1. Discuss how NULLs are treated in comparison operators in SQL. How are NULLs treated when aggregate functions are applied in an SQL query? How are NULLs treated if they exist in grouping attributes?
   Answer:
   Null values in comparison operators: A NULL value in SQL is a marker that denotes an unknown or missing value. As an outcome, whenever NULL is compared, an unknown result is obtained and is denoted by NULL. When comparing an unknown age record (NULL) in a database with another unknown age record (NULL), SQL will not declare them to be the same. Since NULL is treated as an unknown, it yields NULL rather than true or false in the comparison.

   NULLs in Aggregate Functions: Aggregate functions in SQL neglect NULL values. Functions that do not take NULL values into account in their calculations include COUNT(), SUM(), AVG(), MIN(), and MAX(). However, COUNT(*) counts every row—including NULL rows—while COUNT(column) only counts values in the selected column that are not NULL. Suppose some individuals' heights are unknown (NULL) and that you are attempting to determine the average height of the group. These unknown values will not be included in the calculation using SQL. Therefore, only the heights that are known will be used to calculate the average height.

   NULLs in Grouping Attributes: SQL treats all NULL values as a single group when you use the GROUP BY clause. It means that any entries in the grouping column that contain NULL values will be included in the same group. SQL will combine all of the records with unknown cities into a single group if you're grouping data by city and some of the records have the city defined as unknown (NULL). As a result, you will have one group for the unknown city and another group for each recognized city.

2. Define the five basic relational algebra operations. Define the Join, Intersection, and Division operations in terms of these five basic operations with examples.
   Answer:
   a. **Select (σ):** Using this method, a subset of the tuples that satisfy a selection criteria may be chosen from a relation. It is represented by the symbol σConditionR, where R is a relation and condition is a formula for propositional logic that may include and, or, and not as connectors. These could involve contrasting different relational qualities. Suppose we are in a relationship. Students' Name, Age, and Major attributes. If we want to find all students who are majoring in 'Computer Science', we could use the select operation like this: $\sigma_{Major="Computer\ Science"}(STUDENTS)$. This will return a new relation with all tuples (rows) where the Major is 'Computer Science'.

b. **Project (π):** This operation helps in projecting a particular field from a relation. The syntax for this is:, $\pi_{A1,A2,.....,An}R$ where A1, A2, …, An are attributed names of relation R. Suppose we want to project or show the names of the Students only, we could use the project operation like this: $\pi_{Name}(STUDENTS)$. This will return a new relation with a single attribute, Name, containing the names of all students.

c. **Union (∪):** Tuples from two relations are combined and duplicates are eliminated using the union operation. It is necessary for the relations to be union-compatible, which means that they have the same number of attributes and that the domains of their corresponding attributes match. To get a list of all students, we can use the union operation undergraduates ∪ graduates if we have two relations, graduates and undergraduates, that have the same properties. After removing any duplicates, this produces a new relation with all of the tuples from the grads and undergraduates.

d. **Set Difference (-):** Tuples that are in the first relation but not the second relation are included in the result of the set difference operation, which operates on two relations. We might use the set difference operation in the following way to find all students who are not graduates: Undergraduates - Graduates. A new relation with all the tuples in Undergraduates but not in Graduates will be produced as a result of this.

e. **Cartesian Product (x):** Tuples from two relations are combined in a combinatorial way using the Cartesian product operation. The resultant relation will contain (n+m) characteristics if R and S are two relations with n and m attributes, respectively. To identify every conceivable combination of students and courses, we may use the Cartesian product operation, which looks like this: Students x Courses, if we have a relation called Courses with the values CourseID and CourseName. With each combination of tuples from Students and Courses, this will provide a new relation.

f. **Join (⋈):** The syntax for this operation is denoted as $R_{condition}S$ It helps in combining tuples from two relations R and S based on a given join condition. For example, we could use the join operation in a particular way to locate all orders made by a certain customer if we had two relations, Orders and Customers: $Orders_{Orders.CustomerID=Customers.CustomerID}Customers.$ When the CustomerID in Orders and the CustomerID in Customers match, a new relation will be returned.

g. **Intersection:** The relation that results from the intersection operation, represented as R ∩ S, is formed up of the common tuples that are shared by R and S. The intersection operation can be used in the following way to find all undergraduates who received a scholarship: Undergraduates ∩ ScholarshipRecipients. This requires that we have two relations, Undergraduates and ScholarshipRecipients. All of the tuples that are in both the Undergraduates and ScholarshipRecipients tables will be returned in a new relation as a result.

h. **Division:** The divide method is a little more complicated. When looking for tuples in a relation R1(A, B) that are connected to every tuple in a different relation R2(B), we employ it. Although the statement is more complicated and requires several applications of the other operations, it may be written using the fundamental operations. For example, we could use the division operation to locate all suppliers that provide every part if we had two relations: SupplierParts with attributes SupplierName and PartNumber, and Parts with attribute PartNumber. To get the intended outcome, this action would require several applications of the other operations, making it complex and missing a clear symbol.

