SQL Assignment Solutions (Completed)

* 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(12,2) DEFAULT 30000.00);

* 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 are rules defined on table columns that enforce data integrity and correctness. They prevent invalid data, enforce relationships, and help maintain consistency. Common constraints:  
- PRIMARY KEY: Uniquely identifies each row and implies NOT NULL and UNIQUE.  
- NOT NULL: Ensures a column cannot be NULL.  
- UNIQUE: Ensures all values in a column are distinct.  
- FOREIGN KEY: Enforces referential integrity between related tables.  
- CHECK: Enforces a condition on column values (e.g., age >= 18).  
- DEFAULT: Provides a default value when none is supplied.  
These constraints reduce corruption, enforce business rules, and make querying reliable.

* 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 prevents missing values where a value is required for correctness (e.g., emp\_name). A primary key cannot contain NULL values because it must uniquely identify each row; NULL represents unknown/absent and cannot serve as a unique identifier. Most DBMS enforce NOT NULL on primary key columns.

* 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:

To add a constraint:  
- Use ALTER TABLE ... ADD CONSTRAINT ... depending on constraint type.  
Example: Add UNIQUE to email:  
ALTER TABLE employees ADD CONSTRAINT uq\_employees\_email UNIQUE (email);  
  
To add CHECK:  
ALTER TABLE employees ADD CONSTRAINT chk\_age CHECK (age >= 18);  
  
To remove a constraint:  
- Use ALTER TABLE ... DROP CONSTRAINT constraint\_name; (syntax varies: some DBs use DROP CONSTRAINT, MySQL uses DROP CHECK or DROP INDEX for UNIQUE depending)  
Example (Postgres/MySQL 8+):  
ALTER TABLE employees DROP CONSTRAINT uq\_employees\_email;  
  
In MySQL if constraint was created implicitly for UNIQUE index:  
ALTER TABLE employees DROP INDEX uq\_employees\_email;

* 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:

When operations violate constraints, the DBMS rejects the operation and returns an error; the transaction may be rolled back depending on transaction settings. Consequences:  
- INSERT of duplicate primary key or unique value -> error and no row inserted.  
- INSERT/UPDATE violating CHECK or NOT NULL -> error.  
- DELETE of referenced parent row without ON DELETE CASCADE -> foreign key violation error.  
Example errors:  
- PostgreSQL: ERROR: duplicate key value violates unique constraint "employees\_pkey"  
- MySQL: ERROR 1452: Cannot add or update a child row: a foreign key constraint fails  
- CHECK violation: ERROR: new row for relation "employees" violates check constraint "chk\_age"

* 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 key  
   The price should have a default value of 50.00

Answer:

Option A — modify existing table (ALTER TABLE):  
ALTER TABLE products  
 ADD CONSTRAINT pk\_products PRIMARY KEY (product\_id);  
  
ALTER TABLE products  
 ALTER COLUMN price SET DEFAULT 50.00; -- PostgreSQL syntax  
  
MySQL syntax to set default:  
ALTER TABLE products MODIFY price DECIMAL(10,2) DEFAULT 50.00;  
  
Option B — recreate table with constraints:  
CREATE TABLE products (  
 product\_id INT PRIMARY KEY,  
 product\_name VARCHAR(50),  
 price DECIMAL(10,2) DEFAULT 50.00  
);

* 7. You have two tables:  
    
  Write a query to fetch the student\_name and class\_name for each student using an INNER JOIN.

Answer:

-- Assuming tables: students(student\_id, student\_name, class\_id) and classes(class\_id, class\_name)  
SELECT s.student\_name, c.class\_name  
FROM students s  
INNER JOIN classes c ON s.class\_id = c.class\_id;

* 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  
    
  Hint: (use INNER JOIN and LEFT JOIN)

Answer:

-- Assuming tables: orders(order\_id, customer\_id, product\_id), customers(customer\_id, customer\_name), products(product\_id, product\_name)  
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  
ORDER BY p.product\_id;

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

Answer:

-- Assuming tables: orders(order\_id, product\_id, quantity, price\_each) and products(product\_id, product\_name)  
SELECT p.product\_id, p.product\_name, SUM(o.quantity \* o.price\_each) AS total\_sales  
FROM products p  
INNER JOIN orders o ON p.product\_id = o.product\_id  
GROUP BY p.product\_id, p.product\_name;

