SQL Workbench Assignment: MavenMovies Database

🟩 Part 1: SQL Table Creation & Constraints

Q1. Create the employees table

```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY NOT NULL,
    emp_name VARCHAR(100) NOT NULL,
    age INT CHECK (age >= 18),
    email VARCHAR(100) UNIQUE,
    salary DECIMAL(10,2) DEFAULT 30000.00
);
```

Q2. Purpose of Constraints

Constraints enforce rules on the data in a table. They ensure data accuracy and integrity.

- **PRIMARY KEY**: Unique and not null
- **FOREIGN KEY**: Maintains referential integrity
- **NOT NULL**: Ensures a column always has a value
- **UNIQUE**: Ensures no duplicate values
- **CHECK**: Limits values (e.g., age >= 18)

Q3. NOT NULL & PRIMARY KEY Explanation

- `NOT NULL` ensures that a column cannot contain NULL values.
- A `PRIMARY KEY` must be unique and not null as it uniquely identifies each row.

Q4. Add & Remove Constraints

```
ALTER TABLE employees ADD CONSTRAINT chk_age CHECK (age >= 18);
ALTER TABLE employees DROP CONSTRAINT chk_age;
```

Q5. Constraint Violation Example

```
ERROR 3819 (HY000): Check constraint 'chk_age' is violated.
```

Q6. Add Constraints to Existing Products Table

```
ALTER TABLE products ADD PRIMARY KEY (product_id);
ALTER TABLE products ALTER COLUMN price SET DEFAULT 50.00;
```

🟩 Part 2: Joins & Queries

Q7. INNER JOIN Students and Classes

```
SELECT s.student_name, c.class_name
FROM students s
INNER JOIN classes c ON s.class_id = c.class_id;
```

Q8. Show all orders and products

```
SELECT o.order_id, cu.customer_name, p.product_name
FROM products p
LEFT JOIN orders o ON p.product_id = o.product_id
LEFT JOIN customers cu ON o.customer_id = cu.customer_id;
```

Q9. Total Sales Amount per Product

```
SELECT p.product_name, SUM(o.amount) AS total_sales
FROM products p
JOIN orders o ON p.product_id = o.product_id
GROUP BY p.product_name;
```

Q10. Orders with Quantity per Customer

```
SELECT o.order_id, cu.customer_name, o.quantity
FROM customers cu
JOIN orders o ON cu.customer_id = o.customer_id;
```

🟩 Part 3: MavenMovies DB Queries

Basic Queries

```
SELECT * FROM actor;
SELECT * FROM customer;
SELECT DISTINCT country FROM country;
SELECT * FROM customer WHERE active = 1;
SELECT rental_id FROM rental WHERE customer_id = 1;
SELECT * FROM film WHERE rental_duration > 5;
SELECT COUNT(*) FROM film WHERE replacement_cost BETWEEN 15 AND 20;
SELECT COUNT(DISTINCT first_name) FROM actor;
```

```
SELECT * FROM customer LIMIT 10;
```

Pattern Matching, IN, BETWEEN

```
SELECT * FROM customer WHERE first_name LIKE 'b%' LIMIT 3;
SELECT title FROM film WHERE rating = 'G' LIMIT 5;
SELECT * FROM customer WHERE first_name LIKE 'a%';
SELECT * FROM customer WHERE first_name LIKE '%a';
SELECT city FROM city WHERE city LIKE 'a%a' LIMIT 4;
SELECT * FROM customer WHERE first_name LIKE '%NI%';
SELECT * FROM customer WHERE first_name LIKE '_r%';
SELECT * FROM customer WHERE first_name LIKE 'a____%';
SELECT * FROM customer WHERE first_name LIKE 'a%o';
SELECT * FROM film WHERE rating IN ('PG', 'PG-13');
SELECT * FROM film WHERE length BETWEEN 50 AND 100;
SELECT * FROM actor LIMIT 50;
SELECT DISTINCT film_id FROM inventory;
```

