

Séance 1

SGBD I

SQL : SQLServer

Mohamed LACHGAR
Ingénieur d'Etat en Informatique
Doctorant à l'UCA

Introduction

- Base de données :
 - Collection d'informations ou de données qui existent sur une longue période de temps et qui décrivent les activités d'une ou plusieurs organisations
 - Ensemble de données modélisant les objets d'une partie du monde réel et servant de support à une application informatique
- SGBD :
 - Systèmes de Gestion de Bases de Données (DataBase Management Systems - DBMS) ensemble de logiciels systèmes permettant aux utilisateurs d'insérer, de modifier, et de rechercher efficacement des données spécifiques dans une grande masse d'informations (pouvant atteindre plusieurs milliards d'octets) partagée par de multiples utilisateurs

SGBD

- Principaux composants :
 - Système de gestion de fichiers
 - Gestionnaire de requêtes
 - Gestionnaire de transactions

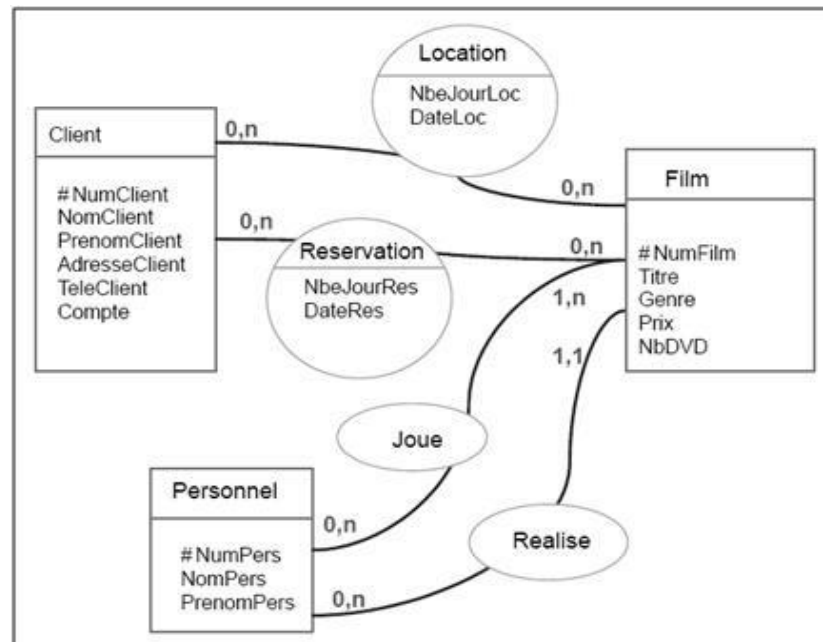
- Principales fonctionnalités :
 - Contrôle de la redondance d'information
 - Partage des données
 - Gestion des autorisations d'accès
 - Vérifications des contraintes d'intégrité
 - Sécurité et reprise sur panne

Modélisation

- Merise
 - MCD
 - MLD vs MRD
- UML
 - Diagramme de classe
- MCD vers MLD
- Diagramme de classe vers MCD
- Outil : Power AMC

Modélisation

À partir du modèle entité-association (MCD) modélisant une location de DVD ci-dessous, effectuez le passage au modèle logique :



N.B. Les champs marqués par # représentent les identifiants des entités.

Contraintes

- Contraintes d'intégrité :
toutes règles implicites ou explicites que doivent suivre les données
 - Contraintes d'entité: toute entité doit posséder un identificateur
 - Contraintes de domaine : les valeurs de certains attributs doivent être prises dans un ensemble donné
 - Contraintes d'unicité : une valeur d'attribut ne peut pas être affectée deux fois à deux entités différentes
 - Contraintes générales : règle permettant de conserver la cohérence de la base de manière générale

Exemples de contraintes

- Contraintes de domaine :

"La fonction d'un enseignant à l'Université prend sa valeur dans l'ensemble {vacataire, moniteur, ATER, MCF, Prof., PRAG, PAST}."

- Contraintes d'unicité :

"Un département, identifié par son numéro, a un nom unique (il n'y a pas deux départements de même nom)."

- Contraintes générales :

"Un même examen ne peut pas avoir lieu dans deux salles différentes à la même date et à la même heure."

SQL : Structured Query Language

- SQL2 : standard adopté en 1992
- SQL3 : extension de SQL2 avec "gestion" d'objets
- Types d'instructions SQL
 - Langage de Définition de Données (DDL) : définir le schéma de la base de données
 - Langage de Manipulation de Données (DML) : interroger et modifier les données de la base
 - Langage de contrôle d'accès aux données (DCL)

DDL : Création d'une base de données

La syntaxe permettant de créer une base de données :

CREATE DATABASE *dbname*;

La syntaxe permettant de sélectionner une base de données :

USE *dbname*;

La syntaxe permettant de supprimer une base de données :

DROP DATABASE *dbname*;

DDL : Création d'une table

```
CREATE TABLE table_name (  
    column_name1 data_type(size) constraint_name,  
    column_name2 data_type(size) constraint_name,  
    column_name3 data_type(size) constraint_name,  
    ....  
);
```

Définition des Contraintes :

```
CONSTRAINT nom_contrainte PRIMARY KEY (liste attributs clé primaire)  
| NOT NULL immédiatement après la déclaration de l'attribut  
| CHECK (condition) après la déclaration de l'attribut  
| UNIQUE après la déclaration de l'attribut  
| FOREIGN KEY (clé étrangère)  
| REFERENCES nom_table_reference (liste-colonne)
```

