# UNIT II – Relational Model

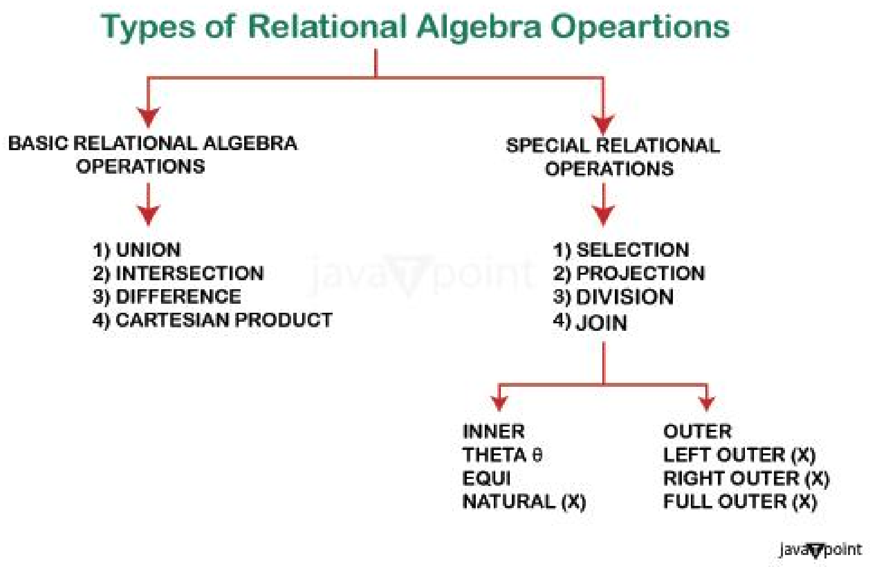
Relational Data Model - keys, referential integrity and foreign keys, Relational Algebra - SQL fundamentals- Introduction, data definition in SQL, table, key and foreign key definitions, update behaviors-Views, Triggers, Joins, Constraints, Stored Procedure-Intermediate SQL-Advanced SQL features -Embedded SQL- Dynamic SQL

## Overview

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.

## What is Relational Algebra?

- Relational algebra consists of a certain set of rules or operations that are widely used to manipulate and query data from a relational database. It can be facilitated by utilizing SQL language and helps users interact with database tables based on querying data from the database more efficiently and effectively.
- Relational Algebra incorporates a collection of operations, such as filtering data or combining data, that help us organize and manipulate data more efficiently.
- This" algebra " is the foundation for most database queries, and it enables us to extract the required information from the databases by using SQL query language.



**Fundamental Operators**

Relational algebra consists of various operators that help us fetch and manipulate data from relational tables in the database to perform certain operations on relational data.

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

- Selection(σ)
- Projection(π)
- Union(U)
- Set Difference (-)
- Set Intersection (∩)
- Rename(ρ)
- Cartesian Product(X)

**Select Operation:**

- The select operation also known as restriction operation results in a new relation that contains those rows of relation that satisfy a specified condition.
- The select operation selects tuples that satisfy a given predicate.
- It is denoted by sigma (σ).
- Notation: σ p(r)

Where:

σ is used for selection prediction

r is used for relation

p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

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 |

Input:

σ BRANCH_NAME="perryride" (LOAN)

Output:

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

Features of Select operation:

- It is a unary operation because works only a single relation.
- The resulting table has the same degree as that of the original table. This means that the number of columns in both the relations is same.
- The number of rows of the resulting relations is always less than or equal to the original relation.
- It operates on each row of the relation independently.

**2. Project Operation:**

- 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 ∏ and the attributes to be retrieved appear as subscripts separated by commas and relations name is given in parenthesis following the PI.
- Notation: ∏ A1, A2, An (r)

Where

A1, A2, A3 is used as an attribute name of relation r.

Example: CUSTOMER RELATION

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

Input:

1. ∏ NAME, CITY (CUSTOMER)

Output:

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

**Features of Project operation:**

- It is a unary operation i.e. it can operate only a single relation.
- The degree of the resulting relation is equal to the number of attributes specified in the attribute list
- If the attribute list contains a primary key attribute then the number of tuples in the resulting relation is equal to the number of tuples in the original relation.
- Non-commutative: It does not hold the commutative property.
- Duplicate Elimination: This operation removes the duplicate rows from the table which results in a valid relation knows as duplicate elimination.

**3. Union Operation:**

- The Union operation of two relations results in a new relation containing rows from both relations with duplicates removed.
- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by ∪.
- Notation: R ∪ S

A union operation must hold the following condition:

- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.

Features of Union operation:

- Input relations must be union compatible.
- Commutativity: This means that the result of (R ∪ S) is same as that of (S ∪ R).
- Associativity: This mean that R ∪ (S ∪ O) = (R ∪ S) ∪ O where R, S and O are relations.

