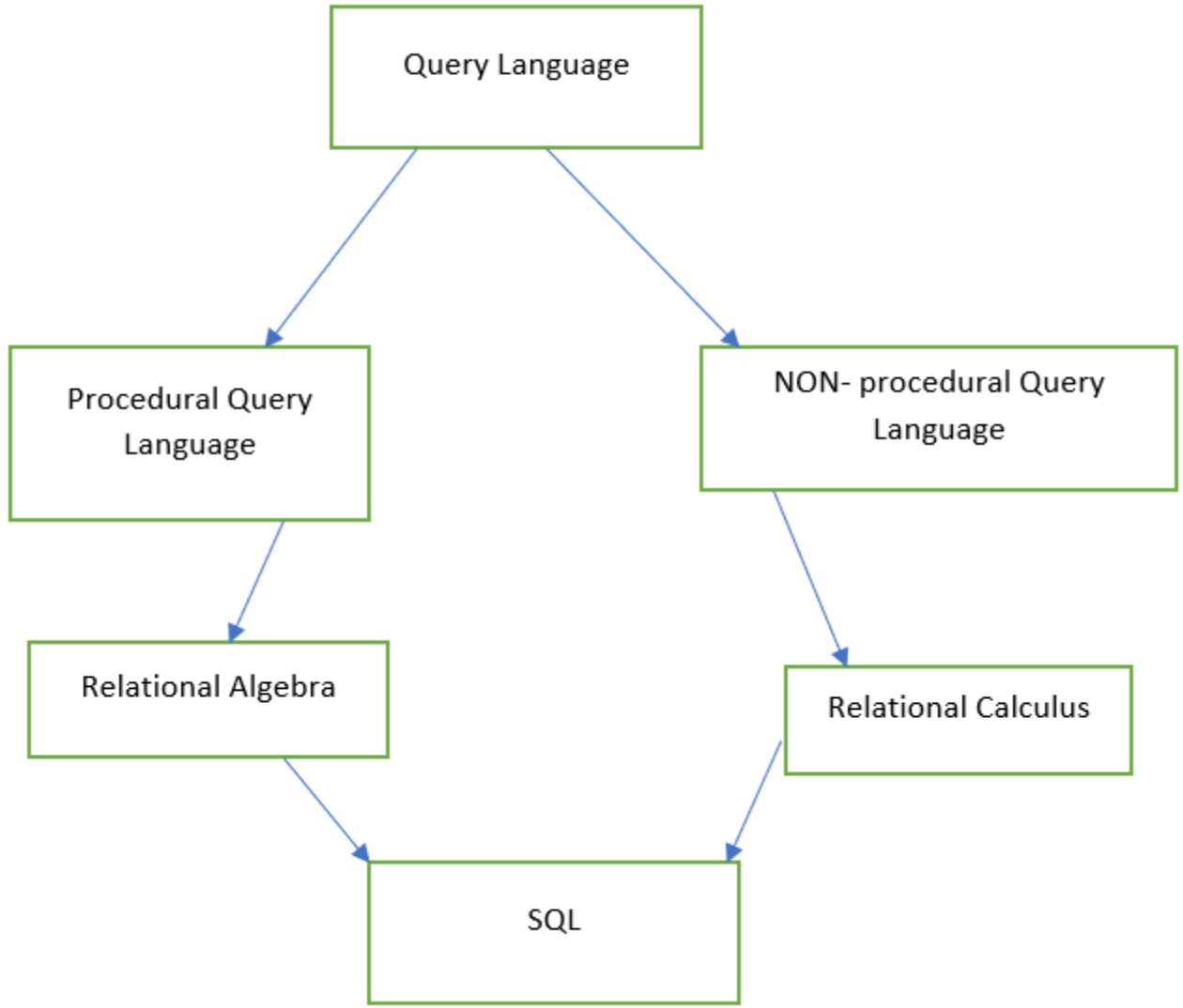
Relational Algebra


Query Language
Procedural Query Language
NON- procedural Query Language
Relational Algebra
Relational Calculus
SQL

What is Relational Algebra?

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.

The primary operations that we can perform using relational algebra are:

- Select
- Project
- Union
- Set Different
- Cartesian product
- Rename

EMployess

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

Selection Operator-

- Selection Operator (σ) is a unary operator in relational algebra that performs a selection operation.
- It selects those rows or tuples from the relation that satisfies the selection condition.

