Résumé SQL Tuning

Rappel : Architectures Oracle

Principes de blocs et ROWID

Les données sur disque sont stockées en fichiers constitués de pages, qui servent d'unité d'échange entre le disque et la mémoire. Le coût des opérations en BDD est principalement mesuré en nombre d'E/S de pages. Chaque fichier contient des enregistrements de taille fixe ou variable, identifiés par un ROWID, composé de l'adresse de la page et de l'indice de l'enregistrement dans celle-ci.

Architecture Oracle Non-CDB

Le serveur oracle contient 2 composants principaux : l'instance et la base de données

L'instance comprend la mémoire **SGA** et les **processus en arrière-plan**.

La base de données comprend un ensemble de fichiers : fichier de contrôle , de données et de journalisation

Structures mémoire et processus Oracle :

La mémoire **SGA** contient plusieurs zones : mémoire partagée, cache, journalisation…

Les principaux **processus Oracle** incluent **PMON, SMON, DBWR, LGWR, ARCn**, assurant la gestion et l'optimisation des bases de données.

Architecture Multitenant

Permet une base de données conteneur (CDB) pouvant héberger plusieurs bases de données enfichables (PDB).Un conteneur (CDB) est une collection logique de métadonnées et de données. Une PDB est une collection logique portable de schémas et objets. Avant Oracle 12c, les bases de données étaient non-CDB.

Contenu d'une CDB

Contient un conteneur racine (CDB$ROOT) avec les métadonnées Oracle. Une base d'amorçage (PDB$SEED) sert de modèle pour créer de nouvelles PDB. Une ou plusieurs PDB contenant les données utilisateur (tables, index, etc.).

Avantages de l'architecture multitenant

- **Isolation des PDB** tout en partageant une même instance.
- **Provisionnement rapide** via clonage et portabilité des PDB.
- **Gestion centralisée** pour les mises à jour et sauvegardes.
- **Optimisation des ressources** grâce à une mémoire partagée et moins de processus d'arrière-plan.

Architecture en grille (Oracle RAC (Real Application Clusters))

Une base de données à instance unique a une relation un-à-un entre l'instance et la base de données. Oracle RAC permet une relation un-à-plusieurs, avec jusqu'à 100 instances accédant à une seule BD.Toutes les instances partagent les mêmes fichiers de BD via un stockage partagé géré par ASM (Automatic Storage Management).

Résumé SQL Tuning

Chapitre 1 : Optimisation algébrique

Optimisation algébrique

L'objectif de l'optimisation algébrique des requêtes et de créer un plan d'exécution logique optimal en termes de temps d'exécution et de mémoire utilisée et ceci en se basant sur les propriétés des opérateurs algébriques.

Un plan d'exécution logique est arbre algébrique qui décrit les étapes d'exécution logique d'une requête sous forme d'opérations algébriques, telles que la projection, la restriction, l'union, la jointure…

Un plan d'exécution optimale est obtenu suite à une réécriture algébrique basée sur un ensemble de règles appliquées sur les opérateurs algébriques se trouvant dans le plan initial.

Passage du SQL vers le plan d'exécution physique

Le passage d'une requête SQL a un plan d'exécution physique passe par deux étapes:

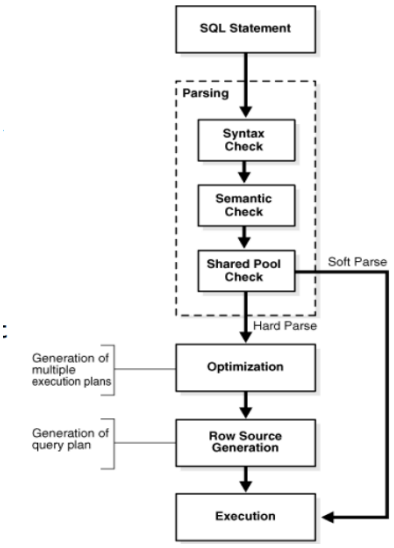
A) La requête SQL est traduite à l'aide des opérateurs d'algèbre en plusieurs plans d'exécution logiques (assez abstraits pour l'exécution).

B) Le système choisit des opérateurs d'exéc en fonction du contexte d'exécution (l'existence ou non d'index, tailles des tables, mémoire disponible …) afin d'obtenir des plans d'exéc physiques équivalents. À la fin, le système choisit le plan d'exécution optimal ayant le meilleur coût.

Chapitre 2 : Processus d'exécution des requêtes et Optimisation

Le processus d'exécution d'une requête SQL

L'exécution d'une instruction SQL dans Oracle se déroule en **3 étapes** : **parsing, execution et fetch**.



- Deux types de parsing existent :
    1. **Soft parsing** (« Library Cache Hit ») : réutilisation d'un plan d'exécution existant, plus performant.
    2. **Hard parsing** (« Library Cache Miss ») : nécessite une optimisation complète, plus coûteux.
- Les **3 premières sous-étapes du parsing** sont communes à toutes les requêtes.
- Les **2 dernières sous-étapes** sont gérées par l'**optimiseur Oracle** et ne concernent que le **Hard parsing**.

1. Parsing (Analyse syntaxique et sémantique)

- **Sous-étape 1 : Vérification de la syntaxe** → Vérifie la structure et les mots-clés de la requête.
- **Sous-étape 2 : Vérification sémantique** → Valide l'existence des objets (tables, colonnes) et les privilèges utilisateur via le Dictionary Cache.
- **Sous-étape 3 : Vérification du pool partagé** → Si la requête est déjà en cache (**Soft Parsing**), elle est exécutée directement. Sinon, elle passe par l'optimiseur (**Hard Parsing**).

2. Exécution (Optimisation et traitement des données)

- **Sous-étape 4 : Optimisation** → L'optimiseur génère plusieurs plans d'exécution (jointures, scans d'index…) et choisit le plus performant.
- **Sous-étape 5 : Génération du plan d'exécution** → Création du plan définitif, visible via **Explain Plan**.
- **Sous-étape 6 : Exécution** → Lecture des blocs de données en cache ou sur disque. Pour les mises à jour (**LMD**), Oracle gère aussi le **redo log** et les **blocs d'annulation**.