3. Discuss the differences between the five Join operations: Theta join, Equijoin, Natural join, Outer join, and Semijoin. Give examples to illustrate your answer.

a. **Natural Join**: An Equijoin type known as a "natural join" bases the join predicate on all common columns shared by the two tables. The Natural connect handles columns that have the same names but unique data types as separate columns. Suppose that the DepartmentID field is shared by the Employee and Department tables. To identify all employee-department combinations where the

DepartmentID in both tables matches, a natural join could be performed. Based on the DepartmentID, this would effectively connect each employee with their own department.

b. **Outer Join:** All of the rows from one table and the matching rows from another are returned by an outer join. On the side of the table without a match, the result is NULL if no match is discovered. Left and Right are the two forms of outer joins. Every employee and their departments (if any) can be found using an Outer join, more especially a Left Outer join, using the same Employee and Department tables. An employee's department fields would be NULL when they are not a part of a department.

c. **Semi Join:** When there is a match in the second table, a semi-join collects rows from the first table. Only the rows from the first table are returned in the event that the second table has a match; the matching rows from the second table are not returned. Once more, a semi-join could be used to find every employee who is a member of a department using the Employee and Department tables. Only the workers who have a matching DepartmentID in the Department table would be returned; no further information about the department would be provided.

4. There are different types of join operations that can be used to retrieve data, based on different relations. Describe the relation between theta and equal join.

Theta Join: A theta join allows combination of columns from two tables in any way you choose, including "=", "\", ">", "<=", ">=", "<>", and so on. The formal theory of relational databases uses θ (theta) to represent an arbitrary comparison operator, which is where the name "theta" comes from. To combine values from two tables, for example, when the value in the first table's "age" column is smaller than the value in the second table's "age" column, you may use a theta join.

Equi Join (Equal Join): An equi join is a particular kind of theta join in which the comparison operator "=" is the only one utilized. In other words, it combines rows in which the value in the first table's specified column is equal to the value in the second table's defined column. In practical use, this kind of join operation is the most popular.

5.

5. Describe the relations that would be produced by the following relational algebra operations:

(a) $\Pi_{hotelNo}(\sigma_{price > 50}(Room))$

(b) $\sigma_{Hotel.hotelNo = Room.hotelNo}(Hotel \times Room)$

(c) $\Pi_{hotelName}(Hotel \bowtie_{Hotel.hotelNo = Room.hotelNo}(\sigma_{price > 50}(Room)))$

(d) $Guest \rtimes (\sigma_{dateTo \geq '1\text{-}Jan\text{-}2007'}(Booking))$

(e) $Hotel \triangleright_{Hotel.hotelNo = Room.hotelNo}(\sigma_{price > 50}(Room))$

(f) $\Pi_{guestName, hotelNo}(Booking \bowtie_{Booking.guestNo = Guest.guestNo} Guest) \div \Pi_{hotelNo}(\sigma_{city = 'London'}(Hotel))$

(a) This method projects the hotelNo characteristic after selecting rooms with prices higher than 50. The hotel numbers of the hotels with rooms cost over fifty will be contained in the resulting relation.

(b) Tuples where the hotelNo attribute matches in both relations are selected by performing a Cartesian product of the Hotel and Room relations. All of the qualities from the Hotel and the Room for which the same hotelNo exists will be included in the resultant relation.

(c) The rooms where the price is more than 50 are initially chosen in this operation, after which a natural join with the Hotel relation on hotelNo is completed and the hotelName attribute is projected. The names of hotels with rooms costing more than fifty will be included in the resultant connection.

(d) This method performs a natural join with the Guest relation after first choosing bookings where the dateTo is greater than "1-Jan-2007." For reservations made after "1-Jan-2007," the resultant relation will include all qualities from both the guest and the booking.

(e) This operation performs a natural join with the Hotel relation on hotelNo after first choosing rooms whose price is more than 50. For rooms costing more than fifty, the resultant relation will include every attribute from the hotel as well as the room.

(f) This operation first selects hotels in London, then projects guestName and hotelNo using the outcome of another operation that connects Booking and Guest on guestNo. The names of the guests and hotel numbers for reservations made at London-based hotels will be contained in the resultant relation.

Q6. Answer

a. **Relational algebra:**

$\pi_{empNo,fName,lName,address,DOB,sex,position,deptNo}(Employee)$

Tuple relational calculus:

{t|Employee(t)}

Domain relational calculus:

{t.fName,t.lName,t.address,t.DOB,t.sex,t.position,t.deptNo|Employee(t)}

b. **Relational algebra:**

$\sigma_{sex='female' \wedge YEAR\ (DOB) > 1990}(Employee)$

Tuple relational calculus:

{t|Employee(t)∧t.sex='female'∧YEAR(t.DOB)>1990}

Domain relational calculus:

{t.fName,t.lName,t.address,t.DOB,t.sex,t.position,t.depthNo|Employee(t)∧ t.sex='female'∧YEAR(t.DOB)>1990}

c. **Relational algebra:**

$\sigma_{position\neq'manager' \wedge salary>1500}(Employee)$

Tuple relational calculus:

{t|Employee(t)∧t.position≠'manager'∧t.salary>1500

Domain relational calculus:

d. **Relational algebra:**

$\pi_{fName,lName,address}(Employee \bowtie_{Employee.deptNo=Department.deptNo \wedge Department.deptName='IT'} Department)$

Tuple relational calculus:

$\{t|Employee(t) \wedge \exists d(Department(d) \wedge d.deptName = 'IT' \wedge t.deptNo = d.deptNo)\}$

Domain relational calculus:

$\{t.fName, t.lName, t.address|Employee(t) \wedge \exists d(Department(d) \wedge d.deptName = 'IT' \wedge t.deptNo = d.deptNo)\}$

e. **Relational algebra:**

$\pi_{fName,lName,address}(Employee \bowtie_{Employee.empNo=WorksOn.empNo \wedge WorksOn.projNo='SCCS'} WorksOn)$

Tuple Relational Calculus:

{t|Employee(t)∧∃ω(WorksOn(ω) ∧ ω.projNo='SCCS'∧t.empNo=ω.empNo)}

Domain Relational Calculus:

{t.fName,t.lName|Employee(t)∧∃ω(WorksOn(ω) ∧ ω.projNo='SCCS'∧t.empNo=ω .empNo)}

f. **Relational algebra:**

$\pi_{fName,lName}(\sigma_{position='manager' \wedge retirementYear=CURRENT\_YEAR}(Employee))$

Tuple Relational Calculus:

{t|t∈Employee∧t.position='manager'∧t.retirementYear=CURRENT_YEAR}

Domain Relational Calculus:

{<fName,lName,...)=’manager’∧
retirementYear(fName,lName,...)=CURRENT_YEAR}

g. Answers:

h. Relational algebra:

COUNT($\sigma_{sex='female''\wedge position='manager'}(Employee)$)

Tuple Relational Calculus:

{t|Employee(t)∧t.position=’manager’∧t.sex=’female’}

Domain Relational Calculus:

{t.fName,t.lName|Employee(t)∧t.position=’manager’∧t.sex=’female’}

i. Relational algebra:

$\sigma_{deptName='IT''}(Project)$

Tuple relational calculus:

{t|Project(t)∧ ∃d(Department(d)∧d.deptName=’IT’∧ $t.deptNo$=d.deptNo)}

Domain relational calculus:

{t.projName|Project(t)∧∃d(Department(d)∧d.deptName=’IT’∧t.deptNo=d.deptNo)}

j. Relational algebra:

$\pi_{fName,lName,position,sex,deptNo}(\sigma_{project\neq'manager'\wedge project\neq'supervisor''}(Employee))$

Tuple relational calculus:

{t|Employee(t)∧t.position≠’manager’∧t.position≠’supervisor’}

Domain Relational calculus:

{t.fName,t.lName,t.position,t.sex,t..deptNo|Employee(t)∧t.position≠’manager’∧
t.position≠’supervisor’

Q7. Answer

1. Horse:
    a. Attributes: horseID, horseName, dateBought, purchasePrice, regNum, gender
    b. Primary Key: horseID
2. Owner:
    a. Attributes: ownerID, ownerName, address
    b. Primary Key: ownerID
3. Stable:
    a. Attributes: stableID, stableName, location
    b. Primary Key: stableID
4. Person:
    a. Attributes: personID, personName, address
    b. Primary Key: personID
5. Trainer/Jockey:
    a. Attributes: trainerJockeyID, trainerName, jockeyName
    b. Primary Key: trainerJockeyID
6. Entry:
    a. Attributes: entryID, horseID, raceID, position
    b. Primary Key: entryID
    c. Foreign Keys: horseID references Horse, raceID references RaceSchedule
7. RaceSchedule:
    a. Attributes: raceID, raceDate, trackID
    b. Primary Key: raceID
    c. Foreign Key: trackID references Track
8. Track:
    a. Attributes: trackID, trackName, location
    b. Primary Key: trackID

Relationship:

1. ownedBy(regNum, pName): The relationship between "Horse" and "Owner" is represented by this relationship. "regNum" and "pName" provide a unique identifier for each connection.

2. worksFor(pId, sId): This connection illustrates how "Person" and "Stable" are related to each other. "pId" and "sId" provide a unique identity for each connection.
3. contact(sId, pId): The "contact" relationship between "Stable" and "Person" is represented by this relationship. "sId" and "pId" provide a unique identification for each connection.
4. trainer and jockey have a "trainedBy" relationship, which is represented by the relationship trainedBy(pId, pId). "pId" and "pId" together uniquely identify each relationship.
5. hasEntries(sDate, regNum): This connection shows the relationship of "hasEntries" between "Race" and "Horse." The variables "sDate" and "regNum" individually identify each connection.
6. hasRace(sDate, lId): The connection "hasRace" between "RaceSchedule" and "Race" is represented by hasRace(sDate, lId). The combination of "sDate" and "lId" uniquely identifies each connection.
7. hasSchedule(sDate, lId): This connection shows the relationship of "hasSchedule" between "RaceSchedule" and "Track." "sDate" and "lId" provide a unique identity for each connection.