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?

NULLs are treated differently in comparison operators in SQL. When a comparison operator is used with a NULL value, the result is always NULL. This is because NULL represents an unknown or missing value, so it cannot be compared to any other value.

When aggregate functions are applied in an SQL query, NULLs are generally ignored. For example, if we use the SUM function to add up several values, any NULL values in the data will be treated as zero. The COUNT function, on the other hand, will not count NULL values at all.

If NULLs exist in grouping attributes, they are treated as a separate group. For example, if we group a set of data by a certain attribute, any NULL values in that attribute will be grouped together in their own separate group. This can be useful in some cases, but it can also cause unexpected results if you are not careful.

To handle NULLs in SQL queries, we can use the IS NULL and IS NOT NULL operators to check for their presence. We can also use the COALESCE function to replace NULL values with a specific value, or use the IFNULL function in MySQL or the NVL function in Oracle to do the same.

2. Define the five basic relational algebra operations. Define the Join, Intersection, and Division operations in terms of these five basic operations with examples

The five basic relational algebra operations are SELECT, PROJECT, UNION, SET DIFFERENCE, and CARTESIAN PRODUCT.

SELECT operation is used to retrieve tuples that satisfy a certain condition. For example, SELECT name FROM employees WHERE salary > 50000 retrieves the names of employees whose salary is greater than 50000.

PROJECT operation is used to retrieve certain columns from a relation. For example, PROJECT name, salary FROM employees retrieves the name and salary columns from the employees relation.

UNION operation is used to combine two relations with the same schema. For example, R UNION S will return a relation that includes all tuples from R and S.

SET DIFFERENCE operation is used to retrieve tuples that are in one relation but not in another. For example, R - S will return all tuples that are in R but not in S.

CARTESIAN PRODUCT operation is used to combine all tuples from two relations. For example, if R has m tuples and S has n tuples, then R × S will have m × n tuples.

JOIN operation combines two relations based on a common attribute. There are several types of JOINs, such as EQUI JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, and FULL OUTER JOIN. For example, SELECT * FROM R JOIN S ON R.A = S.A will retrieve all tuples from R and S where A value is the same.

INTERSECTION operation retrieves tuples that are common to two relations. For example, R ∩ S will return all tuples that are in both R and S.

DIVISION operation is used to find tuples in the first relation that are related to all tuples in the second relation. For example, R ÷ S will return all tuples from R that are related to all tuples in S.

Theta join is a general case of JOIN operation that includes only those tuples that satisfy the matching criteria. For example, SELECT * FROM R JOIN S ON R.A > S.A will retrieve all tuples from R and S where A value in R is greater than A value in S.

3. Discuss the differences between the five Join operations: Theta join, Equijoin, Natural join,

Outer join, and Semijoin. Give examples to illustrate your answer.

There are a total of five join operations in relational algebra, each with its own unique characteristics and purposes.

Theta Join: Theta join is used to combine two tables based on a particular condition represented by a theta symbol (θ). It allows for all comparison operators to be used. For example, if we have two tables T1 and T2, we can perform a theta join on them using the condition T1.A > T2.B.

Equijoin: Equijoin is used to combine two tables based on the equality of their attributes. It uses the equality operator (=) to compare columns between tables. For example, if we have two tables T1 and T2 with a common attribute A, we can perform an equijoin on them using the condition T1.A = T2.A.

Natural Join: Natural join performs a join on attributes with the same name and domain across relations. It automatically matches the columns between tables with the same name. For example, if we have two tables T1 and T2 with a common attribute A, we can perform a natural join on them without specifying any condition, and the join will be performed on the common attribute A.

Outer Join: Outer join allows us to retain all records from one table even if there is no matching record in the other table. There are three types of outer joins: left outer join, right outer join, and full outer join. Left outer join returns all the records from the left table and the matched records from the right table. Right outer join returns all the records from the right table and the matched records from the left table. Full outer join returns all the records from both tables, with NULL values in case of unmatched attributes.

