Database Management System (MDS 505)

Jagdish Bhatta

Unit-2

The Relational Languages and Relational Model: Relational Algebra and Relational Calculus

Relational Algebra and Calculus

- **The relational algebra** is a procedural query language. A relational algebra query describes a procedure for computing the output relation from the input relations by applying the relational algebra operators.
- The relational algebra defines a set of operators from set theory as *Union*, *Intersection*, *Set Difference*, *Cartesian Product*. Similarly, other operations as *Select*, *Project & Join* are incorporated with in the algebra operations. These operations can be categorized as unary or binary according as the number of relation on which they operate on.
- **Importance of Relational Algebra:**

  The relational algebra is very important for several reasons:
  - ***First***, it provides a formal foundation for relational model operations.
  - ***Second*** and perhaps more important, it is used as a basis for implementing and optimizing queries in relational DBMS (RDBMS).
  - ***Third***, some of its concepts are incorporated into the SQL standard query language for RDBMS

Relational Algebra and Calculus

- Whereas the algebra defines a set of operations for the relational model, the **relational calculus** provides a higher-level *declarative* language for specifying relational queries. In a relational calculus expression, there is *no order of operations* to specify how to retrieve the query result—only what information the result should contain. This is the main distinguishing feature between relational algebra and relational calculus. The relational calculus is important because it has a firm basis in mathematical logic and because the standard query language (SQL) for RDBMSs has some of its foundations in a variation of relational calculus known as the tuple relational calculus.

Unary Relational Operations

- **SELECT Operation :-** SELECT operation is used to select **a *subset* of the tuples** from a relation that satisfy a ***selection condition***.
- In general, the select operation is denoted by the symbol **σ** (sigma) as: **σ <selection condition>(R)** .

  The selection condition is a Boolean expression specified on the attributes of relation R and composed of *attributes name* <comparison operator> *attribute name* or *constant value*. {Comparison operator: <, >, =, ≠, ≤, ≥ }
- **Consider an example**:
- To select those tuples of Employee having salary greater than 20000, following notation is used:

Employee

Eid	Name	Salary	Address	Dno
011	Adam	10000	Alaska	4
022	Hilton	30000	Nevada	2

σ Salary >20,000 (EMPLOYEE)

- To select those tuples of Employee having salary equal to 10000 and who lives in Alaska, following notation is used:

σ Salary =10,000 AND Address =“Alaska” (EMPLOYEE)

Unary Relational Operations

- **SELECT Operation :-**
- Properties of SELECT operation:
  - The SELECT operation **σ** **<selection condition>** (R) produces a relation S that has the same schema as R
  - The SELECT operation **σ** is ***commutative*****;** i.e. **σ** <condition1>(**σ** < condition2> (R)) = **σ** <condition2> (**σ** < condition1> (R))
  - A cascaded SELECT operation ***may be applied in any order*****;** i.e.

    σ <condition1>(**σ** < condition2> (**σ** <condition3> (R)) = **σ** <condition2> (**σ** < condition3> (**σ** < condition1> (R)))
  - A cascaded SELECT operation may be replaced by a single selection with a conjunction of all the conditions**;** i.e.

    σ<condition1>(**σ** < condition2> (**σ** <condition3> (R)) = **σ** <condition1> AND < condition2> AND < condition3> (R)))
  - **σ Salary =10,000 AND Address="Alaska" (EMPLOYEE) Or ,**
  - **σ Salary =10,000 (σ Address="Alaska" (EMPLOYEE)) Or Equivalently,**

    σ Address="Alaska"(σ Salary=10,000(EMPLOYEE))

Properties of SELECT operation:…..

- The *SELECT* operation is ***unary***. It operates on only one relation.
- Selection conditions are applied to each tuple *individually* in the relation. Hence the condition cannot be span more than one tuple.
- The degree of the relation resulting from $\sigma_{\mathbf{C}}(R)$ is the same as that of *R* i.e. $|\sigma_{\mathbf{C}}(R)| \leq |R|$

Unary Relational Operations

- **PROJECT Operation :-** This operation selects certain ***columns*** from the table and discards the other columns. The PROJECT creates a vertical partitioning – one with the needed columns (attributes) containing results of the operation and other containing the discarded Columns. The general form of the project operation is:

$$\pi_{<\text{attribute list}>}(R)$$

- **Consider an example**:
- To list each employee's name & salary, we can use the PROJECT operation as;

$$\pi_{\text{Name, Salary}}(\text{EMPLOYEE})$$

Employee

Eid	Name	Salary	Address	Dno
011	Adam	20000	Alaska	4
022	Hilton	30000	Nevada	2

- To retrieve Eid and Name of Employees having salary equal to 20000 and who lives in Alaska, following notation is used:

$$\pi_{\text{Eid, Name}}\ (\sigma_{\text{Salary =20,000 AND Address="Alaska"}}\ (\text{EMPLOYEE}))$$

Or,

$$\text{temp}=\sigma_{\text{Salary =20,000 AND Address="Alaska"}}\ (\text{EMPLOYEE})$$

$$\text{result}=\pi_{\text{Eid, Name}}\ (\text{temp})$$

Unary Relational Operations

- **Consider an example**:
- To list each employee's name & salary, we can use the PROJECT operation as;

$$\pi_{\text{Name, Salary}}(\text{EMPLOYEE})$$

Name	Salary
Adam	20000
Hilton	30000

Employee

Eid	Name	Salary	Address	Dno
011	Adam	20000	Alaska	4
022	Hilton	30000	Nevada	2

- To retrieve Eid and Name of Employees having salary equal to 20000 and who lives in Alaska, following notation is used:

$$\pi_{\text{Eid, Name}}\ (\sigma_{\text{Salary =20,000 AND Address="Alaska"}}\ (\text{EMPLOYEE}))$$

Or,

$$\text{TEMP}=\sigma_{\text{Salary =20,000 AND Address="Alaska"}}\ (\text{EMPLOYEE})$$

$$\text{RESULT}=\pi_{\text{Eid, Name}}\ (\text{TEMP})$$

Unary Relational Operations

Consider an example:

- To list each employee's name & salary, we can use the PROJECT operation as;

$$\pi_{\text{Name, Salary}}(\text{EMPLOYEE})$$

- To retrieve Eid and Name of Employees having salary equal to 20000 and who lives in Alaska, following notation is used:

$$\pi_{\text{Eid, Name}}\ (\sigma_{\text{Salary =20,000 AND Address="Alaska"}}\ (\text{EMPLOYEE}))$$

Or,

$$\text{TEMP}=\sigma_{\text{Salary =20,000 AND Address="Alaska"}}\ (\text{EMPLOYEE})$$

$$\text{RESULT}=\pi_{\text{Eid, Name}}\ (\text{temp})$$

EMPLOYEE

Eid	Name	Salary	Address	Dno
011	Adam	20000	Alaska	4
022	Hilton	30000	Nevada	2

RESULT

Eid	Name
011	Adam

TEMP

Eid	Name	Salary	Address	Dno
011	Adam	20000	Alaska	4

Unary Relational Operations

- **PROJECT Operation :-**
- Properties of PROJECT operation:
  - The number of tuples in the result of projection $\pi_{\text{<list>}}(R)$ is always less or equal to the number of tuples in R.
  - The *PROJECT* operation is ***unary***. It operates on only one relation.
- The PROJECT operation removes any duplicate tuples. So the result of the PROJECT operation is a set of distinct tuples, and hence a valid relation. This is known as **duplicate elimination**.
  - *PROJECT* is not commutative. i.e. $\pi_{\text{<list1>}}(\pi_{\text{<list2>}}(R)) = \pi_{\text{<list1>}}(R)$ as long as <list2> contains the attributes in <list1>. Else the operation is an incorrect expression.

Unary Relational Operations

- **RENAME Operation :-** The rename operation can rename either the relation name or the attribute name or both. ρ is used to represent the rename operation. The general Rename operation can be expressed by any of the following forms:
  - $\rho_{S\,(B_1, B_2, \ldots, B_n)}(R)$ is a renamed relation S based on R with column names $B_1, B_2, \ldots\ldots, B_n$.
  - $\rho_S(R)$ is a renamed relation S based on R (which does not specify column names).
  - $\rho_{(B_1, B_2, \ldots, B_n)}(R)$ is a renamed relation with column names $B_1, B_2, \ldots..B_n$ which does not specify a new relation name.
- **Consider an example**:
- Retrieve name and salary of employee who work in department 2.

$$\rho_{RESULT}\,(\pi_{\textbf{Name, Salary}}((\sigma_{\textbf{DNO = 2}}\,\textbf{(EMPLOYEE)}))$$

Unary Relational Operations

- **RENAME Operation :-**
- Alternatively, we can explicitly show the sequence of operations, giving a name to each intermediate relation, and using the **assignment operation,** denoted by ← (left arrow). It is sometimes simpler to break down a complex sequence of operations by specifying intermediate result relations than to write a single relational algebra expression.

- Retrieve name and salary of employee who work in department 2.

$$\mathbf{TEMP \leftarrow \sigma_{DNO=2}(EMPLOYEE)}$$

$$\mathbf{RESULT \leftarrow \pi_{Name,\ Salary}(TEMP)}$$

Unary Relational Operations

- **Consider an example**:
- **Retrieve name and salary of employee who work in department 2.**

$$\rho_{RESULT\ (Emp\_name,\ Emp\_salary)}\ (\pi_{\textbf{Name, Salary}}((\sigma_{\textbf{DNO = 2}}\ \textbf{(EMPLOYEE)}))\ )$$

EMPLOYEE

Eid	Name	Salary	Address	Dno
011	Adam	20000	Alaska	4
022	Hilton	30000	Nevada	2
033	Ram	25000	Ktm	2

RESULT

Emp_name	Emp_salary
Hilton	30000
Ram	25000

$$\rho_{RESULT1}\ (\pi_{\textbf{Name, Salary}}((\sigma_{\textbf{DNO = 2}}\ \textbf{(EMPLOYEE)}))\ )$$

RESULT1

Name	Salary
Hilton	30000
Ram	25000

Unary Relational Operations

- **Consider an example**:
- **Retrieve name and salary of employee who work in department 2.**

$\rho_{RESULT}(\pi_{Name, Salary}((\sigma_{DNO = 2}(EMPLOYEE))))$

Or, **TEMP** $\leftarrow \sigma_{DNO=2}$**(EMPLOYEE);**

RESULT $\leftarrow \pi_{Name, Salary}$**(TEMP)**

Return employees working in department no 2 ------- TEMP$\leftarrow \sigma_{DNO = 2}$**(EMPLOYEE)**

Eid	Name	Salary	Address	Dno
022	Hilton	30000	Nevada	2
033	Ram	25000	Ktm	2

TEMP

Employee

Eid	Name	Salary	Address	Dno
011	Adam	20000	Alaska	4
022	Hilton	30000	Nevada	2
033	Ram	25000	Ktm	2

Return name and salary employees working in department no 2

Result $\leftarrow \pi_{Name, Salary}$**(Temp)**

Name	Salary
Hilton	30000
Ram	25000

RESULT

Return Eid and Address employees working in department no 2

Result1 $\leftarrow \pi_{Eid, Address}$ **(Temp)**

Eid	Address
022	Nevada
033	Ktm

RESULT1

Relational Algebra Operations from Set Theory

- **The UNION, INTERSECTION, and SET DIFFERENCE operation:**
  - These operations are binary operations.
  - The two relations on which any of the three operations are applied must have the same type of tuples; this condition is called ***union compatibility***. Two relations $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_n)$ are said to be ***union or type compatible*** if they have the same degree n and if $dom(A_i)=dom(B_i)$, for $1 <= i <= n$. This means that the two relations have the same no. of attributes, and each corresponding pair of attributes has the same domain.

