ADVANCED SQL CONCEPTS

COMPARISONS INVOLVING NULL AND THREE-VALUED LOGIC

- Meanings of `NULL`
  - **Unknown value**
  - **Unavailable or withheld value**
  - **Not applicable attribute**
- Each individual `NULL` value considered to be different from every other `NULL` value
- SQL uses a three-valued logic:
  - `TRUE`, `FALSE`, and `UNKNOWN` (like Maybe)
- **NULL = NULL comparison is avoided**

COMPARISONS INVOLVING NULL AND THREE-VALUED LOGIC

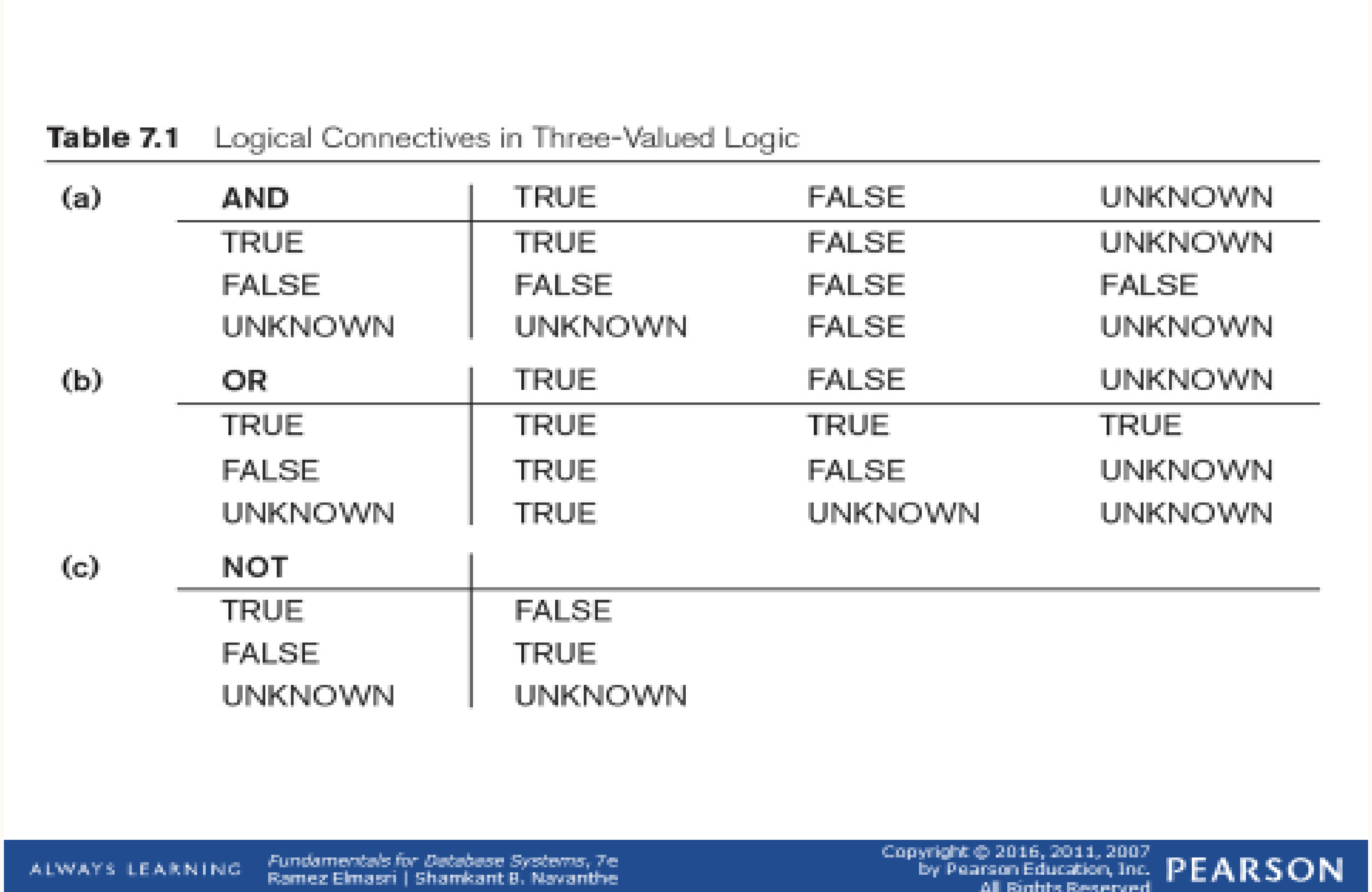


Table 7.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

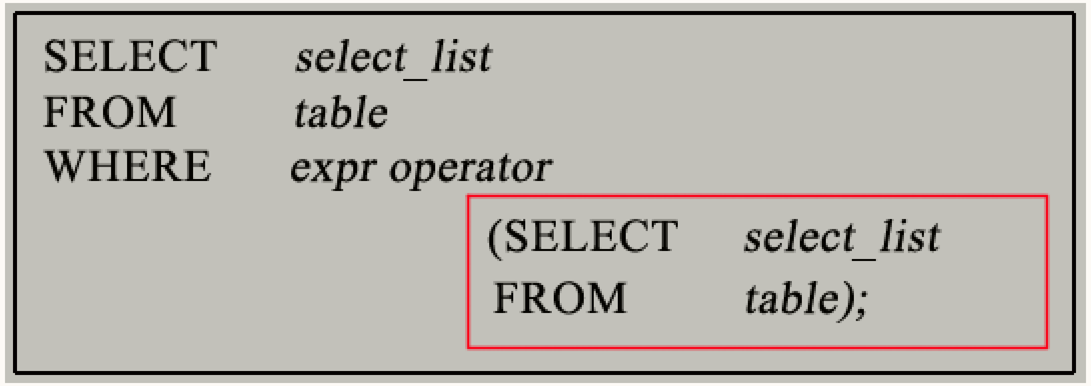
NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

SUBQUERY

- Subquery or Inner query or Nested query
- A subquery is a query within another query.
- The outer query is called as main query and
- The inner query is called as subquery.
- A subquery is usually added in the WHERE Clause of the sql statement.

