# SQL: Structured Query Language

@MOUSSA CAMARA

## SQL: c'est quoi déjà?

► Le SQL (Structured Query Language) en français Langage de requête structurée est un langage permettant de communiquer avec une base de données.



Mais c'est quoi une base de données ?

## Réponse ...

- ► Le concept **Base de Données** (BDD) apparaît vers 1960 suite au nombre croissant d'informations que les entreprises doivent gérer et partager (évolution dans les systèmes informatiques).
- Historiquement, chaque application engendrait ses propres fichiers de données exploités par ses propres programmes. La création d'une base de données va à l'encontre de cette façon de procéder; elle rend possible la centralisation, la coordination, l'intégration et la diffusion de l'information archivée.
- une <u>base de données</u> est un ensemble structuré de données enregistré sur des supports accessibles par l'ordinateur, pour satisfaire simultanément plusieurs utilisateurs de façon sélective et en un temps opportun.

### SYSTEME DE GESTION DE BASE DE DONNEES (SGBD)

- C'est l'ensemble des programmes et des langages de commande (livrés par le constructeur ou la société qui commercialise le produit) qui permettent :
  - la définition des différents "ensembles de données" de la base, et les relations existant entre eux
  - ▶ le traitement, l'exploitation de ces données (interrogations, mises à jour, calculs, extractions...)
- ▶ NB: Le SGBD reçoit des commandes aussi bien des programmes d'application que des utilisateurs; il commande les manipulations de données, généralement par l'intermédiaire du SGF de base de l'ordinateur.

## OBJECTIFS DE L'APPROCHE BDD

- Pour pallier les inconvénients des méthodes classiques (à base de fichiers), les bases de données tentent d'atteindre 4 objectifs principaux :
  - ▶ intégration et corrélation
  - flexibilité (indépendance)
  - Disponibilité
  - sécurité



## A quoi ça sert le SQL?

Comme dans la vie, pour que des personnes puissent se comprendre, elles doivent parler le même langage et bien en informatique, c'est pareil.

Pour que les différents logiciels et le moteur de base de données puissent se comprendre, ils utilisent un langage appelé SQL.

Pour que les différents logiciels et le moteur de base de données puissent se comprendre, ils utilisent un langage appelé SQL.

- Ce langage est complet. Il va être utilisé pour :
  - Lire les données,
  - Ecrire les données,
  - Modifier les données,
  - Supprimer les données
- Il permettra aussi de modifier la structure de la base de données :
  - Ajouter des tables,
  - Modifier les tables,
  - les supprimer
  - Ajouter, ou supprimer des utilisateurs,
  - Gérer les droits des utilisateurs,
  - ▶ Gérer les bases de données : en créer de nouvelles, les modifier

## A quoi ça sert le SQL?

- ► En résumé:
  - Stocker les données
  - Manipuler les données
  - Integrer les données dans un programme informatique

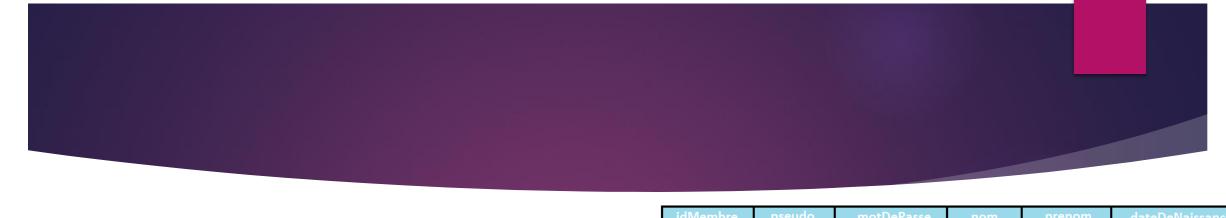
- 1- Enregistrer une commande (table: commande)
- 2- Enregistrer un compte d'utilisateur (table : membre)
- 3- Enregistrer un message (table : commentaire)



Dans le contexte d'un site nous avons régulièrement besoin de garder des informations en mémoire.

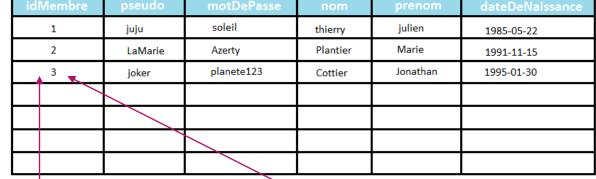
#### Acronyme

- ► MCD = Modèle Conceptuel de Données (Plan de construction)
- ► SGBD = Système de Gestion de Base de Données (Permet d'exploiter les données)
- ▶ BDD = Base De Données (Emplacement des données sauvegardées)
- ► SQL = Structured Query Langage (Langages de requete : Structured Query Langage)

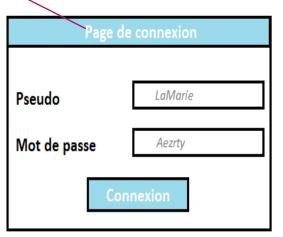


Lors de l'inscription, les comptes des membres sont enregistrés (INSERT) dans une table d'une base de données.

