DBMS Unit-03

By:
Mani Butwall
Asst. Prof. (CSE Dept.)

Contents

- Relational Algebra
- Relational Calculus
- Tuple and domain relational calculus
- SQL
- DDL and DML constructs
- Aggregate Functions

Query Language

- In simple words, a Language which is used to store and retrieve data from database is known as query language.
- For example – **SQL**
- There are two types of query language:
  1.Procedural Query language
  2.Non-procedural query language

1. **Procedural Query language:**

- In procedural query language, user instructs the system to perform a series of operations to produce the desired results. Here users tells what data to be retrieved from database and how to retrieve it.
- **For example –** Let's take a real world example to understand the procedural language, you are asking your younger brother to make a cup of tea, if you are just telling him to make a tea and not telling the process then it is a non-procedural language, however if you are telling the step by step process like switch on the stove, boil the water, add milk etc. then it is a procedural language.

2. **Non-procedural query language:**

- In Non-procedural query language, user instructs the system to produce the desired result without telling the step by step process. Here users tells what data to be retrieved from database but doesn't tell how to retrieve it.

Relational Algebra

- Relational Algebra is a Procedural language.
- In Relational Algebra, The order is specified in which the operations have to be performed. In Relation Algebra frameworks are created to implement the queries. The basic operation included in relational algebra are:
- **1.** Select (σ)
- **2.** Project (Π)
- **3.** Union (U)
- **4.** Set Difference (-)
- **5.** Cartesian product (X)
- **6.** Rename (ρ)

Relational Calculus

- Relational Calculus is the formal query language. It also known as **Declarative language**. In Relational Calculus, The order is not specified in which the operation have to be performed. Relational Calculus means what result we have to obtain.
  Relational Calculus has two variations:

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

Relational Algebra and Relational Calculus

S.NO	RELATIONAL ALGEBRA	RELATIONAL CALCULUS
1.	It is a Procedural language.	While Relational Calculus is Declarative language.
2.	Relational Algebra means how to obtain the result.	While Relational Calculus means what result we have to obtain.
3.	In Relational Algebra, The order is specified in which the operations have to be performed.	While in Relational Calculus, The order is not specified.
4.	Relational Algebra is independent on domain.	While Relation Calculus can be a domain dependent.
5.	Relational Algebra is nearer to a programming language.	While Relational Calculus is not nearer to programming language.

Relation Algebra

- **RELATIONAL ALGEBRA** is a widely used procedural query language.
- It collects instances of relations as input and gives occurrences of relations as output. It uses various operations to perform this action.
- SQL Relational algebra query operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations.

1. Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol: ρ)

2. Relational Algebra Operations From Set Theory

- UNION (υ)
- INTERSECTION ( ),
- DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

3. Binary Relational Operations

- JOIN
- DIVISION

Select

- The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operator selects tuples that satisfy a given predicate.

- $\sigma_p(r)$

1. σ is the predicate
2. r stands for relation which is the name of the table
3. p is prepositional logic

- **Example 1**
  - $\sigma_{\text{topic = "Database"}}$ (University)
- **Output -**
  - Selects tuples from University where topic = 'Database'.
- **Example 2**
  - $\sigma_{\text{topic = "Database" and author = “Ram"}}$(University)
- **Output -**
  - Selects tuples from University where the topic is 'Database' and 'author' is Ram.
- **Example 3**
  - $\sigma_{\text{sales > 50000}}$ (Customers)
- **Output -**
  - Selects tuples from Customers where sales is greater than 50000

Example 4

$$\sigma_{subject = "database"}(Books)$$

Output -

Selects tuples from books where subject is 'database'.

- **Example 5**
  - $\sigma_{subject = "database" \text{ and } price = "450"}(Books)$
- **Output** -

Selects tuples from books where subject is 'database' and 'price' is 450.