PRIMARY KEY = UNIQUE + NOT NULL

DESC nom_table : affiche la structure d'une table

DDL : Création d'une table : NOT NULL

```
CREATE TABLE PersonsNotNull (  
    P_Id int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)  
)
```

DDL : Création d'une table : UNIQUE

```
CREATE TABLE Persons (  
    P_Id int NOT NULL UNIQUE,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)  
)
```

DDL : Création d'une table : PRIMARY KEY

```
CREATE TABLE Persons (  
    P_Id int NOT NULL PRIMARY KEY,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)  
)
```

```
CREATE TABLE Persons (  
    P_Id int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255),  
    CONSTRAINT pk_PersonID PRIMARY KEY (P_Id, LastName)  
)
```

DDL : Création d'une table : FOREIGN KEY

Persons

P_Id	LastName	FirstName	Address	City
1	Alaoui	Hassan	Hassan 10	Marrakech
2	Safi	Amal	Salam 12	Rabat
3	Alami	Ali	Wahda 2	Agadir



O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Orders

DDL : Création d'une table : FOREIGN KEY

```
CREATE TABLE Orders (  
    O_Id int NOT NULL PRIMARY KEY,  
    OrderNo int NOT NULL,  
    P_Id int FOREIGN KEY REFERENCES Persons(P_Id)  
)
```

```
CREATE TABLE Orders (  
    O_Id int NOT NULL,  
    OrderNo int NOT NULL,  
    P_Id int,  
    PRIMARY KEY (O_Id),  
    CONSTRAINT fk_PerOrders FOREIGN KEY (P_Id)  
        REFERENCES Persons(P_Id)  
)
```

DDL : Création d'une table : CHECK

```
CREATE TABLE Persons (  
    P_Id int NOT NULL CHECK (P_Id > 0),  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)  
)
```

```
CREATE TABLE Persons(  
    P_Id int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255),  
    CONSTRAINT chk_Person CHECK (P_Id > 0 AND City = 'Marrakech')  
)
```


DDL : Création d'une table : DEFAULT

```
CREATE TABLE Persons (  
    P_Id int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255) DEFAULT 'Marrakech'  
)
```

```
CREATE TABLE Orders (  
    O_Id int NOT NULL,  
    OrderNo int NOT NULL,  
    P_Id int,  
    OrderDate date DEFAULT GETDATE()  
)
```

GETDATE() : Date système

DDL : Création d'une table

```
CREATE TABLE Enseignant (  
    Enseignant_ID integer,  
    Departement_ID integer NOT NULL,  
    Nom varchar(25) NOT NULL,  
    Prenom varchar(25) NOT NULL,  
    Grade varchar(25),  
    CONSTRAINT CK_Enseignant_Grade  
        CHECK (Grade IN ('Vacataire', 'Moniteur', 'ATER', 'MCF', 'PROF')),  
    Telephone varchar(10) DEFAULT NULL,  
    Fax varchar(10) DEFAULT NULL,  
    Email varchar(100) DEFAULT NULL,  
    CONSTRAINT PK_Enseignant PRIMARY KEY (Enseignant_ID),  
    CONSTRAINT FK_Enseignant_Departement FOREIGN KEY (Departement_ID)  
        REFERENCES Departement (Departement_ID)  
        ON UPDATE RESTRICT ON DELETE RESTRICT );
```

Contrainte de domaine

Définition de la clé primaire

Définition d'une clé étrangère

Auto incrément IDENTITY avec SQL Server & SQL LIKE

- **Créer une colonne auto incrémentée**

La création d'une colonne auto incrémentée se fait avec la propriété **IDENTITY** dont la syntaxe est la suivante :

IDENTITY [(#graine , #pas)]

ou #graine est la valeur de départ et #pas le pas d'incrément.

- **SQL LIKE** : LIKE permet d'effectuer une recherche basée plutôt sur un modèle qu'une spécification exacte de ce qui est souhaité (comme dans **IN**) ou une définition d'un intervalle (comme dans **BETWEEN**). La syntaxe est comme suit :

LIKE {modèle}

{modèle} représente souvent des caractères de remplacement. Voici quelques exemples :

- 'A_Z' : toutes les chaînes commençant par 'A', suivi d'un autre caractère, et terminant par 'Z'. Par exemple, 'ABZ' et 'A2Z' satisferaient la condition, alors 'AKKZ' ne le ferait pas (car il y a deux caractères entre A et Z au lieu d'un).
- 'ABC%' : toutes les chaînes commençant par 'ABC'. Par exemple, 'ABCD' et 'ABCABC' satisferaient la condition.
- '%XYZ' : toutes les chaînes terminant par 'XYZ'. Par exemple, 'WXYZ' et 'ZZXYZ' satisferaient la condition.
- '%AN%' : toutes les chaînes contenant le modèle 'AN' quelle que soit sa position. Par exemple, 'LOS ANGELES' et 'SAN FRANCISCO' satisferaient la condition.

