Sorbonne Data Analytics
- SDA -

Mini Projet SQL

# Exploration de base de donnée
# Étude de cas “Entreprise”
# questions d’entretien

**Auteurs :**
N’GUESSAN Yao Eben-Ezer

**Enseignante :**
Fatimetou ZEINE

**Année Universitaire 2025–2026**

# MINI PROJET SQL

## PARTIE A — Exploration de la base `Sakila`

Nous allons utiliser la base de données `sakila`.

### Préambule

```
USE sakila;
```

### 1. Films de durée > 120 minutes, triés par durée décroissante

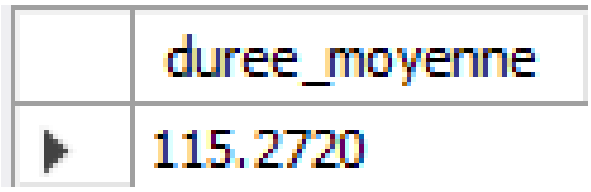
```
SELECT f.title, length
FROM film f
WHERE length > 120
ORDER BY length DESC;
```

La plus grande durée de film est de 185 min.

### 2. Durée moyenne et films au-dessus de la moyenne

#### (a) Durée moyenne des films

```
SELECT AVG(f.length) AS duree_moyenne
FROM film f;
```




La durée moyenne des films est : 115.2720

#### (b) Films dont la durée dépasse la moyenne

```
SELECT f.title, f.length
FROM film f
WHERE f.length > (
  SELECT AVG(f.length)
  FROM film f
);
```

## 3. Pour chaque film : titre, catégorie et langue

```
SELECT f.title, c.name AS category, l.name AS film_language
FROM film f
JOIN language l
  ON f.language_id = l.language_id
JOIN film_category fc
  ON f.film_id = fc.film_id
JOIN category c
  ON fc.category_id = c.category_id;
```

La langue utilisée est : anglais, quel que soit la catégorie du film.

## 4. Les 5 films les plus loués (titre et nombre total de locations)

```
SELECT f.title, COUNT(r.rental_id) AS Total_locations
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r   ON i.inventory_id = r.inventory_id
GROUP BY f.title
ORDER BY Total_locations DESC
LIMIT 5;
```

| title | Total_locations |
|---|---|
| BUCKET BROTHERHOOD | 34 |
| ROCKETEER MOTHER | 33 |
| FORWARD TEMPLE | 32 |
| GRIT CLOCKWORK | 32 |
| JUGGLER HARDLY | 32 |

Les 5 films les plus loués sont : `BUCKET BROTHERHOOD`, `ROCKETEER MOTHER`, `FORWARD TEMPLE`, `GRIT CLOCKWORK`, `JUGGLER HARDLY`.

## 5. Revenu total par film et top 5 des films les plus rentables

```
SELECT f.title, SUM(p.amount) AS Revenu_total
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r    ON i.inventory_id = r.inventory_id
JOIN payment p   ON r.rental_id = p.rental_id
GROUP BY f.title
ORDER BY Revenu_total DESC
LIMIT 5;
```

| title | Revenu_total |
|---|---|
| TELEGRAPH VOYAGE | 231.73 |
| WIFE TURN | 223.69 |
| ZORRO ARK | 214.69 |
| GOODFELLAS SALUTE | 209.69 |
| SATURDAY LAMBS | 204.72 |

## 6. Vue `vip_clients` (clients > 100 dollars dépensés)

```
CREATE VIEW Vip_clients AS
SELECT c.customer_id, c.first_name, c.last_name,
       SUM(p.amount) AS depense_total
FROM customer c
JOIN payment p ON c.customer_id = p.customer_id
GROUP BY c.customer_id, c.first_name, c.last_name
HAVING depense_total > 100;
```

## 7. Insertion puis suppression d'un acteur en respectant l'intégrité référentielle

Insertion :

```
INSERT INTO actor (first_name, last_name)
VALUES ('Didier', 'Drogba');
```

Récupération de l'`actor_id` :

```
SELECT actor_id FROM actor
WHERE first_name = 'Didier' AND last_name = 'Drogba';
-- Suppose : actor_id = 201
```

Suppression des références liées puis de l'acteur :

```
DELETE FROM film_actor
WHERE actor_id = 201;

DELETE FROM actor
WHERE actor_id = 201;
```

Vérification :

```
SELECT a.first_name, a.last_name
FROM actor a
WHERE a.last_name LIKE 'Dro%';
```

Notre acteur a bien été supprimé.

