A.VII. Bases de données

A.VII.1 Définition

Une base de données (BDD) est, comme son nom l'indique, un outil de stockage de données. Elle permet la collecte, le stockage et l'utilisation des données associées.

Le système de gestion des bases de données s'appelle le « SGBD » pour « Système de Gestion de Base de Données ». Il permet d'accéder aux données de la base de données par l'intermédiaire de requêtes simples qui cachent une manipulation complexe des données.

A.VII.2 Systèmes de gestion des données

Le programme d'IPT propose que vous ayez quelques notions sur l'origine des bases de données relationnelles et leur contexte d'utilisation. Ce paragraphe, sans rentrer dans les détails, vous apportera les quelques informations à connaître.

A.VII.2.a Peer to peer

Prenons un premier exemple qui doit vous parler. Le peer to peer est un moyen que possède chaque particulier pour récupérer ou envoyer des données qu'il possède sur sa machine. Chaque utilisateur est donc aussi serveur de données et un logiciel remplissant à la fois les fonctions de client et de serveur gère les échanges.

A.VII.2.b Client-Serveur

L'architecture client-serveur se décompose, évidemment, en deux parties :

- Client : gère la présentation et la partie applicative (ce que l'on fait des données)
- Serveur : stocke les données de façon cohérente

Le client possédant des droits d'accès envoie des requêtes au serveur qui les traite et retourne les résultats. Le serveur intègre alors un logiciel de gestion de base de données (SGBD) ainsi qu'un espace de stockage de données.

Le langage SQL (Structured Query Langage) est classiquement utilisé dans les requêtes entre client et serveur.

L'avantage de ce type d'architecture est que le client n'a pas à connaître ou chercher à maîtriser la manière qu'à le serveur de gérer les données (dimension des disques, répartition des données, sauvegarde…). La qualité de ce type d'architecture est liée à la notion de bases de données relationnelles et au langage SQL (Structured Query Langage). Le modèle relationnel que nous allons aborder est né en 1970 (E. Codd).

Sans rentrer dans les détails, voici quelques avantages et inconvénients de cette architecture :

- Fiabilité et performance du fait de beaucoup de travaux réalisés en presque 60 ans
- Inconvénients : sécurité (tous les clients accèdent à la base de données) et coût

CC BY NC SA

A.VII.2.c Architecture trois/tiers

L'architecture trois-tiers est un peu plus évoluée que l'architecture client/serveur puisqu'au lieu de diviser le travail en deux, il est divisé en trois. Celles-ci sont indépendantes, ce qui permet de changer de mode de stockage, le type de traitement des données ou d'interface homme machine sans pour autant modifier les autres étages. Les étages sont toutefois dépendants entre eux puisqu'ils doivent communiquer.

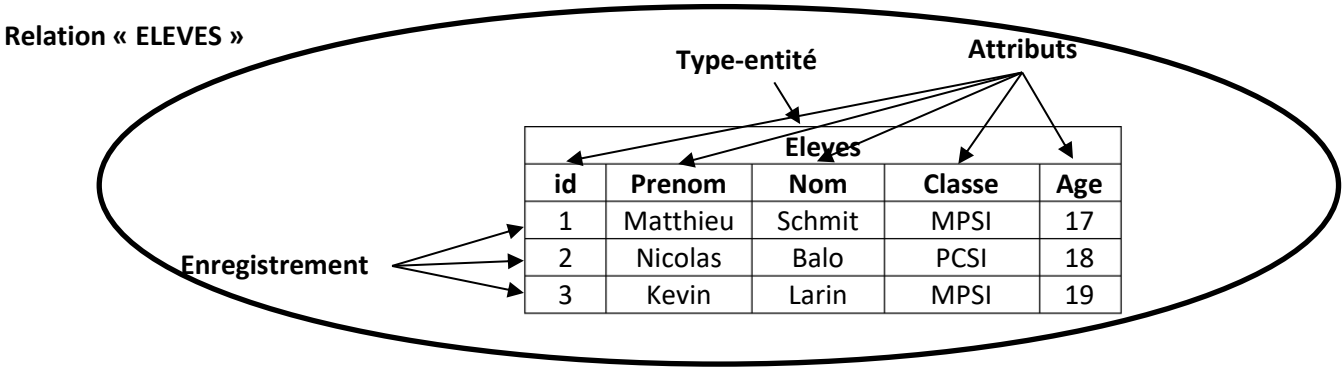
Le gros avantage de cette architecture est que le client ne nécessite qu'une installation minime, souvent un navigateur web, pour traiter ses données, le serveur applicatif étant différencié du client. Evolution de l'architecture client/serveur, elle continue à utiliser les outils de bases de données relationnel et de langage SQL. Cette architecture présente un gain de sécurité puisque les clients n'ont plus chacun accès à la base de données. C'est directement le serveur applicatif qui y accède et il ne donne pas les identifiants d'accès aux clients.

