

UNIVERSITE DE CORSE
2025-2026
Master DFS-DE 1^{ère} année

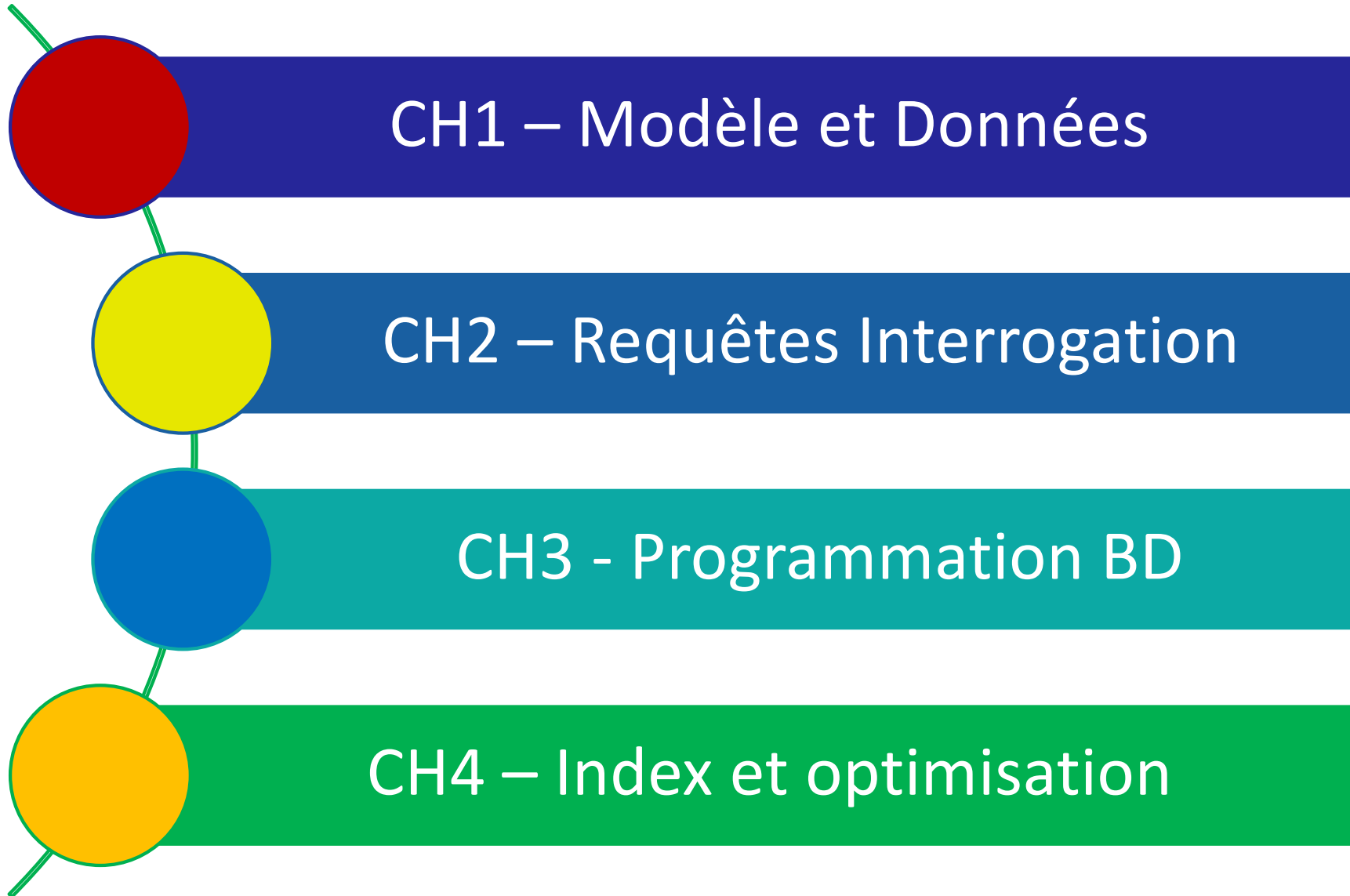
Bases de données et Optimisation

CH4 – Index et Optimisation



Evelyne VITTORI
vittori_e@univ-corse.fr

Optimisation et Bases de données



Objectifs du chapitre



- Comprendre la notion d'index et le rôle des index dans l'optimisation
- Comprendre les étapes de l'exécution des requêtes SQL
- Découvrir l'optimiseur automatique de PostgreSQL
- Générer, comprendre et analyser les plans d'exécution des requêtes
- Savoir utiliser les plans d'exécution et les outils statistiques pour optimiser



Notion d'index

Définition, Utilisation,
Types d'index



UNIVERSITÀ DI CORSICA
PASQUALE PAOLI

Notion d'index



- Qu'est ce qu'un index?
 - structure de donnée (style tableau) associée à un ou plusieurs attributs
 - valeur d'attribut → tuple (Row ID= adresse d'un tuple)

I_BATNOM

BATNOM	Pointeur (ROWID)
INDEPENDANCE	4
KALISTE	5
LIBERTE	2
LOUISIANE	6
ORION	1
SAPHIR	3

Table BATEAUX

RowID	BATNUM	BATNOM	BATPORT	CAPACITE
1	B001	ORION	ANTIBES	10
2	B002	LIBERTE	ANTIBES	10
3	B003	SAPHIR	BASTIA	12
4	B004	INDEPENDANCE	CALVI	6
5	B005	KALISTE	BASTIA	6
6	B006	LOUISIANE	AJACCIO	4

Index sur BATNOM

Notion d'index



- Pour quoi faire?
 - pour **accélérer** l'exécution des requêtes d'interrogation
 - pour assurer la **vérification des contraintes** d'intégrité:
clés primaires et étrangères

- *Exemple*

```
select CAPACITE  
from BATEAUX  
where BATNOM='LIBERTE'
```

Quel est l'intérêt de l'index I_BATNOM sur cette requête?

✓ Recherche rapide de LIBERTE dans I-BATNOM (trié)

✓ Accès direct au tuple de rowID 2

Notion d'index



- Plusieurs index peuvent être définis sur une même table
- Un index peut porter sur un groupe d'attribut (index composé)
 - ex: clé primaire multi-attributs
- Un index peut :
 - assurer l'unicité des valeurs d'un attribut
 - assurer l'unicité et l'absence de valeur nulle (index primaire)

Notion d'index



- Sur quels attributs créer des index?
 - les clés primaires (index créés en général de manière **implicite**)
 - les clés étrangères
 - les attributs souvent utilisés en critères de **jointure** et de sélection

Pourquoi ne faut-il pas abuser des index?

Les performances des commandes d'insertion et de suppression sont fortement pénalisées car tous les index portant sur la table doivent aussi être mis à jour

Informations sur les Index dans PostgreSQL

- Vue Système **pg_indexes** : contient tous les détails sur les index d'une base de données
- Colonnes principales
 - **schemaname** : nom du schéma auquel appartient la table indexée.
 - **tablename** : nom de la table sur laquelle l'index a été créé.
 - **indexname** : nom de l'index.
 - **indexdef** : définition complète de l'index (SQL) qui indique comment il a été créé.

Exercice 1 : Accéder à la liste des index sur une table

- Cherchez une solution pour connaître la liste des index existants (nom, table concernée et définition) :
 1. sur l'ensemble de votre base de données croisières (attention uniquement sur le schéma public)
 2. spécifiquement sur la table croisières de votre base de données bateaux!



Requêtes de gestion des index



- Création d'un index

```
CREATE INDEX nom_index ON nom-Table (attribut1, ...);
```

Création d'index sur les clés étrangères de la table
CROISIERES

```
CREATE INDEX FK_croisBatnum ON CROISIERES(BATNUM);
```

```
CREATE INDEX FK_croisSknum ON CROISIERES(SKNUM);
```