DDL : Manipulation de table

- Modifier la structure d'une table créée dans la base de données.
 - Les raisons classiques de modification d'une table sont les suivantes :
 - Ajouter une colonne
 - Supprimer une colonne
 - Changer un nom de colonne
 - Changer les types de données d'une colonne
 - Ajouter une contrainte

- La syntaxe SQL est :
`ALTER TABLE "nom de table"
[alter spécifications];`

[alter spécifications] dépend du type de modification à effectuer. Pour les utilisations mentionnées ci-dessus, les instructions [alter spécification] sont :

- Ajouter une colonne : `ADD "colonne 1" "type de données pour la colonne 1"`
- Supprimer une colonne : `DROP "colonne 1"`
- Changer un nom de colonne : `CHANGE "vieux nom de colonne" "nouveau nom de colonne" "type de données pour le nouveau nom de colonne"`
- Changer le type de données d'une colonne : `MODIFY "colonne 1" "nouvelle type de données"`
- Ajouter une contrainte : `ADD CONSTRAINT "nom" "type" colonnes`

DDL : Manipulation de table

- Ajouter une colonne :

```
ALTER TABLE table_name  
ADD column_name datatype
```

- Supprimer une colonne :


```
ALTER TABLE table_name  
DROP COLUMN column_name
```

- Modifier le type de données d'une colonne (SQL Server):

```
ALTER TABLE table_name  
ALTER COLUMN column_name datatype
```


- Ajouter une contrainte (Cas UNIQUE):

```
ALTER TABLE Persons  
ADD UNIQUE (P_Id)
```



La colonne
P_Id doit être
unique

```
ALTER TABLE Persons  
ADD CONSTRAINT uc_PersonID UNIQUE (P_Id, LastName)
```



Les colonnes
P_Id et
LastName
doivent être
unique

DDL : Manipulation de table

- Ajouter une colonne :

```
ALTER TABLE table_name  
ADD column_name datatype
```

- Supprimer une colonne :


```
ALTER TABLE table_name  
DROP COLUMN column_name
```

- Modifier le type de données d'une colonne (SQL Server):

```
ALTER TABLE table_name  
ALTER COLUMN column_name datatype
```


- Ajouter une contrainte (Cas UNIQUE):

```
ALTER TABLE Persons  
ADD UNIQUE (P_Id)
```



La colonne
P_Id doit être
unique

```
ALTER TABLE Persons  
ADD CONSTRAINT uc_PersonID UNIQUE (P_Id, LastName)
```



Les colonnes
P_Id et
LastName
doivent être
unique

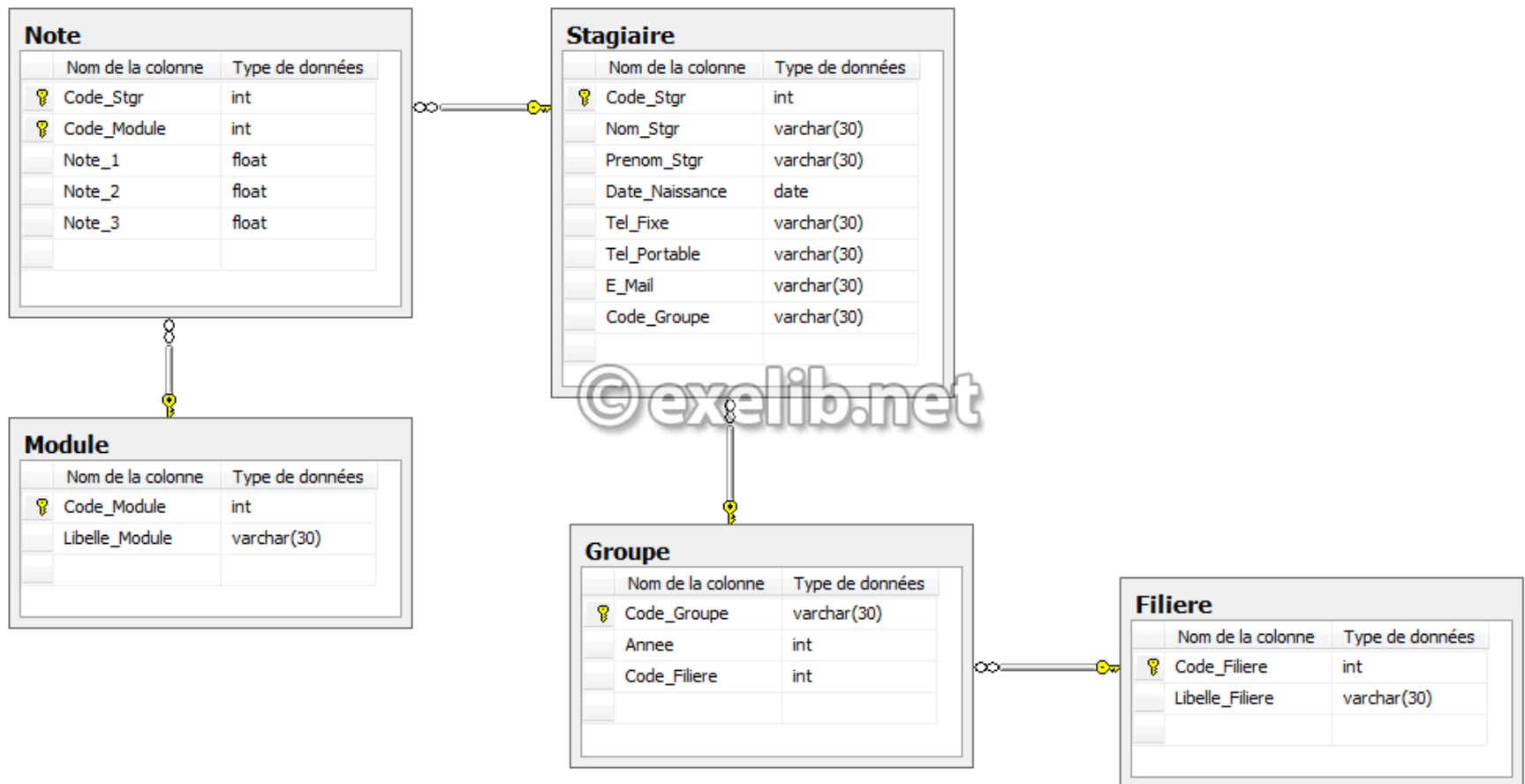
DDL : Manipulation de table

- Supprimer une table :

