

いまさら聞けない Oracle の基本 **上級編**

Oracle の基本、ついに完結！上級編パフォーマンスチューニングとバックアップ方法について解説いたします。実運用においてはトラブルがつきものですが、その場合の対策について解説しますので、ぜひお読みいただけますでしょうか。

また、データベースの勉強には、実際のデータベースを動かしながら学ぶのが一番の近道です。ぜひ、Oracle をインストールの上、本ドキュメントをお読みいただければと思います。Oracle のインストール方法については「初級編」をご覧ください。

1 Oracleのアーキテクチャ

1-1. Oracleのアーキテクチャ

本ドキュメントでは、以下の2つについて解説します。

- ・パフォーマンスチューニング

システムのパフォーマンスを向上（処理速度を改善し、レスポンス時間を短くする）ために、どのようにすればよいのかを解説します。

- ・バックアップ・リカバリ

データやデータベース全体をバックアップする方法、およびリカバリ（復旧）する方法を解説します。

いずれも難しそうな作業ですが、これらの作業は「ツボ」を抑えてしまえば、それほど難しくありません。そのツボとは、「Oracle のアーキテクチャ（内部構造と動作の仕組み）を知ること」です。車に例えれば、ハンドルとブレーキの使い方がわかるだけでも車を走らせることができますが、さらに内部のパーツや動作する仕組みを理解していれば、エンジントラブルで止まってしまった場合も、「どのパーツを調査すれば、修理できるのか?」「早く走るようにするためにはどのパーツを改造すればよいのか?」など、トラブル対策やチューニングもできるようになります。

Oracle も同様で、アーキテクチャを理解してしまえば、「SQL の実行パフォーマンスをはやくするために一番有効な方法は何なのか」「トランザクションを含めてバックアップをとるには、どのファイルをコピーすればよいのか?」等が理解できるようになりますので、まずは Oracle のアーキテクチャを理解しましょう。図 1-1 は Oracle のアーキテクチャ図になります。

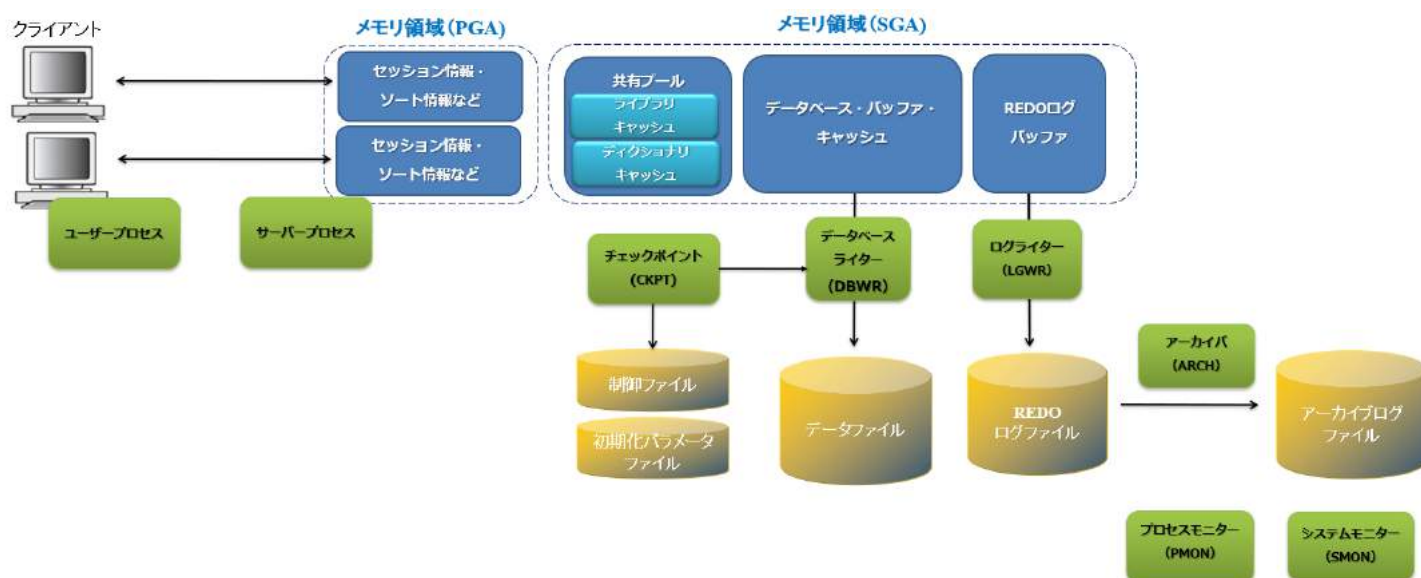


図 1-1. Oracle のアーキテクチャ

Oracle の構造は、クライアント、またはサーバー上で動作するプログラムである「プロセス」、プロセスが書き込み/読み取りを行う「ファイル」、および占有する「メモリ領域」から構成されます。これらの詳細を順に解説します。

ファイル

データを格納する「データファイル」や、データの更新履歴を記録するための「REDO ログファイル」や、その他、データベースの設定関連のファイルから構成されます。

ファイルの種類	役割
データファイル	<p>データを格納する最も重要なファイルです。データその他、インデックスなどのオブジェクト定義情報が格納されます。Oracle インストール時には、以下の複数のデータファイルが用意されています。</p> <ul style="list-style-type: none"> • SYSTEM 表領域…テーブル、ビュー、インデックスの定義情報や組み込みパッケージ、ユーザーが作成した <u>ストアドプロシージャ</u> のソースやコンパイル済みコードが格納される表領域です。 • USERS 領域…一般ユーザーが作成するテーブルやインデックスなどのオブジェクト情報、データを作成するための表領域です。 • TEMP 表領域…データのソート処理や、インデックスの作成をする場合に、Oracle が内部で一時的に作成するための表領域です。 • UNDO 表領域…データをロールバックするための UNDO 情報を格納する領域です。UNDO 領域がない場合は、他の領域が代わりに処理を行ないます。負荷がかかるため、通常は UNDO 表領域を用意しておきます。 <p>なお、新たに表領域を作成し、業務や用途毎にデータを分散して格納することもできます。なお、表領域とは物理上のデータファイルを Oracle で管理するための、論理上の概念となります。表領域とデータファイルは基本的には 1 対 1 ですが、データを分散するために、1 対多にすることも可能です。</p>
REDO ログファイル	<p>データベースに対する更新履歴を記録したファイルです。テーブルのデータの追加/変更/削除の情報や、テーブル、インデックスなどの作成/削除情報など、すべての変更処理を保持します。このファイルによって、変更のロールバックが可能となっています。</p>
アーカイブログファイル	<p>REDO ログファイルが一杯になった場合に、REDO ログファイルのコピーをアーカイブログファイルとして作成します。</p>
制御ファイル	<p>データベースを制御するためのファイルで、コントロールファイルとも呼ばれます。データベース名などの情報のほか、データ・ファイルや REDO ログファイルの場所など、データベースを管理するための情報が格納されます。</p>
初期化パラメータファイル	<p>コンフィグファイルです。Oracle の確保するメモリサイズ、言語の設定などの利用する環境に応じて、様々な設定を変更することができます。初期化パラメータファイルには pfile（パラメータ・ファイル）と spfile（サーバー・パラメーターファイル）の 2 種類の形式があります。pfile はテキスト形式のファイルでエディタで書き換えることができますが、再起動が必要です。spfile はバイナリ形式となっていますが、SQL にてデータベース可動中にも値の変更が可能となっています。</p>

表 1-1. ファイルの種類

プロセス

「プロセス」とは、クライアントまたはサーバーマシンに常駐するプログラムです。Oracle では複数のプロセスが並行で動作しています。通常のプロセスとしては、クライアントのシン上で動作するユーザープロセスと、サーバー上で動作するサーバープロセスがあります。この 2 つのプロセスの役割は、「クライアントとサーバー間のデータのやりとり」です。ユーザーから SQL を発行した場合に、ユーザープロセスが通信を行い、それを窓口となるサーバープロセスが受け付けます。サーバー側で処理後はサーバープロセスがその結果をユーザープロセスに返します。



1つのサーバープロセスが1つのユーザープロセスを処理する構成を「専用サーバー構成」、1つのサーバープロセスが複数のユーザープロセスを処理する構成を「共有サーバー構成」と呼びます。共有サーバー構成は、メモリの消費サイズを抑えるメリットがありますが、処理速度は低下します。最近のサーバーマシンでは、メモリサイズが多く搭載してきたため、専用サーバー構成を使用するのが一般的です。

また、これ以外にもサーバーマシン上で常駐する「バックグラウンドプロセス」と呼ばれるプロセスがあり、各ファイルへの書き込み実行や、システムの監視などを行ないます。主なバックグラウンドプロセスの種類は以下の通りです。

バックグラウンド プロセス名	役割
データベースライター (DBWR)	データベース・バッファ・キャッシュ内の変更されたデータを定期的にデータ・ファイルへ書き込みます。
ログライター (LGWR)	Redo ログ・バッファ・キャッシュ内のデータを下記のタイミングにて Redo ログ・ファイルに書き込みます。COMMIT 命令が発行された場合、または3分の2以上たまった場合に書き込まれます。
アーカイバ (ARCH)	Redo ログファイルが一杯になった場合、Redo ログファイルのコピーをアーカイブログファイルとして作成するためのプロセスです。このプロセスはデータベースを「アーカイブ・ログ・モード」に切り替えた場合に動作します。
チェックポイント (CKPT)	チェックポイントの処理を行うためのプロセスです。チェックポイントとは、REDO ログファイルに書き込まれた変更情報を、データファイルに反映させる作業です。チェックポイントが発生地は、このプロセスがデータファイルと制御ファイルのヘッダ情報を更新したり、DBWR に、データファイルへの書き込みを指示します。これにより、REDO ログファイル、制御ファイル、データファイル間の整合性を持たせることができます。チェックポイントの発生するタイミングは、REDO ログが一杯になった場合、管理者がチェックポイント命令を発行した場合、シャットダウン時など、様々なタイミングがあります。
プロセスモニター (PMON)	サーバープロセスを監視し、ユーザアプリケーションとのセッション間に障害が発生した場合、サーバープロセスの回復を行うためのプロセスです。
サーバーモニター (SMON)	サーバーを監視するためのプロセスです。以前にデータベースが正常終了していない場合、データベースを起動したタイミングで、自動的にデータベースの回復処理を行います。

表1-2.主なバックグラウンドプロセスの種類

それ以外にもメモリのサイズ変更を行うメモリー・マネージャー・プロセス(MMMON)や、分散データベースで使われるリカバラ(RECO)など、多くのバックグラウンドプロセスがあります。

メモリ領域

Oracle が確保するメモリ領域には、大きく PGA (プログラム・グローバル領域) と SGA (システム・グローバル領域) に分類されます。PGA は各サーバープロセスごとに作成される領域であり、セッションの情報の他、データのソート処理に使用されるソート領域からなります。

また、SGA 中身はデータベース・バッファ・REDO ログバッファ・共有プールと3つに分けられます。これらの主な目的は、「キャッシュ」となります。(キャッシュについては後述します)

領域名	役割
データベース バッファキャッシュ	データファイルから読み込んだデータブロックを メモリにキャッシュするための領域です。

REDO ログバッファ	REDO ログファイルへ書き出す情報を一時的に保存するメモリー領域です。
共有プール	表や定義情報をキャッシュする「ディクショナリキャッシュ」があります。「ライブラリキャッシュ」があります。その他、ユーザーのセッション情報を持ちます。

表 1－3.SGA の構成

1-2. アーキテクチャ内の動作

では、実際にこれらのファイル、プロセス、メモリが、ユーザーの要求をうけたときにアーキテクチャ内がどのように動作するかを見てみましょう。例として、SELECT 文や INSERT 文などの SQL が発行された場合の動作を解説します。

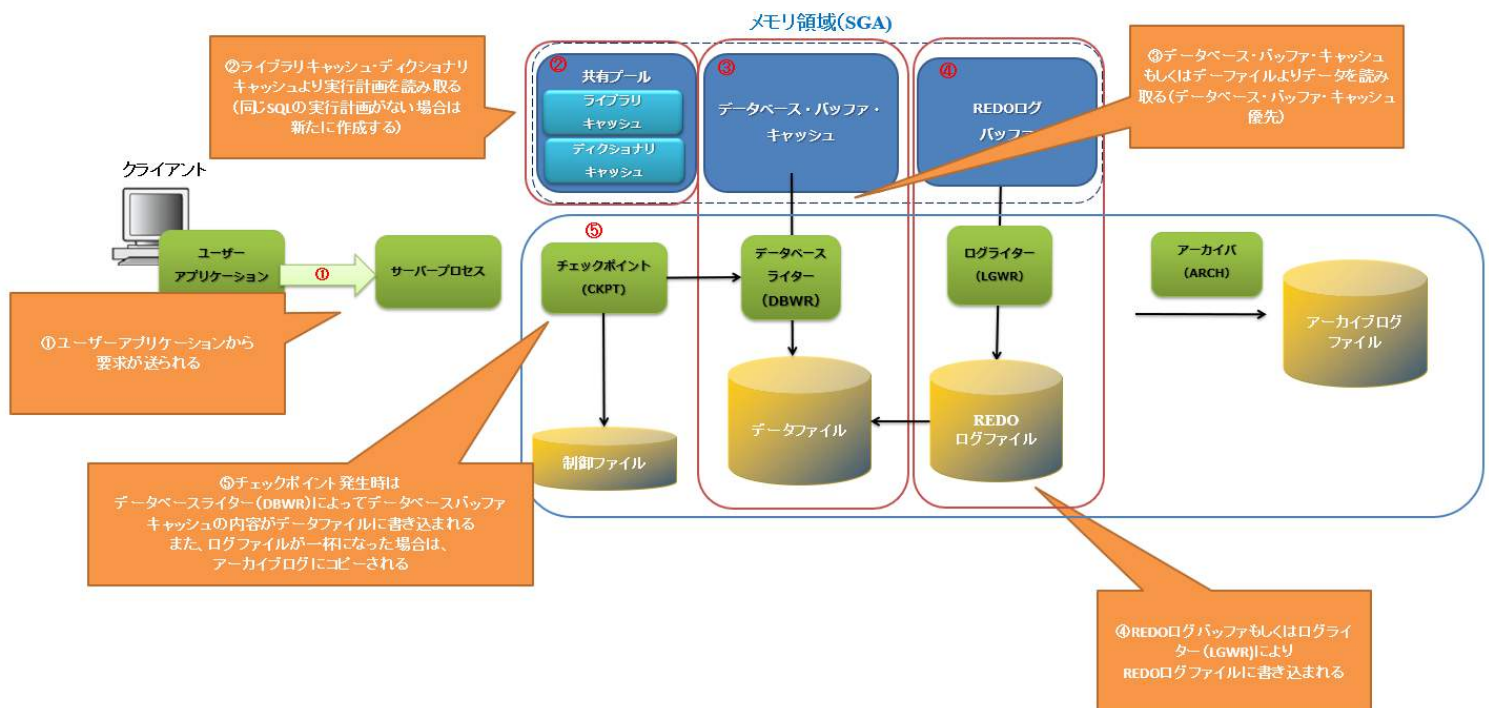


図 1－2. Oracle 内部の動作

① 要求

ユーザアプリケーションより、サーバープロセスへ SQL 文が送られます。

② 実行計画の作成

実行計画を作成します。実行計画については次回以降で詳しく解説しますが、簡単にいえば「SQL 文を処理するための、データの具体的な読み取り手順」です。例えば、対象のレコードをインデックスを使って検索するのか、それともインデックスを使わずに全レコードを読み取るのかといった判断や、複数のテーブルの結合した結果を帰す場合は、どの順序で結合するのかといった内容になります。データ量などの状況により、最も効率のよい方法が作成されますが、共有プール領域のライブラリキャッシュより、同じ SQL 文の解析結果があれば、その実行計画をそのまま採用します。ない場合は新たに解析を行ない、実行計画を作成します。実行計画の作成時は、共有フル領域のディクショナリキャッシュ、または SYSTEM 表領域内にあるディクショナリ情報から SQL で指定された表名や列名が実在するかチェックが行われます。SQL の構文に誤りがある場合や、表名・列名がない場合、テーブルに対してアクセス権限がない場合は、

エラーを返して終了し、問題ない場合はライブラリキャッシュおよびディクショナリキャッシュに登録が行われた後、次のステップに進みます。

③ データの読み取り

データベースバッファキャッシュ内にデータがあれば、そのデータを読み取ります。なければ、データファイルにアクセスしてデータを読み、データベースバッファキャッシュにコピーします。データを読み取り後は、SQL の種類により、以下のような動作をします。

SELECT 文の場合…もし ORDER BY 命令が付与されていた場合は、TEMP 表領域を使用して、並び替えの処理をソートが行われます。最後に、ユーザーアプリケーションにデータを返します。

