Université Internationale de Casablanca – MIAGE 1-

INITIATION À SQL

Mr Mohamed EL KASSIMI

SQL

- SQL = Langage de définition de données
  - CREATE TABLE
  - ALTER TABLE
  - DROP TABLE
- SQL = Langage de manipulation de données
  - INSERT INTO
  - UPDATE
  - DELETE FROM
- SQL = Langage de requêtes
  - SELECT … FROM … WHERE ...
    - Sélection
    - Projection
    - Jointure
  - Les agrégats

Introduction

- **SQL** : **S**tructured **Q**uery **L**anguage
- Inventé chez IBM (centre de recherche d'Almaden en Californie), en 1974 par Astrahan & Chamberlin dans le cadre de System R
- Le langage SQL est normalisé
  - SQL2: adopté (SQL 92)
  - SQL3: adopté (SQL 99)
- Standard d'accès aux bases de données relationnelles

SQL : Trois langages en un

- Langage de définition de données (LDD/DDL)
  - création de relations : CREATE TABLE
  - modification de relations: ALTER TABLE
  - suppression de relations: DROP TABLE
  - vues, index …. : CREATE VIEW ...
- Langage de manipulation de données (LMD /DML)
  - insertion de tuples: INSERT
  - mise à jour des tuples: UPDATE
  - suppression de tuples: DELETE
- Langage de requêtes (LMD/DML)
  - SELECT ……. FROM ..... WHERE .....

Terminologie

- Relation → Table
- Tuple → Ligne
- Attribut → Colonne

SQL

Un langage de définition de données

Types de données

- Une base de données contient des **tables**
- Une table est organisée en **colonnes**
- Une colonne stocke des **données**

- Les données sont séparées en plusieurs **types** !

Type des colonnes (en MySQL)

- Numériques
  - NUMERIC : idem DECIMAL
  - DECIMAL. Possibilité DECIMAL(M,D) M chiffre au total
  - INTEGER
    - TINYINT 1 octet (de -128 à 127)
    - SMALLINT 2 octets (de -32768 à 32767
    - MEDIUMINT 3 octets (de -8388608 à 8388607)
    - INT 4 octets (de -2147483648 à 2147483647)
    - BIGINT 8 octets (de -9223372036854775808 à 9223372036854775807)
    - Possibilité de donner la taille de l'affichage : INT(6) => 674 s'affiche 000674
    - Possibilité de spécifier UNSIGNED
      - INT UNSIGNED => de 0 à 4294967296
  - FLOAT : 4 octets par défaut. Possibilité d'écrire

Type des colonnes (en MySQL)

- Date et Heure
  - DATETIME
    - AAAA-MM-JJ HH:MM:SS
    - de 1000-01-01 00:00:00 à '9999-12-31 23:59:59
  - DATE
    - AAAA-MM-JJ
    - de 1000-01-01 à 9999-12-31
  - TIMESTAMP
    - Date sans séparateur AAAAMMJJHHMMSS
  - TIME
    - HH:MM:SS (ou HHH:MM:SS)
    - de -838:59:59 à 838:59:59
  - YEAR
    - YYYY

Type des colonnes (en MySQL)

- Chaînes
  - CHAR(n) $1 \leq n \leq 255$
  - VARCHAR(n) $1 \leq n \leq 255$

Exemple :

	CHAR(4)		VARCHAR(4)	
Valeur	Stockée	Taille	Stockée	Taille
''	' '	4 octets	''	1 octets
'ab'	'ab '	4 octets	'ab'	3 octets
'abcd'	'abcd'	4 octets	'abcd'	5 octets

Type des colonnes (en MySQL)

- Chaînes
  - TINYBLOB Taille < 2^8 caractères
  - BLOB Taille < 2^8 caractères
  - MEDIUMBLOB Taille < 2^24 caractères
  - LONGBLOB Taille < 2^32 caractères

  - TINYTEXT Taille < 2^8 caractères
  - TEXT Taille < 2^8 caractères
  - MEDIUMTEXT Taille < 2^24 caractères
  - LONGTEXT Taille < 2^32 caractères

Les tris faits sur les BLOB tiennent compte de la casse, contrairement aux tris faits sur les TEXT.

