Manipulation des données avec un langage de requêtes SQL

Technicien Spécialisé en Développement Informatique

I - L'ALGÈBRE RELATIONNELLE:

L'algèbre relationnelle a été inventée par E. Codd comme une collection d'opérations formelles qui agissent sur des relations et produisent des relations en résultats [Codd7O]. On peut considérer que l'algèbre relationnelle est aux relations ce qu'est l'arithmétique aux entiers. Cette algèbre, qui constitue un ensemble d'opérations élémentaires associées au modèle relationnel, est sans doute une des forces essentielles du modèle. Codd a initialement introduit huit opérations, dont certaines peuvent être composées à partir d'autres. Dans cette section, nous allons introduire six opérations qui permettent de déduire les autres et qui sont appelées ici opérations de base. Nous introduirons ensuite quelques opérations additionnelles qui sont parfois utilisées. Des auteurs ont proposé d'autres opérations qui peuvent toujours se déduire des opérations de base [Delobel83. Maier83].

Les opérations de base peuvent être classées en deux types: les opérations ensemblistes traditionnelles (une relation étant un ensemble de tuples, elle peut être traitée comme tel) et les opérations spécifiques.

Les opérations ensemblistes sont des opérations binaires, c'est-à-dire qu'à partir de deux relations elles en construisent une troisième. Ce sont l'union, la différence et le produit cartésien.

Les opérations spécifiques sont les opérations unaires de projection et restriction qui, à partir d'une relation, en construisent une autre, et l'opération binaire de jointure. Nous allons définir toutes ces opérations plus précisément.

1. OPERATIONS DE BASE

A - Opération PROJECTION

Formalisme : R = PROJECTION (R1, liste des attributs)

Exemples :

CHAMPIGNONS

Espèce	Catégorie	Conditionnement
Rosé des prés	Conserve	Bocal
Rosé des prés	Sec	Verrine
Coulemelle	Frais	Boîte
Rosé des prés	Sec	Sachet plastique

R1 = PROJECTION (CHAMPIGNONS, Espèce)

Espèce
Rosé des prés
Coulemelle

R2 = PROJECTION (CHAMPIGNONS, Espèce, Catégorie)

Espèce	Catégorie
Rosés des prés	Conserve
Rosé des prés	Sec
Coulemelle	Frais

- Cet opérateur ne porte que sur 1 relation.
- Il permet de ne retenir que certains attributs spécifiés d'une relation.
- On obtient tous les n-uplets de la relation à l'exception des doublons.

B - Opération SELECTION

Formalisme : R = SELECTION (R1, condition)

Exemple :

CHAMPIGNONS

Espèce	Catégorie	Conditionnement
Rosé des prés	Conserve	Bocal
Rosé des prés	Sec	Verrine
Coulemelle	Frais	Boîte
Rosé des prés	Sec	Sachet plastique

R3 = SELECTION (CHAMPIGNONS, Catégorie = "Sec")

Espèce	Catégorie	Conditionnement
Rosé des prés	Sec	Verrine
Rosé des prés	Sec	Sachet plastique

- Cet opérateur porte sur 1 relation.
- Il permet de ne retenir que les n-uplets répondant à une condition exprimée à l'aide des opérateurs arithmétiques (=, >, <, >=, <=, <>) ou logiques de base (ET, OU, NON).
- Tous les attributs de la relation sont conservés.
- Un attribut peut ne pas avoir été renseigné pour certains n-uplets. Si une condition de sélection doit en tenir compte, on indiquera simplement : nomattribut « non renseigné ».

C – Opération JOINTURE (équijointure)

Formalisme : R = JOINTURE (R1, R2, condition d'égalité entre attributs)

Exemple :

PRODUIT

CodePrd	Libellé	Prix unitaire
590A	HD 1,6 Go	1615
588J	Scanner HP	1700
515J	LBP 660	1820

DETAIL_COMMANDE

N°cde	CodePrd	quantité
97001	590A	2
97002	515J	1
97003	515J	3

R = JOINTURE (PRODUIT, DETAIL_COMMANDE,
Produit.CodePrd=Détail_Commande.CodePrd)

A.CodePrd	Libellé	Prix unitaire	N°cde	B.CodePrd	quantité
590A	HD 1,6 Go	1615	97001	590A	2
515J	LBP 660	1820	97002	515J	1
515J	LBP 660	1820	97003	515J	3

- Cet opérateur porte sur 2 relations qui doivent avoir au moins un attribut défini dans le même domaine (ensemble des valeurs permises pour un attribut).
- La condition de jointure peut porter sur l'égalité d'un ou de plusieurs attributs définis dans le même domaine (mais n'ayant pas forcément le même nom).
- Les n-uplets de la relation résultat sont formés par la concaténation des n-uplets des relations d'origine qui vérifient la condition de jointure.

Remarque : Des jointures plus complexes que l'équijointure peuvent être réalisées en généralisant l'usage de la condition de jointure à d'autres critères de comparaison que l'égalité (<,>, <=,>=, <>).

2 - Les opérations ensemblistes

a - Opération UNION

Formalisme : R = **UNION** (R1, R2)

Exemple :

E1 : Enseignants élus au CA

n° enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND

E2 : Enseignants représentants syndicaux

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN
6	MICHEL

On désire obtenir l'ensemble des enseignants élus au CA ou représentants syndicaux.

R1 = UNION (E1, E2)

n°enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND
6	MICHEL

- Cet opérateur porte sur deux relations qui doivent avoir le même nombre d'attributs définis dans le même domaine (ensemble des valeurs permises pour un attribut). On parle de relations ayant le même schéma.
- La relation résultat possède les attributs des relations d'origine et les n-uplets de chacune, avec élimination des doublons éventuels.

B - Opération INTERSECTION Formalisme

: R = **INTERSECTION** (R1, R2) Exemple :

E1 : Enseignants élus au CA

n° enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND

E2 : Enseignants représentants syndicaux

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN
6	MICHEL

On désire connaître les enseignants du CA qui sont des représentants syndicaux.

R2 = INTERSECTION (E1, E2)

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN

- Cet opérateur porte sur deux relations de même schéma.
- La relation résultat possède les attributs des relations d'origine et les n-uplets communs à chacune.

C - Opération DIFFERENCE Formalisme

: R = **DIFFERENCE** (R1, R2) Exemple :

E1 : Enseignants élus au CA

n° enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND

E2 : Enseignants représentants syndicaux

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN
6	MICHEL

On désire obtenir la liste des enseignants du CA qui ne sont pas des représentants syndicaux.

