HEIG VD | MEI
Media Engineering Institute

# 03 – Introduction à SQL

## Infrastructure de données 1

Noemi Romano | 07 mars 2025

# Semaine passée

- **Pourquoi modéliser**

# Semaine passée

- **Pourquoi modéliser** Organiser l’information, besoins clients

# Semaine passée

- **Pourquoi modéliser** Organiser l'information, besoins clients
- **Types de modèles**

# Semaine passée

- **Pourquoi modéliser** Organiser l’information, besoins clients
- **Types de modèles** Conceptuel, Logique, Physique

# Semaine passée

- **Pourquoi modéliser** Organiser l'information, besoins clients
- **Types de modèles** Conceptuel, Logique, Physique
- **Contraintes**

# Semaine passée

- **Pourquoi modéliser** Organiser l'information, besoins clients
- **Types de modèles** Conceptuel, Logique, Physique
- **Contraintes** Clé primaires/étrangère, Unicité, Nullabilité

# SQL

# S Q L

Structured Query Language

📌 1970 : Edgar F. Codd, chercheur chez IBM, publie un article sur le modèle relationnel des bases de données, qui deviendra la base de SQL

📌 1974 : IBM développe un langage appelé SEQUEL (Structured English Query Language) pour interagir avec ses bases de données relationnelles

📌 1979 : Oracle (alors appelé Relational Software Inc.) commercialise la première version de SQL, devenant le premier SGBD (Système de Gestion de Bases de Données) relationnel basé sur SQL

📌 1986 : SQL devient un standard reconnu par l'ANSI (American National Standards Institute)

📌 1987 - Aujourd'hui : SQL continue d'évoluer avec des ajouts comme les requêtes récursives, les vues matérialisées, les fonctions analytiques, et de nombreuses variantes adaptées aux besoins modernes (PostgreSQL, MySQL, SQL Server...).

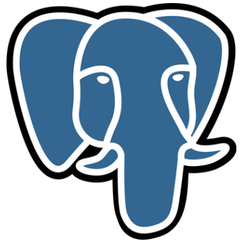
# SGBD

# Systèmes de gestion de base de données (SGBD)

## OLTP (Online Transaction Processing)

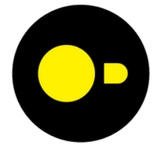
Bases optimisées pour les transactions rapides et fréquentes





## OLAP (Online Analytical Processing)

Bases optimisées pour l'analyse de grands volumes de données.

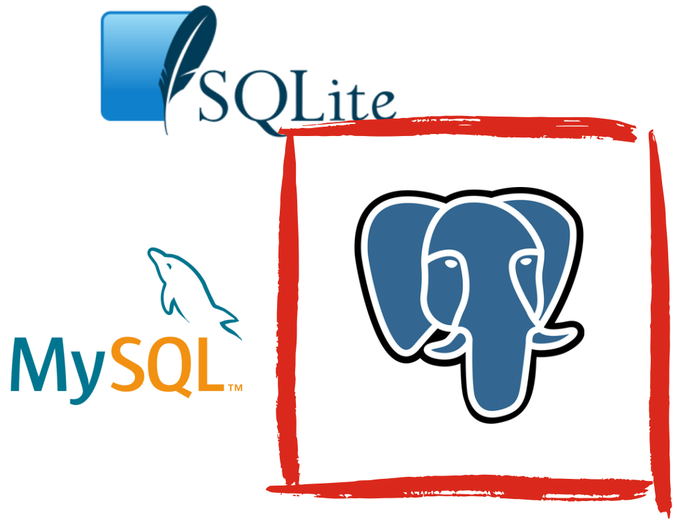


DuckDB


# Systèmes de gestion de base de données (SGBD)

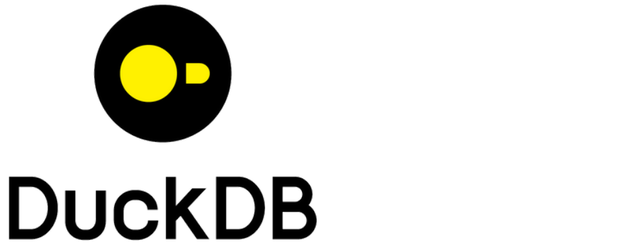
## OLTP (Online Transaction Processing)

Bases optimisées pour les transactions rapides et fréquentes




## OLAP (Online Analytical Processing)

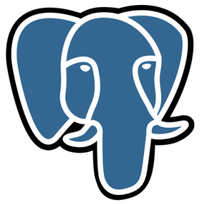
Bases optimisées pour l'analyse de grands volumes de données.