- Suppression d'un index

```
DROP INDEX nom_index ;
```

Notion de densité

$D = \text{Nombre de clés} / \text{Nombre de tuples}$

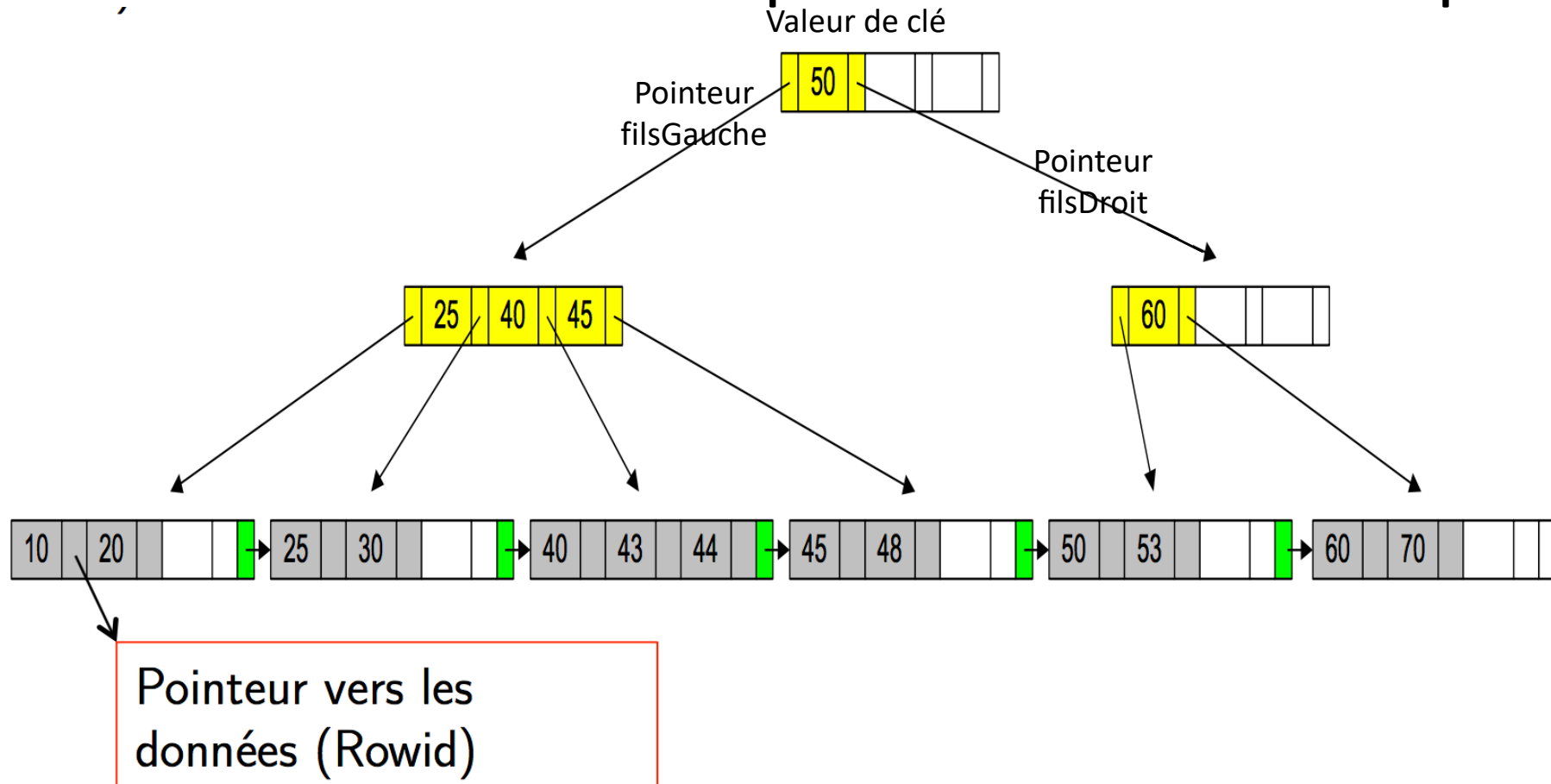
- Un index est dit « **dense** » s'il possède autant de clés que de tuples
 - $D=1$
- Un index est dit « **creux** », « non dense » ou « épars » s'il associe une clé à un ensemble de tuples (**bloc**).

Types d'organisation d'index

- **Index B-TREE**: Type d'index par défaut de PostgreSQL et d'ORACLE Et de la plupart des SGBDr
 - Arbre « balancé » : arbre binaire ordonné et équilibré
 - Toutes les feuilles sont à la même profondeur
 - Chaque nœud contient une valeur de clé (ou un bloc de k clés triées pour les arbres-B +) et deux pointeurs vers les nœuds fils (k+1 fils pour les arbres B+)
 - Les feuilles comportent la valeur de clé et le ROWID du tuple associé
 - Ne permet pas d'indexer les valeurs NULL

Types d'organisation d'index

■ Index B-TREE+: exemple avec clés numériques



Avantage: temps de recherche constant quelque soit la valeur recherchée

Principaux types d'index

■ Index BITMAP

- Pour chaque valeur de clé, un tableau de bits (dit bitmap) est créé avec autant de bits que de lignes dans la table
- Recherche **très rapide** si le nombre de valeurs possibles est faible

Clés	1	2	3	4	5
M.	0	0	0	0	1
Melle	0	0	1	1	0
Mme	1	0	0	0	0
NULL	0	1	0	0	0

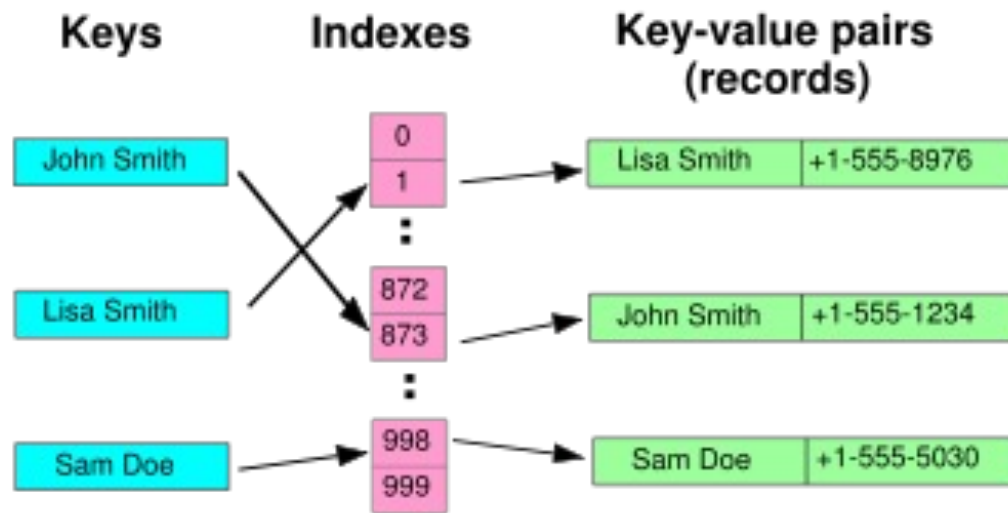


Uniquement dans
ORACLE

Principaux types d'index

■ Index Hachage

- Accès à un tuple à partir d'un calcul appliqué sur la valeur de la clé (fonction de hachage ayant comme résultat le rowID)



Inconvénient:
Gestion des conflits
lorsque deux clés
ont la même valeur
de hachage

Ces index sont adaptés pour des clés
de recherche uniques ou des valeurs
souvent utilisées dans des requêtes
avec =

Quelques conseils d'indexation

- Créer uniquement les index nécessaires
- Supprimer les index inutilisés
- Créer des index sur les clés étrangères
- Attention:
 - La création d'un index peut impacter négativement la performance des requêtes de mise à jour
 - Seules les colonnes uniques ou de type Clés Primaires sont **automatiquement indexées**

Utilisation des outils proposés par le SGBD (cf. suite du chapitre)

Index et optimisation

- Les index jouent un rôle essentiel dans les performances de l'exécution des requêtes d'interrogation.
- L'analyse des plans d'exécution des requêtes permettra de vérifier leur utilisation effective et leur impact.

Orientation du Choix
(cf. suite du chapitre)

Par exemple, on pourra constater que l'utilisation de fonctions dans l'expression des prédicats Where sur des colonnes indexées invalide l'utilisation de l'index



Démarche d'optimisation



UNIVERSITÀ DI CORSICA
PASQUALE PAOLI

Optimisation et indépendance logique

- Principe d'Indépendance logique/physique
 - La BD est créée et manipulée sans avoir besoin de connaître :
 - L'organisation physique des données stockées (fichiers, ect...)
 - Les algorithmes d'exécution des requêtes
- L'optimisation de l'exécution des requêtes doit être assurée de manière automatique par le SGBD

C'est le rôle des optimiseurs intégrés

Mais le DBA et le développeur ont aussi leur mot à dire!!

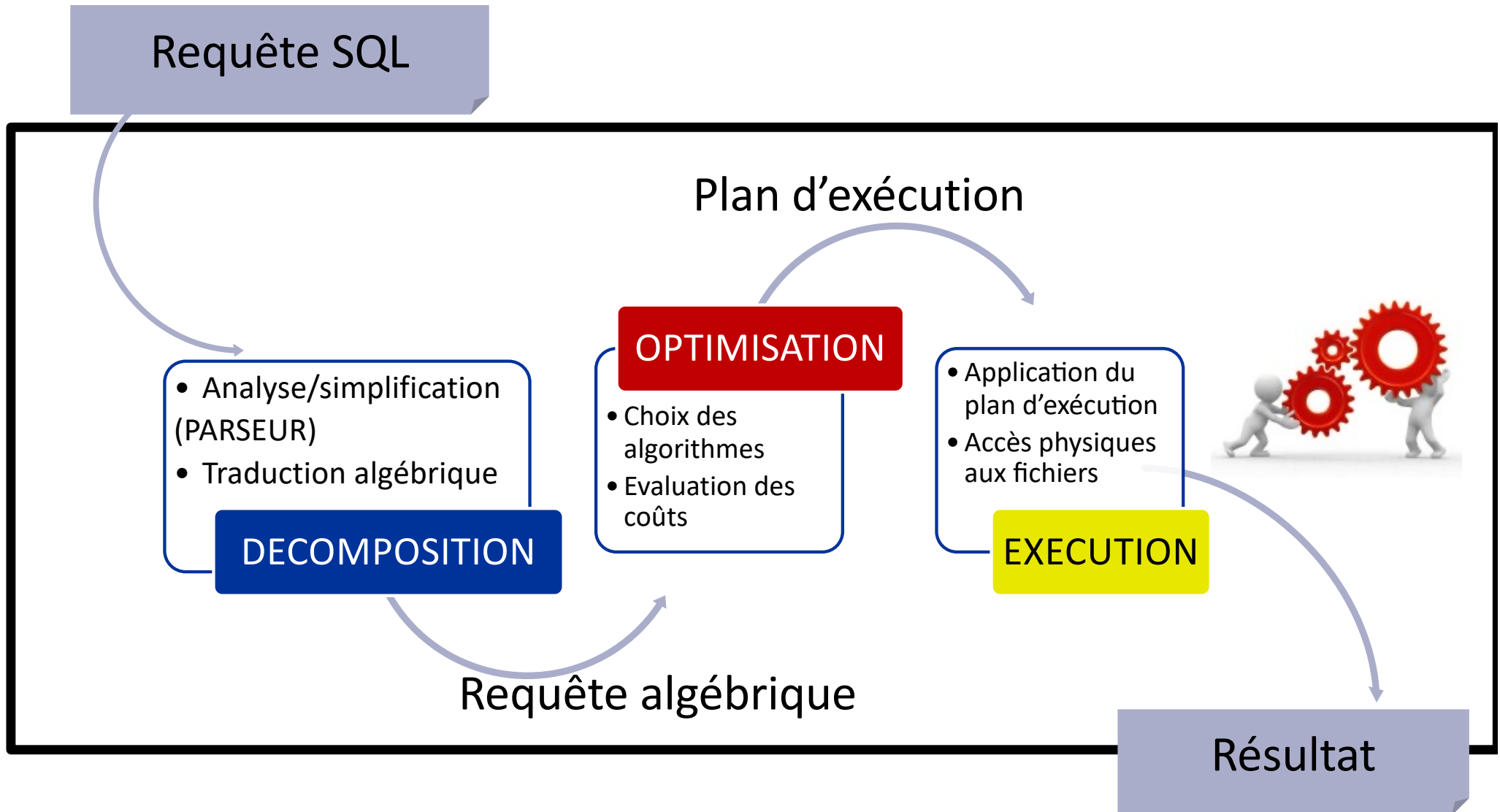
Le rôle du DBA se situe au-delà du principe d'indépendance logique

