```
-- SQL Assignment

USE mavenmovies;


-- SQL Basic

-- Answer 1

CREATE TABLE employees (

   emp_id INT NOT NULL PRIMARY KEY,

   emp_name VARCHAR(100) NOT NULL,

   age INT CHECK (age >= 18),

   email VARCHAR(255) NOT NULL UNIQUE,

   salary DECIMAL(10, 2) DEFAULT 30000

);


-- Answer 2

/* Constraints are rules applied to database columns to ensure data integrity, which helps maintain the accuracy,

consistency, and validity of data within a database. Constraints restrict the types of data that can be inserted,

updated, or deleted, preventing errors and maintaining reliable relationships between tables.

Here's an overview of common constraints and their purposes:


1. NOT NULL Constraint

Ensures that a column cannot store NULL values, meaning every row must have a value for that column.

Example: In an employees table, a NOT NULL constraint on the emp_name column ensures every employee has a name.

emp_name VARCHAR(50) NOT NULL


2. UNIQUE Constraint

Ensures that all values in a column are distinct, preventing duplicate entries.
```

Example: In a users table, a UNIQUE constraint on the email column ensures that each user has a unique email address.

email VARCHAR(255) UNIQUE

3. PRIMARY KEY Constraint

Combines NOT NULL and UNIQUE constraints, enforcing that a column (or a set of columns) uniquely identifies each row in a table.

Example: In an employees table, the emp_id column can be a PRIMARY KEY to uniquely identify each employee.

emp_id INT NOT NULL PRIMARY KEY

4. FOREIGN KEY Constraint

Enforces a link between two tables by requiring that a column's value corresponds to a value in another table. This constraint is essential

for maintaining referential integrity across tables.

Example: In an orders table, a FOREIGN KEY on customer_id can reference the customer_id in the customers table,

ensuring that every order is linked to a valid customer.

customer_id INT, FOREIGN KEY (customer_id) REFERENCES customers(customer_id)

5. CHECK Constraint

Specifies a condition that each value in the column must satisfy. CHECK constraints can limit the range or format of data allowed in a column.

Example: In an employees table, a CHECK constraint on the age column ensures that employees are at least 18 years old.

age INT CHECK (age >= 18)

6. DEFAULT Constraint

Assigns a default value to a column if no value is provided during an INSERT operation.

Example: In an employees table, a DEFAULT constraint on salary can set a default salary of 30,000 for new employees who do not have a

specified salary.

salary DECIMAL(10, 2) DEFAULT 30000

Benefits of Constraints

Constraints are essential for:

Maintaining Consistency: They prevent inconsistent data (e.g., duplicate or missing values).

Ensuring Accuracy: They enforce valid data, such as a minimum age for employees or a valid reference in a foreign key.

Enabling Referential Integrity: They create logical links across tables that reflect real-world relationships, ensuring

linked data is accurate and meaningful.

By enforcing these rules, constraints play a fundamental role in maintaining the overall integrity and reliability of the database.

*/

-- Answer 3

/* The NOT NULL constraint is applied to a column in SQL to ensure that every row in that column must contain a value.

This is important for the following reasons:

Data Integrity: By enforcing that a column cannot have a NULL value, you ensure that the data in that column is always present.

This is essential for maintaining the integrity of your data, especially for columns that are crucial for business logic or calculation.

Avoiding Incomplete Data: In some cases, it is important that certain fields, such as email, phone_number, or order_date,

should always contain valid information for the proper functioning of applications, reports, or queries.

Enforcing Business Rules: Many applications require that certain information must always be recorded, such as mandatory fields

(e.g., customer_id or employee_id). The NOT NULL constraint helps enforce such rules.

Improving Query Performance: Since NULL values do not need to be handled during operations, excluding them can result in better performance,

particularly when querying large datasets.

No, a primary key cannot contain NULL values. This is because a primary key is defined to uniquely identify each record in a table,

and NULL values are inherently not unique.

Here’s why:

Uniqueness Requirement: A primary key must ensure that every row in the table can be uniquely identified.

NULL values are considered "unknown" and cannot guarantee uniqueness because two NULLs in a column are treated as distinct values

when checking uniqueness, which would violate the primary key’s requirement for uniqueness.

Non-nullable: A primary key must always have a value (non-NULL) in every row. This ensures that each row can be consistently referenced

and identified. If a column allows NULL values, it could cause confusion when trying to identify a specific row or associate it

with other tables in the database.