    (Field names are not used in defining union compatibility)

Operations from Set Theory

- **UNION Operation :-** For any two union-compatible relations R and S, the union operation denoted by $R \cup S$ is a relation that includes all the tuples that are either in R or in S or in both R and S. Here duplicate tuples are eliminated, if any. So it can be defined as:

$$R \cup S = \{t \mid t \in R \text{ or } t \in S\}$$

- For Example:

Student

Fname	Lname
Bryan	Adam
Michal	Clark
Susan	Yao

Teacher

Fname	Lname
Ramez	Elmsari
Michal	Clark
Jimmy	Wang

Student ∪ Teacher

Fname	Lname
Bryan	Adam
Michal	Clark
Susan	Yao
Ramez	Elmsari
Jimmy	Wang

Operations from Set Theory

- **INTERSECTION Operation :-** For any two union-compatible relations R and S, the intersection operation denoted by $R \cap S$ is a relation that includes all the tuples that are in both R and S. So it can be defined as:

$$R \cap S = \{t \mid t \in R \text{ and } t \in S\}$$

- For Example:

Student

Fname	Lname
Bryan	Adam
Michal	Clark
Susan	Yao

Teacher

Fname	Lname
Ramez	Elmsari
Michal	Clark
Jimmy	Wang

Student $\cap$ Teacher

Fname	Lname
Michal	Clark

Operations from Set Theory

- **SET DIFFERENCE (Minus) Operation :-** For any two union-compatible relations R and S, the set difference operation denoted by R - S is a relation that includes all the tuples that are in R but not in S. So it can be defined as:

$$R - S = \{t \mid t \in R \textbf{ and } t \notin S\}$$

- For Example:

Student

Fname	**Lname**
Bryan	Adam
Michal	Clark
Susan	Yao

Teacher

Fname	**Lname**
Ramez	Elmsari
Michal	Clark
Jimmy	Wang

Student - Teacher

Fname	**Lname**
Bryan	Adam
Susan	Yao

Operations from Set Theory

- **Properties of Union, Intersection, Set Difference:-**
  - Both UNION & INTERSECTION are commutative while SET DIFFERENCE is not.

    i.e. **$R \cup S = S \cup R$, and $R \cap S = S \cap R$**

    But, $R - S \neq S - R$
  - Both UNION and INTERSECTION are associative.

    i.e. **$R \cup (S \cup T) = (R \cup S) \cup T$**

    And, $(R \cap S) \cap T = R \cap (S \cap T)$

Note:

For any two type compatible relations, R_1 & R_2. *The resulting relation for* $R_1 \cup R_2$, $R_1 \cap R_2$, *or* $R_1 - R_2$ *has the same attribute names as the* ***first*** *operand relation* R_1 *(by convention).*

Operations from Set Theory

Note:

For any two type compatible relations, R_1 & R_2. *The resulting relation for* $R_1 \cup R_2$, $R_1 \cap R_2$, *or* $R_1 - R_2$ *has the same attribute names as the* ***first*** *operand relation* R_1 *(by convention).*

Student

Fname	**Lname**
Bryan	Adam
Michal	Clark
Susan	Yao

Teacher

Firstname	**Lastname**
Ramez	Elmsari
Michal	Clark
Jimmy	Wang

Student $\cap$ Teacher

Fname	**Lname**
Michal	Clark

Operations from Set Theory

Note:

For any two type compatible relations, R_1 & R_2. *The resulting relation for* $R_1 \cup R_2$, $R_1 \cap R_2$, *or* $R_1 - R_2$ *has the same attribute names as the* ***first*** *operand relation* R_1 *(by convention).*

Student

Fname	**Lname**
Bryan	Adam
Michal	Clark
Susan	Yao

Teacher

Firstname	**Lastname**
Ramez	Elmsari
Michal	Clark
Jimmy	Wang

Teacher $\cap$ Student

Firstname	**Lastname**
Michal	Clark

Operations from Set Theory

Student

Fname	Lname	Adress
Ramez	Elmsari	Ktm

Teacher

Firstname	Lastname
Ramez	Elmsari
Michal	Clark
Jimmy	Wang

Manager

Fname	Mid
Ramez	011

Here *Teacher{∩, ∪, - }Student* is not possible. Since relations student and teacher are not union compatible. Since number of attributes are not same.

Here *Teacher {∩, ∪, - } Manager* is not possible. Since relations manager and teacher are not union compatible. Since domain of corresponding attributes is not same i.e. domain of Lastname is string while domain of Mid is integer.

Binary Relational Operations

- **Cartesian Product Operation :-**
  - Often known as cross join or Cartesian join.
  - The relations on which it is to be applied do not have to be union compatible.
  - This operation is used to combine tuples from two relations in combinatorial fashion.
  - It is denoted by $\times$.
  - For any two relations $R(A_1, A_2, \ldots, A_n)$ & $S(B_1, B_2, \ldots, B_m)$ the cross product $R \times S$ is a relation Q with degree n + m, $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$ in the same order as the attributes appear in R & S. The resulting relation Q has one tuple for each combination of tuples—one from R and one from S. So here we will have $n \times m$ tuples.

Binary Relational Operations

- **Cartesian Product Operation :-**
  - Consider an Example:

Employee

SSN	Fname	Lname	Dno
111	Bryan	Adams	1
222	John	Smith	2
333	Ravi	Sastri	2

Department

Dno	Dname	MgrSSN
1	HRM	111
2	Account	222

Employee × Department

Employee.SSN	Employee.Fname	Employee.Lname	Employee.Dno	Department.Dno	Department.Dname	Departmen
111	Bryan	Adams	1	1	HRM	111
111	Bryan	Adams	1	2	Account	222
222	John	Smith	2	1	HRM	111
222	John	Smith	2	2	Account	222
333	Ravi	Sastri	2	1	HRM	111
333	Ravi	Sastri	2	2	Account	222

Binary Relational Operations

- **JOIN Operation :-**
  - This operation is used to combine related tuples from two relations into single tuples. It is very useful for any relational DB with more than a single relation because it allows us to process relationships among relations. JOIN operation is denoted by $\bowtie$.
  - The general form of a join operation on two relations R (A_1, A_2, . . ., A_n) and S(B_1, B_2, . . ., B_m) is $R \bowtie_{\text{<join condition>}} S$.
  - Each tuple in $R \bowtie S$ is a combination of one tuple from R & one from S whenever the combination satisfies the *join condition*. The join condition is specified on attributes from two relations R & S and is evaluated for each combination of tuples.
  - Thus in join, only the condition satisfying tuples appear in result, whereas in Cartesian product all the combination of tuples are included in the result.