- **Example 6**
  - $\sigma_{subject = "database" \text{ and } price = "450" \text{ or } year > "2010"}(Books)$
- **Output** -

Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

- **Example 7**
  - $\sigma_{age > 17}$ (Student)
- **Output:**
  - This will fetch the tuples(rows) from table **Student**, for which **age** will be greater than **17**.
- **Example 8**
  - $\sigma_{age > 17 \text{ and } gender = 'Male'}$ (Student)
- **Output:**
  - This will return tuples(rows) from table **Student** with information of male students, of age more than 17.

Projection

- The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.
- This helps to extract the values of specified attributes to eliminates duplicate values. (pi) symbol is used to choose attributes from a relation.
- This operator helps you to keep specific columns from a relation and discards the other columns.

- **Example of Projection:**
- Consider the following table

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Here, the projection of CustomerName and status will give

$\Pi_{\text{CustomerName, Status}}$ (Customers)

CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

- **Example 1**
  - $\Pi_{subject,\ author}$ (Books)
- **Output:**
  - Selects and projects columns named as subject and author from the relation Books.
- **Example 2**
  - $\Pi_{Name,\ Age}$(Student)
- **Output:**
  - Above statement will show us only the **Name** and **Age** columns for all the rows of data in **Student** table.

Rename (ρ)

- Rename is a unary operation used for renaming attributes of a relation.
- ρ (a/b)R will rename the attribute 'b' of relation by 'a'.

- **Syntax:**
  - ρ(RelationNew, RelationOld)

Union

- 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 -

1. R and S must be the same number of attributes.
2. Attribute domains need to be compatible.
3. Duplicate tuples should be automatically removed.

Example

- Consider the following tables.

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

A ∪ B gives

column 1	column 2
1	1
1	2
1	3

- **Example 1:**
  - Π_{author} (Books) $\cup$ Π_{author} (Articles)
- **Output:**
  - Projects the names of the authors who have either written a book or an article or both.
- **Example 2**
  - Π_{Student}(RegularClass) $\cup$ Π_{Student}(ExtraClass)
- **Output:**
  - Above operation will give us name of **Students** who are attending either regular classes or extra classes or both, eliminating repetition.

Set Difference (-)

- **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.
- The attribute name of A has to match with the attribute name in B.
- The two-operand relations A and B should be either compatible or Union compatible.
- It should be defined relation consisting of the tuples that are in relation A, but not in B.

Table A - B	
column 1	column 2
1	2

- **Example 1**
  - Π_{author} (Books) – Π_{author} (Articles)
- **Output** –
  - Provides the name of authors who have written books but not articles.
- **Example 2**
  - $\Pi_{Student}$(RegularClass) - $\Pi_{Student}$(ExtraClass)
- **Output**
  - if we want to find name of students who attend the regular class but not the extra class, then, we can use the above operation.

Intersection

- An intersection is defined by the symbol ∩
- **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.

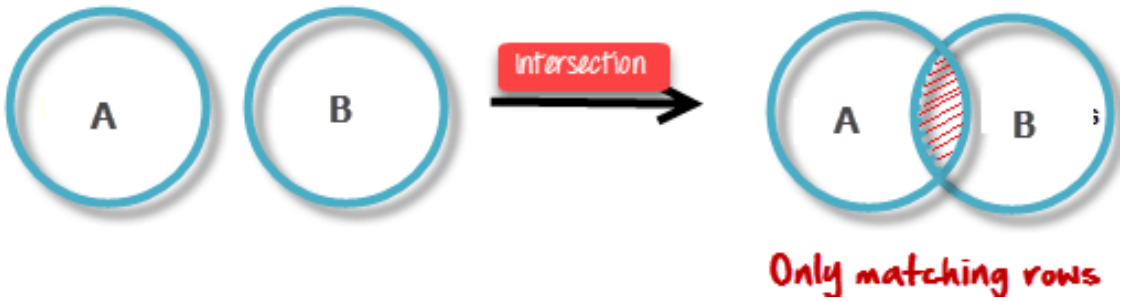


Table A ∩ B	
column 1	column 2
1	1

Cartesian Product

- This type of operation is helpful to merge columns from two relations. Generally, a Cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations.