*/

-- Answer 4

/* Adding or Removing Constraints on an Existing Table

In SQL, constraints are rules that define how data in a table should behave. These constraints can be added or removed from existing tables

using specific SQL commands. Let's go through the steps for both actions:

1. Adding a Constraint to an Existing Table

To add a constraint to an existing table, we use the ALTER TABLE statement with the ADD CONSTRAINT clause.

General Syntax or Example/Steps:

ALTER TABLE table_name

ADD CONSTRAINT constraint_name constraint_type (column_name);

2. Removing a Constraint from an Existing Table

To remove a constraint from an existing table, we use the ALTER TABLE statement with the DROP CONSTRAINT clause.

General Syntax or Example/Steps:

ALTER TABLE table_name

DROP CONSTRAINT constraint_name;

Important Points for Adding and removing constrain.

a. When adding or removing constraints, ensure that the operation does not violate existing data or business logic.

For example, if you are adding a NOT NULL constraint, make sure that the column does not contain

any NULL values before applying it.

b. In some cases (like in older versions of MySQL or other RDBMS), you might need to explicitly drop a constraint

by referring to its type, such as dropping a PRIMARY KEY, UNIQUE, or CHECK constraint using DROP PRIMARY KEY or similar syntax.

c. The names of constraints are user-defined in most cases, but they are automatically generated by the system for certain types

of constraints (e.g., primary keys, foreign keys).

--Answer 5

Consequences of Violating Constraints

When attempting to insert, update, or delete data that violates constraints, the following consequences occur:

Operation Fails: The database engine will reject the operation (insert, update, or delete) and will not modify the data.

Error Message: The database will return an error message indicating the violation, providing insight into what constraint was violated.

Data Integrity Loss: Violating constraints can lead to inconsistent or invalid data, which can compromise the integrity and correctness

of the database.

Transaction Rollback: If the constraint violation occurs within a transaction, the database may automatically roll back the entire

transaction to maintain data integrity.

Example of Violating Constraints

1. Inserting a Duplicate Value (Unique Constraint Violation):

If a UNIQUE constraint exists on a column, attempting to insert a duplicate value will violate the constraint.

INSERT INTO employees (employee_id, email)

VALUES (1, 'john.doe@example.com');

If the email column has a UNIQUE constraint and the same email already exists, the error message will be:

Error Message:

ERROR: duplicate key value violates unique constraint "employees_email_key"

DETAIL: Key (email)=(john.doe@example.com) already exists.

2. Inserting a NULL into a NOT NULL Column:

If a column has a NOT NULL constraint, inserting a NULL value will violate the constraint.

INSERT INTO employees (employee_id, email)

VALUES (2, NULL);

Error Message:

ERROR: null value in column "email" violates not-null constraint

DETAIL: Failing row contains (2, NULL).

3. Foreign Key Violation (Referential Integrity Violation):

If a FOREIGN KEY constraint is violated by inserting a value that doesn't exist in the referenced table, the operation will fail.

INSERT INTO employees (employee_id, department_id)

VALUES (3, 999); -- department_id 999 doesn't exist in the departments table

Error Message:

ERROR: insert or update on table "employees" violates foreign key constraint "fk_department_id"

DETAIL: Key (department_id)=(999) is not present in table "departments".

*/

-- Answer 6

/* To modify the products table based on the given requirements, you need to:

Add a Primary Key Constraint to the product_id column.

Set a Default Value for the price column to 50.00.

Steps:

Add Primary Key: Use the ALTER TABLE command to add a primary key to the product_id column.

Set Default Value: Use the ALTER TABLE command to set a default value for the price column.

ALTER TABLE products

ADD CONSTRAINT pk_product_id PRIMARY KEY (product_id);

ALTER TABLE products

ALTER COLUMN price SET DEFAULT 50.00;

The first command adds a primary key constraint to the product_id column, ensuring that each product has a unique identifier.

The second command sets a default value of 50.00 for the price column, meaning if no value is provided when inserting a product,

the price will automatically be set to 50.00.

These changes will ensure data integrity and default behavior for the products table.

