最終更新 2014/07/17

■インストールとアップグレードの新機能

インストールの新機能

-> Optimal Flexible Architecture (OFA) の強化

・$ORACLE_BASE を指定する必要がある（空にするとアラートログにメッセージ）

・フラッシュリカバリ領域とデータファイルは$ORACLE_BASE 配下に作成し、それぞれを別ディスクにマウントすることを推奨

・$CRS_HOME は $ORACLE_BASE 配下に作成できない

・Oracle インベントリのディレクトリパス変更（$ORACLE_BASE が/opt/app/oracle の場合）

10g /opt/app/oracle/oraInventory

11g /opt/app/oraInventory ※$ORACLE_BASE の 1 つ上のディレクトリに作成される

-> インストールの変更点

11g 新しく加わったコンポーネント

名前	説明	ｲﾝｽﾄｰﾙ
Oracle Application Express	Web アプリを簡単に開発実行できる統合環境（10g では Oracle HTML DB として提供）	自動
SQL Developer	DB オブジェクトの参照、SQL 文と PL/SQL 文の編集やデバッグができる GUI ツール	自動
Oracle Warehouse Builder	DWH 環境のための ETL（抽出変換ロード）などの管理作業を実行出来る GUI ツール	自動
Oracle Database Vault	Oracle の特権ユーザを管理し、セキュリティを強化する	手動
Oracle Real Application Testing	DB リプレイと SQL パフォーマンスアナライザを提供する	手動
Oracle Configuration Manager	構成情報の収集と Oracle 構成リポジトリへのアップロードを行う（10g では Customer Configuration Repository として提供）	手動

廃止されたコンポーネント

名前	説明	11g からのコンポーネント
iSQL*Plus	ブラウザから SQL*Plus 機能を提供	EM の SQL ﾜｰｸﾍﾞﾝﾁ、SQL Developer
Oracle Workflow	Oracle E-Business Suite エンジンを使用したビジネスルールに基づくワークフロー管理システム	Oracle BPEL Process Mnager のワークフローサービス
Oracle Data Mining Scoring Engine	Oracle Data Mining の実行環境を提供	Oracle Data Miner
OEM Java コンソール	スタンドアロンの Java アプリケーションとして提供される EM を使用するコンソール	EM Database Control EM Grid Control

非推奨のコンポーネント

名前	説明	11g からのコンポーネント
Oracle Ultra Search	企業向けサーチエンジンとして様々なリソースを収集し、必要な情報に迅速にアクセスする機能を提供	Oracle Secure Enterprise Search
JDK/JRE 1.4	Java 言語による開発/実行のためのソフトウェアセット	JDK/JRE 5.0（1.5 もサポート）
CTXPATH 索引	Oracle Text ドキュメントに対する効果的なパスを見つける索引	XMLIndex

OS グループの指定

UNIX 環境では、特権ユーザ認証として OSDBA グループと OSOPER グループがあったが 11g で ASM 管理者用の OSASM が追加。

カスタムインストール時または、Oracle オーナーの所属グループが DBA でない場合に指定する画面が表示される。

-> データベース作成の変更点

OUI でインストールと同時に DB 作成では、カスタム以外は「DB の名前」と「キャラクタセット」しか指定できなかったが、
「メモリー管理の詳細」と「セキュリティ設定の無効化」を設定できるようになった。
RAW デバイスの指定はできなくなった。(RAW デバイスを指定するにはカスタム DB 作成を選択する)
DBCA を使用した場合も同様だが「自動化タスクメンテナンスの有効化と無効化」の設定も可能。

アップグレードの新機能

-> アップグレード前の分析

既存の DB を直接アップグレードする場合、9.2.0.4、10.1.0.2、10.2.0.1 以上のバージョンが必要。
直接アップグレードできるかの検証は 11g の $ORACLE_HOME/rdbms/admin/utlu111i.sql を既存の DB にコピーし実行する。
※R2 の場合は、utlu112i.sql

-> アップグレードのパフォーマンス向上

無効となった PL/SQL や Java オブジェクトを再コンパイルする為に使用する utlrp.sql のパフォーマンスが向上。
utlrp.sql の引数に並列度を指定可能。0 の場合は CPU_COUNT、PARALLEL_THREADS_PER_CPU パラメータに基づく。
1 の場合は、順次再コンパイルを行う。

-> アップグレード後の検証

アップグレードしたコンポーネントに問題ないかは、$ORACLE_HOME/rdbms/admin/utlu111s.sql を実行する。
Status の意味は次の通り。
VALID　　：アップグレードは正常に完了
UPGRADING：アップグレードは完了していない
INVALID　：問題が生じている。コンパイル問題であれば手動で utlrp.sql を実行することで解決する場合がある

-> DBUA (Database Upgrade Assistant) の拡張機能

Oracle10g Express Edition からのアップグレードも可能。
ASM インスタンスが検出されると ASM インスタンスのアップグレードページも表示される。
シングル環境の場合、DB のファイルをそのままにすることも ASM に移動させることも可能。
アップグレード完了しても、COMPATIBLE 初期化パラメータは元のまま (10.0.0) である。
11g の機能の大半は 11.1.0.0.0 以上の為、パラメータ変更すること。

-> 変更された初期化パラメータ

パラメータ	説明
XXXX_DUMP_DEST	DIAGNOSTIC_DEST パラメータに変更 (アラートログなどの格納先を指定するパラメータ)
UNDO_MANAGEMENT	値が AUTO に変更。デフォルトで自動 UNDO 管理を行う
CONTROL_MAANGEMENT_PACK_ACCESS	使用できる管理パックを制限するには EM を使用する必要があったが、パラメータでの指定が可能 DIGNOSTIC　　：Dignostic Pack に含まれる機能 (AWR,ADDM など) の使用 DIGNOSTIC+TUNING：上記に加え、Tuning Pack に含まれる機能 (SQL チューニングアドバイザ) の使用も可能 NONE　　：上記 2 つの何れの機能も使用できない

廃止された初期化パラメータを確認するには、V$OBSOLETE_PARAMETER ビューで確認可能。

ダイレクト NFS クライアント

DB ファイルはローカルではなく NAS を使用している場合もあり、ダイレクト NFS クライアントを使うことで

NFS クライアントを簡略化しパフォーマンスと管理性を向上させることが可能。
NAS をマウントするのは OS のカーネルと使うが、直接制御は Oracle で行うことでパフォーマンスが向上。
10g までは OS キャッシュ回避の構成パラメータ設定などが必要だが、それも不要になった。

-> ダイレクト NFS クライアントの設定

① NFS サーバでのディレクトリ公開（/vol/DATA/oradata）
② カーネルの NFS ドライバにてマウント（/DATA/oradata にマウント）
③ orafstab の作成（任意）
④ ODM NFS ライブラリへの置換（シンボリックリンクを libodm11.so ⇒ libnfsodm11.so（ODM NFS ライブラリ）にする）

orafstab のファイルはマウントされたエントリを検索するファイルで、次の順序で検索する。
$ORACLE_HOME/dbs/orafstab ⇒ /etc/orafstab ⇒ /etc/mtab

orafstab server:NFS サーバ名 path:NFS サーバへのパス export:NFS のパス mount:マウントポイント	サンプル server:sv2 path:192.168.11.2 export:/vol/DATA/oradata mount:/vol/DATA/oradata

④までの手順を実行し DB を再起動することで有効になる。アラートログに以下の情報が記録される。
Oracle Direct NFS ODM Library Version 2.0

NFS マウントされたディレクトリに DB ファイルを作成すると事で、NFS クライアントによるファイル管理が実行される。
ダイレクト NFS の監視に利用できるビュー

ビューの名前	説明
V$DNFS_SERVERS	ダイレクト NFS クライアントがアクセスする NFS サーバに関する情報
V$DNFS_FILES	ダイレクト NFS でオープンしている Oracle ファイル情報
V$DNFS_CHANNELS	ダイレクト NFS で接続している Oracle プロセス（チャンネル）情報
V$DNFS_STATS	ダイレクト NFS による Oracle プロセスの操作統計情報

ホットパッチ（オンラインパッチ）

> 概要

通常のパッチは Oracle バイナリモジュールの再リンクを伴う場合、インスタンスの停止が必要だった。
ホットパッチは、インスタンスの停止が不要。通常のパッチは「.o（オブジェクト）ファイル」や
「.a（アーカイブ）ライブラリ」のみ含まれていたが、ホットパッチは「.so（共有ライブラリ）」も含まれている為、
Oracle バイナリモジュールの再リンクが不要になっている。

ホットパッチか確認するには、パッチのディレクトリに移動し以下のコマンドを実行する。
$ORACLE_HOME/OPatch/opatch query -is_online_patch
末尾の出力 Patch is a online patch: true が true になってればホットパッチである。

> 考慮事項

ホットパッチが適用できるのは、Linux x86、Linux x86-64（AMD64/EM64T）、Solaris（SPARC 64bit）のみ。
起動しているインスタンスで実行中の Oracle プロセス毎に 1 つの OS ページが必要。
Linux は 4MB、Solaris は 8MB のメモリーを追加消費。（同時に実行されている Oracle プロセスやパッチサイズ依存）

■ストレージ管理の新機能

管理性の拡張

> SYSASM ロール

管理者権限（as sysdba など）で接続する時は、OS 認証または、パスワード認証を行う。

ASM インスタンスに接続する為に SYSASM ロールが追加された。（今まで通り SYSDBA での管理も可能）

SYSASM と対応付ける OS グループはカスタムインストール時に明示的に OS グループ設定画面が表示される。

OS 認証

インストール時に指定した OS グループが不明な場合は $ORACLE_HOME/rdbms/lib/config.c の中に記述されている

#define SS_DBA_GRP、SS_OPER_GRP、SS_ASM_GRP の定義で確認可能。

OS ユーザーを上記の定義のグループに属させることで OS 認証が可能になる。

パスワードファイル認証

orapwd ユーティリティを使いパスワード作成を行い、REMOTE_LOGIN_PASSWORDFILE パラメータに

EXCLUSIVE を設定（デフォルト）することでパスワードファイル認証が有効になる。

```
パスワード認証の例
$ export ORACLE_SID=+ASM
$ sqlplus / as sysasm
SQL> CREATE USER scott IDENTIFIDE BY tiger;
SQL> GRANT SYSASM TO scott;
SQL> connect scott/tiger@sv1/+ASM as sysasm
```

管理者権限が割り振られているユーザー及び権限は、V$PWFILE_USERS で確認可能。

SYSDBA 接続ではパスワード認証より OS 認証が優先されるが、SYSASM ロールの場合、SYSASM ロールを付与された OS ユーザであってもパスワード認証になる。「@sv1/+ASM」といった簡易接続（@<ホスト名>:<リスナーポート>/<サービス名>）などの Oracle Net 接続を使用した場合もパスワードが使われる。（リモート OS 認証を無効にしている為）

> ディスクグループの互換性

ASM ディスクグループは ASM 互換性と、RDBMS 互換性で管理される。

互換性	属性	説明
ASM	COMPATIBLE.ASM	ディスクグループを管理する ASM の最小バージョンを指定。メタデータ構造や機能を決定する
RDBMS	COMPATIBLE.RDBMS	ディスクグループを管理する DB の最小バージョンを指定。使用する ASM のファイル形式を決定する

互換性を設定する（*マークはデフォルト）

{ CREATE | ALTER } DISKGROUP ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名

ATTRIBUTE 'COMPATIBLE.ASM'='{10.1* | 10.2 | 11.1}', 'COMPATIBLE.RDBMS'='{10.1* | 10.2 | 11.1}',

※高い値に設定した後は、元に戻すことはできない

互換性バージョン設定のルール

DB インスタンスの互換性 >= COMPATIBLE.RDBMS <= COMPATIBLE.ASM <= ASM インスタンスの互換性

ディスクグループの互換は V$ASM_DISKGROUP の COMPATIBILITY 列と、DATABASE_COMPATIBILITY 列で確認可能。

インスタンスの互換性は COMPATIBLE 初期化パラメータで確認可能。

スケーラビリティとパフォーマンスの拡張

> 割り当てユニットサイズの指定

ディスクグループに格納される ASM ファイルは均等に分散され割り当てユニット（AU）単位で領域が確保される。

10g では 1MB 固定だったが、1～64MB（2 の累乗）の指定が可能になった。（作成後の変更は不可）

大きな値を指定することで、シーケンシャルリードの性能が向上するが、分散効率が悪くなり I/O 競合の可能性が高くなる。

（例）CREATE DISKGROUP EXTERNAL REDUNDANCY DATA1 DISK '/dev/raw/raw3','/dev/raw/raw4' ATTRIBUTE 'AU_SIZE'='4M'

> 可変エクステントサイズ

10g では 1 エクステント 1AU の固定だったが、11g では、エクステント数に応じて AU サイズが可変する。

ASM ファイルのエクステント数	1 エクステントのサイズ(11gR1)	1 エクステントのサイズ(11gR2)
0 ～ 19,999	1AU	1AU
20,000 ～ 39,999	8AU	4AU
40,000 ～	64AU	16AU

これにより大きな ASM データファイルがサポートされ、大規模なデータベースに必要な SGA メモリーが少なくてすむ。

また、ファイルの作成操作やオープン操作のパフォーマンスが向上する。

AU_SIZE が 1MB の場合の最大データサイズ

ディスクグループ構成	10.1	11.2
外部冗長性（External Redundancy）	16TB	140PB
標準冗長性（Normal Redundancy）	5.8TB	23PB
高冗長性（High Redundancy）	3.9TB	15PB

新しい管理オプション

10g では ASM ディスクグループに障害があると、ディスクグループのマウントや削除ができなかった。

11g では強制オプションを使用することで上記が可能になっており、障害チェックの拡張もされている。

> ディスクグループのチェック

メタデータに対する整合性チェックは、次の構文に統一されている。

ALTER DISKGROUP ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名 CHECK [{REPAIR | NOREPAIR*}]

一貫性がない状態が検出されたときにエラー修復をするか指定。NOREPAIR は修復せず警告表示のみ。

> ディスクグループの強制マウント

通常の冗長性（2 方向）と高い冗長性（3 方向）の場合、障害グループが作成され、異なる障害グループにプライマリ AU とミラーAU を配置する。10g の時は冗長性を設定していても全てのディスクが使用可能でなければマウントできなかった。

11g では使用できないディスクが存在してもマウントすることが可能になった。

ALTER DISKGROUP ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名 MOUNT FORCE

> ディスクグループの強制削除

ディスクグループをマウントしている場合は普通に削除が可能。

マウントできないディスクグループを削除するには OS レベルでの初期化が必要だった。

11g ではマウントできないディスクの強制削除が可能。（マウントしてるディスクに対してやるとエラーになる）

DROP DISKGROUP ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名 FORCE INCLUDING CONTENTS

> 高速リバランス

ディスクグループにディスクを追加/削除した場合、リバランスが実行される。

RAC 環境など複数ホストで構成される場合、ホスト間の通信負荷がある為、リバランス効率が悪くなる。

RESTRICTED モードでマウントすることで、ディスクグループにアクセスできるインスタンスを単一ノードに限定できる。

ALTER DISKGROUP ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名 MOUNT RESTRICTED　または

STARTUP RESTRICT（ASM_DISKGROUPS 初期化パラメータで指定されたディスクグループを RESTRICTED モードでマウント）

（流れ）ディスマウント　⇒　RESTRICTED モードでマウント　⇒　メンテナンス　⇒　ディスマウント　⇒　通常のマウント

ASM 高速ミラー再同期化

> 概要

10g の場合、ディスク障害、コントローラ障害の何れも ASM ディスクがオフライン化されディスクが削除される。

削除されたディスクに格納されたエクステントを残りのディスクに再作成する。

ディスクパス障害が復旧すると再びエクステントのリバランスが実行される。

※ディスクパス障害の場合は、ディスクが壊れている訳ではないので、ディスク削除が無駄

11g のディスクパス障害の場合、ディスクをオフラインにするが一定時間（disk_repair_time で指定）削除は保留する。

オフライン中に変更されたエクステントが追跡され、オンライン時に変更されたエクステントのみリバランスされる。

削除時間の設定（デフォルトは 3.6 時間）

ALTER DISKGROUP ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名 SET ATTRIBUTE 'disk_repair_time' = '4.5h'

オンライン状態のディスクに対して有効。オフラインになってるディスクに設定は反映されない。

※V$_ASM_ATTRIBUTE で、DISKGROUP に設定されている属性が確認可能

ディスクがオフラインになった後、削除されるまでの時間は、V$ASM_DISK の REPAIR_TIMER 列で確認可能。

特定のディスクをオフライン

ALTER DISKGROUP ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名 OFFLINE { ALL | DISK ﾃﾞｨｽｸ名 } [DROP AFTER 時間]

特定の障害グループに属するディスクのみオフライン

ALTER DISKGROUP ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名 OFFLINE ALL IN FAILGROUP 障害ｸﾞﾙｰﾌﾟ名 [DROP AFTER 時間]

ASM 優先読み取りミラー

ASM 拡張クラスタ環境では遠隔地のディスクもローカルノードにあるように構成される。

10g は常にプライマリ AU を読み込んでいたが、11g の優先読み取り設定を行うと近いノードの AU を読み込むようになる。

-> 設定

各ノードの ASM インスタンスで ASM_PREFERRED_READ_FAILURE_GROUPS 初期化パラメータを指定する。

ASM_PREFERRED_READ_FAILURE_GROUPS = ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名.障害ｸﾞﾙｰﾌﾟ名

で設定した障害グループから読み取りが実行される。

ASMCMD の拡張機能

ASMCMD は自動ストレージ管理（ASM）ディスク・グループ内のファイルおよびディレクトリを簡単に表示したり

操作するために使用するコマンドラインユーティリティ（ASMCMD でユーティリティを起動し各コマンドを実行する）

-> ASM 情報の確認

ASM ディスクに関する情報を表示する。引数を何も指定しない場合は ASM のディスクパスを表示する。

lsdsk ｵﾌﾟｼｮﾝ [-d ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ] [ﾊﾟｽ名]

ｵﾌﾟｼｮﾝ	説明
-k	サイズ情報を表示
-s	ディスク I/O の統計情報を表示
-p	ステータス情報を表示（ディスク番号、マウントステータス、ディスクモード、ディスクの状態など）
-t	時間情報を表示（グループ追加日付、マウント日付、自動削除されるまでの残り時間、検出で戻される OS パス）
-c	V$ASM_DISK ビューから情報を取得する
-g	GV$ASM_DISK ビューから情報を取得する
-H	ヘッダー情報（列見出し）を非表示にする
-I	非接続モードの指定。ディスクヘッダーをスキャンし情報を表示。省略時は接続モードとして V$ビューから情報を取得

-> メタデータのバックアップ/リストア

10g の時はオブジェクトを誤って削除した場合は、手動で再作成が必要だった。

11g ではディスクグループや、メタデータ（ディレクトリやテンプレート）のバックアップ及びリストアが可能になった。

ASM ファイル自体が破損した場合は、RMAN などからリカバリが必要。

md_backup [-b ﾊﾞｯｸｱｯﾌﾟ ﾌｧｲﾙ名] [-g 'ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名']

md_restore -b ﾊﾞｯｸｱｯﾌﾟ ﾌｧｲﾙ名　[ｵﾌﾟｼｮﾝ]

オプション	説明
-i	エラーを無視する
-t full | nodg | newdg	ディスクグループの作成方法（省略時は full） full ：ディスクグループの作成とメタデータのリストア nodg ：メタデータのみリストア newdg：新しい名前のディスクグループを作成し、メタデータをリストア（-o による指定が必要）
-f ﾌｧｲﾙ名	リストアを行う SQL 文を指定のテキストファイルに出力する（リストアは実施しない）
-g ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名	リストアするディスクグループ名。複数指定は,で区切る。省略時は全てのディスクグループをリストア
-o 旧ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名 新ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名	-t newdg を指定した場合に、指定した古いディスクグループ名を、新しいディスクグループ名として作成する

-> 不良ブロック修復

通常の冗長性、または高い冗長性を使用している場合、プライマリ AU に障害が発生した場合、ミラーAU が読み取られる。

これに加え 11g では、I/O エラーが発生したブロックを不良ブロックとしてミラーAU からプライマリ AU にコピーし修復する機能が実装された。自動で修復されるが以下のコマンドで手動で修復させることも可能。

remap　ﾃﾞｨｽｸｸﾞﾙｰﾌﾟ名　ﾃﾞｨｽｸ名　開始ﾌﾞﾛｯｸ番号-終了ﾌﾞﾛｯｸ番号

■変更リスクに対する新機能

スナップショットスタンバイデータベース

ALTER DATABASE CONVERT TO SNAPSHOT { STANDBY | PHYSICAL STANDBY }

・フィジカルスタンバイデータベースを、スナップショットスタンバイに切り替えることでアクティブにし、
　レポート作成やテストが実行可能になる。プライマリからのスイッチオーバーや F/O は出来なくなる

・スナップショットスタンバイに切り替えたタイミングで保証付きリストアポイントを作成し、

フィジカルスタンバイに戻した時にリストアされる

・スナップショットスタンバイの時もプライマリデータベースからの REDO ログは受信し続ける

データベースリプレイ

本番データベースのワークロードを取得し、テスト環境でリプレイすることで、アップグレードなどの検証が可能。

-> 実行方法

① ワークロードの取得

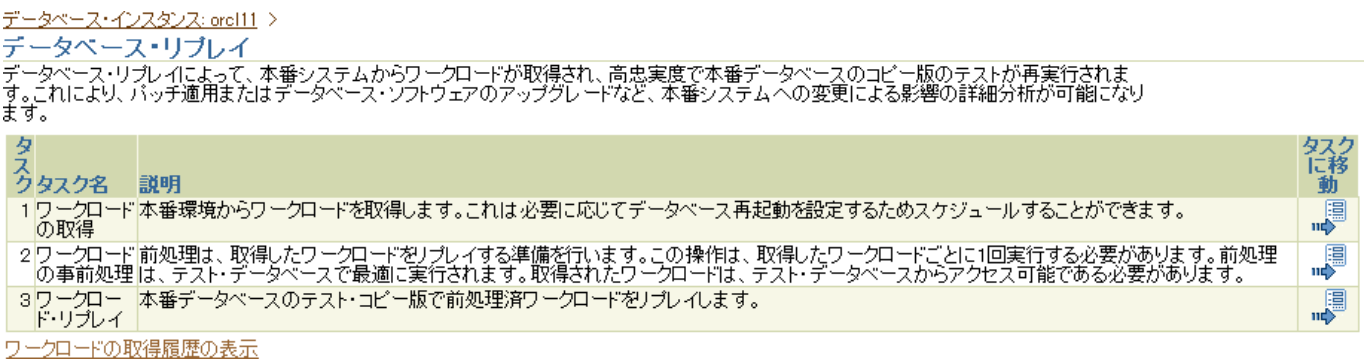
② ワークロードの事前処理

③ ワークロードのリプレイ

④ 分析とレポート

① ワークロードの取得

[ソフトウェアとサポート] - [データベースリプレイ] - [ワークロードの取得] の右側にあるアイコンをクリック



ワークロード取得時にトランザクションが進行中である場合、リプレイ時にエラーが発生する原因になる。

DB を RESTRICTED モードで起動し取得することを推奨。、ワークロード取得開始で自動で UNRESTRICTED モードになる。

取得対象は、SQL テキスト、バインド変数と値、トランザクション情報など。

ファイルは指定したディレクトリオブジェクト（SELECT * FROM ALL_DIRECTORIES で確認）に作成される。

ディレクトリオブジェクトは空でないと取得できない。

以下のものは取得できない。

・SQL*Loader を使用したダイレクトパスロード

・共有サーバーからのリクエスト

・フラッシュバック問い合わせ

・Oracle Call Interface（OCI）オブジェクトナビゲーション機能の使用

・SQL 以外を使用したオブジェクト型の使用

・Oracle Streams による処理

・PL/SQL 以外のアドバンストキューイング

・アドバンストレプリケーションによる処理（分散トランザクション、リモートの DESCRIBE、COMMIT 操作）

② ワークロードの事前処理

取得したリプレイファイルを変換し、必要なメタデータ生成を行う。

時間が掛かる為、取得したワークロードを転送し、テスト環境で実施するのが望ましい。

[ソフトウェアとサポート] - [データベースリプレイ] - [ワークロードの事前処理] の右側にあるアイコンをクリック

③ ワークロードのリプレイ

リプレイクライアントからリプレイを実行する。$ORACLE_HOME/bin/wrc コマンドとして実行されている。

wrc UID/PASS@接続記述子 ﾊﾟﾗﾒｰﾀ=値 [ﾊﾟﾗﾒｰﾀ=値] ...