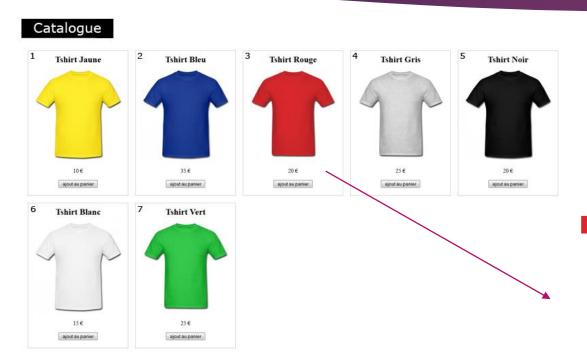
Lors de la connexion, nous sélectionnons (SELECT) les informations en BDD pour vérifier que l'identification est bonne.







### Cas Concret: Etudions le cas d'une boutique ecommerce



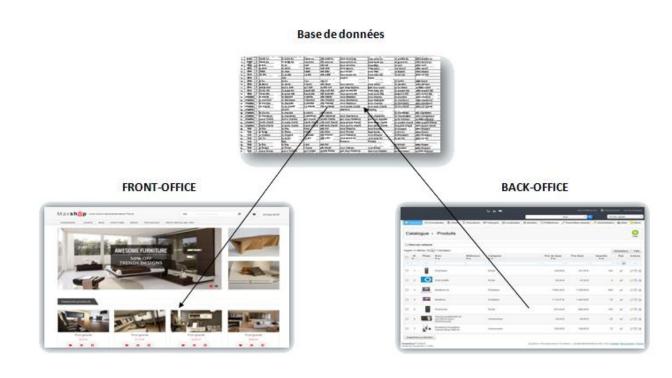
Les produits sont sélectionnés en BDD et affichés dans la page web.

Il y a 1 seule page fiche produit qui peut potentiellement présentés tous les produits (template).



### Front & Back: Fonctionnement d'un site web

- Chaque site est scindé en deux parties :
- Front -> affichage.
- Back -> paramétrage.
- Ce système peut exister grâce à la présence d'une base de données.





- CREATE
- DROP
- ALTER
- RENAME

## Créer une BDD: CREATE database

- Pour créer une base donnée en ligne de commande:
- CREATE DATABASE ma\_base
- CREATE DATABASE IF NOT EXISTS ma\_base
- On peut aussi rajouter des options:
  - On peut rajouter l'encodage avec un UTF-8
  - CREATE DATABASE ma\_base CHARACTER SET 'utf8';

## Créer une table : CREATE TABLE

- La commande de création de tables la plus simple ne comportera que le nom et le type de chaque colonne de la table.
- À la création, la table sera vide, mais un certain espace lui sera alloué. La syntaxe est la suivante :
  - Ex: CREATE TABLE nom\_table ()
- Supprimer une table : DROP TABLE
  - ▶ DROP TABLE nom\_table
- Modifier une table : ALTER TABLE
  - Ex: ALTER TABLE nom\_table {MODIFY} ()

- ► La syntaxe de base est:
- Create table: permet de créer une table en SQL
  - Ex: CREATE TABLE nom\_table (nom VARCHAR(100), prenom VARCHAR(100))
- Alter table permet de modifier une table existante.
  - ► Ex: ALTER TABLE utilisateur <u>ADD/DROP/MODIFY</u> adresse\_rue VARCHAR(255)
  - Pour renommer une colonne:
    - ► Ex: ALTER TABLE nom\_table MODIFY nom\_colonne type\_donnees
- Drop table permet de supprimer une table
  - ► Ex: DROP TABLE nom\_table

### Utilisation

- ▶ Select: permet d'afficher les champs de la table
  - ► Ex: select id, nom, prenom ou select \* ( \* Affiche tous les champs)
- ▶ From: extraire les données de la table
  - ► Ex: from Table
- Les options
  - Where condition
    - ► Ex: where id=2
  - And
    - ► Ex: and nom='moussa'
- ► Exemple complet:
  - ► Select \* from user where id=2 and nom='moussa'; // Affiche tous les champs du user ayant id =2 et s'appelle moussa dans la table user

- ► Insert into: permet d'ajouter en base
  - Ex: insert into user (nom, prenom), ("Camara", "Moussa") // ajoute moussa camara dans user
- Update: permet de modifier les infos
  - Ex: update user SET prenom = "mouskito" WHERE nom="Camara" // Change leprenom moussa par mouskito
- ▶ Delete: permet de supprimer de la base
  - Ex: delete from user //Supprime toute la table user
- Delete from
  - ► Ex: DELETE FROM user WHERE nom="Camara" AND prenom="Moussa"

## Exemple

```
CREATE database test;
 1
 2
       show databases;
 3 •
       use test;
       create table user (nom varchar(50), prenom varchar(30));
       use user;
 8
       insert into user(nom,prenom) value ('camara', 'prenom');
 9 •
10
       select * from user;
11 •
12
Export: Wrap Cell Content: TA
 nom
         prenom
 camara
        prenom
```

### Exercice: insert into

- Créer les tables stagiaire et centre dans votre base de données.
- ► Ajouter les stagiaires:
  - Moussa camara
  - Jean Dupont
  - Zack Henri
  - ► Machin Bidule
- Ajouter les centres: avec le nom, le lieu et specialité
  - ► Afpa ,Paris, CDA
  - ▶ Afpa, Marseille, Dev Mobile
  - Afpa,Lyon, marketing