3. Fetch (Récupération des résultats)

Après exécution, le serveur retourne les données au client pour affichage.

Chapitre 3 : Index, méthodes d'accès et de jointures

Index

Un index est une structure de données qui représente un objet dans un schéma de base de données. Il contient une entrée pour chaque valeur présente dans la colonne indexée, permettant un accès rapide aux enregistrements.

L'indexation permet de retrouver rapidement un enregistrement particulier en utilisant un **ROWID**, qui est une adresse d'enregistrement composée de plusieurs éléments : object_id, data_file_id, data_bloc_id, et position_in_block.

L'index dans une base de données améliore les performances des opérations telles que la recherche, le tri, la jointure et l'agrégation.

Dans Oracle, un index est automatiquement créé pour une clé primaire (**PRIMARY KEY**) ou une contrainte d'unicité (**UNIQUE**) sur une colonne.

Les index doivent être utilisés dans les situations suivantes :

- Pour les colonnes fréquemment soumises à des recherches.
- Pour les colonnes souvent utilisées dans des jointures (comme les clés étrangères).
- Pour les colonnes très discriminantes, c'est-à-dire celles dont les valeurs sont peu répétées dans la base.
- Pour les colonnes rarement modifiées.

Cependant, l'utilisation d'index présente aussi des inconvénients :

- Les index peuvent ralentir les performances lors des mises à jour, car il est nécessaire de mettre à jour les index en même temps que les données.
- Les index augmentent la taille de la base de données, et leur volume peut devenir significatif.

Syntaxe pour créer un index :

```
CREATE INDEX index_name ON table_name(column1 [asc], column2 [desc], ...);
```

Pour afficher les index d'une table :

SELECT index_name FROM USER_INDEXES WHERE table_name='EMPLOYEES';

SELECT column_name FROM USER_IND_COLUMNS WHERE table_name='EMPLOYEES';

Types d'index Oracle

Arbre équilibré (B-Tree) : l'index par défaut dans Oracle. Il se compose des éléments suivants :

- Racine : Le point d'entrée principal de l'arbre.
- Branches : Contiennent les clés triées en ordre ascendant.
- Feuilles : Contiennent à la fois les clés et les ROWID (adresses vers les blocs d'enregistrements).

L'arbre est parcouru de gauche à droite, en commençant par la racine, en passant par les branches, puis en atteignant les feuilles.

Le niveau le plus bas, appelé niveau des feuilles, contient des entrées d'index sous forme de paires clé/ROWID, permettant d'accéder directement à un bloc d'enregistrement.

Un index unique est créé automatiquement lorsqu'une contrainte UNIQUE est ajoutée à une colonne. Il garantit l'unicité des valeurs dans la colonne et est de type B-tree. Il peut aussi être créé manuellement.

Syntaxe : CREATE UNIQUE INDEX index_name ON table_name(column1);

Un index inversé est aussi un B-tree, mais il parcourt l'arbre de droite à gauche. Il est utilisé pour rechercher les valeurs les plus élevées efficacement.

Syntaxe : CREATE INDEX index_name ON table_name(column1) REVERSE;

Index composé (composite Index) : est un index qui s'applique sur plusieurs colonnes. Il est ordonné d'abord par la première colonne, puis par la deuxième, et ainsi de suite. L'ordre des colonnes est important, il est préférable de commencer par celle avec la plus haute cardinalité (plus de valeurs distinctes).

Syntaxe : CREATE INDEX Emp_Nom_Ville_IDX ON Employees(Nom, ville);

Un Bitmap Index est utilisé lorsque le nombre de valeurs distinctes dans une colonne est faible par rapport au nombre total d'enregistrements (par exemple, pour une colonne gender). Il peut être simple ou composé et convertit les ROWID en bitmaps. Des opérations AND et OR sont effectuées sur ces bitmaps avant de les reconvertir en ROWID. Il est efficace pour des tables en lecture seule ou peu mises à jour, et consomme moins d'espace que les index B-tree.

Syntaxe : CREATE BITMAP INDEX index_name ON table_name(column1[, column2, ...]);

Un index basé sur une fonction permet de créer un index sur une fonction ou une expression appliquée à une colonne. Si une fonction est utilisée sur une colonne indexée, l'index classique ne sera pas utilisé. Par exemple, pour une requête utilisant LOWER(email), un index classique sur email ne suffira pas.donc il **faut créer un index sur la fonction, comme LOWER(email) :**

CREATE INDEX Emp_Fun_IDX ON Employees(LOWER(email));

Il est également possible de créer un index sur une expression, par exemple :

CREATE INDEX emp_total_sal_idx ON employees (12 * salary * commission_pct);

Les tables organisées par index (IOT) stockent les clés primaires et les données de colonnes non-clés dans la même structure B-Tree, éliminant ainsi la nécessité de conserver les ROWID. Cela permet un accès plus rapide aux données via la clé primaire, car les données et l'index sont dans la même structure, évitant ainsi une lecture séparée de l'index et des données.

L'IOT est particulièrement adapté aux petites tables où les recherches sont principalement effectuées sur la clé primaire et où il y a peu de mises à jour.

Syntaxe :

CREATE TABLE table_name (id_name INT PRIMARY KEY, column1, ...) ORGANIZATION INDEX INCLUDING column1, column2, ...;

Reconstruire un Index (Rebuild)

La reconstruction d'un index est nécessaire lorsque l'index devient fragmenté à cause des insertions, modifications et suppressions de données. La fragmentation peut ralentir les performances des requêtes. La reconstruction supprime l'index existant et en crée un nouveau, ce qui élimine la fragmentation et réorganise les données.

Pendant la reconstruction, des verrous sont placés sur l'index, empêchant l'accès à celui-ci. La reconstruction doit être effectuée régulièrement pour maintenir des bonnes performances, de préférence la nuit ou pendant les week-ends.

Syntaxe : ALTER INDEX index_name REBUILD [ONLINE];

L'option ONLINE permet de maintenir l'accès à la table pour les opérations DML (insertion, mise à jour, suppression) et partitionnement pendant la reconstruction.

