Bases de données avancées

Agnès Braud
agnes.braud@unistra.fr

Organisation du module

- Cours : 2 séances
- TD : 5 séances
- TP : 7 séances

Évaluation du module :

- contrôle intermédiaire d'1h pendant la séance de TD du jeudi 12 octobre de 9h20 à 10h20
- contrôle final de 2h en semaine 8
- Note du module = 1/3 CI + 2/3 CF

Programme du module

- Révisions SQL, catalogue système
- Sécurisation des données : privilèges, vues, transactions
- Optimisations (orientées développeur, pas administrateur de bases de données)
  - Optimisation au niveau de la structure de la base de données - Index
  - Optimisation de requêtes (comprendre, manipuler les outils)
- PL/SQL (SQL procédural)

Programme du cours d'aujourd'hui

- Organisation du stockage des données dans une base Oracle
- Index
- Exécution d'une requête, optimiseur, plan d’exécution

Références

Disponibles à la bibliothèque

- SQL - 4e édition 2012. Frédéric Brouard, Christian Soutou, Rudi Bruchez. Collection Synthex Informatique. Pearson. Juillet 2012
- Optimisation des bases de données - Mise en œuvre sous Oracle. Laurent Navarro. Pearson. Juin 2010
- SQL pour Oracle - Applications avec Java, PHP et XML - Optimisation des requêtes et schémas - Avec 50 exercices corrigés. Christian Soutou. Collection Noire. Eyrolles. 2015 (disponible en ligne
- SQL : Au cœur des performances. Markus Winand. 2013

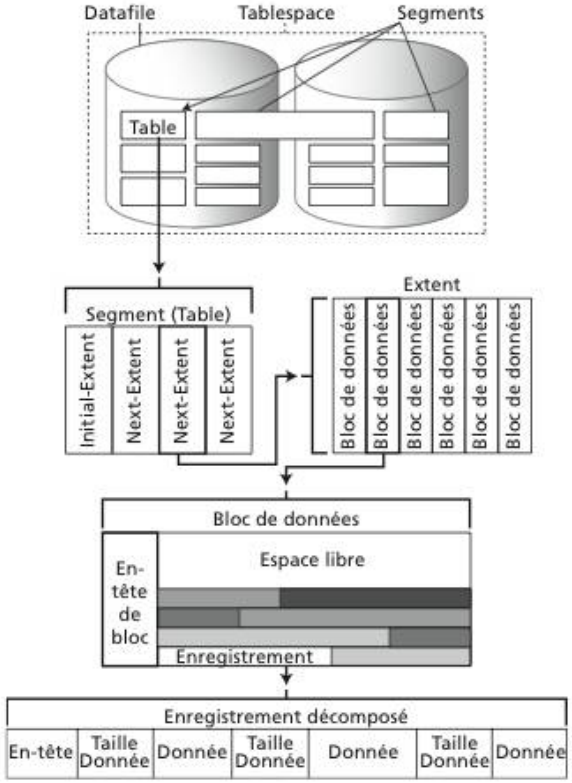
NB : des ressources sont disponibles en ligne (voir bu.unistra.fr, dans Recherche avancée choisir Ressources en ligne comme contexte de recherche)

Références

- Cours de bases de données – Aspects systèmes. Philippe Rigaux. http://sys.bdpedia.fr/index.html
- MOOC Databases for Developers: Performance de Chris Saxon

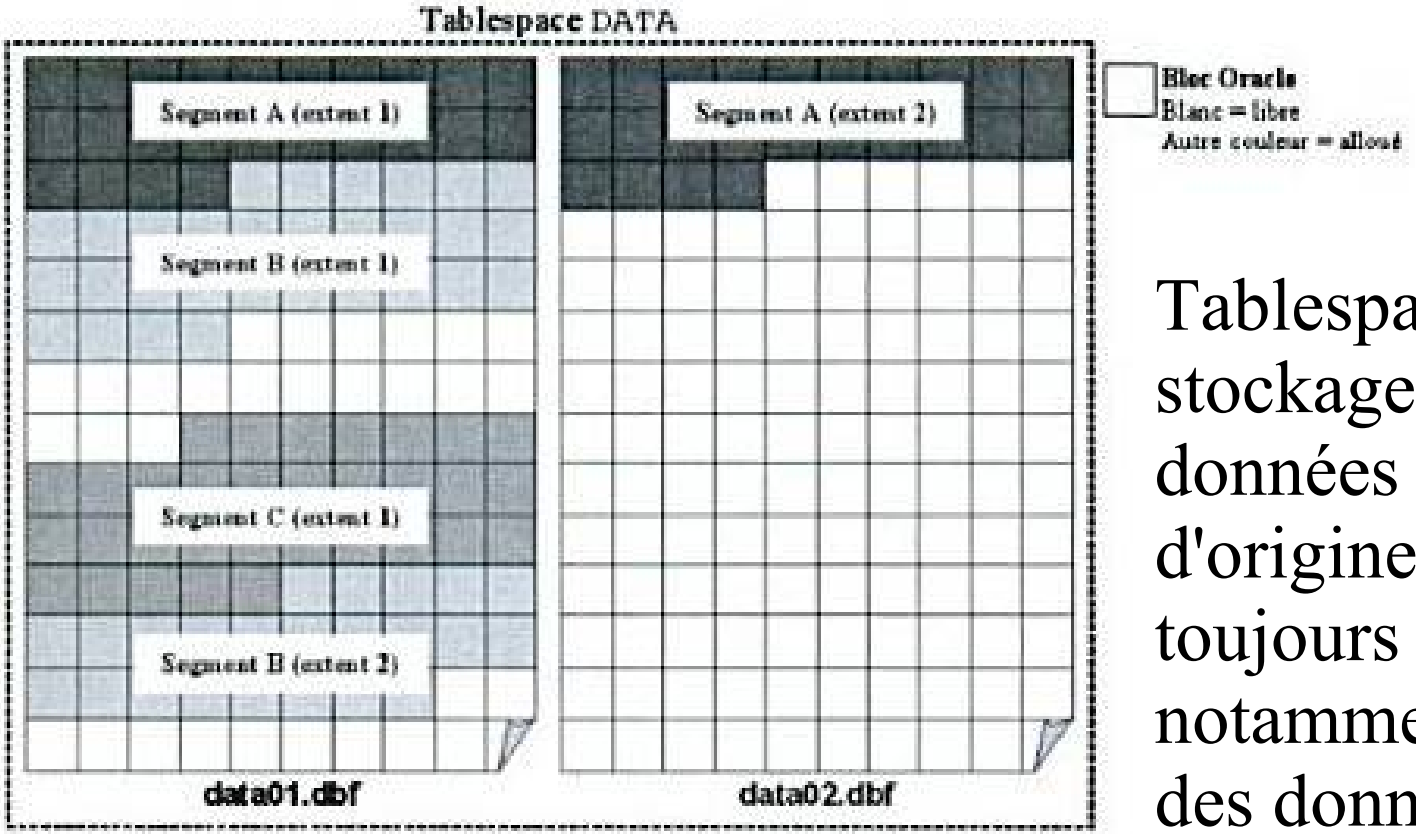
Organisation du stockage

Organisation du stockage des données dans une base Oracle



- Les données sont stockées dans des fichiers de données
- Les fichiers de données sont découpés en blocs
  Le bloc = l'unité de base pour Oracle (plus petite unité d'entrée/sortie, fichiers organisés en blocs, cache de données également)
- Extension = ensemble de blocs contigus
- Segment = espace occupé par un objet (table, index, ...)
  Segment constitué d'extensions
- Création d'un segment → allocation d'extensions. En cas d'insertion, si plein, nouvelle allocation (pas forcément contiguë !)