Le principe de l'architecture trois/tiers est d'avoir 3 niveaux de traitements (Présentation, Traitement, Accès aux données) :

- Couche d'accès aux données : Le système de stockage de données peut prendre plusieurs formes différentes, par exemple l'utilisation de bases de données relationnelles.
- Couche de traitement : C'est la couche dans laquelle il y a la partie fonctionnelle de l'application (logique applicative).
- Couche de présentation : Cette couche correspond à l'interface homme/machine. Elle peut avoir différentes formes adaptées aux utilisateurs pour réaliser le même travail. Elle relaie les requêtes de l'utilisateur vers la couche de traitement et renvoie à l'utilisateur les traitements de cette même couche.

A.VII.3 Vocabulaire

Le programme d'IPT propose d'aborder le concept de bases de données relationnelles, c'est-à-dire dans laquelle l'information est organisée dans des tableaux à deux dimensions appelés des **relations** ou des **tables**. Les lignes de ces tables sont appelées **n-uplets** (tuples en anglais) ou **enregistrements**, et les colonnes sont associées à des **attributs**.



Un **Type-entité** définit la caractéristique de la relation, dans notre cas ce sont des « ELEVES ». Une **entité** est une occurrence de son type-entité. A chaque enregistrement d'une relation correspond donc une entité du type-entité, ici nos 3 élèves Matthieu, Nicolas et Kevin.

Le **domaine** d'un attribut, fini ou infini, est l'ensemble des valeurs qu'il peut prendre. Le domaine de l'attribut « Age » est l'ensemble des entiers de 0 à 130 (actuellement) par exemple.

Le **degré** d'une relation est son nombre d'attributs, 5 dans notre exemple (id, Prenom, Nom, Classe, Age).

Un **schéma de relation** précise le nom d'une relation ainsi que la liste des attributs avec leur domaine : Eleves(id : Entier, Prenom : Chaîne , Nom : Chaîne , Classe : Chaîne , Age : Entier). On appelle **Schéma relationnel** l'ensemble des schémas de relation.

On appelle « **clé primaire** » une ou plusieurs données permettent d'identifier de manière unique un enregistrement dans une table. On ne peut avoir deux fois le même enregistrement, il existe donc toujours une clé primaire. On la souligne dans le schéma de relation: Eleves(id : Entier, Prenom : Chaîne , Nom : Chaîne , Classe : Chaîne , Age : Entier). On ajoute souvent un identifiant « id » en première colonne, entier unique qui permet d'être clé primaire.

Une **base de données relationnelle** est constituée par l'ensemble des n-uplets des différentes relations du schéma relationnel.

A.VII.4 Exemple support du cours

Soient les deux tables suivantes :

Eleves					
id	**Prenom**	**Nom**	**Classe**	**Age**	**Ville**
1	Steph	ANE	MPSI	17	Paris
2	Marc	IMBUT	PCSI	18	Marseille
3	Jo	NID	MPSI	19	Montpelier
4	Rayan	AIR	MP	19	Paris
5	Tom	DESAVOIE	PCSI	18	Nice
6	Jerry	CANE	PSI	19	Paris
7	Paul	AINE	MP	19	Marseille
8	Jean	NAIMAR	PCSI	18	Toulouse
9	Jack	OUILLE	MPSI	18	Montpelier
10	Sam	OURAIL	PC	20	Paris

Profs					
id	**Titre**	**Prenom**	**Nom**	**Classe**	**Salle**
1	Mr	Denis	DEFAUCHY	MPSI	D21
2	Mr	Paul	YMERE	PCSI	A05
3	Mme	Sylvie	DEFOU	MP	B18
4	Mr	Georges	AITTE	PC	C15
5	Mme	Marie	AGE	PSI	E8

La base de données relationnelle est donc :

- Eleves(id : Entier, Prenom : Chaîne , Nom : Chaîne , Classe : Chaîne , Age : Entier,Ville : Chaîne)
- Profs(id : Entier, Titre : Chaîne, Prenom : Chaîne , Nom : Chaîne , Classe : Chaîne , Salle : Chaîne)

CC BY NC SA

A.VII.5 Généralités

Nous serons ici amenés à détailler le langage SQL (Structured Query Language) pour interroger les bases de données, c'est-à-dire pour réaliser des **Requêtes** sur une base de données.

