Langage de requêtes S.Q.L

PREMIERE PARTIE

C ’est quoi SQL?

- C’est un langage d’interrogation des bases de données relationnelles ;
- C’est un langage de requêtes ;
- Comprenant :
  - » Langage de définition de données (LDD)
  - » Langage de manipulation de données (LMD)
  - » Langage de contrôle de données (LCD)

Modèle relationnel

Pour la gestion des notes des étudiants de la Licence 3 Informatique, on considère le modèle logique de données suivant :

- Etudiants(Code_Etudiant, nom_Etudiant, Pre_Etutiant,Date_Naissance)
- Matieres(Code_Matiere,Code_Professeur, Nom_Matiere, Coef_Matiere)
- Notes(Code_Etudiant, Code_Matiere, Note_CC1,Note_CC2)
- Professeurs(Code_Professeur, Nom_professeur, Prenom_Professeur)

On suppose la lites des enregistrements suivants :

Table Etudiants :

C_E	N_E	P_E	V_E
E1	Alaoui	Karim	Tanger
E2	Chakour	Rachide	Tétouan
E3	Daoud	Noufal	Fès
E4	Kalouch	Ouard	Tanger

Table Professeurs :

C_P	N_P	P_P
P1	Chaoui	Adnan
P2	Banani	Adil
P3	Samoud	Koutar
P4	Zine	Monir
P5	Daouda	Kirima

Table Matiere :

C_M	N_M	C_P	Coef
M1	BDD	P1	2
M2	Prog	P4	1
M3	Reseau	P1	3
M4	Comp	P2	1
M5	Arch	P3	3

Table Notes :

Code_Mat	Code_Etud	Note_CC1	Note_CC2
M1	E1	12	-
M2	E1	6	12
M3	E1	14	12
M1	E2	14	10
M2	E2	2	13
M3	E2	3	14
M1	E3	2	-
M2	E3	1	15
M3	E3	14	12
M1	E4	16	13
M2	E4	1	-
M3	E5	2	9
M1	E5	16	-

Modèle relationnel

S.Q.L

- **Les expression de table (Clause).**
  - **FROM :**
    - » Spécifie les noms des tables sur lesquelles on va effectuer l'opération.
  - **WHERE :**
    - » Spécifie la condition de recherche qui permet de filtrer les enregistrements concernés par l'opération
  - **GROUPE BY :**
    - » Sépare les lignes choisies en groupes selon un ou plusieurs champs.
  - **HAVING :**
    - » Spécifie la condition qui doit être satisfaite par le groupe recherché.
  - **ORDER BY :**
    - » Spécifie l'ordre d'affichage croissant et décroissant.

Modèle relationnel

S.Q.L

- Commandes

  - **CREATE TABLE, ALTER TABLE, CREATE VIEW, CREATE INDEX,**
  - **SELECT, DELETE, DROPE, INSERT, UPDATE,COMMENT,**
  - **GRANT, REVOKE, ROLLBACK, COMMIT,...**

Modèle relationnel

S.Q.L

- Les Mots clés
  - Begin, By, Close, Create, Float, Select,
  - Declare, Where, AVG, All, Union, Update...
- Les types de données
  - Caracter, Numeric, Decimal, Integer
  - Smallint, Float, Real, Doubleprecision.

Langage SQL!!!

Langage de Définition de données

- **Création d'une base de données**
  - Syntaxe :
    - **CREATE DATABASE** Nom_Base_données;
  - Exemple :
    - **CREATE DATABASE** Gestion_Notes;

    // Création d'un base de donnée qui pote le nom

    // Gestion_Notes.

Langage SQL!!!

Langage de Définition de données

- **Création des tables.**
  - Syntaxe :
    - CREATE TABLE Nom_Table

(Nom_Colonne_1 TYPE_1(Taille1) CIS_1
Nom_Colonne_2 TYPE_2(Taille2) CIS_2
….
Nom_Colonne_N TYPE_N(TailleN)) CIS_n;

Avec

Nom_colonne_i : Le nom du champs de la table Nom_Table
Type_i : Le type de données du champs i
Taille _i : La taille maximale du champs
CIS_i : contrainte d'intégrité (NULL, UNIQUE)

Langage SQL!!!

