# RELATIONAL CALCULUS

⇒ Relational Calculus is a non-procedural query language.

⇒ It uses mathematical predicate calculus (or first-order logic) instead of algebra.

⇒ The relational calculus tells what to do but never explains how to do.

## Types of Quantifiers used :-

i) Universal Quantifiers –

The universal quantifier denoted by ∀ is read as for all which means that in a given set of tuples exactly all tuples satisfy a given condition.

ii) Existential Quantifiers –

The existential quantifier denoted by ∃ is read as there exists which means that in a given set of tuples there is atleast one occurence

## FREE VARIABLE AND BOUND VARIABLE :-

⇒ A tuple variable is said to be free variable unless it is quantified by a ∃ or ∀.

⇒ A tuple variable is said to be bound variable if it is quantified by ∀ or ∃ (quantifiers).

## Types of Relational Calculus :-

i) Tuple Relational Calculus.

ii) Domain Relational Calculus

### i) Tuple Relational Calculus :-

⇒ It is specified to select the tuples in a relation. In TRC, (tuple relational calculus), filtering variable uses the tuple of a relation.

⇒ The result of the relation can have one or more tuples.

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

(t) → resulting tuple.

P(t) → condition.

⇒ It is a set of all tuples t such that, predicate P is true for t where t is the resulting tuple, P(t) denotes predicate or condition use to fetch the tuple.

### Predicate Calculus Formula :-

i) Set of attribute and constraint.

ii) Set of comparision operator.

iii) Set of connectivity. ($\wedge, \vee, \neg$)

iv) Implication ($x \Rightarrow y$)

v) Quantifiers ($\forall, \exists$)

For Example :-

{ T.name | Author (T) AND T.article = 'database' }

Author (T)

| T.id | T.Name | T.article |
|---|---|---|
| 1 | Ram | database |
| 2 | Shyam | Network |
| 3 | Hari | database |
| 4 | Geeta | Network |

⇒ TRC can be quantified. In TRC, we can use Existential Quantifiers and Universal Quantifiers (∀).

{ R | ∃T ∈ Authors (T.article = 'database' AND R.name = T.name) }

Q Find the loan no., branch name and amount for all loans of over 1200.

{ Branch_name | Loan (T) } { Branch. { t | t ∈ loan ∧ t[amount] > 1200 } }

{ R | ∀ T ∈ Loan (

⇒ { Branch_name | Loan (T), Amount | Loan (T)

⇒ { Branch_name

⇒ { t | t ∈ loan AND t[amount] > 1200 } .

Q Find the loan no for each loan of amount > 1200.

{ R | ∃ T ∈ Loan ( ( (amount) > 1200 and . R.Loan_no = T.Loan_no ) ;

$\Rightarrow \{t \mid \exists s \in loan\ (t[loan\ number] = s[loan\ number] \wedge s[amount] > 1200)\}$

| T.id | T.Name | T... |
|---|---|---|
| 1 | Ram | data |
| 3 | Hari | data |

Q Find the name of all customer having loan at Gurupur branch.

$\Rightarrow \{R \mid \exists t \in borrower\ (R[customer\_name] = T.customer\_name) \text{ AND } \exists t \in loan\ (R[branch\_name] = \text{"Gurupur"}) \text{ AND } R[loan\_no] = s[loan\_no])\}$

$\Rightarrow \{t \mid \exists s \in borrower\ (t[customer\_name] = s[customer\_name] \wedge \exists u \in loan\ (u[branch\_name] = \text{"Gurupur"} \wedge u[loan\_number] = s[loan\_number]))\}$

Q Find the names of all customers having a loan, an account, or both at the bank.

$\{t \mid \exists s \in borrower\ (t[customer\_name] = s[customer\_name] \vee \exists u \in depositor\ (t[customer\_name] = u[customer\_name])\}$

Q. Find the names of all customers who have a loan and an account at the bank.

$\{t \mid \exists s \in borrower\ (t[customer\_name] = s[customer\_name] \wedge \exists u \in depositor\ (t[customer\_name] = u[customer\_name])\}$

## Domain Relational Calculus :-

⇒ The second form of relation is known as Domain relational calculus

⇒ In domain relational calculus, filtering variable uses the domain of attributes.

⇒ Domain relational calculus uses the same operators as tuple calculus.

⇒ It uses logical connectives $\wedge$ (and), $\vee$ (or) and $\neg$ (not). It uses Existential ($\exists$) and Universal ($\forall$) Quantifiers to bind the variable.

Notation -

⇒ $\{ < a_1, a_2, \ldots, a_n > | P(a1, a2, \ldots, an) \}$ where a1, a2 represents domain variables.

⇒ P represents a predicate similar the predicate calculus.

⇒ $< a1, a2, \ldots an > \in r$, where r is the relation on n attributes and a1, a2, .... an are domain variable or domain constants.

Q Find Loan_no, Branch_name and Amount for loans over 1200.