Méthodes d'accès d'indexes

Full Index Scan : Utilisée lorsque les colonnes peuvent être directement sélectionnées de l'index (par exemple, pour un index composé ou une Table Organisée par Index - IOT).

Fast Full Index Scan : Choisie lorsque les colonnes sélectionnées font partie de l'index et qu'au moins une de ces colonnes à la contrainte NOT NULL. Elle permet d'accéder directement aux données de l'index, sans accéder à la table.

Index Range Scan : Utilisée lorsque la condition sur la colonne indexée porte sur un ensemble de valeurs, comme les opérateurs >, >=, <, <=, BETWEEN, ou l'opérateur = dans le cas d'un index non-unique.

Index Unique Scan : Utilisée lorsque l'index est unique (comme une clé primaire).

Index Skip Scan : Utilisée lorsqu'une condition est appliquée à une colonne qui fait partie d'un index mais qui n'est pas la première colonne de cet index. La première colonne de l'index est ignorée.

Méthodes d'accès de tables

FULL SCAN : utilisée lorsque Il n'y a pas d'index sur les colonnes sélectionnées ou La clause WHERE ne contient pas de conditions, ou les conditions sont faites sur des colonnes non indexées aussi Oracle peut aussi choisir un FULL SCAN si cela est jugé plus performant, par exemple lorsque la requête retourne une grande partie ou la totalité des données, ou si les données sont physiquement mal ordonnées ou contiennent beaucoup d'espace libre.

BY INDEX ROWID : Cette méthode est choisie lorsqu'un index est utilisé. Pour chaque ligne de l'index dont les colonnes correspondent aux conditions de sélection, un accès à la table est effectué en utilisant le ROWID.

Méthodes de Jointure

Nested Loops : Utilisée lorsque les tables à joindre sont petites. La recherche dans la table interne doit être supporté par index. Moins performant que les jointures hash et Sort Merge. Fonctionne mieux si la table externe est plus petite et que la table interne a un index hautement sélectif sur la clé de jointure de la table externe.

Hash Join : Utilisée lorsque les tables à joindre sont grandes et il y a un prédicat de jointure d'égalité. Elle fonctionne en créant une table de hachage pour une des tables et en cherchant ensuite les correspondances dans l'autre table

Sort Merge Join : adapté lorsque les tables à joindre sont grandes et qu'un prédicat de jointure de non-égalité est appliqué (par exemple, avec des opérateurs comme > ou <). Cette méthode devient encore plus efficace si les tables sont déjà triées sur les colonnes utilisées pour la jointure. Elle est utile lorsque la majorité des données des deux tables doit être incluse dans le résultat, ou lorsqu'il n'y a pas d'index disponibles dans la table interne .

Chapitre 4 : Optimiseur Oracle et plans d'exécution

L'optimiseur de requêtes est un composant de la base de données qui détermine la méthode la plus efficace pour exécuter une requête SQL. Il utilise des statistiques sur les données pour évaluer différentes méthodes d'accès et de jointure, ainsi que les ordres et transformations possibles. L'optimiseur calcule un coût pour chaque plan d'exécution et choisit celui qui présente le coût le plus bas. Le modèle le plus courant est l'optimiseur basé sur les coûts (CBO), qui est plus utilisé que l'optimiseur basé sur les règles (RBO).

Composants de l'optimiseur

Le Query Transformer évalue si la requête doit être modifiée pour générer un meilleur plan d'exécution,

L'Estimator estime le coût de chaque plan en se basant sur les statistiques du dictionnaire de données.

Le Plan Generator génère ensuite le plan optimal.

L'Estimateur utilise trois mesures principales pour évaluer le coût total :

1. **Sélectivité** : représente le % de lignes sélectionnées par la requête, influencée par les prédicats. Plus la sélectivité est proche de 0, plus le prédicat est sélectif, et vice versa.
2. **Cardinalité** : nombre estimé de lignes renvoyées par chaque opération dans le plan d'exécution, calculée comme le nombre total de lignes divisé par le nombre de valeurs distinctes (NDV).
3. **Coût** : mesure numérique qui reflète l'utilisation des ressources pour un plan, prenant en compte les E/S disque, l'utilisation du processeur et la taille de la mémoire.

Affichage du plan d'exécution

Graphique : Utilisation de SQL Developer (en appuyant sur F10) pour visualiser le plan d'exécution sous forme graphique.

Textuel : Utilisation du package PL/SQL DBMS_XPLAN :

Dbms_xplan.display : Affiche le plan d'exécution théorique sans exécuter la requête.

Dbms_xplan.display_cursor : Affiche le plan de la dernière requête exécutée. l'utilisateur (autre que SYS) doit disposer des privilèges de sélection appropriés sur plusieurs vues système.

Dbms_xplan.display_awr(sql_id) : Affiche le plan d'exécution d'un Top SQL stocké dans les vues historiques d'AWR, où sql_id peut être obtenu depuis un rapport AWR.

Générer un plan explicatif (textuel)

Génération :

EXPLAIN PLAN [SET STATEMENT_ID='id de requête'] [INTO Nom2Table] FOR instruction SQL;

L'instruction SQL peut être un SELECT, INSERT, UPDATE, DELETE, CREATE TABLE, CREATE INDEX, ALTER INDEX, etc.

Affichage :

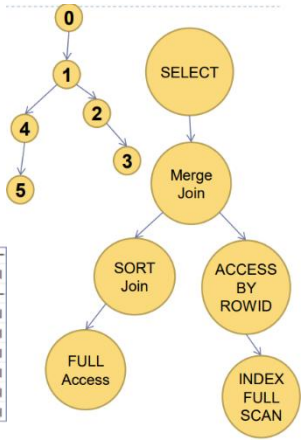
On utilise la fonction **dbms_xplan.display()** pour afficher le plan d'exécution.