Type	Description
CHARACTER	Chaîne de caractères de longueur connue
NUMERIC	La précision et l'échelle sont spécifiés
DECIMAL	L'échelle et spécifiée et la précision donnée
INTEGER	Précision donnée par l'utilisateur et l'échelle 0.
SMALLINT	Échelle 0 et la précision inférieur a celle de INT
FLOAT	Précision binaire supérieur ou égale à la valeur fourni par l'utilisateur
REAL	Précision définie par l'utilisateur
DOUBLE PRECISION	Précision donnée par l'utilisateur, supérieure à celle de réel

Langage SQL!!!

Langage de Définition de données

- **Exemple de création des tables.**
  - Syntaxe :
    - **CREATE TABLE Etudiants**

```
(Code_Etudiant      INT(10)      UNIQUE ,NOT NULL
 Nom_Etudiant       CHAR(20)   NOT NULL
 Prenom_Etudiant    CHAR(40)
 Ville_Etudiant      CHAR(100));
```

Langage SQL!!!

Langage de Définition de données

- **Clause CONSTRAINT**
  - **Les contraintes sont utilisées pour restreindre les valeurs qui peuvent être ajoutées à la table.**
    - **Index primaires**
    - **Indexe étrangères**
    - **Unicité et la valeurs non NULL**
    - **Règles de validation supplémentaires (CHECK)**

Langage SQL!!!

Langage de Définition de données

- La clause CONSTRAINT peut prendre deux formes :
  - **Cas d'une colonne**
    - **CONSTRAINTE nom { PRIMARY KEY | UNIQUE | REFERENCES Table_Externe [colonne_Externe)]}**
  - **Cas multicolonne**
    - **CONSTRAINTE nom { PRIMARY KEY (Colonne1 [, Colonne2, [,…]])| UNIQUE | REFERENCES Table_Externe [(colonne_Externe1 [,colonne_Externe2 [,…]])]}**

Langage SQL!!!

Langage de Définition de données

- Création de la table « Etudiant »
  - **CREATE TABLE Etudiant**

    (Cod_Etudiant LONG CONSTRAINT PRIMARY KEY, Nom_Etudiant Text (20), Pre_Etudiant TEXT (20), Date_Naissance DATETIME)

Langage SQL!!!

Langage de Définition de données

- Création de la table « Professeur »
  - **CREATE TABLE Professeur (Cod_Professeur LONG CONSTRAINT Cle_primaire PRIMARY KEY UNIQUE NOT NULL, Nom_ProfesseurText (20) CONSTRAINT nn NOT NULL, Pre_Professeur TEXT (20))**

Langage SQL!!!

Langage de Définition de données

- Création de la table « Matiere »
  - **CREATE TABLE Matiere**

    (Cod_Matiere LONG CONSTRAINT Cle_primaire PRIMARY KEY, Nom_Matiere Text (20), Coef_MatiereLONG, Cod_Professeur LONG CONSTRAINTE Ref_Professeur REFERENCES Professeur)

Langage SQL!!!

Langage de Définition de données

- Création de la table « Notes »
  - **CREATE TABLE Notes**

    (Code_etudiant LONG CONSTRAINT Ref_Etudiant REFERENCES Etudiant, Cod_Matiere LONG CONSTRAINT Ref_Matiere REFERFENCES Matiere,CC_1 Integer, CC_2 Integer, CONSTRAINT Cle_primaire PRIMARY KEY (Code_etudiant, Cod_Matiere);

Langage SQL!!!

Default & Check … Constrainte

- **Default :** *Permet d'initialiser le champs par une valeur par défaut.*
- **Check … Constrainte :** *Permet de créer des règles de validation pour une table, en se propageant sur plusieurs tables*

Langage SQL!!!

Default & Check … Constrainte

- **CREATE TABLE Notes (Code_etudiant LONG CONSTRAINT Ref_Etudiant REFERENCES Etudiant, Cod_Matiere LONG CONSTRAINT Ref_Matiere REFERFENCES Matiere,CC_1 Integer DEFAULT 0, CC_2 Integer DEFAULT 0, CONSTRAINTE Cle_primaire PRIMARY KEY (Code_etudiant , Cod_Matiere);**

Langage SQL!!!