R3 = DIFFERENCE (E1, E2)

n°enseignant	nom_enseignant
3	DURAND
5	BERTRAND

- Cet opérateur porte sur deux relations de même schéma.
- La relation résultat possède les attributs des relations d'origine et les n-uplets de la première relation qui n'appartiennent pas à la deuxième.
- Attention ! DIFFERENCE (R1, R2) ne donne pas le même résultat que DIFFERENCE (R2, R1)

D - Opération PRODUIT CARTESIEN

Formalisme : R = **PRODUIT** (R1, R2)

Exemple :

Etudiants

n°étudiant	nom
101	DUPONT
102	MARTIN

Epreuves

libellé épreuve	coefficient
Informatique	2
Mathématiques	3
Gestion financière	5

Examen = PRODUIT (Etudiants, Epreuves)

n°étudiant	nom	libellé épreuve	coefficient
101	DUPONT	Informatique	2
101	DUPONT	Mathématiques	3
101	DUPONT	Gestion financière	5
102	MARTIN	Informatique	2
102	MARTIN	Mathématiques	3
102	MARTIN	Gestion financière	5

- Cet opérateur porte sur deux relations.
- La relation résultat possède les attributs de chacune des relations d'origine et ses n-uplets sont formés par la concaténation de chaque n-uplet de la première relation avec l'ensemble des n-uplets de la deuxième

II - Présentation du langage SQL

1. Création d'une table

La syntaxe élémentaire de création d'une table Oracle est:

CREATE TABLE nom_de table (définition d'une colonne, ,…);

Un nom de table doit être unique et ne doit pas être un mot clé SQL. Une table doit contenir au minimum une colonne et au maximum 254 colonnes.

Définition de colonne
La définition élémentaire d'une colonne consiste à lui attribuer un nom et un type. Les types de données gérés par Oracle sont:

CHAR(size)	Données de type caractère de longueur fixe size octets. Le maximum de la valeur de size varie en fonction de la version d'oracle, pour la version 8i, il est de 2000 octets.
VARCHAR2(size) ou **VARCHAR(size)**	Chaine de caractères ayant comme longueur maximale la valeur de size en caractères. Maximum de size est 4000, et le minimum est 1.
NUMBER(p,s)	p: nombre total de chiffres, p varie de 1 à 38 s : nombre de chiffres après la virgule (scale), s varie de -84 to 127.
LONG	Données de type caractère de taille variable allant jusqu'à 2 gigabytes (ou GO).
DATE	Donnée de type date.
RAW (size)	Données de type binaire ayant une longueur fixe de size bytes (ou octets) allant jusqu'à 2000 bytes.
LONG RAW	Données de type binaire ayant une longueur variable de size bytes (octets) allant jusqu'à 2 gigabytes.
ROWID	Chaine Hexadecimale representant l'adresse unique d'une ligne de la table. Sa valeur est retournée par la pseudo-colonne de même nom

Un exemple de création d'une table de nom Departement :

CREATE TABLE Departement (Dnum **NUMBER(3)**,Dname **VARCHAR2(15)**, DLoc **VARCHAR2(25)**);

Nous allons regarder avec plus de détails le type NUMBER. Soit le nombre **7456123.89**; pour chaque expression de NUMBER nous donnons le résultat généré:

NUMBER	7456123.89
NUMBER(9)	7456124
NUMBER(9,2)	7456123.89
NUMBER(9,1)	7456123.9
NUMBER(6)	exceeds precision
NUMBER(7,-2)	7456100
NUMBER(-7,2)	exceeds precision

Nous avons deux erreurs car on a une précision inférieure à la taille du nombre.
Lorsque la valeur de s est négative, Oracle arrondi le nombre à gauche du points suivant la valeur de **s** comme le montre le tableau précédent (cas du -2).

La valeur NULL
Pour spécifier que la valeur d'un champ est indéterminé Oracle utilise la clause NULL lors de la définition d'une colonne. Dans le cas de la table Departement, et pour indiquer que l'attribut Dloc est généralement indéterminé on peut écrire:

CREATE TABLE Departement (Dnum **NUMBER(3)**, Dname **VARCHAR2(15)**, Dloc **VARCHAR2(25) NULL);**

2 Insertion des données dans une table

2.2.1 Insertion de données externes
Une fois la table crée il s'agit d'y inserer des occurences et ce en utilisant l'instruction INSERT:

INSERT INTO nom_de table (col 1,… , col n) **VALUES** (val 1, ..., val n) **;**

Supposons que nous disposons de la table des fournisseurs F ayant la structure suivante: F (f# char(3), fnom varchar(20), statut number, ville varchar(20))
Si nous désirons insérer l'occurence relative au fournisseur f1 de nom smith ayant comme statut 20 et habitant Londre la requête SQL sera:

INSERT INTO F (f#,Fnom ,Statut, Ville) **VALUES** ('f1', 'smith', 20,'Londre' **);**

Remarques

1) Une chaine de caractère est insérée entre cote simple ' chaine'

2) Il est important d'avoir une correspondance entre l'ordre d'une valeur et l'ordre de la colonne associée.

3) Si on site tous les noms des colonnes on n'est pas obligé de se conformer à l'ordre selon lequel les colonnes ont été crées. La requête précédente donne le même résultat que celle qui suit:

INSERT INTO F (f#, Ville,Fnom ,Statut) **VALUES** ('f1','Londre', 'smith', 20 **);**

4) La syntaxe de l'instruction INSERT peut être simplifiée dans le cas où on a autant de valeur que de colonnes et en respectant l'ordre de presentation des colonnes et des valeurs. En effet les deux requêtes précedentes donnent le même résultat que celle qui suit:

INSERT INTO F **VALUES** ('f1', 'smith', 20,'Londre')**;**

Cette forme est possible car la structure de la table est composée de 4 colonnes et on a exactement une valeur pour chaque colonne. Ces valeurs sont en plus présentée dans l'ordre de création des colonnes.