*/

-- Answer 7

/*

```
SELECT
   Students.student_name,
   Classes.class_name
FROM
   Students
INNER JOIN
   Classes
ON
   Students.class_id = Classes.class_id;
*/

-- Answer 8
/*
SELECT
   Orders.order_id,
   Customers.customer_name,
   Products.product_name
FROM
   Products
LEFT JOIN
   Orders ON Products.order_id = Orders.order_id
LEFT JOIN
   Customers ON Orders.customer_id = Customers.customer_id;
*/

-- Answer 9
/*
SELECT
   Products.product_name,
```

```
    SUM(Sales.amount) AS total_sales_amount
FROM
    Sales
INNER JOIN
    Products ON Sales.product_id = Products.product_id
GROUP BY
    Products.product_name;
*/

-- Answer 10
/*
SELECT
    Orders.order_id,
    Customers.customer_name,
    Order_Details.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;
*/

-- SQL Commands
-- Answer 1
/* The "Maven Movies" database includes tables like Movies, Directors, Actors, and Movie_Actors,
here’s how the primary keys and foreign keys might be set up:
Movies Table
Primary Key: movie_id (each movie has a unique identifier).
```

Foreign Key: director_id (references director_id in the Directors table to indicate the director of each movie).

Directors Table

Primary Key: director_id (unique identifier for each director).

No foreign keys in this table, as it is independent.

Actors Table

Primary Key: actor_id (unique identifier for each actor).

No foreign keys in this table, as it is independent.

Movie_Actors Table (associative table for many-to-many relationship between Movies and Actors)

Primary Key: A composite key of movie_id and actor_id (each unique pair represents a specific actor in a specific movie).

Foreign Keys:

movie_id (references movie_id in the Movies table).

actor_id (references actor_id in the Actors table).

Differences between Primary Keys and Foreign Keys:

Uniqueness: A primary key uniquely identifies each record in a table, while a foreign key can have duplicate values as it is used

to link records between tables.

Nullability: Primary keys cannot contain NULL values, whereas foreign keys can, depending on the database design and requirements.

Relationship Enforcement: A primary key defines the unique identity of a record, whereas a foreign key enforces referential

integrity by linking tables based on primary key values in another table.

*/

-- Answer 2

SELECT * FROM actor;

