L'optimisation des applications

Historique de l'optimiseur d'Oracle.

L'optimisateur syntaxique

Avant la version 7.0, l'optimiseur syntaxique (rule-based optimizer) se contentait d'optimiser le code en utilisant un ensemble de règles internes fixes et en les appliquant à partir d'une analyse syntaxique du code :

- Règle n° 1 : Lecture des lignes isolées à l'aide du ROWID. (Cette règle posait problème lorsque les tables étaient réorganisées ou lorsque la base était portée sous Oracle8, impliquant un changement de format du ROWID).
- ……
- Règle n° 8 : Accès par l'intermédiaire d'un index composite avec toutes les clés contenues dans la clause where.
- Règle n° 9 : Accès par l'intermédiaire d'un index sur une colonne.
- Règle n° 10 : Accès par l'intermédiaire d'un index composite, avec préfixes de clés contenus dans la clause where.
- ……
- Règle n° 15 : Balayage complet de la table.

Dans le principe, pour traiter une instruction, l'optimiseur commençait à la fin de la liste, par exemple la règle n° 15, puis il remontait dans la liste. S'il trouvait un index, il décidait d'appliquer la règle n° 8 ou n° 9, etc.…
L'hypothèse de base était la suivante : à partir du moment où une instruction SQL validait une règle, et que le numéro de la règle diminuait, le plan d'exécution était réputé meilleur.
C'est à ce niveau que l'optimiseur syntaxique atteignait ses limites, incapable de déterminer la méthode la moins coûteuse, dans la mesure où il ne faisait usage d'aucun type de fonction de coût ou de statistiques. Cependant, il avait été initialement conçu pour les BDD transactionnelles, car les Datawarehouses n'existaient pas encore.

L'optimiseur statistique

Il apparaît avec la version 7.0 d'Oracle, dans le but de permettre l'utilisation d'un plus grand nombre d'options lors de la construction des plans d'exécution du code SQL, mais il lui aura fallu sept ans pour atteindre un niveau de maturité satisfaisant.
Dès Oracle 7.3, l'optimiseur avait la possibilité de générer et d'enregistrer des histogrammes de colonnes, fonctionnalité capable de déterminer la distribution effective des données pour une colonne particulière.

L'initialisation des paramètres de l'optimiseur

Le paramètre OPTIMIZER_MODE configure l'instance Oracle. Sa valeur par défaut (si elle n'est pas explicitée dans le fichier init.ora) est CHOOSE, nécessaire pour l'optimisation statistique. La valeur RULE correspond à l'optimisation syntaxique. Jusqu'à la 8i, deux autres valeurs sont également possibles : FIRST_ROWS et ALL_ROWS.

Pour modifier ce paramètre au niveau session, entrer la commande suivante :

```
Alter session set OPTMIZER_MODE=FIRST_ROWS ;
```

Les Hints

Un hint est une indication placée dans une requête pour orienter le plan d'exécution.
Il ressemble beaucoup à un commentaire, à l'exception du + placé après le /*.
Il est très important de placer le hint à l'emplacement approprié du code SQL, idéalement avant la référence à la première colonne du code SQL.

```
Select /*+ FIRST_ROWS */ last_name, hire_date, salary
From EMP
Where mgr_id = 12;
```

Si vous incluez un hint incorrect, il sera tout simplement ignoré par l'optimiseur, sans affichage de la moindre erreur.

Quel optimiseur est utilisé ?

Si le paramètre OPTIMIZER_MODE est positionné à CHOOSE, c'est la présence de statistiques dans le dictionnaire qui détermine si l'optimiseur statistique est utilisé. **En l'absence de statistiques pour l'ensemble des objets du code SQL, c'est l'optimisation syntaxique qui sera appliquée**.

Si la table est dotée d'un niveau de parallélisme, l'optimiseur statistique sera utilisé, même en l'absence de statistiques.

Il est important que les statistiques soient générées pour tous les objets dans tous les schémas. En effet, la présence de statistiques partielles pour une instruction select peut amener le processus serveur à évaluer des statistiques sur des objets qui n'en disposent pas. Ce type d'échantillonnage de stat réalisé au cours de l'exécution n'est pas enregistré de façon permanente dans le dictionnaire de données. Il est donc répété à chaque exécution de la même requête. Si votre application tierce ne prend pas en charge l'optimisation statistique, vérifiez que toutes les données statistiques de votre schéma sont effacées.
Interrogez la colonne num_row dans la vue USER_TABLES. S'il existe des valeurs pour toutes les tables, cela signifie que des stats ont été déterminées pour celles-ci. En présence de statistiques partielles, les performances du système pourront être imprévisibles et le résultat sera pénalisant pour les utilisateurs, car les stats seront calculées au moment de l'exécution.
Il est donc important de calculer les stats de tous les objets ou de ne disposer d'aucune stat pour aucun objet, de telle sorte que l'optimiseur syntaxique puisse être appliqué.