### Les clauses

- ▶ GROUP BY : Cette clause permet de définir des groupes
  - ► Ex: SELECT \* FROM stagiaire GROUP BY nom
- ► HAVING : Cette clause permet de spécifier un filtre
  - Ex: SELECT \* FROM stagiaire GROUP BY nom HAVING nom LIKE 'camara'
  - // 1 Camara moussa
- UNION, INTERSECT et EXCEPT : Cette clause permet d'effectuer des opérations ensemblistes entre plusieurs résultats de requête
  - Ex: SELECT nom, penom FROM stagiaire UNION SELECT nom\_centre, lieu FROM centre
- ORDER BY : Cette clause permet de trier les n-uplets du résultat
  - ► Ex: SELECT \* FROM stagiaire ORDER BY penom

- ▶ LIMIT: permet de spécifier le nombre maximum de résultats que l'on souhaite obtenir
  - Ex: SELECT une\_colonne FROM une\_table LIMIT une\_limite;
    - ▶ SELECT \* FROM clients LIMIT 2;
      - ▶ SELECT \* FROM stagiaire ORDER BY penom LIMIT 2
- OFFSET permet de ne pas retourner une partie des résultats de la requête.
  - ► Ex: SELECT \* FROM clients LIMIT 2 OFFSET 1;
    - //sélectionner uniquement les 2 premiers résultats sans le tout premier.
      - ► Ex: SELECT \* FROM stagiaire ORDER BY penom LIMIT 3 OFFSET 1

## Expression de contraintes d'intégrité

- Une contrainte d'intégrité est une clause permettant de contraindre la modification de tables, faite par l'intermédiaire de requêtes d'utilisateurs, afin que les données saisies dans la base soient conformes aux données attendues.
- Ces contraintes doivent être exprimées dès la création de la table grâce aux mots clés suivants :
  - NULL / NOT NULL: précise si une valeur doit obligatoirement être saisie dans la colonne ou non
    - Exemple:

```
CREATE TABLE STAGIAIRE
```

(id INTEGER NOT NULL

nom VARCHAR(32) NOT NULL,

prenom VARCHAR(32) NULL,

date\_naissance DATE)

Crée une table dont les colonnes id et nom doivent obligatoirement être renseignés.

DEFAULT : valeur par défaut qui est placée dans la colonne lors des insertions et de certaines opération particulières, lorsque l'on a pas donné de valeur explicite à la colonne

```
CREATE TABLE STAGIAIRE

(

id INTEGER NOT NULL

nom VARCHAR(32) NOT NULL,

prenom VARCHAR(32) NULL,

date_naissance DATE,

genre CHAR(1) DEFAULT 'M'
)
```

## Clé primaire

- Dans une base de données relationnelle, une clé primaire est la donnée qui permet d'identifier de manière unique un enregistrement dans une table.
- Une clé primaire PRIMARY KEY identifie de façon unique chaque enregistrement dans une table de base de données.
- La clé primaire doit contenir des valeurs uniques.
- Une colonne de clé primaire ne peut pas contenir des valeurs NULL.
- Chaque table doit avoir une clé primaire, et chaque table peut avoir qu'une seule clé primaire.

#### SQL contrainte PRIMARY KEY CREATE TABLE

Le SQL suivante lorsque le "Personnes" table est créée pour créer une contrainte PRIMARY KEY sur la colonne "p\_id":

#### MySQL:

```
CREATE TABLE User

(

user_Id int NOT NULL,

nom varchar(255) NOT NULL,

prenom varchar(255),

addresse varchar(255),

ville varchar(255),

PRIMARY KEY (user_Id)

)
```

### SQL PRIMAIRE de contrainte KEY ALTER TABLE

- Lorsque la table a été créée, créer la contrainte PRIMARY KEY pour une colonne dans "user\_id", s'il vous plaît utiliser l'instruction SQL suivante:
- MySQL / SQL Server / Oracle / MS Accès:
  - ALTER TABLE Persons ADD PRIMARY KEY (user\_Id)

Pour nommer les contraintes de clé primaire, et de définir une pluralité de colonnes PRIMARY KEY, utilisez la syntaxe SQL suivante:

#### MySQL / SQL Server / Oracle / MS Accès:

► ALTER TABLE User ADD CONSTRAINT pk\_UserID PRIMARY KEY (user\_Id,nom)

NOTE: Si vous utilisez l'instruction ALTER TABLE pour ajouter une clé primaire, vous devez déclarer la colonne de clé primaire ne contient pas une valeur NULL

### Pour supprimer une contrainte PRIMARY KEY

#### MySQL:

► ALTER TABLE User

DROP PRIMARY KEY

#### ► SQL Server / Oracle / MS Access:

► ALTER TABLE User

DROP CONSTRAINT pk\_UserID

## Unicité (UNIQUE)

- La contrainte d'unicité exige que toutes les valeurs explicites contenues dans la colonne soient uniques au sein de la table.
- **Exemple:**

```
CREATE TABLE STAGIAIRE

(
nom VARCHAR(32) NOT NULL,
prenom VARCHAR(32) NULL,
Email varchar(50) UNIQUE
)
```



- ▶ INSERT INTO STAGIAIRE (nom, prenom, email) VALUES ('Sultan', 'Vanessa', 'manager@afpa.fr')
- INSERT INTO STAGIAIRE (nom, prenom, email) VALUES ('Kasi', 'Zack', 'test@test.fr')

Error Code: 1062. Duplicate entry 'test@test.fr' for key 'stagiaire.email'

### La contrainte CHECK

- La contrainte CHECK de validation est celle qui offre le plus de possibilité. En contre partie son exécution est très coûteuse.
- Elle permet de définir un prédicat complexe, basé sur une comparaison pouvant contenir une requête de type SELECT. Pour valider la contrainte, le prédicat doit être évalué à TRUE ou UNKNOWN (présence de NULL).