Organisation du stockage des données dans une base Oracle

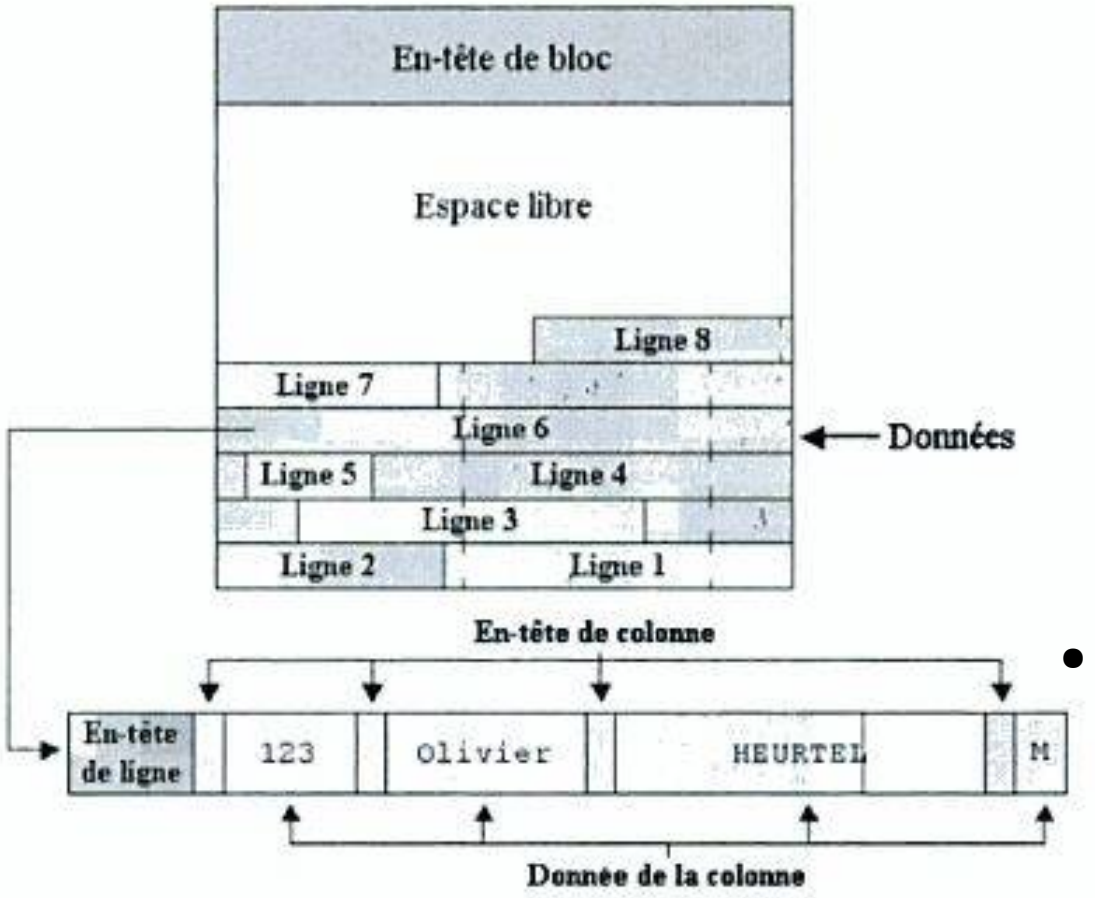


Extrait du livre « Oracle 11g – Administration » d'Olivier Heurtel

Tablespace : espace de stockage logique pour les données (ex. le tablespace d'origine, SYSTEM, toujours présent qui contient notamment le dictionnaire des données). Correspond à plusieurs fichiers physiques

Un segment s'étend sur un unique tablespace

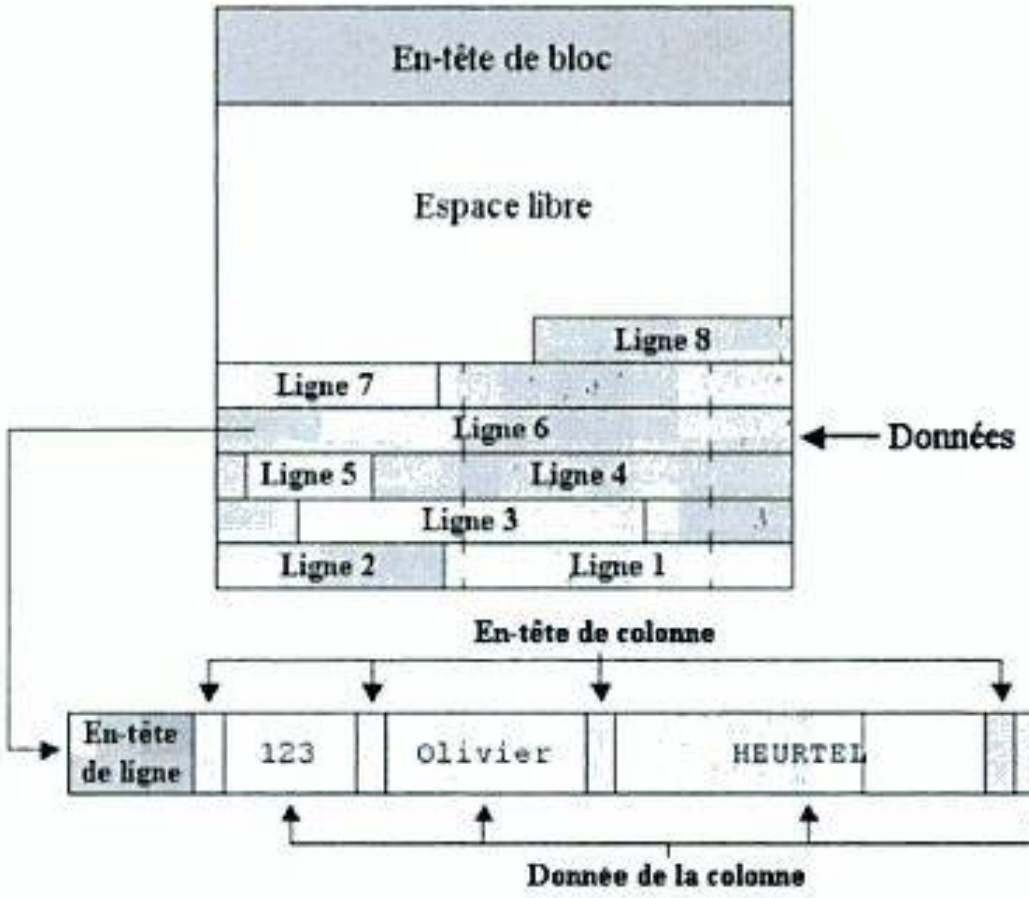
Structure des blocs



Extrait du livre « Oracle 11g – Administration » d'Olivier Heurtel

- En-tête d'un bloc (adresse du bloc, type de segment, répertoire des tables (en cas de cluster), des lignes – de 100 à 200 octets). Grossit vers le bas (insertions → répertoire de lignes grossit, ne rétrécit jamais)
- Reste du bloc : données et espace libre (pour insertion de nouvelles données ou modifications - il n'y a plus d'insertions en dessous d'un seuil d'espace libre afin de permettre les modifications)

Structure des lignes



Extrait du livre « Oracle 11g – Administration » d'Olivier Heurtel

- En-tête d'une ligne (nombre de colonnes, chaînage éventuel si ligne trop grande pour tenir dans un bloc, verrou).
- Puis colonnes (en-tête+valeur)

ROWID

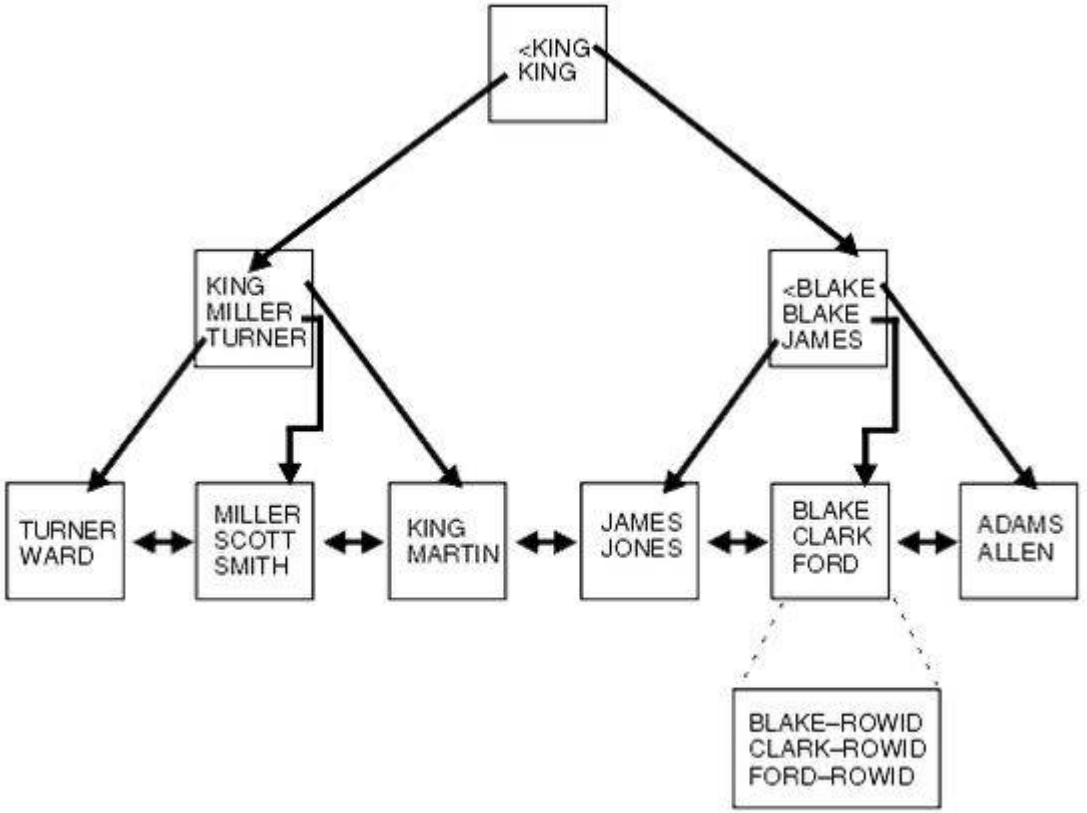
- Colonne présente dans chaque table
- Donne l'adresse physique de stockage de la ligne
- Utilisé par Oracle pour les index