5) Si l'ordre de présentation des valeurs n'est pas le même que celui de la définition ou si on a moins de valeurs que le nombre total des colonnes il faut revenir à la première syntaxe en mentionnant les colonnes concernées. Si par exemple pour le fournisseur f1 je n'ai que la ville la requête serait:

INSERT INTO F (f#,Ville) **VALUES** ('f1','Londre')**;**

3. consultation de données

3.1 Consultation simple

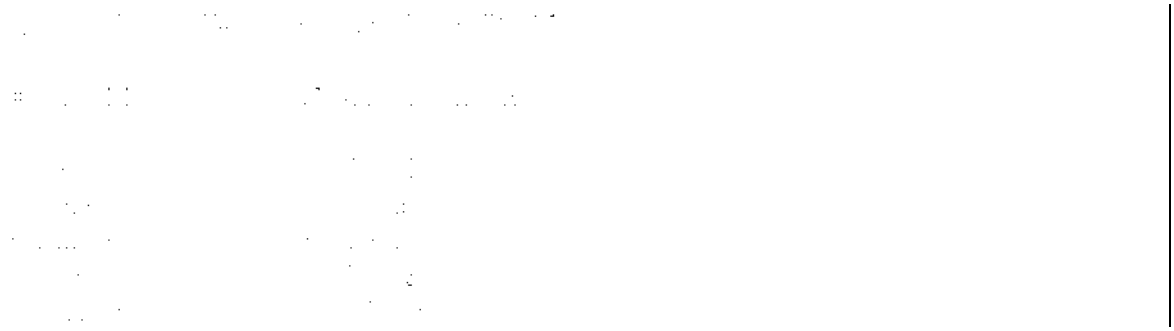
La syntaxe élémentaire de consultation d'une table Oracle est:

SELECT nom_colonne_1,… , nom_colonne_j **FROM** nom_de table;

Ainsi si on désire consulter la table F qui a cette structure F(F#,Fnom,Statut,Ville) on écrirait:

SELECT F#,Fnom,Statut,Ville **FROM** F;

Le résultat sous SQL*PLUS sera:



Lorsqu'on désire consulter tous les attributs d'une table on n'est pas obligé de les citer tous, on peut écrire:

SELECT * **FROM** nom_de table;

Il est à remarquer qu'on peut demander que les résultats soit affichés en étant triés par une ou plusieurs colonnes en utilisant la clause ORDER BY. La syntaxe du select devient:

SELECT nom_colonne_1,… , nom_colonne_j **FROM** nom_de table
[**ORDER BY** {expression/ position} [**ASC/DESC**]...];

On peut spécifier l'ordre de tri (ascendant ou descendant) relativement à une expression (généralement des colonnes) ou l'ordre de colonnes dans la requête select.

3.1.1 Consultation avec qualification

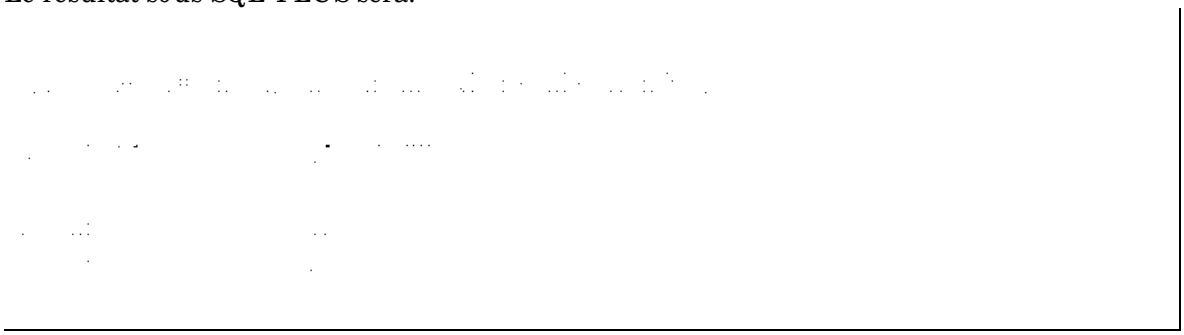
Si on désire consulter uniquement quelques lignes d'une table et ce relativement à une ou plusieurs conditions la syntaxe peut être enrichie et devient:

SELECT nom_colonne_1,… , nom_colonne_j **FROM** nom_de table
[**where** condition];

Ainsi pour consulter uniquement les fournisseur de londre on peut écrire:

SELECT F#,Fnom,Statut **FROM** F **WHERE** ville = 'Londre';

Le résultat sous SQL*PLUS sera:



Remarque:
1) Si on a écrit 'LONDRE' on n'aurait aucune réponse, car dans la phase de comparaison entre chaine de caractère, SQL tient compte des majuscules et minuscule. Ainsi 'LONDRE' est différente de 'LoNDre'.
2) Plusieurs conditions peuvent être combinées à l'aide des opérateurs logiques **AND** et **OR**.

3.2 Consultation avec critères

On peut répartir la consultation d'une table selon des critères en trois groupes:

1) Consultation avec condition de comparaison
2) Consultation avec condition en sous-requête
3) Consultation avec condition de jointure

3.2.1 Les conditions de comparaison

Nous pouvons citer différentes formes de spécification de condition de comparaison:
forme 1 :

expression [**NOT**] **BETWEEN** expression **AND** expression

forme 2 :

expression [**NOT**] opérateur-relationnel expression

forme 3 :

expression [**NOT**] **IN** (liste d'expression/ requête)

forme 4 :

colonne [**NOT**] **LIKE** 'chaine'

forme 5 :

colonne **IS** [**NOT**] **NULL**

Le terme expression peut désigner un nom de colonne, une constante (numérique ou caractère), une pseudo-colonne, une valeur nulle ou une combinaison de ces éléments par des opérateurs arithmétiques (+, -, *, /).
Exemple: Soit la table des pièces suivantes

P	P#	Pnom	Couleur	Poids	Ville
	p1	Nut	Rouge	12	Londre
	p2	Boltes	Vert	17	Paris
	p3	Screw	Bleue	17	Rome
	p4	Screw	Rouge	14	Londre
	p5	Cam	Bleue	12	Paris
	p6	Cog	Rouge	19	Londre

Si on désire avoir les numéros des pièces ayant un poids entre 14 et 17 il suffit d'écrire la requête suivante:

SELECT p# **FROM** P **WHERE** poids **BETWEEN** 14 **AND** 17;

Si on désire avoir les numéros des pièces ayant un poids supérieur à 14 il suffit d'écrire la requête suivante:

SELECT p# **FROM** P **WHERE** poids > 14;

Oracle offre les opérateurs relationnels suivants:

=; !=; <>; >; <=; >=; <;

Si on désire avoir les numéros des pièces ayant un poids égal à 14 ou à 17 il suffit d'écrire la requête suivante:

SELECT p# **FROM** P **WHERE** poids **IN** (14,17);

Oracle permet aussi la comparaison de deux listes d'expressions qui doivent avoir le meme nombre d'éléments.Dans cette liste les valeurs de type caractère et date doivent être entre cotes.

Si on désire avoir les numéros des pièces dont le nom commence par 'S' il suffit d'écrire la requête suivante:

SELECT p# **FROM** P **WHERE** pnom **LIKE** 'S%';

Remarque:
Le signe '%' indique que le reste de la chaine peut comprendre de 0 à n caractères. Si on désire qu'il y ait exactement un caractère on peut utiliser le symbole '-'.
Les caratères de substitution '-' et '%' peuvent être précédés du ssigne '\' si on désire les utiliser pour leur juste valeur.
La dernière forme peut être utilisée pour tester si le contenu d'une colonne est indéterminée ou non. Il ne faut jamais le tester avec le signe '=' mais avec IS ou bien IS NOT.