ﾊﾟﾗﾒｰﾀ	説明	デフォルト
MODE	REPLAY　：リプレイを行う CALIBRATE ：リプレイに必要なクライアントと CPU 数を見積もる LIST_HOSTS：キャプチャしたホスト名を表示	REPLAY
REPLAYDIR	ワークロード保存ディレクトリ	-
MODE=REPLAY	説明	デフォルト
WORKDIR	DEBUG ファイルの出力先ディレクトリの指定	-
CONNECTION_OVERRRIDE	接続マップの利用 TRUE ：SERVER パラメータ使用 FALSE：接続マップを利用	FALSE
SERIALIZE_CONNECTION	スレッド接続方法 TRUE ：クライアントはシリアルに接続 FALSE：クライアントはオリジナルと同様に接続	FALSE
DEBUG	デバッグ情報の出力 NONE　：出力しない　　FILES ：WORKDIR に出力 STDOUT：標準出力　　BOTH　：FILES＋STDOUT	NONE
MODE=CALIBRATE	説明	デフォルト
PROCESS_PER_CPU	CPU あたりのクライアントプロセス最大数	4
THREADS_PER_PROCESS	1 クライアントプロセスで実行可能な最大スレッド数	50

[ソフトウェアとサポート] - [データベースリプレイ] - [ワークロード・リプレイ] の右側にあるアイコンをクリック

リプライの設定で、以下のオプションを指定する。

ｵﾌﾟｼｮﾝ	説明
synchronization	リプレイの同期化有無。TRUE にすることでワークロード取得時のトランザクションと COMMIT 順序を保持 FALSE は順序を保持しない為、負荷テストを行う場合などに適している
connect_time_scale	セッション開始までの経過時間をワークロード取得時の%で指定。 デフォルトは 100 でワークロード取得時と同じタイミングで実行される
think_time_scale	後続の SQL 実行までのアイドル時間をワークロード取得時の%で指定（デフォルトは 100）
think_time_auto_correct	後続の SQL 実行までのアイドル時間を自動調整するか（デフォルトは TRUE）

wrc コマンドにてクライアントが接続されている状態で、発行ボタンを押すことで、リプレイが実行される。

ワークロードを取得した時と同じタイミング、同時実行性、トランザクションの依存性に従って完全に再現される。

ワークロードリプレイオプションを使用し依存性など変更しリプレイすることも可能。

④ 分析とレポート

[ワークロード・リプレイ] の画面でリプレイを行ったディレクトリオブジェクトを指定することでレポート表示が可能。

・取得時と、リプレイ時の AWR スナップショット

・データベース時間、平均アクティブセッション、ユーザーコールなどの統計情報

・ワークロードの取得時、リプレイ時に発生しなかったエラー、エラータイプの違い

-> プロシージャによるワークロードの取得

DBMS_WORKLOAD_CAPTURE.ﾌﾟﾛｼｰｼﾞｬ･ﾌｧﾝｸｼｮﾝ(ﾊﾟﾗﾒｰﾀ => '値' ...);

ﾌﾟﾛｼｰｼﾞｬ/ﾌｧﾝｸｼｮﾝ	説明
ADD_FILTER	指定したフィルタを追加
DELETE_FILTER	指定したフィルタを削除
START_CAPTURE	ワークロードの取得開始
FINISH_ CAPTURE	ワークロードの取得終了
GET_CAPTURE_INFO	規定されたディレクトリに存在するワークロード取得ファイルから情報を取得し、 ディクショナリ情報（DBA_WORKLOAD_CAPTURES、DBA_WORKLOAD_FILTERS）にインポートし、 適切な DBA_WORKLOAD_CAPTURE.ID（ワークロード取得 ID）を戻す
DELETE_CAPTURE_INFO	指定したワークロード取得 ID に対するディクショナリ情報を削除する
REPORT	ワークロードの取得に関するレポートを戻す
EXPORT_AWR	指定したワークロード取得 ID に関連付けられた AWR をエクスポートする
IMPORT_AWR	指定したワークロード取得 ID に関連付けられた AWR をインポートする

実行例

```
DBMS_WORKLOAD_CAPTURE.START_CAPTURE
(  name              => 'CAPTURE1',
   dir               => 'CAPTURE_DIR',
   default_action    => 'INCLUDE' );
```

キャプチャ取得開始
ワークロードファイル名
ファイル出力先のディレクトリオブジェクト
フィルタ以外の全てを取得。EXCLUDE はフィルタしたもののみ取得

-> プロシージャによるリプレイ実行

DBMS_WORKLOAD_REPLAY.ﾌﾟﾛｼｰｼﾞｬ･ﾌｧﾝｸｼｮﾝ(ﾊﾟﾗﾒｰﾀ => '値' ...);

ﾌﾟﾛｼｰｼﾞｬ/ﾌｧﾝｸｼｮﾝ	説明
CALIBRATE	ワークロードリプレイに必要なリプレイクライアントの数を見積もる
PROCESS_CAPTURE	リプレイの事前処理（リプレイ固有のメタデータ作成）を行う
INITIALIZE_REPLAY	リプレイシステムにデータをロードし、接続文字列などをリセットする
REMAP_CONNECTION	リプレイで使用する接続マッピング情報を設定
PREPARE_REPLAY	リプレイオプションの設定
START_REPLAY	リプレイの開始。起動していたリプレイクライアント（wrc）にリプレイ開始が表示される
CANCEL_REPLAY	実行中のリプレイを取り消す
GET_REPLAY_INFO	指定したディレクトリに存在するワークロードの取得ファイルからリプレイ履歴を戻す
DELETE_REPLAY_INFO	指定したリプレイ ID に対応するディクショナリ情報を削除する
REPORT	ワークロードのリプレイに関するレポートを戻す
EXPORT_AWR	指定したリプレイ ID に関連付けられた AWR をエクスポートする
IMPORT_AWR	指定したリプレイ ID に関連付けられた AWR をインポートする

実行例

```
DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE
(  capture_dir => 'CAPTURE_DIR' );

DBMS_WORKLOAD_REPLAY.INITIALIZE_REPLAY
(  replay_name => 'REPLAY1',
   replay_dir  => 'REPLAY_DIR' );

DBMS_WORKLOAD_REPLAY.PREPARE_REPLAY
(  synchronization => TRUE,
   connect_time_scale  => 100 );
```

ワークロードの事前準備

リプレイデータのロード

リプレイオプションの設定

DBMS_WORKLOAD_REPLAY.**START_REPLAY**	リプレイ開始

-> ワークロードに関するディクショナリ

ビュー	説明
DBA_WORKLOAD_CAPTURES	実行されたワークロードの取得を表示
DBA_WORKLOAD_FILTERS	ワークロード取得用に設定されたフィルタを表示
DBA_WORKLOAD_REPLAY_FILTER_SET	ワークロードリプレイ用に設定されたフィルタを表示
DBA_WORKLOAD_REPLAYS	指定したワークロードのリプレイを表示
DBA_WORKLOAD_REPLAY_DIVERGENCE	ワークロードリプレイ時に発生したエラーに関する情報を表示
DBA_WORKLOAD_CONNECTION_MAP	ワークロードリプレイ用に設定された接続マッピング情報を表示
V$WORKLOAD_REPLAY_THREAD	実行中のリプレイクライアント内のセッションに関する情報を表示

SQL パフォーマンスアナライザ

本番環境で SQL チューニングセット（STS）を取得し、STS をテスト環境で最初のテストを行う。

環境変化（DB アップグレードや、パラメータ変更、統計情報収集など）後に再度テストを行い変化前後で影響を調べる。

SYS ユーザーでは作成した STS をエクスポートはできない為、DBA 権限を付与した専用ユーザーで実行すること。

-> SQL パフォーマンアナライザのワークフロー

① SQL チューニング・セット（STS）の作成（ワークロードの取得）

② テスト環境に STS を転送、インポート

③ SQL パフォーマンス・アナライザの実行

④ 変更前後の結果を比較

① SQL チューニング・セットの作成

［パフォーマンス］－［SQL チューニング・セット］から「作成」をクリックする。

取得するワークロードは、現在実行されているカーソルの SQL 文、AWR に格納された SQL のどちらか選択可能。

含まれる情報

・SQL テキスト、実行頻度（SQL の実行回数）

・実行コンテキスト（バインド変数の値、解析スキーマ、実行計画、実行統計）

② 作成したチューニングセットのエクスポート

作成されたチューニングセットを選択し「エクスポート」ボタンを押下しダンプファイルを出力する。

③ テスト環境に STS を転送、インポート

②でエクスポートしたファイルを転送し［パフォーマンス］－［SQL チューニング・セット］から「インポート」クリック

④ SQL パフォーマンス・アナライザの実行

［パフォーマンス］－［SQL パフォーマンス・アナライザ］より 以下のテストが可能。

・データベースアップグレードによる影響確認

・初期化パラメータの変更による SQL 文のパフォーマンス変化

・Exadata を導入した場合の動作シミュレーション（R2 で追加）

・ガイド付きワークフロー（その他の変更に対応するカスタムテスト用のウィザード）

作成方法のプルダウンから「SQL 実行」と「計画の作成」のどちらかを選択できる。

SQL 実行 は実際に SQL を実行(UPDATE/DELTE 文は解析と論理 I/O のみ取得し、INSERT と DDL 文は対象外)した統計を比較。

計画の作成 は実行計画のみの比較となる為、精度は下がるが実行時間が短い。

⑤ 更前後の結果を比較

④を実行することで、変更前と、変更後の STS を実行し、その比較結果のレポートを確認することが出来る。

影響の大きい上位 10 位までの SQL 一覧が表示される。

-> プロシージャによる SQL チューニング･セットの取得

DBMS_SQLTUNE.ﾌﾟﾛｼｰｼﾞｬ･ﾌｧﾝｸｼｮﾝ(ﾊﾟﾗﾒｰﾀ => '値' ...);

ﾌﾟﾛｼｰｼﾞｬ/ﾌｧﾝｸｼｮﾝ	説明
CREATE_SQLSET	SQL チューニング・セット・オブジェクトを作成（空のチューニング・セット）
LOAD_SQLSET	選択した SQL のセットを SQL チューニング・セットに入力する
CAPTURE_CURSOR_CACHE_SQLSET	指定した時間に実行された SQL のセットを SQL チューニング・セットに入力する
CREATE_STGTAB_SQLSET	SQL チューニング・セットのインポートおよびエクスポートに使用するステージング表を作成
PACK_STGTAB_SQLSET	SYS スキーマからステージング表にチューニング・セットをコピー

実行例

DBMS_SQLTUNE.**CREATE_SQLSET** (sqlset_name => 'DBMS_SQLSET', sqlset_owner => 'SCOTT');	SQL チューニング･セットの作成（空）
DBMS_SQLTUNE.**CREATE_STGTAB_SQLSET** (table_name => 'TUNETBL', schema_name => 'SCOTT');	チューニングセットをインポート・エクスポートする為のテーブル作成
DBMS_SQLTUNE.**PACK_STGTAB_SQLSET** (sqlset_name => 'DBMS_SQLSET', staging_table_name => 'TUNETBL', staging_schema_owner => 'SCOTT') ;	SQL チューニング･セットをテーブルにコピーする

-> プロシージャによる SQL パフォーマンスアナライザ

DBMS_SQLPA.ﾌﾟﾛｼｰｼﾞｬ･ﾌｧﾝｸｼｮﾝ(ﾊﾟﾗﾒｰﾀ => '値' ...);

ﾌﾟﾛｼｰｼﾞｬ/ﾌｧﾝｸｼｮﾝ	説明
CREATE_ANALYSIS_TASK	１つ以上の SQL 文を処理および分析するアドバイザ・タスクを作成する
EXECUTE_ANALYSIS_TASK	CREATE_ANALYSIS_TASK で作成した分析タスクを実行しワークロードのパフォーマンスを作成 パラメータの変更前と、変更後のそれぞれで REPORT_ANALYSIS_TASK を実行し比較する execution_type => 'compare performance' を入れると（デフォルトでは）最後の 2 回の実行計画を経過時間（elapsed_time メトリック）で比較する
REPORT_ANALYSIS_TASK	分析タスクの結果を表示

SQL 計画管理

各 SQL 実行すると実行計画を SQL 計画履歴として、SYSAUX 領域内の「SQL 管理ベース（SMB)」に格納する。

計画履歴から計画ベースラインを作成（自動ロード/手動ロード）することで、実行計画を制御させることができる。

SQL 文が実行されるとオプティマイザが実行計画を作成し、計画履歴内の計画ベースラインに含まれているか確認する。

SQL 計画履歴のうち「有効」且つ「承認」されている計画が、計画ベースラインに含まれる。
計画ベースラインに含まれていれば、その実行計画を使用し、含まれてなければ実行時に作成された実行計画を使用する。

SQL 計画管理を利用するには OPTIMIZER_USE_SQL_PLAN_BASELINES パラメータが TRUE（デフォルト）になってること。
SQL 計画管理を利用しない場合は、常に実行時の実行計画が使用される。

-> SQL 計画管理による実行計画の選択

DBA_SQL_PLAN_BASELINES ビューで実行計画属性を確認可能。

列名	説明
SQL_HANDLE	一意の SQL 識別子
SQL_TEXT	対象となる SQL テキスト
PLAN	一意のプラン識別子
ORIGIN	実行計画の作成方法 MANUAL-LOAD ：手動ロード　MANUAL-SQLTUNE：SQL チューニングアドバイザによる SQL プロファイルの手動適用 AUTO-CAPTURE：自動ロード　AUTO-SQLTUNE　：SQL チューニングアドバイザによる SQL プロファイルの自動適用
ENABLED	実行計画が有効か
ACCEPTED	実行計画が承認済みか　※ENABLED と ACCEPTED が YES の場合のみ実行計画は使用可能
FIXED	ENABLED と ACCEPTED が YES が複数ある中で、特定の実行計画を使用させたい場合に YES とする ENABLED と ACCEPTED が YES が 5 つ、FIXED が YES が 3 つの場合、3 つの中で一番コストの低い実行計画を使用する

-> SQL 計画ベースラインに対する実行計画のロード（計画ベースラインに登録）

実行計画の自動ロード

OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES パラメータを TRUE（デフォルトは FALSE）にする。
繰り返し(2 回以上)実行される SQL 文に対する計画履歴が自動的に作成され計画ベースラインに自動的に追加される。
一番最初に登録された実行計画は ENABLED と ACCEPTED が YES となり、2 個目以降は ACCEPTED が NO になる。
計画ベースラインの展開（後述）を行わない限り、一番最初の実行計画を使い続ける。

実行計画の手動ロード

DBMS_SPM パッケージを使用する。以下の 3 つの方法がある。ENABLE と ACCEPTED は YES で FIXED は NO になる。

ﾌﾟﾛｼｰｼﾞｬ/ﾌｧﾝｸｼｮﾝ	説明
LOAD_PLANS_FROM_CURSOR_CACHE	現在のカーソルキャッシュ上に存在する 1 つ以上の実行計画をロードする
LOAD_PLANS_FROM_SQLSET	SQL チューニングセット（STS)に保存されている実行計画をロードする
XXX_STGTAB_BASELINE	別のデータベースの実行計画をロードする CREATE_STGTAB_BASELINE プロシージャでステージング表を作成し、 PACK_STGTAB_BASELINE ファンクションを使用し計画ベースラインをステージング表にパックする。 ステージング表をエクスポート/インポートしターゲット DB へ転送後、 UNPACK_STGTAB_BASELINE ファンクションでアンパックすることで計画ベースラインを転送する

実行例（LOAD_PLANS_FROM_CURSOR_CACHE）

```
declare
plan pls_integer;
begin
plan := DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE(sql_id=>'bc0bhrkm6256g');
end;
/
```

型の変数を宣言

カーソルキャッシュにある実行計画をロード

-> 計画ベースラインの展開

自動ロードでは最初にロードされた実行計画は、ENABLE=YES、ACCEPTED=YES だが、

後からロードされた実行計画は ACCEPTED=NO となっている。DBMS_SPM パッケージを使うことで明示的に属性変更が可能。

ﾌﾟﾛｼｰｼﾞｬ/ﾌｧﾝｸｼｮﾝ	説明
EVOLVE_SQL_PLAN_BASELINE	計画ベースラインに登録されている未承認の実行計画と、承認済みの実行計画を比較する 実行計画が優れていた場合は、承認済みになる（commit パラメータを YES にしてる場合）
ALTER_SQL_PLAN_BASELINE	実行計画属性（ENABLE、ACCEPTED、FIXED、AUTOPURGE など）を明示的に変更する
DROP_SQL_PLAN_BASELINE	実行計画履歴を削除する

実行例（EVOLVE_SQL_PLAN_BASELINE）

```
variable report CLOB
BEGIN
    :report := DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE(
    sql_handle => 'SYS_SQL_39832bbd98dd2');
END;
/

set long 50000 longchunk 120
print report
～～省略
Number of SQL plan baselines evolved : 1
```

```
CLOB 型の変数を宣言

計画ベースラインを検証する
SQL$TEXT や DBA_SQL_PLAN_BASELINES テーブルにて確認




戻り値の結果を表示

evolved が 進化した実行計画の数
```

実行例（ALTER_SQL_PLAN_BASELINE）

```
variable cnt NUMBER
BEGIN
    :cnt := DBMS_SPM.ALTER_SQL_PLAN_BASELINE(
    sql_handle => 'SYS_SQL_39832bbd98dd2',
    plan_name  => 'SYS_SQL_PLAN_23089feeerwe',
    attribute_name  => 'FIXED',
    attribute_value => 'YES');
END;
/
```

```
NUMBER 型の変数を宣言

実行計画属性を変更する


FIXED 属性を
YES に変更する
```

-> SQL 管理ベース（SMB）の保守

SQL 管理ベースは SYSAUX 表に格納され、使用領域が毎週チェックされる。

10%を超過するとアラートログに警告が書き込まれ、下回るまで毎週警告が記録される。

53 週（1 年）を超過すると毎週実行される消去タスクによって自動的に消去される。

DBA_SQL_MANAGEMENT_CONFIG ビューの SPACE_BUDGET_PERCENT で閾値、PLAN_RETENTION_WEEK で削除閾値の確認が可能。

閾値は DBMS_SPM.CONFIGURE プロシージャで、変更可能。（変更可能な値は 1～50%、5～523 週）

（例）exec DBMS_SPM.CONFIGURE('SPACE_BUDGET_PERCENT',20)　※削除閾値のパラメータは PLAN_RETENTION_WEEKS