# Installation

# Environnement de travail

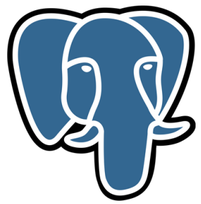
- Systeme de gestion de base de données (**SGBD**)



PostgreSQL

- Editeur **SQL**

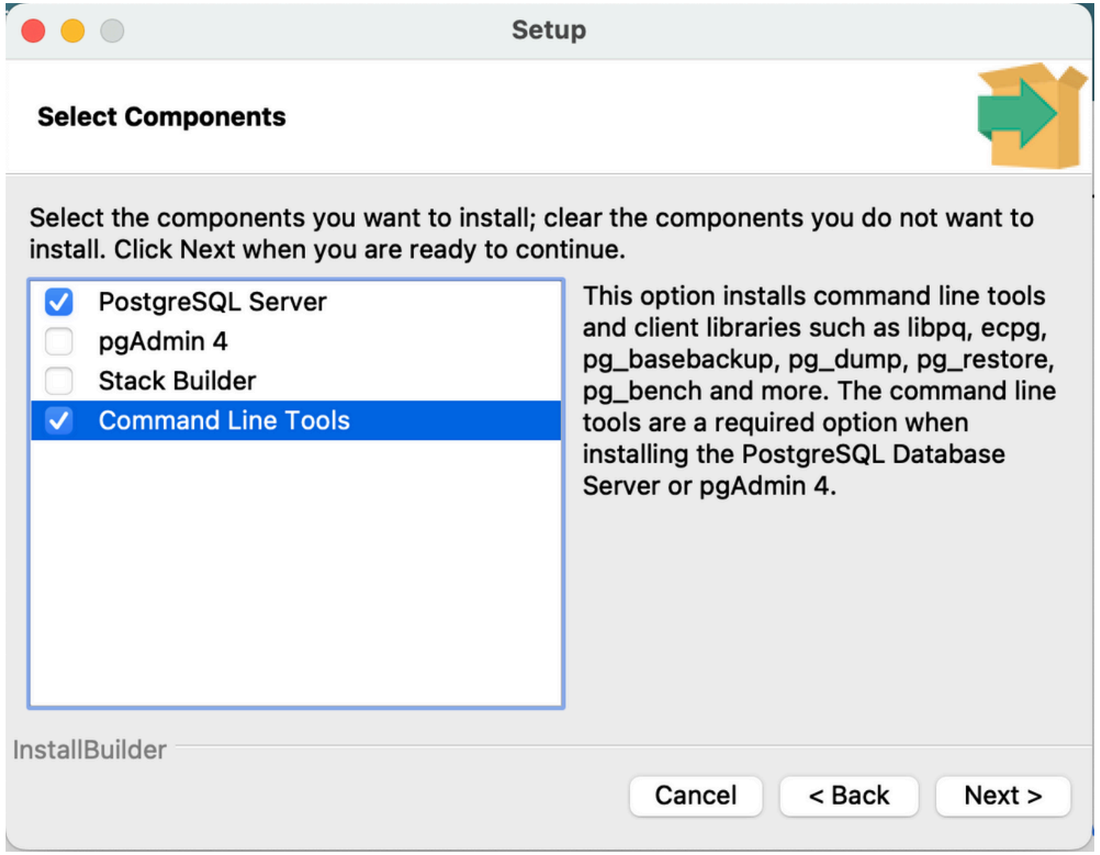
# Installation



PostgreSQL

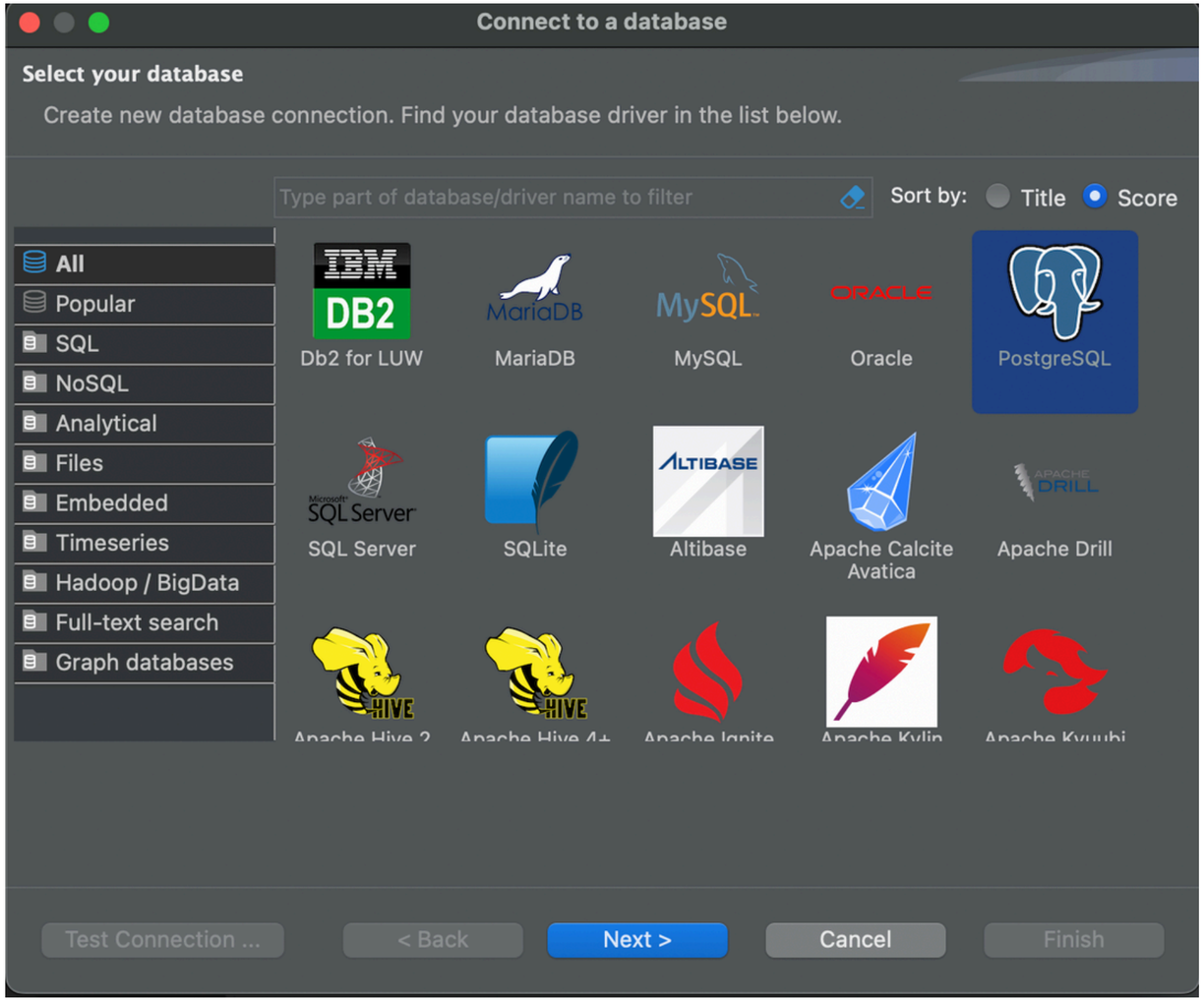
# Installation

PostgreSQL



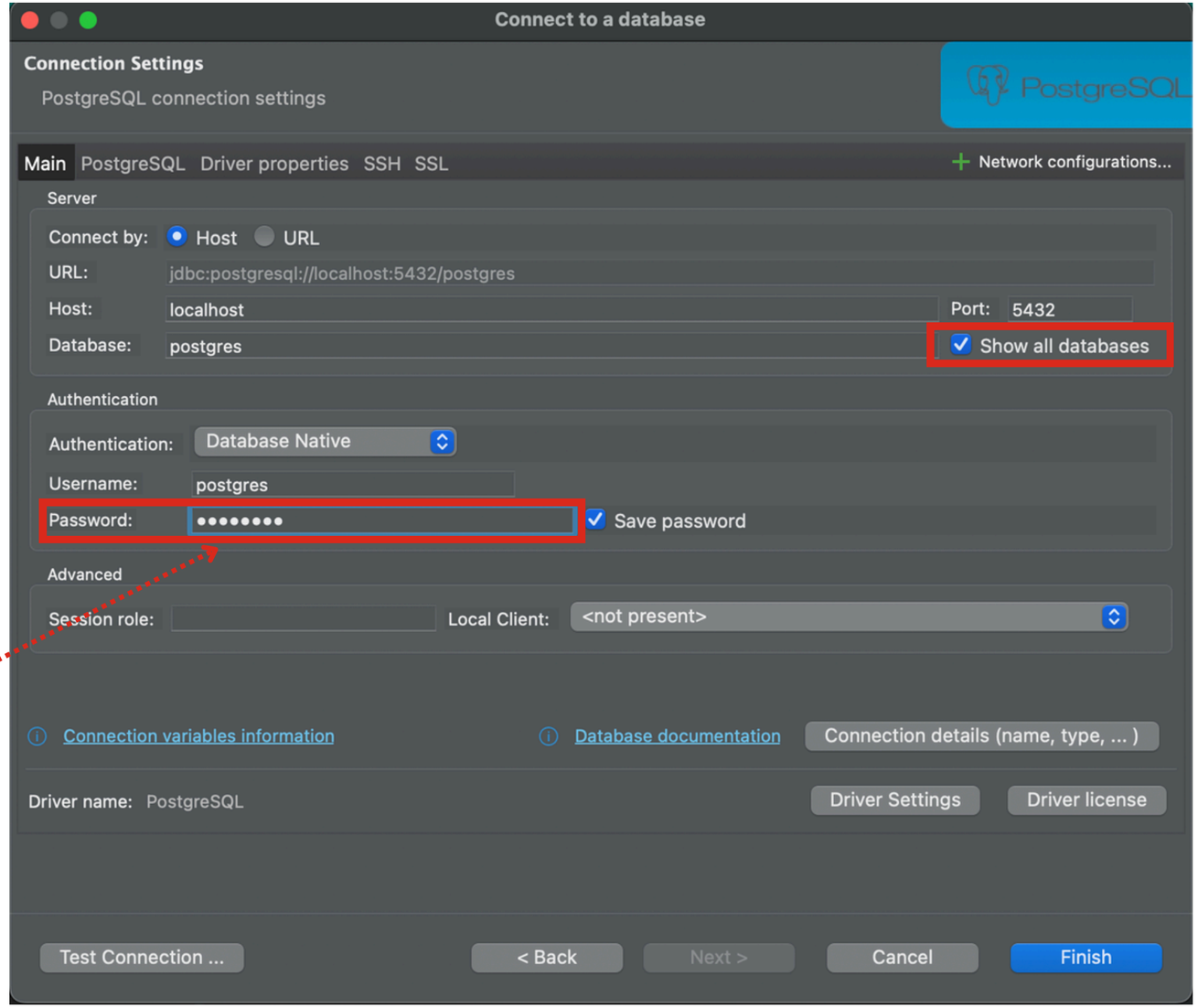
# Connexion

DBeaver



# Connexion


Le mot de passe que vous avez choisi lors de l'installation PostgreSQL



# Syntaxe SQL

# Syntaxe SQL

- **Langage déclaratif** : on décrit ce qu'on veut comme résultat, mais pas comment le faire

- **Data Definition Language (DDL) :** créer manipuler les *objets du modele de données*

  `CREATE`, `DROP`, `ALTER`, `RENAME`, `TRUNCATE`

- **Data Manipulation Language (DML) :** manipuler les *données*

  `INSERT`, `SELECT`, `UPDATE`, `DELETE`

# Objectifs

1. Créer une base de données
2. Lire les données
3. Mettre à jour les données
4. Supprimer des données
5. Filtrer les données
6. Agréger les données
7. Trier les données
8. Mettre en relation les données

# ❶ Créer

# Créer

- **CREATE** : créer un objet de la base de données (i.e. **DATABASE**, **TABLE**, **INDEX** etc.)

