Database Management System

Relational Algebra Operations

Bca 3rd semester
Ashmita Mahanty
Assistant professor
Departement of IT

Relational Algebra operations

In **procedural languages,** the user would specify what has to be done and how it can be done, i.e. the step by step procedure of it. Therefore, a program written using procedural language works with the state of machine. However, the size of the program would be large. But, the overall efficiency of a procedural language program is high. The common examples procedural programming languages are BASIC, FORTRAN, COBOL, C, Pascal, etc.

Non-procedural languages are fact-oriented programing languages. The programs written in non-procedural languages specify what is to be done and do not state exactly how a result is to be evaluated. In the non-procedural programming language, the user would specify what has to be done but doesn't get into the how it has to be done part

Relational Algebra: It is a procedural query language**,** which takes the instance of a relation as an input and yields the relation as an output. Queries are performed by the operator, these operator are either **Unary or binary.** There are following fundamental operators that are used in relational algebra:-

1. Select
2. Project
3. Union
4. Set difference
5. Cartesian product
6. Joins

1. Select Operation: 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.

Consider the Tutorial table:-

Roll No.	**Topic**	**Author**	**Cost**
1	Database	Navate	500
2	OS	Puneet Kaur	1000
3	Graphic	Saurbh Jain	1205
4	TOC	Naveen	1234
5	DS	N.K.Sharma	650

a. Select the topic whose name is database.

$\sigma_{\text{topic=”Databse “}}$ (Tutorial)

b. Select the book whose price is greater than 1000.

$\sigma_{\text{cost}>1000}$(Tutorial)

2. Project Operation: The select operation selects some of the rows from the table while discarding other rows, the PROJECT operation on the other hand, select certain columns from the table and discard the other columns. If we are interested in certain attribute of relation, we use the RPOJECT operation to project the relation over these attribute only.

Consider the following table:-

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

Π CustomerName, Status (Customers)

3. **UNION:** The result of this operation, denoted by R U S, is a relation that includes all the tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

Table R

Name	Address
Ram	U.P.
Shaym	Haryana
Mohan	Delhi

Table S:

Name	Mobile No.
Ram	9999999393
Shyam	9999999563
S.K.	8899999393
Shruti	9999779393

∏ **NAME (R)** ∪ ∏ **NAME (S):**

Name
Ram
Shaym
Mohan
S.K.
Shruti

4. **Set Difference (or MINUS):-** The result of this operation, denoted by R-S, is relation that includes all the tuples that are in R but not in S.

∏ NAME (R) - ∏ NAME (S)

Name
Mohan

5. **Cartesian product**: It is also known as cross product or cross join which is denoted by X. It is used to combine each row of one table with the each row of other table.

Employee:

Name	Salary
Ram	10000
Shaym	20000
Mohan	5000

Department:-

Dprt_No.	Dprt_Name
1	H.R.
2	Management
3	IT

Employee X Department

Name	Salary	Dprt_No.	Dprt_Name
Ram	10000	1	H.R.
Ram	10000	2	Management
Ram	10000	3	IT
Shaym	20000	1	H.R.
Shaym	20000	2	Management
Shaym	20000	3	IT
Mohan	5000	1	H.R.
Mohan	5000	2	Management
Mohan	5000	3	IT

6. RENAME

The RENAME operation is used to rename the output of a relation.

Sometimes it is simple and suitable to break a complicated sequence of operations and rename it as a relation with different names. Reasons to rename a relation can be many, like –

- We may want to save the result of a relational algebra expression as a relation so that we can use it later.
- We may want to join a relation with itself, in that case, it becomes too confusing to specify which one of the tables we are talking about, in that case, we rename one of the tables and perform join operations on them.

Notation:
ρ X (R)

Example-1: Query to rename the relation Student as Male Student and the attributes of Student – RollNo, SName as (Sno, Name).

Sno	Name
2600	Ronny
2655	Raja

ρ MaleStudent(Sno, Name) πRollNo, SName(σCondition(Student))

Example-2: Query to rename the attributes Name, Age of table Department to A,B.

ρ (A, B) (Department)

Example-3: Query to rename the table name Project to Pro and its attributes to P, Q, R.

ρ Pro(P, Q, R) (Project)

Extended Operators in Relational Algebra

Join
Intersection
Divide

Table R

Name	Address
Ram	U.P.
Shaym	Haryana
Mohan	Delhi

Table S:

Name	**Mobile No.**
Ram	9999999393
Shyam	9999999563
S.K.	8899999393
Shruti	9999779393

Intersection: The result of this operation, denoted by R∩S is a relation that includes all the tuples that are common in both R and S.

∏ NAME (R) ∩ ∏ NAME (S)

Name
Ram
Shaym

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 ⋈.

1. INNER JOIN

An **INNER JOIN** returns only the rows that have matching values in both tables. If there is no match, the row is not included in the result.

