université
PARIS-SACLAY

DATA WAREHOUSE I

WEEK 6

BASED ON BENOÎT GROZ'S SLIDES

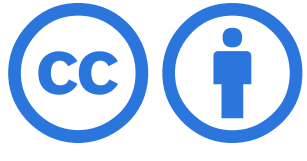


TABLE DES MATIÈRES

INDEXATION

INDEXATION DANS LES SGBD

Objectif des index : accélérer l'accès aux données nécessaires à la requête.

```sql
SELECT * FROM Employee WHERE EmployeeKey = 1234;
-- En utilisant un index sur EmployeeKey: 1 accès disque.
-- Sans index : nécessaire de scanner toute la table Employee.
```

Inconvénient : toute mise à jour d'un attribut indexé impose de mettre à jour l'index.

⇒ maintenir trop d'index aboutit à une dégradation des performances.

TAXONOMIE DES INDEX

- Monocolonne vs multicolonnes
- Groupant (ordonné ou clustering) vs non-groupant :

  Dans un index groupant, les tuples sont physiquement ordonnés selon la clé d'index.

  On a au plus 1 index groupant, mais peut avoir plusieurs index non-groupants.
- Unique vs non-unique : dans un index unique, il existe un seul tuple pour chaque valeur de la clé: un index non-unique permet à plusieurs tuples de partager la même valeur de clé.
- Dense vs épars : un index dense a une entrée pour chaque tuple, un index épars ne référence qu'une partie des enregistrements.

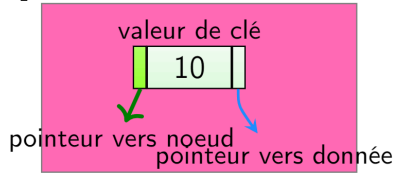
INDEX ET PARTITIONS

- Index local : les partitions de l'index suivent les partitions de la BD (bijections entre les partitions de la table et de l'index). Facilite la maintenance. Le plus courant en entrepôt.
- Index global partitionné : partitions de l'index ne correspondent pas aux partitions de la table. Plutôt OLTP.
- Index global non-partitionné : index non partitionné, surtout utilisé pour assurer les contraintes de clé unique. En entrepôt ce rôle peut être assuré par ETL.

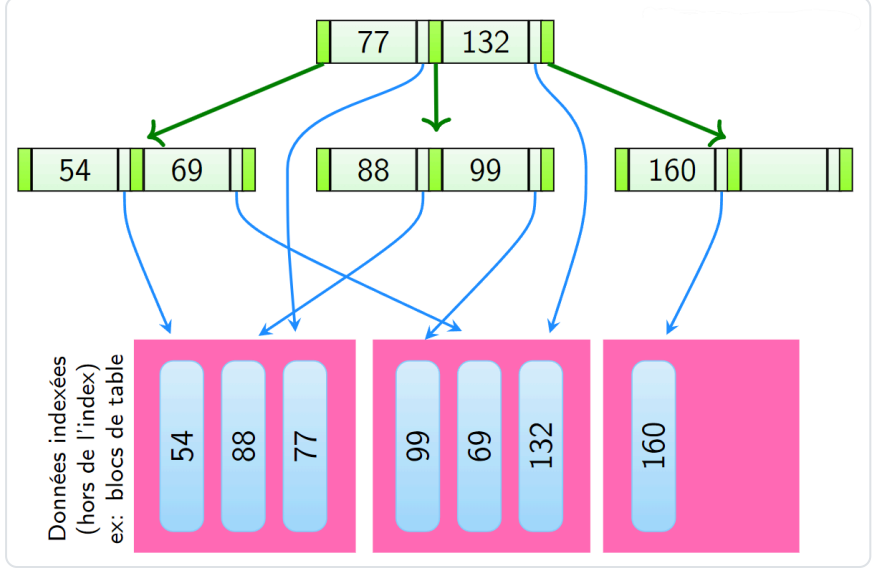
B-TREES

B-TREES

Bayer, McCreight '72



Index typique en BD opérationnelle : B-tree (B+).



Caractéristiques :

- Entre m et $2m$ clés par noeud (ou moins à la racine).
- Les feuilles sont au même niveau (arbre équilibré).

