UNIT-2

Introduction to the Relational Model: Integrity constraint over relations, enforcing integrity constraints, querying relational data, logical database design, introduction to views, destroying/altering tables and views. Relational Algebra, Tuple relational Calculus, Domain relational calculus.

INTEGRITY CONSTRAINTS OVER RELATION

- Database integrity refers to the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate. Constraints may apply to each attribute or they may apply to relationships between tables.
- Integrity constraints ensure that changes (update deletion, insertion) made to the database by authorized users do not result in a loss of data consistency. Thus, integrity constraints guard against accidental damage to the database.

EXAMPLE- A brood group must be ‘A’ or ‘B’ or ‘AB’ or ‘O’ only (can not any other values else).

TYPES OF INTEGRITY CONSTRAINTS

Various types of integrity constraints are-

1. Domain Integrity
2. Entity Integrity Constraint
3. Referential Integrity Constraint
4. Key Constraints

1. Domain Integrity-

Domain integrity means the definition of a valid set of values for an attribute. You define data type, length or size, is null value allowed , is the value unique or not for an attribute ,the default value, the range (values in between) and/or specific values for the attribute.

2. Entity Integrity Constraint-

This rule states that in any database relation value of attribute of a primary key can't be null.

EXAMPLE- Consider a relation "STUDENT" Where "Stu_id" is a primary key and it must not contain any null value whereas other attributes may contain null value e.g "Branch" in the following relation contains one null value.

Stu_id	Name	Branch
11255234	Aman	CSE
11255369	Kapil	EcE
11255324	Ajay	ME
11255237	Raman	CSE
11255678	Aastha	ECE

3.Referential Integrity Constraint-

It states that if a foreign key exists in a relation then either the foreign key value must match a primary key value of some tuple in its home relation or the foreign key value must be null.

The rules are:

1. You can't delete a record from a primary table if matching records exist in a related table.
2. You can't change a primary key value in the primary table if that record has related records.
3. You can't enter a value in the foreign key field of the related table that doesn't exist in the primary key of the primary table.
4. However, you can enter a Null value in the foreign key, specifying that the records are unrelated.

EXAMPLE-

Consider 2 relations "stu" and "stu_1" Where "Stu_id " is the primary key in the "stu" relation and foreign key in the "stu_1" relation.

Relation "stu"

Stu_id	Name	Branch
11255234	Aman	CSE
11255369	Kapil	EcE
11255324	Ajay	ME
11255237	Raman	CSE
11255678	Aastha	ECE

Relation "stu_1"

Stu_id	Course	Duration
11255234	B TECH	4 years
11255369	B TECH	4 years
11255324	B TECH	4 years

Stu_id	Course	Duration
11255237	B TECH	4 years
11255678	B TECH	4 years

Examples

Rule 1. You can't delete any of the rows in the ”stu” relation that are visible since all the ”stu” are in use in the “stu_1” relation.

Rule 2. You can't change any of the ”Stu_id” in the “stu” relation since all the “Stu_id” are in use in the ”stu_1” relation. * *Rule 3.** The values that you can enter in the” Stu_id” field in the “stu_1” relation must be in the” Stu_id” field in the “stu” relation.

Rule 4 You can enter a null value in the "stu_1" relation if the records are unrelated.

4.Key Constraints-

A Key Constraint is a statement that a certain minimal subset of the fields of a relation is a unique identifier for a tuple. The types of key constraints-

1. Primary key constraints
2. Unique key constraints
3. Foreign Key constraints
4. NOT NULL constraints
5. Check constraints

1. Primary key constraints

Primary key is the term used to identify one or more columns in a table that make a row of data unique. Although the primary key typically consists of one column in a table, more than one column can comprise the primary key.

For example, either the employee's Social Security number or an assigned employee identification number is the logical primary key for an employee table. The objective is for every record to have a unique primary key or value for the employee's identification number. Because there is probably no need to have more than one record for each employee in an employee table, the employee identification number makes a logical primary key. The primary key is assigned at table creation.