INSERT/UPDATE/DELETE 場合…読み込んだデータを更新し、データベースバッファキャッシュに記録します。このとき、REDO ログバッファに更新情報が書き込まれます。さらに、3分の2以上になった場合、または COMMIT 命令が発行された場合は、ログライター (LGWR) プロセスにより、同時に REDO ログファイルの更新が行われます。最後にユーザー、アプリケーションに完了の通知を行いません。

データの入出力の動作において重要なポイントは「できるだけメモリ経由で処理が行われる」という点です。この仕組みは「キャッシュ」と呼ばれます。例えば、③では、ハードディスク上に格納されたデータファイルではなく、SGA 内のデータベースバッファキャッシュを優先して読み取ろうとしています。データをやり取りする際は、ハードディスクに格納されたデータファイルから直接書き込み/読み取りを行うのではなく、メモリを経由する方がアクセスが高速になります。そこで、Oracle ではできるだけメモリに実行計画やデータなどのコピーを保存しておき、次回以降のユーザーからあった SQL をメモリだけで処理しようとしています。②も同様に、ライブラリキャッシュ、ディクショナリキャッシュを使用して、解析の手間を省略しています。これにより、パフォーマンスを向上させています。

その他にも様々なファイル・プロセスがありますが、本ドキュメントの目的であるパフォーマンスチューニングとバックアップ・リカバリーを学ぶには、今回ご説明した内容が理解できれば十分です。この構造と動作を理解できれば、データが増加するにつれてパフォーマンスが急激に悪化した場合に、「データサイズがメモリのサイズを超えたために、キャッシュ機能が働かなかったのではないか？」バックアップが少し古い状態で復旧されてしまった場合は、「REDO ログファイルがうまく復旧されていないため？」などの推測ができるようになります。

実際にこれらの問題が起きた場合に、解決する手順については、次章以降で詳しく解説していきます。

2 ボトルネックの調査方法

2-1. パフォーマンスチューニングの流れ

第2章では、パフォーマンスチューニングの概要を解説します。パフォーマンスチューニングの基本的な流れは図1の通りとなります。

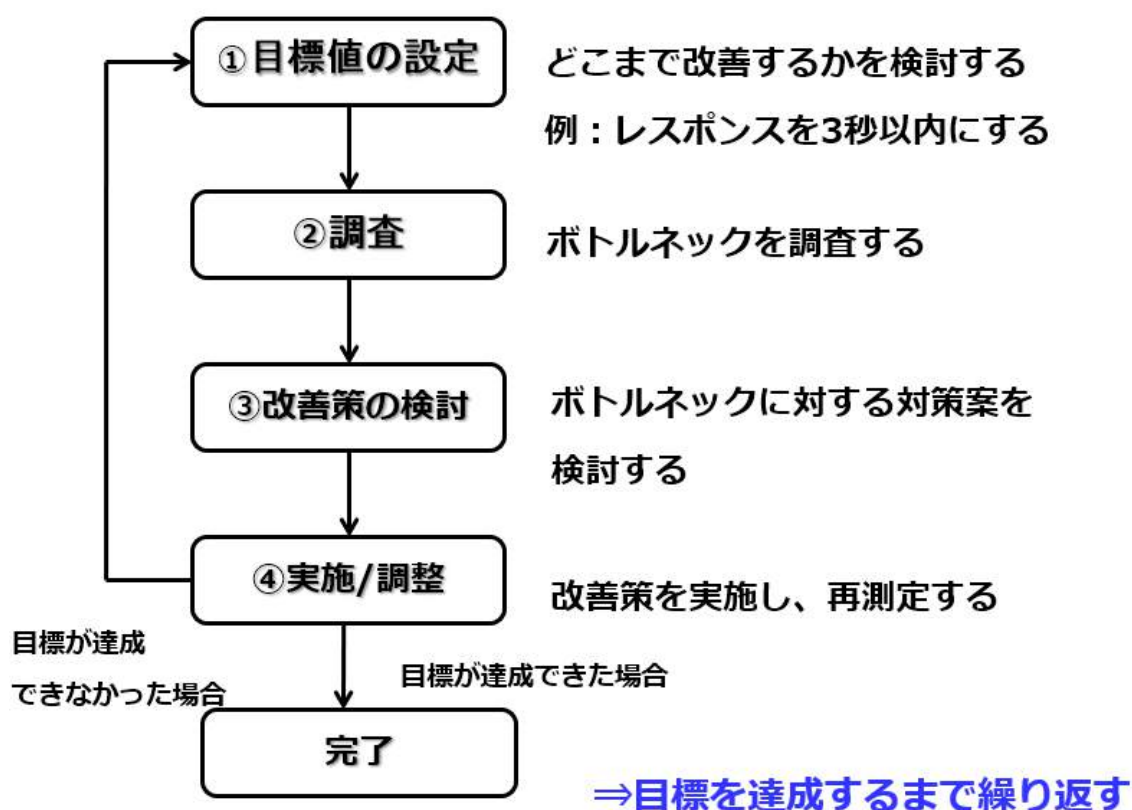


図2-1. パフォーマンスチューニングの流れ

パフォーマンスチューニングにおいては、まず目標値を決めることが大事です。パフォーマンスチューニングの方法は数多くあり、やりだすとキリがありません。Web アプリケーションの世界では「3秒ルール」という言葉があります。これは、ユーザーがボタンなどをクリックなどのイベントでサーバーリクエストをしてから、結果を返すまでの待ち時間を3秒以内にしなければならないという意味です（最近では1秒ルールが主流となっています）。このように開発システムに求められている要件に応じて、どこまで早くするかを最初に決めておけば、いくつもの対策を無駄に実施することがありません。次に、原因の調査を行ない、処理が遅くなっている根本の原因である「ボトルネック」を求めます。アプリケーションが効率の悪いループ処理を行っている、あるいはネットワーク速度に問題があるなど、様々な原因が考えられます。原因が判明した後は、それに対する改善策を検討します。例えばメモリーが足りないのであれば、メモリーサイズを増加することや、メモリーの消費量を少なくするなどの複数のアプローチがあります。これらから最も効果的かつ短時間で実施できるものを検討します。対策が決まったら、実際に実施し、再度パフォーマンスを測定します。目標値が達成できていれば完了となります。達成できなかった場合は、追加で改善策を実施し、目標値を達成するまで繰り返すという流れになります。

2-2. 待機イベントビュー

Oracle では、「待機イベントビュー」と呼ばれるビューがあります。待機イベントビューとは、Oracle 内部にある DBWnなどの各プロセスが行う処理が発行されてから終了するまでにかかった時間が自動で記録されるディクショナリビューです。これを見ればパフォーマンス悪化時の原因調査ができます。



ディクショナリビューとは、Oracleに標準で作成された、データベース内の表や、索引、ユーザー権限などデータベースに係わる様々な情報を表形式を管理したものです。ユーザーはSELECT文を発行することで情報を取得することが可能です。ただし、一部のディクショナリビューは権限がないと見れない場合がありますので、sysかsystemユーザーでログインの上、見るようにしてください。

「待機イベント」と呼ばれる、待ち時間の種類ごとに時間が記録されますので、待機イベントビューから時間のかかっている待機イベントを調べることで、パフォーマンス問題の原因となるボトルネックを調べる事が可能です。待機イベントビューの種類は以下のとおりです。

待機イベントビュー名	内容
V\$SYSTEM_EVENT	データベースを起動してから発生した待機イベントを表示
V\$SESSION_EVENT	現行セッションについての、待機イベントをすべて表示
V\$SESSION_WAIT	セッションが現在待機中、または待機を完了した直後の待機イベントを表示

表 2-1.待機イベントビューの種類

「V\$SYSTEM_EVENT」はデータベースを起動してから発生した待機イベントを表示します。通常はこの待機ビューを使用しますが、データベースの稼働時間が長い場合は、ボトルネックとなっている待機イベントがわからない場合があります。あるボタンを押したときにレスポンスがかえってこないなど、パフォーマンス問題をすぐに再現できる状況であれば、現象再現後に「V\$SESSION_EVENT」か「V\$SESSION_WAIT」を参照することで、直近の待機イベントを確認することができ、原因を正確に特定することができます。

代表的な待機イベント名と、考えられるその原因は以下のとおりです。

待機イベント名	発生するタイミング	このイベントの待機時間が長い場合に考えられる原因
db file scattered read	全表スキャンによる読み込みで発生する。	SQL の実行計画に問題がある場合が多い。
free buffer waits	ディスクからブロックを読み込む再、バッファ・キャッシュに空きがない場合に発生する。	データベースバッファキャッシュが小さすぎるか、SQL の実行計画に問題がある
library cache load lock	ライブラリキャッシュ内にオブジェクトをロードするために必要な待機。	ライブラリキャッシュミス（ライブラリキャッシュを検索して見つからないこと）が多い
library cache lock/ library cache pin	複数のプロセスがライブラリキャッシュに同時にアクセスしていることで発生する。	
log buffer space	Oracle の SGA に確保される ログバッファが不足した場合に発生する。	更新データ量が多いか、更新データの書き出しに時間がかかっている
log file parallel write/ log file single write	ログ・ライターの REDO ログファイルへの書き込みに関して発生する。	ログファイルへの書き込み負荷が高い可能性がある。

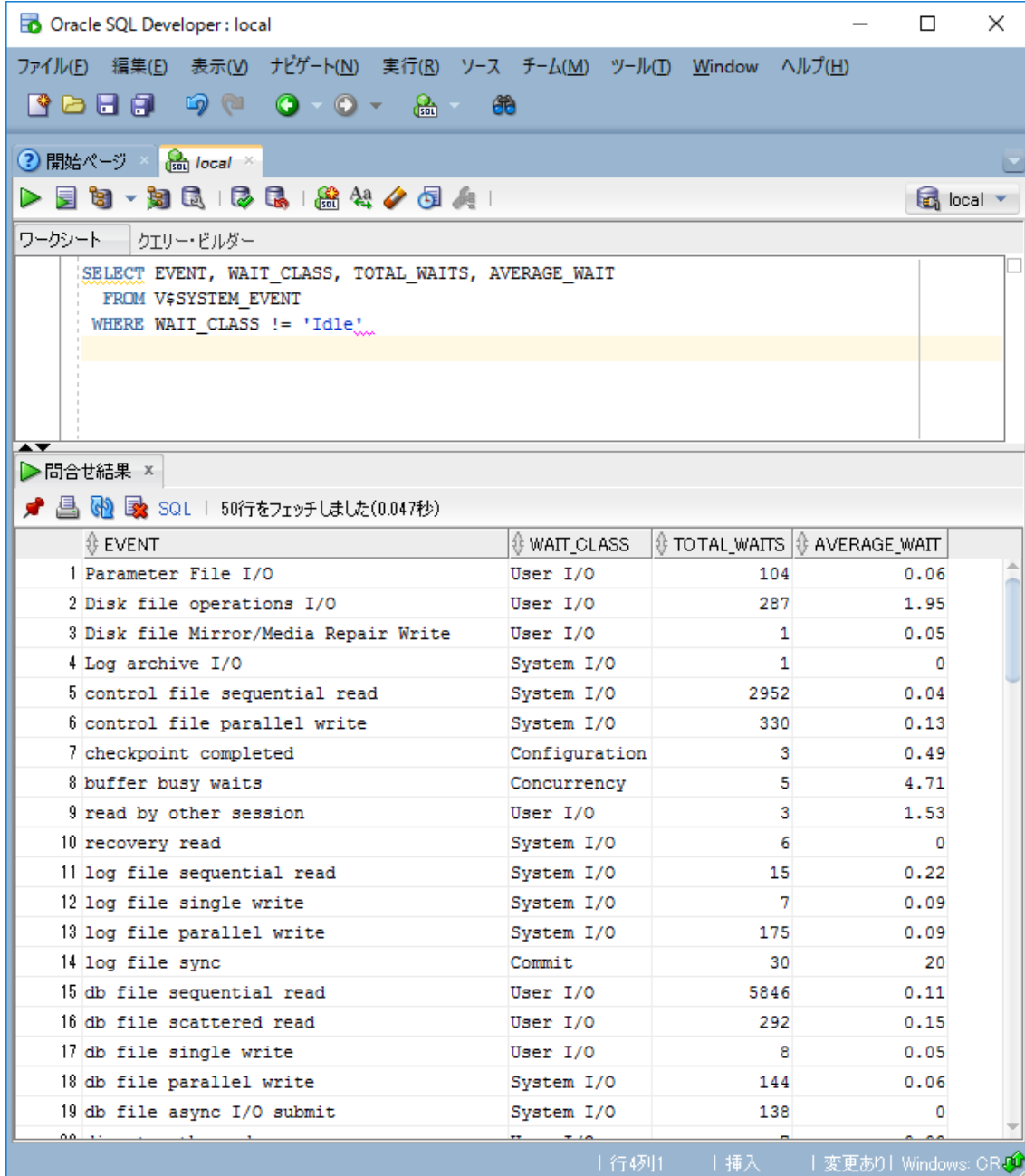
表 2-2.代表的な待機イベント

待機イベントを確認するための実際の SQL は以下のようになります。

```
SELECT EVENT, WAIT_CLASS, TOTAL_WAITS, AVERAGE_WAIT
FROM V$SYSTEM_EVENT
```


WHERE WAIT_CLASS != 'Idle'

Oracle 付属のツール「SQL Developer」での実行結果は以下の通りです。



The screenshot shows the Oracle SQL Developer interface. The query editor contains the following SQL statement:

```
SELECT EVENT, WAIT_CLASS, TOTAL_WAITS, AVERAGE_WAIT
FROM V$SYSTEM_EVENT
WHERE WAIT_CLASS != 'Idle'
```

The results window displays the following data:

EVENT	WAIT_CLASS	TOTAL_WAITS	AVERAGE_WAIT
1 Parameter File I/O	User I/O	104	0.06
2 Disk file operations I/O	User I/O	287	1.95
3 Disk file Mirror/Media Repair Write	User I/O	1	0.05
4 Log archive I/O	System I/O	1	0
5 control file sequential read	System I/O	2952	0.04
6 control file parallel write	System I/O	330	0.13
7 checkpoint completed	Configuration	3	0.49
8 buffer busy waits	Concurrency	5	4.71
9 read by other session	User I/O	3	1.53
10 recovery read	System I/O	6	0
11 log file sequential read	System I/O	15	0.22
12 log file single write	System I/O	7	0.09
13 log file parallel write	System I/O	175	0.09
14 log file sync	Commit	30	20
15 db file sequential read	User I/O	5846	0.11
16 db file scattered read	User I/O	292	0.15
17 db file single write	User I/O	8	0.05
18 db file parallel write	System I/O	144	0.06
19 db file async I/O submit	System I/O	138	0

画面 1-1.待機イベントの確認結果



SQLは「SQL*Plus」でも実行できますが、SQL Developerの方が実行結果が見やすいため、本ドキュメントでのSQL実行結果は、すべてSQL Developerを使うものとします。

V\$SYSTEM_EVENT ビューより、待機イベント名である「EVENT」、待機イベントのカテゴリである「WAIT_CLASS」、合計の待機時間となる「TOTAL_WAITS」、平均時間となる「AVERAGE_WAIT」

を参照します。V\$SESSION_EVENT ビューから調べる場合は、上記 SQL の FROM 句のみ変更してください。WHERE 条件では、「WAIT_CLASS」が「Idle」であるものを除外しています。WAIT_CLASS が「Idle」となっているものは、ユーザーが Oracle に何も要求を発行せず、Oracle がユーザーの要求を待っているアイドル時間に関連するイベントとなりますので、ボトルネックの原因とはなりません。

2-3. 実行計画

パフォーマンス問題の原因となるボトルネックがわかった後は、それに対する対策を実施します。ボトルネックは様々な種類がありますので、それによって対策も異なってきますが、実はパフォーマンス問題の6割～7割はSQLの「実行計画」が原因と言われています。実行計画とは、Oracle が内部で作成する「ユーザーが発行したSQLを処理するための、内部的なデータの読み取り手順」のことです。例えば、インデックスを使用する/しないの判断や、テーブルの結合順序になります。実行計画はOracleの実行計画生成エンジンである「オプティマイザ」によって自動で解析され作成されます。このとき、同じテーブルから同じ結果データをユーザーに返す場合であっても、実行計画内のインデックスの使用状況、内部の結合順序、結合方法が異なる場合は、内部で消費されるデータサイズが異なります。データサイズを大きく消費する実行計画の場合は、パフォーマンスが悪化してしまいます。パフォーマンスが悪化するかについては、「データベースバッファキャッシュ」が影響しています。データベースバッファキャッシュについては前回説明しましたが、ここでもう1度おさらいしましょう。図2がデータベースバッファキャッシュの動作になります。

