Chapter 5

The relational algebra & relational calculus

Unary relational operations (SELECT & PROJECT)

The relational algebra operations that take single relation as operand are called unary operations. Most common unary operations are SELECT and PROJECT operations.

1. SELECT operation :-

The select operation is used to choose a subset of tuples from a relation that satisfies a selection condition. It works as a filter that keeps only those tuples that satisfy the selection condition. The select operation partitions the given set of tuples into two sets, one that satisfy the condition & other that don't. In relational algebra operation, σ notation is used to represent the SELECT operation.

The SELECT operation is expressed by using the following syntax;

$$\sigma_{<selection\_condition>}(R)$$

where σ (sigma) is used to represent SELECT operator and the <select_condition> is a Boolean expression or condition specified on the attributes of the relation R.

Eg: $\sigma_{salary > 40,000}(EMPLOYEE)$ ⇒ selects the tuples from EMPLOYEE relation whose salary is greater than 40,000/-.

$\sigma_{salary > 50000 \text{ AND } age < 30}(EMPLOYEE)$ ⇒ selects the tuples from EMPLOYEE relation where salary is more than 50,000 & age is less than 30 yrs.

In the selection condition we can combine a no. of predicates using the Boolean operators AND, OR and NOT.

The result of a selection operation is also a relation of same degree (no. of attributes) but with the tuples satisfying the selection condition. The no. of tuples after selection operation is always less than or equal to that in the operand relation.

The selection operation satisfies the following conditions.

(i) Degree of the relation is preserved.
ie. $Deg(\sigma_{<condition>}(R)) = Deg(R)$

(ii) Number of tuples may reduce.
ie. $|\sigma_{<condition>}(R)| \leq |R|$

(iii) Selection operation is commutative.
ie. $\sigma_{<cond-1>}(\sigma_{<cond-2>}(R)) = \sigma_{<cond-2>}(\sigma_{<cond-1>}(R))$

(iv) Selection operations follow the cascade combine rule.
ie. $\sigma_{<cond1>}(\sigma_{<cond-2>}(\ldots\sigma_{<condn>}(R))\ldots) = \sigma_{<cond1> \text{ AND } <cond2>\ldots<condn}$

2(iv) PROJECT Operation :-

Project operation is used to choose the tuples corresponding to certain attributes of a relation R. It operates as a filter for the attributes in a relation. Like SELECT operation, it is also an unary operation that takes a single relation as operand. The PROJECT operation selects all the tuples over the certain attributes only.

The SELECT operation can be visualized as a horizontal filter on tuples whereas PROJECT operation can be visualized as a vertical filter over attributes. We use the following syntax for PROJECT operation:

$\Pi_{<attribute\_list>}(R)$

Here, R may be a single relation or an algebraic expression resulting an operation. The result of the PROJECT operation is a relation having the attributes specified in the <attribute_list> in the same order as they appear in R.

The PROJECT operation satisfies the following properties:

(i) The degree of the resultant relation is less than or equal to the degree of the operand relation.

ie. $Deg(\Pi_{<attr\_list>}(R)) \leq Deg(R)$

(ii) PROJECT operation eliminates the duplicate tuple from the resultant relation. This property is observed when the projection includes the non-key attributes only.
Eg: $\Pi_{(name)}$ (EMPLOYEE) ⇒ A relation including the distinct names from the relation EMPLOYEE.

(iii) Number of tuples may reduce.
ie. $|\Pi_{<attr\_list>}(R)| \leq |R|$.
The no. of tuples remains same if the attribute list includes a key attribute, otherwise the no. of tuples may reduce due to duplicate tuple elimination.

(iv) $\Pi_{<list-1>}(\Pi_{<list-2>}(R)) = \Pi_{<list-1>}(R)$ only if <list-2> contains the attributes in <list-1> or more.

Note that the PROJECT operation in relational algebra is equivalent to SELECT DISTINCT operation in SQL.

Example: $\Pi_{name, add, contact}$ (EMPLOYEE) ⇒ a relation including the attributes name, address and contact with distinct tuples.