Type des colonnes (en MySQL)

- ENUM
  - Enumération
  - ENUM("un", "deux", "trois")
  - Valeurs possibles : "" , "un", "deux", "trois"
  - Au plus 65535 éléments

- SET
  - Ensemble
  - SET("un", "deux")
  - Valeurs possibles : "" , "un", "deux", "un,deux"
  - Au plus 64 éléments

Type des colonnes (en MySQL)

- Dans quelles situations faut-il utiliser ENUM ou SET ?

JAMAIS !!

- il faut toujours éviter autant que possible les fonctionnalités propres à un seul SGBD.

Un langage de définition de données

Commandes pour Créer et supprimer une base de données

- CREATE DATABASE : créer une base de données,
- CREATE DATABASE bibliotheque CHARACTER SET 'utf8' : créer une base de données et encoder les tables en UTF-8
- DROP DATABASE bibliotheque : supprimer la base de données,
- DROP DATABASE IF EXISTS bibliotheque ;

Utilisation d'une base de données

- USE bibliotheque ;

Un langage de définition de données

- Commandes pour créer, modifier et supprimer les éléments du schéma
- CREATE TABLE : créer une table (une relation),
- CREATE VIEW : créer une vue particulière sur les données à partir d'un SELECT,
- DROP {TABLE ¦ VIEW } : supprimer une table ou une vue,
- ALTER {TABLE ¦ VIEW } : modifier une table ou une vue.

CREATE TABLE

Commande créant une table en donnant son nom, ses attributs et ses contraintes

```
CREATE TABLE [IF NOT EXISTS]  nom_table (
 colonne1 description_colonne1,
    [colonne2 description_colonne2,
    colonne3 description_colonne3,
    ...,]
    [PRIMARY KEY (colonne_clé_primaire)]
)
[ENGINE=moteur];
```

Les moteurs de tables

Les moteurs de tables sont une spécificité de MySQL. Ce sont des moteurs de stockage. Cela permet de gérer différemment les tables selon l'utilité qu'on en a.

Les deux moteurs les plus connus sont **MyISAM** et **InnoDB**.

MyISAM : C'est le moteur par défaut. Les commandes sont particulièrement rapides sur les tables utilisant ce moteur. Cependant, il ne gère pas certaines fonctionnalités importantes comme les clés étrangères.

InnoDB : Plus lent et plus gourmand en ressources que MyISAM, ce moteur gère les clés étrangères

CREATE TABLE

Exemples:

```
CREATE TABLE Emprunteur(
 id SMALLINT UNSIGNED NOT NULL
  AUTO_INCREMENT,
 nom VARCHAR(20) NOT NULL,
 prenom VARCHAR(15) NOT NULL,
 annee_insc YEAR DEFAULT 2018,
 PRIMARY KEY (id)
)
ENGINE=INNODB;
```

Vérifications

Deux commandes pour vérifier la création des tables :

SHOW TABLES; **-- liste les tables de la base de données**

DESCRIBE Emprunteur; **-- liste les colonnes de la table avec leurs caractéristiques**

DROP TABLE

- DROP TABLE : Supprimer une table
  - supprime la table et tout son contenu

- DROP TABLE nom_table [CASCADE CONSTRAINTS];

- CASCADE CONSTRAINTS
  - Supprime toutes les contraintes référençant une clé primaire (primary key) ou une clé unique (UNIQUE) de cette table
  - Si on cherche à détruire une table dont certains attributs sont référencés sans spécifier CASCADE CONSTRAINT, on a un message d'erreur.