The following example identifies the EMP_ID column as the PRIMARY KEY for the EMPLOYEES table:

```
CREATE TABLE EMPLOYEE_TBL
(EMP_ID       CHAR(9)       NOT NULL PRIMARY KEY,
EMP_NAME      VARCHAR (40)   NOT NULL,
EMP_ST_ADDR   VARCHAR (20)   NOT NULL,
EMP_CITY      VARCHAR (15)   NOT NULL,
EMP_ST        CHAR(2)       NOT NULL,
EMP_ZIP       INTEGER(5)    NOT NULL,
EMP_PHONE     INTEGER(10)   NULL,
EMP_PAGER     INTEGER(10)   NULL);
```

2. Unique Constraints

A unique column constraint in a table is similar to a primary key in that the value in that column for every row of data in the table must have a unique value. Although a primary key constraint is placed on one column, you can place a unique constraint on another column even though it is not actually for use as the primary key.

```
CREATE TABLE EMPLOYEE_TBL
(EMP_ID       CHAR(9)       NOT NULL     PRIMARY KEY,
EMP_NAME      VARCHAR (40)   NOT NULL,
EMP_ST_ADDR   VARCHAR (20)   NOT NULL,
EMP_CITY      VARCHAR (15)   NOT NULL,
EMP_ST        CHAR(2)       NOT NULL,
EMP_ZIP       INTEGER(5)    NOT NULL,
EMP_PHONE     INTEGER(10)   NULL         UNIQUE,
EMP_PAGER     INTEGER(10)   NULL)
```

3. Foreign Key Constraints

A foreign key is a column in a child table that references a primary key in the parent table. A foreign key constraint is the main mechanism used to enforce referential integrity between tables in a relational database. A column defined as a foreign key is used to reference a column defined as a primary key in another table.

```
CREATE TABLE EMPLOYEE_PAY_TBL
(EMP_ID         CHAR(9)       NOT NULL,
```

```
POSITION          VARCHAR2(15)  NOT NULL,
DATE_HIRE         DATE          NULL,
PAY_RATE          NUMBER(4,2)   NOT NULL,
DATE_LAST_RAISE   DATE          NULL,
```

4. NOT NULL Constraints

Previous examples use the keywords NULL and NOT NULL listed on the same line as each column and after the data type. NOT NULL is a constraint that you can place on a table's column. This constraint disallows the entrance of NULL values into a column; in other words, data is required in a NOT NULL column for each row of data in the table. NULL is generally the default for a column if NOT NULL is not specified, allowing NULL values in a column.

5. Check Constraints

Check (CHK) constraints can be utilized to check the validity of data entered into particular table columns. Check constraints are used to provide back-end database edits, although edits are commonly found in the front-end application as well. General edits restrict values that can be entered into columns or objects, whether within the database itself or on a front-end application. The check constraint is a way of providing another protective layer for the data.

```
CREATE TABLE EMPLOYEE_TBL
(EMP_ID       CHAR(9)       NOT NULL,
EMP_NAME      VARCHAR2(40)  NOT NULL,
EMP_ST_ADDR   VARCHAR2(20)  NOT NULL,
EMP_CITY      VARCHAR2(15)  NOT NULL,
EMP_ST        CHAR(2)       NOT NULL,
EMP_ZIP       NUMBER(5)     NOT NULL,
EMP_PHONE     NUMBER(10)    NULL,
EMP_PAGER     NUMBER(10)    NULL),
PRIMARY KEY (EMP_ID),
CONSTRAINT CHK_EMP_ZIP CHECK ( EMP_ZIP = '46234');
```

Relational Algebra

Relational Algebra is a procedural query language. Relational algebra mainly provides a theoretical foundation for relational databases and SQL. The main purpose of using Relational Algebra is to define operators that transform one or more input relations into an output relation. Given that these operators accept relations as input and produce relations as output, they can be combined and used to express potentially complex queries that transform potentially many input

relations (whose data are stored in the database) into a single output relation (the query results). As it is pure mathematics, there is no use of English Keywords in Relational Algebra and operators are represented using symbols.

Fundamental Operators

These are the basic/fundamental operators used in Relational Algebra.

1. Selection(σ)
2. Projection(π)
3. Union(U)
4. Set Difference(-)
5. Set Intersection(∩)
6. Rename(ρ)
7. Cartesian Product(X)

1. Selection(σ): It is used to select required tuples of the relations.

Example:

A	**B**	**C**
1	2	4
2	2	3
3	2	3
4	3	4

For the above relation, **σ(c>3)R** will select the tuples which have c more than 3.