DROP TABLE *table_name*;

Exercice 1

Soit le schéma relationnel suivant:



Exercice 1

Créer les tables ci-dessus en précisant les clés primaires et les clés étrangères (Code_Module et Code_Filière sont générés automatiquement par le système, sauf que le premier est toujours paire et l'autre est impaire)

Ajouter les contraintes suivantes:

1. Année vaut 1 ou 2.
2. Tél_Fixe commence par 05 et Tél_Portable par 06, les deux ne peuvent pas dépasser 10 chiffres.
3. L'e-mail doit contenir @ et .
4. Les colonnes (Nom_Stgr, Prénom_Stgr, Date_Naissance) ne peuvent pas être, toutes les trois, redondantes.
5. Code_Groupe dépend de l'année, s'il s'agit d'un groupe de première année alors le code ressemble à G[un caractère entre A et H] et G[un chiffre entre 1 et 8] pour un groupe de la deuxième année.
6. Note_1, Note_2 et Note_3 sont comprises entre 0 et 20, s'elles ne sont pas remplies elles valent 0.
7. Ajouter une colonne Moyenne à la table Note qui vaut :
$$(Note_1 + Note_2 + Note_3)/3$$

Correction

- Création des tables :

```
create table Filliere(  
    code_filiere int primary key identity(1,2),  
    libelle_filiere varchar(30) not null unique);
```

```
create table Groupe (  
    code_group varchar(30) primary key,  
    annee int check (annee = 1 or annee = 2),  
    code_filiere int foreign key references filliere (code_filiere));
```

```
create table Module(  
    code_module int primary key identity (0,2),  
    libelle_module varchar(30));
```

Correction

- Création des tables :

```
create table stagiaire (  
    code_stgr int primary key identity(1,1),  
    nom_stgr varchar(30) not null,  
    prenom_stgr varchar(30) not null,  
    date_naissance date ,  
    tel_fixe varchar(30) ,  
    tel_portable varchar(30),  
    e_mail varchar(30),  
    code_groupe varchar(30));
```

- Contraintes :

```
alter table stagiaire  
    add constraint ck_tel_fixe_stagiaire  
    check (tel_fixe like '05[0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
```

```
alter table stagiaire  
    add constraint ck_tel_portable_stagiaire  
    check (tel_portable like '06_____')
```

```
alter table stagiaire  
    add constraint ck_email_stagiaire  
    check (e_mail like '%@%.%');
```

Correction

```
alter table stagiaire  
  add constraint ck_unique  
  unique(nom_stgr,prenom_stgr,date_naissance);
```

```
alter table Groupe  
  add constraint c_gr  
  check((annee=1 and code_group like 'G[A-H]')  
        or (annee=2 and code_group like 'G[1-8]'));
```

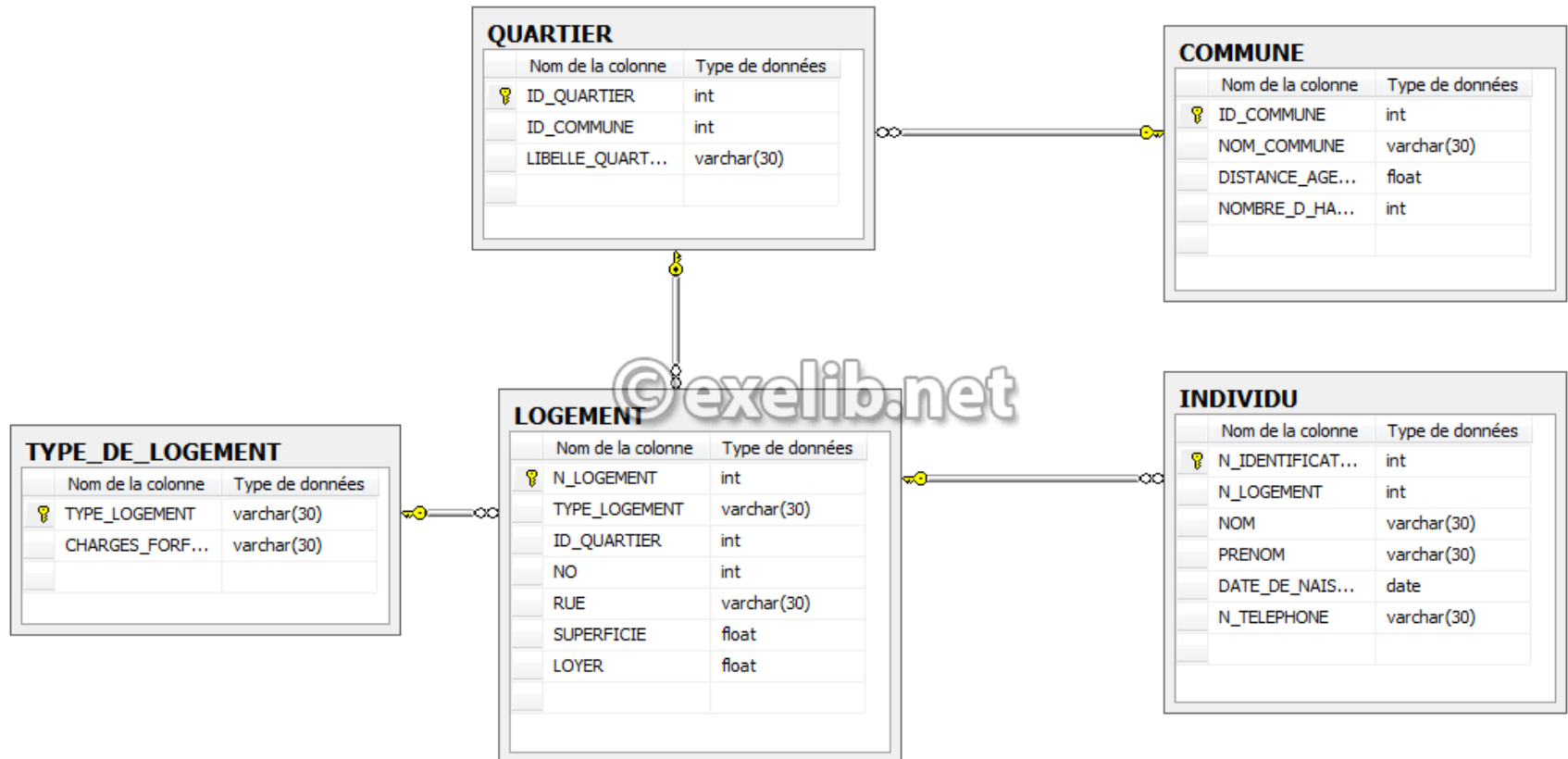
Correction

```
create table note(  
    code_stgr int ,  
    code_mod int ,  
    note1 float default 0,  
    note2 float default(0),  
    note3 float default(0),  
    constraint code_note1 check (note1 between 0 and 20),  
    constraint code_note2 check (note2 between 0 and 20),  
    constraint code_note3 check (note3 between 0 and 20),  
    primary key (code_stgr, code_mod),  
    constraint fk_note_stagiaire foreign key(code_stgr)  
        references stagiaire(code_stgr),  
    constraint fk_note_module foreign key (code_mod)  
        references Module(code_Module)  
);
```

```
alter table note  
    add moyenne as(note1 + note2 + note3)/3;
```