Index

Index

- Une définition : « Structure de données organisée de manière à accélérer certaines recherches, afin de diminuer de manière très sensible la durée des traitements (recherches, tris, groupes, jointures, etc...) » (Brouard, Bruchez, Soutou)
- Sans index l'accès aux lignes des tables se fait séquentiellement
- Idée : répartir les données dans différents sous-espaces plus fins (donc contenant moins de données) dont les limites sont connues et ordonnées

Exemple d'index



Source : doc oracle

Index

- Copie des données sur lesquelles porte l'index (la clé de l'index)
- Organisation de ces données
- Pointeur vers les données initiales

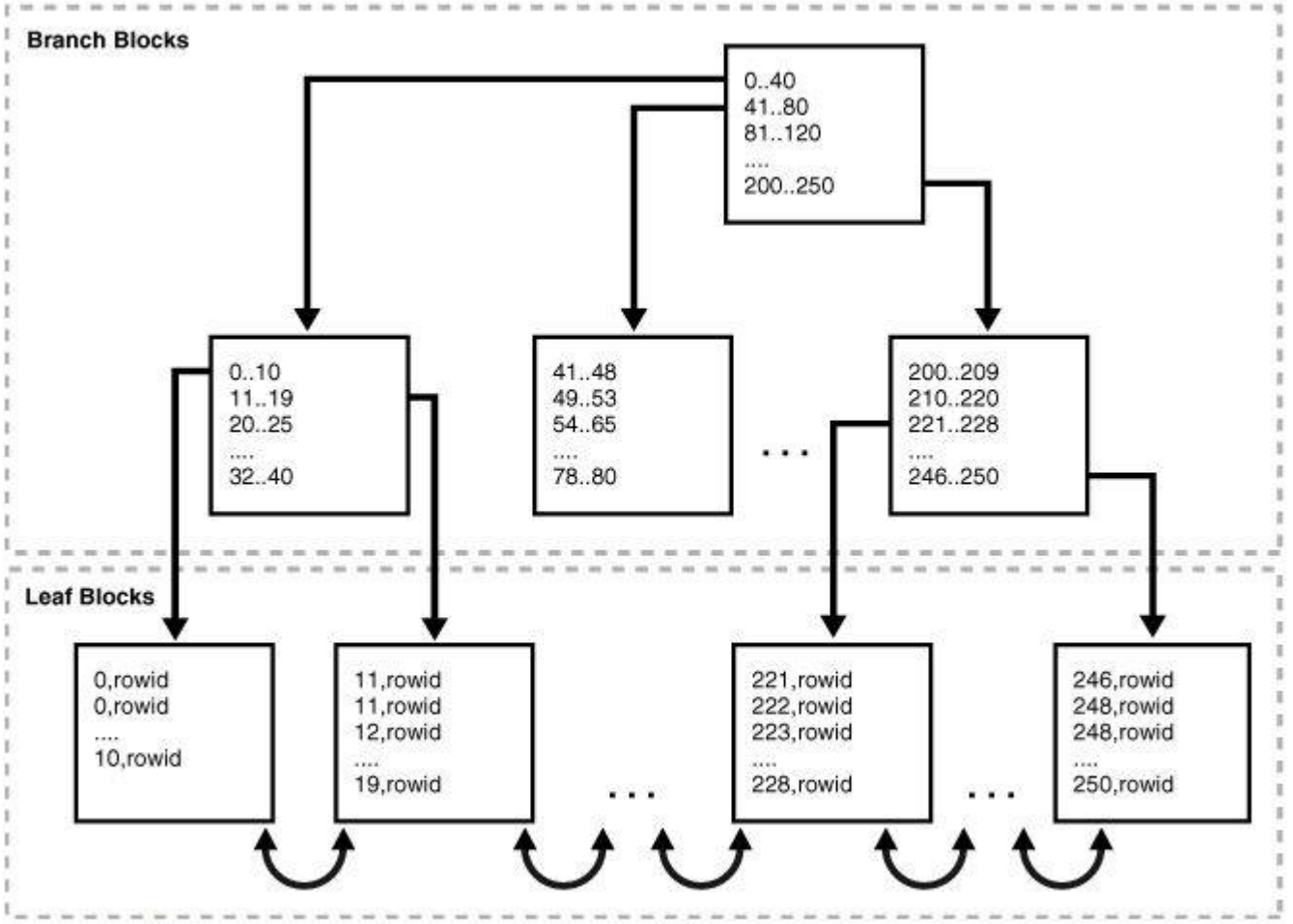
Des types d'index

- B-tree, B+-Tree, B*-Tree
- Bitmap

- Index à clé inversée
- Index sur fonction
- Index composé (plusieurs colonnes dans la clé. Efficace pour des requêtes portant sur des champs placés en tête de l'index) → placer les colonnes les plus interrogées en début d'index, et les plus restrictives

Index B+-Tree

Arbre équilibré (Balanced Search Tree) : tous les chemins racine-feuille ont la même longueur



Description of "Figure 3-1 Internal Structure of a B-tree Index"

Extrait doc Oracle

Index B+-Tree

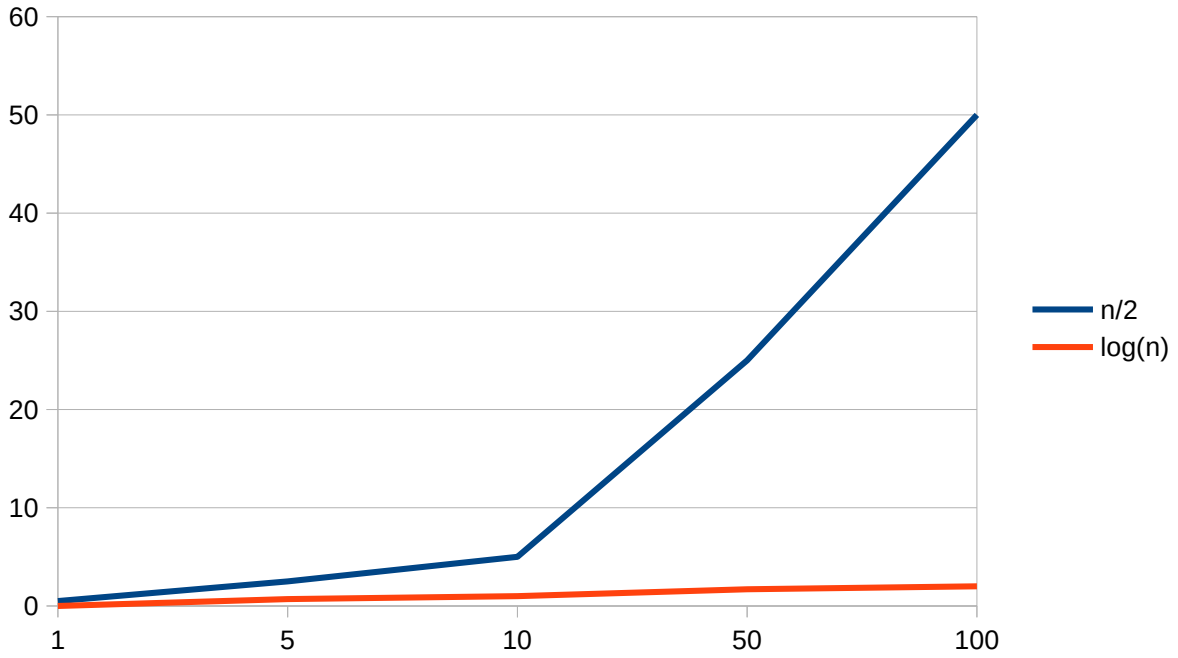
Soit une table de n lignes SANS index

- Pour trouver une ligne unique, il faut en moyenne parcourir n/2 lignes
- Pour trouver toutes les occurrences d'une valeur dans une colonne il faut parcourir n lignes