Semi-join: Semi-join is a type of join that returns only the matching rows from the left table and discards the non-matching rows from both tables. It is similar to the inner join, but it only returns the rows from the left table. For example, if we have two tables T1 and T2, we can perform a semi-join on them using the condition T1.A = T2.A and only the matching rows from T1 will be returned.

Overall, each join operation has its own specific use case and should be chosen based on the requirements of the query.

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 and equijoin are two types of join operations in relational algebra. Both join operations are used to combine two or more tables based on a common attribute. However, the key difference between the two is that equijoin performs a join based on equality between two attributes, while theta join performs a join based on any comparison operator other than equality. In equijoin, the join condition is expressed using the equality operator, while in theta join, the join condition can be expressed using any comparison operator. Therefore, equijoin is a special case of theta join where the join condition involves only the equality operator.

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

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

The operation selects the hotel numbers from the Room relation where the price is greater than 50. The resulting relation would have only the hotel numbers. In other words, the relation would have the hotel numbers of all those hotels whose rooms have a price greater than 50.

b. $\sigma_{(Hotel, HotelNo) = Room,HotelNo}(\mathbf{HOTEL * Room})$

The operation joins the Hotel and Room relations on the common attribute of hotel number. The resulting relation would have combined attributes from both relations. In other words, the relation would have all the details of hotels and their rooms where the hotel number is common in both the relations.

c. $\pi_{hotelname}(Hotel \bowtie_{Hotel.hotelNo = Room.HotelNo}(\sigma_{price > 50}(Room)))$

The given relational algebra expression selects hotel names from the Hotel relation which have rooms with a price greater than 50. First, it applies the selection operation on the Room relation to select only those rooms where the price is greater than 50. Then, it performs the natural join operation on the

resulting relation and the Hotel relation, where the hotel numbers are common in both relations. Finally, it projects the hotel names from the resulting relation.

Therefore, the resulting relation will contain the names of hotels that have rooms with a price greater than 50. The relation will only include hotel names and no other attributes from either relation.

d. **Guest** (σdateTo≥'1jan 2007' (Booking))

The relational algebra operation σ (sigma) is used to select tuples that satisfy certain conditions from a relation. In this case, σdateTo≥'1jan 2007' (Booking) selects all the bookings where the dateTo is greater than or equal to January 1st, 2007. The resulting relation would contain all the attributes of the Booking relation, but only include the tuples that satisfy the given condition. Therefore, the resulting relation would contain all the bookings made on or after January 1st, 2007.

e. Hotel▷ Hotel.hotelNo = Room.hotelNo (σ price > 50(Room)):

This operation selects the hotel numbers from the Room relation where the price is greater than 50. Then, it joins the Hotel relation and the Room relation on the common attribute of hotel number. The resulting relation would contain all the attributes from both relations where the hotel number is common, and the price of the room is greater than 50.

f. πguestName, hotelNo (Booking⋈ Booking.guestNo = Guest guestNo Guest) ÷ πhotelNo(σcity ='London' (Hotel)):

This operation joins the Booking and Guest relations on the common attribute of guest number. Then, it selects the guest names and hotel numbers from the resulting relation. The third operation selects hotel names from the Hotel relation which have rooms with a price greater than 50 and are located in London. Finally, it performs a natural join operation on the resulting relation and the Hotel relation, projecting only the hotel names. The resulting relation would contain only the selected attributes, i.e., guest name and hotel number, of the guests who booked rooms in London hotels with prices greater than 50.

6. Using relational algebra, produce a report of all employees from the IT and planning departments who are born after 1990. The following tables form part of a database held in an RDBMS:

To produce a report of all employees from the IT and planning departments who were born after 1990 using relational algebra, we can first obtain the employee numbers of all employees in the IT and planning departments using the selection operator. We can then join the resulting table with the Employee table to obtain all the details of these employees, and finally select those employees who were born after 1990 using the selection operator again.