3.3 Consultation avec jointures

Une jointure est un lien entre tables ayant au moins une colonne en commun. Le système crée une table temporaire composée des lignes répondants à la condition de jointure. Nous distinguons quatres types de jointures:

equijointure
thétajointure
jointure multiple
autojointure

3.3.1 Equijointure

Une équijointure est une jointure dont la comparaison est une égalité entre deux colonnes appartenant à deux tables différentes.

Exemple: Soit les tables P et FPJ qui ont les structures suivantes:

P	P#	Pnom	Couleur	Poids	Ville

FPJ	F#	P#	J#	Qty

Si on désire avoir les villes des pièces qui ont été fournies il suffit d'écrire la requête suivante:

SELECT ville **FROM** P, FPJ **WHERE** P.p#=FPJ.p#;

Remarques:

- Notons l'invocation des deux tables P et FPJ même si l'attribut résultat existe uniquement en P, mais comme on utilise des attributs de FPJ, elle doit être active dans la requête.
- La notation P.p# est utilisée pour qu'il n' y ait pas de confusion entre attributs car le même nom existe dans les deux tables P et FPJ.

3.3.2 Thétajointure

Une thétajointure est une jointure dont la condition est une comparaison entre deux colonnes appartenant à deux tables différentes qui utilise un autre opérateur que l'égalité.

Exemples: Soit la table des projets ayant la structure suivante:

J	J#	Jnom	Ville

Si on désire avoir les numéros des pièces et projets qui n'ont pas la même ville il suffit d'écrire la requête suivante:

SELECT p#, j# **FROM** P, J **WHERE** P.ville !=J.ville;

3.3.3 Jointure multiple

Une jointure multiple est une jointure qui met en relation plusieurs colonnes appartenant à plusieurs tables.

Exemple: Si on désire avoir les numéros des pièces qui ont été fournies par des fournisseurs de Londre il suffit d'écrire la requête suivante:

SELECT p# **FROM** P, J, FPJ **WHERE** J.ville ='Londre' and P.p#=FPJ.p# and J.j#=FPJ.j#;

3.3.4 Autojointure

Une autojointure est une jointure d'une table avec elle-même. Le problème est qu'on travaille sur la même table et donc on aura à utiliser les même noms de colonnes. La solution de nom de table suivi du nom de colonne (ex: P.ville) n'est pas bonne ici car on est sur la même table. Une solution est d'utiliser la notion de variable table:
l'utilisateur déclare dans sa requête une variable par table puis il fait précéder ce nom par celui de la colonne à utiliser. C'est comme si on a fait une copie de la table en question. Ces variables n'ont d'existence que durant la requête.

Exemples: Donner les numéros des pièces dont la quantité en stock du premier est le double de celle du second article:
La requête suivante peut répondre à ce problème:

SELECT X.p#, Y.p# **FROM** P X, P Y **WHERE** X.qty=2*Y.qty;

3.4 Les sous-requête

SQL permet de comparer une expression ou une colonne au résultat d'une autre requête select. Cette condition est dite condition de sous requête et les deux requêtes sont dites requêtes imbriquées.
Les formes générales des sous-requêtes suivent généralement le WHERE elles se présentent comme:

WHERE expression opérateur_de_comparaison {**ALL/ANY/SOME**} (requête SELECT)
WHERE expression [**NOT**] **IN** (requête SELECT)
WHERE [**NOT**] **EXISTS** (requête SELECT)

Remarques:
- Les sous-requêtes situées après les mots clefs IN,ALL,ANY et SOME doivent avoir le même nombre de colonnes que celui spécifié dans l'expression.
- Ces sous-requêtes peuvent rendre plusieurs valeurs qui seront évaluées en fonction de l'opérateur de comparaison:

ALL/ANY/SOME: la condition est vraie si la comparaison est vraie pour chacune des valeurs retournées par la sous-requête.

Exemple: Pour cet exemple nous supposons que la structure de la table P des pièces est:

P	P#	Pnom	Couleur	Poids	Ville	qtstck

Le dernier attribut indique la quantité en stock de l'article.
La table FPJ a la structure suivante:

FPJ	F#	P#	J#	Qty

Si nous désirons lister les pièces dont la quantité en stock est supérieure à toute quantité commandée on peut écrire:

SELECT p#, pnom **FROM** p **WHERE** qtstck > **ALL**
(**select** qty **from** FPJ **where** P.p#= FPJ.p#);

IN: la condition est vraie si la comparaison est vraie pour une des valeurs retournées par la sous-requête.

EXISTS: renvoie le boolean vraie ou faux selon le résultat de la sous-requête. Si l'évaluation de la sous-requête donne lieu à un ou plusieurs lignes, la valeur retournées est vraie.

3.5 Fonctions d'agrégat et groupements

3.5.1 Les fonctions d'agrégat

SQL permet de consulter les contenus des attributs avec aussi l'application d'opérations arithmétiques et ce via un certains nombre de fonctions appelées fonctions d'agrégats dont nous citons quelques unes.

La fonction **COUNT**

Cette fonction a deux formes syntaxiques:

COUNT (*): donne le nombre de lignes satisfaisants la requête.

COUNT ([DISTINCT/ALL] colonne): donne le nombre de valeur de colonne satisfaisants la requête.

La fonction **SUM**

SUM ([DISTINCT/ALL] colonne) : donne la somme des valeurs de la colonne répondant à la requête. La colonne doit être de type numérique. L'option DISCTINCT somme les valeurs uniques de la colonne.

La fonction **AVG**

AVG ([DISTINCT/ALL] colonne) : donne la moyenne des valeurs de la colonne répondant à la requête. La colonne doit être de type numérique. L'option DISCTINCT calcule la moyenne des valeurs distinctes de la colonne.

La fonction **MAX**

MAX ([DISTINCT/ALL] colonne) : donne le maximum relatif à la colonne.

La fonction **MIN**

MIN ([DISTINCT/ALL] colonne) : donne le minimum relatif à la colonne.

La fonction **STDDEV**

STDDEV ([DISTINCT/ALL] colonne) : donne l'écart type relatif à la colonne.

La fonction **VARIANCE**

VARIANCE ([DISTINCT/ALL] colonne) : donne la variance relative à la colonne.