* 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:

-- Assuming orders(order\_id, customer\_id), customers(customer\_id, customer\_name), order\_items(order\_item\_id, order\_id, product\_id, quantity)  
SELECT o.order\_id, c.customer\_name, oi.quantity  
FROM orders o  
INNER JOIN customers c ON o.customer\_id = c.customer\_id  
INNER JOIN order\_items oi ON o.order\_id = oi.order\_id;

# SQL Commands (Maven Movies DB - Sakila-like)

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

Answer:  
Primary keys uniquely identify rows. Foreign keys reference primary keys in other tables to enforce referential integrity.  
Examples (Sakila-style): film(film\_id PK), actor(actor\_id PK), film\_actor(film\_id, actor\_id) with FK film\_id -> film(film\_id), FK actor\_id -> actor(actor\_id).

* 2- List all details of actors

Answer:  
SELECT \* FROM actor;

* 3 -List all customer information from DB.

Answer:  
SELECT \* FROM customer;

* 4 -List different countries.

Answer:  
SELECT DISTINCT country FROM country;

* 5 -Display all active customers.

Answer:  
SELECT \* FROM customer WHERE active = 1; -- or active = 'true' depending on schema

* 6 -List of all rental IDs for customer with ID 1.

Answer:  
SELECT rental\_id FROM rental WHERE customer\_id = 1;

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

Answer:  
SELECT \* FROM film WHERE rental\_duration > 5;

* 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;

* 9 - Display the count of unique first names of actors.

Answer:  
SELECT COUNT(DISTINCT first\_name) FROM actor;

* 10- Display the first 10 records from the customer table .

Answer:  
SELECT \* FROM customer LIMIT 10;

* 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;

* 12-Display the names of the first 5 movies which are rated as ‘G’.

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

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

Answer:  
SELECT \* FROM customer WHERE first\_name LIKE 'a%';

* 14- Find all customers whose first name ends with "a".

Answer:  
SELECT \* FROM customer WHERE first\_name LIKE '%a';

* 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;

* 16- Find all customers whose first name have "NI" in any position.

Answer:  
SELECT \* FROM customer WHERE first\_name LIKE '%NI%'; -- case-sensitive depending on DB; use ILIKE in Postgres for case-insensitive

* 17- Find all customers whose first name have "r" in the second position .

Answer:  
SELECT \* FROM customer WHERE first\_name LIKE '\_r%';

* 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 CHAR\_LENGTH(first\_name) >= 5;

* 19- Find all customers whose first name starts with "a" and ends with "o".

Answer:  
SELECT \* FROM customer WHERE first\_name LIKE 'a%o';

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

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

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

Answer:  
SELECT \* FROM film WHERE length BETWEEN 50 AND 100;

* 22 - Get the top 50 actors using limit operator.

Answer:  
SELECT \* FROM actor LIMIT 50;

* 23 - Get the distinct film ids from inventory table.

Answer:  
SELECT DISTINCT film\_id FROM inventory;

# Functions

1. 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\_duration FROM film; -- or AVG(EXTRACT(day FROM (return\_date - rental\_date))) if rental durations computed from dates

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

Answer:  
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.

Answer:  
SELECT rental\_id, EXTRACT(MONTH FROM rental\_date) AS rental\_month FROM rental; -- use MONTH(rental\_date) in MySQL

* Question 5: Retrieve the count of rentals for each customer (display customer ID and the count of rentals).

Answer:  
SELECT customer\_id, COUNT(\*) AS rentals\_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 fc.category\_id, c.name AS category\_name, COUNT(r.rental\_id) AS rental\_count  
FROM film\_category fc  
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  
JOIN category c ON fc.category\_id = c.category\_id  
GROUP BY fc.category\_id, c.name;

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

Answer:  
SELECT l.language\_id, l.name AS language\_name, AVG(f.rental\_rate) AS avg\_rental\_rate  
FROM film f  
JOIN language l ON f.language\_id = l.language\_id  
GROUP BY l.language\_id, 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.customer\_id, 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;

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

Answer:  
SELECT c.customer\_id, 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.customer\_id, c.first\_name, c.last\_name, f.title;

