Exercices : thème 2 - Question 4 (Partie 2)

Question 4 (Partie 2) : Comment peut-on produire de l'information à partir de données contenues dans une base ?

Au travers de la partie 1 (Question 4), nous nous sommes fait une première idée de ce que pouvait être une base de données : une base de données est constituée de tables. Une table est constituée de champs. Chaque table a une clef primaire et peut avoir aucune, une ou plusieurs clefs étrangères.
Avec ce bagage en tête, nous allons à présent apprendre comment créer des bases de données et comment les manipuler à l'aide du langage SQL.

Exercice 1 : introduction

Le langage SQL	Le SQL est le langage que vous utiliserez cette année pour créer (langage de définition de données) et manipuler (langage de manipulation des données) les données de bases de données.

Au travers de cet exercice, nous considérerons la base de données de « gestion des commandes » dont le schéma relationnel est le suivant :

Client(Num, Civilite, Prenom, Nom, Adresse, CodePostal, Ville, Pays)
Clef primaire : Num
Commande(Num, #NumClient, Emission)
Clef primaire : Num
Clef étrangère : NumClient en référence au champ Num de la relation Client
Autre : Emission correspond à la date de la commande
Produit(Code, Libelle, Prix)
Clef primaire : Code
Autre : le champ Prix est un nombre décimal à 6 chiffres, dont 2 après la « virgule ».
Ligne(Num, #NumCommande, #CodeProduit, Quantite)
Clef primaire : Num
Clefs étrangères :
- NumCommande en référence au champ Num de la relation Commande ;
- CodeProduit en référence au champ Code de la relation Produit.

Questions :

1.1. Quelles sont les tables ?

D'après le schéma relationnel, les tables sont « Client », « Commande », « Produit » et « Ligne ».

1.2. Quelle est la signification de la relation Ligne ?

La table ligne représente les lignes des commandes. Elle permet de lister les produits commandés au travers d'une commande, c'est-à-dire de faire la relation entre « Produit » et « Commande » :

- Un produit peut figurer dans aucune à plusieurs commandes ;
- Une commande comporte aucun à plusieurs produits.

Exercice 2 : création de tables

Les types de données

Nous l'avons vu, chaque champ d'une table a un type. Les différents types disponibles sur les SGBDR (dont MySQL, SGBD que nous utiliserons) sont entre autres les suivants :

Type	Type SQL	Description
Alphanumérique	CHAR(n)	Chaîne de caractères de longueur fixes (n caractères)
	VARCHAR(n)	Chaîne de caractères de longueur variable (n car. max)
Numérique	INTEGER	Entier de -2^{31} à $2^{31}-1$ (entier sur 32 bits)
	FLOAT	Décimal (parties entière et décimale séparée par un point)
	DECIMAL(n[,d])	Décimal à n chiffres dont d décimales (d est facultatif)
Date/heure	DATE	Date sous la forme AAAA-MM-JJ
	TIME	Heure sous la forme hh:mm:ss.ns
	TIMESTAMP	Date et heure
Booléen	BOOLEAN	0 ou False pour FAUX, 1 ou True pour VRAI

La création de tables

Créer une table consiste à préciser le nom de la table, à lister les champs de la table avec leur type, puis à préciser la clef primaire et les éventuelles clefs étrangères.

Par exemple, pour créer les tables Client et Commande, on utilisera les requêtes SQL suivantes :

```
CREATE TABLE Client(
    Num INTEGER auto_increment,
    Civilite VARCHAR(3) NOT NULL,
    Prenom VARCHAR(30) NOT NULL,
    Nom VARCHAR(30) NOT NULL,
    Adresse VARCHAR(80) NOT NULL,
    CodePostal VARCHAR(5) NOT NULL,
    Ville VARCHAR(30) NOT NULL,
    PRIMARY KEY (Num)
) ;
```

```
CREATE TABLE Commande(
    Num INTEGER auto_increment,
    NumClient INTEGER NOT NULL,
    Emission DATE NOT NULL,
    PRIMARY KEY (Num),
    FOREIGN KEY (NumClient)
        REFERENCES Client(Num)
) ;
```

Commentaire :
- les champs Num étant ici les clefs primaires de chacune des deux tables, ils sont NOT NULL par convention et il n'est pas utile de le préciser ;
- auto_increment permet de préciser que l'entier est un compteur qui augmentera tout seul à chaque insertion de ligne (=enregistrement) dans la table ;
- le champ NumClient est et doit être du même type que la clef primaire à laquelle il fait référence.