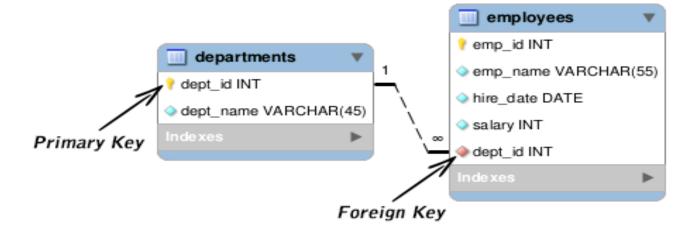
```
CREATE TABLE STAGIAIRE (
nom VARCHAR(32) NOT NULL,
prenom VARCHAR(32) NULL,
genre CHAR(1),
CHECK (genre IN ('M', 'F'))
);
```

### La contrainte de FOREIGN KEY

- La contrainte de type FOREIGN KEY permet de mettre en place une intégrité référentielle entre une (ou plusieurs) colonnes d'une table et la (ou les) colonne composant la clef d'une autre table afin d'assurer les relations existantes et joindre les tables dans le requête selon le modèle relationnel que l'on a défini.
- Une clé étrangère dans un des points de table à une clé primaire dans une autre table.
- La contrainte FOREIGN KEY est utilisé pour prévenir les actions qui détruiraient les liens entre les tables.
- La contrainte FOREIGN KEY empêche également des données non valides d'être insérée dans la colonne de clé étrangère, car elle doit être l'une des valeurs contenues dans le tableau il pointe.

### La contrainte de FOREIGN KEY

**Exemple:** 



```
Msg 547, Level 16, State 0, Line 2
The INSERT statement conflicted with the FOREIGN KEY constraint
"FK_CustomerO_Custo_6FE99F9F". The conflict occurred
in database "TestDB", table "dbo.Customers", column 'ID'.
The statement has been terminated.
```

## Exercices

- Créer les tables suivantes:
- Etudiant
  - Matricule, nom, prenom, date de naissance, niveau
- ► Cours
  - Code, nom du cours, enseignant
- Examen
  - matricule, code, notes

Correction en live

### BETWEEN

- ▶ BETWEEN permet de sélectionner des enregistrements en fonction d'une intervalle
- ▶ SELECT une\_colonne FROM une\_table WHERE une\_colonne BETWEEN "valeur\_1" AND "valeur\_2";
  - ► Ex: SELECT \* FROM commandes WHERE date\_ajout BETWEEN "2021-02-10" AND "2021-02-20";

### |N|

- ► IN permet de sélectionner des enregistrements si la valeur d'une colonne est comprise dans une liste de valeurs.
- ▶ Elle permet d'éviter d'avoir à recourir à de multiples OR.
- SELECT une\_colonne FROM une\_table WHERE une\_colonne IN ("valeur\_1", "valeur\_2", "valeur\_3");
  - Ex:
  - ▶ SELECT \* FROM commandes WHERE id\_commande = 3 OR id\_commande = 4 OR id\_commande = 5;
  - ▶ SELECT \* FROM commandes WHERE id\_commande IN(3, 4, 5);

### NOT IN

- NOT IN permet de sélectionner des enregistrements si la valeur d'une colonne n'est pas comprise dans une liste de valeurs.
- Ex: sélectionner les commandes qui n'ont pas les identifiants 1 ou 5.
  - ▶ SELECT \* FROM commandes WHERE id\_commande NOT IN(1, 5);

# Les opérateurs

- L'opérateur étoile (\*) permet de récupérer automatiquement tous les attributs de la table générée par la clause FROM de la requête.
  - ► SELECT \* FROM user

- Lorsque le SGBD construit la réponse d'une requête, il rapatrie toutes les lignes qui satisfont la requête, généralement dans l'ordre ou il les trouve, même si ces dernières sont en double (comportement ALL par défaut).
- C'est pourquoi il est souvent nécessaire d'utiliser le mot-clé DISTINCT qui permet d'éliminer les doublons dans la réponse.
  - SELECT DISTINCT nom FROM user

# Les opérations mathématiques de base

► Les maths ... (encore, ils sont partout)



- ▶ Il est possible d'utiliser les opérateurs mathématiques de base (+, -, \* et /) pour générer de nouvelles colonnes à partir, en général, d'une ou plusieurs colonnes existantes.
  - ► Ex: SELECT nom, prenom, salaire\*12 FROM employee

### L'opérateur AS

- Le mot-clé AS permet de renommer une colonne, ou de nommer une colonne créée dans la requête.
  - ► Ex1: SELECT surname AS nom, name AS prénom FROM user
  - Ex2: FROM nom\_de\_table AS nouveau\_nom

# Operation de comparaison

=	égal
!=	différent
<	strictement inférieur
<=	inférieur ou égal
>	strictement supérieur
>=	supérieur ou égal

## Operation de comparaison

Un prédicat simple peut également correspondre à un test de description d'une chaîne de caractères par une expression régulière :

~	décrit par l'expression régulière
~*	comme LIKE, mais sans tenir compte de
	la casse
!~	non décrit par l'expression régulière
!~*	comme NOT LIKE, mais sans tenir compte
	de la casse

L'opérateur LIKE 'chaîne' permet d'insérer des jokers dans l'opération de comparaison (alors que l'opérateur = teste une égalité stricte)