ALTER TABLE

- Modifier la définition d'une table:
  - Changer le nom de la table

    mot clé : RENAME
  - Ajouter une colonne ou une contrainte

    mot clé : ADD
  - Modifier une colonne ou une contrainte

    mot clé : MODIFY
  - Supprimer une colonne ou une contrainte

    mot clé : DROP
  - renommer une colonne ou une contrainte

    mot clé : RENAME

ALTER TABLE

Syntaxe :

```
ALTER TABLE nom-table
{   RENAME TO nouveau-nom-table
    | ADD (( nom-col type-col [DEFAULT valeur] [contrainte-col])*)
    | MODIFY (nom-col [type-col] [DEFAULT valeur] [contrainte-col])*
    | DROP COLUMN nom-col [CASCADE CONSTRAINTS]
    | RENAME COLUMN old-name TO new-name
};
```

Ajout et suppression d'une colonne

```
ALTER TABLE nom_table
ADD [COLUMN] nom_colonne description_colonne;
```

- Exemple :

```
ALTER TABLE Emprunteur
ADD  COLUMN  date_emprunt DATE NOT NULL ;
```

Ajout et suppression d'une colonne

```
ALTER TABLE nom_table
DROP [COLUMN] nom_colonne;
```

- Exemple :

```
ALTER TABLE Emprunteur
DROP  COLUMN  date_emprunt ;
```

Modification d'une colonne

```
ALTER TABLE nom_table
CHANGE ancien_nom nouveau_nom description_colonne;
```

- Exemple :

```
ALTER TABLE Emprunteur
CHANGE  nom nom_famille VARCHAR(10) NOT NULL ;
```

Changement du type de données

ALTER TABLE nom_table
CHANGE ancien_nom nouveau_nom description_colonne;

Ou

ALTER TABLE nom_table
MODIFY nom_colonne description_colonne;

Des exemples pour illustrer :

```
ALTER TABLE Emprunteur
CHANGE  nom nom_famille VARCHAR(10) NOT NULL ;
```

–> Changement du type + changement du nom

```
ALTER TABLE Emprunteur
CHANGE  id id BIGINT NOT NULL ;
```

–> Changement du type sans renommer

```
ALTER TABLE Emprunteur
MODIFY  id BIGINT NOT NULL AUTO_INCREMENT;
```

–> Ajout de l'auto-incrémentation

```
ALTER TABLE Emprunteur
```

Renommer une table

- ... RENAME TO nouveau-nom-table

- Exemple :

ALTER TABLE Emprunteur RENAME TO Emprunteurs ;

Les clé étrangères

```
CREATE TABLE [IF NOT EXISTS] Nom_table (
    colonne1 description_colonne1,
    [colonne2 description_colonne2,
    colonne3 description_colonne3,
    ...,]
    [ [CONSTRAINT [symbole_contrainte]]  FOREIGN KEY
    (colonne(s)_clé_étrangère) REFERENCES
    table_référence (colonne(s)_référence)]
)
[ENGINE=moteur];
```

Exemple

On imagine les tables Client et Commande, pour créer la table Commande avec une clé étrangère ayant pour référence la colonne numero de la table Client, on utilisera :

```
CREATE TABLE Commande (
    numero INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    client INT UNSIGNED NOT NULL,
    produit VARCHAR(40),
    quantite SMALLINT DEFAULT 1,
    CONSTRAINT fk_client_numero          -- On donne un nom à notre clé
    FOREIGN KEY (client)                 -- Colonne sur laquelle
```

Après création de la table

```
ALTER TABLE Commande
ADD CONSTRAINT fk_client_numero FOREIGN KEY (client)
    REFERENCES Client(numero);
```

Suppression d'une clé étrangère

```
ALTER TABLE nom_table
DROP FOREIGN KEY symbole_contrainte
```

Petit TP

- Prenez le MLD de l'exercice 1 du dernier TP et créer une base de données et les différentes tables !

