Institut Supérieur des Etudes Technologiques de Sfax

LE LANGAGE SQL

(Structured Query Language)

Préparé par :

Souhaïl SMAOUI

Année Universitaire 2016-2017

Objectif du cours : Maîtriser le langage SQL.

Pré requis : Algorithmique et base de données

Objectifs	Conditions de réalisation	Méthodologies
Maitriser le langage de définition des	A partir du support de cours, des travaux dirigés et des travaux pratiques, l'étudiant devra créer, modifié et supprimés des tables.	Tableau et data show. Salle de TP.
Maitriser le langage de manipulation des.	A partir du support de cours, des travaux dirigés et des travaux pratiques, l'étudiant devra ajouter, modifier, supprimer et interroger des données.	Tableau et data show. Salle de TP.
Maitriser le langage de contrôle des données	A partir du support de cours, des travaux dirigés et des travaux pratiques, l'étudiant devra créer de nouveaux utilisateurs, ajouter et supprimer des droits d'accès.	Tableau et data show. Salle de TP.

SOMMAIRE

1. INTRODUCTION

Le langage **SQL** (Structured Query Language) est un langage d'accès normalisé aux bases de données, supporté par la majorité des produits commerciaux.

La première version de SQL a été développée à IBM en 1970 par Donald D. Chamberlin et Raymond F. Boyce. Cette version, d'origine était appelé SEQUEL, a été conçue pour manipuler et éditer des données emmagasinées dans la base de données relationnelles IBM.

En 1979, Relational Software, (actuellement Oracle Corporation) présenta la première version commercialement disponible de SQL.

SQL a été adopté comme recommandation par l'Institut de normalisation américaine (ANSI) en 1986, puis comme norme internationale par l'ISO en 1987 sous le nom de ISO/CEI 9075 - Technologies de l'information - Langages de base de données - SQL.

La norme internationale SQL est passée par un certain nombre de révisions :

Année	Nom	Appellation	Commentaires
1986	ISO/CEI 9075:1986	SQL-86 ou SQL-87	Édité par l'ANSI puis adopté par l'ISO en 1987.
1989	ISO/CEI 9075:1989	SQL-89 ou SQL-1	Révision mineure.
1992	ISO/CEI 9075:1992	SQL-92 alias SQL2	Révision majeure.
1999	ISO/CEI 9075:1999	SQL-99 alias SQL3	Expressions rationnelles, requêtes récursives, déclencheurs, types non-scalaires et quelques fonctions orientées objet (les deux derniers points sont quelque peu controversés et pas encore largement implémentés).
2003	ISO/CEI 9075:2003	SQL:2003	Introduction de fonctions pour la manipulation XML, « window functions », ordres standardisés et colonnes avec valeurs auto-produites (y compris colonnes d'identité).
2008	ISO/CEI 9075:2008	SQL:2008	Ajout de quelques fonctions de fenêtrage (ntile, lead, lag, first value, last value, nth value),

			limitation du nombre de ligne (OFFSET / FETCH), amélioration mineure sur les types distincts, curseurs et mécanismes d'auto incréments.
2011	ISO/IEC 9075:2011	SQL:2011	

SQL est un langage interactif non procédural, peut être décomposé en trois sous langages :

- Le **langage de définition de données (LDD)** qui permet de définir les structures et les contraintes des bases de données,
- Le **langage de manipulation de données (LMD)** qui permet d'interroger une base de données indépendamment de l'organisation physique des données,
- Le **langage de contrôle de données (LCD)** qui permet de contrôler la sécurité et les accès aux données.

NB :

- Tout au long de ce cours on utilisera la base de données suivante :

fournisseur(cod_frs, nom_frs, adresse_frs, tele_frs)

produit (cod_prd, lib_prd, quantite_stock, prix_unitaire, cod_tva#)

tva (cod_tva, taux_tva)

fournir (cod_frs#, cod_prd#)

commande (num_cde, date_cde, #cod_frs)

ligne_commande (num_cde, cod_prd, quantité)

Dictionnaire de données :

CODE	DESCRIPTION	TYPE	TAILLE
cod_frs	Code fournisseur	Chaîne de caractères	10

nom_frs	Nom fournisseur	Chaîne de caractères	35
Adresse_frs	Adresse fournisseur	Chaîne de caractères	100
tele_frs	Téléphone fournisseur	Chaîne de caractères	13
cod_prd	Code fournisseur	Chaîne de caractères	10
lib_prd	Libéllé produit	Chaîne de caractères	25
quantite_stock	Quantité stock	Décimal	(10,3)
cod_tva	Code TVA	Entier	1
taux_tva	Taux TVA	Entier	2
prix_unitaire	Prix unitaire	Décimal	(10,3)
Num_cde	Numéro commande	Entier	4
date_cde	Date commande	date	-
quantité	Quantité commandée	Décimal	(10,3)

- Tout ce qui est écrit entre [ ] est facultatif.

