

# SQL et bases de données relationnelles

# Sommaire

1. Les bases de données et SQL
2. Types de données MySQL
3. Langage de définition de données (DDL)
4. Langage de manipulation des données (DML)
5. Mécanisme de transaction (TCL)
6. Gestion des droits et des utilisateurs (DCL)

# 1. Les bases de données et SQL

# Qu'est-ce qu'un SGBD ?

- Acronyme : **S**ystème de **g**estion de **b**ase de **d**onnées
- Né des travaux d'IBM dans les années 1960 / 1970
- 1<sup>er</sup> SGBD commercialisé : Oracle Version 2
- Acteurs majeurs : Oracle , Microsoft (SQL Server), IBM (DB2)

# Type de SGBD

- SGBD relationnels (SGBDR) :
  - mise en œuvres des théories de Codd
- SGBD orientés objet (SGBDO) :
  - utilise la conception et programmation objet (*ex : O2*)
- SGBD relationnels objets (SGBDRO) :
  - (*ex : Oracle, PostgreSQL*)
- NOSQL (Not Only SQL) :
  - utilisés pour les gros volumes de données peu ou mal structurées.  
(*ex : MongoDB, Firebase*)

# Les principaux objets d'un SGBD

## Objets

Base de données

Schémas

Tables

Indexes

Vues

Synonymes

Séquences

Procédures

Fonctions

Déclencheurs

usagers

# Qu'est-ce-que le SQL ?

- Acronyme : **S**tructured **Q**uery **L**anguage
- Langage né dans les années 1980 créé par IBM
- Basé sur l'algèbre relationnel et la modélisation de données
- Le langage est standardisé en 1986 (ANSI)

# Principales normes SQL

Année	Appellations	Apports majeurs
1986	SQL86,87	Première publication par l'ANSI, ratification par l'ISO en 1987
1999	SQL:1999, SQL3	Expression régulières, récursivité, déclencheurs, langage procédurale, type abstraits et aspect objet
2003	SQL:2003	Prise en compte de XML, fonction de fenêtrage, séquences, colonnes d'identité
2008	SQL:2008	Correction de certains défauts et de petits manques de versions antérieures (fonctions, types, curseur, etc)



# Structure du langage SQL

- Le **DDL** (Data Definition Language) :
  - Ce sont les ordres SQL permettant de créer (*CREATE*), modifier (*ALTER*) ou supprimer (*DROP*) les objets de la base
- Le **DML** (Data Manipulation language) :
  - Ce sont les ordres SQL permettant d'ajouter (*INSERT*), de modifier (*UPDATE*), de supprimer (*DELETE*) ou d'extraire des données (*SELECT*)
- Le **DCL** (Data Control Language) :
  - Ce sont les ordres permettant de définir les privilèges afférents aux utilisateurs (*GRANT*, *REVOKE*)
- Le **TCL** (Transaction Control Language) :
  - Ce sont les ordres qui permettent de gérer les transactions englobant des ordres des trois premières subdivisions

# MySQL

Avantages de MySQL :

- SGBD Open Source
- Maintenu par Oracle
- Populaire (LAMP)
- Cross-platform



## 2. Types de données MySQL

# Types de chaînes de caractères

CHAR( <i>n</i> )	Une chaîne de caractère de taille fixe de longueur <i>n</i> dont les caractères sont codés sur 1 octet
VARCHAR( <i>n</i> )	Une chaîne de longueur variable. Le paramètre <i>n</i> spécifie la longueur maximale de la colonne comprise entre 0 et 65535 caractères
BINARY( <i>n</i> )	Équivalent à CHAR(), mais stocke des chaînes d'octets binaires. Le paramètre <i>n</i> spécifie la longueur de la colonne en octets. La valeur par défaut est 1
VARBINARY( <i>n</i> )	Équivalent à VARCHAR(), mais stocke des chaînes d'octets binaires. Le paramètre <i>n</i> spécifie la longueur maximale de la colonne en octets
TEXT( <i>n</i> )	Contient une chaîne de caractères d'une longueur maximale de 65 535 octets
...	



[Liens vers la documentation](#)

# Types numériques

BIT( $n$ )	Un type de valeur binaire. Le paramètre $n$ peut contenir une valeur de 1 à 64.
TINYINT( $n$ )	Un très petit nombre entier. Le paramètre $n$ spécifie la largeur maximale de l'affichage (qui est de 255).
BOOL	Le zéro est considéré comme faux, les valeurs non nulles sont considérées comme vraies.
INT( $n$ )	Un nombre entier moyen. Le paramètre $n$ spécifie la largeur maximale de l'affichage (qui est de 255).
DECIMAL( $n, d$ )	Un nombre exact à virgule fixe. Le nombre total de chiffres est spécifié dans $n$ . Le nombre de chiffres après le point décimal est spécifié dans le paramètre $d$ .
...	