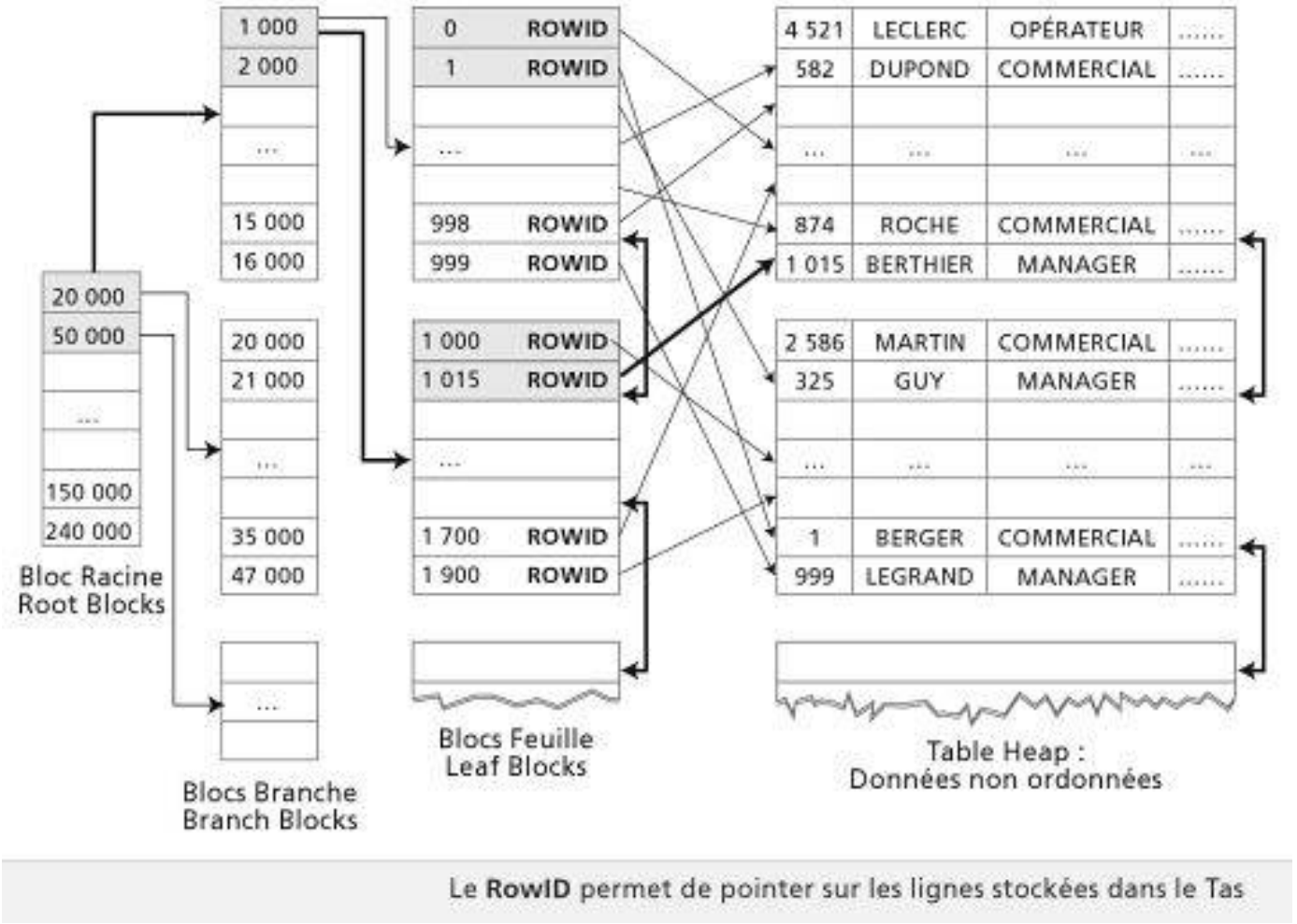
Soit une table de n lignes AVEC index

- Pour trouver une ligne, il faut en moyenne parcourir une proportion de log(n) lignes

Index B+-Tree

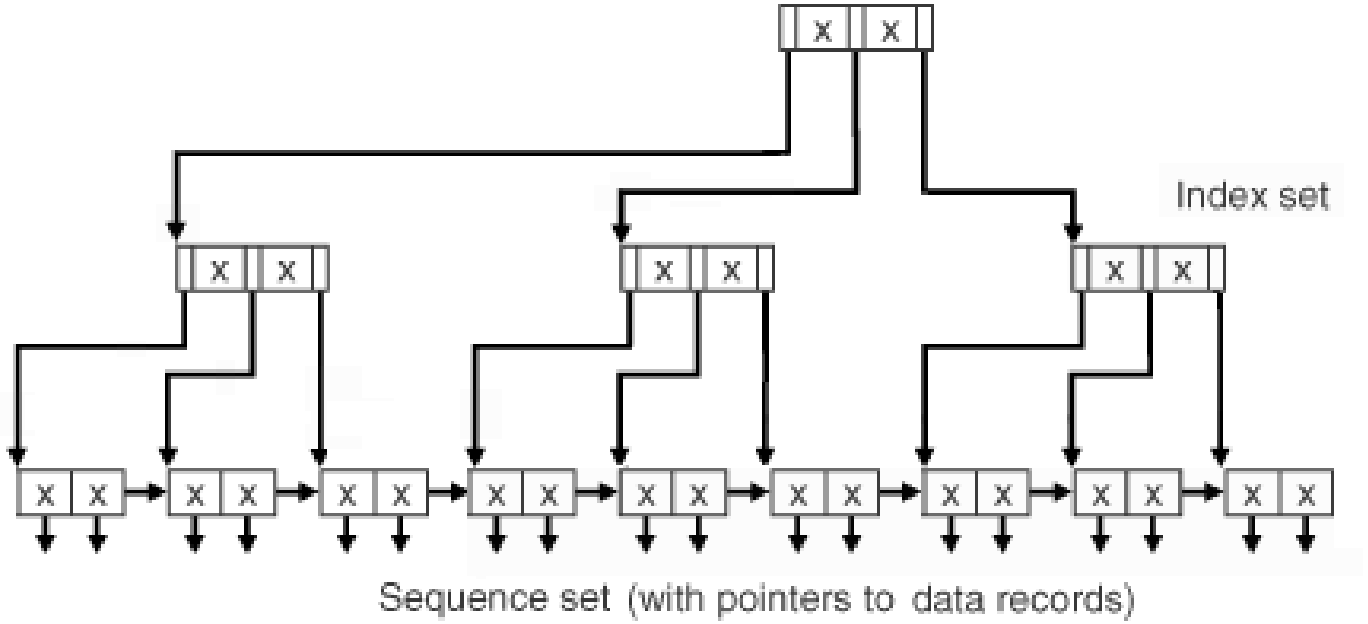


Structure d'un index B+-tree



Extrait du livre « Optimisation de bases de données » - Laurent Navarro

Chaînage des nœuds



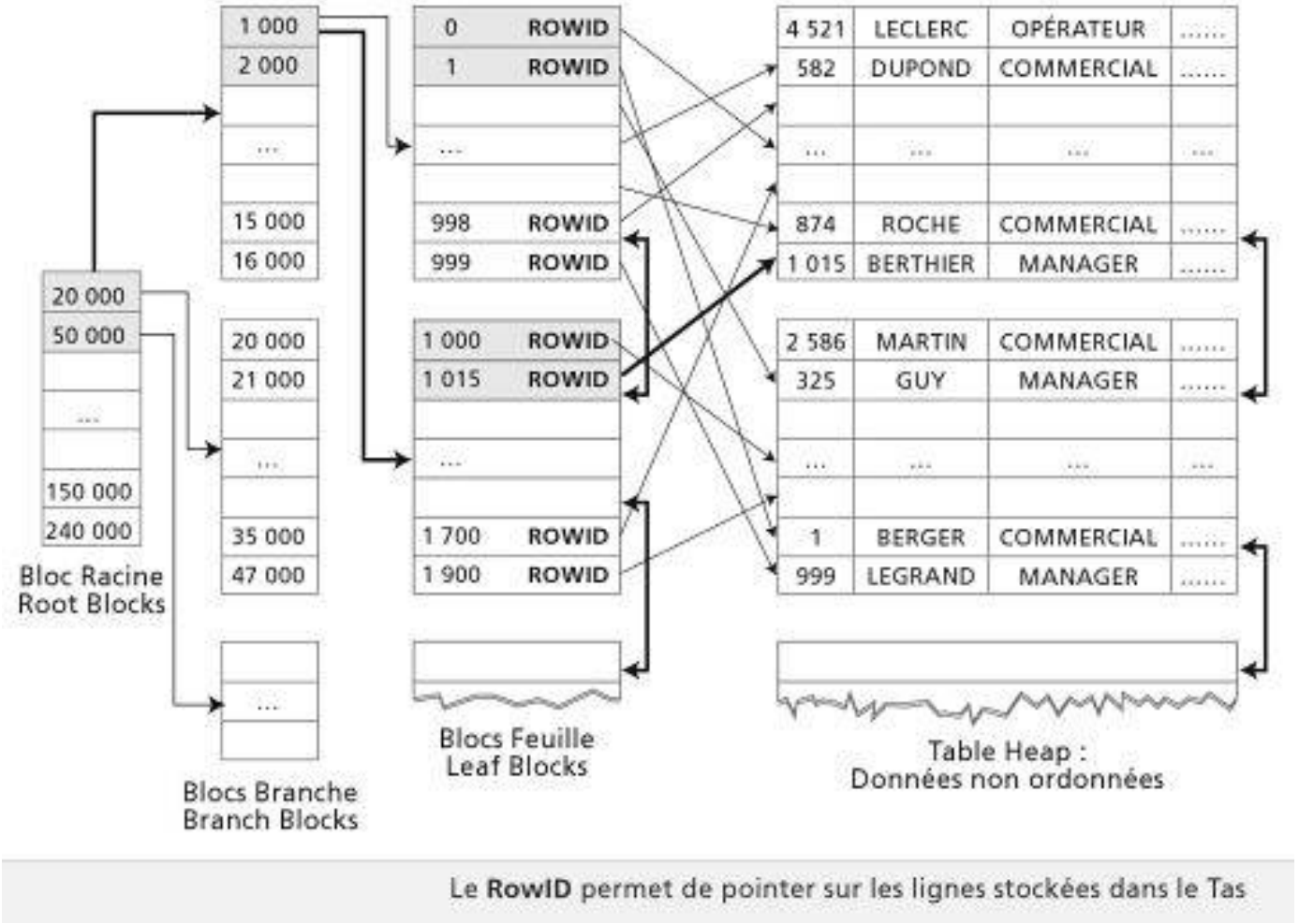
Source :
http://docs.oracle.com/cd/B28359_01/appdev.111/b28425/ext_idx_frmwork.htm

