Ronish Shrestha (19707)
CS457 A - assignment (Relational Algebra)

Q.No.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 in Comparison Operators:

NULLs are treated differently in comparison operators in SQL. When a comparison operator is applied to a NULL value, the outcome is always NULL. This is because NULL symbolizes an unknown or missing value, which cannot be compared to another value.

For instance, if you compare a column value with NULL using the = operator, the result will not be TRUE for any row, because NULL is not considered equal to any value, not even another NULL. To specifically check for NULL values, SQL provides the IS NULL and IS NOT NULL operators.

NULLs in Aggregate Functions:

When aggregate functions are applied in an SQL query, NULLs are generally ignored.

SUM, AVG, MIN, MAX: These functions typically ignore NULL values when calculating the result. For example, SUM(column) will only add non-NULL values in the column.

COUNT: This function counts the number of non-NULL values in the column.

If we use the SUM function to add up several values, any NULL values in the data will be treated as zero. In contrast, the COUNT function does not count NULL values.

To handle NULLs in SQL queries, we can use the IS NULL and IS NOT NULL operators to determine their presence.

NULLs in Grouping Attributes:

NULL values in grouping attributes create their own separate groups. This means rows with NULL in the grouping column won't be grouped with rows having any other value, including other NULLs. NULL values in the columns being grouped are treated as equal, and thus all rows with NULL in the grouping column(s) are aggregated into a single group.

Q.No.2) ***Define the five basic relational algebra operations. Define the Join, Intersection, and Division operations in terms of these five basic operations with examples.***

Selection (σ): This operation is used to select rows from a relation that satisfy a specific predicate (condition). The result is a relation that includes only those tuples (rows) for which the condition holds true.

Projection (π): Project selects columns from a relation. It eliminates the repetitive columns in the result table and returns a relation which represents only some specified features (columns) in the initial relationship.
For example, PROJECT name, salary FROM employees retrieves the name and salary columns from the employees relation.

Cartesian Product (×): Two relations are combined through the Cartesian product operation to form a new relation. For every tuple in relation 1, it matches with every tuple in relation 2. For example, if R has m tuples and S has n tuples, then R × S will have m × n tuples.

Union (∪): This operation merges the tuples of two relations into a single relation that eliminates duplicates.

Set Difference (−): The result of the set difference operation is composed by the tuples that are in the first relation but not in the second.

Join (⋈): Join operations combine tuples from two relations based on a condition that typically involves equality between attributes. There are several types of joins: Natural join, theta join, self join, outer join (left outer join, right 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 (∩): This operation identifies the tuples that appear in both relations. It only locates the common components. 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. R ÷ S will return all tuples from R that are related to all tuples in S. For example, if R represents "Students who have taken Courses" and S represents "Courses," R ÷ S would give "Students who have taken all Courses."

Q.No.3) Discuss the differences between the five Join operations: Theta join, Equijoin, Natural join, Outer join, and Semijoin. Give examples to illustrate your answer.

→

1. Theta Join

A Theta join is a join where the tables are joined on a relationship other than the equality of two columns. The theta condition is a boolean expression which can use any comparison operators not just equality operators.

Example:

Column age is present in Table A, and Table B has column min_age. A Theta join on these tables could be A.age >= B.min_age In this case, the rows would be combined where age in Table A is greater than or equal to min_age in Table B.

2. Equijoin

An Equijoin is a special case of Theta join when the theta condition is in fact an equality requirement.

Example:

Table A (emp_id, emp_name, dept_id) and Table B (dept_id, dept_name) An Equijoins on dept_id will unite rows from Table A and Table B, where the dept_id matches.

3. Natural Join

A natural join automatically joins tables on common columns with the same name and data type. It performs an Equijoin on all columns with the same names.

Example:

If both the Table A (emp_id, emp_name, dept_id) and the Table B (dept_id, dept_name) have a dept_id column, the Natural join between A and B would joint the tables on dept_id without the need to specify the column name.

4. Outer Join

An Outer join yields all the rows of one table and the matched rows of the second table. If there is no match, NULL appears on the side with no match. There are three types of Outer joins: Left, Right, and Full.

Example:
Left outer join on Table A (employees) and Table B (departments) on dept_id would include all employees even if they are not with a department.

5. Semijoin
The Semijoin between two tables returns all rows in the first table in which there is at least one row in the second table which satisfies the join condition.

Example:
If you have Table A (employees) and Table B (departments), a Semijoin on dept_id would give you the rows from Table A (employees) that have matching dept_id in Table B (departments), but the result will contain only columns from Table A.

Q.No.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 both relational joins operations that derive rows from different tables. The main difference is that it uses different conditions to match rows. A theta join is a more general form of operation that uses a comparison operator, such as less than (<) greater than (>) equal to (=) , etc., to combine rows based on a given condition. A Equijoin is a particular type of Theta join which uses only the equality (=) operator on rows from the combined tables.