- Il établit des diagnostics:
 - En examinant les algorithmes choisis par **l'optimiseur** pour l'exécution des requêtes
 - En étudiant les **statistiques** liées à l'exécution (mesures de performances)
- Il améliore les performances:
 - Directement en influençant l'optimiseur
 - Indirectement en modifiant le paramétrage et la structure physique de la BD

SQL Tuning

Database Tuning

Optimisation automatique et Etapes d'exécution d'une requête



Optimiseurs Oracle et PostgreSQL

ORACLE dispose de deux optimiseurs: RBO et CBO

Fortement déconseillé
depuis Oracle V9i

- **RBO (Rule Based Optimizer)**

- Se base sur les **règles** déjà définies par Oracle pour choisir le meilleur plan d'exécution d'un ordre SQL.

- **CBO (Cost Based Optimizer)**

Le seul sous
PostgreSQL

- Se base sur le **coût** pour déterminer le meilleur plan d'exécution
- Le coût d'une requête n'est qu'une estimation qui n'a pas d'unité

Fonctions principales de l'optimiseur CBO

- Il évalue les expressions et les conditions
- Il utilise les statistiques objet et système
 - taille des tables notamment
- Il décide du **mode d'accès** aux données
- Il décide de **l'algorithme de jointure** des tables
- Il décide du chemin le plus efficace
- *Il prend en compte les conseils (**hint**) fournis par le développeur*

HINT non disponibles sous
PostgreSQL

Outils liés à l'optimisation des requêtes par le DBA

- Génération et visualisation des plans d'exécution
- Production de statistiques
- Mesure des temps d'exécution
- *Définition de conseils (Hints)*

On parle d'outils de
TUNING SQL

Plus d'outils dans la version Entreprise ORACLE:

- SQL Profile (depuis version 10g)
- SQL Plan Management (depuis version 11g)

Etapes d'une démarche d'optimisation de requête

- Générer un plan d'exécution
- Comprendre et bien interpréter le plan d'exécution
- Afficher les statistiques des instructions SQL
- Comprendre le mécanisme des différents types de jointures
- Ajouter éventuellement
 - des **index**
 - des « **conseils** »(**HINT**) dans les instructions SQL



Générer et Collecter des statistiques

Statistiques

- Le SGBD permet de générer des statistiques:
 - sur les différents objets de la base de données : tables, index,...
 - sur le fonctionnement général du système
- Les statistiques sont accessibles à travers des vues et tables système.

Cf. Dictionnaire de Données

- **Outils associés:**

- Utilisation de l'interface pgAdmin
- Commande **Analyze**

ANALYSE;

ou ANALYZE nom d'une table.



Mise à jour des
statistiques

L'optimiseur CBO utilise les statistiques pour établir les plans d'exécution

Statistiques sur les tables et colonnes

- **Volumétrie:**
 - Nombre de lignes
 - Nombre de blocs (regroupement de lignes au niveau du stockage)
- **Pour chaque colonne:**
 - nombre de valeurs distinctes,
 - nombre de valeurs NULL
 - valeurs les plus fréquentes

Principales vues système pour accéder aux statistiques

■ Statistiques sur les Tables

Vue système : pg_class

- **relname** : Nom de la table
- **reltuples** : Estimation du nombre de lignes dans la table
- **relpages** : Nombre de pages (blocs) utilisées pour stocker la table

Ne pas oublier
ANALYSE
au préalable

```
SELECT
    relname AS table_name,
    reltuples AS number_of_rows,
    relpages AS number_of_blocks
FROM
    pg_class
WHERE
    relname = 'croisieres';
```

Principales vues système pour accéder aux statistiques

■ Statistiques sur les Colonnes

Vue système : pg_stats

- **attname** : Nom de la colonne.
- **null_frac** : Fraction des valeurs NULL dans la colonne.
- **n_distinct** : Nombre de valeurs distinctes dans la colonne.
 - 1 signifie que toutes les valeurs sont uniques
- **most_common_vals** : Les valeurs les plus fréquentes dans la colonne
- **histogram_bounds** : Limites de l'histogramme pour la colonne, qui indique la distribution des données

Statistiques sur les colonnes

```
SELECT
    attname AS column_name,
    null_frac AS fraction_of_nulls,
    n_distinct AS number_of_distinct_values,
    most_common_vals AS most_common_values,
    histogram_bounds,
FROM
    pg_stats
WHERE
    tablename = 'skippers';
```

column_name name	fraction_of_nulls real	number_of_distinct_values real	most_common_values anyarray	most_common_frequencies real[]	histogram_bounds anyarray
sknum	0	-1	[null]	[null]	{1,2,3,4}
sknom	0	-1	[null]	[null]	{JEAN,LAURA,PAUL,PIERRE}
skport	0	-0.5	{AJACCIO}	{0.75}	[null]
salaire	0	-0.75	{2500}	{0.5}	{2535,3000}

Principales vues système pour accéder aux statistiques

Statistiques sur les Index

Vue système :

- pg_index
- pg_stat_user_indexes
- pg_statio_user_indexes

Nombre de tuples récupérés
à partir de l'index

```
SELECT
    c.relname AS table_name,
    i.relname AS index_name,
    idx_stat.idx_scan AS number_of_scans,
    idx_stat.idx_tup_read AS tuples_read,
    idx_stat.idx_tup_fetch AS tuples_fetched,
    idx_io.idx_blks_read AS index_blocks_read,
    idx_io.idx_blks_hit AS index_blocks_hit
FROM
    pg_class c
JOIN
    pg_index ix ON c.oid = ix.indrelid
JOIN
    pg_class i ON i.oid = ix.indexrelid
LEFT JOIN
    pg_stat_user_indexes idx_stat ON i.oid =
    idx_stat.indexrelid
LEFT JOIN
    pg_statio_user_indexes idx_io ON i.oid =
    idx_io.indexrelid
WHERE
    c.relname = 'croisieres';
```

table_name name	index_name name	number_of_scans bigint	tuples_read bigint	tuples_fetched bigint	index_blocks_read bigint	index_blocks_hit bigint
croisieres	idx_croisieres_batnum	6	6	0	4	9
croisieres	idx_croisieres_sknum	10	10	0	4	17
croisieres	croisieres_pkey	2	2	2	1	9616

Nombre de blocs trouvés en cache

Exercice 2



Visualisez les statistiques sur les tables SKIPPERs, CROISIERES et BATEAUX

Donnez le nombre de lignes et le nombre de blocs de chacune de ces tables dans les deux versions de la BD croisieres :

- BDcroisieresCours
- BDcroisieresBig



Plan d'exécution d'une requête



UNIVERSITÀ DI CORSICA
PASQUALE PAOLI

Qu'est-ce qu'un plan d'exécution?

- Un plan d'exécution est un **arbre** représentant **l'ordre d'exécution** des opérations internes à une requête :
 - Nœud= opérations
 - Arêtes =Flux de données

Généré par l'optimiseur
CBO avant d'exécuter
une requete

Comment visualiser un plan d'exécution?

- Commandes EXPLAIN avec différentes options
- Visualisation directe (interface pgAdmin)

solution 1

solution 2

Que contient un plan d'exécution?

