1. Create a table called employees with the following structure?

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 DEFAULT 30000

);

---------------------------------------------------------------------------------------------------------------

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

examples of common types of constraints?

Answer) SQL constraints are the roles which applied to table column to enforce data integrity while data insert and update, such as primary key make the field unique as well as not null. Not null defines the field should not be null, it should have data. Otherwise, it will not insert the data into database. Unique key defines that the value of the column should not be repetitive. Check constraint checks the condition then insert the data for example age >= 18, if not then data will not be inserted. Default defines that if data is not given by the user, then it will update the record with the default value.

----------------------------------------------------------------------------------------------------------------

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

Answer) The not null defines that the column value should not be empty. If a table column has not null constraint, then without value the data will not inserted into the database as not null constraint stop the empty entry.

Yes, primary key is the combination of unique and not null constraint. So, if a column with primary key constraint, then it should have value. It can’t proceed with empty value.

------------------------------------------------------------------------------------------------------------------

4) 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.

**create table department ( //** here we first create a table with 3 columns

**department\_name varchar(40),**

**department\_location varchar(40),**

**department\_vol int);**

**select \* from department**; //read the table here.

**alter table department //** here we add constaint on the existing table column department name

**add constraint different unique (department\_name);**

**insert into department values //** later we add few data to see the table properly.

**("IT", "noida", 50),**

**("HR", "pune", 20);**

select \* from department;

insert into department values

("IT", "Bang", 100);

alter table department

add department\_avg\_salary int;

select \* from department;

alter table department // we write condition to drop the constraint

drop constraint different; //the format is ( drop constraint constraint name)

----------------------------------------------------------------------------------------------------------------

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) while update, insert and delete data where constraints are applying the system is giving us error message and not process with the query/code.

For example :-

**create table employee (**

**emp\_id int primary key,**

**emp\_name varchar(30) not null,**

**age int check(age>=18),**

**email varchar(50) unique,**

**salary decimal );**

**select \* from employee;**

**insert into employee values**

**(1001, "ajay", 19, "ajay@gmail.com", 90),**

**(1002, "vijay", 23, "vijay@gmail.com", "20"),**

**(1003, "sanjay", 57, "sanjay@gmail.com", "65"),**

**(1004, "mayank", 34, "mayank@gmail.com", "49"),**

**(1004, "mayank", 34, "mayank@gmail.com", "49");**

**// here we insert duplicate record with emp\_id repeated. So its give us an error that 1001 key employee.PRIMARY KEY**

**----------------------------------------------------------------------------------------------**

**6)** You created a products table without constraints as follows: CREATE TABLE products

(product\_id INT,     product\_name VARCHAR(50),     price DECIMAL(10, 2));   Now, you realise that? : The product\_id should be a primary keyQ : The price should have a default value of 50.00

solution)

**create table products(**

**product\_id int,**

**product\_name varchar(50),**

**price decimal(10,2));**

**select \* from products;**

**alter table products** // here we adding constraints

**add constraint primary\_value primary key (product\_id),**

**modify price decimal(10,2) default**

**------------------------------------------------------------------------------------------------------**

**7) You have two tables: student and classes table.**

create database school; **// create database**

**// create table called classes**

create table classes (

class\_id int primary key,

class\_name varchar(30)

);

**//create table called students**

create table students (

student\_id int primary key,

student\_name varchar(30),

class\_id int,

foreign key (class\_id) references classes(class\_id)

);

insert into classes values //**insert data in classes table**

(101, "Math"),

(102, "Science"),

(103, "History");

insert into students values // **insert data in student table**

(1, "Alice", "101"),

(2, "Bob", 102),

(3, "Charlie", 103);

select \* from classes;

select \* from students;

select class\_name, student\_name from **// join both the table here**

classes inner join students

on

classes.class\_id = students.class\_id;

--------------------------------------------------------------------------------------------

8) . Consider the following three tables: - Order table, customer table and product table.

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;

9) 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

create database amazon; // create database

create table productss (

product\_id int,

product\_name varchar(40)

);

insert into productss values

(101, "laptop"),

(102, "Mobile");

alter table productss

add constraint unique\_value unique (product\_id);

create table sales (

sales\_id int,

product\_id int,

amount int,

foreign key (product\_id) references productss(product\_id)

);

insert into sales values

(1,101,500),

(2,102,300),

(3,101,700);

select product\_name, sum(amount) as Total\_Sales from //Here join the 2 table and group by product\_id

productss inner join sales

on

productss.product\_id = sales.product\_id

group by product\_name;

10) 3 table Order table, customer table and order\_detail table .

Answer) SELECT o.order\_id, c.customer\_name, od.quantity

FROM orders o

INNER JOIN customers c ON o.customer\_id = c.customer\_id

INNER JOIN order\_details od ON o.order\_id = od.order\_id;

);

**SQL COMMANDS**

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

Answer) In maven movies, we have so many table such as actor, actor\_award, address, category, city, film actor etc. in each table one is primary key.

