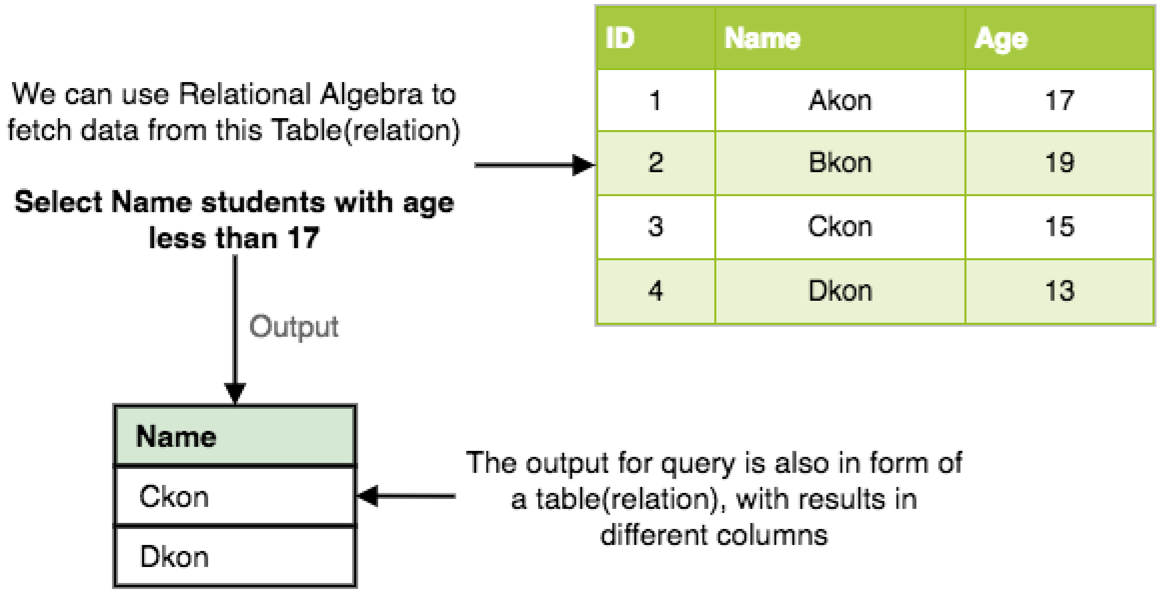
What is Relational Algebra in DBMS?

Relational algebra is a **procedural** query language that works on relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data. When I say that relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved.

Every database management system must define a query language to allow users to access the data stored in the database. **Relational Algebra** is a procedural query language used to query the database tables to access data in different ways.

In relational algebra, input is a relation(table from which data has to be accessed) and output is also a relation(a temporary table holding the data asked for by the user).



We can use Relational Algebra to fetch data from this Table(relation)

ID	Name	Age
1	Akon	17
2	Bkon	19
3	Ckon	15
4	Dkon	13

Select Name students with age less than 17

Name
Ckon
Dkon

The output for query is also in form of a table(relation), with results in different columns

Relational Algebra works on the whole table at once, so we do not have to use loops etc to iterate over all the rows(tuples) of data one by one. All we have to do is specify the table name from which we need the data, and in a single line of command, relational algebra will traverse the entire given table to fetch data for you.

Types of operations in relational algebra

We have divided these operations in two categories:
1. Basic Operations
2. Derived Operations

Basic/Fundamental Operations:

1. Select (σ)
2. Project (∏)
3. Union (∪)
4. Set Difference (-)
5. Cartesian product (X)
6. Rename (ρ)

Derived Operations:

1. Natural Join (⋈)
2. Left, Right, Full outer join (⟕, ⟖, ⟗)
3. Intersection (∩)
4. Division (÷)

<u>BASIC/FUNDAMENTAL OPERATIONS</u>

Select Operator (σ)

Select Operator is denoted by sigma (σ) and it is used to find the tuples (or rows) in a relation (or table) which satisfy the given condition.

Syntax of Select Operator (σ)

```
σ Condition/Predicate(Relation/Table name)
```

Select Operator (σ) Example

```
Table: CUSTOMER
---------------

Customer_Id     Customer_Name      Customer_City
-----------     -------------      -------------
C10100           Steve              Agra
C10111           Raghu              Agra
C10115           Chaitanya          Noida
C10117           Ajeet              Delhi
C10118           Carl               Delhi
```

Query:

```
σ Customer_City="Agra" (CUSTOMER)
```

Output:

```
Customer_Id   Customer_Name    Customer_City
-----------   -------------    -------------
C10100        Steve            Agra
C10111        Raghu            Agra
```

Project Operator (∏)

Project operator is denoted by ∏ symbol and it is used to select desired columns (or attributes) from a table (or relation).