Notation − $\sigma_p(r)$ (Greek letter sigma () to denote selection)

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

Syntax-

$$\sigma_{<selection\_condition>}(R)$$

Examples-

Select tuples from a relation "Employees" where job_id is "ST_MAN"

Select * from employees
Where job_id='ST_MAN';

But in relational algebra

$\sigma_{job\_id = "ST\_MAN"}$ (employees)

Select tuples from a relation “Employees” where job_id is “ST_MAN” and department_id is 90

Select * from employees
Where job_id=‘ST_MAN’
And department_id=90;

But in relational algebra

$$\sigma_{\text{job\_id} = \text{“ST\_MAN”} \wedge \text{department\_id} = \text{“90”}} \text{ (employees)}$$

Select tuples from a relation “Employees” where job_id is “ST_MAN” and department_id is 90 or salary less than 7000.

$$\sigma_{\text{job\_id} = \text{“ST\_MAN”} \wedge \text{department\_id} = \text{“90”} \vee \text{salary} < \text{”7000}''} \text{ (employees)}$$

Project Operation The project operation is a unary operation that returns its argument relation, with certain attributes left out. Since a relation is a set, any duplicate rows are eliminated. Projection is denoted by the uppercase Greek letter pi (π).

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A1, A2, An}(r)$

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example −

$\prod_{employee\_id, first\_name}$ (Employees)

SQL Query:

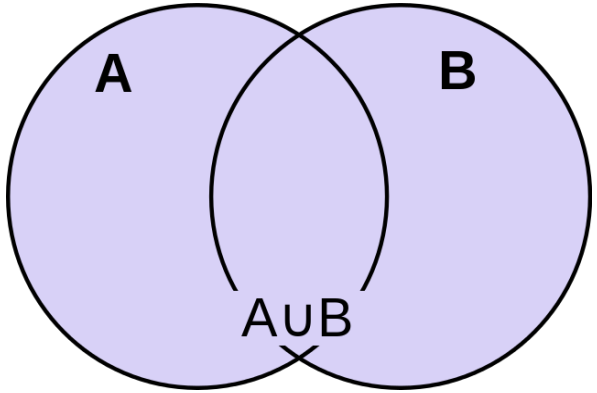
```
Select employee_id, frist_name from employees;
```

Select tuples from a relation “Employees” where job_id is “ST_MAN” and department_id is 90

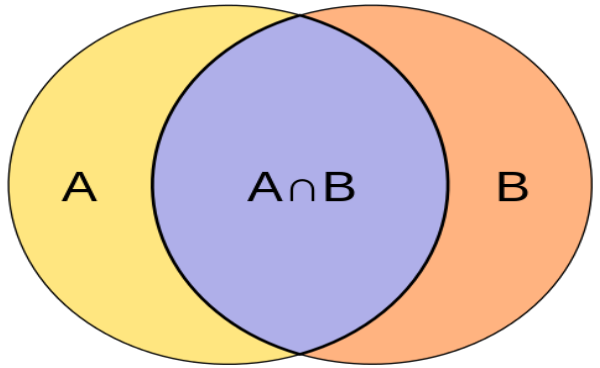
$\Pi_{\text{employee\_id, first\_name, last\_name, job\_id}} (\sigma_{\text{job\_id} = \text{“ST\_MAN”} \wedge \text{department\_id} = \text{“90”}} (\text{employees}))$

Select employee_id, first_name, last_name, job_id from employees
Where job_id=‘ST_MAN’
And department_id=90;

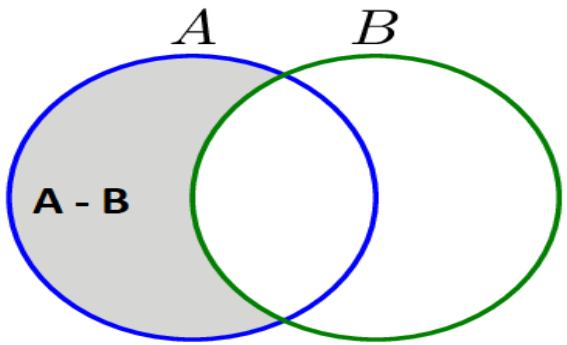
A = {1, 3, 5, 7, 9}
B = {3, 6, 9, 12, 15}
A∪B = {1, 3, 5, 6, 7, 9, 12, 15}



A = {2, 3, 5, 7, 11}
B = {1, 3, 5, 7, 9, 11}
A∩B = {3, 5, 7, 11}



A = {2, 3, 5, 7, 11}
B = {1, 3, 5, 7, 9, 11}
A-B = {2}



Union Operator (∪)-

Let R and S be two relations.