-- Answer 3

```
SELECT * FROM Customer;

-- Answer 4
SELECT DISTINCT country FROM Country;

-- Answer 5
SELECT * FROM Customer WHERE active = 1;

-- Answer 6
SELECT rental_id FROM Rental WHERE customer_id = 1;

-- Answer 7
SELECT * FROM Film WHERE rental_duration > 5;

-- Answer 8
SELECT COUNT(*) FROM Film WHERE replacement_cost > 15 AND replacement_cost < 20;

-- Answer 9
SELECT COUNT(DISTINCT first_name) FROM Actor;

-- Answer 10
SELECT * FROM Customer LIMIT 10;

-- Answer 11
SELECT * FROM Customer WHERE first_name LIKE 'b%' LIMIT 3;

-- Answer 12
SELECT title FROM Film WHERE rating = 'G' LIMIT 5;
```

```
-- Answer 13
SELECT * FROM Customer WHERE first_name LIKE 'a%';

-- Answer 14
SELECT * FROM Customer WHERE first_name LIKE '%a';

-- Answer 15
SELECT city FROM City WHERE city LIKE 'a%' AND city LIKE '%a' LIMIT 4;

-- Answer 16
SELECT * FROM Customer WHERE first_name LIKE '%NI%';

-- Answer 17
SELECT * FROM Customer WHERE first_name LIKE '_r%';

-- Answer 18
SELECT * FROM Customer WHERE first_name LIKE 'a%' AND LENGTH(first_name) >= 5;

-- Answer 19
SELECT * FROM Customer WHERE first_name LIKE 'a%o';

-- Answer 20
SELECT * FROM Film WHERE rating IN ('PG', 'PG-13');

-- Answer 21
SELECT * FROM Film WHERE length BETWEEN 50 AND 100;

-- Answer 22
SELECT * FROM Actor LIMIT 50;
```

```
-- Answer 23
SELECT DISTINCT film_id FROM Inventory;

-- SQL Function
-- Answer 1
SELECT COUNT(*) AS total_rentals FROM Rental;

-- Answer 2
SELECT AVG(rental_duration) AS average_rental_duration FROM Film;

-- Answer 3
SELECT UPPER(first_name) AS first_name_upper, UPPER(last_name) AS last_name_upper FROM
Customer;

-- Answer 4
SELECT rental_id, MONTH(rental_date) AS rental_month FROM Rental;

-- Answer 5
SELECT customer_id, COUNT(*) AS rental_count
FROM Rental
GROUP BY customer_id;

-- Answer 6
SELECT payment_id, SUM(amount) AS total_revenue
FROM Payment
GROUP BY payment_id;

-- Answer 7
```

```
SELECT fc.category_id, COUNT(*) AS rental_count
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
GROUP BY fc.category_id;

-- Answer 8
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;

-- SQL Joins
-- Answer 1
SELECT f.title AS movie_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;

-- Answer 2
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';

-- Answer 3
```

```
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;

-- Answer 4
SELECT c.first_name, c.last_name, f.title AS movie_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, f.title;

-- Advanced Joins and Group By
-- Answer 1
SELECT f.title AS movie_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;

-- Answer 2
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;

-- SQL Windows Function
-- Answer 1
SELECT c.customer_id, c.first_name, c.last_name, SUM(p.amount) AS total_spent,
    RANK() OVER (ORDER BY SUM(p.amount) DESC) AS spending_rank
FROM Customer c
JOIN Payment p ON c.customer_id = p.customer_id
GROUP BY c.customer_id;

-- Answer 2
SELECT f.film_id, 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;

-- Answer 3
SELECT f.film_id, f.title, f.length, AVG(f.rental_duration) AS average_rental_duration
FROM Film f
GROUP BY f.length;

-- Answer 4
SELECT fc.category_id, f.film_id, f.title, COUNT(r.rental_id) AS rental_count,
```

```
    RANK() OVER (PARTITION BY fc.category_id ORDER BY COUNT(r.rental_id) DESC) AS rental_rank
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
HAVING rental_rank <= 3;

-- Answer 5
WITH CustomerRentalCounts AS (
  SELECT customer_id, COUNT(*) AS total_rentals
  FROM Rental
  GROUP BY customer_id
),
AvgRentals AS (
  SELECT AVG(total_rentals) AS avg_rentals
  FROM CustomerRentalCounts
)
SELECT c.customer_id, c.first_name, c.last_name, crc.total_rentals,
    (crc.total_rentals - ar.avg_rentals) AS rental_difference
FROM Customer c
JOIN CustomerRentalCounts crc ON c.customer_id = crc.customer_id
CROSS JOIN AvgRentals ar;

-- Answer 6
SELECT DATE_FORMAT(p.payment_date, '%Y-%m') AS month, SUM(p.amount) AS monthly_revenue
FROM Payment p
GROUP BY month
ORDER BY month;
```

```
-- Answer 7
WITH CustomerSpending AS (
   SELECT customer_id, SUM(amount) AS total_spent
   FROM Payment
   GROUP BY customer_id
),
SpendingThreshold AS (
   SELECT PERCENTILE_CONT(0.8) WITHIN GROUP(ORDER BY total_spent DESC) AS
top_20_percent_threshold
   FROM CustomerSpending
)
SELECT c.customer_id, c.first_name, c.last_name, cs.total_spent
FROM Customer c
JOIN CustomerSpending cs ON c.customer_id = cs.customer_id
JOIN SpendingThreshold st ON cs.total_spent >= st.top_20_percent_threshold;

-- Answer 8
SELECT fc.category_id, COUNT(r.rental_id) AS rental_count,
     SUM(COUNT(r.rental_id)) OVER (PARTITION BY fc.category_id ORDER BY COUNT(r.rental_id)) AS
running_total
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;

-- Answer 9
WITH CategoryAverages AS (
```

```
    SELECT fc.category_id, AVG(rental_count) AS avg_rentals
    FROM (SELECT fc.category_id, f.film_id, 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
        GROUP BY fc.category_id, f.film_id) AS CategoryFilmCounts
    GROUP BY fc.category_id
)
SELECT f.title, fc.category_id, COUNT(r.rental_id) AS rental_count
FROM Film f
JOIN Film_Category fc ON f.film_id = fc.film_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, fc.category_id
HAVING COUNT(r.rental_id) < (SELECT avg_rentals FROM CategoryAverages WHERE fc.category_id =
category_id);


-- Answer 10
SELECT DATE_FORMAT(payment_date, '%Y-%m') AS month, SUM(amount) AS monthly_revenue
FROM Payment
GROUP BY month
ORDER BY monthly_revenue DESC
LIMIT 5;


--  Normalisation and CTE
-- Answer 1
/*  Example Table: The Address table could violate 1NF if there were multiple phone numbers stored in a
single phone field for an address.
```

Normalization Process: To achieve 1NF, ensure each field contains only atomic values (one value per field). In this case, create a new table,

Address_Phone, with fields address_id and phone_number, where each row represents a single phone number associated with an address.