Exemple : **SELECT * FROM table(dbms_xplan.display());**

Construire l'arbre d'exécution à partir du plan

L'arbre peut être construit à partir de ces règles:

- Nœud père a plusieurs fils.
- Nœud fils a un seul père.
- La seule opération sans père est la racine de l'arbre.
- Fils sont en retrait à droite de leurs père.
- Le id du père est inférieur à celui du fils.




```
-----------------------------------------------------------
| Id  | Operation                    | Name               |
-----------------------------------------------------------
|   0 | SELECT STATEMENT             |                    |
|   1 |  MERGE JOIN                  |                    |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPARTMENTS        |
|   3 |    INDEX FULL SCAN           | DEPT_ID_PK         |
|*  4 |   SORT JOIN                  |                    |
|   5 |    TABLE ACCESS FULL         | EMPLOYEES          |
```

Format d'affichage du plan

Le format d'affichage du plan d'exécution peut être spécifié avec l'option format dans la fonction **dbms_xplan.display(format => 'format_option')**. Voici les différentes options disponibles :

1. **BASIC** : Affiche les informations minimales du plan, telles que l'ID d'opération, le nom de l'opération et l'objet de l'opération.
2. **TYPICAL** : Valeur par défaut. Affiche les informations essentielles du plan, comme l'ID d'opération, le nom, l'objet, le nombre de lignes, le nombre d'octets et le coût de l'optimiseur. Les informations sur le parallélisme et les prédicats sont affichés si elles sont applicables.
3. **SERIAL** : Comme **TYPICAL**, mais sans afficher les informations sur le parallélisme, même si l'opération s'exécute en parallèle.
4. **ALL** : Affiche un niveau détaillé avec des informations supplémentaires telles que **PROJECTION**, **ALIAS** et des informations sur le **REMOTE SQL** si l'opération est distribuée.

Dbms_xplan.display_cursor

La fonction **dbms_xplan.display_cursor** est utilisée pour afficher le plan d'exécution d'une requête déjà exécutée dans le curseur. Voici comment l'utiliser :

1. **Collecter les statistiques de la dernière requête exécutée dans le curseur** :
    - Exécutez la requête normalement.
    - Utilisez la commande : **Select * from table(dbms_xplan.display_cursor());**
2. **Comparer les statistiques réelles et celles estimées** :
    - Ajoutez-le **hint /*+ GATHER_PLAN_STATISTICS*/** à la requête avant de l'exécuter. Cela permet de collecter les statistiques réelles, comme le nombre de lignes.
    - Ensuite, exécutez la commande avec le format : **Select * from table(dbms_xplan.display_cursor(format => 'ALLSTATS LAST'));**

La comparaison permet de savoir est ce que l'optimiseur Oracle a effectué une bonne estimation des cardinalités des opérations du plan ou non. C'est une technique rapide de commencer le tuning de la requête.

- Techniques d'influence sur l'optimisateur
- Collecte des statistiques,
- Profils SQL,
- Plans de gestion SQL,
- Hints (conseils),
- Paramètres d'initialisation

Chapitre 5 : Oracle Statistiques

Statistiques

Oracle collecte les statistiques automatiquement pour tous les objets de base de données dont les statistiques sont manquantes ou dont les statistiques sont obsolètes grâce à un job qui s'exécute (de 22h à 02h tous les jours de la semaine et de 06h à 02 les weekends).

La collecte des statistiques permettent de mettre à jour les vues statiques du dictionnaire USER ou DBA tels que: USER_TAB_STATISTICS, USER_IND_STATISTICS, etc.

Collecte des statistiques manuellement

En cas de statistiques obsolètes ou manquantes, un DBA peut collecter manuellement des statistiques via le package **PL/SQL DBMS_STATS** pour améliorer l'exécution des requêtes. Ce package permet de collecter, modifier, afficher, exporter, importer et supprimer des statistiques sur divers objets tels que les **index**, **tables**, **colonnes**, **schémas**, **bases de données**, et même le **système**.

Les principales procédures incluent :

- **GATHER_INDEX_STATS** : pour les index.
- **GATHER_TABLE_STATS** : pour les tables et leurs éléments associés.
- **GATHER_SCHEMA_STATS** : pour tous les objets d'un schéma.

- **GATHER_DATABASE_STATS** : pour l'ensemble de la base de données.
- **GATHER_SYSTEM_STATS** : pour les statistiques du processeur et des entrées/sorties.

Collecte des statistiques

Collecte des statistiques pour une table :

```
EXEC DBMS_STATS.GATHER_TABLE_STATS(
  ownname => 'hr',
  tabname => 'employees',
  estimate_percent => DBMS_STATS.AUTO_SAMPLE_SIZE,
  cascade => TRUE
);
```

La clause estimate_percent => DBMS_STATS.AUTO_SAMPLE_SIZE permet de déterminer automatiquement le pourcentage de lignes à utiliser pour la collecte des statistiques, tandis que cascade => TRUE permet de collecter également des statistiques pour les index associés à la table. Pour afficher ces statistiques, la requête suivante peut être utilisée :

```
SELECT * FROM USER_TAB_STATISTICS WHERE table_name = 'EMPLOYEES';
```

Collecte des statistiques pour un index :

```
EXEC DBMS_STATS.GATHER_INDEX_STATS(
  ownname => 'hr',
  indname => 'EMP_NAME_IX',
  estimate_percent => DBMS_STATS.AUTO_SAMPLE_SIZE
);
```

Pour afficher ces statistiques, on peut exécuter la requête suivante :

```
SELECT * FROM USER_IND_STATISTICS WHERE index_name = 'EMP_NAME_IX';
```

Collecte des statistiques pour un schéma entier :

```
EXEC DBMS_STATS.GATHER_SCHEMA_STATS(
  ownname => 'hr',
  estimate_percent => DBMS_STATS.AUTO_SAMPLE_SIZE,
  cascade => TRUE
);
```

Les statistiques peuvent être affichées via les vues statiques mentionnées précédemment (USER_TAB_STATISTICS, USER_IND_STATISTICS, etc.).

