Sql Assignment Solution – PW SKILLS

Ans of Question 1 - CREATE TABLE employees (

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

Ans of Question 2 - **Purpose of Constraints**
Constraints enforce rules at the database level, ensuring data accuracy and integrity. Common types:

- **Primary Key**: Uniquely identifies each row.
- **Foreign Key**: Links data across tables.
- **Unique**: Prevents duplicate values.
- **Check**: Sets condition(s) for values.
- **Not Null**: Ensures the column always has a value.

Ans of Question 3 - **NOT NULL and Primary Key Constraints**
Applying NOT NULL ensures a column always contains data. A primary key cannot contain NULL values as it uniquely identifies records.

Ans of Question 4 - **Adding/Removing Constraints**

- **Add**: ALTER TABLE employees ADD CONSTRAINT emp_email_unique UNIQUE (email);
- **Remove**: ALTER TABLE employees DROP CONSTRAINT emp_email_unique;

Ans of Question 5 - Violating constraints triggers errors, like inserting a duplicate in a UNIQUE column. Example error: ERROR: duplicate key violates unique constraint "employees_email_key"

Ans 0f Question 6 - ALTER TABLE products

ADD PRIMARY KEY (product_id),

ALTER COLUMN price SET DEFAULT 50.00;

Ans of Question 7 - SELECT student_name, class_name

FROM students

INNER JOIN classes ON students.class_id = classes.class_id;

Ans of Question 8 - SELECT o.order_id, c.customer_name, p.product_name

FROM orders o

LEFT JOIN products p ON o.product_id = p.product_id

LEFT JOIN customers c ON o.customer_id = c.customer_id;

Ans of Question 9 - SELECT p.product_name, SUM(o.amount) AS total_sales

FROM orders o

INNER JOIN products p ON o.product_id = p.product_id

GROUP BY p.product_name;

Ans of Question 10 - SELECT o.order_id, c.customer_name, o.quantity

FROM orders o

INNER JOIN customers c ON o.customer_id = c.customer_id;

SQL Commands

1. identify the primary keys and foreign keys in maven movies db

Primary Keys:

actor: actor_id

address: address_id

category: category_id

city: city_id

country: country_id

customer: customer_id

film: film_id

film_actor: actor_id, film_id (composite key)

film_category: film_id, category_id (composite key)

film_text: film_id

inventory: inventory_id

language: language_id

payment: payment_id

rental: rental_id

staff: staff_id

store: store_id

Foreign Keys:

address → city_id (references city)

city → country_id (references country)

customer → store_id, address_id (references store, address)

film → language_id, original_language_id (references language)

film_actor → actor_id, film_id (references actor, film)

film_category → film_id, category_id (references film, category)

inventory → film_id, store_id (references film, store)

payment → customer_id, staff_id, rental_id (references customer, staff, rental)

rental → inventory_id, customer_id, staff_id (references inventory, customer, staff)

staff → address_id (references address)

store → manager_staff_id, address_id (references staff, address)

2. list all details of actors

```
select * from actor
```

3. list all customer information

```
select * from customer
```

4. list different countries

```
select distinct country from country
```

5. display all active customers

```
select * from customer where active = 1
```

6. list of all rental ids for customer with id 1

select rental_id from rental where customer_id = 1

7. display all films whose rental duration is greater than 5

select * from film where rental_duration > 5

8. total number of films with replacement cost between 15 and 20

select count(*) from film where replacement_cost > 15 and replacement_cost < 20

9. count of unique actor first names

select count(distinct first_name) from actor

10. display the first 10 records from the customer table

select * from customer limit 10

11. display the first 3 records from customer table where first name starts with b

select * from customer where first_name like 'B%' limit 3

12. names of the first 5 movies rated g

select title from film where rating = 'G' limit 5

13. find all customers whose first name starts with a

select * from customer where first_name like 'A%'

14. find all customers whose first name ends with a

select * from customer where first_name like '%A'

15. first 4 cities starting and ending with a

select city from city where city like 'A%' and city like '%A' limit 4

16. customers whose first name contains ni in any position

select * from customer where first_name like '%NI%'

17. customers whose first name has r in the second position

select * from customer where first_name like '_R%'

18. customers whose first name starts with a and is at least 5 characters

select * from customer where first_name like 'A%' and length(first_name) >= 5

19. customers whose first name starts with a and ends with o

```
select * from customer where first_name like 'A%' and first_name like '%O'
```

20. get films with pg and pg-13 ratings using in operator

```
select * from film where rating in ('PG', 'PG-13')
```

21. films with length between 50 and 100

```
select * from film where length between 50 and 100
```

22. top 50 actors using limit operator

```
select * from actor limit 50
```

23. distinct film ids from inventory table

```
select distinct film_id from inventory
```

Basic Aggregate Functions

Question 1: Retrieve the total number of rentals made in the Sakila database.

Hint: Use the COUNT() function.

`SELECT COUNT(*) AS total_rentals FROM rental;`

Question 2: Find the average rental duration (in days) of movies rented from the Sakila database.

Hint: Utilize the AVG() function.

`SELECT AVG(rental_duration) AS avg_rental_duration FROM film;`

String Functions

Question 3: Display the first name and last name of customers in uppercase.

Hint: Use the UPPER() function.

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

Question 4: Extract the month from the rental date and display it alongside the rental ID.

Hint: Employ the MONTH() function.

`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).

Hint: Use COUNT() in conjunction with GROUP BY.

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

Question 6: Find the total revenue generated by each store.

Hint: Combine SUM() and GROUP BY.