SUBQUERY

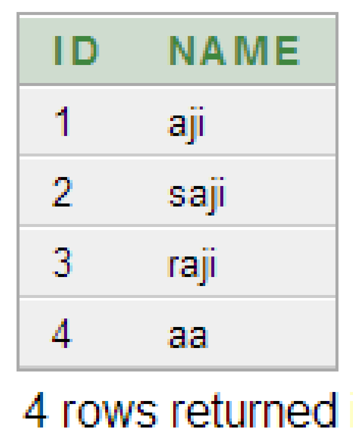
- Subquery or Inner query or Nested query
- Syntax



```
SELECT    select_list
FROM      table
WHERE     expr operator
                         (SELECT    select_list
                          FROM      table);
```

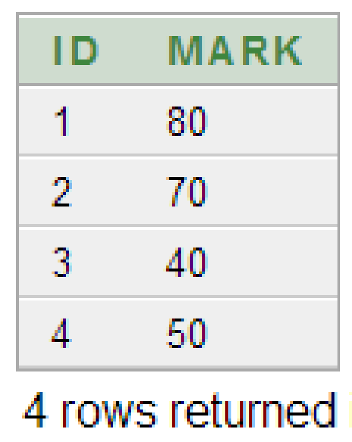
- The subquery (inner query) executes once before the main query (outer query) executes.

SUBQUERY



ID	NAME
1	aji
2	saji
3	raji
4	aa

4 rows returned

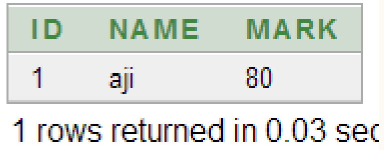


ID	MARK
1	80
2	70
3	40
4	50

4 rows returned

- Subquery or Inner query or Nested query
- Two tables ‘STUDENT' and ‘MARKS' with common field ‘ID'.

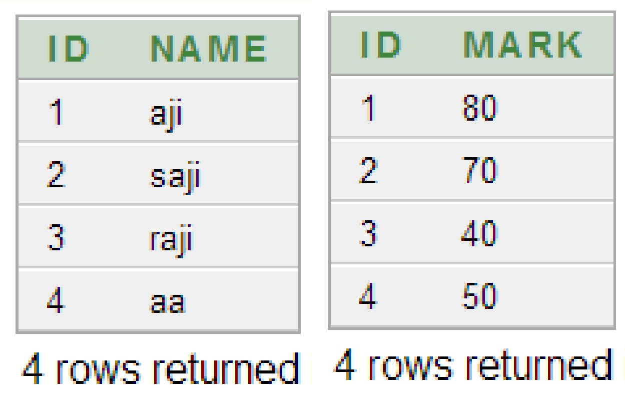
- To write a query to identify all students who get better marks than that of the student who's ID is ‘2‘.
- If we know the mark of ID ’2’ then
  - SELECT A.ID,A.NAME, B.MARK FROM STUDENT A, MARK B WHERE A.ID=B.ID AND B.MARK >70;



ID	NAME	MARK
1	aji	80

1 rows returned in 0.03 sec

SUBQUERY



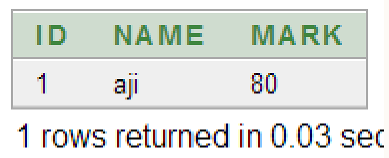
ID	NAME
1	aji
2	saji
3	raji
4	aa

4 rows returned

ID	MARK
1	80
2	70
3	40
4	50

4 rows returned

- Subquery or Inner query or Nested query STUDENT MARKS
- But we do not know the marks of ‘2'.
- we require two queries (Nested query)
- One query returns the marks of '2' and
- Second query identifies the students who get better marks than the result of the first query
- SELECT A.ID,A.NAME, B.MARK FROM STUDENT A, MARKS B WHERE A.ID=B.ID AND B.MARK > (SELECT MARK FROM MARKS WHERE ID=2);



ID	NAME	MARK
1	aji	80

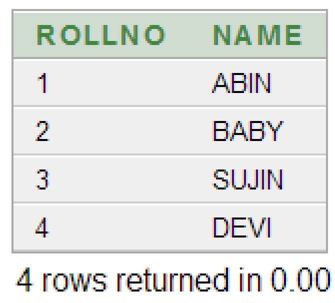
1 rows returned in 0.03 sec

SUBQUERY

- Subqueries can be used with
- SELECT
- INSERT
- UPDAT E, and
- DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