Suppression des statistiques

Résumé SQL Tuning

Pour supprimer les statistiques d'une table, un index ou un schéma, les procédures PL/SQL peuvent respectivement être utilisées:

- DBMS_STATS.DELETE_TABLE_STATS
- DBMS_STATS.DELETE_INDEX_STATS
- DBMS_STATS.DELETE_SCHEMA_STATS

Interpréter les statistiques générées sur les tables

1. **Espace inutilisé alloué à une table** :
    - Si le nombre de blocs inutilisés (EMPTY_BLOCKS) est élevé, cela signifie que l'espace alloué à la table n'est pas efficacement utilisé. Ce problème est souvent causé par une mauvaise gestion de la clause **STORAGE** (paramètres comme INITIAL, NEXT, etc.).
2. **Faible taux d'occupation des blocs** :
    - Un faible taux d'occupation des blocs peut être dû à des valeurs inadaptées de **PCTFREE** et **PCTUSED** ou à une suppression massive de données. Cela entraîne une sous-utilisation des ressources allouées et une fragmentation du stockage.
3. **Grand nombre de blocs de feuilles dans l'index** :
    - Si le nombre de blocs de feuilles dans l'index augmente, cela peut entraîner une surcharge des **E/S**. La cause principale est la **fragmentation de l'index** due à des insertions et suppressions fréquentes.
4. **Profondeur excessive de l'index** :
    - Une profondeur d'index (BLEVEL) trop élevée peut ralentir les performances de recherche, souvent à cause d'un **PCTFREE mal adapté** ou d'un **index très volatil**.

Solutions :

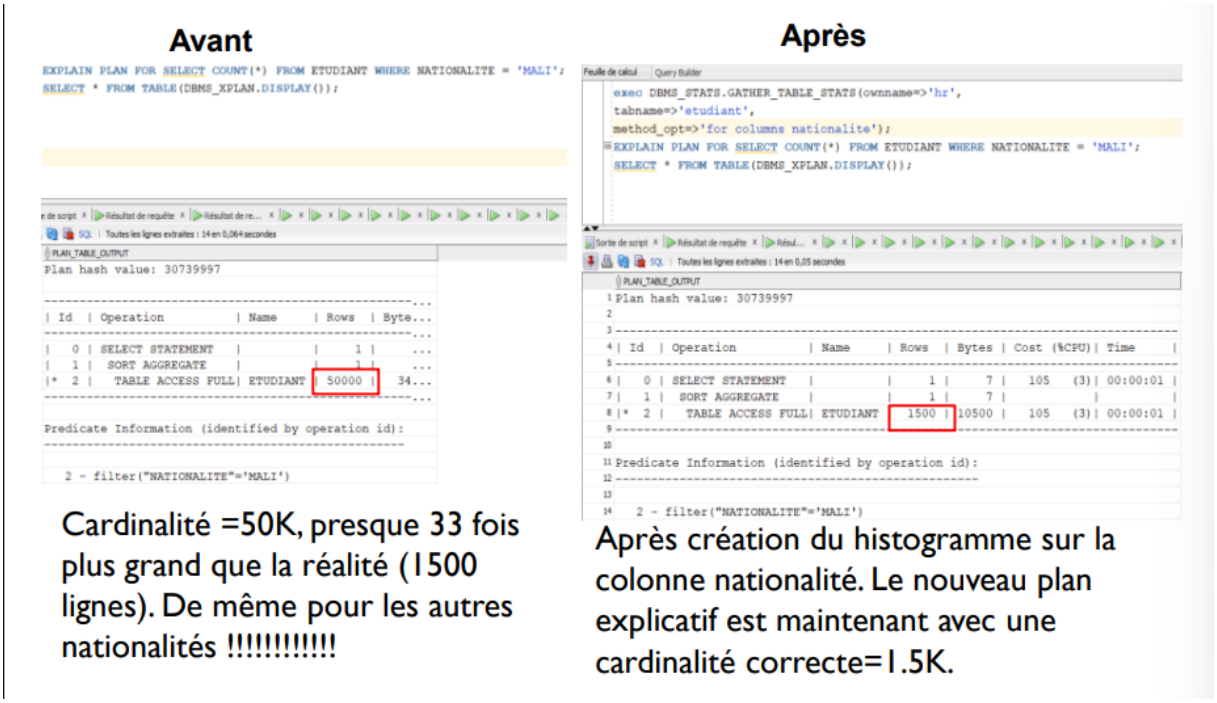
- **Reconstruction d'index** : Pour résoudre la fragmentation ou la profondeur excessive de l'index, il est nécessaire de **reconstruire l'index** pour améliorer les performances :

ALTER INDEX index_name REBUILD;

Statistiques et Histogrammes

Par défaut, l'optimiseur suppose une distribution uniforme des lignes sur les différentes valeurs d'une colonne. Pour les colonnes contenant des données asymétriques (une distribution non uniforme des données au sein de la colonne), un histogramme permet à l'optimiseur de générer des estimations de cardinalité précises pour les prédicats de filtre et de jointure qui impliquent ces colonnes. Donc, **un histogramme est un type de statistiques** qui permet à l'optimiseur d'analyser la distribution des données et estimer correctement la cardinalité.

Impact d'histogramme sur la cardinalité



Gestion des histogrammes

1. **Création d'un histogramme** :

```
EXEC dbms_stats.gather_table_stats(
   ownname => 'owner_name',
   tabname => 'table_name',
   method_opt => 'for columns skewed_column_name'
);
```

Exemple :

```
EXEC dbms_stats.gather_table_stats(
   ownname => 'hr',
   tabname => 'etudiant',
   method_opt => 'for all columns nationalite'
);
```

2. **Suppression d'un histogramme** :

```
BEGIN
   dbms_stats.delete_column_stats(
      ownname => 'hr',
      tabname => 'etudiant',
      colname => 'nationalite',
      col_stat_type => 'HISTOGRAM'
   );
END;
```

Statistiques dynamiques

Ou **échantillonnage dynamique**, sont utilisées par l'optimiseur lors de la compilation d'une requête SQL lorsque les statistiques d'une table sont manquantes ou insuffisantes.

- **Utilisation** : Si certaines tables de la requête n'ont pas de statistiques, l'optimiseur utilise les statistiques dynamiques pour collecter des données de base avant d'optimiser la requête.
- **Limites** : Ces statistiques sont moins complètes et de moins bonne qualité que celles collectées avec le package DBMS_STATS. Ce compromis est fait pour minimiser l'impact sur le temps de compilation de la requête.