Remarques:

- la colonne paramètre de ces fonctions, peut elle aussi, être une expression de SUM, AVG, MAX, MIN, STDDEV.
- la fonction COUNT comptabilise les occurences ayant une valeur NULL, ce qui n'est pas le cas pour les autres fonctions.
- d'autres fonctions sont aussi offertes par SQL relatives à la manipulation des chaines de caractères et de calcul simple.

3.5.2 Le groupement

SQL permet de grouper des lignes de données ayant des valeurs communes via la clause GROUP BY associée à la clause SELECT et la syntaxe est:

SELECT nom_colonne_1,... , nom_colonne_j **FROM** nom_de table
[**WHERE** condition]
[**GROUP BY** liste d'expression [**HAVING** condition]];

Remarques:
- La liste de sélection peut contenir uniquement les types des expressions suivantes:
* constantes
* fonctions d'agrégat
* expression identique à celle spécifiée dans la clause group by
* expression évaluant la même valeur por toutes les lignes dans un groupe.
-l'expressions de la clause GROUP BY peut contenir toute colonne faisant partie de la liste de tables de la clause FROM.

Exemple : Soit la structure de la table FPJ:

FPJ	F#	P#	J#	Qty

S'il s'agit d'écrire la requête qui donne pour chaque pièce le nombre de fois où elle a été commandée:

SELECT p#, count(*) from FPJgroup by p#;

3.5.2.1 Le groupement avec HAVING

Il est possible d'éliminer avec a clause group by les groupes ne vérifiant pas une condition donnée et ce en utilisant la clause HAVING.
La condition permet de comparer une valeur du groupe à une constante ou à une autre valeur résultante d'une sous-requête.
Une contrainte syntaxique est que:a condition WHERE ne doit pas contenir de fonction d'agrégat :
on a alors recours à la clause HAVING.

Exemple :Soit la table **commande**

numcom	idarticle	datecom

Si on désire avoir la somme des quantités commandées par produit et par client sans inclure les produits dont le cumul des commandes par client soit inférieur à 10 on peut écrire:

```
SELECT idproduit,idclient, SUM(qtecom)
FROM commande GROUP BY idproduit, idclient
HAVING SUM(qtecom)>=10;
```

Attention on ne peut pas écrire WHERE SUM...

3.6 Les opérateurs ensemblistes

Les bases de données relationnelles adoptent les concepts du langage algébrique, il est donc possible d'effectuer les opération ensemblistes sur les tables.
Les instructions de manipulation de données de SQL permettent de réaliser différentes opérations.

3.6.1 L'union

Il est possible de fusionner avec SQL des données résultants de plusieurs requêtes. Les résultats de ces requêtes peuvent être unis en utilisant la clause UNION dont la syntaxe est:

```
requête SELECT
UNION [ALL] requête SELECT
[UNION [ALL] requête SELECT...]
```

Remarques :
- par défaut l'opérateur UNION supprime toutes les redondances sauf si l'option ALL est utilisée.
- les requêtes select associées à l'opérateur UNION doivent être compatibles relativement au type de données et ordre des colonnes.
- en utilisant UNION, seule la dernière requête contenant SELECT peut avoir la clause ORDER BY en se référant à la colonne de tri par les numéro de colonne. Ceci est dû au fait que la même colonne peut avoir un nom différent dans chaque requête.

Exemple:
S'il s'agit d'avoir l'ensemble des villes des pièces de la table P et des fournisseurs de la table F on peut écrire:

```
SELECT ville FROM P
UNION
SELECT ville FROM F;
```

3.6.2 L'intersection

Pour obtenir les lignes appartenants à deux requêtes select il suffit d'utiliser la clause INTERSECT donc la syntaxe est:

```
requête SELECT
INTERSECT
requête SELECT
```

Il est nécessaire d'avoir le même nombre de colonnes dans les deux select ainsi qu'une compatibilité entre les types.

3.6.3 La différence

Pour faire la différence entre les résultats de deux requêtes il suffit d'utiliser la clause MINUS dont la syntaxe est:

```
requête SELECT
MINUS
requête SELECT
```

4. Mise à jour des données

4.1 Insertion de données internes

Nous avons vu l'insertion de données quand on doit saisir les occurences ligne par ligne, toutefois on peut avoir le cas ou ces données proviennent d'autres tables, on peut alors compléter la syntaxe de la clause INSERT :

```
INSERT INTO nom_de table [(liste de colonnes)] requête SELECT;
```

Exemple: S'il s'agit d'avoir une table (CLI_PARIS) contenant les identificateurs des fournisseurs parisiens de F on peut écrire:

```
INSERT INTO CLI_PARIS (SELECT f# FROM f where ville = 'Paris');
```

Remarque:
- La requête select ne doit pas contenir de clause order by.
- On ne doit pas utiliser select * avec cet insert car les deux tables n'ont pas forcément le même nombre de colonnes.

4.2 Modification des données par expression

Il arrive que l'on ait besoin de modifier le contenu d'une (ou plusieurs) colonne pour une occurence ou toute une table, cela peut se faire par SQL en utilisant l'instruction UPDATE avec la forme suivante:

```
UPDATE nom_de table
SET nom_colonne= expression,...
[WHERE condition];
```

Exemple: S'il s'agit de multiplier le poids de toutes les pièces de la table P on peut écrire:

```
UPDATE P set poids=poids*100;
```

Remarque:
- L'expression qui suit le signe = peut être une constante, une autre colonne ou une combinaison de colonnes et de constantes.

4.2.2 Modification des données par requête

Il est possible que les données qui servent à la modification proviennent d'une requête select, et la syntaxe devient:

```
UPDATE nom_de table
SET (liste de nom_colonne)= (requête select)
[WHERE condition];
```

4.2.3 Modification des données avec les deux cas

Il est possible que pour la modification de certaines colonnes les données proviennent d'une autre requête select, et pour les autres colonnes les valeurs sont à saisir:

```
UPDATE nom_de table
SET (liste de nom_colonne)= (requête select),
nom_colonne=expression,...
[WHERE condition];
```

Exemple : Si on a besoin de modifier le poids de la pièce p1 par celle de la pièce p4 et d'affecter la ville de Paris à cette même pièce (p1) la requête serait:

```
UPDATE P SET
poids= (select poids from P where p#='p4'),
ville ='Paris'
WHERE p#='p1';
```

La syntaxe complète de la clause UPDATE est:

```
UPDATE nom_de table
SET {(colonne [,colonne]... ) = (requête SELECT)
/ colonne = {expression/ (requête) }}
```

[,(colonne [,colonne]...) = (requête SELECT)

/ colonne = {expression/ (requête) }]...

