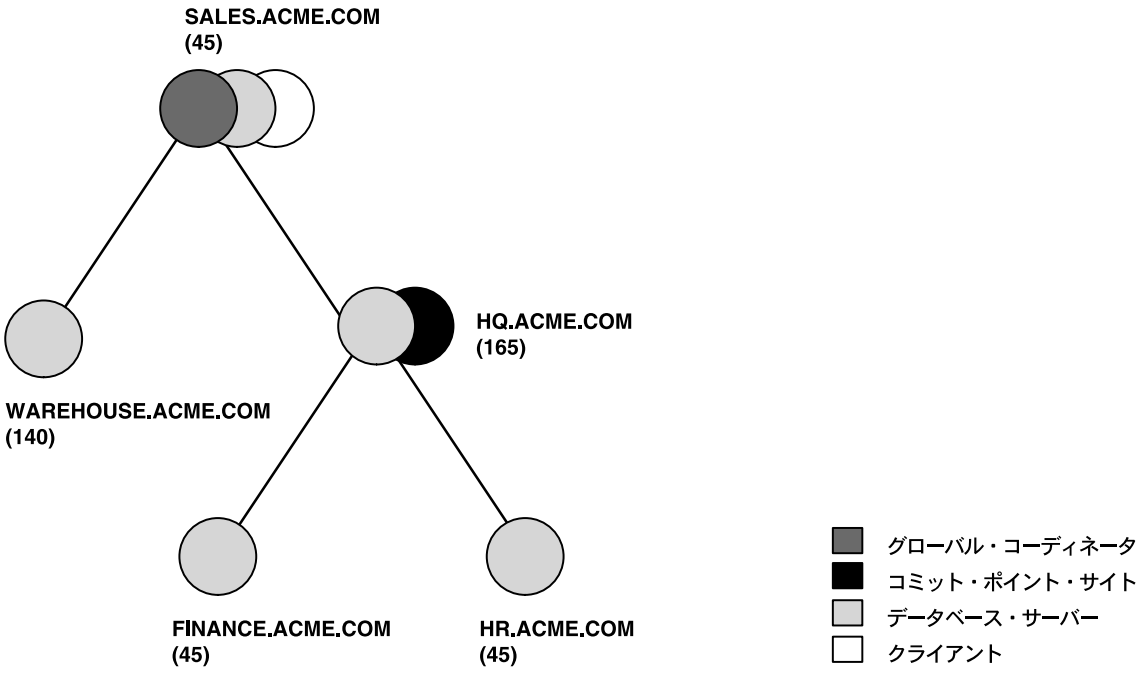
データベース・サーバーには、必ずコミット・ポイント強度を割り当てる必要があります。データベース・サーバーが分散トランザクション内で参照される場合、そのコミット・ポイント強度の値によって、2 フェーズ・コミットにおける役割が決まります。具体的には、このコミット・ポイント強度によって、どのノードが分散トランザクションのコミット・ポイント・サイトになり、他のすべてのノードより前にコミットするかが決まります。この値を指定するには、初期化パラメータ COMMIT_POINT_STRENGTH を使用します。ここでは、データベースがコミット・ポイント・サイトを決定する仕組みについて説明します。

準備フェーズの冒頭で決定されるコミット・ポイント・サイトは、トランザクションに参加しているノードの中からのみ選択されます。次に示す一連のイベントが発生します。

1. データベースは、グローバル・コーディネータが直接参照しているノードの中で、最も高いコミット・ポイント強度を持つノードをコミット・ポイント・サイトとして選択します。
2. 最初に選択されたノードは、このトランザクションの情報を取得する必要のあるノードの中で、自身よりも高いコミット・ポイント強度を持っているノードがないかを判断します。
3. トランザクションで直接参照しているノードの中で最も高いコミット・ポイント強度を持つノードか、またはそのノードのサーバーの中でより高いコミット・ポイント強度を持つもののどちらかが、コミット・ポイント・サイトになります。
4. 最終的なコミット・ポイント・サイトが決定した後、グローバル・コーディネータは、トランザクションに参加しているすべてのノードに準備応答を送ります。

図 32-4 は、各ノードのコミット・ポイント強度（かっこ内の値）を示したセッション・ツリーの例です。また、コミット・ポイント・サイトとして選択されたノードも示しています。

図 32-4 コミット・ポイント強度とコミット・ポイント・サイトの決定



コミット・ポイント・サイトを決定するときは、次の条件が適用されます。

- 読取り専用ノードは、コミット・ポイント・サイトにはなれません。
- グローバル・コーディネータが直接参照している複数のノードが同じコミット・ポイント強度を持っている場合、データベースはそれらのうちの 1 つをコミット・ポイント・サイトとして指定します。
- 分散トランザクションがロールバックで終了する場合は、準備フェーズとコミット・フェーズは不要です。したがって、データベースはコミット・ポイント・サイトを決定し

ません。そのかわりに、グローバル・コーディネータは ROLLBACK 文をすべてのノードに送り、分散トランザクションの処理を終了します。

図 32-4 のように、コミット・ポイント・サイトとグローバル・コーディネータがセッション・ツリーの異なるノードになることもあります。各ノードのコミット・ポイント強度は、最初に接続が確立されたときにコーディネータに渡されます。コーディネータは、2 フェーズ・コミット時のコミット・ポイント・サイトを効率的に選択するために、自身が直接やり取りしている各ノードのコミット・ポイント強度を保持しています。そのため、コミットが発生するたびにコーディネータとノード間でコミット・ポイント強度を交換する必要がありません。

関連項目：

- ノードのコミット・ポイント強度の設定方法の詳細は、33-2 ページの「ノードのコミット・ポイント強度の指定」を参照してください。
- 初期化パラメータ COMMIT_POINT_STRENGTH の詳細は、『Oracle Database リファレンス』を参照してください。

2 フェーズ・コミット・メカニズム

ローカル・データベースのトランザクションとは異なり、分散トランザクションには複数のデータベースでのデータの変更が伴います。そのため、データベースは、トランザクションの変更のコミットまたはロールバックを自己完結単位として調整する必要があり、分散トランザクションの処理はより複雑になります。言い換えれば、トランザクション全体がコミットするか、またはトランザクション全体がロールバックするかのどちらかの結果になります。

データベースは、**2 フェーズ・コミット・メカニズム**を使用することで、分散トランザクションにおけるデータの整合性を保証します。**準備フェーズ**では、トランザクション内の開始ノードが他の参加ノードに対して、トランザクションをコミットまたはロールバックすることを確約するように要求します。**コミット・フェーズ**では、開始ノードがすべての参加ノードに対して、トランザクションをコミットするように要求します。この結果が達成できない場合、すべてのノードはロールバックするように要求されます。

分散トランザクションに参加しているすべてのノードは、必ず同じ動作を実行します。つまり、ノードすべてがトランザクションをコミットするか、またはノードすべてがトランザクションをロールバックします。データベースは、分散トランザクションのコミットまたはロールバックを自動的に制御および監視しており、2 フェーズ・コミット・メカニズムを使用して**グローバル・データベース**（トランザクションに参加しているデータベースの集まり）の整合性を維持します。このメカニズムは完全に透過的であり、ユーザーやアプリケーション開発者の側でプログラミングを行う必要はまったくありません。

コミット・メカニズムは、次の各フェーズで構成されています。データベースは、ユーザーが分散トランザクションをコミットすると、必ずこれらのフェーズを自動的に実行します。

フェーズ	説明
準備フェーズ	**グローバル・コーディネータ**と呼ばれる開始ノードは、コミット・ポイント・サイト以外の参加ノードに対して、たとえ障害が起きた場合でもトランザクションをコミットまたはロールバックすることを確約するように要求します。準備ができないノードがある場合、トランザクションはロールバックされます。
コミット・フェーズ	すべての参加ノードが準備完了の応答をコーディネータに伝えると、コーディネータは、コミット・ポイント・サイトにコミットを要求します。コミット・ポイント・サイトのコミット後、コーディネータは、トランザクションをコミットするように他のすべてのノードに要求します。
情報消去フェーズ	グローバル・コーディネータは、トランザクションに関する情報を消去します。

この項の内容は、次のとおりです。

準備フェーズ

準備フェーズは、分散トランザクションのコミットにおける 1 番目のフェーズです。このフェーズでは、データベースがトランザクションを実際にコミットまたはロールバックすることはありません。ここでは、分散トランザクションで参照されているすべてのノード（32-6 ページの「コミット・ポイント・サイト」で説明されているコミット・ポイント・サイトを除く）がコミットを準備するように指示されます。ノードは、準備を完了するために次の処理を実行します。

- REDO ログの情報を記録し、それ以降障害が発生してもトランザクションをコミットまたはロールバックできるようにします。
- 変更済みの表に分散ロックを設定し、読取りを防ぎます。

各ノードは、コミットの準備が完了したという応答をグローバル・コーディネータに伝えることによって、その後トランザクションをコミットまたはロールバックすることを確約します。しかし、トランザクションをコミットまたはロールバックするという決定を各ノードが一方的に下すわけではありません。この確約の意味は、この時点でインスタンス障害が発生した場合、ノードはオンライン・ログの REDO レコードを使用してデータベースをリカバリし、準備フェーズに戻すことができるということです。

注意： ノードの準備完了後に発行した問合せは、すべてのフェーズが完了するまで、関連するロック済みデータにアクセスできません。この時間は、障害が発生しないかぎり問題にはなりません（33-7 ページの「インダウト・トランザクションの処理方法の決定」を参照）。

準備フェーズでの応答のタイプ

準備を指示されたノードは、次の方法で応答できます。

応答	意味
準備完了	ノードのデータの変更が分散トランザクション内の文によって完了しており、ノードの準備が正常に完了しています。
読取り専用	ノードで変更されるデータがない（問合せのみ）か、または変更できないので、準備は不要です。
異常終了	ノードが正常に準備できません。

準備レスポンス ノードの準備が正常に完了すると、ノードは**準備完了メッセージ**を発行します。このメッセージは、ノードの変更レコードがオンライン・ログに格納されており、ノードでコミットまたはロールバックのどちらかを実行できる準備が整っていることを示します。また、このメッセージによって、トランザクションに対して保持されているロックが障害発生時にも残ることが保証されます。

読取り専用応答 ノードが準備を要求されたときに、データベースにアクセスする SQL 文がノードのデータを変更しない場合、ノードは**読取り専用メッセージ**で応答します。このメッセージは、ノードがコミット・フェーズに参加しないことを示します。

分散トランザクションの全部または一部が読取り専用になるケースとして、次の 3 つがあります。

ケース	条件	結果
一部読取り専用	次のいずれかが発生した場合 ■ 1 つ以上のノードで問合せのみが発行された。 ■ データが変更されない。 ■ トリガーの起動または制約違反のために変更がロールバックされた。	読取り専用ノードは、準備を要求されたときに自身のステータスを認識します。読取り専用ノードは、読取り専用応答をローカル・コーディネータに伝えます。この結果、データベースは読取り専用ノードを以降の処理から除外するので、コミット・フェーズがより高速に完了します。
準備フェーズでの完全な読取り専用	次のすべてが発生した場合 ■ データが変更されない。 ■ トランザクションが SET TRANSACTION READ ONLY 文で始まっていない。	準備フェーズ時にすべてのノードが読取り専用であることを認識するので、コミット・フェーズは不要になります。グローバル・コーディネータにはすべてのノードが読取り専用かどうかわからないため、グローバル・コーディネータは引き続き準備フェーズを実行する必要があります。
2 フェーズ・コミットなしの完全な読取り専用	次のすべてが発生した場合 ■ データが変更されない。 ■ トランザクションが SET TRANSACTION READ ONLY 文で始まっている。	このトランザクションでは問合せのみが許可されるため、グローバル・コーディネータは、2 フェーズ・コミットを実行する必要がありません。また、システム変更番号（SCN）の調整がノード間でグローバルに行われるため、他のトランザクションによる変更によってグローバル・トランザクション・レベルの読込み一貫性が損なわれることはありません。このトランザクションでは、UNDO セグメントは使用されません。