L'optimiseur syntaxique est voué à disparaître et ne sera bientôt plus supporté.

Si vous souhaitez savoir quand les stats ont été calculées, interrogez la colonne last_analyzed dans la vue DBA_TAB_COLUMNS.

Le calcul des statistiques d'objets

Le calcul des stats au niveau des objets est une opération fondamentale, car elle contrôle le comportement de l'optimiseur statistique. En l'absence de calcul effectif de stats d'un objet, le processus de détermination du coût d'un code SQL est réalisé par Oracle en s'appuyant sur des valeurs fixes.

Le calcul des stats est lancé à l'aide de la commande analyze. Cette commande calcule non seulement les stats de la table passée en paramètre, mais également celles de tous ses index.
Vous pouvez estimer (estimate) les stats en définissant une taille d'échantillon(sample) ou les calculer(compute) pour l'ensemble des lignes. La méthode d'estimation est préférable pour les BDD contenant des tables de très grande taille dans des environnements qui ne peuvent pas consacrer le temps ou les ressources nécessaires à un calcul complet.

Il est fortement recommandé de ne pas générer les stats d'objets pour l'utilisateur SYS, afin d'éviter le risque de verrouillage (deadlocking) de la base pendant le processus. D'autre part,

vous risquez de dégrader les performances du fait de la présence de stats dans les objets de l'utilisateur SYS.

A partir de la version Oracle8i, un nouveau package appelé DBMS_STATS est fourni pour réaliser et compléter les opérations de la commande analyze (préparation de la génération des stats, amélioration du calcul (traitement parallèle), transfert des stats entre le dictionnaire et vos propres tables de stats, obtention d'informations sur les stats).

Sous Oracle8i, les plans d'exécution de votre code SQL peuvent être stabilisés, pour rester statiques. Ce mécanisme s'applique tout particulièrement aux environnements qui ne doivent pas courir le risque d'un changement de leur plan d'exécution lors des changements de version, de paramètres d'initialisation, de volumes de données dans les tables, etc.
Oracle prend en charge la stabilisation du plan à l'aide des profils stockés (stored outlines).

Quelle est la quantité optimale de statistiques nécessaire ?

Si vous estimez les stats de vos objets en vous fondant sur un échantillon (estimate), il faut que la taille de celui-ci soit appropriée. C'est un élément essentiel pour fournir à l'optimiseur des stats dotées d'un bon intervalle de confiance.
Un niveau de 20% est souvent utilisé et semble approprié. Naturellement, si vous préférez calculer les stats (compute), le niveau de confiance sera de 100% et les stats parfaitement précises, si tant est que vous ayez les ressources et le temps pour réaliser ces calculs.
Si votre version est Oracle8i ou supérieure, le package DBMS_STATS pourra analyser vos tables en parallèle.

Si vous exécutez une commande analyze estimate sur une taille d'échantillon supérieure à 49%, le module calculera les stats sur l'ensemble de la table.

Les différentes méthodes de calcul des statistiques d'objets

`Analyze table EMP compute statistics` (calcul des stats pour la table EMP et tous ses index)

`Analyze table EMP compute statistics for all indexed colums` (calcul des stats pour toutes les colonnes de tous les index de la table EMP)

`Analyze table EMP estimate statistics sample size 20 percent`(calcul des stats pour la table EMP et tous ses index sur un échantillon de 20% des lignes)

La procédure `analyze_schema` du package DBMS_UTILITY provoque le calcul des stats pour l'ensemble du schéma.

```
Execute dbms_utility.analyze_schema( 'SCOTT', 'estimate',
estimate_percent=>20) ;
```

Le nouveau package de stats d'objets d'Oracle8i, DBMS_STATS permet d'exécuter les calculs avec différentes options. L'exemple suivant montre comment estimer des stats uniquement sur les tables du schéma SCOTT, avec une taille d'échantillonnage de 20% et le degré de parallélisme par défaut sur les tables. Cette commande ne calculera pas les stats des index, dès lors que l'argument cascade de la procédure `gather_schema_statistics` sera positionné (par défaut) à FALSE. En le positionnant à TRUE, vous autorisez le calcul des stats sur les index en même temps que celui réalisé sur les tables.
Pour une analyse en parallèle des index, utilisez `gather_index_stats`.

```
Execute dbms_stats.gather_schema_statistics('SCOTT', 20,
estimate_percent=>20) ;
```

La procédure exécute deux opérations. Elle réalise d'abord un export_schema_stats, puis évalue les stats pour le schéma. Compte tenu du risque que peut représenter l'implémentation de nouvelles stats, l'exécution de import_schema_stats permet une solution de repli.

