**SQL Basics Assignment - Answers**

**1. SQL Query to Create the 'employees' Table**

```
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. Purpose of Constraints & Common Types**

Constraints help maintain data integrity by ensuring that only valid and meaningful data is stored in the database. Common types of constraints:

- **NOT NULL**: Ensures a column cannot have NULL values.
- **UNIQUE**: Ensures all values in a column are different.
- **PRIMARY KEY**: Uniquely identifies a record (combines NOT NULL + UNIQUE).
- **FOREIGN KEY**: Establishes a relationship between tables.
- **CHECK**: Defines conditions that data must meet.
- **DEFAULT**: Assigns a default value if no value is provided.

**3. NOT NULL & Primary Key Constraint**

- **NOT NULL** prevents missing values in a column, ensuring data completeness.
- **Primary Key** is a combination of **NOT NULL + UNIQUE**, meaning it cannot contain NULL values.
- Example:
- CREATE TABLE students (
-     student_id INT PRIMARY KEY,  -- Cannot be NULL or duplicate
-     student_name TEXT NOT NULL
- );

**4. Adding & Removing Constraints**

- **Adding a Constraint:**
- ALTER TABLE employees ADD CONSTRAINT chk_age CHECK (age >= 18);
- **Removing a Constraint:**
- ALTER TABLE employees DROP CONSTRAINT chk_age;

**5. Consequences of Violating Constraints**

If a constraint is violated, an error occurs. Example:

```
INSERT INTO employees (emp_id, emp_name, age) VALUES (1, 'John', 15);
```

This will fail due to the CHECK (age >= 18) constraint. **Error Message:** ERROR: new row for relation "employees" violates check constraint "chk_age"

**6. Altering 'products' Table to Add Constraints**

```
ALTER TABLE products

ADD CONSTRAINT pk_product PRIMARY KEY (product_id),

ADD CONSTRAINT df_price DEFAULT 50.00 FOR price;
```

**7. INNER JOIN Query (Students & Classes)**

```
SELECT student_name, class_name

FROM students

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

**8. INNER JOIN & LEFT JOIN Query (Orders, Customers, Products)**

```
SELECT orders.order_id, customers.customer_name, products.product_name

FROM orders

LEFT JOIN order_details ON orders.order_id = order_details.order_id

LEFT JOIN products ON order_details.product_id = products.product_id

LEFT JOIN customers ON orders.customer_id = customers.customer_id;
```

**9. Total Sales Amount per Product**

```
SELECT products.product_name, SUM(order_details.quantity * order_details.price) AS total_sales

FROM order_details

INNER JOIN products ON order_details.product_id = products.product_id

GROUP BY products.product_name;
```

**10. Order Details with Customer Names & Quantity**

```
SELECT orders.order_id, customers.customer_name, SUM(order_details.quantity) AS total_quantity

FROM orders

INNER JOIN customers ON orders.customer_id = customers.customer_id

INNER JOIN order_details ON orders.order_id = order_details.order_id

GROUP BY orders.order_id, customers.customer_name;
```

---

**SQL Commands for Maven Movies Database**

**1. Identifying Primary & Foreign Keys**

- **Primary Key**: Uniquely identifies a record in a table.
- **Foreign Key**: References a primary key from another table to establish relationships.
- Example:
- SELECT COLUMN_NAME, CONSTRAINT_TYPE
- FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
- WHERE TABLE_NAME = 'maven_movies';

**2. List All Actors:**

SELECT * FROM actors;

**3. List All Customer Information:**

SELECT * FROM customers;

**4. List Different Countries:**

SELECT DISTINCT country FROM addresses;

**5. Display All Active Customers:**

SELECT * FROM customers WHERE status = 'Active';

**6. Rental IDs for Customer with ID 1:**

SELECT rental_id FROM rentals WHERE customer_id = 1;

**7. Movies with Rental Duration > 5 Days:**

SELECT * FROM films WHERE rental_duration > 5;

**8. Count of Films with Replacement Cost Between $15-$20:**

SELECT COUNT(*) FROM films WHERE replacement_cost BETWEEN 15 AND 20;

**9. Count of Unique Actor First Names:**

SELECT COUNT(DISTINCT first_name) FROM actors;

**10. First 10 Records from Customer Table:**

SELECT * FROM customers LIMIT 10;

---

**Basic Aggregate Functions**

**1. Total Rentals in Sakila Database:**

SELECT COUNT(*) FROM rentals;

**2. Average Rental Duration:**

```
SELECT AVG(rental_duration) FROM films;
```

## String Functions

**3. Customers' Names in Uppercase:**

```
SELECT UPPER(first_name), UPPER(last_name) FROM customers;
```

**4. Extract Month from Rental Date:**

```
SELECT rental_id, MONTH(rental_date) FROM rentals;
```

---

## Joins & GROUP BY

**5. Count of Rentals Per Customer:**

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

**6. Total Revenue Per Store:**

```
SELECT store_id, SUM(amount) FROM payments GROUP BY store_id;
```

**7. Top 5 Most Rented Movies:**

```
SELECT films.title, COUNT(rentals.rental_id) AS rental_count
FROM films
JOIN inventory ON films.film_id = inventory.film_id
JOIN rentals ON inventory.inventory_id = rentals.inventory_id
GROUP BY films.title
ORDER BY rental_count DESC
LIMIT 5;
```

---

## CTE (Common Table Expressions)

**1. CTE for Total Revenue Per Customer:**

```
WITH CustomerRevenue AS (
    SELECT customer_id, SUM(amount) AS total_spent FROM payments GROUP BY customer_id
)
SELECT * FROM CustomerRevenue;
```

**2. CTE with Window Function (Ranking Films by Rental Duration):**

```
WITH FilmRanking AS (
    SELECT film_id, title, rental_duration,
           RANK() OVER (ORDER BY rental_duration DESC) AS rank
    FROM films
)
SELECT * FROM FilmRanking WHERE rank <= 3;
```