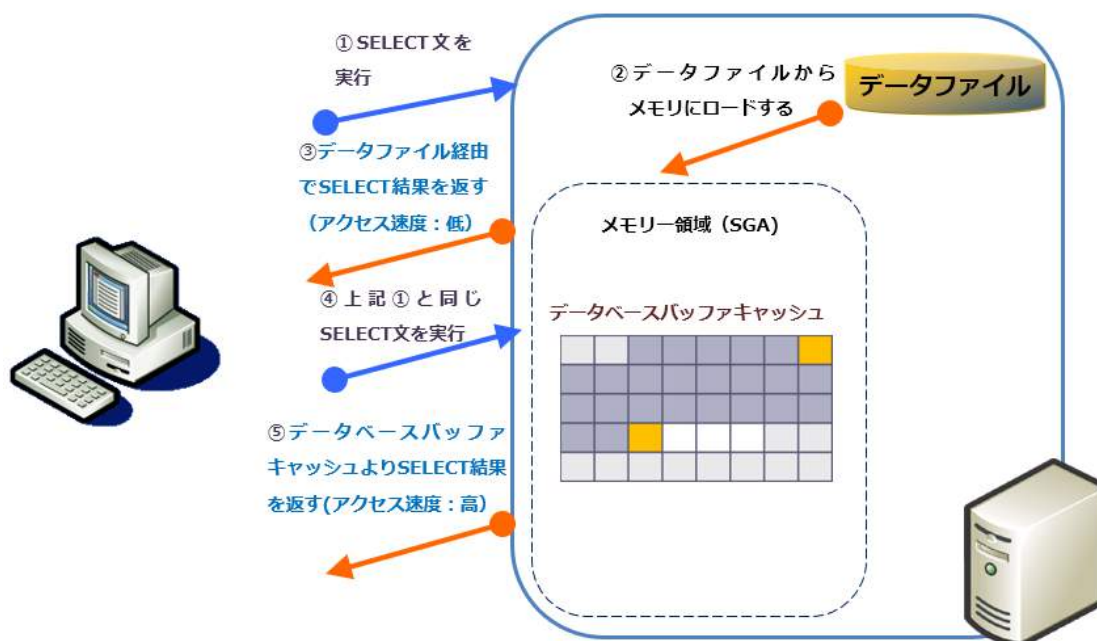


図2-2.データベースバッファキャッシュの動作

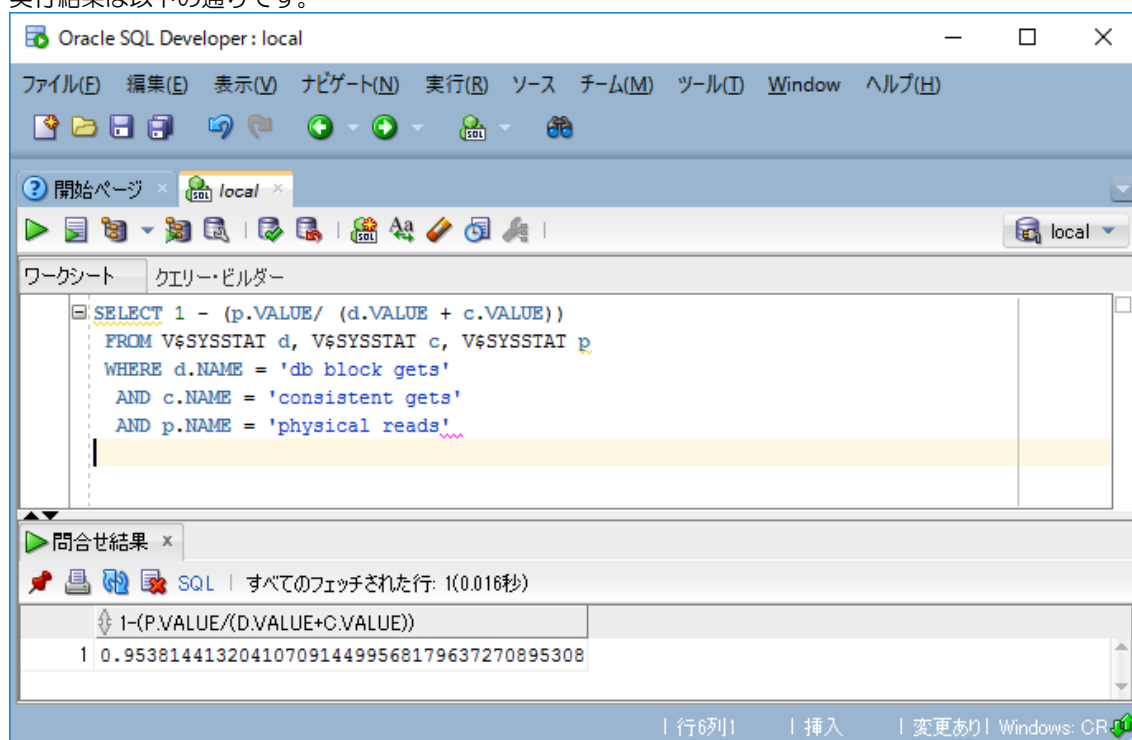
データベースバッファキャッシュは、メモリ領域である「SGA」の内部にあるSELECT結果などのデータのコピーを格納するための領域です。ユーザーからSQLが発行された場合は、Oracleは、まずデータベースバッファキャッシュからユーザーが発行したSQLの結果データがないかを検索し、ある場合は、データベースバッファキャッシュからデータを返します。もし、見つからない場合は、ハードディスク内にあるデータファイルからデータにアクセスしますが、この場合もデータベースバッファキャッシュにコピーを作成します。これにより、次にユーザーから全く同じSQLが発行された場合は、データベースバッファキャッシュからデータを返すことができます。データベースバッファキャッシュはメモリ領域ですので、ハードディスクであるデータファイルよりもアクセス速度が圧倒的に早く、ユーザーの要求を高速に処理することができます。しかし、実行計画が悪い場合は、データのサイズが大きくなってしまい、データベースバッファキャッシュのサイズを超えてしまう場合があります。このような場合は、データベースバッファキャッシュは使用されず、毎回データファイルへのアクセスとなります。これにより、急激に

パフォーマンスが悪化してしまいます。実行計画が悪い場合は、その他にもライブラリキャッシュが使用されない、REDO ログファイルの切り替えが頻繁に発生するなどの現象が発生することもあります。データの増加に連れて、処理が急激に遅くなったなどのケースにおいては、データベースバッファキャッシュの兼ね合いでパフォーマンス問題が起きていると考えて問題ないでしょう。

データベースバッファキャッシュが原因であるかどうかについては、待機ビューより、待機イベントが「free buffer waits」となっているレコードから待ち時間を確認することや、「バッファキャッシュヒット率」を調べることで調査することができます。バッファキャッシュヒット率とは、全体のアクセスのうち、データファイルとデータベースバッファキャッシュのそれぞれのアクセス数の割合となり、「 $1 - (\text{データファイルの読み込み回数} / \text{全体のアクセス回数})$ 」の式で求めることができます。SQL ですと、Oracle のシステム統計ビューである「V\$SYSSTAT」ビューに対して以下のような SELECT 文で求めることができます。

```
SELECT 1 - (p.VALUE / (d.VALUE + c.VALUE))
FROM V$SYSSTAT d, V$SYSSTAT c, V$SYSSTAT p
WHERE d.NAME = 'db block gets'
      AND c.NAME = 'consistent gets'
      AND p.NAME = 'physical reads'
/
```

実行結果は以下の通りです。



画面2-2. バッファキャッシュヒット率の確認結果

上記の例ではバッファキャッシュヒット率は95.3%となっています。バッファキャッシュヒット率は90%以上が理想とされています。もし、データベースバッファキャッシュに問題がある場合は、実行計画を調査します。そのためには、まず、処理が遅いと思われる実行計画を生成している「SQL」を特定する必要があります。これには、アプリケーション上でパフォーマンス問題が発生するタイミングから、アプリケーション内部で発行するSQLを抜き出します。例えば、商品の検索ボタンを押すと、フリーズしてしまうような現象が発生しているのであれば、商品検索用のSQLを抜き出します。SQLが特定できたら、実際にそのSQLの実行計画を調べます。

以下の手順で、実行計画を確認することが可能です。

① PLAN_TABLE の作成 (utlxplan スクリプトの実行)

実行計画を確認するために、まず、「PLAN_TABLE」と呼ばれるテーブルを作成する必要があります。
SYS ユーザーでログイン後、「<Oracle ホーム>%rdbsmx%admin」の下にあります「utlxplan.sql」
と呼ばれるスクリプトを実行します。



「Oracleホーム」とはOracleデータベースがインストールされる際に設定する、ベースとなるディレクトリパスのことです。

② 権限の付与

PUBLIC スキーマに対して、上記 PLAN_TABLE に対するアクセス権を付与します。

```
GRANT SELECT ON SYS.PLAN_TABLE TO PUBLIC  
/  
GRANT INSERT ON SYS.PLAN_TABLE TO PUBLIC  
/  
GRANT UPDATE ON SYS.PLAN_TABLE TO PUBLIC  
/  
GRANT DELETE ON SYS.PLAN_TABLE TO PUBLIC  
/
```

③ 以下の SQL にて、すべてのユーザーが使用できるように、PLAN_TABLE に対して、パブリックシノニムを作成します。パブリックシノニムとは、PUBLIC スキーマが所有するシノニムとなります。パブリックシノニムを作成することにより、全てのスキーマが参照できるテーブルとなります。

```
CREATE PUBLIC SYNONYM PLAN_TABLE FOR SYS.PLAN_TABLE;
```

上記①～③は実行計画を参照するための、準備作業となりますので、最初に一度実施するだけでかまいません。準備ができましたら、その後は、以下の④⑤の手順で、都度、実行計画を確認することが可能です。

③ 実行計画の生成

実行計画を生成するには「EXPLAIN PLAN」と呼ばれる SQL を入力します。

EXPLAIN PLAN 文は以下のような形式となります。

```
EXPLAIN PLAN FOR <SQL 文>;
```

<SQL 文>は実際に測定したい SQL です。ボトルネックと思われる SQL を入力します。EXPLAIN PLAN 文を実行する場合は、実行計画の生成のみとなりますので、実際に入力した SQL は実行されません。

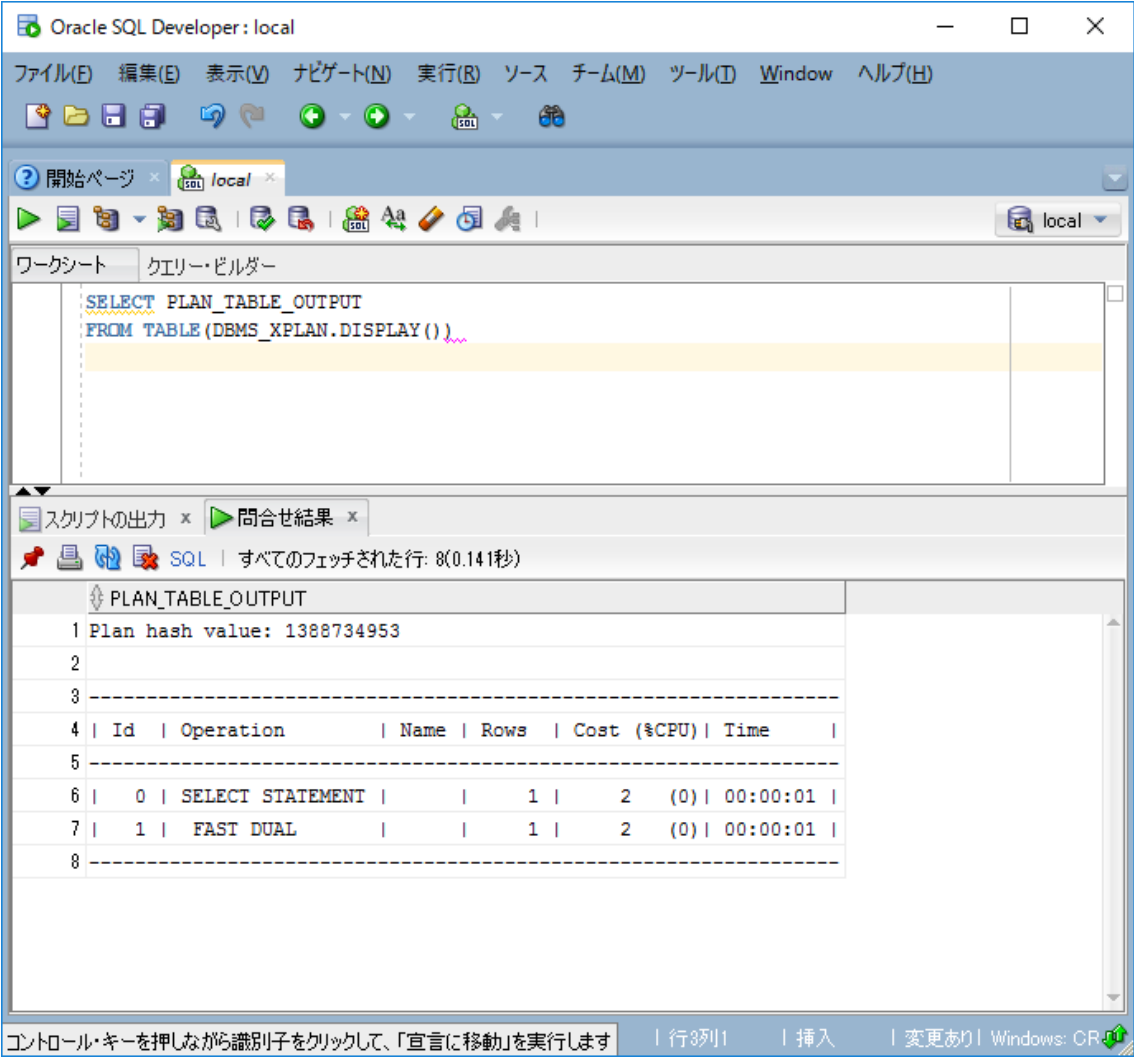
④ 実行計画の表示

先ほど作成した実行計画を参照します。これには、PLAN_TABLE に対して SQL 文を発行すること

でも可能ですが、以下の SQL 文で、PLAN_TABLE の内容を整形してみやすい結果を表示することができます。

```
SELECT PLAN_TABLE_OUTPUT
FROM TABLE(DBMS_XPLAN.DISPLAY())
/
```

実行結果は以下の通りです。



画面 2-3. PLAN_TABLE の確認結果

上記 SQL を実行すると、実行計画の内容が表示されます。複数のテーブルにアクセスしているような場合は、個々のテーブルの操作ごとにツリー構造で表示されます。表示される項目とその意味は以下のとおりです。

項目名	内容
Operation	結合の種類や全表スキャン、インデックスなどの操作内容が表示されます。
Rows	操作によって返されるレコード数が表示されます。
Bytes	操作によって返されるサイズがバイト単位で表示されます。

Cost	パフォーマンスの指標となる「コスト」が表示されます。Oracle によって見積られた論理値ですが、この値が大きいほど、処理が重いという目安になります。
Time	実際に計測された時間が表示されます。

表 2-3.PLAN_TABLE の項目名

「Time」は実際にかかった時間です。これにより、ボトルネックとなっている SQL かどうかわかります。ボトルネックとなっている場合は、操作内容である「Operation」を確認します。インデックスが使用されている場合は、「INDEX RANGE SCAN」「INDEX UNIQUE SCAN」などになりますが、全表スキャンの場合は「TABLE ACCESS FULL」と表示されます。また、「Rows」「Bytes」は、実際に返されるレコード数とそのデータサイズとなります。ID が「O」となっているレコードに記載のバイトが最終的に返される結果データのサイズですが、結合途中のデータも含めてメモリーが消費されることになります。全表スキャンの場合は、レコード数に応じてサイズが大きくなります。これらの内容で、ボトルネックとなっている SQL であるかを確認し、もし実行計画に問題がある場合は、改善し、メモリーの消費サイズを抑えることで、バッファキャッシュヒット率を向上することができます。

3 SQLチューニング

3-1. 統計情報の更新

この章からはパフォーマンスチューニング方法を説明します。第2章では実行計画が原因となることを説明しましたが、実行計画を改善するにはSQL文を書き直すことでできます。SQL文を修正する方法は、副問い合わせや、結合するテーブル数を減らすなどのロジックの修正の他、ヒント文を使用する方法もあります。

もし、Oracleのバージョンが9i以前の場合は、SQLチューニングの前にあらかじめ「統計情報」を更新する必要があります。「統計情報」とは、テーブルごとのレコード数やデータの分布状況が記録された情報です。データの分布状況とは、データの値がどのくらい重複し、どのくらいバラついているかの割合を示した情報のことです。実行計画を作成する際には、「オブティマイザ」と呼ばれるOracle内部のエンジンによってSQL文が解析されますが、この際、より最適な実行計画を作るために統計情報が利用されます。例えば、「テーブルのデータを読み取る際にインデックスを使うのか、全レコードをスキャンするのか」という判断は、SQL文の他にも、統計情報の中のテーブルレコード数の値によって決定されます。Oracle9iの場合は統計情報が最新の状態より古い場合があります。もし、統計情報が古い場合は、SQL文を適切に修正しても最適な実行計画を作成することができないことになりますので、SQLをチューニングする前に、必ず最新にしておきましょう。