- **INSERT INTO** : ajouter une ou plusieurs lignes (**VALUES**) dans une table

Insérer des données dans une base de données directement à partir d'un fichier CSV :
**COPY** table_name(column1, column2, ...)
**FROM** *chemin/du/fichier.csv* [options]

**COPY** command PostgreSQL

```sql
# Créer une base de données
CREATE DATABASE database_name;

# Créer une table
CREATE TABLE table_name (
 column1 type,
 column2 type,
 column3 type,
 ....
);


INSERT INTO table_name
(column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

# Type de données

PostgreSQL Datatypes

| Catégorie | Type PostgreSQL | Description |
|---|---|---|
| **Numériques** | `SMALLINT` | Entier sur 2 octets (-32,768 à 32,767) |
| | `INTEGER` | Entier sur 4 octets (-2,147,483,648 à 2,147,483,647) |
| | `BIGINT` | Entier sur 8 octets (très grand nombre entier) |
| | `DECIMAL (NUMERIC)` | Nombre décimal avec précision définie |
| | `REAL` | Nombre flottant 4 octets (moins précis) |
| | `SERIAL / BIGSERIAL` | Auto-incrémentation d'entiers (clé primaire) |
| **Chaine de caractères** | `CHAR(n)` | Chaîne de longueur fixe |
| | `VARCHAR(n)` | Chaîne de longueur variable jusqu'à n caractères |
| | `TEXT` | Chaîne de longueur illimitée |
| **Booléen** | `BOOLEAN` | Valeurs TRUE, FALSE ou NULL |
| **Dates et heures** | `DATE` | Stocke une date (YYYY-MM-DD) |
| | `TIME` | Stocke une heure (HH:MI:SS) |
| | `TIMESTAMP` | Date et heure sans fuseau horaire |
| | `TIMESTAMP WITH TIME ZONE` | Date et heure avec fuseau horaire |
| | `INTERVAL` | Durée ou différence entre deux dates |
| **JSON/XML** | `JSON` | Stocke du JSON brut |
| | `JSONB` | JSON optimisé et indexable |
| | `XML` | Stocke des documents XML |
| **Autres** | `UUID` | Identifiant unique universel |
| | `BYTEA` | Stocke des données binaires (images, fichiers) |
| | `ARRAY` | Stocke un tableau de valeurs |
| | `ENUM` | Type énuméré défini par l'utilisateur |

# Contraintes

PostgreSQL DDL constraints

| Commande | Description | Exemple |
|---|---|---|
| `NOT NULL` | Garantit qu'une colonne ne peut pas contenir de valeurs NULL. | `CREATE TABLE users (name TEXT NOT NULL);` |
| `UNIQUE` | Assure que toutes les valeurs d'une colonne sont uniques. | `CREATE TABLE users (email TEXT UNIQUE);` |
| `PRIMARY KEY` | Identifie de manière unique chaque ligne d'une table (implique NOT NULL et UNIQUE). | `CREATE TABLE users (id SERIAL PRIMARY KEY);` |
| `FOREIGN KEY` | Assure l'intégrité référentielle entre les tables. | `CREATE TABLE orders (id SERIAL PRIMARY KEY, customer_id INTEGER REFERENCES customers(id));` |
| `CHECK` | Valide les données selon une condition. | `CREATE TABLE users (age INTEGER, CHECK (age >= 18));` |
| `DEFAULT` | Définit une valeur par défaut pour une colonne. | `CREATE TABLE users (status TEXT DEFAULT 'active');` |
| `AUTO_INCREMENT` | Génère automatiquement des valeurs uniques (avec SERIAL). | `CREATE TABLE users (id SERIAL PRIMARY KEY);` |
| `CONSTRAINT` | Permet de nommer une contrainte et l'appliquer explicitement. | `CREATE TABLE orders (id SERIAL, customer_id INTEGER, CONSTRAINT fk_customer FOREIGN KEY (customer_id) REFERENCES customers(id));` |

# Challenge

## Films

| id | title | release_year |
|---|---|---|
| 1 | Toy Story | 1995 |
| 2 | A Bug's Life | 1998 |
| 3 | Toy Story 2 | 1999 |
| 4 | Monsters, Inc. | 2001 |
| ... | ... | ... |

## Boxoffice

| movie_id | rating | domestic_sales | international_sales |
|---|---|---|---|
| 3 | 7.9 | 245852179 | 239163000 |
| 1 | 8.3 | 191796233 | 170162503 |
| 4 | 8.1 | 289916256 | 272900000 |
| 2 | 7.2 | 162798565 | 200600000 |
| ... | ... | ... | ... |

Accès aux fichiers CSV

# Challenge

- Créer une base de données **movies_db**

- Créer une table **movies,** ajouter les contraintes d’intégrité, et charger les données

- Créer une table **boxoffice,** ajouter les contraintes d’intégrité, et charger les données

# ❷ Lire

# 2 Lire

- **SELECT** : choisir une ou plusieurs colonnes spécifiques (*column_name*) ou toutes les colonnes (*).

- **FROM**: à partir d'une table (*table_name*)

