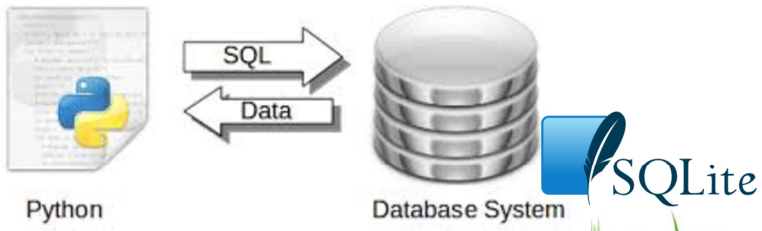
Les Bases de données relationnelles

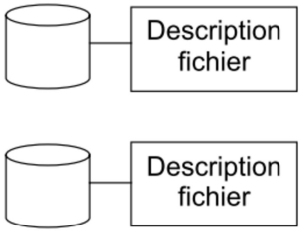


Motivations : Des fichier au BDD

Séparation des données et des programme

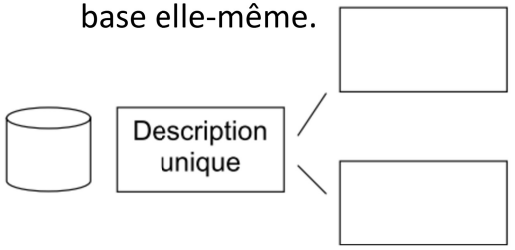
Fichier

Les données des fichiers sont décrites dans les programmes



Base de données

Les données de la BD sont décrites hors des programmes dans la base elle-même.



Base de données : concepts

Définition

Une base de données relationnelle est une base de données structurée suivant les principes de l'algèbre relationnelle.

Éléments 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.**

Base de données : éléments

Attribut ou colonne

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

tuple ou n-uplet

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

Modèle relationnel : Clé primaire

Définition :

- Une clé primaire d'une relation est un ensemble minimal des attributs de la relation dont les valeurs identifient un tuple unique.

Propriétés :

- 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 un et un seul clé primaire
- Par convention, on représente la clé primaire en la soulignant dans l'énoncé de la relation

Modèle relationnel : Clé primaire

Exemple :

Quels sont les clé primaires des relations ?

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

Modèle relationnel : Clé étrangère

Définition :

- Une clé étrangère ou un clé secondaire est un ensemble d'attributs qui fait référence à la clé primaire d'une autre relation.

Clé étrangère = clé primaire dans une autre relation

Propriétés :

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

Modèle relationnel : Clé étrangère

Exemple :

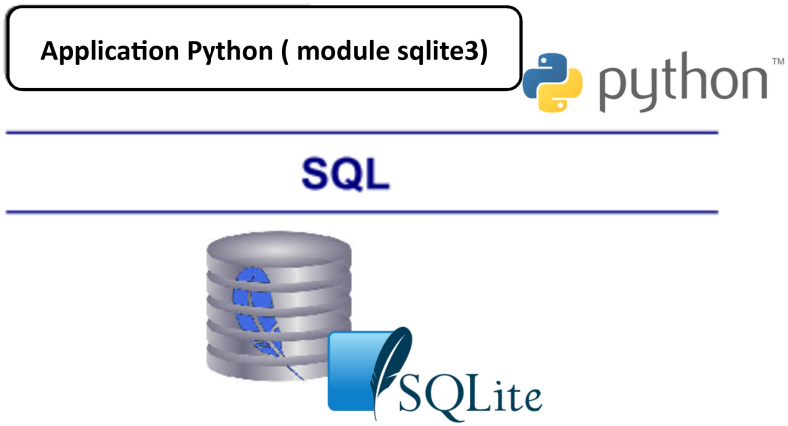
Quels sont les clé étrangère des relations ?