Questions :

2.1. En vous inspirant des deux exemples de requête fournis ci-avant, rédiger les requêtes de création des tables Produit et Ligne.
Voir page suivante.

2.2. Que se passerait-il si la champ NumClient pouvait être NULL ?
Si le champ NumClient de la table Commande pouvait être « NULL », cela signifierait qu'une commande

peut n'avoir été passée par aucun client.

```
CREATE TABLE Produit(
        Code VARCHAR(7),
        Libelle VARCHAR(80) NOT NULL,
        Prix DECIMAL(6,2) NOT NULL,
        PRIMARY KEY (Code)
) ENGINE = InnoDB;
```

```
CREATE TABLE Ligne(
        Num INTEGER auto_increment,
        NumCommande INTEGER NOT NULL,
        CodeProduit VARCHAR(7) NOT Null,
        Quantite INTEGER NOT NULL,
        PRIMARY KEY (Num),
        FOREIGN KEY (NumCommande)
           REFERENCES Commande(Num),
        FOREIGN KEY (CodeProduit)
           REFERENCES Produit(Code)
) ENGINE = InnoDB;
```

Commentaires :

- ***DECIMAL(6, 2) :*** *permet de définir (type) un nombre décimal à 6 chiffres dont 2 après la virgule, soit de -9999.99 à 9999.99.*
- ***auto_increment :*** *permet d'avoir un champ de type « compteur automatique », c'est-à-dire un nombre entier qui augmente tout seul de 1 en 1 à chaque insertion. On ne peut utiliser « auto_increment » que sur la clef primaire.*

Exercice 3 : modification de tables

Une fois créée, la structure de la table peut encore être modifiée, voire la table peut être supprimée. Pour ce faire, on utilisera les requêtes SQL suivantes :

Suppression d'une table	**DROP TABLE** nomTable ;
Ajout d'un champ	**ALTER TABLE** nomTable **ADD** champAjouté type[, …] ;
Ajout d'une clef primaire	**ALTER TABLE** nomTable **ADD PRIMARY KEY**(nomChamp1[, …]) ;
Ajout d'une clef étrangère	**ALTER TABLE** nomTable **ADD FOREIGN KEY** (nomChamp1[, …]) **REFERENCES** nomTablePointée(nomChampPointé1[, …]) ;
Suppression d'un champ	**ALTER TABLE** nomTable **DROP** champSupprimé[, …] ;
Suppression de la clef primaire	**ALTER TABLE** nomTable **DROP PRIMARY KEY** ;
Suppression d'une clef étrangère	**ALTER TABLE** nomTable **DROP CONSTRAINT** nomContrainte ; *Commentaire : lorsqu'une clef étrangère est créée, un nom de contrainte lui est attribué. Pour la supprimer cette clef étrangère, il faut utiliser ce nom.*

Questions :

3.1. En vous aidant du tableau ci-dessus, rédiger la requête SQL permettant d'ajouter un champ Email à la table Client. Mettre à jour (en rouge) le schéma relationnel.
ALTER TABLE Client **ADD** Email VARCHAR(100) ;
N.B. : on choisit ici que l'adresse email soit une chaîne de caractères comportant au maximum 100 caractères, ce qui semble suffisant.

3.2. On souhaite à présent pouvoir ajouter une description complète de chaque produit. Proposer une évolution de la base de données et rédiger la requête SQL permettant cette évolution. Enfin, mettre à jour (en rouge) le schéma relationnel.

