FORMATION DEV WEB WEB MOBILE
CCP 2 - DÉVELOPPER LA PARTIE BACK-END D'UNE APPLICATION WEB

LES BASES DE DONNEES RELATIONNELLES

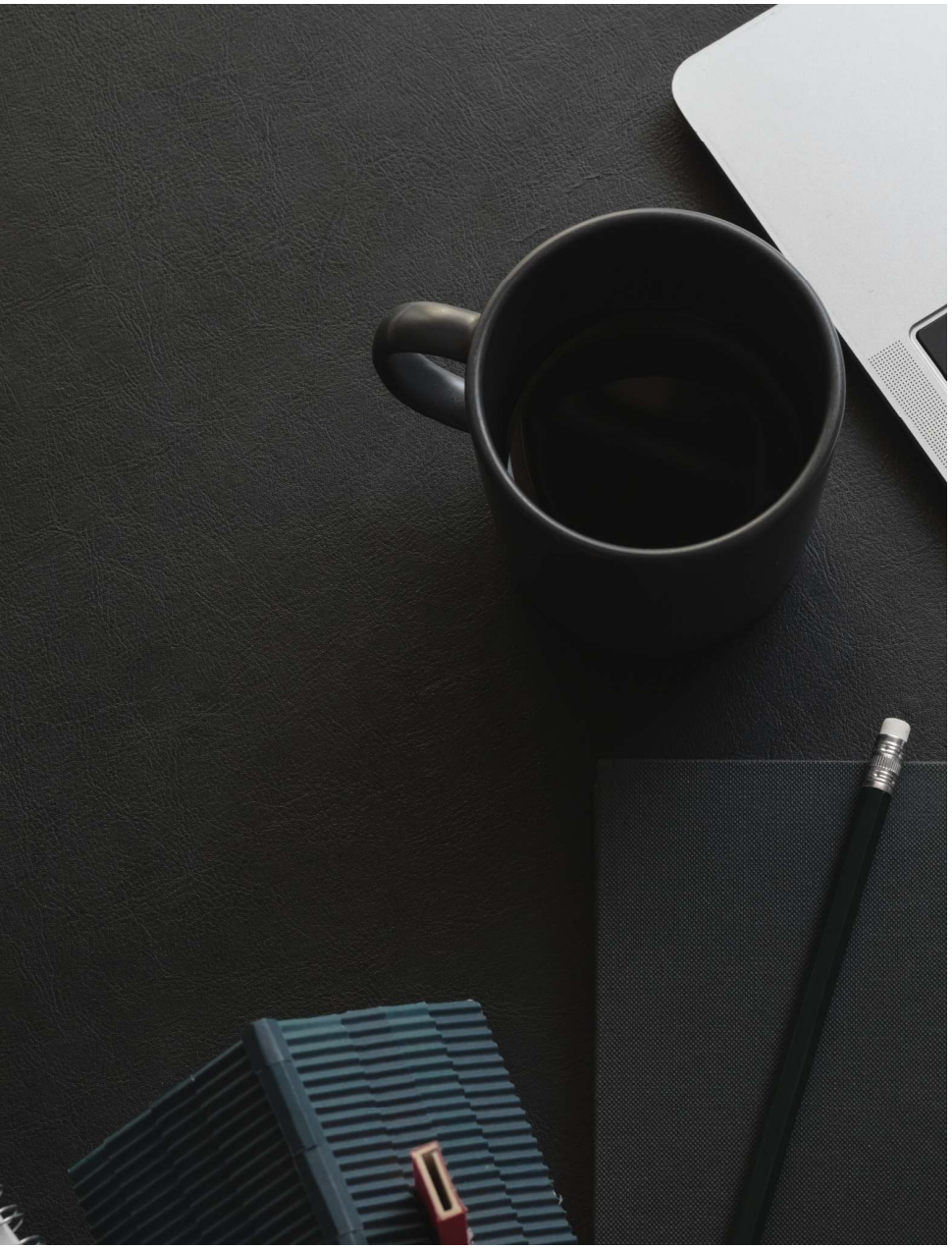
JOUR 4 : JEU D'ESSAI, INSERTION ET AGREGATION, REQUÊTES AVANCEES

PRESENTE PAR NATACHA DESSE

AFEC

DEROULE MODULE

- Découverte des BDD et Installation
- Les relations et modélisation (DEA)
- Contraintes et schéma physique
- **Jeu d’essai, insertion et agrégations, requêtes avancées**
- Jointures et sous requêtes
- Sauvegardes SQL et gestion des rôles