🟩 Part 4: Functions & GROUP BY

Aggregate Functions

```
SELECT COUNT(*) FROM rental;
SELECT AVG(rental_duration) FROM film;
SELECT UPPER(first_name), UPPER(last_name) FROM customer;
SELECT rental_id, MONTH(rental_date) AS rental_month FROM rental;
```

GROUP BY Examples

```
SELECT customer_id, COUNT(*) AS total_rentals FROM rental GROUP BY
customer_id;
SELECT store_id, SUM(amount) FROM payment GROUP BY store_id;

SELECT c.name, COUNT(*)
FROM category c
JOIN film_category fc ON c.category_id = fc.category_id
JOIN inventory i ON fc.film_id = i.film_id
JOIN rental r ON r.inventory_id = i.inventory_id
GROUP BY c.name;

SELECT l.name, AVG(f.rental_rate)
FROM film f
JOIN language l ON f.language_id = l.language_id
GROUP BY l.name;
```

🟩 Part 5: Joins and Advanced GROUP BY

```
SELECT f.title, c.first_name, c.last_name
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
JOIN customer c ON r.customer_id = c.customer_id;

SELECT a.first_name, a.last_name
FROM actor a
JOIN film_actor fa ON a.actor_id = fa.actor_id
JOIN film f ON fa.film_id = f.film_id
WHERE f.title = 'Gone with the Wind';

SELECT c.first_name, c.last_name, SUM(p.amount) AS total_spent
FROM customer c
JOIN payment p ON c.customer_id = p.customer_id
GROUP BY c.customer_id;

SELECT c.first_name, f.title
FROM customer c
JOIN address a ON c.address_id = a.address_id
JOIN city ct ON a.city_id = ct.city_id
JOIN rental r ON c.customer_id = r.customer_id
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN film f ON i.film_id = f.film_id
WHERE ct.city = 'London'
GROUP BY c.first_name, f.title;

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

SELECT customer_id
FROM rental r
JOIN inventory i ON r.inventory_id = i.inventory_id
GROUP BY customer_id
HAVING COUNT(DISTINCT i.store_id) = 2;
```

🟩 Part 6: Window Functions

```
SELECT c.customer_id, c.first_name, c.last_name,
       RANK() OVER (ORDER BY SUM(p.amount) DESC) AS rank
FROM customer c
JOIN payment p ON c.customer_id = p.customer_id
GROUP BY c.customer_id;

SELECT f.title, r.rental_date,
       SUM(p.amount) OVER (PARTITION BY f.film_id ORDER BY r.rental_date)
AS cum_revenue
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;

SELECT film_id, length,
       AVG(rental_duration) OVER (PARTITION BY length) AS avg_duration
FROM film;

WITH Film_Category_Rank AS (
  SELECT c.name AS category, f.title,
         RANK() OVER (PARTITION BY c.name ORDER BY COUNT(*) DESC) AS rnk
  FROM category c
  JOIN film_category fc ON c.category_id = fc.category_id
  JOIN inventory i ON fc.film_id = i.film_id
  JOIN rental r ON r.inventory_id = i.inventory_id
  JOIN film f ON fc.film_id = f.film_id
  GROUP BY c.name, f.title
)
SELECT * FROM Film_Category_Rank WHERE rnk <= 3;
```

🟩 Part 7: Normalization & CTEs

Q1. Define Normalization:
Normalization is a process used to organize a database into tables and columns to reduce data redundancy and improve data integrity. It involves dividing large tables into smaller ones and defining relationships between them.