分散トランザクションが読取り専用に設定されている場合、そのトランザクションで UNDO セグメントは使用されません。多数のユーザーがデータベースに接続していて、ユーザーのトランザクションが READ ONLY に設定されていない場合は、それらのトランザクションが問合せを実行するのみであっても、UNDO 領域が割り当てられます。

異常終了時のエラー ノードは、正常に準備できないときに次の処理を実行します。

1. トランザクションが現在保持しているリソースを解放し、トランザクションのローカル部分をロールバックします。
2. 分散トランザクション内で自身を参照しているノードに対し、異常終了メッセージで応答します。

これらの処理は、分散トランザクションに関係している他のノードに伝播します。これにより、他のノードはトランザクションをロールバックできるので、グローバル・データベースのデータの整合性が保証されます。この応答によって、「トランザクションに関係しているすべてのノードが同じ論理時間ですべてコミットするかまたはすべてロールバックする」という分散トランザクションの基本原則が守られます。

準備フェーズの手順

準備フェーズを完了するために、コミット・ポイント・サイトを除く各ノードは次の手順を実行します。

1. ノードは、自身の**子**（以降参照する各ノード）に対して、コミットの準備をするように要求します。
2. ノードは、トランザクションによって自分自身のデータまたは子のデータが変更されるかどうかをチェックします。データが変更されない場合、ノードは残りの手順を省略し、読取り専用応答を返します（32-9 ページの「読取り専用応答」を参照）。
3. データが変更される場合、ノードはトランザクションのコミットに必要なリソースを割り当てます。
4. ノードは、トランザクションによる変更に対応する REDO レコードを REDO ログに保存します。
5. ノードは、トランザクションに対して保持されているロックが障害発生時にも残ることを保証します。
6. ノードは、準備レスポンスを開始ノードに伝えます（32-9 ページの「準備レスポンス」を参照）。あるいは、自身またはその子のいずれかが準備の試行に失敗した場合は、異常終了応答を伝えます（32-10 ページの「異常終了時のエラー」を参照）。

これらの処理によって、ノードが後で自身のトランザクションをコミットまたはロールバックできることが保証されます。この後、準備完了ノードは、グローバル・コーディネータから `COMMIT` または `ROLLBACK` 要求を受け取るまで待機します。

各ノードの準備が完了した後、分散トランザクションは**インダウト**と呼ばれる状態になります（32-12 ページの「インダウト・トランザクション」を参照）。分散トランザクションは、すべての変更がコミットまたはロールバックされるまで、インダウト状態のままです。

コミット・フェーズ

コミット・フェーズは、分散トランザクションのコミットにおける 2 番目のフェーズです。このフェーズになる前に、分散トランザクションで参照されるコミット・ポイント・サイト以外のすべてのノードが準備を完了していること、つまり、コミット・ポイント・サイト以外のすべてのノードがトランザクションのコミットに必要なリソースを確保していることが保証されます。

コミット・フェーズの手順

コミット・フェーズは次の手順で構成されています。

1. グローバル・コーディネータは、コミット・ポイント・サイトにコミットを指示します。
2. コミット・ポイント・サイトは、コミットを実行します。
3. コミット・ポイント・サイトは、コミットが完了したことをグローバル・コーディネータに伝えます。
4. グローバル・コーディネータおよびローカル・コーディネータは、すべてのノードに対してトランザクションをコミットするように指示するメッセージを送ります。
5. 各ノードで、データベースは分散トランザクションのローカル部分をコミットし、ロックを解放します。
6. 各ノードで、データベースは、トランザクションのコミットが完了したことを示す追加の REDO エントリをローカル REDO ログに記録します。
7. 各参加ノードは、自身のコミットが完了したことをグローバル・コーディネータに伝えます。

コミット・フェーズが完了するときは、分散システムの全ノードのデータについて一貫性が保たれています。

グローバル・データベースの一貫性の保証

コミットされた各トランザクションには、そのトランザクション内部の SQL 文によって行われた変更を一意に識別するための SCN が対応付けられます。SCN は、データベースのコミット済バージョンを一意に識別する内部的なタイムスタンプの役割を持ちます。

分散システムでは、次の処理がすべて発生したときに、通信中のノードの SCN が調整されます。

- 1 つ以上のデータベース・リンクによって表されるパスを使用した接続の確立
- 分散 SQL 文の実行
- 分散トランザクションのコミット

特に、分散システムのノード間で SCN が調整されることにより、文とトランザクションの両方のレベルでグローバルな読込み一貫性が保証されるという利点があります。必要であれば、グローバルな時間ベースのリカバリを実行することもできます。

準備フェーズ時に、データベースは、トランザクションに関係しているすべてのノードで最も高い SCN を判断します。次に、コミット・ポイント・サイトにおいて最も高い SCN でトランザクションをコミットします。さらに、すべての準備完了ノードに対して、コミットの指示とともにコミット SCN を送ります。

> **関連項目：** 読込み一貫性におけるタイム・ラグ問題の管理の詳細は、33-22 ページの「読込み一貫性の管理」を参照してください。

情報消去フェーズ

参加ノードが自身のコミットの完了をコミット・ポイント・サイトに通知した後、コミット・ポイント・サイトは、トランザクションに関する情報を消去できます。次の手順が発生します。

1. すべてのノードのコミットが完了したことをグローバル・コーディネータから通知された後、コミット・ポイント・サイトは、このトランザクションに関するステータス情報を消去します。
2. コミット・ポイント・サイトは、ステータス情報を消去したことをグローバル・コーディネータに伝えます。
3. グローバル・コーディネータは、トランザクションに関する自分自身の情報を消去します。

インダウト・トランザクション

2 フェーズ・コミット・メカニズムは、すべてのノードがコミットされるかまたはロールバックを一斉に実行することを保証します。システムやネットワークのエラーのために、3 つのフェーズのいずれかが失敗した場合はどうなるのでしょうか。この場合、トランザクションはインダウトになります。

分散トランザクションは、次の要因によってインダウトになる可能性があります。

- Oracle Database ソフトウェアを実行しているサーバー・マシンがクラッシュした。
- 分散処理に関係している複数の Oracle Database 間のネットワーク接続が切断された。
- 未処理のソフトウェア・エラーが発生した。

マシン、ネットワークまたはソフトウェアの問題が解決されると、RECO プロセスによってインダウト・トランザクションが自動的に解決されます。RECO がトランザクションを解決できるまで、データは読取りおよび書込みの両方についてロックされます。読取りをブロックするのは、データベースが問合せに対してどのバージョンのデータを表示すればよいかを判断できないためです。

この項の内容は、次のとおりです。

- インダウト・トランザクションの自動解決
- インダウト・トランザクションの手動解決
- インダウト・トランザクションの SCN の関連性

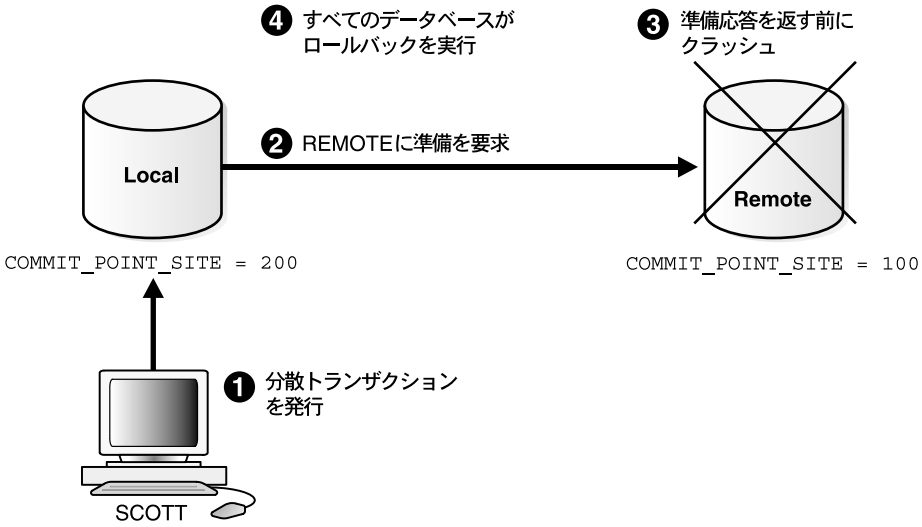
インダウト・トランザクションの自動解決

データベースでは、多くの場合、インダウト・トランザクションは自動的に解決されます。たとえば、次の使用例において local と remote の 2 つのノードがあるとします。ローカル・ノードはコミット・ポイント・サイトです。ユーザー scott は、local に接続して local と remote を更新する分散トランザクションを実行し、コミットします。

準備フェーズ中の障害

図 32-5 は、分散トランザクションの準備フェーズ中に障害が発生したときの一連のイベントを示しています。

図 32-5　準備フェーズ中の障害



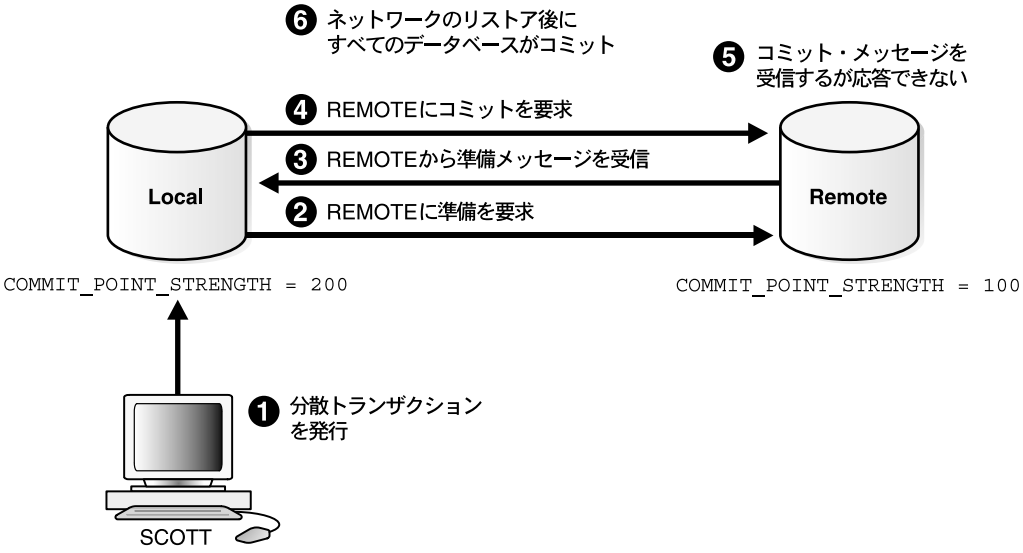
次の手順が発生します。

1. ユーザー SCOTT は Local に接続して分散トランザクションを実行します。
2. グローバル・コーディネータ（この例ではコミット・ポイント・サイトを兼務）は、コミット・ポイント・サイト以外のすべてのデータベースに対して、コミットまたはロールバックの指示があったときにその動作を実行することを確約するように要求します。
3. remote データベースが、local に準備応答を発行する前にクラッシュします。
4. トランザクションは、リモート・サイトのリストア時に、RECO プロセスによって各データベースで最終的にロールバックされます。