# Advanced Joins and GROUP BY

* 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 times\_rented  
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.film\_id, f.title  
ORDER BY times\_rented 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 rental r  
JOIN customer c ON r.customer\_id = c.customer\_id  
JOIN inventory i ON r.inventory\_id = i.inventory\_id  
GROUP BY c.customer\_id, c.first\_name, c.last\_name  
HAVING SUM(CASE WHEN i.store\_id = 1 THEN 1 ELSE 0 END) > 0  
 AND SUM(CASE WHEN i.store\_id = 2 THEN 1 ELSE 0 END) > 0;

# Window Functions

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

Answer:  
SELECT customer\_id, total\_spent,  
 RANK() OVER (ORDER BY total\_spent DESC) AS rank  
FROM (  
 SELECT c.customer\_id, SUM(p.amount) AS total\_spent  
 FROM customer c  
 JOIN payment p ON c.customer\_id = p.customer\_id  
 GROUP BY c.customer\_id  
) t;

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

Answer:  
SELECT film\_id, payment\_date, revenue,  
 SUM(revenue) OVER (PARTITION BY film\_id ORDER BY payment\_date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cumulative\_revenue  
FROM (  
 -- derived table listing film payments by date  
) s;

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

Answer:  
SELECT f.film\_id, f.title, AVG(r.rental\_duration) OVER (PARTITION BY f.length) AS avg\_rental\_duration\_for\_length  
FROM film f  
JOIN inventory i ON f.film\_id = i.film\_id  
JOIN rental r ON i.inventory\_id = r.inventory\_id;

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

Answer:  
SELECT category\_id, film\_id, title, times\_rented  
FROM (  
 SELECT fc.category\_id, f.film\_id, f.title, COUNT(r.rental\_id) AS times\_rented,  
 ROW\_NUMBER() OVER (PARTITION BY fc.category\_id ORDER BY COUNT(r.rental\_id) DESC) AS rn  
 FROM film\_category fc  
 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 fc.category\_id, f.film\_id, f.title  
) t  
WHERE rn <= 3;

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

Answer:  
WITH cust\_counts AS (  
 SELECT customer\_id, COUNT(\*) AS rentals\_count  
 FROM rental  
 GROUP BY customer\_id  
), avg\_r AS (  
 SELECT AVG(rentals\_count) AS avg\_rentals FROM cust\_counts  
)  
SELECT c.customer\_id, c.rentals\_count, c.rentals\_count - a.avg\_rentals AS diff\_from\_avg  
FROM cust\_counts c CROSS JOIN avg\_r a;

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

Answer:  
SELECT DATE\_TRUNC('month', payment\_date) AS month, SUM(amount) AS revenue  
FROM payment  
GROUP BY DATE\_TRUNC('month', payment\_date)  
ORDER BY month;

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

Answer:  
WITH totals 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,  
 CUME\_DIST() OVER (ORDER BY total\_spent) AS cum\_dist  
 FROM totals  
) t  
WHERE cum\_dist >= 0.80;

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

Answer:  
SELECT category\_id, title, times\_rented,  
 SUM(times\_rented) OVER (ORDER BY times\_rented DESC ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS running\_total  
FROM (  
 SELECT fc.category\_id, f.title, COUNT(r.rental\_id) AS times\_rented  
 FROM film\_category fc  
 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 fc.category\_id, f.title  
) s;

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

Answer:  
WITH counts AS (  
 SELECT fc.category\_id, f.film\_id, f.title, COUNT(r.rental\_id) AS times\_rented  
 FROM film\_category fc  
 JOIN film f ON fc.film\_id = f.film\_id  
 LEFT JOIN inventory i ON f.film\_id = i.film\_id  
 LEFT JOIN rental r ON i.inventory\_id = r.inventory\_id  
 GROUP BY fc.category\_id, f.film\_id, f.title  
), category\_avg AS (  
 SELECT category\_id, AVG(times\_rented) AS avg\_rented  
 FROM counts  
 GROUP BY category\_id  
)  
SELECT c.film\_id, c.title, c.times\_rented, ca.avg\_rented  
FROM counts c  
JOIN category\_avg ca ON c.category\_id = ca.category\_id  
WHERE c.times\_rented < ca.avg\_rented;

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

Answer:  
SELECT month, revenue  
FROM (  
 SELECT DATE\_TRUNC('month', payment\_date) AS month, SUM(amount) AS revenue  
 FROM payment  
 GROUP BY DATE\_TRUNC('month', payment\_date)  
) t  
ORDER BY revenue DESC  
LIMIT 5;