Sequences of Operations & the RENAME operation

In relational algebra ~~operations~~ expression, we can combine a no. of operations to form a complex & nested query. For eg we can combine SELECT & PROJECT operations as:

$$\Pi_{Fname, Lname, Salary}(\sigma_{Salary > 50,000}(EMPLOYEE))$$

We can breakdown the nested expression as shown above into a no. of simpler expressions by using the rename operation. The RENAME operation is denoted by the symbol ρ (rho) and is used by using any of the following syntaxes:

- $\rho_{S(B1, B2, \cdots, Bn)}(R)$ ⇒ To rename the resultant relation as well as attributes.
- $\rho_S(R)$ ⇒ To rename the resultant relation only.
- $\rho_{(B1, B2, \cdots, Bn)}(R)$ ⇒ To rename the attributes only.

Where, ρ is the rename operator, S is the new relation name & B1, B2, ---, Bn are the new attribute names.

Example: Consider an expression

$$\Pi_{name, post, salary}(\sigma_{post = officer}(EMPLOYEE))$$

We can decompose the above nested as follows:

$$\rho_{OFFICER}(\sigma_{post=officer}(EMPLOYEE))$$
$$\Pi_{name, post, salary}(OFFICER)$$

Eg: We can also rename the attributes ~~only~~ as

$$\rho_{OFFICER(n,p,s)}(\sigma_{post=officer}(EMPLOYEE))$$
$$\Pi_{n,p,s}(OFFICER)$$

Similarly, we can also rename the attributes only. But it may not be required in normal situations.

We can also use assignment operator ($\leftarrow$) to rename.

Eg:
$$OFFICER \leftarrow \sigma_{post=officer}(EMPLOYEE)$$
$$\Pi_{name, post, salary}(OFFICER)$$

OR

$$OFFICER(n,p,s) \leftarrow \sigma_{post=officer}(EMPLOYEE)$$
$$\Pi_{n,p,s}(OFFICER)$$

Relational algebra operations from set theory

1. UNION Operation :-

It is a binary operation that is used to combine the tuples from two compatible relations. Two relations $R(A_1, A_2, \cdots, A_n)$ and $S(B_1, B_2, \cdots, B_n)$ are said to be union compatible if:

(i) Relations R and S are of same degree (ie. no. of attributes is equal).

(ii) Corresponding attributes of R & S have the same domain (ie. $dom(A_i) = dom(B_i) \; \forall \; i = 1 \text{ to } n$).

The union operation is represented by "∪" operator and the resultant relation R∪S contains the tuples in R or S or in both. The duplicate tuples are eliminated.

Example: Consider a relation EMPLOYEE (eid, name, add, contact, salary, post). If we want to get a set of employee having post officer or helper, we can use union operation as:

$$RESULT1 \leftarrow \sigma_{post = officer}(EMPLOYEE)$$
$$RESULT2 \leftarrow \sigma_{post = helper}(EMPLOYEE)$$
$$RESULT3 \leftarrow RESULT1 \cup RESULT2$$
$$RESULT \leftarrow \Pi_{name, add}(RESULT3)$$

2. Intersection Operation :-

It is a binary operation that is used to retrieve the tuples that are common to both the relations. INTERSECTION operation can be performed between two union-compatible relations. This operation is represented by "$\cap$" operator as in set theory.

Example: consider a relation STUDENT (sid, name, add, contact, gender, nationality), the following expressions give the set of female students name who are from India.

$$RESULT1 \leftarrow \sigma_{gender = female}(STUDENT)$$
$$RESULT2 \leftarrow \sigma_{nationality = India}(STUDENT)$$
$$RESULT3 \leftarrow RESULT1 \cap RESULT2$$
$$RESULT \leftarrow \Pi_{name,sid}(RESULT3)$$

Note that :- The union & intersection operations both satisfy the commutative and associative properties.

(i) Commutative property :-

- $R \cup S = S \cup R$
- $R \cap S = S \cap R$

(ii) Associative property :-

- $R \cup (S \cup T) = (R \cup S) \cup T$
- $R \cap (S \cap T) = (R \cap S) \cap T$

3. MINUS / DIFFERENCE operation :-

It is a binary operation that is used to get the tuples from relations R and S such that they belong to R but not S, denoted by R-S.
The MINUS operation can be performed between two UNION-compatible relations.

Example: $\sigma_{gender = female}(STUDENT) - \sigma_{age > 30}(STUDENT)$

The above query gives the resultant relation containing the records of those students who are female of age less than 30.

example: EMPLOYEE $-$ $\sigma_{post = officer}$ (EMPLOYEE)

This query gives the resultant relation containing the set of employees which are not officer.

