SQL Assignment - Questions and Answers

SQL Basics

Question 1

Create a table called employees with the following structure:
- emp_id (integer, should not be NULL and should be a primary key)
- emp_name (text, should not be NULL)
- age (integer, should have a check constraint to ensure the age is at least 18)
- email (text, should be unique for each employee)
- salary (decimal, with a default value of 30,000).

Write the SQL query to create the above table with all constraints.

Answer:

```
CREATE TABLE employees (
   emp_id INT PRIMARY KEY NOT NULL,
   emp_name TEXT NOT NULL,
   age INT CHECK (age >= 18),
   email TEXT UNIQUE,
   salary DECIMAL(10,2) DEFAULT 30000
);
```

Question 2

Explain the purpose of constraints and how they help maintain data integrity in a database. Provide examples of common types of constraints.

Answer:

Constraints enforce rules on table columns to maintain data integrity:
- NOT NULL → prevents missing values.
- PRIMARY KEY → uniquely identifies each record.
- UNIQUE → prevents duplicate entries (e.g., emails).
- CHECK → ensures valid ranges (e.g., age >= 18).
- DEFAULT → supplies a standard value if none is provided.
- FOREIGN KEY → maintains referential integrity between tables.

Question 3

Why would you apply the NOT NULL constraint to a column? Can a primary key contain NULL values? Justify your answer.

Answer:
- NOT NULL ensures a column must always have a value.
- A Primary Key cannot have NULL values because it uniquely identifies each row — NULL would mean 'unknown,' which breaks uniqueness.

Question 4

Explain the steps and SQL commands used to add or remove constraints on an existing table. Provide an example for both adding and removing a constraint.

Answer:
Add constraint:
ALTER TABLE employees ADD CONSTRAINT chk_salary CHECK (salary >= 20000);

Remove constraint:
ALTER TABLE employees DROP CONSTRAINT chk_salary;

Question 5

Explain the consequences of attempting to insert, update, or delete data in a way that violates constraints. Provide an example of an error message that might occur when violating a constraint.

Answer:
If data breaks a constraint, the DBMS rejects it.

Example:
INSERT INTO employees (emp_id, emp_name, age, email)
VALUES (1, 'John', 15, 'john@example.com');

Error: CHECK constraint failed: age >= 18

Question 6

You created a products table without constraints. Now, you realise that product_id should be a primary key and price should have a default value of 50.00.

Answer:

ALTER TABLE products
ADD CONSTRAINT pk_product PRIMARY KEY (product_id);

ALTER TABLE products
ALTER COLUMN price SET DEFAULT 50.00;

Question 7

You have two tables. Write a query to fetch the student_name and class_name for each student using an INNER JOIN.

Answer:

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

Question 8

Consider the following three tables. Write a query that shows all order_id, customer_name, and product_name, ensuring that all products are listed even if they are not associated with an order.

Answer:

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

Question 9

Write a query to find the total sales amount for each product using an INNER JOIN and the SUM() function.

Answer:

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

Question 10

You are given three tables. Write a query to display the order_id, customer_name, and the quantity of products ordered by each customer using an INNER JOIN between all three tables.

Answer:

```
SELECT o.order_id, c.customer_name, o.quantity
FROM orders o
INNER JOIN customers c ON o.customer_id = c.customer_id
INNER JOIN products p ON o.product_id = p.product_id;
```

SQL Commands

Question 1

Identify the primary keys and foreign keys in maven movies db. Discuss the differences

Answer:

Primary Keys: Uniquely identify records in a table (e.g., actor_id in actor table, film_id in film table).
Foreign Keys: Maintain referential integrity by linking columns to primary keys in another table (e.g., film_id in inventory table refers to film table).
Difference: PK uniquely identifies a row, FK references a PK in another table.

Question 2

List all details of actors

Answer:

SELECT * FROM actor;

Question 3

List all customer information from DB.

Answer:

SELECT * FROM customer;

Question 4

List different countries.

Answer:

SELECT DISTINCT country FROM country;

Question 5

Display all active customers.

Answer:

SELECT * FROM customer WHERE active = 1;

Question 6

List of all rental IDs for customer with ID 1.

Answer:

SELECT rental_id FROM rental WHERE customer_id = 1;

Question 7

Display all the films whose rental duration is greater than 5.

Answer:

SELECT * FROM film WHERE rental_duration > 5;

Question 8

List the total number of films whose replacement cost is greater than $15 and less than $20.