Default & Check … Constrainte

Exemple : ***Création de la table*** **Prime_Etudiant** ***si la moyenne est la plus grande note de la classe.***

- **CREATE TABLE Prime_Etudiant (Cod_Etudiant LONG CONSTRAINT Cle_primaire PRIMARY KEY, Nom_Etudiant Text (20), Pre_Etudiant TEXT (20), Date_Naissance DATETIME, Prime LONG, Moyenne Long, CONSTRAINT PrimeContrainte CHECK (Moyenne >=(Selectionner le Max))**

Langage SQL!!!

Intégrité référentielle en cascade

- La création des contraintes d'intégrité référentielle en cascade.

CONSTRAINT FORIGN KEY
(Colonne1[Colonne2 [,….]])

REFERENCES Table_Etrangère
[(Colonne_Etr1, [, Colonne_Etr2 [,…]])]

[ON UPDATE (NO ACTION | CASCADE)]

[ON DELETE (NO ACTION | CASCADE)]

Langage SQL!!!

Intégrité référentielle en cascade

- Si spécifiez ON UPDATE NO ACTION, ou si vous n'incluez pas le mot clé NO UPDATE, vous ne pourrez pas changer la valeur de la clé primaire si elle est référencée par un ou plusieurs enregistrements de la table étrangère, si vous précisez CASCADE, la nouvelle valeur de clé primaire sera calculée à partir des enregistrement référencés dans la table étrangère.

Langage SQL!!!

Intégrité référentielle en cascade

Si vous spécifiez ON DELETE NO ACTION, ou si le mot clé NO DELETE n'est pas présent, vous ne pourrez pas supprimer d'enregistrement dans la table principale si ceux-ci sont référencés dans une ou plusieurs lignes de la table étrangère.

Avec l'ajout de CASCADE, les enregistrements référencés seront aussi supprimés dans la table étrangère.

Langage SQL!!!

Modification des tables.

- Syntaxe :
  - Ajout d'une colonne
    - **ALTER TABLE Nom_table**

    ADD Nom_Colonne Type(Taille) [CONSTRAINT Contraite];
  - Modification de la taille d'une colonne
    - **ALTER TABLE Nom_table**

    MODIFY Nom_Colonne Type (Nouvelle_Taille);
  - Suppression d'un colonne
    - **ALTER TABLE Nom_table**

    DROP Nom_Colonne;

- Remarque :
  - Dans le cas de modification de la taille d'une colonne il faut que la table soit vide si la nouvelle taille est inférieure à la précédente.

Langage SQL!!!

Langage de Définition de données

- **Modification des tables.**
  - Ajout de la colonne '**Ville'** dans la table **Etudiants**
    - ✓ **ALTER TABLE Etudiants**

    ADD Ville CHAR(40) CONSTRAINT NOT NULL;
  - Modification de la taille de la colonne **Nom_Client**
    - ✓ **ALTER TABLE Etidiants**

    MODIFY Nom_Etudiant CHAR (30);
  - Suppression de la colonne ville de la table C**lient**
    - ✓ **ALTER TABLE Etidiants**

    DROP Ville;

Langage SQL!!!

Langage de Définition de données

- **Suppression des contraintes.**
  - Suppression d'une contrainte XXX sur la table YYYY
    - ✓ **ALTER TABLE YYYY**

      DROP CONSTRAINT XXX;
    - ✓ **Exemple :**

      ALTER TABLE Etudiants

      DROP CONSTRAINT Cle_primaire;

Langage SQL!!!

Langage de Définition de données

- **Suppression des tables.**
  - Syntaxe
    - **DROP TABLE Nom_Table;**
  - Exemple
    - **DROP TABLE Etudiants;**

- **Remarques :**

  - Avant la suppression il faut vérifier les contraintes référentielles avec d'autres tables

Langage SQL!!!