Here's the relational algebra expression:

σ deptName='IT' ∨ deptName='planning' (Department ⋈ WorksOn ⋈ σ year(DOB)>1990 Employee)

This will give us a table with all the details of employees who are in the IT or planning departments and were born after 1990.

a) List all employees.

Relational Algebra:

π empNo, fName, lName, address, DOB, sex, position, deptNo (Employee)

Tuple Relational Calculus:

{<e.empNo, e.fName, e.lName, e.address, e.DOB, e.sex, e.position, e.deptNo> | e ∈ Employee}

Domain Relational Calculus:

{<empNo, fName, lName, address, DOB, sex, position, deptNo> | ∃e (Employee(e) ∧ empNo = e.empNo)}

b) List all the details of employees who are female and born after 1990.

Relational Algebra:

σ sex = 'F' ∧ DOB > '1990-01-01' (Employee)

Tuple Relational Calculus:

{<e.empNo, e.fName, e.lName, e.address, e.DOB, e.sex, e.position, e.deptNo> | e ∈ Employee ∧ e.sex = 'F' ∧ e.DOB > '1990-01-01'}

Domain Relational Calculus:

{<empNo, fName, IName, address, DOB, sex, position, deptNo> | ∃e (Employee(e) ∧ sex = 'F' ∧ DOB > '1990-01-01' ∧ empNo = e.empNo)}

c) List all employees who are not managers and are paid more than $1500.

Relational Algebra:

σ position ≠ 'Manager' ∧ salary > 1500 (Employee)

Tuple Relational Calculus:

{<e.empNo, e.fName, e.IName, e.address, e.DOB, e.sex, e.position, e.deptNo> | e ∈ Employee ∧ e.position ≠ 'Manager' ∧ e.salary > 1500}

Domain Relational Calculus:

{<empNo, fName, IName, address, DOB, sex, position, deptNo> | ∃e (Employee(e) ∧ position ≠ 'Manager' ∧ salary > 1500 ∧ empNo = e.empNo)}

d) Produce a list of the names and addresses of all employees who work for the IT department.

Relational Algebra:

π fName, IName, address (Employee ⋈ (σ deptName = 'IT' (Department)))

Tuple Relational Calculus:

{<e.fName, e.IName, e.address> | e ∈ Employee ∧ ∃d (Department(d) ∧ d.deptNo = e.deptNo ∧ d.deptName = 'IT')

Domain Relational Calculus

{<fName, IName, address> | ∃d,e (Department(d) ∧ Employee(e) ∧ d.deptNo = e.deptNo ∧ d.deptName = 'IT' ∧ fName = e.fName ∧ IName = e.IName ∧ address = e.address)

e) Produce a list of the names of all employees who work on the SCCS project.

Relational Algebra:

π fName (Employee ⋈ (σ projName = 'SCCS' (Project)))

Tuple Relational Calculus:

{<e.fName> | e ∈ Employee ∧ ∃p (Project(p) ∧ p.projNo = e.projNo ∧ p.projName = 'SCCS')}

Domain Relational Calculus:

{<fName> | ∃p,e (Project(p) ∧ Employee(e) ∧ p.projNo = e.projNo ∧ p.projName = 'SCCS' ∧ fName = e.fName)}

f**) Produce a complete list of all managers who are due to retire this year, in alphabetical order of surname.**

Relational Algebra:

π fName, lName, address (Employee ⋈ (π deptNo (σ mgrEmpNo = empNo ∧ DOB < '1960-01-01' (Employee))))

Tuple Relational Calculus:

{<e.fName, e.lName, e.address> | e ∈ Employee ∧ ∃m (Employee(m) ∧ m.empNo = e.mgrEmpNo ∧ m.DOB < '1960-01-01')}

Domain Relational Calculus:

{<fName, lName, address> | ∃m,e (Employee(m) ∧ Employee(e) ∧ m.empNo = e.mgrEmpNo ∧ m.DOB < '1960-01-01' ∧ fName = e.fName ∧ lName = e.lName ∧ address = e.address)}

g) Formulate the following queries in relational algebra. The queries have already been formulated in relational algebra.

h) Find out how many managers are female.

