

Aggregate Functions and Joins in SQL

Aggregate Functions

- **Types of Aggregate Functions:**

- SUM()
- AVG()
- MAX()
- MIN()
- COUNT()
- STDDDEV()
- VARIANCE()

Note: Avg, Sum, Variance, Stddev functions can be used only with numeric data types.

- **Syntax:**

```
SELECT      [column,] group_function(column), ...  
FROM        table  
[WHERE      condition]  
[GROUP BY   column]  
[ORDER BY   column];
```

Aggregate Functions

- **SUM(n):** Returns sum of *n* numbers. Sum function ignores the null values.

- What will be the result for the following SQL Queries on the given employee table:

➤ **SELECT** sum (salary) **FROM** *employee*;

- Result:

Sum (Salary)
21,500

➤ **SELECT** sum (commission) **FROM** *employee*;

- Result:

Sum (Commission)
1200

➤ **SELECT** sum (ename) **FROM** *employee*;

- Result:

Error at line 1 :
ORA-01722 : invalid number

Eid	Ename	Hiredate	Salary	Commission
10A1	Ramesh	17-Jan-1980	5000	
10A2	Ajay	05-Jul-1985	5000	500
10A3	Ravi	12-Aug-1981	1500	
10A4	Nikesh	03-Mar-1983	3000	700
10A5	Ravi	05-Jul-1985	3000	
10A6	Suresh	17-Jan-1980	4000	

Note: Applicable only on numeric data.

Aggregate Functions

- **Avg(n):** Returns average of *n* numbers. Avg function ignores the null values.

- What will be the result for the following SQL Queries on the given employee table:

➤ **SELECT Avg (salary) FROM employee;**

- Result:

Avg (Salary)
3583.33

➤ **SELECT Avg (commission) FROM employee;**

- Result:

Avg (Commission)
600

➤ **SELECT Avg (ename) FROM employee;**

- Result:

Error at line 1 :
ORA-01722 : invalid number

Eid	Ename	Hiredate	Salary	Commission
10A1	Ramesh	17-Jan-1980	5000	
10A2	Ajay	05-Jul-1985	5000	500
10A3	Ravi	12-Aug-1981	1500	
10A4	Nikesh	03-Mar-1983	3000	700
10A5	Ravi	05-Jul-1985	3000	
10A6	Suresh	17-Jan-1980	4000	

Note: Only applicable on numeric data.

Aggregate Functions

- **Max(n):** Returns the maximum in *n* values. Max function ignores the null values.

- What will be the result for the following SQL Queries on the given employee table:

➤ **SELECT Max (Salary) FROM employee;**

- Result:

Max (Salary)

5000

➤ **SELECT Max (Commission) FROM employee;**

- Result:

Max (Commission)

700

➤ **SELECT Max (Ename) FROM employee;**

- Result:

Max (Ename)

Suresh

Eid	Ename	Hiredate	Salary	Commission
10A1	Ramesh	17-Jan-1980	5000	
10A2	Ajay	05-Jul-1985	5000	500
10A3	Ravi	12-Aug-1981	1500	
10A4	Nikesh	03-Mar-1983	3000	700
10A5	Ravi	05-Jul-1985	3000	
10A6	Suresh	17-Jan-1980	4000	

Note: Max Function operates on both numeric and character value. When applied on character value, this function compares ASCII code..

Aggregate Functions

- **Min(n):** Returns the minimum in *n* values. Min function ignores the null values.

- What will be the result for the following SQL Queries on the given employee table:

➤ **SELECT Min (Salary) FROM employee;**

- Result:

Min (Salary)
1500

➤ **SELECT Min (Commission) FROM employee;**

- Result:

Min (Commission)
500

➤ **SELECT Min (Ename) FROM employee;**

- Result:

Min (Ename)
Ajay

Eid	Ename	Hiredate	Salary	Commission
10A1	Ramesh	17-Jan-1980	5000	
10A2	Ajay	05-Jul-1985	5000	500
10A3	Ravi	12-Aug-1981	1500	
10A4	Nikesh	03-Mar-1983	3000	700
10A5	Ravi	05-Jul-1985	3000	
10A6	Suresh	17-Jan-1980	4000	

Note: Min Function operates on both numeric and character value. When applied on character value, this function compares ASCII code..