[Liens vers la documentation](#)

# Types temporels

DATE	Une date au format : AAAA-MM-JJ
DATETIME	Une combinaison de date et d'heure. Format : AAAA-MM-JJ hh:mm:ss
TIMESTAMP	Les valeurs TIMESTAMP sont stockées sous forme de nombre de secondes depuis l'époque Unix ('1970-01-01 00:00:00' UTC). Format : AAAA-MM-JJ hh:mm:ss
TIME	Une heure au format : hh:mm:ss
YEAR	Une année à 4 chiffres



[Liens vers la documentation](#)

# 3. Langage de définition de données

# Instructions du DDL

On distingue 4 types de commandes SQL de définition de données :

- *CREATE* : création d'une structure de données
- *ALTER* : modification d'une structure de données
- *DROP* : suppression d'une structure de données
- *RENAME* : renommage d'une structure de données

Ces commandes peuvent porter sur les structures de données de type suivantes :

- *TABLE* : table
- *INDEX* : indice
- *VIEW* : table virtuelle
- *SEQUENCE* : suite de nombres
- *SYNONYM* : synonyme
- *USER* : utilisateur



# Convention de nommage des objets

- Ne pas dépasser 128 caractères
- Commencer par une lettre
- Être composé de lettres, de chiffres et du caractère « \_ »
- Ne pas être un mot réservé du SQL (sauf entre guillemets)
- Ne pas utiliser d'accents
- Utiliser des lettres en minuscules
- Ex : *client, bdd\_ecole, date\_inscription*

# Ordres de gestion de base de données

- Utilisation de l'ordre *CREATE DATABASE* pour créer une base de données
- *ALTER* pour modifier une base de données
- *DROP* pour supprimer une base de données
- *USE* pour utiliser une base de données

```
CREATE DATABASE my_example;  
DROP DATABASE my_example;  
ALTER DATABASE example READ ONLY = 1;  
USE my_example;
```

# Ordres de gestion des objets TABLE

- Les objets tables se gèrent principalement par les ordres suivants :



```
CREATE TABLE ...  
DROP TABLE ...  
ALTER TABLE ...  
RENAME TABLE ...
```

# L'instruction CREATE TABLE

```
CREATE TABLE nom_de_la_table  
(  
    colonne1 type_donnees, -- prenom VARCHAR(50)  
    colonne2 type_donnees, -- nom VARCHAR(50)  
    colonne3 type_donnees, -- date_naissance DATE NOT NULL  
    colonne4 type_donnees -- pays VARCHAR(255)  
)
```

# Contraintes d'une TABLE

- (NOT) NULL : valeur nulle (non) autorisée
- DEFAULT : indique une valeur par défaut
- UNIQUE : indique que la colonne de la table est unique
- CHECK : ajoute une vérification lors de l'enregistrement
- PRIMARY KEY : indique que la colonne est la clé primaire
- FOREIGN KEY : clé étrangère d'une autre table
- INDEX : permet l'optimisation lors de requête

# La clé primaire (Primary Key)

- Une table contient généralement une colonne ou une combinaison de colonnes dont les valeurs identifient de façon unique chaque ligne dans la table.
- Cette colonne (ou ces colonnes), appelée clé primaire (PK, Primary Key), assure l'intégrité de l'entité de la table.
- Les contraintes de clé primaire garantissent des données uniques, c'est pourquoi elles sont souvent définies pour une colonne d'identité.

# La clé étrangère (Foreign Key)

- On appelle « clé étrangère » une colonne ou une combinaison de colonnes utilisée pour établir et conserver une liaison entre les données de deux tables pour contrôler les données qui peuvent être stockées dans la table de clés étrangères.
- Dans une référence de clé étrangère, la création d'une liaison entre deux tables s'effectue lors du référencement de la ou des colonnes contenant les valeurs de clé primaire d'une table dans la ou les colonnes de l'autre table. Cette colonne devient alors une clé étrangère dans la seconde table.

# Notation des contraintes

On peut ajouter les contraintes de deux façons :

1. A la fin d'une déclaration de colonne : écriture plus compacte
2. A la fin de la déclaration de toutes les colonnes : on peut nommer une contrainte et y faire référence plus tard

On privilégie généralement la 2<sup>ème</sup> approche

```
CREATE TABLE employe (  
    employe_id INT NOT NULL,  
    prenom VARCHAR(100),  
    nom VARCHAR(50),  
    departement_id INT,  
    CONSTRAINT pk_employe_employe_id PRIMARY KEY(employe_id),  
    CONSTRAINT fk_departement_employe_departement_id FOREIGN KEY(depart  
ement_id) REFERENCES departement(departement_id)  
);
```



# Modifier une table - ALTER

ALTER TABLE nom\_table

| ADD COLUMN nom\_col type\_col [contrainte\_col] [, ...]

| ADD CONSTRAINT [nom\_contrainte] {PRIMARY KEY | UNIQUE} paramètres [, ...] |

CHANGE [COLUMN] nom\_col nouveau\_nom\_col type\_col [contrainte\_col] [, ...] | MODIFY  
[COLUMN] nom\_col type\_col [contrainte\_col] [, ...]

| DROP [COLUMN] nom\_col [, ...]

| DROP PRIMARY KEY

| DROP FOREIGN KEY nom\_contrainte [, ...]

| RENAME [TO | AS] nouv\_nom\_table;

# Exemple de modifications

```
ALTER TABLE vehicule  
ADD couleur VARCHAR(50),  
ADD commentaire VARCHAR(255);
```

```
ALTER TABLE vehicule  
MODIFY commentaire VARCHAR(100) NOT NULL;
```



## Exercice 1 (1/5) :

1. Créer une table personne avec les caractéristiques suivantes :

- personne\_id - clé primaire
- titre - M. ou Mme
- prenom
- nom
- telephone
- email

Chaque personne peut avoir une ou plusieurs adresses.

