OFPPT

ISTA BM

ROYAUME DU MAROC

مكتب التكوين المهني وإنعاش الشغل

Office de la Formation Professionnelle et de la Promotion du Travail

INSTITUT SUPERIEUR DE GESTION ET D'INFORMATIQUE

Filière : TSTDI

MODULE

Base de données

Année Scolaire : 2009-2010

Introduction

HISTORIQUE

S.Q.L. est un langage structuré permettant d'interroger et de modifier les données contenues dans une base de données relationnelle. S.Q.L. signifie Structured Query Language. Il est issu de SEQUEL : Structured English Query Language.
C'est le premier langage pour les S.G.B.D Relationnels. Il a été développé par IBM en 1970 pour système R, son 1er SGBDR.
S.Q.L. a été reconnu par l'ANSI (Association de Normalisation des Systèmes d'Information) puis imposé comme norme. Il n'existe pas de S.G.B.D.R sans S.Q.L..
Malheureusement, malgrè la norme S.Q.L., il existe un ensemble de dialectes. Les différences entre ces différents dialectes sont souvent minimes et tous respectent un minimum commun : ce que nous allons étudier ici.

DEFINITION

S.Q.L. est un langage relationnel qui permet d'effectuer les tâches suivantes :
Définition et modification de la structure de la base de données
Interrogation et modification non procédurale (c'est à dire itérative) de la base de données
Contrôle de sécurité et d'intégrité de la base

S.Q.L. est un langage interactif, mais il peut aussi être intégré dans un langage de programmation pour le développement d'applications.
S.Q.L. n'est pas le meilleur langage, en particulier pour la manipulation des données, mais c'est un standard.

Dans ce document nous allons prendre comme exemple la syntaxe du SQL SERVER .

Ces exemples sont bâtis sur une base de données composée des trois relations suivantes :

- **emp (num, nom, fonction, n_sup, embauche, salaire, comm, n_dept)**
- **dept(n_dept, nom, lieu)**
- **salgrade(grade, Maxsal , Minsal)**

Description des tables :

EMP : REPRESENTE LA TABLE DES EMPLOYES	
CHAMPS	**DESCRIPTION**
NUM	Numéro De l'employé
NOM	Nom De l'employé
FONCTION	Fonction De l'employé
N_SUP	Matricule du supérieur De l'employé
EMBAUCHE	Date d'embauche De l'employé
SALAIRE	Salaire De l'employé
COMM	Commission De l'employé
N_DEPT	Numéro du département dans lequel il travaille l'employé

DEPT : REPRESENTE LA TABLE DES DEPARTEMENTS	
N_DEPT	Numéro Du département
NOM	Nom Du département
LIEU	Localité Du département

SALGRADE : REPRESENTE LA TABLE DES GRADES

GRADE	Numéro du grade
MINSAL	Salaire minimum du grade
MAXSAL	salaire maximum du grade

LE CONTENU DES TABLES

TABLE EMP

```
SELECT * FROM EMP ;
```

	NUM	NOM	FONCTION	N_SUP	EMBAUCHE	SALAIRE	COMM	N_DEPT
1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800.00	NULL	20
2	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600.00	300.00	30
3	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250.00	500.00	30
4	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975.00	NULL	20
5	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250.00	1400.00	30
6	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850.00	NULL	30
7	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450.00	NULL	10
8	7788	SCOTT	ANALYST	7566	1987-04-19 00:00:00.000	3000.00	NULL	20
9	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000.00	NULL	10
10	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500.00	.00	30
11	7876	ADAMS	CLERK	7788	1987-05-23 00:00:00.000	1100.00	NULL	20
12	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950.00	NULL	30
13	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000.00	NULL	20
14	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300.00	NULL	10

TABLE DEPT

```
SELECT * FROM DEPT ;
```

	N_DEPT	NOM	LIEU
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON

TABLE SALGRADE

```
SELECT * FROM SALGRADE ;
```

	GRADE	MINSAL	MAXSAL
1	1	700.00	1200.00
2	2	1201.00	1400.00
3	3	1401.00	2000.00
4	4	2001.00	3000.00
5	5	3001.00	9999.00

TABLES DES PILOTES

	Nom de la colonne	Type de données	Longueur	Null autorisé
🔑	IDPILOTE	smallint	2	
	NOMPILOTE	char	12	
	DNAISS	datetime	8	
	ADR	char	20	✓
	TEL	char	12	✓
	SAL	numeric	5	
	AGE	smallint	2	✓

TABLE DES AVIONS

	Nom de la colonne	Type de données	Longueur	Null autorisé
🔑	IDAVION	smallint	2	
	TYPEAV	varchar	50	
	CAP	int	4	
	LOC	varchar	20	
	REMARQ	varchar	100	✓

TABLE DES VOLS

	Nom de la colonne	Type de données	Longueur	Null autorisé
🔑	IDVOL	int	4	
	IDPILOTE	int	4	
	IDAVION	int	4	
	VD	varchar	20	
	VA	varchar	20	
	HD	int	4	
	HA	int	4	
	DATVOL	datetime	8	

Contenu des tables :

Table PILOTES :

IDPILOTE	PLNOM	DNAISS	ADR	TEL	SAL	AGE
1	MIRANDA	16-AUG-52	SOPHIA-ANTIPOLIS	93548254	18009.00	52
2	ST-EXUPERY	16-OCT-62	LYON	91548254	12300.00	42
3	ARMSTRONG	11-MARS-60	WAPAKONETA	96548254	24500.00	44
4	TINTIN	01-AUG-50	BRUXELLES	93548254	21100.00	54
5	GAGARINE	12-AUG-55	KLOUCHINO	93548454	22100.00	49
6	BAUDRY	31-AUG-59	TOULOUSE	93548444	21000.00	50
8	BUSH	28-FEB-74	MILTON	44556254	22000.00	30
9	RUSKOI	16-AUG-60	MOSCOU	73548254	22000.30	44
10	MATH	12-AUG-66	PARIS	23548254	15000.09	38
11	YEN	19-SEPT-65	MUNICH	13548254	29000.70	39
12	ICARE	17-DEC-62	ITHAQUES	73548211	17000.68	42
13	MOPOLO	04-NOV-55	NICE	93958211	17000.90	49
14	CHRETIEN	17-OCT-45	NANTE	73223322	15000.10	59
15	VERNES	14-SEP-68	PARIS	38877388	17000.65	36
16	TOURNESOL	04-NOV-49	BRUXELLES	20098833	15000.00	55
17	CONCORDE	15-AUG-66	PARIS	2668833	21000.65	38

Table AVIONS :

IDAV	TYPEAV	CAP	LOC	REMARQ
1	A300	300	Nice	En service
2	A300	300	Nice	En service
3	A320	320	Paris	En service

4	A300	300	Paris	En service
5	CONCORDE	300	Nice	En service
6	B707	400	Paris	En panne
7	CARAVELLE	300	Paris	En service
8	B727	250	Toulouse	En service
9	CONCORDE	350	Toulouse	En service
10	A300	400	Paris	En service

Table VOLS :

IDVOL	IDPILOTE	IDAVION	VD	VA	HD	HA	DAT	PL_LIBRE
100	1	1	NICE	PARIS	1345	1500	03-MARS-89	10
110	3	10	NICE	TOULOUSE	1230	1325	06-MARS-89	8
120	4	3	NICE	PARIS	745	900	21-JUIN-89	0
125	12	10	PARIS	NICE	1330	1845	10-JANV-89	0
130	4	8	TOULOUSE	BEAUVAIS	630	750	27-MARS-89	0
111	5	3	NICE	PARIS	800	920	4-DEC-89	34
135	8	5	PARIS	TOULOUSE	1200	1320	22-MARS-89	12
140	14	9	LYON	NICE	700	750	04-JUIN-89	3
150	1	1	PARIS	NANTES	1630	1725	28-MARS-89	0
153	2	3	LYON	NICE	1210	1300	06-NOV-89	9
156	9	2	PARIS	LYON	230	320	14-JANV-89	0
200	5	3	NICE	TOULOUSE	2030	2125	17-JUIN-89	0
210	14	7	NICE	NANTES	1430	1525	14-OCT-89	5
236	8	4	LYON	TOULOUSE	2130	2250	15-OCT-89	0
240	13	10	NICE	PARIS	2300	2355	19-NOV-89	0
250	13	4	BORDEAUX	PARIS	2300	2355	25-DEC-89	0
260	13	5	BORDEAUX	PARIS	2300	2355	30-NOV-89	1
270	13	9	PARIS	NEW YORK	1400	2300	03-JANV-89	1
280	8	9	NICE	MULHOUSE	1200	1320	21-MARS-89	2
290	3	8	BEAUVAIS	MARSEILLE	1230	1425	09-MARS-89	0

I – L'ALGEBRE RELATIONNELLE

Introduction

"La conception et l'utilisation de **bases de données relationnelles sur micro-ordinateurs** n'est pas un domaine réservé aux informaticiens". C'est en tout cas ce que pensent beaucoup d'utilisateurs en voyant ce type de logiciel intégré aux suites bureautiques les plus connues.

Cependant la maîtrise d'un **SGBDR** micro (Système de Gestion de Bases de Données Relationnelles) est loin d'être aussi facile à acquérir que celle d'un logiciel de traitement de texte ou d'un tableur.

Plusieurs étapes sont nécessaires à la mise en place d'une base de données, dès lors que l'on a précisément défini ses besoins (ce qui n'est déjà pas chose facile !) : **la création de la structure de la base** sous forme de tables (tableaux de données) reliées entre elles par des données clés, **la conception des requêtes** qui permettront d'extraire ou de mettre à jour les informations qu'elle contient, **la conception de l'interface** homme-machine (écrans et états) qui rendra plus conviviale la saisie et la restitution des informations.

Le degré de difficulté dans la conception de l'interface varie beaucoup selon le logiciel utilisé qui est d'ailleurs le plus souvent différent du SGBDR.

La conception de la structure de la base de données, si elle est un peu complexe à appréhender, peut nécessiter, en amont, l'utilisation d'**outils de modélisation conceptuels** de type **entités-associations** (Modèle Conceptuel des Données de la méthode MERISE ou diagramme de classes du langage UML). Mais, même dans les cas les plus simples il faut obligatoirement connaître les concepts du **Modèle Relationnel**, sans quoi un utilisateur non averti pourra toujours arriver à créer une structure inadaptée et sera vite bloqué dans la conception des requêtes.

Il s'agit ici, d'étudier les principaux opérateurs de l'algèbre relationnelle servant de base à l'élaboration des requêtes.

Bon nombre d'utilisateurs qui voient les matériels informatiques et les logiciels changer tous les trois mois, seraient surpris d'apprendre que l'algèbre relationnelle a été définie par Codd en 1970.

Elle est à l'origine du langage **SQL** (Structured Query Language) d'IBM, langage d'interrogation et de manipulation de tous les SGBDR actuels (Oracle, Sybase, Informix, Ingres, DB2, MS Access et tous les autres).

Elle est également mise en œuvre de manière plus conviviale à travers le langage **QBE** (Query By Example) que l'on retrouve seulement sur certains SGBDR (Access par exemple) et qui n'est pas présenté ici.

Une bonne maîtrise de l'algèbre relationnelle permet de concevoir n'importe quelle requête aussi complexe soit elle avant de la mettre en œuvre à l'aide du langage QBE ou SQL.

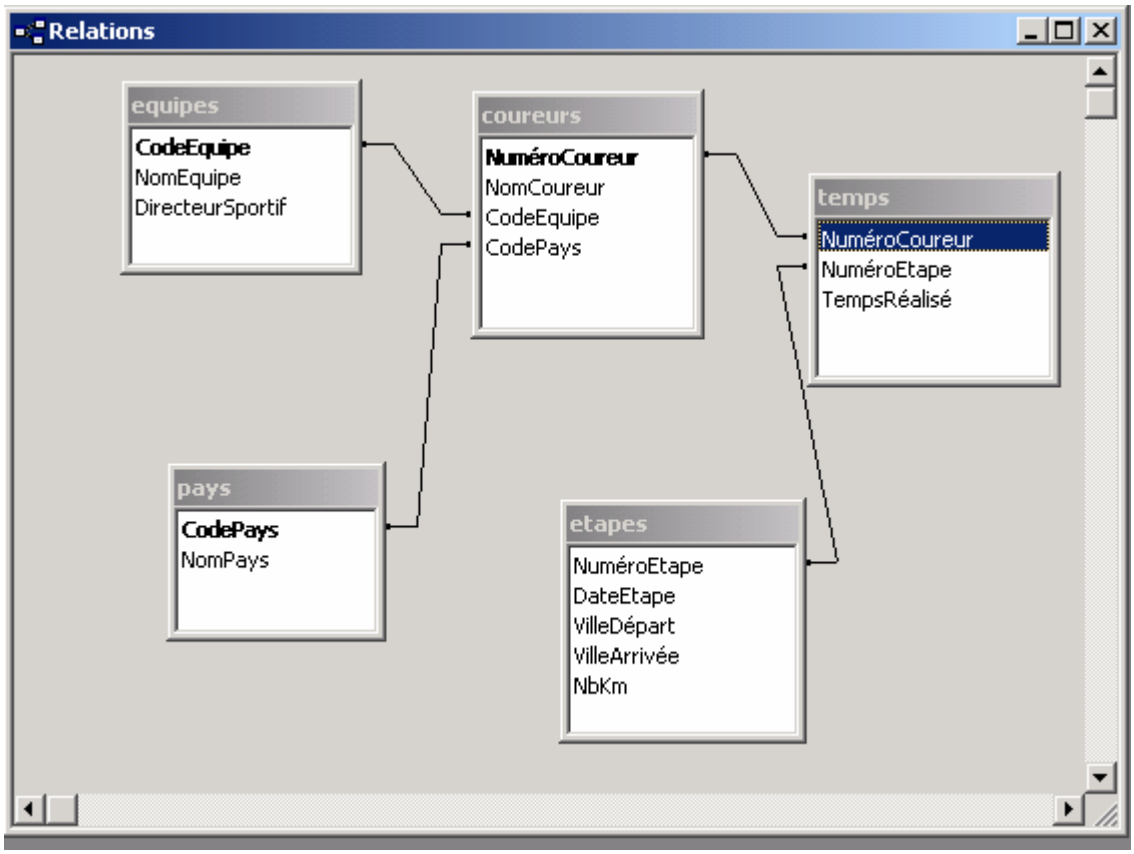
Parmi les opérations de l'algèbre relationnelle, on dispose **d'opérations classiques sur les ensembles** (union, intersection, différence, produit cartésien) puis d'**opérations propres** (projection, sélection, jointure, division).

Sont également exposées ici des **opérations de calcul, de regroupement**, de comptage et de tri, non définies à l'origine par Codd mais très utiles.

Tous les opérateurs sont présentés à l'aide d'exemples clairs. Pris séparément, ils sont faciles à appréhender. La rédaction de requêtes (combinaison d'opérateurs) est illustrée par des exercices concrets.

Le langage SQL n'est abordé que dans le cadre des opérations évoquées ci-dessus. Seule l'instruction SELECT et ses multiples aspects sont donc présentés.

Introduction au Modèle Relationnel



L'exemple suivant, relatif à la gestion simplifiée des étapes du Tour de France, va nous servir à introduire le vocabulaire lié au modèle relationnel.

CodeEquipe	NomEquipe	DirecteurSportif
BAN	BANESTO	Eusebio UNZUE
COF	COFIDIS	Cyrille GUIMARD
CSO	CASINO	Vincent LAVENU
FDJ	LA FRANCAISE DES JEUX	Marc MADIOT
FES	FESTINA	Bruno ROUSSEL
GAN	GAN	Roger LEGEAY
ONC	O.N.C.E.	Manolo SAIZ
TEL	TELEKOM	Walter GODEFROOT
...	...	...

NuméroCoureur	NomCoureur	CodeEquipe	CodePays
8	ULLRICH Jan	TEL	ALL
31	JALABERT Laurent	ONC	FRA
61	ROMINGER Tony	COF	SUI
91	BOARDMAN Chris	GAN	G-B
114	CIPOLLINI Mario	SAE	ITA
151	OLANO Abraham	BAN	ESP

...	...	...	...

NuméroEtape	DateEtape	VilleDépart	VilleArrivée	NbKm
1	06-jul-97	ROUEN	FORGES-LES-EAUX	192
2	07-jul-97	ST-VALERY-EN-CAUX	VIRE	262
3	08-jul-97	VIRE	PLUMELEC	224
...	...	...	...	...

NuméroCoureur	NuméroEtape	TempsRéalisé
8	3	04:54:33
8	1	04:48:21
8	2	06:27:47
31	3	04:54:33
31	1	04:48:37
31	2	06:27:47
61	1	04:48:24
61	2	06:27:47
91	3	04:54:33
91	1	04:48:19
91	2	06:27:47
114	3	04:54:44
114	1	04:48:09
114	2	06:27:47
151	3	04:54:33
151	1	04:48:29
151	2	06:27:47
...	...	...

CodePays	NomPays
ALL	ALLEMAGNE
AUT	AUTRICHE
BEL	BELGIQUE
DAN	DANEMARK

ESP	ESPAGNE
FRA	FRANCE
G-B	GRANDE BRETAGNE
ITA	ITALIE
P-B	PAYS-BAS
RUS	RUSSIE
SUI	SUISSE
…	…

- Comme nous pouvons le constater, le modèle relationnel est un modèle d'organisation des données sous forme de Tables (Tableaux de valeurs) ou chaque **Table** représente une **Relation**, au sens mathématique d'**Ensemble**.

  C'est ainsi que dans l'exemple présenté, figurent l'ensemble des Equipes, des Coureurs, des Etapes, des Temps réalisés par les coureurs à chacune des étapes, et enfin l'ensemble des pays.

- Les colonnes des tables s'appellent des **attributs** et les lignes des **n-uplets** (où n est le degré de la relation, c'est à dire le nombre d'attributs de la relation). Un attribut ne prend qu'une seule valeur pour chaque n-uplet. L'ordre des lignes et des colonnes n'a pas d'importance.

- Chaque table doit avoir une **clé primaire** constituée par un ensemble minimum d'attributs permettant de distinguer chaque n-uplet de la Relation par rapport à tous les autres. Chaque ensemble de valeurs formant la clé primaire d'un n-uplet est donc **unique** au sein d'une table.

  C'est ainsi que dans la table COUREURS, chaque coureur a un NuméroCoureur différent.

  Dans certains cas, plusieurs clés primaires sont possibles pour une seule table. On parle alors de clés candidates. Il faut alors en choisir une comme clé primaire.

- Les liens sémantiques (ou règles de gestion sur les données) existants entre les ensembles sont réalisés par l'intermédiaire de **clés étrangères** faisant elles-mêmes référence à des clés primaires d'autres tables.

  C'est ainsi que dans la table COUREURS, la clé étrangère CodeEquipe (faisant référence à la clé primaire de même nom dans la table EQUIPES) traduit les deux règles de gestion suivantes :

  Un COUREUR appartient à une EQUIPE

  Une EQUIPE est composée de plusieurs COUREURS

- Il existe deux grands types de liens : **Un Plusieurs** (comme le précédent) et **Plusieurs - Plusieurs**. La réalisation de ce dernier type de liens, un peu plus complexe, passe par l'utilisation d'une table intermédiaire dont la clé primaire est formée des clés étrangères des tables qu'elle relie.

  C'est ainsi que la table des TEMPS réalisés à chaque étape par chacun des coureurs exprime les deux règles de gestion suivantes :

  Un COUREUR participe à plusieurs ETAPES

  Une ETAPE fait participer plusieurs COUREURS

- Le modèle relationnel est le plus souvent décrit sous la forme suivante, les clés primaires étant soulignées et les clés étrangères marquées par un signe distinctif (ici *).

