Oracle® Database
VLDB およびパーティショニング・ガイド
19c

F16114-08(原本部品番号:E96199-11)

2021年8月

タイトルおよび著作権情報

Oracle Database VLDBおよびパーティショニング・ガイド, 19c

F16114-08

原本著者: Eric Belden

原本協力者: Penny Avril、Hermann Baer、Yasin Baskan、Gregg Christman、Jean-Pierre Dijcks、Sandeep Doraiswamy、Amit Ganesh、Lilian Hobbs、Kevin Jernigan、Dominique Jeunot、Hariharan Lakshmanan、Paul Lane、Sue K。Lee、Diana Lorentz、Vineet Marwah、Valarie Moore、Sujatha Muthulingam、Ajit Mylavarapu、Tony Morales、Ananth Raghavan、Venkatesh Radhakrishnan、Vivekanandhan Raja、Andy Rivenes、Chandrajith Unnithan、Mark Van de Wiel

原本協力者: Frederick Kush

目次

- 索引

例一覧

- 4-29　レンジ - レンジ・パーティション表へのパーティションの追加
- 4-30　CREATE TABLEのFOR EXCHANGE WITH句の使用
- 4-31　レンジ・パーティションの交換
- 4-32　参照パーティション表のパーティションの交換
- 4-33　参照パーティション表のカスケードを使用するパーティションの交換
- 4-34　レンジ・パーティションのマージ
- 4-35　パーティション化戦略の変更
- 4-36　レンジ・パーティション表のパーティションの分割、および索引の再作成
- 4-37　オンラインでのレンジ・パーティション表のパーティションの分割
- 4-38　リスト・パーティション表のデフォルト・パーティションの分割
- 4-39　複数のパーティションの切捨て
- 4-40　複数のサブパーティションの切捨て
- 4-41　コレクション表のパーティションの再定義
- 4-42　パーティション表へのオンライン変換にALTER TABLEのMODIFY句を使用する方法
- 4-43　ハイブリッド・レンジ・パーティション表の作成
- 4-44　ハイブリッド・レンジ・パーティション表への外部パーティションの追加
- 4-45　ハイブリッド・レンジ・パーティション表への変換
- 4-46　ハイブリッド・パーティション表から内部表への変換
- 4-47　ハイブリッド・パーティション表でのADOの使用
- 4-48　ハイブリッド・パーティション表のデフォルト・パーティションの分割
- 4-49　ハイブリッド・パーティション表の内部パーティションのデータと外部非パーティション表の交換
- 4-50　外部非パーティション表のデータとハイブリッド・パーティション表の内部パーティションの交換
- 5-1　ヒート・マップ・ビュー
- 5-2　DBMS_HEAT_MAPパッケージのサブプログラムの使用
- 5-3　ILM ADOポリシーを含む表の作成
- 5-4　ILM ADOポリシーの追加
- 5-5　ILM ADOポリシーの無効化と削除
- 5-6　セグメント・レベルの圧縮層およびストレージ層の使用
- 5-7　行レベルのハイブリッド列圧縮の使用によるADOポリシーの作成
- 5-8　行レベルの拡張圧縮の使用によるADOポリシーの作成
- 5-9　CUSTOMIZE_ILMを使用したADO設定のカスタマイズ
- 5-10　インデータベース・アーカイブの使用
- 5-11　時間的な有効性による表の作成
- 5-12　ILMシステムの手動実装
- 6-1　圧縮パーティション・マテリアライズド・ビュー
- 7-1　一意索引および主キー制約の作成
- 8-1　CustomersおよびSalesに対する問合せの実行計画の実行
- 8-2　CustomersおよびSalesに対する問合せの実行計画出力
- 8-3　コンシューマ・グループを使用したパラレル文キュー内の優先順位の設定
- 8-4　UNION ALLの実行計画
- 8-5　INSERT SELECTのパラレル化
- 8-6　UPDATEとDELETEのパラレル化
- 8-7　UPDATEとDELETEのパラレル化

図一覧

表一覧

はじめに

このマニュアルでは、VLDB計画の重要な要素であるパーティション化およびパラレル実行を中心に、大規模データベース(VLDB)の概要を説明します。パーティション化を行うと、様々なアプリケーションのパフォーマンス、管理性および可用性が向上し、大量のデータを保存するための総所有コストの削減に役立ちます。パラレル実行では、大容量データおよび効率が悪いSQL操作を処理して、処理時間を大幅に短縮できます。

この章に含まれる内容は次のとおりです。

対象読者

このマニュアルは、データベース管理者(DBA)、および大規模データベース(VLDB)向けのアプリケーションを作成、管理、記述する開発者を対象としています。

ドキュメントのアクセシビリティについて

Oracle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/pls/topic/lookup?ctx=acc&id=docacc)を参照してください。

Oracle Supportへのアクセス

サポートを購入したオラクル社のお客様は、My Oracle Supportを介して電子的なサポートにアクセスできます。詳細情報は(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info)か、聴覚に障害のあるお客様は(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs)を参照してください。

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概要
- Oracle Database管理者ガイド
- Oracle Databaseデータウェアハウス・ガイド
- Oracle Databaseリファレンス
- Oracle Database SQL言語リファレンス
- Oracle Database SQLチューニング・ガイド
- Oracle Databaseパフォーマンス・チューニング・ガイド

表記規則

このマニュアルでは次の表記規則を使用します。

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

Oracle Database VLDBおよびパーティショニング・ガイドのこのリリースの変更

この章では、『*Oracle Database VLDB*およびパーティショニング・ガイド』の変更点について説明します。

- Oracle Database 19cでのVLDBおよびパーティション化の変更点
- Oracle Databaseリリース18cでのVLDBおよびパーティショニングの変更点

関連項目:

- Oracle Databaseの各エディションで使用できる機能を判断する場合は、『Oracle Databaseライセンス情報ユーザー・マニュアル』を参照
- このリリースの新機能に関する完全な説明は、『Oracle Database新機能ガイド』を参照
- このリリースで非推奨になった機能およびサポートが終了した機能の詳細は、『Oracle Databaseアップグレード・ガイド』を参照

Oracle Database 19cでのVLDBおよびパーティション化の変更点

Oracle Database release 19c、バージョン19.1の大規模データベースおよびパーティション化の変更点は次のとおりです。

新機能

Oracle Databaseリリース19c、バージョン19.1には、大規模データベースをサポートするための次のような新機能があります。

- Oracleハイブリッド・パーティション表

  Oracleハイブリッド・パーティション表は、クラシック内部パーティション表と外部パーティション表を、ハイブリッド・パーティション表と呼ばれるより一般的なパーティション化に結合します。この機能を使用すると、内部パーティションと外部パーティション(データベース外のソースに存在するパーティション)を単一のパーティション表に簡単に統合できます。この機能を使用すると、ストレージ・ソリューションのコストを削減するために、非アクティブなパーティションを外部ファイルに簡単に移動できます。

  ハイブリッド・パーティション表のパーティションは、Oracle表領域と外部ソース(カンマ区切り値(CSV)レコードを含むLinuxファイルや、Javaサーバーを使用したHadoop Distributed File System (HDFS)上のファイルなど)の両方に存在できます。ハイブリッド・パーティション表では、外部パーティションの、すべての既存の外部表タイプ(ORACLE_DATAPUMP、ORACLE_LOADER、ORACLE_HDFS、ORACLE_HIVE)がサポートされます。

  関連項目:

  - ハイブリッド・パーティション表の概要は、ハイブリッド・パーティション表を参照してください
  - ハイブリッド・パーティション表の管理の詳細は、ハイブリッド・パーティション表の管理を参照してください
  - パーティション化されたハイブリッド表の詳細は、Oracle Database管理者ガイドを参照してください
  - パーティション表の概念の詳細は、Oracle Database概要を参照してください
  - インメモリー列ストアおよびハイブリッド・パーティション表の詳細は、Oracle Database In-Memoryガイドを

参照してください

- ハイブリッド・パーティション表の最適化の詳細は、Oracle Database SQLチューニング・ガイドを参照してください
- CREATE TABLEおよびALTER TABLE SQLコマンドを使用したハイブリッド・パーティション表の作成および変更の詳細は、Oracle Database SQL言語リファレンスを参照してください
- ハイブリッド・パーティション表でのSQL*Loaderの使用、ハイブリッド・パーティション表でのOracle Data Pumpの使用および外部表の管理の詳細は、*Oracle Database*ユーティリティを参照してください
- DBMS_HADOOPパッケージのCREATE_HYBRID_PARTNED_TABLEプロシージャなど、ハイブリッド・パーティション表でのPL/SQLプロシージャの使用方法の詳細は、Oracle Database PL/SQLパッケージおよびタイプ・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・ビュー(USER_TABLES、USER_ALL_TABLES、ALL_TABLES、ALL_ALL_TABLES、DBA_TABLES、DBA_ALL_TABLESビューなど)でのハイブリッド・パーティション表の詳細は、Oracle Databaseリファレンスを参照してください
- マテリアライズド・ビューおよびハイブリッド・パーティション表の詳細は、Oracle Databaseデータ・ウェアハウス・ガイドを参照してください

Oracle Databaseリリース18cでのVLDBおよびパーティショニングの変更点

Oracle Database release 18c、バージョン18.1の大規模データベースおよびパーティション化の変更点は次のとおりです。

- 新機能

新機能

Oracle Databaseリリース18c、バージョン18.1には、大規模データベースをサポートするための次のような新機能があります。

- 拡張パラレル・パーティション・ワイズ操作

  パラレル・パーティション・ワイズSQL操作によって、問合せのパフォーマンスが大幅に向上し、応答時間が改善されます。パラレル・パーティション・ワイズ結合は、大きな結合を効率的かつ高速に処理するために一般的に使用されます。

  パラレル・パーティション・ワイズ結合に加えて、SELECT DISTINCT句およびSQLウィンドウ関数を使用した問合せでは、パラレル・パーティション・ワイズ操作を実行できます。

  関連項目:

  - パーティション・ワイズ操作
  - データ・ウェアハウスでのパーティション・ワイズ結合
  - データ・ウェアハウスおよび最適化方法の詳細は、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください

- パーティション化戦略の変更

  ALTER TABLE MODIFY PARTITION SQL文を使用して、通常の(ヒープ構成)表のパーティション化戦略を変更できま

す。パーティション化戦略の変更(ハッシュ・パーティション化からレンジ・パーティション化に変更するなど)は、オフラインまたはオンラインで実行できます。索引は、表の変更の際にメンテナンスされます。オンライン・モードで実行した場合、進行中のDML操作は変更の影響を受けません。

この機能を使用すると、パーティション化された表は、表を手動で再作成することなく更新されます。表の既存のパーティション化戦略をオンラインで変更すると、アプリケーションのダウンタイムなしに新規ビジネス要件用にパーティション化を調整できます。

関連項目:

- パーティション化戦略の変更について

- パーティションおよびサブパーティションのオンラインでのマージ

ONLINEキーワードをALTER TABLE MERGE PARTITIONおよびSUBPARTITION SQL文とともに使用すると、通常の(ヒープ構成)表に対するオンライン・マージ操作が有効になり、進行中のパーティション・マージ操作と同時にデータ操作言語(DML)の操作を実行できるようになります。

オンラインのパーティション・メンテナンス操作を有効にすると、問合せ専用ウィンドウの期間を計画することなく、必要に応じてすべての操作をスケジュールし実行できます。この機能により、アプリケーションの可用性の向上とアプリケーション開発の簡素化の両方が実現します。

関連項目:

- パーティションおよびサブパーティションのマージについて

- ヒート・マップ・データを使用した自動インメモリー管理

ヒート・マップ・データは、インメモリー列ストア(IM列ストア)のコンテンツを自動的に管理するための、自動データ最適化(ADO)を支援します。列統計およびその他の関連統計が含まれるヒート・マップ・データを使用すると、IM列ストアがいつ満杯(メモリー不足)になるかを判断できます。ほぼ満杯と判断されたときに、アクセス頻度が高くIM列ストアに移入することで利点が得られるセグメントが存在する場合は、非アクティブなセグメントが削除されます。

関連項目:

- ヒート・マップの使用
- インメモリー列ストアの有効化およびサイズ設定の詳細は、『Oracle Database In-Memoryガイド』を参照してください

- マルチテナント・パラレル文のキューイングの機能強化

パラレル実行の機能が強化され、マルチテナント・データベースでより効率的に動作するようになりました。PARALLEL_MAX_SERVERSおよびPARALLEL_SERVERS_TARGETの各初期化パラメータの更新などの機能強化によって、マルチテナント環境でのパラレル文のキューイングは、非PDBマルチテナントでの動作と同じように効率的に動作します。

関連項目:

パラレル実行(PX)サーバーとCDBおよびPDBの使用率制限の詳細は、『Oracle Multitenant管理者ガイド』を参照してください

- パラレル文のキューイングのタイムアウトおよびデキュー処理

  PARALLEL_QUEUE_TIMEOUT_ACTIONリソース・マネージャ・ディレクティブおよびDBMS_RESOURCE_MANAGER.DEQUEUE_PARALLEL_STATEMENT PL/SQLプロシージャの拡張機能を使用して、パラレル文のキューのタイムアウトおよびデキュー処理を指定できます。

  関連項目:

  - Oracle Database Resource Managerによるパラレル文のキューイングの管理について
  - 各コンシューマ・グループに対するパラレル文キューのタイムアウトの指定
  - パラレル文キューの順序の管理について

- PARALLEL_MIN_DEGREE初期化パラメータ

  関連項目:

  - 自動並列度の制御
  - PARALLEL_MIN_DEGREEの詳細は、『Oracle Databaseリファレンス』を参照してください

1 大規模データベースの概要

大規模データベース(VLDB)には、複数の計画が必要となる、管理上の課題があります。パーティション化は、VLDB計画の重要部分です。

現代では、数百GB以上、多くの場合、数TB以上のデータを含む重要なデータベースを運用している企業は珍しくありません。これらの企業は、大規模データベース(VLDB)のサポートやメンテナンスという課題を抱えており、これらの課題に対応するための方法を考える必要があります。

この章の構成は、次のとおりです。

- パーティション化の概要
- VLDBおよびパーティション化
- 情報ライフサイクル管理の基礎としてのパーティション化
- すべてのデータベースのパーティション化

ノート:

パーティション化機能を使用できるのは、Oracle Partitioning オプションを購入した場合のみです。

1.1 パーティション化の概要

パーティション化は、大規模な表および索引について、それらをより小さく管理しやすい単位に分割することでサポートを提供します。

パーティション化を行うと、アプリケーションに対して完全に透過的な**パーティション**と呼ばれるより小規模で管理しやすい単位に分解することにより、大規模な表や索引のサポートにおける主な問題に対応できます。パーティション表にアクセスするために、SQL問合せやデータ操作言語(DML)文を変更する必要はありません。ただし、パーティションの定義後にデータ定義言語(DDL)文がアクセスおよび操作できるのは、表や索引全体ではなく、個々のパーティションです。これが、パーティション化により、大規模データベース・オブジェクトの管理が簡略化される仕組みです。

表または索引の各パーティションでは、列名、データ型および制約などの論理属性は同一である必要がありますが、圧縮の有効化や無効化、物理記憶域設定および表領域などの物理属性は異なってもかまいません。

パーティション化は、様々なタイプのアプリケーション、特に大量のデータを管理するアプリケーションに便利です。OLTPシステムでは管理性や可用性が向上し、データ・ウェアハウス・システムではパフォーマンスや管理性が向上します。

次にパーティション化の利点を示します。

- 表全体ではなくパーティション・レベルでのデータのロード、索引の作成や再作成、バックアップやリカバリなどのデータ管理操作が可能です。これにより、これらの操作にかかる時間が大幅に削減されます。
- 問合せパフォーマンスが向上します。多くの場合、表全体ではなくパーティションのサブセットにアクセスすることで、問合せの結果を取得できます。一部の問合せでは、この技術(**パーティション・プルーニング**)により、パフォーマンスが大幅に向上します。
- メンテナンス操作のスケジュールされた停止時間の影響を大幅に軽減できます。

パーティション・メンテナンス操作におけるパーティションの独立性により、同じ表または索引の異なるパーティションでメンテナンス操作を同時に実行できます。メンテナンス操作に影響されないパーティションに対しても、SELECT操作やDML操作を同時に実行できます。

- メンテナンス期間、リカバリ時間および障害の影響を低減するために、重要な表や索引がパーティションに分割されていると、重要なデータベースの可用性が向上します。
- パラレル実行を行うと、リソース使用率が最適化され、実行時間が最短化されるという特定の利点があります。パラレル実行は、問合せおよびDMLやDDLに対してサポートされています。

パーティション化により、Oracle Database内でのデータ・アクセスが高速になります。データベースのデータが10GBか10TBかに関係なく、パーティション化によりデータ・アクセスを大幅に改善できます。パーティション化は、アプリケーションに変更を加えずに実装できます。たとえば、表にアクセスするSELECT文やDML文を変更せずに、パーティション化されていない表をパーティション表に変換できます。パーティション化を使用するために、アプリケーション・コードをリライトする必要はありません。

1.2 VLDBおよびパーティション化

パーティション化は、大規模データベース(VLDB)の管理のための重要な計画です。

大規模データベースには、最小絶対サイズはありません。VLDBはより小規模なデータベースと同じようなデータベースですが、その管理には固有の問題があります。それらの問題は、全体のサイズおよびそのサイズのシステムに対する操作実行のコスト・パフォーマンスに関連しています。

データベース・サイズの恒常的な増大に関連している傾向をいくつか示します。

- 長い間、システムは個別に開発されてきました。企業は、それらのシステムを結合することで、システムの維持費を削減できるだけでなく、複数の部門にまたがる分析も可能になる利点に注目しはじめました。データベースとアプリケーションの統合は、データベース・サイズが増大し続ける主要な要因の1つになっています。
- 多くの企業は、最小限の間データを保存するための規則に対応しています。一般的に、規則が、より多くのデータをより長い期間保存する原因になっています。
- 企業が販売や運用の拡張により成長することや、合併や吸収により成長することが、生成および処理されるデータ量の増大の原因になっています。同時に、日常業務でデータベースに依存するユーザー人口も増加します。

パーティション化は、大規模データベースを管理するためのクリティカルな機能です。データ量の増大は、大規模データベースでパーティション化が対応する基本的な問題です。また、パーティション化を行うことにより、特に表や索引の増大に伴い分断攻略法を使用して、データベースの表や索引を管理できます。パーティション化は、一貫したパフォーマンスを維持しながら、管理リソースまたはハードウェア・リソースを増やしすぎずに、データベースを大規模データ・セットに拡張できる機能です。

関連項目:

- パーティション化の実装に関する可用性、管理性およびパフォーマンスの考慮事項の詳細は、「可用性、管理性およびパフォーマンスのためのパーティション化」を参照してください
- 大規模データベースのバックアップおよびリカバリに関連する問題の詳細は、「VLDBのバックアップおよびリカバリ」を参照してください
- 大規模データベースの重要な要素である記憶域のベスト・プラクティスの詳細は、「VLDBの記憶域管理」を参照してください

1.3 情報ライフサイクル管理の基礎としてのパーティション化

パーティション化は、情報ライフサイクル管理(ILM)のサポートを提供します。

情報ライフサイクル管理(ILM)は、使用できる期間中、データを管理するための一連のプロセスおよびポリシーです。ILM計画の重要な要素の1つは、アクセス頻度の低い古いデータはよりコストがかからず効率の低いストレージ層に保存し、日常業務で使用されるより新しいデータは最も高速で、可用性の高いストレージ層に保存するなど、データの存続期間中のいつにおいてもデータを保存するために最適で費用効率が高い中間点を決定することです。古いデータは更新される頻度も低いため、そのような場合には、圧縮して読取り専用としてデータを保存することをお薦めします。

Oracle Databaseは、ILMソリューションの実装に理想的な環境を提供します。複数のストレージ層がサポートされており、すべてのデータがOracle Database内に存在するため、複数のストレージ層はアプリケーションに対して透過的で、データも安全に保たれます。パーティション化により、表内のデータを別々のパーティションに保存することを可能にする基礎となるテクノロジが提供されます。

エンタープライズ・レベルのシステムでは複数のストレージ層や複雑なILMポリシーが採用されている場合が多いですが、大部分の企業およびデータベースで、ある程度の情報ライフサイクル管理が必要です。最も基本的なILM操作である古いデータのアーカイブや、それらのデータのデータベースからの削除またはパージは、パーティション化を使用している場合には大幅に高速になります。

関連項目:

ILMの詳細は、「時間ベース情報の管理およびメンテナンス」を参照してください

1.4 すべてのデータベースのパーティション化

パーティション化は、大規模および小規模のデータベースに役立ちます。

パーティション化のメリットは、大規模データベースだけのものではありません。小規模データベースも含めすべてのデータベースがパーティション化の恩恵を受けます。サイズがMB単位のデータベースでさえ、数TBに及ぶ最大級サイズのシステムと同様に、パーティション化によってパフォーマンスと管理性が向上します。

関連項目:

- データ・ウェアハウス環境においてパーティション化によりもたらされる利点の詳細は、「データ・ウェアハウス環境でのパーティション化の使用」を参照してください
- OLTP環境においてパーティション化によりもたらされる利点の詳細は、「オンライン・トランザクション処理環境でのパーティション化の使用」を参照してください

2 パーティション化の概念

パーティション化を行うと、様々なアプリケーションのパフォーマンス、管理性および可用性が向上し、大量のデータを保存するための総所有コストの削減に役立ちます。

パーティション化により、表、索引および索引構成表をより細かい単位に細分化できるようになり、これらのデータベース・オブジェクトのよりきめ細かい管理およびアクセスが可能になります。あらゆるビジネス要件に対応するための、パーティション化計画および拡張が豊富に用意されています。完全に透過的であるため、コストが高く時間のかかるアプリケーションの変更を行わずに、パーティション化をほぼすべてのアプリケーションに適用できます。

この章の構成は、次のとおりです。

- パーティション化の概要
- パーティション化の利点
- パーティション化計画
- パーティション化の拡張
- パーティション表の索引

2.1 パーティション化の概要

パーティション化は、オブジェクトをさらに小さな部分に分割する技法を提供します。

パーティション化により、表、索引および索引構成表をより細かい単位に細分化できるようになります。このようなデータベース・オブジェクトの単位をパーティションと呼びます。各パーティションには独自の名前があり、独自の記憶特性を持つことができます。

次の内容について説明します。

- パーティション化の基本
- パーティション化キー
- パーティション表
- パーティション化索引構成表
- システム・パーティション化
- 情報ライフサイクル管理のためのパーティション化
- ハッシュ・クラスタのレンジ・パーティション化
- パーティション化およびLOBデータ
- 外部表のパーティション化
- ハイブリッド・パーティション表
- XMLTypeデータおよびオブジェクト・データのコレクション

2.1.1 パーティション化の基本

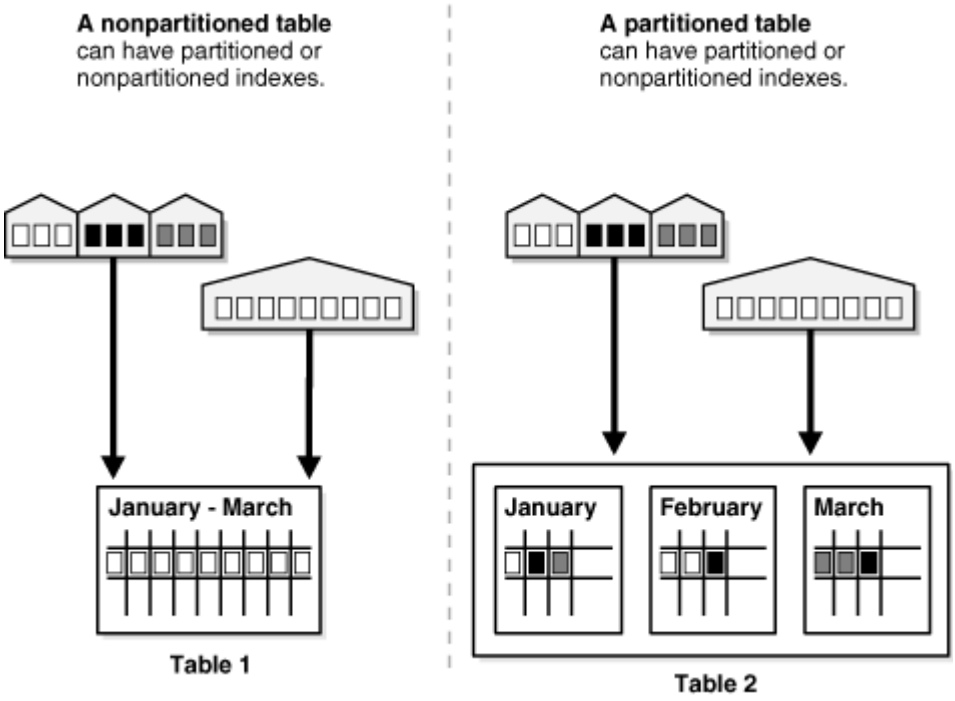
パーティション化により、まとめてまたは個別にオブジェクトを管理できます。

データベース管理者の視点からすると、パーティション・オブジェクトには、まとめて管理することも個別に管理することも可能な複数の単位があります。このため、管理者は、パーティション・オブジェクトをかなり柔軟に管理できます。ただし、アプリケーションに

とっては、パーティション表はパーティション化されていない表と同じであるため、SQL問合せやDML文を使用してパーティション表にアクセスする際に変更は必要ありません。

図2-1に、パーティション表とパーティション化されていない表の違いを図で示します。

図2-1 パーティション表と非パーティション表のビュー



ノート:

パーティション・オブジェクトのすべてのパーティションは、同じブロック・サイズの表領域に存在する必要があります。

関連項目:

- 複数ブロック・サイズの詳細は、『*Oracle Database概要*』を参照してください。
- パーティション化の一般的な制限、パーティション表や索引を作成および変更するためのパーティション化句の正確な構文、その使用に関する制限、および表の作成や変更に必要な特定の権限の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

2.1.2 パーティション化キー

パーティション表内の各行は、キーを使用して1つのパーティションに明確に割り当てられます。

パーティション化キーは、各行が格納されるパーティションを決定する1つ以上の列で構成されています。パーティション化キーにより、適切なパーティションに対する挿入、更新および削除操作が自動的に指示されます。

パーティションのコンテンツの検証

パーティション内の行がパーティション定義に準拠しているかどうか、または行のパーティション化キーがパーティション定義に違反しているかどうかを識別するには、ORA_PARTITION_VALIDATION SQL関数を使用します。このSQL関数は行IDを入力として受け取り、行が正しいパーティションにある場合は1を返し、それ以外の場合は0を返します。この関数は、内部パーティション表、

外部パーティション表、および内部と外部のパーティションやサブパーティションのハイブリッド・パーティション表に適用できます。

例:

```
SQL>  CREATE TABLE test1 (column1 NUMBER)
             PARTITION BY RANGE(column1)
             (PARTITION p1 VALUES LESS THAN (10),
              PARTITION p2 VALUES LESS THAN (20));

SQL> CREATE TABLE test2 (column1 NUMBER);

SQL> INSERT INTO test1 VALUES (1);

SQL> INSERT INTO test2 VALUES (99);

SQL> ALTER TABLE test1 EXCHANGE PARTITION p2 WITH TABLE test2 WITHOUT VALIDATION;

SQL> SELECT test1.*, ORA_PARTITION_VALIDATION(rowid) FROM test1;

COL1       ORA_PARTITION_VALIDATION(ROWID)
---------- ------------------------------
         1                               1
        99                               0
```

関連項目:

- SQL関数の詳細は、Oracle Database SQL言語リファレンスを参照してください

2.1.3 パーティション表

ほとんどの表は、パーティション化できます。

LONGまたはLONG RAWデータ型の列を含む表を除き、任意の表を最大100万の別々のパーティションにパーティション化できます。ただし、使用できるのはCLOBまたはBLOBデータ型の列を含む表です。

次の内容について説明します。

- 表をパーティション化するタイミング
- 索引をパーティション化するタイミング

> ノート:
>
> ディスク使用率およびメモリー使用量(特にバッファ・キャッシュ)を低減するために、表およびパーティション表のパーティションをデータベース内に圧縮形式で保存できます。多くの場合、これにより、読取り専用操作がさらにスケールアップされます。表の圧縮も、問合せ実行の高速化につながります。ただし、CPU オーバーヘッドに多少コストがかかります。

関連項目:

表の管理のガイドラインの詳細は、『Oracle Database管理者ガイド』を参照してください

2.1.3.1 表をパーティション化するタイミング

表をパーティション化する必要がある、特定の状況があります。

次に、表のパーティション化を検討する状況に関する推奨事項を示します。

- 2GBを超える表。

  これらの表は、常にパーティション化の候補として考慮する必要があります。

- 新規データが最新のパーティションに追加される、履歴データを含む表。

  典型的な例は、現在の月のデータのみが更新可能で、残りの11か月分は読取り専用の履歴表です。

- コンテンツを種類の異なるストレージ・デバイスに分散する必要がある表。

2.1.3.2 索引をパーティション化するタイミング

索引をパーティション化する必要がある、特定の状況があります。

次に、索引のパーティション化を検討するタイミングに関する推奨事項を示します。

- データが削除される場合、索引メンテナンスを回避します。
- 索引全体を無効化せずにデータの一部でメンテナンスを実行する場合。
- 値が増え続ける列の索引が原因で発生する索引の誤差の影響を軽減する場合。

2.1.4 パーティション化索引構成表

パーティション化索引構成表は、索引構成表のパフォーマンス、管理性および可用性を向上させるのに非常に便利です。

索引構成表をパーティション化する場合の注意点は次のとおりです。

- パーティション化列は主キー列のサブセットである必要があります。
- 2次索引もパーティション化できます(ローカルおよびグローバルの両方)。
- OVERFLOWデータ・セグメントは、常に表パーティションと同一レベルでパーティション化されます。

関連項目:

索引構成表の詳細は、『Oracle Database概要』を参照してください。

2.1.5 システム・パーティション化

システム・パーティション化では、データベースによりデータの配置を制御せずに、アプリケーション制御のパーティション化を実行できます。

個々のパーティションの用途を指定せずに表をパーティションに分割する機能が、データベースにより提供されます。パーティション化のすべてをアプリケーションによって制御する必要があります。たとえば、パーティションを明示的に指定せずにシステム・パーティション表に挿入しようとすると失敗します。

システム・パーティション化により、パーティション化の利点(スケーラビリティ、可用性および管理性)がもたらされますが、パーティション化および実際のデータの配置はアプリケーションによって制御されています。

関連項目:

システム・パーティション化の詳細は、『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。

2.1.6 情報ライフサイクル管理のためのパーティション化

情報ライフサイクル管理(ILM)は、存続期間中のデータ管理に関連しています。

パーティション化は、データのグループ(パーティション)を種類の異なるストレージ・デバイスに分散し、個別に管理することを可能にするため、ILMにおいて重要な役割を果します。

関連項目:

情報ライフサイクル管理の詳細は、「時間ベース情報の管理およびメンテナンス」を参照してください

2.1.7 ハッシュ・クラスタのレンジ・パーティション化

パーティション化されたハッシュ・クラスタは、Oracle Databaseでサポートされています。

パーティション化されたハッシュ・クラスタでは、単一レベルのレンジ・パーティション化のみサポートされます。

関連項目:

パーティション化されたハッシュ・クラスタの詳細は、『Oracle Databaseリファレンス』を参照してください。

2.1.8 パーティション化およびLOBデータ

データベース内のLOB列に格納される画像やドキュメントなどの非構造化データもパーティション化できます。

表がパーティション化されたとき、すべての列はそのパーティションの表領域に含まれますが、LOB列のみは独自の表領域に格納されます。

LOBをメイン・データとは別に格納できるため、表が大規模なLOBで構成されている場合、この機能は非常に便利です。これは、メイン・データは頻繁に更新されるがLOBデータは更新されない場合に役立ちます。たとえば、従業員レコードに、頻繁に変更される可能性が低い写真が含まれている場合などです。ただし、従業員の個人情報(住所、部署、管理者など)は変更される可能性があります。この方法では、LOBデータの保存にはより安価なストレージを使用し、従業員レコードにはより高価で高速なストレージを使用することができます。

2.1.9 外部表のパーティション化

パーティション化は、外部表でサポートされています。

この機能では、パーティション化された複数の外部表にわたる問合せのための静的パーティション・プルーニング、動的プルーニングおよびパーティション・ワイズ結合など、最適化が可能です。また、この機能では、より優れたオプティマイザ計画を可能にする、外部表パーティションごとの、パーティションベースの増分統計収集を提供します。

ヒント:

クラウドのデータ・ソースの場合、外部表は手動で作成できますが、DBMS_CLOUD パッケージを使用することをお薦めします。

関連項目:

- 外部表の詳細は、『Oracle Databaseユーティリティ』を参照してください。
- DBMS_CLOUD PL/SQLパッケージの詳細は、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。

2.1.10 ハイブリッド・パーティション表

Oracleハイブリッド・パーティション表は、クラシック内部パーティション表をOracle外部パーティション表と結合し、ハイブリッド・パーティション表と呼ばれるより一般的なパーティション化を作成します。

ハイブリッド・パーティション表を使用すると、内部パーティションと外部パーティション(データベース外のソースに存在するパーティション)を単一のパーティション表に簡単に統合できます。この機能を使用すると、ストレージ・ソリューションのコストを削減するために、非アクティブなパーティションを外部ファイルに簡単に移動できます。

ハイブリッド・パーティション表のパーティションは、カンマ区切り値(CSV)レコードを含むLinuxファイル、Javaサーバーを使用したHadoop Distributed File System (HDFS)上のファイルまたはクラウド上のオブジェクト・ストアなど、Oracle表領域と外部ソースの両方に存在できます。ハイブリッド・パーティション表では、外部パーティションの、すべての既存の外部表タイプ(ORACLE_DATAPUMP、ORACLE_LOADER、ORACLE_HDFS、ORACLE_HIVE)がサポートされます。外部パーティションの外部表タイプでは、次のアクセス・ドライバ・タイプを使用します。

- ORACLE_DATAPUMP
- ORACLE_LOADER
- ORACLE_HDFS
- ORACLE_HIVE

ORACLE_LOADERおよびORACLE_DATAPUMPアクセス・ドライバ・タイプの外部パーティションの場合、ユーザーに次の権限を付与する必要があります。

- データファイルを含むディレクトリに対するREAD権限
- ロギング・ファイルや不良ファイルを含むディレクトリに対するWRITE権限
- プリプロセッサ・プログラムを含むディレクトリに対するEXECUTE権限

表レベルの外部パラメータは、ハイブリッド・パーティション表のすべての外部パーティションに適用されます。たとえば、EXTERNAL PARTITION ATTRIBUTES句で定義されたDEFAULT DIRECTORY値は、データファイル、ロギング・ファイルおよび不良ファイルのデフォルトの場所です。デフォルト・ディレクトリの場所は、PARTITION句のDEFAULT DIRECTORY値で上書きできます。ORACLE_HIVEおよびORACLE_HDFSアクセス・ドライバ・タイプの外部パーティションの場合、DEFAULT DIRECTORYは、ログ・ファイルの指定を格納するためにのみ使用されます。

制約は表全体に適用されるため、外部パーティションに格納されたデータに対しては、制約の施行はサポートされません。たとえば、主キー制約または外部キー制約は、ハイブリッド・パーティション表に対しては施行できません。ハイブリッド・パーティション表

では、RELY DISABLEモードの制約(NOT NULL、主キー、一意キー、外部主キーなど)のみがサポートされます。これらの制約に基づいて最適化をアクティブ化するには、セッション・パラメータQUERY_REWRITE_INTEGRITYをTRUSTEDまたはSTALE_TOLERATEDに設定します。

ハイブリッド・パーティション表では、内部パーティションと外部パーティション間で、パーティションベースの最適化を使用できます。パーティションベースの最適化には、内部データ・ソースと外部データ・ソース間で、次のものが含まれます。

- 静的パーティション・プルーニング
- 動的パーティション・プルーニング
- ブルーム・プルーニング

ハイブリッド・パーティション表を使用すると、コスト効率の目的で、内部パーティションと外部パーティションの間でデータを移動できます。ただし、表レベルで定義された自動データ最適化(ADO)は、表の内部パーティションにのみ影響します。

ハイブリッド・パーティション表でサポートされる操作

ハイブリッド・パーティション表でサポートされる操作は、次のとおりです。

- 単一レベルのRANGEおよびLISTパーティション化メソッドの作成
- ALTER TABLE ..の使用ADD、DROP、RENAMEパーティションなどのDDL
- パーティション・レベルでの外部データ・ソースの場所の変更(外部パーティションの場合)
- パーティション化された既存の内部表の、内部と外部の両方のパーティションを含むハイブリッド・パーティション表への変更
- 既存の場所を空の場所に変更することによる、空の外部パーティションの生成
- 内部パーティションでの一意でないグローバル部分索引の作成
- 内部パーティションでのマテリアライズド・ビューの作成
- 外部パーティションを含むマテリアライズド・ビューの作成(QUERY_REWRITE_INTEGRITY失効許可モードのみ)
- 外部パーティションでのフル・パーティション・ワイズ・リフレッシュ
- 内部パーティションでのハイブリッド・パーティション表に対するDMLトリガー操作
- 内部パーティションに対するANALYZE TABLE ... VALIDATE STRUCTUREを使用した検証(ハイブリッド・パーティション表のみ)
- 外部パーティション表を含まない既存のハイブリッド・パーティション表の、内部パーティションのみを含むパーティション表への変更
- 外部パーティションは、外部非パーティション表と交換できます。また、内部パーティションは、内部非パーティション表と交換できます。

ハイブリッド・パーティション表の制限

ハイブリッド・パーティション表の制限は、次のとおりです。

- 特に明記しないかぎり、外部表に適用される制限は、ハイブリッド・パーティション表にも適用されます
- REFERENCEおよびSYSTEMパーティション化メソッドはサポートされていません
- 単一レベルのLISTおよびRANGEパーティション化のみがサポートされます。
- 一意索引やグローバル一意索引は存在しません。部分索引のみが許可され、一意索引は部分索引にできません。

- HIVEについては、単一レベルのリスト・パーティション化のみがサポートされます。
- 属性クラスタリング(CLUSTERING句)は使用できません。
- DML操作は、ハイブリッド・パーティション表の内部パーティションに対してのみ実行できます(外部パーティションは読取り専用パーティションとして処理されます)
- 表レベルで定義されたインメモリーは、ハイブリッド・パーティション表の内部パーティションにのみ影響します。
- 列のデフォルト値はありません
- 非表示の列は使用できません。
- CELLMEMORY句は使用できません。
- SPLIT、MERGEおよびMOVEメンテナンス操作は、外部パーティションに対しては使用できません。
- LOB、LONGおよびADT型は使用できません。
- RELY制約のみが許可されます

関連項目:

- ハイブリッド・パーティション表の管理の詳細は、ハイブリッド・パーティション表の管理を参照してください
- パーティション化されたハイブリッド表の詳細は、Oracle Database管理者ガイドを参照してください
- パーティション表の概念の詳細は、Oracle Database概要を参照してください
- インメモリー列ストアおよびハイブリッド・パーティション表の詳細は、Oracle Database In-Memoryガイドを参照してください
- ハイブリッド・パーティション表の最適化の詳細は、Oracle Database SQLチューニング・ガイドを参照してください
- CREATE TABLEおよびALTER TABLE SQLコマンドを使用したハイブリッド・パーティション表の作成および変更の詳細は、Oracle Database SQL言語リファレンスを参照してください
- ハイブリッド・パーティション表でのSQL*Loaderの使用、ハイブリッド・パーティション表でのOracle Data Pumpの使用および外部表の管理の詳細は、*Oracle Database*ユーティリティを参照してください
- DBMS_HADOOPパッケージのCREATE_HYBRID_PARTNED_TABLEプロシージャなど、ハイブリッド・パーティション表でのPL/SQLの使用方法の詳細は、Oracle Database PL/SQLパッケージおよびタイプ・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・ビュー(データ・ディクショナリ・ビューの外部ファミリおよび*_TABLESビューを含む)内のハイブリッド・パーティション表の詳細は、Oracle Databaseリファレンスを参照してください
- マテリアライズド・ビューおよびハイブリッド・パーティション表の詳細は、Oracle Databaseデータ・ウェアハウス・ガイドを参照してください

2.1.11 XMLTypeデータおよびオブジェクト・データのコレクション

XMLTypeおよびオブジェクトの表や列を使用するときにパーティション化すると、パーティション化の一般的な利点が得られます。つまり、表および索引をより細かい単位に分割できるため、これらのデータベース・オブジェクトの管理やアクセスをよりきめ細かいレベルで行えるようになります。

XMLType表、すなわちXMLType列を含む表を、リスト、レンジまたはハッシュ・パーティション化を使用してパーティション化すると、

データに含まれるOrdered Collection Table(OCT)もデフォルトで自動的にパーティション化されます。このような同一レベル・パーティション化では、OCTのパーティション化は親(実)表のパーティション化方法に従って行われます。実表の各パーティションに対して、対応するコレクション表パーティションがあります。子要素は、その親要素の実表パーティションに対応するコレクション表パーティションに格納されます。

ネストした表を含む表をパーティション化する場合、Oracle Databaseは、ネストした表をパーティション化する方法の基礎として元の実表のパーティション化方法を利用します。このように、ネストした表の各パーティションに実表の1パーティションを対応させるパーティション化は、同一レベル・パーティション化と呼ばれます。デフォルトでは、ネストした表は、実表がパーティション化されるときに自動的にパーティション化されます。ただし、OCTまたはネストした表ではコンポジット・パーティション化はサポートされないので注意してください。

関連項目:

- XMLType表のパーティション化の詳細は、「XMLTypeおよびオブジェクトのコレクションのパーティション化」を参照してください
- ネストした表の構文は、『Oracle Database SQL言語リファレンス』を参照してください
- Oracle Databaseオブジェクト・リレーショナル開発者ガイド

2.2 パーティション化の利点

パーティション化によって、パフォーマンス、管理性および可用性が向上し、多様なアプリケーションに大きなメリットがもたらされます。

特定の問合せまたはメンテナンス操作のパフォーマンスが大きく向上するのは、パーティション化では珍しいことではありません。また、パーティション化によって、通常の管理タスクを大幅に簡略化できます。

また、パーティション化を使用して、データベースの設計者や管理者が、最先端のアプリケーションによってもたらされる難しい問題を解決することも可能になります。パーティション化は、数TBのシステムまたは非常に高い可用性を必要とするシステムを構築するための主要なツールです。

次の内容について説明します。

- パフォーマンス改善のためのパーティション化
- 管理性のためのパーティション化
- 可用性のためのパーティション化

2.2.1 パフォーマンス改善のためのパーティション化

パーティション化を使用してパフォーマンスを改善できます。

調査または操作対象のデータ量を制限し、パラレル実行を行うためにデータを分散することで、パーティション化によりパフォーマンスに関連する多くの利点がもたらされます。パーティション化の機能を次に示します。

- パフォーマンスのためのパーティション・プルーニング
- パフォーマンスのためのパーティション・ワイズ結合

2.2.1.1 パフォーマンスのためのパーティション・プルーニング

パーティション・プルーニングは非常に簡単で、パーティション化を使用してパフォーマンスを向上するための最も実質的な方法でもあります。

パーティション・プルーニングにより、問合せのパフォーマンスが大幅に向上します。たとえば、アプリケーションに注文の履歴レコードを含むOrders表があり、この表が週ごとにパーティション化されているとします。1週間の注文をリクエストする問合せでは、Orders表の単一のパーティションにのみアクセスします。Orders表に2年分の履歴データがある場合、この問合せでは104のパーティションではなく1つのパーティションにのみアクセスします。この問合せは、単純にパーティション・プルーニングの効果で、100倍高速に実行される可能性があります。

パーティション・プルーニングは、Oracle製品のすべてのパフォーマンス機能と連携します。パーティション・プルーニングは、索引や結合の技術またはパラレル・アクセスの手法とともに使用されます。

2.2.1.2 パフォーマンスのためのパーティション・ワイズ結合

パーティション化では、パーティション・ワイズ結合と呼ばれる技術が使用され、複数表の結合のパフォーマンスも向上します。

パーティション・ワイズ結合は、2つの表を結合する際に両方の表が結合キーでパーティション化される場合や、参照パーティション表を親表と結合する場合に適用できます。パーティション・ワイズ結合では、大きな結合が小さな結合に分割され、その分割が各パーティションで行われるため結合全体がこれまでよりも短い時間で完了します。これにより、シリアル実行およびパラレル実行の両方でパフォーマンスが大幅に向上します。

2.2.2 管理性のためのパーティション化

パーティション化により、表および索引をより細かく管理しやすい単位にパーティション化できるため、データベース管理者は分断攻略法でデータを管理できます。

パーティション化を使用すると、表の特定の部分に集中してメンテナンス操作を実行できます。たとえば、表全体ではなく表の単一のパーティションをバックアップできます。データベース・オブジェクト全体にメンテナンス操作を実行する場合には、それらの操作をパーティションごとに実行できるため、メンテナンス・プロセスをさらに管理しやすい単位に分割できます。

管理性を向上させるための一般的なパーティション化の使用方法は、データ・ウェアハウスでローリング・ウィンドウ・ロード・プロセスをサポートすることです。新規データを週単位で表にロードするとします。各パーティションに1週間分のデータが保存されるように、その表をパーティション化できます。ロード・プロセスは、パーティション交換ロードを使用した、単純な新規パーティションの追加です。その他のパーティションを変更する必要がないため、単一のパーティションの追加は、表全体を変更するよりはるかに効率的です。

2.2.3 可用性のためのパーティション化

パーティション化データベース・オブジェクトにより、パーティションの独立性が実現されます。パーティションの独立性のこの特性は、高可用性計画の重要な要素の一部です。

たとえば、パーティション表の1つのパーティションが使用できなくなっても、その表のその他すべてのパーティションはオンラインで使用可能なままです。アプリケーションは、その表の使用可能なパーティションに対して問合せやトランザクションを実行し続けることができ、使用できないパーティションにアクセスする必要がなければ、それらのデータベース操作は正常に実行されます。

データベース管理者は、各パーティションを別の表領域に格納するように指定できます。一般的には、これらの表領域は異なるストレージ層に格納されます。様々なパーティションをそれぞれ異なる表領域に格納すると、バックアップ操作やリカバリ操作を各パーティションに対して、表の他のパーティションとは独立して実行できます。このため、データベースのアクティブな部分はすぐに使用できるようになり、非アクティブなデータをリストアしながら、システムへのアクセスは継続できます。また、パーティション化により、スケジュールされる停止時間を短縮できます。パーティション化によってパフォーマンスが向上するため、大規模なデータベース・オ

ブジェクトのメンテナンス操作を比較的短いバッチ期間で終了できます。

2.3 パーティション化計画

Oracle Partitioningには、個々のパーティションにデータを配置する方法を制御する基本的なパーティション化計画として、3つの基本的なデータ配分方法が用意されています。

これらの計画を次に示します。

- レンジ
- ハッシュ
- リスト

これらのデータ分散方法を使用して、単一レベルまたはコンポジット・パーティション表として表をパーティション化できます。

- 単一レベル・パーティション化
- コンポジット・パーティション化

各パーティション化計画の利点と設計の指針は異なります。そのため、各計画にはそれぞれに適した状況があります。

2.3.1 単一レベル・パーティション化

単一レベル・パーティション化には、レンジ、ハッシュおよびリスト・パーティション化があります。

1つ以上の列をパーティション化キーとして使用し、次のデータ分散方法のいずれかを指定することで表を定義します。

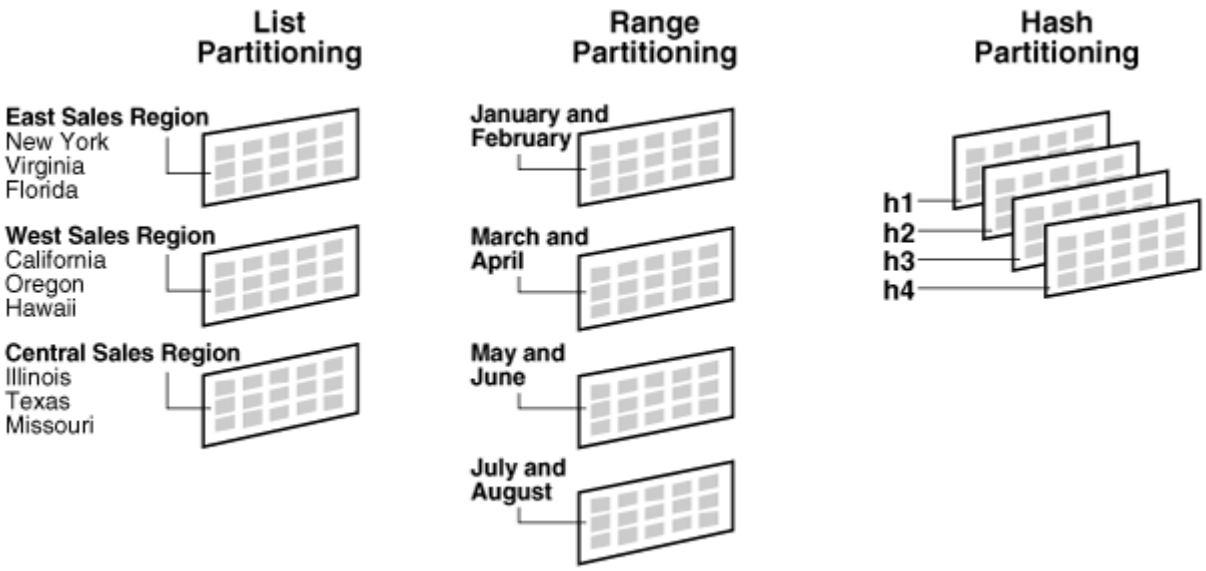
- レンジ・パーティション化
- ハッシュ・パーティション化
- リスト・パーティション化

たとえば、NUMBER型の列がパーティション化キーで、**less_than_five_hundred**および**less_than_one_thousand**という2つのパーティションのある表があるとします。**less_than_one_thousand**パーティションには、次の条件に一致する行が含まれています。

```
500 <= partitioning key < 1000
```

図2-2に、単一レベル・パーティション表の基本的なパーティション化計画を図で示します。

図2-2 リスト、レンジおよびハッシュ・パーティション化



2.3.1.1 レンジ・パーティション化

レンジ・パーティション化では、パーティションごとに設定するパーティション化キーの値範囲に基づいて、データがパーティションにマッピングされます。

レンジ・パーティション化は、最も一般的なタイプのパーティション化であり、通常は日付とともに使用されます。日付列がパーティション化キーの表では、**January-2017**パーティションには、パーティション化キーの値が**01-Jan-2017**から**31-Jan-2017**までの行が含まれます。

各パーティションには、パーティションの上限を指定するVALUES LESS THAN句(最上限値は含まない)があります。値がこのリテラル以上のパーティション化キーは、次に大きなパーティションに追加されます。最初のパーティションを除くすべてのパーティションには、前のパーティションのVALUES LESS THAN句によって指定された暗黙的な下限があります。

MAXVALUEリテラルは、最大のパーティションに定義できます。MAXVALUEは、NULL値も含み、そのパーティション化キーで可能なその他の値よりも大きい値を選別する仮想無限値を表します。

2.3.1.2 ハッシュ・パーティション化

ハッシュ・パーティション化では、ユーザーが指定するパーティション化キーに適用されるハッシング・アルゴリズムに基づいて、データがパーティションにマッピングされます。

ハッシング・アルゴリズムでは、パーティションの行が均等に分散され、パーティションはほぼ同じサイズになります。

ハッシュ・パーティション化は、データをデバイス全体に均等に分散する場合に理想的な方法です。また、ハッシュ・パーティション化は、特に、パーティション化するデータが履歴データではない場合や、パーティション化キーが明示的に設定されていない場合に、レンジ・パーティション化のかわりに簡単に使用できます。

ノート:

パーティション化によって使用されているハッシング・アルゴリズムは変更できません。

2.3.1.3 リスト・パーティション化

リスト・パーティション化では、各パーティションの説明にパーティション化キーの離散値のリストを指定することで、行のパーティションへのマッピング方法を明示的に制御できます。

リスト・パーティション化の利点は、順序付けおよび関連付けされていない一連のデータを自然にグループ化および編成できることです。パーティション化キーがリージョン列の表の場合、**East Sales Region**パーティションには**New York**、**Virginia**および**Florida**という値が含まれます。

DEFAULTパーティションでは、デフォルトのパーティションを使用するため、リスト・パーティション表に可能な値をすべて指定する必要がありません。そのため、その他のパーティションにマッピングされていないすべての行でエラーが発生しません。

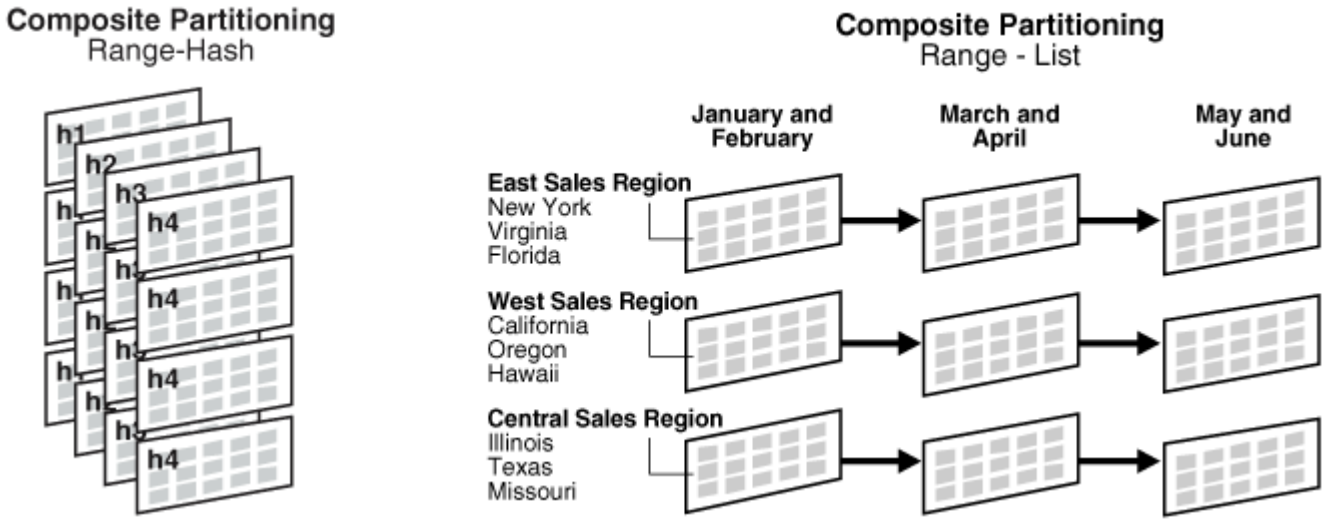
2.3.2 コンポジット・パーティション化

コンポジット・パーティション化は、基本的なデータ配分方法の組合せです。

コンポジット・パーティション化では、1つのデータ配分方法で表をパーティション化し、別のデータ配分方法を使用して、各パーティションをさらに小さなサブパーティションに分割します。1つのパーティションのすべてのサブパーティションは、データの論理的サブセットを表します。

コンポジット・パーティション化では、新規レンジ・パーティションの追加などの履歴操作がサポートされていますが、高レベルで可能なパーティション・プルーニングや、サブパーティションを使用した粒度の細かいデータ配置も提供されています。図2-3に、レンジ - ハッシュおよびレンジ - リスト・コンポジット・パーティション化の図を例として示します。

図2-3 コンポジット・レンジ - リスト・パーティション化



コンポジット・パーティション化のタイプは次のとおりです。

- コンポジット・レンジ - レンジ・パーティション化
- コンポジット・レンジ - ハッシュ・パーティション化
- コンポジット・レンジ - リスト・パーティション化
- コンポジット・リスト - レンジ・パーティション化
- コンポジット・リスト - ハッシュ・パーティション化
- コンポジット・リスト - リスト・パーティション化
- コンポジット・ハッシュ - ハッシュ・パーティション化
- コンポジット・ハッシュ - リスト・パーティション化
- コンポジット・ハッシュ - レンジ・パーティション化

2.3.2.1 コンポジット・レンジ - レンジ・パーティション化

コンポジット・レンジ - レンジ・パーティション化では、2つのディメンションに従って論理レンジ・パーティション化できます。

コンポジット・レンジ - レンジ・パーティション化の例は、order_dateによるパーティションおよびshipping_dateによるレンジ・サブパーティションです。

2.3.2.2 コンポジット・レンジ - ハッシュ・パーティション化

コンポジット・レンジ - ハッシュ・パーティション化では、レンジ・メソッドを使用してデータがパーティション化され、各パーティションはハッシュを使用してサブパーティション化されます。

コンポジット・レンジ - ハッシュ・パーティション化では、レンジ・パーティション化の管理性が向上し、ハッシュ・パーティション化のデータ配置、ストライプ化および並列性も向上します。

2.3.2.3 コンポジット・レンジ - リスト・パーティション化

コンポジット・レンジ - リスト・パーティション化では、レンジ・メソッドを使用してデータがパーティション化され、各パーティションはリストを使用してサブパーティション化されます。

コンポジット・レンジ - リスト・パーティション化では、レンジ・パーティション化の管理や、サブパーティションのリスト・パーティション化の明示的な制御が可能です。

2.3.2.4 コンポジット・リスト - レンジ・パーティション化

コンポジット・リスト - レンジ・パーティション化では、指定されたリスト・パーティション化計画内で論理レンジ・サブパーティション化できます。

コンポジット・リスト - レンジ・パーティション化の例は、country_idによるリスト・パーティションおよびorder_dateによるレンジ・サブパーティションです。

2.3.2.5 コンポジット・リスト - ハッシュ・パーティション化

コンポジット・リスト - ハッシュ・パーティション化では、リスト・パーティション化されたオブジェクトのハッシュ・サブパーティション化が可能です。

コンポジット・リスト - ハッシュ・パーティション化は、パーティション・ワイズ結合を可能にするために役立ちます。

2.3.2.6 コンポジット・リスト - リスト・パーティション化

コンポジット・リスト - リスト・パーティション化では、2つのディメンションに従って論理リスト・パーティション化できます。

コンポジット・リスト - リスト・パーティション化の例は、country_idによるリスト・パーティションおよびsales_channelによるリスト・サブパーティションです。

2.3.2.7 コンポジット・ハッシュ - ハッシュ・パーティション化

コンポジット・ハッシュ - ハッシュ・パーティション化は、2つのディメンションにハッシュ・パーティション化が可能です。

コンポジット・ハッシュ - ハッシュ・パーティション化技法は、2つのディメンションに従ったパーティション・ワイズ結合を可能にするために役立ちます。

2.3.2.8 コンポジット・ハッシュ - リスト・パーティション化

このトピックでは、コンポジット・ハッシュ - リスト・パーティション化を紹介します。

コンポジット・ハッシュ - リスト・パーティション化は、2つのディメンションにハッシュ・パーティション化が可能です。

2.3.2.9 コンポジット・ハッシュ - レンジ・パーティション化

このトピックでは、コンポジット・ハッシュ - レンジ・パーティション化を紹介します。

コンポジット・ハッシュ - レンジ・パーティション化は、2つのディメンションにハッシュ・パーティション化が可能です。

2.4 パーティション化の拡張

基本的なパーティション化計画に加え、Oracle Databaseにはパーティション化の拡張が用意されています。

Oracle Databaseには、次のタイプのパーティション化拡張が用意されています。

- 管理性の拡張
- パーティション化キーの拡張

2.4.1 管理性の拡張

このトピックでは、パーティション化の管理性の拡張を紹介します。

これらの拡張により、パーティション表の管理性が大幅に向上します。

- 時間隔パーティション化
- パーティション・アドバイザ

2.4.1.1 時間隔パーティション化

時間隔パーティション化は、レンジ・パーティション化の拡張機能です。

時間隔パーティション化は、表に挿入されたデータが既存のすべてのレンジ・パーティションを超える場合に、指定された間隔のパーティションを自動的に作成するようデータベースに指示します。少なくとも1つのレンジ・パーティションを指定してください。レンジ・パーティション化キーの値は、遷移ポイントと呼ばれるレンジ・パーティションの値の上限を決定します。値が遷移ポイントを超えるデータのために、データベースによって時間隔パーティションが作成されます。各時間隔パーティションの下限は、前の範囲つまり時間隔の上限であり、そのパーティションには含まれません。

たとえば、間隔が月単位の時間隔パーティション表を作成し、遷移ポイントを2007年1月に設定した場合、2007年1月の間隔の下限は2007年1月1日です。2007年7月の間隔の下限は、2007年6月のパーティションがすでに作成されているかどうかに関係なく2007年7月1日です。

単一レベルの時間隔パーティション表と次に示すコンポジット・パーティション表を作成できます。

- 時間隔 - レンジ
- 時間隔 - ハッシュ
- 時間隔 - リスト

時間隔パーティション化では、レンジ・パーティション化の機能のサブセットがサポートされます。

関連項目:

時間隔パーティション化を使用する際の制限事項の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

2.4.1.2 パーティション・アドバイザ

パーティション・アドバイザはSQLアクセス・アドバイザに含まれます。

パーティション・アドバイザにSQL文のワークロードを指定すると、それに基づいて表のパーティション化計画が提案されます。ワークロードは、SQLキャッシュまたはSQLチューニング・セットによって提供されます。あるいは、ユーザーが定義できます。

2.4.2 パーティション化キーの拡張

このトピックでは、パーティション化キーの拡張を紹介します。

これらの拡張により、パーティション化キーの定義の柔軟性が高まります。

- 参照パーティション化
- 仮想列ベースのパーティション化

2.4.2.1 参照パーティション化

参照パーティション化では、参照制約により相互に関連する2つの表のパーティション化が可能です。

パーティション化キーは、既存の親子関係を介して解決され、有効化されたアクティブな主キーおよび外部キー制約により強制されます。

この拡張の利点は、キー列を複製せずに親表からパーティション化キーを継承することで、親子関係にある複数の表を論理的に同一レベル・パーティション化できることです。論理的な依存性もパーティションのメンテナンス操作に自動的にカスケードされるため、アプリケーション開発が容易になり、ミスも少なくなります。

参照パーティション化の例として、参照制約orderid_refconstraintによって互いに関連するOrders表とLineItems表があります。つまり、LineItems.order_idがOrders.order_idを参照しています。Orders表は、order_dateでレンジ・パーティション化されています。LineItemsのorderid_refconstraintで参照パーティション化を行うと、図2-4および図2-5に示す、Orders表と同一レベルでパーティション化されたパーティション表が作成されることになります。

図2-4 参照パーティション化前

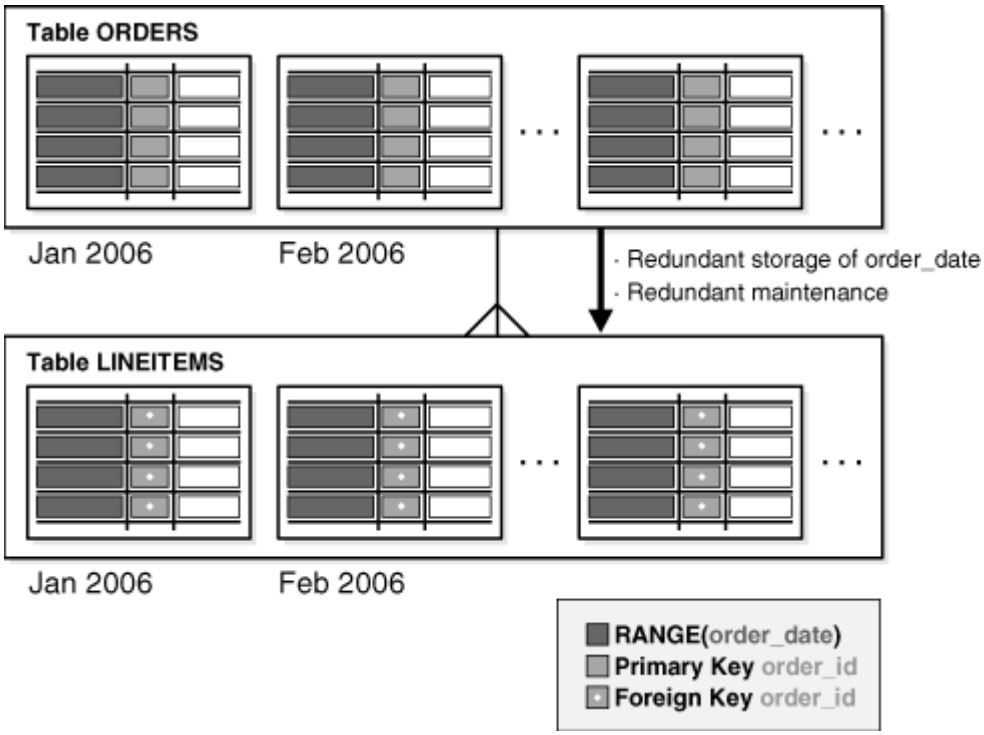
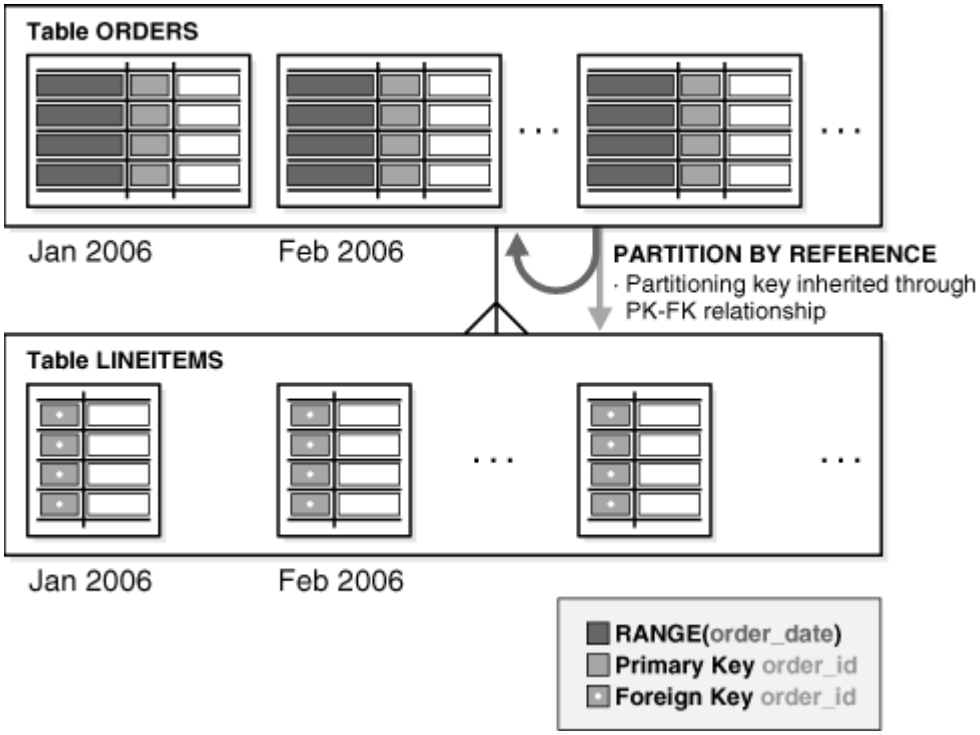


図2-5 参照パーティション化後



参照パーティション化では、基本的なパーティション化計画をすべて使用できます。参照パーティション化では、時間隔パーティション化も使用できます。

ノート:

参照パーティション化は、オンライン再定義パッケージ(DBMS_REDEFINITION)ではサポートされません。

2.4.2.2 仮想列ベースのパーティション化

Oracleのパーティション化には、仮想列で定義されたパーティション化計画が含まれます。

仮想列を使用することで、表の1つ以上の既存の列を使用し、パーティション化キーを式で定義できるようになりました。式はメタデータとしてのみ保存されています。たとえば、10桁の口座番号の上3桁に支店情報が含まれる場合があります。仮想列ベースのパーティション化という拡張機能により、ACCOUNT_ID列を含むACCOUNTS表を仮想(導出)列ACCOUNT_BRANCHを使用して拡張できます。ACCOUNT_BRANCHは、ACCOUNT_ID列の上3桁から導出され、この表のパーティション化キーになります。

仮想列ベースのパーティション化は、参照パーティション化、時間隔パーティション化および時間隔 - *コンポジット・パーティション化を含め、基本的なすべてのパーティション化計画でサポートされています。

2.5 パーティション表の索引

パーティション表の索引を非パーティション化またはパーティション化できます。

パーティション表と同じように、パーティション索引の管理性、可用性、パフォーマンスおよびスケーラビリティも向上しました。個別にパーティション化(グローバル索引)することも、表のパーティション化メソッド(ローカル索引)に自動的にリンクすることも可能です。一般的に、OLTPアプリケーションにはグローバル索引を使用し、データ・ウェアハウスまたは意思決定支援(DSS)アプリケーションにはローカル索引を使用します。

次の内容について説明します。

- 使用するパーティション索引の種類の決定

- ローカル・パーティション索引
- グローバル・パーティション索引
- グローバル非パーティション索引
- パーティション表に索引を作成する場合のその他の情報
- パーティション表の部分索引
- コンポジット・パーティションでのパーティション索引

2.5.1 使用するパーティション索引の種類の決定

使用するパーティション索引のタイプは、様々な要因を確認した後に選択する必要があります。

使用するパーティション索引の種類を決定する際には、次のガイドラインに順に従ってください。

1. 表パーティション化列が索引キーのサブセットである場合は、ローカル索引を使用します。この場合、作業はこれで終了です。そうでない場合には、ガイドライン2に進みます。
2. 索引が一意でパーティション化キー列が含まれていない場合は、グローバル索引を使用します。この場合、作業はこれで終了です。そうでない場合には、ガイドライン3に進みます。
3. 管理性を重視する場合は、ローカル索引を考慮します。この場合、作業はこれで終了です。そうでない場合には、ガイドライン4に進みます。
4. アプリケーションがOLTPでレスポンス時間を短くする必要がある場合は、グローバル索引を使用します。アプリケーションがDSSでスループットを重視する場合には、ローカル索引を使用します。

関連項目:

- パーティション索引、およびデータ・ウェアハウス環境で使用するタイプの決定方法の詳細は、「データ・ウェアハウス環境でのパーティション化の使用」を参照してください
- パーティション索引、およびオンライン・トランザクション処理環境で使用するタイプの決定方法の詳細は、「オンライン・トランザクション処理環境でのパーティション化の使用」を参照してください

2.5.2 ローカル・パーティション索引

次に示すように、ローカル・パーティション索引は、他の種類のパーティション索引よりも管理が容易です。

また、可用性が非常に高く、DSS環境で一般的です。これは同一レベル・パーティション化されているためです。つまり、ローカル索引の各パーティションは、表の1つのパーティションに一対一で関連付けられています。このため、索引パーティションと表パーティションの同期を自動的に維持でき、表と索引の各ペアが独立した状態になります。あるパーティションのデータを無効または使用不可にするアクションが発生しても、影響を受けるのは1つのパーティションだけです。

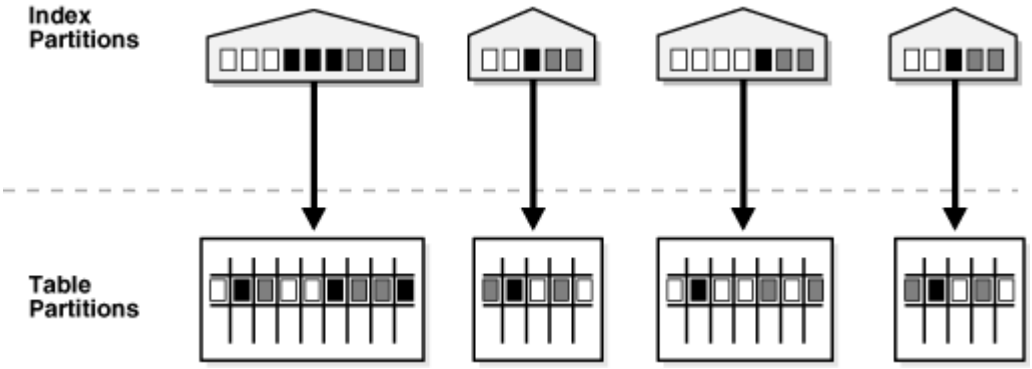
ローカル・パーティション索引を使用すると、表でパーティションまたはサブパーティションのメンテナンス操作が行われる場合の可用性が高まります。ローカル非同一キー索引と呼ばれるタイプの索引は、履歴データベースに非常に便利です。このタイプの索引では、パーティション化は索引列の左側の接頭辞では行われません。

パーティションは、明示的にローカル索引に追加することはできません。基礎となる表にパーティションを追加した場合のみ、新しいパーティションがローカル索引に追加されます。同様に、パーティションを明示的にローカル索引から削除することもできません。かわりに、基礎となる表からパーティションを削除した場合にのみ、ローカル索引パーティションが削除されます。

ローカル索引は一意索引にできます。ただし、ローカル索引を一意にするためには、表のパーティション化キーが索引のキー列の一部である必要があります。

図2-6に、ローカル・パーティション索引を図で示します。

図2-6 ローカル・パーティション索引



関連項目:

- 同一キー索引の詳細は、「索引のパーティション化」を参照してください
- ローカル・パーティション索引の詳細は、「ローカル・パーティション索引」を参照してください

2.5.3 グローバル・パーティション索引

このトピックでは、グローバル・パーティション索引について説明します。

Oracleでは、グローバル・レンジ・パーティション索引とグローバル・ハッシュ・パーティション索引が提供されます。これらについて次に説明します。

- グローバル・レンジ・パーティション索引
- グローバル・ハッシュ・パーティション索引
- グローバル・パーティション索引のメンテナンス

2.5.3.1 グローバル・レンジ・パーティション索引

グローバル・レンジ・パーティション索引は、パーティション化の程度とパーティション化キーが表のパーティション化メソッドから独立しているため柔軟です。

グローバル索引の最高位パーティションにはパーティション・バウンドが必要で、値はすべてMAXVALUEです。これにより、基礎となる表のすべての行を索引に含めることができます。グローバル同一キー索引は、一意索引にすることも一意ではない索引にすることもできます。

最高位パーティションにはMAXVALUEというパーティション・バウンドがあるため、グローバル索引にはパーティションを追加できません。新しい最高位パーティションを追加するには、ALTER INDEX SPLIT PARTITION文を使用します。グローバル索引パーティションが空の場合は、ALTER INDEX DROP PARTITION文を発行することでそのパーティションを明示的に削除できます。グローバル索引パーティションにデータが含まれる場合は、パーティションを削除すると次の最高位パーティションが使用不可とマークされる原因になります。グローバル索引では、最高位パーティションを削除できません。

2.5.3.2 グローバル・ハッシュ・パーティション索引

グローバル・ハッシュ・パーティション索引では、索引が増え続けた場合に競合が分散され、パフォーマンスが向上します。

つまり、ほとんどの索引の挿入は、グローバル・ハッシュ・パーティション索引のNハッシュ・パーティションで均一に分散される索引の右端でのみ発生します。

2.5.3.3 グローバル・パーティション索引のメンテナンス

このトピックでは、グローバル・パーティション索引のメンテナンスについて説明します。

デフォルトでは、ヒープ構成表のパーティションに次の操作を行うと、グローバル索引すべてが使用不可とマークされます。

```
ADD (HASH)
COALESCE (HASH)
DROP
EXCHANGE
MERGE
MOVE
SPLIT
TRUNCATE
```

これらの索引は、操作用のSQL文にUPDATE INDEXES句を追加することでメンテナンスできます。ただし、UPDATE INDEXES句を追加すると、パーティション・メンテナンス操作の一部としてグローバル索引がメンテナンスされ、操作の実行時間が延長されてリソース要件が増える可能性があるので注意してください。

グローバル索引のメンテナンスには、次の2つの利点があります。

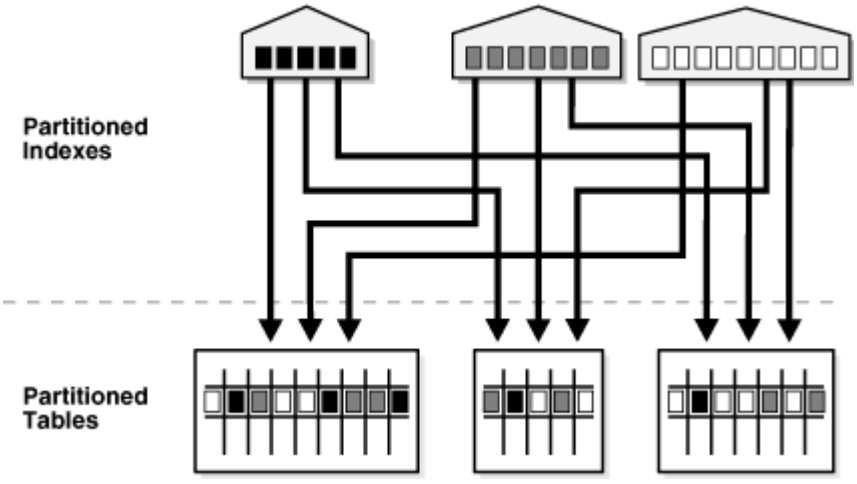
- 操作中、索引が使用可能かつオンラインに保たれます。そのため、その他のアプリケーションがこの操作に影響されません。
- 操作後に索引を再作成する必要がありません。
- DROPおよびTRUNCATEのグローバル索引メンテナンスは、メタデータのみの操作として実装されます。

ノート:

この機能は、ヒープ構成表でのみサポートされています。

図2-7に、グローバル・パーティション索引を図で示します。

図2-7 グローバル・パーティション索引



関連項目:

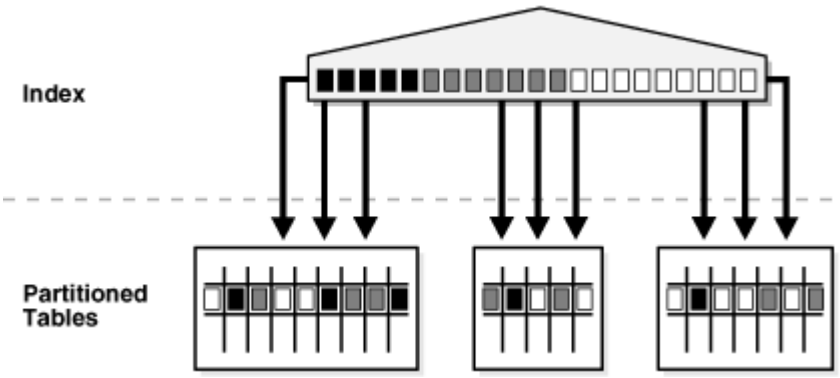
グローバル・パーティション索引の詳細は、「グローバル・パーティション索引」を参照してください

2.5.4 グローバル非パーティション索引

グローバル非パーティション索引は、ローカルの非パーティション索引と同様に動作します。

図2-8に、グローバル非パーティション索引を図で示します。

図2-8 グローバル非パーティション索引



2.5.5 パーティション表に索引を作成する場合のその他の情報

いくつかの制約がありますが、パーティション表でビットマップ索引を作成できます。

ビットマップ索引は、そのパーティション表に対してしか使用できなくする必要があります。グローバル索引にはできません。

グローバル索引は一意索引にすることができます。ローカル索引は、パーティション化キーが索引キーの一部である場合、一意索引にしかできません。

2.5.6 パーティション表の部分索引

表のパーティションのサブセットにローカルおよびグローバル索引を作成して、索引の作成の柔軟性を高めることができます。

この機能は、デフォルトの表索引付けプロパティを使用してサポートされます。表が作成または変更される場合、デフォルト索引付けプロパティを表またはそのパーティションに指定できます。表索引付けプロパティは、部分索引にのみ考慮されます。

索引が表にPARTIALとして作成される場合:

- ローカル索引: 索引付けが表パーティションに有効な場合に使用でき、無効な場合に使用できない索引パーティションが作成されます。索引または索引パーティション・レベルでUSABLE/UNUSABLEを指定して、この動作を上書きできます。
- グローバル索引: 索引付けが有効なパーティションのみを含み、その他は除外します。

この機能は、一意の索引または一意の制約の施行に使用される索引にサポートされません。FULLおよびPARTIALの両方が指定されない場合、FULLがデフォルトです。

デフォルトでは、索引がFULL索引として作成され、表索引付けプロパティから索引を分離します。

INDEXING句は、パーティションおよびサブパーティション・レベルで指定することもできます。

次のSQL DDLは、次の項目で表を作成します。

- パーティションORD_P1およびORD_P3がすべての部分グローバル索引に含まれます
- 上の2つの表パーティションに対応するローカル索引パーティション(PARTIALで作成された索引用)が作成され、デフォルトで使用できます。

- 他のパーティションはすべての部分グローバル索引から除外され、ローカル索引で使用不可で作成されます(PARTIALで作成された索引用)。

```
CREATE TABLE orders (
  order_id NUMBER(12),
  order_date DATE CONSTRAINT order_date_nn NOT NULL,
  order_mode VARCHAR2(8),
  customer_id NUMBER(6) CONSTRAINT order_customer_id_nn NOT NULL,
  order_status NUMBER(2),
  order_total NUMBER(8,2),
  sales_rep_id NUMBER(6),
  promotion_id NUMBER(6),
  CONSTRAINT order_mode_lov CHECK (order_mode in ('direct','online')),
  CONSTRAINT order_total_min CHECK (order_total >= 0))
   INDEXING OFF
   PARTITION BY RANGE (ORDER_DATE)
   (PARTITION ord_p1 VALUES LESS THAN (TO_DATE('01-MAR-1999','DD-MON-YYYY'))
     INDEXING ON,
   PARTITION ord_p2 VALUES LESS THAN (TO_DATE('01-JUL-1999','DD-MON-YYYY'))
     INDEXING OFF,
   PARTITION ord_p3 VALUES LESS THAN (TO_DATE('01-OCT-1999','DD-MON-YYYY'))
     INDEXING ON,
   PARTITION ord_p4 VALUES LESS THAN (TO_DATE('01-MAR-2000','DD-MON-YYYY')),
   PARTITION ord_p5 VALUES LESS THAN (TO_DATE('01-MAR-2010','DD-MON-YYYY')));
```

INDEXING PARTIAL句を指定する前のSQL例の表索引付けプロパティに従って、ローカルまたはグローバル部分索引を作成できます。

```
CREATE INDEX ORDERS_ORDER_TOTAL_GIDX ON ORDERS (ORDER_TOTAL)
   GLOBAL INDEXING PARTIAL;
```

ORDERS_ORDER_TOTAL_GIDX索引が作成され、INDEXING ONを指定したパーティションのみ索引付けされ、残りのパーティションは除外されます。

ビューの更新には、次の内容が含まれます。

- 表索引付けプロパティ - 列INDEXINGが*_PART_TABLES、*_TAB_PARTITIONSおよび*_TAB_SUBPARTITIONSビューに追加されます。

  この列は、索引付けの有効または無効を指定する2つの値ONおよびOFFのいずれかを含みます。

- 索引レベル・プロパティとしての部分グローバル索引 - 新しい列INDEXINGがUSER_INDEXESビューに追加されます。この列をFULLまたはPARTIALに設定できます。

- 部分グローバル索引の最適化 - グローバル索引(パーティション)がDROP/TRUNCATE PARTITIONまたはMODIFY PARTITION INDEXING OFFの実行中に遅延した索引メンテナンスのために古いエントリを含む場合、列ORPHANED_ENTRIESがディクショナリ・ビューUSER_INDEXESおよびUSER_IND_PARTITIONSに追加されて表示されます。列には、次のいずれかの値を含むことができます。

  - YES => 索引(パーティション)は親がないエントリを含みます
  - NO => 索引(パーティション)は親がないエントリを含みません

関連項目:

データベース・ビューの詳細は、『Oracle Databaseリファレンス』を参照してください

2.5.7 コンポジット・パーティションでのパーティション索引

コンポジット・パーティションでのパーティション索引についていくつかの留意点があります

コンポジット・パーティションでパーティション索引を使用する場合は、次のことに注意してください。

- サブパーティション索引は常にローカルで、デフォルトでは表のサブパーティションとともに格納されます。
- 表領域は、索引または索引サブパーティション・レベルで指定できます。

3 可用性、管理性およびパフォーマンスのためのパーティション化

パーティション化により、可用性、管理性およびパフォーマンスが実現されます。

この章では、パーティション化によって、可用性、管理性およびパフォーマンスの効果がどのように上がるかについて概要を説明します。与えられたパーティション化計画を使用する場合のガイドラインが用意されています。表のパーティション化の使用を中心に説明しますが、ほとんどの推奨事項および考慮事項は索引のパーティション化にも当てはまります。

この章の構成は、次のとおりです。

- パーティション・プルーニング
- パーティション・ワイズ操作
- 索引のパーティション化
- パーティション化と表の圧縮
- パーティション化戦略の選択に関する推奨事項

3.1 パーティション・プルーニング

パーティション・プルーニングは、データ・ウェアハウスの重要なパフォーマンス機能です。

パーティション・プルーニングでは、オプティマイザによりSQL文のFROM句とWHERE句が分析され、パーティション・アクセス・リストを構築するときに不要なパーティションが削除されます。この機能によって、Oracle Databaseは、SQL文に関連するパーティションに限定して処理を実行できるようになります。

次の内容について説明します。

- パーティション・プルーニングの利点
- パーティション・プルーニングに使用できる情報
- パーティション・プルーニングが使用されたかどうかを識別する方法
- 静的パーティション・プルーニング
- 動的パーティション・プルーニング
- ゾーン・マップを使用するパーティション・プルーニング
- パーティション・プルーニングのヒント

3.1.1 パーティション・プルーニングの利点

パーティション・プルーニングを行うと、ディスクから取得するデータ容量が大幅に削減され、処理時間が短縮します。このため、問合せのパフォーマンスが向上し、リソース使用率が最適化されます。

グローバルなパーティション索引を使用して異なる列の索引および表をパーティション化する場合は、基礎となる表が排除できない場合でもパーティション・プルーニングにより索引パーティションが排除されます。

実際のSQL文に応じて、Oracle Databaseにより静的プルーニングおよび動的プルーニングが使用されます。静的プルーニングは、事前にアクセスしたパーティションの情報を使用してコンパイル時に行われます。動的プルーニングは実行時に行われます。つまり、文がアクセスする正確なパーティションは事前にはわかりません。静的プルーニングのサンプルの使用例は、パーティション・キー列に固定リテラルがあるWHERE条件を含むSQL文です。動的プルーニングの例は、WHERE条件内の演算子または関数の使用です。

パーティション・プルーニングは、プルーニングが行われるオブジェクトの統計に影響します。また文の実行計画にも影響します。

3.1.2 パーティション・プルーニングに使用できる情報

パーティション・プルーニングは、パーティション化列で実行できます。

Oracle Databaseでパーティション・プルーニングが行われるのは、レンジ・パーティション化列またはリスト・パーティション化列で、範囲述語、LIKE述語、等価述語、INリスト述語を使用したとき、またハッシュ・パーティション化列で等価述語およびINリスト述語を使用したときです。

コンポジット・パーティション化されたオブジェクトでは、Oracle Databaseは関連する述語を使用して両方のレベルでプルーニングを実行できます。たとえば、表sales_range_hashを確認してください。これは、例3-1に示すように、列s_saledateでレンジ・パーティション化され、列s_productidでハッシュ・サブパーティション化されています。

パーティション列に対する述語を使用して、次のようにパーティション・プルーニングが行われます。

- レンジ・パーティション化を使用して、1999年の第3四半期と第4四半期に相当するパーティションsal99q2とsal99q3

のみにアクセスします。

- ハッシュ・サブパーティション化を使用して、各パーティションで、s_productid=1200の行を格納する1つのサブパーティションのみにアクセスします。サブパーティションと述語のマッピングは、Oracle内部のハッシュ分散関数に基づいて計算されます。

参照パーティション表では、参照表の結合を介してパーティション・プルーニングを利用できます。仮想列に基づくパーティション表では、仮想列定義式を使用するSQL文でパーティション・プルーニングを利用できます。

例3-1 パーティション・プルーニングによる表の作成

```
CREATE TABLE sales_range_hash(
  s_productid  NUMBER,
  s_saledate   DATE,
  s_custid     NUMBER,
  s_totalprice NUMBER)
PARTITION BY RANGE (s_saledate)
SUBPARTITION BY HASH (s_productid) SUBPARTITIONS 8
 (PARTITION sal99q1 VALUES LESS THAN
   (TO_DATE('01-APR-1999', 'DD-MON-YYYY')),
  PARTITION sal99q2 VALUES LESS THAN
   (TO_DATE('01-JUL-1999', 'DD-MON-YYYY')),
  PARTITION sal99q3 VALUES LESS THAN
   (TO_DATE('01-OCT-1999', 'DD-MON-YYYY')),
  PARTITION sal99q4 VALUES LESS THAN
   (TO_DATE('01-JAN-2000', 'DD-MON-YYYY')));

SELECT * FROM sales_range_hash
WHERE s_saledate BETWEEN (TO_DATE('01-JUL-1999', 'DD-MON-YYYY'))
  AND (TO_DATE('01-OCT-1999', 'DD-MON-YYYY')) AND s_productid = 1200;
```

3.1.3 パーティション・プルーニングが使用されたかどうかを識別する方法

Oracleがパーティション・プルーニングを使用するかどうかは、EXPLAIN PLAN文の計画表または共有SQL領域にある文の実行計画に示されます。

パーティション・プルーニング情報は、計画列PSTART(PARTITION_START)およびPSTOP(PARTITION_STOP)に反映されます。シリアル文の場合、プルーニング情報はOPERATION列とOPTIONS列にも反映されます。

関連項目:

EXPLAIN PLANとその解釈方法の詳細は、Oracle Database SQLチューニング・ガイドを参照してください

3.1.4 静的パーティション・プルーニング

Oracleでは、主に静的述語に基づいて、静的プルーニングを使用する時期が決定されます。

多くの場合、Oracleはコンパイル時にアクセスするパーティションを決定します。静的パーティション・プルーニングが行われるのは、静的な述語を使用したときですが、次の場合を除きます。

- 副問合せの結果を使用してパーティション・プルーニングが行われる場合。
- オプティマイザがスター型変換を使用して問合せを再作成し、スター型変換の後でプルーニングが行われる場合。

- 最も有効な実行計画がネステッド・ループである場合。

これら3つの場合には、動的プルーニングが使用されます。

解析時にOracleが、隣接するどのパーティション・セットがアクセスされるかを識別できると、実行計画のPSTART列とPSTOP列に、アクセスされるパーティションの開始値と終了値が示されます。動的プルーニングを含む他のパーティション・プルーニングでは、PSTARTおよびPSTOPのKEY値と、オプションでその他の属性が表示されます。

次に、例を示します。

```
SQL> explain plan for select * from sales where time_id = to_date('01-jan-2001', 'dd-mon-yyyy');
Explained.

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
Plan hash value: 3971874201
------------------------------------------------------------------------------------------------
| Id | Operation              | Name  | Rows | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
------------------------------------------------------------------------------------------------
|  0 | SELECT STATEMENT       |       | 673  | 19517 | 27      (8)| 00:00:01 |       |       |
|  1 |  PARTITION RANGE SINGLE|       | 673  | 19517 | 27      (8)| 00:00:01 | 17    | 17    |
|* 2 |   TABLE ACCESS FULL    | SALES | 673  | 19517 | 27      (8)| 00:00:01 | 17    | 17    |
------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("TIME_ID"=TO_DATE('2001-01-01 00:00:00', 'yyyy-mm-dd hh24:mi:ss'))
```

この計画では、PSTART列とPSTOP列に示されているようにOracleがパーティション番号17にアクセスします。OPERATION列にPARTITION RANGE SINGLEとあり、1つのパーティションのみにアクセスすることを示します。OPERATIONにPARTITION RANGE ALLと表示されている場合は、すべてのパーティションにアクセスし、実質的にはプルーニングは行われません。このとき、PSTARTには表の最初のパーティションが示され、PSTOPには最後のパーティションが示されます。

時間隔パーティション表の全表スキャンを含む実行計画では、作成された時間隔パーティションの数にかかわらず、PSTARTに1、PSTOPに1048575が示されます。

3.1.5 動的パーティション・プルーニング

このトピックでは、Oracleの動的パーティション・プルーニングを紹介します。

動的プルーニングが行われるのは、プルーニングが可能で、静的プルーニングが不可能な場合です。次に動的プルーニングのいくつかの例を示します。

- バインド変数を含む動的プルーニング
- 副問合せを含む動的プルーニング
- スター型変換を含む動的プルーニング
- ネステッド・ループ結合を含む動的プルーニング

3.1.5.1 バインド変数を含む動的プルーニング

パーティション列に対してバインド変数を使用する文では、動的プルーニングが行われます。

次にSQL文の例を示します。

```
SQL> explain plan for select * from sales s where time_id in ( :a, :b, :c, :d);
Explained.
```

```
SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------
Plan hash value: 513834092
-----------------------------------------------------------------------------------------------------------
| Id | Operation                          |    Name |Rows|Bytes|Cost (%CPU)|  Time  | Pstart| Pstop|
-----------------------------------------------------------------------------------------------------------
|  0 | SELECT STATEMENT                   |         |2517|72993|    292 (0)|00:00:04|       |      |
|  1 |  INLIST ITERATOR                   |         |    |     |           |        |       |      |
|  2 |   PARTITION RANGE ITERATOR         |         |2517|72993|    292 (0)|00:00:04|KEY(I) |KEY(I)|
|  3 |    TABLE ACCESS BY LOCAL INDEX ROWID| SALES |2517|72993|    292 (0)|00:00:04|KEY(I) |KEY(I)|
|  4 |     BITMAP CONVERSION TO ROWIDS    |         |    |     |           |        |       |      |
|* 5 |      BITMAP INDEX SINGLE VALUE     |SALES_TIME_BIX| |   |           |        |KEY(I) |KEY(I)|
-----------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
5 - access("TIME_ID"=:A OR "TIME_ID"=:B OR "TIME_ID"=:C OR "TIME_ID"=:D)
```

パラレル実行計画では、次の例のようにpstart列とpstop列のみにパーティション・プルーニング情報が含まれ、operation列にはパラレル操作の情報が含まれます。

```
SQL> explain plan for select * from sales where time_id in (:a, :b, :c, :d);
Explained.

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------
Plan hash value: 4058105390
-----------------------------------------------------------------------------------------------------------
| Id| Operation          | Name  |Rows|Bytes|Cost(%CP|  Time  |Pstart| Pstop|  TQ |INOUT| PQ Dis|
-----------------------------------------------------------------------------------------------------------
|  0| SELECT STATEMENT   |       |2517|72993|  75(36)|00:00:01|      |      |     |     |       |
|  1|  PX COORDINATOR    |       |    |     |        |        |      |      |     |     |       |
|  2|  PX SEND QC(RANDOM)|:TQ10000|2517|72993| 75(36)|00:00:01|      |      |Q1,00| P->S|QC(RAND|
|  3|   PX BLOCK ITERATOR|       |2517|72993|  75(36)|00:00:01|KEY(I)|KEY(I)|Q1,00| PCWC|       |
|* 4|   TABLE ACCESS FULL| SALES |2517|72993|  75(36)|00:00:01|KEY(I)|KEY(I)|Q1,00| PCWP|       |
-----------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
  4 - filter("TIME_ID"=:A OR "TIME_ID"=:B OR "TIME_ID"=:C OR "TIME_ID"=:D)
```

関連項目:

EXPLAIN PLANとその解釈方法の詳細は、Oracle Database SQLチューニング・ガイドを参照してください

3.1.5.2 副問合せを含む動的プルーニング

パーティション列に対して明示的に副問合せを使用する文では、動的プルーニングが行われます。

次にSQL文の例を示します。

```
SQL> explain plan for select sum(amount_sold) from sales where time_id in
     (select time_id from times where fiscal_year = 2000);
Explained.
```

```
SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------
Plan hash value: 3827742054

------------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |       |     1 |    25 |   523   (5)| 00:00:07 |       |       |
|   1 |  SORT AGGREGATE            |       |     1 |    25 |            |          |       |       |
|*  2 |   HASH JOIN                |       |   191K|  4676K|   523   (5)| 00:00:07 |       |       |
|*  3 |    TABLE ACCESS FULL       | TIMES |   304 |  3648 |    18   (0)| 00:00:01 |       |       |
|   4 |    PARTITION RANGE SUBQUERY|       |   918K|    11M|   498   (4)| 00:00:06 |KEY(SQ)|KEY(SQ)|
|   5 |     TABLE ACCESS FULL      | SALES |   918K|    11M|   498   (4)| 00:00:06 |KEY(SQ)|KEY(SQ)|
------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("TIME_ID"="TIME_ID")
   3 - filter("FISCAL_YEAR"=2000)
```

関連項目:

EXPLAIN PLANとその解釈方法の詳細は、Oracle Database SQLチューニング・ガイドを参照してください

3.1.5.3 スター型変換を含む動的プルーニング

データベースによってスター型変換を使用して変換される文では、動的なプルーニングが行われます。

次にSQL文の例を示します。

```
SQL> explain plan for select p.prod_name, t.time_id, sum(s.amount_sold)
     from sales s, times t, products p
     where s.time_id = t.time_id and s.prod_id = p.prod_id and t.fiscal_year = 2000
     and t.fiscal_week_number = 3 and p.prod_category = 'Hardware'
     group by t.time_id, p.prod_name;
Explained.

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------
Plan hash value: 4020965003

------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name                | Rows  | Bytes | Pstart| Pstop |
------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                     |     1 |    79 |       |       |
|   1 |  HASH GROUP BY                      |                     |     1 |    79 |       |       |
|*  2 |   HASH JOIN                         |                     |     1 |    79 |       |       |
|*  3 |    HASH JOIN                        |                     |     2 |    64 |       |       |
|*  4 |     TABLE ACCESS FULL               | TIMES               |     6 |    90 |       |       |
|   5 |     PARTITION RANGE SUBQUERY        |                     |   587 |  9979 |KEY(SQ)|KEY(SQ)|
|   6 |      TABLE ACCESS BY LOCAL INDEX ROWID| SALES             |   587 |  9979 |KEY(SQ)|KEY(SQ)|
|   7 |       BITMAP CONVERSION TO ROWIDS   |                     |       |       |       |       |
```

```
|   8 |        BITMAP AND                     |                        |       |       |       |       |
|   9 |         BITMAP MERGE                  |                        |       |       |       |       |
|  10 |          BITMAP KEY ITERATION         |                        |       |       |       |       |
|  11 |           BUFFER SORT                 |                        |       |       |       |       |
|* 12 |            TABLE ACCESS FULL          | TIMES                  |     6 |    90 |       |       |
|* 13 |           BITMAP INDEX RANGE SCAN     | SALES_TIME_BIX         |       |       |KEY(SQ)|KEY(SQ)|
|  14 |         BITMAP MERGE                  |                        |       |       |       |       |
|  15 |          BITMAP KEY ITERATION         |                        |       |       |       |       |
|  16 |           BUFFER SORT                 |                        |       |       |       |       |
|  17 |            TABLE ACCESS BY INDEX ROWID| PRODUCTS               |    14 |   658 |       |       |
|* 18 |             INDEX RANGE SCAN          | PRODUCTS_PROD_CAT_IX   |    14 |       |       |       |
|* 19 |           BITMAP INDEX RANGE SCAN     | SALES_PROD_BIX         |       |       |KEY(SQ)|KEY(SQ)|
|  20 |    TABLE ACCESS BY INDEX ROWID        | PRODUCTS               |    14 |   658 |       |       |
|* 21 |     INDEX RANGE SCAN                  | PRODUCTS_PROD_CAT_IX   |    14 |       |       |       |
-------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("S"."PROD_ID"="P"."PROD_ID")
   3 - access("S"."TIME_ID"="T"."TIME_ID")
   4 - filter("T"."FISCAL_WEEK_NUMBER"=3 AND "T"."FISCAL_YEAR"=2000)
  12 - filter("T"."FISCAL_WEEK_NUMBER"=3 AND "T"."FISCAL_YEAR"=2000)
  13 - access("S"."TIME_ID"="T"."TIME_ID")
  18 - access("P"."PROD_CATEGORY"='Hardware')
  19 - access("S"."PROD_ID"="P"."PROD_ID")
  21 - access("P"."PROD_CATEGORY"='Hardware')

Note
-----
   - star transformation used for this statement
```

ノート:

Cost (%CPU)列とTime列は、この例の計画表の出力では削除されています。

関連項目:

EXPLAIN PLANとその解釈方法の詳細は、Oracle Database SQLチューニング・ガイドを参照してください

3.1.5.4 ネステッド・ループ結合を含む動的プルーニング

ネステッド・ループ結合を使用すると最も効率的に実行される文では、動的プルーニングが使用されます。

次にSQL文の例を示します。

```
SQL> explain plan for select t.time_id, sum(s.amount_sold)
     from sales s, times t
     where s.time_id = t.time_id and t.fiscal_year = 2000 and t.fiscal_week_number = 3
     group by t.time_id;
Explained.

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
```

```
-------------------------------------------------------------------------------------------------------------
Plan hash value: 50737729

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |       |     6 |   168 |   126   (4)| 00:00:02 |       |       |
|   1 |  HASH GROUP BY             |       |     6 |   168 |   126   (4)| 00:00:02 |       |       |
|   2 |   NESTED LOOPS             |       |  3683 |   100K|   125   (4)| 00:00:02 |       |       |
|*  3 |    TABLE ACCESS FULL       | TIMES |     6 |    90 |    18   (0)| 00:00:01 |       |       |
|   4 |    PARTITION RANGE ITERATOR|       |   629 |  8177 |    18   (6)| 00:00:01 |   KEY |   KEY |
|*  5 |     TABLE ACCESS FULL      | SALES |   629 |  8177 |    18   (6)| 00:00:01 |   KEY |   KEY |
-------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("T"."FISCAL_WEEK_NUMBER"=3 AND "T"."FISCAL_YEAR"=2000)
   5 - filter("S"."TIME_ID"="T"."TIME_ID")
```

関連項目:

EXPLAIN PLANとその解釈方法の詳細は、Oracle Database SQLチューニング・ガイドを参照してください

3.1.6 ゾーン・マップを使用するパーティション・プルーニング

パーティション・プルーニングが拡張され、完全なパーティションのプルーニングにゾーン・マップを利用できるようになりました。拡張されたプルーニング機能を提供することで、リソース消費が削減され、情報を得るまでの時間が短縮され、パフォーマンスが向上します。

ゾーン・マップは、表に対して作成できる、独立したアクセス構造です。表スキャン中にゾーン・マップを使用して、表のディスク・ブロックおよびパーティション表のパーティションを、表の列の述語に基づいてプルーニングできます。ゾーン・マップは、パーティション表のパーティション・キー列と相関関係がないため、ゾーン・マップを持つパーティション表の文では、非パーティション・キー列に基づいてパーティションをプルーニングできます。

関連項目:

ゾーン・マップおよび属性クラスタリングの詳細は、『*Oracle Database*データ・ウェアハウス・ガイド』を参照してください。

ゾーン・マップを使用するパーティション・プルーニングは、ゾーン・マップ列の値とパーティション・キー列の値に相関関係がある場合に特に有効です。たとえば、同じパーティション表内のパーティション・キー列「受注日」と相関関係のある「出荷日」など、パーティション表自体の列間、またはパーティション表のパーティション・キー列「日」と相関関係のあるディメンション表「時間」の結合ゾーン・マップ列「月の説明」など、結合ゾーン・マップ列およびパーティション表には、相関関係があります。

例3-2に、パーティション表の相関関係のある列での、ゾーン・マップを使用するパーティション・プルーニングを示します。パーティション表sales_rangeの列s_shipdateはパーティション・キー列order_dateと相関関係があります。受注は一般に、受注を受けとった日から数日以内に出荷されるためです。

s_shipdateとパーティション・キー列に相関関係があるため、列がパーティション・キーの一部でなくても、この列の選択的な述

語でパーティション表sales_rangeをパーティション・プルーニングできる見込みが高くなります。

次のSELECT文は、1999年の第1四半期に出荷されたすべての受注を検索します。

```
SELECT * FROM sales_range
      WHERE s_shipdate BETWEEN to_date('01/01/1999','dd/mm/yyyy')
      AND to_date('03/01/1999','mm/dd/yyyy');
```

前述のSELECT文の次の実行計画では、ゾーン・マップがパーティション・プルーニングに使用され、アクセスされるパーティションからブロックをプルーニングするためにも使用されます。

ゾーン・マップを使用するパーティション・プルーニングは、実行計画のPSTARTおよびPSTOP列にあるKEY(ZM)によって識別されます。アクセスされるすべてのパーティションのブロック・レベルのプルーニングは、表アクセス時間のフィルタ述語によって識別されます(id 2)。

例3-2 属性クラスタリングのあるパーティション表sales_rangeおよび相関する列のゾーン・マップ

```
CREATE TABLE sales_range(
  s_productid      NUMBER,
  s_saledate       DATE,
  s_shipdate       DATE,
  s_custid         NUMBER,
  s_totalprice     NUMBER)
CLUSTERING BY (s_shipdate)
WITH MATERIALIZED ZONEMAP
PARTITION BY RANGE (s_saledate)
 (PARTITION sal99q1 VALUES LESS THAN
   (TO_DATE('01-APR-1999', 'DD-MON-YYYY')),
  PARTITION sal99q2 VALUES LESS THAN
   (TO_DATE('01-JUL-1999', 'DD-MON-YYYY')),
  PARTITION sal99q3 VALUES LESS THAN
   (TO_DATE('01-OCT-1999', 'DD-MON-YYYY')),
  PARTITION sal99q4 VALUES LESS THAN
   (TO_DATE('01-JAN-2000', 'DD-MON-YYYY')));
```

例3-3 ゾーン・マップを使用するパーティション・プルーニングの実行計画

```
-------------------------------------------------------------------------------------------------------
---------
| Id  | Operation                         | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
Pstart| Pstop |
-------------------------------------------------------------------------------------------------------
---------
|   0 | SELECT STATEMENT                  |              |       |       |     3 (100)|          |
|       |
|   1 |  PARTITION RANGE ITERATOR         |              |    58 |  3306 |     3   (0)| 00:00:01
|KEY(ZM)|KEY(ZM)|
|*  2 |   TABLE ACCESS FULL WITH ZONEMAP| SALES_RANGE |    58 |  3306 |     3   (0)| 00:00:01
|KEY(ZM)|KEY(ZM)|
-------------------------------------------------------------------------------------------------------
---------


Predicate Information (identified by operation id):
---------------------------------------------------

2 - filter((SYS_ZMAP_FILTER('/* ZM_PRUNING */ SELECT "ZONE_ID$", CASE WHEN
 BITAND(zm."ZONE_STATE$",1)=1 THEN 1 ELSE CASE WHEN (zm."MAX_1_S_SHIPDATE" < :1 OR
 zm."MIN_1_S_SHIPDATE" > :2) THEN 3 ELSE 2 END END FROM "SH"."ZMAP$_SALES_RANGE" zm WHERE
 zm."ZONE_LEVEL$"=0 ORDER BY zm."ZONE_ID$"',SYS_OP_ZONE_ID(ROWID),TO_DATE(' 1999-01-01 00:00:00',
 'syyyy-mm-dd hh24:mi:ss'),TO_DATE(' 1999-03-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))<3 AND
```

```
"S_SHIPDATE">=TO_DATE(' 1999-01-01 00:00:00','syyyy-mm-dd hh24:mi:ss') AND "S_SHIPDATE"<=TO_DATE('
 1999-03-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')))
```

3.1.7 パーティション・プルーニングのヒント

このトピックでは、パーティション・プルーニングのヒントを紹介します。

パーティション・プルーニングを使用するときは、次の点を考慮する必要があります。

- データ型の変換
- 関数のコール
- コレクション表

> ノート:
>
> CAST や TRUNC などのファンクションまたは変換を使用してパーティション列を操作する場合、パーティション・プルーニングは行われません。

3.1.7.1 データ型の変換

パーティション・プルーニングから最大限のパフォーマンスを得るには、データベースによって変換が必要となるデータ型を指定しないようにしてください。

データ型が変換されると、その他の点で静的プルーニングが可能な場合でも、通常は動的プルーニングが行われます。静的プルーニングを利用するSQL文は、動的プルーニングを利用するSQL文よりもパフォーマンスが高くなります。

OracleのDATEデータ型を使用すると、一般的なデータ型変換が行われます。OracleのDATEデータ型は文字列ではありませんが、データベースを問い合せたときに文字列のように表示されます。表示形式はインスタンスまたはセッションのNLS設定で定義されます。したがって、データをDATEフィールドに挿入するとき、またはそのようなフィールドに述語を指定するとき、同じ変換を逆向きに実行する必要があります。

変換は暗黙に実行するか、TO_DATE変換を指定して明示的に実行することができます。TO_DATE関数を適切に適用した場合のみ、データベースが一意に日付値を判別して、その値を静的プルーニングに潜在的に使用できるようになります。これは、1つのパーティションにアクセスするために特に役立ちます。

sales表に対して実行する、次の例を考察します。2000年の合計収益を問い合せる場合があります。この問合せの結果を取得する方法はたくさんありますが、すべての方法の効率が同じではありません。

```
explain plan for SELECT SUM(amount_sold) total_revenue
FROM sales,
WHERE time_id between '01-JAN-00' and '31-DEC-00';
```

計画は次のようになります。

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                 | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |       |     1 |    13 |   525   (8)| 00:00:07 |       |       |
|   1 |  SORT AGGREGATE           |       |     1 |    13 |            |          |       |       |
|*  2 |   FILTER                  |       |       |       |            |          |       |       |
|   3 |    PARTITION RANGE ITERATOR|       |   230K|  2932K|   525   (8)| 00:00:07 |   KEY |   KEY |
```

```
|*  4 |     TABLE ACCESS FULL      | SALES |   230K|  2932K|   525   (8)| 00:00:07 |   KEY |   KEY |
-----------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter(TO_DATE('01-JAN-00')<=TO_DATE('31-DEC-00'))
   4 - filter("TIME_ID">='01-JAN-00' AND "TIME_ID"<='31-DEC-00')
```

この場合、PSTARTとPSTOP両方のキーワードKEYは、動的パーティション・プルーニングが実行時に行われることを示します。次の例について考えてください。

```
explain plan for select sum(amount_sold)
from sales
where time_id between '01-JAN-2000' and '31-DEC-2000' ;
```

この実行計画は次のようになります。

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                 | Name  | Rows  | Bytes | Cost (%CPU)| Pstart| Pstop |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |       |     1 |    13 |   127   (4)|       |       |
|   1 |  SORT AGGREGATE           |       |     1 |    13 |            |       |       |
|   2 |   PARTITION RANGE ITERATOR|       |   230K|  2932K|   127   (4)|    13 |    16 |
|*  3 |    TABLE ACCESS FULL      | SALES |   230K|  2932K|   127   (4)|    13 |    16 |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("TIME_ID"<=TO_DATE(' 2000-12-31 00:00:00', "syyyy-mm-dd hh24:mi:ss"))
```

ノート:

Time 列は、この実行計画からは削除されています。

この実行計画では静的パーティション・プルーニングが示されます。問合せは、隣接するパーティション13から16にアクセスします。この場合は、日付形式の指定方法が、NLS日付形式の設定と一致しています。この例では最も効果的な実行計画が示されていますが、NLS日付形式設定を利用して特定の形式を定義することはできません。

```
alter session set nls_date_format='fmdd Month yyyy';

explain plan for select sum(amount_sold)
from sales
where time_id between '01-JAN-2000' and '31-DEC-2000' ;
```

この実行計画は次のようになります。

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                 | Name  | Rows  | Bytes | Cost (%CPU)| Pstart| Pstop |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |       |     1 |    13 |   525   (8)|       |       |
|   1 |  SORT AGGREGATE           |       |     1 |    13 |            |       |       |
|*  2 |   FILTER                  |       |       |       |            |       |       |
|   3 |    PARTITION RANGE ITERATOR|       |   230K|  2932K|   525   (8)|   KEY |   KEY |
|*  4 |     TABLE ACCESS FULL     | SALES |   230K|  2932K|   525   (8)|   KEY |   KEY |
```

```
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter(TO_DATE('01-JAN-2000')<=TO_DATE('31-DEC-2000'))
   4 - filter("TIME_ID">='01-JAN-2000' AND "TIME_ID"<='31-DEC-2000')
```

ノート:

Time 列は、この実行計画からは削除されています。

動的プルーニングを使用するこの計画は、静的プルーニングの実行計画よりも効率が悪くなります。静的パーティション・プルーニング計画を保証するには、パーティション列のデータ型と一致するようにデータ型を明示的に変換する必要があります。例:

```
explain plan for select sum(amount_sold)
from sales
where time_id between to_date('01-JAN-2000','dd-MON-yyyy')
  and to_date('31-DEC-2000','dd-MON-yyyy') ;


-------------------------------------------------------------------------------------------
| Id  | Operation                | Name  | Rows  | Bytes | Cost (%CPU)| Pstart| Pstop |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |       |     1 |    13 |   127   (4)|       |       |
|   1 |  SORT AGGREGATE          |       |     1 |    13 |            |       |       |
|   2 |   PARTITION RANGE ITERATOR|       |   230K|  2932K|   127   (4)|    13 |    16 |
|*  3 |    TABLE ACCESS FULL     | SALES |   230K|  2932K|   127   (4)|    13 |    16 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("TIME_ID"<=TO_DATE(' 2000-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

ノート:

Time 列は、この実行計画からは削除されています。

関連項目:

- DATEデータ型の詳細は、『Oracle Database SQL言語リファレンス』を参照してください
- NLSの設定およびグローバリゼーションの問題の詳細は、『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

3.1.7.2 関数のコール

関数により、プルーニングを実行するオプティマイザの機能を制限できます。

オプティマイザがプルーニングを実行できない場合がいくつかあります。一般的な理由の1つは、パーティション列に対する演算子の使用です。これには、明示的な演算子(たとえば関数)も、文を実行するために必要なデータ型変換の一環としてOracleが導入した暗黙の演算子も該当します。たとえば、次の問合せを考えてみます。

```
EXPLAIN PLAN FOR
SELECT SUM(quantity_sold)
FROM sales
WHERE time_id = TO_TIMESTAMP('1-jan-2000', 'dd-mon-yyyy');
```

time_idはDATE型ですが、Oracleは同じデータ型を取得するためにTIMESTAMP型に変換する必要があるため、この述語は内部で次のように変更されます。

```
TO_TIMESTAMP(time_id) = TO_TIMESTAMP('1-jan-2000', 'dd-mon-yyyy')
```

この文の実行計画は次のとおりです。

```
-------------------------------------------------------------------------------------------------
|Id | Operation            | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------
| 0 | SELECT STATEMENT     |       |     1 |    11 |     6  (17)| 00:00:01 |       |       |
| 1 |  SORT AGGREGATE      |       |     1 |    11 |            |          |       |       |
| 2 |   PARTITION RANGE ALL|       |    10 |   110 |     6  (17)| 00:00:01 |     1 |    16 |
|*3 |    TABLE ACCESS FULL | SALES |    10 |   110 |     6  (17)| 00:00:01 |     1 |    16 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
3 - filter(INTERNAL_FUNCTION("TIME_ID")=TO_TIMESTAMP('1-jan-2000',:B1))

15 rows selected
```

このSELECT文は、プルーニングによって1つのパーティションが限定される場合でも、すべてのパーティションにアクセスします。
2000年の合計売上高を調べる例を考えてみます。問合せを構成するもう1つの方法は、次のとおりです。

```
EXPLAIN PLAN FOR
SELECT SUM(amount_sold)
FROM sales
WHERE TO_CHAR(time_id,'yyyy') = '2000';
```

この問合せは関数コールをパーティション・キー列に適用します。通常はこのためにパーティション・プルーニングが無効になります。実行計画では全表スキャンが示され、パーティション・プルーニングはありません。

```
-------------------------------------------------------------------------------------------------
| Id  | Operation            | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |       |     1 |    13 |   527   (9)| 00:00:07 |       |       |
|   1 |  SORT AGGREGATE      |       |     1 |    13 |            |          |       |       |
|   2 |   PARTITION RANGE ALL|       |  9188 |   116K|   527   (9)| 00:00:07 |     1 |    28 |
|*  3 |    TABLE ACCESS FULL | SALES |  9188 |   116K|   527   (9)| 00:00:07 |     1 |    28 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter(TO_CHAR(INTERNAL_FUNCTION("TIME_ID"),'yyyy')='2000')
```

パーティション列に対しては暗黙的にも明示的にも関数を使用しないことをお薦めします。問合せがよく関数コールを使用する場合は、このようなケースでパーティション・プルーニングを利用できるように仮想列と仮想列パーティション化の使用を検討してく

ださい。

3.1.7.3 コレクション表

コレクション表により、プルーニングを実行するオプティマイザの機能を制限できます。

次の例は、EXPLAIN PLAN文がコレクション表を含むときに、どのように表示されるかを示します(コレクション表は、ここでは便宜上、ソートされたコレクション表またはネスト表とします)。該当するパーティションのみに制約されるため、全表アクセスは実行されません。

```
EXPLAIN PLAN FOR
SELECT p.ad_textdocs_ntab
FROM print_media_part p;

Explained.

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 2207588228

------------------------------------------------------------------------------
| Id  | Operation                  | Name             | Pstart| Pstop |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                  |       |       |
|   1 |  PARTITION REFERENCE SINGLE|                  |   KEY |   KEY |
|   2 |   TABLE ACCESS FULL        | TEXTDOC_NT       |   KEY |   KEY |
|   3 |  PARTITION RANGE ALL       |                  |     1 |     2 |
|   4 |   TABLE ACCESS FULL        | PRINT_MEDIA_PART |     1 |     2 |
------------------------------------------------------------------------------

Note
-----
  - dynamic sampling used for this statement
```

関連項目:

EXPLAIN PLANのベースとなるCREATE TABLE文の例は、「XMLTypeおよびオブジェクトのコレクションのパーティション化」を参照してください

3.2 パーティション・ワイズ操作

パーティション・ワイズ操作により、レスポンス時間は大幅に短縮され、CPUとメモリー・リソースの使用率が改善されます。

パーティション・ワイズ結合では、結合がパラレルで実行されるときにパラレル実行サーバー間で交換されるデータ量が最小限に抑えられ、問合せのレスポンス時間が短縮されます。Oracle Real Application Clusters (Oracle RAC)環境では、パーティション・ワイズ結合でも、インターコネクトでのデータ・トラフィックが回避されるか、少なくとも制限されます。これは、大規模な結合操作で優れたスケーラビリティを実現するために重要です。パラレル・パーティション・ワイズ結合は、大きな結合を効率的かつ高速に処理するために一般的に使用されます。パーティション・ワイズ結合では、フル(完全)またはパーシャル(部分)を選択できます。どちらの種類の結合を使用するかはOracle Databaseによって判別されます。

パラレル・パーティション・ワイズ結合に加えて、SELECT DISTINCT句およびSQLウィンドウ関数を使用した問合せでは、パラレル・パーティション・ワイズ操作を実行できます。

次の内容について説明します。

- フル・パーティション・ワイズ結合
- パーシャル・パーティション・ワイズ結合

関連項目:

- データ・ウェアハウス環境でのパラレル・パーティション・ワイズ操作の詳細は、「データ・ウェアハウスでのパーティション・ワイズ結合」を参照してください
- データ・ウェアハウスおよび最適化方法の詳細は、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください

3.2.1 フル・パーティション・ワイズ結合

フル・パーティション・ワイズ結合では、結合される2つの表の対のパーティション間で、大きな結合が小さな結合に分割されます。

フル・パーティション・ワイズ結合を使用するには、両方の表を結合キーで同一レベルでパーティション化するか、参照パーティション化を使用する必要があります。

2つの表を同一レベル・パーティション化するには様々なパーティション化方法を使用できます。これらの方法の概要を次のトピックで説明します。

- フル・パーティション・ワイズ結合による問合せ
- フル・パーティション・ワイズ結合: 単一レベル-単一レベル
- フル・パーティション・ワイズ結合: コンポジット-単一レベル
- フル・パーティション・ワイズ結合: コンポジット-コンポジット

3.2.1.1 フル・パーティション・ワイズ結合による問合せ

フル・パーティション・ワイズ結合を使用して問合せできます。

たとえば、例3-4に示すように、sales表とcustomer表の列cust_idによる大規模な結合について考えてみます。「1999年の第3四半期に品物の購入数が100を超えたすべての顧客のレコードを検索する」という問合せは、このような結合を実行するSQL文の典型的な例です。

このような大規模な結合は、データ・ウェアハウス環境ではよく行われます。この場合、customer表全体が1四半期分のsalesデータに結合されます。大規模なデータ・ウェアハウス・アプリケーションの場合には、数百万行の結合を意味することもあります。ここは明らかにハッシュ結合が使用されます。両方の表がcust_id列で同一レベル・パーティション化されている場合は、このハッシュ結合の処理時間をさらに短縮できます。この機能によりフル・パーティション・ワイズ結合が使用可能になります。

フル・パーティション・ワイズ結合をパラレルで実行するとき、パラレル化のグラニュルはパーティションです。このため、並列度はパーティション数に制限されます。たとえば、問合せの並列度を16に設定するには少なくとも16のパーティションが必要です。

例3-4 フル・パーティション・ワイズ結合による問合せ

```
SELECT c.cust_last_name, COUNT(*)
  FROM sales s, customers c
  WHERE s.cust_id = c.cust_id AND
        s.time_id BETWEEN TO_DATE('01-JUL-1999', 'DD-MON-YYYY') AND
        (TO_DATE('01-OCT-1999', 'DD-MON-YYYY'))
  GROUP BY c.cust_last_name HAVING COUNT(*) > 100;
```

3.2.1.2 フル・パーティション・ワイズ結合: 単一レベル-単一レベル

単一レベルから単一レベルへのフル・パーティション・ワイズ結合は、最も簡単な方法であり、2つの表は両方とも結合列によってパーティション化されます。

例では、customers表とsales表の両方がcust_id列でパーティション化されます。このパーティション化方法では、表をcust_id (両方とも同じ顧客ID番号を表す)で結合するフル・パーティション・ワイズ結合が有効になります。このシナリオは、レンジ - レンジ、リスト - リストおよびハッシュ - ハッシュ・パーティション化に対応します。時間隔 - レンジおよび時間隔 - 時間隔のフル・パーティション・ワイズ結合もサポートされ、レンジ - レンジと比較できます。

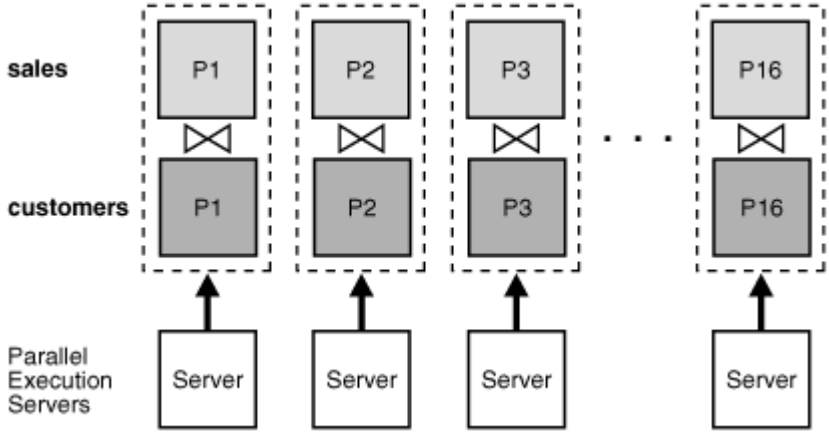
シリアルでは、この結合は、ハッシュ・パーティションの一致する組の間で、1回ずつ順番に実行されます。1組のパーティションが結合されると、別の組の結合が開始されます。すべてのパーティションの組が処理されると、結合が完了します。作業負荷を均等に分散するためには、要求した並列度よりも多数のパーティションを使用するか、要求した並列度と同数の同一サイズ・パーティションを使用する必要があります。同一サイズ・パーティションを作成するには、パーティション数を2の累乗として、一意の列(またはほぼ一意の列)に対してハッシュ・パーティション化を使用することをお薦めします。

ノート:

- 一致する 1 組のハッシュ・パーティションは、同じパーティション番号を持つ各表のパーティションとして定義されています。たとえば、ハッシュ・パーティション化に基づくフル・パーティション・ワイズ結合では、sales のパーティション 0 と customers のパーティション 0、sales のパーティション 1 と customers のパーティション 1 のように結合されます。
- 参照パーティション化は、2 つの表をともにパーティション化する簡単な方法です。これにより、文で表が結合される際に、オプティマイザがフル・パーティション・ワイズ結合を常に検討できるようになります。

フル・パーティション・ワイズ結合のパラレル実行は、シリアル実行を単にパラレル化したものです。パーティションが1組ずつ結合されるかわりに、問合せサーバーによってパーティションの複数の組がパラレルに結合されます。図3-1に、フル・パーティション・ワイズ結合のパラレル実行を示します。