For instance, you have two tables, one holding employee information and another holding department information and both tables have a field in common called 'dept_id'. A Theta join could then match an employee to a department where the employees "dept_id" would be greater than say 100. On the contrary, an Equijoin would match employees to departments whose "dept_id" fields are identical in both tables, therefore, linking rows with the same department ID.

Q.No.5)Describe the relations that would be produced by the following relational algebra operations:

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

→This expression first filters the Room relation to only include those rooms where the price attribute is greater than 50. This is the selection operation (σ). After this selection, the projection operation (π) reduces each tuple to just the hotelNo attribute.

In other words, the relation would have the hotel numbers of all those hotels whose rooms have a price greater than 50.

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

→ Here, we take the Cartesian product (×) of the Hotel and Room relations, which pairs each row from Hotel with each row from Room. After this, we apply a selection (σ) to keep only those pairs where the hotelNo attribute matches in both Hotel and Room.

A relation with full details from both Hotel and Room where the hotel numbers are the same. 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) $\pi_{hotelName}(Hotel \bowtie_{Hotel.hotelNo\ =\ Room.HotelNo}(\sigma_{price\ >\ 50}(Room)))$

→ In this expression, we first select the tuples from Room where the price is greater than 50. Then we perform a natural join (⋈) between Hotel and the result of that selection. The natural join will match tuples from Hotel and Room where they share common attributes (likely hotelNo). Finally, we project (π) the hotelName attribute from the resulting relation.

Result: A list of hotel names where there are rooms with a price greater than 50.

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

→ This expression selects all bookings from the Booking relation that have an end date (dateTo) on or before January 1, 2007. Then it performs a natural join with the Guest relation, matching on a common attribute (probably guestNo).

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))

→ First, we select rooms from Room with a price greater than 50. Then we perform a theta join (⋈) between Hotel and the result of that selection on the condition that Hotel.hotelNo equals Room.hotelNo.

Result: A relation of hotels paired with their respective rooms, specifically for rooms that cost more 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.

Q.No.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:

Formulate the following queries in relational algebra, tuple relational calculus, and domain

(a) List all employees.

→Relational algebra:π empNo, fName, lName, address, DOB, sex, position, deptNo (Employee)

→Tuple relational calculus:{ | e ∈ Employee}

→Domain relational calculus: {<empNo, fName, lName, address, DOB, sex, position, deptNo> | ∃t (Employee(empNo, fName, lName, address, DOB, sex, position, deptNo)) }

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

→Relational algebra: σ sex='Female' ∧ DOB > '1990-01-01' (Employee)

→Tuple relational calculus:{t | Employee(t) ∧ t.sex='Female' ∧ t.DOB > '1990-01-01'}

→Domain relational calculus: {<a, b, c, d, e, f, g, h> | ∃t (Employee(t) ∧ t.sex='Female' ∧ t.DOB > '1990-01-01' ∧ t.empNo=a ∧ t.fName=b ∧ t.lName=c ∧ t.address=d ∧ t.DOB=e ∧ t.sex=f ∧ t.position=g ∧ t.deptNo=h)}

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

→Relational algebra: σ ¬(empNo IN (π mgrEmpNo (Department))) Λ salary > 1500 (Employee)

→Tuple relational calculus:{t | Employee(t) ∧ ¬∃d (Department(d) ∧ d.mgrEmpNo = t.empNo) ∧ t.salary > 1500}

→Domain relational calculus:{<a, b, c, d, e, f, g, h> | ∃t (Employee(t) ∧ ¬∃u (Department(u) ∧ u.mgrEmpNo = t.empNo) ∧ t.salary > 1500 ∧ t.empNo=a ∧ t.fName=b ∧ t.lName=c ∧ t.address=d ∧ t.DOB=e ∧ t.sex=f ∧ t.position=g ∧ t.deptNo=h)}

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

→Relational algebra: π fName, lName, address (σ deptName='IT' (Department) ⋈ Employee)

→Tuple relational calculus:{t | ∃d (Employee(t) ∧ Department(d) ∧ d.deptName='IT' ∧ t.deptNo=d.deptNo)}

→Domain relational calculus: {<a, b, c> | ∃t, d (Employee(t) ∧ Department(d) ∧ d.deptName='IT' ∧ t.deptNo=d.deptNo ∧ t.fName=a ∧ t.lName=b ∧ t.address=c)}

e)
Produce a list of the names of all employees who work on the SCCS project.
→Relational algebra:π fName, lName (Employee ⋈ (π empNo (σ projName='SCCS' (Project ⋈ WorksOn))))

→Tuple relational calculus: {t | ∃p, w(Employee(t) ∧ Project(p) ∧ WorksOn(w) ∧ p.projName='SCCS' ∧ w.empNo = t.empNo ∧ p.projNo = w.projNo)}

→Domain relational calculus: {<e.fName, e.lName> | ∃p, w(Project(p) ∧ WorksOn(w) ∧ p.projName='SCCS' ∧ w.empNo = e.empNo ∧ p.projNo = w.projNo)}

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 (σ (2024 - year(DOB)) ≥ 65 (Employee ⋈ Department))

→Tuple relational calculus:{t | ∃d(Employee(t) ∧ Department(d) ∧ t.empNo = d.mgrEmpNo ∧ (2024 - t.DOB.year) ≥ 65)} ORDER BY t.lName ASC

→Domain relational calculus: {<e.fName, e.lName> | ∃d(Employee(e) ∧ Department(d) ∧ e.empNo = d.mgrEmpNo ∧ (2024 - e.DOB.year) ≥ 65)} ORDER BY e.lName ASC

g)
Formulate the following queries in relational algebra.
Queries are already formulated in the relational algebra.

h)
Find out how many managers are female.