- Etudiant (**<u>numero</u>** , nom , prénom , adresse, âge, *département*)
  - l'attribut département est un clé étrangère qui fait référence au clé primaire de la relation Département
- Département (**<u>nom</u>** , *chef*)
  - l'attribut chef est un clé étrangère qui fait référence au clé primaire de la relation Enseignant
- Enseignant (nom , prénom , **<u>CIN</u>**)
  - La relation enseignant ne possède pas de clé étrangère
- Module (**<u>réf_module</u>**, *responsable* , *département*)
  - l'attribut responsable est un clé étrangère qui fait référence au clé primaire de la relation Enseignant
  - l'attribut département est un clé étrangère qui fait référence au clé primaire de la relation Département

SQL
(structured Query Language)

Introduction

- SQL est l'interface logiciel/logiciel entre les applications et les BDR.



Introduction

- SQL est un langage standard pour décrire et manipuler les BDD
- Les commandes SQL :
  - Définition des données : LDD
    - **CREATE , DROP , ALTER**
  - Manipulation des données : LMD
    - **SELECT**
    - **INSERT , UPDATE , DELETE**

LDD : Création de Table

La commande CREATE TABLE crée la définition d'une table

Syntaxe :

```
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 )
```

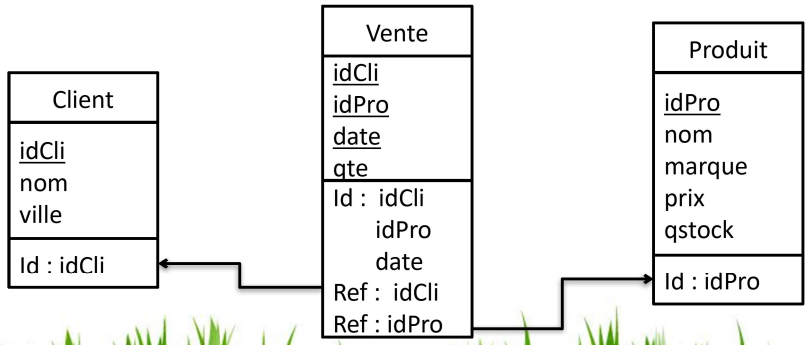
LDD : Création de Table

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

LDD : Création de Table

Soit les relations suivantes :

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



LDD : création de Table

```
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,
    marque      CHAR(30) ,
    prix        DEC(6,2) ,
    qstock      SMALLINT CHECK (qstock BETWEEN 0 AND 100)
    )
```


LDD : création de Table

```
CREATE TABLE vente (
   IdCli    CHAR(4)    NOT NULL REFERENCES client ,
   IdPro    CHAR(6)    NOT NULL ,
   vdate    DATE        NOT NULL ,
   qte          SMALLINT CHECK (qte BETWEEN 1 AND 10) ,
   -- contrainte de table
   PRIMARY KEY (IdCli, IdPro, date) ,
   FOREIGN KEY (IdPro) REFERENCES produit
   )
```

LDD : Suppression / Modification de Table

- **Suppression d'une table** : Toute table peut être supprimée :

  drop table Vente

- **Ajout d'une colonne :**

  ALTER TABLE client ADD COLUMN tel CHAR(16) default '0000'

  – 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**).

LMD : Manipulation des données

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

LMD : Insertion de Lignes

La commande **INSERT** permet d'ajouter de nouvelles lignes à une table

Syntaxe :

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

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

Exemple : Ajouter le client ('TN100', 'Ali', Tunis') dans la table client

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

LMD : Modification de données

La commande **UPDATE** permet de changer des valeurs d'attributs de lignes existantes.

Syntaxe :

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

LMD : Modification de données

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'
```


LMD : Suppression de données

La commande **DELETE** permet d'enlever des lignes dans une table

Syntaxe :

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

Remarque:

– L'absence de clause WHERE signifie que toutes les lignes de la table cible sont enlevées

Exemple:

Supprimer les ventes antérieures au 01-jan-2021 :

```
DELETE
FROM vente
WHERE date < '01-jan-2021'
```


LMD : Extraction de données

La commande **SELECT** 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
```


LMD : Extraction de données

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

LMD : Extraction de données avec condition

Q5 : Donner les noms des produits de marque IBM

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

- La condition de recherche (qualification) 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**

LMD : Extraction de données avec condition

Q6 : Lister les clients dont le nom comporte la lettre A en 2 ième position

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

- Le prédicat **LIKE** compare une chaîne avec un modèle
  - (_) remplace n'importe quel caractère (tiret de8: under score)
  - (%) 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

LMD : Extraction de données avec condition

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

LMD : Extraction de données avec condition

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...