Binary Relational Operations

- **Join Operation :-**
  - Consider an Example:

Employee

SSN	Fname	Lname	Dno
111	Bryan	Adams	1
222	John	Smith	2
333	Ravi	Sastri	2

Department

Dno	Dname	MgrSSN
1	HRM	111
2	Account	222

Employee $\bowtie_{\text{Employee.Dno=Department.Dno}}$ Department

Employee.SSN	Employee.Fname	Employee.Lname	Employee.Dno	Department.Dno	Department.Dname	Departmen
111	Bryan	Adams	1	1	HRM	111
222	John	Smith	2	2	Account	222
333	Ravi	Sastri	2	2	Account	222

Binary Relational Operations

- **JOIN Operation :-**
- Example:
  - **To select the name of Managers of each departments. The relational algebra query can be written as;**

    $Mgrlist \leftarrow Employee \bowtie_{SSN=MgrSSN} Department$

    $Result \leftarrow \pi_{FName,\ LName}(Mgrlist)$

  - **To find the name of Managers having lasts name "Adams". The relational algebra query can be written as;**

    $Mgrlist \leftarrow Employee \bowtie_{SSN=MgrSSN} Department$

    $Temp \leftarrow \sigma_{LName="Adams"}(Mgrlist)$

    $Result \leftarrow \pi_{FName,\ LName}(Temp)$

Binary Relational Operations

- **Join Operation :- To select the name of Managers of each departments. The relational algebra query can be written as;**

Employee

SSN	Fname	Lname	Dno
111	Bryan	Adams	1
222	John	Smith	2
333	Ravi	Sastri	2

Department

Dno	Dname	MgrSSN
1	HRM	111
2	Account	222

Mgrlist ← Employee $\bowtie_{SSN=MgrSSN}$ Department

Employee.SSN	Employee.Fname	Employee.Lname	Employee.Dno	Department.Dno	Department.Dname	Departmen
111	Bryan	Adams	1	1	HRM	111
222	John	Smith	2	2	Account	222

Result ← $\pi_{FName, LName}$(Mgrlist)

Employee. Fname	Employee.Lname
Bryan	Adams
John	Smith

Equivalently:

$\pi_{FName, LName}$(Employee $\bowtie_{SSN=MgrSSN}$ Department)

Binary Relational Operations

- **Join Operation :- To find the name of Managers having lasts name "Adams".**

  The relational algebra query can be written as

$$\text{Mgrlist} \leftarrow \text{Employee} \bowtie_{\text{SSN=MgrSSN}} \text{Department}$$

Employee.SSN	Employee.Fname	Employee.Lname	Employee.Dno	Department.Dno	Department.Dname	Departmen
111	Bryan	Adams	1	1	HRM	111
222	John	Smith	2	2	Account	222

$$\text{Temp} \leftarrow \sigma_{\text{LName="Adams"}} (\text{Mgrlist})$$

Employee.SSN	Employee.Fname	Employee.Lname	Employee.Dno	Department.Dno	Department.Dname	Department
111	Bryan	Adams	1	1	HRM	111

$$\text{Result} \leftarrow \pi_{\text{FName, LName}}(\text{Temp})$$

Employee. Fname	Employee.Lname
Bryan	Adams

Equivalently:

$$\pi_{\text{FName, LName}}(\sigma_{\text{LName="Adams"}} (\text{Employee} \bowtie_{\text{SSN=MgrSSN}} \text{Department}))$$

Binary Relational Operations

- **Theta JOIN Operation :-**
- Join operation between relations R and S with join condition $A_i \theta B_i$; A_i and B_i are attributers belonging to relations R and S respectively and have the same domain. And θ={ =, <, >, <=, >=, !=}.

- Thus, join operation with generalized join condition is the **theta join** and has following notation;

$$R \bowtie_{Ai\ \theta\ Bi} S$$

Binary Relational Operations

- **EQUIJOIN Operation :-**
- The join operation where the join condition involves only equality comparison is called equijoin. Here only the comparison operator used is "=".

- In the result of an EQUIJOIN, we always have one or more pairs of attributes (whose names need not be identical) that have *identical values* in every tuple.

- The EQUIJOIN has following representation

$R \bowtie_{Ai=Bi} S$; where A_i and B_i are attributes of relations R & S respectively.

Binary Relational Operations

- **Natural Join Operation :-**
- In equijoin operation, the resulting relation have a pair or more attributes with identical value so having multiple fields in a tuple with same value is unnecessary. To get rid of it, a new join type called **natural join** is introduced. This natural join is denoted by * or ⋈ .
- The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, have the same name in both relations. If this is not the case, a renaming operation is applied first.
- Example:
  - **To get records of Managers of each departments, the relational algebra query can be written as;**

$$\text{Mgrlist} \leftarrow \text{Employee} * \rho_{(\text{Dno, Dname, SSN})} \text{Department}$$

In this realtion Mgrlist, it will have just single entry SSN rather than having both SSN & MgrSSN as in equijoin, since MgrSSN in Department is renamed to SSN. Hence natural join will result.

- In general, natural join is performed by equating all attribute pairs that have the same name in two relations.