⇒ $\{ < l, b, a > | < l, b, a > \in loan \wedge a > 1200 \}$

Q Find the Loan_no of each loan of an amount greater than 1200.

⇒ $\{ < l > | \exists b, a \, (l, b, a) \in loan \wedge a > 1200 \}$

Q Find the name of all customers who have a loan of over 1200.

⇒ ~~$< x > | \exists s, c ( x, s, c )$~~

⇒ $< c\_name > | \exists (l, b) \in borrow \wedge < l > | \exists (l, b, a) \in loan \wedge$ ~~$a > 1200$~~

$\Rightarrow \{\langle c \rangle \mid \exists l, b, a (\langle C, l \rangle \in borrow \wedge \langle l, b, a \rangle \in loan \wedge a > 1200\}$

S1

| Sid | name | age |
|---|---|---|
| 22 | x | 45 |
| 31 | y | 55 |
| 58 | z | 35 |

R1

| Sid | bid | date |
|---|---|---|
| 22 | 101 | today |
| 58 | 103 | tomw |

| S1.sid | name | age | R1.sid | bid | dat |
|---|---|---|---|---|---|

$\Theta =$ $S1 \bowtie_{S1.sid < R1.sid} R1$

Equi = $S1 \bowtie_{S1.sid = R1.sid} R1$

Inner Join :—

An inner join include only those tuples that satisfy the matching criteria, while rest of the tuples are excluded.

Θ / Conditional Join :—

A join operation with such general condition is called Θ join / Inner join : $\{\langle , \rangle, \leq, \geq, =, \neq\}$

Equi join :—

When the join condition involves with equality comparision then it's called equi join.

$Dept \bowtie_{mgrcode = Ecode} Emp$

Emp (Ecode, name, address, Salary)

Dept (Deptno, Dname, summary, Ecode)

We need to find all the managers of dept having salary > 3000.

$Dept \bowtie_{Emp.Salary > 3000} Emp$

# JOINTS :-

⇒ It is a binary operation which takes two relation as an input

⇒ The join operation will check the key attribute is present in both the relation or not. If present then it will retrieve the required information from both the tables by comparing the value of the common key attribute.

⇒ It is denoted by ⋈

⇒ A join operation combine related tuples from different relation if and only if an even join condition is satisfie

⇒ Types of join :-

i) Natural Join
ii) Outer Join → Left outer Join
→ Right outer Join
→ Full outer Join
iii) Equi Join.

## i) Natural Join :-

⇒ A natural join is the set of tuples of all combination in R and S that are equal on there common attribute name.

⇒ It is denoted by ⋈.

⇒ It natural join operation first perform cartesian, then it forms a selection forcing equality on those attribute that appear in both relation and finally removes duplicate attribute

Q Select Ename from employee natural join department where employee Empno = department Empno.

Dept ⋈ Emp

→ Employee –

| Empno | Ename | Address |
|---|---|---|
| 1 | A | Delhi |
| 2 | B | Mumbai |
| 3 | C | Mumbai |
| 4 | D | Delhi |

→ Department –

| Deptno | Dname | Empno |
|---|---|---|
| $D_1$ | HR | 1 |
| $D_2$ | It | 2 |
| $D_3$ | Marketing | 4 |

| Empno | Ename | Address | Deptno | Dname | Empno |
|---|---|---|---|---|---|
| 1 | A | Delhi | $D_1$ | HR | 1 |
| 1 | A | Delhi | $D_2$ | It | 2 |
| 1 | A | Delhi | $D_3$ | Marketing | 4 |
| 2 | B | Mumbai | $D_1$ | HR | 1 |
| 2 | B | Mumbai | $D_2$ | It | 2 |
| 2 | B | Mumbai | $D_3$ | Marketing | 4 |
| 3 | C | Mumbai | $D_1$ | HR | 1 |
| 3 | C | Mumbai | $D_2$ | It | 2 |
| 3 | C | Mumbai | $D_3$ | Marketing | 4 |
| 4 | D | Delhi | $D_1$ | HR | 1 |
| 4 | D | Delhi | $D_2$ | It | 2 |
| 4 | D | Delhi | $D_3$ | Marketing | 4 |

⇒ $E \bowtie D$

ii) Outer Join :-

⇒ The outer join operation is an extension of the join operation. It returns all the records from both the tables that satisfy the join condition.

⇒ In other words, these join will not return only the matching records but also return the all unmatch row from one or both tables.

⇒ Types : a) Left Outer Join
b) Right Outer Join
c) Full Outer Join.

a) Left Outer Join -

⇒ The left outer join retrieves all the records from the left table and matching row from right table.

⇒ It will return null when no matching record is find in right side table.