コミット・フェーズ中の障害

図 32-6 は、分散トランザクションのコミット・フェーズ中に障害が発生したときの一連のイベントを示しています。

図 32-6　コミット・フェーズ中の障害



次の手順が発生します。

1. ユーザー Scott は local に接続して分散トランザクションを実行します。
2. グローバル・コーディネータ（この場合はコミット・ポイント・サイトを兼務）は、コミット・ポイント・サイト以外のすべてのデータベースに対して、コミットまたはロールバックの指示があったときにその動作を実行することを確約するように要求します。
3. コミット・ポイント・サイトは、コミットの確約を示す準備完了メッセージを remote から受け取ります。
4. コミット・ポイント・サイトはトランザクションをローカルにコミットし、それからコミット・メッセージを remote に送ってコミットを要求します。
5. remote データベースはコミット・メッセージを受け取りましたが、ネットワーク障害のために応答できません。
6. トランザクションは、ネットワークがリストアされた後、RECO プロセスによってリモート・データベースで最終的にコミットされます。

> **関連項目：** 障害状況の説明と、2 フェーズ・コミット中に障害が発生した場合のデータベースによる解決方法の詳細は、33-7 ページの「インダウト・トランザクションの処理方法の決定」を参照してください。

インダウト・トランザクションの手動解決

次の場合のみ、インダウト・トランザクションを手動で解決する必要があります。

- インダウト・トランザクションが重要なデータまたは UNDO セグメントをロックしている場合
- マシン、ネットワークまたはソフトウェアの障害の原因をすぐに修復できない場合

インダウト・トランザクションの解決が複雑になる場合があります。その場合は、次の手順を実行する必要があります。

- インダウト・トランザクションのトランザクション識別番号を特定します。
- `DBA_2PC_PENDING` ビューおよび `DBA_2PC_NEIGHBORS` ビューを問い合せ、トランザクションに関係しているデータベースのコミットが完了しているかどうかを確認します。
- 必要であれば、`COMMIT FORCE` 文を使用してコミットを強制実行するか、または `ROLLBACK FORCE` 文を使用してロールバックを強制実行します。

> **関連項目：** インダウト・トランザクションの解決方法の詳細は、次の項を参照してください。
>
> - 33-7 ページ「インダウト・トランザクションの処理方法の決定」
> - 33-9 ページ「インダウト・トランザクションの手動上書き」

インダウト・トランザクションの SCN の関連性

SCN は、コミット済みバージョンのデータベースの内部的なタイムスタンプです。Oracle Database サーバーは、SCN クロック値を使用することでトランザクションの一貫性を保証します。たとえば、ユーザーがトランザクションをコミットするとき、データベースはそのコミットの SCN を REDO ログに記録します。

データベースは、SCN を使用して、異なるデータベース間での分散トランザクションを調整します。たとえば、データベースは、次のときに SCN を使用します。

1. アプリケーションがデータベース・リンクを使用して接続を確立するとき
2. 分散トランザクションが、それに関係しているすべてのデータベースの中で最も高いグローバル SCN でコミットされるとき
3. コミット・グローバル SCN が、トランザクションに関係しているすべてのデータベースに送られるとき

SCN は、トランザクションが失敗した場合も含め、トランザクションの同期化されたコミット・タイムスタンプとして機能するため、分散トランザクションにとって非常に重要な存在です。トランザクションがインダウトになった場合、管理者はこの SCN を使用して、グローバル・データベースへの変更を調整できます。トランザクション・コミットのグローバル SCN は、分散リカバリの実行時など、後でトランザクションの識別に使用することもできます。

分散トランザクション処理 : 事例

この使用例では、ある会社が sales.acme.com および warehouse.acme.com という異なる Oracle Database サーバーを持っています。ユーザーが売上レコードを sales データベースに挿入すると、対応付けられたレコードが warehouse データベースで更新されます。

この分散処理の事例では、次のことを示します。

- セッション・ツリーの定義。
- コミット・ポイント・サイトがどのように決定されるか。
- 準備メッセージがいつ送られるか。
- トランザクションがいつ実際にコミットされるか。
- トランザクションに関するどのような情報がローカルに格納されるか。

第 1 段階 : クライアント・アプリケーションによる DML 文の発行

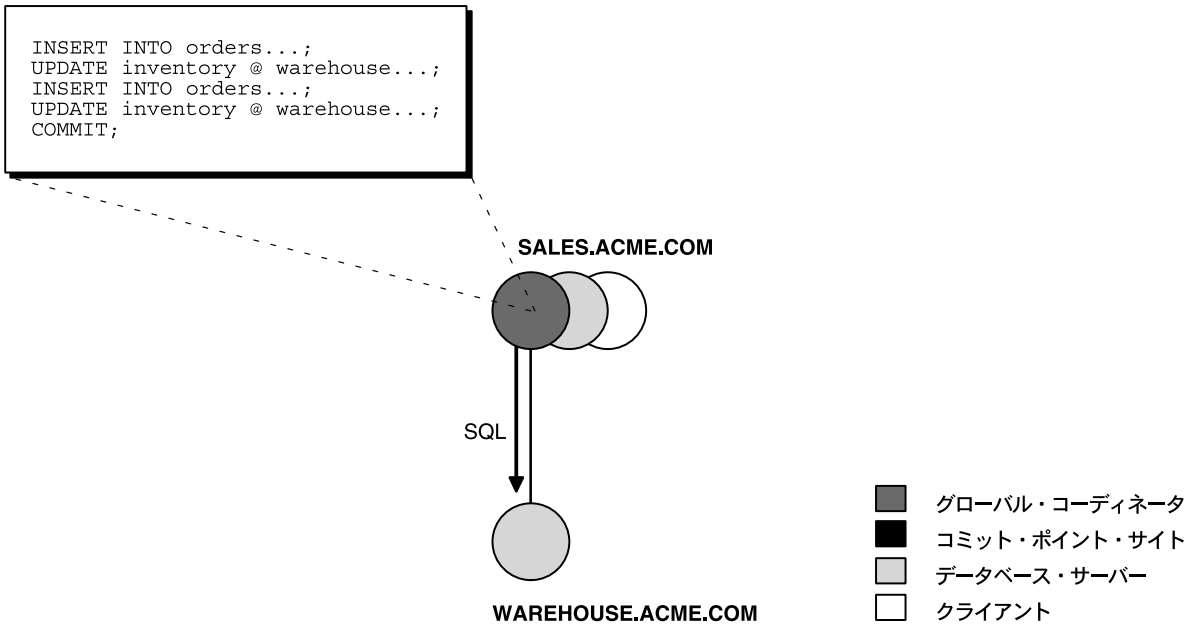
営業部門で、営業担当が SQL*Plus を使用して売上注文を入力し、コミットします。アプリケーションは次のような SQL 文を発行し、sales データベースに注文を入力して、warehouse データベースの inventory 表を更新します。

```
CONNECT scott@sales.acme.com ...;
INSERT INTO orders ...;
UPDATE inventory@warehouse.acme.com ...;
INSERT INTO orders ...;
UPDATE inventory@warehouse.acme.com ...;
COMMIT;
```

これらの SQL 文は単一の分散トランザクションの一部であり、発行されるすべての SQL 文がユニットとして成功または失敗することが保証されています。文をユニットとして扱うことにより、注文が設定されているにもかかわらず、注文を反映するように在庫が更新されていないという事態を防ぐことができます。実際は、トランザクションによってグローバル・データベースのデータの一貫性が保証されています。

トランザクションの各 SQL 文が実行されると、図 32-7 のようにセッション・ツリーが定義されます。

図 32-7　セッション・ツリーの定義



トランザクションの次の点に注意してください。

- トランザクションを開始するのは、sales データベースで実行されている注文入力アプリケーションです。したがって、分散トランザクションのグローバル・コーディネータは、sales.acme.com になります。
- 注文入力アプリケーションは、新しい売上レコードを sales データベースに挿入し、warehouse データベースの inventory 表を更新します。したがって、ノード sales.acme.com と warehouse.acme.com はどちらもデータベース・サーバーになります。
- sales.acme.com は inventory 表を更新するため、warehouse.acme.com のクライアントになります。

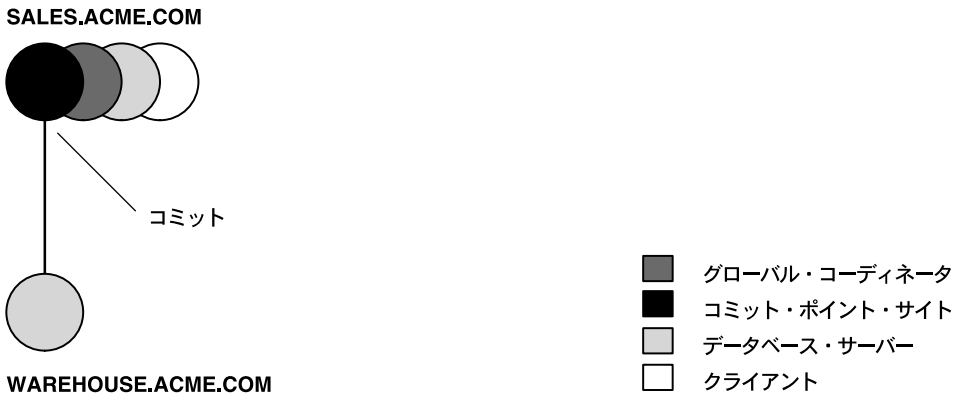
この段階では、この分散トランザクションのセッション・ツリーの定義が完了します。ツリー内の各ノードは、必要なデータ・ロックを獲得して、ローカル・データを参照する SQL 文を実行します。これらのロックは、SQL 文の実行が完了した後も、2 フェーズ・コミットが完了するまで保持されます。

第 2 段階 : Oracle Database によるコミット・ポイント・サイトの判別

データベースは、COMMIT 文の直後にコミット・ポイント・サイトを判別します。図 32-8 のように、グローバル・コーディネータの sales.acme.com がコミット・ポイント・サイトとして判別されます。

関連項目 : コミット・ポイント・サイトの判別の詳細は、32-7 ページの「コミット・ポイント強度」を参照してください。

図 32-8 コミット・ポイント・サイトの判別



第 3 段階 : グローバル・コーディネータによる準備応答の送信

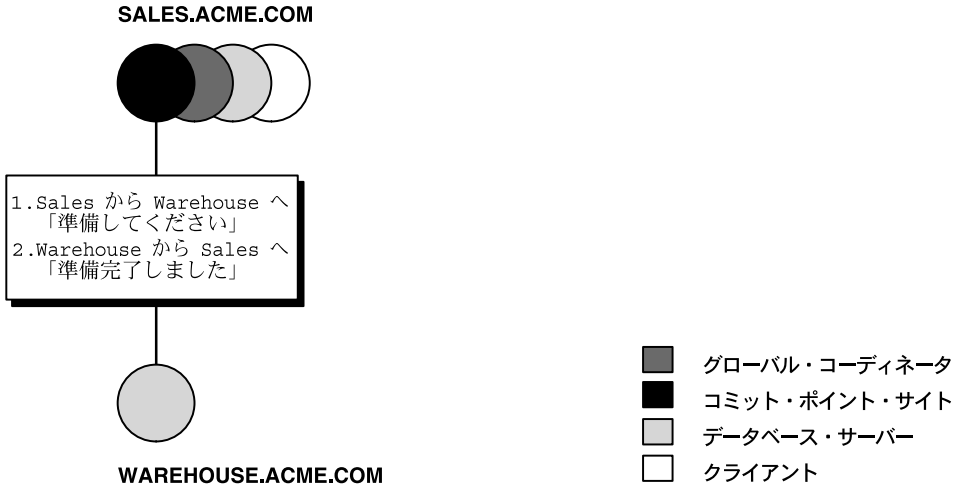
準備段階では、次の手順が実行されます。

1. データベースがコミット・ポイント・サイトを判別した後、グローバル・コーディネータは、コミット・ポイント・サイトを除くセッション・ツリーの直接参照先のノードすべてに準備メッセージを送ります。この例では、準備を要求されるノードは warehouse.acme.com のみです。
2. ノード warehouse.acme.com は準備を試みます。トランザクション内のローカルに独立した部分をコミットし、自身のローカル REDO ログにコミット情報を記録できることをノードが保証できれば、そのノードは正常に準備できます。この例では、sales.acme.com がコミット・ポイント・サイトなので、warehouse.acme.com のみが準備メッセージを受け取ります。
3. ノード warehouse.acme.com は、準備完了メッセージで sales.acme.com に応答します。