For example in **actor table – actor\_id is primary key, in actor award table – actor aware\_id is primary key and actor\_id is foreign key.**

The primary key contains unique and not null constraint. Each table should have at least one primary key. And the key which is reference from another table is call foreign key.

1. List all details of actors.

Answer) select \* from actor;

1. List all customer information from DB.

Answer) select \* from customer;

1. List different countries.

Answer) select \* from country;

1. Display all active customers.

Answer) select \* from customer

where active = 1;

1. List of all rental IDs for customer with ID 1.

Answer) select \* from rental

where customer\_id = 1;

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

Answer) select \* from film

where rental\_duration >5;

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

Answer) select \* from film

where replacement\_cost >15 and replacement\_cost <20;

1. Display the count of unique first names of actors.

Answer) select count(distinct first\_name) as Actor\_first\_name from actor;

1. Display the first 10 records from the customer table .

Answer) select \* from customer

limit 10;

1. 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;

1. Display the names of the first 5 movies which are rated as ‘G’.

Answer) select \* from film

where rating = "G"

limit 5;

1. -Find all customers whose first name starts with "a".

Answer) select \* from customer

where first\_name like "a%";

1. Find all customers whose first name ends with "a”.

Answer) select \* from customer

where first\_name like "%a";

1. Display the list of first 4 cities which start and end with ‘a’

Answer) select \* from city

where city like "a%" and city like "%a"

limit 4;

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

Answer) select \* from customer

where first\_name like "%ni%";

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

Answer) select \* from customer

where first\_name like "\_r%";

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

Asmwer) select \* from customer

where first\_name like "a%" and length(first\_name) >=5;

1. 9- Find all customers whose first name starts with "a" and ends with "o".

Answer) select \* from customer

where first\_name like "a%" and first\_name like "%o";

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

Answer) select \* from film

where rating IN ("PG", "PG-13");

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

Answer) select \* from film

where film\_id between 50 and 100;

1. Get the top 50 actors using limit operator.

Answer) select \* from actor

limit 50;

1. Get the distinct film ids from inventory table.

Answer) select distinct film\_id from inventory;

**FUNCTIONS**

1. Retrieve the total number of rentals made in the Sakila database. Hint: Use the COUNT() function.

Answer) SELECT count(rental\_id) FROM sakila.`sakila.rental`;

Or other way is -

SELECT count(\*)

FROM sakila.`sakila.rental`;

1. Find the average rental duration (in days) of movies rented from the Sakila database. Hint: Utilize the AVG() function.

Answer)

select round(avg(datediff(return\_date, rental\_date)),0) as Avg\_duration

from sakila.`sakila.rental`;

1. Display the first name and last name of customers in uppercase. Hint: Use the UPPER () function.

Answer) select upper(first\_name), upper(last\_name) from customer;

1. Extract the month from the rental date and display it alongside the rental ID. Hint: Employ the MONTH() function.

Answer) select rental\_id, month(rental\_date) from sakila.`sakila.rental`;

1. Retrieve the count of rentals for each customer (display customer ID and the count of rentals). Hint: Use COUNT () in conjunction with GROUP BY.

Answer) select customer\_id, count(rental\_id) as rental from sakila.`sakila.rental`

group by customer\_id;

1. Find the total revenue generated by each store. Hint: Combine SUM() and GROUP BY.

Answer)

SELECT store\_id,

SUM(amount) AS total\_revenue

FROM payment

JOIN staff ON payment.staff\_id = staff.staff\_id

GROUP BY store\_id;

> `payment.staff\_id → staff.store\_id` is used to associate payments with stores.

1. 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.

Answer)

SELECT c.name AS category\_name,

COUNT(r.rental\_id) AS total\_rentals

FROM rental r

JOIN inventory i ON r.inventory\_id = i.inventory\_id

JOIN film f ON i.film\_id = f.film\_id

JOIN film\_category fc ON f.film\_id = fc.film\_id

JOIN category c ON fc.category\_id = c.category\_id

GROUP BY c.name;

1. Find the average rental rate of movies in each language. Hint: JOIN film and language tables, then use AVG () and GROUP BY

Answer)

SELECT l**.**name AS language,

AVG(f**.**rental\_rate) AS average\_rental\_rate

FROM film f

JOIN language l ON f**.**language\_id **=** l**.**language\_id

GROUP BY l**.**name;

**JOINS**

1. 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.

Answer)

SELECT f.title,

c.first\_name,

c.last\_name

FROM rental r

JOIN inventory i ON r.inventory\_id = i.inventory\_id

JOIN film f ON i.film\_id = f.film\_id

JOIN customer c ON r.customer\_id = c.customer\_id;

1. 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.

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';

1. 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

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.customer\_id, c.first\_name, c.last\_name;

1. 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.