σ column 2 = '1' (A X B)	
column 1	**column 2**
1	1
1	1

- **Example 1**
  - $\sigma_{time = 'morning'}$ (RegularClass X ExtraClass)
- **Output:**
  - if we want to find the information for Regular Class and Extra Class which are conducted during morning, then, we can use the following operation:
- **Example 2**
  - $\sigma_{author = 'ITM'}$(Books X Articles)
- **Output** :
  - Yields a relation, which shows all the books and articles written by ITM.

A:

a1	a2
1	2
3	4
5	6

B:

b1
7
8

A × B ⇨

a1	a2	b1
1	2	7
1	2	8
3	4	7
3	4	8
5	6	7
5	6	8

- Relations *r*, *s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$
- $r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- $\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

Join Operations

- A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.
- Example:
- **EMPLOYEE**

EMP_CODE	EMP_NAME
101	Stephan
102	Jack
103	Harry

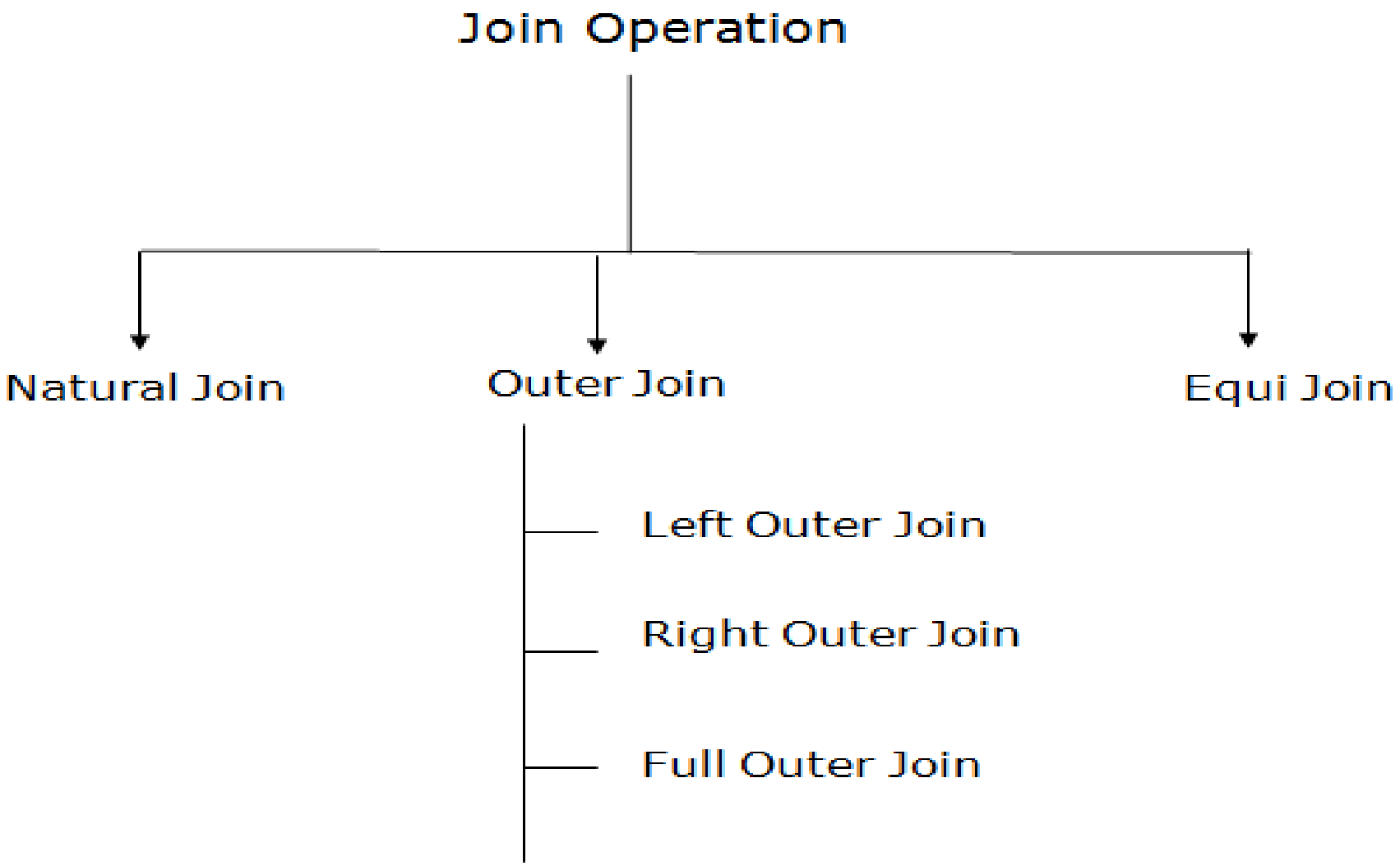
- **SALARY**

EMP_CODE	SALARY
101	50000
102	30000
103	25000

Operation: (EMPLOYEE ⋈ SALARY)

EMP_CODE	EMP_NAME	SALARY
101	Stephan	50000
102	Jack	30000
103	Harry	25000

Types of Join operations



1. Natural Join

- A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
- It is denoted by ⋈.
- **Example:** Let's use the above EMPLOYEE table and SALARY table:
- **Input:**
- ∏EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

2. Outer Join

- The outer join operation is an extension of the join operation. It is used to deal with missing information.
- **Example:**
- **EMPLOYEE**

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

- **FACT_WORKERS**

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

- (EMPLOYEE ⋈ FACT_WORKERS)

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

- An outer join is basically of three types:

1. Left outer join
2. Right outer join
3. Full outer join

1. Left outer join:

- Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In the left outer join, tuples in R have no matching tuples in S.
- It is denoted by ⟕.
- **Example:** Using the above EMPLOYEE table and FACT_WORKERS table

- EMPLOYEE ⋈ FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

2. Right outer join:

- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.
- It is denoted by ⟖.
- **Example:** Using the above EMPLOYEE table and FACT_WORKERS Relation
- **Input:**
- EMPLOYEE ⟖ FACT_WORKERS

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

3. Full outer join:

- Full outer join is like a left or right join except that it contains all rows from both tables.
- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
- It is denoted by ⟗.
- **Example:** Using the above EMPLOYEE table and FACT_WORKERS table
- **Input:**
- EMPLOYEE ⟗ FACT_WORKERS
-

