Bases de données

Objectifs:

- structure d'une base de données
- interrogation d'une base de donnée

I. Introduction

Besoin

Stockage d'un grand nombre de données :
optimisation du volume de stockage
intégrité des données

Redondance ces stockages
Redondance des alimentations électriques / systèmes de refroidissements

Rapidité d'accès à l'information

Délocalisation du stockage dans le Cloud

Définition d'une relation

Relation :

- modèle abstrait → structure mais pas d'opérations
- pas associé à un langage
- pratique pour l'analyse du monde réel

Relation / table : $R = (A_0 : D_0, A_1 : D_1, \ldots)$ où A_i est le nom de l'attribut et D_i est le domaine de l'attribut A_i :
→ ensemble des valeurs possibles (fini ou non)
→ type asocié

Base de données → ensemble de relations

Nécessité de représenter les relations : tableau, enregistrements,...

Vocabulaire

- Relation et tables sont synonymes.
- Attributs, champs et traits sont synonymes
- Instance, enregistrement, observation ($\simeq$ ligne de données)

Clé primaire: ensemble d'attributs permettant l'identification d'un enregistrement sans ambigüité.

L'implémentation de SQL n'est pas au programme, uniquement la syntaxe l'est.

Opérations sur les données

Type d'opération sur les données → requêtes

- Créer / insérer
- Mettre à jour / modifier
- Détruire
- Rechercher → la seule au programme

Traitement de la requête:

1. Analyse syntaxique
2. Optimisation par le SGBD (système de gestion de base de données) → Hors programme
3. Exécution pour obtenir le résultat

Interrogation de la base de données :

calcul relationnel → hors programme
algèbre relationnelle → hors programme
langage de requête : SQL → au programme

Sélection : renvoie une table contenant une partie des enregistrements.

Projection : Renvoie une table contenant une partie des attributs.

Le résultat de la requête ci-dessous est une table.

```
SELECT att_1, att_1, ... FROM nom_table WHERE condition;
```

Pour récupérer tous les attributs, on utilise `*`.

Pour les conditions, on peut utiliser `+`, `-`, `*`, `/`, `=`, `<>`, `<=`, `>=`, `<`, `>`, `and`, `or`, `not`, `is null`, `is not null`.

Exemples :

```
SELECT Id, Nom FROM villes WHERE Region = "Pays de la Loire";
SELECT Id, Nom, Nb_hab FROM villes WHERE Id = 5;
SELECT Id, Nom, Nb_hab FROM villes WHERE Nom = "Nantes";
SELECT Id, Nom, Nb_hab FROM villes WHERE Nb_hab = 934000;
SELECT * FROM villes WHERE Region = "Pays de la Loire";
SELECT Id, Nom FROM villes WHERE Id >= 5;
SELECT Id, Nom, Nb_hab FROM villes WHERE Region = "Pays de la Loire" ORDER BY Id DESC;
SELECT DISTINCT Region FROM villes ORDER BY Region ASC;
```

L'utilisation de `ORDER BY` permet d'ordonner les éléments d'une table lors d'une requête.

ordre lexicographique, comparaison de nombres
possibilités de combiner l'ordre : `ORDER BY Region ASC, Nom DESC`.

L'utilisation de `DISTINCT` permet de supprimer les redondances de la table de retour de requête.

Au lieu de faire `X = "Possibilité 1" OR X = "Possibilité 2"`, on peut écrire `X IN ("Possibilité 1", "Possibilité 2")`.

Attention, si on a une valeur à `NULL`, on ne peut pas utiliser l'égalité (`X = NULL`) mais on doit utiliser le mot clé `IS` (`X IS NULL`).

Autre possibilité pour le 5. : utiliser `GROUP BY` en fin de requête pour regrouper les attributs identiques (normalement prévu pour une utilisation plus complexe).

Le mot clé `LIMIT` permet de limiter le nombre de lignes retournées. Si on veut omettre des lignes au début, on utilise l'instruction `OFFSET`.

Exemples :

```
SELECT Id, Nom, Region FROM Villes
ORDER BY Region ASC, Nom DESC;

SELECT Id, Nom, Region FROM Villes
ORDER BY Region ASC, Nom DESC LIMIT 3;

SELECT Id, Nom, Region FROM Villes
ORDER BY Region ASC, Nom DESC LIMIT 3 OFFSET 5;
```

Aggrégations : opérations sur les valeurs d'un attribut `MIN`, `MAX`, `AVG`, `COUNT`, `SUM`.

`GROUP BY` : regroupement (en fin de requête).

`NULL` : ignorés.

Attention : pas d'agrégation dans le `WHERE`.

`AS` : alias

Filtrage des agrégats avec `HAVING` (en fin de requête) → après un `GROUP BY` (mais cela dépend des environnements)

`WHERE` → attribut unique

`HAVING` →

Exemple de syntaxe :

```
SELECT MIN(att_2) FROM nom_table WHERE condition_1
GROUP_BY att_1 HAVING condition_2
```

Exemples :

```
SELECT MIN(Nb_hab) FROM Villes;

SELECT AVG(Nb_hab) AS A_TROUVER FROM Villes;

SELECT Region, AVG(Nb_hab_region) as moyenne FROM Villes
GROUP BY Region ORDER BY moyenne DESC;
```

```
# 3 solutions :

SELECT Nom FROM Villes GROUP BY Id
HAVING Nb_hab_region = MIN(Nb_hab_region); # je ne sais pas si ça marche...

SELECT Nom FROM Villes WHERE Nb_hab = (SELECT MIN(Nb_hab FROM Villes);

SELECT Nom FROM Villes ORDER BY Nb_hab LIMIT 1;

SELECT COUNT(Nb_hab) AS PAS_UN_PIEGE FROM Villes;

SELECT COUNT(DISTINCT Nb_hab_region) as Plus_compliquee FROM Villes;
```

La question 15 sera revue Jeudi prochain.