Langage de Définition de données

- **Création et suppression des index.**
  - CREAT INDEX Nom_Index
    ON Nom_Table
  - DROP INDEX Nom_Index
    ON Nom_Table

- **Remarque**

Si vous avez des index de noms identiques, la partie '<u>ON</u>' doit être utilisée. Si le nom de l'index est unique on peut utiliser la syntaxe suivante :

DROP INDEX Nom_Index

Langage SQL!!!

Langage de Définition de données

- **Syntaxe générale de création des index.**
  - CREATE [UNIQUE] INDEX Non_index ON Non_Table (Colonne1[, Colonne2, [….]])
    [WITH (PRIMARY | DISALLOW NULL | IGNORE NULL)]

- **UNIQUE : Les doublons ne sont autorisés dans l'index**
- **PRIMARY : Toutes les clés primaires sont automatiquement définies sans doublons**
- **IGNOR NULL : Ne pas créer d'entrées d'index pour les valeurs vide**
- **DISALLOW NULL : Interdiction toute valeur vide dans le colonne**

Langage SQL!!!

Langage de Manipulation de données

- **Insertion des enregistrements dans la table.**
  - INSERT INTO Nom_Table
    (Col1, Col2, Col3,…)
    VALUES (Val1,Val2,Val3,….)
- **Remarques**

1. Si la liste des valeurs est dans le même ordre que la liste des colonnes de la tables et s'il il y'a des valeurs pour chaque colonne dans la tables, alors la liste de nom de colonnes peut être omise.
2. Les valeurs insérées doivent s'accorder au type de la colonne dans laquelle elles sont insérées.
3. Les valeur de type CHAR doivent être entourées d'apostrophes.

Langage SQL!!!

Langage de Manipulation de données

- **Exemples**

- **Insertion de l'étudiant (10, 'Touré','Ahmed','Bouna') dans la table Etudiants.**

```
INSERT INTO      Etudiant
            VALUES (10,'Touré','Ahmed','Bouna');
```

- **Insertion de l'étudiant (12,'Touré', ,'Bouna') dans la table Etudiants.**

```
INSERT INTO      Etudiant
                 (Code_etudiant,Nom_etudiant, ,Ville)
            VALUES (12,'Touré', ,'Bouna');
```

- **Insertion de tous les étudiants de la ville de Tanger dans la table EtudiantsTetouan existante.**

```
INSERT  INTO EtudiantsTetouan
            Select  Code_etudiant,Nom_etudiant,Pre_etudiant ,Ville
            From Etudiant
             Where ville = 'Bouna';
```

Langage SQL!!!

Langage de Manipulation de données

- **Insertion des enregistrements dans une nouvelle table.**
  - SELECT INTO Nouvelle_Table FROM Liste_Tables [WHERE Condition] [ORDER BY Ordre];

Exemple:

Insertion de tous les étudiants de la ville de Bouna dans une nouvelle table EtudiantsBouna.

```
SELECT Code_Etudiant, Nom_Etudiant INTO EtudiantsBouna
  FROM Etudiant
 Where Ville = 'Bouna';
```

Langage SQL!!!

Langage de Manipulation de données

- **Exemples**

- **Insertion de l'étudiant (10, 'Touré','Ahmed','Bouna') dans la table Etudiants.**

```
INSERT INTO      Etudiant
            VALUES (10,'Touré','Ahmed','Bouna');
```

- **Insertion del'étudiant (12,'Touré', ,'Bouna') dans la table Etudiants.**

```
INSERT INTO      Etudiant
                 (Code_etudiant,Nom_etudiant, ,Ville)
            VALUES (12,'Touré', ,'Bouna');
```

- **Insertion de tous les étudiants de la ville de Tanger dans la table EtudiantsTetouan.**

```
INSERT  INTO EtudiantsBouna
            Select  Code_etudiant,Nom_etudiant,Pre_etudiant ,Ville
            From Etudiant
             Where ville = 'Bouna';
```

Langage SQL!!!