Obtention d'une ligne

Obtenir la ligne ayant pour clé 1015 dans l'index B+-Tree :

- Parcours du bloc racine jusqu'à rencontrer la valeur supérieure
- Aller sur le bloc pointé par la valeur qui précède
- Répéter l'opération dans le bloc branche
- Dans le bloc feuille, rechercher la valeur (arrêt à la valeur supérieure si la valeur n'est pas présente), l'entrée pointée permet d'obtenir le ROWID de la ligne
- Le ROWID est obtenu dans le « tas »

Obtention d'une ligne



Extrait du livre « Optimisation de bases de données » - Laurent Navarro

=> seulement 3 blocs de données à lire

Index B+-Tree

- Moins il y a de lignes dans la table, moins l'intérêt est grand : il n'est pas pertinent de créer un index sur une petite table (moins de 1000 lignes)
- Par défaut un index B+-Tree est créé automatiquement pour toutes les clés primaires
- S'il y a plus de 5% des valeurs retournées (sélectivité), l'optimiseur n'utilise plus l'index

Efficacité d'un index

- Économie en nombre de pages lues

  Exemple : une recherche sur le nom parmi 1 milliard de lignes (Facebook par exemple) peut se résumer en un B-tree de 3 étages et il est nécessaire de lire 4 pages (blocs) uniquement (on suppose des blocs de 8 Ko)

- Sélectivité
  nombre de valeurs distinctes / nombre de lignes
  (1 pour une clé primaire)
  Sélectivité augmente → nombre de lignes retournées pour la recherche d'une valeur diminue

Bon à savoir

- Les index sont toujours synchronisés avec les données de la table
- Une maintenance des index est souhaitable, insertions et suppressions de données provoquent une fragmentation
- L'index entraîne un coût de traitement lors des opérations de mise à jour
- Les optimiseurs des SGBDR type Oracle ou SQL Server sont meilleurs sur l'indexation et l'optimisation que des SGBDR type PostgreSQL ou MySQL

Syntaxe

Index créés dans le create table ou ajoutés à une table existante :

```
CREATE [ONLINE|OFFLINE] [UNIQUE|FULLTEXT|SPATIAL]
INDEX index_name [index_type]
ON tbl_name (index_col_name,...)[index_option] ...


index_col_name:
      col_name [(length)] [ASC | DESC]

index_type:
   USING {BTREE | HASH}

index_option:
      KEY_BLOCK_SIZE [=] value
    | index_type
    | WITH PARSER parser_name
```

Taille d'un index

- L= long_ligne_ind = longueur de la donnée indexée + longueur du ROWID + octets pour les informations de contrôle
- B = taille de bloc (paramètre DB_BLOCK_SIZE)

  espace disponible pour les lignes de l'index = B – taille en-tête

Type	Longueur du stockage
char(n)	n octets
varchar2(n)	Variable entre 0 et n octets
number(x,y)	Variable entre 1 et 21 octets
date	8 octets

Exercice : taille d'un index

- L= long_ligne_ind = longueur de la donnée indexée + longueur du ROWID + octets pour les informations de contrôle
- espace disponible pour les lignes de l'index = taille bloc - taille en-tête

On suppose

- Nombre de données = 1000000
- Longueur ROWID = 8 octets
- B = taille de bloc = 8192 octets
- Taille de l'en-tête du bloc = 100 octets
- Taille de la donnée indexée = 8 octets
  (on néglige les informations de contrôle)

- → Nombre maximal d'enregistrements par bloc ?
- → Nombre de blocs d'index nécessaires pour le dernier niveau de l'arbre ?
- → Taille de l'index ?

Exercice : taille d'un index

- L= long_ligne_ind = longueur de la donnée indexée + longueur du ROWID + octets pour les informations de contrôle
- espace disponible pour les lignes de l'index = taille bloc – taille en-tête

On suppose

- Nombre de données 1000000
- Longueur ROWID = 8 octets
- B = taille de bloc = 8192 octets
- Taille de l'en-tête du bloc = 100 octets
- Taille de la donnée indexée = 8 octets
  (on néglige les informations de contrôle)

→ Nombre maximal d'enregistrements par bloc ?
(B - taille en-tête)/(longueur ROWID+taille donnée indexée)=(8192-100)/(8+8)=environ 505

→ Nombre de blocs d'index nécessaires pour le dernier niveau de l'arbre ?
Nb données/nb max enreg par bloc = 1000000 / 505 = environ 1977
Pour l'arbre complet : ajouter les autres niveaux

→ Taille de l'index ? Nb données * (taille donnée indexée+ROWID) = 1000000 * 16 octets → non négligeable

Les B-Tree

- N'indexent pas les valeurs NULL
- Très efficaces lorsque la cardinalité (nombre de valeurs différentes) est forte
- Fréquemment utilisés dans des environnements OLTP (mises à jour nombreuses, opposition avec les environnements pour le décisionnel plus orientés interrogation)

Index bitmap

Index bitmap pour la colonne job

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
ANALYST	0	0	0	0	0	0	0	1	0	1	0	0	1	0
CLERK	1	0	0	0	0	0	0	0	0	0	1	1	0	1
MANAGER	0	0	0	1	0	1	1	0	0	0	0	0	0	0
PRESIDENT	0	0	0	0	0	0	0	0	1	0	0	0	0	0
SALESMAN	0	1	1	0	1	0	0	0	0	0	0	0	0	0

Les index Bitmap

- Coûteux à mettre à jour, utilisation en environnement OLTP limitée
- Très performants en cas de sélectivité faible
- Indexent les valeurs NULL

Recommandés si :

- La cardinalité des colonnes est faible (Très peu de valeurs différentes)
- Il y a beaucoup de lignes dans votre table
- Lorsqu'il y a quasiment uniquement des activités de lecture sur la table

Syntaxe

```
CREATE BITMAP INDEX <index name>  ON <table name>
(<column name>, [column name]...) …
```

Recommandations

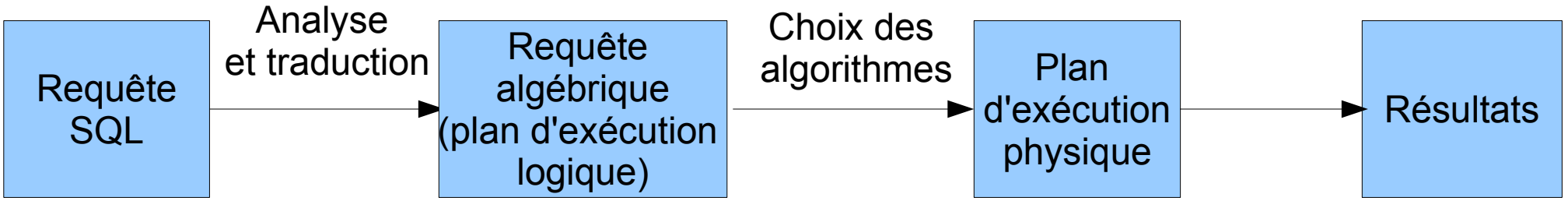
- Pour créer de bons index, il faut :
  - Comprendre leur fonctionnement
  - Connaître les besoins de l'application en termes d'interrogation des données
- Ne pas placer des index B-Tree si beaucoup de valeurs dupliquées
- Indexer les clés étrangères (si ce n'est pas fait automatiquement)

