**Relational Algebra**

A **query language** is a language in which user requests information from the database. it can be categorized as either **procedural** or **nonprocedural.** In a procedural language the user instructs the system to do a sequence of operations on database to compute the desired result. In nonprocedural language the user describes the desired information without giving a specific procedure for obtaining that information.  
  
The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produces a new relation as output.

**Fundamental Operations**

* SELECT
* PROJECT
* UNION
* SET DIFFERENCE
* CARTESIAN PRODUCT
* RENAME

Select and project operations are unary operation as they operate on a single relation.Union, set difference, Cartesian product and rename operations are binary operations as they operate on pairs of relations.

**Other Operations**

* SET INTERSECTION
* NATURAL JOIN
* DIVISION
* ASSIGNMENT

**The select operation**: - to identify a set of tuples which is a part of a relation and to extract only these tuples out. The select operation selects tuples that satisfy a given predicate or condition.

* It is a unary operation defined on a single relation.
* It is denoted as **σ**.

Consider the following table "Book" :-

Code:

+--------+--------+------------------+

| Acc-no | Yr-pub | title |

+--------+--------+------------------+

| 734216 | 1982 | Algorithm design |

| 237235 | 1995 | Database systems |

| 631523 | 1992 | Compiler design |

| 543211 | 1991 | Programming |

| 376112 | 1992 | Machine design |

+--------+--------+------------------+

Example1:- Select from the relation “Book” all the books whose year of publication is 1992.

Code:

**σ Yr-pub=1992(Book)**

Example2:- Select from the relation “Book” all the books whose Acc-no is greater than equal to 56782.

Code:

**σ Acc-no>=56782(Book)**

**The project operation**: - returns its argument relation with certain attributes left out.

* It is a unary operation defined on a single relation
* It is denoted as **Π**.

Example:- List all the Title and Acc-no of the “Book” relation.

Code:

**Π Acc-no, Title (Book)**

**The union operation:** - is used when we need some attributes that appear in either or both of the two relations.

* It is denoted as **U**.

Example:  
Borrower (customer-name, loan-number)  
Depositor (customer-name, account-number)  
Customer (customer-name, street-number, customer-city)  
  
  
List all the customers who have either an account or a loan or both

Code:

**Π customer-name (Borrower) U Π customer-name (Depositor)**

For a union operation **r U s** to be valid, two conditions must hold:

* The relation r and s must be of the same arity, i.e. they must have the same number of attributes.
* The domains of the ith attribute of r and the ith attribute of s must be the same for all i.

**The set difference operation: -**finds tuples in one relation but not in other.

* It is denoted as **–**

Example:  
Find the names of all customers who have an account but not a loan.

Code:

**Π customer-name (Depositor) - Π customer-name (Borrower)**

**The Cartesian product operation: -**allows combining information from two relations.

* It is denoted as **r X s**where r and s are relations.

Consider the following relation or table "r" :-

Code:

+--------+-------+

| **A** | **B** |

+--------+-------+

| a | 1 |

| b | 2 |

| a | 2 |

+--------+-------+

Consider another relation or table "s" :-

Code:

+--------+-------+

| **B** | **C** |

+--------+-------+

| 3 | 1a |

| 2 | 2b |

+--------+-------+

Therefore, rXs gives:-

Code:

+-----------+---------+---------+----------+

|  **r.A** |**r.B** | **s.B** | **s.C**  |

+-----------+---------+---------+----------+

| a | 1 | 3 | 1a |

| a | 1 | 2 | 2b |

| b | 2 | 3 | 1a |

| b | 2 | 2 | 2b |

| a | 2 | 3 | 1a |

| a | 2 | 2 | 2b |

+-----------+---------+---------+----------+

If relation r has n1 tuples and relation s has n2 tuples then **r** **X s**has n1\*n2 tuples.  
  
Example:  
Borrower (customer-name, loan-number)  
Loan (loan-number, branch-name, city, amount)  
  
List the names of all customers who have a loan in “Perryridge” branch

Code:

**Π customer-name (σ Borrower.loan-number=Loan.loan-number (σ branch-name=”Perryridge” (Borrower X Loan)))**

**The rename operation: -**used to rename.

* It is denoted as **ρ**.