図3-1 フル・パーティション・ワイズ結合のパラレル実行



次の例は、同数のパーティションを含むようにハッシュで同等にパーティション化されたsalesとcustomersの実行計画を示します。この計画はフル・パーティション・ワイズ結合を示しています。

```
explain plan for SELECT c.cust_last_name, COUNT(*)
FROM sales s, customers c
WHERE s.cust_id = c.cust_id AND
s.time_id BETWEEN TO_DATE('01-JUL-1999', 'DD-MON-YYYY') AND
     (TO_DATE('01-OCT-1999', 'DD-MON-YYYY'))
GROUP BY c.cust_last_name HAVING COUNT(*) > 100;

-------------------------------------------------------------------------------------------------------
-------
| Id  | Operation                    | Name      | Rows  | Bytes | Pstart| Pstop|    TQ  |IN-OUT| PQ
Distrib|
-------------------------------------------------------------------------------------------------------
-------
|   0 | SELECT STATEMENT             |           |    46 |  1196 |       |      |        |      |
|
|   1 |  PX COORDINATOR              |           |       |       |       |      |        |      |
|
|   2 |   PX SEND QC (RANDOM)        | :TQ10001  |    46 |  1196 |       |      |  Q1,01 | P->S | QC
(RAND) |
|*  3 |    FILTER                    |           |       |       |       |      |  Q1,01 | PCWC |
|
|   4 |     HASH GROUP BY            |           |    46 |  1196 |       |      |  Q1,01 | PCWP |
|
|   5 |      PX RECEIVE              |           |    46 |  1196 |       |      |  Q1,01 | PCWP |
|
|   6 |       PX SEND HASH           | :TQ10000  |    46 |  1196 |       |      |  Q1,00 | P->P | HASH
|
|   7 |        HASH GROUP BY         |           |    46 |  1196 |       |      |  Q1,00 | PCWP |
|
|   8 |         PX PARTITION HASH ALL|           | 59158 |  1502K|     1 |   16 |  Q1,00 | PCWC |
|
|*  9 |          HASH JOIN           |           | 59158 |  1502K|       |      |  Q1,00 | PCWP |
|
|  10 |           TABLE ACCESS FULL  | CUSTOMERS | 55500 |   704K|     1 |   16 |  Q1,00 | PCWP |
|
|* 11 |           TABLE ACCESS FULL  | SALES     | 59158 |   751K|     1 |   16 |  Q1,00 | PCWP |
|
-------------------------------------------------------------------------------------------------------
-------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter(COUNT(SYS_OP_CSR(SYS_OP_MSR(COUNT(*)),0))>100)
```

```
   9 - access("S"."CUST_ID"="C"."CUST_ID")
  11 - filter("S"."TIME_ID"<=TO_DATE(' 1999-10-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
"S"."TIME_ID">=TO_DATE(' 1999-07-01
              00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

ノート:

Cost (%CPU)列とTime列は、この例の計画表の出力では削除されています。

超並列処理(MPP)プラットフォームで実行されているOracle RAC環境で適切なスケーラビリティを得るには、パーティションをノード上に配置することが重要です。リモートI/Oを回避するには、一致する2つのパーティションが、同じノードに対してアフィニティを持っている必要があります。ボトルネックを回避し、システムで使用可能なすべてのCPUリソースを使用するために、パーティションの組がすべてのノードに分散される必要があります。

ノードの数よりパーティションの組の数が多い場合は、ノードで複数の組を管理できます。たとえば、ノード数が8のシステムでパーティションが16組ある場合は、各ノードが2組のパーティションに対応します。

関連項目:

データ・アフィニティの詳細は、『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

3.2.1.3 フル・パーティション・ワイズ結合: コンポジット-単一レベル

コンポジットから単一レベルへのフル・パーティション・ワイズ結合は、単一レベル-単一レベル方法の一種です。

このシナリオでは、1つの表(通常は大きい方の表)が2つのディメンションでコンポジット・パーティション化され、サブパーティション・キーとして結合列が使用されます。この例ではsales表が、履歴データを格納する表の一般的な例です。レンジ・パーティション化は、履歴情報を格納する表に対して、通常最初に行うパーティション化方法です。

たとえば、sales表を列time_idの範囲によって8個のパーティションにパーティション化するとします。また、2年分のデータがあり、各パーティションは四半期を表します。レンジ・パーティション化のかわりに、コンポジット・パーティション化を使用して、time_idでのパーティション化を維持しながらフル・パーティション・ワイズ結合を有効にすることができます。たとえば、sales表をtime_idの範囲でパーティション化し、各パーティションを16個のサブパーティションを使用してcust_idのハッシュでサブパーティション化します(サブパーティションの合計は128個)。customers表は、16パーティションでのハッシュ・パーティション化を使用できます。

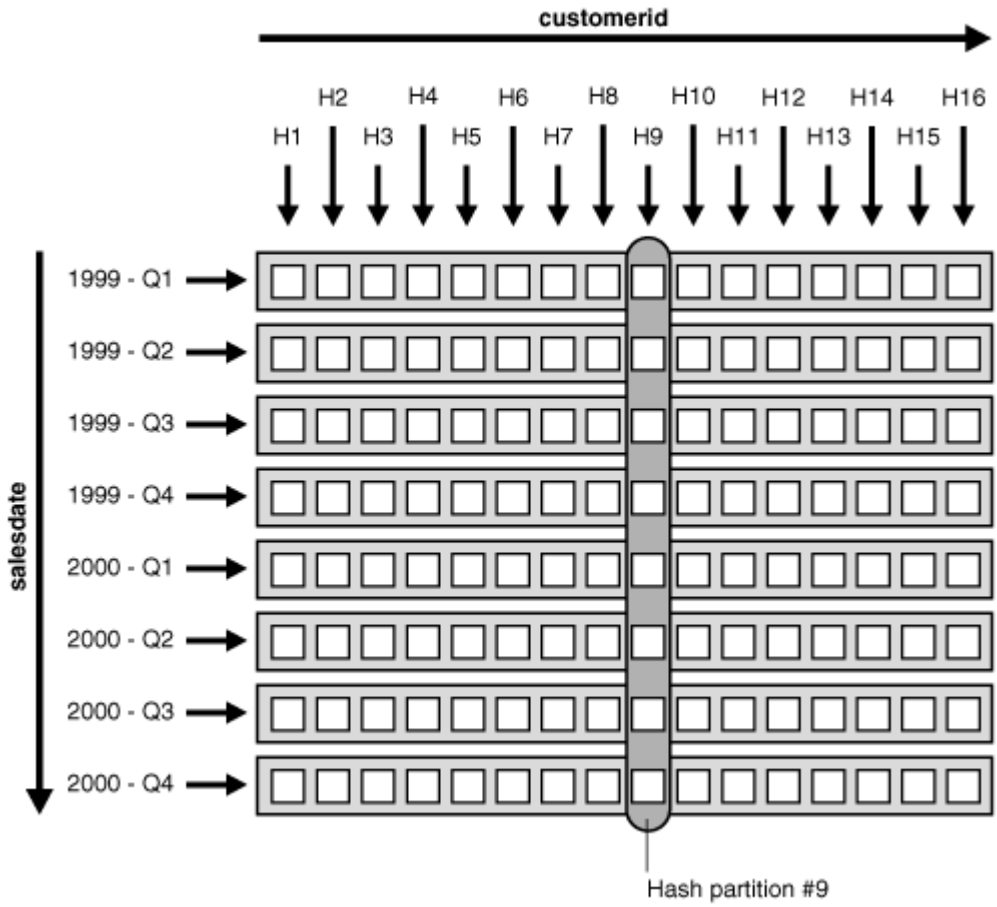
ここで説明した方法を使用すると、フル・パーティション・ワイズ結合は、単一レベル-単一レベルのハッシュ-ハッシュ方法と同様に動作します。また、結合も、両方の表のハッシュ・パーティションの組の間で、16の小規模な結合に分割されます。違いは、sales表の各ハッシュ・パーティションが、各レンジ・パーティションから1つずつ、8つのサブパーティションの集合で構成されているということです。

図3-2は、sales表でハッシュ・パーティション化がどのように実現されるかを示します。各セルがサブパーティションを表します。全体で8個のレンジ・パーティションがあり、各行は1つのレンジ・パーティションに対応します。各レンジ・パーティションには16個のサブパーティションがあります。全体で16個のハッシュ・パーティションがあり、各列は1つのハッシュ・パーティションに対応します。各ハッシュ・パーティションには8個のサブパーティションがあります。ハッシュ・パーティションを定義できるのは、すべてのパーティションに同数のサブパーティション(ここでは16個)がある場合のみです。

ハッシュ・パーティションはコンポジット表では暗黙的に扱われます。ただし、データ・ディクショナリには記録されず、レンジ・パーティ

ションやリスト・パーティションのようにDDLコマンドを使用して処理することはできません。

図3-2 コンポジット表のレンジ・パーティションおよびハッシュ・パーティション



次の例では、sales表のフル・パーティション・ワイズ結合の実行計画が示されます。この表は、time_idについてレンジ・パーティション化され、cust_idについてハッシュでサブパーティション化されています。

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name      | Pstart| Pstop |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |           |       |       |      |            |
|   1 |  PX COORDINATOR                    |           |       |       |      |            |
|   2 |   PX SEND QC (RANDOM)              | :TQ10001  |       |       | P->S | QC (RAND)  |
|*  3 |    FILTER                          |           |       |       | PCWC |            |
|   4 |     HASH GROUP BY                  |           |       |       | PCWP |            |
|   5 |      PX RECEIVE                    |           |       |       | PCWP |            |
|   6 |       PX SEND HASH                 | :TQ10000  |       |       | P->P | HASH       |
|   7 |        HASH GROUP BY               |           |       |       | PCWP |            |
|   8 |         PX PARTITION HASH ALL      |           |     1 |    16 | PCWC |            |
|*  9 |          HASH JOIN                 |           |       |       | PCWP |            |
|  10 |           TABLE ACCESS FULL        | CUSTOMERS |     1 |    16 | PCWP |            |
|  11 |           PX PARTITION RANGE ITERATOR|         |     8 |     9 | PCWC |            |
|* 12 |            TABLE ACCESS FULL       | SALES     |   113 |   144 | PCWP |            |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter(COUNT(SYS_OP_CSR(SYS_OP_MSR(COUNT(*)),0))>100)
   9 - access("S"."CUST_ID"="C"."CUST_ID")
  12 - filter("S"."TIME_ID"<=TO_DATE(' 1999-10-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
"S"."TIME_ID">=TO_DATE(' 1999-07-01
              00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

ノート:

Rows 列、Cost （%CPU）列、Time 列および TQ 列は、この例の計画表の出力では削除されています。

コンポジット-単一レベルのパーティション化が効率的なのは、あるディメンションではプルーニングを行い、別のディメンションではフル・パーティション・ワイズ結合を行えるためです。前述の問合せの例では、1999年の第3四半期に相当するサブパーティション(図3-2の3番目の行)のみをスキャンして、プルーニングが実現されます。Oracleによって、フル・パーティション・ワイズ結合が使用され、これらのサブパーティションがcustomer表と結合されます。

単一レベル-単一レベルのパーティション・ワイズ結合のすべての特性は、コンポジット-単一レベルのパーティション・ワイズ結合にも該当します。特に、この例では、2つの方法は次の2つの点で共通しています。

- この場合のフル・パーティション・ワイズ結合の並列度は、16を超えることはありません。sales表に128のサブパーティションがあっても、ハッシュ・パーティションは16のみであるためです。
- パーティションはサブパーティションの集合です。たとえば、図3-2では、sales表のハッシュ・パーティション9(楕円で囲まれた8つのサブパーティション)をcustomers表のハッシュ・パーティション9と同じノードに格納します。

3.2.1.4 フル・パーティション・ワイズ結合: コンポジット-コンポジット

より柔軟性を求める場合、コンポジットからコンポジットへのフル・パーティション・ワイズ結合を使用できます。

必要に応じて、customers表をコンポジット方法でパーティション化することもできます。たとえば、郵便番号の列についてこの表をレンジ・パーティション化して、郵便番号に基づくプルーニングを使用可能にできます。その後、同数(16)のパーティションを使用し、cust_idについてハッシュでサブパーティション化し、ハッシュ・ディメンションでパーティション・ワイズ結合を使用可能にします。

フル・パーティション・ワイズ結合は、パーティションとサブパーティションのすべての組合せ(パーティションとパーティション、パーティションとサブパーティション、サブパーティションとパーティション、サブパーティションとサブパーティション)で行うことができます。

3.2.2 パーシャル・パーティション・ワイズ結合

パーシャル・パーティション・ワイズ結合では、1つの表のみを結合キーでパーティション化する必要があります。

Oracle Databaseでは、パラレル時のみパーシャル・パーティション・ワイズ結合が実行できます。フル・パーティション・ワイズ結合と異なり、パーシャル・パーティション・ワイズ結合では、両方の表ではなく、一方の表のみを結合キーでパーティション化する必要があります。パーティション化された表は、参照表と呼ばれます。もう一方の表は、パーティション化してもしなくてもかまいません。パーシャル・パーティション・ワイズ結合は、フル・パーティション・ワイズ結合よりも一般的です。

パーシャル・パーティション・ワイズ結合を実行するために、データベースによって、参照表のパーティション化に基づいて、もう一方の表が動的に再パーティション化されます。もう一方の表が再パーティション化された後は、フル・パーティション・ワイズ結合と同様に実行されます。

パーシャル・パーティション・ワイズ結合が、非パーティション表の結合よりパフォーマンス上でメリットがあるのは、結合操作の間に参照表が移動しないことです。非パーティション表間のパラレル結合では、両方の入力表を結合キーについて再分散する必要があります。この再分散操作には、パラレル実行サーバー間での行の交換が伴います。これはCPU集中型の操作であり、Oracle RAC環境ではインターコネクト・トラフィックが増大する原因になります。結合キー(外部キーまたは主キー)で大規模な表をパーティション化すると、そのキーで表を結合するたびにこの再分散が発生することを防止できます。ただし、外部キーで表をパーティション化する場合(最も一般的な場合)は、多くの問合せに含まれる外部キーを選択する必要があります。

パーシャル・パーティション・ワイズ結合を説明するために、前のsales/customersの例を使用します。customersがパーティショ

ン化されていないか、cust_id以外の列でパーティション化されているとします。salesは多くの場合cust_idでcustomersと結合されることが多く、この結合がアプリケーションのワークロードを占有します。このため、salesをcust_idでパーティション化して、customersとsalesが結合されるたびに、パーシャル・パーティション・ワイズ結合が使用可能になるようにします。フル・パーティション・ワイズ結合と同じく、次に示す他の方法もあります。

- パーシャル・パーティション・ワイズ結合: 単一レベル・パーティション化
- パーシャル・パーティション・ワイズ結合: コンポジット

3.2.2.1 パーシャル・パーティション・ワイズ結合: 単一レベル・パーティション化

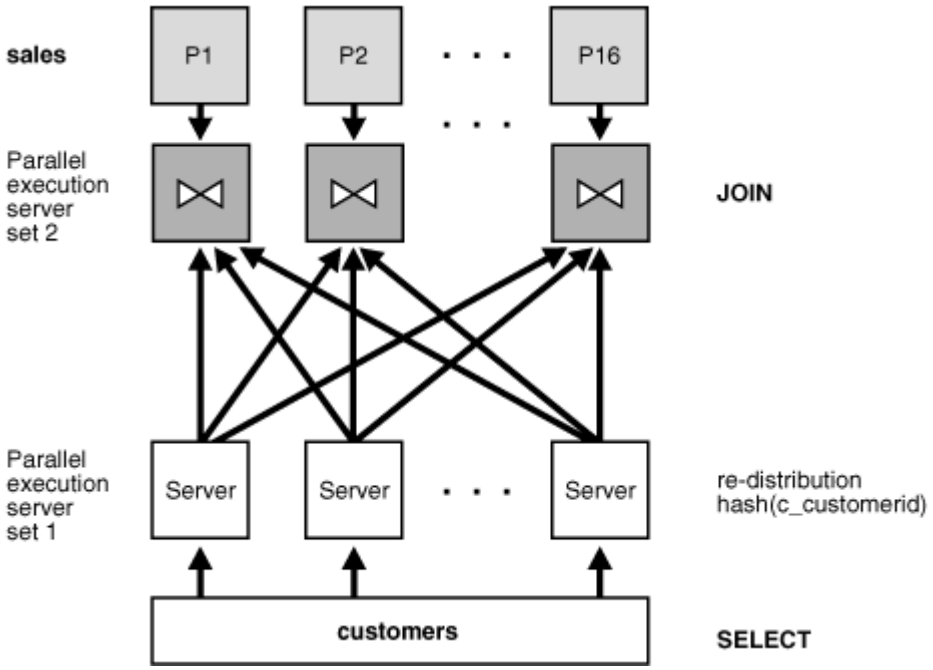
単一レベル・パーシャル・パーティション・ワイズ結合は、パーシャル・パーティション・ワイズ結合を使用可能にする最も簡単な方法です。

たとえば、単一レベル・パーシャル・パーティション・ワイズ結合を使用可能にして、salesをcust_idでハッシュ・パーティション化できます。パーティションは、パーシャル・パーティション・ワイズ結合操作におけるパラレル化の最小グラニュルであるため、並列度の最大値はパーティションの数によって決定されます。

パーシャル・パーティション・ワイズ結合のパラレル実行を図3-3に示します。ここでは、並列度とsalesのパーティション数はどちらも16です。この実行では、2つのセットの問合せサーバーが使用されます。図3-3のセット*1*はcustomers表をパラレルでスキャンします。スキャン操作のパラレル化のグラニュルはブロックの範囲です。

セット1によって選択されたcustomers表の行(この場合はすべての行)は、cust_idをハッシュすることで、セット2の問合せサーバーに再分散されます。たとえば、sales表のパーティションP1の行と一致する可能性があるcustomers表の行はすべて、セット2の問合せサーバー1に送られます。セット2の問合せサーバーが受け取った行は、sales表にある対応するパーティションの行と結合されます。セット2の問合せサーバー1は、受け取ったcustomers表のすべての行とsales表のパーティションP1を結合します。

図3-3 パーシャル・パーティション・ワイズ結合

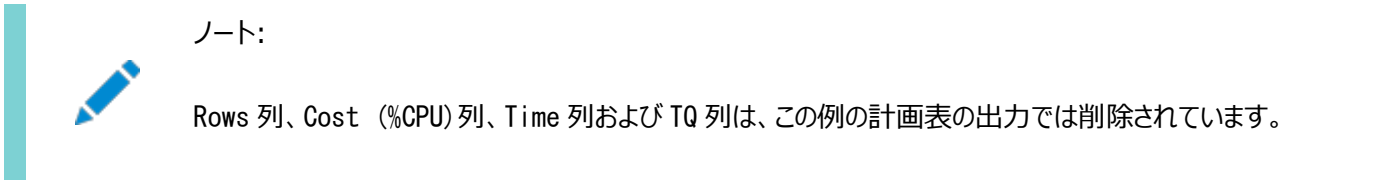


次の例に、sales表とcustomers表のパーシャル・パーティション・ワイズ結合の実行計画を示します。

```
-------------------------------------------------------------------------------------------
| Id  | Operation                          | Name      | Pstart| Pstop |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |           |       |       |      |            |
|   1 |  PX COORDINATOR                    |           |       |       |      |            |
|   2 |   PX SEND QC (RANDOM)              | :TQ10002  |       |       | P->S | QC (RAND)  |
|*  3 |    FILTER                          |           |       |       | PCWC |            |
```

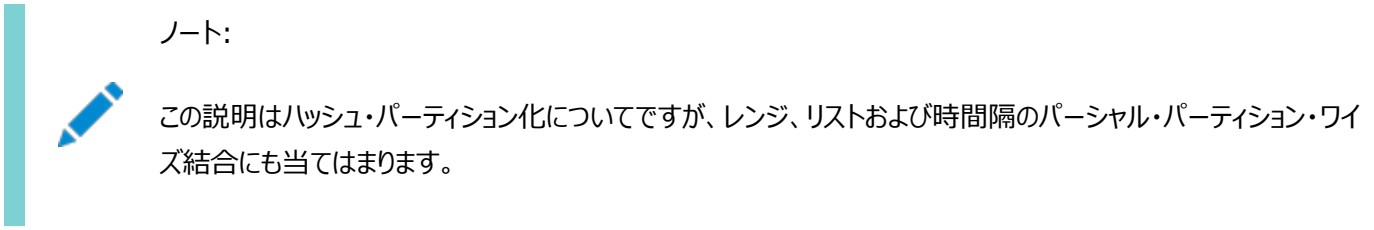
```
|   4 |     HASH GROUP BY                    |           |       |       | PCWP |            |
|   5 |      PX RECEIVE                      |           |       |       | PCWP |            |
|   6 |       PX SEND HASH                   | :TQ10001  |       |       | P->P | HASH       |
|   7 |        HASH GROUP BY                 |           |       |       | PCWP |            |
|*  8 |         HASH JOIN                    |           |       |       | PCWP |            |
|   9 |          PART JOIN FILTER CREATE     | :BF0000   |       |       | PCWP |            |
|  10 |           PX RECEIVE                 |           |       |       | PCWP |            |
|  11 |            PX SEND PARTITION (KEY)   | :TQ10000  |       |       | P->P | PART (KEY) |
|  12 |             PX BLOCK ITERATOR        |           |       |       | PCWC |            |
|  13 |              TABLE ACCESS FULL       | CUSTOMERS |       |       | PCWP |            |
|  14 |          PX PARTITION HASH JOIN-FILTER|          |:BF0000|:BF0000| PCWC |            |
|* 15 |           TABLE ACCESS FULL          | SALES     |:BF0000|:BF0000| PCWP |            |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter(COUNT(SYS_OP_CSR(SYS_OP_MSR(COUNT(*)),0))>100)
   8 - access("S"."CUST_ID"="C"."CUST_ID")
  15 - filter("S"."TIME_ID"<=TO_DATE(' 1999-10-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
"S"."TIME_ID">=TO_DATE(' 1999-07-01
              00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

PX行ソースが存在するため、計画に表示されているように、この問合せはパラレルで実行されます。1つの表、SALES表がパーティション化されています。PX PARTITION HASH行ソースに、PX SEND PARTITIONによって結合を実行する別のワーカー・セットに分散されたパーティション化されていない表CUSTOMERSが含まれているため、このことを判別できます。



ノート:

Rows 列、Cost (%CPU)列、Time 列および TQ 列は、この例の計画表の出力では削除されています。



ノート:

この説明はハッシュ・パーティション化についてですが、レンジ、リストおよび時間隔のパーシャル・パーティション・ワイズ結合にも当てはまります。

フル・パーティション・ワイズ結合に関する考慮点は、次のようにパーシャル・パーティション・ワイズ結合にも適用されます。

- 並列度は、パーティションの数と同じでなくてもかまいません。図3-3では、16の問合せサーバー2組で問合せが実行されています。ここでは、セット2の各問合せサーバーに1つのパーティションが割り当てられます。パーティションの数は常に並列度の倍数である必要があります。
- MPPにおけるOracle RAC環境では、リモートI/Oを回避するために、sales表の各ハッシュ・パーティションが1つのノードのみに対してアフィニティを持つことが望ましいとされます。また、ボトルネックを回避し、システムで使用可能なすべてのCPUリソースを使用するために、パーティションをすべてのノードに分散させます。ノードの数よりパーティションの数が多い場合は、1つのノードで複数のパーティションを管理できます。

関連項目:

データ・アフィニティの詳細は、『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

3.2.2.2 パーシャル・パーティション・ワイズ結合: コンポジット

コンポジット・パーシャル・パーティション・ワイズ結合を使用できます。

フル・パーティション・ワイズ結合と同様に、sales表の最適なパーティション化方法は、time_id列に対してレンジ方法を使用することです。これは、sales表が、履歴データを格納する表の典型であるためです。このレンジ・パーティション化を維持して、パーシャル・パーティション・ワイズ結合を使用可能にするには、sales表をcust_id列でハッシュによってサブパーティション化し、各パーティションが16のサブパーティションに分かれるようにします。問合せによってcustomers表とsales表が結合され、time_idに関する選択述語がその問合せにある場合、プルーニングとパーシャル・パーティション・ワイズ結合の両方を使用できます。

sales表がコンポジット・パーティション化されているとき、パーシャル・パーティション・ワイズ結合のパラレル化のグラニュルは、サブパーティションではなくハッシュ・パーティションです。コンポジット表のハッシュ・パーティションの図は、図3-2を参照してください。ハッシュ・パーティションの数は並列度の倍数である必要があります。また、MPPシステムでは、各ハッシュ・パーティションが1つのノードに対してアフィニティを持つようにしてください。前の例では、1つのハッシュ・パーティションを構成する8個のサブパーティションが同じノードに対するアフィニティを持っています。

ノート:

この説明はレンジ-ハッシュに関してですが、他のすべての組合せのコンポジット・パーシャル・パーティション・ワイズ結合にも当てはまります。

次の例では、salesとcustomersの間の問合せの実行計画が示されます。sales表は、time_idについてレンジ・パーティション化され、cust_idについてハッシュでサブパーティション化されています。

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name      | Pstart| Pstop |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |           |       |       |      |            |
|   1 |  PX COORDINATOR                    |           |       |       |      |            |
|   2 |   PX SEND QC (RANDOM)              | :TQ10002  |       |       | P->S | QC (RAND)  |
|*  3 |    FILTER                          |           |       |       | PCWC |            |
|   4 |     HASH GROUP BY                  |           |       |       | PCWP |            |
|   5 |      PX RECEIVE                    |           |       |       | PCWP |            |
|   6 |       PX SEND HASH                 | :TQ10001  |       |       | P->P | HASH       |
|   7 |        HASH GROUP BY               |           |       |       | PCWP |            |
|*  8 |         HASH JOIN                  |           |       |       | PCWP |            |
|   9 |          PART JOIN FILTER CREATE   | :BF0000   |       |       | PCWP |            |
|  10 |           PX RECEIVE               |           |       |       | PCWP |            |
|  11 |            PX SEND PARTITION (KEY) | :TQ10000  |       |       | P->P | PART (KEY) |
|  12 |             PX BLOCK ITERATOR      |           |       |       | PCWC |            |
|  13 |              TABLE ACCESS FULL     | CUSTOMERS |       |       | PCWP |            |
|  14 |          PX PARTITION RANGE ITERATOR|          |     8 |     9 | PCWC |            |
|  15 |           PX PARTITION HASH ALL    |           |     1 |    16 | PCWC |            |
|* 16 |            TABLE ACCESS FULL       | SALES     |   113 |   144 | PCWP |            |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
```

```
   3 - filter(COUNT(SYS_OP_CSR(SYS_OP_MSR(COUNT(*)),0))>100)
   8 - access("S"."CUST_ID"="C"."CUST_ID")
  16 - filter("S"."TIME_ID"<=TO_DATE(' 1999-10-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
"S"."TIME_ID">=TO_DATE(' 1999-07-01
              00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

ノート:

Rows 列、Cost (%CPU)列、Time 列および TQ 列は、この例の計画表の出力では削除されています。

3.3 索引のパーティション化

索引のパーティション化には、パーティション表共通の推奨事項および考慮事項があります。

索引のパーティション化に関するルールは、表についてのルールと似ています。

- 次の項目に該当しないかぎり、索引のパーティション化が可能です。
    - 索引がクラスタ索引である。
    - 索引がクラスタ化表に定義されている。
- 次のように、パーティション索引および非パーティション索引は、パーティション表および非パーティション表に混在させることができます。
    - パーティション表にパーティション索引または非パーティション索引を定義できる。
    - 非パーティション表にパーティション索引または非パーティション索引を定義できる。
- 非パーティション表のビットマップ索引は、パーティション化できません。
- パーティション表のビットマップ索引は、ローカル索引にする必要があります。

ただし、パーティション索引はパーティション表よりも複雑です。パーティション索引には次の3つのタイプがあるためです。

- ローカル同一キー索引
- ローカル非同一キー索引
- グローバル同一キー索引

Oracle Databaseでは、これら3つのタイプがすべてサポートされています。ただし、いくつかの制限があります。たとえば、パーティション表でローカル一意索引を作成する場合、キーを式にすることはできません。

次の内容について説明します。

- ローカル・パーティション索引
- グローバル・パーティション索引
- パーティション索引のまとめ
- 非同一キー索引の重要性
- 同一キー索引および非同一キー索引がパフォーマンスに与える影響
- パーティション索引での拡張索引圧縮
- 索引のパーティション化のガイドライン
- 索引パーティションの物理属性

関連項目:

DBA_INDEXES、DBA_IND_PARTITIONS、DBA_IND_SUBPARTITIONSおよびDBA_PART_INDEXES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。

3.3.1 ローカル・パーティション索引

ローカル索引の場合、特定の索引パーティションのキーはすべて、基礎となる表の1つのパーティションに格納されている行のみを参照します。

ローカル索引は、LOCAL属性を指定することで作成されます。Oracleは、基礎となる表と同一レベルでパーティション化されるようにローカル索引を作成します。基礎となる表と同じ列で索引をパーティション化し、同じ数のパーティションまたはサブパーティションを作成して、基礎となる表の対応するパーティションと同じパーティション・バウンドを設定します。

また、Oracleは、基礎となる表のパーティションが追加、削除、マージまたは分割されたり、ハッシュ・パーティションやサブパーティションが追加または結合されたりした場合は、索引のパーティション化を自動的にメンテナンスします。これにより、索引のパーティションが表と同一レベルに保たれます。

パーティション列が索引列のサブセットを形成している場合は、ローカル索引をUNIQUEとして作成できます。この制限により、同一の索引キーを持つ行は同じパーティションにマップされることが保証され、一意性の違反を検出できるようになります。

ローカル索引には次のメリットがあります。

- 基礎となる表パーティションに対してSPLIT PARTITIONまたはADD PARTITION以外のメンテナンス操作を実行する際、再作成する必要のある索引パーティションが1つで済みます。
- パーティション表にローカル索引しかない場合、パーティションのメンテナンス操作にかかる時間はパーティションのサイズに比例します。
- ローカル索引によって、パーティションの独立性がサポートされます。
- ローカル索引では、履歴表の古いデータのロールアウト、新しいデータのロールインをスムーズに実行できます。
- Oracleは、ローカル索引は基礎となる表と同一レベルでパーティション化されるという特性を利用して、より適切な問合せのアクセス計画を生成できます。
- ローカル索引を使用すると、表領域の不完全リカバリ作業が簡素化されます。表のパーティションまたはサブパーティションをある時点までリカバリするには、対応する索引エントリも同じ時点までリカバリする必要があります。この処理は、ローカル索引を使用している場合にのみ行うことができます。ローカル索引を使用している場合は、対応する表と索引のパーティションまたはサブパーティションをリカバリできます。

次の内容について説明します。

- ローカル同一キー索引
- ローカル非同一キー索引

関連項目:

DBMS_PCLXUTIL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

3.3.1.1 ローカル同一キー索引

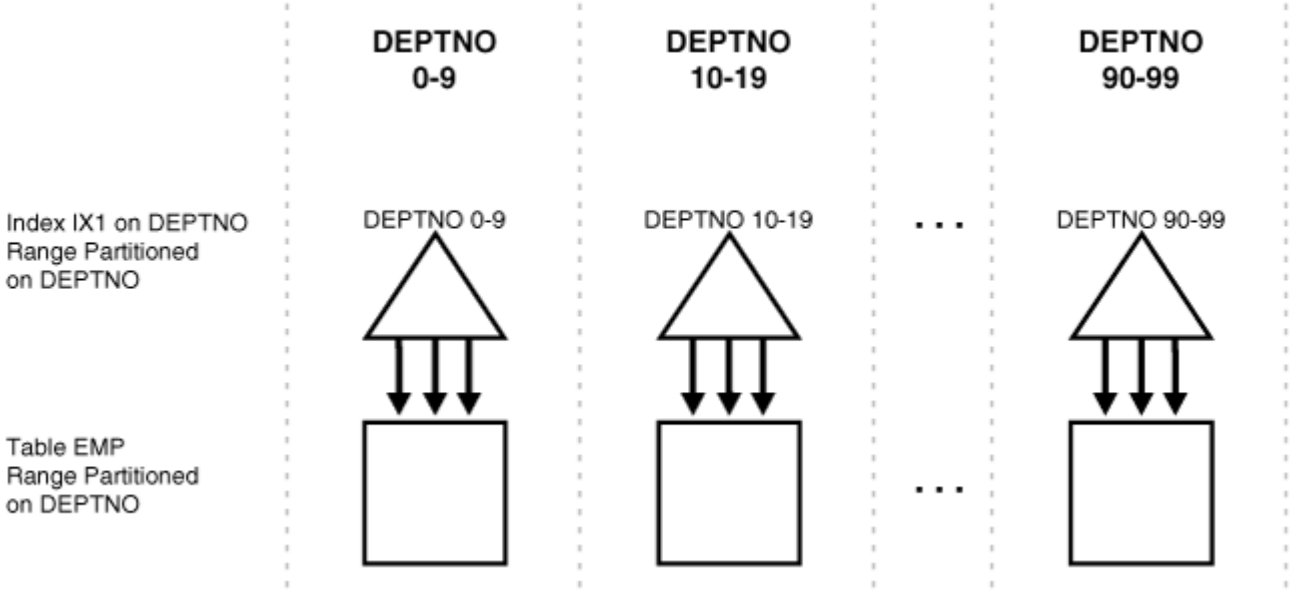
ローカル索引が同一キーになるのは、索引列の左プリフィックスでパーティション化され、サブパーティション化キーが索引キーに含まれる場合です。ローカル同一キー索引は、一意にも非一意にもできます。

たとえば、sales表とそのローカル索引sales_ixがweek_num列でパーティション化されているとき、索引sales_ixがローカル同

一キーになるのは列week_numおよびxaction_numに定義されている場合です。これに対して、索引sales_ixが列product_numに定義されている場合は、同一キーにはなりません。

図3-4に、ローカル同一キー索引の別の例を示します。

図3-4 ローカル同一キー索引



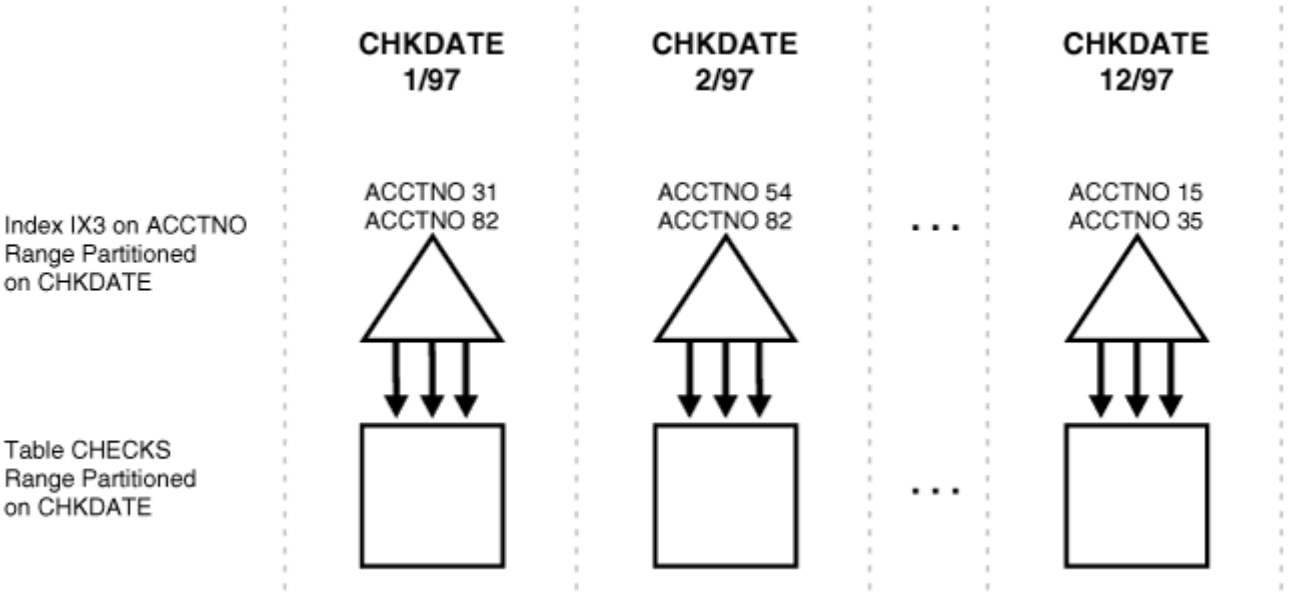
3.3.1.2 ローカル非同一キー索引

索引列の左プリフィックスでパーティション化されていないローカル索引は、非同一キー索引です。あるいは、索引キーにサブパーティション化キーが含まれない場合、ローカル索引は非同一キー索引です。

パーティション化キーが索引キーのサブセットでない場合、一意のローカル非同一キー索引は定義できません。

図3-5に、ローカル非同一キー索引の例を示します。

図3-5 ローカル非同一キー索引



3.3.2 グローバル・パーティション索引

グローバル・パーティション索引では、特定の索引パーティションのキーが、複数の基礎となる表のパーティションまたはサブパーティションに格納されている行を参照する場合があります。

グローバル索引ではレンジ・パーティション化またはハッシュ・パーティション化が可能ですが、定義するパーティション表はどのタイプ

でもかまいません。グローバル索引は、GLOBAL属性を指定することで作成されます。データベース管理者は、グローバル索引の作成時に最初のパーティション化を定義して、それ以降はパーティション化のメンテナンスを行う必要があります。索引パーティションは、必要に応じてマージまたは分割できます。

通常、グローバル索引を、基礎となる表と同一レベルでパーティション化することはありません。基礎となる表と同一レベルで索引をパーティション化することにデメリットはありませんが、問合せ計画を生成したりパーティションのメンテナンス操作を実行したりする際に、Oracleが同一レベル・パーティション化のメリットを利用することもありません。したがって、基礎となる表と同一レベルでパーティション化する索引は、LOCALとして作成する必要があります。

グローバル・パーティション索引には、すべてのパーティションのすべての行に対するエントリを持つBツリーが1つ含まれます。各索引パーティションは、表の中の様々なパーティションまたはサブパーティションを参照するキーを含むことができます。

グローバル索引の最上位パーティションのパーティション・バウンドは、含まれるすべての値がMAXVALUEであることが必要です。これにより、基礎となる表のすべての行を、確実に索引の中で表すことができるようになります。

次の内容について説明します。

- 同一キーおよび非同一キーのグローバル・パーティション索引
- グローバル・パーティション索引の管理

3.3.2.1 同一キーおよび非同一キーのグローバル・パーティション索引

索引列の左プリフィックスでパーティション化されているグローバル・パーティション索引は、同一キー索引です。

索引列の左プリフィックスでパーティション化されていないグローバル・パーティション索引は、非同一キー索引です。Oracleでは、グローバル非同一キー・パーティション索引はサポートされていません。例は図3-6を参照してください。

グローバル同一キー・パーティション索引は、一意にも非一意にもできます。非パーティション索引は、グローバル同一キー非パーティション索引として扱われます。

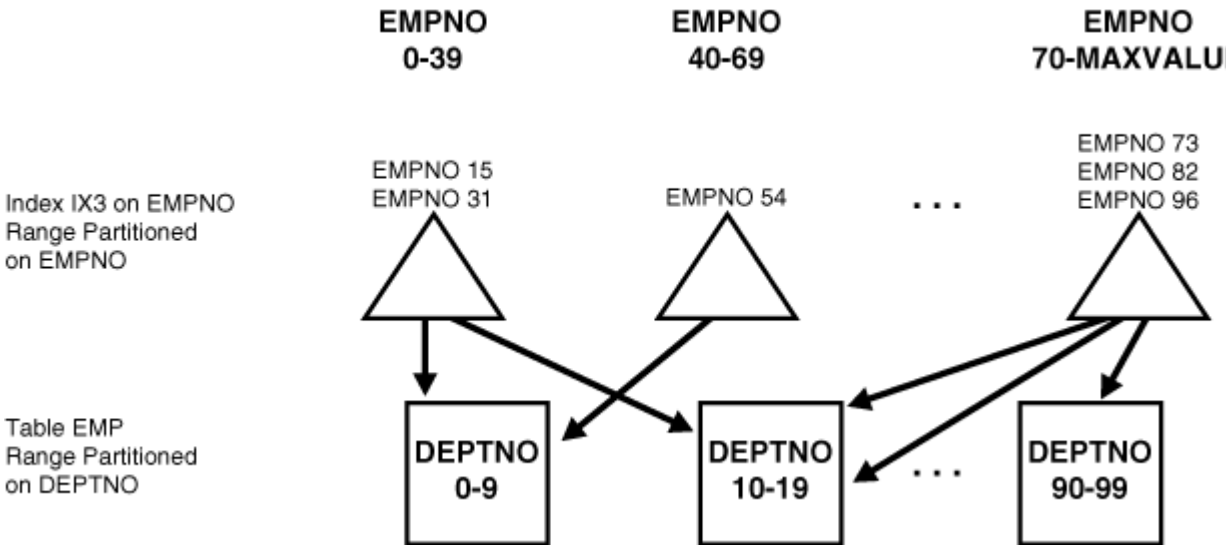
3.3.2.2 グローバル・パーティション索引の管理

グローバル・パーティション索引の管理には、いくつかの課題があります。

次の理由から、グローバル・パーティション索引は、ローカル索引よりも管理が煩雑です。

- 基礎となる表のパーティションのデータが移動または削除された場合(SPLIT、MOVE、DROPまたはTRUNCATE)、グローバル索引のすべてのパーティションが影響を受けます。つまり、グローバル索引では、パーティションの独立性がサポートされていません。
- 基礎となる表のパーティションまたはサブパーティションをある時点までリカバリする場合は、グローバル索引の対応するすべてのエントリを同じ時点までリカバリする必要があります。これらのエントリは、索引のすべてのパーティションまたはサブパーティションに点在していたり、リカバリしない他のパーティションまたはサブパーティションのエントリと混在していたりする可能性があるので、この処理を行うには、グローバル索引全体を再作成する以外に方法はありません。

図3-6 グローバル同一キー・パーティション索引



3.3.3 パーティション索引のまとめ

このトピックでは、パーティション索引タイプの概要を示します。

表3-1に、Oracleでサポートされている各タイプのパーティション索引をまとめます。重要な点は次のとおりです。

- 索引がローカルの場合は、基礎となる表と同一レベルでパーティション化されます。これ以外の索引はグローバルです。
- 同一キー索引は、索引列の左プリフィックスでパーティション化されます。これ以外の索引は非同一キーです。

表3-1 パーティション索引のタイプ

索引のタイプ	表と同一レベルでパーティション化された索引	索引列の左プリフィックスでパーティション化された索引	UNIQUE属性の可/不可	例: 表のパーティション化キー	例: 索引列	例: 索引のパーティション化キー
ローカル同一キー(任意のパーティション化方法)	可	可	可	A	A、B	A
ローカル非同一キー(任意のパーティション化方法)	可	不可	可脚注 1	A	B、A	A
グローバル同一キー(レンジ・パーティション化のみ)	不可脚注 2	可	可	A	B	B

脚注1

一意ローカル非同一キー索引の場合、パーティション化キーは索引キーのサブセットにする必要があり、部分索引にできません。

脚注2

グローバル・パーティション索引は基礎となる表と同一レベルでパーティション化することもできますが、Oracleが同一レベル・パーティション化のメリットを利用したり、パーティションのメンテナンス操作(DROPまたはSPLIT PARTITION)の後に同一レベル・パーティション化をメンテナンスしたりすることはありません。

3.3.4 非同一キー索引の重要性

非同一キー索引は、履歴データベースで特に役立つため重要です。

履歴データを含む表では、索引を1つの列に定義して、その列への高速アクセスの要件を満たすことが一般的です。ただし、古いデータの削除と新しいデータの取得の期間を合せるために、索引を別の列(基礎となる表と同じ列)でパーティション化することもできます。

sales表が週単位でパーティション化されているとします。1年分のデータが含まれ、13個のパーティションに分かれます。week_noでレンジ・パーティション化されており、4週が1パーティションになります。非同一キー・ローカル索引sales_ixをsalesに作成します。問合せでは口座番号を使用してデータに高速アクセスする必要があるため、sales_ix索引はacct_noに定義されます。ただし、これはsales表に合せてweek_noでパーティション化されています。4週ごとにsalesとsales_ixの一番古いパーティションが削除され、新しいパーティションが追加されます。

3.3.5 同一キー索引および非同一キー索引がパフォーマンスに与える影響

同一キー索引および非同一キー索引がパフォーマンスに与える影響があります。

同一キー索引を使用すると、パーティション・プルーニングの取得の見込みが、非同一キー索引を使用する場合と比べて大幅に高くなります。列が索引の一部である場合、この列はフィルタ述語として使用されると想定でき、これは、フィルタされた列が同一キー索引列の場合に、ある程度のプルーニングであることを自動的に意味します。この結果は、同一キー索引のプローブは、非同一キー索引のプローブよりも低コストであることを意味します。索引が同一キー索引のときに(ローカルまたはグローバルのいずれでも)、索引列を含む条件がOracleに発行されると、パーティション・プルーニングによって、条件の適用範囲を索引パーティションのサブセットに限定できます。

たとえば、図3-4において、条件がdeptno=15である場合、オプティマイザはこの条件を索引の2番目のパーティションにのみ適用すればよいことを認識します。(条件にバインド変数が含まれている場合、オプティマイザは条件を適用するべきパーティションを正確には認識できませんが、関係があるパーティションは1つのみであることはわかっており、実行時には1つの索引パーティションにのみアクセスします。)

索引が非同一キー索引である場合、Oracleは通常、索引列を含む条件をN個の索引パーティションすべてに適用する必要があります。このためには、単一のキーを参照するか、または索引レンジ・スキャンを実行する必要があります。レンジ・スキャンの場合、Oracleは、N個の索引パーティションの情報を結合する必要もあります。たとえば、図3-5で、ローカル索引はchkdateでパーティション化されており、索引キーはacctnoにあります。条件がacctno=31である場合、Oracleは12個の索引パーティションすべてをプローブします。

パーティション列に対する条件もある場合は、索引のプローブを複数実行する必要はありません。Oracleは、ローカル索引は基礎となる表と同一レベルでパーティション化されるという特性を利用し、パーティション・キーに基づいてパーティションをプルーニングします。たとえば、図3-5において条件がchkdate<3/97である場合、Oracleがプローブする必要のあるパーティションは2つのみです。

このように、非同一キー索引では、パーティション・キーがWHERE句に含まれるが索引キーの一部ではない場合には、オプティマイザが、基礎となる表のパーティションに基づいてプローブするべき索引パーティションを判断します。

ローカルの非同一キー索引のキーを使用する大量の問合せおよびDML文がすべての索引パーティションをプローブする必要がある場合は、この仕組みによって、ローカル非同一キー索引がもたらすパーティションの独立性の程度が実質的に低くなります。

表3-2 同一キー・ローカル索引、非同一キー・ローカル索引およびグローバル索引の比較

索引の特性	同一キー・ローカル	非同一キー・ローカル	グローバル

索引の特性	同一キー・ローカル	非同一キー・ローカル	グローバル
一意索引の可/不可	可	可	可。パーティション列以外の列の索引を使用する場合はグローバルである必要がある。
管理のしやすさ	容易	容易	煩雑
OLTP	適切	不適切	適切
長時間実行(DSS)	適切	適切	適切でない

3.3.6 パーティション索引での拡張索引圧縮

パーティション索引での拡張索引圧縮により、索引の記憶域要件を低減できます。

拡張索引圧縮を使用して索引を作成することで、サポートされているすべての一意索引と非一意索引のサイズが低減します。拡張索引圧縮は、索引への効率的なアクセスを提供しつつ、圧縮率を著しく向上させます。拡張圧縮は、接頭辞圧縮の候補として適切ではない索引を含め、サポートされているすべての索引で適切に機能します。

パーティション索引の場合は、パーティションごとにパーティションの圧縮タイプを指定できます。親索引が圧縮されていない場合でも、索引パーティションに対して拡張索引圧縮を指定できます。

次の例は、パーティション索引での圧縮属性の混在を示しています。

```
CREATE INDEX my_test_idx ON test(a, b) COMPRESS ADVANCED HIGH LOCAL
   (PARTITION p1 COMPRESS ADVANCED LOW,
    PARTITION p2 COMPRESS,
    PARTITION p3,
    PARTITION p4 NOCOMPRESS);
```

次の例では、親索引が圧縮されていないパーティションでの拡張索引圧縮のサポートを示します。

```
CREATE INDEX my_test_idx ON test(a, b) NOCOMPRESS LOCAL
   (PARTITION p1 COMPRESS ADVANCED LOW,
    PARTITION p2 COMPRESS ADVANCED HIGH,
    PARTITION p3);
```

関連項目:

拡張索引圧縮の詳細は、『Oracle Database管理者ガイド』を参照してください

3.3.7 索引のパーティション化のガイドライン

索引のパーティション化に関しては、いくつかのガイドラインがあります。

表の索引のパーティション化方法を決める際は、表にアクセスする必要があるアプリケーションの組合せを考慮します。どの方法を使用するかは、パフォーマンスと、可用性および管理性とのトレードオフになります。この項では、考慮する必要のあるいくつかのガイドラインを示します。

- OLTPアプリケーションの場合:

- グローバル索引およびローカル同一キー索引では、索引パーティションのプローブ数が最小限になるので、ローカル非同一キー索引よりもパフォーマンスが向上します。
- ローカル索引では、表のパーティションまたはサブパーティションのメンテナンス操作中でも、より高い可用性が得られます。ローカル非同一キー索引は、履歴データベースで特に有効です。

- DSSアプリケーションでは、ローカル非同一キー索引によってパフォーマンスを向上させることができます。これは、索引キーに基づくレンジ問合せによって、多数の索引パーティションをパラレルにスキャンできるためです。

  たとえば、図3-5のchecks表に対してacctno between 40 and 45という述語を使用する問合せでは、非同一キー索引ix3のすべてのパーティションのパラレル・スキャンが実行されます。また、図3-4のdeptno表に対してdeptno BETWEEN 40 AND 45という述語を使用する問合せは、同一キー索引ix1の1つのパーティションにアクセスするため、パラレル化できません。

- 履歴表では、索引はできるかぎりローカルにする必要があります。これにより、定期的なパーティション削除操作の影響を抑えることができます。
- パーティション列以外の列の一意索引はグローバルにする必要があります。これは、キーにパーティション化キーが含まれていない一意ローカル非同一キー索引はサポートされていないためです。
- 使用できない索引は領域を消費しません。

関連項目:

表の管理のガイドラインの詳細は、『Oracle Database管理者ガイド』を参照してください

3.3.8 索引パーティションの物理属性

このトピックでは、索引パーティションの物理属性について説明します。

デフォルトの物理属性は、CREATE INDEX文でパーティション索引を作成する際に初期指定されます。パーティション索引自体に対応するセグメントはないので、これらの属性はメンバー・パーティションの物理属性を導出する際にのみ使用されます。デフォルトの物理属性は、ALTER INDEX MODIFY DEFAULT ATTRIBUTESを使用して後から変更できます。

CREATE INDEXで作成されるパーティションの物理属性は、次のようにして決定されます。

- 索引に指定される(明示的またはデフォルト)物理属性の値が使用されるのは、対応するパーティション属性の値が指定されない場合です。LOCAL索引のパーティションのTABLESPACE属性の扱いは、このルールの重要な例外です。つまり、ユーザー指定のTABLESPACE値(パーティション・レベルと索引レベルの両方)がない場合、基礎となる表の対応するパーティションの物理属性の値が使用されます。
- ALTER TABLE ADD PARTITIONの処理中に作成されるローカル索引のパーティションの物理属性(前項で説明したTABLESPACE以外)は、各索引のデフォルトの物理属性に設定されます。

ALTER TABLE SPLIT PARTITIONで作成される索引パーティションの物理属性(TABLESPACE以外)は、次のようにして決定されます。

- 分割される索引パーティションの物理属性の値が使用されます。

既存の索引パーティションの物理属性は、ALTER INDEX MODIFY PARTITIONおよびALTER INDEX REBUILD PARTITIONで変更できます。その結果の属性は、次のようにして決定されます。

- 新しい値が指定されていない場合は、文が発行される前のパーティションの物理属性の値が使用されます。ALTER

INDEX REBUILD PARTITION SQL文で、パーティションが含まれる表領域を変更できます。

ALTER INDEX SPLIT PARTITIONで作成されるグローバル索引パーティションの物理属性は、次のようにして決定されます。

- 新しい値が指定されていない場合は、分割されるパーティションの物理属性の値が使用されます。
- 索引のすべてのパーティションの(デフォルト値を持つ)物理属性は、ALTER INDEXで変更できます。たとえば、ALTER INDEX indexname NOLOGGINGは、indexnameのすべてのパーティションのロギング・モードをNOLOGGINGに変更します。

関連項目:

パーティションの追加や索引の再作成の詳しい例は、「パーティションの管理」を参照してください

3.4 パーティション化と表の圧縮

圧縮は、複数のパーティション、または1つの完成したパーティション化されたヒープ構成表に対して実行できます。

完全なパーティション化された表を圧縮対象として定義するかパーティション・レベルごとに定義することで、この圧縮を実行できます。特定の宣言のないパーティションは表定義から属性を継承し、表レベルの指定がない場合は表領域定義から継承します。

パーティションを圧縮するかまたは未圧縮のままにするかについての決定は、パーティション化されていない表と同じルールに従います。ただし、データを論理的に個別パーティションに区切るパーティション化のため、パーティション表は、データ(パーティション)の圧縮部分として適切な候補です。たとえばこれは、古いデータが使用不可になる前の中間の段階としてのすべてのローリング・ウィンドウ操作に役立ちます。データ・セグメントを圧縮することにより、より多くの古いデータをオンラインで保持でき、追加の記憶域の使用量の負荷を最小化できます。

圧縮されていない既存の表パーティションを後で変更したり、圧縮済または圧縮されていない新しいパーティションを追加したり、データの移動を必要とするパーティション・メンテナンス操作(MERGE PARTITION、SPLIT PARTITION、MOVE PARTITIONなど)の中で圧縮属性を変更することもできます。パーティションにはデータを含めることもできますが、空のままでもかまいません。

部分的または完全に圧縮されたパーティション表のアクセスおよびメンテナンスは、まったく圧縮されていないパーティション表の場合と同じです。まったく圧縮されていないパーティション表に適用されることは、部分的または完全に圧縮されたパーティション表にも適用されます。

次の内容について説明します。

- 表の圧縮とビットマップ索引
- 表の圧縮とパーティション化の例

関連項目:

- データ・ウェアハウスの最適化および技術の概要は、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください
- 表の管理のガイドラインの詳細は、『Oracle Database管理者ガイド』を参照してください
- 圧縮要素の予測は、『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

3.4.1 表の圧縮とビットマップ索引

ビットマップ索引のあるパーティション表で圧縮を使用するには、必要なステップがいくつかあります。

ビットマップ索引のあるパーティション表に対して表の圧縮を使用するには、最初に圧縮属性を導入する前に、次の処理を行う必要があります。

1. ビットマップ索引を使用不可としてマークします。
2. 圧縮属性を設定します。
3. 索引を再作成します。

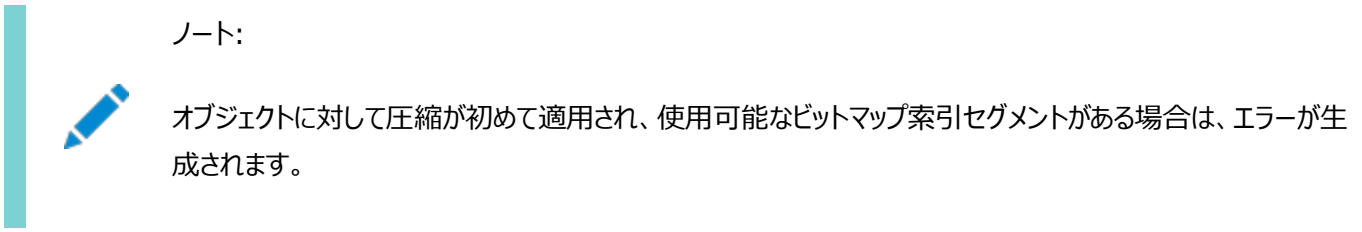
圧縮パーティションを、まったく圧縮されていない既存のパーティション表に含める場合は、圧縮パーティションを追加する前に、既存のビットマップ索引をすべて削除するか、UNUSABLEとしてマークする必要があります。これは、パーティションにデータが含まれるかどうかにかかわらず実行する必要があります。また、表の1つ以上のパーティションを圧縮する操作とも無関係です。これは、

Bツリー索引のみを含むパーティション表には適用されません。

表の圧縮が有効になっている各データ・ブロックでは、多数の行を収容する可能性があるため、ビットマップ索引構造のこのような再構築が必要です。表の圧縮の有効化は、最初に1回だけ実行する必要があります。その後のすべての操作は、圧縮されたパーティションや圧縮されていないパーティションへの影響や、圧縮属性の変更にかかわらず、圧縮されていないパーティション表、部分的に圧縮されたパーティション表または完全に圧縮されたパーティション表で、理想的に動作します。

パーティション表の部分的または完全な圧縮を将来計画している場合は、ビットマップ索引構造の再作成を回避するために、すべてのパーティション表を作成するときに圧縮パーティションを少なくとも1つ指定することをお薦めします。この圧縮パーティションは空のまま残しても、パーティション表を作成した後に削除してもかまいません。

圧縮パーティションを含むパーティション表では、未圧縮パーティション部分のビットマップ索引構造が多少大きくなる可能性があります。ただし、圧縮パーティションのビットマップ索引構造は、表の圧縮を行う前の適切なビットマップ索引構造よりも、通常は多少小さくなります。これは、実際の圧縮率によってかなり異なります。



ノート:

オブジェクトに対して圧縮が初めて適用され、使用可能なビットマップ索引セグメントがある場合は、エラーが生成されます。

3.4.2 表の圧縮とパーティション化の例

このトピックでは、パーティション表による表圧縮の例について説明します。

次の文では、表salesの既存のパーティションsales_q1_1998が移動および圧縮されます。

```
ALTER TABLE sales
  MOVE PARTITION sales_q1_1998 TABLESPACE ts_arch_q1_1998 COMPRESS;
```

または、次のようにハイブリッド列圧縮(HCC)を選択できます。

```
ALTER TABLE sales
  MOVE PARTITION sales_q1_1998 TABLESPACE ts_arch_q1_1998
  COMPRESS FOR ARCHIVE LOW;
```

MOVE文を使用すると、パーティションsales_q1_1998のローカル索引が使用不可になります。ローカル索引は、次のように、後で再作成する必要があります。

```
ALTER TABLE sales
  MODIFY PARTITION sales_q1_1998 REBUILD UNUSABLE LOCAL INDEXES;
```

ユーザーが表にアクセスすることで悪影響が生じないように、操作全体を自動的に完了させるために、MOVE文にUPDATE INDEXES句を含めることもできます。

次の文では、既存の2つのパーティションが、別の表領域に新しい圧縮パーティションとしてマージされます。ローカル・ビットマップ索引は、次のように、後で再作成する必要があります。

```
ALTER TABLE sales MERGE PARTITIONS sales_q1_1998, sales_q2_1998
  INTO PARTITION sales_1_1998 TABLESPACE ts_arch_1_1998
  COMPRESS FOR OLTP UPDATE INDEXES;
```

関連項目:

- パーティション管理操作の詳細および例は、「パーティションの管理」を参照してください
- 表の圧縮を使用するときに圧縮率を予測する方法の詳細は、『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
- SQLの構文は、『Oracle Database SQL言語リファレンス』を参照してください
- ハイブリッド列圧縮の詳細は、『Oracle Database概要』を参照してください。ハイブリッド列圧縮は、特定のOracleストレージ・システムの機能です。

3.5 パーティション化戦略の選択に関する推奨事項

パーティション化計画の選択時には、パフォーマンスに関する考慮事項に基づいた次の推奨事項を確認してください。

次の各トピックでは、パーティション化戦略の選択に関する推奨事項を示します。

- レンジ・パーティション化または時間隔パーティション化を使用する場合
- ハッシュ・パーティション化を使用する場合
- リスト・パーティション化を使用する場合
- コンポジット・パーティション化を使用する場合
- 時間隔パーティション化を使用する場合
- 参照パーティション化を使用する場合
- 仮想列でパーティション化する場合
- 読取り専用表領域を使用する場合の考慮事項

3.5.1 レンジ・パーティション化または時間隔パーティション化を使用する場合

レンジまたは時間隔パーティション化は、特に日付および時間データなど類似のデータを整理する場合に便利です。

レンジ・パーティション化は、履歴データをパーティション化する場合に便利です。レンジ・パーティション化の境界は、表または索引内でのパーティションの順序を定義します。

時間隔パーティション化は、レンジ・パーティション化の機能が拡張されたものです。ある時点を過ぎると、パーティションが時間の間隔によって定義されます。データがパーティションに挿入されると、データベースによって時間隔パーティションが自動的に作成されます。

レンジ・パーティション化または時間隔パーティション化は、DATE型の列の時間隔でデータを整理するためによく使用されます。したがって、レンジ・パーティションにアクセスするほとんどのSQL文では期間が指定されます。たとえば、特定の期間のデータを選択するSQL文です。このようなシナリオで、各パーティションが1か月のデータを表す場合、「2006年12月のデータを検索する」という問合せは2006年12月のパーティションだけにアクセスする必要があります。これにより、スキャンされるデータ容量が、存在するデータ全体の一部にまで減少します。これはパーティション・プルーニングと呼ばれる最適化方法です。

レンジ・パーティション化は、パーティションを簡単に追加または削除できるため、定期的に新しいデータをロードして、古いデータを削除する場合にも最適です。たとえば、過去36か月間のデータをオンラインでアクセスできるように、データのローリング・ウィンドウを維持することが一般的です。レンジ・パーティション化によりこのプロセスが簡略化されます。新しい月のデータを追加するには、別の表にロードし、クリーニングし、索引を作成してから、EXCHANGE PARTITION文を使用してレンジ・パーティション表に追加します。この間、元の表はオンラインになっています。新しいパーティションを追加したら、DROP PARTITION文を使用して一番古い月を削除できます。DROP PARTITION文を使用するかわりに、パーティションをアーカイブして読取り専用にすることもできますが、この方法を利用できるのはパーティションが別の表領域にある場合のみです。パーティション表への挿入を使用して、データのローリング・ウィンドウを実装することもできます。

時間隔パーティション化を使用すると、データが届いたときに時間隔パーティションを自動的かつ容易に作成できるようになります。また、時間隔パーティションは、その他すべてのパーティション・メンテナンス操作でも使用できます。

つまり、次のような場合にレンジ・パーティション化または時間隔パーティション化の使用を検討してください。

- 非常に大規模な表が、適切なパーティショニング化列(ORDER_DATEやPURCHASE_DATEなど)の範囲述語により頻繁にスキャンされる場合。その列で表をパーティション化することで、パーティション・プルーニングが可能になります。

- データのローリング・ウィンドウを保持する場合。
- 大きい表では、割り当てられた時間枠内でバックアップやリストアなどの管理操作を完了できない場合。パーティション・レンジ列に基づいて表を小さい論理単位に分割できます。

例3-5では、2005年および2006年の2年間分の表salestableを作成し、その表を列s_salesdateの範囲でパーティション化して、8つの四半期にデータを分割します(1四半期が1つのパーティションに対応)。将来のパーティションは、月単位の間隔の定義により自動的に作成されます。時間隔パーティションは、ラウンドロビン法で指定リストの表領域に作成されます。短い間隔で売上数を分析するときに、パーティション・プルーニングを利用できます。sales表では、ローリング・ウィンドウの使用もサポートされます。

例3-5 レンジおよび時間隔パーティション化による表の作成

```
CREATE TABLE salestable
  (s_productid  NUMBER,
   s_saledate   DATE,
   s_custid     NUMBER,
   s_totalprice NUMBER)
PARTITION BY RANGE(s_saledate)
INTERVAL(NUMTOYMINTERVAL(1,'MONTH')) STORE IN (tbs1,tbs2,tbs3,tbs4)
 (PARTITION sal05q1 VALUES LESS THAN (TO_DATE('01-APR-2005', 'DD-MON-YYYY')) TABLESPACE tbs1,
  PARTITION sal05q2 VALUES LESS THAN (TO_DATE('01-JUL-2005', 'DD-MON-YYYY')) TABLESPACE tbs2,
  PARTITION sal05q3 VALUES LESS THAN (TO_DATE('01-OCT-2005', 'DD-MON-YYYY')) TABLESPACE tbs3,
  PARTITION sal05q4 VALUES LESS THAN (TO_DATE('01-JAN-2006', 'DD-MON-YYYY')) TABLESPACE tbs4,
  PARTITION sal06q1 VALUES LESS THAN (TO_DATE('01-APR-2006', 'DD-MON-YYYY')) TABLESPACE tbs1,
  PARTITION sal06q2 VALUES LESS THAN (TO_DATE('01-JUL-2006', 'DD-MON-YYYY')) TABLESPACE tbs2,
  PARTITION sal06q3 VALUES LESS THAN (TO_DATE('01-OCT-2006', 'DD-MON-YYYY')) TABLESPACE tbs3,
  PARTITION sal06q4 VALUES LESS THAN (TO_DATE('01-JAN-2007', 'DD-MON-YYYY')) TABLESPACE tbs4);
```

関連項目:

時間隔パーティションのパーティション・メンテナンス操作の詳細は、「パーティションの管理」を参照してください

3.5.2 ハッシュ・パーティション化を使用する場合

ハッシュ・パーティション化は、類似データをグループ化するのではなく、ハッシュ・アルゴリズムに基づいて、データをパーティションにランダムに分散する場合に便利です。

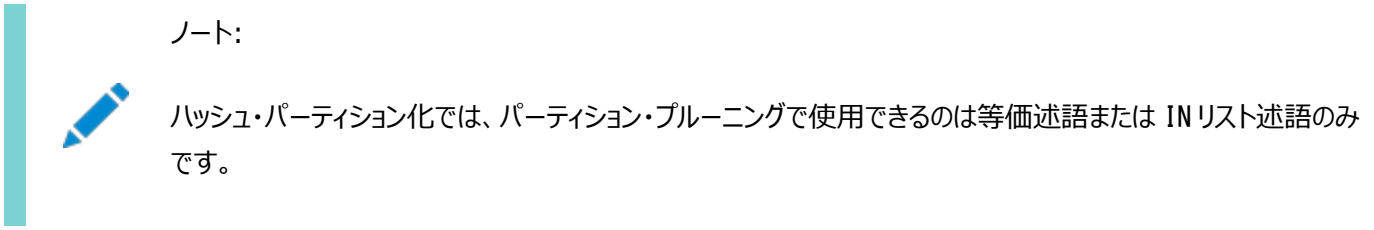
パーティション化キーを指定できても、どのパーティションにデータを含めるべきかがわかりにくい場合があります。また、レンジ・パーティション化のように、類似したデータをグループ化するのではなく、データの業務上の意味や論理的な意味に対応しないようにデータを分散することが望ましい場合があります。ハッシュ・パーティション化では、パーティション化キーをハッシング・アルゴリズムに渡した結果に基づいて行がパーティションに配置されます。

この方法を使用すると、データはグループ化されるのではなく複数のパーティションにランダムに分散されます。この方法はデータによっては適していますが、履歴データを管理する方法としては効率的ではありません。ただし、ハッシュ・パーティションの一部のパフォーマンス特性は、レンジ・パーティションと同じです。たとえば、パーティション・プルーニングが行われるのは等価述語の場合のみです。また、パーティション・ワイズ結合、パラレル索引アクセスおよびパラレルDMLを使用することもできます。

一般的なルールとして、次のような場合にハッシュ・パーティション化を使用します。

- パーティション・サイズが同一である可能性が高いときに、パラレルのパーシャル・パーティション・ワイズ結合またはフル・パーティション・ワイズ結合を使用可能にする場合。

- Oracle Real Application Clustersを使用するMPPプラットフォームのノード間に、データを均等に分散させる場合。結果として、インターノード・パラレル文を処理するときにインターコネクト・トラフィックが最小限になります。
- パーティション化キー(多くの場合、値指定か値リストの制約がある)に基づいてパーティション・プルーニングおよびパーティション・ワイズ結合を使用する場合。
- 使用可能なすべてのデバイスに対してストライプ化およびミラー化を行うストレージ管理方法を使用しないときに、I/Oボトルネックを回避するためにデータをランダムに分散させる場合。



ノート:

ハッシュ・パーティション化では、パーティション・プルーニングで使用できるのは等価述語または INリスト述語のみです。

最適なデータ分散を実現するには次の要件を満たす必要があります。

- 一意またはほぼ一意の列または列の組合せを選択します。
- 作成するパーティションの数と、各パーティションのサブパーティションの数を2の累乗にします。たとえば、2、4、8、16、32、64、128などです。

例3-6では、パーティション化キーとして列s_productidを使用して、表sales_hashに4つのハッシュ・パーティションを作成します。productsとのパラレル結合では、パーシャル・パーティション・ワイズ結合またはフル・パーティション・ワイズ結合を利用できます。1つの製品または一連の製品の売上金額にアクセスする問合せでは、パーティション・プルーニングが利用されます。

パーティション名を明示的に指定せず、ハッシュ・パーティション数を指定した場合は、パーティションの内部名が自動的に生成されます。また、STORE IN句を使用して、ラウンドロビン法で表領域にハッシュ・パーティションを割り当てることもできます。

例3-6 ハッシュ・パーティション化による表の作成

```
CREATE TABLE sales_hash
  (s_productid  NUMBER,
   s_saledate   DATE,
   s_custid     NUMBER,
   s_totalprice NUMBER)
PARTITION BY HASH(s_productid)
( PARTITION p1 TABLESPACE tbs1
, PARTITION p2 TABLESPACE tbs2
, PARTITION p3 TABLESPACE tbs3
, PARTITION p4 TABLESPACE tbs4
);
```

関連項目:

- パーティション・ワイズ結合の詳細は、「パーティション・ワイズ操作」を参照してください
- VLDBの記憶域の管理の詳細は、「VLDBの記憶域管理」を参照してください
- ハッシュ・パーティション表の作成のその他の例は、「パーティションの管理」を参照してください
- パーティション化の構文は、『Oracle Database SQL言語リファレンス』を参照してください

3.5.3 リスト・パーティション化を使用する場合

リスト・パーティション化は、離散値に基づいて行をパーティションに明示的にマップする場合に便利です。

例3-7では、オレゴン州とワシントン州のすべての顧客が1つのパーティションに格納され、他の州の顧客はその他のパーティションに格納されているとします。地域ごとに顧客の取引を分析する顧客口座担当者は、パーティション・プルーニングを利用できます。

例3-7 リスト・パーティション化による表の作成

```
CREATE TABLE accounts
( id             NUMBER
, account_number NUMBER
, customer_id    NUMBER
, branch_id      NUMBER
, region         VARCHAR(2)
, status         VARCHAR2(1)
)
PARTITION BY LIST (region)
( PARTITION p_northwest VALUES ('OR', 'WA')
, PARTITION p_southwest VALUES ('AZ', 'UT', 'NM')
, PARTITION p_northeast VALUES ('NY', 'VM', 'NJ')
, PARTITION p_southeast VALUES ('FL', 'GA')
, PARTITION p_northcentral VALUES ('SD', 'WI')
, PARTITION p_southcentral VALUES ('OK', 'TX')
);
```

3.5.4 コンポジット・パーティション化を使用する場合

コンポジット・パーティション化には、複数のディメンションでパーティション化する利点があります。

パフォーマンスの観点では、SQL文にもよりますが1つまたは2つのディメンションでのパーティション・プルーニングを利用できます。また、いずれかのディメンションでのフル・パーティション・ワイズ結合またはパーシャル・パーティション・ワイズ結合を使用できます。

1つの表でパラレル・バックアップおよびパラレル・リカバリを利用できます。また、コンポジット・パーティション化ではパーティション数が大幅に増加するため、効果的なパラレル実行が可能になります。管理の観点では、履歴データをサポートするためのローリング・ウィンドウを実装でき、多くの文がパーティション・プルーニングまたはパーティション・ワイズ結合の恩恵を受ける場合には、別のディメンションでのパーティション化も可能です。

表のバックアップを分割して、パーティション化キーによる指定に基づいてデータの格納方法を変更することができます。たとえば、特定の製品タイプのデータは読取り専用として圧縮形式で格納し、その他の製品タイプのデータは圧縮せずに保存することができます。

データベースでは、コンポジット・パーティション化された表の各サブパーティションが個別のセグメントとして格納されます。このため、サブパーティションのプロパティは、表のプロパティや、そのサブパーティションが属するパーティションとは異なる場合があります。

次の内容について説明します。

- レンジ・パーティション化または時間隔パーティション化を使用する場合
- ハッシュ・パーティション化を使用する場合
- リスト・パーティション化を使用する場合
- コンポジット・パーティション化を使用する場合
- 時間隔パーティション化を使用する場合
- 参照パーティション化を使用する場合

- 仮想列でパーティション化する場合

関連項目:

構文と制約の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

3.5.4.1 コンポジット・レンジ・ハッシュ・パーティション化を使用する場合

コンポジット・レンジ - ハッシュ・パーティション化が特によく使用されるのは、履歴を格納する表、結果として非常に大規模になる表、および他の大きな表と頻繁に結合される表です。

このようなタイプの表(データ・ウェアハウス・システムの典型的な表)では、コンポジット・レンジ・ハッシュ・パーティション化によって、レンジ・レベルでのパーティション・プルーニングと、ハッシュ・レベルでのパラレルのフル・パーティション・ワイズ結合またはパーシャル・パーティション・ワイズ結合を利用できるようになります。場合によっては、特定のSQL文で両方のディメンションでのパーティション・プルーニングを利用できます。

また、コンポジット・レンジ・ハッシュ・パーティション化は、従来ハッシュ・パーティション化を使用し、同時にローリング・ウィンドウ方法も利用していた表で使用できます。時間経過に伴い、データをストレージ層間で移動し、圧縮して読取り専用表領域に格納し、最終的にはパージすることができます。情報ライフサイクル管理(ILM)シナリオでは、階層ストレージ方式を実装するためにレンジ・パーティションがよく使用されます。

例3-8は、インターネット・サービス・プロバイダのレンジ・ハッシュ・パーティション化されたpage_history表です。この表定義は、特定のclient_ip値(問合せがパーティション・プルーニングを利用可能)または多数のIPアドレス(問合せがフル・パーティション・ワイズ結合またはパーシャル・パーティション・ワイズ結合を利用可能)に関する履歴分析用に最適化されています。

この例では、時間隔パーティション化が使用されています。データが表に挿入されたときに時間隔パーティションを自動的に作成するために、レンジ・パーティション化に加えて時間隔パーティション化を使用できます。

例3-8 コンポジット・レンジ - ハッシュ・パーティション化による表の作成

```
CREATE TABLE page_history
( id                NUMBER NOT NULL
, url               VARCHAR2(300) NOT NULL
, view_date         DATE NOT NULL
, client_ip         VARCHAR2(23) NOT NULL
, from_url          VARCHAR2(300)
, to_url            VARCHAR2(300)
, timing_in_seconds NUMBER
) PARTITION BY RANGE(view_date) INTERVAL (NUMTODSINTERVAL(1,'DAY'))
SUBPARTITION BY HASH(client_ip)
SUBPARTITIONS 32
(PARTITION p0 VALUES LESS THAN (TO_DATE('01-JAN-2006','dd-MON-yyyy')))
PARALLEL 32 COMPRESS;
```

関連項目:

情報ライフサイクル管理(ILM)およびパーティション化を使用した階層ストレージの実装の詳細は、「時間ベース情報の管理およびメンテナンス」を参照してください

3.5.4.2 コンポジット・レンジ・リスト・パーティション化を使用する場合

コンポジット・レンジ・リスト・パーティション化は、履歴データを格納する大きな表でよく使用され、一般的に複数のディメンションでアクセスされます。

通常は、データの履歴ビューが1つのアクセス・パスですが、業務によってはアクセス・パスとしてその他のカテゴリも使用されます。たとえば、地区の経理担当者は、その地区で特定の期間に加入した新顧客数に高い関心があります。ILMとその階層ストレージ方式が、レンジ・リスト・パーティション表を作成する一般的な理由です。古いデータを移動して圧縮しても、リスト・ディメンションのパーティション・プルーニングを引き続き行うことができます。

例3-9では、レンジ・リスト・パーティション化されたcall_detail_records表を作成します。電気通信会社はこの表を使用して、時間経過に応じて特定の種類の通信を分析できます。この表では、from_numberとto_numberに対するローカル索引が使用されます。

この例では、時間隔パーティション化が使用されています。データが表に挿入されたときに時間隔パーティションを自動的に作成するために、レンジ・パーティション化に加えて時間隔パーティション化を使用できます。

例3-9 コンポジット・レンジ - リスト・パーティション化による表の作成

```
CREATE TABLE call_detail_records
( id NUMBER
, from_number         VARCHAR2(20)
, to_number           VARCHAR2(20)
, date_of_call        DATE
, distance            VARCHAR2(1)
, call_duration_in_s NUMBER(4)
) PARTITION BY RANGE(date_of_call)
INTERVAL (NUMTODSINTERVAL(1,'DAY'))
SUBPARTITION BY LIST(distance)
SUBPARTITION TEMPLATE
( SUBPARTITION local VALUES('L') TABLESPACE tbs1
, SUBPARTITION medium_long VALUES ('M') TABLESPACE tbs2
, SUBPARTITION long_distance VALUES ('D') TABLESPACE tbs3
, SUBPARTITION international VALUES ('I') TABLESPACE tbs4
)
(PARTITION p0 VALUES LESS THAN (TO_DATE('01-JAN-2005','dd-MON-yyyy')))
PARALLEL;

CREATE INDEX from_number_ix ON call_detail_records(from_number)
LOCAL PARALLEL NOLOGGING;

CREATE INDEX to_number_ix ON call_detail_records(to_number)
LOCAL PARALLEL NOLOGGING;
```

3.5.4.3 コンポジット・レンジ・レンジ・パーティション化を使用する場合

コンポジット・レンジ - レンジ・パーティション化は、時間関連のデータを複数の時間ディメンションで格納するアプリケーションに役立ちます。

多くの場合、このようなアプリケーションは、データにアクセスするために特定の1つの時間ディメンションを使用するのではなく、別の時間ディメンションを使用することもあれば、同時に両方の時間ディメンションを使用することもあります。たとえば、Webショップが、発注時間と出荷時間(運送会社に渡した時間)に基づいて売上データを分析する場合があります。

コンポジット・レンジ・レンジ・パーティション化の他の業務例としては、ILMシナリオや、履歴データを格納して、データを別のディメンションの範囲で分類するアプリケーションがあります。

例3-10では、レンジ・レンジ・パーティション化された表account_balance_historyを示します。銀行は、低残高通知や特定

の顧客カテゴリ向けキャンペーンについて顧客に連絡するために、個々のサブパーティションへのアクセスを利用できます。

この例では、時間隔パーティション化が使用されています。データが表に挿入されたときに時間隔パーティションを自動的に作成するために、レンジ・パーティション化に加えて時間隔パーティション化を使用できます。ここでは、2007年1月1日(月曜日)から開始する7日間(1週間)の間隔が作成されています。

例3-10 コンポジット・レンジ - レンジ・パーティション化による表の作成

```
CREATE TABLE account_balance_history
( id                 NUMBER NOT NULL
, account_number     NUMBER NOT NULL
, customer_id        NUMBER NOT NULL
, transaction_date   DATE NOT NULL
, amount_credited    NUMBER
, amount_debited     NUMBER
, end_of_day_balance NUMBER NOT NULL
) PARTITION BY RANGE(transaction_date)
INTERVAL (NUMTODSINTERVAL(7,'DAY'))
SUBPARTITION BY RANGE(end_of_day_balance)
SUBPARTITION TEMPLATE
( SUBPARTITION unacceptable VALUES LESS THAN (-1000)
, SUBPARTITION credit VALUES LESS THAN (0)
, SUBPARTITION low VALUES LESS THAN (500)
, SUBPARTITION normal VALUES LESS THAN (5000)
, SUBPARTITION high VALUES LESS THAN (20000)
, SUBPARTITION extraordinary VALUES LESS THAN (MAXVALUE)
)
(PARTITION p0 VALUES LESS THAN (TO_DATE('01-JAN-2007','dd-MON-yyyy')));
```

3.5.4.4 コンポジット・リスト・ハッシュ・パーティション化を使用する場合

コンポジット・リスト - ハッシュ・パーティショニングが役立つのは、通常1つのディメンションに関してアクセスされる大規模な表で、(そのサイズのために)他の大きな表との結合で、別のディメンションに関してパラレルのフル・パーティション・ワイズ結合またはパーシャル・パーティション・ワイズ結合を利用する必要がある表です。

例3-11にcredit_card_accounts表を示します。この表は、経理担当者がその地域の口座にすぐにアクセスできるように、regionでリスト・パーティション化されています。サブパーティション計画はcustomer_idでのハッシュです。これにより、transactions表(customer_idでサブパーティション化されている)に対する問合せはフル・パーティション・ワイズ結合を利用できます。ハッシュ・パーティション化されたcustomers表との結合では、フル・パーティション・ワイズ結合を利用することもできます。この表にはis_active列にローカル・ビットマップ索引があります。

例3-11 コンポジット・リスト - ハッシュ・パーティション化による表の作成

```
CREATE TABLE credit_card_accounts
( account_number  NUMBER(16) NOT NULL
, customer_id     NUMBER NOT NULL
, customer_region VARCHAR2(2) NOT NULL
, is_active       VARCHAR2(1) NOT NULL
, date_opened     DATE NOT NULL
) PARTITION BY LIST (customer_region)
SUBPARTITION BY HASH (customer_id)
SUBPARTITIONS 16
( PARTITION emea VALUES ('EU','ME','AF')
, PARTITION amer VALUES ('NA','LA')
, PARTITION apac VALUES ('SA','AU','NZ','IN','CH')
) PARALLEL;

CREATE BITMAP INDEX is_active_bix ON credit_card_accounts(is_active)
```

```
LOCAL PARALLEL NOLOGGING;
```

3.5.4.5 コンポジット・リスト・リスト・パーティション化を使用する場合

コンポジット・リスト・リスト・パーティション化が役立つのは、様々なディメンションに関してアクセスされることが多い大きな表です。

特に、離散値に基づいてそれらのディメンションに行をマップできます。

例3-12に、非常にアクセス回数の多いcurrent_inventory表を示します。この表は、スーパーマーケット納入業者の地方倉庫の現在の在庫を反映するように絶えず更新されています。生鮮食品はその倉庫からスーパーマーケットに供給されるため、供給と配送を最適化することが重要です。この表は、warehouse_idとproduct_idにローカル索引があります。

例3-12 コンポジット・リスト - リスト・パーティション化による表の作成

```
CREATE TABLE current_inventory
( warehouse_id      NUMBER
, warehouse_region  VARCHAR2(2)
, product_id        NUMBER
, product_category  VARCHAR2(12)
, amount_in_stock   NUMBER
, unit_of_shipping  VARCHAR2(20)
, products_per_unit NUMBER
, last_updated      DATE
) PARTITION BY LIST (warehouse_region)
SUBPARTITION BY LIST (product_category)
SUBPARTITION TEMPLATE
( SUBPARTITION perishable VALUES ('DAIRY','PRODUCE','MEAT','BREAD')
, SUBPARTITION non_perishable VALUES ('CANNED','PACKAGED')
, SUBPARTITION durable VALUES ('TOYS','KITCHENWARE')
)
( PARTITION p_northwest VALUES ('OR', 'WA')
, PARTITION p_southwest VALUES ('AZ', 'UT', 'NM')
, PARTITION p_northeast VALUES ('NY', 'VM', 'NJ')
, PARTITION p_southeast VALUES ('FL', 'GA')
, PARTITION p_northcentral VALUES ('SD', 'WI')
, PARTITION p_southcentral VALUES ('OK', 'TX')
);

CREATE INDEX warehouse_id_ix ON current_inventory(warehouse_id)
LOCAL PARALLEL NOLOGGING;

CREATE INDEX product_id_ix ON current_inventory(product_id)
LOCAL PARALLEL NOLOGGING;
```

3.5.4.6 コンポジット・リスト・レンジ・パーティション化を使用する場合

コンポジット・リスト・レンジ・パーティション化が役立つのは、様々なディメンションに関してアクセスされる大きな表です。

最もよく使用されるディメンションでは、離散値について行をパーティションに具体的にマップすることができます。リスト・レンジ・パーティション化は、リスト・パーティション内で範囲値を使用する表でよく使用されます。一方、レンジ・リスト・パーティション化は、レンジ・パーティション内の離散リスト値についてよく使用されます。リスト・レンジ・パーティション化は、履歴データの格納にはそれほど使用されませんが、同様のシナリオにはすべて対応します。レンジ・リスト・パーティション化は、時間隔リスト・パーティション化を使用して実装できますが、リスト・レンジ・パーティション化は時間隔パーティション化をサポートしていません。

例3-13に、様々な通貨単位での寄付額を格納するdonations表を示します。donationsは、金額に応じて、低、中、高にカテゴリ分けされています。通貨によってレンジは異なります。

例3-13 コンポジット・リスト - レンジ・パーティション化による表の作成

```
CREATE TABLE donations
( id             NUMBER
, name           VARCHAR2(60)
, beneficiary    VARCHAR2(80)
, payment_method VARCHAR2(30)
, currency       VARCHAR2(3)
, amount         NUMBER
) PARTITION BY LIST (currency)
SUBPARTITION BY RANGE (amount)
( PARTITION p_eur VALUES ('EUR')
  ( SUBPARTITION p_eur_small VALUES LESS THAN (8)
  , SUBPARTITION p_eur_medium VALUES LESS THAN (80)
  , SUBPARTITION p_eur_high VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_gbp VALUES ('GBP')
  ( SUBPARTITION p_gbp_small VALUES LESS THAN (5)
  , SUBPARTITION p_gbp_medium VALUES LESS THAN (50)
  , SUBPARTITION p_gbp_high VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_aud_nzd_chf VALUES ('AUD','NZD','CHF')
  ( SUBPARTITION p_aud_nzd_chf_small VALUES LESS THAN (12)
  , SUBPARTITION p_aud_nzd_chf_medium VALUES LESS THAN (120)
  , SUBPARTITION p_aud_nzd_chf_high VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_jpy VALUES ('JPY')
  ( SUBPARTITION p_jpy_small VALUES LESS THAN (1200)
  , SUBPARTITION p_jpy_medium VALUES LESS THAN (12000)
  , SUBPARTITION p_jpy_high VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_inr VALUES ('INR')
  ( SUBPARTITION p_inr_small VALUES LESS THAN (400)
  , SUBPARTITION p_inr_medium VALUES LESS THAN (4000)
  , SUBPARTITION p_inr_high VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_zar VALUES ('ZAR')
  ( SUBPARTITION p_zar_small VALUES LESS THAN (70)
  , SUBPARTITION p_zar_medium VALUES LESS THAN (700)
  , SUBPARTITION p_zar_high VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_default VALUES (DEFAULT)
  ( SUBPARTITION p_default_small VALUES LESS THAN (10)
  , SUBPARTITION p_default_medium VALUES LESS THAN (100)
  , SUBPARTITION p_default_high VALUES LESS THAN (MAXVALUE)
  )
) ENABLE ROW MOVEMENT;
```

3.5.5 時間隔パーティション化を使用する場合

時間隔パーティション化は、レンジ・パーティション化されており、新しいパーティションで固定間隔を使用するほぼすべての表で使用できます。

時間隔パーティションは、そのパーティションのデータが挿入されたときに自動的に作成されます。このときまで、時間隔パーティションは存在しますが、そのパーティションのセグメントは作成されていません。

時間隔パーティション化の利点は、レンジ・パーティションを明示的に作成する必要がないことです。様々な間隔のレンジ・パーティションを作成しない場合、またはレンジ・パーティションを作成するときに常に特定のパーティション属性を設定する場合は、時

間隔パーティション化の使用を検討する必要があります。時間隔の定義では表領域のリストを指定できます。時間隔パーティションは、データベースによってラウンドロビン法で指定リストの表領域に作成されます。

アプリケーションをアップグレードし、レンジ・パーティション化または、様々なコンポジット・レンジ・パーティション化を使用する場合は、既存の表定義を簡単に変更して、時間隔パーティション化を使用することができます。時間隔パーティション表へのパーティションの手動追加はできません。新しいパーティションの作成を自動化している場合は、将来、レンジ・パーティションの明示的な作成が行われないようにアプリケーション・コードを変更する必要があります。

次のSQL文では、sales表でのレンジ・パーティション化から月次時間隔パーティション化の使用への変更が開始されます。

```
ALTER TABLE sales SET INTERVAL (NUMTOYMINTERVAL(1,'MONTH'));
```

参照パーティション表では時間隔パーティションを使用することはできません。

シリアル化可能トランザクションは、時間隔パーティションでは動作しません。まだセグメントのない時間隔パーティション表のパーティションにデータを挿入すると、エラーが発生します。

3.5.6 参照パーティション化を使用する場合

参照パーティション化は、特定の状況で役に立ちます。

参照パーティション化は次のシナリオで役立ちます。

- マスター表と子表の両方でパーティション・プルーニングを利用できるように、マスター表の列を子表に非正規化した場合(または、これから非正規化する場合)。

  たとえば、order_dateはorders表には格納されますが、注文ごとに1つ以上の品目を格納するorder_items表には格納されません。注文データの履歴分析のパフォーマンスを高めるには、従来であれば、order_items表にorder_date列を複製して、order_items表でパーティション・プルーニングを使用できるようにするはずです。

  このようなシナリオでは、order_dateを複製せずにすむように、参照パーティション化を検討する必要があります。両方の表を結合してorder_dateに対する条件を使用する問合せは、両方の表のパーティション・プルーニングを自動的に利用します。

- 2つの大きな表を頻繁に結合するときに、2つの表が結合キーでパーティション化されていないが、パーティション・ワイズ結合を利用する場合。

  参照パーティション化により、フル・パーティション・ワイズ結合が暗黙に使用可能になります。

- 複数の表のデータが1つのライフサイクルに関連している場合。参照パーティション化を使用すると、管理上で大きなメリットが得られます。

  マスター表でパーティション管理操作を行うと、子表でも自動的に行われます。たとえば、マスター表にパーティションを追加すると、その追加がすべての子表に自動的に伝播されます。

  参照パーティション化を使用するには、マスター表と参照表の間の外部キー関係を有効にして施行する必要があります。参照パーティション表をカスケードできます。

  主キー - 外部キーの関係を常に有効にする必要があり、無効にできません。また、関係は遅延として宣言できません。有効な主キーと外部の関係が子表のデータの配置を決定するために必要なため、これらは必須要件です。

3.5.7 仮想列でパーティション化する場合

仮想列でのパーティション化により、導出列でより柔軟にパーティション化できます。

仮想列でのパーティション化では、式についてパーティション化できます。つまり、他の列のデータを使用し、それらの列で計算を

実行できます。PL/SQL関数コールは、パーティション化キーとして使用される仮想列の定義ではサポートされません。

仮想列のパーティション化では、すべてのパーティション化方法に加えて、パフォーマンスや管理のための機能もサポートされます。表のアクセスに頻繁に使用される述語が、列から直接取得するのではなく導出する場合には、パーティション・プルーニングの効果を得るために仮想列の使用を検討してください。従来、パーティション・プルーニングの効果を上げるためには、正しい値を取得して計算するために別の列を追加し、問合せで適切な検索を行えるように、列が常に正しい値を含むようにする必要がありました。

例3-14にcar_rentals表を示します。顧客の確認番号には、レンタカーを借りた場所として2文字の国名が含まれます。レンタカーの分析では通常は地域パターンが評価されるため、国別にパーティション化することに意味があります。

この例では、列countryは、確認番号から導出される仮想列として定義されています。仮想列には記憶域は必要ありません。この例が示すように、行の移動は仮想列でサポートされています。仮想列が別のパーティションにある異なる値と評価されると、その行はデータベースによって別のパーティションに移動されます。

例3-14 仮想列でのパーティション化による表の作成

```
CREATE TABLE car_rentals
( id                   NUMBER NOT NULL
 , customer_id          NUMBER NOT NULL
 , confirmation_number VARCHAR2(12) NOT NULL
 , car_id               NUMBER
 , car_type             VARCHAR2(10)
 , requested_car_type  VARCHAR2(10) NOT NULL
 , reservation_date    DATE NOT NULL
 , start_date           DATE NOT NULL
 , end_date             DATE
 , country as (substr(confirmation_number,9,2))
) PARTITION BY LIST (country)
SUBPARTITION BY HASH (customer_id)
SUBPARTITIONS 16
( PARTITION north_america VALUES ('US','CA','MX')
 , PARTITION south_america VALUES ('BR','AR','PE')
 , PARTITION europe VALUES ('GB','DE','NL','BE','FR','ES','IT','CH')
 , PARTITION apac VALUES ('NZ','AU','IN','CN')
) ENABLE ROW MOVEMENT;
```

3.5.8 読取り専用表領域を使用する場合の考慮事項

読取り専用表を使用する際の考慮事項を確認してください。

参照整合性制約が親表と子表の間に定義されているとき、索引は外部キーに定義され、その索引が格納される表領域は読取り専用になり、制約の整合性チェックが、読取り一貫性バッファ・アクセスを介してではなく、SQLで実装されます。

この実装が意味するのは、子がパーティション化されると、一部の子パーティションの索引のみが読取り専用表領域に含まれることと、非読取り専用子セグメントの1つに対して挿入が行われると、TMエンキューが子表に対してSXモードで獲得されることです。

SXモードはSリクエストとの互換性がありません。したがって、親に対する挿入の試行は、子に対するS TMエンキューを獲得しようとするため妨げられます。

4 パーティションの管理

パーティションの管理は、パーティション表および索引を使用する際の重要なタスクです。

この章では、パーティション表や索引の作成およびメンテナンスの様々な局面について説明します。

この章の構成は、次のとおりです。

- 表および索引を作成する場合のパーティション化の指定
- 表を作成する場合のコンポジット・パーティション化の指定
- パーティションでサポートされているメンテナンス操作
- パーティション表および索引のメンテナンス操作
- パーティション表の削除について
- パーティション表への非パーティション表の変更
- ハイブリッド・パーティション表の管理
- パーティション表および索引の情報の表示

ノート:

パーティション表や索引を作成する前、または任意のパーティション表でメンテナンス操作を実行する前に、「パーティション化の概念」の情報を確認することをお薦めします。

関連項目:

パーティション化の一般的な制限、パーティション表や索引を作成および変更するためのパーティション化句の正確な構文、その使用に関する制限、および表の作成や変更に必要な特定の権限の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

4.1 表および索引を作成する場合のパーティション化の指定

パーティション表または索引の作成は、非パーティション表または索引の作成と非常に似ています。

パーティション表または索引を作成する場合は、CREATE TABLE文にパーティション化句を含めます。文に含めるパーティション化句および副次句は、実行するパーティション化のタイプによって異なります。

パーティション化は、LONGまたはLONG RAW列を含む表を除き、通常の(ヒープ構成)表および索引構成表の両方で実行できます。パーティション表には、非パーティション・グローバル索引、レンジまたはハッシュ・パーティション・グローバル索引、およびローカル索引を作成できます。

パーティション表を作成(または変更)する際には、行移動句(ENABLE ROW MOVEMENTまたはDISABLE ROW MOVEMENTのいずれか)を指定できます。この句は、キーが更新された場合に、新しいパーティションへの行の移行を有効化または無効化します。デフォルトはDISABLE ROW MOVEMENTです。

単一レベル・パーティション表の1024K-1パーティションまたはコンポジット・パーティション表のサブパーティションの合計まで指定できます。

自動リスト・コンポジット・パーティション表および時間隔サブパーティションを作成することで、領域を節約できます。これは、これらの方法では、データが存在する場合のみサブパーティションが作成されるためです。要求に応じて新しいパーティションを作成する場合にサブパーティション・セグメントの作成を延期することにより、最初の一致行が挿入されるときのみサブパーティション・セグメントが作成されるようになります。

次のトピックでは、様々なタイプのパーティション表および索引に関するパーティション作成の詳細および例を示します。

- レンジ・パーティション表およびグローバル索引の作成について
- レンジ時間隔パーティション表の作成
- ハッシュ・パーティション表およびグローバル索引の作成について
- リスト・パーティション表の作成について
- 参照パーティション表の作成
- 時間隔 - 参照パーティション表の作成
- インメモリー列ストアとパーティション化を使用した表の作成
- 読取り専用パーティションまたはサブパーティションを含む表の作成
- パーティション化された外部表の作成
- キー列に対するパーティション化の指定
- 仮想列ベースのパーティション化の使用
- パーティション表での表圧縮の使用
- パーティション索引でのキー圧縮の使用
- セグメントによるパーティション化の指定
- 索引構成表を作成する場合のパーティション化の指定
- 複数ブロック・サイズのパーティション化制限
- XMLTypeおよびオブジェクトのコレクションのパーティション化

関連項目:

- 表の管理については、『*Oracle Database*管理者ガイド』を参照してください。
- パーティション表や索引を作成および変更するためのパーティション化句の正確な構文、その使用に関する制限、および表の作成や変更に必要な特定の権限の詳細は、『*Oracle Database SQL*言語リファレンス』を参照してください。
- LOBまたはLOBとして保存されているその他のオブジェクトがある列を含むパーティション表の作成に固有の情報は、『*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
- オブジェクト・タイプ、ネストした表またはVARRAYを含む表の作成に固有の情報は、『*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。

4.1.1 レンジ・パーティション表およびグローバル索引の作成について

CREATE TABLE文のPARTITION BY RANGE句では、表または索引をレンジ・パーティション化することを指定します。

PARTITION句では個々のパーティション・レンジを特定し、PARTITION句のオプションの副次句では、パーティション・セグメントに固有の物理属性およびその他の属性を指定できます。パーティション・レベルで上書きされない場合、パーティションは基礎となる表の属性を継承します。

次の内容について説明します。

- レンジ・パーティション表の作成
- より複雑なレンジ・パーティション表の作成
- レンジ・パーティション・グローバル索引の作成

4.1.1.1 レンジ・パーティション表の作成

レンジ・パーティション表を作成するには、CREATE TABLE文のPARTITION BY RANGE句を使用します。

例4-1では、1つのパーティションが各四半期の売上に対応する、4つのパーティションの表が作成されます。time_idは**パーティション化列**で、その値は特定の行の**パーティション化キー**を含みます。VALUES LESS THAN句によって**パーティション・バウンド**が決まります。この句で指定された値の順序リストと比較して、パーティション化キーの値がそれより小さい行がそのパーティションに保存されます。各パーティションには名前が付けられ(sales_q1_2006、sales_q2_2006、sales_q3_2006、sales_q4_2006)、各パーティションは別の表領域(tsa、tsb、tsc、tsd)に格納されます。time_id=17-MAR-2006を含む行は、パーティションsales_q1_2006に格納されます。

Live SQL:

関連する例を Oracle Live SQL の *Oracle Live SQL: レンジ・パーティション表の作成*で参照して実行してください。

例4-1 レンジ・パーティション表の作成

```
CREATE TABLE sales
  ( prod_id       NUMBER(6)
  , cust_id       NUMBER
  , time_id       DATE
  , channel_id    CHAR(1)
  , promo_id      NUMBER(6)
```

```
 , quantity_sold NUMBER(3)
 , amount_sold   NUMBER(10,2)
 )
PARTITION BY RANGE (time_id)
( PARTITION sales_q1_2006 VALUES LESS THAN (TO_DATE('01-APR-2006','dd-MON-yyyy'))
   TABLESPACE tsa
, PARTITION sales_q2_2006 VALUES LESS THAN (TO_DATE('01-JUL-2006','dd-MON-yyyy'))
   TABLESPACE tsb
, PARTITION sales_q3_2006 VALUES LESS THAN (TO_DATE('01-OCT-2006','dd-MON-yyyy'))
   TABLESPACE tsc
, PARTITION sales_q4_2006 VALUES LESS THAN (TO_DATE('01-JAN-2007','dd-MON-yyyy'))
   TABLESPACE tsd
);
```

4.1.1.2 より複雑なレンジ・パーティション表の作成

属性および記憶域パラメータにより、より複雑なレンジ・パーティション表を作成できます。

例4-2では、記憶域パラメータおよびLOGGING属性が表レベルで指定されています。これらは、表自体の表領域レベルから継承された対応するデフォルトを置き換え、レンジ・パーティションに継承されます。ただし、第1四半期には取引が少なかったため、パーティションsales_q1_2006の記憶域属性は小さく設定されています。ENABLE ROW MOVEMENT句は、キー値に、行を別のパーティションに配置するような更新が行われた場合に、新しいパーティションへの自動的な行の移行を可能にするために指定されています。

例4-2 LOGGINGおよびENABLE ROW MOVEMENTを使用したレンジ・パーティション表の作成

```
CREATE TABLE sales
  ( prod_id       NUMBER(6)
  , cust_id       NUMBER
  , time_id       DATE
  , channel_id    CHAR(1)
  , promo_id      NUMBER(6)
  , quantity_sold NUMBER(3)
  , amount_sold   NUMBER(10,2)
  )
 STORAGE (INITIAL 100K NEXT 50K) LOGGING
 PARTITION BY RANGE (time_id)
 ( PARTITION sales_q1_2006 VALUES LESS THAN (TO_DATE('01-APR-2006','dd-MON-yyyy'))
    TABLESPACE tsa STORAGE (INITIAL 20K NEXT 10K)
 , PARTITION sales_q2_2006 VALUES LESS THAN (TO_DATE('01-JUL-2006','dd-MON-yyyy'))
    TABLESPACE tsb
 , PARTITION sales_q3_2006 VALUES LESS THAN (TO_DATE('01-OCT-2006','dd-MON-yyyy'))
    TABLESPACE tsc
 , PARTITION sales_q4_2006 VALUES LESS THAN (TO_DATE('01-JAN-2007','dd-MON-yyyy'))
    TABLESPACE tsd
 )
 ENABLE ROW MOVEMENT;
```

4.1.1.3 レンジ・パーティション・グローバル索引の作成

レンジ・パーティション・グローバル索引を作成する際のルールは、レンジ・パーティション表を作成する際のルールに似ています。

例4-3では、前の例で作成された表のsale_monthにレンジ・パーティション・グローバル索引が作成されます。各索引パーティションには名前が付けられていますが、索引のデフォルトの表領域に保存されています。

例4-3 レンジ・パーティション・グローバル索引表の作成

```
CREATE INDEX amount_sold_ix ON sales(amount_sold)
```

```
  GLOBAL PARTITION BY RANGE(sale_month)
     ( PARTITION p_100 VALUES LESS THAN (100)
     , PARTITION p_1000 VALUES LESS THAN (1000)
     , PARTITION p_10000 VALUES LESS THAN (10000)
     , PARTITION p_100000 VALUES LESS THAN (100000)
     , PARTITION p_1000000 VALUES LESS THAN (1000000)
     , PARTITION p_greater_than_1000000 VALUES LESS THAN (maxvalue)
     );
```

ノート:

エンタープライズで異なる文字セットを使用するデータベースを使用している場合、または今後使用する予定の場合は、文字のソート順序はすべての文字セットで同一ではないため、文字の列をパーティション化する際には注意してください。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

4.1.2 レンジ時間隔パーティション表の作成

CREATE TABLE文のINTERVAL句で、表の時間隔パーティション化が設定されます。

PARTITIONを使用して少なくとも1つのレンジ・パーティションを指定してください。レンジ・パーティション化キーの値は、遷移ポイントと呼ばれるレンジ・パーティションの値の上限を決定します。遷移ポイントを超えるデータのために、データベースによって時間隔パーティションが自動的に作成されます。各時間隔パーティションの下限は、前の範囲つまり時間隔の上限であり、そのパーティションには含まれません。

たとえば、間隔が月単位で遷移点が2010年1月1日の時間隔パーティション表を作成した場合、2010年1月の間隔の下限は2010年1月1日です。2007年7月の間隔の下限は、2010年6月のパーティションがすでに作成されているかどうかに関係なく2010年7月1日です。ただし、パーティションの上限または下限が、格納のために設定されたレンジの外部になるような日付を使用すると、エラーが発生します。たとえば、TO_DATE('9999-12-01', 'YYYY-MM-DD')では上限が10000-01-01になりますが、10000が法的なレンジの外になる場合は、格納できなくなります。

オプションのSTORE IN句を使用して1つ以上の表領域を指定できます。この表領域には、データベースが、後から作成される時間隔パーティションのために、ラウンドロビン・アルゴリズムを使用して時間隔パーティション・データを格納できます。

時間隔パーティション化では、1つのパーティション化キー列のみを指定でき、そのデータ型は制限されています。

次の例では、時間隔の幅が異なる4つの表を指定しています。また、2010年1月1日の遷移ポイントを超えると、パーティションの時間隔が1か月で作成されることも指定しています。パーティションp3の上位バウンドは、遷移点を表しています。p3およびそれより下位のすべてのパーティション(この例ではp0、p1およびp2)はレンジ・セクションにありますが、それをより上位のすべてのパーティションは時間隔セクションに分類されます。

```
CREATE TABLE interval_sales
    ( prod_id        NUMBER(6)
    , cust_id        NUMBER
    , time_id        DATE
    , channel_id     CHAR(1)
    , promo_id       NUMBER(6)
    , quantity_sold  NUMBER(3)
    , amount_sold    NUMBER(10,2)
    )
  PARTITION BY RANGE (time_id)
  INTERVAL(NUMTOYMINTERVAL(1, 'MONTH'))
    ( PARTITION p0 VALUES LESS THAN (TO_DATE('1-1-2008', 'DD-MM-YYYY')),
```

```
      PARTITION p1 VALUES LESS THAN (TO_DATE('1-1-2009', 'DD-MM-YYYY')),
      PARTITION p2 VALUES LESS THAN (TO_DATE('1-7-2009', 'DD-MM-YYYY')),
      PARTITION p3 VALUES LESS THAN (TO_DATE('1-1-2010', 'DD-MM-YYYY')) );
```

関連項目:

パーティション化キーの制限、パーティション表や索引を作成および変更するためのパーティション化句の正確な構文、その使用に関する制限、および表の作成や変更に必要な特定の権限の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

4.1.3 ハッシュ・パーティション表およびグローバル索引の作成について

CREATE TABLE文のPARTITION BY HASH句では、表のハッシュ・パーティション化を指定しています。

PARTITIONS句は、作成するパーティション数の指定や、オプションで、パーティションを格納する表領域の指定に使用できます。また、PARTITION句を使用して、個々のパーティションおよびその表領域に名前を付けることも可能です。

ハッシュ・パーティションに指定できる属性はTABLESPACEのみです。表のすべてのハッシュ・パーティションで、表レベルから継承される同一のセグメント属性(TABLESPACEを除く)を共有する必要があります。

次の内容について説明します。

- ハッシュ・パーティション表の作成
- ハッシュ・パーティション・グローバル索引の作成

4.1.3.1 ハッシュ・パーティション表の作成

このトピックの例は、ハッシュ・パーティション表の作成方法を示します。

パーティション化列はidで、4つのパーティションが作成されてシステム生成の名前が割り当てられます。4つのパーティションは、名前の付けられた4つの表領域(gear1、gear2、gear3、gear4)に配置されます。

```
CREATE TABLE scubagear
     (id NUMBER,
      name VARCHAR2 (60))
   PARTITION BY HASH (id)
   PARTITIONS 4
   STORE IN (gear1, gear2, gear3, gear4);
```

次の例では、ハッシュ・パーティション表を作成する場合のパーティション数が指定されていますが、システム生成の名前が割り当てられ、表のデフォルトの表領域に格納されます。

```
CREATE TABLE departments_hash (department_id NUMBER(4) NOT NULL,
              department_name VARCHAR2(30))
     PARTITION BY HASH(department_id) PARTITIONS 16;
```

次の例では、個々のパーティション名、およびそれらのパーティションを配置する表領域が指定されています。各ハッシュ・パーティション(セグメント)の初期エクステント・サイズも表レベルで明示的に指定されており、すべてのパーティションでこの属性が継承されます。

```
CREATE TABLE departments_hash (department_id NUMBER(4) NOT NULL,
              department_name VARCHAR2(30))
     STORAGE (INITIAL 10K)
```

```
    PARTITION BY HASH(department_id)
      (PARTITION p1 TABLESPACE ts1, PARTITION p2 TABLESPACE ts2,
       PARTITION p3 TABLESPACE ts1, PARTITION p4 TABLESPACE ts3);
```

この表にローカル索引を作成すると、データベースにより、基礎となる表と同一レベル・パーティション化されるように索引が作成されます。また、基礎となる表でメンテナンス操作が実行された場合には、データベースにより自動的に索引がメンテナンスされます。次に、表にローカル索引を作成する例を示します。

```
CREATE INDEX loc_dept_ix ON departments_hash(department_id) LOCAL;
```

オプションで、ハッシュ・パーティションおよびローカル索引パーティションが格納される表領域に名前を付けることができますが、そうしない場合、データベースでは対応する基礎となるパーティションの名前が索引パーティション名として使用され、索引パーティションは表パーティションと同じ表領域に格納されます。

関連項目:

キー列のパーティション化の詳細は、「キー列に対するパーティション化の指定」を参照してください

4.1.3.2 ハッシュ・パーティション・グローバル索引の作成

ハッシュ・パーティション・グローバル索引を使用すると、マルチ・ユーザーのOLTP環境において、索引内の少数のリーフ・ブロックの競合率が高い索引のパフォーマンスが向上します。

ハッシュ・パーティション・グローバル索引でも、増加し続ける列値における索引の誤差の影響を限定できます。索引パーティション化キーに等価およびIN述語が含まれる問合せでは、ハッシュ・パーティション・グローバル索引を効率的に使用できます。

ハッシュ・パーティション・グローバル索引を作成する構文は、ハッシュ・パーティション表に使用する構文に似ています。たとえば、例4-4の文では、ハッシュ・パーティション・グローバル索引を作成します。

例4-4 ハッシュ・パーティション・グローバル索引の作成

```
CREATE INDEX hgidx ON tab (c1,c2,c3) GLOBAL
     PARTITION BY HASH (c1,c2)
     (PARTITION p1  TABLESPACE tbs_1,
      PARTITION p2  TABLESPACE tbs_2,
      PARTITION p3  TABLESPACE tbs_3,
      PARTITION p4  TABLESPACE tbs_4);
```

4.1.4 リスト・パーティション表の作成について

リスト・パーティションを作成するセマンティックは、レンジ・パーティションを作成するためのセマンティックと非常に似ています。

ただし、リスト・パーティションを作成するには、CREATE TABLE文にPARTITION BY LIST句を指定し、PARTITION句で、パーティションに含まれる行を修飾するパーティション化列の離散値であるリテラル値のリストを指定します。リスト・パーティション化の場合、パーティション化キーは表の1つ以上の列名にすることができます。

リスト・パーティションでのみ使用可能なキーワードDEFAULTを使用して、パーティションの値リストを説明できます。これにより、その他のパーティションにマッピングされていない行を格納するパーティションが特定されます。

レンジ・パーティションと同様、PARTITION句のオプションの副次句で、パーティション・セグメントに固有の物理属性およびその他の属性を指定できます。パーティション・レベルで上書きされない場合、パーティションは親表の属性を継承します。

次の内容について説明します。

- リスト・パーティション表の作成
- デフォルトのパーティションを使用したリスト・パーティション表の作成
- 自動リスト・パーティション表の作成
- 複数列リスト・パーティション表の作成

4.1.4.1 リスト・パーティション表の作成

このトピックの例は、リスト・パーティション表の作成方法を示します。

例4-5では、アメリカの州のグループを含む地域でパーティション化された表q1_sales_by_regionが作成されます。行のパーティション化列の値が、パーティションを説明する値リストの値に一致するかどうかを確認して、行がパーティションにマッピングされます。たとえば、次のリストは、いくつかのサンプル行の表への挿入を示します。

- (10, 'accounting', 100, 'WA')はパーティションq1_northwestにマッピングされます。
- (20, 'R&D', 150, 'OR')はパーティションq1_northwestにマッピングされます。
- (30, 'sales', 100, 'FL')はパーティションq1_southeastにマッピングされます。
- (40, 'HR', 10, 'TX')はパーティションq1_southwestにマッピングされます。
- (50, 'systems engineering', 10, 'CA')は表内のいずれのパーティションにもマッピングされず、エラーが発生します。

Live SQL:

関連する例をOracle Live SQLの*Oracle Live SQL: リスト・パーティション表の作成*で参照して実行してください。

例4-5 リスト・パーティション表の作成

```
CREATE TABLE q1_sales_by_region
      (deptno number,
       deptname varchar2(20),
       quarterly_sales number(10, 2),
       state varchar2(2))
   PARTITION BY LIST (state)
      (PARTITION q1_northwest VALUES ('OR', 'WA'),
       PARTITION q1_southwest VALUES ('AZ', 'UT', 'NM'),
       PARTITION q1_northeast VALUES  ('NY', 'VM', 'NJ'),
       PARTITION q1_southeast VALUES ('FL', 'GA'),
       PARTITION q1_northcentral VALUES ('SD', 'WI'),
       PARTITION q1_southcentral VALUES ('OK', 'TX'));
```

4.1.4.2 デフォルトのパーティションを使用したリスト・パーティション表の作成

レンジ・パーティションとは異なり、リスト・パーティションには、パーティション間に明白な順序はありません。

また、その他のパーティションにマッピングされていない行がマッピングされる、**デフォルトのパーティション**を指定できます。前述の例でデフォルトのパーティションを指定した場合、州CAはそのパーティションにマッピングされます。

例4-6では、表sales_by_regionを作成し、リスト・メソッドを使用してパーティション化します。最初の2つのPARTITION句には、表レベルのデフォルトを上書きする物理属性が指定されています。残りのPARTITION句には属性は指定されておらず、パーティ

ションは表レベルのデフォルトから物理属性を継承します。デフォルトのパーティションも指定されています。

例4-6 デフォルトのパーティションを使用したリスト・パーティション表の作成

```
CREATE TABLE sales_by_region (item# INTEGER, qty INTEGER,
             store_name VARCHAR(30), state_code VARCHAR(2),
             sale_date DATE)
     STORAGE(INITIAL 10K NEXT 20K) TABLESPACE tbs5
     PARTITION BY LIST (state_code)
     (
     PARTITION region_east
        VALUES ('MA','NY','CT','NH','ME','MD','VA','PA','NJ')
        STORAGE (INITIAL 8M)
        TABLESPACE tbs8,
     PARTITION region_west
        VALUES ('CA','AZ','NM','OR','WA','UT','NV','CO')
        NOLOGGING,
     PARTITION region_south
        VALUES ('TX','KY','TN','LA','MS','AR','AL','GA'),
     PARTITION region_central
        VALUES ('OH','ND','SD','MO','IL','MI','IA'),
     PARTITION region_null
        VALUES (NULL),
     PARTITION region_unknown
        VALUES (DEFAULT)
     );
```

4.1.4.3 自動リスト・パーティション表の作成

自動リスト・パーティション化メソッドを使用すると、要求に応じてリスト・パーティションを作成できます。

自動リスト・パーティション表は、このパーティション表がより管理しやすいという点を除いて、通常のリスト・パーティション表に似ています。自動リスト・パーティション表は、既知のパーティション化キー値のみを使用して作成できます。データはこの表にロードされるため、ロードされたパーティション化キー値が既存のパーティションのいずれにも対応しない場合は、データベースで新しいパーティションが自動的に作成されます。パーティションは要求に応じて自動的に作成されるため、自動リスト・パーティション化メソッドは、既存の間隔パーティション化メソッドと概念的に似ています。

値が頻繁に変わるデータ型での自動リスト・パーティション化は、データを調整できる場合を除いて、このメソッドには不向きです。たとえば、日付値を含むSALES_DATEフィールドは、フォーマットが削除されない場合、毎秒増加します。各SALES_DATE値(05-22-2016 08:00:00、05-22-2016 08:00:01など)は、独自のパーティションを生成します。膨大な数のパーティションの作成を避けるには、入力されるデータに注意し、適宜調整する必要があります。たとえば、必要なパーティションの数に応じて、SALES_DATEの日付値を日または他の特定の時間に切り捨てることができます。

CREATEおよびALTER TABLE SQL文は、AUTOMATICまたはMANUALリスト・パーティション化を指定する追加の句を使用して更新されます。自動リスト・パーティション表の作成時には、少なくとも1つのパーティションが必要です。新規および不明のパーティション・キー値に対して新しいパーティションが自動的に作成されるため、自動リスト・パーティションはDEFAULTパーティションを持つことができません。

*_PART_TABLESビューのAUTOLIST列を確認して、表が自動リスト・パーティションであるかどうかを判断できます。

Live SQL:

Oracle Live SQL の *Oracle Live SQL: 自動リスト・パーティション表の作成*で関連する例を参照して実

行してください。

例4-7は、sales_stateフィールドで自動リスト・パーティション化のAUTOMATICキーワードを使用するCREATE TABLE文の例です。CREATE TABLE SQL文により、必要に応じて少なくとも1つのパーティションが作成されます。追加の行が挿入されると、新しいsales_state値が追加されるとパーティションの数が増加します。

例4-7 自動リスト・パーティション表の作成

```
CREATE TABLE sales_auto_list
(
   salesman_id   NUMBER(5)    NOT NULL,
   salesman_name VARCHAR2(30),
   sales_state   VARCHAR2(20) NOT NULL,
   sales_amount  NUMBER(10),
   sales_date    DATE         NOT NULL
)
  PARTITION BY LIST (sales_state) AUTOMATIC
 (PARTITION P_CAL VALUES ('CALIFORNIA')
);

SELECT TABLE_NAME, PARTITIONING_TYPE, AUTOLIST, PARTITION_COUNT FROM USER_PART_TABLES WHERE TABLE_NAME
='SALES_AUTO_LIST';
TABLE_NAME        PARTITIONING_TYPE  AUTOLIST  PARTITION_COUNT
----------------  -----------------  --------  ---------------
SALES_AUTO_LIST   LIST               YES                     1

SELECT TABLE_NAME, PARTITION_NAME, HIGH_VALUE FROM USER_TAB_PARTITIONS WHERE TABLE_NAME
='SALES_AUTO_LIST';

TABLE_NAME        PARTITION_NAME    HIGH_VALUE
----------------  ---------------   -----------------
SALES_AUTO_LIST     P_CAL           'CALIFORNIA'

INSERT INTO SALES_AUTO_LIST VALUES(021, 'Mary Smith', 'FLORIDA', 41000, TO_DATE ('21-DEC-2018','DD-
MON-YYYY'));
1 row inserted.

INSERT INTO SALES_AUTO_LIST VALUES(032, 'Luis Vargas', 'MICHIGAN', 42000, TO_DATE ('31-DEC-2018','DD-
MON-YYYY'));
1 row inserted.

SELECT TABLE_NAME, PARTITIONING_TYPE, AUTOLIST, PARTITION_COUNT FROM USER_PART_TABLES WHERE TABLE_NAME
='SALES_AUTO_LIST';
TABLE_NAME        PARTITIONING_TYPE  AUTOLIST  PARTITION_COUNT
----------------  -----------------  --------  ---------------
SALES_AUTO_LIST   LIST               YES                     3

INSERT INTO SALES_AUTO_LIST VALUES(015, 'Simone Blair', 'CALIFORNIA', 45000, TO_DATE ('11-JAN-
2019','DD-MON-YYYY'));
1 row inserted.

INSERT INTO SALES_AUTO_LIST VALUES(015, 'Simone Blair', 'OREGON', 38000, TO_DATE ('18-JAN-2019','DD-
MON-YYYY'));
1 row inserted.

SELECT TABLE_NAME, PARTITIONING_TYPE, AUTOLIST,PARTITION_COUNT FROM USER_PART_TABLES WHERE TABLE_NAME
='SALES_AUTO_LIST';
```

```
TABLE_NAME        PARTITIONING_TYPE  AUTOLIST  PARTITION_COUNT
----------------  -----------------  --------  ---------------
SALES_AUTO_LIST   LIST               YES                     4


SELECT TABLE_NAME, PARTITION_NAME, HIGH_VALUE FROM USER_TAB_PARTITIONS WHERE TABLE_NAME
='SALES_AUTO_LIST';
TABLE_NAME        PARTITION_NAME    HIGH_VALUE
----------------  ---------------   -----------------
SALES_AUTO_LIST     P_CAL           'CALIFORNIA'
SALES_AUTO_LIST     SYS_P478        'FLORIDA'
SALES_AUTO_LIST     SYS_P479        'MICHIGAN'
SALES_AUTO_LIST     SYS_P480        'OREGON'
```

関連項目:

*_PART_TABLESビューの詳細は、Oracle Databaseリファレンスを参照してください

4.1.4.4 複数列リスト・パーティション表の作成

複数列リスト・パーティション化では、複数列のリスト値に基づいて表をパーティション化できます。

単一列リスト・パーティション化と同様に、各パーティションには、値のリストを含むセットを含めることができます。

複数列リスト・パーティション化は、表の複数列でPARTITION BY LIST句を使用する表でサポートされます。例:

```
PARTITION BY LIST (column1, column2)
```

複数列リスト・パーティション表に含めることができるのは、1つのDEFAULTパーティションのみです。

Live SQL:

Oracle Live SQL の *Oracle Live SQL: 複数列リスト・パーティション表の作成*で関連する例を参照して実行してください。

次に、stateおよびchannel列で複数列パーティション化を使用するCREATE TABLE文の例を示します。

例4-8 複数列リスト・パーティション表の作成

```
CREATE TABLE sales_by_region_and_channel
 (dept_number      NUMBER NOT NULL,
  dept_name        VARCHAR2(20),
  quarterly_sales  NUMBER(10,2),
  state            VARCHAR2(2),
  channel          VARCHAR2(1)
 )
 PARTITION BY LIST (state, channel)
 (
  PARTITION yearly_west_direct VALUES (('OR','D'),('UT','D'),('WA','D')),
  PARTITION yearly_west_indirect VALUES (('OR','I'),('UT','I'),('WA','I')),
  PARTITION yearly_south_direct VALUES (('AZ','D'),('TX','D'),('GA','D')),
  PARTITION yearly_south_indirect VALUES (('AZ','I'),('TX','I'),('GA','I')),
  PARTITION yearly_east_direct VALUES (('PA','D'), ('NC','D'), ('MA','D')),
```

```
  PARTITION yearly_east_indirect VALUES (('PA','I'), ('NC','I'), ('MA','I')),
  PARTITION yearly_north_direct VALUES (('MN','D'),('WI','D'),('MI','D')),
  PARTITION yearly_north_indirect VALUES (('MN','I'),('WI','I'),('MI','I')),
  PARTITION yearly_ny_direct VALUES ('NY','D'),
  PARTITION yearly_ny_indirect VALUES ('NY','I'),
  PARTITION yearly_ca_direct VALUES ('CA','D'),
  PARTITION yearly_ca_indirect VALUES ('CA','I'),
  PARTITION rest VALUES (DEFAULT)
 );

SELECT PARTITION_NAME, HIGH_VALUE FROM USER_TAB_PARTITIONS WHERE TABLE_NAME
='SALES_BY_REGION_AND_CHANNEL';
PARTITION_NAME            HIGH_VALUE
--------------------      ------------------------------------------
REST                      DEFAULT
YEARLY_CA_DIRECT          ( 'CA', 'D' )
YEARLY_CA_INDIRECT        ( 'CA', 'I' )
YEARLY_EAST_DIRECT        ( 'PA', 'D' ), ( 'NC', 'D' ), ( 'MA', 'D' )
YEARLY_EAST_INDIRECT      ( 'PA', 'I' ), ( 'NC', 'I' ), ( 'MA', 'I' )
YEARLY_NORTH_DIRECT       ( 'MN', 'D' ), ( 'WI', 'D' ), ( 'MI', 'D' )
YEARLY_NORTH_INDIRECT     ( 'MN', 'I' ), ( 'WI', 'I' ), ( 'MI', 'I' )
YEARLY_NY_DIRECT          ( 'NY', 'D' )
YEARLY_NY_INDIRECT        ( 'NY', 'I' )
YEARLY_SOUTH_DIRECT       ( 'AZ', 'D' ), ( 'TX', 'D' ), ( 'GA', 'D' )
YEARLY_SOUTH_INDIRECT     ( 'AZ', 'I' ), ( 'TX', 'I' ), ( 'GA', 'I' )
YEARLY_WEST_DIRECT        ( 'OR', 'D' ), ( 'UT', 'D' ), ( 'WA', 'D' )
YEARLY_WEST_INDIRECT      ( 'OR', 'I' ), ( 'UT', 'I' ), ( 'WA', 'I' )
13 rows selected.

INSERT INTO SALES_BY_REGION_AND_CHANNEL VALUES (005, 'AUTO DIRECT', 701000, 'OR', 'D' );
INSERT INTO SALES_BY_REGION_AND_CHANNEL VALUES (006, 'AUTO INDIRECT', 1201000, 'OR', 'I' );
INSERT INTO SALES_BY_REGION_AND_CHANNEL VALUES (005, 'AUTO DIRECT', 625000, 'WA', 'D' );
INSERT INTO SALES_BY_REGION_AND_CHANNEL VALUES (006, 'AUTO INDIRECT', 945000, 'WA', 'I' );
INSERT INTO SALES_BY_REGION_AND_CHANNEL VALUES (005, 'AUTO DIRECT', 595000, 'UT', 'D' );
INSERT INTO SALES_BY_REGION_AND_CHANNEL VALUES (006, 'AUTO INDIRECT', 825000, 'UT', 'I' );
INSERT INTO SALES_BY_REGION_AND_CHANNEL VALUES (003, 'AUTO DIRECT', 1950000, 'CA', 'D' );
INSERT INTO SALES_BY_REGION_AND_CHANNEL VALUES (004, 'AUTO INDIRECT', 5725000, 'CA', 'I' );
INSERT INTO SALES_BY_REGION_AND_CHANNEL VALUES (010, 'AUTO DIRECT', 925000, 'IL', 'D' );
INSERT INTO SALES_BY_REGION_AND_CHANNEL VALUES (010, 'AUTO INDIRECT', 3250000, 'IL', 'I' );

SELECT DEPT_NUMBER, DEPT_NAME, QUARTERLY_SALES, STATE, CHANNEL FROM SALES_BY_REGION_AND_CHANNEL
PARTITION(yearly_west_direct);
DEPT_NUMBER DEPT_NAME            QUARTERLY_SALES ST C
----------- -------------------- --------------- -- -
          5 AUTO DIRECT                  701000 OR D
          5 AUTO DIRECT                  625000 WA D
          5 AUTO DIRECT                  595000 UT D

SELECT DEPT_NUMBER, DEPT_NAME, QUARTERLY_SALES, STATE, CHANNEL FROM SALES_BY_REGION_AND_CHANNEL
PARTITION(yearly_west_indirect);
DEPT_NUMBER DEPT_NAME            QUARTERLY_SALES ST C
----------- -------------------- --------------- -- -
          6 AUTO INDIRECT               1201000 OR I
          6 AUTO INDIRECT                945000 WA I
          6 AUTO INDIRECT                825000 UT I

SELECT DEPT_NUMBER, DEPT_NAME, QUARTERLY_SALES, STATE, CHANNEL FROM SALES_BY_REGION_AND_CHANNEL
PARTITION(yearly_ca_direct);
DEPT_NUMBER DEPT_NAME            QUARTERLY_SALES ST C
----------- -------------------- --------------- -- -
```

```
         3 AUTO DIRECT                  1950000 CA D

SELECT DEPT_NUMBER, DEPT_NAME, QUARTERLY_SALES, STATE, CHANNEL FROM SALES_BY_REGION_AND_CHANNEL
PARTITION(yearly_ca_indirect);
DEPT_NUMBER DEPT_NAME            QUARTERLY_SALES ST C
----------- -------------------- --------------- -- -
         4 AUTO INDIRECT                5725000 CA I

SELECT DEPT_NUMBER, DEPT_NAME, QUARTERLY_SALES, STATE, CHANNEL FROM SALES_BY_REGION_AND_CHANNEL
PARTITION(rest);
DEPT_NUMBER DEPT_NAME            QUARTERLY_SALES ST C
----------- -------------------- --------------- -- -
        10 AUTO DIRECT                  925000 IL D
        10 AUTO INDIRECT               3250000 IL I
```

4.1.5 参照パーティション表の作成

参照パーティション表を作成するには、CREATE TABLE文にPARTITION BY REFERENCE句を指定します。

PARTITION BY REFERENCE句では参照制約の名前を指定します。この制約がパーティション化参照制約になり、これに基づいて表の参照パーティション化が行われます。参照制約を有効にして強制する必要があります。

その他のパーティション表と同じように、オブジェクト・レベルのデフォルト属性を指定できます。また、オプションで、パーティションごとにオブジェクト・レベルのデフォルトを上書きするパーティション記述子も指定できます。

例4-9は、order_dateでレンジ・パーティション化された親表ordersを作成します。参照パーティションの子表order_itemsは、Q1_2005、Q2_2005、Q3_2005およびQ4_2005の4つのパーティションとともに作成されています。各パーティションには、それぞれの親パーティションの注文に対応するorder_items行が含まれています。

パーティション記述子を指定する場合、指定するパーティション数は、参照表のパーティションまたはサブパーティションの数と正確に一致する必要があります。親表がコンポジット・パーティション表の場合、表は、親の各サブパーティションに対して1つのパーティションを含みます。それ以外の場合、親の各パーティションに対して1つのパーティションを含みます。

参照パーティション表のパーティションには、パーティション・バウンドを指定できません。

参照パーティション表のパーティションには名前を付けることができます。パーティションに明示的に名前が付けられていない場合は、親表の対応するパーティションの名前が既存の明示的に付けられた名前と競合しないかぎり、その名前が継承されます。この場合、パーティションにはシステム生成の名前が使用されます。

参照パーティション表のパーティションに明示的に表領域が指定されていない場合、参照パーティション表のパーティションは、親表の対応するパーティションと連結されます。

例4-9 参照パーティション表の作成

```
CREATE TABLE orders
    ( order_id           NUMBER(12),
      order_date         DATE,
      order_mode         VARCHAR2(8),
      customer_id        NUMBER(6),
      order_status       NUMBER(2),
      order_total        NUMBER(8,2),
      sales_rep_id       NUMBER(6),
      promotion_id       NUMBER(6),
      CONSTRAINT orders_pk PRIMARY KEY(order_id)
    )
  PARTITION BY RANGE(order_date)
    ( PARTITION Q1_2005 VALUES LESS THAN (TO_DATE('01-APR-2005','DD-MON-YYYY')),
      PARTITION Q2_2005 VALUES LESS THAN (TO_DATE('01-JUL-2005','DD-MON-YYYY')),
```

```
      PARTITION Q3_2005 VALUES LESS THAN (TO_DATE('01-OCT-2005','DD-MON-YYYY')),
      PARTITION Q4_2005 VALUES LESS THAN (TO_DATE('01-JAN-2006','DD-MON-YYYY'))
    );

CREATE TABLE order_items
    ( order_id           NUMBER(12) NOT NULL,
      line_item_id       NUMBER(3)  NOT NULL,
      product_id         NUMBER(6)  NOT NULL,
      unit_price         NUMBER(8,2),
      quantity           NUMBER(8),
      CONSTRAINT order_items_fk
      FOREIGN KEY(order_id) REFERENCES orders(order_id)
    )
    PARTITION BY REFERENCE(order_items_fk);
```

4.1.6 時間隔 - 参照パーティション表の作成

時間隔パーティション表を参照パーティション化の親表として使用できます。レコードを参照パーティション表に挿入すると、親表の時間隔パーティションに対応する参照パーティション表のパーティションが作成されます。

子表に時間隔パーティションを作成すると、パーティション名が関連付けられている親表フラグメントから継承されます。子表に表レベルのデフォルトの表領域がある場合、新しい時間隔パーティションの表領域として使用され、それ以外の場合は表領域が親表フラグメントから継承されます。

SQL ALTER TABLE SET INTERVAL文は参照パーティション表に使用できませんが、参照パーティションの子を持つ表で実行できます。特に、ALTER TABLE SET INTERVALは、ターゲット表から時間隔プロパティを削除して、時間隔 - 参照の子を通常の参照パーティション表に変換します。SQL ALTER TABLE SET STORE IN文も参照パーティション表に使用できませんが、参照パーティションの子を持つ表で実行できます。

時間隔パーティションのALTER TABLE SPLIT PARTITIONなど、親表の時間隔パーティションを従来のパーティションに変換する操作は、子表に対応する変換を作成し、必要に応じて子表にパーティションを作成します。

たとえば、次のSQL文では、親表に3つの時間隔パーティションを指定し、子表には何も指定していません。

```
CREATE TABLE par(pk INT CONSTRAINT par_pk PRIMARY KEY, i INT)
 PARTITION BY RANGE(i) INTERVAL (10)
 (PARTITION p1 VALUES LESS THAN (10));

CREATE TABLE chi(fk INT NOT NULL, i INT,
 CONSTRAINT chi_fk FOREIGN KEY(fk) REFERENCES par(pk))
 PARTITION BY REFERENCE(chi_fk);

INSERT INTO par VALUES(15, 15);
INSERT INTO par VALUES(25, 25);
INSERT INTO par VALUES(35, 35);
```

USER_TAB_PARTITIONSビューを使用して、パーティションの情報を表示できます。

```
SELECT table_name, partition_position, high_value, interval
   FROM USER_TAB_PARTITIONS WHERE table_name IN ('PAR', 'CHI')
   ORDER BY 1, 2;

TABLE_NAME       PARTITION_POSITION HIGH_VALUE INT
---------------- ------------------ ---------- ---
CHI                               1            NO
PAR                               1         10 NO
PAR                               2         20 YES
PAR                               3         30 YES
```

```
PAR                              4         40 YES
```

時間隔パーティションが親表で分割される場合、階層のすべての表に対して一部の時間隔パーティションが従来のパーティションに変換され、処理中に子表に従来のパーティションを作成します。例:

```
ALTER TABLE par SPLIT PARTITION FOR (25) AT (25)
   INTO (partition x, partition y);

SELECT table_name, partition_position, high_value, interval
   FROM USER_TAB_PARTITIONS WHERE table_name IN ('PAR', 'CHI')
   ORDER BY 1, 2;

TABLE_NAME       PARTITION_POSITION HIGH_VALUE INT
---------------- ------------------ ---------- ---
CHI                               1            NO
CHI                               2            NO
CHI                               3            NO
CHI                               4            NO
PAR                               1         10 NO
PAR                               2         20 NO
PAR                               3         25 NO
PAR                               4         30 NO
PAR                               5         40 YES
```