Syntax:

```
SELECT column_names
FROM table1
INNER JOIN table2 ON table1.common_column = table2.common_column;
```

Example:

Assume we have two tables:

Employees Table

emp_id	**name**	**dept_id**
1	Alice	101
2	Bob	102
3	Charlie	103

Departments Table

dept_id	**department_name**
101	HR
102	IT
104	Finance

Query using INNER JOIN:

```
SELECT employees.name, departments.department_name
FROM employees
INNER JOIN departments ON employees.dept_id = departments.dept_id;
```

Result:

name	department_name
Alice	HR
Bob	IT

Explanation:

- Only rows where `dept_id` matches in both tables are returned.
- The row with `dept_id = 103` from Employees and `dept_id = 104` from Departments are excluded.

Types of INNER JOIN:

1. **Equi JOIN:** Uses = operator in the condition.

```
SELECT * FROM employees INNER JOIN departments ON employees.dept_id = departments.dept_id;
```

2. **Natural JOIN:** Automatically joins tables with the same column name (not explicitly used in SQL Server).

```
SELECT * FROM employees NATURAL JOIN departments;
```

2. OUTER JOIN

An **OUTER JOIN** returns all records from one table and the matched records from another table. If there is no match, NULL values are returned.

Types of OUTER JOIN:

1. **LEFT JOIN (LEFT OUTER JOIN)**
   - Returns all rows from the **left** table and matching rows from the right table.
   - If there's no match, NULL is returned from the right table.

   Syntax:

   ```
   SELECT column_names
   FROM table1
   LEFT JOIN table2 ON table1.common_column = table2.common_column;
   ```

   Example:

   ```
   SELECT employees.name, departments.department_name
   FROM employees
   LEFT JOIN departments ON employees.dept_id = departments.dept_id;
   ```

   Result:

name	department_name
Alice	HR
Bob	IT
Charlie	NULL

Explanation:

- All employees are shown, even if their `dept_id` has no match in the Departments table.
- `Charlie` has no matching department, so `NULL` appears.

2. **RIGHT JOIN (RIGHT OUTER JOIN)**
   - Returns all rows from the **right** table and matching rows from the left table.
   - If there's no match, NULL is returned from the left table.

Syntax:

```
SELECT column_names
FROM table1
RIGHT JOIN table2 ON table1.common_column = table2.common_column;
```

Example:

```
SELECT employees.name, departments.department_name
FROM employees
RIGHT JOIN departments ON employees.dept_id = departments.dept_id;
```

Result:

name	department_name
Alice	HR
Bob	IT

NULL Finance

Explanation:

- All departments are displayed, even if no employees belong to them.
- The "Finance" department has no employees, so `NULL` appears in the `name` column.

3. **FULL JOIN (FULL OUTER JOIN)**
   - Returns all records when there is a match in either table.
   - If there is no match, NULL is returned for missing values.

Syntax:

```
SELECT column_names
FROM table1
FULL JOIN table2 ON table1.common_column = table2.common_column;
```

Example:

```
SELECT employees.name, departments.department_name
```

```
FROM employees
FULL JOIN departments ON employees.dept_id = departments.dept_id;
```

Result:

name	department_name
Alice	HR
Bob	IT
Charlie	NULL
NULL	Finance

Explanation:

- All employees and all departments are displayed.
- If there is no match, `NULL` appears.

Divide Operation in DBMS (Relational Algebra)

The **DIVISION (÷) operation** is a special type of operation in **Relational Algebra** that is used to retrieve records that are related to **all** entries of another set. It is typically used when dealing with "all" conditions in a query.

1. Definition

- The **division operation (÷)** is used when a relation contains a set of values that need to be matched with all values in another relation.
- It is used in scenarios where we need to find entities that have a relationship with **all possible values** of another set.

2. Syntax

If we have two relations **A**(X, Y) and **B**(Y), the result of `A ÷ B` will return a set of values from X that are related to **all** values in B.

```
A(X, Y) ÷ B(Y) = C(X)
```

- **A(X, Y)**: A table containing attributes **X** and **Y**.
- **B(Y)**: A table containing only **Y** values.
- **C(X)**: The result contains only **X** values that are associated with **all** Y values in **B**.

3. Example of DIVISION Operation

Given Two Relations:

Student_Course Table (A - X, Y)

Student	Course
Alice	DBMS
Alice	OS
Bob	DBMS
Bob	OS

Charlie	DBMS

Course Table (B - Y)

Course
DBMS
OS

Query:

Find students who have taken **all** courses listed in the **Course Table (B)**.

Result:

Student
Alice
Bob

Explanation:

- **Alice** has taken **DBMS** and **OS** → Included
- **Bob** has taken **DBMS** and **OS** → Included
- **Charlie** has taken only **DBMS** → Not included (missing OS)

So, **Alice** and **Bob** are the only students who have taken **all** courses in the Course Table.