→Relational algebra:COUNT(σ sex='Female' (Employee ⋈ Department))

→Tuple relational calculus: {COUNT(t) | ∃d(Employee(t) ∧ Department(d) ∧ t.empNo = d.mgrEmpNo ∧ t.sex='Female')}

→Domain relational calculus: {COUNT(<e.empNo>) | ∃d(Employee(e) ∧ Department(d) ∧ e.empNo = d.mgrEmpNo ∧ e.sex='Female')}

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

→Relational algebra: π projName (σ deptName='IT' (Department ⋈ Project))

→Tuple relational calculus: {p | ∃d(Project(p) ∧ Department(d) ∧ d.deptName='IT' ∧ d.deptNo = p.deptNo)}

→Domain relational calculus:{<p.projName> | ∃d(Project(p) ∧ Department(d) ∧ d.deptName='IT' ∧ d.deptNo = p.deptNo)}

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, lName, position, sex, deptNo (Employee) - π fName, lName, position, sex, deptNo (Employee ⋈ σ position='Manager' ∨ position='Supervisor' (Employee)))

→Tuple relational calculus: {t | Employee(t) ∧ ¬∃d(Department(d) ∧ d.mgrEmpNo = t.empNo) ∧ ¬∃s(Employee(s) ∧ s.position='Supervisor')}

→Domain relational calculus: { | ∃e (Employee(e) ∧ e.empNo ∉ {empNo | ∃m (Manager(m) ∧ m.empNo = empNo)} ∧ e.empNo ∉ {empNo | ∃s (Supervisor(s) ∧ s.empNo = empNo)} ∧ fName = e.fName ∧ IName = e.IName ∧ position = e.position ∧ sex = e.sex ∧ deptNo = e.deptNo)}

Q.NO.7) Provide the relational schema for the following ERD.

→ Following are the entities and their attributes with primary keys:

Horse:

- regNum (Primary Key)
- name
- gender
- type
- dateBought
- purchasePrice
- sire (Foreign Key referencing Horse.regNum)
- dam (Foreign Key referencing Horse.regNum)
- trainedBy (Foreign Key referencing Trainer.tId)

Owner:

- pId (Primary Key)
- pName
- pPhone
- pAddress

Stable:

- sId (Primary Key)
- sName
- sPhone
- contact (Foreign Key referencing Owner.pId)

Person (Supertype of Owner, Trainer, and Jockey):

- pId (Primary Key, shared with Owner, Trainer, Jockey)
- pName
- pPhone
- pAddress

Trainer:

- tId (Primary Key, Foreign Key referencing Person.pId)
- salary

Jockey:

- jId (Primary Key, Foreign Key referencing Person.pId)
- weight

- weightDate

TrainerJockey:

- tId (Primary Key, Foreign Key referencing Trainer.tId)
- jId (Primary Key, Foreign Key referencing Jockey.jId)

Race:

- rNumber (Primary Key)
- purse

Entry:

- eId (Primary Key)
- horse (Foreign Key referencing Horse.regNum)
- race (Foreign Key referencing Race.rNumber)
- gate
- finalPos

RaceSchedule:

- rsId (Primary Key)
- sDate
- track (Foreign Key referencing Track.tId)

Track:

- tId (Primary Key)
- tName

Here are the cardinality relation of each entities:

Horse to Owner (ownedBy):

- A horse can be owned by one or more owners (n).
- An owner can own one or more horses (n).

Horse to Stable (stabled):

- A horse is stabled in one stable (1).

- A stable can stable one or more horses (n).

Horse to Trainer (trainedBy):

- A horse is trained by one trainer (1).
- A trainer can train one or more horses (n).

Horse to Sire and Dam:

- A horse has one sire (male parent) (1) and one dam (female parent) (1).
- A sire can have many offspring (n), and a dam can also have many offspring (n).

Horse to Race (hasRace and Entry):

- A horse can enter in many races (n).
- A race can have many horse entries (n).

Race to RaceSchedule (hasRace):
- A race is scheduled in one race schedule (1).
- A race schedule can include many races (n).

Stable to Person (worksFor):
- A person can work for one stable (1).
- A stable can have many persons working for it (n).

Trainer to Jockey (TrainerJockey):
- A trainer can be associated with zero or many jockeys (0 to n).
- A jockey can be associated with one or more trainers (1 to n).

Track to RaceSchedule (hasSchedule):
- A track can have one or many race schedules (n).
- A race schedule is associated with one track (1).