## 8. Clients dont le montant total payé est > moyenne de tous les clients

```
SELECT
  c.customer_id,
  c.first_name,
  c.last_name,
  SUM(p.amount) AS montant_total_paye
FROM customer c
INNER JOIN payment p
  ON c.customer_id = p.customer_id
```

```
GROUP BY c.customer_id, c.first_name, c.last_name
HAVING SUM(p.amount) > (
  SELECT AVG(total)
  FROM (
    SELECT SUM(amount) AS total
    FROM payment
    GROUP BY customer_id
  ) AS t
);
```

# PARTIE B — Étude de cas “Entreprise” et questions d’entretien

## Section 1 — Création de la base, tables et insertion

La table a été créée en utilisant les commandes directement, ce qui a permis de définir les clés primaires et les contraintes (PK, NN, ...).

Utilisation de la base de données de l’entreprise `entreprise_nye` :

```
USE entreprise_nye;
```

## Section 2 — Analyse des données

### 3. Top 3 des commandes par revenu, avec client et détail des produits

```
SELECT
  o.order_id,
  c.first_name, c.last_name,
  t.order_total,
  p.product_name,
  oi.quantity,
  oi.price,
  (oi.quantity * oi.price) AS line_amount
FROM (
  SELECT
    oi.order_id,
    SUM(oi.quantity * oi.price) AS order_total
  FROM OrderItems oi
  GROUP BY oi.order_id
  ORDER BY order_total DESC
  LIMIT 3
) AS t
JOIN Orders o      ON o.order_id = t.order_id
JOIN Customers c   ON c.customer_id = o.customer_id
JOIN OrderItems oi ON oi.order_id = o.order_id
JOIN Products p    ON p.product_id = oi.product_id
ORDER BY t.order_total DESC, o.order_id, p.product_name;
```

| order_id | first_name | last_name | order_total | product_name | quantity | price | line_amount |
|---|---|---|---|---|---|---|---|
| 6 | Yao | Nguessan | 1598.00 | Smartphone 128GB | 2 | 799.00 | 1598.00 |
| 1 | Alice | Kouadio | 999.99 | Laptop 14" | 1 | 999.99 | 999.99 |
| 4 | Paul | Koffi | 597.00 | 27" Monitor | 3 | 199.00 | 597.00 |

### 4. Clients ayant dépensé > 500 dans un même mois (mois, client, montant)

```
SELECT
  MONTH(o.order_date) AS mois,
  c.first_name,
  c.last_name,
  SUM(o.total) AS montant_total
FROM Orders o
JOIN Customers c ON c.customer_id = o.customer_id
GROUP BY c.customer_id, mois, c.first_name, c.last_name
HAVING montant_total > 500
ORDER BY mois, montant_total DESC;
```

| mois | first_name | last_name | montant_total |
|---|---|---|---|
| 10 | Yao | Nguessan | 1598.00 |
| 10 | Alice | Kouadio | 999.99 |
| 10 | Paul | Koffi | 597.00 |

Le mois où des clients ont dépensé plus de 500 est le mois d'octobre.

### 5. Revenu total et quantité vendue par produit et catégorie

Afficher les 3 produits les plus rentables et le classement des catégories par revenu décroissant.

```
SELECT
  tp.product_id,
  tp.product_name,
  tp.category,
  tp.total_quantity,
  tp.total_revenue,
  tp.rank_product,
  cr.category_revenue,
  cr.rank_category
FROM (
  SELECT
    ps.product_id,
    ps.product_name,
    ps.category,
    ps.total_quantity,
    ps.total_revenue,
    RANK() OVER (ORDER BY ps.total_revenue DESC) AS rank_product
  FROM (
    SELECT
      p.product_id,
      p.product_name,
      p.category,
```

```
      SUM(oi.quantity) AS total_quantity,
      SUM(oi.quantity * oi.price) AS total_revenue
    FROM Products p
    JOIN OrderItems oi ON oi.product_id = p.product_id
    GROUP BY p.product_id, p.product_name, p.category
  ) AS ps
) AS tp
JOIN (
  SELECT
    c.category,
    c.category_revenue,
    RANK() OVER (ORDER BY c.category_revenue DESC) AS rank_category
  FROM (
    SELECT
      p.category,
      SUM(oi.quantity * oi.price) AS category_revenue
    FROM Products p
    JOIN OrderItems oi ON oi.product_id = p.product_id
    GROUP BY p.category
  ) AS c
) AS cr
  ON tp.category = cr.category
WHERE tp.rank_product <= 3
ORDER BY tp.total_revenue DESC;
```

| product_id | product_name | category | total_quantity | total_revenue | rank_product | category_revenue | rank_category |
|---|---|---|---|---|---|---|---|
| 6 | Smartphone 128GB | Electronics | 2 | 1598.00 | 1 | 3543.99 | 1 |
| 1 | Laptop 14" | Electronics | 1 | 999.99 | 2 | 3543.99 | 1 |
| 4 | 27" Monitor | Electronics | 3 | 597.00 | 3 | 3543.99 | 1 |