LMD : Expression des jointures

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
```

- La condition de jointure est exprimée après WHERE
- La jointure permet de produire une table constituée de données extraites de plusieurs tables
- On peut utilser aussi les mots clés **JOIN … ON :**

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


LMD : Expression des jointures

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'
```


LMD : Expression des jointures

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 = 'martin'
AND C2.nom <> 'martin'
```

- 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.

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

LMD : Les Sous-requêtes

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'
```


LMD : Les Sous-requêtes

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 predicat EXISTS permet de tester si le résultat d'une sous-requête est vide ou non

LMD : Opérations ensemblistes

Q20 : Donner les numeros des produits de marque IBM ou ceux achetés par le client d'identifiant '100'

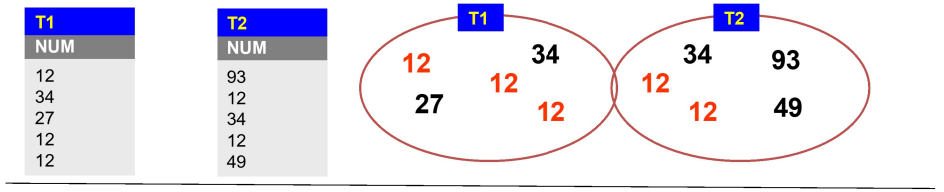
```
SELECT P.IdPro
        FROM produit P
        WHERE P.marque = 'IBM'
        UNION
        SELECT V.IdPro
        FROM vente V
        WHERE V.IdCli = '100'
```

– SQL permet d'exprimer l'opération d'union en connectant des SELECT par des UNION

– L'union élimine les doublons,

LMD : Opérations ensemblistes

- Opérateurs ensemblistes entre 2 tables :




```
select
NUM
from  T1
union
select
NUM
from  T2
```

NUM: 12, 34, 27, 93, 49

```
select NUM
from  T1
intersect
select NUM
from  T2
```