Aggregate Functions

- What will be the result of applying count on the given employee table:

➤ **SELECT Count (*) FROM** *employee*;

- Result:

Count (*)
6

➤ **SELECT Count (Ename) FROM** *employee*;

- Result:

Count (Ename)
6

➤ **SELECT Count (distinct ename) FROM** *employee*;

- Result:

Count (distinct ename)
5

➤ **SELECT Count (commission) FROM** *employee*;

- Result:

Count (Commission)
2

Eid	Ename	Hiredate	Salary	Commission
10A1	Ramesh	17-Jan-1980	5000	
10A2	Ajay	05-Jul-1985	5000	500
10A3	Ravi	12-Aug-1981	1500	
10A4	Nikesh	03-Mar-1983	3000	700
10A5	Ravi	05-Jul-1985	3000	
10A6	Suresh	17-Jan-1980	4000	

Creating Groups of Data

Find out the depart wise average salary.

EMPLOYEES

DEPARTMENT_ID	SALARY
10	4400
20	13000
20	6000
50	5800
50	3500
50	3100
50	2500
50	2600
60	9000
60	6000
60	4200
80	10500
80	8600
80	11000
90	24000
90	17000

...

20 rows selected.

The average salary in EMPLOYEES table for each department.

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

Group By Clause

- Divide rows in a table into smaller groups by using the GROUP BY clause.

- **Syntax:**

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

- Using a **Where** clause , we can exclude rows before dividing the into groups.
- We must use the columns in the **Group By** clause. We cannot use a column *alias* in the **Group By** clause.

Group By Clause

- Divide rows in a table into smaller groups by using the GROUP BY clause.

- **Syntax:**

```
SELECT          column, group_function(column)
FROM            table
[WHERE          condition]
[GROUP BY       group_by_expression]
[ORDER BY       column];
```

- The Group By statement is used along with the SQL aggregate/group functions to provide means of grouping the result dataset by certain database table column(s).
- *All columns in the **SELECT** list that are not in group functions must be in the **Group By** clause.*
- *The other columns from the **Select** list should have columns which are single valued per group, therefore, either constant **or** aggregate functions on the other columns.*

Group By Clause

- Divide rows in a table into smaller groups by using the GROUP BY clause.

- **Syntax:**

```
SELECT          column, group_function(column)
FROM            table
[WHERE          condition]
[GROUP BY      group_by_expression]
[ORDER BY      column];
```

- *By default rows are sorted by ascending order of the columns included in the **Group By** clause.*
- *We can override this by using the **Order By** clause.*

Group By Clause: Example

- Find out the department wise average salary.

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY  department_id ;
```

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

8 rows selected.

- Case1:** All columns in the *SELECT* list that are not in group functions must be in the **GROUP BY** clause.

Grouping By and Having Clause

- Write a SQL query to find the average salary of each department but show only those department whose average salary is greater than 8000.

Solution:

```
SELECT    department_id, AVG(salary)
FROM      employees
WHERE     AVG(salary) > 8000
GROUP BY department_id;
```

Error:

```
WHERE  AVG(salary) > 8000
      *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

- You **cannot** use the **WHERE** clause to restrict groups.
- You **use** the **HAVING** clause to restrict groups.
- You **cannot** use group functions in the **WHERE** clause.

Excluding Groups

EMPLOYEES

DEPARTMENT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
80	10500
80	11000
80	8600

...

20	6000
110	12000
110	8300

20 rows selected.

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

Having Clause

- Use the HAVING clause to restrict groups:
 1. Rows are grouped.
 2. The group function is applied.
 3. Groups matching the HAVING clause are displayed.

- **Syntax:**

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[HAVING     group_condition]
[ORDER BY   column];
```

Having Clause: Example

- Write a SQL query to find the average salary of each department but show only those department whose average salary is greater than 8000.
- **Solution:**