Oracle10g以上の場合は、毎日の22:00～8:00の間に自動で収集されるようになっているため、統計情報を手動で実行する必要はありません。また、Oracle9i以前の場合、オブティマイザの動作として、「ルールベース」「コストベース」という2つがありますが、ルールベースとなっている場合は統計情報を更新する必要はありません。ルールベースの場合は、「WHERE句に使用されている列がインデックスの対象列に含まれているか」などの、一定のルールをもとにして実行計画を作成し、統計情報は参照しないためです。ただし、一般的には統計情報を使用するコストベースの方がパフォーマンスの良い実行計画を作成するため、ルールベースにすることはおすすめできません。なお、Oracle10g以上の場合は、コストベースのみとなり、ルールベースは廃止されています。

統計情報を修正するには、以下のANALYZE文と呼ばれるSQL文を発行します。

ANALYZE TABLE テーブル名 COMPUTE STATISTICS;

「テーブル名」には、改善したいSQLが参照する対象テーブルを指定して実行します。これにより、指定したテーブルの全レコードの情報を元に完全な行数の統計に更新することができますが、レコード数が極端に多い場合は、時間がかかることがあります。その場合は、上位のレコード数または比率を指定して、部分的なレコードの情報を元に統計情報を更新することもできます。レコード数を指定して更新する場合は、以下の形式で記述します。

ANALYZE TABLE テーブル名 ESTIMATE STATISTICS SAMPLE 数値 {ROWS|PERCENT};

「ESTIMATE STATISTICS SAMPLE」に続いて、レコードの行数または比率のいずれかを指定します。レコード数を指定したい場合は数値の後に「ROWS」、比率を指定したい場合は数値の後に「PERCENT」と記述することで可能です。

3-2. SQLチューニング

統計情報が最新に更新されたら、実際に SQL のチューニングを行っていきましょう。SQL のチューニングの方法は大きく、「実行処理の時間を改善する」「解析時間を改善する」の2つに分かれます。以下に、これらの代表的な方法をご紹介します。

・実行処理の時間を改善する方法

範囲検索では BETWEEN を使用する

例えば「商品の価格が 1000 円から 5000 円まで」といった範囲指定検索をおこないたい場合、WHERE 句にて「<=」「>=」などの演算子と AND 条件を使用する方法と、BETWEEN 句を使用する方法があります。この場合、BETWEEN 句を使用したほうが処理速度は向上します。

【例】SELECT 商品名 FROM 商品マスタ WHERE 価格 >= 1000 AND 価格 <= 5000;

↓

SELECT 商品名 FROM 商品マスタ WHERE 価格 BETWEEN 1000 AND 5000;

AND 条件で指定した場合、内部ではテーブルの各レコードの「価格」の列が 1000 円以上であることと、5000 円以下であることの2回の判定が別々に行われます。BETWEEN を使用した場合は、「価格」列が 1000 円から 5000 円の範囲であるかが 1 度の判定で行われるため、処理速度が向上します。

IN 句よりも EXIST 句を使用する

SELECT 文の WHERE 句において、別の SELECT 文を使用するテクニックとして「サブクエリ」があります。サブクエリを使用する場合、WHERE 句の接続詞として IN 句と EXIST 句を使用することができますが、この場合、EXIST 句を使用したほうが処理速度は向上します。

【例】SELECT 顧客名 FROM 売上

WHERE 顧客商品コード IN (SELECT 商品コード
FROM 商品マスタ
WHERE 価格 >= 10000)

/

↓

SELECT 顧客名 FROM 売上

WHERE EXISTS (SELECT 商品コード
FROM 商品マスタ
WHERE 価格 >= 10000 AND 商品コード = 顧客商品コード)

/

IN 句を使用する場合は、サブクエリ内の SELECT 文を実行し、その結果を一時テーブルに保存した上で、呼び出し元の SELECT 文が処理されます。EXIST を使用した場合は、一時テーブルを作成されずに処理が行うことができ、その分、処理速度が向上します。ただし、EXIST の場合は、SQL の見た目は IN 句の方が分かりやすいため、実行速度を優先したい場合にこのような修正を検討してください。

・解析時間を改善する方法

アスタリスク(*)は列名に変更する

「SELECT * FROM テーブル名」というように、SELECT 句をアスタリスク(*)で指定した場合、テーブルのすべての列を取得することができますが、すべて列名で指定したほうが解析速度は向上します。例えば、商品コード、商品名、価格の3つの列からなる商品マスタのテーブルを SELECT する場合は、

以下のように、すべての列を明示的に指定します。

【例】 `SELECT * FROM 商品マスタ;`



`SELECT 商品コード, 商品名, 価格 FROM 商品マスタ;`

このとき、求められる結果は同じですが、アスタリスクを使用した場合、オプティマイザが SQL を解析する際に SQL のテーブルの各列を調べて展開する処理が余分に必要となります。テーブルの列名を指定することで、このような無駄な処理が不要となり、解析速度が向上します。同様に、「`SELECT COUNT(*) FROM テーブル名`」でテーブルのレコード件数を調べる際も「`SELECT COUNT(列名) FROM テーブル名`」と修正したほうが解析速度は向上します。



ただし、`COUNT(列名)`と指定した場合は、指定した列がNULL値であるレコードをのぞいた件数となりますので、主キー項目に使用している列などのNULL値のない列を指定する必要があります。

列名の指定にはテーブル名を指定する

複数のテーブルを結合する SQL の場合、列名の箇所は必ず「テーブル名.列名」としたほうが解析速度は向上します。

【例】 `SELECT 商品名 FROM 商品マスタ, 売上 WHERE 商品マスタ.商品コード = 売上.商品コード AND 売上日 >= '2010/04/01';`



【例】 `SELECT 商品名 FROM 商品マスタ, 売上 WHERE 商品マスタ.商品コード = 売上.商品コード AND 売上.売上日 >= '2010/04/01';`

上記の修正前の例では、WHERE 句の「売上日」にテーブル名の指定がありません。もし、「売上日」という列が「商品マスタ」か「売上」テーブルのどちらかのテーブルにしかない列であれば、修正前の SQL でもエラーにはなりませんが、内部では、すべてのテーブルにある列情報を調べる処理が必要となります。明示的にテーブル名を指定しておけば、そのような必要がなくなるため、解析の時間が向上します。もし、FROM 句にて「商品マスタ S, 売上 U」などのようにエイリアスをつけているのであれば、「U.売上日」のように「エイリアス.売上日」の形式で指定しても構いません。



3-3. バインド変数

バインド変数とは、SQL 文内で、実行時に値を割り当てる際に使用する変数です。バインド変数を用いると、「ライブラリキャッシュ」が活用できるようになるので、解析時間を短縮できるメリットがあります。ライブラリキャッシュとは、Oracle の共有プールと呼ばれるメモリー領域にある、実行計画のコピーを作成しておくための領域です。ユーザーが同じ SQL を何度も実行するケースにおいて、オプティマイザが都度 SQL を解析し、同じ実行計画を再作成することは無駄な作業です。そこで Oracle では、1 度作成した実行計画はライブラリキャッシュに保存しておき、ユーザーが再度同じ SQL を発行した場合は、ライブラリキャッシュから前回の実行計画を読み出すようになっています。ライブラリキャッシュを使用することによって、SQL を都度解析する必要がなくなるため、解析時間を大きく向上させることができます。ただし、SQL が少しでも異なる場合、ライブラリキャッシュは利用されません。例えば、以下の 2 つの SQL 文を例にとってみましょう。

- ① `SELECT EMP FROM SCOTT.EMP WHERE EMPNO = 1`
- ② `SELECT EMP FROM SCOTT.EMP WHERE EMPNO = 2`

WHERE 句の右辺で使われる数値や文字列値のことを「リテラル値」と呼ばれます。この2つの SQL はリテラル値のみが異なっています。ユーザーが順に①②の順に発行した際、リテラル値のみが異なっている別の SQL とみなされてしまうため、ライブラリキャッシュは使用されません。



リテラル値の他、大文字小文字やスペースも含めてまったく同じ SQL でないと、ライブラリキャッシュの実行計画は採用されません。

このようにリテラル値のみが異なる SQL の場合は、「バインド変数」を使うことによって、ライブラリキャッシュを使用できるように改善することができます。バインド変数を使用した SQL は以下ようになります。

```
SELECT EMP FROM SCOTT.EMP WHERE EMPNO = :EMPNO
```

WHERE 句の右辺が「:EMPNO」に変わっていますが、このようにコロンで始まる名前がバインド変数となります。バインド変数を使用した場合、実際のリテラル値についてはオプティマイザが解析処理を行った後に割り当てようになります。実行する際のリテラル値が異なっても、オプティマイザが解析する SQL 文としては同一となるため、ライブラリキャッシュに保存された実行計画が使用されるようになります。

以上で代表的な SQL チューニングのテクニックをご紹介しました。今回ご紹介した方法は一例となり、その他にも様々なテクニックがありますので、実際に SQL チューニングを行われる際は、専門書などもあわせてお読みいただければと思います。

4 その他のチューニング

4-1. SQL以外のチューニング

第3章でご紹介した通り、SQL文を修正することで実行計画および速度を改善することができますが、SQLチューニングは場合によってはテーブル構造を見直す必要があり、影響範囲が広くなるというデメリットがあります。例えば、調査の結果SQLに問題があるところまで特定できても、テーブル定義を修正しないといけない場合は稼働中のシステムで修正することは難しいでしょう。また、ネットワーク速度など、実行計画以外が原因で遅くなっている場合もあります。そこで、本章ではSQL以外のチューニング方法をご紹介します。ひとつはデータベースオブジェクトを追加する方法です。データベースオブジェクトとは、データベースの拡張機能のことです。（詳しくは中級編でご紹介しています。）データベースオブジェクトを使えば、実アプリケーションで発行するSQLを修正せずに、データベース側で修正することができるため、比較的修正がしやすいというメリットがあります。データベースオブジェクトにはいくつかの種類がありますが、中でもパフォーマンスチューニングに役立つ「インデックス」「マテリアライズド・ビュー」「ストアドプログラム」をご紹介します。また、その他の方法として「初期化パラメータの修正」「パーティショニング」の方法もご紹介します。

4-2. インデックス

インデックスとは、データの検索速度を向上させるために、どの行がどの領域に格納されているかを示した索引用の領域のことです。データを検索する際、目的のデータが見つかるまですべての行を一行ずつ調べていくよりも、インデックスを調べて、目的のレコードを見つけてからその行のデータを読み取るように変更することで、アクセス速度を向上することができます。また、実行計画内でインデックスが使用された場合は、データベースバッファキャッシュに格納されるサイズが、実データから索引領域のみに変更されるため、消費サイズが大きく抑えられます。さらに、インデックスは実装が簡単というメリットもあります。反面、テーブル更新時にインデックスの再構築処理がはしるため、更新のパフォーマンスが遅くなるというデメリットがあります。

インデックスを作成するには、まず、ボトルネックとなっているSQL文を確認します。実行計画中でインデックスが使用されるか、されないかについては、SQLで参照されているテーブルにインデックスが作成されていること、インデックスの対象列で決まります。基本的には、SQL文で使用されている、「WHERE句」に指定している列に対してインデックスを作成しているかどうかで、使用される可動化が決まります。

例えば、以下のようなSQLに対してインデックスを作成する場合は「カテゴリコード」「在庫数」を対象列としたインデックスを作成します。

```
SELECT 商品コード, 商品名 FROM 商品マスタ
WHERE  カテゴリコード = 1  AND 在庫数 > 100
/
```

インデックスを作成するSQL文は以下の通りです。

```
CREATE INDEX IDX_商品マスタ ON 商品マスタ("カテゴリコード", "在庫数");
```



また、SELECT句で使用される列もインデックス列に含めると、インデックス対象のテーブルにアクセスする必要がなくなり、さらにアクセス速度が高速となります。他のSQL文との汎用性を考えて問題ない場合はSELECT句も対象列に含めることも検討してください。

ただし、上記のルールに従って作成した場合も、実行計画でそのインデックスが必ず使われるとは限りません。極端にレコード数が少ない場合は、全表スキャンの方が早い場合もあります。Oracle ではレコード数なども考慮して実行計画を作成します。



このように実際のレコード数なども考慮してインデックスを使用する/しないなどを決める方法は「コストベース」と呼ばれます。Oracleではその他にも、SQL文とインデックス列のみで判定する「ルールベース」がありましたが、Oracle10g以降は、ルールベースは廃止され、コストベースのみとなっています。。

また、WHERE 条件の列数が多い場合は、すべての列を対象列とする必要はありません。あまりに多いとインデックス領域のサイズも大きくなり、結果的に使用されない場合もあります。このように適切なインデックスをつくることは一筋縄ではいかない部分もありますので、インデックス作成後は、再度実行計画を確認し、インデックス使用されるようになったかを確認してください。



4-3. マテリアライズド・ビュー

「マテリアライズド・ビュー」は同じくデータベースオブジェクトの1つである「ビュー」に機能を拡張したデータベースオブジェクトです。「マテリアライズド・ビュー」を理解するために、まず「ビュー」についておさらいしてみましょう。ビューとは、テーブルのデータを自由に加工した上でデータを表示するデータベースオブジェクトです。例えば、以下のような顧客マスタと商品マスタを結合した SELECT 文があるとします。

```
SELECT 顧客マスタ.顧客名, 商品マスタ.商品名, 商品マスタ.税抜価格 * 1.05 税込価格
FROM   顧客マスタ, 商品マスタ
WHERE  顧客マスタ.商品コード = 商品マスタ.商品コード
/
```

この SELECT 文では、商品情報が格納された商品マスタと、商品購入客の情報が格納された顧客マスタを結合しそれぞれのテーブルが持つ情報を取得しています。また、商品の価格に関しては、税抜価格の列を 1.05 倍し、税込価格に変換した結果を表示しています（列名も「税込価格」に変更しています）。このように SELECT 文では複数テーブルの結合や、列名を変更した上で結果を表示することができますが、このような SQL を何度も実行したい場合、毎回 SQL を記述するのは手間になります。このような場合に役立つのがビューです。ビューを作成する SQL は以下のとおりです。

```
CREATE VIEW V_顧客購入商品
SELECT 顧客マスタ.顧客名, 商品マスタ.商品名, 商品マスタ.税抜価格 * 1.05 税込価格
FROM   顧客マスタ, 商品マスタ
WHERE  顧客マスタ.商品コード = 商品マスタ.商品コード
/
```

ビューを作成するには、まず「CREATE VIEW」に続けてビュー名を記述します。ここではビュー名を「V_顧客購入商品」としました。その後は、このビューの対象の SELECT 文を記述します。今回は、先ほどと同じ SELECT 文を記述しています。SELECT 文を記述できた後は、この SQL を実行することで、ビューが作成することができます。ビューが作成された後は、「SELECT * FROM V_顧客購入商品ビュー」というビューに対する SELECT 文を発行することで、さきほどの SELECT 文と同様の結合データを取得することができますようになります。

このように一度ビューを作成してしまえば、どのような複雑な加工をした SELECT 文でも、簡単に求めたいデータが求められるようになるというメリットや、テーブル内から不要な列やレコードを隠すこともでき、セキュリティを向上できるというメリットもあります。このように、「ビュー」では、テーブルからデータを加工して求めるためのデータベースオブジェクトですが、そのデータについてはビューが保持しているわけではありません。ユーザーが「SELECT * FROM 顧客購入商品ビュー」というビューに対して SELECT 文が発行するたびに、商品マスタ、顧客マスタのテーブル結合を行い、求めた結果をユーザーに返すことになります。この、ユーザーが SQL を発行してから結果を受け取るまでの一連の処理の中で、結合にかかる時間はかなり比重の高いものです。大規模 EC サイトにおいては、膨大な商品と顧客を扱いますが、例えば、2万点の商品を持つ商品マスタと10万人の顧客をもつ顧客マスタを結合した場合、結合されるデータは2万×10万=20億レコードのボリュームとなります。このような結合データをビュー経由で求めると、結合処理にかなり時間がかかってしまうことになります。このような場合に役に立つのが「マテリアライズド・ビュー」です。マテリアライズド・ビューは、ビューの機能に実データを保持する機能を加えたデータベースオブジェクトです。複数のテーブルを結合した SELECT 部を元にマテリアライズド・ビューを作成した場合は、すでに結合されたデータが内部で保持されますので、結合にかかる時間を省略することができ、SQL 全体のパフォーマンスを大きく向上することができます。さきほどのビューと同じ SELECT 文を、今度はマテリアライズド・ビューで作成してみましょう。マテリアライズド・ビューを作成する SQL は以下のとおりです。

```
CREATE MATERIALIZED VIEW MV_顧客購入商品
REFRESH COMPLETE
START WITH SYSDATE NEXT SYDATE + 7
AS
SELECT 顧客マスタ.顧客名, 商品マスタ.商品名, 商品マスタ.税抜価格 * 1.05 税込価格
FROM 顧客マスタ, 商品マスタ
WHERE 顧客マスタ.商品コード = 商品マスタ.商品コード
/
```