時間隔 - 参照機能は、データベース互換性レベル(Oracle Database COMPATIBLE初期化パラメータ設定)を12.0.0.0以上に設定する必要があります。

4.1.7 インメモリー列ストアとパーティション化を使用した表の作成

INMEMORY句によりインメモリー列ストアを使用してパーティション表を作成できます。

次の例では、各パーティションがINMEMORY句とCREATE TABLE SQL文のパーティショニング句を使用してインメモリー列ストアにロードされることを示しています。

```
CREATE TABLE list_customers
   ( customer_id             NUMBER(6)
   , cust_first_name         VARCHAR2(20)
   , cust_last_name          VARCHAR2(20)
   , cust_address            CUST_ADDRESS_TYP
   , nls_territory           VARCHAR2(30)
   , cust_email              VARCHAR2(40))
   PARTITION BY LIST (nls_territory) (
   PARTITION asia VALUES ('CHINA', 'THAILAND')
         INMEMORY MEMCOMPRESS FOR CAPACITY HIGH,
   PARTITION europe VALUES ('GERMANY', 'ITALY', 'SWITZERLAND')
         INMEMORY MEMCOMPRESS FOR CAPACITY LOW,
   PARTITION west VALUES ('AMERICA')
         INMEMORY MEMCOMPRESS FOR CAPACITY LOW,
   PARTITION east VALUES ('INDIA')
         INMEMORY MEMCOMPRESS FOR CAPACITY HIGH,
   PARTITION rest VALUES (DEFAULT);
```

関連項目:

- インメモリー列ストアの概要は、『Oracle Database In-Memoryガイド』を参照してください
- インメモリー列ストアおよびADOのサポートにおける、移入のためのオブジェクトの有効化の詳細は、『Oracle

Database In-Memoryガイド』を参照してください

- インメモリー列ストアに関連するSQL構文については、『Oracle Database SQL言語リファレンス』を参照してください

4.1.8 読取り専用パーティションまたはサブパーティションを含む表の作成

表、パーティションおよびサブパーティションを読取り専用ステータスに設定し、ユーザーまたはトリガーによる意図しないDML操作からデータを保護できます。

読取り専用に設定されているパーティションまたはサブパーティションのデータを更新しようとするとエラーが発生しますが、読取り/書込みに設定されているパーティションまたはサブパーティションのデータの更新は成功します。

CREATE TABLEおよびALTER TABLE SQL文は、パーティションおよびサブパーティションに読取り専用句を指定します。読取り専用句の値は、READ ONLYまたはREAD WRITEです。READ WRITEがデフォルト値です。読取り専用句がパーティションまたはサブパーティションに明示的に設定されていない場合は、読取り専用句の高レベルの設定がパーティションおよびサブパーティションに適用されます。

次に、読取り専用ステータスおよび読取り/書込みステータスのコンポジット・レンジ - リスト・パーティション表の作成例を示します。orders_read_write_onlyはREAD WRITEとして明示的に指定されているため、表のデフォルト属性は読取り/書込みです。パーティションorder_p1のデフォルト属性は読取り専用として指定されているため、サブパーティションord_p1_northwestおよびorder_p1_southwestは、パーティションorder_p1から読取り専用ステータスを継承します。サブパーティションord_p2_southwestおよびorder_p3_northwestは読取り専用として明示的に指定されているため、デフォルトの読取り/書込みステータスが上書きされます。

例4-10 読取り専用および読取り/書込みパーティションを含む表の作成

```
CREATE TABLE orders_read_write_only (
   order_id NUMBER (12),
   order_date DATE CONSTRAINT order_date_nn NOT NULL,
   state VARCHAR2(2)
   ) READ WRITE
     PARTITION BY RANGE (order_date)
     SUBPARTITION BY LIST (state)
     ( PARTITION order_p1 VALUES LESS THAN (TO_DATE ('01-DEC-2015','DD-MON-YYYY')) READ ONLY
     ( SUBPARTITION order_p1_northwest VALUES ('OR', 'WA'),
       SUBPARTITION order_p1_southwest VALUES ('AZ', 'UT', 'NM')
      ),
     PARTITION order_p2 VALUES LESS THAN (TO_DATE ('01-MAR-2016','DD-MON-YYYY'))
     ( SUBPARTITION order_p2_northwest VALUES ('OR', 'WA'),
       SUBPARTITION order_p2_southwest VALUES ('AZ', 'UT', 'NM') READ ONLY
      ),
     PARTITION order_p3 VALUES LESS THAN (TO_DATE ('01-JUL-2016','DD-MON-YYYY'))
     (
     SUBPARTITION order_p3_northwest VALUES ('OR', 'WA') READ ONLY,
     SUBPARTITION order_p3_southwest VALUES ('AZ', 'UT', 'NM')
     )
 );
```

読取り専用ステータスは、*_PART_TABLESビューのDEF_READ_ONLY列、*_TAB_PARTITIONSビューのREAD_ONLY列、および*_TAB_SUBPARTITIONSビューのREAD_ONLY列を使用して確認できます。物理セグメント(単一レベル・パーティション化のパーティションおよびコンポジット・パーティション化のサブパーティション)にのみ、ステータスがあることに注意してください。その他のすべてのレベルは論理的で、デフォルト・ステータスのみになります。

```
SQL> SELECT PARTITION_NAME, READ_ONLY FROM USER_TAB_PARTITIONS WHERE TABLE_NAME
='ORDERS_READ_WRITE_ONLY';
```

```
PARTITION_NAME                 READ
------------------------------ ----
ORDER_P1                       YES
ORDER_P2                       NONE
ORDER_P3                       NONE

SQL> SELECT PARTITION_NAME, SUBPARTITION_NAME, READ_ONLY FROM USER_TAB_SUBPARTITIONS WHERE TABLE_NAME
='ORDERS_READ_WRITE_ONLY';
PARTITION_NAME                 SUBPARTITION_NAME              REA
------------------------------ ------------------------------ ---
ORDER_P1                       ORDER_P1_NORTHWEST             YES
ORDER_P1                       ORDER_P1_SOUTHWEST             YES
ORDER_P2                       ORDER_P2_NORTHWEST             NO
ORDER_P2                       ORDER_P2_SOUTHWEST             YES
ORDER_P3                       ORDER_P3_NORTHWEST             YES
ORDER_P3                       ORDER_P3_SOUTHWEST             NO
```

関連項目:

*_PART_TABLES、*_TAB_PARTITIONSおよび*_TAB_SUBPARTITIONSビューの詳細は、Oracle Databaseリファレンスを参照してください

4.1.9 パーティション化された外部表の作成

外部表のためにパーティションを作成できます。

構成外部句は、外部表の仕様およびアクセス・パラメータに従い、表を外部表として識別します。デフォルト・ディレクトリなどのパラメータはパーティションまたはサブパーティション・レベルでオーバーライドできますが、外部表タイプおよびそのアクセス・パラメータは、表レベル属性であり、すべてのパーティションまたはサブパーティションに適用できます。

例4-11で作成した表には、様々な場所からアクセスされる外部データのための3つのパーティションがあります。パーティションp1は、表のデフォルト・ディレクトリにあり、Californiaの顧客データを格納します。パーティションp2は、Washingtonのデータを格納するファイルを指し示します。パーティションp3は、ファイル記述子がなく、空となります。

例4-11 パーティション化された外部表の作成

```
CREATE TABLE sales (loc_id number, prod_id number, cust_id number, amount_sold number, quantity_sold
number)
 ORGANIZATION EXTERNAL
 (TYPE oracle_loader
  DEFAULT DIRECTORY load_d1
  ACCESS PARAMETERS
  ( RECORDS DELIMITED BY NEWLINE CHARACTERSET US7ASCII
    NOBADFILE
    LOGFILE log_dir:'sales.log'
    FIELDS TERMINATED BY ","
   )
 )
  REJECT LIMIT UNLIMITED
 PARTITION BY RANGE (loc_id)
 (PARTITION p1 VALUES LESS THAN (1000) LOCATION ('california.txt'),
  PARTITION p2 VALUES LESS THAN (2000) DEFAULT DIRECTORY load_d2 LOCATION ('washington.txt'),
  PARTITION p3 VALUES LESS THAN (3000))
;
```

関連項目:

外部表のパーティション化の詳細は、『Oracle Database管理者ガイド』を参照してください

4.1.10 キー列に対するパーティション化の指定

レンジ・パーティション表およびハッシュ・パーティション表では、最大16のパーティション化キー列を指定できます。

パーティション化キーが複数の列で構成されていて、後続の列に前の列より細かい粒度が定義されている場合は、複数列のパーティション化を使用してください。最も一般的なシナリオは、年、月、日が別々の列で構成され、分解されているDATEまたはTIMESTAMPキーです。

複数列のパーティション化キーを評価するとき、データベースが第2の値を使用するのは、第1の値で1つのターゲット・パーティションを一意に特定できない場合のみです。第3の値を使用するのは、第1の値と第2の値で正しいパーティションが判別できない場合のみで、第4以降も同様です。1つの値で正しいパーティションを判別できないのは、パーティション・バウンドがその値とまったく同一であり、次のパーティションに同じ上限が定義されている場合だけです。*n*番目の列が調査されるのは、複数列キーの(n-1)より前のすべての値が、パーティションの(n-1)の上限に正確に一致する場合のみです。たとえば、第2の列が評価されるのは、最初の列がパーティション・バウンド値と正確に一致する場合のみです。すべての列値がパーティションのすべてのバウンド値に正確に一致する場合、データベースでは、行がこのパーティションには適合しないとみなされ、次のパーティションに適合すると判断されます。

決定性のないバウンド定義(少なくとも1列に対して同一の値が設定された連続するパーティション)の場合、パーティション・バウンド値は、「以下」を表す上限を含む値になります。値の上限が常に「より小さい」であるとみなされる、決定性のあるバウンドとは対照的です。

次の内容について説明します。

- 複数列の日付別レンジ・パーティション表の作成
- 同一サイズのパーティションを強制する複数列のレンジ・パーティション表の作成

4.1.10.1 複数列の日付別レンジ・パーティション表の作成

このトピックの例は、複数列の日付別レンジ・パーティション表の作成方法を示します。

例4-12では、year、monthおよびdayの別々の3列に実際のDATE情報が格納されている複数列のレンジ・パーティション表の列評価を示します。パーティション化の粒度は四半期です。評価されるパーティション表は、次のように作成されます。

12-DEC-2000の年の値は、1番目のパーティションbefore2001に一致しているため、これ以上評価する必要はありません。

```
SELECT * FROM sales_demo PARTITION(before2001);

      YEAR      MONTH        DAY AMOUNT_SOLD
---------- ---------- ---------- -----------
      2000         12         12        1000
```

17-MAR-2001の情報はパーティションq1_2001に格納されます。1番目のパーティション化キー列yearのみでは適切なパーティションを判断できないため、2番目のパーティション化キー列monthを評価する必要があります。

```
SELECT * FROM sales_demo PARTITION(q1_2001);

      YEAR      MONTH        DAY AMOUNT_SOLD
---------- ---------- ---------- -----------
```

```
      2001          3         17         2000
```

前のレコードと同じ決定ルールに従い、2番目の列monthにより、1-NOV-2001の適切なパーティションはパーティション q4_2001であると判断されます。

```
SELECT * FROM sales_demo PARTITION(q4_2001);

      YEAR      MONTH        DAY AMOUNT_SOLD
---------- ---------- ---------- -----------
      2001         11          1        5000
```

01-JAN-2002のパーティションはyear列のみの評価で、futureパーティションが適切であると判断されます。

```
SELECT * FROM sales_demo PARTITION(future);

      YEAR      MONTH        DAY AMOUNT_SOLD
---------- ---------- ---------- -----------
      2002          1          1        4000
```

パーティション化キー列にMAXVALUEが含まれている場合、後続の列のその他すべての値は無視されます。つまり、前述の例のパーティションfutureの定義における(MAXVALUE,0)という上限は、上限(MAXVALUE,100)または上限(MAXVALUE,MAXVALUE)と同一です。

例4-12 複数列のレンジ・パーティション表の作成

```
CREATE TABLE sales_demo (
   year          NUMBER,
   month         NUMBER,
   day           NUMBER,
   amount_sold   NUMBER)
PARTITION BY RANGE (year,month)
  (PARTITION before2001 VALUES LESS THAN (2001,1),
   PARTITION q1_2001    VALUES LESS THAN (2001,4),
   PARTITION q2_2001    VALUES LESS THAN (2001,7),
   PARTITION q3_2001    VALUES LESS THAN (2001,10),
   PARTITION q4_2001    VALUES LESS THAN (2002,1),
   PARTITION future     VALUES LESS THAN (MAXVALUE,0));

REM  12-DEC-2000
INSERT INTO sales_demo VALUES(2000,12,12, 1000);
REM  17-MAR-2001
INSERT INTO sales_demo VALUES(2001,3,17, 2000);
REM  1-NOV-2001
INSERT INTO sales_demo VALUES(2001,11,1, 5000);
REM  1-JAN-2002
INSERT INTO sales_demo VALUES(2002,1,1, 4000);
```

4.1.10.2 同一サイズのパーティションを強制する複数列のレンジ・パーティション表の作成

このトピックの例は、同一サイズのパーティションを強制する複数列のレンジ・パーティション表の作成方法を示します。

次の例では、どのサプライヤがどの部品を配送するかに関する情報が格納されている表supplier_partsに対して、複数列のパーティション化を使用する方法を示します。少数の専門の部品のみを提供するサプライヤが存在する一方で、非常に多くの部品を提供するサプライヤも存在するため、同じサイズのパーティションにデータを分散するには、表をsupplier_idに基づいてパーティション化するのでは十分ではありません。かわりに、(supplier_id, partnum)で表をパーティション化し、同じサイズのパーティションを手動で強制します。

supplier_idが10より小さいすべての行は、partnum値に関係なくパーティションp1に格納されます。列partnumはsupplier_idが10の場合にのみ評価され、partnumが200以上の場合、対応する行はパーティションp1、p2またはp3に挿入されます。supplier_partsのレンジのパーティション・サイズを同一にするには、supplier_idでレンジ・パーティション化され、partnumでハッシュ・サブパーティション化されたコンポジット・レンジ - ハッシュ・パーティション表を選択します。

複数列のパーティション表にパーティション・バウンドを定義する場合、いくつかのルールに従う必要があります。たとえば、3つの列a、bおよびcでレンジ・パーティション化された表があるとします。個々のパーティションには、次のようなレンジ値があります。

```
P0(a0, b0, c0)
P1(a1, b1, c1)
P2(a2, b2, c2)
...
Pn(an, bn, cn)
```

各パーティションに指定するレンジ値は、次のルールに準拠している必要があります。

- a0はa1以下で、a1はa2以下である必要があります(以降同様)。
- a0＝a1の場合、b0はb1以下である必要があります。a0 < a1の場合、b0およびb1は任意の値でかまいません。a0＝a1かつb0＝b1の場合、c0はc1以下である必要があります。b0<b1の場合、c0およびc1は任意の値でかまいません(以降同様)。
- a1＝a2の場合、b1はb2以下である必要があります。a1＜a2の場合、b1およびb2は任意の値でかまいません。a1＝a2かつb1＝b2の場合、c1はc2以下である必要があります。b1＜b2の場合、c1およびc2は任意の値でかまいません(以降同様)。

```
CREATE TABLE supplier_parts (
   supplier_id      NUMBER,
   partnum          NUMBER,
   price            NUMBER)
PARTITION BY RANGE (supplier_id, partnum)
  (PARTITION p1 VALUES LESS THAN  (10,100),
   PARTITION p2 VALUES LESS THAN (10,200),
   PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE));
```

次の3つのレコードを表に挿入します。

```
INSERT INTO supplier_parts VALUES (5,5, 1000);
INSERT INTO supplier_parts VALUES (5,150, 1000);
INSERT INTO supplier_parts VALUES (10,100, 1000);
```

supplier_idで一意に識別されるため、最初の2つのレコードはパーティションp1に挿入されます。ただし、3番目のレコードは、パーティションp1のすべてのレンジの上限値に正確に一致するため、データベースでは次のパーティションに一致しているとみなされ、パーティションp2に挿入されます。partnumの値が200より小さいという基準に一致しているため、パーティションp2に挿入されます。

```
SELECT * FROM supplier_parts PARTITION (p1);

SUPPLIER_ID    PARTNUM      PRICE
----------- ---------- ----------
          5          5       1000
          5        150       1000

SELECT * FROM supplier_parts PARTITION (p2);

SUPPLIER_ID    PARTNUM      PRICE
----------- ---------- ----------
         10        100       1000
```

4.1.11 仮想列ベースのパーティション化の使用

パーティション化では、仮想列を任意の正規列として使用できます。

仮想列を使用する場合、時間隔パーティション化およびコンポジット・パーティション化のすべての組合せを含み、パーティション・メソッドがすべてサポートされます。パーティション化列として使用する仮想列では、PL/SQLファンクションへのコールは使用できません。

次の例では、サブパーティション化キーに仮想列を使用して、レンジ - レンジでパーティション化されたsales表を示します。仮想列では、amount_soldとquantity_soldを掛けて販売の合計値が計算されます。この例で示されているように、仮想列では行移動もサポートされています。行移動が有効化されている場合、仮想列が別のパーティションに属する値に評価されると、行はあるパーティションから別のパーティションに移行されます。

```
CREATE TABLE sales
  ( prod_id       NUMBER(6) NOT NULL
  , cust_id       NUMBER NOT NULL
  , time_id       DATE NOT NULL
  , channel_id    CHAR(1) NOT NULL
  , promo_id      NUMBER(6) NOT NULL
  , quantity_sold NUMBER(3) NOT NULL
  , amount_sold   NUMBER(10,2) NOT NULL
  , total_amount AS (quantity_sold * amount_sold)
  )
 PARTITION BY RANGE (time_id) INTERVAL (NUMTOYMINTERVAL(1,'MONTH'))
 SUBPARTITION BY RANGE(total_amount)
 SUBPARTITION TEMPLATE
   ( SUBPARTITION p_small VALUES LESS THAN (1000)
   , SUBPARTITION p_medium VALUES LESS THAN (5000)
   , SUBPARTITION p_large VALUES LESS THAN (10000)
   , SUBPARTITION p_extreme VALUES LESS THAN (MAXVALUE)
   )
 (PARTITION sales_before_2007 VALUES LESS THAN
        (TO_DATE('01-JAN-2007','dd-MON-yyyy'))
)
ENABLE ROW MOVEMENT
PARALLEL NOLOGGING;
```

関連項目:

仮想列を作成する構文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

4.1.12 パーティション表での表圧縮の使用

ヒープ構成パーティション表では、表圧縮を使用してパーティションの一部またはすべてを圧縮できます。

表領域、表、または表のパーティションに対して圧縮属性を宣言できます。圧縮属性が指定されていない場合は、その他の記憶域属性と同じように継承されます。

例4-13では、圧縮されたパーティションcosts_oldを含むレンジ・パーティション表を作成します。表およびその他すべてのパーティションの圧縮属性は、表領域レベルから継承されます。

例4-13 圧縮されたパーティションを含むレンジ・パーティション表の作成

```
CREATE TABLE costs_demo (
```

```
   prod_id     NUMBER(6),     time_id     DATE,
   unit_cost   NUMBER(10,2), unit_price  NUMBER(10,2))
PARTITION BY RANGE (time_id)
   (PARTITION costs_old
      VALUES LESS THAN (TO_DATE('01-JAN-2003', 'DD-MON-YYYY')) COMPRESS,
    PARTITION costs_q1_2003
      VALUES LESS THAN (TO_DATE('01-APR-2003', 'DD-MON-YYYY')),
    PARTITION costs_q2_2003
      VALUES LESS THAN (TO_DATE('01-JUN-2003', 'DD-MON-YYYY')),
    PARTITION costs_recent VALUES LESS THAN (MAXVALUE));
```

4.1.13 パーティション索引でのキー圧縮の使用

キー圧縮を使用して、Bツリー索引のパーティションの一部またはすべてを圧縮できます。

キー圧縮は、Bツリー索引にのみ適用できます。ビットマップ索引は、デフォルトで圧縮されて格納されます。キー圧縮を使用する索引では、キー列の接頭辞の値が繰り返し出現しないため、領域を節約しI/Oを削減できます。

次の例では、最新のパーティション以外すべてのパーティションが圧縮されたローカル・パーティション索引を作成します。

```
CREATE INDEX i_cost1 ON costs_demo (prod_id) COMPRESS LOCAL
   (PARTITION costs_old, PARTITION costs_q1_2003,
    PARTITION costs_q2_2003, PARTITION costs_recent NOCOMPRESS);
```

索引サブパーティションには、明示的にCOMPRESS(またはNOCOMPRESS)を指定できません。指定されたパーティションのすべての索引サブパーティションでは、親パーティションからキー圧縮設定を継承します。

指定されたパーティションのすべてのサブパーティションのキー圧縮属性を変更するには、まずALTER INDEX...MODIFY PARTITION文を発行し、すべてのサブパーティションを再作成する必要があります。MODIFY PARTITION句により、すべての索引サブパーティションがUNUSABLEとマークされます。

4.1.14 セグメントによるパーティション化の指定

このトピックでは、セグメントによるパーティション化を紹介します。

このトピックでは、セグメントによるパーティション化を使用する場合の機能について説明します。

- パーティションの遅延セグメント作成
- 空のセグメントの切捨て
- オンデマンドでのセグメント作成のメンテナンス・プロシージャ

4.1.14.1 パーティションの遅延セグメント作成

パーティション表を作成する際に、パーティションに最初の行が挿入されるまでセグメントの作成を遅延させることができます。

最初の行が挿入されると、実表のパーティション、LOB列、すべてのグローバル索引、およびローカル索引パーティションのセグメントが作成されます。遅延セグメント作成は、次の方法で制御できます。

- 初期化パラメータ・ファイルで初期化パラメータDEFERRED_SEGMENT_CREATIONをTRUEまたはFALSEに設定します。
- SQL文ALTER SESSIONまたはALTER SYSTEMにより、初期化パラメータDEFERRED_SEGMENT_CREATIONをTRUEまたはFALSEに設定します。
- SQL文CREATE TABLEを発行する際に、パーティション句でキーワードSEGMENT CREATION IMMEDIATEまたはSEGMENT CREATION DEFERREDを指定します。

既存の作成済パーティションに対するセグメントの作成は、SQL文ALTER TABLE MODIFY PARTITION ALLOCATE EXTENTにより強制できます。この文により、CREATE TABLE時に指定された初期エクステント数より1つ多くエクステントが割り当てられます。

シリアル化可能トランザクションは、遅延セグメント作成ではサポートされていません。セグメントが作成されていない空の表、またはまだセグメントのない時間隔パーティション表のパーティションにデータを挿入すると、エラーが発生する場合があります。

関連項目:

- 初期化パラメータDEFERRED_SEGMENT_CREATIONの詳細は、『Oracle Databaseリファレンス』を参照してください
- SQL文ALTER SESSIONおよびALTER SYSTEMの詳細は、『Oracle Database SQL言語リファレンス』を参照してください
- SQL文CREATE TABLEのキーワードSEGMENT CREATION IMMEDIATEおよびSEGMENT CREATION DEFERREDの詳細は、『Oracle Database SQL言語リファレンス』を参照してください

4.1.14.2 空のセグメントの切捨て

DBMS_SPACE_ADMIN.DROP_EMPTY_SEGMENTSプロシージャにより、表および表フラグメント内の空のセグメントを削除できます。

さらに、パーティションまたサブパーティションにセグメントがある場合は、SQL文ALTER TABLE TRUNCATE PARTITIONでDROP ALL STORAGE句を指定すると、切捨て機能によりセグメントが削除されます。

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
- ALTER TABLEのDROP ALL STORAGE句の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

4.1.14.3 オンデマンドでのセグメント作成のメンテナンス・プロシージャ

DBMS_SPACE_ADMINパッケージでMATERIALIZE_DEFERRED_SEGMENTSプロシージャを使用して、表のセグメントと表の従属オブジェクトを遅延セグメント・プロパティで作成できます。

DBMS_SPACE_ADMIN.MATERIALIZE_DEFERRED_SEGMENTSプロシージャを使用すると、作成済の既存の表や表フラグメントで、セグメントを強制的に作成することもできます。MATERIALIZE_DEFERRED_SEGMENTSプロシージャは、表または表フラグメントにエクステントを1つ追加で割り当てることはしません。この点で、SQL文ALTER TABLE MODIFY PARTITION ALLOCATE EXTENTとは異なります。

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

4.1.15 索引構成表を作成する場合のパーティション化の指定

索引構成表では、レンジ、リストまたはハッシュ・パーティション化メソッドを使用できます。

パーティション索引構成表を作成する際のセマンティックは、通常の表のセマンティックに似ていますが、次のような違いがあります。

- 表の作成時に、ORGANIZATION INDEX句、および必要に応じてINCLUDINGとOVERFLOW句を指定します。
- PARTITION句にOVERFLOW副次句を指定できます。この副次句を使用すると、オーバーフロー・セグメントの属性をパーティション・レベルで指定できます。

OVERFLOW句を指定すると、オーバーフロー・データ・セグメント自体が主キー索引セグメントと同一レベル・パーティション化されます。このため、オーバーフローを含むパーティション索引構成表では、各パーティションに索引セグメントおよびオーバーフロー・データ・セグメントがあります。

索引構成表では、一連のパーティション化列は主キー列のサブセットである必要があります。索引構成表の行は表の主キー索引に格納されるため、パーティション化の基準は可用性に影響します。主キーのサブセットになるパーティション化キーを選択することにより、挿入操作では単一パーティションの主キーの一意性のみを検証すればよくなるため、パーティションの独立性が保たれます。

索引構成表の2次索引のサポートは、通常の表のサポートに似ています。2次索引の論理的な特性のため、通常の表ではUNUSABLEとマークされるような特定の操作に、索引構成表のグローバル索引を使用できます。

次の内容について説明します。

- レンジ・パーティションの索引構成表の作成
- ハッシュ・パーティションの索引構成表の作成
- リスト・パーティションの索引構成表の作成

関連項目:

- 索引構成表のメンテナンス操作の詳細は、「パーティション表および索引のメンテナンス操作」を参照してください
- 索引構成表の管理の詳細は、『Oracle Database管理者ガイド』を参照してください
- 索引構成表の詳細は、『Oracle Database概要』を参照してください。

4.1.15.1 レンジ・パーティションの索引構成表の作成

索引構成表およびその2次索引をレンジ・パーティション化できます。

例4-14では、レンジ・パーティション化された索引構成表salesを作成します。INCLUDING句は、week_noの後のすべての列がオーバーフロー・セグメントに格納されるように指定します。各パーティションには1つのオーバーフロー・セグメントがあり、すべて同じ表領域(overflow_here)に格納されます。オプションで、個々のパーティション・レベルでOVERFLOW TABLESPACEを指定できます。そうすると、一部またはすべてのオーバーフロー・セグメントが別のTABLESPACE属性を持つことができます。

例4-14 レンジ・パーティションの索引構成表の作成

```
CREATE TABLE sales(acct_no NUMBER(5),
                   acct_name CHAR(30),
                   amount_of_sale NUMBER(6),
                   week_no INTEGER,
                   sale_details VARCHAR2(1000),
```

```
            PRIMARY KEY (acct_no, acct_name, week_no))
    ORGANIZATION INDEX
            INCLUDING week_no
            OVERFLOW TABLESPACE overflow_here
    PARTITION BY RANGE (week_no)
           (PARTITION VALUES LESS THAN (5)
                  TABLESPACE ts1,
            PARTITION VALUES LESS THAN (9)
                  TABLESPACE ts2 OVERFLOW TABLESPACE overflow_ts2,
            ...
            PARTITION VALUES LESS THAN (MAXVALUE)
                  TABLESPACE ts13);
```

4.1.15.2 ハッシュ・パーティションの索引構成表の作成

索引構成表のパーティション化のその他のオプションは、ハッシュ・メソッドの使用です。

例4-15では、ハッシュ・メソッドでsales索引構成表をパーティション化しています。

ノート:

ハッシュ関数は、パーティション間にバランスよく行を分散するよう綿密に設計されています。そのため、行の主キー列を更新すると、その行は高確率で別のパーティションに移動されます。変更可能なパーティション化キーを使用してハッシュ・パーティションの索引構成表を作成する場合には、明示的に ENABLE ROW MOVEMENT 句を指定することをお薦めします。デフォルトでは、ENABLE ROW MOVEMENT は無効化されています。

例4-15 ハッシュ・パーティションの索引構成表の作成

```
CREATE TABLE sales(acct_no NUMBER(5),
                  acct_name CHAR(30),
                  amount_of_sale NUMBER(6),
                  week_no INTEGER,
                  sale_details VARCHAR2(1000),
            PRIMARY KEY (acct_no, acct_name, week_no))
    ORGANIZATION INDEX
            INCLUDING week_no
    OVERFLOW
         PARTITION BY HASH (week_no)
            PARTITIONS 16
            STORE IN (ts1, ts2, ts3, ts4)
            OVERFLOW STORE IN (ts3, ts6, ts9);
```

4.1.15.3 リスト・パーティションの索引構成表の作成

索引構成表のパーティション化の別のオプションは、リスト・メソッドの使用です。

例4-16では、リスト・メソッドでsales索引構成表をパーティション化しています。

例4-16 リスト・パーティションの索引構成表の作成

```
CREATE TABLE sales(acct_no NUMBER(5),
                  acct_name CHAR(30),
                  amount_of_sale NUMBER(6),
                  week_no INTEGER,
                  sale_details VARCHAR2(1000),
            PRIMARY KEY (acct_no, acct_name, week_no))
```

```
    ORGANIZATION INDEX
            INCLUDING week_no
            OVERFLOW TABLESPACE ts1
    PARTITION BY LIST (week_no)
           (PARTITION VALUES (1, 2, 3, 4)
                  TABLESPACE ts2,
            PARTITION VALUES (5, 6, 7, 8)
                  TABLESPACE ts3 OVERFLOW TABLESPACE ts4,
            PARTITION VALUES (DEFAULT)
                  TABLESPACE ts5);
```

4.1.16 複数ブロック・サイズのパーティション化制限

ブロック・サイズの異なる表領域を含むデータベースに、パーティション・オブジェクトを作成する際には注意してください。

そのような表領域のパーティション・オブジェクトの記憶域は、一部の制限の対象です。特に、次に示すエンティティのすべてのパーティションは、同じブロック・サイズの表領域に存在する必要があります。

- 従来型の表
- 索引
- 索引構成表の主キーの索引セグメント
- 索引構成表のオーバーフロー・セグメント
- 表外に格納されているLOB列

したがって、次のようになります。

- 従来型の各表では、その表のすべてのパーティションをブロック・サイズが同一の表領域に格納する必要があります。
- 各索引構成表では、すべての主キーの索引パーティション、およびその表のすべてのオーバーフロー・パーティションが、ブロック・サイズが同一の表領域に存在する必要があります。ただし、索引パーティションおよびオーバーフロー・パーティションは、ブロック・サイズが異なる表領域に存在してもかまいません。
- 各索引(グローバルまたはローカル)では、その索引の各パーティションはブロック・サイズが同一の表領域に存在する必要があります。ただし、同じオブジェクトに定義されている異なる索引のパーティションは、ブロック・サイズが異なる表領域に存在してもかまいません。
- 各LOB列では、その列の各パーティションをブロック・サイズが同一の表領域に格納する必要があります。ただし、異なるLOB列は、ブロック・サイズが異なる表領域に格納してもかまいません。

パーティション表またはパーティション索引を作成または変更するとき、各エンティティのパーティションとサブパーティションに明示的に指定する表領域は、同じブロック・サイズであることが必要です。表領域記憶域をエンティティに明示的に指定しない場合、データベースがデフォルトで使用する表領域は、同じブロック・サイズである必要があります。そのため、パーティション・オブジェクトの各レベルのデフォルト表領域を把握している必要があります。

4.1.17 XMLTypeおよびオブジェクトのコレクションのパーティション化

XMLTypeまたはオブジェクトの表と列を使用する際のパーティション化は、パーティション化の基本的なルールに従います。

説明のために、 ここではコレクション表という語を、(1)XMLTypeの表または列内のOrdered Collection Tables、(2)オブジェクトの表または列内のネスト表、という2つのカテゴリに使用します。

コレクション表をパーティション化するときは、実表のパーティション化方法がOracle Databaseによって使用されます。また、コレクション表は、実表がパーティション化されるときに自動的にパーティション化されます。パーティション化されたネスト表に対する

DMLは、参照パーティション表に対する場合と似た方法で動作します。

Oracle Databaseで提供されるLOCALキーワードを使用して、コレクション表を、パーティション化された実表に対応するように同一レベル・パーティション化します。これは、このリリースでのデフォルト動作です。以前のリリースのデフォルトでは、コレクション表は、パーティション化された実表に対して同一レベル・パーティション化されませんでした。現在、パーティション化されていないコレクション表をパーティション化された実表と一緒に格納するためには、GLOBALキーワードを指定する必要があります。

アウトオブライン(OOL)表のパーティション化がサポートされています。ただし、アウトオブライン表を持つ同じXMLスキーマの表を2つ作成することはできません。つまり、同じスキーマの表を2つ持つことはできないため、OOL表を含むスキーマでは、パーティションの交換は行えません。

次の例の文によって、ネストした表のパーティションが作成されます。

```
CREATE TABLE print_media_part (
   product_id NUMBER(6),
   ad_id NUMBER(6),
   ad_composite BLOB,
   ad_sourcetext CLOB,
   ad_finaltext CLOB,
   ad_fltextn NCLOB,
   ad_textdocs_ntab TEXTDOC_TAB,
   ad_photo BLOB,
   ad_graphic BFILE,
   ad_header ADHEADER_TYP)
NESTED TABLE ad_textdocs_ntab STORE AS textdoc_nt
PARTITION BY RANGE (product_id)
  (PARTITION p1 VALUES LESS THAN (100),
   PARTITION p2 VALUES LESS THAN (200));
```

関連項目:

- その他の関連する例については、「コレクション表を含むパーティションに対するPMOの実行」および「バイナリXML表のXMLIndexのパーティション化」を参照してください
- パーティション化されたネスト表に対して問合せを発行し、EXPLAIN PLANを使用してパフォーマンスを改善する方法の例は、「コレクション表」を参照してください
- オンライン再定義を使用した既存の非パーティション・コレクション表からパーティション表への変換の詳細は「パーティション表への非パーティション表の変更」を参照してください
- CREATE TABLE構文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

4.1.17.1 コレクション表を含むパーティションに対するPMOの実行

パーティションにコレクション表が含まれるかどうかは、パーティションのメンテナンス操作(PMO)の実行にそれほど影響しません。

通常、コレクション表のメンテナンス操作は実表に対して行われます。次の例は、前に示したネストした表のパーティションに基づいて、典型的なADD PARTITION操作を示します。

```
ALTER TABLE print_media_part
   ADD PARTITION p4 VALUES LESS THAN (400)
   LOB(ad_photo, ad_composite) STORE AS (TABLESPACE omf_ts1)
   LOB(ad_sourcetext, ad_finaltext) STORE AS (TABLESPACE omf_ts1)
   NESTED TABLE ad_textdocs_ntab STORE AS nt_p3;
```

ネストした表の記憶域列ad_textdocs_ntabの記憶域表はnt_p3という名前が付けられ、その他すべての属性を表レベルのデフォルトおよび表領域のデフォルトから継承します。

次のパーティション・メンテナンス操作は、コレクション列に対応する記憶域表に対して直接起動する必要があります。

- パーティションの変更
- パーティションの移動
- パーティション名の変更
- パーティションのデフォルト属性の変更

関連項目:

- ADD PARTITIONの構文については、『Oracle Database SQL言語リファレンス』を参照してください
- パーティション表およびコンポジット・パーティション表に対して実行できるパーティション・メンテナンス操作のリストは、「パーティションでサポートされているメンテナンス操作」を参照してください

4.1.17.2 バイナリXML表のXMLIndexのパーティション化

バイナリXML表の場合、XMLIndexは、レンジ、ハッシュ、リスト、時間隔および参照パーティションの実表で同一レベル・パーティション化されます。

次の例では、レンジ・パーティション表でXMLIndexが作成されます。

```
CREATE TABLE purchase_order
   (id NUMBER, doc XMLTYPE)
    PARTITION BY RANGE (id)
    (PARTITION p1 VALUES LESS THAN (10),
     PARTITION p2 VALUES LESS THAN (MAXVALUE));

CREATE INDEX purchase_order_idx ON purchase_order(doc)
       INDEXTYPE IS XDB.XMLINDEX LOCAL;
```

関連項目:

- Oracle XML DBおよびバイナリXML表のXMLIndexのパーティション化の詳細は、『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
- XMLIndexの詳細は、『*Oracle XML DB*開発者ガイド』を参照してください
- XMLTypeの表および列のパーティション化の詳細は、『*Oracle XML DB*開発者ガイド』を参照してください

4.2 表を作成する場合のコンポジット・パーティション化の指定

コンポジット・パーティション表を作成する場合は、CREATE TABLE SQL文のPARTITION句およびSUBPARTITION句を使用します。

コンポジット・パーティション表を作成するには、まずCREATE TABLE文のPARTITION BY {HASH | RANGE [INTERVAL]| LIST} 句を使用します。次に、PARTITION BY句と同様の構文とルールに準拠する、SUBPARTITION BY句を指定します。

次の内容について説明します。

- コンポジット・ハッシュ - *パーティション表の作成
- コンポジット時間隔 - *パーティション表の作成
- コンポジット・リスト - *パーティション表の作成
- コンポジット・レンジ - *パーティション表の作成
- コンポジット・パーティション表を説明するサブパーティション・テンプレートの指定

4.2.1 コンポジット・ハッシュ - *パーティション表の作成

コンポジット・ハッシュ - *パーティション化では、2つのディメンションに従ってハッシュ・パーティション化できます。

コンポジット・ハッシュ - ハッシュ・パーティション化計画には、コンポジット・ハッシュ - *パーティション表の大部分のビジネス価値が含まれています。この技術は、2つのディメンションでパーティション・ワイズ結合を可能にする利点があります。

次の例では、コンポジット・ハッシュ - ハッシュ・パーティション表の作成時にサブパーティションの数を指定します。ただし、名前は指定しません。システム生成の名前がパーティションおよびサブパーティションに割り当てられます。それらは、表のデフォルト表領域に格納されます。

Live SQL:

関連する例を Oracle Live SQL の *Oracle Live SQL: コンポジット・ハッシュ - ハッシュ・パーティション表の作成*で参照して実行してください。

例4-17 コンポジット・ハッシュ - ハッシュ・パーティション表の作成

```
CREATE TABLE departments_courses_hash (
             department_id NUMBER(4) NOT NULL,
             department_name VARCHAR2(30),
             course_id NUMBER(4) NOT NULL)
    PARTITION BY HASH(department_id)
    SUBPARTITION BY HASH (course_id) SUBPARTITIONS 32 PARTITIONS 16;
```

関連項目:

サブパーティション・テンプレートの使用によって、コンポジット・パーティション表の指定がどのように簡素化されるか方法を学習するには、「コンシューマ・グループを使用したユーザーに対するリソース数の制限時期」を参照してください

4.2.2 コンポジット時間隔 - *パーティション表の作成

時間隔 - *コンポジット・パーティション化の概念は、レンジ - *パーティション化に似ています。

INTERVAL定義を含むようにPARTITION BY RANGE句を拡張します。PARTITIONを使用して少なくとも1つのレンジ・パーティションを指定してください。レンジ・パーティション化キーの値は、遷移ポイントと呼ばれるレンジ・パーティションの値の上限を決定します。遷移ポイントを超えるデータのために、データベースによって時間隔パーティションが自動的に作成されます。

時間隔 - *パーティション表の時間隔のサブパーティションは、データベースによる時間隔の作成時に作成されます。後続のサブパーティションの定義は、サブパーティション・テンプレートでのみ指定できます。

次の各トピックでは、各種の時間隔 - *コンポジット・パーティション化メソッドの例を示します。

- コンポジット時間隔 - ハッシュ・パーティション表の作成
- コンポジット時間隔 - リスト・パーティション表の作成
- コンポジット時間隔 - レンジ・パーティション表の作成

関連項目:

サブパーティション・テンプレートの使用によって、コンポジット・パーティション表の指定がどのように簡素化されるか方法を学習するには、「コンシューマ・グループを使用したユーザーに対するリソース数の制限時期」を参照してください

4.2.2.1 コンポジット時間隔 - ハッシュ・パーティション表の作成

複数のハッシュ・パーティションを含む時間隔 - ハッシュ・パーティション表は、PARTITION句で複数のハッシュ・パーティションを指定するか、サブパーティション・テンプレートを使用して作成できます。

これらのいずれの方法も使用しない場合、後続の時間隔パーティションにはハッシュ・サブパーティションが1つのみ作成されます。

次の例では、time_idにより月間隔で時間隔パーティション化され、cust_idでハッシュ・サブパーティション化されたsales表を示します。この例では、個々のハッシュ・パーティションに特定の表領域を割り当てずに、複数のハッシュ・パーティションが指定されています。

Live SQL:

関連する例を Oracle Live SQL の *Oracle Live SQL: コンポジット時間隔 - ハッシュ・パーティション表の作成*で参照して実行してください。

```
CREATE TABLE sales
```

(prod_id NUMBER(6) , cust_id NUMBER , time_id DATE , channel_id CHAR(1) , promo_id NUMBER(6) , quantity_sold NUMBER(3) , amount_sold NUMBER(10,2)) PARTITION BY RANGE (time_id) INTERVAL (NUMTOYMINTERVAL(1,'MONTH')) SUBPARTITION BY HASH (cust_id) SUBPARTITIONS 4 (PARTITION before_2000 VALUES LESS THAN (TO_DATE('01-JAN-2000','dd-MON-yyyy'))) PARALLEL;

次の例では、time_idにより月間隔で時間隔パーティション化され、cust_idでハッシュ・サブパーティション化された同じsales表を示します。ただし、この例では、個々のハッシュ・パーティションは別々の表領域に格納されます。後続のハッシュ・サブパー

ティションへの表領域の割当てを定義するために、サブパーティション・テンプレートが使用されています。

```
CREATE TABLE sales
  ( prod_id       NUMBER(6)
  , cust_id       NUMBER
  , time_id       DATE
  , channel_id    CHAR(1)
  , promo_id      NUMBER(6)
  , quantity_sold NUMBER(3)
  , amount_sold   NUMBER(10,2)
  )
 PARTITION BY RANGE (time_id) INTERVAL (NUMTOYMINTERVAL(1,'MONTH'))
 SUBPARTITION BY hash(cust_id)
   SUBPARTITION template
   ( SUBPARTITION p1 TABLESPACE ts1
   , SUBPARTITION p2 TABLESPACE ts2
   , SUBPARTITION p3 TABLESPACE ts3
   , SUBPARTITION P4 TABLESPACE ts4
   )
 (PARTITION before_2000 VALUES LESS THAN (TO_DATE('01-JAN-2000','dd-MON-yyyy'))
  )
PARALLEL;
```

4.2.2.2 コンポジット時間隔 - リスト・パーティション表の作成

後続の時間隔 - リスト・パーティションのリスト・サブパーティションを定義するには、サブパーティション・テンプレートを使用する必要があります。

サブパーティション・テンプレートを使用しない場合、各時間隔パーティションに作成されるサブパーティションはDEFAULTサブパーティションのみです。

例4-18では、sales_dateにより月間隔で時間隔パーティション化され、channel_idでリスト・サブパーティション化されたsales_interval_list表を示します。

例4-18 コンポジット時間隔 - リスト・パーティション表の作成

```
CREATE TABLE sales_interval_list
  ( product_id       NUMBER(6)
  , customer_id      NUMBER
  , channel_id       CHAR(1)
  , promo_id         NUMBER(6)
  , sales_date       DATE
  , quantity_sold    INTEGER
  , amount_sold      NUMBER(10,2)
  )
 PARTITION BY RANGE (sales_date) INTERVAL (NUMTOYMINTERVAL(1,'MONTH'))
 SUBPARTITION BY LIST (channel_id)
   SUBPARTITION TEMPLATE
   ( SUBPARTITION p_catalog VALUES ('C')
   , SUBPARTITION p_internet VALUES ('I')
   , SUBPARTITION p_partners VALUES ('P')
   , SUBPARTITION p_direct_sales VALUES ('S')
   , SUBPARTITION p_tele_sales VALUES ('T')
   )
 (PARTITION before_2017 VALUES LESS THAN (TO_DATE('01-JAN-2017','dd-MON-yyyy'))
  )
PARALLEL;
```

```
SELECT TABLE_NAME, PARTITION_NAME, SUBPARTITION_NAME FROM USER_TAB_SUBPARTITIONS WHERE TABLE_NAME
='SALES_INTERVAL_LIST';
```

4.2.2.3 コンポジット時間隔 - レンジ・パーティション表の作成

後続の時間隔 - レンジ・パーティションのレンジ・サブパーティションを定義するには、サブパーティション・テンプレートを使用する必要があります。

サブパーティション・テンプレートを使用しない場合、各時間隔パーティションに作成されるサブパーティションは上限MAXVALUE付きのレンジ・サブパーティションのみです。

例4-19では、time_idにより日次間隔で時間隔パーティション化され、amount_soldでレンジ・サブパーティション化されたsales表を示します。

例4-19 コンポジット時間隔 - レンジ・パーティション表の作成

```
CREATE TABLE sales
  ( prod_id       NUMBER(6)
  , cust_id       NUMBER
  , time_id       DATE
  , channel_id    CHAR(1)
  , promo_id      NUMBER(6)
  , quantity_sold NUMBER(3)
  , amount_sold   NUMBER(10,2)
  )
 PARTITION BY RANGE (time_id) INTERVAL (NUMTODSINTERVAL(1,'DAY'))
SUBPARTITION BY RANGE(amount_sold)
   SUBPARTITION TEMPLATE
   ( SUBPARTITION p_low VALUES LESS THAN (1000)
   , SUBPARTITION p_medium VALUES LESS THAN (4000)
   , SUBPARTITION p_high VALUES LESS THAN (8000)
   , SUBPARTITION p_ultimate VALUES LESS THAN (maxvalue)
   )
 (PARTITION before_2000 VALUES LESS THAN (TO_DATE('01-JAN-2000','dd-MON-yyyy'))
  )
PARALLEL;
```

4.2.3 コンポジット・リスト - *パーティション表の作成

リスト - ハッシュ、リスト - リストおよびリスト - レンジのコンポジット・パーティション化の概念は、レンジ - ハッシュ、レンジ - リストおよびレンジ - レンジのパーティション化の概念と似ています。

ただし、リスト - *コンポジット・パーティション化の場合、PARTITION BY LISTを指定してパーティション化計画を定義します。

リスト - *コンポジット・パーティション表のリスト・パーティションは、非コンポジット・レンジ・パーティション表に似ています。この構成により、PARTITION句のオプションの副次句で、パーティション・セグメントに固有の物理属性およびその他の属性(表領域を含む)を指定できます。パーティション・レベルで上書きされない場合、パーティションは基礎となる表の属性を継承します。

SUBPARTITIONまたはSUBPARTITIONS句のサブパーティションの説明は、レンジ - *コンポジット・パーティション化に似ています。

次の各トピックでは、各種のリスト - *コンポジット・パーティション化メソッドの例を示します。

- コンポジット・リスト - ハッシュ・パーティション表の作成
- コンポジット・リスト - リスト・パーティション表の作成
- コンポジット・リスト - レンジ・パーティション表の作成

関連項目:

- サブパーティション・テンプレートの使用によって、コンポジット・パーティション表の指定がどのように簡素化されるか方法を学習するには、「コンシューマ・グループを使用したユーザーに対するリソース数の制限時期」を参照してください
- リスト - ハッシュ・コンポジット・パーティション化方法のサブパーティション定義の詳細は、「コンポジット・レンジ - ハッシュ・パーティション表の作成について」を参照してください
- リスト - リスト・コンポジット・パーティション化方法のサブパーティション定義の詳細は、「コンポジット・レンジ - リスト・パーティション表の作成について」を参照してください
- リスト - レンジ・コンポジット・パーティション化方法のサブパーティション定義の詳細は、「コンポジット・レンジ - レンジ・パーティション表の作成について」を参照してください

4.2.3.1 コンポジット・リスト - ハッシュ・パーティション表の作成

このトピックの例は、コンポジット・リスト - ハッシュ・パーティション表の作成方法を示します。

例4-20では、リージョンでリスト・パーティション化され、顧客IDによるハッシングを使用してサブパーティション化されたaccounts表を示します。

例4-20 リスト - ハッシュ・パーティション表の作成

```
CREATE TABLE accounts
( id              NUMBER
 , account_number NUMBER
 , customer_id    NUMBER
 , balance        NUMBER
 , branch_id      NUMBER
 , region         VARCHAR(2)
 , status         VARCHAR2(1)
)
PARTITION BY LIST (region)
SUBPARTITION BY HASH (customer_id) SUBPARTITIONS 8
( PARTITION p_northwest VALUES ('OR', 'WA')
 , PARTITION p_southwest VALUES ('AZ', 'UT', 'NM')
 , PARTITION p_northeast VALUES ('NY', 'VM', 'NJ')
 , PARTITION p_southeast VALUES ('FL', 'GA')
 , PARTITION p_northcentral VALUES ('SD', 'WI')
 , PARTITION p_southcentral VALUES ('OK', 'TX')
);
```

4.2.3.2 コンポジット・リスト - リスト・パーティション表の作成

このトピックの例は、コンポジット・リスト - リスト・パーティション表の作成方法を示します。

例4-21では、リージョンでリスト・パーティション化され、アカウント・ステータスによるリスト化を使用してサブパーティション化されたaccounts表を示します。

Live SQL:

関連する例を Oracle Live SQL の *Oracle Live SQL: コンポジット・リスト - リスト・パーティション表の作*

成で参照して実行してください。

例4-21 コンポジット・リスト - リスト・パーティション表の作成

```
CREATE TABLE accounts
( id             NUMBER
, account_number NUMBER
, customer_id    NUMBER
, balance        NUMBER
, branch_id      NUMBER
, region         VARCHAR(2)
, status         VARCHAR2(1)
)
PARTITION BY LIST (region)
SUBPARTITION BY LIST (status)
( PARTITION p_northwest VALUES ('OR', 'WA')
  ( SUBPARTITION p_nw_bad VALUES ('B')
  , SUBPARTITION p_nw_average VALUES ('A')
  , SUBPARTITION p_nw_good VALUES ('G')
  )
, PARTITION p_southwest VALUES ('AZ', 'UT', 'NM')
  ( SUBPARTITION p_sw_bad VALUES ('B')
  , SUBPARTITION p_sw_average VALUES ('A')
  , SUBPARTITION p_sw_good VALUES ('G')
  )
, PARTITION p_northeast VALUES ('NY', 'VM', 'NJ')
  ( SUBPARTITION p_ne_bad VALUES ('B')
  , SUBPARTITION p_ne_average VALUES ('A')
  , SUBPARTITION p_ne_good VALUES ('G')
  )
, PARTITION p_southeast VALUES ('FL', 'GA')
  ( SUBPARTITION p_se_bad VALUES ('B')
  , SUBPARTITION p_se_average VALUES ('A')
  , SUBPARTITION p_se_good VALUES ('G')
  )
, PARTITION p_northcentral VALUES ('SD', 'WI')
  ( SUBPARTITION p_nc_bad VALUES ('B')
  , SUBPARTITION p_nc_average VALUES ('A')
  , SUBPARTITION p_nc_good VALUES ('G')
  )
, PARTITION p_southcentral VALUES ('OK', 'TX')
  ( SUBPARTITION p_sc_bad VALUES ('B')
  , SUBPARTITION p_sc_average VALUES ('A')
  , SUBPARTITION p_sc_good VALUES ('G')
  )
);
```

4.2.3.3 コンポジット・リスト - レンジ・パーティション表の作成

このトピックの例は、コンポジット・リスト - レンジ・パーティション表の作成方法を示します。

例4-22では、リージョンでリスト・パーティション化され、勘定残高によるレンジ化を使用してサブパーティション化され、行移動が有効化されたaccounts表を示します。異なるリスト・パーティションのサブパーティションには、別々のレンジを指定できます。

例4-22 コンポジット・レンジ - レンジ・パーティション表の作成

```
CREATE TABLE accounts
( id             NUMBER
, account_number NUMBER
```

```
, customer_id    NUMBER
, balance        NUMBER
, branch_id      NUMBER
, region         VARCHAR(2)
, status         VARCHAR2(1)
)
PARTITION BY LIST (region)
SUBPARTITION BY RANGE (balance)
( PARTITION p_northwest VALUES ('OR', 'WA')
  ( SUBPARTITION p_nw_low VALUES LESS THAN (1000)
  , SUBPARTITION p_nw_average VALUES LESS THAN (10000)
  , SUBPARTITION p_nw_high VALUES LESS THAN (100000)
  , SUBPARTITION p_nw_extraordinary VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_southwest VALUES ('AZ', 'UT', 'NM')
  ( SUBPARTITION p_sw_low VALUES LESS THAN (1000)
  , SUBPARTITION p_sw_average VALUES LESS THAN (10000)
  , SUBPARTITION p_sw_high VALUES LESS THAN (100000)
  , SUBPARTITION p_sw_extraordinary VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_northeast VALUES ('NY', 'VM', 'NJ')
  ( SUBPARTITION p_ne_low VALUES LESS THAN (1000)
  , SUBPARTITION p_ne_average VALUES LESS THAN (10000)
  , SUBPARTITION p_ne_high VALUES LESS THAN (100000)
  , SUBPARTITION p_ne_extraordinary VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_southeast VALUES ('FL', 'GA')
  ( SUBPARTITION p_se_low VALUES LESS THAN (1000)
  , SUBPARTITION p_se_average VALUES LESS THAN (10000)
  , SUBPARTITION p_se_high VALUES LESS THAN (100000)
  , SUBPARTITION p_se_extraordinary VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_northcentral VALUES ('SD', 'WI')
  ( SUBPARTITION p_nc_low VALUES LESS THAN (1000)
  , SUBPARTITION p_nc_average VALUES LESS THAN (10000)
  , SUBPARTITION p_nc_high VALUES LESS THAN (100000)
  , SUBPARTITION p_nc_extraordinary VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_southcentral VALUES ('OK', 'TX')
  ( SUBPARTITION p_sc_low VALUES LESS THAN (1000)
  , SUBPARTITION p_sc_average VALUES LESS THAN (10000)
  , SUBPARTITION p_sc_high VALUES LESS THAN (100000)
  , SUBPARTITION p_sc_extraordinary VALUES LESS THAN (MAXVALUE)
  )
) ENABLE ROW MOVEMENT;
```

4.2.4 コンポジット・レンジ - *パーティション表の作成

このトピックでは、コンポジット・レンジ - *パーティション表の作成方法を紹介します。

次の各トピックでは、各種のレンジ - *コンポジット・パーティション化メソッドの例を示します。

- コンポジット・レンジ - ハッシュ・パーティション表の作成について
- コンポジット・レンジ - リスト・パーティション表の作成について
- コンポジット・レンジ - レンジ・パーティション表の作成

関連項目:

サブパーティション・テンプレートの使用によって、コンポジット・パーティション表の指定がどのように簡素化されるか方法を学習するには、「コンシューマ・グループを使用したユーザーに対するリソース数の制限時期」を参照してください

4.2.4.1 コンポジット・レンジ - ハッシュ・パーティション表の作成について

レンジ - ハッシュ・パーティション表のパーティションは、そのデータがサブパーティションのセグメントに格納されるため、論理構造のみです。

パーティション同様、これらのサブパーティションでも同じ論理属性を共有します。レンジ・パーティション表のレンジ・パーティションとは異なり、所有パーティションと同じ表領域に存在する必要はありませんが、サブパーティションの物理属性は所有パーティションの属性と同一である必要があります。

次の内容について説明します。

- 同じ表領域のコンポジット・レンジ - ハッシュ・パーティション表の作成
- 異なる表領域のコンポジット・レンジ - ハッシュ・パーティション表の作成
- 複数の表領域にまたがるローカル索引の作成

関連項目:

サブパーティション・テンプレートの使用によって、コンポジット・パーティション表の指定がどのように簡素化されるか方法を学習するには、「コンシューマ・グループを使用したユーザーに対するリソース数の制限時期」を参照してください

4.2.4.1.1 同じ表領域のコンポジット・レンジ - ハッシュ・パーティション表の作成

このトピックの例は、同じ表領域を使用するコンポジット・レンジ - ハッシュ・パーティション表の作成方法を示します。

例4-23の文は、レンジ - ハッシュ・パーティション表を作成します。4つのレンジ・パーティションが作成され、それぞれに8つのサブパーティションが含まれています。サブパーティションには名前が付けられていないため、システム生成の名前が割り当てられていますが、STORE IN句で指定された4つの表領域(ts1、ts2、ts3、ts4)に格納されます。

例4-23 1つのSTORE IN句を使用した複合レンジ - ハッシュ・パーティション表の作成

```
CREATE TABLE sales
  ( prod_id       NUMBER(6)
  , cust_id       NUMBER
  , time_id       DATE
  , channel_id    CHAR(1)
  , promo_id      NUMBER(6)
  , quantity_sold NUMBER(3)
  , amount_sold   NUMBER(10,2)
  )
 PARTITION BY RANGE (time_id) SUBPARTITION BY HASH (cust_id)
  SUBPARTITIONS 8 STORE IN (ts1, ts2, ts3, ts4)
 ( PARTITION sales_q1_2006 VALUES LESS THAN (TO_DATE('01-APR-2006','dd-MON-yyyy'))
  , PARTITION sales_q2_2006 VALUES LESS THAN (TO_DATE('01-JUL-2006','dd-MON-yyyy'))
  , PARTITION sales_q3_2006 VALUES LESS THAN (TO_DATE('01-OCT-2006','dd-MON-yyyy'))
  , PARTITION sales_q4_2006 VALUES LESS THAN (TO_DATE('01-JAN-2007','dd-MON-yyyy'))
 );
```

4.2.4.1.2 異なる表領域のコンポジット・レンジ - ハッシュ・パーティション表の作成

このトピックの例は、異なる表領域を使用するコンポジット・レンジ - ハッシュ・パーティション表の作成方法を示します。

レンジ・パーティションに指定された属性は、そのパーティションのすべてのサブパーティションに適用されます。各レンジ・パーティションに異なる属性を指定することができ、そのパーティションのサブパーティションを格納する表領域のリストが、その他のパーティションのリストと異なる場合は、パーティション・レベルでSTORE IN句を指定することもできます。これは、次の例に示されています。

```
CREATE TABLE employees_range_hash
         (department_id NUMBER(4) NOT NULL,
          last_name VARCHAR2(25),
          job_id VARCHAR2(10))
     PARTITION BY RANGE(department_id) SUBPARTITION BY HASH(last_name)
        SUBPARTITIONS 8 STORE IN (ts1, ts3, ts5, ts7)
    (PARTITION p1 VALUES LESS THAN (1000),
     PARTITION p2 VALUES LESS THAN (2000)
        STORE IN (ts2, ts4, ts6, ts8),
     PARTITION p3 VALUES LESS THAN (MAXVALUE)
       (SUBPARTITION p3_s1 TABLESPACE ts4,
        SUBPARTITION p3_s2 TABLESPACE ts5));
```

4.2.4.1.3 複数の表領域にまたがるローカル索引の作成

このトピックの例は、複数の表領域にまたがるローカル索引の作成方法を示します。

次の文で、索引セグメントが表領域ts7、ts8およびts9に分散している表にローカル索引を作成する例を示します。

```
CREATE INDEX employee_ix ON employees_range_hash(department_id)
     LOCAL STORE IN (ts7, ts8, ts9);
```

このローカル索引は、ベース表と同じ量のパーティションで構成されるように、ベース表と同一レベルでパーティション化されます。各索引パーティションには、対応するベース表のパーティションと同じ数のサブパーティションが含まれます。ベース表の指定されたサブパーティションに存在する行の索引エントリは、索引の対応するサブパーティションに格納されます。

4.2.4.2 コンポジット・レンジ - リスト・パーティション表の作成について

レンジ - リスト・コンポジット・パーティション表のレンジ・パーティションは、非コンポジット・レンジ・パーティション表と同様に記述されます。

この構成により、PARTITION句のオプションの副次句で、パーティション・セグメントに固有の物理属性およびその他の属性(表領域を含む)を指定できます。パーティション・レベルで上書きされない場合、パーティションは基礎となる表の属性を継承します。

SUBPARTITION句のリスト・サブパーティションの説明は、指定できる唯一の物理属性が表領域(オプション)であることを除き、非コンポジット・リスト・パーティションと同様に記述されます。サブパーティションは、パーティションの説明からその他すべての物理属性を継承します。

次の内容について説明します。

- コンポジット・レンジ - リスト・パーティション表の作成
- 表領域を指定するコンポジット・レンジ - リスト・パーティション表の作成

関連項目:

サブパーティション・テンプレートの使用によって、コンポジット・パーティション表の指定がどのように簡素化されるか方法を学習するには、「コンシューマ・グループを使用したユーザーに対するリソース数の制限時期」を参照してください

4.2.4.2.1 コンポジット・レンジ - リスト・パーティション表の作成

このトピックの例は、コンポジット・レンジ - リスト・パーティション表の作成方法を示します。

例4-24では、レンジ - リスト・パーティション化の使用方法を示します。この例では、製品の販売データを四半期ごとに追跡し、各四半期内で、指定された州ごとにグループ化しています。

行のパーティション化列の値が、特定のパーティション・レンジに含まれているかどうかを確認して、行がパーティションにマッピングされます。その後、記述子の値リストにサブパーティション列の値と一致する値が含まれるサブパーティションを特定して、そのパーティション内のサブパーティションに行がマッピングされます。たとえば、次のリストは、いくつかのサンプル行の挿入を示します。

- (10, 4532130, '23-Jan-1999', 8934.10, 'WA')はサブパーティションq1_1999_northwestにマッピングされます。
- (20, 5671621, '15-May-1999', 49021.21, 'OR')はサブパーティションq2_1999_northwestにマッピングされます。
- (30, 9977612, '07-Sep-1999', 30987.90, 'FL')はサブパーティションq3_1999_southeastにマッピングされます。
- (40, 9977612, '29-Nov-1999', 67891.45, 'TX')はサブパーティションq4_1999_southcentralにマッピングされます。
- (40, 4532130, '5-Jan-2000', 897231.55, 'TX')は表内のいずれのパーティションにもマッピングされないため、エラーが表示されます。
- (50, 5671621, '17-Dec-1999', 76123.35, 'CA')は表内のいずれのサブパーティションにもマッピングされないため、エラーが表示されます。

Live SQL:

関連する例をOracle Live SQLの*Oracle Live SQL: コンポジット・レンジ - リスト・パーティション表の作成*で参照して実行してください。

例4-24 コンポジット・レンジ - リスト・パーティション表の作成

```
CREATE TABLE quarterly_regional_sales
      (deptno number, item_no varchar2(20),
       txn_date date, txn_amount number, state varchar2(2))
  TABLESPACE ts4
  PARTITION BY RANGE (txn_date)
    SUBPARTITION BY LIST (state)
      (PARTITION q1_1999 VALUES LESS THAN (TO_DATE('1-APR-1999','DD-MON-YYYY'))
         (SUBPARTITION q1_1999_northwest VALUES ('OR', 'WA'),
          SUBPARTITION q1_1999_southwest VALUES ('AZ', 'UT', 'NM'),
          SUBPARTITION q1_1999_northeast VALUES ('NY', 'VM', 'NJ'),
          SUBPARTITION q1_1999_southeast VALUES ('FL', 'GA'),
          SUBPARTITION q1_1999_northcentral VALUES ('SD', 'WI'),
          SUBPARTITION q1_1999_southcentral VALUES ('OK', 'TX')
         ),
       PARTITION q2_1999 VALUES LESS THAN ( TO_DATE('1-JUL-1999','DD-MON-YYYY'))
         (SUBPARTITION q2_1999_northwest VALUES ('OR', 'WA'),
          SUBPARTITION q2_1999_southwest VALUES ('AZ', 'UT', 'NM'),
          SUBPARTITION q2_1999_northeast VALUES ('NY', 'VM', 'NJ'),
```

```
         SUBPARTITION q2_1999_southeast VALUES ('FL', 'GA'),
         SUBPARTITION q2_1999_northcentral VALUES ('SD', 'WI'),
         SUBPARTITION q2_1999_southcentral VALUES ('OK', 'TX')
        ),
       PARTITION q3_1999 VALUES LESS THAN (TO_DATE('1-OCT-1999','DD-MON-YYYY'))
        (SUBPARTITION q3_1999_northwest VALUES ('OR', 'WA'),
         SUBPARTITION q3_1999_southwest VALUES ('AZ', 'UT', 'NM'),
         SUBPARTITION q3_1999_northeast VALUES ('NY', 'VM', 'NJ'),
         SUBPARTITION q3_1999_southeast VALUES ('FL', 'GA'),
         SUBPARTITION q3_1999_northcentral VALUES ('SD', 'WI'),
         SUBPARTITION q3_1999_southcentral VALUES ('OK', 'TX')
        ),
       PARTITION q4_1999 VALUES LESS THAN ( TO_DATE('1-JAN-2000','DD-MON-YYYY'))
        (SUBPARTITION q4_1999_northwest VALUES ('OR', 'WA'),
         SUBPARTITION q4_1999_southwest VALUES ('AZ', 'UT', 'NM'),
         SUBPARTITION q4_1999_northeast VALUES ('NY', 'VM', 'NJ'),
         SUBPARTITION q4_1999_southeast VALUES ('FL', 'GA'),
         SUBPARTITION q4_1999_northcentral VALUES ('SD', 'WI'),
         SUBPARTITION q4_1999_southcentral VALUES ('OK', 'TX')
        )
     );
```

4.2.4.2.2 表領域を指定するコンポジット・レンジ - リスト・パーティション表の作成

このトピックの例は、表領域を指定するコンポジット・レンジ - リスト・パーティション表の作成方法を示します。

レンジ - リスト・パーティション表のパーティションは、そのデータがサブパーティションのセグメントに格納されるため、論理構造のみです。リスト・サブパーティションの特性は、リスト・パーティションと同様です。リスト・パーティション化にデフォルトのパーティションを指定する場合と同じように、デフォルトのサブパーティションを指定できます。

次の例では、パーティション・レベルおよびサブパーティション・レベルで表領域を指定する表を作成します。各パーティション内のサブパーティションの数は様々で、デフォルトのサブパーティションが指定されています。この例のサブパーティションの説明は次のようになります。

- すべてのサブパーティションで、表領域レベルのデフォルトから、表領域以外の物理属性が継承されます。これは、パーティションまたはサブパーティションに指定された唯一の物理属性が表領域であるためです。表レベルで物理属性が指定されていないため、表領域レベルのデフォルトがすべてのレベルで継承されます。
- tbs_4に格納され、その他のいずれのパーティションにもマッピングされていないすべての行を含むサブパーティションq1_othersを除き、パーティションq1_1999の最初の4つのサブパーティションは、すべてtbs_1に格納されます。
- パーティションq2_1999の6つのサブパーティションは、すべてtbs_2に格納されます。
- tbs_4に格納され、その他のいずれのパーティションにもマッピングされていないすべての行を含むサブパーティションq3_othersを除き、パーティションq3_1999の最初の2つのサブパーティションは、すべてtbs_3に格納されます。
- パーティションq4_1999にはサブパーティションの説明はありません。このため、デフォルトのサブパーティションが1つ作成され、tbs_4に格納されます。サブパーティション名は、SYS_SUBP*n*という書式でシステム生成されます。

```
CREATE TABLE sample_regional_sales
      (deptno number, item_no varchar2(20),
       txn_date date, txn_amount number, state varchar2(2))
  PARTITION BY RANGE (txn_date)
    SUBPARTITION BY LIST (state)
      (PARTITION q1_1999 VALUES LESS THAN (TO_DATE('1-APR-1999','DD-MON-YYYY'))
         TABLESPACE tbs_1
        (SUBPARTITION q1_1999_northwest VALUES ('OR', 'WA'),
         SUBPARTITION q1_1999_southwest VALUES ('AZ', 'UT', 'NM'),
```

```
         SUBPARTITION q1_1999_northeast VALUES ('NY', 'VM', 'NJ'),
         SUBPARTITION q1_1999_southeast VALUES ('FL', 'GA'),
         SUBPARTITION q1_others VALUES (DEFAULT) TABLESPACE tbs_4
        ),
      PARTITION q2_1999 VALUES LESS THAN ( TO_DATE('1-JUL-1999','DD-MON-YYYY'))
         TABLESPACE tbs_2
        (SUBPARTITION q2_1999_northwest VALUES ('OR', 'WA'),
         SUBPARTITION q2_1999_southwest VALUES ('AZ', 'UT', 'NM'),
         SUBPARTITION q2_1999_northeast VALUES ('NY', 'VM', 'NJ'),
         SUBPARTITION q2_1999_southeast VALUES ('FL', 'GA'),
         SUBPARTITION q2_1999_northcentral VALUES ('SD', 'WI'),
         SUBPARTITION q2_1999_southcentral VALUES ('OK', 'TX')
        ),
      PARTITION q3_1999 VALUES LESS THAN (TO_DATE('1-OCT-1999','DD-MON-YYYY'))
         TABLESPACE tbs_3
        (SUBPARTITION q3_1999_northwest VALUES ('OR', 'WA'),
         SUBPARTITION q3_1999_southwest VALUES ('AZ', 'UT', 'NM'),
         SUBPARTITION q3_others VALUES (DEFAULT) TABLESPACE tbs_4
        ),
      PARTITION q4_1999 VALUES LESS THAN ( TO_DATE('1-JAN-2000','DD-MON-YYYY'))
         TABLESPACE tbs_4
    );
```

4.2.4.3 コンポジット・レンジ - レンジ・パーティション表の作成

レンジ - レンジ・コンポジット・パーティション表のレンジ・パーティションは、非コンポジット・レンジ・パーティション表に似ています。

この構成により、PARTITION句のオプションの副次句で、パーティション・セグメントに固有の物理属性およびその他の属性(表領域を含む)を指定できます。パーティション・レベルで上書きされない場合、パーティションは基礎となる表の属性を継承します。

SUBPARTITION句のレンジ・サブパーティションの記述は、指定できる唯一の物理属性がオプションの表領域であることを除き、非コンポジット・レンジ・パーティションと似ています。サブパーティションは、パーティションの説明からその他すべての物理属性を継承します。

次の例では、レンジ - レンジ・パーティション化の使用方法を示します。この例では出荷を追跡しています。顧客とのサービス内容合意書には、すべての注文品は、注文が行われてから1か月以内に配送すると記載されています。注文には次のタイプがあります。

行のパーティション化列の値が、特定のパーティション・レンジに含まれているかどうかを確認して、行がパーティションにマッピングされます。その後、サブパーティション化列の値が特定のレンジ内に含まれているかどうかを確認して、そのパーティション内のサブパーティションに行がマッピングされます。たとえば、発注日が2006年9月で配送日が2006年10月28の出荷は、パーティションp06_oct_aに含まれます。

- e(早期): 注文が行われた次の月の中旬までに配送される注文。これらの注文は、顧客の予想より早く配送されます。
- a(合意どおり): 注文が行われてから1か月以内に配送される注文(ただし早期の注文ではありません)。
- l(遅延): 注文が行われた後、1か月を過ぎてから配送される注文。

```
CREATE TABLE shipments
( order_id      NUMBER NOT NULL
, order_date    DATE NOT NULL
, delivery_date DATE NOT NULL
, customer_id   NUMBER NOT NULL
, sales_amount  NUMBER NOT NULL
)
PARTITION BY RANGE (order_date)
SUBPARTITION BY RANGE (delivery_date)
```

```
( PARTITION p_2006_jul VALUES LESS THAN (TO_DATE('01-AUG-2006','dd-MON-yyyy'))
  ( SUBPARTITION p06_jul_e VALUES LESS THAN (TO_DATE('15-AUG-2006','dd-MON-yyyy'))
  , SUBPARTITION p06_jul_a VALUES LESS THAN (TO_DATE('01-SEP-2006','dd-MON-yyyy'))
  , SUBPARTITION p06_jul_l VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_2006_aug VALUES LESS THAN (TO_DATE('01-SEP-2006','dd-MON-yyyy'))
  ( SUBPARTITION p06_aug_e VALUES LESS THAN (TO_DATE('15-SEP-2006','dd-MON-yyyy'))
  , SUBPARTITION p06_aug_a VALUES LESS THAN (TO_DATE('01-OCT-2006','dd-MON-yyyy'))
  , SUBPARTITION p06_aug_l VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_2006_sep VALUES LESS THAN (TO_DATE('01-OCT-2006','dd-MON-yyyy'))
  ( SUBPARTITION p06_sep_e VALUES LESS THAN (TO_DATE('15-OCT-2006','dd-MON-yyyy'))
  , SUBPARTITION p06_sep_a VALUES LESS THAN (TO_DATE('01-NOV-2006','dd-MON-yyyy'))
  , SUBPARTITION p06_sep_l VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_2006_oct VALUES LESS THAN (TO_DATE('01-NOV-2006','dd-MON-yyyy'))
  ( SUBPARTITION p06_oct_e VALUES LESS THAN (TO_DATE('15-NOV-2006','dd-MON-yyyy'))
  , SUBPARTITION p06_oct_a VALUES LESS THAN (TO_DATE('01-DEC-2006','dd-MON-yyyy'))
  , SUBPARTITION p06_oct_l VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_2006_nov VALUES LESS THAN (TO_DATE('01-DEC-2006','dd-MON-yyyy'))
  ( SUBPARTITION p06_nov_e VALUES LESS THAN (TO_DATE('15-DEC-2006','dd-MON-yyyy'))
  , SUBPARTITION p06_nov_a VALUES LESS THAN (TO_DATE('01-JAN-2007','dd-MON-yyyy'))
  , SUBPARTITION p06_nov_l VALUES LESS THAN (MAXVALUE)
  )
, PARTITION p_2006_dec VALUES LESS THAN (TO_DATE('01-JAN-2007','dd-MON-yyyy'))
  ( SUBPARTITION p06_dec_e VALUES LESS THAN (TO_DATE('15-JAN-2007','dd-MON-yyyy'))
  , SUBPARTITION p06_dec_a VALUES LESS THAN (TO_DATE('01-FEB-2007','dd-MON-yyyy'))
  , SUBPARTITION p06_dec_l VALUES LESS THAN (MAXVALUE)
  )
);
```

関連項目:

サブパーティション・テンプレートの使用によって、コンポジット・パーティション表の指定がどのように簡素化されるか方法を学習するには、「コンシューマ・グループを使用したユーザーに対するリソース数の制限時期」を参照してください

4.2.5 コンポジット・パーティション表を説明するサブパーティション・テンプレートの指定

サブパーティション・テンプレートを使用して、コンポジット・パーティション表にサブパーティションを作成できます。

サブパーティション・テンプレートを使用すると、表の各パーティションにサブパーティション記述子を指定する必要がないため、サブパーティションの指定を簡略化できます。かわりに、テンプレートに一度のみサブパーティションを記述し、そのサブパーティション・テンプレートを表の各パーティションに適用します。時間隔 - *コンポジット・パーティション表の場合、時間隔パーティションのサブパーティションを定義するには、サブパーティション・テンプレートを使用する必要があります。

サブパーティション・テンプレートは、パーティションにサブパーティション記述子が指定されていない場合に適用されます。サブパーティション記述子が指定されている場合は、そのパーティションのサブパーティション・テンプレートのかわりに適用されます。サブパーティション・テンプレートが指定されておらず、パーティションにもサブパーティション記述子が指定されていない場合は、デフォルトのサブパーティションが1つ作成されます。

次の内容について説明します。

- * - ハッシュ・パーティション表へのサブパーティション・テンプレートの指定
- * - リスト・パーティション表へのサブパーティション・テンプレートの指定

4.2.5.1 * - ハッシュ・パーティション表へのサブパーティション・テンプレートの指定

レンジ - ハッシュ、時間隔 - ハッシュおよびリスト - ハッシュ・パーティション表の場合、サブパーティション・テンプレートでサブパーティションを詳細に指定することも、ハッシュ・サブパーティションの数のみを指定することも可能です。

例4-25では、サブパーティション・テンプレートを使用して、レンジ - ハッシュ・パーティション表を作成し、サブパーティションの名前と表領域を表示します。

この例では、次の説明を含む表を作成します。

- 各パーティションには、サブパーティション・テンプレートに記述されているように4つのサブパーティションがあります。
- 各サブパーティションには表領域が指定されています。サブパーティション・テンプレートで1つのサブパーティションに表領域が指定されている場合は、すべてのサブパーティションに表領域を1つ指定する必要があります。
- 時間隔 - *サブパーティション化を使用している場合以外、サブパーティションの名前は、次の書式でパーティション名とサブパーティション名を連結して生成されます。

  partition name_subpartition name

  時間隔 - *サブパーティション化の場合、サブパーティション名は次の書式でシステム生成されます。

  SYS_SUBP*n*

例4-25 サブパーティション・テンプレートを使用したレンジ - ハッシュ・パーティション表の作成

```
CREATE TABLE employees_sub_template (department_id NUMBER(4) NOT NULL,
             last_name VARCHAR2(25), job_id VARCHAR2(10))
     PARTITION BY RANGE(department_id) SUBPARTITION BY HASH(last_name)
     SUBPARTITION TEMPLATE
         (SUBPARTITION a TABLESPACE ts1,
          SUBPARTITION b TABLESPACE ts2,
          SUBPARTITION c TABLESPACE ts3,
          SUBPARTITION d TABLESPACE ts4
         )
    (PARTITION p1 VALUES LESS THAN (1000),
     PARTITION p2 VALUES LESS THAN (2000),
     PARTITION p3 VALUES LESS THAN (MAXVALUE)
    );

SQL> SELECT TABLESPACE_NAME, PARTITION_NAME, SUBPARTITION_NAME
  2  FROM DBA_TAB_SUBPARTITIONS WHERE TABLE_NAME='EMPLOYEEES_SUB_TEMPLATE'
  3  ORDER BY TABLESPACE_NAME;

TABLESPACE_NAME PARTITION_NAME  SUBPARTITION_NAME
--------------- --------------- ------------------
TS1             P1              P1_A
TS1             P2              P2_A
TS1             P3              P3_A
TS2             P1              P1_B
TS2             P2              P2_B
TS2             P3              P3_B
TS3             P1              P1_C
TS3             P2              P2_C
TS3             P3              P3_C
TS4             P1              P1_D
TS4             P2              P2_D
```

```
TS4             P3              P3_D

12 rows selected.
```

4.2.5.2 * - リスト・パーティション表へのサブパーティション・テンプレートの指定

- リスト・パーティション表の場合、サブパーティション・テンプレートではサブパーティションを詳細に記述できます。

例4-26では、レンジ - リスト・パーティション表について、サブパーティション・テンプレートを使用して表領域間でデータをストライプ化する方法を示します。この例では、表が作成されると、表のサブパーティションは縦にストライプ化されます。つまり、すべてのパーティションのサブパーティション*n*が同じ表領域にあります。

サブパーティション・テンプレートではなく、パーティション・レベルで表領域を指定した場合(パーティションq1_1999にtbs_1、パーティションq2_1999にtbs_2、パーティションq3_1999にtbs_3およびパーティションq4_1999にtbs_4など)、表は水平にストライプ化されます。すべてのサブパーティションは、所有パーティションの表領域に格納されます。

例4-26 サブパーティション・テンプレートを使用したレンジ - リスト・パーティション表の作成

```
CREATE TABLE stripe_regional_sales
             ( deptno number, item_no varchar2(20),
               txn_date date, txn_amount number, state varchar2(2))
   PARTITION BY RANGE (txn_date)
   SUBPARTITION BY LIST (state)
   SUBPARTITION TEMPLATE
      (SUBPARTITION northwest VALUES ('OR', 'WA') TABLESPACE tbs_1,
       SUBPARTITION southwest VALUES ('AZ', 'UT', 'NM') TABLESPACE tbs_2,
       SUBPARTITION northeast VALUES ('NY', 'VM', 'NJ') TABLESPACE tbs_3,
       SUBPARTITION southeast VALUES ('FL', 'GA') TABLESPACE tbs_4,
       SUBPARTITION midwest VALUES ('SD', 'WI') TABLESPACE tbs_5,
       SUBPARTITION south VALUES ('AL', 'AK') TABLESPACE tbs_6,
       SUBPARTITION others VALUES (DEFAULT ) TABLESPACE tbs_7
      )
  (PARTITION q1_1999 VALUES LESS THAN ( TO_DATE('01-APR-1999','DD-MON-YYYY')),
   PARTITION q2_1999 VALUES LESS THAN ( TO_DATE('01-JUL-1999','DD-MON-YYYY')),
   PARTITION q3_1999 VALUES LESS THAN ( TO_DATE('01-OCT-1999','DD-MON-YYYY')),
   PARTITION q4_1999 VALUES LESS THAN ( TO_DATE('1-JAN-2000','DD-MON-YYYY'))
  );
```

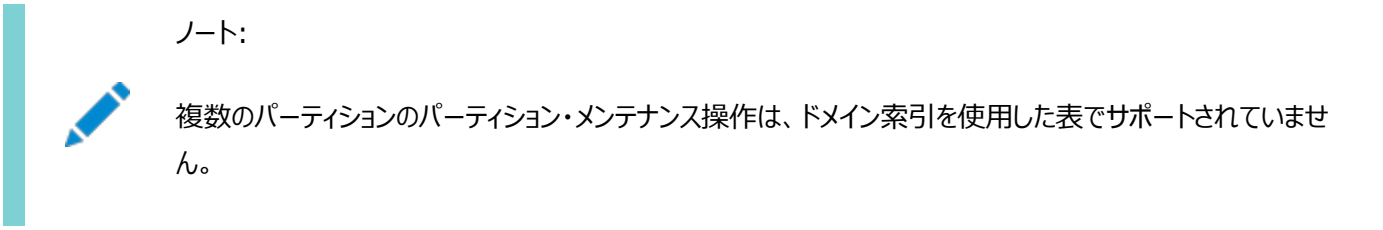
4.3 パーティションでサポートされているメンテナンス操作

パーティション、サブパーティションおよび索引パーティションで実行できる、様々なメンテナンス操作があります。

次の表およびトピックで、パーティション、サブパーティションおよび索引パーティションでサポートされているメンテナンス操作について説明します。

- 表4-1に、パーティション表およびコンポジット・パーティション表で実行できるパーティション・メンテナンス操作を示します。
- 表4-2に、コンポジット・パーティション表で実行できるサブパーティション・メンテナンス操作を示します。
- 表4-3に、索引パーティションで実行可能なメンテナンス操作を示し、どのタイプの索引(グローバルまたはローカル)を実行可能かを記載します。
- 索引の自動更新
- パーティションを削除および切り捨てる非同期グローバル索引メンテナンス
- サブパーティション・テンプレートの変更
- メンテナンス操作のフィルタ処理

表4-1および表4-2にパーティション化およびサブパーティション化の種類ごとに、メンテナンス操作の実行に使用されるALTER TABLE文の特定の句が示されます。



ノート:

複数のパーティションのパーティション・メンテナンス操作は、ドメイン索引を使用した表でサポートされていません。

表4-1 表パーティションに対するALTER TABLEメンテナンス操作

メンテナンス操作	レンジ・コンポジット・レンジ - *	時間隔コンポジット時間隔 - *	ハッシュ	リスト・コンポジット・リスト - *	参照
パーティションの追加。「パーティションおよびサブパーティションの追加について」を参照してください	ADD PARTITION、単一および複数のパーティション	該当なし	ADD PARTITION	ADD PARTITION、単一および複数のパーティション	該当なし。(これらの操作は、参照パーティション表では実行できません。これらの操作は、親表で実行するとすべての子表にカスケードされます。)
パーティションの結合。「パーティションおよびサブパーティションの結合について」を参照してくだ	該当なし	該当なし	COALESCE PARTITION	該当なし	N/A (これらの操作は、参照パーティション表では実行できません。これらの操作は、親

メンテナンス操作	レンジ・コンポジット・レンジ - *	時間隔コンポジット時間隔 - *	ハッシュ	リスト・コンポジット・リスト - *	参照
さい					表で実行するとすべての子表にカスケードされます。)
パーティションの削除。「パーティションおよびサブパーティションの削除について」を参照してください	DROP PARTITION、単一および複数のパーティション	DROP PARTITION、単一および複数のパーティション	該当なし	DROP PARTITION、単一および複数のパーティション	N/A (これらの操作は、参照パーティション表では実行できません。これらの操作は、親表で実行するとすべての子表にカスケードされます。)
パーティションの交換。「パーティションおよびサブパーティションの交換について」を参照してください	EXCHANGE PARTITION	EXCHANGE PARTITION	EXCHANGE PARTITION	EXCHANGE PARTITION	EXCHANGE PARTITION
パーティションのマージ。「パーティションおよびサブパーティションのマージについて」を参照してください	MERGE PARTITIONS、単一および複数のパーティション	MERGE PARTITIONS、単一および複数のパーティション	該当なし	MERGE PARTITIONS、単一および複数のパーティション	N/A (これらの操作は、参照パーティション表では実行できません。これらの操作は、親表で実行するとすべての子表にカスケードされます。)
デフォルトの属性の変更について	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES
パーティションの実際の属性の変更について	MODIFY PARTITION	MODIFY PARTITION	MODIFY PARTITION	MODIFY PARTITION	MODIFY PARTITION
リスト・パーティションの変更について: 値の追加	該当なし	該当なし	該当なし	MODIFY PARTITION ADD VALUES	該当なし
リスト・パーティションの変更について: 値の削	該当なし	該当なし	該当なし	MODIFY PARTITION DROP	該当なし

メンテナンス操作	レンジ・コンポジット・レンジ - *	時間隔コンポジット時間隔 - *	ハッシュ	リスト・コンポジット・リスト - *	参照
除				VALUES	
パーティションの移動。「パーティションおよびサブパーティションの移動について」を参照してください	MOVE SUBPARTITION	MOVE SUBPARTITION	MOVE PARTITION	MOVE SUBPARTITION	MOVE PARTITION
パーティション名の変更。「パーティションおよびサブパーティション名の変更について」を参照してください	RENAME PARTITION	RENAME PARTITION	RENAME PARTITION	RENAME PARTITION	RENAME PARTITION
パーティションの分割。「パーティションおよびサブパーティションの分割について」を参照してください	SPLIT PARTITION、単一および複数のパーティション	SPLIT PARTITION、単一および複数のパーティション	該当なし	SPLIT PARTITION、単一および複数のパーティション	N/A (これらの操作は、参照パーティション表では実行できません。これらの操作は、親表で実行するとすべての子表にカスケードされます。)
パーティションの切捨て。「パーティションおよびサブパーティションの切捨てについて」を参照してください	TRUNCATE PARTITION、単一および複数のパーティション	TRUNCATE PARTITION、単一および複数のパーティション	TRUNCATE PARTITION、単一および複数のパーティション	TRUNCATE PARTITION、単一および複数のパーティション	TRUNCATE PARTITION、単一および複数のパーティション

表4-2 表サブパーティションに対するALTER TABLEメンテナンス操作

メンテナンス操作	コンポジット* - レンジ	コンポジット* - ハッシュ	コンポジット* - リスト
サブパーティションの追加。「パーティションおよびサブパーティションの追加について」を参照してください	MODIFY PARTITION ADD SUBPARTITION、単一および複数のサブパーティション	MODIFY PARTITION ADD SUBPARTITION	MODIFY PARTITION ADD SUBPARTITION、単一および複数のサブパーティション
サブパーティションの結合。「パーティションおよびサブパーティションの結合について」を参照してください	該当なし	MODIFY PARTITION COALESCE SUBPARTITION	該当なし

メンテナンス操作	コンポジット* - レンジ	コンポジット* - ハッシュ	コンポジット* - リスト
サブパーティションの削除。「パーティションおよびサブパーティションの削除について」を参照してください	DROP SUBPARTITION、単一および複数のサブパーティション	該当なし	DROP SUBPARTITION、単一および複数のサブパーティション
サブパーティションの交換。「パーティションおよびサブパーティションの交換について」を参照してください	EXCHANGE SUBPARTITION	該当なし	EXCHANGE SUBPARTITION
サブパーティションのマージ。「パーティションおよびサブパーティションのマージについて」を参照してください	MERGE SUBPARTITIONS、単一および複数のサブパーティション	該当なし	MERGE SUBPARTITIONS、単一および複数のサブパーティション
デフォルトの属性の変更について	MODIFY DEFAULT ATTRIBUTES FOR PARTITION	MODIFY DEFAULT ATTRIBUTES FOR PARTITION	MODIFY DEFAULT ATTRIBUTES FOR PARTITION
サブパーティションの実際の属性の変更。「パーティションの実際の属性の変更について」を参照してください	MODIFY SUBPARTITION	MODIFY SUBPARTITION	MODIFY SUBPARTITION
リスト・サブパーティションの変更。「リスト・パーティションの変更について: 値の追加」を参照してください	該当なし	該当なし	MODIFY SUBPARTITION ADD VALUES
リスト・サブパーティションの変更。「リスト・パーティションの変更について: 値の削除」を参照してください	該当なし	該当なし	MODIFY SUBPARTITION DROP VALUES
サブパーティション・テンプレートの変更	SET SUBPARTITION TEMPLATE	SET SUBPARTITION TEMPLATE	SET SUBPARTITION TEMPLATE
サブパーティションの移動。「パーティションおよびサブパーティションの移動について」を参照してください	MOVE SUBPARTITION	MOVE SUBPARTITION	MOVE SUBPARTITION
サブパーティション名の変更。「パーティションおよびサブパーティション名の変更について」を参照してください	RENAME SUBPARTITION	RENAME SUBPARTITION	RENAME SUBPARTITION
サブパーティションの分割。「パーティションおよびサブパーティションの分	SPLIT SUBPARTITION、単一および複数のサブパーティ	該当なし	SPLIT SUBPARTITION、単一および複数のサブパー

メンテナンス操作	コンポジット* - レンジ	コンポジット* - ハッシュ	コンポジット* - リスト
割について」を参照してください	ション		ティション
サブパーティションの切捨て。「パーティションおよびサブパーティションの切捨てについて」を参照してください	TRUNCATE SUBPARTITION、単一および複数のサブパーティション	TRUNCATE SUBPARTITION、単一および複数のサブパーティション	TRUNCATE SUBPARTITION、単一および複数のサブパーティション

ノート:

ビットマップ索引のあるパーティション表や、現在圧縮されていないパーティションのみを含むパーティション表に、圧縮されたパーティションを組み込むために最初に表圧縮を使用する場合は、次の手順を実行する必要があります。

- 既存のすべてのビットマップ索引およびビットマップ索引パーティションを削除するか、UNUSABLE とマークします。
- 表圧縮属性を設定します。
- 索引を再作成します。

これらの処理は、パーティションにデータが含まれているかどうかには関係なく、圧縮されたパーティションを組み込む操作です。

これは、B ツリー索引を含むパーティション表、またはパーティション索引構成表には適用されません。

表4-3に索引パーティションで実行可能なメンテナンス操作を示し、どのタイプの索引(グローバルまたはローカル)で実行可能かを記載します。メンテナンス操作に使用するALTER INDEX句を表示します。

グローバル索引は、基礎となる表の構造を反映しません。パーティション化されている場合は、レンジまたはハッシュでパーティション化できます。

ローカル索引では基礎となる表の構造が反映されるため、表パーティションおよびサブパーティションがメンテナンス・アクティビティの影響を受けると、パーティション化も自動的にメンテナンスされます。そのため、ローカル索引でのパーティション・メンテナンスの必要性は低く、オプションの数も少なくなっています。

表4-3 索引パーティションに対するALTER INDEXメンテナンス操作

		索引パーティション化のタイプ		
メンテナンス操作	索引のタイプ	レンジ	ハッシュおよびリスト	コンポジット
索引パーティションの追加	グローバル	–	ADD PARTITION(ハッシュのみ)	-
索引パーティションの追加	ローカル	該当なし	該当なし	該当なし

		索引パーティション化のタイプ		
メンテナンス操作	索引のタイプ	レンジ	ハッシュおよびリスト	コンポジット
索引パーティションの削除	グローバル	DROP PARTITION	-	-
索引パーティションの削除	ローカル	該当なし	該当なし	該当なし
索引パーティションのデフォルト属性の変更	グローバル	MODIFY DEFAULT ATTRIBUTES	-	-
索引パーティションのデフォルト属性の変更	ローカル	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES MODIFY DEFAULT ATTRIBUTES FOR PARTITION
索引パーティションの実際の属性の変更	グローバル	MODIFY PARTITION	-	-
索引パーティションの実際の属性の変更	ローカル	MODIFY PARTITION	MODIFY PARTITION	MODIFY PARTITION MODIFY SUBPARTITION
索引パーティションの再作成について	グローバル	REBUILD PARTITION	-	-
索引パーティションの再作成について	ローカル	REBUILD PARTITION	REBUILD PARTITION	REBUILD SUBPARTITION
索引パーティション名の変更について	グローバル	RENAME PARTITION	-	-
索引パーティション名の変更について	ローカル	RENAME PARTITION	RENAME PARTITION	RENAME PARTITION RENAME SUBPARTITION
索引パーティションの分割	グローバル	SPLIT PARTITION	-	-
索引パーティションの分割	ローカル	該当なし	該当なし	該当なし

4.3.1 索引の自動更新

パーティション表および索引に対する個々のメンテナンス操作を説明する前に、ALTER TABLE文に指定可能なUPDATE INDEXES句の影響を説明します。

デフォルトでは、パーティション表に対する表メンテナンス操作の多くにより、対応する索引または索引パーティションが無効化(UNUSABLEとマーク)されます。索引全体または各パーティション(グローバル索引の場合)を再作成する必要があります。メンテナンス操作で、ALTER TABLE文にUPDATE INDEXESを指定すると、データベースによりこのデフォルトの動作が上書きされます。この句を指定すると、メンテナンス操作のDDL文の実行時に、データベースにより索引が更新されます。これには、次の利点があります。

- 索引が、ベース表操作とともに更新されます。後で更新して、個別に索引を再作成する必要はありません。
- グローバル索引はUNUSABLEとマークされないため、高い可用性を備えています。パーティションDDLの実行中でもこれらの索引は使用可能なままで、表内の影響を受けないパーティションにアクセスできます。
- すべての無効な索引を再作成するために、名前を調べる必要はありません。

ローカル索引のオプションの句で、更新されたローカル索引やそのパーティションに物理特性および記憶域特性を指定できます。

- 各ローカル索引のそれぞれのパーティションに物理属性、表領域記憶域およびロギングを指定できます。または、PARTITIONキーワードのみを指定して、次のようにデータベースによるパーティション属性の更新を行うことも可能です。
    - 単一の表パーティションに対する操作(MOVE PARTITIONやSPLIT PARTITIONなど)では、対応する索引パーティションは影響を受ける表パーティションの属性を継承します。データベースでは新しい索引パーティションの名前は生成されないため、この操作により作成される新しい索引パーティションは、対応する新しい表パーティションから名前を継承します。
    - MERGE PARTITION操作の場合、この操作によって作成されたローカル索引パーティションは、作成された表パーティションの名前とローカル索引の属性を継承します。
- コンポジット・パーティション索引では、各サブパーティションに表領域記憶域を指定できます。

次の操作では、UPDATE INDEXES句がサポートされています。

- ADD PARTITION | SUBPARTITION
- COALESCE PARTITION | SUBPARTITION
- DROP PARTITION | SUBPARTITION
- EXCHANGE PARTITION | SUBPARTITION
- MERGE PARTITION | SUBPARTITION
- MOVE PARTITION | SUBPARTITION
- SPLIT PARTITION | SUBPARTITION
- TRUNCATE PARTITION | SUBPARTITION

SKIP_UNUSABLE_INDEXES初期化パラメータ

SKIP_UNUSABLE_INDEXESは、デフォルト値がTRUEの初期化パラメータです。この設定により、UNUSABLEとマークされた索引および索引パーティションのエラー・レポートが無効化されます。使用できない要素を避けるために、データベースが別の実行計画を選択しないようにするには、このパラメータをFALSEに設定する必要があります。

索引を自動更新する際の考慮事項

UPDATE INDEXESを指定する際は、次の影響に注意してください。

- 事前にUNUSABLEとマークされていた索引が更新されるため、パーティションDDL文の実行時間が長くなる可能性があります。ただし、この増加した時間を、索引を更新せずにDDLを実行し、すべての索引を再作成する場合にかかる時間と比較する必要があります。パーティションのサイズが表サイズの5%に満たない場合は、経験則から言って、索引を更

新する方が短時間ですみます。

- EXCHANGE操作が高速な操作ではなくなりました。こちらも、DDLを実行してすべての索引を再作成する場合にかかる時間と比較する必要があります。
- グローバル索引を含む表を更新する際の考慮事項は、次のとおりです。
  - 索引はその場で更新されます。索引の更新は記録され、REDOおよびUNDOレコードが生成されます。反対に、グローバル索引全体を再作成する場合は、NOLOGGINGモードで実行できます。
  - 索引全体を手動で再作成すると、領域がより有効に使用されて圧縮されるため、より効率的な索引が作成されます。
- UPDATE INDEXES句は、索引構成表ではサポートされていません。ただし、DROP PARTITION、TRUNCATE PARTITIONおよびEXCHANGE PARTITION操作でUPDATE GLOBAL INDEXES句を使用して、索引構成表のグローバル索引を使用可能なままにできます。前述のリストのその他の操作では、索引構成表のグローバル索引は使用可能なままになります。また、索引構成表のローカル索引パーティションは、MOVE PARTITION操作後も使用可能なままです。

関連項目:

索引の更新の構文のALTER TABLEの*update_all_indexes_clause*の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

4.3.2 パーティションを削除および切り捨てる非同期グローバル索引メンテナンス

パーティション・メンテナンス操作DROP PARTITIONおよびTRUNCATE PARTITIONは、メタデータのみの索引メンテナンスを実行して最適化されます。

DROPおよびTRUNCATEの非同期グローバル索引メンテナンスはデフォルトで実行されますが、UPDATE INDEXES句が下位互換性のために引き続き必要です。

次のリストは、非同期グローバル索引メンテナンスの制限事項をまとめています。

- ヒープ表にのみ実行されます
- オブジェクト型を含む表はサポートされません
- ドメイン索引を含む表はサポートされません
- ユーザーSYSには実行されません

索引のメンテナンス操作は、自動スケジューラ・ジョブSYS.PMO_DEFERRED_GIDX_MAINT_JOBを使用して実行し、すべてのグローバル索引をクリーンアップできます。デフォルトでは、このジョブは定期的に実行するようスケジュールされます。索引を事前にクリーンアップする場合、DBMS_SCHEDULER.RUN_JOBを使用していつでもこのジョブを実行できます。ジョブを変更して、特定の要件に基づくスケジュールで実行することもできます。ジョブを削除しないことをお薦めします。

次のオプションのいずれかを使用して、メンテナンスを必要とする索引のクリーンアップを強制することもできます。

- DBMS_PART.CLEANUP_GIDX - このPL/SQLプロシージャは、クリーンアップを必要とする可能性があるシステムのグローバル索引のリストを収集し、索引をクリーンな状態にリストアするために必要な操作を実行します。
- ALTER INDEX REBUILD [PARTITION] – このSQL文は、Oracle Database 12cリリース1 (12.1)より前のリリース

で実行されるように、索引全体または索引パーティションを再作成します。結果の索引(パーティション)は、古いエントリを含みません。

- ALTER INDEX [PARTITION] COALESCE CLEANUP - このSQL文は、索引ブロックの親がないエントリをクリーンアップします。

関連項目:

Oracle Schedulerによるジョブの管理の詳細は、『Oracle Database管理者ガイド』を参照してください

4.3.3 サブパーティション・テンプレートの変更

新しいサブパーティション・テンプレートと置き換えることで、コンポジット・パーティション表のサブパーティション・テンプレートを変更できます。

サブパーティション・テンプレートを使用する後続の操作(ADD PARTITIONまたはMERGE PARTITIONSなど)で、新しいサブパーティション・テンプレートが使用されます。既存のサブパーティションは変更されません。

時間隔 - *コンポジット・パーティション表のサブパーティション・テンプレートを変更すると、作成されていない時間隔パーティションに新しいサブパーティション・テンプレートが使用されます。

新しいサブパーティション・テンプレートを指定するには、ALTER TABLE SET SUBPARTITION TEMPLATE文を使用します。例:

```
ALTER TABLE employees_sub_template
   SET SUBPARTITION TEMPLATE
         (SUBPARTITION e TABLESPACE ts1,
          SUBPARTITION f TABLESPACE ts2,
          SUBPARTITION g TABLESPACE ts3,
          SUBPARTITION h TABLESPACE ts4
         );
```

空のリストを指定することで、サブパーティション・テンプレートを削除できます。

```
ALTER TABLE employees_sub_template
   SET SUBPARTITION TEMPLATE ( );
```

4.3.4 メンテナンス操作のフィルタ処理

パーティション・メンテナンス操作は、データのフィルタ処理の追加をサポートするため、パーティションとデータ・メンテナンスを組み合せることができます。

フィルタ処理されたパーティション・メンテナンス操作は、パーティション・メンテナンスの一部としてデータのフィルタ処理を満たすデータのみを保持します。データのフィルタ処理の機能は、MOVE PARTITION、MERGE PARTITIONおよびSPLIT PARTITIONに適用されます。

例4-27に、ALTER TABLE文を使用してオープンでない注文(クローズ済注文)をすべて削除すると同時にパーティションを移動する方法を示します。

パーティション表にはフィルタ処理述語が必要です。また、オンラインで実行できるすべてのパーティション・メンテナンス操作(MOVEおよびSPLIT)は、フィルタ処理されたパーティション・メンテナンス操作として実行することもできます。ONLINEを指定すると、メンテナンス対象のパーティションに対するDML操作が許可されます。

オンライン・モードで実行されるフィルタ処理されたパーティション・メンテナンス操作は、同時進行中のDML操作にフィルタ述語を

使用しません。フィルタ条件は、パーティション・メンテナンス操作の最初に一度だけ適用されます。そのため、後続のDMLは成功しますが、フィルタ処理の観点からは無視されます。どのようなDML操作であっても、パーティション・メンテナンスの開始時にフィルタ条件に一致しないレコードは保持されません。新しく挿入されたレコードは、パーティション・メンテナンス操作のフィルタ条件を満たすかどうかに関係なく、パーティション・キー基準に一致すると挿入されます。フィルタ条件はパーティション表自体に制限され、結合や副問合せ式など、他の表への参照は許可されません。

キーワードONLINEが例4-27のSQL文で指定されている、次のシナリオを考えてみます。

- パーティション・メンテナンス操作の開始後にstatus='open'に更新されるパーティションq1_2016の既存の注文レコードが、このパーティションに保持されていません。
- パーティション・メンテナンス操作の開始後およびパーティション・メンテナンス操作の進行中に、status='closed'の新しい注文レコードをパーティションq1_2016に挿入できます。

例4-27 メンテナンス操作実行時のフィルタ処理句の使用

```
ALTER TABLE orders_move_part
  MOVE PARTITION q1_2016 TABLESPACE open_orders COMPRESS ONLINE
  INCLUDING ROWS WHERE order_state = 'open';
```

関連項目:

パーティション表や索引を作成および変更するためのパーティション化句の正確な構文、その使用に関する制限、および表の作成や変更に必要な特定の権限の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

4.4 パーティション表および索引のメンテナンス操作

パーティション表および索引で実行できる、様々なメンテナンス操作があります。

次の各トピックで、表と索引の両方に対するパーティションおよびサブパーティション・メンテナンスを実行する操作について説明します。

- パーティションおよびサブパーティションの追加について
- パーティションおよびサブパーティションの結合について
- パーティションおよびサブパーティションの削除について
- パーティションおよびサブパーティションの交換について
- パーティションおよびサブパーティションのマージについて
- 表、パーティションおよびサブパーティションの属性の変更について
- リスト・パーティションの変更について
- パーティション化戦略の変更について
- パーティションおよびサブパーティションの移動について
- 索引パーティションの再作成について
- パーティションおよびサブパーティション名の変更について
- パーティションおよびサブパーティションの分割について
- パーティションおよびサブパーティションの切捨てについて

ノート:

メンテナンス操作に影響される索引または索引パーティションの使用性が説明されている箇所では、次のことを考慮してください。

- UNUSABLE とマークできるのは、空ではない索引と索引パーティションのみです。空の場合には、USABLE/UNUSABLE のステータスは変更されません。
- ステータスが USABLE の索引または索引パーティションは、後続の DML で更新されます。

関連項目:

- 表の管理の詳細は、『Oracle Database管理者ガイド』を参照してください
- パーティション表や索引を変更するためのパーティション化句の正確な構文、その使用に関する制限、および表の作成や変更に必要な特定の権限の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

4.4.1 パーティションおよびサブパーティションの追加について

この項では、パーティション表に新しいパーティションを手動で追加する方法、および多くのパーティション索引に明示的にパーティションを追加できない理由を説明します。

この項では、次の項目について説明します。

- レンジ・パーティション表へのパーティションの追加
- ハッシュ・パーティション表へのパーティションの追加
- リスト・パーティション表へのパーティションの追加
- 時間隔パーティション表へのパーティションの追加
- コンポジット* - ハッシュ・パーティション表へのパーティションの追加について
- コンポジット* - リスト・パーティション表へのパーティションの追加について
- コンポジット* - レンジ・パーティション表へのパーティションの追加について
- 参照パーティション表へのパーティションまたはサブパーティションの追加について
- 索引パーティションの追加
- 複数のパーティションの追加

4.4.1.1 レンジ・パーティション表へのパーティションの追加

表の最後の既存パーティションの後、あるいは表の最初または表の中間にパーティションを追加できます。

ハイエンド(既存の最後のパーティションの後ろのポイント)に新しいパーティションを追加するには、ALTER TABLE ADD PARTITION文を使用します。表の最初または中間にパーティションを追加するには、SPLIT PARTITION句を使用します。

たとえば、前の12か月分に加え、今月のデータも含む表salesがあるとします。1999年1月1日を基準に、表領域tsxに格納される1月用のパーティションを追加します。

```
ALTER TABLE sales
      ADD PARTITION jan99 VALUES LESS THAN ( '01-FEB-1999' )
      TABLESPACE tsx;
```

レンジ・パーティション表に関連付けられているローカルおよびグローバル索引は、使用可能なままです。

4.4.1.2 ハッシュ・パーティション表へのパーティションの追加

ハッシュ・パーティション表にパーティションを追加すると、データベースにより、新しいパーティションに既存のパーティション(データベースが選択)から再度ハッシュされた行が、ハッシュ関数で決定されたとおりに移入されます。

その結果、表にデータが含まれる場合、ハッシュ・パーティションの追加に時間がかかる場合があります。

次の文では、表scubagearにハッシュ・パーティションを追加する2つの方法を示します。最初の文では、パーティション名がシステム生成される新しいハッシュ・パーティションが追加され、デフォルトの表領域に配置されます。2番目の文でも、新しいハッシュ・パーティションが追加されますが、そのパーティションには明示的にp_namedという名前が付けられ、表領域gear5に作成されます。

```
ALTER TABLE scubagear ADD PARTITION;

ALTER TABLE scubagear
      ADD PARTITION p_named TABLESPACE gear5;
```

次の表で説明するように、索引はUNUSABLEとマークされます。

表のタイプ	索引の動作
通常(ヒープ)	ALTER TABLE 文の一部に UPDATE INDEXES を指定しない場合: ● 新しいパーティションおよび行の再配布元の既存のパーティションのローカル索引が UNUSABLE とマークされ、再作成する必要があります。 ● すべてのグローバル索引またはパーティション・グローバル索引のすべてのパーティションが UNUSABLE とマークされ、再作成する必要があります。
索引構成	● ローカル索引の場合、動作はヒープ表と同じです。 ● すべてのグローバル索引は使用可能なままです。

4.4.1.3 リスト・パーティション表へのパーティションの追加

このトピックの例は、パーティションをリスト - パーティション表に追加する方法を示します。

次の文で、リスト・パーティション表に新しいパーティションを追加する方法を示します。この例では、追加されるパーティションに物理属性およびNOLOGGINGが指定されています。

```
ALTER TABLE q1_sales_by_region
   ADD PARTITION q1_nonmainland VALUES ('HI', 'PR')
      STORAGE (INITIAL 20K NEXT 20K) TABLESPACE tbs_3
      NOLOGGING;
```

追加するパーティションを説明する一連のリテラル値は、表のどのパーティションにも存在していない必要があります。

デフォルトのパーティションがあるリスト・パーティション表にはパーティションを追加できませんが、デフォルトのパーティションを分割することはできます。分割することで、指定した値で定義された新しいパーティションと、デフォルトのパーティションとして残る2つ目のパーティションを効率的に作成できます。

リスト・パーティション表に関連付けられているローカルおよびグローバル索引は、使用可能なままです。

4.4.1.4 時間隔パーティション表へのパーティションの追加

時間隔パーティション表には、明示的にパーティションを追加できません。ある時間隔のデータが挿入されると、データベースによってその時間隔のパーティションが自動的に作成されます。

ただし、データ・ディクショナリのマテリアライズ化されていない時間隔パーティション表のパーティションの交換、つまり時間隔定義を超えてデータ・ディクショナリの明示的なエントリを持つ場合、ALTER TABLE LOCK PARTITIONコマンドを使用して、パーティションを手動でマテリアライズ化する必要があります。

後続のパーティションの時間隔を変更するには、ALTER TABLE文のSET INTERVAL句を使用します。SET INTERVAL句は、既存の時間隔パーティションをレンジ・パーティションに変換し、定義済レンジ・パーティションの上限を決定し、その上限を超えるデータのために必要な場合は指定された時間隔のパーティションを自動的に作成します。副次的作用として、時間隔パーティション表にはMAXVALUESの表記がありません。

既存のレンジ・パーティションまたはレンジ - *コンポジット・パーティション表を、時間隔または時間隔 - *パーティション表に移行するには、SET INTERVAL句も使用します。後続の時間隔パーティションの作成を無効化し、効率的にレンジ・パーティション

表に戻すには、SET INTERVAL句に空の値を使用します。作成された時間隔パーティションは、現在の上限値を使用してレンジ・パーティションに変換されます。

日付レンジの時間隔を増加するには、新しい時間隔の関連する上限値を指定していることを確認する必要があります。たとえば、1日単位の時間隔パーティション表トランザクションの時間隔パーティションの上限値が2007年1月30日で、月単位の時間隔パーティションに変更する場合、次の文ではエラーが発生します。

```
ALTER TABLE transactions SET INTERVAL (NUMTOYMINTERVAL(1,'MONTH');

ORA-14767: Cannot specify this interval with existing high bounds
```

月単位の時間隔に正常に変更するには、上限値が2007年2月1日の日付単位の別のパーティションを作成する必要があります。

```
LOCK TABLE transactions PARTITION FOR(TO_DATE('31-JAN-2007','dd-MON-yyyy'))
   IN SHARE MODE;

ALTER TABLE transactions SET INTERVAL (NUMTOYMINTERVAL(1,'MONTH');
```

時間隔パーティション表の下位パーティションはレンジ・パーティションです。時間隔パーティション表のレンジ部分にさらにパーティションを追加するには、レンジ・パーティションを分割します。

transactions表の時間隔パーティション化を無効化するには、次の文を使用します。

```
ALTER TABLE transactions SET INTERVAL ();
```

4.4.1.5 コンポジット* - ハッシュ・パーティション表へのパーティションの追加について

パーティションは、パーティション・レベルとハッシュ・サブパーティション・レベルの両方で追加できます。

- * - ハッシュ・パーティション表へのパーティションの追加
- * - ハッシュ・パーティション表へのサブパーティションの追加

4.4.1.5.1 * - ハッシュ・パーティション表へのパーティションの追加

このトピックの例は、新規パーティションを[レンジ | リスト | 時間隔] - ハッシュ・パーティション表に追加する方法を示します。

時間隔 - ハッシュ・パーティション表の場合、時間隔パーティションは自動的に作成されます。指定した数のサブパーティションを追加できるSUBPARTITIONS句を指定することも、特定のサブパーティションに名前を付けるSUBPARTITION句を指定することも可能です。SUBPARTITIONSまたはSUBPARTITION句が指定されていない場合、パーティションはサブパーティションの表レベルのデフォルトを継承します。時間隔 - ハッシュ・パーティション表の場合、マテリアライズ化されたレンジまたは時間隔パーティションに追加できるのはサブパーティションのみです。

この例では、2000年の第1四半期のデータが移入されるレンジ・パーティション q1_2000を、レンジ - ハッシュ・パーティション表salesに追加しています。表領域tbs5に格納される8つのサブパーティションがあります。サブパーティションを、表圧縮を使用するように明示的に設定することはできません。サブパーティションはパーティション・レベルから圧縮属性を継承し、この例では圧縮された形式で格納されます。

```
ALTER TABLE sales ADD PARTITION q1_2000
      VALUES LESS THAN (2000, 04, 01) COMPRESS
      SUBPARTITIONS 8 STORE IN tbs5;
```

4.4.1.5.2 * - ハッシュ・パーティション表へのサブパーティションの追加

ハッシュ・サブパーティションを[レンジ | リスト | 時間隔] - ハッシュ・パーティション表に追加するには、ALTER TABLE文の

MODIFY PARTITION ADD SUBPARTITION句を使用します。

新しく追加されたサブパーティションには、同じパーティションのその他のサブパーティションから再度ハッシュされた行が、ハッシュ関数で決定されたとおりに移入されます。時間隔 - ハッシュ・パーティション表の場合、マテリアライズ化されたレンジまたは時間隔パーティションに追加できるのはサブパーティションのみです。

次の例では、表divingのレンジ・パーティションlocations_usに、表領域us1に格納される新しいハッシュ・サブパーティションus_loc5が追加されています。

```
ALTER TABLE diving MODIFY PARTITION locations_us
      ADD SUBPARTITION us_locs5 TABLESPACE us1;
```

UPDATE INDEXESを指定しない場合は、追加および再度ハッシュされたサブパーティションに対応する索引サブパーティションを再作成する必要があります。

4.4.1.6 コンポジット* - リスト・パーティション表へのパーティションの追加について

パーティションは、パーティション・レベルとリスト・サブパーティション・レベルの両方で追加できます。

- * - リスト・パーティション表へのパーティションの追加
- * - リスト・パーティション表へのサブパーティションの追加

4.4.1.6.1 * - リスト・パーティション表へのパーティションの追加

このトピックの例は、新規パーティションを[レンジ | リスト | 時間隔] - リスト・パーティション表に追加する方法を示します。

特定の時間隔のデータが挿入されると、データベースにより時間隔パーティションが自動的に作成されます。サブパーティションに名前を付け、値リストを提供するには、SUBPARTITION句を指定できます。SUBPARTITION句が指定されていない場合、パーティションはサブパーティション・テンプレートを継承します。サブパーティション・テンプレートがない場合には、単一のデフォルトのサブパーティションが作成されます。

例4-28の文では、レンジ - リスト・メソッドでパーティション化されたquarterly_regional_sales表に、新しいパーティションを追加しています。指定されていないパーティションでは表レベルのデフォルトが継承されますが、この新しいパーティションにはいくつかの新しい物理属性が指定されています。

例4-28 レンジ - リスト・パーティション表へのパーティションの追加

```
ALTER TABLE quarterly_regional_sales
   ADD PARTITION q1_2000 VALUES LESS THAN (TO_DATE('1-APR-2000','DD-MON-YYYY'))
      STORAGE (INITIAL 20K NEXT 20K) TABLESPACE ts3 NOLOGGING
         (
          SUBPARTITION q1_2000_northwest VALUES ('OR', 'WA'),
          SUBPARTITION q1_2000_southwest VALUES ('AZ', 'UT', 'NM'),
          SUBPARTITION q1_2000_northeast VALUES ('NY', 'VM', 'NJ'),
          SUBPARTITION q1_2000_southeast VALUES ('FL', 'GA'),
          SUBPARTITION q1_2000_northcentral VALUES ('SD', 'WI'),
          SUBPARTITION q1_2000_southcentral VALUES ('OK', 'TX')
         );
```

4.4.1.6.2 * - リスト・パーティション表へのサブパーティションの追加

リスト・サブパーティションを[レンジ | リスト | 時間隔] - リスト・パーティション表に追加するには、ALTER TABLE文のMODIFY PARTITION ADD SUBPARTITION句を使用します。

時間隔 - リスト・パーティション表の場合、マテリアライズ化されたレンジまたは時間隔パーティションに追加できるのはサブパーティションのみです。

次の文では、レンジ - リスト・パーティション表quarterly_regional_salesの一連の既存のサブパーティションに、新しいサブパーティションを追加します。新しいサブパーティションは、表領域ts2に作成されます。

```
ALTER TABLE quarterly_regional_sales
   MODIFY PARTITION q1_1999
      ADD SUBPARTITION q1_1999_south
         VALUES ('AR','MS','AL') tablespace ts2;
```

4.4.1.7 コンポジット* - レンジ・パーティション表へのパーティションの追加について

パーティションは、パーティション・レベルとレンジ・サブパーティション・レベルの両方で追加できます。

- * - レンジ・パーティション表へのパーティションの追加
- * - レンジ・パーティション表へのサブパーティションの追加

4.4.1.7.1 * - レンジ・パーティション表へのパーティションの追加

このトピックの例は、新規パーティションを[レンジ | リスト | 時間隔] - レンジ・パーティション表に追加する方法を示します。

特定の時間隔のデータが挿入されると、データベースにより時間隔 - レンジ・パーティション表に時間隔パーティションが自動的に作成されます。特定のサブパーティションに名前を付け、レンジを指定するには、SUBPARTITION句を指定します。SUBPARTITION句が指定されていない場合、パーティションは、表レベルに指定されたサブパーティション・テンプレートを継承します。サブパーティション・テンプレートがない場合には、最大値がMAXVALUEの単一のサブパーティションが作成されます。

例4-29では、2007年1月に注文された出荷品に関するデータが移入されるレンジ・パーティションp_2007_janを、レンジ - レンジ・パーティション表shipmentsに追加します。3つのサブパーティションがあります。サブパーティションはパーティション・レベルから圧縮属性を継承し、この例では圧縮された形式で格納されます。

例4-29 レンジ - レンジ・パーティション表へのパーティションの追加

```
ALTER TABLE shipments
   ADD PARTITION p_2007_jan
      VALUES LESS THAN (TO_DATE('01-FEB-2007','dd-MON-yyyy')) COMPRESS
      ( SUBPARTITION p07_jan_e VALUES LESS THAN (TO_DATE('15-FEB-2007','dd-MON-yyyy'))
      , SUBPARTITION p07_jan_a VALUES LESS THAN (TO_DATE('01-MAR-2007','dd-MON-yyyy'))
      , SUBPARTITION p07_jan_l VALUES LESS THAN (TO_DATE('01-APR-2007','dd-MON-yyyy'))
      ) ;
```

4.4.1.7.2 * - レンジ・パーティション表へのサブパーティションの追加

レンジ・サブパーティションを[レンジ | リスト | 時間隔] - レンジ・パーティション表に追加するには、ALTER TABLE文のMODIFY PARTITION ADD SUBPARTITION句を使用します。

時間隔 - レンジ・パーティション表の場合、パーティションを追加できるのは、マテリアライズ化されたレンジまたは時間隔パーティションのみです。

次の例では、order_dateが2007年1月で、delivery_dateが2007年4月1日以降のすべての値が含まれるshipments表に、レンジ・サブパーティションを追加します。

```
ALTER TABLE shipments
   MODIFY PARTITION p_2007_jan
      ADD SUBPARTITION p07_jan_vl VALUES LESS THAN (MAXVALUE) ;
```

4.4.1.8 参照パーティション表へのパーティションまたはサブパーティションの追加について

レンジ、ハッシュ、リストまたはコンポジット・パーティション表にパーティションやサブパーティションを追加できるのと同じように、参照

パーティション定義の親表にもパーティションまたはサブパーティションを追加できます。

追加操作は、子参照パーティション表に自動的にカスケードされます。マスター表へのパーティションまたはサブパーティションの追加時に、DEPENDENT TABLES句により、依存表に特定のプロパティを設定できます。

関連項目:

Oracle Database SQL言語リファレンス

4.4.1.9 索引パーティションの追加

ローカル索引には、明示的にパーティションを追加できません。かわりに、基礎となる表にパーティションを追加する場合にのみ、ローカル索引に新しいパーティションが追加されます。

特に、表にローカル索引が定義されていて、パーティションを追加するALTER TABLE文を発行する場合には、一致するパーティションがローカル索引にも追加されます。データベースにより、新しい索引パーティションに名前とデフォルトの物理記憶域属性が割り当てられますが、ADD PARTITION操作が完了した後にそれらを変更できます。

ADD PARTITION操作で、最初に索引のデフォルト属性を変更することにより、索引パーティションの新しい表領域を事実上指定できます。たとえば、リスト・パーティション表q1_sales_by_regionに、ローカル索引q1_sales_by_region_locixを作成したとします。「リスト・パーティション表へのパーティションの追加」に示されているように、新しいパーティションq1_nonmainlandを追加する前に次の文を発行した場合、対応する索引パーティションは表領域tbs_4に作成されます。

```
ALTER INDEX q1_sales_by_region_locix
   MODIFY DEFAULT ATTRIBUTES TABLESPACE tbs_4;
```

それ以外の場合は、追加した後に、次の文を使用して索引パーティションをtbs_4に移動する必要があります。

```
ALTER INDEX q1_sales_by_region_locix
   REBUILD PARTITION q1_nonmainland TABLESPACE tbs_4;
```

ALTER INDEXのADD PARTITION構文を使用して、ハッシュ・パーティション・グローバル索引にパーティションを追加できます。データベースによりハッシュ・パーティションが追加され、索引の既存のハッシュ・パーティションから再度ハッシュされた索引エントリが、ハッシュ関数で決定されたとおりに移入されます。次の文では、「ハッシュ・パーティション・グローバル索引の作成」に示されている索引hgidxに、パーティションを追加します

```
ALTER INDEX hgidx ADD PARTITION p5;
```

最高位パーティションにはMAXVALUEというパーティション・バウンドがあるため、レンジ・パーティション・グローバル索引にはパーティションを追加できません。新しい最高位パーティションを追加するには、ALTER INDEX SPLIT PARTITION文を使用します。

4.4.1.10 複数のパーティションの追加

ALTER TABLE文のADD PARTITIONおよびADD SUBPARTITION句を使用して、複数の新しいパーティションおよびサブパーティションを追加できます。

複数のパーティションを追加する場合、ローカルおよびグローバル索引操作は、単一のパーティションを追加する場合と同じ操作です。複数のパーティションおよびサブパーティションの追加は、レンジ、リストおよびシステムのパーティションまたはサブパーティションにのみサポートされています。

MAXVALUEパーティションが定義されていない場合、レンジ・パーティションまたはコンポジット・レンジ・パーティション表の最高位

(既存の最後のパーティションの後)に上限値の昇順で示される複数のレンジ・パーティションを追加できます。同様に、DEFAULTパーティションが存在しない場合、一連の新しいパーティション値を使用して、複数のリスト・パーティションを表に追加できます。

個々のパーティションを指定することで、単一のSQL文を使用して複数のシステム・パーティションを追加できます。たとえば、次のSQL文は、複数のパーティションを例4-1で作成されたレンジ・パーティション表salesに追加します。

```
ALTER TABLE sales ADD
  PARTITION sales_q1_2007 VALUES LESS THAN (TO_DATE('01-APR-2007','dd-MON-yyyy')),
  PARTITION sales_q2_2007 VALUES LESS THAN (TO_DATE('01-JUL-2007','dd-MON-yyyy')),
  PARTITION sales_q3_2007 VALUES LESS THAN (TO_DATE('01-OCT-2007','dd-MON-yyyy')),
  PARTITION sales_q4_2007 VALUES LESS THAN (TO_DATE('01-JAN-2008','dd-MON-yyyy'))
;
```

BEFORE句を使用して、1つの既存のパーティションのみとの関係で複数の新しいシステム・パーティションを追加できます。次のSQL文は、BEFORE句を使用して複数の個々のパーティションを追加する例を示します。

```
CREATE TABLE system_part_tab1 (number1 integer, number2 integer)
PARTITION BY SYSTEM
( PARTITION p1,
  PARTITION p2,
  PARTITION p3,
  PARTITION p_last);

ALTER TABLE system_part_tab1 ADD
  PARTITION p4,
  PARTITION p5,
  PARTITION p6
  BEFORE PARTITION p_last;

SELECT SUBSTR(TABLE_NAME,1,18) table_name, TABLESPACE_NAME,
   SUBSTR(PARTITION_NAME,1,16) partition_name
   FROM USER_TAB_PARTITIONS WHERE TABLE_NAME='SYSTEM_PART_TAB1';
TABLE_NAME         TABLESPACE_NAME                PARTITION_NAME
------------------ ------------------------------ ----------------
SYSTEM_PART_TAB1   USERS                          P_LAST
SYSTEM_PART_TAB1   USERS                          P6
SYSTEM_PART_TAB1   USERS                          P5
SYSTEM_PART_TAB1   USERS                          P4
SYSTEM_PART_TAB1   USERS                          P3
SYSTEM_PART_TAB1   USERS                          P2
SYSTEM_PART_TAB1   USERS                          P1
```

4.4.2 パーティションおよびサブパーティションの結合について

パーティションの結合は、ハッシュ・パーティション表または索引のパーティション数や、* - ハッシュ・パーティション表のサブパーティション数を減らす方法の1つです。

ハッシュ・パーティションを結合すると、そのコンテンツはハッシュ関数で決定された1つ以上の残りのパーティションに再配布されます。結合される特定のパーティションはデータベースにより選択され、コンテンツの再配布後に削除されます。参照パーティション表定義の親表のハッシュ・パーティションまたはサブパーティションを結合すると、参照パーティション表は新しいパーティション化定義を自動的に継承します。

次の表で説明するように、索引パーティションはUNUSABLEとマークされます。

表のタイプ	索引の動作

表のタイプ	索引の動作
通常(ヒープ)	ALTER TABLE 文の一部に UPDATE INDEXES を指定しない場合: ● 選択したパーティションに対応するローカル索引パーティションも削除されます。取り込む側の 1 つ以上のパーティションに対応するローカル索引パーティションは UNUSABLE とマークされ、再作成する必要があります。 ● すべてのグローバル索引またはパーティション・グローバル索引のすべてのパーティションが UNUSABLE とマークされ、再作成する必要があります。
索引構成	● ヒープ索引で説明したように、一部のローカル索引は UNUSABLE とマークされます。 ● すべてのグローバル索引は使用可能なままです。

この項では、次の項目について説明します。

- ハッシュ・パーティション表でのパーティションの結合
- * - ハッシュ・パーティション表でのサブパーティションの結合
- ハッシュ・パーティション・グローバル索引の結合

4.4.2.1 ハッシュ・パーティション表でのパーティションの結合

ハッシュ・パーティション表でパーティションを結合する場合には、ALTER TABLE COALESCE PARTITION文が使用されます。

次の文では、パーティションを結合することにより、表のパーティション数が1つ削減されています。

```
ALTER TABLE ouu1
     COALESCE PARTITION;
```

4.4.2.2 * - ハッシュ・パーティション表でのサブパーティションの結合

ALTER TABLE COALESCE SUBPARTITION文は、ハッシュ・パーティション表内のサブパーティションの結合に使用されます。

次の文により、パーティションus_locationsのサブパーティションのコンテンツが、同じパーティションの1つ以上の残りのサブパーティション(ハッシュ関数で決定)に分散されます。時間隔パーティション表の場合、結合できるのは、マテリアライズ化されたレンジまたは時間隔パーティションのハッシュ・サブパーティションのみです。基本的に、この操作は、「* - ハッシュ・パーティション表へのサブパーティションの追加」で説明したMODIFY PARTITION ADD SUBPARTITION句の逆の操作です。

```
ALTER TABLE diving MODIFY PARTITION us_locations
     COALESCE SUBPARTITION;
```

4.4.2.3 ハッシュ・パーティション・グローバル索引の結合

ALTER INDEXのCOALESCE PARTITION句を使用して、ハッシュ・パーティション・グローバル索引の索引パーティション数を1つ減らすようにデータベースに指示できます。

データベースにより、ハッシュ・パーティションの要件に基づいて、結合するパーティションが選択されます。次の文では、「ハッシュ・パーティション・グローバル索引の作成」で作成されたhgidx索引のパーティション数が1つ削減されます。

```
ALTER INDEX hgidx COALESCE PARTITION;
```

4.4.3 パーティションおよびサブパーティションの削除について

レンジ、時間隔、リストまたはコンポジット* - [レンジ | リスト]パーティション表からパーティションを削除できます。

時間隔パーティション表の場合は、マテリアライズ化されたレンジまたは時間隔パーティションのみ削除できます。ハッシュ・パーティション表またはコンポジット* - ハッシュ・パーティション表のハッシュ・サブパーティションの場合は、かわりに、結合操作を実行する必要があります。

参照パーティション表からパーティションを削除することはできません。かわりに、親表での削除操作がすべての子表にカスケードされます。

この項では、次の項目について説明します。

- 表パーティションの削除
- 時間隔パーティションの削除
- 索引パーティションの削除
- 複数のパーティションの削除

4.4.3.1 表パーティションの削除

表パーティションを削除するには、ALTER TABLE SQL文でDROP PARTITIONまたはDROP SUBPARTITIONを使用します。

次の文を使用して、表、パーティションまたはサブパーティションを削除できます。

- ALTER TABLE DROP PARTITION: 表パーティションを削除する場合
- ALTER TABLE DROP SUBPARTITION: コンポジット* - [レンジ | リスト]パーティション表のサブパーティションを削除する場合