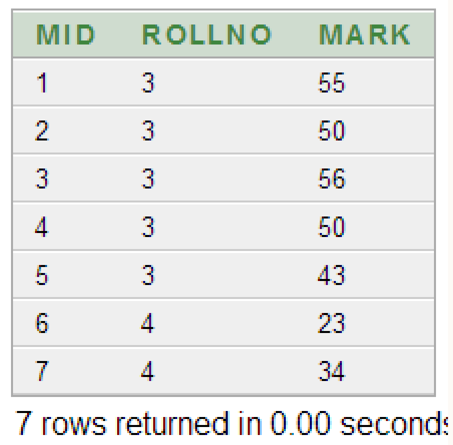
SUBQUERY

- Subqueries in <u>select statement</u> Tab1 Tab2



ROLLNO	NAME
1	ABIN
2	BABY
3	SUJIN
4	DEVI

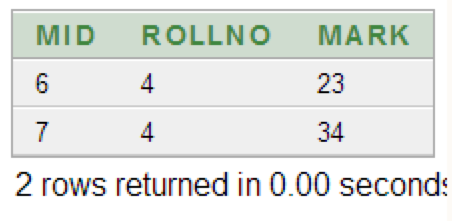
4 rows returned in 0.00



MID	ROLLNO	MARK
1	3	55
2	3	50
3	3	56
4	3	50
5	3	43
6	4	23
7	4	34

7 rows returned in 0.00 seconds

- List all details from tab2, where rollno =4 in tab1
- select * from tab2 where rollno in (select rollno from tab1 where rollno=4)

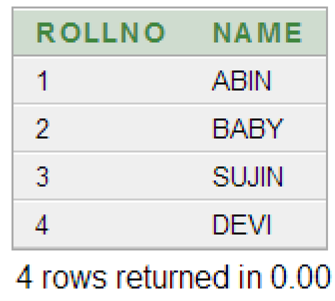


MID	ROLLNO	MARK
6	4	23
7	4	34

2 rows returned in 0.00 seconds

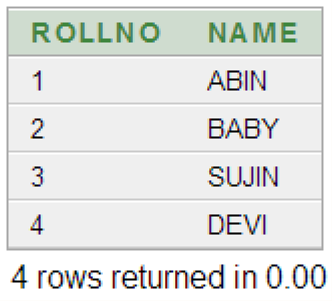
SUBQUERY

- Subqueries in insert statement Tab1 Tab4



ROLLNO	NAME
1	ABIN
2	BABY
3	SUJIN
4	DEVI

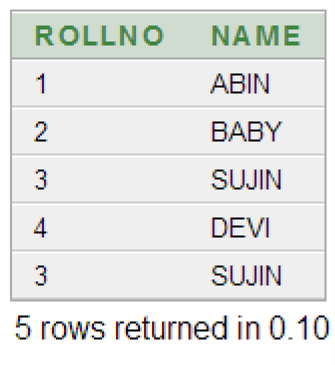
4 rows returned in 0.00



ROLLNO	NAME
1	ABIN
2	BABY
3	SUJIN
4	DEVI

4 rows returned in 0.00

- Consider table tab1 & tab4 with similar structure.
- To copy the records from tab1 to tab4 where rollno=3
- insert into tab4 select * from tab1 where rollno in (select rollno from tab1 where rollno=3)

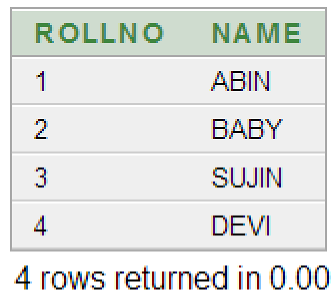


ROLLNO	NAME
1	ABIN
2	BABY
3	SUJIN
4	DEVI
3	SUJIN

5 rows returned in 0.10

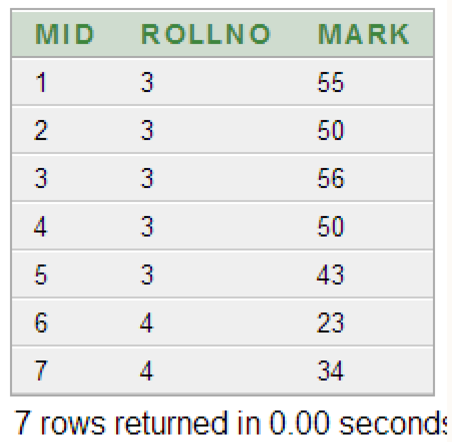
SUBQUERY

- Subqueries in update statement Tab1 Tab2



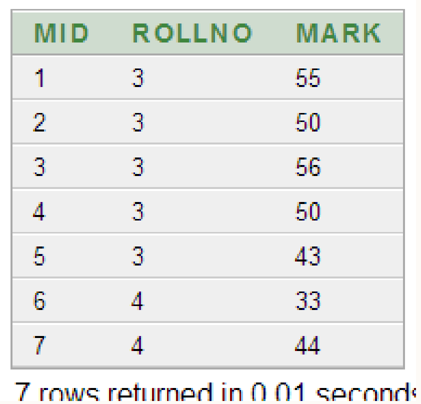
ROLLNO	NAME
1	ABIN
2	BABY
3	SUJIN
4	DEVI

4 rows returned in 0.00



MID	ROLLNO	MARK
1	3	55
2	3	50
3	3	56
4	3	50
5	3	43
6	4	23
7	4	34

7 rows returned in 0.00 seconds



MID	ROLLNO	MARK
1	3	55
2	3	50
3	3	56
4	3	50
5	3	43
6	4	33
7	4	44

7 rows returned in 0.01 seconds

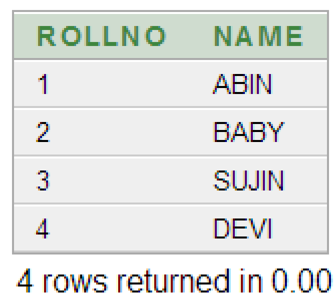
- Increment the marks with 10 in tab2,

  whose rollno = 4 in tab1

- update tab2 set mark=mark+10 where rollno in (select rollno from tab1 where rollno=4)