Example:

DEPOSITOR RELATION

| CUSTOMER_NAME | ACCOUNT_NO |
|---|---|
| Johnson | A-101 |
| Smith | A-121 |
| Mayes | A-321 |
| Turner | A-176 |
| Johnson | A-273 |
| Jones | A-472 |
| Lindsay | A-284 |

BORROW RELATION

| CUSTOMER_NAME | LOAN_NO |
|---|---|
| Jones | L-17 |
| Smith | L-23 |
| Hayes | L-15 |
| Jackson | L-14 |
| Curry | L-93 |
| Smith | L-11 |
| Williams | L-17 |

Input:

∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

| CUSTOMER_NAME |
|---|
| Johnson |
| Smith |
| Hayes |

| |
|---|
| Turner |
| Jones |
| Lindsay |
| Jackson |
| Curry |
| Williams |
| Mayes |

**4. Intersection:**

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
- It is denoted by intersection ∩.
- Notation: R ∩ S

**Features of Intersection operation:**

- Input relations must be union compatible.
- **Commutativity:** This means that result of (R ∩ S) is same as that of (S ∩ R).
- **Associativity:** This mean that R ∩ (S ∩ O) = (R ∩ S) ∩ O where R, S and O are relations.

**Example:** Using the above DEPOSITOR table and BORROW table

**Input:**

1. ∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)

**Output:**

| **CUSTOMER_NAME** |
|---|
| Smith |
| Jones |

**5. Difference:**

- The difference of two relations results in a new relation that contains tuples that occur in the first relation but not in the second relation.
- Suppose there are two tuples R and S. The set difference 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:**

1. ∏ CUSTOMER_NAME (BORROW) - ∏ CUSTOMER_NAME (DEPOSITOR)

**Output:**

| **CUSTOMER_NAME** |
|---|
| Jackson |
| Hayes |
| Willians |
| Curry |

**Features of Set Difference operation:**

- Input relations must be union compatible.
- They are not commutative. This means that the result of R - Sis not the same as the result S - P.
- They are not associative. This means that result of Q - (R - S) is not the same as the result of (Q - S) - R where Q, S and R are relations.
- Intersection can be expressed as difference (- ) operations: (R ∩ S) = R - (R - S)

But writing the equation with a single intersection operation is more convenient than involving a pair of difference operations. Here R and S unions are favorable.

6. Cartesian product

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by X.
- It is a binary relation which means that it always operates on two relations.
- Notation: E X D

Example:

**EMPLOYEE**

| **EMP_ID** | **EMP_NAME** | **EMP_DEPT** |
|---|---|---|
| 1 | Smith | A |
| 2 | Harry | C |

| | | |
|---|---|---|
| 3 | John | B |

**DEPARTMENT**

| **DEPT_NO** | **DEPT_NAME** |
|---|---|
| A | Marketing |
| B | Sales |
| C | Legal |

**Input:**

1. EMPLOYEE X DEPARTMENT

**Output:**

| **EMP_ID** | **EMP_NAME** | **EMP_DEPT** | **DEPT_NO** | **DEPT_NAME** |
|---|---|---|---|---|
| 1 | Smith | A | A | Marketing |
| 1 | Smith | A | B | Sales |
| 1 | Smith | A | C | Legal |
| 2 | Harry | C | A | Marketing |
| 2 | Harry | C | B | Sales |
| 2 | Harry | C | C | Legal |
| 3 | John | B | A | Marketing |
| 3 | John | B | B | Sales |
| 3 | John | B | C | Legal |

Properties of Cartesian product operation:

- The relations to which Cartesian product operation is applied need not necessary to be union compatible.
- The total number of rows in the result operation is equal to the product of the number of rows in the first and second relations, i.e.

**Total number of tuples of data = Total number of tuples of E + Total number of tuples of D**

- The resulting relation may have duplicate properties if some properties of the two relations are defined on common domains.
- The resulting degree of action is equal to the sum of the degrees of all relations

**Degree of E = Degree of E + Degree of D**

- **Commutativity:** This means that result of E X D is same as that of D X
- **Associativity:** This mean that E X (D X F) = (E X D) X F where E, D and F are relations.

**7. Rename Operation:**

The rename operation is used to rename the output relation. It is denoted by **rho** (ρ). If no rename operation is applied then the names of the attributes in a resulting relation are the same as those in the original relation and in the same order.

**Example:** We can use the rename operator to rename STUDENT relation to STUDENT1.

1. ρ(STUDENT1, STUDENT)

*Note: Apart from these common operations Relational algebra can be used in Join operations.*

**8. Join Operation:**

A join operation combines two or more relations to form a new relation such that new relation contains only those tuples from different relations that satisfy the specified criteria. It forms a new relation which contains all the attributes from both the joined relations whose tuples are those defined by the restrictions applies i.e. the join condition applied on two participating relations. The join is performed on two relations, who have one or more attributes in common. These attributes must be domain compatible i.e. they have same data type. It is a binary operation. The Join operation is denoted by a Join symbol. The general from of representing a join operation on two relations P and Q is

1. P⋈ <join_condion> Q

When we use equality operator in the join condition then such a join is called EQUI Join. It is mostly commonly used Join.