Langage de Manipulation de données

- **Mise à jour**

```
UPDATE   Nom_Table
SET      Colonne1  =  Exp1,
         Colonne2 =  Exp2,
          ……
         ColonneN = ExpN1,
WHERE    Condition;
```

- **Remarque**

1. **La clause SET indique quelles sont les colonnes à mettre à jour et quelle sont les nouvelle valeurs**
2. **La mise a jour des colonnes ne peur être lieu que si la condition ‘Condition’ est valide**
3. **‘Expi’ c’est une expression dont l’évaluation donne une valeur de même type que celui de la colonne i**

Langage SQL!!!

Langage de Manipulation de données

- **Exemple**

- **Augmenter la note de CC1 pour que la matière de BDD soit validée, si la moyenne compris entre 9,5 et 10.**

```
UPDATE   Notes
SET      CC1 = 10
WHERE    Code_Matiere='BDD' and
         (cc1+cc2)/2<10 and cc1+cc2)/2>=9,5;
```

- **Augmenter la note de CC1 et CC2 de la matière BDD pour chaque étudiant.**

```
UPDATE   Notes
SET      CC1 = CC1+1,
         CC2 = CC2+1,
WHERE    Code_Matiere='BDD';
```

Langage SQL!!!

Langage de Manipulation de données

- **Suppression des enregistrements**

```
DELETE
FROM    Nom_Table
WHERE   Condition;
```

- **Remarque**

1. **Les enregistrements supprimés sont ceux qui vérifiés la condition ‘Condition’.**
2. **La condition peut être une sous interrogation d’une autre table.**

Langage SQL!!!

Langage de Manipulation de données

- **Recherche des enregistrements**

```
SELECT    Col1, Col2, … , ColN
FROM      Nom_Table
WHERE     Condition;
```

- **Remarque**

1. **Les enregistrements sélectionnés sont ceux qui vérifiés la condition ‘Condition’.**
2. **La condition peut être une sous interrogation d’une autre table.**

Langage SQL!!!

Langage de Manipulation de données

- **Supprimer toutes les matières de la table 'Notes' dont la moyenne est inférieur à 3.**

```
DELETE
FROM    Notes
WHERE   (Note_CC1 + Note_CC2 )/2<3;
```

- **Sélectionner tous les étudiants dont la moyenne de la matière BDD est inférieur à 3.**

```
DELETE  Code_Etudiant, CC1, CC2
FROM    Notes
WHERE   (CC1 + CC2 )/2<3 and Code_Matiere ='BDD';
```

LABEL

```
SELECT  Code_Professeur,  Nom_Professeur,Nom_Matiere,  Coef
FROM    Matieres, Professeurs;
```

Remarques 1:

Le champs code_professeur appartient a la table ‘Matieres’ et ‘Professeurs’ il faut donc le précède par le nom de la table mère :

Professeurs.Code_Professeurs

Remarques 2:

Pour inviter réécriture de la totalité du nom d’une table, on peut utiliser un **Label** pour chaque nom de la table dans la clause FROM.

FROM Profeseseurs P, Matiere M

LABEL

SELECT P.Code_Professeur, Nom_Professeur,Nom_Matiere, Coef
FROM Matieres M, Professeurs P;

Table Professeurs:

Table Matieres:

Code_Prof	Nom_Prof	Nom_Mat	Coef
P1	**Chaoui**	M1	0.2 P1
P1	**Chaoui**	M2	0.1 P4
P1	**Chaoui**	M3	0.3 P1
P2	**Banani**	M1	0.2
P2	**Banani**	M2	0.1
P2	**Banani**	M3	0.3

Résultat

P1	**Chaoui**	**Adnan**
P2	**Banani**	**Adil**

M1	**BDD**	**P1**	**0.2**
M2	**Prog**	**P4**	**0.1**
M3	**Reseau**	**P1**	**0.3**

FROM & WHERE

```
SELECT P.Code_Professeur, Nom_Professeur,Nom_Matiere, Coef
FROM Matieres M, Professeurs P
WHERE P.Code_Professeur = M.Code_Professeur;
```

Table Professeurs :

P1	**Chaoui**	**Adnan**
P2	**Banani**	**Adil**

Table Matière

M1	**BDD**	**P1**	**0.2**
M2	**Prog**	**P4**	**0.1**
M3	**Reseau**	**P1**	**0.3**

Code_Prof	Nom_Prof	Nom_Mat	Coef
P1	**Chaoui**	M1	0.2
P1	**Chaoui**	M2	0.1
P1	**Chaoui**	M3	0.3
P2	**Banani**	M1	0.2
P2	**Banani**	M2	0.1
P2	**Banani**	M3	0.3

ALL/DISTINCT/DISTINCTROW

Prédicat de qualification, permettant de contrôler la façon dont les valeurs et les enregistrements dupliquées sont gérés.