Binary Relational Operations

- **Natural Join Operation :-** **Select name of employees and their departments.**

Employee

SSN	Fname	Lname	Dno
111	Bryan	Adams	1
222	John	Smith	2
333	Ravi	Sastri	2

Department

Dno	Dname	MgrSSN
1	HRM	111
2	Account	222

Mgrlist ← Employee * Department

SSN	Fname	Lname	Dno	Dname	MgrSSN
111	Bryan	Adams	1	HRM	111
222	John	Smith	2	Account	222
333	Ravi	Sastri	2	Account	222

Result←$\pi_{FName,\ LName,\ Dname}$ (Mgrlist)

Fname	Lname	Dname
Bryan	Adams	HRM
John	Smith	Account
Ravi	Sastri	Account

Equivalently:

$\pi_{FName,\ LName,\ Dname}$ (Employee * Department)

Consider few Examples:

- Suppose we have following relations;

Employee (Fname, Minit, Lname, SSN, Bdate, Address, Sex, Salary, SuperSSN, Dno)

Department (Dname, Dnumber, MgrSSN, Mgrsstartdate)

Project (Pnumber, Pname, Plocation, Dnum)

Works_on (ESSN, Pno, Hours)

Dep_Locations (Dnumber, Dlocation)

Dependent (ESSN, Dep_name, Sex, Bdate, Relationship)

Consider few Examples:

- Select employees having salary greater than 30000
- Select project name and project location of projects
- Select all employees having salary less than 1000 and address "KTM"
- Select project name and project location of projects having project name "Construction"

Consider few Examples:

- Select employees having salary greater than 30000

  $\sigma_{Salary>30000}$ (Employee)

- Select project name and project location of projects

  $\pi_{Pname,\ Plocation}$ (Project)

- Select all employees having salary less than 1000 and address "KTM"

  $\sigma_{Salary<30000\ AND\ Address="KTM"}$ (Employee)

- Select project name and project location of projects having project name "Construction"

  $\pi_{Pname,\ Plocation}$ ($\sigma_{Pname="Construction"}$ (Project))

- **Lets try few more queries:**
  - **Retrieve the name and address of all employee who works for the "Research Department".**
  - **Retrieve the list of project numbers that "John Smith" works on.**

- **Lets try few more queries:**
  - **Retrieve the name and address of all employee who works for the "Research Department".**

$Research\_dept \leftarrow \sigma_{DName="Research"}(Department)$

$REmp \leftarrow Research\_dept \bowtie_{Dnumber=Dno} (Employee)$

$Result \leftarrow \pi_{FName\ LName,\ Address}(REmp)$

Or,

$REmp \leftarrow Department \bowtie_{Dnumber=Dno} (Employee)$

$Research\_dept \leftarrow \sigma_{DName="Research"}(REmp)$

$Result \leftarrow \pi_{FName\ LName,\ Address}(Research\_dept)$

Or,

$\pi_{FName\ LName,\ Address}(\sigma_{DName="Research"}(Department \bowtie_{Dnumber=Dno} (Employee)))$

  - **Retrieve the list of project numbers that "John Smith" works on.**

$Emp\_John \leftarrow \sigma_{FName="John"\ AND\ LName="Smith"}(Employee)$

$John\_Proj \leftarrow Emp\_John \bowtie_{SSN=ESSN} (Works\_on)$

$Result \leftarrow \pi_{Pno}(John\_Proj)$

Or,

$Emp\_John \leftarrow \pi_{SSN}(\sigma_{FName="John"\ AND\ LName="Smith"}(Employee))$

$John\_Proj \leftarrow (\rho_{(ESSN)} Emp\_John) * (Works\_on)$

$Result \leftarrow \pi_{Pno}(John\_Proj)$

- **Retrieve the name of employee who have no dependents.**

- **Retrieve the list names of employees and projects they are working on**

- **Retrieve the name of employee who have no dependents.**

  $Emp\_List \leftarrow \pi_{SSN}(Employee)$

  $Emp\_Depend \leftarrow \pi_{ESSN}(Dependent)$

  $Emp\_Nodepend \leftarrow Emp\_List - Emp\_Depend$

  $Emp \leftarrow Employee * Emp\_Nodepend$

  $Result \leftarrow \pi_{FName,\ Lname}(Emp)$

- **Retrieve the list names of employees and projects they are working on**

  $Working\_Proj \leftarrow Project \bowtie_{Pnumber=Pno} (Works\_on)$

  $Employee\_Proj \leftarrow Working\_Proj \bowtie_{ESSN=SSN} (Employee)$

  $Result \leftarrow \pi_{FName,\ Lname,\ Pname}(Employee\_Proj)$

Complete Set of Relational Operations

- The set of operations including ***select*** σ, ***project*** π , ***union*** $\cup$, ***set difference*** **-** , ***and Cartesian product*** ***X*** is called a complete set because any other relational algebra expression can be expressed by a combination of these five operations.

Binary Relational Operations

- **Division Operation :-**
- Let *r* and *s* be relations on schemas R and S respectively where

  $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$ & $S = (B_1, \ldots, B_n)$

  The result of r ÷ s is a relation on schema

  $R \div S = (A_1, \ldots, A_m)$
- Alternatively to define the operation, consider two relations A & B in which A has two fields x & y and B has just one field y with the same domain as in A, then we define A ÷ B as a set of all x values in unary tuple such that for every y value in B, there is a tuple (x, y) in A.
- **Eg:-**

A

X	Y
x1	y1
x2	y1
x1	y2
x2	y2
x3	y1
x4	y2

B

Y
y1
y2

A ÷ B

X
x1
x2

Binary Relational Operations

- **Division Operation :-**
- Generally, Division operation is suited to queries that include the phrase "for all".
- **Eg:** ***Retrieve the names of employees who work on all the projects that 'John Smith' works on.***
- First retrieve the list of project numbers that 'John Smith' works on in the intermediate relation SMITH_PNOS:

  $SMITH \leftarrow \sigma_{Fname='John' \textbf{ AND } Lname='Smith'}(EMPLOYEE)$

  $SMITH\_PNOS \leftarrow \pi_{Pno}(WORKS\_ON \text{ JOIN } Essn=Ssn \text{ } SMITH)$
- Next, create a relation that includes a tuple <Pno, Essn> whenever the employee whose Ssn is Essn works on the project whose number is Pno in the intermediate relation SSN_PNOS:

  $SSN\_PNOS \leftarrow \pi_{Essn,\ Pno}(WORKS\_ON)$
- Finally, apply the DIVISION operation to the two relations, which gives the desired employees' Social Security numbers:

  $SSNS(Ssn) \leftarrow SSN\_PNOS \div SMITH\_PNOS$

  $RESULT \leftarrow \pi_{Fname,\ Lname}(SSNS * EMPLOYEE)$

Binary Relational Operations

- **Division Operation :-**

The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

– **Find the names of employees who work on *all* the projects controlled by department number 5.**

DEPT5_PROJS ← ρ(Pno)(πPnumber(σDnum=5(PROJECT)))

EMP_PROJ ← ρ(Ssn, Pno)(πEssn, Pno(WORKS_ON))

RESULT_EMP_SSNS ← EMP_PROJ ÷ DEPT5_PROJS

RESULT ← πLname, Fname(RESULT_EMP_SSNS * EMPLOYEE)

– In this query, we first create a table DEPT5_PROJS that contains the project numbers of all projects controlled by department 5. Then we create a table EMP_PROJ that holds (Ssn, Pno) tuples, and apply the division operation. Notice that we renamed the attributes so that they will be correctly used in the division operation. Finally, we join the result of the division, which holds only Ssn values, with the EMPLOYEE table to retrieve the Fname, Lname attributes from EMPLOYEE.

Additional Relational Operations

- **Generalized Projection:-**
- The generalized projection operation extends the projection operation by allowing functions of attributes to be included in the projection list. The generalized form can be expressed as:

$$\pi_{F1, F2, ..., Fn}(R)$$

- $F1, F2, \ldots, Fn$ are functions over the attributes in relation R and may involve arithmetic operations and constant values. This operation is helpful when developing reports where computed values have to be produced in the columns of a query result.

Additional Relational Operations

- **Generalized Projection:-**
- As an example, consider the relation

  EMPLOYEE (Ssn, Salary, Deduction, Years_service)
- A report may be required to show

  Net Salary = Salary – Deduction,

  Bonus = 2000 * Years_service, and

  Tax = 0.25 * Salary
- Then a generalized projection combined with renaming may be used as follows:

$$\text{REPORT} \leftarrow \rho_{(\text{Ssn, Net\_salary, Bonus, Tax})}(\pi_{\text{Ssn, Salary – Deduction, 2000 * Years\_service, 0.25 * Salary}}(\text{EMPLOYEE}))$$

Additional Relational Operations

- **Aggregate Functions and Groupings:-**
  - SUM, AVERAGE, MAXIMUM, MINIMUM, COUNT
  - Aggregate functions take a collection of values and return a single value as a result.
  - Aggregate function operations are defined using the symbol $\mathscr{F}$ (pronounced "script F"), or $\mathcal{G}$ (pronounced "calligraphic G").
  - The general form of ***aggregate operation*** in relational algebra

    $$G1, G2, \ldots, Gn \; \mathscr{F} \; F1(A1), F2(A2), \ldots, Fn(An) \; (E),$$

    where, E is any relational-algebra expression

    $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)

    Each F_i is an aggregate function

    Each A_i is an attribute name

Additional Relational Operations

- **Aggregate Functions and Groupings:-**
  - Consider an example;

R

A	B	C
A1	B1	7
A2	B2	3
A3	B3	10

$\mathcal{F}_{\mathbf{sum(c)}}$ (R)

Sum_C
20

  - **Now using aggregate function, the query to find the total number of employees and their average salary will be:**

$\mathcal{F}_{\text{COUNT(SSN), AVERAGE(Salary)}}$ (Employee)

The result of this query will be such as;

COUNT_SSN	AVERAGE_Salary
8	257876

Additional Relational Operations

- **Aggregate Functions and Groupings:-**
  - We may group the tuples in the relation by the value of some attributes and the apply aggregate function. For example:

    To retrieve Department number, total number of employees and their average salary, the query may be as

    $$\text{Dno}\mathcal{F}_{\text{COUNT(SSN), AVERAGE(Salary)}}\ (\text{Employee})$$

    The result of this query will be such as;

Dno	COUNT_SSN	AVERAGE_Salary
1	8	257876
2	10	300000

  - There are cases where we must eliminate multiple occurrences of a value before computing an aggregate function. For this, we use the function name with the addition of the hyphenated string **"distinct"** appended to the end of the function name (for example, **count-distinct**).

Additional Relational Operations

- **Outer Join:-**
- In NATURAL JOIN, tuples without a *matching* (or *related*) tuple are eliminated from the join result. Tuples with null in the join attributes are also eliminated. This amounts to loss of information. A set of operations, called ***outer joins***, can be used when we want to keep all the tuples in R, or all those in S, or all those in both relations in the result of the join, regardless of whether or not they have matching tuples in the other relation.
- The ***left outer join*** operation keeps every tuple in the ***first*** or ***left*** relation R in R⟕S; if no matching tuple is found in S, then the attributes of S in the join result are filled or "padded" with null values. A similar operation, ***right outer join***, keeps every tuple in the ***second*** or ***right*** relation S in the result of R⟖S; if no matching tuple is found in R, then the attributes of R in the join result are filled or "padded" with null values. While, the ***full outer join***, denoted by R⟗S keeps all tuples in both the left and the right relations in the result; when no matching tuples are found, they are padded with null values as needed.

Additional Relational Operations

- **Outer Join:-**
- **Example**

Employee

Eid	Ename	Address	Dno
1	Ram	KTM	111
2	Rita	PKR	222
3	Hari	KTM	333

Department

Dno	Dname
111	HRM
222	Admin
444	Account

Employee ⟕ Department

Eid	Ename	Address	Dno	Dname
1	Ram	KTM	111	HRM
2	Rita	PKR	222	Admin
3	Hari	KTM	333	NULL

Employee ⟖ Department

Eid	Ename	Address	Dno	Dname
1	Ram	KTM	111	HRM
2	Rita	PKR	222	Admin
NULL	NULL	NULL	444	Account

Employee ⟗ Department

Eid	Ename	Address	Dno	Dname
1	Ram	KTM	111	HRM
2	Rita	PKR	222	Admin
3	Hari	KTM	333	NULL
NULL	NULL	NULL	444	Account

Tuple Relational Calculus

- In tuple relational calculus, we write one **declarative** expression to specify a retrieval request; hence, there is no description of how, or *in what order*, to evaluate a query. A calculus expression specifies *what* is to be retrieved rather than *how* to retrieve it. Therefore, the relational calculus is considered to be a **nonprocedural** language. This differs from relational algebra, where we must write a *sequence of operations* to specify a retrieval request *in a particular order* of applying the operations; thus, it can be considered as a **procedural** way of stating a query.