Niveaux des statistiques dynamiques

Les **statistiques dynamiques** sont contrôlées par le paramètre **OPTIMIZER_DYNAMIC_SAMPLING**, qui peut être défini sur des niveaux allant de **0 à 11**. Ces niveaux influencent deux aspects importants :

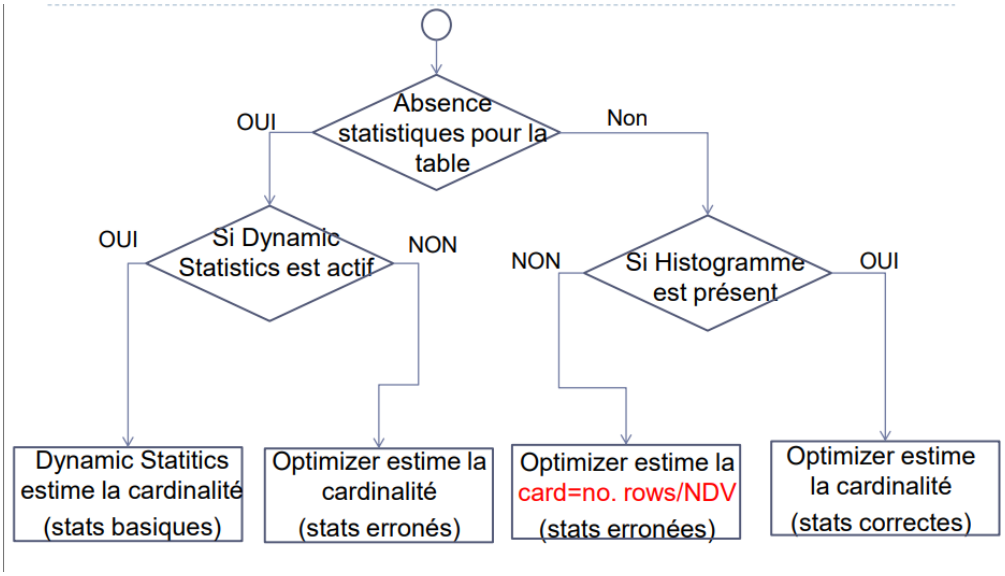
1. **Moment d'activation** : Quand les statistiques dynamiques doivent être utilisées.
2. **Taille de l'échantillon** : La quantité de données à échantillonner pour collecter les statistiques.

- **Niveau 0** : Désactive les statistiques dynamiques pour l'optimiseur.
- **Niveau 2** : Active les statistiques dynamiques si une ou plusieurs tables de la requête n'ont pas de statistiques, avec un échantillon pouvant atteindre **64 blocs**.

Exemple de configuration du niveau :

```
ALTER SESSION SET OPTIMIZER_DYNAMIC_SAMPLING = 6;
```

Synthèse



Chapitre 6 : Oracle Hints

Hint

Les *hints* en SQL sont des instructions ou des commentaires adressés à l'optimiseur de la base de données, l'incitant à choisir un plan d'exécution différent pour une requête SQL. Elles sont utilisées principalement lorsque les statistiques sont à jour mais que l'optimiseur ne choisit toujours pas le plan optimal.

Points clés :

Résumé SQL Tuning

1. **But** : Les *hints* influencent le choix du plan d'exécution par l'optimiseur.
2. **Utilisation** : À utiliser en dernier recours, lorsque les statistiques sont correctes mais que la requête ne suit pas un plan optimal.
3. **Syntaxe** : Un *hint* est placé dans un commentaire et suit le mot-clé de la requête (SELECT, UPDATE, INSERT, MERGE, DELETE).
    - Exemple : SELECT /*+ hint */ ...
4. **Types de requêtes** : Les *hints* sont utilisés avec des requêtes SELECT, UPDATE, INSERT, MERGE, et DELETE.

Exemples de *hints* :

- **FIRST_ROWS(n)** : Optimise la récupération des premières n lignes.

```
SELECT /*+ FIRST_ROWS(15) */ * FROM emp WHERE deptno = 10;
```

- **ALL_ROWS** : Optimise la requête pour une meilleure performance globale, pas seulement pour les premières lignes.

```
SELECT /*+ ALL_ROWS */ * FROM emp WHERE deptno = 10;
```

- **PARALLEL** : Suggère une exécution parallèle pour une table.

```
SELECT /*+ PARALLEL */ FROM emp;
```

- **NO_INDEX :** Interdit l'utilisation d'un index spécifique.

```
SELECT /*+ NO_INDEX(emp, emp_dept_idx) */ FROM emp, dept WHERE emp.deptno = dept.deptno;
```

- **INDEX** : Force l'utilisation d'un index spécifique.

```
SELECT /*+ INDEX(e emp_dept_idx) */ FROM emp e;
```

- **LEADING** : Spécifie l'ordre des tables dans une jointure.

```
SELECT /*+ LEADING(dept) */ FROM emp, dept WHERE emp.deptno = dept.deptno;
```

- **USE_MERGE, USE_HASH, USE_NL** : Influence le type de jointure (fusion, hachage, boucle imbriquée).

```
SELECT /*+ USE_MERGE(emp dept) */ FROM emp, dept WHERE emp.deptno = dept.deptno;
```

- **APPEND** : Optimise l'insertion de données dans une table.

```
INSERT /*+ APPEND */ INTO mytab SELECT /*+ CACHE (e) */ FROM emp e;
```

- **GATHER_PLAN_STATISTICS** : Collecte des statistiques d'exécution de la requête, ce qui peut être utile pour l'analyse des plans d'exécution.

```
SELECT /*+ GATHER_PLAN_STATISTICS */ * FROM emp WHERE deptno = 10;
```

Chapitre 7 : Oracle SQL Tuning

Introduction à SQL Tuning

Résumé SQL Tuning

Le **SQL Tuning** (Réglage SQL) est un processus itératif visant à améliorer les performances des requêtes SQL pour atteindre des objectifs spécifiques et mesurables.

Types de Tuning :

1. **Tuning proactif** : On utilise régulièrement des outils comme SQL Tuning Advisor pour identifier les opportunités d'amélioration des performances des requêtes SQL.
2. **Tuning réactif** : On intervient pour résoudre un problème spécifique lié à une requête SQL rencontrée par un utilisateur.

