

# Bases de Données

Aide-Mémoire pour la préparation de l'épreuve  
d'Informatique du Concours National d'Entrée aux  
Ecoles d'Ingénieurs MP-PC-T 2020

Elaboré par :



*Wafa Meftah (IPEIS)*

*Hayet Latrach (IPEIT)*

*Houcemeddine Filali (IPEIM)*

*Laila Hammami (IPEIN)*

*Mohamed Hammami (IPEIEM)*

*Besma Fayeche (ENSIT)*

## Préambule

*Ce document est destiné aux étudiants de deuxième année MP-PC-T des Instituts Préparatoires aux Etudes d'Ingénieurs.*

*Dans les circonstances exceptionnelles liées à la crise sanitaire mondiale et nationale du Covid19, un ensemble de supports de révision pour le programme d'informatique du concours national d'entrée aux écoles d'ingénieurs ont été élaborés essentiellement par les membres de la commission informatique du concours, suite à une action bénévole.*

*Le but de ces supports est d'accompagner les étudiants de tout le territoire Tunisien dans leur révision en leur fournissant à tous les mêmes documents, afin d'assurer une équité entre eux, qu'ils bénéficient ou non d'un enseignement ou tutorat à distance.*

*Toutefois, ces documents sont considérés comme des aide-mémoires et ne remplacent pas les cours assurés en présentiel.*

*Le document présent comporte les concepts de base associés aux bases de données relationnelles. Il expose ainsi la représentation et la manipulation des tables et relations, ainsi que l'accès à la base à travers les requêtes exprimées en algèbre relationnelle ou en SQL. L'utilisation de la bibliothèque Python SQLite3 est également présentée. Des exercices d'application avec leurs corrigés sont intégrés à la fin.*

## Table des matières

I	Introduction .....	4
II	Base de données : concepts.....	4
II.1	Eléments du Modèle Relationnel .....	4
II.2	Modèle relationnel : Clé primaire.....	4
II.3	Modèle relationnel : Clé étrangère.....	5
II.4	Exemple : Modèle relationnel de la base de données Magasin .....	5
III	Algèbre relationnelle .....	6
III.1	Projection.....	6
III.2	Restriction (sélection) .....	7
III.3	Jointure.....	7
III.4	Union .....	8
III.5	Intersection.....	9
III.6	Différence .....	9
IV	SQL ( Structured Query Language) .....	10
IV.1	Introduction .....	10
IV.2	Définition des données : LDD .....	10
IV.2.1	Création de table :.....	10
IV.2.2	Suppression / Modification de Table .....	11
IV.3	Manipulation des données : LMD.....	12
IV.3.1	Insertion des lignes .....	12
IV.3.2	Modification des données .....	13
IV.3.3	Suppression des données .....	13
IV.3.4	Extraction des données .....	13
IV.3.5	Extraction de données avec condition .....	14
IV.3.6	Expression des jointures.....	16
IV.3.7	Les sous-requêtes.....	17
IV.3.8	Les opérations ensemblistes.....	18
IV.3.9	Fonctions de calcul .....	18
IV.3.10	La clause Group By ... Having .....	19
IV.3.11	La clause Limit .....	20
IV.3.12	Forme générale de SELECT.....	20
V	SQLITE.....	20
V.1	Instructions de base sous python .....	20

[Tapez ici]

V.2	Les fonctions <code>execute()</code> et <code>executemany()</code> .....	21
V.3	Les fonctions <code>fetchone()</code> et <code>fetchall()</code> .....	22
VI	Exercices d'Application .....	23
VI.1	Exercice 1 .....	23
VI.2	Exercice 2 .....	24
VII	Corrigé des exercices.....	25
VII.1	Corrigé de l'exercice 1 .....	25
VII.2	Corrigé de l'exercice 2 .....	27
VIII	Bibliographie et Netographie .....	29

## I INTRODUCTION

Une base de données relationnelle (BDR) est une base de données où l'information est organisée selon le modèle relationnel. Les données sont organisées en tables et manipulées par des opérateurs algébriques. Dans ce qui suit, les notions de base associées aux BDR sont présentées afin de pouvoir créer et alimenter une base de données avec Python et écrire des requêtes en langage algébrique ou langage SQL.

## II BASE DE DONNEES : CONCEPTS

### II.1 Eléments du Modèle Relationnel

De façon informelle, on peut définir le modèle relationnel de la manière suivante :

- Il est composé de trois briques de base : relation, attribut, domaine.
- Les données sont organisées sous forme de tables à deux dimensions, encore appelées **relations**, dont les lignes sont appelées n-uplet ou **tuple** en anglais ;
- Les données sont manipulées par **des opérateurs de l'algèbre relationnelle** ;
- L'état cohérent de la base est défini par un ensemble de **contraintes d'intégrité**.

#### Exemple

Numéro	Nom	Prénom	Adresse	âge	département
0001	Tounsi	Mohamed	Sfax	24	Info
0236	Sfar	Mariem	Rjich	20	mécanique
2105	Hamdi	Imen	Monastir	22	biologie

*Modèle Relationnel : Exemple*

- La relation Etudiant ayant pour attributs Numéro, Nom, prénom, adresse, âge, département.
- Chaque attribut possède un domaine.

### II.2 Modèle relationnel : Clé primaire

Une clé primaire d'une relation est l'ensemble minimal des attributs de la relation dont les valeurs identifient un tuple unique, selon les propriétés suivantes :

- Une relation a au moins une clé et peut en avoir plusieurs clés : ce sont les clés candidates.
- Parmi les clés candidates, le concepteur choisit une et une seule clé primaire.
- Par convention, on représente la clé primaire en la soulignant dans l'énoncé de la relation

### Exemple

Voici les clés primaires (attributs soulignés) de quelques relations :

- Etudiant ( numero , nom , prénom , adresse, âge, département )
- Département ( nom , chef )
- Enseignant ( nom , prénom , CIN )
- Module ( réf\_module, responsable , département )

## II.3 Modèle relationnel : Clé étrangère

Une clé étrangère ou une clé secondaire est un ensemble d'attributs qui fait référence à la clé primaire d'une autre relation, selon les propriétés suivantes :

- Une clé étrangère exprime un lien obligatoire entre deux relations.
- Une relation possède une clé primaire, peut spécifier plusieurs clés étrangères.
- Convention d'écriture : *italique*

### Exemple

Voici des exemples de clés étrangères (en *italique*) de quelques relations :