A	**B**	**C**
1	2	4
4	3	4

Note: The selection operator only selects the required tuples but does not display them. For display, the data projection operator is used.

2. Projection(π): It is used to project required column data from a relation.

Example: Consider Table 1. Suppose we want columns B and C from Relation R.

π(B,C)R will show following columns.

B	C
2	4
2	3
3	4

Note: By Default, projection removes duplicate data.

3. Union(U): Union operation in relational algebra is the same as union operation in set theory.

Example:

FRENCH

Student_Name	Roll_Number
Ram	01
Mohan	02
Vivek	13
Geeta	17

GERMAN

Student_Name	Roll_Number
Vivek	13
Geeta	17
Shyam	21
Rohan	25

Consider the following table of Students having different optional subjects in their course.

π(Student_Name)FRENCH U π(Student_Name)GERMAN

Student_Name
Ram
Mohan
Vivek
Geeta
Shyam
Rohan

Note: The only constraint in the union of two relations is that both relations must have the same set of Attributes.

4. Set Difference(-): Set Difference in relational algebra is the same set difference operation as in set theory.

Example: From the above table of FRENCH and GERMAN, Set Difference is used as follows

π(Student_Name)FRENCH - π(Student_Name)GERMAN

Student_Name
Ram
Mohan

Note: The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

5. Set Intersection(∩): Set Intersection in relational algebra is the same set intersection operation in set theory.

Example: From the above table of FRENCH and GERMAN, the Set Intersection is used as follows

π(Student_Name)FRENCH ∩ π(Student_Name)GERMAN

Student_Name
Vivek
Geeta

Note: The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

6. Rename(ρ): Rename is a unary operation used for renaming attributes of a relation.

ρ(a/b)R will rename the attribute 'b' of the relation by 'a'.

7. Cross Product(X): Cross-product between two relations. Let's say A and B, so the cross product between A X B will result in all the attributes of A followed by each attribute of B. Each record of A will pair with every record of B.

Example:

A

Name	Age	Sex
Ram	14	M
Sona	15	F
Kim	20	M

B

ID	Course
1	DS
2	DBMS

A X B

Name	Age	Sex	ID	Course
Ram	14	M	1	DS
Ram	14	M	2	DBMS

Name	Age	Sex	ID	Course
Sona	15	F	1	DS
Sona	15	F	2	DBMS
Kim	20	M	1	DS
Kim	20	M	2	DBMS

Note: If A has ‘n’ tuples and B has ‘m’ tuples then A X B will have ‘ n*m ‘ tuples.

Derived Operators

These are some of the derived operators, which are derived from the fundamental operators.

1. Natural Join(⋈)
2. Conditional Join

1. Natural Join(⋈): Natural join is a binary operator. Natural join between two or more relations will result in a set of all combinations of tuples where they have an equal common attribute.

Example:

EMP

Name	ID	Dept_Name
A	120	IT
B	125	HR
C	110	Sales
D	111	IT

DEPT

Dept_Name	Manager
Sales	Y
Production	Z

Dept_Name	Manager
IT	A

Natural join between EMP and DEPT with condition :

EMP.Dept_Name = DEPT.Dept_Name

EMP ⋈ DEPT

Name	ID	Dept_Name	Manager
A	120	IT	A
C	110	Sales	Y
D	111	IT	A

2. Conditional Join: Conditional join works similarly to natural join. In natural join, by default condition is equal between common attributes while in conditional join we can specify any condition such as greater than, less than, or not equal.

Example:

R

ID	Sex	Marks
1	F	45
2	F	55
3	F	60

S

ID	Sex	Marks
10	M	20
11	M	22

ID	Sex	Marks
12	M	59

Join between R and S with condition **R.marks >= S.marks**

R.ID	R.Sex	R.Marks	S.ID	S.Sex	S.Marks
1	F	45	10	M	20
1	F	45	11	M	22
2	F	55	10	M	20
2	F	55	11	M	22
3	F	60	10	M	20
3	F	60	11	M	22
3	F	60	12	M	59

Relational Calculus

As Relational Algebra is a procedural query language, Relational Calculus is a non-procedural query language. It basically deals with the end results. It always tells me what to do but never tells me how to do it.