E **:** relational algebra expression  
**ρ x (E):**returns the result of expression E under the name x.  
**ρ x (A1, A2, A3… An) (E):**returns the result of expression E under the name x with attributes renamed to A1, A2, A3… An.  
  
  
  
 **The set intersection operation: -**finds tuples in both the relations.

* It is denoted as **∩**.

Example:  
Borrower (customer-name, loan-number)  
Depositor (customer-name, account-number)  
Customer (customer-name, street-number, customer-city)  
  
List all the customers who have both a loan and an account.

Code:

**Π customer-name (Borrower) ∩ Π customer-name (Depositor)**

**The natural join operation: -**it is a binary operation and a combination of certain selections and a Cartesian product into one operation.

* It is denoted as |X| .
* It is associative.

It forms a Cartesian product of its two arguments.  
Then performs a selection forcing equality on those attributes those appear in both the relations.  
And finally removes duplicates attributes.  
  
r(R): r is a relation with attributes R.  
s(S): s is a relation with attributes S.  
  
If R **∩**S = Ф i.e. they have no attributes in common then **r |X| s = r X s**Example:-  
Table "r":-

Code:

+--------+--------+-------+

| **A** | **B**  | **C**  |

+--------+--------+-------+

| a | b | c |

| d | e | f |

| g | h | i |

+--------+--------+-------+

Table "s" :-

Code:

+--------+-------+

| **B** | **D** |

+--------+-------+

| b | g |

| p | r |

| e | t |

+--------+-------+

Therefore, r |X| s :-

Code:

+-----------+---------+---------+----------+

| **A**  | **B** | **C** | **D**  |

+-----------+---------+---------+----------+

| a | b | c | g |

| d | e | f | t |

+-----------+---------+---------+----------+

i.e;  
if r (A, B, C), s (B, D) then

Code:

**r|X|s** = **Π r.A, r.B, r.C, s.D (σ r.B = s.B (r X s))**

**The division / quotient operation: -**

* It is denoted as ÷.

Letr(R) and s(S) be relations  
  
**r ÷ s: -** the result consists of the restrictions of tuples in r to the attribute names unique to R, i.e. in the Header of r but not in the Header of s, for which it holds that all their combinations with tuples in s are present in r.  
  
Example:  
  
  
Relation or table "r":-

Code:

+--------+-------+

| **A** | **B** |

+--------+-------+

| a | 1 |

| b | 2 |

| a | 2 |

| p | 3 |

| p | 4 |

+--------+-------+

Relation or table "s":-

Code:

+------+

| B |

+------+

| 2 |

| 3 |

+------+

Therefore, r ÷ s

Code:

+------+

| A |

+------+

| b |

| a |

| p |

+------+

**Extended Relational Algebra Operations**

**GENERALIZED PROJECTION: -** It extends the projection operation by allowing arithmetic functions to be used in projection list.  
  
**Π F1,F2 … Fn (E)**  
  
Where E: relational algebra expression  
Fi: arithmetic expression  
  
Example:  
  
Table "Credit-info" :-

Code:

+----------------------+-----------+-----------------+

| Customer-name | Limit | Credit\_Balance |

+----------------------+-----------+-----------------+

| abc | 2000 | 500 |

| xyz | 500 | 250 |

| pqr | 700 | 100 |

| mno | 1500 | 1000 |

+----------------------+-----------+-----------------+

Find how much money a person can spend.

Code:

**Π Customer-name,(Limit-Credit\_balance)(Credit-info)**

**AGGREGATE FUNCTION:-**It takes a collection of values and returns a single value as a result.  
  
  
Table "Record" :-

Code:

+----------------------+-----------+-----------------+

| Student | Marks | Address |

+----------------------+-----------+-----------------+

| abc | 20 | Garia |

| xyz | 50 | Behala |

| pqr | 70 | Hindmotor |

| mno | 15 | Garia |

+----------------------+-----------+-----------------+

So the Aggregate Functions are:-

Code:

**G sum (Marks) (Record) =** returns sum total of the Marks attribute of Record.

**G average (Marks) (Record) =** returns average of the Marks attribute of Record.

**G min (Marks) (Record) =** returns the minimum of the Marks attribute of Record.

**G max (Marks) (Record) =** returns the maximum of the Marks attribute of Record.

**G count distinct (Address) (Record) =** returns the number of distinct values of Address attribute of Record.

Let's assume that we have a table named Account with three columns, namely Account\_Number, Branch\_Name and Balance.   
  