2. LE LANGAGE DE DEFINITION DE DONNEES (LDD)

Le langage de définition de données se base sur trois types de commandes :

- Create : pour la création d'objets.
- Drop : pour la suppression d'objets.
- Alter : pour la modification d'objets.

2.1. CREATION DE TABLES

Avec ORACLE ont peut créer une table avec deux formes :

- création simple,

- création avec insertion.

- **Création Simple**

Syntaxe :

CREATE TABLE nom_table

(

Col_1 type [(taille)] [DEFAULT valeur_par_defaut] [NULL/NOT NULL] [contraintes de colonne],

Col_2 type [(taille)] [DEFAULT valeur_par_defaut] [NULL/NOT NULL] [contraintes de colonne],

Col_i type [(taille)] [DEFAULT valeur_par_defaut] [NULL/NOT NULL] [contraintes de colonne],

[Contraintes de table]

);

L'option NOT NULL assure qu'ORACLE interdit que cette colonne contienne la valeur NULL.

L'option DEFAULT permet d'affecter une valeur par défaut à cette colonne si sa valeur n'a pas été précisée lors d'une insertion.

- **Création avec Insertion de données**

On peut avec une seule commande créer et remplir une table avec des données d'une requête d'ordre select. Si les colonnes ne sont pas précisées, la nouvelle table va hériter celles provenant de la requête.

Syntaxe :

CREATE TABLE nom_table

[(nom_col1…,

nom_col2….,

...)]

AS SELECT... ;

2.2. LES TYPES DE DONNEES

Les types de données peuvent être :

NUMBER [(longueur,[précision]) : permet de déclarer des colonnes de type entier et réel dont la longueur ne dépasse pas 38 chiffres dans les deux cas.

CHAR (longueur) : permet de déclarer des colonnes de type caractère dont la longueur doit être inférieure à 255.

VARCHAR2 (longueur) : permet de déclarer des colonnes de type caractère de longueur variable. Longueur doit être inférieure à 2000.

DATE : permet de déclarer des colonnes de type date.

LONG : Ce type de données permet des stocker des chaînes de caractères de longueur variable et inférieure à 2^31 -1. Les colonnes de ce type sont soumises à certaines restrictions :

Une table ne peut contenir qu'une seule colonne de ce type ;

Les colonnes de ce type ne peuvent pas apparaître dans des contraintes d'intégrité;

Les colonnes de ce type ne peuvent pas être indexées ;

Les colonnes de ce type ne peuvent pas apparaître dans des clauses : WHERE, GROUP BY, ORDER BY ou CONNECT BY ainsi que dans un DISTINCT.

2.3. DEFINITION DES CONTRAINTES

Les contraintes définies sur une base de données doivent être vérifiées par le système à chaque manipulation des données. Elles peuvent être de trois types :

2.3.1 Les contraintes de domaine

Ces contraintes sont appliquées sur les colonnes de la table. Elles sont décrites directement après son type et sa longueur.

✓ [NOT] NULL indique si la colonne peut avoir ou non la valeur nulle, par défaut elle peut avoir la valeur nulle.

Exemple :

create table tva

(cod_tva number(1) **not null**,

taux_tva number(2));

✓ DEFAULT Permet de donner une valeur par défaut à l'attribut.

Exemple :

create table tva

(cod_tva number(1) not null,

taux_tva number(2) **default 18)**;

✓ CHECK permet d'affecter une contrainte sur les valeurs d'une colonne.

Exemple :

create table produit

(cod_prd char(10),

lib_prd char(10),

quantite_stock number(10,3) **check** (quantite_stock>0),

cod_tva number(1));

✓ UNIQUE indique que la valeur d'une colonne doit être unique, c'est-à-dire ne peut pas être dupliquée.

Exemple :

create table produit

(cod_prd char(10),

lib_prd char(10) **unique**,

quantite_stock number(10,3),

cod_tva number(1));

2 .3.2. Les contraintes d'intégrité d'entité

Chaque table doit avoir une clé primaire qui doit être NOT NULL.

Exemple :

create table produit

(cod_prd char(10) [**constraint pk_produit**] **primary key**,

lib_prd char(10),

quantite_stock number(10,3),

cod_tva number(1));

Ou bien :

create table produit

(cod_prd char(10),

lib_prd char(10),

quantite_stock   number(10,3),

cod_tva number(1),);

constraint pk_produit primary key(cod_prd)) ;

2.3.3. Les contraintes d'intégrité référentielle

Ce type de contrainte indique un lien entre une clé étrangère et la clé primaire avec laquelle elle est liée.

Exemple :

create table produit

(cod_prd char(10) [constraint pk_produit] primary key,

lib_prd char(10),

quantite_stock   number(10,3),

cod_tva number(1) **[constraint fk_prd_tva] references tva(cod_tva**)) ;

Ou bien :

create table produit

(cod_prd char(10) [constraint pk_produit] primary key,

lib_prd char(10),

quantite_stock   number(10,3),

cod_tva number(1),

constraint fk_prd_tva foreign key(cod_tva) references tva(cod_tva)) ;

Remarque:
On distingue deux syntaxes de définition d'une contrainte :

- Syntaxe de définition d'une contrainte de colonne.
- Syntaxe de définition d'une contrainte de table.