ALL : est prise par défaut si aucun mot clé n'est spécifié, il retourne toutes les rangées qui vérifient le critère.

Distinct : Elimine toutes les lignes en doubles dans le résultat en se basant sur les colonnes dans la clause SELECT.

Distinctrow : Elimine toutes les lignes en doubles dans le résultat en se basant sur l'ensembles des colonnes des tables sources.

ALL

Exemple : Lister tous les professeurs qui enseigne au mois une matière.

```
SELECT  P.Code_Professeur, Nom_Professeur,Pre_professeur
FROM    Matieres M, Professeurs P
WHERE   P.Code_Professeur = M.Code_Professeur;
```

Table Professeurs :

P1	**Chaoui**	**Adnan**
P2	**Banani**	**Adil**

Table Matière

M1	**BDD**	**P1**	**0.2**
M2	**Prog**	**P2**	**0.1**
M3	**Reseau**	**P1**	**0.3**

Code_Prof	Nom_Prof	Pre_professeure
P1	**Chaoui**	**Adnan**
P1	**Chaoui**	**Adnan**
P2	**Banani**	**Adil**

Distinct

Remarque : Si un professeur enseigne deux matières, sera affiché plusieurs fois. Deux ligne porte les même informations

```
SELECT DISTINCT  P.Code_Professeur, Nom_Professeur,
                 Pre_professeur
FROM    Matieres M, Professeurs P
WHERE   P.Code_Professeur = M.Code_Professeur;
```

Table Matière

Table Professeurs :

Code_Prof	Nom_Prof	Pre_professeure
P1	**Chaoui**	**Adnan**
P2	**Banani**	**Adil**

M1	**BDD**	**P1**	**0.2**
M2	**Prog**	**P2**	**0.1**
M3	**Reseau**	**P1**	**0.3**

P1	**Chaoui**	**Adnan**
P2	**Banani**	**Adil**

DistinctRow

Remarque : Si un professeur enseigne deux matières,
sera affiché plusieurs fois, Mais les matières enseignées sont différentes

```
SELECT DISTINCTROW  P.Code_Professeur, Nom_Professeur, Pre_professeur
FROM    Matieres        M,        Professeurs        P
WHERE   P.Code_Professeur = M.Code_Professeur;
```

Table Professeurs :

P1	**Chaoui**	**Adnan**
P2	**Banani**	**Adil**

Table Matière

M1	**BDD**	**P1**	**0.2**
M2	**Prog**	**P2**	**0.1**
M3	**Reseau**	**P1**	**0.3**

Code_Prof	Nom_Prof	Pre_professeure
P1	**Chaoui**	**Adnan**
P1	**Chaoui**	**Adnan**
P2	**Banani**	**Adil**

TOP

Le prédicat permet de limiter la liste retournée aux n lignes du haut, ou encore aux n [PERCENT] de ligne haut.

```
SELECT TOP 2   P.Code_Professeur, Nom_Professeur,
                        Pre_professeur
FROM    Matieres M, Professeurs P
WHERE   P.Code_Professeur = M.Code_Professeur;
```

Table Professeurs :

P1	**Chaoui**	**Adnan**
P2	**Banani**	**Adil**

Table Matière

M1	**BDD**	**P1**	**0.2**
M2	**Prog**	**P2**	**0.1**
M3	**Reseau**	**P1**	**0.3**

Code_Prof	Nom_Prof	Pre_professeur
P1	**Chaoui**	**Adnan**
P2	**Banani**	**Adil**

Jointure

Inclure tous les enregistrements pour lesquels les champs joints sont égaux.

```
SELECT P.Code_Professeur, Nom_Professeur,Code_Mat
FROM Matieres M, Professeurs P
WHERE Professeur JOIN Matiere ON
[P].[code_Professeur]=[M].[code_Professeur];
```