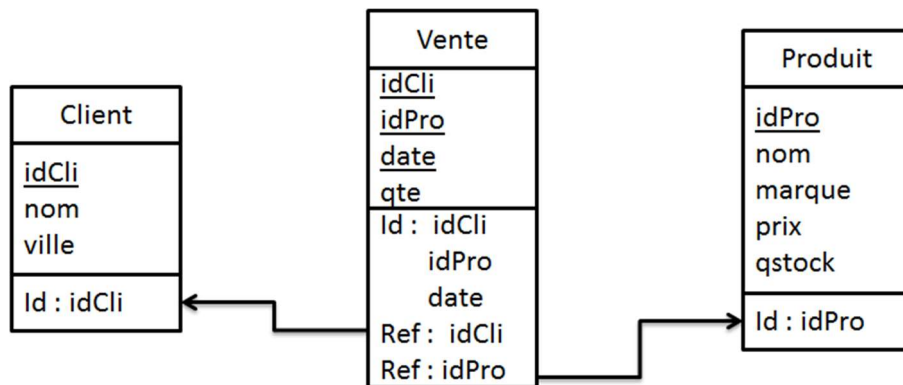
- Etudiant ( numero , nom , prénom , adresse, âge, *département* ) :  
L'attribut département est une clé étrangère qui fait référence à la clé primaire de la relation Département.
- Département ( nom , *chef* )  
L'attribut chef est une clé étrangère qui fait référence à la clé primaire de la relation Enseignant.
- Enseignant ( nom , prénom , CIN )  
La relation enseignant ne possède pas de clé étrangère.
- Module ( réf\_module, *responsable* , *département* )  
L'attribut responsable est une clé étrangère qui fait référence à la clé primaire de la relation Enseignant.  
L'attribut département est une clé étrangère qui fait référence à la clé primaire de la relation Département.

## II.4 Exemple : Modèle relationnel de la base de données Magasin

Dans la suite, nous nous basons sur un exemple d'application d'une base de données fournie pour les magasins. La conception de la BD magasin a donné trois relations principales : Client, Produit et Vente.

Ces relations sont modélisées par le schéma relationnel suivant :

- Client( IdCli, nom, ville )
- Produit( IdPro, nom, marque, prix, qstock )
- Vente ( IdCli, IdPro, date, qte )



Une représentation graphique du Schéma Relationnel de la BD 'Magasin'

### III ALGÈBRE RELATIONNELLE

L'algèbre relationnelle (AR) est un langage procédural qui permet d'indiquer comment construire une nouvelle relation à partir d'une ou plusieurs relations existantes.

C'est un langage abstrait, avec des opérations qui travaillent sur une (ou plusieurs) relation(s) pour définir une nouvelle relation sans changer la (ou les) relation(s) originale(s).

Le résultat de toute opération est une relation.

Il y a deux types d'opérations :

- Opérations spécifiques : Projection, restriction, jointure.
- Opérations ensemblistes : Union, intersection, différence ;

#### III.1 Projection

La projection d'une relation R1 est la relation R2 obtenue en supprimant les attributs de R1 non mentionnés.

On notera :  $R2 = \pi R1 (A_i, A_j, \dots, A_m)$

La projection permet d'éliminer des attributs d'une relation. Elle correspond à un découpage vertical :

A1	A2	A3	A4	A5

**Exemple :**

**Requête :** Lister les références et les prix des produits :

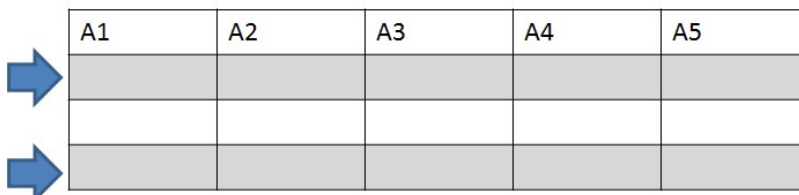
**Réponse en AR :**  $\pi \text{PRODUIT} (\text{IdPro}, \text{Prix})$

### III.2 Restriction (sélection)

La restriction d'une relation R1 est une relation R2 de même schéma n'ayant que les n-uplets de R1 répondant à la condition énoncée.

On notera :  **$R2 = \sigma R1$  (condition)**

La restriction permet d'extraire les n-uplets qui satisfont une condition.



A1	A2	A3	A4	A5

**Exemple :**

**Requête :** Lister les produits de marque 'IBM'

**Réponse en AR :**  **$\sigma \text{PRODUIT (Marque = 'IBM')}$**

### III.3 Jointure

La jointure de deux relations R1 et R2 est une relation R3 dont les n-uplets sont obtenus en concaténant les n-uplets de R1 avec ceux de R2 et en ne gardant que ceux qui vérifient la condition de liaison

On notera :  **$R3 = R1 \bowtie R2$  (condition).**

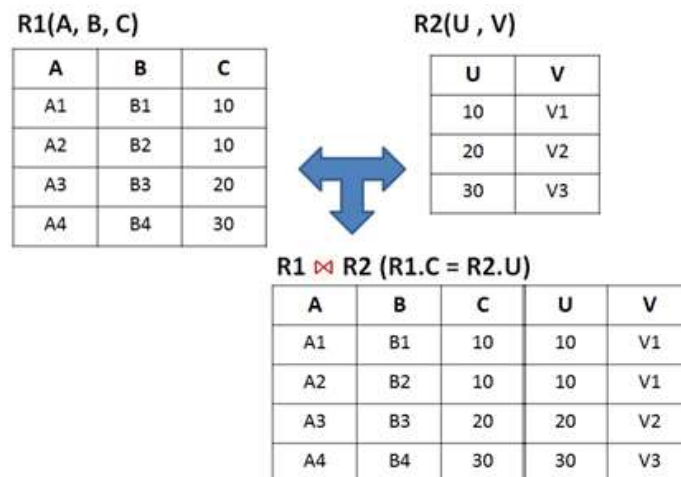
**Remarques**

Le schéma de la relation résultante de la jointure est la concaténation des schémas des opérandes (s'il y a des attributs de même nom, il faut les renommer).

Les n-uplets de  **$R1 \bowtie R2$  (condition)** sont tous les couples (u1,u2) d'un n-uplet de R1 avec un n-uplet de R2 qui satisfont "condition".

La jointure de deux relations R1 et R2 est le produit cartésien des deux relations, suivi d'une restriction.



**Exemple :**

**Requête :** Donner les identifiants des clients ayant acheté un produit de marque Apple.

**Réponse en AR :**  $R1 = \sigma_{\text{PRODUIT (marque = 'Apple')}}$

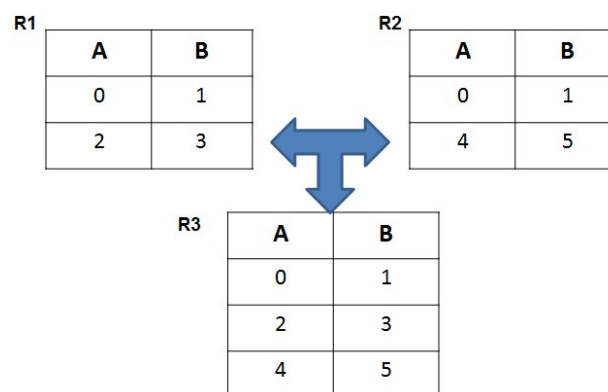
$R2 = R1 \bowtie \text{VENTE (R1.IdPro = VENTE.IdPro)}$

$\text{RESUL} = \pi_{R2} (\text{IdCli})$

**III.4 Union**

L'union de deux relations R1 et R2 de même schéma est une relation R3 de schéma identique qui a pour n-uplets les n-uplets de R1 et/ou R2.