## Exemple de LIKE

- ► LIKE est inséparable des caractères jokers (wildcards) : % (pourcentage) et \_ (underscore).
- ▶ LIKE est un opérateur SQL qui permet d'effectuer une comparaison partielle, ce qui peut être une utile pour effectuer une recherche dans la base de données.
- % : il représente aucun, un seul ou plusieurs caractères. Il peut remplacer n'importe qu'elle chaîne de caractères
- ▶ Joker \_ : il représente un seul et unique caractère. Il remplace un seul et unique caractère.
  - Ex: SELECT \* FROM clients WHERE email LIKE "%@yeah.com";
  - ▶ SELECT \* FROM clients WHERE nom LIKE " e%";

# Exemple de LIKE

LIKE "a%"	Recherche toutes les chaînes de caractères qui <b>commencent par le caractère a</b> .
LIKE "%a"	Recherche toutes les chaînes de caractères qui <b>terminent par le caractère a</b> .
LIKE "%a%"	Recherche toutes les chaînes de caractères qui <b>contiennent au moins un</b> caractère a.
LIKE "a%b"	Recherche toutes les chaînes de caractères qui <b>commencent par le caractère a et terminent par le caractère b</b> .
LIKE "a"	Recherche toutes les chaînes de caractères de trois caractères qui commencent par le caractère a.
LIKE "_a%"	Recherche toutes les chaînes de caractères qui <b>possèdent le caractère a en deuxième position</b> .

## Exemple

- ► EX: SELECT nom FROM stagiaire WHERE nom LIKE 'A%'
  - recherche tous les noms commençant par le caractère "A", il faut donc écrire "A"" pour que % soit égale à n'importe quelle suite de caractères.
- SELECT nom FROM stagiaire WHERE nom LIKE '%ma%'
  - cherche tous les noms contenant la chaîne de caractères 'ma',
- SELECT nom FROM stagiaire WHERE nom LIKE 'ca\_\_\_\_%'
  - Recherche des stagiaires dont le prénom comporte exactement 6 caractéres et qui commence par ca
- SELECT \* FROM stagiaire WHERE prenom LIKE "\_\_\_\_";
  - Recherche des stagiaires dont le prénom comporte exactement 4 caractéres

# Operation de comparaison

▶ Un prédicat simple peut enfin correspondre à l'un des tests suivants :

expr IS NULL	test sur l'indétermination de expr
expr IN (expr_1 [])	comparaison de expr à une liste de valeurs
expr NOT IN (expr_1 [])	test d'absence d'une liste de valeurs
expr IN (requête) expr NOT IN (requête)	même chose, mais la liste de valeurs est le résultat d'une sous-requête qui doit impérativement retourner une table ne contenant qu'une colonne
EXIST (requête)	vraie si la sous-requête retourne au moins un n-uplet
expr operateur ANY (requête)	vraie si au moins un n-uplet de la sous-requête vérifie la comparaison « expr opérateur n-uplet » ; la sous-requête doit impérativement retourner une table ne contenant qu'une colonne ; IN est équivalent à = ANY
expr operateur ALL (requête)	vraie si tous les n-uplets de la sous-requête vérifient la comparaison « expr opérateur n-uplet » ; la sous-requête doit impérativement retourner une table ne contenant qu'une colonne

# Opération de logique

- ▶ OR
  - Ex: SELECT une\_colonne FROM une\_table WHERE une\_condition OR encore\_condition;
- AND
  - ► Ex: SELECT une\_colonne FROM une\_table WHERE une\_condition AND une\_autre\_condition;
- ► NOT
  - Ex: SELECT une\_colonne FROM une\_table WHERE une\_condition AND NOT une\_autre\_condition;

### EXERCICE

- Créer les tables:
  - Avion
    - NumeroAv, NomAv, Capacité, Localisation
  - Vol
    - ▶ NumVol, NumPil, NumAv, Ville\_depart, Ville\_arrive, Heure\_dep, heure\_arrivee
  - Pilote
    - ▶ NomPil, NumPil, Adresse, Salaire

### Insert into

### Avions

- ▶ Avion 1: 00001, Alpha, 300, Paris
- Avion 2: 00002, Delta,400, Milan
- Avion 3: 00002, Delta,350, Rome

#### ► Vols:

- ▶ Vol 1 : AF351, 152010, 00001, Paris, new York, 9h00, 17h00
- ▶ Vol 2: RU154, 458950, 00002, Canada, Londres, 13h00, 22h00
- ▶ Vol 3 : LH1792, 538710, 00003, Paris, Tunis 11h00, 21h00

### Pilotes:

- ▶ Pilote1: Moussa, 00001, Paris, 650 000
- ▶ Pilote2: Afpa, 00003, Marseille, 900 000
- ▶ Pilote3: Zack, 00002, Cergy, 750 901
- ▶ Pilote4: Zack, 00002, Cergy, 350 000

► Correction en live

## Les types de données

- Les types de données peuvent être :
  - ► INTEGER:
    - ▶ Ce type permet de stocker des entiers signés codés sur 4 octets.
  - ▶ BIGINT:
    - ▶ Ce type permet de stocker des entiers signés codés sur 8 octets.
  - ► REAL:
    - ▶ Ce type permet de stocker des réels comportant 6 chiffres significatifs codés sur 4 octets.
  - ► DOUBLE PRÉCISION :
    - ▶ Ce type permet de stocker des réels comportant 15 chiffres significatifs codés sur 8 octets.
  - ► NUMERIC[(précision, [longueur])]:
    - ► Ce type de données permet de stocker des données numériques à la fois entières et réelles avec une précision de 1000 chiffres significatifs. longueur précise le nombre maximum de chiffres significatifs stockés et précision donne le nombre maximum de chiffres après la virgule.