Relational Algebra:

σ sex = 'F' ∧ position = 'Manager' (Employee)

Tuple Relational Calculus:

{<e.empNo, e.fName, e.IName, e.address, e.DOB, e.sex, e.position, e.deptNo> | e ∈ Employee ∧ e.sex = 'F' ∧ e.position = 'Manager'}

Domain Relational Calculus:

{<empNo, fName, IName, address, DOB, sex, position, deptNo> | ∃e (Employee(e) ∧ sex = 'F' ∧ position = 'Manager' ∧ empNo = e.empNo)}

i) Produce a report of all projects under the IT department.

Relational Algebra:

π projNo, projName (σ deptName = 'IT' (Project))

Tuple Relational Calculus:

{<p.projNo, p.projName> | p ∈ Project ∧ ∃d (Department(d) ∧ d.deptNo = p.deptNo ∧ d.deptName = 'IT')}

Domain Relational Calculus:

{<projNo, projName> | ∃p,d (Project(p) ∧ Department(d) ∧ p.deptNo = d.deptNo ∧ d.deptName = 'IT' ∧ projNo = p.projNo ∧ projName = p.projName)}

j) Using the union operator, retrieve the list of employees who are neither managers nor supervisors. Attributes to be retrieved are first name, last name, position, sex and department number

Relational Algebra:

π fName, IName, position, sex, deptNo ((Employee - Manager) - Supervisor)

Tuple Relational Calculus:

{<e.fName, e.lName, e.position, e.sex, e.deptNo> | e ∈ Employee ∧ e.empNo ∉ (π empNo (Manager) ∪ π empNo (Supervisor))}

Domain Relational Calculus:

{<fName, lName, position, sex, deptNo> | ∃e (Employee(e) ∧ e.empNo ∉ {empNo | ∃m (Manager(m) ∧ m.empNo = empNo)} ∧ e.empNo ∉ {empNo | ∃s (Supervisor(s) ∧ s.empNo = empNo)} ∧ fName = e.fName ∧ lName = e.lName ∧ position = e.position ∧ sex = e.sex ∧ deptNo = e.deptNo)}

7. Provide the relational schema for the following ERD :

Entites:

Horse

Owner

Trainer

Jockey

Entry

Race

Track

Race Schedule

Attributes

Horse: name, regNum, dateBought, gender, type, purchasePrice

Owner: pID, pPhone, pAddress, pName

RaceSchedule: sDate

Stable: sName, sId, sPhone

Entry: finalPos, gate

Race: purse, rNumber

Track: tId, tname

Primary Keys:

Horse: regNum

Owner: pID

Stable: sName

Entry: gate

Race: rNumber

RaceSchedule: sDate

Track: tId

Cardinality Constraints:

Horse and Stable: One-to-One relationship (1:1)

Horse and Race Entry: One-to-Many relationship (1:M)

Horse and Race: Many-to-Many Relationship (M:M)

Horse and Trainer: One-to-One relationship (1:1)

Trainer and Horse: One-to-Many Relationship (1:M)

Horse and Owner: Many-to-Many relationship (M:M)

Relationships:

OwnedBy: A single horse can be owned by many owners. Also, One owner can own multiple horses.

JRaces: A single entry can have only one jockey while a jockey can be in multiple entries.