Exercice 2

Soit le schéma relationnel suivant:



Options des clés étrangères

- ON DELETE : permet de déterminer le comportement en cas de suppression d'une référence ;
- ON UPDATE : permet de déterminer le comportement cas de modification d'une référence.

Option de suppression :

- RESTRICT est le comportement par défaut. Si l'on essaye de supprimer une valeur référencée par une clé étrangère, l'action est avortée et on obtient une erreur. NO ACTION a exactement le même effet.
- SET NULL : NULL est substitué aux valeurs dont la référence est supprimée.
- CASCADE : Cela supprime purement et simplement toutes les lignes qui référençaient la valeur supprimée.

Option de modification :

- RESTRICT et NO ACTION : Empêche la modification si elle casse la contrainte (comportement par défaut).
- SET NULL : Met NULL partout où la valeur modifiée était référencée.
- CASCADE : Modifie également la valeur là où elle est référencée.

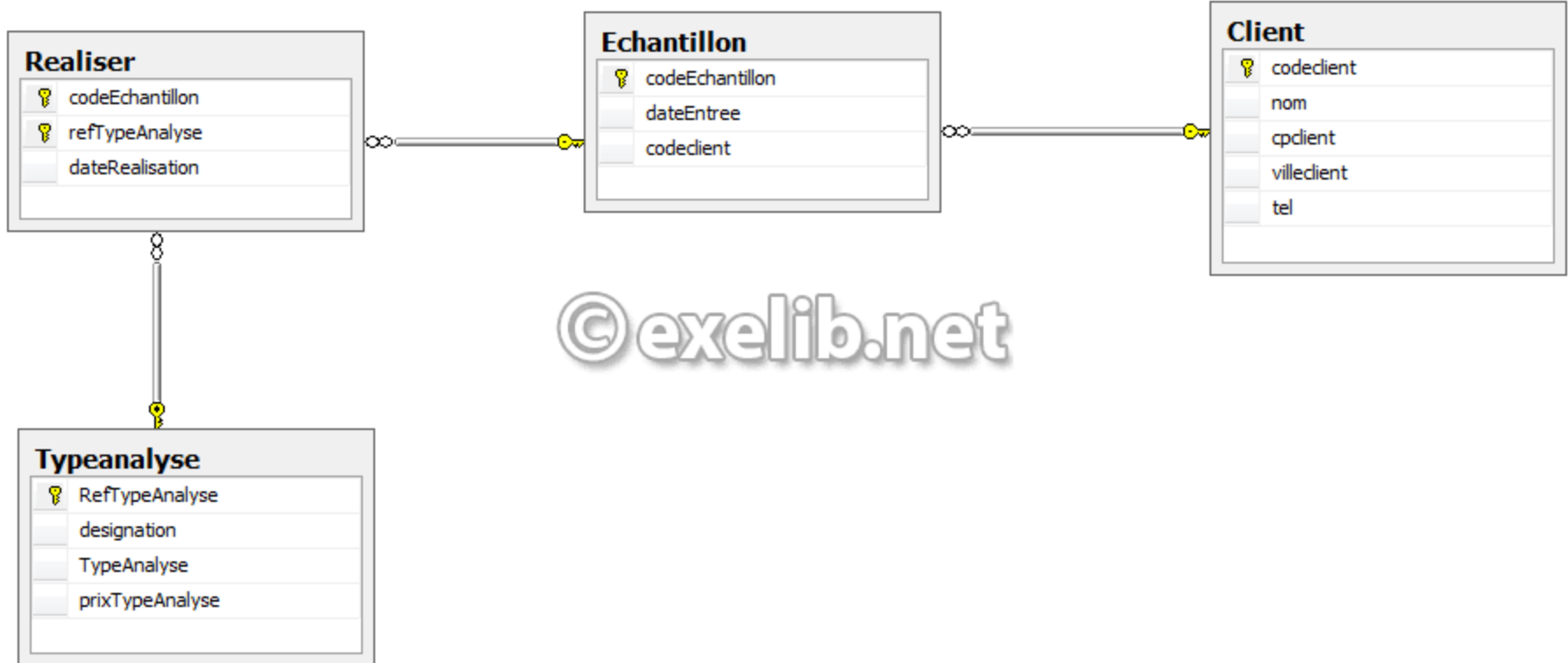
Exercice 2

Donnez les requêtes SQL permettant de réaliser les opérations suivantes :

1. Créer la base de données GestionLogement
2. Créer les cinq tables en désignant les clés primaires mais pas les clés étrangères.
3. Créer les contraintes permettant de préciser les clés étrangères avec suppression et modification en cascade.
4. Modifier la colonne N_TELEPHONE de la table INDIVIDU pour qu'elle n'accepte pas la valeur nulle.
5. Créer une contrainte df_Nom qui permet d'affecter 'SansNom' comme valeur par défaut à la colonne Nom de la table INDIVIDU.
6. Créer une contrainte ck_dateNaissance sur la colonne DATE_DE_NAISSANCE qui empêche la saisie d'une date postérieure à la date d'aujourd'hui ou si l'âge de l'individu ne dépasse pas 18 ans.
7. Supprimer la contrainte df_Nom que vous avez défini dans la question 5.

Exercice 3

Soit le modèle relationnel suivant :



Exercice 3

1. Créer la base de données ANALYSES
2. Créer la table CLIENT en précisant la clé primaire
3. Modifier les colonnes cpclient et villeclient pour qu'elles n'acceptent pas une valeur nulle.
4. Modifier les colonnes Nom pour qu'elle prend la valeur 'Anonyme' par défaut.
5. Créer la table Echantillon en précisant la clé primaire qui commence de 10 et s'incrémente automatiquement de 1, codeclient est la clé étrangère vers la table Client.
6. Créer la table Typeanalyse en précisant de clé primaire.
7. Créer une contrainte ck_prixTypeAnalyse qui impose de saisir un prixTypeAnalyse dans la table Typeanalyse qui doit être entre 100 et 1000.
8. Créer la table Realiser en précisant que le couple (codeEchantillon,refTypeAnalyse) est une clé primaire, en même temps, codeEchantillon est une clé étrangère vers la table Echantillon et refTypeAnalyse est clé étrangère vers la table TypeAnalyse.
