Administration des SGBD

Optimisation

M1 Informatique

Damien Ploix

Performance ?

- De quelles performance parle-t'on ?
- Quels sont les éléments actifs et quels sont leurs rôles respectifs ?
- Quels sont les mesures de performance significatives sur ces éléments ?
- Quels sont les optimisations possibles, comment les mettre en œuvre ?

- La notion de performance d'un système indique sa capacité à répondre dans les délais prévus aux questions prévues dans le contexte prévu → la question essentielles est :

« qu'est-ce qui est prévu ? »…

Performance ?

Questions principales :

- Combien d'utilisateurs sont prévus ?
- Comment les utilisateurs interagiront-ils ?
- Où sont positionnés les systèmes ?
- Quelle est la vitesse du réseau ?
- Quelle quantité de données seront accédées par les utilisateurs et quelle proportion l'est en principalement en lecture seule ?
- Quel est le temps de réponse exigé par les utilisateurs ?
- Le système doit-il être disponible 24/24 ?
- Tous les changements doivent-ils être répercutés en temps réel ?

Questions plus liées à un impact budgétaire et de complexité que de tunning :

- Quelle sera la taille de la base de données ?
- Quelle est le format de sortie des transactions ?
- Quelles sont les conditions de disponibilité ?
- Les qualifications (RH) sont-elles disponibles pour construire et administrer et optimiser cette application ?
- Quels sorte de compromis sera édicté par des contraintes budgétaires ?

Quoi regarder ?

1) Code applicatif :
   - Design de la base de données
   - Mise en place d'indexes
   - Codage de l'application
     - Efficience du code
     - Bonne ordonnancement des taches
2) l'instance de la base :
   - Réglage des paramètres de mémoire et de buffers
   - Élimination des blocages
   - Surveillance des IO sur les données
3) l'infrastructure sous jacente
   - Système d'exploitation
   - Ressources matérielles (contention des IO, réseau)

Optimisation du modèle

- Les modélisations sont adapté aux algorithmes implémentés dans les moteurs de SGBD
  - Entité / Relation :
    - adapté au calcul ensembliste *transactionnel*
      - Limitation des redondances,
      - Utilisation de jointures,
      - Cohérence (intégrité) fonctionnelle
- Représente 95 % des bases de données en place

Schéma de base de données optimal ?

Optimisation par la dénormalisation

- Non si
  - Le système peut fonctionner correctement sans dénormalisation, ou
  - Les performances du système ne sont pas plus acceptables après la dénormalisation, ou
  - Le système sera moins fiable après dénormalisation.

- Précautions nécessaires :
  - Avoir une forme dénormalisée mise à jours à partir d'une forme normalisée,
  - Automatiser la mise à jour des données dénormalisées via des triggers, des scripts, des ETL,
  - Automatiser via des scripts, programmes, contraintes ou des triggers afin d'assurer le maintient de la cohérence relative à la dénormalisation.

- Avantages :
  - Optimisation du temps de calcul

Dénormalisation : tables préjointes

- Condition :
  - Ne contient pas de colonnes redondantes,
  - Doit uniquement contenir les colonnes nécessaires au calcul,
  - Doit être créées périodiquement via une requête SQL sur les tables normalisées (possible de l'émuler via du code si le SGBD ne les proposent pas en natif).
- Avantage : pré-calcul des jointures
- Désavantage : maintient de la synchronisation
- Cette dénormalisation existe via les « vues matérialisées » (Oracle, PostgerSQL) ou « vues indexées » (SQL Server) dont l'objet est de stocker le résultat de la requête définissant la vue dans une table interne.
- La définition de la vue matérialisée inclus alors les conditions de mise à jours de la table.

Dénormalisation : tables de reporting

- Condition :
  - Une colonne par colonne du rapport
  - Séquencement physique propre
  - Ne pas recomposer des champs (1FN)
- Permet le pré-calcul (batch) des rapports
- Utilisé dans les outils/projets analytiques (par exemple, reportServer de la suite Microsoft)
  - Via les mats views...

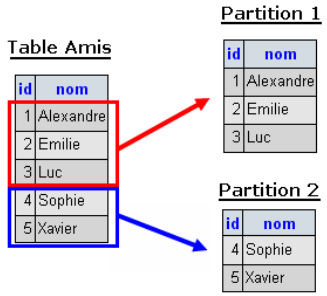
Vue matérialisé : exemple

```
CREATE MATERIALIZED VIEW cust_mth_sales_mv
BUILD IMMEDIATE
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE AS
SELECT s.time_id, s.prod_id, SUM(s.quantity_sold), SUM(s.amount_sold),
      p.prod_name, t.calendar_month_name, COUNT(*),
      COUNT(s.quantity_sold), COUNT(s.amount_sold)
FROM sales s, products p, times t
WHERE  s.time_id = t.time_id AND s.prod_id = p.prod_id
GROUP BY t.calendar_month_name, s.prod_id, p.prod_name, s.time_id;

BEGIN
  DBMS_REFRESH.make(
    name               => 'SH.MINUTE_REFRESH',
    list               => '',
    next_date          => SYSDATE,
    interval           => '/*1:Mins*/ SYSDATE + 1/(60*24)',
    implicit_destroy   => FALSE,
    lax                => FALSE, (permet chg group)
    job                => 0,
    rollback_seg       => NULL,
    push_deferred_rpc  => TRUE,
    refresh_after_errors => TRUE,
    purge_option       => NULL,
    parallelism        => NULL,
    heap_size          => NULL);
END;
/
```

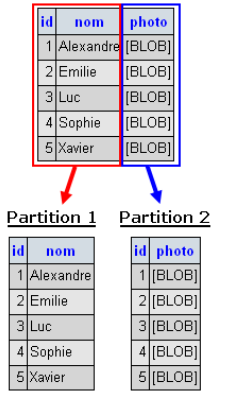
```
BEGIN
  DBMS_REFRESH.add(
    name => 'SCOTT.MINUTE_REFRESH',
    list => 'SCOTT.EMP_MV',
    lax  => TRUE);
END;
/

EXEC DBMS_MVIEW.refresh('EMP_MV');
```

Dénormalisation : tables miroirs / scindées / groupées

- Scission Horizontale : split de la table en tranches de données…
  - Mise en place du « partitionnement » des tables. Très avantageux pour des grandes tables puisque cela équivaut à construire N tables contenant chacune une partie des données ➔ gestion distinctes des indexes, réduction des collisions, …
  - Difficultés :
    - trouver la clé de partitionnement la plus pertinente par sa séparation en partitions de taille aussi égales que possibles.
    - Deviens très complexe à mettre en œuvre dans le cas de modèles composés d'un grand nombre de tables.
  - La plus part des SGBD offrent la fonctionnalité. Le partitionnement est alors soit
    - Par rapport à une plage (range partitionning) : définition des bornes des différentes partitions (ensemble des données par années, …)
    - Par liste : applicable à des attributs à faible cardinalité (répartition de la population par taille, …)
    - Par clé de hash :
      - » Soit via la définition d'une fonction de calcul de clé
      - » Soit via l'utilisation d'un calcul de clé de hash laissé au SGBD



- Scission Verticale :
  - Des colonnes sont placées dans N tables qui partagent la clé primaire…
  - Des colonnes sont scindées en plusieurs colonnes afin d'éviter les champs de grande taille (qui souvent mal gérés physiquement par les DBMS).
  - Utilisation de vues pour redonner la vision externe d'une seule table.
  - Oracle propose le mécanisme de « cluster » de table permettant à des tables qui partagent une même clé primaire et sont souvent consultées en mêmes temps d'être stockée dans les mêmes datablock ce qui en accélèrent les IO.
  - Cas d'usage : gestion des BLOB par oracle ==> mieux vaux avoir 2 x 4000 au lieu de 8000...

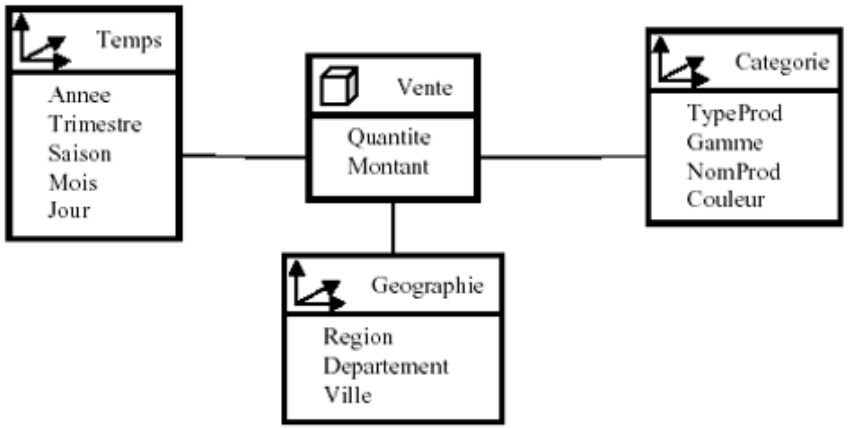


Dénormalisation :
données redondantes et groupes répétés

- La redondance des données si elle permet d'éviter des jointures systématiques et si
  - Peut de colonnes sont nécessaires,
  - Les colonnes sont stables rarement modifiées,
  - Utilisées par beaucoup d'utilisateurs ou par peut d'utilisateurs critiques.

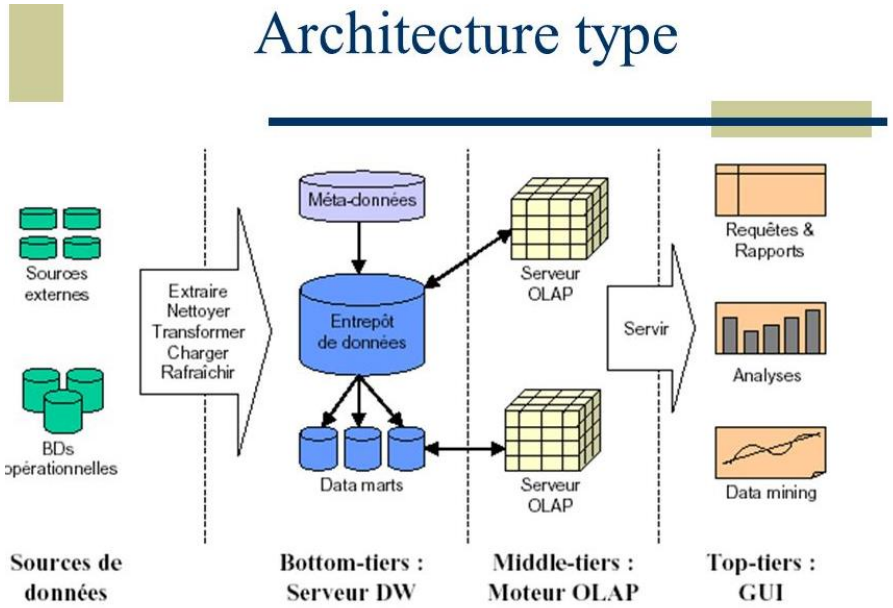
- Les groupes répétés (supprime une table d'association) si :
  - Les données sont rarement voir jamais agrégées,
  - Les données se présentent statistiquement selon un pattern correct
  - Le nombre d'occurrence est stable,
  - Les données sont accédées collectivement,
  - Le pattern d'insertion ou d'effacement est prédictible.