JEU D'ESSAI, INSERTION ET AGREGATION

LES OBJECTIFS

Créer un jeu de données réaliste

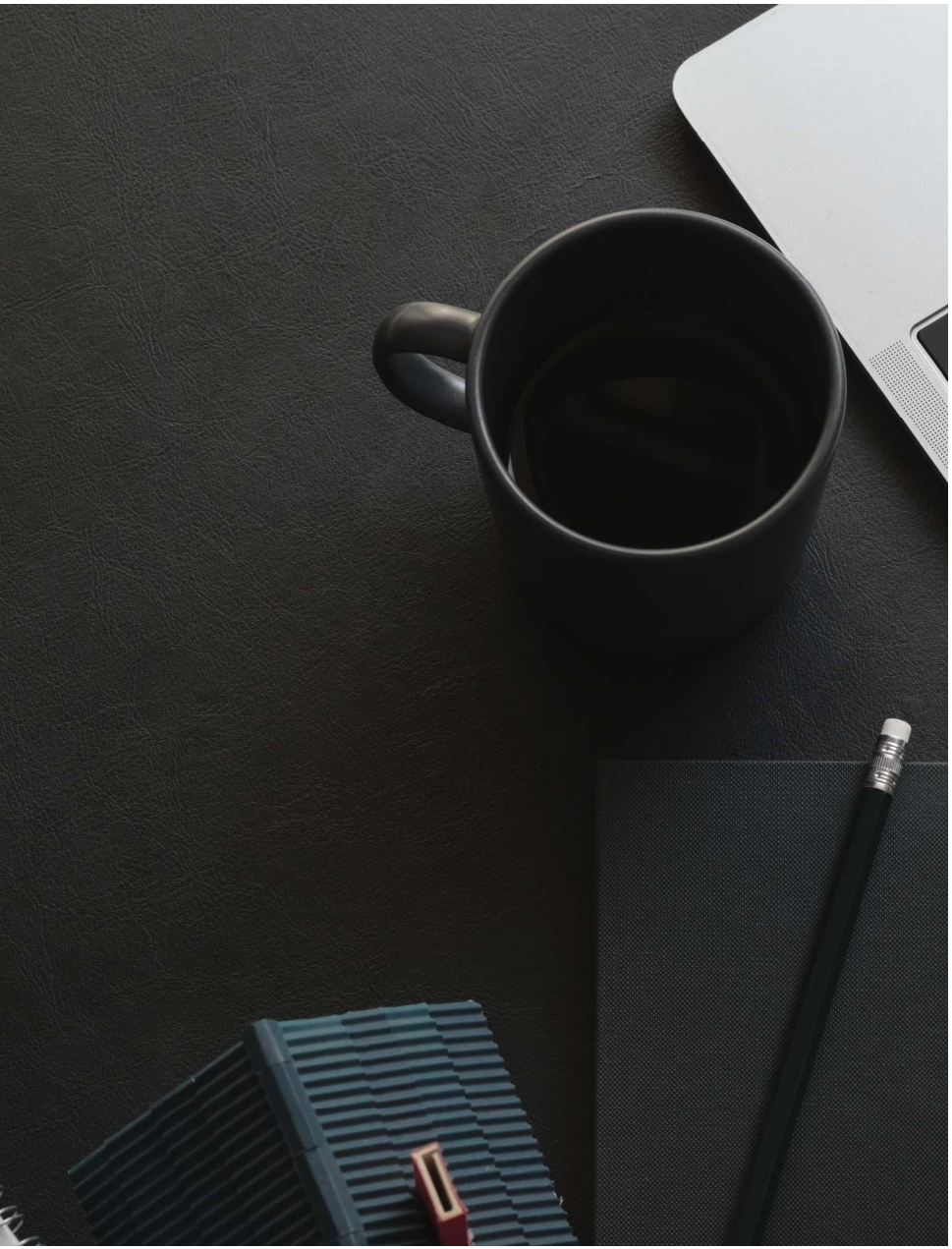
Insérer massivement des données dans une base SQL

Maîtriser les fonctions d'agrégation (COUNT, SUM ...)

Optimiser ses requêtes d'agrégation

NUAGE DE MOTS

Citez des contraintes que vous connaissez



LES FONCTIONS D'AGREGATIONS

POURQUOI FAIRE ?

Définition :

Les **fonctions** d'**agrégation** permettent de **résumer** ou **calculer** une valeur unique à partir de **plusieurs lignes**.

Utilisation typique :

regrouper les données (GROUP BY)
Dans des **statistiques** simples (somme, moyenne, nombre d'éléments...)