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

3. Equi join

- It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

- **CUSTOMER**

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

- **PRODUCT**

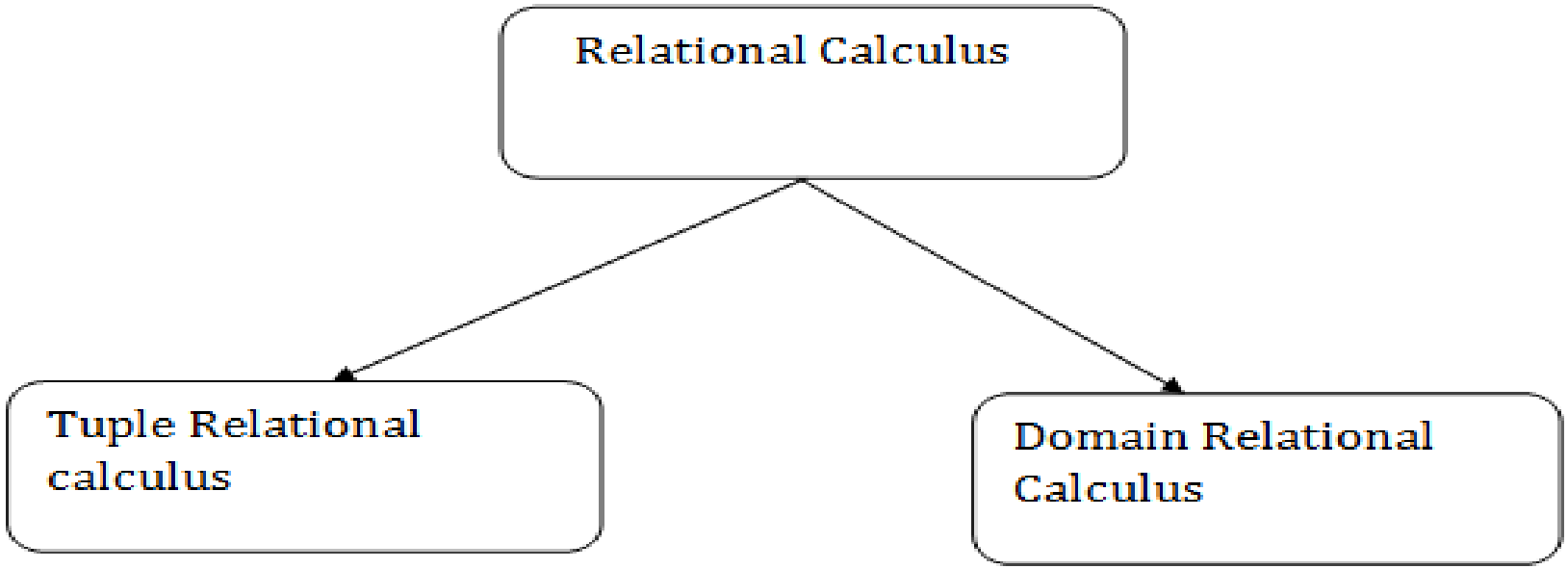
PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

- **Input:**
- CUSTOMER ⋈ PRODUCT
- **Output:**

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai
3	Harry	3	Noida

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.



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'.
- TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (∃) and Universal Quantifiers (∀).
- **For example:**
  - **{ R| ∃T ∈ Authors(T.article='database' AND R.name=T. name)}**
- **Output:** This query will yield the same result as the previous one.

Table: Student

```
First_Name      Last_Name      Age
----------      ---------      ----
Ajeet           Singh          30
Chaitanya       Singh          31
Rajeev          Bhatia         27
Carl            Pratap         28
```