There are two types of Relational Calculus

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

Tuple Relational Calculus (TRC) in DBMS

Tuple Relational Calculus (TRC) is a non-procedural query language used in relational database management systems (RDBMS) to retrieve data from tables. TRC is based on the

concept of tuples, which are ordered sets of attribute values that represent a single row or record in a database table.
TRC is a declarative language, meaning that it specifies what data is required from the database, rather than how to retrieve it. TRC queries are expressed as logical formulas that describe the desired tuples.
Syntax: The basic syntax of TRC is as follows:

```
{ t | P(t) }
```

where t is a **tuple variable** and P(t) is a **logical formula** that describes the conditions that the tuples in the result must satisfy. The **curly braces {}** are used to indicate that the expression is a set of tuples.
For example, let's say we have a table called "Employees" with the following attributes:

Employee ID
Name
Salary
Department ID

To retrieve the names of all employees who earn more than $50,000 per year, we can use the following TRC query:

{ t | Employees(t) ∧ t.Salary > 50000 }

In this query, the "Employees(t)" expression specifies that the tuple variable t represents a row in the "Employees" table. The "∧" symbol is the logical AND operator, which is used to combine the condition "t.Salary > 50000" with the table selection.

The result of this query will be a set of tuples, where each tuple contains the Name attribute of an employee who earns more than $50,000 per year.

TRC can also be used to perform more complex queries, such as joins and nested queries, by using additional logical operators and expressions.

While TRC is a powerful query language, it can be more difficult to write and understand than other SQL-based query languages, such as Structured Query Language (SQL). However, it is useful in certain applications, such as in the formal verification of database schemas and in academic research.
Tuple Relational Calculus is a **non-procedural query language,** unlike relational algebra. Tuple Calculus provides only the description of the query but it does not provide the methods to solve it. Thus, it explains what to do but not how to do it.
Tuple Relational Query

In Tuple Calculus, a query is expressed as

```
{t| P(t)}
```

where t = resulting tuples,
P(t) = known as Predicate and these are the conditions that are used to fetch t. Thus, it generates a set of all tuples t, such that Predicate P(t) is true for t.

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.

Domain Relational Calculus (DRC)
Domain Relational Calculus is similar to Tuple Relational Calculus, where it makes a list of the attributes that are to be chosen from the relations as per the conditions.

```
{<a1,a2,a3,.....an> | P(a1,a2,a3,.....an)}
```

where a1,a2,…an are the attributes of the relation and P is the condition.

Tuple Relational Calculus Examples
Table Customer

Customer name	Street	City
Saurabh	A7	Patiala
Mehak	B6	Jalandhar
Sumiti	D9	Ludhiana
Ria	A5	Patiala

Table Branch

Branch name	Branch City
ABC	Patiala

Branch name	Branch City
DEF	Ludhiana
GHI	Jalandhar

Table Account

Account number	Branch name	Balance
1111	ABC	50000
1112	DEF	10000
1113	GHI	9000
1114	ABC	7000

Table Loan

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L49	GHI	9000

Loan number	Branch name	Amount
L98	DEF	65000

Table Borrower

Customer name	Loan number
Saurabh	L33
Mehak	L49
Ria	L98

Table Depositor

Customer name	Account number
Saurabh	1111
Mehak	1113
Suniti	1114

Example 1: Find the loan number, branch, and amount of loans greater than or equal to 10000 amount.

$\{t \mid t \in loan \ \wedge t[amount] >= 10000\}$

Resulting relation:

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L98	DEF	65000

In the above query, t[amount] is known as a tuple variable.

Example 2: Find the loan number for each loan of an amount greater or equal to 10000.

$\{t \mid \exists\ s \in loan(t[loan\ number] = s[loan\ number]$

$\wedge\ s[amount] >= 10000)\}$

Resulting relation:

Loan number
L33
L35
L98

Example 3: Find the names of all customers who have a loan and an account at the bank.

