Chapter 05: Relational Calculus

1. Introduction :

Relational calculus is a declarative language used to query relational databases. Unlike relational algebra (which specifies how to perform a query), relational calculus focuses on what the query should retrieve without specifying how to achieve it. It is based on formal mathematics and exists in two main forms:

TRC (Tuple Relational Calculus):
Uses tuple variables (complete records). Each variable represents an entire row in a relation.
Example 01: *If the relation* Employees *contains the columns* name *and* salary*, a variable T in TRC would refer to a complete row such as (Alice, 6000).*

DRC (Domain Relational Calculus):
Uses variables for each domain (i.e., each column). Each variable corresponds to an individual value in a column.
Example 02: *If the relation* Employees *contains the columns* name *and* salary*, two variables like X (for* name*) and Y (for* salary*) would be used.*

2. Query Notation:

TRC (Tuple Relational Calculus):
Queries are written in the form:

{T | condition(T)}

Where T is a tuple variable, and condition(T) describes the criteria that the tuple must satisfy.

Example 03: *Retrieve the names of employees with a salary greater than 5000:*

$$\{T.name \mid T \in Employees \land T.salary > 5000\}$$

DRC (Domain Relational Calculus):
Queries are written in the form:

{X, Y, ... | condition(X, Y, ...)}

Where X, Y, ... are variables for the domains, and condition(X, Y, ...) describes the criteria for these values.

Example 04: *Retrieve the names of employees with a salary greater than 5000:*

{Name | ∃Salary (Employees(Name, Salary) ∧ Salary > 5000)}

3. Difference beteween TRC and DRC

In summary:

Characteristic	***TRC (Tuple Relational Calculus)***	***DRC (Domain Relational Calculus)***
Type of variable	*Tuples*	*Individual columns*
Readability	*More intuitive*	*Can be complex*
Focus	*Complete rows*	*Specific values*
Usage	*Tuple-oriented queries*	*Value-oriented queries*

Both approaches are equivalent in expressive power, but the choice often depends on the designer's preference or the specific situation.

In this course, we see only the TRC.

4. TRC (Tuple Relational Calculus)

4.1 Union:

*In **Tuple Relational Calculus (TRC)**, the union of two relations can be expressed by combining the conditions of the two relations using the logical operator **OR (∨)**. The syntax for the union is therefore based on defining tuples that belong to either one of the relations.*

General Syntax for Union in TRC:

{T | T ∈ R1 ∨ T ∈ R2}

- ***R1*** *and* ***R2*** *are the two relations.*
- ***T*** *is a tuple variable.*
- *The logical condition* ***T ∈ R1 ∨ T ∈ R2*** *indicates that the tuple T belongs to either R1 or R2.*

Example 05:

Suppose we have two relations:

- ***Employees1****(name, salary)*
- ***Employees2****(name, salary)*

To get the union of the two relations:

{T | T ∈ Employees1 ∨ T ∈ Employees2}

This query returns all tuples that are present in ***Employees1****,* ***Employees2****, or both.*

4.2 Intersection:

In ***Tuple Relational Calculus (TRC)****, the intersection of two relations is expressed by specifying that a tuple must belong to* ***both*** *relations. This is achieved using the logical operator* ***AND (∧)****.*

General Syntax for Intersection in TRC:

{T | T ∈ R1 ∧ T ∈ R2}

- ***R1*** *and* ***R2*** *are the two relations.*
- ***T*** *is a tuple variable.*
- *The condition* ***T ∈ R1 ∧ T ∈ R2*** *ensures that the tuple T exists in both R1 and R2.*

Example 06:

Suppose we have two relations:

- ***Employees1****(name, salary)*
- ***Employees2****(name, salary)*

To get the intersection of the two relations:

{T | T ∈ Employees1 ∧ T ∈ Employees2}

This query returns all tuples that are present in ***both*** *Employees1 and Employees2.*

4.3 Difference:

*In **Tuple Relational Calculus (TRC)**, the **difference** of two relations is expressed by specifying that a tuple belongs to the first relation but **not** to the second relation. This is achieved using the logical operator **AND (∧)** combined with **NOT (¬)**.*

General Syntax for Difference in TRC:

{T | T ∈ R1 ∧ ¬(T ∈ R2)}

- ***R1** and **R2** are the two relations.*
- ***T** is a tuple variable.*
- *The condition **T ∈ R1 ∧ ¬(T ∈ R2)** ensures that the tuple T is in **R1** but **not** in **R2**.*

Example 07:

Suppose we have two relations:

- ***Employees1**(name, salary)*
- ***Employees2**(name, salary)*

To get the difference of the two relations (i.e., tuples in Employees1 but not in Employees2):

{T | T ∈ Employees1 ∧ ¬(T ∈ Employees2)}

*This query returns all tuples that are present in **Employees1** but not in **Employees2**.*

4.4. Projection

*In **Tuple Relational Calculus (TRC)**, **projection** is used to select specific attributes (columns) from a relation by filtering out other attributes. It is expressed by specifying a tuple variable and listing the desired attributes to retrieve.*

General Syntax for Projection in TRC:

{T.attribute1, T.attribute2, ... | T ∈ R}

- ***T** is a tuple variable.*
- ***attribute1, attribute2, ...** are the attributes (columns) that you want to project (retrieve).*
- ***R** is the relation from which the projection is made.*

Example 08:

Suppose we have a relation:

- *Employees(name, salary, department)*

*To project only the **name** and **salary** columns from the relation Employees:*

{T.name, T.salary | T ∈ Employees}

*This query retrieves all tuples with only the **name** and **salary** attributes, excluding the **department** attribute.*

4.5 Selection:

*In **Tuple Relational Calculus (TRC)**, **selection** is used to filter rows based on certain conditions. It is expressed by specifying a tuple variable and applying a condition (predicate) that the tuple must satisfy.*

General Syntax for Selection in TRC:

{T | T ∈ R ∧ condition(T)}

- ***T** is a tuple variable.*
- ***R** is the relation from which the tuples are selected.*
- ***condition(T)** is a predicate that filters the tuples, ensuring only those that satisfy the condition are selected.*

Example 09:

Suppose we have a relation:

Employees(name, salary, department)

To select the employees whose salary is greater than 5000:

{T | T ∈ Employees ∧ T.salary > 5000}

*This query retrieves all tuples from the Employees relation where the **salary** attribute is greater than 5000.*

4.6 Join

*In **Tuple Relational Calculus (TRC)**, a **join** operation is used to combine tuples from two relations based on a matching condition. The join condition specifies how the tuples from the two relations are related, typically based on matching attributes.*

The syntax for a join in TRC combines the conditions for both relations and specifies the criteria for matching the tuples.

General Syntax for Join in TRC:

{T1, T2 | T1 ∈ R1 ∧ T2 ∈ R2 ∧ condition(T1, T2)}

- ***T1** and **T2** are tuple variables for the relations **R1** and **R2**, respectively.*
- ***R1** and **R2** are the two relations being joined.*

- ***condition(T1, T2)*** *is the condition that specifies how the tuples from* ***R1*** *and* ***R2*** *are related (e.g., matching a common attribute).*

Example 10:

Suppose we have two relations:

- ***Employees****(name, emp_id, dep_id)*
- ***Departments****(dep_id, name)*

To perform a ***natural join*** *between Employees and Departments based on a matching dep_id:*

{T1.name, T2.name | T1 ∈ Employees ∧ T2 ∈ Departments ∧ T1.dep_id = T2.dep_id}

This query retrieves the ***name*** *of employees and their corresponding* ***department name*** *by joining the Employees and Departments relations based on the matching dep_id attribute.*

4.7 Division

In ***Tuple Relational Calculus (TRC)****, the* ***division*** *operation is used to find tuples in one relation that are associated with all tuples in another relation. This operation is often used when you want to find records in one relation that match a set of values in another relation, essentially performing a "for all" check.*

General Syntax for Division in TRC:

{T1 | T1 ∈ R1 ∧ ∀T2 (T2 ∈ R2 → ∃T3 (T3 ∈ R1 ∧ T3.attribute = T1.attribute ∧ T3.other_attribute = T2.other_attribute))}

- ***R1*** *and* ***R2*** *are the two relations involved in the division.*
- ***T1*** *is the tuple variable for the relation from which we are retrieving results.*
- *The condition* ***∀T2 (T2 ∈ R2 → ∃T3 (T3 ∈ R1 ∧ ...))*** *ensures that for every tuple in* ***R2****, there is a corresponding tuple in* ***R1*** *that matches.*

Example 11:

Suppose we have two relations:

- ***Employees****(emp_id, skill)*
- ***Skills****(skill)*

If we want to find the emp_id of employees who have all the skills listed in the Skills relation (i.e., those who have every skill), the division operation in TRC would look like this:

{T1.emp_id | T1 ∈ Employees ∧ ∀T2 (T2 ∈ Skills → ∃T3 (T3 ∈ Employees ∧ T3.emp_id = T1.emp_id ∧ T3.skill = T2.skill))}

This query retrieves the emp_id of employees who have every skill listed in the Skills relation. The ***∀T2 (T2 ∈ Skills → ...)*** *ensures that for each skill in Skills, there is a corresponding tuple in Employees for that employee.*

Exercise :

Consider the following two relations:

Students*(student_id, name, age)*

Courses*(course_id, course_name, student_id)*

Questions:

1. ***Query 1:***

Write a TRC query to find the names of all students who are older than 20.

2. ***Query 2:***

Write a TRC query to find the names of all students who are enrolled in both "Math" and "Science" courses.

3. ***Query 3:***

Write a TRC query to find the names of students who are not enrolled in any course.

Solution:

1. ***{T.name | T ∈ Students ∧ T.age > 20}***
2. ***{T1.name | T1 ∈ Students ∧ ∃ T2 ∈ Courses ∧ (T1.student_id = T2.student_id) ∧ T2.course_name ='math') ∧ ∃ T3 ∈ Courses∧ (T1.student_id = T3.student_id) ∧ T3.course_name ='science') }***
3. ***{T1.name | T1∈ Students ∧ ¬(T2 ∈ Courses ∧ T1.student_id= T2.student_id)}***