CHAPTER - 3

RELATIONAL MODEL

3.1 Definitions and Terminology

- **Relation**

  In a relational database, the table is known as relation because it stores the relation between data in row-column format.

- **Attribute**

  The columns are the table's attributes while the rows represent the data records.

- **Tuple**

  A single row is known as a tuple to database designers.

- **Base Relation**

  Relation that are physically stored on database are called base relations.

- **Derived Relation**

  Some relations do not physically exist on database but provide different ways of looking at the base relations known as derived relations. They are also called virtual relation.

3.2 Structure of Relational Databases

A relational database is made up of a number of tables, each of which has an unique name. For example, consider the instructor table of Figure a , which stores information about instructors. The table has four column headers: ID, name, dept name, and salary. Each row of this table records information about an instructor, consisting of the instructor's ID, name, dept name, and salary. Similarly, the course table of Figure b stores information about courses, consisting of a course id, title, dept name, and credits, for each course. Note that each instructor is identified by the value of the column ID, while each course is identified by the value of the column course id.

CamScanner

ID	name	dept_name	salary
110	Srijan	Comp. Sci.	68000
121	Sujan	Finance	80000
153	Madhav	Music	50000

Figure a: The instructor relation.

course_id	title	dept_name	credits
BIO678	Intro. to Biology	Biology	4
BIO680	Genetics	Biology	4
CS234	Intro. To Comp. Sci.	Comp. Sci.	4
CS240	Image Processing	Comp. Sci.	3
FIN987	Financial Accounting	Finance	3

Figure b: The course relation.

Figure c shows a third table, prereq, which stores the prerequisite courses for each course. The table has two columns, course id and prereq id. Each row consists of a pair of course identifiers such that the second course is a prerequisite for the first course. Thus, a row in the prereq table indicates that two courses are related in the sense that one course is a prerequisite for the other. As another example, when we consider the table instructor, a row in the table can be thought of as representing the relationship between a specified ID and the corresponding values for name, dept name, and salary values.

course_id	prereq_id
BIO680	BIO78
CS240	CS234

Figure c: The *prereq* relation.

In general, a row in a table represents a relationship among a set of values. We can see that the relation instructor has four attributes:

ID, name, dept name, and salary.

We use the term relation instance to refer to a specific instance of a relation, that is, containing a specific set of rows.

The instance of instructor shown in Figure a has 3 tuples, corresponding to 3 instructors.

The order in which tuples appear in a relation is irrelevant, since a relation is a set of tuples. Thus, whether the tuples of a relation are listed in sorted order or are unsorted, as does not matter; the relations in the both case are the same, since both contain the same set of tuples.

For each attribute of a relation, there is a set of permitted values, called the domain of that attribute. Thus, the domain of the salary attribute of the instructor relation is the set of all possible salary values, while the domain of the name attribute is the set of all possible instructor names.

3.3 The Relational Algebra

The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result. The fundamental operations in the relational algebra are: select, project, union, set difference, cartesian product, rename, select, project, and rename operations are called unary operations, because they operate on one relation. Union, set difference and Cartesian product operate on pairs of relations and are, therefore, called binary operations. Operations such as set intersection, natural join, and assignment are defined in terms of fundamental operations.

Schema of University Database

Classroom (<u>building</u>, <u>room number</u>, capacity)

department (<u>dept name</u>, building, budget)

course (<u>course id</u>, title, dept_name, credits)

instructor (<u>ID</u>, name, dept_name, salary)

CamScanner

teaches (ID, course_id, sec_id, semester, year)

section (course_id, sec_id, semester, year, building, room_number, time_slot_id)

Schema of kathmandu Database

dept(deptno, dname, d_head)

emp(emp_id, deptno, ename, position, salary)

SELECT Operation

The select operation selects tuples that satisfy a given predicate. Lowercase greek letter sigma (σ) is used to denote selection. The predicate appears as a subscript to σ. The argument relation is in parentheses after the σ.

Syntax:

$\sigma_p(r)$

'σ' denotes selection 'p' denotes predicate 'r' denote relation/table

Example:

1. **Find the record of IT department**

   $\sigma_{dname = "IT"}(dept)$

   Here, dname = "IT" is a predicate (p) and dept is a relation (r)

2. **Find the record of employees whose salary is above 50000**

   $\sigma_{salary > 50000}$ (emp)

   In general, we allow comparisons using =, !=, <, ≤, >, and ≥ in the selection predicate. Furthermore, we can combine several predicates into a larger predicate by using the connectives AND (∧), OR (∨), and NOT (¬).