Then-

R ∪ S is the set of all tuples belonging to either R or S or both.
In R ∪ S, duplicates are automatically removed.

Union

In relational algebra, an operator used to merge (append) two tables into a new table, dropping the duplicate rows. The tables must be *union compatible (*Two or more tables that have the same number of columns and the corresponding columns have compatible domains*)*

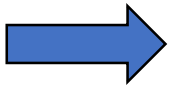
PhD_student

Id	Name
101	ravi
103	rohit
102	ram
104	mohit

∪

employee

id	name
101	ram
109	ashi
201	Ayush



Id	Name
101	ravi
103	rohit
102	ram
104	mohit
109	**ashi**
201	Ayush

$\prod_{\text{id, first\_name}}$ **(Student)** **U** $\prod_{\text{id, first\_name}}$ **(Employees)**

Intersection

An intersection is defined by the symbol ∩

A ∩ B

Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.

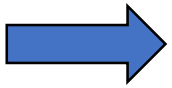
PhD_student

Id	Name
101	ravi
103	rohit
102	ram
104	mohit

∩

employee

id	name
101	ram
109	ashi
201	Ayush



Id	Name
102	ram

$$\prod_{\text{id, first\_name}} \textbf{(Student)} \cap \prod_{\text{id, first\_name}} \textbf{(Employees)}$$

Set Difference (-)

- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.

•The attribute name of A has to match with the attribute name in B.

•The two-operand relations A and B should be either compatible or Union compatible.

•It should be defined relation consisting of the tuples that are in relation A, but not in B.

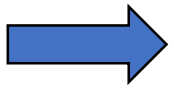
PhD_student

Id	Name
101	ravi
103	rohit
102	ram
104	mohit

U

employee

id	name
101	ram
109	ashi
201	Ayush



Id	Name
101	ravi
103	rohit
104	mohit

$\prod_{\text{id, first\_name}}$ **(Student)** - $\prod_{\text{id, first\_name}}$ **(Employees)**

The Cartesian-Product Operation

The **Cartesian-product** operation, denoted by a cross (×), allows us to combine information from any two relations. We write the Cartesian product of relations r_1 and r_2 as $r_1 \times r_2$.

A = {7, 8} and B = {2, 4, 6}
Then,

A × B = {(7, 2); (7, 4); (7, 6); (8, 2); (8, 4); (8,6)}

R

Col_A	Col_B
AA	100
BB	200
CC	300

(×)

S

Col_X	Col_Y
XX	99
YY	11
ZZ	101

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

Employees

Name	Type
EMPLOYEE_ID	NUMBER(6)
FIRST_NAME	VARCHAR2(20)
LAST_NAME	VARCHAR2(25)
EMAIL	VARCHAR2(25)
PHONE_NUMBER	VARCHAR2(20)
HIRE_DATE	DATE
JOB_ID	VARCHAR2(10)
SALARY	NUMBER(8,2)
COMMISSION_PCT	NUMBER(2,2)
MANAGER_ID	NUMBER(6)
DEPARTMENT_ID	NUMBER(4)

Departments

Name	Type
DEPARTMENT_ID	NUMBER(4)
DEPARTMENT_NAME	VARCHAR2(30)
MANAGER_ID	NUMBER(6)
LOCATION_ID	NUMBER(4)

$$\Pi_{\text{employee\_id, job\_id,depart\_name}} (\sigma_{\text{job\_id = “ST\_MAN” } \wedge \text{ department\_id = “90”}} \text{ (employees x departments))}$$

$$\Pi_{\text{employee\_id, job\_id,department\_name}} (\sigma_{\text{job\_id = “ST\_MAN” } \wedge \text{ department\_id = “90” } \wedge \text{ employees.department\_id=depaerment.department\_id}} \text{ (employees x departments))}$$

Rename (ρ) Example

Lets say we have a table Departments, we are fetching departments names and we are renaming the resulted relation to DEPT_NAMES. Denoted by the lowercase Greek letter rho **(ρ).**

$$\rho(\text{Dept\_NAMES}, \prod_{(\text{Departments\_Name})} (\text{CUSTOMER}))$$

