Chapitre 7
Le Langage SQL

7-1- Introduction

SQL (**S**tructured **Q**uery **L**anguage, traduit Langage de requêtes structuré ou langage d'interrogation structuré) est un langage de quatrième génération (**L4G**), non procédural, conçu par IBM dans les années 70. **SQL** est basée sur l'algèbre relationnelle (opérations ensemblistes et relationnelles). SQL a été normalisé dès 1986 mais les premières normes, trop incomplètes, ont été ignorées par les éditeurs de SGBD. La norme actuelle SQL-2 (appelée aussi SQL-92) date de 1992. Elle est acceptée par tous les SGBD relationnels. Ce langage permet l'accès aux données et se compose de quatre sous-ensembles :

❶ Le **L**angage d'**I**nterrogation de **D**onnées : **LID** : Ce langage permet de rechercher des informations utiles en interrogeant la base de données. Certains considèrent ce langage comme étant une partie du LMD.

❷ Le **L**angage de **M**anipulation de **D**onnées : **LMD** (**D**ata **M**anipulation **L**anguage **DML**) : Ce langage permet de manipuler les données de la base et de les mettre à jour.

❸ Le **L**angage de **D**éfinition de **D**onnées : **LDD** (**D**ata **D**efinition **L**anguage **DDL**) : Ce langage permet la définition et la mise à jour de la structure de la base de données (tables, attributs, vues, index, ...).

❹ Le **L**angage de **C**ontrôle de **D**onnées : **LCD** (**D**ata **C**ontrol **L**anguage **DCL**) : Ce langage permet de définir les droits d'accès pour les différents utilisateurs de la base de données, donc il permet de gérer la sécurité de la base et de confirmer et d'annuler les transactions.

SQL[1]			
LDD	**LMD**	**LID**	**LCD**
Create Alter Drop	Insert Update Delete	Select	Grant Revoke

Exemple : Soit la base de données relationnelle (BD commerciale) suivante :

Produit (NP, LibP, Coul, Poids, PU, Qtes) - *Désigne l'ensemble des produits.*
Client (NCl, NomCl, AdrCl) - *Désigne l'ensemble des clients.*
Commande (NCmd, DateCmd, #NCl) - *Désigne l'ensemble des commandes.*
Ligne_Cmd (#NCmd, #NP, Qte) - *Désigne l'ensemble des lignes de commandes.*

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

[1] *SQL ne fait pas la différence entre majuscules et minuscules*

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

Toutes les manipulations concernant les commandes SQL seront faites sur cette base de données.

7-2- Langage d'Interrogation de Données

7-2-1- Syntaxe générale

Le **SQL** est à la fois un langage de manipulation de données et un langage de définition de données. Toutefois, la définition de données est l'oeuvre de l'administrateur de la base de données, c'est pourquoi la plupart des personnes qui utilisent le langage SQL ne se servent que du langage de manipulation de données et plus précisément du langage d'interrogation des données, permettant de sélectionner les données qui les intéressent.
La principale commande du langage d'interrogation de données est la commande **SELECT**.
La commande **SELECT** est basée sur l'algèbre relationnelle, en effectuant des opérations de sélection de données sur plusieurs tables relationnelles par projection. Sa syntaxe générale est la suivante :

SELECT[2] **[ALL] | [DISTINCT]** <*liste des attributs*> | ***
FROM[3] <*liste des tables*>
[WHERE <*condition*>**]**
[GROUP BY <*liste des attributs*>**]**
[HAVING <*condition de groupement* >**]**
[ORDER BY <*liste des attributs de tri*>**]** **;**

- […] : le contenu entre crochet est facultatif.
- L'option **ALL** est, par opposition à l'option **DISTINCT**, l'option par défaut. Elle permet de sélectionner l'ensemble des lignes satisfaisant à la condition logique.
- L'option **DISTINCT** permet de ne conserver que des lignes distinctes, en éliminant les doublons.
- La *liste des attributs* indique la liste des attributs choisis, séparés par des virgules. Lorsque l'on désire sélectionner l'ensemble des colonnes d'une table il n'est pas nécessaire de saisir la liste de ses attributs, l'option * permet de réaliser cette tâche.
- La *liste des tables* indique l'ensemble des tables (séparées par des virgules) sur lesquelles on opère.
- La *condition* permet d'exprimer des qualifications complexes à l'aide d'opérateurs logiques et de comparateurs arithmétiques.

[2] Permet d'indiquer quelles colonnes ou quelles expressions doivent être retournées par l'interrogation.
[3] Spécifie les tables participant à l'interrogation.

7-2-2- Projection

Une projection est une instruction permettant de sélectionner un ensemble d'attributs dans une table.

Syntaxe :
SELECT [ALL] | [DISTINCT] <*liste des attributs*> | ***
FROM <*la table***>** ;

Applications :

✓ Donner la liste des clients.

SELECT *
FROM Client ;

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

✓ Donner l'ensemble des produits qui ont été commandés (NP seulement).

SELECT NP
FROM Ligne_Cmd;

SELECT DISTINCT NP
FROM Ligne_Cmd;

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

NP
P001
P004
P006
P002
P007
P001
P002
P004
P005
P008
P001
P002

NP
P001
P004
P006
P002
P007
P005
P008

DISTINCT est utilisé pour éliminer les duplications, par défaut on obtient tous les tuples (**ALL**).

7-2-3- Restrictions

Une restriction consiste à sélectionner les lignes satisfaisant à une condition logique effectuée sur leurs attributs. En SQL, les restrictions s'expriment à l'aide de la clause **WHERE** suivie d'une condition logique exprimée à l'aide d'opérateurs logiques (**AND**, **OR**, **NOT**), d'opérateurs

arithmétiques (+, -, *, /, %), de comparateurs arithmétiques (=, !=, >, <, >=, <=) et des prédicats (**NULL**, **IN**, **BETWEEN**, **LIKE**, **ALL**, **SOME**, **ANY**, **EXISTS**). Ces opérateurs s'appliquent aux valeurs numériques, aux chaînes de caractères et aux dates[4].

Syntaxe :

SELECT [ALL] | [DISTINCT] <*liste des attributs*> | *
FROM <*la table*>
WHERE <*condition*> **;**

Au niveau de la clause **WHERE**, les prédicats sont les suivants :

WHERE exp1 = exp2	Condition est vraie si les deux expressions exp1 et exp2 sont égales.
WHERE exp1 != exp2	Condition est vraie si les deux expressions exp1 et exp2 sont différentes.
WHERE exp1 < exp2	Condition est vraie si exp1 est inférieure à exp2.
WHERE exp1 > exp2	Condition est vraie si exp1 est supérieure à exp2.
WHERE exp1 <= exp2	Condition est vraie si exp1 est inférieure ou égale à exp2.
WHERE exp1 >= exp2	Condition est vraie si exp1 est supérieure ou égale à exp2.
WHERE exp1 **BETWEEN** exp2 **AND** exp3	Condition est vraie si exp1 est comprise entre exp2 et exp3, bornes incluses.
WHERE exp1 **LIKE** exp2	Condition est vraie si la sous–chaîne exp2 est présente dans exp1.
WHERE exp1 **NOT LIKE** exp2	Condition est vraie si la sous–chaîne exp2 n'est pas présente dans exp1.
WHERE exp1 **IN** (exp2, exp3,…)	Condition est vraie si exp1 appartient à l'ensemble (exp2, exp3, …).
WHERE exp1 **NOT IN** (exp2, exp3,…)	Condition est vraie si exp1 n'appartient pas à l'ensemble (exp2, exp3, …).
WHERE exp1 **IS NULL**	Condition est vraie si exp1 est nulle.
WHERE exp1 **IS NOT NULL**	Condition est vraie si exp1 n'est pas nulle.
WHERE exp1 Op **ALL** (exp2, exp3,…)	Op représente un opérateur de comparaison (<, >, =, !=, <=, >=) Condition est vraie si la comparaison de l'exp1 est vraie avec toutes les valeurs de la liste (exp2, exp3, …). Si la liste est vide, le résultat est vrai.
WHERE exp1 Op **ANY** (exp2, exp3,…) **WHERE** exp1 Op **SOME** (exp2, exp3,…)	Op représente un opérateur de comparaison (<, >, =, !=, <=, >=) Condition est vraie si la comparaison de l'exp1 est vraie avec au moins une valeur de la liste (exp2, exp3, …). Si la liste est vide, le résultat est faux.
WHERE EXISTS sous-requête	Condition est vraie si le résultat de la sous-requête n'est pas vide.