3. **Find the record of employees who works in department 1 and salary is above 50000**

   $\sigma_{deptno = 1 \wedge salary > 50000}$ (emp)

Project Operation

Project operation is denoted by the uppercase greek letter pi (Π). This operator helps to keep specific columns from a relation and discards the other columns.. Desired column list appears in the subscript to Π. Project operation is used to filter columns from a table.

Syntax:

$\Pi_{c1, c2, c3, ... cn}$ (R), where c1, c2 , c3 are columns of relation R

Example:

1. **Find id, name and salary of employee from all the tuples of emp table.**

   $\Pi_{emp\_id, ename, salary}$ (emp)

2. **Find name and salary of employees who work in department no. 1**

   $\Pi_{ename, salary}(\sigma_{deptno = 1}$ (emp))

Rename Operation

Rename operation is denoted with small greek letter rho ρ. The results of relational algebra operation are also relations but without any name. The rename operation allows to rename result of such expression. $\rho_x(E)$, where the result of expression E is saved with name of x. It is also used to rename existing relation. ρ_x (r), where r is an existing relation on database. It is also used to rename attributes of a result.

Syntax:

$\rho_{x[A1, A2, A3...An]}$ (E),

Returns the result of expression E under name X and attributes renamed to A1, A2, A3...An

Union Operation

It performs binary union between two given relations and defined as:

$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$

Notation:

$r \cup s$

Where r and s are either database relations or relation result set (temporary relation).

CamScanner

For a union operation to be valid, the following conditions must hold :

- r and s must have same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

Example:

Find the set of all courses taught in the Fall 2009 semester, the Spring 2010 semester, or both.

Step1:

To find the set of all courses taught in the Fall 2009 semester, we can write:

Π_{course_id} ($\sigma_{semester = "Fall" \wedge year = 2009}$ (section))

Step 2:

To find the set of all courses taught in the Spring 2010 semester, we write:

Π_{course_id} ($\sigma_{semester = "Spring" \wedge year = 2010}$ (section))

To answer the above query, we need the union of these two sets

Step 3:

Π course_id (σ semester = "Fall" $\wedge$ year =2009 (section)) $\cup$

Π course_id (σ semester = "Spring" $\wedge$ year =2010 (section))

Set Difference Operation

The set-difference operation is denoted by Minus (-). It allows to find tuples in one relation but not in another relation. The expression r – s produces a relation containing those tuples in r but not in s.

Syntax:

r–s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold:

- The relations r and s must be union-compatible
- The schema of the result is defined to be identical to the schema of *R*.

Example :

1. **Provide the name of authors who have written books but not articles.**

   Π_{author}(Books) - Π_{author}(Articles)

2. **Find the Courses offered in the Fall 2009 semester but not in Spring 2010 semester.**

   We can find all the courses taught in the Fall 2009 semester but not in Spring 2010 semester by writing:

   Π_{course_id} ($\sigma_{semester = "Fall" \wedge year = 2009}$ (section)) –

   Π_{course_id} ($\sigma_{semester = "Spring" \wedge year = 2010}$ (section))

Cartesian Product Operation

The Cartesian-product operation is denoted by a *cross (×)*. It combines information of two different relations into one. Cartesian product of relations r1 and r2 is written as r1 × r2.

Notation:

r X s

Where r and s are relations and their output will be defined as

r X s = { q t | q ∈ r and t ∈ s}

Example:

1. **Find tuples from table dept and emp**

   dept X emp

2. **Find name of employee from table dept and emp where department name is IT**

   Π_{ename} ($\sigma_{dname = "IT"}$ (dept X emp))

Additional operation in RA

Set Intersection (∩)

An intersection is defined by the symbol ∩. r ∩ s defines a relation consisting of a set of all tuple that are in *both r and s*. For set intersection to be valid, *r* and *s* must be union-compatible.

CamScanner

Example:

Find the set of all courses taught in both the Fall 2009 and the Spring 2010 semesters, using set intersection.

$\Pi_{\text{course id}}(\sigma_{\text{semester = "Fall"} \wedge \text{year = 2009}}(\text{section})) \cap$

$\Pi_{\text{course id}}(\sigma_{\text{semester = "Spring"} \wedge \text{year = 2010}}(\text{section}))$

Note: we can rewrite any relational-algebra expression that uses set intersection by replacing the intersection operation with a pair of set-difference operations as $r \cap s = r - (r - s)$.

Thus, set intersection is not a fundamental operation and does not add any power to the relational algebra. It is simply more convenient to write r ∩ s than to write r − (r − s).