`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.

Hint: JOIN film_category, film, and rental tables, then use COUNT() and GROUP BY.

`SELECT category.category_id, category.name AS category_name, COUNT(*) AS rental_count

FROM rental

JOIN inventory ON rental.inventory_id = inventory.inventory_id

JOIN film ON inventory.film_id = film.film_id

JOIN film_category ON film.film_id = film_category.film_id

JOIN category ON film_category.category_id = category.category_id

GROUP BY category.category_id, category.name;`

Question 8: Find the average rental rate of movies in each language.

Hint: JOIN film and language tables, then use AVG() and GROUP BY.

`SELECT language.language_id, language.name AS language_name, AVG(film.rental_rate) AS avg_rental_rate

FROM film

JOIN language ON film.language_id = language.language_id

GROUP BY language.language_id, language.name;`

Joins

Question 9: Display the title of the movie, customer's first name, and last name who rented it.

Hint: Use JOIN between the film, inventory, rental, and customer tables.

`SELECT film.title, customer.first_name, customer.last_name

FROM rental

JOIN inventory ON rental.inventory_id = inventory.inventory_id

JOIN film ON inventory.film_id = film.film_id

JOIN customer ON rental.customer_id = customer.customer_id;`

Question 10: Retrieve the names of all actors who have appeared in the film "Gone with the Wind."

Hint: Use JOIN between the film_actor, film, and actor tables.

`SELECT actor.first_name, actor.last_name

FROM film

JOIN film_actor ON film.film_id = film_actor.film_id

JOIN actor ON film_actor.actor_id = actor.actor_id

WHERE film.title = 'Gone with the Wind';`

Question 11: Retrieve the customer names along with the total amount they've spent on rentals.

Hint: JOIN customer, payment, and rental tables, then use SUM() and GROUP BY.

`SELECT customer.first_name, customer.last_name, SUM(payment.amount) AS total_spent

FROM customer

JOIN payment ON customer.customer_id = payment.customer_id

GROUP BY customer.customer_id, customer.first_name, customer.last_name;`

Question 12: List the titles of movies rented by each customer in a particular city (e.g., 'London').

Hint: JOIN customer, address, city, rental, inventory, and film tables, then use GROUP BY.

`SELECT film.title, customer.first_name, customer.last_name, city.city

FROM rental

JOIN inventory ON rental.inventory_id = inventory.inventory_id

JOIN film ON inventory.film_id = film.film_id

JOIN customer ON rental.customer_id = customer.customer_id

JOIN address ON customer.address_id = address.address_id

JOIN city ON address.city_id = city.city_id

WHERE city.city = 'London'

GROUP BY film.title, customer.first_name, customer.last_name, city.city;`

Advanced Joins and GROUP BY

Question 13:
Display the top 5 rented movies along with the number of times they've been rented.
Hint: JOIN film, inventory, and rental tables, then use COUNT() and GROUP BY, and limit the results.

Answer -

```
SELECT film.title, COUNT(*) AS rental_count

FROM rental

JOIN inventory ON rental.inventory_id = inventory.inventory_id

JOIN film ON inventory.film_id = film.film_id

GROUP BY film.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).
Hint: Use JOIN with rental, inventory, and customer tables and consider COUNT() and GROUP BY.

Answer –

```
SELECT customer.customer_id, customer.first_name, customer.last_name

FROM rental

JOIN inventory ON rental.inventory_id = inventory.inventory_id

JOIN customer ON rental.customer_id = customer.customer_id

GROUP BY customer.customer_id

HAVING COUNT(DISTINCT inventory.store_id) = 2;
```

Window Functions

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

   Answer -

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

2. Calculate the cumulative revenue generated by each film over time.

   Answer –

```
SELECT film_id, title, rental_date,
    SUM(amount) OVER (PARTITION BY film_id ORDER BY rental_date) AS
cumulative_revenue
FROM film
JOIN inventory ON film.film_id = inventory.film_id
JOIN rental ON inventory.inventory_id = rental.inventory_id
JOIN payment ON rental.rental_id = payment.rental_id;
```

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

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

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

```
SELECT category_id, title, rental_count,
      RANK() OVER (PARTITION BY category_id ORDER BY rental_count
DESC) AS category_rank
FROM (
   SELECT category.category_id, film.title, COUNT(*) AS rental_count
   FROM rental
   JOIN inventory ON rental.inventory_id = inventory.inventory_id
   JOIN film ON inventory.film_id = film.film_id
   JOIN film_category ON film.film_id = film_category.film_id
   JOIN category ON film_category.category_id = category.category_id
   GROUP BY category.category_id, film.title
) AS rental_counts
WHERE category_rank <= 3;
```

5. 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 total_rentals
   FROM rental
   GROUP BY customer_id
)
SELECT customer_id, total_rentals,
      total_rentals - AVG(total_rentals) OVER () AS rental_difference
FROM customer_rentals;
```

Normalization & CTE

1. **First Normal Form (1NF):**
   - Identify any table that stores multiple values in a single column. For example, if film table has a column storing multiple actor IDs, split it into a separate table film_actor.

2. **Second Normal Form (2NF):**
    - Choose a table with composite keys (e.g., film_actor). Ensure that all non-key columns depend on the entire key. If any columns depend only on part of the key, move them to a separate table.
3. **Third Normal Form (3NF):**
    - In rental, if there is a transitive dependency (like storing customer details redundantly), ensure each non-key attribute depends directly on the primary key by creating separate tables.

Common Table Expressions (CTEs)

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

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