Exécution d'une requête

Exécution d'une requête

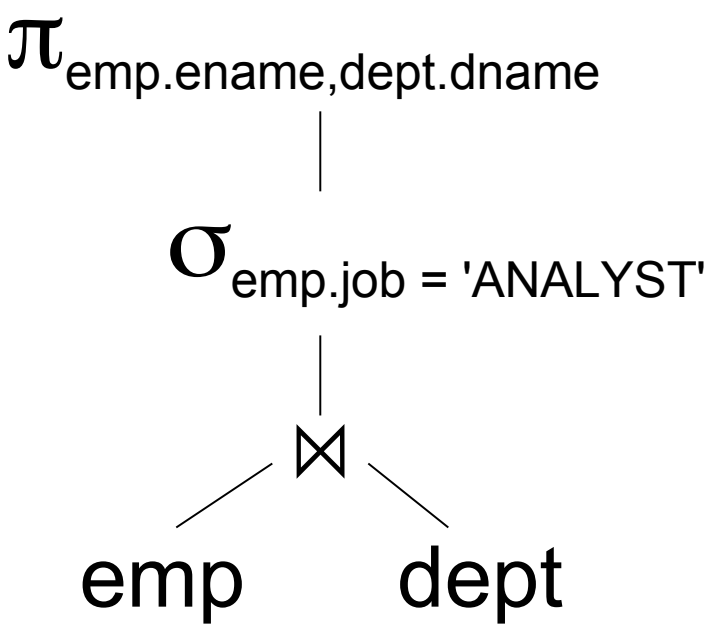
- Une requête est déclarative, elle décrit le résultat attendu → elle ne dit pas comment calculer ce résultat
- Pour obtenir le résultat, nécessité d'une forme opératoire : un programme
- C'est le SGBD qui va déterminer le bon programme à exécuter pour une requête donnée, en fonction des ressources et de l'organisation des données
- Forme opératoire = plan d'exécution

Exécution d'une requête



Arbres algébriques

Une requête SQL est équivalente à une requête en algèbre relationnelle qui peut être représentée sous la forme d'un arbre
π : projection, σ : restriction, $\bowtie$: jointure



Réécriture de la requête en langage algébrique

Plusieurs arbres peuvent être équivalents.
On peut obtenir, à partir d'un arbre, un arbre équivalent en utilisant des règles

Exemples de règles

- Commutativité des jointures : $A \bowtie B = B \bowtie A$
- Associativité des jointures $(A \bowtie B) \bowtie C = A \bowtie (B \bowtie C)$

Règles de restructuration algébrique

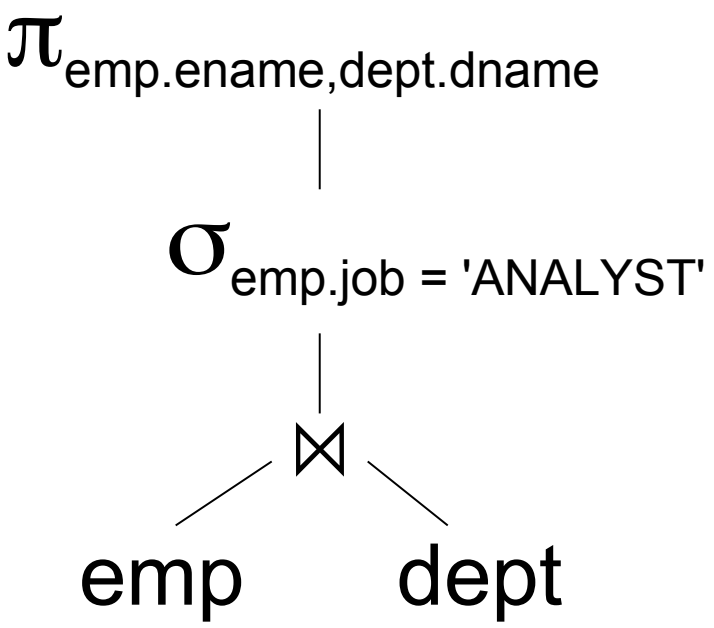
- Effectuer les restrictions le plus tôt possible (les descendre le plus bas possible dans l'arbre)
- Regrouper les restrictions successives portant sur une même relation

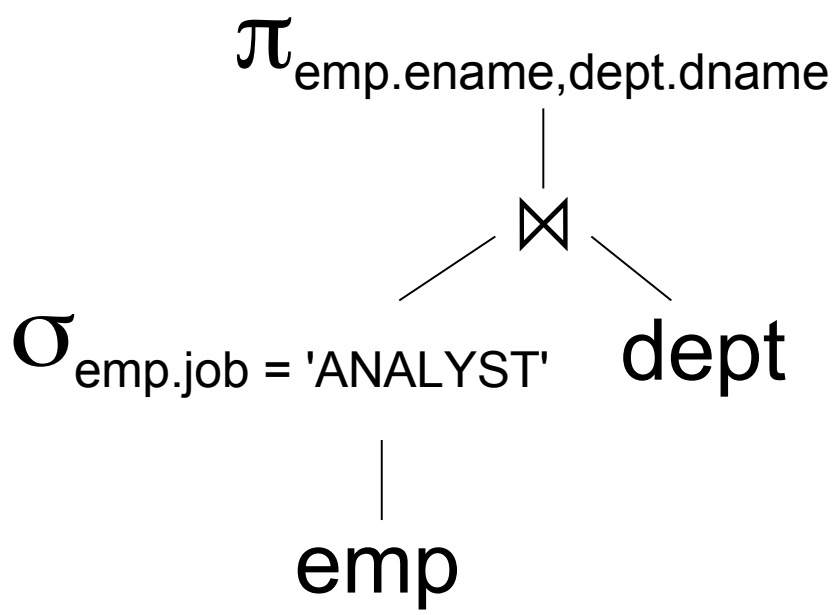
  → la restriction est effectuée au plus tôt car c'est l'opérateur le plus réducteur
- Descendre les projections le plus bas possible dans l'arbre
- Regrouper les projections portant sur une même relation

  → on réalise les projections dès que possible pour éliminer des attributs

  → les jointures sont réalisées une fois la taille des données réduite

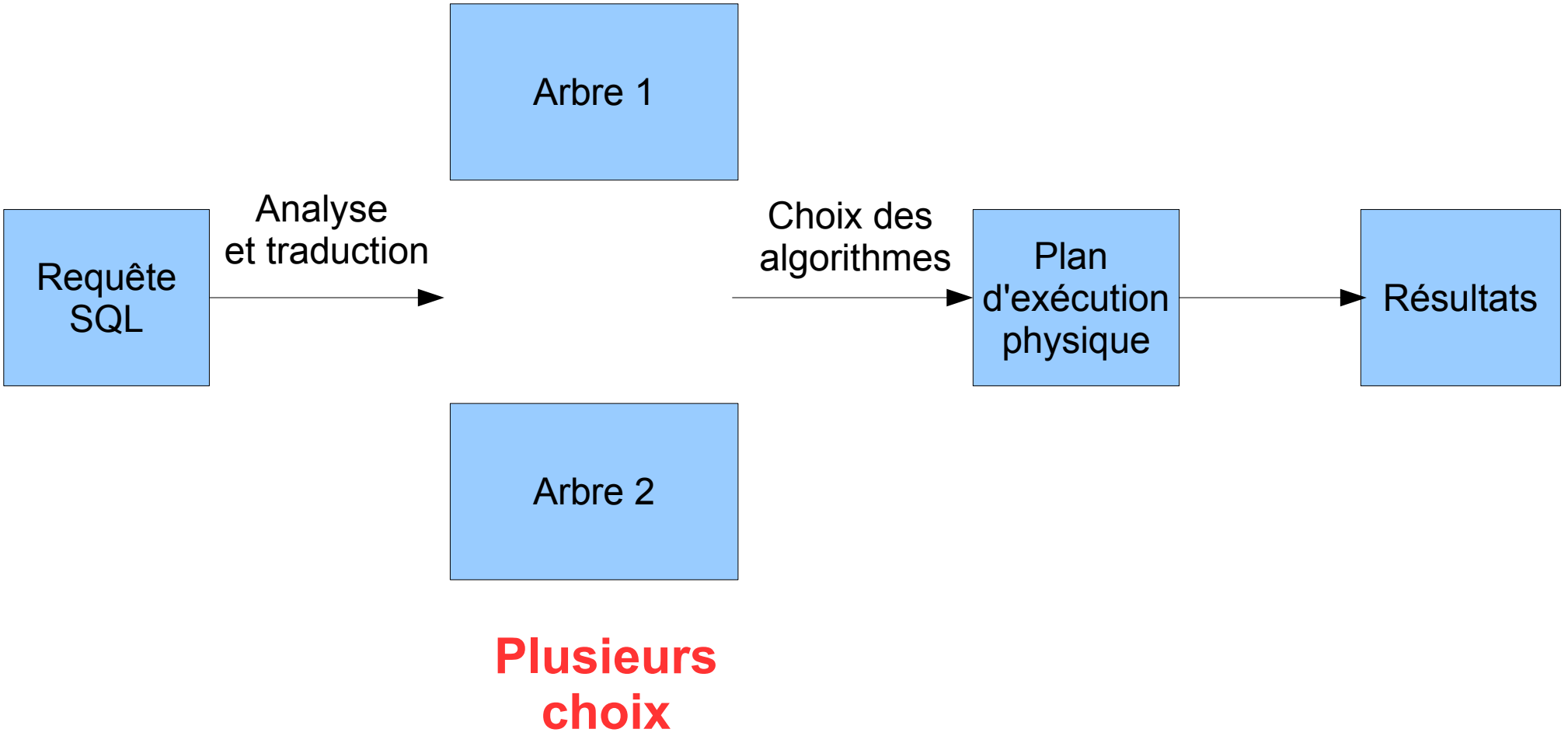
Exemple d'arbres algébriques équivalents