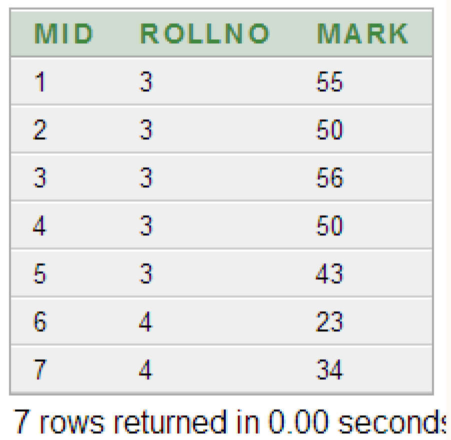
SUBQUERY

- Subqueries in delete statement Tab1 Tab2



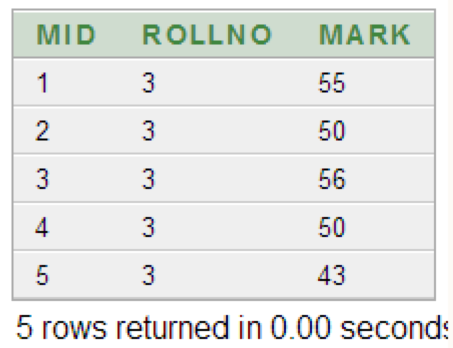
ROLLNO	NAME
1	ABIN
2	BABY
3	SUJIN
4	DEVI

4 rows returned in 0.00



MID	ROLLNO	MARK
1	3	55
2	3	50
3	3	56
4	3	50
5	3	43
6	4	23
7	4	34

7 rows returned in 0.00 seconds



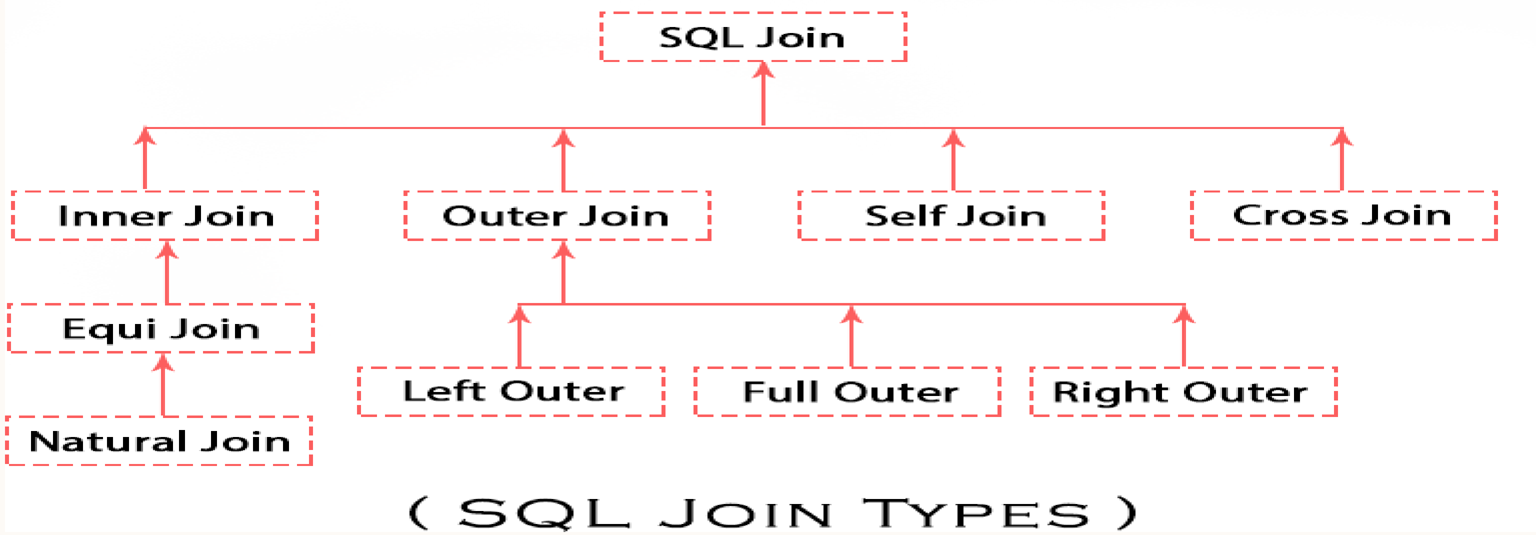
MID	ROLLNO	MARK
1	3	55
2	3	50
3	3	56
4	3	50
5	3	43

5 rows returned in 0.00 seconds

- Delete all details from tab2, where rollno is 4 in tab1
- delete from tab2 where rollno in (select rollno from tab1 where rollno=4)

JOIN CLAUSE

- In Relational Database, JOIN is used to combine columns from one or more tables.
- There must be some common identifiers that allow information from multiple tables to be combined easily.