SQL

Un langage de manipulation de données

Manipulation des données

- **INSERT INTO** : ajouter un tuple dans une table ou une vue
- **UPDATE** : changer les tuples d'une table ou d'une vue
- **DELETE FROM** : éliminer les tuples d'une table ou d'une vue

INSERT INTO

- Syntaxe :

INSERT INTO
{nom_table ¦ nom_vue}
[(nom_col (, nom_col)*)]
{ VALUES (valeur (, valeur)*) ¦ sous-requête };

Insertion sans préciser les colonnes

- Nous travaillons toujours sur la table Emprunteur composée de 4 colonnes : id, nom, prenom, annee_insc

```
INSERT INTO Emprunteur
VALUES (1, 'Buard', 'Jeremy', '2018');
```

```
INSERT INTO Emprunteur
VALUES (NULL, 'Zuckerberg', 'Mark', NULL);
```

–> Insert un tuple avec un id=2 et une année = NULL

Insertion en précisant les colonnes

```
INSERT INTO Emprunteur (nom, prenom, annee_insc)
VALUES ('Chan', 'Priscilla', '2018');

INSERT INTO Emprunteur (nom, prenom)
VALUES ('Gates', 'Bill');
```

–> Insert un tuple avec une année = 2018

Insertion multiple

```
INSERT INTO Emprunteur (nom, prenom, annee_insc)
VALUES ('Jobes', 'Steve', '2010'),
       ('Moskovitz', 'Dustin', '2011'),
        ('Musk', 'Elon', '2013');
```

UPDATE

- Exemples :
  - UPDATE Emprunteur
    SET annee_insc = '2019'
    WHERE nom = 'Musk'

  - UPDATE Emprunteur
      SET annee_insc = annee_insc+2
    WHERE id < 3
- Syntaxe :
  - UPDATE {nom_table ¦ nom_vue}
    SET { (nom_col)* = (sous-requête)
        ¦ nom_col = { valeur ¦ (sous-requête)} }*

DELETE FROM

- Exemple :
  - DELETE FROM Emprunteur
    WHERE annee_insc < 2000

- Syntaxe :
  - DELETE FROM {nom_table ¦ nom_vue}
    WHERE condition;

Petit TP suite

- Insérer des tuples à l'aide des commandes INSERT INTO que nous venons de voir !!

SQL

Un langage de requêtes

Structure générale d'une requête

- Structure d'une requête formée de trois clauses:

  SELECT <liste_attributs>

  FROM <liste_tables>

  WHERE <condition>

- **SELECT** définit le format du résultat cherché
- **FROM** définit à partir de quelles tables le résultat est calculé
- **WHERE** définit les prédicats de sélection du résultat

Exemple de requête

SELECT * FROM Emprunteur

Afficher tous les attributs de tous les tuples dans la table “Emprunteur”

Opérateurs de comparaison

- = égal
  - WHERE id = 2
- <> différent
  - WHERE nom <> 'Buard'
- > plus grand que
  - WHERE annee_insc > 2010
- >= plus grand ou égal
  - WHERE annee_insc >= 2018
- < plus petit que
  - WHERE id < 3
- <= plus petit ou égal
  - WHERE id <= 2

Opérateurs logiques

- **AND**
  - WHERE annee_insc < 2010 **AND** id<5
- **OR**
  - WHERE annee_insc < 2010 **OR** id<5

- Négation de la condition : **NOT**
  - SELECT *
    FROM Emprunteur
    WHERE nom = 'Buard'
    AND NOT annee_insc = '2019' ;

Expressions logiques

Combinaisons:

WHERE
(ensoleillement > 80 **AND** pluviosité < 200 **)**
OR température > 30

WHERE
ensoleillement > 80
AND (pluviosité < 200 **OR** température > 30 **)**

Appartenance à un ensemble : IN

WHERE monnaie = 'Pound'
OR monnaie = 'Schilling'
OR monnaie = 'Euro'

Équivalent à:

WHERE monnaie **IN** ('Pound', 'Schilling', 'Euro')

NOT IN: non appartenance à un ensemble

Comparaison à un ensemble : ALL

```
SELECT * FROM Employe
WHERE salaire >= 1400
AND salaire >= 3000 ;
```

Équivalent à:

```
SELECT * FROM Employe
WHERE salaire >= ALL ( 1400, 3000);
```

Valeur dans un intervalle : BETWEEN

WHERE population >= 50 **AND** population <= 60

Équivalent à:

WHERE population **BETWEEN** 50 **AND** 60

NOT BETWEEN

Conditions partielles (joker)

- % : un ou plusieurs caractères
  - WHERE nom LIKE '%uard'
  - WHERE prenom LIKE '%erem%'
- _ : exactement un caractère
  - WHERE nom LIKE 'B_ard'
- **NOT LIKE**

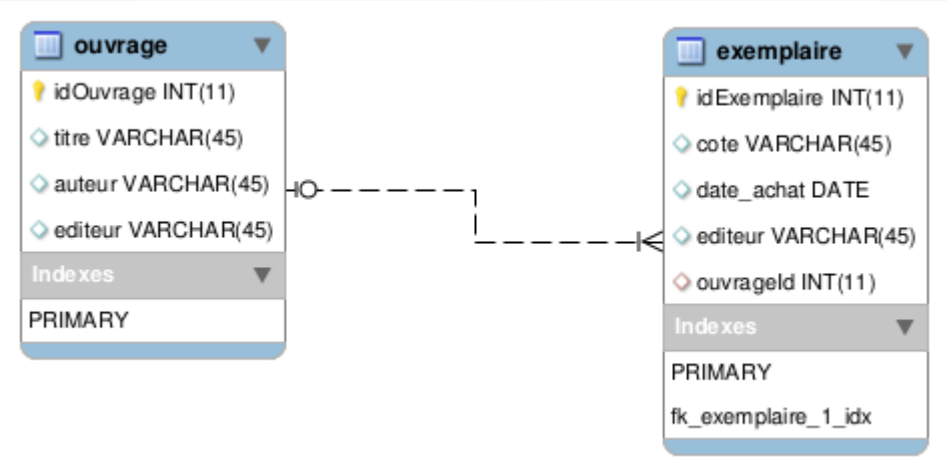
Valeurs calculées

- SELECT nom, population, surface, natalité
  FROM Pays
  WHERE (population * 1000 / surface) < 50
  AND (population * natalité / surface) > 0

- SELECT nom, (population * 1000 / surface)
  FROM Pays

Les jointures

- Principe :
  - Joindre plusieurs tables
  - On utilise les informations communes des tables



Les jointures

- Prenons pour exemple un ouvrage de V. Hugo
- Si l'on souhaite des informations sur la cote d'un exemplaire il faudrait le faire en 2 temps
  - 1) je récupère l'id de l'ouvrage :

SELECT id FROM ouvrage where auteur LIKE 'V. Hugo'

  - 2) Je récupère la ou les cote avec l'id récupéré

SELECT cote FROM exemplaire WHERE ouvrageId = id_récupéré

Les jointures

Ne serait-ce pas merveilleux de pouvoir faire tout ça (et plus encore) en une seule requête ?

C'est là que les jointures entrent en jeu

```
SELECT exemplaire.cote
FROM exemplaire
INNER JOIN ouvrage
        ON exemplaire.ouvrageId = ouvrage.idOuvrage
WHERE ouvrage.auteur LIKE 'V. Hugo' ;
```

Remarques

- Le résultat d'une requête peut contenir plusieurs occurrences d'un tuple,
  - pour avoir une seule occurrence de chaque n-uplet dans une relation : DISTINCT
  - Exemple : select DISTINCT nom FROM Personne
- Le résultat d'une requête peut être trié,
- Il existe une valeur spéciale dite indéfinie (NULL) utilisée pour remplir un champ dont on ne connait pas la valeur.

Remarques

- En SQL, le produit cartésien est possible sans renommer les attributs communs.
  - Exemple : schéma(RxS) = A (de R), B (de R), B (de S), C (de S).