9. Créer une contrainte ck_dateRealisation qui vérifie que la date de dateRealisation est entre la date du jour même et 3 jours après.
10. Supprimer la colonne rue de la table Client.

SGBD I

Séance 2

DML : Insérer les données dans une table

Il est possible d'écrire l'instruction INSERT INTO sous deux formes.

- La première forme ne précise pas les noms des colonnes où les données à insérer, seules leurs valeurs qui seront spécifiées :

```
INSERT INTO table_name  
VALUES (value1,value2,value3,...);
```

- La seconde forme spécifie les deux : noms de colonnes et les valeurs à insérer:

```
INSERT INTO table_name (column1,column2,column3,...)  
VALUES (value1,value2,value3,...);
```

DML : Insérer les données dans une table (Exemple 1)

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
12	Wartian Herkku	Pirkko Koskitalo	Torikatu 38	Oulu	90110	Finland
13	Wellington Importadora	Kmil Samir	Massira 23	Marrakech	40000	Maroc

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal','Tom B. Erichsen','Skagen 21','Stavanger','4006','Norway');
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
12	Wartian Herkku	Pirkko Koskitalo	Torikatu 38	Oulu	90110	Finland
13	Wellington Importadora	Kmil Samir	Massira 23	Marrakech	40000	Maroc
14	Cardinal	Tom B. Erichsen	Skagen 21	Stavanger	4006	Norway

DML : Insérer les données dans une table (Exemple 2)

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
12	Wartian Herkku	Pirkko Koskitalo	Torikatu 38	Oulu	90110	Finland
13	Wellington Importadora	Kmil Samir	Massira 23	Marrakech	40000	Maroc

```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
12	Wartian Herkku	Pirkko Koskitalo	Torikatu 38	Oulu	90110	Finland
13	Wellington Importadora	Kmil Samir	Massira 23	Marrakech	40000	Maroc
14	Cardinal	null	null	Stavanger	null	Norway

DML : Sélectionner les données à partir d'une base de données

- `SELECT column_name_1,column_name_2
FROM table_name;`

Permet de sélectionner les données dans les colonnes : column_name_1 et column_name_2 de la table « table_name ».

Exemple : `SELECT CustomerName,City FROM Customers;`

- `SELECT * FROM table_name;`

Permet de sélectionner toutes les données de la table « table_name ».

Exemple : `SELECT * FROM Customers;`

DML : Sélectionner les données à partir d'une base de données (DISTINCT)

Dans une table, une colonne peut contenir de des valeurs dupliquées; et parfois vous voulez seulement énumérer les différentes valeurs (distinctes).

Le mot-clé DISTINCT peut être utilisé pour renvoyer les valeurs distinctes.

CustomerID	Address	City
1		Berlin
2		Rabat
3		Rabat
4		London
5		Luleå

```
SELECT DISTINCT City FROM Customers;
```

Berlin
Rabat
London
Luleå

DML : Sélectionner les données à partir d'une base de données (WHERE)

- La clause WHERE est utilisée pour filtrer les enregistrements.