A.VII.5.a Manipulations et requêtes

Les bases de données se trouvent sous différentes extensions (csv, sql, xml, xls, bd…). On en trouve par exemple beaucoup concernant la France ici : https://www.data.gouv.fr/fr/

La gestion des bases de données est totalement indépendante du logiciel Python. En effet, le langage SQL peut être directement utilisé dans des logiciels comme « Sqlite browser » gratuit et disponible ici : http://sqlitebrowser.org/

Je vous recommande d'importer un fichier .csv en faisant :

- Création d'une nouvelle bdd et enregistrement
- Ne pas remplir de tables (Cancel)
- Fichier – Importer – Table vers un fichier csv – Afficher toutes les extensions – Ouvrir le fichier
- Choisir les bonnes options d'importation en visualisant le résultat

Ecriture des requêtes dans l'onglet « Exécuter le SQL »

Nous allons toutefois voir dans ce cours comment traiter les bases de données sous Python, même s'il est possible d'en être totalement indépendant.

A.VII.5.b Généralités de langage

Les instructions SQL dans un code doivent impérativement être écrites en lettres majuscules et se finir par un point-virgule « ; ».

Quelques symboles peuvent vous être utiles pour effectuer des recherches particulières :

- « * » permet de dire que l'on recherche tous les attributs
- « % » désigne une chaîne de caractères quelconque – « A% » désignera un mot commençant par A – « %a% » un mot contenant a…
- « _ » désigne un caractère quelconque
- « '…' » désigne un champ au format texte
- « 'mm/jj/aaa' » désigne un champ au format date

A.VII.5.c Structure d'une manipulation de bases de données sous Python

La gestion d'une base de données sous Python s'organise ainsi :

```
import sqlite3
BDD = sqlite3.connect('BDD.db')
cursor = BDD.cursor()
cursor.execute("""…""")
cursor.executemany("""…""")
BDD.commit()
BDD.close()
```

C'est normal qu'à ce stade, vous ne compreniez pas grand-chose à ce code ! En quelques mots, et nous développerons tout cela dans les prochains paragraphes :

- On importe la librairie de bases de données SQL sous Python
- On ouvre ou crée la base de données BDD.db
- On crée un outil « cursor » qui va manipuler la base de données
- On exécute alors différentes requêtes, et les « … » sont des requêtes que vous apprendrez à manipuler par la suite
- On applique les modifications au fichier BDD.db
- On ferme la base de données

Remarque : Dans `cursor.execute("""…""")`, il faut un texte sous forme de string, on peut donc utiliser `",'` ou `'''` ou `"""`. J'ai choisi ici les `"""` (équivalent à `'''`) pour les raisons suivantes :

- Pas de problèmes pour utiliser une seule Guillement pour définir un texte dans le texte : `"""("A","B")"""`
- La couleur du texte sous Python est bleue et permet d'identifier les codes SQL facilement

A.VII.5.d Premiers pas – Création - Ouverture

A.VII.5.d.i Librairie sous Python

Nous allons manipuler les bases de données sous Python avec la librairie sqlite3. Il faut donc écrire :

```
import sqlite3
```

A.VII.5.d.ii Création/ouverture d'une base de données

Que la base de données existe ou non, une commande permet de l'ouvrir ou de la créer. Il faut écrire :

```
BDD = sqlite3.connect('BDD.db')
```

Où `'BDD.db'` est la base de données à ouvrir ou à créer, dans le répertoire de travail (on peut préciser un chemin absolu) comme pour les fichiers textes.

Concrètement, on demande à Python de se « connecter » à la base de données `'BDD.db'` qu'il ouvre ou crée si elle n'existe pas.

A.VII.5.d.iii Création de l'outil de manipulation de la base de données

Pour manipuler la base de données BDD, il est nécessaire de créer un outil sous Python qui va permettre de travailler dessus. Il faut écrire :

```
cursor = BDD.cursor()
```

Ne l'oubliez pas !