Tuple Relational Calculus

- It has been shown that any retrieval that can be specified in the basic relational algebra can also be specified in relational calculus, and vice versa; in other words, the **expressive power** of the languages is *identical*. This led to the definition of the concept of a *relationally complete* language. A relational query language *L* is considered **relationally complete** if we can express in *L* any query that can be expressed in relational calculus.

Tuple Relational Calculus

- **Tuple Variables and Range Relations**
- The tuple relational calculus is based on specifying a number of **tuple variables**. Each tuple variable usually *ranges over* a particular database relation, meaning that the variable may take as its value any individual tuple from that relation. A simple tuple relational calculus query is of the form:

  $\{t \mid \text{COND}(t)\}$

- where t is a tuple variable and COND(t) is a conditional (Boolean) expression involving t that evaluates to either TRUE or FALSE for different assignments of tuples to the variable t. The result of such a query is the set of all tuples t that evaluate COND(t) to TRUE. These tuples are said to **satisfy** COND(t).

Tuple Relational Calculus

- **Tuple Variables and Range Relations**
- For example, to find all employees whose salary is above $50,000, we can write the following tuple calculus expression:

$$\{t \mid \text{EMPLOYEE}(t) \textbf{ AND } t.\text{Salary}>50000\}$$

- The condition EMPLOYEE(t) specifies that the **range relation** of tuple variable t is EMPLOYEE. Each EMPLOYEE tuple t that satisfies the condition t.Salary>50000 will be retrieved. Notice that t.Salary references attribute Salary of tuple variable t.
- The previous query retrieves all attribute values for each selected EMPLOYEE tuple t. To retrieve only *some* of the attributes—say, the first and last names—we write

$$\{t.\text{Fname}, t.\text{Lname} \mid \text{EMPLOYEE}(t) \textbf{ AND } t.\text{Salary}>50000\}$$

Tuple Relational Calculus

- **Tuple Variables and Range Relations**
- Retrieve the birth date and address of the employee (or employees) whose name is John B. Smith.

Tuple Relational Calculus

- **Tuple Variables and Range Relations**
- Retrieve the birth date and address of the employee (or employees) whose name is John B. Smith.

{*t*.Bdate, *t*.Address | EMPLOYEE(*t*) **AND** *t*.Fname='John' **AND** *t*.Minit='B' **AND** *t*.Lname='Smith'}

- In tuple relational calculus, we first specify the requested attributes *t*.Bdate and *t*.Address for each selected tuple *t*. Then we specify the condition for selecting a tuple following the bar (|)—namely, that *t* be a tuple of the EMPLOYEE relation whose Fname, Minit, and Lname attribute values are 'John', 'B', and 'Smith', respectively.

Tuple Relational Calculus

- **Expressions and Formulas in Tuple Relational Calculus**

- A general **expression** of the tuple relational calculus is of the form

  $\{t1.Aj, t2.Ak, \ldots, tn.Am \mid \mathbf{COND}(t1, t2, \ldots, tn, tn+1, tn+2, \ldots, tn+m)\}$

- where $t1, t2, \ldots, tn, tn+1, \ldots, tn+m$ are tuple variables, each Ai is an attribute of the relation on which ti ranges, and COND is a **condition** or **formula** of the tuple relational calculus.

Tuple Relational Calculus

- **The Existential and Universal Quantifiers**

- Two special symbols called **quantifiers** can appear in formulas; these are the **universal quantifier** (∀) and the **existential quantifier** (∃). They are used to **bound** a tuple variable.

- Informally, a tuple variable *t* is bound if it is quantified, meaning that it appears in an (∃*t*) or (∀*t*) clause; otherwise, it is free. Formally, we define a tuple variable.

Tuple Relational Calculus

- **Examples (With reference to schemas in slide number 34)**

- **Query 1.** List the name and address of all employees who work for the 'Research' department.

Q1: {*t*.Fname, *t*.Lname, *t*.Address | EMPLOYEE(*t*) **AND** (∃*d*)(DEPARTMENT(*d*) **AND** *d*.Dname='Research' **AND** *d*.Dnumber=*t*.Dno)}

- The *only free tuple variables* in a tuple relational calculus expression should be those that appear to the left of the bar (|). In Q1, *t* is the only free variable; it is then *bound successively* to each tuple. If a tuple *satisfies the conditions* specified after the bar in Q1, the attributes Fname, Lname, and Address are retrieved for each such tuple. The conditions EMPLOYEE(*t*) and DEPARTMENT(*d*) specify the range relations for *t* and *d*. The condition *d*.Dname = 'Research' is a **selection condition** and corresponds to a SELECT operation in the relational algebra, whereas the condition *d*.Dnumber = *t*.Dno is a **join condition** and is similar in purpose to the (INNER) JOIN operation used in relational algebra query in slide number 35.

Tuple Relational Calculus

- **Examples (With reference to schemas in slide number 34)**

- Several tuple variables in a query can range over the same relation. For example, to specify Q2 — for each employee, **retrieve the employee's first and last name and the first and last name of his or her immediate supervisor**—we specify two tuple variables *e* and *s* that both range over the EMPLOYEE relation:

Q2: {*e*.Fname, *e*.Lname, *s*.Fname, *s*.Lname | EMPLOYEE(*e*) **AND** EMPLOYEE(*s*) **AND** *e*.Super_ssn=*s*.Ssn}

Tuple Relational Calculus

- **Examples (With reference to schemas in slide number 34)**

- List the name of each employee who works on *some* project controlled by department number 5. In this case we need two join conditions and two existential quantifiers.