Requêtes SQL
ALTER TABLE Produit **ADD** Description TEXTE ;
N.B. : on choisit ici d'utiliser un champ de type TEXT, type permettant de stocker une très longue chaîne de caractères.

Corrections du schéma relationnel
Client(Num, Civilite, Prenom, Nom, Adresse, CodePostal, Ville, Pays, Email)
Produit(Code, Libelle, Prix, Description)

3.3. Après création de la base de données, on s'aperçoit que la mise en place de la clef étrangère de la table Commande a été omise. Rédiger la requête SQL permettant de corriger cette erreur.
ALTER TABLE Commande **ADD FOREIGN KEY** (NumClient) **REFERENCES** Client(Num) ;

3.4. La société utilisant cette base de données n'a que des clients français. Dès lors, il apparaît que le champ Pays de la table Client est inutile. Ecrire le requête SQL permettant de le supprimer. Mettre à jour le schéma relationnel (en rouge toujours).
ALTER TABLE Client **DROP** Pays ;

Exercice 4 : insertion, modification et suppression d'enregistrements

On rappelle qu'on appelle « enregistrements » les lignes d'une table. A cet égard, le langage SQL permet d'insérer de nouveaux enregistrements, de modifier des enregistrements existant mais encore de supprimer des enregistrements. Le format des requêtes SQL est le suivant :

Insertion	**INSERT INTO** nomTable[(champ1, champ2, …)] **VALUES** (valeur1, valeur2, …), -- 1ère ligne ajoutée (valeur1, valeur2, …), -- 2ème ligne ajoutée … (valeur1, valeur2, …) ; -- dernière ligne ajoutée
Modification	**UPDATE** nomTable **SET** champModifié = nouvelleValeur [, …] [**WHERE** condition1 **AND**/**OR** condition2 …] ;
Suppression	**DELETE FROM** nomTable [**WHERE** condition1 **AND**/**OR** condition2 …] ;

Bien entendu, rien ne vaut quelques exemples :

Ajout d'un nouveau client
INSERT INTO Client(Civilite, Nom, Prenom, Adresse, CodePostal, Ville, Email) **VALUES** ("M.", "Washington", "Denzel", "Av. de Wagram", "75000", "PARIS", "dw@gmail.com") ; Autre possibilité : **INSERT INTO** Client **VALUES** (NULL, "M.", "Denzel", "Washington", "Av. de Wagram", "75000", "PARIS", "dw@gmail.com") ; *Commentaire : si on décide de ne pas préciser la liste des champs, les valeurs doivent être précisée dans l'ordre des champs de la table.*
Mise à jour du prix du produit n°1
UPDATE Produit **SET** Prix = 75.32 **WHERE** Num = 1
Augmentation de 10% du prix des produits
UPDATE Produit **SET** Prix = Prix * 1.1
Suppression du produit n°5
DELETE FROM Produit **WHERE** Num = 5
Suppression des produits dont le prix excède 1000€
DELETE FROM Produit **WHERE** Prix > 1000
Suppression des clients dont le prénom commence par la letter B
DELETE FROM Produit **WHERE** Prenom **LIKE** "B%" *Commentaire : le symbole « % » dans la chaîne « B% » permet d'indiquer que le prénom peut commencer par la lettre « B » suivie de n'importe quels caractères. L'opérateur de comparaison « LIKE » ressemble à un « = » mais permet de plus d'effectuer ce genre de comparaisons approximatives.*

Questions :

4.1. Rédiger la requête qui permet de vous ajouter en tant que client.

```
INSERT INTO Client(Civilite, Nom, Prenom, Adresse, CodePostal, Ville, Email)
VALUES ("M.", "PAQUEREAU", "Jimmy", "Bd des codeurs", "37260", "VILLEPERDUE", "jp@gmail.com") ;
```