Fonction	Description
COUNT(*)	Compte **toutes les lignes**
COUNT(col)	Compte les lignes où la **colonne n'est pas NULL**
SUM(col)	Calcule la **somme** des valeurs numériques
AVG(col)	Calcule la **moyenne**
MIN(col)	Renvoie la **valeur minimale**
MAX(col)	Renvoie la **valeur maximale**

COUNT()

Objectif : Compter le nombre de lignes

```
SELECT COUNT(*) FROM Employes;
SELECT COUNT(Dep) FROM Employes;
SELECT COUNT(DISTINCT Age) FROM Employes;
```

La fonction COUNT est utilisée pour compter le nombre de lignes dans une table de base de données. Il peut fonctionner sur les types de données numériques et non numériques.

- COUNT(*) compte toutes les lignes (y compris celles avec des valeurs NULL)
- COUNT(colonne) ne compte que les lignes non nulles
- COUNT (DISTINCT colonne) compte les valeurs distinctes

Table - Employes

Id	Nom	Age	Salaire	Profession	Dep
1	Ismail	25	6000.00	Assistant	2
2	Mohamed	30	8000.40	Directeur	1
3	Fatima	29	6000.00	Directeur	3
4	Dounia	30	7000.00	Assistant	4
5	Omar	29	9000.00	Ingenieur	1
7	Mostafa	29	7500.00	Ingenieur	NULL

SELECT count(*) FROM Employes;

count(*)
6

SELECT count(Dep) FROM Employes;

count(Dep)
5

SELECT COUNT(DISTINCT Age) FROM Employes;

count(DISTINCT Age)
3

SUM()

Objectif : Faire la somme d'une colonne numérique

```
SELECT SUM(Salaire) FROM Employes;
SELECT SUM(DISTINCT Age) FROM Employes;
```

Cas d'usage : Calcul du chiffre d'affaires, total des quantités vendues, etc.

Table - Employes

Id	Nom	Age	Salaire	Profession	Dep
1	Ismail	25	6000.00	Assistant	2
2	Mohamed	30	8000.40	Directeur	1
3	Fatima	29	6000.00	Directeur	3
4	Dounia	30	7000.00	Assistant	4
5	Omar	29	9000.00	Ingenieur	1
7	Mostafa	29	7500.00	Ingenieur	NULL

SELECT SUM(Salaire) FROM Employes;

SUM(Salaire)
43500.40

SELECT SUM(DISTINCT Age) FROM Employes;

SUM(DISTINCT Age)
84

AVG()

Objectif : Calculer la moyenne d'une colonne

```
SELECT AVG(Salaire) FROM Employes;
```

Remarques :

- Ignore les NULL
- Résultat souvent en décimal

Table - Employes

Id	Nom	Age	Salaire	Profession	Dep
1	Ismail	25	6000.00	Assistant	2
2	Mohamed	30	8000.40	Directeur	1
3	Fatima	29	6000.00	Directeur	3
4	Dounia	30	7000.00	Assistant	4
5	Omar	29	9000.00	Ingenieur	1
7	Mostafa	29	7500.00	Ingenieur	NULL

SELECT AVG(Salaire)
FROM Employes;

AVG(Salaire)
7250.066667

MIN() ET MAX()

Objectif : Obtenir la valeur minimale ou maximale

```
SELECT MIN(Salaire) FROM Employes;
SELECT MAX(Salaire) FROM Employes;
```

Cas d'usage :

- Date la plus récente / plus ancienne
- Valeur extrême dans une série

Table - Employes

Id	Nom	Age	Salaire	Profession	Dep
1	Ismail	25	6000.00	Assistant	2
2	Mohamed	30	8000.40	Directeur	1
3	Fatima	29	6000.00	Directeur	3
4	Dounia	30	7000.00	Assistant	4
5	Omar	29	9000.00	Ingenieur	1
7	Mostafa	29	7500.00	Ingenieur	NULL

SELECT MIN(Salaire)
FROM Employes;

MIN(Salaire)
6000.00

SELECT MAX(Salaire)
FROM Employes;

MAX(Salaire)
9000.00

ATELIER

Agrégations

CONSIGNES

Importer les données à l'aide du fichier 04-exercice-agregation-SQL.sql

Astuce pour éxécuter le fichier (attention pas de antislach ni de 'single' or "double quote")

```
mysql > source chemin/vers/fichier.sql;
```

CONSIGNES

1. **Combien** d'utilisateurs sont enregistrés ?
2. **Combien** d'emails **valides** (non NULL) **uniques** sont présents ?
3. Quel est l'âge **moyen** des utilisateurs ?
4. Quel est l'âge **minimum** ? Et **maximum** ?
5. **Combien** de commandes ont été passées ?
6. Quel est le **montant total** de toutes les commandes ?
7. Quel est le **montant moyen** des commandes ?
8. **Combien** de commandes ont un montant spécifié (**non NULL**) ?
9. Quelle est la date de commande la plus **récente** ?
10. Quel est le nombre total de clients ayant passé au moins une commande ?