Syntaxe de définition d'une contrainte de colonne :

```
[CONSTRAINT nom_contrainte] {  [not null],
                               [unique | primary key],
                               [references [schema.] table (colonne) [on delete
                               cascade]/[on update cascade]
                               [check (condition)] }
```

Syntaxe de définition d'une contrainte de table :

```
[CONSTRAINT nom_contrainte]   {  [not null],
                                  [unique | primary key] (colonne [,colonne]...)
                                  [foreign key (colonne [,colonne]...references [schema.]
                                   table  (colonne [,colonne]...)[on delete cascade] /[on
                                   update cascade]
                                  [check (condition)]
                                  }
```

L'option update cascade permet la mise à jour automatique des valeurs correspondantes lors d'une modification.

L'option delete cascade permet la suppression d'une ligne même si elle est référencée.

2.4. MODIFICATION DE LA STRUCTURE D'UNE TABLE

Une structure d'une table peut être modifiée grâce à la commande ALTER TABLE.

Cette commande permet de :

- ajouter, supprimer ou redéfinir une colonne,
- ajouter, supprimer, activer ou désactiver une contrainte d'intégrité,

Syntaxe :

```
ALTER TABLE nom_table
[ADD ( col1 type [(taille)][NULL/NOT NULL],….
        coln type [(taille)][NULL/NOT NULL],
        constraint nom_contrainte1 description,…
        constraint nom_contrainte description) )]
[MODIFY ( col1 type [(taille)][NULL/NOT NULL],….
            coln type [(taille)][NULL/NOT NULL]) ]
[drop  column nom_colonne],
[drop  Constraint nom contrainte]
[drop  primary key [cascade]]
[enable constraint nom_contrainte]
[disable  constraint nom_contrainte] ;
```

Exemple d'ajout de colonnes et de contrainte :

```
alter table produit add
(coleur char(10),
masse number(5,2),
constraint coul check(couleur in ('blanc,'rouge') ) ;
```

Exemple de modification de colonnes :

```
alter table produit modify
( lib_prd char(25),
coleur char(5)) ;
```

Exemples d'activation et désactivation d'une contrainte :

```
alter table produit enable constraint pk_produit ;
alter table produit disable constraint pk_produit ;
```

Exemples de suppression d'une colonne et d'une contrainte :

alter table produit drop constraint pk_produit ;

alter table produit drop column masse ;

Exemples de suppression d'une clé primaire:

alter table produit drop primary key [cascade] ;

L'option cascade permet de supprimer la clé même si elle est référencée, la suppression de ses liens est implicite.

Remarque :

La commande suivante permet d'afficher les contraintes d'une table :

SELECT * FROM user_constraints [WHERE table_name ='NOM_TABLE'];

Le nom de la table doit être écrit en majuscule.

2.5. SUPPRESSION D'UNE TABLE

Syntaxe :

DROP TABLE nom_table [cascade constraints];

Exemple:

drop table produit cascade constraints ;

L'option cascade constraints permet de supprimer toutes les contraintes appliquées sur la table.

Remarque :

On peut renommer une table avec la commande rename :

Syntaxe :

Rename ancien_nom to nouveau_nom.

Exemple:

rename produit to produits;

3. LE LANGAGE DE MANIPULATION DES DONNEES (LMD)

Le langage de manipulation de données permet d'appliquer les trois opérations habituelles de mise à jour des données :

- l'insertion,
- la modification,
- la consultation.

3.1. INSERTION DE DONNEES

3.3.1. Insertion par saisie

Syntaxe :

INSERT INTO nom_table VALUES [(col1,col2,...,coln)] VALUES (val1,val2,... ,valn);

Les colonnes ne figurant pas dans la liste des colonnes auront la valeur NULL.

S'il s'agit d'une insertion globale de toutes les colonnes, l'indication des différentes colonnes sera inutile.

Exemple :

Insert into produit values ('100','stylo rouge', 560,1);

Insert into produit(cod_prd, lib_prd) values ('101','stylo vert') ;

3.1.2. Insertion par copie

Syntaxe :

INSERT INTO nom_table1 VALUES [(col1,col2,...,coln)]

Select [(col1,col2,...,coln)] from nom_table2

Where condition ;

Exemple :

Soit la table suivante contenant les produits du fournisseur numéro20:

Fournir_20(cod_frs#, cod_prd#, prix_unitaire)

insert into Fournir_20

select * from fournir where cod_frs = 20 ;

3.2. MODIFICATION DE DONNEES

Il s'agit de modifier les valeurs d'une ou plusieurs colonnes d'une ou plusieurs lignes d'une table.