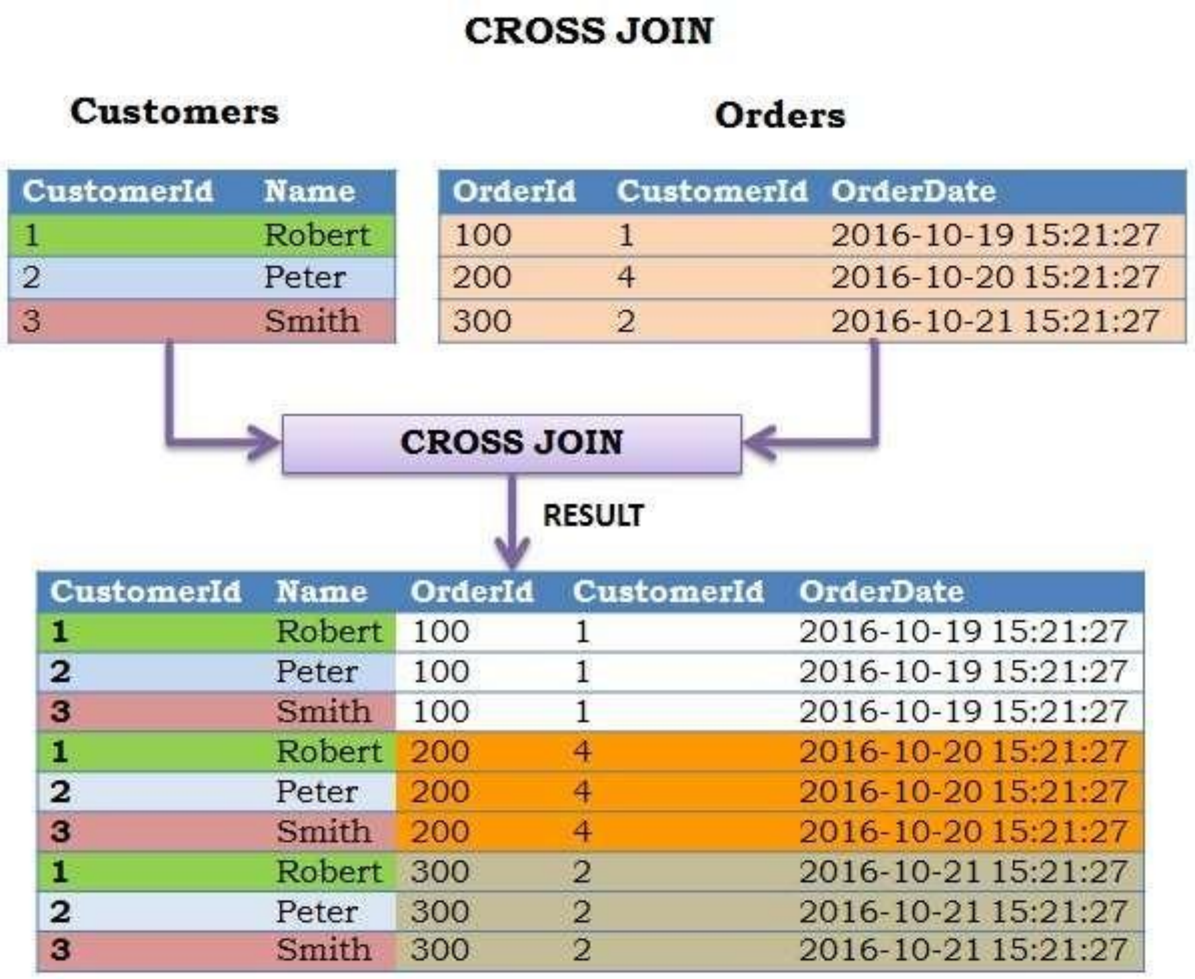
(SQL JOIN TYPES)

JOIN AND ON

- After the FROM statement, we have two new statements: JOIN, which Is followed by a table name, and ON, which is followed by a couple column names separated by an equals sign.

```
SELECT employee.LastName, employee.DepartmentID,
department.DepartmentName
 FROM employee
 JOIN department
ON employee.DepartmentID = department.DepartmentID
```

CROSS JOIN



CROSS JOIN

Customers

CustomerId	Name
1	Robert
2	Peter
3	Smith

Orders

OrderId	CustomerId	OrderDate
100	1	2016-10-19 15:21:27
200	4	2016-10-20 15:21:27
300	2	2016-10-21 15:21:27

CROSS JOIN

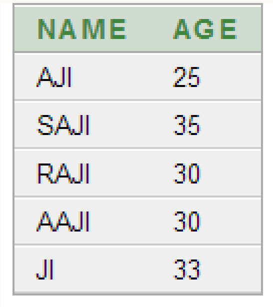
RESULT

CustomerId	Name	OrderId	CustomerId	OrderDate
1	Robert	100	1	2016-10-19 15:21:27
2	Peter	100	1	2016-10-19 15:21:27
3	Smith	100	1	2016-10-19 15:21:27
1	Robert	200	4	2016-10-20 15:21:27
2	Peter	200	4	2016-10-20 15:21:27
3	Smith	200	4	2016-10-20 15:21:27
1	Robert	300	2	2016-10-21 15:21:27
2	Peter	300	2	2016-10-21 15:21:27
3	Smith	300	2	2016-10-21 15:21:27

- CROSS JOIN returns the Cartesian product of rows from tables in the join.