Le script suivant, permet sous Sql*Plus (utilisateur SYS) de calculer les stats sur l'ensemble des tables d'un schéma pour les versions non 8i.

```
Set echo off
Set feedback off
Set verify off
Set pagesize 0
Spool analyse_tables.sql
Select 'analyze table ' || owner || '.' || table_name || ' estimate
statistics sample size 20 percent;'
From DBA_TABLES
Where owner in ('SCOTT');
Spool off

@analyze_tables.sql
```

A quelle fréquence les statistiques doivent être calculées ?

Cela dépend du taux et du volume de changement des données dans la base.
Les stats d'un objet deviennent obsolètes lorsqu'un volume important d'activité DML est opéré sur l'objet.
Certaine stats peuvent être réalisées tous les jours, chaque semaine, chaque mois …
Il faut veiller à faire suivre une insertion ou une suppression massive par une nouvelle analyse, afin d'assurer la correspondance entre les stats du dictionnaire, la distribution et le contenu des lignes de la table. Si votre table contient après-coup dix millions de lignes et que vos stats portent sur cinq millions d'entre-elles, il est possible que le plan d'exécution construit par l'optimiseur ne soit justement pas optimal …

Le package DBMS_STATS offre la possibilité d'un calcul automatique sur une table spécifique. Pour cela il faut positionner l'option `monitoring` (yes) au niveau table. Le calcul peut être réalisé en utilisant les procédures `dbms_stats.gather_schema_stats` ou `dbms_stats.gather_database_stats`.

Selon la version de votre BDD Oracle, la commande `analyze` effectue certains verrouillages pendant le calcul. Ce verrouillage a cependant été restreint par rapport aux premières versions. Sous Oracle 7.2, par exemple, la table entière était verrouillée.

Les stratégies d'indexation

Qu'est-ce qu'un index ?

Un index est un objet supplémentaire créé sur une ou plusieurs colonnes d'une table pour faciliter un accès rapide aux données. Il contient un certain volume d'informations (overhead) dont le coût de traitement peut dépasser celui d'un balayage complet de la table, selon le nombre de blocs qu'il faut lire dans la table pour renvoyer la ligne pointée par le ROWID de l'index.

La structure de données utilisée par Oracle pour stocker un index est un B*-tree, y compris pour les index de type bitmap, même si, dans ce cas, le contenu des feuilles est différent.

Le nœud principal d'un index est appelé nœud racine (root node). Le deuxième niveau est constitué par des branches (branch). Le niveau le plus faible étant constitué des feuilles (leaf) qui contiennent les données d'indexation pour chaque valeur ainsi que le ROWID correspondant. Les feuilles sont liées entre-elles par une liste doublement chaînée, permettant un parcours dans les deux directions. Les index des colonnes contenant des données de type caractère sont fondés sur les valeurs binaires des caractères, dans le jeu de caractères de la BDD.

Quand faut-il utiliser les index ?

Le seul objectif d'un index est de réduire les entrées-sorties. Une requête qui, en utilisant l'index, induirait un plus grand nombre d'entrées-sorties qu'il n'en faudrait pour réaliser un balayage complet de la table réduirait de manière significative l'intérêt de l'utilisation de cet index.
Par exemple, si une table contient un million de lignes stockées dans cinq mille blocs, et que les lignes contenant une valeur donnée d'une colonne soient réparties sur plus de quatre mille blocs, il est loin d'être optimal de créer et d'utiliser un index sur cette colonne, même si le pourcentage brut de lignes renvoyées par la table est inférieur à 1%, dès lors qu'il faut parcourir 80% du nombre total de blocs de la table pour renvoyer les données.
Autre exemple : si une table contient mille lignes et a subi un volume significatif d'opérations de type insert et delete, le niveau de flottaison (high water mark) de la table peut être élevé. Si ce niveau est de mille blocs, mais que les milles lignes soient localisées physiquement dans cent blocs, l'utilisation de l'index peut être judicieux. En effet, le nombre de blocs à lire et le nombre d'entrées-sorties à réaliser seront nettement plus faibles que pour un balayage complet de la table.