パーティション内のデータを保持するには、DROP PARTITION文ではなくMERGE PARTITION文を使用します。

パーティションを削除せずにパーティションのデータを削除するには、TRUNCATE PARTITION文を使用します。

表にローカル索引が定義されている場合、この文では、ローカル索引からも一致するパーティションまたはサブパーティションが削除されます。次のいずれかが当てはまる場合、すべてのグローバル索引またはパーティション・グローバル索引のすべてのパーティションがUNUSABLEとマークされます。

- UPDATE INDEXESを指定する(索引構成表には指定できません。かわりにUPDATE GLOBAL INDEXESを使用してください)。
- 削除するパーティションまたはそのサブパーティションが空である。

ノート:

- 表にパーティションが 1 つだけ含まれる場合、そのパーティションは削除できません。かわりに表を削除する必要があります。
- 時間隔パーティション表または時間隔 - *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・セクションにある最高位のレンジ・パーティションは削除できません。
- 非同期グローバル索引メンテナンスでは、パーティション削除の索引更新操作はメタデータのみで、すべてのグローバル索引が有効なままです。

- Oracle Database のごみ箱の設定にかかわらず、パーティションの削除によってそのパーティションがごみ箱に入れられることはありません。削除したパーティションは、ただちにシステムから削除されます。

次の項では、表パーティションの削除に関するいくつかのシナリオを示します。

- データおよびグローバル索引を含む表からのパーティションの削除
- データおよび参照整合性制約を含むパーティションの削除

関連項目:

- パーティションのマージの詳細は、「パーティションおよびサブパーティションのマージについて」を参照してください
- パーティションの切捨ての詳細は、「パーティションおよびサブパーティションの切捨てについて」を参照してください
- パーティションを削除する非同期グローバル索引メンテナンスの詳細は、「パーティションを削除および切り捨てる非同期グローバル索引メンテナンス」を参照してください

4.4.3.1.1 データおよびグローバル索引を含む表からのパーティションの削除

いくつかの方法を使用して、データおよびグローバル索引を含む表からパーティションを削除できます。

パーティションにデータが含まれている場合や、表に1つ以上のグローバル索引が定義されている場合には、次のいずれかの方法(方法1、2または3)を使用して表パーティションを削除します。

方法1

グローバル索引をメンテナンスせずにALTER TABLE DROP PARTITION文を発行します。その後、索引(または索引パーティション)がUNUSABLEとマークされるため、(パーティション化されているかどうかに関係なく)すべてのグローバル索引を再作成する必要があります。次の文で、sales表からパーティションdec98を削除し、パーティション化されていないグローバル索引を再作成する例を示します。

```
ALTER TABLE sales DROP PARTITION dec98;
ALTER INDEX sales_area_ix REBUILD;
```

索引sales_area_ixがレンジ・パーティション・グローバル索引の場合には、その索引のすべてのパーティションを再作成する必要があります。さらに、1文で索引のすべてのパーティションを再作成することはできません。索引の各パーティションに対して、別々にREBUILD文を発行する必要があります。次の文では、索引パーティションjan99_ixからdec99_ixを再作成します。

```
ALTER INDEX sales_area_ix REBUILD PARTITION jan99_ix;
ALTER INDEX sales_area_ix REBUILD PARTITION feb99_ix;
ALTER INDEX sales_area_ix REBUILD PARTITION mar99_ix;
...
ALTER INDEX sales_area_ix REBUILD PARTITION dec99_ix;
```

これは、削除されるパーティションに表の合計データの大部分が含まれる大規模な表に最適な方法です。非同期グローバル索引メンテナンスは、索引メンテナンスを必要とせずにグローバル索引を有効なまま保持しますが、UPDATE INDEXES句を使用してこの新しい機能を有効にする必要があります。この動作によって、下位互換性が保証されます。

方法2

ALTER TABLE DROP PARTITION文を発行する前に、DELETE文を発行してパーティションからすべての行を削除します。DELETE

文により、グローバル索引が更新されます。

たとえば、最初のパーティションを削除するには、次の文を発行します。

```
DELETE FROM sales partition (dec98);
ALTER TABLE sales DROP PARTITION dec98;
```

これは、小規模な表、または削除されるパーティションに含まれる表の合計データの割合が少ない場合に、大規模な表に最適な方法です。

方法3

ALTER TABLE文にUPDATE INDEXESを指定します。実行すると、新しい非同期グローバル索引メンテナンスが利用されます。索引は有効なまま変化しません。

```
ALTER TABLE sales DROP PARTITION dec98
     UPDATE INDEXES;
```

4.4.3.1.2 データおよび参照整合性制約を含むパーティションの削除

いくつかの方法を使用して、データおよび参照整合性制約を含むパーティションを削除できます。

パーティションにデータが含まれ、表に参照整合性制約がある場合には、次のいずれかの方法(方法1または2)を選択して表パーティションを削除します。この表にあるのはローカル索引のみであるため、索引を再作成する必要はありません。

方法1

削除するパーティションのデータを参照しているデータがない場合は、参照表の整合性制約を無効化してALTER TABLE DROP PARTITION文を発行し、整合性制約を再度有効化します。

これは、削除されるパーティションに表の合計データの大部分が含まれる大規模な表に最適な方法です。削除するパーティションのデータを参照するデータがある場合には、参照元のすべてのデータを削除して、参照整合性制約を再度有効化できるようにしてください。

方法2

参照表にデータがある場合は、ALTER TABLE DROP PARTITION文を発行する前に、DELETE文を発行してパーティションからすべての行を削除できます。DELETE文によって、参照整合性制約が施行されます。また、トリガーが起動されて、REDOログとUNDOログが生成されます。ON DELETE CASCADEオプションを指定して制約を作成すると削除が成功し、参照表からもすべての行が削除されます。

```
DELETE FROM sales partition (dec94);
ALTER TABLE sales DROP PARTITION dec94;
```

これは、小規模な表、または削除されるパーティションに含まれる表の合計データの割合が少ない場合に、大規模な表に最適な方法です。

4.4.3.2 時間隔パーティションの削除

時間隔パーティション表の時間隔パーティションを削除できます。

この操作により、その時間隔のデータのみが削除され、時間隔定義はそのまま残ります。削除されたばかりの時間隔にデータが挿入された場合は、データベースにより時間隔パーティションが再度作成されます。

時間隔パーティション表のレンジ・パーティションも削除できます。時間隔パーティション表のレンジ・パーティションを削除するルールは、レンジ・パーティション表のレンジ・パーティションの削除ルールと同じです。一連のレンジ・パーティションの中間にあるレンジ・パーティションを削除すると、次のレンジ・パーティションの下限が、削除したレンジ・パーティションの下限に切り替わります。時

間隔パーティション表のレンジ・パーティション・セクションにある最高位のレンジ・パーティションは削除できません。

次の例では、sales表から2007年9月の時間隔パーティション表を削除します。ローカル索引しかないため、索引は無効化されません。

```
ALTER TABLE sales DROP PARTITION FOR(TO_DATE('01-SEP-2007','dd-MON-yyyy'));
```

4.4.3.3 索引パーティションの削除

ローカル索引のパーティションは明示的に削除できません。かわりに、基礎となる表からパーティションを削除した場合にのみ、ローカル索引パーティションが削除されます。

グローバル索引パーティションが空の場合は、ALTER INDEX DROP PARTITION文を発行することでそのパーティションを明示的に削除できます。ただし、グローバル索引パーティションにデータが含まれる場合は、パーティションを削除すると次の最高位パーティションがUNUSABLEとマークされる原因になります。たとえば、索引パーティションP1を削除する必要があり、P2が次の最高位パーティションであるとします。次の文を発行する必要があります。

```
ALTER INDEX npr DROP PARTITION P1;
ALTER INDEX npr REBUILD PARTITION P2;
```

ノート:

グローバル索引では、最高位パーティションを削除できません。

4.4.3.4 複数のパーティションの削除

SQL ALTER TABLE文のDROP PARTITIONおよびDROP SUBPARTITION句を使用して、レンジまたはリスト・パーティション表から複数のパーティションまたはサブパーティションを削除できます。

たとえば、次のSQL文は、レンジ・パーティション表salesから複数のパーティションを削除します。

```
ALTER TABLE sales DROP PARTITION sales_q1_2008, sales_q2_2008,
     sales_q3_2008, sales_q4_2008;
```

表のすべてのパーティションを削除できません。複数のパーティションを削除する場合、ローカルおよびグローバル索引操作は、単一のパーティションを削除する場合と同じ操作です。

4.4.4 パーティションおよびサブパーティションの交換について

データ・セグメントを交換することで、パーティションまたはサブパーティションを非パーティション表に、非パーティション表を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に変換できます。

ハッシュ・パーティション表をコンポジット* - ハッシュ・パーティション表のパーティションに変換することも、コンポジット* - ハッシュ・パーティション表のパーティションをハッシュ・パーティション表に変換することも可能です。同様に、レンジ - リスト・パーティション表をコンポジット* - レンジまたはリスト・パーティション表のパーティションに変換することも、コンポジット* - レンジまたはリスト・パーティション表のパーティションをレンジ - リスト・パーティション表に変換することも可能です。

表パーティションの交換は、パーティション表の内外で迅速にデータを取得するのに便利です。たとえば、データ・ウェアハウス環境で、パーティションの交換を行うと、新しい増分データの既存のパーティション表への高速データ・ローディングが容易になります。

交換プロセスでは、ソースのデータがターゲットに移動され、ターゲットのデータがソースに移動されることに注意してください。

OLTP環境やデータ・ウェアハウス環境では、パーティション表の古いデータ・パーティションを交換することで利点が得られます。実際には削除されずにパーティション表からデータがパージされ、後から個別にアーカイブできます。

パーティションを交換すると、ロギング属性が保持されます。INCLUDING INDEXES句でローカル索引も交換するかどうか、およびWITH VALIDATION句で行が適切にマッピングされていることを検証するかどうかをオプションで指定できます。

ノート:

パーティションの交換操作に WITHOUT VALIDATION を指定する場合、これに関連するのはデータ・ディクショナリの更新のみであるため、通常は高速な操作です。ただし、交換操作に関連する表またはパーティション表に主キーがある場合や一意制約が有効化されている場合には、制約の整合性を維持するため、交換操作は WITH VALIDATION が指定されているように実行されます。

この検証アクティビティのオーバーヘッドを避けるには、パーティションの交換操作を実行する前に、各制約に対して次の文を発行します。

```
ALTER TABLE table_name
     DISABLE CONSTRAINT constraint_name KEEP INDEX
```

交換後に制約を有効化します。

WITHOUT VALIDATION を指定する場合は、交換するデータが交換するパーティションに含まれることを確認する必要があります。間違ったパーティションに不正に挿入されたレコードを識別するには、ORA_PARTITION_VALIDATION SQL 関数を使用できます。

UPDATE INDEXESを指定しないかぎり、パーティションをUNUSABLEとして交換する表のグローバル索引またはすべてのグローバル索引パーティションが、Oracle Databaseによりマークされます。交換が行われる表のグローバル索引またはグローバル索引パーティションは、無効化されたままになります。

索引構成表にはUPDATE INDEXESは使用できません。かわりにUPDATE GLOBAL INDEXESを使用してください。

DBMS_STATS表プリファレンスのINCREMENTALがtrueに設定され、INCREMENTAL_LEVELがTABLEに設定されると、統計が非パーティション表で収集された場合にパーティション表の増分統計がパーティション交換操作でメンテナンスされます。

ノート:

仮想列の列統計が不適当な場合、その古い統計を保持するのではなく、列統計は削除されます。この削除についての情報は、アラート・ログ・ファイルに書き込まれます。

この項では、次の項目について説明します。

- パーティション表と交換するための表の作成
- レンジ、ハッシュまたはリスト・パーティションの交換
- 時間隔パーティション表のパーティションの交換
- 参照パーティション表のパーティションの交換

- 仮想列を含む表のパーティションの交換について
- ハッシュ・パーティション表と* - ハッシュ・パーティションの交換
- * - ハッシュ・パーティション表のサブパーティションの交換
- リスト・パーティション表と* - リスト・パーティションの交換
- * - リスト・パーティション表のサブパーティションの交換について
- レンジ・パーティション表と* - レンジ・パーティションの交換
- * - レンジ・パーティション表のサブパーティションの交換について
- カスケード・オプションを使用したパーティションの交換について

関連項目:

- パーティションのコンテンツの検証の詳細は、パーティション化キーを参照してください
- パーティション表および索引の詳細を監視するビューの詳細は、「パーティション表および索引の情報の表示」を参照してください
- 増分統計の詳細は、『Oracle Database SQL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パッケージおよびタイプ・リファレンス』を参照してください

4.4.4.1 パーティション表と交換するための表の作成

FOR EXCHANGE WITH句を使用すると、パーティション表の形状と完全に一致し、パーティション交換コマンドに適するように表を作成できます。ただし、索引はこのコマンドの操作としては作成されません。

CREATE TABLEのFOR EXCHANGE WITH句は、非パーティション表とパーティション表の完全一致を提供するため、CREATE TABLE AS SELECT文が改善されます。

次のリストは、CREATE TABLE FOR EXCHANGE WITH DDL操作の効果をまとめたものです。

- このDDL操作のユースケースにより、パーティション交換DDLで使用される表の作成が容易になります。
- この操作により、列の順序および列のプロパティの点で交換対象の表のクローンが作成されます。
- 列の名前は変更できません。作成される表は、交換対象の表から名前を継承します。
- DDL操作中に指定できる論理プロパティのみが、表のパーティション化指定です。

  パーティション化句は、コンポジット・パーティション表のパーティションとの交換にのみ関連します。この場合、*n*サブパーティションを含むパーティションは、サブパーティションと一致する*n*パーティションを含むパーティション表と交換されます。このシナリオのこの交換には、パーティション化句の定義が必要です。

  サブパーティション化は、パーティション間で非対象となります。パーティション化句は、交換対象のパーティションのサブパーティション化と完全に一致する必要があります。
- 指定できる物理プロパティは、主な表セグメント属性です。
- このDDL操作でコピーされる列プロパティには、使用不可の列、非表示の列、仮想式の列、関数索引式の列、他の内部設定や属性などが含まれます。

次に、列の順序付けとプロパティについて既存の表の形式を模倣する表を作成するには、CREATE TABLE文でFOR EXCHANGE WITH句を使用する例を示します。

例4-30 CREATE TABLEのFOR EXCHANGE WITH句の使用

```
CREATE TABLE sales_by_year_table
  ( prod_id       NUMBER        NOT NULL,
    cust_id       NUMBER        NOT NULL,
    time_id       DATE          NOT NULL,
    channel_id    NUMBER        NOT NULL,
    promo_id      NUMBER        NOT NULL,
    quantity_sold NUMBER(10,2)  NOT NULL,
    amount_sold   NUMBER(10,2)  NOT NULL
  )
    PARTITION BY RANGE (time_id)
     (PARTITION sales_2016 VALUES LESS THAN (TO_DATE('01-01-2017','dd-mm-yyyy')),
      PARTITION sales_2017 VALUES LESS THAN (TO_DATE('01-01-2018','dd-mm-yyyy')),
      PARTITION sales_2018 VALUES LESS THAN (TO_DATE('01-01-2019','dd-mm-yyyy')),
      PARTITION sales_2019 VALUES LESS THAN (TO_DATE('01-01-2020','dd-mm-yyyy')),
      PARTITION sales_future VALUES LESS THAN (MAXVALUE)
 );

DESCRIBE sales_by_year_table
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PROD_ID                                   NOT NULL NUMBER
 CUST_ID                                   NOT NULL NUMBER
 TIME_ID                                   NOT NULL DATE
 CHANNEL_ID                                NOT NULL NUMBER
 PROMO_ID                                  NOT NULL NUMBER
 QUANTITY_SOLD                             NOT NULL NUMBER(10,2)
 AMOUNT_SOLD                               NOT NULL NUMBER(10,2)

CREATE TABLE sales_later_year_table
  FOR EXCHANGE WITH TABLE sales_by_year_table;

DESCRIBE sales_later_year_table
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PROD_ID                                   NOT NULL NUMBER
 CUST_ID                                   NOT NULL NUMBER
 TIME_ID                                   NOT NULL DATE
 CHANNEL_ID                                NOT NULL NUMBER
 PROMO_ID                                  NOT NULL NUMBER
 QUANTITY_SOLD                             NOT NULL NUMBER(10,2)
 AMOUNT_SOLD                               NOT NULL NUMBER(10,2)
```

4.4.4.2 レンジ、ハッシュまたはリスト・パーティションの交換

レンジ、ハッシュまたはリスト・パーティション表のパーティションを非パーティション表と交換する場合、またはその逆を行う場合は、ALTER TABLE EXCHANGE PARTITION文を使用します。

次に、レンジ・パーティションを非パーティション表と交換する例を示します。

例4-31 レンジ・パーティションの交換

```
CREATE TABLE sales_future_table
  ( prod_id       NUMBER        NOT NULL,
    cust_id       NUMBER        NOT NULL,
    time_id       DATE          NOT NULL,
```

```
    channel_id    NUMBER        NOT NULL,
    promo_id      NUMBER        NOT NULL,
    quantity_sold NUMBER(10,2)  NOT NULL,
    amount_sold   NUMBER(10,2)  NOT NULL
  )
    PARTITION BY RANGE (time_id)
     (PARTITION s_2020 VALUES LESS THAN (TO_DATE('01-01-2021','dd-mm-yyyy')),
      PARTITION s_2021 VALUES LESS THAN (TO_DATE('01-01-2022','dd-mm-yyyy')),
      PARTITION s_2022 VALUES LESS THAN (TO_DATE('01-01-2023','dd-mm-yyyy'))
);

CREATE TABLE sales_exchange_table
  FOR EXCHANGE WITH TABLE sales_future_table;

INSERT INTO sales_exchange_table VALUES (1002,110,TO_DATE('19-02-2020','dd-mm-yyyy'),12,18,150,4800);
INSERT INTO sales_exchange_table VALUES (1001,100,TO_DATE('12-03-2020','dd-mm-yyyy'),10,15,400,6500);
INSERT INTO sales_exchange_table VALUES (1001,100,TO_DATE('31-05-2020','dd-mm-yyyy'),10,15,600,8000);
INSERT INTO sales_exchange_table VALUES (2105,101,TO_DATE('25-06-2020','dd-mm-yyyy'),12,19,100,3000);
INSERT INTO sales_exchange_table VALUES (1002,120,TO_DATE('31-08-2020','dd-mm-yyyy'),10,15,400,6000);
INSERT INTO sales_exchange_table VALUES (2105,101,TO_DATE('25-10-2020','dd-mm-yyyy'),12,19,250,7500);

ALTER TABLE sales_future_table
    EXCHANGE PARTITION s_2020 WITH TABLE sales_exchange_table;

SELECT * FROM sales_future_table PARTITION(s_2020);
   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      1002        110 19-FEB-20         12         18           150        4800
      1001        100 12-MAR-20         10         15           400        6500
      1001        100 31-MAY-20         10         15           600        8000
      2105        101 25-JUN-20         12         19           100        3000
      1002        120 31-AUG-20         10         15           400        6000
      2105        101 25-OCT-20         12         19           250        7500
6 rows selected.

REM Note that all records have been removed from the sales_exchange_table
SELECT * FROM sales_exchange_table;
no rows selected

INSERT INTO sales_exchange_table VALUES (1002,110,TO_DATE('15-02-2021','dd-mm-yyyy'),12,18,300,9500);
INSERT INTO sales_exchange_table VALUES (1002,120,TO_DATE('31-03-2021','dd-mm-yyyy'),10,15,200,3000);
INSERT INTO sales_exchange_table VALUES (2105,101,TO_DATE('25-04-2021','dd-mm-yyyy'),12,19,150,9000);

ALTER TABLE sales_future_table
    EXCHANGE PARTITION s_2021 WITH TABLE sales_exchange_table;

SELECT * FROM sales_future_table PARTITION(s_2021);
   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      1002        110 15-FEB-21         12         18           300        9500
      1002        120 31-MAR-21         10         15           200        3000
      2105        101 25-APR-21         12         19           150        9000
3 rows selected.
```

4.4.4.3 時間隔パーティション表のパーティションの交換

時間隔パーティション表の時間隔パーティションを交換できます。ただし、パーティションを交換する前に、時間隔パーティションが作成されていることを確認する必要があります。

次の例では、2007年1月1日現在に、月単位のパーティションを使用して時間隔パーティション化された、interval_sales表のパーティションの交換を示します。この例では、パーティション交換ロードを使用して、表に2007年6月のデータを追加する方法を示します。interval_sales表にはローカル索引のみがあり、interval_sales_june_2007表に対応する索引が作成されているとします。

```
ALTER TABLE interval_sales
  EXCHANGE PARTITION FOR (TO_DATE('01-JUN-2007','dd-MON-yyyy'))
  WITH TABLE interval_sales_jun_2007
  INCLUDING INDEXES;
```

システム生成されたパーティションを識別するためのFOR構文の使用方法に注意してください。*_TAB_PARTITIONSデータ・ディクショナリ・ビューを問い合せてシステム生成のパーティション名を表示することで、パーティション名を判別できます。

4.4.4.4 参照パーティション表のパーティションの交換

参照パーティション表のパーティションを交換できますが、参照するデータが親表のそれぞれのパーティションで使用可能であることを確認する必要があります。

例4-32では、レンジ・パーティション化されたorders表、および参照パーティション化されたorder_items表のパーティション交換ロードのシナリオを示します。order_items_2018_dec表のデータには、order_dateが2018年12月の注文の注文アイテム・データのみが含まれています。

主キー索引を使用可能なままにするには、親表のパーティションの交換では、UPDATE GLOBAL INDEXESまたはUPDATE INDEXESを使用する必要があります。また、参照パーティション表でのパーティションの交換を正常に行うには、order_items_2018_dec表で外部キー制約を作成するか有効化する必要があります。

CASCADEキーワードで交換する場合の詳細と例は、カスケード・オプションを使用したパーティションの交換についての項を参照してください。

例4-32 参照パーティション表のパーティションの交換

```
CREATE TABLE orders (
       order_id number NOT NULL,
       order_date DATE,
       CONSTRAINT order_pk PRIMARY KEY (order_id))
       PARTITION by range (order_date)
       (PARTITION p_2018_dec values less than ('01-JAN-2019'));

CREATE TABLE order_items  (
       order_item_id NUMBER NOT NULL,
       order_id NUMBER not null,
       order_item VARCHAR2(100),
       CONSTRAINT order_item_pk PRIMARY KEY (order_item_id),
       CONSTRAINT order_item_fk FOREIGN KEY (order_id) references orders(order_id) on delete cascade)
       PARTITION by reference (order_item_fk);

CREATE TABLE orders_2018_dec (
       order_id NUMBER,
       order_date DATE,
       CONSTRAINT order_2018_dec_pk PRIMARY KEY (order_id));

INSERT into orders_2018_dec values (1,'01-DEC-2018');
COMMIT;

CREATE TABLE order_items_2018_dec (
       order_item_id NUMBER,
       order_id NUMBER NOT NULL,
```

```
       order_item VARCHAR2(100),
       CONSTRAINT order_item_2018_dec_pk PRIMARY KEY (order_item_id),
       CONSTRAINT order_item_2018_dec_fk FOREIGN KEY (order_id) references orders_2018_dec (order_id)
on delete cascade);

INSERT into order_items_2018_dec values (1,1,'item A');
INSERT into order_items_2018_dec values (2,1,'item B');

REM You must disable or DROP the constraint before the exchange
ALTER TABLE order_items_2018_dec DROP CONSTRAINT order_item_2018_dec_fk;

REM ALTER TABLE is successful with disabled PK-FK
ALTER TABLE orders
  EXCHANGE PARTITION p_2018_dec
  WITH TABLE orders_2018_dec
  UPDATE GLOBAL INDEXES;

REM You must establish the PK-FK with the future parent prior to this exchange
ALTER TABLE order_items_2018_dec
  ADD CONSTRAINT order_items_dec_2018_fk
  FOREIGN KEY (order_id)
  REFERENCES orders(order_id) ;

REM Complete the exchange
ALTER TABLE order_items
  EXCHANGE PARTITION p_2018_dec
  WITH TABLE order_items_2018_dec;

REM Display the data
SELECT * FROM orders;
  ORDER_ID ORDER_DAT
---------- ---------
         1 01-DEC-18

SELECT * FROM order_items;
ORDER_ITEM_ID   ORDER_ID ORDER_ITEM
------------- ---------- ------------
            1          1 item A
            2          1 item B
```

4.4.4.5 仮想列を含む表のパーティションの交換について

仮想列が存在する場合もパーティションを交換できます。

仮想列を含むパーティション表でパーティションの交換を正常に行うには、パーティション表の単一のパーティションにある仮想列以外のすべての列の定義に一致する表を作成する必要があります。仮想列に制約または索引が定義されている場合を除き、仮想列の定義を含める必要はありません。

定義されている場合は、パーティション表の制約および索引の定義に一致させるために、仮想列の定義を含める必要があります。このシナリオは、仮想列ベースのパーティション表にも適用されます。

4.4.4.6 ハッシュ・パーティション表と* - ハッシュ・パーティションの交換

すべてのパーティションを含むハッシュ・パーティション表全体を、* - ハッシュ・パーティション表のパーティションおよびそのすべてのハッシュ・サブパーティションと交換できます。

次の例では、レンジ - ハッシュ・パーティション表のこの概念を説明します。

最初に、ハッシュ・パーティション表を作成します。

```
CREATE TABLE t1 (i NUMBER, j NUMBER)
     PARTITION BY HASH(i)
       (PARTITION p1, PARTITION p2);
```

表に移入し、次のようにしてレンジ - ハッシュ・パーティション表を作成します。

```
CREATE TABLE t2 (i NUMBER, j NUMBER)
     PARTITION BY RANGE(j)
     SUBPARTITION BY HASH(i)
        (PARTITION p1 VALUES LESS THAN (10)
            (SUBPARTITION t2_pls1,
             SUBPARTITION t2_pls2),
         PARTITION p2 VALUES LESS THAN (20)
            (SUBPARTITION t2_p2s1,
             SUBPARTITION t2_p2s2)
         );
```

表t1のパーティション化キーが、表t2のサブパーティション化キーと同一であることが重要です。

t1のデータをt2に移行して行を検証するには、次の文を使用します。

```
ALTER TABLE t2 EXCHANGE PARTITION p1 WITH TABLE t1
     WITH VALIDATION;
```

4.4.4.7 * - ハッシュ・パーティション表のサブパーティションの交換

* - ハッシュ・パーティション表のハッシュ・サブパーティションを非パーティション表に変換する場合、またはその逆を行うには、ALTER TABLE EXCHANGE SUBPARTITION文を使用します。

次の例では、表salesのサブパーティションq3_1999_s1を、非パーティション表q3_1999に変換します。ローカル索引パーティションは、q3_1999の対応する索引と交換されます。

```
ALTER TABLE sales EXCHANGE SUBPARTITION q3_1999_s1
      WITH TABLE q3_1999 INCLUDING INDEXES;
```

4.4.4.8 リスト・パーティション表と* - リスト・パーティションの交換

ALTER TABLE EXCHANGE PARTITION文を使用すると、リスト - パーティション表を* - リスト・パーティションと交換できます。

セマンティックは、「ハッシュ・パーティション表と* - ハッシュ・パーティションの交換」で前述されているものと同じです。次の例では、リスト - リスト・パーティション表のパーティションの交換シナリオを示します。

```
CREATE TABLE customers_apac
( id            NUMBER
, name          VARCHAR2(50)
, email         VARCHAR2(100)
, region        VARCHAR2(4)
, credit_rating VARCHAR2(1)
)
PARTITION BY LIST (credit_rating)
( PARTITION poor VALUES ('P')
, PARTITION mediocre VALUES ('C')
, PARTITION good VALUES ('G')
, PARTITION excellent VALUES ('E')
);
```

表にAPACの顧客を移入します。次に、リスト - リスト・パーティション表を作成します。

```
CREATE TABLE customers
( id            NUMBER
, name          VARCHAR2(50)
, email         VARCHAR2(100)
, region        VARCHAR2(4)
, credit_rating VARCHAR2(1)
)
PARTITION BY LIST (region)
SUBPARTITION BY LIST (credit_rating)
SUBPARTITION TEMPLATE
( SUBPARTITION poor VALUES ('P')
, SUBPARTITION mediocre VALUES ('C')
, SUBPARTITION good VALUES ('G')
, SUBPARTITION excellent VALUES ('E')
)
(PARTITION americas VALUES ('AMER')
, PARTITION emea VALUES ('EMEA')
, PARTITION apac VALUES ('APAC')
);
```

customers_apac表のパーティション化キーが、customers表のサブパーティション化キーと一致していることが重要です。

次に、apacパーティションを交換します。

```
ALTER TABLE customers
EXCHANGE PARTITION apac
WITH TABLE customers_apac
WITH VALIDATION;
```

4.4.4.9 * - リスト・パーティション表のサブパーティションの交換について

* - リスト・パーティション表のサブパーティションを交換するには、ALTER TABLE EXCHANGE SUBPARTITION文を使用します。

ALTER TABLE EXCHANGE SUBPARTITION文のセマンティックは、「ハッシュ・パーティション表のサブパーティションの交換」で説明されているセマンティックと同じです。

4.4.4.10 レンジ・パーティション表と* - レンジ・パーティションの交換

ALTER TABLE EXCHANGE PARTITION文を使用すると、レンジ - パーティション表を* - レンジ・パーティションと交換できます。

ALTER TABLE EXCHANGE PARTITION文のセマンティックは、ハッシュ・パーティション表と* - ハッシュ・パーティションの交換で説明されているセマンティックと同じです。次の例では、order_dateで時間隔パーティション化され、order_totalのレンジでサブパーティション化されたorders表を示します。この例では、単一の月単位の時間隔をレンジ・パーティション表と交換する方法を示します。

```
CREATE TABLE orders_mar_2007
 ( id          NUMBER
  , cust_id     NUMBER
  , order_date  DATE
  , order_total NUMBER
 )
PARTITION BY RANGE (order_total)
 ( PARTITION p_small VALUES LESS THAN (1000)
  , PARTITION p_medium VALUES LESS THAN (10000)
  , PARTITION p_large VALUES LESS THAN (100000)
  , PARTITION p_extraordinary VALUES LESS THAN (MAXVALUE)
 );
```

表に2007年5月の注文を移入します。次に時間隔 - レンジ・パーティション表を作成します。

```
CREATE TABLE orders
 ( id          NUMBER
  , cust_id     NUMBER
  , order_date  DATE
  , order_total NUMBER
 )
PARTITION BY RANGE (order_date) INTERVAL (NUMTOYMINTERVAL(1,'MONTH'))
  SUBPARTITION BY RANGE (order_total)
  SUBPARTITION TEMPLATE
  ( SUBPARTITION p_small VALUES LESS THAN (1000)
   , SUBPARTITION p_medium VALUES LESS THAN (10000)
   , SUBPARTITION p_large VALUES LESS THAN (100000)
   , SUBPARTITION p_extraordinary VALUES LESS THAN (MAXVALUE)
  )
 (PARTITION p_before_2007 VALUES LESS THAN (TO_DATE('01-JAN-2007','dd-MON-yyyy')));
```

orders_mar_2007表のパーティション化キーが、orders表のサブパーティション化キーと一致していることが重要です。

次に、パーティションを交換します。

```
ALTER TABLE orders
  EXCHANGE PARTITION
   FOR (TO_DATE('01-MAR-2007','dd-MON-yyyy'))
   WITH TABLE orders_mar_2007
   WITH VALIDATION;
```

4.4.4.11 * - レンジ・パーティション表のサブパーティションの交換について

* - レンジ・パーティションのサブパーティションを交換するには、ALTER TABLE EXCHANGE SUBPARTITION文を使用します。

ALTER TABLE EXCHANGE SUBPARTITIONのセマンティックは、「ハッシュ・パーティション表のサブパーティションの交換」で説明されているセマンティックと同じです。

4.4.4.12 カスケード・オプションを使用したパーティションの交換について

ALTER TABLE EXCHANGE PARTITIONおよびALTER TABLE EXCHANGE SUBPARTITION SQL文のCASCADEオプションを使用して、子の参照パーティション表に交換操作をカスケードできます。

交換操作のカスケードは、すべての外部キー制約をON DELETE CASCADEとして定義する必要があります。

ALTER TABLE EXCHANGE PARTITIONおよびALTER TABLE EXCHANGE SUBPARTITIONのCASCADEオプションを指定すると、EXCHANGE操作は、ターゲット表の子である参照パーティション表にカスケードします。交換操作は、参照パーティション階層の任意のレベルで指定でき、ターゲット表から開始される子表にカスケードします。子表に権限は必要ありませんが、交換操作の通常の制限が操作に影響されるすべての表に適用されます。参照パーティションの子を持たない表に指定されている場合、CASCADEオプションは無視されます。

ターゲット表の参照パーティション階層および交換表の参照パーティション階層は一致する必要があります。同じ親キーが複数の依存表によって参照される場合、CASCADEオプションはサポートされません。複数の依存表が同じ主キーを使用していると、カーネルは依存パーティションの交換方法を明示的に識別できません。UPDATE INDEXESなど、操作に指定される他のオプションは、操作に影響されるすべての表に適用されます。

カスケード・オプションはデフォルトで無効になっているため、Oracle Database互換性に影響しません。

次の例は、参照パーティション表のパーティションを交換する場合のCASCADEの使用を示しています。

例4-33 参照パーティション表のカスケードを使用するパーティションの交換

```
CREATE TABLE orders (
       order_id number NOT NULL,
       order_date DATE,
       CONSTRAINT order_pk PRIMARY KEY (order_id))
       PARTITION by range (order_date)
       (PARTITION p_2018_dec values less than ('01-JAN-2019'));

CREATE TABLE order_items  (
       order_item_id NUMBER NOT NULL,
       order_id NUMBER not null,
       order_item VARCHAR2(100),
       CONSTRAINT order_item_pk PRIMARY KEY (order_item_id),
       CONSTRAINT order_item_fk FOREIGN KEY (order_id) references orders(order_id) on delete cascade)
       PARTITION by reference (order_item_fk);

CREATE TABLE orders_2018_dec (
       order_id NUMBER,
       order_date DATE,
       CONSTRAINT order_2018_dec_pk PRIMARY KEY (order_id));

INSERT into orders_2018_dec values (1,'01-DEC-2018');

CREATE TABLE order_items_2018_dec (
       order_item_id NUMBER,
       order_id NUMBER NOT NULL,
       order_item VARCHAR2(100),
       CONSTRAINT order_item_2018_dec_pk PRIMARY KEY (order_item_id),
       CONSTRAINT order_item_2018_dec_fk FOREIGN KEY (order_id) references orders_2018_dec (order_id)
on delete cascade);

INSERT into order_items_2018_dec values (1,1,'item A new');
INSERT into order_items_2018_dec values (2,1,'item B new');

REM Display data from reference partitioned tables before exchange
SELECT * FROM orders;
no rows selected

SELECT * FROM order_items;
no rows selected

REM ALTER TABLE using cascading exchange
ALTER TABLE orders
   EXCHANGE PARTITION p_2018_dec
   WITH TABLE orders_2018_dec
    CASCADE
   UPDATE GLOBAL INDEXES;

REM Display data from reference partitioned tables after exchange
SELECT * FROM orders;
  ORDER_ID ORDER_DAT
---------- ---------
         1 01-DEC-18

SELECT * FROM order_items;
ORDER_ITEM_ID   ORDER_ID ORDER_ITEM
------------- ---------- ------------
            1          1 item A new
            2          1 item B new
```

4.4.5 パーティションおよびサブパーティションのマージについて

ALTER TABLE MERGE PARTITIONおよびSUBPARTITION SQL文を使用して、2つのパーティションまたはサブパーティションのコンテンツをマージします。

元の2つのパーティションまたはサブパーティションと、対応するローカル索引は削除されます。この文は、ハッシュ・パーティション表、またはコンポジット* - ハッシュ・パーティション表のハッシュ・サブパーティションには使用できません。

参照パーティション表のパーティションはマージできません。かわりに、親表でのマージ操作がすべての子表にカスケードされます。ただし、パーティションまたはサブパーティションをマージするために、マスター表でマージ操作を発行する場合は、DEPENDENT TABLES句を使用して依存表に特定のプロパティを設定できます。

ONLINEキーワードをALTER TABLE MERGE PARTITIONおよびSUBPARTITION SQL文とともに使用すると、通常の(ヒープ構成)表に対するオンライン・マージ操作が有効になります。ONLINEキーワードの使用例は、例4-34を参照してください。

関連するパーティションまたはサブパーティションにデータが含まれる場合は、次の表で説明するように、索引はUNUSABLEとマークされます。

表のタイプ	索引の動作
通常(ヒープ)	ALTER TABLE 文の一部に UPDATE INDEXES を指定しない場合: ● データベースにより、対応する結果のローカル索引パーティションまたはサブパーティションがすべて UNUSABLE とマークされます。 ● グローバル索引またはパーティション・グローバル索引のすべてのパーティションが UNUSABLE とマークされ、再作成する必要があります。
索引構成	● データベースにより、対応する結果のローカル索引パーティションがすべて UNUSABLE とマークされます。 ● すべてのグローバル索引は使用可能なままです。

この項では、次の項目について説明します。

- レンジ・パーティションのマージ
- 時間隔パーティションのマージ
- リスト・パーティションのマージ
- * - ハッシュ・パーティションのマージ
- * - リスト・パーティションのマージについて
- * - レンジ・パーティションのマージについて
- 複数のパーティションのマージ

関連項目:

Oracle Database SQL言語リファレンス

4.4.5.1 レンジ・パーティションのマージ

隣接する2つのレンジ・パーティションの内容を、1つのパーティションにマージできます。

隣接しないレンジ・パーティションはマージできません。結果のパーティションは、マージされた2つのパーティションの高い方の上限を継承します。

レンジ・パーティションをマージする理由の1つは、大規模なパーティションで、履歴データをオンラインに保つためです。たとえば、最も古いパーティションが週次パーティションにロールアップされ、そのパーティションは月次パーティションにロールアップされる日次パーティションを作成できます。

例4-34は、ONLINEキーワードを使用したレンジ・パーティションのマージ例を示しています。

例4-34 レンジ・パーティションのマージ

```
-- First, create a partitioned table with four partitions, each on its own
-- tablespace, partitioned by range on the date column
--
CREATE TABLE four_seasons
(       one DATE,
        two VARCHAR2(60),
      three NUMBER
)
PARTITION BY RANGE (one)
(
PARTITION quarter_one
   VALUES LESS THAN ( TO_DATE('01-APR-2017','dd-mon-yyyy'))
   TABLESPACE quarter_one,
PARTITION quarter_two
   VALUES LESS THAN ( TO_DATE('01-JUL-2017','dd-mon-yyyy'))
   TABLESPACE quarter_two,
PARTITION quarter_three
   VALUES LESS THAN ( TO_DATE('01-OCT-2017','dd-mon-yyyy'))
   TABLESPACE quarter_three,
PARTITION quarter_four
   VALUES LESS THAN ( TO_DATE('01-JAN-2018','dd-mon-yyyy'))
   TABLESPACE quarter_four
);
--
-- Create local PREFIXED indexes on four_seasons
-- Prefixed because the leftmost columns of the index match the
-- Partitioning key
--
CREATE INDEX i_four_seasons_l ON four_seasons (one,two)
  LOCAL (
  PARTITION i_quarter_one TABLESPACE i_quarter_one,
  PARTITION i_quarter_two TABLESPACE i_quarter_two,
  PARTITION i_quarter_three TABLESPACE i_quarter_three,
  PARTITION i_quarter_four TABLESPACE i_quarter_four
);


SELECT TABLE_NAME, PARTITION_NAME FROM USER_TAB_PARTITIONS WHERE TABLE_NAME ='FOUR_SEASONS';
TABLE_NAME                       PARTITION_NAME
-------------------------------- ------------------------
FOUR_SEASONS                     QUARTER_FOUR
FOUR_SEASONS                     QUARTER_ONE
FOUR_SEASONS                     QUARTER_THREE
```

```
FOUR_SEASONS                          QUARTER_TWO

-- Next, merge the first two partitions
ALTER TABLE four_seasons
  MERGE PARTITIONS quarter_one, quarter_two INTO PARTITION quarter_two
  UPDATE INDEXES
  ONLINE;

SELECT TABLE_NAME, PARTITION_NAME FROM USER_TAB_PARTITIONS WHERE TABLE_NAME ='FOUR_SEASONS';
TABLE_NAME                          PARTITION_NAME
----------------------------------- -------------------------
FOUR_SEASONS                          QUARTER_FOUR
FOUR_SEASONS                          QUARTER_THREE
FOUR_SEASONS                          QUARTER_TWO
```

ALTER TABLE four_season文からUPDATE INDEXES句を除外した場合は、影響を受けるパーティションのローカル索引を再作成する必要があります。

```
-- Rebuild the index for quarter_two, which has been marked unusable
-- because it has not had all of the data from quarter_one added to it.
-- Rebuilding the index corrects this condition.
--
ALTER TABLE four_seasons MODIFY PARTITION quarter_two
  REBUILD UNUSABLE LOCAL INDEXES;
```

4.4.5.2 時間隔パーティションのマージ

隣接する2つの時間隔パーティションのコンテンツを、1つのパーティションにマージできます。

隣接しない時間隔パーティションはマージできません。また、最初の時間隔パーティションは、最高位のレンジ・パーティションとマージできます。結果のパーティションは、マージされた2つのパーティションの高い方の上限を継承します。

時間隔パーティションをマージすると、遷移点が、マージされた2つのパーティションの高い方の上限に移動します。その結果、時間隔パーティション表のレンジ・セクションが、マージされた2つのパーティションの上限に拡張されます。新しくマージされたパーティションより上限が低いマテリアライズ化された時間隔パーティションは、上限が時間隔の上限によって定義されたレンジ・パーティションに自動的に変換されます。

次の時間隔パーティション表transactionsを例にあげます。

```
CREATE TABLE transactions
( id               NUMBER
, transaction_date DATE
, value            NUMBER
)
PARTITION BY RANGE (transaction_date)
INTERVAL (NUMTODSINTERVAL(1,'DAY'))
( PARTITION p_before_2007 VALUES LESS THAN (TO_DATE('01-JAN-2007','dd-MON-yyyy')));
```

データを表の時間隔セクションに挿入すると、これらの日付用の時間隔パーティションが作成されます。2007年1月15日および2007年1月16日のデータは、隣接する時間隔パーティションに格納されます。

```
INSERT INTO transactions VALUES (1,TO_DATE('15-JAN-2007','dd-MON-yyyy'),100);
INSERT INTO transactions VALUES (2,TO_DATE('16-JAN-2007','dd-MON-yyyy'),600);
INSERT INTO transactions VALUES (3,TO_DATE('30-JAN-2007','dd-MON-yyyy'),200);
```

次に、隣接する2つの時間隔パーティションをマージします。新しいパーティションには、システム生成の名前が付けられます。

```
ALTER TABLE transactions
```

```
MERGE PARTITIONS FOR(TO_DATE('15-JAN-2007','dd-MON-yyyy'))
, FOR(TO_DATE('16-JAN-2007','dd-MON-yyyy'));
```

transactions表の遷移点が2007年1月17日に移動します。時間隔パーティション表のレンジ・セクションには、値が2007年1月1日より少ないものと、2007年1月17日より少ないものの、2つのレンジ・パーティションが含まれます。2007年1月17日より大きい値は、時間隔パーティション表の時間隔部分に分類されます。

4.4.5.3 リスト・パーティションのマージ

リスト・パーティションをマージする場合、マージするパーティションは任意の2つのパーティションでかまいません。

リスト・パーティション化ではパーティションの順序が想定されていないため、レンジ・パーティションのように隣接している必要はありません。結果のパーティションには、元の2つのパーティションのすべてのデータが含まれます。デフォルトのリスト・パーティションをその他のパーティションとマージすると、結果のパーティションがデフォルトのパーティションになります。

次の文では、リスト・メソッドを使用してパーティション化された表の2つのパーティションを、すべての属性を表レベルのデフォルト属性から継承するパーティションにマージします。文にはMAXEXTENTSが指定されています。

```
ALTER TABLE q1_sales_by_region
   MERGE PARTITIONS q1_northcentral, q1_southcentral
   INTO PARTITION q1_central
      STORAGE(MAXEXTENTS 20);
```

元の2つのパーティションの値リストは、次のとおりです。

```
PARTITION q1_northcentral VALUES ('SD','WI')
PARTITION q1_southcentral VALUES ('OK','TX')
```

結果として生成されるsales_westパーティションの値リストは、これら2つのパーティションの値リストを合せたものです。つまり、次のようになります。

```
('SD','WI','OK','TX')
```

4.4.5.4 * - ハッシュ・パーティションのマージ

* - ハッシュ・パーティションをマージすると、サブパーティションが、SUBPARTITIONS *n*またはSUBPARTITION句で指定されたサブパーティション数に再度ハッシュされます。どちらも含まれない場合、表レベルのデフォルトが使用されます。

2つの* - ハッシュ・パーティションをマージする場合と、* - ハッシュ・パーティションを分割する場合では、プロパティの継承方法が異なります。パーティションを分割する場合は親が1つのみであるため、新しいパーティションは元のパーティションからプロパティを継承できます。ただし、パーティションをマージする場合は、表レベルからプロパティを継承する必要があります。

時間隔 - ハッシュ・パーティション表の場合、マージできるのは、隣接する2つの時間隔パーティションか、最高位のレンジ・パーティションと最初の時間隔パーティションのみです。時間隔 - ハッシュ・パーティション表で時間隔をマージする際、遷移点が移動します。

次の例では、2つのレンジ - ハッシュ・パーティションをマージします。

```
ALTER TABLE all_seasons
   MERGE PARTITIONS quarter_1, quarter_2 INTO PARTITION quarter_2
   SUBPARTITIONS 8;
```

関連項目:

- ハッシュ・パーティションの詳細は、「* - ハッシュ・パーティションの分割」を参照してください

- 時間隔パーティションのマージの詳細は、「時間隔パーティションのマージ」を参照してください

4.4.5.5 * - リスト・パーティションのマージについて

パーティションはパーティション・レベルで、サブパーティションはリスト・サブパーティション・レベルでマージできます。

この項では、次の項目について説明します。

- * - リスト・パーティション表のパーティションのマージ
- * - リスト・パーティション表のサブパーティションのマージ

4.4.5.5.1 * - リスト・パーティション表のパーティションのマージ

2つの* - リスト・パーティションをマージする場合、結果の新しいパーティションは、サブパーティション・テンプレートからサブパーティションの説明を継承します(サブパーティション・テンプレートが存在する場合)。サブパーティション・テンプレートがない場合には、新しいパーティションに単一のデフォルトのサブパーティションが作成されます。

時間隔 - リスト・パーティション表の場合、マージできるのは、隣接する2つの時間隔パーティションか、最高位のレンジ・パーティションと最初の時間隔パーティションのみです。時間隔 - リスト・パーティション表で時間隔をマージする際、遷移点が移動します。

次の文では、レンジ - リスト・パーティション化されたstripe_regional_sales表の2つのパーティションをマージします。表には、サブパーティション・テンプレートがあります。

```
ALTER TABLE stripe_regional_sales
   MERGE PARTITIONS q1_1999, q2_1999 INTO PARTITION q1_q2_1999
      STORAGE(MAXEXTENTS 20);
```

指定されていないパーティションでは表レベルのデフォルトが継承されますが、この新しいパーティションにはいくつかの新しい物理属性が指定されています。結果の新しいパーティションq1_q2_1999は、パーティションq2_1999の上限値と、表のサブパーティション・テンプレートの説明からサブパーティションの値リストの説明を継承しています。

結果のパーティションのデータには、両方のパーティションのデータが含まれます。ただし、データベースによりエラーが戻される場合があります。これは、次に示す両方の条件に当てはまる場合には、データが新しいパーティション外にマップされているためです。

このエラー条件は、デフォルトのサブパーティション・テンプレートに必ずデフォルトのパーティションを指定することでなくすことができます。

- マージされたサブパーティションの一部のリテラル値がサブパーティション・テンプレートに含まれていない。
- サブパーティション・テンプレートにデフォルトのパーティション定義が含まれていない。

関連項目:

- * - リスト・パーティション表のパーティションのマージの詳細は、「リスト・パーティションのマージ」を参照してください
- 時間隔パーティションのマージの詳細は、「時間隔パーティションのマージ」を参照してください

4.4.5.5.2 * - リスト・パーティション表のサブパーティションのマージ

同じパーティションに所属する2つの任意のリスト・サブパーティションのコンテンツをマージできます。

結果のサブパーティションの値リストの記述子には、マージされるパーティションの値リストのすべてのリテラル値が含まれます。

次の文では、レンジ - リスト・メソッドを使用してパーティション化された表の2つのサブパーティションを、表領域ts4に配置された新しいサブパーティションにマージします。

```
ALTER TABLE quarterly_regional_sales
   MERGE SUBPARTITIONS q1_1999_northwest, q1_1999_southwest
      INTO SUBPARTITION q1_1999_west
         TABLESPACE ts4;
```

元の2つのパーティションの値リストは、次のとおりです。

- サブパーティションq1_1999_northwestは('WA','OR')
- サブパーティションq1_1999_southwestは('AZ','NM','UT')

結果のサブパーティションの値リストは、次に示すように、これら2つのサブパーティションの値リストを合せたセットで構成されます。

- サブパーティションq1_1999_westの値リストは('WA','OR','AZ','NM','UT')

結果のサブパーティションが配置されている表領域およびサブパーティションの属性は、明示的に指定されている場合を除き、パーティション・レベルのデフォルト属性によって決定されます。既存のサブパーティション名を再利用する場合、新しいサブパーティションは、名前が再利用されるサブパーティションのサブパーティション属性を継承します。

4.4.5.6 * - レンジ・パーティションのマージについて

パーティションはパーティション・レベルで、サブパーティションはレンジ・サブパーティション・レベルでマージできます。

- * - レンジ・パーティション表のパーティションのマージ

4.4.5.6.1 * - レンジ・パーティション表のパーティションのマージ

2つの* - レンジ・パーティションをマージする場合、結果の新しいパーティションは、サブパーティション・テンプレートからサブパーティションの説明を継承します(サブパーティション・テンプレートが存在する場合)。サブパーティション・テンプレートがない場合には、上限がMAXVALUEの単一のサブパーティションが新しいパーティションに作成されます。

時間隔 - レンジ・パーティション表の場合、マージできるのは、隣接する2つの時間隔パーティションか、最高位のレンジ・パーティションと最初の時間隔パーティションのみです。時間隔 - レンジ・パーティション表で時間隔をマージする際、遷移点が移動します。

次の文では、月次の時間隔 - レンジでパーティション化されたorders表の2つのパーティションをマージします。表には、サブパーティション・テンプレートがあります。

```
ALTER TABLE orders
MERGE PARTITIONS FOR(TO_DATE('01-MAR-2007','dd-MON-yyyy')),
FOR(TO_DATE('01-APR-2007','dd-MON-yyyy'))
INTO PARTITION p_pre_may_2007;
```

2007年3月および2007年4月のパーティションが時間隔 - レンジ・パーティション表の時間隔セクションにある場合、マージ操作により、遷移点が2007年5月1日に移動します。

パーティションp_pre_may_2007のサブパーティションは、サブパーティション・テンプレートからプロパティを継承します。結果のパーティションのデータには、両方のパーティションのデータが含まれます。ただし、データベースによりエラーが戻される場合があります。これは、次に示す両方の条件に当てはまる場合には、データが新しいパーティション外にマップされているためです。

このエラー条件は、上限がMAXVALUEのサブパーティションをサブパーティション・テンプレートに必ず指定することでなくすことができます。

- マージされたサブパーティションの一部のレンジ値がサブパーティション・テンプレートに含まれていない。
- サブパーティション・テンプレートに、上限がMAXVALUEのサブパーティションの定義がない。

関連項目:

- * - レンジ・パーティション表のパーティションのマージの詳細は、「レンジ・パーティションのマージ」を参照してください
- 時間隔パーティションのマージの詳細は、「時間隔パーティションのマージ」を参照してください

4.4.5.7 複数のパーティションのマージ

2つ以上のパーティションまたはサブパーティションのコンテンツを新しい1つのパーティションまたはサブパーティションにマージし、ALTER TABLE SQL文のMERGE PARTITIONSおよびMERGE SUBPARTITIONS句を使用して元のパーティションまたはサブパーティションを削除できます。

MERGE PARTITIONSおよびMERGE SUBPARTITIONS句は、MERGE PARTITIONおよびMERGE SUBPARTITION句と同義です。

たとえば、次のSQL文は、4つのパーティションを1つのパーティションにマージし、マージされた4つのパーティションを削除します。

```
ALTER TABLE t1 MERGE PARTITIONS p01, p02, p03, p04 INTO p0;
```

複数のレンジ・パーティションをマージする場合、パーティションは隣接し、パーティション・バウンド値の昇順で指定する必要があります。新しいパーティションは、元のパーティションの最高位のパーティション上限を継承します。

TO構文を使用して複数のレンジ・パーティションをマージする場合、マージする最下位および最高位のパーティションを指定できます。指定されたパーティション間のすべてのパーティション(指定されたパーティションを含む)は、ターゲット・パーティションにマージされます。この構文は、リストおよびシステム・パーティションに使用できません。

たとえば、次のSQL文は、パーティションp01からp04をパーティションp0にマージします。

```
ALTER TABLE t1 MERGE PARTITIONS p01 TO p04 INTO p0;
```

パーティションの順序が想定されていないため、マージするリスト・パーティションおよびシステム・パーティションは隣接する必要がありません。複数のリスト・パーティションをマージする場合、結果のパーティションの値リストは、マージするすべてのパーティションのパーティション値リストをまとめたものです。他のリスト・パーティションとマージするDEFAULTリスト・パーティションは、DEFAULTパーティションになります。

コンポジット・パーティション表の複数のパーティションをマージする場合、結果の新しいパーティションは、存在する場合にサブパーティション・テンプレートからサブパーティションの説明を継承します。サブパーティション・テンプレートが存在しない場合、新しいパーティションのレンジ・サブパーティションから1つのMAXVALUEサブパーティションまたはリスト・パーティションから1つのDEFAULTサブパーティションが作成されます。コンポジット・パーティション表の複数のサブパーティションをマージする場合、マージするサブパーティションは同じパーティションに属する必要があります。

複数のパーティションをマージする場合、ローカルおよびグローバル索引操作および指定されていない物理属性を継承するセマンティックは、2つのパーティションのマージで同じです。

次のSQL文では、レンジ・パーティション表salesの4つのパーティションがマージされます。最も古い年の4四半期に対応するこれらの4つのパーティションは、その年の売上データ全体を含む単一のパーティションにマージされます。

```
ALTER TABLE sales
  MERGE PARTITIONS sales_q1_2009, sales_q2_2009, sales_q3_2009, sales_q4_2009
  INTO PARTITION sales_2009;
```

前述のSQL文を次のSQL文にリライトしても、同じ結果を取得できます。

```
ALTER TABLE sales
  MERGE PARTITIONS sales_q1_2009 TO sales_q4_2009
  INTO PARTITION sales_2009;
```

4.4.6 表、パーティションおよびサブパーティションの属性の変更について

このトピックでは、表、パーティションおよびサブパーティションの属性の変更について説明します。

- デフォルトの属性の変更について
- パーティションの実際の属性の変更について

4.4.6.1 デフォルトの属性の変更について

表またはコンポジット・パーティション表のパーティションのデフォルト属性を変更できます。

デフォルト属性を変更する場合、新しい属性の影響を受けるのは、作成される今後のパーティションまたはサブパーティションのみです。新しいパーティションまたはサブパーティションの作成時には、デフォルト値を明示的に上書きできます。参照パーティション表のデフォルト属性を変更できます。

この項では、次の項目について説明します。

- 表のデフォルト属性の変更
- パーティションのデフォルト属性の変更
- 索引パーティションのデフォルト属性の変更

4.4.6.1.1 表のデフォルト属性の変更

ALTER TABLEのMODIFY DEFAULT ATTRIBUTES句を使用して、レンジ、ハッシュ、リスト、時間隔または参照パーティションに継承されるデフォルト属性を変更できます。

ハッシュ・パーティション表の場合、変更できるのはTABLESPACE属性のみです。

4.4.6.1.2 パーティションのデフォルト属性の変更

サブパーティションの作成時に継承されたデフォルト属性を変更するには、ALTER TABLE MODIFY DEFAULT ATTRIBUTES FOR PARTITIONを使用します。

次の文では、レンジ - ハッシュ・パーティション表のパーティションp1の後続のサブパーティションが配置されるTABLESPACEを変更します。

```
ALTER TABLE employees_subpartitions
     MODIFY DEFAULT ATTRIBUTES FOR PARTITION p1 TABLESPACE ts1;
```

TABLESPACEを除き、レンジ - ハッシュ・パーティション表のすべてのサブパーティションでは同じ属性を共有する必要があるため、これは変更できる唯一の属性です。

作成されていない時間隔パーティションのデフォルト属性は変更できません。時間隔パーティション表の今後のサブパーティションの作成方法を変更するには、サブパーティション・テンプレートを変更する必要があります。

4.4.6.1.3 索引パーティションのデフォルト属性の変更

表パーティションと同じような方法で、レンジ・パーティション・グローバル索引のパーティション、またはパーティション表のローカル索引パーティションにより継承されるデフォルト属性を変更できます。

これには、ALTER INDEX MODIFY DEFAULT ATTRIBUTES文を使用します。コンポジット・パーティション表のサブパーティションにより継承されるデフォルト属性を変更する場合は、ALTER INDEX MODIFY DEFAULT ATTRIBUTES FOR PARTITION文を使用します。

4.4.6.2 パーティションの実際の属性の変更について

表または索引の既存のパーティションの属性を変更できます。

TABLESPACE属性は変更できません。新しい表領域にパーティションまたはサブパーティションを移動する場合は、ALTER TABLE MOVE PARTITION/SUBPARTITIONを使用します。

この項では、次の項目について説明します。

- レンジまたはリスト・パーティションの実際の属性の変更
- ハッシュ・パーティションの実際の属性の変更
- サブパーティションの実際の属性の変更
- 索引パーティションの実際の属性の変更

4.4.6.2.1 レンジまたはリスト・パーティションの実際の属性の変更

レンジ・パーティションまたはリスト・パーティションの既存の属性を変更するには、ALTER TABLE MODIFY PARTITION文を使用します。

セグメント属性(TABLESPACEを除く)の変更、エクステントの割当てや割当て解除、ローカル索引パーティションへのUNUSABLEのマーク付け、またはUNUSABLEとマークされたローカル索引の再作成が可能です。

* - ハッシュ・パーティション表のレンジ・パーティションの場合は、次の内容に注意してください。

- エクステントの割当てまたは割当て解除を行うと、このアクションは指定されたパーティションのすべてのサブパーティションに対して実行されます。
- 同様に、その他の属性を変更した場合も、対応する変更がそのパーティションのすべてのサブパーティションの属性に対して行われます。パーティション・レベルのデフォルト属性も変更されます。既存のサブパーティションの属性が変更されないようにするには、MODIFY DEFAULT ATTRIBUTES文のFOR PARTITION句を使用します。

次に、パーティションの実際の属性の変更例をいくつか示します。

この例では、表salesのレンジ・パーティションsales_q1のMAXEXTENTS記憶域属性を変更します。

```
ALTER TABLE sales MODIFY PARTITION sales_q1
     STORAGE (MAXEXTENTS 10);
```

次の例では、レンジ - ハッシュ・パーティション表scubagearのパーティションts1のローカル索引サブパーティションがすべてUNUSABLEとマークされます。

```
ALTER TABLE scubagear MODIFY PARTITION ts1 UNUSABLE LOCAL INDEXES;
```

時間隔パーティション表の場合、変更できるのは、作成済のレンジ・パーティションまたは時間隔パーティションの実際の属性のみです。

4.4.6.2.2 ハッシュ・パーティションの実際の属性の変更

ALTER TABLE MODIFY PARTITION文を使用して、ハッシュ・パーティションの属性を変更できます。

ただし、個々のハッシュ・パーティションの物理属性は(TABLESPACEを除き)すべて同一である必要があるため、実行できるのは

次の操作のみです。

- 新規エクステントの割当て
- 未使用のエクステントの割当て解除
- ローカル索引サブパーティションへのUNUSABLEのマーク付け
- UNUSABLEとマークされたローカル索引サブパーティションの再作成

次の例では、表のハッシュ・パーティションp1に関連付けられた使用できないローカル索引パーティションを再作成します。

```
ALTER TABLE departments_rebuild_index MODIFY PARTITION p1
     REBUILD UNUSABLE LOCAL INDEXES;
```

4.4.6.2.3 サブパーティションの実際の属性の変更

ALTER TABLEのMODIFY SUBPARTITION句を使用すると、前の項でパーティションに関してリストされていたのと同じアクションを実行できますが、実行できるのは、特定のコンポジット・パーティション表のサブパーティション・レベルにおいてです。

例:

```
ALTER TABLE employees_rebuild_index MODIFY SUBPARTITION p3_s1
     REBUILD UNUSABLE LOCAL INDEXES;
```

4.4.6.2.4 索引パーティションの実際の属性の変更

ALTER INDEXのMODIFY PARTITION句を使用すると、索引のパーティションまたはサブパーティションの実際の属性を変更できます。

ルールは表パーティションのルールとよく似ています。ALTER INDEXのMODIFY PARTITION句とは異なり、使用不可の索引パーティションを作成する副句はありませんが、索引のパーティションまたはサブパーティションを結合する副句があります。ここでの結合は索引ブロックのマージを意味します。これは再利用のために解放できます。

MODIFY PARTITION句を使用して、ローカル索引のサブパーティションに対する記憶域の割当てや割当て解除、またはそれらをUNUSABLEとマークすることもできます。

4.4.7 リスト・パーティションの変更について

このトピックでは、パーティションおよびサブパーティションのリストの値の変更について説明します。

- リスト・パーティションの変更について: 値の追加
- リスト・パーティションの変更について: 値の削除

4.4.7.1 リスト・パーティションの変更について: 値の追加

リスト・パーティション化では、オプションで、定義された値リストのリテラル値を追加できます。

この項では、次の項目について説明します。

- リスト・パーティションへの値の追加
- リスト・サブパーティションへの値の追加

4.4.7.1.1 リスト・パーティションへの値の追加

既存のパーティションの値リストを拡張するには、ALTER TABLE文のMODIFY PARTITION ADD VALUES句を使用します。

追加するリテラル値は、その他のパーティションの値リストに含まれていない必要があります。対応するローカル索引パーティションのパーティションの値リストはそれに応じて拡張され、グローバル索引や、グローバルまたはローカル索引パーティションは使用可能なままです。

次の文では、既存のパーティション・リストに、一連の新しい州コード(OK、KS)を追加します。

```
ALTER TABLE sales_by_region
   MODIFY PARTITION region_south
      ADD VALUES ('OK', 'KS');
```

デフォルトのパーティションがあると、その他のパーティションへの値の追加時にパフォーマンスに影響があります。これは、リスト・パーティションに値を追加するために、追加する値がデフォルトのパーティションに存在しないことをデータベースで確認する必要があるためです。いずれかの値がデフォルト・パーティションに存在する場合は、エラーが表示されます。

ノート:

データベースにより、追加されるリテラル値に対応する行がデフォルトのパーティションに存在するかどうかを確認する問合せが実行されます。そのため、表にローカルの同一キー索引を作成することをお薦めします。これにより、問合せの実行および操作全体が高速化されます。

デフォルトのリスト・パーティションには値を追加できません。

4.4.7.1.2 リスト・サブパーティションへの値の追加

既存のサブパーティションの値リストを拡張するには、ALTER TABLE文のMODIFY SUBPARTITION ADD VALUES句を使用します。

この操作は、「リスト・パーティションの変更について: 値の追加」に説明されている操作と基本的に同じですが、MODIFY PARTITION句ではなくMODIFY SUBPARTITION句を使用します。たとえば、サブパーティションq1_1999_southeastの値リストにあるリテラル値のレンジを拡張するには、次の文を使用します。

```
ALTER TABLE quarterly_regional_sales
   MODIFY SUBPARTITION q1_1999_southeast
      ADD VALUES ('KS');
```

追加するリテラル値が、所有パーティション内のその他のサブパーティションの値リストに含まれていない必要があります。ただし、表内にあるその他のパーティションのサブパーティションの値リストのリテラル値を複製したものであってもかまいません。

時間隔 - リスト・コンポジット・パーティション表の場合、作成済のレンジ・パーティションか時間隔パーティションのサブパーティションにのみ値を追加できます。まだ作成されていない時間隔パーティションのサブパーティションに値を追加するには、サブパーティション・テンプレートを変更する必要があります。

4.4.7.2 リスト・パーティションの変更について: 値の削除

リスト・パーティション化では、オプションで、定義された値リストのリテラル値を削除できます。

この項では、次の項目について説明します。

- リスト・パーティションからの値の削除
- リスト・サブパーティションからの値の削除

4.4.7.2.1 リスト・パーティションからの値の削除

既存のパーティションの値リストからリテラル値を削除するには、ALTER TABLE文のMODIFY PARTITION DROP VALUES句を使用します。

この文は常に検証しながら実行されます。つまり、削除する一連の値に対応する行がパーティションに存在するかどうかが確認されます。そのような行が見つかると、データベースによってエラー・メッセージが返され、操作が失敗します。必要な場合は、DELETE文を使用して、対応する行を表から削除した後で値の削除を試行します。

> ノート:
>
> パーティションを説明する値リストからすべてのリテラル値を削除することはできません。かわりに、ALTER TABLE DROP PARTITION 文を使用してください。

対応するローカル索引パーティションのパーティションの値リストには新しい値リストが反映され、グローバル索引や、グローバルまたはローカル索引パーティションは使用可能なままです。

次の文では、既存のパーティションの値リストから、一連の州コード(OKおよびKS)を削除します。

```
ALTER TABLE sales_by_region
   MODIFY PARTITION region_south
      DROP VALUES ('OK', 'KS');
```

> ノート:
>
> データベースにより、削除されるリテラル値に対応する行がパーティションに存在するかどうかを確認する問合せが実行されます。そのため、表にローカルの同一キー索引を作成することをお薦めします。これにより、問合せおよび操作全体が高速化されます。

デフォルトのリスト・パーティションからは値を削除できません。

4.4.7.2.2 リスト・サブパーティションからの値の削除

既存のサブパーティションの値リストからリテラル値を削除するには、ALTER TABLE文のMODIFY SUBPARTITION DROP VALUES句を使用します。

この操作は、「リスト・パーティションの変更について: 値の削除」に説明されている操作と基本的に同じですが、MODIFY PARTITION句ではなくMODIFY SUBPARTITION句を使用します。たとえば、サブパーティションq1_1999_southeastの値リストにある一連のリテラル値を削除するには、次の文を使用します。

```
ALTER TABLE quarterly_regional_sales
   MODIFY SUBPARTITION q1_1999_southeast
      DROP VALUES ('KS');
```

時間隔 - リスト・コンポジット・パーティション表の場合、作成済のレンジ・パーティションか時間隔パーティションのサブパーティションからのみ値を削除できます。まだ作成されていない時間隔パーティションのサブパーティションから値を削除するには、サブパーティション・テンプレートを変更する必要があります。

4.4.8 パーティション化戦略の変更について

ALTER TABLE MODIFY PARTITION SQL文を使用して、通常の(ヒープ構成)表のパーティション化戦略を変更できます。

パーティション化戦略の変更(ハッシュ・パーティション化からコンポジット・レンジ - ハッシュ・パーティション化に変更するなど)は、オフラインまたはオンラインで実行できます。オンライン・モードで実行した場合、進行中のDML操作は変更の影響を受けません。オフライン・モードで実行した場合、変更中の同時DML操作は許可されません。

索引は、表の変更の際にメンテナンスされます。パーティション化戦略を変更する場合、索引列が新しいパーティション化キーのプリフィックスであるすべての未指定の索引は自動的にローカル・パーティション索引に変換されます。そうでない場合、索引はグローバル索引に変換されます。

変更操作は、ドメイン索引ではサポートされていません。UPDATE INDEXES句では、索引のリストが最初に定義されている列、索引の一意性プロパティ、またはその他の索引プロパティを変更することはできません。

非パーティション表からパーティション表への変換の詳細は、「パーティション表への非パーティション表の変換」を参照してください。

例4-35に、レンジ・パーティション表をコンポジット・レンジ- ハッシュ・パーティション表にオンラインで変換するためのALTER TABLEの使用方法を示します。この例では、ALTER TABLEでの変更中に索引が更新されます。

Live SQL:

Oracle Live SQL の表のパーティション化戦略の変更で関連する例を参照して実行してください。

例4-35 パーティション化戦略の変更

```
CREATE TABLE mod_sales_partitioning
  ( prod_id       NUMBER        NOT NULL,
    cust_id       NUMBER        NOT NULL,
    time_id       DATE          NOT NULL,
    channel_id    NUMBER        NOT NULL,
    promo_id      NUMBER        NOT NULL,
    quantity_sold NUMBER(10,2)  NOT NULL,
    amount_sold   NUMBER(10,2)  NOT NULL
  )
  PARTITION BY RANGE (time_id)
  (PARTITION sales_q1_2017 VALUES LESS THAN (TO_DATE('01-APR-2017','dd-MON-yyyy')),
   PARTITION sales_q2_2017 VALUES LESS THAN (TO_DATE('01-JUL-2017','dd-MON-yyyy')),
   PARTITION sales_q3_2017 VALUES LESS THAN (TO_DATE('01-OCT-2017','dd-MON-yyyy')),
   PARTITION sales_q4_2017 VALUES LESS THAN (TO_DATE('01-JAN-2018','dd-MON-yyyy'))
 );

CREATE INDEX i1_cust_id_indx ON mod_sales_partitioning (cust_id) LOCAL;
CREATE INDEX i2_time_id_indx ON mod_sales_partitioning (time_id);
CREATE INDEX i3_prod_id_indx ON mod_sales_partitioning (prod_id);

SELECT TABLE_NAME, PARTITIONING_TYPE FROM USER_PART_TABLES WHERE TABLE_NAME ='MOD_SALES_PARTITIONING';
TABLE_NAME                PARTITION_NAME
------------------------- --------------
MOD_SALES_PARTITIONING    RANGE

SELECT TABLE_NAME, PARTITION_NAME FROM USER_TAB_PARTITIONS WHERE TABLE_NAME ='MOD_SALES_PARTITIONING';
TABLE_NAME                PARTITION_NAME
------------------------- --------------
MOD_SALES_PARTITIONING    SALES_Q1_2017
MOD_SALES_PARTITIONING    SALES_Q2_2017
MOD_SALES_PARTITIONING    SALES_Q3_2017
```

```
MOD_SALES_PARTITIONING    SALES_Q4_2017
...

ALTER TABLE mod_sales_partitioning
 MODIFY
 PARTITION BY RANGE (time_id) SUBPARTITION BY HASH (cust_id)
  SUBPARTITIONS 8
 ( PARTITION  sales_q1_2017 VALUES LESS THAN (TO_DATE('01-APR-2017','dd-MON-yyyy')),
    PARTITION sales_q2_2017 VALUES LESS THAN (TO_DATE('01-JUL-2017','dd-MON-yyyy')),
    PARTITION sales_q3_2017 VALUES LESS THAN (TO_DATE('01-OCT-2017','dd-MON-yyyy')),
    PARTITION sales_q4_2017 VALUES LESS THAN (TO_DATE('01-JAN-2018','dd-MON-yyyy')))
    ONLINE
  UPDATE INDEXES
   ( i1_cust_id_indx LOCAL,
     i2_time_id_indx GLOBAL PARTITION BY RANGE (time_id)
     (PARTITION ip1_indx VALUES LESS THAN (MAXVALUE) ) );

SELECT TABLE_NAME, PARTITIONING_TYPE, SUBPARTITIONING_TYPE FROM USER_PART_TABLES WHERE TABLE_NAME
='MOD_SALES_PARTITIONING';
TABLE_NAME                  PARTITION      SUBPARTIT
--------------------------- -------------- ----------
MOD_SALES_PARTITIONING      RANGE          HASH

SELECT TABLE_NAME, PARTITION_NAME, SUBPARTITION_NAME FROM USER_TAB_SUBPARTITIONS WHERE TABLE_NAME
='MOD_SALES_PARTITIONING';
TABLE_NAME                  PARTITION_NAME     SUBPARTITION_NAME
--------------------------- ------------------ ------------------
MOD_SALES_PARTITIONING      SALES_Q1_2017      SYS_SUBP567
MOD_SALES_PARTITIONING      SALES_Q1_2017      SYS_SUBP568
MOD_SALES_PARTITIONING      SALES_Q1_2017      SYS_SUBP569
MOD_SALES_PARTITIONING      SALES_Q1_2017      SYS_SUBP570
...
```

4.4.9 パーティションおよびサブパーティションの移動について

パーティションの物理記憶域属性を変更するには、ALTER TABLE文でMOVE PARTITION句を使用します。

ALTER TABLE文のMOVE PARTITION句は、次の目的に使用できます。

- データを再クラスタリングして断片化を減らす場合
- 別の表領域にパーティションを移動する場合
- 作成時間属性の変更
- 表圧縮を使用してデータを圧縮形式で保存する場合

通常は、ALTER TABLE/INDEX MODIFY PARTITION文を使用して、一度にパーティションの物理記憶域属性を変更できます。ただし、TABLESPACEなどの一部の物理属性は、MODIFY PARTITIONを使用して変更できません。そのような場合には、MOVE PARTITION句を使用します。表圧縮などのその他の属性を変更すると、既存のデータではなく、後続の記憶域のみが影響されます。

移動するパーティションにデータが含まれる場合は、次の表で説明するように、索引はUNUSABLEとマークされます。

表のタイプ	索引の動作
通常(ヒープ)	ALTER TABLE 文の一部に UPDATE INDEXES を指定しない場合:

表のタイプ	索引の動作
	● 各ローカル索引の一致するパーティションは UNUSABLE とマークされます。MOVE PARTITION の発行後にそれらの索引パーティションを再作成する必要があります。 ● すべてのグローバル索引またはパーティション・グローバル索引のすべてのパーティションが UNUSABLE とマークされます。
索引構成	移動するパーティションに定義された任意のローカルまたはグローバル索引は、主キー・ベースの論理行 ID であるため使用可能なままです。ただし、これらの行 ID の不確定情報は不適切になります。

この項では、次の項目について説明します。

- 表パーティションの移動
- サブパーティションの移動
- 索引パーティションの移動

関連項目:

- ALTER TABLE MOVE 文の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください
- 表およびパーティションの移動の詳細は、『Oracle Database 管理者ガイド』を参照してください

4.4.9.1 表パーティションの移動

パーティションを移動するには、MOVE PARTITION句を使用します。

たとえば、(I/Oを調整するために)最もアクティブなパーティションをアクションは記録せずにデータを圧縮し、一連の独自のディスクに存在する表領域に移動するには、次の文を発行します。

```
ALTER TABLE parts MOVE PARTITION depot2
     TABLESPACE ts094 NOLOGGING COMPRESS;
```

この文では、新しい表領域を指定しない場合でも、古いパーティション・セグメントが削除され、新しいセグメントが作成されます。

パーティション化索引構成表のパーティションを移動する場合は、MOVE PARTITION句の一部としてMAPPING TABLE句を指定できます。これにより、マッピング表のパーティションは表パーティションとともに新しい場所に移動されます。

時間隔または時間隔 - *パーティション表の場合、移動できるのは、マテリアライズ化済のレンジ・パーティションまたは時間隔パーティションのみです。パーティションの移動操作では、時間隔または時間隔 - *パーティション表の遷移点は移動されません。

マスター表のパーティションには関係なく、参照パーティション表のパーティションを移動できます。

4.4.9.2 サブパーティションの移動

サブパーティションを移動するには、MOVE SUBPARTITION句を使用します。

次の文では、表のサブパーティションにあるデータの移動方法を示します。この例では、PARALLEL句も指定されています。

```
ALTER TABLE scuba_gear MOVE SUBPARTITION bcd_types
     TABLESPACE tbs23 PARALLEL (DEGREE 2);
```

マスター表のサブパーティションには関係なく、参照パーティション表のサブパーティションを移動できます。

4.4.9.3 索引パーティションの移動

通常の表に対してALTER TABLE MOVE PARTITION文を実行すると、グローバル索引のすべてのパーティションがUNUSABLEとマークされます。

ALTER INDEX REBUILD PARTITION文を使用して各パーティションを再作成して、索引全体を再作成できます。これらの再作成は同時に実行できます。

単純に索引を削除して再作成することもできます。

4.4.10 索引パーティションの再作成について

索引の再作成には複数の利点があります。

次に、索引パーティションを再作成する理由をいくつか示します。

- 領域のリカバリおよびパフォーマンスの向上
- メディア障害により破損した索引パーティションの修復
- SQL*Loaderまたはインポート・ユーティリティを使用して基礎となる表パーティションをロードした後のローカル索引パーティションの再作成
- UNUSABLEとマークされた索引パーティションの再作成
- Bツリー索引のキー圧縮の有効化

次の項では、索引パーティションおよびサブパーティションを再作成するためのオプションを説明します。

この項では、次の項目について説明します。

- グローバル索引パーティションの再作成について
- ローカル索引パーティションの再作成について

4.4.10.1 グローバル索引パーティションの再作成について

グローバル索引パーティションは、いくつかの方法で再作成できます。

- ALTER INDEX REBUILD PARTITION文を発行して各パーティションを再作成する方法(再作成は同時に実行できます)。
- グローバル索引全体を削除して再作成する方法。表がスキャンされるのは一度のみであるため、この方法はより効率的です。

索引付きのパーティション表におけるほとんどのメンテナンス操作では、DDL文にUPDATE INDEXESを指定することにより、オプションで索引の再作成を回避できます。

4.4.10.2 ローカル索引パーティションの再作成について

ローカル索引パーティションは、いくつかの方法で再作成できます。

ALTER INDEXまたはALTER TABLEを使用し、次のようにしてローカル索引を再作成します。

- ALTER INDEX REBUILD PARTITION/SUBPARTITION

  この文では、索引パーティションまたはサブパーティションが無条件に再作成されます。

- ALTER TABLE MODIFY PARTITION/SUBPARTITION REBUILD UNUSABLE LOCAL INDEXES

  この文では、表のパーティションまたはサブパーティションに対する使用不可の索引がすべて検索され、それらが再作成されます。UNUSABLEとマークされた索引パーティションのみが再作成されます。

次の各項では、索引の再作成の例について説明します。

- ALTER INDEXを使用したパーティションの再作成
- ALTER TABLEを使用した索引パーティションの再作成

4.4.10.2.1 ALTER INDEXを使用したパーティションの再作成

ALTER INDEX REBUILD PARTITION文によって、索引の1つのパーティションが再作成されます。

これは、コンポジット・パーティション表には使用できません。このコマンドで再作成できるのは実際の物理セグメントのみです。索引を再作成するとき、パーティションの新しい表領域への移動や属性の変更を選択することもできます。

コンポジット・パーティション表では、ALTER INDEX REBUILD SUBPARTITIONを使用して、索引のサブパーティションを再作成します。サブパーティションを別の表領域に移動するか、PARALLEL句を指定できます。次の文では、表のローカル索引のサブパーティションを再作成し、索引サブパーティションを別の表領域に移動します。

```
ALTER INDEX scuba
   REBUILD SUBPARTITION bcd_types
   TABLESPACE tbs23 PARALLEL (DEGREE 2);
```

4.4.10.2.2 ALTER TABLEを使用した索引パーティションの再作成

ALTER TABLE MODIFY PARTITIONのREBUILD UNUSABLE LOCAL INDEXES句により、使用できない索引パーティションを再作成できます。

ただし、文では、索引パーティションの再作成の新規属性を指定できません。次の例では、表scubagearのパーティションp1の使用できないローカル索引パーティションを検出して再作成します。

```
ALTER TABLE scubagear
   MODIFY PARTITION p1 REBUILD UNUSABLE LOCAL INDEXES;
```

ALTER TABLE MODIFY SUBPARTITIONは、使用できないローカル索引サブパーティションの再作成のための句です。

4.4.11 パーティションおよびサブパーティション名の変更について

表と索引の両方のパーティションおよびサブパーティション名を変更できます。

パーティション名を変更する理由の1つは、別のメンテナンス操作でパーティションに割り当てられたデフォルトのシステム名とは対照的に、意味のわかる名前を割り当てるためです。

すべてのパーティション化メソッドで、パーティションを識別するためのFOR(*value*)メソッドがサポートされています。このメソッドを使用して、システム生成されたパーティション名をより意味のわかりやすい名前に変更できます。これは、時間隔または時間隔 - *パーティション表に特に便利です。

参照パーティション化されたマスター表および子表のパーティションやサブパーティション名を個別に変更できます。マスター表での名前の変更操作は、子表にカスケードされません。

この項では、次の項目について説明します。

- 表パーティション名の変更
- 表サブパーティション名の変更
- 索引パーティション名の変更について

4.4.11.1 表パーティション名の変更

ALTER TABLE RENAME PARTITION部門を使用して、レンジ、ハッシュまたはリスト・パーティションの名前を変更できます。

例:

```
ALTER TABLE scubagear RENAME PARTITION sys_p636 TO tanks;
```

4.4.11.2 表サブパーティション名の変更

表のサブパーティションに新しい名前を割り当てられます。

この場合は、ALTER TABLE RENAME SUBPARTITION構文を使用します。

4.4.11.3 索引パーティション名の変更について

ALTER INDEX文により、索引パーティションおよびサブパーティションの名前を変更できます。

- 索引パーティション名の変更
- 索引サブパーティション名の変更

4.4.11.3.1 索引パーティション名の変更

索引パーティション名を変更するには、ALTER INDEX RENAME PARTITION文を使用します。

ALTER INDEX文では、パーティションを識別するためのFOR(*value*)を使用できません。名前の変更操作では、元のパーティション名を使用する必要があります。

4.4.11.3.2 索引サブパーティション名の変更

索引サブパーティション名を変更するには、ALTER INDEX RENAME SUBPARTITION文を使用します。

次の文では、基礎となる表にパーティションを追加した結果、システム生成名が付けられたサブパーティションの名前を変更する方法を示します。

```
ALTER INDEX scuba RENAME SUBPARTITION sys_subp3254 TO bcd_types;
```

4.4.12 パーティションおよびサブパーティションの分割について

パーティションの内容を2つの新しいパーティションに分割できます。

ALTER TABLEまたはALTER INDEX文のSPLIT PARTITION句は、パーティションのコンテンツを2つの新しいパーティションに再分散するために使用されます。パーティションが大きくなりすぎてバックアップ、リカバリまたはメンテナンス操作の完了に長い時間がかかるようになった場合や、単純にパーティションのデータが増えすぎた場合に、再分散を検討します。SPLIT PARTITION句を使用して、I/Oロードを再分散することもできます。この句は、ハッシュ・パーティションまたはサブパーティションには使用できません。

分割するパーティションにデータが含まれる場合は、次の表で説明するように、索引はUNUSABLEとマークされます。

表のタイプ	索引の動作

表のタイプ	索引の動作
通常(ヒープ)	ALTER TABLE 文の一部に UPDATE INDEXES を指定しない場合: ● データベースにより、各ローカル索引の(2 つある)新しいパーティションが UNUSABLE とマークされます。 ● すべてのグローバル索引またはパーティション・グローバル索引のすべてのパーティションが UNUSABLE とマークされ、再作成する必要があります。
索引構成	● データベースにより、各ローカル索引の(2 つある)新しいパーティションが UNUSABLE とマークされます。 ● すべてのグローバル索引は使用可能なままです。

親表を除いて、参照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を分割できません。親表のパーティションまたはサブパーティションを分割すると、すべての子表にカスケードされます。ただし、パーティションまたはサブパーティションを分割するために、マスター表でSPLIT文を発行する場合は、DEPENDENT TABLES句を使用して依存表に特定のプロパティを設定できます。

SPLIT操作によるパーティション・メンテナンスは、キーワードONLINEが指定されたオンライン操作としてヒープ構成表でサポートされているため、パーティション・メンテナンス操作が進行中の同時DML操作が可能になります。

ONLINE操作では、UPDATE INDEXES句を指定したかどうかに関係なく、索引の分割はデフォルトで常に更新されます。

SPLIT操作でのキーワードONLINEの使用の例は、例4-37を参照してください。

この項では、次の項目について説明します。

- レンジ・パーティション表のパーティションの分割
- リスト・パーティション表のパーティションの分割
- 時間隔パーティション表のパーティションの分割
- * - ハッシュ・パーティションの分割
- * - リスト・パーティション表のパーティションの分割
- * - レンジ・パーティションの分割
- 索引パーティションの分割
- 複数のパーティションへの分割
- 高速なSPLIT PARTITIONおよびSPLIT SUBPARTITION操作

関連項目:

Oracle Database SQL言語リファレンス

4.4.12.1 レンジ・パーティション表のパーティションの分割

ALTER TABLE SPLIT PARTITION文を使用して、レンジ・パーティションを分割できます。

このSQL文では、パーティションの分割点である、パーティションのレンジ内のパーティション化キー列の値を指定する必要があります。

分割の結果得られるパーティションに、オプションで新しい属性を指定できます。表にローカル索引が定義されている場合、この文では、各ローカル索引の一致するパーティションも分割されます。

新しいパーティションの名前を指定しない場合、データベースにより、SYS_P*n*という書式の名前が割り当てられます。データ・ディクショナリを調査して、新しいローカル索引パーティションに割り当てられた名前を検出できます。それらの名前を変更する必要がある場合もあります。指定しない属性は、元のパーティションから継承されます。

例4-36 レンジ・パーティション表のパーティションの分割、および索引の再作成

この例のfee_katyは、表vet_catsのパーティションです。これにはローカル索引jaf1があります。この表にはグローバル索引vetもあります。vetには2つのパーティションvet_partaとvet_partbが含まれます。結果の新しい2つのパーティションの1つ目には、パーティション化キー列の値が、指定された値より小さいものをマッピングする元のパーティションのすべての行が含まれます。2つ目のパーティションには、パーティション化キー列値が、指定された値以上のものをマッピングするすべての行が含まれます。次のSQL文では、パーティションfee_katyが分割され、索引パーティションが再作成されます。

```
ALTER TABLE vet_cats SPLIT PARTITION
      fee_katy at (100) INTO ( PARTITION
      fee_katy1, PARTITION fee_katy2);
ALTER INDEX JAF1 REBUILD PARTITION fee_katy1;
ALTER INDEX JAF1 REBUILD PARTITION fee_katy2;
ALTER INDEX VET REBUILD PARTITION vet_parta;
ALTER INDEX VET REBUILD PARTITION vet_partb;
```

例4-37 オンラインでのレンジ・パーティション表のパーティションの分割

この例では、ORDERS表のsales_q4_2016パーティションが、月ごとに別個のパーティションに分割されます。ONLINEキーワードは、パーティション・メンテナンス操作の進行中に同時DML操作を可能にするために指定されます。

ORDERS表に索引がある場合、これらはオンライン分割の一部として自動的にメンテナンスされます。

```
CREATE TABLE orders
 (prod_id       NUMBER(6),
  cust_id       NUMBER,
  time_id       DATE,
  channel_id    CHAR(1),
  promo_id      NUMBER(6),
  quantity_sold NUMBER(3),
  amount_sold   NUMBER(10,2)
  )
  PARTITION BY RANGE (time_id)
  (PARTITION sales_q1_2016 VALUES LESS THAN (TO_DATE('01-APR-2016','dd-MON-yyyy')),
   PARTITION sales_q2_2016 VALUES LESS THAN (TO_DATE('01-JUL-2016','dd-MON-yyyy')),
   PARTITION sales_q3_2016 VALUES LESS THAN (TO_DATE('01-OCT-2016','dd-MON-yyyy')),
   PARTITION sales_q4_2016 VALUES LESS THAN (TO_DATE('01-JAN-2017','dd-MON-yyyy'))
   );

ALTER TABLE orders
  SPLIT PARTITION sales_q4_2016 INTO
  (PARTITION sales_oct_2016 VALUES LESS THAN (TO_DATE('01-NOV-2016','dd-MON-yyyy')),
   PARTITION sales_nov_2016 VALUES LESS THAN (TO_DATE('01-DEC-2016','dd-MON-yyyy')),
   PARTITION sales_dec_2016
   )
  ONLINE;
```

4.4.12.2 リスト・パーティション表のパーティションの分割

ALTER TABLE SPLIT PARTITION文を使用して、リスト・パーティションを分割できます。

SPLIT PARTITION句を使用すると、リテラル値のリストを指定してパーティションを定義できます。このパーティションに、対応するパーティション化キー値を含む行が挿入されます。元のパーティションのその他の行は、元のパーティションの残りの値が値リストに含まれる2番目のパーティションに挿入されます。分割の結果である2つのパーティションに、オプションで新しい属性を指定できます。

次の文では、パーティションregion_eastが2つのパーティションに分割されます。

```
ALTER TABLE sales_by_region
   SPLIT PARTITION region_east VALUES ('CT', 'MA', 'MD')
   INTO
    ( PARTITION region_east_1
         TABLESPACE tbs2,
      PARTITION region_east_2
        STORAGE (INITIAL 8M))
   PARALLEL 5;
```

元のregion_eastパーティションのリテラル値のリストは、次のように指定されています。

```
PARTITION region_east VALUES ('MA','NY','CT','NH','ME','MD','VA','PA','NJ')
```

2つの新しいパーティションは次のとおりです。

- リテラル値のリストが('CT','MA','MD')のregion_east_1
- 残りのリテラル値のリスト('NY','NH','ME','VA','PA','NJ')を継承しているregion_east_2

個々のパーティションには、パーティション・レベルで指定された新しい物理属性があります。操作は並列度5で実行されます。

その他のリスト・パーティションの分割と同じように、デフォルトのリスト・パーティションを分割できます。これは、デフォルトのパーティションがあるリスト・パーティション表に新しいパーティションを追加する唯一の方法でもあります。デフォルトのパーティションを分割する際には、指定した値で定義された新しいパーティションと、デフォルトのパーティションとして残る2つ目のパーティションを作成します。

Live SQL:

Oracle Live SQL の *Oracle Live SQL: リストパーティション表の DEFAULT パーティションの分割*で関連する例を参照して実行してください。

例4-38 リスト・パーティション表のデフォルト・パーティションの分割

この例では、sales_by_regionのデフォルトのパーティションを分割することで、新しいパーティションを作成します。

```
CREATE TABLE sales_by_region
     (dept_number          NUMBER NOT NULL,
      dept_name            VARCHAR2(20),
      quarterly_sales      NUMBER(10,2),
      state                VARCHAR2(2)
 )
 PARTITION BY LIST (state)
 (
  PARTITION yearly_north VALUES ('MN','WI','MI'),
  PARTITION yearly_south VALUES ('NM','TX','GA'),
  PARTITION yearly_east VALUES  ('MA','NY','NC'),
```

```
  PARTITION yearly_west VALUES  ('CA','OR','WA'),
  PARTITION unknown VALUES (DEFAULT)
 );

SELECT PARTITION_NAME, HIGH_VALUE FROM USER_TAB_PARTITIONS WHERE TABLE_NAME ='SALES_BY_REGION';
PARTITION_NAME       HIGH_VALUE
--------------       ---------------
UNKNOWN              DEFAULT
YEARLY_EAST          'MA', 'NY', 'NC'
YEARLY_NORTH         'MN', 'WI', 'MI'
YEARLY_SOUTH         'NM', 'TX', 'GA'
YEARLY_WEST          'CA', 'OR', 'WA
5 rows selected.

INSERT INTO SALES_BY_REGION VALUES (002, 'AUTO NORTH', 450000, 'MN');
INSERT INTO SALES_BY_REGION VALUES (002, 'AUTO NORTH', 495000, 'WI');
INSERT INTO SALES_BY_REGION VALUES (002, 'AUTO NORTH', 850000, 'MI');

INSERT INTO SALES_BY_REGION VALUES (004, 'AUTO SOUTH',  595000, 'NM');
INSERT INTO SALES_BY_REGION VALUES (004, 'AUTO SOUTH', 4825000, 'TX');
INSERT INTO SALES_BY_REGION VALUES (004, 'AUTO SOUTH',  945000, 'GA');

INSERT INTO SALES_BY_REGION VALUES (006, 'AUTO EAST', 2125000, 'MA');
INSERT INTO SALES_BY_REGION VALUES (006, 'AUTO EAST', 6101000, 'NY');
INSERT INTO SALES_BY_REGION VALUES (006, 'AUTO EAST',  741000, 'NC');

INSERT INTO SALES_BY_REGION VALUES (008, 'AUTO WEST', 7201000, 'CA');
INSERT INTO SALES_BY_REGION VALUES (008, 'AUTO WEST',  901000, 'OR');
INSERT INTO SALES_BY_REGION VALUES (008, 'AUTO WEST', 1125000, 'WA');

INSERT INTO SALES_BY_REGION VALUES (009, 'AUTO MIDWEST', 1950000, 'AZ');
INSERT INTO SALES_BY_REGION VALUES (009, 'AUTO MIDWEST', 5725000, 'UT');

SELECT DEPT_NUMBER, DEPT_NAME, QUARTERLY_SALES, STATE FROM SALES_BY_REGION PARTITION(yearly_north);
DEPT_NUMBER DEPT_NAME            QUARTERLY_SALES ST
----------- -------------------- --------------- --
2           AUTO NORTH                    450000 MN
2           AUTO NORTH                    495000 WI
2           AUTO NORTH                    850000 MI

SELECT DEPT_NUMBER, DEPT_NAME, QUARTERLY_SALES, STATE FROM SALES_BY_REGION PARTITION(yearly_south);
DEPT_NUMBER DEPT_NAME            QUARTERLY_SALES ST
----------- -------------------- --------------- --
4           AUTO SOUTH                    595000 NM
4           AUTO SOUTH                   4825000 TX
4           AUTO SOUTH                    945000 GA

...

SELECT DEPT_NUMBER, DEPT_NAME, QUARTERLY_SALES, STATE FROM SALES_BY_REGION PARTITION(unknown);
DEPT_NUMBER DEPT_NAME            QUARTERLY_SALES ST
----------- -------------------- --------------- --
9           AUTO MIDWEST                 1950000 AZ
9           AUTO MIDWEST                 5725000 UT

REM Note that the following ADD PARTITION statement fails. This action fails because
REM all undefined values are automatically included in the DEFAULT partition.
ALTER TABLE sales_by_region ADD PARTITION yearly_midwest VALUES ('AZ', 'UT');
ORA-14323: cannot add partition when DEFAULT partition exists
```

```
REM You must SPLIT the DEFAULT partition to add a new partition.
ALTER TABLE sales_by_region
   SPLIT PARTITION unknown VALUES ('AZ', 'UT')
   INTO
    ( PARTITION yearly_midwest,
      PARTITION unknown);

SELECT PARTITION_NAME, HIGH_VALUE FROM USER_TAB_PARTITIONS WHERE TABLE_NAME ='SALES_BY_REGION';
PARTITION_NAME       HIGH_VALUE
--------------       ---------------
UNKNOWN              DEFAULT
YEARLY_EAST          'MA', 'NY', 'NC'
YEARLY_MIDWEST       'AZ', 'UT'
YEARLY_NORTH         'MN', 'WI', 'MI'
YEARLY_SOUTH         'NM', 'TX', 'GA'
YEARLY_WEST          'CA', 'OR', 'WA'
6 Rows selected.

SELECT DEPT_NUMBER, DEPT_NAME, QUARTERLY_SALES, STATE FROM SALES_BY_REGION PARTITION(yearly_midwest);
DEPT_NUMBER DEPT_NAME            QUARTERLY_SALES ST
----------- -------------------- --------------- --
          9 AUTO MIDWEST                 1950000 AZ
          9 AUTO MIDWEST                 5725000 UT

SELECT DEPT_NUMBER, DEPT_NAME, QUARTERLY_SALES, STATE FROM SALES_BY_REGION PARTITION(unknown);
no rows selected

REM Split the DEFAULT partition again to add a new 'yearly_mideast' partition.
ALTER TABLE sales_by_region
   SPLIT PARTITION unknown VALUES ('OH', 'IL')
   INTO
    ( PARTITION yearly_mideast,
      PARTITION unknown);
Table altered.

SELECT PARTITION_NAME, HIGH_VALUE FROM USER_TAB_PARTITIONS WHERE TABLE_NAME ='SALES_BY_REGION';
PARTITION_NAME      HIGH_VALUE
------------------  ------------------
UNKNOWN             DEFAULT
YEARLY_EAST         'MA', 'NY', 'NC'
YEARLY_MIDEAST      'OH', 'IL'
YEARLY_MIDWEST      'AZ', 'UT'
YEARLY_NORTH        'MN', 'WI', 'MI'
YEARLY_SOUTH        'NM', 'TX', 'GA'
YEARLY_WEST         'CA', 'OR', 'WA'
7 rows selected.

INSERT INTO SALES_BY_REGION VALUES (007, 'AUTO MIDEAST',  925000, 'OH');
INSERT INTO SALES_BY_REGION VALUES (007, 'AUTO MIDEAST', 1325000, 'IL');

SELECT DEPT_NUMBER, DEPT_NAME, QUARTERLY_SALES, STATE FROM SALES_BY_REGION PARTITION(yearly_mideast);
DEPT_NUMBER DEPT_NAME            QUARTERLY_SALES ST
----------- -------------------- --------------- --
          7 AUTO MIDEAST                  925000 OH
          7 AUTO MIDEAST                 1325000 IL

SELECT DEPT_NUMBER, DEPT_NAME, QUARTERLY_SALES, STATE FROM SALES_BY_REGION PARTITION(unknown);
no rows selected
```

4.4.12.3 時間隔パーティション表のパーティションの分割

時間隔パーティション表でALTER TABLE SPLIT PARTITION文を使用して、レンジまたはマテリアライズ化された時間隔パーティションを分割できます。

時間隔パーティション表のレンジ・パーティションの分割については、「レンジ・パーティション表のパーティションの分割」で説明しています。

マテリアライズ化された時間隔パーティションを分割するには、パーティションの分割点である、時間隔パーティション内のパーティション化キー列の値を指定します。結果の新しい2つのパーティションの1つ目には、パーティション化キー列の値が、指定された値より小さいものをマッピングする元のパーティションのすべての行が含まれます。2つ目のパーティションには、パーティション化キー列値が、指定された値以上のものをマッピングするすべての行が含まれます。パーティションの分割操作により、分割したパーティションの高い方の上限まで遷移点が移動され、新しく分割されたパーティションより小さいすべてのマテリアライズ化された時間隔パーティションは、上限が時間隔の上限によって定義された状態で、暗黙的にレンジ・パーティションに変換されます。

分割の結果である2つのレンジ・パーティションに、オプションで新しい属性を指定できます。表にローカル索引が定義されている場合、この文では、各ローカル索引の一致するパーティションも分割されます。作成されていない時間隔パーティションは分割できません。

次の例では、月単位の時間隔パーティション表transactionsの2007年5月のパーティションを分割する方法を示します。

```
ALTER TABLE transactions
    SPLIT PARTITION FOR(TO_DATE('01-MAY-2007','dd-MON-yyyy'))
    AT (TO_DATE('15-MAY-2007','dd-MON-yyyy'));
```

4.4.12.4 * - ハッシュ・パーティションの分割

ALTER TABLE SPLIT PARTITION文を使用して、ハッシュ・パーティションを分割できます。

これは、* - ハッシュ・パーティションのマージの逆の操作です。* - ハッシュ・パーティションを分割すると、新しいサブパーティションが、SUBPARTITIONSまたはSUBPARTITION句で指定されたサブパーティション数に再度ハッシュされます。または、そのような句が含まれていない場合、新しいパーティションは、分割されるパーティションのサブパーティション(および表領域)の数を継承します。

2つの* - ハッシュ・パーティションをマージする場合と、* - ハッシュ・パーティションを分割する場合では、プロパティの継承方法が異なります。パーティションを分割する場合は親が1つのみであるため、新しいパーティションは元のパーティションからプロパティを継承できます。ただし、パーティションがマージされるとき、プロパティを表レベルのデフォルトから継承する必要があります。親が2つあり、新しいパーティションが片方を無視していずれか1つから継承することはできないためです。

次の例では、レンジ - ハッシュ・パーティションを分割します。

```
ALTER TABLE all_seasons SPLIT PARTITION quarter_1
     AT (TO_DATE('16-dec-1997','dd-mon-yyyy'))
     INTO (PARTITION q1_1997_1 SUBPARTITIONS 4 STORE IN (ts1,ts3),
           PARTITION q1_1997_2);
```

時間隔 - ハッシュ・パーティション表の分割のルールは、時間隔パーティション表の分割のルールと同じです。「時間隔パーティション表のパーティションの分割」に説明されているように、遷移点は分割されたパーティションの上限に変更されます。

4.4.12.5 * - リスト・パーティション表のパーティションの分割

パーティションは、リスト・パーティション表のパーティション・レベルとサブパーティション・レベルの両方で分割できます。

- * - リスト・パーティションの分割
- * - リスト・サブパーティションの分割

4.4.12.5.1 * - リスト・パーティションの分割

ALTER TABLE SPLIT PARTITION文を使用して、リスト・パーティションを分割できます。

* - リスト・パーティション表のパーティションの分割は、「リスト・パーティション表のパーティションの分割」での説明と似ています。新しいパーティションのいずれにも、サブパーティションのリテラル値のリストを指定できません。新しいパーティションは、分割元のパーティションからサブパーティションの説明を継承します。

次の例では、quarterly_regional_sales表のq1_1999パーティションを分割します。

```
ALTER TABLE quarterly_regional_sales SPLIT PARTITION q1_1999
   AT (TO_DATE('15-Feb-1999','dd-mon-yyyy'))
   INTO ( PARTITION q1_1999_jan_feb
             TABLESPACE ts1,
          PARTITION q1_1999_feb_mar
             STORAGE (INITIAL 8M) TABLESPACE ts2)
   PARALLEL 5;
```

この操作により、パーティションq1_1999が、q1_1999_jan_febおよびq1_1999_feb_marの2つの結果のパーティションに分割されます。どちらのパーティションも、元のパーティションからサブパーティションの説明を継承します。個々のパーティションには、パーティション・レベルで指定された新しい物理属性(表領域を含む)があります。これらの新しい属性は、新しいパーティションのデフォルト属性になります。この操作は並列度5で実行されます。

ALTER TABLE SPLIT PARTITION文には、コンポジット・パーティション表のパーティションを分割した結果のサブパーティションに、明示的に名前を付ける方法がありません。ただし、*partition name_subpartition name*という書式の名前が付けられた親パーティションのサブパーティションには、データベースにより、新規に作成されたサブパーティションに新しいパーティション名を使用して対応する名前が生成されます。その他すべてのサブパーティションには、SYS_SUBP*n*という書式のシステム生成の名前が割り当てられます。システム生成の名前は、名前が指定されていないパーティションを分割した結果のサブパーティションにも割り当てられます。名前が付けられていないパーティションには、SYS_P*n*という書式のシステム生成のパーティション名が割り当てられます。

次の問合せでは、表quarterly_regional_salesに対する前のパーティション分割操作で生成されたサブパーティション名が表示されます。「コンポジット・レンジ - リスト・パーティション表の作成について」で作成されてから、この表に対してこの章でこれまでに実行された他の操作の結果も反映しています。

```
SELECT PARTITION_NAME, SUBPARTITION_NAME, TABLESPACE_NAME
  FROM DBA_TAB_SUBPARTITIONS
  WHERE TABLE_NAME='QUARTERLY_REGIONAL_SALES'
  ORDER BY PARTITION_NAME;

PARTITION_NAME       SUBPARTITION_NAME               TABLESPACE_NAME
-------------------- ------------------------------- ---------------
Q1_1999_FEB_MAR      Q1_1999_FEB_MAR_WEST            TS2
Q1_1999_FEB_MAR      Q1_1999_FEB_MAR_NORTHEAST       TS2
Q1_1999_FEB_MAR      Q1_1999_FEB_MAR_SOUTHEAST       TS2
Q1_1999_FEB_MAR      Q1_1999_FEB_MAR_NORTHCENTRAL    TS2
Q1_1999_FEB_MAR      Q1_1999_FEB_MAR_SOUTHCENTRAL    TS2
Q1_1999_FEB_MAR      Q1_1999_FEB_MAR_SOUTH           TS2
Q1_1999_JAN_FEB      Q1_1999_JAN_FEB_WEST            TS1
Q1_1999_JAN_FEB      Q1_1999_JAN_FEB_NORTHEAST       TS1
Q1_1999_JAN_FEB      Q1_1999_JAN_FEB_SOUTHEAST       TS1
Q1_1999_JAN_FEB      Q1_1999_JAN_FEB_NORTHCENTRAL    TS1
Q1_1999_JAN_FEB      Q1_1999_JAN_FEB_SOUTHCENTRAL    TS1
Q1_1999_JAN_FEB      Q1_1999_JAN_FEB_SOUTH           TS1
Q1_2000              Q1_2000_NORTHWEST               TS3
Q1_2000              Q1_2000_SOUTHWEST               TS3
Q1_2000              Q1_2000_NORTHEAST               TS3
Q1_2000              Q1_2000_SOUTHEAST               TS3
```

```
Q1_2000              Q1_2000_NORTHCENTRAL          TS3
Q1_2000              Q1_2000_SOUTHCENTRAL          TS3
Q2_1999              Q2_1999_NORTHWEST             TS4
Q2_1999              Q2_1999_SOUTHWEST             TS4
Q2_1999              Q2_1999_NORTHEAST             TS4
Q2_1999              Q2_1999_SOUTHEAST             TS4
Q2_1999              Q2_1999_NORTHCENTRAL          TS4
Q2_1999              Q2_1999_SOUTHCENTRAL          TS4
Q3_1999              Q3_1999_NORTHWEST             TS4
Q3_1999              Q3_1999_SOUTHWEST             TS4
Q3_1999              Q3_1999_NORTHEAST             TS4
Q3_1999              Q3_1999_SOUTHEAST             TS4
Q3_1999              Q3_1999_NORTHCENTRAL          TS4
Q3_1999              Q3_1999_SOUTHCENTRAL          TS4
Q4_1999              Q4_1999_NORTHWEST             TS4
Q4_1999              Q4_1999_SOUTHWEST             TS4
Q4_1999              Q4_1999_NORTHEAST             TS4
Q4_1999              Q4_1999_SOUTHEAST             TS4
Q4_1999              Q4_1999_NORTHCENTRAL          TS4
Q4_1999              Q4_1999_SOUTHCENTRAL          TS4

36 rows selected.
```

4.4.12.5.2 * - リスト・サブパーティションの分割

ALTER TABLE SPLIT SUBPARTITION文を使用して、リスト・サブパーティションを分割できます。

* - リスト・パーティション表のリスト・サブパーティションの分割は、「リスト・パーティション表のパーティションの分割」で説明されている内容と似ていますが、構文はPARTITIONではなくSUBPARTITIONです。たとえば、次の文では、quarterly_regional_sales表のサブパーティションを分割します。

```
ALTER TABLE quarterly_regional_sales SPLIT SUBPARTITION q2_1999_southwest
   VALUES ('UT') INTO
      ( SUBPARTITION q2_1999_utah
           TABLESPACE ts2,
        SUBPARTITION q2_1999_southwest
           TABLESPACE ts3
      )
   PARALLEL;
```

この操作により、サブパーティションq2_1999_southwestが、次の2つのサブパーティションに分割されます。

- リテラル値のリストが('UT')のq2_1999_utah
- 残りのリテラル値のリスト('AZ','NM')を継承しているq2_1999_southwest

個々のサブパーティションには、分割されたサブパーティションから継承された新しい物理属性があります。

時間隔 - リスト・パーティション表のサブパーティションは、レンジ・パーティションまたはマテリアライズ化された時間隔パーティションにのみ分割できます。後続の時間隔パーティションのサブパーティション値を変更するには、サブパーティション・テンプレートを変更する必要があります。

4.4.12.6 * - レンジ・パーティションの分割

ALTER TABLE SPLIT PARTITION文を使用して、レンジ・パーティションを分割できます。

* - レンジ・パーティション表のパーティションの分割は、「レンジ・パーティション表のパーティションの分割」での説明と似ています。新しいパーティションのいずれにも、サブパーティションのレンジ値を指定できません。新しいパーティションは、分割元のパーティ

ションからサブパーティションの説明を継承します。

次の例では、時間隔 - レンジ・パーティション表ordersの2007年5月の時間隔パーティションを分割します。

```
ALTER TABLE orders
    SPLIT PARTITION FOR(TO_DATE('01-MAY-2007','dd-MON-yyyy'))
    AT (TO_DATE('15-MAY-2007','dd-MON-yyyy'))
    INTO (PARTITION p_fh_may07,PARTITION p_sh_may2007);
```

この操作により、時間隔パーティションFOR('01-MAY-2007')が、p_fh_may07およびp_sh_may_2007の2つの結果のパーティションに分割されます。どちらのパーティションも、元のパーティションからサブパーティションの説明を継承します。「時間隔パーティションのマージ」で説明されているように、2007年6月のパーティションより前の時間隔パーティションはレンジ・パーティションに変換されます。

ALTER TABLE SPLIT PARTITION文には、コンポジット・パーティション表のパーティションを分割した結果のサブパーティションに、明示的に名前を付ける方法がありません。ただし、*partition name*_*subpartition name*という書式の名前が付けられた親パーティションのサブパーティションには、データベースにより、新規に作成されたサブパーティションに新しいパーティション名を使用して対応する名前が生成されます。その他すべてのサブパーティションには、SYS_SUBP*n*という書式のシステム生成の名前が割り当てられます。システム生成の名前は、名前が指定されていないパーティションを分割した結果のサブパーティションにも割り当てられます。名前が付けられていないパーティションには、SYS_P*n*という書式のシステム生成のパーティション名が割り当てられます。

次の問合せでは、表ordersに対する前のパーティション分割操作で生成されたサブパーティション名と上限値が表示されます。表が作成されてから、この章でこれまでに実行された他の操作の結果も反映しています。

```
BREAK ON partition_name

SELECT partition_name, subpartition_name, high_value
FROM user_tab_subpartitions
WHERE table_name = 'ORCERS'
ORDER BY partition_name, subpartition_position;

PARTITION_NAME            SUBPARTITION_NAME              HIGH_VALUE
------------------------- ------------------------------ ---------------
P_BEFORE_2007             P_BEFORE_2007_P_SMALL          1000
                          P_BEFORE_2007_P_MEDIUM         10000
                          P_BEFORE_2007_P_LARGE          100000
                          P_BEFORE_2007_P_EXTRAORDINARY  MAXVALUE
P_FH_MAY07                SYS_SUBP2985                   1000
                          SYS_SUBP2986                   10000
                          SYS_SUBP2987                   100000
                          SYS_SUBP2988                   MAXVALUE
P_PRE_MAY_2007            P_PRE_MAY_2007_P_SMALL         1000
                          P_PRE_MAY_2007_P_MEDIUM        10000
                          P_PRE_MAY_2007_P_LARGE         100000
                          P_PRE_MAY_2007_P_EXTRAORDINARY MAXVALUE
P_SH_MAY2007              SYS_SUBP2989                   1000
                          SYS_SUBP2990                   10000
                          SYS_SUBP2991                   100000
                          SYS_SUBP2992                   MAXVALUE
```

4.4.12.6.1 * - レンジ・サブパーティションの分割

ALTER TABLE SPLIT SUBPARTITION文を使用して、レンジ・サブパーティションを分割できます。

* - レンジ・パーティション表のレンジ・サブパーティションの分割は、「レンジ・パーティション表のパーティションの分割」で説明されている内容と似ていますが、構文はPARTITIONではなくSUBPARTITIONです。たとえば、次の文では、orders表のサブパーティションを分割します。

```
ALTER TABLE orders
SPLIT SUBPARTITION p_pre_may_2007_p_large AT (50000)
INTO (SUBPARTITION p_pre_may_2007_med_large TABLESPACE TS4
     , SUBPARTITION p_pre_may_2007_large_large TABLESPACE TS5
     );
```

この操作により、サブパーティションp_pre_may_2007_p_largeが、次の2つのサブパーティションに分割されます。

- 値が10000から50000のp_pre_may_2007_med_large
- 値が50000から100000のp_pre_may_2007_large_large

個々のサブパーティションには、分割されたサブパーティションから継承された新しい物理属性があります。

時間隔 - レンジ・パーティション表のサブパーティションは、レンジ・パーティションまたはマテリアライズ化された時間隔パーティションにのみ分割できます。後続の時間隔パーティションのサブパーティションの上限を変更するには、サブパーティション・テンプレートを変更する必要があります。

4.4.12.7 索引パーティションの分割

ローカル索引のパーティションは明示的に分割できません。ローカル索引パーティションが分割されるのは、基礎となる表のパーティションを分割した場合のみです。

ただし、グローバル索引パーティションは、次の例のようにして分割できます。

```
ALTER INDEX quon1 SPLIT
    PARTITION canada AT ( 100 ) INTO
    PARTITION canada1 ..., PARTITION canada2 ...);
ALTER INDEX quon1 REBUILD PARTITION canada1;
ALTER INDEX quon1 REBUILD PARTITION canada2;
```

分割する索引に索引データが含まれていても問題はなく、元のパーティションがUNUSABLEとマークされていなければ、結果のパーティションを再作成する必要はありません。

4.4.12.8 複数のパーティションへの分割

ALTER TABLE文のSPLIT PARTITIONおよびSPLIT SUBPARTITION句を使用して、1つのパーティションまたはサブパーティションのコンテンツを複数のパーティションまたはサブパーティションに再分散できます。

複数のパーティションに分割すると、現在のパーティションに関連付けられているセグメントが破棄されます。新しい各パーティションで、新しいセグメントが取得され、現在のソース・パーティションから、指定されていない物理属性がすべて継承されます。必要な条件が満たされると、高速分割の最適化が複数パーティションの分割操作に適用されます。

拡張分割構文を使用して、ATまたはVALUES句を指定せずにパーティション表の作成のSQL文と似ている新しいパーティションの説明のリストを指定できます。また、最新パーティションの説明のレンジまたはリスト値句は、ソース・パーティションの上限、および分割結果の最初(N-1)の新しいパーティションに指定されたバウンド値に基づいて導出されます。

次のSQL文は、パーティションを複数のパーティションに分割する例です。

```
ALTER TABLE SPLIT PARTITION p0 INTO
  (PARTITION p01 VALUES LESS THAN (25),
   PARTITION p02 VALUES LESS THAN (50),
   PARTITION p03 VALUES LESS THAN (75),
   PARTITION p04);

ALTER TABLE SPLIT PARTITION p0 INTO
  (PARTITION p01 VALUES LESS THAN (25),
   PARTITION p02);
```

2番目のSQLの例では、パーティションp02が元のパーティションp0の上限を持ちます。

レンジ・パーティションをNパーティションに分割するには、パーティション化キー列の(N-1)値は、パーティションを分割するパーティションの範囲内で指定する必要があります。指定される新しい上限値(最上限値は含まない)は、昇順にする必要があります。N番目の新しいパーティションの上限には、分割されるパーティションの上限の値が割り当てられます。分割の結果のN番目の新しいパーティションの名前および物理属性は、オプションで指定できます。

リスト・パーティションをNパーティションに分割するには、リテラル値の(N-1)リストを指定する必要があり、それぞれで対応するパーティション・キー値を持つ行を挿入する最初(N-1)のパーティションを定義します。元のパーティションのその他の行は、元のパーティションの残りのリテラル値が値リストに含まれるN番目の新しいパーティションに挿入されます。2つの値リストを同じパーティション値に含むことはできません。指定される(N-1)値リストは、N番目の新しいパーティションが空であるため、現在のパーティションのすべてのパーティション値を含むことはできません。また、(N-1)値リストは、現在のパーティションに存在しないパーティション値を含むことはできません。

DEFAULTリスト・パーティションまたはMAXVALUEレンジ・パーティションを複数のパーティションに分割する場合、分割結果のN番目の新しいパーティションにDEFAULT値またはMAXVALUEがありますが、指定されたリテラル値リストまたは上限値を使用して最初(N-1)の新しいパーティションが作成されます。コンポジット・パーティション表のパーティションを複数のパーティションに分割する場合、分割結果の新しいパーティションのサブパーティションに関する数値、名前、バウンドおよび物理プロパティの継承について既存の動作を想定しています。SPLIT_TABLE_SUBPARTITION句が同様に拡張され、レンジまたはリスト・サブパーティションをN番目の新しいサブパーティションに分割できます。

ローカルおよびグローバル索引に関するSQL文の動作は、変更されないままです。対応するローカル索引パーティションは、複数のパーティションに分割されます。パーティション表にLOB列が含まれる場合、SPLIT PARTITION句の既存のセマンティックが拡張された構文で適用されます。つまり、現在のパーティションのLOBデータおよび索引セグメントが削除され、新しいパーティションごとに各LOB列の新しいセグメントが作成されます。必要な条件が満たされると、高速分割の最適化が複数パーティションの分割操作に適用されます。

たとえば、次のSQL文は、レンジ・パーティション化されている表salesのsales_Q4_2007パーティションを次の年の四半期に対応する5つのパーティションに分割します。この例では、パーティションsales_Q4_2008は、暗黙的に分割されたパーティションの上限になります。

```
ALTER TABLE sales SPLIT PARTITION sales_Q4_2007 INTO
( PARTITION sales_Q4_2007 VALUES LESS THAN (TO_DATE('01-JAN-2008','dd-MON-yyyy')),
  PARTITION sales_Q1_2008 VALUES LESS THAN (TO_DATE('01-APR-2008','dd-MON-yyyy')),
  PARTITION sales_Q2_2008 VALUES LESS THAN (TO_DATE('01-JUL-2008','dd-MON-yyyy')),
  PARTITION sales_Q3_2008 VALUES LESS THAN (TO_DATE('01-OCT-2008','dd-MON-yyyy')),
  PARTITION sales_Q4_2008);
```

リスト・パーティション化されているサンプル表customersの場合、次の文はパーティション**Europe**を3つのパーティションに分割します。

```
ALTER TABLE list_customers SPLIT PARTITION Europe INTO
  (PARTITION western-europe VALUES ('GERMANY', 'FRANCE'),
   PARTITION southern-europe VALUES ('ITALY'),
   PARTITION rest-europe);
```

4.4.12.9 高速なSPLIT PARTITIONおよびSPLIT SUBPARTITION操作

Oracle Databaseでは、2つ以上の新しいパーティションを作成し、分割元のパーティションの行を複数の新しいパーティションに再分配することで、SPLIT PARTITION操作が実施されます。

この操作には時間がかかります。分割対象のパーティションのすべての行をスキャンしてから、新しいパーティションに1行ずつ挿入する必要があるためです。さらに、UPDATE INDEXES句を使用しない場合、ローカルおよびグローバル索引も再作成する必要が

あります。

分割操作後に、分割元のパーティションのすべての行が、新しいあるパーティションに含まれ、他のパーティションには行が含まれていない場合があります。これは、表の最初または最後のパーティションを分割した場合によく起こります。このような状況はデータベースにより検出され、分割操作が最適化されます。この最適化により、パーティションの追加操作のように動作する分割操作が高速になります。

具体的には、次の条件をすべて満たしている場合に、SPLIT PARTITION操作がデータベースにより最適化および高速化されます。

- 結果となるパーティションの1つに、すべての行が含まれます。
- 空ではない結果のパーティションの記憶域特性が、分割されたパーティションの記憶域特性と同一である必要があります。具体的には、次のようになります。
    - 分割元のパーティションがコンポジットの場合、結果となる新しいパーティションの各サブパーティションの記憶域特性は、分割元のパーティションのサブパーティションの記憶域特性と同一である必要があります。
    - 分割元のパーティションにLOB列が含まれる場合、空ではない新しい結果のパーティションの各LOB(サブ)パーティションの記憶域特性は、分割元のパーティションのLOB(サブ)パーティションの記憶域特性と同一である必要があります。
    - オーバーフローが含まれる索引構成表のパーティションを分割する場合、空ではない新しい結果のパーティションの各オーバーフロー(サブ)パーティションの記憶域特性は、分割元のパーティションのオーバーフロー(サブ)パーティションの記憶域特性と同一である必要があります。
    - マッピング表が含まれる索引構成表のパーティションを分割する場合、空ではない新しい結果のパーティションの各マッピング表(サブ)パーティションの記憶域特性は、分割元のパーティションのマッピング表(サブ)パーティションの記憶域特性と同一である必要があります。

分割後にこれらの条件に一致していれば、UPDATE INDEXES句を指定しなかった場合でも、グローバル索引はすべて使用可能なままです。結果となるパーティションに関連付けられているローカル索引(サブ)パーティションは、分割前に使用可能であった場合は、使用可能なままとなります。空ではない結果のパーティションに対応するローカル索引(サブ)パーティションは、分割されたパーティションのローカル索引(サブ)パーティションと同一になります。SPLIT SUBPARTITION操作にも、同じ最適化が行われます。

4.4.13 パーティションおよびサブパーティションの切捨てについて

パーティションの切捨ては、パーティションのデータはなくなりますが物理的に削除されないことを除き、パーティションの削除に似ています。

表パーティションからすべての行を削除するには、ALTER TABLE TRUNCATE PARTITION文を使用します。索引パーティションは切り捨てられません。ただし、表にローカル索引が定義されている場合には、ALTER TABLE TRUNCATE PARTITION文により各ローカル索引の一致するパーティションが切り捨てられます。UPDATE INDEXESを指定しない場合、グローバル索引はUNUSABLEとマークされ、再作成する必要があります。索引構成表にはUPDATE INDEXESは使用できません。かわりにUPDATE GLOBAL INDEXESを使用してください。

この項では、次の項目について説明します。

- 表パーティションの切捨てについて
- 複数のパーティションの切捨て
- サブパーティションの切捨て
- カスケード・オプションを使用したパーティションの切捨て

関連項目:

- パーティションを切り捨てる非同期グローバル索引メンテナンスの詳細は、「パーティションを削除および切り捨てる非同期グローバル索引メンテナンス」を参照してください
- パーティションの削除の詳細は、「パーティションおよびサブパーティションの削除について」を参照してください

4.4.13.1 表パーティションの切捨てについて

領域を解放するかどうかに関係なく、表パーティションからすべての行を削除するには、ALTER TABLE TRUNCATE PARTITION文を使用します。

時間隔パーティション表のパーティションを切り捨てても、遷移点は移動されません。参照パーティション表のパーティションおよびサブパーティションは切り捨てられません。

- データおよびグローバル索引を含む表パーティションの切捨て
- データおよび参照整合性制約を含むパーティションの切捨て

4.4.13.1.1 データおよびグローバル索引を含む表パーティションの切捨て

データおよびグローバル索引を含む表パーティションを切り捨てる場合、次の方法のいずれかを使用できます。

パーティションにデータおよびグローバル索引が含まれている場合は、次のいずれかの方法(方法1、2または3)を使用して表パーティションを切り捨てます。

方法1

ALTER TABLE TRUNCATE PARTITION文の実行中は、グローバル索引はそのままにします。この例では、表salesにはグローバル索引sales_area_ixがあり、この索引は再作成されます。

```
ALTER TABLE sales TRUNCATE PARTITION dec98;
ALTER INDEX sales_area_ix REBUILD;
```

これは、切り捨てられるパーティションに表の合計データの大部分が含まれる大規模な表に最適な方法です。

方法2

ALTER TABLE TRUNCATE PARTITION文を発行する前に、DELETE文を実行してパーティションからすべての行を削除します。DELETE文によりグローバル索引が更新され、トリガーの起動や、REDOおよびUNDOログの生成も行われます。

たとえば、最初のパーティションを切り捨てるには、次の文を実行します。

```
DELETE FROM sales PARTITION (dec98);
ALTER TABLE sales TRUNCATE PARTITION dec98;
```

これは、小規模な表、または切り捨てられるパーティションに含まれる表の合計データの割合が少ない場合に、大規模な表に最適な方法です。

方法3

ALTER TABLE文にUPDATE INDEXESを指定します。これにより、パーティションが切り捨てられる時にグローバル索引も切り捨てられます。

```
ALTER TABLE sales TRUNCATE PARTITION dec98
     UPDATE INDEXES;
```

非同期グローバル索引メンテナンスでは、この操作はメタデータのみの操作です。

4.4.13.1.2 データおよび参照整合性制約を含むパーティションの切捨て

パーティションにデータが含まれ、参照整合性制約がある場合には、パーティションを切り捨てられません。ただし、削除するパーティションのデータを参照しているデータがない場合は、次のいずれかの方法を使用できます。

表パーティションを切り捨てるには、次のいずれかの方法(方法1または2)を選択してください。

方法1

整合性制約を無効化し、ALTER TABLE TRUNCATE PARTITION文を実行し、整合性制約を再度有効化します。これは、切り捨てられるパーティションに表の合計データの大部分が含まれる大規模な表に最適な方法です。別の表に参照元のデータがある場合には、そのデータを削除して、整合性制約を再度有効化できるようにする必要があります。

方法2

ALTER TABLE TRUNCATE PARTITION文を発行する前に、DELETE文を発行してパーティションからすべての行を削除します。DELETE文によって、参照整合性制約が施行されます。また、トリガーが起動されて、REDOログとUNDOログが生成されます。参照表のデータが削除されるのは、外部キー制約がON DELETE CASCADEオプション付きで作成されていた場合です。

```
DELETE FROM sales partition (dec94);
ALTER TABLE sales TRUNCATE PARTITION dec94;
```

これは、小規模な表、または切り捨てられるパーティションに含まれる表の合計データの割合が少ない場合に、大規模な表に最適な方法です。

4.4.13.2 複数のパーティションの切捨て

ALTER TABLE文のTRUNCATE PARTITION句を使用して、レンジまたはリスト・パーティション表から複数のパーティションを切り捨てることができます。

ローカル索引の対応するパーティションは、操作で切り捨てられます。UPDATE INDEXESが指定されていないかぎり、グローバル索引を再構築する必要があります。

次の例では、SQL文のALTER TABLEで表の複数のサブパーティションを切り捨てます。データは切り捨てられますが、パーティションは削除されない点に注意してください。

Live SQL:

関連する例を Oracle Live SQL の *Oracle Live SQL: レンジ・パーティション表の切捨て*で参照して実行してください。

例4-39 複数のパーティションの切捨て

```
CREATE TABLE sales_partition_truncate
  ( product_id       NUMBER(6) NOT NULL,
    customer_id      NUMBER    NOT NULL,
    channel_id       CHAR(1),
    promo_id         NUMBER(6),
    sales_date       DATE,
    quantity_sold    INTEGER,
    amount_sold      NUMBER(10,2)
  )
 PARTITION BY RANGE (sales_date)
 SUBPARTITION BY LIST (channel_id)
```

```
(  PARTITION q3_2018 VALUES LESS THAN (TO_DATE('1-OCT-2018','DD-MON-YYYY'))
   ( SUBPARTITION q3_2018_p_catalog VALUES ('C'),
     SUBPARTITION q3_2018_p_internet VALUES ('I'),
     SUBPARTITION q3_2018_p_partners VALUES ('P'),
     SUBPARTITION q3_2018_p_direct_sales VALUES ('S'),
     SUBPARTITION q3_2018_p_tele_sales VALUES ('T')
   ),
  PARTITION q4_2018 VALUES LESS THAN (TO_DATE('1-JAN-2019','DD-MON-YYYY'))
   ( SUBPARTITION q4_2018_p_catalog VALUES ('C'),
     SUBPARTITION q4_2018_p_internet VALUES ('I'),
     SUBPARTITION q4_2018_p_partners VALUES ('P'),
     SUBPARTITION q4_2018_p_direct_sales VALUES ('S'),
     SUBPARTITION q4_2018_p_tele_sales VALUES ('T')
   ),
  PARTITION q1_2019 VALUES LESS THAN (TO_DATE('1-APR-2019','DD-MON-YYYY'))
   ( SUBPARTITION q1_2019_p_catalog VALUES ('C')
   , SUBPARTITION q1_2019_p_internet VALUES ('I')
   , SUBPARTITION q1_2019_p_partners VALUES ('P')
   , SUBPARTITION q1_2019_p_direct_sales VALUES ('S')
   , SUBPARTITION q1_2019_p_tele_sales VALUES ('T')
   ),
  PARTITION q2_2019 VALUES LESS THAN (TO_DATE('1-JUL-2019','DD-MON-YYYY'))
   ( SUBPARTITION q2_2019_p_catalog VALUES ('C'),
     SUBPARTITION q2_2019_p_internet VALUES ('I'),
     SUBPARTITION q2_2019_p_partners VALUES ('P'),
     SUBPARTITION q2_2019_p_direct_sales VALUES ('S'),
     SUBPARTITION q2_2019_p_tele_sales VALUES ('T')
   ),
  PARTITION q3_2019 VALUES LESS THAN (TO_DATE('1-OCT-2019','DD-MON-YYYY'))
   ( SUBPARTITION q3_2019_p_catalog VALUES ('C'),
     SUBPARTITION q3_2019_p_internet VALUES ('I'),
     SUBPARTITION q3_2019_p_partners VALUES ('P'),
     SUBPARTITION q3_2019_p_direct_sales VALUES ('S'),
     SUBPARTITION q3_2019_p_tele_sales VALUES ('T')
   ),
  PARTITION q4_2019 VALUES LESS THAN (TO_DATE('1-JAN-2020','DD-MON-YYYY'))
   ( SUBPARTITION q4_2019_p_catalog VALUES ('C'),
     SUBPARTITION q4_2019_p_internet VALUES ('I'),
     SUBPARTITION q4_2019_p_partners VALUES ('P'),
     SUBPARTITION q4_2019_p_direct_sales VALUES ('S'),
     SUBPARTITION q4_2019_p_tele_sales VALUES ('T')
   )
  );

SELECT TABLE_NAME, PARTITION_NAME, SUBPARTITION_NAME FROM USER_TAB_SUBPARTITIONS
       WHERE TABLE_NAME ='SALES_PARTITION_TRUNCATE';
TABLE_NAME                  PARTITION_NAME    SUBPARTITION_NAME
--------------------------  ---------------   -------------------
SALES_PARTITION_TRUNCATE    Q1_2019           Q1_2019_P_CATALOG
SALES_PARTITION_TRUNCATE    Q1_2019           Q1_2019_P_DIRECT_SALES
...
30 rows selected.

INSERT INTO sales_partition_truncate VALUES (1001,100,'C',150,'10-SEP-2018',500,2000);
INSERT INTO sales_partition_truncate VALUES (1021,200,'C',160,'16-NOV-2018',100,1500);
INSERT INTO sales_partition_truncate VALUES (1001,100,'C',150,'10-FEB-2019',500,2000);
INSERT INTO sales_partition_truncate VALUES (1021,200,'S',160,'16-FEB-2019',100,1500);
INSERT INTO sales_partition_truncate VALUES (1002,110,'I',180,'15-JUN-2019',100,1000);
INSERT INTO sales_partition_truncate VALUES (5010,150,'P',200,'20-AUG-2019',1000,10000);
INSERT INTO sales_partition_truncate VALUES (1001,100,'T',150,'12-OCT-2019',500,2000);
```

```
SELECT * FROM sales_partition_truncate;
PRODUCT_ID CUSTOMER_ID C   PROMO_ID SALES_DAT QUANTITY_SOLD AMOUNT_SOLD
---------- ----------- - ---------- --------- ------------- -----------
      1001         100 C        150 10-SEP-18           500        2000
      1021         200 C        160 16-NOV-18           100        1500
      1001         100 C        150 10-FEB-19           500        2000
      1021         200 S        160 16-FEB-19           100        1500
      1002         110 I        180 15-JUN-19           100        1000
      5010         150 P        200 20-AUG-19          1000       10000
      1001         100 T        150 12-OCT-19           500        2000
7 rows selected.

ALTER TABLE sales_partition_truncate
   TRUNCATE PARTITIONS q3_2018, q4_2018;

SELECT * FROM sales_partition_truncate;
PRODUCT_ID CUSTOMER_ID C   PROMO_ID SALES_DAT QUANTITY_SOLD AMOUNT_SOLD
---------- ----------- - ---------- --------- ------------- -----------
      1001         100 C        150 10-FEB-19           500        2000
      1021         200 S        160 16-FEB-19           100        1500
      1002         110 I        180 15-JUN-19           100        1000
      5010         150 P        200 20-AUG-19          1000       10000
      1001         100 T        150 12-OCT-19           500        2000
5 rows selected.

SELECT TABLE_NAME, PARTITION_NAME, SUBPARTITION_NAME FROM USER_TAB_SUBPARTITIONS
       WHERE TABLE_NAME ='SALES_PARTITION_TRUNCATE';
TABLE_NAME                  PARTITION_NAME    SUBPARTITION_NAME
--------------------------  ---------------   -------------------
SALES_PARTITION_TRUNCATE    Q1_2019           Q1_2019_P_CATALOG
SALES_PARTITION_TRUNCATE    Q1_2019           Q1_2019_P_DIRECT_SALES
...
SALES_PARTITION_TRUNCATE    Q3_2018           Q3_2018_P_CATALOG
SALES_PARTITION_TRUNCATE    Q3_2018           Q3_2018_P_DIRECT_SALES
SALES_PARTITION_TRUNCATE    Q3_2018           Q3_2018_P_INTERNET
SALES_PARTITION_TRUNCATE    Q3_2018           Q3_2018_P_PARTNERS
SALES_PARTITION_TRUNCATE    Q3_2018           Q3_2018_P_TELE_SALES
...
SALES_PARTITION_TRUNCATE    Q4_2018           Q4_2018_P_CATALOG
SALES_PARTITION_TRUNCATE    Q4_2018           Q4_2018_P_DIRECT_SALES
SALES_PARTITION_TRUNCATE    Q4_2018           Q4_2018_P_INTERNET
SALES_PARTITION_TRUNCATE    Q4_2018           Q4_2018_P_PARTNERS
SALES_PARTITION_TRUNCATE    Q4_2018           Q4_2018_P_TELE_SALES
...
30 rows selected.
```

4.4.13.3 サブパーティションの切捨て

コンポジット・パーティション表のサブパーティションからすべての行を削除するには、ALTER TABLE TRUNCATE SUBPARTITION文を使用します。

サブパーティションを切り捨てる場合、対応するローカル索引サブパーティションも切り捨てられます。

次の例では、ALTER TABLE文で表のサブパーティションのデータを切り捨てます。この例では、DROP STORAGE句を使用して、削除済の行で占有された領域を表領域のその他のスキーマ・オブジェクトで使用できるようにしています。データは切り捨てられますが、サブパーティションは削除されない点に注意してください。

例4-40 複数のサブパーティションの切捨て

```
CREATE TABLE sales_partition_truncate
  ( product_id       NUMBER(6) NOT NULL,
    customer_id      NUMBER    NOT NULL,
    channel_id       CHAR(1),
    promo_id         NUMBER(6),
    sales_date       DATE,
    quantity_sold    INTEGER,
    amount_sold      NUMBER(10,2)
  )
 PARTITION BY RANGE (sales_date)
 SUBPARTITION BY LIST (channel_id)
(  PARTITION q3_2018 VALUES LESS THAN (TO_DATE('1-OCT-2018','DD-MON-YYYY'))
   ( SUBPARTITION q3_2018_p_catalog VALUES ('C'),
     SUBPARTITION q3_2018_p_internet VALUES ('I'),
     SUBPARTITION q3_2018_p_partners VALUES ('P'),
     SUBPARTITION q3_2018_p_direct_sales VALUES ('S'),
     SUBPARTITION q3_2018_p_tele_sales VALUES ('T')
   ),
  PARTITION q4_2018 VALUES LESS THAN (TO_DATE('1-JAN-2019','DD-MON-YYYY'))
   ( SUBPARTITION q4_2018_p_catalog VALUES ('C'),
     SUBPARTITION q4_2018_p_internet VALUES ('I'),
     SUBPARTITION q4_2018_p_partners VALUES ('P'),
     SUBPARTITION q4_2018_p_direct_sales VALUES ('S'),
     SUBPARTITION q4_2018_p_tele_sales VALUES ('T')
   ),
  PARTITION q1_2019 VALUES LESS THAN (TO_DATE('1-APR-2019','DD-MON-YYYY'))
   ( SUBPARTITION q1_2019_p_catalog VALUES ('C')
   , SUBPARTITION q1_2019_p_internet VALUES ('I')
   , SUBPARTITION q1_2019_p_partners VALUES ('P')
   , SUBPARTITION q1_2019_p_direct_sales VALUES ('S')
   , SUBPARTITION q1_2019_p_tele_sales VALUES ('T')
   ),
  PARTITION q2_2019 VALUES LESS THAN (TO_DATE('1-JUL-2019','DD-MON-YYYY'))
   ( SUBPARTITION q2_2019_p_catalog VALUES ('C'),
     SUBPARTITION q2_2019_p_internet VALUES ('I'),
     SUBPARTITION q2_2019_p_partners VALUES ('P'),
     SUBPARTITION q2_2019_p_direct_sales VALUES ('S'),
     SUBPARTITION q2_2019_p_tele_sales VALUES ('T')
   ),
  PARTITION q3_2019 VALUES LESS THAN (TO_DATE('1-OCT-2019','DD-MON-YYYY'))
   ( SUBPARTITION q3_2019_p_catalog VALUES ('C'),
     SUBPARTITION q3_2019_p_internet VALUES ('I'),
     SUBPARTITION q3_2019_p_partners VALUES ('P'),
     SUBPARTITION q3_2019_p_direct_sales VALUES ('S'),
     SUBPARTITION q3_2019_p_tele_sales VALUES ('T')
   ),
  PARTITION q4_2019 VALUES LESS THAN (TO_DATE('1-JAN-2020','DD-MON-YYYY'))
   ( SUBPARTITION q4_2019_p_catalog VALUES ('C'),
     SUBPARTITION q4_2019_p_internet VALUES ('I'),
     SUBPARTITION q4_2019_p_partners VALUES ('P'),
     SUBPARTITION q4_2019_p_direct_sales VALUES ('S'),
     SUBPARTITION q4_2019_p_tele_sales VALUES ('T')
   )
  );

SELECT TABLE_NAME, PARTITION_NAME, SUBPARTITION_NAME FROM USER_TAB_SUBPARTITIONS
       WHERE TABLE_NAME ='SALES_PARTITION_TRUNCATE';
TABLE_NAME                  PARTITION_NAME    SUBPARTITION_NAME
--------------------------  ---------------   -------------------
SALES_PARTITION_TRUNCATE    Q1_2019           Q1_2019_P_CATALOG
```

```
SALES_PARTITION_TRUNCATE    Q1_2019           Q1_2019_P_DIRECT_SALES
...
30 rows selected.

INSERT INTO sales_partition_truncate VALUES (1001,100,'C',150,'10-SEP-2018',500,2000);
INSERT INTO sales_partition_truncate VALUES (1021,200,'C',160,'16-NOV-2018',100,1500);
INSERT INTO sales_partition_truncate VALUES (1001,100,'C',150,'10-FEB-2019',500,2000);
INSERT INTO sales_partition_truncate VALUES (1021,200,'S',160,'16-FEB-2019',100,1500);
INSERT INTO sales_partition_truncate VALUES (1002,110,'I',180,'15-JUN-2019',100,1000);
INSERT INTO sales_partition_truncate VALUES (5010,150,'P',200,'20-AUG-2019',1000,10000);
INSERT INTO sales_partition_truncate VALUES (1001,100,'T',150,'12-OCT-2019',500,2000);

SELECT * FROM sales_partition_truncate;
PRODUCT_ID CUSTOMER_ID C   PROMO_ID SALES_DAT QUANTITY_SOLD AMOUNT_SOLD
---------- ----------- - ---------- --------- ------------- -----------
      1001         100 C        150 10-SEP-18           500        2000
      1021         200 C        160 16-NOV-18           100        1500
      1001         100 C        150 10-FEB-19           500        2000
      1021         200 S        160 16-FEB-19           100        1500
      1002         110 I        180 15-JUN-19           100        1000
      5010         150 P        200 20-AUG-19          1000       10000
      1001         100 T        150 12-OCT-19           500        2000
7 rows selected.

ALTER TABLE sales_subpartition_truncate
   TRUNCATE SUBPARTITIONS q3_2018_p_catalog, q4_2018_p_catalog, q1_2019_p_catalog,
                          q2_2019_p_catalog, q3_2019_p_catalog, q4_2019_p_catalog
        DROP STORAGE;

SELECT * FROM sales_partition_truncate;
PRODUCT_ID CUSTOMER_ID C   PROMO_ID SALES_DAT QUANTITY_SOLD AMOUNT_SOLD
---------- ----------- - ---------- --------- ------------- -----------
      1021         200 S        160 16-FEB-19           100        1500
      1002         110 I        180 15-JUN-19           100        1000
      5010         150 P        200 20-AUG-19          1000       10000
      1001         100 T        150 12-OCT-19           500        2000
4 rows selected.

SELECT TABLE_NAME, PARTITION_NAME, SUBPARTITION_NAME FROM USER_TAB_SUBPARTITIONS
       WHERE TABLE_NAME ='SALES_PARTITION_TRUNCATE';
TABLE_NAME                  PARTITION_NAME    SUBPARTITION_NAME
--------------------------  ---------------   -------------------
SALES_PARTITION_TRUNCATE    Q1_2019           Q1_2019_P_CATALOG
SALES_PARTITION_TRUNCATE    Q1_2019           Q1_2019_P_DIRECT_SALES
...
30 rows selected.
```

4.4.13.4 カスケード・オプションを使用したパーティションの切捨て

TRUNCATE TABLE、ALTER TABLE TRUNCATE PARTITIONおよびALTER TABLE TRUNCATE SUBPARTITION SQL文のCASCADEオプションを使用して、参照パーティション化されている子表に切捨て操作をカスケードできます。

CASCADEオプションがTRUNCATE TABLEに指定されている場合、表の切捨て操作は、ON DELETE CASCADEを有効化している有効な参照制約を介してターゲット表を参照する子表も切り捨てます。このカスケード・アクションは、孫、ひ孫などに再帰的に適用されます。有効なON DELETE CASCADE参照制約に基づいて切り捨てる表セットを決定した後、このセットの表がセットの外側の子から有効な制約を介して参照されると、エラーが発生します。親および子が複数の参照制約で接続される場合、少なくとも1つの制約のON DELETE CASCADEが有効であると、親を対象とするTRUNCATE TABLE CASCADE操作に成功します。

権限は、操作に影響されるすべての表に必要です。DROP STORAGEまたはPURGE MATERIALIZED VIEW LOGなど、操作に指定される他のオプションは、操作に影響されるすべての表に適用されます。

CASCADEオプションを指定すると、TRUNCATE PARTITIONおよびTRUNCATE SUBPARTITION操作は、ターゲット表の子である参照パーティション表にカスケードされます。TRUNCATEは、参照パーティション階層の任意のレベルで指定でき、ターゲット表から開始される子表にカスケードします。権限は子表に必要ありませんが、表をパーティション化制約ではない有効な参照制約で参照できないなど、TRUNCATE操作の通常の制約が操作に影響されるすべての表に適用されます。

参照パーティションの子を持たない表に指定されている場合、CASCADEオプションは無視されます。DROP STORAGEまたはUPDATE INDEXESなど、操作に指定される他のオプションは、操作に影響されるすべての表に適用されます。

カスケード・オプションはデフォルトで無効になっているため、Oracle Database互換性に影響しません。

```
ALTER TABLE sales
     TRUNCATE PARTITION dec2016
     DROP STORAGE
     CASCADE
     UPDATE INDEXES;
```

4.5 パーティション表の削除について

パーティション表の削除は、非パーティション表の削除と同様です。

Oracle Databaseでは、パーティション表に対するDROP TABLE文は、非パーティション表に対する文と同様に処理されます。例外はPURGEキーワードを使用する場合です。

リソースの制限が発生しないようにするには、パーティション表に対するDROP TABLE...PURGE文を使用して複数のトランザクションで表を削除します。この場合、各トランザクションが、パーティションまたはサブパーティションのサブセットを削除してコミットします。表が削除されるのは、最後のトランザクションが終了したときです。

この動作には、DROP TABLE文に対するいくつかの変更が伴います。1つ目は、DROP TABLE...PURGE文が失敗した場合には修正処理を実行でき、必要な場合は文を再発行することです。文は失敗した場所から再開されます。2つ目は、次に示すデータ・ディクショナリ・ビューでSTATUS列の値をUNUSABLEに設定すると、DROP TABLE...PURGE文の実行中に表がunusableとマークされることです。

- USER_TABLES、ALL_TABLES、DBA_TABLES
- USER_PART_TABLES、ALL_PART_TABLES、DBA_PART_TABLES
- USER_OBJECT_TABLES、ALL_OBJECT_TABLES、DBA_OBJECT_TABLES

これらのビューのSTATUS列を問い合せることで、UNUSABLEとマークされたパーティション表をすべてリストできます。

DBA_TAB_PARTITIONSおよびDBA_TAB_SUBPARTITIONSなど、パーティション化に関連するその他のデータ・ディクショナリ・ビューに対する問合せでは、UNUSABLEとマークされた表に属する行は除外されます。

表がUNUSABLEとマークされた後に、その表に対して発行できる唯一の文は別のDROP TABLE...PURGE文で、これは前のDROP TABLE...PURGE文が失敗した場合に限られます。UNUSABLEとマークされた表に対して発行したその他の文はエラーになります。削除操作が完了するまで、表はUNUSABLEの状態のままです。

関連項目:

- パーティション化に関連する情報を含むビューのリストは、「パーティション表および索引の情報の表示」を参照してください
- DROP TABLE文の構文は、『Oracle Database SQL言語リファレンス』を参照してください
- この項に記載されているデータ・ディクショナリ・ビューの説明は、『Oracle Databaseリファレンス』を参照してください

4.6 パーティション表への非パーティション表の変更

非パーティション表をパーティション表に変更できます。

次の内容について説明します。

- オンライン再定義を使用したコレクション表のパーティション化
- パーティション表への非パーティション表の変換

関連項目:

表のパーティションの再定義の詳細は、『Oracle Database管理者ガイド』を参照してください

4.6.1 オンライン再定義を使用したコレクション表のパーティション化

Oracle Databaseは、1つ以上のパーティションを移動したり、DMLに対するパーティションの可用性にそれほど影響することなくパーティションの物理構造への他の変更を行うメカニズムを提供します。このメカニズムはオンライン表再編成と呼ばれます。

オンライン再定義を使用すると、非パーティション・コレクション表をパーティション・コレクション表にコピーできます。Oracle Databaseによって、コレクション表の該当するパーティションに行が挿入されます。例4-41は、Objects列内のネスト表でこの処理がどのように行われるかを示します。同様の例は、XMLType表または列内のOrdered Collection Type Tablesにも対応します。新たに定義するコレクション表で索引と制約を維持する場合は、copy_table_dependentsの操作で、索引と制約をコピーするために0またはfalseを指定します。ただし、コレクション表とそのパーティションの名前は、暫定表(例4-41のprint_media2)のものと同じです。コレクション表の名前を維持するために明示的なステップを実行する必要があります。

例4-41 コレクション表のパーティションの再定義

```
REM Connect as a user with appropriate privileges, then run the following
DROP USER eqnt CASCADE;
CREATE USER eqnt IDENTIFIED BY eqnt;
GRANT CONNECT, RESOURCE TO eqnt;

-- Grant privleges required for online redefinition.
GRANT EXECUTE ON DBMS_REDEFINITION TO eqnt;
GRANT ALTER ANY TABLE TO eqnt;
GRANT DROP ANY TABLE TO eqnt;
GRANT LOCK ANY TABLE TO eqnt;
GRANT CREATE ANY TABLE TO eqnt;
GRANT SELECT ANY TABLE TO eqnt;

-- Privileges required to perform cloning of dependent objects.
GRANT CREATE ANY TRIGGER TO eqnt;
GRANT CREATE ANY INDEX TO eqnt;

CONNECT eqnt/eqnt

CREATE TYPE textdoc_typ AS OBJECT ( document_typ VARCHAR2(32));
/
CREATE TYPE textdoc_tab AS TABLE OF textdoc_typ;
/

-- (old) non partitioned nested table
```

```
CREATE TABLE print_media
    ( product_id        NUMBER(6) primary key
    , ad_textdocs_ntab  textdoc_tab
    )
NESTED TABLE ad_textdocs_ntab STORE AS equi_nestedtab
(   (document_typ NOT NULL)
    STORAGE (INITIAL 8M)
)

;

-- Insert into base table
INSERT INTO print_media VALUES (1,
   textdoc_tab(textdoc_typ('xx'), textdoc_typ('yy')));
INSERT INTO print_media VALUES (11,
   textdoc_tab(textdoc_typ('aa'), textdoc_typ('bb')));
COMMIT;

-- Insert into nested table
INSERT INTO TABLE
  (SELECT p.ad_textdocs_ntab FROM print_media p WHERE p.product_id = 11)
   VALUES ('cc');

SELECT * FROM print_media;

PRODUCT_ID   AD_TEXTDOCS_NTAB(DOCUMENT_TYP)
----------   ------------------------------
         1   TEXTDOC_TAB(TEXTDOC_TYP('xx'), TEXTDOC_TYP('yy'))
        11   TEXTDOC_TAB(TEXTDOC_TYP('aa'), TEXTDOC_TYP('bb'), TEXTDOC_TYP('cc'))

-- Creating partitioned Interim Table
CREATE TABLE print_media2
    ( product_id        NUMBER(6)
    , ad_textdocs_ntab  textdoc_tab
    )
NESTED TABLE ad_textdocs_ntab STORE AS equi_nestedtab2
(   (document_typ NOT NULL)
    STORAGE (INITIAL 8M)
)
PARTITION BY RANGE (product_id)
(
    PARTITION P1 VALUES LESS THAN (10),
    PARTITION P2 VALUES LESS THAN (20)
);

EXEC dbms_redefinition.start_redef_table('eqnt', 'print_media', 'print_media2');

DECLARE
 error_count pls_integer := 0;
BEGIN
  dbms_redefinition.copy_table_dependents('eqnt', 'print_media', 'print_media2',
                                          0, true, false, true, false,
                                          error_count);

  dbms_output.put_line('errors := ' || to_char(error_count));
END;
/

EXEC  dbms_redefinition.finish_redef_table('eqnt', 'print_media', 'print_media2');
```

```
-- Drop the interim table
DROP TABLE print_media2;

-- print_media has partitioned nested table here

SELECT * FROM print_media PARTITION (p1);

PRODUCT_ID   AD_TEXTDOCS_NTAB(DOCUMENT_TYP)
----------   ------------------------------
         1   TEXTDOC_TAB(TEXTDOC_TYP('xx'), TEXTDOC_TYP('yy'))

SELECT * FROM print_media PARTITION (p2);

PRODUCT_ID   AD_TEXTDOCS_NTAB(DOCUMENT_TYP)
----------   ------------------------------
        11   TEXTDOC_TAB(TEXTDOC_TYP('aa'), TEXTDOC_TYP('bb'), TEXTDOC_TYP('cc'))
```

4.6.2 パーティション表への非パーティション表の変換

非パーティション表をパーティション表に変換するには、ALTER TABLE SQL文にMODIFY句を追加します。

また、キーワードONLINEを指定すると、変換が進行中の同時DML操作が可能になります。

次に、パーティション表へのオンライン変換にONLINEキーワードを使用するALTER TABLE文の例を示します。

例4-42 パーティション表へのオンライン変換にALTER TABLEのMODIFY句を使用する方法

```
ALTER TABLE employees_convert MODIFY
  PARTITION BY RANGE (employee_id) INTERVAL (100)
  ( PARTITION P1 VALUES LESS THAN (100),
    PARTITION P2 VALUES LESS THAN (500)
   ) ONLINE
  UPDATE INDEXES
 ( IDX1_SALARY LOCAL,
   IDX2_EMP_ID GLOBAL PARTITION BY RANGE (employee_id)
  ( PARTITION IP1 VALUES LESS THAN (MAXVALUE))
 );
```

UPDATE INDEXES句を使用する際の考慮事項

UPDATE INDEXES句を使用する際は、次のことに注意してください。

- この句を使用して、索引のパーティション化状態および変換される索引の記憶域プロパティを変更できます。
- UPDATE INDEXES句の指定はオプションです。

  索引は、パーティション表へのオンライン変換とオフライン変換の両方でメンテナンスされます。
- この句は、索引の元のリストが定義されている列を変更できません。
- この句は、索引の一意性プロパティまたは他の索引プロパティを変更できません。
- いずれかの索引の表領域を指定しない場合は、次の表領域のデフォルトが適用されます。
  - 変換後のローカル索引が表パーティションと連結します。
  - 変換後のグローバル索引が、非パーティション表の元のグローバル索引の同じ表領域に存在します。
- INDEXES句を指定しないか、INDEXES句が元の非パーティション表のすべての索引を指定しない場合、次のデフォルトの動作がすべての未指定の索引に適用されます。

- グローバル・パーティション索引は同じまま、元のパーティション形状が保持されます。
- 非同一キー索引が、グローバル非パーティション索引になります。
- 同一キー索引はローカル・パーティション索引に変換されます。

  同一キーとは、パーティション・キー列が索引定義に含まれていることを意味しますが、索引定義は、パーティション化キーのみを含めることに限定されません。
- ビットマップ索引は、同一キーかどうかに関係なく、ローカル・パーティション索引になります。

  ビットマップ索引は、常にローカル・パーティション索引である必要があります。

- 変換操作は、ドメイン索引がある場合は実行できません。

4.7 ハイブリッド・パーティション表の管理

この項の内容は、次のとおりです。

- ハイブリッド・パーティション表の作成
- ハイブリッド・パーティション表への変換
- 内部パーティション表へのハイブリッド・パーティション表の変換
- ハイブリッド・パーティション表でのADOの使用
- ハイブリッド・パーティション表のパーティションの分割

関連項目:

- ハイブリッド・パーティション表の制限事項に関する情報などの概要は、「ハイブリッド・パーティション表」を参照

4.7.1 ハイブリッド・パーティション表の作成

CREATE TABLE文のEXTERNAL PARTITION ATTRIBUTES句を使用して、表のハイブリッド・パーティション化を決定できます。表のパーティションは、外部または内部にすることができます。

ハイブリッド・パーティション表では、パーティションは、データベース・データファイル(内部パーティション)と外部のファイルおよびソース(外部パーティション)の両方に存在できます。ハイブリッド・パーティション表を作成して問い合せることで、内部と外部の両方のパーティションに含まれるデータについて、プルーニングなどのクラシック・パーティション表を使用したパーティション化の利点を活用できます。

ハイブリッド・パーティション表に表レベルの外部パラメータを指定する場合、CREATE TABLE文のEXTERNAL PARTITION ATTRIBUTES句は、表レベルで定義されます。次に例を示します。

- ORACLE_LOADER、ORACLE_DATAPUMP、ORACLE_HDFS、ORACLE_HIVEなどのアクセス・ドライバ・タイプ
- すべての外部パーティション・ファイルのデフォルト・ディレクトリ
- アクセス・パラメータ

PARTITION句のEXTERNAL句は、パーティションを外部パーティションとして定義します。EXTERNAL句が存在しない場合、パーティションは内部パーティションです。各外部パーティションに対して、ディレクトリなど、表レベルで定義されたデフォルト属性とは異なる属性を指定できます。たとえば、例4-43では、パーティションsales_data2、sales_data3およびsales_data_acfsのDEFAULT DIRECTORY値は、EXTERNAL PARTITION ATTRIBUTES句で定義されたDEFAULT DIRECTORY値とは異なります。

外部パーティションに対して外部ファイルが定義されていない場合、外部パーティションは空です。ALTER TABLE MODIFY PARTITION文を使用して、外部ファイルを移入できます。少なくとも1つのパーティションが内部パーティションである必要があることに注意してください。

例4-43では、ハイブリッド・レンジ・パーティション表は、4つの外部パーティションと2つの内部パーティションで作成されます。外部カンマ区切り(CSV)データファイルは、DEFAULT DIRECTORY句で定義されたsales_data、sales_data2、sales_data3およびsales_data_acfsディレクトリに格納されます。sales_dataは、EXTERNAL PARTITION ATTRIBUTES句で、全体のDEFAULT DIRECTORYとして定義されます。他のディレクトリは、パーティション・レベルで定義されます。sales_2014およびsales_2015は、内部パーティションです。データ・ディレクトリsales_data_acfsは、該当するストレージ・オプションの使用を示

すためにOracle ACFSファイルシステムに格納されます。

例4-44では、追加の外部パーティションがハイブリッド・レンジ・パーティション表に追加されます。

例4-43 ハイブリッド・レンジ・パーティション表の作成

```
REM Connect as a user with appropriate privileges,
REM then run the following to set up data directories that contain the data files
CREATE DIRECTORY sales_data AS '/u01/my_data/sales_data1';
GRANT READ,WRITE ON DIRECTORY sales_data TO hr;

CREATE DIRECTORY sales_data2 AS '/u01/my_data/sales_data2';
GRANT READ,WRITE ON DIRECTORY sales_data2 TO hr;

CREATE DIRECTORY sales_data3 AS '/u01/my_data/sales_data3';
GRANT READ,WRITE ON DIRECTORY sales_data3 TO hr;

REM set up a data directory on an Oracle ACFS mount point (file system)
CREATE DIRECTORY sales_data_acfs AS '/u01/acfsmounts/acfs1';
GRANT READ,WRITE ON DIRECTORY sales_data_acfs TO hr;

CONNECT AS hr, run the following
CREATE TABLE hybrid_partition_table
  ( prod_id       NUMBER        NOT NULL,
    cust_id       NUMBER        NOT NULL,
    time_id       DATE          NOT NULL,
    channel_id    NUMBER        NOT NULL,
    promo_id      NUMBER        NOT NULL,
    quantity_sold NUMBER(10,2)  NOT NULL,
    amount_sold   NUMBER(10,2)  NOT NULL
  )
    EXTERNAL PARTITION ATTRIBUTES (
      TYPE ORACLE_LOADER
      DEFAULT DIRECTORY sales_data
       ACCESS PARAMETERS(
         FIELDS TERMINATED BY ','
         (prod_id,cust_id,time_id DATE 'dd-mm-yyyy',channel_id,promo_id,quantity_sold,amount_sold)
       )
      REJECT LIMIT UNLIMITED
     )
    PARTITION BY RANGE (time_id)
    (PARTITION sales_2014 VALUES LESS THAN (TO_DATE('01-01-2015','dd-mm-yyyy')),
     PARTITION sales_2015 VALUES LESS THAN (TO_DATE('01-01-2016','dd-mm-yyyy')),
     PARTITION sales_2016 VALUES LESS THAN (TO_DATE('01-01-2017','dd-mm-yyyy')) EXTERNAL
          LOCATION ('sales2016_data.txt'),
     PARTITION sales_2017 VALUES LESS THAN (TO_DATE('01-01-2018','dd-mm-yyyy')) EXTERNAL
          DEFAULT DIRECTORY sales_data2 LOCATION ('sales2017_data.txt'),
     PARTITION sales_2018 VALUES LESS THAN (TO_DATE('01-01-2019','dd-mm-yyyy')) EXTERNAL
          DEFAULT DIRECTORY sales_data3 LOCATION ('sales2018_data.txt'),
     PARTITION sales_2019 VALUES LESS THAN (TO_DATE('01-01-2020','dd-mm-yyyy')) EXTERNAL
          DEFAULT DIRECTORY sales_data_acfs LOCATION ('sales2019_data.txt')
 );
```

例4-44 ハイブリッド・レンジ・パーティション表への外部パーティションの追加

```
ALTER TABLE hybrid_partition_table
      ADD PARTITION sales_2020 VALUES LESS THAN (TO_DATE('01-01-2021','dd-mm-yyyy'))
          EXTERNAL DEFAULT DIRECTORY sales_data_acfs  LOCATION ('sales2020_data.txt');
```

関連項目:

- ハイブリッド・パーティション表

4.7.2 ハイブリッド・パーティション表への変換

内部パーティションのみを含む表は、ハイブリッド・パーティション表に変換できます。

例4-45では、内部レンジ・パーティション表がハイブリッド・パーティション表に変換されます。まず、既存の表に外部パーティション属性を追加してから、外部パーティションを追加する必要があります。少なくとも1つのパーティションが内部パーティションである必要があることに注意してください。

例4-45 ハイブリッド・レンジ・パーティション表への変換

```
CREATE TABLE internal_to_hypt_table (
    prod_id       NUMBER        NOT NULL,
    cust_id       NUMBER        NOT NULL,
    time_id       DATE          NOT NULL,
    channel_id    NUMBER        NOT NULL,
    promo_id      NUMBER        NOT NULL,
    quantity_sold NUMBER(10,2)  NOT NULL,
    amount_sold   NUMBER(10,2)  NOT NULL
  )
   PARTITION by range (time_id)
   (PARTITION sales_2014 VALUES LESS THAN (TO_DATE('01-01-2015','dd-mm-yyyy'))
  );

SELECT HYBRID FROM USER_TABLES WHERE TABLE_NAME = 'INTERNAL_TO_HYPT_TABLE';
HYB
---
NO

ALTER TABLE internal_to_hypt_table
  ADD EXTERNAL PARTITION ATTRIBUTES
   (TYPE ORACLE_LOADER
     DEFAULT DIRECTORY sales_data
     ACCESS PARAMETERS (
       FIELDS TERMINATED BY ','
       (prod_id,cust_id,time_id DATE 'dd-mm-yyyy',channel_id,promo_id,quantity_sold,amount_sold)
     )
   )
;

ALTER TABLE internal_to_hypt_table
      ADD PARTITION sales_2015 VALUES LESS THAN (TO_DATE('01-01-2016','dd-mm-yyyy'))
          EXTERNAL LOCATION ('sales2015_data.txt');

ALTER TABLE internal_to_hypt_table
      ADD PARTITION sales_2016 VALUES LESS THAN (TO_DATE('01-01-2017','dd-mm-yyyy'))
          EXTERNAL LOCATION ('sales2016_data.txt');

SELECT HYBRID FROM USER_TABLES WHERE TABLE_NAME = 'INTERNAL_TO_HYPT_TABLE';
HYB
---
YES

SELECT DEFAULT_DIRECTORY_NAME FROM USER_EXTERNAL_TABLES WHERE TABLE_NAME = 'INTERNAL_TO_HYPT_TABLE';
DEFAULT_DIRECTORY_NAME
```

```
--------------------------------------------------------------------------------
SALES_DATA
```

関連項目:

- ハイブリッド・パーティション表

4.7.3 内部パーティション表へのハイブリッド・パーティション表の変換

ハイブリッド・パーティション表は、内部パーティションのみを含む表に変換できます。

例4-46では、ハイブリッド・パーティション表が内部レンジ・パーティション表に変換されます。まず、外部パーティションを削除してから、外部パーティション属性を削除できます。

例4-46 ハイブリッド・パーティション表から内部表への変換

```
CREATE TABLE hypt_to_int_table
  ( prod_id       NUMBER        NOT NULL,
    cust_id       NUMBER        NOT NULL,
    time_id       DATE          NOT NULL,
    channel_id    NUMBER        NOT NULL,
    promo_id      NUMBER        NOT NULL,
    quantity_sold NUMBER(10,2)  NOT NULL,
    amount_sold   NUMBER(10,2)  NOT NULL
  )
    EXTERNAL PARTITION ATTRIBUTES (
     TYPE ORACLE_LOADER
     DEFAULT DIRECTORY sales_data
       ACCESS PARAMETERS(
        FIELDS TERMINATED BY ','
        (prod_id,cust_id,time_id DATE 'dd-mm-yyyy',channel_id,promo_id,quantity_sold,amount_sold)
       )
      REJECT LIMIT UNLIMITED
     )
    PARTITION BY RANGE (time_id)
    (PARTITION sales_2014 VALUES LESS THAN (TO_DATE('01-01-2015','dd-mm-yyyy')),
     PARTITION sales_2015 VALUES LESS THAN (TO_DATE('01-01-2016','dd-mm-yyyy')),
     PARTITION sales_2016 VALUES LESS THAN (TO_DATE('01-01-2017','dd-mm-yyyy'))
          EXTERNAL LOCATION ('sales2016_data.txt'),
     PARTITION sales_2017 VALUES LESS THAN (TO_DATE('01-01-2018','dd-mm-yyyy'))
          EXTERNAL DEFAULT DIRECTORY sales_data2 LOCATION ('sales2017_data.txt'),
     PARTITION sales_2018 VALUES LESS THAN (TO_DATE('01-01-2019','dd-mm-yyyy'))
          EXTERNAL DEFAULT DIRECTORY sales_data3 LOCATION ('sales2018_data.txt'),
     PARTITION sales_2019 VALUES LESS THAN (TO_DATE('01-01-2020','dd-mm-yyyy'))
          EXTERNAL DEFAULT DIRECTORY sales_data_acfs LOCATION ('sales2019_data.txt')
 );

SELECT HYBRID FROM USER_TABLES WHERE TABLE_NAME = 'HYPT_TO_INT_TABLE';
HYB
---
YES

ALTER TABLE hypt_to_int_table DROP PARTITION sales_2016;
ALTER TABLE hypt_to_int_table DROP PARTITION sales_2017;
ALTER TABLE hypt_to_int_table DROP PARTITION sales_2018;
ALTER TABLE hypt_to_int_table DROP PARTITION sales_2019;
```

```
ALTER TABLE hypt_to_int_table DROP EXTERNAL PARTITION ATTRIBUTES();

SELECT HYBRID FROM USER_TABLES WHERE TABLE_NAME = 'HYPT_TO_INT_TABLE';
HYB
---
NO
```

関連項目:

- ハイブリッド・パーティション表

4.7.4 ハイブリッド・パーティション表でのADOの使用

ハイブリッド・パーティション表では、自動データ最適化(ADO)ポリシーを条件付きで使用できます。

例4-47では、ADOポリシーが表の内部パーティションにのみ定義されています。

例4-47 ハイブリッド・パーティション表でのADOの使用

```
SQL> CREATE TABLE hypt_ado_table
  ( prod_id       NUMBER        NOT NULL,
    cust_id       NUMBER        NOT NULL,
    time_id       DATE          NOT NULL,
    channel_id    NUMBER        NOT NULL,
    promo_id      NUMBER        NOT NULL,
    quantity_sold NUMBER(10,2)  NOT NULL,
    amount_sold   NUMBER(10,2)  NOT NULL
  )
    EXTERNAL PARTITION ATTRIBUTES (
     TYPE ORACLE_LOADER
     DEFAULT DIRECTORY sales_data
       ACCESS PARAMETERS(
        FIELDS TERMINATED BY ','
        (prod_id,cust_id,time_id DATE 'dd-mm-yyyy',channel_id,promo_id,quantity_sold,amount_sold)
       )
      REJECT LIMIT UNLIMITED
     )
    PARTITION BY RANGE (time_id)
    (PARTITION sales_2014 VALUES LESS THAN (TO_DATE('01-01-2015','dd-mm-yyyy')),
     PARTITION sales_2015 VALUES LESS THAN (TO_DATE('01-01-2016','dd-mm-yyyy')),
     PARTITION sales_2016 VALUES LESS THAN (TO_DATE('01-01-2017','dd-mm-yyyy'))
          EXTERNAL LOCATION ('sales2016_data.txt'),
     PARTITION sales_2017 VALUES LESS THAN (TO_DATE('01-01-2018','dd-mm-yyyy'))
          EXTERNAL DEFAULT DIRECTORY sales_data2 LOCATION ('sales2017_data.txt'),
     PARTITION sales_2018 VALUES LESS THAN (TO_DATE('01-01-2019','dd-mm-yyyy'))
          EXTERNAL DEFAULT DIRECTORY sales_data3 LOCATION ('sales2018_data.txt'),
     PARTITION sales_2019 VALUES LESS THAN (TO_DATE('01-01-2020','dd-mm-yyyy'))
          EXTERNAL DEFAULT DIRECTORY sales_data4 LOCATION ('sales2019_data.txt')
 );
Table created.

SQL> SELECT HYBRID FROM USER_TABLES WHERE TABLE_NAME = 'HYPT_ADO_TABLE';
HYB
---
YES
```

```
SQL> ALTER TABLE hypt_ado_table MODIFY PARTITION sales_2014 ILM ADD POLICY ROW STORE COMPRESS ADVANCED
ROW AFTER 6 MONTHS OF NO MODIFICATION;
Table altered.

SQL> ALTER TABLE hypt_ado_table MODIFY PARTITION sales_2015 ILM ADD POLICY ROW STORE COMPRESS ADVANCED
ROW AFTER 6 MONTHS OF NO MODIFICATION;
Table altered.

SQL> SELECT POLICY_NAME, POLICY_TYPE, ENABLED FROM USER_ILMPOLICIES;
POLICY_NAME    POLICY_TYPE     ENA
-------------  --------------- -----
P1             DATA MOVEMENT   YES
P2             DATA MOVEMENT   YES
```

関連項目:

- ADOポリシーの詳細は、「自動データ最適化の使用」を参照

4.7.5 ハイブリッド・パーティション表のパーティションの分割

例4-48では、デフォルト(MAXVALUE)パーティションが、新しいパーティションと既存のデフォルト位置の2つに分割されます。デフォルト・パーティションは、その他のパーティションを分割する場合と同様に分割できます。

例4-48 ハイブリッド・パーティション表のデフォルト・パーティションの分割

```
CREATE TABLE hybrid_split_table
  ( prod_id       NUMBER        NOT NULL,
    cust_id       NUMBER        NOT NULL,
    time_id       DATE          NOT NULL,
    channel_id    NUMBER        NOT NULL,
    promo_id      NUMBER        NOT NULL,
    quantity_sold NUMBER(10,2)  NOT NULL,
    amount_sold   NUMBER(10,2)  NOT NULL
  )
    EXTERNAL PARTITION ATTRIBUTES (
     TYPE ORACLE_LOADER
     DEFAULT DIRECTORY sales_data
       ACCESS PARAMETERS(
        FIELDS TERMINATED BY ','
        (prod_id,cust_id,time_id DATE 'dd-mm-yyyy',channel_id,promo_id,quantity_sold,amount_sold)
       )
      REJECT LIMIT UNLIMITED
     )
    PARTITION BY RANGE (time_id)
     (PARTITION sales_2016 VALUES LESS THAN (TO_DATE('01-01-2017','dd-mm-yyyy'))
          EXTERNAL LOCATION ('sales2016_data.txt'),
      PARTITION sales_2017 VALUES LESS THAN (TO_DATE('01-01-2018','dd-mm-yyyy'))
          EXTERNAL LOCATION ('sales2017_data.txt'),
     PARTITION sales_2018 VALUES LESS THAN (TO_DATE('01-01-2019','dd-mm-yyyy')),
     PARTITION sales_2019 VALUES LESS THAN (TO_DATE('01-01-2020','dd-mm-yyyy')),
     PARTITION sales_future VALUES LESS THAN (MAXVALUE)
);

SELECT HYBRID FROM USER_TABLES WHERE TABLE_NAME = 'HYBRID_SPLIT_TABLE';
HYB
```

```
---
YES

SELECT DEFAULT_DIRECTORY_NAME FROM USER_EXTERNAL_TABLES WHERE TABLE_NAME = 'HYBRID_SPLIT_TABLE';
DEFAULT_DIRECTORY_NAME
--------------------------------------------------------------------------------
SALES_DATA

INSERT INTO hybrid_split_table VALUES (1001,100,TO_DATE('10-02-2018','dd-mm-yyyy'),10,15,500,7500);
INSERT INTO hybrid_split_table VALUES (1002,110,TO_DATE('15-06-2018','dd-mm-yyyy'),12,18,100,3200);
...
INSERT INTO hybrid_split_table VALUES (1002,110,TO_DATE('12-01-2019','dd-mm-yyyy'),12,18,150,4800);
INSERT INTO hybrid_split_table VALUES (1001,100,TO_DATE('16-02-2019','dd-mm-yyyy'),10,15,400,6500);
...
INSERT INTO hybrid_split_table VALUES (1002,110,TO_DATE('19-02-2020','dd-mm-yyyy'),12,18,150,4800);
INSERT INTO hybrid_split_table VALUES (1001,100,TO_DATE('12-03-2020','dd-mm-yyyy'),10,15,400,6500);
...

SELECT * FROM hybrid_split_table PARTITION(sales_2016);
   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      1001        100 10-JAN-16         10         15           500        7500
      1002        110 25-JAN-16         12         18           100        3200
...

SELECT * FROM hybrid_split_table PARTITION(sales_2017);
   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      1002        110 15-JAN-17         12         18           100        3200
      1001        100 10-FEB-17         10         15           500        7500
...

SELECT * FROM hybrid_split_table PARTITION(sales_2018);
   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      1001        100 10-FEB-18         10         15           500        7500
      1002        110 15-JUN-18         12         18           100        3200
...

SELECT * FROM hybrid_split_table PARTITION(sales_2019);
   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      1002        110 12-JAN-19         12         18           150        4800
      1001        100 16-FEB-19         10         15           400        6500
...

SELECT * FROM hybrid_split_table PARTITION(sales_future);
   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      1002        110 19-FEB-20         12         18           150        4800
      1001        100 12-MAR-20         10         15           400        6500
      1001        100 31-MAR-20         10         15           600        8000
      2105        101 25-APR-20         12         19           100        3000

ALTER TABLE hybrid_split_table
  SPLIT PARTITION sales_future INTO
  (PARTITION sales_2020 VALUES LESS THAN (TO_DATE('01-01-2021','dd-mm-yyyy')),
   PARTITION sales_future
   );
```

```
SELECT * FROM hybrid_split_table PARTITION(sales_2020);
   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      1002        110 19-FEB-20         12         18           150        4800
      1001        100 12-MAR-20         10         15           400        6500
      1001        100 31-MAR-20         10         15           600        8000
      2105        101 25-APR-20         12         19           100        3000

SELECT * FROM hybrid_split_table PARTITION(sales_future);
no rows selected
```

4.7.6 ハイブリッド・パーティション表のデータの交換

ハイブリッド・パーティション表の内部パーティションのデータを外部非パーティション表と交換したり、外部非パーティション表のデータをハイブリッド・パーティション表の内部パーティションと交換できます。Oracleは内部ストレージと外部ストレージ間の交換をサポートしていますが、これらの層の間の移動操作はサポートしていません。内部ストレージと外部ストレージの間でのデータの移動は、交換前の個別の操作です。

例4-49 ハイブリッド・パーティション表の内部パーティションのデータと外部非パーティション表の交換

この例では、ハイブリッド・パーティション表の内部パーティションのデータが、まったく同じデータを含むパーティション化されていない外部表とのパーティション交換を使用して外部ストレージに移動されます。

TYPE ORACLE_DATAPUMPのハイブリッド・パーティション表を作成します。

```
CREATE TABLE hybrid_datapump_sales
  ( prod_id  NUMBER     NOT NULL,
    cust_id       NUMBER        NOT NULL,
    time_id       DATE          NOT NULL,
    channel_id    NUMBER        NOT NULL,
    promo_id      NUMBER        NOT NULL,
    quantity_sold NUMBER(10,2)  NOT NULL,
    amount_sold   NUMBER(10,2)  NOT NULL
  )
  EXTERNAL PARTITION ATTRIBUTES
  (TYPE ORACLE_DATAPUMP
   DEFAULT DIRECTORY sales_data
   ACCESS PARAMETERS (NOLOGFILE)
  )
  PARTITION by range (time_id)
  (
   PARTITION sales_old VALUES LESS THAN (TO_DATE('01-01-2018', 'DD-MM-YYYY'))
     EXTERNAL LOCATION ('sales_old.dmp'),
   PARTITION sales_2018 VALUES LESS THAN (TO_DATE('01-01-2019','dd-mm-yyyy')),
   PARTITION sales_2019 VALUES LESS THAN (TO_DATE('01-01-2020','dd-mm-yyyy')),
   PARTITION sales_2020 VALUES LESS THAN (TO_DATE('01-01-2021','dd-mm-yyyy')),
   PARTITION sales_future VALUES LESS THAN (MAXVALUE)
);
```

この例のデータをハイブリッド・パーティション表hybrid_datapump_salesに移入します。

```
SELECT * FROM hybrid_datapump_sales PARTITION(sales_2018);

   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      1001        100 10-FEB-18         10         15           500        7500
      1002        110 15-JUN-18         12         18           100        3200
      1002        110 30-MAR-18         10         15           500        6500
```

```
      2105         102 21-APR-18          18          12          100         2000
      1200         155 30-APR-18          20          20          300         3600
```

sales_2018パーティションと同じ構造で外部表を作成します。SELECT句で、データの移動を完了します。交換の前にデータ移動操作を行う必要があります。

```
CREATE TABLE year_2018_datapump
   ORGANIZATION EXTERNAL
   ( TYPE ORACLE_DATAPUMP DEFAULT DIRECTORY sales_data
     ACCESS PARAMETERS (NOLOGFILE) LOCATION ('sales_2018.dmp')
    )
   AS SELECT * FROM hybrid_datapump_sales PARTITION(sales_2018);
```

sales_2018パーティションのデータを外部表のデータと交換します。

```
ALTER TABLE hybrid_datapump_sales
      EXCHANGE PARTITION(sales_2018) WITH TABLE year_2018_datapump;

SELECT * FROM year_2018_datapump;

   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      1001        100 10-FEB-18         10         15           500        7500
      1002        110 15-JUN-18         12         18           100        3200
      1002        110 30-MAR-18         10         15           500        6500
      2105        102 21-APR-18         18         12           100        2000
      1200        155 30-APR-18         20         20           300        3600
```

例4-50 外部非パーティション表のデータとハイブリッド・パーティション表の内部パーティションの交換

この例では、ハイブリッド・パーティション表のパーティションに新しいデータを追加するために、外部表のデータがハイブリッド・パーティション表の内部パーティションと交換されます。外部表にロードされたテキスト・データは、最初に一時的な非パーティション内部表にコピーされます。次に、非パーティション内部表がハイブリッド・パーティション表の内部パーティションと交換されます。

TYPE ORACLE_DATAPUMPのハイブリッド・パーティション表を作成します。

```
CREATE TABLE hybrid_datapump_sales
    ( prod_id  NUMBER     NOT NULL,
      cust_id       NUMBER        NOT NULL,
      time_id       DATE          NOT NULL,
      channel_id    NUMBER        NOT NULL,
      promo_id      NUMBER        NOT NULL,
      quantity_sold NUMBER(10,2)  NOT NULL,
      amount_sold   NUMBER(10,2)  NOT NULL
     )
     EXTERNAL PARTITION ATTRIBUTES
      (TYPE ORACLE_DATAPUMP
       DEFAULT DIRECTORY sales_data
       ACCESS PARAMETERS (NOLOGFILE)
      )
     PARTITION by range (time_id)
     (
      PARTITION sales_old VALUES LESS THAN (TO_DATE('01-01-2018', 'dd-mm-yyyy'))
       EXTERNAL LOCATION ('sales_old.dmp'),
      PARTITION sales_2018 VALUES LESS THAN (TO_DATE('01-01-2019','dd-mm-yyyy')),
      PARTITION sales_2019 VALUES LESS THAN (TO_DATE('01-01-2020','dd-mm-yyyy')),
      PARTITION sales_2020 VALUES LESS THAN (TO_DATE('01-01-2021','dd-mm-yyyy')),
      PARTITION sales_future VALUES LESS THAN (MAXVALUE)
  );
```

sales_2020パーティションにレコードがないことに注意してください。

```
SELECT * FROM hybrid_datapump_sales PARTITION(sales_2020);

no rows selected
```

この例では、外部表を作成し、アプリケーションによって生成されたテキスト・ファイルをロードします。

```
CREATE TABLE ext_sales_year_2020
        ( prod_id       NUMBER        NOT NULL,
          cust_id       NUMBER        NOT NULL,
          time_id       DATE          NOT NULL,
          channel_id    NUMBER        NOT NULL,
          promo_id      NUMBER        NOT NULL,
          quantity_sold NUMBER(10,2)  NOT NULL,
          amount_sold   NUMBER(10,2)  NOT NULL
         )
        ORGANIZATION EXTERNAL
        (
         TYPE ORACLE_LOADER DEFAULT DIRECTORY sales_data
         ACCESS PARAMETERS (
         FIELDS TERMINATED BY ','
         (prod_id,cust_id,time_id DATE 'dd-mm-yyyy',channel_id,promo_id,quantity_sold,amount_sold)
         )
         LOCATION ('sales2020_data.txt')
 );

SELECT * FROM ext_sales_year_2020;

   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      1001        100 10-JAN-20         10         15           500        7500
      1002        110 15-JAN-20         12         18           100        3200
      1001        100 20-JAN-20         10         15           500        7500
      2105        101 15-FEB-20         12         19            10         300
      2105        102 21-MAR-20         18         12           100        2000
      1200        155 30-MAR-20         20         20           300        3600
      1400        165 05-JUN-20         22         15           100        4000
      2105        125 05-JUN-20         12         16            40        8500
      2105        302 15-SEP-20         10         11            75        4350
      2108        305 18-NOV-20         10         11            70        4250

10 rows selected.
```

ハイブリッド・パーティション表とデータを交換するための一時内部表を作成します。

```
CREATE TABLE sales_year_2020 AS SELECT * FROM ext_sales_year_2020;

SELECT * FROM sales_year_2020;

   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      1001        100 10-JAN-20         10         15           500        7500
      1002        110 15-JAN-20         12         18           100        3200
      1001        100 20-JAN-20         10         15           500        7500
      2105        101 15-FEB-20         12         19            10         300
      2105        102 21-MAR-20         18         12           100        2000
      1200        155 30-MAR-20         20         20           300        3600
      1400        165 05-JUN-20         22         15           100        4000
      2105        125 05-JUN-20         12         16            40        8500
```

```
      2105         302 15-SEP-20         10         11         75       4350
      2108         305 18-NOV-20         10         11         70       4250

10 rows selected.
```

ハイブリッド・パーティション表にデータをロードするために、一時内部表のデータをsales_2020パーティションと交換します。

```
ALTER TABLE hybrid_datapump_sales
       EXCHANGE PARTITION(sales_2020) WITH TABLE sales_year_2020;

SELECT * FROM hybrid_datapump_sales PARTITION(sales_2020);

   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      1001        100 10-JAN-20         10         15           500        7500
      1002        110 15-JAN-20         12         18           100        3200
      1001        100 20-JAN-20         10         15           500        7500
      2105        101 15-FEB-20         12         19            10         300
      2105        102 21-MAR-20         18         12           100        2000
      1200        155 30-MAR-20         20         20           300        3600
      1400        165 05-JUN-20         22         15           100        4000
      2105        125 05-JUN-20         12         16            40        8500
      2105        302 15-SEP-20         10         11            75        4350
      2108        305 18-NOV-20         10         11            70        4250

10 rows selected.
```

4.8 パーティション表および索引の情報の表示

Oracle Databaseのビューでパーティション表および索引に関する情報を表示できます。

表4-4に、パーティション表および索引に固有の情報を含むビューを示します。

表4-4 パーティション表および索引に固有の情報を含むビュー

ビュー	説明
DBA_PART_TABLES ALL_PART_TABLES USER_PART_TABLES	DBA ビューは、データベース内のすべてのパーティション表に関するパーティション化情報を表示します。ALL ビューは、ユーザーがアクセス可能なすべてのパーティション表に関するパーティション化情報を表示します。USER ビューは、ユーザーが所有するパーティション表に関するパーティション化情報に制限されています。
DBA_TAB_PARTITIONS ALL_TAB_PARTITIONS USER_TAB_PARTITIONS	パーティション・レベルのパーティション化情報、パーティションの記憶域パラメータ、および DBMS_STATS パッケージまたは ANALYZE 文により生成されたパーティション統計が表示されます。
DBA_TAB_SUBPARTITIONS ALL_TAB_SUBPARTITIONS USER_TAB_SUBPARTITIONS	サブパーティション・レベルのパーティション化情報、サブパーティションの記憶域パラメータ、および DBMS_STATS パッケージまたは ANALYZE 文により生成されたサブパーティション統計が表示されます。
DBA_PART_KEY_COLUMNS ALL_PART_KEY_COLUMNS USER_PART_KEY_COLUMNS	パーティション表のパーティション化キー列が表示されます。
DBA_SUBPART_KEY_COLUMNS ALL_SUBPART_KEY_COLUMNS USER_SUBPART_KEY_COLUMNS	コンポジット・パーティション表(およびコンポジット・パーティション表のローカル索引)のサブパーティション化キー列が表示されます。
DBA_PART_COL_STATISTICS ALL_PART_COL_STATISTICS USER_PART_COL_STATISTICS	表のパーティションに関する列統計およびヒストグラム情報が表示されます。
DBA_SUBPART_COL_STATISTICS ALL_SUBPART_COL_STATISTICS	表のサブパーティションに関する列統計およびヒストグラム情報が表示されます。

ビュー	説明
USER_SUBPART_COL_STATISTICS	
DBA_PART_HISTOGRAMS ALL_PART_HISTOGRAMS USER_PART_HISTOGRAMS	表パーティションのヒストグラムに関するヒストグラム・データ(各ヒストグラムのエンドポイント)が表示されます。
DBA_SUBPART_HISTOGRAMS ALL_SUBPART_HISTOGRAMS USER_SUBPART_HISTOGRAMS	表サブパーティションのヒストグラムに関するヒストグラム・データ(各ヒストグラムのエンドポイント)が表示されます。
DBA_PART_INDEXES ALL_PART_INDEXES USER_PART_INDEXES	パーティション索引のパーティション化情報が表示されます。
DBA_IND_PARTITIONS ALL_IND_PARTITIONS USER_IND_PARTITIONS	索引パーティションのパーティション・レベルのパーティション化情報、パーティションの記憶域パラメータ、DBMS_STATS パッケージまたは ANALYZE 文により収集された統計が表示されます。
DBA_IND_SUBPARTITIONS ALL_IND_SUBPARTITIONS USER_IND_SUBPARTITIONS	索引サブパーティションのパーティション・レベルのパーティション化情報、パーティションの記憶域パラメータ、DBMS_STATS パッケージまたは ANALYZE 文により収集された統計が表示されます。
DBA_SUBPARTITION_TEMPLATES ALL_SUBPARTITION_TEMPLATES USER_SUBPARTITION_TEMPLATES	既存のサブパーティション・テンプレートの情報が表示されます。

関連項目:

- データベース・ビューの説明は、『Oracle Databaseリファレンス』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 SQLチューニング・ガイド』を参照してください
- 表、索引およびクラスタの分析の詳細は、『Oracle Database管理者ガイド』を参照してください

5 時間ベース情報の管理およびメンテナンス

Oracle Databaseには、時間に基づいてデータを管理およびメンテナンスするための計画が用意されています。

この章では、時間に基づくデータの管理およびメンテナンスの戦略を作成できるOracle Databaseのコンポーネントについて説明します。

ほとんどの組織で、保管データは最も貴重な企業資産の1つであると考えられてきましたが、データの管理方法やメンテナンス方法は企業によって大きく異なります。本来、データの使用目的は、経営目標の達成を支援し、事業を経営し、また将来の方向性を特定し、企業を成功に導くことにありました。

ただし、新しい政府規制とガイドラインが、データ保持の方法や理由において大きな促進力となっています。規制のために、組織は非常に長い期間にわたって情報を保持して管理する必要が生じています。その結果、今日では、情報技術(IT)マネージャは、その他の次のような目標を達成しようとしています。

- 可能なかぎり低いコストで大量のデータを格納すること
- データの保持および保護のための新しい法的要件を満たすこと
- データ量の増加に基づく、より詳しい分析により、ビジネス機会を改善すること

この章の構成は、次のとおりです。

- ILMを使用したOracle Databaseのデータの管理
- ヒート・マップおよびADOを使用したILM戦略の実装
- Oracle Databaseのデータの有効性および表示の制御
- パーティション化を使用したILMシステムの手動実装
- Oracle Enterprise ManagerでのILMヒート・マップおよびADOの管理

5.1 ILMを使用したOracle Databaseのデータの管理

情報ライフサイクル管理(ILM)では、Oracle Database内のデータを、そのデータに適用されるルールおよび規則を使用して管理できます。

現在、情報の形式は、電子メール・メッセージから、写真、オンライン・トランザクション処理(OLTP)システムの注文まで多岐にわたります。データの種類と使用方法がわかれば、データの発展方法や最終的な状態を理解したことになります。

各組織が直面する1つの目標は、データに適用されるようになった規則や規制を守る一方で、自らのデータがどのように発展および増大するかを理解し、使用方法が時間につれて変化する様子を監視し、保管すべき期間を決定することです。情報ライフサイクル管理(ILM)は、プロセス、ポリシー、ソフトウェアおよびハードウェアを組み合せて、これらの課題に取り組み、データのライフサイクルの各ステージで適切な技術を使用できるようにします。

この項では、次の項目について説明します。

- ILMのOracle Databaseについて
- Oracle Databaseを使用したILMの実装

5.1.1 ILMのOracle Databaseについて

Oracle Databaseは、ILMソリューションの実装に理想的なプラットフォームを提供します。

Oracle Databaseプラットフォームは、次を提供します。

- アプリケーションの透過性

  アプリケーションの透過性はILMでは大変重要です。アプリケーションをカスタマイズする必要がないことを意味し、様々な変更をデータに加えてもデータを使用するアプリケーションに影響を及ぼさないためです。データをライフサイクルの別のステージに簡単に移動することができ、データベースでのデータ・アクセスを最適化することができます。もう1つの大きな利点は、アプリケーションの透過性により、新たな規制要件に迅速に対応するために必要な柔軟性が得られることです。この場合も、既存のアプリケーションが影響を受けません。

- ファイングレイン・データ

  Oracleでは、データを非常に細かいレベルや関連グループごとに表示することができます。これに対して、ストレージ・デバイスではバイトやブロックが表示されるだけです。

- 低コスト・ストレージ

  保管するデータが非常に多い場合、低コスト・ストレージの使用が、ILMを実装する際の重要な要素になります。Oracleは多様なストレージ・デバイスを利用できるため、最小限のコストで最大限の容量のデータを保持できます。

- 強制可能なコンプライアンス・ポリシー

  コンプライアンスの理由で情報を保存する場合は、規則に従ってデータを保管および管理していることを取締機関に示すことが不可欠です。Oracle Databaseでは、セキュリティ・ポリシーと監査ポリシーを定義し、データのすべてのアクセスに対して施行して、アクセスを記録することができます。

この項では、次の項目について説明します。

- Oracle Databaseでのデータ型の管理
- 規制の要件
- オンライン・アーカイブの利点

5.1.1.1 Oracle Databaseでのデータ型の管理

I情報ライフサイクル管理は、組織のすべてのデータに関連します。

このデータには、OLTPシステムでの注文やデータ・ウェアハウスでの売上履歴など、構造化されたデータの他に、電子メール、ドキュメント、イメージなど、構造化されていないデータも含まれます。Oracle Databaseでは、BLOBおよびOracle SecureFilesによって非構造化データの格納がサポートされ、優れたドキュメント管理システムをOracle Textで使用できます。

組織のすべての情報がOracle Databaseに含まれる場合は、データベースで提供される機能を利用して、存続期間の経過に伴う展開に応じてデータを管理および移動できます。複数の種類のデータ・ストアを管理する必要はありません。

5.1.1.2 規制の要件

現在、多くの組織は、特定のデータを特定の期間にわたって保管する必要があります。このような規制に従わない組織は、非常に重い罰金を支払うことになります。

世界中の様々な規制要件(米国のSarbanes-Oxley、HIPAA、DOD5015.2-STD、欧州連合の欧州データ保護指令(European Data Privacy Directive)など)により、組織のデータ管理方法が変化しています。このような規制によって、保管する必要があるデータ、データを変更できるかどうか、保管する必要がある期間(30年以上に及ぶ可能性もある)が定められます。

これらの規制によって頻繁に要求されるのは、未許可のアクセスや変更から電子データを保護することと、データに対する変更や変更者の監査証跡を保持することです。Oracle Databaseでは、アプリケーションのパフォーマンスに影響せずに大容量のデータを保存できます。また、アクセスの制限や未許可のデータ変更の防止に必要な機能も含み、Oracle Audit Vault and Database Firewallを使用して拡張することができます。Oracle Databaseでは暗号化機能も提供され、これにより、高度な権限を持つユーザーが意図的にデータを変更したのではないことを示すことができます。フラッシュバック・データ・テクノロジを使用すると、改ざん防止履歴アーカイブの存続期間における行のすべてのバージョンを格納できます。

5.1.1.3 オンライン・アーカイブの利点

オンライン・アーカイブには複数の利点があります。

データのライフサイクルのある時点になると、データは定期的にアクセスされなくなり、アーカイブの対象とみなされます。従来、データはデータベースから削除されて大容量の情報を非常に低いコストで格納できるテープに格納されました。現在、データをテープにアーカイブする必要はありません。データベースに残しておくか、集中オンライン・アーカイブ・データベースに移すことができます。このようなすべての情報を、1GB当たりのコストがテープとほとんど変わらない低コスト・ストレージを使用して格納できます。

アーカイブ目的ですべてのデータをOracle Databaseに置いておくと多数の利点が得られます。最も重要な利点は、いつでもすぐにデータを利用できるということです。このため、データがアーカイブされたテープを探したり、テープが読取り可能かどうか、またデータベースにロードできる形式になっているかどうかを判断したりして、時間を無駄にせずにすみます。

データが長い間アーカイブされていた場合は、テープ・アーカイブからデータベースにデータをリロードするためのプログラムを作成する開発時間も必要になります。これにはコストや時間がかかることがわかっており、データが古い場合は特に顕著です。データベースに保存したデータは、オンライン状態で最新のデータベース・フォーマットになっているためこの問題はありません。

現在、データベースに履歴データを保持しても、データベースのバックアップに必要な時間やバックアップのサイズに影響することはありません。RMANを使用してデータベースをバックアップすると、変更されたデータのみがバックアップに取得されます。履歴データが変更される可能性は少ないため、一度データがバックアップされると、その後のバックアップはありません。

検討する必要があるもう1つの重要な要素は、データベースからデータを物理的に除去する方法です(特に本番システムから集中アーカイブに移す場合)。Oracleではトランスポータブル表領域またはパーティションを使用して、データベース間でこのデータを短時間で移動する機能が提供されます。これにより、データがひとまとまりの単位で移動されます。

データベースからデータを削除するにあたり、最も速い方法はデータのセットを削除することです。これは、データがパーティションに保持された状態で実現できます。パーティションの削除は非常に高速で処理できます。ただし、データの関係を維持する必要があるためにこの方法が不可能な場合は、従来のSQL delete文を発行する必要があります。delete文の発行に必要な時間を少なく見積らないようにしてください。

データをデータベースから削除する必要があり、将来、データベースにデータを戻す必要が生じる可能性がある場合は、トランスポータブル表領域などのデータベース・フォーマットのままデータを移動するか、Oracle DatabaseのXML機能を使用してオープン・フォーマットで情報を抽出することを検討してください。

Oracle Databaseでのデータのオンライン・アーカイブを検討する理由は次のとおりです。

- ディスクのコストはテープのコストに近づいているため、データが入っているテープを探す時間やデータをリストアするためのコストをなくすことができます。
- 必要時にデータがオンラインになっており、高速アクセスを提供し、業務要件を満たします。
- データがオンラインになっているためすぐにアクセスできます。このため、データを提供できないことで規制機関から罰金を課せられる可能性が少なくなります。
- 現在のアプリケーションを使用してデータにアクセスできるため、新しいアプリケーションを構築するためにリソースを無駄にする必要がありません。

5.1.2 Oracle Databaseを使用したILMの実装

Oracle Databaseを使用した情報ライフサイクル管理ソリューションの作成は非常に簡単です。

ILMソリューションは、次の4つの簡単なステップによって完了できますが、ILMがコンプライアンス目的で実装されていない場合、ステップ4はオプションです。

- ステップ1: データ・クラスの定義
- ステップ2: データ・クラスに対応したストレージ層の作成
- ステップ3: データのアクセス・ポリシーおよび移行ポリシーの作成
- ステップ4: コンプライアンス・ポリシーの定義と施行

5.1.2.1 ステップ1: データ・クラスの定義

情報ライフサイクル管理を有効に利用するには、まず最初に、組織のすべてのデータを確認してから情報ライフサイクル管理ソリューションを実装します。

データの確認後、次を決定します。

- 重要なデータは何か、どこに格納されているか、保管する必要があるデータは何か
- そのデータの組織でのフロー
- 時間経過につれてそのデータがどのように扱われるか、そのデータはまだ必要か
- データ可用性の程度と必要な保護
- 法的および業務上の要件に対応するデータ保存

データの使用方法を理解すれば、それに基づいてデータを分類できます。最も一般的なタイプの分類は、経過時間つまり日付によるものですが、製品やプライバシなどその他のタイプの分類も可能です。プライバシと経過時間など、混合型の分類も使用できます。

データ・クラスの処理方法を変えるには、データを物理的に分ける必要があります。情報が作成されると最初のうちはよくアクセスされますが、時間が経過するとほとんど参照されなくなることがあります。たとえば、顧客が注文を行うと、ステータスや注文品が発送されたかどうかを確認するために定期的に注文を表示します。注文品が届いた後は、おそらくその注文を参照することはありません。この注文も、注文されている品物を確認するために実行する定期レポートに含まれます。ただし、時間がたつとどのレポートにも含まれなくなり、将来的に参照されるのは、そのデータに関連する詳しい分析が行われる場合のみになります。たとえば、注文を財務四半期Q1、Q2、Q3およびQ4に分け、さらに履歴注文として分類することができます。

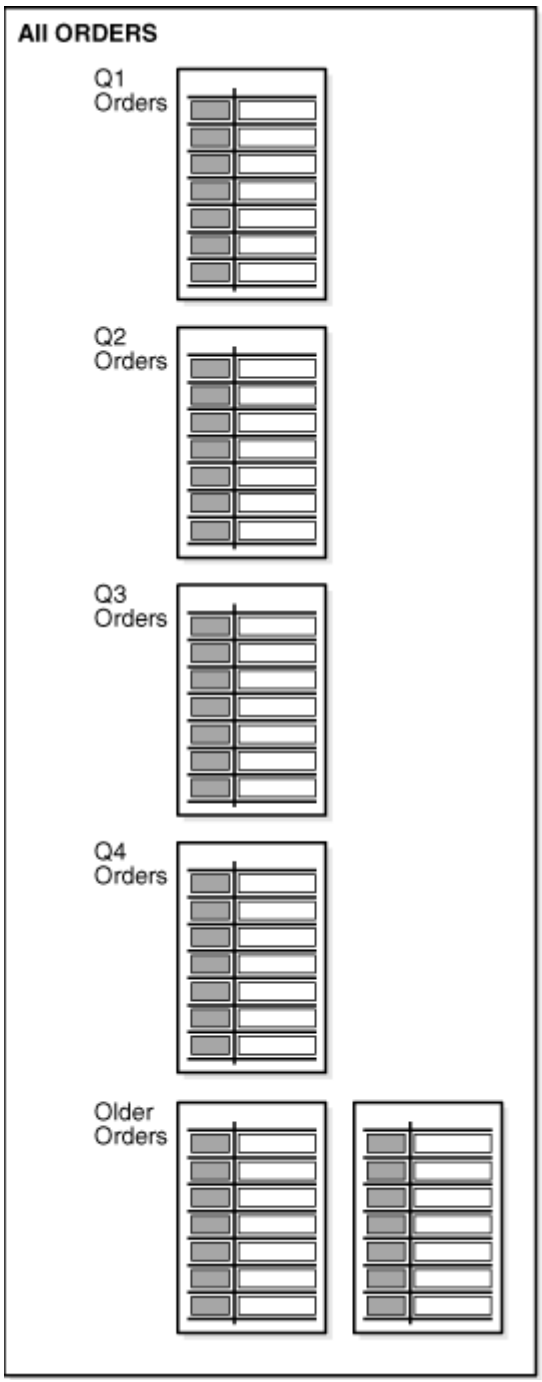
この方法を使用する利点は、データがクラス(この例では注文日付)ごとに行レベルでグループ化されることです。Q2の注文は別のクラスに存在するため、Q1のすべての注文を自己完結した単位として管理できます。これはパーティション化の使用によって実現できます。パーティションはアプリケーションに対して透過的であるため、データを物理的に分離しても、アプリケーションはすべての注文を見つけることができます。

5.1.2.1.1 ILMでのパーティション化

パーティション化では、データ値に基づいてデータを物理的に配置します。日付によってデータをパーティション化する方法がよく使用されます。

図5-1に示すシナリオでは、Q1、Q2、Q3およびQ4の注文を個別のパーティションに格納し、前年までの注文を他のパーティションに格納します。

図5-1 データ・クラスのパーティションへの割当て



Oracleでは複数の異なるパーティション化方法を提供します。レンジ・パーティション化は、ILMのためによく使用されるパーティション化方法の1つです。時間隔パーティション化および参照パーティション化も、ILM環境での使用に特に適しています。

データのパーティション化には様々な利点があります。パーティション化により、データを使用方法に応じて適切なストレージ・デバイスに簡単に分散すると同時に、データをオンラインに保ち、最もコスト効果の高いデバイスに格納できるようになります。パーティション化はデータにアクセスするすべてのユーザーにとって透過的であるため、アプリケーションの変更が必要ありません。このため、いつでもパーティション化を実装できます。新しいパーティションが必要なときには、ADD PARTITION句を使用して追加するだけです。また、時間隔パーティションを使用している場合には、パーティションが自動的に作成されます。

その他の利点として、各パーティションが独自のローカル索引を持つことが可能です。オプティマイザがパーティション・プルーニングを使用すると、問合せは、すべてのパーティションではなく関連するパーティションのみにアクセスするため、問合せのレスポンス時間が短縮されます。

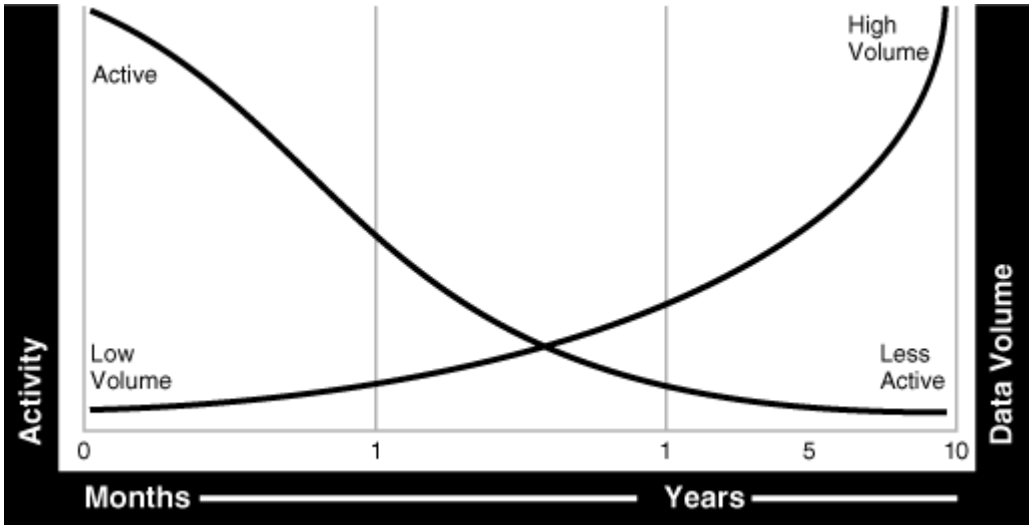
5.1.2.1.2 データのライフサイクル

データを分析すると、多くの場合、当初はアクセスや更新が非常に頻繁に行われることがわかります。データが古くなるにつれて、アクセス頻度は減少し、あるとしてもごく少数になります。

図5-2に示すように、ほとんどの組織では、多くのユーザーが現行データにアクセスするが、それよりも古いデータにアクセスするユーザーはほとんどいないという状況が見られます。データは、アクティブ、非アクティブ、履歴、アーカイブ可能のいずれかとみなすことができます。

非常に多くのデータを保持するときは、その存続期間において物理的に場所を移す必要があります。データがライフサイクルのどの時点にあるかによって異なりますが、最も適切なストレージ・デバイスに格納する必要があります。

図5-2 時間経過に伴うデータ使用状況



5.1.2.2 ステップ2: データ・クラスに対応したストレージ層の作成

Oracle Databaseでは、多様なストレージ・オプションを利用できるため、情報ライフサイクル管理ソリューションの実装における2番目のステップは、必要なストレージ層の設定です。

必要に応じていくつでもストレージ層を作成できますが、最初は次のストレージ層をお薦めします。

- 高パフォーマンス

  高パフォーマンス・ストレージ層には、Q1の注文を保持するパーティションなど、重要で頻繁にアクセスされるデータがすべて格納されます。この層には、高パフォーマンスのストレージ・デバイスの小型で高速なディスクが使用されます。

- 低コスト

  低コスト・ストレージ層には、Q2、Q3、Q4の注文を保持するパーティションなど、それほど頻繁にアクセスされないデータが格納されます。この層は、モジュール式ストレージ・アレイや低コストATAディスクなど、低コストで最大限の記憶域を提供する大容量ディスクを使用して構築されます。

- オンライン・アーカイブ

  オンライン・アーカイブ・ストレージ層には、ほとんどアクセスまたは変更されないすべてのデータが格納されます。このストレージ層は、非常に大きくなり、最大のデータ容量を格納する可能性があります。様々な方法を使用してデータを圧縮できます。ATAドライブのような低コスト・ストレージ・デバイスに格納しても、データをオンラインで利用できます。コストは情報をテープに格納するよりわずかに高いだけで、テープにアーカイブした場合の不便さもありません。オンライン・アーカイブ・ストレージ層を読取り専用に指定すると、データを変更できなくなります。最初にデータベース・バックアップを取得すると、それ以降バックアップする必要はありません。

- オフライン・アーカイブ(オプション)

  オフライン・アーカイブ・ストレージ層はオプションです。データベースからデータを削除する必要がある場合のみ使用され、データはXMLなど別の形式でテープに格納されます。

図5-2は、一定の期間でデータがどのように使用されるかを示しています。この図から、すべての情報を保管するには、すべての

データを保存する複数のストレージ層が必要となり、さらに、それによってストレージの合計コストが大幅に節減されるというメリットがもたらされることがわかります。

ストレージ層を作成すると、「ステップ1： データ・クラスの定義」で指定したデータ・クラスが、パーティションを使用してデータベース内に物理的に実装されます。この方法により、データを使用方法に応じて適切なストレージ・デバイスに簡単に分散し、一方で、データをオンラインに保っていつでも利用できるようにし、最もコスト効果の高いデバイスに格納することができるようになります。

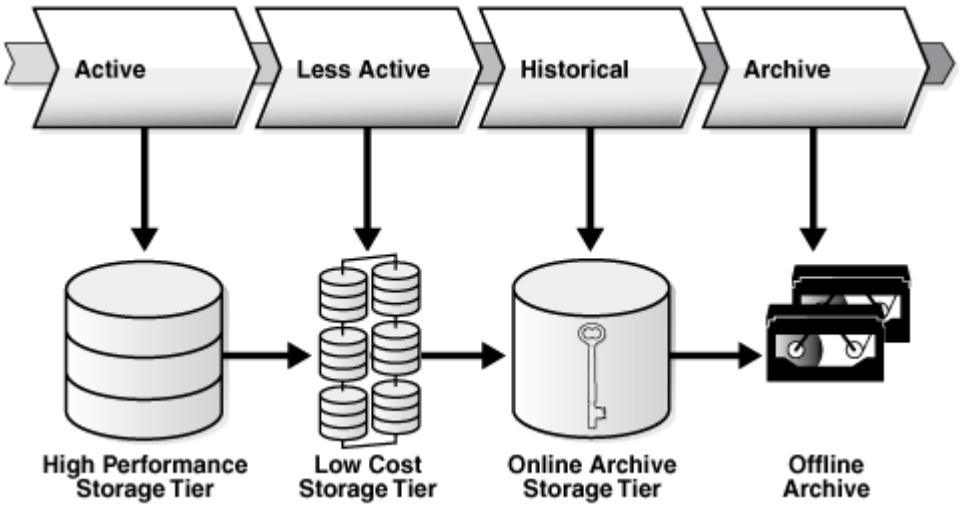
Oracle Automatic Storage Management(Oracle ASM)を使用して、ストレージ層間のデータを管理することもできます。Oracle ASMは、Oracle Databaseファイル用の高パフォーマンスで管理が容易なストレージ・ソリューションです。Oracle ASMはボリューム・マネージャであり、データベースでのみ使用できるよう設計されたファイルシステムが備えられています。Oracle ASMを使用するには、ストライプ化およびミラー化のプリファレンスを使用して、Oracle Databaseのパーティション化したディスクを割り当てます。Oracle ASMによってディスク領域が管理され、使用可能なすべてのリソースにI/O負荷が配分されるため、I/Oを手動でチューニングしなくてもパフォーマンスが最適化されます。たとえば、データベースを停止しなくても、データベースのディスクのサイズを増加し、データベースの一部を新しいデバイスに移動できます。

5.1.2.2.1 ストレージ層へのクラスの割当て

ストレージ層を定義すると、ステップ1で指定したデータ・クラス(パーティション)を適切なストレージ層に割り当てることができます。

この割当てにより、データを使用方法に応じて適切なストレージ・デバイスに簡単に分散し、一方で、データをオンラインに保っていつでも利用できるようにし、最もコスト効果の高いデバイスに格納することができます。図5-3で、アクティブ、非アクティブ、履歴またはアーカイブ可能として識別されるデータは、高パフォーマンス層、低コスト・ストレージ層、オンライン・アーカイブ・ストレージ層およびオフライン・アーカイブにそれぞれ割り当てられます。この方法を使用すると、アプリケーションによってデータが引き続き認識されるのでアプリケーションを変更する必要はありません。

図5-3 データのライフサイクル



5.1.2.2.2 階層ストレージの使用によるコスト節減

ILM戦略を実装する利点の1つは、複数階層ストレージの使用によりコストを節減できることです。

格納するデータが3TBあり、内訳は高パフォーマンス200GB、低コスト800GB、オンライン・アーカイブ2TBであるとします。また、1GB当たりのコストは、高パフォーマンス層では72ドル、低コスト層では14ドル、オンライン・アーカイブ層では7ドルと仮定します。

表5-1に、すべてのデータを1クラスのストレージに格納するかわりに階層ストレージを使用することで実現できるコスト節減を示します。ここでわかるように、非常に大きなコスト節減が可能です。データがOLTPおよびHCCデータベース圧縮に適している場合は、さらにコスト節減を行うことができます。

表5-1 階層ストレージ使用によるコスト節減

ストレージ層	単一層(高パフォーマンス・ディスク使用)	複数ストレージ層	複数層(データベース圧縮)
高パフォーマンス(200 GB)	$14,400	$14,400	$14,400
低コスト(800 GB)	$57,600	$11,200	$11,200
オンライン・アーカイブ(2 TB)	$144,000	$14,000	$5,600
各列の合計	$216,000	$39,600	$31,200

5.1.2.3 ステップ3: データのアクセス・ポリシーおよび移行ポリシーの作成

情報ライフサイクル管理ソリューションの実装における3番目のステップは、許可されたユーザーのみがデータにアクセスできるようにすること、およびデータの存続期間中のデータの移動方法を指定することです。

データが古くなったときにストレージ層の間でデータを移行する方法がいくつかあります。

5.1.2.3.1 データへのアクセス制御

データのアクセス権を存続期間中に変更できるため、データのセキュリティは情報ライフサイクル管理のもう1つの重要な側面です。

さらに、データのアクセス方法について厳密に要求する規制要件が存在する場合があります。

Oracle Database内のデータは、次のようなデータベース機能を使用して保護されます。

- データベース・セキュリティ
- ビュー
- 仮想プライベート・データベース

仮想プライベート・データベース(VPD)では、データベースに対して非常に細かいレベルのアクセスを定義できます。セキュリティ・ポリシーによって、表示できる行と列が決まります。複数のポリシーを定義することで、様々なユーザーやアプリケーションが同じデータの異なるビューを表示できます。たとえば、Q1、Q2、Q3およびQ4の情報は大半のユーザーが見られるようにして、履歴データは許可されたユーザーのみが見られるようにできます。

セキュリティ・ポリシーはデータベース・レベルで定義され、すべてのデータベース・ユーザーに透過的に適用されます。この方法の利点は、データにアクセスするための安全で管理された環境が提供されることです。この環境を変更することはできず、実装するためにアプリケーションを変更する必要もありません。また、データが変更されないことを保証する読取り専用表領域を定義できます。

5.1.2.3.2 パーティション化を使用したデータの移動

存続期間中、データを移動する必要があり、使用できる技法はパーティション化です。

データの移動は次の理由で発生する可能性があります。

- パフォーマンスを維持するため、高パフォーマンス・ディスクに保持できる注文数には制限があります。
- データが頻繁にアクセスされなくなっても、高価な高パフォーマンス・ストレージを使用している場合には、低コスト・ストレージ・デバイスに移動する必要があります。

- 法的要件により、情報は指定の期間にわたって利用可能であることが必要です。最低限のコストで安全に保存する必要があります。

様々なストレージ層を活用するようにOracle Database内でデータを物理的に移動する方法は多数あります。たとえば、データがパーティション化されている場合、Q2の注文を含むパーティションを、高パフォーマンス・ストレージ層から低コスト・ストレージ層にオンラインのまま移動することができます。データはデータベース内で移動されるため、物理的に移動しても、データを必要とするアプリケーションに影響したり、通常のユーザーの作業を中断したりすることはありません。

場合によっては、データのグループではなく、個々のデータ項目を移動する必要があります。たとえば、プライバシとレポートのレベルに従ってデータが分類されており、以前は機密扱いだったデータが、現在は公開できるようになったとします。データがプライバシの分類に基づいてパーティション化されているときに、分類が機密から公開に変更されると、データ行は公開データを含むパーティションに自動的に移動されます。

データを元の場所から移動するときは、選択したプロセスがすべての規制要件に準拠するように常に確認することがきわめて重要です。たとえば、データを変更できない、未許可のアクセスから保護する、容易に参照できる、承認された場所に格納するといった要件があります。

5.1.2.4 ステップ4: コンプライアンス・ポリシーの定義と施行

情報ライフサイクル管理ソリューションの4番目のステップは、コンプライアンスのためのポリシーの作成です。

データが集中管理されず断片化している場合は、すべての場所でコンプライアンス・ポリシーを定義して施行する必要がありますが、コンプライアンス・ポリシーが見逃されやすくなります。ただし、Oracle Databaseを使用して、データを集中的に格納できる場所を提供すると、すべてのコンプライアンス・ポリシーを1箇所で管理および施行でき、コンプライアンス・ポリシーの施行が非常に容易になります。

コンプライアンス・ポリシーを定義するときは、次の点を考慮してください。

- データの保存
- 不変性
- プライバシ
- 監査
- 有効期限

5.1.2.4.1 データの保存

保存ポリシーでは、データの保存方法、保存が必要な期間、最終的な処理方法が指定されます。

例としては、レコードを元の形式で格納する必要があり、変更は許可されず、7年間保存する必要があるが、その後は削除できるというような保存ポリシーがあります。Oracle Databaseセキュリティを使用すると、データを変更せずに保持し、許可されたプロセスのみが適切なときにデータを削除することを保証できます。保存ポリシーは、ILMアシスタントのライフサイクル定義を使用して定義することもできます。

5.1.2.4.2 不変性

不変性は、データが完全であり変更されていないことを外部機関に証明することに関連します。

データが変更されていないことを示すために、暗号署名すなわちデジタル署名をOracle Databaseによって生成して、データベースの内外に保存できます。

5.1.2.4.3 プライバシ

Oracle Databaseではデータ・プライバシを保証するためにいくつかの方法が提供されます。

データへのアクセスは、仮想プライベート・データベース(VPD)を使用して定義したセキュリティ・ポリシーにより厳しく制御できます。また、生データを見るユーザーがデータの内容を把握できないように、個々の列を暗号化することができます。

5.1.2.4.4 監査

Oracle Databaseは、データに対するすべてのアクセスと変更を追跡できます。

これらの監査機能は表レベルで定義することも、ファイングレイン監査を使用して定義することもできます。ファイングレイン監査では、監査レコードをいつ生成するかという基準を指定します。Oracle Audit Vault and Database Firewallを使用して、監査をさらに拡張することができます。

関連項目:

Oracle Audit Vault and Database Firewallの詳細は、『*Oracle Audit Vault and Database Firewall*管理者ガイド』を参照してください。

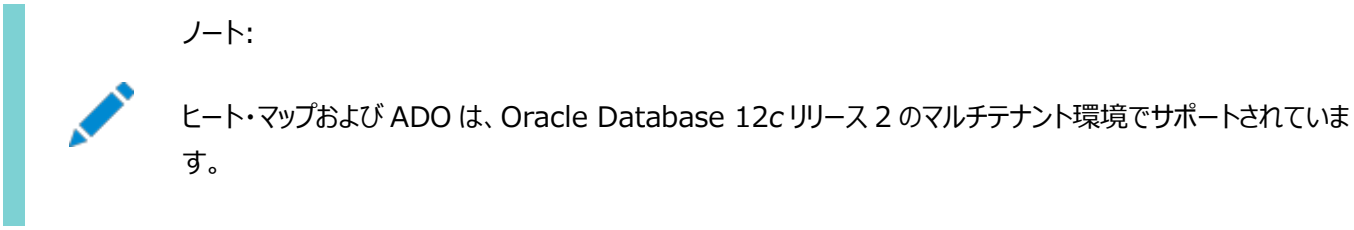
5.1.2.4.5 有効期限

最終的には、業務または規制に関する理由からデータの有効期間が終わり、データベースから削除する必要が生じます。

Oracle Databaseでは、削除対象として指定された情報を含むパーティションを削除することにより、データをすばやく効率よく削除できます。

5.2 ヒート・マップおよびADOを使用したILM戦略の実装

データベースでのデータ移動のために情報ライフサイクル管理(ILM)計画を実装するには、ヒート・マップおよび自動データ最適化(ADO)の機能を使用します。



ノート:

ヒート・マップおよび ADO は、Oracle Database 12c リリース 2 のマルチテナント環境でサポートされています。

この項では、次の項目について説明します。

- ヒート・マップの使用
- 自動データ最適化の使用
- ADOおよびヒート・マップの制限事項

関連項目:

- ヒート・マップおよびADOを使用するOracle Enterprise Manager Cloud Controlの使用の詳細は、「Oracle Enterprise ManagerでのILMヒート・マップおよびADOの管理」を参照してください
- Database Vaultで保護されたオブジェクト上でのILM操作をADO管理ユーザーが実行できるようにする認可の付与など、Oracle Database Vaultレルムおよびコマンド・ルールによる情報ライフサイクル管理(ILM)の使用の詳細は、Oracle Database Vault管理者ガイドを参照してください。

5.2.1 ヒート・マップの使用

ILM戦略を実装するために、Oracle Databaseのヒート・マップを使用すればデータのアクセスと変更を追跡できます。

ヒート・マップは、セグメント・レベルのデータ・アクセス・トラッキングおよびセグメントおよび行レベルのデータ変更トラッキングを提供します。HEAT_MAP初期化パラメータを使用して、この機能を有効にできます。

ヒート・マップ・データは、ADOポリシーを使用してインメモリー列ストア(IM列ストア)のコンテンツを管理するための、自動データ最適化(ADO)を支援します。列統計およびその他の関連統計が含まれるヒート・マップ・データを使用すると、IM列ストアがいつ満杯(メモリー不足)になるかを判断できます。ほぼ満杯と判断されたときに、アクセス頻度が高くIM列ストアに移入することで利点が得られるセグメントが存在する場合は、非アクティブなセグメントが削除されます。

この項では、次の項目について説明します。

- ヒート・マップの有効化および無効化
- ビューを使用したヒート・マップ・トラッキング・データの表示
- DBMS_HEAT_MAPサブプログラムを使用したヒート・マップ・データの管理

関連項目:

- インメモリー列ストアの有効化およびサイズ設定の詳細は、『Oracle Database In-Memoryガイド』を参照してください

5.2.1.1 ヒート・マップの有効化および無効化

HEAT_MAP句を含むALTER SYSTEMまたはALTER SESSION文を使用して、システムまたはセッション・レベルでヒート・マップ・トラッキングを有効化および無効化できます。

たとえば、次のSQL文は、データベース・インスタンスのヒート・マップ・トラッキングを有効化します。

```
ALTER SYSTEM SET HEAT_MAP = ON;
```

ヒート・マップを有効にすると、すべてのアクセスがインメモリー・アクティビティ・トラッキング・モジュールによって追跡されます。SYSTEMおよびSYSAUX表領域のオブジェクトは追跡されません。

次のSQL文は、ヒート・マップ・トラッキングを無効化します。

```
ALTER SYSTEM SET HEAT_MAP = OFF;
```

ヒート・マップを無効にすると、アクセスがインメモリー・アクティビティ・トラッキング・モジュールによって追跡されません。HEAT_MAP初期化パラメータのデフォルト値はOFFです。

HEAT_MAP初期化パラメータでも自動データ最適化(ADO)を有効化および無効化できます。ADOの場合、システム・レベルでヒート・マップを有効にする必要があります。

関連項目:

- ADOの詳細は、「自動データ最適化の使用」を参照してください
- HEAT_MAP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

5.2.1.2 ビューを使用したヒート・マップ・トラッキング・データの表示

ヒート・マップ・トラッキング・データは、V$*、ALL*、DBA*およびUSER*ヒート・マップ・ビューで表示されます。

例5-1に、ヒート・マップ・ビューで提供される情報の例を示します。V$HEAT_MAP_SEGMENTビューは、リアルタイム・セグメント・アクセス情報を表示します。ALL_、DBA_およびUSER_HEAT_MAP_SEGMENTビューは、ユーザーに表示されるすべてのセグメントの最新のセグメント・アクセス時間を表示します。ALL_、DBA_およびUSER_HEAT_MAP_SEG_HISTOGRAMビューは、ユーザーに表示されるすべてのセグメントのセグメント・アクセス情報を表示します。DBA_HEATMAP_TOP_OBJECTSビューは、最もアクティブなオブジェクトのヒート・マップ情報を表示します。DBA_HEATMAP_TOP_TABLESPACESビューは、最もアクティブな表領域のヒート・マップ情報を表示します。

関連項目:

ヒート・マップ・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5-1 ヒート・マップ・ビュー

```
/* enable heat map tracking if necessary*/

SELECT SUBSTR(OBJECT_NAME,1,20), SUBSTR(SUBOBJECT_NAME,1,20), TRACK_TIME, SEGMENT_WRITE,
  FULL_SCAN, LOOKUP_SCAN FROM V$HEAT_MAP_SEGMENT;

SUBSTR(OBJECT_NAME,1 SUBSTR(SUBOBJECT_NAM TRACK_TIM SEG FUL LOO
-------------------- -------------------- --------- --- --- ---
SALES                SALES_Q1_1998        01-NOV-12 NO  NO  NO
SALES                SALES_Q3_1998        01-NOV-12 NO  NO  NO
SALES                SALES_Q2_2000        01-NOV-12 NO  NO  NO
SALES                SALES_Q3_1999        01-NOV-12 NO  NO  NO
SALES                SALES_Q2_1998        01-NOV-12 NO  NO  NO
SALES                SALES_Q2_1999        01-NOV-12 NO  NO  NO
SALES                SALES_Q4_2001        01-NOV-12 NO  NO  NO
SALES                SALES_Q1_1999        01-NOV-12 NO  NO  NO
SALES                SALES_Q4_1998        01-NOV-12 NO  NO  NO
SALES                SALES_Q1_2000        01-NOV-12 NO  NO  NO
SALES                SALES_Q1_2001        01-NOV-12 NO  NO  NO
SALES                SALES_Q2_2001        01-NOV-12 NO  NO  NO
SALES                SALES_Q3_2000        01-NOV-12 NO  NO  NO
SALES                SALES_Q4_2000        01-NOV-12 NO  NO  NO
EMPLOYEES                                 01-NOV-12 NO  NO  NO
...

SELECT SUBSTR(OBJECT_NAME,1,20), SUBSTR(SUBOBJECT_NAME,1,20), SEGMENT_WRITE_TIME,
  SEGMENT_READ_TIME, FULL_SCAN, LOOKUP_SCAN FROM USER_HEAT_MAP_SEGMENT;

SUBSTR(OBJECT_NAME,1 SUBSTR(SUBOBJECT_NAM SEGMENT_W SEGMENT_R FULL_SCAN LOOKUP_SC
-------------------- -------------------- --------- --------- --------- ---------
SALES                SALES_Q1_1998                              30-OCT-12 01-NOV-12
SALES                SALES_Q1_1998                              30-OCT-12 01-NOV-12
SALES                SALES_Q1_1998                              30-OCT-12 01-NOV-12
SALES                SALES_Q1_1998                              30-OCT-12 01-NOV-12
SALES                SALES_Q1_1998                              30-OCT-12 01-NOV-12
SALES                SALES_Q1_1998                              30-OCT-12 01-NOV-12
...

SELECT SUBSTR(OBJECT_NAME,1,20), SUBSTR(SUBOBJECT_NAME,1,20), TRACK_TIME, SEGMENT_WRITE, FULL_SCAN,
  LOOKUP_SCAN FROM USER_HEAT_MAP_SEG_HISTOGRAM;

SUBSTR(OBJECT_NAME,1 SUBSTR(SUBOBJECT_NAM TRACK_TIM SEG FUL LOO
-------------------- -------------------- --------- --- --- ---
SALES                SALES_Q1_1998        31-OCT-12 NO  NO  YES
SALES                SALES_Q1_1998        01-NOV-12 NO  NO  YES
SALES                SALES_Q1_1998        30-OCT-12 NO  YES YES
SALES                SALES_Q2_1998        01-NOV-12 NO  NO  YES
SALES                SALES_Q2_1998        31-OCT-12 NO  NO  YES
SALES                SALES_Q2_1998        30-OCT-12 NO  YES YES
SALES                SALES_Q3_1998        01-NOV-12 NO  NO  YES
SALES                SALES_Q3_1998        30-OCT-12 NO  YES YES
SALES                SALES_Q3_1998        31-OCT-12 NO  NO  YES
SALES                SALES_Q4_1998        01-NOV-12 NO  NO  YES
SALES                SALES_Q4_1998        31-OCT-12 NO  NO  YES
SALES                SALES_Q4_1998        30-OCT-12 NO  YES YES
SALES                SALES_Q1_1999        01-NOV-12 NO  NO  YES
SALES                SALES_Q1_1999        31-OCT-12 NO  NO  YES
...

SELECT SUBSTR(OWNER,1,20), SUBSTR(OBJECT_NAME,1,20), OBJECT_TYPE, SUBSTR(TABLESPACE_NAME,1,20),
    SEGMENT_COUNT FROM DBA_HEATMAP_TOP_OBJECTS ORDER BY SEGMENT_COUNT DESC;
```

```
SUBSTR(OWNER,1,20)   SUBSTR(OBJECT_NAME,1 OBJECT_TYPE         SUBSTR(TABLESPACE_NA SEGMENT_COUNT
-------------------- -------------------- ------------------ -------------------- -------------
SH                   SALES                TABLE              EXAMPLE                       96
SH                   COSTS                TABLE              EXAMPLE                       48
PM                   ONLINE_MEDIA         TABLE              EXAMPLE                       22
OE                   PURCHASEORDER        TABLE              EXAMPLE                       18
PM                   PRINT_MEDIA          TABLE              EXAMPLE                       15
OE                   CUSTOMERS            TABLE              EXAMPLE                       10
OE                   WAREHOUSES           TABLE              EXAMPLE                        9
HR                   EMPLOYEES            TABLE              EXAMPLE                        7
OE                   LINEITEM_TABLE       TABLE              EXAMPLE                        6
IX                   STREAMS_QUEUE_TABLE  TABLE              EXAMPLE                        6
SH                   FWEEK_PSCAT_SALES_MV TABLE              EXAMPLE                        5
SH                   CUSTOMERS            TABLE              EXAMPLE                        5
HR                   LOCATIONS            TABLE              EXAMPLE                        5
HR                   JOB_HISTORY          TABLE              EXAMPLE                        5
SH                   PRODUCTS             TABLE              EXAMPLE                        5
...

SELECT SUBSTR(TABLESPACE_NAME,1,20), SEGMENT_COUNT
    FROM DBA_HEATMAP_TOP_TABLESPACES ORDER BY SEGMENT_COUNT DESC;

SUBSTR(TABLESPACE_NA SEGMENT_COUNT
-------------------- -------------
EXAMPLE                        351
USERS                           11

SELECT COUNT(*) FROM DBA_HEATMAP_TOP_OBJECTS;

  COUNT(*)
----------
        64

SELECT COUNT(*) FROM DBA_HEATMAP_TOP_TABLESPACES;

  COUNT(*)
----------
         2
```

5.2.1.3 DBMS_HEAT_MAPサブプログラムを使用したヒート・マップ・データの管理

DBMS_HEAT_MAPパッケージは、DBMS_HEAT_MAPサブプログラムを使用したヒート・マップ・データの表示にさらに柔軟性を提供します。

DBMS_HEAT_MAPには、ブロック、エクステント、セグメント、オブジェクト、表領域などの様々なレベルの記憶域でヒート・マップを外部化する1番目のAPIのセットおよび上位の表領域のバックグラウンド・プロセスでマテリアライズド化されたヒート・マップを外部化する2番目のAPIのセットが含まれます。

例5-2に、DBMS_HEAT_MAPパッケージのサブプログラムの使用の例を示します。

関連項目:

DBMS_HEAT_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

例5-2 DBMS_HEAT_MAPパッケージのサブプログラムの使用

```
SELECT SUBSTR(segment_name,1,10) Segment, min_writetime, min_ftstime
  FROM TABLE(DBMS_HEAT_MAP.OBJECT_HEAT_MAP('SH','SALES'));

SELECT SUBSTR(tablespace_name,1,16) Tblspace, min_writetime, min_ftstime
  FROM  TABLE(DBMS_HEAT_MAP.TABLESPACE_HEAT_MAP('EXAMPLE'));

SELECT relative_fno, block_id, blocks, TO_CHAR(min_writetime, 'mm-dd-yy hh-mi-ss') Mintime,
   TO_CHAR(max_writetime, 'mm-dd-yy hh-mi-ss') Maxtime,
   TO_CHAR(avg_writetime, 'mm-dd-yy hh-mi-ss') Avgtime
   FROM TABLE(DBMS_HEAT_MAP.EXTENT_HEAT_MAP('SH','SALES')) WHERE ROWNUM < 10;

SELECT SUBSTR(owner,1,10) Owner, SUBSTR(segment_name,1,10) Segment,
  SUBSTR(partition_name,1,16) Partition, SUBSTR(tablespace_name,1,16) Tblspace,
  segment_type, segment_size FROM TABLE(DBMS_HEAT_MAP.OBJECT_HEAT_MAP('SH','SALES'));

OWNER      SEGMENT    PARTITION        TBLSPACE         SEGMENT_TYPE          SEGMENT_SIZE
---------- ---------- ---------------- ---------------- -------------------- ------------
SH         SALES      SALES_Q1_1998    EXAMPLE          TABLE PARTITION           8388608
SH         SALES      SALES_Q2_1998    EXAMPLE          TABLE PARTITION           8388608
SH         SALES      SALES_Q3_1998    EXAMPLE          TABLE PARTITION           8388608
SH         SALES      SALES_Q4_1998    EXAMPLE          TABLE PARTITION           8388608
SH         SALES      SALES_Q1_1999    EXAMPLE          TABLE PARTITION           8388608
...
```

5.2.2 自動データ最適化の使用

ILM戦略を実装するには、自動データ最適化(ADO)を使用して、データベース内の異なる層のストレージ間のデータの圧縮および移動を自動化できます。

この機能には、各層の異なる圧縮レベルを指定するポリシーを作成する機能およびデータの移動が発生する時間を制御する機能が含まれます。

この項では、次の項目について説明します。

- 自動データ最適化のポリシーの管理
- ILM ADOポリシーを含む表の作成
- ILM ADOポリシーの追加
- ILM ADOポリシーの無効化と削除
- ADOを使用したセグメント・レベルの圧縮層およびストレージ層の指定
- ADOを使用した行レベルの圧縮層の指定
- ILM ADOパラメータの管理
- ポリシー管理のPL/SQL関数の使用
- ビューを使用したADOのポリシーの監視

自動データ最適化を使用するには、システム・レベルでヒート・マップを有効にする必要があります。HEAT_MAP初期化パラメータを使用して、この機能を有効にします。HEAT_MAP初期化パラメータの設定の詳細は、ヒート・マップの有効化および無効化を参照してください。

5.2.2.1 自動データ最適化のポリシーの管理

SQL文を使用して表を作成および変更する場合、行、セグメントおよび表領域の粒度レベルでADOのポリシーを指定できます。また、ADOポリシーは索引に対してアクションを実行できます。

ADOのポリシーを指定すると、データベース内の異なる層のストレージ間のデータの移動を自動化できます。これらのポリシーにより、各層に異なる圧縮レベルを指定し、データの移動が発生する時間を制御することもできます。

表のADOポリシー

SQL CREATEおよびALTER TABLE文のILM句により、ADOのポリシーを作成、削除、有効化または無効化できます。ILMポリシー句は、圧縮およびストレージ層ポリシーを決定します。また、ポリシー・アクションを起こす条件を指定するAFTER句やON句など、その他の句を含みます。表を作成する場合、ADOの新しいポリシーを追加できます。表を変更してポリシーを追加したり、既存のポリシーを有効化、無効化または削除できます。ポリシーは、表全体または表のパーティションに追加できます。ADOポリシーを表、または表のパーティションに追加する場合、AFTER句には1つの条件タイプのみを指定できます。ILM ADOポリシーには、P1、P2および*Pn*などのシステム生成名が付けられます。

セグメント・レベルのポリシーは1回のみ実行されます。ポリシーが正常に実行されると、そのポリシーは無効になり、再度評価されることはありません。ただし、ポリシーを明示的に再び有効化することはできます。行レベルのポリシーは継続的に実行され、正常な実行後も無効になることはありません。

ADOポリシーの有効範囲は、キーワードGROUP、ROWまたはSEGMENTを使用して、関連オブジェクトのグループに対して、またはセグメントや行のレベルで指定できます。

ポリシーのグループ化に適用できる圧縮のデフォルト・マッピングは次のとおりです。

- ヒープ表のCOMPRESS ADVANCEDは、索引に対して標準の圧縮およびLOBセグメントに対してLOWにマップされます。
- ヒープ表のCOMPRESS FOR QUERY LOW/QUERY HIGHは、索引に対して標準の圧縮およびLOBセグメントに対してMEDIUMにマップされます。
- ヒープ表のCOMPRESS FOR ARCHIVE LOW/ARCHIVE HIGHは、索引に対して標準の圧縮およびLOBセグメントに対してHIGHにマップされます。

圧縮マッピングは変更できません。GROUPは、セグメント・レベルのポリシーにのみ適用できます。ストレージ層ポリシーはセグメント・レベルでのみ適用可能で、行レベルで指定できません。

インメモリー列ストアのADOポリシー

自動データ最適化(ADO)では、INMEMORY、INMEMORY MECOMPRESS、NO INMEMORYポリシー・タイプを使用してインメモリー列ストア(IM列ストア)がサポートされています。

- オブジェクトをインメモリー列ストアに移入できるようにするには、ADD POLICY句にINMEMORYを含めます。
- IM列ストア内のオブジェクトに対する圧縮レベルを上げるには、ADD POLICY句にINMEMORY MEMCOMPRESSを含めます。
- 利点が最も少ないオブジェクトをIM列ストアから明示的に削除するには、ADD POLICY句にNO INMEMORYを含めます。
  例:

NO INMEMORY句を使用してIM列ストアからオブジェクトを削除する例を次に示します。

```
ALTER TABLE sales_2015 ILM ADD POLICY NO INMEMORY
      AFTER 7 DAYS OF NO ACCESS;
```

インメモリー列ストア句を使用するADOポリシーは、セグメント・レベルのポリシーのみとなります。
USER/DBA_ILMDATAMOVEMENTPOLICIESビューおよびV$HEAT_MAP_SEGMENTビューには、インメモリー列ストアについてADOポリシーの情報が含まれています。

ADOポリシーのカスタマイズ

ポリシーを実行する時間を決定する機能を提供するON PL/SQL_functionオプションを使用して、ポリシーをカスタマイズできます。ON PL/SQL_functionオプションは、セグメント・レベルのポリシーでのみ使用できます。例:

```
CREATE OR REPLACE FUNCTION my_custom_ado_rules (objn IN NUMBER) RETURN BOOLEAN;

ALTER TABLE sales_custom ILM ADD POLICY COMPRESS ADVANCED SEGMENT
      ON my_custom_ado_rules;
```

関連項目:

- インメモリー列ストアおよびADOのサポートの詳細は、『Oracle Database In-Memoryガイド』を参照してください
- SQL文のILM句の構文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

5.2.2.2 ILM ADOポリシーを含む表の作成

ILM ADOポリシーを含む表を作成するには、CREATE TABLE文でILM ADD POLICY句を使用します。

例5-3のSQL文は、表を作成し、ILMポリシーを追加します。

例5-3 ILM ADOポリシーを含む表の作成

```
/* Create an example table with an ILM ADO policy */
CREATE TABLE sales_ado
 (PROD_ID NUMBER NOT NULL,
  CUST_ID NUMBER NOT NULL,
  TIME_ID DATE NOT NULL,
  CHANNEL_ID NUMBER NOT NULL,
  PROMO_ID NUMBER NOT NULL,
  QUANTITY_SOLD NUMBER(10,2) NOT NULL,
  AMOUNT_SOLD NUMBER(10,2) NOT NULL )
 PARTITION BY RANGE (time_id)
 ( PARTITION sales_q1_2012 VALUES LESS THAN (TO_DATE('01-APR-2012','dd-MON-yyyy')),
   PARTITION sales_q2_2012 VALUES LESS THAN (TO_DATE('01-JUL-2012','dd-MON-yyyy')),
   PARTITION sales_q3_2012 VALUES LESS THAN (TO_DATE('01-OCT-2012','dd-MON-yyyy')),
   PARTITION sales_q4_2012 VALUES LESS THAN (TO_DATE('01-JAN-2013','dd-MON-yyyy')) )
  ILM ADD POLICY COMPRESS FOR ARCHIVE HIGH SEGMENT
      AFTER 12 MONTHS OF NO ACCESS;

/* View the existing ILM ADO polices */
SELECT SUBSTR(policy_name,1,24) POLICY_NAME, policy_type, enabled
       FROM USER_ILMPOLICIES;

POLICY_NAME              POLICY_TYPE   ENABLE
------------------------ ------------- ------
P1                       DATA MOVEMENT YES
```

5.2.2.3 ILM ADOポリシーの追加

ILM ADOポリシーを表に追加するには、ALTER TABLE文でILM ADD POLICY句を使用します。

例5-4のSQL文では、sales表のパーティションへのILMポリシーの追加の例を示します。

例5-4 ILM ADOポリシーの追加

```
/* Add a row-level compression policy after 30 days of no modifications */
ALTER TABLE sales MODIFY PARTITION sales_q1_2002
  ILM ADD POLICY ROW STORE COMPRESS ADVANCED ROW
  AFTER 30 DAYS OF NO MODIFICATION;

/* Add a segment level compression policy for data after 6 months of no modifications */
ALTER TABLE sales MODIFY PARTITION sales_q1_2001
  ILM ADD POLICY COMPRESS FOR ARCHIVE HIGH SEGMENT
  AFTER 6 MONTHS OF NO MODIFICATION;

/* Add a segment level compression policy for data after 12 months of no access */
ALTER TABLE sales MODIFY PARTITION sales_q1_2000
      ILM ADD POLICY COMPRESS FOR ARCHIVE HIGH SEGMENT
      AFTER 12 MONTHS OF NO ACCESS;

/* Add storage tier policy to move old data to a different tablespace */
/* that is on low cost storage media */
ALTER TABLE sales MODIFY PARTITION sales_q1_1999
  ILM ADD POLICY
  TIER TO my_low_cost_sales_tablespace;

/* View the existing polices */
SELECT SUBSTR(policy_name,1,24) POLICY_NAME, policy_type, enabled
   FROM USER_ILMPOLICIES;

POLICY_NAME              POLICY_TYPE   ENABLE
------------------------ ------------- ------
P1                       DATA MOVEMENT YES
P2                       DATA MOVEMENT YES
P3                       DATA MOVEMENT YES
P4                       DATA MOVEMENT YES
P5                       DATA MOVEMENT YES
```

5.2.2.4 ILM ADOポリシーの無効化と削除

ILM ADOポリシーを無効化または削除するには、ALTER TABLE文でILM DISABLE POLICYまたはILM DELETE POLICY句を使用します。

例5-5のSQL文に示すように、ADOのILMポリシーを無効化または削除できます。既存のILMポリシーが追加する新しいポリシーと競合する場合、既存のポリシーの削除が必要になることがあります。

例5-5 ILM ADOポリシーの無効化と削除

```
/* You can disable or delete an ADO policy in a table with the following */
ALTER TABLE sales_ado ILM DISABLE POLICY P1;
ALTER TABLE sales_ado ILM DELETE POLICY P1;

/* You can disable or delete all ADO policies in a table with the following */
ALTER TABLE sales_ado ILM DISABLE_ALL;
ALTER TABLE sales_ado ILM DELETE_ALL;

/* You can disable or delete an ADO policy in a partition with the following */
ALTER TABLE sales MODIFY PARTITION sales_q1_2002 ILM DISABLE POLICY P2;
ALTER TABLE sales MODIFY PARTITION sales_q1_2002 ILM DELETE POLICY P2;

/* You can disable or delete all ADO policies in a partition with the following */
ALTER TABLE sales MODIFY PARTITION sales_q1_2000 ILM DISABLE_all;
ALTER TABLE sales MODIFY PARTITION sales_q1_2000 ILM DELETE_ALL;
```

5.2.2.5 ADOを使用したセグメント・レベルの圧縮層およびストレージ層の指定

セグメント・レベルの圧縮層ポリシーを使用して、表内のセグメント・レベルで圧縮を指定できます。

行レベルの圧縮層ポリシーと組み合せて、データベースのデータの格納および管理方法を非常に細かく制御します。

例5-6に、ADOのポリシーを作成してsales_ado表の圧縮およびストレージ層ポリシーを施行する方法を示し、次のビジネス要件を反映します。

1. バルク・ロード・データ
2. OLTPワークロードの実行
3. 更新が6か月間なければ、ARCHIVE HIGHで圧縮
4. 低コスト・ストレージへの移動

例5-6 セグメント・レベルの圧縮層およびストレージ層の使用

```
/* Add a segment level compression policy after 6 months of no changes */
ALTER TABLE sales_ado ILM ADD POLICY
  COMPRESS FOR ARCHIVE HIGH SEGMENT
  AFTER 6 MONTHS OF NO MODIFICATION;

Table altered.

/* Add storage tier policy */
ALTER TABLE sales_ado ILM ADD POLICY
  TIER TO my_low_cost_tablespace;

SELECT SUBSTR(policy_name,1,24) POLICY_NAME, policy_type, enabled
  FROM USER_ILMPOLICIES;

POLICY_NAME              POLICY_TYPE   ENABLED
------------------------ ------------- -------
...
P6                       DATA MOVEMENT  YES
P7                       DATA MOVEMENT  YES
```

5.2.2.6 ADOを使用した行レベルの圧縮層の指定

自動データ最適化(ADO)のポリシーでは、基本および拡張圧縮に加えて、ハイブリッド列圧縮(HCC)がサポートされています。

HCC行レベル・ポリシーは、表の圧縮タイプに関係なく、任意の表で定義できます。セグメントの他の部分にDMLアクティビティがある場合は、コールド・ブロックの行をHCCで圧縮できます。

非HCC表のHCCポリシーでは、行がHCC圧縮単位(CU)にある場合は、更新中に行が移動される可能性があります。また、行移動のその他の使用事例と同様に、移動された行を参照している索引エントリを更新するには、索引メンテナンスが必要となります。

行レベル・ポリシーは、Oracle Database 12*c*リリース1 (12.1)でサポートされています。ただし、HCC行レベル圧縮ポリシーを使用するには、データベースが12.2互換であるか、それ以上である必要があります。

関連項目:

表の圧縮の詳細は、『Oracle Database管理者ガイド』を参照してください

例5-7 行レベルのハイブリッド列圧縮の使用によるADOポリシーの作成

例5-7のSQL文では、HCCを使用して表employees_ilmの行でポリシーが作成されます。

```
ALTER TABLE employees_ilm
   ILM ADD POLICY COLUMN STORE COMPRESS FOR QUERY ROW
   AFTER 30 DAYS OF NO MODIFICATION;
```

例5-8 行レベルの拡張圧縮の使用によるADOポリシーの作成

例5-8のSQL文では、拡張圧縮を使用して表sales_adoの行でポリシーが作成されます。

```
ALTER TABLE sales_ado
  ILM ADD POLICY ROW STORE COMPRESS ADVANCED ROW
  AFTER 60 DAYS OF NO MODIFICATION;

SELECT policy_name, policy_type, enabled
   FROM USER_ILMPOLICIES;

POLICY_NAME              POLICY_TYPE   ENABLE
------------------------ ------------- -------

...
P8                       DATA MOVEMENT YES
```

5.2.2.7 ILM ADOパラメータの管理

DBMS_ILM_ADMIN PL/SQLパッケージでCUSTOMIZE_ILMプロシージャを使用して設定する、ILM ADOパラメータを使用して、ADO環境をカスタマイズできます。

様々なILM ADOパラメータを表5-2に示します。

表5-2 ILM ADOパラメータ

名前	説明
ABSOLUTEJOB LIMIT	ABSOLUTEJOB LIMIT の値は、同時に存在する ADO ジョブの絶対数を制限します。
DEGREEOF PARALLELISM	DEGREEOF PARALLELISM の値は、ADO ポリシーのジョブが実行される並列度を決定します。
ENABLED	ENABLED パラメータは、ADO バックグラウンド評価および実行を制御します。デフォルトでは有効化されています(TRUE または 1)。 ENABLED および HEAT_MAP 初期化パラメータの設定は、次のように相互作用します。 ● HEAT_MAP 初期化パラメータが ON に設定され、ENABLED パラメータが FALSE に設定されている場合(0)、ヒート・マップ統計は収集されますが、ADO は統計に対して自動的には機能しません。 ● HEAT_MAP 初期化パラメータが OFF に設定され、ENABLED パラメータが TRUE に設定されている場合(1)、ヒート・マップ統計は収集されず、ADO はヒート・マップ統計を信頼できないため、ADO は何も動作しません。ADO は、ENABLED が FALSE に設定されているかのように動作します。

名前	説明
EXECUTION MODE	EXECUTION MODE の値は、ADO がオンライン・モードまたはオフライン・モードのどちらで実行されるかを制御します。デフォルトはオンライン(2)です。
EXECUTION INTERVAL	EXECUTION INTERVAL のタイプは、ADO がバックグラウンド評価を開始する頻度を決定します。デフォルトは 15 分です。
JOB LIMIT	JOB LIMIT の値は、任意の時点での ADO ジョブの最大数を制御します。同時 ADO ジョブの最大数は、(JOB LIMIT)*(インスタンス数)*(インスタンスごとの CPU 数)として計算されます。デフォルト値は 2 です。
POLICY TIME	POLICY TIME の値は、ADO ポリシーが秒単位または日単位で指定されるかどうかを決定します。値は、秒の場合は 1、または日(デフォルト)の場合は 0 となります。
RETENTION TIME	RETENTION TIME の値は、そのデータがパージされるまで、完了した ADO タスクが保持される時間の長さを指定します。デフォルトは 30 日です。
TBS PERCENT USED	TBS_PERCENT_USED パラメータの値は、表領域が一杯と見なされる場合の表領域割当て率を指定します。デフォルトは 85 パーセントです。
TBS PERCENT FREE	TBS_PERCENT_FREE パラメータの値は、表領域のターゲット空き領域比率を示します。デフォルトは 25 パーセントです。

TBS_PERCENT*パラメータの値のために、ADOは最善を尽くしますが、保障はされません。表領域割当て率がTBS_PERCENT_USEDの値に到達すると、表領域割当ての空き領域比率がTBS_PERCENT_FREEの値に近づくように、ADOはデータの移動を開始します。例として、TBS_PERCENT_USEDが85およびTBS_PERCENT_FREEが25に設定されると、表領域が90パーセント・フルになると想定します。すると、ADOは表領域割当ての少なくとも25パーセントが空き領域になるように(75パーセント未満が表領域割当てに使用されるようにとも言えます)、データを移動する操作を開始します。

DBA_ILMPARAMETERSビューではパラメータを表示できます。たとえば、次の問合せでは、ADO関連パラメータの値を表示します。

```
SQL> SELECT NAME, VALUE FROM DBA_ILMPARAMETERS;

------------------------------------------------------------ ----------
ENABLED                                                               1
RETENTION TIME                                                       30
JOB LIMIT                                                             2
EXECUTION MODE                                                        2
EXECUTION INTERVAL                                                   15
TBS PERCENT USED                                                     85
TBS PERCENT FREE                                                     25
POLICY TIME                                                           0
ABSOLUTE JOB LIMIT                                                   10
DEGREE OF PARALLELISM                                                 4
...
```

関連項目:

- DBMS_ILM_ADMIN PL/SQLパッケージのCUSTOMIZE_ILMプロシージャでILM ADOパラメータを設定する方法を示す例は、例5-9を参照してください
- Oracle Enterprise Manager Cloud ControlによるILM ADOパラメータの設定の詳細は、「Oracle Enterprise ManagerでのILMヒート・マップおよびADOの管理」を参照してください
- ILM ADOパラメータの完全なリストは、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください
- DBMS_ILM_ADMINパッケージの詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください

5.2.2.8 ポリシー管理のPL/SQL関数の使用

より複雑なADOシナリオを実装してポリシーがデータをアクティブに移動および圧縮する時間を制御する高度なポリシー管理およびカスタマイズには、PL/SQL DBMS_ILMおよびDBMS_ILM_ADMINパッケージを使用できます。

PL/SQL DBMS_ILMおよびDBMS_ILM_ADMINパッケージを使用すると、重要な本番ワークロードに悪影響を与えないように、ADOのILMアクティビティを管理できます。これらのパッケージを使用するには、データベース互換性を最低でも12.0に設定する必要があります。

DBMS_ILMパッケージのEXECUTE_ILMプロシージャは、ジョブを作成およびスケジュールして、ADOのポリシーを施行します。以前にスケジュールされたILMジョブに関係なく、EXECUTE_ILM()プロシージャはこの機能を提供します。すべてのジョブはすぐに実行するように作成およびスケジュールされますが、すぐに実行されるかどうかはスケジューラでキューに入れられているジョブの数に依存します。

ILMジョブの実行時間をさらに制御し、次のメンテナンス・ウィンドウまで待てない場合、EXECUTE_ILMプロシージャを使用できます。

DBMS_ILMパッケージのSTOP_ILMプロシージャは、すべてのジョブ、すべての実行中ジョブ、タスクIDに基づくジョブ、または特定のジョブを停止します。

例5-9で示すように、DBMS_ILM_ADMIN PL/SQLパッケージのCUSTOMIZE_ILMプロシージャを使用して、ADOの設定をカスタマイズできます。

たとえば、TBS_PERCENT_USEDおよびTBS_PERCENT_FREE ILMパラメータの値を設定するか、ABS_JOBLIMIT ILMパラメータを設定することができます。TBS_PERCENT_USEDおよびTBS_PERCENT_FREEは、表領域の割当てに基づいてデータを移動する時期を決定し、ABS_JOBLIMITは、同時に存在するADOジョブの絶対数を設定します。

DBMS_METADATA PL/SQLパッケージを使用して、ポリシーでオブジェクトを再作成することもできます。

関連項目:

- ILM ADOパラメータの詳細は、「ILM ADOパラメータの管理」を参照してください
- DBMS_ILM、DBMS_ILM_ADMINおよびDBMS_METADATA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5-9 CUSTOMIZE_ILMを使用したADO設定のカスタマイズ

```
SQL> BEGIN
  2  DBMS_ILM_ADMIN.CUSTOMIZE_ILM(DBMS_ILM_ADMIN.TBS_PERCENT_USED, 85);
```

```
  3  DBMS_ILM_ADMIN.CUSTOMIZE_ILM(DBMS_ILM_ADMIN.TBS_PERCENT_FREE, 25);
  4  END;
  5  /

SQL> BEGIN
  2  DBMS_ILM_ADMIN.CUSTOMIZE_ILM(DBMS_ILM_ADMIN.ABS_JOBLIMIT, 10);
  3  END;
  4  /
```

5.2.2.9 ビューを使用したADOのポリシーの監視

DBA_ILM*およびUSER_ILM*ビューを使用してデータベース・オブジェクトに関連付けられているADOのポリシーを表示および監視し、必要に応じてポリシーを容易に変更できます。

- DBA/USER_ILMDATAMOVEMENTPOLICIESビューは、ADOのILMポリシーのデータ移動の関連属性に固有の情報を表示します。
- DBA/USER_ILMTASKSビューは、プロシージャEXECUTE_ILMのタスクIDを表示します。ユーザーがプロシージャEXECUTE_ILMを呼び出すたびに、この特定の呼出しを追跡するためにタスクIDが戻されます。データベースによる期間の内部ILMタスクを追跡するため、タスクIDも生成されます。このビューは、ADOのすべてのILMタスクの情報を含みます。
- DBA/USER_ILMEVALUATIONDETAILSビューは、特定のタスクに考慮されるポリシーの詳細を表示します。ポリシーが評価用に選択された場合、ポリシーを実行するジョブの名前も表示されます。ポリシーが実行されなかった場合、このビューは理由も提供します。
- DBA/USER_ILMOBJECTSビューは、データベースのADOのすべてのオブジェクトおよびポリシーを表示します。特定の表領域で作成されたため、多くのオブジェクトは親オブジェクトを介してポリシーを継承します。このビューは、ポリシーとオブジェクト間のマッピングを提供します。継承されたポリシーの場合、このビューはポリシーが継承されるレベルも示します。
- DBA/USER_ILMPOLICIESビューは、データベースのADOのすべてのポリシーの詳細を表示します。
- DBA/USER_ILMRESULTSビューは、データベースのADOのデータ移動関連ILMジョブの情報を表示します。
- DBA_ILMPARAMETERSビューは、ADO関連パラメータの情報を表示します。

関連項目:

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

5.2.3 ADOおよびヒート・マップの制限事項

このトピックでは、ADOおよびヒート・マップに関連する制限事項について説明します。

ADOおよびヒート・マップに関連付けられている制限事項を次に示します。

- パーティション・レベルのADOおよび圧縮は、有効期間が経過した行を圧縮する行レベルADOポリシーを除き、時間的な有効性に対してサポートされます(アクセスまたは変更)。
- パーティション・レベルのADOおよび圧縮は、ORA_ARCHIVE_STATE列でパーティション化される場合に、インデータベース・アーカイブに対してサポートされます。
- ADOのカスタム・ポリシー(ユーザー定義関数)は、ポリシーが表領域レベルでデフォルトの場合にサポートされません。

- ストレージ層を使用する場合、ADOはターゲット表領域の記憶領域のチェックを実行しません。
- ADOは、オブジェクト型を使用した表またはマテリアライズド・ビューでサポートされません。
- ADOは、索引構成表またはクラスタではサポートされません。
- ADO同時実行性(ADOの同時ポリシー・ジョブの数)は、Oracleスケジューラの同時実行性に依存します。ADOのポリシー・ジョブが2回以上失敗した場合、ジョブが無効とマークされ、後でジョブを手動で有効にする必要があります。
- ADOには、表および表パーティションの移動に関連する制限事項があります。

関連項目:

- 表の移動の制限事項の詳細は、『Oracle Database SQL言語リファレンス』を参照してください
- 表パーティションの移動の制限事項の詳細は、Oracle Database SQL言語リファレンス』を参照してください

5.3 Oracle Databaseのデータの有効性および表示の制御

インデータベース・アーカイブおよび時間的な有効性を使用して、Oracle Databaseのデータの有効性および表示を制御できます。

この項では、次の項目について説明します。

- インデータベース・アーカイブの使用
- 時間的な有効性の使用
- 時間的な有効性による表の作成
- インデータベース・アーカイブおよび時間的な有効性の制限事項

5.3.1 インデータベース・アーカイブの使用

インデータベース・アーカイブでは、非アクティブとしてマークすることで、表内の行をアーカイブできます。

これらの非アクティブの行はデータベース内にあり、圧縮を使用して最適化できますが、アプリケーションに対して表示されません。セッション・パラメータを設定することで、必要に応じてコンプライアンス目的でこれらの行のデータを使用できます。

インデータベース・アーカイブを使用すると、アプリケーション・パフォーマンスを損なうことなく単一データベース内に長期間データをさらに格納できます。アーカイブされたデータを圧縮してバックアップ・パフォーマンスを向上でき、アプリケーションのアップグレード中にアーカイブされたデータのアップグレードを延期してアップグレードのパフォーマンスを向上できます。

表のインデータベース・アーカイブを管理するには、表のROW ARCHIVALを有効にし、表の非表示の列ORA_ARCHIVE_STATEを操作する必要があります。オプションで、ROW ARCHIVAL VISIBILITYセッション・パラメータのACTIVEまたはALLを指定します。

たとえば、例5-10のようなSQL文を使用して、表の行を表示または非表示にできます。ほとんどの状況でアクティブなデータのみを表示する目的ですが、特定の状況で必要な場合に備えてすべてのデータを保持します。

関連項目:

- SQL文を使用したインデータベース・アーカイブ機能の詳細は、『Oracle Database SQL言語リファレンス』を参照してください
- DBMS_ILMパッケージのARCHIVESTATENAME関数の詳細は、『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

Live SQL:

関連する例を Oracle Live SQL の *Oracle Live SQL: インデータベース・アーカイブの使用の例* で参照して実行してください。

例5-10 インデータベース・アーカイブの使用

```
/* Set visibility to ACTIVE to display only active rows of a table.*/
ALTER SESSION SET ROW ARCHIVAL VISIBILITY = ACTIVE;
```

```
CREATE TABLE employees_indbarch
 (employee_id NUMBER(6) NOT NULL,
  first_name VARCHAR2(20), last_name VARCHAR2(25) NOT NULL,
  email VARCHAR2(25) NOT NULL, phone_number VARCHAR2(20),
  hire_date DATE NOT NULL, job_id VARCHAR2(10) NOT NULL, salary  NUMBER(8,2),
  commission_pct NUMBER(2,2), manager_id NUMBER(6), department_id NUMBER(4)) ROW ARCHIVAL;

/* Show all the columns in the table, including hidden columns */
SELECT SUBSTR(COLUMN_NAME,1,22) NAME, SUBSTR(DATA_TYPE,1,20) DATA_TYPE, COLUMN_ID AS COL_ID,
  SEGMENT_COLUMN_ID AS SEG_COL_ID, INTERNAL_COLUMN_ID AS INT_COL_ID, HIDDEN_COLUMN, CHAR_LENGTH
  FROM USER_TAB_COLS WHERE TABLE_NAME='EMPLOYEES_INDBARCH';

NAME                   DATA_TYPE               COL_ID SEG_COL_ID INT_COL_ID HID CHAR_LENGTH
---------------------- -------------------- ---------- ---------- ---------- --- -----------
ORA_ARCHIVE_STATE      VARCHAR2                              1          1 YES        4000
EMPLOYEE_ID            NUMBER                        1          2          2 NO            0
FIRST_NAME             VARCHAR2                      2          3          3 NO           20
LAST_NAME              VARCHAR2                      3          4          4 NO           25
EMAIL                  VARCHAR2                      4          5          5 NO           25
PHONE_NUMBER           VARCHAR2                      5          6          6 NO           20
HIRE_DATE              DATE                          6          7          7 NO            0
JOB_ID                 VARCHAR2                      7          8          8 NO           10
SALARY                 NUMBER                        8          9          9 NO            0
COMMISSION_PCT         NUMBER                        9         10         10 NO            0
MANAGER_ID             NUMBER                       10         11         11 NO            0
DEPARTMENT_ID          NUMBER                       11         12         12 NO            0

/* Insert some data into the table */
INSERT INTO employees_indbarch(employee_id, first_name, last_name, email,
  hire_date, job_id, salary, manager_id, department_id)
  VALUES (251, 'Scott', 'Tiger', 'scott.tiger@example.com', '21-MAY-2009',
 'IT_PROG', 50000, 103, 60);

INSERT INTO employees_indbarch(employee_id, first_name, last_name, email,
  hire_date, job_id, salary, manager_id, department_id)
  VALUES (252, 'Jane', 'Lion', 'jane.lion@example.com', '11-JUN-2009',
  'IT_PROG', 50000, 103, 60);

/* Decrease the ORA_ARCHIVE_STATE column size to improve formatting in queries */
COLUMN ORA_ARCHIVE_STATE FORMAT a18;

/* The default value for ORA_ARCHIVE_STATE is '0', which means active */
SELECT employee_id, ORA_ARCHIVE_STATE FROM employees_indbarch;

EMPLOYEE_ID ORA_ARCHIVE_STATE
----------- ------------------
        251 0
        252 0

/* Insert a value into ORA_ARCHIVE_STATE to set the record to inactive status*/
UPDATE employees_indbarch SET ORA_ARCHIVE_STATE = '1' WHERE employee_id = 252;

/* Only active records are in the following query */
SELECT employee_id, ORA_ARCHIVE_STATE FROM employees_indbarch;

EMPLOYEE_ID ORA_ARCHIVE_STATE
----------- ------------------
        251 0

/* Set visibility to ALL to display all records */
```

```
ALTER SESSION SET ROW ARCHIVAL VISIBILITY = ALL;

SELECT employee_id, ORA_ARCHIVE_STATE FROM employees_indbarch;

EMPLOYEE_ID ORA_ARCHIVE_STATE
----------- ------------------
        251 0
        252 1
```

5.3.2 時間的な有効性の使用

時間的な有効性は、実際の有効性の期間を追跡できます。有効時間はデータのユーザーおよびアプリケーションによって設定でき、指定された有効時間または有効な時間範囲でデータを選択できます。

多くの場合、アプリケーションは、ビジネスの管理に関連する日付またはタイムスタンプでデータベースに記録されるファクトの有効性を示します。たとえば、保険業の補償の有効日を決定する人事管理(HR)アプリケーションの従業員の雇用日は有効な日付です。この日付は、従業員レコードがデータベースに入力された日付または時間とは異なります。前者の一時属性(雇用日)は有効時間(VT)と呼ばれ、後者(データベースに入力された日付)はトランザクション時間(TT)と呼ばれます。有効時間は通常ユーザーによって制御されますが、トランザクション時間はシステムが管理します。

ILMでは、有効時間属性は、ファクトが実業界で有効な時間および無効な時間を示すことができます。有効時間属性を使用すると、問合せは現在有効な行のみを表示できますが、閉じた注文や将来の雇用などの現在有効でないファクトを含む行は表示できません。

有効時間時制モデリングに不可欠な概念は次のとおりです。

- 有効期間

  これは時間のユーザー定義表現です。有効時間の例には、プロジェクトの開始日および終了日、従業員の雇用日および退職日が含まれます。

- 有効時間セマンティックを使用した表

  これらの表にはユーザー定義時間の1つ以上のディメンションがあり、それぞれ開始と終了があります。

- 有効時間フラッシュバック問合せ

  これは、有効時間ディメンションを使用して、ある時点および複数バージョンの問合せを実行する機能です。

有効期間は、表定義で指定された2つの日時列で構成されます。列を明示的に追加して有効期間を追加したり、列を自動的に作成できます。表の作成または表の変更プロセス中に有効期間を追加できます。

一時的な表問合せのセッション・レベルの表示制御をサポートするため、DBMS_FLASHBACK_ARCHIVE PL/SQLパッケージはENABLE_AT_VALID_TIMEプロシージャを提供します。プロシージャを実行するには、必要なシステムおよびオブジェクト権限が必要です。

次のPL/SQLプロシージャは、指定された時点で有効時間の表示を設定します。

```
SQL> EXECUTE DBMS_FLASHBACK_ARCHIVE.enable_at_valid_time
          ('ASOF', '31-DEC-12 12.00.01 PM');
```

次のPL/SQLプロシージャは、セッション・レベルで有効期間内の現在有効なデータに一時的なデータの表示を設定します。

```
SQL> EXECUTE DBMS_FLASHBACK_ARCHIVE.enable_at_valid_time('CURRENT');
```

次のプロシージャは、デフォルトの一時的な表の表示である全表の一時的なデータの表示を設定します。

```
SQL> EXECUTE DBMS_FLASHBACK_ARCHIVE.enable_at_valid_time('ALL');
```

関連項目:

- Oracle Temporalの詳細は、『Oracle Database開発ガイド』を参照してください
- DBMS_FLASHBACK_ARCHIVEパッケージの詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください
- CREATE TABLEまたはALTER TABLEを使用した有効時間の一時モデリングの開始の詳細は、『Oracle Database SQL言語リファレンス』を参照してください
- 表情報の監視に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

5.3.3 時間的な有効性による表の作成

このトピックの例は、時間的な有効性による表の作成方法を示します。

例5-11に、時間的な有効性の使用を示します。

Live SQL:

関連する例を Oracle Live SQL の *Oracle Live SQL: 時間的な有効性による表の作成* で参照して実行してください。

例5-11 時間的な有効性による表の作成

```
/* Create a time with an employee tracking timestamp */
/* using the specified columns*/
CREATE TABLE employees_temp (
       employee_id NUMBER(6) NOT NULL, first_name VARCHAR2(20), last_name VARCHAR2(25) NOT NULL,
       email VARCHAR2(25) NOT NULL, phone_number VARCHAR2(20), hire_date DATE NOT NULL,
       job_id VARCHAR2(10) NOT NULL, salary  NUMBER(8,2), commission_pct NUMBER(2,2),
       manager_id NUMBER(6), department_id NUMBER(4),
       PERIOD FOR emp_track_time);

DESCRIBE employees_temp

 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ---------------
 EMPLOYEE_ID                                           NOT NULL NUMBER(6)
 FIRST_NAME                                                     VARCHAR2(20)
 LAST_NAME                                             NOT NULL VARCHAR2(25)
 EMAIL                                                 NOT NULL VARCHAR2(25)
 PHONE_NUMBER                                                   VARCHAR2(20)
 HIRE_DATE                                             NOT NULL DATE
 JOB_ID                                                NOT NULL VARCHAR2(10)
 SALARY                                                         NUMBER(8,2)
 COMMISSION_PCT                                                 NUMBER(2,2)
 MANAGER_ID                                                     NUMBER(6)
 DEPARTMENT_ID                                                  NUMBER(4)

SQL> SELECT SUBSTR(COLUMN_NAME,1,22) NAME, SUBSTR(DATA_TYPE,1,28) DATA_TYPE, COLUMN_ID AS COL_ID,
     SEGMENT_COLUMN_ID AS SEG_COL_ID, INTERNAL_COLUMN_ID AS INT_COL_ID, HIDDEN_COLUMN
     FROM USER_TAB_COLS WHERE TABLE_NAME='EMPLOYEES_TEMP';
```

```
NAME                 DATA_TYPE                    COL_ID SEG_COL_ID INT_COL_ID HID
-------------------- ---------------------------- ------ ---------- ---------- ---
EMP_TRACK_TIME_START TIMESTAMP(6) WITH TIME ZONE                 1          1 YES
EMP_TRACK_TIME_END   TIMESTAMP(6) WITH TIME ZONE                 2          2 YES
EMP_TRACK_TIME       NUMBER                                                 3 YES
EMPLOYEE_ID          NUMBER                        1          3          4 NO
FIRST_NAME           VARCHAR2                      2          4          5 NO
LAST_NAME            VARCHAR2                      3          5          6 NO
EMAIL                VARCHAR2                      4          6          7 NO
PHONE_NUMBER         VARCHAR2                      5          7          8 NO
HIRE_DATE            DATE                          6          8          9 NO
JOB_ID               VARCHAR2                      7          9         10 NO
SALARY               NUMBER                        8         10         11 NO
COMMISSION_PCT       NUMBER                        9         11         12 NO
MANAGER_ID           NUMBER                       10         12         13 NO
DEPARTMENT_ID        NUMBER                       11         13         14 NO

/* Insert/update/delete with specified values for time columns */
INSERT INTO employees_temp(emp_track_time_start, emp_track_time_end, employee_id, first_name,
   last_name, email, hire_date, job_id, salary, manager_id, department_id)
   VALUES (TIMESTAMP '2009-06-01 12:00:01 Europe/Paris',
           TIMESTAMP '2012-11-30 12:00:01 Europe/Paris', 251, 'Scott', 'Tiger',
          'scott.tiger@example.com', DATE '2009-05-21', 'IT_PROG', 50000, 103, 60);

INSERT INTO employees_temp(emp_track_time_start, emp_track_time_end, employee_id, first_name,
     last_name, email, hire_date, job_id, salary, manager_id, department_id)
     VALUES (TIMESTAMP '2009-06-01 12:00:01 Europe/Paris',
             TIMESTAMP '2012-12-31 12:00:01 Europe/Paris', 252, 'Jane', 'Lion',
             'jane.lion@example.com', DATE '2009-06-11', 'IT_PROG', 50000, 103, 60);

UPDATE employees_temp set salary = salary + salary * .05
       WHERE emp_track_time_start <= TIMESTAMP '2009-06-01 12:00:01 Europe/Paris';

SELECT employee_id, SALARY FROM employees_temp;

EMPLOYEE_ID     SALARY
----------- ----------
        251      52500
        252      52500

/* No rows are deleted for the following statement because no records */
/* are in the specified track time. */
DELETE employees_temp WHERE emp_track_time_end < TIMESTAMP '2001-12-31 12:00:01 Europe/Paris';

0 rows deleted.

/* Show rows that are in a specified time period */
SELECT employee_id FROM employees_temp
       WHERE emp_track_time_start > TIMESTAMP '2009-05-31 12:00:01 Europe/Paris' AND
             emp_track_time_end < TIMESTAMP '2012-12-01 12:00:01 Europe/Paris';

EMPLOYEE_ID
-----------
        251

/* Show rows that are in a specified time period */
SELECT employee_id FROM employees_temp AS OF PERIOD FOR
       emp_track_time TIMESTAMP '2012-12-01 12:00:01 Europe/Paris';

EMPLOYEE_ID
```

```
----------
        252
```

5.3.4 インデータベース・アーカイブおよび時間的な有効性の制限事項

このトピックでは、インデータベース・アーカイブおよび時間的な有効性に関連付けられている制限事項を示します。

次のような制限があります。

- ILMは、時間的な有効性のOLTP表圧縮でサポートされていません。セグメント・レベルのILMおよび圧縮は、終了時間列でパーティション化される場合にサポートされます。
- ILMは、インデータベース・アーカイブのOLTP表圧縮でサポートされていません。セグメント・レベルのILMおよび圧縮は、ORA_ARCHIVE_STATE列でパーティション化される場合にサポートされます。

5.4 パーティション化を使用したILMシステムの手動実装

パーティション化を使用して、情報ライフサイクル管理(ILM)システムを手動で実装できます。

例5-12では、手動でストレージ層を作成し、それらのストレージ層に対して表をパーティション化し、そのデータベースに対して仮想プライベート・データベース(VPD)・ポリシーを設定して、オンライン・アーカイブ層のデータへのアクセスを制限する方法を示します。

関連項目:

- CREATE TABLE SQL文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください
- DBMS_RLS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5-12 ILMシステムの手動実装

```
REM Setup the tablespaces for the data

REM These tablespaces would be placed on a High Performance Tier
CREATE SMALLFILE TABLESPACE q1_orders DATAFILE 'q1_orders'
  SIZE 2M AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED LOGGING
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO ;

CREATE SMALLFILE TABLESPACE q2_orders DATAFILE 'q2_orders'
  SIZE 2M AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED LOGGING
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO ;

CREATE SMALLFILE TABLESPACE q3_orders DATAFILE 'q3_orders'
  SIZE 2M AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED LOGGING
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO ;

CREATE SMALLFILE TABLESPACE q4_orders DATAFILE 'q4_orders'
  SIZE 2M AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED LOGGING
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO ;

REM These tablespaces would be placed on a Low Cost Tier
CREATE SMALLFILE TABLESPACE "2006_ORDERS" DATAFILE '2006_orders'
  SIZE 5M AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED LOGGING
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO ;

CREATE SMALLFILE TABLESPACE "2005_ORDERS"  DATAFILE '2005_orders'
  SIZE 5M AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED LOGGING
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO ;

REM These tablespaces would be placed on the Online Archive Tier
CREATE SMALLFILE TABLESPACE "2004_ORDERS" DATAFILE '2004_orders'
  SIZE 5M AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED LOGGING
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO ;

CREATE SMALLFILE TABLESPACE old_orders DATAFILE 'old_orders'
  SIZE 15M AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED LOGGING
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO ;

REM Now create the Partitioned Table
```

```
CREATE TABLE allorders (
    prod_id       NUMBER       NOT NULL,
    cust_id       NUMBER       NOT NULL,
    time_id       DATE         NOT NULL,
    channel_id    NUMBER       NOT NULL,
    promo_id      NUMBER       NOT NULL,
    quantity_sold NUMBER(10,2) NOT NULL,
    amount_sold   NUMBER(10,2) NOT NULL)
 --
 -- table wide physical specs
 --
  PCTFREE 5 NOLOGGING
 --
 -- partitions
 --
 PARTITION BY RANGE (time_id)
  ( partition allorders_pre_2004 VALUES LESS THAN
     (TO_DATE('2004-01-01 00:00:00'
             ,'SYYYY-MM-DD HH24:MI:SS'
             ,'NLS_CALENDAR=GREGORIAN'
             )) TABLESPACE old_orders,
    partition allorders_2004 VALUES LESS THAN
     (TO_DATE('2005-01-01 00:00:00'
             ,'SYYYY-MM-DD HH24:MI:SS'
             ,'NLS_CALENDAR=GREGORIAN'
             )) TABLESPACE "2004_ORDERS",
    partition allorders_2005 VALUES LESS THAN
     (TO_DATE('2006-01-01 00:00:00'
             ,'SYYYY-MM-DD HH24:MI:SS'
             ,'NLS_CALENDAR=GREGORIAN'
             )) TABLESPACE "2005_ORDERS",
    partition allorders_2006 VALUES LESS THAN
     (TO_DATE('2007-01-01 00:00:00'
             ,'SYYYY-MM-DD HH24:MI:SS'
             ,'NLS_CALENDAR=GREGORIAN'
             )) TABLESPACE "2006_ORDERS",
    partition allorders_q1_2007 VALUES LESS THAN
     (TO_DATE('2007-04-01 00:00:00'
             ,'SYYYY-MM-DD HH24:MI:SS'
             ,'NLS_CALENDAR=GREGORIAN'
             )) TABLESPACE q1_orders,
    partition allorders_q2_2007 VALUES LESS THAN
     (TO_DATE('2007-07-01 00:00:00'
             ,'SYYYY-MM-DD HH24:MI:SS'
             ,'NLS_CALENDAR=GREGORIAN'
             )) TABLESPACE q2_orders,
    partition allorders_q3_2007 VALUES LESS THAN
     (TO_DATE('2007-10-01 00:00:00'
             ,'SYYYY-MM-DD HH24:MI:SS'
             ,'NLS_CALENDAR=GREGORIAN'
             )) TABLESPACE q3_orders,
    partition allorders_q4_2007 VALUES LESS THAN
     (TO_DATE('2008-01-01 00:00:00'
             ,'SYYYY-MM-DD HH24:MI:SS'
             ,'NLS_CALENDAR=GREGORIAN'
             )) TABLESPACE q4_orders);

ALTER TABLE allorders ENABLE ROW MOVEMENT;

REM Here is another example using INTERVAL partitioning
```

```
REM These tablespaces would be placed on a High Performance Tier
CREATE SMALLFILE TABLESPACE cc_this_month DATAFILE 'cc_this_month'
  SIZE 2M AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED LOGGING
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO ;

CREATE SMALLFILE TABLESPACE cc_prev_month DATAFILE 'cc_prev_month'
  SIZE 2M AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED LOGGING
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO ;

REM These tablespaces would be placed on a Low Cost Tier
CREATE SMALLFILE TABLESPACE cc_prev_12mth DATAFILE 'cc_prev_12'
  SIZE 2M AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED LOGGING
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO ;

REM These tablespaces would be placed on the Online Archive Tier
CREATE SMALLFILE TABLESPACE cc_old_tran DATAFILE 'cc_old_tran'
  SIZE 2M AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED LOGGING
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO ;

REM Credit Card Transactions where new partitions
REM are automatically placed on the high performance tier
CREATE TABLE cc_tran (
    cc_no       VARCHAR2(16) NOT NULL,
    tran_dt     DATE         NOT NULL,
    entry_dt    DATE         NOT NULL,
    ref_no      NUMBER       NOT NULL,
    description VARCHAR2(30) NOT NULL,
    tran_amt    NUMBER(10,2) NOT NULL)
 --
 -- table wide physical specs
 --
 PCTFREE 5 NOLOGGING
 --
 -- partitions
 --
 PARTITION BY RANGE (tran_dt)
 INTERVAL (NUMTOYMINTERVAL(1,'month') ) STORE IN (cc_this_month )
  ( partition very_old_cc_trans VALUES LESS THAN
     (TO_DATE('1999-07-01 00:00:00'
             ,'SYYYY-MM-DD HH24:MI:SS'
             ,'NLS_CALENDAR=GREGORIAN'
             )) TABLESPACE cc_old_tran ,
    partition old_cc_trans VALUES LESS THAN
     (TO_DATE('2006-07-01 00:00:00'
             ,'SYYYY-MM-DD HH24:MI:SS'
             ,'NLS_CALENDAR=GREGORIAN'
             )) TABLESPACE cc_old_tran ,
    partition last_12_mths VALUES LESS THAN
     (TO_DATE('2007-06-01 00:00:00'
             ,'SYYYY-MM-DD HH24:MI:SS'
             ,'NLS_CALENDAR=GREGORIAN'
             )) TABLESPACE cc_prev_12mth,
    partition recent_cc_trans VALUES LESS THAN
    (TO_DATE('2007-07-01 00:00:00'
            ,'SYYYY-MM-DD HH24:MI:SS'
            ,'NLS_CALENDAR=GREGORIAN'
            )) TABLESPACE cc_prev_month,
    partition new_cc_tran VALUES LESS THAN
     (TO_DATE('2007-08-01 00:00:00'
```

```
             ,'SYYYY-MM-DD HH24:MI:SS'
             ,'NLS_CALENDAR=GREGORIAN'
             )) TABLESPACE cc_this_month);

REM Create a Security Policy to allow user SH to see all credit card data,
REM PM only sees this years data,
REM and all other uses cannot see the credit card data

CREATE OR REPLACE FUNCTION ilm_seehist
  (oowner IN VARCHAR2, ojname IN VARCHAR2)
   RETURN VARCHAR2 AS con VARCHAR2 (200);
BEGIN
  IF SYS_CONTEXT('USERENV','CLIENT_INFO') = 'SH'
  THEN -- sees all data
    con:= '1=1';
  ELSIF SYS_CONTEXT('USERENV','CLIENT_INFO') = 'PM'
  THEN -- sees only data for 2007
    con := 'time_id > ''31-Dec-2006''';
  ELSE
    -- others nothing
    con:= '1=2';
  END IF;
  RETURN (con);
END ilm_seehist;
/
```

5.5 Oracle Enterprise ManagerでのILMヒート・マップおよびADOの管理

Oracle Enterprise Manager Cloud Controlでは、ヒート・マップおよび自動データ最適化を管理できます。

この項では、次の項目について説明します。

- データベース管理メニューへのアクセス
- 表領域レベルの自動データ最適化アクティビティの表示
- 任意の表領域のセグメント・アクティビティ詳細の表示
- 任意のオブジェクトのセグメント・アクティビティ詳細の表示
- 任意のオブジェクトのセグメント・アクティビティ履歴の表示
- 自動データ最適化でのアクティビティ・セグメントの検索
- セグメントのポリシーの表示
- バックグラウンド・アクティビティの無効化
- バックグラウンド自動データ最適化の実行頻度の変更
- 過去24時間のポリシー実行の表示
- 過去24時間で移動したオブジェクトの表示
- ポリシー詳細の表示
- ポリシーに関連付けられているオブジェクトの表示
- 実行前のポリシーの評価
- 単一のポリシーの実行
- ポリシー実行の停止
- ポリシー実行履歴の表示

関連項目:

- ヒート・マップおよびADOポリシー詳細の表示に使用できるビューの詳細は、「ビューを使用したヒート・マップ・トラッキング・データの表示」および「ビューを使用したADOのポリシーの監視」を参照してください
- Oracle Enterprise Manager Cloud Controlの管理の詳細は、*Oracle Enterprise Manager Cloud Control*管理者ガイドを参照してください

5.5.1 データベース管理メニューへのアクセス

データベースの**「管理」**メニューにアクセスするには:

1. Oracle Enterprise Manager Cloud Controlにログインします。
2. **「ターゲット」**メニューから**「データベース」**を選択します。

3. リストでデータベース名をクリックします。

4. データベース・ホームページで**「管理」**メニューが表示されます。

5.5.2 表領域レベルの自動データ最適化アクティビティの表示

データベースのサイズ別に選択した上位100の表領域のアクティビティを監視するには、次のステップに従います。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページでは、最終アクセス時刻、最終書込み時刻、最終全体スキャン時刻および最終参照スキャン時刻に基づいて上位100の表領域アクティビティ・ヒート・マップを表示できます。

   デフォルトでは、ヒート・マップ内の各ボックスのサイズはヒート・マップ内の表領域を表し、表領域のサイズによって決まります。情報ライフサイクル管理を使用すると、表領域からセグメント・レベルのヒート・マップにドリルダウンできます。

5.5.3 任意の表領域のセグメント・アクティビティ詳細の表示

任意の表領域のセグメント・アクティビティ詳細を表示するには、次のステップに従います。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページで、**「追加の表示」**ボタンをクリックします。

   Enterprise Managerにより、任意の表領域のセグメント・アクティビティ詳細を検索できるダイアログ・ボックスが表示されます。

3. ダイアログ・ボックスに**表領域名**を入力して、**「検索」**ボタンをクリックします。

   Enterprise Managerにより、表領域のセグメント・アクティビティ詳細が表示されます。

4. **「表領域ポリシーの編集」**ボタンをクリックすると、「ADO」タブが選択された状態で、表領域の編集ページが表示されます。圧縮または移動を許可する表領域のポリシーを作成できます。

5.5.4 任意のオブジェクトのセグメント・アクティビティ詳細の表示

任意のオブジェクトのセグメント・アクティビティ詳細を表示するには、次のステップに従います。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. カーソルをデータベース・レベルのヒート・マップ内の任意のボックスに移動します(各ボックスは表領域を表します)。表示するオブジェクトが属する**表領域**をクリックします。

   Enterprise Managerにより、表領域に属する上位100の最大オブジェクトの表領域レベルのヒート・マップが表示されます。「セグメント・アクティビティ」表には、上位100の最大オブジェクトのセグメント・アクティビティ詳細が表示されます。

3. 情報ライフサイクル管理ページで、**「追加の表示」**ボタンをクリックします。

   Enterprise Managerにより、表領域に属する任意のオブジェクトのセグメント・アクティビティ詳細を検索できるダイアログ・ボックスが表示されます。

4. ダイアログ・ボックスに**スキーマ名**と**オブジェクト名**を入力して、**「検索」**をクリックします。

   Enterprise Managerにより、オブジェクトのセグメント・アクティビティ詳細が表示されます。

5. **「オブジェクト・ポリシーの編集」**ボタンをクリックすると、「ADO」タブが選択された状態で、オブジェクトの編集ページが表示されます。圧縮または移動を許可するオブジェクトのポリシーを作成できます。

5.5.5 任意のオブジェクトのセグメント・アクティビティ履歴の表示

任意のオブジェクトのセグメント・アクティビティ履歴を表示するには、次のステップに従います。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. カーソルをデータベース・レベルのヒート・マップ内の任意のボックスに移動します(各ボックスは表領域を表します)。表示するオブジェクトが属する**表領域**をクリックします。

   Enterprise Managerにより、表領域に属する上位100の最大オブジェクトの表領域レベルのヒート・マップが表示されます。「セグメント・アクティビティ」表には、上位100の最大オブジェクトのセグメント・アクティビティ詳細が表示されます。

3. セグメント・アクティビティ詳細表でオブジェクトを選択し、**「アクティビティ履歴」**ボタンをクリックします。

   Enterprise Managerにより、「ADO」タブが選択された状態で、オブジェクトの編集ページが表示されます。「ADO」タブには、ポリシーのリストとセグメント・アクセス履歴が表示されます。

4. セグメントを選択して、日付範囲が過去60日間になるよう変更し、**「毎日」**オプションを選択して、**「リフレッシュ」**ボタンをクリックすると、そのオブジェクトの過去60日間に渡るセグメント・アクセス履歴が表示されます。

5.5.6 自動データ最適化でのアクティビティ・セグメントの検索

様々な期間中に自動データ最適化でセグメント・アクティビティを検索するには、次のステップに従います:

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページで、ヒート・マップの内のボックスのいずれかをクリックします。データベース・レベルの初期表示から、ヒート・マップのボックスをクリックして、その表領域の上位100の最大オブジェクトを表示します。その後にヒート・マップのボックスをクリックして、そのオブジェクトの上位100の最大セグメントを表示できます。

3. 最終アクセス時刻の1年前のタイムスタンプを入力し、**「実行」**をクリックします。昨年アクセスまたは変更されていないセグメントのリストが表示されます。セグメントはセグメント・サイズの降順にソートされます。

   オブジェクト・レベルのヒート・マップでは、表領域、名前、パーティション、タイプ、最終アクセス時刻、最終書込み時刻、最終全体スキャン時刻および最終参照スキャン時刻に基づいて、特定のセグメントを検索できます。

   行(セグメント)を選択してそのセグメントの行アクティビティを表示でき、**「ポリシー」**列をクリックするとセグメントに関連付けられているポリシーが表示されます。

5.5.7 セグメントのポリシーの表示

セグメントに関連付けられているポリシーを表示するには、次のステップに従います。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページで、**「実行」**をクリックします。検索結果のセグメントに関連付けられたポリシーがある場合は、ゼロ以外の件数が「ポリシー」列に表示されます。件数の上にマウスを移動すると、継承されたポリシーを含む、セグメントに関連付けられているポリシーが表示されます。件数がゼロの場合は、ポリシーはセグメントに関連付けられていません。

   データベース・レベルのヒート・マップから表領域レベルのヒート・マップにどりダウンします。表領域レベルのヒート・マップでは、Enterprise Managerに、表領域に属する上位100のオブジェクトが表示されます。Enterprise Managerで、各オブジェクトのポリシーの件数が列に表示されます。

   表領域レベルのヒート・マップからオブジェクトを選択し、オブジェクト・レベルのヒート・マップにドリルダウンします。Enterprise Managerには、オブジェクトに属する上位100の最大セグメントが表示されます。Enterprise Managerで、各セグメントのポリシーの件数が「ポリシー」列に表示されます。

5.5.8 バックグラウンド・アクティビティの無効化

自動データの最適化のバックグラウンド評価およびスケジューラを無効にするには、次のステップに従います:

ILM ADO ENABLEDパラメータの詳細は、ILM ADOパラメータの管理を参照してください。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページで、**「ポリシー」**タブをクリックします。

3. ポリシー実行の設定セクションで、**「構成」**をクリックします。

   ポリシー実行の設定ダイアログ・ボックスが表示されます。

4. **「ステータス」**ドロップダウンを**「無効」**に変更して、**「OK」**をクリックします。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。このページの「ポリシー実行設定」セクションにある「ポリシー」タブで、ステータスが「無効」と表示されます。

5.5.9 バックグラウンド自動データ最適化の実行頻度の変更

情報ライフサイクル管理のバックグランドの評価とスケジューラの実行頻度を変更するには、次のステップに従います。

ILM ADO EXECUTION INTERVALパラメータの詳細は、ILM ADOパラメータの管理を参照してください。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページで、**「ポリシー」**タブをクリックします。

3. ポリシー実行の設定セクションで、**「構成」**をクリックします。

   ポリシー実行の設定ダイアログ・ボックスが表示されます。

4. **「実行間隔」**の値を現在表示されている値より低いまたは高い数値に変更し、**「OK」**をクリックします。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。このページの「ポリシー実行設定」にある「ポリシー」タブで、実行間隔に新しい値が表示されます。

5.5.10 過去24時間のポリシー実行の表示

過去24時間で実行されたポリシーを表示し、ポリシーの実行で移動または圧縮されたオブジェクトを表示するには、次のステップに従います。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページで、**「ポリシー」**タブをクリックします。

3. **「過去24時間のポリシー実行サマリー」**行の**完了したポリシー**・リンクまたは**失敗したポリシー**・リンクをクリックします。クリックすると、いずれかの過去24時間の実行履歴が表示されます。

   ポリシー実行の詳細ダイアログ・ボックスが表示され、過去24時間のポリシーの実行の詳細が表示されます。

5.5.11 過去24時間で移動したオブジェクトの表示

過去24時間に移動されたオブジェクトを表示し、これらのオブジェクトを移動したポリシー/ジョブを表示するには、次のステップに従います。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページで、**「ポリシー」**タブをクリックします。

3. **「過去24時間のポリシー実行サマリー」**行で、**移動したオブジェクト**・リンクをクリックします。

   ポリシー実行履歴ダイアログ・ボックスが表示され、過去24時間で実行されたジョブとポリシー、および移動されたオブジェクトの実行履歴が表示されます。

5.5.12 ポリシー詳細の表示

特定のADOポリシーの詳細を表示するには、次のステップに従います。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページで、**「ポリシー」**タブをクリックします。

3. ポリシー詳細を表示するには、ポリシー表のポリシー名のリンクをクリックするか、ポリシー表の行を選択して**「ポリシーの詳細を表示」**ボタンをクリックします。

5.5.13 ポリシーに関連付けられているオブジェクトの表示

特定のポリシーに関連付けられているオブジェクトを表示するには、次のステップに従います。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページで、**「ポリシー」**タブをクリックします。

3. 「オブジェクト」列の件数をクリックします。

   ポリシーに関連付けられたオブジェクトが表示されます。

5.5.14 実行前のポリシーの評価

ポリシーを実行する前にポリシーを評価するには、次のステップに従います。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページで、**「ポリシー」**タブをクリックします。

3. 「評価」リージョンで、**「評価」**をクリックします。

   表示されるダイアログ・ボックスでは、データベース内のすべてのポリシーを評価するか、特定スキーマのオブジェクトに影響するすべてのポリシーを評価するかを選択できます。

4. **スキーマ名**を入力し、**「OK」**をクリックして評価を開始します。

   「評価」ダイアログ・ボックスが閉じられ、評価が発行されます。ページをリフレッシュしたり、その他のEnterprise Managerタスクを実行して、後から「ポリシー」タブを再表示したりできます。実行すると、**「評価」**リージョンの**「完了」**の数が1つ増えます。

5. 「評価」リージョンの**「完了」**の数のリンクをクリックすると、完了したすべての評価がリストされたダイアログ・ボックスが表示されます。

6. 最新評価の**タスクID**をクリックすると、評価タスクに現在含まれているすべてのオブジェクトがリストされた「評価詳細」ダイアログ・ボックスが表示されます(これらのオブジェクトは、評価が実行されると圧縮または移動されます)。

7. **「OK」**をクリックすると、実行にオブジェクトのリストが含まれます。「評価詳細」ダイアログ・ボックスが閉じられます。

8. 最新の評価が含まれる「評価」表の行(一番上の行)を選択し、**「実行」**をクリックします。

   実行中に影響を受けるオブジェクトがリストされた、評価の実行ダイアログ・ボックスが表示されます。

9. **「OK」**をクリックして実行します。

   評価の実行ダイアログ・ボックスが閉じられます。実行結果は、「過去24時間のポリシー実行サマリー」の「ジョブ」または「ポリシー」リージョンにある**「完了」**か**「失敗」**のリンクをクリックすると参照できます。また、タスクの状態が**INACTIVE**から**ACTIVE**や**COMPLETED**に変わると、「評価」の「完了」の数が1つ減ります。

5.5.15 単一のポリシーの実行

ポリシーに関連付けられているオブジェクトでポリシーをすぐに実行するには、次のステップに従います。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページで、**「ポリシー」**タブをクリックします。

3. ポリシーを選択し、**「ポリシーの実行」**をクリックします。

   「ポリシーの実行」ダイアログ・ボックスが表示され、選択したポリシーによって評価されるすべてのオブジェクトがリストされます。また、ダイアログ・ボックスには、実行されるEXECUTE_ILMコマンドを表示するための**非表示/表示**ボタンも含まれます。このポリシーが有効化されたオブジェクトのみが含まれます。

5.5.16 ポリシー実行の停止

ポリシーの実行を停止するには、次のステップに従います。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページで、**「ポリシー」**タブをクリックします。

3. 「過去24時間のポリシー実行サマリー」の「ジョブ」リージョンで、**「進行中」**リンクをクリックします。このステップでは、進行中のタスク/ジョブが少なくとも1つあることを想定しています。

   表示されるダイアログ・ボックスには、現在実行されているすべてのタスクがリストされています。

4. 表にリストされているいずれかのタスクの**「ジョブ」**リンクをクリックします。

   タスクの一部として実行されているジョブの詳細がリストされたダイアログ・ボックスが表示されます。

5. **「OK」**をクリックします。

   「ジョブ詳細」ダイアログ・ボックスが閉じられます。

6. 表で行を選択し、**「実行停止」**をクリックします。

   実行プロセスの停止を確認するダイアログ・ボックスが表示されます。

7. **「OK」**をクリックします。

   確認のダイアログ・ボックスが閉じられます。

5.5.17 ポリシー実行履歴の表示

特定のポリシーの実行履歴を表示するには、次のステップに従います。

1. **「管理」**メニューから、**「記憶域」**、**情報ライフサイクル管理**の順に選択します。

   Enterprise Managerでは、情報ライフサイクル管理ページが表示されます。

2. 情報ライフサイクル管理ページで、**「ポリシー」**タブをクリックします。

3. ポリシーを選択し、**「実行履歴」**をクリックします。

   ポリシー実行履歴ダイアログ・ボックスが表示され、選択したポリシーの実行履歴が表示されます。詳細には、ジョブ情報および移動または圧縮されたオブジェクトが含まれます。

6 データ・ウェアハウス環境でのパーティション化の使用

パーティション化機能により、データ・ウェアハウス環境でのパフォーマンスを改善できます。

この章では、データ・アクセスを大幅に強化し、アプリケーション・パフォーマンス全体を向上させるパーティション化機能について説明します。パーティション化による改善は、データ・ウェアハウス環境で見られるような、何百万もの行や何GBものデータを含む表や索引にアクセスするアプリケーションでは特に効果があります。データ・ウェアハウスには大規模な表が含まれることが多く、これらの大規模な表を管理し、これらの表で最適な問合せパフォーマンスを得るためのテクニックが必要です。

この章の構成は、次のとおりです。

6.1 データ・ウェアハウスについて

データ・ウェアハウスとは、トランザクション処理ではなく、問合せおよび分析用に設計されたリレーショナル・データベースです。

データ・ウェアハウスには、通常、トランザクション・データから導出された履歴データが含まれますが、別のソースからのデータを含めることもできます。データ・ウェアハウスは分析処理をトランザクション処理の負荷と分離し、組織で、様々なソースからのデータを整理統合できるようにします。

リレーショナル・データベースの他に、データ・ウェアハウス環境には、ETL(抽出、変換、ロード)ソリューション、分析処理やデータ・マイニングの機能、クライアント分析ツール、およびその他のアプリケーション(データ収集およびビジネス・ユーザーへのデータ配布のプロセスを管理する)が含まれます。

関連項目:

Oracle Databaseデータウェアハウス・ガイド

6.2 データ・ウェアハウスでのスケーラビリティ

パーティション化により、データベース・オブジェクトが小さく分割され、管理しやすい小さなオブジェクトへのアクセスが可能になり、データ・ウェアハウスの拡張が推進されます。小さなオブジェクトに直接アクセスすることにより、次のようなデータ・ウェアハウスのスケーラビリティ要件に対処できます。

この項では、次の項目について説明します。

6.2.1 データベース・サイズの拡大

大きいデータベース・オブジェクトを小さく分割する機能によって、大規模データベースの効率的な管理が透過的に簡略化されます。

個々のパーティションやサブパーティションを指定して操作することで、大きいデータベース・オブジェクトを管理できます。パーティション化されたオブジェクトの次の利点について考慮してください。

- 大きなサイズのデータベースを管理する際にも、バックアップおよびリカバリを小さな単位で実行できます。
- データベース・オブジェクトの一部を圧縮ストレージに配置することができます(それ以外の部分は圧縮しません)。
- パーティション化により、データを様々なストレージ層で透過的に保持することができ、大容量データを格納するコストを節減できます。詳細は、「時間ベース情報の管理およびメンテナンス」を参照してください。

6.2.2 個々の表の拡大(表の行数の増大)

大きな表のスキャンは小さな表のスキャンに比べて時間がかかります。パーティション表に対する問合せでは、表全体のサイズに比べて小さい、1つ以上のパーティションがアクセスされます。

また、問合せは、索引についてのパーティション絞込みを利用できます。索引の一部をディスクから読み取るには、索引全体を読み取るよりも時間がかかりません。ローカル・パーティション索引のようにパーティション化戦略が表と同じ構造の索引は、パーティション単位でアクセスおよびメンテナンスを行うことができます。

問合せ、DMLおよびDDL文の処理速度を上げるためにパラレル実行を使用すると、個別のパーティションに個々のデータセットが存在するメリットが活用されます。個々のパラレル実行サーバーは、パーティション境界ごとに、独自のデータセットを処理します。

6.2.3 システムに問合せを行うユーザーの増加

パーティション化されている状態では、ユーザーは孤立した小さなデータセットを問い合せる可能性が高くなります。

その結果、すべてのユーザーが1つの大きなデータセットを問い合せる場合よりも、データベースは結果を速く返せるようになります。データ競合が発生する可能性が低くなります。

6.2.4 複雑な問合せの増加

小さなデータセットを使用すると、複雑な問合せを速く実行することができます。

小さなデータセットがアクセスされる場合、複雑な計算はメモリーで処理される可能性が高くなります。これは、パフォーマンスの観点でメリットがあり、アプリケーションのI/O要件が削減されます。大きなデータセットでは、場合によっては問合せを完了するために一時表領域に書き込む必要があり、データベースの記憶域に対して追加のI/Oが発生します。

6.3 データ・ウェアハウスでのパフォーマンスのためのパーティション化

適切なパフォーマンスは、データ・ウェアハウスが成功するために重要です。

データベースに対して実行する分析は、問合せがアクセスするデータが大容量で表のサイズが数TBに及ぶ場合でも、適度な時間で返されることが必要です。パーティション化によってデータ・アクセスやアプリケーション処理の速度が向上します。これによって、ハードウェア・コストもかかりすぎない優れたデータ・ウェアハウスが実現します。

この項では、次の項目について説明します。

- データ・ウェアハウスでのパーティション・プルーニング

- データ・ウェアハウスでのパーティション・ワイズ結合
- データ・ウェアハウスにおける索引とパーティション索引
- データ・ウェアハウスでのマテリアライズド・ビューとパーティション化

6.3.1 データ・ウェアハウスでのパーティション・プルーニング

パーティション・プルーニングは、データ・ウェアハウスの重要なパフォーマンス機能です。

パーティション・プルーニングでは、オプティマイザによりSQL文のFROM句とWHERE句が分析され、パーティション・アクセス・リストを構築するときに不要なパーティションが削除されます。結果として、Oracle Databaseは、SQL文に関連するパーティションに限定して処理を実行できるようになります。

パーティション・プルーニングを行うと、ディスクから取得するデータ容量が大幅に削減され、処理時間が短縮します。このため、問合せのパフォーマンスが向上し、リソース使用率が最適化されます。

この項では、次の項目について説明します。

- 基本的なパーティション・プルーニング技法
- 高度なパーティション・プルーニング技法

パーティション・プルーニングの詳細および静的パーティション・プルーニングと動的パーティション・プルーニングの違いは、「可用性、管理性およびパフォーマンスのためのパーティション化」を参照してください。

6.3.1.1 基本的なパーティション・プルーニング技法

オプティマイザでプルーニングに使用される述語のタイプには様々なものがあります。

パーティション・プルーニングに最もよく使用される述語のタイプは、等価、範囲およびINリストの3つです。例として次の問合せを考えてみます。

```
SELECT SUM(amount_sold) day_sales
FROM sales
WHERE time_id = TO_DATE('02-JAN-1998', 'DD-MON-YYYY');
```

salesのパーティション列に対する等価述語が含まれるので、この問合せでは1つの述語にプルーニングされ、次の実行計画に反映されます。

```
-------------------------------------------------------------------------------------------------
|  Id | Operation                | Name  | Rows| Bytes | Cost (%CPU)| Time     |Pstart| Pstop |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |       |     |       | 21 (100)   |          |      |       |
|   1 |  SORT AGGREGATE          |       | 1   | 13    |            |          |      |       |
|   2 |   PARTITION RANGE SINGLE |       | 485 | 6305  | 21 (10)    | 00:00:01 | 5    | 5     |
| * 3 |    TABLE ACCESS FULL     | SALES | 485 | 6305  | 21 (10)    | 00:00:01 | 5    | 5     |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
  3 - filter("TIME_ID"=TO_DATE('1998-01-02 00:00:00', 'yyyy-mm-dd hh24:mi:ss'))
```

同様に、time_id列に対して範囲述語またはINリスト述語があれば、オプティマイザを使用して一連のパーティションにプルーニングされます。使用できる述語のタイプは、パーティション化のタイプにより決まります。範囲述語は、ハッシュ・パーティション表でのプルーニングには使用できませんが、その他のすべてのパーティション化のプルーニングには使用できます。ただし、リスト・パーティション表では、範囲述語は、連続したパーティションのセットへはマッピングされません。等価述語およびINリスト述語は、すべてのパーティション化方法でプルーニングできます。

6.3.1.2 高度なパーティション・プルーニング技法

Oracle Databaseのプルーニング機能では、複数のパーティション表を対象とする、より複雑な述語やSQL文も効率よく処理されます。

パーティション表が、WHERE条件により限定された別の表のサブセットに結合される場合などは、その典型例です。たとえば、次の問合せを考えてみます。

```
SELECT t.day_number_in_month, SUM(s.amount_sold)
  FROM sales s, times t
  WHERE s.time_id = t.time_id
    AND t.calendar_month_desc='2000-12'
  GROUP BY t.day_number_in_month;
```

データベースが、右側にあるtimes表とのネステッド・ループ結合を実行した場合、問合せはtimes表のこの行に対応するパーティションにのみアクセスすることになります。つまりプルーニングは暗黙に実行されます。ただし、データベースでハッシュ結合またはソート・マージ結合が実行される場合は、そうはなりません。WHERE述語の対象となる表がパーティション表と比べて小さく、かつパーティション表から多くのレコードやパーティションが除かれると予想される場合は、再帰的な副問合せを使用して動的パーティション・プルーニングが実行されます。副問合せによるプルーニングを実行するかどうかは、オプティマイザによるコストベースの内部的な判断により決定されます。

ハッシュ結合操作を使用した実行計画は、たとえば次のようになります。

```
-------------------------------------------------------------------------------------------------------
| Id| Operation                    |  Name |  Rows | Bytes| Cost (%CPU)|  Time  | Pstart | Pstop |
-------------------------------------------------------------------------------------------------------
|  0| SELECT STATEMENT             |       |       |      | 761 (100)  |        |        |       |
|  1|  HASH GROUP BY               |       |    20 | 640  | 761 (41)   |00:00:10|        |       |
|* 2|   HASH JOIN                  |       | 19153 | 598K | 749 (40)   |00:00:09|        |       |
|* 3|    TABLE ACCESS FULL         | TIMES |    30 |  570 |  17 (6)    |00:00:01|        |       |
|  4|     PARTITION RANGE SUBQUERY |       |  918K | 11M  |   655 (33) |00:00:08| KEY(SQ)|KEY(SQ)|
|  5|      TABLE ACCESS FULL       | SALES |   918 | 11M  |   655 (33) |00:00:08| KEY(SQ)|KEY(SQ)|
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------------
  2 - access("S"."TIME_ID"="T"."TIME_ID")
  3 - filter("T"."CALENDAR_MONTH_DESC"='2000-12')
```

この実行計画では、PSTART列とPSTOP列のKEY(SQ)値に示されるように、副問合せを使用してsales表で動的パーティション・プルーニングが実行されたことがわかります。

次にOR述語を使用した高度なプルーニングの例を示します。

```
SELECT p.promo_name promo_name, (s.profit - p.promo_cost) profit
FROM
   promotions p,
   ( SELECT
      sales.promo_id,
      SUM(sales.QUANTITY_SOLD * (costs.UNIT_PRICE - costs.UNIT_COST)) profit
   FROM
      sales, costs
   WHERE
      ((sales.time_id BETWEEN TO_DATE('01-JAN-1998','DD-MON-YYYY',
                  'NLS_DATE_LANGUAGE = American') AND
      TO_DATE('01-JAN-1999','DD-MON-YYYY', 'NLS_DATE_LANGUAGE = American')
   OR
      (sales.time_id BETWEEN TO_DATE('01-JAN-2001','DD-MON-YYYY',
```

```
                  'NLS_DATE_LANGUAGE = American') AND
      TO_DATE('01-JAN-2002','DD-MON-YYYY', 'NLS_DATE_LANGUAGE = American')))
      AND sales.time_id = costs.time_id
      AND sales.prod_id = costs.prod_id)
   GROUP BY
      sales.promo_id) s
WHERE s.promo_id = p.promo_id
ORDER BY profit
DESC;
```

この問合せでは、sales表とcosts表が結合されます。sales表は、time_id列でレンジ・パーティション化されています。この問合せの条件の1つはtime_idに関する2つの述語です。それらがOR演算子で1つにまとめられています。このOR述語を基にsales表のパーティションに対してプルーニングが行われ、sales表とcosts表の単一結合が実行されます。実行計画は次のようになります。

```
-------------------------------------------------------------------------------------------------------
| Id| Operation               |  Name    |Rows |Bytes |TmpSp|Cost(%CPU)| Time    | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------
|  0| SELECT STATEMENT        |          | 4   |  200 |     | 3556 (14)| 00:00:43|       |       |
|  1|  SORT ORDER BY          |          | 4   |  200 |     | 3556 (14)| 00:00:43|       |       |
|* 2|   HASH JOIN             |          | 4   |  200 |     | 3555 (14)| 00:00:43|       |       |
|  3|    TABLE ACCESS FULL    |PROMOTIONS| 503 | 16599|     |    16 (0)| 00:00:01|       |       |
|  4|    VIEW                 |          |   4 |   68 |     | 3538 (14)| 00:00:43|       |       |
|  5|     HASH GROUP BY       |          |   4 |  164 |     | 3538 (14)| 00:00:43|       |       |
|  6|      PARTITION RANGE OR |          | 314K|   12M|     |  3321 (9)| 00:00:40|KEY(OR)|KEY(OR)|
|* 7|       HASH JOIN         |          | 314K|   12M| 440K|  3321 (9)| 00:00:40|       |       |
|* 8|        TABLE ACCESS FULL| SALES    | 402K| 7467K|     |  400 (39)| 00:00:05|KEY(OR)|KEY(OR)|
|  9| TABLE ACCESS FULL       | COSTS    |82112| 1764K|     |   77 (24)| 00:00:01|KEY(OR)|KEY(OR)|
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
  2 - access("S"."PROMO_ID"="P"."PROMO_ID")
  7 - access("SALES"."TIME_ID"="COSTS"."TIME_ID" AND "SALES"."PROD_ID"="COSTS"."PROD_ID")
  8 - filter("SALES"."TIME_ID"<=TO_DATE('1999-01-01 00:00:00', 'yyyy-mm-dd hh24:mi:ss') AND
      "SALES"."TIME_ID">=TO_DATE('1998-01-01 00:00:00', 'yyyy-mm-dd hh24:mi:ss') OR
      "SALES"."TIME_ID">=TO_DATE('2001-01-01 00:00:00', 'yyyy-mm-dd hh24:mi:ss') AND
      "SALES"."TIME_ID"<=TO_DATE('2002-01-01 00:00:00', 'yyyy-mm-dd hh24:mi:ss'))
```

ある列が複数の列でレンジ・パーティション化されている場合は、さらにプルーニングが実行されます。ある特定の述語に対して、決して合致しないと判断できるパーティションはスキップされます。これにより、複数列に対する範囲述語がある場合またはパーティション列のプリフィックスに対する述語が存在しない場合のパフォーマンスが最適化されます。

パーティション・プルーニングのヒントは、「パーティション・プルーニングのヒント」を参照してください。

6.3.2 データ・ウェアハウスでのパーティション・ワイズ結合

パーティション・ワイズ結合では、結合がパラレルで実行されるときにパラレル実行サーバー間で交換されるデータ量が最小限に抑えられ、問合せのレスポンス時間が短縮されます。

パーティション・ワイズ結合を使用すると、レスポンス時間は大幅に短縮され、CPUとメモリー・リソースの使用率が改善されます。パラレル・パーティション・ワイズ結合は、大きな結合を効率的かつ高速に処理するために一般的に使用されます。パーティション・ワイズ結合では、フル(完全)またはパーシャル(部分)を選択できます。どちらの種類の結合を使用するかはOracle Databaseによって判別されます。

パラレル・パーティション・ワイズ結合に加えて、SELECT DISTINCT句およびSQLウィンドウ関数を使用した問合せでは、パラレル・パーティション・ワイズ操作を実行できます。

この項では、次の項目について説明します。

- フル・パーティション・ワイズ結合
- パーシャル・パーティション・ワイズ結合
- パーティション・ワイズ結合のメリット
- パラレル・パーティション・ワイズ結合のパフォーマンスに関する考慮点

関連項目:

- パーティション・ワイズ操作の詳細は、「パーティション・ワイズ操作」を参照してください
- データ・ウェアハウスおよび最適化方法の詳細は、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください

6.3.2.1 フル・パーティション・ワイズ結合

フル・パーティション・ワイズ結合が行われるのは、同じキーでともにパーティション化された2つの表が問合せで結合される場合です。

同等のパーティション化は、表のパーティション・レベル、サブパーティション・レベル、あるいはパーティションとサブパーティションを組み合せたレベルで可能です。参照パーティション化は、同等のパーティション化を保証する簡単な方法です。フル・パーティション・ワイズ結合は、シリアルでもパラレルでも実行できます。

パーティション・ワイズ結合の詳細は、「可用性、管理性およびパフォーマンスのためのパーティション化」を参照してください。

次の例は、orders表とorder_items表のフル・パーティション・ワイズ結合を示します。order_items表が参照パーティション化されています。

```
CREATE TABLE orders
( order_id     NUMBER(12) NOT NULL
, order_date   DATE NOT NULL
, order_mode   VARCHAR2(8)
, order_status VARCHAR2(1)
, CONSTRAINT orders_pk PRIMARY KEY (order_id)
)
PARTITION BY RANGE (order_date)
( PARTITION p_before_jan_2006 VALUES LESS THAN (TO_DATE('01-JAN-2006','dd-MON-yyyy'))
, PARTITION p_2006_jan VALUES LESS THAN (TO_DATE('01-FEB-2006','dd-MON-yyyy'))
, PARTITION p_2006_feb VALUES LESS THAN (TO_DATE('01-MAR-2006','dd-MON-yyyy'))
, PARTITION p_2006_mar VALUES LESS THAN (TO_DATE('01-APR-2006','dd-MON-yyyy'))
, PARTITION p_2006_apr VALUES LESS THAN (TO_DATE('01-MAY-2006','dd-MON-yyyy'))
, PARTITION p_2006_may VALUES LESS THAN (TO_DATE('01-JUN-2006','dd-MON-yyyy'))
, PARTITION p_2006_jun VALUES LESS THAN (TO_DATE('01-JUL-2006','dd-MON-yyyy'))
, PARTITION p_2006_jul VALUES LESS THAN (TO_DATE('01-AUG-2006','dd-MON-yyyy'))
, PARTITION p_2006_aug VALUES LESS THAN (TO_DATE('01-SEP-2006','dd-MON-yyyy'))
, PARTITION p_2006_sep VALUES LESS THAN (TO_DATE('01-OCT-2006','dd-MON-yyyy'))
, PARTITION p_2006_oct VALUES LESS THAN (TO_DATE('01-NOV-2006','dd-MON-yyyy'))
, PARTITION p_2006_nov VALUES LESS THAN (TO_DATE('01-DEC-2006','dd-MON-yyyy'))
, PARTITION p_2006_dec VALUES LESS THAN (TO_DATE('01-JAN-2007','dd-MON-yyyy'))
)
PARALLEL;

CREATE TABLE order_items
( order_id NUMBER(12) NOT NULL
, product_id NUMBER NOT NULL
```

```
, quantity NUMBER NOT NULL
, sales_amount NUMBER NOT NULL
, CONSTRAINT order_items_orders_fk FOREIGN KEY (order_id) REFERENCES
orders(order_id)
)
PARTITION BY REFERENCE (order_items_orders_fk)
PARALLEL;
```

典型的なデータ・ウェアハウス問合せでは、大容量のデータがスキャンされます。基礎となっている実行計画では、列Rows、Bytes、Cost (%CPU)、TimeおよびTQが削除されます。

```
EXPLAIN PLAN FOR
SELECT o.order_date
, sum(oi.sales_amount) sum_sales
FROM orders o
, order_items oi
WHERE o.order_id = oi.order_id
AND o.order_date BETWEEN TO_DATE('01-FEB-2006','DD-MON-YYYY')
                     AND TO_DATE('31-MAY-2006','DD-MON-YYYY')
GROUP BY o.order_id
, o.order_date
ORDER BY o.order_date;

-----------------------------------------------------------------------------------------------
| Id  | Operation                          | Name        | Pstart| Pstop |IN-OUT| PQ Distrib |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |             |       |       |      |            |
|   1 |  PX COORDINATOR                    |             |       |       |      |            |
|   2 |   PX SEND QC (ORDER)               | :TQ10001    |       |       | P->S | QC (ORDER) |
|   3 |    SORT GROUP BY                   |             |       |       | PCWP |            |
|   4 |     PX RECEIVE                     |             |       |       | PCWP |            |
|   5 |      PX SEND RANGE                 | :TQ10000    |       |       | P->P | RANGE      |
|   6 |       SORT GROUP BY                |             |       |       | PCWP |            |
|   7 |        PX PARTITION RANGE ITERATOR|             |     3 |     6 | PCWC |            |
|*  8 |         HASH JOIN                  |             |       |       | PCWP |            |
|*  9 |          TABLE ACCESS FULL         | ORDERS      |     3 |     6 | PCWP |            |
|  10 |          TABLE ACCESS FULL         | ORDER_ITEMS |     3 |     6 | PCWP |            |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   8 - access("O"."ORDER_ID"="OI"."ORDER_ID")
   9 - filter("O"."ORDER_DATE"<=TO_DATE(' 2006-05-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

6.3.2.2 パーシャル・パーティション・ワイズ結合

Oracle Databaseでは、パラレル時のみパーシャル・パーティション・ワイズ結合が実行できます。

フル・パーティション・ワイズ結合と異なり、パーシャル・パーティション・ワイズ結合では、両方の表ではなく、一方の表のみを結合キーでパーティション化する必要があります。パーティション化された表は、参照表と呼ばれます。もう一方の表は、パーティション化してもしなくてもかまいません。パーシャル・パーティション・ワイズ結合は、フル・パーティション・ワイズ結合よりも一般的です。

パーシャル・パーティション・ワイズ結合を実行するために、データベースによって、参照表のパーティション化に基づいて、もう一方の表が動的にパーティション化または再パーティション化されます。もう一方の表が再パーティション化された後は、フル・パーティション・ワイズ結合と同様に実行されます。

次の例は、データ・ウェアハウスの一般的なシナリオでの、通話明細レコードの表cdrsを示します。この表は、時間隔ハッシュ・

パーティション化されています。

```
CREATE TABLE cdrs
( id                 NUMBER
, cust_id            NUMBER
, from_number        VARCHAR2(20)
, to_number          VARCHAR2(20)
, date_of_call       DATE
, distance           VARCHAR2(1)
, call_duration_in_s NUMBER(4)
) PARTITION BY RANGE(date_of_call)
INTERVAL (NUMTODSINTERVAL(1,'DAY'))
SUBPARTITION BY HASH(cust_id)
SUBPARTITIONS 16
(PARTITION p0 VALUES LESS THAN (TO_DATE('01-JAN-2005','dd-MON-yyyy')))
PARALLEL;
```

cdrs表は、非パーティション表callersとcust_id列によって結合され、通話時間の長い顧客順に並べられます。

```
EXPLAIN PLAN FOR
SELECT c.cust_id
,      c.cust_last_name
,      c.cust_first_name
,      AVG(call_duration_in_s)
,      COUNT(1)
,      DENSE_RANK() OVER
       (ORDER BY (AVG(call_duration_in_s) * COUNT(1)) DESC) ranking
FROM   callers c
,      cdrs    cdr
WHERE cdr.cust_id = c.cust_id
AND cdr.date_of_call BETWEEN TO_DATE('01-JAN-2006','dd-MON-yyyy')
                         AND TO_DATE('31-DEC-2006','dd-MON-yyyy')
GROUP BY c.cust_id
, c.cust_last_name
, c.cust_first_name
ORDER BY ranking;
```

この実行計画はパーシャル・パーティション・ワイズ結合を示しています。計画では、列Rows、Bytes、Cost （%CPU）、TimeおよびTQが削除されます。

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name     | Pstart| Pstop |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |          |       |       |      |            |
|   1 |  WINDOW NOSORT                     |          |       |       |      |            |
|   2 |   PX COORDINATOR                   |          |       |       |      |            |
|   3 |    PX SEND QC (ORDER)              | :TQ10002 |       |       | P->S | QC (ORDER) |
|   4 |     SORT ORDER BY                  |          |       |       | PCWP |            |
|   5 |      PX RECEIVE                    |          |       |       | PCWP |            |
|   6 |       PX SEND RANGE                | :TQ10001 |       |       | P->P | RANGE      |
|   7 |        HASH GROUP BY               |          |       |       | PCWP |            |
|*  8 |         HASH JOIN                  |          |       |       | PCWP |            |
|   9 |          PART JOIN FILTER CREATE   | :BF0000  |       |       | PCWP |            |
|  10 |           BUFFER SORT              |          |       |       | PCWC |            |
|  11 |            PX RECEIVE              |          |       |       | PCWP |            |
|  12 |             PX SEND PARTITION (KEY)| :TQ10000 |       |       | S->P | PART (KEY) |
|  13 |              TABLE ACCESS FULL     | CALLERS  |       |       |      |            |
|  14 |          PX PARTITION RANGE ITERATOR|          |   367 |   731 | PCWC |            |
|  15 |           PX PARTITION HASH ALL    |          |     1 |    16 | PCWC |            |
|* 16 |            TABLE ACCESS FULL       | CDRS     |  5857 | 11696 | PCWP |            |
```

```
---------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   8 - access("CDR"."CUST_ID"="C"."CUST_ID")
  16 - filter("CDR"."DATE_OF_CALL">=TO_DATE(' 2006-01-01 00:00:00', 'syyyy-mm-dd
hh24:mi:ss') AND "CDR"."DATE_OF_CALL"<=TO_DATE('
              2006-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

6.3.2.3 パーティション・ワイズ結合のメリット

パーティション・ワイズ結合には、次のようないくつかの利点があります。

これらのメリットは、次の項目で説明しています。

- 通信オーバーヘッドの削減
- メモリー要件の削減

6.3.2.3.1 通信オーバーヘッドの削減

パーティション・ワイズ結合では、パラレルで実行する場合の通信オーバーヘッドが削減されます。

この削減は、デフォルトでは、パラレル実行サーバー・セットが結合操作をパラレル実行するときに、各表を結合列に関して非結合行のサブセットとして再分配する必要があるため発生します。これらの非結合行のサブセットは、1つのパラレル実行サーバーによって、組単位で結合されます。

パーティション・ワイズ結合では、2つの表が結合列でパーティション化されているため、パーティションの再分配を回避できます。この機能により、各パラレル実行サーバーが、一致するパーティションの組を結合できます。パラレル実行によるパフォーマンス向上は、ノード間パラレル実行を行うOracle Real Application Clusters構成ではさらに顕著になります。

パーティション・ワイズ結合により、インターコネクト・トラフィックが大幅に削減されます。この機能の使用は、Oracle Real Application Clustersを使用する大規模な意思決定支援(DSS)構成に欠かせません。現在、超並列処理(MPP)や対称型マルチプロセッシング(SMP)クラスタなど、ほとんどのOracle Real Application Clustersプラットフォームでは、その処理能力に比べてインターコネクト帯域幅が制限されています。インターコネクト帯域幅は、ディスク帯域幅と同等であることが理想的ですが、このような例はまれです。そのため、Oracle Real Application Clustersでのほとんどの結合操作では、パーティション・ワイズ結合がパラレル実行されない場合にインターコネクトの待ち時間が長くなります。

6.3.2.3.2 メモリー要件の削減

パーティション・ワイズ結合は、同じ表のデータセット全体で同様の結合操作を行う場合に比べて、メモリー所要量が少なくなります。

シリアル結合では、一致するパーティション1組ずつに対して結合が実行されます。データがパーティション間で均等に分散されている場合、メモリー要件はパーティションの数で分割され、パラレル・サーバー間のデータの分散に偏りはありません。

パラレル結合の場合、メモリー要件は、パラレルで結合されるパーティションの組数によって異なります。たとえば、並列度が20でパーティションの数が100の場合、2つのパーティションの結合が20ずつ同時に実行されるため、必要なメモリー量は5分の1になります。パーティション・ワイズ結合でメモリーの必要量が減ることは、パフォーマンスに直接的な好影響があります。たとえば、結合では、ハッシュ結合の作成フェーズ中にブロックがディスクに書き込まれる必要がなくなります。

6.3.2.4 パラレル・パーティション・ワイズ結合のパフォーマンスに関する考慮点

オプティマイザは、パーティション・ワイズ結合を使用するかどうかを決定する場合に、メリットとデメリットを比較します。

オプティマイザは、次に基づいてパーティション・ワイズ結合を使用するかどうかを選択します。

- レンジ・パーティション化の場合、パーティション・サイズが異なると、データの偏りによって応答時間が長くなることがあります。つまり、パラレル実行サーバーによっては、他のサーバーより結合完了までの時間が長くなります。ハッシュ・(サブ)パーティション化を使用して、パーティション・ワイズ結合を使用可能にすることをお薦めします。パーティションの数が2の累乗であれば、ハッシュ・パーティション化によって偏りが発生する可能性が低く抑えられるためです。偏りの発生を最小限にするために、ハッシュ・パーティション化キーが一意であることが理想的です。
- パーティション・ワイズ結合に使用されるパーティションの数は、可能であれば問合せサーバーの数の倍数にします。たとえば、並列度が16の場合は、パーティションの数を16、32または64にします。パーティションの数が半端な場合、一部のパラレル実行サーバーの使用率が低くなります。たとえば、17組の均等に分散されたパーティションがある場合、1組のみが最後の結合で処理され、そのときに他の組は待機する必要があります。これは、実行開始時には、各パラレル実行サーバーが別のパーティションの組に対して動作するためです。この最初のフェーズ後には、1組のみが残ります。そのため、1つのパラレル実行サーバーがこの残った組を結合し、その間、他のすべてのパラレル実行サーバーはアイドル状態になります。

パラレル結合によって、リモートI/O処理が発生する場合があります。たとえば、MPP構成で実行されるOracle Real Application Clusters環境では、一致するパーティションの組が同じノードに存在しない場合、パーティション・ワイズ結合を行うために、リモートI/Oのための追加のノード間通信が必要になります。これは、結合が実行されるノードに、少なくとも1つのパーティションを転送する必要があるためです。この場合は、パーティション・ワイズ結合を使用するより、データを明示的に再配置する方が適しています。

6.3.3 データ・ウェアハウスにおける索引とパーティション索引

索引は表と関連付けられているオプション構造で、これにより表に対するSQL文をより迅速に実行することができます。

表スキャンは多くのデータ・ウェアハウスでごく一般的ですが、多くの場合、索引を使用することで問合せの速度がさらに速くなることがあります。

Bツリー索引とビットマップ索引はどちらも、パーティション表のローカル索引として作成できます。この場合、索引は表のパーティション化戦略を継承します。Bツリー索引は、パーティション表または非パーティション表のグローバル・パーティション索引として作成することもできます。

この項では、次の項目について説明します。

- ローカル・パーティション索引
- 非パーティション索引
- グローバル・パーティション索引

パーティション索引の詳細は、「可用性、管理性およびパフォーマンスのためのパーティション化」を参照してください。

6.3.3.1 ローカル・パーティション索引

ローカル索引の場合、特定の索引パーティションのキーはすべて、基礎となる表の1つのパーティションに格納されている行のみを参照します。

ローカル索引は、基礎となる表と同一レベルでパーティション化されます。基礎となる表と同じ列で索引をパーティション化し、同じ数のパーティションまたはサブパーティションを作成して、基礎となる表の対応するパーティションと同じパーティション・バウンドを設定します。

また、Oracle Databaseは、基礎となる表のパーティションが追加、削除、マージまたは分割されたり、ハッシュ・パーティションやサブパーティションが追加または結合されたりした場合は、索引のパーティション化を自動的にメンテナンスします。これにより、索引のパーティションが表と同一レベルに保たれます。

データ・ウェアハウス・アプリケーションでは、ローカル非同一キー索引によってパフォーマンスを向上させることができます。これは、索引キーに基づくレンジ問合せによって、多数の索引パーティションをパラレルにスキャンできるためです。次の例では、ローカルBツリー索引がパーティション表customers_dwに作成されます。

```
CREATE INDEX cust_last_name_ix
 ON customers_dw(last_name) LOCAL
 PARALLEL NOLOGGING ;
```

ビットマップ索引では、カーディナリティが低い列を非常に効率よく格納できる方法が使用されます。ビットマップ索引はデータ・ウェアハウス環境で使用され、特にスター・スキーマを実装しているデータ・ウェアハウスで広く使用されています。1つのスター・スキーマは、中心の大きなファクト表と、ファクト表のデータについて説明する複数の小さなディメンション表で構成されます。

たとえば、ディメンション表customers、products、promotions、timesおよびchannelsで説明されている、ファクト表であるsales表を考察します。ビットマップ索引では、スター・スキーマ(またはスターに似ているスキーマ)に対する高速の問合せ取得を最適化する、スター型変換を使用できます。

ファクト表の外部キー列は、ビットマップ索引の理想的な候補です。一般的に、全行数に比べて固有の値が少ないためです。ファクト表は、レンジ・パーティション化、またはレンジと他の方法を組み合せてパーティション化されていることが多いですが、その場合は、ローカル・ビットマップ索引を作成する必要があります。パーティション表のグローバル・ビットマップ索引はサポートされません。

次の例では、ローカル・パーティション・ビットマップ索引がsales表に作成されます。

```
CREATE BITMAP INDEX prod_id_ix
ON sales(prod_id) LOCAL
PARALLEL NOLOGGING;
```

関連項目:

スター型変換の詳細は、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください

6.3.3.2 非パーティション索引

非パーティション索引は、非パーティション表にもパーティション表にも作成できます。

非パーティション索引はデータ・ウェアハウス環境で非パーティション表で主に使用され、一般的に一意の制約のステータスをデータ・ウェアハウス環境で施行する必要がある場合に一意性を施行します。パーティション表で非パーティション・グローバル索引を使用すると、主キーまたは一意キーを施行できます。非パーティション(グローバル)索引は、パーティション化キーに含まれない1つの列または複数の列に対する等価述語またはINリストに基づいて、通常ごく少数の行を取得する問合せで役立ちます。このような場合、一致するすべての行を検出するために、1つの索引をスキャンする方が多数の索引パーティションをスキャンするよりも速く行えます。

パーティション列以外の列の一意索引はグローバルにする必要があります。これは、キーにパーティション化キーが含まれていない一意ローカル非同一キー索引はサポートされていないためです。制御されたデータ・ロード処理や一意制約の施行によるパフォーマンス・コストのために、データ・ウェアハウスでは必ずしも一意キーが施行されるとはかぎりません。グローバル索引は、数十億の行を含む表では非常に大きくなることがあります。

次の例では、グローバル一意索引がsales表に作成されます。

```
CREATE UNIQUE INDEX sales_unique_ix
  ON sales(cust_id, prod_id, promo_id, channel_id, time_id)
  PARALLEL NOLOGGING;
```

この索引からメリットを得る問合せはほとんどありません。データ・ロード・ウィンドウが非常に限られているシステムでは、この索引の作成や管理を行わないことを検討する必要があります。

6.3.3.3 グローバル・パーティション索引

グローバル・パーティション索引は、非パーティション表にもパーティション表にも作成できます。

グローバル・パーティション索引では、特定の索引パーティションのキーが、複数の基礎となる表のパーティションまたはサブパーティションに格納されている行を参照する場合があります。グローバル索引ではレンジ・パーティション化またはハッシュ・パーティション化が可能ですが、定義するパーティション表はどのタイプでもかまいません。

グローバル索引は、GLOBAL属性を指定することで作成されます。データベース管理者は、グローバル索引の作成時に最初のパーティション化を定義して、それ以降はパーティション化のメンテナンスを行う必要があります。索引パーティションは、必要に応じてマージまたは分割できます。

グローバル索引が役立つのは、少数の行を取得するために索引を介した表へのアクセス・パスを使用する問合せがある場合です。索引をパーティション化することで、問合せの大半に対して索引の大部分を排除できます。パーティション表では、パーティション・プルーニングの実行のために含めるべき列または複数の列に、表のパーティション化キーが含まれない場合に、グローバル・パーティション索引を検討してください。

次の例では、グローバル・ハッシュ・パーティション索引がsales表に作成されます。

```
CREATE INDEX cust_id_prod_id_global_ix
ON sales(cust_id,prod_id)
GLOBAL PARTITION BY HASH (cust_id)
( PARTITION p1 TABLESPACE tbs1
, PARTITION p2 TABLESPACE tbs2
, PARTITION p3 TABLESPACE tbs3
, PARTITION p4 TABLESPACE tbs4
)
PARALLEL NOLOGGING;
```

6.3.4 データ・ウェアハウスでのマテリアライズド・ビューとパーティション化

パフォーマンス向上のためにデータ・ウェアハウスで使用されている1つのテクニックに、サマリーの作成があります。これは、特殊なタイプの集計ビューであり、問合せを実行する前に、効率が悪い結合および集計操作を事前に計算し、その結果をデータベース内の表に格納することで、問合せ実行時間を短縮します。

たとえば、サマリー表が、地域別および製品別の売上合計を含むように作成できます。

サマリー、すなわちこのマニュアルやデータ・ウェアハウスに関するマニュアルで説明されている集計は、マテリアライズド・ビューと呼ばれるスキーマ・オブジェクトを使用してOracle Databaseに作成されます。データ・ウェアハウスではマテリアライズド・ビューによって問合せパフォーマンスが改善されます。

データベースでは、マテリアライズド・ビューに対する透過的なリライトがサポートされます。したがって、マテリアライズド・ビューの計算済結果を利用するために元の問合せを変更する必要はありません。問合せを実行するかわりに、データベースは計算済結果を1つ以上のマテリアライズド・ビューから取得し、必要であればデータに対してその他の操作を実行し、問合せ結果を返します。

関連項目:

データ・ウェアハウスおよびマテリアライズド・ビューの詳細は、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください

6.3.4.1 パーティション・マテリアライズド・ビュー

マテリアライズド・ビューの基礎となる記憶域は表構造です。表をパーティション化するのと同様にマテリアライズド・ビューをパーティション化できます。

マテリアライズド・ビューに対して実行するようにデータベースが問合せをリライトしたとき、その問合せは、表に対して直接実行される問合せと同じパフォーマンス特性を得られます。リライトされた問合せでは、マテリアライズド・ビューのパーティションを排除できます。問合せ結果を取得するために、表または他のマテリアライズド・ビューとの結合が必要な場合は、リライトされた問合せで、パーティション・ワイズ結合を利用できます。

例6-1は、売上結果を国レベルで集計する圧縮パーティション・マテリアライズド・ビューを作成する方法を示します。このマテリアライズド・ビューは、売上数を国レベル(またはさらに大きな地区や地域のレベル)で要約する問合せを利用します。

例6-1 圧縮パーティション・マテリアライズド・ビュー

```
CREATE MATERIALIZED VIEW country_sales
PARTITION BY HASH (country_id)
PARTITIONS 16
COMPRESS FOR OLTP
PARALLEL NOLOGGING
ENABLE QUERY REWRITE
AS SELECT co.country_id
, co.country_name
, co.country_subregion
, co.country_region
, sum(sa.quantity_sold) country_quantity_sold
, sum(sa.amount_sold) country_amount_sold
FROM sales sa
, customers cu
, countries co
WHERE sa.cust_id = cu.cust_id
AND cu.country_id = co.country_id
GROUP BY co.country_id
, co.country_name
, co.country_subregion
, co.country_region;
```

関連項目:

データ・ウェアハウスおよびマテリアライズド・ビューの詳細は、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください

6.4 データ・ウェアハウスでの管理性

データ・ウェアハウスには履歴データが格納されます。データ・ウェアハウスの重要な部分はデータのロードとパージです。パーティション化は、データ・ウェアハウスでのデータ管理に役立つ高機能テクノロジです。

この項では、次の項目について説明します。

- パーティション交換ロード
- パーティション化と索引
- 表からのデータの削除
- パーティション化とデータの圧縮

関連項目:

パーティション索引、交換および表の統計の収集および管理の詳細は、『Oracle Database SQLチューニング・ガイド』を参照してください

6.4.1 パーティション交換ロード

パーティション交換ロード(PEL)を使用してパーティションを追加できます。

PELを使用するときは、1つのパーティションとまったく同じに見える別の表を作成し、索引や制約(存在する場合)も同じものを含めます。コンポジット・パーティション表を使用する場合、この表は、コンポジット・パーティション表のサブパーティション化戦略と一致するパーティション化戦略を使用する必要があります。次に、既存の表パーティションをこの表と交換します。データ・ロード・シナリオでは、この表にデータをロードしておきます。表のユーザーの問合せに影響しないように、表に索引を構築して制約を実装します。次にPELを実行します。これはデータのロードに比べると非常に影響の少ないトランザクションです。日単位のレンジ・パーティション戦略と毎日のロードの組合せは、データ・ウェアハウス環境では一般的です。

次の例は、sales表のパーティション交換ロードを示します。

```
ALTER TABLE sales ADD PARTITION p_sales_jun_2007
VALUES LESS THAN (TO_DATE('01-FEB-2007','dd-MON-yyyy'));

CREATE TABLE sales_jun_2007 COMPRESS FOR OLTP
AS SELECT * FROM sales WHERE 1=0;
```

次に、表sales_jun_2007に2007年6月の売上数を移入して、sales表に実装されていたのと同じビットマップ索引と制約を作成します。

```
CREATE BITMAP INDEX time_id_jun_2007_bix ON sales_jun_2007(time_id) NOLOGGING;
CREATE BITMAP INDEX cust_id_jun_2007_bix ON sales_jun_2007(cust_id) NOLOGGING;
CREATE BITMAP INDEX prod_id_jun_2007_bix ON sales_jun_2007(prod_id) NOLOGGING;
CREATE BITMAP INDEX promo_id_jun_2007_bix ON sales_jun_2007(promo_id) NOLOGGING;
CREATE BITMAP INDEX channel_id_jun_2007_bix ON sales_jun_2007(channel_id) NOLOGGING;

ALTER TABLE sales_jun_2007 ADD CONSTRAINT prod_id_fk FOREIGN KEY (prod_id) REFERENCES
products(prod_id);
ALTER TABLE sales_jun_2007 ADD CONSTRAINT cust_id_fk FOREIGN KEY (cust_id) REFERENCES
customers(cust_id);
ALTER TABLE sales_jun_2007 ADD CONSTRAINT promo_id_fk FOREIGN KEY (promo_id) REFERENCES
promotions(promo_id);
ALTER TABLE sales_jun_2007 ADD CONSTRAINT time_id_fk FOREIGN KEY (time_id) REFERENCES times(time_id);
ALTER TABLE sales_jun_2007 ADD CONSTRAINT channel_id_fk FOREIGN KEY (channel_id) REFERENCES
channels(channel_id);
```

次に、パーティションを交換します。

```
ALTER TABLE sales
EXCHANGE PARTITION p_sales_jun_2007
WITH TABLE sales_jun_2007
INCLUDING INDEXES;
```

パーティション交換ロードの詳細は、「パーティションの管理」を参照してください。

6.4.2 パーティション化と索引

パーティション・メンテナンス操作の実行が最も簡単にできるのはローカル索引です。

ローカル索引は、パーティション管理が行われるときにグローバル索引を無効にしません。ローカル索引を別の表の同じ索引と交換するとき、索引パーティションが無効にならないように、PEL文でINCLUDING INDEXESを使用します。PELの場合は、ロードの際にグローバル索引を更新できます。PEL文でUPDATE GLOBAL INDEXES拡張機能を使用します。索引の更新が必要な場合、PELの実行時間が長くなります。

6.4.3 表からのデータの削除

通常、データ・ウェアハウスでは一定の時間枠のデータが保存されます。たとえば、3年分の履歴データが格納されます。

パーティション化を行うと、非常に簡単に表からデータをパージすることができます。データをパージするにはDROP PARTITION文または TRUNCATE PARTITION文を使用します。一般的な戦略として、表のデータをアンロードするためにパーティション交換ロードを使用する方法、パーティションを空の表と置き換えてからパーティションを削除する方法もあります。交換した別の表は、空にしたり削除したりする前にアーカイブします。

削除または切捨て操作によって、グローバル索引またはグローバル・パーティション索引が無効になる場合があります。ローカル索引は有効なまま変化しません。ローカル索引パーティションは、表パーティションを削除するときに削除されます。

次の例は、sales表のsales_1995パーティションを削除する方法を示します。

```
ALTER TABLE sales
 DROP PARTITION sales_1995
 UPDATE GLOBAL INDEXES PARALLEL;
```

6.4.4 パーティション化とデータの圧縮

パーティション表のデータは、パーティション単位で圧縮することができます。

圧縮データの使用は、データが頻繁に変更されない場合に最も効果的です。一般的なデータ・ウェアハウスのシナリオでは、データが古くなるにつれてデータの変更は減ります。あるいは、シナリオによってはデータの挿入のみが行われます。パーティション管理機能を使用して、パーティション単位でデータを圧縮できます。Oracle Databaseでは、圧縮データに対してすべてのDML操作を行えますが、データの変更は圧縮されていない表で実行する方が効率的です。

圧縮を有効化するパーティションの変更は、パーティションにこれから挿入されるデータのみに適用されます。パーティションの既存のデータを圧縮するには、パーティションを移動する必要があります。圧縮の有効化とパーティションの移動は、1つの操作で実行できます。

ビットマップ索引のあるパーティション表に対して表の圧縮を使用するには、最初に圧縮属性を導入する前に、次の処理を行う必要があります。

1. ビットマップ索引をUNUSABLEとしてマークします。
2. 圧縮属性を設定します。
3. 索引を再作成します。

圧縮パーティションを、まったく圧縮されていない既存のパーティション表に含める場合は、圧縮パーティションを追加する前に、既存のビットマップ索引をすべて削除するか、UNUSABLEとしてマークする必要があります。これは、パーティションにデータが含まれるかどうかにかかわらず実行する必要があります。また、表の1つ以上のパーティションを圧縮する操作とも無関係です。これは、Bツリー索引のみを含むパーティション表には適用されません。

次の例は、sales表のSALES_1995パーティションを圧縮する方法を示します。

```
ALTER TABLE sales
 MOVE PARTITION sales_1995
 COMPRESS FOR OLTP
 PARALLEL NOLOGGING;
```

表またはパーティションがディスクに占める領域が減ると、I/Oに制約がある環境では、大きな表スキャンのパフォーマンスが改善されることがあります。

7 オンライン・トランザクション処理環境でのパーティション化の使用

パーティション化機能は、OLTPシステムに非常に役立ちます。

オンライン・トランザクション処理(OLTP)システムの急速な発展とユーザー数の増大のため、パーティション化はデータ・ウェアハウス環境に加えてOLTPシステムできわめて有効です。

多くの場合、OLTPシステムでは、競合を減らして大多数のユーザーをサポートするために、パーティション化が使用されます。また、コスト効率のよい方法での大容量データの格納など、OLTPシステムが直面する規制要件の対処にも役立ちます。

この章の構成は、次のとおりです。

- オンライン・トランザクション処理システムについて
- オンライン・トランザクション処理環境でのパフォーマンス
- オンライン・トランザクション処理環境での管理性

7.1 オンライン・トランザクション処理システムについて

オンライン・トランザクション処理(OLTP)システムは、現在の企業でよく使用されているデータ処理システムです。典型的なOLTPシステムの例としては、受注、小売および金融取引のシステムがあります。

OLTPシステムの最大の特徴は、データ・ウェアハウス環境とは異なる独特のデータ使用方法にあります。ただし、大容量データの保持やライフサイクルに応じたデータの使用状況と重要性といった特徴は同じです。

OLTP環境の主な特性は次のとおりです。

- 短いレスポンス時間

  OLTP環境では、電話勧誘販売での調査結果の入力など、様々な種類の対話的で臨機応変の使用方法が大半を占めます。OLTPシステムでは、ユーザーの生産性を保つためにレスポンス時間を短くする必要があります。

- 小さなトランザクション

  通常、OLTPシステムは、限定された少量のデータを読み取って処理します。データ処理の大部分は単純で、複雑な結合は比較的まれです。問合せとDMLのワークロードが常に混在しています。たとえば、コール・センターでは多数の従業員のうちの一人が、通話ごとに顧客詳細を取得し、顧客の苦情を入力しながら、その顧客の過去の通信記録を確認します。

- データ・メンテナンス操作

  定期的または臨時に実行する必要があるレポート・プログラムやデータ更新プログラムがあることは珍しくありません。このようなプログラムは、ユーザーが他のタスクを行っているときにバックグラウンドで実行されますが、多数のデータ集約型処理を伴うことがあります。たとえば、大学では学生を講義に振り分けるバッチ・ジョブを起動する一方で、学生が自らオンラインで講義の申込みを行うことができます。

- 多数のユーザー

  OLTPシステムのユーザーは非常に多数に及び、そのうち多くのユーザーが同じデータに同時にアクセスしようとします。たとえば、オンライン・オークションのWebサイトでは、数十万人(数百万人ではないにしても)ものユーザーが同時にWebサイトのデータにアクセスする可能性があります。

- 高い同時実行性

ユーザー数が多く、レスポンス時間が短く、トランザクションが小さいため、OLTP環境の同時実行性は非常に高くなります。数千単位の同時ユーザー数の要件も珍しくありません。

- 大容量データ

  アプリケーションのタイプ、ユーザー数およびデータ保存期間によって変わりますが、OLTPシステムは非常に大規模になることがあります。たとえば、ある銀行ではすべての顧客が、過去12か月のすべての取引を示すオンライン・バンキング・システムにアクセスできます。

- 高可用性

  OLTPシステムの可用性要件は、しばしば非常に高くなります。OLTPシステムが使用できない場合、非常に多くのユーザーが影響を受け、組織は大きな損失を被ることがあるためです。たとえば、株式取引システムは取引時間においてきわめて高い可用性の要件があります。

- ライフサイクルに関連するデータ使用状況

  データ・ウェアハウス環境と同じく、OLTPシステムでも、時間の経過に伴ってデータ・アクセス・パターンが変化します。たとえば、月末にはすべての活動口座の月利が計算されます。

OLTP環境をパーティション化する利点を次に示します。

- 大きなデータベースのサポート

  高可用性戦略の一環として、大きなサイズのデータベースを効率よく管理するために、バックアップおよびリカバリを小さな単位で実行できます。通常、OLTPシステムはバックアップ中もオンラインのままであり、バックアップの実行中もユーザーは引き続きシステムにアクセスできます。バックアップ・プロセスによって、オンライン・ユーザーがパフォーマンスの大幅な劣化を被ることはありません。

  パーティション化では、データベース・オブジェクトを部分的に圧縮して格納できるため、OLTPシステムの領域要件を抑えることにつながります。圧縮していない行の更新トランザクションは、圧縮データの更新よりも効率よく処理されます。

  パーティション化により、データを様々なストレージ層で透過的に保持することができ、大容量データを格納するコストを節減できます。

- データ・メンテナンスのためのパーティション・メンテナンス操作(DMLのかわりに使用)

  データ・メンテナンス操作(多くの場合はパージ)の場合、Oracle Databaseのオンライン索引メンテナンス機能と組み合せてパーティション・メンテナンス操作を活用できます。パーティション管理操作では、同内容のDML操作に比べて生成されるREDOが少なくなります。

- ホット・スポットの消去によって高い同時実行性を得る可能性

  OLTP環境の一般的なシナリオでは、主キー制約を施行するために使用される索引値は単調に増加します。このため、同時実行性や潜在的な競合率が高い領域が生成されます。新たに挿入を実行するたびに、同じセットの索引ブロックの更新が試行されるためです。パーティション索引、特にハッシュ・パーティション索引を使用すると、このような状況が緩和されます。

7.2 オンライン・トランザクション処理環境でのパフォーマンス

OLTP環境のパフォーマンスは、効率のよい索引アクセスに大きく左右されるため、最適な索引戦略の選択が重要です。

次の項では、OLTP環境で索引をパーティション化するかどうかを決定するためのベスト・プラクティスを説明します。

- 索引をパーティション化するかどうかの決定
- 索引構成表でのパーティション化の使用方法

7.2.1 索引をパーティション化するかどうかの決定

問合せの選択性とOLTPアプリケーションの高い同時実行性のため、OLTP環境でのパーティション化の使用に関して、正しい索引戦略の選択が重要な決定事項であることは疑う余地もありません。競合が減ることにより、アプリケーションでサポートできるユーザー数が増加します。

様々な索引構造の主な利点とトレードオフを説明するために、次に基本的なルールを示します。

- 非パーティション索引(パーティション索引の個々のセグメントよりも大きい)では、索引のアクセス・パスが選択される場合は常に索引プローブ(スキャン)が1回行われます。1つの表に対して1つのセグメントしかないためです。データ・アクセス時間とアクセスされるブロック数は、パーティション表でも非パーティション表でも同じです。

  非パーティション索引には、パーティションの自律性はないため、行IDに影響するすべてのパーティション・メンテナンス操作(たとえば、削除、切捨て、移動、マージ、結合、分割などの操作)について索引メンテナンス操作が必要です。

- パーティション索引には常に複数のセグメントがあります。Oracle Databaseが1つの索引セグメントにプルーニングできない場合、データベースは常に複数のセグメントにアクセスする必要があります。これによってI/O要件が高まる可能性が潜在的にあり(非パーティション索引の1回のプローブに対して、*n*個の索引セグメントのプローブ)、実行時パフォーマンスに(測定可能または不可能にかかわらず)影響が出ることがあります。これはすべてのパーティション索引に当てはまります。

  パーティション索引は、ローカル・パーティション索引またはグローバル・パーティション索引のいずれかになります。ローカル・パーティション索引は、常に表のパーティション化キーを継承し、表パーティションとまったく同様に配置されます。結果として、あらゆる種類のパーティション・メンテナンス操作で、索引メンテナンス作業はほとんどまたはまったく必要ありません。たとえば、パーティションの削除または切捨てによって発生する索引メンテナンスのオーバーヘッドはほとんど目立ちません。このとき、ローカル索引パーティションは削除または切り捨てされます。

  表と一緒に配置されていないパーティション索引は、グローバル・パーティション索引と呼ばれます。ローカル索引とは異なり、表と索引パーティションの間に関係はありません。グローバル・パーティション索引は柔軟性が高く、効率のよいパーティション索引アクセスに最適なパーティション化キーを選択できます。パーティション・メンテナンス操作は、操作のタイプや索引のパーティション化キーによって異なりますが、通常はグローバル・パーティション索引の(すべてでないとしても)いくつかのパーティションに影響します。

- 状況によっては、1つの索引を複数のセグメントに分けることでパフォーマンスが向上することがあります。OLTP環境では、連番を利用して人工的なキーを作成することがごく一般的です。したがって、単調に増加するキー値を作成することになり、そのために多くの挿入プロセスが同じ索引ブロックを競合するようになります。グローバル・パーティション索引の導入(たとえば、キー列でのグローバル・ハッシュ・パーティション化の使用)により、この問題が緩和されます。たとえば、そのような索引1つに4つのハッシュ・パーティションがある場合は、データを挿入するために4つの索引セグメントがあるため、挿入プロセスに関してこれらのセグメントに対する同時実行が4分の1に減少します。

一意性の施行は、OLTP環境のための重要なデータベース機能です。一意性は、非パーティション索引でもパーティション索引でも施行できます。ただし、パーティション索引にはパーティションの自律性があるため、一意索引を実装するために次の要件を満たす必要があります。

- 非パーティション索引は、任意の列、または列の組合せに対して一意性を施行できます。非パーティション索引の動作は、非パーティション表についてもパーティション表についても同じです。
- パーティション索引の各パーティションは、自律型セグメントとみなされます。これらのセグメントの自律性を施行するには、一意キー定義のサブセットとしてパーティション化キー列を必ず含める必要があります。
    - グローバル・パーティション索引には、常に、索引列の少なくとも先頭列(パーティション・グローバル索引のパーティション化列)に接頭辞を付ける必要があります。

- 一意ローカル索引の一意キー定義のサブセットは、表のパーティション化キーであることが必要です。

例7-1は、orders_oltp表のorder_id列での一意索引の作成を示します。このOLTPアプリケーションのorder_idには連番が入力されます。この一意索引では、ハッシュ・パーティション化を使用して、単調に増加するorder_id値への競合を減らします。また、この一意キーが主キー制約を作成するために使用されます。

例7-1 一意索引および主キー制約の作成

```
CREATE UNIQUE INDEX orders_pk
  ON orders_oltp(order_id)
  GLOBAL PARTITION BY HASH (order_id)
  ( PARTITION p1 TABLESPACE tbs1
   , PARTITION p2 TABLESPACE tbs2
   , PARTITION p3 TABLESPACE tbs3
   , PARTITION p4 TABLESPACE tbs4
  ) NOLOGGING;

ALTER TABLE orders_oltp ADD CONSTRAINT orders_pk
  PRIMARY KEY (order_id)
  USING INDEX;
```

7.2.2 索引構成表でのパーティション化の使用方法

ワークロードが索引構成表の使用に適している場合は、索引構成表および2次索引でのパーティション化の使用方法を検討する必要があります。

索引構成表の2次索引をパーティション化するかどうかは、通常のヒープ表の索引と同じように検討してください。索引構成表をパーティション化できますが、パーティション化キーは主キーのサブセットであることが必要です。索引構成表をパーティション化する一般的な理由は競合の低減です。通常、これはハッシュ・パーティション化を使用して達成されます。

索引構成表をパーティション化するもう1つの理由は、主キー列に基づいて物理的にデータセットを分割できることです。たとえば、アプリケーション・ホスティング企業が、企業IDのリスト・パーティション化により顧客別にアプリケーション・インスタンスを物理的に分割することができます。このようなシナリオの問合せでは、索引のパーティション・プルーニングを利用して、索引スキャンの時間を短縮できます。索引構成表とパーティション化を使用するILMシナリオは、主キーの一部として日付列が必要なため、それほど一般的ではありません。

パーティション索引構成表の作成方法の詳細は、「パーティションの管理」を参照してください。

関連項目:

索引構成表の詳細は、『Oracle Database管理者ガイド』を参照してください

7.3 オンライン・トランザクション処理環境での管理性

パーティション化では、パフォーマンスの利点に加え、OLTP環境のラージ・オブジェクトに関して最適なデータ管理が実現できます。

Oracle Databaseでのすべてのパーティション・メンテナンス操作は、アトミック性に対応してグローバル索引やローカル索引のメンテナンスを含むように拡張できます。OLTP環境の24時間体制の可用性に影響を与えずに、あらゆるパーティション・メンテナンス操作の実行が可能です。

OLTPシステムのパーティション・メンテナンス操作は、ILMシナリオではよく行われます。このようなシナリオでは、レンジ・パーティ

ション表または時間隔パーティション表、あるいはレンジまたは時間隔と他の方法を組み合せたコンポジット・パーティション表が一般的です。

パーティション・メンテナンス操作を含む事例としては、アプリケーション・データの分割に関連するシナリオがあります。たとえば、ある小売企業が1つのスキーマ内の複数の支店について同じアプリケーションを実行します。支店の収益によって異なりますが、アプリケーション(別のパーティションとして)は効率のよい記憶域に格納されます。リスト・パーティション化またはリストとその他の方法を組み合せたコンポジット・パーティション化は、このような事例のパーティション化戦略として一般的です。

表のハッシュ・(サブ)パーティション化をOLTPシステムで使用すると、データ・ウェアハウス環境で達成できるのと同様のパフォーマンスの向上が得られます。日常的なOLTPワークロードの大半は、シリアルで実行される比較的小さな操作です。ただし、定期的なバッチ操作はパラレルで実行されることがあり、ハッシュ・パーティション化とサブパーティション化がパーティション・ワイズ結合にもたらす優れた分散のメリットを得られます。たとえば、月末の金利計算は、毎晩のバッチ期間内に完了するようにパラレルで実行する必要があります。

この項では、次の項目について説明します。

- ローカル索引があるパーティション表でのパーティション・メンテナンス操作の影響
- グローバル索引でのパーティション・メンテナンス操作の影響
- OLTP環境での一般的なパーティション・メンテナンス操作

パーティション化によるパフォーマンスの利点の詳細は、「可用性、管理性およびパフォーマンスのためのパーティション化」を参照してください。

7.3.1 ローカル索引があるパーティション表でのパーティション・メンテナンス操作の影響

パーティション・メンテナンス操作が発生すると、Oracle Databaseは、ONLINE MOVEの場合を除いて、DML操作の影響する表パーティションをロックします。

DROPまたはTRUNCATE操作の場合を除き、影響を受けるパーティションのデータは、すべてのSELECT操作で完全にアクセス可能です。ローカル索引は論理的に表(データ)のパーティションと対になっているため、パーティション・メンテナンス操作の際にメンテナンスを行う必要があるのは、影響を受ける表パーティションのローカル索引パーティションのみです。これで、索引メンテナンスの最適な処理が実現します。

たとえば、ALTER TABLE MOVE ONLINE機能を使用してハイエンド・ストレージ層から低コスト・ストレージ層に古くなったパーティションを移動しても、データと索引に対するSELECTおよびDML操作は常に可能です。必要な索引メンテナンスは、データの新しい物理位置を反映するための既存索引パーティションの更新です。古いパーティションをアーカイブした後で削除すると、ローカル索引パーティションも削除され、グローバル索引では削除されたパーティションの親がないエントリがマークされます。このとき、データ・ディクショナリのみに作用する、瞬間的なパーティション・メンテナンス操作が行われます。

7.3.2 グローバル索引でのパーティション・メンテナンス操作の影響

グローバル索引がパーティション表または非パーティション表に定義されている場合、常に、個別の表パーティションと索引の間に相関関係はありません。したがって、あらゆるパーティション・メンテナンス操作はすべてのグローバル索引またはグローバル索引パーティションに影響します。

ローカル索引を含む表のパーティションは、ONLINE MOVE操作を除いて、影響表パーティションに対するDML操作を防ぐためにロックされます。ただし、ローカル索引の索引メンテナンスとは異なり、すべてのグローバル索引に対してはDML操作もすべて可能です。OLTPシステムのオンライン可用性にも影響がありません。

概念と技術の面では、パーティション・メンテナンス操作に対するグローバル索引のメンテナンスは、グローバル索引メンテナンスが遅延するDROPおよびTRUNCATEを除いて、同じセマンティックのDML操作で必要になる索引メンテナンスと同じです。グローバル

索引の管理の詳細は、「グローバル・パーティション索引の管理」を参照してください。

たとえば、古いパーティションの削除(Drop)は、SQL DELETE文を使用して古いパーティションのすべてのレコードを削除することとセマンティックは同じです。DMLの場合、削除されるデータセットのすべての索引エントリは、通常の索引メンテナンス操作としてどのグローバル索引からも削除する必要があります。この操作は、SELECTおよびDML操作に関する索引の可用性には影響しません。

DROP PARTITIONも索引の可用性に影響しませんが、グローバル索引の可用性に影響を与えずに初期データの削除から必要な索引メンテナンスを分離できます。このシナリオでは、削除操作(Drop)が最適な方法です。従来のDELETE操作に伴うオーバーヘッドなしでデータが削除され、可用性を損わずにグローバル索引がメンテナンスされます。

7.3.3 OLTP環境での一般的なパーティション・メンテナンス操作

2つの一般的なパーティション・メンテナンス操作は、データの削除と、低コスト・ストレージ層デバイスへのデータの移動です。

- 古いデータの削除(パージ)
- 低コスト・ストレージ層デバイスへの古いパーティションの移動またはマージ

7.3.3.1 古いデータの削除(パージ)

DROPまたはTRUNCATE操作を使用し、パーティション化キーの基準に基づいて古くなったデータを削除します。

Drop操作ではデータとパーティション・メタデータが削除されます。TRUNCATE操作ではデータは削除されますがメタデータは保存されます。すべてのローカル索引パーティションはそれぞれ削除され、切り捨てされます。パーティション・グローバル索引または非パーティション・グローバル索引では非同期グローバル索引メンテナンスが行われます。このとき、SELECTおよびDML操作はすべて可能です。

次の例では、2006年1月よりも前のすべてのデータがorders_oltp表から削除されます。DROP文の一部としてUPDATE GLOBAL INDEXES文が実行されます。このため、グローバル索引はメンテナンス操作中も使用可能です。ローカル索引パーティションがある場合はこの操作で削除されます。

```
ALTER TABLE orders_oltp DROP PARTITION p_before_jan_2006
 UPDATE GLOBAL INDEXES;
```

7.3.3.2 低コスト・ストレージ層デバイスへの古いパーティションの移動またはマージ

情報ライフサイクル管理(ILM)戦略の一環としてMOVEまたはMERGE操作を使用して、古いパーティションをコスト効率が最も高いストレージ層に移動できます。

ALTER TABLE ONLINE MOVE機能を使用して、問合せおよびDML操作にデータを使用できます。ローカル索引はメンテナンスされます。マージまたは移動操作と同時に移動する場合があります。パーティション・グローバル索引または非パーティション・グローバル索引では通常の索引メンテナンスが行われます。このとき、SELECTおよびDML操作はすべて可能です。

次の例は、orders_oltp表の2006年1月と2006年2月のパーティションをマージして、別の表領域に格納する方法を示します。ローカル索引パーティションも、この操作でts_low_cost表領域に移動されます。UPDATE INDEXES句により、再構築を行わなくてもすべての索引が操作の間および後で使用可能であることが保証されます。

```
ALTER TABLE orders_oltp
  MERGE PARTITIONS p_2006_jan, p_2006_feb
  INTO PARTITION p_before_mar_2006 COMPRESS
  TABLESPACE ts_low_cost
  UPDATE INDEXES;
```

情報ライフサイクル管理でのパーティション・メンテナンス操作の利点の詳細は、「時間ベース情報の管理およびメンテナンス」を

参照してください。

8 パラレル実行の使用

パラレル実行とは、複数のプロセスの使用により、複数のCPUリソースおよびI/Oリソースを単一SQL文の実行に適用する機能です。

この章では、パラレル実行がどのように機能するかと、Oracle Databaseでのパラレル実行の制御、管理および監視の方法を説明します。

この章の構成は、次のとおりです。

- パラレル実行の概念
- 並列度の設定
- インメモリー・パラレル実行
- パラレル文のキューイング
- 並列処理の種類
- パラレル実行のためのパラメータの初期化とチューニングについて
- パラレル実行のパフォーマンスの監視
- パラレル実行のチューニングのヒント

関連項目:

Oracle Databaseでのパラレル実行の詳細は、http://www.oracle.com/technetwork/database/database-technologies/parallel-execution/overview/index.htmlを参照してください

8.1 パラレル実行の概念

パラレル実行では、複数のCPUリソースおよびI/Oリソースを1つのSQL文の実行に適用できます。

パラレル実行を使用すると、通常ディシジョン・サポート・システム(DSS)およびデータ・ウェアハウスに関連付けられているサイズの大きなデータベース上で、データ集中型の操作のレスポンス時間を大幅に削減できます。オンライン・トランザクション処理(OLTP)システム上で、索引の作成などのバッチ処理またはスキーマ・メンテナンス操作のためにパラレル実行を実装することもできます。

パラレル実行はパラレル化とも呼ばれます。パラレル化の概念は、タスクへの分解であり、これにより1つのプロセスで問合せに関するすべての処理を実行するのではなく、多くのプロセスが同時に各処理を実行します。たとえば、1年の合計売上げを4つのプロセスで計算するのに、1つのプロセスですべての四半期を処理するのではなく、各プロセスが1年の四半期それぞれを処理する場合です。これを使用するとパフォーマンスの大幅な向上が見込めます。

パラレル実行では、次のプロセスのパフォーマンスを向上できます。

- 大規模な表のスキャン、結合またはパーティション索引スキャンを必要とする問合せ
- 大規模な索引の作成
- マテリアライズド・ビューを含む大規模な表の作成
- バルク挿入、更新、マージおよび削除

この項では、次の項目について説明します。

8.1.1 パラレル実行を実装する場合

パラレル実行は、ハードウェア内のCPU機能およびI/O機能を活用することで、問合せの実行時間を短縮するために使用されます。

パラレル実行は、次の場合に、シリアル実行よりも適切な選択となります。

- 問合せで大きなデータ・セットが参照される。
- 同時並行性が低い。
- 経過時間が重要である。

パラレル実行では、多数のプロセスを一緒に処理して、SQL問合せなどの単一操作を実行できます。パラレル実行は、次のすべての特性を持つシステム上で有効です。

- 対称型マルチプロセッサ(SMP)、クラスタ、または大規模なパラレル・システム

- 十分なI/Oバンド幅
- 稼働中でないCPUまたは断続的に使用されているCPU(CPUの使用率が通常30%未満のシステムなど)
- ソート、ハッシュおよびI/Oバッファなどの追加のメモリー集中処理をサポートする十分なメモリー

システムでこれらの特徴が1つでも欠けていると、パラレル実行を使用してもパフォーマンスが大幅には改善されないことがあります。実際に、使用率の高すぎるシステムまたはI/O帯域幅が小さいシステムでは、パラレル実行によりシステム・パフォーマンスが低下する場合もあります。

パラレル実行のメリットは、DSSおよびデータ・ウェアハウス環境でわかります。OLTPシステムでも、バッチ処理やスキーマ・メンテナンス操作(索引の作成など)の際にはパラレル実行の利点が得られます。OLTPアプリケーションを特徴づける通常の単純なDMLまたはSELECT文では、パラレルで実行することによるメリットはありません。

8.1.2 パラレル実行を実装しない場合

シリアル実行は、1つのプロセスのみでSQL問合せなどの単一のデータベース操作を実行するという点で、パラレル実行とは異なります。

シリアル実行は、次の場合に、パラレル実行よりも適切な選択となります。

- 問合せで小さいデータ・セットが参照される。
- 同時並行性が高い。
- 効率が重要である。

通常、次のような場合にはパラレル実行は適していません。

- 標準的な問合せまたはトランザクションが非常に短い(数秒またはそれ以下)環境。

  これには、ほとんどのオンライン・トランザクション・システムが含まれます。パラレル実行はこのような環境では役立ちません。パラレル実行サーバーの調整に関連するコストが発生するためです。短時間のトランザクションの場合、この調整のコストが並列処理のメリットを上回ります。

- CPU、メモリーまたはI/Oリソースが大量に使用されている環境。

  パラレル実行は追加の使用可能なハードウェア・リソースを利用するように設計されています。そのようなリソースが使用できない場合、パラレル実行はなんのメリットももたらさず、パフォーマンスに悪影響を及ぼす可能性があります。

8.1.3 ハードウェアの基本要件

パラレル実行は、問合せに迅速に応じるために複数のCPUおよびディスクを効率よく使用するように設計されています。

本質的に高度なI/O集中型処理です。最適なパフォーマンスを達成するには、同一レベルのスループットを維持するように、ハードウェア構成の各コンポーネント(CPUや計算ノードのホスト・バス・アダプタ(HBA)からスイッチまで、また記憶域コントローラや物理ディスクなどのI/Oサブシステム)をサイズ設定する必要があります。システムがOracle Real Application Cluster (Oracle RAC)システムの場合、インターコネクトのサイズも適切に設定する必要があります。最も弱いリンクのために、構成における処理のパフォーマンスとスケーラビリティが制限されるためです。

Oracle Databaseを除いて、ハードウェア構成によって実現できる最大のI/Oパフォーマンスを測定することをお薦めします。将来のシステム・パフォーマンス評価の基礎としてこの測定結果を使用できます。パラレル実行で達成できるI/Oスループットが、基礎となるハードウェアによって実現可能なスループットを上回ることはありません。Oracle Databaseには、フリーの測定ツール、Orionが用意されています。このツールは、Oracle I/Oワークロードをシミュレーションしてシステムの I/Oパフォーマンスを測定します。通常、パラレル実行では大容量のランダムI/Oを行います。

関連項目:

I/O構成および設計の詳細は、『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

8.1.4 パラレル実行の仕組み

パラレル実行では、1つのプロセスで1つの問合せに関する処理のすべてを実行するのではなく、多数のプロセスで処理の部分部分を同時に実行するように、タスクが分割されます。

この項では、次の項目について説明します。

- SQL文のパラレル実行
- プロデューサ/コンシューマ・モデル
- 並列処理のグラニュル
- プロデューサとコンシューマの間の配分方法
- パラレル実行サーバーの通信方法

8.1.4.1 SQL文のパラレル実行

各SQL文では、解析の際に最適化およびパラレル化のプロセスが行われます。

文がパラレルで実行されることが決定された場合は、実行計画で次のステップが行われます。

1. ユーザー・セッションまたはシャドウ・プロセスが、コーディネータの役割を引き受けます。これは、問合せコーディネータ(QC)またはパラレル実行(PX)コーディネータと呼ばれることもあります。QCは、パラレルSQL文を開始するセッションです。
2. PXコーディネータは、パラレル実行(PX)サーバーというプロセスを必要な数だけ取得します。PXサーバーは、セッションを開始するかわりにパラレルで処理を実行する個々のプロセスです。
3. SQL文は、全表スキャンまたはORDER BY句などの、一連の操作として実行されます。各操作は、可能な場合はパラレルで実行されます。
4. PXサーバーで文の実行が完了すると、PXコーディネータで、パラレルで実行できない処理の部分が実行されます。たとえば、SUM()演算を含むパラレル問合せでは、各PXサーバーで計算された小計それぞれを合計する必要があります。
5. 最後に、PXコーディネータによって結果がユーザーに返されます。

8.1.4.2 プロデューサ/コンシューマ・モデル

パラレル実行では、プロデューサ/コンシューマ・モデルが使用されます。

パラレル実行計画は、一連のプロデューサ/コンシューマ操作として実行されます。後続操作のためにデータを生成するパラレル実行(PX)サーバーはプロデューサと呼ばれ、他の操作の出力を必要とするPXサーバーはコンシューマと呼ばれます。プロデューサまたはコンシューマ・パラレル操作はそれぞれ、PXサーバー・セットという一連のPXサーバーによって実行されます。PXサーバー・セット内のPXサーバーの数は、並列度(DOP)と呼ばれます。PXサーバー・セットの基本処理単位は、データ・フロー操作(DFO)と呼ばれます。

1つのPXコーディネータで複数のレベルのプロデューサ/コンシューマ操作(複数のDFO)が可能ですが、1つのPXコーディネータのためのPXサーバー・セットの数は、2つまでに制限されています。そのため、同時に2つのPXサーバー・セットのみを1つのPXコー

ディネータのためにアクティブにできます。結果として、1つのDFO内および複数DFO間の操作の両方に、並列処理が存在します。個々のDFOの並列処理はイントラ・オペレーション並列処理と呼ばれ、複数DFO間の並列処理はインター・オペレーション並列処理と呼ばれます。次の文に関して、イントラ・オペレーション並列化とインター・オペレーション並列化を説明します。

```
SELECT * FROM employees ORDER BY last_name;
```

実行計画により、employees表の全体スキャンが実装されます。この操作の後で、取得された行がlast_name列の値に基づいてソートされます。この例では、last_name列には索引がないとします。また、問合せのDOPが4に設定されているとします。これは、どの操作に対しても4つのパラレル実行サーバーが使用できることを意味します。

図8-1は、この例の問合せのパラレル実行を示しています。

図8-1 インター・オペレーション並列化と動的パーティション化

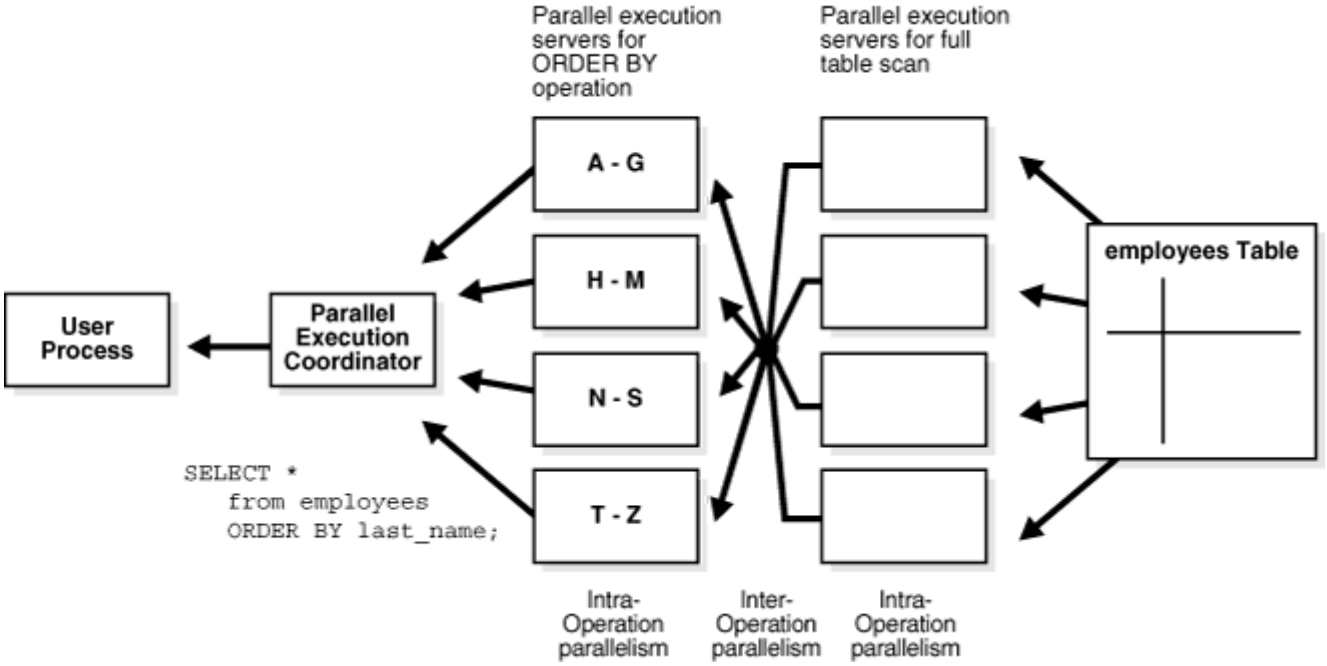


図8-1に示すように、この問合せのDOPは4ですが、実際は8つのPXサーバーが関係しています。これは、プロデューサ演算子とコンシューマ演算子が同時に実行できるためです(インター・オペレーション並列化)。

また、スキャン操作に関係するすべてのPXサーバーが、SORT操作を実行する適切なPXサーバーに行を送信します。PXサーバーでスキャンされる行のlast_name列の値がAからGの場合、その行は最初のORDER BYパラレル実行サーバーに送信されます。スキャン操作が完了すると、ソート・プロセスはソート結果を問合せコーディネータに返し、コーディネータが完全な問合せ結果をユーザーに返します。

8.1.4.3 並列処理のグラニュル

並列処理の基本作業ユニットはグラニュルと呼ばれます。

Oracle Databaseによって、表のスキャンや索引の作成などのパラレル化対象の操作がグラニュル単位に分割されます。パラレル実行(PX)サーバーは操作を1回に1グラニュルずつ実行します。グラニュルの数とサイズは並列度(DOP)と相関関係があります。グラニュルの数は、PXサーバー間で処理を適切に均衡できるかどうかにも影響します。

8.1.4.3.1 ブロック・レンジ・グラニュル

ブロック・レンジ・グラニュルは、ほとんどのパラレル操作の基本ユニットです。パーティション表の場合でも同様です。Oracle Databaseの観点では、並列度はパーティション数に関係しません。

ブロック・レンジ・グラニュルは、表の物理ブロックのレンジです。グラニュルの数とサイズは、関連するすべてのパラレル実行(PX)サーバーで処理の配分を最適化して均衡させるように、実行時にOracle Databaseによって計算されます。グラニュルの数と

サイズは、オブジェクトのサイズとDOPによって決まります。ブロック・レンジ・グラニュルは、表または索引の静的な事前割当てには影響を受けません。グラニュルの計算の際に、Oracle DatabaseはDOPを考慮に入れて、可能な場合には競合を避けるために、グラニュルをさまざまなデータ・ファイルから各PXサーバーに割り当てようとします。また、Oracle Databaseは、PXサーバーとディスクの物理的な近接性を利用するために、超並列処理(MPP)システム上のグラニュルのディスク・アフィニティを考慮します。

8.1.4.3.2 パーティション・グラニュル

パーティション・グラニュルが使用される場合、パラレル実行(PX)サーバーは、表または索引のパーティションまたはサブパーティション全体を処理します。

パーティション・グラニュルは表または索引の作成時の構造によって静的に決まるため、パーティション・グラニュルではブロック・グラニュルのように操作を柔軟にパラレル実行することはできません。使用可能な最大の並列度(DOP)はパーティション数です。このため、システムの使用率とPXサーバー間のロード・バランシングが制限されることがあります。

表または索引に対するパラレル・アクセスにパーティション・グラニュルが使用される場合は、比較的多数のパーティション(理想的にはDOPの3倍)を使用することをお薦めします。これによって、Oracle Databaseで複数のPXサーバーにわたり処理を効率よく均衡化できます。

パーティション・グラニュルは、パラレル索引レンジ・スキャン、2つの同一レベル・パーティション表の結合(問合せオプティマイザがパーティション・ワイズ結合を選択した場合)、およびパーティション・オブジェクトの複数パーティションを変更するパラレル操作の基本ユニットです。これらの操作には、パーティション索引のパラレル作成やパーティション表のパラレル作成も含まれます。

文の実行計画を調べることによって、どのタイプのグラニュルが使用されたかがわかります。表または索引アクセスの上の行PX BLOCK ITERATORは、ブロック・レンジ・グラニュルが使用されたことを示しています。次の例では、SALES表のTABLE FULL ACCESSのすぐ上の実行計画出力の7行目に示されています。

```
-------------------------------------------------------------------------------------------------------
|Id|      Operation          |  Name  |Rows|Bytes|Cost%CPU|  Time  |Pst|Pst|  TQ |INOUT|PQDistri|
-------------------------------------------------------------------------------------------------------
| 0|SELECT STATEMENT         |        |  17| 153 |565(100)|00:00:07|   |   |     |     |        |
| 1| PX COORDINATOR          |        |    |     |        |        |   |   |     |     |        |
| 2|  PX SEND QC(RANDOM)     |:TQ10001|  17| 153 |565(100)|00:00:07|   |   |Q1,01|P->S |QC(RAND)|
| 3|   HASH GROUP BY         |        |  17| 153 |565(100)|00:00:07|   |   |Q1,01|PCWP |        |
| 4|    PX RECEIVE           |        |  17| 153 |565(100)|00:00:07|   |   |Q1,01|PCWP |        |
| 5|     PX SEND HASH        |:TQ10000|  17| 153 |565(100)|00:00:07|   |   |Q1,00|P->P | HASH   |
| 6|      HASH GROUP BY      |        |  17| 153 |565(100)|00:00:07|   |   |Q1,00|PCWP |        |
| 7|       PX BLOCK ITERATOR |        | 10M| 85M | 60(97) |00:00:01| 1 | 16|Q1,00|PCWC |        |
|*8|        TABLE ACCESS FULL|  SALES | 10M| 85M | 60(97) |00:00:01| 1 | 16|Q1,00|PCWP |        |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
8 - filter("CUST_ID"<=22810 AND "CUST_ID">=22300)
```

パーティション・グラニュルが使用された場合、実行計画出力の表または索引アクセスの上に行PX PARTITION RANGEが表示されます。次の例では、この文が表内の16個のパーティションすべてにアクセスするため、計画の6行目にPX PARTITION RANGE ALLと示されています。すべてのパーティションにアクセスするのでない場合は、単にPX PARTITION RANGEと表示されます。

```
-------------------------------------------------------------------------------------------------------
---
|Id|      Operation                    |  Name    |Rows|Byte|Cost%CPU|  Time  |Ps|Ps|  TQ
|INOU|PQDistri|
-------------------------------------------------------------------------------------------------------
---
```

```
| 0|SELECT STATEMENT                    |          |  17| 153|   2(50)|00:00:01|  |  |     |    |
|
| 1| PX COORDINATOR                     |          |    |    |        |        |  |  |     |    |
|
| 2|  PX SEND QC(RANDOM)                |:TQ10001  |  17| 153|   2(50)|00:00:01|  |  |Q1,01|P-
>S|QC(RAND)|
| 3|   HASH GROUP BY                    |          |  17| 153|   2(50)|00:00:01|  |  |Q1,01|PCWP|
|
| 4|    PX RECEIVE                      |          |  26| 234|    1(0)|00:00:01|  |  |Q1,01|PCWP|
|
| 5|     PX SEND HASH                   |:TQ10000  |  26| 234|    1(0)|00:00:01|  |  |Q1,00|P->P| HASH
|
| 6|      PX PARTITION RANGE ALL        |          |  26| 234|    1(0)|00:00:01|  |  |Q1,00|PCWP|
|
| 7|       TABLEACCESSLOCAL INDEX ROWID|SALES     |  26| 234|    1(0)|00:00:01| 1|16|Q1,00|PCWC|
|
|*8|        INDEX RANGE SCAN            |SALES_CUST|  26|    |    1(0)|00:00:01| 1|16|Q1,00|PCWP|
|
---------------------------------------------------------------------------------------------------
---

Predicate Information (identified by operation id):
---------------------------------------------------
8 - access("CUST_ID"<=22810 AND "CUST_ID">=22300)
```

8.1.4.4 プロデューサとコンシューマの間の配分方法

配分方法とは、一方のパラレル実行(PX)サーバー・セットから他方へデータが送信(または再配分)される方法です。

次に、パラレル実行で最も一般的に使用される配分方法を示します。

- ハッシュ配分

  ハッシュ配分方法では、行内の1つ以上の列でハッシュ関数が使用され、それによってその後、プロデューサがその行を送信するコンシューマが決定されます。この配分では、ハッシュ値に基づいて複数コンシューマ間で等しく処理が分割されるよう試みられます。

- ブロードキャスト配分

  ブロードキャスト配分方法では、各プロデューサがすべての行をすべてのコンシューマに送信します。この方法は、ジョイン演算の左側の結果セットが小さく、すべての行をブロードキャストするコストが高くない場合に使用されます。この場合は、ジョインの右側の結果セットを配分する必要はありません。ジョイン演算に割り当てられたコンシューマPXサーバーで、右側のスキャンおよびジョインの実行が可能です。

- レンジ配分

  レンジ配分は、主にパラレル・ソート操作で使用されます。この方法では、各プロデューサが、ある範囲の値を含む行を同じコンシューマに送信します。これは、図8-1で使用されている方法です。

- ハイブリッド・ハッシュ配分

  ハイブリッド・ハッシュは、ジョイン演算で使用される適応配分方法です。実際の配分方法は、オプティマイザによって、実行時にジョインの左側の結果セットのサイズに応じて決定されます。左側から返される行の数はカウントされ、しきい値と照合されます。行の数がしきい値以下の場合は、ジョインの左側に対してブロードキャスト配分が使用されます。また、ジョイン演算に割り当てられた同じコンシューマPXサーバーが右側をスキャンしジョインを実行するため、右側は配分されません。左側から返された行の数がしきい値より大きい場合は、ジョインの両側にハッシュ配分が使用されます。

配分方法を決定するには、パラレル実行(PX)コーディネータでSQL文の実行計画内の各操作を検証してから、その操作に

よって影響を受ける行をPXサーバー間で再配分する方法を決定します。パラレル問合せの例として、例8-1の問合せを想定します。図8-2では、例8-1の問合せのデータ・フローおよび問合せ計画を示し、例8-2では、同じ問合せについて実行計画出力を示します。

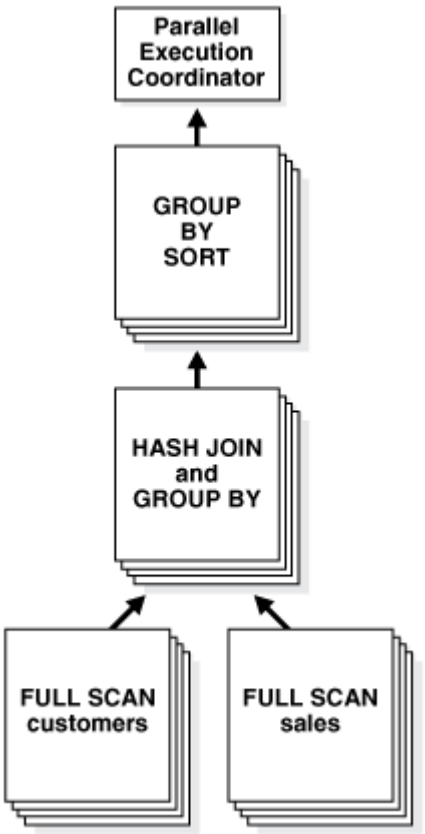
問合せ計画には、PXコーディネータによって適応配分方法が選択されたことが示されています。実行時にオプティマイザでハッシュ配分が選択されるとすると、実行は次のように進行します。SS1およびSS2という、PXサーバーの2つのセットが問合せのために割り当てられ、文のDOPを指定するPARALLELヒントにより、各サーバー・セットに4つのPXサーバーがあります。

PXセットSS1は、最初に表customersをスキャンし、行をSS2に送信します。それにより、それらの行に対してハッシュ表が作成されます。つまり、SS2のコンシューマとSS1のプロデューサは同時に働きます。一方はcustomersをパラレルでスキャンし、もう一方は行を受け取って、パラレルでハッシュ結合を実行できるようにハッシュ表を構築します。インター・オペレーション並列処理の例を次に示します。

SS1内のPXサーバー・プロセスでcustomers表の行がスキャンされた後、SS2内のどのPXサーバー・プロセスにそれが送信されるでしょうか。この例では、customersのパラレル・スキャンを実行するSS1から、パラレル・ハッシュ結合を実行するSS2に送られる行の再配分は、結合列に対するハッシュ配分によって行われます。つまり、customersをスキャンしているPXサーバー・プロセスが、列customers.cust_idの値でハッシュ関数を計算して、送り先となるSS2内のPXサーバー・プロセスを決定します。使用される再配分方法は、問合せのEXPLAIN PLANのDistrib列に明示的に示されます。図8-2では、これはEXPLAIN PLANの5、9および14行目に示されています。

SS1はcustomers表全体のスキャンを終了すると、sales表をパラレルでスキャンします。その行をSS2内のPXサーバーに送り、それがその後プローブを実行してハッシュ結合をパラレルで完了します。これらのPXサーバーは、結合後にGROUP BY操作も実行します。SS1は、パラレルでsales表をスキャンし、行をSS2に送信すると、パラレルでの最後のgroup by操作の実行に切り替わります。この時点で、SS2内のPXサーバーは、group by操作のために、ハッシュ配分を使用してそれらの行をSS1内のPXサーバーに送信します。これは、2つのサーバー・セットを同時に実行して、問合せツリーの様々な演算子に対してインター・オペレーション並列化を実現する方法です。

図8-2 表結合のためのデータ・フロー図



例8-1 CustomersおよびSalesに対する問合せの実行計画の実行

```
EXPLAIN PLAN FOR
SELECT /*+ PARALLEL(4) */ customers.cust_first_name, customers.cust_last_name,
  MAX(QUANTITY_SOLD), AVG(QUANTITY_SOLD)
  FROM sales, customers
  WHERE sales.cust_id=customers.cust_id
  GROUP BY customers.cust_first_name, customers.cust_last_name;

Explained.
```

例8-2 CustomersおよびSalesに対する問合せの実行計画出力

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
--------------------------------
Plan hash value: 3260900439
------------------------------------------------------------------------------------------------------------------------
--------------------------------
|Id  |Operation                      |Name     |Rows  | Bytes |TempSpc|Cost (%CPU)| Time
|Pstart|Pstop |   TQ  |IN-OUT|PQ Distrib |
------------------------------------------------------------------------------------------------------------------------
--------------------------------
|  0 |SELECT STATEMENT               |         |  960 | 26880 |       |    6  (34)| 00:00:01 |      |
|       |      |           |
|  1 | PX COORDINATOR                |         |      |       |       |           |          |      |
|       |      |           |
|  2 |  PX SEND QC (RANDOM)          |:TQ10003 |  960 | 26880 |       |    6  (34)| 00:00:01 |      |
| Q1,03 | P->S |QC (RAND)  |
|  3 |   HASH GROUP BY               |         |  960 | 26880 | 50000 |    6  (34)| 00:00:01 |      |
| Q1,03 | PCWP |           |
|  4 |    PX RECEIVE                 |         |  960 | 26880 |       |    6  (34)| 00:00:01 |      |
| Q1,03 | PCWP |           |
|  5 |     PX SEND HASH              |:TQ10002 |  960 | 26880 |       |    6  (34)| 00:00:01 |      |
| Q1,02 | P->P |HASH       |
|  6 |      HASH GROUP BY            |         |  960 | 26880 | 50000 |    6  (34)| 00:00:01 |      |
| Q1,02 | PCWP |           |
|* 7 |       HASH JOIN               |         |  960 | 26880 |       |    5  (20)| 00:00:01 |      |
| Q1,02 | PCWP |           |
|  8 |        PX RECEIVE             |         |  630 | 12600 |       |    2   (0)| 00:00:01 |      |
| Q1,02 | PCWP |           |
|  9 |         PX SEND HYBRID HASH   |:TQ10000 |  630 | 12600 |       |    2   (0)| 00:00:01 |      |
| Q1,00 | P->P |HYBRID HASH|
| 10 |          STATISTICS COLLECTOR |         |      |       |       |           |          |      |
| Q1,00 | PCWC |           |
| 11 |           PX BLOCK ITERATOR   |         |  630 | 12600 |       |    2   (0)| 00:00:01 |      |
| Q1,00 | PCWC |           |
| 12 |            TABLE ACCESS FULL  |CUSTOMERS|  630 | 12600 |       |    2   (0)| 00:00:01 |      |
| Q1,00 | PCWP |           |
| 13 |        PX RECEIVE             |         |  960 |  7680 |       |    2   (0)| 00:00:01 |      |
| Q1,02 | PCWP |           |
| 14 |         PX SEND HYBRID HASH   |:TQ10001 |  960 |  7680 |       |    2   (0)| 00:00:01 |      |
| Q1,01 | P->P |HYBRID HASH|
| 15 |          PX BLOCK ITERATOR    |         |  960 |  7680 |       |    2   (0)| 00:00:01 |    1 |
16 | Q1,01 | PCWC |           |
| 16 |           TABLE ACCESS FULL   |SALES    |  960 |  7680 |       |    2   (0)| 00:00:01 |    1 |
16 | Q1,01 | PCWP |           |
------------------------------------------------------------------------------------------------------------------------
--------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
7 - access("SALES"."CUST_ID"="CUSTOMERS"."CUST_ID")
```

```
Note
-----
   - Degree of Parallelism is 4 because of hint
```