  EQUIPES (CodeEquipe, NomEquipe, DirecteurSportif)

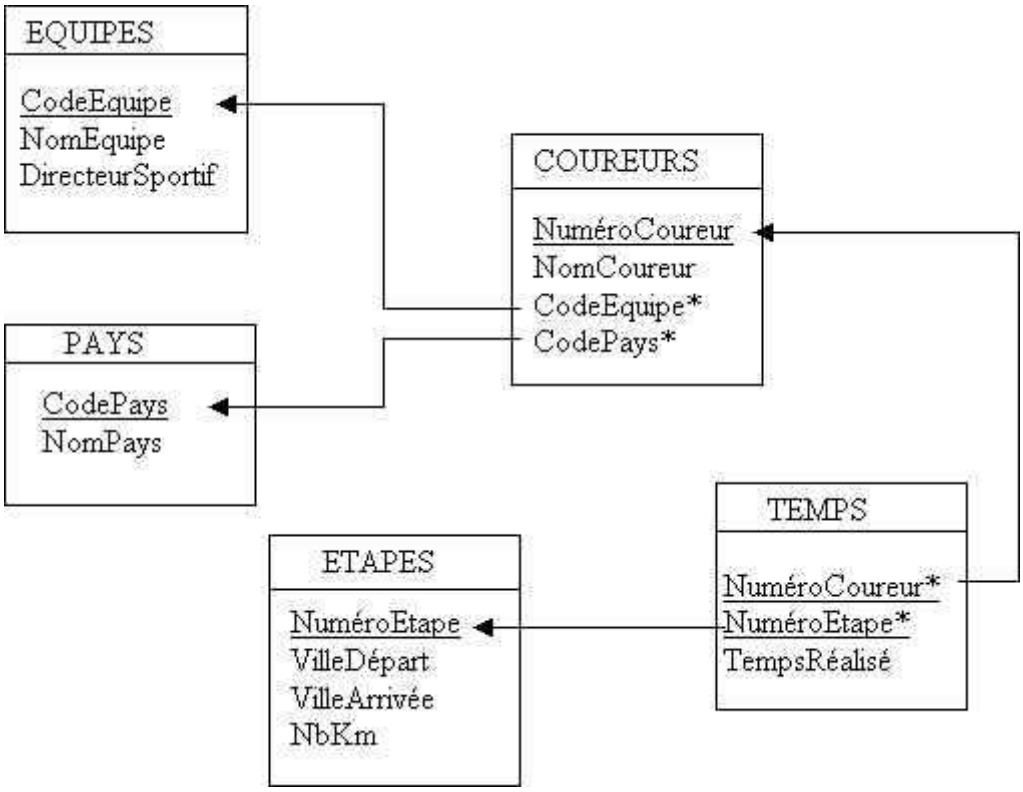
  COUREURS(NuméroCoureur,NomCoureur, CodeEquipe*, CodePays*)

  ETAPES (NuméroEtape, VilleDépart, VilleArrivée, NbKm)

  TEMPS (NuméroCoureur*, NuméroEtape*, TempsRéalisé)

  PAYS (CodePays, NomPays)

- On peut aussi le représenter sous forme graphique, de manière à mieux visualiser et interpréter les liens :



Un COUREUR appartient à une EQUIPE

Une EQUIPE est composée de plusieurs COUREURS

Un COUREUR est originaire d'un PAYS

Un PAYS est représenté par plusieurs COUREURS

Un COUREUR participe à plusieurs ETAPES

Une ETAPE fait participer plusieurs COUREURS

Dans le cadre d'un projet d'informatisation, la conception d'une base de données relationnelle passe d'abord par l'identification des objets de gestion (Coureurs, Etapes, …) et des règles de gestion du domaine modélisé (interviews des utilisateurs, étude des documents manipulés, des fichiers existants, …). Une fois énoncées et validées, ces règles nous conduisent automatiquement à la structure du modèle relationnel correspondant.

OPERATIONS DE L'ALGEBRE RELATIONNELLE

Opération PROJECTION

Formalisme : R = **PROJECTION (**R1, liste des attributs**)** **OU** $\prod$**(**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***.

Opération SELECTION ou de restriction

Formalisme : R = **SELECTION (**R1, condition**)** **ou Restrict(**R1/comdition**)**

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.

Opération JOINTURE (équijointure)

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

$JOIN_Q$(R1,R2) avec Q condition d'égalité entre attributs

Exemple :

PRODUIT

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

DETAIL_COMMANDE

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

515J	LBP 660	1820	97003	515J	3

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

CodePrd	Libellé	Prix unitaire	N°cde	quantité
590A	HD 1,6 Go	1615	97001	2
515J	LBP 660	1820	97002	1
515J	LBP 660	1820	97003	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 (<,>, <=,>=, <>), dans ce cas, on les appelle les non-équijointure ou les ϑ-jointure.

Opération DIVISION

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

Exemple :

PARTICIPER

Athlète	Epreuve
Dupont	200 m
Durand	400 m
Dupont	400 m
Martin	110 m H
Dupont	110 m H
Martin	200 m

EPREUVE

Epreuve
200 m
400 m
110 m H

DIVISION (PARTICIPER, EPREUVE)

Athlète
Dupont

"L'athlète Dupont participe à toutes les épreuves"

- Cet opérateur porte sur 2 relations qui doivent avoir au moins un attribut défini dans le même domaine.
- Tous les attributs du diviseur (ici EPREUVE) doivent être des attributs du dividende (ici PARTICIPER).
- La relation dividende doit avoir au moins une colonne de plus que la relation diviseur.
- La relation résultat, le quotient, possède les attributs non communs aux deux relations initiales et est formée de tous les n-uplets qui, concaténés à chacun des n-uplets du diviseur (ici EPREUVE) donne toujours un n-uplet du dividende (ici PARTICIPER).

Opération UNION

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

Qui représente l'ensemble des tuples de R1 fusionnés avec les tuples de R2

Opération INTERSECTION

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

Qui représente l'ensemble des tuples qui sont a la fois des tuples de R1 des tuples de 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.
En sql

Opération INTERSECTION

SELECT attribut1**,** attribut2**,** ... **FROM** table1

INTERSECT
SELECT attribut1**,** attribut2**,** ... **FROM** table2 **;**

ou

SELECT attribut1, attribut2, ... **FROM** table1

WHERE attribut1 **IN (SELECT** attribut1 **FROM** table2**) ;**

Exemple :

SELECT n°enseignant, NomEnseignant FROM E1

INTERSECT
SELECT n°enseignant, NomEnseignant FROM E2 ;

ou

SELECT n°enseignant, NomEnseignant FROM E1

WHERE n°enseignant IN (SELECT n°enseignant FROM E2) ;

Opération DIFFERENCE

Formalisme : R = **DIFFERENCE (**R1**,** R2**)** **Et en langage SQL**

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)

En sql

Opération DIFFERENCE

SELECT attribut1**,** attribut2**,** ... **FROM** table1
EXCEPT
SELECT attribut1**,** attribut2**,** ... **FROM** table2 **;**

ou

SELECT attribut1**,** attribut2**,** ... **FROM** table1
WHERE attribut1 **NOT IN (SELECT** attribut1 **FROM** table2**) ;**

ou encore (avec la jointure externe du SQL2, si par exemple vous utilisez MySQL qui ne dispose ni du EXCEPT, ni de la possiblité de SELECT imbriqués !) :

SELECT table1.attribut1, table1.attribut2,...
FROM table1 **LEFT JOIN** table2 **ON** table1.attribut1 = table2.attribut1 **WHERE** table2.attribut1 **IS NULL ;**

Exemple :

SELECT n°enseignant, NomEnseignant FROM E1

EXCEPT

SELECT n°enseignant, NomEnseignant FROM E2 ;

ou

SELECT n°enseignant, NomEnseignant FROM E1

WHERE n°enseignant NOT IN (SELECT n°enseignant FROM E2) ;

ou encore

SELECT E1.n°enseignant, E1.NomEnseignant

FROM E1 LEFT JOIN E2 ON E1.n°enseignant = E2.n°enseignant WHERE E2.n°enseignant IS NULL ;

Pour mieux comprendre cette dernière version, voici le résultat renvoyé par la jointure externe gauche entre E1 et E2 :

E1.n°enseignant	E1.NomEnseignant	E2.n°enseignant	E2.NomEnseignant
1	DUPONT	1	DUPONT
3	DURAND	NULL	NULL
4	MARTIN	4	MARTIN
5	BERTRAND	NULL	NULL

Opération PRODUIT CARTESIEN

Formalisme : R = **PRODUIT (**R1**,** R2**)** **Et en langage SQL**

Exemple :

Etudiants

n°étudiant	**nom**
101	DUPONT

Epreuves

libellé épreuve	**coefficient**
Informatique	2

102	MARTIN	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.

En sql

Opération PRODUIT CARTESIEN

SELECT * FROM table1**,** table2 **;**

Exemple :

SELECT * FROM Etudiants, Epreuves ;

Principe d'écriture d'une requête *Et en langage SQL*

La plupart des requêtes (ou interrogations) portant sur une base de données relationnelle ne peuvent pas être réalisées à partir d'une seule opération mais en enchaînant successivement plusieurs opérations.

Exemple

Soient les deux tables (ou relations) suivantes :

CLIENT(CodeClient, NomClient, AdrClient, TélClient)

COMMANDE(N°Commande, Date, CodeClient*)

Remarque : les clés primaires sont soulignées et les clés étrangères sont marquées par *

On désire obtenir le code et le nom des clients ayant commandé le 10/06/97 :

R1=SELECTION(COMMANDE, Date=10/06/97)

R2=JOINTURE(R1, CLIENT, R1.CodeClient=CLIENT.CodeClient)

R3=PROJECTION(R2, CodeClient, NomClient)

En sql

Principe d'écriture d'une requête

Une même instruction SELECT permet de combiner Sélections, Projections, Jointures.

Exemple :

Soient les relations suivantes :

CLIENT(CodeClient, NomClient, AdrClient, TélClient)

COMMANDE(N°Commande, Date, CodeClient*)

Remarque : les clés primaires sont soulignées et les clés étrangères sont marquées par *

On désire obtenir le code et le nom des clients ayant commandé le 10/06/97 :

```
SELECT DISTINCT CLIENT.CodeClient, NomClient
FROM CLIENT, COMMANDE
WHERE CLIENT.CodeClient=COMMANDE.CodeClient AND Date='10/06/97';
```

Remarque importante

Si l'on ne précisait pas CLIENT.CodeClient au niveau du SELECT, la commande SQL ne pourrait pas s'exécuter. En effet il y aurait ambiguïté sur le CodeClient à projeter : celui de la table CLIENT ou celui de la table COMMANDE ?

Opération TRI Et en langage SQL

R = **TRI(**R0, att1À, att2Å, ...**)**

Exemple :

CHAMPIGNONS

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

R1 = TRI (CHAMPIGNONS, EspèceÅ, CatégorieÀ, ConditionnementÀ)

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

- Le tri s'effectue sur un ou plusieurs attributs, dans l'ordre croissant ou décroissant.
- La relation résultat a la même structure et le même contenu que la relation de départ.

En sql

Opération TRI Et en langage SQL

R = **TRI(**R0, att1À, att2Å, ...**)**

Exemple :

CHAMPIGNONS

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

R1 = TRI (CHAMPIGNONS, EspèceÅ, CatégorieÀ, ConditionnementÀ)

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

- Le tri s'effectue sur un ou plusieurs attributs, dans l'ordre croissant ou décroissant.
- La relation résultat a la même structure et le même contenu que la relation de départ.

En sql

Attributs calculés et renommés

Atributs calculés

SELECT N°BonCommande, CodeProduit, Quantité*PuHt

FROM LIGNE_COMMANDE ;

Attributs renommés

SELECT N°BonCommande, CodeProduit, Quantité*PuHt **AS** Montant

FROM LIGNE_COMMANDE ;

Opération CALCULER *Et en langage SQL*

R=**CALCULER(**R0, fonction1, fonction2, ...**)** ou N=**CALCULER(**R0, fonction**)**

Exemple

LIGNE_COMMANDE

N°BonCommande	CodeProduit	Quantité	PuHt
96008	A10	10	83
96008	B20	35	32
96009	A10	20	83
96010	A15	4	110
96010	B20	55	32

On désire obtenir le chiffre d'affaires total Ht, ainsi que le nombre total de produits commandés :
R1=CALCULER(LIGNE_COMMANDE, Somme(Quantité*PuHt), Somme(Quantité))

Somme(Quantité*PuHt)	Somme(Quantité)
5810	124

- Les calculs et/ou comptage portent sur la relation R0.
- La relation résultat ne comportera qu'une ligne avec autant de colonnes que de résultats demandés ou pourra simplement être considérée comme un nombre N utilisable ultérieurement en tant que tel dans le cas où un seul résultat est attendu.

Sql

Opération CALCULER

SELECT fonction1(attribut1), fonction2(attribut2), ...

FROM table **;**

Exemple :

```
SELECT SUM(Quantité*PuHt), SUM(Quantité)
    FROM LIGNE_COMMANDE;
```

Opération REGROUPER_ET_CALCULER Et en langage SQL

R=REGROUPER_ET_CALCULER(R0, att1, att2, ..., fonction1, fonction2, ...**)**

Exemple

LIGNE_COMMANDE

N°BonCommande	CodeProduit	Quantité	PuHt
96008	A10	10	83
96008	B20	35	32
96009	A10	20	83
96010	A15	4	110
96010	B20	55	32

On désire obtenir le montant total Ht de chaque bon de commande : R2=REGROUPER_ET_CALCULER(LIGNE_COMMANDE, N°BonCommande, MontantHt : Somme(Quantité*PuHt))

N°BonCommande	MontantHt
96008	1950
96009	1660
96010	2200

- Le regroupement s'effectue sur un sous ensemble des attributs de la relation R0.
- La relation résultat comportera autant de lignes que de groupes de n-uplets, les fonctions s'appliquant à chacun des groupes séparément.

Sql

Opération REGROUPER_ET_CALCULER

SELECT attribut1, attribut2, ..., fonction1(attribut3), fonction2(attribut4), ...

FROM table

GROUP BY attribut1, attribut2, ... **;**

Exemple :

```
SELECT N°BonCommande, SUM(Quantité*PuHt)
    FROM LIGNE_COMMANDE
    GROUP BY N°BonCommande ;
```

- Tous les attributs placés derrière la clause SELECT doivent être présents derrière la clause GROUP BY. La proposition inverse n'est pas vraie.
- La clause GROUP BY est obligatoire dès lors qu'il y a à la fois des attributs et des fonctions de calcul derrière la clause SELECT.

Il est possible de sélectionner des lignes issues d'un regroupement (grâce à la clause HAVING) et même de les trier.

Exemple : on souhaite, parmi l'ensemble des commandes, ne retenir que celles dont la montant total hors taxes est supérieur à 10000. De plus on souhaite les voir apparaître par ordre décroissant de leurs montants respectifs.

```
SELECT N°BonCommande, SUM(Quantité*PuHt)
FROM LIGNE_COMMANDE
GROUP BY N°BonCommande HAVING SUM(Quantité*PuHt)>10000
 ORDER BY 2 DESC ;
```

- Il n'est pas toujours possible de placer une fonction derrière la clause ORDER BY. Mais les colonnes projetées derrière la clause SELECT étant implicitement numérotées de 1 à n, il est facile d'y faire référence. Ceci explique la présence du chiffre 2 derrière le ORDER BY de l'exemple : il fait référence à SUM(Quantité*PuHt).

Les Fonctions *Et en langage SQL*

Les fonctions sont utilisées dans les opérateurs **CALCULER** et **REGROUPER_ET_CALCULER**.

Les fonctions de calcul

Les fonctions de calculs portent sur un ou plusieurs groupes de n-uplets et évidemment sur un attribut de type numérique.

Somme(attribut) : total des valeurs d'un attribut
Moyenne(attribut) : moyenne des valeurs d'un attribut
Minimum(attribut) : plus petite valeur d'un attribut
Maximum(attribut) : plus grande valeur d'un attribut

La fonction de comptage : Comptage()

La fonction de comptage donne le nombre de n-uplets d'un ou de plusieurs groupes de n-uplets. Il n'est donc pas nécessaire de préciser d'attribut.

En sql

Les Fonctions

Les fonctions sont utilisées avec les opérateurs de calcul et de regroupement.

Les fonctions de calcul

SUM(attribut) : total des valeurs d'un attribut
AVG(attribut) : moyenne des valeurs d'un attribut
MIN(attribut) : plus petite valeur d'un attribut
MAX(attribut) : plus grande valeur d'un attribut

La fonction de comptage

COUNT(*) : nombre de n-uplets

COUNT(DISTINCT attribut) : nombre de valeurs différentes de l'attribut

Exercice d'application n°2

Soit le modèle relationnel suivant relatif à la gestion des notes annuelles d'une promotion d'étudiants :

ETUDIANT(N°Etudiant, Nom, Prénom)
MATIERE(CodeMat, LibelléMat, CoeffMat)
EVALUER(N°Etudiant*, CodeMat*, Date, Note)

Remarque : les clés primaires sont soulignées et les clés étrangères sont marquées par *

Questions :

1 - Quel est le nombre total d'étudiants ?

2 - Quelles sont, parmi l'ensemble des notes, la note la plus haute et la note la plus basse ?

3 - Quelles sont les moyennes de chaque étudiant dans chacune des matières ?

4 - Quelles sont les moyennes par matière ?

5 - Quelle est la moyenne générale de chaque étudiant ?

6 - Quelle est la moyenne générale de la promotion ?

7 - Quels sont les étudiants qui ont une moyenne générale supérieure ou égale à la moyenne générale de la promotion ?

Exercice d'application n°3

Soit le modèle relationnel suivant relatif à la gestion simplifiée des étapes du Tour de France 97, dont une des étapes de type "contre la montre individuel" se déroula à Saint-Etienne :

EQUIPE(CodeEquipe, NomEquipe, DirecteurSportif)
COUREUR(NuméroCoureur, NomCoureur, CodeEquipe*, CodePays*)
PAYS(CodePays, NomPays)
TYPE_ETAPE(CodeType, LibelléType)
ETAPE(NuméroEtape, DateEtape, VilleDép, VilleArr, NbKm, CodeType*)
PARTICIPER(NuméroCoureur*, NuméroEtape*, TempsRéalisé)
ATTRIBUER_BONIFICATION(NuméroEtape*, km, Rang, NbSecondes, NuméroCoureur*)

Remarque : les clés primaires sont soulignées et les clés étrangères sont marquées par *

Questions :

1 - Quelle est la composition de l'équipe Festina (Numéro, nom et pays des coureurs) ?

2 - Quel est le nombre de kilomètres total du Tour de France 97 ?

3 - Quel est le nombre de kilomètres total des étapes de type "Haute Montagne" ?

4 - Quels sont les noms des coureurs qui n'ont pas obtenu de bonifications ?

5 - Quels sont les noms des coureurs qui ont participé à toutes les étapes ?

6 - Quel est le classement général des coureurs (nom, code équipe, code pays et temps des coureurs) à l'issue des 13 premières étapes sachant que les bonifications ont été intégrées dans les temps réalisés à chaque étape ?

7 - Quel est le classement par équipe à l'issue des 13 premières étapes (nom et temps des équipes) ?