各ノードが準備を終えると、準備を要求した側のノードに応答メッセージが送り返されます。応答に応じて、次のいずれかの動作が発生します。

- 準備を要求されたノードのうち、異常終了メッセージでグローバル・コーディネータに応答したノードが 1 つでもある場合、グローバル・コーディネータはトランザクションをロールバックするようにすべてのノードに指示し、操作は完了します。
- 準備を要求されたノードのすべてが準備完了または読取り専用のメッセージでグローバル・コーディネータに応答した場合、つまり、すべてのノードの準備が正常に完了した場合、グローバル・コーディネータはトランザクションをコミットするようにコミット・ポイント・サイトに要求します。

図 32-9 準備メッセージの送信と確認



第 4 段階 : コミット・ポイント・サイトによるコミット

コミット・ポイント・サイトによるトランザクションのコミットでは、次の手順が実行されます。

1. ノード sales.acme.com は、warehouse.acme.com の準備が完了しているという確認を受け取り、トランザクションをコミットするようにコミット・ポイント・サイトに指示します。
2. この時点で、コミット・ポイント・サイトはトランザクションをローカルにコミットし、この操作を自身のローカル REDO ログに記録します。

warehouse.acme.com がまだコミットを完了していない場合でも、このトランザクションの結果は事前に決まっています。つまり、対象となるノードのコミットが遅れた場合でも、トランザクションはすべてのノードでコミットされます。

第 5 段階 : コミット・ポイント・サイトによるグローバル・コーディネータへのコミットの通知

この段階では、次の手順が実行されます。

1. コミット・ポイント・サイトは、トランザクションが完了したことをグローバル・コーディネータに伝えます。この例では、コミット・ポイント・サイトとグローバル・コーディネータが同じノードなので、必要な操作はありません。コミット・ポイント・サイトでは、トランザクションがコミットされたことがわかっています。これは、トランザクションがコミットされたことが自身のオンライン・ログに記録されているためです。
2. グローバル・コーディネータは、分散トランザクションに関係している他のすべてのノードでトランザクションがコミットされたことを確認します。

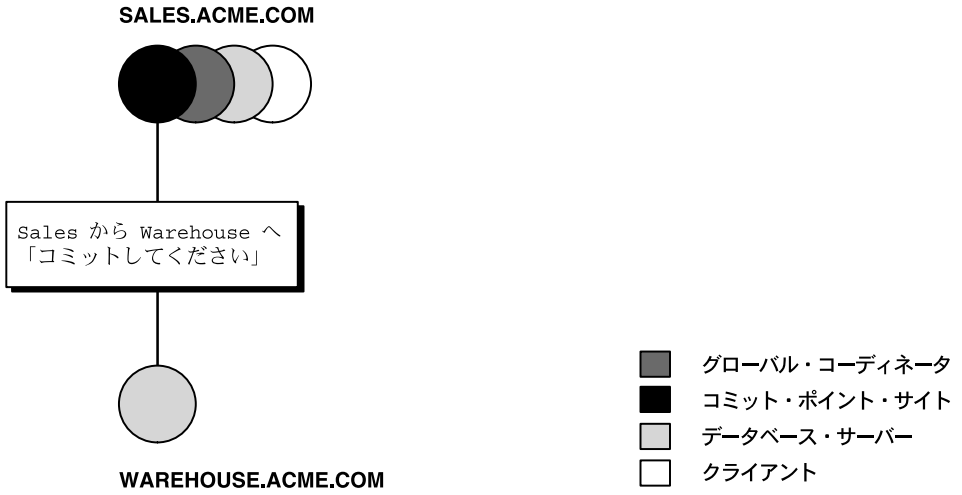
第 6 段階 : グローバルおよびローカル・コーディネータによる全ノードへのコミットの要求

トランザクション内のすべてのノードによるトランザクションのコミットでは、次の手順が実行されます。

1. コミット・ポイント・サイトでのコミットがグローバル・コーディネータに通知された後、グローバル・コーディネータは、直接参照している他のすべてのノードに対してコミットを指示します。
2. コミットの指示を受けたローカル・コーディネータは、次に自身のサーバーに対してコミットを指示し、以下同様にコミットの指示が伝播されます。
3. グローバル・コーディネータを含む各ノードは、トランザクションをコミットし、該当する REDO ログ・エントリをローカルに記録します。各ノードのコミットが完了すると、そのトランザクションのためにローカルに保持されていたリソース・ロックが解放されます。

図 32-10 では、コミット・ポイント・サイトとグローバル・コーディネータを兼務している sales.acme.com が、トランザクションのローカルのコミットをすでに完了しています。この時点で sales は warehouse.acme.com に対してトランザクションのコミットを指示します。

図 32-10 ノードへのコミットの指示



第 7 段階 : グローバル・コーディネータとコミット・ポイント・サイトによるコミットの完了

トランザクションのコミットの完了は、次の手順で実行されます。

1. すべての参照先ノードとグローバル・コーディネータがトランザクションのコミットを完了した後、グローバル・コーディネータはこれをコミット・ポイント・サイトに通知します。
2. このメッセージを待っていたコミット・ポイント・サイトは、この分散トランザクションに関するステータス情報を消去します。
3. コミット・ポイント・サイトは、消去が完了したことをグローバル・コーディネータに伝えます。つまり、コミット・ポイント・サイトは、分散トランザクションのコミットに関する情報をまったく保持しません。2 フェーズ・コミットに関係しているすべてのノードでトランザクションのコミットが正常に完了すると、それらのノードで今後ノード自身のステータスを判断する必要はありません。したがって、このような処理が許可されます。
4. グローバル・コーディネータは、トランザクション自体に関する情報を消去することでトランザクションを完了します。

COMMIT フェーズの完了後、分散トランザクションそのものが完了します。これまで説明した手順は、1 秒以内に自動的に実行されます。

33

分散トランザクションの管理

この章の内容は次のとおりです。

ノードのコミット・ポイント強度の指定

分散トランザクションでどのノードが最初にコミットされるかは、最も高いコミット・ポイント強度を持つデータベースによって決まります。各ノードのコミット・ポイント強度を指定するときは、準備フェーズまたはコミット・フェーズ中に障害が発生した場合に最も重要なサーバーがブロックされないようにしてください。ノードのコミット・ポイント強度は、COMMIT_POINT_STRENGTH 初期化パラメータによって指定します。

デフォルト値は、オペレーティング・システムによって異なります。値の範囲は、0（ゼロ）から 255 までの任意の整数です。たとえば、データベースのコミット・ポイント強度を 200 に設定するには、そのデータベースの初期化パラメータ・ファイルに次の行を追加します。

```
COMMIT_POINT_STRENGTH = 200
```

コミット・ポイント強度は、分散トランザクションでコミット・ポイント・サイトを決定する際にのみ使用されます。

データベースのコミット・ポイント強度を設定するときは、次の点を考慮してください。

- コミット・ポイント・サイトにはトランザクションのステータスに関する情報が格納されます。他のノードがトランザクションのステータスに関する情報を必要とする場合に備えて、コミット・ポイント・サイトには、信頼性が低下したり、使用できなくなったりする頻度の高いノードを使用しないでください。
- データベースのコミット・ポイント強度は、データベース内の重要な共有データの量に比例するように設定してください。たとえば、メインフレーム・コンピュータのデータベースは、一般に PC のデータベースよりも多くのデータをユーザー間で共有しています。したがって、メインフレームのコミット・ポイント強度は、PC のコミット・ポイント強度よりも高い値に設定します。

> **関連項目：** コミット・ポイントの概念の詳細は、32-6 ページの「コミット・ポイント・サイト」を参照してください。

トランザクションの命名

トランザクションの命名が可能です。この機能は、特定の分散トランザクションを識別する際に便利であり、同じ目的の COMMIT COMMENT 文にかわるものです。

トランザクションに命名するには SET TRANSACTION...NAME 文を使用します。次に例を示します。

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
     NAME 'update inventory checkpoint 0';
```

この例は、SERIALIZABLE に等しい分離レベルを持つ新しいトランザクションをユーザーが開始し、それに 'update inventory checkpoint 0' という名前を付けたことを示しています。

分散トランザクションでは、トランザクションがコミットされるときに、名前が参加サイトに送られます。トランザクション名が存在する場合は、COMMIT COMMENT があっても無視されます。

トランザクション名は、V$TRANSACTION ビューの NAME 列に表示され、トランザクションがコミットされると、DBA_2PC_PENDING ビューの TRAN_COMMENT フィールドに表示されます。

分散トランザクション情報の表示

各データベースのデータ・ディクショナリには、オープンしているすべての分散トランザクションに関する情報が格納されています。データ・ディクショナリの表とビューを使用すると、トランザクションに関する情報を取得できます。この項の内容は、次のとおりです。

- 準備完了トランザクションの ID 番号と状態の判断
- インダウト・トランザクションのセッション・ツリーのトレース

準備完了トランザクションの ID 番号と状態の判断

次のビューには、ローカル・データベースで定義され、データ・ディクショナリに格納されているデータベース・リンクが表示されます。

ビュー	用途
DBA_2PC_PENDING	インダウト分散トランザクションがすべてリストされます。このビューには、インダウト・トランザクションが移入されるまで何もデータが入っていません。トランザクションが解決された後、ビューはパージされます。

このビューは、特定のトランザクション ID のグローバル・コミット番号を判断するために使用します。このグローバル・コミット番号は、インダウト・トランザクションを手動で解決するときに使用できます。

関連性の最も高い列を次の表に示します（ビューのすべての列の詳細は、『Oracle Database リファレンス』を参照してください）。

表 33-1 DBA_2PC_PENDING

列	説明
LOCAL_TRAN_ID	「*integer.integer.integer*」という書式のローカル・トランザクション識別子。 **注意：**接続の LOCAL_TRAN_ID と GLOBAL_TRAN_ID が同じ場合、そのノードはトランザクションのグローバル・コーディネータです。
GLOBAL_TRAN_ID	「*global_db_name.db_hex_id.local_tran_id*」という書式のグローバル・データベース識別子。ここで、*db_hex_id* は、データベースを一意に識別するために使用される 8 文字の 16 進値です。この共通のトランザクション ID は、1 つの分散トランザクションに関係するすべてのノードで同じです。 **注意：**接続の LOCAL_TRAN_ID と GLOBAL_TRAN_ID が同じ場合、そのノードはトランザクションのグローバル・コーディネータです。

表 33-1 DBA_2PC_PENDING（続き）

列	説明
STATE	STATE に可能な値は、次のとおりです。 ■ Collecting 通常、このカテゴリは、グローバル・コーディネータまたはローカル・コーディネータにのみ適用されます。ノードは現在他のデータベース・サーバーから情報を収集中であり、その後ノード自身が準備できるかどうかが判断されます。 ■ 準備完了 ノードは準備を完了していますが、そのことをローカル・コーディネータに準備完了メッセージで通知しているかどうかは不明です。しかし、コミット要求はまだ受け取っていません。ノードは準備完了状態にあり、トランザクションのコミットに必要なローカル・リソースのロックをすべて保持しています。 ■ Committed ノード（任意のタイプ）はトランザクションのコミットを完了していますが、トランザクションに関係している他のノードが同じように完了していない可能性があります。つまり、トランザクションは 1 つ以上のノードでまだ保留しています。 ■ Forced Commit ペンディング・トランザクションは、データベース管理者（DBA）の判断で強制的にコミットできます。このエントリは、ローカル・ノードでトランザクションが手動でコミットされた場合に表示されます。 ■ Forced termination (rollback) ペンディング・トランザクションは、DBA の判断で強制的にロールバックできます。このエントリは、ローカル・ノードでこのトランザクションが手動でロールバックされた場合に表示されます。
MIXED	YES は、トランザクションの一部があるノードではコミットされ、別のノードではロールバックされたことを示します。
TRAN_COMMENT	トランザクションがコミットされると、トランザクション・コメントまたはトランザクション名（トランザクションに命名している場合）がこの列に設定されます。
HOST	ホスト・マシンの名前。
COMMIT#	コミットされたトランザクションのグローバル・コミット番号。

DBA_2PC_PENDING の関連情報を問い合せるには、次のスクリプト pending_txn_script を実行します（出力例も含まれています）。