The MINUS operation is not commutative
ie. $A - B \neq B - A$
but it is associative.
ie. $A - (B - C) = (A - B) - C$

4. CROSS-PRODUCT Operation :-

The CROSS-PRODUCT operation is also known as CARTESIAN PRODUCT or CROSS JOIN. It is denoted by "X" operator. Unlike UNION, INTERSECTION and MINUS, this operation can be performed between two relations which are not union compatible.

The resultant relation obtained by the cross product of two sets/relations A and B contains the tuples produced by combining each tuple of A with every tuples of B. If the degree of relations A & B be n and m respectively, then the degree of A x B is n+m. If n_A and n_B be the no. of tuples in A and B, the relation A x B has $n_A \times n_B$ tuples.

Example: Consider the EMPLOYEE & PROJECT relations

EMPLOYEE

en	name	post	salary
1	Ram	A	50,000
2	Sita	B	40,000
3	Gita	C	30,000

PROJECT

PN	title	Budget
1	X	100000
2	Y	200000
3	Z	300000

EMPLOYEE X PROJECT

en	name	post	salary	PN	title	Budget
1	Ram	A	50000	1	X	100000
2	Sita	B	40000	2	X	100000
3	gita	C	30000	3	X	100000
1	Ram	A	50000	1	Y	100000
2	Sita	B	40000	2	Y	200000
3	Gita	C	30000	3	Y	200000
1	Ram	A	50000	3	Z	300000
2	Sita	B	40000	3	Z	300000
3	Gita	C	30000	3	Z	300000

Binary Relational Operations: JOIN & DIVISION:-

Concept & Example of JOIN Operation and its Variations

The JOIN operation is used to combine the related tuples from two relations into a single longer tuple. It joins two relations and forms a new relation that includes all the attributes of the joining relations in order. The JOIN operation is represented by the operator ⋈. We can join two relations only if they have some common attribute. This common attribute that is used to join the tables is called joining attribute.

Example: Consider the PRICE & QUANTITY relations

PRICE

product	price
pen	20
copy	100
bag	2200
pant	2000

QUANTITY

product	quantity
pen	2
copy	6
Helmet	1
mobile	1

The operation PRICE ⋈ QUANTITY results the following relation;

product	price	quantity
pen	20	2
copy	100	6

Those tuples from two relations will be combined for which the value of joining attribute is common. The duplicate tuples are removed from the result.

Note that the JOIN operation is equivalent to the cartesian product followed by SELECT operation. for example, the operation PRICE ⋈ QUANTITY is equivalent to $\sigma_{PRICE.product = QUANTITY.product}$ (PRICE × QUANTITY).

Variations of JOIN operation :-

a. Natural Join :-

The natural join operation is denoted by the ⋈ operator. It forms the cartesian product between the argument relations and selects those tuples for which the value of common attribute is equal. The duplicate tuples are removed. The joining attribute must have the same name & domain. The joining attribute appears only once in the resultant relation. It is also denoted by * operator.

Example: EMPLOYEE

eid	name	depart
1	Ram	Account
2	Shyam	Sales
3	Sita	HR
4	Gita	Production
5	Hari	Sales
6	Rohan	HR

MANAGER :

depart	manager_eid
Account	20
Sales	5
HR	3
Production	7

The result of natural join between EMPLOYEE and MANAGER is as given ⇒ (EMPLOYEE ⋈ MANAGER)

eid	name	depart	manager_eid
1	Ram	Account	20
2	Shyam	Sales	5
3	Sita	HR	3
4	Gita	Prod.	7
5	Hari	Sales	5
6	Rohan	HR	3

Note: Tuples with NULL value in the joining attribute are also eliminated.

b. Theta JOIN:-

It is the JOIN operation with some condition. The condition includes the comparision operation such as $=, \neq, <, >, <=, >=$ & Γ. The resultant relation includes the tuples from the cartesian product that satisfy the JOIN condition. It is denoted/expressed as $R \bowtie_\theta S$, where θ is some condition involving comparision operation.

Example: Consider the two relations: XIAOMI and HUAWEI as follows:

XIAOMI

X-model	X-price
X5	17000
Redmi	25000
A6 Note	28000
Y3	42000

HUAWEI

H-model	H-price
Nova	18000
P20	21000
Y9	16000
Y9 Prime	30000

The theta join XIAOMI $\bowtie_{\text{X-price} < \text{H-price}}$ HUAWEI gives the following resultant relation.