Answer:

```
SELECT COUNT(*) FROM film WHERE replacement_cost > 15 AND replacement_cost < 20;
```

Question 9

Display the count of unique first names of actors.

Answer:

```
SELECT COUNT(DISTINCT first_name) FROM actor;
```

Question 10

Display the first 10 records from the customer table.

Answer:

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

Question 11

Display the first 3 records from the customer table whose first name starts with 'b'.

Answer:

```
SELECT * FROM customer WHERE first_name LIKE 'b%' LIMIT 3;
```

Question 12

Display the names of the first 5 movies which are rated as 'G'.

Answer:

```
SELECT title FROM film WHERE rating = 'G' LIMIT 5;
```

Question 13

Find all customers whose first name starts with 'a'.

Answer:

```
SELECT * FROM customer WHERE first_name LIKE 'a%';
```

Question 14

Find all customers whose first name ends with 'a'.

Answer:

```
SELECT * FROM customer WHERE first_name LIKE '%a';
```

Question 15

Display the list of first 4 cities which start and end with 'a'.

Answer:

SELECT city FROM city WHERE city LIKE 'a%a' LIMIT 4;

Question 16

Find all customers whose first name have 'NI' in any position.

Answer:

SELECT * FROM customer WHERE first_name LIKE '%NI%';

Question 17

Find all customers whose first name have 'r' in the second position.

Answer:

SELECT * FROM customer WHERE first_name LIKE '_r%';

Question 18

Find all customers whose first name starts with 'a' and are at least 5 characters in length.

Answer:

SELECT * FROM customer WHERE first_name LIKE 'a%' AND LENGTH(first_name) >= 5;

Question 19

Find all customers whose first name starts with 'a' and ends with 'o'.

Answer:

SELECT * FROM customer WHERE first_name LIKE 'a%o';

Question 20

Get the films with pg and pg-13 rating using IN operator.

Answer:

SELECT * FROM film WHERE rating IN ('PG','PG-13');

Question 21

Get the films with length between 50 to 100 using between operator.

Answer:

SELECT * FROM film WHERE length BETWEEN 50 AND 100;

Question 22

Get the top 50 actors using limit operator.

Answer:

SELECT * FROM actor LIMIT 50;

Question 23

Get the distinct film ids from inventory table.

Answer:

SELECT DISTINCT film_id FROM inventory;

Functions

Basic Aggregate Functions

Question 1

Retrieve the total number of rentals made in the Sakila database.

Answer:

SELECT COUNT(*) AS total_rentals FROM rental;

Question 2

Find the average rental duration (in days) of movies rented from the Sakila database.

Answer:

SELECT AVG(rental_duration) AS avg_rental_days FROM film;

String Functions

Question 3

Display the first name and last name of customers in uppercase.

Answer:

SELECT UPPER(first_name), UPPER(last_name) FROM customer;

Question 4

Extract the month from the rental date and display it alongside the rental ID.

Answer:

SELECT rental_id, MONTH(rental_date) AS rental_month FROM rental;

GROUP BY

Question 5

Retrieve the count of rentals for each customer (display customer ID and the count of rentals).

Answer:

SELECT customer_id, COUNT(*) AS rental_count FROM rental GROUP BY customer_id;

Question 6

Find the total revenue generated by each store.

Answer:

SELECT store_id, SUM(amount) AS total_revenue FROM payment GROUP BY store_id;

Question 7

Determine the total number of rentals for each category of movies.

Answer:

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

Question 8

Find the average rental rate of movies in each language.

Answer:

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

Joins

Question 9

Display the title of the movie, customer's first name, and last name who rented it.

Answer:

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;

Question 10

Retrieve the names of all actors who have appeared in the film 'Gone with the Wind.'

Answer:

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';

Question 11

Retrieve the customer names along with the total amount they've spent on rentals.

Answer:

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.first_name, c.last_name;

Question 12

List the titles of movies rented by each customer in a particular city (e.g., 'London').

Answer:

```
SELECT c.first_name, c.last_name, f.title FROM customer c JOIN address a ON c.address_id =
a.address_id JOIN city ci ON a.city_id = ci.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 ci.city =
'London' GROUP BY c.first_name, c.last_name, f.title;
```

Question 13

Display the top 5 rented movies along with the number of times they've been rented.

Answer:

```
SELECT f.title, COUNT(r.rental_id) AS rental_count 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 rental_count
DESC LIMIT 5;
```

Question 14

Determine the customers who have rented movies from both stores (store ID 1 and store ID 2).