- Cas d'usage :
  - Dimensions d'une base BI



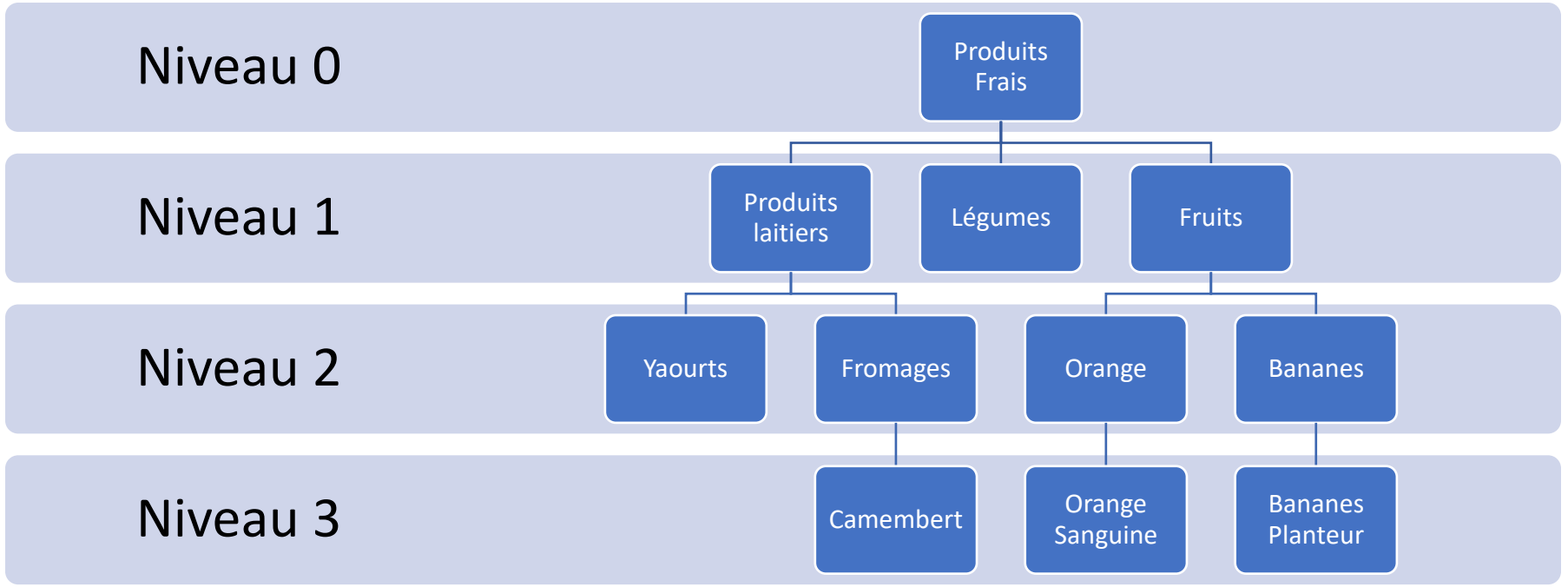
Dénormalisation : stockage des données calculées/dérivées

- Pré-calcul à condition que :
  - Les données sources soient relativement stables,
  - Le coût de calcul des dérivation est élevé,
  - L'utilisation des tables sources permet un recalcul rapide en cas de modification.
- Cas d'usage :
  - Cubes des bases BI



Dénormalisation : parcours de hiérarchie

- *Par l'exemple !*



Dénormalisation : parcours de hiérarchie

Table Hiérarchique

ID	IDPARENT	DESCRIPTION
1	null	Produits frais
2	1	Produits laitiers
3	1	Légumes
4	1	Fuits
5	2	Yaourts
6	2	Fromages
7	6	Camembert
8	4	Orange
9	8	Orange sanguines
10	4	Bananes
11	10	Bananes Planteur

Flat/Speed Table

ID	ID_N1	ID_N2	ID_N3	DESCRIPTION
1	null	null	null	Produits Frais
2	1	null	null	Produits laitiers
3	1	null	null	Légumes
4	1	null	null	Fuits
5	1	2	null	Yaourts
6	1	2	null	Fromages
7	1	2	6	Camembert
8	1	4	null	Orange
9	1	4	8	Orange sanguines
10	1	4	null	Bananes
11	1	4	10	Bananes Planteur

Closure Table

ID	DESCRIPTION
1	Produits frais
2	Produits laitiers
3	Légumes
4	Fuits
5	Yaourts
6	Fromages
7	Camembert
8	Orange
9	Orange sanguines
10	Bananes
11	Bananes Planteur

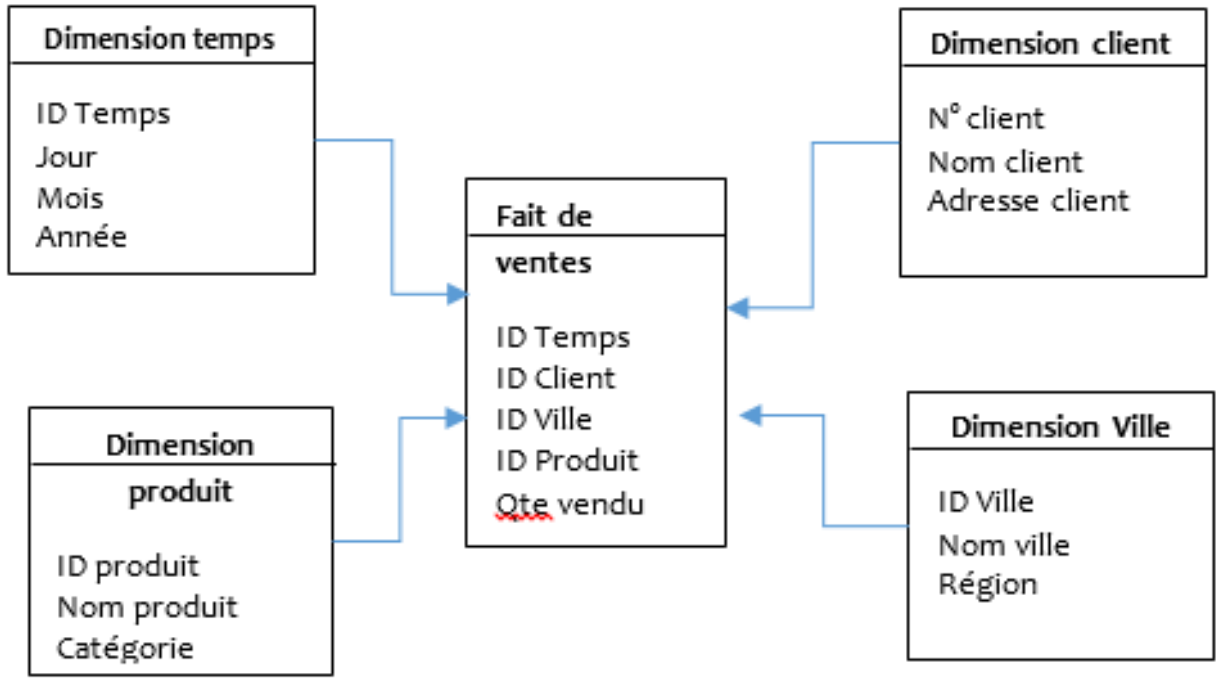
MAPID	IDPARENT	IDENFANT	DISTANCE
1	1	2	1
2	1	3	1
3	1	4	1
4	1	5	2
5	2	5	1
6	1	6	2
7	2	6	1
8	1	7	3
9	2	7	2
10	6	7	1
11	…		

Pour chaque table, quelle est la requête SQL permettant de récupérer l'ensemble des produits laitiers disponibles (niveau 3) ?

Quel est l'impact de l'ajout d'un nouveau niveau ?

Analytique (BI) : Modèle dénormalisé

- Table de Faits et ses Dimensions



Dénormalisation : résumé

Dénormalisation	**A utiliser lorsque**
Tables préjointes	Le coût des jointures est prohibitif
Reporting	Des rapports spécifiques et critiques sont nécessaires
Miroir	Des données sont accédées simultanément par # env.
Scission	Des groupes distincts accèdent à des éléments distincts
Combinaison	Consolidation de relation 1-1 ou 1-N dans une table unique
Redondance	Réduction du nombre de jointure nécessaires
Répétition de groupes	Réduction du nombre d'I/O et (potentiellement) de l'espace de stockage nécessaire
Dérivation	Transforme le calcul dynamique en calcul batch
Speed table	Gain en efficacité de parcours des hiérarchies
Faits/Dimension	Tables préjointes + Speed Table + Redondance + ….

indexes

- Les indexes aide à trouver **des enregistrements** dans **une table** qui correspondent à un/des **critère**(s) **particulier**(s). Ils peuvent être basés sur plusieurs attributs de la table (exemple : nom + prénom).
  - ➔Les indexes qui ne correspondent pas à des critères de recherche (clause where) ne servent à rien !

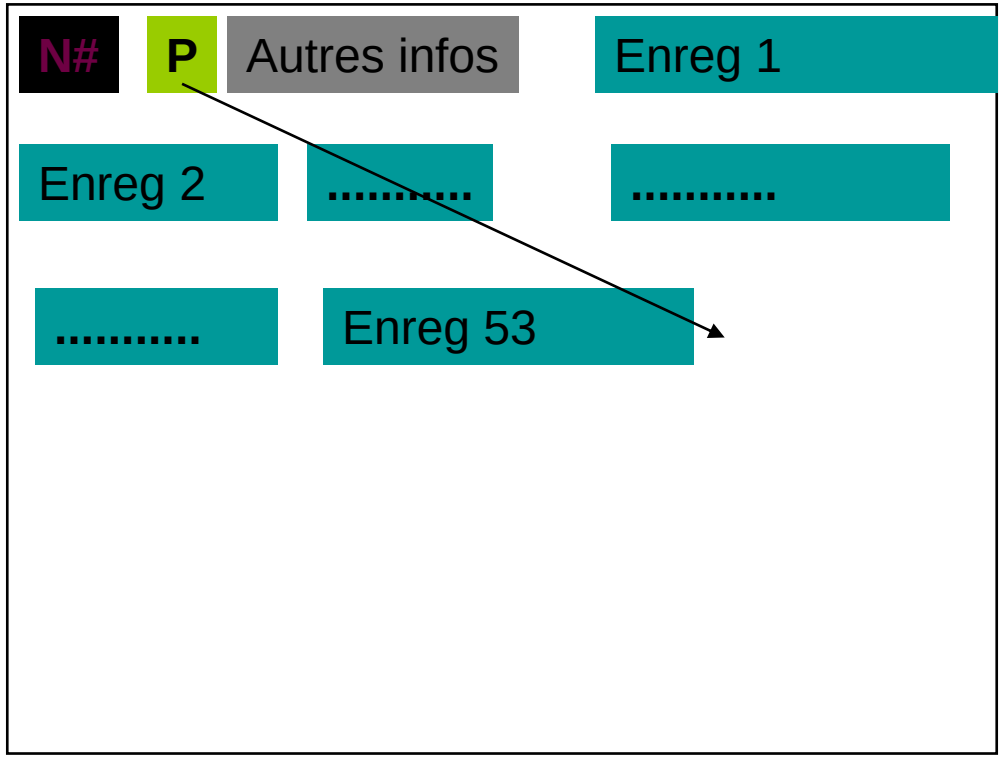
- Type (globaux) :
  - Clustered : définit la manière dont la table est physiquement ordonnée (typiquement, les PK),
  - Non-clustered : les données de l'indexe sont maintenues séparément de la table (un tablespace spécifique permet d'optimiser les accès concurrents aux espaces de stockage entre les données des tables et celles des indexes).

- Bons / mauvais indexes ?
  - Bon : valeurs qui changent peut/rarement, utilisés dans des clauses Where, permet de chercher des valeurs et/ou des plages de valeurs.
  - Mauvais : pas utilisés dans des clauses Where, mauvaise sélectivité (exception : l'indexation des clés étrangères optimise les jointures), beaucoup de colonnes dans l'indexe, peut d'enregistrements dans la table.

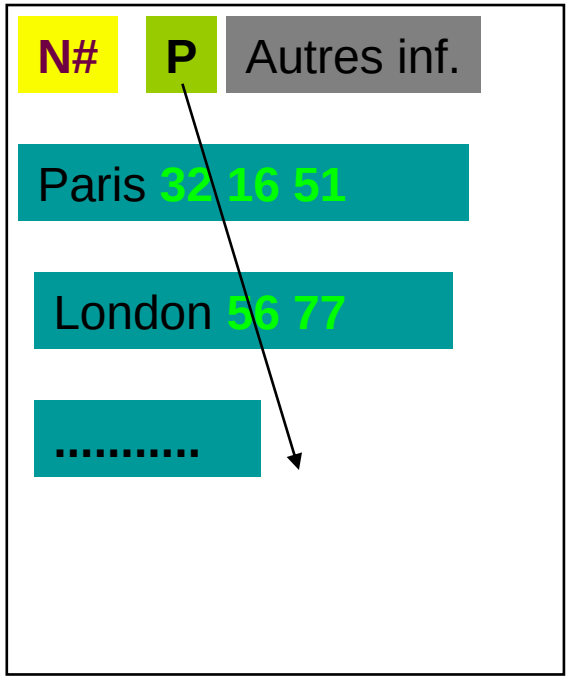
Rappels sur les indexes / Principes

- Les indexes sont des fichiers structurés qui contiennent les paires
  - attribut(s) d'accès -> numéro de page
- les indexes peuvent être
  - séquentiels ou directs ou non-ordonnés
  - hachés
  - arbres B
  - Binaires,
  - autres

Structure d'une page



Page d'index typique



Accélération de 10:1000 fois selon la capacité d'une page d'index

Fichiers ordonnés

- Arbre-B est le plus populaire (tout SGBD)
  - En fait l'arbre B^+ lié (linked B^+ - tree de Lehma & Yao)
    - Pour le parcours séquentiel et l'accès concurrent + efficace
      - Loquets individuels sur chaque page parcourue
- Une page contient *d*>>1 paires en ordre lexicographique de clés
  - clé - enreg.
  - ou clé-pointeur de page avec l'enreg.
- Une page qui devient pleine éclate en deux pages semi-pleines
- La clé d'éclatement est dupliquée dans la page de niveau supérieur avec les pointeurs vers les nouvelles pages

Arbres B (+)

Un Arbre B d'ordre m est un arbre tel que :

1/ Chaque nœud contient k clés triées avec :

m <= k <= 2m (nœuds non racine)

1 <= k <= 2m (nœuds racine)

2/ Chaque chemin de la racine à une feuille est de même longueur h (hauteur)

3/ Un nœud est soit :

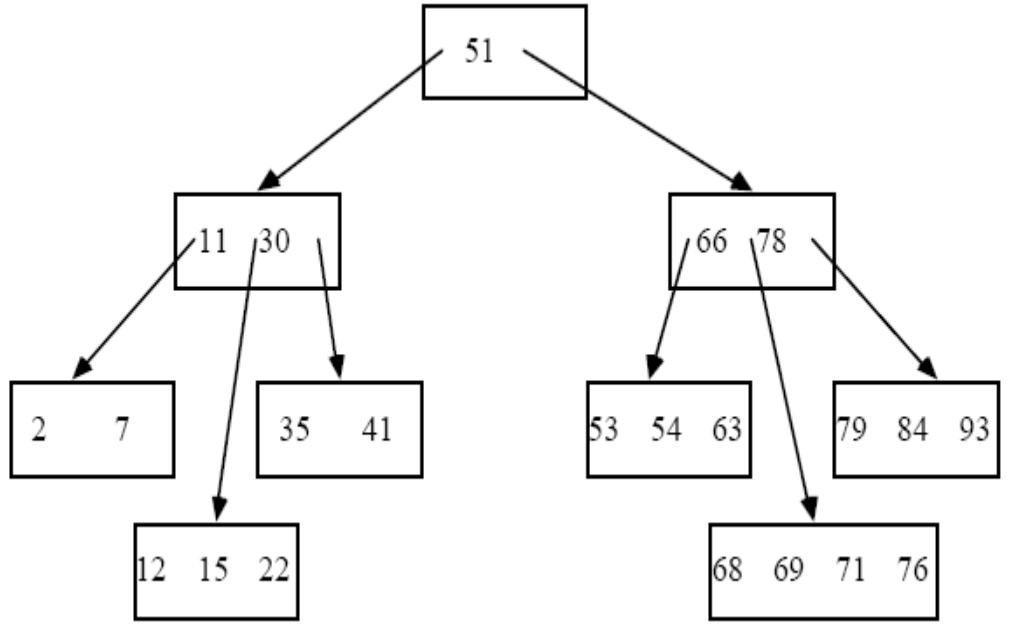
- Terminal (feuille)
- Possède k+1 fils
  - Les clés du ième fils ont des valeurs comprises entre les valeurs des (i-1)ème et ième clés du père.