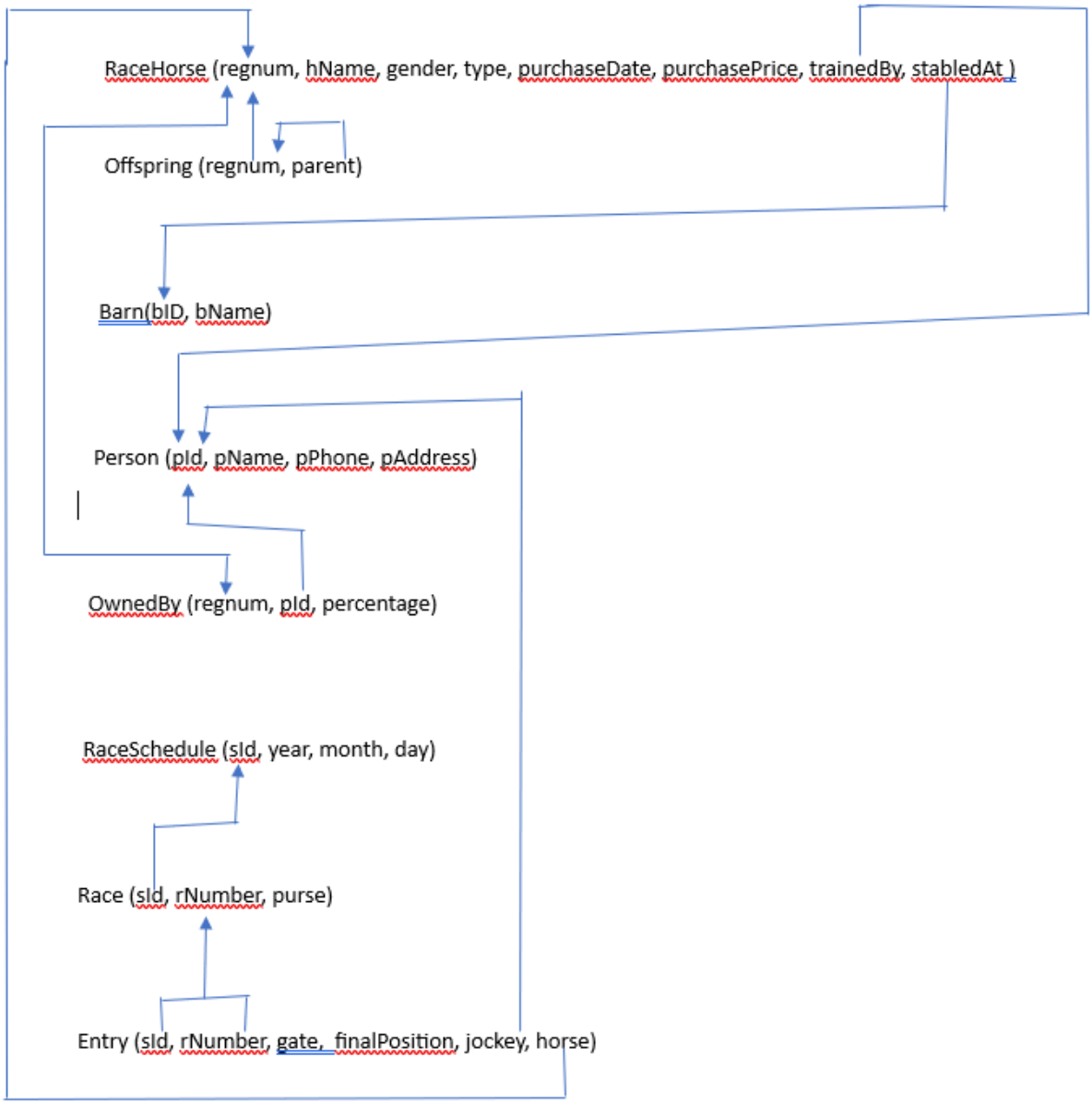
HRaces: A horse can have multiple race entries. A single race entry belongs to a single horse.

StabledAt: A horse is always stabled at one barn but the barn can have many horses.

TrainedBy: One horse can have only one trainer. A trainer can train multiple horses.

HasEntries: A race can have multiple entries.

HasSchedule: A race schedule has a unique date and is associated with one or multiple races.


RaceHorse (regnum, hName, gender, type, purchaseDate, purchasePrice, trainedBy, stabledAt)
Offspring (regnum, parent)
Barn(bID, bName)
Person (pId, pName, pPhone, pAddress)
OwnedBy (regnum, pId, percentage)
RaceSchedule (sId, year, month, day)
Race (sId, rNumber, purse)
Entry (sId, rNumber, gate, finalPosition, jockey, horse)