Syntax of Project Operator (∏)

```
∏ column_name1, column_name2, ...., column_nameN(table_name)
```

Project Operator (∏) Example

In this example, we have a table CUSTOMER with three columns, we want to fetch only two columns of the table, which we can do with the help of Project Operator ∏.

```
Table: CUSTOMER

Customer_Id     Customer_Name      Customer_City
-----------     -------------      -------------
C10100           Steve              Agra
```

```
C10111           Raghu              Agra
C10115           Chaitanya          Noida
C10117           Ajeet              Delhi
C10118           Carl               Delhi
```

Query:

```
∏ Customer_Name, Customer_City (CUSTOMER)
```

Output:

```
Customer_Name      Customer_City
-------------      -------------
Steve              Agra
Raghu              Agra
Chaitanya          Noida
Ajeet              Delhi
Carl               Delhi
```

Union Operator (∪)

Union operator is denoted by ∪ symbol and it is used to select all the rows (tuples) from two tables (relations).

We have two relations R1 and R2 both have same columns and we want to select all the tuples(rows) from these relations then we can apply the union operator on these relations.

Note: The rows (tuples) that are present in both the tables will only appear once in the union set. In short you can say that there are no duplicates present after the union operation.

Syntax of Union Operator (∪)

```
table_name1 ∪ table_name2
```

Union Operator (∪) Example

Table 1: COURSE

```
Course_Id  Student_Name   Student_Id
---------  ------------   ----------
C101        Aditya         S901
C104        Aditya         S901
C106        Steve          S911
C109        Paul           S921
C115        Lucy           S931
```

Table 2: STUDENT

```
Student_Id     Student_Name   Student_Age
------------   ----------     -----------
S901           Aditya         19
S911           Steve          18
S921           Paul           19
S931           Lucy           17
S941           Carl           16
S951           Rick           18
```

Query:

```
∏ Student_Name (COURSE) ∪ ∏ Student_Name (STUDENT)
```

Output:

```
Student_Name
------------
Aditya
Carl
Paul
Lucy
Rick
Steve
```

Note: As you can see there are no duplicate names present in the output even though we had few common names in both the tables, also in the COURSE table we had the duplicate name itself.

Intersection Operator (∩)

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

We have two relations R1 and R2 both have same columns and we want to select all those tuples(rows) that are present in both the relations, then in that case we can apply intersection operation on these two relations R1 ∩ R2.

Note: Only those rows that are present in both the tables will appear in the result set.

Syntax of Intersection Operator (∩)

```
table_name1 ∩ table_name2
```

Intersection Operator (∩) Example

Lets take the same example that we have taken above.
Table 1: COURSE

```
Course_Id  Student_Name   Student_Id
---------  ------------   ----------
C101        Aditya         S901
C104        Aditya         S901
C106        Steve          S911
C109        Paul           S921
C115        Lucy           S931
```

Table 2: STUDENT

```
Student_Id     Student_Name   Student_Age
------------   ----------     -----------
S901           Aditya         19
S911           Steve          18
S921           Paul           19
S931           Lucy           17
S941           Carl           16
S951           Rick           18
```

Query:

```
∏ Student_Name (COURSE) ∩ ∏ Student_Name (STUDENT)
```

Output:

```
Student_Name
------------
Aditya
Steve
Paul
Lucy
```

Set Difference (-)

Set Difference is denoted by – symbol. Lets say we have two relations R1 and R2 and we want to select all those tuples(rows) that are present in Relation R1 but **not** present in Relation R2, this can be done using Set difference R1 – R2.

Syntax of Set Difference (-)

```
table_name1 - table_name2
```

Set Difference (-) Example

Lets take the same tables COURSE and STUDENT that we have seen above.

Query:
Lets write a query to select those student names that are present in STUDENT table but not present in COURSE table.

```
∏ Student_Name (STUDENT) - ∏ Student_Name (COURSE)
```

Output:

```
Student_Name
------------
Carl
Rick
```

Cartesian product (X)

Cartesian Product is denoted by X symbol. Lets say we have two relations R1 and R2 then the cartesian product of these two relations (R1 X R2) would combine each tuple of first relation R1 with the each tuple of second relation R2. I know it sounds confusing but once we take an example of this, you will be able to understand this.

Syntax of Cartesian product (X)

```
R1 X R2
```

Cartesian product (X) Example

Table 1: R

```
Col_A    Col_B
-----    ------
AA        100
BB        200
CC        300
```

Table 2: S

```
Col_X     Col_Y
-----     -----
XX         99
YY         11
ZZ         101
```

Query:
Lets find the cartesian product of table R and S.

```
R X S
```

Output:

```
Col_A    Col_B     Col_X     Col_Y
-----    ------    ------    ------
AA        100      XX         99
AA        100      YY         11
AA        100      ZZ         101
BB        200      XX         99
BB        200      YY         11
BB        200      ZZ         101
CC        300      XX         99
CC        300      YY         11
CC        300      ZZ         101
```

Note: The number of rows in the output will always be the cross product of number of rows in each table. In our example table 1 has 3 rows and table 2 has 3 rows so the output has 3×3 = 9 rows.

Rename (ρ)

Rename (ρ) operation can be used to rename a relation or an attribute of a relation.