4.2. Rédiger la requête qui permet d'ajouter deux ordinateurs, un « Ordinateur Asus trop bien » à 1150€ ainsi qu'un « Ordinateur AlienWare un peu cher » à 2500€.

```
INSERT INTO Produit(Code, Libelle, Prix, Description)
VALUES
("ORDIASUS", "Ordinateur Asus trop bien", 1150.00, "La description de l'ordi. Asus…"),
("ORDIALIEN", "Ordinateur AlienWare un peu cher", 2500.00, "La description de l'ordi. AlienWare…") ;
```

4.3. Rédiger la requête permettant de supprimer les clients des Yvelines (78).

```
DELETE FROM Client
WHERE CodePostal LIKE "78%" ;
```

4.4. Rédiger la requête permettant de modifier le client Grégoire LEMARCHE dont le nom a été mal saisi (LEMACHIN a été saisi au lieu de LEMARCHE).

```
UPDATE Client
SET Nom = "LEMARCHE"
WHERE Nom = "LEMACHIN"
AND Prenom = "Grégoire" ;
```

4.5. Rédiger la requête permettant de supprimer les commandes antérieures aux 31/12/2016. Pourquoi cette requête échouera ?

```
DELETE FROM Commande
WHERE Emission <= "2016-12-31" ;
```

Cette requête échouera car les commandes antérieures au 31/12/2016 peuvent comporter des lignes. Si celles-ci comportent des lignes, en cas de suppression, les lignes pointeraient sur des commandes qui n'existent plus, ce qui pose un problème de contrainte d'intégrité (dans la table ligne, présence d'une clef étrangère faisant référence à la commande). Une erreur sera donc levée.

Exercice 5 : récupérer/projeter des données

Le cœur du SQL, pourrait-on dire, réside dans les requêtes que nous allons étudier dans cette cinquième partie : les requêtes SELECT. On parle de requêtes de projection, et souvent, à tort, de sélection.
Ces requête SQL permettent en quelque sorte d'afficher/retourner des données. Elles ont la syntaxe suivante :

Projection	**SELECT** champProjeté1 [, …] -- projection de champs **FROM** table1 [, …] -- tables utilisées **WHERE** condition1 [**AND**/**OR** condition2 …] -- restrictions **GROUP BY** champRegroupement1 [, …] -- regroupements sur les champs **HAVING** condition1 [**AND**/**OR** condition2 …] -- restrictions après regroupements **ORDER BY** champTri **ASC**/**DESC** [, …] -- tri croissant/décroissant sur les champs *Commentaire :* *- les clauses SQL (SELECT, FROM, WHERE, etc.) doivent, si elles figu-rant dans la requête, figurer dans cet ordre ;* *- seules les clauses SELECT et FROM sont obligatoires (en fait, seule la clause SELECT l'est réellement).*
Restrictions	Les critères (≈conditions) de restriction la clause WHERE (resp. HAVING) permettent de limiter les lignes retourner. Ces critères permettent de ne retourner que les lignes vérifiant les conditions précisées dans le WHERE (resp. HAVING). Il s'agit de conditions de la forme : expression1 operateur expression2. Les opérateurs disponibles sont : ○ les opérateurs de comparaison habituels : =, >, <, >=, <=, <> ; ○ les opérateurs propres au SQL : - expr **BETWEEN** valeur1 **AND** valeur2 : champ/expression comprise entre 2 valeurs - expr **LIKE** chaine : sorte d'égalité permettant d'utiliser les caractère « % » (n'importe quels caractères) et « _ » (n'importe quel caractère) pour préciser des conditions telles que : champ/expression commençant par…, contenant… ou terminant par… - expr **IN** (valeur1, valeur2[, …]) : champ/expression faisant partie de l'une des valeurs listées.