## Les types de données

- CHAR(longueur):
  - Ce type de données permet de stocker des chaînes de caractères de longueur fixe. longueur doit être inférieur à 255, sa valeur par défaut est 1.
- ► VARCHAR(longueur):
  - Ce type de données permet de stocker des chaînes de caractères de longueur variable. longueur doit être inférieur à 2000, il n'y a pas de valeur par défaut.
- DATE:
  - ▶ Ce type de données permet de stocker des données constituées d'une date.
- ► TIMESTAMP:
  - ▶ Ce type de données permet de stocker des données constituées d'une date et d'une heure.
- BOOLEAN:
  - ▶ Ce type de données permet de stocker des valeurs Booléenne.
- MONEY:
  - ► Ce type de données permet de stocker des valeurs monétaires.
- ► TEXT:
  - ▶ Ce type de données permet de stocker des chaînes de caractères de longueur variable.

## Seriesd'exercices

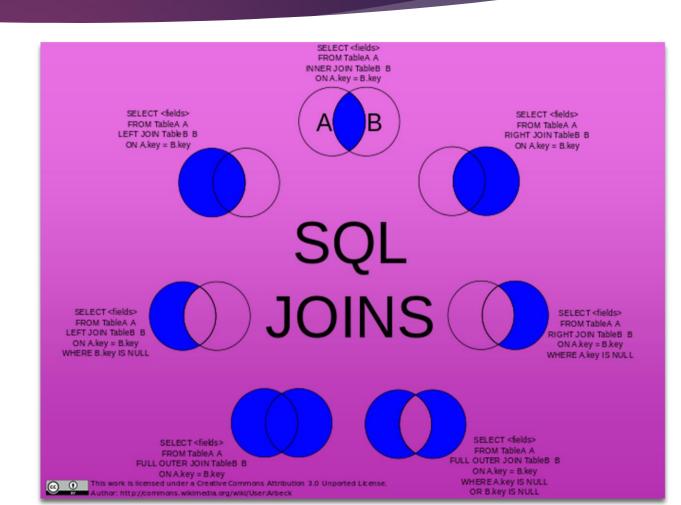
- ► Series\_exercices1
  - Exercice 1,2,3

▶ Il existe une notion simple en apparence, mais assez puissante pour manipuler des bases de données. Ce sont les jointures



# C'est quoi une jointure SQL déjà?

- Une jointure permet de créer une liaison entre deux ou plusieurs tables pour pouvoir récupérer un résultat bien précis.
- Ce résultat est en quelque sorte une fusion virtuelle de plusieurs tables. Chaque entrée de chaque table est retournée dans le résultat en fonction de la jointure utilisée, ça peut vite faire beaucoup de lignes!



# Les différents types de jointures SQL

► Il existe plusieurs types de jointures et il est facile de s'emmêler les pinceaux lorsqu'on commence à faire des requêtes SQL complexes avec des jointures en tout genre.

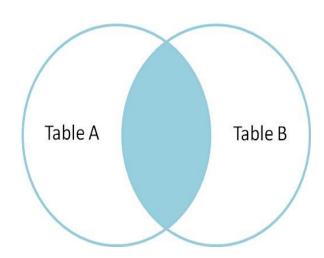


### INNER JOIN

► INNER JOIN, aussi appelée EQUIJOIN, est un type de jointures très communes pour lier plusieurs tables entre-elles. Cette commande retourne les enregistrements lorsqu'il y a au moins une ligne dans chaque colonne qui correspond à la condition.

### **Exemple:**

SELECT \* FROM stagiaire
INNER JOIN centre
on stagiaire.id = centre.id\_centre





- **Exemple:**
- La liste des noms des musiciens et des titres et les titres des représentations auxquelles ils participent.

from exercice1\_festival\_musicien,
exercice1\_festival\_programmer,exercice1\_festival\_re
presentation
where exercice1\_festival\_musicien.num\_rep =
exercice1\_festival\_programmer.num\_rep and
exercice1\_festival\_musicien.num\_rep =
exercice1\_festival\_musicien.num\_rep;

SELECT mus.nom, rep.titre

FROM exercice1\_festival\_musicien as mus

INNER JOIN exercice1\_festival\_representation as rep

ON rep.Num\_rep=mus.Num\_rep

### Requete avec where

SELECT CLIENT.NomClient,
 COMMANDE.NumCde
FROM CLIENT, COMMANDE
WHERE CLIENT.NumClient = COMMAND
E.NumClient

### Requete avec inner join

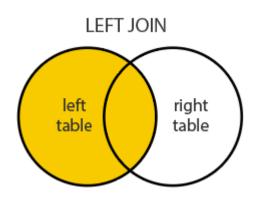
SELECT CLIENT.NomClient,
 COMMANDE.NumCde
FROM (CLIENT INNER JOIN COMMANDE
ON CLIENT.NumClient = COMMANDE.
 NumClient)

### Left Join

- ▶ LEFT JOIN(LEFT OUTER JOIN) est utilisé pour renvoyer la partie « GAUCHE » c'est à dire que toutes les lignes de la table gauche seront retournées, même s'il n'y a pas de ligne correspondante dans la table droite.
- Jointure la plus courante. Permet de sélectionner :
  - les enregistrements satisfaisant à la jointure.
  - les enregistrements du premier fichier cité qui ne satisfont pas la jointure. Le nom du premier fichier cité est placé à gauche de la jointure OUTER JOIN.