Remarques :

- L'opérateur **IN** est équivalent à **= ANY**
- L'opérateur **NOT IN** est équivalent à **!= ANY**

[4] Les constantes de type chaîne de caractères et date doivent être encadrées par apostrophes ' … ' contrairement aux nombres.

Applications :

✓ Donner le numéro et le nom des clients de la ville de Sousse.

SELECT NCl, NomCl
FROM Client
WHERE AdrCl = 'Sousse';

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

NCl	NomCl
CL03	AMS
CL04	GLOULOU
CL06	ELECTRON
CL07	SBATIM

✓ Donner la liste des commandes dont la date est supérieure à '01/01/2004'.

SELECT *
FROM Commandes
WHERE DateCmd > '01/01/2004';

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

NCmd	DateCmd	NCl
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C005	11/03/2004	CL03

✓ Donner la liste des produits dont le prix est compris entre 20 et 50.

SELECT *
FROM Produit
WHERE PU **BETWEEN** 20 **AND** 50;

Ou bien

SELECT *
FROM Produit
WHERE (PU >= 20) **AND** (PU <= 50);

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

NP	LibP	Coul	Poids	PU	Qtes
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P006	Serrure	Jaune	2	47.000	1250

✓ Donner la liste des clients dont les noms commencent par 'B'.

SELECT *
FROM Client
WHERE NomCl **LIKE** 'B%';

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL12	BATFER	Tunis

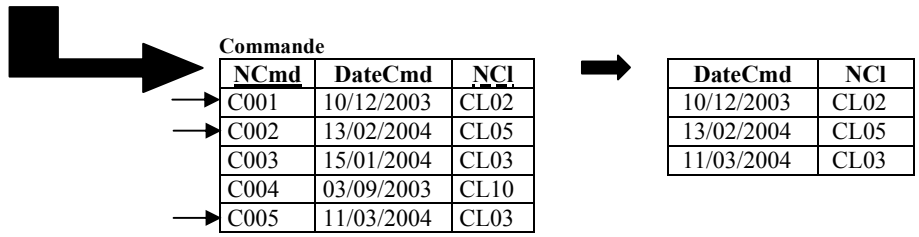
Le prédicat **LIKE** permet de faire des comparaisons sur des chaînes grâce à des caractères, appelés caractères *jokers* :

- Le caractère % permet de remplacer une séquence de caractères (éventuellement nulle).
- Le caractère _ permet de remplacer un caractère.
- Les caractères [-] permettent de définir un intervalle de caractères (par exemple [J-M]).

La sélection des clients dont les noms ont un E en deuxième position se fait par l'instruction : **WHERE** NomCl **LIKE** "_E%"

✓ Donner les numéros des clients dont les dates de leurs commandes se trouvent parmi les dates suivantes : (‘10-12-03’, ‘10-12-04’,’13-02-04’,’11-03-04’).

SELECT DateCmd, NCl
FROM Commande
WHERE DateCmd **IN** (‘10-12-2003’,’10-12-2004’,’13-02-2004’,’11-03-2004’);



Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

DateCmd	NCl
10/12/2003	CL02
13/02/2004	CL05
11/03/2004	CL03

✓ Donner les noms des clients qui n’ont pas d’adresse.

SELECT NomCl
FROM Client
WHERE AdrCl **IS NULL;**

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

NomCl
MBATIM

*Lorsqu'un champ n'est pas renseigné, le SGBD lui attribue une valeur spéciale que l'on note **NULL**. La recherche de cette valeur ne peut pas se faire à l'aide des opérateurs standards, il faut utiliser les prédicats **IS NULL** ou bien **IS NOT NULL**.*

7-2-4- Expressions, alias et fonctions

a. Expression

Les expressions acceptées par SQL portent sur des attributs, des constantes et des fonctions. Ces trois types d’éléments peuvent être reliés par des opérateurs arithmétiques (+, -, *, /). Les expressions peuvent figurer :

✓ En tant que colonne résultat d’un ordre **SELECT**.
✓ Dans une clause **WHERE**.
✓ Dans une clause **ORDER BY** (cf. Tri).
✓ Dans les ordres de manipulation des données (**INSERT**, **UPDATE**, **DELETE** cf. LMD).

b. Alias

Les alias permettent de renommer des attributs ou des tables.

SELECT attr1 **AS** aliasa1, attr2 **AS** aliasa2, …
FROM table1 aliast1, table2 aliast2… ;

Pour les attributs, l’alias correspond aux titres des colonnes affichées dans le résultat de la requête. Il est souvent utilisé lorsqu’il s’agit d’attributs calculés (expression).

NB : Le mot clé **AS** est optionnel.

c. Fonction

Le tableau suivant donne le nom des principales fonctions prédéfinies :

Nom de la fonction	Rôle de la fonction
AVG	Moyenne
SUM	Somme
MIN	Minimum
MAX	Maximum
COUNT (*)	Nombre de lignes
COUNT (Attr)	Nombre de valeurs non nulles de l'attribut
COUNT([DISTINCT] Attr)	Nombre de valeurs non nulles différentes de l'attribut

Applications :

✓ Donner la valeur des produits en stock.

SELECT NP, (Qtes * PU) **AS** "Valeur Totale"
FROM Produit ;

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

NP	Valeur Totale
P001	21600
P002	1500
P003	37500
P004	29700
P005	15600
P006	58750
P007	11000
P008	72000

On peut exprimer cette requête sans spécifier le mot clé **AS** :

SELECT NP, (Qtes* PU) "Valeur Totale"
FROM Produit ;

On ne met les noms des colonnes résultats entre guillemets que s'ils contiennent des espaces.

SELECT NP, (Qtes * PU) Valeur
FROM Produit ;

✓ Donner la moyenne des prix unitaires des produits.

SELECT AVG(PU)
FROM Produit;

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

AVG(PU)
29

7-2-5- Sélection avec jointure

Il s'agit ici de sélectionner les données provenant de plusieurs tables ayant un ou plusieurs attributs communs. Cette jointure sera assurée grâce aux conditions spécifiées dans la clause **WHERE**.

Syntaxe :

SELECT [ALL] | [DISTINCT] <*liste des attributs*> | *
FROM <*liste des tables*>
WHERE Nom_Table1.Attrj = Nom_Table2.Attrj **AND** …
AND <*condition*> ;

Applications :

✓ Donner les libellés des produits de la commande numéro 'C002'.

```
SELECT DISTINCT LibP
FROM Produit, Ligne_Cmd
WHERE Produit.NP = Ligne_Cmd.NP
      AND NCmd='C002';
```

Ou bien

```
SELECT DISTINCT LibP
FROM Produit P, Ligne_Cmd L
WHERE P.NP = L.NP
      AND NCmd='C002';
```

Pour ne pas spécifier à chaque fois le nom complet des tables Produit et Ligne_Cmd dans la condition, on peut renommer ces tables.

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

LibP
Prise
Verrou

On peut exprimer cette jointure autrement (cf. Sous-requêtes) :

```
SELECT LibP
FROM Produit
WHERE NP IN      (SELECT NP
                  FROM Ligne_Cmd
                  WHERE NCmd = 'C002');
```

✓ Donner les produits (toutes les informations) commandés au cours de l'année 2003 et qui sont vendus aux clients de Tunis.

```
SELECT DISTINCT P.NP, LibP, Coul, Poids, PU, Qtes
FROM Produit P, Commande C, Client Cl, Ligne_Cmd L
WHERE P.NP = L.NP
      AND C.NCmd = L.NCmd
      AND Cl.NCl = C.NCl
      AND Cl.AdrCl ='Tunis'
      AND DateCmd BETWEEN '01/01/2003' AND '31/12/2003';
```

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18	1200
P004	Peinture	Blanc	25	33	900
P006	Serrure	Jaune	2	47	1250

✓ Donner les libellés des produits qui ont un prix unitaire supérieur à celui du produit 'Robinet'.

```
SELECT P.LibP
FROM Produit P, Produit P1
WHERE P.PU > P1.PU
      AND P1.LibP ='Robinet';
```

Dans certaines requêtes, on est obligé de renommer soit des tables, soit des attributs (ici les tables).