Answer:

```
SELECT c.customer_id, c.first_name, c.last_name FROM customer c JOIN rental r ON c.customer_id =
r.customer_id JOIN inventory i ON r.inventory_id = i.inventory_id WHERE i.store_id IN (1, 2) GROUP
BY c.customer_id, c.first_name, c.last_name HAVING COUNT(DISTINCT i.store_id) = 2;
```

Window Functions

Question 1

Rank the customers based on the total amount they've spent on rentals.

Answer:

```
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, c.first_name, c.last_name;
```

Question 2

Calculate the cumulative revenue generated by each film over time.

Answer:

```
SELECT f.title, p.payment_date,
    SUM(p.amount) OVER (PARTITION BY f.film_id ORDER BY p.payment_date) AS
cumulative_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;
```

Question 3

Determine the average rental duration for each film, considering films with similar lengths.

Answer:

```
SELECT f.title, f.length,
    AVG(f.rental_duration) OVER (PARTITION BY f.length) AS avg_rental_duration
FROM film f;
```

Question 4

Identify the top 3 films in each category based on their rental counts.

Answer:

```
SELECT c.name AS category, f.title,
    RANK() OVER (PARTITION BY c.name ORDER BY COUNT(r.rental_id) DESC) AS rank
FROM category c
JOIN film_category fc ON c.category_id = fc.category_id
JOIN film f ON fc.film_id = f.film_id
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
GROUP BY c.name, f.title;
```

Question 5

Find the monthly revenue trend for the entire rental store over time.

Answer:

```
SELECT DATE_TRUNC('month', payment_date) AS month,
    SUM(amount) AS monthly_revenue
FROM payment
GROUP BY DATE_TRUNC('month', payment_date)
ORDER BY month;
```

Question 6

Identify the customers whose total spending on rentals falls within the top 20% of all customers.

Answer:

```
WITH customer_spending AS (
  SELECT customer_id, SUM(amount) AS total_spent
  FROM payment
  GROUP BY customer_id
)
SELECT customer_id, total_spent
FROM (
  SELECT customer_id, total_spent,
      NTILE(5) OVER (ORDER BY total_spent DESC) AS spending_group
  FROM customer_spending
) sub
WHERE spending_group = 1;
```

Question 7

Calculate the running total of rentals per category, ordered by rental count.

Answer:

```
SELECT c.name AS category, COUNT(r.rental_id) AS rental_count,
    SUM(COUNT(r.rental_id)) OVER (PARTITION BY c.name ORDER BY c.name) AS running_total
FROM category c
JOIN film_category fc ON c.category_id = fc.category_id
JOIN film f ON fc.film_id = f.film_id
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
GROUP BY c.name;
```

Question 8

Find the films that have been rented less than the average rental count for their respective categories.

Answer:

```
WITH film_rental_counts AS (
  SELECT f.film_id, f.title, c.name AS category, COUNT(r.rental_id) AS rental_count
  FROM film f
  JOIN film_category fc ON f.film_id = fc.film_id
  JOIN category c ON fc.category_id = c.category_id
  JOIN inventory i ON f.film_id = i.film_id
  JOIN rental r ON i.inventory_id = r.inventory_id
  GROUP BY f.film_id, f.title, c.name
)
SELECT *
FROM film_rental_counts fr
WHERE rental_count < (
  SELECT AVG(rental_count)
  FROM film_rental_counts
  WHERE category = fr.category
);
```

Question 9

Identify the top 5 months with the highest revenue and display the revenue generated in each month.

Answer:

```
SELECT DATE_TRUNC('month', payment_date) AS month,
      SUM(amount) AS revenue
FROM payment
GROUP BY DATE_TRUNC('month', payment_date)
ORDER BY revenue DESC
LIMIT 5;
```

Question 10

Calculate the difference in rental counts between each customer's total rentals and the average rentals across all customers.

Answer:

```
WITH customer_rentals AS (
   SELECT customer_id, COUNT(*) AS rental_count
   FROM rental
   GROUP BY customer_id
)
SELECT customer_id, rental_count,
      rental_count - (SELECT AVG(rental_count) FROM customer_rentals) AS diff_from_avg
FROM customer_rentals;
```

Normalization & CTE

Question 1

First Normal Form (1NF): Identify a table in the Sakila database that violates 1NF. Explain how you would normalize it to achieve 1NF.

Answer:

Example: A table storing multiple phone numbers in a single column violates 1NF.

