Unit - III

Relational Model

&

Storage Indexing

Relational Model

RELATIONAL MODEL: Introduction to Relational Model, Basic Structure, Database Schema, Keys, Relational Algebra and Relational Calculus.

Storage and Indexing: File Organizations and Indexing-Overview of Indexes, Types of Indexes, Index Data Structures, Tree structured Indexing, Hash based Indexing.

Introduction to Relational Model

- Relational data model is the primary data model, which is used widely around the world for data storage and processing.
- This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

Terminology Used in Basic Structure of Relational Model:

- **Tables** – In relational data model, relations are saved in the format of Tables. This format stores the relation among entities.
- A table has rows and columns, where rows represents records and columns represent the attributes.

- **Tuple** – A single row of a table, which contains a single record for that relation is called a tuple.

- **Relation instance** – A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Cont….

- **Relation schema** – A relation schema describes the relation name (table name), attributes, and their names.

- **Relation key** – Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

- **Attribute domain** – Every attribute has some pre-defined value scope, known as attribute domain.

Basic Structure of Relational Model

- A relational database is a collection of tables.
  - Each table has a unique name.
  - Each table consists of multiple rows.
  - Each row is a set of values that by definition are related to each other in some way;
  - these values conform to the attributes or columns of the table.

Features of the relational model

- **Conceptually simple:** the fundamentals are intuitive and easy to pick up.
- **Powerful underlying theory:** the relational model is the only database model that is powered by formal mathematics, which results in excellent dividends when developing database algorithms and techniques.
- **Easy-to-use database language:** though not formally part of the relational model, part of its success is due to SQL, the de facto language for working with relational databases.

Database Schemas

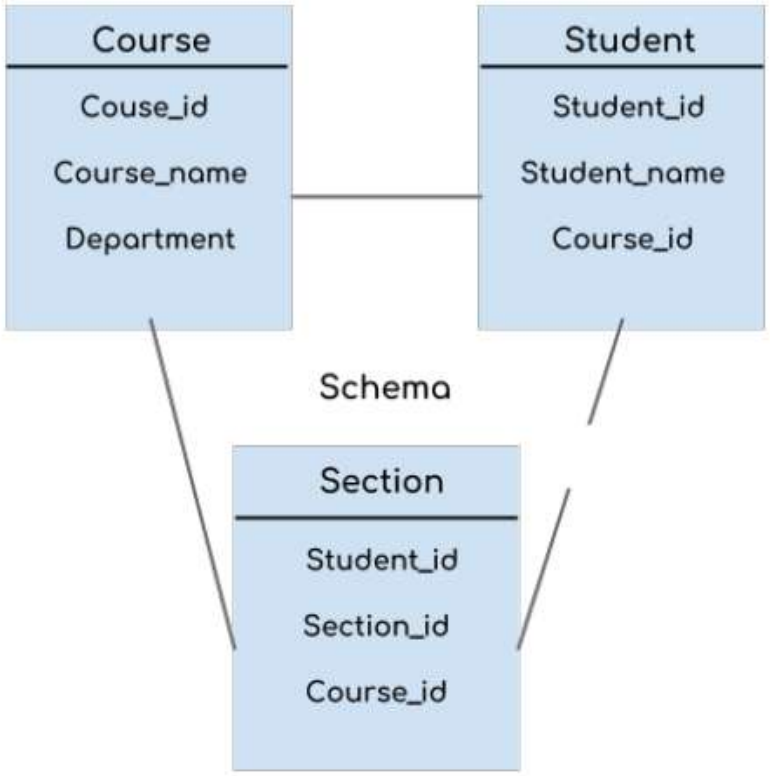
- **Definition of schema**: Design of a database is called the schema. Schema is of three types:
  1. Physical schema
  2. logical schema
  3. view schema.
- The design of a database at physical level is called **physical schema**, how the data stored in blocks of storage is described at this level.

- Design of database at logical level is called **logical schema**, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data structure is hidden at this level (available at physical level).