X-model	X-price	H-model	H-price
X5	17000	Nova	18000
X5	17000	P20	21000
X5	17000	Y9 Prime	30000
Redmi	25000	Y9 prime	30000
A6 Note	28000	Y9 Prime	30000

The special type of theta join that includes equility operator in the condition is called as equijoin.

Join-selectivity:-

If no combination of the tuples satisfy the join condition, the result of a JOIN is an empty relation with zero tuple. If the relation R has n_R tuples and the relation S has n_S tuples, the join operation R ⋈<condition> S will have tuples between zero and $n_R \times n_S$. The expected size of the join result divided by the maximum size $n_R \times n_S$ is called as join-selectivity. A join operation with join selectivity equal to 1 is equivalent to CARTESIAN PRODUCT also called as CROSS JOIN.

Inner Join vs Outer Join:-

The resultant relation obtained from the inner join includes the tuples that are formed by combining the tuples from argument relations. THETA JOIN, EQUI JOIN, NATURAL JOIN are examples of inner join.

An outer join operation results the relation that includes the partial tuples which are not formed by the proper combination of tuples from the argument relations. There are three types of outer joins.

1. LEFT OUTER JOIN
2. RIGHT OUTER JOIN
3. FULL OUTER JOIN

1. LEFT OUTER JOIN :-

It is a type of JOIN operation in which the resultant relation includes all the tuples from the first (or left) relation. Each tuple from the left relation is combined with matching tuple from the right ~~fu~~relation. If the match is not found in the right relation, the corresponding attributes are padded with NULL values. It is denoted by the left outer join operator ⟕.

Example:

EMPLOYEE

eid	name	depart
1	Ram	A
2	Shyam	B
3	Sita	C
4	Gita	A
5	Rita	D

MANAGER

depart	m-eid
A	11
B	22
X	33

The query EMPLOYEE ⟕ MANAGER results the relation as shown below.

eid	name	depart	meid
1	Ram	A	11
2	Shyam	B	22
3	Sita	C	NULL
4	Gita	A	11
5	Rita	D	NULL

2. RIGHT OUTER JOIN :-

It is a type of JOIN operation in which the resultant relation includes all the tuples from the second (or right) relation. Each tuple from the right relation is combined with matching tuple from the left relation. If the match is not found in the left relation, the corresponding attributes are padded with NULL values. It is denoted by the right outer join operator ⟖.

Example: If we perform right outer join operation between EMPLOYEE and MANAGER relations (see earlier example), we get the result as shown below.

EMPLOYEE ⟖ MANAGER ⇒

eid	name	depart	m-eid
1	Ram	A	11
2	Shyam	B	22
NULL	NULL	X	33
4	Gita	A	11

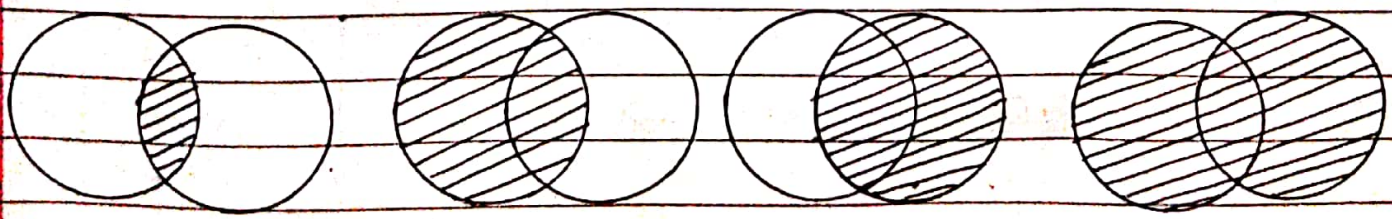
3. FULL OUTER JOIN

It is a type of JOIN operation in which the resultant relation includes all the tuples from left relation and right relation. The tuples are combined if the value of joining attribute matches. If the match is not found, the corresponding attribute values are padded with NULL values. It is denoted by the full outer join operator ⟗. It is the union of left and right outer join.