4/ Un arbre B+ est un arbre B dont les feuilles contiennent toutes les clés.

Arbre B d'ordre 2

-> Chaque nœud, sauf la racine, contient **k** clés avec 2 <= **k** <= 4.

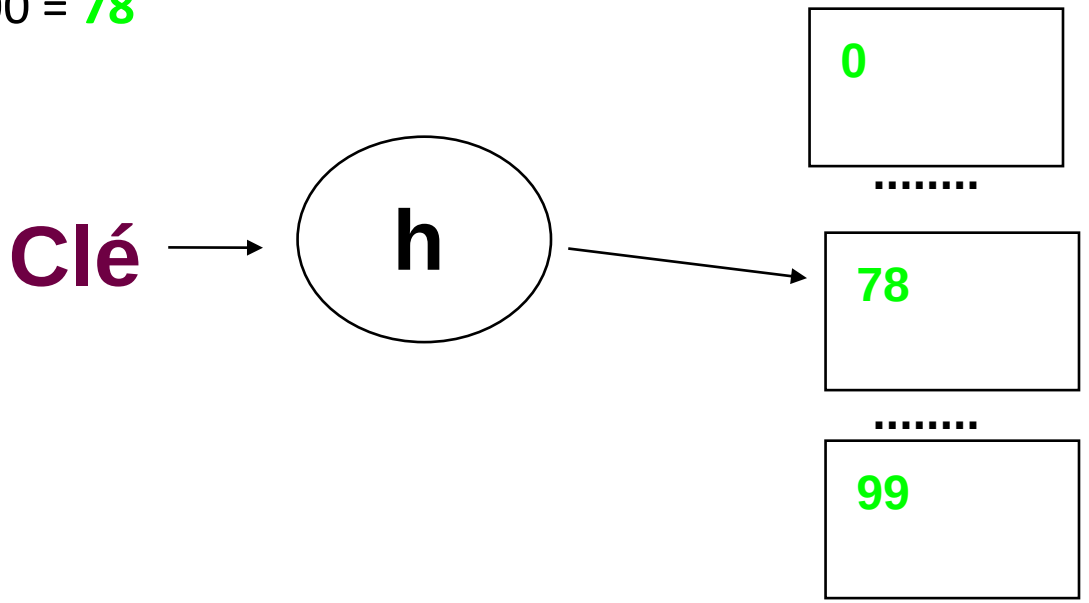
-> la racine contient **k** clés avec 1 <= **k** <= 4.



Accès haché

Le n# de page est obtenu par calcul à partir de la clé primaire de fichier

12 34 56 78 mod 100 = **78**



Adapté à des cardinalité moyenne et une bonne répartition afin de ne pas avoir de pages par code de hash trop importante.

Facilite l'accès à une donnée ponctuelle mais pas adaptée à la recherche de plage de données

Index BITMAP

À Chaque valeur possible pour le champ indexé une table de bit correspondant à la présence de la valeur dans l'enregistrement est généré.

À la table trois clés d'indexes seront générés :

Paris : 10001

Montreuil : 01000

Evry : 00110

ID	Ville
1	Paris
2	Montreuil
3	Evry
4	Evry
5	Paris

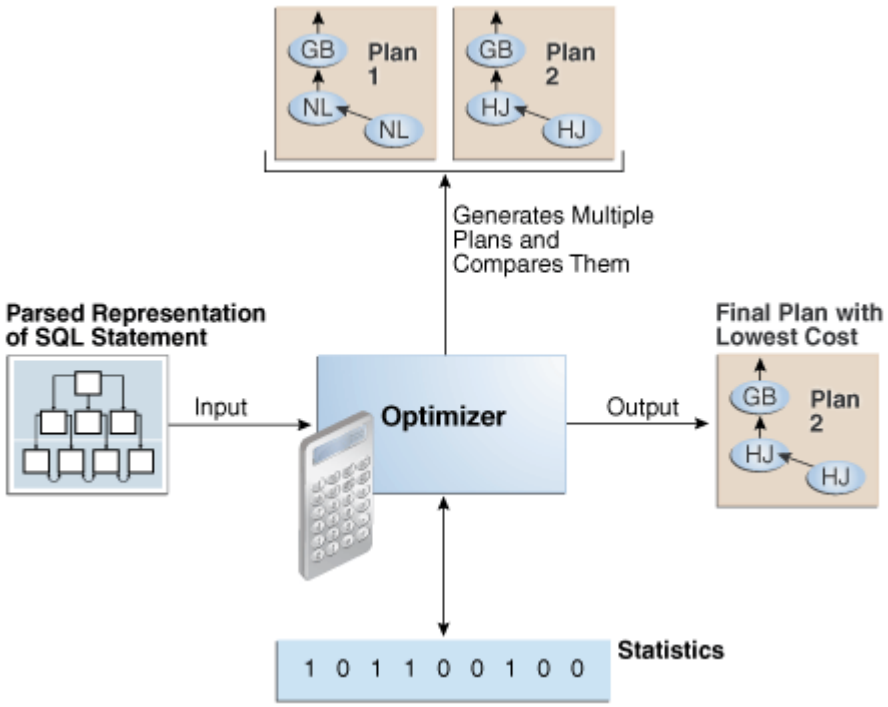
Donc facilité de calcul (or, and). Mais la mise à jours dans un contexte OLTP génère rapidement des points de blocages sur l'accès à la ressource modifiée.

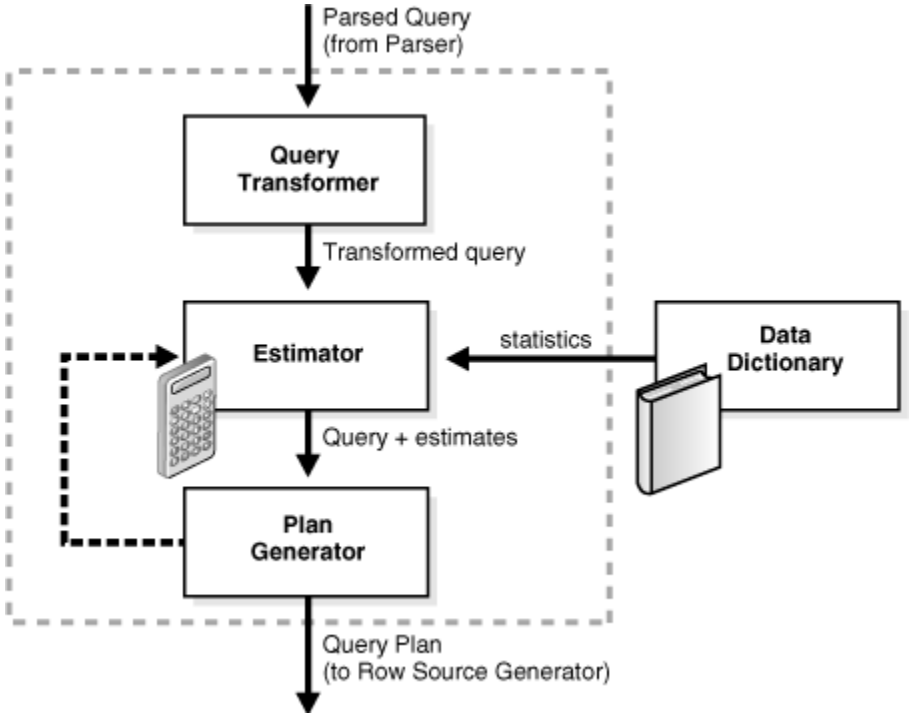
Donc, adapté à des valeurs à cardinalité réduite et à faible évolutivité ➔ OLAP.

Indexes : sélectivité

- Un index (hash, B+, …) doit avoir une sélectivité (« plage » de clés) de 15 à 20 % maximum pour être intéressant. Si elle est moindre (+ de 20%, alors il est trop coûteux).
- L'utilisation de ce type d'indexe est adapté dans le cas d'une forte cardinalité, d'un nombre important de valeurs et de requête de sélectivité réduite (si la requête ramène plus de 5% des données l'indexe ne sera pas systématiquement utilisé / approprié). Donc, particulièrement pour les traitements de type OLTP.

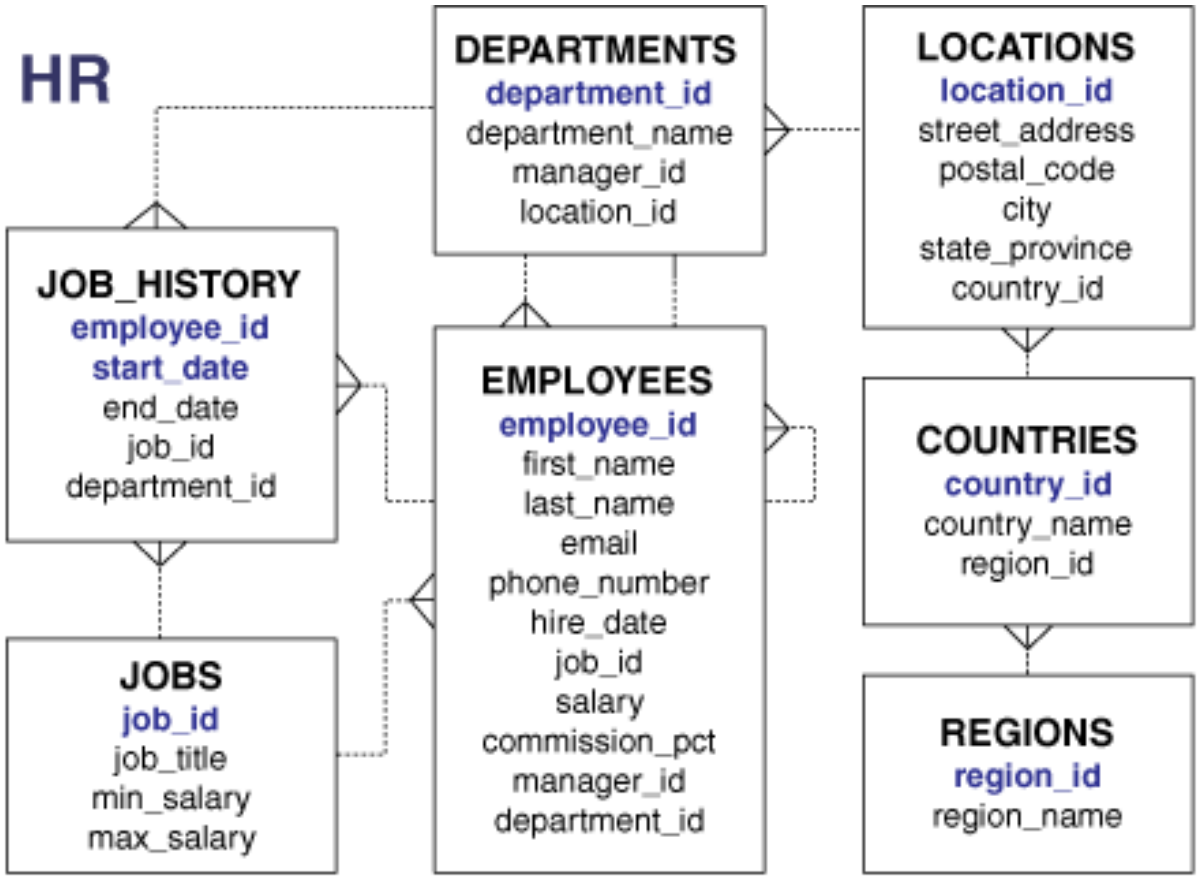
Analyse de la performance



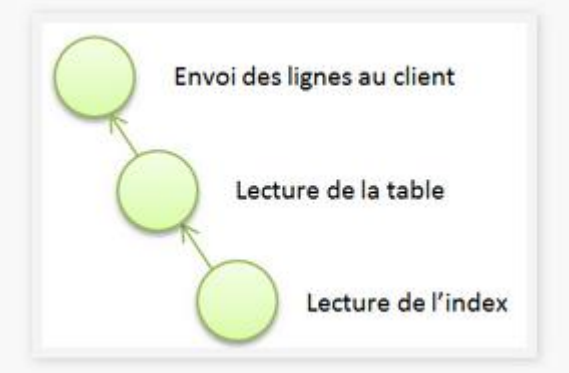


Plan d'exécutions (par l'exemple)