A.VII.5.d.iv Création/suppression de relations/tables

- **Création d'une relation**

Nous pouvons maintenant créer des relations à l'aide d'une requête « CREATE TABLE ». Créons les deux tables de l'exemple proposé précédemment :

```
cursor.execute("""CREATE TABLE Eleves(id INTEGER PRIMARY KEY,Prenom
TEXT,Nom TEXT,Classe TEXT,Age INTERGER,Ville TEXT)""")
cursor.execute("""CREATE TABLE Profs(id INTEGER PRIMARY KEY,Titre
TEXT,Prenom TEXT,Nom TEXT,Classe TEXT, Salle TEXT)""")
```

Quelques explications :

- Pour exécuter une commande, il faut écrire `cursor.execute("""  """)` avec entre les guillemets les commandes du langage SQL.
- Pour créer une table, on écrit : `CREATE TABLE` suivi de son nom, puis entre parenthèses, on précise le nom de chaque attribut suivi de son type en majuscules (TEXT pour un texte, INTEGER pour un entier…).
- La mention `PRIMARY KEY` est optionnelle pour nos applications simples – Elle impose en tout cas que le champ soit unique pour chaque n-uplet et il sera donc impossible d'ajouter un n-uplet ayant le même id dans notre exemple

Ça y est, les tables sont initialisées, encore vides, et pas encore enregistrées dans le fichier associé à la base de données.

Remarque : si une table existe, il est possible de préciser de créer la table si elle n'existe pas afin d'éviter une erreur si elle existe en précisant : `CREATE TABLE IF NOT EXISTS`

- **Lister les relations d'une base de données (pour info)**

```
cursor.execute("""SELECT name FROM sqlite_master WHERE type='table';""")
print(cursor.fetchall())
```

A écrire tel quel sans rien changer !

- **Lister les attributs d'une relation (pour info)**

```
cursor.execute("""PRAGMA table_info(Eleves);""")
print(cursor.fetchall())
```

- **Suppression d'une relation**

Il suffit d'écrire :

```
cursor.execute("""DROP TABLE Eleves; """)
```

On supprime ainsi la table « Eleves ».

A.VII.5.d.v Ajout/modification/suppression d'enregistrements

Il existe plusieurs manières de procéder, voyons les deux plus classiques.

- **Ajout ligne par ligne**

On peut ajouter les donnes ligne par ligne, voici deux possibilités :

```
cursor.execute("""INSERT INTO Eleves VALUES
(1,"Steph","ANE","MPSI",17);""")
```

Cette méthode très simple ajoute à la table « Eleves » le nuplet `(1,"Steph","ANE","MPSI",17);"""`. Attention, il doit avoir la même taille que lors de la définition de la relation (CREATE TABLE).

```
cursor.execute("""INSERT INTO Eleves VALUES (?,?,?,?,?)
;""",(1,"Steph","ANE","MPSI",17))
```

Cette méthode un peu moins évidente au premier abord se comprend ainsi : Ajouter à la table « Eleves » les 5 valeurs (?, ?, ?, ?, ?) précisées dans le nuplet un peu plus loin… Ce fonctionne tant qu'il y a au moins 2 éléments par n-uplet. Je la précise surtout pour que vous compreniez l'écriture de l'ajout d'une liste au paragraphe suivant.

- **Ajout d'une liste de lignes**

On crée une liste de nuplets puis on l'ajoute à la table :

```
Liste_Eleves =
[(1,"Steph","ANE","MPSI",17,"Paris"),(2,"Marc","IMBUT","PCSI",18,"Marseil
le")]
cursor.executemany("""INSERT INTO Eleves VALUES
(?,?,?,?,?);""",Liste_Eleves)
```

On notera qu'il faut alors utiliser la commande « executemany » et non plus « execute ». Comme précisé précédemment, il faut au moins 2 attributs pour utiliser la méthode (?, ?…).

- **Modification d'un enregistrement : UPDATE SET**

```
cursor.execute('''UPDATE Eleves SET Nom = New_Nom, Prenom = New_Prenom
WHERE id = 2 ; ''')
```

On met à jour le nom et le prénom de l'élève d'id 2.

- **Suppression d'un enregistrement : DELETE**

```
cursor.execute('''DELETE FROM Eleves WHERE id = 3 ;''')
```

On supprime l'élève d'id 3.