- Design of database at view level is called **view schema**. This generally describes end user interaction with database systems.

Example

- **For example:** In the following diagram, we have a schema that shows the relationship between three tables: Course, Student and Section.
- The diagram only shows the design of the database, it doesn't show the data present in those tables. Schema is only a structural view(design) of a database as shown in the diagram .



Database Instances

- **Definition of instance**: The data stored in database at a particular moment of time is called instance of database.

- Database schema defines the variable declarations in tables that belong to a particular database; The value of these variables at a moment of time is called the instance of that database.

- **For example:** lets say we have a single table student in the database, today the table has 100 records, so today the instance of the database has 100 records. Lets say we are going to add another 100 records in this table by tomorrow so the instance of database tomorrow will have 200 records in table. In short, at a particular moment the data stored in database is called the instance, that changes over time when we add or delete data from the database.

Relational Query Languages

- Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances.
- There are two kinds of query languages – relational algebra and relational calculus.

- **Query languages:** Allow manipulation and retrieval of data from a database.
- Query Languages **!=** programming languages!
  - QLs not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

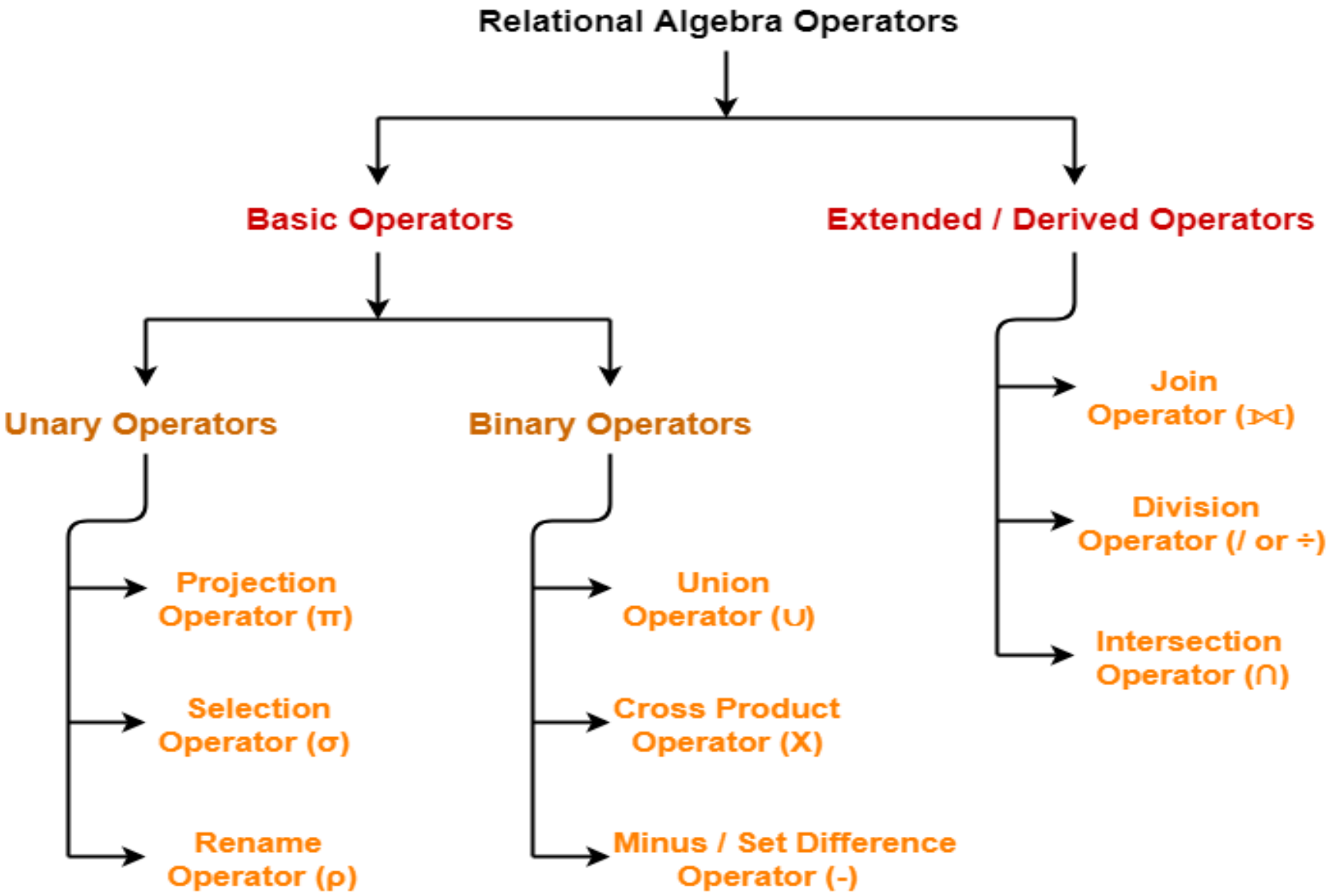
Formal Relational Query Languages

- Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:
  - ***Relational Algebra***: More operational (procedural), very useful for representing execution plans.
  - ***Relational Calculus***: Lets users describe what they want, rather than how to compute it: Non-operational, declarative.

Relational Algebra

- Relational algebra is a procedural query language, which takes **instances of relations as input** and **yields instances of relations as output.**
- It uses operators to perform queries. An operator can be either **unary** or **binary**.
- They accept relations as their input and yield relations as their output.
- Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

Relational Algebra



Projection (π)

- This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.
- It is denoted by ∏.
- **Notation: ∏ A1, A2....An (r)**
- Where **A1**, **A2**, **A3** is used as an attribute name of relation **r.**
- **Input:**
- **∏ NAME, CITY (CUSTOMER)**

Example: CUSTOMER RELATION

NAME	STREET	CITY
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison
Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Output:

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

Selection (σ)

- Selects rows that satisfy *selection condition*.
- **Notation** – $\sigma_p(r)$
- Where
  - **σ** stands for selection predicate and
  - **r** stands for relation.
  - ***p*** is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like – =, ≠, ≥, < , >, ≤. No duplicates in result.

Example: σ BRANCH_NAME="perryride" (LOAN)

For example: LOAN Relation

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

Output:

BRANCH_NAME	LOAN_NO	AMOUNT
Perryride	L-15	1500
Perryride	L-16	1300

Rename(ρ)

- The rename operation is used to rename the output relation. It is denoted by **rho** (ρ).
- **Example:** We can use the rename operator to rename STUDENT relation to STUDENT1.
- ρ(STUDENT1, STUDENT)

Union(υ)

- UNION is symbolized by ∪ symbol.
- It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples.
- So, set A UNION set B would be expressed as:
- **The result : A ∪ B**
- For a union operation to be valid, the following conditions must hold
  - **A and B must be the same number of attributes.**
  - **Attribute domains need to be compatible.**
  - **Duplicate tuples should be automatically removed.**
- **Example:**
  - **Π_{author} (Books) ∪ Π_{author} (Articles)**

Union(∪) Example:

A ∪ B

Table A		Table B	
column 1	column 2	column 1	column 2
1	1	1	1
1	2	1	3

A ∪ B gives

Table A ∪ B	
column 1	column 2
1	1
1	2
1	3

Cross-Product (Cartesian Product)(X)

- Cross product between two relations let say A and B, so cross product between A X B will results all the attributes of A followed by each attribute of B. Each record of A will pairs with every record of B.
- ***Note:*** *if A has 'n' tuples and B has 'm' tuples then A X B will have 'n*m' tuples.*
- ***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
Sona	15	F	1	DS
Sona	15	F	2	DBMS
Kim	20	M	1	DS
Kim	20	M	2	DBMS

Set Difference(-)