■インテリジェントインフラストラクチャ

自動 SQL チューニング

自動化メンテナンスタスクによって SQL チューニングアドバイザが自動的に実行されるようになった。

自動 SQL チューニングの実行により、SQL プロファイル、オプティマイザ統計収集、索引の作成、SQL 文の再構築の推奨事項が生成されるが、自動的に適用はされない。適用は手動で行う。SQL プロファイルのみ自動適用させることは可能。

SQL プロファイルを適用すると ENABLEED、ACCEPTED が YES、FIXID が NO で計画ベースラインにロードされる。

自動化メンテナンスタスクによって自動 SQL チューニングが実行するとデフォルトで 1 時間を限度に次の処理を順次実行。

自動ワークリポジトリ（AWR）から候補 SQL を識別

負荷が高く繰り返し実行されている SQL 文をチューニング候補リストを作成する。

対象外は、DDL、DML、再帰的 SQL、パラレル問い合わせ、同時実行を原因としたパフォーマンス問題のある SQL 文。
対象外の SQL 文も手動で SQL チューニングアドバイザを実行してチューニングすることは可能。

各 SQL 文を個別にチューニング

候補となった SQL 文は CPU 時間と I/O 時間の合計からパフォーマンス影響が大きいと判断され、チューニング順序を決定する。
個々のチューニングは次のことが実行される。

① SQL チューニングアドバイザの実行

候補の SQL 文を 1 つずつ SQL チューニングアドバイザを使用してチューニングする。(推奨事項の作成)

② SQL プロファイル(クエリの実行計画を制御する為のヒントの集まり)のテスト

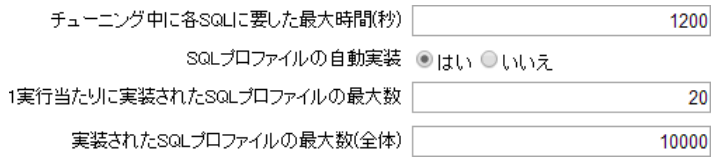
SQL プロファイルが存在する場合、使用した場合と使用しない場合の実行テストを行う。

③ SQL プロファイルの適用(自動適用が有効な場合)

パフォーマンス(CPU 時間と I/O 時間の合計)が 3 倍以上向上する場合、SQL プロファイルを自動的に適用する。

-> 自動 SQL チューニングの設定

[サーバー] - [自動化メンテナンスタスク] にアクセスし [構成] ボタンをクリックし設定ページにアクセスする。



DBMS_SQLTUNE.SET_TUNING_TASK_PARAMETER プロシージャでも上記設定可能

パラメータ	説明
LOCAL_TIME_LIMIT	SQL 文ごとのタイムアウト値
ACCEPT_SQL_PROFILES	SQL プロファイルの自動実装有無(デフォルトは FALSE)
MAX_SQL_PROFILES_PER_EXEC	1 回のチューニング(デフォルトで 1 日に 1 回)で作成可能な SQL プロファイル最大数(初期値 20)
MAX_AUTO_SQL_PROFILES	自動 SQL チューニングで適用可能な SQL プロファイル最大数
EXECUTION_DAYS_TO_EXPIRE	アドバイザフレームワークにてタスク履歴を保存しておく日数

実行例

```
DBMS_SQLTUNE.SET_TUNING_TASK_PARAMETER (
    task_name => 'SYS_AUTO_SQL_TUNING_TASK',
    parameter => 'ACCEPT_SQL_PROFILES',
    value     => 'FALSE');
```

SQL プロファイル自動実装を無効にする
予約された自動チューニングタスク SYS_AUTO_SQL_TUNING_TASK を指定(固定)

-> 自動 SQL チューニングの結果レポート

EM で表示させる他にも、DBMS_SQLTUNE.REPORT_AUTO_TUNING_TASK ファンクションを利用して出力させることも可能。

実行例

```
select execution_name,execution_start,
execution_last_modified from dba_advisor_executions;

BEGIN
DBMS_ADVISOR.CREATE_FILE(
    DBMS_SQLTUNE.REPORT_AUTO_TUNING_TASK(
      begin_exec => 'EXEC_1_21',
      end_exec   => 'EXEC_1_22',
      section    => 'FINDINGS'),
   'DATA_PUMP_DIR','auto_tune.txt');
END;
/
```

実行されたタスクの時間を確認する
execution_name の後続のファンクションのパラメータに指定する

ファンクションの結果をファイル出力する

begin_exec と end_exec を NULL 指定した場合、
最後に実行されたタスクの結果が表示される
レポート取得セクションを「FINDINGS」に限定する
CREATE DIRECTORY 名前 AS 'パス'; で作成した名前のパスに作成
※DBA_DIRECTORYS で設定されてるディレクトリの一覧が確認できる

自動化メンテナンスタスク

データベーススケジューラから提供されるメンテナンスウィンドウがオープンされると実行されるタスクのこと。

オプティマイザ集計、セグメントアドバイザ、自動 SQL チューニングが含まれている。

MAINTENANCE_WINDOW_GROUP ウィンドウグループに対応付けられている。

11g では、WEEKNIGHT_WINDOW と、WEEKEND_WINDOW の 2 つだったが、11g では曜日毎に日次ウィンドウで定義されている。

月～金 22:00-26:00、土～日 06:00-26:00 で設定されている。

名前 **MAINTENANCE_WINDOW_GROUP**
有効 **TRUE**

メンバー

名前	リソース・プラン	有効	次のオープン日	終了日	期間(分)	説明
THURSDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	TRUE			240	Thursday window for maintenance tasks
FRIDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	TRUE			240	Friday window for maintenance tasks
SATURDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	TRUE			1200	Saturday window for maintenance tasks
SUNDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	TRUE			1200	Sunday window for maintenance tasks
MONDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	TRUE			240	Monday window for maintenance tasks
TUESDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	TRUE			240	Tuesday window for maintenance tasks
WEDNESDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	TRUE			240	Wednesday window for maintenance tasks

DEFAULT_MAINTENANCE_PLAN は、ORA$AUTOTASK_SUB_PLAN が含まれ、自動化メンテナンスと対応付けられている。

リソース割当て

グループ/サブプラン	レベル1	レベル2	レベル3	レベル4	レベル5	レベル6	レベル7	レベル8
ORA$AUTOTASK_SUB_PLAN		25						
ORA$DIAGNOSTICS		5						
OTHER_GROUPS		70						
SYS_GROUP	75							

ORA$AUTOTASK_SUB_PLAN のリソース表示

リソース割当て

グループ/サブプラン	レベル1	レベル2	レベル3	レベル4	レベル5	レベル6	レベル7	レベル8
ORA$AUTOTASK_HIGH_SUB_PLAN		100						
ORA$AUTOTASK_MEDIUM_GROUP			100					
ORA$AUTOTASK_URGENT_GROUP	100							

ORA$AUTOTASK_HIGH_SUB_PLAN リソース表示

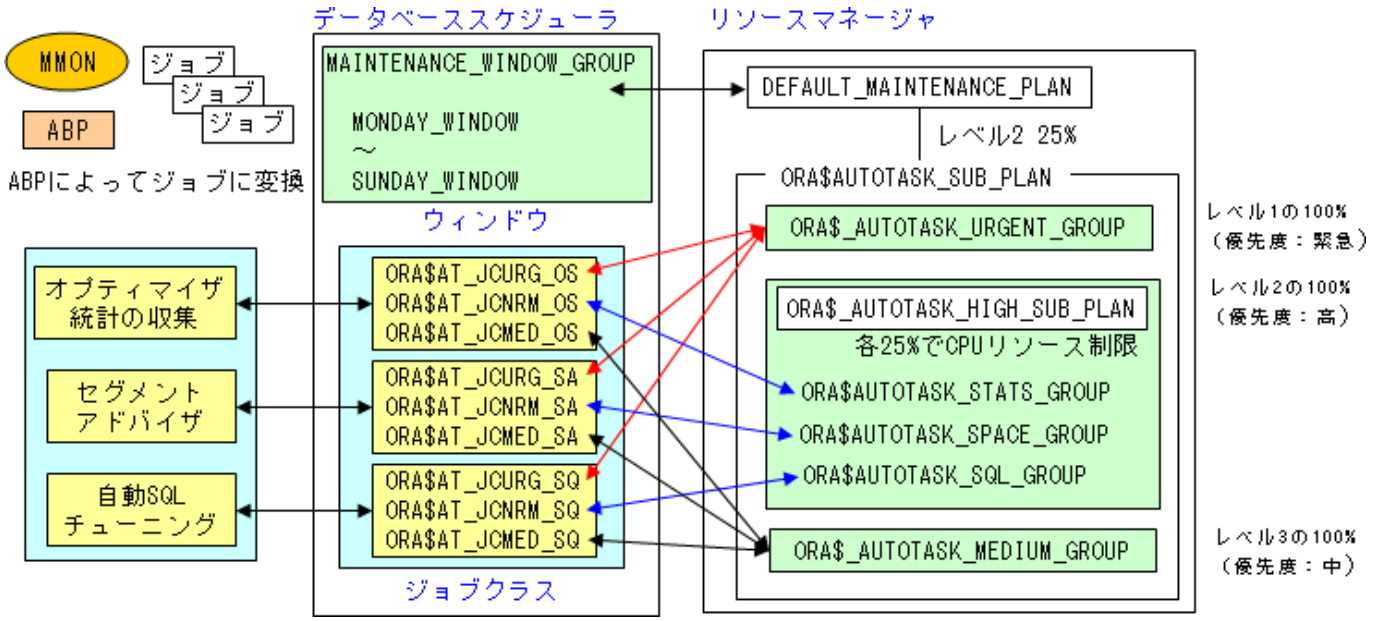
リソース割当て

グループ/サブプラン	割合
ORA$AUTOTASK_HEALTH_GROUP	25
ORA$AUTOTASK_SPACE_GROUP	25
ORA$AUTOTASK_SQL_GROUP	25
ORA$AUTOTASK_STATS_GROUP	25

コンシューマグループと自動化メンテナンスタスクの対応付けはスケジューラのジョブクラスで実装。

○	ORA$AT_JCURG_OS	オプティマイザ統計の収集	1000000	ORA$AUTOTASK_URGENT_GROUP
○	ORA$AT_JCNRM_OS		1000000	ORA$AUTOTASK_STATS_GROUP
○	ORA$AT_JCMED_OS		1000000	ORA$AUTOTASK_MEDIUM_GROUP
○	ORA$AT_JCURG_SA	セグメントアドバイザ	1000000	ORA$AUTOTASK_URGENT_GROUP
○	ORA$AT_JCNRM_SA		1000000	ORA$AUTOTASK_SPACE_GROUP
○	ORA$AT_JCMED_SA		1000000	ORA$AUTOTASK_MEDIUM_GROUP
○	ORA$AT_JCURG_SQ	自動SQLチューニング	1000000	ORA$AUTOTASK_URGENT_GROUP
○	ORA$AT_JCNRM_SQ		1000000	ORA$AUTOTASK_SQL_GROUP
○	ORA$AT_JCMED_SQ		1000000	ORA$AUTOTASK_MEDIUM_GROUP

自動化メンテナンスタスクとの対応付け



自動化メンテナンスタスクは「Autotask Background Process（ABP)」によって AUTOTASK ジョブに変換される。

よって永続的なジョブは存在しない。ABP は MMON によってメンテナンスウィンドウの開始時に実行される。

ジョブの実行ログは DBA_SCHEDULER_JOB_LOG ビューで確認可能。

-> 自動化メンテナンスタスクの制御

ジョブとして制御されていない為、DBMS_SCHEDULER パッケージでの制御は出来ない。

EM（[サーバー] - [自動化メンテナンスタスク] の 「構成」）か、DBMS_AUTO_TASK_ADMIN パッケージを使用する。

DBMS_AUTO_TASK_ADMIN は、ENABLE プロシージャと DISABLE プロシージャで有効無効を切り替える。

パラメータ	説明
CLIENT_NAME	対象となるクライアント名（タスク） auto optimizer stats collection　オプティマイザ統計の収集 auto space advisor　セグメントアドバイザ sql tuning advisor　自動 SQL チューニングアドバイザ
OPERATION	対象となる操作名（ジョブのステータス）NULL または DBA_AUTOTASK_OPERATION ビューの OPERATION_NAME 列
WINDOW_NAME	対象となるウィンドウ名。NULL の場合全てのウィンドウが対象になる（OPERATION が NULL 以外の場合は無視される）

自動化メンテナンスタスク自体の有効無効はパラメータを何も入れずにプロシージャを実行する。

自動化メンテナンスの関連ビュー

ビュー	説明
DBA_AUTOTASK_CLIENT	過去 7 日間と 30 日間で集計された自動化メンテナンスタスクの統計データ
DBA_AUTOTASK_CLIENT_HISTORY	メンテナンスウィンドウ毎の自動化メンテナンスタスクの実行回数と履歴を表示
DBA_AUTOTASK_CLIENT_JOB	現在実行中の自動化メンテナンスタスクを表示
DBA_AUTOTASK_JOB_HISTORY	自動化メンテナンスタスクのジョブ履歴を表示
DBA_AUTOTASK_OPERATION	自動化メンテナンスタスクに関する属性（操作）を表示
DBA_AUTOTASK_SCHEDULE	今後 32 日間のメンテナンスウィンドウのスケジュールを表示
DBA_AUTOTASK_TASK	自動化メンテナンスタスクの各種統計データ
DBA_AUTOTASK_WINDOW_CLIENTS	メンテナンスウィンドウ毎のステータスを表示
DBA_AUTOTASK_WINDOW_HISTORY	メンテナンスウィンドウの実行履歴を表示

AWR ベースライン

-> 概要

ベースラインは特定期間の AWR スナップショットのこと。

現在のパフォーマンスと比較することで、メトリックのデータに問題が出ているか確認できる。

サーバー生成アラートを生成するためのしきい値にベースラインを使用することも可能。

ベースラインの種類	説明
固定ベースライン	過去の特定期間の AWR データに対応する
ベースラインテンプレート	将来の特定期間の AWR データに対応する。単一期間または繰り返し期間を指定できる
変動ウィンドウベースライン	AWR 保存期間内に存在する AWR データに対応する。直近データとの比較に便利

-> 変動ウィンドウベースライン

SYSTEM_MOVING_WINDOW という名前の変動ウィンドウベースラインがデフォルトで定義されている。

過去 8 日間の AWR に対応。期間は AWR 保存期間以下の値にする必要がある。

しきい値に使用するベースラインで必要な統計計算は（BSLN_MAINTAIN_STATS_JOB ジョブで定義）1 週間に 1 度実行される。

期間を変更するには EM か、パッケージで行う。

［サーバー］－［AWR ベースライン］－「SYSTEM_MOVING_WINDOW」のラジオボタンを選択し「編集」

```
DBMS_WORKLOAD_REPOSITORY.MODIFIY_BASELINE_WINDOW_SIZE( window_size => 期間);
```

-> ベースラインテンプレート

単一（期間指定）または繰り返し（毎週月曜日 AM07:00～12:00 など）で指定し、指定した時刻になると終了し、

ベースラインが作成される。ベースラインの保存期間の設定も可能で期間が過ぎたら自動的に削除される。

DBMS_WORKLOAD_REPOSITORY パッケージによるベースラインテンプレートの作成

プロシージャ	説明
CREATE_BASELINE	固定ベースラインの作成（過去の期間に対するベースライン）
CREATE_BASELINE_TEMPLATE	ベースラインテンプレートの作成（未来の期間に対するベースライン）

実行例

```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE (
    start_time     => TO_DATE('20140822','YYYYMMDD'),
    end_time       => TO_DATE('20140922','YYYYMMDD'),
    baseline_name => 'BSLINE',
    template_name => 'BALSELINE0822',
    expiration     => NULL);
```

```
単一の AWR ベースライン
期間は 2014 年 08 月 22 日　から
       2014 年 09 月 22 日　まで
ベースライン名（後ろに時間が付与される）

ベースラインの保存期間を指定（NULL または省略は無期限）
```

```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE (
    day_of_week    => 'MONDAY',
    hour_in_day    => 9,
    duration       => 3,
    start_time     => TO_DATE('20140822','YYYYMMDD'),
    end_time       => TO_DATE('20140922','YYYYMMDD'),
    baseline_name_prefix => 'BSLINE_MONDAY',
    template_name => 'BALSELINE0822',
    expiration     => NULL);
```

```
繰り返しの AWR ベースライン
月曜日
09:00～
3 時間取得
期間は 2014 年 08 月 22 日　から
       2014 年 09 月 22 日　まで
ベースライン名（後ろに時間が付与される）

ベースラインの保存期間を指定（NULL または省略は無期限）
```

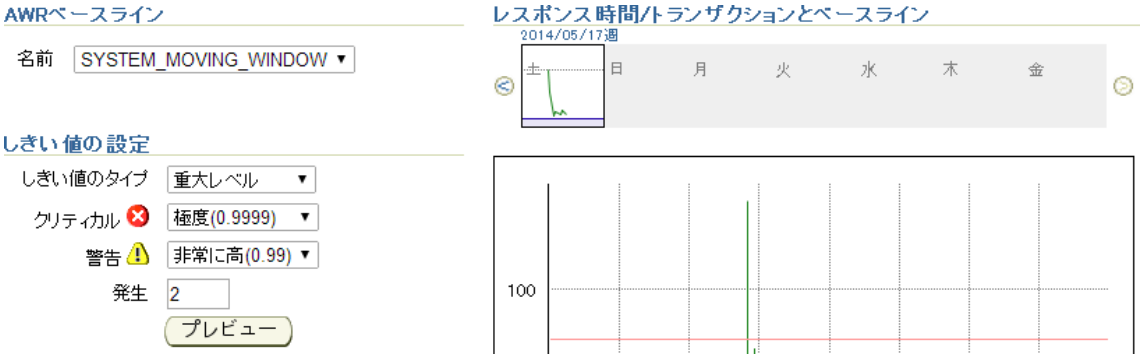
-> 適応しきい値の設定

10g のサーバー生成アラートはメトリックに固定しきい値を設定することでアラート、超過した時にアラート生成していた。

※EM － 関連リンクの［メトリックとポリシー設定］で設定

10gR2 は計算されたベースラインを基にパフォーマンスの乖離をチェックする適応しきい値を設定できるようになった。

ベースラインの計算は、変動ウィンドウベースラインでは BSLN_MAINTAIN_STATS_JOB で 1 週間に 1 度行われるが、
静的ベースラインの場合は手動で計算する必要がある。
手動計算は［AWR ベースライン］－［アクション］リストから「スケジュール統計の計算」を選択し「実行」する。
適応しきい値を設定するには［AWR ベースライン］－［ベースラインメトリックしきい値］にアクセスする。
AWR ベースラインのプルダウンから適用するベースラインの変更も可能。



しきい値タイプ	説明
重大レベル	しきい値を超過する頻度が、どの程度異常かを設定する。何回中、何回超過したかを 4 レベルで分類。 設定できる値は高（5/100）、非常に高い（1/100）、重度（1/1000）、極度（1/10000）
最大パーセント	ベースラインのデータの最大に対する%を指定し、実際の値がその値を越えるとサーバー生成アラートを生成する。 ベースラインのピーク値やピーク値を越えているメトリックを判断する）

リソースマネージャの拡張機能

I/O を測定する機能や I/O 統計の新しい統計レベルが追加されている。

-> I/O 測定

［パフォーマンス］－［I/O］サブタブを表示し「I/O 測定」をクリックする。
I/O メトリックとして情報を収集する機能が追加されストレージレベルの情報が収集できるようになった。

メトリック	説明
IOPS	1 秒あたり処理できる I/O 数。ディスクの回転率やシーク時間が早いほど、ストレージアレイならばディスク数が多いほど、多くの I/O が可能。OLTP ではランダムな I/O が必要な為、IOPS が高いことが望ましい
MBPS	1 秒あたり転送可能なビット（100 万ビット単位）。I/O チャンネルの帯域幅が大きい程、多くの転送が可能 DWH や OLTP アプリは高スループットと連続した I/O が必要な為、MBPS が高いことが望ましい
I/O 待機時間	ディスク上の特定箇所にアクセスするのに必要な時間

DBMS_RESOURCE_MANAGER パッケージによる測定

```
VARIABLE max_iops NUMBER                          戻り値を入れる変数を定義
VARIABLE max_mbps NUMBER
VARIABLE wait_time NUMBER

BEGIN
DBMS_RESOURCE_MANAGER.CALIBRATE_IO(               ストレージの I/O 機能を検証する（結果は　ビューに格納される）
    num_physical_disks => 1,                      データベース用のディスク数
    max_latency        => 10,                     I/O リクエストあたりの最大待機許容時間（ミリ秒）
    max_iops           => :max_iops,              VARIABLE で宣言した変数を指定
    max_mbps           => :max_mbps,              VARIABLE で宣言した変数を指定
    actual_latency     => :wait_time);            VARIABLE で宣言した変数を指定
```

```
END;
/
```

I/O 測定するには非同期 I/O が有効（filesystemio_options 初期化パラメータが ASYNCH か STEALL）且つ
時間統計収集が有効（timed_statistics=TRUE）であること。同時に実行しないこと。そうしないとエラーになる。

-> I/O メトリック

I/O に関する統計は 3 つの次元から確認出来る統計情報が追加されている。
［パフォーマンス］－［I/O］タブを選択「I/O ブレークダウン」のチェックボックスで選択する。

データベースコンポーネント単位

EM の「I/O ファンクション」または V$IOSTAT_FUNCTION ビューで Oracle 機能が使用する I/O の統計を表示する。
RMAN、XDB、DataPump、DirectReads、DirectWrites、BufferCacheReads、LGWR、ARCH、DBWR、StreamsAQ、Recovery、Others
の 12 カテゴリに分類されている。