Lets 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
```

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 > | ∈ ITM ∧ subject = 'database'}
- **Output:** This query will yield the article, page, and subject from the relational ITM, where the subject is a database.

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:

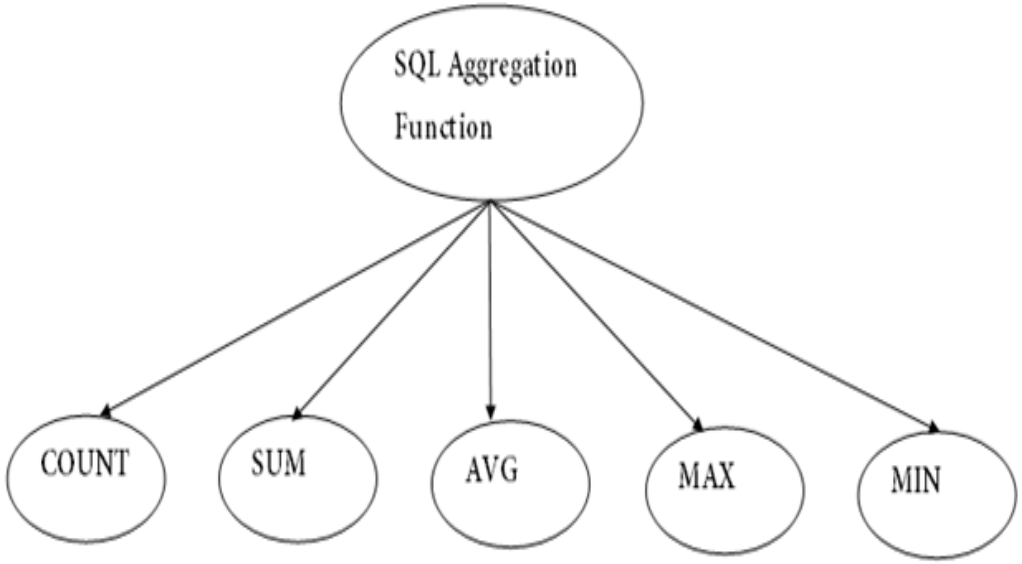
```
First_Name      Age
----------      ----
Ajeet           30
Chaitanya       31
Carl            28
```

TUPLE RELATIONAL CALCULUS (TRC)	DOMAIN RELATIONAL CALCULUS (DRC)
In TRS, the variables represent the tuples from specified relation.	In DRS, the variables represent the value drawn from specified domain.
A tuple is a single element of relation.In database term, it is a row.	A domain is equivalent to column data type and any constraints on value of data.
In this filtering variable uses tuple of relation.	In this filtering is done based on the domain of attributes.
Notation : {T | P (T)} or {T | Condition (T)}	Notation : { a1, a2, a3, ..., an | P (a1, a2, a3, ..., an)}
Example : {T | EMPLOYEE (T) AND T.DEPT_ID = 10}	Example : { | < EMPLOYEE > DEPT_ID = 10 }

Aggregate Operators

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.

Types of SQL Aggregation Function



1. COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax :

- **COUNT(*):** Counts all the number of rows of the table including null.
- **COUNT(COLUMN_NAME):** Count number of non-null values in column.
- **COUNT(DISTINCT COLUMN_NAME):** Count number of distinct values in a column.

Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example: COUNT()

```
SELECT COUNT(*)
FROM PRODUCT_MAST;
```

Output:

```
10
```

Example: COUNT with WHERE

```
SELECT COUNT(*)
FROM PRODUCT_MAST;
WHERE RATE>=20;
```

Output:

```
7
```

Example: COUNT() with DISTINCT

```
SELECT COUNT(DISTINCT COMPANY)
FROM PRODUCT_MAST;
```

Output:

```
3
```

Example: COUNT() with GROUP BY

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY;
```

Output:

```
Com1    5
Com2    3
Com3    2
```

Example: COUNT() with HAVING

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY
HAVING COUNT(*)>2;
```

Output:

```
Com1    5
Com2    3
```

2. SUM Function

- Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax : SUM()

Example: SUM()

```
SELECT SUM(COST)
FROM PRODUCT_MAST;
```

Output:

Example: SUM() with WHERE

```
SELECT SUM(COST)
FROM PRODUCT_MAST
WHERE QTY>3;
```

Output:

```
328
```

Example: SUM() with GROUP BY

```
SELECT SUM(COST)
FROM PRODUCT_MAST
WHERE QTY>3
GROUP BY COMPANY;
```

Output:

```
Com1    150
Com2    170
```

Example: SUM() with HAVING

```
SELECT COMPANY, SUM(COST)
FROM PRODUCT_MAST
GROUP BY COMPANY
HAVING SUM(COST)>=170;
```

Output:

```
Com1    335
Com3    170
```

3. AVG function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax : AVG()

Example:

```
SELECT AVG(COST)
FROM PRODUCT_MAST;
```

Output:

```
67.00
```

4. MAX Function

- MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax : MAX()

Example:

```
SELECT MAX(RATE)
FROM PRODUCT_MAST;
```

5. MIN Function

- MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax :MIN()

Example:

```
SELECT MIN(RATE)
FROM PRODUCT_MAST;
```

Output:

```
10
```


Thank
You