On notera :  $R3 = R1 \cup R2$



**Requête :** Donner les identifiants des clients de Nice et les clients de Paris.

**Réponse en AR :**  $R1 = \sigma_{\text{Client (ville= 'Nice')}}$

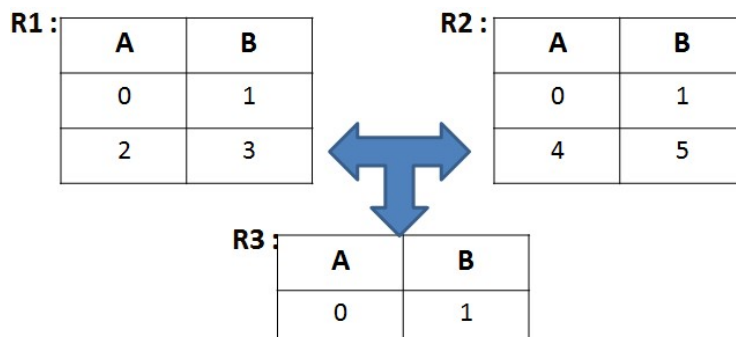
$R2 = \sigma_{\text{Client (ville='paris')}}$

$R3 = R1 \cup R2$

$\text{RESUL} = \pi_{R3} (\text{IdPro})$

### III.5 Intersection

L'intersection entre deux relations R1 et R2 de même schéma est une relation R3 de schéma identique ayant pour n-uplets les n-uplets communs à R1 et R2. On notera :  **$R3 = R1 \cap R2$** .



**Exemple :**

**Requête :** Donner les identifiants des produits de marque Apple et de prix < 2000

**Réponse en AR :**  $R1 = \sigma_{\text{PRODUIT}} (\text{marque} = \text{'Apple'})$

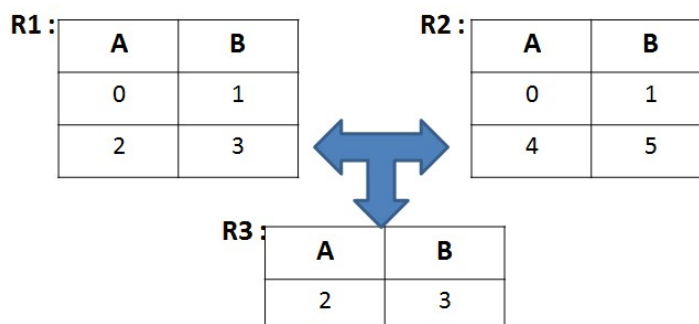
$R2 = \sigma_{\text{PRODUIT}} (\text{prix} < 2000)$

**$R3 = R1 \cap R2$**

$\text{RESUL} = \pi_{R3} (\text{IdPro})$

### III.6 Différence

La différence entre deux relations R1 et R2 de même schéma est une relation R3 de schéma identique ayant pour n-uplets les n-uplets de R1 n'appartenant pas à R2. On notera :  **$R3 = R1 - R2$** .



**Exemple :**

**Requête :** Donner les identifiants des clients n'ayant acheté que des produits de marque Apple

**Réponse en AR :**  $R1 = \text{VENTE} \bowtie \text{PRODUIT} (\text{VENTE.IdPro} = \text{PRODUIT.IdPro})$

$R2 = \sigma_{R1} (\text{marque} = \text{'Apple'})$

$R3 = \pi_{R2} (\text{IdCli})$

$R4 = \sigma_{R1} (\text{marque} \neq \text{'Apple'})$

$R5 = \pi_{R4} (\text{IdCli})$

**$\text{RESUL} = R3 - R5$**

## IV SQL ( STRUCTURED QUERY LANGUAGE)

### IV.1 Introduction

SQL est un langage standard pour accéder aux bases de données. Il est composé de plusieurs sous-langages, dont :

- Le Langage de Définition des données (LDD) pour la création et la suppression d'objets dans la base de données. Il comporte les commandes SQL : **CREATE**, **DROP**, **ALTER**.
- Le Langage de Manipulation des données (LMD) pour la recherche, l'insertion, la mise à jour et la suppression de données. Il comporte les commandes SQL : **SELECT**, **INSERT**, **UPDATE**, **DELETE**.

### IV.2 Définition des données : LDD

#### IV.2.1 Création de table :

La commande **CREATE TABLE** crée la définition d'une table selon la syntaxe suivante :

```
CREATE TABLE table (
  -- définition des colonnes
  colonne    type    [ NOT NULL [UNIQUE] ]
  [ DEFAULT valeur ]
  [ PRIMARY KEY ]
  [ REFERENCES table ]
  [ CHECK condition ] ,
  ... ,
  -- contraintes de table
  [ PRIMARY KEY (liste de colonnes) ],
  [ UNIQUE (liste de colonnes) ] ,
  [ FOREIGN KEY (liste de colonnes) REFERENCES table )
```

Types des colonnes	Contraintes d'intégrité
CHAR(n) TEXT SMALLINT INTEGER DECIMAL(n,m) : DATE BOOLEAN	NOT NULL UNIQUE PRIMARY KEY FOREIGN KEY CHECK

**Exemple**

Pour l'exemple associé au magasin (section II.4) :

- ```
CREATE TABLE client (
    IdCli CHAR(4) PRIMARY KEY ,
    nom CHAR(20) ,
    ville CHAR(30) ,
    CHECK (ville IN ('Nice', 'Paris', 'Rome', 'Tunis') ) )
```
- ```
CREATE TABLE produit (
    IdPro CHAR(6) PRIMARY KEY ,
    nom CHAR(30) NOT NULL UNIQUE ,
    marque CHAR(30) ,
    prix DEC(6,2) ,
    qstock SMALLINT CHECK (qstock BETWEEN 0 AND 100)
)
```
- ```
CREATE TABLE vente (
    IdCli CHAR(4) NOT NULL REFERENCES client ,
    IdPro CHAR(6) NOT NULL ,
    date DATE NOT NULL ,
    qte SMALLINT CHECK (qte BETWEEN 1 AND 10) ,
    -- contrainte de table
    PRIMARY KEY (IdCli, IdPro, date) ,
    FOREIGN KEY (IdPro) REFERENCES produit
)
```

***IV.2.2 Suppression / Modification de Table***

- **Suppression d'une table** : Toute table peut être supprimée à l'aide de **DROP**. Par exemple :

```
DROP TABLE Vente
```

- **Ajout d'une colonne** : Pour ajouter une colonne à une table, on utilise **ALTER**. Par exemple :

```
ALTER TABLE client ADD COLUMN tel CHAR(16)
```

A l'exécution de cette commande, la table possède une nouvelle colonne qui ne contient que des valeurs NULL pour toutes les lignes.

