RECHERCHE(2) ANALYSE ET CONCEPTION DES DONNEES

➢ ALGEBRE RELATIONNELLE

L'algèbre relationnelle est un ensemble d'opérations mathématiques permettant de manipuler les relations dans une base de données relationnelle. Ces opérations sont très importantes pour les requêtes SQL car elles permettent de combiner les données de différentes tables. Voici une brève description des principales opérations de l'algèbre relationnelle :

1. **Sélection** : permet de sélectionner les lignes qui satisfont une condition donnée. La syntaxe en SQL est la suivante :

   *SELECT * FROM table WHERE condition;*

2. **Projection** : permet de sélectionner les colonnes qui nous intéressent. La syntaxe en SQL est la suivante :

   SELECT colonne1, colonne2 FROM table;

3. **Produit cartésien** : permet de combiner toutes les lignes de deux tables. La syntaxe en SQL est la suivante :

   *SELECT * FROM table1, table2;*

4. **Union** : permet de combiner les lignes de deux tables en éliminant les doublons. La syntaxe en SQL est la suivante :

   *SELECT * FROM table1 UNION SELECT * FROM table2;*

5. **Différence** : permet de sélectionner les lignes qui sont dans une table mais pas dans l'autre. La syntaxe en SQL est la suivante :

   *SELECT * FROM table1 EXCEPT SELECT * FROM table2;*

6. **Intersection** : permet de sélectionner les lignes qui sont communes à deux tables. La syntaxe en SQL est la suivante :

   *SELECT * FROM table1 INTERSECT SELECT * FROM table2;*

7. **Division** : permet de sélectionner les lignes d'une table qui correspondent à toutes les valeurs d'une autre table. La syntaxe en SQL est plus complexe et nécessite une sous-requête ;

8. **Jointure (s)** : permet de combiner les lignes de deux tables en utilisant une condition de jointure. Les types de jointures les plus courants en SQL sont la jointure interne, la jointure externe gauche, la jointure externe droite et la jointure externe complète. La syntaxe en SQL est la suivante :

   *SELECT * FROM table1 JOIN table2 ON condition_de_jointure.*

➢ LANGAGE DE DEFINITION DES DONNEES (LDD)

Le Langage de Définition des Données (LDD) est une partie du langage SQL qui permet de définir et de manipuler la structure d'une base de données. Voici quelques-unes des principales instructions du LDD :

1. **CREATE** : permet de créer des objets dans la base de données tels que des tables, des vues, des index, etc. La syntaxe pour créer une table est la suivante :

   CREATE TABLE nom_table (colonne1 type_de_donnee, colonne2 type_de_donnee, ...);

2. **DROP** : permet de supprimer des objets de la base de données, tels que des tables, des vues, des index, etc. La syntaxe pour supprimer une table est la suivante :

   DROP TABLE nom_table;

3. **ALTER** : permet de modifier la structure d'un objet existant dans la base de données, comme ajouter ou supprimer une colonne d'une table, modifier le type de données d'une colonne, etc. La syntaxe pour ajouter une colonne à une table est la suivante :

   ALTER TABLE nom_table ADD colonne type_de_donnee;

4. **RENAME** : permet de renommer un objet existant dans la base de données, comme renommer une table, une colonne ou un index. La syntaxe pour renommer une table est la suivante :

   RENAME nom_table TO nouveau_nom_table;

5. **TRUNCATE** : permet de supprimer toutes les lignes d'une table, mais de conserver sa structure. C'est plus rapide que l'instruction DROP TABLE suivie de CREATE TABLE pour recréer la table. La syntaxe pour vider une table est la suivante :

   TRUNCATE TABLE nom_table;

➢ LANGAGE DE MANIPULATION DES DONNEES (LMD)

Le Langage de Manipulation de Données (LMD) est une partie du langage SQL qui permet d'effectuer des opérations de manipulation sur les données dans une base de données. Voici les principales instructions du LMD :

1. **INSERT** : permet d'insérer de nouvelles lignes de données dans une table. La syntaxe pour insérer des données dans une table est la suivante :

   INSERT INTO nom_table (colonne1, colonne2, ...) VALUES (valeur1, valeur2, ...);

2. **UPDATE** : permet de modifier les valeurs des colonnes dans une ou plusieurs lignes d'une table. La syntaxe pour mettre à jour des données dans une table est la suivante :

   ***UPDATE** nom_table SET colonne1 = nouvelle_valeur1, colonne2 = nouvelle_valeur2, ... WHERE condition;*

   La clause WHERE est utilisée pour spécifier la condition qui détermine les lignes à mettre à jour. Si aucune clause WHERE n'est spécifiée, toutes les lignes de la table seront mises à jour.