A.VII.5.d.vi Validation des modifications dans le fichier 'BDD.db'

Toutes les modifications qui sont faites sur la base de données sont en réalité réalisées dans la mémoire de Python, et non dans la base de données elle-même. Il faut donc, pour que le fichier de la base de données soit modifié, exécuter une commande « d'enregistrement » en exécutant la ligne :

```
BDD.commit()
```

A.VII.5.d.vii Fermeture de la base de données

Comme pour les fichiers textes, il faut obligatoirement refermer la base de données après utilisation sous Python.

```
BDD.close()
```

A.VII.6 Requêtes SQL au programme

A.VII.6.a.i Requêtes et résultats associés

Maintenant que la base de données est créée, on peut effectuer des requêtes. Voici quelques exemples :

- Combien d'élèves viennent de paris
- Quel âge moyen ont les élèves en 2° année
- En quelle salle doivent aller les élèves de MPSI pour voir leur prof principal
- Etc

La structure des requêtes est toujours à peu près la même :

```
cursor.execute("""SELECT … FROM … WHERE …;""")
```

Remarque : Il y aura quelques variantes que nous allons voir avec des exemples.

On pourra alors récupérer les résultats de la requête à l'aide de l'une des commandes suivantes :

```
Resultats = cursor.fetchall()
Resultat = cursor.fetchone()
```

Remarque : cursor contient le résultat de la requête précédente uniquement.

La variable Resultat(s) contiendra alors l'ensemble des résultats de la requête avec :

- Pour fetchone, le premier résultat
- Pour fetchall, tous les résultats

Il restera à utiliser Python classiquement pour traiter ces résultats, par exemple :

```
Resultats = cursor.fetchall()
for Resultat in Resultats:
    print("Le premier eleve s'appelle : %s %s, il est en classe de %s" %
(Resultat[2], Resultat[3], Resultat[4]))
```

A.VII.6.a.ii Opérateurs de base

- **Projection : SELECT FROM**

La projection permet de n'afficher qu'une partie des attributs (colonnes) d'une relation.

Objectif	Requête
Obtenir toutes les informations d'un tableau	`SELECT * FROM Eleves ;` *Toutes les informations concernant les élèves*
Sélectionner certains attributs dans un tableau	`SELECT Nom,Prenom FROM Profs ;` *Noms et prénons des profs*
Obtenir toutes les occurrences d'un attribut en supprimant les doublons	`SELECT DISTINCT Age FROM Eleves ;` *Tous les âges des élèves*

- **Sélection (ou restriction) : WHERE**

La sélection permet de n'afficher qu'une partie des enregistrement (lignes) d'une relation. On utilise alors le mot **WHERE**.

Les opérateurs de sélection utilisables sont :

- « = » ou « != » utilisables avec tous types de données
- « > », « < », « >= », « <= » utilisables uniquement avec des données numériques
- LIKE, BETWEEN, IN, AND, OR, NOT

Exemples :

Objectif	Requête
Récupérer les attributs des enregistrements possédant un attribut spécifique	`SELECT Nom,Prenom FROM Eleves WHERE Classe = "MPSI" ;` *Noms et prénoms des élèves de MPSI*
	`SELECT DISTINCT Age FROM Eleves WHERE Classe LIKE "MPSI" OR Classe LIKE "PCSI" ;` *Tous les âges des élèves de première année sans doublons*
	`SELECT DISTINCT Age FROM Eleves WHERE Age BETWEEN 18 AND 19 ;` *Tous les âges des élèves de première année sans doublons*

- **Jointure : JOIN ON**

La jointure consiste à trouver des données en lien avec plusieurs tables. Par exemple :

- « Dans quelle salle doivent aller les élèves pour voir leur prof ? ».
- « Dans quelle salle doivent aller les élèves de MPSI pour voir leur prof ? ».

Il faut identifier les élèves de la classe MPSI dans la table « Eleves », chercher le prof de la classe MPSI dans la table « Profs », et associer la salle aux élèves concernés.

Le programme d'IPT propose de ne se limiter qu'aux jointures simples utilisant Join…ON…=….