On ne peut ajouter une colonne obligatoire (NOT NULL) que si la table est vide ou si cette colonne possède une valeur par défaut (DEFAULT).

## IV.3 Manipulation des données : LMD

Le sous-langage **LMD** de SQL permet de consulter le contenu des tables et de les modifier. Il comporte 4 commandes :

- La requête **INSERT** insère de nouvelles lignes dans une table.
- La requête **DELETE** supprime des lignes d'une table.
- La requête **UPDATE** modifie les valeurs de colonnes de lignes existantes.
- La requête **SELECT** extrait des données des tables.

### IV.3.1 Insertion des lignes

La commande **INSERT** permet d'ajouter de nouvelles lignes à une table selon la syntaxe suivante :

```
INSERT
INTO table [ (liste de colonnes) ]
{VALUES (liste de valeurs) | requête}
```

#### Remarques :

- Dans le cas où la liste de colonnes n'est pas spécifiée tous les attributs de la table cible doivent être fournis dans l'ordre de déclaration.
- Si seulement certaines colonnes sont spécifiées, les autres sont insérées avec la valeur **NULL**.
- Une insertion à partir d'une requête permet d'insérer plusieurs lignes dans la table cible à partir d'une autre table.

#### Exemples

##### - Insertion d'une seule ligne

Pour ajouter le client ('TN100', 'Ali', Tunis) dans la table client :

```
INSERT
INTO client (IdCli, nom, ville)
VALUES ('TN100', 'Ali', 'Tunis')
```

##### - Insertion de plusieurs lignes

Pour ajouter dans une table « temp » de même schéma que la table Vente avec toutes les ventes qui sont antérieures au 01-Jan-2019 :

```
INSERT
INTO temp (IdCli, IdPro, date, qte)
      SELECT V.no_cli, V.IdPro, V.date, V.qte
      FROM vente V
      WHERE V.date < '01-jan-2019'
```

### IV.3.2 Modification des données

La commande **UPDATE** permet de changer des valeurs d'attributs des lignes existantes, selon la syntaxe suivante :

```
UPDATE table
SET liste d'affectations
[ WHERE qualification ]
```

**Remarque** : L'absence de clause WHERE signifie que les changements doivent être appliqués à toutes les lignes de la table cible.

#### Exemple

- Augmenter de 20% les prix de tous les produits :  

```
UPDATE produit
SET prix = prix * 1.2
```
- Augmenter de 50% les prix des produits de marque **hp** :  

```
UPDATE produit
SET prix = prix * 1.5
WHERE marque = 'hp'
```

### IV.3.3 Suppression des données

La commande **DELETE** permet d'enlever des lignes dans une table, selon la syntaxe suivante :

```
DELETE
FROM table
[ WHERE qualification ]
```

**Remarque** : L'absence de clause WHERE signifie que toutes les lignes de la table cible sont enlevées.

#### Exemple

Pour supprimer les ventes antérieures au 01-jan-2019 :

```
DELETE
FROM vente
WHERE date < '01-jan-2019'
```

### IV.3.4 Extraction des données

La commande **SELECT** (projection en AR) permet de rechercher des données à partir de plusieurs tables ; le résultat est présenté sous forme d'une table réponse.

#### Exemples

- **Q1** : Donner les noms, marques et prix des produits.  

```
SELECT nom, marque, prix
FROM produit
```

On peut introduire dans la clause `FROM` un **synonyme (alias)** à un nom de table, en le plaçant immédiatement après le nom de la table.

Les noms de table ou les synonymes peuvent être utilisés pour préfixer les noms de colonnes dans le `SELECT`.

```
SELECT P.nom, P.marque, P.prix
FROM produit P
```

- **Q2 :** Donner les différentes marques de produit :

```
SELECT P.marque
FROM produit P
```

Pour éliminer les doublons, il faut spécifier **DISTINCT**.

```
SELECT DISTINCT P.marque
FROM produit P
```

- **Q3 :** Donner les références des produits et leurs prix majorés de 20%.

```
SELECT P.IdPro, P.prix * 1.20
FROM produit P
```

Il est possible d'effectuer des opérations arithmétiques (+, -, \*, /) sur les colonnes extraites.

- **Q4 :** Donner tous les renseignements sur les clients.

```
SELECT *
FROM client
```

Une étoile (\*) permet de lister tous les attributs.

#### IV.3.5 Extraction de données avec condition

La condition de recherche (restriction en AR) est spécifiée après la clause **WHERE** par un prédicat. Un prédicat simple peut-être :

- un prédicat d'égalité ou d'inégalité (=, <>, <, >, <=, >=)
- un prédicat **LIKE**
- un prédicat **BETWEEN**
- un prédicat **IN**
- un test de valeur **NULL**
- un prédicat **EXISTS**

Un prédicat composé est construit à l'aide des connecteurs **AND**, **OR** et **NOT**.

- **Q5 :** Donner les noms des produits de marque IBM.

```
SELECT P.nom
FROM produit P
WHERE P.marque = 'IBM'
```

- **Q6 :** Lister les clients dont le nom comporte la lettre A en 2<sup>ème</sup> position.

```
SELECT *
FROM client C
WHERE C.nom LIKE '_A%'
```

Le prédicat **LIKE** compare une chaîne avec un modèle :

- ( ) remplace n'importe quel caractère
- (%) remplace n'importe quelle suite de caractères.

- **Q7 :** Lister les produits dont le prix est compris entre 50 dinars et 120 dinars

```
SELECT *
FROM produit P
WHERE P.prix BETWEEN 50 AND 120
```

Le prédicat **BETWEEN** teste l'appartenance à un intervalle.

- **Q8 :** Lister les produits de marque IBM, Apple ou hp.

```
SELECT *
FROM produit P
WHERE P.marque IN ('IBM', 'Apple', 'hp')
```

Le prédicat **IN** teste l'appartenance à une liste de valeurs.

- **Q9 :** Lister les produits dont le prix est inconnu.

```
SELECT *
FROM produit P
WHERE P.prix IS NULL
```

La valeur **NULL** signifie qu'une donnée est inconnue.

- **Q10 :** Lister les produits de marque IBM dont le prix est inférieur à 120 dinars

```
SELECT *
FROM produit P
WHERE P.marque = 'IBM' AND P.prix < 120
```

Le connecteur **AND** relie les 2 prédicats de comparaison.