LibP
Câble
Peinture
Serrure
Fer

7-2-6- Groupement

Il est possible de grouper des lignes de données ayant une valeur commune à l'aide de la clause **GROUP BY** et des fonctions de groupe (cf. Fonction).

Syntaxe :

```
SELECT Attr1, Attr2,…, Fonction_Groupe
FROM Nom_Table1, Nom_Table2,…
WHERE Liste_Condition
GROUP BY Liste_Groupe
HAVING Condition;
```

- La clause **GROUP BY**, suivie du nom de chaque attribut sur laquelle on veut effectuer des regroupements.
- La clause **HAVING** va de pair avec la clause **GROUP BY**, elle permet d'appliquer une restriction sur les groupes créés grâce à la clause **GROUP BY**.

NB : Les fonctions d'agrégat, utilisées seules dans un **SELECT** (sans la clause **GROUP BY**) fonctionnent sur la totalité des tuples sélectionnés comme s'il n'y avait qu'un groupe.

Applications :

✓ Donner le nombre de commandes par client.

```
SELECT NCl, COUNT((NCmd) NbCmd
FROM Commande
GROUP BY NCl ;
```

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

NCl	NbCmd
CL02	1
CL05	1
CL03	2
CL10	1

✓ Donner la quantité totale commandée par produit.

```
SELECT NP, SUM(Qte) Som
FROM Ligne_Cmd
GROUP BY NP;
```

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

NP	Som
P001	950
P002	400
P004	450
P005	70
P006	100
P007	550
P008	90

✓ Donner le nombre de produits par commande.

```
SELECT NCmd, COUNT(NP) NbProd
FROM Ligne_Cmd
GROUP BY NCmd;
```

… Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

NCmd	NbProd
C001	3
C002	2
C003	1
C004	4
C005	2

✓ Donner les commandes dont le nombre de produits dépasse 2.

```
SELECT NCmd, COUNT(NP) NbProd
FROM Ligne_Cmd
GROUP BY NCmd
HAVING COUNT(NP) > 2;
```

… Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

NCmd	NbProd
C001	3
C004	4

✓ Donner le total des montants par commande.

```
SELECT NCmd, SUM(PU*Qte) TotMontant
FROM Produit P, Ligne_Cmd L
WHERE L.NP = P.NP
GROUP BY NCmd;
```

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

NCmd	TotMontant
C001	19100
C002	3325
C003	900
C004	14040
C005	11850

7-2-7- Tri

Les lignes constituant le résultat d'un **SELECT** sont obtenues dans un ordre quelconque. La clause **ORDER BY** précise l'ordre dans lequel la liste des lignes sélectionnées sera donnée.

Syntaxe :
SELECT Attr1, Attr2,…, Attrn
FROM Nom_Table1, Nom_Table2, …
WHERE Liste_Condition
ORDER BY Attr1 [**ASC**| **DESC**], Attr2 [**ASC**| **DESC**], …;

NB : *L'ordre de tri par défaut est croissant (ASC).*

Application :
✓ Donner les noms des clients suivant l'ordre décroissant.

```
SELECT NomCl
FROM Client
ORDER BY NomCl DESC ;
```

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

NomCl
SOUDURE
SBATIM
SANITAIRE
PRODELEC
MELEC
MBATIM
GLOULOU
ELECTRON
BATIMENT
BATFER
BATAM
AMS

✓ Donner le nombre de produits et la quantité totale par commande suivant l'ordre décroissant du nombre de produits et l'ordre croissant de la quantité totale.

```
SELECT NCmd,
COUNT DISTINCT(NP) NbProd,
SUM(Qte) SomQte
FROM Ligne_Cmd
GROUP BY NCmd
ORDER BY NbProd DESC,
         SomQte ASC ;
```

… **Ligne_Cmd**

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

NCmd	NbProd	SomQte
C004	4	410
C001	3	650
C005	2	750
C002	2	750
C003	1	50

7-2-8- Sous requêtes

Effectuer une sous-requête consiste à effectuer une requête à l'intérieur d'une autre, ou en d'autres termes d'utiliser une requête afin d'en réaliser une autre (on entend parfois le terme de requêtes en cascade).
Une sous-requête doit être placée à la suite d'une clause **WHERE** ou **HAVING**, et doit remplacer une constante ou un groupe de constantes qui permettraient en temps normal d'exprimer la qualification.

- lorsque la sous-requête remplace une constante utilisée avec des opérateurs classiques, elle doit obligatoirement renvoyer une seule réponse (une table d'une ligne et une colonne). Par exemple :
  SELECT … **FROM** …
  WHERE … < (**SELECT** … **FROM** …) ;

- lorsque la sous-requête remplace une constante utilisée dans une expression mettant en jeu les opérateurs **IN**, **ALL** ou **ANY**, elle doit obligatoirement renvoyer une seule colonne.
  SELECT … **FROM** …
  WHERE … **IN** (**SELECT** … **FROM** …) ;

- lorsque la sous-requête remplace une constante utilisée dans une expression mettant en jeu l'opérateur **EXISTS**, elle peut renvoyer une table de n colonnes et m lignes.
  SELECT … **FROM** …
  WHERE … **EXISTS** (**SELECT** … **FROM** …) ;

Applications :

✓ Donner les produits dont les prix unitaires dépassent la moyenne des prix.

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

Le résultat de la sous-requête :

AVG(PU)
29

… **Produit**

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

NP	LibP	Coul	Poids	PU	Qtes
P004	Peinture	Blanc	25	33.000	900
P006	Serrure	Jaune	2	47.000	1250
P008	Fer	Noir	50	90.000	800

✓ Donner les commandes qui ont une date inférieure à chacune des commandes du client 'CL03'.

```
SELECT *
FROM Commande
WHERE DateCmd < ALL
        (SELECT DateCmd
         FROM Commande
         WHERE NCl = 'CL03');
```

Le résultat de la sous-requête :

DateCmd
15/01/2004
11/03/2004

… **Commande**

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C004	03/09/2003	CL10

✓ Donner les commandes qui ont une date inférieure à au moins une des commandes du client 'CL03'.

```
SELECT *
FROM Commande
WHERE DateCmd < ANY
        (SELECT DateCmd
         FROM Commande
         WHERE NCl = 'CL03');
```

Le résultat de la sous-requête :

DateCmd
15/01/2004
11/03/2004

… **Commande**

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10

✓ Donner les clients qui ont passé au moins une commande.

SELECT NomCl
FROM Client Cl
WHERE EXIST (**SELECT** * **FROM** Commande C **WHERE** Cl.NCl =C.NCl);

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

➡

NomCl
BATIMENT
AMS
PRODELEC
MELEC

✓ Donner des clients qui n'ont passé aucune commande.

SELECT NomCl
FROM Client Cl
WHERE NOT EXIST (**SELECT** * **FROM** Commande C **WHERE** Cl.NCl =C.NCl);

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

➡

NomCl
BATAM
GLOULOU
ELECTRON
SBATIM
SANITAIRE
SOUDURE
MBATIM
BATFER

7-2-9- Opérateurs ensemblistes

❶ *Union* : L'opérateur **UNION** permet de fusionner deux sélections de tables pour obtenir un ensemble de lignes égal à la réunion des lignes des deux sélections. Les lignes communes n'apparaîtront qu'une seule fois.

Syntaxe :
Requête1
UNION
Requête2 ;

NB :

- Requête1 et Requête2 doivent avoir la même structure.
- Par défaut les doublons sont automatiquement éliminés. Pour conserver les doublons, il est possible d'utiliser une clause **UNION ALL**.