Plusieurs restructurations sont possibles

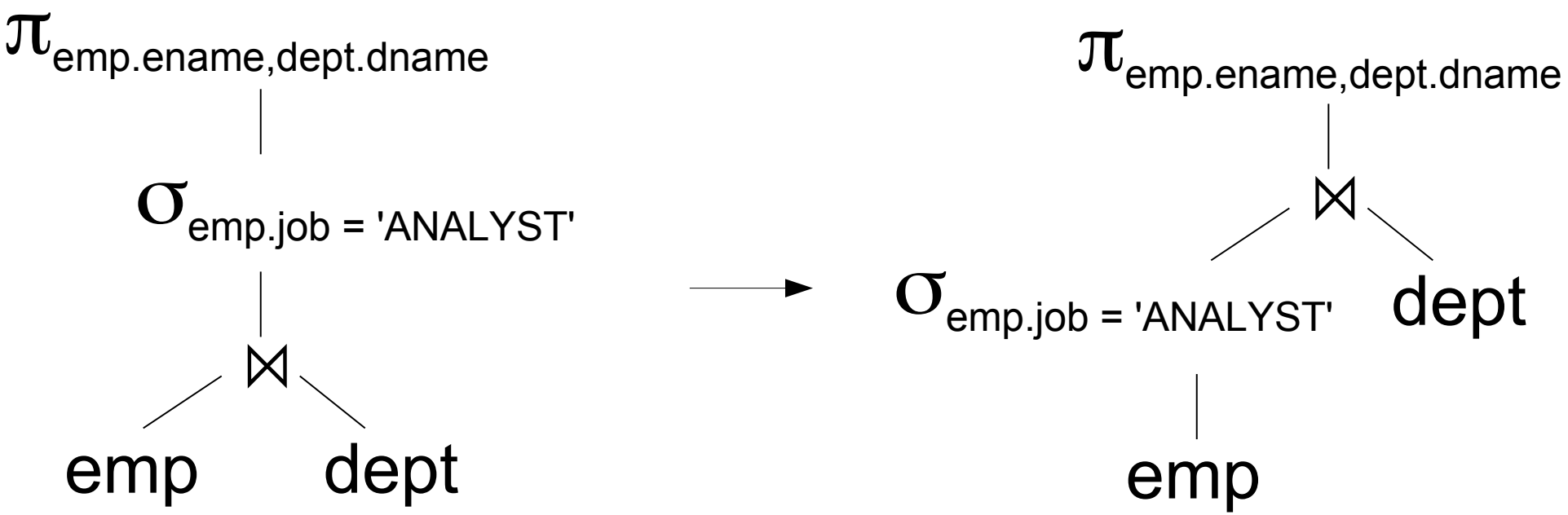
Exécution d'une requête



Exécution d'une requête

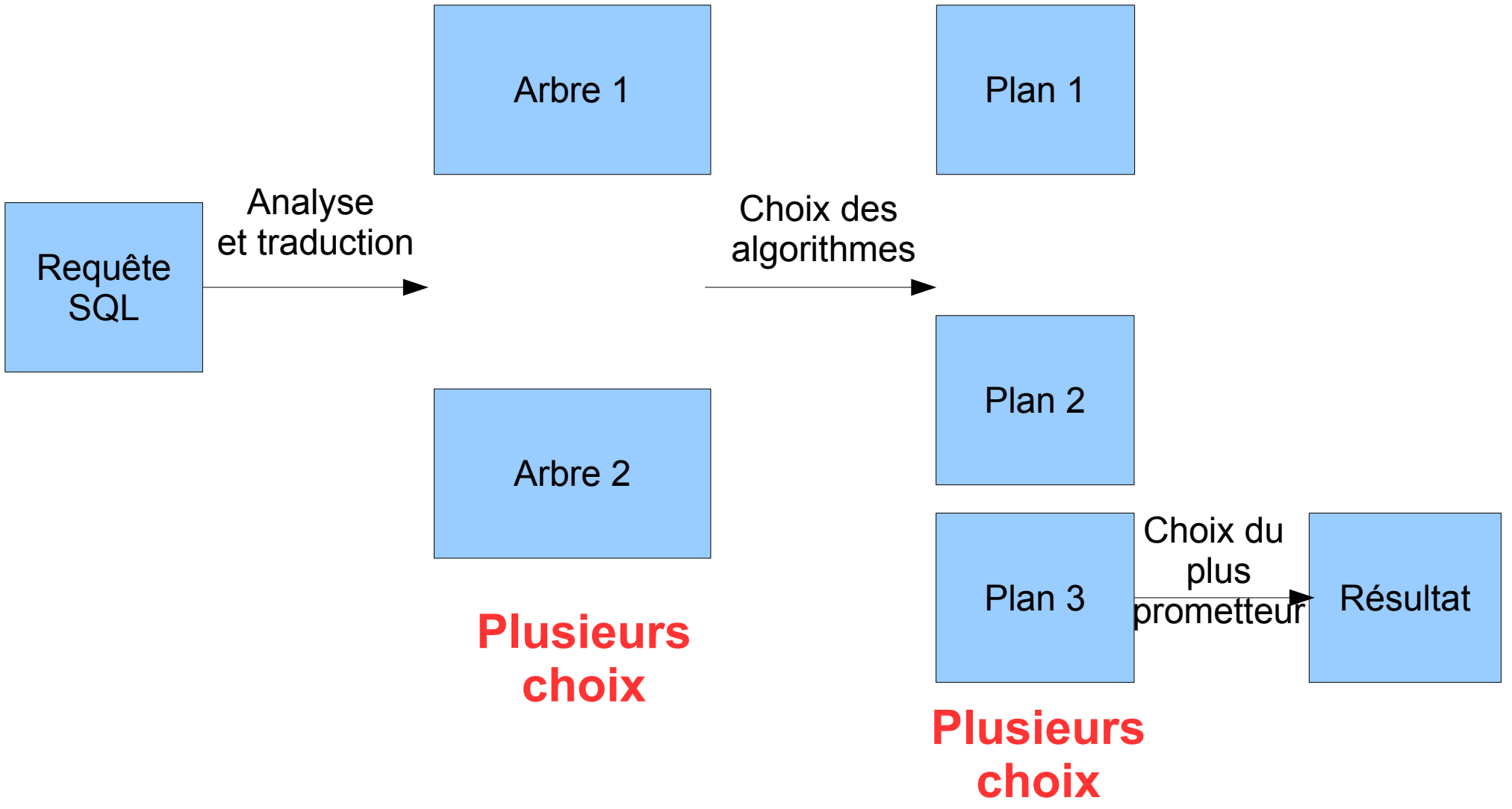
Pour ne pas énumérer tous les plans d'exécution : heuristiques
→ exemple : réduire la taille des données (filtrer par des sélections, puis des projections dès que possible)

Exemple d'arbres algébriques équivalents



La sélection est ramenée avant la jointure, l'opérateur de jointure sera plus efficace en manipulant moins de données

Exécution d'une requête



Avertissement

Les informations concernant le plan d'exécution et l'optimiseur sont basées sur Oracle, il peut y avoir des différences dans d'autres SGBD

Exemple 1

Exemple de plan d'exécution physique

```
-----------------------------------------------
| Id  | Operation                     | Name    |
-----------------------------------------------
|   0 | SELECT STATEMENT              |         |
|*  1 |  HASH JOIN                    |         |
|*  2 |   TABLE ACCESS FULL           | BIGEMP  |
|   3 |   TABLE ACCESS FULL           | BIGDEPT |
-----------------------------------------------
```

Statistiques

- Permettent de prédire si l'utilisation d'un index est intéressante
- Utilisées par l'optimiseur
- Stockées dans le dictionnaire de données (USER_TAB_COLUMNS, USER_TABLES, USER_INDEXES, …)
- Concernent
  - le système (utilisation CPU, ...),
  - les tables (volumétrie, taille moyenne des lignes, ...),
  - les colonnes (nombre de valeurs distinctes, nombre de NULL, …)
  - les données de la table (valeurs min et max, ...)