Example: the operation EMPLOYEE ⟗ MANAGER gives the following result.

eid	name	depart	m-eid
1	Ram	A	11
2	Shyam	B	22
3	Sita	C	NULL
4	Gita	A	11
5	Rita	D	NULL
NULL	NULL	X	33

The inner & outer joins can be expressed in ven-diagram as given below:



INNER JOIN LEFT JOIN RIGHT JOIN FULL JOIN

Concept and Example of Division Operation

Division operator (÷) is used for specific type of query that occurs in database application. It is a derived operation that involves the fundamental operations such as cartesian product, set difference and projection. Basically, division operation is a special type of partitioning.

The division operation R ÷ S can be applied only if:

(i) Attributes of S is proper subset of attributes of R.

(ii) The resultant relation includes the attributes that are in R but not in S.

(iii) The resultant relation includes the tuples from R which are associated to every tuple in S.

Example: Consider the relations EMPLOYEE & PROJECT.

EMPLOYEE

eid	name	project_title
1	Ram	P1
2	Sita	P2
3	Hari	P1
4	Rita	P2
1	Ram	P2
2	sita	P1

PROJECT

project_title
P1
P2

Now,

EMPLOYEE ÷ PROJECT ⇒

eid	name
1	Ram
2	Sita

If we want to retrieve the eid of all those employees who are involved in all the projects, we can write the query as follows;

$$\Pi_{eid}\ (\text{EMPLOYEE} \div \text{PROJECT})$$

example-2: Consider the relations ISSUE & BOOK.

ISSUE

stud_id	book-id	title
S1	b1	C
S2	b2	C++
S3	b3	Java
S4	b2	C++
S5	b3	Java
S2	b1	C
S4	b1	C
S1	b2	C++
S2	b3	Java
S1	b3	Java

BOOK

book-id	title
b1	C
b2	C++
b3	Java

Note to remember

Division operation can also be expressed as the sequence of Π, X & − operations as:

$$T_1 \leftarrow \Pi_X (R)$$
$$T_2 \leftarrow \Pi_X (T_1 \times S - R)$$
$$T \leftarrow T_1 - T_2$$

Now, ISSUE ÷ BOOK ⇨

Stud-id
S_1
S_2

Includes stud-id of a those students who have issued all the books.

Note that, we use division operation when we have to determine the tuples from first relation that are associated with all the tuples of second relation.

Additional Operations on Relational Algebra :-

1. Generalized Projection :-

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

$$\Pi_{F_1, F_2, \ldots, F_n}(R)$$

Where $F_1, F_2, \ldots, F_n$ are the functions of attributes in relation R and may involve arithmetic operations and constants. This operation is helpful when we need to project the computed values of attributes.

Example: Consider the relation EMPLOYEE as

EMPLOYEE (eid, salary, deduction, years_service).

We may design query to display a report including net_salary & tax as:

net_salary = salary - deduction

tax = salary * 0.25

Now, we may use generalized projection as follows:

$$REPORT \leftarrow \rho_{(eid, net\_salary, tax)}(\Pi_{eid, salary - deduction, salary * 0.25.}(EMPLOYEE))$$

2. Aggregate functions :-

The aggregate functions such as SUM, AVERAGE, COUNT, MAX, MIN etc which are applied on some collection of values, are expressed in relational algebra using aggregate function operation. It is denoted by symbol/operator F. This operation involves grouping the tuples according to the value of some attribute and applying some aggregate function ~~etc~~ to each group independently. We use the following expression for the aggregate function operation in relational algebra.

<grouping_attributes> F <function list> (R)

where <grouping attributes> is a list of the ~~rela~~ attributes in R whose values are used to group tuples. function list is a list of (<function> <attribute>) pair. where <function> is one of the aggregate function - such as SUM, AVERAGE, MAX etc and <attribute> is the attribute of R on which aggregate function is to be applied.

The resultant ~~ope~~ relation includes the grouping attributes and those attributes on which the aggregate functions are applied. The resultant relation includes single tuple for each group.

Example: Consider a schema EMPLOYEE as
EMPLOYEE (eid, name, gender, post, salary).

We can use aggregate operation to obtain the average salary of male & female employee as:

$_{gender}\mathcal{F}_{Average(salary)}$ (EMPLOYEE).

Resultant relation is of the following form:

gender	Avg-salary
male	value
female	value

Sometimes we may exclude <grouping attribute>.
Eg: to get the maximum salary of all the employees from above EMPLOYEE relation, we can write query as:

$\mathcal{F}_{MAX(salary)}$ (EMPLOYEE)

3. Outer union:-

The outer union operation is used to take the union of tuples from two relations that have some common attributes but are not union compatible. It is similar to FULL-OUTER-JOIN.