Construire des index optimaux

Quelques options d'indexation :

- Non-unique index (index avec doublons)
- Unique index (index sans doublons)
- Bitmap index
- Local prefixed partitioned index (index de partitionement local préfixé)
- Local nonprefixed partitioned index (index de partitionement local non préfixé)
- Global prefixed partitioned index (index de global local préfixé)
- Hash partitioned index
- Composite partitioned index
- Reverse-key index
- Function-based index
- Descending index
- Index-organized-table

Les questions à se poser avant de construire des index optimaux :

- Combien faut-il de blocs d'entrées-sorties pour un balayage par index en comparaison d'un balayage complet de la table ?

- Quelle est la combinaison de colonnes les plus couramment utilisées pour accéder aux données ?
  Analysez le code de l'application, voir, consultez V$SQLAREA ou V$SQLTEXT. Cherchez les instructions dotées de la plus grande valeur exécutions dans SQLAREA et déterminez la composition de leur clause where.

- Quelle est la sélectivité d'un ensemble de colonnes que vous envisagez d'utiliser pour créer un index ?
  Si certaines colonnes possèdent toujours des valeurs et qu'elles soient relativement uniques, elles peuvent constituer les colonnes fondamentales de votre index. Lors de la création, triez les colonnes dans l'ordre de la plus grande sélectivité.

- Toutes les colonnes référencées dans la clause where doivent-elles être indexées ?
  Si ces colonnes possèdent une cardinalité très faible et/ou si elles peuvent avoir des valeurs nulles, il faut les éliminer de votre liste d'indexation.

- Avez-vous besoin de conserver l'index pour les processus par lots, ou pouvez-vous le supprimer (drop) ou l'inactiver (unusable) ?

Index mono-colonne ou composite ?

Si deux colonnes sont fréquemment utilisées simultanément, il vaut mieux utiliser un index composite que deux index mono-colonnes.

Si la première colonne d'un index composite n'apparaît pas dans la clause where, l'index n'est pas utilisé …

L'index de fonction

Nouveauté de la version Oracle8i, il combine la création d'un index avec une ou plusieurs fonctions, permettant à la requête d'utiliser l'index plutôt que de réaliser un balayage complet de la table.

```
Create index IDX_CLIENTS_NOM on CLIENT (upper(nom),prenom);
```

voir avec incorporation d'une procédure PL/SQL.

```
Create index IDX_CLIENTS_CA on CLIENT (calcul_CA(id_client));
```

Quand faut-il reconstruire les index ?

Dans la mesure où un index est enregistré dans un B*-tree, et que son objectif est de permettre un accès rapide aux données de la table, il est évident que chaque consultation d'index doit impliquer le plus faible nombre de lectures de nœuds et de blocs. En effet, la réduction du nombre d'entrées-sorties est un élément clé de la pertinence d'utilisation d'un index. Le seul élément réellement significatif est le nombre de blocs de feuilles qui doivent être lus. Plus ce nombre est faible, plus le nombre d'opérations d'entrées-sorties le sera également, et plus grande sera la vitesse de lecture des lignes de la table. Cela dit, il faut noter que les index d'une table soumise à de fréquentes opérations d'insertion et de suppression présentent un risque élevé de fragmentation.
La question est donc de connaître le taux de fragmentation des blocs de feuilles de l'index. Le nombre de blocs lus et vides est-il significatif ? Il faut déterminer la densité des blocs de feuilles (leaf block density) d'un index. Plus cette densité est élevée, meilleure est la santé de l'index. Il est utile de déterminer cette densité en écrivant un script renvoyant de nombre de valeurs de lignes des colonnes d'un index et le nombre de blocs de feuilles présents dans cet index.
Par exemple, si une table contient mille valeurs et dix blocs de feuilles, la densité est de 1000/10 = 100 lignes. Si dans une semaine, le nombre de lignes passe à mille deux cents, mais qu'il y ait vingt blocs de feuilles, la densité baissera à 1200/20 = 60 lignes. Il est temps de reconstruire l'index quand il contient potentiellement un grand nombre de blocs vides.

Il est évident qu'il faut planifier les temps d'arrêt nécessaires pour la reconstruction des index, sauf depuis Oracle8i qui autorise la reconstruction des index en ligne. Les deux éléments qui vous aident lors de vos travaux de reconstruction sont le parallélisme et l'annulation de génération de redo.
Le premier est obtenu par le positionnement de la clause `parallel` dans la commande ALTER INDEX … REBUILD.
Le second peut être atteint en utilisant la clause `unrecoverable` (avant 8.0) ou `nologging` (depuis 8.0).
Les paramètres d'initialisation du parallélisme des requêtes doivent être positionnés dans le fichier init.ora avant toute tentative d'opération parallèle.

```
Alter index IDX_CLIENT_ID_CLIENT rebuild
Parallel (degree 4)
Nologging
Tablespace TBS_INDX_02 ;
```

Vous pouvez également utiliser la clause ONLINE depuis la version 8i.

