RELATIONAL CALCULUS

Relational Calculus is a formal Query language for the relational model. In Relational calculus, we write one declarative expression to specify a retrieval request, and hence there is no description of how to evaluate a query. A calculus expression specifies what is to be retrieved rather than how to retrieve it. Therefore R.C. is considered to be a non proceedural language. This differs from relational algebra where we must write a sequence of operations to specify a retrieval request; hence, it can be considered as a proceedural way of stating a query. RC has same expressive power as R.Algebra.

TUPLE VARIABLES & 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:-

$\{t \mid cond(t)\}$, where t is tuple variable and cond(t) is a condition expression involving t. The result of such a query is the set of all tuples t that satisfy cond(t).

for eg. to find all employees whose salary is above 50000 we write: $\{t \mid EMPLOYEE(t) \wedge t.Salary > 50000\}$

To retrieve some attributes like fname, lname we write.

$$\{t.Fname, t.Lname \mid EMPLOYEE(t) \wedge t.salary > 50000\}.$$

Informally, we need to specify the following information in TRC expression:

↳ For Each tuple variable t, the range of relation R of t

↳ A Condition to select particular combinations of tuples

↳ A Set of attributes to be retrieved.

Example: Suppose we have a relation Student (Rollno, Name, Deptno, Sex) and need to findout all male students of deptno 2 with rollno + name

Sol^n $\{t.rollno, t.name \mid Student(t) \wedge t.deptno=2 \wedge t.sex='male'\}$

The Existential and Universal Qualifiers :→

In order to write Query with complex expressions we use 2 symbols called Quantifiers; these are universal quantifier ($\forall$) and the existential quantifier ($\exists$). To expresss a condition we can use these quantifiers with tuple variables as shown below.

Example: Emp(eid, name, Address)
Dependent (did ,name, eid) find out employee names who have no dependents.

Solⁿ: Let e be the tuple variable over Emp.

e.name | Emp(e) $\wedge$ (true for emp having no dependents)

⇓

(false for emp having some dependents)

⇓

$\neg$ (true for emp having some dependents)

⇓

$\neg \exists d\,(\text{deplndent}(d) \wedge d.eid = e.eid)$

$\therefore \{e.name \mid Emp(e) \wedge (\neg \exists d\,(\text{dependent}(d) \wedge deid = e.eid)\}$

In above Query we have 2 tuple variables e, d. As with tuple variable e we have not used any Quatifier it is called free variable and d is called Bound Variable.

Transforming the Universal & Existential Quantifiers

It is posible to transform universal to existential Quantifier and vice versa to get an equivalent expression. One general transformation can be described informally as follows: Transform one type of Quantifier into the other with negation (preceded by NOT); AND or OR replace one another.

$\neg\exists() = \forall\neg()$

$\neg\forall() = \exists\neg()$

1.) $(\forall x)(P(x)) \equiv \neg(\exists x)(\neg P(x))$

Hint: Taking negation 2ice

$\neg\neg(\forall x)(P(x)) \equiv \neg(\exists x)(\neg P(x))$

2.) $(\exists x)(P(x)) \equiv \neg(\forall x)(\neg P(x))$

3.) $(\forall x)(P(x) \wedge Q(x)) \equiv \neg(\exists x)(\neg P(x) \vee \neg Q(x))$ -- De morgans law

4.) $(\forall x)(P(x) \vee Q(x)) \equiv \neg(\exists x)(\neg P(x) \wedge \neg Q(x))$

$\overline{(P \vee Q)} = \bar{P} \wedge \bar{Q}$

$\overline{(P \wedge Q)} = \bar{P} \vee \bar{Q}$

5.) $(\exists x)(P(x) \wedge Q(x)) \equiv \neg(\forall x)(\neg P(x) \vee \neg Q(x))$

6) $(\exists x)(P(x) \vee Q(x)) \equiv \neg(\forall x)(\neg P(x) \wedge \neg Q(x))$

Example: Emp(eid, name, address)

Dependent(did, name, eid)

List names of Emplyoee who do not have dependents

Sol^n: $\{e.name \mid employee(e) \wedge (\neg\exists d\,(Dependent(d) \wedge (e.Eid = d.eid)))\}$

Now let us change $\exists$ to $\forall$

$\{e.ename \mid employee(e) \wedge (\forall d\, \neg(Dependent(d) \wedge (e.eid = d.eid)))\}$ — a

$e.name \mid employee(e) \wedge (\forall d\,(\neg Dependent(d)) \vee \neg(e.eid = d.eid))\}$ — b

Now equation a & b will give same result.