「CREATE MATERIALIZED VIEW」に引き続き マテリアライズド・ビュー名を指定します。ここでは「MV_顧客購入商品」としました。次に「REFRESH」に続けて、マテリアライズドビューのデータを更新する方法を指定します。マテリアライズド・ビューの参照元テーブルのデータが更新された場合、マテリアライズド・ビューが保持するデータと不整合が生じてしまうことになるため、同期を取る必要があります。この、参照元テーブルとマテリアライズド・ビューの同期をとる作業のことを「リフレッシュ」と呼びます。リフレッシュにはいくつかの方法がありますので、「マテリアライズド・ビュー」の作成時はその方法を指定します。選択できるリフレッシュ方法は表1の通りです。

方法	詳細
COMPLETE（完全）	いったんマテリアライズド・ビュー上の全てのデータを削除の上、再度作り直します。
FAST（高速）	前回から差異のあるデータのみをリフレッシュします。関連リフレッシュよりも高速にリフレッシュが可能ですが、高速リフレッシュを使用する場合は、あらかじめ「マテリアライズド・ビュー・ログ」と呼ばれるデータベースオブジェクトを作成している必要があります（詳しくは後述で説明します）。
FORCE（強制）	高速リフレッシュが可能であれば高速リフレッシュ、高速リフレッシュが不可能であれば完全リフレッシュを実行します。マテリアライズド・ビュー作成時にリフレッシュ方法を指定しなかった場合は、強制となります。

表4-1. マテリアライズド・ビューのリフレッシュ方法の種類

また、このリフレッシュに関しては、自動で行うことも、手動で行うことも可能です。リフレッシュを自動で行う場合は、あわせてリフレッシュのタイミングを指定します。リフレッシュのタイミングは以下の

3種類となります。

タイミング	詳細
ON COMMIT (コミット時)	参照元のテーブルがコミットされたタイミングで自動リフレッシュを行ないます。コミット時のリフレッシュについては、リフレッシュ方法が高速リフレッシュの場合のみ、選択可能です。
START WITH / NEXT (定期間隔)	指定した時刻で自動リフレッシュを行ないます。定期感覚で行う場合は、START WITH に続けて初回の更新日時、NEXT 句に続けて次回更新日時を、それぞれ DATE 型で指定します。2 回目以降のリフレッシュは、START WITH と NEXT の間隔ごとに実行されます。
ON DEMAND (手動)	自動リフレッシュを行わず、手動でリフレッシュを行ないます。手動で行う場合は、DBMS_MVIEW. REFRESH(マテリアライズド・ビュー名)という SQL でリフレッシュを行うことができます。

表4-2. マテリアライズド・ビューのリフレッシュ タイミングの種類

今回作成するマテリアライズド・ビューでは、リフレッシュ方法を「完全」、リフレッシュのタイミングを「定期間隔」としています。もし、定期間隔で行う場合は、START WITH 句に続けて初回の更新日時、NEXT 句に続けて次回更新日時をそれぞれ日付型で指定します。今回の SQL では START WITH 句の後に「SYSDATE」となっていますが、「SYSDATE」は現在の日付を求める Oracle 標準の関数です。START WITH 句で SYSDATE を指定した場合、マテリアライズド・ビューの作成と同時にリフレッシュが行われることになります。また、NEXT 句に続けては、「SYDATE + 7」と記載されていますが、これは現在日付より 7 日後ということになります。「AS」の後はビューと同様に、対象の SELECT 文を記載いたします。

なお、リフレッシュ方法に「高速」を使う場合は、「完全」の場合よりも高速に更新することができますが、「高速」を使用したい場合は別途、「マテリアライズド・ビュー・ログ」というデータベースオブジェクトを作成する必要があります。「マテリアライズド・ビュー・ログ」はマテリアライズド・ビュー専用で使用する、テーブルの変更情報を記録したログデータです。マテリアライズド・ビュー・ログの作成により、マテリアライズド・ビュー内でテーブルの変更履歴が把握できるようになり、参照元テーブルとのデータ同期が差分のみの更新で可能となるため、高速にリフレッシュが使用できるようになります。マテリアライズド・ビュー・ログを作成する場合は、「CREATE MATERIALIZED VIEW LOG」に続けてマテリアライズド・ビューが参照する参照元テーブルを指定します。今回は2つのテーブルを結合していますので、以下の2つの SQL となります。

CREATE MATERIALIZED VIEW LOG ON 顧客マスタ;

CREATE MATERIALIZED VIEW LOG ON 商品マスタ;

あらかじめ上記の SQL でマテリアライズド・ビュー・ログを作成している場合はマテリアライズド・ビューのリフレッシュ方法で「高速」の指定が可能となります。

もし、アプリケーションで複数テーブルを結合したビューを使用しており、その結合処理に大きく時間がかかっている場合は、マテリアライズド・ビューに変更することで、大きくパフォーマンスを改善することができます。しかし、マテリアライズド・ビューでは参照元テーブルとは別にデータを保持するため、参照元テーブルとのデータの同期がとられるまでは古いデータが保持されていることになります。例えば在庫の引き当てのトランザクションをマテリアライズド・ビュー経由で処理した場合、古い在庫数が取得されるために、アプリケーションの不具合が生じる可能性があります。もし、データに一貫性が求められる場合は、リフレッシュ方法を「高速」、リフレッシュのタイミングを「コミット時」にしてマテリアライズド・ビューを作成することで、コミットと同時にリフレッシュが行われますので、トランザクション中のデータの不整合問題を防止することができます。ただしこの場合は、テーブルをコミットすることに、マテリアライズド・ビューの同期処理が発生するため、更新時のパフォーマンスが悪化するというデメリットがあります。このようにマテリアライズド・ビューはビューと比較してすべての点に優れているわけではなく、使いどころをよく考える必要があります。もし、データの更新よりも表示のパフォーマンスを改善したい場合は、ぜひマテリアライズド・ビューの利用を検討するとよいでしょう。

4-4. ストアドプログラム

パフォーマンスチューニング問題の多くは Oracle のメモリー領域が使用されず、ディスクアクセスが多発することが多く原因だと説明しましたが、それ以外にもネットワークの転送速度が遅いために、アプリケーションの画面表示までに時間がかかっている場合もあります。データベースと連携したアプリケーションでは、通常アプリケーションのソースコード内に、SQL の処理を記述しますが、この方法ではアプリケーションサーバーとデータベースサーバー間のネットワークを介して SQL の命令と結果データの送受信が行われることになります。アプリケーションサーバーとデータベースサーバーが遠隔にあるような環境や、繰り返し処理により何度も SQL のやりとりが行われる場合は、ネットワークを何度も行き来することになり、パフォーマンス低下を招いてしまいます。このようにネットワークの処理がボトルネックとなっている場合、アプリケーションから呼び出している SQL を、「ストアドプログラム」に変更することにより改善することが可能です。中級編でもご紹介しましたが、ストアドプログラムはデータベース内に蓄積（ストア）し、実行するデータベースオブジェクトです。アプリケーション側で発行しているデータベースの更新処理をストアドプログラム化することにより、ネットワークの行き来が減るため、パフォーマンスの向上が可能です。

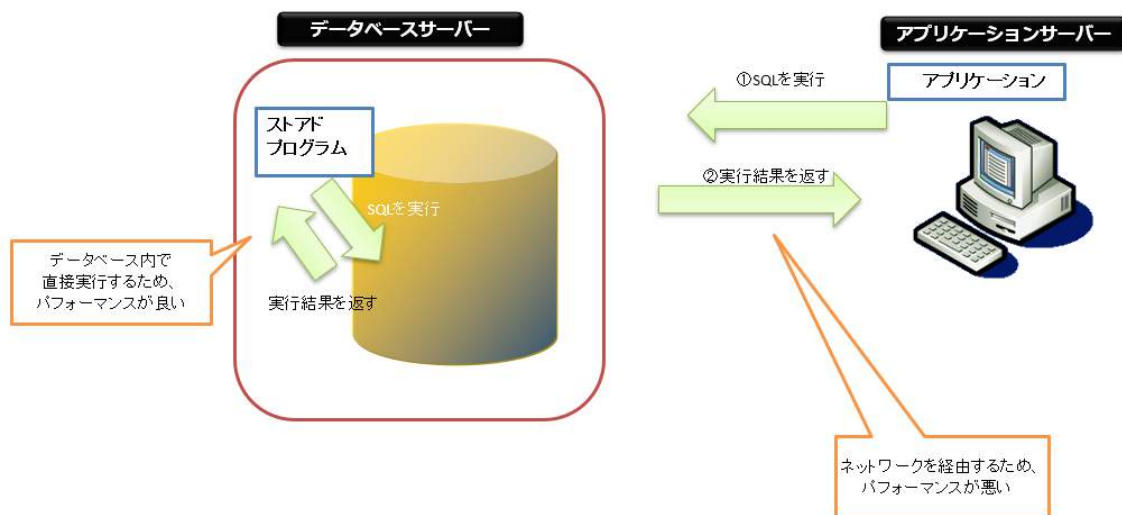


図4-1. ストアドプログラム

ストアドプログラムには、「ストアドプロシージャ」と「ストアドファンクション」があります。この2つの違いは、ストアドプロシージャが処理を実行するだけであるのに対して、ストアドファンクションは戻り値として結果を返す点です。また、Oracle には複数のストアドプロシージャ、ストアドファンクションひとまとめにした「パッケージ」というオブジェクトがあります。パッケージには、仕様部である「パッケージヘッダ」と、「パッケージ本体」の2つの部分から構成されています。パッケージヘッダには実際に複数のプログラムの仕様やパッケージ共通で使用する変数を宣言し、パッケージ本体で実際のプログラムを定義します。パッケージでは関連する処理をまとめてモジュール化して管理できるため、プログラムを分かりやすくできる他、一度パッケージを呼び出すと、パッケージ全体がメモリーにロードされるため、その後のアクセスが早くなるなどのメリットがあります。

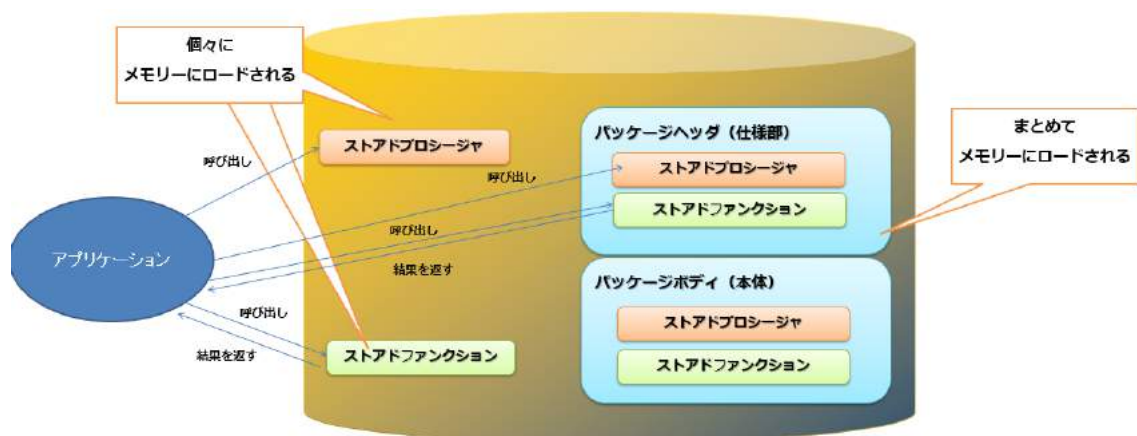


図4-2.ストアドプロシージャ、ストアドファンクション、パッケージの違い

ストアドプログラムは、「CREATE PROCEDURE」または「CREATE FUNCTION」で始まる SQL 文を実行することで作成することができますが、ストアドプログラムを作成する SQL においては、他のデータベースオブジェクトの場合と異なり、「PL/SQL (Programming Language for SQL)」という言語を使用します。PL/SQL は、通常の SQL を拡張し、変数や IF 文、繰り返し命令、例外処理などのプログラムに必要な処理を記述可能としたストアドプログラム専用のプログラミング言語です。以下のコードは、サンプルのストアドファンクションを作成する PL/SQL です。

```
CREATE FUNCTION SAMPLE_FUNCTION(P_DEPARTMENT_ID NUMBER, P_GRADE FLOAT)
RETURN NUMBER
IS
/***** 宣言部 *****/
-- 変数の宣言
RET NUMBER;

-- カーソルの宣言
CURSOR EMPLOYEES_CUR IS
SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY
FROM HR.EMPLOYEES
WHERE DEPARTMENT_ID = P_DEPARTMENT_ID;

-- カーソル変数の宣言
EMPLOYEES_REC EMPLOYEES_CUR%ROWTYPE;

/***** 処理部 *****/
BEGIN
RET := 0;

--引数が設定されていない場合は終了
IF P_DEPARTMENT_ID IS NULL OR P_GRADE IS NULL THEN
RETURN -1;
END IF;
```

```

FOR EMPLOYEES_REC IN EMPLOYEES_CUR LOOP
    --画面に表示する
    DBMS_OUTPUT.PUT_LINE('    NAME:'    ||    EMPLOYEES_REC.FIRST_NAME    ||
EMPLOYEES_REC.LAST_NAME ||
                        '    SALARY:'    ||    EMPLOYEES_REC.SALARY    ||    ' ⇒ '    ||
EMPLOYEES_REC.SALARY * P_GRADE);

    --給料を更新する
    UPDATE HR.EMPLOYEES
    SET    SALARY = SALARY * P_GRADE
    WHERE  EMPLOYEE_ID = EMPLOYEES_REC.EMPLOYEE_ID;

    --更新したレコード件数をカウント
    RET := RET + SQL%ROWCOUNT;
END LOOP;

COMMIT; --コミットする

RETURN RET;

/***** 例外処理部 *****/
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK; --ロールバックする
        DBMS_OUTPUT.PUT_LINE('ERROR!!');
        RETURN -1;
END;

```

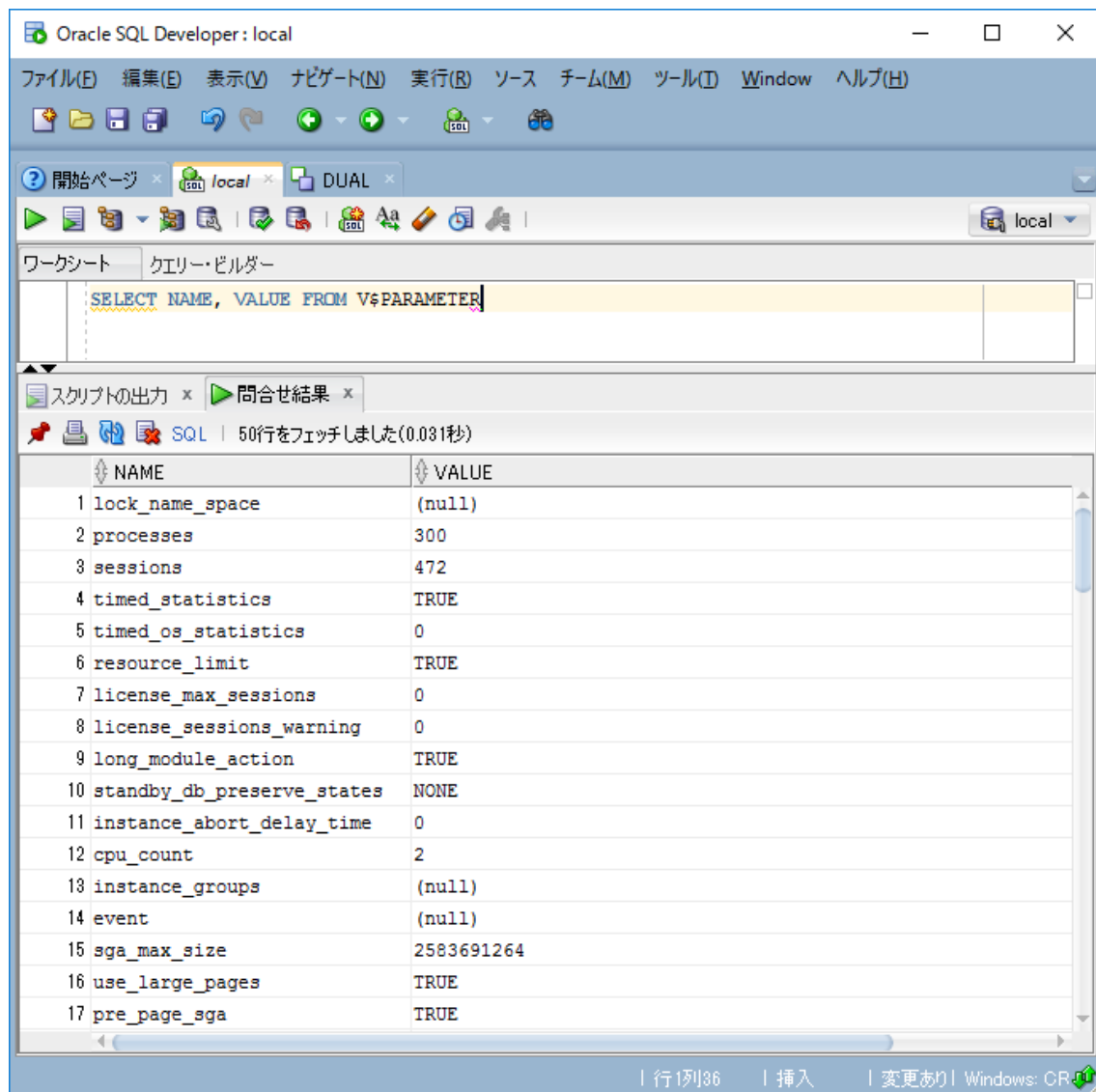
このサンプルは、「特定の部署に所属する社員の給料をアップ」というストアドファンクションです。PL/SQL のコードに関しては、大きく「定義部」、「宣言部」、「処理部」、「例外処理部」の4つのブロックに分かれます。詳細やストアドプログラムの実行方法については中級編をご覧ください。