$\{t \mid \exists\ s \in borrower(\ t[customer\text{-}name] = s[customer\text{-}name])$

$\wedge\ \exists\ u \in depositor(\ t[customer\text{-}name] = u[customer\text{-}name])\}$

Resulting relation:

Customer name
Saurabh
Mehak

Example 4: Find the names of all customers having a loan at the "ABC" branch.

$$\{t \mid \exists\ s \in borrower(t[customer\text{-}name] = s[customer\text{-}name] \wedge \exists\ u \in loan(u[branch\text{-}name] = \text{“ABC”} \wedge u[loan\text{-}number] = s[loan\text{-}number]))\}$$

Resulting relation:

Customer name
Saurabh

Logical Database Design

Logical database design is the process of determining the logical data structures that are required to support information resources within an organization. The logical design process helps you to implement a database that satisfies the requirements of your business organization.
Logical design is critical to the implementation of a corporate database. An incomplete or flawed logical design can cause costly changes to the means of data collection, storage, and protection later on. By using a well-conceived preliminary design, you can easily implement and test a database. A sound logical design therefore helps to ensure a successful implementation.
A complete and accurate logical design for a database helps to ensure:

Data independence

-- The logical design process yields a database model that is independent of program or physical storage requirements. This model represents the way data structures appear to users. The model does not specify how data structures are maintained or processed by the computer.

Physical database flexibility

-- Logical design is independent of storage and performance requirements. Therefore, you can use it to implement a database that is used with any hardware or software system. During the physical design process, the logical design can be tailored to satisfy the needs of particular users or to suit a particular data processing environment.

Integrity

-- The logical design identifies both the data that is maintained in your corporation and the rules of the business. These business rules can be used later to define integrity rules for the physical design.

User satisfaction

-- The logical design represents data structures in a simple, understandable format. You can show the design to users at any stage of development without intimidating them. The logical design can be easily modified to incorporate user suggestions and feedback.

Many viable approaches exist for logical database design. In this section, we combine several design techniques, including systems analysis, the entity-relationship approach, and normalization.

The entity-relationship approach was developed by Peter Chen. For more information, see

Entity-Relationship Approach to Information Modeling and Analysis

, Peter P. Chen, editor, ER Institute (1981).

By using these techniques, you can create a logical model that consists of:

- Descriptions of the data required by each user application
- A comprehensive picture of the data in a corporation

Relational Query Language in DBMS

SQL has its own querying methods to interact with the database. But how do these queries work in the database? These queries work similarly to Relational Algebra that we study in mathematics. In the database, we have tables participating in relational Algebra.

Relational Database systems are expected to be equipped with a query language that assists users to query the database. Relational Query Language is used by the user to communicate with the database user requests for the information from the database. Relational algebra breaks the user requests and instructs the DBMS to execute the requests. It is the language by which the user communicates with the database. They are generally on a higher level than any other programming language. These relational query languages can be Procedural and Non-Procedural.

Types of Relational Query Language

There are two types of relational query language:

- Procedural Query Language
- Non-Procedural Language

Procedural Query Language

In Procedural Language, the user instructs the system to perform a series of operations on the database to produce the desired results. Users tell what data to be retrieved from the database and how to retrieve it. Procedural Query Language performs a set of queries instructing the DBMS to perform various transactions in sequence to meet user requests.

Relational Algebra is a Procedural Query Language

Relational Algebra could be defined as the set of operations on relations.

There are a few operators that are used in relational algebra –

1. **Select (sigma):** Returns rows of the input relation that satisfy the provided predicate. It is unary Operator means requires only one operand.
2. **Projection (π):** Show the list of those attribute which we desire to appear and rest other attributes are eliminated from the table. It seperates the table vertically.
3. **Set Difference (-):** It returns the difference between two relations . If we have two relations R and S them R-S will return all the tuples (row) which are in relation R but not in Relation S , It is binary operator.
4. **Cartesian Product (X):** Combines every tuple (row) of one table with every tuple (row) in other table ,also referred as cross Product . It is a binary operator.
5. **Union (U):** Outputs the union of tuples from both the relations. Duplicate tuples are eliminated automatically. It is a binary operator means it require two operands.

Non-Procedural Language

In Non Procedural Language user outlines the desired information without giving a specific procedure or without telling the steps by step process for attaining the information. It only gives a single Query on one or more tables to get .The user tells what is to be retrieved from the database but does not tell how to accomplish it.