Outils de tuning

1. **Un outil est automatisé** si la base de données elle-même peut fournir un diagnostic, des conseils ou des actions correctives. Exemple d'outils: ADDM, SQL Tuning Advisor,
2. **Un outil manuel** nécessite que vous effectuiez toutes ces opérations.

Tous les outils de tuning dépendent des vues dynamiques de performances, des statistiques et des métriques que l'instance de base de données collecte.

Tâches de SQL Tuning

Utilisant un outil automatisé ou manuel, SQL Tuning comporte les tâches suivantes:

1. Identifier les instructions SQL à charge élevée.
2. Collecter des données relatives aux performances.
3. Déterminer les causes des problèmes de performances SQL.
4. Définir la portée du problème.
5. Implémenter des actions correctives pour les instructions SQL moins performante (éviter hard parsing, utiliser equi-jointures, éviter les fonctions dans where, etc).
6. Prévenir les régressions des performances SQL.

Causes de Problèmes de Performance dans SQL

- **Mauvaise conception des instructions SQL** : Par exemple, des jointures cartésiennes, l'utilisation de *hints* pour forcer une grande table en tant que table principale dans une jointure, ou des sous-requêtes exécutées pour chaque ligne.
- **Choix des plans d'exécution sous-optimaux** : Lorsque l'optimiseur sélectionne un plan inefficace, comme une analyse complète de table au lieu d'un index pour une requête à faible sélectivité.
- **Structures d'accès SQL manquantes** : L'absence d'index ou de vues matérialisées peut mener à des performances sous-optimales.
- **Statistiques d'optimisation obsolètes** : Des statistiques non mises à jour peuvent entraîner des décisions erronées par l'optimiseur.
- **Problèmes matériels** : Des problèmes liés à la mémoire, aux entrées/sorties (E/S), ou au processeur peuvent également affecter les performances.

Tuning manuel

Le **tuning manuel SQL** utilise plusieurs outils pour diagnostiquer et optimiser les requêtes :

1. **Plans d'exécution** : Ils sont essentiels pour le diagnostic. Différentes méthodes pour les afficher incluent :

- **DBMS_XPLAN** : Affiche les plans d'exécution générés par la commande Explain for.
- **V$SQL_PLAN** : Vue contenant des informations sur les plans d'exécution des requêtes exécutées.
- **AUTOTRACE** : Commande SQL*Plus qui génère le plan d'exécution et les statistiques des performances d'une requête.

2. **Real-Time SQL Monitoring** : Fonctionnalité de surveillance des performances des requêtes pendant leur exécution. Elle se déclenche automatiquement pour les requêtes exécutées en parallèle ou prenant plus de 5 secondes de CPU ou d'E/S. Le hint /*+ MONITOR*/ permet de surveiller même les requêtes rapides.
3. **SQL Trace Facility et TKPROF** : Ces outils permettent d'évaluer l'efficacité des instructions SQL. SQL Trace crée des fichiers de trace détaillant les performances, et TKPROF formate ces fichiers pour générer des rapports.
   - Le fichier de trace contient des informations sur les analyses, les exécutions, les lectures, le CPU, etc.
   - La traçabilité peut se faire au niveau de la session ou de l'instance (via les paramètres INIT.ora ou SQL>alter session).

TKPROF

TKPROF est un programme qui formate le contenu des fichiers de trace SQL, les rendant lisibles. Il peut aussi générer des plans d'exécution et des scripts SQL pour enregistrer les statistiques dans la base de données.

Voici les arguments principaux de TKPROF :

1. **filename1** : Spécifie le fichier d'entrée (fichier de trace SQL).
2. **filename2** : Spécifie le fichier de sortie formaté par TKPROF.
3. **AGGREGATE** : Par défaut, TKPROF agrège plusieurs utilisateurs du même texte SQL. Si AGGREGATE = NO est défini, il ne les agrège pas.
4. **EXPLAIN** : Permet de déterminer et d'ajouter le plan d'exécution de chaque instruction SQL dans le fichier de sortie.
5. **TABLE** : Spécifie la table où TKPROF place temporairement les plans d'exécution avant de les écrire dans le fichier de sortie.
6. **INSERT** : Crée un script SQL pour stocker les statistiques du fichier de trace dans la base de données.
7. **SORT** : Trie les instructions SQL selon des critères comme :
   - **PRSCNT** : Nombre de fois que la requête a été analysée.
   - **PRSCPU** : Temps CPU pour l'analyse.
   - **PRSELA** : Temps écoulé pour l'analyse.
   - **PRSDSK** : Nombre de lectures physiques effectuées.

- **PRSQRY** : Nombre de buffers lus en mode courant.
- **PRSCU** : Nombre de buffers lus en mode consistant.
- **PRSMIS** : Nombre de "misses" du cache de bibliothèque.
- **EXEROW/FCHROW** : Nombre de lignes traitées et renvoyées pendant l'exécution/fetch.

PRINT : Liste les 1ers instructions SQL triées par nombre d'occurrences dans le fichier de sortie.

Exemple : **TKPROF ora53269.trc ora 53269.prf SORT = (PRSDSK, EXEDSK, FCHDSK) PRINT = 10**

Dans cet exemple, l'instruction suivante imprimera les dix instructions du fichier de trace qui ont généré le plus d'E/S physiques.

Rapport TKPROF

Le **rapport TKPROF** est généré après l'exécution de la commande **TKPROF** sur un fichier de trace SQL. Il contient 3 sections principales : l'en-tête, le corps du rapport TKPROF et les appels récursifs.

1. Option Explain :

Lorsque l'option EXPLAIN est utilisée, TKPROF génère également les plans d'exécution pour chaque requête SQL tracée. Par exemple, la commande suivante :

tkprof hrdb_ora_6712.trc hrdb_ora_6712.tkp explain=hr/hr@pdb

Cette commande permet de se connecter en tant qu'utilisateur hr et d'exécuter la commande EXPLAIN PLAN pour chaque instruction SQL tracée.