```
SELECT Department_Id, Avg (salary)
FROM employees
Group By Department_ID;
Having Avg (salary) > 8000;
```

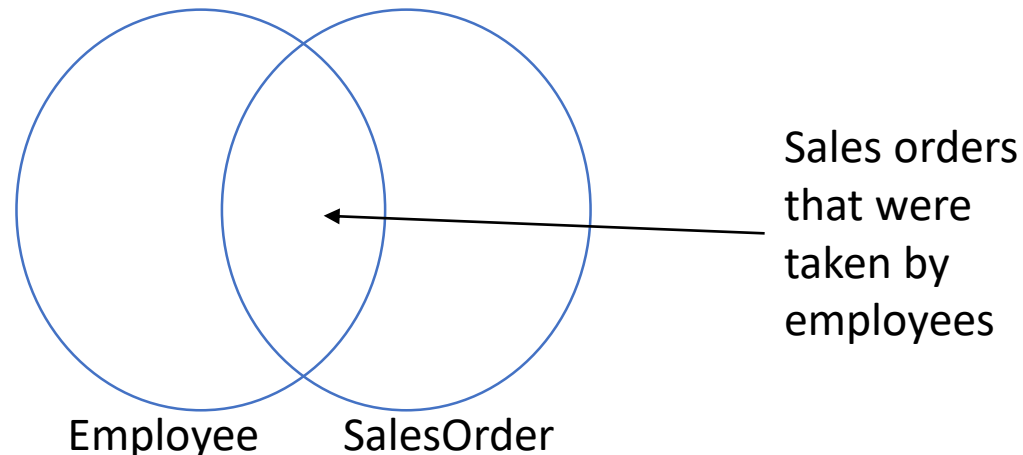

Joins in SQL

Joins on Multiple Tables:

- SQL provides a convenient operation to retrieve information from multiple tables.
- This operation is called **join**.
- The join operation will **combine** the tables into one large table with all possible combinations (Math: Cartesian Product), and then it will filter the rows of this combined table to yield useful information.

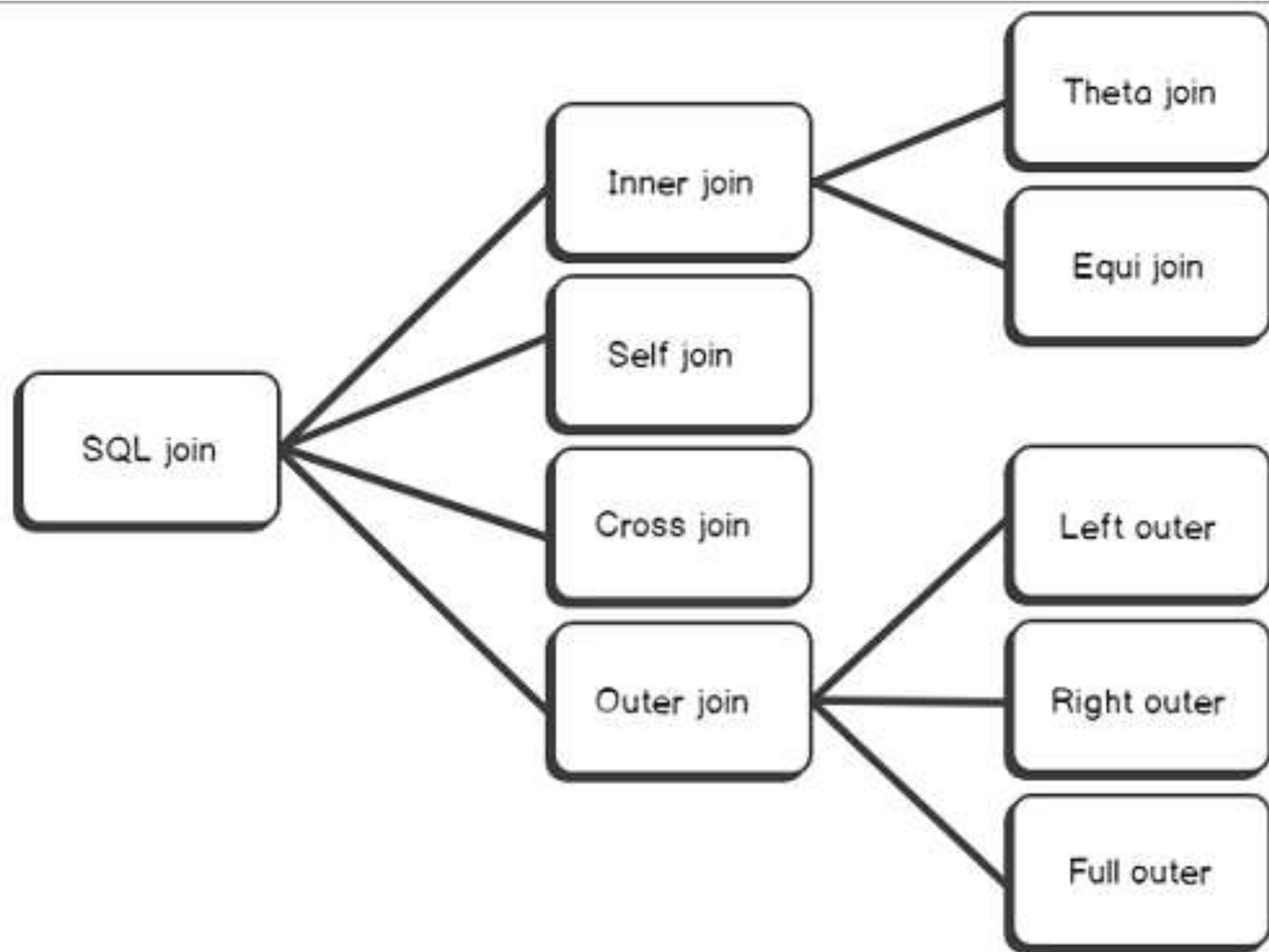
Join Concepts

- Combine rows from multiple tables by specifying matching criteria
 - Usually based on primary key – foreign key relationships
 - For example, return rows that combine data from the **Employee** and **SalesOrder** tables by matching the **Employee.EmployeeID** primary key to the **SalesOrder.EmployeeID** foreign key