MEMO

Fonction	Description
COUNT(*)	Compte **toutes les lignes**
COUNT(col)	Compte les lignes où la **colonne n'est pas NULL**
SUM(col)	Calcule la **somme** des valeurs numériques
AVG(col)	Calcule la **moyenne**
MIN(col)	Renvoie la **valeur minimale**
MAX(col)	Renvoie la **valeur maximale**

REPONSES

1. Combien d'utilisateurs sont enregistrés ?

COUNT(*)

2. Combien d'emails valides (non NULL) et uniques sont présents ?

COUNT(DISTINCT email)

3. Quel est l'âge moyen des utilisateurs ?

AVG(age)

4. Quel est l'âge minimum ? Et maximum ?

MIN(age), MAX(age)

5. Combien de commandes ont été passées ?

COUNT(*) sur la table commande

6. Quel est le montant total de toutes les commandes ?

SUM(montant)

7. Quel est le montant moyen des commandes ?

AVG(montant)

8. Combien de commandes ont un montant spécifié (non NULL) ?

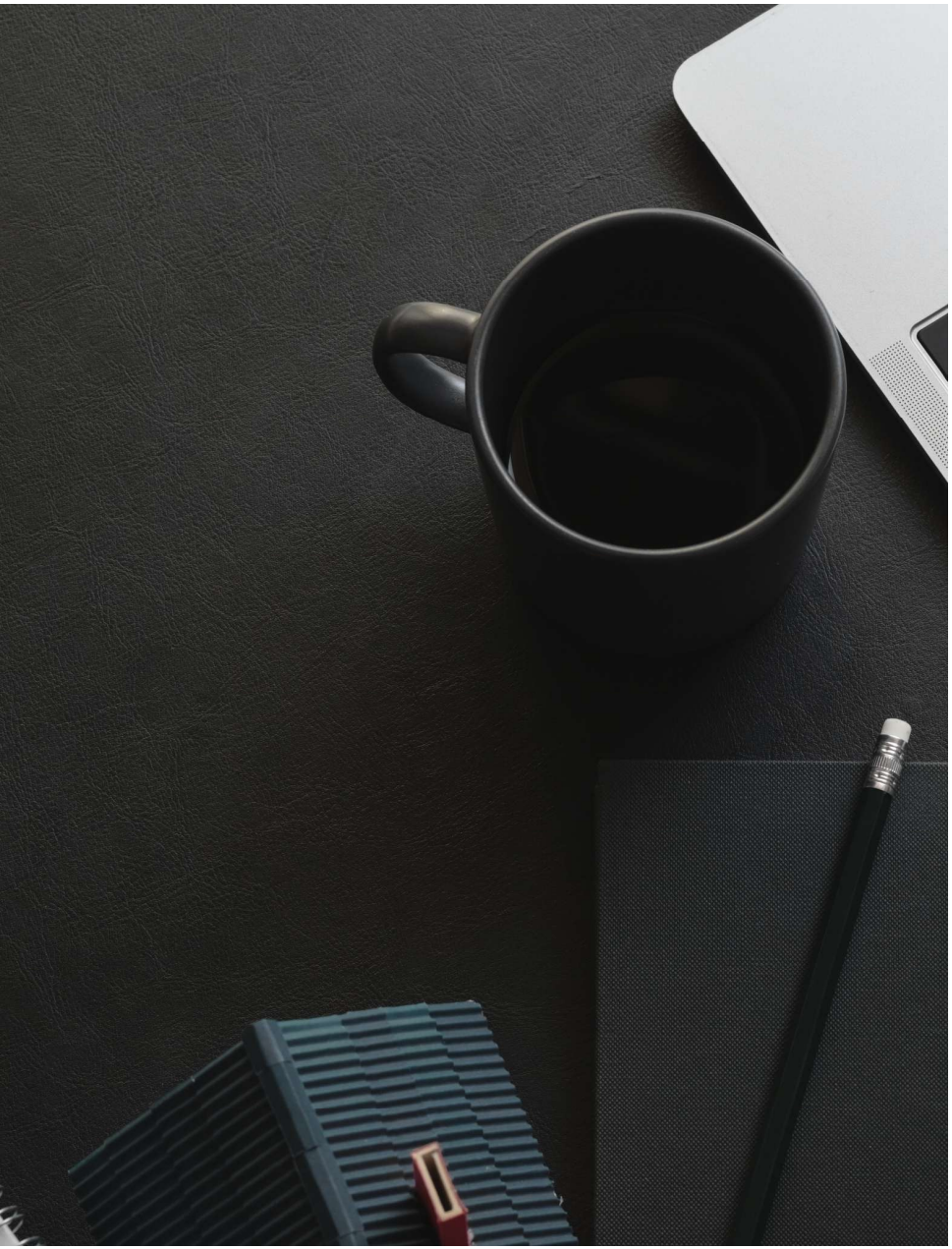
COUNT(montant)