To normalize: create a new table CustomerPhones(customer_id, phone_number) so each row stores only one value per field.

Question 2

Second Normal Form (2NF): Choose a table in Sakila and describe how you would determine whether it is in 2NF. If it violates 2NF, explain the steps to normalize it.

Answer:

A table is in 2NF if it is already in 1NF and all non-key attributes depend on the whole primary key.

Example: If a table has a composite PK (order_id, product_id) but customer_name depends only on order_id, it violates 2NF.

To fix: separate into Orders(order_id, customer_name) and OrderDetails(order_id, product_id, quantity).

Question 3

Third Normal Form (3NF): Identify a table in Sakila that violates 3NF. Describe the transitive dependencies present and outline the steps to normalize.

Answer:

Example: If Orders(order_id, customer_id, customer_address) exists, customer_address depends on customer_id not order_id.

To normalize: move customer_address to the Customers table and keep only customer_id in Orders.

Question 4

Normalization Process: Take a specific table in Sakila and guide through normalization up to 2NF.

Answer:

Start: Rental(rental_id, customer_id, customer_name, film_id, film_title, rental_date)

1NF: Split repeated/multivalued data → separate Customer and Film.

2NF: Remove partial dependency → Customer(customer_id, customer_name), Film(film_id, film_title), Rental(rental_id, customer_id, film_id, rental_date).

Question 5

CTE Basics: Retrieve the distinct list of actor names and the number of films they acted in.

Answer:

```
WITH actor_films AS (
  SELECT a.actor_id, a.first_name, a.last_name, COUNT(fa.film_id) AS film_count
  FROM actor a
  JOIN film_actor fa ON a.actor_id = fa.actor_id
  GROUP BY a.actor_id, a.first_name, a.last_name
)
SELECT * FROM actor_films;
```

Question 6

CTE with Joins: Combine film and language to show film title, language, and rental rate.

Answer:

```
WITH film_lang AS (
  SELECT f.title, l.name AS language, f.rental_rate
  FROM film f
  JOIN language l ON f.language_id = l.language_id
)
SELECT * FROM film_lang;
```

Question 7

CTE for Aggregation: Find total revenue generated by each customer.

Answer:

```
WITH customer_revenue AS (
  SELECT customer_id, SUM(amount) AS total_revenue
  FROM payment
  GROUP BY customer_id
)
SELECT * FROM customer_revenue;
```

Question 8

CTE with Window Functions: Rank films by rental duration.

Answer:

```
WITH ranked_films AS (
  SELECT title, rental_duration,
      RANK() OVER (ORDER BY rental_duration DESC) AS rank
  FROM film
)
SELECT * FROM ranked_films;
```

Question 9

CTE and Filtering: List customers with more than 2 rentals.

Answer:

```
WITH frequent_customers AS (
  SELECT customer_id, COUNT(*) AS rental_count
  FROM rental
  GROUP BY customer_id
  HAVING COUNT(*) > 2
)
SELECT c.customer_id, c.first_name, c.last_name
FROM frequent_customers fc
JOIN customer c ON fc.customer_id = c.customer_id;
```

Question 10

CTE for Date Calculations: Find number of rentals made each month.

Answer:

```
WITH monthly_rentals AS (
  SELECT DATE_TRUNC('month', rental_date) AS rental_month, COUNT(*) AS total_rentals
  FROM rental
  GROUP BY DATE_TRUNC('month', rental_date)
)
SELECT * FROM monthly_rentals;
```

Question 11

CTE and Self-Join: Show pairs of actors who appeared in the same film.

Answer:

```
WITH actor_pairs AS (
  SELECT fa1.actor_id AS actor1, fa2.actor_id AS actor2, fa1.film_id
  FROM film_actor fa1
  JOIN film_actor fa2 ON fa1.film_id = fa2.film_id
  WHERE fa1.actor_id < fa2.actor_id
)
SELECT * FROM actor_pairs;
```

Question 12

Recursive CTE: Find all employees who report to a specific manager.

Answer:

```
WITH RECURSIVE employee_hierarchy AS (
  SELECT staff_id, first_name, last_name, reports_to
  FROM staff
  WHERE reports_to = 1  -- manager id
  UNION ALL
  SELECT s.staff_id, s.first_name, s.last_name, s.reports_to
  FROM staff s
  INNER JOIN employee_hierarchy eh ON s.reports_to = eh.staff_id
)
SELECT * FROM employee_hierarchy;
```