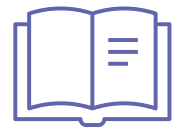
2. Créer la table adresse avec une clé étrangère et chaque adresse possède une rue, une ville et un code postal



## Exercice 2 (1/2) :

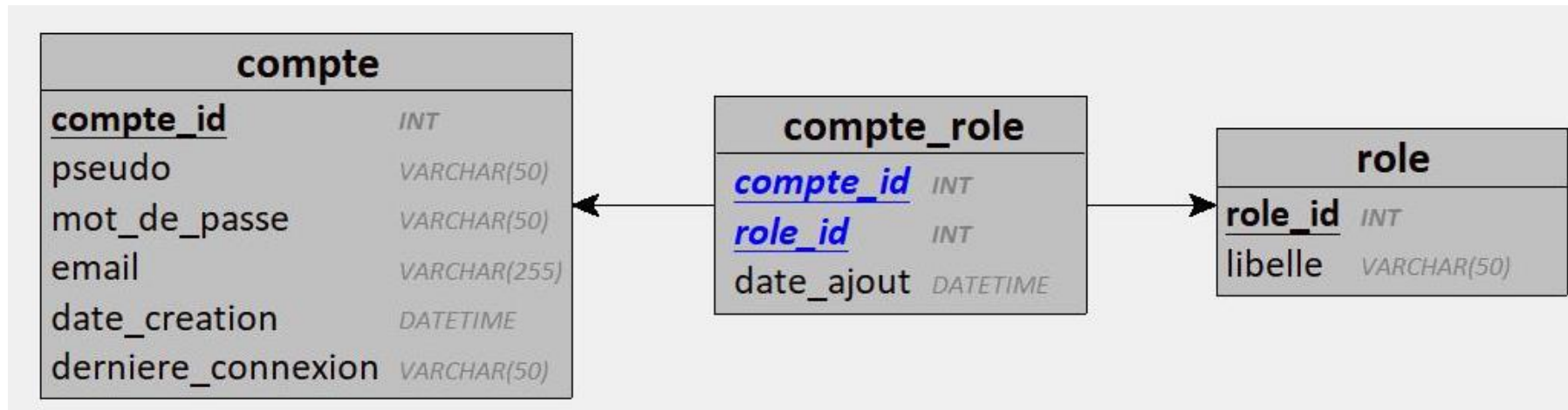
1. Créer une table qui correspond au schéma suivant :

compte	
<b><u>compte_id</u></b>	<i>INT</i>
pseudo	<i>VARCHAR(50)</i>
mot_de_passe	<i>VARCHAR(50)</i>
email	<i>VARCHAR(255)</i>
date_creation	<i>DATETIME</i>
derniere_connexion	<i>VARCHAR(50)</i>



## Exercice 2 (2/2) :

2. Créer les tables qui correspondent au schéma suivant :





## Exercice 3 :

1. Créer une table professeur qui possède les colonnes suivantes :
  - professeur(id, prénom, nom, n° classe, n° de département, email, téléphone)
2. Créer une table étudiant avec les attributs suivants :
  - etudiant(id, prenom, nom, téléphone, n° classe, date d'obtention du diplôme)
3. Créer une table qui permet d'associer les étudiants à des professeurs composée d'identifiant d'élèves et d'identifiants de professeurs
4. Ajoutez les contraintes suivantes :
  - Avoir au minimum 1 numéro de téléphone pour chaque étudiant
  - Avoir des clés primaires pour chaque table
  - Le numéro de téléphone et l'email doivent être unique
5. Tester le fonctionnement des contraintes en insérant des données

# 4. Langage de manipulation des données (DML)

# Définition du CRUD

L'acronyme informatique anglais CRUD désigne les quatre opérations de base pour la persistance des données, en particulier le stockage d'informations en base de données.