B-TREES EN PRATIQUE

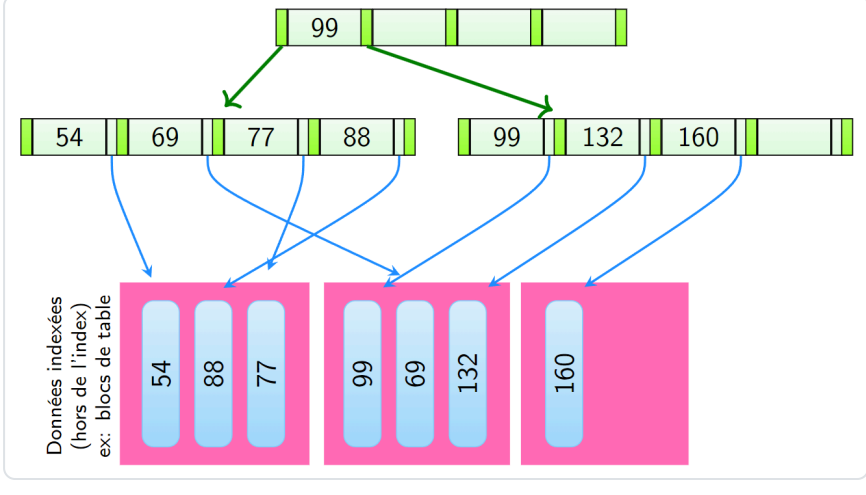
La plupart du temps, un arbre B d'ordre t est par un arbre qui satisfait les propriétés suivantes :

1. Chaque nœud a au plus `2t-1` clés.
2. Chaque nœud qui n'est ni racine ni feuille possède au moins `t-1` clés.
3. Si l'arbre est non vide, la racine est aussi non vide.
4. Un nœud qui possède `k` fils contient `k - 1` clefs.
5. Toutes les feuilles se situent à la même hauteur.

B+-TREES (VARIANTE LA PLUS UTILISÉE)

Bayer, McCreight '72

Comme B-tree, mais les pointeurs vers les données sont uniquement dans les feuilles.



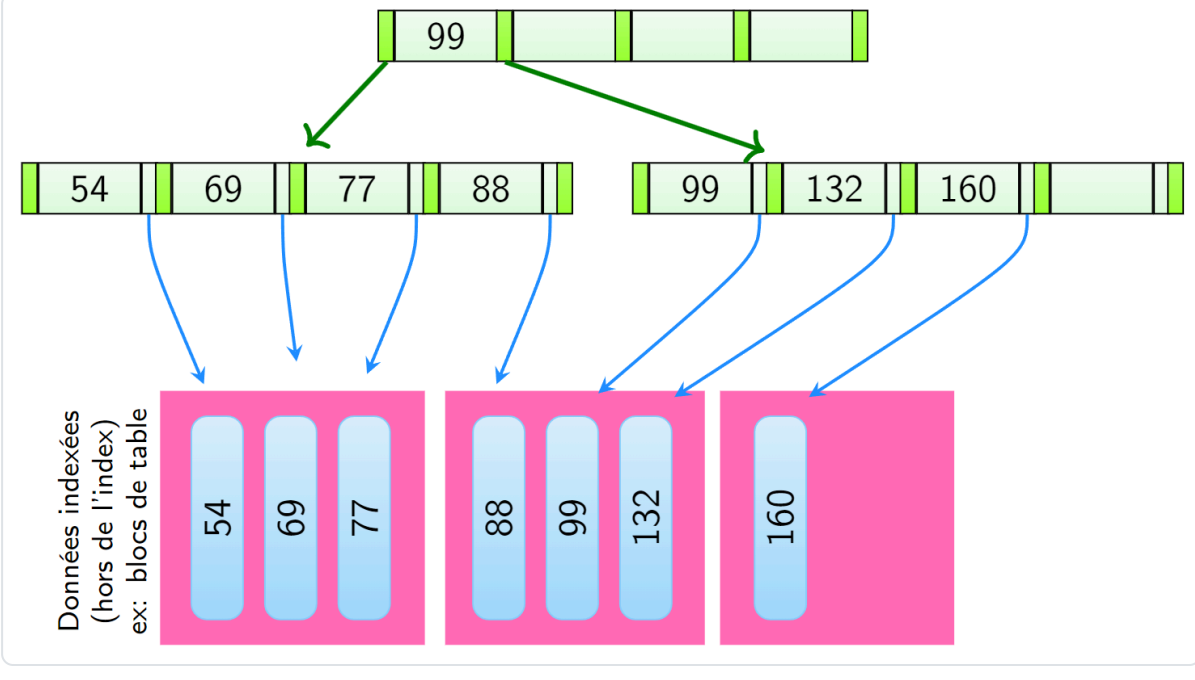
Caractéristiques :