- Schéma HR des exemples inclus avec la base Oracle



Plan d'exécutions




```
select * from employees
where email like 'LDEHAAN'
```

Résultat de requête | Plan d'exécution

SQL | 0,057 secondes

OPERATION	OBJECT_NAME	OPTIONS	BYTES	CARDINALITY	COST
SELECT STATEMENT			69	1	1
TABLE ACCESS	EMPLOYEES	BY INDEX ROWID	69	1	1
INDEX	EMP_EMAIL_UK	UNIQUE SCAN		1	0
Access Predicates					
EMAIL='LDEHAAN'					

0,025 secondes — HR

Feuille de calcul | Query Builder

```
select e.first_name, e.last_name, d.department_name
from employees e JOIN departments d USING (department_id)
where email like 'LDEHAAN'
```

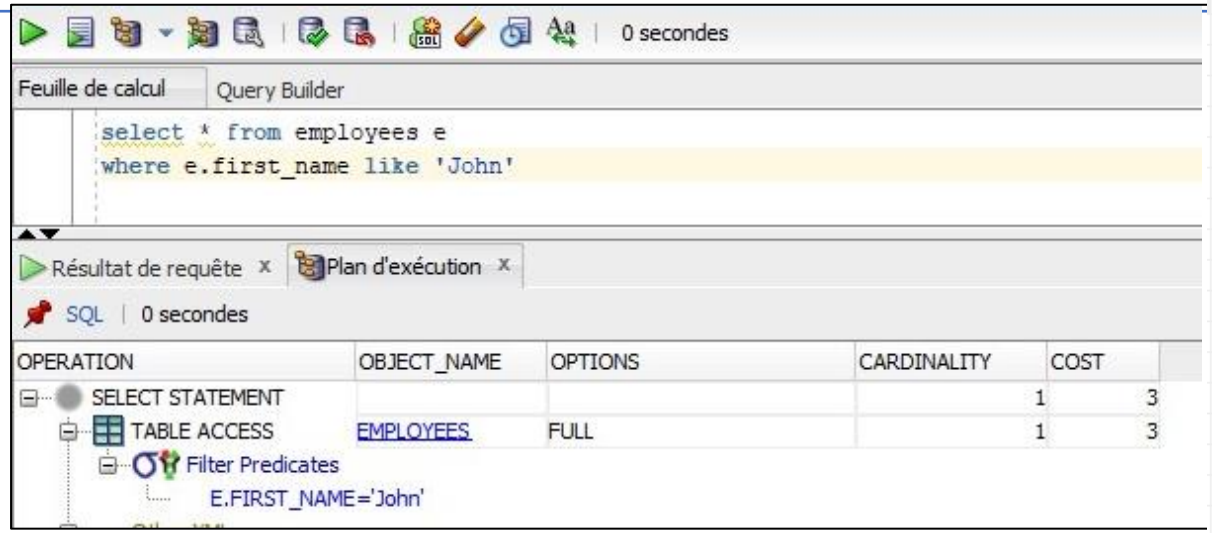
Résultat de requête | Plan d'exécution

SQL | 0,025 secondes

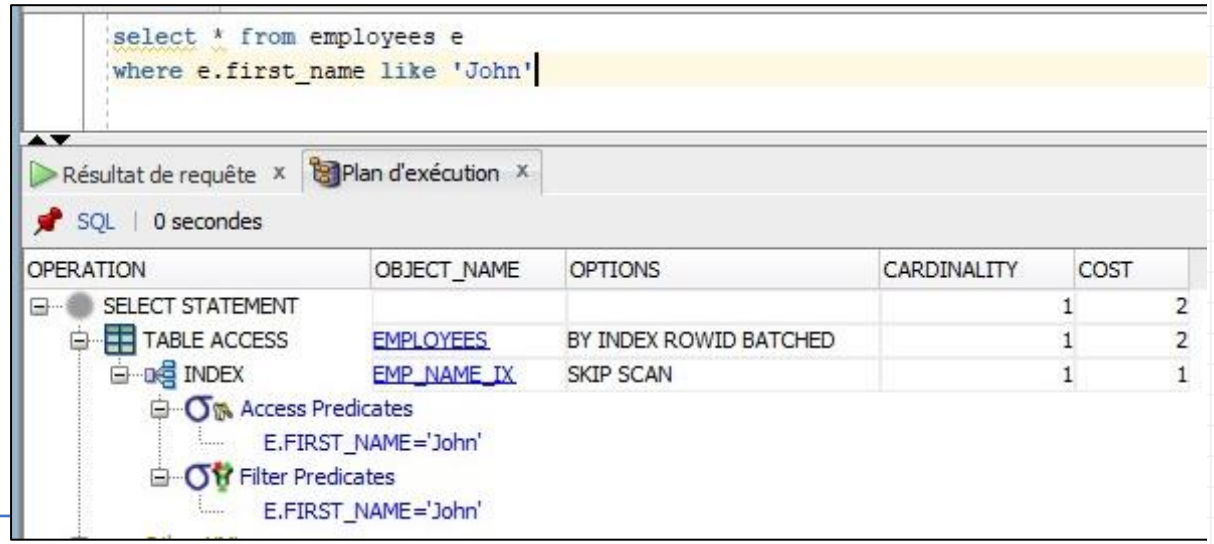
OPERATION	OBJECT_NAME	OPTIONS	BYTES	CARDINALITY	COST	CPU
SELECT STATEMENT			42	1	2	
NESTED LOOPS			42	1	2	
TABLE ACCESS	EMPLOYEES	BY INDEX ROWID	26	1	1	
INDEX	EMP_EMAIL_UK	UNIQUE SCAN		1	0	
Access Predicates						
E.EMAIL='LDEHAAN'						
TABLE ACCESS	DEPARTMENTS	BY INDEX ROWID	16	1	1	
INDEX	DEPT_ID_PK	UNIQUE SCAN		1	0	
Access Predicates						
E.DEPARTMENT_ID=D.DEPARTMENT_ID						

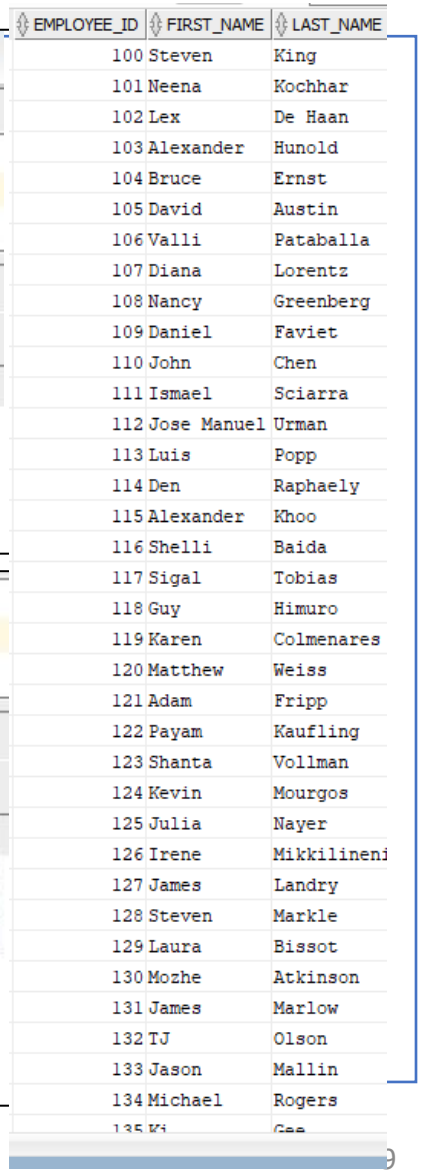
Plan d'exécutions (par l'exemple)

- Sans index



- Index positionné





EMPLOYEE_ID	FIRST_NAME	LAST_NAME
100	Steven	King
101	Neena	Kochhar
102	Lex	De Haan
103	Alexander	Hunold
104	Bruce	Ernst
105	David	Austin
106	Valli	Pataballa
107	Diana	Lorentz
108	Nancy	Greenberg
109	Daniel	Faviet
110	John	Chen
111	Ismael	Sciarra
112	Jose Manuel	Urman
113	Luis	Popp
114	Den	Raphaely
115	Alexander	Khoo
116	Shelli	Baida
117	Sigal	Tobias
118	Guy	Himuro
119	Karen	Colmenares
120	Matthew	Weiss
121	Adam	Fripp
122	Payam	Kaufling
123	Shanta	Vollman
124	Kevin	Mourgos
125	Julia	Nayer
126	Irene	Mikkilineni
127	James	Landry
128	Steven	Markle
129	Laura	Bissot
130	Mozhe	Atkinson
131	James	Marlow
132	TJ	Olson
133	Jason	Mallin
134	Michael	Rogers
135	Ki	Gee

Plan d'exécutions : Membres d'un département

- Aucun index : jointure des deux tables via l'algorithme « Hash »

Feuille de calcul | Query Builder

```
select e.first_name, e.last_name
from employees e, departments d
where d.department_name = 'Shipping' and (e.department_id+1-1) = d.department_id
```

Résultat de requête | Plan d'exécution

SQL | 0,01 secondes

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	DEPTH
SELECT STATEMENT			4	6	0
HASH JOIN			4	6	1
Access Predicates					
D.DEPARTMENT_ID=E.DEPARTMENT_ID+1-1					
TABLE ACCESS	DEPARTMENTS	FULL	1	3	2
Filter Predicates					
D.DEPARTMENT_NAME='Shipping'					
TABLE ACCESS	EMPLOYEES	FULL	107	3	2

Plan d'exécutions : Membres d'un département

- Utilisation de l'index sur l'ID département dans la table employees (EMP_DEPARTMENT_IX) possible (suppression de l'opération +1-1) :
  - L'accès à la table employees pour y trouver les informations recherchées se fera via le ROWID récupéré dans l'index.

```
select e.first_name, e.last_name
from employees e, departments d
where d.department_name = 'Shipping' and e.department_id = d.department_id
```

Résultat de requête | Plan d'exécution

SQL | 0,01 secondes

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	DEPTH
SELECT STATEMENT			10	4	0
NESTED LOOPS			10	4	1
NESTED LOOPS			10	4	2
TABLE ACCESS	DEPARTMENTS	FULL	1	3	3
Filter Predicates					
D.DEPARTMENT_NAME='Shipping'					
INDEX	EMP_DEPARTMENT_IX	RANGE SCAN	10	0	3
Access Predicates					
E.DEPARTMENT_ID=D.DEPARTMENT_ID					
TABLE ACCESS	EMPLOYEES	BY INDEX ROWID	10	1	2

Plan d'exécutions : Membres d'un département

- Ajout d'un index sur le nom du département (DEPT_NAME_IX) :
  - L'accès à la table département pour y trouver l'ID se fera via les ROWID récupérés dans l'index.