- **Operation** : type d'opération utilisée
 - ex Seq Scan, hash Join, Filter, ...
- **Name** : Nom de la table utilisée
- **Cost**: coût en %CPU estimé par l'optimiseur
- **Rows** : Le nombre de lignes concernées
- **Planning time** : temps nécessaire au système pour planifier et optimiser l'exécution de la requête.
- **Execution time** : temps nécessaire à l'exécution de la requete

Description d'une opération dans un plan d'exécution

Le plan d'exécution décrit chaque opération sur un objet (table, index, ect...) à travers :

- L'ordre des opérateurs exécutés
- Le **mode d'accès** aux objets consultés (table, index, ...)
- Les **opérations physiques** exécutées (**algorithmes** spécifiques)

Commande **EXPLAIN**

solution 1

- Génère et affiche le plan d'exécution défini par l'optimiseur
- N'exécute pas l'ordre SQL
- Syntaxe : **EXPLAIN** *Instruction_SQL*

EXPLAIN

```
SELECT SKNUM, SKNOM  
FROM SKIPPERS  
WHERE SALAIRE >2000;
```

QUERY PLAN

text



Seq Scan on skippers (cost=0.00..13.38 rows=90 width=142)

Filter: (salaire > 2000)

Commande EXPLAIN **VERBOSE**

solution 1

- Affiche le plan d'exécution défini par l'optimiseur avec plus de détails
- N'exécute pas l'ordre SQL
- Syntaxe : **EXPLAIN VERBOSE** *Instruction_SQL*

EXPLAIN VERBOSE

```
SELECT SKNUM, SKNOM  
FROM SKIPPERS  
WHERE SALAIRE > 2000
```

QUERY PLAN

text



Seq Scan on public.skippers (cost=0.00..13.38 rows=90 width=142)

Output: sknum, sknom

Filter: (skippers.salaire > 2000)

Commande EXPLAIN ANALYZE

solution 1

- Exécute la requete SQL
- Affiche le plan d'exécution choisi par l'optimiseur avec les statistiques réelles

Syntaxe : **EXPLAIN ANALYZE** *Instruction_SQL*

EXPLAIN ANALYZE

```
SELECT SKNUM, SKNOM  
FROM SKIPPERS  
WHERE SALAIRE >2000;
```

QUERY PLAN

text



Seq Scan on skippers (cost=0.00..13.38 rows=90 width=142) (actual time=0.009..0.011 rows=4 loops=1)

Filter: (salaire > 2000)

Planning Time: 0.037 ms

Execution Time: 0.021 ms

Visualisation dans pgAdmin

Lancement explain

Paramétrage
Analyses

The screenshot shows the pgAdmin interface. At the top, the connection is 'croisieres/postgres@postgres'. Below the toolbar, the 'Requête' tab is active, showing a SQL query:

```
1 SELECT SKNUM, SKNOM
2 FROM SKIPPERS
3 WHERE SALAIRE >2000;
4
5
```

The 'EXPLAIN' button (labeled 'E' and 'F7') is highlighted in the toolbar. A red arrow points from the 'Lancement explain' label to this button. Another red arrow points from the 'Paramétrage Analyses' label to the 'EXPLAIN' menu.

The 'EXPLAIN' menu is open, showing the following options:

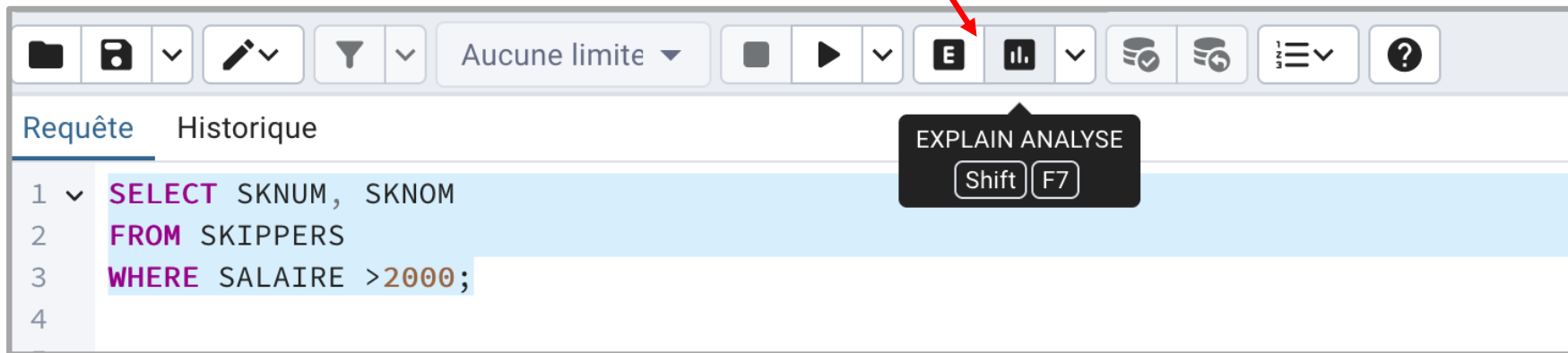
- ✓ Verbeux
- Coût
- Tampon
- Chronométrage
- Synthèse
- Paramètres
- Wal

Below the query editor, the 'EXPLAIN' tab is selected in the 'Data Output' section. The 'Analysis' sub-tab is active, showing the following table:

#	Node
1.	→ Seq Scan on public.skippers as skippers Filtrer: (skippers.salaire > 2000)

Visualisation dans pgAdmin

Lancement Analyses
(exécution effective)



Graphical Analysis Statistics				
#	Node	Chronométrages		
		Exclusif	Inclusif	
1.	Seq Scan on public.skippers as skippers (cost=0..1.38 rows=... Filtrer: (skippers.salaire > '2000'::numeric) Lignes rejetées par filtre (Rows Removed by Filter): 3	0.182 ms	0.182 ms	



Comprendre un plan d'exécution

- Modes d'accès
- Notion d'opération physique

Notion d'opération physique

- Une opération physique est un algorithme réellement **exécuté** lors de l'exécution d'une requête
- Il est choisi par l'optimiseur lors de la phase d'optimisation physique après la mise sous **forme canonique** de la requête.
- On peut classer les opérations par catégorie:
 - Parcours de tables
 - Jointure
 - Tri, agrégations, sélection, projections

Modes d'accès aux tables

■ SEQ SCAN

- Accès **séquentiel** :
- Parcours séquentiel de toutes les lignes

■ INDEX SCAN

- Accès par **index**
- L'index est consulté

- **INDEX Full SCAN**: parcours complet de l'index
- **INDEX Unique SCAN** pour les index uniques et primaires
- **INDEX Range SCAN**: parcours ciblé pour des recherches avec comparateurs (ex: <, >, ...)

pour accéder aux lignes spécifiques de la table qui correspondent aux conditions de la requete

■ INDEX ONLY SCAN

- Accès uniquement à l' **index** car toutes les colonnes nécessaires sont dans l'index

Sélection et projections

- **Sélections**

- **Filter** : Applique un filtre de sortie sur les données extraites (prédicats Where dans la requête SQL)

- **Projections**

- **Output** : liste des colonnes projetées (Select en SQL)

Tris et agrégations

■ Tris

- **Sort Unique** : Renvoie un résultat unique des données extraites
- **Sort Order By** : Renvoie les données extraites triées

■ Aggregate

- **Calculs sommes, count, min, max ...**
- **Sort Aggregate | Sort Group By** : Regroupe les lignes selon les attributs sélectionnés

Jointures

L'optimiseur choisit l'algorithme le plus performant selon les données

- **Boucles imbriquées (Nested / Loop)** : utilise au moins un index sur une colonne de jointure
 - Même s'il y a deux index, les deux ne sont pas obligatoirement utilisés
- **Jointure par Tri Fusion (Sort / Merge)** n'utilise aucun index
- **Jointure par hachage (Hash Join)** : n'utilise aucun index sur les colonnes de jointure
 - Possible si au moins une des tables tient en mémoire