Application :

✓ Donner l'ensemble des clients de Tunis et de Sousse qui ont des commandes.

```
SELECT Cl.NCl, NomCl, AdrCl
FROM Client Cl, Commande C
WHERE Cl.NCl=C.NCl
        AND J.AdrCl='Tunis'
UNION
SELECT Cl.NCl, NomCl, AdrCl
FROM Client Cl, Commande C
WHERE Cl.NCl=C.NCl
        AND J.AdrCl='Sousse';
```

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

R1

NCl	NomCl	AdrCl
CL02	BATIMENT	Tunis
CL05	PRODELEC	Tunis

R2

NCl	NomCl	AdrCl
CL03	AMS	Sousse

NCl	NomCl	AdrCl
CL02	BATIMENT	Tunis
CL05	PRODELEC	Tunis
CL03	AMS	Sousse

❷ *Intersection* : L'opérateur **INTERSECT** permet d'obtenir l'ensemble des lignes communes à deux requêtes.

Syntaxe :
Requête1
INTERSECT
Requête2 ;

NB:
- Requête1 et Requête2 doivent avoir la même structure.
- Il est possible de remplacer l'opérateur **INTERSECT** par des commandes usuelles :

```
SELECT a, b
FROM table1
WHERE EXISTS (    SELECT c, d FROM table2
                  WHERE a=c AND b=d) ;
```

Application :

✓ Donner l'ensemble des produits communs aux commandes C001 et C005.

```
SELECT P.NP, LibP
FROM Produit P, Ligne_Cmd L
WHERE P.NP=L.NP
        AND NCmd='C001'
INTERSECT
SELECT P.NP, LibP
FROM Produit P, Ligne_Cmd L
WHERE P.NP=L.NP
        AND NCmd='C005' ;
```

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

R1

NP	LibP
P001	Robinet
P004	Peinture
P006	Serrure

R2

NP	LibP
P001	Robinet
P002	Prise

NP	LibP
P001	Robinet

❸ *Différence* : L'opérateur **MINUS** permet d'obtenir les lignes de la première requête et qui ne figurent pas dans la deuxième.

Syntaxe :
Requête1
MINUS (ou **EXCEPT)**
Requête2 ;
Avec Requête1 et Requête2 de même structure.

Application :
✓ Donner l'ensemble des produits qui n'ont pas été commandés.

```
SELECT NP, LibP
FROM Produit
MINUS
SELECT P.NP, LibP
FROM Produit P, Ligne_Cmd L
WHERE P.NP=L.NP;
```

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

Tous les produits

Les produits commandés

R1

NP	LibP
P001	Robinet
P002	Prise
P003	Câble
P004	Peinture
P005	Poignée
P006	Serrure
P007	Verrou
P008	Fer

R2

NP	LibP
P001	Robinet
P002	Prise
P004	Peinture
P005	Poignée
P006	Serrure
P007	Verrou
P008	Fer

NP	LibP
P003	Câble

❸ *Produit cartésien* : Le produit cartésien est appliqué sur deux tables n'ayant pas nécessairement la même structure pour obtenir une autre table.
Syntaxe :
SELECT *
FROM table1, table2, … **;**

Application :
✓ Faire le produit cartésien entre la table Client et la table Produit.

SELECT *
FROM Client, Produit;

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000

NCl	NomCl	AdrCl	NP	LibP	Coul	Poids	PU	Qtes
CL01	BATAM	Tunis	P001	Robinet	Gris	5	18.000	1200
CL01	BATAM	Tunis	P002	Prise	Blanc	1.2	1.500	1000
CL02	BATIMENT	Tunis	P001	Robinet	Gris	5	18.000	1200
CL02	BATIMENT	Tunis	P002	Prise	Blanc	1.2	1.500	1000
CL03	AMS	Sousse	P001	Robinet	Gris	5	18.000	1200
CL03	AMS	Sousse	P002	Prise	Blanc	1.2	1.500	1000

7-2-10- Langage algébrique versus Langage SQL

Sélection	Clause **WHERE** dans **SELECT**
Projection	Liste des attributs de **SELECT**
Jointure	Présence de plusieurs tables dans la clause **FROM** (avec la condition de jointure) Ex : **SELECT * FROM** T1, T2 **WHERE** T1.Attri=T2.Attrj ;
Produit cartésien	Présence de plusieurs tables dans la clause **FROM** (sans la condition de jointure) Ex : **SELECT * FROM** T1, T2 ;
Union	**UNION**
Différence	**MINUS** ou **EXCEPT**
Intersection	**INTERSECT**
Division	Il n'existe pas en SQL d'équivalent direct à la division. Cependant, il est toujours possible de trouver une autre solution notamment par l'intermédiaire des opérations de calcul et de regroupement.

7-2-11- Exemple récapitulatif

Donner le nombre de commandes et la somme des quantités du produit P001 commandé par client en ne gardant que les clients ayant un nombre de commandes >= à 2. Le résultat doit être trié selon l'ordre croissant des sommes des quantités.

Étape	Requête
4	**SELECT** NCl, **COUNT**(NCmd) NbCmd, **SUM**(Qte) SomQte
1	**FROM** Commande C, Ligne_Cmd L
2	**WHERE** C.No = L.NoCommande
3	**AND** NP = 'P001'
4	**GROUP BY** NCl
5	**HAVING** NbCmd >= 2
6	**ORDER BY** SomQte ;

1ère étape : Sélection des deux tables Commande et Ligne_Cmd.

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03
C006	01/03/2004	CL11
C007	01/01/2004	CL02

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100
C006	P001	20
C007	P001	40
C007	P002	30

2ème étape : Application de la condition de jointure.

NCmd	DateCmd	NCl	NP	Qte
C001	10/12/2003	CL02	P001	250
C001	10/12/2003	CL02	P004	300
C001	10/12/2003	CL02	P006	100
C002	13/02/2004	CL05	P002	200
C002	13/02/2004	CL05	P007	550
C003	15/01/2004	CL03	P001	50
C004	03/09/2003	CL10	P002	100
C004	03/09/2003	CL10	P004	150
C004	03/09/2003	CL10	P005	70
C004	03/09/2003	CL10	P008	90
C005	11/03/2004	CL03	P001	650
C005	11/03/2004	CL03	P002	100
C006	01/03/2004	CL11	P001	20
C007	01/01/2004	CL02	P001	40
C007	01/01/2004	CL02	P002	30

3ème étape : Sélection des commandes du produit P001 uniquement.

NCmd	DateCmd	NCl	NP	Qte
C001	10/12/2003	CL02	P001	250
C003	15/01/2004	CL03	P001	50
C005	11/03/2004	CL03	P001	650
C006	01/03/2004	CL11	P001	20
C007	01/01/2004	CL02	P001	40

4ème étape : Groupement par NCl en calculant le nombre de commandes et la somme des quantités.

NCl	NbCmd	SomQte
CL02	2	290
CL03	2	700
CL11	1	20

5ème étape : Sélection des groupes ayant un nombre de commandes >= 2.

NCl	NbCmd	SomQte
CL02	2	290
CL03	2	700

6ème étape : Tri selon la somme des quantités.

NCl	NbCmd	SomQte
CL03	2	700
CL02	2	290

7-3- Langage de Manipulation de Données

Le LMD est un ensemble de commandes permettant la consultation et la mise à jour des objets créés. La mise à jour englobe l'insertion de nouvelles données, la modification et la suppression de données existantes.

7-3-1- Insertion de données

Syntaxe :

INSERT INTO Nom_Table [(Attr1, Attr2,…, Attrn)]
VALUES (Val1, Val2,…, Valn);

Permet d'insérer un tuple à la fois.

Ou

INSERT INTO Nom_Table (Attr1, Attr2, …, Attrn)
SELECT…;

Permet d'insérer plusieurs tuples à partir d'une ou plusieurs autres tables.