The resultant relation includes the attributes from both the relations but the common attributes are expressed only once.

Tuples from two relations are combined if the value of common attributes are same, otherwise padded with NULL values.

Example:

NEP_MARKS

sid	Nepali
1	50
2	60
3	70
4	80

SCI_ENG_MARKS

sid	English	Science
3	90	80
4	80	70
5	85	75
6	95	68

The resultant relation obtained by outer union operation betn NEP_MARKS & SCI-ENG_MARKS is as follows:

Sid	NEP	ENG	SCI
1	50	NULL	NULL
2	60	NULL	NULL
3	70	90	80
4	80	80	70
5	NULL	85	75
6	95	68	NULL

~~Tuple~~ Relational calculus

The relational algebra we discussed so far help to design the procedural queries; where we express explicitely in our query expression about what to retrieve and in which order. The relational calculus on the other hand allows to design the queries that are non-procedural but declarative. In relational calculus, we can write the queries using declarative expressions to express the retrieval request. In such query expressions, there is no description of how or in what order to evaluate a query. A relational calculus query expression specifies what to retrieve rather than how to retrieve.

The relational algebra is procedural approach to query design, where we must write a sequence of operations to specify a retrieval request in a particular order. If we change the order of expressions in relational algebra queries, it also changes the query execution strategy and also possibly the result. On the other hand a relational calculus query can be expressed in various order without altering the query evaluation strategy.

The expresive power of relational calculus is equivalent to that of relational algebra. That means we any query in relational algebra can also be designed using relational calculus.

Any query language L is said to be relationally complete if any query in relational calculus can be expressed in L.

The relational calculus is important for two reasons.
(i) It has a firm basis in mathematical logic.
(ii) The standard SQL for RDBMS has its basic foundation in the tuple relational calculus.

The relational calculus exists in two forms. Known as tuple relational calculus and domain relational calculus.

Tuple Relational Calculus

The tuple relational calculus is based on defining the tuple variable and the conditions under which the tuple is selected. A simple tuple relational algebra query is

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

where, t is the tuple variable and COND(t) is the condition under which a tuple is selected. The result of the above query is the set of all tuples that satisfy the COND(t).

eg: Consider a relation EMPLOYEE (eid, name, post, sal). The following query results the set of ~~tuples~~ name of the employee having salary more than 50,000.

$\{t.name \mid EMPLOYEE(t) \text{ AND } t.salary > 50,000\}$

Where
t.name ⇒ selected attribute
EMPLOYEE(t) ⇒ Range of the tuple variable t.
t.salary > 50000 ⇒ selection condition.

We can generalize the syntax/form of query as

$\{t_1.A_j, t_2.A_K, \ldots, t_n.A_m \mid COND(t_1, t_2, \ldots, t_{n+m})\}$

Where t_i represent tuples
A_j represents attributes
COND represent condition/formula to retrieve tuples.

Generally we have the following conditions

i) $R(t_i)$ ⇒ tuple t_i is bound to relation R.
ii) $t_i.A$ Op $t_j.B$ ⇒ comparision between the attribute of tuples t_i & t_j.
iii) $t_i.A$ Op C ⇒ comparing the attribute A of tuple t_i with constant value C.

Here Op ⇒ any comparision operation ($=, \neq, <, \leq, >, \geq$ etc)

Note that, one or more atomic condition can be connected by logical operators AND, OR, NOT.

Each atomic condition is a Boolean expression & the logical combination of two or more atomic condition is also a Boolean expression.

A tuple relational calculus query may contain two more operators known as quantifiers. The two quantifiers are

1. Existential quantifier ($\exists$) and
2. Universal quantifier ($\forall$).

Note that, the expression $\exists t(F)$ is true if there exists some tuple t that satisfies the Boolean formula/expression F. Similarly the expression $\forall t(F)$ is true if the Boolean expression F is true for all the tuples t.

