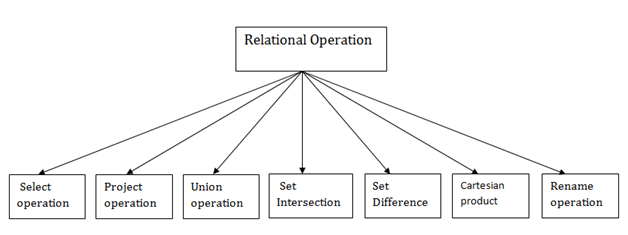
Relational Algebra

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.

Types of Relational operation



1. Select Operation:

* The select operation selects tuples that satisfy a given predicate.
* It is denoted by sigma (σ).

1. 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:**

1. σ BRANCH\_NAME="perryride" (LOAN)

**Output:**

|  |  |  |
| --- | --- | --- |
| **BRANCH\_NAME** | **LOAN\_NO** | **AMOUNT** |
| Perryride | L-15 | 1500 |
| Perryride | L-16 | 1300 |

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 ∏.

1. 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 |

3. Union Operation:

* 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 ∪.

1. 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.

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:**

1. ∏ CUSTOMER\_NAME (BORROW) ∪ ∏ CUSTOMER\_NAME (DEPOSITOR)

**Output:**

|  |
| --- |
| **CUSTOMER\_NAME** |
| Johnson |
| Smith |
| Hayes |
| Turner |
| Jones |
| Lindsay |
| Jackson |
| Curry |
| Williams |
| Mayes |

4. Set 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 ∩.

1. Notation: R ∩ S

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

**Input:**

1. ∏ CUSTOMER\_NAME (BORROW) ∩ ∏ CUSTOMER\_NAME (DEPOSITOR)

**Output:**

|  |
| --- |
| **CUSTOMER\_NAME** |
| Smith |
| Jones |

5. 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 (-).

1. 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 |

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.

1. 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 |

7. Rename Operation:

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.

1. ρ(STUDENT1, STUDENT)

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**

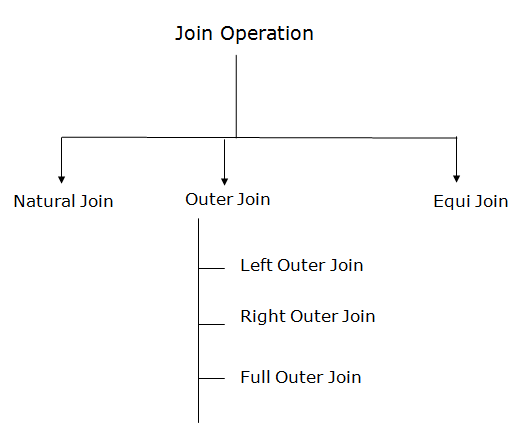
|  |  |
| --- | --- |
| **EMP\_CODE** | **SALARY** |
| 101 | 50000 |
| 102 | 30000 |
| 103 | 25000 |

1. Operation: (EMPLOYEE ⋈ SALARY)

**Result:**

|  |  |  |
| --- | --- | --- |
| **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:**

1. ∏EMP\_NAME, SALARY (EMPLOYEE ⋈ SALARY)

**Output:**

|  |  |
| --- | --- |
| **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 |

**Input:**

1. (EMPLOYEE ⋈ FACT\_WORKERS)

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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

a. 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

**Input:**

1. 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 |

b. 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:**

1. EMPLOYEE ⟖ FACT\_WORKERS

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

c. 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:**

1. EMPLOYEE ⟗ FACT\_WORKERS

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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(=).

**Example:**

**CUSTOMER RELATION**

|  |  |
| --- | --- |
| **CLASS\_ID** | **NAME** |
| 1 | John |
| 2 | Harry |
| 3 | Jackson |

**PRODUCT**

|  |  |
| --- | --- |
| **PRODUCT\_ID** | **CITY** |
| 1 | Delhi |
| 2 | Mumbai |
| 3 | Noida |

**Input:**

1. CUSTOMER ⋈ PRODUCT

**Output:**

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASS\_ID** | **NAME** | **PRODUCT\_ID** | **CITY** |
| 1 | John | 1 | Delhi |
| 2 | Harry | 2 | Mumbai |
| 3 | Harry | 3 | Noida |