```
COL LOCAL_TRAN_ID FORMAT A13
COL GLOBAL_TRAN_ID FORMAT A30
COL STATE FORMAT A8
COL MIXED FORMAT A3
COL HOST FORMAT A10
COL COMMIT# FORMAT A10

SELECT LOCAL_TRAN_ID, GLOBAL_TRAN_ID, STATE, MIXED, HOST, COMMIT#
   FROM DBA_2PC_PENDING
/

SQL> @pending_txn_script

LOCAL_TRAN_ID GLOBAL_TRAN_ID                  STATE    MIX HOST       COMMIT#
------------- ------------------------------ -------- --- ---------- ----------
1.15.870      HQ.ACME.COM.ef192da4.1.15.870  commit   no  dlsun183   115499
```

この出力は、ローカル・トランザクション 1.15.870 がこのノードではコミットを完了しているものの、他の 1 つ以上のノードで保留している可能性があることを示しています。LOCAL_TRAN_ID と GLOBAL_TRAN_ID のローカル部分が同じなので、このノードはトランザクションのグローバル・コーディネータです。

インダウト・トランザクションのセッション・ツリーのトレース

次のビューには、リモート・クライアントから受信されるインダウト・トランザクションと、リモート・サーバーに送信されるインダウト・トランザクションが表示されます。

ビュー	用途
DBA_2PC_NEIGHBORS	インダウト分散トランザクションについて、リモート・クライアントから受信されるものとリモート・サーバーに送信されるものがすべてリストされます。また、ローカル・ノードがトランザクションのコミット・ポイント・サイトかどうかも示されます。 このビューには、インダウト・トランザクションが移入されるまで何もデータが入っていません。トランザクションが解決された後、ビューはパージされます。

トランザクションがインダウトのときは、セッション・ツリー内のどのノードがどのロールを実行しているのかを判断することが必要な場合があります。このビューを使用すると、次のことが判断できます。

- 特定のトランザクションのすべての受信接続と送信接続。
- ノードが特定のトランザクションのコミット・ポイント・サイトかどうか。
- ノードが特定のトランザクションのグローバル・コーディネータかどうか（ノードのローカル・トランザクション ID とグローバル・トランザクション ID が同じかどうかで判断する）。

関連性の最も高い列を次の表に示します（ビューのすべての列の詳細は、『Oracle Database リファレンス』を参照してください）。

表 33-2 DBA_2PC_NEIGHBORS

列	説明
LOCAL_TRAN_ID	「*integer.integer.integer*」という書式のローカル・トランザクション識別子。 **注意：**接続の LOCAL_TRAN_ID と GLOBAL_TRAN_ID.DBA_2PC_PENDING が同じ場合、そのノードはトランザクションのグローバル・コーディネータです。
IN_OUT	受信トランザクションの場合は IN、送信トランザクションの場合は OUT です。
DATABASE	受信トランザクションの場合は、このローカル・ノードから情報を要求したクライアント・データベースの名前です。送信トランザクションの場合は、リモート・サーバーの情報へのアクセスに使用されたデータベース・リンクの名前です。
DBUSER_OWNER	受信トランザクションの場合は、リモート・データベース・リンクによる接続で使用されるローカル・アカウントです。送信トランザクションの場合は、データベース・リンクの所有者です。

表 33-2 DBA_2PC_NEIGHBORS（続き）

列	説明
INTERFACE	C はコミット・メッセージ、N は準備完了状態を示すメッセージまたは読取り専用コミットの要求のどちらかです。 IN_OUT が OUT の場合、C は、接続のリモート側の子がコミット・ポイント・サイトであり、コミットと終了のどちらを実行するかを知っていることを意味します。N は、ローカル・ノードの準備が完了したことをリモート・ノードに通知中であることを意味します。 IN_OUT が IN の場合、C は、送信接続のリモート側のローカル・ノードまたはデータベースがコミット・ポイント・サイトであることを表します。N は、リモート・ノードの準備が完了したことをローカル・ノードに通知中であることを意味します。

DBA_2PC_PENDING の関連情報を問い合せるには、次のスクリプト neighbors_script を実行します（出力例も含まれています）。

```
COL LOCAL_TRAN_ID FORMAT A13
COL IN_OUT FORMAT A6
COL DATABASE FORMAT A25
COL DBUSER_OWNER FORMAT A15
COL INTERFACE FORMAT A3
SELECT LOCAL_TRAN_ID, IN_OUT, DATABASE, DBUSER_OWNER, INTERFACE
   FROM DBA_2PC_NEIGHBORS
/

SQL> CONNECT SYS@hq.acme.com AS SYSDBA
SQL> @neighbors_script

LOCAL_TRAN_ID IN_OUT DATABASE                  DBUSER_OWNER    INT
------------- ------ ------------------------- --------------- ---
1.15.870      out    SALES.ACME.COM            SYS             C
```

この出力は、トランザクション 1.15.870 をコミットするための送信要求をローカル・ノードがリモート・サーバー sales に送ったことを示します。sales がトランザクションをコミットしたものの、他のノードがコミットしなかった場合は、sales がコミット・ポイント・サイトであることが判明します。コミット・ポイント・サイトは、常に最初にコミットされるためです。

インダウト・トランザクションの処理方法の決定

2 フェーズ・コミットの間に障害が発生すると、トランザクションはインダウトになります。分散トランザクションがインダウトになる要因は、次のとおりです。

- Oracle Database ソフトウェアを実行しているサーバー・マシンがクラッシュした。
- 分散処理に関係している複数の Oracle Database 間のネットワーク接続が切断された。
- 未処理のソフトウェア・エラーが発生した。

> **関連項目：** インダウト・トランザクションの概念の詳細は、32-12 ページの「インダウト・トランザクション」を参照してください。

ローカルのインダウト分散トランザクションは、手動で強制的にコミットまたはロールバックできます。この操作では一貫性の問題が生じる可能性があるため、特定の条件が成り立つとき以外は実行しないでください。

この項の内容は、次のとおりです。

- 2 フェーズ・コミットに関する問題の検出
- 手動上書きを実行するかどうかの判断
- トランザクション・データの分析

2 フェーズ・コミットに関する問題の検出

分散トランザクションをコミットするユーザー・アプリケーションには、次のいずれかのエラー・メッセージによって問題が通知されます。

```
ORA-02050: トランザクション ID はロールバックされました。
           some remote dbs may be in-doubt
ORA-02053: トランザクション ID はコミットしました。
           some remote dbs may be in-doubt
ORA-02054: トランザクション ID はインダウトです
```

アプリケーションでこれらのエラーを受け取った場合は、トランザクションに関する情報を保存するようにしてください。この情報は、後で分散トランザクションの手動リカバリが必要になった場合に使用できます。

ネットワークまたはシステムの障害のために、任意のノードでインダウト分散トランザクションが 1 つ以上発生した場合でも、そのノードの管理者がなんらかの処理を実行する必要はありません。ネットワークやシステムの障害が解決した後、データベースの自動リカバリ機能によって、セッション・ツリーのすべてのノードが同じ結果になるように（つまり、すべてコミットされるかすべてロールバックされる）、すべてのインダウト・トランザクションの処理が透過的に完了します。

ただし、障害が長期にわたる場合には、トランザクションを強制的にコミットまたはロールバックすることで、ロックされたデータをすべて解放できます。アプリケーションは、この操作を想定しておく必要があります。

手動上書きを実行するかどうかの判断

次の条件が成り立つときのみ、特定のインダウト・トランザクションを手動で上書きしてください。

- インダウト・トランザクションが他のトランザクションに必要なデータをロックしている場合。この状況は、ORA-01591 エラー・メッセージによってユーザーのトランザクションが妨げられたときに起こります。
- インダウト・トランザクションによって、他のトランザクションが UNDO セグメントのエクステントを使用できなくなった場合。インダウト分散トランザクションのローカル・トランザクション ID は、その最初の部分が UNDO セグメントの ID に対応しています。この ID は、データ・ディクショナリ・ビュー DBA_2PC_PENDING にリストされます。
- 2 フェーズ・コミットの各フェーズの完了を妨げている障害を許容される時間範囲内で訂正できない場合。このような例としては、通信ネットワークやデータベースの損傷など、リカバリに長い時間を要する障害があります。

通常は、他の場所の管理者と相談した上で、インダウト分散トランザクションを強制的にローカルに処理するかどうかを決めてください。判断を誤った場合は、トレースの困難なデータベースの非一貫性が生じる可能性があります。この非一貫性は、手動で訂正する必要があります。

前述の条件が当てはまらない場合は、必ずデータベースの自動リカバリ機能によってトランザクションを完了するようにしてください。しかし、前述のいずれかの条件に当てはまる場合には、インダウト・トランザクションのローカルでの修正を検討してください。

トランザクション・データの分析

トランザクションの強制完了を決めた場合は、次の目的に留意しながら、使用可能な情報を分析します。

コミットまたはロールバックされたノードの特定

DBA_2PC_PENDING ビューを使用し、トランザクションをコミットまたはロールバックしたノードを探します。トランザクションをすでに解決しているノードが見つかった場合は、そのノードで実行された処理に従うことができます。

トランザクション・コメントの確認

DBA_2PC_PENDING の TRAN_COMMENT 列に、対象の分散トランザクションに関するなんらかの情報が与えられていないかを確認します。コメントは、COMMIT 文の COMMENT 句で指定されます。また、トランザクションに命名している場合は、トランザクションがコミットされたときに、トランザクション名が TRAN_COMMENT フィールドに設定されます。

たとえば、インダウト分散トランザクションのコメントで、トランザクションの開始元とタイプを示すことができます。

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

また、この情報をトランザクション名として提供するために、SET TRANSACTION...NAME 文を使用することも可能です。

関連項目： 33-2 ページ「トランザクションの命名」

トランザクション・アドバイスの確認

DBA_2PC_PENDING の ADVICE 列に、対象の分散トランザクションに関するなんらかの情報が与えられていないかを確認します。アプリケーションで ALTER SESSION 文の ADVISE 句を使用すれば、分散トランザクションの別々の部分を強制的にコミットまたはロールバックする際のアドバイスを規定できます。

準備フェーズ中に各ノードに送られたアドバイスは、現行トランザクション内でそのデータベースに対し最新の DML 文が実行されたときに有効なアドバイスです。

たとえば、あるノードの emp 表から別のノードの emp 表に従業員レコードを移動する分散トランザクションを考えます。次の一連の SQL 文を追加することによって、管理者が各ノードで個別にインダウト・トランザクションを強制的に完了した場合でも、トランザクションでレコードを保護できます。

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

ALTER SESSION ADVISE NOTHING;
```