Syntaxe :

UPDATE nom_table

SET nom_col1 = {expression1 | (requête)},

nom_col2 = {expression2 | (requête)}

WHERE condition ;

Si la clause where n'est pas indiquée, toutes les lignes de la table seront modifiées.

Exemple :

Update fournisseur set adresse_frs ='Sfax'

where cod_frs = 20 ;

3.3. SUPPRESSION DE DONNEES

Il s'agit de supprimer une ou plusieurs lignes d'une table.

Syntaxe :

DELETE nom_table WHERE condition;

Exemple :

Delete from fournir

Where cod_frs = 20 ;

3.4. CONSULTATION DE DONNEES

Syntaxe simplifiée:

SELECT col1/expr1,col2/expr2,..,coln/exprn

FROM nom_table

[WHERE condition] ;

La clause SELECT indique la liste des informations à exploiter.

La clause FROM indique les tables nécessaires à partir desquelles les données seront extraites.

La clause WHERE permet d'extraire uniquement les lignes qui vérifient une condition donnée.

Pour afficher toutes les colonnes on utilise le caractère '*'.

Exemple :

Select cod_prd from fournir ;

Select * from fournir where cod_frs = 20 ;

Remarque :

Pour éliminer les doubles il faut utiliser la clause (distinct/unique).

Exemple :

Select distinct cod_prd from fournir ;

3.4.1. Opérateurs de comparaison

WHERE peut utiliser les opérateurs de comparaison qui sont offerts par SQL tels que :

=, != / <>,<,>,<=,>=

is null, is not null,

[not] { In(liste de valeur), like (chaîne générique), Between val1 and val2

= / != / <> /< / > / <= / >= any/all

IS NULL : permet de tester si une valeur est indéfinie ou non.

IN (liste de valeurs) : permet de tester la présence d'une valeur dans une liste de valeurs.

BETWEEN val1 AND val2 : permet de tester l'appartenance d'une valeur entre les valeurs val1 et val2.

LIKE chaîne générique : l'opérateur LIKE permet de comparer la valeur d'une colonne avec une chaîne générique en utilisant des caractères génériques de substitution:

% : remplace zéro ou plusieurs caractères pour une chaîne.

- : remplace un seul caractère.

ALL : permet de tester une valeur avec chacune des valeurs de l'ensemble et retourne "VRAI" si la comparaison est évaluée pour chacun des éléments.

ANY : permet de tester une valeur avec chacune des valeurs de l'ensemble et retourne "VRAI" si la comparaison est évaluée à "VRAI" pour au moins un des éléments.

Exemples :

- Afficher les fournisseurs habitant à Sfax.

Select * from fournisseur where upper(adresse_frs) like '% Sfax%';

- Afficher les produits fournis par au moins d'un fournisseur de Sfax.

Select cod_prd from fournir where cod_frs = any (select code_frs from fournisseur where upper(adresse_frs) like '% Sfax%');

Ou bien :

Select cod_prd

from fournir

where cod_frs in (select code_frs

from fournisseur

where upper(adresse_frs) like '% Sfax%');

- Afficher la désignation et le prix des produits dont le prix unitaire est compris entre 600 et 1500.

Select lib_prd, prix_unitaire

from produit

where prix_unitaire BETWEEN 600 and1500;

3.4.2. Les Expressions Arithmétiques

Dans la clause SELECT et WHERE, on peut utiliser des expressions arithmétiques et des fonctions telles que :

* ABS(n) : permet de calculer la valeur absolue de n.

* CEIL(n) : permet d'avoir le plus petit entier supérieur ou égal à n.

* FLOOR(n) : permet d'avoir la partie entière de n.

* MOD(n,m) : permet d'avoir le reste de la division entière de m par n.

* ROUND(m,n) : permet d'arrondir la valeur n à m décimal.

* POWER(m,n) : permet d'avoir m puissance n.

* SIGN(n) : donne -1 si n < 0, 0 si n =0 et 1 si n> 1.

* SQRT(n) : permet d'avoir racine carrée de n.

* TRUNC(m,n) : permet de tronquer la valeur après m décimales. Si n est négatif, la valeur de m est tronquée avant le point décimal.

3.4.3. Fonctions standards

Les fonctions couramment supportées sont :

COUNT permet d'avoir le nombre de valeurs

SUM permet d'avoir la somme de valeurs

AVG permet d'avoir la moyenne de valeurs

MAX permet d'avoir la plus grande valeur d'une expression sur une colonne

MIN permet d'avoir la plus petite valeur d'une expression sur une colonne

NVL : permet de teste la nullité d'une valeur

DECODE : permet la simulation de test à Choix Multiple

….

LA FONCTION TO_CHAR:

Syntaxe : TO_CHAR(*expr* , ['*format*'])

Convertit l'expression ***expr*** , représentant un nombre ou une date, en un caractère selon le format spécifié.

Lorsque le format '***format***' est omis, si ***expr*** représente une date elle sera convertie selon le format standard ORACLE; si ***expr*** est un nombre elle sera convertie en une chaîne de caractère de longueur assez suffisante pour représenter les digits significatifs.