Voici comment écrire les requêtes proposées :

```
SELECT Eleves.Prenom, Eleves.Nom, Profs.Salle FROM Eleves JOIN Profs ON
Eleves.Classe = Profs.Classe ;

SELECT Eleves.Prenom, Eleves.Nom, Profs.Salle FROM Eleves JOIN Profs ON
Eleves.Classe = Profs.Classe WHERE Eleves.Classe = "MPSI" ;
```

Comme il faut préciser ce que l'on sélectionne parmi les deux relations, on précise les attributs en ajoutant le nom de la table et un point avant chacun d'entre eux.

Quelques explications : La jointure **(Eleves JOIN Profs ON Eleves.Classe = Profs.Classe)** crée une nouvelle table de la basée sur le point commun de la classe de la forme :

Profs						**Eleves**					
id	**Titre**	**Prenom**	**Nom**	**Classe**	**Salle**	**id**	**Prenom**	**Nom**	**Classe**	**Age**	**Ville**
1	Mr	Denis	DEFAUCHY	MPSI	D21	1	Steph	ANE	MPSI	17	Paris
1	Mr	Denis	DEFAUCHY	MPSI	D21	3	Jo	NID	MPSI	19	Montpelier
1	Mr	Denis	DEFAUCHY	MPSI	D21	9	Jack	OUILLE	MPSI	18	Montpelier
2	Mr	Paul	YMERE	PCSI	A05	2	Marc	IMBUT	PCSI	18	Marseille
2	Mr	Paul	YMERE	PCSI	A05	5	Tom	DESAVOIE	PCSI	18	Nice
2	Mr	Paul	YMERE	PCSI	A05	8	Jean	NAIMAR	PCSI	18	Toulouse
3	Mme	Sylvie	DEFOU	MP	B18	4	Rayan	AIR	MP	19	Paris
3	Mme	Sylvie	DEFOU	MP	B18	7	Paul	AINE	MP	19	Marseille
4	Mr	Georges	AITTE	PC	C15	10	Sam	OURAIL	PC	20	Paris
5	Mme	Marie	AGE	PSI	E8	6	Jerry	CANE	PSI	19	Paris

Il reste alors comme d'habitude à écrire : SELECT … FROM **(table de jointure)** WHERE …

Remarques :

- Il pourrait y avoir plusieurs profs ayant la même classe, la jointure va créer autant de lignes qu'il le faut pour que tous les élèves de MPSI soient en face de chaque prof de MPSI.
- Vous remarquerez que rien ne change si on intervertit les 2 tables droite/gauche. Ce qui peut changer si l'instruction est écrite selon Eleves JOIN Profs ou Profs JOIN Eleves, c'est l'ordre de l'organisation des classes, mais les requêtes sur cette table seront inchangées.
- Ecrire **Eleves JOIN Profs** sans condition ON revient à créer une table dans laquelle pour chaque n-uplet d'une des tables initiales, on associe tous les n-uplets de l'autre table, ce qui revient à faire un produit cartésien CROSS JOIN que nous verrons dans un prochain paragraphe

A.VII.6.a.iii Opérateurs ensemblistes

- **Union – Intersection - Différence**

Les trois opérateurs usuels en bases de données sont l'union, l'intersection et la différence. Ces opérateurs s'utilisent entre deux tables (relations) de même structure.

Le principe de ces opérateurs consiste à ajouter un mot (UNION, EXCEPT, INTERSECT) entre deux requêtes SELECT.

Exemple : Soient les deux tables suivantes contenant la référence et le prix d'objets d'un magasin :

T1	
Ref	**Prix**
A27	12
C32	17
E15	18
Z7	20

T2	
Ref	**Prix**
C32	17
A27	15
B20	20
A12	21

Obtention d'une table de l'ensemble des produits Rq : UNION ne garde pas les doublons	`SELECT * FROM T1 UNION SELECT * FROM T2 ;` `[('A12', 21), ('A27', 12), ('A27', 15), ('B20', 20), ('C32', 17), ('E15', 18), ('Z7', 20)]`
Enlever les produits de T1 présents dans T2 (même ref & prix)	`SELECT * FROM T1 EXCEPT SELECT * FROM T2 ;` `[('A27', 12), ('E15', 18), ('Z7', 20)]`
Trouver les produits présents dans les deux tables (même ref & prix)	`SELECT * FROM T1 INTERSECT SELECT * FROM T2 ;` `[('C32', 17)]`

Remarque : on pourra bien-sûr ajouter des restrictions (WHERE) pour affiner les recherches.