Opération	SQL	Action
CREATE	INSERT	Ajouter un/des enregistrement(s)
READ	SELECT	Rechercher
UPDATE	UPDATE	Mettre à jour
DELETE	DELETE	Supprimer



# L'instruction INSERT

- L'insertion de données dans une table s'effectue à l'aide de la commande INSERT INTO
- Cette commande permet au choix d'inclure une seule ligne à la base existante ou plusieurs lignes d'un coup

```
INSERT INTO
```

```
    nom_table [nom_colonne1 [, nom_colonne2  
    ...]]
```

```
VALUES ( {valeur_colonne1 | DEFAULT} [, v  
valeur_colonne2 | DEFAULT ...]  
);
```

# Insérer une ligne en spécifiant toutes les colonnes

- Contraintes :
  - Il faut remplir toutes les données en respectant l'ordre des colonnes
  - L'ordre des colonnes doit rester identique sinon certaines valeurs prennent le risque d'être complétée dans la mauvaise colonne

```
INSERT INTO  
    nom_table  
VALUES  
    ('valeur 1', 'valeur 2', ...)
```

## Insérer une ligne en spécifiant uniquement les colonnes souhaitées

- Syntaxe similaire à la précédente
- Il faut indiquer le nom des colonnes à renseigner après le nom de la table
- Il est possible de ne pas renseigner le nom de toutes les colonnes
- On utilisera généralement cette approche

```
INSERT INTO
    nom_table
    (nom_colonne_1, nom_colonne_2, ..)
VALUES
    ('valeur 1', 'valeur 2', ...),
    ('valeur 1', 'valeur 2', ...)
```

# Insérer des données via requête imbriquée

- On peut insérer plusieurs lignes grâce à une requête imbriquée. Il faut cependant que les schémas soient les mêmes

```
INSERT INTO
    table2 (colonne1, colonne2, colonne3,...)
SELECT
    colonne1,
    colonne2,
    colonne3,
    ...
```



## Exercice 1 (2/5) :

3. Créer un fichier de requête SQL permettant l'insertion d'un jeu de données dans les tables créées dans l'exercice précédent

# L'instruction DELETE

- L'instruction DELETE est utilisée pour supprimer les enregistrements existants dans une table.
- Cette instruction ne permet pas de supprimer une table mais son contenu. Même si le contenu de la table est vidé, elle existe toujours avec le même schéma.
- ⚠ Si la clause WHERE n'est pas spécifiée, la table est entièrement vidée de son contenu.

```
DELETE FROM  
    nom_table  
WHERE  
    CONDITION;
```



## Exercice 1 (3/5) :

4. Supprimer l'adresse de la personne avec l'identifiant n° 3
5. Essayer de supprimer la personne dont le prénom est égal à 'toto'

# L'instruction UPDATE

- L'instruction UPDATE permet de modifier les valeurs d'attributs d'une ou plusieurs lignes.
- ⚠ Si la clause WHERE n'est pas spécifiée, toutes les lignes de la table sont modifiées.

```
UPDATE
```

```
    nom_table
```

```
SET
```

```
    colonne1 = value1,
```

```
    colonne2 = value2,
```

```
...
```

```
WHERE CONDITION;
```





## Exercice 1 (4/5) :

6. Changer le numéro de téléphone des personnes portant le nom 'dupont'

# L'instruction SELECT

- L'instruction SELECT est l'instruction la plus complexe du langage SQL
- Néanmoins, elle offre une grande flexibilité pour la recherche d'informations dans les tables
- Elle est constituée de 2 clauses obligatoires et de 4 clauses optionnelles

```
SELECT
    colonne [, ...]
FROM
    TABLE [, ...]
[WHERE expression]
[GROUP BY colonne [, ...]]
[HAVING expression]
[ORDER BY colonne [{ASC | DESC}] [, ...]];
```

# Précisions sur l'instruction SELECT

- cette clause permet de sélectionner les colonnes à retourner
- on sépare les colonnes à retourner par des virgules « , »
- il est possible de préfixer une colonne par : « **nom\_table.** »
- l'usage de \* indique que toutes les colonnes sont sélectionnées

```
-- sélection de toutes les colonnes de la
table employe
SELECT *
FROM employe

-- sélection de tous les prénoms et noms de
la table employe

SELECT employe.nom, prenom
FROM employe
```

# La clause FROM

- cette clause permet de choisir quelle(s) table(s) est utilisée(s) pour la requête
- on sépare les tables à utiliser dans la requête par des virgules « , »

```
SELECT
    employe.employe_id, departement.departement_id
FROM
    employe, departement

SELECT
    *
FROM
    employe, departement
```

# Concept des alias

- Il est possible de créer un alias pour une colonne et/ou une table
- Cela permet d'alléger la syntaxe des requêtes (l'usage du mot clé AS est optionnel mais recommandé)
- Les alias de colonnes ont l'avantage de renommer la colonne pour la sortie, souvent pertinent pour apporter des précisions (l'usage d'apostrophes permet des mots avec espaces)
- les alias de table sont parfois essentielle lors de requêtes récursives afin de lever l'ambiguïté

```
SELECT
    nom_colonne [[AS] alias_colonne ] [, ...]
FROM
    nom_table [[AS] alias_table ] [, ...];

SELECT
    emp.nom AS Employé,
    sup.nom AS Superviseur
FROM
    employe AS emp,
    employe AS sup
WHERE
    emp.Superviseur = sup.nas;
```

# La clause WHERE

- la clause WHERE permet d'ajouter une restriction sur les lignes retournées
- elle prend la forme d'une expression conditionnelle
- lorsque la clause WHERE est omise, toutes les lignes sont affichés (équivalent à WHERE TRUE).

```
SELECT
    nom
FROM
    employe
WHERE
    salaire > 50000

SELECT
    *
FROM
    employe,
    departement
WHERE
    employe.departement_id = departement.departement_id
```