- **Q11 :** Lister tous les produits en les triant par marques.

```
SELECT *
FROM produit P
ORDER BY P.marque
```

La clause **ORDER BY** permet de spécifier les colonnes définissant les critères de tri

L'ordre de tri est précisé par **ASC** (croissant) ou **DESC** (décroissant) ; par défaut **ASC**.

- **Q12 :** Lister tous les produits en les triant par marques et à l'intérieur d'une marque par prix décroissants.

```
SELECT *
FROM produit P
ORDER BY P.marque, P.prix DESC
```

Le tri se fera d'abord selon la première colonne spécifiée, puis selon la deuxième colonne, etc.



### IV.3.6 Expression des jointures

La jointure (jointure en AR) permet de produire une table constituée de données extraites de plusieurs tables. La condition de jointure est exprimée après **WHERE**. On peut utiliser aussi **JOIN ... ON**.

- **Q13 :** Donner les références et les noms des produits vendus.

```
SELECT P.IdPro, P.nom
FROM produit P , vente V
WHERE P.IdPro = V.IdPro
```

ou bien :

```
SELECT P.IdPro, P.nom
FROM produit P JOIN vente V
ON P.IdPro = V.IdPro
```

- **Q14 :** Donner les noms des clients qui ont acheté le produit de nom 'PS1'.

```
SELECT C.nom
FROM client C , produit P, vente V
WHERE V.IdCli = C.IdCli
AND V.IdPro = P.IdPro
AND P.nom = 'PS1'
```

ou bien :

```
SELECT C.nom
FROM VENTE V join produit P
ON V.IdPro = P.IdPro
join Client C
ON V.IdCli = C.IdCli
WHERE P.nom = 'PS1'
```

- **Q15 :** Donner les noms des clients de la même ville que Ali.

```
SELECT C2.nom
FROM client C1 JOIN client C2
ON C1.ville = C2.ville
WHERE C1.nom = 'Ali'
AND C2.nom <> 'Ali'
```

Cet exemple utilise, pour le couplage des villes, la jointure de la table Client avec elle-même (**auto-jointure**).

Pour pouvoir distinguer les références ville dans les 2 copies, il faut introduire 2 alias différents C1 et C2 de la table client.

### IV.3.7 Les sous-requêtes

SQL permet l'imbrication de **sous-requêtes** au niveau de la clause **WHERE**.

Les sous-requêtes sont utilisées :

- dans des prédicats de comparaison (=, <>, <, <=, >, >=)
- dans des prédicats **IN**
- dans des prédicats **EXISTS**

Une sous-requête dans un prédicat de **comparaison** doit se réduire à une seule valeur ("singleton select").

Une sous-requête dans un prédicat **IN** doit représenter une table à colonne unique.

L'utilisation de constructions du type "**IN sous-requête**" permet d'exprimer des jointures.

- **Q16 :** Donner les noms des clients qui ont acheté le produit d'identifiant 'SHP1'

☐ Avec sous-requête

```
SELECT C.nom
FROM client C
WHERE IdCli IN
(
    SELECT V.IdCli
    FROM vente V
    WHERE V.IdPro = 'SHP1'
)
```

☐ Avec jointure

```
SELECT C.nom
FROM client C , vente V
WHERE C.IdCli = V.IdCli
AND V.IdPro = 'SHP1'
```

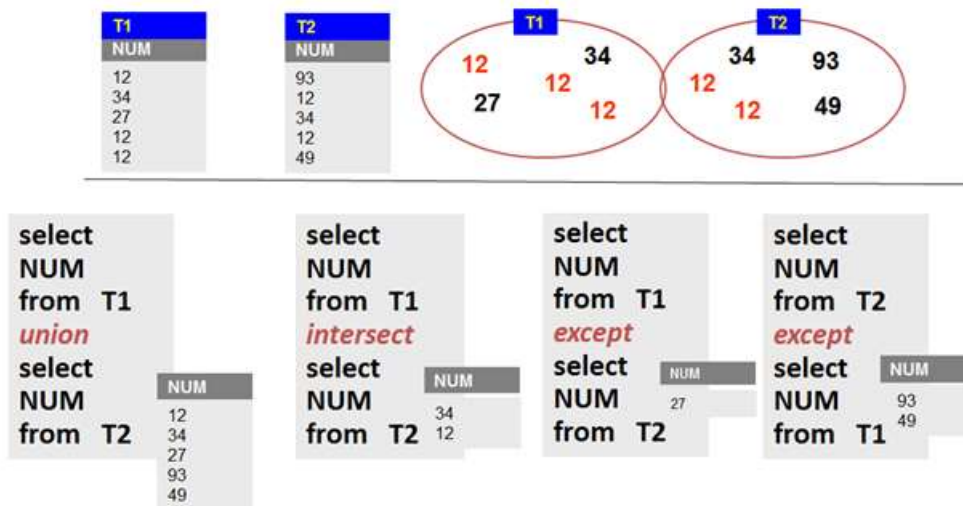
- **Q17 :** Donner les noms des produits qui n'ont pas été acheté

```
SELECT P.nom
FROM produit P
WHERE NOT EXISTS
( SELECT *
  FROM vente V
  WHERE V.IdPro = P.IdPro )
```

Le prédicat **EXISTS** permet de tester si le résultat d'une sous-requête est vide ou non.

### IV.3.8 Les opérations ensemblistes

Les opérations ensemblistes d'union, intersection ou différence peuvent être exprimées en SQL, en connectant **SELECT** avec **UNION**, **INTERSECT** ou **EXCEPT**.



*Exemple pour les opérateurs ensemblistes entre deux tables*

- **Q18 :** Donner les numéros des produits de marque IBM ou ceux achetés par le client d'identifiant '100'.

```
SELECT P.IdPro
FROM produit P
WHERE P.marque = 'IBM'
UNION
SELECT V.IdPro
FROM vente V
WHERE V.IdCli = '100'
```

L'union élimine les doublons.

### IV.3.9 Fonctions de calcul

SQL fournit des fonctions de calcul opérant sur l'ensemble des valeurs d'une colonne de table :

- **COUNT** : nombre de valeurs
- **SUM** : somme des valeurs
- **AVG** : moyenne des valeurs
- **MAX** : plus grande valeur
- **MIN** : plus petite valeur

- **Q19 :** Donner le nombre total de clients.

```
SELECT COUNT ( IdCli )
FROM client
```

- **Q20 :** Donner le nombre total de clients ayant acheté des produits.

```
SELECT COUNT ( DISTINCT IdCli )
FROM vente
```

On peut faire précéder l'argument du mot clé **DISTINCT** pour indiquer que les valeurs redondantes doivent être éliminées avant application de la fonction.

La fonction spéciale **COUNT (\*)** compte toutes les lignes dans une table.