Natural Join

The natural join is a binary operation that allows us to combine certain selection operations and a cartesian product into one operation. It is denoted by the join symbol ⋈. The natural-join operation forms a cartesian product of two relations, performs a selection based on equality of those attributes that appear in both the relations, and finally removes duplicate attributes. Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Example :

1. **Find the names of all instructors together with the course id of all courses they taught.**

   $\Pi_{\text{name, course id}}(\text{instructor} \bowtie \text{teaches})$

2. **Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach.**

   $\Pi_{\text{name, title}}(\sigma_{\text{dept name="Comp. Sci."}}(\text{instructor} \bowtie \text{teaches} \bowtie \text{course}))$

Theta Join Operation

The theta join operation is a variant of the natural-join operation. It allows to combine a selection and a Cartesian product into a single operation. Consider relations r and s , and let

θ be a predicate on attributes in R U S. The theta join operation between r and s is defined as follows:

$r \bowtie_{\theta} s = \sigma_{\theta}(r \times s)$

Example:

Find the tuple from dept and emp where department is "IT" and salary is higher than 50000.

$\text{dept} \bowtie_{\text{dept.dname = "IT"} \wedge \text{emp.salary > 50000}} \text{emp}$

is equivalent to:

$\sigma_{\text{dept.dname = "IT"} \wedge \text{emp.salary > 50000}}(\text{dept} \times \text{emp})$

Outer Join

1. **Left Outer Join**

   All the tuples from left relation R are included in the resulting relation. If there are tuples in R without any matching tuple in right relation S then S-attributes of the resulting relation are made NULL.

   The left outer join between r and s is denoted by $r \mathbin{⟕} s$.

   Example:

   dept ⟕ emp

2. **Right Outer Join**

   All the tuples from right relation S are included in the resulting relation. If there are tuples in S without any matching tuple in left relation R then R-attributes of the resulting relation are made NULL.

   The left outer join between r and s is denoted by $r \mathbin{⟖} s$.

   Example:

   dept ⟖ emp

3. **Full Outer Join**

   All tuples from both relations are included in the result, irrespective of the matching condition.

   The left outer join between r and s is denoted by by $r \mathbin{⟗} s$.

   Example:

   dept ⟗ emp

CamScanner

Extended Relational Algebra Operations

Operations that provide the ability to write queries that cannot be expressed using the basic relational- algebra operations are called extended relational-algebra operations.

Generalized Projection

Generalized-projection operation extends the projection operation by allowing operations such as arithmetic and string functions to be used in the projection list. The generalized-projection operation has the form:

$\Pi_{F1, F2, ..., Fn}(E)$

where E is any relational-algebra expression, and each of F1, F2, ..., Fn is an arithmetic expression involving constants and attributes in the schema of E

In general, an expression can use arithmetic operations such as +, −, *, and ÷ on numeric valued attributes, numeric constants, and on expressions that generate a numeric result.

Example:

$\Pi_{emp\_id, ename, salary \div 12}(emp)$

gives the emp_id, ename and the monthly salary of each instructor

Aggregate Functions

Aggregate functions take a collection of values and return a single value as a result. Aggregate operation in relational algebra is denoted by G 'calligraphic G'

Some example of aggregate functions are:

1. **SUM**

   SUM takes a collection of values and returns the sum of the values.

   $G_{SUM(salary)}$ (emp)

2. **AVG**

   AVG returns the average of the values.

   $G_{AVG(salary)}$ (emp)

3. **COUNT**

   COUNT returns the number of the elements in the collection including duplicate elements.

   $G_{COUNT(emp\_id)}$ (emp)

4. **COUNT- DISTINCT**

   COUNT- DISTINCT Returns distinct no of elements removing duplicate elements.

   $G_{COUNT\text{-}DISTINCT(emp\_id)}$ (emp)

5. **MIN**

   MIN return the minimum values in a collection.

   $G_{MIN(salary)}$ (emp)

6. **MAX**

   MAX return the maximum values in a collection.

   $G_{MAX(salary)}$ (emp)

Group by Function

Group by function is used to apply aggregate function on group of sets of tuples.

General form:

$_{A1,A2}G_{sum(A3)}(\Pi_{A1, A2, ..., An}(\sigma_P(r_1 \times r_2 \times \cdots \times r_m)))$

Example :

1. **Find the average salary in each department (group by department name)**

   $_{dept\_name}G_{avg(salary)}(emp)$

3.4 Schema and Views

Relation Schema

The name of a relation and set of attributes for a relation is called a schema. The schema for a relation consists of relation name followed by parenthesized list of its attributes.

If A1, A2, A3, ..., An are attributes then R = (A1, A2, A3, ... , An) is a relation schema.