ファイルタイプ単位

EM の「I/O タイプ」または V$IOSTAT_FILE ビューで大規模、小規模ファイルに対する I/O 統計を表示する。
データファイル以外に、制御ファイル、REDO ログファイル、アーカイブログファイル、DataPump のダンプファイル、
RMAN によるバックアップファイルなども含まれる。

コンシューマグループ単位

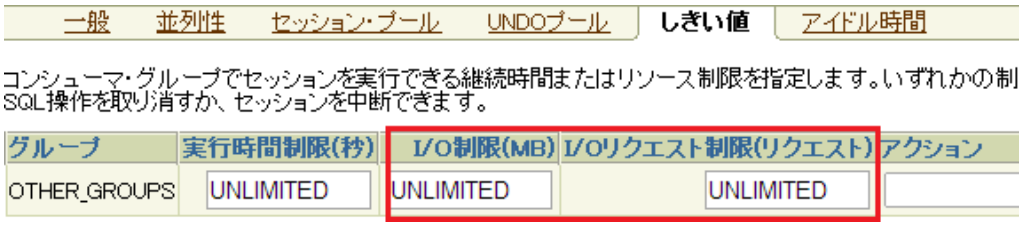
EM の「コンシューマグループ」または V$IOSTAT_CONSUMER_GROUP ビューでリソースマネージャが有効な場合に
コンシューマグループが使用する I/O の統計情報を表示する。

-> I/O リソースしきい値

新しいディレクティブとして I/O リソース制限が追加された。

ディレクティブ	説明
I/O 制限（MB）	セッションで発行できる I/O の量を指定。指定した量を超えるとアクションが実行される
I/O リクエスト制限	セッションで発行できる I/O リクエスト量を指定。指定した量を超えるとアクションが実行される

［サーバー］－［リソース］で新規にリソースプランを作成するか、既存プランの変更ページでしきい値タブを使用する。



DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE プロシージャで、しきい値の作成

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    plan              => 'PLAN1',          リソースプラン名
    group_or_subplan  => 'LOW_GROUP',      コンシューマ・グループ名またはサブプラン名
    switch_group      => 'SYS_GROUP',      切り替え先グループ
    switch_io_megabytes => 100,            I/O 量 100MB で切り替え
    switch_io_reqs    => 1000              I/O リクエスト回数 1000 回で切り替え
    switch_for_call   => TRUE);            処理の完了後、元のグループにスイッチバックさせる
```

スケジューラの拡張機能

10g でサポートされたスケジューラジョブを「標準ジョブ」と呼ぶ。11g では「軽量ジョブ」がサポートされた。

-> 軽量ジョブの概要

標準ジョブではジョブクラスによるグループ化、ジョブ定義に実行内容を組み込む、自己完結型のジョブを作成できる。

DB オブジェクトを作成する為、REDO ログ生成が必要な為、作成と削除を繰り返すようなジョブには不向き。

軽量ジョブは DB オブジェクトとして作成されない為、作成/削除のオーバーヘッドが少ない。

軽量ジョブの作成はテンプレート（PL/SQL やストアドプロシージャで作成されたスケジューラプログラム）が必要。

条件	標準ジョブ	軽量ジョブ
オブジェクトタイプ	スキーマオブジェクト	非スキーマオブジェクト
ジョブ作成/削除の負荷	REDO ログ作成が必要	REDO ログの作成が不要
ジョブ実行のためのセッション作成時間	通常のセッションと同様	標準ジョブより短い
ジョブ実行に必要なメタデータと実行時のデータ	DB で管理	ディスクを使用（負荷が掛かる）
自己完結型のジョブ作成	可能	不可能（テンプレート：プログラムが必須）
ジョブ属性（最大失敗回数など）	全て設定可能	殆ど設定不可（引数の設定は可能）
ジョブの権限設定	全て設定可能	設定不可（テンプレート側で行う）

-> 軽量ジョブの作成

DBMS_SCHEDULER.CREATE_JOB プロシージャの「JOB_STYLE」パラメータに「LIGHTWEIGHT」を指定する。(EM から作成不可)

軽量ジョブの作成例

```
CREATE OR REPLACE PROCEDURE truncate_tab
    ( table_name IN VARCHAR2 )
AUTHID CURRENT_USER AS
BEGIN  EXECUTE IMMEDIATE
    'TRUNCATE TABLE ' || table_name; END;
/
```

プログラムから使用するプロシージャの作成

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM(
    program_name   => 'TRUNCATE_PROG',
    program_action => 'TRUNCATE_TAB',
    program_type   => 'STORED_PROCEDURE',
    number_of_arguments => 1);
DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT(
    program_name   => 'TRUNCATE_PROG',
    argument_name  => 'TABLE_NAME',
    argument_position => 1,
    argument_type  => 'VARCHAR2',
    default_value  => NULL);
DBMS_SCHEDULER.ENABLE('TRUNCATE_PROG'); END;
/
```

プログラムの作成
作成するプログラム名
作成済みのプロシージャを指定
プログラムタイプはストアドプロシージャ

プログラムの属性定義
属性を定義するプログラム名
引数に割り当てる名前（プログラム内で一意であること）

プログラムの有効化

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB(
    job_name      => 'TRUNCATE_JOB',
    program_name => 'TRUNCATE_PROG',
    job_class     => 'DEFAULT_JOB_CLASS',
    job_style     => 'LIGHTWEIGHT',
    repert_interval => 'FREQ=DAILY;BYHOUR=10');
DBMS_SCHEDULER.JOB_ARGUMENT_VALUE(
    job_name      => 'TRUNCATE_JOB',
    argument_position => 1,
    argument_value     => 'SCOTT.TEST');
DBMS_SCHEDULER.ENABLE('TRUNCATE_PROG'); END;
/
```

ジョブの作成（job_style で標準ジョブか軽量ジョブかを定義）

CREATE_PROGRAM で作成したプログラム名を指定
このジョブに関連付けさせるジョブクラス
軽量ジョブを作成する
ジョブを繰り返す間隔。カレンダ式または PL/SQL 式を使用

引数を SCOTT.TEST にする
ジョブの有効化

exec DBMS_SCHEDULER.**RUN_JOB**('TRUNCATE_JOB')	作成した軽量ジョブを実行する

-> ジョブの配列

ジョブの配列を作成可能になった。単一ジョブを格納する JOB 型、配列として JOB_ARRAY 型。

格納されたジョブを実行する DBMS_SCHEDULER.CREATE_JOBS プロシージャが追加されている。

job_array に JOB_ARRAY 型の配列を指定。commit_semantics に ジョブ失敗時のトランザクション制御方法を記述する。

commit_semantics	説明
STOP_ON_FIRST_ERROR	エラーが発生した時点で処理終了。エラーまでの処理はコミットする
TRANSACTIONAL	エラーが発生した時点で処理終了。エラーまでの処理はロールバックする
ABSORB_ERRORS	エラーを無視して残りのジョブを実行する。エラー以外のジョブを最後にコミットする

■パフォーマンス関連の新機能

ADDM の拡張機能

-> RAC 用の ADDM 機能の拡張

10g の ADDM は、いずれかのインスタンスで AWR の作成が指示されると、各インスタンスの MMON によって AWR が作成され、各インスタンスで ADDM が実行されていた。その結果、ADDM の推奨事項は個々のインスタンス固有のものが多かった。

11g では RAC 環境において以下の ADDM タイプが実行できるようになった。

ADDM のタイプ	説明
インスタンス ADDM	インスタンス固有の診断を実行。10g の ADDM と同じ
データベース ADDM	すべてのインスタンスに関する診断を実行。グローバルリソース、ネットワーク待機時間の問題、特定インスタンスにおけるレスポンス時間の偏りなどを分析できる
部分分析 ADDM	指定したインスタンス（複数可能）に関する診断を実行

ADDM の実行は、[アドバイザ・セントラル] - [ADDM] か、DBMS_ADDM パッケージを使用する。

ﾌﾟﾛｼｰｼﾞｬ/ﾌｧﾝｸｼｮﾝ	説明
ANALYZE_DB	データベース ADDM を実行
ANALYZE_INST	インスタンス ADDM を実行
ANALYZE_PARTIAL	部分分析 ADMM を実行。INSTANCE_NUMBERS パラメータでインスタンスを指定
DELETE	ADDM タスクを削除
GET_REPORT	ADDM タスクの結果を取得（CLOB 型の値で戻される）

データベース ADDM タスクの実行

```
var tname VARCHAR2(60)                          作成するタスクの名前
BEGIN
:tname := 'DB_ANALYZE'
DBMS_ADDM.ANALYZE_DB(
    task_name =>    => :tname,
    begin_snapshot => 108,                      スナップ ID 108 から
    end_snapshot   => 110);                     スナップ ID 110 までを診断対象とする
END;
/

SET LONG 100000
SET PAGESIZE 50000
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;  結果を出力する
```

-> ディレクティブの制御

ADDM の実行による推奨結果にフィルタを追加できるようになった。(ディレクティブと呼ぶ)

ディレクティブを追加するには、DBMS_ADDM パッケージを使用する。

ﾌﾟﾛｼｰｼﾞｬ/ﾌｧﾝｸｼｮﾝ	制限(非表示)を追加/削除するディレクティブ
INSERT_FINDING_DIRECTIVE	特定の検索結果タイプ（ DBA_ADVISOR_FINDINGS ビューFINDING_NAME 列 ） を制限
INSERT_PARAMETER_DIRECTIVE	初期化パラメータを変更することを推奨するアクションを制限する
INSERT_SEGMENT_DIRECTIVE	セグメントアドバイザの実行を推奨するアクションを制限する（以下の場合はスキーマ単位で制限） DELETE_SEGMENT_DIRECTIVE('ﾀｽｸ名','ﾃﾞｨﾚｸﾃｨﾌﾞ名','ｽｷｰﾏ名')
INSERT_SQL_DIRECTIVE	SQL（SQL_ID）に対する推奨を制限する
DELETE_XXXX	INSERT_XXXX と同じものがあり、制限を削除する

既存のタスク名にフィルタする場合は、ADDM タスクのステータスが「INITIAL（推奨項目なし)」の場合のみフィルタ可能。

ディレクティブ名は、タスク名の中で一意の必要があり、重複してるとエラーになる。

今後の ADDM 結果をフィルタする例

```
DBMS_ADDM.INSERT_FINDING_DIRECTIVE(
    task_name => NULL,
    dir_name  => '上位 SQL 文ディレクティブ',
    finding_name => '上位 SQL 文',
    min_active_sessions => 10,
    min_perc_impact => 30
);
```

作成するタスクの名前
今後作成されるタスクに対してフィルタする
任意のディレクティブ名
フィルタしたい検索結果タイプ（DBA_ADVISOR_FINDINGS ビューで確認）
検索結果が 10 件より少ない場合はフィルタ
検索結果の影響(%)が、30 より少ない場合はフィルタ
※フィルタの条件は OR

DBA_ADDM_TASK_DIRECTIVES ビューで作成したタスク（TASK_NAME）とディレクティブ（INSTANCE_NAME）の確認ができる。

EM で追加する場合は[アドバイザ・セントラル] から 表示したいタスクを選択し「結果の表示」をクリック。

診断が必要なタスクがある場合は以下のように表示されるので、結果（FINDING_NAME）をクリック。

各・自動診断画面に遷移するので「フィルタ」を押下し、条件を指定して登録する。

影響(%)	結果	発生数(分析期間で終わる24時間)
50.2	上位SQL文	1/11
35.4	"スケジューラ"待機クラス	1/11
7.3	ソフト解析	1/11
6.8	ハード解析	1/11
5	異常な"アプリケーション"待機イベント	1/11
4.9	セッション接続と切断	1/11
2.5	コミットとロールバック	1/11

自動メモリー管理

自動メモリー管理に関して以下のパラメータが追加されている。

初期化パラメータ	説明
MEMORY_TARGET	0 以外の指定で自動メモリー管理が有効、PGA、SGA を自動的に調整する（動的パラメータ）
MEMORY_MAX_TARGET	MEMORY_TARGET で使用する最大メモリーサイズ（静的パラメータ）

自動メモリー管理が有効（MEMORY_TARGET≠0）な場合の動作(SGA_TARGET と PGA_AGGREGATE_TARGET の有効有無)

SGA	PGA	説明
○	○	MEMORY_TARGET と PGA_AGGREGATE_TARGET で設定されたサイズを下限値として動作する（その他の TARGET も同様）
○	―	PGA は、MEMORY_TARGET - SGA_TARGET の値に自動的に調整される

－	○	SGA は、MEMORY_TARGET － PGA_AGGREGATE_TARGET の値または、SGA_MAX_SIZE（設定時）の何れか小さい方に設定される
－	－	MEMORY_TARGET の 60% が SGA、40%が PGA に設定される

自動メモリー管理が無効（MEMORY_TARGET=0 または無指定）な場合の動作

条件	説明
●MEMORY_MAX_TARGET ×MEMORY_TARGET	MEMORY_TARGET が 0 に設定される
●MEMORY_MAX_TARGET ×SGA_MAX_SIZE	MEMORY_MAX_TARGET の値が SGA_MAX_SIZE に設定される
●SGA_TARGET ●PGA_AGGREGATE_TARGET	SGA と PGA が個別に管理される（自動共有メモリー管理と、自動 PGA メモリー管理）
●SGA_TARGET ×PGA_AGGREGATE_TARGET	PGA_AGGREGATE_TAGERT が SGA_TARGET の 20% または 10MB の何れか大きい方を設定
×SGA_TARGET ●PGA_AGGREGATE_TARGET	SGA は自動調整されず、各コンポーネントを明示的に設定する
×SGA_TARGET ×PGA_AGGREGATE_TARGET	SGA は自動調整されず、各コンポーネントを明示的に設定する PGA はインスタンス起動時の SGA サイズの 20% または 10MB の何れか大きい方を設定

自動メモリー管理に関連するビュー

ビュー	説明
V$MEMORY_TARGET_ADVICE	MEMORY_TARGET を変更したことによる DBMS_RESOURCE_MANAGER パッケージ時間の影響をアドバイスする。現在のメモリーサイズの 25～200%のサイズでアドバイスが行われる
V$MEMORY_DYNAMIC_COMPONENTS	すべてのメモリーコンポーネント（SGA と PGA）の現在の状況を表示する
V$MEMORY_CURRENT_RESIZE_OPS	現在進行中のメモリーサイズ変更の状況を表示
V$MEMORY_RESIZE_OPS	完了したメモリーサイズ変更の状況（最新 800 個）を表示

データ圧縮の拡張

DML（INSERT、UPDATE、DELETE）時に圧縮可能になった。（今まではダイレクトパスロード時のみ圧縮）

表領域、パーティション、表単位での圧縮が可能。

PCTFREE の割合に達するまでは未圧縮で格納し、達したら圧縮される。再び PCTFREE に達するまでは非圧縮で格納。

-> データ圧縮の設定

```
{ CREATE | ALTER } TABLE 表名 ... COMPRESS FOR { { ALL | DIRECT_LOAD } OPERATIONS | OLTP };
{ CREATE | ALTER } TABLESPACE 表領域名 ... COMPRESS FOR { { ALL | DIRECT_LOAD } OPERATIONS | OLTP }

ALTER TABLE 表名 NOCOMPRESS;
```

COMRESS FOR	説明
ALL	OLTP	ダイレクトパスロード時に加え、DML 操作時に圧縮を行う（ALL は 11gR1 OLTP は 11gR2 での指定）
DIRECT_LOAD	ダイレクトパスロード時のみ圧縮（10g までの機能）デフォルト

ALTER で圧縮を有効にしても、既存のデータは圧縮対象にならず後から追加/変更されたデータが圧縮対象になる。

ALTER で圧縮を無効にしても、既存のデータは圧縮されたままで、後から追加/変更されたデータのみ非圧縮対象になる。

オプティマイザ統計のプリファレンス

オプティマイザ統計の収集時に使用される各種デフォルト設定（属性）を個別に設定できるようになった。

-> プリファレンスの概要

統計プリファレンスとは、オプティマイザ統計情報の収集に使用されるデフォルト属性で以下を設定できる。

プリファレンス	説明

CASCADE	索引の統計を同時に収集するか
DEGREE	統計の収集時に使用される並列度
ESTMATE_PERCENT	サンプリングによる統計収集で使用する割合
METHOD_OPT	列の統計情報とヒストグラム統計情報の収集を制御
NO_INVALIDATE	統計収集時に共有プール内のカーソルを無効にするかどうか
GRANULARITY	パーティション表に対する統計収集時の粒度。表パーティション統計とグルーバル統計取得の制御
PUBLISH	収集した統計情報を通常使用できるように公開するか
INCREMENTAL	パーティション表のグローバル統計を増分で取得するか（後述）
STALE_PERCENT	統計の失効を判断する為のしきい値

-> プリファレンスの設定

［サーバー］－［オプティマイザ統計の管理］　または、DBMS_STATS パッケージを使用して定義する。

ﾌﾟﾛｼｰｼﾞｬ/ﾌｧﾝｸｼｮﾝ	説明
SET_TABLE_PREFS	表レベルのプリファレンスを設定（DELETE_TABLE_PREFS で削除）
SET_SCHEMA_PREFS	スキーマレベルのプリファレンスを設定（DELETE_SCHEMA_PREFS で削除）
SET_DATABASE_PREFS	データベース内のすべての表のプリファレンスを設定（DELETE_DATABASE_PREFS で削除）
SET_GROBAL_PREFS	プリファレンスを持たない表で使用されるプリファレンスを設定
DELETE_XXXX	SET_XXXX と同じプロシージャがあり設定を削除する
RESET_GROBAL_PREF_DEFAULT	グローバルレベルのプリファレンスを Oracle のデフォルト設定に戻す
GET_PREFS	現在のプリファレンスを取得。パラメータは、PNAME、OWNNAME、TABNAME の 3 つ TABNAME まで指定すれば表レベル、PNAME のみならグローバルレベルのプリファレンス取得

プリファレンス設定例

```
exec DBMS_STATS.SET_GLOBAL_PREFS(
    pname  => 'STALE_PERCENT',
    pvalue => '15');

SELECT DBMS_STATS.GET_PREFS('STALE_PERCENT') FROM DUAL;
```

失効に関するプリファレンスを設定
15 に変更

変更結果の確認

オプティマイザ統計収集の拡張

-> パーティション表によるグローバル統計

パーティション表の統計は、個々のパーティション統計と、表全体のグローバル統計が必要。

11g では「INCREMENTAL」プリファレンスを使用することでグローバル統計を増分で取得できるようになった。

増分取得をしている場合、DML が監視され、しきい値（10%以上の変更）を越えたパーティション表のみが、

グローバル統計の収集の為にスキャンされるようになる。結果グローバル統計が低コストで収集できる。

統計収集の際、GRANULARITY パラメータに「GLOBAL」または「GLOBAL AND PARTITION」を指定すると増分収集になる。

-> 拡張統計（複数列、式）

拡張統計と呼ばれるヒストグラム統計が追加されている。

複数列（列グループ）の統計

列の組み合わせから計算されたヒストグラム統計を提供する。

```
WHERE c1='A' and c2='A'
c1   c2   count(*)
A    A    10000      列グループ統計なし
```

```
A     B     10          C1 列=1/3 (33%)、c2 列=1/3 (33%) ⇒ 1/3 x 1/3 = 1/9 (11%)
B     A     10
B     B     10000       列グループ統計あり
C     A     10          (C1 列,c2 列) でヒストグラム = 10000/30040 (33%)
C     B     10          ※c1 が A と c2 が A の組み合わせは、30040 レコード中、10000 個
C     C     10000
```

式の統計

計算や関数による式に対するヒストグラム統計を提供する。

式に対する選択性は常に 1%で想定され常に索引を使用できるように考慮されているが、式の結果値に偏りがある場合、

不正確な選択性により不適切な実行計画を作成されることになる。

11g では仮想列を作成し、仮想列に索引を作成することで、式の結果値に基づく選択性を正確に見積もることが可能。

```
WHERE c1||'_'||c2='A_A'
c1    c2    count(*)
A     A     10000       ファンクション索引を使用した場合
A     B     10          create index i1 on t1(c1||'_'c2);  ⇒ 自動的に 1%と想定される
B     A     10
B     B     10000       式の統計と仮想列に索引作成
C     A     10          create index i1 on t1(仮想列名);
C     B     10          仮想列名は DBA_STAT_EXTENSIONS ビューの EXTENSION_NAME 列で確認可能
C     C     10000       (c1||'_'c2)の計算結果でヒストグラム = 10000/30040 (33%)
```

拡張統計を収集するには、DBMS_STATS パッケージによるオプティマイザ収集時に

ヒストグラムを指定する METHOD_OPT パラメータの列名を指定する箇所で列グループまたは式を指定する。

拡張統計の収集例

```
DBMS_STATS.GATHER_TABLE_STATS('SCOTT','T1',
    method_opt => 'FOR COLUMNS c1,c2 SIZE 10');      method_opt の記述により変わる
    method_opt => 'FOR COLUMNS (c1,c2) SIZE 10');    通常のヒストグラム収集
    method_opt => 'FOR COLUMNS (c1||'_'||c2) SIZE 10'); 列グループの統計収集
                                                     式の統計収集
```

指定した列グループまたは式がない場合は、自動的にシステムで付与した名前の仮想列が作成される。

自動オプティマイザ統計収集で分析されるワークロードに基づき拡張統計の必要有無が自動判定されるが、

事前に拡張統計を定義しておき、オプティマイザ統計収集でも使用させることが可能。

DBMS_STATS パッケージ

ﾌﾟﾛｼｰｼﾞｬ/ﾌｧﾝｸｼｮﾝ	説明
CREATE_EXTENDED_STATS	指定した列グループまたは式を保存（統計収集対象になる）し仮想列名を戻す
SHOW_EXTENDED_STATS_NAME	指定した列グループまたは式に対する仮想列名を戻す
DROP_EXTENDED_STATS	指定した列グループまたは式に対する仮想列を削除する

拡張統計の定義