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;

1. 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 f**.**title,

COUNT(r**.**rental\_id) AS times\_rented

FROM rental r

JOIN inventory i ON r**.**inventory\_id **=** i**.**inventory\_id

JOIN film f ON i**.**film\_id **=** f**.**film\_id

GROUP BY f**.**title

ORDER BY times\_rented DESC

LIMIT 5;

1. Determine the customers who have rented movies from both stores (store ID 1 and store ID 2). Hint: Use JOINS with rental, inventory, and customer tables and consider COUNT() and GROUP BY.

Answer)

SELECT customer\_id

FROM (

SELECT DISTINCT customer\_id, store\_id

FROM rental r

JOIN inventory i ON r**.**inventory\_id **=** i**.**inventory\_id

JOIN staff s ON r**.**staff\_id **=** s**.**staff\_id

) AS customer\_stores

GROUP BY customer\_id

HAVING COUNT(DISTINCT store\_id) **=** 2;

**Windows Function:**

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

```sql

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, first\_name, last\_name;

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

```sql

SELECT f.title,

r.rental\_date,

SUM(p.amount) OVER (PARTITION BY f.film\_id ORDER BY r.rental\_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;

```

3. Average rental duration for each film, considering films with similar lengths\*\*

```sql

SELECT film\_id,

title,

length,

AVG(rental\_duration) OVER (PARTITION BY length) AS avg\_rental\_duration\_for\_length\_group

FROM film;

```

---4. Top 3 films in each category based on rental counts\*\*

```sql

SELECT category\_name, title, rental\_count

FROM (

SELECT c.name AS category\_name,

f.title,

COUNT(r.rental\_id) AS rental\_count,

RANK() OVER (PARTITION BY c.name ORDER BY COUNT(r.rental\_id) DESC) AS film\_rank

FROM rental r

JOIN inventory i ON r.inventory\_id = i.inventory\_id

JOIN film f ON i.film\_id = f.film\_id

JOIN film\_category fc ON f.film\_id = fc.film\_id

JOIN category c ON fc.category\_id = c.category\_id

GROUP BY c.name, f.title

) ranked

WHERE film\_rank <= 3;

```

---5. Difference in rental counts between each customer and the average rentals\*\*

```sql

SELECT customer\_id,

COUNT(rental\_id) AS total\_rentals,

ROUND(AVG(COUNT(rental\_id)) OVER (), 2) AS avg\_rentals,

COUNT(rental\_id) - ROUND(AVG(COUNT(rental\_id)) OVER (), 2) AS rental\_diff

FROM rental

GROUP BY customer\_id;

```

---6. Monthly revenue trend for the entire rental store\*\*

```sql

SELECT DATE\_FORMAT(payment\_date, '%Y-%m') AS month,

SUM(amount) AS total\_revenue

FROM payment

GROUP BY DATE\_FORMAT(payment\_date, '%Y-%m')

ORDER BY month;

```

---7. Customers in top 20% by total spending\*\*

```sql

WITH ranked\_customers AS (

SELECT customer\_id,

first\_name,

last\_name,

SUM(amount) AS total\_spent,

PERCENT\_RANK() OVER (ORDER BY SUM(amount) DESC) AS percentile

FROM customer

JOIN payment ON customer.customer\_id = payment.customer\_id

GROUP BY customer\_id, first\_name, last\_name

)

SELECT \*

FROM ranked\_customers

WHERE percentile <= 0.2;

```

---8. Running total of rentals per category\*\*

```sql

SELECT c.name AS category\_name,

COUNT(r.rental\_id) AS rental\_count,

SUM(COUNT(r.rental\_id)) OVER (ORDER BY c.name) AS running\_total

FROM rental r

JOIN inventory i ON r.inventory\_id = i.inventory\_id

JOIN film f ON i.film\_id = f.film\_id

JOIN film\_category fc ON f.film\_id = fc.film\_id

JOIN category c ON fc.category\_id = c.category\_id

GROUP BY c.name;

```

---9. Films rented less than the average rental count in their category\*\*

```sql

WITH film\_rentals 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

),

avg\_rentals AS (

SELECT category,

AVG(rental\_count) AS avg\_rentals

FROM film\_rentals

GROUP BY category

)

SELECT fr.title,

fr.category,

fr.rental\_count,

ar.avg\_rentals

FROM film\_rentals fr

JOIN avg\_rentals ar ON fr.category = ar.category

WHERE fr.rental\_count < ar.avg\_rentals;

```

---10. Top 5 months with highest revenue\*\*

```sql

SELECT revenue\_month,

total\_revenue

FROM (

SELECT DATE\_FORMAT(payment\_date, '%Y-%m') AS revenue\_month,

SUM(amount) AS total\_revenue,

RANK() OVER (ORDER BY SUM(amount) DESC) AS revenue\_rank

FROM payment

GROUP BY DATE\_FORMAT(payment\_date, '%Y-%m')

) ranked\_months

WHERE revenue\_rank <= 5;