- Les clés sont uniquement dans les feuilles.
- Les feuilles peuvent être liées en une liste chaînée.

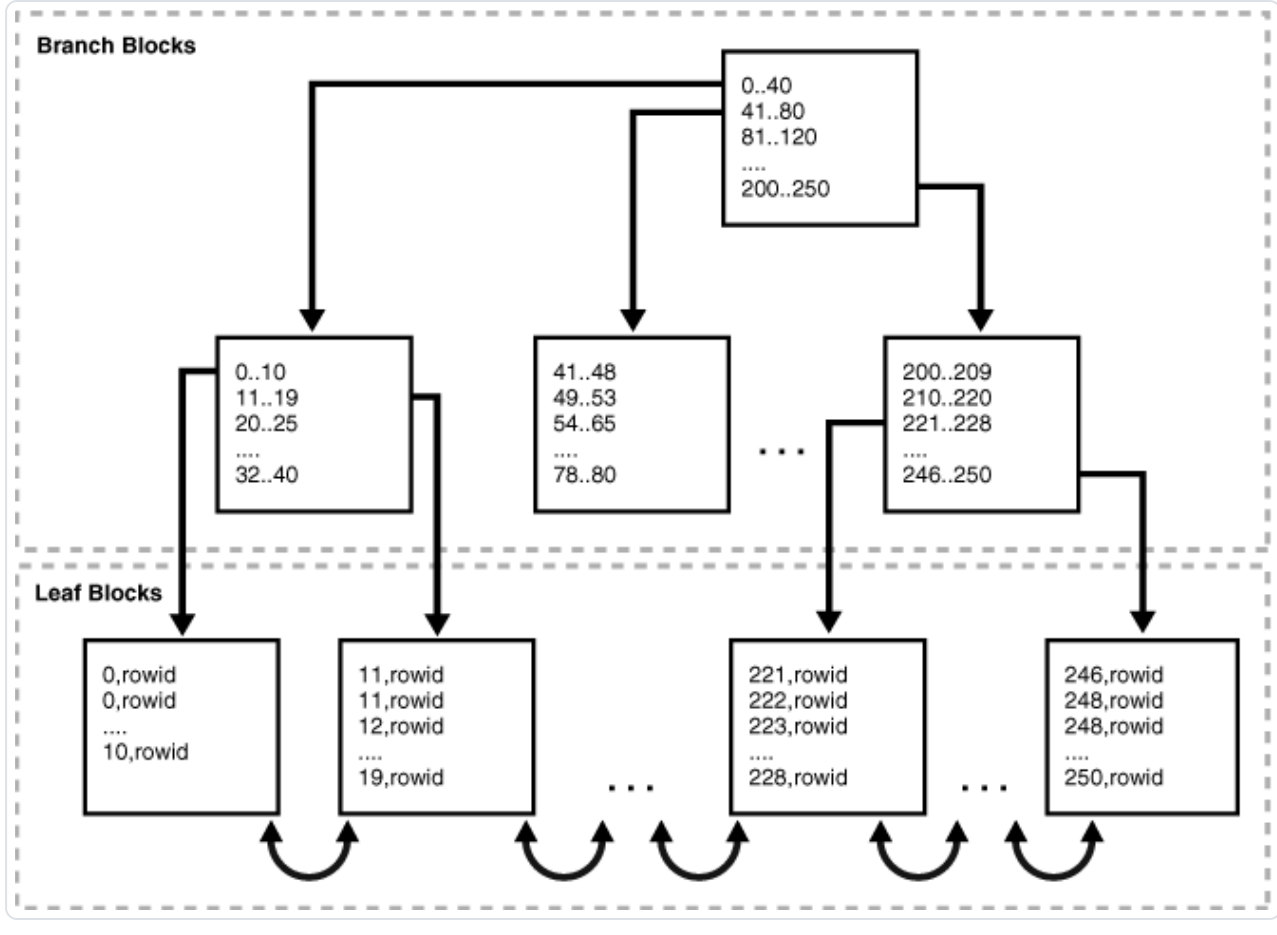
B+-TREES : INDEX GROUPANT (CLUSTERED)

Bayer, McCreight '72

Caractéristique principale : quand la table est triée dans l'ordre de l'index.



