C9 Base de données

cours 9_2 : Langage SQL

Langage SQL : requêtes d'interrogation et de mise à jour d'une base de données.	Identifier les composants d'une requête. Construire des requêtes d'interrogation à l'aide des clauses du langage SQL : SELECT, FROM, WHERE, JOIN. Construire des requêtes d'insertion et de mise à jour à l'aide de : UPDATE, INSERT, DELETE.	On peut utiliser DISTINCT, ORDER BY ou les fonctions d'agrégation sans utiliser les clauses GROUP BY et HAVING.

FIGURE 1 – BO

Le langage SQL (Structured Query Langage) permet de communiquer avec un SGBD pour gérer ses données.

1 Requêtes en écriture (INSERT INTO, UPDATE, DELETE)

Afin de mettre à jour la base de données on peut réaliser des **requêtes**.

1.1 Requête INSERT INTO

L'**ajout de nouvelles données** dans une table s'effectue avec une requête `INSERT INTO` dont voici le modèle :

```
-------------------------------------------------------
INSERT INTO TableName (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
-------------------------------------------------------
```

Exemples :

```
-- Ajout en fournissant toutes les données
INSERT INTO Clients
VALUES (7, 'SHOWKANAPE', 'Ophélie', 23, 'Niort');
-- Ajout en sélectionnant les colonnes concernées
INSERT INTO Clients (nom, prenom)
VALUES ('SHOWKANAPE', 'Ophélie');
```

Remarque : la clef primaire est rarement renseignée car elle est très souvent configurée pour s'auto-incrémenter automatiquement

1.2 Requête UPDATE

La **mise à jour** de données existantes dans une table s'effectue avec une requête UPDATE dont voici le modèle :

```
-----------------------------------------
UPDATE TableName
SET column1 = value1, column2 = value2, ...
WHERE condition;
-----------------------------------------
```

Exemples :

```
-- Modification de l'âge
UPDATE Clients
SET age = 25
WHERE id_client = 7;
-- Modification de l'âge et du prénom
UPDATE Clients
SET age = 25, prenom = 'Ofely'
WHERE id_client = 7;
```

1.3 Requête DELETE

La **suppression de données existantes** dans une table s'effectue avec une requête DELETE dont voici le modèle :

```
----------------------
DELETE FROM TableName
WHERE condition;
----------------------
```

Exemples :

```
-- Suppression d'un client
DELETE FROM Clients
WHERE id_client = 7;
```

2 Requêtes d'interrogation (SELECT, FROM, WHERE)

> 📢 définition : La sélection des données dans une table s'effectue avec une requête `SELECT` dont voici le modèle. La clause WHERE est optionnelle.

```
---------------------------
SELECT column1, column2, ...
FROM TableName
WHERE condition;
---------------------------
```

Exemples :

```
-- Sélection des noms et prénoms de tous les clients
SELECT nom, prenom
FROM Clients;
-- Sélection de tous les attributs de tous les clients
```

```sql
SELECT *
FROM Clients;
-- Sélection du nom de tous les clients niortais
SELECT nom
FROM Clients
WHERE ville = 'Niort';
-- Sélection du nom et age des "jeunes" clients niortais
SELECT nom, age
FROM Clients
WHERE ville = 'Niort'
AND age < 30;
```

3 Requêtes d'interrogation (DISTINCT, ORDER BY, AVG)

Parfois une colonne peut contenir des valeurs identiques qu'on pourra filtrer en ajoutant le mot clef `DISTINCT` juste après `SELECT` :

```sql
-- On ne sélectionne que les villes distinctes
SELECT DISTINCT ville
FROM Clients;
```

Souvent, on souhaite récupérer les données d'une table classées selon un certain ordre. Le mot clef `ORDER BY` pourra alors être ajouté à la fin de la requête `SELECT` :

```sql
-- Sélection des clients du plus vieux plus jeune
SELECT nom, prenom, age
FROM Clients
ORDER BY age DESC;
-- Sélection des clients par ordre alphabétique
SELECT nom, prenom, age
FROM Clients
ORDER BY nom ASC;
```

Il existe également des fonctions d'agrégation (**AVG, MIN, MAX, SUM, COUNT**) permettant d'effectuer différents calculs statistiques sur un ensemble d'enregistrements :

```sql
-- Calcul de l'âge moyen de tous les clients
SELECT AVG(age)
FROM Clients;
-- Qui est le plus jeune client ?
SELECT nom, prenom, MIN(age)
FROM Clients;
-- Où habite le plus vieux client ?
SELECT ville, MAX(age)
FROM Clients;
-- Quel est l'âge cumulé de tous les clients ?
SELECT SUM(age)
FROM Clients;
-- Combien y a-t-il de clients à Niort ?
SELECT COUNT(id_client)
FROM Clients
WHERE ville = 'Niort';
```

Remarque : il vaut mieux sous-traiter les calculs au SGBD plutôt que les faire soit même en Python,

c'est plus simple et le SGBD le fait en général plus efficacement.