Comme de coutumes, quelques exemples vaudront mieux que de longs discours :

Liste de tous les produits (tous les champs sont projetés)
SELECT * **FROM** Produit
Liste des produits (code et libellé) dont le prix est supérieur à 1000€
SELECT code, libelle **FROM** Produit **WHERE** Prix > 1000
Liste des produits dont le prix est compris entre 1000€ et 2000€
SELECT * **FROM** Produit **WHERE** Prix **BETWEEN** 1000 **AND** 2000
Commandes (numéro) passées le 13 mars 2016
SELECT num **FROM** Commande **WHERE** Emission = "2016-03-13"

Commandes passées le 13 ou 14 mars 2016
SELECT * **FROM** Commande **WHERE** Emission = "2016-03-13" **OR** Emission = "2016-03-14" **SELECT** * **FROM** Commande **WHERE** Emission **BETWEEN** "2016-03-13" **ABD** Emission = "2016-03-14" **SELECT** * **FROM** Commande **WHERE** Emission **IN** ("2016-03-13", "2016-03-14")
Listes des produits par ordre alphabétique sur les libellés
SELECT * **FROM** Produit **ORDER BY** Libelle ASC **SELECT** * **FROM** Produit **ORDER BY** Libelle -- le tri est croissant par défaut
Listes des commandes de l'année 2016
SELECT * **FROM** Commande **WHERE** YEAR(Emission) = 2016 **ORDER BY** Libelle ASC **SELECT** * **FROM** Produit **ORDER BY** Libelle -- le tri est croissant par défaut

Questions :

5.1. Récupérer le détail de la commande 58.
SELECT * **FROM** Commande **WHERE** Num = 58 ;

5.2. Lister les produits par ordre décroissant de prix.
SELECT * **FROM** Produit **ORDER BY** Prix **DESC** ;

5.3. Lister les commandes de l'année 2015 par date décroissante.
SELECT * **FROM** Commande **WHERE** Year(Emission) = 2015 ;
Alternatives :
SELECT * **FROM** Commande **WHERE** Emission **BETWEEN** "2015-01-01" **AND** "2015-12-31" ;
SELECT * **FROM** Commande **WHERE** Emission >= "2015-01-01" **AND** Emission <= "2015-12-31" ;

5.4. Lister les commandes du mois de janvier 2016.
SELECT * **FROM** Commande **WHERE** Year(Emission) = 2016 **AND** Month(Emission) = 1 ;
Alternative :
SELECT * **FROM** Commande **WHERE** Emission **BETWEEN** "2016-01-01" **AND** "2016-01-31" ;

5.5. Lister les commandes ordonnées par années croissantes.
SELECT * **FROM** Commande **ORDER BY** Year(Emission) **ASC** ;
Plus simplement :
SELECT * **FROM** Commande **ORDER BY** Year(Emission) ;

Exercice 6 : les jointures

Les jointures interviennent lorsque l'on souhaite récupérer des données en provenance de plusieurs tables. La jointure permet en quelque sorte de préciser au SGBD comment mettre en relation les données. S'agissant de « mettre en relation » les données, on comprendra qu'il s'agit de mettre en relation la clef étrangère d'une table avec la clef primaire d'une autre.

Exemple de jointure :

Liste de toutes les commandes avec le prénom et le nom du client.

SELECT Commande.*, Client.Prenom, Client.Nom
FROM Commande, Client
WHERE Commande.NumClient = Client.Num ;

Commentaire :
- cette requête (avec jointure) comporte deux tables dans le FROM ;
- l'égalité Commande.NumClient = Client.Num permet d'effectuer la fameuse jointure ;
- Commande. permet de projeter tous les champs de la table Commande ;*
- les champs sont préfixés par le nom de la table suivi d'un point pour éviter les ambiguïtés et bien savoir de quel table provient chaque champ.