# Opérateurs de la clause WHERE

Operateur	Description	Exemple
=	Egalité	WHERE nom = prenom
>	Supérieur à	WHERE age > 18
<	Inférieur à	WHERE date_anniversaire < NOW()
>=	Supérieur ou égal	WHERE age >= 21
<=	Inférieur ou égal	WHERE age <= 65
<>	Différent de	WHERE nom <> 'Bernard'
BETWEEN	Entre	WHERE age BETWEEN 13 AND 17
LIKE	Cherche un pattern particulier	WHERE prenom LIKE 'An%'
IN	Spécifier une liste de choix	WHERE departement IN ('Nord', 'Bretagne')

# La clause DISTINCT

- SQL n'élimine pas les doublons des lignes retournées
- la clause DISTINCT permet de retirer les doublons (un doublon est une ligne entièrement identique)
- on verra plus loin que cette clause peut être utilisé avec les fonctions d'agrégation

```
SELECT
    DISTINCT ville
FROM
    employe
```



# La clause LIMIT (ou TOP)

- La clause LIMIT est utilisée pour spécifier le nombre d'enregistrements à retourner
- Elle est utile pour les grandes tables contenant des milliers d'enregistrements
- Cette clause est différentes selon le SGBD (TOP, ROWNUM)

```
SELECT  
    nom  
FROM  
    employe  
LIMIT  
    10
```

# ORDER BY

- La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL.
- Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.
- ASC : croissant par défaut si non précisé
- DESC : décroissant

```
SELECT
    nom,
    prenom,
    age
FROM
    personne
ORDER BY
    nom DESC,
    age ASC
```



## Exercice 1 (1/5) :

A l'aide de l'exercice n°1 :

7. Créez une requête SQL permettant d'afficher l'intégralité des personnes présentes dans la BDD ordonnées par nom de famille de manière décroissante
8. Créer une requête SQL permettant d'afficher l'intégralité des personnes triées par le titre « Mlle » puis « Mme », puis « Mr »
9. Créer une requête SQL permettant de rechercher une personne par son adresse email

# Requêtes imbriquées

- il est possible d'utiliser le résultat d'une requête comme entrée dans une autre
- ainsi, on peut imbriquer plusieurs requêtes les unes dans les autres
- ce mécanisme offre des solutions élégantes et puissantes à plusieurs situations
- comme nous le verrons plus loin, la notion d'imbrication fait partie du langage SQL et s'applique sur tous les éléments du langage (n'est pas réservée à la clause SELECT)

```
SELECT
nom,
prenom
FROM
    (SELECT * FROM employe WHERE sexe = 'f')
    AS femme_employe
WHERE nom LIKE 'Mont%'
```

# Utilisation de requêtes imbriquées

- puisque les résultats sont toujours des tables, on peut utiliser ces tables comme données d'entrée
- par contre, si la requête utilisée retourne une table n'ayant qu'une seule colonne et une seule valeur, il est alors possible d'utiliser ce résultat comme un scalaire
- si la requête utilisée retourne une table ayant une colonne de plusieurs valeurs, il est possible d'utiliser ce résultat comme une liste

```
SELECT nom, prenom
FROM employe
WHERE
departement_id =
(SELECT departement_id
FROM departement
WHERE nom = 'Ventes')
AND
ville IN (SELECT nom_ville
FROM geographie
WHERE province = 'Haut de frances')
```

# GROUP BY

- La commande GROUP BY est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction de totaux sur un groupe de résultat
- Sur une table qui contient toutes les ventes d'un magasin, il est par exemple possible de liste regrouper les ventes par clients identiques et d'obtenir le coût total des achats pour chaque client.

```
SELECT
    COUNT(employe_id),
    pays
FROM
    employe
GROUP BY
    pays;
```

# Les fonctions d'agrégations

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

Utilisation : **SELECT** fonction(colonne) **FROM** nom\_table

- Selon les SGBD, plusieurs autres fonctions d'agrégation sont disponible :
  - valeur médiane, écart type, variance et plusieurs autres
- Les fonctions d'agrégations ne sont possibles que dans la clause SELECT et HAVING
- Sans la clause GROUP BY, l'usage des fonctions d'agrégations se fait en considérant toutes les lignes de la table.

# HAVING

- La condition HAVING en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que SUM(), COUNT(), AVG(), MIN() ou MAX()

```
SELECT
    service_id,
    AVG(salaire)
FROM
    employe
GROUP BY
    service_id
HAVING
    AVG(salaire) > 3000
```





## Exercice 4 (1/2) :

1. Créer une table nommée « livre » détenant les champs suivant :
  - Id (INT)
  - titre (VARCHAR(150))
  - auteur (VARCHAR(50))
  - editeur (VARCHAR(50))
  - date\_publication (DATE)
  - isbn\_10 (VARCHAR(10))
  - isbn\_13 (VARCHAR(15))
2. Exécuter le script d'injection « AjoutLivres.sql » dans la BDD