Quand on souhaite limiter le nombre de lignes dans les résultats, on peut utiliser `LIMIT` comme ici :

```
-- On limite à 3 lignes les résultats
SELECT nom, prenom
FROM Clients
LIMIT 3;
```

Les recherches impliquant des chaînes de caractères sont plus aisées avec `LIKE` car on peut lui associer des filtres

```
-- Recherche des noms comme Axxxxx
SELECT nom
FROM Clients
WHERE nom LIKE 'A%';
-- Recherche des noms comme xDxxxxx
SELECT nom
FROM Clients
WHERE nom LIKE '_D%';
```

Remarque : on peut également utiliser l'opérateur OR et les parenthèses pour construire des clauses WHERE plus sophistiquées

4 Requêtes avec jointures (JOIN)

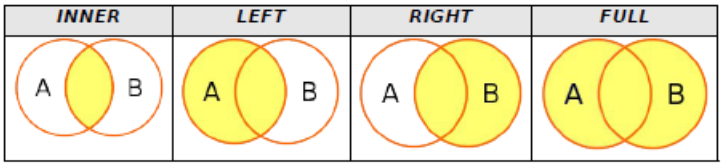
définition : Les **jointures** comme le nom l'indique permettent de joindre les données issues de 2 tables différentes.
Les jointures utilisent les **clefs primaires et étrangères** pour réaliser la relation entre les 2 tables.

```
-------------------------------------------------
SELECT TableName1.column3, TableName2.column5, ...
FROM TableName1
INNER JOIN TableName2
ON TableName1.column2 = TableName2.column1;
-------------------------------------------------
```

```
Exemple :
-- Affiche qui a emprunté quoi
SELECT Clients.nom, Clients.prenom, Livres.nom
FROM Clients
INNER JOIN Livres
ON Clients.id_client = Livres.id_client_fk;
-- Affiche les livres empruntés par BIJOBA Jo
SELECT Livres.nom
FROM Clients
INNER JOIN Livres
ON Clients.id_client = Livres.id_client_fk
WHERE Clients.nom = 'BIJOBA';
```

Remarque1 : pour éviter les conflits de nom de colonne, il est préférable d'ajouter en *préfixe le nom de la table.*

❑ *Remarque2* : les mots `WHERE`, `DISTINCT`, `AVG`, `MIN`, `MAX`, `SUM`, `COUNT`, `LIMIT` et `LIKE` sont utilisables également avec les jointures.
❑ *Remarque3* : il existe plusieurs types de jointures INNER, LEFT, RIGHT, FULL... mais c'est la jointure INNER qui est la plus utilisée et au programme en NSI.



5 Contraintes d'intégrité

Dans le chapitre 9_1 nous avons survolé les contraintes d'intégrité que l'on pouvait ajouter pour rendre les données plus cohérentes et robustes. Ces contraintes peuvent être ajoutées lors de la création des tables. La plupart des outils graphiques proposent une méthode simple pour le faire. Dans ce qui suit, on observera comment cela est possible avec le langage SQL.

5.1 Contrainte de domaine

Ce premier exemple propose une création de table avec le choix pour chaque attribut d'un domaine ou type particulier ainsi que de vérifications plus spécifiques avec le mot clef `CHECK`

```
-- Choix des domaines pour les attributs de la table Clients
CREATE TABLE "Clients" (
"nom" TEXT,
"prenom" TEXT,
"age" INTEGER,
"ville" TEXT
);
```

On peut aussi y ajouter des vérifications plus spécifiques avec le mot clef `CHECK` et des **conditions** :

```
-- Choix des domaines pour les attributs de la table Clients
CREATE TABLE "Clients" (
"nom" TEXT,
"prenom" TEXT CHECK(length(prenom) <= 30),
"age" INTEGER CHECK(age>0 AND age<200),
"ville" TEXT CHECK(ville IN ('Niort', 'Aiffres'))
);
---
```

❑ A noter que les domaines/types proposés varient d'un SGBD à l'autre. Se référer à la documentation technique du SGBD pour en connaître les détails.

5.2 Contrainte de relation

Afin de rendre chaque enregistrement unique, on ajoute une **clef primaire** id_client de type `INTEGER` avec les mots-clefs `PRIMARY KEY`.

Le paramétrage `AUTOINCREMENT` permet d'auto-incrémenter la valeur de chaque enregistrement à chaque requête `INSERT INTO` sans avoir à se soucier des valeurs déjà dans la table.

```
CREATE TABLE "Clients" (
"id_client" INTEGER PRIMARY KEY AUTOINCREMENT,
"nom" TEXT,
```

```
"prenom" TEXT,
"age" INTEGER CHECK(age>0 AND age<200),
"ville" TEXT CHECK(ville IN ('Niort','Aiffres'))
);
```

5.3 Contrainte de référence

La contrainte de référence se fera en ajoutant une **clef étrangère** dans la table Livres référençant la clef primaire de la table Clients. Les mots-clefs FOREIGN KEY et REFERENCES sont alors utilisés :

```
-- Table Livres avec tous ses attributs
CREATE TABLE "Livres" (
"id_livre" INTEGER PRIMARY KEY AUTOINCREMENT,
"nom" TEXT,
"prix" REAL,
"id_client_fk" INTEGER,
FOREIGN KEY("id_client_fk") REFERENCES "Clients"("id_client")
);
```