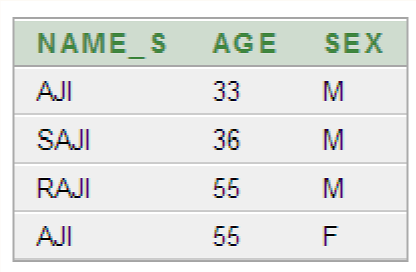
STRUCTURED QUERY LANGUAGE

- JOIN
- Table test1



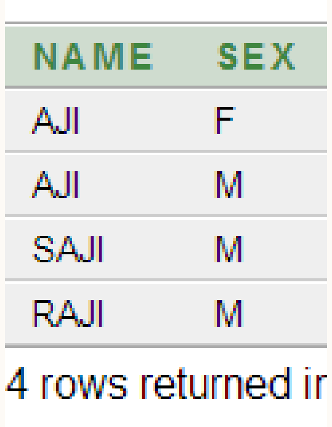
NAME	AGE
AJI	25
SAJI	35
RAJI	30
AAJI	30
JI	33

Table test2



NAME_S	AGE	SEX
AJI	33	M
SAJI	36	M
RAJI	55	M
AJI	55	F

SELECT NAME,SEX FROM TEST1,TEST2 WHERE TEST1.NAME=TEST2.NAME_S;



NAME	SEX
AJI	F
AJI	M
SAJI	M
RAJI	M

4 rows returned ir

STRUCTURED QUERY LANGUAGE

- JOIN
- Need at least one common field and have a relationship between them
- The are two types of SQL JOINS - EQUI JOIN and NON EQUI JOIN

1) SQL EQUI JOIN :

  - The SQL EQUI JOIN is a simple sql join uses the equal sign(=) as the comparison operator for the condition.
  - It has two types - SQL Outer join and SQL Inner join.

2) SQL NON EQUI JOIN :

  - The SQL NON EQUI JOIN is a join uses comparison operator other than the equal sign like >, <, >=, <= with the condition.

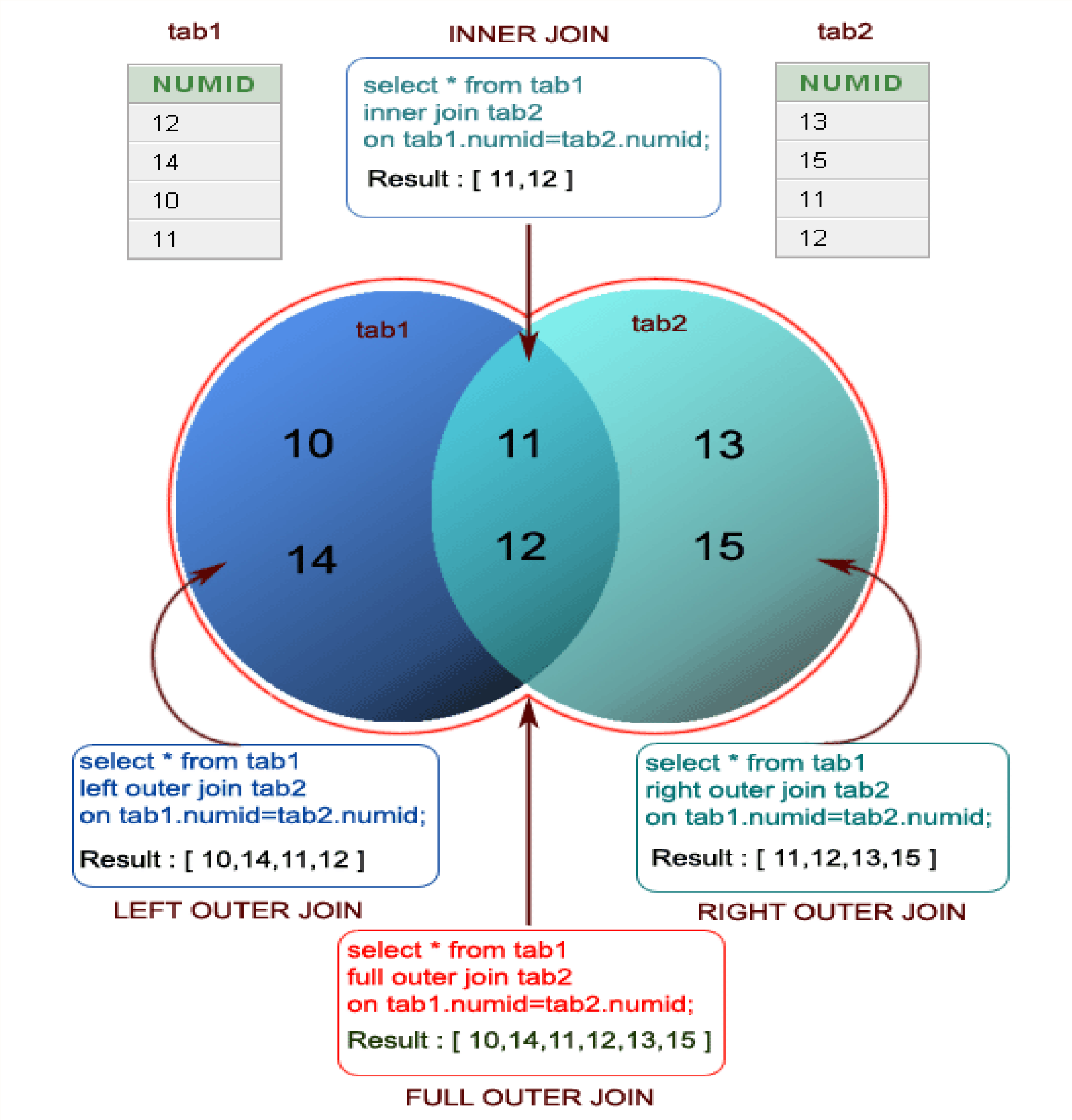
STRUCTURED QUERY LANGUAGE

- SQL EQUI JOIN can be classified into two types –
- INNER JOIN and OUTER JOIN

1. SQL INNER JOIN
    - This type of EQUI JOIN returns all rows from tables where the key record of one table is equal to the key records of another table.
2. SQL OUTER JOIN
    - This type of EQUI JOIN returns all rows from one table and only those rows from the secondary table where the joined condition is satisfying i.e. the columns are equal in both tables.