Liens vers d'autres cours

Le langage SQL, Yolaine BOURDA, http://wwwlsi.supelec.fr/www/yb/poly_bd/sql/tdm_sql.html

SQL Pro, Frédéric BROUARD, http://sqlpro.multimania.com

BD et SGBD, Yolaine BOURDA, http://wwwlsi.supelec.fr/www/yb/poly_bd/tdm.html

Modélisation des Systèmes d'Information, CORMIER, http://gea-serv.iut.u-picardie.fr/si/introsi.html

Introduction à la Conception des Bases de Données Relationnelles, Didier Boulle, http://www.iut.univ-st-etienne.fr/coursgea/informatique/introbd/index.htm

Bibliographie

MATHIEU P., Des Bases de Données à l'Internet, Vuibert, 2000

BROUARD F., SQL, CampusPress, Collection Référence, 2001

MOINE C., HERZ B., Informatique appliquée à la gestion 1re et 2e année, Foucher, 1996

MATHERON J-P., Comprendre MERISE : outils conceptuels et organisationnels, Eyrolles, 1994

RIGAUX P., Pratique de MySQL et PHP, O'Reilly, 2001

GARDARIN G., Bases de données : les systèmes et leurs langages, Eyrolles, 1994

MAREE C., SQL 2 Initiation Programmation, Armand Colin, 1994

MORLEY C., HUGUES J., LEBLANC B., UML pour l'analyse d'un système d'information, Dunod, 2000

BOUZEGHOUB M., ROCHFELD A., OOM La conception objet des systèmes d'information, Hermès, 2000

Correction de l'exercice d'application n°1 Et en langage SQL

1 - Donner la liste des titres des représentations.
R = PROJECTION(REPRESENTATION, titre_représentation)

2 - Donner la liste des titres des représentations ayant lieu à l'opéra Bastille.
R1 = SELECTION(REPRESENTATION, lieu="Opéra Bastille")
R2 = PROJECTION(R1, titre_représentation)

3 - Donner la liste des noms des musiciens et des titres des représentations auxquelles ils participent.
R1 = JOINTURE(MUSICIEN, REPRESENTATION, Musicien.n°représentation=Représentation.n°représentation)
R2 = PROJECTION(R1, nom, titre_représentation)

4 - Donner la liste des titres des représentations, les lieux et les tarifs pour la journée du 14/09/96.
R1 = SELECTION(PROGRAMMER, date=14/09/96)
R2 = JOINTURE(R1, REPRESENTATION, R1.n°représentation=Représentation.n°représentation)
R3 = PROJECTION(R2, titre_représentation, lieu, tarif)

Correction de l'exercice d'application n°2 Et en langage SQL

1 - Quel est le nombre total d'étudiants ?
N=CALCULER(ETUDIANT, Comptage())

2 - Quelles sont, parmi l'ensemble des notes, la note la plus haute et la note la plus basse ?

R=CALCULER(EVALUER, Minimum(Note), Maximum(Note))

3 - Quelles sont les moyennes de chaque étudiant dans chacune des matières ?
R1=REGROUPER_ET_CALCULER(EVALUER, N°Etudiant, CodeMat, MoyEtuMat : Moyenne(Note))
R2=JOINTURE(R1, MATIERE, MATIERE.CodeMat=R1.CodeMat)
R3=JOINTURE(R2, ETUDIANT, ETUDIANT.N°Etudiant=R2.N°Etudiant)
MOYETUMAT=PROJECTION(R3, N°Etudiant, Nom, Prénom, LibelléMat, CoeffMat, MoyEtuMat)

4 - Quelles sont les moyennes par matière ?
Idem question 3 puis :
R4=REGROUPER_ET_CALCULER(MOYETUMAT, LibelléMat, Moyenne(MoyEtuMat))

5 - Quelle est la moyenne générale de chaque étudiant ?
Idem question 3 puis
MGETU=REGROUPER_ET_CALCULER(MOYETUMAT, N°Etudiant, Nom, Prénom, MgEtu :
Somme(MoyEtuMat*CoeffMat)/Somme(CoeffMat))

6 - Quelle est la moyenne générale de la promotion ?
Idem question 5 puis
MG=CALCULER(MGETU, Moyenne(MgEtu))

7 - Quels sont les étudiants qui ont une moyenne générale supérieure ou égale à la moyenne générale de la promotion ?
idem question 5 et 6 puis :
R=SELECTION(MGETU, MgEtu>=MG)

Correction de l'exercice d'application n°3 ***Et en langage SQL***

1 - Quelle est la composition de l'équipe FESTINA (Numéro, nom et pays des coureurs) ?
R1=SELECTION(EQUIPE, NomEquipe="FESTINA")
R2=JOINTURE(R1, COUREUR, R1.CodeEquipe=COUREUR.CodeEquipe)
R3=JOINTURE(R2, PAYS, R2.CodePays=PAYS.CodePays)
R4=PROJECTION(R3, NuméroCoureur, NomCoureur, NomPays)

2 - Quel est le nombre de kilomètres total du Tour de France 97 ?
N=CALCULER(ETAPE, SOMME(NbKm))

3 - Quel est le nombre de kilomètres total des étapes de type HAUTE MONTAGNE ?
R1=SELECTION(TYPE_ETAPE, LibelléType="HAUTE MONTAGNE")
R2=JOINTURE(R1, ETAPE, R1.CodeType=ETAPE.CodeType)
N=CALCULER(R2, SOMME(NbKm))

4 - Quels sont les noms des coureurs qui n'ont pas obtenu de bonifications ?
R1=PROJECTION(COUREUR, NuméroCoureur)
R2=PROJECTION(ATTRIBUER_BONIFICATION, NuméroCoureur)
R3=DIFFERENCE(R1,R2)
R4=JOINTURE(R3, COUREUR, R3.NuméroCoureur=COUREUR.NuméroCoureur)
R5=PROJECTION(R4, NomCoureur)

5 - Quels sont les noms des coureurs qui ont participé à toutes les étapes ?
R1=PROJECTION(PARTICIPER, NuméroCoureur, NuméroEtape)
R2=PROJECTION(ETAPE, NuméroEtape)
R3=DIVISION(R1, R2)
R4=JOINTURE(R3, COUREUR, R3.NuméroCoureur=COUREUR.NuméroCoureur)
R5=PROJECTION(R4, NomCoureur)

ou

N=CALCULER(ETAPE, Comptage())
R1=REGROUPER_ET_CALCULER(PARTICIPER, NuméroCoureur, Nb:Comptage())
R2=SELECTION(R1, Nb=N)
R3=JOINTURE(R2, COUREUR, R2.NuméroCoureur=COUREUR.NuméroCoureur)
R4=PROJECTION(R3, NomCoureur)

6 - Quel est le classement général des coureurs (nom, code équipe, code pays et temps des coureurs) à

l'issue des 13 premières étapes sachant que les bonifications ont été intégrées dans les temps réalisés à chaque étape ?
R1=SELECTION(PARTICIPER, NuméroEtape<=13)
R2=REGROUPER_ET_CALCULER(R1, NuméroCoureur, Total:Somme(TempsRéalisé))
R3=JOINTURE(R2, COUREUR, R2.NuméroCoureur=COUREUR.NuméroCoureur)
R4=PROJECTION(R3, NomCoureur, CodeEquipe, CodePays, Total)
R5=TRI(R4, Total)

7 - Quel est le classement par équipe à l'issue des 13 premières étapes (nom et temps des équipes) ?
R1=SELECTION(PARTICIPER, NuméroEtape<=13)
R2=JOINTURE(R1, COUREUR, R1.NuméroCoureur=COUREUR.NuméroCoureur)
R3=REGROUPER_ET_CALCULER(R2, CodeEquipe, Total:Somme(TempsRéalisé))
R4=JOINTURE(R3, EQUIPE, R3.CodeEquipe=EQUIPE.CodeEquipe)
R5=TRI(R4, Total)
R6=PROJECTION(R5, Total, NomEquipe)

SQL : Syntaxe simplifiée de l'instruction SELECT

SELECT [***** | **DISTINCT**] att1 [, att2, att3, ...]

FROM Table1 [, Table2, Table3, ...]

[**WHERE** conditions de sélection et/ou de jointure]

[**GROUP BY** att1 [, att2, ...] [**HAVING** conditions de sélection]]

[**ORDER BY** att1 [**ASC** | **DESC**] [, att2 [**ASC** | **DESC**], ...] ;

[ ] : optionnel
| : ou

II - Le Langage De Description De Données(LDD)

Objectif principal	**Définir la structure de la base de données**		
Objectifs intermédiaires	➢ **Créer des tables** ➢ **Décrire les différents types de données utilisables pour les définitions de colonne** ➢ **Modifier la définition des tables** ➢ **Supprimer, renommer et tronquer une table** ➢ **Définir les contraintes** ➢ **Créer des contraintes et les maintenir** ➢ **Décrire une vue** ➢ **Créer une vue** ➢ **Extraire des données par le biais d'une vue** ➢ **Modifier la définition d'une vue** ➢ **Insérer, mettre à jour et supprimer des données par une vue** ➢ **Supprimer une vue** ➢ **Créer et mettre à jour des index** ➢ **Mettre en cluster une table**		
Volume horaire théorique		**Volume horaire pratique**	

Définition:

Le langage de définition des données est le langage permettant de créer ou de modifier le schéma d'une relation et donc d'une table.
Il permet de créer, de modifier et de supprimer non seulement les tables, mais aussi les vues, les index et les contraintes.

1- Les TABLES

1-1 Créer une table :Create table

La table est la structure de base contenant les données des utilisateurs. Quand on crée une table, on peut spécifier les informations suivantes :

- la définition des colonnes,
- les contraintes d'intégrité,
- la tablespace contenant la table,

- les caractéristiques de stockage,
- le cluster contenant la table,
- les données résultant d'une éventuelle requête.

1.1.1-Création simple:

La commande de création de table la plus simple ne comportera que le nom et le type de chaque colonne de la table. L'on peut créer une table par la commande CREATE TABLE en spécifiant le nom et le type de chaque colonne. A la création, la table sera vide mais un certain espace lui sera alloué. La syntaxe est la suivante :

```
CREATE TABLE nom_table (nom_col1  TYPE1  contrainte1,  nom_col2  TYPE2  contrainte2, ….)
```

L'option NOT NULL assure qu'SQL interdit lors d'un INSERT ou d'un UPDATE que cette colonne contienne la valeur NULL, par défaut elle est autorisée.

Exemple :

```
CREATE TABLE T_CLIENT
(CLI_NOM    CHAR(32),
 CLI_PRENOM VARCHAR(32))
```

Une table de clients dotée de deux colonnes avec les nom et prénom des clients.

Exemple :

```
CREATE TABLE T_CLIENT
(CLI_ID     INTEGER      NOT NULL PRIMARY KEY,
 CLI_NOM    CHAR(32)     NOT NULL,
 CLI_PRENOM VARCHAR(32))
```

Une table de clients dotée de trois colonnes avec la clef (numéro du client) les nom et prénom des clients.

1.1.é Les types de données :

Les types de données peuvent être :

On trouve couramment les types suivants dans certains SGBDR :

SMALLDATETIME : un format raccourcis pour des dates dans une plage de valeur restreintes (à éviter)
DATETIME : Ce type de données permet de stocker des données constituées d'une date et d'une heure.
MONEY, SMALLMONEY : un format spécifique aux valeurs monétaires (à éviter)
TINYINT : un entier avec une plage de valeur très restreinte (par exemple 0 à 255)
BIGINT : un entier avec une plage de valeur très étendue
INT : Entier simple
SMALLINT : Entier court
TEXT, NTEXT : Texte avec une grande capacité d'espace
IMAGE : un BLOB spécialisé pour stocker des images
BINARY(n): champs de type binaire de taille n
ROWVERSION, UNIQUEIDENTIFIER: des identifiants spécifiques auto générés
CHAR(longueur) : Ce type de données permet de stocker des chaînes de caractères de longueur fixe. Longueur doit être inférieur à 255, sa valeur par défaut est 1.
VARCHAR(longueur) : Ce type de données permet de stocker des chaînes de caractères de longueur variable. Longueur doit être inférieur à 2000, il n'y a pas de valeur par défaut.

NUMERIC[(longueur,[précision]):Ce type de données permet de stocker des données numériques à la fois entières et réelles. Longueur : précise le nombre maximum de chiffres significatifs stockés Précision : donne le nombre maximum de chiffres après la virgule
FLOAT : Reel simple precision

Ces types ne sont pas normatifs, il convient donc de les éviter pour des raisons de probabilité.

- une table ne peut pas contenir plus d'une 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 ainsi que dans un DISTINCT.

1-2- Contraintes d'intégrité

A la création d'une table, les contraintes d'intégrité se déclarent de la façon suivante :

```
CREATE TABLE nom_table (  nom_col_1   type_1   contrainte_1,
                          nom_col_2   type_2   contrainte_2  , ... ,
                          nom_col_n   type_n   contrainte_n ,
                          CONSTRAINT [nom_contrainte_1] contrainte_1,
                          CONSTRAINT [nom_contrainte_2] contrainte_2, ...
                          CONSTRAINT [nom_contrainte_m] contrainte_m );
```

Ou bien de la façon suivante :

```
CREATE TABLE nom_table (

nom_col_1    type_1   CONSTRAINT [nom_contrainte_1_1]   contrainte_1_1
                      CONSTRAINT [nom_contrainte_1_2]   contrainte_1_2
                       ...
                      CONSTRAINT [nom_contrainte_1_m]   contrainte_1_m,

nom_col_2 type_2      CONSTRAINT [nom_contrainte_2_1]   contrainte_2_1
                      CONSTRAINT [nom_contrainte_2_2]   contrainte_2_2
                       ...
                      CONSTRAINT [nom_contrainte_2_p]   contrainte_2_p,
…

nom_col_n type_n      CONSTRAINT [nom_contrainte_n_1]   contrainte_n_1
                      CONSTRAINT [nom_contrainte_n_2]   contrainte_n_2
                       ...
                      CONSTRAINT [nom_contrainte_n_q]   contrainte_n_q );
```

Les contraintes différentes que l'on peut déclarer sont les suivantes :

NOT NULL : La colonne ne peut pas contenir de valeurs NULL.

UNIQUE : Chaque ligne de la table doit avoir une valeur différente ou NULL pour cette (ou ces) colonne.

PRIMARY KEY: Chaque ligne de la table doit avoir une valeur différente pour cette (ou ces) colonne. les valeurs NULL sont rejetées.

FOREIGN KEY: Cette colonne fait référence à une colonne clé d'une autre table.

CHECK : Permet de spécifier les valeurs acceptables pour une colonne.

DEFAULT : Permet de spécifier les valeurs par défaut pour une colonne.

Exemples de contraintes colonne:

```
Create Table TCLIENT(numcli  int primary key check(numcli <999),....)
```

```
CREATE TABLE TPERSONNE1
(PRS_ID             INTEGER,
 PRS_NOM            VARCHAR(32),
 PRS_PRENOM         VARCHAR(32),
 PRS_SEXE           CHAR(1)     DEFAULT 'M',
 PRS_DATE_NAISSANCE DATETIME    DEFAULT GETDATE())
```

```
CREATE TABLE TPERSONNE2
( PRS_ID             INT     NOT NULL PRIMARY KEY,
  PRS_NOM            VARCHAR(32),
  PRS_PRENOM         VARCHAR(32))
```

```
CREATE TABLE TPERSONNE3
(PRS_ID             INT     CHECK (PRS_ID > 0),
 PRS_NOM            VARCHAR(32) CHECK (DATALENGTH(PRS_NOM) > 2),
 PRS_PRENOM         VARCHAR(32) CHECK (SUBSTRING(PRS_PRENOM, 1, 1)
                                BETWEEN 'A' AND  'Z'),
 PRS_SEXE           CHAR(1)     CHECK (PRS_SEXE IN ('M', 'F')),
 PRS_TELEPHONE      CHAR(14)    CHECK (SUBSTRING(PRS_TELEPHONE, 1, 3) IN ('061','062')))
```

```
CREATE TABLE TPERSONNE4
(PRS_NOM            VARCHAR(32),
 PRS_PRENOM         VARCHAR(32),
 PRS_TELEPHONE      CHAR(14)     UNIQUE)
```

Exemples de contraintes table:

```
CREATE TABLE TPERSONNE5
(PRS_NOM            VARCHAR(32) NOT NULL,
 PRS_PRENOM         VARCHAR(32) NOT NULL,
 PRS_TELEPHONE      CHAR(14),
 CONSTRAINT PK_TPERSONNE5_NOM_PRENOM PRIMARY KEY (PRS_NOM, PRS_PRENOM))
```

```
CREATE TABLE TPERSONNE6
(PRS_ID             INT,
 PRS_NOM            VARCHAR(32),
 PRS_PRENOM         VARCHAR(32),
 CONSTRAINT UK_ TPERSONNE6_NOM_PRENOM UNIQUE (PRS_NOM, PRS_PRENOM))
```

```
CREATE TABLE TFACTURE
(FTC_ID             INT,
 PRS_NOM            VARCHAR(32),
 PRS_PRENOM         VARCHAR(32),
 FCT_DATE           DATETIME,
 FCT_MONTANT        NUMERIC(16,2),
 CONSTRAINT FK_ TFACTURE_NOM_ PRENOM FOREIGN KEY (PRS_NOM, PRS_PRENOM) REFERENCES
 T_PERSONNE5 (PRS_NOM, PRS_PRENOM))
```

La table TFACTURE est liée à la table TPERSONNE5 et ce lien se fait entre la clef étrangère composite PRS_NOM / PRS_PRENOM de la table TFACTURE et la clef de la table TPERSONNE5 elle même composée des colonnes PRS_NOM / PRS_PRENOM.

```
CREATE TABLE TFOURNISSEUR
 (FRN_NOM     CHAR(16) NOT NULL,
  FRN_PRENOM  CHAR(16) NOT NULL,
  CONSTRAINT PK_ TFOURNISSEUR_NOM_PRENOM PRIMARY KEY (FRN_NOM, FRN_PRENOM))
```

```
CREATE TABLE TCOMMANDE
(CMD_ID      INT NOT NULL PRIMARY KEY,
 FRN_NOM     CHAR(16),
 FRN_PRENOM  CHAR(16),
```

```
CONSTRAINT  FK_ TCOMMANDE_NOM_PRENOM
            FOREIGN KEY (FRN_NOM, FRN_PRENOM)
            REFERENCES TFOURNISSEUR (FRN_NOM, FRN_PRENOM)
            ON DELETE CASCADE)
```

1-3-Modifier une table : Alter table

On peut modifier dynamiquement la définition d'une table grâce a la commande ALTER TABLE.
Deux types de modifications sont possibles : ajout d'une colonne et modification d'une colonne existante.
Il n'est pas possible de supprimer une colonne. Par contre une colonne qui n'est plus utilisée peut être mise à la valeur NULL, auquel cas elle n'occupe plus d'espace disque. Si on désire vraiment supprimer une colonne, il faut :

- se créer une nouvelle table sans la colonne en question,
- détruire l'ancienne table,
- donner à la nouvelle table le nom de l'ancienne.

1..3.1- Ajouter une colonne :

La commande suivante permet d'ajouter une ou plusieurs colonnes à une table existante :

```
ALTER TABLE nom_table ADD  ( nom_col1  TYPE1  CONTRAINT1,
                             nom_col2  TYPE2  CONTRAINT2,   ...
                             nom_coln  TYPEn  CONTRAINTn )
```

Les types possibles sont les mêmes que ceux décrits avec la commande CREATE TABLE.
Si la table contient déjà des lignes, la nouvelle colonne aura des valeurs NULL pour les lignes existantes.

Exemple :

```
ALTER table tpersonne1 ADD (prs_age   SMALLINT  DEFAULT 10)
```

1.3.2- Modifier une colonne :

Il est possible de modifier la définition d'une colonne, à condition que la nouvelle définition soit compatible avec le contenu de la colonne et en respectant les contraintes suivantes :

- Dans tous les cas il est possible d'augmenter la taille d'une colonne ;
- Il est possible de diminuer la taille, ou même de changer le type d'une colonne vide ;
- On peut spécifier NOT NULL si la colonne ne contient aucune valeur NULL ;
- On peut dans tous les cas spécifier NULL pour autoriser les valeurs NULL.

Syntaxe :

```
ALTER TABLE nom_table ALTER COLUMN   nom_col1   TYPE1   CONTRAINT1
```

Exemple :

```
ALTER table tpersonne1 ALTER COLUMN  prs_nom   VARCHAR(50)  NOT NULL
```

1.3.3- Ajouter une contrainte :

Vous pouvez ajouter des contraintes dans une table existante en utilisant l'ordre ALTER TABLE avec la clause ADD.
Syntaxe :

```
ALTER TABLE table  ADD [CONSTRAINT constraint] type (column);
```

table	nom de la table
constraint	nom de la contrainte
type	type de contrainte
column	nom de la colonne concernée par la contrainte

Il est recommandé de préciser le nom de la contrainte, même si ce n'est pas obligatoire. A défaut, le système générera lui-même des noms de contraintes.