3. **DELETE** : permet de supprimer une ou plusieurs lignes d'une table. La syntaxe pour supprimer des données d'une table est la suivante :

   DELETE FROM nom_table WHERE condition;

   La clause WHERE est utilisée pour spécifier la condition qui détermine les lignes à supprimer. Si aucune clause WHERE n'est spécifiée, toutes les lignes de la table seront supprimées.

➢ **LANGAGE D'INTERROGATION DES DONNEES (LMD)**

1. **JOINTURE** : Supposons que nous avons deux tables, "Customers" et "Orders", avec une colonne commune "customer_id". Pour obtenir des informations sur les commandes passées par chaque client, nous pouvons utiliser une jointure :

```
SELECT Customers.customer_id, Customers.customer_name, Orders.order_date,
Orders.order_amount

FROM Customers

JOIN Orders

ON Customers.customer_id = Orders.customer_id;
```

2. **PROJECTION** : Pour sélectionner uniquement les colonnes "customer_name" et "customer_city" de la table "Customers", nous pouvons utiliser la projection :

```
SELECT customer_name, customer_city

FROM Customers;
```

3. **Sous-requête** : Pour obtenir les informations sur les clients qui ont passé des commandes pour un produit spécifique, nous pouvons utiliser une sous-requête :

```
SELECT customer_name

FROM Customers

WHERE customer_id IN (SELECT customer_id FROM Orders WHERE product_name =
'T-shirt');
```

4. **Alias (AS)** : Pour renommer la colonne "customer_name" en "name" dans le résultat de la requête, nous pouvons utiliser un alias :

```
SELECT customer_name AS name, customer_city

FROM Customers;
```

5. Commande **UNION** : Pour combiner les résultats de deux requêtes SELECT, nous pouvons utiliser la commande UNION :

```
SELECT customer_name, customer_city

FROM Customers

WHERE customer_city = 'Paris'

UNION

SELECT customer_name, customer_city

FROM Customers

WHERE customer_city = 'London';
```

6. Commande **GROUP BY** : Pour regrouper les commandes par année et calculer le montant total pour chaque année, nous pouvons utiliser la commande GROUP BY :

```
SELECT YEAR(order_date) AS year, SUM(order_amount) AS total_amount
FROM Orders
GROUP BY YEAR(order_date);
```

7. Commande **ORDER BY** : Pour trier les clients par ordre alphabétique de nom, nous pouvons utiliser la commande ORDER BY :

```
SELECT customer_name, customer_city
FROM Customers
ORDER BY customer_name ASC;
```

8. Commande **HAVING** : Pour filtrer les commandes par année et retourner uniquement les années avec un montant total supérieur à 10000, nous pouvons utiliser la commande HAVING :

```
SELECT YEAR(order_date) AS year, SUM(order_amount) AS total_amount
FROM Orders
GROUP BY YEAR(order_date)
HAVING total_amount > 10000;
```

9. Fonction **DISTINCT** : Utilisée pour sélectionner uniquement des valeurs distinctes dans une colonne. Par exemple :

```
SELECT DISTINCT pays FROM utilisateurs;
```

10. **COUNT** : Utilisé pour compter le nombre de lignes ou d'éléments dans une colonne. Par exemple :

```
SELECT COUNT(*) FROM utilisateurs;
```

11. **LIMIT** : Utilisé pour limiter le nombre de lignes retournées par une requête. Par exemple :

```
SELECT * FROM utilisateurs LIMIT 10;
```

12. **LIKE %% (B%A)** : Utilisé pour rechercher des valeurs qui correspondent à un modèle spécifié à l'aide de caractères de remplacement. Par exemple, pour rechercher des noms commençant par "B" et se terminant par "A", vous pouvez utiliser :

```
SELECT * FROM utilisateurs WHERE nom LIKE 'B%A';
```

13. **IN & NOT IN** : Utilisés pour spécifier plusieurs valeurs possibles dans une condition WHERE. Par exemple :

```
SELECT * FROM utilisateurs WHERE pays IN ('France', 'Allemagne',
'Espagne');
```

14. **BETWEEN** : Utilisé pour spécifier une plage de valeurs dans une condition WHERE. Par exemple :

```
SELECT * FROM utilisateurs WHERE age BETWEEN 18 AND 30;
```

15. **AVG** : Utilisé pour calculer la moyenne des valeurs d'une colonne numérique. Par exemple :

```
SELECT AVG(salaire) FROM employes;
```

16. **MIN** : Utilisé pour trouver la valeur minimale dans une colonne. Par exemple :

```
SELECT MIN(prix) FROM produits;
```

17. **MAX** : Utilisé pour trouver la valeur maximale dans une colonne. Par exemple :

```
SELECT MAX(quantite) FROM stock;
```

18. **SUM** : Utilisé pour calculer la somme des valeurs d'une colonne numérique. Par exemple :

```
SELECT SUM(ventes) FROM produits;
```

Mor Anta SENE