B-TREES SOUS ORACLE



Internal Structure of a B-tree Index

De quelle sorte d'index s'agit-il ? B/B+ ? Particularités ?

LES B-TREES SOUS ORACLE: PROPRIÉTÉS

- Généralement, une clé d'index associée à un ROWID.
- Pour un index composite, possibilité d'effectuer un *index skip-scan*.
- *Index clustering factor* : mesure la corrélation entre l'ordre de l'index et la répartition des enregistrements dans les blocs de données. Estime le nombre de blocs à lire pour parcourir toute la table dans l'ordre de l'index.

```
-- Définir un index B-tree
CREATE INDEX cust_gender_adr_idx ON Customers(gender, address);

SELECT * FROM Customers WHERE address = '36 Quai des Orfèvres';
```

B-TREES: PROPRIÉTÉS

Avantages :

- Insertions, mises à jour, suppressions en $O(log_m(n))$.
- Indépendant du type de données (exige seulement un ordre).
- Doit être rééquilibré si modifications des données (pas si grave en DW)
- Structure à 1 dimension: clé = 1 attribut ou suite d'attributs (ordonnée).
- Efficace pour les requêtes d'intervalles et sur clé primaire.

Faiblesses :

- Peu approprié pour les requêtes multidimensionnelles.
- Inefficace sur les attributs de faible cardinalité.

BESOINS D'INDEXATION EN ENTREPÔT

- Utilisation de l'ETL pour appliquer les contraintes UNIQUE.
- Requêtes avec plusieurs filtres dans la clause WHERE et jointures fréquentes entre table de faits et dimensions.

Autres considérations :

- Aggregations sur niveaux de hiérarchie.
- Données de cubes éparses.
- Mises à jour en batch.

⇒ B-tree sera utilisé presque exclusivement pour les colonnes unique.

Les index ROLAP typiques incluent :

- Bitmap index
- Bitmap join index

BITMAP INDEXES

BITMAP INDEX

Idée : coder chaque valeur d'un attribut dans un bitmap (bit-array).

Bitmap index sur attribut a dans relation R (n enregistrements) :

- Structure qui stocke un bitmap (n bits) par valeur de a
- $i^{ème}$ bit = 1 dans le bitmap correspondant à la valeur de a dans le tuple i, 0 dans les autres bitmaps.

Book

BookID	Title	Binding	Language
3240	Le Petit Prince	Hardcover	French
2211	Winnie the Pooh	Hardcover	English
9754	Paddington Bear	Paperback	NULL
4315	Pinocchio	Hardcover	Italian
2368	Les Vacances	Paperback	French

Bitmap index sur Binding

	Hardcover	Paperback
Row 1	1	0
Row 2	1	0
Row 3	0	1
Row 4	1	0
Row 5	0	1

Représentation binaire

```
Hardcover: 11010
Paperback: 00101
```

BITMAP COMPRESSION

Les vecteurs de bits sont épars lorsque l'attribut a une haute cardinalité :

- Opportunités pour compression, en particulier RLE (Run-Length Encoding).

Inconvénients :

- Compression rend les mises à jour coûteuses.
- Il faut décompresser à l'exécution de la requête.
- En OLTP, utiliser des bitmaps compressés peut entraîner des verrous sur de nombreux enregistrements.

BITMAP INDEXES: INTÉRÊT

Avantages :

- Pas besoin de stocker ROWID : chaque bit-vecteur est plus compact qu'un B-tree.
- Les bitmaps sont suffisamment petits pour être stockés en mémoire.
- Opérations booléennes sur les bitmaps (AND, OR, XOR, NOT) sont très rapides.
- Efficace sur les attributs de faible cardinalité.
- Pas trop inefficace sur attributs de haute cardinalité.
- Compression efficace pour les vecteurs épars.