The ***natural join*** is a binary operation that allows us to combine certain selections and a Cartesian product into one operation. It is denoted by the join symbol ⋈

Cross join performed a cartesian product among the rows of two different tables. where the columns name may or may not be matched.....but in natural join its mandatory that in order to perform join operation columns name of two tables must be matched

$$\Pi_{\text{employee\_id, job\_id,department\_name}} (\sigma_{\text{department.department\_id=employees.department\_id}} (\text{employees} \bowtie \text{departments}))$$

Theta Join

Theta Join allows you to merge two tables based on the condition represented by theta. Theta joins work for all comparison operators. The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ**

$$A \bowtie_{\theta} B$$

$$\Pi_{\text{employee\_id, job\_id, depart\_name}} (\sigma_{\text{locations.location\_id between 1700 and 2400} \ \wedge \ \text{department\_id} > 50} (\text{employees} \bowtie_{\theta} \text{departments}))$$

```
SELECT a.department_id, a.department_name, b.city FROM departments a, locations b
WHERE b.location_id BETWEEN 1700 AND 2400
 AND
 a.department_id < 50;
```

EQUI join:

When a theta join uses only equivalence condition, it becomes an equi join.

$$\Pi_{\text{employee\_id, job\_id, department\_name}} (\sigma_{\text{department.department\_id=employees.department\_id}} (\text{employees} \bowtie \text{departments}))$$

```
SELECT first_name, job_id, departments.department_id, department_name FROM employees, departments
WHERE employees.department_id = departments.department_id
```

Outer Join

An outer join doesn't require each record in the two join tables to have a matching record. In this type of join, the table retains each record even if no other matching record exists.

Three types of Outer Joins are:

- Left Outer Join
- Right Outer Join
- Full Outer Join

LEFT OUTER JOIN 

RIGHT OUTER JOIN 

FULL OUTER JOIN 

Left Outer Join ⟕

The LEFT JOIN returns all the rows from the table on the left even if no matching rows have been found in the table on the right. Where no matching record found in the table on the right, NULL is returned.

A ⟕ B

$\Pi_{\text{employee\_id, job\_id, department\_name}} (\sigma_{\text{department.department\_id=employees.department\_id}} (\text{employees} ⟕ \text{departments}))$

```
SELECT e.last_name, e.department_id,
d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id(+);
```

```
SELECT e.last_name, e.department_id,
d.department_name FROM employees e LEFT
OUTER JOIN departments d
on (e.department_id= d.department_id);
```

The query in the above example retrieves all rows in the `EMPLOYEES` table, even if there is no match in the `DEPARTMENTS` table.

Right Outer Join:

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.

A ⟗ B

$$\Pi_{\text{employee\_id, job\_id, department\_name}} (\sigma_{\text{department.department\_id=employees.department\_id}} (\text{employees} ⟗ \text{departments}))$$

```
SELECT e.last_name, e.department_id,
d.department_name
FROM employees e, departments d
WHERE e.department_id (+)= d.department_id;
```

```
SELECT e.last_name, e.department_id,
d.department_name FROM employees e RIGHT
OUTER JOIN departments d
on (e.department_id= d.department_id);
```

The query in the above example retrieves all rows in the DEPARTMENTS table, even if there is no match in the EMPLOYEES table.

Full Outer Join

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

$A \bowtie B$

$\Pi_{\text{employee\_id, job\_id, department\_name}} (\sigma_{\text{department.department\_id=employees.department\_id}} (\text{employees} \bowtie \text{departments}))$

```
SELECT e.last_name, e.department_id, d.department_name FROM employees e Full
OUTER JOIN departments d
on (e.department_id= d.department_id);
```

Aggregate functions

Aggregate functions take a collection of values and return a single value as a result. For example, the aggregate function **sum, avg, min, max etc** takes a collection of values and returns the single resultant value of these values. Thus, the function **sum** applied on the collection:

- $G_{sum(salary)}$ **(Employees)**
- $G_{avg(salary)}$ **(Employees)**
- $G_{min(salary)}$ **(Employees)**
- $G_{max(salary)}$ **(Employees)**

Find the average salary in each department.

$_{Department\_id}G_{avg(salary)}$ **(Employees)**

$G_{\text{count-distinct}}(department\_id)(\sigma_{salary>6000}(Employees))$

Thank You