Mais il peut être utilisé même s'il y a des index

Jointure par boucles imbriquées

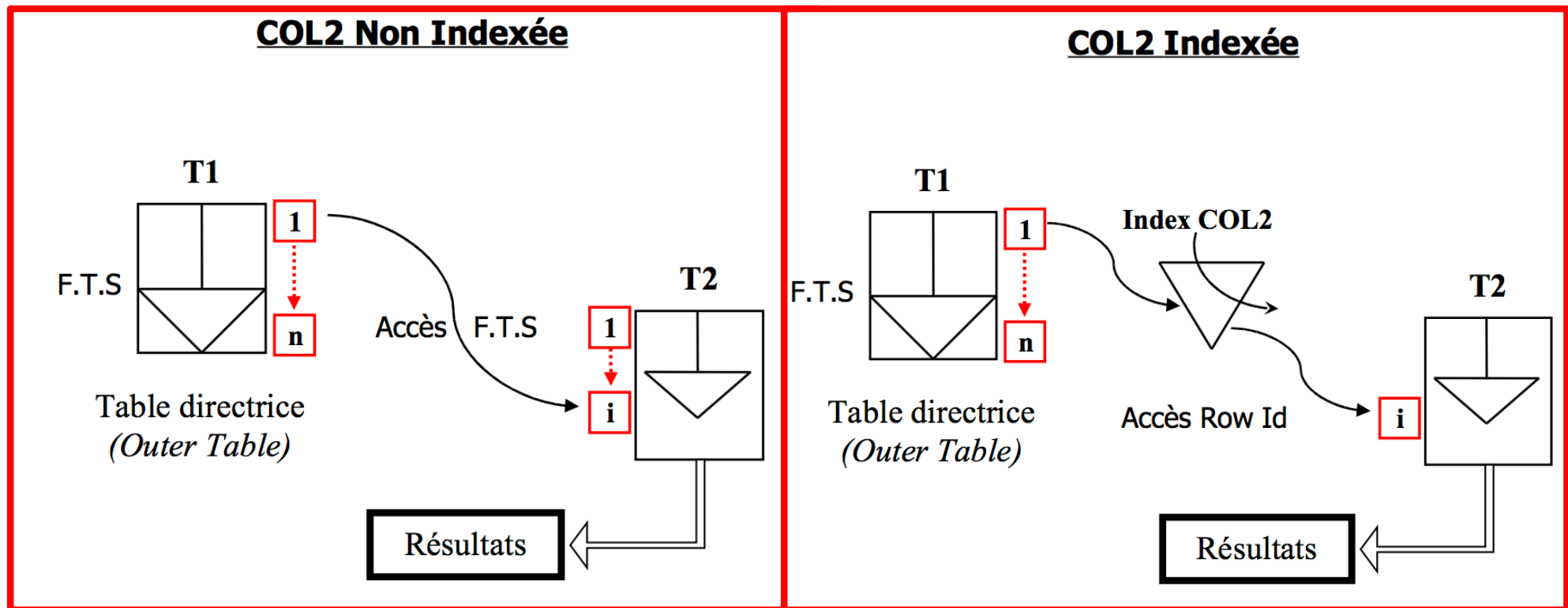
Nested / Loop

index sur attribut de
jointure

OU

SELECT * FROM T1,T2 WHERE T1.COL1=T2.COL2

SELECT * FROM T1 JOIN T2 ON T1.COL1=T2.COL2



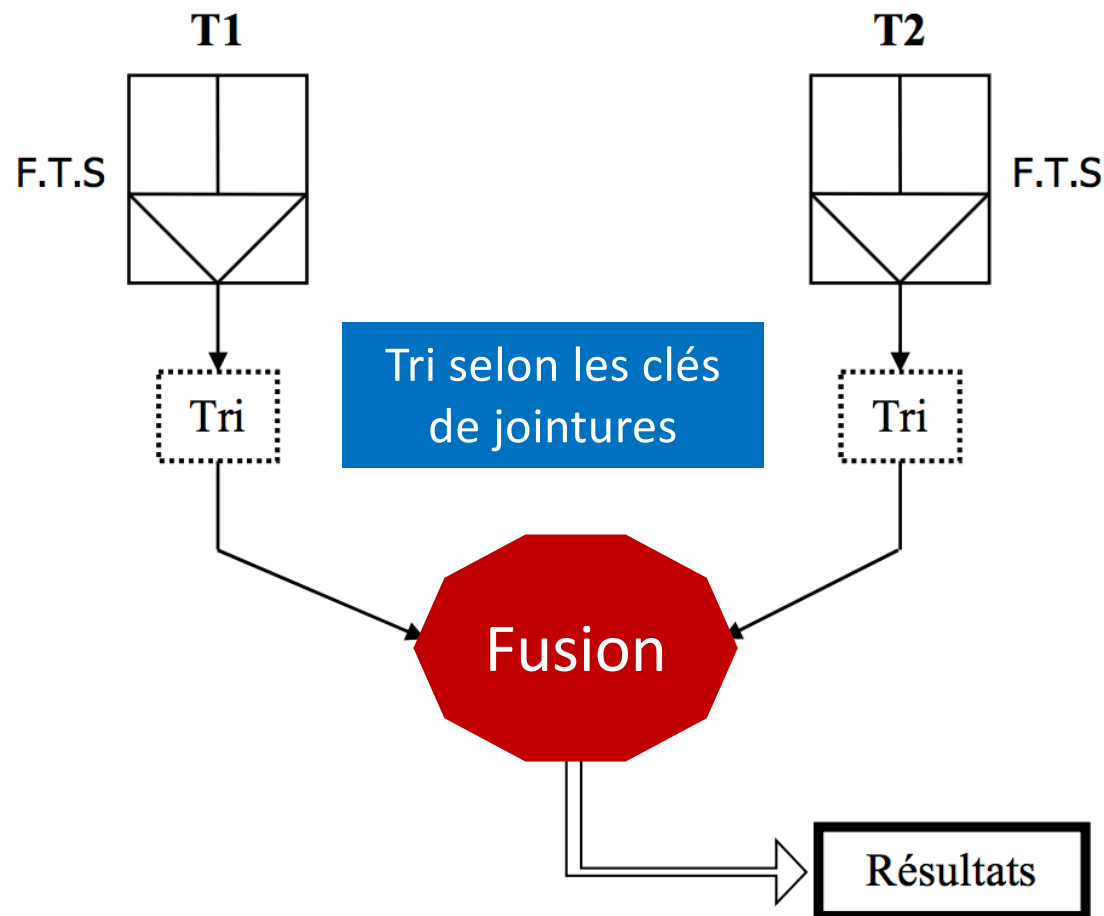
FTS = Full Table Scan = Parcours séquentiel

Jointure par tri fusion

Sort / Merge

Aucun index

SELECT * FROM T1,T2 WHERE T1.COL1=T2.COL2



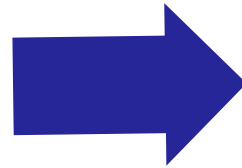
FTS: Full
Table Scan
*Parcours
séquentiel*

Jointure par table de hachage

Hash Join

Build Phase

- Construction **en mémoire** d'une table de hachage à partir de la table la plus petite (**Inner Table**)



Probe Phase

- Parcours des tuples de la 2^{ème} table (**Outer table**) et recherche de correspondances dans la table de hachage

Utilisable uniquement pour les équi-jointures (=)

Jointure par Hash Join Exemple

SELECT name, subject, score FROM student st INNER JOIN score s ON st.id = s.stu_id

Table student

id	name	age
10	Jack	25
13	Ariel	25
12	Tom	26

Inner Table

Table score

id	stu_id	subject	score
1	10	math	95
2	10	history	98
3	12	math	97
4	15	history	92

Outer Table

Jointure par Hash Join

Phase 1 : Build Phase

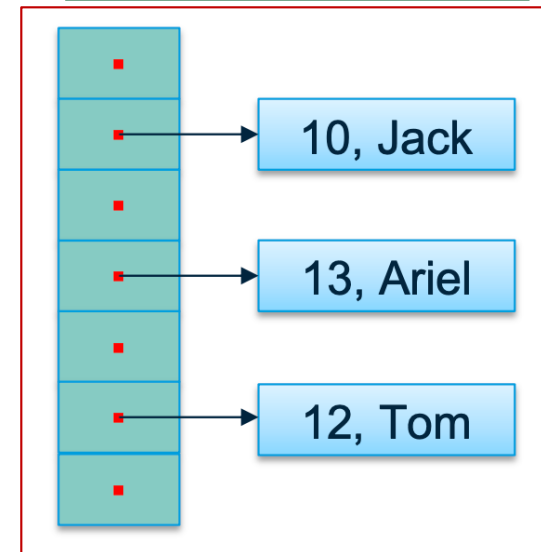
Construction de la table de hachage

id	name	age
10	Jack	25
13	Ariel	25
12	Tom	26

Inner Table



Hash Table



clé de hachage = student.id


```
SELECT name, subject, score FROM student st INNER JOIN score s ON st.id = s.stu_id
```

Jointure par Hash Join

Phase 2 : Probe Phase

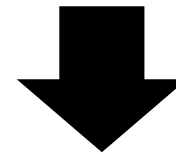
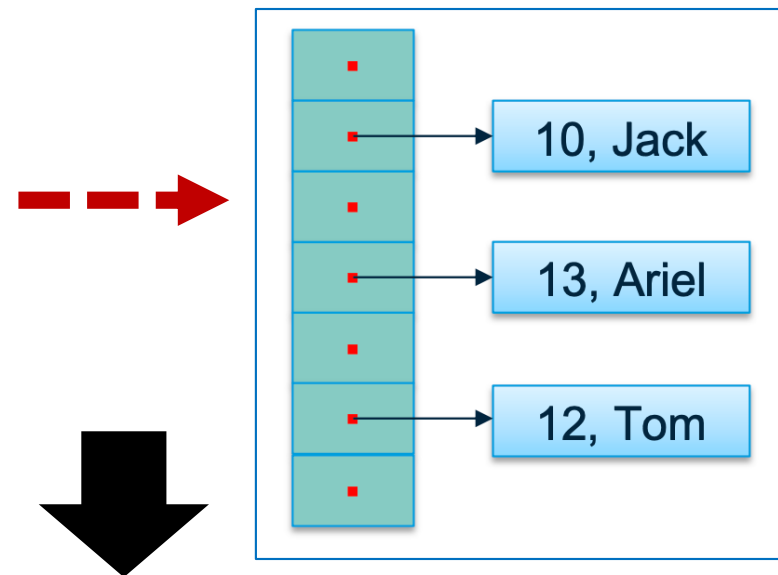
Recherche de correspondances dans la hash table

Table score



id	stu_id	subject	score
1	10	math	95
2	10	history	98
3	12	math	97
4	15	history	92

