SQL Assignment Theory part

'''2. Explain the purpose of constraints and how they help maintain data integrity in a database. Provide examples of common types of constraints'''
'''Constraints are rules applied to database tables to ensure the accuracy, consistency, and integrity of the data. They restrict the types of data that can be inserted, updated, or deleted in a table, preventing invalid or inconsistent data from being stored.
The main purposes of constraints are:
Data Integrity: Ensure the data entered into a database is accurate and reliable.
Prevent Errors: Avoid mistakes caused by human input or faulty applications.
Maintain Relationships: Enforce relationships between tables using keys and referential integrity.
Optimize Performance: Improve query performance by eliminating unnecessary data checks.
Common Types of Constraints
Here are some of the most commonly used database constraints:
1-Primary Key Constraint
2-Foreign Key Constraint
3-Unique Constraint
4-Check Constraint
5-Not Null Constraint
6-Default Constraint '''
'''3.Why would you apply the NOT NULL constraint to a column? Can a primary key contain NULL values? Justify
your answer.'''
''' The NOT NULL constraint is applied to a column to ensure that it cannot store NULL values.
This is particularly useful when a column's data is essential for the integrity or functionality of the
 application.
 No, a Primary Key cannot contain NULL values.
Justification:
1- Uniqueness and Identification
2-Database Integrity
3-Indexing
'''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.'''
''' you can use the ALTER TABLE statement to add or remove constraints from an existing table.
The process may vary depending on the type of constraint.
Adding constraint

```
ALTER TABLE employee
MODIFY COLUMN Age int NOT NULL;
```

Removing constraint

```
ALTER TABLE Employees
DROP CONSTRAINT CHK_Age;
```

'''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.'''
'''When attempting to insert, update, or delete data that violates constraints in MySQL,

the database management system (DBMS) will prevent the action and return an error. Constraints are enforced to maintain data integrity and ensure that the database remains consistent.
Here are the typical consequences of violating different types of constraints:
insert into employee Values (101, "Abha","ayadav989@gmail.com",50000);
Consequence: You cannot insert duplicate values or NULL in a primary key column.
ERROR 1062 (23000): Duplicate entry '101' for key 'PRIMARY'''
#1-Identify the primary keys and foreign keys in maven movies db. Discuss the differences
''' SELECT actor, actor_id
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
WHERE TABLE_SCHEMA = 'mavenmovies'
AND CONSTRAINT_NAME = 'PRIMARY';

Primary Key	Foreign Key
Uniquely identifies each row in a table.	Establishes a relationship between tables.
Cannot contain NULL values.	Can contain NULL if the relationship allows it.
Only one primary key is allowed per table.	Multiple foreign keys can exist in a table.
Automatically enforces uniqueness.	Does not enforce uniqueness in the child table.
Typically used for table identification.	Used for maintaining referential integrity.

'''1. First Normal Form (1NF):
a. Identify a table in the Sakila database that violates 1NF. Explain how you would normalize it to achieve 1NF.'''
''' CREATE TABLE film_text (
  film_id SMALLINT NOT NULL,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  PRIMARY KEY (film_id)
);
The description column often contains multiple pieces of information (e.g., genre, plot details, actors, etc.) stored as a single text field.
Storing multiple values in one column violates 1NF since it’s not atomic.
 Steps to Normalize the Table to 1NF
To achieve 1NF, follow these steps:
1. Split the Data into Separate Tables
Create separate tables for distinct types of information.
Remove the non-atomic description field and store the details in relevant tables like film_category, film_actor, or film_description.
2. Ensuring Atomic Data
Each column now contains a single value instead of combined information.
Queries can be performed efficiently using JOIN operations.
'''2. Second Normal Form (2NF):
a. Choose a table in Sakila and describe how you would determine whether it is in 2NF.
If it violates 2NF, explain the steps to normalize it.'''
''' Second Normal Form (2NF) builds on First Normal Form (1NF) and requires that:
The table must be in 1NF (no repeating groups, atomic data).
No partial dependency should exist. This means no non-prime attribute (a column that is not part of a primary key) should depend only on a part of a composite primary key.
Step 1: Check for 2NF Compliance
The table is in 1NF since all data is atomic, and there are no repeating groups.
However, to satisfy 2NF, we must ensure that all non-key attributes are fully dependent on the entire primary key, not just a part of it.

Analysis:
The last_update column is not dependent on both actor_id and film_id.
It is just a general timestamp, not specifically related to the relationship between an actor and a film.
This means the table violates 2NF.
Step 2: Normalize to 2NF
Solution:
Remove the last_update column from the film_actor table.
Create a separate table to store timestamps and general film updates.'''
'''3. Third Normal Form (3NF):
a. Identify a table in Sakila that violates 3NF. Describe the transitive dependencies
present and outline the steps to normalize the table to 3NF.'''
'''Understanding Third Normal Form (3NF)
Third Normal Form (3NF) is achieved when:
The table is in Second Normal Form (2NF).
There are no transitive dependencies.
Transitive Dependency means:
A non-prime attribute (not part of the primary key) depends on another non-prime attribute instead of depending directly on the primary key.

```
CREATE TABLE customer (
  customer_id SMALLINT NOT NULL,
  store_id TINYINT NOT NULL,
  first_name VARCHAR(45) NOT NULL,
  last_name VARCHAR(45) NOT NULL,
  email VARCHAR(50),
  address_id SMALLINT NOT NULL,
  active BOOLEAN NOT NULL,
  create_date DATETIME NOT NULL,
  last_update TIMESTAMP,
  PRIMARY KEY (customer_id)
);
```

Transitive Dependency
The table has an address_id column, which references the address of the customer.
The address contains city_id, and further, city_id refers to the city in the city table, which also contains country_id to identify the country.
Therefore, the customer's city and country are indirectly determined through the address.
This creates a transitive dependency:
Steps to Normalize the Table to 3NF
Step 1: Remove Transitive Dependencies
Create separate tables for Address, City, and Country instead of storing redundant data.
Link the customer table to the Address table using address_id.'''
'''4. Normalization Process:
a. Take a specific table in Sakila and guide through the process of normalizing it from the initial
unnormalized form up to at least 2NF.'''
'''Normalization Process: From Unnormalized Form to 2NF
Let's go through the normalization process step by step using a specific example from the Sakila database.
Step 1: Choose a Table to Normalize
We'll work with a hypothetical Unnormalized Table (UNF) that combines data from multiple tables for easy understanding.

Issues in UNF:
Repeating Groups: John Doe is listed multiple times for different rentals.
Redundant Data: Film titles and categories are duplicated.
No Primary Key: There's no clear unique identifier for each record.
Step 2: Convert to First Normal Form (1NF)
1NF Rules:
Ensure each column contains atomic (indivisible) values.
Remove repeating groups.
Identify a Primary Key.
Convert to 1NF:
Each value is now atomic (e.g., separate columns for customer and film details).
Rental_ID is assigned as the Primary Key.
Step 3: Convert to Second Normal Form (2NF)
2NF Rules:
Ensure it is in 1NF.
Remove Partial Dependencies.
Every non-key column should depend on the whole primary key.
Issue in 1NF:
Customer_Name depends only on the Rental_ID but not on the Film data.
Film_Category depends on the Film_Title, not directly on the Rental_ID.
Solution:
Create three tables to remove partial dependencies.