1.Why are duplicate tuples not allowed in a relation?

Duplicate tuples are not allowed in a relation because they create redundancy of data base which makes the data processing like querying, inserting, deleting, and updating more complicated and time-consuming. In other words, they violate the requirements of the constraints of relational integrity, which ensure that the data in a database is consistent, accurate, and valid. Therefore, avoiding duplicate tuples is essential for maintaining the quality and efficiency of a database.

2.Discuss the various reasons that lead to the occurrence of NULL values in relations.

Null values in a relation can occur due to various reasons. One of the main reasons is the absence of information, where the value of an attribute is not known or not applicable to a certain tuple. Another reason is the presence of optional attributes, where certain attributes may not be relevant or necessary for some tuples. Additionally, null values can occur when data is being transferred from one database to another, and the receiving database does not have the required information. It is important to note that inserting null values into the primary key of a new tuple violates entity integrity, as every table must have a primary key without any null values.

3.Discuss the entity integrity and referential integrity constraints. Why is each considered important?

Entity integrity is a type of integrity constraint that ensures that each row in a table has a unique identifier, usually a primary key, which distinguishes it from all other rows. This constraint is important because it ensures that no two rows in a table are identical, and it facilitates updates and deletes to individual rows. Violating entity integrity can cause data redundancy and inconsistencies, leading to incorrect results when querying the database.

Referential integrity is another type of integrity constraint that ensures the consistency between related tables in a database. It ensures that foreign key values in a table match the primary key values in another related table, and that no orphan records exist. This constraint is important because it prevents data loss and inconsistencies when updating or deleting related records. If referential integrity is violated, it can lead to data inconsistencies and incorrect results when querying the database.

Both entity integrity and referential integrity are essential for maintaining the accuracy and consistency of data in a database. They prevent data redundancy, inconsistencies, and data loss, which can lead to incorrect results when querying the database. Therefore, it is important to ensure that these constraints are properly enforced in any database design.

4.Define foreign key. What is this concept used for? Explain with an example.

A foreign key is a column or set of columns in a table that references the primary key of another table. It is used to link data between tables in a relational database. The purpose of using a foreign key is to ensure data consistency and maintain referential integrity between related tables.

For example, consider a database for a library. There could be two tables, one for storing information about books and another for storing information about borrowers. The books table could have a primary key called book_id, and the borrowers table could have a primary key called borrower_id. To track which books have been borrowed by which borrowers, a new table called loans could be created. This table would have columns for book_id and borrower_id, both of which would be foreign keys referencing the book_id and borrower_id columns in their respective tables. This would allow the loans table to link the book and borrower information together and ensure that only valid book and borrower IDs are entered into the table, maintaining data accuracy and consistency.

5.What is a transaction? How does it differ from an Update operation?

A transaction is a logical unit of work in a database that includes one or more database access operations and must maintain ACID properties such as atomicity, consistency, isolation, and durability. It represents real-world events and includes one or more database access operations executed as a single unit in a DBMS. An update operation, on the other hand, is a specific database operation that modifies data in a database. While a transaction can include an update operation, it is not limited to this type of operation and can include multiple operations that are executed as a single unit to ensure data consistency and integrity in the database. In contrast, an update operation is a single operation that modifies existing data in a database.

6. Suppose that each of the following Update operations is applied directly to the database state shown in the following figure. Discuss all integrity constraints violated by each operation, if any, and the different ways of enforcing these constraints.

In the employee table, the SSN (Social Security Number) attribute serves as the primary key, ensuring uniqueness for each employee record. Additionally, there is another attribute called super_ssn. If both the SSN and super_ssn are updated simultaneously, it would result in a violation of the primary key constraint and the unique key constraint.

Updating the SSN directly is not permissible because it is a unique key, and altering its value would compromise the integrity of the table's data.

In the department and department locations scenario, the dnumber attribute acts as a foreign key within the department locations table. This foreign key establishes a dependency on the primary key of the department table. Consequently, deleting a foreign key that references a primary key violates the foreign key constraint since it would leave the department locations table with an invalid reference.

In summary, attempting to update both the SSN and super_ssn in the employee table violates primary key and unique key constraints, while deleting a foreign key that references a primary key violates foreign key constraints.

11. Consider the AIRLINE relational database schema shown in Figure below, which describes a database for airline flight information. Each FLIGHT is identified by a Flight_number, and consists of one or more FLIGHT_LEGs with Leg_numbers 1, 2, 3, and so on. Each FLIGHT_LEG has scheduled arrival and departure times, airports, and one or more LEG_INSTANCEs—one for each Date on which the flight travels. FAREs are kept for each FLIGHT. For each FLIGHT_LEG instance, SEAT_RESERVATIONs are kept, as are the AIRPLANE used on the leg and the actual arrival and departure times and airports. An AIRPLANE is identified by an Airplane_id and is of a particular AIRPLANE_TYPE. CAN_LAND relates AIRPLANE_TYPEs to the AIRPORTs at which they can land. An AIRPORT is identified by an Airport_code. Consider an update for the AIRLINE database to enter a reservation on a particular flight or flight leg on a given date.

a. Give the operations for this update.

1.Check if the new seat is available for this particular leg. If not reject the operation

2. Update the existing info about seats and passengers for this leg.

3. Delete the previous reservation

b. What types of constraints would you expect to check?

The following constraints would be checked

1.The reservation should only go through if all the flight legs are available

2. The specified seat should not have been taken already for each leg for the reservation to be completed.

c. Which of these constraints are key, entity integrity, and referential integrity constraints, and which are not?

1.The availability of all flight legs is not applied through any of the mentioned constraint as It's a business constraint and not a database constraint.

2.The second operation is an entity integrity constraint as the seat should be unique for each flight leg reservation.

d. Specify all the referential integrity constraints that hold on the schema shown in Figure.

FLIGHT_LEG.FLIGHT_NUMBER → FLIGHT.FLIGHT_NUMBER

FLIGHT_LEG.DEPARTURE_AIRPORT_CODE → AIRPORT.AIRPORT_CODE

FLIGHT_LEG.ARRIVAL_AIRPORT_CODE → AIRPORT.AIRPORT_CODE

LEG_INSTANCE.(FLIGHT_NUMBER,LEG_NUMBER) → FLIGHT_LEG.(FLIGHT_NUMBER,LEG_NUMBER)

LEG_INSTANCE.DEPARTURE_AIRPORT_CODE → AIRPORT.AIRPORT_CODE

LEG_INSTANCE.ARRIVAL_AIRPORT_CODE → AIRPORT.AIRPORT_CODE

LEG_INSTANCE.AIRPLANE_ID → AIRPLANE.AIRPLANE_ID

FARES.FLIGHT_NUMBER → FLIGHT.FLIGHT_NUMBER

CAN_LAND.AIRPLANE_TYPE_NAME → AIRPLANE_TYPE.AIRPLANE_TYPE_NAME

CAN_LAND.AIRPORT_CODE → AIRPORT.AIRPORT_CODE

AIRPLANE.AIRPLANE_TYPE → AIRPLANE_TYPE.AIRPLANE_TYPE_NAME

SEAT_RESERVATION.(FLIGHT_NUMBER,LEG_NUMBER,DATE) → LEG_INSTANCE.(FLIGHT_NUMBER,LEG_NUMBER,DATE)