'***format*':** est une combinaison quelconque des codes suivants:

YYYY	Année sans virgule
YYY	3 derniers chiffres de l'année

YY	2 derniers chiffres de l'année
Y	Dernier chiffre de l'année
YEAR	Année en toute lettre
Q	Numéro de trimestre de l'année (1 à 4)
MM	Numéro du mois (1 à 12)
MONTH	Nom du mois abrégé en toute lettre
WW	Numéro de semaine de l'année (1 à 52)
W	Numéro de semaine dans le mois
DDD	Numéro de jour dans l'année (1 à 366)
DD	Numéro du jour dans le mois (1 à 31)
D	Numéro de jour dans la semaine (1 à 7)
DAY	Nom du jour sur 9 caractères
DY	Nom du jour abrégé en 3 lettres
HH	Heures
MI	Minutes
SS	Secondes.

etc…

Note: Le format par défaut de la date dépend de l'implémentation,

- *"DD-MON-YY" pour l'Anglais comme "28-DEC-99"*
- *"JJ/MM/AA" pour le Français comme "28/12/99".*

Exemple:

```
SELECT  TO_CHAR(SYSDATE,'DD MM YYYY HH:Mi' )
FROM DUAL;
```

LA FONCTION NVL :

Cette fonction permet de substituer les valeurs nulles d'une colonne avec d'autres valeurs.

Par exemple, l'expression SAL + COMM pourrait engendrer un problème si la colonne commission est nulle : naturellement on ne peut additionner une valeur inconnue (nulle).

Pour résoudre ce problème, la fonction **NVL** doit être appliquée à la colonne COMM pour substituer le *nul* par une autre valeur - comme zéro – dans le calcul.

Syntaxe :

NVL (expression, val)

La fonction NVL simule un test de la forme :

SI *expression* nulle **ALORS**

Retourne ***val***

SINON

Retourne ***expression***

FinSi

Exemple:

Soit la table suivante : emp(empno, ename,sal, comm)

SELECT ENAME, SAL, COMM, SAL+NVL(COMM,0) “REVENU MENSUEL”

FROM EMP;

Sans utilisation de NVL, l'expression SAL+COMM prend une valeur indéfinie pour les employés sans commissions.

LA FONCTION DECODE :

SQL est dépourvu de structures algorithmiques.

DECODE simule un test à choix multiple.

Syntaxe :

DECODE (*expression, val1, Res1 [,val2, Res2]....., default*)

Si *expression = val1* Alors Retourne *Res1*

= val2 Alors Retourne *Res2*

autrement Retourne *default*

FinSi

Exemple :

Afficher cod_frs et nom_frs pour tous les fournisseurs sauf pour le cas du fournisseur ali, remplacer le nom par '***'

SELECT cod_frs, decode(nom_frs,'ali','***',nom_frs)

From fournisseur;

Quelques fonctions s'appliquant à des chaînes de caractères :

* RTRIM(ch) : supprime l'espace à la fin de la chaîne à l'opposé de LTRIM.

* RPAD(ch,n) : ajoute n espaces à la fin de la chaîne à l'opposé de LPAD.

* INITCAP(ch) : met en majuscule la première lettre de chaque mot de la chaîne.

* INSTR(ch1,ch2,[n[,m]]) : cherche la sous-chaîne ch2 dans la chaîne ch1 à partir de la position n et si l'on souhaite à partir de la m ème occurence de ch2 dan ch1. n et m sont optionnels et ont pour valeur par défaut 1.

* LENGTH(ch) : renvoie la longueur d'une chaîne.

* LOWER (ch) : transform la chaîne ch en miniscule à l'opposé de UPPER.

* SUBSTR(ch,m,n) : extraire la sous-chaîne de ch qui commence à partir du caractère à la position m et de longueur n.

* TRANSLATE(ch,ch1,ch2) : permet de transformer dans la chaîne ch toutes les occurences de ch1 par ch2.

* REPLACE(chaine,ch1 [,ch2]) : remplace une chaîne par une autre dans une colonne. Si on ne met pas ch2, ch1 va être emplacée par un vide.

* ch1 ||ch2 : concatène les deux chaînes.

Quelques fonctions s'appliquant à des Dates

* ADD_MONTHS(d,n) : ajoute n mois à la date (n est un entier).

* GREATEST(d1,d2) : donne la date la plus récente parmi d1 et d2 : c'est l'opposé de LEAST.

* MONTHS_BETWEEN(d1,d2) : donne le nombre de mois qui se trouvent entre les deux dates.

* LASR_DAY(d) : donne la date du dernier jour du mois de cette date.

* SYSDATE : donne la date et l'heure système.

3.4.4. Tri de résultat

Pour ordonner le résultat avant l'affichage on utilise la clause ORDER BY.

Syntaxe :

SELECT col1, col2,…,col2

FROM nom_table

[Where condition]….