```
SELECT column_name1,
column_name2, ...
FROM table_name;

SELECT *
FROM table_name;
```

Utiliser des alias (**AS**) pour renommer les colonnes dans vos résultats. Par exemple:
**SELECT** movie_title **AS** 'Title'
**FROM** movies;

# Challenge

- Lister toutes les données de **movies**
- Lister tous les données de **boxoffice**
- Lister l'*identifiant* et le *titre* de tous les films

# 3 Mettre à jour

# 3 Mettre à jour

- **UPDATE** : mettre à jour une table et établir (**SET**) une valeur d'une colonne

**UPDATE** PostgreSQL

```
UPDATE table_name
SET column_name = <value>;


UPDATE table_name
SET column_name_1 = <value>,
column_name_2 = <value>;
```

# 3 Mettre à jour

- **ALTER TABLE**: modifier la structure d'une table existante
  - Ajouter une colonne **ADD COLUMN**
  - Supprimer des colonnes **DROP COLUMN**
  - Ajouter des contraintes **ADD CONSTRAINT**
  - Supprimer des contraintes **DROP CONSTRAINT**
  - Renommer des colonnes **RENAME COLUMN**

**ALTER TABLE** PostgreSQL

# Challenge

- Créer une colonne *category*
- Mettre à jour la catégorie avec la valeur “Animation”

# ❹ Supprimer

# 4 Supprimer

- **DELETE** : supprimer les lignes d'une table

- **DROP** : supprimer un objet de la base de données (**DATABASE**, **TABLE**, **INDEX** etc.)

```
# Supprimer toutes les lignes
DELETE FROM table_name;

# Supprimer la base de données
DROP DATABASE database_name;

# Supprimer une table
DROP TABLE table_name;
```

# Challenge

- Effacer les données de la table **boxoffice**
- Supprimer la table **boxoffice**

# 5 Filtrer

# 5 Filtrer (WHERE)

- **WHERE** : filtre les enregistrements dans une table en fonction de conditions spécifiques (si plusieurs, enchaîner avec **AND** ou **OR**)

- Utilisé avec : **SELECT**, **UPDATE**, **DELETE**, **INSERT**

- Comparaisons: égalité (**=**), inégalité (**!=**), supérieure/inférieure (**<**, **>**, **<=**, **>=**), châine de charactères (**LIKE**), **BETWEEN**, **IN**, **IS NULL** etc.

```
SELECT *
FROM table_name
WHERE column_name > value;

UPDATE table_name
SET column_name = value
WHERE column_name = value;

SELECT *
FROM table_name
WHERE column_name LIKE ‘value’
AND column_name2 > value;
```

# 5 Filtrer (LIMIT)

- **LIMIT N**: restreindre le nombre de lignes retournées par une requête

- **OFFSET M**:  ignorer un certain nombre de lignes avant de commencer à retourner les résultats

```
SELECT *
FROM table_name
LIMIT 10 OFFSET 8;
```

# Challenge

- Lister les films sortis après 2000
- Lister tous les films Toy Story
- Effacer les films qui sont sortis avant 2000

# 6 Agréger

# 6 Agréger

- **GROUP BY** : regrouper les lignes d'un jeu de résultats en fonction d'une ou plusieurs colonnes

- Fonctions agrégation: **SUM**, **MAX**, **MIN**, **AVG**, **COUNT** etc.

```
SELECT column1,
function_name(column2), ...
FROM table_name
GROUP BY column1, column2, ...;
```

# EXEMPLE

- Créer une colonne *epoque*
- Assigner une époque (90's, 00's, 10's) en fonction de l'année de sortie de chaque film
- Compter le nombre de films sortis pour chaque époque

# ❼ Trier

# 7 Trier

- **ORDER BY** : tri les résultats d'une requête en fonction d'une ou plusieurs colonnes (**ASC**, **DESC**)

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1 [ASC | DESC], column2
[ASC | DESC], ...;
```