Faiblesses :

- Maintenance coûteuse : tous les index doivent être mis à jour lors d'une insertion.
- L'espace de stockage croît avec les attributs de haute cardinalité.
- Coût de décompression.

BITMAP INDEX: OPÉRATIONS BOOLÉENNES

Book

BookID	Title	Binding	Language	Price
3240	Le Petit Prince	Stapled	French	35
2211	Winnie the Pooh	Hardcover	English	40
9754	Paddington Bear	Paperback	NULL	35
4315	Pinocchio	Stapled	Italian	15
2368	Les Vacances	Paperback	French	40

```sql
SELECT * FROM Book
WHERE Binding != 'Stapled'
    AND Price BETWEEN 20 AND 40;
```

```
Hardcover: 01000     40: 01001
Paperback: 00101     35: 10100
Stapled: 10010       15: 00010
NOT Stapled: 01101   35 OR 40: 11101
 ----------------------------------
| NOT Stapled AND (35 OR 40): 01101 |
 ----------------------------------
```

BITMAP JOIN INDEXES

BITMAP JOIN INDEX

Objectif : indexer une relation R sur un attribut a d'une autre relation R'.

But : précalculer la(les) jointure(s) pour économiser du temps d'exécution.

Stocker un bitmap sur R pour chaque valeur de n'.

Sales

ShopID	BookID	Count
1	3240	2
1	2368	1
2	3240	4
2	9754	8
2	2211	5
3	9754	3

Book

BookID	Title	Binding	Language
3240	Le Petit Prince	Hardcover	French
2211	Winnie the Pooh	Hardcover	English
9754	Paddington Bear	Paperback	NULL
4315	Pinocchio	Hardcover	Italian
2368	Les Vacances	Paperback	French

Bitmap (join) index sur Sales de l'attribut Binding:

BITMAP/JOIN INDEXES SOUS ORACLE

Exemples de commandes SQL :

```
-- Bitmap index
CREATE BITMAP INDEX binding_ix ON Book(binding);

-- Bitmap join index
CREATE BITMAP JOIN INDEX binding_bjix ON Sales(Book.binding)
    FROM Sales, Book WHERE Sales.BookID = Book.BookID;
```

Note : JOIN INDEX ≠ INDEX JOIN

JOINTURES ET PARTITIONNEMENT

REQUÊTES EN ÉTOILE

Requêtes typiques en entrepôt : joignent une grande table de faits avec plusieurs petites tables de dimensions.

Les jointures traditionnelles dans les SGBD se calculent par paires, l'optimiseur choisit l'ordre des jointures pour minimiser la taille des résultats intermédiaires.

Problème : la seule table liée aux autres est la table des Faits.