ORDER BY co1,col2,…,coli[ASC/DESC] ;

Ou bien

ORDER BY 1,2,…,i[ASC/DESC] ;

ASC : (ASCending) dans l'ordre croissant par défaut.

DESC : (DESCending) dans l'ordre décroissant.

Exemples :

Liste des produits classés par ordre décroissant de leur prix unitaire.

Select * from produit order by prix_unitaire desc ;

3.4.5. Jointure de tables

JOINTURE AVEC SQL2

Il s'agit de joindre des n-uplets issues de deux ou plusieurs tables en vue de retrouver des données associées.

Pour effectuer une jointure, il faut spécifier :

- Les noms des tables, dans la clause FROM, séparés par des virgules.
- La condition de jointure dans la clause WHERE en préfixant les noms de colonnes par les noms de tables pour éviter toute ambiguïté.

Syntaxe :

SELECT col1, col2,….,coln

FROM table1 , table2,…, table n

WHERE condition

Exemples :

Afficher le code, le libellé, le code fournisseur et le nom de fournisseur de tous les produits.

```
Select produit.cod_prd, produit.lib_prd, fournir.cod_frs, fournisseur.nom_frs
 from produit, fournir, fournisseur
where  produit.cod_prd = fournir.cod_prd
and fournir.cod_frs = fournisseur.cod_frs ;
```

Ou bien

```
Select p.cod_prd, p.lib_prd, fr.cod_frs, f.nom_frs
 from produit p, fournir fr, fournisseur f
where  p.cod_prd = fr.cod_prd
and fr.cod_frs = f.cod_frs ;
```

Jointure externe

La jointure externe de deux tables T1 et T2 est une jointure particulière retournant le résultat normal d'une jointure augmenté par tous les tuples issues de T1 (respectivement T2) pour lesquels il ne correspond aucune ligne dans T2 (respectivement T1).

Syntaxe :

```
SELECT col1,col2,...coln
FROM T1 , T2(+)
WHERE condition ;
```

Le (+) concatène une ligne vide à chacun des tuples de T1 qui ne participent pas à la jointure.

Exemple :

Afficher pour chaque produit la somme des quantités commandées même si le produit n'était pas commandé par aucune commande.

```
select p.cod_prd, sum(quantite)
from produit p , ligne_commande l
where p.cod_prd=l.cod_prd (+)
```

group by p.cod.prd;

JOINTURE AVEC SQL3

Trois syntaxes possibles de l'expression d'une jointure dans la clause FROM en SQL 3 :

table_1 { [INNER] { LEFT | RIGHT | FULL } [OUTER] } JOIN table_2 ON predicat [...]

table_1 { [INNER] { LEFT | RIGHT | FULL } [OUTER] } JOIN table_2 USING (colonnes) [...]

table_1 NATURAL { [INNER] { LEFT | RIGHT | FULL } [OUTER] } JOIN table_2 [...]

Ces trois syntaxes diffèrent par la condition de jointure spécifiée par les clauses ON ou USING, ou implicite dans le cas d'une jointure naturelle introduite par le mot-clé NATURAL.

ON :

La clause ON correspond à la condition de jointure la plus générale. Le prédicat predicat est une expression logique de la même nature que celle de la clause WHERE décrite dans la section 4.5.7.

USING :

La clause USING est une notation abrégée correspondant à un cas particulier de la clause ON. Les deux tables, sur lesquelles portent la jointure, doivent posséder toutes les colonnes qui sont mentionnées, en les séparant par des virgules, dans la liste spécifiée entre parenthèses juste après le mot-clé USING. La condition de jointure sera l'égalité des colonnes au sein de chacune des paires de colonnes. De plus, les paires de colonnes seront fusionnées en une colonne unique dans la table résultat de la jointure. Par rapport à une jointure classique, la table résultat comportera autant de colonnes de moins que de colonnes spécifiées dans la liste de la clause USING.

NATURAL :

Il s'agit d'une notation abrégée de la clause USING dans laquelle la liste de colonnes est implicite et correspond à la liste des colonnes communes aux deux tables participant à la jointure. Tout comme dans le cas de la clause USING, les colonnes communes n'apparaissent qu'une fois dans la table résultat.

INNER et OUTER :

Les mots-clé INNER et OUTER permettent de préciser s'il s'agit d'une jointure *interne* ou *externe*. INNER et OUTER sont toujours optionnels. En effet, le comportement par défaut est celui de la jointure interne (INNER) et les mots clefs LEFT, RIGHT et FULL impliquent forcément une jointure externe (OUTER).

INNER JOIN :

La table résultat est constituée de toutes les juxtapositions possibles d'une ligne de la table table_1 avec une ligne de la table table_2 qui satisfont la condition de jointure.

LEFT OUTER JOIN :

Dans un premier temps, une jointure interne (*i.e.* de type INNER JOIN) est effectuée. Ensuite, chacune des lignes de la table table_1 qui ne satisfait pas la condition de jointure avec aucune des lignes de la table table_2 (*i.e.* les lignes de table_1 qui n'apparaissent pas dans la table résultat de la jointure interne) est ajoutée à la table résultats. Les attributs correspondant à la table table_2, pour cette ligne, sont affectés de la valeur NULL. Ainsi, la table résultat contient au moins autant de lignes que la table table_1.