Il est judicieux de compacter les nœuds des feuilles d'un index en utilisant l'option `coalesce` dans l'instruction alter index, afin de faciliter la combinaison des blocs et/ou des niveaux dans un index, de telle sorte que les blocs libres puissent être réutilisés. Il faut savoir qu'un bloc d'index vide n'est pas réutilisé tant que l'index n'est pas compacté ou reconstruit.

Pour acquérir des informations supplémentaires sur vos index, vous pouvez consulter la vue INDEX_STATS. Cette vue est remplie lors de l'exécution de la commande
`analyze index nom_index validate structure`
cette ligne n'est conservée que durant le temps de la session.

Le code SQL à ne pas écrire.

- Eviter l'utilisation des index dans les clauses where qui amèneraient les instructions SQL à consulter un plus grand nombre de blocs de données en utilisant l'index plutôt qu'en exécutant un balayage complet de la table. Ce résultat peut être obtenu en ajoutant une expression anodine à la colonne d'index, par exemple en ajoutant +0 à une colonne numérique ou en concaténant une chaîne vide à une colonne alphanumérique, ou encore, en positionnant le hint /*+ FULL si vous utilisez l'optimiseur statistique.

  ```
  Select /*+ FULL(EMP) PARALLEL(EMP, 2) */ from EMP Where……
  ```

- Ne mélangez ni ne comparez des valeurs et des types de données de colonnes car l'optimiseur ignorerait l'index.
  Si le type de colonne est NUMBER, n'utilisez pas de guillemets pour encadrer la valeur.
  De même, n'oubliez pas d'encadrer une valeur avec ces guillemets lorsqu'elle est définie comme de type alphanumérique.

  ```
  Select … from EMP where SALARY > '1000' ; -- Mauvais
  Select … from EMP where SALARY > 1000 ; -- Bon

  Select … from EMP where MANAGER = SMITH ; -- Mauvais
  Select … from EMP where MANAGER = 'SMITH' ; -- Bon
  ```

  Prêtez une attention toute particulière à votre modèle de données dès la phase de conception.

Table X		**Table Y**	
ID_COL	NUMBER	ID_COL	VARCHAR2(10)
NOM	VARCHAR2(30)	NOM	VARCHAR2(30)
ADRESSE	VARCHAR2(20)	ADRESSE	VARCHAR2(20)

  Aussi surprenant que cela puisse paraître, ce cas de figure peut se retrouver et l'on imagine le temps perdu à comprendre pourquoi la jointure (ID_COL) entre ces deux tables de fortes volumétries prend autant de temps !

- N'utilisez pas l'opérateur IS NULL dans une colonne indexée, sinon l'optimiseur ignorera l'index.

- Si vous utilisez des curseurs ou du SQL dynamique, ne codez pas vos instructions avec des valeurs en dur, ce qui empêche la réutilisation du code SQL dans le pool partagé. Utilisez des variables liées.

  ```
  Select … from EMP where EMPNO = 2324 ; -- Mauvais
  Select … from EMP where EMPNO = :1 -> Bon

  EXECUTE IMMEDIATE 'Delete from EMP where EMPNO = 2324' ; -- Mauvais
  EXECUTE IMMEDIATE 'Delete from EMP where EMPNO = :1'  USING variable
  ; -- Bon
  ```

  A partir d'Oracle8i, une amélioration peut résulter de l'insertion du paramètre CURSOR_SHARING=force dans le fichier init.ora. Ce paramètre peut également être défini au niveau session, mais peut augmenter les délais d'analyse, ce qui implique qu'il vaut mieux réécrire le code.

- N'écrivez pas d'instructions itératives insert, update ou delete sur une table à l'intérieur d'une boucle PL/SQL si les opérations peuvent être réalisées en vrac (bulk).

```
Declare
  Cursor … select from X
Begin
 For Cursor in … Loop
   Insert into Z values( Cursor.col1, Cursor.col2 *1.5, … ) ;
   Commit;
 End loop;
End ; -- Mauvais

Begin
  Insert into Z select col1, col2 * 1.5 from X where ;
  Commit;
End; -- Bon
```