Conseils
•Vous pouvez ajouter, supprimer, activer ou désactiver une contrainte, mais il est impossible d'en modifier la structure.
•Vous pouvez ajouter une contrainte NOT NULL à une colonne existante en utilisant la clause MODIFY à la place de la clause ADD de l'ordre ALTER TABLE.
Remarque : Vous pouvez définir une colonne NOT NULL seulement si la table est vide, ou bien si vous spécifiez une valeur par défaut, celle-ci sera alors automatiquement attribuée à toutes les lignes existantes de la table.

Exemple :

```
CREATE TABLE CLIENT
( COD_CLI INT NOT NULL,
  CLI_NOM    CHAR(32),
  CLI_PRENOM VARCHAR(32))

ALTER TABLE CLIENT ADD CONSTRAINT pk_T_CLIENT_code_cli PRIMARY KEY(cod_cli)
```

```
CREATE TABLE EMP
( NUM      INT NOT NULL PRIMARY KEY,
  NOM      CHAR(12),
  SUP      INT NOT NULL,
  EMBAUCHE DATETIME NOT NULL, …
  N_DEPT   INT NOT NULL)


ALTER TABLE emp  ADD CONSTRAINT  fk_emp_n_dept
      FOREIGN KEY(N_DEPT) REFERENCES DEPT(n_dept)
```

```
ALTER TABLE emp  ADD CONSTRAINT  Uk_emp_nom
      UNIQUE(nom)
```

1.3.4- Suppression d'une Contrainte

Si vous souhaitez supprimer une contrainte, utilisez l'ordre ALTER TABLE avec la clause DROP. L'option CASCADE de la clause DROP provoque également la suppression de toutes les contraintes associées.

Syntaxe

ALTER TABLE table
DROP PRIMARY KEY | UNIQUE (column) | CONSTRAINT constraint [CASCADE];

où :

table	représente le nom de la table
column	représente le nom de la colonne concernée par la contrainte
constraint	représente le nom de la contrainte

Lorsque vous supprimez une contrainte d'intégrité, elle n'est plus contrôlée par le Serveur et n'existe plus dans le dictionnaire de données.

Exemple :

```
ALTER TABLE  t_client  DROP PRIMARY KEY
```

```
ALTER TABLE emp  DROP CONSTRAINT fk_emp_n_dept
```

```
ALTER TABLE emp  DROP CONSTRAINT  Uk_emp_nom
```

1.4 suppression d'une colonne

Syntaxe de l'ordre de suppression d'une colonne d'une table :

```
ALTER TABLE nom_table
 DROP  COLUMN  nom_colonne
```

Exemple :

```
ALTER TABLE emp  DROP COLUMN embauche
```

1-5- Supprimer une table:'Drop table'

La commande DROP TABLE permet de supprimer une table, sa syntaxe est la suivante :

```
DROP TABLE nom_table ;
```

La table nom_table est alors supprimée. La définition de la table ainsi que son contenu sont détruits, et l'espace occupé par la table est libéré.

2- Les vues

Les vues permettent d'assurer l'objectif d'indépendance logique. Grâce à elles, chaque utilisateur pourra avoir sa vision propre des données.

On a vu que le résultat d'un SELECT est lui-même une table.
Une telle table, qui n'existe pas dans la base mais est créée dynamiquement lors de l'exécution du SELECT, peut être vue comme une table réelle par les utilisateurs. Pour cela, il suffit de cataloguer le SELECT en tant que vue.

Les utilisateurs pourront consulter la base, ou modifier la base (avec certaines restrictions) à travers la vue, c'est-à-dire manipuler la table résultat du SELECT comme si c'était une table réelle.

2.1-Créer une vue : Create view

La commande CREATE VIEW permet de créer une vue en spécifiant le SELECT constituant la définition de la vue :

```
CREATE VIEW nom_vue [(nom_col1,...)AS SELECT ...  WITH CHECK OPTION ;
```

La spécification des noms de colonnes de la vue est facultative. Par défaut, les noms des colonnes de la vue sont les mêmes que les noms des colonnes résultat du SELECT (si certaines colonnes résultat du SELECT sont des expressions, il faut renommer ces colonnes dans le SELECT, ou spécifier les noms de colonne de la vue).

Une fois créée, une vue s'utilise comme une table. Il n'y a pas de duplication des informations mais stockage de la définition de la vue.

Exemple : Création d'une vue constituant une restriction de la table emp aux employés du département 10.

```
CREATE VIEW emp_vu10 AS  SELECT *    FROM emp   WHERE  n_dept = 10 ;
```

Le CHECK OPTION permet de vérifier que la mise à jour ou l'insertion faite à travers la vue ne produisent que des lignes qui font partie de la sélection de la vue.

Exemple :

```
CREATE VIEW emp_vu10 AS  SELECT *
                         FROM emp
                         WHERE  n_dept = 20
                          WITH CHECK OPTION;
```

Ainsi donc, si la vue emp_vu20 a été créée avec CHECK OPTION on ne pourra à travers cette vue ni modifier, ni insérer des employés ne faisant pas partie du département 20.

L'ordre SQL suivant :

```
UPDATE emp_vu20
SET n_dept=10
WHERE salaire BETWEEN 1000 AND 3000;
```

Qui tente de d'affecter tous les employés du département 20 au département 10, échouera car il viole la contrainte WITH CHECK OPTION créée par le biais de la vue.

2.2 Mise à jour "à travers une vue":

Il est possible d'effectuer des modifications des données par INSERT , UPDATE et DELETE à travers une vue, en tenant compte des restrictions suivantes :

- le SELECT définissant la vue ne doit pas comporter de jointure : (la vue doit etre construite sur une seule table)
- les colonnes résultat du SELECT doivent être des colonnes réelles et non pas des expressions
- l'ordre SELECT utilisé pour définir la vue ne doit comporter ni jointure ni de clause GROUP BY
- la vue contient toutes les colonnes ayant l'option NOT NULL de la table de base

Exemple : Modification des salaires du département 10 à travers la vue emp10.

```
UPDATE emp10  SET sal = sal *1.1;
```

Tous les employés du département 10 auront une augmentation salariale de 10%.

2.3- Supprimer une vue: Drop view

Une vue peut être détruite par la commande :

```
DROP  VIEW nom_vue;
```

3- Les index

3.1- Introduction

Selon le modèle relationnel les sélections peuvent être faites en utilisant le contenu de n'importe quelle colonne et les lignes sont stockées dans n'importe quel ordre.
Considérons le SELECT suivant :

```
SELECT * FROM emp WHERE nom = 'FORD'
```

Un moyen de retrouver la ou les lignes pour lesquelles le nom est égal à FORD est de balayer toute la table.

Un tel moyen d'accès conduit à des temps de réponse prohibitifs pour des tables dépassant quelques centaines de lignes.

Une solution offerte par tous les systèmes de gestion de bases de données est la création d'index, qui permettra de satisfaire aux requêtes les plus fréquentes avec des temps de réponse acceptables.

Un index sera matérialisé par la création de blocs disque contenant des couples (valeurs d'index, numéro de bloc) donnant le numéro de bloc disque dans lequel se trouvent les lignes correspondant à chaque valeur d'index.

3.2- Structure d'un index :

Les index sont des structures permettant de retrouver une ligne dans une table à partir de la valeur d'une colonne ou d'un ensemble de colonnes. Un index contient la liste triée des valeurs des colonnes indexées avec les adresses des lignes (numéro de bloc dans la partition et numéro de ligne dans le bloc) correspondantes.

Tous les index sont stockés sous forme d'arbres équilibrés : une structure arborescente permet de retrouver rapidement dans l'index la valeur de clé cherchée, et donc l'adresse de la ligne correspondante dans la table.

Dans un tel arbre, toutes les feuilles sont à la même profondeur, et donc la recherche prend approximativement le même temps quelle que soit la valeur de la clé.

Lorsqu'un bloc d'index est plein, il est éclaté en deux blocs. en conséquence, tous les blocs d'index ont un taux de remplissage variant de 50% à 100%. Sans index on balaie séquentiellement toute la table quelle que soit la position de élément recherché.

3.3- Utilisation des index :

L'adjonction d'un index à une table ralentit les mises à jour (insertion, suppression, modification de la clé) mais **accélère beaucoup la recherche** d'une ligne dans la table.

L'index accélère la recherche d'une ligne à partir d'une valeur donnée de clé, mais aussi la recherche des lignes ayant une valeur d'index supérieure ou inférieure à une valeur donnée, car les valeurs de clés sont triées dans l'index.

Exemple : Les requêtes suivantes bénéficieront d'un index sur le champ num.

```
SELECT * FROM emp WHERE num = 16034  ;
SELECT * FROM emp WHERE num >= 27234 ;
SELECT * FROM emp WHERE num BETWEEN 16034  AND 27234 ;
```

Un index est utilisable même si le critère de recherche est constitué seulement du début de la clé.

Exemple : La requête suivante bénéficiera d'un index sur la colonne nom.

```
SELECT *
FROM emp
WHERE nom LIKE 'M%' ;
```

Par contre si le début de la clé n'est pas connu, l'index est inutilisable.

Exemple : La requête suivante ne bénéficiera pas d'un index sur le champ nom.

```
SELECT *
FROM emp
WHERE ename LIKE '?????????' ;
```

Valeurs NULL
Elles ne sont pas représentées dans l'index, ceci afin de minimiser le volume nécessaire pour stocker l'index. En contrepartie, l'index ne sera d'aucune utilité pour retrouver les valeurs NULL lorsque le critère de recherche est du type IS NULL.

3.4-Conversions

L'index n'est utilisable que si le critère de sélection est le contenu de la colonne indexée, sans aucune transformation. Par exemple un index sur salaire ne sera pas utilisé pour la requête suivante :

```
SELECT * FROM emp
WHERE salaire * 12 > 300000 ;
```

Attention en particulier aux conversions de type qui peuvent empêcher l'utilisation de l'index.

SQL est un langage typé, chaque type de données (numérique, caractère, date) ayant ses propres opérateurs, ses propres fonctions et sa propre relation d'ordre. En conséquence, si dans une expression, figurent à la fois un nombre et une chaîne de caractères, SQL convertira la chaîne de caractères en nombre. De même si dans une expression, figurent à la fois une chaîne de caractères et une date, SQL convertira la chaîne de caractères en date.

Or, dans un prédicat du type :

WHERE fonction(col_indexée) = constante

SQL ne peut pas utiliser l'index.

Ceci peut se produire , de façon insidieuse, lorsque SQL est obligé d'ajouter un appel à une fonction de conversion à cause d'une discordance de type.

Exemple : Le prédicat suivant ne bénéficiera pas d'un index sur le champ embauche.

```
SELECT * FROM emp
WHERE embauche LIKE '????' ;
```

En effet, SQL est obligé d'effectuer une conversion, et le prédicat qui sera évalué est :

```
WHERE convert(varchar(12),embauche)) LIKE '????'
```

Le critère de recherche est une fonction de embauche, et non le champ embauche lui-même, dans ce cas l'index est inutilisable.

3.5-Choix des index

Indexer en priorité :

1.les clés primaires
2.les colonnes servant de critère de jointure
3.les colonnes servant souvent de critère de recherche

Ne pas indexer :

1.les colonnes contenant peu de valeurs distinctes (index alors peu efficace)
2.les colonnes fréquemment modifiées

3.6-Créer un index

Un index peut être créé par la commande suivante :

```
CREATE [UNIQUE]  INDEX nom_index ON nom_table (nom_col1 , nom_col2, ...) ;
```

dans laquelle :

- L'option UNIQUE indique que l'on interdit que deux lignes aient la même valeur dans la colonne indexée.

Un index peut être créé dynamiquement sur une table contenant déjà des lignes. Il sera ensuite tenu à jour automatiquement lors des modifications de la table.

Un index peut porter sur plusieurs colonnes, la clé d'accès sera alors la concaténation des différentes colonnes.
On peut créer plusieurs index indépendants sur une même table.

Les requêtes SQL sont transparentes au fait qu'il existe un index ou non. C'est l'optimiseur du système de gestion de bases de données qui, au moment de l'exécution de chaque requête, recherche s'il peut s'aider ou non d'un index.

Un index concaténé est un index portant sur plusieurs colonnes.

Exemple :

```
CREATE UNIQUE  INDEX   IDX_emp_fonction
ON   (  fonction  ) ;
```

Exemple :

```
CREATE   INDEX   IDX_emp_ n_dept_num
ON   (  n_dept  ,  num  ) ;
```

Les index concaténés peuvent être utilisés pour matérialiser une clé composée de plusieurs colonnes.

SQL sait utiliser un index concaténé même si le critère de recherche ne porte pas sur toutes les colonnes présentes dans l'index.

Exemple : L'index ci-dessus est utilisable si l'on ne connaît que le numéro de département.

```
SELECT nom FROM emp WHERE n_dept = 20 ;
```

3-7- Supprimer un index

Un index peut être supprimé dynamiquement par la commande :

```
DROP  INDEX nom_table.nom_index;
```

L'espace libéré reste attaché au segment d'index de la table : il pourra être utilisé pour un autre index sur la même table.

L'espace ne sera rendu à la partition que lors de la suppression de la table.

5. Exercices:

Série 7

7.1 Créer une relation FORMATION, qui contiendra les renseignements suivants :
- *le numéro de pilote ,*
- *le type de formation (ATT, VDN, PAI, ...)*
- *type d'appareil*
- *date de la formation*

Attention :
- un pilote à une date donnée participe à une formation
- un type d'appareil doit être : 'A300', 'A310', 'A320', 'B707', 'Caravelle', 'B727' ou 'Concorde'

7.2 Créer la clé primaire (sans utiliser la clause PRIMARY KEY) sur le numéro du pilote et la date de formation.

7.3 Créer un index unique sur la colonne PLNOM de PILOTE. Que constatez vous.

7.4 Créer également un index sur la colonne AVTYPE de la table FORMATION.

Série 8

8.1 Ajouter la colonne AGE à la table PILOTE. Un pilote doit avoir entre 25 et 60 ans.

8.2 Ajouter une contrainte d'intégrité de référence au niveau table à la relation FORMATION (colonne PILOTE)

8.3 Modifier la colonne PL# de la table PILOTE en number(5).

8.4 Ajouter une valeur par défaut à la colonne VD dans VOL.

8.5 Associer à l'attribut SALAIRE d'un pilote un commentaire puis s'assurer de son existence. Comment supprime - t-on un commentaire ?

8.6 Consulter la liste des colonnes de la table FORMATION

8.7 Attribuer un synonyme "Conducteurs" à la table PILOTE.

Série 9

9.1 Indépendance logique/externe : *vue de sélection*
- *"Créer une vue AVA300 qui donne tous les A300 dans la compagnie"*
- *"Que se passe - t-il à l'insertion d'un "B707" dans la vue ?"*

9.2 Indépendance logique/externe : *renommage et ré-ordonnancement des colonnes*
- "Créer une vue PAYE qui donne pour chaque pilote son salaire mensuel et annuel"
- "Créer une vue AVPLUS qui donne tous les numéros d'avions conduits par plus d'un pilote."
- "Créer une vue PILPARIS qui donne les noms, les numéros de vols, des pilotes qui assurent au moins un vol au départ de Paris"

9.3 Création d'une vue pour assurer la confidentialité
"Créer une vue PILSANS qui donne les renseignements concernant les pilotes, sans le salaire."

9.4 Création d'une vue pour assurer l'intégrité : *contraintes structurelles*

Intégrité de domaine

"Créer une vue qui assurer les contraintes de domaine suivantes dans la table AVION :
- AVTYPE {A300, A320, Concorde, B707, Caravelle }
- AV# entre 200 et 500

"Créer une vue PIL25 qui vérifie que chaque pilote inséré a plus de 25 ans."

Intégrité de référence

"Créer les tables PILOTE6, AVION6 et VOL6 (sans les clauses REFERENCES et FOREIGN KEY) à partir de PILOTE, AVION, VOL. Créer ensuite une vue VOLSURS vérifiant l'intégrité de référence en insertion dans VOL6. La contrainte à vérifier est : pour tout nouveau vol, le pilote et l'avion doivent exister.
Test :

```
insert into volsurs values(150,1,20,'NICE', 'Paris',1345,1500,'3-MAR-89' );
insert into volsurs values(100,1,1,'NICE', 'Nantes',1345,1500,'4-MAR-89' );
```

"Créer une vue PILOTSUP sur PILOTE6 et VOL6 dans laquelle on accède à tous les pilotes qui ne conduisent aucun vol. La contrainte à vérifier est qu'un Pilote ne peut - être supprimé que s'il ne conduit aucun Avion."
Test :

```
delete from pilotsup where pl# = 1;
delete from pilotsup where pl# = 6;
```

9.5 Vues issues d'une table

"Créer une vue AVIONNICE : Ensemble des avions localisés à Nice"

Modification à travers une vue

1) Lister l'extension de la vue AVIONNICE
2) Mise à jour d'un tuple dans cette vue : localiser l'avion de n° 5 à Paris
3) Mise à jour d'un tuple dans cette vue : localiser l'avion n° 7 à Paris
4) Lister la table de base AVION. Que constatez-vous ?

Insertion dans la vue

1) Insérer le tuple (11, 'A300', 220, 'Nice', 'EN service');
2) lister l'extension de la vue AVIONNICE
3) lister la table de base AVION.

Suppression dans la vue

1) Suppression de l'avion N° 11
2) lister l'extension de la vue AVIONNICE
3) lister la table AVION.

9.6 Vues issues de plusieurs tables

"Créer une vue AVSERVPARIS : Ensemble des avions en service localisés à Paris"

Modification de la vue

1) lister l'extension de la vue AVSERVPARIS
2) mise à jour d'un tuple de cette vue. Que remarque-t-on ?"

Insertion dans la vue

1) recréez la vue avec jointure
2) insertion d'un tuple dans la vue AVSERVPARIS. Que remarque-t-on?

suppression dans la vue

1) suppression de tous les pilotes de n° inférieur à 7 dans AVSERVPARIS

9.7 Vues contenant des colonnes virtuelles

"Reprendre la vue PAYE et lister son contenu"

Modification via la vue

1) Mise à jour d'un tuple dans cette vue : mettre le salaire du pilote 1 à 0
2) lister le contenu de cette vue. Que remarque--on ?

Insertion via la vue

1) insertion d'un tuple dans la vue PAYE . Que remarque-t-on ?

Suppression via la vue

1) suppression de tous les pilotes dont le salaire annuel est supérieur à 180000.

9.8 Vues contenant une clause GROUP BY

"Reprenons la vue AVPLUS. Llister cette vue"

"Quels sont le n° d'avions conduits par plus d'un pilote et localisés à Paris ?

II - L'Ordre Select

Objectif principal	**Extraire les données de la base en utilisant l'ordre Select**		
Objectifs intermédiaires	➢ **Enumérer toutes les possibilités de l'ordre SQL SELECT** ➢ **Trier les lignes retournées par une requête** ➢ **Utiliser les différentes types de fonctions SQL** ➢ **Ecrire des ordres SELECT pour accéder aux données de plusieurs tables en utilisant des équijointures et des non-équijointures** ➢ **Regrouper les données avec la clause GROUP BY** ➢ **Inclure ou exclure des groupes de lignes avec la clause HAVING** ➢ **Décrire les opérateurs ensemblistes** ➢ **utiliser les sous-interrogations mono-ligne et multi-ligne** ➢ **Ecrire une sous-interrogation multi-colonne**		
Volume horaire théorique		**Volume horaire pratique**	

1-introduction:

Ce chapitre expose la partie du langage **sql** permettant d'extraire des informations stockées dans une base de données. Il s'agit d'un langage déclaratif dont la syntaxe est très simple (comme beaucoup de langages de ce type) ce qui permet de se concentrer sur le problème à résoudre.
Les exemples cités dans ce cours ont tous été testés sous SQL SERVER , un des systèmes de gestion de bases de données relationnels les plus répandus sur le marché.