# EXEMPLE

- Lister les films par ordre décroissant en fonction de l'année de sortie

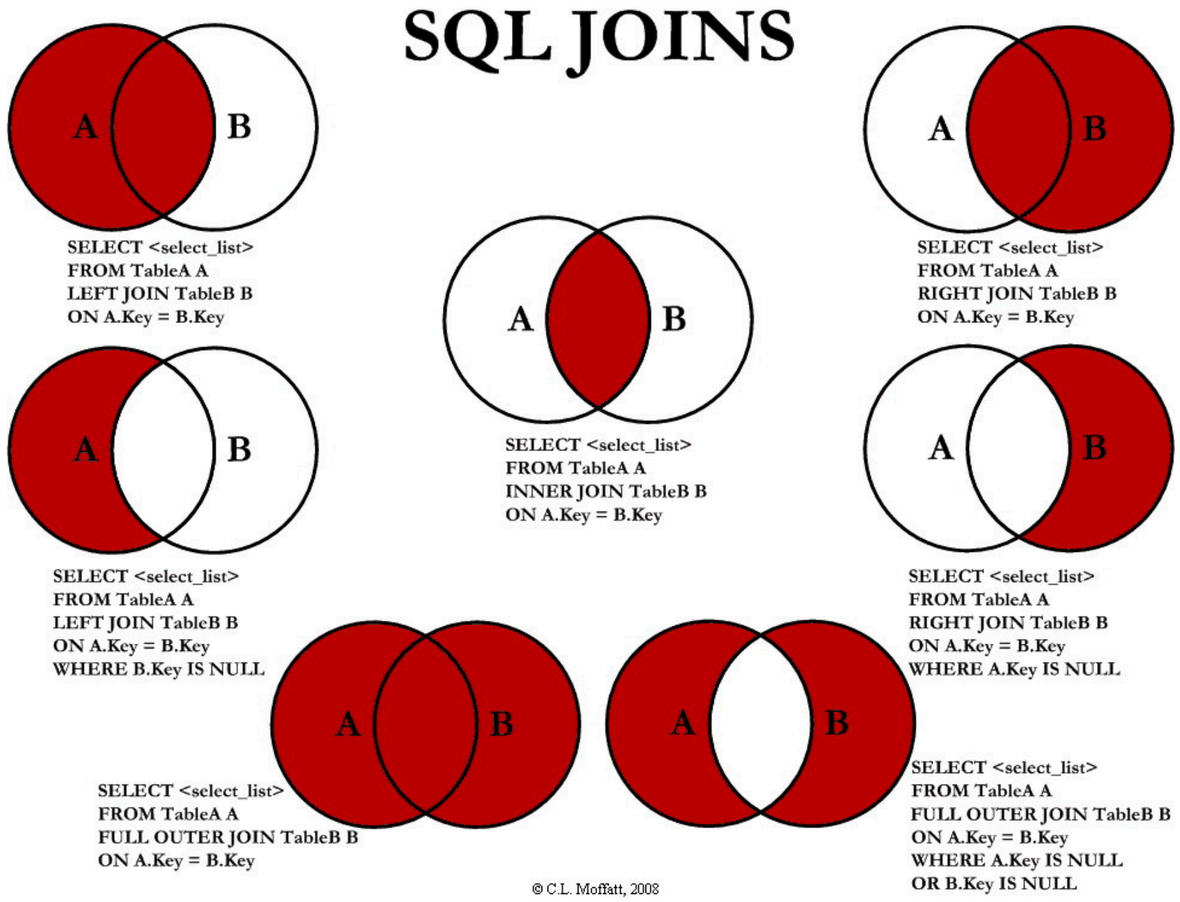
- Lister le nombre de films par époque par ordre décroissant

8

# Mettre en relation

# 8 Mettre en relation les données

- **JOIN** : combiner les lignes provenant de deux ou plusieurs tables avec une condition de correspondance
- Types de jointures : **LEFT**, **RIGHT**, **FULL**, **INNER** etc.



# EXEMPLE

- Récréer la table **boxoffice**, avec contraintes et données respectives
- Lister les films ainsi que leur entrées internationales respectives
- Lister le film qui a connu le plus d’entrées domestiques

# Bravo ! ☑

1. Créer une base de données
2. Lire les données
3. Mettre à jour les données
4. Supprimer des données
5. Filtrer les données
6. Agréger les données
7. Trier les données
8. Mettre en relation les données

# Projet

## Nouveaux besoins

- **Séance** avec *Chef·fes de projet*

## Diagramme UML

- **Intégrer** nouveaux besoins dans la logique de la base de données
- **Adapter** le schéma UML avec les bons types ainsi que les contraintes que vous jugez pertinentes
- **Créer** la base de données