- **Q21 :** Donner le nombre total de 'PS1' vendus.

```
SELECT SUM ( V.qte )
FROM vente V , produit P
WHERE P.IdPro = V.IdPro
AND P.nom = 'PS1'
```

- **Q22 :** Donner les noms des produits moins chers que la moyenne des prix de tous les produits

```
SELECT P1.nom
FROM produit P1
WHERE P1.prix <
    (
        SELECT AVG ( P2.prix )
        FROM produit P2
    )
```

Cet exemple montre un "**singleton select** " pour calculer la moyenne des prix.

#### IV.3.10 La clause **Group By ... Having**

La clause **GROUP BY** permet de partitionner une table en plusieurs groupes.

Toutes les lignes d'un même groupe ont la même valeur pour la liste des attributs de partitionnement spécifiés après **GROUP BY**.

Les fonctions de calcul opèrent sur chaque groupe de valeurs.

La clause **HAVING** permet de spécifier une condition de restriction des groupes. Elle sert à éliminer certains groupes, comme **WHERE** sert à éliminer des lignes.

- **Q23 :** Donner pour chaque référence de produit la quantité totale vendue.

```
SELECT V.IdPro, SUM ( V.qte )
FROM vente V
GROUP BY V.IdPro
```

- **Q24 :** Donner les noms des marques dont le prix moyen des produits est < 50 dinars

```
SELECT P.marque, AVG ( P.prix )
FROM produit P
GROUP BY P.marque
HAVING AVG ( P.prix ) < 50
```

### IV.3.11 La clause *Limit*

La clause **LIMIT** est à utiliser dans une requête SQL pour spécifier le nombre maximum de résultats que l'on souhaite obtenir.

- **Q25** : Donner le nom et la marque des deux plus chers produits.

```
SELECT P.nom, P.marque
FROM produit P
Order by P.prix DESC
Limit 2
```

### IV.3.12 Forme générale de *SELECT*

|                          |                                    |
|--------------------------|------------------------------------|
| <b>SELECT</b> [DISTINCT] | liste d'attributs, expressions     |
| <b>FROM</b>              | liste de tables                    |
| <b>WHERE</b>             | qualification                      |
| <b>GROUP BY</b>          | attributs de partitionnement       |
| <b>HAVING</b>            | qualification de groupe            |
| <b>ORDER BY</b>          | liste de colonnes [ ASC   DESC ]   |
| <b>Limit</b>             | nombre d'enregistrements souhaités |

## V SQLITE

SQLite est un système de gestion de base de données (SGBD) qui sauvegarde la base sous forme d'un fichier multi-plateforme. C'est une bibliothèque qui fournit une base de données légère sur disque ne nécessitant pas de processus serveur distinct et permet d'accéder à la base de données à l'aide d'une variante du langage de requête SQL. Il permet ainsi une gestion simple et rapide des bases de données.

### V.1 Instructions de base sous python

La librairie qu'il faut utiliser est *sqlite3* :

```
import sqlite3
```

Les bases créées avec cette librairie sont enregistrées sous forme de fichier .db, .sq3.

Un fichier .db/.sq3 s'ouvre en créant un objet de type **Connection** :

```
conn = sqlite3.connect('magasin.sq3')
```

Un curseur est ensuite créé à partir de l'objet Connection. Ce curseur représente un canal entre la base de données et le script Python :

```
cur = conn.cursor()
```

Pour exécuter des commandes dans la base, on crée des requêtes SQL sous forme de chaînes de caractères. Ensuite, on l'exécute à partir de la fonction `execute()` :

```
requete = 'SELECT * FROM vente'
cur.execute(requete)
```

Lorsque le travail prévu est terminé, il est nécessaire d'enregistrer les modifications faites sur la base :

```
conn.commit()
```

Enfin, il faut fermer la connexion vers la base :

```
conn.close()
```

## V.2 Les fonctions `execute()` et `executemany()`

La fonction `execute()` permet d'exécuter des requêtes SQL dans la base soit pour :

- Créer / modifier des tables (CREATE, DROP, ALTER)
- Modifier des données (INSERT, DELETE, UPDATE),
- Extraire des données (SELECT)

Il existe plusieurs manières d'insérer des données :

- La plus simple étant celle-ci :

```
c = (136 , 'Yassine' , 'Tunis' )
cur.execute(""" insert into client(idcli,Nom,Ville) VALUES (? ,? , ?) """, c)
```

- En insérant des champs précis :

```
cur.execute("""INSERT INTO client(nom, ville) VALUES ("Aloui" , "Tunis")""")
```

- Ou encore, en insérant tous les champs :

```
cur.execute("""INSERT INTO client VALUES (3,"Hamed", "Gabes")""")
```

- En passant un dictionnaire :

```
data = {"name" : "ramzi", "city" : "Tunis"}
cur.execute("""INSERT INTO client(nom,ville) VALUES(:name, :city)""", data)
```

- En réalisant plusieurs insertions en une seule fois avec la fonction `executemany()` :

```
cli= []
cli.append(("ali", "Gabes"))
cli.append(("lara", "Sfax"))
cli.append(("lamia" , "Mahdia"))
cur.executemany(
    """INSERT INTO client(nom, ville) VALUES(?,?)""", cli
)
```

### V.3 Les fonctions `fetchone()` et `fetchall()`

La requête SQL **SELECT** demande l'extraction d'un ensemble particulier d'enregistrements, qui devront être transférés de la base de données au curseur.

On peut récupérer la première ligne correspondant à une recherche à l'aide de la fonction `fetchone()` :

```
cur.execute("""SELECT nom , ville FROM client""")
client1 = cur.fetchone()
print(client1)    # Le résultat est un tuple: ('Yassine' , 'Tunis')
```

On peut récupérer toutes les lignes de la même recherche en utilisant la fonction `fetchall()` :

```
lignes = cur.fetchall() # Le résultat est une liste de tuples
```

L'objet curseur fonctionne comme un itérateur, invoquant la méthode `fetchall()` automatiquement :

```
for ligne in cur :
    print(ligne)
```

## VI EXERCICES D'APPLICATION

### VI.1 Exercice 1

Soit les relations suivantes de la société Gavasoft :

Emp(NumE, NomE, Fonction, NumS, Embauche, Salaire, Comm, NumD )

Dept(NumD, NomD, Lieu)

#### Exemple

| NumD | NomD     | Lieu    |
|------|----------|---------|
| 1    | Droit    | Créteil |
| 2    | Commerce | Laval   |

| Num | NomE       | Fonction   | NumSup | Embauche | Salaire | Comm | NumD |
|-----|------------|------------|--------|----------|---------|------|------|
| 1   | Gava       | Doyen      | NULL   | 1979     | 10000   | NULL | NULL |
| 2   | Guimezanes | Président  | 1      | 2006     | 5000    | NULL | 1    |
| 3   | Toto       | Stagiaire  | 1      | 2006     | 0       | NULL | 1    |
| 4   | Al-Capone  | Commercial | 2      | 2006     | 5000    | 100  | 2    |

#### Partie Algèbre Relationnelle :

1. Donner les noms des personnes embauchées depuis 2006.
2. Donner la liste des employés travaillant à Créteil .
3. Donner les noms des doyens travaillant dans le département de Commerce.