```
SELECT column_name, column_name  
FROM table_name  
WHERE column_name operator value;
```

- Exemples:

```
SELECT * FROM Customers  
WHERE Country='Mexico';
```

```
SELECT * FROM Customers  
WHERE CustomerID=1;
```

DML : Sélectionner les données à partir d'une base de données (WHERE)

- Les opérateurs suivants peuvent être utilisés dans la clause WHERE:

Operator	Description
=	Equal
<>	Not equal. Note: In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

DML : Sélectionner les données à partir d'une base de données (WHERE)

■ Opérateurs : AND & OR

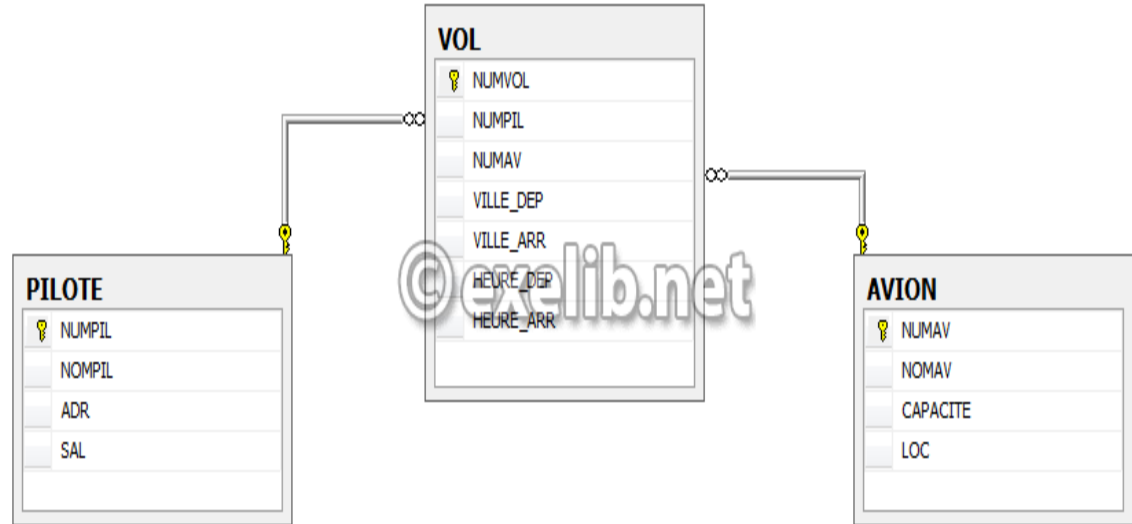
```
SELECT * FROM Customers  
WHERE Country='Morocco'  
AND City='Rabat';
```

```
SELECT * FROM Customers  
WHERE City='Rabat'  
OR City='Marrakech';
```

```
SELECT * FROM Customers  
WHERE Country='Morocco'  
AND (City='Rabat' OR City='Marrakech');
```

Exercice 4

Soit le schéma relationnel suivant:



NUMPIL: clé de PILOTE, nombre entier

NOMPIL: nom du pilote, chaîne de caractères

ADR: ville de la résidence du pilote, chaîne de caractères

SAL: salaire du pilote, nombre réel

NUMAV: clé d'AVION, nombre entier

CAPACITE: nombre de places d'un avion, nombre entier

LOC: ville de l'aéroport d'attache de l'avion, chaîne de caractères

NUMVOL: clé de VOL, nombre entier

VILLE_DEP: ville de départ du vol, chaîne de caractères

VILLE_ARR: ville d'arrivée du vol, chaîne de caractères

H_DEP: heure de départ du vol, nombre entier entre 0 et 23

H_ARR: heure d'arrivée du vol, nombre entier entre 0 et 23

Exercice 4

Ecrire les requêtes SQL répondant aux questions suivantes :

1. Donnez toutes les informations sur les pilotes de la compagnie.
2. Quels sont les numéros des pilotes en service et les villes de départ de leurs vols ?
3. Donnez la liste des avions dont la capacité est supérieure à 350 passagers.
4. Quels sont les numéros et noms des avions localisés à 'Tanger' ?
5. Dans combien de villes distinctes sont localisées des avions ?
6. Quel est le nom des pilotes domiciliés à 'Casa' dont le salaire est supérieur à 15000 DH ?
7. Quels sont les avions (numéro et nom) localisés à 'Tanger' ou dont la capacité est inférieure à 350 passagers ?
8. Liste des vols au départ de 'Rabat' allant à 'Paris' après 18 heures ?
9. Quels sont les numéros des pilotes qui ne sont pas en service ?
10. Quels sont les vols (numéro, ville de départ) effectués par les pilotes de numéro 100 et 204 ?
11. Quels sont les pilotes dont le nom commence par « S » ?
12. Quels sont les pilotes qui comportent le groupe de caractères « cie » ?
13. Quels sont les pilotes dont la troisième lettre est un « b » ?

DML : Sélectionner les données à partir d'une base de données (SELECT TOP)

- La clause SELECT TOP est utilisé pour spécifier le nombre d'enregistrements à retourner.