```
DECLARE
    ename VARCHAR2(30);                              戻り値を格納する変数を宣言
BEGIN
ename := DBMS_STATS.CREATE_EXTENDED_STATS(           作成された仮想列名を ename に格納する
    ownname   => 'SCOTT',                            DBA_STAT_EXTENSIONS ビューで作成した仮想列の確認が可能
    tabname   => 'T1',
    extension => '(C1,C2)');                         列グループを定義。式の場合は(C1||'_'||C2)
END;
/
```

-> 統計の保留と公開

統計を取得すると新規の統計情報が自動的に使用されていたが、11g では使用を保留できるようになった。

DBMS_STATS.SET_TABLE_PREFS プロシージャにてプリファレンス名 PUBLISH の値を FALSE にすることで
指定したテーブルの統計を取得しても保留状態になる。
保留状態の統計は DBA_TAB_PENDING_STATS ビューで確認可能。
保留した統計でテストを行いたい場合は、OPTIMIZER_USE_PENDING_STATISTICS パラメータを TRUE にする。
保留した統計を公開するには、DBMS_STATS.PUBLISH_PENDING_STATS プロシージャを使う。

統計保留と公開

```
DBMS_STATS.SET_TABLE_PREFS(
    ownname => 'SCOTT',
    tabname => 'EMP',                    EMP テーブルを統計保留にする
    pname   => 'PUBLISH',                統計取得後の公開に関するプリファレンス
    pvalue  => 'FALSE');                 FALSE で保留にする。TRUE だと統計取得と同時に公開される

DBMS_STATS.GATHER_TABLE_STATS('SCOTT','EMP');    統計取得を行う。この時点で取得した統計は実際には使われない

ALTER SESSION SET
OPTIMIZER_USE_PENDING_STATISTICS=TRUE            保留されてる統計を使用する

DBMS_STATS.PUBLISH_PENDING_STATS('SCOTT','EMP');    保留にしていた統計を公開する
```

適応カーソルの共有

10g ではバインド変数を使った実行計画は、最初に使われた値で作成され、以降はどの値でも同じ実行計画が使われる。
（バインドピーク機能）
11g ではバインド変数値でカーソルを共有せず、必要に応じて異なる実行計画を作成する。
CURSOR_SHARING 初期化パラメータを EXACT（完全一致のみカーソル共有）ではなくそれ以外（FORCE、SIMILAR）にすることで
述語のリテラルが自動的にバインド変数に変換されカーソルを共有できるようになる。

-> 適応カーソル共有の動作

バインド変数を使用した SQL 解析では「バインド依存」と「バインド認識」のマーク付けにより
適応カーソルを共有するか判定する。

マーク	説明
バインド依存	バインド変数を含む SQL 文が使用するオブジェクトで、ヒストグラムが使用されているか判定 V$SQL の IS_BIND_AWARE 列　で確認
バインド認識	異なる実行計画が必要と判定されたカーソル（V$SQL の IS_BIND_AWARE 列　で確認）

バインド依存カーソルの識別

バインド変数を含む SQL 文の最初の実行時に行われるハード解析で、バインドピークが行われる。
コスト計算にヒストグラムが使用される場合、SQL 文はバインド依存としてマークされ、
実行後にバインド変数と実行統計がカーソル内に保存される。
実行統計は V$SQL_CS_STATISTICS ビューにて確認

バインド認識のマーク付け

バインド依存カーソルは、異なるバインド変数値が割り当てられても従来通りのソフト解析によってカーソルを共有し、
実行統計のパターンを観測する。
実行計画が大幅に異なることが確認されるとバインド依存カーソルにバインド認識のマーク付けが行われる。

ソフト解析とハード解析

バインド認識のマーク付けが行われると、選択性が検証され、今後の実行においてカーソルを共有する為のソフト解析と選択性が大幅に異なる場合のハード解析が必要に応じて実行される。

■パーティション化の新機能

時間隔パーティション化

-> 時間隔パーティション化の概要

レンジパーティションを使用している場合、定期的にパーティションを追加する必要があった。

時間隔パーティションは、指定の間隔に基づいて自動的にパーティションを作成する。

パーティション格納の最大値以上の表パーティションが必要になると、INSERT 時に自動的に作成される。

以下のルールがある。

・パーティションキーは 1 列のみで、NUMBER 型か、DATE 型である

・索引構成表では時間隔パーティション化はサポートされない

・ドメイン索引は作成できない

・最後のパーティションに MAXVALUE を使用できない（追加されなくなるので意味がない）

・基準パーティションを複数作成した場合、間隔以上離れていても間のパーティションは作成されない

・STORE IN 句を省略した場合はユーザーのデフォルト表領域に作成する

・STORE IN 句で複数の表領域を作成した場合は、ラウンドロビンにて複数の表領域にパーティションを作成する

-> 時間隔パーティション化の設定

レンジパーティションに INTERVAL 句で時間隔と初期パーティションを最低 1 つ指定する。

時間隔や表領域リストは ALTER 文で変更可能だが、既存の表パーティションには影響せず、追加される表パーティションから適用される。

時間隔パーティション作成

```
CREATE TABLE テーブル名 (列名 型 ...)
PARTITION BY RANGE (パーティションキー)
  INTERVAL(インターバル) [STORE IN (表領域名[,表領域名])]
  ( PARTITION 初期パーティション名 1 VALUES LESS THAN (基準値 1),
    PARTITION 初期パーティション名 2 VALUES LESS THAN (基準値 2),
    ...)

-- 変更
ALTER TABLE テーブル名 SET INTERVAL(インターバル) [STORE IN (表領域名[,表領域名])]
※インターバルを空白することで無効化できる
```

インターバルの指定は、NUMBER 型の列であれば数値。DATE 型の場合は以下の関数を使用する。

関数名	説明
NUMTODSINTERVAL(期間,'文字列')	文字列は DAY、HOUR、MINUTE、SECOND の何れか指定し、期間は数値を入れる
NUMTOYMINTERVAL(期間,'文字列')	文字列には YEAR、MONTH の何れか指定し、期間は数値を入れる

-> 遷移ポイントの移動

隣接した表パーティションはマージさせることで新しい表パーティションにすることが可能（隣接してないとエラー）

マージ後は、後側のパーティション境界になる。

パーティションのマージ

```
ALTER TABLE EMP MERGE PARTITIONS
    FOR (TO_DATE('2014-05-15','YYYY-MM-DD')),
    FOR (TO_DATE('2014-06-15','YYYY-MM-DD'))
INTO PARTITION sys_pz;
```

2014-05-15 の値を含むパーティションと
2014-06-15 の値を含むパーティションをマージする
新しいパーティション名を指定（省略も可能）

仮想化ベースのパーティション化

ファンクションや式による「仮想列」を表に定義できるようになった。

パーティションキーの利用や、や索引（ファンクション索引になる）の作成が可能。

仮想列は他の列の値を元に計算された値になるので、INTO で明示的に値を入力することは出来ない。

仮想列の定義は 列名 AS 式 VIRTUAL で行う。(VIRTUAL は不要だが仮想列であること明記させる為だけに使用)

-> 仮想列ベースのパーティション化の設定

設定例

```
CREATE TABLE EMP (
    no   NUMBER(4),
    sal NUMBER(7,2),
    sal_rank AS (
      CASE
        WHEN NVL(sal,0)<1000 THEN 'C'
        WHEN NVL(sal,0)<3000 THEN 'B'
        ELSE                      'A'
      END) VIRTUAL)
PARTITION BY LIST(sal_rank)
( PARTITION A VALUES('A'),
  PARTITION B VALUES('B'),
  PARTITION C VALUES('C'),
  PARTITION N VALUES(DEFAULT));

UPDATE EMP SET sal=3500 WHERE no=50;
※ORA-14402 が発生

ALTER TABLE EMP ENABLE ROW MOVEMENT;
```

```
仮想列 sal_rank の作成
sal の値により、A～C の値を決定する

仮想列をパーティションキーにする

パーティションキー列（sal）の値が変更されることで、
パーティション格納先が変わる場合はエラーになる

行移動を有効化することで、エラーを回避できる
```

参照パーティション化

通常、マスター/ディテールでアクセスする表をパーティション化する場合、パーティション毎に結合させる為に各表に同じパーティションキーと範囲を設定する必要があった。参照パーティションを使うことでディテール側にパーティションキーを追加することなくマスター側のパーティションメンテナンス操作を参照パーティション（ディテール側）に自動的に反映させる。

-> 参照パーティション化の設定

ディテール表側で「PARTITION BY REFERENCE（参照整合性制約名）...」を指定する。

設定例

```
CREATE TABLE orders (
    order_id     NUMBER(4),
    order_date   DATE,
    order_mode   VARCHAR(5),
CONSTRAINT orders_pk PRIMARY KEY(order_id))
PARTITION BY RANGE(order_date)
(PARTITION p2013 VALUES LESS THAN(TO_DATE('20140101','YYYYMMDD')),
 PARTITION p2014 VALUES LESS THAN(TO_DATE('20150101','YYYYMMDD')));

CREATE TABLE orders_items (
    order_id     NUMBER(4) NOT NULL,
    item_id      NUMBER(4),
CONSTRAINT orders_item_fk
    FOREIGN KEY(order_id) REFERENCES orders(order_id))
PARTITION BY REFERENCE(orders_item_fk);


ALTER TABLE orders
ADD PARTITION p2015 VALUES LESS THAN(TO_DATE('20160101','YYYYMMDD'))
DEPENDENT TABLES (orders_items (PARTITION P2015F));
```

```
参照表（マスター/親表）の作成

プライマリキー制約を指定
パーティションキーは order_date

参照パーティション（ディテール/子表）の作成

親表の order_id 列の値にある数値のみ、
子表の order_id 列に入力可能な制約を定義
参照整合制約から継承されたパーティションキーを内部的
に使用し、パーティションを自動作成する

パーティション追加

子表のパーティション名を指定することも可能
```

参照パーティションの制約

・インターバルパーティションでは利用不可

・索引構成表、外部テーブル、ドメイン索引の格納テーブルには設定できない

・親テーブルには主キー制約（PRIMARY KEY）または一意キー制約（UNIQUE KEY）が必要

・子テーブルの外部キーには NOT NULL 制約の指定が必要

・参照パーティション表にパーティションを追加することはできない（参照表から継承されるため）

・参照パーティション化を行った後、参照整合性制約を無効にすることはできない

システムパーティション化

パーティションキーを持たず、INSERT する際に格納するパーティションを指定する。※指定しないとエラーになる

-> システムパーティション化の設定

CREATE TABLE ... PARTITION BY SYSTEM ... を指定する。

設定例

```
CREATE TABLE visitor (                                  システムパーティションを作成する
    id          NUMBER(4),
    name        VARCHAR(5),
    visit_date  DATE )
PARTITION BY SYSTEM
(PARTITION left  TABLESPACE ts1,
 PARTITION right TABLESPACE ts2);

INSERT INTO visitor VALUES (1,'MAI',sysdate);           パーティションを指定しないエラーになる
※ORA-14701 エラーが発生し INSERT できない

INSERT INTO visitor PARTITION (left) VALUES (1,'MAI',sysdate);   INSERT する場合はパーティション表を指定する必要がある
```

-> システムパーティション化の制約

・一意のローカル索引の作成ができない（CREATE UNIQUE INDEX ～ LOCAL ではなく CREATE INDEX ～ LOCAL なら可能）

・CREATE TABLE AS SELECT 文による表作成ができない

・SPLIT PARTITION によるパーティション分割ができない

コンポジットパーティション化の拡張

10g までは、レンジ--ハッシュ または レンジーリスト のみ可能だった。

11g は親パーティションに レンジ、時間隔、リスト　サブパーティションに、レンジ、ハッシュ、リスト が利用可能。

PARTITION BY パーティション種類(パーティションキー)　SUBPARTITION BY サブパーティション種類(パーティションキー)

親パーティション		サブパーティション	
レンジ	PARTITION BY RANGE	レンジ	SUBPARTITION BY RANGE
時間隔	PARTITION BY RANGE ... INTERVAL	ハッシュ	SUBPARTITION BY HASH
リスト	PARTITION BY LIST	リスト	SUBPARTITION BY LIST

リストパーティションのレンジサブパーティション作成例

```
CREATE TABLE orders (
    id   NUMBER(1),
    cust NUMBER(3) )
PARTITION BY LIST (id)                                  リストパーティションを作成
```

```
SUBPARTITION BY RANGE (cust)
 ( PARTITION p0_4 VALUES ('0','1','2','3','4')
  (SUBPARTITION p0_4_300 VALUES LESS THAN(300),
   SUBPARTITION p0_4_600 VALUES LESS THAN(600),
   SUBPARTITION p0_4_max VALUES LESS THAN(MAXVALUE)),
  PARTITION p5_9 VALUES ('5','6','7','8','9')
  (SUBPARTITION p5_9_300 VALUES LESS THAN(300),
   SUBPARTITION p5_9_600 VALUES LESS THAN(600),
   SUBPARTITION p5_9_max VALUES LESS THAN(MAXVALUE))
);
```

サブパーティションとしてレンジパーティションを作成
id が 0～4 の時は、p0_4 へ格納される
cust が 300 未満、600 未満、それ以外で、更に
レンジパーティションへ格納される

SQL アクセスアドバイザの拡張機能

SQL アクセスアドバイザで（索引、マテリアライズド・ビューの他に）パーティション化の推奨が可能になった。

表だけではなくパーティション化された索引も推奨出来る。

EM の[アドバイザ・セントラル] ページの [SQL アドバイザ] - [SQL アクセスアドバイザ] で実行するウィザード内の

推奨オプションページで「パーティショニング」が選択できる。

-> パーティション推奨の制約

・10000 以上のレコードを持つ表が対象

・述語や結合で NUMBER 型または DATE 型を使用していること

・パーティションタイプは「時間隔」または「ハッシュ」のみ

■Recovery Manager の新機能

バックアップ最適化

-> 圧縮バックアップのパフォーマンス改善

圧縮アルゴリズムに従来の「BASIC（BZIP2)」の他に「MEDIUM（ZLIB)」をサポートした。

ZLIB の方が高速（60%程早い）だが圧縮率は低い（大差はない）

```
RMAN> CONFIGURE COMPRESSION ALGORITHM '{ BASIC | LOW | MEDIUM | HIGH }';
```

-> UNDO バックアップの最適化

リカバリに不要な UNDO データはバックアップ対象外になった。

-> 増分バックアップの高速化

増分バックアップは更新したブロックを確認する為に、バックアップ対象のデータファイルをすべて読み込む必要がある。

バックアップ対象ブロックを確認する I/O 負荷がボトルネックとなる。

「ALTER DATABASE **ENABLE** BLOCK CHANGE TRACKING」コマンドを実行することで、変更された情報をブロックチェンジトラッキングファイルに書き込むことで、増分バックアップ時はこのファイルを参照するだけで済むようにし高速化を実現している。

```
(例) alter database enable block change tracking using file '/opt/app/oracle/oradata/orcl11/users01.dbf'
```

-> マルチセクションバックアップ

バックアップセットの作成は従来 1 つのデータファイルに複数チャンネルでアクセスできなかった。

11g ではマルチセッションバックアップにより 1 つのファイルを指定のサイズで分割し、1 つのバックアップセットが

異なるピースとしてバックアップされることで、パラレルによるバックアップを行うことが出来るようになった。

```
RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK PARALLELISM 3;
RMAN> <BACKUP | VALIDATE> {DATABASE | DATAFILE | TABLESPACE} SECTION SIZE ｻｲｽﾞ [K | M | G] ;
```

バックアップまたは破損ブロック検証（VALIDATE）時に指定したサイズで分割され、

チャンネルが１つ（デフォルト）であればシリアルで、複数指定していればパラレルにバックアップが実行される。

アーカイブログ管理の改善

11g では RMAN によるバックアップ削除指示があっても即時に削除せず、
必要とするコンポーネント（フラッシュバック DB など）がなくなった時に削除できるようになっている。
フラッシュリカバリ領域内にあるアーカイブログを自動的に削除する条件を指定することが可能。

CONFIGURE ARCHIVELOG DELETION POLICY 削除ポリシー

削除ポリシー	説明
TO NONE	削除ポリシーを指定しない
TO BACKED UP num TIMES TO DEVICE TYPE device	num 回バックアップ取得後に削除可能。device は、DISK や SBT（テープ）を指定
TO SHIPPED TO ALL STANDBY	スタンバイサイトへの**転送**が終わった場合に削除可能にする
TO APPLIED ON ALL STANDBY	スタンバイサイトへの**適用**が終わった場合に削除可能にする
CLEAR	設定をクリアする

V$ARCHIVED_LOG ビューでアーカイブログのバックアップ状況が確認できる。

-> 長期バックアップ（アーカイブバックアップの改善）

アーカイブログはオンラインバックアップに必要なファイルのみ取得対象になった。

BACKUP {DATABASE | DATAFILE | TABLESPACE} **KEEP UNTIL TIME** '日付' [RESTORE POINT ﾘｽﾄｱﾎﾟｲﾝﾄ名]

CHANGE ... { KEEP { FOREVER | UNTIL TIME '日付' } | NOKEEP };

オプション	説明
KEEP UNTIL TIME	バックアップを不要としない期間。この期間を超過すると削除対象になる
RESTORE POINT	リストアポイント名を RESTORE 句でリストアする時に指定することで、取得したバックアップ時点に戻せる
KEEP FOREVER	不要とされないようにする。（リカバリ・カタログが必要）
NOKEEP	KEEP 属性が削除され、保存方針に従って削除対象になる

KEEP 句を指定したバックアップの制約。

・データファイル、アーカイブログファイル、制御ファイル、SPFILE がバックアップされる
・オンラインバックアップのリカバリに必要なアーカイブログファイルのみ含まれる
・新規にバックアップするときに限り、リストアポイントの指定が可能
・フラッシュリカバリ領域にはバックアップを保存できない

アクティブデータベースの複製

10g ではソースデータベースのバックアップから複製が可能だったが、11g では、バックアップがなくても
Oracle Net を使用し補助 DB に接続し、ネットワーク経由で DB が複製できるようになった。
SPFILE やパスワードファイルをソース DB からコピーすることも可能になっている。

複製する為の条件

・ソース DB（ターゲット DB）へは connect target、複製 DB へは connect auxiliary にて Oracle Net で接続する
・ソース DB をオープン状態で複製する場合、ARCHIVLOG モードであること（MOUNT 状態なら NOARCHIVELOG モードで良い）
・複製 DB にはパスワードファイルが存在し、ソース DB と同じ SYS パスワードが設定されていること

-> データベースの複製

EM の［データ移動］-［データベースのクローニング］-［クローンデータベース］から行う。

RMAN でデータベースを複製する場合（orcl1 を orcl2 へ複製）

① データファイル格納先となるフォルダを作成する

② 起動用 PFILE を作成する（ /tmp/init.ora などのファイルを作成し以下を記述する）

```
DB_NAME=orcl2
REMOTE_LOGIN_PASSWORDFILE=exclusive
```

③ 複製するインスタンスをリスナーに追加（listener.ora に 追記する）し lsnrctl reload で再読込を行う。

④ 接続用のネットサービス名の構成を追加（tnsnames.ora に 追記する）

⑤ パスワードファイルの準備（orapwd file=$ORACLE_HOME/dbs/orapworcl2 password=oracle entries=5 force=y）

⑥ 複製インスタンスの起動