9. Quelle est la date de commande la plus récente ?

MAX(date_commande)

10. Quel est le nombre total de clients ayant passé au moins une commande ?

SELECT COUNT(DISTINCT utilisateur_id) FROM commande;



AGREGATIONS AVANCEES

GROUP BY

Objectif : permet de regrouper des lignes ayant des valeurs communes sur une ou plusieurs colonnes pour appliquer des fonctions d'agrégation (COUNT, SUM, AVG, etc.).

```
SELECT colonne,
FONCTION_AGREGAT(colonne)
FROM table
GROUP BY colonne;
```

Table - Employes

Id	Nom	Age	Salaire	Profession	Dep
1	Ismail	25	6000.00	Assistant	2
2	Mohamed	30	8000.40	Directeur	1
3	Fatima	29	6000.00	Directeur	3
4	Dounia	30	7000.00	Assistant	4
5	Omar	29	9000.00	Ingenieur	1
7	Mostafa	29	7500.00	Ingenieur	NULL

SELECT Age
FROM Employes;

Age
25
30
29
30
30
29

SELECT Age
FROM Employes
GROUP BY Age;

Age
25
29
30

SELECT Age, count(*) AS "Nombre d'employés"
FROM Employes
GROUP BY Age;

Age	Nombre d'employés
25	1
29	2
30	3

GROUP BY & AVG

Compter le salaire moyen sur chaque groupe d'âge

```
SELECT Age, AVG(Salaire) AS "Salaire moyen"
FROM Employes
GROUP BY Age;
```

Table - Employes

Id	Nom	Age	Salaire	Profession	Dep
1	Ismail	25	6000.00	Assistant	2
2	Mohamed	30	8000.40	Directeur	1
3	Fatima	29	6000.00	Directeur	3
4	Dounia	30	7000.00	Assistant	4
5	Omar	29	9000.00	Ingenieur	1
7	Mostafa	29	7500.00	Ingenieur	NULL

SELECT Age, AVG(Salaire) AS "Salaire moyen"
FROM Employes GROUP BY Age;

Age	Salaire moyen
25	6000.000000
29	6750.000000
30	8000.133333

HAVING

Nous savons que la clause WHERE est utilisée pour imposer des conditions aux colonnes, mais que se passe-t-il si nous voulons imposer des conditions aux groupes?

C'est ici que la clause HAVING entre en vigueur.

```
SELECT Age, AVG(Salaire) AS salaire_moyen
FROM Employes
GROUP BY Age
HAVING AVG(Salaire) > 6000;
```

Nous ne pouvons pas utiliser les fonctions d'agrégation telles que SUM(), COUNT(), etc. avec la clause WHERE. Nous devons donc utiliser la clause HAVING si nous voulons utiliser l'une de ces fonctions dans les conditions.

Table - Employes

Id	Nom	Age	Salaire	Profession	Dep
1	Ismail	25	6000.00	Assistant	2
2	Mohamed	30	8000.40	Directeur	1
3	Fatima	29	6000.00	Directeur	3
4	Dounia	30	7000.00	Assistant	4
5	Omar	29	9000.00	Ingenieur	1
7	Mostafa	29	7500.00	Ingenieur	NULL

afficher l'âge des groupes dont le salaire moyen est > 6000 :

age	salaire_m
29	6750.00
30	8000.13

ATELIER

Agrégations la suite

ENONCE

Trouvez les commandes SQL pour savoir :

