Création de la base de données, et stratégie d'indexation

Pour concevoir notre base de données, j'ai suivi une approche méthodique en plusieurs étapes. Exemple pour la conception d'une base de données d'une entreprise de vente en ligne, j'ai créé une base de données e_commerce avec une table Produits :

1. Analyse des besoins : J'ai identifié les informations pour chaque produit, comme le SKU (Stock Keeping Unit), le nom, le prix, la quantité en stock, et la catégorie.

2. Conception du schéma :

J'ai défini la structure de la table Produits, en tenant compte des contraintes business et des performances.

3. Implémentation :

```
mysql> CREATE DATABASE e_commerce;
Query OK, 1 row affected (0.07 sec)

mysql> USE e_commerce;
Database changed
mysql> CREATE TABLE Produits (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     sku VARCHAR(20) NOT NULL UNIQUE,
    ->     nom VARCHAR(100) NOT NULL,
    ->     description TEXT,
    ->     prix DECIMAL(10, 2) NOT NULL,
    ->     quantite_stock INT NOT NULL,
    ->     categorie VARCHAR(50),
    ->     date_ajout TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.12 sec)

mysql>
```

2. Optimisation :

J'ai créé des index pour améliorer les performances des requêtes fréquentes :

```
mysql> CREATE INDEX idx_sku ON Produits(sku);
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_categorie ON Produits(categorie);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Vérification de l'index :

Pour vérifier que l'index a bien été pris en compte :

```
mysql> SHOW INDEX FROM Produits;
+----------+------------+---------------+--------------+-------------+-----------+-------------+----------+--------+----
--+------------+---------+---------------+---------+------------+
| Table    | Non_unique | Key_name      | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Nul
l | Index_type | Comment | Index_comment | Visible | Expression |
+----------+------------+---------------+--------------+-------------+-----------+-------------+----------+--------+----
--+------------+---------+---------------+---------+------------+
| produits |          0 | PRIMARY       |            1 | id          | A         |           5 |     NULL |   NULL |
  | BTREE      |         |               | YES     | NULL       |
| produits |          0 | sku           |            1 | sku         | A         |           5 |     NULL |   NULL |
  | BTREE      |         |               | YES     | NULL       |
| produits |          1 | idx_sku       |            1 | sku         | A         |           5 |     NULL |   NULL |
  | BTREE      |         |               | YES     | NULL       |
| produits |          1 | idx_categorie |            1 | categorie   | A         |           3 |     NULL |   NULL | YES
  | BTREE      |         |               | YES     | NULL       |
+----------+------------+---------------+--------------+-------------+-----------+-------------+----------+--------+----
--+------------+---------+---------------+---------+------------+
4 rows in set (0.07 sec)
```

Détermination des plans d'exécution

Plans d'exécution pour les requêtes suivantes :

1) Avec clause order by

Select titre **from** Film order by **annee**

- Un itérateur de parcours séquentiel, suivi d'un itérateur de tri, suivi d'un itérateur pour la projection.

2) Recherche d'un élément minimal

Select min (annee) **from** Film

- Un itérateur de parcours séquentiel, suivi de notre itérateur Min, suivi d'un itérateur pour la projection. Autre possibilité : on trie, et on ne garde que le premier.

Elimination des doublons

Select distinct genre from Film

- Un itérateur de parcours séquentiel, suivi d'un itérateur de tri, suivi de notre itérateur distinct.

Coût des jointures par boucles imbriquées

Pour ces deux relations *R* et *S*, de tailles respectives $|R|$ et $|S|$ (en nombre de blocs). On dispose d'une mémoire *mem* de taille *M*, dont les blocs sont dénotés $mem[1], mem[2], \ldots, mem[M]$.

Détermination du coût d'une jointure $R \bowtie S$, en nombre d'entrées/sorties, pour l'algorithme 1:

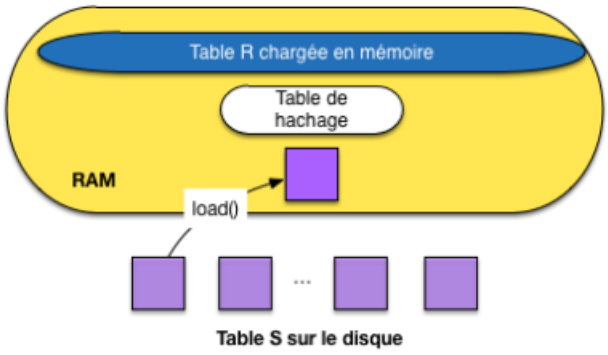
```
$posR = 1   # On se place au début de R
while [$posR <= |R|] do
    Lire  R[$posR] dans $mem[1] # On lit les blocs 1 par 1
    $posS = 1  # On se place au début de S
    while ($posS <= |S|)  do
       Lire  S[$posS] dans $mem[2] # On lit les blocs 1 par 1
       # JoinList est l'algorithme donné en cours
       JoinList (mem[1], mem[2])
      done
      $posS = $posS + 1 # Bloc suivant de S
    done
   $posR = $posR + 1 # Bloc suivant de R
 done
```

Cet algorithme est une jointure par boucle imbriquée qui exploite très mal la mémoire puisque seuls les deux premiers blocs de *m* sont utilisés. Le coût est de $|R|+|R|\times|S|$.

2) Coût d'une jointure $R \bowtie S$, en nombre d'entrées/sorties, pour l'algorithme 2:

```
$posR = 1   # On se place au début de R
while [$posR <= |R|] do
   Lire  R[p$osR..($posR+M-1)] dans $mem[1..M-1] # On lit M-1 blocs de R
   $posS = 1
   while ($posS <= |S|) do
       Lire  S[$posS] dans $mem[2] # On lit les blocs 1 par 1
       # JoinList est l'algorithme donné en cours
      JoinList (mem[1], mem[2])
   done
   $posS = $posS + 1 # Bloc suivant de S
 done
 $posR = $posR + M # On lit les M blocs suivants de R
done
```

est en fait celui illustré par la figure ci-dessous. La mémoire *m* est utilisée pour réduire le nombre d'itérations sur *R*, avec un coût de $\text{Cout} = |R| + [\frac{R}{M-1}]\times|S|$.



Boucle imbriquée avec chargement complet d'une table en RAM

Conclusion

Au vu des formules de coût il faut prendre la petite table comme table directrice.