Feuille de calcul | Query Builder

```
select e.first_name, e.last_name
from employees e, departments d
where d.department_name = 'Shipping' and e.department_id = d.department_id
```

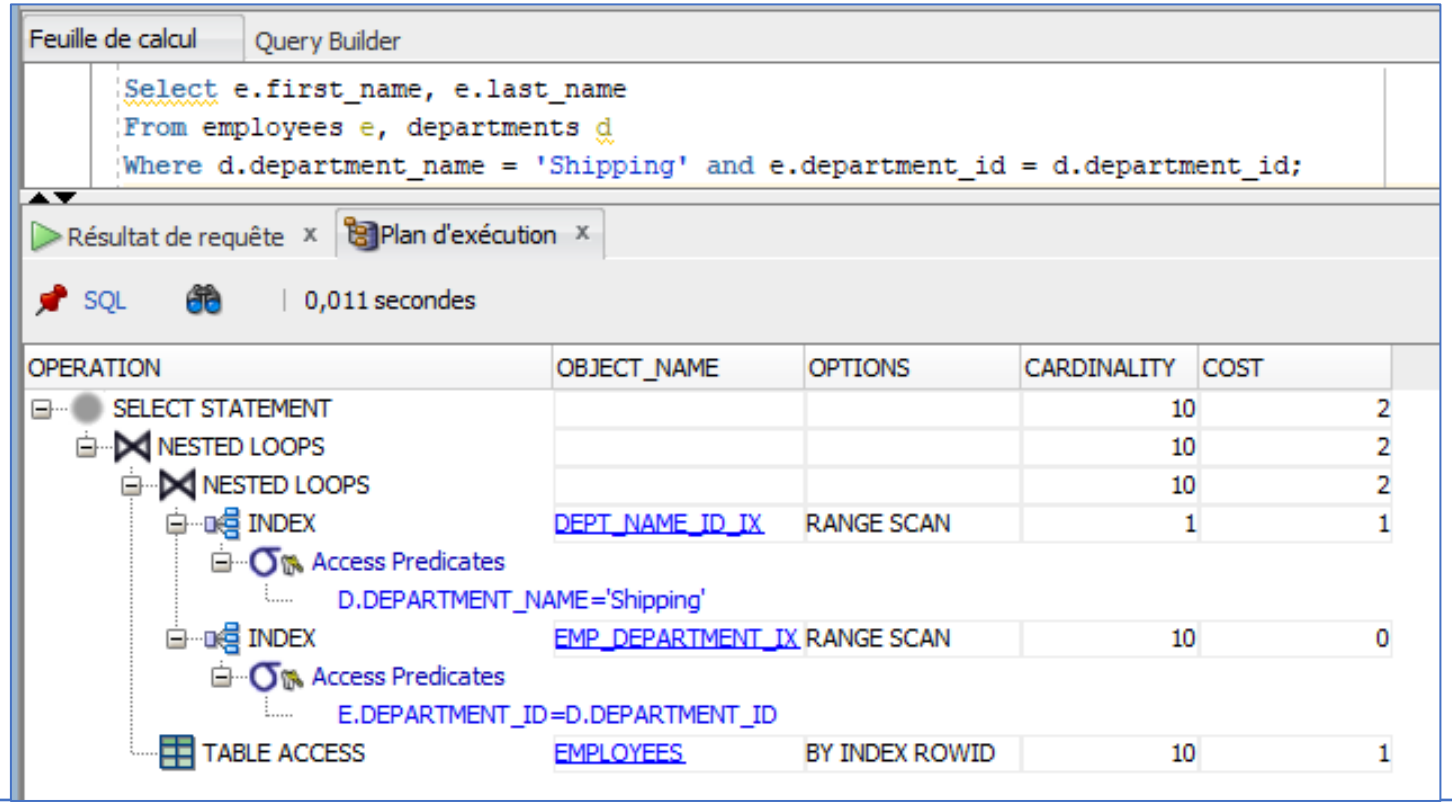
Résultat de requête | Plan d'exécution

SQL | 0 secondes

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	DEPTH
SELECT STATEMENT			10	3	0
NESTED LOOPS			10	3	1
NESTED LOOPS			10	3	2
TABLE ACCESS	DEPARTMENTS	BY INDEX ROWID BATCHED	1	2	3
INDEX	DEPT_NAME_IX	RANGE SCAN	1	1	4
Access Predicates					
D.DEPARTMENT_NAME='Shipping'					
INDEX	EMP_DEPARTMENT_IX	RANGE SCAN	10	0	3
Access Predicates					
E.DEPARTMENT_ID=D.DEPARTMENT_ID					
TABLE ACCESS	EMPLOYEES	BY INDEX ROWID	10	1	2

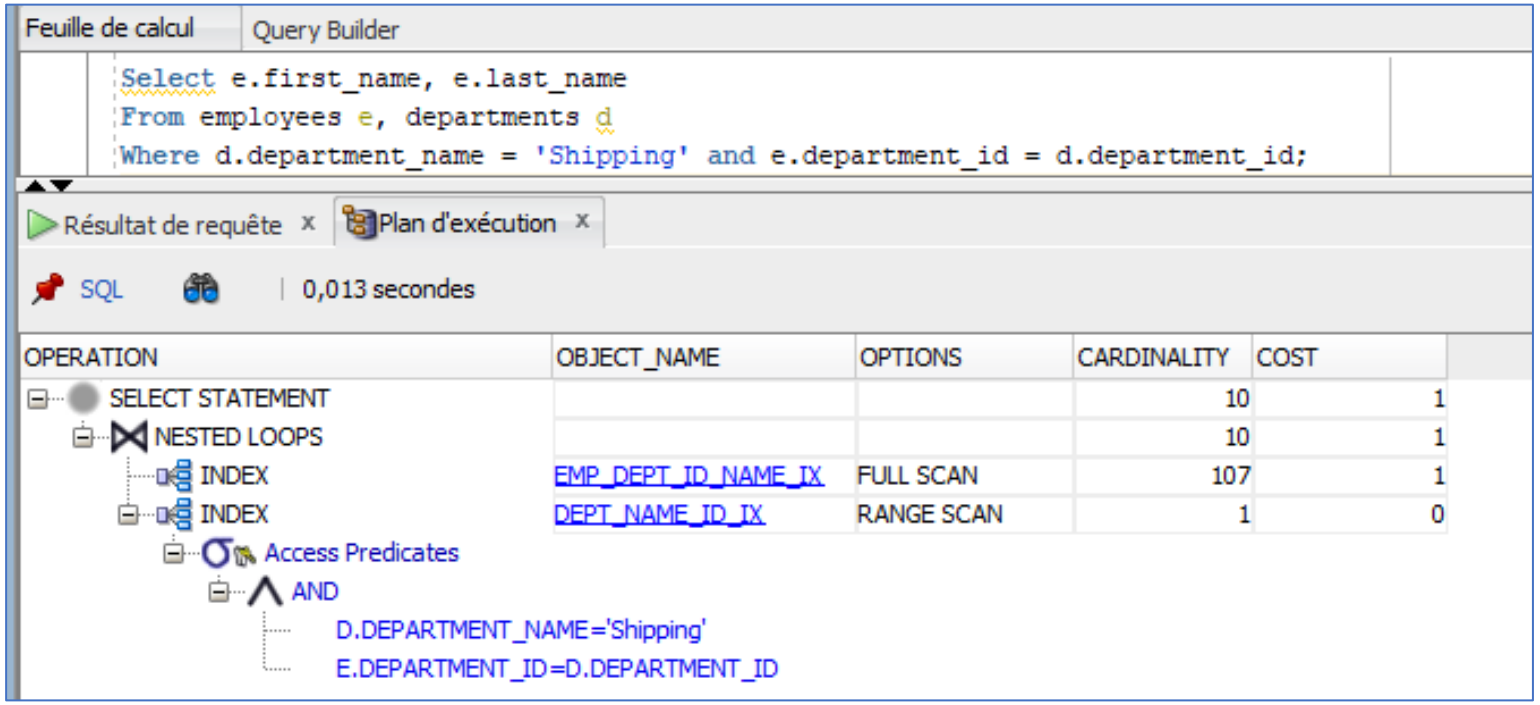
Plan d'exécutions : Membres d'un département

- Ajout d'un index composé nom du département + id du département dans la table département (DEPT_NAME_ID_IX).
  - Plus besoin d'accéder à la table département, toutes les informations sont dans l'index.



Plan d'exécutions : Membres d'un département

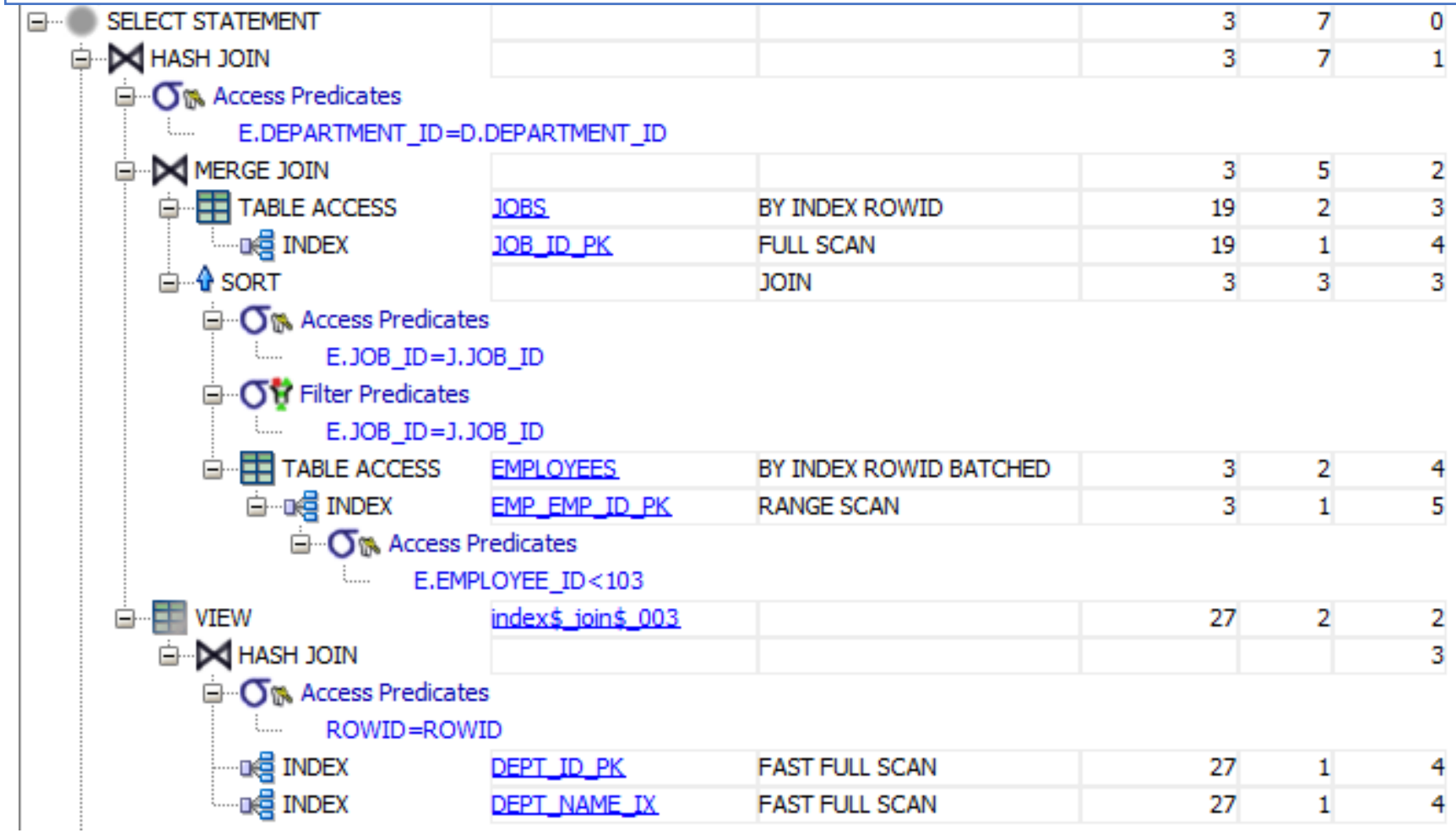
- Ajout d'un index composé de l'ID du département + le nom et prénoms dans la table employees :
  - Plus besoin d'accéder aux tables, toutes les informations sont dans les index.



```
5 - SELECT e.employee_id, j.job_title, e.salary, d.department_name
  -   FROM employees e, jobs j, departments d
1 -   WHERE  e.employee_id < 103
2 -     AND e.job_id = j.job_id
        AND e.department_id = d.department_id;
```