Table Professeurs :

P1	**Chaoui**	**Adnan**
P2	**Banani**	**Adil**

Table Matière

M1	**BDD**	**P1**	**0.2**
M2	**Prog**	**P2**	**0.1**
M3	**Reseau**	**P3**	**0.3**

Code_Prof	Nom_Prof	Code_Matière
P1	**Chaoui**	**M1**
P2	**Banani**	**M2**

Jointure à droite

Inclure tous les enregistrements de la table mère et ceux de la table fils pour lesquels les champs joints sont égaux.

```
SELECT  P.Code_Professeur, Nom_Professeur,Code_Mat
FROM    Matieres M, Professeurs P
WHERE Professeur RIGHT JOIN Matiere ON
[P].[code_Professeur]=[Matieres].[code_Professeur];
```

Table Professeurs :

P1	**Chaoui**	**Adnan**
P2	**Banani**	**Adil**

Table Matière

M1	**BDD**	**P1**	**0.2**
M2	**Prog**	**P2**	**0.1**
M3	**Reseau**	**P3**	**0.3**

Code_Prof	Nom_Prof	Code_Matière
P1	**Chaoui**	**M1**
P2	**Banani**	**M2**
-	**P3**	**M3**

Jointure à gauche

Inclure tous les enregistrements de la table fils et ceux de la table mère pour lesquels les champs joints sont égaux.

```
SELECT  P.Code_Professeur, Nom_Professeur,Code_Mat
FROM    Matieres M, Professeurs P
WHERE Professeur LEFT JOIN Matiere ON
[P].[code_Professeur]=[Matieres].[code_Professeur];
```

Table Professeurs :

P1	**Chaoui**	**Adnan**
P2	**Banani**	**Adil**

Table Matière

M2	**Prog**	**P2**	**0.1**
M3	**Réseau**	**P3**	**0.3**

Code_Prof	Nom_Prof	Code_Matiere
P2	**Chaoui**	**-**
P3	**Banani**	**M3**
P2	**Banani**	

Jointure réflexives

La jointure réflexive est utiles dans les cas suivants :

1. Des relation récursives.
   - On suppose que les étudiants sont partagés par groupe désigné par un étudiant responsable. Pour visualiser les étudiants avec leurs responsables sur la même ligne il faut faire une jointure entre la table étudiant avec elle-même.
2. Accoler ou mettre à plat des colonnes provenant d'une même table
   - On suppose que les différentes adresse d'un étudiants sont placées sur des lignes différentes. On peut faire appel à une jointure réflexives afin de faire apparaître tous les adresse sur la même ligne

Les opérateurs arithmétiques et logiques:

Opérateur	Signification	Exemple
+	Somme	a+b
-	Différence	3124-2541
*	Produit	a*20
/	Division	a/b
AND	Et logique	a>20 AND b<30
OR	Ou logique	a<30 OR b=c
NOT	Négation logique	NOT 3 BETWEEN 1 AND 5

Les prédicats :

1. Prédicats de comparaison

Opérateur	Signification	Exemple
=	égalité	10=10; ‘ab’=‘ab’
<>	Différence	3124<>2541
<	inférieur	‘ab’<‘b’; 10<20
<=	Inférieur ou égal	‘cdcd’<=‘cdcd’
>	supérieur	12>20
>=	Supérieur ou égal	20>20
BETWEEN	Compris entre	3 BETWEEN 1 AND 5

Exemple de BETWEEN :

Affiche tous les étudiants dont la note de CC1 de base de données entre 12 et 16.

```
SELECT   Code_Etudiant, Nom_etudiant, CC1
FROM     Notes N, Etudiants E
WHERE    N.Cod_etudiant = E.Cod_Etudiant    AND
         Nom_Matiere =  'BDD'     AND
         CC1 BETWEEN 12 AND 16;
```

Remarque :

Pour extraire les informations en dehors de la borne en utilisant le terme NOT BETWEEN.