1. Combien de commandes ont été passées par utilisateur ?
2. Quels utilisateurs ont passé plus d'une commande ?

REPONSE

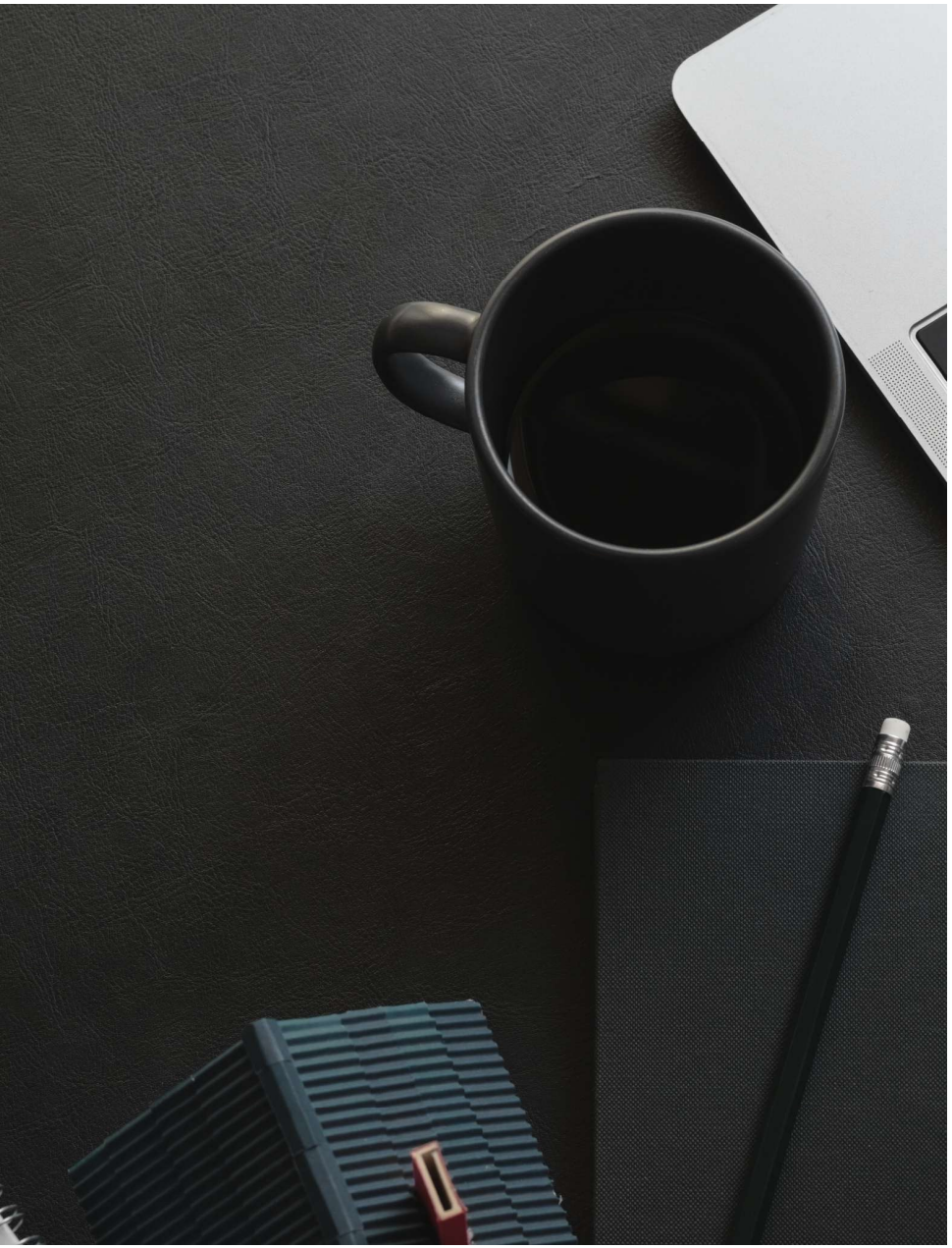
Trouvez les commandes SQL pour savoir :

1. Combien de commandes ont été passées par utilisateur ?

```sql
SELECT utilisateur_id, COUNT(*) AS nb_commandes
FROM commande
GROUP BY utilisateur_id;
```

2.Quels utilisateurs ont passé plus d'une commande ?

```sql
SELECT utilisateur_id, COUNT(*) AS nb_commandes
FROM commande
GROUP BY utilisateur_id
HAVING COUNT(*) > 1;
```



JEU DE DONNEES ET INSERTION

INSERTION DE DONNES CLASSIQUE

Insertion avec INSERT

```
INSERT INTO client (nom, email, ville)
VALUES ('Dupont', 'dupont@mail.com', 'Lyon');
```