Questions :

6.1. Récupérer le détail de la commande 58 avec l'adresse e-mail du client qui a passé la commande.
SELECT Commande.*, Client.Email
FROM Commande, Client <<< on a besoin de récupérer des information (SELECT) provenant de 2 tables
WHERE Commande.NumClient = Client.Num <<< jointure entre Commande et Client
AND Commande.Num = 58 ; <<< finalement, seul la commande 58 nous intéresse ici

6.2. Récupérer le détail de la commande 35 avec ses lignes de commandes.
SELECT *
FROM Commande, Ligne
WHERE Ligne.NumCommande = Commande.Num <<< jointure entre Ligne et Commande
AND Commande.Num = 35 ;

6.3. Récupérer les ligne de commande de la commande 5 avec le libellé et le prix du produit commandé.
SELECT Ligne.*, Produit.Libelle, Produit.Prix
FROM Ligne, Produit
WHERE Ligne.CodeProduit = Produit.Code <<< jointure entre Ligne et Produit
AND Ligne.NumCommande = 5 ;

6.4. Récupérer le détail de la commande 47 avec ses lignes de commandes, et pour chaque ligne de commande le libellé et le prix du produit commandé.
SELECT Commande.*, Ligne.*, Produit.Libelle, Produit.Prix
FROM Commande, Ligne, Produit
WHERE Ligne.NumCommande = Commande.Num <<< jointure entre Ligne et Commande
AND Ligne.CodeProduit = Produit.Code <<< jointure entre Ligne et Produit
AND Commande.Num = 47 ;

6.5. Compléter la requête 6.4. afin d'afficher (calcul) le prix de chaque lignes de la commande 47.

```
SELECT Commande.*, Ligne.*, Produit.Libelle, Produit.Prix, Ligne.Quantite * Produit.Prix AS PrixTotal
FROM Commande, Ligne, Produit
WHERE Ligne.NumCommande = Commande.Num <<< jointure entre Ligne et Commande
AND Ligne.CodeProduit = Produit.Code <<< jointure entre Ligne et Produit
AND Commande.Num = 47 ;
```

*Commentaire : l'opération « Ligne.Quantite * Produit.Prix » permet de calculer le prix total de chaque ligne en multiplié la quantité de produit commandée par le prix du produit. Le « **AS** PrixTotal » permet de donner un nom/surnom à la colonne, ce qu'on appelle un alias.*

Exercice 7 : les agrégats

Les agrégats permettent de projeter des données calculées et ainsi de de répondre à des questions comme : quel est le nombre de … ? Quelle est la moyenne de … ? Etc. Les agrégats figurent dans la clauses SELECT :

COUNT(expression)	Permet de compter un nombre d'enregistrements
SUM(expression)	Permet de calculer la somme de valeurs numériques
MIN(expression)	Permet de trouver le minimum parmi des valeurs numériques
MAX(expression)	Permet de trouver le maximum parmi des valeurs numériques
AVG(expression)	Permet de calculer la moyenne de valeurs numériques

Exemples :

Afficher le nombre de produits ou quel est le nombre de produit ?
SELECT count(Num) **FROM** Produit ;
Afficher le prix du produit le plus cher ou quel est le prix du produit le plus cher ?
SELECT max(Prix) **FROM** Produit ;

Les deux exemples précédents ne retournent qu'une ligne. La clause GROUP BY permet de répondre à des questions plus complexes en regroupant des enregistrements ensemble. Les agrégats sont alors calculés par groupes d'enregistrements. La clause HAVING se comporte comme la clause WHERE à ceci près qu'elle permet d'effectuer une restriction en fonction des agrégats, à savoir une fois que ces derniers ont été calculés.

Exemples :

Quel est le nombre de commande par année ?
SELECT count(Num) **FROM** Commande **GROUP BY** YEAR(Emission) ;
Quel est le nombre de lignes par commande ?
SELECT count(Num) **FROM** Ligne **GROUP BY** NumCommande ;
Afficher le nombre de produits commandés de chaque commande.
SELECT sum(quantite) **FROM** Ligne **GROUP BY** NumCommande **;**
Afficher les années où le nombre de commande a été supérieur à 1000.
SELECT Year(Emission) **FROM** Commande **GROUP BY** YEAR(Emission) **HAVING** count(Num) > 1000 ;

Questions :

7.1. Quel est le nombre de clients ?
SELECT COUNT(*) **AS** NbClients **FROM** Client ;

7.2. Afficher le nombre de clients par ville trié par nom de ville croissant.

```
SELECT Ville, COUNT(*) AS NbClients
FROM Client
GROUP BY Ville
ORDER BY Ville ;
```