2. Structure du rapport :

Le rapport généré par TKPROF contient généralement les sections suivantes :

- **En-tête** : Informations générales sur le fichier de trace et la commande utilisée.
- **Corps du rapport TKPROF** : Détails sur les requêtes SQL exécutées, y compris les plans d'exécution et les statistiques de performance.
- **Appels récursifs** : Certaines instructions SQL génèrent des appels récursifs, par exemple, lors de l'ajout de nouvelles lignes dans une table ou lorsque des informations du dictionnaire de données doivent être récupérées depuis le disque.

3. Appels récursifs :

Les appels récursifs sont générés lorsqu'Oracle doit émettre des instructions supplémentaires pour compléter une requête. Par exemple :

- Allocation d'espace dans une table.
- Récupération des informations du dictionnaire de données depuis le disque si elles ne sont pas dans le cache.

Lorsque des appels récursifs sont présents dans le fichier de trace, TKPROF inclut ces statistiques dans le rapport et les marque clairement, ce qui permet d'analyser leur impact sur les performances.

Chapitre 8 : Outils automatisés de Oracle SQL Tuning

Tuning proactif avec SQL Tuning Advisor

Le **SQL Tuning Advisor** est un outil qui analyse les requêtes SQL et fournit des recommandations pour les optimiser. Il peut traiter une ou plusieurs requêtes et utilise l'**AutomaticTuning Optimizer** pour appliquer des réglages afin d'améliorer les performances des requêtes.

Fonctionnement :

- **Lancement manuel ou automatique** : Il peut être lancé à la demande du DBA ou de manière automatique comme une tâche de maintenance (Automatic SQL Tuning Advisor).
- **Recommandations** : Le SQL Tuning Advisor propose des actions spécifiques pour améliorer les performances des requêtes SQL, telles que :
    - Collecte de statistiques sur les objets,
    - Création de nouveaux index,
    - Restructuration des instructions SQL,
    - Création d'un profil SQL.

Ces recommandations sont fournies dans un rapport détaillé, avec les avantages attendus.

Privilèges nécessaires :

Pour que les utilisateurs non-**sysdba** utilisent SQL Tuning Advisor, des privilèges spécifiques doivent leur être attribués :

- **grant advisor to hr;**
- **grant administrer sql tuning set to hr;**

SQL Profile

Un **SQL profile** est un objet stocké dans le dictionnaire de la base de données, conçu pour corriger les estimations d'optimiseur sous-optimales identifiées lors du SQL Tuning automatique. Il est créé par l'**Automatic Tuning Optimizer** en utilisant les données collectées lors de l'exécution réelle des requêtes. Cela permet d'améliorer la précision des estimations de cardinalité et de sélectivité.

Pour afficher les profils SQL du dictionnaire :

```
SQL> SELECT * FROM DBA_SQL_PROFILES;
```

SQL Access Advisor

Le **SQL Access Advisor** est un outil qui aide à améliorer les performances SQL en recommandant la création, la suppression ou la conservation d'**index**, de **vues matérialisées**, de **journaux de vues matérialisées** ou de **partitions**. Il est essentiel pour optimiser les requêtes complexes et gourmandes en ressources. Pour fonctionner, il nécessite une charge de travail composée de requêtes SQL et de leurs statistiques, provenant de la **Zone SQL partagée** ou d'un **Ensemble de réglages SQL (STS)**.

Résumé SQL Tuning

SQL Tuning Set (STS)

Le **SQL Tuning Set (STS)** est un ensemble de requêtes SQL et des informations associées, permettant d'appliquer des outils de tuning comme le **SQL Tuning Advisor** et le **SQL Access Advisor**. Il regroupe des requêtes stockées dans le **SHARED POOL**, **AWR** ou d'autres sets. Pour créer un SQL Tuning Set, vous pouvez utiliser la commande suivante :

```
SQL> begin
dbms_sqltune.create_sqlset(sqlset_name=>'Elbeggar_STS',
description=>'elbeggar SQL Tuning Set');
end;
```

Remplissage du SQL Tuning Set

Ajouter des requêtes à SQL Tuning Set Par exemple, le programme PL/SQL suivant remplit le STS précédent avec toutes les instructions de cache de curseur qui appartiennent au schéma hr :

```
SQL> DECLARE
 c_sqlarea_cursor DBMS_SQLSET.SQLSET_CURSOR;
BEGIN
 OPEN c_sqlarea_cursor FOR
  SELECT VALUE(p)
  FROM TABLE(
   DBMS_SQLSET.SELECT_CURSOR_CACHE(
    'module = ''SQLT_WKLD'' AND parsing_schema_name = ''HR''')
  ) p;
 -- Charger le tuning set
 DBMS_SQLSET.LOAD_SQLSET (
  sqlset_name => 'Elbeggar_STS',
  populate_cursor => c_sqlarea_cursor
 );
END;
```

SQL Tuning Advisor avec les STS

Pour lancer le **SQL Tuning Advisor** avec un **SQL Tuning Set (STS)**, vous pouvez créer une tâche de maintenance avec le code suivant :

```
SQL> DECLARE
 l_sql_tune_task_id VARCHAR2(100);
```

```
BEGIN
 l_sql_tune_task_id := DBMS_SQLTUNE.create_tuning_task (
  sqlset_name => 'Elbeggar_STS',  -- Nom du SQL Tuning Set
  scope => DBMS_SQLTUNE.scope_comprehensive,  -- Scope complet pour un tuning détaillé
  time_limit => 60,  -- Limite de temps en secondes
  task_name => 'elbeggar_TUNING_TASK',  -- Nom de la tâche de tuning
  description => 'Tuning task for an SQL tuning set.'  -- Description de la tâche
 );
 DBMS_OUTPUT.put_line('l_sql_tune_task_id: ' || l_sql_tune_task_id);
END;
```

Dégradation des performances suite aux changements des plans d'exécution

La dégradation des performances due aux changements des plans d'exécution dans Oracle peut être causée par des facteurs tels que la mise à niveau d'Oracle Optimizer, la mise à jour des statistiques, des changements dans le schéma ou les paramètres d'optimisation. Pour stabiliser les performances, il est recommandé d'utiliser des *Plan Baseline* pour forcer l'utilisation d'un plan spécifique, de maintenir des statistiques à jour avec DBMS_STATS, et de vérifier les impacts des modifications du schéma sur les plans d'exécution.