与えられたアドバイスに従ってインダウト・トランザクションを手動で強制的に完了すると、最悪の場合、各ノードに従業員レコードがコピーされ、消去できなくなる可能性があります。

インダウト・トランザクションの手動上書き

COMMIT 文または ROLLBACK 文に、FORCE オプションと、コミットまたはロールバックするインダウト・トランザクションのローカル・トランザクション ID またはグローバル・トランザクション ID を示すテキスト文字列を指定します。

> **注意：** 以降のすべての例で、トランザクションはローカル・ノードでコミットまたはロールバックされます。ローカルのペンディング・トランザクション表では、このトランザクションの行の STATE 列に強制コミットまたは強制終了の値が記録されます。

この項の内容は、次のとおりです。

- インダウト・トランザクションの手動コミット
- インダウト・トランザクションの手動ロールバック

インダウト・トランザクションの手動コミット

トランザクションのコミットを試みる前に、適正な権限を持っていることを確認してください。必要な権限は次のとおりです。

トランザクションをコミットするユーザー	必要な権限
管理者	FORCE TRANSACTION
別のユーザー	FORCE ANY TRANSACTION

トランザクション ID のみを使用したコミット

次の SQL 文は、インダウト・トランザクションをコミットします。

```
COMMIT FORCE 'transaction_id';
```

変数 *transaction_id* は、DBA_2PC_PENDING データ・ディクショナリ・ビューの LOCAL_TRAN_ID 列または GLOBAL_TRAN_ID 列のどちらかで指定されているトランザクション識別子です。

たとえば、DBA_2PC_PENDING を問い合せて、分散トランザクションの LOCAL_TRAN_ID が 1:45.13 であることを確認するとします。

そして、次の SQL 文を発行し、このインダウト・トランザクションのコミットを強制実行します。

```
COMMIT FORCE '1.45.13';
```

システム変更番号（SCN）を使用したコミット

必要であれば、トランザクションを強制的にコミットするときに、トランザクションの SCN を指定することもできます。この機能を使用すると、他のノードでトランザクションがコミットされたときに割り当てられた SCN を使用して、インダウト・トランザクションをコミットできます。

これにより、障害が発生した場合でも、分散トランザクションのコミット時間の同期を保つことができます。SCN は、別のノードですでにコミットされた同じトランザクションの SCN が判別できるときのみ指定してください。

たとえば、次のグローバル・トランザクション ID を使用してトランザクションを手動でコミットするとします。

```
SALES.ACME.COM.55d1c563.1.93.29
```

まず、対象のトランザクションに関係しているリモート・データベースの DBA_2PC_PENDING ビューを問い合せます。次に、そのノードでトランザクションのコミットに使用された SCN をメモします。そして、ローカル・ノードでトランザクションをコミットするときにこの SCN を指定します。たとえば、SCN が 829381993 の場合は、次の文を発行します。

```
COMMIT FORCE 'SALES.ACME.COM.55d1c563.1.93.29', 829381993;
```

> **関連項目：** COMMIT 文の使用の詳細は、『Oracle Database SQL リファレンス』を参照してください。

インダウト・トランザクションの手動ロールバック

インダウト分散トランザクションのロールバックを試みる前に、適正な権限を持っていることを確認してください。必要な権限は次のとおりです。

トランザクションをコミットするユーザー	必要な権限
管理者	FORCE TRANSACTION
別のユーザー	FORCE ANY TRANSACTION

次の SQL 文は、インダウト・トランザクションをロールバックします。

```
ROLLBACK FORCE 'transaction_id';
```

変数 *transaction_id* は、DBA_2PC_PENDING データ・ディクショナリ・ビューの LOCAL_TRAN_ID 列または GLOBAL_TRAN_ID 列のどちらかで指定されているトランザクション識別子です。

たとえば、ローカル・トランザクション ID が 2.9.4 のインダウト・トランザクションをロールバックするには、次の文を使用します。

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

注意： インダウト・トランザクションをセーブポイントまでロールバックすることはできません。

> **関連項目：** ROLLBACK 文の使用の詳細は、『Oracle Database SQL リファレンス』を参照してください。

データ・ディクショナリからの保留行のパージ

インダウト・トランザクションは、RECO（リカバラ・プロセス）によってリカバリされる前に、DBA_2PC_PENDING.STATE 列に COLLECTING、COMMITTED または PREPARED として表示されます。COMMIT FORCE または ROLLBACK FORCE を使用してインダウト・トランザクションを強制的に完了すると、FORCED COMMIT または FORCED ROLLBACK の状態になることがあります。

自動リカバリでは通常、これらの状態にあるエントリが削除されます。唯一の例外として、リカバリ時に、トランザクション内の他のサイトと一貫性のない状態にある強制トランザクションが検出されるときがあります。この場合は、エントリが表内に残る可能性があり、DBA_2PC_PENDING の MIXED 列の値が YES になります。これらのエントリは、DBMS_TRANSACTION.PURGE_MIXED プロシージャを使用してクリーンアップできます。

リモート・データベースが永続的に失われたために自動リカバリが不可能な場合は、データベースを再作成してもリカバリではそのデータベースを識別できません。これは、再作成時に新しいデータベース ID が割り当てられるためです。この場合は、DBMS_TRANSACTION パッケージの PURGE_LOST_DB_ENTRY プロシージャを使用して、エントリをクリーンアップする必要があります。このエントリによってデータベース・リソースが保持されることはないため、エントリのクリーンアップを急ぐ必要はありません。

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

PURGE_LOST_DB_ENTRY プロシージャの実行

エントリをデータ・ディクショナリから手動で削除するには、次の構文を使用します（ここで、*trans_id* はトランザクションの識別子を表します）。

```
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('trans_id');
```

たとえば、保留中の分散トランザクション 1.44.99 をパージするには、SQL*Plus で次の文を入力します。

```
EXECUTE DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('1.44.99');
```

このプロシージャは、自動リカバリによってトランザクションを解決できないほどの大幅な再構成を行った場合にのみ実行してください。たとえば、次のような場合です。

- リモート・データベースが完全に失われた場合
- ソフトウェアを再構成した結果、2 フェーズ・コミットの機能がなくなった場合
- TPMonitor などの外部トランザクション・コーディネータからの情報が失われた場合

DBMS_TRANSACTION を使用する時期の判断

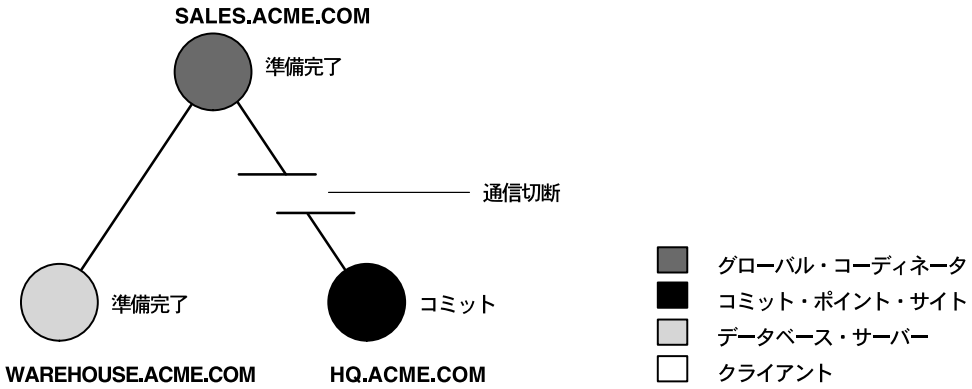
次の表は、分散トランザクションに関する各種の状態と、それに対して管理者が実行すべき処理を示したものです。

STATE 列	グローバル・トランザクションの状態	ローカル・トランザクションの状態	通常の処理	代替処理
Collecting	Rolled back	Rolled back	なし	PURGE_LOST_DB_ENTRY (自動リカバリでトランザクションを解決できない場合のみ)
Committed	Committed	Committed	なし	PURGE_LOST_DB_ENTRY (自動リカバリでトランザクションを解決できない場合のみ)
準備完了	Unknown	準備完了	なし	強制コミットまたは強制ロールバック
Forced commit	Unknown	Committed	なし	PURGE_LOST_DB_ENTRY (自動リカバリでトランザクションを解決できない場合のみ)
Forced rollback	Unknown	Rolled back	なし	PURGE_LOST_DB_ENTRY (自動リカバリでトランザクションを解決できない場合のみ)
Forced commit	Mixed	Committed	非一貫性部分を手動で削除してから PURGE_MIXED を使用する	-
Forced rollback	Mixed	Rolled back	非一貫性部分を手動で削除してから PURGE_MIXED を使用する	-

インダウト・トランザクションの手動コミット : 例

図 33-1 は、分散トランザクションのコミット中の障害を示しています。この障害例では、準備フェーズは完了しています。しかし、コミット・フェーズ時に、コミット・ポイント・サイトがトランザクションをコミットしていても、コミット・ポイント・サイトのコミット確認がグローバル・コーディネータに到達しません。インダウト・トランザクションは他のトランザクションにとっても重要なものであるため、在庫データはロックされており、アクセスできません。また、インダウト・トランザクションがコミットまたはロールバックされるまで、ロックが必ず保持されます。

図 33-1　インダウト分散トランザクションの例



次の手順に従って、インダウト・トランザクションのローカル部分を手動で強制完了できます。詳細は、以降の項を参照してください。

手順 1: ユーザーからのフィードバックの記録

手順 2: DBA_2PC_PENDING の問合せ

手順 3: ローカル・ノードでの DBA_2PC_NEIGHBORS の問合せ

手順 4: 全ノードでのデータ・ディクショナリ・ビューの問合せ

手順 5: インダウト・トランザクションのコミット

手順 6: DBA_2PC_PENDING を使用した MIXED 結果のチェック

手順 1: ユーザーからのフィードバックの記録

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

```
ORA-01591: インダウト分散トランザクション 1.21.17 がロックを保持しています。
```

この場合、1.21.17 はインダウト分散トランザクションのローカル・トランザクション ID です。どのインダウト・トランザクションを強制処理すればよいかを特定するために、問題を報告したユーザーからこの ID 番号を聞き、記録しておきます。

手順 2: DBA_2PC_PENDING の問合せ

SQL*Plus を使用して warehouse に接続した後、ローカルの DBA_2PC_PENDING データ・ディクショナリ・ビューを問い合せて、インダウト・トランザクションに関する情報を取得します。

```
CONNECT SYS@warehouse.acme.com AS SYSDBA
SELECT * FROM DBA_2PC_PENDING WHERE LOCAL_TRAN_ID = '1.21.17';
```

データベースは次の情報を返します。

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

グローバル・トランザクション ID の判断

グローバル・トランザクション ID は、分散トランザクションのすべてのノードで同一である共通のトランザクション ID です。次のような書式で記録されています。

global_database_name.hhhhhhhh.local_transaction_id

各項目の意味は次のとおりです。

- *global_database_name* は、グローバル・コーディネータのデータベース名です。
- *hhhhhhhh* は、グローバル・コーディネータの内部データベース識別子（16 進値）です。
- *local_transaction_id* は、グローバル・コーディネータで割り当てられた、対応するローカル・トランザクション ID です。