L'optimiseur utilise l'index

- Quand le parcours seul de l'index suffit à obtenir la réponse
- Quand la condition de la requête a une forte sélectivité (importance d'avoir des statistiques fidèles aux données)

Evaluer la situation

On ne peut améliorer que ce que l'on mesure !
(William Deming)

Principaux critères d'évaluation

- Le temps de réponse (varie avec le nombre d'utilisateurs sur la base)

Consommation des ressources :

- La consommation mémoire (varie avec le nombre d'utilisateurs sur la base)
- La consommation CPU (varie avec le nombre d'utilisateurs sur la base)
- Le nombre d'entrées/sorties disque (stable)

Sous SQL*Plus

- Mesurer le temps d'exécution :

  ```
  set timing on;
  ```

- Afficher les statistiques et le plan d'exécution d'une requête :

  activation du mode autotrace

  `set autotrace on;` (plutôt `traceonly`)

  Ou

  ```
  explain plan for (requete);
  select plan_table_output from
  table(dbms_xplan.display());
  ```

AUTOTRACE

```
SQL> set autotrace traceonly stat;

SQL> select * from bigemp where empno>2000000;

520098 ligne(s) sélectionnée(s).

Statistiques

----------------------------------------------------------
          18  recursive calls
           0  db block gets
       37865  consistent gets
           0  physical reads
           0  redo size
    26916354  bytes sent via SQL*Net to client
      381767  bytes received via SQL*Net from client
       34675  SQL*Net roundtrips to/from client
           0  sorts (memory)
           0  sorts (disk)
      520098  rows processed

SQL>
```

Plan d'exécution

- Liste des opérations établie par le SGBDR pour répondre à une requête de la manière estimée la plus efficace (un arbre algébrique)
- Estimation basée sur des statistiques
- Réduire les physical reads et les sorts (disks)
- Possibilité d'imposer un autre chemin : les hints (à utiliser avec beaucoup de précautions !)

Optimiseur Oracle

- CBO : Cost Based Optimizer
- Basé sur les coûts

Choix à faire pour l'optimiseur

- Opérations d'accès aux tables : comment seront récupérées les lignes d'une table
- Mécanismes de jointure
- Ordre des jointures

Objectif de l'optimiseur

Différents plans possibles : réduire le nombre de lignes au plus tôt

- Est-ce qu'une ligne source donne au plus une ligne ? (basé sur les contraintes de clé primaire et d'unicité des tables)
- Pour une jointure externe (OUTER JOIN), en général l'optimiseur place la table dominée en second dans la condition de jointure

Opérations d'accès aux tables

Plusieurs façons d'accéder à l'information :

- Full Table Scan : toutes les lignes de la table sont lues en mémoires → coûteux pour une grosse table car nombreux accès disque
- Index Scan/Table Access by Rowid : utilisation de l'index pour décider quelles lignes sont lues en mémoire, puis ces lignes sont récupérées en utilisant le Rowid de l'index
- Index Scan : accès à l'index uniquement

Opérations d'accès aux index

- Unique scan : parcourt l'arborescence de l'index pour localiser une clé unique
- Range scan : parcours un index de façon ordonnée. Généralement déclenché par les clauses d'égalité, supériorité ou infériorité
- Full scan : parcourt un index en entier en respectant l'ordre de l'index
- Fast full scan : analogue à full scan mais ne respecte pas l'ordre des données
- Unique index : s'applique lorsque l'index repose sur une colonne possédant une contrainte d'unicité
- Bitmap : utilise un index de type bitmap

Des opérations de jointure

- jointure par boucles imbriquées (NESTED LOOPS)
- jointure par hachage (HASH JOIN)
- jointure par tri-fusion (SORT-MERGE)

NESTED LOOPS

Jointure par boucles imbriquées : énumération de toutes les paires de n-uplets entre les deux tables et test de celles qui sont à apparier

→ performant lorsqu'un faible nombre de lignes d'une table est mis en jointure et que la condition de jointure accède facilement à la 2e table

NESTED LOOPS

Optimiseur : Désignation de la table externe (dominante) et de la table interne

Fonctionnement (comme des boucles imbriquées) :
1.Récupérer une ligne de la source externe
2. Examiner les lignes de la source interne pour trouver celles qui vérifient la condition
3.Répéter jusqu'à ce qu'il n'y ait plus de lignes à récupérer de la source externe

Plan d'exécution (2 `NESTED LOOPS` correspondent à une optimisation de l'implémentation)

```
NESTED LOOPS
outer_loop
inner_loop
```

HASH JOIN

Jointure par hachage : optimisation de la jointure par boucles imbriquées. Utilisation d'une table de hachage pour une des deux tables (t2) qui permet de trouver très vite pour un n-uplet de t1 les tuples à apparier dans t2

Utilisée pour des jeux de données relativement larges et pour une équi-jointure

Utilisation du plus petit jeu de données pour construire la table de hachage (plus efficace quand la table de hachage tient en mémoire)

HASH JOIN

Variantes :

- les deux tables tiennent en mémoire
- (plus courant) une table en mémoire dans une table de hachage et l'autre défile dans un bloc en mémoire
- phase de préparation par hachage ou tri qui introduit une latence

Exemple de plan d'exécution

```
SELECT e.ename, e.sal, d.dname
  2  FROM   bigemp e JOIN bigdept d on e.deptno=d.deptno;

1400014 lignes selectionnees.


Plan d'execution
----------------------------------------------------------
Plan hash value: 1288316262

-------------------------------------------------------------------------------------
| Id  | Operation           | Name    | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |         |  1400K|    42M|       |  5553   (1)| 00:00:01 |
|*  1 |  HASH JOIN          |         |  1400K|    42M|    10M|  5553   (1)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| BIGDEPT |   400K|  6250K|       |   445   (1)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| BIGEMP  |  1400K|    21M|       |  2715   (1)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   1 - access("E"."DEPTNO"="D"."DEPTNO")
```

SORT-MERGE

Jointure par tri-fusion (SORT-MERGE) : les deux tables doivent être triées sur la colonne de jointure, puis fusion des deux

Choisi plutôt qu'une jointure pour de grands ensembles de données quand

- il ne s'agit pas d'une équi-jointure
- parce qu'un tri est requis par d'autres opérations

Optimisation des jointures

- Algorithmes de jointure traitent 2 tables à la fois
- Si plus de 2 tables à joindre, les jointures sont réalisées séquentiellement : la jointure entre deux tables donne un résultat intermédiaire joint avec la table suivante
- Le choix de l'ordre des jointures affecte les performances → choix de l'optimiseur parmi toutes les permutations possibles
- Le choix de l'algorithme dépend de la mémoire disponible, de la taille des relations

Optimisation des jointures

L'optimiseur doit estimer les coûts des jointures en fonction de l'opération de jointure et de l'ordre des tables

Join Method	Cost of date_dim, lineorder	Cost of lineorder, date_dim
Nested Loops	39,480	6,187,540
Hash Join	187,528	194,909
Sort Merge	217,129	217,129

https://docs.oracle.com/en/database/oracle/oracle-database/21/tgsql/joins.html#GUID-BD96F1B4-76D4-43DF-98B6-D07F46838C4A

Recommandations pour l'écriture de requêtes

- Ne garder que les lignes utiles
- Ne garder que les colonnes utiles (peut permettre d'utiliser un index uniquement et d'éviter l'accès à une table)

Exemple plan d'exécution 1

Nouvelle table bigemp : 548000 lignes
empno le plus élevé : 39190902

```
select ename, job from bigemp where empno > 200000;

545298 ligne(s) sélectionnée(s).


Plan d'exécution

----------------------------------------------------------------------------
| Id  | Operation          | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |        |   545K|    10M|  1027   (1)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| BIGEMP |   545K|    10M|  1027   (1)| 00:00:01 |
----------------------------------------------------------------------------

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

   1 - filter("EMPNO">200000)
```

Exemple plan d'exécution 1

```
Statistiques
----------------------------------------------------------

          8  recursive calls
          0  db block gets
      39544  consistent gets
          0  physical reads
          0  redo size
   12288395  bytes sent via SQL*Net to client
     400247  bytes received via SQL*Net from client
      36355  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
     545298  rows processed
```

Exemple plan d'exécution

```
select ename, job from bigemp where empno > 35000000;

Plan d'exécution

-------------------------------------------------------------------------------
|Id|Operation                          |Name     |Rows |Bytes| Cost (%CPU)|
-------------------------------------------------------------------------------
| 0|SELECT STATEMENT                   |         |58612|1144K|     808 (1)|
| 1| TABLE ACCESS BY INDEX ROWID BATCHED|BIGEMP   |58612|1144K|     808 (1)|
|*2|  INDEX RANGE SCAN                 |PK_BIGEMP|58612|     |     213 (0)|
-------------------------------------------------------------------------------

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

   2 - access("EMPNO">35000000)
```