**Relational Calculus**

Contrary to Relational Algebra which is a procedural query language to fetch data and which also explains how it is done, Relational Calculus in non-procedural query language and has no description about how the query will work or the data will be fetched. It only focusses on what to do, and not on how to do it.

Relational Calculus exists in two forms:

1. Tuple Relational Calculus (TRC)
2. Domain Relational Calculus (DRC)

* **Tuple Relational Calculus (TRC)**

In tuple relational calculus, we work on filtering tuples based on the given condition. The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation. The result of the relation can have one or more tuples.

**Syntax: {T | Condition}** or **{T | P (T)}** or **{T | Condition (T)}**

where,

T is the resulting tuples

P(T) is known as Predicate and these are the condition used to fetch T.

In this form of relational calculus, we define a tuple variable, specify the table(relation) name in which the tuple is to be searched for, along with a condition.

We can also specify column name using a . dot operator, with the tuple variable to only get a certain attribute(column) in result.

A tuple variable is nothing but a name, can be anything, generally we use a single alphabet for this, so let's say T is a tuple variable.

To specify the name of the relation(table) in which we want to look for data, we do the following:

Relation(T), where T is our tuple variable.

**For example,** if our table is Student, we would put it as Student(T)

Then comes the condition part, to specify a condition applicable for a particular attribute(column), we can use the . dot variable with the tuple variable to specify it, like in table Student, if we want to get data for students with age greater than 17, then, we can write it as,

T.age > 17, where T is our tuple variable.

Putting it all together, if we want to use Tuple Relational Calculus to fetch names of students, from table Student, with age greater than 17, then, for T being our tuple variable,

{T.name | Student(T) AND T.age > 17}

P(t) may have various conditions logically combined with OR (∨), AND (∧), NOT (¬).

It also uses quantifiers:

∃ t ∈ r (Q(t)) =” there exists” a tuple in t in relation r such that predicate Q(t) is true.

∀ t ∈ r (Q(t)) = Q(t) is true “for all” tuples in relation r.

**For example:**

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

**OUTPUT:** This query selects the tuples from the AUTHOR relation. It returns a tuple with 'name' from Author who has written an article on 'database'.

TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (∃) and Universal Quantifiers (∀).

**Example:**

Table: Student

**First\_Name Last\_Name Age**

**------------------------------------------**

Ajeet Singh 30

Chaitanya Singh 31

Rajeev Bhatia 27

Carl Pratap 28

Let’s write relational calculus queries.

**Query to display the last name of those students where age is greater than 30**

{ t.Last\_Name | Student(t) AND t.age > 30 }

In the above query you can see two parts separated by | symbol. The second part is where we define the condition and in the first part, we specify the fields which we want to display for the selected tuples.

The result of the above query would be:

**Last\_Name**

**---------------**

Singh

**Query to display all the details of students where Last name is ‘Singh’**

{ t | Student(t) AND t.Last\_Name = 'Singh' }

**Output:**

**First\_Name Last\_Name Age**

**-----------------------------------------**

Ajeet Singh 30

Chaitanya Singh 31

* **Domain Relational Calculus (DRC)**

In domain relational calculus, filtering is done based on the domain of the attributes and not based on the tuple values.

Syntax: **{c1, c2, c3, ..., cn | F (c1, c2, c3, ..., cn)}**

where, c1, c2... etc. represents domain of attributes(columns) and F defines the formula including the condition for fetching the data.

Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives ∧ (and), ∨ (or) and ┓ (not).

It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable.

**For example,**

{< name, age > | ∈ Student ∧ age > 17}

Again, the above query will return the names and ages of the students in the table Student who are older than 17.

**For example:**

{< article, page, subject > | ∈ javatpoint ∧ subject = 'database'}

Output: This query will yield the article, page, and subject from the relation javatpoint, where the subject is a database.

**Example:**

Table: Student

**First\_Name Last\_Name Age**

**-----------------------------------------**

Ajeet Singh 30

Chaitanya Singh 31

Rajeev Bhatia 27

Carl Pratap 28

**Query to find the first name and age of students where student age is greater than 27**

{< First\_Name, Age > | ∈ Student ∧ Age > 27}

**Note:** The symbols used for logical operators are: ∧ for AND, ∨ for OR and ┓ for NOT.

**Output:**

**First\_Name Age**

**-----------------------**

Ajeet 30

Chaitanya 31

Carl 28