8.1.4.5 パラレル実行サーバーの通信方法

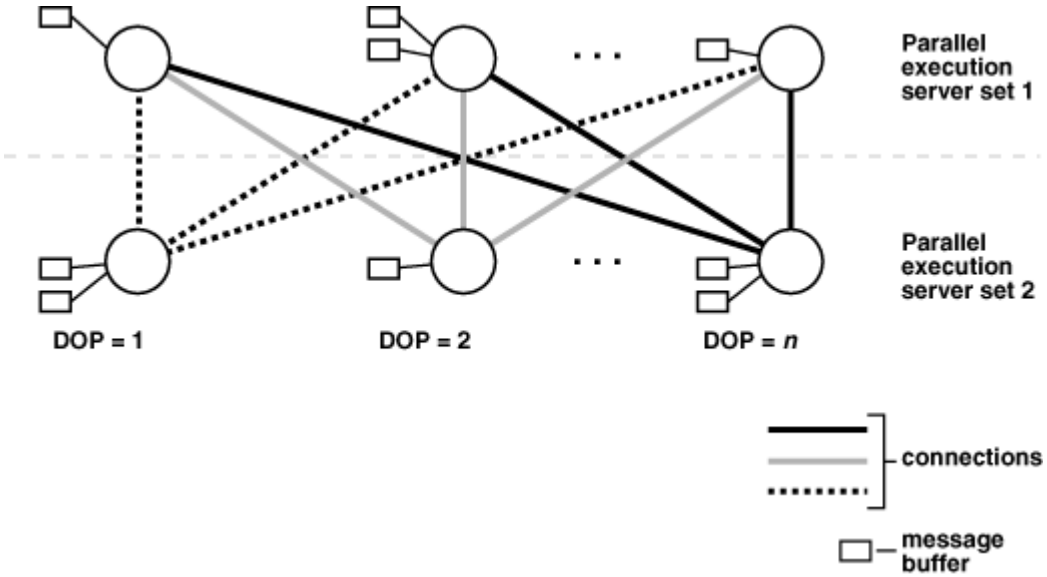
問合せをパラレルで実行するために、通常、Oracle Databaseはプロデューサ・パラレル実行サーバー・セットとコンシューマ・パラレル実行サーバー・セットを作成します。

プロデューサ・サーバーは表から行を取得し、コンシューマ・サーバーはそれらの行に対して結合、ソート、DML、DDLなどの操作を実行します。プロデューサ・セットの各サーバーは、コンシューマ・セットの各サーバーに接続しています。パラレル実行サーバー間の仮想接続数は並列度の2乗で増加します。

各通信チャネルには、少なくとも1つ、最大で4つのメモリー・バッファがあり、共有プールから割り当てられます。複数のメモリー・バッファを使用すると、パラレル実行サーバー間の非同期通信が促進されます。

シングル・インスタンス環境では、各通信チャネルで最大3つのバッファを使用します。Oracle Real Application Clusters環境では、各チャネルで最大4つのバッファを使用します。図8-3は、メッセージ・バッファと、プロデューサ・パラレル実行サーバーがコンシューマ・パラレル実行サーバーにどのように接続するかを示します。

図8-3 パラレル実行サーバーの接続とバッファ



同一インスタンスで2つのプロセス間に接続が存在するとき、サーバーはメモリー内で(共有プールで)バッファを受け渡すことにより通信を行います。異なるインスタンスのプロセス間に接続が存在するとき、メッセージはインターコネクトを介して外部の高速ネットワーク・プロトコルを使用して送信されます。図8-3では、DOPはパラレル実行サーバーの数と同じです(この場合はn)。図8-3にはパラレル実行コーディネータは示されていません。実際には各パラレル実行サーバーはパラレル実行コーディネータとも接続しています。パラレル実行を使用する際は、共有プールのサイズを適切に設定することが重要です。共有プールに、パラレル・サーバーに必要なメモリー・バッファを割り当てるための十分な空き領域がない場合、開始することができません。

8.1.5 パラレル実行サーバーのプール

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。

初期化パラメータPARALLEL_MIN_SERVERSによって、Oracle Databaseがインスタンス起動時に作成するパラレル実行サーバーの数が指定されます。

パラレル操作を実行するとき、パラレル実行コーディネータは、パラレル実行サーバーをプールから獲得して操作に割り当てます。

必要であれば、Oracle Databaseは操作のためにパラレル実行サーバーを追加作成することもできます。これらのパラレル実行サーバーは、実行の間は操作とともにあります。文が処理されると、パラレル実行サーバーはプールに戻ります。

パラレル操作の数が増えると、着信リクエストを扱うためにOracle Databaseによって追加のパラレル実行サーバーが作成されます。ただし、初期化パラメータPARALLEL_MAX_SERVERSで指定された値を超えるパラレル実行サーバーが、1つのインスタンスに対して作成されることはありません。

パラレル操作の数が減ると、一定期間アイドル状態になっていたパラレル実行サーバーがOracle Databaseによって停止されます。パラレル実行サーバーのアイドル状態が長く続いても、プールのサイズがPARALLEL_MIN_SERVERSの値よりも小さくなることはありません。

8.1.5.1 十分なパラレル実行サーバーなしでの処理

Oracle Databaseでは、リクエストよりも少ないプロセス数でパラレル操作を処理することができます。

プールのすべてのパラレル実行サーバーが占有され、最大数のパラレル実行サーバーが起動されている場合、パラレル実行コーディネータはシリアル処理に切り替えます。

関連項目:

- PARALLEL_MIN_PERCENTおよびPARALLEL_MAX_SERVERSの詳細は、「パラレル実行のための一般的なパラメータのチューニング」を参照してください
- 初期化パラメータPARALLEL_MIN_PERCENTの使用の詳細は、『Oracle Databaseリファレンス』を参照してください

8.1.6 パフォーマンスを最適化するためのワークロードのバランシング

パフォーマンスを最適化するには、すべてのパラレル実行サーバーのワークロードを均等にする必要があります。

ブロック・レンジまたはパラレル実行サーバーによってパラレルで実行されるSQL文では、ワークロードはパラレル実行サーバー間で動的に分割されます。この方法では、一部のパラレル実行サーバーで実行する作業が他のプロセスよりも大幅に多くなる、ワークロードの偏りが最小限に抑えられます。

パーティション単位でパラレル実行される比較的少数のSQL文では、ワークロードがパーティションで均等に分散されていれば、パラレル実行サーバーの数とパーティションの数を一致させるか、パーティション数がプロセス数の倍数になるようにDOPを選択することで、パフォーマンスを最適化できます。これは、Oracle9*i*よりも前に作成された表に対するパーティション・ワイズ結合とパラレルDMLに適用されます。詳細は、「並列度の制限」を参照してください。

たとえば、表に16個のパーティションがあり、パラレル操作の処理がそれらのパーティション間で均等に分割されるとします。16個のパラレル実行サーバー(DOPが16)を使用すると、1プロセスの場合のおよそ10分の1の時間で作業を行うことができます。また、5プロセスを使用すると5分の1の時間、2プロセスを使用すると2分の1の時間になります。

ただし、16個のパーティションで作業を行うために15個のプロセスを使用した場合は、1つのパーティションの作業を最初に終了したプロセスが16番目のパーティションの作業を開始し、それ以外のプロセスは作業を終了してアイドルになります。この構成では、作業をパーティション間で均等に分割しても、優れたパフォーマンスは得られません。作業の分割が不均等な場合、最後に残ったパーティションの作業が他のパーティションに比べて多いか少ないかによってパフォーマンスが変わります。

同様に、6個のプロセスを使用して16個のパーティションを処理し、作業を均等に分割するとします。この場合、各プロセスは最初のパーティションを終了してから2番目のパーティションを処理しますが、3番目のパーティションを処理するのは4つのプロセスのみで、残りの2つはアイドルになります。

一般的に、P個のパラレル実行サーバーを使用したN個のパーティションに対するパラレル操作の実行時間がN/Pになると想定することはできません。この計算式は、最後のパーティションを処理する間に待機する必要のあるプロセスが存在する可能性を考慮に入れていません。ただし、適切なDOPを選択すると、ワークロードの偏りを最小にしてパフォーマンスを最適化することができます。

8.1.7 複数のパラレライザ

実行計画内の各パラレル実行(PX)コーディネータは、パラレライザと呼ばれます。

SQL文によって使用されるPXサーバーの数は、文の並列度(DOP)、およびパラレライザの数によって決定されます。1つのパラレライザのためのPXサーバー・セットの数は2つまでに制限されているため、ほとんどの文のPXサーバーの数はDOP*2となります。一部の文では、複数のパラレライザを使用できます。各パラレライザで2つのPXサーバー・セットを使用できるため、これらの文のためのPXサーバーの数はDOP*2より多くできます。EXPLAIN PLANを調べることで、これらの文を識別できます。計画に複数のPXコーディネータがある場合は、文に複数のパラレライザがあることを意味します。

SQL文で複数のパラレライザが使用される数少ない例としては、副問合せファクタ、グルーピング・セット、スター・クエリー、インメモリー集計、および無相関な副問合せがあります。

1つのSQL文の複数のパラレライザは、同時に、または実行計画に従って順々にアクティブにできます。

パラレライザが1つある文は、必要な数のPXサーバーを実行の開始時に割り当て、これらの割り当てられたPXサーバーを、文の完了まで解放することなく保持します。これにより、実行の間のPXサーバーの数が必ず一定になります。パラレライザが複数ある文は、各パラレライザの開始時にPXサーバーを割り当てるため異なります。パラレライザは実行中の様々な時点で開始できるため、システム内の使用可能プロセスの数に基づいて異なる数のPXサーバーを使用して各パラレライザを実行できます。

複数のパラレライザが同時に実行される場合は、文でDOP*2より多くのPXサーバーを使用できます。

ビューV$PQ_SESSTATでは、STATISTIC列にパラレライザの数が示されます。データ・フロー操作統計DFO Treesには、パラレライザの数が示されます。Server Threads統計には、1つのSQL文のために同時に使用されたPXサーバーの最大数が示されます。

関連項目:

V$PQ_SESSTATおよびその他の動的ビューの詳細は、『Oracle Databaseリファレンス』を参照してください。

8.1.8 Oracle RACでのパラレル実行

デフォルトでは、Oracle RAC環境で、パラレルで実行されるSQL文はクラスタ内のすべてのノードで実行できます。

このクロスノードまたはノード間パラレル実行を実現するには、ノード間パラレル実行によってインターコネクト・トラフィックが増大する可能性があるため、Oracle RAC環境でのインターコネクトのサイズが適切である必要があります。ノード間パラレル実行は、インターコネクトのサイズが小さいと対応できません。

使用可能なインスタンス数の制限

Oracle RAC環境では、サービスを使用して、パラレルSQL文の実行に参加するインスタンスの数を制限できます。デフォルトのサービスには使用可能なすべてのインスタンスが含まれます。それぞれが1つ以上のインスタンスを含むサービスを必要な数のみ作成できます。ユーザーがサービスを使用してデータベースに接続する場合は、そのサービスのメンバーであるインスタンス上のPXサーバーのみがパラレル文の実行に参加できます。

パラレル実行を単一ノードに限定するには、PARALLEL_FORCE_LOCAL初期化パラメータをTRUEに設定します。この場合は、セッションが接続するインスタンス上のPXサーバーのみが、そのセッションからのパラレル文の実行に使用されます。このパラメータが

TRUEに設定されている場合は、セッションがインスタンスに直接接続するかサービスを使用して接続するかに関係なく、そのインスタンス上で実行されているすべてのパラレル文がローカルで実行されることに注意してください。

フレックス・クラスタでのパラレル実行

フレックス・クラスタ上で実行されるパラレル文は、ハブおよびリーフ・ノードの両方で使用できます。ユーザー・セッションではハブ・ノードへの接続のみが許可されているため、コーディネータ・プロセス(問合せコーディネータまたはPXコーディネータ)はハブ・ノード上に存在し、クラスタ内の任意のノードからPXサーバー・プロセスを使用できます。パラレル問合せでは、任意のノード上の任意のPXサーバーが文の実行に参加できます。パラレルDML操作では、ハブ・ノードのみがDML操作の実行を許可されているため、ハブ・ノード上のPXサーバーのみが文のDML部分の実行に参加できます。

DML操作のためにリーフ・ノードからハブ・ノードへのデータ配分が存在する場合は、実行計画がこの配分を示します。次の例では、行Id 3にあるロード操作がハブ・ノード上でのみ実行されることを示し、行Id 5でデータがハブ・ノードに配分されます。

```
-----------------------------------------------------
| Id  | Operation                          | Name     |
-----------------------------------------------------
|   0 | CREATE TABLE STATEMENT             |          |
|   1 |  PX COORDINATOR                    |          |
|   2 |   PX SEND QC (RANDOM)              | :TQ10001 |
|   3 |    LOAD AS SELECT (HYBRID TSM/HWMB)| SALESTEMP|
|   4 |     PX RECEIVE                     |          |
|   5 |      PX SEND ROUND-ROBIN (HUB)     | :TQ10000 |
|   6 |       PX BLOCK ITERATOR            |          |
|   7 |        TABLE ACCESS FULL           | SALES    |
-----------------------------------------------------
```

関連項目:

- ハブ、リーフおよびフレックス・クラスタ・アーキテクチャの詳細は、『Oracle Clusterware管理およびデプロイメント・ガイド』を参照してください
- Grid Infrastructureのクラスタ・インストール・オプションの詳細は、『Oracle Grid Infrastructureインストレーションおよびアップグレード・ガイドfor Linux』を参照してください
- インスタンス・グループの詳細は、『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

8.2 並列度の設定

並列度(DOP)は、単一の処理に対応付けられるパラレル実行サーバーの数で表されます。

パラレル実行は複数のCPUを効率よく使用するためのものです。Oracle Databaseのパラレル実行フレームワークでは、特定の並列度をユーザーが明示的に選択することも、Oracle Databaseによって自動的に並列度を制御することもできます。

この項では、次の項目について説明します。

- 手動での並列度の指定
- デフォルトの並列度
- 自動並列度
- 自動DOPでの並列度の決定
- 自動並列度の制御
- 問合せ調整並列処理

8.2.1 手動での並列度の指定

表および索引に対して、Oracle Databaseで特定の並列度(DOP)をリクエストできます。

たとえば、次のように表レベルで固定DOPを設定できます。

```
ALTER TABLE sales PARALLEL 8;
ALTER TABLE customers PARALLEL 4;
```

この例では、sales表のみにアクセスする問合せはDOPである8をリクエストし、customers表にアクセスする問合せはDOPとして4をリクエストします。sales表とcustomers表の両方にアクセスする問合せは、DOPが8として処理され、16個のパラレル実行サーバーを割り当てる可能性があります(プロデューサ/コンシューマ・モデルのため)。Oracle Databaseでは、異なるDOPが指定されるときは必ず、高いほうのDOPが使用されます。

文レベルまたはオブジェクト・レベルのパラレル・ヒントを使用することで、特定のDOPをリクエストすることもできます。

表または索引のPARALLEL句で指定されたDOPは、PARALLEL_DEGREE_POLICYがMANUALまたはLIMITEDに設定されている場合のみ有効となります。

文の実際の実行時のDOPは、Oracle Database Resource Managerで制限できます。

関連項目:

- 並列処理のヒントの詳細は、『Oracle Database SQL言語リファレンス』を参照してください
- Oracle Database Resource Managerの詳細は、『Oracle Database管理者ガイド』を参照してください。

8.2.2 デフォルトの並列度

PARALLEL句が指定されているが並列度(DOP)の指定がない場合、オブジェクトのDOPはデフォルトになります。

たとえば、次のSQL文により、表をデフォルトのDOPに設定できます。

```
ALTER TABLE sales PARALLEL;
```

デフォルトの並列処理では、次に示すように、計算式が使用され、システム構成に基づいてDOPが決定されます。

- シングル・インスタンスの場合、DOP = PARALLEL_THREADS_PER_CPU x CPU_COUNT
- Oracle RAC構成の場合、DOP = PARALLEL_THREADS_PER_CPU x sum(CPU_COUNT)

デフォルトでは、sum(CPU_COUNT)は、クラスタ内のCPUの合計数です。ただし、Oracle RACサービスを使用して1つのパラレル操作で実行可能なノードの数を制限している場合は、sum(CPU_COUNT)は、そのサービスに属するすべてのノードにわたるCPUの合計数になります。たとえば、ノード4つのOracle RACクラスタで、各ノードに8個のCPUコアがあり、Oracle RACサービスがない場合、デフォルトのDOPは2 x (8+8+8+8) = 64になります。

文レベルまたはオブジェクト・レベルのパラレル・ヒントを使用することで、デフォルトのDOPをリクエストすることもできます。

表または索引のPARALLEL句で指定されたデフォルトDOPは、PARALLEL_DEGREE_POLICYがMANUALに設定されている場合のみ有効となります。

デフォルトのDOPアルゴリズムは、最大限のリソースを使用するように設計されており、操作で使用できるリソースが多いほど、操作は速く終了すると想定しています。デフォルトDOPは、単一ユーザーのワークロードが対象となるため、複数ユーザーの環境ではお薦めできません。

SQL文の実際の実行時のDOPは、Oracle Database Resource Managerで制限できます。

関連項目:

- 並列処理のヒントの詳細は、『Oracle Database SQL言語リファレンス』を参照してください
- Oracle Database Resource Managerの詳細は、『Oracle Database管理者ガイド』を参照してください。

8.2.3 自動並列度

自動並列度(自動DOP)により、Oracle Databaseで、文をパラレルで実行する必要があるかどうかと、使用する必要があるDOPを自動的に決定できます。

次に、自動DOPが有効になっている場合のパラレル文処理の概要を示します。

1. SQL文が発行されます。
2. 文が解析され、オプティマイザが実行計画を決定します。
3. 初期化パラメータPARALLEL_MIN_TIME_THRESHOLDによって指定されたしきい値がチェックされます。
    a. 予想される実行時間がしきい値より短い場合は、SQL文はシリアルで実行されます。
    b. 予想される実行時間がしきい値より長い場合は、定義されたリソース制限の考慮を含むオプティマイザが計算するDOPに基づいて文はパラレルで実行されます。

8.2.4 自動DOPでの並列度の決定

自動並列度(DOP)の使用時は、オプティマイザは、文のリソース要件に基づいて文のDOPを自動的に決定します。

オプティマイザは、全表スキャンや索引の高速全スキャンなど、実行計画内のすべてのスキャン操作のコスト、およびすべての操作のCPU操作のコストを使用して、必要なDOPを決定します。

ただし、オプティマイザは、実際の最大DOPの上限を設定して、多数のパラレル実行サーバーによってシステムの障害が発生しないようにします。この上限値は、パラメータPARALLEL_DEGREE_LIMITで設定されます。このパラメータのデフォルト値はCPUです。

これは、デフォルトDOPとも呼ばれ、DOPがシステム上のCPUの数(PARALLEL_THREADS_PER_CPU * sum(CPU_COUNT))によって制限されることを意味します。このデフォルトのDOPにより、1つのユーザー操作でシステムの障害が発生しないようにします。このパラメータの設定を調整することによって、オプティマイザがSQL文に対して選択できるDOPの最大値を制御できます。オプティマイザでは、DOPの制限にOracle Database Resource Managerが使用される場合に選択できる最大DOPを、さらに制限できます。

ノート:

PARALLEL_DEGREE_LIMIT の値 AUTO には、値 CPU と同じ機能があります。

SQL文の処理のコストを計算するために、自動DOPでは、システムのハードウェア特性に関する情報が使用されます。ハードウェア特性にはI/Oキャリブレーション統計が含まれるため、これらの統計を収集する必要があります。

必要な統計を収集するようI/Oキャリブレーションが実行されていない場合、デフォルトのキャリブレーション値を使用して操作コストおよびDOPが計算されます。

I/Oキャリブレーション統計は、PL/SQL DBMS_RESOURCE_MANAGER.CALIBRATE_IOプロシージャを使用して収集できます。I/Oキャリブレーションは、ハードウェアの物理的な交換を行わないかぎり、1回のみの処理です。

オプティマイザによって決定されたDOPは、次の実行計画出力で示すように、実行計画出力のNoteセクションに示され、実行計画文またはV$SQL_PLANのどちらかを使用して表示できます。

```
EXPLAIN PLAN FOR
SELECT SUM(AMOUNT_SOLD) FROM SH.SALES;

PLAN TABLE OUTPUT
Plan hash value: 1763145153

-------------------------------------------------------------------------------------------------------------
----------------
|Id| Operation              | Name     | Rows | Bytes| Cost (%CPU)| Time     | Pstart| Pstop|    TQ
|IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------------
-----------------
| 0| SELECT STATEMENT       |          |    1|     4|     2   (0)| 00:00:01 |       |       |      |
|          |
| 1|  SORT AGGREGATE        |          |    1|     4|            |          |       |      |       |
|          |
| 2|   PX COORDINATOR       |          |     |      |            |          |       |      |       |
|          |
| 3|    PX SEND QC (RANDOM) | :TQ10000 |    1|     4|            |          |       |      | Q1,00|
P->S | QC (RAND) |
| 4|     SORT AGGREGATE     |          |    1|     4|            |          |       |      | Q1,00|
PCWP |          |
| 5|      PX BLOCK ITERATOR |          |   960|  3840|     2   (0)| 00:00:01 |     1 |    16| Q1,00|
PCWC |          |
| 6|       TABLE ACCESS FULL| SALES    |   960|  3840|     2   (0)| 00:00:01 |     1 |    16| Q1,00|
PCWP |          |
-------------------------------------------------------------------------------------------------------------
-----------------
Note
-----
   - automatic DOP: Computed Degree of Parallelism is 4
```

関連項目:

DBMS_RESOURCE_MANAGER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

8.2.5 自動並列度の制御

自動並列度(自動DOP)を制御する初期化パラメータがあります。これらの初期化パラメータには、PARALLEL_DEGREE_POLICY、PARALLEL_DEGREE_LIMIT、PARALLEL_MIN_TIME_THRESHOLD、PARALLEL_MIN_DEGREEなどがあります。

初期化パラメータPARALLEL_DEGREE_POLICYは、自動DOP、パラレル文のキューイング、およびインメモリー・パラレル実行を有効にするかどうかを制御します。このパラメータには、次の値を指定できます。

- MANUAL

  この設定では、自動DOP、パラレル文のキューイング、およびインメモリー・パラレル実行が無効になります。パラレル実行の動作をOracle Database 11gリリース2 (11.2)よりも前の設定に戻します。これがデフォルトです。

  PARALLEL_DEGREE_POLICYのデフォルト設定MANUALの使用時は、DOPがオブジェクトに明示的に設定されている場合、またはパラレル・ヒントがSQL文に指定されている場合にかぎり、パラレル実行が使用されます。使用されるDOPは、指定されたとおりのものになります。パラレル文のキューイング、およびインメモリー・パラレル実行は起こりません。

- LIMITED

  この設定では、一部の文については自動DOPが有効になりますが、パラレル文のキューイングおよびインメモリー・パラレル実行は無効になります。自動DOPは、明示的にDOPを指定することなく、PARALLEL句で明示的に宣言されている表または索引にアクセスする文に対してのみ適用されます。特定のDOPが指定された表および索引は、その明示的なDOP設定を使用します。

  Oracle Databaseでオブジェクトの特定のサブセットにアクセスするSQL文のサブセットに対してのみDOPを自動的に決定するようにする場合は、PARALLEL_DEGREE_POLICYをLIMITEDに設定し、明示的なDOPを指定せずに、そのオブジェクト・サブセットにパラレル・プロパティを設定します。

- AUTO

  この設定では、すべての文について自動DOPが有効になり、パラレル文のキューイングおよびインメモリー・パラレル実行も有効になります。

  Oracle DatabaseですべてのSQL文のDOPを自動的に決定するようにする場合は、PARALLEL_DEGREE_POLICYをAUTOに設定します。

- ADAPTIVE

  この設定では、AUTO値と同様に、自動DOP、パラレル文のキューイング、およびインメモリー・パラレル実行が有効になります。さらに、パフォーマンス・フィードバックを使用可能にします。

PARALLEL_DEGREE_LIMIT初期化パラメータは、システム全体にわたり自動DOPで使用できる、最大DOPを指定します。最大DOPの非常に細かい制御には、Oracle Database Resource Managerを使用できます。

PARALLEL_MIN_TIME_THRESHOLD初期化パラメータは、文を自動DOPの対象とするための最小推定実行時間を指定します。最初に、オプティマイザで、SQL文のシリアル実行計画が計算されます。推定された実行時間がPARALLEL_MIN_TIME_THRESHOLDの値より長い場合、その文は自動DOPの候補となります。

PARALLEL_MIN_ DEGREE初期化パラメータは、自動並列度によって計算された最小並列度を制御します。ただし、PARALLEL_MIN_DEGREEの値がCPU_COUNTの値より大きい場合、またはオブジェクトがOracle所有のもの(ディクショナリ表やディクショナリ表に対して作成されたビューなど)である場合、PARALLEL_MIN_DEGREEの影響はありません。

文レベルまたはオブジェクト・レベルの適切なSQLヒントを指定することで、自動DOPをリクエストすることもできます。

関連項目:

- 自動DOPの詳細は、「自動並列度」を参照してください
- パラレル文のキューイングの詳細は、「パラレル文のキューイング」を参照してください
- インメモリー・パラレル実行の詳細は、「インメモリー・パラレル実行」を参照してください
- 並列性の制御に使用できる他の技術の詳細は、「パラレル実行のチューニングのヒント」を参照してください
- PARALLEL_DEGREE_POLICY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
- PARALLELヒントの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

8.2.6 問合せ調整並列処理

マルチユーザー問合せ調整アルゴリズムでは、システムの負荷が増加すると並列度が低下します。

Oracle Databaseの問合せ調整並列処理を使用すると、データベースはSQL実行時にアルゴリズムを使用して、パラレル操作がリクエストどおりのDOPを受け取るか、低いDOPに下げてシステムのオーバーロードを回避するかを決定します。

高いDOPを使用してパラレル実行を積極的に利用するシステムでは、問合せ調整アルゴリズムによってDOPが下がるのは、ごく少数の操作がパラレルで実行する場合のみです。このアルゴリズムでは最適なリソース使用率が保証されますが、ユーザーによってレスポンス時間が一定しないことがあります。確定したレスポンス時間を必要とする環境では問合せ調整並列処理のみを使用することはお薦めしません。問合せ調整並列処理は、データベース初期化パラメータPARALLEL_ADAPTIVE_MULTI_USERを介して制御されます。

ノート:

初期化パラメータ PARALLEL_ADAPTIVE_MULTI_USER は、Oracle Database 12c リリース 2 (12.2.0.1)で非推奨となり、将来のリリースではサポートされなくなるため、かわりにパラレル文のキューイングを使用することをお薦めします。

8.3 インメモリー・パラレル実行

インメモリー機能には、パラレル実行のための技法が用意されています。

この項では、インメモリー・パラレル実行について説明します。

- パラレル実行でのバッファ・キャッシュの使用
- 自動ビッグ・テーブル・キャッシング

8.3.1 パラレル実行でのバッファ・キャッシュの使用

デフォルトでは、オブジェクトが非常に小さいかCACHEとして宣言されている場合を除き、パラレル実行ではスキャン済ブロックをキャッシュするためにSGA (バッファ・キャッシュ)は使用されません。

インメモリー・パラレル実行では、パラメータPARALLEL_DEGREE_POLICYをAUTOに設定することで有効になっている場合、パラレル文でSGAを使用してオブジェクト・ブロックをキャッシュできます。Oracle Databaseでは、パラレル実行を使用してアクセスされるオブジェクトにとって、SGAにキャッシュされることが有益かどうかが判断されます。オブジェクトをキャッシュするかどうかの決定は、サイズやアクセス頻度などの明確な経験則に基づいて行われます。Oracle Real Applications Cluster (Oracle RAC)環境では、Oracle Databaseにより、オブジェクトのピースがアクティブ・インスタンスの各バッファ・キャッシュにマップされます。このマッピングを作成することで、オブジェクトの様々な部分もしくはピースを見つけるためにアクセスするバッファ・キャッシュが自動的にわかります。この情報を使用すると、複数のインスタンスが、ディスクから同じ情報を何度も繰り返して読み取ることがなくなり、オブジェクトをキャッシュできるメモリー容量を最大化することができます。これは、ブロックがキャッシュされるインスタンス上でPXサーバーを使用することで実行されます。

オブジェクトのサイズが、バッファ・キャッシュ(単一インスタンス)の合計サイズに基づくか、バッファ・キャッシュのサイズにOracle RACクラスタ内のアクティブ・インスタンスの数を乗算した値に基づく、特定のしきい値より大きい場合、オブジェクトはダイレクト・パス読取りを使用して読み取られ、SGAにキャッシュされません。

8.3.2 自動ビッグ・テーブル・キャッシング

自動ビッグ・テーブル・キャッシングでは、問合せをバッファ・キャッシュと統合して、単一インスタンス環境とOracle RAC環境の両方で、Oracle Databaseのインメモリー問合せ機能を拡張します。

Oracle Real Application Clusters (Oracle RAC)環境では、この機能はパラレル問合せでのみサポートされます。単一インスタンス環境では、この機能はパラレル問合せとシリアル問合せの両方でサポートされます。

大きな表のキャッシュに予約されているキャッシュ・セクションが、表スキャンのためのデータのキャッシュに使用されます。大きな表のキャッシュは主としてデータ・ウェアハウスのワークロードのパフォーマンスを高めるために設計されていますが、混在するワークロードを実行するOracle Databaseのパフォーマンスも向上します。

自動ビッグ表キャッシュでは、温度およびオブジェクト・ベースのアルゴリズムを使用して中間表およびビッグ表を追跡します。Oracleではかなり小規模の表をキャッシュしますが、自動ビッグ表キャッシュではこれらの表は追跡されません。

シリアル問合せで自動ビッグ・テーブル・キャッシングを有効化するには、DB_BIG_TABLE_CACHE_PERCENT_TARGET初期化パラメータの値(パーセント)を設定する必要があります。さらに、パラレル問合せで自動ビッグ・テーブル・キャッシングの使用を有効化するには、PARALLEL_DEGREE_POLICY初期化パラメータをAUTOまたはADAPTIVEに設定する必要があります。Oracle RAC環境では、自動ビッグ・テーブル・キャッシングはパラレル問合せでのみサポートされるため、両方の設定が必要です。

大きい表のサイズがほぼ、すべてのインスタンスの大きい表のキャッシュを合計したサイズである場合、すべてのインスタンスで、表はパーティション化されてキャッシュされます(または大部分がキャッシュされます)。インメモリー問合せで表の問合せのほとんどの

ディスク読取りが排除されるか、データベースで大きい表のキャッシュに入らない表の部分についてのみディスク読取りが行われます。大きい表のキャッシュでスキャン対象のすべての表をキャッシュできない場合、最も頻繁にアクセスされる表のみがキャッシュされ、残りは直接読取りで自動的に読み取られます。

DB_BIG_TABLE_CACHE_PERCENT_TARGETパラメータは、スキャンに使用するバッファ・キャッシュ・サイズのパーセントを決定します。DB_BIG_TABLE_CACHE_PERCENT_TARGETを80 (%)に設定した場合、バッファ・キャッシュの80 (%)がスキャンに使用され、残りの20 (%)がOLTPワークロードに使用されます。

PARALLEL_DEGREE_POLICYがAUTOまたはADAPTIVEに設定されている場合、DB_BIG_TABLE_CACHE_PERCENT_TARGETパラメータはOracle RAC環境でのみ有効化されます。DB_BIG_TABLE_CACHE_PERCENT_TARGETのデフォルトは0 (無効)で、上限は90 (%)で、少なくとも10%のバッファ・キャッシュを表スキャン以外の使用のために予約しています。値が0の場合、インメモリー問合せは既存のLeast Recently Used (LRU)メカニズムで実行されます。DB_BIG_TABLE_CACHE_PERCENT_TARGETパラメータは動的に調整できます。

DB_BIG_TABLE_CACHE_PERCENT_TARGETパラメータの設定時には、次のガイドラインを使用してください。

- Oracle RAC環境で自動並列度(DOP)を有効にしない場合は、このパラメータを設定しないでください。このような場合、大きい表のキャッシュ・セクションは使用されないためです。
- このパラメータを設定する際は、ワークロードの混在を検討する必要があります(OLTPにどれだけのワークロードを要するか; 挿入、更新、ランダム・アクセス; 表スキャンにどれだけのワークロードを要するか)。データ・ウェアハウスのワークロードではしばしば、大きい表のスキャンが実行されるため、大きい表のキャッシュ・セクションに、データ・ウェアハウス用に大きい割合のバッファ・キャッシュ領域を付与することを検討できます。
- このパラメータは、ワークロードが変わると、動的に変わる可能性があります。その時にバッファ・キャッシュ・メモリーがアクティブに使用されている可能性があるため、目標を達成する現在のワークロードによっては、変更に多少時間がかかることがあります。

PARALLEL_DEGREE_POLICYがAUTOまたはADAPTIVEに設定されている場合、データ・ウェアハウスのロードおよびスキャン・バッファのオブジェクトレベルの統計が追加され、特定(ヘルパー)インスタンスのオブジェクトのパラレル(PQ)スキャン数が示されます。

V$BT_SCAN_CACHEおよびV$BT_SCAN_OBJ_TEMPSビューには、大きい表のキャッシュに関する情報が表示されます。

関連項目:

- 自動ビッグ・テーブル・キャッシングの詳細は、『Oracle Database管理者ガイド』を参照してください
- 自動ビッグ・テーブル・キャッシングの詳細は、『Oracle Database概要』を参照してください
- DB_BIG_TABLE_CACHE_PERCENT_TARGET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
- V$BT_SCAN*ビューの詳細は、『Oracle Databaseリファレンス』を参照してください

8.4 パラレル文のキューイング

状況によっては、パラレル文はリソースの待機中にキューに入れられます。

パラメータPARALLEL_DEGREE_POLICYをAUTOに設定すると、必要な数のパラレル実行サーバー・プロセスが使用可能でない場合、パラレル実行を必要とするSQL文はキューに入ります。必要なリソースが使用可能になると、SQL文はデキューされ、実行が許可されます。デフォルトのデキュー順序は、文が発行された時刻に基づく単純な先入先出キューです。

次に、パラレル文処理のサマリーを示します。

1. SQL文が発行されます。
2. 文が解析され、DOPが自動的に決定されます。
3. 使用可能なパラレル・リソースがチェックされます。
    a. 十分なパラレル実行サーバーが使用でき、キューにリソース待ちの文がない場合は、そのSQL文が実行されます。
    b. 十分なパラレル実行サーバーが使用できない場合、SQL文は指定された条件に基づいてキューに入れられ、指定された条件が満たされたときにキューの先頭からデキューされます。

文を実行するとアクティブなパラレル・サーバーの数が初期化パラメータPARALLEL_SERVERS_TARGETの値を超える場合、そのパラレル文はキューに入れられます。たとえば、PARALLEL_SERVERS_TARGETが64に設定されており、現在アクティブなサーバーの数が60で、新しいパラレル文が16のパラレル・サーバーを必要とする場合、60に16を加えるとPARALLEL_SERVERS_TARGETの値である64より大きくなるため、そのパラレル文はキューに入れられます。

これはシステムで使用可能なパラレル・サーバー・プロセスの最大数ではなく、パラレル文のキューイングが使用されるまでにパラレル文の実行に使用できるパラレル・サーバー・プロセスの数です。各パラレル文に必要なパラレル・サーバー・プロセスをすべて確保するとともに、パラレル・サーバー・プロセスによるシステムのオーバーロードを回避するため、システムで許容されるパラレル・サーバー・プロセスの最大数(PARALLEL_MAX_SERVERS)未満に設定されます。ただし、文のキューイングがアクティブになっている場合でも、すべてのシリアル(非パラレル)文はただちに実行されます。

文がキューに入っているかどうかを確認するには、resmgr：pq queued待機イベントを使用します。

このセクションのトピックは次のとおりです：

- Oracle Database Resource Managerによるパラレル文のキューイングの管理について
- パラレル文のグループ化: BEGIN_SQL_BLOCK END_SQL_BLOCK
- ヒントによるパラレル文のキューイングの管理について

関連項目:

- パラレル文のキューの監視および分析のためのビューに関する詳細は、V$RSRC_SESSION_INFOおよびV$RSRCMGRMETRIC
- PARALLEL_SERVERS_TARGET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

8.4.1 Oracle Database Resource Managerによるパラレル文のキューイングの管理について

デフォルトでは、パラレル文のキューは先入先出キューとして動作しますが、リソース・プランを使用して、デフォルトの動作を変更できます。

リソース・プランを構成および設定して、パラレル文をデキューする順序と、各ワークロードまたはコンシューマ・グループが使用するパラレル実行サーバーの数を管理できます。

Oracle Database Resource Managerは、コンシューマ・グループおよびリソース・プランの概念に基づいて動作します。コンシューマ・グループは、同じリソース権限および要件を使用したユーザーのグループを識別します。リソース・プランは、各コンシューマ・グループに対するディレクティブのコレクションから構成され、パラレル・サーバーなどの各種データベース・リソースの管理と割当てを指定します。マルチテナント・コンテナ・データベース(CDB)およびプラガブル・データベース(PDB)では、パラレル文キューの順序はsharesと呼ばれるディレクティブによって管理されます。

リソース・プランは、RESOURCE_MANAGER_PLANパラメータをリソース・プランの名前に設定すると有効になります。

並列度ポリシーがAUTOに設定されている場合、次の項で説明するディレクティブを使用して、コンシューマ・グループのパラレル文の処理を管理できます。

- パラレル文キューの順序の管理について
- コンシューマ・グループに対するパラレル・サーバー・リソースの制限について
- 各コンシューマ・グループに対するパラレル文キューのタイムアウトの指定
- コンシューマ・グループに対する並列度制限の指定
- クリティカルなパラレル文の優先順位
- パラレル・キュー内の文を管理するシナリオ例

どのような場合でも、指定されたコンシューマ・グループのパラレル文のキューはOracle RACデータベース上の単一キューとして管理されます。各コンシューマ・グループに対する制限は、そのコンシューマ・グループに属するOracle RACデータベースのすべてのセッションに適用されます。パラレル文のキューイングは、データベース・インスタンス全体での初期化パラメータPARALLEL_SERVERS_TARGETの合計値に基づいて発生します。

マルチテナント・コンテナ・データベース(CDB)およびプラガブル・データベース(PDB)のパラレル文のキューイングも管理できます。

関連項目:

- Oracle Database Resource ManagerによるOracle Databaseリソースの管理の詳細は、『Oracle Database管理者ガイド』を参照してください
- パラレル実行(PX)サーバーおよびマルチテナント・コンテナ・データベース(CDB)とプラガブル・データベース(PDB)での使用率制限の詳細は、『Oracle Multitenant管理者ガイド』を参照してください
- V$RSRC*ビュー、DBA_HIST_RSRC_CONSUMER_GROUPビューおよびパラレル問合せの待機イベントの詳細は、『Oracle Databaseリファレンス』を参照してください
- DBMS_RESOURCE_MANAGER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

8.4.1.1 パラレル文キューの順序の管理について

Oracle Database Resource Managerを使用して、複数のコンシューマ・グループ間のパラレル文のキューからパラレル文をデキューする優先順位を管理できます。

特定のコンシューマ・グループのパラレル文は、デフォルトで先入先出の順序でデキューされます。ディレクティブsharesを使用すると、コンシューマ・グループのパラレル文がデキューされる順序を決定できます。これらのディレクティブをDBMS_RESOURCE_MANAGER PL/SQLパッケージのCREATE_PLAN_DIRECTIVEまたはUPDATE_PLAN_DIRECTIVEプロシージャを使用して構成します。また、PDB間のパラレル文の順序を管理するために、CDBリソース・プランでsharesを設定できます。

たとえば、コンシューマ・グループPQ_HIGH、PQ_MEDIUMおよびPQ_LOWを作成し、優先順位に基づいてパラレル文セッションをこれらのコンシューマ・グループにマップします。次にリソース・プランを作成し、PQ_HIGHにshares=14、PQ_MEDIUMにshares=5、PQ_LOWにshares=1を設定します。これは、PQ_HIGHの文がデキューされる確率は70% (14/(14+5+1)=.70)、PQ_MEDIUMがデキューされる確率は25% (5/(14+5+1)=.25)、そしてPQ_LOWがデキューされる確率は5% (1/(14+5+1)=.05)であることを示しています。

パラレル文がキューに入っており、そのパラレル文をただちに実行する必要があると判断した場合は、DBMS_RESOURCE_MANAGER.DEQUEUE_PARALLEL_STATEMENT PL/SQLプロシージャを実行して、そのパラレル文をデキューできます。

関連項目:

- DBMS_RESOURCE_MANAGERパッケージ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
- リソース・プラン・ディレクティブの作成の詳細は、『Oracle Database管理者ガイド』を参照してください

8.4.1.2 コンシューマ・グループに対するパラレル・サーバー・リソースの制限について

Oracle Database Resource Managerを使用して、優先順位の低いコンシューマ・グループのパラレル文がパラレル文処理に使用できるパラレル・サーバーの数を制限できます。

Oracle Database Resource Managerを使用してパラレル文セッションを異なるコンシューマ・グループにマップし、それぞれに使用できるパラレル・サーバーの数に特定の制限を持たせることができます。各コンシューマ・グループには、独自の個々のパラレル文キューがあります。コンシューマ・グループのこの制限を指定した場合、コンシューマ・グループのパラレル文は、その制限を超えるとキューに入れられます。

この制限は、データベースにコンシューマ・グループの優先順位が高いものと低いものがある場合に役立ちます。制限を設けないと、優先順位の低いコンシューマ・グループから大量のパラレル文が発行されて、すべてのパラレル・サーバーが使用される可能性があります。優先順位の高いコンシューマ・グループからパラレル文が発行されたときに、リソース割当てディレクティブにより、その優先順位の高いパラレル文を確実に最初にデキューすることができます。優先順位の低いコンシューマ・グループが使用できるパラレル・サーバーの数を制限することにより、優先順位の高いコンシューマ・グループが使用できるパラレル・サーバーを常に確保しておくことができます。

コンシューマ・グループが使用するパラレル・サーバーを制限するには、DBMS_RESOURCE_MANAGERパッケージで、CREATE_PLAN_DIRECTIVEプロシージャのparallel_server_limitパラメータを使用するか、UPDATE_PLAN_DIRECTIVEプロシージャのnew_parallel_server_limitパラメータを使用します。parallel_server_limitパラメータは、PARALLEL_SERVERS_TARGETで指定されるOracle RAC全体のパラレル・サーバー・プールのうち、コンシューマ・グループが使用できる最大パーセンテージを指定します。

マルチテナント・コンテナ・データベース(CDB)のリソース・プランでは、プラガブル・データベース(PDB)にパラレル・サーバー制限が適用されます。PDBリソース・プランまたはCDB以外のリソース・プランの場合、この制限がコンシューマ・グループに適用されます。

たとえば、非マルチテナント構成のOracle RACデータベースでは、初期化パラメータPARALLEL_SERVERS_TARGETが2つのノードで32に設定されているため、キューイングが開始されるまでに使用できるパラレル・サーバーの合計数は32 x 2 = 64です。使用可能なパラレル・サーバーのうち50%をコンシューマ・グループPQ_LOWが使用するように設定し(parallel_server_limit = 50)、優先順位の低い文をPQ_LOWコンシューマ・グループにマップできます。このシナリオでは、コンシューマ・グループPQ_LOWのパラレル文は、64 x 50% = 32のパラレル・サーバーに制限されます。これは、たとえ非アクティブなまたは未使用のパラレル・サーバーがこれより多くあったとしても同じです。このシナリオでは、コンシューマ・グループPQ_LOWの文は、32のパラレル・サーバーをすべて使用すると、キューに入れられます。

1つのデータベース内に並列度ポリシーをMANUALに設定したセッションとAUTOに設定したセッションを混在させることが可能です。このシナリオでは、並列度ポリシーをAUTOに設定したセッションのみキューに入れることができます。ただし、並列度ポリシーをMANUALに設定したセッションで使用されるパラレル・サーバーは、コンシューマ・グループで使用されるパラレル・サーバーの合計数に含まれます。

関連項目:

ユーザーに対するパラレル・リソースの制限の詳細は、PARALLEL_SERVERS_TARGETを参照してください

8.4.1.3 各コンシューマ・グループに対するパラレル文キューのタイムアウトの指定

Oracle Database Resource Managerを使用して、コンシューマ・グループに対して特定のキュー・タイムアウト最大値を設定し、パラレル文がキューに長時間入ったままにならないようにすることができます。

キューのタイムアウトを管理するには、DBMS_RESOURCE_MANAGERパッケージで、CREATE_PLAN_DIRECTIVEプロシージャのparallel_queue_timeoutパラメータを使用するか、UPDATE_PLAN_DIRECTIVEプロシージャのnew_parallel_queue_timeoutパラメータを使用します。parallel_queue_timeoutおよびnew_parallel_queue_timeoutパラメータは、文がコンシューマ・グループのパラレル文のキューに留まることのできる時間を秒単位で指定します。タイムアウト時間が経過すると、文はエラーORA-7454で終了するか、パラレル文のキューから削除され、リソース・マネージャ計画のPQ_TIMEOUT_ACTIONディレクティブの値に基づいて実行するように有効化されます。

PQ_TIMEOUT_ACTIONリソース・マネージャ・ディレクティブを使用してパラレル文のキュー・タイムアウト・アクションを指定できます。このディレクティブをCANCELに設定すると、エラーORA-7454で文が終了します。このディレクティブをRUNに設定すると、文は実行可能になります。

8.4.1.4 コンシューマ・グループに対する並列度制限の指定

Oracle Database Resource Managerを使用して、特定のコンシューマ・グループに対する並列度を制限できます。

Oracle Database Resource Managerを使用して、リソース・プランでパラレル文セッションをそれぞれ特定の並列度制限を持つ異なるコンシューマ・グループにマップできます。

コンシューマ・グループの並列度制限を管理するには、DBMS_RESOURCE_MANAGERパッケージでCREATE_PLAN_DIRECTIVEプロシージャのparallel_degree_limit_p1パラメータを使用するか、DBMS_RESOURCE_MANAGERパッケージでUPDATE_PLAN_DIRECTIVEプロシージャのnew_parallel_degree_limit_p1パラメータを使用します。parallel_degree_limit_p1およびnew_parallel_degree_limit_p1パラメータは、任意の操作について並列度の制限を指定します。

たとえば、コンシューマ・グループPQ_HIGH、PQ_MEDIUMおよびPQ_LOWを作成し、優先順位に基づいてパラレル文セッションをこれらのコンシューマ・グループにマップします。それから並列度の制限を指定するリソース・プランを作成し、PQ_HIGHの制限値を16、PQ_MEDIUMの制限値を8、PQ_LOWの制限値を2に設定します。

PARALLEL_DEGREE_POLICYがAUTOに設定されていない場合でも、並列度の制限が強制されます。

8.4.1.5 クリティカルなパラレル文の優先順位

PARALLEL_STMT_CRITICALパラメータの設定は、パラレル文のキューに関連するプラン・ディレクティブのパラレル文のクリティカル指定に影響します。

- PARALLEL_STMT_CRITICALパラメータがQUEUEに設定されている場合、PARALLEL_DEGREE_POLICYがMANUALに設定されているパラレル文はキューに入れられます。
- PARALLEL_STMT_CRITICALパラメータがBYPASS_QUEUEに設定されている場合、パラレル文はパラレル文のキューを回避し、即座に実行されます。
- PARALLEL_STMT_CRITICALがFALSEに設定されている場合は、デフォルトの動作が指定され、クリティカルとして指定される文はありません。

クリティカルなパラレル文はパラレル文のキューを回避するため、システムは、PARALLEL_SERVERS_TARGETパラメータの指定より多くのアクティブなパラレル・サーバーを検出する場合があります。パラレル・サーバーの数がPARALLEL_MAX_SERVERSに達した後にクリティカルなパラレル文を実行できるため、一部のクリティカルなパラレル文がダウングレードされる場合があります。

DBA_RSRC_PLAN_DIRECTIVESビューのPARALLEL_STMT_CRITICAL列は、パラレル文がクリティカルとマークされたコンシューマ・グループかどうかを示します。

8.4.1.6 パラレル・キュー内の文を管理するシナリオ例

このシナリオでは、Oracle Database Resource Managerを使用してコンシューマ・グループを設定してパラレル・キュー内の文を管理する方法について説明します。

このシナリオでは、3種類のSQL文で構成されるデータ・ウェアハウスのワークロードについて考えます。

- 実行時間の短いSQL文

  実行時間が短い文とは、実行時間が1分以内の文を指します。このような文はレスポンスが非常によいことが予想されます。

- 実行時間が中程度のSQL文

  実行時間が中程度の文とは、実行時間が1分より長く15分以内の短い文を指します。このような文はレスポンスがある程度よいことが予想されます。

- 実行時間の長いSQL文

  実行時間の長い文とは、実行時間が15分より長い非定型の問合せや複雑な問合せを指します。このような文は長時間かかると予想されます。

このデータ・ウェアハウス・ワークロードでは、実行時間の短い文でよりよいレスポンスが求められています。この目標を達成するには、次の点について確認する必要があります。

- 実行時間の長い文がパラレル・サーバー・リソースをすべて使用して、実行時間のより短い文がパラレル文のキューで待機させられることがないようにします。
- 実行時間の短い文と長い文の両方がキューに入っている場合は、実行時間の短い文を長い文よりも先にデキューするようにします。

- 実行時間の短い問合せのDOPを制限します。これはDOPを高くすることで得られる高速化は、多数のパラレル・サーバーの使用を正当化するほど重要ではないためです。

例8-3に、Oracle Database Resource Managerを使用してコンシューマ・グループを設定して、パラレル文キュー内の文に優先順位を設定する方法を示します。この例については、次の内容に注意してください。

- デフォルトでは、ユーザーはコンシューマ・グループOTHER_GROUPSに割り当てられています。SQL文の推定実行時間が1分(60秒)より長い場合は、ユーザーはMEDIUM_SQL_GROUPに切り替わります。switch_for_callがTRUEに設定されている場合、その文の完了後、ユーザーはOTHER_GROUPSに戻ります。ユーザーがMEDIUM_SQL_GROUPにあり、文の推定実行時間が15分(900秒)より長い場合は、ユーザーはLONG_SQL_GROUPに切り替わります。同様に、switch_for_callがTRUEに設定されている場合は、問合せの完了後、ユーザーはOTHER_GROUPSに戻ります。切替え処理の実行に使用されるディレクティブは、switch_time、switch_estimate、switch_for_callおよびswitch_groupです。
- アクティブなパラレル・サーバーの数が初期化パラメータPARALLEL_SERVERS_TARGETの値に達すると、それ以降のパラレル文はキューに入れられます。sharesディレクティブは、パラレル・サーバーが使用可能になったときにパラレル文をデキューする順番を制御します。この例では、SYS_GROUPのsharesが100%に設定されているため、SYS_GROUPのパラレル文は常に最初にデキューされます。SYS_GROUPのパラレル文がキューにない場合、OTHER_GROUPSのパラレル文は70%、MEDIUM_SQL_GROUPの文は20%、LONG_SQL_GROUPの文は10%の確率でデキューされます。
- OTHER_GROUPSから発行されるパラレル文は、parallel_degree_limit_p1ディレクティブの設定により、DOPが4に制限されます。
- LONG_SQL_GROUPグループのパラレル文がパラレル・サーバーをすべて使用して、OTHER_GROUPSやMEDIUM_SQL_GROUPのパラレル文を長時間待機させることのないようにするには、そのparallel_server_limitディレクティブを50%に設定します。つまり、LONG_SQL_GROUPが初期化パラメータPARALLEL_SERVERS_TARGETで設定したパラレル・サーバーの50%まで使用すると、それ以降はそのグループのパラレル文を強制的にキューで待機させます。
- LONG_SQL_GROUPグループのパラレル文はかなり長い時間キューに留まる可能性があるため、14400秒(4時間)のタイムアウトが設定されます。LONG_SQL_GROUPのパラレル文は、キューで4時間待機すると、エラーORA-7454により停止されます。

例8-3 コンシューマ・グループを使用したパラレル文キュー内の優先順位の設定

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();

  /* Create consumer groups.
   * By default, users start in OTHER_GROUPS, which is automatically
   * created for every database.
   */
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(
    'MEDIUM_SQL_GROUP',
    'Medium-running SQL statements, between 1 and 15 minutes.  Medium priority.');

  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(
    'LONG_SQL_GROUP',
    'Long-running SQL statements of over 15 minutes.  Low priority.');

  /* Create a plan to manage these consumer groups */
  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    'REPORTS_PLAN',
    'Plan for daytime that prioritizes short-running queries');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    'REPORTS_PLAN', 'SYS_GROUP', 'Directive for sys activity',
```

```
    shares => 100);

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    'REPORTS_PLAN', 'OTHER_GROUPS', 'Directive for short-running queries',
    shares => 70,
    parallel_degree_limit_p1 => 4,
    switch_time => 60, switch_estimate => TRUE, switch_for_call => TRUE,
    switch_group => 'MEDIUM_SQL_GROUP');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    'REPORTS_PLAN', 'MEDIUM_SQL_GROUP', 'Directive for medium-running queries',
    shares => 20,
    parallel_server_limit => 80,
    switch_time => 900, switch_estimate => TRUE, switch_for_call => TRUE,
    switch_group => 'LONG_SQL_GROUP');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    'REPORTS_PLAN', 'LONG_SQL_GROUP', 'Directive for medium-running queries',
    shares => 10,
    parallel_server_limit => 50,
    parallel_queue_timeout => 14400);

  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/

/* Allow all users to run in these consumer groups */
EXEC DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP(
  'public', 'MEDIUM_SQL_GROUP', FALSE);

EXEC DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP(
  'public', 'LONG_SQL_GROUP', FALSE);
```

8.4.2 パラレル文のグループ化: BEGIN_SQL_BLOCK END_SQL_BLOCK

複数のパラレル文からなるレポートやバッチ・ジョブをできるだけ高速に完了することが重要な場合がよくあります。

たとえば、多数のレポートを同時に開始した場合、すべてのレポートをできるだけすばやく完了させる必要があります。ただし、すべてのレポートを同時に終わらせるのではなく、特定のレポートを最初に完了させたい場合もあります。

レポートに複数のパラレル文が含まれており、PARALLEL_DEGREE_POLICYがAUTOに設定されている場合、それぞれのパラレル文がビジー状態のデータベース上でキューに待機する可能性があります。たとえば、次のステップでは、SQL文の処理でのシナリオについて説明します。

```
serial statement
parallel query - dop 8
  -> wait in queue
serial statement
parallel query - dop 32
  -> wait in queue
parallel query - dop 4
  -> wait in queue
```

レポートをすばやく完了させるには、パラレル文が次のように動作するようグループ化する必要があります。

```
start SQL block
serial statement
```

```
parallel query - dop 8
  -> first parallel query: ok to wait in queue
serial statement
parallel query - dop 32
  -> avoid or minimize wait
parallel query - dop 4
  -> avoid or minimize wait
end SQL block
```

パラレル文をグループ化するには、DBMS_RESOURCE_MANAGER PL/SQLパッケージでBEGIN_SQL_BLOCKおよびEND_SQL_BLOCKプロシージャを使用します。各コンシューマ・グループに対して、コンシューマ・グループの各パラレル文に関連付けられた時間によってパラレル文キューの順序付けを行います。通常は、パラレル文に関連付けられた時間は、その文がエンキューされた時間で、つまりキューは先入先出(FIFO)であることを意味します。BEGIN_SQL_BLOCKおよびEND_SQL_BLOCKプロシージャを使用してパラレル文をSQLブロックにグループ化する場合、最初にキューイングされたパラレル文は同様にエンキューされた時間を使用します。ただし、2番目とその後続のパラレル文はすべて特別な扱いを受け、SQLブロック内の最初にキューイングされたパラレル文のエンキュー時間を使用してエンキューされます。この機能により、文がパラレル文キューの前方に移動することがよくあります。この優先的な扱いにより、待機時間を最小限に抑えられます。

関連項目:

DBMS_RESOURCE_MANAGER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

8.4.3 ヒントによるパラレル文のキューイングの管理について

SQL文のNO_STATEMENT_QUEUINGおよびSTATEMENT_QUEUINGヒントを使用して、文がパラレル文のキューイングでキューに入れられるかどうかに影響を与えることができます。

- NO_STATEMENT_QUEUING

  PARALLEL_DEGREE_POLICYをAUTOに設定すると、このヒントにより文がパラレル文キューに入らないようにすることができます。ただし、文のキューを回避する文は、パラレル文のキューイングを開始する制限を決定するPARALLEL_SERVERS_TARGET初期化パラメータの値で定義されるパラレル実行サーバーの最大数を超える可能性があります。

  システムで使用できるパラレル実行サーバーの数がPARALLEL_MAX_SERVERS初期化パラメータの値までに制限されるため、パラレル文のキューイングを回避する文がリクエストされたパラレル実行サーバーの数を受け取る保証はありません。

  例:

  ```
  SELECT /*+ NO_STATEMENT_QUEUING */ last_name, department_name
    FROM employees e, departments d
    WHERE e.department_id = d.department_id;
  ```

- STATEMENT_QUEUING

  PARALLEL_DEGREE_POLICYをAUTOに設定しない場合は、このヒントにより文をパラレル文のキューイングに考慮し、リクエストされたDOPで十分なパラレル処理を実行できる場合にのみ実行させることができます。キューイングを有効にする前の使用できるパラレル実行サーバーの数は、使用するパラレル実行サーバーの数とPARALLEL_SERVERS_TARGET初期化パラメータで定義されるシステムで許可される最大数の違いと同じです。

  例:

```
SELECT /*+ STATEMENT_QUEUING */ last_name, department_name
  FROM employees e, departments d
  WHERE e.department_id = d.department_id;
```

8.5 並列処理の種類

並列処理には複数のタイプがあります。

ここでは、次の項目で並行処理の種類について説明します。

- パラレル問合せについて
- パラレルDDL文について
- パラレルDML操作について
- 関数のパラレル実行について
- その他の並列処理の種類について
- SQL文の並列度ルール

8.5.1 パラレル問合せについて

SELECT文でパラレル問合せとパラレル副問合せを使用できます。DDL文とDML文(INSERT、UPDATEおよびDELETE)の問合せ部分もパラレルで実行できます。

また、外部表をパラレルで問い合せることもできます。

SQL問合せのパラレル化決定には、パラレル化の決定と並列度(DOP)という2つの要素があります。これらの要素の決定方法は、問合せ、DDL操作およびDML操作で異なります。DOPを決定するときに、Oracle Databaseは次の参照オブジェクトを調べます。

- パラレル問合せは、パラレルで実行される問合せの部分で、それぞれの表および索引を調べて、どれが参照表かを決定します。原則として、最大のDOPが設定された表または索引を選択します。
- パラレルDML(INSERT、UPDATE、MERGEおよびDELETE)では、DOPを決定する参照オブジェクトは、挿入、更新または削除操作によって変更される表です。パラレルDMLではデッドロックを防ぐためにDOPに制限が課せられます。パラレルDML文に副問合せが含まれる場合、副問合せのDOPはDML操作のDOPと同じになります。
- パラレルDDLでは、DOPを決定する参照オブジェクトは、作成、再構築、分割または移動される表、索引またはパーティションです。パラレルDDL文に副問合せが含まれる場合、副問合せのDOPはDDL操作と同じになります。

この項では、次の項目について説明します。

- 索引構成表のパラレル問合せ
- 非パーティション索引構成表
- パーティション化索引構成表
- オブジェクト型のパラレル問合せ
- 問合せのパラレル化のルール

関連項目:

- パラレル問合せの実行方法の詳細は、「SQL文のパラレル実行」を参照してください
- リモート・オブジェクトを参照する問合せの例は、「分散トランザクションの制限」を参照してください

- 問合せをパラレルで実行するための条件とDOPを決定する要素の詳細は、「問合せのパラレル化のルール」を参照してください

8.5.1.1 索引構成表のパラレル問合せ

索引構成表ではいくつかのパラレル・スキャン方法がサポートされています。

パラレル・スキャン方法には、次が含まれます。

- 非パーティション索引構成表のパラレル高速全スキャン
- パーティション索引構成表のパラレル高速全スキャン
- パーティション索引構成表のパラレル索引レンジ・スキャン

オーバーフロー領域を含む索引構成表およびLOBを含む索引構成表に対して、これらのスキャン方法を使用できます。

8.5.1.2 非パーティション索引構成表

非パーティション索引構成表に対するパラレル問合せでは、パラレル高速全スキャンが使用されます。

作業の割当ては、索引セグメントを十分な数のブロック・レンジに分割し、ブロック・レンジをデマンドドリブン方式でパラレル実行サーバーに割り当てることで行われます。任意の行に対応するオーバーフロー・ブロックは、その行を所有するプロセスによってのみデマンドドリブン方式でアクセスされます。

8.5.1.3 パーティション化索引構成表

索引レンジ・スキャンと高速全スキャンの両方をパラレルで実行できます。

パラレル高速全スキャンでは、パラレル化は非パーティション索引構成表の場合と同じです。DOPによって異なりますが、各パラレル実行サーバーは、1つ以上のパーティションを取得します。各パーティションは、主キー索引セグメントと関連するオーバーフロー・セグメント(ある場合)を含みます。

8.5.1.4 オブジェクト型のパラレル問合せ

パラレル問合せを、オブジェクト型の表およびオブジェクト型の列を含む表に実行することができます。

オブジェクト型に対するパラレル問合せでは、オブジェクト型に対する順次問合せで使用できるすべての機能がサポートされます。

- オブジェクト型に対するメソッド
- オブジェクト型の属性アクセス
- オブジェクト型インスタンスを作成するためのコンストラクタ
- オブジェクト・ビュー
- オブジェクト型に対するPL/SQLおよびOracle Call Interface (OCI)問合せ

パラレル問合せではオブジェクト型のサイズの制限はありません。

オブジェクト型に対してパラレル問合せを使用する際には次の制限が適用されます。

- 結合とソート(ORDER BY、GROUP BYまたは集合操作による)を含む問合せをパラレルで実行するにはMAP関数が必要です。MAP関数がない場合、問合せは自動的にシリアルで実行されます。
- パラレルDMLおよびパラレルDDLはオブジェクト型に対してはサポートされず、そのような文は常にシリアルで実行されま

す。

これらいずれかの制限のために問合せをパラレルで実行できない場合は、常に問合せ全体がシリアルで実行されエラー・メッセージは返されません。

8.5.1.5 問合せのパラレル化のルール

SQL問合せは、特定の条件下でのみパラレルで実行できます。

SELECT文をパラレルで実行できるのは、次の条件のいずれかが満たされている場合のみです。

- 問合せに、文レベルまたはオブジェクト・レベルのパラレル・ヒント仕様(PARALLELまたはPARALLEL_INDEX)が含まれている。
- 問合せ内で参照されるスキーマ・オブジェクトに、PARALLEL宣言が関連付けられている。
- 自動並列度(自動DOP)が有効になっている。
- ALTER SESSION FORCE PARALLEL QUERY文を使用してパラレル問合せが強制されている。

また、実行計画には、少なくとも次のいずれかが必要となります。

- 全表スキャン
- 複数パーティションに及ぶ索引レンジ・スキャン
- 索引高速全体スキャン
- パラレル表関数

関連項目:

- 自動DOPの詳細は、「自動並列度」を参照してください
- 並列度(DOP)を決定するルールの詳細は、「SQL文の並列度ルール」を参照してください
- ALTER SESSION SQL文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

8.5.2 パラレルDDL文について

このトピックでは、DDL文の並列処理について説明します。

DDL文の並列処理に関するこの項には、次のトピックが含まれます。

- パラレル化できるDDL文
- パラレルでのCREATE TABLE AS SELECTの使用について
- リカバリ可能性とパラレルDDL
- パラレルDDLの領域管理
- ディクショナリ管理表領域使用時の記憶領域
- 空き領域とパラレルDDL
- DDL文のルール
- CREATE TABLE AS SELECTのルール

8.5.2.1 パラレル化できるDDL文

表および索引(パーティションまたは非パーティション)に対するDDL文をパラレルで実行できます。

非パーティション表および非パーティション索引のパラレルDDL文は次のとおりです。

- CREATE INDEX
- CREATE TABLE AS SELECT
- ALTER TABLE MOVE
- ALTER INDEX REBUILD

パーティション表およびパーティション索引のパラレルDDL文は次のとおりです。

- CREATE INDEX
- CREATE TABLE AS SELECT
- ALTER TABLE {MOVE|SPLIT|COALESCE} PARTITION
- ALTER INDEX {REBUILD|SPLIT} PARTITION
    - この文をパラレルで実行できるのは、分割対象の(グローバル)索引パーティションが使用可能な場合のみです。

このようなDDL操作はすべて、パラレル実行でもシリアル実行でもNOLOGGINGモードで実行できます。

索引構成表のCREATE TABLE文は、AS SELECT句の有無にかかわらずパラレルで実行できます。

パラレルDDLは、オブジェクト列を含む表では実行できません。パラレルDDLは、LOB列を含む非パーティション表では実行できません。

8.5.2.2 パラレルでのCREATE TABLE AS SELECTの使用について

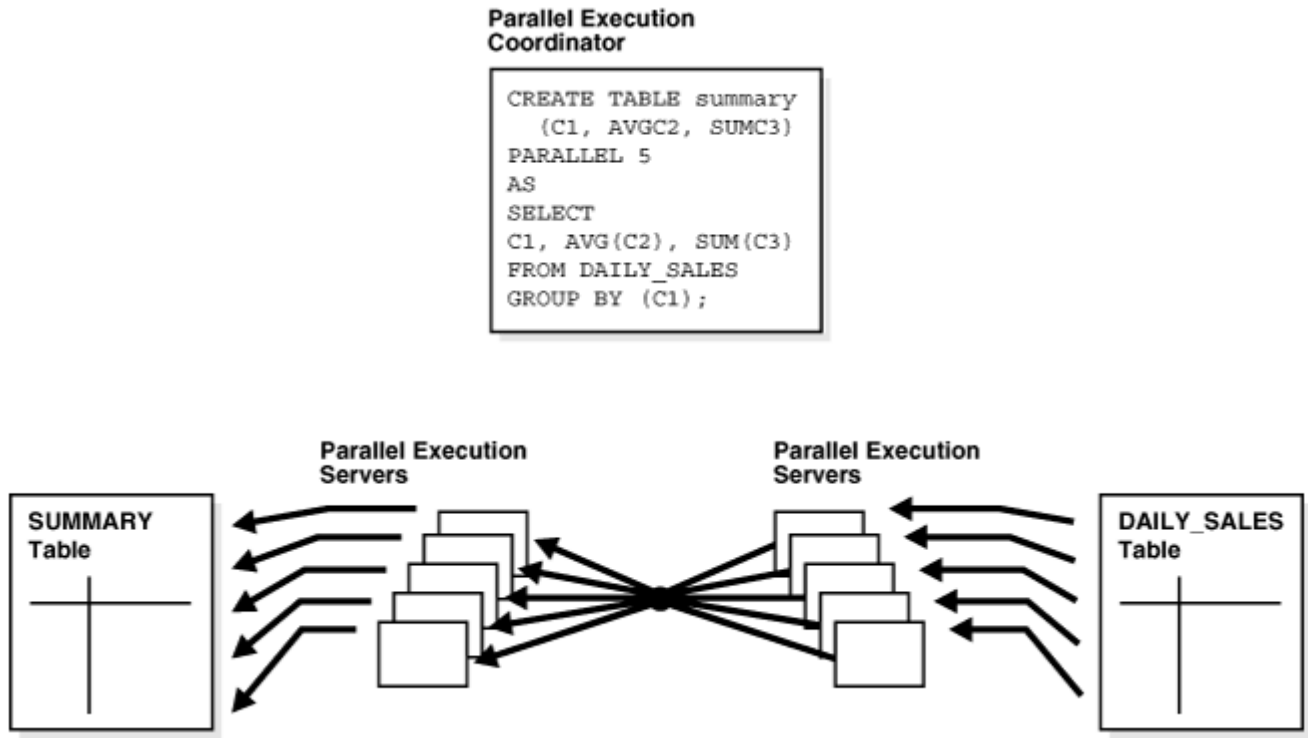
パラレル実行により、問合せをパラレルで実行でき、別の表または表セットから副問合せとして表を作成する操作を作成できます。

このパラレル機能は、サマリーまたはロールアップ表の作成で非常に役に立ちます。

クラスタ表の作成と移入は、パラレルで行えないことに注意してください。

図8-4に、パラレルでの副問合せを使用したサマリー表の作成を示します。

図8-4 パラレルでのサマリー表の作成



8.5.2.3 リカバリ可能性とパラレルDDL

パラレルDDLは、サマリー表の作成や、スタンドアロン・トランザクションである大量データ・ロードの実行(必ずしもリカバリ可能でなくてもよい)によく使用されます。

Oracle Databaseロギングをオフにするとundoログまたはredoログは生成されないため、パラレルDML操作のパフォーマンスは多くの場合向上しますがオールオアナッシング操作になります。つまり、操作がなんらかの理由で失敗した場合、操作をやりなおす必要があります。再開することはできません。

パラレル表作成(またはその他のパラレルDDL操作)時にロギングを無効化する場合は、メディア障害による表の損失を防ぐために、表が作成されてからその表を含む表領域をバックアップする必要があります。

undoログおよびredoログの生成を無効にするには、CREATE TABLE、CREATE INDEX、ALTER TABLEおよびALTER INDEX文でNOLOGGING句を使用します。

8.5.2.4 パラレルDDLの領域管理

表または索引のパラレルでの作成は、領域管理と密接に関連します。

これらの領域管理は、パラレル操作時に必要な記憶領域、および表または索引が作成された後に使用可能な空き領域の両方に影響します。

8.5.2.5 ディクショナリ管理表領域使用時の記憶領域

表または索引をパラレルで作成するとき、各パラレル実行サーバーはCREATE文のSTORAGE句の値を使用して、行を格納するための一時セグメントを作成します。

NEXT設定が4MB、PARALLEL DEGREEが16で作成される表は、作成時に少なくとも64MBの記憶域を消費します。各パラレル・サーバー・プロセスが4MBのエクステントで開始するためです。パラレル実行コーディネータがセグメントを結合するときに、セグメントの一部が切り捨てられ、生成される表はリクエストの64MBよりも小さくなることがあります。

8.5.2.6 空き領域とパラレルDDL

索引と表をパラレルで作成するとき、各パラレル実行サーバーが新しいエクステントを割り当て、エクステントに表または索引の

データを格納します。

たとえば、DOPを4として索引を作成する場合、索引には最初の時点で少なくとも4つのエクステントがあります。このエクステントの割当ては、パラレルでの索引再構築の場合や、パラレルでのパーティションの移動、分割または再構築の場合と同様に行われます。

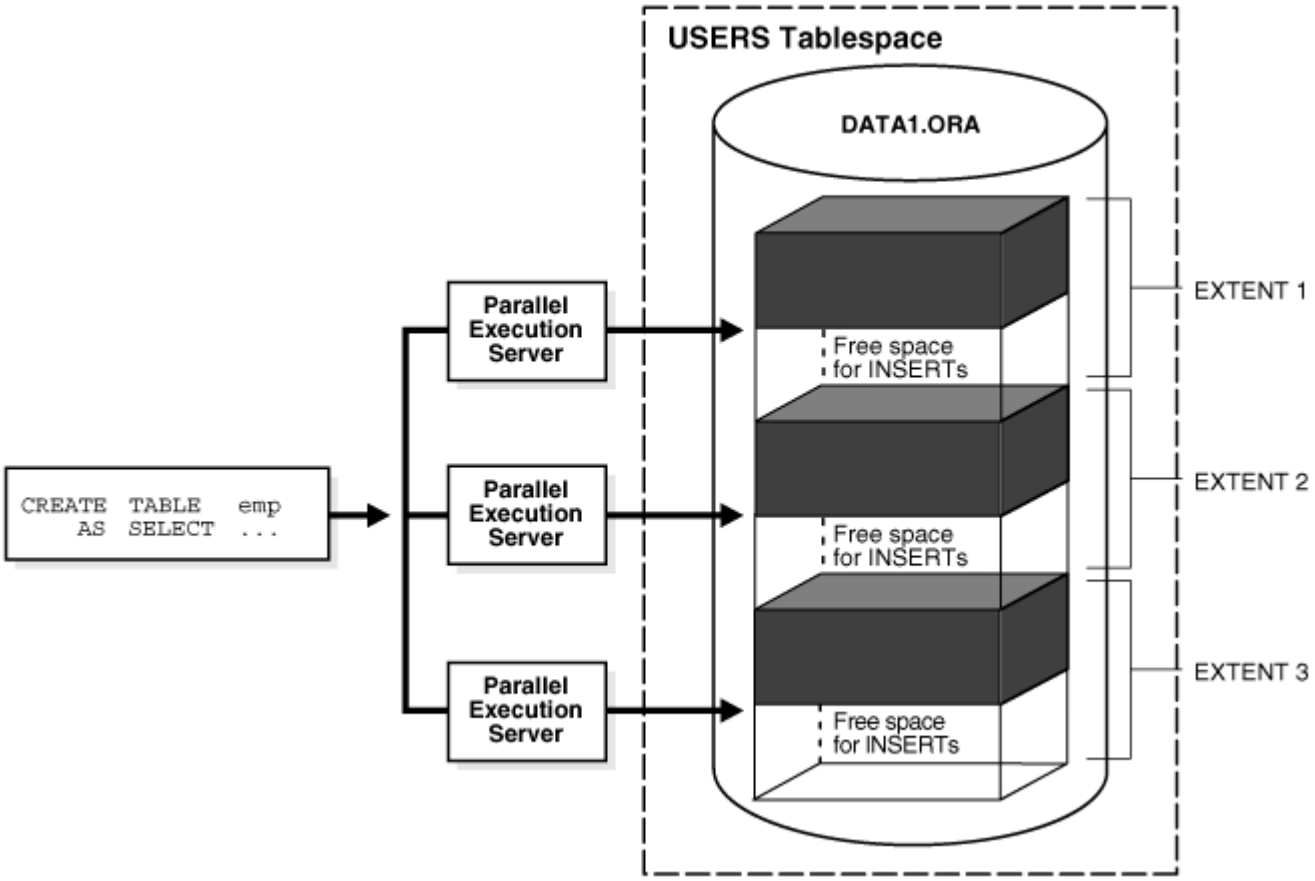
シリアル操作では、スキーマ・オブジェクトに少なくとも1つのエクステントが必要です。パラレル作成では、表または索引に、スキーマ・オブジェクトを作成するパラレル実行サーバーと同数のエクステントがあることが必要です。

表または索引をパラレルで作成するとき、空き領域が作成されることがあります。これは、パラレル実行サーバーで使用される一時セグメントが、行の格納に必要なサイズよりも大きい場合に発生します。

- 各一時セグメント内の未使用領域が、表領域レベルで設定されたMINIMUM EXTENTパラメータの値よりも大きい場合、Oracle Databaseがすべての一時セグメントの行を表または索引にマージするときに未使用領域を切り捨てます。未使用領域はシステムの空き領域に戻され、新しいエクステント用に割り当てることができます。ただし、連続した領域ではないため、結合してより大きいセグメントにすることはできません(外部断片化)。
- 各一時セグメントの未使用領域がMINIMUM EXTENTパラメータの値よりも小さい場合は、一時セグメントの行をマージするときに未使用領域を切り捨てることはできません。この未使用領域はシステムの空き領域に戻されません。表または索引の一部として残り(内部断片化)、後から行われる挿入や追加領域を必要とする更新のためにのみ使用されます。

たとえば、CREATE TABLE AS SELECT文でDOPを3と指定したが、表領域には1つしかデータファイルがない場合、図8-5に示すように内部断片化が発生することがあります。データファイルの内部表エクステント内の空き領域を他の空き領域と結合して、エクステントとして割り当てることはできません。

図8-5 使用できない空き領域(内部断片化)



関連項目:

パラレルでの表および索引作成の詳細は、『Oracle Database SQLチューニング・ガイド』を参照してください

8.5.2.7 DDL文のルール

DDL操作は、特定の条件下でパラレルで実行できます。

DDL操作をパラレルで実行できるのは、次の条件のうち少なくとも1つが満たされている場合のみです。

- 構文でPARALLEL句(宣言)が指定されている。CREATE TABLE、CREATE INDEX、ALTER INDEX REBUILDおよびALTER INDEX REBUILD PARTITIONの場合、パラレル宣言はデータ・ディクショナリに格納されます。
- 自動並列度(自動DOP)が有効になっている。
- ALTER SESSION FORCE PARALLEL DDL文を使用してパラレルDDLが強制されている。

関連項目:

- 自動DOPの詳細は、「自動並列度」を参照してください
- 並列度(DOP)を決定するルールの詳細は、「SQL文の並列度ルール」を参照してください
- ALTER SESSION SQL文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

8.5.2.8 CREATE TABLE AS SELECTのルール

CREATE TABLE AS SELECT文のCREATE操作は、パラレル化DDL文のルールに基づいてパラレル化されます。

また、文のSELECT部分で指定されている文レベルPARALLELヒントでも、DDL操作をパラレル化できます。パラレル化DDL文のルールの詳細は、DDL文のルールを参照してください。

CREATE TABLE AS SELECTのCREATE操作がパラレル化されるとき、可能であればスキャン操作もパラレル化されます。

DDL部分がパラレル化されない場合でも、問合せのパラレル化のルールに基づいて、SELECT部分をパラレル化できます。

自動並列度(自動DOP)では、文のDDL部分と問合せ部分の両方がパラレル化されます。

並列度(DOP)を決定するためのルールの詳細は、SQL文の並列度ルールを参照してください。

8.5.3 パラレルDML操作について

このトピックでは、パラレルDML操作について説明します。

パラレルDML(PARALLEL INSERT、UPDATE、DELETEおよびMERGE)は、パラレル実行メカニズムを使用して、大規模なデータベース表や索引に対する大規模なDML操作を高速化または拡張します。

ノート:

通常、DML には問合せが含まれますが、この章では DML という語は INSERT、UPDATE、MERGE および DELETE 操作のみを指すものとして使用しています。

ここでは、パラレルDMLの次の項目について説明します。

- パラレルDMLを使用する場合
- パラレルDMLモードの有効化
- UPDATE、MERGEおよびDELETEのルール
- INSERT SELECTのルール
- パラレルDMLのトランザクション制限
- ロールバック・セグメント
- パラレルDMLのリカバリ
- パラレルDMLの領域に関する考慮事項
- パラレルDMLの制限
- データ整合性の制限
- トリガーの制限
- 分散トランザクションの制限
- 分散トランザクション並列処理の例
- UNION ALLの同時実行

8.5.3.1 パラレルDMLを使用する場合

パラレルDMLが役立つのは、大容量オブジェクトにアクセスする際のパフォーマンスとスケーラビリティが重要になる意思決定支援システム(DSS)環境です。パラレルDMLは、DSSデータベースのための問合せ機能と更新機能の両方を提供し、パラレル問合せを補います。

並列処理の設定に伴うオーバーヘッドのため、パラレルDML操作は短時間のOLTPトランザクションでの実行には適していません。ただし、パラレルDML操作を使用してOLTPデータベースでのバッチ・ジョブの実行を高速処理することができます。

パラレルDMLが使用されるシナリオの一部を次に示します。

- データ・ウェアハウス・システムでの表のリフレッシュ
- 中間サマリー表の作成
- スコアリング・テーブルの使用
- 履歴表の更新
- バッチ・ジョブの実行

8.5.3.1.1 データ・ウェアハウス・システムでの表のリフレッシュ

データ・ウェアハウス・システムでは、大容量の表を本番システムの新規データまたは変更データで定期的にリフレッシュ(更新)する必要があります。

MERGE文を使用すると、これを効率よく実行できます。

8.5.3.1.2 中間サマリー表の作成

DSS環境では、多くのアプリケーションで、多数の大容量中間サマリー表の作成や操作を行う複雑な計算が必要です。

このようなサマリー表は多くの場合は一時的に使用され、ロギングする必要がないものが大半です。パラレルDMLを使用すると、このような大容量中間表に対する操作が高速化されます。メリットの1つは、増分結果を中間表に格納して、パラレル更新を実行できることです。

また、サマリー表には、アプリケーション・セッションが終了しても保存する必要がある累積情報または比較情報が含まれる場合もあります。つまり、一時表は使用できません。パラレルDML操作を使用すると、このような大容量サマリー表に対する変更が高速化されます。

8.5.3.1.3 スコアリング・テーブルの使用

多くのDSSアプリケーションは、一連の基準に基づいて定期的に顧客をスコアリングしています。

通常、スコアは大容量のDSS表に格納されます。その後、スコア情報は意思決定(たとえばメーリング・リストへの指定)で使用されます。

このようなスコアリング・アクティビティでは、表の多数の行の問合せと更新が行われます。パラレルDMLを使用すると、このような大容量表に対する操作が高速化されます。

8.5.3.1.4 履歴表の更新

履歴表は、最新の時間間隔における企業のビジネス・トランザクションを示します。

DBAは定期的に、この表から最も古い行セットを削除し、新しい行セットを挿入します。パラレルINSERT SELECTおよびパラレルDELETE操作を使用すると、このロールオーバー・タスクが高速化されます。

パーティションの削除を使用して古い行を削除することもできます。ただし、表を日付でパーティション化して、適切な時間間隔を設定する必要があります。

8.5.3.1.5 バッチ・ジョブの実行

業務時間外にOLTPデータベースで実行されるバッチ・ジョブは、ジョブを完了する必要のある時間枠が決まっています。時間どおりに確実にジョブを完了するための最適な手段が、パラレルでの操作の実行です。

ワーク・ロードが増えると、コンピュータ・リソースを追加できます。パラレル操作のスケールアップ・プロパティにより時間制限を守ることができます。

8.5.3.2 パラレルDMLモードの有効化

DML文をパラレル化できるのは、セッションまたはSQL文におけるパラレルDMLを明示的に有効化した場合のみです。

セッションでこのモード有効にするには、次のSQL文を実行します。

```
ALTER SESSION ENABLE PARALLEL DML;
```

特定のSQL文でパラレルDMLモードを有効にするには、ENABLE_PARALLEL_DML SQLヒントを含めます。例:

```
INSERT /*+ ENABLE_PARALLEL_DML */ …
```

パラレルDMLとシリアルDMLでは、ロック、トランザクションおよびディスク領域の要件が異なるため、このモードが必要になり、パラレルDMLはデフォルト設定によりセッションに対して無効化されています。

パラレルDMLが無効になっていると、PARALLELヒントが使用されてもDMLはパラレルで実行されません。

パラレルDMLがセッションで有効化されていると、そのセッションのすべてのDML文がパラレル実行の対象とみなされます。ENABLE_PARALLEL_DMLヒントを使用してパラレルDMLがSQL文で有効になっている場合、その特定の文のみがパラレル実行の対象として考慮されます。ただし、パラレルDMLが有効になっていても、パラレル・ヒントがない場合や表にパラレル属性がない

場合、またはパラレル操作の制限に違反している場合、DML操作はシリアルで実行されます。

セッションのPARALLEL DMLモードはSELECT文、DDL文、およびDML文の問合せ部分の並列処理には影響しません。このモードが設定されていない場合、DML操作はパラレル化されませんが、DML文内のスキャン操作または結合操作はパラレル化されることがあります。

セッションに対してパラレルDMLモードが有効になっている場合、DISABLE_PARALLEL_DML SQLヒントを使用して、特定のSQL文のモードを無効にできます。

詳細は、「パラレルDMLの領域に関する考慮事項」および「パラレルDMLの制限」を参照してください。

8.5.3.3 UPDATE、MERGEおよびDELETEのルール

更新、マージまたは削除操作は、特定の条件下でのみパラレル化されます。

UPDATE、MERGEおよびDELETE操作は、次の条件のうち少なくとも1つが満たされている場合のみパラレル化されます。

- 前のCREATE TABLEまたはALTER TABLE文によって、更新、マージまたは削除の対象となる表にPARALLEL宣言が設定されている。
- DML文で文レベルまたはオブジェクト・レベルのPARALLELヒントが指定されている。
- 自動並列度(自動DOP)が有効になっている。
- ALTER SESSION FORCE PARALLEL DML文を使用してパラレルDMLが強制されている。

文に副問合せまたは更新可能なビューが含まれている場合は、問合せのパラレル化のルールに基づいて、それらもパラレルで実行される可能性があります。UPDATE、MERGEおよびDELETE部分のパラレル化の決定は、問合せ部分から独立しており、逆もまた同様です。文レベルのPARALLELヒントまたは自動DOPでは、DML部分と問合せ部分の両方がパラレル化されます。

関連項目:

- 自動DOPの詳細は、「自動並列度」を参照してください
- 更新、マージまたは削除操作について可能性のある制限の詳細は、「並列度の制限」を参照してください
- 並列度(DOP)を決定するルールの詳細は、「SQL文の並列度ルール」を参照してください
- ALTER SESSION SQL文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

8.5.3.4 INSERT SELECTのルール

挿入操作は、特定の条件下でのみパラレルで実行されます。

INSERT操作がパラレルで実行されるのは、次の条件のうち少なくとも1つが満たされている場合のみです。

- 前のCREATE TABLEまたはALTER TABLE文によって、挿入の対象となる表(参照オブジェクト)にPARALLEL宣言が設定されている。
- DML文でINSERTの後に文レベルまたはオブジェクト・レベルのPARALLELヒントが指定されている。
- 自動並列度(自動DOP)が有効になっている。
- ALTER SESSION FORCE PARALLEL DML文を使用してパラレルDMLが強制されている。

INSERT操作のパラレル化の決定は、SELECT操作とは別に行われます。逆の場合も同様です。問合せのパラレル化のルールに

基づいて、SELECT操作をパラレル化できます。文レベルのPARALLELヒントまたは自動DOPでは、INSERT操作とSELECT操作の両方がパラレル化されます。

関連項目:

- 自動DOPの詳細は、「自動並列度」を参照してください
- 並列度(DOP)を決定するルールの詳細は、「SQL文の並列度ルール」を参照してください
- ALTER SESSION SQL文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

8.5.3.5 パラレルDMLのトランザクション制限

DML操作をパラレルで実行するには、パラレル実行コーディネータがパラレル実行サーバーを取得し、パラレル実行サーバーがそれぞれのパラレル・プロセス・トランザクションで作業の一部を実行します。

次の条件に注意してください。

- 各パラレル実行サーバーは異なるパラレル・プロセス・トランザクションを生成します。
- 自動UNDO管理のかわりにロールバック・セグメントを使用する際は、同じロールバック・セグメントに存在するパラレル・プロセス・トランザクションの数を制限することによって、ロールバック・セグメントに対する競合を減らした方がよい場合があります。詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

コーディネータにも独自のコーディネータ・トランザクションがあり、独自のロールバック・セグメントが対応しています。ユーザーレベルのトランザクション原子性を確保するには、コーディネータは2フェーズのコミット・プロトコルを使用して、パラレル・プロセス・トランザクションで実行される変更をコミットします。

パラレルDMLが有効になっているセッションでは、セッションのトランザクションが特殊なモードになる場合があります。トランザクションで任意のDML文が表をパラレルで変更した後、そのトランザクションでは同じ表に対してシリアル問合せ、パラレル問合せまたはDML文で再びアクセスすることはできません。パラレル変更の結果は、そのトランザクション内では確認できません。

同じトランザクション内でパラレルで変更された表に、シリアル文またはパラレル文でアクセスしようとすると、拒否され、エラー・メッセージが返されます。

パラレルDMLが有効になっているセッションでPL/SQLプロシージャまたはブロックが実行される場合、プロシージャまたはブロックの文にはこのルールが適用されます。

8.5.3.6 ロールバック・セグメント

自動UNDO管理のかわりにロールバック・セグメントを使用する場合、パラレルDMLを使用するときにいくつかの制限があります。

関連項目:

パラレルDMLおよびロールバック・セグメントの制限の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

8.5.3.7 パラレルDMLのリカバリ

パラレルDML操作をロールバックするために必要な時間は、フォワード操作の実行にかかる時間とほぼ同じです。

Oracle Databaseでは、トランザクションとプロセスが失敗した後、およびインスタンスとシステムで障害が発生した後のパラレル・ロールバックをサポートしています。Oracle Databaseでは、トランザクション・リカバリのロールフォワード・ステージとロールバック・ステージの両方をパラレル化できます。

関連項目:

パラレル・ロールバックの詳細は、『Oracle Databaseバックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください

8.5.3.7.1 ユーザー発行のロールバックでのトランザクション・リカバリ

文のエラーのためにトランザクションが失敗した際のユーザー発行のロールバックは、パラレル実行コーディネータとパラレル実行サーバーによってパラレルで実行されます。

このロールバックには、フォワード・トランザクションとほぼ同じ時間がかかります。

8.5.3.7.2 プロセス・リカバリ

パラレル実行コーディネータまたはパラレル実行サーバーの障害からのリカバリは、PMONプロセスによって実行されます。

パラレル実行サーバーまたはパラレル実行コーディネータで障害が発生すると、PMONによってそのプロセスの作業がロールバックされ、トランザクションの他のすべてのプロセスによってその変更内容がロールバックされます。

8.5.3.7.3 システム・リカバリ

システム障害のリカバリでは新規起動が必要です。

リカバリは、SMONプロセスとSMONで生成されるすべてのリカバリ・サーバー・プロセスによって実行されます。パラレルDML文はパラレル・ロールバックを使用してリカバリできます。初期化パラメータCOMPATIBLEが8.1.3以上に設定されている場合は、ファスト・スタート・オン・デマンド・ロールバックによって、中止されたトランザクションを1回に1ブロックずつオンデマンドでリカバリできます。

8.5.3.8 パラレルDMLの領域に関する考慮事項

パラレルUPDATEではオブジェクト内の既存の空き領域が使用されますが、ダイレクト・パスINSERTではデータのために新しいエクステントが取得されます。

パラレル実行では複数の同時子トランザクションがオブジェクトを変更するため、領域使用の特徴がシリアル実行と異なることがあります。

8.5.3.9 パラレルDMLの制限

パラレルDMに適用されるいくつかの制限事項があります。

パラレルDML(ダイレクト・パスINSERTを含む)には次の制限が適用されます。

- UPDATE、MERGEおよびDELETE操作のイントラ・パーティション並列化を行うには、COMPATIBLE初期化パラメータを9.2以上に設定する必要があります。
- INSERT VALUES文はパラレルで実行されません。
- 1つのトランザクションには、様々な表を変更する複数のDML文が含まれる場合があります。ただし、パラレルDML文で1つの表が変更された後、後続のシリアル文またはパラレル文(DMLまたは問合せ)が、そのトランザクション内で同じ表に再度アクセスすることはできません。
    - シリアル・ダイレクト・パスINSERT文の後にもこのような制限があります。同じトランザクションでは、変更された

表に後続のSQL文(DMLまたは問合せ)がアクセスできません。

- 同じ表にアクセスする問合せが許可されるのは、パラレルDMLまたはダイレクト・パスINSERT文の前であり、後には許可されません。
- シリアル文またはパラレル文が、パラレルUPDATE、DELETEまたはMERGE、あるいはダイレクト・パスINSERTによって同じトランザクションで変更された表にアクセスしようとすると拒否され、エラー・メッセージが返されます。

- パラレルDML操作は、トリガーを含む表に対しては実行できません。
- レプリケーション機能はパラレルDMLに対してはサポートされません。
- パラレルDMLは、自己参照型整合性、削除カスケードおよび遅延整合性の制約がある場合には実行されません。また、ダイレクト・パスINSERTではすべての参照整合性がサポートされません。
- オブジェクト列を含む表に対してパラレルDMLを実行できますが、それはオブジェクト列にアクセスしない場合です。
- LOB列を含む表に対してパラレルDMLを実行できますが、それは表がパーティション化されている場合です。ただし、イントラ・パーティション並列処理はサポートされません。

  LOB列を含む非パーティション化表では、LOB列がSecureFiles LOBとして宣言される場合に、パラレルのINSERT操作がサポートされます。パラレルのUPDATE、DELETEおよびMERGE操作は、そのような表に対してはサポートされません。
- DML操作が分散トランザクションに含まれる場合、またはDMLまたは問合せ操作の対象がリモート・オブジェクトである場合、そのDML操作はパラレルで実行できません。
- クラスタ化表がサポートされていません。
- パラレルのUPDATE、DELETEおよびMERGE操作は、一時表に対してはサポートされません。
- 表がパーティション化されていない場合、パラレルDMLは、ビットマップ索引を使用した表でサポートされません。

これらの制限に違反すると文がシリアルで実行されます。警告またはエラー・メッセージは返されません(トランザクション内で同じ表にアクセスする文に関する制限を除きます。この場合はエラー・メッセージが生成されます)。

8.5.3.9.1 パーティション化キーの制限

パーティション表のパーティション化キーを新しい値に更新できるのは、更新によってその行が新しいパーティションに移動されない場合のみです。

表定義で行移動の句が有効になっている場合は更新が可能です。

8.5.3.9.2 関数の制限

パラレルDMLの関数の制限は、パラレルDDLおよびパラレル問合せの関数の制限と同じです。

詳細は、「関数のパラレル実行について」を参照してください。

8.5.3.10 データ整合性の制限

このトピックでは、整合性制約とパラレルDML文の相互作用について説明します。

この項の内容は次のとおりです。

- NOT NULLおよびCHECK
- UNIQUEおよびPRIMARY KEY
- FOREIGN KEY(参照整合性)

- 削除カスケード
- 自己参照型整合性
- 遅延可能整合性制約

8.5.3.10.1 NOT NULLおよびCHECK

このトピックでは、NOT NULLおよびCHECKの整合性制約について説明します。

NOT NULLおよびCHECKの整合性制約は許可されます。列レベルおよび行レベルそれぞれで施行されるため、パラレルDMLでは問題になりません。

8.5.3.10.2 UNIQUEおよびPRIMARY KEY

このトピックでは、UNIQUEおよびPRIMARY KEYの整合性制約について説明します。

UNIQUEおよびPRIMARY KEYの整合性制約は許可されます。

8.5.3.10.3 FOREIGN KEY(参照整合性)

参照整合性の制限が発生するのは、ある表に対するDML操作が別の表に対して再帰的なDML操作を引き起こす可能性があるときです。

整合性チェックを実行するために、変更対象のオブジェクトに行われるすべての変更を同時に確認する必要がある場合にも、この制限が適用されます。

表8-1に、参照整合性制約に関連する表に対して実行可能なすべての操作を示します。

表8-1 参照整合性の制限

DML文	親での発行	子での発行	自己参照型
INSERT	(該当せず)	パラレル化なし	パラレル化なし
MERGE	(該当せず)	パラレル化なし	パラレル化なし
UPDATE No Action	サポートされている	サポートされている	パラレル化なし
DELETE No Action	サポートされている	サポートされている	パラレル化なし
DELETE Cascade	パラレル化なし	(該当せず)	パラレル化なし

8.5.3.10.4 削除カスケード

このトピックでは、削除カスケード・データ整合性制限について説明します。

外部キーを含む表に対する削除カスケードを使用した削除はパラレル化されません。パラレル実行サーバーが、複数のパーティション(親表および子表)から行を削除しようとするためです。

8.5.3.10.5 自己参照型整合性

自己参照型整合性制約のある表に対するDMLは、参照されるキー(主キー)に関係する場合はパラレル化されません。

その他すべての列に対するDMLでは並列処理が可能です。

8.5.3.10.6 遅延可能整合性制約

このトピックでは、遅延可能整合性制約について説明します。

操作対象の表に遅延可能制約が適用される場合、DML操作はパラレルで実行されません。

8.5.3.11 トリガーの制限

影響する表に有効なトリガーが含まれており、DML文の結果によってトリガーが起動される可能性がある場合、そのDML操作はパラレルで実行されません。

つまり、レプリケート中の表に対するDML文はパラレル化されません。

表に対するDMLをパラレル化するには関連するトリガーを無効にしておく必要があります。トリガーを有効または無効にすると、従属する共有カーソルが無効化されます。

8.5.3.12 分散トランザクションの制限

このトピックでは、分散トランザクションの制限について説明します。

DML操作が分散トランザクションに含まれる場合、またはDMLまたは問合せ操作の対象がリモート・オブジェクトである場合、そのDML操作はパラレルで実行できません。

8.5.3.13 分散トランザクション並列処理の例

このトピックでは、分散トランザクション処理の例をいくつか説明します。

最初の例では、DML文がリモート・オブジェクトを問い合せます。DML操作はリモート・オブジェクトを参照しているため、通知なくシリアルで実行されます。

```
INSERT /*+ APPEND PARALLEL (t3,2) */ INTO t3 SELECT * FROM t4@dblink;
```

次の例では、DML操作がリモート・オブジェクトに適用されます。DELETE操作はリモート・オブジェクトを参照しているため、パラレル化されません。

```
DELETE /*+ PARALLEL (t1, 2) */ FROM t1@dblink;
```

最後の例では、DML操作が分散トランザクションに含まれます。DELETE操作は分散トランザクション(SELECT文によって開始される)で実行されるため、パラレルで実行されません。

```
SELECT * FROM t1@dblink;
 DELETE /*+ PARALLEL (t2,2) */ FROM t2;
 COMMIT;
```

8.5.3.14 UNION ALLの同時実行

UNIONやUNION ALLなどの集合演算子は、単一のSQL文に結合される複数の問合せ(分岐)で構成されます。

従来、集合演算子は順に処理されます。個々の分岐をシリアルまたはパラレルで処理できますが、一度に1つの分岐のみで順に処理されます。この方法は多くのユースケースを満たしますが、UNIONまたはUNION ALL文の複数の分岐の処理を同時に実行する状況があります。ほとんどの一般的な状況は、複数またはすべての分岐がリモートのSQL文である場合です。この状況では、すべての参加するリモート・システムの同時処理は、参加するシステムのワークロードを増やさずに処理時間全体の処理速度を上げるよう求められます。

UNIONまたはUNION ALL文の同時実行のデフォルト動作は、OPTIMIZER_FEATURES_ENABLE初期化パラメータの設定によって制御されます。12.1.0.1以上に設定すると、同時実行がデフォルトで有効になります。少なくとも1つの分岐の文がローカルで

パラレルの処理が考慮される文は、UNIONまたはUNION ALL文全体も同時に処理されます。システムは個々のローカル分岐の文のDOPを計算し、UNIONまたはUNION ALL文全体の実行のDOPの最大値を選択します。次に、システムは、パラレルで処理される分岐のパラレル化およびシリアルおよびリモート文の同時ワーカーに選択されたDOPを使用して、できるだけ多くの分岐を同時に処理します。

OPTIMIZER_FEATURES_ENABLE初期化パラメータが12.1.0.1より小さい値に設定されている場合、UNIONまたはUNION ALL文の同時実行は、PQ_CONCURRENT_UNIONヒントを使用して明示的に有効にする必要があります。

ただし、1つずつの分岐の連続処理とは異なり、同時処理は個々の分岐の結果の順序付けられた戻りを保証しません。1つずつの分岐の順序付けられた戻りが必要な場合、NO_PQ_CONCURRENT_UNIONヒントを使用して同時処理を無効にするか、SQL文を拡張して個々の分岐の文を一意に識別してその指定された識別子でソートする必要があります。

特にPQ_CONCURRENT_UNIONヒントを使用しないかぎり、シリアルまたはリモート分岐のみで構成されるUNIONまたはUNION ALL文は同時に処理されません。このSQL文のDOPは、最大でシリアルおよびリモート入力の数です。

UNIONまたはUNION ALL文の同時処理が発生するかどうかは、SQL文の実行計画で簡単に識別できます。パラレルで実行される場合、シリアルおよびリモート分岐の実行は、PX SELECTORとして識別可能な行ソースで管理されます。同時に処理されない文は、シリアルおよびリモート分岐のコーディネータとして問合せコーディネータ(QC)を示します。

例8-4では、SQL文はローカルおよびリモート分岐で構成されています。SQL文は、ローカル・データベースのゴールドおよびプラチナ顧客の情報およびリモート・データベースの3つの主要都市の顧客の情報をロードします。ローカルSELECT文がパラレルで発生するため、この処理はパラレルで自動的に実行されます。各シリアル分岐は、1つのパラレル実行サーバー・プロセスのみによって実行されます。各パラレル実行サーバーは1つのシリアル分岐を実行できるため、同時に実行されます。

例8-4 UNION ALLの実行計画

```
SQL> EXPLAIN PLAN FOR INSERT INTO all_customer
  SELECT * FROM GOLD_customer UNION ALL
  SELECT * FROM PLATINUM_customer UNION ALL
  SELECT * FROM SF_customer@san_francisco UNION ALL
  SELECT * FROM LA_customer@los_angeles UNION ALL
  SELECT * FROM LV_customer@las_vegas;
-------------------------------------------------------------------------------
| Id | Operation              | Name          | TQ/Ins |IN-OUT | PQ Distrib|
-------------------------------------------------------------------------------
| 0 | INSERT STATEMENT        |               |        |       |           |
| 1 |  LOAD TABLE CONVENTIONAL | ALL_CUSTOMER |        |       |           |
| 2 |   PX COORDINATOR        |               |        |       |           |
| 3 |    PX SEND QC (RANDOM)  | :TQ10003      |        | P->S  | QC (RAND) |
| 4 |     UNION-ALL           |               |        | PCWP  |           |
| 5 |      PX BLOCK ITERATOR  |               |        | PCWC  |           |
| 6 |       TABLE ACCESS FULL | GOLD_CUSTOMER|        | PCWP  |           |
| 7 |      PX BLOCK ITERATOR  |               |        | PCWC  |           |
| 8 |       TABLE ACCESS FULL | PLATINUM_CUST|        | PCWP  |           |
| 9 |      PX SELECTOR        |               |        | PCWP  |           |
|10 |       REMOTE            | SF_CUSTOMER  |        | PCWP  |           |
|11 |      PX SELECTOR        |               |        | PCWP  |           |
|12 |       REMOTE            | LA_CUSTOMER  |        | PCWP  |           |
|13 |      PX SELECTOR        |               |        | PCWP  |           |
|14 |       REMOTE            | LV_CUSTOMER  |        | PCWP  |           |
-------------------------------------------------------------------------------
```

8.5.4 関数のパラレル実行について

SQL文には、PL/SQLまたはJavaで作成したユーザー定義関数、あるいはC言語で作成した外部プロシージャとしてのユーザー定義関数を含めることができます。これらは、SELECTリスト、SET句またはWHERE句に指定できます。

SQL文がパラレル化されると、このような関数はパラレル実行サーバー・プロセスによって1行ずつ実行されます。関数で使用されるPL/SQLパッケージ変数またはJava静的属性は、完全に個々のパラレル実行プロセスのみで使用され、元のセッションからコピーされるのではなく各行が処理されるたびに新たに初期化されます。この処理のため、パラレルで実行した場合、すべての関数によって正しい結果が生成されるとはかぎりません。

ユーザーが作成した表関数を文のFROMリストに指定できます。このような関数は、行出力を生成するためソース・テーブルのように機能します。表関数はその文で1回、各パラレル実行プロセスの開始時に初期化されます。すべての変数は、対応するパラレル実行プロセスでしか使用されません。

この項では、次の項目について説明します。

- パラレル問合せでの関数
- パラレルDMLおよびDDL文での関数

8.5.4.1 パラレル問合せでの関数

ユーザー関数は、SQL問合せ文またはDMLまたはDDL文の副問合せでパラレルで実行できます。

SELECT文、あるいはDMLまたはDDL文の副問合せでは、次の場合にユーザー作成関数がパラレルで実行されることがあります。

- PARALLEL_ENABLEキーワードで宣言された場合。
- パッケージまたはタイプで宣言され、PRAGMA RESTRICT_REFERENCES句にWNDS、RNPSおよびWNPSのすべてが指定された場合。
- CREATE FUNCTIONで宣言され、システムがPL/SQLコードの本体を分析して、コードがデータベースに書き込まず、パッケージ変数の読取りや変更も行わないことを判別できた場合。

関数の実行をシリアルで行うことが必要な場合でも、問合せまたは副問合せの他の部分はパラレルで実行されることがあります。

関連項目:

- PRAGMA RESTRICT_REFERENCES句の詳細は、『Oracle Database開発ガイド』を参照してください
- CREATE FUNCTION文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

8.5.4.2 パラレルDMLおよびDDL文での関数

ユーザー関数は、特定の条件下でDMLまたはDDL文で実行できます。

パラレルDMLまたはDDL文においても、パラレル問合せと同じく、次の場合にユーザー作成関数がパラレルで実行されることがあります。

- PARALLEL_ENABLEキーワードで宣言された場合。
- パッケージまたはタイプで宣言され、PRAGMA RESTRICT_REFERENCES句にRNDS、WNDS、RNPSおよびWNPSのすべてが指定された場合。
- CREATE FUNCTION文で宣言され、システムがPL/SQLコードの本体を分析して、コードがデータベースに読取りや書込みを行わず、パッケージ変数の読取りや変更も行わないことを判別できた場合。

パラレルDML文の場合、パラレルで実行できない関数コールが含まれていると、そのDML文全体がシリアルで実行されます。

INSERT SELECT文またはCREATE TABLE AS SELECT文の場合は、ここで説明したパラレル問合せルールに従って、問合せ部分の関数コールはパラレル化されます。文のその他の部分をシリアルで実行する必要があっても、問合せはパラレル化できます。または、その逆の場合もあります。

8.5.5 その他の並列処理の種類について

Oracle Databaseは、複数の操作タイプでパラレル処理を使用できます。

パラレルSQL実行の他に、Oracle Databaseは次の操作について並列処理を使用できます。

- パラレル・リカバリ
- パラレル伝播(レプリケーション)
- パラレル・ロード(外部表およびSQL*Loaderユーティリティ)

パラレルSQLと同じく、 パラレル・リカバリ、パラレル伝播およびパラレル外部表ロードは、パラレル実行コーディネータと複数のパラレル実行サーバーによって実行されます。ただし、SQL*Loaderを使用したパラレル・ロードでは異なるメカニズムが使用されます。

パラレル実行コーディネータとパラレル実行サーバーの動作は、実行する操作の種類(SQL、リカバリまたは伝播)に応じて変わることがあります。たとえば、プールのすべてのパラレル実行サーバーが占有されており、最大数のパラレル実行サーバーが起動されている場合は、次のようになります。

- パラレルSQLおよび外部表ロードでは、パラレル実行コーディネータがシリアル処理に切り替えます。
- パラレル伝播では、パラレル実行コーディネータがエラーを返します。

1つのセッションで、パラレル実行コーディネータは1種類の操作のみを調整します。たとえば、パラレル実行コーディネータは、パラレルSQLとパラレル・リカバリまたはパラレル伝播を同時に調整することはできません。

関連項目:

- パラレル・ロードおよびSQL*Loaderの詳細は、『Oracle Databaseユーティリティ』を参照してください
- パラレル・メディア・リカバリの詳細は、『Oracle Databaseバックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください
- パラレル・インスタンス・リカバリの詳細は、『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

8.5.6 SQL文の並列度ルール

SQL文のパラレル化決定には、パラレル化の決定と並列度(DOP)という2つの要素があります。

これらの要素の決定方法は、問合せ、DDL操作およびDML操作で異なります。

次の各項で、並列度について説明します。

- 問合せのパラレル化のルール
- DDL文のルール
- CREATE TABLE AS SELECTのルール
- UPDATE、MERGEおよびDELETEのルール

- INSERT SELECTのルール

各種SQL文の並列度は、文レベルまたはオブジェクト・レベルのPARALLELヒント、PARALLEL句、ALTER SESSION FORCE PARALLEL文、自動並列度(自動DOP)、または表や索引のPARALLEL宣言によって決定できます。これらの方法のうち複数を使用する場合、Oracle Databaseでは、優先順位ルールを使用して、DOPの決定に使用する方法が決定されます。

表8-2には、各種SQL文の並列度(DOP)を決定するための優先順位ルールを示します。この表では、方法の優先順位番号が小さいほうが、大きい番号よりも優先されます。たとえば、優先度(1)は、優先度(2)、優先度(3)、優先度(4)および優先度(5)よりも優先されます。

表8-2 パラレル化の優先順位

パラレル操作	文レベルのPARALLELヒント	オブジェクト・レベルのPARALLELヒント	PARALLEL句	ALTER SESSION	自動DOP	パラレル宣言
パラレル問合せ表/索引スキャン。詳細は、問合せのパラレル化のルールを参照してください。	優先度(1)	優先度(2)	該当なし	優先度(3) FORCE PARALLEL QUERY	優先度(4)	優先度(5)
パラレル UPDATE、DELETE または MERGE。詳細は、UPDATE、MERGE および DELETE のルールを参照してください。	優先度(1)	優先度(2)	該当なし	優先度(3) FORCE PARALLEL DML	優先度(4)	優先度(5)のターゲット表
パラレル INSERT... SELECT の INSERT 操作。詳細は、INSERT SELECT のルールを参照してください。	優先度(1)	優先度(2)	該当なし	優先度(3) FORCE PARALLEL DML	優先度(4)	挿入対象となる優先度(5)の表
INSERT がシリアルの場合の、INSERT SELECT の SELECT 操作。詳細は、INSERT SELECT のルールを参照してください。	優先度(1)	優先度(2)	該当なし	優先度(3) FORCE PARALLEL QUERY	優先度(4)	選択対象となる優先度(5)の表
パラレル CREATE TABLE AS SELECT の CREATE 操作(パーティション表または非パーティション表) 詳細は、CREATE	優先度(1)	該当なし	優先度(4)	優先度(2) FORCE PARALLEL DDL	優先度(3)	該当なし

パラレル操作	文レベルのPARALLELヒント	オブジェクト・レベルのPARALLELヒント	PARALLEL句	ALTER SESSION	自動DOP	パラレル宣言
TABLE AS SELECTのルールを参照してください。						
CREATE がシリアルの場合の、CREATE TABLE AS SELECT の SELECT 操作。詳細は、CREATE TABLE AS SELECT のルールを参照してください。	優先度(1)	優先度(2)	該当なし	優先度(3) FORCE PARALLEL QUERY	優先度(4)	優先度(5)
その他の DDL 操作。詳細は、DDL 文のルールを参照してください。	該当なし	該当なし	優先度(3)	優先度(1) FORCE PARALLEL DDL	優先度(2)	該当なし

関連項目:

- PARALLELヒントの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- PARALLEL句の詳細は、『Oracle Database SQL言語リファレンス』を参照してください
- ALTER SESSION SQL文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

8.6 パラレル実行のためのパラメータの初期化とチューニングについて

パラメータを使用して、パラレル実行を初期設定およびチューニングできます。

Oracle Databaseでは、データベース起動時のCPU_COUNTとPARALLEL_THREADS_PER_CPUの値に基づいて、パラレル実行パラメータのデフォルトを計算します。パラメータを手動でチューニングして、特定のシステム構成またはパフォーマンス目標に合うように値を増減することもできます。たとえば、パラレル実行がまったく使用されないシステムではPARALLEL_MAX_SERVERSを0に設定できます。

パラレル実行パラメータを手動でチューニングすることもできます。パラレル実行はデフォルトで有効になっています。

パラレル実行の初期設定およびチューニングについては、次のトピックで説明します。

- デフォルトのパラメータ設定
- セッションでのパラレル実行の強制
- パラレル実行のための一般的なパラメータのチューニング

8.6.1 デフォルトのパラメータ設定

Oracle Databaseでは、パラレル実行パラメータがデフォルトで自動設定されます。

パラレル実行のパラメータを表8-3に示します。

表8-3パラメータとデフォルト値

パラメータ	デフォルト	コメント
PARALLEL_ADAPTIVE_MULTI_USER	FALSE	パラレル実行により SQL の並列度(DOP)リクエストの数を制限し、システムのオーバーロードを回避します。 PARALLEL_ADAPTIVE_MULTI_USER は、Oracle Database 12c リリース 2 (12.2.0.1)で非推奨となり、将来のリリースではサポートされなくなります。かわりに、パラレル文のキューイングを使用することをお薦めします。
PARALLEL_DEGREE_LIMIT	CPU	自動 DOP が使用された場合に、文に許可される DOP の最大値を制御します。最大 DOP は、次のとおりです。 SUM(CPU_COUNT)*PARALLEL_THREADS_PER_CPU PARALLEL_DEGREE_LIMIT の値 AUTO には、値 CPU と同じ機能があります。
PARALLEL_DEGREE_POLICY	MANUAL	自動 DOP、パラレル文のキューイングおよびインメモリー・パラレル実行を使用するかどうかを制御します。デフォルトでは、これらの機能はすべて無効化されています。

パラメータ	デフォルト	コメント
PARALLEL_EXECUTION_MESSAGE_SIZE	16 KB	パラレル実行サーバー、およびパラレル実行サーバーと問合せコーディネータの通信に使用されるバッファのサイズを指定します。これらのバッファは、共有プールの中から割り当てられます。
PARALLEL_FORCE_LOCAL	FALSE	パラレル実行を現在の Oracle RAC インスタンスに制限します。
PARALLEL_INSTANCE_GROUP	なし。デフォルトでは、現在アクティブなすべてのインスタンスでパラレル実行が有効になっています。	パラレル問合せ操作をインスタンスの制限数に制限できます。サービスおよび非推奨のパラメータ INSTANCE_GROUPS とともに使用します。
PARALLEL_MAX_SERVERS	PARALLEL_MAX_SERVERS を参照してください。	1 インスタンスに対するパラレル実行プロセスとパラレル・リカバリ・プロセスの最大数が指定されます。需要が増加するにつれて、インスタンスの起動時に作成された数からこの値までプロセス数が増加します。 このパラメータの設定が低すぎると、問合せが処理中に十分なパラレル実行プロセスを得られない場合があります。設定が高すぎると、ピーク時にメモリー・リソース不足が発生してパフォーマンスが低下する可能性があります。
PARALLEL_MIN_DEGREE	1	自動並列度によって計算される最小並列度を制御します。
PARALLEL_MIN_SERVERS	CPU_COUNT * PARALLEL_THREADS_PER_CPU * 2	Oracle Database の起動時に、パラレル実行のために起動および予約するパラレル実行プロセス数を指定します。この設定の値を大きくすると、パラレル文の起動コストを均衡化するのに役立ちますが、パラレル実行プロセスはデータベースが停止されるまで削除されないため、必要なメモリー使用量は増大します。
PARALLEL_MIN_PERCENT	0	パラレル実行に必要なリクエストされたパラレル実行プロセスの最小パーセンテージを指定します。デフォルト値は 0 で、使用可能なパラレル・サーバー・プロセスがない場合、パラレル文はシリアルで実行されます。
PARALLEL_MIN_TIME_THRESHOLD	AUTO	オプティマイザによって見積もられた実行時間を指定します。これより大きい値の場合、文は自動パラレル問合せおよび自動 DOP 導出の候補となります。AUTO の解釈は、データベース・インメモリーが使用されているかどうかによって異なります。
PARALLEL_SERVERS_TARGET	PARALLEL_SERVERS_TARGET を参照してください。	パラレル文のキューイングが使用されるまでに問合せを実行するのに使用可能なパラレル実行サーバー・プロセスの数を指定します。文のキューイングは、PARALLEL_DEGREE_POLICY が AUTO

パラメータ	デフォルト	コメント
		に設定されている場合にのみアクティブ化されることに注意してください。
PARALLEL_THREADS_PER_CPU	1	パラレル実行中に CPU が処理できるパラレル実行プロセスまたはスレッドの数を示します。

一部のパラメータの設定方法によってはOracle Databaseが制約を受けます。たとえば、PROCESSESを20に設定すると、25個の子プロセスを取得できなくなります。

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

8.6.2 セッションでのパラレル実行の強制

セッションでのパラレル実行を強制できます。

パラレルで実行する必要があるが、表に対するDOPの設定または関連する問合せの変更を避けたい場合は、次の文を使用して並列処理を強制できます。

```
ALTER SESSION FORCE PARALLEL QUERY;
```

この後のすべての問合せは、制限に違反しないかぎりパラレルで実行されます。DML文およびDDL文も強制できます。この句は、セッションの後続の文に指定されるすべてのパラレル句よりも優先されますが、パラレル・ヒントに対しては優先されません。

たとえば、一般的なOLTP環境では表にはパラレルの設定がありませんが、毎晩バッチ・スクリプトを使用してこのような表からデータをパラレルで収集する場合があります。セッション中にDOPを設定することで、ユーザーは、各表をパラレルで変更してから、終了時にシリアルに戻す必要がなくなります。

8.6.3 パラレル実行のための一般的なパラメータのチューニング

このトピックでは、パラレル実行の一般的なパラメータのチューニングについて説明します。

この項の内容は次のとおりです。

- パラレル操作のリソース制限を設定するパラメータ
- リソース消費に影響するパラメータ
- I/Oに関連するパラメータ

8.6.3.1 パラレル操作のリソース制限を設定するパラメータ

リソース制限を決定するための初期化パラメータを設定できます。

リソース制限を設定するパラメータを次で説明します。

- PARALLEL_FORCE_LOCAL
- PARALLEL_MAX_SERVERS

- PARALLEL_MIN_PERCENT
- PARALLEL_MIN_SERVERS
- PARALLEL_MIN_TIME_THRESHOLD
- PARALLEL_SERVERS_TARGET
- SHARED_POOL_SIZE
- メッセージ・バッファの追加メモリー要件
- 処理開始後のメモリー使用状況の監視

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

8.6.3.1.1 PARALLEL_FORCE_LOCAL

PARALLEL_FORCE_LOCALパラメータは、パラレルで実行されるSQL文がOracle RAC環境のシングル・インスタンスに制限されるかどうかを示します。

このパラメータをTRUEに設定することにより、問合せコーディネータが実行しているシングルOracle RACインスタンスのために処理されるパラレル・スレーブのスコープを制限できます。

PARALLEL_FORCE_LOCALパラメータの推奨値はFALSEです。

関連項目:

PARALLEL_FORCE_LOCAL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

8.6.3.1.2 PARALLEL_MAX_SERVERS

PARALLEL_MAX_SERVERSパラメータには、1つのインスタンスに対するパラレル実行プロセスおよびパラレル・リカバリ・プロセスの最大数を指定します。

需要が増加するにつれて、インスタンスの起動時に作成された数からこの値までプロセス数が増加します。

たとえば、この値を64に設定した場合、各問合せが2つのワーカー・セットを使用し、各セットのDOPが8とすると、4つのパラレル問合せを同時に実行できるようになります。

ユーザーのプロセスが多すぎる場合

同時ユーザーの問合せサーバー・プロセスが多すぎると、メモリー競合(ページング)、I/O競合または過剰なコンテキストのスイッチングが発生することがあります。

この競合により、パラレル実行が使用されなかった場合のレベルよりもシステム・スループットが低下する可能性があります。PARALLEL_MAX_SERVERS値を増やすのは、それによって生成されるロードのための十分なメモリーおよびI/O帯域幅がシステムにある場合のみにしてください。

オペレーティング・システムのパフォーマンス・モニタリング・ツールを使用すると、メモリー、スワップ領域およびI/O帯域幅の空き状況を判別できます。CPUとディスク両方の実行キューの長さ、およびシステムのI/O処理のサービス時間を調べます。プロセスを

追加する場合は、システムに十分なスワップ領域があることを確認します。問合せサーバー・プロセスの総数を制限すると、パラレル操作を実行できる同時ユーザー数が制限されることがありますが、システム・スループットは安定するようになります。

コンシューマ・グループを使用したユーザーに対するリソース数の制限時期

必要な場合、ユーザーに対してリソース・コンシューマ・グループを設定することで、所定のユーザーが使用可能な並列処理の量を制限できます。

これは、1ユーザーまたは1ユーザー・グループが使用できる、セッション数、同時ログオンおよびパラレル・プロセス数を制限するために行います。

パラレル実行文を処理する各問合せサーバー・プロセスは、セッションIDを使用してログオンしています。各プロセスは、ユーザーの同時セッションの制限に対してカウントされます。たとえば、あるユーザーが使用できるパラレル実行プロセスを10個に制限するには、ユーザーの制限を11に設定します。1プロセスがパラレル実行コーディネータ用、その他の10プロセスが2セットの問合せサーバーを構成します。こうすると、パラレル実行コーディネータが1セッション、パラレル実行プロセスが10セッションを使用できます。

関連項目:

- PARALLEL_MAX_SERVERS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
- ユーザー・プロファイルを使用したリソースの管理の詳細は、『Oracle Database管理者ガイド』を参照してください
- GV$ビューの問合せの詳細は、『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

8.6.3.1.3 PARALLEL_MIN_PERCENT

PARALLEL_MIN_PERCENTパラメータにより、使用中のアプリケーションに応じて、許容できるDOPを待機できます。

PARALLEL_MIN_PERCENTパラメータの推奨値は0です。このパラメータを0以外の値に設定すると、リクエストされたDOPがその時点でシステムによって実現されない場合、Oracle Databaseがエラーを返します。たとえば、PARALLEL_MIN_PERCENTを50(50パーセント)に設定したときに、マルチユーザー問合せ調整アルゴリズムまたはリソース制限によってDOPが50パーセントを超えて低減されると、Oracle DatabaseによってORA-12827が返されます。例:

```
SELECT /*+ FULL(e) PARALLEL(e, 8) */ d.department_id, SUM(SALARY)
  FROM employees e, departments d WHERE e.department_id = d.department_id
  GROUP BY d.department_id ORDER BY d.department_id;
```

Oracle Databaseから次のメッセージが返されます。

```
ORA-12827: insufficient parallel query slaves available
```

関連項目:

PARALLEL_MIN_PERCENT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

8.6.3.1.4 PARALLEL_MIN_SERVERS

PARALLEL_MIN_SERVERSパラメータは、1つのインスタンスで、パラレル操作のために起動および予約するプロセス数を指定します。

PARALLEL_MIN_SERVERSの設定により、メモリー使用量と起動コストのバランスを保つことができます。
PARALLEL_MIN_SERVERSを使用して起動されるプロセスは、データベースが停止されるまで終了しません。これによって、問合せが発行されるときには、多くの場合、プロセスが使用可能になっています。

関連項目:

PARALLEL_MIN_SERVERS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

8.6.3.1.5 PARALLEL_MIN_TIME_THRESHOLD

PARALLEL_MIN_TIME_THRESHOLDパラメータには、文が自動並列度の対象とみなされるまでの最小実行時間を指定します。

関連項目:

PARALLEL_MIN_TIME_THRESHOLD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

8.6.3.1.6 PARALLEL_SERVERS_TARGET

PARALLEL_DEGREE_POLICYパラメータは、パラレル文のキューイングが使用されるまでにパラレル文を実行できるパラレル・サーバー・プロセスの数を指定します。

PARALLEL_DEGREE_POLICYがAUTOに設定されていると、システムで現在使用されているパラレル・プロセスの数がPARALLEL_SERVERS_TARGET以上の場合、パラレル実行の必要な文はキューに入れられます。これはシステムで許可されているパラレル・サーバー・プロセスの最大数ではありません(それはPARALLEL_MAX_SERVERSによって制御されます)。ただし、PARALLEL_SERVERS_TARGETおよびパラレル文のキューイングを使用すると、パラレル実行の必要なそれぞれの文が必要なパラレル・サーバー・リソースに割り当てられ、パラレル・サーバー・プロセスの過多によるフラッドを防止できます。

関連項目:

PARALLEL_SERVERS_TARGET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

8.6.3.1.7 SHARED_POOL_SIZE

SHARED_POOL_SIZEパラメータは、共有プールのメモリー・サイズを指定します。

パラレル実行では、シリアルSQL実行で必要なメモリー・リソースに加えてさらにメモリー・リソースが必要です。追加のメモリーは、問合せサーバー・プロセスと問合せコーディネータ間での通信とデータの受渡しに使用されます。

Oracle Databaseによって、共有プールから問合せサーバー・プロセスにメモリーが割り当てられます。次のように共有プールをチューニングします。

- 共有プールの他のクライアント(共有カーソルやストアド・プロシージャなど)を考慮します。
- 大きな値ではマルチユーザー・システムでのパフォーマンスが向上しますが、小さな値ではメモリー使用量が減ることに注意します。
- その後、パラレル実行で使用されるバッファ数を監視し、shared pool PX msg poolと、ビューV$PX_PROCESS_SYSSTATの出力に示されている現在の最高水位標を比較します。

ノート:

使用可能な十分なメモリーがない場合は、エラー・メッセージ 12853(「PX バッファのメモリーが不足しています: 現在は *string*KB ですが、最大 *string*KB が必要です」)が生成されます。これは、PX バッファのための SGA メモリーが十分でない場合に発生します。少なくとも(MAX - CURRENT)バイトを追加するように SGA メモリーを再構成する必要があります。

デフォルトでは、Oracle Databaseはパラレル実行バッファを共有プールから割り当てます。

Oracle Databaseで起動時に次のエラーが表示された場合、SHARED_POOL_SIZEの値を、データベースが起動できるように十分に下げる必要があります。

```
ORA-27102: out of memory
SVR4 Error: 12: Not enough space
```

SHARED_POOL_SIZEの値を下げた後で、次のエラーが発生する場合があります。

```
ORA-04031: unable to allocate 16084 bytes of shared memory
   ("SHARED pool","unknown object","SHARED pool heap","PX msg pool")
```

その場合は、次の問合せを実行して、Oracle Databaseが16,084バイトを割り当てられなかった理由を判別します。

```
SELECT NAME, SUM(BYTES) FROM V$SGASTAT WHERE UPPER(POOL)='SHARED POOL'
  GROUP BY ROLLUP (NAME);
```

出力は次のようになります。

```
NAME                         SUM(BYTES)
---------------------------- ----------
PX msg pool                     1474572
free memory                      562132
                                2036704
```

SHARED_POOL_SIZEを指定したときに、保持するよう指定したメモリー容量がプールよりも大きい場合、Oracle Databaseでは、取得できるメモリーのすべてを割り当てません。一部の領域が残されます。問合せが実行されるとき、Oracle Databaseは必要なメモリーを取得しようとします。Oracle Databaseは560KBを使用し、失敗するとさらに16KBが必要になります。このエラーでは、必要とされた容量の累積は報告されません。必要な追加メモリー容量を判別する最適な方法としては、「メッセージ・バッファの追加メモリー要件」の式を使用します。

この例の問題を解決するには、SHARED_POOL_SIZEの値を増やします。サンプル出力に表示されるように、SHARED_POOL_SIZEは約2MBです。使用可能なメモリー容量によって異なりますが、SHARED_POOL_SIZEの値を4MBに増やしてから、データベースの起動を試行してください。Oracle Databaseで引き続きORA-4031メッセージが表示される場合は、起動が成功するまでSHARED_POOL_SIZEの値を次第に増やします。

関連項目:

SHARED_POOL_SIZE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

8.6.3.1.8 メッセージ・バッファの追加メモリー要件

このトピックでは、パラレル実行計画を使用する際のメッセージ・バッファおよびカーソル用の追加メモリー要件について説明します。

共有プールの初期設定を決定したら、メッセージ・バッファの追加メモリー要件を計算し、カーソル用に必要な追加領域の容量を決定する必要があります。

メッセージ・バッファに必要なメモリー

メッセージ・バッファを収容できるようにSHARED_POOL_SIZEパラメータの値を増やす必要があります。メッセージ・バッファによって、問合せサーバー・プロセスが相互に通信できます。

Oracle Databaseでは、プロデューサ問合せサーバーとコンシューマ問合せサーバーの間の仮想接続ごとに一定数のバッファが使用されます。接続数は、DOPの増加に合せてその2乗の数に増加します。この理由から、パラレル実行で使用されるメモリーの最大容量は、システムで許可されるDOPの最大値によって制限されます。この値を制御するには、PARALLEL_MAX_SERVERSパラメータを使用するか、ポリシーとプロファイルを使用します。

必要なメモリー容量を計算するには、次のいずれかの計算式を使用します。

- SMPシステム:

```
mem in bytes = (3 x size x users x groups x connections)
```

- Oracle Real Application ClustersおよびMPPシステム:

```
mem in bytes = ((3 x local) + (2 x remote)) x (size x users x groups)
  / instances
```

各インスタンスで、この式で計算されたメモリーが使用されます。

用語の意味は次のとおりです。

- SIZE = PARALLEL_EXECUTION_MESSAGE_SIZE
- USERS = DOPが最適な場合に実行が予想される同時パラレル実行ユーザー数
- GROUPS = 各問合せで使用される問合せサーバー・プロセス・グループ数

  単純なSQL文で必要になるのは1グループのみです。ただし、問合せに関連する副問合せがパラレルで処理される場合、Oracle Databaseはもう1つの問合せサーバー・プロセス・グループを使用します。

- CONNECTIONS = (DOP^2 + 2 x DOP)

  システムがクラスタまたはMPPの場合は、インスタンス数を考慮する必要があります。インスタンス数によってDOPが増加するためです。つまり、2つのインスタンス・クラスタでDOPを4にすると、結果としてDOPが8になります。控えめな見積りとして、初期値にはPARALLEL_MAX_SERVERSとインスタンス数を掛けて4で割った値を使用することをお薦めします。

- LOCAL = CONNECTIONS/INSTANCES
- REMOTE = CONNECTIONS - LOCAL

共有プールの元の設定にこの容量を追加します。ただし、これらのメモリー構造体いずれかの値を設定する前に、次の項で説明するカーソル用の追加メモリーも考慮する必要があります。

カーソルの追加メモリー

パラレル実行計画は、シリアル実行計画よりもSQL領域を多く消費します。共有プール・リソース使用状況を定期的に監視して、メッセージとカーソルの両方で使用されるメモリーがシステムの処理要件を満たすことを確認します。

8.6.3.1.9 処理開始後のメモリー使用状況の監視

自動チューニングと手動チューニングのどちらを使用するにしても、実行時の使用状況を監視して、メモリーのサイズが大きすぎないか、小さすぎないかを確認する必要があります。

この項の計算式はあくまで開始値を設定するためのものです。正しいメモリー・サイズを確認するため、次の問合せを使用して共有プールをチューニングします。