```
sqlplus sys/oracle@orcl2 as sysdba
startup mount pfile=/tmp/init.ora
```

⑦ ソース DB（ターゲット DB）に接続（ RMAN> connect target sys/oracle@orcl1）

⑧ 複製インスタンスに接続（ RMAN> connect auxiliary sys/oracle@orcl2）

⑨ データベースの複製を行う（RMAN でコマンド実行）

DUPLICATE TARGET DATABASE TO orcl2
[FOR STANDBY]
FROM ACTIVE DATABASE
SPFILE
  SET MEMORY_TARGET='200M'
  SET CONTROL_FILES='ﾌｧｲﾙﾊﾟｽ'
  SET DB_FILE_NAME_CONVERT='ﾌｧｲﾙﾊﾟｽ'
NOFILENAMECHECK
	TO に複製したいインスタンス名を指定する
	フィジカルスタンバイ DB を作成する場合は FOR STANDBY 句を入れる
	アクティブデータベース（ソース DB から直接複製）複製を行う
	SPFILE 句を使い一部の初期化パラメータを変更できる
	MEMORY_TARGET 初期化パラメータを 200MB に変更。
	制御ファイルのパスを指定
	DB ファイルのパスを指定
	DB ファイルの指定をしない場合、ソース DB と同じパスに作成するが、
	その場合エラーが発生するので、エラーを回避する為に指定する

リカバリカタログの拡張機能

-> リカバリカタログのマージ

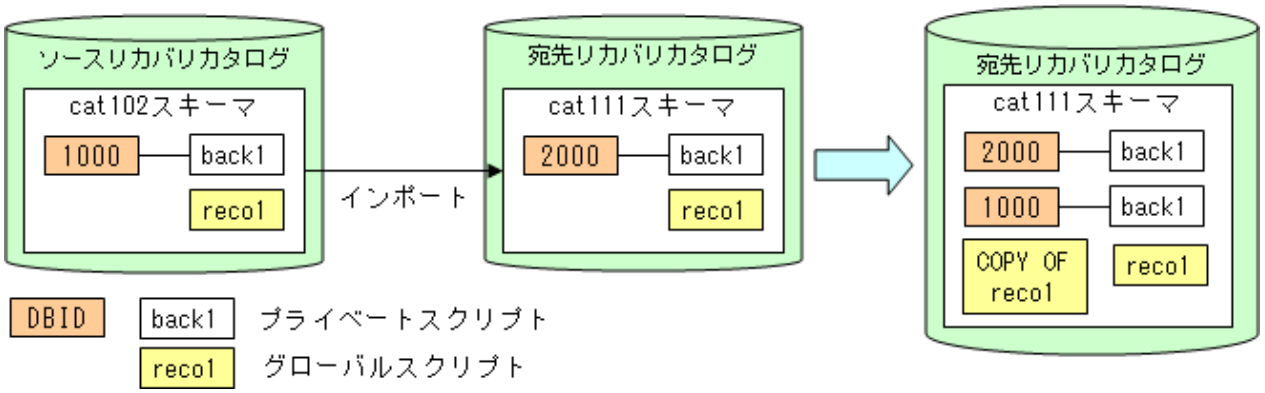
既存のリカバリカタログで管理しているターゲット DB の情報（RMAN リポジトリ）を別のリカバリカタログにまとめること。

IMPORT CATALOG 接続文字列 [DBID=DBID ﾘｽﾄ] [DB_NAME=DB 名ﾘｽﾄ] [NO UNREGISTER];

パラメータ名	説明
接続文字列	インポートするソースリカバリカタログに接続するための接続文字列。UID/PASS@ﾈｯﾄｻｰﾋﾞｽ名
DBID	ソースリカバリカタログからインポートするターゲット DB を識別する DBID のリスト（,で区切る）
DB_NAME	ソースリカバリカタログからインポートするターゲット DB を識別する DB 名のリスト（,で区切る）
NO UNREGISTER	インポート後、ソースリカバリカタログから登録を解除しない（デフォルトでは解除される）

オプションを使用しない場合は、ソースリカバリカタログに存在する全てのターゲット DB 情報がインポートされる。

（例）RMAN> CONNECT CATALOG cat111/oracle@db11g
RMAN> IMPORT CATALOG cat102/oracle@db10g [DBID=1000]



・RMAN のリポジトリとして格納されるバックアップ情報、永続構成、ストアドスクリプトがインポートされる
・ストアドスクリプトは、ターゲット DB 毎に使用されるプライベートスクリプトと全てのターゲット DB で使われる
グローバルスクリプトの 2 種類あり、宛先リカバリカタログと名前が競合した場合は、
COPY OF 元の名前　の形式でインポートされる。(RMAN> LIST SCRIPT NAMES で確認可能)
・宛先リカバリカタログに既にターゲット DB が含まれている場合はエラーとなりロールバックされる

-> 仮想化プライベートカタログ

リカバリ・カタログの所有者はリカバリ・カタログに登録された全てのターゲット・データベースのバックアップ情報にアクセス可能。11g では仮想プライベート・カタログを構成し、バックアップ対象データベースを管理者ごとに制限できる。

① 基本カタログの所有者と仮想プライベートカタログ（VPC）所有者の作成（CREATE USER ～）
作成したユーザーに対し RECOVERY_CATALOG_OWNER 権限を付与する（GRANT RECOVERY_CATALOG_OWNER TO USER）
② 基本カタログの作成（基本カタログの所有者で接続）
RMAN> connect catalog 基本ｶﾀﾛｸﾞ所有者/ﾊﾟｽﾜｰﾄﾞ@SID
RMAN> CREATE CATALOG;
③ VPC 所有者への権限付与（基本カタログの所有者で接続後、以下のコマンドを実行）
RMAN> GRANT REGISTER DATABASE TO VPC 所有者（ターゲット DB の登録を可能にする権限を付与する）
RMAN> GRANT CATALOG FOR DATABASE ﾀｰｹﾞｯﾄ DB TO VPC 所有者（指定したターゲット DB へのアクセスを許可させる）
④ 仮想カタログを作成（VPC 所有者でリカバリカタログ DB に接続）
RMAN> CREATE VIRTUAL CATALOG;
⑤ 新しい DB をカタログに登録（登録権限（③の GRANT REGISTER ～ で付与）がある場合は DB の登録が可能）
ターゲット DB と、リカバリカタログ DB の両方に接続して、カタログ登録を行う。
$ rman TARGET DBUID/PASS@SID CATALOG VPCUID/PASS@SID
RMAN> REGISTER DATABASE;

■障害診断の新機能

自己診断リポジトリ（ADR）

アラートログやトレースファイルの個別ディレクトリを廃止し、自動診断リポジトリ（ADR）に全て格納される。
ADR ベースディレクトリは以下の優先順位で決定する。
DIAGNOSTIC_DEST パラメータの値　⇒　$ORACLE_BASE　⇒　$ORACLE_HOME/log

-> ディレクトリの階層

ADR ベース

```
└ diag
    ├ asm          ASA インスタンス用。rdbms 同様に ASM インスタンス名のサブディレクトリが ADR ホーム
    ├ clients      クライアント処理用。trace ディレクトリに sqlnet.log
    ├ tnslsnr      リスナー用。trace サブディレクトリに listener.log
    ├ ...
    └ rdbms
        └ <DB 名>
            └ <SID 名>        ADR ホーム
                ├ alert       XML 形式のアラートログ
                ├ cdump       コアダンプ
                ├ incpkg      インシデントパッケージ
                ├ incident    インシンデントダンプ（インシデント ID 毎のサブディレクトリあり）
                ├ ir          データリカバリアドバイザによる修復スクリプト
                ├ hm          ヘルスモニターの結果
```

└─ trace　　テキスト形式のアラートログ、バックグラウンドプロセスのトレースファイル、ユーザートレース

V$DIAG_INFO ビューでディレクトリのパスが確認できる。

ADR の各ディレクトリは削除されても自動的に作成される。

-> 問題とインシデント

内部エラー（ORA-00600）、OS 例外（ORA-00745）、共有メモリ割り当てエラー（ORA-004031）などのクリティカルエラーを「問題」と呼び、問題が発生すると、ADR で追跡できるように問題キーが割り当てられる。

問題キーはエラーコードを含む、テキストの文字列（例：ORA 7445[qcstda()+490）で同一問題の場合は同じ問題キーになる。

個々のエラーを「インシデント」と呼び、エラーの度に一意のインシデント ID を作成し次の処理が行われる。

(1) アラートログにエントリを作成

(2) インシデントに関する障害診断データダンプ（インシデントダンプ）を取得し、インシデント ID と対応付けた後、ADR ホームの incident サブディレクトリに格納する

フラッド制御機能

複数セッションで大量のインシデントダンプが発生すると ADR 領域を逼迫する為、フラッド制御機能が設定されている。

・1 時間に同じ問題キーで 5 インシデントか、1 日に 25 のインシデントが発生

　アラートログエントリは作成するが、インシデントダンプは作成しない。

・1 時間に同じ問題キーで 50 インシデントか、1 日に 250 のインシデントが発生

　アラートログエントリ、インシデントダンプを作成しない。記録されなくなった旨のアラートは記録される

ADR の領域のメンテナンス

MMON によって自動的にメンテナンスされる。デフォルトでは以下の期間を過ぎると自動的に削除される。

・インシデントダンプは 30 日間

・インシデントのメタデータ（インシデント関連付けなど）は 1 年間

上記動作を変更するには、EM の［ソフトウェアとサポート］タブ［サポートワークベンチ］－［関連リンク］セクションにある［インシデントパッケージ構成］にアクセスする。

-> サポートワークベンチ

EM －［ソフトウェアとサポート］－［サポートワークベンチ］から「問題」タブで問題を表示させ「インシデント ID」の番号をクリックする。Oracle サポート連携する SR の作成などが可能。

ADRCI コマンドラインツール

ADR へのアクセスは EM または、ADRCI で行うことが可能。OS から adrci を実行することで各コマンドの実行が可能。

ADRCI コマンド	説明
HELP［コマンド］	コマンドのヘルプを表示。引数をつけるとそのコマンドの説明
SET HOME パス	使用する ADR ホームを diag からの相対パスで指定する
SHOW CONTROL	パージ・ポリシー情報
SHOW HOME	現在の ADR ホームを表示。引数に RDBMS を指定すると DB インスタンスの ADR ホームのみ表示
SHOW PROBLEM	問題を表示（-P で指定可能な条件：PROBLEM_ID　PROBLEM_KEY　LAST_INCIDENT　LASTINC_TIM） -LIST 数：指定した最新の数の問題のみ表示 -P 条件　：　指定した条件の問題のみ表示（例：show problem -P "PROBLEM_KEY LIKE '%0600%'"）
SHOW INCIDENT	インシデントを表示する（-P で指定可能な条件：INCIDENT_ID　PROBLEM_KEY　CREATE_TIME）

	-LIST 数：指定した最新の数のインシデントのみ表示 -P 条件 ： 指定した条件のインシデントのみ表示
SHOW ALERT	アラートログを EDITOR 変数で指定されてるエディタで表示する 使用するエディタは SET EDITOR ｴﾃﾞｨﾀ で設定できる -TAIl [数] [-F]：tail コマンドで表示 -P 条件 ：MESSAGE_TEXT、PROBLEM_KEY など指定した条件に一致するアラートログのみ表示
SHOW TRACEFILE	トレースファイル一覧を表示する
SHOW HM_RUN	ヘルスチェックの実行結果を確認

-> インシデントパッケージングサービス（IPS）

インシデントが発生した場合、ADR 内のトレースファイルやダンプファイルに関連付けられたファイルを
ZIP ファイルにパッケージしたものを「インシデントパッケージ」と呼ぶ。

サポートワークベンチの「クイックパッケージング」を利用すると、最小の手順でパッケージングから Oracle サポートへの送信まで行うことが可能だが、事前に Oracle Configuretion Manager が構成されてる必要がある。

ADRCI から IPS コマンドで作成（ADRCI コマンドユーティリティを起動し各コマンドを実行する）

① ロジカル・パッケージの作成

IPS CREATE PACKAGE [オプション]

オプション省略時は空パッケージの作成。コマンド実行後に表示される、create package X の X がパッケージ名。

オプション	説明
INCIDENT ｲﾝｼﾃﾞﾝﾄ ID	インシデント単位
PROBLEMKEY ﾌﾟﾛﾌﾞﾚﾑ KEY	プロブレム KEY 単位
PROBLEM ﾌﾟﾛﾌﾞﾚﾑ ID	プロブレム ID 単位
SECONDS 秒数	指定秒数前までの全インシデント
TIME 開始 TO 終了	指定時刻範囲の全インシデント（時刻はインシデントファイルのフォーマットに合わせて指定）

② ロジカル・パッケージに情報を追加。

IPS ADD <オプション>

オプション	説明
INCIDENT ｲﾝｼﾃﾞﾝﾄ ID PACKAGE ﾊﾟｯｹｰｼﾞ ID	インシデント単位
PROBLEMKEY ﾌﾟﾛﾌﾞﾚﾑ KEY PACKAGE ﾊﾟｯｹｰｼﾞ ID	プロブレム KEY 単位
FILE ﾌｧｲﾙ名 PACKAGE ﾊﾟｯｹｰｼﾞ ID	プロブレム ID 単位

③ フィジカル・パッケージの作成（ロジカル・パッケージを ZIP で固めたもの）

IPS GENERATE PACKAGE ﾊﾟｯｹｰｼﾞ ID IN ﾃﾞｨﾚｸﾄﾘ [INCREMENTAL]

INCREMENTAL を指定した場合は、差分が格納される

その他、IPS コマンド	説明
IPS ADD NEW INCIDENTS PACKAGE	指定したパッケージの問題について新しい最新インシデントを 3 つまで追加 追加されるインシデント数は、IPS パラメータ：NUM_LATE_INCIDENTS で指定
IPS COPY IN FILE	外部ファイルを ADR に追加 追加した外部ファイルをパッケージに追加するには別途 IPS ADD FILE コマンドが必要

IPS COPY OUT FILE	ADR から外部ファイルにコピーする
IPS FINALIZE PACKAGE	指定したパッケージに関連するファイルを追加する。 状態モニターによりヘルスチェックの結果が追加され最新のトレースファイルなどが追加される
IPS SET CONFIGURATION	IPS 標準オプションを設定
IPS SHOW CONFIGURATION	IPS 標準オプションの表示
IPS GET MANIFEST FROM FILE	パッケージの ZIP ファイルからマニフェストを取得して表示
IPS GET METADATA	パッケージの ZIP ファイルからメタデータを抽出して表示します。
IPS REMOVE	既存のパッケージから特定の内容を削除する INCIDENT、PROBLEMKEY オプションでインシデントや問題を指定可能
IPS DELETE PACKAGE	パッケージを削除

状態モニター

自己診断機能として DB ファイル、メモリー、トランザクション、メタデータなどの一貫性に不整合がある場合、
自動的にチェック（ヘルスチェック）が動作する。手動での実行も可能。
ヘルスチェックはルールに従って自動的に実行される。
手動で実行する場合は、EM - [アドバイスセントラル] - [チェッカ]ページや、DBMS_HM パッケージを利用する。
実行結果が「再アクティブ」となっているものは自動でヘルスチェック、「手動」となっているのは手動で実行。

-> DBMS_HM パッケージによるヘルスチェック実行

チェック対象	チェック名（run_name）	説明
DB 構造	DB Structure Integrity Check	アクセス出来ないファイルや破損状態、不整合があるとエラー検出
REDO	Redo Integrity Check	REDO ログと、アーカイブログファイルの破損状態を検出
データブロック	Data Block Integrity Check	ﾁｪｯｸｻﾑｴﾗｰ、ﾌﾞﾛｯｸ内の先頭と末尾の不一致、論理的不整合などのﾌﾞﾛｯｸ破損を検出
トランザクション	Transaction Integrity Check	指定したﾄﾗﾝｻﾞｸｼｮﾝのﾛｰﾙﾊﾞｯｸ操作で発生する可能性のある UNDO 破損を検出 破損が検出されると SMON と PMON を使用し可能な限りﾄﾗﾝｻﾞｸｼｮﾝをﾘｶﾊﾞﾘする
UNDO	Undo Segment Integrity Check	指定した UNDO セグメントの UNDO 破損を検出 破損が検出されると SMON と PMON を使用し可能な限り UNDO ﾃﾞｰﾀをﾘｶﾊﾞﾘする
ディクショナリ	Dictionary Integrity Check	TAB$や COL$などの中心となるディクショナリの整合性を検証する

DB が OPEN/MOUNT 状態であれば全てのヘルスチェックが実行可能。
DB が NOMOUNT 状態の場合は　REDO、DB 構造 のみ実行可能。

DBMS_HM パッケージの実行サンプル

```
DBMS_HM.RUN_CHECK (
    check_name   => 'Dictionary Integrity Check',
    run_name     => 'DicCheck',
    timeout      => 0,
    input_params => 'TABLE_NAME=tab$' );

BEGIN
DBMS_ADVISOR.CREATE_FILE(
    DBMS_HM.GET_RUN_REPORT(
        run_name    => 'DicCheck',
        report_type => 'HTML' ),
    'DATA_PUMP_DIR','hm_DictCheck.html');
END;
/
```

ディクショナリの整合性をチェック
名前をつける（存在する名前があるとエラーになる）
タイムアウトしない
パラメータの指定

ファイル作成
ヘルスチェックのレポート出力

レポート形式は HTML
CREATE DIRECTORY 名前 AS 'パス'; で作成した名前のパスに作成
※DBA_DIRECTORYS で設定されてるディレクトリの一覧が確認できる

ADRCI コマンドでの実行サンプル

adrci> SET HOME diag/rdbms/orcl11/orcl11 adrci> SHOW HM_RUN adrci> CREATE REPORT HM_RUN DicCheck adrci> SHOW REPORT HM_RUN DicCheck adrci> PURGE -AGE 60 TYPE hm	ADR ホームを指定する（複数ある場合はエラーになり実行なるため） ヘルスチェックを確認する レポートを作成 作成したレポートを表示（XML 形式） 60 分経過したヘルスチェック結果は消去される

SQL テストケースビルダー

問題を再現するための再現環境を作成する表と索引の定義（データは含まない）、問題発生の SQL、
オプティマイザ統計、初期化パラメータなどを収集し SQL スクリプトを作成する。

EM - [ソフトウェアとサポート] - [サポートワークベンチ] から 対象の問題を選択し、
[Oracle サポート] タブから「追加的なダンプとテスト・ケースの生成」を選択する。

DBMS_SQLDIAG パッケージでの作成も可能。

DECLARE
　result BOOLEAN;
BEGIN
　result := DBMS_SQLDIAG.**EXPORT_SQL_TESTCASE_DIR_BY_INC** (
　incident_id => 48233,
　directory　=> '/tmp/testcase') ;
END;
/
	ファンクションなので結果を返す変数を宣言
	
	指定したインシデント ID を再現する為の環境を作る
	SQL スクリプトを指定したディレクトリに作成する

SQL 文や SQL_ID を引数にして SQL テストケースを作成する DBMS_SQLDIAG.**EXPORT_SQL_TESTCASE**　プロシージャもある。

データ破損からの保護

ブロック破損の検出方法やリカバリ方法が向上しており、初期化パラメータの調整が一括管理できるようになった。

-> DB_ULTRA_SAFE 初期化パラメータ

「OFF」「DATA_ONLY」「DATA_AND_INDEX」の値を設定することで、
DB_BLOCK_CHECKING、DB_BLOCK_CHECKSUM、DB_LOST_WRITE_PROTECT 初期化パラメータのデフォルト値を設定する。
デフォルトの値の為、各パラメータを明示的に設定することも可能。

初期化パラメータ	OFF	DATA_ONLY	DATA_AND_INDEX
DB_BLOCK_CHECKING	FALSE	MEDIUM	FULL
DB_BLOCK_CHECKSUM	TYPICAL	FULL	FULL
DB_LOST_WRITE_PROTECT	NONE	TYPICAL	TYPICAL

DB_BLOCK_CHECKING

データブロックのチェック（論理的な一貫性のチェック）によりメモリー破損とデータ破損を防止（1～10%の負荷）

ﾊﾟﾗﾒｰﾀ値	説明
OFF (FLASE)	SYSTEM 表領域のみをチェック
LOW	メモリー内でブロックが変更された後（ディスクからの読み取り後や DML の実行後など）ブロックヘッダーをチェック
MEDIUM	LOW のチェックに加え、索引と索引構成表を除くブロックに対するセマンティック（意味）チェックを実行
FULL (TRUE)	MEDIUM のチェックに加え、索引と索引構成表のブロックも含めたセマンティック（意味）チェックを実行

DB_BLOCK_CHECKSUM

DBWn や ダイレクトパス・ローダがデータブロックをディスクに書き込むときにチェックサムを計算し、

ブロックヘッダーに含めるように動作を変更する。こうすることでストレージ破損や I/O システム破損を検出できる。

ﾊﾟﾗﾒｰﾀ値	説明
OFF (FLASE)	SYSTEM 表領域のみチェックサムを検証
TYPICAL	変更が行われ、ディスクへの書込みが行われるときにチェックサムを検証（1～2%の負荷）
FULL (TRUE)	TYPICAL に加え、変更の実行前にもチェックサムを検証（4～5%の負荷） LGWR が REDO ログに書き込む前にも、各 REDO ログブロックのチェックサムを検証

DB_LOST_WRITE_PROTECT

欠落した書込みを検出できる。

フィジカルスタンバイ・データベースでは、プライマリ・データベースから送信された変更が正常に書き込まれたことを検証する必要があるが、実際に書込みが完了していないにもかかわらず、I/O システムがブロック書込みの完了を通知してしまう場合、「欠落した書込み」と呼ばれる。

ﾊﾟﾗﾒｰﾀ値	説明
NONE	欠落した書き込みを検証しない
TYPICAL	読み書き可能（Read Write）表領域について、欠落した書き込みを検証する
FULL (TRUE)	TYPICAL に加え、読込み専用（Read Only）表領域について、欠落した書き込みを検証する

-> V$DATABASE_BLOCK_CORRUPTION ビュー

破損ブロック情報を格納するには BACKUP コマンドで VALIDATE 句を指定するか、一定のしきい値まで破損ブロックを無視してバックアップを行う必要があった。11g からは破損ブロックにアクセスした時点で情報を確認するようになった。

-> VALIDATE コマンド

RMAN の「BACKUP DATABASE VALIDATE」コマンドでデータベースについてのみ破損を検出していた。

11g では VALIDATE 検査対象 コマンドを使用しデータファイルや表領域の破損ブロックも検出できるようになった。

検査対象	検査対象
DATAFILE 'ファイル名'	データファイル
TABLESPACE 表領域名	表領域
SPFILE	現在使用されている SPFILE（コピーされた SPFILE は検査しない）
CURRENT CONTROLFILE	現在使用されている制御ファイル
RECOVERY AREA	現行および前回のすべての高速リカバリ領域の指定先に作成されたリカバリ・ファイル フラッシュバック・ログ、現行の制御ファイルおよびオンライン REDO ログは除外
RECOVERY FILES	ディスク上のすべてのリカバリファイル（フラッシュバックログは除く）

-> 破損ブロックのリカバリ

10g までは「BLOCKRECOVER」コマンドで破損ブロックをリカバリ。

11g では 「RECOVER」コマンドで破損ブロックのリカバリも行う。

リストアするブロックは既存のバックアップセットからだけでなく、フラッシュバック DB ログから取得することも可能。

■リカバリの新機能

SQL 修復アドバイザ

オプティマイザが生成する実行計画が SQL 文の実行時にクリティカルエラーを発生させてしまう場合、

SQL 修復アドバイザを使用し、クリティカルエラーを回避する実行計画をパッチ化することが出来る。

-> SQL 修復アドバイザの実行

EM の［サポートワークベンチ］から問題の詳細ページにアクセスし「SQL 修復アドバイザ」を起動する。

SQL 修復アドバイザタスクが完了し、SQL パッチが作成された場合、詳細を確認し適用することが可能。

パッチ適用有無は［サーバー］タブの［SQL 計画実行］－［SQL パッチ］タブで確認可能。

DBMS_SQLDIAG パッケージでの実行