L'ordre **INSERT** attend la clause **INTO**, suivie du nom de la table, ainsi que du nom de chacun des attributs entre parenthèses (les attributs omis prendront la valeur **NULL** par défaut). Les

données sont affectées aux attributs (colonnes) dans l'ordre dans lequel les attributs ont été déclarés dans la clause **INTO**.

Les valeurs à insérer peuvent être précisées de deux façons :

- avec la clause **VALUES** : une seule ligne est insérée, elle contient comme valeurs, l'ensemble des valeurs passées en paramètre dans la parenthèse qui suit la clause **VALUES**.
  INSERT INTO Nom_table (Attr1, Attr2, ...)
  VALUES (Valeur1, Valeur2, ...) ;
  Lorsque chaque attribut de la table est modifié, l'énumération de l'ensemble des attributs est facultative. Lorsque les valeurs sont des chaînes de caractères, il ne faut pas omettre de les délimiter par des guillemets.

- avec la clause **SELECT** : plusieurs lignes peuvent être insérées, elle contiennent comme valeurs, l'ensemble des valeurs découlant de la sélection.
  INSERT INTO Nom_table (Attr1, Attr2, ...)
  SELECT Attr1, Attr2, ...
  FROM Nom_table2
  WHERE condition ;
  Lorsque l'on remplace un nom d'attribut suivant la clause **SELECT** par une constante, sa valeur est affectée par défaut aux tuples. Il n'est pas possible de sélectionner des tuples dans la table dans laquelle on insère des lignes (en d'autres termes Nom_table doit être différent de Nom_table2).

Applications :

✓ Remplissage de la table **Client**.

INSERT INTO Client **VALUES**('CL01','BATAM','Tunis');
INSERT INTO Client **VALUES**('CL02','BATIMENT','Tunis');
INSERT INTO Client **VALUES**('CL03','AMS','Sousse');
INSERT INTO Client **VALUES**('CL04','GLOULOU','Sousse');
INSERT INTO Client **VALUES**('CL05','PRODELEC','Tunis');
INSERT INTO Client **VALUES**('CL06','ELECTRON','Sousse');
INSERT INTO Client **VALUES**('CL07','SBATIM','Sousse');
INSERT INTO Client **VALUES**('CL08','SANITAIRE','Tunis');
INSERT INTO Client **VALUES**('CL09','SOUDURE','Tunis');
INSERT INTO Client **VALUES**('CL10','MELEC','Monastir');
INSERT INTO Client **VALUES**('CL11','MBATIM','');
INSERT INTO Client **VALUES**('CL12','BATFER','Tunis');

✓ Remplissage de la table **Produit**.

INSERT INTO Produit **VALUES**('P001','Robinet','Gris',5,18,1200);
INSERT INTO Produit **VALUES**('P002','Prise','Blanc',1.2,1.5,1000);
INSERT INTO Produit **VALUES**('P003','Câble','Blanc',2,25,1500);
INSERT INTO Produit **VALUES**('P004','Peinture','Blanc',25,33,900);
INSERT INTO Produit **VALUES**('P005','Poignée','Gris',3,12,1300);
INSERT INTO Produit **VALUES**('P006','Serrure','Jaune',2,47,1250);
INSERT INTO Produit **VALUES**('P007','Verrou','Gris',1.7,5.5,2000);
INSERT INTO Produit **VALUES**('P008','Fer','Noir',50,90,800);

✓ Remplissage de la table **Commande**.

INSERT INTO Commande **VALUES**('C001', '10/12/2003', 'CL02');
INSERT INTO Commande **VALUES**('C002', '13/02/2004', 'CL05');
INSERT INTO Commande **VALUES**('C003', '15/01/2004', 'CL03');
INSERT INTO Commande **VALUES**('C004', '03/09/2003', 'CL10');
INSERT INTO Commande **VALUES**('C005', '11/03/2004', 'CL03');

✓ Remplissage de la table **Ligne_Cmd**.

INSERT INTO Ligne_Cmd **VALUES**('C001','P001',250);
INSERT INTO Ligne_Cmd **VALUES**('C001','P004',300);
INSERT INTO Ligne_Cmd **VALUES**('C001','P006',100);
INSERT INTO Ligne_Cmd **VALUES**('C002','P002',200);
INSERT INTO Ligne_Cmd **VALUES**('C002','P007',550);
INSERT INTO Ligne_Cmd **VALUES**('C003','P001',50);
INSERT INTO Ligne_Cmd **VALUES**('C004','P002',100);
INSERT INTO Ligne_Cmd **VALUES**('C004','P004',150);
INSERT INTO Ligne_Cmd **VALUES**('C004','P005',70);
INSERT INTO Ligne_Cmd **VALUES**('C004','P008',90);
INSERT INTO Ligne_Cmd **VALUES**('C005','P001',650);
INSERT INTO Ligne_Cmd **VALUES**('C005','P002',100);

7-3-2- Modification de données

En utilisant la commande **UPDATE**, on peut modifier les valeurs d'un ou plusieurs champs, dans une ou plusieurs lignes existantes d'une table.

Syntaxe :
UPDATE Nom_Table
SET Attr1 = Expr1, Attr2 = Expr2, …
WHERE Condition;

Ou

UPDATE Nom_Table
SET (Attr1, Attr2, …) = (**SELECT** …)
WHERE Condition;

La modification à effectuer est précisée après la clause **SET**. Il s'agit d'une affectation d'une valeur à un attribut grâce à l'opérateur = suivi d'une expression algébrique, d'une constante ou du résultat provenant d'une clause **SELECT**. La clause **WHERE** permet de préciser les tuples sur lesquels la mise à jour aura lieu.

Application :

✓ Modifier le Poids du produit numéro P002 à 1.

```
UPDATE Produit
SET Poids = 1
WHERE NP = 'P002';
```

✓ Augmenter la quantité en stock des différents produits de 10%.

```
UPDATE Produit
SET Qtes = 1.1 * Qtes;
```

7-3-3- Suppression de données

L'ordre **DELETE** permet de supprimer des lignes d'une table.

Syntaxe :
DELETE FROM Nom_Table
WHERE Condition;

Ici la clause **FROM** précise la table sur laquelle la suppression s'effectue et la clause **WHERE** décrit la qualification, c'est-à-dire l'ensemble des lignes qui seront supprimées.

NB : L'ordre **DELETE** est à utiliser avec précaution car l'opération de suppression est irréversible. Il faudra donc s'assurer dans un premier temps que les lignes sélectionnées sont bien les lignes que l'on désire supprimer!

7-4- Langage de Définition de Données

7-4-1- Types de données

Pour chaque attribut que l'on crée, il faut préciser le type de données que le champ va contenir. Celui-ci peut être un des types suivants :

Type de donnée	**Syntaxe**	**Description**
Type alphanumérique	CHAR(n)	Chaîne de caractères de longueur fixe n (n<16383) Ajout de blancs pour garder la longueur fixe
Type alphanumérique	VARCHAR(n)	Chaîne de caractères de longueur variable de n caractères maximum (n<16383)
Type numérique	SMALLINT	Entier signé de 16 bits (-32768 à 32757)
Type numérique	INTEGER	Entier signé de 32 bits (-2E31 à 2E31-1)
Type numérique	NUMBER(n,[d])	Nombre de n chiffres [optionnellement d après la virgule]
Type numérique	NUMERIC(n, d) DECIMAL(n, d)	Nombres décimaux à nombre fixe de décimales
Type numérique	FLOAT DOUBLE PRECISION	Nombre à virgule flottante (double précision avec au moins 15 chiffres significatifs)
Type numérique	REAL	Nombre à virgule flottante (simple précision avec 7 chiffres significatifs)
Type horaire	DATE	Date sous la forme 16/07/99
Type horaire	TIME	Heure sous la forme 12:54:24.85
Type horaire	TIMESTAMP	Date et Heure

7-4-2- Valeur NULL

Un attribut qui n'est pas renseigné, et donc vide, est dit contenir la valeur '**NULL**'. Cette valeur n'est pas zéro, c'est une absence de valeur.