CamScanner

Example:

cust_schema=(customer_name,customer_street,customer_city)

r(R) denotes a relation r on relation schema R.

Example:

customer(cust_schema)

Views:

A view is a tailored presentation of the data contained in one or more tables. In other words, a view is a logical or virtual table, which does not exist physically in the database. Views are also called as dictionary objects. It takes the output of a query and treats it as a table.

3.5 Data Dictionary

A data dictionary is a centralized repository of information about data that is used within an organization. It provides a detailed description of the data elements that are used in a database or other data storage system, including their structure, format, and relationships to other data elements.

In general, a data dictionary contains the following types of information:

- **Data element names and definitions**
  This includes the names of all data elements used in the system, as well as their definitions and descriptions.
- **Data element attributes**
  This includes any attributes associated with each data element, such as data type, length, format, and validation rules.
- **Data element relationships**
  This includes any relationships between data elements, such as foreign keys and other constraints.
- **Metadata**
  This includes any metadata associated with the data elements, such as data source, date of creation, and ownership.

- **Data element usage**
  This includes any rules or guidelines for using each data element, such as naming conventions and data entry guidelines.

A data dictionary is an important tool for managing and maintaining data within an organization. It ensures that all stakeholders have a consistent understanding of the data being used, which can help to prevent errors and inconsistencies. Additionally, it can be used to automate data integration and transformation processes, which can save time and reduce errors.

CamScanner

SOLUTION TO EXAMS' AND OTHER IMPORTANT QUESTIONS

1. Suppose we have the following relation
Employee(person_name,street,city)
Works(person_name,company_name,salary)
Company(company_name,city)
Write relational algebraic expressions for the follwoing queries:

   i. List the name and city of employees who work in "Pokhara" and have salary greater than Rs.50,000
   ii. Find the names of all employees who work for "ABC bank".
   iii. Delete all employees who come from "Chitwan"
   iv. Increase salary of all employee by 15% [2023 Spring]

ANS:

i. $\Pi_{person\_name, city}(\sigma_{city='Pokhara' \wedge salary>50000}(Employee \bowtie Works))$
ii. $\Pi_{person\_name}(\sigma_{company\_name = 'ABC\ bank'}(Works))$
iii. $Employee \leftarrow Employee - \sigma_{city = 'Chitwan'}(Employee)$
iv. $Employee \leftarrow \Pi_{person\_name, street, city, salary * 1.15}(Employee)$

2. Suppose we have the following relation:
Employee(person_name, street, city)
Works (person_name, company_name, salary) Company (company_name, city)
Write relational algebra expressions for the following queries:

   i. Find names of all employees who live in 'Butwal' and whose salary is less than Rs. 50,000.
   ii. Find the names of all employees who work for "Nabil Bank Limited".
   iii. Find the names and cities of residence of all employees who work for "Global bank".
   iv. Update the salary of all employees by 10%. [2022 Fall]

ANS:

i. $\sigma_{city='Butwal' \wedge salary < 50000}(Employee \bowtie Works)$
ii. $\Pi_{person\_name}(\sigma_{company\_name='Nabil\ Bank\ Limited'}(Works)$
iii. $\Pi_{person\_name, city}(\sigma_{company\_name='Global\ Bank'}(Works \bowtie Employee)$
iv. $Works \leftarrow \Pi_{person\_name, company\_name, salary*1.10}(Works)$

3. Considering the following schemas:
Sailors (sid, sname, rating, age)
Boats (bid,bname, color)
Reserves (sid, bid, day)
Write relational algebra expressions for the following queries:

   i. Find the records of sailors who have reserved boat number 103 (bid=103).
   ii. Update the color of the boat, where bid is 104, into green.
   iii. Find the names of sailors who have reserved a red or green boat.
   iv. Find the names of sailors who have reserved boat number 103 on day 5.
   v. Find the names of sailors whose name is not 'Ram'
   vi. Find the names of all boats. [2021 Spring]

ANS:

i. $(\sigma_{bid=103}(Reserves)) \bowtie Sailors$
ii. $Boats \leftarrow (Boats - \sigma_{bid='104'}(Boats)) \cup \Pi_{bid, bname, 'green'}(\sigma_{bid='104'}(Boats)$
iii. $\Pi_{sname}((Sailors \bowtie Reserves \bowtie \sigma_{color='red' \vee color='green'}(Boats))$
iv. $\Pi_{sname}((Sailors \bowtie Reserves \bowtie \sigma_{bid=103 \wedge day=5}(Reserves))$
v. $\sigma_{sname \neq 'Ram'}(Sailors)$
vi. $\Pi_{bname}(Boats)$