4-5. メモリーサイズの増加

パフォーマンスチューニング問題の多くは Oracle のメモリー領域が使用されず、ディスクアクセスが多発さきほどの SQL チューニングによって、インデックスを使用できるように改善を行った場合は、メモリーのアクセス率である「バッファキャッシュヒット率」を向上することができます。しかし、レコード数が極端に大きすぎる場合などは、インデックスを使用するように改善しても、メモリー容量に収まりきらず、バッファキャッシュヒット率が向上しない場合もあります。このような場合は、メモリーサイズ自体を大きくすることで、バッファキャッシュのヒット率を向上することができます。メモリーサイズを変更するには、Oracle の設定項目である「初期化パラメータ」を変更します。初期化パラメータとは、メモリーのサイズやログファイルの場所など、データベースごとに個々に設定できるオプション項目のことです。以下の SQL にて、各初期化パラメータの名前と、現在の設定値を確認することができます。

```
SELECT NAME, VALUE FROM V$PARAMETER;
```



画面4-1.初期化パラメータの表示

メモリーサイズ関連の初期化パラメータ名は以下の通りです。

キャッシュ名	Oracle8i	Oracle9i	Oracle10g	Oracle11g 以降
バッファ キャッシュ	DB_BLOCK_ BUFFERS	DB_CACHE_SIZE	SGA_TARGET	MEMORY_TARGE T
REDO ログバッファ	LOG_BUFFER	LOG_BUFFER		

共有プール	SHARED_POOL_SIZE	SHARED_POOL_SIZE		
PGA	SORT_AREA_SIZE HASH_AREA_SIZE	PGA_AGGREGATE_TARGET	PGA_AGGREGATE_TARGET	

表4-3.メモリーサイズ関連の初期化パラメータ名

初期化パラメータ名は Oracle のバージョンにより異なるので注意が必要です。例えば、10g では、「SGA_TARGET」で SGA 項目であるデータベースバッファキャッシュ、REDO ログバッファ、共有プールの合計サイズをまとめて指定することが可能となっています。（Oracle9i 以前と同様に、DB_CACHE_SIZE など個別に指定することも可能です。）また、11g 以降では、SGA と PGA を「MEMORY_TARGET」でまとめて指定することが可能となっています。まとめて指定した場合は使用状況により、個々のサイズが動的に調整されるようになります。

初期化パラメータの値を変更したい場合は以下の SQL で可能です。

`ALTER SYSTEM SET 初期化パラメータ名 = サイズ;`

初期化パラメータ名には、表4-3のいずれかの初期化パラメータ名を指定し、サイズには設定変更後のサイズを指定します。

なお、バッファキャッシュヒット率を最大にするための最適なメモリーサイズはいくらか？といったような明確な基準はありません。サイズを実際に変更した後、データベースバッファキャッシュヒット率を確認しながら、最適なサイズを調べる必要があります。しかし、Oracle9i 以上では、

「V\$DB_CACHE_ADVICE」というビューを利用すると、ある程度自動化することができます。以下の SQL を実行してください。

```
SELECT ROWNUM, SIZE_FOR_ESTIMATE, ESTD_PHYSICAL_READ_FACTOR
FROM V$DB_CACHE_ADVICE
/
```

	ROWNUM	SIZE_FOR_ESTIMATE	ESTD_PHYSICAL_READ_FACTOR
1	1	128	73.9565
2	2	256	40.5155
3	3	384	18.4362
4	4	512	7.9203
5	5	640	4.3865
6	6	768	2.9786
7	7	896	2.2387
8	8	1024	1.8472
9	9	1152	1.5506
10	10	1280	1.1176
11	11	1328	1
12	12	1408	0.8925
13	13	1536	0.8345
14	14	1664	0.8002
15	15	1792	0.7912
16	16	1920	0.7893
17	17	2048	0.7893
18	18	2176	0.7893
19	19	2304	0.7893
20	20	2432	0.7893
21	21	2560	0.7893

画面4-2.V\$DB_CACHE_ADVICE ビューの表示結果

V\$DB_CACHE_ADVICE ビューの表示した結果が画面4-2です。この結果では、現在のバッファキャッシュサイズの値を10～200%に変化させた場合の計測値が表示されています。「ROWNUM」は行番号、「SIZE_FOR_ESTIMATE」がバッファキャッシュサイズとなります。また、「ESTD_PHYSICAL_READ_FACTOR」がバッファキャッシュサイズに対する物理ハードディスクの読み込み係数となり、低いほどバッファキャッシュヒット率が上がっていることとなります。ただし、この結

果は、サンプルデータを元にした予測値のため、この結果に従ってバッファキャッシュサイズを変更しても、必ず効果が出るとは限りません。参考程度に利用するとよいでしょう。

x4-6. パーティショニング

パフォーマンスチューニング問題の多くは Oracle のメモリー領域が使用されず、ディスクアクセスが多発ここまでは、SQL を処理する際に、データベースバッファキャッシュやライブラリキャッシュなどのメモリーアクセスを活用し、物理ディスクへのアクセスを減らすことによってパフォーマンスを向上する方法を主に説明しました。しかし、これらの対策を施しても目標値が達成できない場合や、メモリーサイズが極端に小さいため実施できない場合もあります。このような場合は、ディスクアクセスのパフォーマンスを向上する方法を検討します。ディスクアクセスのパフォーマンスを向上する方法として「パーティショニング」があります。パーティショニングとはテーブルのレコードを「パーティション」と呼ばれる単位に分割して格納する方法のことです。パーティショニングを使わない場合、1つのテーブルデータに格納される物理ファイルは1つとなりますが、SELECT 文や UPDATE 文などによってテーブルの全データを読み書きするような場合にアクセスが集中し、処理のボトルネックとなってしまいます。このような場合「パーティション」を複数のハードディスクに分散して格納することで、全レコードを読み書きする場合でも、並行で読み書き処理が行なえるため、処理時間を大きく軽減することができます。



パーティションを単一のハードディスクに格納した場合でも、WHERE条件付きのSQLによって部分的なレコードの読み書きが行われる場合は、テーブル内にある一部のパーティションのみのスキャンで済むようになるため、読み取り速度は軽減します。



図4-3.パーティショニングのイメージ

図4-3にある「表領域」とは、テーブルと、そのデータが格納されるデータファイルを対応づけるためのデータベースオブジェクトです。標準では、Oracle の管理テーブル用の「SYSTEM 表領域」、ユーザーが作成したテーブル用の「USERS 表領域」などが用意されています。これらの表領域は、物理上は Oracle のインストールディレクトリ内にある「SYSTEM01.DBF」「USERS01.DBF」という名前のデータファイルでそれぞれ対応付けられています。テーブルのデータの格納先の物理ファイルは、テーブルの作成時の CREATE TABLE 文で指定する「表領域」によって決定されます。



テーブルを作成する際に表領域を指定しなかった場合は、ログインしているスキーマで設定している「デフォルト表領域」が自動で設定されます。

パーティショニングを使用する場合は、まず、表領域を作成した上で、テーブル作成時に表領域を対応付けるという2段階のステップで行ないます。

表領域を作成する SQL の例は以下のとおりです。

```
CREATE TABLESPACE USERS2
DATAFILE 'C:\APP\USER\ORADATA\SID\USERS02.DBF' SIZE 500M
/
```

CREATE TABLESPACE の跡に続けて表領域名を指定します。この例では「USERS2」としています。表領域名には、表領域名とかぶらないユニークな名前を指定します。今回は「USERS2」としています。続けて「DATAFILE」と記述し、その後にデータファイル名をフルパスで指定します。ディスクが複数ある場合は、別々のディスクに分散します。この SQL の例も、Oracle 標準のデータファイルが格納されているパスの例としていますが、実際は、パス内の「USER」は OS 上のユーザー名、「SID」は作成したデータベースのインスタンス名となりますのでご注意ください。拡張子は任意ですが、こちらも標準のデータファイルとあわせて「DBF」にしておくといよいでしょう。最後に「SIZE」に続いて、サイズを指定します。サイズの指定では、数値の後に「M」をつけることでメガバイト単位、「G」をつけることでギガバイト単位の指定が可能です。個々のシステムに応じて、必要なサイズを指定してください。表領域を作成後は、テーブル作成時の CREATE TABLE 文でパーティショニング設定を行ないます。

パーティショニング設定では、分散するためのキーとなるテーブルの列と、振り分け方法として、レンジパーティション・リストパーティション・ハッシュパーティションのいずれかを選択します。

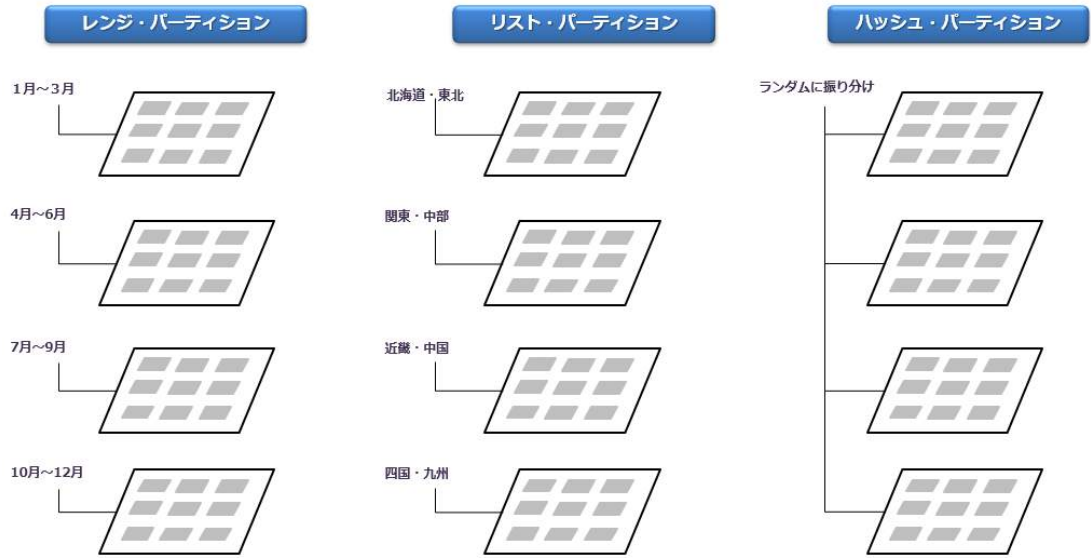


図4-4. パーティションの種類

パーティション種別	説明
レンジパーティション	データの範囲で表領域を指定する
リストパーティション	個々のデータごとに表領域を指定する
ハッシュパーティション	ランダムに表領域を振り分ける

表4-2. パーティションの種類

例えば、「会計期間」といった期間によって範囲指定ができる列などの場合は、レンジパーティションを使用し、範囲ごとに振り分け先を指定することができます。また、「地域」のようなデータの値が何種類かに限られている場合は、リストパーティションを使用し、各値によって振り分け先を指定することができます。範囲や値によって振り分け先を指定せずに、ランダムに振り分けたい場合はハッシュパーティションを使用します。



なお、パーティショニングは、テーブルの作成時にしか設定できません。既存のテーブルにパーティショニングを使用したい場合は、テーブルを再作成する必要があります。

レンジパーティションを作成する SQL は以下のとおりです。

```
CREATE TABLE "商品マスタ"
```

```
(
  "商品コード"          NUMBER,
  "商品名"              VARCHAR2(20),
  "価格"                NUMBER,
  "サイズ"              VARCHAR2(2),
  "更新日"              DATE
)
PARTITION BY RANGE("更新日")
(
  PARTITION PARTITION1 VALUES LESS THAN (TO_DATE('2010/03/31', 'YYYY/MM/DD'))
  TABLESPACE USERS,
  PARTITION PARTITION2 VALUES LESS THAN (TO_DATE('2010/06/30', 'YYYY/MM/DD'))
  TABLESPACE USERS2,
  PARTITION PARTITION3 VALUES LESS THAN (TO_DATE('2010/09/30', 'YYYY/MM/DD'))
  TABLESPACE USERS3,
  PARTITION PARTITION4 VALUES LESS THAN (TO_DATE('2010/12/31', 'YYYY/MM/DD'))
  TABLESPACE USERS4
)
/
```

CREATE TABLE 文の記述に続けて、「PARTITION BY RANGE」と記述し、キーとなる列名を括弧つきで指定します。さらに、振り分けの指定として「パーティション」の設定を行ないます。パーティションとは、パーティション名と、値の範囲、格納する表領域をひとまとめにしたものです。レンジパーティションの場合は、PARTITION の後に任意のパーティション名を記述し、「VALUES LESS THAN」の後にそのパーティションの上限値となる値を指定します。もし、上限値を設定しない場合は「MAXVALUE」と指定します。今回の例はキーが日付型の列ですので、TO_DATE 関数を使用して値の範囲を指定しています。最後に、「TABLESPACE」の後に格納先の表領域を指定する形となります。このパーティションの設定を分散したい数だけ繰り返し記述します。今回の例では、この例では「更新日」の列をキー列として、3 ヶ月ごとに USERS～USERS4 の表領域に指定しています。

リストパーティションを作成する SQL は以下のとおりです。

```
CREATE TABLE "地域マスタ"
```

```
(
  "地域コード"          NUMBER,
  "地域名"              VARCHAR2(6)
)
```

```

)
PARTITION BY LIST("地域名")
(
    PARTITION PARTITION1 VALUES ('北海道', '東北') TABLESPACE USERS,
    PARTITION PARTITION2 VALUES ('関東', '中部') TABLESPACE USERS2,
    PARTITION PARTITION3 VALUES ('近畿', '中国') TABLESPACE USERS3,
    PARTITION PARTITION4 VALUES ('四国', '九州') TABLESPACE USERS4
)
/

```

リストパーティションの場合は、CREATE TABLE 文の記述に続けて、「PARTITION BY LIST」と記述し、「VALUES」の後は各パーティションで格納する「値」をカンマ区切りで指定します。その他はレンジパーティションと同様です。この例では、「地域名」列をキーとし、その文字列の値によって、USERS～USERS4 の表領域に振り分けています。

ハッシュパーティションを作成する SQL は以下のとおりです。

```

CREATE TABLE "商品マスタ"
(
    "商品コード"          NUMBER,
    "商品名"              VARCHAR2(20),
    "価格"                NUMBER,
    "サイズ"              VARCHAR2(2),
    "更新日"              DATE
)
PARTITION BY HASH("商品コード")
(
    PARTITION PARTITION1 TABLESPACE USERS,
    PARTITION PARTITION2 TABLESPACE USERS2,
    PARTITION PARTITION3 TABLESPACE USERS3,
    PARTITION PARTITION4 TABLESPACE USERS4
)
/

```

ハッシュパーティションの場合は、CREATE TABLE 文の記述に続けて、「PARTITION BY HASH」と記述しとします。ハッシュパーティションの場合は、キー列の値によってランダムに振り分けを行うため、値の範囲設定は不要です。この例では、商品コードをキーとして、USERS～USERS4 の表領域にランダムに振り分けています。

また、2つのキー列の値によってパーティションを分散したい場合は、「コンポジット・パーティション」を使用することもできます。「コンポジット・パーティション」とは、レンジ・リスト・ハッシュのパーティションを2種類組み合わせて分散する方法です。以下はレンジパーティションと、リストパーティションを組み合わせた「レンジ・リストパーティション」の SQL の例です。

```

CREATE TABLE "商品マスタ"
(
    "商品コード"          NUMBER,
    "商品名"              VARCHAR2(20),