グローバル・コーディネータでは、グローバル・トランザクション ID の最後の部分とローカル・トランザクション ID が一致します。この例ではこれらの番号が一致しないので、warehouse はグローバル・コーディネータでないことがわかります。

```
LOCAL_TRAN_ID          1.21.17
GLOBAL_TRAN_ID         ... 1.93.29
```

トランザクションの状態の判断

このノードのトランザクションは、準備完了状態です。

```
STATE          prepared
```

したがって、warehouse は、自身のコーディネータからコミット要求またはロールバック要求が送られてくるのを待っています。

コメントまたはアドバイスの確認

トランザクションのコメントまたはアドバイスに、このトランザクションに関する情報が含まれている可能性があります。トランザクションに関する情報が含まれていれば、そのコメントを利用します。この例では、トランザクションのコメントに開始元とトランザクション・タイプが含まれています。

```
TRAN_COMMENT            Sales/New Order/Trans_type 10B
```

また、SET TRANSACTION...NAME 文によって、トランザクションに関する情報がトランザクション名として提供されている可能性もあります。

この情報から、トランザクションのローカル部分をコミットすべきか、またはロールバックすべきかを決める際に役立つ情報を得ることができます。インダウト・トランザクションに有益なコメントが付けられていない場合は、その他の管理作業を実行し、セッション・ツリーをトレースして、トランザクションをすでに解決しているノードを探す必要があります。

手順 3: ローカル・ノードでの DBA_2PC_NEIGHBORS の問合せ

この手順の目的は、セッション・ツリーを探索してコーディネータを見つけ、最終的にグローバル・コーディネータに到達することです。それと同時に、トランザクションをすでに解決しているコーディネータが見つかる場合もあります。トランザクションをすでに解決しているコーディネータが見つからない場合は、最終的にコミット・ポイント・サイトを探し出します。コミット・ポイント・サイトでは、常にインダウト・トランザクションが解決されています。セッション・ツリーをトレースするには、各ノードの DBA_2PC_NEIGHBORS ビューを問い合せます。

この例では、warehouse データベースでこのビューを問い合せます。

```
CONNECT SYS@warehouse.acme.com AS SYSDBA
SELECT * FROM DBA_2PC_NEIGHBORS
  WHERE LOCAL_TRAN_ID = '1.21.17'
  ORDER BY SESS#, IN_OUT;

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

データベースのロールとデータベース・リンク情報の取得

DBA_2PC_NEIGHBORS ビューは、インダウト・トランザクションに対応付けられた接続に関する情報を提供します。情報は、接続が**受信**（IN_OUT = in）か**送信**（IN_OUT = out）かによって異なります。

IN_OUT	意味	DATABASE	DBUSER_OWNER
in	このノードは別のノードのサーバーです。	このノードに接続しているクライアント・データベースの名前がリストされます。	インダウト・トランザクションに対応するデータベース・リンク接続のローカル・アカウントがリストされます。
out	このノードは他のサーバーのクライアントです。	リモート・ノードに接続しているデータベース・リンクの名前がリストされます。	インダウト・トランザクションのデータベース・リンクの所有者がリストされます。

この例では、IN_OUT 列によって、warehouse データベースが sales クライアント（DATABASE 列に示されている）のサーバーであることがわかります。

```
IN_OUT                in
DATABASE              SALES.ACME.COM
```

warehouse への接続は、swilliams アカウント（DBUSER_OWNER 列に示されている）からのデータベース・リンクを介して確立されています。

```
DBUSER_OWNER          SWILLIAMS
```

コミット・ポイント・サイトの判別

この他、INTERFACE 列によって、ローカル・ノードまたは下位ノードがコミット・ポイント・サイトなのかどうかがわかります。

```
INTERFACE             N
```

INTERFACE 列が示すように、warehouse もその子もコミット・ポイント・サイトではありません。

手順 4: 全ノードでのデータ・ディクショナリ・ビューの問合せ

この時点で、各設置ノードの管理者に連絡を取り、グローバル・トランザクション ID を使用して手順 2 および 3 を繰り返すように各管理者に依頼できます。

> **注意 :** これらのノードに別のネットワークを介して直接接続できる場合は、手順 2 および 3 を独力で実行できます。

たとえば、sales および hq で手順 2 および 3 を実行すると、次の結果が返されます。

sales でのペンディング・トランザクションの状態の確認

この段階では、sales の管理者が DBA_2PC_PENDING データ・ディクショナリ・ビューを問い合せます。

```
SQL> CONNECT SYS@sales.acme.com AS SYSDBA
SQL> SELECT * FROM DBA_2PC_PENDING
   > WHERE GLOBAL_TRAN_ID = 'SALES.ACME.COM.55d1c563.1.93.29';

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

sales でのコーディネータとコミット・ポイント・サイトの判別

次に、sales の管理者は DBA_2PC_NEIGHBORS を問い合せて、グローバル・コーディネータ、ローカル・コーディネータおよびコミット・ポイント・サイトを判別します。

```
SELECT * FROM DBA_2PC_NEIGHBORS
   WHERE GLOBAL_TRAN_ID = 'SALES.ACME.COM.55d1c563.1.93.29'
   ORDER BY SESS#, IN_OUT;
```

この問合せは、次の 3 行を返します。

- warehouse への接続
- hq への接続
- ユーザーが確立した接続

warehouse 接続の行に対応する情報を、書式を変えて示すと次のようになります。

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

hq 接続の行に対応する情報を、書式を変えて示すと次のようになります。

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

前の問合せから得られた情報によって、次のことがわかります。

- sales のローカル・トランザクション ID とグローバル・トランザクション ID が一致しているので、sales はグローバル・コーディネータです。
- このノードからは送信接続が 2 つ確立されていますが、受信接続は確立されていません。したがって、sales は別のノードのサーバーではありません。
- コミット・ポイント・サイトは、hq またはそのサーバーの 1 つです。

HQ でのペンディング・トランザクションの状態の確認

この段階では、hq の管理者が DBA_2PC_PENDING データ・ディクショナリ・ビューを問い合せます。

```
SELECT * FROM DBA_2PC_PENDING@hq.acme.com
   WHERE GLOBAL_TRAN_ID = 'SALES.ACME.COM.55d1c563.1.93.29';

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

この時点で、トランザクションを解決しているノードが見つかりました。ビューが示しているように、このノードではコミットが完了しており、コミット ID 番号が割り当てられています。

```
STATE                  COMMIT
COMMIT#                129314
```

したがって、自分のローカル・データベースでインダウト・トランザクションを強制的にコミットできます。この調査結果を他の管理者に伝えれば、他の場所での解決にも役立つ可能性があります。

手順 5: インダウト・トランザクションのコミット

sales データベースの管理者と連絡を取り、グローバル ID を使用してインダウト・トランザクションを手動でコミットしてもらうように依頼します。

```
SQL> CONNECT SYS@sales.acme.com AS SYSDBA
SQL> COMMIT FORCE 'SALES.ACME.COM.55d1c563.1.93.29';
```

同時に、warehouse データベースの管理者として、グローバル ID を使用してインダウト・トランザクションを手動でコミットします。

```
SQL> CONNECT SYS@warehouse.acme.com AS SYSDBA
SQL> COMMIT FORCE 'SALES.ACME.COM.55d1c563.1.93.29';
```

手順 6: DBA_2PC_PENDING を使用した MIXED 結果のチェック

トランザクションを手動で強制的にコミットまたはロールバックした後も、ペンディング・トランザクション表の対応する行はそのまま残っています。トランザクションの状態は、トランザクションをどのように強制完了したかに応じて変わります。

Oracle Database には、必ず**ペンディング・トランザクション表**があります。これは、2 フェーズ・コミットの各フェーズの進行に従って分散トランザクションに関する情報を格納する特別な表です。データベースのペンディング・トランザクション表は、DBA_2PC_PENDING データ・ディクショナリ・ビューを介して問い合せることができます（詳細は表 33-1 を参照）。

また、ペンディング・トランザクション表で特に重要なものとして、DBA_2PC_PENDING.MIXED に示される MIXED 結果フラグがあります。ペンディング・トランザクションを強制的にコミットまたはロールバックする場合は、選択を誤る可能性があります。たとえば、ローカル管理者がトランザクションをロールバックしたにもかかわらず、他のノードではそのトランザクションをコミットしてしまうといったことが考えられます。このような誤った判断は自動的に検出され、対応するペンディング・トランザクションのレコードに損傷フラグが設定されます（MIXED=yes）。

RECO バックグラウンド・プロセスは、ペンディング・トランザクション表の情報を使用して、インダウト・トランザクションの状態を最終決定します。また、管理者がペンディング・トランザクション表の情報を使用して、保留中の分散トランザクションの自動リカバリ手順を手動で上書きすることもできます。

RECO によって自動的に解決されたトランザクションは、すべてペンディング・トランザクション表から削除されます。また、管理者によって正しく解決されたインダウト・トランザクションに関する情報も、RECO が通信を再確立したときにチェックされて、すべてペンディング・トランザクション表から自動的に削除されます。ただし、管理者が解決したことによってノード間にまたがって MIXED の結果が生じた行は、DBMS_TRANSACTIONS.PURGE_MIXED を使用して手動で削除しないかぎり、関係しているすべてのノードのペンディング・トランザクション表にそのまま残ります。

ロックによるデータ・アクセスの障害

SQL 文を発行すると、データベースは文を正常に実行するために必要なリソースのロックを試みます。しかし、要求されたデータが、コミットされていない他のトランザクションの文によって現在保持されていて、長時間ロックされたままになっていると、タイムアウトが発生します。

データ・アクセスの障害に関しては、次の事例について考慮してください。

- トランザクションのタイムアウト
- インダウト・トランザクションによるロック

トランザクションのタイムアウト

リモート・データベースのロックを必要とする DML 文は、要求したデータのロックを別のトランザクションが所有している場合にブロックされる可能性があります。このようなロックによって SQL 文の要求がブロックされ続けると、次の一連のイベントが発生します。

1. タイムアウトが発生します。
2. データベースは文をロールバックします。
3. データベースは次のエラー・メッセージをユーザーに返します。

```
ORA-02049: タイムアウト : 分散トランザクションがロックを待機しています。
```

トランザクションによってデータは変更されていないので、タイムアウトの結果として必要な処理はありません。アプリケーションでは、デッドロックが発生したものとして処理を継続してください。文を実行したユーザーは、後で同じ文の再実行を試みることができます。それでもロックが続く場合には、ユーザーは管理者に連絡して問題を報告してください。

インダウト・トランザクションによるロック

ローカル・データベースのロックを必要とする問合せまたは DML 文は、インダウト分散トランザクションによるリソースのロックのために無期限にブロックされる可能性があります。この場合、データベースは次のエラー・メッセージを発行します。