7.3. Quel est le nombre de commandes par mois et par année ?

```
SELECT Year(Emission), Month(Emission), COUNT(*) AS NbCommandes
FROM Commande
GROUP BY Year(Emission), Month(Emission) ;
```

7.4. Quel est le nombre de commandes par ville ?

```
SELECT Ville, COUNT(*) AS NbCommandes
FROM Commande, Client
WHERE Commande.NumClient = Client.Num
GROUP BY Client.Ville ;
```

7.5. Quel est le prix moyen d'un produit ?

```
SELECT AVG(Prix)
FROM Produit ;
```

7.6. Quel est le prix total de la commande 58 ?

```
SELECT SUM(Ligne.Quantite * Produit.Prix) AS PrixTotal
FROM Commande, Ligne, Produit
WHERE Ligne.NumCommande = Commande.Num
AND Ligne.CodeProduit = Produit.Code
AND Commande.Num = 58 ;
```

7.7. Quel est le chiffre d'affaires réalisé sur l'année 2016 ?

```
SELECT SUM(Ligne.Quantite * Produit.Prix) AS ChiffreAffaires
FROM Commande, Ligne, Produit
WHERE Ligne.NumCommande = Commande.Num
AND Ligne.CodeProduit = Produit.Code
AND Year(Commande.Emission) = 2016 ;
```

7.8. Quel est le chiffre d'affaires réalisé par client ?

```
SELECT Client.Email, SUM(Ligne.Quantite * Produit.Prix) AS ChiffreAffaires
FROM Commande, Ligne, Produit, Client
WHERE Ligne.NumCommande = Commande.Num
AND Ligne.CodeProduit = Produit.Code
AND Commande.NumClient = Client.Num
GROUP BY Client.Email ;
```

N.B. : on a choisi ici de supposer que les adresses emails des clients sont uniques.

Exercice 8 : les sous-requêtes

Les requêtes peuvent elles-mêmes utiliser des requêtes, imbriquées dans la requête principale. Les requêtes imbriquées sont qualifiées de sous-requêtes. En particulier, les sous-requêtes sont particulièrement utilisées pour effectuer des restrictions avancées.

Exemples :

Commandes passées par des clients de STRASBOURG

Avec jointure :
```
SELECT Commande.*
FROM Commande, Client
WHERE Commande.NumClient = Client.Num
AND Client.Ville = "STRASBOURG" ;
```

Avec sous-requête :
```
SELECT *
FROM Commande
WHERE Commande.NumClient IN (
    SELECT Num
    FROM Client
    WHERE Ville = "STRASBOURG"
) ;
```

Produits dont le prix est supérieur à la moyenne

```
SELECT *
FROM Produit
WHERE Prix > (
    SELECT AVG(Prix) FROM Produit
) ;
```

Commande(s) dont la date est supérieure à celle de toutes les autres (=dernières commandes)

```
SELECT *
FROM Commande
WHERE Emission > ALL (
    SELECT Emission FROM Commande
) ;
```

Remarque : il faut bien souvent essayer de décomposer un problème en sous-problèmes plus simples. En ce sens, une sous-requête répond à un sous-problème plus simple.

Questions :

8.1. Quel est le produit le produit le moins cher ? Quel est le plus cher ?

Produit le moins cher :
```
SELECT Produit.*
FROM Produit
WHERE Prix = ( SELECT MIN(Prix) FROM Produit ) ;
```
Produit le plus cher :
```
SELECT Produit.*
FROM Produit
WHERE Prix = ( SELECT MAX(Prix) FROM Produit ) ;
```

8.2. Quel est ou quels sont les clients à avoir passé le plus de commandes ?

```
SELECT Client.Num, Client.Email
FROM Client, Commande
WHERE Commande.NumClient = Client.Num
GROUP BY Client.Num
HAVING COUNT(*) >= ALL ( <<< Voir commentaire ci-dessous
    SELECT COUNT(*) FROM Commande GROUP BY  NumClient <<< Nombre de commandes par client
) ;
```

Commentaire : la question posée peut être reformulée « on cherche le(s) client(s) dont le nombre de commandes est supérieur ou égal au nombre de commande que chaque client a passé ».