RIGHT OUTER JOIN :

Même scénario que pour l'opération de jointure de type LEFT OUTER JOIN, mais en inversant les rôles des tables table_1 et table_2.

FULL OUTER JOIN :

La jointure externe bilatérale est la combinaison des deux opérations précédentes (LEFT OUTER JOIN et RIGHT OUTER JOIN) afin que la table résultat contienne au moins une occurrence de chacune des lignes des deux tables impliquées dans l'opération de jointure.

3.4.6. Les opérateurs ensemblistes

On peut articuler les résultats de plusieurs requêtes homogènes à l'aide des opérations ensemblistes union, intersect et minus :

- **L'union**

L'union porte sur deux relations de même schéma.

Le résultat donne une troisième relation de même schéma et ayant comme enregistrements ceux de la première relation avec ceux de la deuxième relation.

Syntaxe :

Requête 1

UNION

Requête 2 ;

Exemple :

Afficher les produits dont la quantité de stock est supérieure à 500 et ceux dont le prix unitaire est supérieur à 200 :

Select * from produit

Where quantite_stock>500

Union

Select * from produit

Where prix_unitaire >200.

- **L'intersection**

L'intersection porte sur deux relations de même schéma.

Le résultat donne une troisième relation de même schéma et ayant comme enregistrements ceux qui existe dans la première relation et dans la deuxième relation à la fois.

Syntaxe :

Requête 1

INTERSECT

Requête 2 ;

Exemple :

Afficher les produits dont la quantité de stock est supérieure à 500 et dont le prix unitaire est supérieur à 200 :

Select * from produit

Where quantite_stock>500

intersect

Select * from produit

Where prix_unitaire >200.

- **La différence**

La différence porte sur deux relations de même schéma.

Le résultat donne une troisième relation de même schéma et ayant comme enregistrements ceux de la première relation et qui n'existe pas dans la deuxième relation.

Syntaxe :

Requête 1

MINUS

Requête 2 ;

Exemple :

Afficher les produits dont la quantité de stock est supérieure à 500 et dont le prix unitaire est inférieur à 200 :

```
Select * from produit

Where quantite_stock>500

minus

Select * from produit

Where prix_unitaire >=200.
```

3.4.7. Les requêtes imbriquées

Les requêtes imbriquées sont des requêtes à l'intérieur d'autres requêtes. Ceci se produit lorsque la clause WHERE d'une requête contient elle-même une sous requête.

Exemple :

Afficher les produits qui ont le même prix unitaire que le produit numéro 1.

```
Select  * from produit

where prix_unitaire  = (select prix_unitaire
                        from produit
                        where cod_prd=1) ;
```

Lorsque la sous requête est susceptible de retourner plus qu'une ligne, l'opérateur de comparaison doit permettre de comparer une valeur avec un ensemble de valeurs. Dans ce cas les opérateurs recommandés peuvent être :

- IN

- = / != / > / >= / < / <= ANY / ALL

Exemple :

Afficher les produits qui ont une quantité en stock > à la quantité des produits qui ont un prix unitaire supérieur à 500.

```
Select * from produit
where quantite_stock > all (select quantite_stock
                            from produit
                            where prix_unitaire >500) ;
```

3.4.8. La clause GROUP BY

La clause GROUP BY divise l'ensemble des lignes d'une table en groupes.

Utilités :

- Calculer un résultat par groupe,
- Exprimer une condition qui porte sur un Groupe

Syntaxe :

```
SELECT col1,col2,…,coln
FROM table1, table2,…,tablen
WHERE condition
GROUP BY expression
HAVING condition;
```

Exemples:

- Afficher le nombre de commandes par fournisseur :

```
Select cod_frs, count (num_cde)
```

From commande

Group by cod_frs;

- Afficher le montant de chaque commande

Select num_cde , sum(quantité * prix_unitaire)

From commande, ligne_commande, produit

Where commande. num_cde= ligne_commande.num_cde

And produit.cod_prd= ligne_commande.cod_prd

Group bynum_cde;

- Afficher les commandes dont le montant est >1500

Pour demander une condition sur un groupe (construit avec GROUP BY), il faut absolument utiliser la clause HAVING avec l'une des fonctions de groupe comme prédicat.

Select num_cde

From commande, ligne_commande, produit

Where commande. num_cde= ligne_commande.num_cde

And produit.cod_prd= ligne_commande.cod_prd

Group bynum_cde

Having sum(quantité * prix_unitaire)>1500 ;

3.5. LES VUES

Une vue est une table virtuelle construite à partir d'autres tables. Le système stocke uniquement son schéma de définition.