```
SELECT POOL, NAME, SUM(BYTES) FROM V$SGASTAT WHERE POOL LIKE '%pool%'
  GROUP BY ROLLUP (POOL, NAME);
```

出力は次のようになります。

```
POOL        NAME                           SUM(BYTES)
----------- -------------------------- ----------
shared pool Checkpoint queue                38496
shared pool KGFF heap                        1964
shared pool KGK heap                         4372
shared pool KQLS heap                     1134432
shared pool LRMPD SGA Table                 23856
shared pool PLS non-lib hp                   2096
shared pool PX subheap                     186828
shared pool SYSTEM PARAMETERS               55756
shared pool State objects                 3907808
shared pool character set memory            30260
shared pool db_block_buffers               200000
shared pool db_block_hash_buckets           33132
shared pool db_files                       122984
shared pool db_handles                      52416
shared pool dictionary cache               198216
shared pool dlm shared memory             5387924
shared pool event statistics per sess      264768
shared pool fixed allocation callback        1376
shared pool free memory                   26329104
shared pool gc_*                             64000
shared pool latch nowait fails or sle        34944
shared pool library cache                 2176808
shared pool log_buffer                       24576
shared pool log_checkpoint_timeout           24700
shared pool long op statistics array         30240
shared pool message pool freequeue         116232
shared pool miscellaneous                  267624
shared pool processes                       76896
shared pool session param values            41424
shared pool sessions                       170016
shared pool sql area                      9549116
shared pool table columns                  148104
shared pool trace_buffers_per_process     1476320
shared pool transactions                    18480
shared pool trigger inform                  24684
shared pool                               52248968
                                          90641768
```

出力に表示されたメモリー使用量を評価し、処理ニーズに基づいてSHARED_POOL_SIZEの設定を変更します。

メモリー使用量の統計をさらに取得するには、次の問合せを実行します。

```
SELECT * FROM V$PX_PROCESS_SYSSTAT WHERE STATISTIC LIKE 'Buffers%';
```

出力は次のようになります。

```
STATISTIC                       VALUE
------------------              -----
Buffers Allocated               23225
Buffers Freed                   23225
Buffers Current                     0
Buffers HWM                      3620
```

使用されたメモリー容量は、Buffers CurrentとBuffers HWMの統計に表示されます。バッファ数にPARALLEL_EXECUTION_MESSAGE_SIZEの値を掛けた値(バイト数)を計算します。この最高水位標をパラレル実行メッセージ・プール・サイズと比較して、割当てメモリーが多すぎないかを判別します。たとえば、最初の出力で、px msg poolに示されるラージ・プールの値は38,092,812すなわち38MBです。2番目の出力のBuffers HWMは3,620です。これにパラレル実行メッセージ・サイズの4,096を掛けると、14,827,520すなわち約15MBになります。この場合、最高水位標は容量の約40パーセントに達しています。

8.6.3.2 リソース消費に影響するパラメータ

このトピックでは、リソース消費に影響するパラメータについて説明します。

> ノート:
>
> 次の項を検討する前に、詳細について MEMORY_TARGET および MEMORY_MAX_TARGET 初期化パラメータの説明を参照してください。MEMORY_TARGET によって SGA コンポーネントおよび PGA コンポーネントが自動チューニングされるため、PGA_AGGREGATE_TARGET 初期化パラメータを設定する必要はありません。

ここで説明するパラメータの最初のグループは、すべてのパラレル操作(特にパラレル実行)のメモリーとリソースの消費に影響します。これらのパラメータは次のとおりです。

- PGA_AGGREGATE_TARGET
- PARALLEL_EXECUTION_MESSAGE_SIZE

パラメータの2つ目のサブセットについては、「パラレルDMLおよびパラレルDDLのリソース消費に影響するパラメータ」で説明しています。

リソース消費を制御するには、次の2つのレベルでメモリーを構成する必要があります。

- データベース・レベル。データベース・システムがオペレーティング・システムの適切な容量のメモリーを使用できるようにします。
- オペレーティング・システム・レベル(一貫性のため)。

  プラットフォームによっては、使用可能な仮想メモリーの合計容量(全プロセスの合計)を制御するオペレーティング・システム・パラメータを設定する必要があります。

データ・ウェアハウス操作で使用されるメモリーの大きな部分は(OLTPと比べて)動的に使用されます。このメモリーはプロセス・グローバル領域(PGA)から割り当てられ、プロセス・メモリーのサイズとプロセスの数はどちらも幅広い範囲で変化する可能性があります。そのような場合は、PGA_AGGREGATE_TARGET初期化パラメータを使用して、プロセス・メモリーとプロセス数の両方を制御します。PGA_AGGREGATE_TARGETとMEMORY_TARGETを合せて明示的に設定すると、自動チューニングは行われますが、PGA_AGGREGATE_TARGETが指定値未満になるようにチューニングされることはありません。

関連項目:

- MEMORY_TARGETおよびMEMORY_MAX_TARGET初期化パラメータの詳細は、『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
- MEMORY_TARGETおよびMEMORY_MAX_TARGET初期化パラメータの使用の詳細は、『Oracle Database管理者ガイド』を参照してください

8.6.3.2.1 PGA_AGGREGATE_TARGET

PGA_AGGREGATE_TARGETなどの初期化パラメータの設定により、自動PGAメモリー管理を有効にできます。

自動PGAメモリー管理を有効にすると、PGAメモリーの割当て方法が単純化されて強化されます。このモードでは、DBAによって明示的に設定されたPGAメモリー全体の目標に基づいて、Oracle DatabaseがPGAメモリーの作業領域専用部分のサイズを動的に調整します。自動PGAメモリー管理を有効にするには、PGA_AGGREGATE_TARGET初期化パラメータを設定する必要があります。新しいインストールでは、PGA_AGGREGATE_TARGETおよびSGA_TARGETは自動的にDatabase Configuration Assistant (DBCA)によって設定されます。MEMORY_TARGETは0です。つまり、自動メモリー管理は無効になっています。したがって、集計PGAの自動チューニングはデフォルトで有効になっています。ただし、集計PGAは、MEMORY_TARGETを0以外の値に設定して自動メモリー管理を有効にしないかぎり、増えることはありません。

関連項目:

- PGA_AGGREGATE_TARGET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
- 様々なシナリオでのPGA_AGGREGATE_TARGET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

8.6.3.2.1.1 HASH_AREA_SIZE

このパラメータは現在非推奨になっています。

HASH_AREA_SIZEは非推奨です。かわりにPGA_AGGREGATE_TARGETを使用してください。詳細は、PGA_AGGREGATE_TARGETを参照してください。

8.6.3.2.1.2 SORT_AREA_SIZE

このパラメータは現在非推奨になっています。

SORT_AREA_SIZEは非推奨です。かわりにPGA_AGGREGATE_TARGETを使用してください。詳細は、PGA_AGGREGATE_TARGETを参照してください。

8.6.3.2.2 PARALLEL_EXECUTION_MESSAGE_SIZE

PARALLEL_EXECUTION_MESSAGE_SIZEパラメータでは、パラレル実行メッセージで使用されるバッファのサイズを指定します。

PARALLEL_EXECUTION_MESSAGE_SIZEのデフォルト値は、オペレーティング・システム固有ですが、通常、16Kです。この値はほとんどのアプリケーションで適切です。

関連項目:

PARALLEL_EXECUTION_MESSAGE_TIME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

8.6.3.2.3 パラレルDMLおよびパラレルDDLのリソース消費に影響するパラメータ

このトピックでは、パラレルDMLおよびパラレルDDL操作のリソース消費に影響するパラメータについて説明します。

パラレルDMLおよびパラレルDDLのリソース消費に影響するパラメータを次に示します。

- TRANSACTIONS
- FAST_START_PARALLEL_ROLLBACK
- DML_LOCKS

パラレルの挿入、更新および削除操作では、シリアルDML操作よりも多くのリソースが必要です。同じく、PARALLEL CREATE TABLE AS SELECTおよびPARALLEL CREATE INDEXでも、より多くのリソースが必要となることがあります。このため、場合によっては、さらにいくつかの初期化パラメータの値を増やす必要があります。これらのパラメータは問合せのためのリソースには影響しません。

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

8.6.3.2.3.1 TRANSACTIONS

TRANSACTIONSパラメータは、パラレルDMLおよびDDLでのトランザクション数に影響します。

パラレルのDMLおよびDDLでは、各問合せサーバー・プロセスがトランザクションを開始します。パラレル実行コーディネータは2フェーズのコミット・プロトコルを使用してトランザクションをコミットします。このため、処理されるトランザクション数はDOPに応じて増加します。この結果、TRANSACTIONS初期化パラメータの値を増やす必要が生じることがあります。

TRANSACTIONSパラメータでは、同時トランザクションの最大数を指定します。TRANSACTIONSのデフォルト値は、並列なしを想定しています。たとえば、DOPが20の場合は、追加の新しいサーバー・トランザクション20(サーバー・セットが2つの場合は40)とコーディネータ・トランザクション1が生成されます。この場合、トランザクションが同一インスタンスで実行されているときは、TRANSACTIONSに21(または41)を加えた値に増やします。このパラメータを設定しない場合は、Oracle Databaseによって値が1.1 x SESSIONSになるように設定されます。サーバー管理のUNDOを使用している場合、この説明は適用されません。

8.6.3.2.3.2 FAST_START_PARALLEL_ROLLBACK

コミットされていないパラレルDMLまたはパラレルDDLトランザクションがあるときにシステムで障害が発生した場合に、FAST_START_PARALLEL_ROLLBACKパラメータを使用して、起動時のトランザクションのリカバリを高速化できます。

FAST_START_PARALLEL_ROLLBACKパラメータは、終了したトランザクションをリカバリするときに使用するDOPを制御します。異常終了したトランザクションとは、システム障害の前にアクティブだったトランザクションです。デフォルトでは、最大でCPU_COUNTパラメータ値の2倍になるようにDOPが選択されます。

このデフォルトDOPが不十分な場合は、パラメータをHIGHに設定します。これにより、最大DOPがCPU_COUNTパラメータの4倍になります。この機能はデフォルトで使用できます。

8.6.3.2.3.3 DML_LOCKS

DML_LOCKSパラメータは、パラレルDML操作によって保持されるロック数を含むように設定する必要があります。

DML_LOCKSパラメータは、DMLロックの最大数を指定します。この値は、すべてのユーザーが参照するすべての表でのロック数の合計と等しくなるように指定する必要があります。パラレルDML操作のロック要件は、シリアルDMLの要件と大きく異なります。

パラレルDMLで保持されるブロックはかなり多いため、DML_LOCKSパラメータの値を倍量に増やす必要があります。

ノート:

ターゲット表の表ロックが使用禁止の場合、パラレル DML 操作は実行されません。

表8-4に、様々なパラレルDML文ごとにコーディネータとパラレル実行サーバー・プロセスによって取得されるロックの種類を示します。この情報を使用して、このようなパラメータで必要な値を決定することができます。

表8-4 パラレルDML文で取得されるロック

文のタイプ	コーディネータ・プロセスが取得するロック	各パラレル実行サーバーが取得するロック
パーティション表に対するパラレル UPDATE または DELETE(WHERE 句によりパーティションまたはサブパーティションのサブセットにプルーニング)	1 つの表ロック SX プルーニングされたパーティションまたはサブパーティション当たり 1 つのパーティション・ロック X	1 つの表ロック SX 問合せサーバー・プロセスが所有するプルーニングされたパーティションまたはサブパーティション当たり 1 つのパーティション・ロック NULL 問合せサーバー・プロセスが所有するプルーニングされたパーティションまたはサブパーティション当たり 1 つのパーティション待機ロック S
パーティション表に対するパラレル行移行 UPDATE(WHERE 句によりパーティションまたはサブパーティションのサブセットにプルーニング)	1 つの表ロック SX プルーニングされたパーティションまたはサブパーティション当たり 1 つのパーティション・ロック X 他のすべてのパーティションまたはサブパーティションの 1 つのパーティション・ロック SX	1 つの表ロック SX 問合せサーバー・プロセスが所有するプルーニングされたパーティションまたはサブパーティション当たり 1 つのパーティション・ロック NULL 問合せサーバー・プロセスが所有するプルーニングされたパーティション当たり 1 つのパーティション待機ロック S 他のすべてのパーティションまたはサブパーティションの 1 つのパーティション・ロック SX
パーティション表に対するパラレル UPDATE、MERGE、DELETE または INSERT	1 つの表ロック SX すべてのパーティションまたはサブパーティションのパーティション・ロック X	1 つの表ロック SX パーティションまたはサブパーティション当たり 1 つのパーティション・ロック NULL パーティションまたはサブパーティション当たり 1 つのパーティション待機ロック S

文のタイプ	コーディネータ・プロセスが取得するロック	各パラレル実行サーバーが取得するロック
パーティション表に対するパラレルINSERT(目的の表はパーティション句またはサブパーティション句を含む)	1 つの表ロック SX 指定されたパーティションまたはサブパーティション当たり 1 つのパーティション・ロック X	1 つの表ロック SX 指定されたパーティションまたはサブパーティション当たり 1 つのパーティション・ロック NULL 指定されたパーティションまたはサブパーティション当たり 1 つのパーティション待機ロック S
非パーティション表に対するパラレルINSERT	1 つの表ロック X	なし

ノート:

表、パーティションおよびパーティション待機の DML ロックはすべて、TM ロックとして V$LOCK ビューに表示されます。

DOPを100として実行する600のパーティションを含む表について検討します。すべてのパーティションが、行移行のないパラレルUPDATE文またはDELETE文に関連するとします。

コーディネータでは次のロックが取得されます。

- 1つの表ロックSX
- 600のパーティション・ロックX

サーバー・プロセス全体では次のロックが取得されます。

- 100の表ロックSX
- 600のパーティション・ロックNULL
- 600のパーティション待機ロックS

8.6.3.3 I/Oに関連するパラメータ

このトピックでは、I/Oに影響するパラメータについて説明します。

I/Oに影響するパラメータを次に示します。

- DB_CACHE_SIZE
- DB_BLOCK_SIZE
- DB_FILE_MULTIBLOCK_READ_COUNT
- DISK_ASYNCH_IOおよびTAPE_ASYNCH_IO

これらのパラメータは、パラレル実行I/O操作の最適なパフォーマンスを確保するオプティマイザにも影響します。

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

8.6.3.3.1 DB_CACHE_SIZE

DB_CACHE_SIZEパラメータは、バッファのDEFAULTバッファ・プールのサイズをプライマリ・ブロック・サイズで設定します。

パラレルの更新、マージおよび削除操作を実行するとき、バッファ・キャッシュの動作は、大容量の更新を実行するOLTPシステムと非常によく似ています。

8.6.3.3.2 DB_BLOCK_SIZE

DB_BLOCK_SIZEパラメータは、Oracleデータベース・ブロックを設定します。

このパラメータの推奨値は8KBまたは16KBです。

データベース・ブロック・サイズはデータベースを作成するときに設定します。新しいデータベースを作成する場合は、8KBまたは16KBの大きなブロック・サイズを使用します。

8.6.3.3.3 DB_FILE_MULTIBLOCK_READ_COUNT

DB_FILE_MULTIBLOCK_READ_COUNTパラメータでは、オペレーティング・システムのREADコール1回で読み取られるデータベース・ブロック数を決定します。

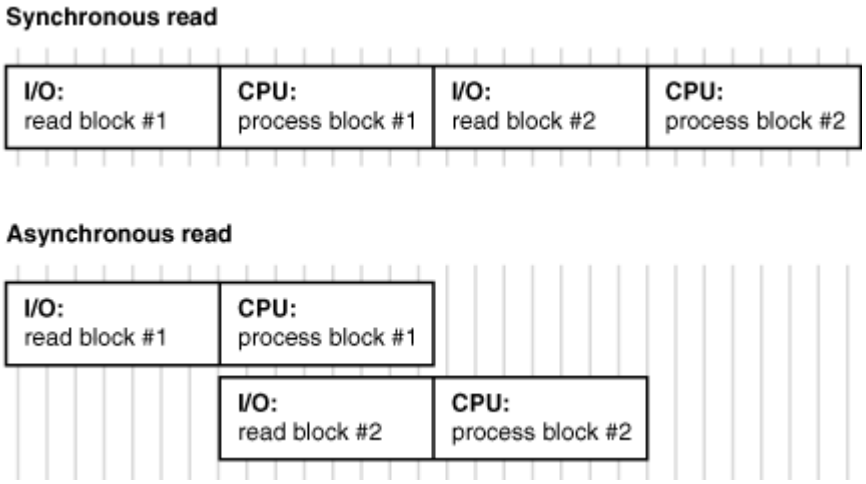
このパラメータのデフォルト値は効率よく実行できる最大I/Oサイズに対応する値です。最大I/Oサイズの値はプラットフォームによって異なり、ほとんどのプラットフォームで1MBです。DB_FILE_MULTIBLOCK_READ_COUNTに設定した値が高すぎる場合、データベースの起動時に、オペレーティング・システムによって許容範囲内の最高レベルまで値が下げられます。

8.6.3.3.4 DISK_ASYNCH_IOおよびTAPE_ASYNCH_IO

DISK_ASYNCH_IOおよびTAPE_ASYNCH_IOパラメータにより、オペレーティング・システムの非同期I/O機能を有効化または無効化できます。

DISK_ASYNCH_IOとTAPE_ASYNCH_IOパラメータの両方の推奨値はTRUEです。これらのパラメータによって、問合せサーバー・プロセスが、表スキャンを実行するときにI/Oリクエストと処理を同時に行うことができます。オペレーティング・システムで非同期I/Oがサポートされる場合は、これらのパラメータをデフォルト値のTRUEにしておきます。図8-6に、非同期読取りの仕組みを示します。

図8-6 非同期読取り



非同期操作が現在サポートされているのは、パラレル表スキャン、ハッシュ結合、ソートおよびシリアル表スキャンです。ただし、こ

の機能にはオペレーティング・システム固有の構成が必要となることがあり、すべてのプラットフォームでサポートされるとはかぎりません。

8.7 パラレル実行のパフォーマンスの監視

パラレル実行のパフォーマンスに関する問題を診断するときは、次の種類の監視を行う必要があります。

これらの監視の種類を、次に示します。

- 動的パフォーマンス・ビューを使用したパラレル実行パフォーマンスの監視
- セッション統計の監視
- システム統計の監視
- オペレーティング・システム統計の監視

関連項目:

動的ビューの詳細は、『Oracle Databaseリファレンス』を参照してください

8.7.1 動的パフォーマンス・ビューを使用したパラレル実行パフォーマンスの監視

動的パフォーマンス・ビューを使用してパラレル実行パフォーマンスを監視できます。

Oracle Databaseのリアルタイム監視機能を使用すると、SQL文の実行中にパフォーマンスを監視することができます。SQLの監視が自動的に開始するのは、SQL文がパラレルで実行されたとき、または1回の実行でCPUまたはI/O時間を5秒以上消費したときです。

システムが数日間稼働した後に、パラレル実行パフォーマンスの統計を監視して、並列処理が最適かどうかを判別する必要があります。これには、ここで説明するいずれかのビューを使用します。

Oracle Real Application Clustersでは、ここで説明するビューのグローバル・バージョンによって、複数インスタンスの統計が集計されます。グローバル・ビューの名前はGで開始します。たとえば、V$FILESTATに対してはGV$FILESTATとなります。

関連項目:

パフォーマンス監視の詳細は、『Oracle Database SQLチューニング・ガイド』を参照してください

8.7.1.1 V$PX_BUFFER_ADVICE

V$PX_BUFFER_ADVICE動的パフォーマンス・ビューを使用してパラレル実行のパフォーマンスを監視できます。

V$PX_BUFFER_ADVICEビューは、すべてのパラレル問合せによる最大バッファ使用量の履歴と見積りに関する統計を示します。このビューを調べて、パラレル問合せのメモリー不足の問題に応じてSGAサイズを再構成できます。

8.7.1.2 V$PX_SESSION

V$PX_SESSION動的パフォーマンス・ビューを使用してパラレル実行のパフォーマンスを監視できます。

V$PX_SESSIONビューは、問合せサーバーのセッション、グループ、セットおよびサーバー数のデータを示します。パラレル実行のた

めに稼働しているプロセスのリアルタイム・データも示します。この表には、リクエストされた並列度(DOP)と操作に与えられた実際のDOPの情報も含まれます。

8.7.1.3 V$PX_SESSTAT

V$PX_SESSTAT動的パフォーマンス・ビューを使用してパラレル実行のパフォーマンスを監視できます。

V$PX_SESSTATビューは、V$PX_SESSION表とV$SESSTAT表のセッション情報を結合したものです。つまり、通常のセッションで得られるすべてのセッション統計は、パラレル実行を使用して実行されるすべてのセッションでも利用できます。

8.7.1.4 V$PX_PROCESS

V$PX_PROCESS動的パフォーマンス・ビューを使用してパラレル実行のパフォーマンスを監視できます。

V$PX_PROCESSビューには、パラレル・プロセスの情報が含まれます。ステータス、セッションID、プロセスID、その他の情報があります。

8.7.1.5 V$PX_PROCESS_SYSSTAT

V$PX_PROCESS_SYSSTAT動的パフォーマンス・ビューを使用してパラレル実行のパフォーマンスを監視できます。

V$PX_PROCESS_SYSSTATビューは、問合せサーバーのステータスとバッファ割当て統計を示します。

8.7.1.6 V$PQ_SESSTAT

V$PQ_SESSTAT動的パフォーマンス・ビューを使用してパラレル実行のパフォーマンスを監視できます。

V$PQ_SESSTATビューは、システムの現在のサーバー・グループすべてのステータスを示します。問合せによってどのようにプロセスが割り当てられたか、またマルチユーザー問合せ調整アルゴリズムやロード・バランシング・アルゴリズムにより、デフォルト値やヒント指定された値がどのように影響されたかといったデータが含まれます。

場合によっては、これらのビューのデータを確認した後で、パフォーマンスを改善するために一部のパラメータの設定を調整する必要があります。その場合は、「パラレル実行のための一般的なパラメータのチューニング」の説明を参照してください。実行時間の長いパラレル操作の進捗を監視するには、これらのビューを定期的に問い合せます。

多くの動的パフォーマンス・ビューでは、Oracle Databaseが各ビューの統計を収集するためには、パラメータTIMED_STATISTICSをTRUEに設定する必要があります。ALTER SYSTEMまたはALTER SESSION文を使用して、TIMED_STATISTICSの設定を切り替えることができます。

8.7.1.7 V$PQ_TQSTAT

V$PQ_TQSTAT動的パフォーマンス・ビューを使用してパラレル実行のパフォーマンスを監視できます。

単純な例として、固有値が2種類のみの1つの列の結合による、2つの表のハッシュ結合について考えます。このハッシュ関数は、最大でも、パラレル実行サーバーAに対して1つのハッシュ値、パラレル実行サーバーBに対してもう1つのハッシュ値を生成します。DOPは2で問題ありません。4とすると少なくとも2つのパラレル実行サーバーの作業がなくなります。このような偏りを検出するには、次の例のような問合せを使用します。

```
SELECT dfo_number, tq_id, server_type, process, num_rows
FROM V$PQ_TQSTAT ORDER BY dfo_number DESC, tq_id, server_type, process;
```

この問題を解決する最適な方法は、別の結合方法の選択です。ネステッド・ループ結合が最適であると予想されます。または、結合表の一方が他方よりも小さい場合は、PQ_DISTRIBUTEヒントを使用してBROADCAST分散方法を指定できます。オプティマイザによってBROADCAST分散方法が考慮されるためには、OPTIMIZER_FEATURES_ENABLEを9.0.2以上に設定する必要があり

ます。

ここで、カーディナリティの高い結合キーがあるが、値の1つにほとんどのデータが含まれると仮定します。このような例としては、ラーバ・ランプの年次売上があります。大きな売上があったのは1968年のみで、1968年のレコードに対応するパラレル実行サーバーの負荷が高くなります。前述したものと同じ修正処理を使用する必要があります。

V$PQ_TQSTATビューは、表キュー・レベルのメッセージ・トラフィックの詳細なレポートを示します。V$PQ_TQSTATデータが有効なのは、パラレルSQL文を実行しているセッションから問い合せた場合のみです。 表キューは、問合せサーバー・グループ間、パラレル実行コーディネータと問合せサーバー・グループ間、または問合せサーバー・グループとコーディネータ間のパイプラインです。表キューは、PX SEND 〈partitioning type〉 (たとえば、PX SEND HASH)およびPX RECEIVEによって操作列に明示的に示されます。

V$PQ_TQSTATには、各表キューに対して読取りまたは書込みを行う問合せサーバー・プロセスごとに1行があります。10のコンシューマ・プロセスと10のプロデューサ・プロセスを接続する表キューの場合、このビューに20の行があります。バイト列を合計し、TQ_ID (表キュー識別子)によってグループ化すると、各表キューを介して送信された合計バイト数を求めることができます。この値をオプティマイザの見積りと比較します。差が大きいときは、より大きなサンプルを使用したデータの分析が必要な可能性があります。

TQ_IDでグループ化したバイト数の平方偏差を計算します。平方偏差が大きい場合はワークロードの不均衡を意味します。大きな平方偏差について調べて、プロデューサの起動時にデータ分散が不均等だったのか、分散そのものに偏りがあるのかを判別する必要があります。データそのものに偏りがある場合は、カーディナリティが低い、あるいは固有値が少ないことを意味します。

8.7.1.8 V$RSRC_CONS_GROUP_HISTORY

V$RSRC_CONS_GROUP_HISTORY動的パフォーマンス・ビューを使用してパラレル実行のパフォーマンスを監視できます。

V$RSRC_CONS_GROUP_HISTORYビューは、非NULLプランのあるV$RSRC_PLAN_HISTORYに、パラレル文のキューイングの情報など、各エントリに対するコンシューマ・グループ統計の履歴を表示します。

8.7.1.9 V$RSRC_CONSUMER_GROUP

V$RSRC_CONSUMER_GROUP動的パフォーマンス・ビューを使用してパラレル実行のパフォーマンスを監視できます。

V$RSRC_CONSUMER_GROUPビューは、パラレル文の情報など、現在アクティブなリソース・コンシューマ・グループに関連するデータを表示します。

8.7.1.10 V$RSRC_PLAN

V$RSRC_PLAN動的パフォーマンス・ビューを使用してパラレル実行のパフォーマンスを監視できます。

V$RSRC_PLANビューは、パラレル文のキューイングの状態など、現在アクティブなリソース・プランすべての名前を表示します。

8.7.1.11 V$RSRC_PLAN_HISTORY

V$RSRC_PLAN_HISTORY動的パフォーマンス・ビューを使用してパラレル実行のパフォーマンスを監視できます。

V$RSRC_PLAN_HISTORYは、リソース・プランがインスタンスで有効化、無効化または変更された場合の履歴を表示します。履歴にはパラレル文のキューイングの状態を含みます。

8.7.1.12 V$RSRC_SESSION_INFO

V$RSRC_SESSION_INFO動的パフォーマンス・ビューを使用してパラレル実行のパフォーマンスを監視できます。

V$RSRC_SESSION_INFOビューは、パラレル文のキュー統計など、リソース・マネージャ統計をセッション単位で表示します。列には、

PQ_SERVERSおよびPQ_STATUSが含まれます。

セッションがアクティブでパラレル問合せを実行している場合、V$RSRC_SESSION_INFOビューのPQ_SERVERS列には、アクティブなパラレル・サーバーの数が含まれます。問合せがキューに入れられると、この問合せで実行するパラレル・サーバーの数が表示されます。

PQ_STATUS列は、パラレル文がキューに入れられる理由を保持します。

関連項目:

V$RSRC_SESSION_INFOビューの詳細は、『Oracle Databaseリファレンス』を参照してください

8.7.1.13 V$RSRCMGRMETRIC

V$RSRCMGRMETRIC動的パフォーマンス・ビューを使用してパラレル実行のパフォーマンスを監視できます。

V$RSRCMGRMETRICビューは、パラレル文のキューイングに関連する統計を表示します。

パラレル文のキューイングに関連する統計は、指定された1分間の統計を取得して約1時間保持するリソース・マネージャ・メトリックに追加されます。

列には、AVG_ACTIVE_PARALLEL_STMTS、AVG_QUEUED_PARALLEL_STMTS、AVG_ACTIVE_PARALLEL_SERVERS、AVG_QUEUED_PARALLEL_SERVERSおよびPARALLEL_SERVERS_LIMITが含まれます。

関連項目:

V$RSRCMGRMETRICビューの詳細は、『Oracle Databaseリファレンス』を参照してください

8.7.2 セッション統計の監視

動的パフォーマンス・ビューを使用してセッション統計を監視し、パラレル実行のパフォーマンスを診断できます。

GV$PX_SESSIONを使用して、パラレルで実行するサーバー・グループの構成を判別します。この例では、セッション9が問合せコーディネータ、セッション7および21が最初のセットの最初のグループにあります。セッション18および20は2番目のセットの最初のグループです。この問合せでリクエストされたOPと与えられたDOPはどちらも2です。これは、次の問合せによる出力に示されます。

```
SELECT QCSID, SID, INST_ID "Inst", SERVER_GROUP "Group", SERVER_SET "Set",
  DEGREE "Degree", REQ_DEGREE "Req Degree"
FROM GV$PX_SESSION ORDER BY QCSID, QCINST_ID, SERVER_GROUP, SERVER_SET;
```

出力は次のようになります。

```
QCSID      SID        Inst       Group      Set        Degree     Req Degree
---------- ---------- ---------- ---------- ---------- ---------- ----------
         9          9          1
         9          7          1          1          1          2          2
         9         21          1          1          1          2          2
         9         18          1          1          2          2          2
         9         20          1          1          2          2          2
```

シングル・インスタンスの場合、SELECT FROM V$PX_SESSIONを使用します。列名Instance IDは含めません。

GV$PX_SESSIONを使用した前の例の出力に表示されるプロセスが、同じタスクを完了するために連携します。次の例は、物理読取りに関してこれらのプロセスの進捗を判別するための、結合問合せの実行を示します。次の問合せを使用して特定の統計を追跡できます。

```
SELECT QCSID, SID, INST_ID "Inst", SERVER_GROUP "Group", SERVER_SET "Set",
  NAME "Stat Name", VALUE
FROM GV$PX_SESSTAT A, V$STATNAME B
WHERE A.STATISTIC# = B.STATISTIC# AND NAME LIKE 'PHYSICAL READS'
  AND VALUE > 0 ORDER BY QCSID, QCINST_ID, SERVER_GROUP, SERVER_SET;
```

出力は次のようになります。

```
QCSID  SID   Inst   Group  Set    Stat Name          VALUE
------ ----- ------ ------ ------ ------------------ ----------
     9     9      1                physical reads           3863
     9     7      1      1       1 physical reads              2
     9    21      1      1       1 physical reads              2
     9    18      1      1       2 physical reads              2
     9    20      1      1       2 physical reads              2
```

このタイプの問合せを使用して、V$STATNAMEの統計を追跡します。問合せサーバー・プロセスの進捗を確認するには、この問合せを必要な回数繰り返します。

次の問合せは、V$PX_PROCESSを使用して問合せサーバーのステータスを調べます。

```
SELECT * FROM V$PX_PROCESS;
```

出力は次のようになります。

```
SERV STATUS    PID    SPID       SID    SERIAL# IS_GV   CON_ID
---- --------- ------ --------- ------ ------- -----  -------
P002 IN USE        16     16955     21    7729  FALSE        0
P003 IN USE        17     16957     20    2921  FALSE        0
P004 AVAILABLE     18     16959                 FALSE        0
P005 AVAILABLE     19     16962                 FALSE        0
P000 IN USE        12      6999     18    4720  FALSE        0
P001 IN USE        13      7004      7     234  FALSE        0
```

関連項目:

例で使用される動的パフォーマンス・ビューの詳細は、「動的パフォーマンス・ビューを使用したパラレル実行パフォーマンスの監視」を参照してください

8.7.3 システム統計の監視

動的パフォーマンス・ビューを使用してシステム統計を監視し、パラレル実行のパフォーマンスを診断できます。

V$SYSSTATビューおよびV$SESSTATビューには、パラレル実行を監視するためのいくつかの統計が含まれます。これらの統計を使用して、パラレル問合せ、DML、DDL、データ・フロー演算子(DFO)および操作の数を追跡します。問合せ、DMLまたはDDLそれぞれが、複数のパラレル操作および複数のDFOを含むことがあります。

さらに、統計では、マルチユーザー問合せ調整アルゴリズムまたは使用可能なパラレル実行サーバーの不足のために、DOPが減らされた(ダウングレードされた)問合せ操作の数もカウントされます。

また、これらのビューの統計では、パラレル実行のために送信されたメッセージ数もカウントされます。次の構文は、これらの統計を表示する方法の例です。

```
SELECT NAME, VALUE FROM GV$SYSSTAT
  WHERE UPPER (NAME) LIKE '%PARALLEL OPERATIONS%'
  OR UPPER (NAME) LIKE '%PARALLELIZED%' OR UPPER (NAME) LIKE '%PX%';
```

出力は次のようになります。

```
NAME                                             VALUE
------------------------------------------------ ----------
queries parallelized                                    347
DML statements parallelized                               0
DDL statements parallelized                               0
DFO trees parallelized                                  463
Parallel operations not downgraded                       28
Parallel operations downgraded to serial                 31
Parallel operations downgraded 75 to 99 pct             252
Parallel operations downgraded 50 to 75 pct             128
Parallel operations downgraded 25 to 50 pct              43
Parallel operations downgraded 1 to 25 pct               12
PX local messages sent                                74548
PX local messages recv'd                              74128
PX remote messages sent                                   0
PX remote messages recv'd                                 0
```

次の問合せは、システムの各ワーカー(子プロセス)および問合せコーディネータ・プロセスの現在の待機状態を示します。

```
SELECT px.SID "SID", p.PID, p.SPID "SPID", px.INST_ID "Inst",
       px.SERVER_GROUP "Group", px.SERVER_SET "Set",
       px.DEGREE "Degree", px.REQ_DEGREE "Req Degree", w.event "Wait Event"
FROM GV$SESSION s, GV$PX_SESSION px, GV$PROCESS p, GV$SESSION_WAIT w
WHERE s.sid (+) = px.sid AND s.inst_id (+) = px.inst_id AND
      s.sid = w.sid (+) AND s.inst_id = w.inst_id (+) AND
      s.paddr = p.addr (+) AND s.inst_id = p.inst_id (+)
ORDER BY DECODE(px.QCINST_ID,  NULL, px.INST_ID,  px.QCINST_ID), px.QCSID,
DECODE(px.SERVER_GROUP, NULL, 0, px.SERVER_GROUP), px.SERVER_SET, px.INST_ID;
```

8.7.4 オペレーティング・システム統計の監視

Oracle Databaseで得られる情報とオペレーティング・システムのユーティリティ(UNIXベース・システムでのsarやvmstatなど)で得られる情報はかなり重複しています。

オペレーティング・システムでは、I/O、通信、CPU、メモリーとページング、スケジューリング、同期プリミティブに関するパフォーマンス統計が提供されます。V$SESSTATビューにも、オペレーティング・システム統計の主なカテゴリが示されます。

通常、I/Oデバイスやセマフォ操作に関するオペレーティング・システム情報を、データベースのオブジェクトや操作にマップするのは、Oracle Databaseの情報に比べて困難です。ただし、オペレーティング・システムによっては、優れたビジュアル化ツールが備えられており、データを収集する効率のよい手段があります。

CPUやメモリーの使用量に関するオペレーティング・システム情報は、パフォーマンスを調査するために大変重要です。おそらく最も重要な統計はCPU使用率です。下位レベルのパフォーマンス・チューニングの目標は、すべてのCPUでCPUバウンドになることです。これが達成されると、SQLレベルで作業して、CPUの使用率は低いがI/Oの使用率が高い代替計画を見つけることができます。

オペレーティング・システムのメモリーとページングの情報は、メモリー集中型データ・ウェアハウス・サブシステム(パラレルの通信、ソート、ハッシュ結合など)の間でメモリーをどのように分割するかを制御する、多数のシステム・パラメータのチューニングに役立ち

ます。

8.8 パラレル実行のチューニングのヒント

この項では、パラレル実行環境でのパフォーマンスを向上させるための様々な方法を説明します。

この項では、次の項目について説明します。

- パラレル実行計画の実装
- パラレルでの表の作成および移入によるパフォーマンスの最適化
- EXPLAIN PLANを使用したパラレル操作計画の表示
- パラレルDMLのその他の考慮事項
- パラレルでの索引の作成によるパフォーマンスの最適化
- パラレルDMLのヒント
- パラレルでの増分データ・ロード

8.8.1 パラレル実行計画の実装

優れたパラレル実行計画の実装は、高いパフォーマンスを確実にするために重要となります。

優れた計画のための推奨事項を次に示します。

- システム内で起こっていることを理解するために、単純な設定を実装します。
- リソース・マネージャを使用して、システムに負担をかけることなく一定量の処理リソースを各グループに割り当てるよう、コンシューマ・グループの最大並列度(DOP)を指定します。リソース・マネージャ・ポリシーは、パラレル実行を使用してシステムを統御する場合、およびSQL文を必ずパラレルで実行できるようにするために必要となります。
- パラレル実行に使用できるようにするシステム・リソースの量を計画の基礎とします。パラメータPARALLEL_MAX_SERVERSおよびPARALLEL_SERVERS_TARGETの値を調整して、システム内で実行されるパラレル実行(PX)サーバーの数を制限します。
- ETL (抽出、変換およびロード)計画ではなくELT (抽出、ロードおよび変換)計画の採用を検討します。
- より高速なデータ・ロードのために、CTASやIASなど、パラレルSQL文で外部表を使用します。

8.8.2 パラレルでの表の作成および移入によるパフォーマンスの最適化

大量の結果セットを取得するパラレル実行パフォーマンスを最適化するには、パラレルで表を作成および移入します。

Oracle Databaseは、結果をユーザー・プロセスにパラレルで返すことはできません。問合せが多数の行を返す場合、問合せの実行は実際に速くなることもあります。ただし、ユーザー・プロセスは行の受取りをシリアルでしか実行できません。大量の結果セットを取得する問合せのパラレル実行パフォーマンスを最適化するには、PARALLEL CREATE TABLE AS SELECTまたはダイレクト・パスINSERTを使用して結果セットをデータベースに格納します。ユーザーは結果セットを後からシリアルで確認できます。

SELECTをパラレルで実行してもCREATE文には影響しません。ただし、CREATE文がパラレルで実行される場合、オプティマイザはSELECTもパラレルで実行しようとします。

パラレルのCREATE TABLE AS SELECTをNOLOGGINGオプションと組み合せると、次の例のように大変効率のよい中間表機能が提供されます。

```
CREATE TABLE summary PARALLEL NOLOGGING AS SELECT dim_1, dim_2 ...,
SUM (meas_1)
```

```
FROM facts GROUP BY dim_1, dim_2;
```

これらの表も、パラレルINSERTを使用して増分的にロードできます。中間表を活用するには、次の方法を使用します。

- 通常の副問合せは、1回計算すると、何度でも参照できます。これを利用すると、スター・スキーマに対する一部の問合せ(特に、選択のためのWHERE句を含む述語のない問合せ)をより適切にパラレル化できるようになります。スター型変換を使用する、選択のためのWHERE句を含む述語のあるスター問合せは、SQLを変更しなくても自動的に効率よくパラレル化できます。
- アプリケーションレベルのチェックポイントまたは再起動を実現するために、複雑な問合せを単純なステップに分解します。たとえば、1TBのデータベースに対して複雑な複数表の結合を実行すると、数十時間に及ぶ可能性があります。この問合せの最中に障害が発生すると、最初からやりなおすことになります。CREATE TABLE AS SELECTまたはPARALLEL INSERT AS SELECTを使用してこの問合せを再作成し、数時間ずつ実行される単純な問合せの順序を作成します。システム障害が発生しても、最後に完了したステップの次から問合せを再起動できます。
- 元の表の不要な行を省いた新しい表を作成してから、元の表を削除することにより、手動パラレル削除操作を効率よく実装します。または、元の表から行を直接削除する、便利なパラレル削除機能を使用することもできます。
- 効率のよいマルチディメンション・ドリルダウン分析のためにサマリー表を作成します。たとえば、サマリー表には、月、種類、地域、販売担当者ごとにグループ化された収益の合計を含めることができます。
- 行チェーンの消去や空き領域の圧縮などを行い、古い表を新しい表にコピーして、表を再編成します。この方法はエクスポートとインポートよりも速く、再ロードよりも容易です。

新たに作成した表で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てください。I/Oボトルネックを回避するために、少なくともCPUと同数の物理ディスクに対してストライプ化された表領域を指定します。領域を割り当てるときに断片化を回避するために、表領域のファイル数はCPU数の倍数にする必要があります。

関連項目:

データ・ウェアハウスのパラレル実行の詳細は、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください

8.8.3 EXPLAIN PLANを使用したパラレル操作計画の表示

EXPLAIN PLAN文を使用して、パラレル問合せの実行計画を表示します。

EXPLAIN PLANの出力で、COST、BYTESおよびCARDINALITY列にオプティマイザの情報が表示されます。utlxplp.sqlスクリプトを使用して、関連するすべてのパラレル情報と一緒にEXPLAIN PLAN出力を表示することもできます。

結合文のパラレル実行を最適化する方法がいくつかあります。システム構成を変更するか、この章で前に説明したようにパラメータを調整するか、DISTRIBUTIONヒントなどのヒントを使用します。

EXPLAIN PLANを使用する際の重要なポイントは次のとおりです。

- オプティマイザの選択性見積りを確認します。問合せで1行しか生成されないとオプティマイザが予測する場合、ネステッド・ループを使用する傾向が高くなります。このとき、表が分析されていない可能性や、同一の表に対する複数の述語の相関関係についてオプティマイザの見積りが誤っている可能性があります。オプティマイザに正しい選択性を提供したり、オプティマイザに別の結合方法を使用させるために、統計やヒントを拡張することが必要な場合があります。
- 低カーディナリティの結合キーに対するハッシュ結合を使用します。結合キーに固有の値が少ない場合、ハッシュ結合は最適ではない可能性があります。固有値の数が並列度(DOP)を下回る場合は、一部のパラレル問合せサーバーが

特定の問合せの処理を行えないことがあります。

- データの偏りを考慮します。結合キーに過度のデータの偏りがある場合、ハッシュ結合を行うと、一部のパラレル問合せサーバーに多くの作業が偏る可能性があります。オプティマイザによってBROADCAST分散方法が選択されなかった場合は、そのためのヒントの使用を検討してください。オプティマイザによってBROADCAST分散方法が考慮されるのは、OPTIMIZER_FEATURES_ENABLEを9.0.2以上に設定した場合のみです。詳細は、「V$PQ_TQSTAT」を参照してください。

8.8.3.1 例: EXPLAIN PLANを使用したパラレル操作の表示

EXPLAIN PLANを使用してパラレル操作を表示できます。

次の例は、オプティマイザがパラレル問合せを実行しようとする場合を示しています。

```
explain plan for
  SELECT /*+ PARALLEL */ cust_first_name, cust_last_name
   FROM customers c, sales s WHERE c.cust_id = s.cust_id;


-----------------------------------------------------------
| Id  | Operation                         |  Name          |
-----------------------------------------------------------
|   0 | SELECT STATEMENT                  |                |
|   1 |  PX COORDINATOR                   |                |
|   2 |   PX SEND QC (RANDOM)             | :TQ10000       |
|   3 |    NESTED LOOPS                   |                |
|   4 |     PX BLOCK ITERATOR             |                |
|   5 |      TABLE ACCESS FULL            | CUSTOMERS      |
|   6 |     PARTITION RANGE ALL           |                |
|   7 |      BITMAP CONVERSION TO ROWIDS  |                |
|   8 |       BITMAP INDEX SINGLE VALUE   | SALES_CUST_BIX |
-----------------------------------------------------------

Note
-----
   - automatic DOP: Computed Degree of Parallelism is 2
```

8.8.4 パラレルDMLのその他の考慮事項

このトピックでは、パラレルDML操作の使用に関するその他の考慮事項について説明します。

データ・ウェアハウスに対してパラレルの挿入、更新または削除操作を使用して、データ・ウェアハウス・データベースをリフレッシュする予定があるときは、物理データベースを設計する際にさらに考慮すべき項目があります。これらの考慮事項はパラレル実行操作には影響しません。これらの問題は次のとおりです。

- パラレルDMLおよびダイレクト・パス制限
- 並列度の制限
- INITRANSを増加するタイミング
- セグメントで使用可能なトランザクション空きリスト数の制限
- 多数のREDOログの複数のアーカイバ
- データベース・ライター・プロセス(DBWn)のワークロード
- [NO]LOGGING句

8.8.4.1 パラレルDMLおよびダイレクト・パス制限

このトピックでは、パラレルおよびダイレクト・パス操作の制限について説明します。

パラレル制限に違反すると、操作はシリアルで実行されるだけです。ダイレクト・パスINSERT制限に違反すると、APPENDヒントが無視され、従来の挿入操作が実行されます。エラー・メッセージは返されません。

8.8.4.2 並列度の制限

使用しているOracle Databaseのソフトウェア・レベルに基づく並列度の制限があります。

パラレルDML itl不変プロパティのない表(Oracle9iリリース2 (9.2)よりも前に作成された表、またはCOMPATIBLE初期化パラメータを9.2より小さい値に設定して作成された表)については、並列度(DOP)はパーティションまたはサブパーティションの数と等しくなります。つまり、表がパーティション化されていない場合、問合せはシリアルで実行されます。このプロパティのない表を特定するには、次の文を発行します。

```
SELECT u.name, o.name FROM obj$ o, tab$ t, user$ u
 WHERE o.obj# = t.obj# AND o.owner# = u.user#
 AND bitand(t.property,536870912) != 536870912;
```

関連項目:

トランザクション表とも呼ばれるInterested Transaction List (ITL)の詳細は、『Oracle Database概要』を参照してください

8.8.4.3 INITRANSを増加するタイミング

特定の状況では、INITRANSの値を増やす必要があります。

グローバル索引がある場合、グローバル索引セグメントとグローバル索引ブロックは同じパラレルDML文のサーバー・プロセスによって共有されます。操作が同じ行に対して実行されなくても、サーバー・プロセスは同じ索引ブロックを共有できます。各サーバー・トランザクションは、索引ブロックに変更を行う前に、索引ブロック・ヘッダーの1つのトランザクション・エントリを必要とします。

この状況では、CREATE INDEX文またはALTER INDEX文を使用する際、INITRANS(各データ・ブロックに割り当てられるトランザクションの初期値)を、その索引に対する最大DOPなどの大きな値に設定する必要があります。

8.8.4.4 セグメントで使用可能なトランザクション空きリスト数の制限

ディクショナリ管理表領域のセグメントで使用可能なトランザクション空きリスト数には制限があります。

セグメントが作成されると、プロセス空きリストとトランザクション空きリストの数は固定され、変更できません。セグメント・ヘッダーでプロセス空きリストの数として大きな値を指定すると、それによって使用可能なトランザクション空きリストの数が制限されることがあります。この制限を避けるには、次にセグメント・ヘッダーを再作成するときに、プロセス空きリストの数を減らします。こうすると、セグメント・ヘッダーでトランザクション空きリストのための余裕ができます。

UPDATEおよびDELETE操作では、各サーバー・プロセスが独自のトランザクション空きリストを必要とする場合があります。このため、パラレルDMLのDOPは、表とDML文でメンテナンスする必要があるすべてのグローバル索引に対して使用可能な、トランザクション空きリストの最小数によって事実上制限されます。たとえば、表に25のトランザクション空きリストがあるとき、その表に2つのグローバル索引があり、1つの索引に50のトランザクション空きリスト、もう1つの索引に30のトランザクション空きリストがあるとすると、DOPの制限は25になります。この表のトランザクション空きリストが40であれば、DOPの制限は30になります。

STORAGE句のFREELISTSパラメータは、プロセス空きリストの数を設定するために使用されます。デフォルトではプロセス空きリストは作成されません。

トランザクション空きリストのデフォルト数はブロック・サイズによって異なります。たとえば、プロセス空きリストの数が明示的に設定されない場合、デフォルトでは4KBのブロックに対して約80のトランザクション空きリストになります。トランザクション空きリストの最小数は25です。

8.8.4.5 多数のREDOログの複数のアーカイバ

多数のREDOログをアーカイブするには、複数のアーカイバ・プロセスが必要です。

パラレルDDLおよびパラレルDML操作は、多数のREDOログを生成する場合があります。これらのREDOログのアーカイブは、1つのARCHプロセスでは間に合わない可能性があります。この問題を回避するため、複数のアーカイバ・プロセスを生成できます。これは、手動で行うこともジョブ・キューを使用して行うこともできます。

8.8.4.6 データベース・ライター・プロセス(DBWn)のワークロード

データベース・ライター・プロセスの数を増やす必要がある場合があります。

パラレルDML操作により、短時間でバッファ・キャッシュ内の多数のデータ、索引およびUNDOブロックの内容が使用されます。たとえば、次の構文でV$SYSTEM_EVENTビューを問い合せると、free_buffer_waitsの値が高く示されると想定します。

```
SELECT TOTAL_WAITS FROM V$SYSTEM_EVENT WHERE EVENT = 'FREE BUFFER WAITS';
```

この場合は、DBW*n*プロセスを増やすことの検討が必要です。空きバッファの待機時間がない場合、問合せによって行が返されません。

8.8.4.7 [NO]LOGGING句

[NO]LOGGING句を設定する際の考慮事項を理解してください。

[NO]LOGGING句は、表、パーティション、表領域および索引に適用されます。NOLOGGING句を使用すると、事実上、特定の操作(ダイレクト・パスINSERTなど)に対してログが生成されません。NOLOGGING属性はINSERT文のレベルでは指定されませんが、表、パーティション、索引または表領域に対してALTER文またはCREATE文を使用して指定されます。

表または索引にNOLOGGINGが設定されると、パラレルまたはシリアルのダイレクト・パスINSERT操作ではREDOログが生成されません。NOLOGGINGオプションが設定されて実行しているプロセスは、REDOが生成されないため、処理が速くなります。ただし、表、パーティションまたは索引に対するNOLOGGING操作の後、バックアップを実行する前にメディア障害が発生すると、変更されたすべての表、パーティションおよび索引が破損することがあります。

NOLOGGING句が使用されている場合、ダイレクト・パスINSERT操作(ディクショナリ更新以外)では、REDOログは常に生成されません。NOLOGGING属性はUNDOには影響せず、REDOのみに影響します。正確には、NOLOGGINGを使用すると、ダイレクト・パスINSERT操作ではごくわずかのREDO(フル・イメージREDOとは逆のレンジ無効REDO)が生成されます。

下位互換性のため、CREATE TABLE文での代替キーワードとして[UN]RECOVERABLEも引き続きサポートされています。ただし、この代替キーワードは今後のリリースでサポートが終了する可能性があります。

表領域レベルでは、LOGGING句によって、その表領域に作成されるすべての表、索引およびパーティションのデフォルト・ロギング属性が指定されます。既存の表領域のロギング属性がALTER TABLESPACE文によって変更されると、ALTER文の後で作成されたすべての表、索引およびパーティションには新しいロギング属性が適用されます。既存の表、索引、パーティションのロギング属性は変化しません。表領域レベルのロギング属性は、表、索引またはパーティション・レベルの指定で上書きできます。

デフォルトのロギング属性はLOGGINGです。ただし、ALTER DATABASE NOARCHIVELOGを発行してデータベースをNOARCHIVELOGモードにした場合、ロギング属性の指定にかかわらず、ロギングなしで実行できるすべての操作ではログが生成されなくなります。

8.8.5 パラレルでの索引の作成によるパフォーマンスの最適化

パラレルで索引を作成してパフォーマンスを最適化できます。

複数のプロセスが同時に作動して、1つの索引を作成できます。索引を作成するために必要な処理を複数のサーバー・プロセスに分割することで、Oracle Databaseでは、1つのサーバー・プロセスによって索引をシリアルで作成した場合に比べて速く索引を作成できます。

パラレル索引作成では、ORDER BY句での表スキャンとほぼ同様の処理が行われます。表がランダムにサンプリングされ、索引をDOPと同数に均等に分割するための索引キーのセットが検出されます。問合せプロセスの第1セットが表をスキャンしてキーと行IDのペアを抽出し、問合せプロセスの第2セットのプロセスに対してキーに基づいて各ペアを送ります。第2セットの各プロセスは、キーをソートし、通常の方法で索引を構築します。索引のすべての部分が構築されると、パラレル実行コーディネータが各部(ソート済)を連結して、最終的に索引を形成します。

パラレルのローカル索引作成では、1つのサーバー・セットが使用されます。セットの各サーバー・プロセスは、割り当てられた表パーティションをスキャンし、そのパーティションに対して索引パーティションを構築します。所定のDOPに対して半数のサーバー・プロセスが使用されるため、パラレルのローカル索引作成が可能になるのはDOPが高い場合です。ただし、DOPは作成する索引パーティション数以内に制限されます。この制限を回避するには、DBMS_PCLXUTILパッケージを使用できます。

索引作成時にREDOおよびUNDOのロギングを行わないことをオプションで指定できます。こうすると、パフォーマンスが大きく向上する可能性がありますが、索引が一時的にリカバリ不可の状態になります。新しい索引のバックアップを取得すると、リカバリは可能になります。アプリケーションで、索引のリカバリ時に索引を再作成するNOLOGGING句の使用を検討してください。

CREATE INDEX文のPARALLEL句のみが、索引作成のDOPを指定するただ1つの方法です。DOPがCREATE INDEX文のPARALLEL句に指定されないと、CPUの数がDOPとして使用されます。PARALLEL句がない場合、索引作成はシリアルで行われます。

索引をパラレルで作成するとき、STORAGE句には、問合せサーバー・プロセスで作成される各副索引の記憶域を指定します。このため、INITIAL値が5MB、DOPが12で作成される索引は、索引作成時に少なくとも60MBの記憶域を消費します。各プロセスが5MBのエクステントで開始するためです。問合せコーディネータ・プロセスがソート済の副索引を結合するときに、エクステントの一部が切り捨てられ、生成される索引はリクエストの60MBよりも小さくなることがあります。

UNIQUEまたはPRIMARY KEY制約を表に追加するか、有効にすると、必要な索引をパラレルで自動的に作成できなくなります。かわりに、CREATE INDEX文と適切なPARALLEL句を使用して、必要な列に対して手動で索引を作成します。こうすることで、制約を有効にするか追加するときに、Oracle Databaseが既存の索引を使用します。

同じ表の複数の制約を同時にパラレルで有効にできるのは、すべての制約がすでにENABLE NOVALIDATE状態になっている場合です。次の例では、ALTER TABLE ENABLE CONSTRAINT文が、制約をパラレルでチェックする表スキャンを実行します。

```
CREATE TABLE a (a1 NUMBER CONSTRAINT ach CHECK (a1 > 0) ENABLE NOVALIDATE)
PARALLEL;
INSERT INTO a values (1);
COMMIT;
ALTER TABLE a ENABLE CONSTRAINT ach;
```

8.8.6 パラレルDMLのヒント

このトピックでは、パラレルDML機能のヒントについて説明します。

内容は次のとおりです。

- パラレルDMLのヒント1: INSERT
- パラレルDMLのヒント2: ダイレクト・パスINSERT

- パラレルDMLのヒント3: INSERT、MERGE、UPDATEおよびDELETEのパラレル化

関連項目:

- ダイレクト・パスINSERTによるロード・パフォーマンス向上の詳細は、『Oracle Database管理者ガイド』を参照してください
- INSERT文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

8.8.6.1 パラレルDMLのヒント1: INSERT

このトピックでは、SQL INSERT文を使用した場合のパラレルDMLについて説明します。

INSERT文を使用して実行できる機能は表8-5にまとめられます。

表8-5 INSERT機能のまとめ

挿入タイプ	パラレル	シリアル	NOLOGGING
従来型	不可 パラレル DML を有効にして NOAPPEND ヒントを使用して、従来型のパラレル挿入を実行する方法の詳細は、この項の説明を参照してください。	可	不可
ダイレクト・パス INSERT (APPEND)	可能(次の指定が必要) PARALLEL DML モードを有効にするための ALTER SESSION ENABLE PARALLEL DML または ENABLE_PARALLEL_DML SQL ヒント さらに、次のいずれか 1 つの指定 ● 並列処理を明示的に設定する表の PARALLEL 属性または PARALLEL ヒント ● モードを明示的に設定する APPEND ヒント または次の指定 強制的に PARALLEL DML モードにする ALTER SESSION FORCE PARALLEL DML	可能(次の指定が必要): APPEND ヒント	可能(次の指定が必要): パーティションまたは表に対する NOLOGGING 属性の設定

パラレルDMLが有効で、PARALLELヒントがあるか PARALLEL属性がデータ・ディクショナリの表に設定されている場合、制限が適用されないかぎり挿入操作はパラレルで行われ追加されます。PARALLELヒントまたはPARALLEL属性のいずれかがない場合、挿入操作はシリアルで行われます。自動DOPでは、パラレルDMLが有効化または強制される場合にかぎり、SQL文のDML部分のみがパラレル化されます。

パラレルDMLが有効化される場合は、NOAPPENDヒントを使用してパラレルの従来型挿入操作を実行できます。たとえば、/*+

noappend parallel */をSQL INSERT文で使用して、パラレルの従来型挿入を実行できます。

```
SQL> INSERT /*+ NOAPPEND PARALLEL */ INTO sales_hist SELECT * FROM sales;
```

パラレルの従来型挿入操作の利点は、ダイレクト・パスINSERTの制約なしにオンライン操作を実行できることです。パラレルの従来型挿入操作のデメリットは、ダイレクト・パスINSERTよりもプロセスの速度が遅いことです。

8.8.6.2 パラレルDMLのヒント2: ダイレクト・パスINSERT

このトピックでは、ダイレクト・パスINSERT操作を使用した場合のパラレルDMLについて説明します。

追加モードは、パラレルの挿入操作時のデフォルトです。データは、表に割り当てられる新規ブロックに常に挿入されます。APPENDヒントの使用はオプションです。追加モードは、INSERT操作の速度を速める場合には使用しますが、領域の使用率の最適化が必要な場合は使用しないでください。NOAPPENDを使用すると追加モードを上書きできます。

APPENDヒントは、シリアルとパラレルの両方の挿入操作に適用されます。このヒントを使用すると、シリアル挿入もより高速になります。ただし、APPENDは多くの領域を使用する必要があり、ロックのオーバーヘッドも増加します。

NOLOGGINGとAPPENDを一緒に使用すると、プロセスはさらに速くなります。NOLOGGINGは操作のREDOログが生成されないことを意味します。NOLOGGINGはデフォルトではありません。パフォーマンスを最適化するときに使用します。表またはパーティションのリカバリが必要となる通常の場合には使用しないでください。リカバリが必要な場合は、操作の直後にバックアップを取得するようにします。ALTER TABLE [NO]LOGGING文を使用して、適切な値を設定します。

8.8.6.3 パラレルDMLのヒント3: INSERT、MERGE、UPDATEおよびDELETEのパラレル化

このトピックでは、挿入、マージ、更新および削除操作を使用した場合のパラレルDMLについて説明します。

データ・ディクショナリで表またはパーティションにPARALLEL属性があるとき、この属性設定が、INSERT文、UPDATE文、DELETE文および問合せでの並列処理を決定するために使用されます。文での表に対する明示的なPARALLELヒントは、データ・ディクショナリのPARALLEL属性の効果よりも優先されます。

NO_PARALLELヒントを使用すると、データ・ディクショナリの表のPARALLEL属性を上書きできます。一般的にヒントは属性よりも優先されます。

DML操作の並列処理が考慮されるのは、ALTER SESSION ENABLE PARALLEL DML文を使用してセッションがPARALLEL DMLモードになっている場合、または、ENABLE_PARALLEL_DMLヒントを使用して特定のSQL文がPARALLEL DMLモードになっている場合です。このモードは、問合せまたはDML文の問合せ部分の並列処理には影響しません。

8.8.6.3.1 INSERT SELECTのパラレル化

INSERT ... SELECT文では、INSERTキーワードの後にPARALLELヒントを指定できます。このヒントはSELECTキーワードの後にも指定できます。

INSERTキーワードの後のPARALLELヒントはINSERT操作のみに適用され、SELECTキーワードの後のPARALLELヒントはSELECT操作のみに適用されます。したがって、INSERT操作とSELECT操作の並列処理は互いに独立しています。一方の操作をパラレルで実行できなくても、もう一方の操作をパラレルで実行できるかどうかには影響しません。

ユーザーがパラレルDMLをセッションで明示的に有効化した場合、および関連する表にデータ・ディクショナリ・エントリでPARALLEL属性が設定されている場合、挿入操作をパラレル化できる機能により既存の動作が変更されます。この場合、選択操作がパラレル化された既存のINSERT SELECT文で、挿入操作もパラレル化できます。

複数の表を問い合せる場合は、複数のSELECT PARALLELヒントと複数のPARALLEL属性を指定できます。

例8-5に、ACMEの買収後に雇用された新しい従業員の追加を示します。

例8-5INSERT SELECTのパラレル化

```
INSERT /*+ PARALLEL(employees) */ INTO employees
SELECT /*+ PARALLEL(ACME_EMP) */ *  FROM ACME_EMP;
```

この例ではAPPENDキーワードは必要ありません。PARALLELヒントによって追加が暗黙的に指定されるためです。

8.8.6.3.2 UPDATEとDELETEのパラレル化

PARALLELヒント(UPDATEまたはDELETEキーワードの直後に指定)は、基礎となるスキャン操作に適用されるだけではなく、UPDATEまたはDELETE操作にも適用されます。

または、変更対象の表の定義に指定されるPARALLEL句で、UPDATEまたはDELETEのパラレル化を指定できます。

セッションまたはトランザクションについてパラレルDMLを明示的に有効化した場合、問合せ操作がパラレル化されたUPDATE文またはDELETE文は、UPDATE操作またはDELETE操作もパラレル化できます。文の副問合せまたは更新可能ビューには、独自のPARALLELヒントまたは句を指定できますが、これらのパラレル・ディレクティブは更新または削除をパラレル化する決定には影響しません。これらの操作をパラレルで実行できなくても、UPDATEまたはDELETE部分をパラレルで実行できるかどうかには影響しません。

例8-6に、ダラスのすべての店員を10%昇給する更新操作を示します。

例8-6UPDATEとDELETEのパラレル化

```
UPDATE /*+ PARALLEL(employees) */ employees
 SET salary=salary * 1.1 WHERE job_id='CLERK' AND department_id IN
  (SELECT department_id FROM DEPARTMENTS WHERE location_id = 'DALLAS');
```

PARALLELヒントは、UPDATE操作とスキャンに適用されます。

例8-7には、カテゴリ39のすべての商品の削除を示します。最近、事業が別会社として分離されたためです。

例8-7UPDATEとDELETEのパラレル化

```
DELETE /*+ PARALLEL(PRODUCTS) */ FROM PRODUCTS
  WHERE category_id = 39;
```

ここでも、並列性処理、表employeesのスキャンとUPDATE操作に適用されます。

8.8.7 パラレルでの増分データ・ロード

パラレルDMLを更新可能結合ビュー機能と組み合せると、データ・ウェアハウス・システムの表をリフレッシュするための効率のよいソリューションが得られます。

表をリフレッシュするには、OLTP本番システムで生成される差分データを使用して更新します。

次の例では、列c_key、c_nameおよびc_addrを含むcustomersという表をリフレッシュするとします。差分データには、新しい行またはデータ・ウェアハウスの最後のリフレッシュ以降に更新された行が含まれます。この例では、本番システムの更新データはデータ・ウェアハウス・システムにASCIIファイルとして送られます。リフレッシュ・プロセスを開始する前に、これらのファイルを一時表diff_customerにロードする必要があります。このタスクを効率よく実行するには、パラレル・オプションとダイレクト・オプションの両方を指定したSQL*Loaderを使用できます。パラレルでロードする場合はAPPENDヒントも使用します。

diff_customerがロードされたら、リフレッシュ・プロセスを開始できます。これは、次に示すように、2フェーズで実行するか、パラレルでのマージにより実行することができます。

- パラレルでの表の更新のためのパフォーマンスの最適化

- パラレルでの表への新しい行の効率的な挿入
- パラレルでのマージによるパフォーマンスの最適化

8.8.7.1 パラレルでの表の更新のためのパフォーマンスの最適化

このトピックでは、パラレルでの表の更新のためのパフォーマンスの最適化方法について説明します。

次の文は、副問合せを使用して更新する単純なSQLの実装です。

```
UPDATE customers SET(c_name, c_addr) = (SELECT c_name, c_addr
  FROM diff_customer WHERE diff_customer.c_key = customer.c_key)
  WHERE c_key IN(SELECT c_key FROM diff_customer);
```

残念ながら、この文の2つの副問合せはパフォーマンスに影響します。

かわりに、更新可能結合ビューを使用してこの問合せを作成しなおすことができます。問合せを再作成するには、まず主キー制約をdiff_customer表に追加して、変更される列がキー保存表にマップされるようにします。

```
CREATE UNIQUE INDEX diff_pkey_ind ON diff_customer(c_key) PARALLEL NOLOGGING;

ALTER TABLE diff_customer ADD PRIMARY KEY (c_key);
```

その後、次のSQL文を使用してcustomers表を更新できます。

```
UPDATE /*+ PARALLEL(cust_joinview) */
  (SELECT /*+ PARALLEL(customers) PARALLEL(diff_customer) */
  CUSTOMER.c_name AS c_name CUSTOMER.c_addr AS c_addr,
   diff_customer.c_name AS c_newname, diff_customer.c_addr AS c_newaddr
   FROM diff_customer
   WHERE customers.c_key = diff_customer.c_key) cust_joinview
   SET c_name = c_newname, c_addr = c_newaddr;
```

結合ビューcust_joinviewにデータを提供するための基本スキャンはパラレルで実行されます。その後、更新をパラレル化して、さらにパフォーマンスを向上させることができます。ただし、これはcustomers表がパーティション化されている場合のみです。

8.8.7.2 パラレルでの表への新しい行の効率的な挿入

このトピックでは、パラレルでの表への新しい行の効率的な挿入方法について説明します。

リフレッシュ・プロセスの後半のフェーズでは、diff_customer一時表の新しい行をcustomers表に挿入します。更新の場合とは異なり、INSERT文の副問合せの指定は必要です。

```
INSERT /*+PARALLEL(customers)*/ INTO customers SELECT * FROM diff_customer s);
```

ただし、HASH_AJヒントを使用すると、この副問合せは必ずアンチハッシュ結合に変換されます。こうすると、パラレルINSERTを使用して、前の文を効率よく実行できます。パラレルINSERTは、表がパーティション化されていなくても適用できます。

8.8.7.3 パラレルでのマージによるパフォーマンスの最適化

このトピックでは、パラレルでのマージによるパフォーマンス最適化方法について説明します。

次の例に示すように、更新と挿入を組み合せて1つの文にすることができます。これは一般的には**マージ**と呼ばれます。

```
MERGE INTO customers USING diff_customer
ON (diff_customer.c_key = customer.c_key) WHEN MATCHED THEN
  UPDATE SET (c_name, c_addr) = (SELECT c_name, c_addr
  FROM diff_customer WHERE diff_customer.c_key = customers.c_key)
WHEN NOT MATCHED THEN
```

```
INSERT VALUES (diff_customer.c_key,diff_customer.c_data);
```

前述の例のSQL文では、「パラレルでの表の更新のためのパフォーマンスの最適化」および「パラレルでの表への新しい行の効率的な挿入」のすべての文と同じ結果をアーカイブします。

9 VLDBのバックアップおよびリカバリ

バックアップとリカバリは、DBAが業務データを保護するために行う重要で不可欠なジョブです。

データ記憶域が毎年増大するにつれ、DBAは、クリティカル・データをバックアップして、ビジネス・ニーズに合うように迅速かつ容易にリカバリできるように保証することを常に求められます。大規模データベースは、その大きさとデータが多くのリソースに由来するという点が独特です。OLTPとデータウェアハウスにはそれぞれの特性があります。通常、大規模OLTPシステムでの可用性の考慮事項は、小規模OLTPシステムの場合の考慮事項と変わりません。停止可能な時間が同じであれば、大規模OLTPシステムは小規模OLTPシステムよりも多くのハードウェア・リソースが必要になります。

この章では、大規模データベースのための効果的なバックアップとリカバリの戦略を提案します。この戦略は、OLTPシステムとは異なるデータ・ウェアハウスの特性を利用して、バックアップとリカバリをサポートするために必要なリソースを全面的に削減します。

この章の構成は、次のとおりです。

- データ・ウェアハウス
- Oracleのバックアップとリカバリ
- データ・ウェアハウスのバックアップとリカバリ
- データ・ウェアハウスのリカバリ方法

9.1 データ・ウェアハウス

データ・ウェアハウスは、分析や意思決定をサポートするために設計されたシステムです。

一般的な企業では、数百または数千名のユーザーが業務の理解に役立つ情報を得たり、適切な決定を行ったりするために、データ・ウェアハウスを利用しています。したがって、可用性はデータ・ウェアハウスの重要な要件です。この章では、データ・ウェアハウスの可用性の重要なポイントである、データ損失後のデータ・リカバリについて説明します。

バックアップとリカバリの技法を詳しく調べる前に、データ・ウェアハウスのバックアップとリカバリのための固有の技法について理解する必要があります。特に、データ・ウェアハウスのバックアップとリカバリの戦略は、その他のデータベース・システムと同じにするべきかという当然の疑問について説明します。

DBAは、データ・ウェアハウスのバックアップとリカバリの作業を始めるにあたり、OLTPシステムで使用されるのと同じ技法を適用します。つまり、重要性と変更の頻度に応じてデータの優先順位を付けて、保護すべき情報、およびメディア・リカバリが必要になったときに迅速にリカバリすべき情報を決定する必要があります。ただし、データ・ウェアハウスに関して通常生じる問題は、100GBのOLTPシステムでは効率がよくコスト効果が高い方法であっても、10TBのデータ・ウェアハウスには適さないということです。バックアップとリカバリの時間は100倍かかり、記憶域も100倍必要になります。

関連項目:

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

9.1.1 データ・ウェアハウスの特性

データ・ウェアハウスとOLTPシステムにはいくつかの違いがあり、バックアップとリカバリに大きく影響します。

それらの違いは次のとおりです。

1. 通常、データ・ウェアハウスはOLTPシステムよりもかなり大規模です。数十TBを超えるデータ・ウェアハウスもまれではなく、最大規模のデータ・ウェアハウスはさらに桁違いの大きさになります。このため、データ・ウェアハウスのバックアップとリカバリではスケーラビリティは特に重要な考慮事項です。
2. 多くの場合、データ・ウェアハウスの可用性要件はOLTPシステムに比べて低くなります。データ・ウェアハウスがビジネスにとってクリティカルな場合、数TBのデータを数時間でリカバリできるようにするためには、一日かけてリカバリする場合よりもかなりのコストがかかります。データ・ウェアハウスの大部分のリカバリを必要とするような障害は発生する可能性が低いため、組織によっては、バックアップ用のハードウェアやストレージにかける多額の支出を節約できるのであれば、1日以上の停止も容認する場合があります。
3. 通常、データ・ウェアハウスは、ユーザーが自らデータを変更するOLTPシステムとは異なり、ETL (抽出、変換、ロード)という制御プロセスによって更新されます。データ変更が制御プロセスによって行われるため、多くの場合、データ・ウェアハウスの更新はREDOログ以外のソースからも認識でき再現可能です。
4. データ・ウェアハウスには履歴情報が含まれます。また、多くの場合、データ・ウェアハウスの古いデータの大部分は静的です。たとえば、あるデータ・ウェアハウスでは売上の履歴データが5年分管理されます。最も新しい年のデータは(返品、決算修正などのために)変更される可能性がありますが、以前の4年分のデータは完全に静的です。静的データの利点は、頻繁にバックアップを行う必要がないことです。

これらの4つの特性は、データ・ウェアハウスに最適なバックアップおよびリカバリ戦略を検討する際の重要な考慮事項です。

9.2 Oracleのバックアップとリカバリ

一般的に、バックアップとリカバリは、データ損失に対するデータベースの保護や、なんらかのデータ損失の後でのデータベースの再構築に関連する様々な戦略や手順を意味します。

バックアップとはデータのかわりとなるコピーです。このコピーには、データベースの重要な部分(制御ファイル、アーカイブREDOログ、データファイルなど)を含めることができます。バックアップは、元のデータをリストアする手段を提供することにより、アプリケーション・エラーからデータを保護し、予期しないデータ損失に対する安全装置として機能します。

この項では、次の項目について説明します。

- データのリカバリで使用される物理データベース構造
- バックアップのタイプ
- バックアップ・ツール

9.2.1 データのリカバリで使用される物理データベース構造

バックアップおよびリカバリ戦略を本格的に検討する前に、バックアップとリカバリの操作に関連する物理データ構造を確認します。

これらのコンポーネントが、Oracleデータ・ストアのデータを構成するファイルやその他の構造を含み、このデータ・ストアを潜在的な障害から保護します。Oracle Databaseのリカバリには次の3つの基本コンポーネントが必要です。

- データファイル
- REDOログ
- 制御ファイル

9.2.1.1 データファイル

Oracle Databaseは、表領域と呼ばれる1つ以上の論理記憶域単位で構成されます。Oracle Databaseの各表領域は、データファイルという1つ以上のファイルで構成されます。データファイルは、Oracle Databaseが稼働しているホスト・オペレー

ティング・システムに存在する、またはホスト・オペレーティング・システムに接続する物理ファイルです。

データベースのデータは、データファイルにまとめて格納されており、データファイルがデータベースの各表領域を構成します。最も単純なOracle Databaseは、1つの表領域があり、1つのデータファイルが格納されます。データベースのデータファイルのコピーは、すべてのバックアップ戦略に欠かせない部分です。データファイルのサイズが大きいことが、VLDBのバックアップとリカバリにおいて主要な課題になります。

9.2.1.2 REDOログ

REDOログには、データベースのデータファイルに対する変更がすべて記録されます。

REDOログの完全なセットとそれ以前のデータファイルのコピーがあれば、Oracleで、REDOログに記録された変更を再適用して、バックアップ時点から最後のREDOログの終了時点までの任意の時点のデータベースを再作成できます。Oracle Databaseでデータが変更されるたびに、その変更がオンラインREDOログにまず記録されてからデータファイルに適用されます。

Oracle Databaseでは、少なくとも2つのオンラインREDOログ・グループが必要です。各グループには、少なくとも1つのオンラインREDOログ・メンバーつまり1つのREDOログ・ファイルが含まれ、このファイルに変更内容が記録されます。時間隔で、Oracle Databaseはオンライン REDOログ・グループを入れ替えます。現行オンラインREDOログに変更内容を格納する一方で、使用していないグループはアーカイブREDOログと呼ばれるアーカイブ場所にコピーできます。高可用性を実現するために、本番システムは、常に1グループ当たり複数のオンラインREDOメンバーを使用する必要があります。できればメンバーを異なる記憶域システムに配置することをお薦めします。アーカイブREDOログにはデータファイルのすべての更新記録が含まれるため、アーカイブREDOログの保存がバックアップ戦略の主要な部分です。多くのバックアップ戦略では、長期保管のためにアーカイブREDOログをディスクまたはテープにコピーします。

9.2.1.3 制御ファイル

制御ファイルには、データベースの物理構造とそのステータスの重要なレコードが含まれます。

制御ファイルに格納される次のタイプの情報は、バックアップとリカバリに関連します。

- 障害からリカバリするため、つまりメディア・リカバリを実行するために必要なデータベース情報
- データベース構造情報(データファイルの詳細など)
- REDOログの詳細
- アーカイブ・ログ・レコード
- 過去のRMANバックアップのレコード

Oracle Databaseのデータファイル・リカバリ・プロセスのある部分は、制御ファイルのステータス情報(データベース・チェックポイント、現行オンラインREDOログ・ファイル、データファイル・ヘッダー・チェックポイントなど)によって決まります。制御ファイルが失われると、データ損失からのリカバリがさらに困難になります。最新のデータベース構造変更を保存し、リカバリを容易に行えるように、制御ファイルを定期的にバックアップする必要があります。

9.2.2 バックアップ・タイプ

バックアップには、物理バックアップと論理バックアップがあります。

- **物理バックアップ**は、データベースの格納とリカバリに使用される物理ファイル(データファイル、制御ファイル、アーカイブREDOログなど)のバックアップです。究極的には、すべての物理バックアップは、データベース情報を格納するファイルを別の場所(ディスク上またはテープなどのオフライン・ストレージ上)にコピーすることです。
- **論理バックアップ**には、Oracle Data Pump (エクスポートおよびインポート)ユーティリティでデータベースから抽出された論理データ(たとえば、表またはストアド・プロシージャ)が含まれます。データは、Oracle Databaseにインポートで

きるバイナリ・ファイルに格納されます。

物理バックアップは、すべてのバックアップおよびリカバリ戦略の基盤です。論理バックアップは、様々な状況で物理バックアップを補足するために役立ちますが、物理バックアップがなければデータ消失に十分に対処することはできません。

データベースのすべてまたは一部の内容をバックアップから再構築するには、通常は2つの段階があります。まずバックアップからデータファイルのコピーを取り出し、次にバックアップ以降の変更内容をアーカイブREDOログおよびオンラインREDOログからファイルに再適用して、データベースの状態をリカバリ希望の時点に戻します。データファイルまたは制御ファイルをバックアップからリストアするには、テープ、ディスクまたは他のメディアのバックアップ場所からファイルを取得して、Oracle Databaseで使用できるようにします。データファイルをリカバリするには、データファイルをリストアし、データベースのREDOログに記録された変更内容を適用します。データベース全体をリカバリするには、データベースのデータファイルそれぞれについてリカバリを実行します。

9.2.3 バックアップ・ツール

Oracle Databaseのバックアップとリカバリを管理するためにいくつかのツールが提供されます。

各ツールではバックアップを作成するためにいくつかの基本的な方法から選択できます。次の方法があります。

- Oracle Recovery Manager(RMAN)

  RMANでは、すべてのバックアップと必要なリカバリ関連ファイルに関するメタデータの大量の記録が管理されるため、バックアップ戦略に伴う管理作業が削減されます。リストアおよびリカバリ操作ではRMANがこの情報を使用するため、ユーザーが必要なファイルを特定する必要がなくなります。RMANは効率性が高く、ファイル多重化およびパラレル・ストリームの機能をサポートしています。また、バックアップおよびリストアの際には、物理的な破損および論理的な破損(オプション)についてブロックの検査を行います。

  V$BACKUPビューを使用して、バックアップ・アクティビティ・レポートを生成できます。

- Oracle Data Pump

  Oracle Data Pumpでは、Oracle Databaseコンテンツのバルク・データおよびメタデータの高速かつパラレルの移動が可能です。このユーティリティは、Oracle Databaseのデータをオペレーティング・システム・ファイルに書き込むことで、論理バックアップを作成します。このデータは、後からOracle Databaseにインポートできます。

- ユーザー管理バックアップ

  オペレーティング・システム固有のコマンドを実行してデータベースを手動でバックアップします。

9.2.3.1 Oracle Recovery Manager(RMAN)

Oracle Recovery Manager (RMAN)はコマンドラインで、Oracle Databaseを効率よくバックアップおよびリカバリする際に使用することをお薦めします。

RMANは、サーバーと密接に作用するよう設計されており、バックアップおよびリカバリ中のブロックレベルの破損を検出します。RMANでは、ファイルの多重化とバックアップ・セットの圧縮により、バックアップ時のパフォーマンスと領域消費が最適化されます。また、含まれているメディア管理ライブラリ(MML) APIを使用することにより、主なテープおよびストレージ・メディア製品と統合されます。

RMANは、バックアップまたはリストアの前後で、基礎となるデータベース・プロシージャをすべて処理するため、オペレーティング・システムやSQL*Plusスクリプトの依存から解放されます。また、様々なホスト・オペレーティング・システムに対してバックアップ・タスクの共通インタフェースを提供し、ユーザー管理の方法では使用できない機能を提供します。これは、データファイルや表領域レベルのバックアップとリカバリ、バックアップおよびリカバリ・データ・ストリームのパラレル化、増分バックアップ、データベース構造変更に伴う制御ファイルの自動バックアップ、バックアップ保存ポリシー、すべてのバックアップの詳細履歴などです。

関連項目:

RMANの詳細は、『Oracle Databaseバックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください

9.2.3.2 Oracle Data Pump

物理バックアップを補助するために、Oracle Data Pump (エクスポートおよびインポート)ユーティリティを使用して、データの論理バックアップを作成できます。

論理バックアップには、データベースに対して作成されたスキーマ・オブジェクトの情報が格納されます。Oracle Data Pumpによって、データとメタデータが一連のオペレーティング・システム・ファイルにロードされます。これらのファイルは、同じシステムにインポートするか、別のシステムに移してそこでインポートすることができます。

ダンプ・ファイル・セットは、表データ、データベース・オブジェクトのメタデータ、制御情報を含む1つ以上のディスク・ファイルで構成されています。これらのファイルはバイナリ形式です。データ・ポンプ・インポート・ユーティリティは、インポート操作中、これらのファイルを使用してダンプ・ファイル・セット内の各データベース・オブジェクトの位置を特定します。

9.2.3.3 ユーザー管理バックアップ

Recovery Managerを使用しない場合は、オペレーティング・システムのコマンドを使用できます。たとえば、UNIXのddまたはtarコマンドを使用してバックアップを作成します。

ユーザー管理のオンライン・バックアップを作成するには、データベースを手動でホット・バックアップ・モードに切り替える必要があります。ホット・バックアップ・モードでは、オンライン・ログ・ファイルに追加の書き込みが行われ、ファイルのサイズが増大します。

スクリプトを作成してバックアップ操作を自動化することもできます。データベース全体のバックアップを一度に作成するか、表領域、データファイル、制御ファイルまたはアーカイブ・ログを個別にバックアップできます。データベース全体のバックアップを補完するように、表領域、データファイル、制御ファイル、アーカイブ・ログの個別バックアップを使用できます。

オペレーティング・システムのコマンドまたはサードパーティのバックアップ・ソフトウェアで、データベースのバックアップを実行できます。また、データベースのバックアップをリストアするためにはサードパーティのソフトウェアを使用する必要があります。

9.3 データ・ウェアハウスのバックアップとリカバリ

データ・ウェアハウスのリカバリは、OLTPシステムのリカバリに似ています。

ただし、データ・ウェアハウスでは、すべてのデータをバックアップからリカバリしなくても、あるいは全面的な障害時にデータベース全体のリストアを行わなくても、ユーザー・アクセスを開始することが可能です。データ・ウェアハウスで効率的かつ高速のリカバリを行うには、まず適切なバックアップを計画します。

次の項では、バックアップすべきデータの識別について説明し、最短期間でクリティカル・データをリカバリできる方法とツールを示します。

この項では、次の項目について説明します。

- リカバリ時間目標(RTO)
- リカバリ・ポイント目標(RPO)

9.3.1 リカバリ時間目標(RTO)

リカバリ時間目標 (RTO)は、データをリカバリする必要がある期限です。

バックアップおよびリカバリ計画は、企業がデータ・ウェアハウスについて設定したRTOを満たすように設計する必要があります。た

とえば、データベースの全面的な損失に際して、データの5%を12時間以内に使用可能にし、50%を2日以内に使用可能にし、残りのデータを5日以内に使用可能にすると決定する場合があります。この場合、RTOは2つあります。合計のRTOは7.5日です。

RTOを決定するには、データが使用できないことによる影響をまず特定する必要があります。RTOを設定するには、次の4つのステップを実行します。

1. 分析と特定: リカバリの準備状況、リスクがある分野、データが使用できないために発生するビジネス・コストを理解します。データ・ウェアハウスでは、停止後*n*日以内でリカバリする必要があるクリティカル・データを識別する必要があります。
2. 設計: リカバリ要件に基づいてバックアップおよびリカバリの戦略を作成します。これを行うには、論理的な関係と重要性に基づいてデータを分類します。
3. 構築と統合: データのバックアップとリカバリを行う環境にソリューションをデプロイして統合します。バックアップおよびリカバリ計画を文書化します。
4. 管理と展開: リカバリ計画を定期的にテストします。データ、ITインフラストラクチャおよびビジネス・プロセスが変更するにつれて、ソリューションを調整および更新するように変更管理プロセスを実装します。

9.3.2 リカバリ・ポイント目標(RPO)

リカバリ・ポイント目標つまりRPOは、組織に有害な結果をもたらすことのないデータ損失の最大許容量です。

RPOは、一般的に、ビジネス・プロセスまたは組織の許容データ損失量に相当します。このデータ損失は、通常、5時間または2日分のデータ損失というように、時間に換算して測定されます。RPOが0の場合、メディアの損害時にコミット済データの損失が許容されません。一方、RPOが24時間の場合は、1日分のデータ損失が許容されます。

この項では、次の項目について説明します。

- データ量の増加とバックアップ時間の延長
- 分断攻略

9.3.2.1 データ量の増加とバックアップ時間の延長

データ・ウェアハウスの最大の特徴はデータベースのサイズです。

サイズは最大では数百TBを上回ります。このため、ハードウェアが、高速のバックアップとリカバリを制限する要因になります。ただし、現在のテープ・ストレージは、テープにオフロードする必要があるデータ容量を収容できるように進化し続けています(たとえば、標準的なテープ・アクセス・インタフェースを備え、内部でディスクを使用する仮想テープ・ライブラリが出現しています)。RMANは、使用可能なすべてのテープ・デバイスをパラレルで最大限に活用して、バックアップとリカバリで最高のパフォーマンスを実現します。

基本的には、大規模データベースのバックアップに必要な時間は、最小スループット(本番ディスク、ホスト・バス・アダプタ(HBA)およびネットワークからテープ・デバイスへの転送、テープ・ドライブのストリーム仕様のいずれか)とテープ・ドライブ数を掛けて算出できます。また、RMANのバックアップ暗号化または圧縮が使用される場合は、ホストCPUがバックアップ全体のパフォーマンスを制限する要因になることがあります。バックアップおよびリカバリの期間は、適切なハードウェア・リソースがあれば、すべてのビジネス要件に合うように調整できます。

9.3.2.2 分断攻略

データ・ウェアハウスでは、データベースが全面的に使用されない期間があります。

この期間が数時間ずつ細切れになっている場合は、データベース全体をバックアップするには十分ではありません。数日間かけてデータベースのバックアップを分割して行うことを検討してください。RMANでは、所定のバックアップ・ジョブの実行可能時間を指定できます。BACKUP DURATIONを使用すると、バックアップを完了するまでできるだけ迅速に実行するか、バックアップによってデー

タベースが受ける負荷を最小限に抑えるようにゆっくり実行するかを選択できます。

次の例では、RMANによって、過去7日間にバックアップされていないすべてのデータベース・ファイルが最初にバックアップされます。また、実行時間は4時間で、ブロックをできるだけ高速で読み取ります。

```
BACKUP DATABASE NOT BACKED UP SINCE 'sysdate - 7'
  PARTIAL DURATION 4:00 MINIMIZE TIME;
```

このRMANコマンドが実行されるたびに、過去7日間にバックアップされていないデータファイルが最初にバックアップされます。毎晩バックアップされる表領域またはデータファイルを手動で指定する必要はありません。すべてのデータベース・ファイルが数日かけてバックアップされます。

これはデータベース・バックアップの単純な方法ですが、実装が容易であり、大容量データのバックアップに対しても柔軟性があります。リカバリ中には、RMANがリストア操作を実行するために複数の異なるストレージ・デバイスを指定することがあります。結果としてリカバリ時間が長くなることがあります。

9.4 データ・ウェアハウスのリカバリ方法

バックアップとリカバリの計画の考案は、複雑で困難な作業になる場合があります。

障害から保護し、リカバリする必要があるデータが何百TBもある場合、戦略は非常に複雑になります。ここでは、バックアップとリカバリの管理を容易にするために実装できるベスト・プラクティスをいくつか示します。

この項では、次の項目について説明します。

- ベスト・プラクティス1: ARCHIVELOGモードの使用
- ベスト・プラクティス2: RMANの使用
- ベスト・プラクティス3: ブロック・チェンジ・トラッキングの使用
- ベスト・プラクティス4: RMANマルチセクション・バックアップの使用
- ベスト・プラクティス5: 読取り専用表領域の活用
- ベスト・プラクティス6: バックアップまたはリカバリ戦略でのNOLOGGING操作の計画
- ベスト・プラクティス7: すべての表領域は同等に処理されない

9.4.1 ベスト・プラクティス1: ARCHIVELOGモードの使用

アーカイブREDOログは、データベースの変更内容の記録であるため、データの損失が許容されない場合にリカバリに不可欠です。

Oracle Databaseは、次の2つのモードのいずれかで実行できます。

- ARCHIVELOG

  Oracle Databaseでは、再利用される前に満杯になったオンラインのREDOログ・ファイルがアーカイブされます。

- NOARCHIVELOG

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

データベースをARCHIVELOGモードで実行すると次の利点があります。

- インスタンス障害とメディア障害のどちらでもデータベースをリカバリできます。
- データベースがオープンして使用可能な状態でバックアップを実行できます。
- Oracle Databaseは、アーカイブ・ログ上での単一点障害を回避するために、多重化アーカイブ・ログをサポートして

います。

- 表領域のポイント・イン・タイム・リカバリ(TSPITR)の実行など、ほとんどのリカバリ・オプションが使用可能です。
- アーカイブREDOログを、プライマリ・データベースの正確な複製である物理スタンバイ・データベースに送信して適用できます。

データベースをNOARCHIVELOGモードで実行すると次の影響があります。

- データベースをバックアップできるのは、正しく停止してクローズしているときだけです。
- 通常、メディア・リカバリ・オプションのみでは、全体バックアップまたは増分バックアップが行われたポイント・イン・タイムにデータベース全体がリストアされます。このとき、最新のトランザクションが失われることがあります。

9.4.1.1 停止時間の許容

データベースの中断を最小限に抑えるようにバックアップ計画を設計することが重要です。

Oracle Databaseのバックアップは、データベースがオープンしているときにもクローズしているときにも行えます。データベースの計画的な停止は、特に常時複数のタイムゾーンのユーザーをサポートする国際的な企業では、運用に混乱を招くことがあります。

業務によって異なりますが、一部の企業では停止時間が許容されます。業務全体の戦略で停止時間がほとんどまたはまったく不要な場合、バックアップ戦略ではオンライン・バックアップを実装する必要があります。バックアップのためにデータベースを停止する必要はありません。オンライン・バックアップには、データベースがARCHIVELOGモードであることが必要です。

データ・ウェアハウスのサイズ(およびデータ・ウェアハウスをバックアップするための時間)のため、通常、データ・ウェアハウスのオフライン・バックアップの作成は実現可能ではありません。ただし、NOARCHIVELOGモードを使用している場合には、強いて実行することもできます。

9.4.2 ベスト・プラクティス2: RMANの使用

以前のリリースのOracle Databaseで開発された多くのデータ・ウェアハウスには、バックアップとリカバリのためのRMANが統合されていないことがあります。

ただし、ARCHIVELOGモードを利用する理由が多数あるように、RMANを採用する強力な理由も多数あります。次の点を考慮してください。

1. 問題のないバックアップおよびリカバリ
2. 破損ブロックの検出
3. アーカイブ・ログの検証と管理
4. ブロック・メディア・リカバリ(BMR)
5. メディア・マネージャとの容易な統合
6. バックアップとリストアの最適化
7. バックアップとリストアの検証
8. 停止時間の不要なバックアップ
9. 増分バックアップ
10. 拡張性の高いレポート

9.4.3 ベスト・プラクティス3: ブロック・チェンジ・トラッキングの使用

ブロック・チェンジ・トラッキングを有効にすると、最後の全体バックアップまたは増分バックアップ以降に変更されたブロックのみを読み書きすることにより、増分バックアップを速く完了できるようになります。

データ・ウェアハウスでは、通常はデータベースの変更率が低いかもしくは中程度であるため、この機能はきわめて有効です。

関連項目:

ブロック・チェンジ・トラッキングの詳細は、『Oracle Databaseバックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください

9.4.4 ベスト・プラクティス4: RMANマルチセクション・バックアップの使用

bigfile表領域が出現したため、データ・ウェアハウスでは、大多数のデータファイルを管理しやすい少数のデータファイルにまとめることができます。

非常に大きなデータファイルのバックアップでは、RMANにより、ファイル内でのバックアップ操作をパラレル化する方法としてマルチセクション・バックアップが提供されます。この方法では、ファイル単位でバックアップが行われるかわりに、ファイルのセクションがパラレルでバックアップされます。

たとえば、1TBのデータファイルを100GBのバックアップ・ピース10個のセクションに分け、1TBのファイル全体を1つのファイルとしてバックアップするかわりに各セクションをパラレルでバックアップします。大きなデータファイルの全体のバックアップ時間が大幅に短縮できます。

関連項目:

マルチセクション・バックアップの構成の詳細は、『Oracle Databaseバックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください

9.4.5 ベスト・プラクティス5: 読取り専用表領域の活用

データ・ウェアハウスが直面する重要な問題は、典型的なデータ・ウェアハウスのサイズの大きさです。高性能のバックアップ・ハードウェアを使用しても、バックアップには数時間がかかることがあります。

バックアップのパフォーマンスを改善する重要な考慮事項は、バックアップするデータ容量を最小限に抑えることです。読取り専用表領域は、データ・ウェアハウスでバックアップを行うデータ容量を削減する最も単純なメカニズムです。増分バックアップの場合でさえ、表領域が読取り専用に設定されているとバックアップとリカバリの処理が速くなります。

読取り専用表領域の利点は、データを1回のみバックアップすればよいという点です。データ・ウェアハウスに5年分の履歴データが含まれており、最初の4年分のデータを読取り専用にできる場合、理論的には、データベースの定期バックアップではデータの20%のみがバックアップされることになります。このように、データ・ウェアハウスのバックアップに必要な時間が大幅に短縮されます。

ほとんどのデータ・ウェアハウスでは、データは、時間でレンジ・パーティション化された表に格納されます。一般的なデータ・ウェアハウスでは、通常、データがアクティブなのは30日から1年の間です。この期間中であれば、履歴データはまだ更新と変更の可能性があります(たとえば、ある小売業者は購入日から30日間は返品を受け付けるため、売上データのレコードはこの期間に変更できます)。ただし、一定の時間が経過すると、データは多くの場合は静的になります。

パーティション化を利用すると、ユーザーは読取り専用のデータの静的な部分を作成できます。現在、Oracleでは、読取り専

用パーティションまたは読取り専用表ではなく読取り専用表領域がサポートされています。読取り専用表領域を利用して、バックアップにかかる時間を短縮するには、一定のデータ・パーティションを読取り専用表領域に格納する戦略を立てる必要があります。ローリング・ウィンドウを実装するための2つの戦略を次に示します。

1. パーティション内のデータが完全に静的になる時点に達すると、定期的にスケジュールしたプロセスを実装し、現在の読取り/書込み表領域からある表領域にパーティションを移動して、その表領域を読取り専用にできます。
2. それぞれが少数のパーティションを含む一連の表領域を作成し、1つの表領域のデータが古くなると、定期的にその表領域を読取り/書込みから読取り専用に変更します。

データのバックアップはリカバリ・プロセスの半分にすぎないことに注意してください。データ・ウェアハウスの読取り/書込み部分を4時間でバックアップできるようにテープ・システムを構成した場合、そのデータベースの80%が読取り専用のときに完全リカバリが必要になると、そのテープ・システムではデータベースのリカバリに20時間かかることになります。

9.4.6 ベスト・プラクティス6: バックアップまたはリカバリ戦略でのNOLOGGING操作の計画

一般的に、データ・ウェアハウスの最優先事項はパフォーマンスです。データ・ウェアハウスは、オンライン・ユーザーに優れた問合せパフォーマンスを提供する必要があるだけではなく、大容量のデータを最短の時間でロードできるように、抽出、変換およびロード(ETL)プロセスでも効率が高いことが必要です。データ・ウェアハウスでよく利用されている最適化の1つは、NOLOGGINGモードを使用したバルク・データ操作の実行です。

NOLOGGINGモードをサポートするデータベース操作は、ダイレクト・パスのロード操作と挿入操作、索引作成および表作成です。操作をNOLOGGINGモードで実行すると、データはREDOログに書き込まれません(厳密には小さなメタデータのセットのみがREDOログに書き込まれます)。このモードはデータ・ウェアハウスでよく使用され、バルク・データ操作のパフォーマンスが最大で50%改善されます。

ただし、リカバリをサポートするために必要なデータがログ・ファイルに書き込まれることはないので、従来のバックアップ・メカニズムを使用してNOLOGGING操作をリカバリすることはできなくなります。さらに、NOLOGGING操作が発生したデータに対する後続の操作は、たとえその操作がNOLOGGINGモードを使用していなかったとしてもリカバリされません。NOLOGGING操作により提供されるパフォーマンスの向上のため、データ・ウェアハウスでは、一般的にETLプロセスでNOLOGGINGモードを使用することをお薦めします。

バックアップおよびリカバリ計画を考案する場合は、NOLOGGING操作の存在を考慮する必要があります。データベースがNOLOGGING操作に依存している場合、従来のリカバリ戦略(最新のテープ・バックアップからリカバリしてアーカイブ・ログ・ファイルを適用する)は適用できません。ログ・ファイルでは、NOLOGGING操作をリカバリできないためです。

覚えておく必要のある第1の原則は、NOLOGGING操作が発生しているときにバックアップを作成しないことです。このルールは、今のところ強制的には適用されないため、NOLOGGING操作とバックアップ操作が重ならないように、DBAがバックアップ・ジョブとETLジョブをスケジュールする必要があります。

NOLOGGING操作が存在するバックアップおよびリカバリには、2つのアプローチがあります。ETLバックアップまたは増分バックアップです。データ・ウェアハウス内でNOLOGGING操作を使用していない場合は、次のオプションのいずれも選択する必要はありません。アーカイブ・ログを使用してデータ・ウェアハウスをリカバリできます。ただし、次のオプションを使用すると、アーカイブ・ログを使用する方法よりもリカバリのパフォーマンスがある程度向上することがあります。また、フラッシュバック・ログおよび保証付きリストア・ポイントを使用して、データベースを前のポイント・イン・タイムにフラッシュバックすることもできます。

この項では、次の項目について説明します。

- 抽出、変換およびロード
- 抽出、変換およびロード戦略
- 増分バックアップ

- 増分アプローチ
- フラッシュバック・データベースと保証付きリストア・ポイント

9.4.6.1 抽出、変換およびロード

ETLプロセスでは、データをデータ・ウェアハウスにロード(再ロード)するために、Oracle機能といくつかの方法の組合せが使用されます。

次の機能が使用されます。

- トランスポータブル表領域

  トランスポータブル表領域を使用すると、ユーザーがOracle Database間で表領域を迅速に移動できます。この方法は、データベース間で大量データを移動するのに最も効率的です。Oracle Databaseにより、プラットフォーム間で表領域を転送できる機能が提供されます。ソース・プラットフォームとターゲット・プラットフォームが異なるエンディアンネスである場合、RMANによりターゲット・フォーマットに転送されている表領域が変換されます。

- SQL*Loader

  SQL*Loaderにより、外部フラット・ファイルからOracle Database上の複数の表にデータがロードされます。強力なデータ解析エンジンによって、あらゆるデータ形式のデータファイルに対応できます。

- データ・ポンプ(エクスポートおよびインポート)

  Oracle Data Pumpにより、データベース間でデータとメタデータを高速で移動できます。このテクノロジは、Oracleのデータ・ポンプ・エクスポートおよびデータ・ポンプ・インポート・ユーティリティの基本です。

- 外部表

  外部表機能は、既存のSQL*Loader機能を補足する機能です。この機能により、データベース内の表にあるデータと同様に、外部ソースのデータにアクセスできるようになります。外部表をデータ・ポンプ・ドライバと一緒に使用すると、CREATE TABLE AS SELECT * FROMを使用してデータベースのデータをエクスポートしてから、データをOracle Databaseにインポートすることもできます。

9.4.6.2 抽出、変換およびロード戦略

アプローチの1つは、定期的にデータベースのバックアップをして、その週全体のETLプロセスを再作成するために必要なデータファイルを保存することです。

リカバリが必要な場合、データ・ウェアハウスは最新のバックアップからリカバリされます。次に、従来のリカバリのシナリオで行われていたようにアーカイブREDOログを適用してロールフォワードを行うかわりに、データ・ウェアハウスはETLプロセスをリプレイすることでロールフォワードを行います。この方法では、ETLプロセスが容易に再生できることが前提です。通常、再生時にはETLプロセスごとに一連の抽出ファイルが格納されます。

この方法の実装例としては、データ・ウェアハウスのバックアップを毎週末に作成し、ETLプロセスのサポートに必要なファイルを毎晩格納します。データベースをリカバリするために最大で7日間のETL処理を再適用する必要があります。データ・ウェアハウス管理者は、テープからのリカバリ速度と以前のETL実行のパフォーマンス・データに基づいて、データ・ウェアハウスのリカバリにかかる時間を簡単に予測できます。

基本的に、データ・ウェアハウス管理者は、NOLOGGING操作によってETLプロセスのパフォーマンスを改善できますが、リカバリ・プロセスが少し複雑になり自動処理が減るという代償があります。多くのデータ・ウェアハウス管理者はこれを望ましいトレードオフととらえています。

この方法の1つの短所は、データ・ウェアハウスで行われた関連する変更をすべて管理する負担がデータ・ウェアハウス管理者にかかることです。この方法では、ETLプロセスに含まれない変更は取得されません。たとえば、データ・ウェアハウスによってはユー

ザーが独自の表やデータ構造を作成できます。このような変更はリカバリ中に失われてしまいます。

この制限はエンドユーザーに伝達される必要があります。かわりに、エンドユーザーに対して、すべてのプライベート・データベース・オブジェクトを別の表領域に作成するように指示し、リカバリ時には、DBAが従来のリカバリでその表領域をリカバリし、残りのデータベースはETLプロセスを再実行する方法でリカバリすることもできます。

9.4.6.3 増分バックアップ

NOLOGGING操作が存在する場合、バックアップおよびリカバリ戦略に自動化を取り入れるにはRMANの増分バックアップ機能を利用します。

増分バックアップは、直前のバックアップ以降に変更されたブロックのみをバックアップする機能です。データファイルの増分バックアップは、ブロック単位でデータの変更を取得します。データファイルのすべての使用済ブロックのバックアップは必要ありません。データファイルのすべてのブロックに変更がないかぎり、結果として生成されるバックアップ・セットは、通常は完全データファイル・バックアップよりも小さくて効率的です。

ブロック変更トラッキングを有効にする場合は、Oracle Databaseにより、すべてのデータベースの変更の物理的な場所が追跡されます。RMANはチェンジ・トラッキング・ファイルを自動的に使用して、増分バックアップで読み取る必要があるブロックを判別します。ブロック・チェンジ・トラッキング・ファイルのサイズは、データベースの合計サイズのおよそ30000分の1です。

関連項目:

ブロック・チェンジ・トラッキングの詳細と有効化の方法は、『Oracle Databaseバックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください

9.4.6.4 増分アプローチ

このアプローチを使用する通常のバックアップおよびリカバリ計画は、毎週末データ・ウェアハウスをバックアップし、ETLプロセスが完了した後は毎晩データ・ウェアハウスの増分バックアップを取得します。

増分バックアップは従来のバックアップと同様に、NOLOGGING操作と同時には実行できません。データ・ウェアハウスをリカバリするには、データベース・バックアップをリストアし、次に毎晩の増分バックアップを再適用します。

NOLOGGING操作はアーカイブ・ログには取得されませんが、NOLOGGING操作によるデータは増分バックアップに含まれます。さらに、前のアプローチとは異なり、このバックアップおよびリカバリ計画は、RMANを使用して管理できます。

9.4.6.5 フラッシュバック・データベースと保証付きリストア・ポイント

フラッシュバック・データベースは、広範に及ぶ論理エラーを修正するための高速で連続的なポイント・イン・タイム・リカバリ方法です。

フラッシュバック・データベースは、フラッシュバック・ログと呼ばれる別のロギングを利用します。これは、高速リカバリ領域に作成され、リカバリ・ニーズに対応するようにユーザーが定義した期間にわたり保持されます。これらのログには、ブロックの更新時に元のブロック・イメージが記録されます。

フラッシュバック・データベース操作が実行されると、変更されたデータに対応するブロック・イメージのみがリストアおよびリカバリされます。これに対して、従来のデータファイル・リストアでは、バックアップ内のすべてのブロックをリストアしないとリカバリを開始できませんでした。フラッシュバック・ログはREDOログと比例するように作成されます。

非常に大規模で頻繁に使用されるデータベースでは、連続するポイント・イン・タイム・リカバリのために必要なすべてのフラッシュバック・ログを保存しておくことは実現不可能です。ただし、場合によっては、バッチ実行中の論理エラーの発生に備えて、特定のポイント・イン・タイム・スナップショットを作成する必要があります(たとえば、毎晩のバッチ・ジョブの直前など)。このシナリオでは、フ

ラッシュバック・ロギングを有効化せずに保証付きリストア・ポイントを作成できます。

保証付きリストア・ポイントが作成されると、他のポイント・イン・タイムではなく保証付きリストア・ポイントに対するフラッシュバック・データベースのみに対応するように、フラッシュバック・ログが管理され、領域が節約されます。たとえば、保証付きリストア・ポイントは、ロギングなしのバッチ・ジョブの前に作成できます。保証付きリストア・ポイントを作成する前の1時間以内にロギングなしの操作が行われていないかぎり、保証付きリストア・ポイントへのフラッシュバック・データベースによりロギングなしのバッチ・ジョブが元に戻されます。ロギングなしのバッチ・ジョブが終了した後の時刻にフラッシュバックするには、バッチ・ジョブが終了して少なくとも1時間経過してから保証付きリストア・ポイントを作成します。

このシナリオにおいて、保証付きリストア・ポイントに対するフラッシュバック・ログ領域の見積りは、保証付きリストア・ポイントを保持する日数の間にデータベースのどれくらいの容量が変更されるかによって異なります。たとえば、保証付きリストア・ポイントを2日間保持し、データベースで100GB分の変更を予定する場合、フラッシュバック・ログとして100GBを用意します。100GBは、変更の回数ではなく、保証付きリストア・ポイントの作成後に変更されるデータベースのサブセットを表します。

9.4.7 ベスト・プラクティス7: すべての表領域は同等に処理されない

バックアップおよびリカバリという観点で見ると、データ・ウェアハウス内のすべての表領域が同等に重要だということはありません。

データベース管理者は、この情報に基づいて、より効率的なバックアップおよびリカバリ計画を必要に応じて考案できます。バックアップおよびリカバリの基本粒度は表領域なので、異なる表領域には異なるバックアップおよびリカバリ計画が存在する可能性があります。

最も基本的なレベルでは、一時表領域はバックアップする必要がありません(RMANにより規定されるルール)。さらに、一部のデータ・ウェアハウスでは専用のスクラッチ表領域があり、ユーザーが一時的な表や増分結果を格納できます。このような表領域は明示的な一時表領域ではありませんが、本質的には一時表領域として機能します。業務要件によっては、このような表領域をバックアップしてリストアする必要がないこともあります。かわりに、このような表領域に損害が発生した際には、ユーザーが自らのデータ・オブジェクトを再作成します。

データ・ウェアハウスの多くは、その他のデータより重要な一部データを持っています。たとえば、データ・ウェアハウス内の売上データは非常に重要で、リカバリの段階では、このデータはできるだけ早くオンラインにする必要があります。ただし、同じデータ・ウェアハウスにおいて、企業Webサイトのクリックストリーム・データを格納する表は業務にとってそれほど重要ではありません。業務の面では、このデータが数日間オフラインになっていても許容されます。つまり、データベース・ファイルが損失した場合に、クリックストリーム・データが数日間なくても問題ありません。このシナリオでは、売上データを含む表領域は頻繁にバックアップする必要があるが、クリックストリーム・データを含む表領域のバックアップは1週間もしくは2週間に1回ですみます。

最も単純なバックアップおよびリカバリ・シナリオでは、データベースのすべての表領域を同じように扱いますが、Oracle Databaseでは、DBAが必要に応じて表領域ごとにバックアップおよびリカバリ・シナリオを調整できるような柔軟性を備えています。

10 VLDBの記憶域管理

VLDB環境でのデータベース・ファイルのための記憶域管理には、高可用性、パフォーマンスおよび管理性の側面があります。

データ・ウェアハウス環境の記憶域のパフォーマンスは、多くの場合、I/Oスループット(MB/s)に換算されます。オンライン・トランザクション処理(OLTP)システムでは、1秒当たりのI/Oリクエスト数(IOPS)がパフォーマンスの重要な尺度です。

この章では、VLDB環境のみに関してデータベース・ファイルの記憶域管理を説明します。Oracle Databaseソフトウェアなどデータベース以外のファイルについては、管理方法がVLDB以外の環境と変わらないためここでは扱いません。VLDB環境の記憶域システムの高可用性、パフォーマンスおよび管理性の側面を中心に説明します。

この章の構成は、次のとおりです。

- 高可用性
- パフォーマンス
- スケーラビリティと管理性
- VLDB固有のOracle ASM設定

ノート:

Oracle Database では、RAW デバイスやファイルシステム上のデータベース・ファイルを使用できます。また、RAW デバイスまたは論理ボリューム上での Oracle Automatic Storage Management(Oracle ASM)の使用もできます。可能であれば Oracle ASM を使用することが推奨されます。

10.1 高可用性

高可用性は、記憶域の冗長性を実装することで達成できます。

これを記憶域の用語ではミラー化技法と呼びます。データベース環境でのミラー化には次の3つの方法があります。

- ハードウェアベースのミラー化
- Oracle ASMを使用したミラー化
- ソフトウェアベースのミラー化(Oracle ASMは使用しない)

  Oracle ASMを使用しないソフトウェアベースのミラー化はお薦めしません。

この項では、次の項目について説明します。

- ハードウェアベースのミラー化
- Oracle ASMを使用したミラー化

ノート:

クラスタ構成では、使用するソフトウェアがクラスタ機能をサポートする必要があります。Oracle ASM は、Oracle Database ファイル用のクラスタ・ファイルシステムです。

10.1.1 ハードウェアベースのミラー化

ほとんどの外部ストレージ・デバイスでは、様々なRAID (Redundant Array of Independent Disks)レベルがサポートされています。

VLDB環境で最もよく使用される高可用性ハードウェアRAIDレベルは、RAID 1およびRAID 5です。VLDB環境ではそれほど一般的ではありませんが、他の高可用性RAIDレベルも使用できます。

この項では、次の項目について説明します。

- RAID 1でのミラー化
- RAID 5でのミラー化

10.1.1.1 RAID 1でのミラー化

RAID 1は基本的なミラー化技術です。

記憶域に書き込まれるすべての記憶域ブロックは、RAID設定で定義されたように異なる2つの物理デバイスに格納されます。RAID 1ではフォルト・トレランスが提供されます。つまり、1台のデバイスで障害が発生すると、もう1台のミラー化されたデバイスがデータのリクエストに応答できます。RAID 1設定での2つの書込みは記憶域レベルで生成されます。RAID 1が有効に機能するには少なくとも2つの物理ディスクが必要です。

ストレージ・デバイスには、通常、リクエストが届いたときにプライマリとミラーのいずれかを読み取る機能があります。これにより、高可用性対応に設計された他のRAID構成に比べて高いパフォーマンスが得られます。RAID 1は、最も単純なハードウェア高可用性実装ですが、データを格納するために必要な記憶域が2倍になります。多くの場合、RAID 1はRAID 0 (ストライプ化)と組み合せてRAID 0+1構成で使用されます。最も単純なRAID 0+1構成では、個々のストライプが2台の物理デバイス間でミラー化されます。

10.1.1.2 RAID 5でのミラー化

RAID 5には、少なくとも3台のストレージ・デバイスが必要です。通常は4から6台のデバイスが1つのRAID 5グループで使用されます。

RAID 5を使用すると、デバイスに書き込まれるデータ・ブロックごとにパリティが計算されて、別のデバイスに格納されます。読取り操作時にパリティがチェックされます。パリティ計算は記憶域レイヤーで行われます。RAID 5でデバイス障害に対する高可用性が得られるのは、別のデバイスに格納されているパリティに基づいてデバイスの内容を再構築できるためです。

RAID 5では優れた読取りパフォーマンスも提供されます。書込みのパフォーマンスは、記憶域レイヤーでのパリティ計算のために遅くなることがあります。RAID 5では、グループ内のデバイス数によって異なりますが、必要な記憶域容量は2倍よりは少なくなります。RAID 5は比較的複雑なため、ストレージ・デバイスによってはRAID 5設定がサポートされないものもあります。

10.1.2 Oracle ASMを使用したミラー化

Oracle Automatic Storage Management(Oracle ASM)は、ソフトウェアベースのミラー化機能を提供します。

Oracle ASMでは、通常の冗長性(ミラー化)と高い冗長性(トリプル・ミラー化)がサポートされます。また、Oracle ASMでは、外部冗長性の使用もサポートされます。この場合、Oracle ASMでさらにミラー化が実行されることはありません。Oracle ASMの通常の冗長性は、RAID 1ハードウェア・ミラー化と同等です。

Oracle ASMによるミラー化では、ミラーはデータベース・サーバーによって作成されます。このため、Oracle ASMミラー化を使用すると、ハードウェアベースのミラー化に比べて、書込み操作時にさらにI/Oスループットが必要になります。構成やハードウェアRAIDコントローラの速度によって異なりますが、Oracle ASMによるミラー化またはハードウェアRAIDではデータ・ロードのボトル

ネックが発生することがあります。

Oracle ASMでは、障害グループの定義により冗長性が有効になります。Oracle ASMによって障害グループの境界を超えてデータがミラー化されるためです。たとえば、VLDB環境では、1つのディスク配列ごとに1つの障害グループを定義できます。この場合、Oracle ASMによって、ミラー化されたデータは必ず別のディスク配列に格納されます。このようにして、1つのディスク配列の1つのディスクの障害を切り抜けるだけではなく、ディスク配列全体の障害やディスク配列のすべてのチャネルの障害も乗り越えることができます。通常、ハードウェアRAID構成ではこのようなフォルト・トレランスはサポートされません。

通常の冗長性を使用するOracle ASMでは、データを格納するために必要なディスク領域が2倍になります。高い冗長性では3倍のディスク領域が必要です。

関連項目:

Oracle Automatic Storage Management管理者ガイド

10.2 パフォーマンス

ストレージ・デバイスの最適なスループットを実現するには、複数のディスクをパラレルで作動させる必要があります。

ストライプ化という技法を使用して、最適スループットを実現できます。この技法では、データ・ブロックは、複数のデバイスにわたり同一サイズの部分(ストライプ)に格納されます。ストライプ化により、優れたパフォーマンスとスループットを得るための記憶域構成が可能になります。

ストレージ・デバイスの最適なパフォーマンスは、シーク・タイムと、ディスク上の連続ブロックへのアクセスのトレードオフで決まります。VLDB環境では、OLTPシステムとデータ・ウェアハウス・システムの両方において、ストライプ・サイズを1MBにすると、最適なパフォーマンスとスループットを実現するための適切なバランスを得られます。データベース環境でのストライプ化には次の3つの方法があります。

- ハードウェアベースのストライプ化
- ソフトウェアベースのストライプ化(Oracle ASMを使用する)
- ソフトウェアベースのストライプ化(Oracle ASMを使用しない)

ストライプ化技法を組み合せて使用することが可能ですが、ストライプ化によってパフォーマンスを向上させるためには、ストライプを異なるデバイスに物理的に格納することを確認してください。概念の面では、Oracle ASMを使用しないソフトウェアベースのストライプ化はハードウェアベースのストライプ化と大変よく似ています。

この項では、次の項目について説明します。

- ハードウェアベースのストライプ化
- Oracle ASMを使用したストライプ化
- 情報ライフサイクル管理
- パーティション配置
- ビッグファイル表領域
- Oracle Database File System(DBFS)

ノート:

クラスタ構成では、使用するソフトウェアがクラスタ機能をサポートする必要があります。Oracle ASM は、Oracle Database ファイル用のクラスタ・ファイルシステムです。

10.2.1 ハードウェアベースのストライプ化

一番外側にあるストレージ・デバイスがストライプ化機能を提供します。記憶域のパフォーマンスを向上するために最もよく使用されるストライプ化技法は、RAID 0およびRAID 5です。

この項では、次の項目について説明します。

- RAID 0でのストライプ化
- RAID 5でのストライプ化

10.2.1.1 RAID 0でのストライプ化

RAID 0では少なくとも2台のデバイスを実装する必要があります。

デバイスに書き込まれるデータ・ブロックは、ストライプ・サイズに分割され、複数のデバイスに交互に格納されます。この技法により、複数デバイスと、デバイスの複数チャネルの使用が可能になります。

RAID 0には、RAIDという名前が付いていますが冗長性はありません。RAID 0構成でデバイスに障害が発生すると、データは損失されます。重要な環境では必ずなんらかの冗長性と組み合せる必要があります。RAID 0を使用するデータベース実装は、多くの場合、RAID 1 (基本のミラー化)と組み合せてRAID 0+1構成で使用されます。

10.2.1.2 RAID 5でのストライプ化

RAID 5構成では、ハードウェア固有のストライプ・サイズを使用して、RAIDグループ内の使用可能なデバイスにデータが分散されます。

結果として、複数のデバイスとチャネルがデータの読書きに使用されます。複雑なパリティ計算のため、ストレージ・デバイスによってはRAID 5構成がサポートされないこともあります。

10.2.2 Oracle ASMを使用したストライプ化

Oracle Automatic Storage Management(Oracle ASM)は、ディスク・グループとして認識されるすべてのデバイスに対して常にストライプ化を行います。

ディスク・グループは、データファイルを作成するための論理的な記憶域プールです。通常は、デフォルトのOracle ASMストライプ・サイズは、VLDBに適したストライプ・サイズです。

ディスク・グループ内ではパフォーマンス特性が同じディスクを使用してください。また、データを最適に分散して最適なパフォーマンスとスループットを得るには、ディスク・グループのすべてのディスクを同じサイズにする必要があります。最高のパフォーマンスを実現するために、ディスク・グループはできるだけ多くの物理スピンドルを含むようにしてください。VLDB用のディスク・グループ構成を、VLDB以外のディスク・グループ構成と変える必要はありません。

Oracle ASMは、すでにストライプ化されたストレージ・デバイスで使用できます。そのような構成を使用する場合は、論理デバイスにまたがるディスク・グループを定義してホットスポットを生成しないようにしてください。それらの論理デバイスが、他の使用可能なリソースではなく同一のリソース(ディスク、コントローラ、またはディスクのチャネル)を物理的に使用していることがあるためです。Oracle ASMストライプがすべての物理デバイスに常に均等に分散されるようにします。

関連項目:

Oracle ASMストライプ化の詳細は、『Oracle Automatic Storage Management管理者ガイド』を参照してください

10.2.3 情報ライフサイクル管理

情報ライフサイクル管理(ILM)環境では、すべてのデバイスに対するストライプ化は、すべてのデータがすべての記憶域プールに分散されるため使用できません。

ILM環境では、通常、記憶域プールごとにパフォーマンス特性が異なります。表領域が記憶域プールにまたがらないようにしてください。つまり、同じ表領域のデータファイルを複数の記憶域プールに格納しないでください。

ILM環境の記憶域は、1つの記憶域プールのすべてのデバイスに対するストライプ化を使用するように構成する必要があります。Oracle ASMを使用する場合は、記憶域プールごとに別のディスク・グループを作成します。この方法では、表領域のデータファイルが異なるディスク・グループに格納されることはありません。データはオンラインのまま表領域間で移動できます。パーティション表にはパーティション移動操作を使用し、表がパーティション化されていない場合はDBMS_REDEFINITIONパッケージを使用します。

関連項目:

情報ライフサイクル管理環境の詳細は、「時間ベース情報の管理およびメンテナンス」を参照してください

10.2.4 パーティション配置

使用可能なすべてのデバイスに対してストライプ化し、使用可能なすべてのリソースに負荷を分散する場合、パーティション配置は問題になりません。

使用可能なすべてのデバイスにデータファイルをストライプ化できない場合は、使用可能なすべてのハードウェア・リソース(物理ディスク・スピンドル、ディスク・コントローラ、およびディスクのチャネル)の使用状況を最適化するためにパーティション配置を検討します。

I/O集中型の問合せまたはDML操作では、使用可能なすべてのリソースを最適に活用する必要があります。データベース・オブジェクト・パーティションを特定の表領域に格納し、各表領域が異なるハードウェア・リソースのセットを使用する場合、1つのパーティション・データベース・オブジェクトに対する操作ですべてのリソースを使用できるようになります。適切なパーティション化技法を使用して、I/O集中型の操作がすべてのリソースを使用できるようにしてください。

パーティション配置を使用してI/Oリソース使用率を最適化する場合、ワークロードをできるだけ均等に分散するただ1つの方法としては、ハッシュ・パーティション数を2の累乗とした、一意またはほぼ一意の列(または一連の列)に対するハッシュ・パーティション化とハッシュ・サブパーティション化があります。その他のパーティション化およびサブパーティション化の技法でも、アプリケーションによっては同様の効果が得られる場合があります。

10.2.5 ビッグファイル表領域

Oracle Databaseではbigfile表領域を作成できます。

bigfile表領域は、最大128TBの1つのデータファイルまたは一時ファイルで構成されます。bigfile表領域を使用すると、データベースのデータファイル数を大幅に削減できます。Oracle Databaseでは、1つのデータファイルに対するRMANのバックアップ

とリストアのパラレル実行をサポートします。

結果として、bigfile表領域の使用にデメリットはありません。データファイルや一時ファイルの数を大幅に削減するために、bigfile表領域の使用を選択できます。

ファイルの割当てはシリアル・プロセスで行われます。表および自動拡張可能データファイルの自動割当てを使用すると、bigfile表領域を使用するかどうかにかかわらず、ファイルの拡張に時間がかかるため大きなデータのロードが影響を受けることがあります。ただし、データファイルを事前に割り当てて、複数データファイルを使用する場合は、複数のプロセスを生成してデータファイルを同時に追加することができます。

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』

10.2.6 Oracle Database File System(DBFS)

Oracle Database File System(DBFS)は、ファイルを格納するデータベースの利点と、リレーショナル・データを効率的に管理するデータベースの強みを活用して、データベースに格納されたファイルに対する標準のファイルシステム・インタフェースを実装します。

このインタフェースを使用すると、BLOBおよびCLOBプログラム・インタフェースを使用するために特別に記述されたプログラムに制限されずに、データベースにファイルを格納できるようになります。これで、ファイルに作用する任意のオペレーティング・システム(OS)プログラムを使用して、データベース内のファイルに透過的にアクセスできるようになります。

Oracle Database File System(DBFS)は、データベース表に格納されたファイルおよびディレクトリの上部に標準ファイルシステム・インタフェースを作成します。DBFSを使用する場合、サーバーはデータベースです。ファイルは、Oracle SecureFiles LOBとしてデータベース表に格納されます。PL/SQLプロシージャのセットにより、作成、オープン、読取り、書込みおよびリスト・ディレクトリなどのファイルシステム・アクセスのプリミティブが実装されます。データベース内のファイルシステムの実装は、DBFSコンテンツ・ストアと呼ばれます。DBFSコンテンツ・ストアにより、すべてのデータベース・ユーザーは、クライアントによってマウント可能な1つ以上のファイルシステムを作成できます。各ファイル・システムには、ファイル・システム・コンテンツが保持されるシステム専用の表があります。

関連項目:

Oracle SecureFiles LOB、ストアおよびOracle Database File System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

10.3 スケーラビリティと管理性

記憶域のスケーラビリティと管理性は、VLDB環境の重要な要素です。

VLDBのきわめて重要な特性はサイズが大きいことです。サイズが大きいために次の課題があります。

- VLDBでは使用されるコンポーネントが多いため、単純な統計により、記憶域コンポーネントで障害が発生する可能性が高いことが示されます。
- VLDBでの小規模な相対成長は、大規模な絶対成長に相当することがあり、多くのデバイスの追加につながる場合があります。

- サイズの大きさにかかわらず、パフォーマンスおよび可用性の要件は小規模システムと変わりません。

これらの課題に対処できる記憶域構成を選択する必要があります。記憶域の追加や削除を計画的に行うか状況に応じて行うかにかかわらず、システムはパフォーマンスと高可用性の観点で最適な状態を保つ必要があります。

この項では、次の項目について説明します。

- SAME(Stripe and Mirror Everything)
- SAMEと管理性

10.3.1 SAME(Stripe and Mirror Everything)

SAME(Stripe and Mirror Everything)方式は、長い間オラクル社によって推奨されている、高可用性、パフォーマンスおよび管理性を最適化する方法です。

構成をさらに単純化するために、SAME方式では、OLTPシステムとデータ・ウェアハウス・システムの両方での適切な開始値として1MBの固定ストライプ・サイズが推奨されます。Oracle ASMではSAME方式が実装され、さらに自動化も行われます。

10.3.2 SAMEと管理性

最大のパフォーマンスを実現するため、SAME方式では、できるだけ多くの物理デバイスに対するストライプ化が提案されます。

これはOracle ASMなしでも達成できます。ただし、デバイスの追加または削除などにより記憶域構成が変わると、デバイス上のデータベース・ファイルのレイアウトも変える必要があります。Oracle ASMではこのタスクがバックグラウンドで自動的に実行されます。Oracle ASM以外の環境のほとんどでは再ストライプ化が主要な作業であり、多くの場合は手動の処理が必要です。

ILM環境では、すべての記憶域プールにSAME方式を適用します。

10.4 VLDB固有のOracle ASM設定

VLDB用のOracle Automatic Storage Managementの構成は、VLDB以外のOracle ASMの構成と似ています。

Oracle ASMインスタンスへのメモリー割当てなど特定のパラメータ値では、値を高くする必要があります。

Oracle Databaseでは、Oracle ASMの変数割当て単位をサポートします。大きな変数割当て単位は、大きな順次I/O処理を使用する環境で利点があります。一般的にVLDB、特に大規模データ・ウェアハウスは、大きな割当て単位を活用するのに適した環境です。割当て単位は、1MBから64MBの範囲で2の累乗に設定できます(1、2、4、8、16、32および64)。大きな表をスキャンする問合せが多数ワークロードに含まれている場合は、大きなOracle ASM割当て単位を使用する必要があります。非常に大規模なデータ・ウェアハウス・システムでは64MBを使用してください。また、大きな割当て単位を使用すると、Oracle ASMのメモリー要件が減少し、Oracle ASMの起動時間が短縮されます。

関連項目:

Oracle ASMの設定および構成方法の詳細は、Oracle Automatic Storage Managementの管理者ガイドを参照してください。

用語集

自動データ最適化

自動データ最適化(ADO)は、データベース内の様々なストレージ階層の間でのデータの圧縮および移動を自動化する、情報ライフサイクル管理(ILM)のコンポーネントです。

コンポジット・パーティション化

コンポジット・パーティション化は、基本的なデータ配分方法の組合せです。データ配分方法で表をパーティション化した後、各パーティションは、同一または異なるデータ配分方法を使用して複数のサブパーティションに再分割されます。

並列度(DOP)

並列度(DOP)は、単一の処理に対応付けられるパラレル実行(PX)サーバーの数で表されます。

配分方法(パラレル実行)

配分方法とは、一方のパラレル実行(PX)サーバー・セットから他方へデータが送信または再配分される方法です。

ハッシュ・パーティション化

ハッシュ・パーティション化では、ユーザーが指定するパーティション化キーに適用されるハッシング・アルゴリズムに基づいて、データがパーティションにマッピングされます。

ヒート・マップ

ヒート・マップは、セグメントレベルでのデータ・アクセス、およびセグメントレベルおよび行レベルでのデータ変更を追跡する、情報ライフサイクル管理(ILM)のコンポーネントです。

情報ライフサイクル管理

情報ライフサイクル管理(ILM)は、使用できる期間中、データを管理するための一連のプロセスおよびポリシーです。

リスト・パーティション化

リスト・パーティション化では、各パーティションの説明にパーティション化キーの離散値のリストを指定することで、行をパーティションにマップします。

パラレル実行(PX)

パラレル実行(PX)とは、複数のプロセスの使用により、複数のCPUリソースおよびI/Oリソースを単一SQL文の実行に適用する機能です。

パラレル実行サーバー

パラレル実行(PX)サーバーとは、セッションを開始するかわりにパラレルで処理を実行する、個々のプロセスです。

並列処理,

パラレル化とは、1つのプロセスですべての処理を実行するのではなく、多数のプロセスで1つの問合せ内の処理の部分部分を

同時に実行するよう、タスクを細分化することです。

パーティション・プルーニング

パーティション・プルーニングは、表全体ではなくパーティションのサブセットにアクセスすることで問合せの結果を得られるときに起こります。

パーティション化

パーティション化とは、表および索引などのオブジェクトを、より小さく扱いやすい単位に分割するプロセスです。

パーティション化キー

パーティション化キーは、各行が格納されるパーティションを決定する1つ以上の列で構成されています。

問合せコーディネータ(QC)

問合せコーディネータ(QC)とは、パラレルSQL文を開始するセッションであり、パラレル実行(PX)コーディネータとも呼ばれます。

レンジ・パーティション

レンジ・パーティション化では、パーティションごとに指定されている、パーティション化キーの値範囲に基づいて、データがパーティションにマップされます。

大規模データベース(VLDB)

大規模データベースとは、非常に多くの行を含むか、非常に大きな物理記憶領域を使用するデータベースです。

索引

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X