4. Consider following relations, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries:
Doctor (SSN, FirstName, LastName, Specialty, YearsOfExperience, PhoneNum)
Patient (SSN, FirstName, LastName, Address, DOB, PrimaryDoctor_SSN)
Medicine (TradeName, UnitPrice, GenericFlag)
Prescription (Id, Date, Doctor_SSN, Patient_SSN)
Prescription_Medicine (Prescription_Id, TradeName, NumOfUnits)

i. List the trade name of generic medicine with unit price less than $50.
ii. List the first and last name of patients whose primary doctor named 'John Smith'
iii. List the first and last name of doctors who are not primary doctors to any patient.
iv. List the SNN of distinct patients who have 'Aspirin' prescribed to them by doctor named 'John Smith'.

[2021 Fall]

ANS:

i. $\Pi_{\text{TradeName}}(\sigma_{\text{GenericFlag='Y'} \wedge \text{UnitPrice} < 50}(\text{Medicine}))$

ii. $\Pi_{\text{FirstName, LastName}}((\text{Patient} \bowtie \text{Doctor} \bowtie \sigma_{\text{FirstName='John'} \wedge \text{LastName='Smith'}}(\text{Doctor}))$

iii. $\Pi_{\text{Firstname, Lastname}}(\text{Doctor}) - \Pi_{\text{Doctor.FirstName,Doctor.LastName}}(\text{Doctor} \bowtie_{\text{Doctor.SSN=Patient. PrimaryDoctor\_SSN}} \text{Patient}))$

iv. $\text{temp} \leftarrow (\sigma_{\text{TradeName='Aspirin'}}(\text{Prescription\_Medicine})) \bowtie_{\text{Prescription\_Medicine.Prescription\_Id= Prescription.Id}} \text{Prescription} \bowtie_{\text{Prescription.Doctor\_SSN=Doctor.SSN}} \text{Doctor}$

$\Pi_{\text{Patient\_SSN}}((\text{temp} \bowtie_{\text{temp.Paient\_SSN=Patient.SSN}} \text{Patient})$

5. Write relational algebra statement for the following schemas (underline indicates primary key):
Employee (Emp_No , Name , Address)
Project (PNo ,Pname)
Workon (Emp_No , PNo)

Part (Partno ,Part_name , Qty_on_hand)
Use (Emp_No , PNo , Partno, Number)

a. Listing all the employee details who are not working yet .
b. Listing partname and Quantity on hand those were used in DBMS project.
c. List the name of the projects that are used by employee from London.
d. Modify the database so that Jones now lives in USA.
e. Update address of an employee 'Japan' to 'USA'.

[2020 Spring]

ANS:

a. $\Pi_{\text{Emp\_No, Name, Address}}(\text{Employee} - \Pi_{\text{Emp\_No, Name, Address}}(\text{Workon} \bowtie \text{Employee}))$

b. $\Pi_{\text{Part\_name, Qty\_on\_hand}}((\text{Part} \bowtie \text{Use}) \bowtie \sigma_{\text{Pname='DBMS'}}(\text{Project}))$

c. $\Pi_{\text{Pname}}((\text{Project} \bowtie \text{Use}) \bowtie \sigma_{\text{Address='London'}}(\text{Employee}))$

d. $\text{Employee} \leftarrow (\text{Employee} - \sigma_{\text{Name='Jones'}}(\text{Employee})) \cup \Pi_{\text{Emp\_No, Name, 'USA'}}(\sigma_{\text{Name='Jones'}}(\text{Employee})$

e. $\text{Employee} \leftarrow (\text{Employee} - \sigma_{\text{Address='Japan'}}(\text{Employee})) \cup \Pi_{\text{Emp\_No, Name, USA}}(\sigma_{\text{Address='Japan'}}(\text{Employee}))$

6. Using the following schema represent the following queries using Relational algebra :
PROJECT (Project num, ProjectName, ProjectType, ProjectManager)
EMPLOYEE (Empnum, Empname)
ASSIGNED_TO (Projectnum, Empnum)

i. Find Employee details working on a project name starts with 'L'
ii. List all the employee details who are working under project manager "Rohan"
iii. List the employees who are still not assigned with any project.
iv. List the employees who are working in more than one project.