7-4-3- Contraintes d'intégrité

Une contrainte d'intégrité est une règle qui permet de contrôler la validité et la cohérence des valeurs entrées dans les différentes tables de la base. Elle peut être définie sous deux formes :

- ✓ Dans les commandes de création des tables.
- ✓ Au moment de la modification de la structure de la table.

Il existe des contraintes :

- ✓ *Sur un attribut* : La contrainte porte sur un seul attribut. Elle suit la définition de l'attribut. Ces contraintes sont :
  - **NOT NULL** : Spécifie que pour toute occurrence, l'attribut doit avoir une valeur (la saisie de ce champ est obligatoire).
  - **UNIQUE** : Toutes les valeurs de l'attribut sont distinctes.
  - **PRIMARY KEY** : L'attribut est une clé primaire pour la table et elle peut être remplacée par **UNIQUE** et **NOT NULL**.
  - **REFERENCES** table (attribut) : Il s'agit d'une contrainte d'intégrité fonctionnelle par rapport à une clé ; chaque valeur de l'attribut doit exister dans

la table dont l'attribut est référencée. On utilise cette contrainte pour les clés étrangères.

- **CHECK** : C'est une contrainte associée à une condition qui doit être vérifiée par toutes les valeurs de l'attribut (domaine des valeurs de l'attribut).

✓ *Sur une table* : La contrainte porte sur un ensemble d'attributs d'une même table, une virgule sépare la définition d'une contrainte de table des définitions des attributs. Ces contraintes sont :

- **UNIQUE** (attri, attrj,…) : L'unicité porte sur le n-uplet des valeurs.
- **PRIMARY KEY** (attri, attrj,…) : Clé primaire de la table (clé composée).
- **FOREIGN KEY** (attri, attrj, …) **REFERENCES** table (attrm, attrn, …) [**ON DELETE CASCADE**] : Désigne une clé étrangère sur plusieurs attributs. L'option **ON DELETE CASCADE** indique que la suppression d'une ligne de la table de référence va entraîner automatiquement la suppression des lignes référencées.

Il est possible de donner un nom à une contrainte grâce au mot clé **CONSTRAINT** suivi du nom que l'on donne à la contrainte, de telle manière à ce que le nom donné s'affiche en cas de non respect de l'intégrité, c'est-à-dire lorsque la clause que l'on a spécifiée n'est pas validée.

7-4-4- Création d'une table

La création de tables se fait à l'aide du couple de mots-clés **CREATE TABLE**.

Syntaxe :
CREATE TABLE nom_table
(Attribut1 Type [Contrainte d'attribut],
Attribut2 Type [Contrainte d'attribut],
…
Attributn Type [Contrainte d'attribut],
[Contrainte de relation], …);

Application :

✓ Création de la table **Client** en donnant à sa clé **NCl** les propriétés **NOT NULL** et **UNIQUE**.

```
CREATE TABLE Client
(NCl VARCHAR2(4) NOT NULL UNIQUE,
NomCl VARCHAR2(15),
AdrCl VARCHAR2(10));
```

✓ Création de la table **Produit** en définissant la contrainte **pk_NP** de clé primaire équivalente à l'attribution des propriétés **NOT NULL** et **UNIQUE**. Ne pas définir l'attribut Qtes ceci sera défini ultérieurement par l'instruction **ALTER TABLE**.

```
CREATE TABLE Produit
(NP VARCHAR2(4) CONSTRAINT pk_NP PRIMARY KEY,
 LibP VARCHAR2(15),
 Coul VARCHAR2(10),
 Poids NUMBER(6,3),
 PU NUMBER(6,3) );
```

✓ Création de la table **Commande** sans définir de contrainte ni de propriétés sur la clé, ceci sera défini ultérieurement par l'instruction **ALTER TABLE**.

```
CREATE TABLE Commande
(NCmd VARCHAR2(4),
 DateCmd DATE,
 NCl VARCHAR2(4));
```

✓ Création de la table **Ligne_Cmd** en spécifiant la contrainte de clé primaire et l'une des contraintes de clé étrangère.

```
CREATE TABLE FPJ
(NCmd VARCHAR2(4),
 NP VARCHAR2(4),
 Qte NUMBER(5),
 CONSTRAINT pk_LCMD PRIMARY KEY (NCmd, NP),
 CONSTRAINT fk_NP FOREIGN KEY (NP) REFERENCES Produit(NP));
```

7-4-5- Insertion de lignes à la création

Il est possible de créer une table en insérant directement des lignes lors de la création. Les lignes à insérer peuvent être alors récupérées d'une table existante grâce au prédicat **AS SELECT**.

Syntaxe :

```
CREATE TABLE nom_table
 (Attr1 Type [Définition de Contrainte],
  Attr2 Type [Définition de Contrainte],
 ...)
AS SELECT Attr1, Attr2, ...
FROM nom_table2
WHERE Condition;
```

7-4-6- Création d'index

La création d'un index permet d'accélérer les recherches d'informations dans la base. La ligne est retrouvée instantanément si la recherche peut utiliser un index, sinon la recherche se fait séquentiellement. Une autre utilité de la création d'index est de garantir l'unicité de la clé en utilisant l'option **UNIQUE**.

Syntaxe :

```
CREATE [UNIQUE] INDEX nom_index
ON nom_table (Attr1[ASC/DESC], Attr2[ASC/DESC], …);
```

- L'option **UNIQUE** permet de définir la présence ou non de doublons pour les valeurs de l'attribut.
- Les options **ASC/DESC** permettent de définir un ordre de classement des valeurs présentes dans l'attribut.

7-4-7- Modification de la structure d'une table

La clause **ALTER** permet l'ajout de nouveaux attributs, la modification des attributs ou la suppression des attributs d'une table.

❶ *Ajout d'un attribut* : Permet d'ajouter un attribut à la structure initiale de la table.

Syntaxe :

```
ALTER TABLE Nom_Table
ADD Attribut Type;
```

Application :

✓ Ajout de l'attribut **Qtes** à la table **Produit**.

```
ALTER TABLE Produit
ADD Qtes NUMBER(5);
```

❷ *Modification d'un attribut* **:** Associée avec la clause **MODIFY**, la clause **ALTER** permet la modification du type de données d'un attribut. On ne peut qu'agrandir la taille d'un attribut.

Syntaxe :

ALTER TABLE Nom_Table
MODIFY Attribut Nouveau_Type;

Application : Modification de la taille de l'attribut LibP à **VARCHAR**(20).
ALTER TABLE Produit
MODIFY LibP **VARCHAR**(20);

❸ *Suppression d'un attribut* : Permet de supprimer un attribut d'une table.

Syntaxe :
ALTER TABLE Nom_Table
DROP COLUMN Attribut ;

Il faut noter que la suppression d'attributs (colonnes) n'est possible que dans le cas où :

- L'attribut ne fait pas partie d'une vue,
- L'attribut ne fait pas partie d'un index,
- L'attribut n'est pas l'objet d'une contrainte d'intégrité.

❹ *Ajout de contrainte* : Permet d'ajouter une contrainte au niveau d'une table.

Syntaxe :
ALTER TABLE Nom_Table
ADD CONSTRAINT Nom_Contrainte Définition_Contrainte;

Applications :

✓ Ajout de la contrainte de **PRIMARY KEY** sur l'attribut **NCmd** de la table **Commande**.
ALTER TABLE Commande
ADD CONSTRAINT pk_NCMD **PRIMARY KEY** (NCmd);

✓ Ajout des contraintes de clés étrangères sur la table **Commande**.
ALTER TABLE Commande
ADD CONSTRAINT fk_NCL **FOREIGN KEY** (NCl) **REFERENCES** Client(NCl);

✓ Ajout des contraintes de clés étrangères sur la table **Ligne_Cmd**.
ALTER TABLE Ligne_Cmd
ADD CONSTRAINT fk_NCMD **FOREIGN KEY** (NCmd) **REFERENCES** Commande(NCmd);

✓ Ajout de la contrainte **CHECK** sur l'attribut **Qte** (**Qte** > 0) de la table **Ligne_Cmd**.
ALTER TABLE Ligne_Cmd
ADD CONSTRAINT ck_QTE **CHECK** (Qte > 0);

❺ *Suppression de contrainte* : Permet de supprimer une contrainte.