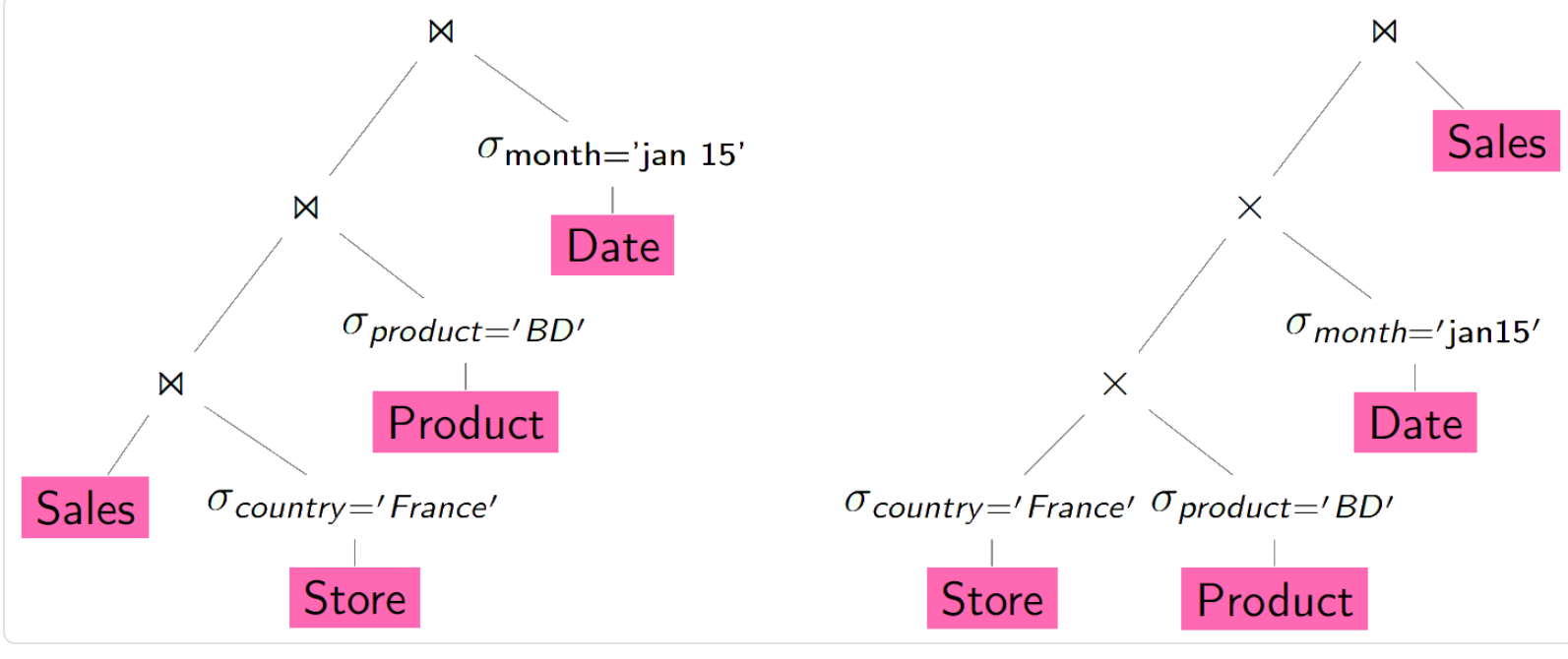
Approche alternative : calculer d'abord le produit cartésien des dimensions filtrées, puis joindre avec les faits. Plus efficace avec des prédicats très sélectifs.

Bitmap indexes/ bitmap join indexes permettent optimiser de telles requêtes.

REQUÊTES EN ÉTOILE (2)

Exemple :

- 10.000.000 tuples de ventes.
- Requête sélectionne 10 stores (parmi 100), 50 produits (parmi 1000), 20 jours (parmi 1000).
- Répartition uniforme des ventes parmi stores, etc.



REQUÊTES EN ÉTOILE (ORACLE)

Si des index bitmap (ou bitmap join index) sont présents sur toutes les clés étrangères nécessaires dans la table de faits, Oracle peut utiliser la "star transformation".

Étapes :

- Sélectionner les tuples pertinents dans la table de faits.
  Utiliser les IDs pertinents des dimensions pour générer un bitmap par FK.
- Intersection des bitmaps pour filtrer les faits à joindre aux dimensions.

Paramètre à régler : `STAR_TRANSFORMATION_ENABLED = true`.

DB2 a une approche similaire, mais utilise un semijoin par dimension avec un B-tree sur les FKs au lieu de bitmaps.

CLUSTERING

"CLUSTERED INDEXES" IN ORACLE

La plupart du temps, les tables d'une base de données ne sont pas stockées dans un ordre particulier (*heap-organized table*).

Index-organized-table (Oracle) : un B+-tree contient directement les données de la table, ce qui économise une indirection (le chargement du bloc de donnée).

Autres indexes sur la table ne stockent qu'un *logical* ROWID.

Exemple de création d'une table organisée par index :

```
-- Creating an index-organized table
CREATE TABLE FrenchCities (
    zip int PRIMARY KEY, city_name VARCHAR2(20)
)
ORGANIZATION INDEX
MAPPING TABLE; -- optional, creates a table mapping logical rowids
-- the mapping table is necessary to support bitmap indexes
```

"CLUSTERS" IN ORACLE: TABLE CLUSTERS

Objectif : grouper des tables qui partagent des colonnes et sont souvent interrogées ensemble dans un *table cluster*.