Q. ~~Consider~~ Obtain the name & address of all employees who work for research depart.

EMPLOYEE (eid, name, address, contact, depart)

→ $\{t.name, t.address \mid EMPLOYEE(t) \text{ AND } t.depart = \text{"research"}\}$

Q. Obtain name and address of employees working for research depart.

EMPLOYEE (eid, name, add, contact, depart_id)
DEPART (depart_id, depart_name)

→ $\{t_1.name, t_1.add \mid EMPLOYEE(t_1) \text{ AND } (\exists t_2)(DEPART(t_2) \text{ AND } t_1.depart\_id = t_2.depart\_id \text{ AND } t_2.depart\_name = \text{"research"}\}$

Q. Obtain the name of employees and the name of their department using tuple relational calculus query.

EMPLOYEE (eid, name, post, salary, depart_no)
DEPART (depart_no, depart_name)

→ $\{t_1.name, t_2.depart\_name \mid EMPLOYEE(t_1)$ AND $DEPART(t_2)$ AND $t_1.depart\_no = t_2.depart\_no\}$

~~Note: Domain Relational Calculus~~

The tuple variables expressed to the left of bar (|) are free tuple variables & those expressed to the right of bar symbol are called bound tuple variables

Q. Obtain the name of those employees who have no dependent.

EMPLOYEE (eid, ename, post, salary)
DEPENDENT (did, dname, age, gender, eid)

→ $\{t_1.ename \mid EMPLOYEE(t_1)$ AND $(\forall t_2)(DEPENDENT(t_2)$ AND $t_2.eid \neq t_1.eid\}$

Domain Relational Calculus :-

The domain relational calculus is based on defining the domain variable and the condition under which domain value is selected. A domain variable ranges over the values of a particular domain. If the resultant relation of a query gives n-degree relation, we need n different domain variables in the query. A domain relational calculus query is of the following form.

$$\{x_1, x_2, \cdots, x_n \mid COND(x_1, x_2, \cdots, x_n, x_{n+1} \cdots)\}$$

Where $x_1, x_2, \cdots$ are the domain variables that range over domains (attributes) and COND is a condition or formula under which domain values are selected. A formula may contain the logical combination of Boolean expressions. We use the lowercase letters a, b, c, d, ... for domain variables.

The popular SQL is based on tuple relational calculus and QBE is based on domain relational calculus.

Q. Obtain name, post and salary of the employees having salary more than 50,000.

EMPLOYEE (eid, name, post, salary, gender, address)

→ $\{b, c, d \mid (\exists a)(\exists e)(\exists f)(EMPLOYEE(abcdef) \text{ AND } d > 50000)\}$

OR

$\{b, c, d \mid EMPLOYEE(abcdef) \text{ AND } d > 500000\}$

Q. Retrieve the name and ~~address~~ post of all employee who work for research depart.

EMPLOYEE (eid, name, post, salary, depart_no)
DEPART (depart_no, depart-name)

→ $\{b, c \mid (\exists a)(\exists d)(\exists e)(\exists x)(\exists y)(EMPLOYEE(abcde) \text{ AND } DEPART(xy) \text{ AND } e = x \text{ AND } y = \text{"Research"})\}$

Q. Assume a database COMPANY as follows:

EMP (eid, name)
COMPANY (cname, address)
WORKS (eid, cname)
SUPERVISE (sup-eid, emp-eid)

a. Find the name of all supervisors that work for companies having address pokhara.
b. Find the name of all companies having more than 4 supervisors.
c. Find the name of supervisor having largest no. of employee.

→ (a) $R_1 \leftarrow \Pi_{eid}(\sigma_{address = \text{"pokhara"}}(COMPANY \bowtie WORKS))$

$R_2 \leftarrow R_1 \bowtie_{R_1.eid = SUPERVISE.sup\text{-}eid} SUPERVISE$

$R \leftarrow \Pi_{name}(R_1 \bowtie EMP)$

(b) $R_1 \leftarrow \Pi_{cname, sup\text{-}eid}(WORKS \bowtie_{WORKS.eid = SUPERVISE.sup\text{-}eid} SUPERVISE)$

$R_2 \leftarrow cname\ F_{COUNT(sup\text{-}eid)\ AS\ sup\_count}(R_1)$

$R \leftarrow \Pi_{cname}(\sigma_{sup\_count > 4}(R_2))$

(c) $R_1 \leftarrow sup\text{-}eid\ F_{COUNT(emp\text{-}eid)\ AS\ emp\_no}(SUPERVISE)$

$R_2 \leftarrow \Pi_{sup\text{-}eid}(F_{MAX(emp\_no)}(R_1))$

$R \leftarrow \Pi_{name}(EMP \bowtie_{EMP.eid = R_2.sup\text{-}eid} R_2)$