Leçon 29 : Requêter en langage SQL

Niveau : MPII / Tle

Prérequis : Schéma entité association ; Modèle relationnel

Pendant toute la leçon, on se base sur l'exemple suivant :

produits (num_produit, nom, prix, poids)
clients (num_client, nom, adresse, ville)
commande (#num_produit, #num_client, quantite)

I - Les tables

Def 1 : Une table est un tableau où chaque colonne est étiquettée par un attribut et contient des données du même type, et où chaque ligne est appelée n-uplet, une entrée ou encore un enregistrement.

Exemple 2 :

num_produit	nom	prix	poids
1	pomme	1,5	1
2	poulet	7	2
3	patate	2	1

produits

II - Requêtes de base (Tle)

II.1) Requête de sélection

Une requête de sélection prend en entrée une ou plusieurs tables et renvoie une nouvelle table.

Syntaxe 3 : FROM : appelle une table
SELECT : sélectionne les colonnes (projection)

Exemple 4 : Afficher le nom de tous les produits :

```
SELECT nom
FROM produits
```

Syntaxe 5 : WHERE est utilisé avant une sélection pour filtrer les n-uplets. Il est suivi d'une condition booléenne : seules les entrées la satisfaisant seront renvoyées.

Exemple 6 : Afficher le nom des produits coutant moins de 50€

```
SELECT nom FROM produits WHERE prix < 50
```

Remarque 7 : Dans un WHERE, on a le droit aux opérations entières, à <, >, =, AND, OR, NOT.

Syntaxe 8 : $table_1$ JOIN $table_2$ ON condition crée la table des éléments de $table_1$ et de $table_2$ (avec les colonnes des deux tables) en ne gardant que les lignes qui vérifient la condition. *

Exemple 9 : Les noms des produits présents dans une commande de gros (avec plus de 10 éléments)

```
SELECT p.nom
FROM produits AS p JOIN
     commandes AS c ON p.num_clients = c.num_clients
WHERE c.quantite > 10.
```

Syntaxe 10 : Autres mot clefs :
- DISTINCT : après un SELECT, pour ne garder que les lignes différentes
- SELECT * : affiche toutes les colonnes
- ORDER BY attribut ASC (resp DESC) : à la fin d'une requête, trie les elts par ordre croissant (resp décroissant) selon attributs
- LIMIT x OFFSET y : à la fin d'une requête, enlève les y premiers elts et affiche seulement les x premiers restant.

II.2) Insertion et mise à jour

Requête d'insertion 11 : On ajoute un n-uplet dans une table avec le mot-clef "INSERT INTO" suivi du nom de la table et du n-uplet.

Exemple 12 : INSERT INTO produit VALUES (4, "baguette", 1.20, 0.2), (5, "navet", 2.5, 1)

Ne rien écrire hors du cadre.

Requête de modification 13 : On utilise UPDATE et SET pour modifier le contenu d'une table, suivi d'un WHERE qui sélectionne les lignes à modifier.

Exemple 14 : UPDATE produits SET prix = 1.3 WHERE id = 1

Requête de suppression 14 : DELETE FROM table WHERE cond supprime de la table tous les n-uplets vérifiant cond.

Exemple 15 : DELETE FROM produits WHERE num_produit = 2.

Propriété 16 : Si une requête de modification viole une contrainte (clef primaire déjà présente, suppression d'un elt dont la clef est secondaire dans une autre table, ...), la requête n'est pas effectuée.

Developpement 1 : Premiers pas en SQL.

III - Pour aller plus loin (MP2I/MPI)

III.1 - Types de données en SQL

SQL possède de nombreux types de données. Les types peuvent être numériques (INT, DECIMAL, REAL), textuels (CHAR(n), VARCHAR(n), TEXT) ou plus spécifiques comme DATE et TIME.

Sur ces types s'appliquent les opérations de comparaisons : <=, <, =, >, >=.

Remarque 17 : Les types DATE et TIME sont des chaines de caractères. La comparaison se fait donc d'après l'ordre lexicographique. Néanmoins, leur format est tel que le résultat est celui attendu.

Remarque 18 : Sur les types numériques, il y a aussi les opérations mathématiques usuelles.

Def 19 : La valeur NULL représente l'absence de valeur.

Propriété 20 : Toute comparaison avec la valeur NULL donnera faux.

Exemple 21 :

```
SELECT *
FROM produits
WHERE NOT (NULL = NULL) AND NOT (NULL <> NULL)
```

renverra une table vide.

Syntaxe 22 : Pour tester si un attribut est à NULL, on utilise IS NULL ou IS NOT NULL.

III.2) Requêter ensemblistes

Les opérations ensemblistes permettent de combiner dans un résultat unique des lignes provenant de deux (ou plus) requêtes. Les lignes peuvent venir de tables différentes mais après projection, on doit obtenir des tables ayant le même schéma de relation.

Les opérateurs ensemblistes sont les suivants :

- UNION : pour obtenir l'union de deux requêtes
- INTERSECT : pour obtenir l'intersection
- EXCEPT : pour obtenir la différence entre deux requêtes

Exemple 23 : Nom de tous les protagonistes de la base

```
SELECT nom FROM produits
UNION
SELECT nom FROM clients
```

Syntaxe 24 : On peut faire un produit cartésien de tables. Avec t_1 JOIN t_2 ON 1=1 ou avec une virgule entre les tables.

Exercice 25 Faire une jointure uniquement à l'aide du produit cartésien et de la clause WHERE.

Solution 26:

```
SELECT p.nom
FROM produit AS p, commander Asc
WHERE p.num_produit= c.num_produit
```

Syntaxe 27 : A mi-chemin entre le produit cartésien et la jointure, il y a les jointures externes.
Dans une jointure externe, l'une des tables utilisées est directrice : tous les renseignements seront présents dans le résultat, même s'ils n'ont pas de valeurs correspondantes avec les autres tables de la jointure. Les colonnes de l'autre table seront alors remplies de NULL.
Les ordres suivant sont utilisés dans les jointures externes:

* LEFT JOIN : retourne tous les n-uplets de la table gauche
* RIGHT JOIN : " " " " " " " droite
* FULL JOIN : Union du left join et du right join

Syntaxe 28 : Dans une clause de condition, le mot clef IN suivi d'une Requête permet de savoir si l'attribut est présent dans la requête.

Remarque 29 : On appelle cela une requête imbriquée.

Exemple 30 :

```
SELECT ville
FROM clients
WHERE num_client IN
    (SELECT num_client FROM commande)
```

III.3) Requêtes agrégatives

Def 31 : Un agrégat est un partitionnement horizontal d'une table en sous-table en fonction des valeurs d'un ou plusieurs attributs de partitionnement.

Schéma 32 :

1	"pomme"
2	"poulet"
1	"navet"
2	"fraise"

→

1	"pomme"
	"navet"
2	"poulet"
	"fraise"

Syntaxe 33 : GROUP BY attribut crée cette agrégation

On peut alors utiliser des fonctions statistiques qui seront appliquées à chaque agrégat :

- COUNT(*) : compte le nombre de valeurs de la sous-table
- SUM(attribut) : somme les valeurs de attribut de la sous-table
- AVG(attribut) : fait la moyenne
- MAX (MIN) (attribut)

Syntaxe 34 : Pour filtrer les sous tables qu'on veut garder, on utilise HAVING condition sous le GROUP BY

Exemple 35 : Liste des villes ayant au moins 5 clients

```
SELECT ville
FROM clients
GROUP BY ville
HAVING COUNT(*) >= 5
```

Développement 2 : Requêtes avancées, avec ou sans agrégation

Ne rien écrire hors du cadre.