```
variable tname VARCHAR2(30)
DECLARE
    v_sql CLOB;
BEGIN
    v_sql := '対象の SQL 文';
   :tname := DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK(          SQL 修復アドバイザタスクの作成
         sql_text   => 'v_sql',
         task_name  => 'bug_5869490',
         problem_type =>
             DBMS_SQLDIAG.PROBLEM_TYPE_COMPILATION_ERROR);

DBMS_SQLDIAG.EXECUTE_DIAGNOSIS_TASK(:tname);           SQL 修復アドバイザタスクの実行
END;
/

DBMS_SQLDIAG.ACCEPT_SQL_PATCH(                         SQL パッチの適用
    task_name  => :tname,
    task_owner => 'SCOTT',
    replace    => TRUE);

DBMS_SQLDIAG.DROP_DIAGNOSIS_TASK(:tname):              SQL パッチの削除
```

データリカバリアドバイザ

-> データリカバリアドバイザの概要

現在発生している DB 障害に対し、効率的なリカバリ方法を選択する。

ブロック破損の場合、破損数が少ない場合はブロックメディアリカバリの方が短時間で済むが、

破損数が多い場合は、データファイルのリカバリの方がよい場合がある。最適のリカバリ方法を提示する。

データリカバリアドバイザの対象は、ADR に格納された診断情報で、以下の障害優先度が設定される。

優先度	説明
CRITICAL	DB 全体が使用できない為、直ぐに対応が必要とされる障害。制御ファイル障害など
HIGH	DB の一部が使用できなかったり、リカバリできない状態の為、可能な限り早急な対応が必要。 ブロック破損や、アーカイブログの欠落など
LOW	DB の可用性やリカバリ可能性に影響のない障害。HIGH の優先度を明示的に LOW にすることで一時的に障害を無視できる

データリカバリアドバイザの制限

・RAC 環境はサポートされない

・スタンバイデータベースの障害を診断・修復することは出来ない（スタンバイデータベースへの F/O は可能）

-> EM によるデータリカバリアドバイザの実行（アクセス方法は 3 つ）

・[可用性]タブにある［リカバリの実行］から[アドバイスとリカバリ］

・[ソフトウェアとサポート]タブにある［サポートワークベンチ］の［問題]ページの[チェック結果]タブから
［リカバリアドバイザの起動］をクリック

・関連リンク［アドバイザ・セントラル］－［アドバイザ]タブから［データ・リカバリ・アドバイザ]をクリック

-> RMAN コマンドの仕様

コマンド	説明
LIST FAILURE	自動診断リポジトリ（ADR）内の障害を一覧表示する
ADVISE FAILURE	障害に対し、推奨される修復オプションを表示し、可能であれば修復スクリプトを作成する
REPAIR FAILURE	ADVISE FAILURE で作成された修復スクリプトを使用し障害を修復し、障害をクローズする
CHANGE FAILURE	既知の障害の優先度を変更（CRITICAL を HIGH や LOW にはできない）したり、明示的にクローズする

LIST FAILURE [{ALL | CRITICAL | HIGH | LOW | CLOSE | 障害 ID ﾘｽﾄ}] [EXCLUDE FAILURE 障害 ID ﾘｽﾄ] [DETAIL]
デフォルトでは CRITICAL と HIGH の障害のみ表示。EXCLUDE で除外する障害を指定。DETAIL で詳細表示。

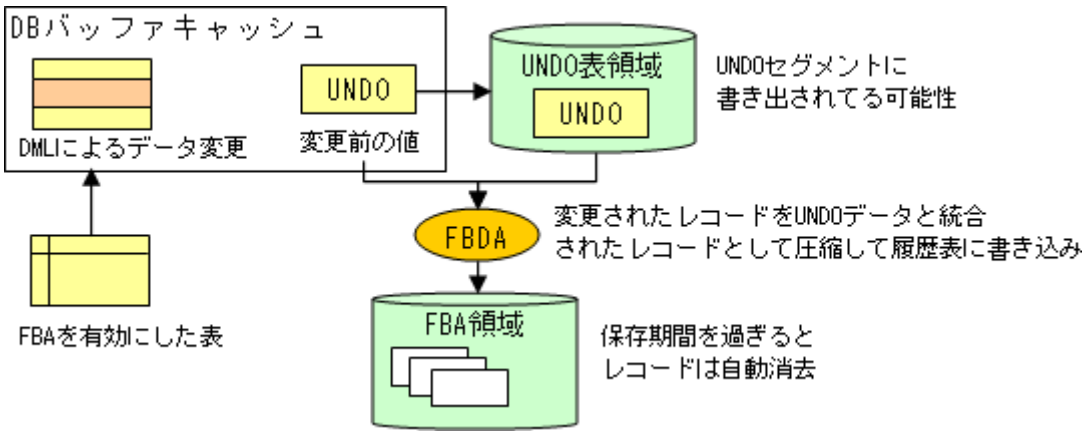
ADVISE FAILURE [{ALL | CRITICAL | HIGH | LOW | CLOSE | 障害 ID ﾘｽﾄ}] [EXCLUDE FAILURE 障害 ID ﾘｽﾄ]
同じセッションで LIST FAILURE コマンドを実行してないとエラーとなる。
最後に実行した LIST FAILURE コマンド以降に新しい障害が発生してる場合、実行前に WARNING が表示される。

REPAIR FAILURE [USING ADVISE OPTIONS 修復 ID] [PREVIEW] [NOPROMPT]
PREVIEW は実際の修復を行わず修復スクリプトのみ表示する。
同じセッションで ADVISE FAILURE が実行されてる必要がある。実行してないとエラーになる。

CHANGE FAILURE {ALL | CRITICAL | HIGH | LOW | CLOSE | 障害 ID ﾘｽﾄ} [EXCLUDE FAILURE 障害 ID ﾘｽﾄ]
{CLOSED | PRIORITY {HIGH | LOW}} [NOPROMPT]
優先度 CRITICAL の障害の優先度変更はできない。

フラッシュバックデータアーカイブ（FBA）

DB に時間軸の概念を追加した機能で、リカバリ処理をすることなくテーブルの履歴を管理出来る機能。
従来のフラッシュバック機能と違い、UNDO データよりも長い期間履歴データを保存することが可能。
FBA を有効化した表のレコードは FBA 領域に UNDO データと統合したレコードとして圧縮して格納される。
FBA 領域への書き込みは FBDA バックグラウンドプロセスによって非同期に行われる。
FBA によって保存された履歴データは変更できないように構成されている。
自動 UNDO 管理を使用している必要があり、手動管理だと作成する際にエラーとなる。
アーカイブ化する表領域は自動セグメント領域管理（ASSM）の必要がある。



-> フラッシュバックデータアーカイブの設定

フラッシュバックデータアーカイブに関する権限

実行内容	必要な権限
FBA 領域の作成、管理、削除	FLASHBACK ARCHIVE ADMINISTTER 権限 デフォルトの FBA 領域を作成する場合は、SYSDBA 権限 FBA 領域領域を格納する表領域に対する領域割り当て制限（QUOTA）
表に対する FBA 有効化	FBA 領域に対する FLASHBACK ARCHIVE 権限 他スキーマの場合は表に対する ALTER オブジェクト権限（または ALTER ANY TALBE システム権限）
表に対する FBA 無効化	FLASHBACK ARCHIVE ADMINISTTER 権限
フラッシュバッククエリー	他スキーマの場合は、表に対する SELECT オブジェクト権限（または ALTER ANY TALBE システム権限） FLASHBACK オブジェクト権限（または FLASHBACK ANY TABLE システム権限）

フラッシュバックデータアーカイブ領域の作成

作成：CREATE FLASHBACK ARCHIVE [DEFAULT] FBA 領域名 TABLESPACE 表領域名 [QUOTA ｻｲｽﾞ]
　　RETENTION 保存期間 {YEAR | MONTH | DAY}

削除：DROP FLASHBACK ARCHIVE FBA 領域名

オプション	説明
DEFAULT	明示的に FBA 領域を指定しない場合に使用される FBA 領域を作成する
QUOTA	省略した場合、無制限になるが作成するユーザーが持つ表領域の割り当て制限を超えているとエラーになる
RETENTION	指定した期間は削除されず、過ぎたら削除可能となり履歴削除機能により実際に削除される

フラッシュバックデータアーカイブの有効化

有効：{ CREATE | ALTER } TABLE 表名 FLASHBACK ARCHIVE [FBA 領域名]

無効：ALTER TABLE 表名 NO FLASHBACK ARCHIVE

デフォルトの FBA 領域がない場合、FBA 領域名を省略したらエラーになる。

LONG 型、LONG RAW 型を持つテーブルに対して有効化できない。

フラッシュバックデータアーカイブの利用

フラッシュバック問い合わせ（AS OF { SCN ｼｽﾃﾑ変更番号 | TIMESTAMP 時間 }）をすると、FBA が自動的に使用される。

FBA クエリーによる副問い合わせを DML の値として使用しレコードをリカバリすることも可能。

FBA を有効化した表の DDL 制限（実行するとエラー）

・表の削除（DROP TABLE 文）

・以下何れかの表変更（ALTER TABLE 文）

　パーティション、サブパーティションの移動及び、EXCHANGE 操作。

　オブジェクト型や列のバージョンアップ（ALTER ～ UPGRADE TABLE 句）

上記制限は 11gR2 のもので、11gR1 は更に以下の制限がある

・以下何れかの表変更（ALTER TABLE 文）

　列の削除（DROP COLUMN 句）、列名の変更（RENAME COLUMN 句）、列定義の変更（MODIFY 句）、表名の変更（RENAME TO 句）

・表の変更（RENAME TABLE 文）、切り捨て（TRUNCATE TABLE 文）

-> フラッシュバックアーカイブの管理

フラッシュバックデータアーカイブに関するビュー

ビュー名	説明
DBA_FLASHBACK_ARCHIVE	デフォルトの FBA 領域や保存期間など FBA 領域に関する情報を表示
DBA_FLASHBACK_ARCHIVE_TS	FBA 領域が格納されている表領域名を表示。QUOTA の確認など
DBA_FLASHBACK_ARCHIVE_TABLES	FBA が有効化されているテーブル表示

フラッシュバックデータアーカイブの設定変更

内容	SQL 文
デフォルト FBA 領域の変更	ALTER FLASHBACK ARCHIVE FBA 領域名 SET DEFAULT
FBA 領域に表領域を追加	ALTER FLASHBACK ARCHIVE FBA 領域名 ADD TABLESPACE 表領域名 [QUOTA ｻｲｽﾞ]
FBA 領域の表領域を削除	ALTER FLASHBACK ARCHIVE FBA 領域名 REMOVE TABLESPACE 表領域名 ※最初の表領域は削除不可、表領域を削除することでデータも削除される
表領域のサイズ変更	ALTER FLASHBACK ARCHIVE FBA 領域名 MODIFY TABLESPACE 表領域名 [QUOTA ｻｲｽﾞ]
保存期間の変更	ALTER FLASHBACK ARCHIVE FBA 領域名 MODIFY RETENTION 保存期間 {YEAR | MONTH | DAY}
履歴データの消去	ALTER FLASHBACK ARCHIVE FBA 領域名 PURGE { ALL | BEFORE {TIMESTAMP 時間 | SNC SCN} }

フラッシュバックトランザクションバックアウト（FBB）

10g のフラッシュバック表では表全体を過去の一時点に戻す為、指定した時点以降のトランザクションが全てロールバックしていたが、FBB を使うことで特定のトランザクションのみバックアウト（過去に戻す）ことが可能になった。

-> フラッシュバックトランザクションバックアウト前提条件

・ARCHIVELOG モードであること

・サプリメンタルロギングの有効化（最小サプリメントロギング：ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;）

・DBMS_FLASHBACK パッケージの実行権限

・SELECT ANY TRANSCATION システム権限

-> 依存トランザクション

FBB の対象となるトランザクションと同じレコードを対象としたレコードのこと（下記の例では NO=30）

元の表（DEPT）

NO	NAME	LOC
10	田中	群馬
20	佐藤	埼玉
30	川澄	北海道

```
UPDATE DEPT SET LOC='新潟' WHERE NO=30;
INSERT INTO DEPT VALUES(25,'竹宮','東京');
```

NO	NAME	LOC
10	田中	群馬
20	佐藤	埼玉
25	竹宮	東京
30	川澄	新潟

```
UPDATE DEPT SET NAME='月宮'
WHERE NO=30;
```

NO	NAME	LOC
10	田中	群馬
20	佐藤	埼玉
25	竹宮	東京
30	月宮	新潟

FBB の際に依存トランザクションのある場合は、以下の何れかの扱いを指定する

オプション	説明
NOCASCADE	依存トランザクションがない場合のみトランザクションをバックアウトする（プロシージャのデフォルト動作）
NONCONFLICT_ONLY	依存トランザクションと競合しない SQL 文のみバックアウトする（EM でのデフォルト動作）
NOCASCADE_FORCE	指定したトランザクションのみバックアウトする
CASCADE	指定したトランザクションと依存トランザクションをバックアウトする

トランザクション 1 をバックアウトした場合の動作

NONCONFLICT_ONLY

NO	NAME	LOC
10	田中	群馬
20	佐藤	埼玉
25	竹宮	東京
30	月宮	新潟

NOCASCADE_FORCE

NO	NAME	LOC
10	田中	群馬
20	佐藤	埼玉
25	竹宮	東京
30	月宮	北海道

CASCADE

NO	NAME	LOC
10	田中	群馬
20	佐藤	埼玉
25	竹宮	東京
30	川澄	北海道

NONCONFLICT_ONLY	トランザクション 1 の INSERT 文のみバックアウトする トランザクション 1 の UPDATE 文はトランザクション 2 の UPDATE 文と競合する為、バックアウトしない
NOCASCADE_FORCE	トランザクション 1 の UPDATE 文と INSERT 文のみバックアウトされる トランザクション 2 の UPDATE 文は実行されたまま
CASCADE	トランザクション 1 とトランザクション 2 がバックアウトされる

-> フラッシュバックトランザクションバックアウトの実行

EM での実行

[スキーマ]タブから [表]ページにアクセスし、対象となる表を選択した状態で[アクション]リストボックスから、
[トランザクションのフラッシュバック]を選択する。

DBMS_FLASHBACK.TRANSACTION_BACKOUT プロシージャでの実行

トランザクション ID（VERSIONS_XID）は以下の SQL 文で確認可能

```
SELECT VERSIONS_XID, VERSIONS_STARTSCN, VERSIONS_ENDSCN, 列 [,列 ...] FROM 表名
VERSIONS BETWEEN { SCN | TIMESTAMP } 最小値 AND 最大値
```

最小値と最大値には UNDO データとして保存されている最小値（MINVALUE）最大値（MAXVALUE）の指定が可能

```
DECLARE
    v_xidarr XID_ARRAY;
BEGIN
   v_xidarr := XID_ARRY();
   v_xidarr.EXTEND(1);
   v_xidarr(1) := '04000700A2020000';
   DBMS_FLASHBACK.TRANSACTION_BACKOUT(
       numtxns => 1 ,
       xids    => v_xidarr,
       options => DBMS_FLASHBACK.NOCASCADE_FORCE);
END;
/
```

バックアウトさせるトランザクション ID を指定する

配列の形式でのトランザクション識別子のリスト
依存トランザクションがあった場合の処理を指定

Enterprise Manager による LogMiner

EM - [可用性]タブから [トランザクションの表示と管理]

LogMiner は SQL インタフェースを介して REDO ログ・ファイル（オンラインおよびアーカイブ）を問い合せることができる。
EM による LogMiner はフラッシュバックトランザクションバックアウトと統合されている為、以下の条件が必要。

・ARCHIVELOG モード
・サプリメンタルロギングの有効化
・DBMS_LOGMNR パッケージの実行権限
・SELECT ANY TRANSACTION システム権限

■リカバリの新機能

セキュアなパスワード

-> 一般ユーザにおけるパスワード大文字/小文字の区別

SEC_CASE_SENSITIVE_LOGIN 初期化パラメータが TRUE（デフォルト）の場合に区別される。

ユーザー名に関しては大文字/小文字は区別されない。

アップグレードした DB の場合、既存ユーザーのパスワードは明示的に変更するまで大文字/小文字を区別しない。

-> パスワードプロファイルの拡張

DEFAULT プロファイルにおけるパスワード設定のデフォルト値が追加された。

リソース名	説明	10gR2	11g
FAILED_LOGIN_ATTEMPTS	最大ログイン失敗数	10	10
PASSWORD_LOCK_TIME	アカウントロック期間（日）	UNLIMITED	1
PASSWORD_LIFE_TIME	有効期限	UNLIMITED	180
PASSWORD_GRACE_TIME	猶予期間	UNLIMITED	7
PASSWORD_REUSE_TIME	再利用に必要な期間	UNLIMITED	UNLIMITED
PASSWORD_REUSE_MAX	再利用に必要な変更回数	UNLIMITED	UNLIMITED
PASSWORD_VERIFY_FUNCTION	検証ファンクション	NULL	NULL

パスワード検証ファンクションを作成する utlpwdmg.sql も中身が変更されている。

utlpwdmg.sql を実行すると VERIFY_FUNCTION と VERIFY_FUNCTION_11g の 2 つのファンクションが作成され、

DEFAULT プロファイルの PASSWORD_VERIFY_FUNCTION に VERIFY_FUNCTION_11g が割り当てられる。

検証条件	VERIFY_FUNCTION	VERIFY_FUNCTION_11g
最小パスワード文字数	4 文字以上	8 文字以上
ユーザーに関する 禁止パスワード	ユーザー名と同じ	ユーザー名と同じ ユーザー名に 1～100 を追加 ユーザー名を逆転
禁止されてる文字列	welcome database account user password oracle computer abcd	oracle の文字列に 1～100 を追加 welcome1 database1 password1 oracle123 computer1 abcdefg1 change_on_install
パスワード変更要件	3 文字以上	3 文字以上

-> 特権ユーザー認証におけるパスワード大文字/小文字の区別

$ orapwd file=ﾌｧｲﾙ名 password=ﾊﾟｽﾜｰﾄﾞ entries=登録ﾕｰｻﾞ数 **ignorecase**={ y | n } force={ y | n }

ignorecase=n にすることで大文字/小文字が区別される。デフォルトは n になってる。

force=y にすることで、既にファイルが存在いた場合でも上書きする。

-> 厳密な認証の追加

特権ユーザーは OS 認証 -> パスワードファイル認証 -> 厳密な認証 の優先順位で認証が行われる。

厳密な認証には次の認証方式がある。

・OID（Oracle Internet Directory）による SYSDBA、SYSOPER エンタープライズロールの使用

・Kerberos チケットの使用

・SSL 証明書の使用

厳密な認証を行うには上記認証方式の何れかを構成した上で次の条件を満たす必要がある。

・LDAP_DIRECTORY_SYSAUTH 初期化パラメータを YES にする

・LDAP_DIRECTORY_ACCESS 初期化パラメータを PASSWORD または SSL の何れかにする

・OS 認証となる OSDBA グループと、OSOPER グループに所属させない

・パスワードファイルに登録されてないユーザー名とパスワードにする

セキュリティの強化

-> データベース接続のセキュリティ

DoS 攻撃と、ブルートフォース攻撃から防御する初期化パラメータが追加された。

初期化パラメータ	
SET_PROTOCOL_ERROR_ FURTHER_ACTION	悪意をもつクライアントから不正パケットを受信した時の動作を指定 CONTINUE　接続を継続する DELAY,秒　次の要求を受け入れる前に指定した秒の遅延を発生させる DROP,数　指定した数の不正パケット受信後、そのクライアントの接続を強制切断する
SEC_PROTOCOL_ERROR_ TRACE_ACTION	悪意をもつクライアントから不正パケットを受信した時の監視アクションを指定 NONE　不正パケットを無視する（何もログを生成しない） TRACE　トレースファイル（サーバープロセスのトレースファイル）を作成する LOG　アラートログとトレースファイルに最低限のメッセージを表示する ALERT　アラートメッセージを監視コンソールに送信する
SEC_MAX_FAILED_LOGIN_ ATTEMPTS	指定した回数ログインが失敗した場合、接続を切断する（特権ユーザは除外） プロファイルのログイン試行失敗と異なり存在しないユーザーも対象

-> デフォルトの監査

EM -［サーバー］-［監査設定］から監査内容の設定や、監査証跡の確認が可能。

AUDIT_TRAIL 初期化パラメータにてデータベースの監査を使用可能または使用禁止する。

デフォルトは「DB」に設定され、監査証跡は SYS.AUD$表に記録される。

OS や XML にすることで、OS ファイル（場所は AUDIT_FILE_DEST 初期化パラメータで設定）に格納可能。

ネットワークコールアウトに対する制御

-> ネットワークコールアウトの概要

ネットワークを利用する UTL_TCP、UTL_HTTP、UTL_SMTP、UTL_MAIL などのパッケージは TCP や HTTP などのプロトコルを使用して指定したホストへのコールアウト（呼びだし）が可能だが、制限するにはパッケージの実行権限のみだった。

DBMS_NETWORK_ACL_ADMIN パッケージを使用することで、使用可能なホストを細かく制御できるようになった。

-> DBMS_NETWORK_ACL_ADMIN パッケージ

パッケージを使用して制御を行うには ACL を作成し、ユーザーまたはロールと権限を設定し、

1 つ以上のネットワークホストに ACL を割り当てることで制限が行われる。

ACL の作成と権限設定