**EMP Table**

| **EMP_ID** | **ENAME** | **SALARY** | **Dept_ID** |
|---|---|---|---|
| 101 | Raj | 450000 | 1 |
| 102 | Lavi | 300000 | 2 |
| 103 | Chandan | 480000 | 3 |
| 104 | Ravi | 340000 | 4 |
| 105 | Abhi | 370000 | 5 |

**DEPT Relation**

| Dept_No | Dname |
|---|---|
| 1 | Marketing |
| 2 | Purchase |
| 3 | Finance |
| 4 | Packing |
| 5 | Marketing |

In the above both table, suppose we want to know the employee information with department name in which each employee is working. Now the employee information is in the Emp relation and Department name information is in dept relation. So to retrieve the columns from both the tables at same time, we need to join the EMP and DEPT relations. The relations can be joined over the column Dept_ID that exist in EMP relation and the Dept_no that exist in the DEPT relation and are domain compatible. Thus the result of EQUI Join where the condition is that Dept_Id attributes values the EMP relation should be equal to the Dept_No attribute values in the DEPT relation, **the result is shown below.**

| EMP_ID | ENAME | SALARY | Dept_ID | Dept_No | Dname |
|---|---|---|---|---|---|
| 101 | Raj | 450000 | 1 | 1 | Marketing |
| 102 | Lavi | 300000 | 2 | 2 | Purchase |
| 103 | Chandan | 480000 | 3 | 3 | Finance |
| 104 | Ravi | 340000 | 4 | 4 | Packing |
| 105 | Abhi | 370000 | 5 | 5 | Marketing |

Another is known as natural join in which there is no need to explicitly name the columns. A join is performed by joining all columns from the first relation of any column to another relation with the same name. The result of natural join is as follows:

| EMP_ID | ENAME | SALARY | Dept_ID | Dname |
|---|---|---|---|---|
| **101** | **Raj** | **450000** | **1** | **Marketing** |
| **102** | **Lavi** | **300000** | **2** | **Purchase** |
| **103** | **Chandan** | **480000** | **3** | **Finance** |
| **104** | **Ravi** | **340000** | **4** | **Packing** |

| **105** | **Abhi** | **370000** | **5** | **Marketing** |
|---|---|---|---|---|

The natural join may also been referred to as INNER JOIN. Another type of JOIN in which a relation is joined to itself by comparing values with a column of the relation is called a self-join.

**Example of SELF Join:**

| **EMP_ID** | **ENAME** | **Mang_Id** |
|---|---|---|
| 101 | Raj | - |
| 102 | Lavi | 101 |
| 103 | Chandan | 104 |
| 104 | Ravi | - |
| 105 | Abhi | - |

**Resultant Relation**

| **ENAME** | **Mang_Id** |
|---|---|
| Lavi | 101 |
| Chandan | 104 |

In the EMP relation, the attribute EMP_ID shows employee'code, ENAME and Mang_Id under which employee is working. In this, some employee are not having Mang_Id i.e. their value is null because they act as a manager itself.

**Features of Join Operation:**

- Like the cartersion product, join operations are commutative. Using this property, we can choose which relation can be the inner and which one the outer while joining two relations. If P and Q are two relations then, P⋈ Q = Q ⋈ P
- Rows whose join attribute is null do not appear in the resulting relation.
- We can also join more than two relations by increasing the complexity.
- Joins are generally used when a relationship exists between relations such as where the join condition is based on the primary key and foreign key columns.
- Join is a very powerful operator and together with projection form a base for normalization i.e. theoretical support for designing database relations.
- If the attributes on which the join is performed have the same name in both the relations then renaming is necessary for the EQUIJOIN operation and

unnecessary for the NATURAL JOIN operation because the former i.e. EQUI JOIN both exist as a result of the common attribute relation but the latter In i.e., natural additive consequent relations have only one common property.

**Division Operation:**

The division operation results in a new relation such that every tuple appearing in the resulting relation must exist in the dividend relation. In the combination of each tuple in the denominator relation. It is a binary operation that operates on two relations. The division is represented by a symbol "÷".

**Example: Retrieve the Subject Name that is taught in all courses.**

**Subject Relation**

| **Subject_Name** | **Cousres** |
|---|---|
| Computer Graphics | BCA |
| Computer Graphics | BSC GWD |
| DBMS | Btech CS |
| DBMS | Btech IT |
| Theory of Computaion | MCA |
| DBMS | BCA |

**Course Relation**

| **Courses Name** |
|---|
| BCA |
| Btech CS |
| Btech IT |
| BSC GWD |
| MCA |
| BCA |

**Subject Relation "÷" Course Relation**

The resultant relation is: Result Relation

| **Subject_Name** |
|---|
| DBMS |

**Properties of Division Relation:**

- It is a binary operation as it operators on two relations
- The division operation is suited to queries that include the phrase "for all".
- It is very rarely used in database applications.