#### Partie SQL :

4. Créer la table Dept (NumD s'incrémente automatiquement).
5. Ajouter le département 'RH' situé à Paris.
6. Attacher tous les doyens à ce nouveau département 'RH'.
7. Donner la liste des employés ayant une commission (non NULL) classés par commission décroissante.
8. Donner la moyenne des salaires des employés travaillant à Laval.
9. Donner la liste des employés gagnant plus que la moyenne des salaires de l'entreprise.
10. Donner le nombre d'employés par département.
11. Donner le lieu de département dont le nombre d'employés dépasse 100.

#### Partie SQLITE :

Soit le script python suivant :

```
import sqlite3
conn = sqlite3.connect('Gavasoft.db')
cur = conn.cursor()
nouveau={'nom' : 'Martin', 'fonction' : 'stagiaire', 'embauche' : 2010}
```

12. Ecrire le/les instructions permettant d'ajouter ce nouvel employé dans la base de données.



13. Ecrire une fonction `Employes(ville)` qui prend en paramètre le lieu ville d'un département et qui retourne la liste des employés de la ville donnée.
14. Ecrire une fonction `getVilles ( )` qui retourne la liste des villes de tous les départements.
15. Ecrire les instructions nécessaires permettant de sauvegarder les renseignements des employés de chaque ville dans des fichiers. Chaque fichier sera nommé par le nom de la ville. Par exemple 'Creteil.txt' comporte dans chaque ligne les données d'un employé séparées par un point-virgule.

## VI.2 Exercice 2

On considère le schéma relationnel suivant qui modélise une application sur la gestion de livres et de disques dans une médiathèque.

`Disque(``CodeOuv``, Titre, Style, Pays, Année, Producteur)`

`E_Disque(``NumEx``, CodeOuv, DateAchat, Etat, CodeA)`

`Livre(``CodeOuv``, Titre, Editeur, Genre, Collection, CodeA)`

`Auteurs(``CodeA``, Nom, Prenom)`

`Abonne(``NumAbo``, Nom, Prénom, Rue, Ville, CodeP, Téléphone)`

`Prêt(``CodeOuv``,` `NumEx``, DisqueOuLivre, NumAbo, DatePret)`

`Personnel(``NumEmp``, Nom, Prénom, Adresse, Fonction, Salaire)`

Ecrire les requêtes SQL permettant de répondre aux questions suivantes :

1. Quel est le contenu de la relation Livre ?
2. Quels sont les titres des romans édités par Gava-Editor ?
3. Quelle est la liste des titres que l'on retrouve à la fois comme titre de disque et titre de livre ?
4. Quelle est l'identité des auteurs qui ont fait des disques et écrit des livres ?
5. Quels sont les différents styles de disques proposés ?
6. Quel est le salaire annuel des membres du personnel gagnant plus de 20000 euros en ordonnant le résultat par salaire descendant et nom croissant ?
7. Donnez le nombre de prêts en cours pour chaque famille en considérant qu'une famille regroupe des personnes de même nom et possédant le même numéro de téléphone ?
8. Quel est le code du disque dont la médiathèque possède le plus grand nombre d'exemplaire ?
9. Quels sont les éditeurs pour lesquels l'attribut Collection n'a pas été renseigné ?
10. Quels sont les abonnés dont le nom contient la chaîne « ALDO » ?
11. Quel est le nombre de prêts en cours ?
12. Quels sont les salaires minimum, maximum et moyen des employés exerçant une fonction de bibliothécaire ?
13. Quel est le nombre de genres de livres différents ?
14. Quel est le nombre de disque acheté en 1998 ?
15. Quel est le salaire annuel des membres du personnel gagnant plus de 20000 euros ?
16. Quel est le nom, prénom et l'adresse des abonnés ayant emprunté un disque le '12/01/2006' ?
17. Quels sont les titres des livres et des disques actuellement empruntés par Frédéric Gava ?
18. Quels sont les titres des ouvrages livres policiers ou disques de Jazz empruntés par Frédéric Gava ?
19. Quel est l'identité des auteurs qui n'ont écrit que des romans policiers (genre=policier) ?
20. Quels sont les codes des ouvrages des livres pour lesquels il y a au moins un exemplaire emprunté et au moins un exemplaire disponible ?

## VII CORRIGE DES EXERCICES

### VII.1 Corrigé de l'exercice 1

#### Partie Algèbre Relationnelle:

1. Donner les noms des personnes embauchées depuis 2006

$$R1 = \sigma_{Emp} (Embauche \geq 2006)$$

$$R2 = \pi R1 (NomE)$$

2. Donner la liste des employés travaillant à Créteil

$$R1 = Emp \bowtie Dept (Emp.NumD = Dept.NumD)$$

$$R2 = \sigma R1 (Lieu = 'Créteil')$$

$$R3 = \pi R2 (NomE)$$

3. Donner les noms des doyens travaillant dans le département de Commerce.

$$R1 = Emp \bowtie Dept (Emp.NumD = Dept.NumD)$$

$$R2 = \sigma R1 (NomD = 'Commerce')$$

$$R3 = \sigma R2 (Fonction = 'Doyen')$$

$$R4 = \pi R3 (NomE)$$

#### Partie SQL:

4. Créer la table Dept (NumD s'incrémente automatiquement)

```
CREATE TABLE `Dept` (
    `NumD`      INTEGER PRIMARY KEY AUTOINCREMENT,
    `NomD`      TEXT,
    `Lieu`      TEXT
);
```

5. Ajouter le département 'RH' situé à Paris

```
Insert into Dept(nomD , Lieu ) values('RH , 'Paris')
```

6. Attacher tous les doyens à ce nouveau département 'RH'

```
Update Emp set numD = 3 where fonction = 'Doyen'
```

7. Donner la liste des employés ayant une commission (non NULL) classé par commission décroissante

```
Select nomE
From Emp
Where Comm is not NULL
Order by comm desc
```

8. Donner la moyenne des salaires des employés travaillant à laval

```
Select avg(salaire)
From emp
```

```
Where numD = (select numD
              From dept
              Where lieu = 'laval'
            )
```

Ou bien :

```
Select avg(salaire)
From emp
Join dept
ON emp.numD = dept.numD
Where lieu = 'laval'
```

9. Donner la liste des employés gagnant plus que la moyenne des salaires de l'entreprise

```
Select *
From emp
Where salaire > (select avg(salaire)
                 From emp
                )
```

10. Donnez le nombre d'employés par département

```
Select count(*)
From Emp
Group by numD
```

11. Donnez le lieu de département dont le nombre d'employés dépasse 100

```
Select lieu
From Dept
Join Emp
On Dept.NumD = Emp.NumD
Group by Dept.NumD
Having count(*) > 100
```