```
SELECT TOP number | percent column_name(s)  
FROM table_name;
```

Exemples :

- L'instruction SQL suivante sélectionne les deux premiers enregistrements de la table « Customers » :

```
SELECT TOP 2 * FROM Customers;
```
- L'instruction SQL suivante sélectionne les 50 premiers % des enregistrements de la table « Customers » :

```
SELECT TOP 50 PERCENT * FROM Customers;
```

DML : Sélectionner les données à partir d'une base de données (ORDRE BY)

- La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.

- Syntaxe :

```
SELECT colonne1, colonne2, colonne3 FROM table  
ORDER BY colonne1 DESC, colonne2 ASC
```

- Exemple :

```
SELECT nom, prenom, ville FROM client  
ORDER BY nom ASC, prenom DESC
```

Fonctions d'agrégation

- **AVG()** pour calculer la moyenne sur un ensemble d'enregistrement
- **COUNT()** pour compter le nombre d'enregistrement sur une table ou une colonne distincte **MAX()** pour récupérer la valeur maximum d'une colonne sur un ensemble de ligne. Cela s'applique à la fois pour des données numériques ou alphanumérique
- **MIN()** pour récupérer la valeur minimum de la même manière que MAX()
- **SUM()** pour calculer la somme sur un ensemble d'enregistrement

DML : Sélectionner les données à partir d'une base de données (Group By)

- Le instruction GROUP BY est utilisé en conjonction avec les fonctions d'agrégation pour regrouper les résultat par une ou plusieurs colonnes.

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;
```

- Exemple :

```
SELECT Num_Dep, Fonction_Emp, COUNT(Num_Emp) as "Nombre d'employés",
       AVG(Salaire_Emp) as "Salaire Moyen"
FROM EMPLOYE
GROUP BY Num_Dep, Fonction_Emp
```

DML : Sélectionner les données à partir d'une base de données (Having)

- La clause HAVING a été ajouté à SQL parce que le mot-clé WHERE ne peut pas être utilisé avec des fonctions d'agrégation.

- Syntaxe :

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;
```

- Exemple :

```
SELECT Fonction_Emp,AVG(Salaire_Emp) as "Salaire Moyen"
FROM EMPLOYE
GROUP BY Fonction_Emp
HAVING COUNT(*)>2
```

Les jointures

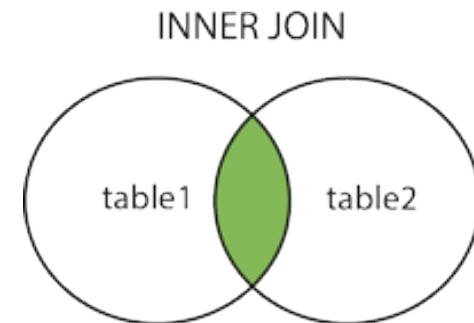
- Une clause SQL JOIN est utilisé pour combiner les lignes à partir de deux ou plusieurs tables, basé sur un champ commun entre eux.
 - INNER JOIN: Renvoie toutes les lignes quand il y a au moins une correspondance dans les deux tables
 - LEFT JOIN: Renvoyer toutes les lignes de la table de gauche, et les lignes correspondantes de la table de droite
 - RIGHT JOIN: Renvoyer toutes les lignes de la table de droite, et les lignes correspondantes de la table de gauche
 - FULL JOIN: Renvoyer toutes les lignes quand il y a une correspondance dans l'une des tables

Jointures : INNER JOIN

- Le mot-clé INNER JOIN sélectionne toutes les lignes des deux tables autant que il y a une correspondance entre les colonnes dans les deux tables.

- Syntaxe :

```
SELECT column_name(s)  
FROM table1  
INNER JOIN table2  
ON table1.column_name = table2.column_name;
```

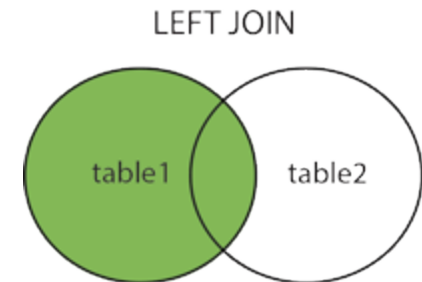


Jointures : LEFT JOIN

- Le mot-clé LEFT JOIN retourne toutes les lignes de la table de gauche (table1), avec les lignes correspondantes dans la table de droite (table2). Le résultat est NULL dans le côté droit quand il n'y a pas de correspondance.

- Syntaxe :

```
SELECT column_name(s)  
FROM table1  
LEFT JOIN table2  
ON table1.column_name=table2.column_name;
```

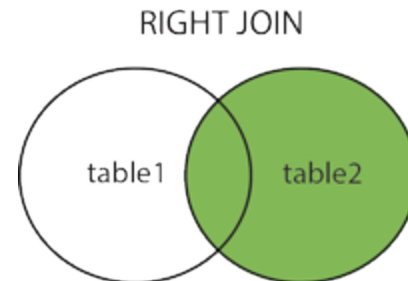


Jointures : RIGHT JOIN

- Le mot-clé RIGHT JOIN retourne toutes les lignes de la table de droite (table2), avec les lignes correspondantes dans la table gauche (table1). Le résultat est NULL dans le côté gauche quand il n'y a pas de correspondance.

- Syntaxe :

```
SELECT column_name(s)  
FROM table1  
RIGHT JOIN table2  
ON table1.column_name=table2.column_name;
```



Jointures : FULL JOIN

- Le mot-clé FULL OUTER JOIN retourne toutes les lignes de la table de gauche (table1) et de la table de droite (table2).

- Syntaxe :

```
SELECT column_name(s)  
FROM table1  
FULL OUTER JOIN table2  
ON table1.column_name=table2.column_name;
```