```

```

        "価格"                NUMBER,
        "サイズ"             VARCHAR2(2),
        "更新日"             DATE
    )
PARTITION BY RANGE("価格")
SUBPARTITION BY LIST("サイズ")
(
    PARTITION PARTITION1 VALUES LESS THAN (100000)
    (
        SUBPARTITION SUBPARTITION1 VALUES ('S') TABLESPACE USERS,
        SUBPARTITION SUBPARTITION2 VALUES ('M') TABLESPACE USERS2,
        SUBPARTITION SUBPARTITION3 VALUES ('L') TABLESPACE USERS3,
        SUBPARTITION SUBPARTITION4 VALUES ('XL') TABLESPACE USERS4
    ),
    PARTITION PARTITION2 VALUES LESS THAN (MAXVALUE)
    (
        SUBPARTITION SUBPARTITION5 VALUES ('S') TABLESPACE USERS5,
        SUBPARTITION SUBPARTITION6 VALUES ('M') TABLESPACE USERS6,
        SUBPARTITION SUBPARTITION7 VALUES ('L') TABLESPACE USERS7,
        SUBPARTITION SUBPARTITION8 VALUES ('XL') TABLESPACE USERS8
    )
)
/

```

レンジ・リストパーティションを使用する場合は、「PARTITION BY RANGE」のあとに続けて「SUBPARTITION BY LIST」と指定し、それぞれにキー列を指定します。この例では、「価格」列をレンジパーティションのキー、「サイズ」列をリストパーティションのキーに指定しています。また、パーティションの設定の後にさらに括弧を記述し、第2キーの振り分け設定となる「サブパーティション」の記述を行ないます。今回の例では、パーティションの設定でレンジの値範囲を設定し、サブパーティションの設定で、サブパーティション名、リストの値範囲設定を行なっています。また、表領域の指定もサブパーティションの中で行ないます。この設定により、「価格」列と「サイズ」列の値の組み合わせによって、USER～USER8 の表領域に振り分けられることになります。



図4-5. コンポジット・パーティションの例

ただし、コンポジット・パーティションについては、Oracle の 9i 以降でサポートされています。また、Oracle 9i、10g では「レンジ・リストパーティション」、「レンジ・ハッシュパーティション」しか使用できません。11g 以降の場合は、「レンジ・レンジパーティション」「リスト・レンジパーティション」など 6 種類の組み合わせが使用できます。（ハッシュパーティションは、コンポジット・パーティションでは、サブパーティションとしてしか指定することができません。）

	レンジ	リスト	ハッシュ
レンジ	11g 以降	9i 以降	9i 以降
リスト	11g 以降	11g 以降	11g 以降

表4-3.コンポジット・パーティションの対応 Oracle バージョン

以上でパーティショニングについて説明しました。ただし、パフォーマンスチューニングにおいては、ディスクアクセスの発生率を下げるのが一番効果的ですので、これまで説明した他の手段を実施しても効果がない場合にパーティショニングを検討してください。

5 バックアップ & リカバリ



5-1. バックアップ & リカバリの概要

最後の章では、データベースの「バックアップ」および「リカバリ」の方法をご説明します。システムの運用に当たってはディスクが故障したなどの不慮の事態に備え、バックアップを定期的に取り取る事が大事です。また、せっかくバックアップファイルを作成しても、正しく復旧できなければ意味がありません。バックアップ方法とセットでリカバリ方法も覚えましょう。

■物理バックアップと論理バックアップ

バックアップ方法は、大きく「物理バックアップ」と「論理バックアップ」に分かれます。物理バックアップとは、Windows や UNIX などの OS 上で管理している物理ファイルを直接コピーするバックアップ方法です。また、論理バックアップとは、データベースのバックアップ専用ユーティリティを使用してバックアップを行う方法です。Oracle では、専用ユーティリティとしてエクスポート/インポートツールなどが提供されています。ユーティリティを使用して出力されるバックアップファイルは、バイナリファイル形式で格納された論理データとなります。論理バックアップでは、一部のテーブルのみバックアップできるなど柔軟にバックアップができるなどのメリットがありますが、バックアップを取得した移行に更新のあったデータが反映できないなど、物理バックアップに比べて不十分な点もあります。

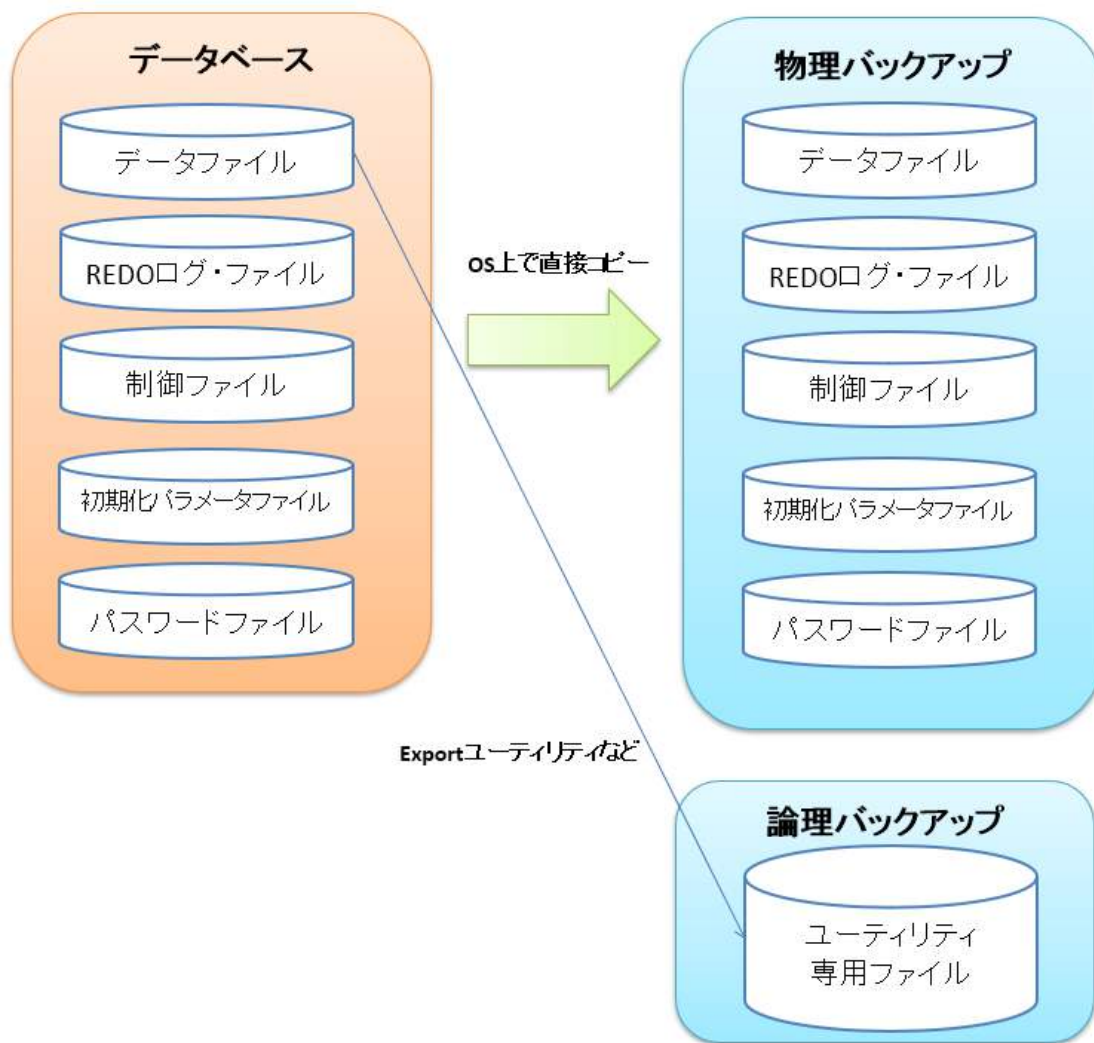


図5-1.物理バックアップと論理バックアップ

それぞれの長所・短所をまとめると以下の通りとなります。

種類	長所	短所
物理バックアップ	<ul style="list-style-type: none"> ・アーカイブログモード（後述）にしないと稼働中にバックアップができない ・REDO ログ情報を適用して、障害発生時に最新のデータに復旧できる ・ファイルが壊れたなどの物理的障害に対応できる 	<ul style="list-style-type: none"> ・稼働中のバックアップしたい際は手順が複雑となる ・ファイルサイズが大きい
論理バックアップ	<ul style="list-style-type: none"> ・稼働中にバックアップが可能 ・コマンドや GUI などのインターフェースを用意しているため、簡単に操作できる ・一部のテーブルのみバックアップできるなど柔軟な設定が可能 ・バックアップファイルのサイズが小さい 	<ul style="list-style-type: none"> ・データを取得した時点となるため、障害発生時のデータに復旧できない場合がある ・ファイルが壊れたなどの物理的障害には対応できない

表5-1.物理バックアップと論理バックアップ

物理バックアップでは物理障害にも対応できるため、全体バックアップとして利用します。また、論理バ

ックはファイルサイズが軽いメリットがありますが、最新のデータに復旧できないデメリットがあるため、差分バックアップとして利用します。本章の前半で物理バックアップおよびそのリカバリ手順を説明し、後半で論理バックアップおよびリカバリ手順をご説明します。

また、バックアップで重要となる「アーカイブログモード」について説明します。Oracle では、データベースの運用方式として「アーカイブログモード」と「ノーアーカイブログモード」の2つのモードがあります。データベースのデータについては、「データファイル」、トランザクション途中のデータは「REDO ログファイル」に記録されていますが、REDO ログファイルは循環して使用されるため、古いトランザクション情報は上書きされてしまいます。そのため、REDO ログファイルのバックアップだけでは、すべてのトランザクション情報を復旧できない可能性があります。アーカイブログモードにしておき、アーカイブログファイルもバックアップ対象に含めることで、必ず最新のデータ状態にすることが可能となります。アーカイブログモードは、トランザクションを記録する「REDO ログファイル」が一杯になったタイミングで、別途「アーカイブログファイル」と呼ばれる別のファイルに REDO ログの情報を保存しておくモードです。また、アーカイブログモードでは、「オンラインバックアップ（後述）」ができるというメリットもありますので、ぜひ、可用性の高いアーカイブログモードにしておくことをおすすめいたします。



ただし、アーカイブログファイルは増え続けるため、ディスクの空き容量を圧迫する問題があります。アーカイブログモードにした場合は、バックアップ実施後に削除するなど、定期的に削除する運用が必要となります。

現在アーカイブログモードであるかどうかは、コマンドラインベースで SQL 実行する専用ツール、「SQL*Plus」を使用することで確認することができます。手順は以下のとおりです。

- ① 「SQL*Plus」を起動します。Windows であれば、スタートメニューより「ファイル名を指定して実行」を選択し、名前の欄に「cmd」と入力することで表示されるコマンドプロンプトより、さらに「sqlplus」と入力することで起動します。
- ② 以下のコマンドにて、SYS ユーザーでログインします。

`sqlplus sys/(sys のパスワード) AS SYSDBA`



sysユーザーのログインパスワードはインストール時に指定します。既定ではsystemと同じパスワードになります。

- 3 以下のコマンドを実行します。

`ARCHIVE LOG LIST`

```
cmd 管理者: C:\Windows\system32\cmd.exe - sqlplus sys/(sysのパスワード) AS SYSDBA
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>sqlplus sys/(sysのパスワード) AS SYSDBA

SQL*Plus: Release 12.2.0.1.0 Production on 月 1月 7 05:11:43 2019

Copyright (c) 1982, 2016, Oracle. All rights reserved.

Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production
に接続されました。
SQL> ARCHIVE LOG LIST;
データベース・ログ・モード      非アーカイブ・モード
自動アーカイブ                  使用禁止
アーカイブ先                    C:\app\Administrator\virtual\product\12.2.0\dbhome_1\RDBMS
最も古いオンライン・ログ順序    1
現在のログ順序                  3
SQL> ─
```

画面 5-1. ARCHIVE LOG LIST コマンドの実行結果

上記実行結果では「データベース・ログ・モード」の箇所が「非アーカイブ・ログ・モード」となっていますが、この場合はノーアーカイブログモードとなります。アーカイブログモードに変更するには、続けて、以下の手順を実施することで可能です。

- ① データベースを停止します。

SHUTDOWN

データベースを停止すると、テーブルのアクセスができないなど、データベースが完全に使用できない状態になりますので、他のユーザーがデータベースを使用していないときに実施してください。

- ② データベースをマウント状態で起動します。

STARTUP MOUNT

マウント状態とは、データベースを起動する前の状態です。データベースは完全に起動されていないため、データベースを使用することはできませんが、システム管理者がデータベースにアクセスし、設定を変更することが可能です。

- ③ 以下の SQL を実行し、アーカイブログモードに変更します。

ALTER DATABASE ARCHIVELOG;

- ④ 再度、データベースを起動します。

ALTER DATABASE OPEN;

データベースの起動した以降は、アーカイブログが自動で作成されるようになります。なお、ノーアーカイブログモードに戻したい場合は、上記の③を「ALTER DATABASE NOARCHIVELOG」に変更の上、同手順を実施することで可能です。

5-2. 物理バックアップの手順

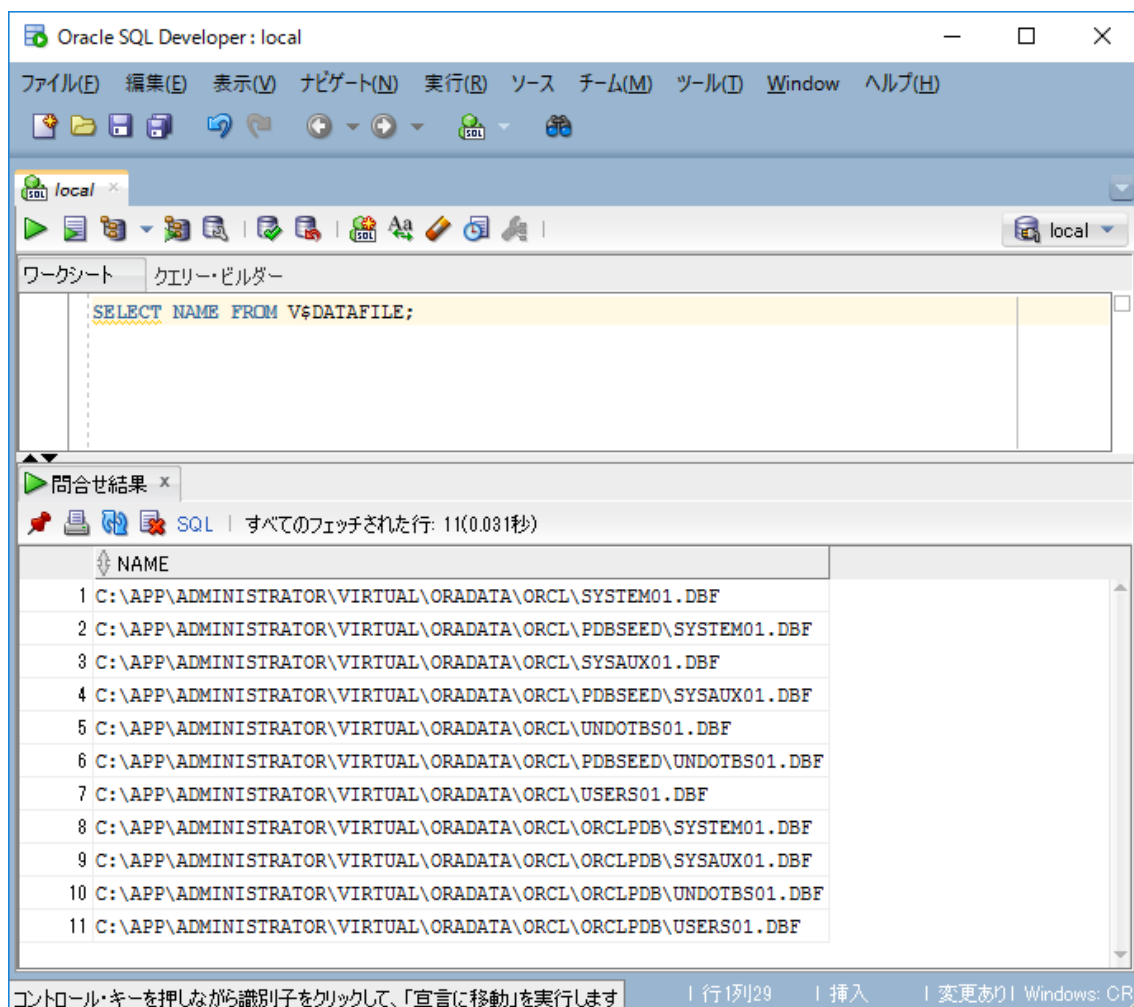
それでは、物理バックアップの手順からご紹介します。物理バックアップを実施するためには、まず、バックアップする対象のファイルを把握する必要があります。バックアップ対象となるファイルは以下の通りです。