La commande SELECT constitue, à elle seule, le langage permettant d'interroger une base de données. Elle permet :

- De sélectionner certaines colonnes d'une table : c'est l'opération de projection ;
- De sélectionner certaines lignes d'une table en fonction de leur contenu : c'est l'opération de restriction ;
- De combiner des informations venant de plusieurs tables : ce sont les opérations de jointure, union, i intersection, différence relationnelle ;
- De combiner entre elles ces différentes opérations.

Une interrogation, on parle plutôt de requête, est une combinaison d'opérations portant sur des tables (relations) et dont le résultat est lui-même une table dont l'existence est éphémère (le temps de la requête).

On peut introduire un commentaire à l'intérieur d'une commande **sql** en l'encadrant par /* */ .

2-Interroger simplement une base

Sélection de colonnes ou projection

La commande SELECT la plus simple a la syntaxe suivante :

```
SELECT *  FROM  nom_table ;
```

Dans laquelle :

- nom_table : est le nom de la table sur laquelle porte la sélection.
- * : signifie que toutes les colonnes de la table sont sélectionnées.

Par défaut toutes les lignes sont sélectionnées. On peut limiter la sélection à certaines colonnes, en indiquant une liste de noms de colonnes à la place de l'astérisque.

```
SELECT nom_col1, nom_col2, ...  FROM  nom_table ;
```

2-1-Syntaxe du verbe SELECT

Cette commande permet de récupérer des données contenues dans une ou plusieurs tables ou vues.

```
SELECT [DISTINCT | ALL]
  { *
      | { [schema.]{table | view }.* | expr [c_alias]
      [, { [schema.]{table | view }.* | expr [c_alias] } ] ... }
FROM [schema.]{table | view } [t_alias]
        [, [schema.]{table | view } [t_alias] ] ...
[WHERE condition ]
[GROUP BY expr [, expr] ... [HAVING condition] ]
[{UNION | UNION ALL | INTERSECT | MINUS} SELECT command ]
[ORDER BY {expr|position} [ASC | DESC] [, {expr | position} [ASC | DESC]] ...]
```

DISTINCT : renvoie toutes les lignes sélectionnées en enlevant les doublons.

ALL : renvoie toutes les lignes sélectionnées sans enlever les doublons. C'est la valeur par défaut.

* : renvoie toutes les colonnes de toutes les tables ou vues précisés dans le FROM.

table.*, view.* : sélectionne toutes les colonnes de la table, ou de la vue .

expr : sélectionne une expression habituellement calculée sur les valeurs des colonnes appartenant à l'une des tables ou vues, de la clause FROM.

c_alias : la chaîne de caractères qui sert d'en-tête à la colonne (par défaut expr)

schema : est le nom du schéma contenant les tables ou sélectionnés. Le schéma par défaut est celui de l'utilisateur qui exécute la requête.

table, view : est le nom de la table ou de la vue contenant les données sélectionnées.

t_alias : synonyme pour la table dont le nom précède, à utiliser dans le reste de la requête.

WHERE : restreint les lignes sélectionnées à celles pour lesquelles la condition est vraie. Si cette clause est omise, toutes les lignes des tables ou des vues précisées derrière le FROM sont renvoyées.

GROUP BY : groupe les lignes sélectionnées en se basant sur la valeur de expr pour chaque ligne et renvoie une seule ligne par groupe.

HAVING : restreint les groupes de lignes renvoyés à ceux pour lesquels la condition spécifiée est vraie. Sans cette clause, tous les groupes sont renvoyés.

UNION, UNION ALL, INTERSECT, MINUS :Combine les lignes retournées par deux SELECT en utilisant une opération ensembliste.

ORDER BY : ordonne les lignes sélectionnées

Expr : en utilisant la valeur de expr. Cette expression est basée sur des colonnes précisées derrière le SELECT ou sur des colonnes appartenant à des tables ou vues présentes derrière le FROM.

Position : donne le numéro de la colonne dans l'ordre du SELECT.

ASC, DESC : mode ascendant ou descendant. La valeur par défaut ASC.

Prérequis

Pour pouvoir sélectionner des lignes d'un objet(table, vue) il faut soit être propriétaire de cet objet, soit avoir le privilège SELECT sur cet objet.

Le privilège SELECT ANY TABLE permet de sélectionner des lignes de n'importe quel objet appartenant à n'importe quel utilisateur.

Exemple : Donner le nom et la fonction de chaque employé.

```
SELECT  nom, fonction  from emp
```

	nom	fonction
1	SMITH	CLERK
2	ALLEN	SALESMAN
3	WARD	SALESMAN
4	JONES	MANAGER
5	MARTIN	SALESMAN
6	BLAKE	MANAGER
7	CLARK	MANAGER
8	SCOTT	ANALYST
9	KING	PRESIDENT
10	TURNER	SALESMAN
11	ADAMS	CLERK
12	JAMES	CLERK
13	FORD	ANALYST
14	MILLER	CLERK

La clause **DISTINCT** ajoutée derrière la commande SELECT permet d'éliminer les duplications.

Exemple : Quelles sont toutes les fonctions différentes.

```
SELECT DISTINCT fonction from emp
```

	fonction
1	ANALYST
2	CLERK
3	MANAGER
4	PRESIDENT
5	SALESMAN

2-2-Sélection de lignes ou restriction

La clause WHERE permet de spécifier quelles sont les lignes à sélectionner. Elle est suivie d'un prédicat qui sera évalué pour chaque ligne de la table. Les lignes pour lesquelles le prédicat est vrai seront sélectionnées.

La syntaxe est la suivante :

```
SELECT *  FROM  nom_table    WHERE predicat ;
```

Un prédicat n'est ni plus ni moins que la façon dont on exprime une propriété. Les prédicats, qu'ils soient simples ou composés, sont constitués à partir d'expressions que l'on compare entre elles.

- Expression simple
  Une expression simple peut être :
  - une variable désignée par un nom de colonne,
  - une constante.

Les expressions peuvent être de trois types : numérique, chaîne de caractères ou date. A chacun de ces types correspond un format de constante :

- Constante numérique

nombre contenant éventuellement un signe, un point décimal et une puissance de dix. Ex : -10, 2.5, 1.2 E-10

- Constante chaîne de caractères

une chaîne de caractères entre apostrophes. Ex :'MARTIN' (Attention, une lettre en majuscules n'est pas considérée comme égale à la même lettre en minuscule).

- Constante date

une chaîne de caractères entre apostrophes au format suivant : jour-mois-année où le jour est sur deux chiffres, le mois est désigné par les trois premières lettres de son nom en anglais, l'année est sur deux chiffres. Ex : '01-FEB-85'

On peut, en SQL, exprimer des expressions plus complexes en utilisant des opérateurs et des fonctions étudiés dans le chapitre Expressions et fonctions.

2.3 Prédicat simple:

Un prédicat simple est le résultat de la comparaison de deux expressions au moyen d'un opérateur de comparaison qui peut être :

= égal, != différent, < inférieur, > supérieur, >= supérieur ou égal

Les trois types d'expressions peuvent être comparés au moyen de ces opérateurs :

- Pour les types date, la relation d'ordre est l'ordre chronologique.
- Pour les types caractère, la relation d'ordre est l'ordre alphabétique.

Il faut ajouter à ces opérateurs arithmétiques classiques les opérateurs suivants :

expr1 **BETWEEN** expr2 AND expr3
vrai si expr1 est compris entre expr2 et expr3, bornes incluses

expr1 **IN** (expr2, expr3, ...)
vrai si expr1 est égale à l'une des expressions de la liste entre parenthèses

expr **LIKE** chaine
où chaine est une chaîne de caractères pouvant contenir l'un des caractères jokers :

_ remplace exactement 1 caractère

% remplace une chaîne de caractères de longueur quelconque, y compris de longueur nulle.

Exemple : Quels sont les employés dont la commission est supérieure au salaire?

```
SELECT nom, salaire, comm FROM emp WHERE  comm > salaire;
```

	nom	salaire	comm
1	MARTIN	1250.00	1400.00

Exemple : Quels sont les employés gagnant entre 2000 et 5000?

```
SELECT nom, salaire, comm FROM emp  WHERE SALAIRE BETWEEN 2500 AND 5000;
```

	nom	salaire	comm
1	JONES	2975.00	NULL
2	BLAKE	2850.00	NULL
3	SCOTT	3000.00	NULL
4	KING	5000.00	NULL
5	FORD	3000.00	NULL

Exemple : Quels sont les employés commerciaux ou ingénieurs?

```
SELECT num, nom, fonction, salaire FROM  emp WHERE  fonction IN ('MANAGER','CLERK');
```

	num	nom	fonction	salaire
1	7369	SMITH	CLERK	800.00
2	7566	JONES	MANAGER	2975.00
3	7698	BLAKE	MANAGER	2850.00
4	7782	CLARK	MANAGER	2450.00
5	7876	ADAMS	CLERK	1100.00
6	7900	JAMES	CLERK	950.00
7	7934	MILLER	CLERK	1300.00

Exemple : Quels sont les employés dont le nom commence par M?

```
SELECT nom FROM emp WHERE nom LIKE 'M%';
```

	nom
1	MARTIN
2	MILLER

2.4 Prédicats composés

Les opérateurs logiques **AND** (et) et **OR** (ou inclusif) peuvent être utilisés pour combiner entre eux plusieurs prédicats. L'opérateur NOT placé devant un prédicat en inverse le sens.

L'opérateur AND est prioritaire par rapport à l'opérateur OR. Des parenthèses peuvent être utilisées pour imposer une priorité dans l'évaluation du prédicat, ou simplement pour rendre plus claire l'expression logique.

Exemple : Quels sont les employés du département 30 ayant un salaire supérieur à 25000?
SELECT nom FROM emp WHERE n_dept = 30 AND salaire > 25000;

Exemple : Quels sont les employés directeurs, ou commerciaux et travaillant dans le département 10?

```
SELECT nom, fonction, salaire, n_dept  FROM emp
WHERE fonction = 'directeur' OR (fonction = 'commercial' AND n_dept = 10);
```

Exemple : Quels sont les employés directeurs ou commerciaux, et travaillant dans le département 10?

```
SELECT num, nom, fonction, n_dept FROM emp
WHERE (fonction='MANAGER' OR fonction = 'PRESIDENT') AND n_dept = 10;
```

	num	nom	fonction	n_dept
1	7782	CLARK	MANAGER	10
2	7839	KING	PRESIDENT	10

2.5 Valeurs NULL

Pour sql, une valeur NULL est une valeur non définie. Il est possible d'ajouter une ligne à une table sans spécifier de valeur pour les colonnes non obligatoires : ces colonnes absentes auront la valeur NULL.

Par exemple les employés dont la rémunération ne prend pas en compte de commission auront une valeur NULL, c'est-à-dire indéfinie, comme commission.

L'opérateur IS NULL permet de tester la valeur NULL : le prédicat expr IS NULL est vrai si l'expression a la valeur NULL (c'est-à-dire si elle est indéfinie).

Exemple : Quels sont les employés dont la commission a la valeur NULL?

```
SELECT nom, salaire, comm FROM  emp WHERE comm  IS NULL;
```

	nom	SALAIRE	COMM
1	SMITH	800.00	NULL
2	JONES	2975.00	NULL
3	BLAKE	2850.00	NULL
4	CLARK	2450.00	NULL
5	SCOTT	3000.00	NULL
6	KING	5000.00	NULL
7	ADAMS	1100.00	NULL
8	JAMES	950.00	NULL
9	FORD	3000.00	NULL
10	MILLER	1300.00	NULL

L'opérateur IS NOT NULL permet de construire un prédicat vrai si la valeur n'est pas NULL (et donc le prédicat expr IS NOT NULL est vrai si expr est définie)

Remarques

- La valeur NULL est différente de la valeur zéro qui, elle, qui est une valeur bien définie.
- Le prédicat expr = NULL est toujours faux, et ne permet donc pas de tester si l'expression a la valeur NULL.
- Une expression de la forme NULL + val donne NULL comme résultat quelle que puisse être la valeur de val.

Nom de colonne

Les colonnes constituant le résultat d'un SELECT peuvent être renommées dans le SELECT, ceci est utile en particulier lorsque la colonne résultat est une expression. Pour cela, il suffit de faire suivre l'expression définissant la colonne d'un nom, selon les règles suivantes :

- le nom (30 caractères maximum) est inséré derrière l'expression définissant la colonne, séparé de cette dernière par un espace.

- si le nom contient des séparateurs (espace, caractère spécial), ou s'il est identique à un mot clé de SQL
(ex : DATE), il doit être mis entre guillemets "".

Ce nom est celui sous lequel la colonne sera connue des interfaces externes. Sous l'analyseur de requêtes de SQL SERVER, par exemple, il constituera le titre par défaut de la colonne, et servira de référence pour définir un format pour la colonne.

Exemple :
Salaire de chaque employé.

```
SELECT nom, salaire "SALAIRE MENSUEL"
```

```
FROM emp;
```

	nom	SALAIRE MENSUEL
1	SMITH	800.00
2	ALLEN	1600.00
3	WARD	1250.00
4	JONES	2975.00
5	MARTIN	1250.00
6	BLAKE	2850.00
7	CLARK	2450.00
8	SCOTT	3000.00
9	KING	5000.00
10	TURNER	1500.00
11	ADAMS	1100.00
12	JAMES	950.00
13	FORD	3000.00
14	MILLER	1300.00

Remarque : Attention, ce nom n'est pas connu à l'intérieur du SELECT.

3- Classer le résultat d'une interrogation

Les lignes constituant le résultat d'un SELECT sont obtenues dans un ordre indéterminé. On peut, dans un SELECT, demander que le résultat soit classé dans un ordre ascendant ou descendant, en fonction du contenu d'une ou plusieurs colonnes (jusqu'à 16 critères de classement possibles). Les critères de classement sont spécifiés dans une clause ORDER BY dont la syntaxe est la suivante :

```
ORDER BY {nom_col1 | num_col1 [DESC] [, nom_col2 | num_col2 [DESC],...]}
```

Le classement se fait d'abord selon la première colonne spécifiée dans l'ORDER BY puis les lignes ayant la même valeur dans la première colonne sont classées selon la deuxième colonne de l'ORDER BY, etc... Pour chaque colonne, le classement peut être ascendant (par défaut) ou descendant (DESC).

L'ORDER BY peut faire référence à une colonne par son nom ou par sa position dans la liste des colonnes présentes derrière le SELECT (la première colonne sélectionnée a le numéro 1, la deuxième a le numéro 2, ...).

Exemple : Donner tous les employés classés par fonction, et pour chaque fonction classés par salaire décroissant

```
SELECT nom, fonction, salaire FROM emp ORDER BY fonction, salaire DESC;
```

Ceci est équivalent à :

```
SELECT nom, fonction, salaire FROM emp ORDER BY 2,3 DESC;
```

Remarque : Dans un classement les valeurs NULL sont toujours en tête quel que soit l'ordre du classement (ascendant ou descendant).

4- Requêtes avancées

4-1- Les jointures

La jointure est une opération permettant de combiner des informations venant de plusieurs tables. Les exemples suivants se limiteront à deux tables, mais on peut joindre jusqu'à 256 tables. Une jointure se formule simplement en spécifiant plusieurs tables derrière le FROM de la façon suivante :

```
SELECT ...
FROM  table gauche, table droite
WHERE <condition de jointure>;
```

Autre syntaxe :

```
SELECT ...
FROM   <table gauche>
      [INNER]JOIN <table droite>
      ON <condition de jointure>
```

Si on ne précise pas de condition de sélection, le résultat obtenu sera **le produit cartésien** des tables présentes derrière le FROM (résultat non souhaité en général).
Exemple :

```
Select * from emp,dept
```

Il n'existe pas d'associations implicites ou explicites entre les tables dans SQL. Les associations entre les tables sont définies dynamiquement lors des interrogations, ce qui contribue à la grande souplesse du langage sql et rend possible toute association même si elle n'a pas été prévue lors de la définition et du chargement de la base.

4-1-1 Jointure naturelle (Equi-jointure)

Le rapprochement de chaque ligne de la table emp avec la ligne de la table dept ayant même numéro de département permet d'obtenir la liste des employés avec la localité dans laquelle ils travaillent. Ce rapprochement entre deux colonnes appartenant à deux tables différentes mais ayant le même sens (ici le numéro de département) et venant vraisemblablement d'une relation 1-n lors de la conception (ici 1 entité département pour n entités employés) est assez naturel. C'est pourquoi ce type de jointure porte le nom de jointure naturelle ou d'équi-jointure.

Exemple : Donner pour chaque employé son nom et son lieu de travail.

```
SELECT emp.nom, lieu                ou      SELECT  emp.nom, lieu
FROM emp, dept                              FROM    emp inner join dept
Where  emp.n_dept=dept.n_dept               on  emp.n_dept=dept.n_dept
```

	num	nom	nom	lieu
1	7369	SMITH	RESEARCH	DALLAS
2	7499	ALLEN	SALES	CHICAGO
3	7521	WARD	SALES	CHICAGO
4	7566	JONES	RESEARCH	DALLAS
5	7654	MARTIN	SALES	CHICAGO
6	7698	BLAKE	SALES	CHICAGO
7	7782	CLARK	ACCOUNTING	NEW YORK
8	7788	SCOTT	RESEARCH	DALLAS
9	7839	KING	ACCOUNTING	NEW YORK
10	7844	TURNER	SALES	CHICAGO
11	7876	ADAMS	RESEARCH	DALLAS
12	7900	JAMES	SALES	CHICAGO
13	7902	FORD	RESEARCH	DALLAS
14	7934	MILLER	ACCOUNTING	NEW YORK

Le fait que la colonne contenant le numéro de département ait le même nom dans les deux tables a rendu nécessaire le préfixage par le nom de table dans le critère de jointure (clause WHERE). Le nom de colonne nom a lui aussi besoin d'être préfixé car il appartient aux deux tables (nom de la personne dans l'une et nom du département dans l'autre). Par contre le nom de colonne lieu n'a pas besoin d'être préfixé car il n'y a pas d'ambiguïté sur la table à laquelle cette colonne appartient.

4-1-2-Autojointure : Jointure d'une table à elle-même

Il peut être utile de rassembler des informations venant d'une ligne d'une table avec des informations venant d'une autre ligne de la même table.

Exemple : Donner pour chaque employé le nom de son supérieur hiérarchique.

```
SELECT emp. nom, mgr.nom FROM emp, emp mgr WHERE emp.n_sup= mgr.num;
```

	nom	nom
1	SMITH	FORD
2	ALLEN	BLAKE
3	WARD	BLAKE
4	JONES	KING
5	MARTIN	BLAKE
6	BLAKE	KING
7	CLARK	KING
8	SCOTT	JONES
9	TURNER	BLAKE
10	ADAMS	SCOTT
11	JAMES	BLAKE
12	FORD	JONES
13	MILLER	CLARK

Remarque : Dans ce cas, il faut impérativement renommer au moins l'une des deux occurrences de la table (ici emp) en lui donnant un synonyme, afin de pouvoir préfixer sans ambiguïté chaque nom de colonne.