Insertion en bloc

```
INSERT INTO livres (titre, auteur, date_publication)
VALUES
 ('L\'Étranger', 'Albert Camus', '1942-05-19'),
 ('1984', 'George Orwell', '1949-06-08'),
 ('Le Petit Prince', 'Antoine de Saint-Exupéry', '1943-04-06');
```

INSERTION AVEC FICHIER

Insertion avec LOAD DATA INFILE

```
LOAD DATA INFILE '/path/clients.csv'
INTO TABLE Client
FIELDS TERMINATED BY ',' ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

Pour créer des jeux de données fictifs

Utilisez des générateurs comme Mockaroo ou Faker (package)
Préparer des fichiers .csv cohérents avec les contraintes

QUE SE PASSE-T-IL SI...

Une contrainte n’est pas respectée ?

Si tu fais un seul INSERT ou UPDATE, il échoue entièrement.

Si tu fais un batch d'instructions (plusieurs requêtes) :

- SQL exécute chaque requête indépendamment.
- Les requêtes valides passent, celles qui violent une contrainte sont ignorées avec erreur

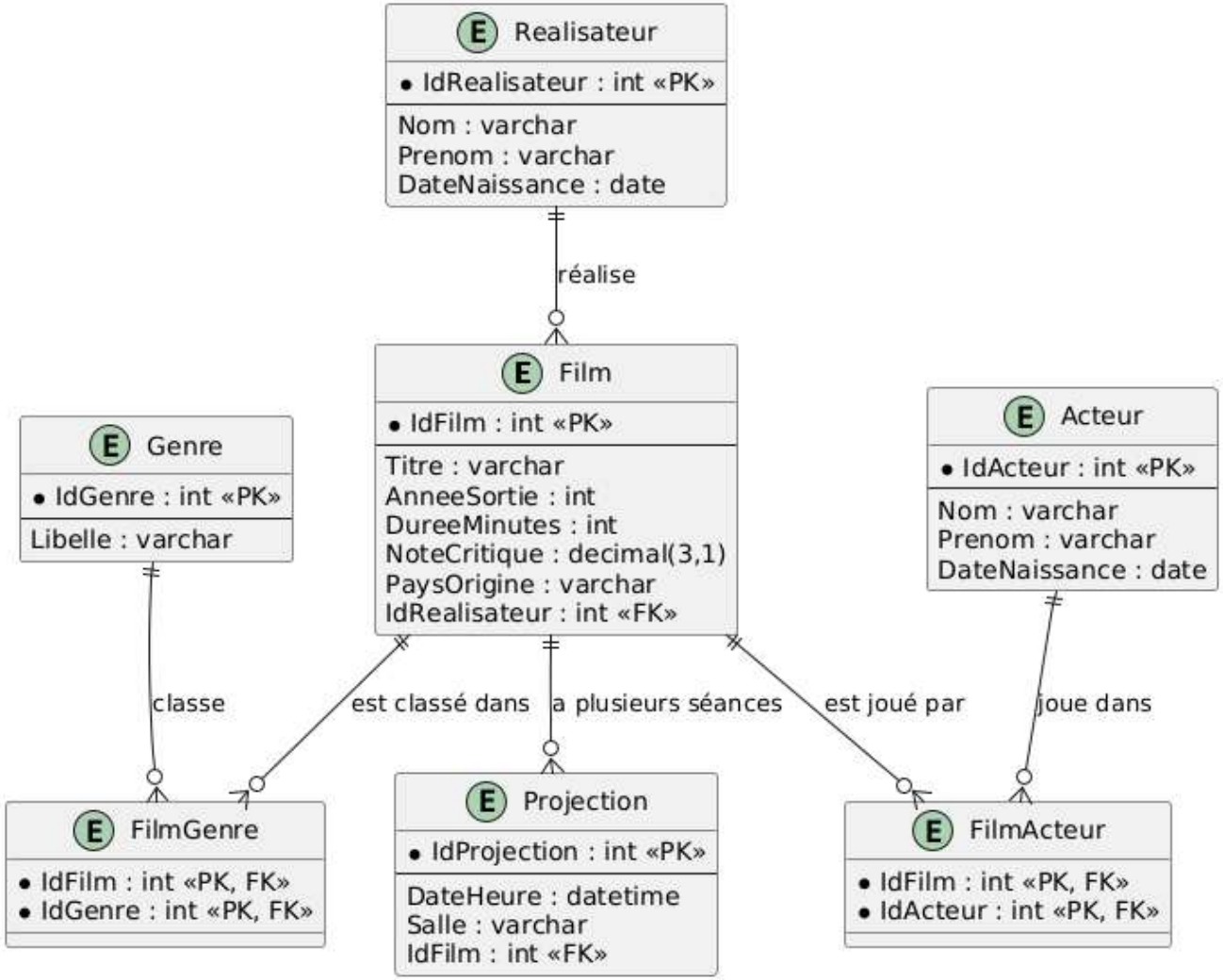
ATELIER

Jeu de données dans CineClub

CONSIGNES

Nous allons désormais insérer un jeu de données fictif dans notre base de données pour simuler son activité

RAPPEL


Realisateur
• IdRealisateur : int «PK»
Nom : varchar
Prenom : varchar
DateNaissance : date
réalise
Film
• IdFilm : int «PK»
Titre : varchar
AnneeSortie : int
DureeMinutes : int
NoteCritique : decimal(3,1)
PaysOrigine : varchar
IdRealisateur : int «FK»
Genre
• IdGenre : int «PK»
Libelle : varchar
Acteur
• IdActeur : int «PK»
Nom : varchar
Prenom : varchar
DateNaissance : date
classe
est classé dans
a plusieurs séances
est joué par
joue dans
FilmGenre
• IdFilm : int «PK, FK»
• IdGenre : int «PK, FK»
Projection
• IdProjection : int «PK»
DateHeure : datetime
Salle : varchar
IdFilm : int «FK»
FilmActeur
• IdFilm : int «PK, FK»
• IdActeur : int «PK, FK»

QUIZZ

QUESTION 1

Quelle fonction SQL permet de compter le nombre de lignes dans une table ?

A. SUM(*)
B. COUNT(*)
C. TOTAL(*)
D. LENGTH(*)

QUESTION 2

2. Quelle est la différence entre WHERE et HAVING ?

A. WHERE filtre les lignes après le GROUP BY, HAVING avant
B. HAVING ne fonctionne qu’avec JOIN