Syntaxe :
ALTER TABLE Nom_Table
DROP CONSTRAINT Nom_Contrainte;

❻ *Désactivation d'une contrainte* : Permet de désactiver une contrainte, elle est par défaut active (au moment de sa création).

Syntaxe :
ALTER TABLE Nom_Table
DISABLE CONSTRAINT Nom_Contrainte;

❼ *Activation d'une contrainte* : Permet d'activer une contrainte désactivée.

Syntaxe :
ALTER TABLE Nom_Table
ENABLE CONSTRAINT Nom_Contrainte;

7-4-8- Suppression d'une table / un index

La clause **DROP** permet d'éliminer des vues, des index et même des tables. Cette clause est toutefois à utiliser avec précaution dans la mesure où elle est irréversible.

Syntaxe :

- Pour supprimer un index : **DROP INDEX** Nom_Index [**ON** Nom_Table];
- Pour supprimer une table : **DROP TABLE** Nom_table ;

NB **:**

- Le nom de la table est obligatoire si on veut supprimer un index d'une table d'un autre utilisateur.
- Un index est automatiquement supprimé dès qu'on supprime la table à laquelle il appartient.

7-5- Langage de Contrôle de Données

Plusieurs personnes peuvent travailler simultanément sur une base de données, toutefois ces personnes n'ont pas forcément les mêmes besoins : certaines peuvent par exemple nécessiter de modifier des données dans la table, tandis que les autres ne l'utiliseront que pour la consulter. Ainsi, il est possible de définir des permissions pour chaque personne en leur octroyant un mot de passe. Cette tâche incombe à l'administrateur de la base de données (en anglais *DBA*, *DataBase Administrator*). Il doit dans un premier temps définir les besoins de chacun, puis les appliquer à la base de donnée sous forme de permissions. Le langage SQL permet d'effectuer ces opérations grâce à deux clauses :

- **GRANT** permet d'accorder des droits à un (parfois plusieurs sur certains SGBD) utilisateur
- **REVOKE** permet de retirer des droits à un (ou plusieurs sur certains SGBD) utilisateur

Les permissions (appelées aussi droits ou privilèges) peuvent être définies pour chaque (un grand nombre) clause. D'autre part il est aussi possible de définir des rôles c'est-à-dire de permettre à d'autres utilisateurs d'accorder des permissions.

❶ *GRANT* : Permet au propriétaire d'une table ou vue de donner à d'autres utilisateurs des droits d'accès à celles ci.

Syntaxe :
GRANT Liste_Privilège **ON** Table/ Vue **TO** Utilisateur [**WITH GRANT OPTION**];

Les privilèges sont :

SELECT	Droit de lecture
INSERT	Droit d'insertion de lignes
UPDATE	Droit de modification de lignes
UPDATE (Attr1**,** Attr2**,** …**)**	Droit de modification de lignes limité à certains attributs
DELETE	Droit de suppression de lignes
ALTER	Droit de modification de la structure de la table
INDEX	Droit de création d'index
ALL	Tous les droits

Application :
GRANT SELECT ON Produit **TO** User1;

❷ *REVOKE* : Un utilisateur ayant accordé un privilège peut l'annuler à l'aide de la commande **REVOKE**.

Syntaxe :
REVOKE Liste_Privilège **ON** Table/Vue **FROM** Utilisateur;
Application :
REVOKE SELECT ON Produit **FROM** User1;

NB **:** Si on enlève un privilège à un utilisateur, ce même privilège est automatiquement retiré à tout autre utilisateur à qui il l'aurait accordé.

Exercices

Exercice 1 : Questions à choix multiples

1. Quelle clause SQL permet d'effectuer un tri sur les données sélectionnées ?

A	WHERE
B	ORDER BY
C	GROUP BY

2. A quoi sert une clé primaire ?

A	Rendre un champ non modifiable
B	Identifier chaque enregistrement de manière unique
C	Verrouiller la base de données

3. Dans une requête SQL, quel est le sens de l'expression **WHERE Nom LIKE 'Be_'** ?

A	Tous les noms qui commencent par Be
B	Tous les noms qui ne commencent pas par Be
C	Tous les noms de 3 caractères qui commencent par Be

4. La fonction **COUNT** permet-elle de réaliser le total des valeurs d'un champ numérique ?

A	OUI
B	NON

5. Dans une requête SQL **CHECK**

A	permet de rechercher un attribut ayant une valeur donnée.
B	est une contrainte associée à une condition qui doit être vérifiée par toutes les valeurs de l'attribut.
C	permet de rechercher un attribut clé ayant une valeur donnée.

6. Pami les requêtes SQL suivantes, quelles sont celles qui donnent le nombre de tuples dans la relation R (AC, A2, A3) ?

A	SELECT COUNT (*) FROM R ;
B	SELECT SUM (*) FROM R ;
C	SELECT COUNT (AC) FROM R ;

7. Pami les requêtes SQL suivantes, quelles sont celles qui donnent la somme des valeurs de l'attribut A2 de la relation R (AC, A2, A3) ?

A	SELECT AC , SOMME (*) FROM R GROUP BY AC ;
B	SELECT SUM (*) FROM R ;
C	SELECT SUM (A2) FROM R ;

8. La requête suivante appliquée sur la relation R (AC, A2, A3) :

   SELECT COUNT (*) FROM R GROUB BY AC;

A	donne le nombre de tuples
B	donne la somme des tuples
C	ne donne rien

9. Si on applique la requête suivante sur la relation Ligne_Cmd (NCmd, NP, Qte) avec les données ci-dessous quel sera le résultat :

   SELECT NCmd, COUNT (*) FROM Ligne_Cmd WHERE Qte > 100 GROUB BY NCmd Having SUM (Qte) > 600;

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

A

NCmd	COUNT
C001	3
C002	2
C005	2

B

NCmd	COUNT
C002	2
C005	1

C

NCmd	COUNT
C001	3
C002	2
C004	2
C005	2

10. Si on applique la requête suivante sur la relation avec les données de la question précédente, quel sera le résultat :

    SELECT COUNT (*) FROM Ligne_Cmd WHERE Qte > 100;

A

COUNT
12

B

COUNT
9

C

COUNT
6

Exercice 2 :

1. Quel est le résultat de la requête suivante :

   SELECT NoDep, SUM (Salaire) FROM Employes WHERE (Annee > 1980) GROUP BY NoDep HAVING SUM (Salaire) > 1000.000 ORDER BY 2 ;

2. Détecter l'erreur qui existe dans la requête suivante :

   ALTER TABLE Employes CHECK (Annee > 1950);

Exercice 3 :

La table CDA repréesente un Calendrier de Dates d'Anniversaire. La table MARIAGES stocke les mariages en vigueur; un homme ou une femme ne peut avoir plusieurs conjoints en même temps.

On maintient l'anniversaire et l'adresse de tous les mariés. On apprend que Tante Odette est née le 27 juin 1936. Elle est mariée avec Oncle Urbain depuis le 1 mai 1950.

CDA

Nom	Anniversaire	Année	Adresse	Ville
Tante Odette	27 juin	1936	17 Rue R. Barre	Mons
Oncle Urbain	27 juin	1927	17 Rue R. Barre	Mons
Mon chat	17 mars	2001	Chez moi	Enghien
Jean Bidon	23 mai	1963	36 Rue d'Egmont	Bruxelles
Anne Lalo	15 mars	1965	35 Rue d'Egmont	Mons
Jean Crevette	12 janvier	1965	23 Place du Parc	Mons

MARIAGES

Femme	Mari	Jour	Année
Tante Odette	Oncle Urbain	1 mai	1950
Anne Lalo	Jean Crevette	14 juillet	1978

1. Soit la requête Q1 suivante :

```
SELECT Femme
FROM   MARIAGES M
WHERE Mari IN (SELECT Nom FROM CDA WEHRE Adresse = M.Adresse);
```

   a. Que donne la requête Q1 ?
   b. Donner le résulat de cette requête.