[2019 Spring]

CamScanner

i. $\Pi_{\text{Empnum, Empname}}$ (EMPLOYEE $\bowtie$ $\sigma_{\text{ProjectName LIKE 'L%'}}$ (PROJECT) $\bowtie$ ASSIGNED_TO)))

ii. $\Pi_{\text{Empnum, Empname}}$ (EMPLOYEE $\bowtie$ (ASSIGNED_TO $\bowtie$ (σ ProjectManager='Rohan' (PROJECT))))

iii. $\Pi_{\text{Empnum, Empname}}$ (EMPLOYEE - $\Pi_{\text{Empnum, Empname}}$ (ASSIGNED_TO $\bowtie$ EMPLOYEE))

iv. $\Pi_{\text{Empnum, Empname}}$ (($\sigma_{COUNT(\text{Projectnum}) > 1}$ ($\Pi_{\text{Empnum, Projectnum}}$ (ASSIGNED_TO))) $\bowtie$ EMPLOYEE)

7. Compare and contrast relational algebra with relational calculas. [2019 Fall, 2014 Fall]

ANS: Relational database management systems (RDBMS) use formal languages called relational algebra and relational calculus to express and manipulate data operations and queries. They differ from one another, nevertheless, in approach, syntax, and level of abstraction. Compare and contrast them as follows:

Relational Algebra

i. **Syntax:** Relational algebra is a procedural query language that manipulates relations through the use of operators like selection, projection, union, intersection, difference, join, and division. It concentrates on the procedure or series of steps.

ii. **Expressiveness:** Compared to relational calculus, relational algebra is a more expressive language. It may describe different operations and transformations on relations and supports complicated queries.

iii. **Implementation:** The actual physical implementation of a database is more similar to relational algebra. It can be converted into a series of fundamental activities that a database management system can effectively carry out.

iv. **Query Optimization:** The clear foundation provided by relational algebra makes query optimization possible. Its operators have clearly specified features, enabling optimization strategies including performing selection and projection operations early in the query evaluation process and reordering operations

v. **Ease of Use:** In terms of understanding and application, relational algebra is simpler to formulate and optimize queries. It is frequently used in relational algebra-based query languages like SQL.

Relational Calculus

i. **Syntax:** Relational calculus is a non-procedural query language that indicates what information should be retrieved from a database but not how. To express the needed facts, it makes use of quantifiers and mathematical predicates.

ii. **Expressiveness:** Less expressive than relational algebra is relational calculus. It emphasizes the selection and filtering of tuples based on conditions and focuses on the logical characteristics and restrictions of the data.

iii. **Implementation:** Database systems do not directly implement relational calculus. In addition to providing a theoretical framework for query languages, it offers a framework for comprehending the semantics of queries.

iv. **Query Optimization:** There is no direct foundation for query optimization in relational calculus. The optimization methods are often used at the query execution level or higher-level query language level.

v. **Ease of Use:** Relational calculus is better suited for expressing sophisticated constraints or logical characteristics of data and is more appropriate for theoretical or academic uses. It may not be as natural for the formulation of practical queries as relational algebra, and it necessitates a solid understanding of mathematical logic.

Relational algebra and relational calculus are two distinct approaches to querying and manipulating relational databases. Relational algebra is procedural, expressive, and closer to the physical implementation, making it more suitable for practical database management systems. On the

other hand, relational calculus is non-procedural, emphasizes logical properties, and serves as a theoretical foundation for query languages.

8. Define TRC and DRC. [2019 Fall]

ANS: Tuple Relational Calculus (TRC) and Domain Relational Calculus (DRC) are the two basic subtypes of relational calculus.

Tuple Relational Calculus (TRC)

- A non-procedural query language called Tuple Relational Calculus describes a collection of tuples that meet a given predicate in order to provide the intended result.
- The formulas are constructed using variables, constants, logical operators, and quantifiers (existential and universal).
- TRC focuses on choosing tuples from a relation that meet a predicate or condition that is given.
- It is rarely utilized as a practical query language and is mostly used for the theoretical research of relational databases.

Domain Relational Calculus (DRC)

- Another non-procedural query language that works with single values rather than tuples is Domain Relational Calculus.
- DRC describes a collection of values for the attributes of the result relation in order to specify the desired result.
- To express the conditions that must be met for the desired values, it uses formulas with variables, constants, logical operators, and quantifiers.
- DRC focuses on selecting individual values from the domains of the attributes in a relation.
- Similar to TRC, DRC is primarily used for theoretical purposes and is not widely used as a practical query language.