ファイルの種類	説明	ファイルの格納場所
データファイル	データを格納するファイルです。	V\$DATAFILE で確認
REDO ログ ファイル	データベースに対する更新履歴を記録したファイルです。テーブルのデータの追加/変更/削除の情報や、すべての変更処理を保持します。	V\$LOGFILE で確認
アーカイブログファイル	REDO ログファイルが一杯になった時点で自動でコピーされるファイルです。※アーカイブログモードになっている場合のみ必要です。	V\$RECOVERY_FILE_DEST で確認
制御ファイル	コントロールファイルとも呼ばれます。データベース名などの情報が格納されます。	V\$CONTROLFILE で確認
初期化パラメータ ファイル	Oracle が専有するメモリサイズや、言語設定など、各環境に応じた設定情報を記録するファイルです。	< ORACLE_HOME > ¥dbs¥SPFIL < ORACLE_SID>EXE.ORA
パスワード ファイル	ログインに使用するパスワード情報が格納されたファイルです。	< ORACLE_HOME > ¥pwd¥SPFIL < ORACLE_SID>EXE.ORA

表5-2.バックアップ対象のファイルと格納場所

ファイルの格納場所は、環境によって異なることがありますが、ディクショナリビューに問い合わせることで、正確な場所を調べることができます。ディクショナリビューとは、Oracle に標準で作成された、データベース内の表や、索引、ユーザー権限などデータベースに係わる様々な情報を表形式で管理したものです。ユーザーは SELECT 文を発行することでこれらの情報を取得することができます。例えば、データファイルに関する情報は「V\$DATAFILE」と呼ばれるビューに格納されていますので、以下の SQL でデータファイルの物理ファイルの場所を確認することができます。

```
SELECT NAME FROM V$DATAFILE;
```

画面5-2. V\$DATAFILE の SELECT 結果

また、初期化パラメータファイル、パスワードファイルについては、決められた場所に格納されています。表1の中で記載している「ORACLE_HOME」は Oracle のインストールディレクトリ、「ORACLE_SID」はデータベース作成時に決めるデータベースの識別子のことです。これらは、Oracle のインストール時、またはデータベースの作成時に指定します。



Oracle Database 10g Express Edition の場合は、初期パラメータファイルは「<ドライブ>:\%oraclexe%\app\oracle\product\10.2.0\server\%dbs%\SPFILEXE.ORA」、パスワードファイルは「<ドライブ>:\%oraclexe%\app\oracle\product\10.2.0\server\%dbs%\SPFILEXE.ORA」となります。

これらのファイルもバックアップに含めておけば、Oracle のサーバー自体がクラッシュした場合でもリカバリが可能ですので、バックアップの際は、これらのファイルも忘れずに保存してください。

物理バックアップ方法は、さらに、データベースを停止した状態でバックアップする「オフラインバックアップ」と、データベース稼働中にバックアップする「オンラインバックアップ」に分かれます。それぞれの方法をご紹介します。



5-3. オフラインバックアップの手順

オフラインバックアップでは、一度データベースを停止した後、手作業でバックアップ対象ファイルをコピーする方法です。データベースを停止し、データを確定した状態でバックアップするため「一貫性バックアップ」とも呼ばれます。オフラインバックアップはノーアーカイブログモードでも実施が可能です。データベースを停止については、さきほどのアーカイブログモードの変更手順と同様、SQL*Plus にログインし、SHUTDOWN コマンドで停止することができます。データベース停止後、表 1 に記載の対象ファイルを任意のフォルダにコピーすることでバックアップができます。ファイルのコピーに関しては、コマンドライン、または Windows エクスプローラのどちらを使用しても構いません。SQL*Plus からは host コマンドを使用して OS のコマンドを実行することが可能です。Windows であれば以下のコマンドで、SQL*Plus からコピー操作を実行することができます。

```
HOST COPY C:¥APP¥USER ¥ORADATA¥SID¥SYSTEM01.DBF C:¥BACKUP
```

対象のファイルをすべてコピーすれば、バックアップは終了です。



このコマンドでは、同じサーバーの同じドライブにバックアップファイルをコピーしていますが、通常は、サーバー自体のクラッシュに備え、別のサーバーに保存します。火災などの自然災害に備えるのであれば、遠隔地のサーバーにバックアップファイルを転送し、別々の場所に保存しておくことが最善策となります。



5-4. オンラインバックアップの手順

オンラインバックアップは、データベース働中にバックアップする方法です。また、バックアップ中にデータの更新があっても問題なく実行できるため、「非一貫性バックアップ」とも呼ばれます。ただし、オンラインバックアップを実施するには、アーカイブログモードになっている必要があります。

オンラインバックを実施する際は、以下の SQL でバックアップモードに変更します。

```
ALTER DATABASE BEGIN BACKUP;
```

バックアップモードにすると、バックアップ中にデータが変更された場合でも、問題なく REDO ログファイルに追加で変更情報が記録されるため、データの一貫性を保つことができます。バックアップモードに変更した後は、以下の手順でバックアップファイルのコピーを実施します。

・データファイル

通常の OS コマンドでは、稼働中にデータの更新があった場合にデータの不整合になるため、OCOPY という専用コマンドを使用してバックアップファイルをコピーする必要があります。

引数は通常の COPY コマンドと同様です。SQL*Plus から続けて実施する場合は、HOST コマンドを使用して以下の SQL で実行できます。

```
HOST OCOPY C:¥APP¥USER ¥ORADATA¥SID¥USER01.DBF C:¥BACKUP;
```

・制御ファイル

以下の ALTER DATABASE コマンドでコピーします。

```
ALTER DATABASE BACKUP CONTROLFILE TO 'C:¥BACKUP¥CONTROL_BK.CTL';
```

・REDO ログファイル（アーカイブログファイル）

REDO ログファイルについては、アーカイブログファイルに変換の上、アーカイブログファイルのみをバックアップします。以下のコマンドでアーカイブログファイルを作成できます。

ALTER SYSTEM ARCHIVE LOG CURRENT;

作成後は、オフラインバックアップと同様、COPY コマンドまたは Windows エクスプローラでコピーして構いません。

・初期化パラメータファイル、パスワードファイル

作成後は、オフラインバックアップと同様、COPY コマンドまたは Windows エクスプローラでコピーして構いません。

バックアップ終了後は以下のコマンドでバックアップモードを終了します。

ALTER DATABASE END BACKUP;

これでオンラインバックアップは完了です。



5-5. 物理バックアップからのリカバリ手順

それでは障害が発生した際のリカバリ方法についてご説明します。リカバリ方法については大きく、「クラッシュ・リカバリ」と「メディア・リカバリ」に分かれます。クラッシュ・リカバリはデータベース稼働中に異常終了した場合など、または、「SHUTDOWN ABORT」コマンドによって強制停止された場合に行うリカバリです。クラッシュ・リカバリについては、データファイルと REDO ログファイルを使用して行われますが、アーカイブログファイルは使用されません。また、データファイルや制御ファイルが壊れた場合は、リカバリすることができません。メディア・リカバリはデータファイルや制御ファイル、初期化パラメータファイルなどが壊れた場合に行うリカバリです。

クラッシュ・リカバリについては、次回データベース起動のタイミングで、データベースによって自動でリカバリ処理が行われるため、ユーザーが手動で実行する必要はありません。

メディア・リカバリについてはユーザーが手動で実行する必要がありますので、実際にデータファイルを削除した上で、復旧するまでの流れを説明します。もし、実際に動作を確認される場合は、一時的にデータが削除されても問題ない環境で実施してください。



障害の種類によって手順は異なります。例えば、データベースが途中で異常終了し、データが不整合になった場合は、次回データベース起動時にOracleが自動で復旧します。

まず、障害状況を発生させるため、既定のデータファイルである「USERS01.DBF」データファイルを削除します。データベースを SHUTDOWN コマンドで停止した後、V\$DATAFILE ディクショナリビューより、格納されている場所を確認の上、削除します。削除前には、必ず、同ファイルのバックアップファイルを作成していることをご確認ください。

ファイルの削除後は、SQL*Plus を起動し、SYS ユーザーなどの SYSDBA 権限を持つユーザーでログインします。ログイン後、以下のコマンドでデータベースを起動します。

STARTUP

STARTUP はデータベースを起動するコマンドです。実行すると、インスタンスの起動、マウントまでは実施できましたが、データベースのオープンの段階で「ORA-01157: データファイル 4 を識別/ロックできません」というエラーとファイル名が表示されます。

```
管理: C:\Windows\system32\cmd.exe - sqlplus sys/(sysのパスワード) AS SYSDBA
SQL> STARTUP
ORACLEインスタンスが起動しました。

Total System Global Area 2583691264 bytes
Fixed Size                  8922232 bytes
Variable Size              687868808 bytes
Database Buffers          1879048192 bytes
Redo Buffers                7852032 bytes
データベースがマウントされました。
ORA-01157: データファイル7を識別/ロックできません -
DBWRトレース・ファイルを参照してください
ORA-01110: データファイル7:
'C:\APP\ADMINISTRATOR\VIRTUAL\ORADATA\ORCL\USERS01.DBF'

SQL> _
```

画面5-3.起動エラーの画面

「USERS01.DBF」データファイルを読み込めないために当エラーが発生しています。

バックアップフォルダより、対象のデータファイルをコピーしてください。コピー後、以下のSQLにて、データベースを再度オープンします。

```
ALTER DATABASE OPEN;
```

このとき、ノーアーカイブログモードの場合は、エラーなしで正常に起動されますが、アーカイブログモードの場合は、さらに、「ORA-01113: ファイルはメディア・リカバリが必要です」というエラーが表示されますので、以下のSQLを実行します。

```
RECOVER DATABASE;
```

今回はデータベースを停止後にデータファイルを削除しましたので、データファイルは必ず最新の状態ですが、場合によってはデータファイルが古い状態があります。RECOVER DATABASE を使用すれば、アーカイブログを適用し、データを最新の状態にすることができます。「メディア・リカバリ」が完了しました」と表示されたら完了です。再度、「ALTER DATABASE OPEN」で起動してください

```
ALTER DATABASE OPEN;
```

今度はエラーなしで起動できたことが確認できると思います。以上でメディア・リカバリは完了です。データファイルが何らかの理由で削除されてしまったり、中身が壊れてしまった場合は、この手順でリカバリが可能です。

5-6. 論理バックアップ & リカバリの手順

続けて、論理バックアップの手順を使用します。論理バックアップはデータベースの専用のユーティリティを使用してバックアップおよびリカバリを行う方法です。Oracle では代表的なツールとして、エクスポート/インポートツールが用意されています。エクスポートはファイルに出力することでバックアップするツールです。また、インポートはファイルからデータベースに復元するツールです。エクスポート/インポートユーティリティは、コマンドベースのプログラムとなっています。Windows であればスタートメニューより「ファイル名を指定して実行」→「cmd」と入力し、コマンドプロンプトから実行します。EXPDP コマンドでバックアップ、IMPDP コマンドでリストアの実行が可能です。コマンドの書式は以下の通りです。

実行する前はあらかじめ「ディレクトリ」の作成が必要です。ディレクトリはエクスポートファイルをはじめ、Oracle の機能でファイルを出力する際に必要となるパスの設定です。以下の SQL で作成できます。（この作業は、SQL*Plus で実行してください）

```
CREATE DIRECTORY WORK_DIR AS 'c:\work';
```

上記 SQL ではディレクトリ名を「WORK_DIR」、対応するパスを「c:\work」としてディレクトリを作成しています。「WORK_DIR」はあくまで Oracle が識別する名前となり、サーバー上の実際のパスは「c:\work」となることに注意してください。

ディレクトリの作成後は以下のコマンドでエクスポートが実行可能です。（移行の作業はコマンドプロンプトにて実行してください。）

```
EXPDP ユーザーID/パスワード DIRECTORY=ディレクトリ名 DUMPFILE= ファイル名 <オプション=値 オプション=値…>
```

「EXPDP」に続けて実行するユーザー名とパスワードを指定します。次に、「DIRECTORY=」に続けて先ほど作成したディレクトリ名、「FILE=」に続けて出力するバックアップファイル（読み込むバックアップファイル）をフルパスで指定します。ファイル名は任意で構いませんが、ファイルの拡張子は「DMP」とするのが慣例となっています。これらの指定に加え、任意でオプションを指定します。バックアップの実行例は以下のとおりです。

```
EXPDP USERID/PASSWORD DIRECTORY=WORK_DIR DUMPFILE= EXPDAT.DMP FULL=Y
```

「FULL=Y」とオプション指定することでデータベース全体をバックアップします。ユーザー単位で指定する場合は以下のようになります。

```
EXPDP USERID/PASSWORD DIRECTORY=WORK_DIR DUMPFILE=EXPDAT.DMP  
SCHMAS=HR, SCOTT
```

「SCHMAS=」に続けて対象ユーザーをカンマ区切りで指定します。この例では、HR ユーザー、SCOTT ユーザーが所有するテーブルとそのデータが対象にバックアップファイルを作成します。

EXPDP コマンドによってバックアップファイルを作成しておけば、IMPDP コマンドを使用して、リカバリが可能です。IMP コマンドによるリカバリの実行例は以下のとおりです。

```
IMPDP USERID/PASSWORD FILE=D:\BACKUP\EXPDAT.DMP IGNORE=Y
```

画面5.インポートの実行結果

この例では、バックアップファイルの内容をすべて復元します（例えば、データベース全体のバックアップファイルを作成していた場合は、データベース単位の復元となります。）バックアップファイルの中から部分的にリカバリすることも可能です。例えば、HR と SCOTT の2ユーザーを対象にバックアップファイルを作成していた場合でも、以下のように記述すれば HR ユーザーだけを復元することができます。

IMPDP USERID/PASSWORD DIRECTORY=WORK_DIR FILE= EXPDAT.DMP SCHEMAS=HR

このようにオプションによって柔軟な設定が可能となります。EXPDP/IMPDP コマンドの代表的なオプションは以下のとおりとなります。オプションによっては IMPDP コマンドでしか指定できない場合がありますのでご注意ください。

オプション名	EXP	IMP	説明
FULL	○	○	データベース全体をエクスポート/インポートする場合は「Y」を指定します。
DUMPFIL	○	○	エクスポートで書き出すファイル/インポートで読み込むファイルを指定します。
LOGFILE	○	○	エクスポート/インポートの実行結果をログに出力する場合はログファイル名を記述します。
SCHEMAS	○	○	ユーザー単位でエクスポートする場合は対象のユーザーを指定します。複数指定する場合は、カンマ区切りで指定します。
TABLES	○	○	テーブル単位でエクスポート/インポートする場合は対象のテーブルを指定します。複数指定する場合は、カンマ区切りで指定します。
CONTENT	○	○	テーブルのレコードのみの場合は「data_only」 テーブル定義のみの場合は「metadata_only」を指定します。
REMAP_SCHEMA		○	インポート先のユーザーをエクスポート時のものより変更したい場合に指定します。
REMAP_TABLESPACE		○	インポート先の表領域をエクスポート時のものより変更したい場合に指定します。
REMAP_DATAFILE		○	インポート先のデータファイルをエクスポート時のものより変更したい場合に指定します。

表5-3.エクスポート/インポートの代表的なパラメータ

エクスポート/インポートユーティリティによる論理バックアップ方法は、物理バックアップと比較して操作が簡単である点、細かな単位でバックアップできる点など様々なメリットがあります。また、オンラインバックアップと同様、データベース稼働中でもバックアップが可能です。ただし、エクスポート/インポートユーティリティではリカバリ時にアーカイブログの適用はできず、バックアップデータを作成した時点までしか復元することができませんのでご注意ください。



最後に

本ドキュメントは以上になります。

以上で初級編から上級編続けて基本的な機能をご紹介しました。ここまで読んでいただいた方は基本がしっかりと身につけていますので、ぜひ実践でスキルを磨いていただければと思います。

また、弊社の Oracle 開発支援ツールとして「SI Object Browser」という製品があります。SI Object Browser では SQL を使わず、GUI 操作で実行計画の表示や適切なインデックスの作成や、エクスポートやインポートが作成でき、より開発効率を高めることができるツールです。以下のブログでも画面入りでご紹介していますので、ご興味があればぜひご覧ください。

<製品ホームページ>

<https://products.sint.co.jp/siob>

<ブログ>

<https://products.sint.co.jp/siob/blog>



お問い合わせ先

本ドキュメントや弊社製品に関しましてご不明点がございましたらお知らせください。

株式会社システムインテグレータ Object Browser 事業部 営業担当

TEL : 03-5768-7979 東京営業所 後迫（ウシロザコ）、横島

06-4706-5471 大阪支社 永田

E-Mail : oob@sint.co.jp

URL : <https://products.sint.co.jp/siob/inquiry>

※記載されている商品名は、各社の商標または登録商標です。

※当社の許可なく、本ドキュメントの全部または一部を、広くご本人以外の方が利用可能な状態にする（ホームページにて公開する、不特定 多数の人にメールを転送するなど）ことはご遠慮ください。