2. Soit la requête Q2 suivante :

```
SELECT Nom
FROM   CDA
WHERE Nom Not IN (SELECT Femme FROM MARIAGES UNION
                  SELECT Mari     FROM MARIAGES);
```

   a. Que donne la requête Q2 ?
   b. Donner le résulat de cette requête.

Exercice 4 :

Soit la base de données décrite par les trois relations suivantes :

Film (CodeFilm, Titre, Année, PaysProd, Réalisateur, Genre)
Acteur (CodeActeur, Nom, Pays)
Jouer (#CodeActeur, #CodeFilm, Salaire)

Les attributs ont la signification suivante :

CodeFilm	: Numéro d'un film	**Genre**	: Genre du film (aventure, horreur, …)
Titre	: Titre du film	**CodeActeur**	: Numéro d'un acteur
Année	: Année de sortie du film	**Nom**	: Nom de l'acteur
PaysProd	: Pays du producteur du film	**Pays**	: Nationalité de l'acteur
Réalisateur	: Nom du réalisateur du film	**Salaire**	: Salaire perçu par un acteur dans un film

Formuler chacune des requêtes suivantes en SQL :

1) LDD

a) En respectant les contraintes d'intégrités référentielles et les contraintes suivantes, créer la table **Jouer** :

- Le CodeActeur est un entier compris entre 1000 et 9000.

- Le CodeFilm est un nombre sur 3 chiffres qui doit être obligatoirement défini.

b) Dans la table Acteur, ajouter l'attribut Sexe : un caractère qui désigne le sexe de l'acteur ('F' pour Féminin ou 'M' pour Masculin).

2) LMD (les attributs entre crochets représentent les attributs résultats)

a) Retrouver les films [Titre] d'horreur sortis après 1950.

b) Retrouver les noms des acteurs anglo-saxons (anglais et américains).

c) Quelles sont les actrices (sexe féminin) des films d'horreur ?

d) Donner les pays d'origine des acteurs qui ont joué dans des films policiers ou des films de guerre.

e) Donner la liste des films [Titre] joués par les acteurs français.

f) Donner la liste des acteurs [Nom] réalisateurs.

g) Quels sont les acteurs [Nom] qui jouent dans des films joués par l'acteur X ?

h) Donner le salaire maximum des acteurs anglais.

i) Donner le nombre de films tournés par acteur [CodeActeur, Nom, Nombre de films].

j) Retrouver pour chaque film les noms des acteurs ayant joué dans ce film [Titre, Nom].

k) Retrouver les noms des acteurs et des réalisateurs sous la direction desquels ils ont joué [Réalisateur, Nom].

l) Retrouver les noms des acteurs qui ne sont pas français.

m) Quels sont les acteurs [Nom] qui ont joué dans des films produits par des pays autres que les leurs ?

n) Donner le nombre de films tournés par les acteurs américains.

o) Donner le salaire total des acteurs français.

p) Donner les acteurs qui ont participé à des films français et anglais.

Exercice 5 :

Soit la base de données décrite par les relations suivantes :

RESTAURANT	(REST, NOMR, ADR, TEL)
CONSOMMATEUR	(CONS, NOMC)
PLAT	(PLAT, NOMP, PRIX)
FREQUENTE	(#CONS, #REST)
SERVICE	(#REST, #PLAT)
PREFERE	(#CONS, #PLAT)

Les relations RESTAURANT, CONSOMMATEUR et PLAT fournissent respectivement les données relatives à un restaurant, un consommateur et un plat du système étudié.

La relation FREQUENTE indique les restaurants que chaque consommateur visite. L'attribut CONS désigne le numéro d'un consommateur. L'attribut REST désigne le numéro d'un restaurant.

La relation SERVICE fournit les plats servis par chaque restaurant. L'attribut PLAT désigne le numéro d'un plat.

La relation PREFERE donne les plats préférés par un consommateur.

Les clés primaires de chaque relation sont soulignées dans le schéma relationnel de la base fournie ci-dessus.

Formuler chacune des requêtes suivantes en SQL :

1) LDD

a) En respectant les contraintes d'intégrités référentielles et les contraintes suivantes, créer la table FREQUENTE :

- Le numéro d'un consommateur est un entier compris entre 1000 et 9000.
- Le numéro d'un restaurant est un nombre sur deux chiffres qui doit être obligatoirement défini.
- La clé primaire de cette relation est composée des attributs CONS et REST.

b) Ajouter l'attribut NUMJ un entier compris entre 1 et 7 à la relation SERVICE. Le NUMJ décrit le numéro du jour de la semaine où le plat peut être servi par le restaurateur.

2) LMD

a) Donner la liste des numéros des consommateurs qui fréquentent le restaurant 'LES DUNES'.

b) Donner la liste des numéros des consommateurs qui fréquentent plus de 10 restaurants.

c) Donner la liste des numéros des restaurants pour lesquels le nombre de jours de service par semaine est supérieur à quatre.

d) Donner les noms des restaurants (NOMR) qui servent des plats que le consommateur 'HABIB' préfère. Proposer deux solutions : une en utilisant les jointures et une autre en utilisant les requêtes imbriquées.

e) Donner la liste des numéros des consommateurs qui préfèrent au moins un plat que le consommateur numéro 1001 préfère. En utilisant l'opérateur ensembliste IN.

f)

a. Donner la liste des numéros des restaurants qui servent le plat 'PAELLA' tous les jours (sachant que le nombre de jours ouvrables est 6).
b. Donner la liste des numéros des restaurants qui servent le plat 'STEAK FRITES' pendant un jour quelconque.
c. En déduire, la liste des numéros des restaurants qui servent le plat 'PAELLA' tous les jours (sachant que le nombre de jours ouvrables est 6) et des 'STEAK FRITES' pendant un jour quelconque.

g) Donner la liste des numéros des restaurants chez lesquels 'HABIB' peut trouver n'importe quel plat préféré à n'importe quel jour de la semaine.

Exercice 6 :

Soit un ensemble de personnes identifiées par un numéro et caractérisées par un nom et un ensemble de banques identifiées par un numéro.

Les banques sont localisées dans une ville. Une personne peut ouvrir un ou plusieurs comptes dans une banque. On connaît le solde de chaque compte. Chaque banque affecte à ses comptes un numéro unique.

La base de données relationnelle résultat est la suivante :

BANQUE (**NoB**, Ville)

PERSONNE (**NoP**, Nom)

COMPTE (**NoC, #NoB**, Solde, #NoP)

On demande d'exprimer en SQL les requêtes suivantes:

1. La liste des numéros de banques de SousseLa liste des comptes, dont le solde est débiteur de plus de 1000 dinars, de la banque 123
3. Le nombre de banques à SousseLa liste des clients des banques de SousseLa liste des banques dans la même ville que la banque 123

Exercice 7 :

Soit la base de données relationnelle suivante :

DEPOT (**NumD**, Demande)

FABRIQUE (**NumF**, Capacité, PrixV, QtéRemise)

TRANSPORT (#**NumD**, #**NumF**, Distance, Quantité)

- NumD : Numéro du dépôt
- Demande : demande d'un dépôt
- NumF : Numéro de la fabrique
- Capacité : capacité de fabrication de la fabrique
- PrixV : Prix de vente adopté par la fabrique
- QtéRemise : Quantité minimale exigé par la fabrique pour faire une remise
- Distance : Distance parcourue pour transporter les produits d'une fabrique à un dépôt
- Quantité : Quantité des produits transportés d'une fabrique à un dépôt

On demande d'exprimer en SQL les requêtes suivantes :

1. Sélectionner la fabrique qui a la plus grande capacité.
2. Sélectionner la fabrique et le dépôt qui sont les plus proches.
3. Sélectionner les dépôts qui ont été livrés par la fabrique 7.
4. Sélectionner les fabriques qui ont effectué plus de 3 transports à l'entrepôt 5
5. Sélectionner les dépôts dont la demande est supérieure à la somme des quantités que reçoit ce dépôt.
6. Sélectionner les dépôts dont la demande est supérieure à la demande moyenne de tous les dépôts.