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.
- It is denoted by intersection minus (-).
- **Notation: R – S**
- **Example:** Using the above DEPOSITOR table and BORROW table
- **Input:**
- ∏ CUSTOMER_NAME (BORROW) - ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Jackson
Hayes
Willians
Curry

Join Operator (⋈)

- Join operation is essentially a cartesian product followed by a selection criterion.
- Join operation denoted by ⋈.
- JOIN operation also allows joining variously related tuples from different relations.
- **Types of JOIN:**
- **Inner Joins:**
  - Theta join
  - EQUI join
  - Natural join
- **Outer join:**
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join

Inner Join

- In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:

- **Theta Join (θ)**
- The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ**
- **Example:** $A \bowtie_{\theta} B$
- Theta join can use any conditions in the selection criteria.
- **For example:**
- $A \bowtie_{A.column\ 2 > B.column\ 2} (B)$

EQUI Join

- When a theta join uses only equivalence condition, it becomes a equi join.
- **For example:**
- $A \bowtie_{A.column\ 2 = B.column\ 2} (B)$
- EQUI join is the most difficult operations to implement efficiently using SQL in an RDBMS and one reason why RDBMS have essential performance problems.

Natural Join(⋈)

- Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.
- **Example:**

C

Num	Square
2	4
3	9

D

Num	Cube
2	8
3	27

C ⋈ D

Num	Square	Cube
2	4	8
3	9	27

Outer Join

- In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.
- **Left Outer Join($A \bowtie B$)**
- In the left outer join operation allows keeping all tuple in the left relation.
- However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.

- **Right Outer Join(A ⟖ B)**
- In the right outer join, operation allows keeping all tuple in the right relation.
- However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.

- **Full Outer Join: (A ⟗ B)**
- In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

Division Operator(÷)

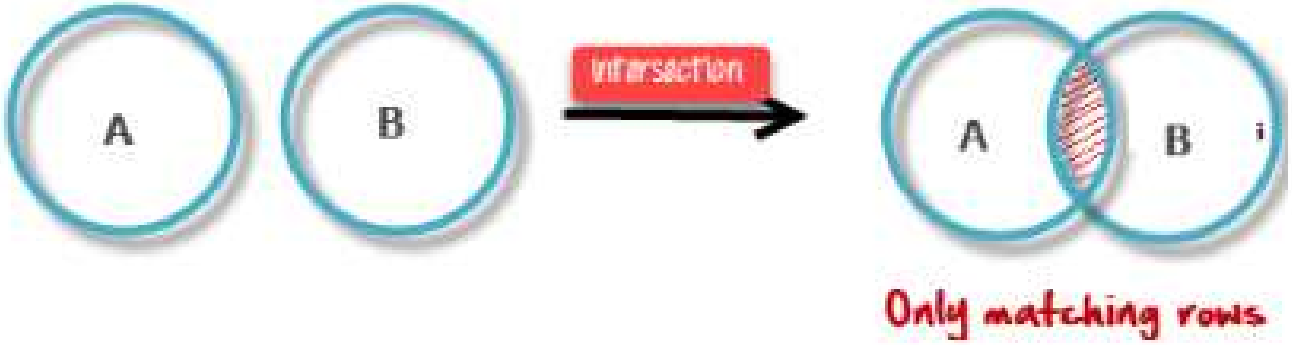
- R1 ÷ R2 = tuples of R1 associated with all tuples of R2.
- Attributes of R2 is proper subset of Attributes of R1.

- The relation returned by division operator will have attributes = (All attributes of R1 – All Attributes of R2).

- The relation returned by division operator will return those tuples from relation R1 which are associated to every R2's tuple.

Intersection(∩)

- An intersection is defined by the symbol ∩
- **Notation: A ∩ B**
- Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.



Summary

Operation(Symbols)	Purpose
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(∪)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference(-)	– Symbol denotes it. The result of A – B, is a relation which includes all tuples that are in A but not in B.
Intersection(∩)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(X)	Cartesian operation is helpful to merge columns from two relations.