- Index cluster : stocke ensemble (même bloc) les lignes de toutes les tables ayant la même valeur de clé du cluster.
- Hash cluster : stocke ensemble les lignes ayant la même valeur de hash sur la clé du cluster.

```
-- Creating a clustered table
CREATE CLUSTER personnel(department NUMBER(4));

CREATE TABLE dept_10
CLUSTER personnel (department_id)
AS SELECT * FROM employees WHERE department_id = 10;

CREATE CLUSTER personnel(department NUMBER(4))
HASHKEYS 10;
-- HASHKEYS est idéalement à peu près le nb de valeurs unique
-- des options permettent de redéfinir fonction de hash, espace de stockage...
```

"CLUSTERS" IN ORACLE: AVANTAGES ET FAIBLESSES

Avantages :

- Accélère les jointures (moins de blocs à charger).
- Économie d'espace (répétition de la clé évitée).

Faiblesses :

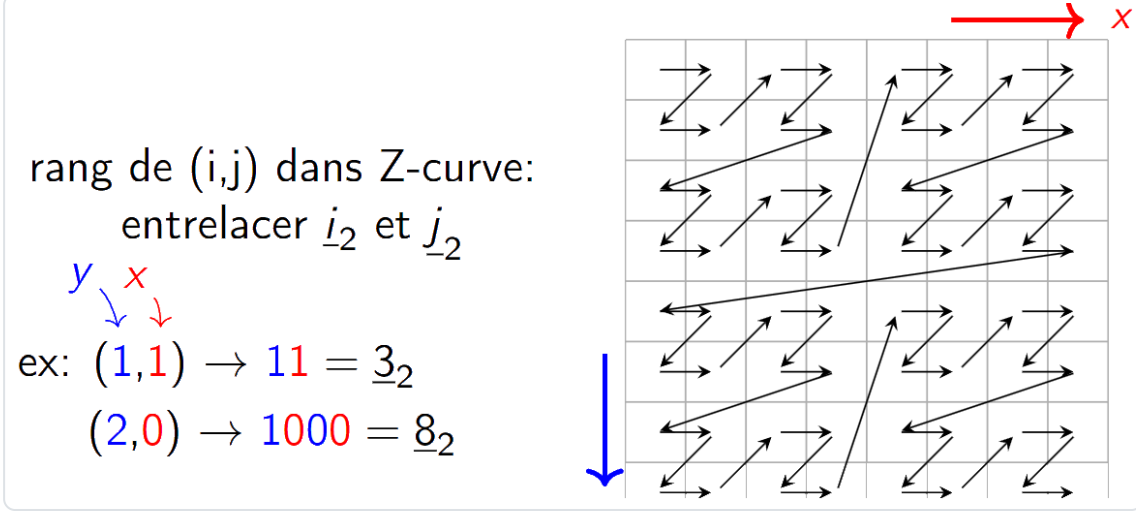
- Coût de maintenance élevé.
- Les scans de tables sont plus lents.

MULTIDIMENSIONAL CLUSTERING

Un index multicolonne serait coûteux et inutile si l'on ne filtre pas sur un préfixe de la clé.

Le multidimensional clustering stocke des tuples proches sur les colonnes du cluster à proximité.

- Linear ordering : ordre lexicographique sur les colonnes du cluster.
- Interleaved ordering : ordre défini par la courbe de Lebesgue (Z-curve).



Z-curve

CRÉATION D'UNE TABLE MULTIDIMENSIONNELLE (ORACLE)

Réorganisation d'une table en une table groupée

```
-- Creating an attribute-clustered table
CREATE TABLE sales (
    prod_id NUMBER(6) NOT NULL,
    cust_id NUMBER NOT NULL,
    time_id DATE NOT NULL,
    amount_sold NUMBER(10,2) NOT NULL
)
CLUSTERING BY INTERLEAVED ORDER (time_id, prod_id, cust_id);
-- CLUSTERING BY LINEAR ORDER (time_id, prod_id,cust_id);
```

Il est possible d'ajouter du clustering d'attributs pour les jointures :

```
-- Join attribute clustering
ALTER TABLE sales
    ADD CLUSTERING
        sales JOIN product ON (sales.prod_id=products.product_id)
        BY INTERLEAVED ORDER (time_id, product_cat);
```

"CLUSTERS" IN ORACLE: AVANTAGES DU CLUSTERING PAR ATTRIBUT

Avantages :

- Demande moins d'espace que les index.
- Réduit les accès en permettant d'accéder directement aux régions pertinentes.