4-1-3-Non-equijointures

Le critère d'égalité est le critère de jointure le plus naturel. Mais on peut utiliser d'autres types de comparaisons comme critères de jointures.
Exemple : Quels sont les employés gagnant plus que SIMON?

```
SELECT emp.nom, emp.salaire, emp.fonction FROM emp, emp j
WHERE emp.salaire > j.salaire AND J.nom = 'SIMON';
```

	nom	salaire	fonction
1	JONES	2975.00	MANAGER
2	BLAKE	2850.00	MANAGER
3	SCOTT	3000.00	ANALYST
4	KING	5000.00	PRESIDENT
5	FORD	3000.00	ANALYST

Exemple : Afficher pour tous les employés du département 20 leurs Numéros, leurs noms, leurs salaires et leurs grades.

```
SELECT num,nom,salaire,grade
```

```
FROM emp,salgrade
WHERE salaire BETWEEN minsal AND maxsal
AND n_dept = 20
```

	num	nom	salaire	grade
1	7369	SMITH	800.00	1
2	7566	JONES	2975.00	4
3	7788	SCOTT	3000.00	4
4	7876	ADAMS	1100.00	1
5	7902	FORD	3000.00	4

4-1-4-Jointure externe

Lorsqu'une ligne d'une table figurant dans une jointure n'a pas de correspondant dans les autres tables, elle ne satisfait pas au critère d'équi-jointure et donc ne figure pas dans le résultat de la jointure.
Une option permet de faire figurer dans le résultat les lignes satisfaisant la condition d'équi-jointure plus celles n'ayant pas de correspondant. Cette option s'obtient en utilisant l'opérateur left/right join entre les deux tables de jointure pour spécifier la jointure externe est ce qu'elle est gauche ou droite

Exemple : Le département 40 ne figurait pas dans le résultat du SELECT précédent. Par contre, il figurera dans le résultat du SELECT suivant.

```
SELECT emp.nom,dept.n_dept, lieu
 FROM emp right join dept
 on  emp.n_dept=dept.n_dept
```

	nom	n_dept	lieu
1	CLARK	10	NEW YORK
2	KING	10	NEW YORK
3	MILLER	10	NEW YORK
4	SMITH	20	DALLAS
5	JONES	20	DALLAS
6	SCOTT	20	DALLAS
7	ADAMS	20	DALLAS
8	FORD	20	DALLAS
9	ALLEN	30	CHICAGO
10	WARD	30	CHICAGO
11	MARTIN	30	CHICAGO
12	BLAKE	30	CHICAGO
13	TURNER	30	CHICAGO
14	JAMES	30	CHICAGO
15	NULL	40	BOSTON

L'opérateur right join peut s'interpréter comme l'ajout d'une ligne fictive dont toutes les colonnes ont la valeur NULL, et qui réalise la correspondance avec les lignes de l'autre table qui n'ont pas de correspondant réel. Dans l'exemple ci-dessus, la valeur de nom associée au département 40 est la valeur NULL.

Exemple : Retrouver les départements n'ayant aucun employé.

```
SELECT emp.nom,dept.n_dept, lieu
 FROM dept left join emp
 on  dept.n_dept=emp.n_dept
where  emp.nom is null
```

	nom	n_dept	lieu
1	NULL	40	BOSTON

4-2 Les sous interrogations

Une caractéristique puissante de SQL est la possibilité qu'un critère de recherche employé dans une clause WHERE (expression à droite d'un opérateur de comparaison) soit lui-même le résultat d'un SELECT ; c'est ce qu'on appelle une sous-interrogation.

4-2-1 Sous interrogation ramenant une seule valeur

Exemple : Quels sont les employés ayant la même fonction que SCOTT?

```
SELECT nom,fonction FROM emp
WHERE fonction = (SELECT fonction FROM emp WHERE nom ='SCOTT');
```

	nom	FONCTION
1	SCOTT	ANALYST
2	FORD	ANALYST

Remarques

- une sous-interrogation qui ne ramène aucune ligne se termine avec un code d'erreur.
- une sous-interrogation ramenant plusieurs lignes provoquera aussi, dans ce cas, une erreur (pour traiter correctement ce cas, voir paragraphe ci-dessous)

4-2-2 Sous interrogation ramenant plusieurs lignes:

Une sous-interrogation peut ramener plusieurs lignes à condition que l'opérateur de comparaison admette à sa droite un ensemble de valeurs.
Les opérateurs permettant de comparer une valeur à un ensemble de valeurs sont:

- l'opérateur IN
- les opérateurs obtenus en ajoutant ANY ou ALL à la suite d'un opérateur de comparaison classique (=, !=, >, >=, <, <=)
- ANY: la comparaison est vraie si elle est vraie pour au moins un des éléments de l'ensemble.
- ALL: la comparaison sera vraie si elle est vraie pour tous les éléments de l'ensemble.

Exemple : Quels sont les employés gagnant plus que tous les employés du département 30.

```
SELECT nom, salaire
FROM emp WHERE salaire > ALL ( SELECT salaire
                               FROM emp WHERE n_dept = 20);
```

	nom	salaire
1	KING	5000.00

Il est possible de comparer le résultat d'un SELECT ramenant plusieurs colonnes à une liste de colonnes. La liste de colonnes figurera entre parenthèses à gauche de l'opérateur de comparaison.

Exemple : Quels sont les employés ayant même fonction et même supérieur que CODD?

```
SELECT nom, fonction, n_sup
FROM emp
WHERE fonction =  (SELECT fonction  FROM emp  WHERE nom = 'SCOTT')
```

```
And      n_sup    =  (SELECT n_sup      FROM emp  WHERE nom = 'SCOTT');
```

	nom	fonction	n_sup
1	SCOTT	ANALYST	7566
2	FORD	ANALYST	7566

4-2-3-Sous-interrogation synchronisée avec l'interrogation principale:

Dans les exemples précédents, la sous-interrogation était évaluée d'abord, puis le résultat pouvait être utilisé pour exécuter l'interrogation principale. SQL sait également traiter une sous-interrogation faisant référence à une colonne de la table de l'interrogation principale. Le traitement dans ce cas est plus complexe car il faut évaluer la sous interrogation pour chaque ligne de l'interrogation principale.

Exemple : Quels sont les employés ne travaillant pas dans le même département que leur supérieur hiérarchique.

```
SELECT nom FROM emp e
WHERE n_dept != (SELECT n_dept FROM emp WHERE e.n_sup = num)
AND n_sup IS NOT NULL;
```

Il a fallu ici renommer la table emp de l'interrogation principale pour pouvoir la référencer dans la sous-interrogation.

	nom
1	JONES
2	BLAKE

4-2-5-Sous-interrogation ramenant au moins une ligne

L'opérateur EXISTS permet de construire un prédicat vrai si la sous-interrogation qui suit ramène au moins une ligne.

Exemple : Quels sont les employés travaillant dans un département qui a procédé à des embauches depuis le début de l'année 94.

```
SELECT * FROM emp e
 WHERE EXISTS ( SELECT *  FROM emp
                    WHERE embauche >= '01-01-1984' AND n_dept = e.n_dept);
```

	NUM	NOM	FONCTION	N_SUP	EMBAUCHE	SALAIRE	COMM	N_DEPT
1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800.00	NULL	20
2	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975.00	NULL	20
3	7788	SCOTT	ANALYST	7566	1987-04-19 00:00:00.000	3000.00	NULL	20
4	7876	ADAMS	CLERK	7788	1987-05-23 00:00:00.000	1100.00	NULL	20
5	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000.00	NULL	20

Remarque : On peut inverser le sens de l'opérateur EXISTS en le faisant précéder de NOT.

4-2-6-Sous-interrogations multiples

Un SELECT peut comporter plusieurs sous-interrogations, soit imbriquées, soit au même niveau dans différents prédicats combinés par des AND ou des OR.

Exemple : Liste des employés du département 10 ayant même fonction que quelqu'un du département de CLARCK.

```
SELECT nom, fonction
FROM emp WHERE n_dept = 10
```

```
AND fonction IN (SELECT fonction FROM emp WHERE n_dept = ( SELECT n_dept FROM emp
                                                           WHERE nom = 'CLARCK'));
```

	nom	fonction
1	CLARK	MANAGER
2	MILLER	CLERK

5 - Les opérateurs ensemblistes:

Les opérateurs ensemblistes permettent de "joindre" des tables verticalement c'est-à-dire de combiner dans un résultat unique des lignes provenant de deux interrogations. Les lignes peuvent venir de tables différentes mais après projection on doit obtenir des tables ayant même schéma de relation.

Les opérateurs ensemblistes sont les suivants :

- l'union : UNION
- l'intersection : INTERSECT
- la différence relationnelle : MINUS

La syntaxe d'utilisation est la même pour ces trois opérateurs :

```
SELECT ...
{UNION | INTERSECT | MINUS }
SELECT ...
.
.
{UNION | INTERSECT | MINUS }
SELECT ...
```

Dans une requête utilisant des opérateurs ensemblistes :

- Tous les SELECT doivent avoir **le même nombre de colonnes sélectionnées, et leur types doivent être un à un identiques.** Les conversions éventuelles doivent être faites à l'intérieur du SELECT à l'aide des fonctions de conversion.
- Les doubles sont éliminés (DISTINCT implicite).
- Les noms de colonnes (titres) sont ceux du premier SELECT.
- la largeur des colonnes est la plus grande parmi tous les SELECT.
- Dans une requête on ne peut trouver qu'un seul ORDER BY. S'il est présent, il doit être mis dans le dernier

SELECT et il ne peut faire référence qu'aux numéros des colonnes et non pas à leurs noms (car les noms peuvent être différents dans chacune des interrogations).

On peut combiner le résultat de plus de deux SELECT au moyen des opérateurs UNION, INTERSECT, MINUS.

```
SELECT ...
UNION
SELECT ...
MINUS SELECT ...
```

Exemple :

```
Select nom, fonction ,n_dept
From emp
Where n_dept=30
UNION
Select nom, fonction ,n_dept
From emp
Where n_dept=10;
```

	nom	fonction	n_...
1	ALLEN	SALESMAN	30
2	BLAKE	MANAGER	30
3	JAMES	CLERK	30
4	MARTIN	SALESMAN	30
5	TURNER	SALESMAN	30
6	WARD	SALESMAN	30
7	CLARK	MANAGER	10
8	KING	PRESIDENT	10
9	MILLER	CLERK	10

Dans ce cas l'expression est évaluée de gauche à droite, mais on peut modifier l'ordre d'évaluation en utilisant des parenthèses.

```
SELECT ...
UNION (  SELECT ...
         MINUS
         SELECT ...
      )
```

Les opérateurs INTERSECT et MINUS ne sont pas supportés par SQL SERVER

6 - Expressions et Fonctions

6-1- Expressions et Fonctions simples

Une expression est un ensemble de variables (contenu d'une colonne), de constantes et de fonctions combinées au moyen d'opérateurs. Les fonctions prennent une valeur dépendant de leurs arguments qui peuvent être eux-mêmes des expressions.

Les expressions peuvent figurer :

- en tant que colonne résultat d'un SELECT,
- dans une clause WHERE,
- dans une clause ORDER BY.

Il existe trois types d'expressions correspondant chacun à un type de données de SQL : arithmétique, chaîne de caractère, date. A chaque type correspondent des opérateurs et des fonctions spécifiques.

SQL autorise les mélanges de types dans les expressions et effectuera les conversions nécessaires : dans une expression mélangeant dates et chaînes de caractères, les chaînes de caractères seront converties en dates, dans une expression mélangeant nombres et chaînes de caractères, les chaînes de caractères seront converties en nombre.

6-1-1-Expressions et fonctions arithmétiques

Une expression arithmétique peut contenir :

- Des noms de colonnes ,des constantes, des fonctions arithmétiques combinés au moyen des opérateurs arithmétiques.

Opérateurs arithmétiques

Les opérateurs arithmétiques présents dans sql sont les suivants :

- addition , soustraction, multiplication , division

Remarque : la division par 0 provoque une fin avec code d'erreur.

6.1.3 Priorité des opérateurs

Une expression arithmétique peut comporter plusieurs opérateurs. Dans ce cas, le résultat de l'expression peut varier selon l'ordre dans lequel sont effectuées les opérations. Les opérateurs de multiplication et de division sont prioritaires par rapport aux opérateurs d'addition et de soustraction. Des parenthèses peuvent être utilisées pour forcer l'évaluation de l'expression dans un ordre différent de celui découlant de la priorité des opérateurs.

Exemple : Donner pour chaque commercial son revenu (salaire + commission).

```
SELECT nom,12*salaire  AS SALAIRE_ANNUEL
FROM emp
WHERE fonction = 'MANAGER';
```

	nom	SALAIRE_ANNUEL
1	JONES	35700.00
2	BLAKE	34200.00
3	CLARK	29400.00

Exemple : Donner la liste des SECRETAIRES classée par commission sur salaire décroissant.

```
SELECT nom, comm/salaire, comm, salaire
FROM emp WHERE fonction = 'SALESMAN'
ORDER BY comm/salaire DESC;
```

	nom	(No column name)	comm	salaire
1	MARTIN	1.1200000000	1400.00	1250.00
2	WARD	.4000000000	500.00	1250.00
3	ALLEN	.1875000000	300.00	1600.00
4	TURNER	.0000000000	.00	1500.00

Exemple : Donner la liste des employés dont la commission est inférieure à 5% du salaire.

```
SELECT nom, salaire, comm FROM emp WHERE comm <= salaire *.05;
```

	nom	salaire	comm
1	TURNER	1500.00	.00

6.1.4 Fonctions arithmétiques

Dans ce paragraphe, ont été regroupées les fonctions ayant un ou plusieurs nombres comme arguments, et renvoyant une valeur numérique.

FLOOR(nb): Renvoie le plus grand entier inférieur ou égal à nb.

LOG(m,n): Renvoie le logarithme en base m de n. m doit être un entier strictement supérieur à 1, et n un entier s

trictement positif.

POWER(m,n): Renvoie m puissance n, m et n peuvent être des nombres quelconques entiers ou rééls mais si m est négatif n doit être un entier.

ROUND(n[,m]): Si m est positif, renvoie n arrondi (et non pas tronqué) à m chiffres après la virgule. Si m est négatif, renvoie n arrondi à m chiffres avant la virgule. m doit être un entier et il vaut 0 par défaut.

SIGN(nb): Renvoie -1 si nb est négatif, 0 si nb est nul, 1 si nb est positif.

Exemple :

```
SELECT FLOOR(10.8) AS VAL1,LOG(2) AS VAL2,
    POWER(2,4) AS VAL3,ROUND(123.456,2) AS VAL4,
    SIGN(123) AS VAL5,SIGN(-2) AS VAL6;
```

	VAL1	VAL2	VAL3	VAL4	VAL5	VAL6
1	10	0.69314718055994529	16	123.460	1	-1

Exemple : Donner pour chaque employé son salaire journalier.

```
SELECT nom, ROUND(salaire/22,2)
FROM emp
WHERE N_dept = 20;
```

	nom	(No column name)
1	SMITH	36.360000
2	JONES	135.230000
3	SCOTT	136.360000
4	ADAMS	50.000000
5	FORD	136.360000

6-1.5-Expressions et fonctions sur les chaînes de caractères:

6.1.5.1 Opérateur sur les chaînes de caractères

Il existe un seul opérateur sur les chaînes de caractères : la **concaténation**. Cet opérateur se note +. Le résultat d'une concaténation est une chaîne de caractères obtenue en écrivant d'abord la chaîne à gauche de + puis celle à droite de +. Exemple :

```
SELECT emp.nom+' travaille a '+dept.nom as employe
FROM emp inner join dept on emp.n_dept=dept.n_dept
and salaire >=3000;
```

	employe
1	SCOTT travaille a RESEARCH
2	KING travaille a ACCOUNTING
3	FORD travaille a RESEARCH

6.1.5.2 Fonctions sur les chaînes de caractères

Le paragraphe suivant contient les fonctions travaillant sur les chaînes de caractères et renvoyant des chaînes de caractères.

LOWER(chaîne):	Renvoie chaîne en ayant mis toutes ses lettres en minuscules.
UPPER(chaîne):	Renvoie chaîne en ayant mis toutes ses lettres en majuscules.
LTRIM(chaîne)	Renvoie la chaîne obtenue en parcourant à partir de la gauche chaîne et en supprimant tous les espaces.
RTRIM(chaîne[,ens]):	Renvoie la chaîne obtenue en parcourant à partir de la droite chaîne et en supprimant tous les espaces.
REPLACE(chaine, avant, après)	Renvoie chaine dans laquelle toutes les occurrences de la chaîne de caractères avant ont été remplacés par la chaîne de caractères après.
Left(chaine,n)	Permet d'extraire les n premiers caractères de chaîne de la gauche vers la droite.
Right(chaine,n)	Permet d'extraire les n premiers caractères de chaîne de la droite vers la gauche.

SOUNDEX(chaîne): Renvoie la chaîne de caractères constituée de la représentation phonétique des mots de chaîne.

SUBSTRING (nom ,n ,m) La fonction SUBSTRING permet d'extraire une sous chaîne d'une chaîne de caractère. Elle a besoin de l'ordre du premier caractère et du nombre de caractères sur lequel elle doit opérer. Extrait la sous chaîne de nom de colonne en commençant à n sur m caractères.

Exemple :

```
select nom, right(nom,4) "right",left(nom,4) "left", substring(nom,2,4) "substring"
from emp
where n_dept=20;
```

	nom	right	left	substring
1	SMITH	MITH	SMIT	MITH
2	JONES	ONES	JONE	ONES
3	SCOTT	COTT	SCOT	COTT
4	ADAMS	DAMS	ADAM	DAMS
5	FORD	FORD	FORD	ORD

LEN(chaîne): Renvoie la longueur de chaîne, exprimée en nombre de caractères.

Datalength(champ) Renvoie la longueur du champ ou variable passée comme argument.

Exemple :

```
SELECT salaire,datalength(salaire) "Datalength",len(salaire) "Len"
FROM emp
where n_dept=20;
```

	salaire	Datalength	Len
1	800.00	5	6
2	2975.00	5	7
3	3000.00	5	7
4	1100.00	5	7
5	3000.00	5	7

Str(champ) Convertit le contenu du champ en chaîne de caractères.

6-1-6-Expressions et fonctions sur les dates

6.1.6.1- Opérateurs sur les dates

Au moyen des opérateurs arithmétiques + et - il est possible de construire les expressions suivantes :

- date +/- nombre : le résultat est une date obtenue en ajoutant le nombre de jours nombre à la date date.

- date2 - date1 : le résultat est une date qui vaut le '01/01/1900' plus le nombre de jours écoulés entre les deux dates.

Exemple :

```
SELECT getdate() , getdate() – convert (datetime , '13-08-2004' ),
getdate()-convert(datetime,'15-08-2004');
```

	(No column name)	(No column name)	(No column name)
1	2004-08-14 16:42:57.343	1900-01-02 16:42:57.343	1899-12-31 16:42:57.343

La fonction GETDATE permet de calculer la date système :

Exemple :

```
SELECT getdate();
```

	(No column name)
1	2001-10-28 23:31:37.973

6.1.6.2 Fonctions sur les dates:

DATEPART(format, date) — Extrait une partie de la date soit le jour soit le mois soit l'année en fonction de la valeur de format [year, month , day, hh(heure), mi(minute), ou ss(seconde) …]

DATEADD(format, valeur , date) — Ajout valeur a la date selon le format spécifié
Si format = year , on ajoute valeur an a la date
Si format = month, on ajoute valeur mois a la date, … etc.

Exemple :

```
select   getdate() today,dateadd(year , 10,getdate()) add_year,
         dateadd(Month , 10,getdate()) add_month,dateadd(day , 10,getdate())  add_day;
```

	today	add_year	add_month	add_day
1	2004-08-14	2014-08-14	2005-06-14	2004-08-24

Exemple : Donner l'année de la date d'embauche de chaque employé travaillant dans le département 20..

```
SELECT DATEPART(year,embauche) annee_embauche
FROM emp
where n_dept=20;
```

	annee_embauche
1	1980
2	1981
3	1987
4	1987
5	1981

DATDIFF(format, date1 , date2) retourne la différence entre date1 et date2, cette différence est estimée selon la valeur du paramètre format :
Si format = year , on calcule la différence entre les années
Si format = month, on calcule la différence entre les mois, … etc.