Outer Table



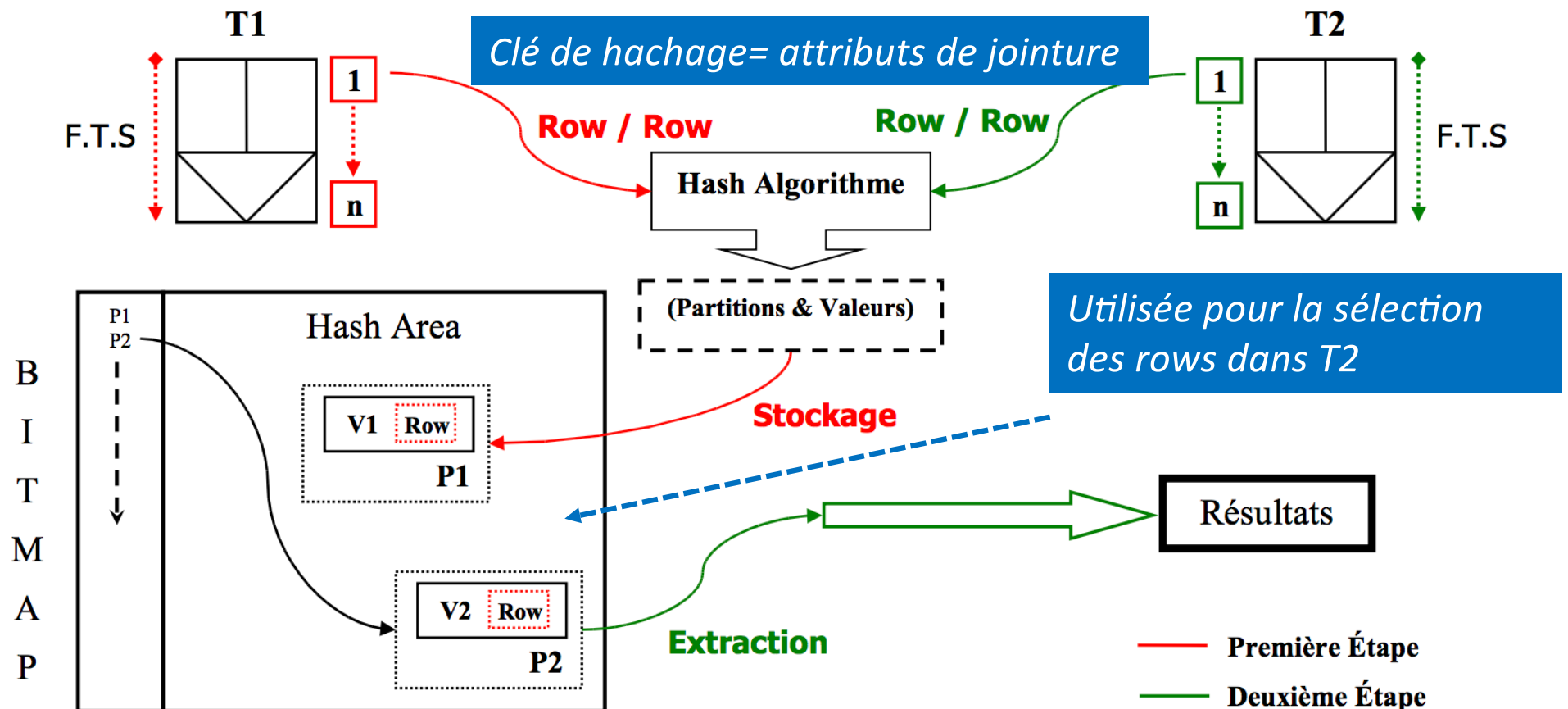
Jack	math	95
Jack	hist	98
Tom	math	97

Jointure par table de hachage

Hash Join

Table de hachage en mémoire

```
SELECT * FROM T1,T2 WHERE T1.COL1=T2.COL2
```



Jointures et indexation

La définition d'index est une action essentielle pour améliorer les performances d'une jointure mais cela dépend de l'algorithme utilisé

- **Nested Join:**

- Indexation sur les colonnes de jointure

- **Hash-Join:**

- Indexation sur les colonnes impliquées dans les prédicats where *indépendants*.

Dans le cas d'une jointure par hachage, les index sur les colonnes de jointure ne sont pas utilisés



Analyse de l'exécution de requêtes

- Comprendre comment fonctionne l'optimiseur CBO

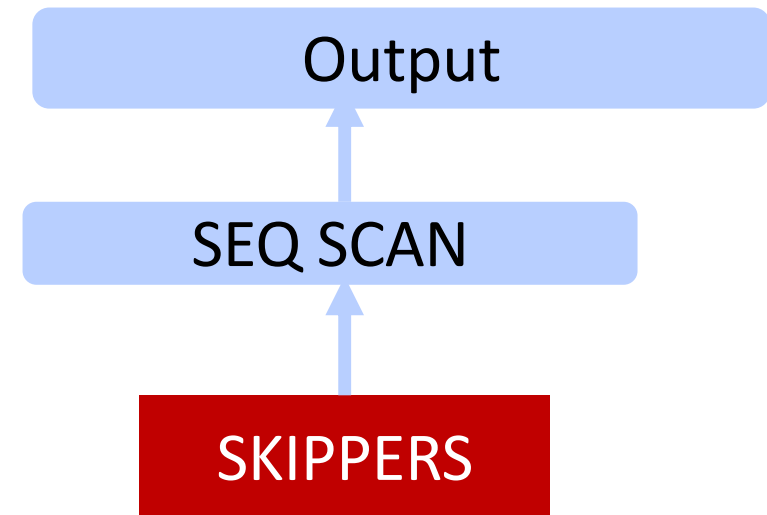


SELECTION SANS INDEX


//création du plan d'exécution

EXPLAIN VERBOSE

```
SELECT *  
FROM SKIPPERS  
WHERE SKPORT='AJACCIO';
```



PLAN d'EXECUTION

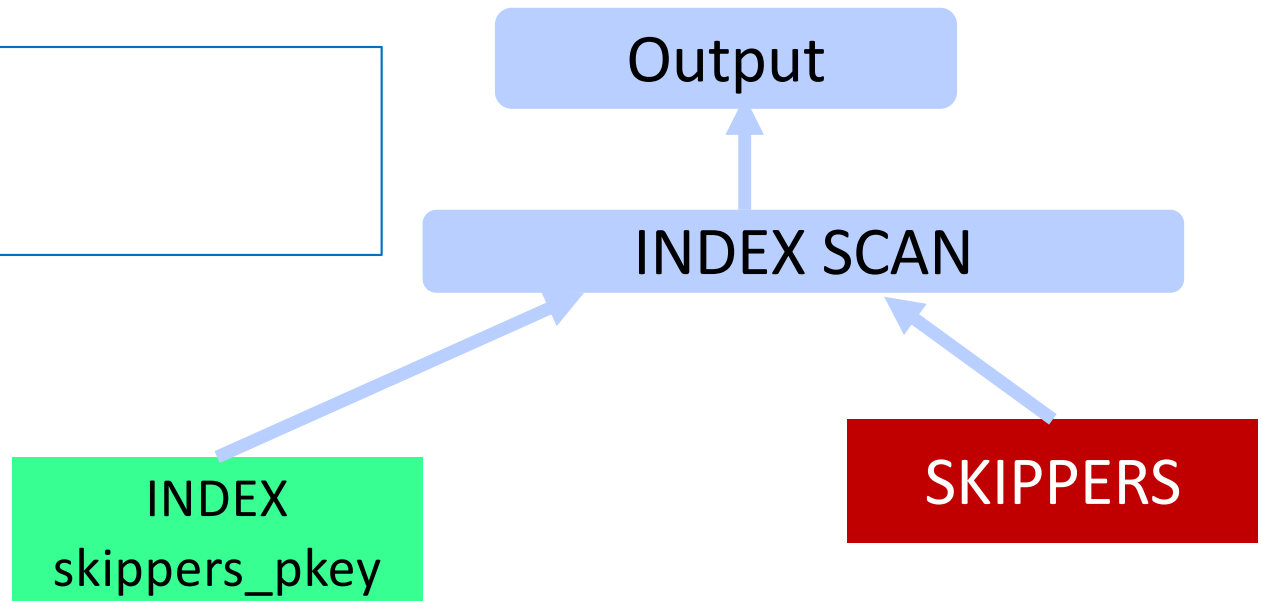
QUERY PLAN		
text		
Seq Scan on public.skippers (cost=0.00..1.38 rows=1 width=3...		
Output: sknum, sknom, skport, salaire	Projection	
Filter: ((skippers.skport)::text = 'AJACCIO'::text)		Sélection sur


Sélection sur la
table

SELECTION AVEC INDEX

```
SELECT *  
FROM SKIPPERS  
WHERE SKNUM=6
```

PLAN d'EXECUTION



QUERY PLAN	
text	
Index Scan using skippers_pkey on public.skippers (cost=0.14..8.15 rows=1 width=31)	
Output: sknum, sknom, skport, salaire	
Index Cond: (skippers.sknum = 6)	

Selection sur l'index clé primaire ici

Exercice 3



On considère les trois requêtes suivantes:

1. `SELECT SKNOM FROM SKIPPERS ORDER BY SALAIRE;`
2. `SELECT MIN(SALAIRE) FROM SKIPPERS;`
3. `SELECT DISTINCT SALAIRE FROM SKIPPERS;`

- a) Donnez le plan d'exécution de chacune de ces requêtes (liste des modes d'accès et opérateurs)
- b) Ajoutez un index sur le salaire dans SKIPPERS et Générez à nouveau les plans d'exécution pour les requêtes 1 à 3.

Quels sont les plans d'exécution modifiés?

Exercice 3



Récupérez le script de création de la base de données croisières (version plus de 1000 lignes) :

BDcroisieresBig.sql.

Testez à nouveau les plans d'exécution des requêtes précédentes (avec et sans index sur salaire)

1. SELECT SKNOM FROM SKIPPERS ORDER BY SALAIRE;
2. SELECT MIN(SALAIRE) FROM SKIPPERS;
3. SELECT DIS TINCT SALAIRE FROM SKIPPERS;

Exercice 4



On suppose que les seuls index définis dans la base de données portent sur les clés primaires.

On considère la requête suivante :

```
SELECT BATNOM FROM BATEAUX WHERE  
BATPORT='Nice';
```

supprimer les index
créés dans les exercices
précédents si nécessaire

- a) Un index est-il utilisé pour l'exécution de cette requête?
- b) La table bateaux est-elle parcourue séquentiellement lors de l'exécution de cette requête?
- c) Créez un index sur BATPORT
- d) Est-ce que la création de cet index améliore les performances de la requête?

JOINTURE

EXPLAIN VERBOSE

```
SELECT SKNOM,SALAIRE  
FROM SKIPPERS,CROISIERES  
WHERE SKIPPERS.SKNUM=CROISIERES.SKNUM;
```

PLAN d'EXECUTION

QUERY PLAN

text

Hash Join (cost=1.68..3.32 rows=50 width=19)

Output: skippers.sknom, skippers.salaire

Inner Unique: true

Hash Cond: (croisieres.sknum = skippers.sknum)

-> Seq Scan on public.croisieres (cost=0.00..1.50 rows=50 width=4)

Output: croisieres.croisnum, croisieres.depport, croisieres.arrport, croisieres.deppdate, croisieres.arrdate, croisieres.batnum, croisieres.sk...