NUM: 34, 12

```
select
NUM
from  T1
except
select
NUM
from  T2
```

NUM: 27

```
select
NUM
from  T2
except
select
NUM
from  T1
```

NUM: 93, 49

LMD : Fonctions de calculs

- 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

Q21 : Donner le nombre total de clients

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


LMD : Fonctions de calculs

Q22 : 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

Q23 : Donner la quantité totale de 'PS1' vendus

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


LMD : Fonctions de calculs

Q24 : 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

LMD : 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

Q25 : Donner pour chaque référence de produit la quantité totale vendue

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

LMD : la clause Group By… Having

Q26 : 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
```

- 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

LMD : Forme générale de SELECT

SELECT [DISTINCT] liste d'attributs, expressions
FROM liste de tables
WHERE qualification
GROUP BY attributs de partitionnement
HAVING qualification de groupe
ORDER BY liste de colonnes [ASC | DESC]

Algèbre relationnelle

L'algèbre relationnelle

- L'algèbre relationnelle est une collection d'opérations
- **Opérations :**
  - Opérandes : 1 ou 2 relations
  - Résultat : une relation
- **Deux types d'operations :**
  - Opération ensemblistes :
    - Union , intersection, différence
  - Opérations spécifiques :
    - Projection, restriction, jointure

L'opération **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 ∪ R2**

R1 :

A	B
0	1
2	3

R2 :

A	B
0	1
4	5

R3 :

A	B
0	1
2	3
4	5

L'opération **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 ∩ R2**

R1 :

A	B
0	1
2	3

R2 :

A	B
0	1
4	5

R3 :

A	B
0	1

L'opération **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**

R1 :

A	B
0	1
2	3

R2 :

A	B
0	1
4	5

R3 :

A	B
2	3

L'opération **PROJECTION**

- La projection d'une relation R1 est la relation R2 obtenue en supprimant les attributs de R1 non mentionnés.
- On notera : **R2 = πR1 (Ai, Aj, ... , Am)**

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

A1	A2	A3	A4	A5

- **Exemple :**
  - Quels sont les références et les prix des produits ?

    πPRODUIT (IdPro, Prix)

L'opération **RESTRICTION**

- 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 = σR1 (condition)**

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

A1	A2	A3	A4	A5

- **Exemple :**
  - Quelles sont les produits de marque 'IBM' ?

    σPRODUIT (Marque = 'IBM')

L'opération **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 × R2 (condition)**
- **Remarques :**
  - Le schéma de la relation résultat 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 × 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
  - La condition de liaison doit être du type : <attribut1> :: <attribut2>

    où : attribut1 ∈ 1ère relation et attribut2 ∈ 2ème relation

    :: est un opérateur de comparaison (égalité ou inégalité)

L'opération **JOINTURE**

Exemple :

R1(A, B, C)

A	B	C
A1	B1	10
A2	B2	10
A3	B3	20
A4	B4	30

R2(U , V)

U	V
10	V1
20	V2
30	V3

R1 × R2 (R1.C = R2.U)

A	B	C	U	V
A1	B1	10	10	V1
A2	B2	10	10	V1
A3	B3	20	20	V2
A4	B4	30	30	V3

Exemples :

1. **Donner les no des produits de marque Apple et de prix < 2000 ?**

   R1 = σPRODUIT (marque = Apple')

   R2 = σPRODUIT (prix < 2000)

   R3 = R1∩R2

   RESUL = πR3 (IdPro)

2. **Donner les no des clients ayant acheté un produit de marque Apple ?**

   R1 = σPRODUIT (marque = 'Apple')

   R2 = R1×VENTE (R1.IdPro = VENTE.IdPro)

   RESUL = πR2 (IdCli)

3. **Donner les no des clients n'ayant acheté que des produits de marque Apple?**

   R1 = VENTE×PRODUIT (VENTE.IdPro = PRODUIT.IdPro)

   R2 = σR1 (marque = 'Apple')

   R3 = πR2 (IdCli)

   R4 = σR1 (marque ≠ 'Apple')

   R5 = πR4 (IdCli)

   RESUL = R3 − R5

Base de donnée sous Python :

SQLITE



Instructions de base sous Python

1. La librairie qu'il faut utiliser est ***sqlite3.***

```
import sqlite3
```

2. Les bases créées avec cette librairie sont enregistrées sous forme de fichier .db, .sq3, etc.
   Un fichier .db/.sq3 s'ouvre en créeant un objet de type ***Connection***

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

3. Un curseur est ensuite crée à partir de l'objet Connection. Ce curseur represente un canal entre la base de donnée et le script Python.

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


Instructions de base sous Python

4. 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)
```

5. Lorsque le travail que vous attendiez est terminé, pensez à enregistrer les modification faite sur la base.

```
conn.commit()
```

6. Pensez enfin à fermer la connexion vers la base.

```
conn.close()
```


Instructions de base sous Python

Exemple Complet:

```
import sqlite3

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

cur = conn.cursor()

requete = ' SELECT * FROM vente '

cur.execute(requete)

conn.commit()

conn.close()
```

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)

Les fonctions **execute()** et **executemany()**

- Il existe plusieurs manière d'insérer des données:
  - la plus simple étant celle-ci:

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

  - Insérer des champs précis:

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

  - Ou encore tous les champs avec:

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

  - Ajouter automatiquement un client lorsqu'on a perdu le fil du compte des id:

```
cursor.execute
'''INSERT INTO client VALUES (?,?,?);''', ((cursor.lastrowid + 1), "chebbi", "Mahdia")
        )
```


Les fonctions **execute()** et **executemany()**

- On peut passer un dictionnaire:

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

- il est également de faire plusieurs insertions en une seule fois avec la fonction executemany

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


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.
- Vous pouvez récupérer la première ligne correspondant à votre 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')
```

- Vous pouvez 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)
```