  Sous Oracle : Utilisé avec *zone maps*, *in-memory min/max pruning*, et *exadata storage indexes*.
- L'ordre d'entrelacement est (à peu près) indépendant de l'ordre des colonnes.
- Permet une meilleure compression des données.

Faiblesses :

- Coût élevé, donc non maintenu par oracle après les mises à jour.

POUR ALLER PLUS LOIN

"CLUSTERS" DANS DIFFÉRENTS SGBD

Clustering dans différents SGBD :

- MySQL (InnoDB) : Clustered index.
- PostgreSQL : CLUSTER table (réorganisation unique).
- DB2 : Clustered index, MDC (Multidimensional Clustering).
- SQL Server : Clustered index, columnstores pour multidimensional clustering.

Note : la Z-curve est aussi utilisée pour des objets spatiaux dans Oracle, SQL Server, et DB2.

AUTRES STRUCTURES DE STOCKAGE PHYSIQUE

Pour les informations multidimensionnelles ou spatiales :

- UB-tree : un B-tree sur Z-order.
- R-tree et ses variantes.
- Kd-tree.
- Grid files.
- Multidimensional hashing.

ORACLE'S ZONE MAPS

IBM DB2 a un outil similaire.

Les zone maps sont des structures d'index qui stockent les valeurs min/max de chaque colonne pour chaque zone.

- Efficace lorsque les données sont groupées sur les attributs de la zone map.
- Plus compact que les index classiques (granularité = 1 zone au lieu d'un tuple).
- Problème : les intervalles de certaines zones peuvent devenir non-fiables après des mises à jour.

RÉSUMÉ

B-trees / B+-trees :

- **Monocolonne vs Multicolonnes** : Ces index peuvent être soit monocolonne (une seule colonne), soit multicolonnes (plusieurs colonnes).
- **Groupant vs Non-groupant** : Les B+-trees peuvent être utilisés comme index groupant (clustered) lorsque la table est triée selon la clé d'index.
- **Unique vs Non-unique** : Les B-trees peuvent supporter des index uniques ou non-uniques.
- **Dense vs Épars** : Les B+-trees sont généralement denses car ils référencent chaque tuple.

Index Bitmap :

- **Monocolonne vs Multicolonnes** : Les index bitmap peuvent être soit monocolonne, soit multicolonnes.
- **Groupant vs Non-groupant** : Les index bitmap sont généralement non-groupants.
- **Unique vs Non-unique** : Les index bitmap sont typiquement utilisés pour des valeurs non-uniques, notamment pour des attributs de faible cardinalité.
- **Dense vs Épars** : Les index bitmap sont en général épars, surtout lorsqu'on utilise des techniques de compression comme l'encodage par longueur de course (RLE).

Index Bitmap de Jointure :

- **Monocolonne vs Multicolonnes** : Les index bitmap de jointure peuvent être multicolonnes puisqu'ils joignent plusieurs tables.
- **Groupant vs Non-groupant** : Ceux-ci sont généralement non-groupants.
- **Unique vs Non-unique** : Ils peuvent être non-uniques puisqu'ils joignent des tables sur des attributs qui ne sont pas nécessairement uniques.
- **Dense vs Épars** : Comme les index bitmap, ils sont généralement épars.

Index Clustered (groupants) :

- **Monocolonne vs Multicolonnes** : Ces index peuvent être soit monocolonne, bien qu'ils soient souvent utilisés avec une seule colonne.
- **Groupant vs Non-groupant** : Ces index sont toujours groupants car les données sont physiquement ordonnées selon l'index.
- **Unique vs Non-unique** : Les index clustered peuvent être uniques ou non-uniques.
- **Dense vs Épars** : Les index clustered sont généralement denses, car chaque tuple est référencé.

BIBLIOGRAPHIE

- Oracle Database Concepts
- Microsoft: Clustered Indexes and Columnstores
- DB2: Indexes and MDC
- (MultiDimensional) Indexes: T. Grust's Lecture

LECTURE DES PLANS

1. `EXPLAIN (ANALYZE, FORMAT JSON) ...`
2. Copiez le plan
3. Allez sur https://tatiyants.com/pev/