-> Hash (cost=1.30..1.30 rows=30 width=23)

Output: skippers.sknom, skippers.salaire, skippers.sknum

-> Seq Scan on public.skippers (cost=0.00..1.30 rows=30 width=23)

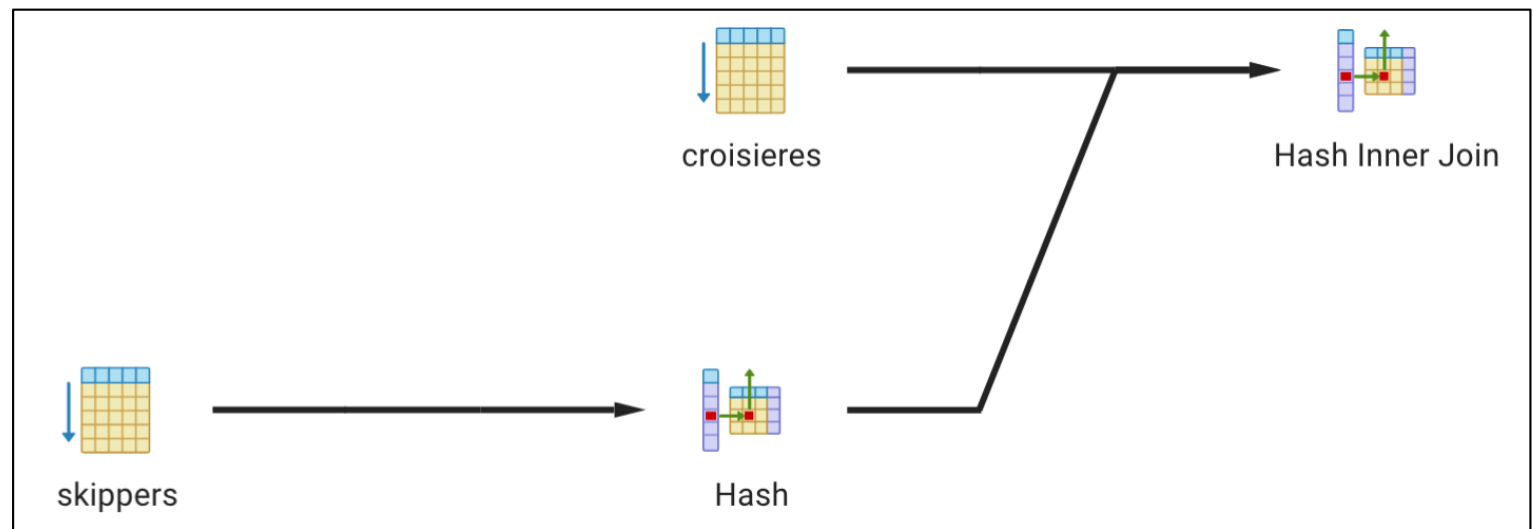
Output: skippers.sknom, skippers.salaire, skippers.sknum

Cout total estimé de la requête

- de 1,68 (coût de démarrage)
- à 3,32 (coût total)

Le plan d'exécution est-il identique avec d'autres écritures de la jointure ? A vérifier (cf Exercice)

JOINTURE



SKNOM, SALAIRE

HASH INNER JOIN

SKNUM,SKNOM,SALAIRE

HASH
Construction table de
hachage

SKNUM,SKNOM,SALAIRE

SEQ SCAN
Parcours séquentiel

SKIPPERS

CROISNUM, ... SKNUM ...

SEQ SCAN
Parcours séquentiel

CROISIERES

JOINTURE AVEC INDEX

EXPLAIN VERBOSE

```
SELECT SKNOM,SALAIRE  
FROM SKIPPERS,CROISIERES  
WHERE SKIPPERS.SKNUM=CROISIERES.SKNUM;
```

- Création d'un index sur SKNUM dans CROISIERE
 - CREATE INDEX fk_croisieres_sknum ON croisieres(sknum);
- Impact sur le plan d'exécution?
 - Aucun !!
 - Comment expliquer cela?

Vérifiez

JOINTURE AVEC INDEX

- Pour se convaincre de la pertinence de l'optimiseur
 1. Désactivez temporairement l'utilisation des scans séquentiels (Seq Scans)

`SET enable_seqscan = OFF;`

2. Régénérez le plan d'exécution.

Que remarquez-vous?

Quel coût obtenez-vous?

Pour annuler la désactivation
`SET enable_seqscan = ON;`

JOINTURE AVEC INDEX

```
SELECT SKNOM,SALAIRE  
FROM SKIPPERS,CROISIERES  
WHERE SKIPPERS.SKNUM=CROISIERES.SKNUM;
```

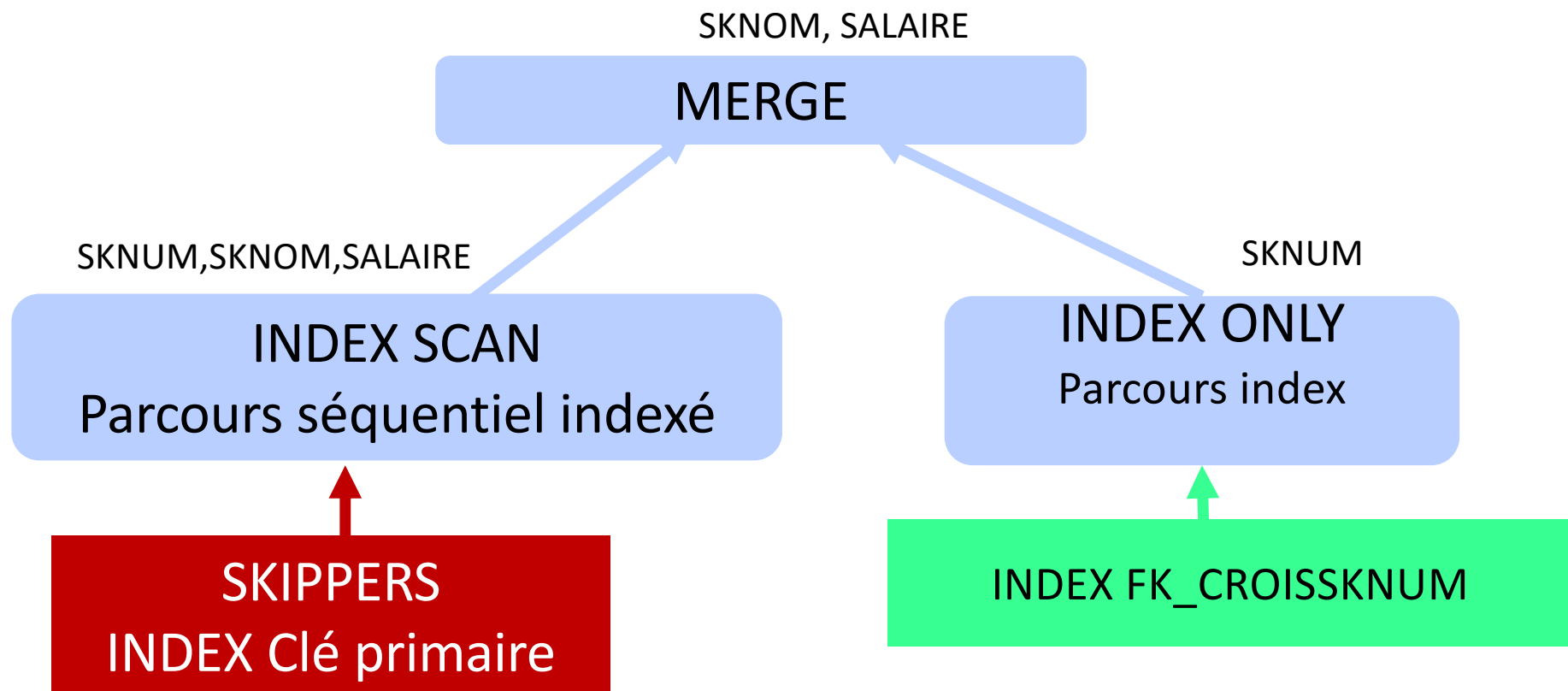
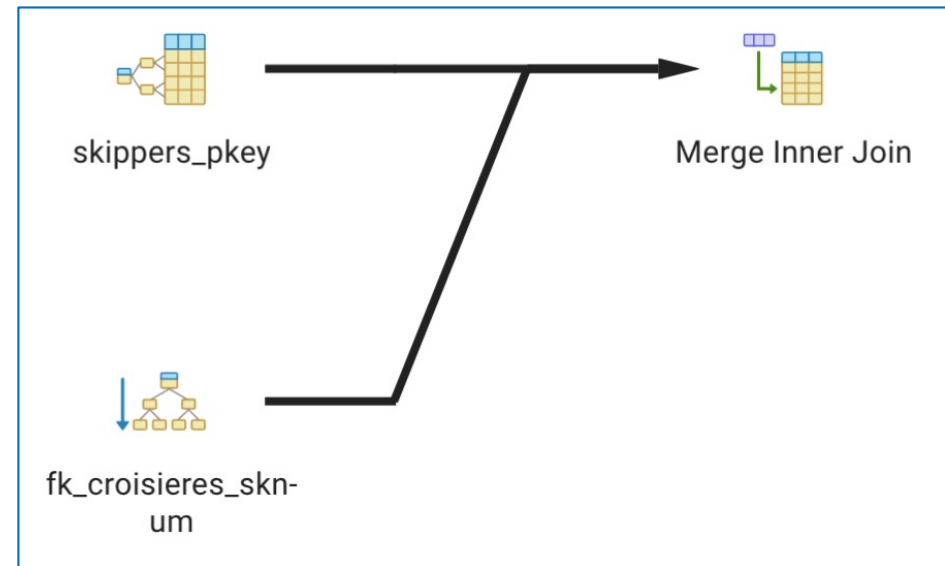
Utilisation de SET
enable_seqscan = OFF;
pour forcer le changement
d'algo

QUERY PLAN
text
Merge Join (cost=0.28..25.98 rows=50 width=19)
Output: skippers.sknom, skippers.salaire
Merge Cond: (skippers.sknum = croisieres.sknum)
-> Index Scan using skippers_pkey on public.skippers (cost=0.14..12.59 rows=30 width=23)
Output: skippers.sknum, skippers.sknom, skippers.skport, skippers.salaire
-> Index Only Scan using fk_croisieres_sknum on public.croisieres (cost=0.14..12.89 rows=50 width=...
Output: croisieres.sknum

Cout total estimé de la requête

- de 0,28 (coût de démarrage)
- à 25,98 (coût total)

JOINTURE AVEC INDEX



Ici le parcours de l'index est suffisant car seul SKNUM est utile

Bilan : Utilisation des index par l'optimiseur

- Non systématique
- Uniquement si le cout estimé est intéressant
- Quels sont les cas où le parcours par index est souvent moins avantageux qu'un parcours séquentiel ?