Normal Forms:

- **1NF**: Eliminate repeating groups; ensure atomicity
- **2NF**: Meet 1NF + remove partial dependencies
- **3NF**: Meet 2NF + remove transitive dependencies

Q2. Convert a Table to 2NF Example

StudentID	StudentName	Course	Instructor
1	Alice	SQL	Mr. Smith
1	Alice	Python	Ms. Lee

Split into:

```
-- Student Table
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    StudentName VARCHAR(100)
);

-- Course Table
CREATE TABLE Course (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100),
    Instructor VARCHAR(100)
);
```

```
-- StudentCourse Mapping
CREATE TABLE StudentCourse (
    StudentID INT,
    CourseID INT,
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);
```

Q3. CTE for Active Customers and Rentals

```
WITH Active_Customers AS (
  SELECT customer_id, first_name, last_name
  FROM customer
  WHERE active = 1
),
Rental_Count AS (
  SELECT customer_id, COUNT(*) AS total_rentals
  FROM rental
  GROUP BY customer_id
)
SELECT ac.first_name, ac.last_name, rc.total_rentals
FROM Active_Customers ac
JOIN Rental_Count rc ON ac.customer_id = rc.customer_id;
```

Q4. CTE: Revenue by Category

```
WITH CategoryRevenue AS (
  SELECT c.name AS category, SUM(p.amount) AS revenue
  FROM category c
  JOIN film_category fc ON c.category_id = fc.category_id
  JOIN inventory i ON fc.film_id = i.film_id
  JOIN rental r ON r.inventory_id = i.inventory_id
  JOIN payment p ON p.rental_id = r.rental_id
  GROUP BY c.name
)
SELECT * FROM CategoryRevenue ORDER BY revenue DESC;
```

Q5. CTE: Most Popular Language Films

```
WITH LanguagePopularity AS (
  SELECT l.name AS language, COUNT(*) AS total_films
  FROM language l
  JOIN film f ON l.language_id = f.language_id
  GROUP BY l.name
)
SELECT * FROM LanguagePopularity ORDER BY total_films DESC;
```

Q6. CTE with ROW_NUMBER – Rank Payments per Customer

```
WITH RankedPayments AS (
```

```
  SELECT customer_id, amount,
         ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY amount DESC)
AS rn
  FROM payment
)
SELECT * FROM RankedPayments WHERE rn = 1;
```

Q7. CTE to Calculate Average Payment per Customer

```
WITH AvgPayment AS (
  SELECT customer_id, AVG(amount) AS avg_amount
  FROM payment
  GROUP BY customer_id
)
SELECT * FROM AvgPayment ORDER BY avg_amount DESC;
```

Q8. CTE with JOIN and Aggregation – Store-wise Revenue

```
WITH StoreRevenue AS (
  SELECT s.store_id, SUM(p.amount) AS revenue
  FROM store s
  JOIN staff st ON s.store_id = st.store_id
  JOIN payment p ON st.staff_id = p.staff_id
  GROUP BY s.store_id
)
SELECT * FROM StoreRevenue;
```

Q9. CTE to Find Customers Who Made More Than 10 Rentals

```
WITH FrequentRenters AS (
  SELECT customer_id, COUNT(*) AS rental_count
  FROM rental
  GROUP BY customer_id
  HAVING COUNT(*) > 10
)
SELECT * FROM FrequentRenters;
```

Q10. CTE for Inventory Status Summary

```
WITH InventoryStatus AS (
  SELECT i.store_id, COUNT(*) AS total_inventory
  FROM inventory i
  GROUP BY i.store_id
)
SELECT * FROM InventoryStatus;
```

Q11. CTE to List Staff Members and Their Total Payments Collected

```
WITH StaffPayments AS (
  SELECT staff_id, SUM(amount) AS total_collected
  FROM payment
  GROUP BY staff_id
)
SELECT * FROM StaffPayments;
```

Q12. CTE for Country-wise Customer Distribution

```
WITH CountryCustomer AS (
  SELECT co.country, COUNT(*) AS customer_count
  FROM customer cu
  JOIN address a ON cu.address_id = a.address_id
  JOIN city ci ON a.city_id = ci.city_id
  JOIN country co ON ci.country_id = co.country_id
  GROUP BY co.country
)
SELECT * FROM CountryCustomer ORDER BY customer_count DESC;
```