### left join:

SELECT \* FROM stagiaire LEFT JOIN inscrit on stagiaire.id = inscrit.id\_stagiaire //Where inscrit.id\_stagiaire is not null;



- ▶ Le LEFT OUTER JOIN spécifie que c'est la <u>première table citée qui doit rester complète.</u>
- Dans notre exemple, cela devient :

SELECT \*

FROM personnes p

LEFT OUTER JOIN ingenieurs i ON i.personne\_id = p.personne\_id;

Voilà, avec ça vous avez toute la table "personnes" complétée avec les données de la table "ingenieurs" si elles existent.

# Right Join

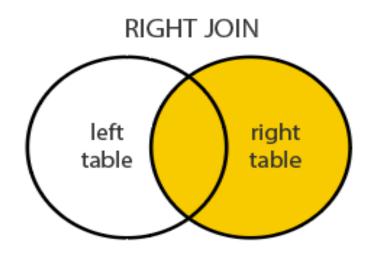
▶ RIGHT JOIN fait la même chose que LEFT JOIN , seulement elle retourne toute la table T2 au lieu de T1

Right join

SELECT \* FROM centre

RIGHT JOIN inscrit

on centre.id\_centre = inscrit.id\_centre



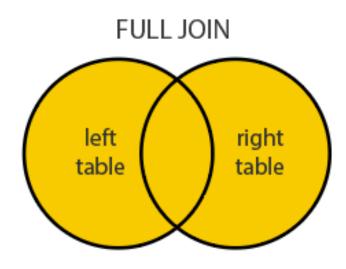
### Full Join

▶ Il permet de faire une jointure entre 2 tables. L'utilisation de cette commande permet de combiner les résultats des 2 tables, les associer entre eux grâce à une condition et remplir avec des valeurs NULL si la condition n'est pas respectée.

NB: n'est toujours pas utilisable en l'état en MySQL mais qu'on va pouvoir émuler en utilisant d'autres ressources.

### Exemple:

select \* from stagiaire union select \*from inscrit;



### Cross Join

- Le CROSS JOIN en MySQL va retourner une liste de résultat qui va être le produit des entrées des deux tables jointes lorsqu'aucune clause WHERE n'est utilisée.
  - ▶ SELECT \*
  - ▶ FROM table1
  - ► CROSS JOIN table2
- Ex:
- select \* from stagiaire cross join inscrit;
- ▶ IL nous retourne le produit cartésien des deux tables: A \* B

### Union

- UNION pour combiner les résultats de deux déclarations SELECT.
- Afin que cet opérateur fonctionne correctement et que les résultats puissent bien être combinés, il va falloir respecter les règles suivantes :
- Chacun des SELECT dans un UNION doit posséder le même nombre de colonnes ;
- ▶ Les colonnes doivent posséder le même type de données une à une dans chaque SELECT (la première colonne du premier SELECT doit posséder des données de même type que la première colonne du deuxième SELECT et etc.)
- NB: l'opérateur SQL UNION ne va pas sélectionner les doublons dans les valeurs (les valeurs qui sont trouvées plusieurs fois n'apparaitront qu'une seule fois).

### EXCEPT: PostgreSQL // MINUS: MySQL et Oracle

- ► EXCEPT s'utilise entre 2 instructions pour récupérer les enregistrements de la première instruction sans inclure les résultats de la seconde requête.
- ▶ Si un même enregistrement devait être présent dans les résultats des 2 syntaxes, ils ne seront pas présent dans le résultat final.

SELECT nom, prénom
FROM Table 1
WHERE Table 1.nom NOT IN
(SELECT nom FROM Table 2)
AND Table 1.prénom NOT IN
(SELECT prénom FROM Table 2);

SELECT nom, prénom FROM Table 1 EXCEPT SELECT nom, prénom FROM Table 2;

# Les fonctions d'agrégation

- AVG([DISTINCT | ALL] expression): Calcule la moyenne des valeurs de l'expression expression. //SELECT AVG(commandes) FROM clients;
- COUNT(\* | [DISTINCT | ALL] expression): Dénombre le nombre de lignes du groupe. Si expression est présent, on ne compte que les lignes pour lesquelles cette expression n'est pas NULL. // <u>SELECT COUNT(email) FROM clients;</u>
- MAX( [ DISTINCT | ALL ] expression ) : Retourne la plus petite des valeurs de l'expression expression.
- ▶ MIN([ DISTINCT | ALL ] expression ) :Retourne la plus grande des valeurs de l'expression expression.
- ▶ SUM([ DISTINCT | ALL ] expression) : Calcule la somme des valeurs de l'expression expression. // SELECT SUM(commandes) AS somme FROM clients;
- ▶ VARIANCE([ DISTINCT | ALL ] expression) :Calcule la variance des valeurs de l'expression expression.
- DISTINCT indique à la fonction de groupe de ne prendre en compte que des valeurs distinctes.
- ALL indique à la fonction de groupe de prendre en compte toutes les valeurs, c'est la valeur par défaut.