9. Consider the following relational schema:
Users (uid, cname, city)
Items (itemid, itemname, city, quantity, price)
Manager (mid, aname, city)
Query (queryno, uid, mid, itemid, query_details, hitratio)
Write the relational algebraic expression for the following tasks:

i. **Find all (queryno, uid) pairs for query with a hitratio value greater than 500.**
ii. **Find all item names of items in Pokhara ordered with query_details as pokhara_details.**
iii. **Find itemids of items ordered through manager 35 but not through manager 27.** [2018 Spring]

ANS:

i. $\Pi_{\text{queryno, uid}} (\sigma_{\text{hitratio} > 500} (\text{Query}))$
ii. $\Pi_{\text{itemname}} (\sigma_{\text{city = 'Pokhara' and query\_details = 'pokhara\_details'}} (\text{Items} \bowtie \text{Query}))$
iii. $\Pi_{\text{itemid}} (\sigma_{\text{mid = 35} \wedge \text{mid = 27}} (\text{Query}) \bowtie \text{Items}))$

10. Consider the following relational Schema:
Department (DepartmentID,DepartmentName)
Designation (DesignationID, DesignationName.Salary)
Allowance (AllowanceID,AllowanceName)
Employee (EmpID, mpName, Gender, DesignationID, DepartmentID)
Allowance Details (DetailID, EmpID, AllowanceID, Amount)
Write the relational algebraic expression for the following task:

i. **Find the number of employees department-wise.**
ii. **List the employee details whose total salary is above Rs. 50000.**
iii. **List the employee those who are getting house allowance.** [2018 Fall]

i. $\cup$ count(EmpID), DepartmentName (Employee $\bowtie$ Department)

ii. Π EmpID, mpName, Gender, DesignationID, DepartmentID (σ Salary > 50000 (Employee $\bowtie$ Designation))

iii. Π EmpID, mpName (Employee $\bowtie$ AllowanceDetails) $\bowtie$ (σ AllowanceName = 'House Allowance' (Allowance))

11. Consider the following schema:
employee (person_name, street, city)
works (person_name, company_name, salary)
company (company_name, city)
manages (person_name, manager name)
Give an expression in relational algebra to express each of the following queries:

a) Find the names of all employees who earn more than their managers
b) Find the names of all employees who live in the same city and on the same street as their managers
c) Find the names of all employees within the database that do not work for "NBL company"
d) Find the names of all employees in the database who earn more than the top earner at "NBL Company" in the database. **[2017 Fall]**

ANS:

a. ρ_{W1}(works)
ρ_{W2}(works)
ρ_{M}(manages)
Π_{person_name}(W1 $\bowtie_{W1.person\_name=M.person\_name}$ M $\bowtie_{M.manager\_name=W2.person\_name \wedge W2.salary>M.salary}$ W2)

b. ρ_{W}(works)
ρ_{M}(manages)
ρ_{E1}(employee)
ρ_{E2}(employee)
Π_{person_name}(W$\bowtie$(M$\bowtie$E1)$\bowtie_{M.manager\_name = E2.person\_name \wedge E1.street = E2.street \wedge E1.city = E2.city}$ E2

c. Π_{person_name} ($\sigma_{company\_name != 'NBL company'}$ (works))

d. W1 $\leftarrow$ $G_{MAX(salary)}$($\sigma_{company\_name='NBL Company'}$(works))
Π_{person_name}($\sigma_{salary>w1.salary}$(works))

12. Consider the following schema of a relational database.
Branch (branch-name, branch-city, assets)
Account (account-number, branch-name, balance)
Customer (customer-id, customer-name, customer-street, customer-city)
Depositor (customer-id, account-name)
Loan (loan-number, branch-name, amount)
Borrow (customer-id, loan-number)
Write the relational algebra for the following queries:

i. Find all customer either account or loan
ii. List the name and city of customer who have their account at the branch location 'Butwal'.
iii. Delete all account in the branch "B 1"
iv. Increase balance by 5% to all branches **[2016 Spring]**

ANS:

i. $\Pi_{customer\text{-}name}$ ((Depositor $\bowtie_{Depositor.customer\_id= Account.account\_number}$ Account))
$\cup$ $\Pi_{customer\text{-}name}$ ((Borrow $\bowtie$ Loan))

ii. $\Pi_{customer\text{-}name,\ customer\text{-}city}$ (Customer $\bowtie_{Customer.customer\_id = Account.account\_number}$ Account $\bowtie$ ($\sigma_{branch\text{-}city = 'Butwal'}$ (Branch)))

iii. Account $\leftarrow$ $\sigma_{branch\text{-}name !='B 1'}$ (Account)

iv. Account $\leftarrow$ $\Pi_{account\_number,branch\_name,balance*1.05}$(Account)

13. Consider a student registration database comprising of the below given schema.
Student(CRN,Name,Gender,Address,Telephone)
Course(CourseID, CourseName, Hour, TeacherID)
Teacher(TeacherID,TeacherName, Office)
Registration(CRN,CourseID, Date)
Write relational algebraic expression for the following tasks:

CamScanner

i. Count the number of student registered subject in year 2015 gender wise.
ii. Show student details taught by teacher Rohit Shrestha.
iii. Delete student information taught by teacher N. Mathema. [2016 Fall]

ANS:

i. $_{Gender}G_{count(name)}$ ($\sigma_{Date=2015}$ (Student ⋈ Registration ⋈ Course)

ii. $\sigma_{TeacherName = 'Rohit Shrestha'}$ (Student ⋈ Registration ⋈ Course ⋈ Teacher)

iii. Student ← Student - $\sigma_{TeacherName = 'N. Mathema'}$ (Student ⋈ Registration ⋈ Course ⋈ Teacher)

---

14. Consider the relational database of Figure below, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries:
employee (person-name, street, city)
works (person-name, bank-name, salary)
bank (bank-name, city)
manages (person-name, manager-name)

i. Find the names of all employees in this database who live in the same city as the company for which they work.
ii. Give all employees of First Bank Corporation a 10 percent salary raise.
iii. Modify the database so that Harish now lives in Biratnagar.
iv. Delete all tuples in works relation for employee of First Bank Corporation. [2015 Spring]

ANS:

i. $\Pi_{person\text{-}name}$ ($\sigma_{employee.city = bank.city}$ (employee ⋈ works ⋈ bank))

ii. Works ←(Works - $\sigma_{bank.bank\text{-}name = 'First\ Bank\ Corporation'}$) ∪ $\Pi_{person\_name,bank\_name,salary*1.10}$($\sigma_{bank.bank\text{-}name = 'First\ Bank\ Corporation'}$ (Works))

iii. employee ← (employee - $\sigma_{person\text{-}name = 'Harish'}$ (employee) ∪ ($\Pi_{person\_name, street, 'Biratnagar'}$ ($\sigma_{person\text{-}name = 'Harish'}$ (employee))

iv. works ← works - $\sigma_{bank\_name = 'First\ Bank\ Corporation'}$ (works)

---

15. Consider the relational database of Figure below, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries:
employee (person-name, street, city)
works (person-name, bank-name, salary)
bank (bank-name, city)
manages (person-name, manager-name)

i. Find the names of all employees who work for Nepal Rastra Bank and Salary greater than $10,000.
ii. Find the names and cities of residence of all employees who work for Nepal Rastra Bank.
iii. Find the names, street address, and cities of residence of all employees who work for Nepal Rastra Bank Corporation and earn more than $10,000 per annum.
iv. Delete all tuples in works relation for employee of Nepal Rastra Bank. [2015 Fall]

ANS:

i. Π_{person_name} ($\sigma_{bank\text{-}name = 'Nepal\ Rastra\ Bank' \wedge salary > 10000}$ (works))

ii. $\Pi_{person\_name,city}$ ($\sigma_{bank\text{-}name = 'Nepal\ Rastra\ Bank'}$(employee ⋈ works))

iii. $\Pi_{person\_name,street,city}$ ($\sigma_{bank\text{-}name = 'Nepal\ Rastra\ Bank' \wedge salary*12>1000}$(employee ⋈ works))

iv. works ← works – $\sigma_{bank\_name = 'Nepal\ Rastra\ Bank'}$ (works))

---

16. Consider the relational database of Figure below, Where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries:
Employee (person-name, street, city)
Works (person-name, bank – name, salary)

CamScanner

Bank (bank-name, city)

Manages (person-name, manager- name)

i. Find the total salary sum of all the banks.

ii. Modify the database so that Ram now lives in Kathmandu.

iii. Fine the name. street address, and cities of residence of all employees who work for Nepal. World Bank Corporation and earn more than $10,000 per annum.

iv. Delete all tuples in works relation for employee of Small Bank Corporation [2014 Spring

---

ANS:

i. $_{bank\_name}G_{bank\_name,sum(salary)}(Works)$

ii. $Employee \leftarrow (Employee - \sigma_{person\_name='Ram'}(Employee)) \cup (\Pi_{person\_name,street,'Kathmandu'}(\sigma_{person\_name='Ram'}(Employee)))$

iii. $\Pi_{person\_name,street,city}(\sigma_{bank\text{-}name = 'Nepal\ Rastra\ Bank' \wedge salary*12>1000}(employee \bowtie works))$

iv. $works \leftarrow works - \sigma_{bank\_name = 'Small\ Bank\ Corporation'}(works)$

◆◆◆

CamScanner