- **Produit cartésien : CROSS JOIN**

Soient les deux tables suivantes contenant les groupes sanguins d'une population de 4 mâles et 4 femelles d'un laboratoire :

MALES	
Groupe	**Resus**
A	-
AB	+
B	+
B	-

FEMELLES	
Groupe	**Resus**
B	-
B	-
AB	+
A	-

On souhaite créer une base de données de tous les mixages possibles afin d'étudier les groupes possibles des enfants issus de tous les croisements possibles.

Il suffit d'utiliser l'opérateur CROSS JOIN :

```
SELECT * FROM MALES CROSS JOIN FEMELLES ;
```

Voici les premières lignes du résultat obtenu :

A	-	B	-
A	-	B	-
A	-	AB	+
A	-	A	-
AB	+	B	-
AB	+	B	-
…	…	…	…

Remarque : comme vu pour la jointure, on obtient le même résultat avec :

Il suffit d'utiliser l'opérateur CROSS JOIN :

```
SELECT * FROM MALES JOIN FEMELLES ;
```

- **Division cartésienne**

Soient les deux tables suivantes désignant pour l'une l'ensemble des lâchés machines des pilotes du club (Nom pilote, Identification Avion) et l'autre regroupant l'ensemble des avions du club :

Lachers	
Nom	**Avion**
DEFAUCHY	XH
SMITH	RL
DEFAUCHY	RL
DEFAUCHY	PQ
AMAR	XH
OLAF	PQ
SMITH	PQ
SMITH	XH

Avions	
Avion	**Type**
XH	DR420
RL	DR480
PQ	Sportstar

On cherche les noms des pilotes étant lâchés sur tous les avions.

```
SELECT Nom FROM Lachers GROUP BY Nom HAVING COUNT(*) = (SELECT COUNT(*)
FROM Avions) ;
```

On cherche dans la table « Lachers » regroupée par noms de pilotes ceux qui ont un nombre distinct d'avions égal au nombre distinct d'avions de la table « Avions ».

Oui, si vous avez bien compris, en réalité le nom des avions de la table « Avions » n'a pas d'importance. Si un pilote de la table « Lachers » est lâché sur 3 avions différents et s'il y a 3 avions différents au club, alors il ce pilote sera inclus dans le résultat de la requête.

Il faut donc veiller à ne pas faire n'importe quoi et bien définir les deux tables ensemble avec des données cohérentes.

Le résultat de la requête proposée sera « DEFAUCHY - SMITH »

A.VII.6.a.iv Fonctions d'agrégation

Voyons dans le tableau suivant quelques exemples de ce que l'on peut faire :

Minimum MIN()	Age min des élèves : `SELECT MIN(Age) FROM Eleves ;`
Maximum MAX()	Age max des élèves : `SELECT MAX(Age) FROM Eleves ;`
Somme SUM()	Somme des âges des élèves : `SELECT SUM(Age) FROM Eleves ;`
Moyenne AVG()	Moyenne des âges des élèves : `SELECT AVG(Age) FROM Eleves ;`

Comptage
COUNT
	Nombre d'élèves :
	`SELECT COUNT(*) FROM Eleves ;`
	Nombre délèves de 17 ans :
	`SELECT COUNT(*) FROM Eleves WHERE Age = 17 ;`

Pour utiliser ou afficher le résultat, on écrit après la ligne `cursor.execute("""…""")` :

```
Res = cursor.fetchone()
print(Res[0])
```

A.VII.7 Création d'une nouvelle bdd avec les résultats d'une requête

Voici comment créer une nouvelle relation basée sur le résultat de la requête réalisée. Voici le code permettant de créer une nouvelle relation basée sur les résultats de la requête réalisée (on pourra utiliser ce code pour toutes les requêtes et pas uniquement dans le cadre des unions, intersections et différence) :

```
# Opération d'union, différence, intersection
cursor.execute("""SELECT … FROM … WHERE … UNION SELECT … FROM … WHERE …
;""")
Res_OP = cursor.fetchall()
# Nouvelle base de données
cursor.execute("""CREATE TABLE T_OP(Ref Text,Prix Integer);""")
cursor.executemany("""INSERT INTO TUnion VALUES (?,?);""",Res_OP)
# Affichage du résultat
cursor.execute("""SELECT * FROM T_OP;""")
print(cursor.fetchall())
```

HAVE FUN WITH PYTHON