- It helps to think of the tables as sets in a Venn diagram

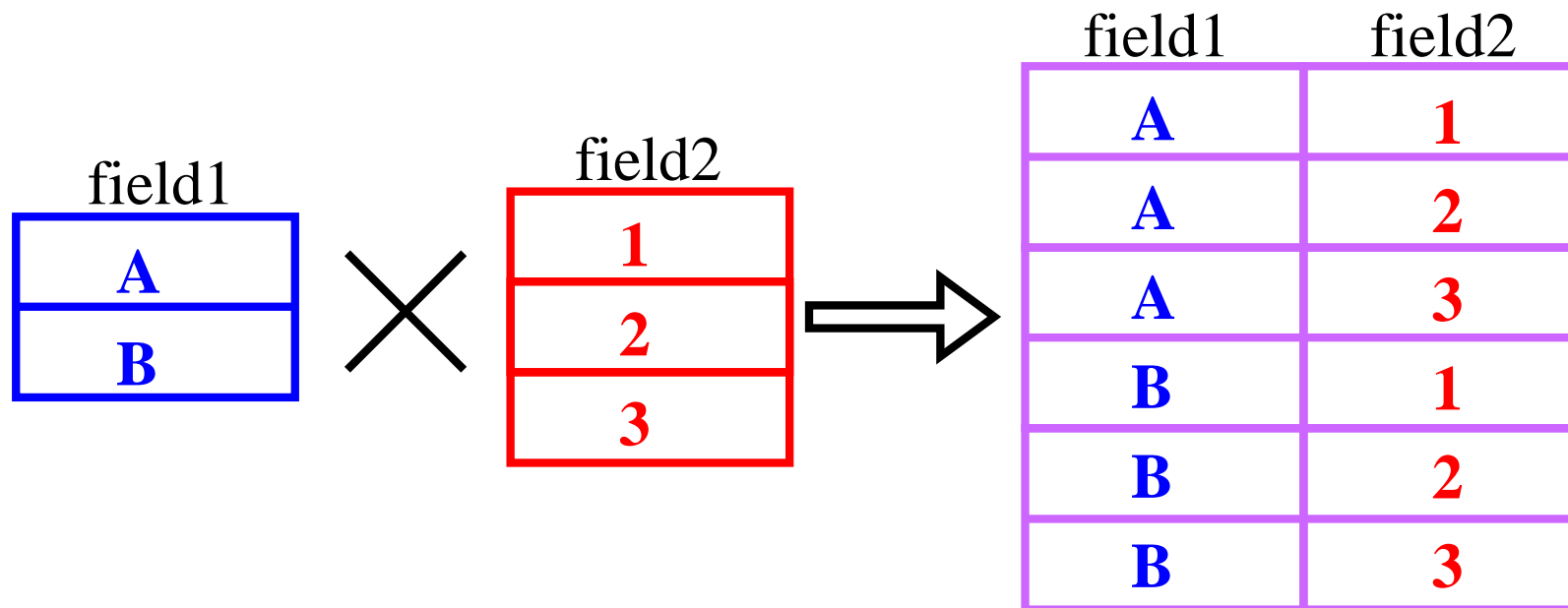


Different type of Joins

1. Cross Join
2. Inner Join
 - i. Equi/Equality join ('=')
 - ii. Natural join
 - iii. Theta join (or non-equi join (<, <=, >, >=, <>))
3. Outer Join
 - i. Left outer join
 - ii. Right outer join
 - iii. Full outer join
4. Self Join
5. Semi Join



Multiple Tables:



What kind of Join is this?

SELECT *

FROM Students S ?? Enrolled E;

S

S.name	S.sid
Jones	11111
Smith	22222

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150

S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	11111	History105
Smith	22222	11111	DataScience194
Smith	22222	22222	French150

The **CROSS JOIN** is used to generate a paired combination of each row of the first table with each row of the second table. This **join** type is also known as cartesian **join**.