We wish to find the maximum balance of each branch.

Code:

**Branch\_Name G Max(Balance)(Account).**

To find the highest balance of all accounts regardless of branch

Code:

**G Max(Balance)(Account)**.

**Limitations Of Relational Algebra**

Although relational algebra seems powerful enough for most practical purposes, there are some simple and natural operators on relations which cannot be expressed by relational algebra. The transitive closure of a binary relation is one of them.

**Relational Algebra Implemented In SQL**

SQL (Structured query Language) is the most popular computer language used to create, modify, retrieve data from relational database management system.The basic structure of an SQL expression consists of three clauses:  
  
**SELECT: -**This clause corresponds to the projection operation of the relational algebra. It is used to list the attributes of the result of a query.  
  
**FROM:** -It corresponds to the Cartesian product operation of the relational algebra. It lists the relations scanned in the evaluation of an expression.  
  
**WHERE**: - This clause corresponds to selection predicate of relational algebra. It consists of a predicate involving attributes of the relations that appear in the FROM clause.  
  
SQL QUERY FORM:  
  
Select A1, A2….An  
From r1, r2…rm  
Where P  
  
Ai : attribute  
Ri : relation  
P : predicate  
  
SELECT clause- specifies the table columns retrieved.  
FROM clause- specifies the tables to be accessed.  
WHERE clause- which rows in the FROM tables to use.  
  
Example:  
  
Table "emp"

Code:

+----------------------+-----------+-----------------+

| Emp-name | E-salary | Emp-Address |

+----------------------+-----------+-----------------+

| abc | 2000 | Garia |

| xyz | 5000 | Behala |

| pqr | 7000 | Hindmotor |

| mno | 1500 | Garia |

+----------------------+-----------+-----------------+

List the names of all employees.

Code:

**Select Emp-name from Emp**

List all the rows of the table.

Code:

**Select \* from Emp**

List all the employees who receive salary greater than 5000.

Code:

**Select Emp-name from Emp where Emp-salary>5000**

List all the employees who live in Garia.

Code:

**Select Emp-name from Emp where E-address=’Garia’**

List all the employees in increasing order of salary.

Code:

**Select Emp-name from Emp Order by Emp-salary**

**Joining Tables**  
  
The FROM clause allows more than 1 table in its list. The rows from one table must be correlated with the rows of the others. This correlation is known as joining.  
  
Table "E1" :-

Code:

+----------------------+-----------------+

| Emp-name | Emp-Address |

+----------------------+-----------------+

| abc |Garia |

| xyz | Behala |

| pqr | Hindmotor |

| mno | Garia |

+----------------------+-----------------+

Table "E2" :-

Code:

+----------------------+-----------+

| Emp-name | E-salary |

+----------------------+-----------+

| abc | 2000 |

| xyz | 5000 |

| pqr | 7000 |

| mno | 1500 |

+----------------------+-----------+

Therefore,

Code:

**Select e1.Emp-Name,E-salary, Emp-address from E1 e1, E2 e2 where e1.Emp-Name=e2.Emp-Name**

gives:-

Code:

+----------------------+-----------+-----------------+

| Emp-name | E-salary | Emp-Address |

+----------------------+-----------+-----------------+

| abc | 2000 | Garia |

| xyz | 5000 | Behala |

| pqr | 7000 | Hindmotor |

| mno | 1500 | Garia |

+----------------------+-----------+-----------------+

**Aggregate Functions**

Code:

**Select MIN (Emp-salary),MAX (Emp-salary) from Emp**

**Set Operations**  
UNION, INTERSECT and EXCEPT operations can be done in SQL corresponding to their operations U, ∩ and **–**in relational algebra only if the domains of the attributes of the relations match and the relations have same arity i.e same number of attributes.