## Exercice 4 (2/2) :

3. Créer une requête permettant d'afficher les 10 livres les plus anciens (avec toutes les colonnes) classés par ordre croissant
4. Créer une requête permettant d'afficher les 10 livres les plus anciens (seulement l'affichage des colonnes : date\_publication, auteur, titre) classés par date de publication croissante
5. Créer une requête permettant d'afficher tous les livres de « Agatha Christie » présent dans la base (à ce stade 3 livres).
6. On nous informe qu'une erreur s'est glissée sur un livre de « Agatha Christie » présent dans la base. En effet, une entrée de la BDD aurait comme auteur « Agatha Christies ». Faites une requête permettant de modifier cette erreur puis exécutez de nouveau la requête de la question 5 afin d'afficher de nouveau le nombre de livre de « Agatha christie » présent dans la BDD (4 à ce stade).
7. Insérer le livre de votre choix dans la BDD en respectant toutes les colonnes.
8. Supprimer le livre de votre choix par les critères d'auteur et titre.

# Requêtes corrélées

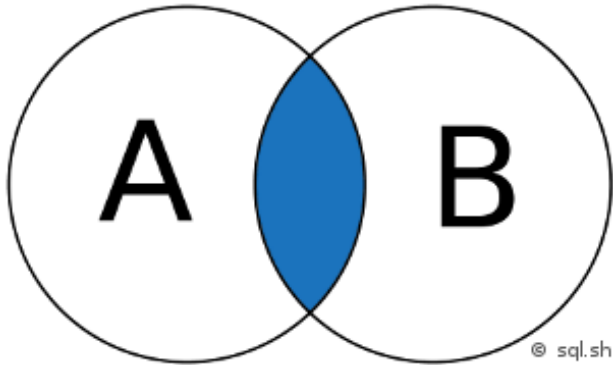
- Le SQL permet des requêtes plus complexes qu'on appelle des requêtes imbriquées corrélées (ou sous-requêtes corrélées ou requêtes synchronisées)
- En contre partie, les requêtes imbriquées que nous avons vues sont nommées requêtes imbriquées non corrélées.
- Ces requêtes ont la forme suivante :
  - une requête est imbriquée à l'intérieure d'une autre
  - contrairement aux requêtes imbriquées non corrélées, la requête interne utilise des valeurs de la requête externe
  - la résolution de telles requêtes implique que la requête interne est évalué pour chaque ligne de la requête externe.

```
SELECT
    nom
FROM
    employe AS chaqueEmploye
WHERE
    salaire > (
        SELECT
            AVG(salaire)
        FROM
            employe AS sousGroupeEmploye
        WHERE
            sousGroupeEmploye.departement_id = chaqueEmploye.departement_id
    );
```

# Les jointures

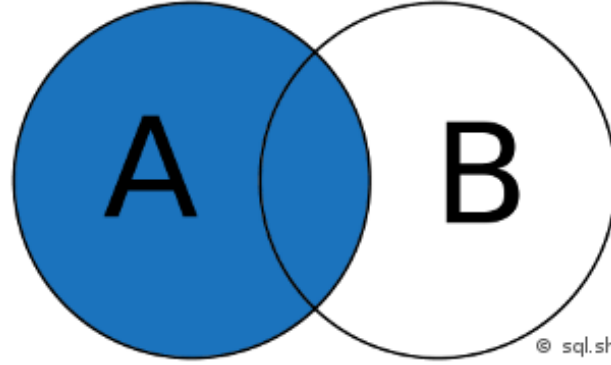
Jointure	Description
INNER JOIN	jointure interne pour retourner les enregistrements quand la condition est vrai dans les 2 tables. C'est l'une des jointures les plus communes.
LEFT JOIN	jointure externe pour retourner tous les enregistrements de la table de gauche (LEFT = gauche) même si la condition n'est pas vérifié dans l'autre table.
RIGHT JOIN	Similaire à LEFT JOIN mais pour la table de droite
FULL JOIN	jointure externe pour retourner les résultats quand la condition est vrai dans au moins une des 2 tables.
SELF JOIN	permet d'effectuer une jointure d'une table avec elle-même comme si c'était une autre table.
...	

# Exemples de jointures



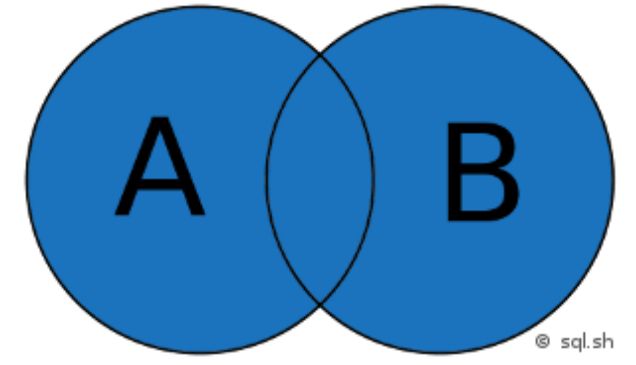
INNER JOIN :

```
SELECT
*
FROM
A
INNER JOIN B ON A.key = B.key
```