⇒ Employee :-

| Ename | Street | City |
|---|---|---|
| Ram | CivilLine | Mumbai |
| Shyam | Parklane | Kolkata |
| Ravi | M.G.Street | Delhi |
| Hari | Nehru Place | Hyderabad |

⇒ Workers :-

| Ename | Branch | Salary |
|---|---|---|
| Ram | Infosys | 10000 |
| Shyam | Wipro | 20000 |
| Robert | HCL | 30000 |
| Hari | TCS | 40000 |

Eno

| Ename | Street | City | EName | Branch | Salary |
|---|---|---|---|---|---|
| Ram | CivilLine | Mumbai | Ram | Infosys | 10000 |
| Ram | CivilLine | Mumbai | Shyam | Wipro | 20000 |
| Ram | CivilLine | Mumbai | Kuber | IHCL | 30000 |
| Ram | CivilLine | Mumbai | Hari | TCS | 40000 |
| Shyam | Parklane | Kolkata | Ram | Infosys | 10000 |
| Shyam | Parklane | Kolkata | Shyam | Wipro | 20000 |
| Shyam | Parklane | Kolkata | Kuber | HCL | 30000 |
| Shyam | Parklane | Kolkata | Hari | TCS | 40000 |
| Ravi | M.G. Street | Delhi | Ram | Infosys | 10000 |
| Ravi | M.G. Street | Delhi | Shyam | Wipro | 20000 |
| Ravi | M.G. Street | Delhi | Kuber | HCL | 30000 |
| Ravi | M.G. Street | Delhi | Hari | TCS | 40000 |
| Hari | Nehru place | Hyderabad | Ram | Infosys | 10000 |
| Hari | Nehru place | Hyderabad | Shyam | Wipro | 20000 |
| Hari | Nehru place | Hyderabad | Kuber | HCL | 30000 |
| Hari | Nehru place | Hyderabad | Hari | TCS | 40000 |

⇒ EMP :-

| EName | Street | City | Branch | Salary |
|---|---|---|---|---|
| Ram | CivilLine | Mumbai | Infosys | 10000 |
| Shyam | Parklame | Kolkata | Wipro | 20000 |
| Ravi | M.G. Street | Delhi | NULL | NULL |
| Hari | Nehru Place | Hyderabad | TCS | 40000 |

b) Right Outer Join -

⇒ The right out retrieve all records from right hand table and matched row from right hand table.

⇒ It will return null when no matching record is found in the left hand table.

⇒ E ⟖ W :-

| Ename | Street | City | Branch | Salary |
|---|---|---|---|---|
| Ram | CivilLine | Mumbai | Infosys | 10000 |
| Shyam | ParkLane | Kolkata | Wipro | 20000 |
| Kuber | NULL | NULL | HCL | 30000 |
| Hari | Nehruplace | Hyderabad | TCS | 40000 |

c) Full Outer Join -

⇒ The full outer join returns a result that includes all rows from both tables

⇒ The column's of right hand table return NULL when no matching record are found in the left hand table and If no matching records are found in right hand table then left hand table returns NULL.

⇒ E ⟗ W

| Ename | Street | City | Branch | Salary |
|---|---|---|---|---|
| Ram | CivilLine | Mumbai | Infosys | 10000 |
| Shyam | ParkLane | Kolkata | Wipro | 20000 |
| Kuber | NULL | NULL | HCL | 30000 |
| Hari | Nehruplace | Hyderabad | TCS | 40000 |
| Ravi | M.G.Street | Delhi | NULL | NULL |

## iii) Equi Join :-

= It is also known as Inner Join. It is the most common join. It is based on matching data as per equality condition.

= The equi join uses comparision operator (=) equally.

Employee

| Eno | EmpName | Permanent address |
|---|---|---|
| 1. | A | Delhi |
| 2. | B | Mumbai |
| 3. | C | Mumbai |
| 4. | D | Delhi |

Dept

| DNo | Work Location | ENo |
|---|---|---|
| $D_1$ | Delhi | 1 |
| $D_2$ | Pune | 2 |
| $D_3$ | Patna | 4 |

Q Find the name of the employee who work in a department having permanent address same as their work location.

⇒ EmpName – A

| Eno | EmpName | Permanent Address | DNo | Work Location | ENo |
|---|---|---|---|---|---|
| 1 | A | Delhi | $D_1$ | Delhi | 1 |
| 1. | A | Delhi | $D_2$ | Pune | 2 |
| 1 | A | Delhi | $D_3$ | Patna | 4 |
| 2 | B | Delhi Mumbai | $D_1$ | Delhi | 1 |
| 2 | B | Delhi Mumbai | $D_2$ | Pune | 2 |
| 2 | B | Delhi Mumbai | $D_3$ | Patna | 4 |
| 3 | C | Mumbai | $D_1$ | Delhi | 1 |
| 3 | C | Mumbai | $D_2$ | Pune | 2 |
| 3 | C | Mumbai | $D_3$ | Patna | 4 |
| 4 | D | Delhi | $D_1$ | Delhi | 1 |
| 4 | D | Delhi | $D_2$ | Pune | 2 |
| 4 | D | Delhi | $D_3$ | Patna | 4 |