- Evitez de coder des sous-requêtes liées dans vos applications. Elles ont un impact négatif sur les performances de votre système en consommant beaucoup de ressources CPU.
  La solution consiste à utiliser des vues en ligne (inline views), c'est-à-dire des sous-requêtes dans la clause from de l'instruction select, disponible depuis la version 7.3.

```
Select e.* from EMP e
Where e.salary > ( select avg( salary) from EMP i where i.dept_id =
e.dept_id ) ; -- Mauvais

Select e.* from EMP e,
(select i.dept_id DEP, avg(i.salary) SAL from EMP I group by dept_id
) EMP_VUE
where e.dept_id = EMP_VUE.dept_id
and e.salary > EMP_VUE.SAL ; -- Bon
```

- Ne construisez pas la clause from de votre instruction select avec des tables qui n'apparaissent pas dans les conditions de jointure de la clause where afin d'éviter de réaliser un produit cartésien.

- Si vous devez créer des tables et les alimenter, regroupez l'opération

```
Create table X ( col1 number, col2 varchar2(30) … ) ;
Insert into X select col1, col2 from Y … ; -- Mauvais

Create table X ( col1 number, col2 varchar2(30) … )  as select col1,
col2 from Y … ; -- Bon
```

- Eviter d'utiliser l'instruction select x from dual ; chaque fois que c'est possible. Bien qu'elle semble innocente, elle peut consommer l'essentiel des performances de votre système.

```
For i in 1..10000 loop
  Select SEQ.NEXTVAL into mavariable from dual ;
  Insert into X values( mavariable, … );
End loop; -- Mauvais

For i in 1..10000 loop
  Insert into X values(SEQ.NEXTVAL, … );
End loop; -- Bon
```

- Utilisez NOT EXISTS plutôt que NOT IN dans les clauses where, voir utiliser une jointure externe

```
-- avec not in (lent)--
Select a.nom, a .prenom
From Salarie a
Where a.nom not in (select nom from Fonction where job =
'INFORMATICEN' );

-- avec jointure externe (plus rapide)--
Select a.nom, a.prenom
From Salarie a, Fonction b
Where a.nom = b.nom(+)
And b.nom is NULL
And b.job = 'INFORMATICIEN';
```

- Utilisez l'opérateur LIKE avec un caractère initial plutôt que la fonction SUBSTR().

- Dans le cas de requêtes très complexes contenant de nombreuses conditions OR, envisagez leur réécriture à l'aide de UNION ALL. Vous arriverez ainsi à découper la requête en modules bien dimensionnés qui pourront être optimisés plus facilement.

- Utilisez les index appropriés, les plus sélectifs. La sélectivité des données est le ratio entre le nombre de clés uniques et le nombre de lignes. Plus elle approche 1.00, meilleur est l'index.

- Créez des index sur les colonnes de clés étrangères.

- Utilisez des index composites. Ceux-ci doivent être classés dans l'ordre décroissant de sélectivité.

- Utilisez des index bitmaps lorsque la clause where contient des colonnes à faible cardinalité ou des opérations logiques comme OR, AND ou NOT exécutées sur ces colonnes, ou renvoit un grand nombre de lignes de la table (sauf pour les tables supportant un grand nombre d'opérations DML simultanées, du fait de leur comportement intrinsèquement bloquant).

- Pensez à utiliser des single-table hashs ou des index clusters. Ces méthodes fournissent des performances excellentes sur les tables relativement statiques mais interrogées sur une large plage de valeurs. Sachant qu'un cluster enregistre les données dans un bloc de manière ordonnée, un balayage de plages utilisant un index sur ce cluster réalisera un plus petit nombre d'opérations d'entrées-sorties pour répondre à la requête.

- Choisissez de manière active le type de jointure : nested loop, merge join ou hash join. Lorsque vous joignez trois tables ou plus, essayez de structurer la requête pour réaliser l'élimination la plus forte sur la première jointure. Pour cela, incorporez toutes les conditions sur une table dans la clause where.

- Privilégiez le traitement en vrac (BULK COLLECT / FORALL)

- A partir d'Oracle 8i, remplacez DBMS_SQL par la nouvelle fonctionnalité execute immediate qui fonctionne nettement mieux.

```
Execute immediate 'CREATE TABLE XX AS SELECT * FROM YY' ;
```

- Si vous utilisez toujours l'optimiseur syntaxique (ce qui ne devrait plus durer), structurez vos clauses from de manière que la table la plus petite soit la dernière définie dans la liste des tables.

- Si vous avez besoin d'accélérer la création d'un index, vous pouvez augmenter la valeur du paramètre SORT_AREA_SIZE au niveau session, de sorte que l'essentiel des tris exécutés le soient en mémoire.