LEFT :

```
SELECT
*
FROM
A
LEFT JOIN B ON A.key = B.key
```



FULL JOIN :

```
SELECT
*
FROM
A
FULL JOIN B ON A.key = B.key
```



## Exercice 5 :

En utilisant les fichiers villes\_frances.sql et departement.sql :

1. Obtenir la liste des 10 villes les plus peuplées en 2012
2. Obtenir la liste des 50 villes ayant la plus faible superficie
3. Obtenir la liste des départements d'outre-mer, c'est-à-dire ceux dont le numéro de département commençant par "97"
4. Obtenir le nom des 10 villes les plus peuplées en 2012, ainsi que le nom du département associé
5. Obtenir la liste du nom de chaque département, associé à son code et du nombre de commune au sein de ces départements, en triant afin d'obtenir en priorité les départements qui possèdent le plus de communes
6. Obtenir la liste des 10 plus grands départements, en termes de superficie
7. Compter le nombre de villes dont le nom commence par "Saint"
8. Obtenir la liste des villes qui ont un nom existants plusieurs fois, et trier afin d'obtenir en premier celles dont le nom est le plus souvent utilisé par plusieurs communes
9. Obtenir en une seule requête SQL la liste des villes dont la superficie est supérieure à la superficie moyenne
10. Obtenir la liste des départements qui possèdent plus de 2 millions d'habitants
11. Remplacez les tirets par un espace vide, pour toutes les villes commençant par "SAINT-" (dans la colonne qui contient les noms en majuscule)

## 5. Mécanisme de transaction

# Définition d'une Transaction SQL

*« Une transaction SQL est une séquence d'exécutions de déclarations SQL qui est atomique en ce qui concerne la récupération. C'est-à-dire que soit le résultat de l'exécution est complètement réussi, soit il n'a aucun effet sur les schémas SQL ou les données SQL. »* - Le standard SQL

- Le moteur de stockage de MySQL prend en charge les transactions conformes à la norme ACID.



# ACID

- En informatique, les propriétés ACID sont un ensemble de propriétés qui garantissent qu'une transaction informatique est exécutée de façon fiable
- **A**tomacité : s'assure qu'une transaction se fait au complet ou pas du tout
- **C**ohérence : chaque transaction amènera le système d'un état valide à un autre état valide
- **I**solation : Toute transaction doit s'exécuter comme si elle était la seule sur le système
- **D**urabilité : s'assure que lorsqu'une transaction a été confirmée, elle demeure enregistrée même à la suite d'une panne d'électricité

# Création d'une transaction

- START TRANSACTION ou BEGIN démarre une nouvelle transaction
- COMMIT engage la transaction en cours, rendant ses changements permanents
- ROLLBACK annule la transaction en cours, en annulant ses changements
- SET autocommit désactive ou active le mode autocommit par défaut pour la session en cours

```
START TRANSACTION
    [transaction_characteristic [, transaction_characteris
tic] ...]

transaction_characteristic: {
    WITH CONSISTENT SNAPSHOT
    | READ WRITE
    | READ ONLY
}

BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}
```

# Exemple de transaction

```
START TRANSACTION;
```

```
DELETE FROM
```

```
    employe
```

```
WHERE
```

```
    employe_id = 5;
```

```
COMMIT;
```