Exemples :

```
SELECT GETDATE(),DATEDIFF(YEAR,'01/04/2002',GETDATE())
```

	(Aucun nom de colonne)	(Aucun nom de colonne)
1	2004-08-14 18:30:14.500	2

```
SELECT GETDATE(),DATEDIFF(MONTH,'01/04/2002',GETDATE())
```

	(Aucun nom de colonne)	(Aucun nom de colonne)
1	2004-08-14 18:37:24.720	28

```
SELECT GETDATE(),DATEDIFF(DAY,EMBAUCHE,GETDATE()) "JOURS TRVAILLES"
FROM EMP
WHERE N_DEPT=10
```

	(Aucun nom de colonne)	JOURS TRVAILLES
1	2004-08-14 18:40:13.023	8467
2	2004-08-14 18:40:13.023	8306
3	2004-08-14 18:40:13.023	8239

6-1-7-Fonctions de conversion:

ASCII(chaine): Renvoie le nombre correspondant au code ascii du premier caractère de chaine.

CHAR(nombre): Renvoie le caractère dont nombre est le code ascii.

Convert(format,champ) Convertit la valeur du champ en format de données spécifié en premier argument.
Format peut être soit int, smallint, float, money,varchar …..etc

Exemple :

```
SELECT str(salaire) "str",convert(int,salaire) "convert",salaire
FROM emp
where fonction='MANAGER';
```

	str	convert	salaire
1	2975	2975	2975.00
2	2850	2850	2850.00
3	2450	2450	2450.00

Exemple :

```
SELECT convert(bigint,getdate()),getdate(),
convert(bigint,convert(datetime,'04-01-1900'));
```

	(No column name)	(No column name)	(No column name)
1	38212	2004-08-14 16:33:43.330	3

Exemple : Donner la liste de tous les employés dont le nom se prononce de la même façon que 'SCOTT'.

```
SELECT nom
FROM emp
WHERE SOUNDEX(nom) = SOUNDEX('SCOTT');
```

6-1-8-Autres fonctions

Les branchements dans le SQL (CASE)

SQL possède un branchement à la manière des IF et autres structures de test des langages procéduraux. Mais il convient de ne l'utiliser qu'a bon escient, c'est à dire aussi peu souvent que possible, beaucoup de cas ouvant être traités soit par le COALESCE soit par des requêtes avec des opérations ensemblistes de type UNION. En effet les performances se dégradent très vite lors de l'usage du CASE à cause de l'impossibilité d'effectuer des traitements par "paquets".

Syntaxe :

```
CASE expression
  WHEN valeur1 THEN expression1
 [WHEN valeur2 THEN expression2]
  ...
 [ELSE expression_défaut]
END
```

Exemple : Augmentation salariale que pour les MANASGER de 10% et les SALESMANS de 5%

```
SELECT NOM, FONCTION ,SALAIRE,
    CASE FONCTION
       WHEN 'MANAGER' THEN SALAIRE*0.1
       WHEN 'SALESMAN' THEN SALAIRE*0.05
       ELSE SALAIRE
    END AS AUGMENTATION
FROM EMP
WHERE N_DEPT=20;
```

	NOM	FONCTION	SALAIRE	AUGMENTATION
1	SMITH	CLERK	800.00	800.0000
2	JONES	MANAGER	2975.00	297.5000
3	SCOTT	ANALYST	3000.00	3000.0000
4	ADAMS	CLERK	1100.00	1100.0000
5	FORD	ANALYST	3000.00	3000.0000

Exemple : Donner la liste des employés en les identifiant par leur fonction dans le département 10 et par leur nom dans les autres départements.

```
SELECT NUM,
    CASE N_DEPT
       WHEN 10 THEN NOM

       ELSE FONCTION
    END AS IDENTIFICATION,
    N_DEPT
FROM EMP;
```

	NUM	IDENTIFICATION	N_DEPT
1	7369	CLERK	20
2	7499	SALESMAN	30
3	7521	SALESMAN	30
4	7566	MANAGER	20
5	7654	SALESMAN	30
6	7698	MANAGER	30
7	7782	CLARK	10
8	7788	ANALYST	20
9	7839	KING	10
10	7844	SALESMAN	30
11	7876	CLERK	20
12	7900	CLERK	30
13	7902	ANALYST	20
14	7934	MILLER	10

Traitement des "valeurs" nulles

ISNULL (EXP, VAL): Prends la valeur **EXP**, sauf si **EXP** est NULL auquel cas **ISNULL** prend la valeur **VAL**.

Une valeur NULL en SQL est une valeur non définie ce n'est pas zero.
Lorsque l'un des termes d'une expression a la valeur NULL, l'expression entière prend la valeur NULL. D'autre part, un prédicat comportant une comparaison avec une expression ayant la valeur NULL prendra toujours la valeur faux. La fonction ISNULL permet de remplacer une valeur NULL par une valeur significative.

Exemple : Donner pour chaque employé ses revenus (salaire + commission) pour tous les employés ayant un salaire >= 3000$

Sans utilisation de la fonction ISNULL

```
SELECT nom, salaire, comm, SALAIRE+comm sal_et_comm
FROM emp
where salaire >=3000;
```

	nom	salaire	comm	sal_et_comm
1	SCOTT	3000.00	NULL	NULL
2	KING	5000.00	NULL	NULL
3	FORD	3000.00	NULL	NULL

Avec utilisation de la fonction ISNULL

```
SELECT nom, salaire, comm, SALAIRE+ISNULL(comm,0) sal_et_comm
FROM emp
where salaire >=3000;
```

	nom	salaire	comm	sal_et_comm
1	SCOTT	3000.00	NULL	3000.00
2	KING	5000.00	NULL	5000.00
3	FORD	3000.00	NULL	3000.00

6-2-Les fonctions de groupe :

6-2-1-Les fonctions de groupe:

Dans les exemples précédents, chaque ligne résultat d'un SELECT était le résultat de calculs sur les valeurs d'une seule ligne de la table consultée. Il existe un autre type de SELECT qui permet d'effectuer des calculs sur l'ensemble des valeurs d'une colonne. Ces calculs sur l'ensemble des valeurs d'une colonne se font au moyen de l'une des fonctions suivantes :

AVG([DISTINCT | ALL] expression):	Renvoie la moyenne des valeurs d'expression.
COUNT(* | [DISTINCT | ALL] expression):	Renvoie le nombre de lignes du résultat de la requête. Si expression est présent, on ne compte que les lignes pour lesquelles cette expression n'est pas NULL.
MAX([DISTINCT | ALL] expression):	Renvoie la plus petite des valeurs d'expression.
MIN([DISTINCT | ALL] expression):	Renvoie la plus grande des valeurs d'expression.
SUM([DISTINCT | ALL] expression):	Renvoie la somme des valeurs
DISTINCT:	Indique à la fonction de groupe de ne prendre en compte que des valeurs distinctes.
ALL	Indique à la fonction de groupe de prendre en compte toutes les valeurs, c'est la valeur par défaut.

Exemple : Donner le total des salaires du département 10.

```
SELECT SUM(salaire) FROM emp WHERE n_dept = 10;
```

	(Aucun nom de colonne)
1	8750.00

Exemple : Donner le nom, la fonction et le salaire de l'employé (ou des employés) ayant le salaire le plus élevé.

```
SELECT nom, fonction, salaire FROM emp WHERE salaire= (SELECT MAX(salaire) FROM emp);
```

	nom	fonction	salaire
1	KING	PRESIDENT	5000.00

Remarques

- Ces SELECT sont différents de ceux vus précédemment. Il est, par exemple, impossible de demander en résultat à la fois une colonne et une fonction de groupe.
- un SELECT comportant une fonction de groupe peut être utilisé dans une sous-interrogation.

6-2-2-Valeurs NULL

Aucune des fonctions de groupe ne tient compte des valeurs NULL à l'exception de count(*). Ainsi, SUM(col) est la somme des valeurs non NULL de la colonne col. De même AVG est la somme des valeurs non NULL divisée par le nombre de valeurs non NULL.

6-2-3-Calcul sur plusieurs groupes

Il est possible de subdiviser la table en groupes, chaque groupe étant l'ensemble des lignes ayant une valeur commune. C'est la clause GROUP BY qui permet de découper la table en plusieurs groupes :

GROUP BY expr_1, expr_2, ...

Si on a une seule expression, ceci définit les groupes comme les ensembles de lignes pour lesquelles cette expression prend la même valeur. Si plusieurs expressions sont présentes les groupes sont définis de la façon suivante : parmi toutes les lignes pour lesquelles expr_1 prend la même valeur, on regroupe celles ayant expr_2 identique, ... Un SELECT de groupe avec une clause GROUP BY donnera une ligne résultat pour chaque groupe.

Exemple : Total des salaires pour chaque département

```
SELECT n_dept,SUM(salaire) "somme salaire"  FROM emp
GROUP BY n_dept;
```

	n_dept	somme salaire
1	10	8750.00
2	20	10875.00
3	30	9400.00

Exemple : Sélectionner pour chaque département et pqr fonction le maximum salaire et le minimum salaire

```
SELECT n_dept, fonction, min(salaire) "minimum sal" , max(salaire) "miximum sal"
FROM emp
GROUP BY n_dept,fonction;
```

	n_dept	fonction	minimum sal	miximum sal
1	20	ANALYST	3000.00	3000.00
2	10	CLERK	1300.00	1300.00
3	20	CLERK	800.00	1100.00
4	30	CLERK	950.00	950.00
5	10	MANAGER	2450.00	2450.00
6	20	MANAGER	2975.00	2975.00
7	30	MANAGER	2850.00	2850.00
8	10	PRESIDENT	5000.00	5000.00
9	30	SALESMAN	1250.00	1600.00

Remarque : Dans la liste des colonnes résultat d'un SELECT comportant une fonction de groupe, ne peuvent figurer que des caractéristiques de groupe, c'est-à-dire :

- soit des fonctions de groupe ;
- soit des noms de colonnes figurant dans le GROUP BY.

6-2-4-Sélection des groupes:

De la même façon qu'il est possible de sélectionner certaines lignes au moyen de la clause WHERE, il est possible dans un SELECT comportant une fonction de groupe de sélectionner par la clause HAVING, qui se place après la clause GROUP BY.

Le prédicat dans la clause HAVING suit les mêmes règles de syntaxe qu'un prédicat figurant dans une clause WHERE.

Cependant, il ne peut porter que sur des caractéristiques du groupe : fonction de groupe ou expression figurant dans la clause GROUP BY, dans ce cas **la clause HAVING doit être placée après la clause GROUP BY**.

Exemple : Donner la liste des salaires moyens par fonction pour les groupes ayant plus de deux employés.

```
SELECT fonction,COUNT(*) compte,AVG(salaire) moyenne
FROM emp
GROUP BY fonction
HAVING COUNT(*) > 2;
```

	fonction	compte	moyenne
1	CLERK	4	1037.500000
2	MANAGER	3	2758.333333
3	SALESMAN	4	1400.000000

Remarque : Un SELECT de groupe peut contenir à la fois une clause WHERE et une clause HAVING. La clause WHERE sera d'abord appliquée pour sélectionner les lignes, puis les groupes seront constitués à partir des lignes sélectionnées, et les fonctions de groupe seront évaluées.

Exemple : Donner le nombre de managers ou de secrétaires des départements ayant au moins deux employés de ces catégories.

```
SELECT n_dept, COUNT(*) compte
FROM emp
WHERE fonction in ('MANAGER','SALESMAN')
GROUP BY n_dept
HAVING COUNT(*) >= 2;
```

	n_dept	compte
1	30	5

Une clause HAVING peut comporter une sous-interrogation.

Exemple : Quel est le département ayant le plus d'employés?

```
SELECT n_dept,COUNT(*) effectif
FROM emp
GROUP BY n_dept
HAVING COUNT(*) >=all (   SELECT COUNT(*)   FROM emp
                          GROUP BY n_dept ) ;
```

	n_dept	effectif
1	30	6

7. EXERCICES :

Durant ces exercices nous serons amenés a travailler avec le schéma relationnel suivant :

Série 1

1.1. Ecrire la requête qui présente tous les pilotes de la compagnie avec le listing suivant:

Numéro Nom Adresse Salaire Mensuel

1.2.Ecrire la requête qui présente tous les pilotes de la compagnie selon le listing suivant

Nom Salaire Mensuel Numéro Adresse

1.3.Ecrire la requête qui renomme(alias) la relation PILOTE en P dans une requête.

1.4.Ecrire la requête qui calcule la durée d'un vol.

1.5.Ecrire une requête qui calcule le salaire annuel SAL_ANN, pour chaque pilote.

1.6.Ecrire une requête qui calcule la somme des salaires des pilotes.

1.7.Donner tous les types d'avions de la compagnie

Série 2

Ecrire les requêtes qui permettent de rechercher :

2.1 Numéros et type d'avions de capacité supérieure à 300

2.2 Nom des pilotes habitants Nice ou Paris

2.3 Quels sont les noms de pilotes comportant un 't' en quatrième position ou dont le nom se prononce 'Bodri'.

2.4 Quels sont les vols au départ de Nice, Paris ou Bordeaux ?

2.5 Quels sont les avions dont la capacité est comprise entre 250 et 310

2.6 Quels sont les pilotes dont l'adresse ou le téléphone sont inconnus ?

2.7 Nom des pilotes ayant un 'a' et un 'e' dans leur nom

2.8 Nom des pilotes ayant 2 'o' dans leur nom

2.9 Nom des pilotes dont le numéro de téléphone est renseign

Série 3

3.1 Lister les pilotes avec leur salaire tronqués au millier

*3.2 Lister les pilotes avec leur salaire. Pour ceux gagnat 17000,6 remplacer le salaire par '****'*

3.3 Sélectionner les pilotes et leur téléphone. Pour ceux dont le télephone n'est pas renseigné, mettre ?

Série 4

4.1 Ecrire une requête qui donne le salaire du pilote qui gagne le plus :
<valeur à calculer> "Max salaire Pilote "

4.2 Quels sont les noms, l'adresse et le salaire des pilotes de la compagnie, triés en ordre croissant sur l'adresse, et pour une même adresse en ordre décroissant sur le salaire ?

4.3 Ecrire une requête qui recherche si l'utilisateur courant de sql server est un pilote ?

4.4 Ecrire une requête qui rend ROWID, USER, SYSDATE, Numéros de vol de tous les vols effectués à la date d'aujourd'hui par le pilote Numéro 4 ?". L'heure de départ et d'arrivée doivent apparaître dans la liste des colonnes de projection.

Série 5

5.1 Donner toutes les paires de noms de pilotes distincts, habitant la même ville

5.2 Donner tous les noms des pilotes qui ont des noms d'avions ?

5.3 Ecrire la requête qui donne les noms des pilotes qui conduisent un A300 ou B727 ?

5.4 Essayer d'exprimer l'intersection a l'aide d'une jointure interne selon la requête suivante :
(SELECT IDPILOTE, VD, VA

FROM vol)
INTERSECT
(SELECT IDAVION, VD, VA
FROM VOL
);
Quel est sa signification en langage naturel ?

5.5 Quel est le nom des avions dont la capacité est supérieure à la capacité de chaque avion localisé à Nice ?

5.6 Quel est le nom des avions dont la capacité est au moins égale à celle d'un avion localisé à Nice ?

5.7 Quel est le nom des pilotes assurant un vol au départ de Nice ?

5.8 Quel est le nom des pilotes assurant au moins un vol ?

5.9 Quel est le nom des pilotes dont le salaire est supérieure au salaire maximum de tous les pilotes effectuant un vol au départ de Paris ?

5.10 Quels sont les noms des pilotes qui gagnent plus que le pilote nr. 5?

5,11 Donner le nom des pilotes, et pour ceux qui sont en service, la liste des numéros de vols qu'ils assurent ?

5.12 Pour chaque ville de localisation d'avions de la compagnie (sauf "Paris") donner le nombre, les capacités minimales et maximales d'avions qui s'y trouvent ?

5.13 Quels sont les pilotes (avec leur nombre de vols) parmi les pilotes N° 1, 2, 3 , 4 et 13 qui assurent au moins 2 vols ?

5.14 Quelle est la capacité moyenne des avions par ville et par type ?

OFPPT

ROYAUME DU MAROC

مكتب التكوين المهني وإنعاش الشغل

Office de la Formation Professionnelle et de la Promotion du Travail

INSTITUT SUPERIEUR DE GESTION ET D'INFORMATIQUE

Filière : TSDI

SEQUENCE III

Le LANGAGE DE MANIPULATION DE DONNEES

(suite)

Modification, suppression et insertion de données

Objectif principal	**Manipuler les données extraites de la base**		
Objectifs intermédiaires	➢ **Décrire chaque ordre du LMD** ➢ **Insérer des lignes dans une table** ➢ **Mettre à jour des lignes dans une table** ➢ **Supprimer des lignes d'une table** ➢ **controler les transactions**		
Volume horaire théorique		**Volume horaire pratique**	

Définition: Le **L**angage de **M**anipulation de **D**onnées (LMD) est le langage permettant de modifier les informations contenues dans une base de données. L'unité manipulée est la ligne. Il existe trois commandes sql permettant d'effectuer les trois types de modifications des données : ajout, modification et suppression.

1- Insertion de lignes : INSERT

La commande INSERT permet d'insérer une ligne dans une table en spécifiant les valeurs à insérer. La syntaxe est la suivante :

```
INSERT INTO nom_table(nom_col1, nom_col2, ...)
VALUES (val1, val2...)
```

La liste des noms de colonne est optionnelle. Si elle est omise, la liste des colonnes sera par défaut la liste de l'ensemble des colonnes de la table dans l'ordre de la création de la table.
Si une liste de colonnes est spécifiée, les colonnes ne figurant pas dans la liste auront la valeur NULL.

Il est possible d'insérer dans une table des lignes provenant d'une autre table. La syntaxe est la suivante :

```
INSERT INTO nom_table(nom_col1, nom_col2, ...)   SELECT ...
```

Le SELECT peut contenir n'importe quelle clause sauf un ORDER BY qui impliquerait un classement des lignes contraire à l'esprit du relationnel.

Exemple : Insérer dans la table bonus les noms et salaires des MANAGERS.

```
INSERT INTO bonus SELECT nom, salaire  FROM emp   WHERE fonction = ' MANAGER';
```

2- Modification de lignes : Update

La commande UPDATE permet de modifier les valeurs d'une ou plusieurs colonnes, dans une ou plusieurs lignes existantes d'une table. La syntaxe est la suivante :

```
UPDATE nom_table SET nom_col1 = {expression1 | ( SELECT ...) },
                                    nom_col2 = {expression2 | ( SELECT ...) }
WHERE predicat
```

Les valeurs des colonnes nom_col1, nom_col2, ... sont modifiées dans toutes les lignes satisfaisant au prédicat. En l'absence d'une clause WHERE, toutes les lignes sont mises à jour.
Les expressions expression1, expression2,... peuvent faire référence aux anciennes valeurs de la ligne.

Exemple : Augmenter de 10% les secretaires.

```
UPDATE emp SET salaire = salaire * 1.1 WHERE fonction = 'CLERK' ;
```

3- Suppression de lignes : Delete

La commande DELETE permet de supprimer des lignes d'une table.
La syntaxe est la suivante :

```
DELETE FROM nom_table  WHERE prédicat ;
```

Toutes les lignes pour lesquelles prédicat est évalué à vrai sont supprimées. En l'absence de clause WHERE, toutes les lignes de la table sont supprimées.

Exemple : Supprimer tous les employés ayant une commission nulle.

```
DELETE FROM emp  WHERE comm IS NULL ;
```

4. Transactions base de données :

4.1 Définition :

Sql Server garantit la cohérence des données par le biais des transactions. Les transactions vous offrent davantage de souplesse et un meilleur contrôle lors de la modification de données. Elles garantissent la cohérence des données en cas d'échec du processus utilisateur ou de panne du système.
Les transactions consistent en un ensemble d'ordres du LMD qui réalisent une modification cohérente des données. Par exemple, un transfert de fonds entre deux comptes implique de débiter un compte et d'en créditer un autre du même montant. Les deux actions doivent, soit réussir, soit échouer en même temps : un crédit ne peut pas être validé sans le débit correspondant.