[**WHERE** condition];

4.3 Suppression de données

Il arrive que l'on ait besoin de supprimer une ou plusieurs occurences d'une table, cela peut se faire par SQL en utilisant l'instruction DELETE avec la forme suivante:

DELETE FROM nom_de table

[**WHERE** condition];

Remarque:
- en l'absence du where toutes les occurences de la tables seront supprimées.
- supprimer toutes les lignes n'est pas équivalent à supprimer la table.La table est uniquement vidée et il est toujours possible d'y insérer des occurences.
- une erreur classique est celle de vouloir supprimer une colonne par cette instruction. Cela se fait par une autre instruction. Cette instruction supprime toute une ligne.

Si nous désirons supprimer tous les fournisseurs de londre de la table F on peut écrire:

DELETE FROM F WHERE ville = 'Londre';

5. commandes de description de données

5.1 Contraintes

Dans le début de ce cours nous avons présenté une forme simple de création d'une table. On a simplement attribué pour chaque colonne un nom et un type, mais en pratique ceci n'est pas suffisant.
En effet on a besoin de spécifier plus de contraintes sur chaque colonne afin d'apporter le maximum de contrôle et de cohérence sur les données des tables.
On peut distinguer deux catégories de contraintes:
- contraintes sur les colonnes
- contraintes sur les tables

5.1.1 Contraintes colonne

Il est possible spécifier des contraintes d'intégrité au niveau des colonnes lors de la phase de création de la table, via l'instruction **CREATE TABLE**.
Si la table est déja créeé il est possible d'utiliser l'instruction **ALTER TABLE**.
Une contrainte d'intégrité est spécifiée juste après le type de la colonne, et il est possible d'en définir plusieurs. La syntaxe générale est:

```
[CONTRAINT contrainte]
{[NOT] NULL
/ {UNIQUE/PRIMARY KEY}
/ REFERENCES nom_table[(colonne)]
[ON DELETE CASCADE]
/ CHECK (condition) }
{[DISABLE]}
```

Le mot **CONSTRAINT** est optionnel et permet d'attribuer un nom à la contrainte à définir. Ainsi si on nomme la contrainte C1 il y aura sauvegarde au niveau du dictionnaire de données d'un objet de type contrainte nommé C1 avec sa définition.
Par défaut Oracle attribut comme nom de contrainte SYS_Cn, où n est un entier.

Le mot **NULL** permet à la colonne d'avoir des valeur nulles. Pour interdire cela on peut spécifier la contraint NOT NULL.
Exemple: Si on désire imposer d'avoir toujours une valeur dans l'attribut pnom de la table pièce P, on peut écrire lors de la définition de cette colonne:

CREATE TABLE P (..., pnom varchar(20) CONSTRAINT C_pnom NOT NULL,...)

La clause **UNIQUE** impose l'unicité des valeurs d'une colonne. Deux occurences ne peuvent pas avoir la même valeur pour cette colonne.
Les valeurs nulles sont autorisées sauf si une contrainte NOT NULL est spécifiée.
L'utilisation de cette clause fait qu'Oracle crée automatiquement un index relatif à cette colonne.

La clause **PRIMARY KEY** indique que la colonne est clef primaire de la table. Cette contrainte est la même que UNIQUE sauf qu'elle interdit les valeurs nulles.
Il y a automatiquement création d'index relatif à cette colonne.

Le mot **REFERENCES** indique que les valeurs de cette colonne doivent être les même que celle dont le nom suit le mot references.
Il s'agit d'une contrainte d'intégrité referentielle par rapport à une clef unique ou primaire.
La clauses **ON DELETE CASCADE** permet de supprimer automatiquement les valeurs d'une clef étrangère dépendant d'une autre clef unique ou primaire si une valeur de cette dernière vient d'être supprimée.

Le mot clef **CHECK** est utilisé pour forcer la vérification de la condition qui le suit à chaque valeur insérée.

Le mot clef **DISABLE** désactive une contrainte car elle sont par défaut activée par Oracle.

5.1.2 Contraintes table
Jusqu'à present nous avons présenté les contraintes spécifiées au niveau d'une colonne, cependant on peut avoir besoin de spécifier une contraine relative à deux ou plusieurs colonnes.
C'est le cas quand on a une clef primaire composée de deux colonnes, on parle alors de contrainte d'intégrite de table:
La syntaxe générale est:

```
[CONTRAINT contrainte]
{[NOT] NULL
```

```
/ { {UNIQUE/PRIMARY KEY}(colonne, [,colonne]...)
/ FOREIGN KEY (colonne, [,colonne]...)
REFERENCESnom_table[(colonne [,colonne]...)]
[ON DELETE CASCADE]
/ CHECK (condition) }
{[DISABLE]}
```

Le mot **FOREIGN KEY** permet de spécifier une ou plusieurs colonnes comme une clef étrangère.

5.2 Modification de structure

La structure d'une base de données est généralement stable, mais il arrive que certaines législations ou nouvelles dispositions impose des changement de structures.
Une nouvelle codification du numéro de telephone peut imposer l'extention de cette zone: de 6 chiffres à 8 chiffres, ajout d'une nouvelle colonne...
Il est possible d'appliquer de telles modification à une table existante et ce via la commande**ALTER** de SQL dont la syntaxe générale est:

```
ALTER TABLE nom de table
[ADD { {colonne/ contrainte_table}
[,{colonne/ contrainte_table}...] }]
]
[MODIFY {colonne}
[,{colonne}]
]
[DROP { PRIMARY KEY
/ UNIQUE (colonne[,colonne])
/  CONSTRAINT nom_contrainte
[CASCADE ]}
]
```

Avec l'instruction **ADD** on peut ajouter une colonne à la structure d'une table déja créée. Pour les lignes déja existantes les valeur de cette colonne seront à NULL. Il est possible d'attribuer la contrainte NOT NULL à cette nouvelle colonne si la table est encore vide.

Avec l'instruction **MODIFY** il est possible de changer la taille d'une colonne, sa valeur par défaut et la contrainte NOT NULL.

L'instruction DROP permet de supprimer une contrainte d'intégrité ou une colonne. L'utilisateur ne peut supprimer une colonne clef primaire ou unique qui fait partie d'une intégrité référentielle.

Suppression de table Il arrive que l'on ait besoin dans la conception d'une application d'avoir une ou plusieurs intermédiaires de travail. Ces tables doivent alors être supprimées à la fin du travail et ce en utilisant l'instruction **DROP TABLE**.

```
DROP TABLE nom_table [CASCADE CONSTRAINTS]
```

A la suite de cette instruction la définition de cette table est supprimée ainsi que tous les objets qui lui sont associés (index, contraintes...).L'espace allouée est aussi libéré et récupéré.