This removes the possibility of storing multiple values in a single phone column.

*/

-- Answer 2

/* Example Table: The Payment table.

2NF Check: To check if Payment is in 2NF, identify if it has a composite primary key (e.g., if customer_id and rental_id were a composite key).

If a column is partially dependent on only one part of the composite key, it violates 2NF.

Normalization Process: Remove columns that are partially dependent on only part of the composite key, placing them in a new table. For instance,

if amount depends solely on customer_id, it should be moved to a Customer_Payments table to remove partial dependencies.

*/

-- Answer 3

/* Example Table: The Rental table may have transitive dependencies if it includes both store_id and staff_id.

Transitive Dependency: If staff_id implies the store_id because each staff member is associated with only one store, then store_id is

transitively dependent on staff_id.

Normalization Process: Remove the transitive dependency by separating store_id from the Rental table. Create a Staff_Store table with staff_id and

store_id to ensure that Rental only contains direct dependencies.

*/

-- Answer 4

/* Example Table: The Inventory table.

Step 1: Check for unnormalized data. Ensure that all fields are atomic.

Step 2: Achieve 1NF by separating any non-atomic fields.

Step 3: Ensure 2NF by identifying partial dependencies. For instance, if Inventory had a composite key of film_id and store_id with

additional details that only depend on film_id, split those into separate tables to remove partial dependencies.

*/

```
-- Answer 5
WITH ActorFilmCount 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
)
SELECT first_name, last_name, film_count FROM ActorFilmCount;

-- Answer 6
WITH FilmLanguage AS (
    SELECT f.title, l.name AS language_name, f.rental_rate
    FROM Film f
    JOIN Language l ON f.language_id = l.language_id
)
SELECT * FROM FilmLanguage;

-- Answer 7
WITH CustomerRevenue AS (
    SELECT customer_id, SUM(amount) AS total_revenue
    FROM Payment
    GROUP BY customer_id
```

```
)
SELECT * FROM CustomerRevenue;

-- Answer 8
WITH FilmRanking AS (
   SELECT film_id, title, rental_duration,
        RANK() OVER (ORDER BY rental_duration DESC) AS duration_rank
   FROM Film
)
SELECT * FROM FilmRanking;

-- Answer 9
WITH FrequentRenters 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, fr.rental_count
FROM Customer c
JOIN FrequentRenters fr ON c.customer_id = fr.customer_id;

-- Answer 10
WITH MonthlyRentals AS (
   SELECT DATE_FORMAT(rental_date, '%Y-%m') AS month, COUNT(*) AS rental_count
   FROM Rental
   GROUP BY month
)
SELECT * FROM MonthlyRentals;
```

```
-- Answer 11
WITH ActorPairs AS (
    SELECT fa1.actor_id AS actor1_id, fa2.actor_id AS actor2_id, fa1.film_id
    FROM Film_Actor fa1
    JOIN Film_Actor fa2 ON fa1.film_id = fa2.film_id AND fa1.actor_id < fa2.actor_id
)
SELECT a1.first_name AS actor1_first_name, a1.last_name AS actor1_last_name,
       a2.first_name AS actor2_first_name, a2.last_name AS actor2_last_name, film_id
FROM ActorPairs ap
JOIN Actor a1 ON ap.actor1_id = a1.actor_id
JOIN Actor a2 ON ap.actor2_id = a2.actor_id;

-- Answer 12
WITH RECURSIVE StaffHierarchy AS (
    SELECT staff_id, first_name, last_name, reports_to
    FROM Staff
    WHERE staff_id = manager_id  -- Replace with the specific manager's ID
    UNION ALL
    SELECT s.staff_id, s.first_name, s.last_name, s.reports_to
    FROM Staff s
    JOIN StaffHierarchy sh ON s.reports_to = sh.staff_id
)
SELECT * FROM StaffHierarchy;
```