C. WHERE filtre avant agrégation, HAVING après agrégation
D. Il n'y a pas de différence

QUESTION 3

3. Quelle commande permet d’insérer massivement un fichier CSV dans une table SQL ?

A. BULK INSERT
B. IMPORT FILE
C. LOAD DATA INFILE
D. COPY FILE

QUESTION 4

Que fait l'instruction GROUP BY dans une requête ?

A. Elle trie les lignes par ordre croissant
B. Elle divise les lignes en groupes pour l'agrégation
C. Elle compte le nombre de groupes
D. Elle masque les doublons

QUESTION 5

Quelle requête retourne la moyenne des salaires de la table employes ?

A. SELECT MOYENNE(salaire) FROM employes;
B. SELECT MEAN(salaire) FROM employes;
C. SELECT AVG(salaire) FROM employes;
D. SELECT SUM(salaire) FROM employes;

QUESTION 6

Quelle function retourne la plus petite valeur dans une colonne ?

A. SMALLEST()
B. MIN()
C. LEAST()
D. LOWER()

QUESTION 7

Que se passe-t-il si on oublie la clause VALUES dans une requête INSERT ?

A. Une ligne vide est insérée
B. La requête s'exécute sans effet
C. Une erreur de syntaxe est levée
D. SQL insère automatiquement des valeurs nulles

QUESTION 8

Que fait la requête suivante ?

```sql
SELECT departement, COUNT(*)
FROM employes
GROUP BY departement;
```

A. Compte le nombre total d'employés
B. Affiche tous les départements
C. Affiche le nombre d'employés par département
D. Ne fonctionne pas car il manque une clause WHERE

QUESTION 9

Quelle est la différence entre COUNT(*) et COUNT(colonne) ?

A. COUNT(*) ignore les lignes avec des NULL
B. COUNT(colonne) compte toutes les lignes même avec NULL
C. COUNT(colonne) ignore les lignes où la colonne est NULL
D. Il n'y a pas de différence

QUESTION 10

Que fais la requête suivante :

```sql
SELECT AVG(age)
FROM utilisateurs
WHERE ville = 'Paris';
```

A. Calcule l'âge moyen de tous les utilisateurs
B. Affiche les utilisateurs par ville
C. Calcule l'âge moyen des utilisateurs vivant à Paris
D. Calcule la somme des âges

CORRECTION

1. B. COUNT(*)
2. C. WHERE filtre avant agrégation, HAVING après agrégation
3. C. LOAD DATA INFILE
4. B. Elle divise les lignes en groupes pour l’agrégation
5. C
6. B
7. C
8. C
9. C
10. C