3

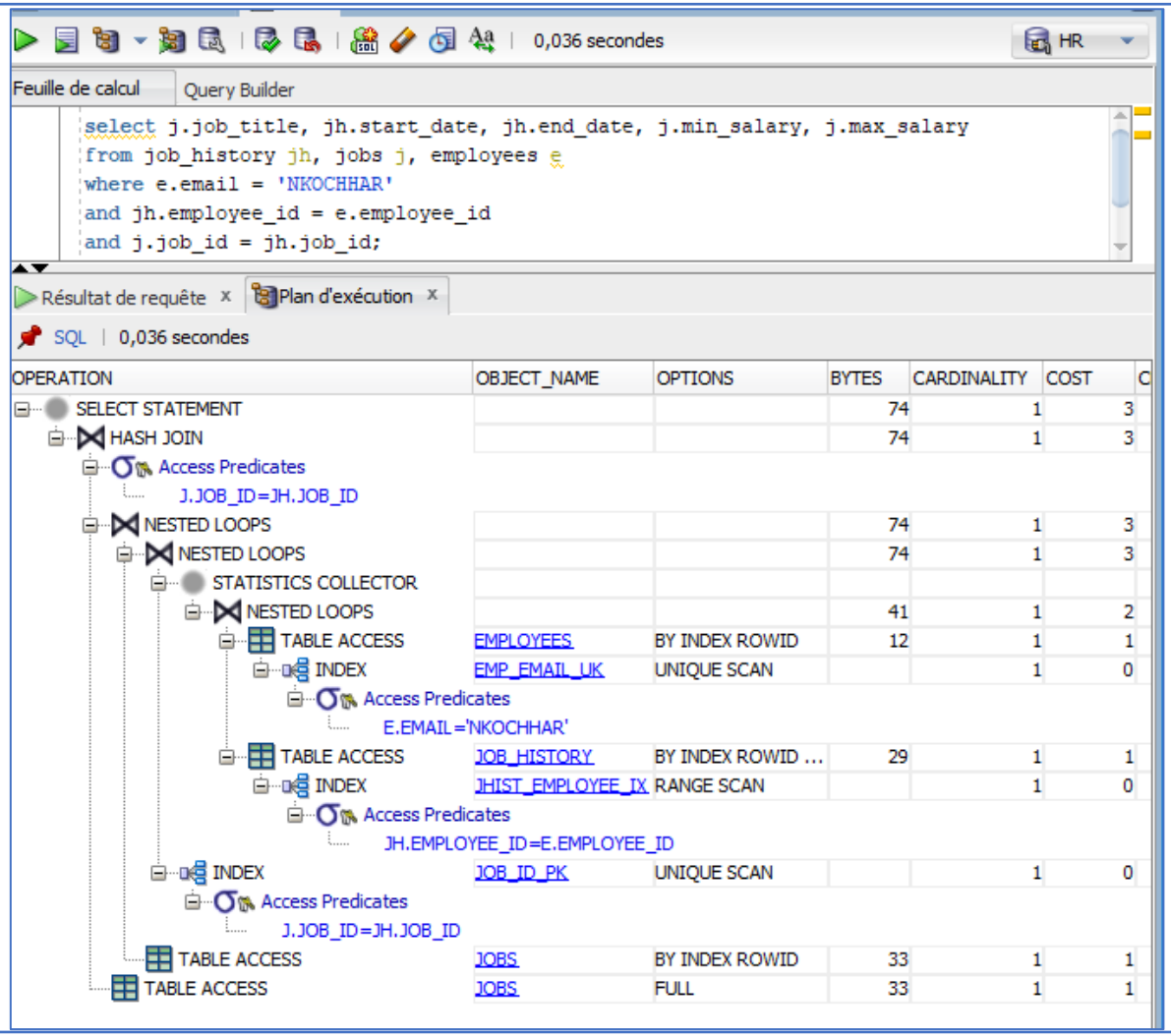
4



SELECT STATEMENT			3	7	0
HASH JOIN			3	7	1
Access Predicates					
E.DEPARTMENT_ID=D.DEPARTMENT_ID					
MERGE JOIN			3	5	2
TABLE ACCESS	JOBS	BY INDEX ROWID	19	2	3
INDEX	JOB_ID_PK	FULL SCAN	19	1	4
SORT		JOIN	3	3	3
Access Predicates					
E.JOB_ID=J.JOB_ID					
Filter Predicates					
E.JOB_ID=J.JOB_ID					
TABLE ACCESS	EMPLOYEES	BY INDEX ROWID BATCHED	3	2	4
INDEX	EMP_EMP_ID_PK	RANGE SCAN	3	1	5
Access Predicates					
E.EMPLOYEE_ID<103					
VIEW	index$_join$_003		27	2	2
HASH JOIN					3
Access Predicates					
ROWID=ROWID					
INDEX	DEPT_ID_PK	FAST FULL SCAN	27	1	4
INDEX	DEPT_NAME_IX	FAST FULL SCAN	27	1	4

Plan d'exécutions

- Historique des jobs d'un employé



OPERATION	OBJECT_NAME	OPTIONS	BYTES	CARDINALITY	COST	C
SELECT STATEMENT			74	1	3	
HASH JOIN			74	1	3	
Access Predicates						
J.JOB_ID=JH.JOB_ID						
NESTED LOOPS			74	1	3	
NESTED LOOPS			74	1	3	
STATISTICS COLLECTOR						
NESTED LOOPS			41	1	2	
TABLE ACCESS	EMPLOYEES	BY INDEX ROWID	12	1	1	
INDEX	EMP_EMAIL_UK	UNIQUE SCAN		1	0	
Access Predicates						
E.EMAIL='NKOCHHAR'						
TABLE ACCESS	JOB_HISTORY	BY INDEX ROWID ...	29	1	1	
INDEX	JHIST_EMPLOYEE_IX	RANGE SCAN		1	0	
Access Predicates						
JH.EMPLOYEE_ID=E.EMPLOYEE_ID						
INDEX	JOB_ID_PK	UNIQUE SCAN		1	0	
Access Predicates						
J.JOB_ID=JH.JOB_ID						
TABLE ACCESS	JOBS	BY INDEX ROWID	33	1	1	
TABLE ACCESS	JOBS	FULL	33	1	1	

Plan d'exécutions

- Somme des ventes faites à des clients dont le nom commence par A
  - Aucun index disponible

```
select sum(s.amount_sold) from sales s, customers c where c.cust_first_name like 'A%' and s.cust_id = c.cust_id;
```

Résultat de requête | Plan d'exécution

SQL | 0,019 secondes

OPERATION	OBJECT_NAME	OPTIONS	...	...	...	CARDINALITY	COST
SELECT STATEMENT						1	952
SORT		AGGREGATE				1	
HASH JOIN						178304	952
Access Predicates							
S.CUST_ID=C.CUST_ID							
TABLE ACCESS	CUSTOMERS	FULL				1370	423
Filter Predicates							
C.CUST_FIRST_NAME LIKE 'A%'							
PARTITION RANGE		ALL	1	28	4	918843	523
TABLE ACCESS	SALES	FULL	1	28	4	918843	523

Plan d'exécutions

- Somme des ventes faites à des clients dont le nom commence par A
  - Index bitmap sur l'ID des clients dans la table des ventes, pas de gain.

```
select sum(s.amount_sold) from sales s, customers c where c.cust_first_name like 'A%' and s.cust_id = c.cust_id;
```

Résultat de requête | Plan d'exécution

SQL | 0,018 secondes

OPERATION	OBJECT_NAME	OPTIONS	...	...	...	CARDINALITY	COST
SELECT STATEMENT						1	952
SORT		AGGREGATE				1	
HASH JOIN						178304	952
Access Predicates							
S.CUST_ID=C.CUST_ID							
NESTED LOOPS						178304	952
NESTED LOOPS							
STATISTICS COLLECTOR							
TABLE ACCESS	CUSTOMERS	FULL				1370	423
Filter Predicates							
C.CUST_FIRST_NAME LIKE 'A%'							
PARTITION RANGE		ALL	1	28	7		
BITMAP CONVERSION		TO ROWIDS					
BITMAP INDEX	SALES_CUST_BIX	SINGLE VALUE	1	28	7		
Access Predicates							
S.CUST_ID=C.CUST_ID							
TABLE ACCESS	SALES	BY LOCAL IND...	1	1	7	130	523
PARTITION RANGE		ALL	1	28	11	918843	523
TABLE ACCESS	SALES	FULL	1	28	11	918843	523

Plan d'exécutions

- Somme des ventes faites à des clients dont le nom commence par A
  - Index sur le nom client + cust_id dans la table des clients, 30 %

```
select sum(s.amount_sold) from sales s, customers c where cust_first_name like 'A%'and s.cust_id = c.cust_id;
```

SQL Tuning Advisor | Résultat de requête | Plan d'exécution

SQL | 0,034 secondes

OPERATION	OBJECT_NAME	OPTIONS	...	...	...	CARDINALITY	COST
SELECT STATEMENT						1	535
SORT		AGGREGATE				1	
HASH JOIN						178304	535
Access Predicates							
S.CUST_ID=C.CUST_ID							
NESTED LOOPS						178304	535
NESTED LOOPS							
STATISTICS COLLECTOR							
INDEX	CUST_NAME_ID_IDX	RANGE SCAN				1370	6
Access Predicates							
CUST_FIRST_NAME LIKE 'A%'							
Filter Predicates							
CUST_FIRST_NAME LIKE 'A%'							
PARTITION RANGE		ALL	1	28	7		
BITMAP CONVERSION		TO ROWIDS					
BITMAP INDEX	SALES_CUST_BIX	SINGLE VALUE	1	28	7		
Access Predicates							
S.CUST_ID=C.CUST_ID							
TABLE ACCESS	SALES	BY LOCAL IND...	1	1	7	130	523
PARTITION RANGE		ALL	1	28	11	918843	523
TABLE ACCESS	SALES	FULL	1	28	11	918843	523

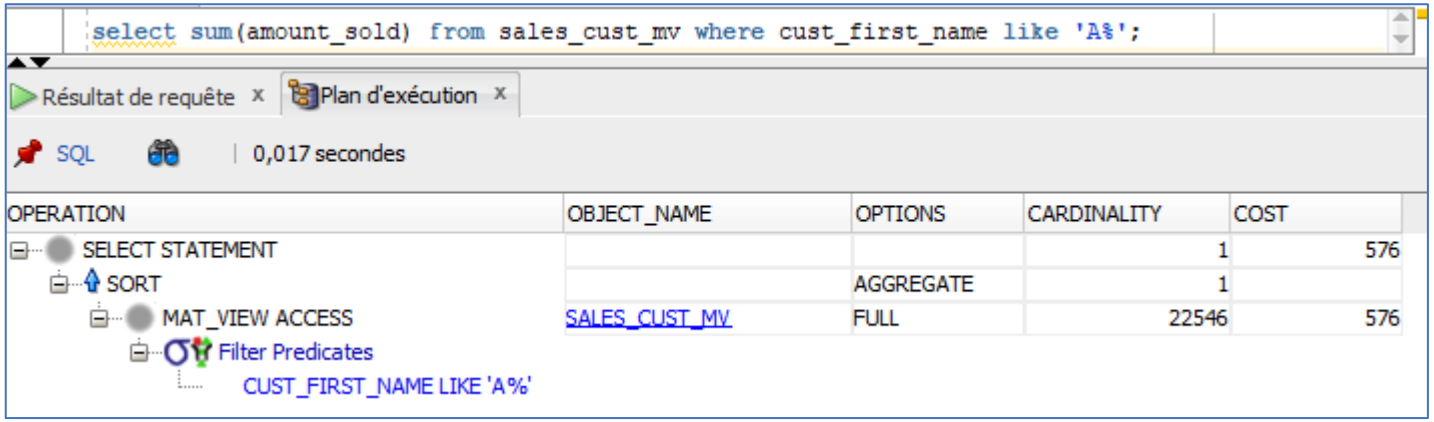
Plan d'exécutions

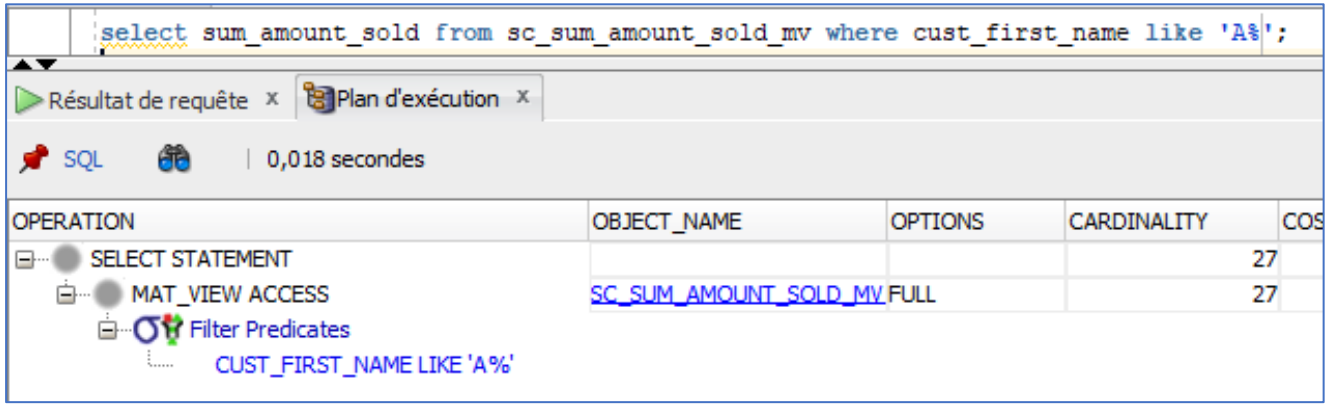
- Somme des ventes faites à des clients dont le nom commence par A
  - Vue matérialisée de pré-calcul de la jointure entre sales et customers :

```
SELECT s.amount_sold, c.cust_first_name
FROM sales s, customers c
where s.cust_id = c.cust_id
```

  - Vue matérialisée de pré-calcul des sommes de ventes par client :

```
SELECT sum(s.amount_sold) as sum_amount_sold,
c.cust_first_name
FROM sales s, customers c
where s.cust_id = c.cust_id
group by c.cust_first_name
```