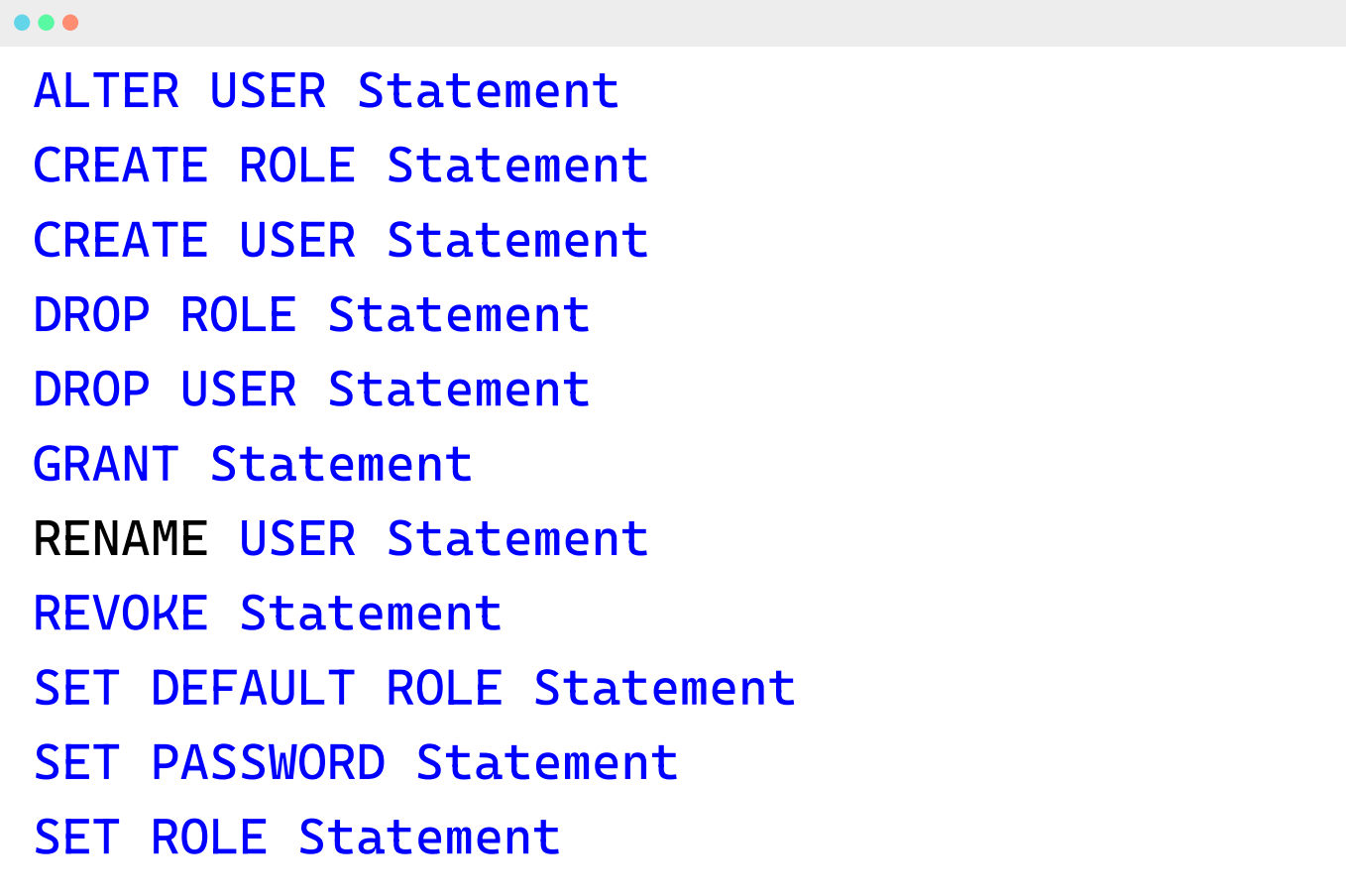
## 6. Gestion des droits et des utilisateurs

# Gestion des utilisateurs

- MySQL permet la création de comptes qui autorisent les utilisateurs clients à se connecter au serveur et à accéder aux données gérées par le serveur.
- La fonction principale du système de privilèges MySQL est d'authentifier un utilisateur qui se connecte à partir d'un hôte donné et d'associer à cet utilisateur des privilèges sur une base de données tels que SELECT, INSERT, UPDATE et DELETE.
- La gestion des utilisateurs consiste en des instructions SQL telles que CREATE USER, GRANT et REVOKE.

# Principaux ordre de gestions des utilisateurs

- Les informations ded utilisateurs MySQL sont stockées dans les tables du schéma système mysql



```
ALTER USER Statement  
CREATE ROLE Statement  
CREATE USER Statement  
DROP ROLE Statement  
DROP USER Statement  
GRANT Statement  
RENAME USER Statement  
REVOKE Statement  
SET DEFAULT ROLE Statement  
SET PASSWORD Statement  
SET ROLE Statement
```

# Créer un utilisateur

- Pour chaque compte, CREATE USER crée une nouvelle ligne dans la table système mysql.user
- Lorsqu'il est créé pour la première fois, un compte n'a aucun privilège et son rôle par défaut est NONE
- [Documentation](#)

```
CREATE USER '<nom>' @ 'localhost'  
IDENTIFIED BY '<mot de passe>';
```

# Accorder des droits aux utilisateurs

- L'instruction GRANT permet d'accorder des privilèges ou des rôles à des comptes
- L'instruction REVOKE permet de révoquer les privilèges accordés
- SHOW GRANTS permet de déterminer les privilèges dont dispose un compte
- [Documentation](#)

## GRANT

```
priv_type [(column_list)]  
    [, priv_type [(column_list)]] ...  
ON [object_type] priv_level  
TO user_specification [ user_options  
...]
```



# Exemple d'utilisation de GRANT

```
-- Donne accès en lecture à la colonne nom et prénom de la table  
employe
```

```
GRANT SELECT (nom, prenom) on employe to 'jean'@'localhost';
```

```
-- Donner tous les droits sur la base de données (root)
```

```
GRANT ALL PRIVILEGES ON * . * TO 'toto'@'localhost' IDENTIFIED BY  
'password';
```

```
SHOW GRANTS FOR 'toto'@'localhost';
```

# Créer des rôles

- L'instruction CREATE ROLE crée un ou plusieurs rôles MariaDB
- La longueur maximale d'un rôle est de 128 caractères
- [Documentation](#)

```
CREATE [OR REPLACE] ROLE [IF NOT EXISTS]
role
    [WITH ADMIN
        {CURRENT_USER | CURRENT_ROLE | user |
role}]
```

# Ressources

- [Cours et Tutoriels sur le Langage SQL](#)
- [MariaDB Server Documentation - MariaDB Knowledge Base](#)
- [MySQL :: MySQL 8.0 Reference Manual](#)
- [MySQL SQL \(w3schools.com\)](#)

Merci pour votre attention  
Des questions ?