Exercice 9 : les droits sur les bases de données

Très souvent, les bases de données sont partagées entre plusieurs utilisateurs. Pour des raisons de sécurité, il convient de restreindre les droits des utilisateurs de sorte qu'ils ne puissent effectuer que les manipulations dont ils ont besoin. Les utilisateurs sont typiquement regroupés par profiles d'utilisateurs ayant les mêmes besoins (exemple : développeur, administrateur et utilisateur final). Finalement, le SQL permet de configurer les droits d'accès des utilisateurs d'une base de données de sorte qu'un utilisateur puisse seulement :

- consulter des enregistrements (**SELECT**) ;
- et/ou ajouter des enregistrements (**INSERT**) ;
- et/ou mettre à jour des enregistrement (**UPDATE**) ;
- et/ou supprimer des enregistrement (**DELETE**) ;
- et/ou créer des tables/vues (**CREATE**) ;
- et/ou modifier des tables/vues (**ALTER**) ;
- et/ou supprimer des tables/vues (**DROP**).

Ajout d'un droit d'accès	**GRANT** droit1 [, …] **ON** nomTable **TO** nomUtilisateur [, …] ;
Révocation d'un droit d'accès	**REVOKE** droit1 [, …] **ON** T_CHAMBRE **FROM** nomUtilisateur [, …] ;

Exemples :

Autorise le profil « CLIENT » à consulter et à ajouter des commandes
GRANT SELECT, **INSERT ON** Commande **TO** CLIENT **;**
Révoque l'autorisation accordée au profil « CLIENT » d'ajouter des commandes
REVOKE INSERT ON Commande **TO** CLIENT **;**

Questions :

La base de données est interconnectée à un logiciel (un portail extranet) au moyen duquel les clients passent leurs commandes et peuvent consulter les commandes qu'ils ont passées. C'est la secrétaire qui enregistre les clients. Pour ce faire, elle consulte la liste des clients puis ajoute un nouveau client. Sur demande expresse, il arrive à celle-ci de supprimer les clients. Ce sont en revanche les commerciaux qui gèrent le catalogue de produits. Finalement, seule le responsable commercial peut modifier ou supprimer une commande passée.

9.1. Identifier les profils d'utilisateurs.
Les profils d'utilisateurs sont ici : « client », « secrétaire », « commercial » et « responsable commercial ».

9.2. Pour chaque profil d'utilisateurs, déterminer les droits d'accès dont il a besoin.

Profil	Table « **Client** »	Table« **Produit** »	Table « **Commande** »	Table « **Ligne** »
CLIENT	---	---	SELECT, INSERT	SELECT, INSERT
SECRETAIRE	SELECT, INSERT, DELETE			
COMMERCIAL		SELECT, INSERT, UPDATE, DELETE		
RESPONSABLE			SELECT, UPDATE, DELETE	SELECT, UPDATE, DELETE

9.3. Rédiger les requêtes SQL permettant d'octroyer les droits d'accès nécessaires à un profil d'utilisateur (au choix).

```
-- Profil CLIENT
GRANT SELECT, INSERT ON Commande TO CLIENT ;
GRANT SELECT, INSERT ON Ligne TO CLIENT ;
-- Profil SECRETAIRE
GRANT SELECT, INSERT, DELETE ON Client TO SECRETAIRE ;
-- Profil COMMERCIAL
GRANT SELECT, INSERT ON Produit TO COMMERCIAL ;
-- Profil RESPONSABLE
GRANT SELECT, UPDATE, DELETE ON Commande TO RESPONSABLE ;
GRANT SELECT, UPDATE, DELETE ON Ligne TO RESPONSABLE ;
```