```
DBMS_NETWORK_ACL_ADMIN.CREATE_ACL(
   acl         => 'perm.xml',
   description=> 'ネットワーク接続',
   principal   => 'SCOTT',
   is_grant    => TRUE,
   privilege   => 'connect',

   start_date => SYSTIMESTAMP,
   end_date    => SYSTIMESTAMP+1);
```

```
ACL の作成
ACL の名前。相対パスの起点は/sys/acls

対象のプリンシパル（データベース・ユーザーまたはロール）
TRUE(許可) または FALSE(拒否) を指定
制御する権限を指定、connect(外部接続) または resolve(名前解決)
  ※connect は resolve の権限も含まれている
ACL 制御を開始する日
ACL 制御を終了する日
```

DBMS_NETWORK_ACL_ADMIN.**ASSIGN_ACL**(acl => 'perm.xml', host => '*.oracle.co.jp', lower_port => 80, upper_port => NULL);	ACL の割り当て ACL の名前 ACL の割当て先のホスト（ホスト名か IP アドレスで指定。*の使用可能） TCP ポート範囲の下限（NULL で無制限） TCP ポート範囲の上限（省略または NULL の時は lower_port と同じ）

割り当てられた定義は、DBA_NETWORK_ACLS ビュー確認可能

透過的データ暗号化の拡張

-> 表領域の暗号化

10gR2 では列レベルの宣言のみ可能だったが、11g では表領域レベルで宣言できるようになった。

透過的データ暗号化はデータ挿入時に自動的に暗号化され、取得時に復号される。

データファイルが盗難されたとしても別ファイルであるウォレットに格納されたマスター鍵がなければ復号化できない。

表レベルの暗号化は永続表領域（PERMANENT）のみ設定可能。

一時表領域（TEMPORARY）や UNDO 表領域を暗号化することはできないが、

暗号化された表領域内のデータを使用する際は UNDO データや一時データ、REDO ログエントリも自動的に暗号化される。

-> 表領域の暗号化設定

① $ORACLE_HOME/network/admin/sqlnet.ora に ウォレットの作成場所を記述する。

```
ENCRYPTION_WALLET_LOCATION=
   (SOURCE=(METHOD=FILE)(METHOD_DATA=
   (DIRECTORY=ウォレットの作成ディレクトリ)))
```

② マスター鍵の作成

```
ALTER SYSTEM SET KEY IDENTIFIED BY "パスワード"
```

sqlnet.ora で指定したディレクトリに既にウォレットが存在する場合はウォレットがオープンされ鍵が再作成される。

③ ウォレットをオープンする（OPEN させないと問い合わせと DML が実行できない）

```
ALTER SYSTEM SET WALLET OPEN IDENTIFIED BY "パスワード"
```

ALTER SYSTEM SET WALLET CLOSE で ウォレットを CLOSE することが可能

④ 暗号化を有効にした表領域の作成

表領域作成時に、ENCRYPTION 句と、DEFAULT STORAGE(ENCRYPT)句を使用し暗号化する。

```
create tablespace ... encryption [using '{3DES168 | AES128 | AES192 | AES256}'] default storage(encrypt)
```

暗号化アルゴリズムを省略した場合は、AES128 が使用される。

⑤ 透過的データ暗号化（TDE）を有効化した表領域にテーブルを作成する

既存の表を TDE に変更することは出来ない為、既存の表を TDE 化したい場合は、TDE を有効化した表領域に移動させる。

BFILE 型はデータが外部に格納される為、暗号化できない。

-> 透過的データ暗号化の機能拡張

LogMiner のサポート

透過的データ暗号化は REDO ログエントリも自動的に暗号化される。

10g では LogMiner の暗号化列を含むオブジェクトを確認すると暗号化されたエントリが表示されていた。

11g では復号化して確認出来るようになった。

ロジカルスタンバイ DB や Streams では REDO ログを暗号化せずに送信し、スタンバイ側のウォレットで暗号化を行う。

ハードウェアベースのマスター鍵の保護

マスター鍵の格納場所としてウォレットの他にハードウェアセキュリティモジュールをサポートしている。

Oracle SecureFiles

-> SecureFiles の概要

大規模なデータを扱う LOB データ型（BFILE、BLOG、CLOB、NCLOB）の新しい管理方式として追加された。

利点としては、パフォーマンスの向上。格納データの圧縮。重複データの除外。暗号化があげられる。

従来通りの管理方式を「BasicFile LOG」と呼ぶ。

SecureFile LOB は自動セグメント領域管理（ASSM）の表領域が必要で、それ以外の表に作成するとエラーになる。

-> SecureFiles の作成

CREATE TABLE ～ LOB(列名) STORE AS { SECUREFILE (SecureFiles 属性) | BASICFILE (BasicFile 属性) } (ストレージ属性,...)

SecureFiles 属性	説明
NOCOMPRESS | COMPRESS	LOB セグメントを圧縮するか（ HIGH | MEDIUM | LOW ）
KEEP_DUPLICATE | DEDUPLICATE	同じ LOB データを削除するか（重複レコードの削除有無）
DECRYPT | ENCRYPT	透過的データ暗号化による暗号化（ USING 'アルゴリズム' IDENTIFIED BY 'パスワード' ）
RETENSION	旧バージョンに関する保存ポリシー MAX ：LOB セグメントの最大値に達するまで旧バージョンを保持 MIN ：指定した保存期間(秒)は旧バージョンを保持 NONE：可能な限り旧バージョンを利用 AUTO：保存期間と領域を考慮し効率的に管理
共通属性	**説明**
CACHE | NOCACHE	DB バッファキャッシュ使用有無 NOCACHE は共有 I/O プールが使用される（DB キャッシュから必要に応じて最大 4%まで取得する）
LOGGING | NOLOGGING	LOB 更新時に REDO ログを生成するか
BasicFile LOB 属性	**説明**
PCTVERSION	LOB ﾃﾞｰﾀの保持の割合最大値（ﾃﾞﾌｫﾙﾄの 10 であれば 10%を越えるまで旧ﾊﾞｰｼﾞｮﾝは上書きされない）
FREELISTS、FREEPOOLS FREELIST GROUPS	LOB セグメントで使用する空きリストと、空きリストグループ
CHUNK	LOB 操作に使用される単位

（例）SecureFile LOB の作成

```
CREATE TABLE emp1 (
    empno NUMBER(4),
    photo BLOB)
LOB(photo) STORE AS SECUREFILE
    (COMPRESS HIGH
     DEDUPLICATE
     ENCRYPT USING '3DES168'
     CACHE
     LOGGING)
STORAGE( MAXSIZE 64M )
```

SecureFile LOB の作成（STORE AS BASICFILE だと BasicFile LOB になる）
圧縮率（高）による圧縮
重複除外
透過的暗号化（ウォレットが OPEN してないとエラー）
キャッシュに DB バッファキャッシュを使用
REDO ログを生成する
LOB セグメントの最大サイズを指定

-> SecureFile LOB の設定

LOB の管理方式は DB_SECUREFILE 初期化パラメータによって決定する。

パラメータ	説明

PERMITTED	STORE AS 句の指定通りに作成。指定した属性で違反がある場合はエラー（デフォルト）
ALWAYS	SecureFile LOB を作成する。LOB セグメントを格納刷る表領域が ASSM でない場合、BasicFile LOB が作成される
FORCE	SecureFile LOB を作成する。LOB セグメントを格納刷る表領域が ASSM でない場合、エラーとなる
NEVER	BasicFile LOB を作成する。SecureFile LOB 固有属性が指定されてる場合はエラーになる
IGNORE	BasicFile LOB を作成する。SecureFile LOB 固有属性が指定されてる場合、固有属性を無視して作成

-> SecureFile への移行

SecureFile LOB 固有属性を ALTER 文で変更することは可能。

BasicFile LOB として作成した LOB を SecureFile LOB に変更するには

オンライン再定義（DBMS_REDEFINITION パッケージ）を使用する。

パーティション表の場合は EXCHANGE PARTITION も可能だが、交換中の DML 文が受け付けられない。

パーティション交換	オンライン再定義
表パーティション毎に交換できる 表パーティションと同じ追加領域が必要 交換後も索引が維持できる（UPDATE INDEXES） 移行中、表パーティションデータは使用不可	表全体を交換する 表全体と同じ追加領域が必要 全てのグローバル索引の再構築が必要 移行中、データの使用が可能 パラレル実行が可能

その他の新機能

オブジェクト管理の拡張機能

-> オンライン再定義

DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS プロシージャを利用して、MV や MV ログのコピーが可能になった。

ただし、オンライン再定義完了後、MV は無効化される為、完全リフレッシュを行う必要がある。（高速はエラーになる）

ビュー、シノニム、プロシージャ、ファンクションは無効化されなくなった。

トリガーは無効化されるが、次回の DML で自動的に再コンパイルされる。

-> 依存関係に基づく再コンパイルの最小化

ビューや、ストアドプロシージャの間接依存が無効化されなくなった。

T1 表を参照するビューV1 があり、T1 に列を追加した場合、V1 は追加した列を参照してない為、無効化されない。

V1 を使用するファンクションや、プロシージャは T1 表に間接依存しているが無効化されない。

パッケージ仕様にユニットを追加しただけの場合、無効化されなくなった。

ファンクション（F1）が定義されたパッケージ仕様があり、F1 を使用するプロシージャ（P1）があった場合、

パッケージ仕様に新しいファンクション（F2）を追加しても、P1 は無効化されない。

-> ネイティブコンパイルの拡張

PL/SQL プログラムを C 言語のコードに変換後、共有ライブラリとして、Oracle プロセスにリンクされるため、

PL/SQL エンジン上でインタープリタで動作するより処理が高速。

ネイティブコンパイルは PL/SQL と Java コードで行うことが可能。

PL/SQL のネイティブコンパイルは外部の C コンパイラが不要になった。

ネイティブコンパイル済みのコードはデータベース内に格納される。

初期化パラメータ	値	説明

JAVA_JIT_ENABLED	TRUE* | FALSE	Java ストアドプログラムの JIT（Just-in-time）コンパイルを制御
PLSQL_CODE_TYPE	INTERPRETED* | NATIVE	PL/SQL ストアドプログラムのネイティブコンパイルの制御

PLSQL_CODE_TYPE を NATIVE にし PL/SQL をコンパイルするか、

ALTER PROCEDURE ﾌﾟﾛｼｰｼﾞｬ名 COMPILE PLSQL_CODE_TYPE = NATIVE にてネイティブコンパイルを実行する。

ロックメカニズムの拡張機能

-> LOCK TABLE 文の待機

ロックを取得できなければ即時に制御を戻す（NOWAIT 句）と、ロック待ちの時間を指定する（WAIT 句）は、

SELECT ... FOR UPDATE 文であれば指定可能だったが、11g では LOCK TABLE 文（意図的にロックする）でも可能になった。

ロックした表は、COMMIT または ROLLBACK することでロック解除される。

LOCK TABLE ﾃｰﾌﾞﾙ名 IN EXCLUSIVE MODE [NOWAIT | WAIT 秒]

MODE	動作
指定なし	表ロックが取得できるまでロック待ち
NOWAIT	表ロックが取得できなければリソースビジーで制御を戻す
WAIT 秒	指定した秒数ロックを待ち、それでも取得できない場合はリソースビジーで制御を戻す

-> DDL 文の待機

ALTER などの DDL 文は DML などで表ロックが取得されていると即座にリソースビジーが発生していた。

11g では、DDL_LOCK_TIMEOUT 初期化パラメータで待機時間を制御することが可能になった。

デフォルトは 0 で即時にリソースビジーとなる。待機は 1000000 秒まで指定可能。

非表示の索引

索引を非表示にすると実行計画で索引が使われなくなる。

OPTIMIZER_USE_INVISIBLE_INDEXES 初期化パラメータが TRUE（デフォルトは FALSE）または、

INDEX ヒント句と、USE_INVISIBLE_INDEXES ヒント句を同時に使用した場合は、オプティマイザで使用される。

索引を削除や、使用不可にした場合、再び索引を使えるようにするには、再作成や、再構築を行う必要があるが、

非表示索引は、表示/非表示を切り替えるだけ。

-> 非表示の索引の設定

SQL 文	説明
CREATE INDEX ... INVISIBLE	非表示索引の作成
ALTER INDEX 索引名 INVISIBLE	非表示索引に変更
ALTER INDEX 索引名 VISIBLE	表示索引に変更

結果キャッシュ

-> SQL 問い合わせ結果のキャッシュ

SELECT 文の結果データを共有プールにキャッシュする。

次回以降、元レコードが格納されているデータブロックが変更されるまで結果キャッシュの利用が可能。

RAC の場合はインスタンス毎に結果キャッシュが作成される為、同一インスタンスに接続してるクライアントが利用可能。

select * from v$sgastat where name like '%Result%' で結果キャッシュの利用量が確認できる。

初期化パラメータ	説明
RESULT_CACHE_MODE	SQL 問い合わせで結果キャッシュを使用するか MANUAL：RESULT_CACHE ヒント句を利用しない限り、結果キャッシュを使用しない（デフォルト） FORCE ：SELECT 文で結果キャッシュを利用できるようにする
RESULT_CACHE_MAX_SIZE	共有プール内で結果キャッシュに使用できる最大サイズ（byte） 0 にすると無効。 デフォルトは SGA 構成依存（SHARED_POOL_SIZE の 1%、SGA_TARGET の 0.5%、MEMORY_TARGET の 0.25%） 共有プールの 75%以上の設定はできない
RESULT_CACHE_MAX_RESULT	結果キャッシュ内（RESULT_CACHE_MAX_SIZE）で確保される各結果キャッシュ最大サイズ（共有プールの 5%）
RESULT_CACHE_REMOTE_EXPIRATION	DATABASE LINK を使用したリモートオブジェクトに対する結果キャッシュの保持時間（分） デフォルトは 0 で結果のキャッシュを行わない。 キャッシュ保持時間内に変更されたデータがある場合、変更前のデータにアクセスしてしまう。

RESULT_CACHE_MODE が MANUAL 時に明示的に結果キャッシュを利用するには /*+ RESULT_CACHE */ ヒント句を利用する。

RESULT_CACHE_MODE が FORCE 時に明示的に結果キャッシュを無効にするには /*+ NO_RESULT_CACHE */ ヒント句を利用する

結果キャッシュが有効な時は、V$RESULT_CACHE_OBJECTS ビューの STSTUS 列が「Published」になる。

SCAN_COUNT 列で結果キャッシュが使用された回数が確認できる。

DBMS_RESULT_CACHE パッケージで結果キャッシュの操作が可能。

ﾌﾟﾛｼｰｼﾞｬ/ﾌｧﾝｸｼｮﾝ	説明
BYPASS	一時的に結果キャッシュの使用有無を操作する。TRUE で結果キャッシュを使用せず、FALSE で使用する
FLUSH	結果キャッシュをフラッシュする。RETAINMEM を TRUE でキャッシュの空きメモリーを保持 RETAINSTA を TRUE で既存のキャッシュ統計を保持。デフォルトは両方 FALSE なので全てのキャッシュを削除
INVALIDATE	指定したオブジェクトに依存する結果キャッシュを無効化する
MEMORY_REPORT	結果キャッシュのメモリー使用状況をレポートする。SET SERVEROUTPUT ON が行われている必要がある

次の項目を含む SELECT 文では結果キャッシュが無効になる。

・一時表、ディクショナリビュー、SYS または SYSTEM スキーマ内のオブジェクト

・順序の CURRVAL と NEXTVAL 疑似列

・実行の度に結果が変わるファンクションを利用している場合（CURRENT_DATE、SYSDATE など）

-> ファンクション結果キャッシュ

ファンクションの実行結果のみをキャッシュする機能。

ファンクションの結果キャッシュを利用するには、ファンクションに RESULT_CACHE 句を指定する。

```
CREATE OR REPLACE FUNCTION sales_count(pid NUMBER)
RETURN NUMBER
RESULT_CACHE RELIES_ON (sales)
IS
    ret NUMBER;
BEGIN
    SELECT count(*) INTO ret FROM sales
    WHERE prod_id=pid;
    RETURN ret;
END;
/
```

sales_count ファンクションの作成

ファンクションの結果キャッシュを利用する
RELIES_ON 句で依存する表を指定することで、指定した表のデータが更新された時にファンクションの結果キャッシュを無効化できる

count の結果を ret 変数に代入する

ファクション実行例

```
select sales_count(20) from dual;
```

次の条件に一致するファンクションは結果キャッシュを利用できない

・実行者権限を持つモジュール内または無名ブロック内で定義されていない

・パイプライン・テーブル・ファンクションでない

・OUT パラメータまたは IN OUT パラメータを含んでいない

・次のいずれの型の IN パラメータも含んでいない

BLOB、CLOB、NCLOB、REF CURSOR、オブジェクト、コレクション、レコード

·戻り型が次のいずれでもない

BLOB、CLOB、NCLOB、REF CURSOR、オブジェクト

-> OCI クライアント結果キャッシュ

クライアント側のメモリに結果キャッシュを保持する。サーバーの結果キャッシュと同時に使用することも可能。

クライアント結果キャッシュを使用するには初期化パラメータまたは sqlnet.ora による構成の 2 パターンある。

初期化パラメータ	説明
CLIENT_RESULT_CACHE_SIZE	クライアント結果キャッシュとして使用可能な最大サイズ。0 は無効（デフォルト）
CLIENT_RESULT_CACHE_LAG	クライアント結果キャッシュに反映させる必要のあるサーバー側の変更を遅延できる最大時間（ミリ秒） デフォルトは 3000

sqlora.net パラメータ	説明
OCI_RESULT_CAHCE_MAX_SIZE	クライアント結果キャッシュ最大サイズ（byte）サーバの設定を上書きする
OCI_RESULT_CAHCE_MAX_RSET_SIZE	クライアント結果キャッシュに格納される結果セットの最大サイズ（byte）
OCI_RESULT_CAHCE_MAX_RSET_ROWS	クライアント結果キャッシュに格納される結果セットの最大サイズ（レコード数）

一時表領域の拡張機能

-> 一時表領域の縮小

従来は一時表領域の縮小は再作成が必要だったが、11g では縮小させることが可能になった。

一時表領域のエクステント管理が「ローカル管理」の場合に可能。

DBA_TEMP_FREE_SPACE ビューが追加され一時表の空き容量が確認できる。

列名	説明
TABLESPACE_SIZE	表領域合計サイズ
ALLOCATED_SPACE	再利用可能な領域を含む、現在割り当てられている使用量（byte）※データファイルの容量に等しい
FREE_SPACE	再利用可能な領域を含む、現在割り当てられてない使用量（byte）

表領域単位で縮小

```
ALTER TABLESPACE 一時表領域名 SHRINK SPACE [KEEP ｻｲｽﾞ]
```

一時ファイル単位で縮小

```
ALTER TABLESPACE 一時表領域名 SHRINK TEMPFILE 'ﾌｧｲﾙ名' [KEEP ｻｲｽﾞ]
```

KEEP を指定しない場合は、作成時のファイルサイズまで可能な限り縮小する。

-> グローバル一時表で使用する表領域の指定

グローバル一時表はセッション中やトランザクション中だけレコードが存在する表で、終了すると切り捨てられる表のこと。

CREATE GLOBAL TEMPORARY 文で TABLESPACE 句を指定し、グローバル一時表で使用される一時表領域の指定が可能になった。

一時表領域を指定しない場合は、ユーザーのデフォルト一時表領域が使用される。

その他の管理

-> SPFILE の管理

現在のメモリー上の初期化パラメータから PFILE、SPFILE の作成が可能になった。

CREATE { PFILE | SPFILE} [='ﾌｧｲﾙﾊﾟｽ'] FROM MEMORY;

現在のインスタンスがどの初期化パラメータファイルを読み込んで起動したかはアラートログで確認可能。

Using parameter settings in **server**-side **spfile** ﾌｧｲﾙﾊﾟｽ

Using parameter settings in **client**-side **pfile** ﾌｧｲﾙﾊﾟｽ

server-side の場合、サーバー側のデフォルト（$ORACLE_HOME/dbs）から読み込まれた SPFILE または PFILE。

client-side の場合、STARTUP PFILE コマンドが使用されている。

-> フォアグランド統計

V$SYSTEM_EVENT ビューと、V$SYSTEM_WAIT_CLASS ビューにフォアグランドプロセス（サーバープロセス）による待機時間を表現する列が追加された。

ビュー	追加された列	説明
V$SYSTEM_EVENT	TOTAL_WAIT_FG	フォアグラウンドセッションで発生した合計待機数
	TOTAL_TIMEOUTS_FG	フォアグラウンドセッションで発生した合計タイムアウト回数
	TIME_WAITED_FG	フォアグラウンドセッションで発生した待機時間（1/100 秒）
	AVERAGE_WAIT_FG	フォアグラウンドセッションで発生した平均待機時間（1/100 秒）
	TIME_WAITED_MICRO_FG	フォアグラウンドセッションで発生した待機時間（マイクロ秒）
V$SYSTEM_WAIT_CLAS	TOTAL_WAIT_FG	フォアグラウンドセッションで発生した合計待機数
	TOTAL_TIMEOUTS_FG	フォアグラウンドセッションで発生した合計タイムアウト回数

TOTAL_WAIT 列、TIME_WAITED 列は、フォアグラウンドとバックグラウンド両方の待機が含まれる為、

TOTAL_WAIT_FG 列、TIME_WAITED_FG 列の値を減算することでバックグラウンドの待機の確認が可能。