# Series d'exercices – les jointures

Series\_exercices\_jointures.pdf



### Les utilisateurs

- Mysql nous propose un utlisateur par defaut "root"
- ▶ On peut tout de même rajouter ou supprimer des utilisateurs avec des privileges.
- Pour voir tous les utilisateurs.
  - Select user from mysql.user;





- ▶ Pour voir l'utilisateur courant.
  - Select current\_user from mysql.user;
    - // root@localhost;

### Creation

- ▶ Pour creation un nouvel utilisateur mysql
  - Create user 'nom\_nouvel\_utilisateur' identified by 'mot de passe';
  - Ex:
    - create user 'moussa@localhost' identified by 'paris';

### Renommer un utilisateur

- Pour renommer un utilisateur déjà créé, la commande est la suivante.
  - ▶ RENAME USER 'user'@'localhost' TO 'user2'@'localhost';

## Changer un mot de passe

- Le changement de mot de passe suit la même logique que pour son attribution lors de la création d'un utilisateur.
  - ALTER USER 'afpa@localhost' IDENTIFIED BY 'moussa1';

# Mettre à jour mes modifications

- Vous pouvez utilser Flush qui permet de vider les caches et prendre nos modifications
- Flush privileges
- Maintenant on peut se connecter avec notre nouvel utilisateur.
- Probleme quand on fait un show databases; nous n'avons aucune données.
- Nous devons lui donner certains priviléges

# Les priviléges

- Le root a tous les privileges par défaut, on peut rajouter des privileges à vos nouveaux utilisateurs avec des options
- grant all on \*.\* to nom utilisateur;
- EX:
  - grant all on BDD. \*to afpa; // donne tous les droits sur toutes tables de la BDD
  - grant all on \*.\* to afpa; // donne tous les droits sur les BDD

https://dev.mysql.com/doc/refman/8.0/en/grant.html

# Type de privileges ou permissions

- CREATE Permet aux utilisateurs de créer des bases de données/tableaux
- SELECT Permet aux utilisateurs de récupérer des données
- ▶ INSERT Permet aux utilisateurs d'ajouter de nouvelles entrées dans les tableaux
- ▶ UPDATE Permet aux utilisateurs de modifier les entrées existantes dans les tableaux
- ▶ DELETE Permet aux utilisateurs de supprimer les entrées de la tableau
- ▶ DROP Permet aux utilisateurs de supprimer des bases de données/tableaux entiers

# Exemple Ajout

- Maintenant, attribuons des privilèges à un utilisateur sur cette base de données.
  - GRANT SELECT, INSERT, UPDATE, DELETE ON `database`.\* TO 'user'@'localhost';
  - Ou lui donner tous les droits
    - ▶ GRANT ALL ON `database`.\* TO 'user'@'localhost';
- Pour que les nouveaux droits attribués soient pris en compte, il est nécessaire de lancer la requête FLUSH.
  - ► FLUSH PRIVILEGES;

- Si on peut donner des droits à un utilisateur on peut bien aussi lui retirer ses droits.
- Avec revoke
  - ► REVOKE TYPE\_DE\_PERMISSION ON base\_de\_donnees.table FROM 'nom\_utilisateur'@'localhost';
- Exemple: pour révoquer tous les privilèges de notre utilisateur non-root, nous devrions utiliser:

REVOKE ALL PRIVILEGES ON \*.\* FROM 'nom\_utilisateur'@'localhost';

Ou

REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'user'@'localhost';



- ▶ REVOKE DELETE ON database.\* FROM 'user'@'localhost';
- ► Exemple:
  - ▶ REVOKE SELECT ON cda314.\* FROM 'afpa@localhost';

### Procédures stockées

- On peut ulitilsier des procedures stockées permettant d'inserer ou manipuler les données.
- Pour cela, selectionner la base de donnée courante et dans stored procedures:
  - Created stored
  - CREATE PROCEDURE `new\_procedure` ()BEGINEND et APPLY
  - EX:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `new_procedure`()

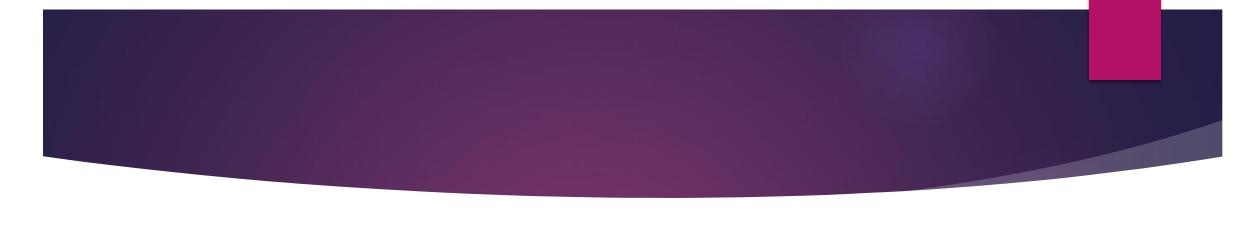
BEGIN

INSERT INTO spectacle (titre,date_deb, duree,salle_id,chanteur) values ('comedi','2021-12-25', 60,1,'JCC');

INSERT INTO spectacle (titre,date_deb, duree,salle_id,chanteur) values ('test','2021-12-31', 30,2,'JCC');

INSERT INTO spectacle (titre,date_deb, duree,salle_id,chanteur) values ('comedi','2021-12-25', 60,1,'JCC');

END
```



- Pour apeller la procedure,
  - Call nom\_de\_votre\_procédure

### QUESTIONS