For Example: get the name and the contact number of the student with a Particular ID will have a single query on STUDENT table.

Relational Calculus is a Non Procedural Language .

Relational Calculus exists in two forms:

1. **Tuple Relational Calculus (TRC):** Tuple Relational Calculus is a non procedural query language , It is used for selecting the tuples that satisfy the given condition or predicate . The result of the relation can have one or more tuples (row).
2. **Domain Relational Calculus (DRC):** Domain Relational Calculus is a Non Procedural Query Language , the records are filtered based on the domains , DRC uses the list of attributes to be selected from relational based on the condition.

Relational Calculus

There is an alternate way of formulating queries known as Relational Calculus. Relational calculus is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results. The relational calculus tells what to do but never explains how to do. Most commercial relational languages are based on aspects of relational calculus including SQL-QBE and QUEL.

Why it is called Relational Calculus?

It is based on Predicate calculus, a name derived from branch of symbolic language. A predicate is a truth-valued function with arguments. On substituting values for the arguments, the function result in an expression called a proposition. It can be either true or

false. It is a tailored version of a subset of the Predicate Calculus to communicate with the relational database.

Many of the calculus expressions involves the use of Quantifiers. There are two types of quantifiers:

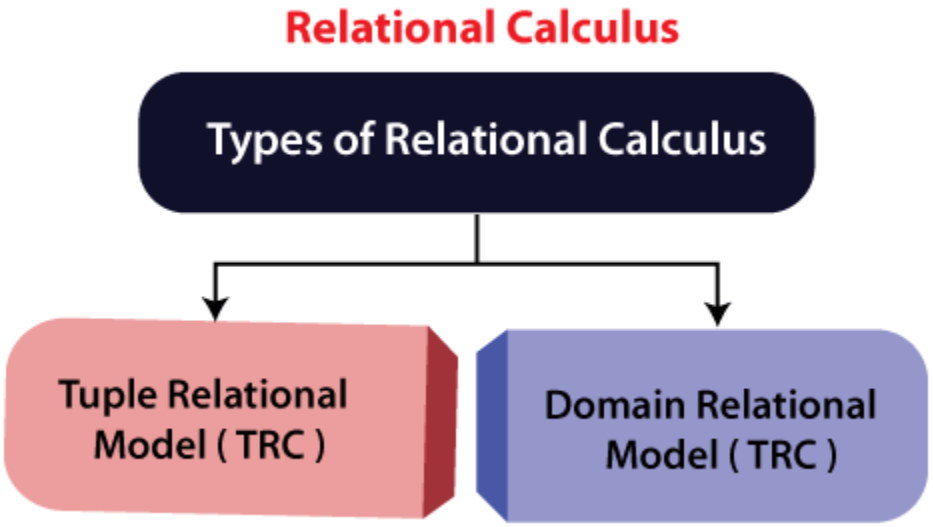
- **Universal Quantifiers:** The universal quantifier denoted by ∀ is read as for all which means that in a given set of tuples exactly all tuples satisfy a given condition.
- **Existential Quantifiers:** The existential quantifier denoted by ∃ is read as for all which means that in a given set of tuples there is at least one occurrences whose value satisfy a given condition.

Before using the concept of quantifiers in formulas, we need to know the concept of Free and Bound Variables.

A tuple variable t is bound if it is quantified which means that if it appears in any occurrences a variable that is not bound is said to be free.

Free and bound variables may be compared with global and local variable of programming languages.

Types of Relational calculus:



1. Tuple Relational Calculus (TRC)

It is a non-procedural query language which is based on finding a number of tuple variables also known as range variable for which predicate holds true. It describes the desired information without giving a specific procedure for obtaining that information. 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.

Notation:

A Query in the tuple relational calculus is expressed as following notation

1. {T | P (T)} or {T | Condition (T)}

Where

T is the resulting tuples

P(T) is the condition used to fetch T.

For example:

1. { 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 (∀).

For example:

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

Output: This query will yield the same result as the previous one.

2. Domain Relational Calculus (DRC)

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 ∧ (and), ∨ (or) and ¬ (not). It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable. The QBE or Query by example is a query language related to domain relational calculus.

Notation:

1. { a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

Where

a1, a2 are attributes
P stands for formula built by inner attributes

For example:

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

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