Rename (ρ) Syntax:

ρ(new_relation_name, old_relation_name)

Rename (ρ) Example

Lets say we have a table customer, we are fetching customer names and we are renaming the resulted relation to CUST_NAMES.

Table: CUSTOMER

```
Customer_Id     Customer_Name      Customer_City
-----------     -------------      -------------
C10100           Steve              Agra
C10111           Raghu              Agra
C10115           Chaitanya          Noida
C10117           Ajeet              Delhi
C10118           Carl               Delhi
```

Query:

```
ρ(CUST_NAMES, ∏(Customer_Name)(CUSTOMER))
```

Output:

```
CUST_NAMES
----------
Steve
Raghu
Chaitanya
Ajeet
Carl
```

Derived Operations

1. JOIN

Join is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

Theta (θ) Join

Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol **θ**.

Notation

$$R1 \ \bowtie_{\theta} \ R2$$

R1 and R2 are relations having attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that the attributes don’t have anything in common, that is R1 ∩ R2 = Φ.

Theta join can use all kinds of comparison operators.

	Student	
SID	**Name**	**Std**
101	Alex	10
102	Maria	11

Subjects	
Class	**Subject**
10	Math
10	English
11	Music
11	Sports

Student_Detail –

$$\mathtt{STUDENT} \bowtie_{\mathtt{Student.Std\ =\ Subject.Class}} \mathtt{SUBJECT}$$

Student_detail				
SID	**Name**	**Std**	**Class**	**Subject**
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

Equijoin

When Theta join uses only **equality** comparison operator, it is said to be equijoin. The above example corresponds to equijoin.

Natural Join (⋈)

Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.

Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Courses		
CID	**Course**	**Dept**
CS01	Database	CS
ME01	Mechanics	ME
EE01	Electronics	EE

<table>
<tr><th colspan="2">HoD</th></tr>
<tr><th>Dept</th><th>Head</th></tr>
<tr><td>CS</td><td>Alex</td></tr>
<tr><td>ME</td><td>Maya</td></tr>
<tr><td>EE</td><td>Mira</td></tr>
</table>

<table>
<tr><th colspan="4">Courses ⋈ HoD</th></tr>
<tr><th>Dept</th><th>CID</th><th>Course</th><th>Head</th></tr>
<tr><td>CS</td><td>CS01</td><td>Database</td><td>Alex</td></tr>
<tr><td>ME</td><td>ME01</td><td>Mechanics</td><td>Maya</td></tr>
<tr><td>EE</td><td>EE01</td><td>Electronics</td><td>Mira</td></tr>
</table>

Outer Joins

Theta Join, Equijoin, and Natural Join are called inner joins. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore, we need to use outer joins to include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins − left outer join, right outer join, and full outer join.

Left Outer Join(R ⟕ S)

All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

Left	
A	**B**
100	Database
101	Mechanics
102	Electronics

Right	
A	B
100	Alex
102	Maya
104	Mira

Courses ⟕ HoD			
A	B	C	D
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya

Right Outer Join: (R ⟖ S)

All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

Courses ⟖ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
102	Electronics	102	Maya
---	---	104	Mira

Full Outer Join: (R ⟗ S)

All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

Courses ⟗ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya
---	---	104	Mira

2. INTERSECTION (∩):

Intersection on two relations R1 and R2 can only be computed if R1 and R2 are **union compatible** (These two relation should have same number of attributes and corresponding attributes in two relations have same domain).

Intersection operator when applied on two relations as R1 ∩ R2 will give a relation with tuples which are in R1 as well as R2. Syntax:

```
Relation1 ∩ Relation2
```

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

EMPLOYEE

EMP_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
5	NARESH	HISAR	9782918192	22
6	SWETA	RANCHI	9852617621	21
4	SURESH	DELHI	9156768971	18

Example: Find a person who is student as well as employee - STUDENT **∩ EMPLOYEE**

RESULT:

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
4	SURESH	DELHI	9156768971	18

DIVIDE (÷):

Division operator A÷B can be applied if and only if:

- Attributes of B is proper subset of Attributes of A.
- The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)
- The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple.
-

Consider the relation STUDENT_SPORTS and ALL_SPORTS given in below:

STUDENT_SPORTS

ROLL_NO	SPORTS
1	Badminton
2	Cricket
2	Badminton
4	Badminton

ALL_SPORTS

SPORTS
Badminton
Cricket

To apply division operator as

```
STUDENT_SPORTS÷ ALL_SPORTS
```

- The operation is valid as attributes in ALL_SPORTS is a proper subset of attributes in STUDENT_SPORTS.
- The attributes in resulting relation will have attributes {ROLL_NO,SPORTS}-{SPORTS}=ROLL_NO
- The tuples in resulting relation will have those ROLL_NO which are associated with all B’s tuple {Badminton, Cricket}. ROLL_NO 1 and 4 are associated to Badminton only. ROLL_NO 2 is associated to all tuples of B. So the resulting relation will be:

RESULT

ROLL_NO
2