### 6. Vue HighValueOrders ($>$ 400) et index suggérés

```
CREATE VIEW HighValueOrders AS
SELECT
  ot.order_id,
  c.first_name,
  c.last_name,
  ot.total_amount,
  ot.total_items
FROM (
  SELECT
    o.order_id,
    o.customer_id,
    SUM(oi.quantity * oi.price) AS total_amount,
    SUM(oi.quantity) AS total_items
  FROM Orders o
  JOIN OrderItems oi ON oi.order_id = o.order_id
  GROUP BY o.order_id, o.customer_id
) AS ot
JOIN Customers c ON c.customer_id = ot.customer_id
WHERE ot.total_amount > 400;
```

| order_id | first_name | last_name | total_amount | total_items |
|---|---|---|---|---|
| 1 | Alice | Kouadio | 999.99 | 1 |
| 4 | Paul | Koffi | 597.00 | 3 |
| 6 | Yao | Nguessan | 1598.00 | 2 |

**Index proposés sur la table Orders et leur impact sur les performances**

— CREATE INDEX idx_orders_total ON Orders(total) ; Cet index accélère la recherche des commandes dont le montant total dépasse 400 (vue HighValueOrders). Sans index, la base doit parcourir toute la table pour vérifier la condition total > 400.

**Impact** : réduction du temps de réponse.

— CREATE INDEX idx_orders_customer_orderdate ON Orders(customer_id, order_date) ;

— Cet index optimise les requêtes d'analyse par client et par période (WHERE customer_id = .... AND order_date BETWEEN ...). L'ordre des colonnes (customer_id, puis order_date) permet d'effectuer un filtrage rapide par client, puis un parcours séquentiel sur la plage de dates.

**Impact** : amélioration des requêtes mensuelles ou trimestrielles, exécutions plus rapides et temps d'exécution plus efficace.

# Section 3 — Questions de type "entretien technique SQL"

**7. Quelle est la différence entre les clauses WHERE et HAVING ?**

WHERE et HAVING permettent de filtrer les résultats, mais il y a une différence : WHERE filtre les lignes selon une condition, tandis que HAVING filtre les lignes sous une condition après une fonction d'agrégation (c'est-à-dire après l'utilisation de la fonction `GROUP BY`).

**8. Expliquez la normalisation d'une base de données et son utilité.**

La normalisation d'une base de données est un processus d'organisation des tables et des relations. Elle permet d'éviter les redondances. La définition des clés primaires et secondaires dans la normalisation permet de les éviter.

**9. Citez deux techniques pour éviter les doublons dans une requête SQL.**

Pour éviter les doublons, on peut utiliser la fonction `GROUP BY` qui regroupe les lignes identiques et permet d'éviter les doublons. On peut aussi utiliser `DISTINCT`, qui élimine les doublons dans le résultat en ne renvoyant qu'une seule fois les lignes identiques.

**10. Quelle est la différence entre les commandes DELETE, TRUNCATE et DROP ?**

Elles ont toutes des fonctions de suppression, mais :

— `DELETE` permet de supprimer des lignes sous une condition ;

— `TRUNCATE` permet de supprimer toutes les lignes d'une table sans condition ;

— `DROP` supprime toute la table ou la base de données (après exécution, la table n'existe plus).

**11. Donnez une méthode pour optimiser une requête SQL lente.**
Pour optimiser une requête SQL lente , on peut éviter d'utiliser SELECT * pour récupérer toutes les colonnes, même celles qui ne sont pas pertinentes pour notre analyse. On peut donc optimiser notre requête en sélectionnant que les colonnes spécifiques dont nous avons besoin, ce qui nous permettra d'optimiser les performances de notre requête.

A titre illustratif :

— **Au lieu d'écrire** :
— SELECT *
— FROM products ;

**Nous devrions écrire** :
— SELECT product_id, product_name, unit_price
— FROM products

**12. Pourquoi une vue SQL est-elle utile dans un environnement d'entreprise ?**
Une vue SQL est un outil stratégique permettant de centraliser les données en entreprise pour simplifier, sécuriser et fiabiliser l'accès aux données tout en garantissant leur cohérence.