Conseils de l'optimiseur…

```
SQL Text    : select count(s.amount_sold) from sales_nopart s, customers c
              where c.cust_first_name like 'A%' and s.cust_id = c.cust_id

-------------------------------------------------------------------------------
FINDINGS SECTION (1 finding)
-------------------------------------------------------------------------------

1- Index Finding (see explain plans section below)
--------------------------------------------------
  Le plan d'exécution de cette instruction peut être amélioré en créant un ou
  plusieurs index.

  Recommendation (estimated benefit: 64.22%)
  ------------------------------------------
  - Envisagez d'exécuter Access Advisor pour améliorer la conception du
    schéma physique ou de créer l'index recommandé.
    create index SH.IDX$$_00970001 on SH.CUSTOMERS("CUST_FIRST_NAME","CUST_ID")
    ;

  - Envisagez d'exécuter Access Advisor pour améliorer la conception du
    schéma physique ou de créer l'index recommandé.
    create index SH.IDX$$_00970002 on SH.SALES_NOPART("CUST_ID");

  Rationale
  ---------
    La création des index recommandés améliore de façon considérable le plan
    d'exécution de cette instruction. Il pourrait cependant être préférable
    d'exécuter "Access Advisor" en utilisant une charge globale SQL
    représentative contrairement à une seule instruction. Ceci permettra
    d'obtenir des recommandations d'index complètes prenant en compte le coût
    de maintenance des index et de la consommation d'espace supplémentaire.
```

```
1- Original
-----------
Plan hash value: 1184928097

-----------------------------------------------------------------------------------------
| Id  | Operation           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |               |     1 |    17 |  1654   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE     |               |     1 |    17 |            |          |
|*  2 |   HASH JOIN         |               |   316K|  5255K|  1654   (1)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL| CUSTOMERS     |  2432 | 29184 |   423   (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| SALES_NOPART  |   918K|  4486K|  1229   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------


2- Using New Indices
--------------------
Plan hash value: 1444798750


-----------------------------------------------------------------------------------------
| Id  | Operation           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |               |     1 |    17 |   592   (2)| 00:00:01 |
|   1 |  SORT AGGREGATE     |               |     1 |    17 |            |          |
|*  2 |   HASH JOIN         |               |   316K|  5255K|   592   (2)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN | IDX$$_00970001|  2432 | 29184 |    10   (0)| 00:00:01 |
|   4 |    INDEX FAST FULL SCAN| IDX$$_00970002|   918K|  4486K|   579   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------
```

Avec les indexes positionnés :

```
-----------------------------------------------------------------------------------------
| Id  | Operation                | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                    |     1 |    17 |   430   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE          |                    |     1 |    17 |            |          |
|*  2 |   HASH JOIN              |                    |   316K|  5255K|   430   (1)| 00:00:01 |
|*  3 |    VIEW                  | index$_join$_002   |  2432 | 29184 |   126   (0)| 00:00:01 |
|*  4 |     HASH JOIN            |                    |       |       |            |          |
|*  5 |      INDEX RANGE SCAN    | CUST_NAME_IDX      |  2432 | 29184 |    10   (0)| 00:00:01 |
|   6 |      INDEX FAST FULL SCAN| CUSTOMERS_PK       |  2432 | 29184 |   145   (0)| 00:00:01 |
|   7 |    BITMAP CONVERSION TO ROWIDS|               |   918K|  4486K|   302   (0)| 00:00:01 |
|   8 |     BITMAP INDEX FAST FULL SCAN| SALES_NOPART_CUST_BIX |  |  |  |  |
```

Rappel : jointures

- Les jointures représentent le croisement de plusieurs tables dans une requête.
- Les jointures « internes » (ou naturelles, ou par défaut), ont comme effet qu'a un enregistrement d'une table, correspond naturellement un enregistrement de l'autre.
- Par exemple : je veux l'ensemble des employés et leur département d'appartenance.
  - SELECT ENAME, DNAME FROM EMP E, DEPT D
    WHERE D.DEPTNO = E.DEPTNO

  retournera tous les employés et leur département d'appartenance.
- Mais que ce passe-t'il si des employés ne sont pas encore dans un département ? il n'apparaissent pas dans les résultats de la requête.
- Si on veux voir ces employés, il faut faire des jointures externes.

Rappel : jointures

- Les jointures externes vont prendre en considération tous les enregistrements d'une table (dite directrice) même ceux qui n'ont pas de correspondants avec la table jointe. Il y a alors trois sortes de jointures externes : gauche, droite, toutes selon que ce sont les engistrements de la table de gauche, de droite ou des deux.
- Par exemple :
  - Si on veux tous les employés même ceux qui n'ont pas de département :
    - **Select** ename, dname **from** emp e **left outer join** dept d **on** e.deptno = d.deptno;
  - Si on veux tous les départements même ceux qui n'ont pas d'employés :
    - **Select** ename, dname **from** emp e **right outer join** dept d **on** e.deptno = d.deptno;
  - Si on veux les deux cas de figure :
    - **Select** ename, dname **from** emp e **full outer join** dept d **on** e.deptno = d.deptno;

Rappel : jointures

- Oracle permet de spécifier l'attribut utilisé pour réaliser la jointure via le suffixe (+).
- Par exemple :

```
SELECT * FROM EMP E, BONUS B
WHERE E.ENAME = B.ENAME(+) AND E.COMM = B.COMM;
```

  va récupérer tous les employés qui ont reçu une comm dont le montant existe dans la table des bonus même s'ils leur nom n'a pas été enregistrés dans la table des bonus.

- Peux aussi s'écrire :

```
SELECT * FROM EMP E LEFT OUTER JOIN BONUS B
ON E.ENAME = B.ENAME
WHERE E.COMM = B.COMM
```

Optimiseur : Indexes et jointures externes

Cas des jointures externes, la table non-externe (celle qui n'est pas suivie du (+)) est la table directrice.

Pour l'utilisation des index, plusieurs cas sont possibles :

- Les clauses WHERE ne portant pas sur la jointure mais sur la table non-externe sont appliquées AVANT la jointure et les éventuels index existants peuvent être utilisés.
- Les clauses WHERE ne portant pas sur la jointure mais sur la table externe et qui sont suivies par un (+) sont appliquées AVANT la jointure et les éventuels index existants peuvent être utilisés.
- Les clauses WHERE ne portant pas sur la jointure mais sur la table externe et sans le (+) sont appliquées APRES la jointure et les éventuels index existants ne peuvent pas être utilisés (Oracle 8).

Optimiseur : Indexes et jointures externes

Par exemple :

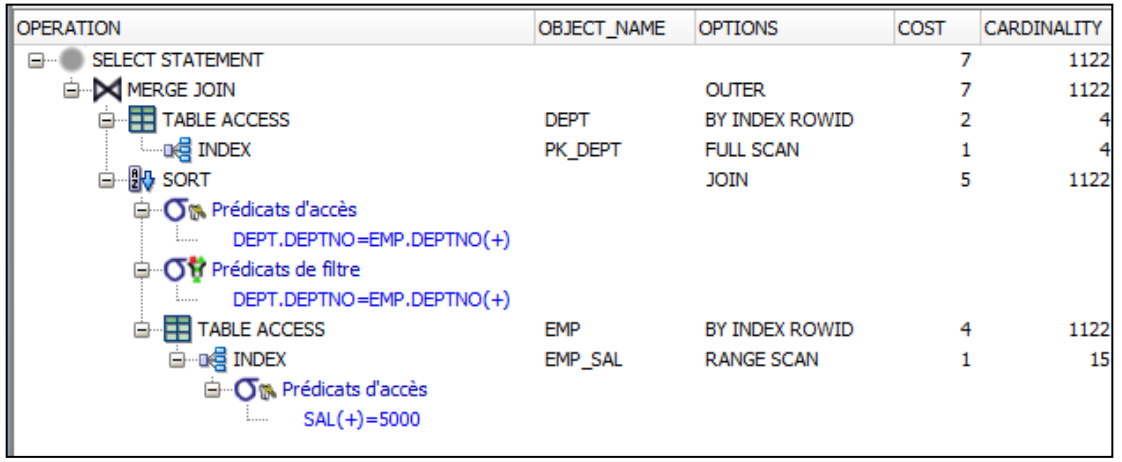
- SELECT DNAME, ENAME FROM DEPT, EMP WHERE DEPT.DEPTNO = EMP.DEPTNO(+) AND DNAME = 'ACCOUNTING'
- CREATE INDEX IDEPT1 ON DEPT(DNAME)

La table directrice est DEPT et l'index sur DNAME sert à y accéder.

Dans le cas :

- SELECT DNAME, ENAME FROM DEPT, EMP WHERE DEPT.DEPTNO = EMP.DEPTNO(+) AND SAL(+) = 5000
- CREATE INDEX EMP_SAL ON EMP (SAL)

La table directrice est toujours DEPT et un FTS est utilisé pour y accéder. La condition sur SAL est appliquée AVANT (grace au (+)) et l'index EMP_SAL est utilisé pour accéder à EMP.



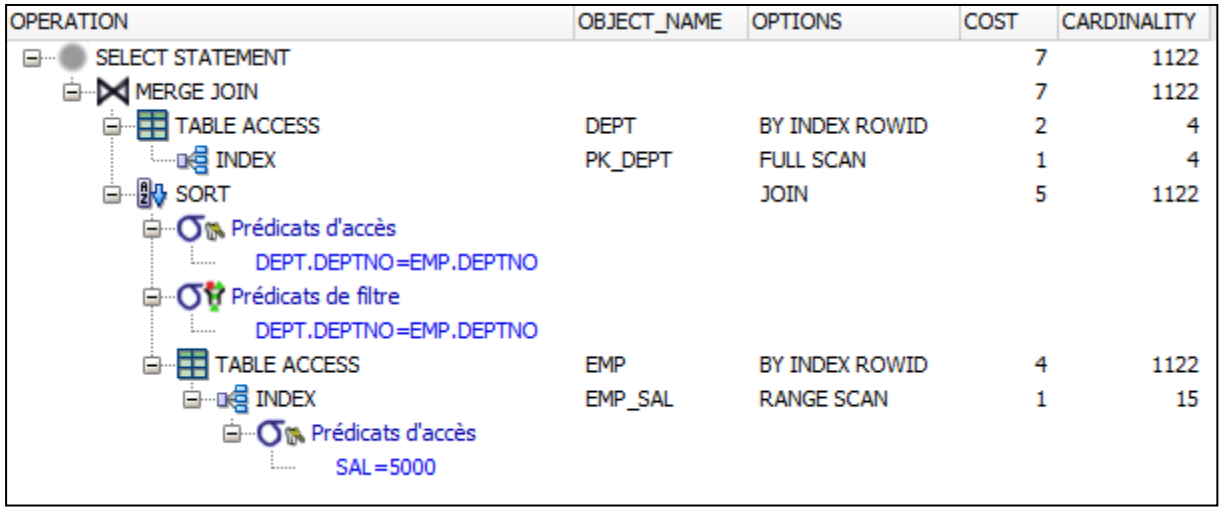
OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			7	1122
MERGE JOIN		OUTER	7	1122
TABLE ACCESS	DEPT	BY INDEX ROWID	2	4
INDEX	PK_DEPT	FULL SCAN	1	4
SORT		JOIN	5	1122
Prédicats d'accès				
DEPT.DEPTNO=EMP.DEPTNO(+)				
Prédicats de filtre				
DEPT.DEPTNO=EMP.DEPTNO(+)				
TABLE ACCESS	EMP	BY INDEX ROWID	4	1122
INDEX	EMP_SAL	RANGE SCAN	1	15
Prédicats d'accès				
SAL(+)=5000				

Optimiseur : Indexes et jointures externes

Egalement :

- SELECT DNAME, ENAME FROM DEPT, EMP WHERE DEPT.DEPTNO = EMP.DEPTNO (+) AND SAL = 5000
- CREATE INDEX EMP_SAL ON EMP (SAL)

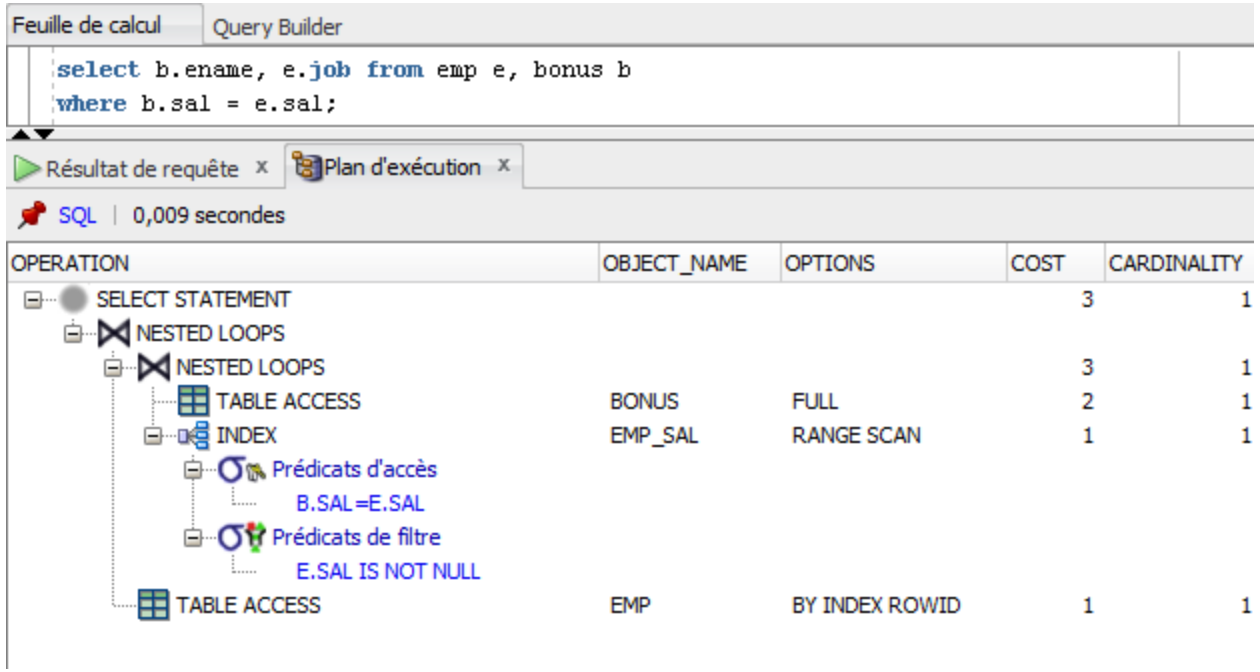
Idem que pour précédemment, et l'index est utilisé.



OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			7	1122
MERGE JOIN			7	1122
TABLE ACCESS	DEPT	BY INDEX ROWID	2	4
INDEX	PK_DEPT	FULL SCAN	1	4
SORT		JOIN	5	1122
Prédicats d'accès				
DEPT.DEPTNO=EMP.DEPTNO				
Prédicats de filtre				
DEPT.DEPTNO=EMP.DEPTNO				
TABLE ACCESS	EMP	BY INDEX ROWID	4	1122
INDEX	EMP_SAL	RANGE SCAN	1	15
Prédicats d'accès				
SAL=5000				

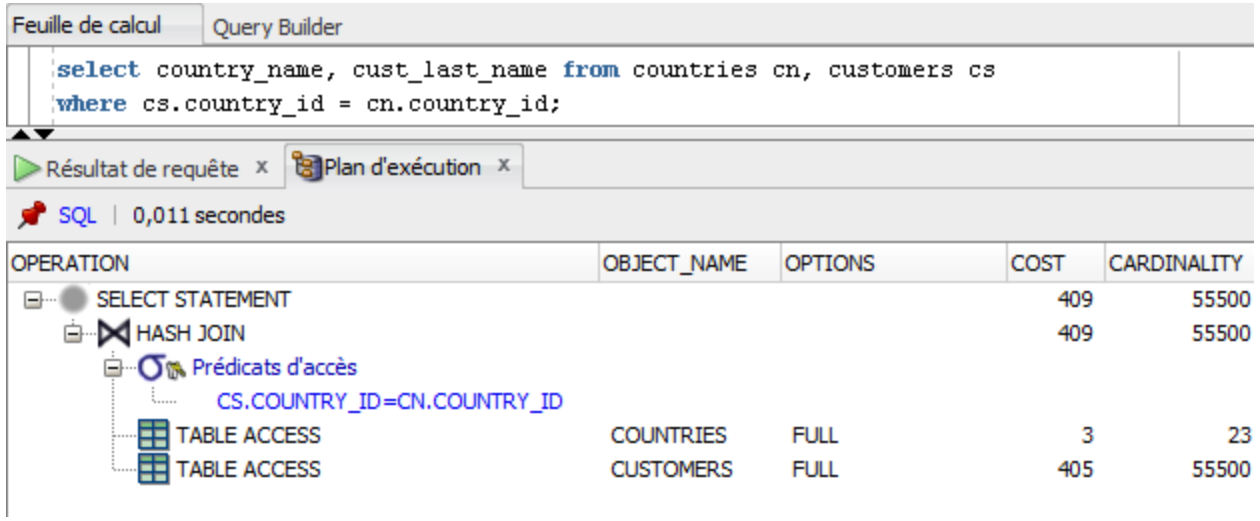
Méthodes de jointures de l'optimiser : Nested Loop

- Méthode des boucles imbriquées :
  - Efficace pour des ensembles réduits de données
  - Requière que la table interne soit dirigée par (dépende de) la table externe pour éviter que les même enregistrements soient récupérés par plusieurs itérations.



Méthodes de jointures de l'optimiser : Hash Join

- Méthode de la jointure par table de hash :
  - Efficace pour des ensembles importants de données
  - Efficace si une partie importante de la plus petite de source de données fait parti de la jointure.
  - L'optimiser construit une table de clés de hash en mémoire à partir de la table de données source la plus petite puis l'utilise pour parcourir la plus grande source de données.



Méthodes de jointures de l'optimiser : Sort Merge Join

- La méthode de la jointure par tri fusion (sort merge) est souvent moins efficace que la méthode par Hash sauf si :
  - Les lignes sont déjà triées ET pas d'opération de tris
- Elle consiste en deux étapes :
  - Tris par rapport à la clé
  - Fusion des ensembles triés

Et sera également plus efficace que les boucles imbriquées sur des ensembles important de données et lorsque les conditions de jointures sont des inégalitées (>, <, …).

```
select dname, ename from emp e, dept d
where e.deptno = d.deptno;
```

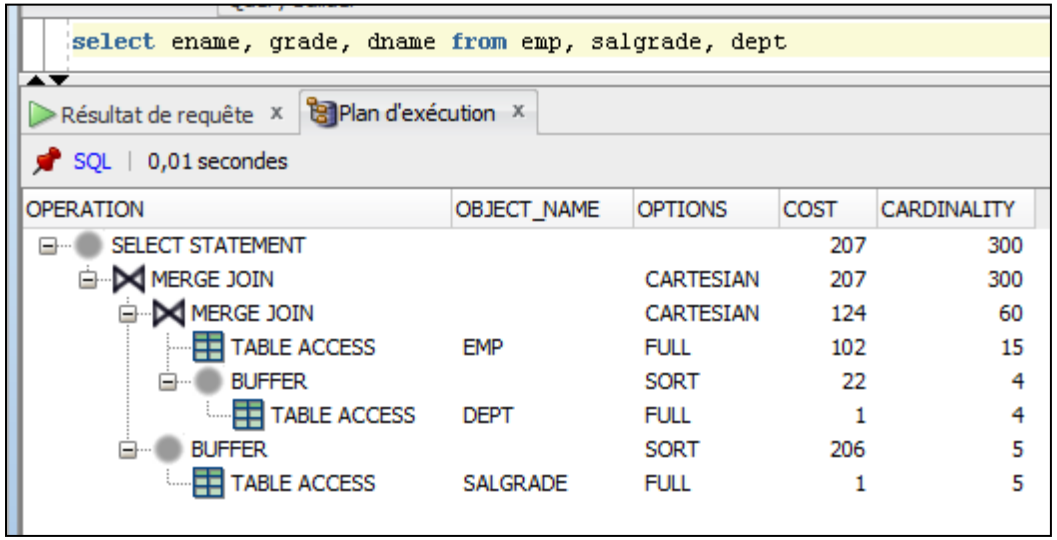
Résultat de requête | Plan d'exécution

SQL | 0,007 secondes

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			8	13
MERGE JOIN			8	13
TABLE ACCESS	DEPT	BY INDEX ROWID	2	4
INDEX	PK_DEPT	FULL SCAN	1	4
SORT		JOIN	6	13
Prédicats d'accès				
E.DEPTNO=D.DEPTNO				
Prédicats de filtre				
E.DEPTNO=D.DEPTNO				
TABLE ACCESS	EMP	BY INDEX ROWID	5	13
INDEX	IEMP2	FULL SCAN	4	13
Prédicats de filtre				
E.DEPTNO IS NOT NULL				

Méthode de jointure de l'optimiser : produit cartésien

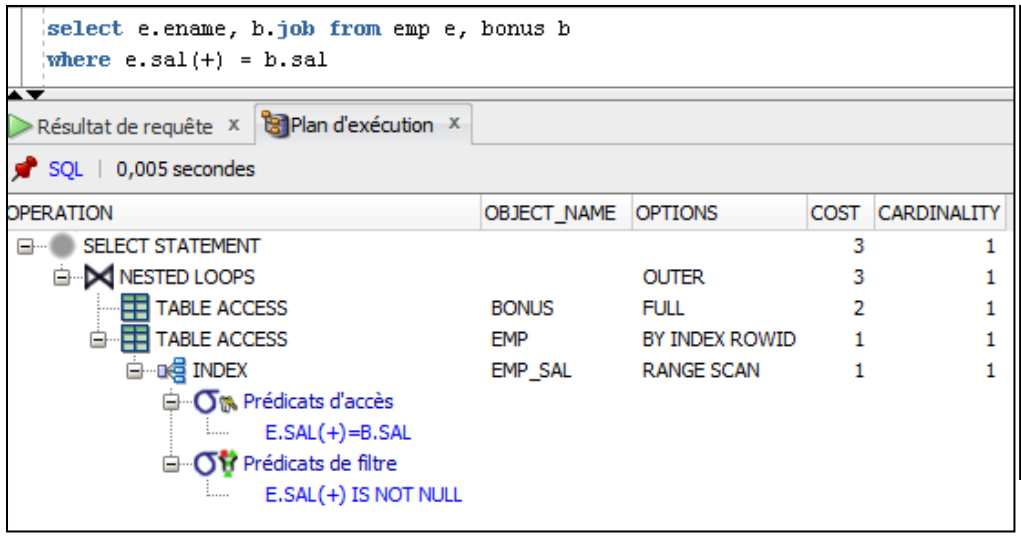
- Utilisé lorsqu'il n'y a pas de condition de jointure.

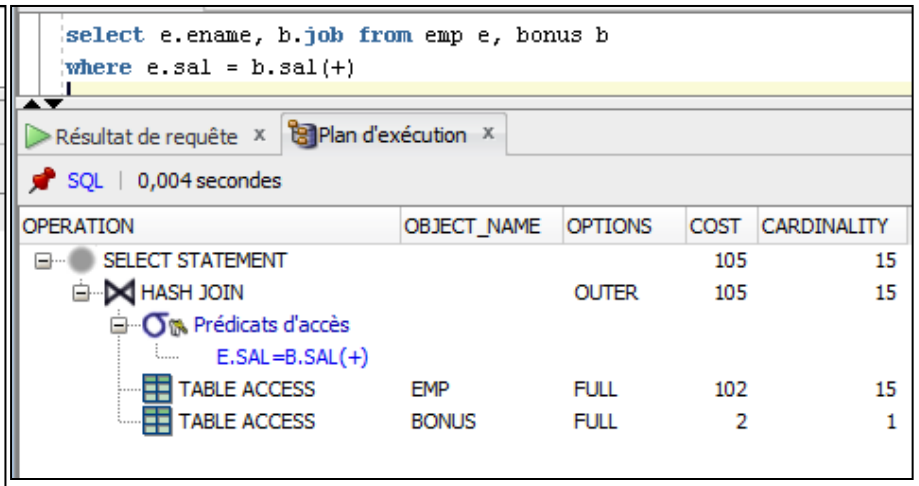


OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			207	300
MERGE JOIN		CARTESIAN	207	300
MERGE JOIN		CARTESIAN	124	60
TABLE ACCESS	EMP	FULL	102	15
BUFFER		SORT	22	4
TABLE ACCESS	DEPT	FULL	1	4
BUFFER		SORT	206	5
TABLE ACCESS	SALGRADE	FULL	1	5

Méthodes de jointures de l'optimiser : jointures externes

- Dans le cas des jointures externes, les tables directrices sont explicites et l'optimiser n'a plus la possibilité de choisir l'ensemble de données le plus efficace pour décider des tables directrice des jointures.





Éléments critiques de la performance de l'application

- « *Si les DBA peuvent identifier les points et problèmes qui sont les causes de ralentissement, les développeurs peuvent les résoudre.* »

- 80 % des gains de performances sont dues à une optimisation du code de l'application
  - Par son modèle,
  - Par le codage des requêtes SQL.

- Mais, la performance est **dynamique** et la performance d'une requête dans un contexte (base de dev) ne sera pas représentatif d'une performance dans un autre (prod) ni dans la durée. Le rôle du DBA est d'apporter au développeur la vue sur le comportement dynamique de son application.

Outillage de la mesure de la performance

- Mise en place de statistiques dynamiques sur le fonctionnement de la base :
  - Observation du fonctionnement des process internes de la base,
  - Observation du fonctionnement des applications (SQL) et du modèle de données (accès aux tables/indexes/enregistrements).
- Demande un temps d'exécution minimal pour être pertinents,
- Processus récurent « d'administration » de la base de données.

Analyse des statistiques : les causes possible de perte de performance

- **Les attentes (wait) sont la cause des pertes (ou baisses de performances).**
- **Elles sont classifiées selon leur cause principale :**

- **Actions d'administration** : commandes réalisées par un utilisateur privilégié qui induit une attente des autres utilisateurs (la reconstruction d'un indexe par exemple).
- **Application** : tous les "défauts" induits par le modèle qui induisent des blocages (mise en attente) du fait de l'accès concurent sur une donnée pour une cause explicite ou implicite (modification)
- **Architecture cluster** : cach global, synchronisation des noeuds, …
- **Actions liées à un commit** : évennements d'attentes liées au redolog
- **Accès concurrent** par beaucoup de sessions vers une même ressource (parsing SQL, verrou sur des buffers, …)
- **Configuration** : construction de ressources (log, logfile, buffer, …) de taille insuffisante
- **Idle** : sessions inactives
- **Réseau** : attente de données envoyée par le réseau
- **Scheduler** : attente liée à une priorisation du manager de ressources
- **System I/O** : attente sur les I/O système pour les processus en arrière plan (autre que ceux liés à la gestion des données utilisateur).
- **User I/O** : attente de lecture de bloc (processus SMON, or MMON).

Droits nécessaires à l'accès aux plans d'exécution

- Les informations relatives au plan d'exécution sont positionnées dans des espaces d'Oracle (V$MYSTAT) accessibles uniquement aux utilisateurs qui ont les droits suivants :
  - SELECT_CATALOG_ROLE
  - SELECT ANY DICTIONARY