Petites tables
(moins de 1000 lignes)

Nécessité d'accès à la
totalité ou une grande
partie des lignes

Exercice 5



- Visualisez les plans d'exécution avec les autres écritures possibles (INNER JOIN, NATURAL JOIN, sous-requete IN) de la jointure dans la requête :

```
SELECT DISTINCT SKNOM,SALAIRE  
FROM SKIPPERS,CROISIERES  
WHERE SKIPPERS.SKNUM=CROISIERES.SKNUM;
```

- Le mode d'écriture de la jointure a-t-il un impact sur le plan d'exécution?

Exercice 6



On considère la requête suivante en supposant qu'un index sur salaire a été défini dans la table SKIPPERS:

```
SELECT SKNOM FROM SKIPPERS WHERE  
SALAIRE >= 2000;
```

L'optimiseur a choisi de ne pas utiliser l'index et on souhaite comprendre pourquoi.

- Forcez l'utilisation de l'index sur le salaire en désactivant le SEQ SCAN et comparez le cout obtenu avec celui que propose initialement l'optimiseur

Exercice 7

On souhaite mettre en évidence l'impact des index sur les performances des mises à jour.

1. On considère la requete Req1 suivante :

```
UPDATE croisieres  
SET depart='Nice'  
WHERE depart='Ajaccio';
```



2. Lancez l'analyse de l'exécution et notez les temps d'exécution.
3. Exécutez la requête inverse *Req1Undo* pour annuler les modifications

```
UPDATE croisieres  
SET depart='Ajaccio'  
WHERE depart='Nice';
```

Exercice 7



4. Créez un index sur depport dans Croisieres
5. Lancez à nouveau l'analyse de la requete 1 et notez le temps d'exécution.

Que constatez-vous?

Exercice 8



Reprenez vos solutions de l'exercice 2 du CH2 :

Quels sont les noms des skippers qui assurent une ou plusieurs croisières sur un bateau ayant une capacité inférieure à celle de tous les bateaux localisés à Ajaccio?

- version avec une sous-requete utilisant la clause IN
 - version avec une sous-requete utilisant la clause Not exists
 - version avec Group by et Having
1. Visualisez les plans d'exécution des 3 versions sans index.
 2. Quelle est la solution la plus performante?
 3. Essayez d'améliorer en créant des index.



Bilan: Bonnes pratiques dans l'écriture des requêtes

SQL Tuning



UNIVERSITÀ DI CORSICA
PASQUALE PAOLI

Qu'est ce que le SQL Tuning?

- Les optimiseurs intégrés dans les SGBD sont (de plus en plus) aptes à optimiser les performances des requêtes quelque soit leur écriture.
- Néanmoins, le développeur peut grâce au « SQL tuning »
 - leur faciliter la tâche, en améliorant l'écriture de ses requêtes
 - les influencer en insérant des *hints* (conseils) dans les requêtes

Optimisation de l'écriture des requêtes

- Les choix opérés lors de la construction d'une requête peuvent avoir un impact significatif sur les performances :
 - Proposition d'une liste de recommandations
- Deux réserves cependant:
 - Les conseils doivent être **validés par des tests** car les résultats peuvent varier selon le SGBD et selon la taille des tables.

Analyse des plans d'exécution – visualisation statistiques


- Certains conseils peuvent s'avérer **inutiles** car les optimiseurs automatiques sont de plus en plus performants.

Bonnes pratiques (1)

- Définition des projections
(**clause Select**)
 - Eviter d'utiliser Select *
 - Préférer count(1) à count(*)

EX: select 1 from Bateau

*Renvoie une table avec juste la valeur de la constante 1
avec autant de lignes que de lignes dans la table
UTILE SI SEUL LE NOMBRE DE LIGNE NOUS INTERESSE*



1
1
1
1

- Définition des tables impliquées
(**clause From**)
 - Utiliser des alias de tables (très courts)

Bonnes pratiques (2)

- Définition des conditions de sélection (**clause Where**)
 - Eviter les conditions IS NULL, IS NOT NULL
 - Remplacer si possible les connecteurs OR par la définition de requêtes de type UNION
 - Limiter au maximum l'utilisation de l'opérateur LIKE
 - Le remplacer par des conditions OR

Bonnes pratiques (3)

- Définition des **jointures**
 - Privilégier la définition de jointures à l'utilisation de sous-requêtes
 - Pour la définition des jointures utiliser la clause **Innerjoin** de préférence à la solution classique (liste des tables dans le from et condition de jointure dans le where)

Bonnes pratiques (4)

- Sous-requêtes
 - Minimiser le nombre de sous-requêtes
 - Préférer le prédicat EXISTS à IN
- Partitionnements
 - Eviter d'utiliser la clause HAVING
 - Placer les conditions de préférence dans la clause Where qui est évaluée avant les opérations d'agrégation et permet donc de les alléger

Pour information

Quelques notions d'organisation du stockage physique



- Clusters
- Partitionnement de tables

Notion de Cluster

- Permet de regrouper le stockage physique de plusieurs tables :

Stockage dans un meme bloc des attributs de jointure des deux tables

- Tables souvent utilisées ensemble dans des interrogations de jointure
- Il s'agit de stocker physiquement des données sémantiquement liées

Clés de cluster

```
CREATE CLUSTER nom_cluster (CléJoin1 type, CléJoin2 type, ...);
```

```
CREATE TABLE nom_table (
```

Colonnes appartenant au groupement

```
    CLUSTER nom_cluster (Col1, Col2, ...);
```

Notion de Cluster

Avantages

- Améliorer les performances des jointures en minimisant les accès disque
 - La clé primaire (utilisée par la jointure) n'est stockée qu'une fois
- Alternative à la dénormalisation

Inconvénient

- Le balayage complet d'une table clusterisée est plus lent
 - Diminution des performances des requêtes portant sur une table isolée

Les clusters peuvent être indexés pour accélérer les recherches

Partitionnement de tables

- Découpage d'une table en plusieurs tables plus petites afin qu'elle soit moins volumineuse :
 - La table initiale doit pouvoir être reconstruite par jointure ou union des tables de partitionnement
- Deux types de partitionnement:
 - Partitionnement vertical: découpage par colonnes
(Projection)
 - Partitionnement horizontal: découpage par ligne
(Sélection)

Exemple de Partitionnement vertical

- La table SKIPPER peut être partitionnée verticalement en:
 - SKIPPER 1(NUM, NOM, ADRESSE)
 - SKIPPER 2(NUM, SALAIRE)

La table initiale peut être récupérée par JOINTURE des différentes tables sur la clé.

Objectif: Améliorer les performances des requêtes portant sur certains attributs en les isolant

Utile uniquement si le nombre d'attributs est élevé

Exemple de Partitionnement horizontal

- La table SKIPPER peut être partitionnée horizontalement en répartissant les tuples dans plusieurs tables selon des critères de sélection:

- SKIPPER 1: skippers habitant à Ajaccio
- SKIPPER 2: skippers habitant à Bastia

La table initiale peut être récupérée par UNION entre les différentes tables.

Objectif: Améliorer les performances des requêtes sur certains tuples particuliers (ex: skippers de Bastia)

Utile uniquement si le nombre de tuples est élevé

Exercice Bilan



Récupérez sur l'ENT :

- le script de la base de données Banque.sql
- le sujet de l'exercice Bilan
ExerciceBilanCH4IndexOptimisation.pdf

Liens



- Un article très intéressant qui approfondi la notion d'index sur le blog sqlPro
 - [https://blog.developpez.com/sqlpro/p9816/langage-sql-norme/tout sur l index](https://blog.developpez.com/sqlpro/p9816/langage-sql-norme/tout%20sur%20l'index)
- Un guide pour améliorer les performances des requêtes SQL par l'indexation:
 - <http://use-the-index-luke.com/fr/sql/l-operation-de-jointure/jointure-de-hachage>
- Pour en savoir plus sur les différents types d'index et les plans d'exécution
 - <http://cedric.cnam.fr/~rigaux/docs/slindex.pdf>
 - [https://www.irit.fr/~Mohand.Boughanem/slides/BDA/C hap3 Optimisation.pdf](https://www.irit.fr/~Mohand.Boughanem/slides/BDA/C hap3%20Optimisation.pdf)