Tracer le code SQL

Voici les étapes du processus d'optimisation du code SQL :

1. Vérifiez que le paramètre TIMED_STATISTICS est positionné à TRUE au niveau instance
2. Vérifiez que la valeur du paramètre MAX_DUMP_FILE_SIZE est suffisante. Ce paramètre détermine la taille maximale de votre fichier de trace.
3. Déterminez l'emplacement pointé par le paramètre USER_DUMP_DEST qui indique l'emplacement ou vos fichiers de trace seront enregistrés.
4. Activez le paramètre SQL_TRACE pour la session concernée
5. Exécutez l'application.
6. Exécutez tkprof sur les fichiers de trace.
7. Etudiez le fichier de sortie de tkprof.
8. Ajustez les instructions SQL les plus coûteuses.
9. Répétez les étapes 4 à 8 jusqu'à ce que l'objectif de performance soit atteint.

Ne positionnez pas le paramètre SQL_TRACE à TRUE dans le fichier init.ora. Toutes les instructions SQL exécutées seraient enregistrées, aboutissant à un ralentissement sensible de votre système, ainsi qu'à la saturation du système de fichiers.

Vous pouvez activer la trace pour votre session courante en exécutant la commande suivante :

```
alter session set timed_statistics = true ;
alter session set sql_trace = true ;
…
-- execution de vos traitements …
…
alter session set timed_statistics = false ;
alter session set sql_trace = false ;
```

Avec les versions antérieures à la 7.2, il fallait attendre l'exécution complète des instructions SQL pour pouvoir désactiver SQL_TRACE à l'intérieur de la session. A partir de 7.2, il est possible de désactiver la trace à partir d'une autre session.
Il ne faut pas désactiver la trace en annulant la session (kill), car cela peut provoquer la troncature du fichier de trace et/ou introduire des informations erronées dans ce fichier.

Voici la méthode pour désactiver SQL_TRACE sur la session d'un autre utilisateur :

```
Select sid, serial# from V$SESSION where username = 'SCOTT' ;

      SID        SERIAL#
---------   ------------
       11             54

Execute DBMS_SYSTEM.set_sql_trace_in_session( '11', '54', FALSE ) ;
```

Cette méthode peut également être utilisée pour lancer la trace :

```
Execute DBMS_SYSTEM.set_sql_trace_in_session( '11', '54', TRUE ) ;
```

Le fichier de trace généré se trouve dans le répertoire désigné par le paramétre d'initialisation USER_DUMP_DEST. Si vous n'êtes pas sûr de cette destination, vous pouvez exécuter la requête suivante :

```
Select value from V$PARAMETER where name = 'user_dump_dest' ;
```

Les fichiers trace sont dénommés en utilisant le format <$ORACLE_SID>_ora_<ID du processus serveur>. L'identificateur du processus serveur correspond à la colonne spid de V$SESSION.

```
Select s.username, p.spid, s.program
From V$SESSION s, V$PROCESS p
Where s.paddr = p.addr
And s.username = 'SCOTT' ;
```

Exécuter tkprof sur les fichiers de trace

L'étape suivante du processus d'optimisation du code SQL consiste à exécuter l'utilitaire tkprof pour analyser les fichiers de trace. Pour obtenir la syntaxe de la commande et afficher les options de tri du fichier de sortie, exécutez la commande sans argument :

```
frbs./opt/oracle-> tkprof
Usage: tkprof tracefile outputfile [explain= ] [table= ]
              [print= ] [insert= ] [sys= ] [sort= ]
  table=schema.tablename   Use 'schema.tablename' with 'explain=' option.
  explain=user/password    Connect to ORACLE and issue EXPLAIN PLAIN.
  print=integer    List only the first 'integer' SQL statements.
  aggregate=yes|no
  insert=filename  List SQL statements and data inside INSERT statements.
  sys=no           TKPROF does not list SQL statements run as user SYS.
  record=filename  Record non-recursive statements found in the trace file.
  sort=option      Set of zero or more of the following sort options:
    prscnt  number of times parse was called
    prscpu  cpu time parsing
    prsela  elapsed time parsing
    prsdsk  number of disk reads during parse
    prsqry  number of buffers for consistent read during parse
    prscu   number of buffers for current read during parse
    prsmis  number of misses in library cache during parse
    execnt  number of execute was called
    execpu  cpu time spent executing
    exeela  elapsed time executing
    exedsk  number of disk reads during execute
    exeqry  number of buffers for consistent read during execute
    execu   number of buffers for current read during execute
    exerow  number of rows processed during execute
    exemis  number of library cache misses during execute
    fchcnt  number of times fetch was called
    fchcpu  cpu time spent fetching
    fchela  elapsed time fetching
    fchdsk  number of disk reads during fetch
    fchqry  number of buffers for consistent read during fetch
    fchcu   number of buffers for current read during fetch
    fchrow  number of rows fetched
    userid  userid of user that parsed the cursor
```

tracefile correspond au fichier trace généré par SQL_TRACE.
Outputfile correspond au fichier en sortie que vous souhaitez.
Il suffit de spécifier sort=<option> dans la syntaxe de ligne de commande. La valeur par défaut est : fchela (elapsed time fetching). La sortie de trace est triée en ordre décroissant des valeurs d'une option.