L'utilisation des vues permet de :

- En cas de requêtes complexes, l'utilisation des vues sera souhaitable.
- Restreindre l'accès à certaines colonnes ou certaines lignes d'une table ;

3.5 .1.Création de vues

Syntaxe :

CREATE VIEW *vue [(colonnel, ...,colonneN)]*

As *requête*
[WITH CHECK OPTION] ;

La clause **WITH CHECK OPTION** permet de vérifier si la mise à jour réalisée à partir de la vue respecte sa condition de sélection ou non**.**

Si cette clause est omise aucune vérification n'aura lieu.

Exemple :

Créer une vue correspondant aux fournisseurs de la ville de Sfax.

```
CREATE VIEW V1
AS    SELECT  *
      FROM fournisseur
      WHERE adresse_frs like '%Sfax%;
```

3.5.2. Suppression de vues

Syntaxe :

DROP VIEW *nom_de_la_vue* ;

3.5.3. Mise à jour à partir des vues

Les opérations de mise à jour ne sont pas permises si la vue dérive de plusieurs tables ou si sa définition contient la clause GROUP BY, la clause DISTINCT ou encore une fonction de groupe.

- Si un attribut déclaré NOT NULL dans la table de base n'est pas repris dans la vue, aucune INSERT n'est possible sur cette vue.

4. LE LANGAGE DE CONTRÔLE DE DONNEES (LCD)

4.1. CREATION D'UTILISATEURS

Le seul qui permet de gérer les utilisateurs c'est l'administrateur du système.

Syntaxe générale de création d'un utilisateur:

Grant connect, resource [, dba]

to nom_utilisateur

identified by mot de_passe ;

L'option **CONNECT** autorise l'utilisateur à :

- ✓ Se connecter à ORACLE.
- ✓ Créer des vues et des synonymes sur des tables existantes.
- ✓ Sélectionner, modifier et supprimer des données dans des tables sur lesquelles une autorisation a été donnée au préalable.

L'option **RESOURCE** ne peut être attribuée que si l'utilisateur a le droit CONNECT. Elle lui permet :

- ✓ De créer des tables, des index et des regroupements (des clusters).
- ✓ Attribuer ou enlever des privilèges sur les tables, index et clusters à d'autres utilisateurs.

L'option **DBA** englobe les droits des deux options précédentes et permet en plus :

- ✓ D'accéder aux données de tous les utilisateurs de la même base.
- ✓ De créer et de supprimer des utilisateurs et des droits.
- ✓ D'administrer la base (sauvegarder et restaurer des données de la base, contrôler l'intégrité,...).

4.2. CREATION ET SUPPRESSION DES DROITS

La protection des objets d'une base de données est décentralisée : c'est le créateur d'un objet qui possède tous les droits de lecture, de modification et de suppression de cet objet. Les autres utilisateurs n'ont aucun droit d'accès à cet objet, sauf si le créateur leur accorde explicitement des droits par une commande GRANT :

GRANT privilege ON {nom_table | nom_vue} TO nom_utilisateur
WITH GRANT OPTION ;

Les privilèges pouvant être accordés sont entre autres les suivants :

SELECT : consultation
INSERT : ajout des lignes
UPDATE(col,...) : modification des lignes (peut-être restreint à certaines colonnes)
DELETE : suppression des lignes
ALTER : modification de la définition de la table
ALL : tous les droits ci-dessus

Un utilisateur ayant reçu un privilège avec l'option GRANT peut le transmettre à son tour (WITH GRANT OPTION).

Exemple :

L'utilisateur scott peut autoriser l'utilisateur ali à lire sa table emp.

GRANT SELECT ON emp TO ali ;

Les droits peuvent être accordés à tous les utilisateurs en employant le mot réservé PUBLIC à la place du nom d'utilisateur.

GRANT SELECT, UPDATE ON emp TO PUBLIC;

Un utilisateur ayant accordé un privilège peut le reprendre à l'aide de l'ordre REVOKE.

REVOKE PRIVILEGE ON {nom_table | nom_vue} FROM nom_utilisateur;

Le propriétaire d'une table peut donner des droits de lecture non pas sur la table entière, mais sur une vue basée sur la table. Ainsi, seules les informations faisant partie de la vue seront accessibles.

```
CREATE VIEW emp1 AS
SELECT ename, job, hiredate
FROM scott.emp
WHERE n_dept != 10 ;
```

GRANT SELECT ON emp1 TO PUBLIC;

Tous les utilisateurs pourront sélectionner les employés à travers la vue emp1 : ils ne verront que les colonnes nom, fonction, embauche de la table emp et n'auront pas accès aux employés du département 10.

Pour Oracle, le nom complet d'une table ou d'une vue est le nom donné par le créateur, préfixé par le nom du créateur. Par exemple scott.emp est le nom complet de la table emp créée par l'utilisateur scott. Ceci permet à plusieurs utilisateurs de créer des objets de même nom sans qu'il y ait confusion. En revanche, pour accéder à un objet dont on n'est pas le créateur, il faut le désigner par son nom complet incluant le nom du créateur.