- En SQL, si plusieurs attributs ont le même nom, pour résoudre l'ambiguité, on spécifie la relation auquel l'attribut appartient.
  - Exemple : SELECT A, R.B, C FROM R, S

Requêtes

avec blocs emboîtés

BD exemple

- Produit(np,nomp,couleur,poids,prix) *les produits*
- Usine(nu,nomu,ville,pays) *les usines*
- Fournisseur(nf,nomf,type,ville,pays) *les fournisseurs*
- Livraison(np,nu,nf,quantité) *les livraisons*
  - *np référence Produit.np*
  - *nu référence Usine.nu*
  - *nf référence Fournisseur.nf*

Jointure par blocs emboîtés

Nom et couleur des produits livrés par le fournisseur 1

- Solution 1 : la jointure déclarative

```
SELECT nomp, couleur FROM Produit,Livraison
WHERE (Livraison.np = Produit.np) AND nf = 1 ;
```

- Solution 2 : la jointure procédurale (emboîtement)

Nom et couleur des produits livrés par le fournisseur 1

```
SELECT nomp, couleur FROM Produit
WHERE np IN
    (SELECT np FROM Livraison WHERE nf = 1) ;
```

Numéros de produits livrés par le fournisseur 1

Jointure par blocs emboîtés

- SELECT nomp, couleur FROM Produit
  WHERE np **IN**
  (SELECT np FROM Livraison
  WHERE nf = 1) ;
- **IN** compare chaque valeur de np avec l'ensemble (ou multi-ensemble) de valeurs retournés par la sous-requête
- **IN** peut aussi comparer un tuple de valeurs:

SELECT nu FROM Usine
WHERE (ville, pays)
IN (SELECT ville, pays FROM Fournisseur)

Composition de conditions

Nom des fournisseurs qui approvisionnent une usine de Londres ou de Paris en un produit rouge

```
SELECT nomf
FROM Livraison, Produit, Fournisseur, Usine
WHERE
   couleur = 'rouge'
   AND Livraison.np = Produit.np
   AND Livraison.nf = Fournisseur.nf
   AND Livraison.nu = Usine.nu
   AND (Usine.ville = 'Londres'
  OR Usine.ville = 'Paris');
```

Composition de conditions

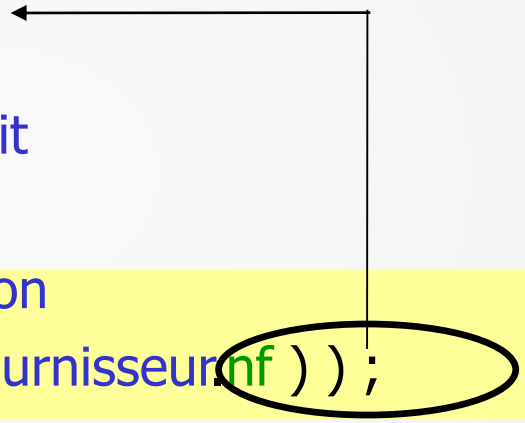
Nom des fournisseurs qui approvisionnent une usine de Londres ou de Paris en un produit rouge

```
SELECT nomf FROM Fournisseur
WHERE nf IN
(SELECT nf FROM Livraison
   WHERE np IN
    (SELECT np FROM Produit
          WHERE couleur = 'rouge')
   AND nu IN
    (SELECT nu FROM Usine
       WHERE ville = 'Londres' OR ville = 'Paris') ) ;
```

Quantificateur ALL

- *Numéros des fournisseurs qui ne fournissent que des produits rouges*

```
SELECT nf FROM Fournisseur
WHERE 'rouge' = ALL
      (SELECT couleur FROM Produit
       WHERE np IN
           (SELECT np FROM Livraison
            WHERE Livraison.nf = Fournisseur.nf ) ) ;
```



- *La requête imbriquée est ré-évaluée pour chaque tuple de la requête (ici pour chaque nf)*
- *ALL: tous les éléments de l'ensemble doivent vérifier la condition*

Condition sur des ensemble : EXISTS

- Test si l'ensemble n'est pas vide (E ≠ ∅)
- Exemple : *Noms des fournisseurs qui fournissent au moins un produit rouge*

```
SELECT nomf
FROM Fournisseur
WHERE EXISTS
   ( SELECT *
     FROM Livraison, Produit
     WHERE Livraison.nf = Fournisseur.nf
    AND Livraison.np = Produit.np
    AND Produit.couleur = 'rouge' );
```

ce fournisseur

Le produit fourni est rouge

Blocs emboîtés - récapitulatif

SELECT ...