SAFE EXPRESSION: A safe expression in relational calculus is one that is guaranteed to yield a finite number of tuples as its results; otherswise, the expression is called unsafe.

for eg:, theexpression

$$\{t \mid \text{Not}(\text{EMPLOYEE}(t))\} \text{ ----- unsafe}$$

is unsafe because it yields all tuples in the universe that are not employee tuples, which are infinitely numerous. The equivalent safe expression can be written as

$$\{t \mid \text{EMPLOYEE}(t)\} \text{ ----- safe.}$$

Example: depositor (cust-name, acc-no)
borrower (cust-name, loan-no)
loan (loan-no, branch-name, amount)
Customer (cust-name, city, street)
Account (acc-no, branch-name, balance)
Branch (Branch-name, branch-city, assets)

Q1: find the loan details of loan above 1200.

Sol: $\{t \mid \text{Loan}(t) \wedge t.\text{amount} > 1200\}$

Q2: Find names of all Customers who have a loan from branch 'x'.

Sol: We will have to join borrower & loan.

$$\{b.\text{name} \mid \text{borrower}(b) \wedge \exists L\,(\text{loan}(L) \wedge L.\text{loan-no} = b.\text{loan-no} \wedge L.\text{branch-name} = \text{'x'})\}$$

$$\Downarrow$$

$$\{b.\text{name} \mid \exists b \in \text{borrower} \wedge \exists L \in \text{loan} \wedge L.\text{loano} = b.\text{loan-no} \wedge L.\text{branch-name} = \text{'x'}\}$$

Ques: Customer who have an account or loan or both

Sol: Here we need to Join Customer-depositor and Customer-borrower relations.

$\{ t \mid Customer(t) \wedge$ $\underline{\text{if borrower}}$ ---①
or
$\underline{\text{if depositor}}$ ---②

①--- $\exists b\,(borrower(b) \wedge b.cust\_name = t.cust\_name)$

②--- $\exists d\,(depositor(d) \wedge d.cust\_name = t.cust\_name)$

Now Complete Query is

$t \mid Customer(t) \wedge (\exists b\,(borrower(b) \wedge b.cust\_name = t.cust\_name)$
$\forall \exists d\,(depositor(d) \wedge d.custname = t.custname)\}$

DOMAIN RELATIONAL CALCULUS

SQL is based on tuple Relational Calculus Query-By-Example QBE is related to Domain Calculus. DC differs from TRC in the type of variables used in formulas: Rather than having variables range over tuples, the variables range over single values from domain 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 DC is of the form

$\{x_1, x_2, x_3, ----x_n \mid COND(x_1, x_2 \ldots x_n, x_{n+1} \ldots x_{n+m})\}$

where $x_1, x_2 \ldots x_{n+m}$ are domain variables.

Example : EMPLOYEE(FirstName, LastName, Eid, DOB, Add, Sex, Salary, deptno)

DEPARTMENT (deptno, dname, ManagerId)

find the names & address of the employee whose name is ASHOK KUMAR

Solⁿ:- Here we need to take domain Variable for each attribute for eg

firstName → a, LastName → b, Eid → c, Dob → d, Add → e

Sex → f Salary → g deptno → h

Similarly, deptno → x dname → y ManagerId → z

$$\{abe \mid \exists_c \exists_d \exists_f \exists_g \exists_h (\text{EMPLOYEE}(abcdefgh) \wedge (a='\text{ASHOK}' \wedge b='\text{KUMAR}'))\}$$

$$\Downarrow$$

$$\{abe \mid \text{EMPLOYEE}(abcdefgh) \wedge (a='\text{ASHOK}') \wedge (b='\text{KUMAR}')\}$$

$$\Downarrow$$

$$\{abe \mid \text{EMPLOYEE}('\text{ASHOK}', '\text{KUMAR}', c, d, e, f, g, h)\}$$

Example: List the name of Employees who have no dept to manage.

Solⁿ:

$$ab \mid \exists_c (\text{EMPLOYEE}(abcdefgh) \wedge \neg\exists_z (\text{DEPARTMENT}(xyz) \wedge (z=c))\}$$