- List the names of employees who have no dependents.

Tuple Relational Calculus

- **Examples (With reference to schemas in slide number 34)**

- List the name of each employee who works on *some* project controlled by department number 5. In this case we need two join conditions and two existential quantifiers.

{*e*.Lname, *e*.Fname | EMPLOYEE(*e*) **AND** ((∃*x*)(∃*w*)(PROJECT(*x*) **AND** WORKS_ON(*w*) **AND** *x*.Dnum=5 **AND** *w*.Essn=*e*.Ssn **AND** *x*.Pnumber=*w*.Pno))}

- List the names of employees who have no dependents.

{*e*.Fname, *e*.Lname | EMPLOYEE(*e*) **AND** (**NOT** (∃*d*)(DEPENDENT(*d*) **AND** *e*.Ssn=*d*.Essn))}

Equivalently, following is same;

{*e*.Fname, *e*.Lname | EMPLOYEE(*e*) **AND** ((∀*d*)(**NOT**(DEPENDENT(*d*)) **OR** **NOT**(*e*.Ssn=*d*.Essn)))}

Domain Relational Calculus

- Domain calculus differs from tuple calculus in the *type of variables* used in formulas:
- Rather than having variables range over tuples, the variables range over single values from domains of attributes. To form a relation of degree *n* for a query result, we must have *n* of these **domain variables**—one for each attribute. An expression of the domain calculus is of the form

$$\{x1, x2, ..., xn \mid \text{COND}(x1, x2, ..., xn, xn+1, xn+2, ..., xn+m)\}$$

- where $x1, x2, \ldots, xn, xn+1, xn+2, \ldots, xn+m$ are domain variables that range over domains (of attributes), and COND is a **condition** or **formula** of the domain relational calculus.

Domain Relational Calculus

- **Examples**
- List the birth date and address of the employee whose name is ‘John B. Smith’.

{*u*, *v* | (∃*q*) (∃*r*) (∃*s*) (∃*t*) (∃*w*) (∃*x*) (∃*y*) (∃*z*) (EMPLOYEE(*qrstuvwxyz*) **AND** *q*=‘John’ **AND** *r*=‘B’ **AND** *s*=‘Smith’)}

- We need ten variables for the EMPLOYEE relation, one to range over each of the domains of attributes of EMPLOYEE in order. Of the ten variables *q*, *r*, *s*, … , *z*, only *u* and *v* are free, because they appear to the left of the bar and hence should not be bound to a quantifier. We first specify the *requested attributes*, Bdate and Address, by the free domain variables *u* for BDATE and *v* for ADDRESS. Then we specify the condition for selecting a tuple following the bar (|)—namely, that the sequence of values assigned to the variables *qrstuvwxyz* be a tuple of the EMPLOYEE relation and that the values for *q* (Fname), *r* (Minit), and *s* (Lname) be equal to ‘John’, ‘B’, and ‘Smith’, respectively.

Domain Relational Calculus

- **Examples**
- List the birth date and address of the employee whose name is ‘John B. Smith’.

$\{u, v \mid (\exists q)\ (\exists r)\ (\exists s)\ (\exists t)\ (\exists w)\ (\exists x)\ (\exists y)\ (\exists z)\ (\text{EMPLOYEE}(qrstuvwxyz)$ **AND** q=‘John’ **AND** r=‘B’ **AND** s=‘Smith’)$\}$

- Alternatively,

$\{u, v \mid \text{EMPLOYEE}(\text{‘John’}, \text{‘B’}, \text{‘Smith’}, t, u, v, w, x, y, z)\}$

Domain Relational Calculus

- **Examples**
- Retrieve the name and address of all employees who work for the 'Research' department.

{*q*, *s*, *v* | (∃*z*) (∃*l*) (∃*m*) (EMPLOYEE(*qrstuvwxyz*) **AND** DEPARTMENT(*lmno*) **AND** *l*='Research' **AND** *m*=*z*)}

- A condition relating two domain variables that range over attributes from two relations, such as *m* = *z* in above query, is a **join condition on dno**, whereas a condition that relates a domain variable to a constant, such as *l* = 'Research', is a **selection condition**.

Domain Relational Calculus

- **Examples**
- List the names of employees who have no dependents.

Domain Relational Calculus

- **Examples**
- List the names of employees who have no dependents.

{*q*, *s* | (∃*t*)(EMPLOYEE(*qrstuvwxyz*) **AND** (**NOT**(∃*l*)(DEPENDENT(*lmnop*) **AND** *t*=*l*)))}

Domain Relational Calculus

- **Examples**
- List the names of managers who have at least one dependent.

{*s*, *q* | (∃*t*)(∃*j*)(∃*l*)(EMPLOYEE(*qrstuvwxyz*) **AND** DEPARTMENT(*hijk*) **AND** DEPENDENT(*lmnop*) **AND** *t*=*j* **AND** *l*=*t*)}

Here, t is SSN in Employee, j is MGRSSN in Department, l is ESSN is Dependent.

Putting Altogether

- **Examples**
- Retrieve the name and address of all employees who work for the 'Research' department.

  Employee (Fname, Minit, Lname, SSN, Bdate, Address, Sex, Salary, SuperSSN, Dno)

  Department (Dname, Dnumber, MgrSSN, Mgrsstartdate)

- Relational Algebra

$\pi_{\text{FName LName, Address}}(\sigma_{\text{DName="Research"}}(\text{Department} \bowtie_{\text{Dnumber=Dno}} (\text{Employee})))$

- Tuple Calculus

{*t*.Fname, *t*.Lname, *t*.Address | EMPLOYEE(*t*) **AND** (∃*d*)(DEPARTMENT(*d*) **AND** *d*.Dname='Research' **AND** *d*.Dnumber=*t*.Dno)}

- Domain Calculus

{*q*, *s*, *v* | (∃*z*) (∃*l*) (∃*m*) (EMPLOYEE(*qrstuvwxyz*) **AND** DEPARTMENT(*lmno*) **AND** *l*='Research' **AND** *m*=*z*)}