```
SELECT *  
FROM Students S CROSS JOIN Enrolled E;
```

S

S.name	S.sid
Jones	11111
Smith	22222

E

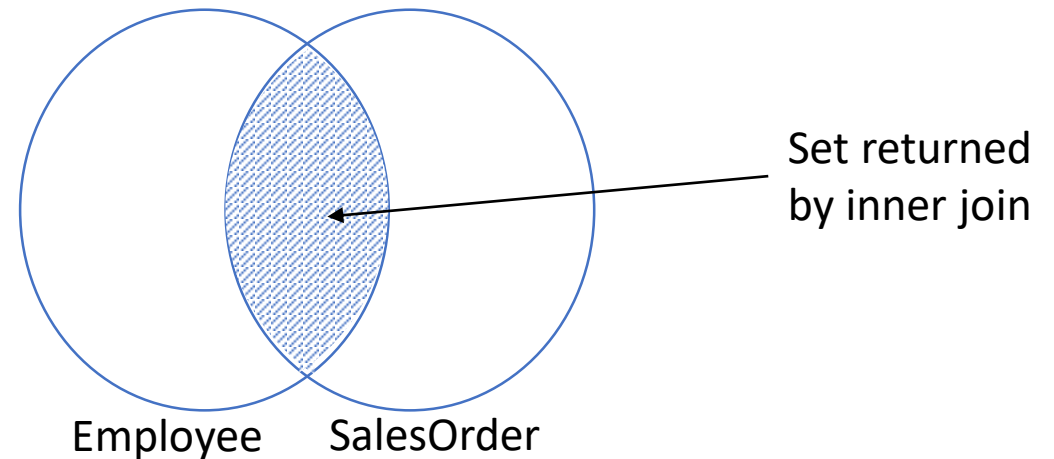
E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150

S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	11111	History105
Smith	22222	11111	DataScience194
Smith	22222	22222	French150

Inner Joins

- Return only rows where a match is found in both input tables
- Match rows based on attributes supplied in predicate
- If join predicate operator is =, then it known as equi-join