EXERCICES

Exercice 01 :

Soit la base de données relationnelle, PUF, de schéma :
U (NU, NomU, Ville)
P (NP, NomP, Couleur, Poids)
F (NF, NomF, Statut, Ville)
PUF (NP, NU, NF, Quantite)
décrivant le fait que :
- U: une usine est d´ecrite par son numéro NU, son nom NomU, la ville Ville dans laquelle elle est située;
- P: un produit est décrit par son numéro NP, son nom NomP, sa Couleur, son Poids;
- F: un fournisseur est décrit par son numéro NF, son nom NomF, son Statut (four-nisseur sous-traitant, fournisseur-client, ...), la Ville où il est domicilié;
- PUF: le produit de numéro NP a été livré `a l'usine de numéro NU par le fournisseur de numéro NF dans une Quantité donnée.

Exprimer en algèbre relationnelle, en calcul tuples, en calcul domaines, et en SQL les requêtes suivantes:
(1) Donner le numéro, le nom et la ville de toutes les usines.
(2) Donner le numéro, le nom et la ville de toutes les usines de Londres.
(3) Donner les numéros des fournisseurs qui approvisionnent l'usine n±1 en produit n±1.
(4) Donner le nom et la couleur des produits livrés par le fournisseur n±1.
(5) Donner les numéros des fournisseurs qui approvisionnent l'usine n±1 en un produit rouge.
(6) Donner les noms des fournisseurs qui approvisionnent une usine de Londres ou de Paris en un produit rouge.
(7) Donner les numéros des produits livrés à une usine par un fournisseur de la même ville.
(8) Donner les numéros des produits livrés à une usine de Londres par un fournisseur de Londres.
(9) Donner les numéros des usines qui ont au moins un fournisseur qui n'est pas de la même ville.
(10) Donner les numéros des fournisseurs qui approvisionnent à la fois les usines n±1 et n±2.
(11) Donner les numéros des usines qui utilisent au moins un produit disponible chez le fournisseur n±3 (c'est-à-dire un produit qu'il livre mais pas nécessairement à cette usine).
(12) Donner le numéro du produit le plus léger (les numéros si plusieurs produits ont ce même poids).
(13) Donner les numéros des usines qui ne reçoivent aucun produit rouge d'un fournisseur londonien.
(14) Donner les numéros des fournisseurs qui fournissent au moins un produit fourni par au moins un fournisseur qui fournit au moins un produit rouge.
(15) Donner tous les triplets (VilleF, NP, VilleU) tels qu'un fournisseur de la première ville approvisionne une usine de la deuxième ville avec un produit NP.
(16) Même question qu'en 15, mais sans les triplets où les deux villes sont identiques.
(17) Donner les numéros des produits qui sont livrés à toutes les usines de Londres.
(18) Donner les numéros des fournisseurs qui approvisionnent toutes les usines avec un même produit.
(19) Donner les num´eros des usines qui achètent au fournisseur n±4 tous les produits qu'il fournit.
(20) Donner les numéros des usines qui s'approvisionnent uniquement chez le fournisseur n±3.

De plus, exprimer en SQL les requêtes et mises à jour suivantes:
(21) Ajouter un nouveau fournisseur : h 45, Alfred, sous-traitant, Chalon i .
(22) Supprimer tous les produits de couleur noire et de numéro compris entre 100 et 199.
(23) Changer la ville du fournisseur n±1 : il a déménagé pour Nice.

Solution :

1. select * from U;
2. select * from U where ville = 'Londres';
3. select nf from PUF where nu = 1 and np = 1 ;
4a. select distinct nomp, couleur from P, PUF where PUF.np = P.np and nf = 1;
4b. select nomp, couleur from P where np in (select np from PUF where nf = 1) 5a. select distinct nf from PUF, P where couleur = 'Rouge' and PUF.np = P.np and nu = 1;
5b. select distinct nf from PUF where np in (select np from P where couleur = 'Rouge') and nu = 1 ;
6a. select nomf from PUF, P, F, U where couleur = 'rouge' and PUF.np = P.np and PUF.nf = F.nf and PUF.nu = U.nu and (U.ville = 'Londres' or U.ville = 'Paris') ;
6b. select nomf from F where nf in (select nf from PUF where np in (select np from P where couleur = 'Rouge') and nu in (select nu from U where ville = 'Londres or ville = 'Paris')) ;
7. select distinct np from PUF, F, U where PUF.nf = F.nf and PUF.nu = U.nu and U.ville = F.ville;
8a. select distinct np from PUF, F, U where PUF.nf = F.nf and PUF.nu = U.nu and U.ville = F.ville and U.ville = 'Londres ;
8b. select distinct np from PUF where nf in (select nf from F where ville = 'Londres') and nu in (select nu from U where ville = 'Londres') ;
9. select distinct PUF.nu from PUF, F, U where PUF.nf = F.nf and PUF.nu = U.nu and U.ville != F.ville 10a. select distinct first.nf from PUF first, PUF second where first.nf = second.nf and first.nu = 1 and second.nu = 2 ;
10b. select distinct nf from PUF where nf in (select nf from PUF where nu = 1) and nu =2;
11. select distinct nu from PUF where np in (select np from PUF where nf = 3) ;
12a. select np from P where poids in (select min(poids) from P);
12b. select np from P p1 where not exists (select * from P where p1.poids > poids) ;
13. select nu from U where nu not in (select nu from PUF, F, P where PUF.np = P.np and PUF.nf = F.nf and couleur = 'Rouge' and ville = 'Londres') ;
14a. select distinct PUF.nf from PUF, PUF PUF1, PUF PUF2, P where couleur = 'rouge' and P.np = puf2.np and PUF2.nf = PUF1.nf and PUF1.np = PUF.np;
14b. select distinct nf from PUF where np in (select np from PUF where nf in (select nf from PUF where np in (select np from P where couleur = 'rouge'))) ;
15. select distinct F.ville, np, U.ville from PUF, U, F where PUF.nf = F.nf and PUF.nu = U.nu ;
16. select distinct F.ville, np, U.ville from PUF, U, F where F.ville !=U.ville and PUF.nf = F.nf and PUF.nu = U.nu ;
17. select np from PUF where not exists (select nu from U where not exists (select * from PUF where not (ville = 'Londres') or (P.np = PUF.np and U.nu = PUF.nu)));
18b. select nf from PUF where not exists (select nu from U where not exists (select * from PUF PUF1 where F.nf = PUF1.nf and U.nu = PUF1.nu and PUF.np = PUF1.np)));
18b. select nf from F where exists (select np from P where not exists (select nu from U where not exists (select * from PUF where F.nf = PUF.nf and U.nu = PUF.nu and P.np = PUF.np)));
20. select nu from U where nu not in (select nu from PUF where nf != 3);
21. insert into F values (45, Alfred, sous-traitant, Chalon) ;
22. delete P where np>= 100 and np <=199 and couleur = 'noire' ;
23. update F set ville = 'Nice' where nf = 1;