### **Partie SQLITE :**

Soit le script python suivant :

```
import sqlite3
conn = sqlite3.connect('Gavasoft.db')
cur = conn.cursor()
nouveau={'nom' : 'Martin', 'fonction' : stagiaire, 'embauche' : 2010}
```

**12.** Ecrire le/les instructions permettant d'ajouter ce nouveau employé dans la base de donnée

```
cur.execute ( 'insert into emp values( :nom, :fonction, :embauche) ,nouveau )
```

**13.** Ecrire une fonction Employes(ville) qui prend en paramètre le lieu ville d'un département et qui retourne la liste des employés de la ville donnée.

```
def getEmployes(ville) :
    req = 'select * from emp where numD = (select numD from dept where lieu
    ={})'.format(ville)
    cur.execute(req)
    employes = cur.fetchall()
    return employes
```

**14.** Ecrire une fonction getVilles ( ) qui retourne la liste des villes de tous les départements.

```
def getVilles ( ) :
    req = 'select distinct lieu from dept'
    cur.execute(req)
    return cur.fetchall()
```

**15.** Ecrire les instructions nécessaires permettant de sauvegarder les renseignements des employés de chaque ville dans des fichiers. Chaque fichier sera nommé par le nom de la ville par exemple 'Creteil.txt' et comporte dans chaque ligne les données d'un employé séparées par un point-virgule

```
villes = getVilles ( )
for ville in villes :
    f = open(ville+'.txt' , 'w')
    Lemp = getEmployes(ville)
    for emp in Lemp :
        emp = [str(x) for x in emp ]
        ligne = ' ;'.join( emp) +'\n'
        f.write(ligne)
    f.close()
```

## VII.2 Corrigé de l'exercice 2

- 1) SELECT \* FROM Livre
- 2) SELECT Titre FROM Livre WHERE Editeur="Droit-Edition" AND Genre="Roman"
- 3) SELECT D.Titre FROM Disque D, Livre L WHERE D.Titre=L.Titre

- 4) SELECT Nom, Prenom FROM Disque D, Auteur A WHERE D.CodeA=A.CodeA INTERSECT SELECT Nom, Prenom FROM Livre L, Auteur A WHERE D.CodeA=L.CodeA
- 5) SELECT DISTINCT Style FROM Disque
- 6) SELECT Nom, Prénom, Salaire\*12 AS Salaire\_Annuel FROM Personnel WHERE Salaire\_Annuel>20000 ORDER BY Salaire DESC, Nom ASC
- 7) SELECT Nom, Téléphone, COUNT(\*) FROM Abonne A, Prêt P WHERE A.NumAbo=P.NimAbo GROUP BY Nom, Téléphone
- 8) SELECT CodeOuv FROM E\_Disque GROUP BY CodeOuv HAVING COUNT(\*)=(SELECT MAX (COUNT(\*)) FROM E\_Disque GROUP BY CodeOuv)
- 9) SELECT Editeur FROM Livre WHERE Collection IS NULL
- 10) SELECT \* FROM Abonne WHERE Nom='%'ALDO%'
- 11) SELECT COUNT(\*) FROM Prêt
- 12) SELECT MIN(Salaire), MAX(Salaire), AVG(Salaire) FROM Personnel WHERE Fonction=« bibliothécaire »
- 13) SELECT COUNT(DISTINCT Genre) FROM Livre
- 14) SELECT COUNT(\*) FROM E\_Disque WHERE DateAchat BETWEEN '01-Jan-2006' AND '10-Dec-2007'
- 15) SELECT Nom, Prénom, Salaire\*12 AS Salaire\_Annuel FROM Personnel WHERE Salaire\_Annuel>20000
- 16) SELECT Nom, Prénom, Rue, Ville, CodeP FROM Abonne A, Prêt P, Disque D WHERE A.NumAbo=P.NumAbo AND P.CodeOuv=D.CodeOuv AND DatePret='12-Jan-2006'
- 17) SELECT Titre FROM Abonne A, Prêt P, Disque D WHERE A.NumAbo=P.NumAbo AND P.CodeOuv=D.CodeOuv AND NOM="Gava" AND Prénom="Frédéric" UNION SELECT Titre FROM Abonne A, Prêt P, Livre L WHERE A.NumAbo=P.NumAbo AND P.CodeOuv=L.CodeOuv AND NOM="Gava" AND Prénom="Frédéric"
- 18) SELECT CodeOuv FROM Prêt P, Abonne A WHERE P.NumAbo=A.NumAbo AND Prénom="Frédéric" AND Nom="Gava" AND CodeOuv IN (SELECT CodeOuv FROM Livre WHERE Genre="Policier") OR CodeOuv IN (SELECT CodeOuv FROM Disque WHERE Style="Jazz")
- 19) SELECT Identité FROM Auteur A, Livre L WHERE A.CodeOuv=L.CodeOuv AND Genre="Policier" AND NOT ALL(SELECT Identité FROM Auteur A, Livre L WHERE A.CodeOuv=L.CodeOuv AND Genre<>"Policier")
- 20) (SELECT P.CodeOuv FROM E\_Livre E, Prêt P WHERE E.CodeOuv=P.CodeOuv) INTERSECT (SELECT CodeOuv FROM E\_Livre E WHERE NOT EXISTS(SELECT \* FROM Prêt P WHERE E.CodeOuv=P.CodeOuv AND E.NumEx=P.NumEx))

## VIII BIBLIOGRAPHIE ET NETOGRAPHIE

### Ouvrages :

Toute l'informatique en CPGE scientifique, Etienne Cochard, Sophie Rainero, Marielle Roussange, Emanuelle Sebert-Cuvillier

Apprendre à programmer avec Python 3, G. Swinnen

Informatique pour tous en classes préparatoires aux grandes écoles, Benjamin Wack& al., Edition Eyrolles, Août 2013, ISBN: 978-2-212-13700-2.

### Supports de Cours

Support de Cours de W.Meftah, Les bases de données relationnelles, pour 2<sup>ème</sup> année préparatoire MP-PC-T et BG, IPEIS, Année universitaire 2019-2020.

### Sites Web :

<https://docs.python.org/3/library/sqlite3.html>

<https://python.doctor/page-cours-python-debutant-documentation>

Cours SGBD 1 Concepts et langages des Bases de Données Relationnelles :

[https://www.i3s.unice.fr/~nlt/cours/licence/sghbd1/sghbd1\\_cours.pdf](https://www.i3s.unice.fr/~nlt/cours/licence/sghbd1/sghbd1_cours.pdf)

Cours de Base de Données, Cours n.3 Algèbre relationnelle : <http://www.i3s.unice.fr/~edemaria/cours/c3.pdf>

<https://www.cours-gratuit.com/langage-sql/>