```
SELECT emp.FirstName, ord.Amount  
FROM Employee emp  
[INNER] JOIN SalesOrder ord  
ON emp.EmployeeID = ord.EmployeeID
```



SQL Inner Joins (Equi join)

```
SELECT S.name, E.classid  
FROM Students S (INNER) JOIN Enrolled E  
ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

Note the previous version of this query (with no join keyword) is an “Implicit join”

SQL Inner Joins (Equi join)

```
SELECT S.name, E.classid
FROM Students S (INNER) JOIN Enrolled E
ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

Unmatched keys

Joins and Inference

- Chaining relations together is the basic inference method in relational DBs. It produces new relations (effectively new facts) from the data:

```
SELECT S.name, M.mortality
FROM Students S, Mortality M
WHERE S.Race=M.Race
```

S

Name	Race
Socrates	Man
Thor	God
Barney	Dinosaur
Blarney stone	Stone

M

Race	Mortality
Man	Mortal
God	Immortal
Dinosaur	Mortal
Stone	Non-living

Joins and Inference

- Chaining relations together is the basic inference method in relational DBs. It produces new relations (effectively new facts) from the data:

```
SELECT S.name, M.mortality  
FROM Students S, Mortality M  
WHERE S.Race=M.Race
```

Name	Mortality
Socrates	Mortal
Thor	Immortal
Barney	Mortal
Blarney stone	Non-living

Natural Join

Natural Join joins two tables based on same attribute name and datatypes. The resulting table will contain all the attributes of both the table but keep only one copy of each common column.

Student

Roll_No	Name
1	A
2	B
3	C

Mark

Roll_No	Marks
2	70
3	50
4	85

Select * from Student natural join Mark;

Roll_No	Name	Marks
2	B	70
3	C	50

Difference between Equi join and Natural join

```
SELECT * FROM student S INNER JOIN Marks M ON S.Roll_No = M.Roll_No;
```

Roll_No	Name	Roll_No	Marks
2	B	2	70
3	C	3	50

1. Natural Join joins two tables based on same attribute name and datatypes.

Inner Join joins two table on the basis of the column which is explicitly specified in the ON clause.

2. In Natural Join, The resulting table will contain all the attributes of both the tables but keep only one copy of each common column

In Inner Join, The resulting table will contain all the attribute of both the tables including duplicate columns also

3. In Natural Join, If there is no condition specifies then it returns the rows based on the common column

In Inner Join, only those records will return which exists in both the tables

4. SYNTAX:
SELECT *
FROM table1 NATURAL JOIN table2;

SYNTAX:
SELECT *
FROM table1 INNER JOIN table2 ON
table1.Column_Name =
table2.Column_Name;

What kind of Join is this?

```
SELECT *  
FROM Students S, Enrolled E  
WHERE S.sid <= E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150

S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	22222	French150

Theta Joins (<, <=, >, >=, <>)

SELECT *

FROM Students S, Enrolled E

WHERE S.sid <= E.sid

S

S.name	S.sid
Jones	11111
Smith	22222

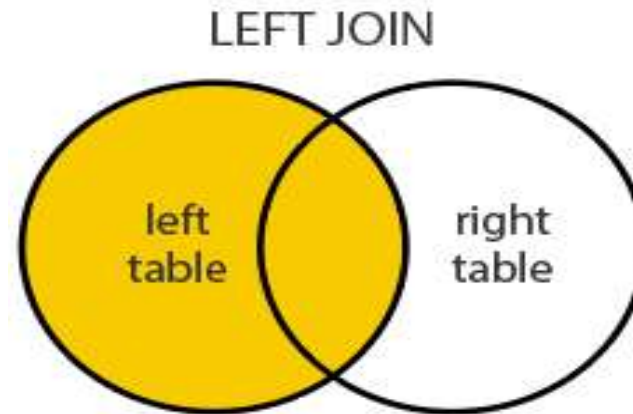
E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150

S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	22222	French150

Left Outer Join

A LEFT JOIN performs a join starting with the first (left-most) table. Then, any matched records from the second table (right-most) will be included. LEFT JOIN and LEFT OUTER JOIN are the same. The result is 0 records from the right side, if no there is no match.



```
SELECT column-names  
FROM table-name1 LEFT OUTER JOIN table-name2  
ON column-name1 = column-name2
```

What kind of Join is this?

```
SELECT S.name, E.classid
FROM Students S ?? Enrolled E
ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
Brown	NULL

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

Query: List all students name and their classid irrespective whether they enrolled for that class or not.

```
SELECT S.name, E.classid
FROM Students S LEFT OUTER JOIN Enrolled E
ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

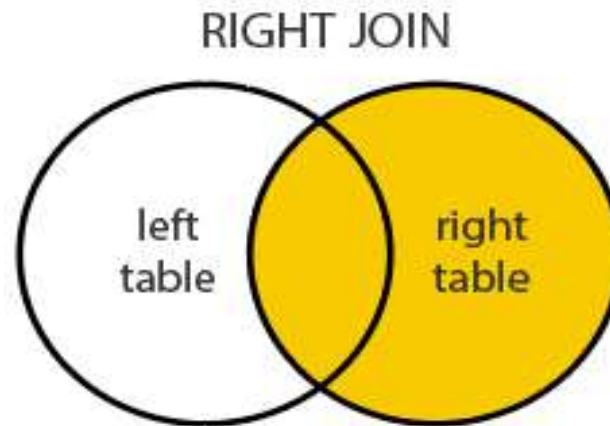
S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
Brown	NULL

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

Right outer join

A RIGHT JOIN performs a join starting with the second (right-most) table and then any matching first (left-most) table records. RIGHT JOIN and RIGHT OUTER JOIN are the same. The result is 0 records from the left side, if no there is no match.