Exercice 02 :

Soit le **MODELE LOGIQUE DE DONNEE** suivant :

ARTICLES (**NOART**, LIBELLE, STOCK, PRIXINVENT)
FOURNISSEURS (**NOFOUR**, NOMFOUR, ADRFOUR, VILLEFOUR)
FOURNIR (**NOFOUR#, NOART#,** PRIXARTICLE, DELAI)

Les règles de gestion :

Un article est fourni par un ou plusieurs fournisseurs
Un fournisseur fourni plusieurs article ou pas d'article
Un article peut avoir plusieurs pris

Question 1 : Etablir le Modèle Conceptuel des Données en y portant les cardinalités (mode graphique).

En utilisant SQL répondre aux questions suivantes

Question 2 : numéros et libellés des articles dont le stock est inférieur à 10 ? **Question 3** : Liste des articles dont le prix d'inventaire est compris entre 100 et 300 ? **Question 4**: Liste des fournisseurs dont le nom commence par "STE" ?
Question 5 : noms et adresses des fournisseurs qui proposent des articles pour lesquels le délai d'approvisionnement est supérieur à 20 jours ?
Question 6 : numéros et libellés des articles triés dans l'ordre décroissant des stocks ?
Question 7 : Liste pour chaque article (numéro et libellé) du prix d'achat maximum, minimum et moyen ?

Exercice 03 :

Un organisme de formation veut réaliser une application de gestion des stagiaires d'un établissement pilote. Une étude est déjà établie et a conduit au schéma relationnel suivant :

STAGIAIRE(NumStg, Nom, Prénom, DateNaissance, #CodeFil)
FILIERE(CodeFil, LibFil, #CodeSec, #CodeNiveau, PlacesPédagogiques)
SECTEUR(CodeSec, LibSec)
NIVEAU(CodeNiv, LibNiv)
MODULE(NumModule, LibModule)
NOTE(#NumStg, #NumModule, Résultat)

Rédiger en code SQL les requêtes qui répondent aux besoins suivants :

1) Afficher la liste des stagiaires inscrits dans la filière "TS en Télécoms" (2pt)
2) Afficher le nombre de stagiaires par filière, trié par ordre décroissant en fonction de nombre de stagiaires. (3pts)
3) Afficher la liste des stagiaires ayant arrivé à une moyenne générale supérieure à 12. (3pts)
4) Afficher la liste des stagiaires ayant réalisé une moyenne générale entre 12 et 15. (Seuls les stagiaires ayant un âge inférieur à 21 feront partie du résultat). (3pts)
5) Augmenter les places pédagogiques de toutes les filières du secteur "Tertiaire" par 10% (3pts)

Exercice 04 :

Soit le modèle relationnel suivant :

EMP(MATR, NOME, POSTE, DATEMB, ID_SUP, SALAIRE, COMMISSION, NUMDEPT)

DEPT(NUMDEPT, NOMDEPT, LIEU)

PROJET(CODEP, NOMP)

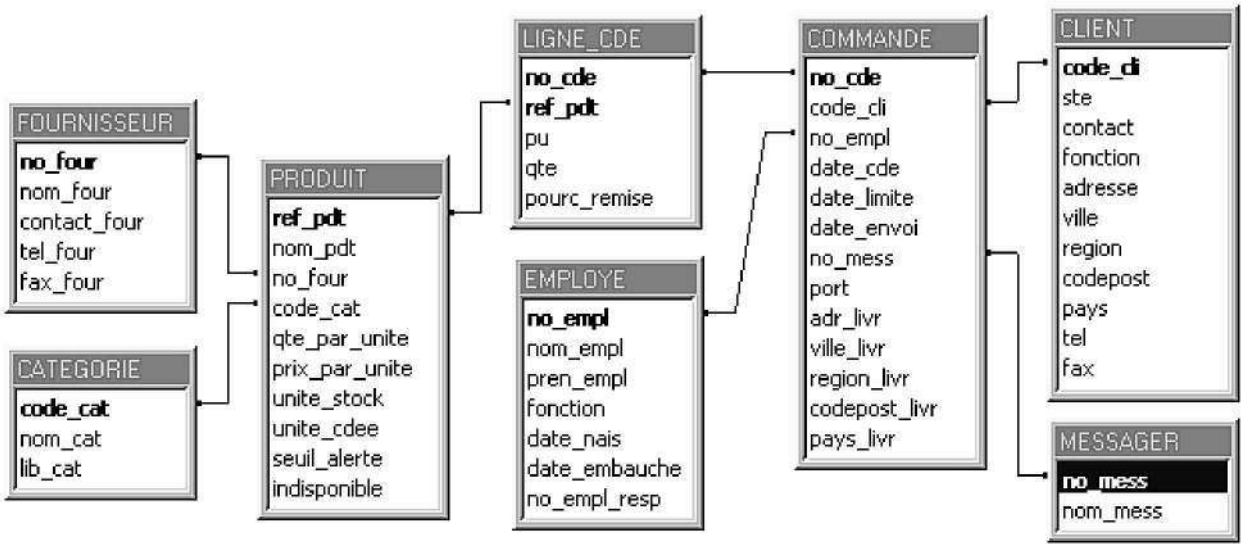
PARTICIPATION(MATR, CODEP, FONCTION)

Ecrire les requête SQL permettant de :

1. Lister les noms des supérieurs directs de chaque employé qui a un supérieur (nom des employés qui ont un supérieur, suivi du nom du supérieur).
2. La liste des employés qui gagnent moins de 50 % du salaire de leur supérieur direct.
3. Lister les noms des employés qui ont le plus gros salaire de leur département.
4. Lister les trois plus gros salaires de chaque département**.**
5. Calculer les totaux des salaires par poste et par département

Exercice 05 :

JoeMeal's Company vous confie sa base de données relationnelle, dont voici le modèle logique sous forme
schématique :



A FAIRE : écrire les requêtes permettant d'obtenir les informations suivantes :

A. Nombre de clients venant de France

B. N° des commandes envoyées après la date limite, triés par date d'envoi décroissante

C. Pour chaque nom d'employé, afficher le nombre de commandes réalisé trié par nombre décroissant
D. Pour chaque n° de commande, afficher le nombre de lignes de la commande, le montant total HT sans remise, mais uniquement les commandes dont le montant total dépasse 5000F