FROM ...

WHERE ...

attr IN requête

attr NOT IN requête

attr opérateur ALL reqûete

EXISTS requête

NOT EXISTS requête

Traitement

des résultats

Fonctions sur les colonnes

- Attributs calculés
  - Exemple : SELECT nom, population*1000/surface FROM Pays

- Opérateurs sur attributs numériques
  - SUM: somme des valeurs des tuples sélectionnés
  - AVG: moyenne

- Opérateurs sur tous types d'attributs
  - MIN: minimum
  - MAX: maximum
  - COUNT: nombre de tuples sélectionnés

Opérateurs d'agrégation

Opérateurs d'agrégation

pays

Nom	Capitale	Population	Surface	Continent
Irlande	Dublin	5	70	Europe
Autriche	Vienne	10	83	Europe
UK	Londres	50	244	Europe
Suisse	Berne	7	41	Europe
USA	Washington	350	441	Amérique

SELECT **MIN**(population), **MAX**(population), **AVG**(population), **SUM**(surface), **COUNT**(*)
FROM Pays WHERE continent = 'Europe'

Donne le résultat :

MIN(population)	**MAX**(population)	**AVG**(population)	**SUM**(surface)	**COUNT**(*)
5	50	18	438	4

DISTINCT

pays

Nom	Capitale	Population	Surface	Continent
Irlande	Dublin	5	70	Europe
Autriche	Vienne	10	83	Europe
UK	Londres	50	244	Europe
Suisse	Berne	7	41	Europe
USA	Washington	350	441	Amérique

Suppression des doubles

SELECT **DISTINCT** continent
FROM Pays

Donne le résultat :

Continent
Europe
Amérique

ORDER BY

Tri des tuples du résultat

pays

Nom	Capitale	Population	Surface	Continent
Irlande	Dublin	5	70	Europe
Autriche	Vienne	10	83	Europe
UK	Londres	50	244	Europe
Suisse	Berne	7	41	Europe
USA	Washington	350	441	Amérique

```
SELECT continent, nom, population
FROM Pays
WHERE surface > 60
ORDER BY continent, nom ASC
```

Continent	Nom	Population
Amérique	USA	350
Europe	Autriche	10
Europe	Irlande	5
Europe	Suisse	7
Europe	UK	50

2 possibilités : ASC / DESC

GROUP BY

Partition de l'ensemble des tuples en groupes homogènes

pays

Nom	Capitale	Population	Surface	Continent
Irlande	Dublin	5	70	Europe
Autriche	Vienne	10	83	Europe
UK	Londres	50	244	Europe
Suisse	Berne	7	41	Europe
USA	Washington	350	441	Amérique

```
SELECT continent, MIN(population), MAX(population),AVG(population),
    SUM(surface), COUNT(*)
FROM Pays GROUP BY continent ;
```

Continent	MIN(population)	MAX(population)	AVG(population)	SUM(surface)	COUNT(*)
Europe	5	50	18	438	4
Amérique	350	350	350	441	1

HAVING

Conditions sur les fonctions d'agrégation

Il n'est pas possible d'utiliser la clause WHERE pour faire des conditions sur une fonction d'agrégation. Donc, si l'on veut afficher les pays dont on possède plus de 3 individus, la requête suivante ne fonctionnera pas.

```
SELECT continent, COUNT(*)
FROM Pays
WHERE COUNT(*) > 3
GROUP BY continent ;
```

Il faut utiliser HAVING qui se place juste après le GROUP BY

```
SELECT continent, COUNT(*)
FROM Pays
```

Renommage des attributs : AS

```
SELECT MIN(population) AS min_pop,
MAX(population) AS max_pop,
AVG(population) AS avg_pop,
SUM(surface) AS sum_surface,
COUNT(*) AS count
FROM Pays
WHERE continent = 'Europe' ;
```

min_pop	max_pop	avg_pop	sum_surface	count
5	50	18	438	4