SELECT column-names

FROM table-name1 RIGHT OUTER JOIN table-name2

ON column-name1 = column-name2

What kind of Join is this?

```
SELECT S.name, E.classid
FROM Students S ?? Enrolled E
ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
NULL	English10

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

Query: List all students name and their classid irrespective whether for a class any student enrolled or not.

```
SELECT S.name, E.classid
FROM Students S RIGHT OUTER JOIN Enrolled E
ON S.sid=E.sid
```

S	S.name	S.sid
	Jones	11111
	Smith	22222
	Brown	33333

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
NULL	English10

E	E.sid	E.classid
	11111	History105
	11111	DataScience194
	22222	French150
	44444	English10

SQL Joins

```
SELECT S.name, E.classid  
FROM Students S ? JOIN Enrolled E  
ON S.sid=E.sid
```

S

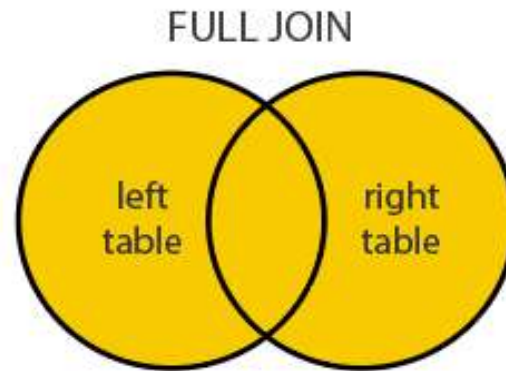
S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
NULL	English10
Brown	NULL

E	E.sid	E.classid
	11111	History105
	11111	DataScience194
	22222	French150
	44444	English10

Full outer join

FULL JOIN returns all matching records from both tables whether the other table matches or not. Be aware that a FULL JOIN can potentially return very large datasets. These two: FULL JOIN and FULL OUTER JOIN are the same.



SELECT column-names

FROM table-name1 FULL OUTER JOIN table-name2

ON column-name1 = column-name2

Query: List all students name and their classid irrespective whether a student enrolled for a class or not, and irrespective whether for a class any student enrolled or not.

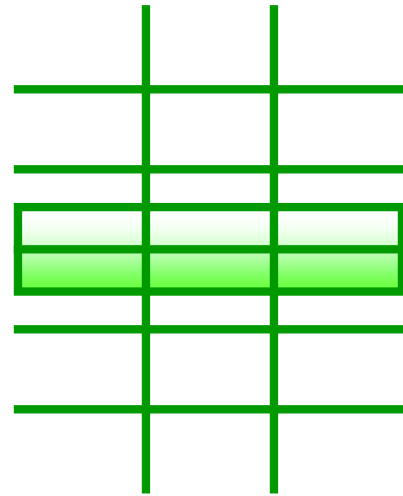
```
SELECT S.name, E.classid
FROM Students S FULL OUTER JOIN Enrolled E
ON S.sid=E.sid
```

S	S.name	S.sid
	Jones	11111
	Smith	22222
	Brown	33333
	S.name	E.classid
	Jones	History105
	Jones	DataScience194
	Smith	French150
	NULL	English10
	Brown	NULL

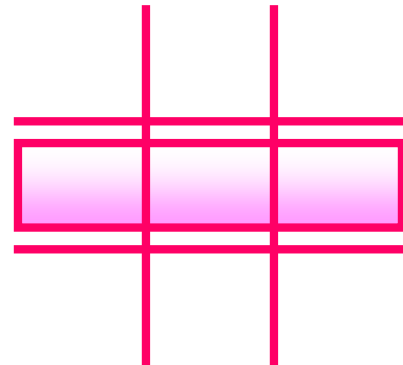
E	E.sid	E.classid
	11111	History105
	11111	DataScience194
	22222	French150
	44444	English10

Outer Join

eg. 28



Natural Join



No Match

Outer Join

Self Join

- A self JOIN occurs when a table takes a 'selfie', that is, it JOINS with itself. A self JOIN is a regular join but the table that it joins to is itself.
- **Joining** a table with itself means that each row of the table is combined with itself and with every other row of the table. SELF JOINS are also useful for comparisons within a table.

SELECT column-names

FROM table-name T1 JOIN table-name T2

WHERE condition

END