Si vous souhaitez afficher le plan d'exécution dans la sortie trace, il faut vous assurer que l'utilisateur (userid) spécifié lors de l'exécution de explain plan dans tkprof possède la PLAN_TABLE dans son schéma.

Pour créer cette table dans le schéma de l'utilisateur, il faut exécuter le script utlxplan.sql qui se trouve dans le répertoire : $ORACLE_HOME/rdbms/admin.

Les options de tri les plus couramment utilisées sont prsela, exeela et fchela.
Ce n'est généralement pas dans les phases d'analyse (parse) et d'exécution (execute) que se trouve le problème d'une instruction select mais pendant la phase de lecture (fetch), ce qui explique que fchela est l'option la plus significative pour une instruction select.
Pour les instructions qui modifient les données (insert, update, delete), exeela est une option plus appropriée.
Si vous souhaitez que vos instructions soient triées selon leur utilisation totale de CPU, utilisez l'option sort=(prsela, exeela, fchela).
Pour un affichage orienté entrées-sorties physiques, utilisez sort=(prsdsk, exedsk, fchdsk).
Pour le nombre total d'entrées-sorties logiques, utilisez sort=(prsqry, prscu, exeqry, execu, fchqry, fchcu).

Si le programme tkprof doit être exécuté par des profils autres que DBA, il faudra positionner le paramètre d'initialisation TRACE_FILES_PUBLIC=TRUE pour leur donner un accès en lecture aux fichiers de trace.

Interpréter les différentes colonnes de la sortie de tkprof.

- **Call**. La phase du traitement de l'instruction SQL (les phases define et bind sont incluses dans la phase parse).
- **Count**. Le nombre d'appels et d'exécutions d'une phase
- **CPU**. Le temps CPU (en secondes) consommé pour l'exécution d'une phase
- **Elapsed**. Le temps CPU (en secondes) écoulé, cumulé avec l'ensemble des temps utilisés par le S.E. pour réaliser les changements de contexte, traiter les interruptions, répondre aux signaux, exécuter les entrées-sorties, attendre les ressources, etc.
- **Disk**. Le nombre de blocs Oracle lus sur le disque pour une phase donnée
- **Query**. Le nombre de blocs Oracle lus en mémoire en mode cohérent (consistent mode).
- **Current**. Le nombre de blocs Oracle lus en mémoire en mode courant.
- **Rows**. Le nombre de lignes traitées dans chaque phase, avec les valeurs pour les instructions select dans la phase fetch, et pour les opération insert, update dans la phase execute.

La fonction AUTOTRACE

Cette fonctionnalité de Sql*Plus est disponible depuis la version 2.3
Elle fournit des informations sans avoir besoin d'exécuter SQL_TRACE.

AUTOTRACE peut être installée en exécutant le script plustrce.sql du répertoire $ORACLE_HOME/sqlplus/admin.
Pour installer AUTOTRACE, il suffit de se connecter comme SYS, et d'exécuter le script.
Il crée entre autres objets, un rôle appelé PLUSTRACE, qu'il ne reste plus qu'à affecter aux utilisateurs.

La fonction AUTOTRACE utilise la PLAN_TABLE qui doit donc avoir été créée dans le schéma du(des) utilisateur(s).

Bien que la méthode par défaut pour utiliser AUTOTRACE consiste à exécuter la commande set autotrace on, elle n'est pas toujours applicable, en particulier si votre requête revoie beaucoup de lignes. L'option traceonly permet de n'afficher que les statistiques.

```
SQL> set autotrace traceonly
```

```
SQL> select ch1, sum(nb2) from tmp_test group by ch1;

10 rows selected.

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=RULE
   1    0   SORT (GROUP BY)
   2    1     TABLE ACCESS (FULL) OF 'TMP_TEST'


Statistics
----------------------------------------------------------
          0  recursive calls
          4  db block gets
          1  consistent gets
          0  physical reads
          0  redo size
        865  bytes sent via SQL*Net to client
        429  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
         10  rows processed

SQL> set autotrace off
```