Summary

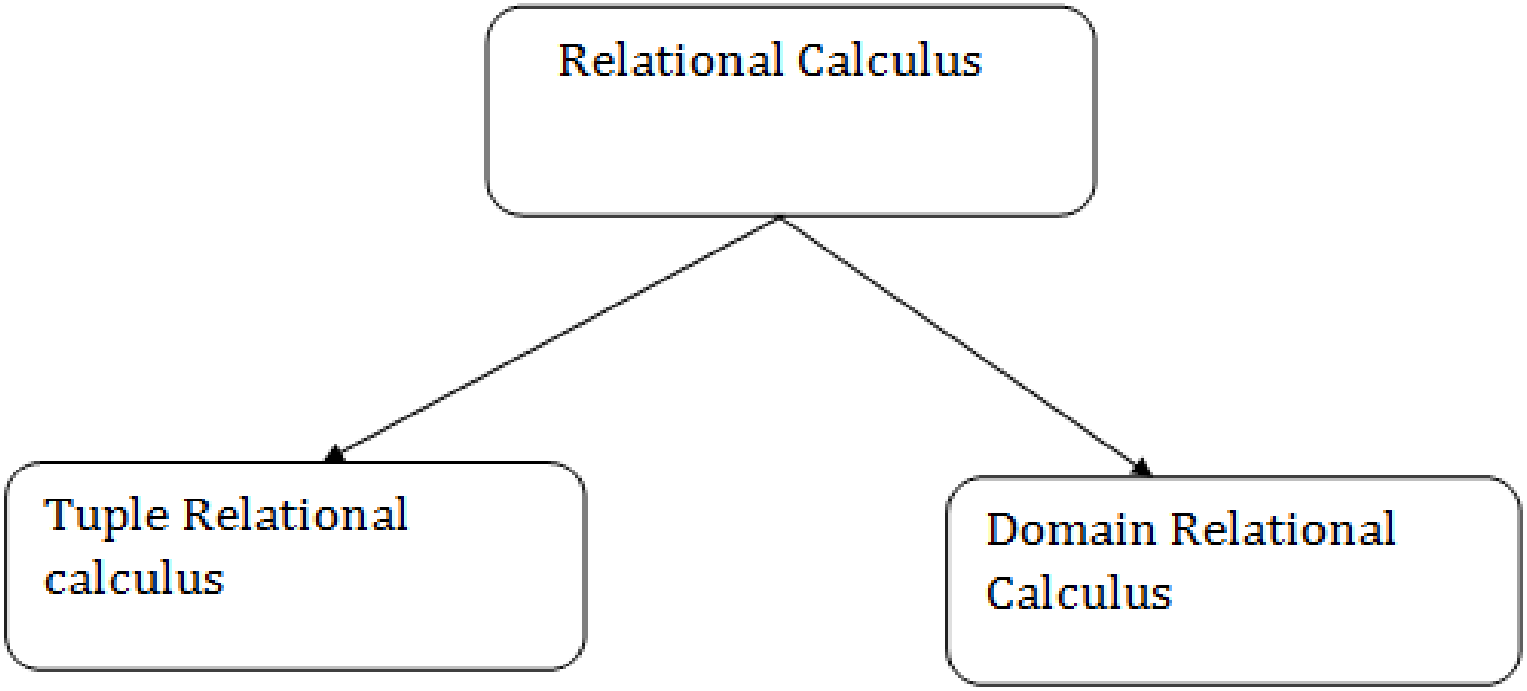
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ.
EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.
Natural Join(⋈)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join(⟕)	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join(⟖)	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join(⟗)	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

Relational Calculus

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.

Types of Relational calculus:



1. Tuple Relational Calculus (TRC)

- 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:

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

- Where, **T** is the resulting tuples
- **P(T)** is the condition used to fetch T.

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'.

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.

- **Notation:**
- { a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

  Where

  a1, a2 are attributes

  P stands for formula built by inner attributes

- **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.