```
ORA-01591: インダウト分散トランザクション identifier がロックを保持しています。
```

この場合、データベースは SQL 文をただちにロールバックします。文を実行したユーザーは、後で同じ文の再実行を試みることができます。それでもロックが続く場合には、ユーザーは管理者に連絡して問題を報告してください。このとき、インダウト分散トランザクションの ID も併せて報告してください。

2 フェーズ・コミットの重要な部分の処理中に障害が発生する確率は低いため、このような状況が起こる可能性はほとんどありません。仮にこのような障害が発生した場合でも、ネットワーク障害やシステム障害から迅速にリカバリできれば、手動で介入しなくても問題は自動的に解決されます。したがって、この種の問題は、ユーザーや DBA に検出される前に解決されるのが普通です。

分散トランザクション障害のシミュレーション

分散トランザクションの障害は、次のような理由で強制的に発生させることができます。

- RECO がトランザクションのローカル部分を自動的に解決する様子を観察するため
- インダウト分散トランザクションの手動解決の演習を行い、その結果を観察するため

ここでは、使用可能な機能とその操作に必要な手順を説明します。

分散トランザクションの強制障害

COMMIT 文の COMMENT パラメータには、コメントを挿入できます。分散トランザクションの 2 フェーズ・コミットのフェーズ時に意図して障害を発生させるには、COMMENT パラメータに、次のコメントを挿入します。

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

コメント内の *n* には、次の整数のいずれかが入ります。

n	影響
1	収集後コミット・ポイントをクラッシュ
2	収集後コミット・ポイント・サイト以外をクラッシュ
3	準備前にクラッシュ（コミット・ポイント・サイト以外）
4	準備後にクラッシュ（コミット・ポイント・サイト以外）
5	コミット前にコミット・ポイント・サイトをクラッシュ
6	コミット後にコミット・ポイント・サイトをクラッシュ
7	コミット前にコミット・ポイント・サイト以外をクラッシュ
8	コミット後にコミット・ポイント・サイト以外をクラッシュ
9	情報消去前にコミット・ポイント・サイトをクラッシュ
10	情報消去前にコミット・ポイント・サイト以外をクラッシュ

たとえば、ローカルのコミット・ポイント強度がリモートのコミット・ポイント強度よりも高く、双方のノードが更新されている場合、次の文では、次のメッセージが返されます。

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

ORA-02054: トランザクション 1.93.29 はインダウトです。
ORA-02059: コミット・コメントに ORA-2PC-CRASH-TEST-7 が含まれています。
```

この時点で、インダウト分散トランザクションが DBA_2PC_PENDING ビューに表示されます。対応している場合は、RECO がそのトランザクションを自動的に解決します。

RECO の有効化と無効化

Oracle Database インスタンスの RECO バックグラウンド・プロセスは、分散トランザクションに伴う障害を自動的に解決します。ノードの RECO バックグラウンド・プロセスは、時間間隔を指数関数的に広げながら、インダウト分散トランザクションのローカル部分のリカバリを試みます。

RECO は、障害を起こしたトランザクションに関係している他のノードに対して、既存の接続を使用するか、または新しい接続を確立できます。接続が確立されると、RECO はすべてのインダウト・トランザクションを自動的に解決します。各データベースのペンディング・トランザクション表内にインダウト・トランザクションに対応している行があれば、自動的に削除されます。

ENABLE/DISABLE DISTRIBUTED RECOVERY オプションを指定して ALTER SYSTEM 文を使用すると、RECO を使用可能または使用禁止にできます。たとえば、RECO を一時的に使用禁止にして 2 フェーズ・コミットの障害を強制的に発生させ、インダウト・トランザクションを手動で解決できます。

次の文は、RECO を使用禁止にします。

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

一方、次の文は RECO を使用可能にし、インダウト・トランザクションが自動的に解決されるようにします。

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

注意： シングル・プロセス・インスタンス（MS-DOS が稼働している PC など）では、インスタンス以外のバックグラウンド・プロセスは稼働しておらず、したがって RECO プロセスはありません。そのため、分散システムに参加するシングル・プロセス・インスタンスを起動したときは、前述の文を使用して手動で分散リカバリを使用可能にする必要があります。

読込み一貫性の管理

Oracle Database の分散読込み一貫性の実装には、重要な制限事項があります。この問題は、各システムが独自の SCN を持っていることに起因します。SCN は、データベースの内部タイムスタンプとして表示できます。Oracle Database サーバーは、SCN を使用して、問合せにどのバージョンのデータを返せばよいかを判断します。

分散トランザクションの SCN は、リモート SQL 文の最後と、各トランザクションの最初および最後で同期化されます。2 つのノード間の通信量が多く、特に分散更新がある場合には、この同期化が頻繁に実行されます。しかし、分散システムの SCN の絶対的な同期を保つための実用的な手段は存在しません。つまり、あるノードの SCN が別のノードの SCN と比べて幾分古くなるというような状況が常に存在します。

このような SCN の食い違いから、実行した問合せがわずかに古いスナップショットを使用する可能性があります。その問合せ結果には、リモート・データベースに対する最新の変更が含まれていません。そのため、問合せを実行したときに、読込み一貫性に従ってデータが取得されているにもかかわらず、そのデータが古い場合があります。そのような問合せによって取得したデータは、すべて古い SCN に基づいています。したがって、ローカルに実行した更新トランザクションによってリモート・ノードの 2 つの表が更新された場合、次回のリモート・アクセスで両方の表から選択したデータには、更新前のデータが含まれることになります。

SCN が食い違っているために起こりうる結果の 1 つとして、`SELECT` 文が 2 つ連続している場合に、それら 2 つの文の間で DML をまったく実行していなくても各文で異なるデータが取得されることがあります。たとえば、UPDATE 文を発行して、リモート・データベースで更新をコミットしたとします。このリモート表に基づくビューに対して `SELECT` 文を発行すると、ビューには更新された行が表示されません。次に `SELECT` 文を発行するときは、更新された行が表示されます。

次の手法を使用すると、問合せの直前に 2 つのマシンの SCN が必ず同期化されます。

- SCN はリモート問合せの最後に同期化されるので、各リモート問合せの前に、同じサイトで `SELECT * FROM DUAL@REMOTE` というようなダミーのリモート問合せを実行します。
- SCN はすべてのリモート・トランザクションの最初に同期化されるので、リモート問合せを発行する前に、現行トランザクションをコミットまたはロールバックします。

第 VI 部

付録

第 VI 部は、このマニュアルの付録です。この章の内容は、次のとおりです。

- 付録 A「リリース 11g における DBMS_JOB のサポート」

A

リリース 11*g* における DBMS_JOB のサポート

この付録の内容は次のとおりです。

DBMS_JOB の概要

DBMS_JOB はジョブのスケジュールに使用する PL/SQL パッケージです。このパッケージは、より強力で柔軟性の高い Oracle Scheduler で置き換えられています。DBMS_JOB から Oracle Scheduler に切り替えることをお薦めしますが、DBMS_JOB は下位互換性を維持するために引き続きサポートされます。

関連項目： 第 27 章「Oracle Scheduler を使用したジョブのスケジューリング」

DBMS_JOB の構成

JOB_QUEUE_PROCESSES 初期化パラメータには、ジョブの実行用に作成できるプロセスの最大数を指定します。Oracle Database リリース 11*g* 以降では、JOB_QUEUE_PROCESSES のデフォルトが 1000 になっています。ジョブ・コーディネータ・プロセスは、実行されるジョブ数と使用可能なリソースに基づいて、必要な数のジョブ・キュー・プロセスのみを起動します。JOB_QUEUE_PROCESSES により低い数字を設定すると、ジョブ・キュー・プロセスの数を制限できます。JOB_QUEUE_PROCESSES を 0 に設定すると、DBMS_JOB が使用禁止になります。

DBMS_JOB と Oracle Scheduler の使用

DBMS_JOB と Oracle Scheduler（スケジューラ）は、同じジョブ・コーディネータを使用して、ジョブ・キュー・プロセスを起動します。（スケジューラの場合、これらのプロセスはジョブ・スレーブ・プロセスと呼ばれます。この項では、これらの用語を同じ意味で使用します。）通常、DBMS_JOB 用のジョブ・キュー・プロセス数を制限するには、JOB_QUEUE_PROCESSES 初期化パラメータを使用し、スケジューラのジョブ・スレーブ・プロセス数を制限するには、スケジューラ属性 max_job_slave_processes を使用しますが、スケジューラは JOB_QUEUE_PROCESSES パラメータの影響を受けます。

スケジューラのジョブ・スレーブ・プロセスの最大数は、JOB_QUEUE_PROCESSES と max_job_slave_processes の値の小さいほうの値で決まります。次に例を示します。

- JOB_QUEUE_PROCESSES が 10、max_job_slave_processes が 20 に設定されている場合、DBMS_JOB とスケジューラの間で共有されるジョブ・スレーブ・プロセスがジョブ・コーディネータで起動される個数の最大値は 10 になります。
- JOB_QUEUE_PROCESSES が 20 で max_job_slave_processes が 10 の場合は、最大 20 個のジョブ・スレーブ・プロセスがコーディネータで起動されます。スケジューラで使用できるのは 10 個までですが、DBMS_JOB では 20 個すべてを使用できます。

JOB_QUEUE_PROCESSES が 0 の場合は、DBMS_JOB が使用禁止になり、スケジューラのジョブ・スレーブ・プロセスの最大数は max_job_slave_processes スケジューラ属性で制御されます。

関連項目：

- 28-6 ページ「タスク 2E: スケジューラ属性の設定」
- JOB_QUEUE_PROCESSES 初期化パラメータの詳細は、『Oracle Database リファレンス』を参照してください。

DBMS_JOB から Oracle Scheduler への移行

この項では、DBMS_JOB パッケージで作成されたジョブを取得し、DBMS_SCHEDULER パッケージを構成して制御する Oracle Scheduler を使用してそれらをリライトする方法の例をいくつか示します。

ジョブの作成

次は、DBMS_JOB を使用してジョブを作成する例です。

```
VARIABLE jobno NUMBER;
BEGIN
DBMS_JOB.SUBMIT(:jobno, 'INSERT INTO employees VALUES (7935, ''SALLY'',
   ''DOGAN'', ''sally.dogan@xyzcorp.com'', NULL, SYSDATE, ''AD_PRES'', NULL,
    NULL, NULL, NULL);', SYSDATE, 'SYSDATE+1');
COMMIT;
END;
/
```

DBMS_SCHEDULER を使用した等価の文は、次のようになります。

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB(
   job_name          =>  'job1',
   job_type          =>  'PLSQL_BLOCK',
   job_action        =>  'INSERT INTO employees VALUES (7935, ''SALLY'',
     ''DOGAN'', ''sally.dogan@xyzcorp.com'', NULL, SYSDATE,''AD_PRES'', NULL,
      NULL, NULL, NULL);');
   start_date        =>  SYSDATE,
   repeat_interval   =>  'FREQ = DAILY; INTERVAL = 1');
END;
/
```

ジョブの変更

次は、DBMS_JOB を使用してジョブを変更する例です。

```
BEGIN
DBMS_JOB.WHAT(31, 'INSERT INTO employees VALUES (7935, ''TOM'', ''DOGAN'',
   ''tom.dogan@xyzcorp.com'', NULL, SYSDATE,''AD_PRES'', NULL,
   NULL, NULL, NULL);');
COMMIT;
END;
/
```

この文では、別の値を挿入するように JOB1 の処理が変更されます。DBMS_SCHEDULER を使用した等価の文は、次のようになります。

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE(
   name          => 'JOB1',
   attribute     => 'job_action',
   value         => 'INSERT INTO employees VALUES (7935, ''TOM'', ''DOGAN'',
      ''tom.dogan@xyzcorp.com'', NULL, SYSDATE, ''AD_PRES'', NULL,
      NULL, NULL, NULL);');
END;
/
```

ジョブ・キューからのジョブの削除

次の例では、DBMS_JOB を使用してジョブを削除します。14144 は実行されているジョブの番号です。

```
BEGIN
DBMS_JOB.REMOVE(14144);
COMMIT;
END;
/
```

DBMS_SCHEDULER を使用して、かわりに次のような文を発行します。

```
BEGIN
   DBMS_SCHEDULER.DROP_JOB('myjob1');
END;
/
```

関連項目：

- DBMS_SCHEDULER パッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 第 27 章「Oracle Scheduler を使用したジョブのスケジューリング」

索引

数字

A

B

C

D

E

F

G

H

P

R

S

T

U

V

W

X

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

に

ね

は

ひ

ふ

へ

ほ

み

む

め

も

ゆ

よ

ら

り

る

れ

ろ

わ