4.2 Types de Transactions

Type	Description
Langage de manipulation des données (LMD)	Comprend un nombre quelconque d'ordres LMD que Sql Server traite en tant qu'entité unique ou unité de travail logique.
Langage de définition des données (LDD)	Comprend un seul ordre LDD
Langage de contrôle des données (LCD)	Comprend un seul ordre LCD

4.3 Quand Commence et Finit une Transaction ?

Une transaction commence dès le premier ordre SQL exécutable rencontré et se termine lorsque l'un des événements suivants se produit :

- Un ordre COMMIT ou ROLLBACK est lancé
- L'utilisateur quitte la session Base de données
- Il se produit une panne de machine ou du système d'exploitation

Lorsqu'une transaction prend fin, le prochain ordre SQL exécutable démarrera automatiquement la transaction suivante.

4.4 Ordres de Contrôle Explicite des Transactions

Vous avez la possibilité de contrôler la logique des transactions au moyen des ordres COMMIT, SAVEPOINT et ROLLBACK.

Ordre	**Description**
BEGIN TRANSACTION [*nom*]	Marque le début d'une transaction base de données
COMMIT [TRANSACTION *nom*]	Met fin à la transaction courante en rendant définitives toutes les modifications de données en instance.
SAVE TRANSACTION [*nom*]	Pose une étiquette dans la transaction courante.
ROLLBACK [TRANSACTION *nom*].	ROLLBACK met fin à la transaction courante et rejette toutes les modifications de données en instance. ROLLBACK TRANSACTION *name* annule toutes les modifications jusqu'au point de sauvegarde et supprime celui-ci.

4.5 Traitement Implicite des Transactions

Etat	**Circonstances**
Commit automatique	Exécution d'un ordre du LDD ou du LCD pour certains SGBD Sortie normale de l'analyseur de requêtes, sans précision d'un ordre COMMIT ou ROLLBACK explicite
Rollback automatique	Fin anormale de l'analyseur de requêtes ou panne du système

Remarque :

Plusieurs transactions peuvent êtres imbriquée, il est alors possible de connaître le niveau de la transaction à l'aide de la variable système *@@trancount*.

4.6 Etat des Données Avant COMMIT ou ROLLBACK

- Les opérations de manipulation des données se déroulant principalement dans le buffer de la base de données, il est possible de restaurer l'état précédent des données.
- L'utilisateur courant peut afficher le résultat de ses opérations de manipulation de données en interrogeant les tables.
- Les autres utilisateurs ne peuvent pas voir le résultat des opérations de manipulation de données réalisées par l'utilisateur courant. Sql Server met en œuvre un principe de lecture cohérente qui garantit que l'utilisateur voit les données telles qu'elles se présentaient lors de la dernière validation.
- Les lignes concernées par la transaction sont verrouillées ; aucun autre utilisateur ne peut modifier les données qu'elles contiennent.

4.7 Etat des Données APRES COMMIT

Pour enregistrer définitivement les modifications en instance, utilisez l'ordre COMMIT. Après l'exécution d'un ordre COMMIT :

- Les modifications de données sont écrites définitivement dans la base de données.
- L'état précédent des données est irrémédiablement perdu.
- Tous les utilisateurs peuvent voir les résultats de la transaction.
- Les lignes qui étaient verrouillées sont libérées et redeviennent accessibles à d'autres utilisateurs pour modification.
- Tous les points de sauvegarde sont effacés.

4.7 Exemples :

Créer un nouveau département ADVERTISING comportant au moins un employé. Enregistrer définitivement cette modification.

```
Begin tran debut

Go

INSERT INTO department(n_dept, nom, loc)
 VALUES      (50, 'ADVERTISING', 'MIAMI');

Go
Select * from dept;


Go
Save tran tr01

Go
UPDATE  emp
SET     n_dept = 50
WHERE   num = 7876;

Go
Select * from emp

Go
Rollback tran tr01

Go
Select * from emp

Go
UPDATE  emp
SET     n_dept = 50
WHERE   num = 7876;

Go
Commit;

Go
Select * from emp;
```

5. Exercices :

Série 6

6.1 Effectuer des insertions respectivement dans pilote, avion et vol. Vérifier si les contraintes l'intégrités structurelles (entitité, domaine et de référence) sont prises en comptes. Vérifier aussi les valeurs nulles.

Note : insérer un pilote ayant votre nom de login SQL SERVER et 2 vols effectués par ce pilote.

6.2 Effectuer une insertion dans la table PILOTE2 via une sous-requête sur PILOTE.

6.3 Mettre à jour le salaire du pilote numéro 3 à 19000 F et Valider.

6.4 Supprimer le pilote numéro 11 et invalider.

6.5Supprimer les lignes de la tables PILOTE2 via TRUNCATE. Tentez un ROLLBACK.

OFPPT

ROYAUME DU MAROC

مكتب التكوين المهني وإنعاش الشغل

Office de la Formation Professionnelle et de la Promotion du Travail

INSTITUT SUPERIEUR DE GESTION ET D'INFORMATIQUE

Filière : TSDI

SEQUENCE IV

LE LANGAGE DE CONTOLE DE DONNEES

(suite)

IV – LA GESTION DES PRIVILEGES

Objectif principal	**Gérer les privilèges et les droits d'accès**		
Objectifs intermédiaires	➢ **Attribuer un droit d'un accès sur objet a un utilisateur** ➢ **Retirer un droit d'accès sur un objet d'un utilisateur**		
Volume horaire théorique		**Volume horaire pratique**	

On emploie à tort le mot "droit" pour expliquer, dans SQL, le sous ensemble DCL dit "data control langage" qui s'occupe de la gestion des privilèges. Cette partie de SQL 2 s'occupe de contrôler quel utilisateur peut ou ne peut pas utiliser tel ou tel ordre SQL sur tel ou tel objet et en cette matière il n'y a pas que des droits, mais aussi des "usages" !

1. La notion d'utilisateur

La notion d'utilisateur possède une lacune importante dans SQL car elle fait l'impasse sur la façon dont on créé un utilisateur...

NOTA : SQL 2 a prévu des règles d'accès spécifiques pour les différentes familles d'ordre SQL. Ainsi pour la création d'un schéma (une base de donnée en fait) ou d'un catalogue (un ensemble de bases de données), il laisse la règle à l'appréciation de l'éditeur du SGBDR. En revanche pour les ordres CREATE, ALTER et DROP, l'utilisateur courant doit être le même que l'utilisateur propriétaire (auteur) du schéma modifié.

Rappelons que la création d'un utilisateur se fait au moment de la création du schéma. Par défaut ce dernier est le créateur des objets.

Souvent dans les implémentation commerciales, on trouve un pseudo ordre SQL du genre *CREATE USER nomUtilisateur*, ou encore une procédure stockée permettant de définir un nouvel utilisateur.

Ainsi, pour MS SQL Server :

sp_adduser 'nomConnexion', 'nomNouvelUtilisateur'

Permet d'ajouter un nouvel utilisateur affecté à la connexion donnée.

La norme SQL2 propose trois fonctions pour connaître l'utilisateur (c'est à dire celui qui se connecte au serveur) et l'auteur, c'est à dire le créateur des objets :

SYSTEM_USER	nom de connexion
SESSION_USER	nom du créateur
CURRENT_USER	utilisateur courant

Ainsi, par défaut, SQL Server utilise les noms suivants :

Exemple 1

```
SELECT SYSTEM_USER system, SESSION_USER session ,CURRENT_USER courant
```

	system	session	courant
1	scott	dbo	dbo

2. Octroyer des privilèges

Les privilèges sont, pour un ou plusieurs utilisateurs la possibilité d'utiliser certains objets et parmi ces objets, certains ordres SQL.

2.1. Attribution de privilèges

C'est l'ordre SQL GRANT qui permet d'attribuer un privilège à différents utilisateurs sur différents objets.

Voici la syntaxe de l'ordre SQL GRANT :

GRANT { <privilège> [{ , <privilège> }...] ON [TABLE] <objet table> |
ALL PRIVILEGES ON [TABLE] <objet table> |
REFERENCES [(<liste de colonne>)] <nom de table> }

TO <user1> [{ , <user2> }...] | public

[WITH GRANT OPTION]

AVEC privilège : **soit SELECT**
soit DELETE
soit INSERT [(<liste de colonne>)]
soit UPDATE [(<liste de colonne>)]

La clause WITH GRANT OPTION, est utilisée pour autoriser la transmission des droits. La clause ALL PRIVILEGES n'a d'intérêt que dans le cadre de la transmission des droits.

Voici maintenant une batterie d'exemples afin de mieux comprendre comment utiliser cet ordre... Pour nos exemples, nous avons considéré que les utilisateurs DUPONT, DURAND, DUBOIS, DUVAL, DULAC et DUFOUR était créé dans la base de données. Celui qui lance les ordres (sauf indication contraire) est l'utilisateur SCOTT.

Exemple 2

```
GRANT SELECT
   ON EMP
```

```
TO DUBOIS
```

Autorise DUBOIS à lancer des ordres SQL SELECT sur la table EMP. Notez l'absence du mot TABLE.

Exemple 3

```
GRANT INSERT, UPDATE, DELETE
 ON TABLE SALGRADE
 TO DUVAL, DUBOIS
```

Autorise DUVAL et DUBOIS à modifier les données de la table SALGRADE par tous les ordres SQL de mise à jour (INSERT, UPDATE, DELETE) mais pas à les lire !

Exemple 4

```
GRANT SELECT
 ON TABLE DEPT
 TO DUFOUR WITH GRANT OPTION
```

Autorise DUFOUR à lancer des ordres SQL SELECT sur la table DEPT mais aussi à transmettre à tout autre utilisateur les droits qu'il a acquis dans cet ordre.

Exemple 5

```
GRANT SELECT, INSERT, DELETE
 ON TABLE EMP
 TO DURAND WITH GRANT OPTION
```

Autorise DURAND à lancer des ordres SQL SELECT, INSERT, DELETE sur la table T_CHAMBRE.

Exemple 6

```
GRANT SELECT, UPDATE
 ON TABLE DEPT
 TO PUBLIC
```

Autorise tous les utilisateurs présent et à venir à lancer des ordres SQL SELECT et UPDATE sur la table DEPT.

Exemple 7 **:** DURAND lance l'ordre suivant :

```
GRANT ALL PRIVILEGES
 ON TABLE PILOTES
 TO DUBOIS
```

Ce qui autorise DUBOIS à lancer sur la table PILOTES, les mêmes ordres SQL, que ceux autorisé à DURAND (SELECT, INSERT, DELETE).

On parle alors **d'héritage de droits** c'est à dire que l'utilisateur dotés de ces droits peut à nouveau les céder à un ou plusieurs autres utilisateurs.

Exemple 9 **:** DURAND lance l'ordre suivant :

```
GRANT UPDATE
 ON TABLE VOL
 TO DUBOIS
```

Cet ordre va provoqué une erreur, car DURAND n'est pas autorisé à lancer des ordres UPDATE sur la table VOL et ne peut donc transmettre un droit qu'il n'a pas !

2.3. Gestion fine des privilèges

Est-il possible de gérer des privilèges plus fins que sur l'intégralité de la table ou de vue ? En particulier, peut-on gérer des privilèges au niveau de certaines colonnes d'une table ?
La réponse est OUI, mais il faut utiliser un peu d'astuce...

2.3.1. Privilèges INSERT et UPDATE sur colonne

On peut employer l'ordre GRANT pour ce faire :

Exemple 10

```
GRANT UPDATE (HISAL, LOSAL)
 ON TABLE SALGRADE
 TO DULAC
```

Cet ordre permet à DULAC de modifier uniquement les colonnes "poste téléphonique" et "nombre de place de couchage" de la table VOL.

Plus curieux, on peut définir les colonnes utilisables pour un ordre d'insertion pour un utilisateur :

Exemple 11

```
GRANT INSERT (N_DEPT, LOC)
 ON TABLE DEPT
 TO DULAC
```

Cet ordre permet à DULAC d'insérer une nouvelle ligne dans la table, uniquement en spécifiant les colonnes listées. Le problème est que dans cette liste ne figure pas la colonne clef... Autrement dit, DULAC ne pourra jamais rien insérer du tout, sauf si la clef est calculée par un déclencheur avant insertion.

NOTA : dans le cas de l'attribution de privilèges d'insertion sur colonne, il est indispensable de faire figurer toutes les colonnes NOT NULL n'ayant ni clause de valeur par défaut, ni remplissage par un trigger avant insertion. Sans cela cette autorisation est illusoire !

2.3.2. Privilèges SELECT sur colonne

Ce type de privilège n'est pas géré directement par un ordre SQL.
En effet, il n'est pas possible d'écrire :

Exemple 12

```
GRANT SELECT (grade, MINSAL, MAXSAL)
 ON TABLE SALGRADE
 TO DULAC
```

Cet ordre n'est pas légal au niveau de SQL.

Mais un ordre GRANT peut porter sur une vue !
Nous voila donc sauvé : créer une vue pour gérer les privilèges de sélection de l'utilisateur DULAC..

Exemple 13

```
CREATE VIEW EMP_VU20
AS
  SELECT *
  FROM   EMP WHERE N_DEPT = 20
```

```
GRANT SELECT
ON EMP_VU20
TO DULAC
```

2.4. Privilèges de référence

Lorsque l'on crée des tables en liaisons les unes aux autres, on utilise très souvent le mécanisme d'intégrité référentiel afin de gérer les clefs étrangères.

Voyons ce qui se passe si, dans notre base exemple, nous attribuons les droits ainsi :

Exemple 14

```
GRANT SELECT, INSERT, UPDATE, DELETE
 ON EMP
 TO DUMONT
```

Dumont pourra sélectionner, et supprimer sans problèmes dans la table EMP.
En revanche il se heurtera parfois à un refus de la base de données pour la mise à jour de la même table.
Quel en est la raison ?

En effet, Elle utilise une table en référence : DEPT. Or notre utilisateur DUMONT n'a aucun privilège sur cette table. Il lui sera donc impossible lors de l'insertion, comme lors de la mise à jour de préciser une valeur pour cette colonne sans qu'il se voit automatiquement infligé un refus du serveur.

Or donc, pour pouvoir définir une valeur pour la colonne N_DEPT lors de l'exécution des ordres UPDATE et INSERT, notre utilisateur DUMONT, doit avoir un privilège supplémentaire définit comme suit :

Exemple 15

```
GRANT REFERENCES (N_DEPT)
 ON DEPT
 TO DUMONT
```

NOTA : le privilège de référence ne porte pas exclusivement sur les contraintes d'intégrité mais sur toute contrainte faisant référence à une colonne d'une table externe.

3. Révocation des privilèges

L'ordre SQL REVOKE permet de révoquer, c'est à dire "retirer" un privilège.
Sa syntaxe est la suivante :

```
REVOKE [ GRANT OPTION FOR ]
        {  <privilège> [ { , <privilège> }... ]  ON  [TABLE] <objet table> |
           ALL PRIVILEGES   ON  [TABLE] <objet table> |
           REFERENCES [ ( <liste de colonne> ) ] <nom de table>  }

FROM <user1> [ { , <user2> }... ] | public

[ RESTRICT | CASCADE ]
```

AVEC privilège : **soit SELECT**
soit DELETE
soit INSERT [(<liste de colonne>)]
soit UPDATE [(<liste de colonne>)]

La grande différence réside en fait dans l'usage des mots clefs RESTRICT et CASCADE. En cas de RESTRICT, si le ou les utilisateurs visés ont cédés leurs droits à d'autres, un message d'erreur apparaîtra et le SGBDR refusera de révoquer le ou les droits. En revanche l'usage du mot clef CASCADE entraînera la révocation des droits cédés à la manière d'un château de carte.

A noter que l'utilisation de l'expression GRANT OPTION FOR ne révoque pas les droits mais supprime la possibilité de cession des droits lorsque ces droits ont été définis en utilisant la clause WITH GRANT OPTION.

3.1. Quelques exemples simples

Exemple 16

```
REVOKE SELECT
 ON EMP
 FROM DUBOIS
```

Supprime le privilège de sélection de la table DEPT attribué à DUBOIS dans l'exemple 1.

Exemple 17

```
REVOKE INSERT, DELETE
 ON TABLE SALGRADE
  FROM DUVAL, DUBOIS
```

Supprime les privilèges d'insertion et de suppression de la table SALGRADE attribué à DUVAL et DUBOIS dans l'exemple 3, mais pas celui de mise à jour (UPDATE).

Exemple 18

```
REVOKE GRANT OPTION FOR SELECT
    ON TABLE DEPT
    FROM DUFOUR
```

Supprime la possibilité pour DUFOUR de transmettre le privilège de sélection sur la table DEPT.

3.2. Problématique de révocation

Il existe cependant quelques pièges dans l'utilisation du mécanisme de révocation. Nous allons en montrer quelques uns à l'aide de différents exemples. Rappelons simplement que celui qui lance les ordres (sauf indication contraire) est l'utilisateur DUHAMEL.

Contrairement aux droits "systèmes" les privilèges sont cumulatifs. On peut ainsi obtenir plusieurs fois le même privilège sur le même objet en provenance de différents utilisateurs. Le privilège sera totalement retiré lorsque tous les utilisateurs ayant donné ce privilège l'auront retiré.

Exemple 19

```
GRANT SELECT
 ON EMP
 TO DUCROS WITH GRANT OPTION
```

```
GRANT SELECT
 ON EMP
 TO DUGLAND
```

C'est maintenant DUCROS qui est l'utilisateur qui va lancer l'ordre suivant :

```
GRANT SELECT
ON EMP
TO DUGLAND
```

Enfin, DUHAMEL reprend la main pour révoquer ainsi :

```
REVOKE SELECT
ON EMP
FROM DUGLAND
```

DUGLAND peut-il sélectionner des lignes de la table EMP? La réponse est OUI, par ce qu'il possède encore un droit de sélection venant de DUCROS !
Voici une autre problématique. Le super utilisateur PUBLIC ne vise personne en particulier ni en général. On ne peut donc retirer un privilège particulier à un utilisateur donné même si l'on a attribué des privilèges à "PUBLIC".

Exemple 20

```
GRANT SELECT
  ON T_CLIENT
  TO PUBLIC
```

```
REVOKE INSERT, DELETE
  ON EMP
  FROM DUMOULIN
```

Ce dernier ordre SQL va retourner un message d'erreur et ne sera pas exécuté parce que DUMOULIN n'a jamais reçu les privilèges INSERT et DELETE sur la table CLIENT, bien qu'il hérite des privilèges de PUBLIC !
En fait il faut comprendre que le modèle de gestion des privilèges dans SQL repose sur la théorie des graphes et peut devenir vite compliqué lorsque l'on veut gérer finement de multiples droits.

4. Retrouver les privilèges

Les vues d'information de schema permettent de retrouver les privilèges, les objets et les utilisateur visés (originaires et destinataires). Pour cela la norme SQL 2 à prévue 3 vues spécifiques :

INFORMATION_SCHEMA	Colonnes
TABLE_PRIVILEGES	GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, PRIVILEGE_TYPE, IS_GRANTABLE
COLUMN_PRIVILEGES	GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, PRIVILEGE_TYPE, IS_GRANTABLE

Exemple 21

```
SELECT GRANTOR, GRANTEE, TABLE_NAME, PRIVILEGE_TYPE, IS_GRANTABLE
FROM   INFORMATION_SCHEMA.TABLE_PRIVILEGES
WHERE  GRANTEE IN ('MILLER', 'SCOTT', 'FORD')
UNION
SELECT GRANTOR, GRANTEE, TABLE_NAME, COLUMN_NAME, PRIVILEGE_TYPE, IS_GRANTABLE
FROM   INFORMATION_SCHEMA.COLUMN_PRIVILEGES
WHERE  GRANTEE IN ('MILLER', 'SCOTT', 'FORD')
ORDER  BY GRANTEE, TABLE_NAME, PRIVILEGE_TYPE, COLUMN_NAME
```