STRUCTURED QUERY LANGUAGE

- JOIN



tab1

NUMID
12
14
10
11

tab2

NUMID
13
15
11
12

INNER JOIN

```
select * from tab1
inner join tab2
on tab1.numid=tab2.numid;
```

Result : [11,12]

```
select * from tab1
left outer join tab2
on tab1.numid=tab2.numid;
```

Result : [10,14,11,12]

LEFT OUTER JOIN

```
select * from tab1
right outer join tab2
on tab1.numid=tab2.numid;
```

Result : [11,12,13,15]

RIGHT OUTER JOIN

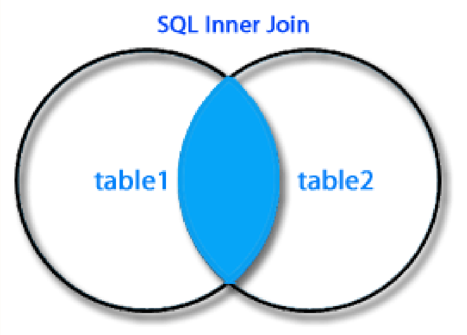
```
select * from tab1
full outer join tab2
on tab1.numid=tab2.numid;
```

Result : [10,14,11,12,13,15]

FULL OUTER JOIN

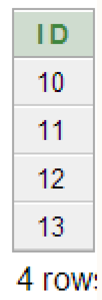
STRUCTURED QUERY LANGUAGE

- INNER JOIN: Returns rows when there is a match in both tables.
- When the join-condition is satisfied, column values for each matched pair of rows of A and B are combined into a result row.
- Syntax:
- SELECT table1.column1, table2.column2... FROM table1 INNER JOIN table2 ON table1.common_field = table2.common_field;



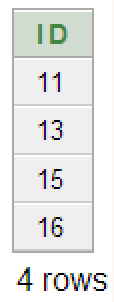
STRUCTURED QUERY LANGUAGE

- TAB1 TAB2



ID
10
11
12
13

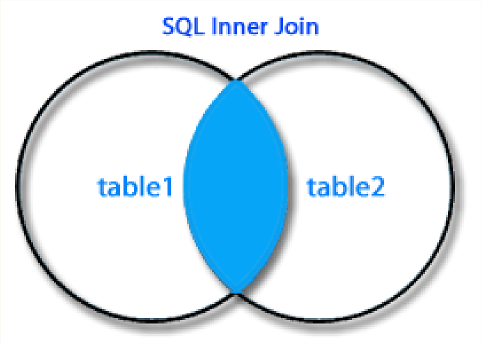
4 row:

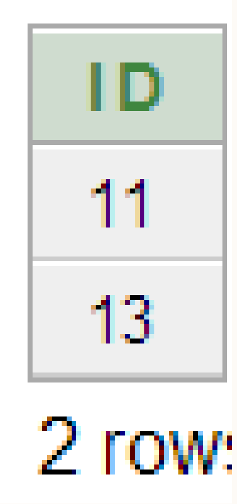


ID
11
13
15
16

4 rows

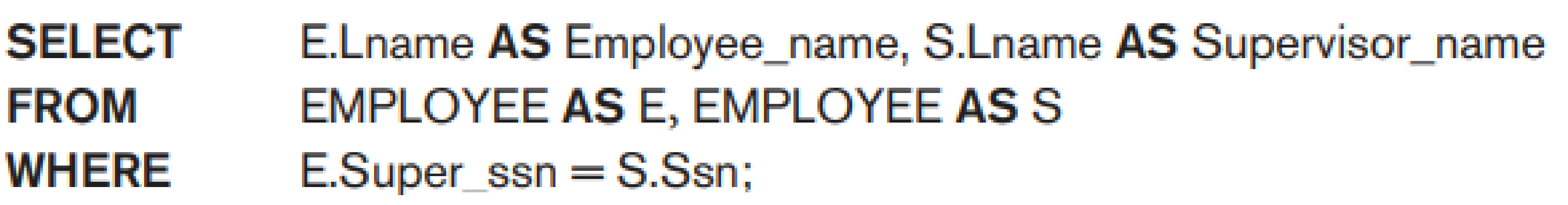
SELECT ID FROM TAB1 INNER JOIN TAB2 ON TAB1.ID=TAB2.SID;





ID
11
13

2 row:

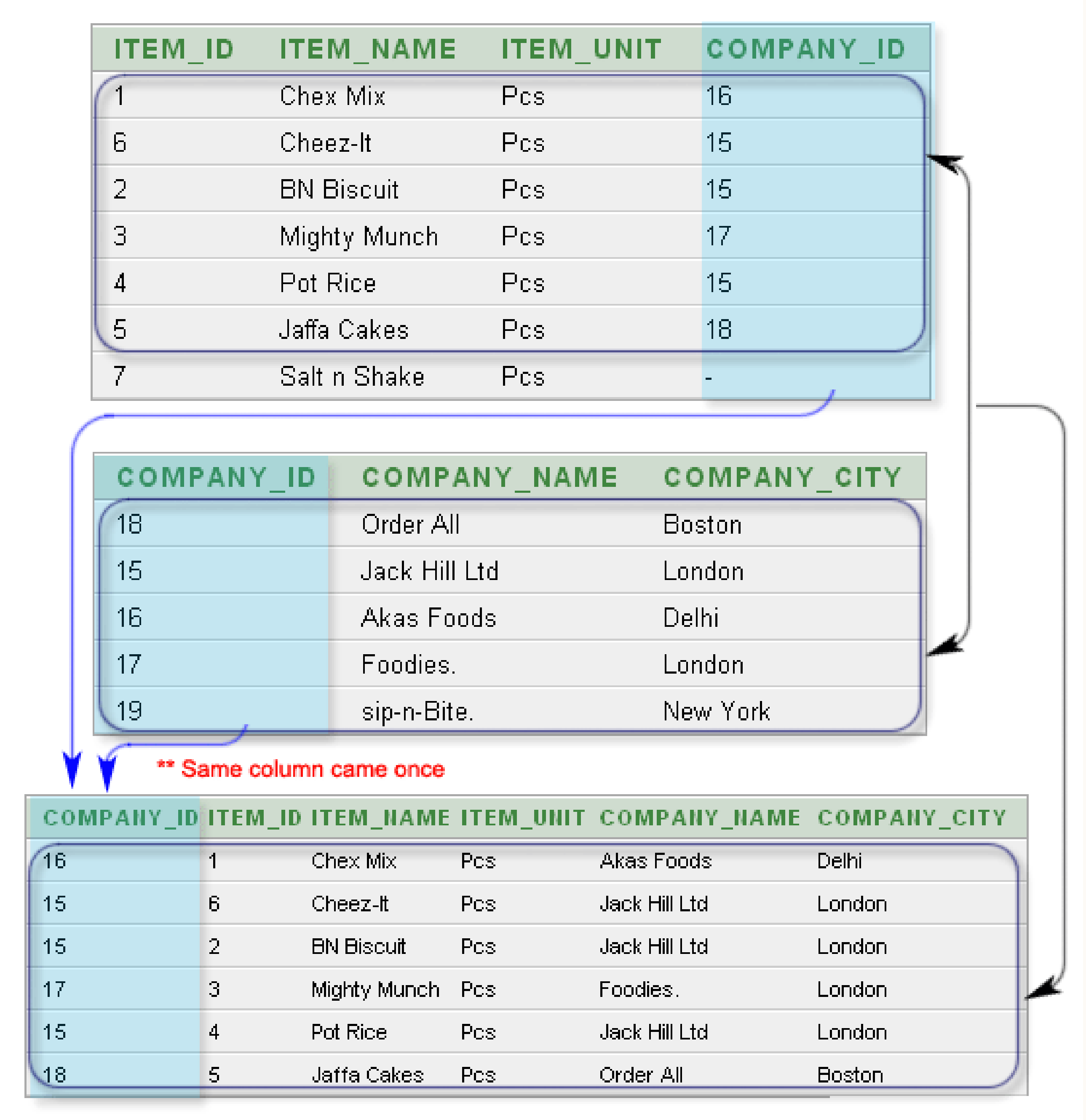


```
SELECT      E.Lname AS Employee_name, S.Lname AS Supervisor_name
FROM        EMPLOYEE AS E, EMPLOYEE AS S
WHERE       E.Super_ssn = S.Ssn;
```

STRUCTURED QUERY LANGUAGE

- NATURAL JOIN
  - Type of EQUI JOIN and is structured in such a way that, columns with same name of associate tables will appear once only
  - The associated tables have one or more pairs of identically named columns.
  - The columns must be the same data type.
  - Don’t use ON clause in a natural join.
- Syntax
  - Select * FROM table1 NATURAL JOIN table2;

- NATURAL JOIN
- SELECT * FROM foods NATURAL JOIN company;



ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	-

COMPANY_ID	COMPANY_NAME	COMPANY_CITY
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

** Same column came once

COMPANY_ID	ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_NAME	COMPANY_CITY
16	1	Chex Mix	Pcs	Akas Foods	Delhi
15	6	Cheez-It	Pcs	Jack Hill Ltd	London
15	2	BN Biscuit	Pcs	Jack Hill Ltd	London
17	3	Mighty Munch	Pcs	Foodies.	London
15	4	Pot Rice	Pcs	Jack Hill Ltd	London
18	5	Jaffa Cakes	Pcs	Order All	Boston

STRUCTURED QUERY LANGUAGE

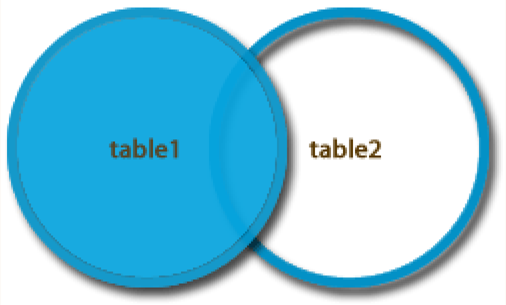
- OUTER JOIN
  - Returns all rows from both the tables which satisfy the join condition along with rows which do not satisfy the join condition.
- They are
  - LEFT OUTER JOIN or LEFT JOIN
  - RIGHT OUTER JOIN or RIGHT JOIN
  - FULL OUTER JOIN

STRUCTURED QUERY LANGUAGE

- LEFT JOIN: Returns all rows from the left table, even if there are no matches in the right table.
- joins two tables and fetches rows based on a condition, which are matching in both the tables, and the unmatched rows will also be available from the table before the JOIN clause
- Syntax:
    - Select * FROM table1 LEFT OUTER JOIN table2 ON table1.column_name=table2.column_name;

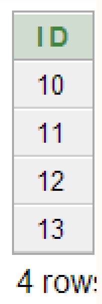
STRUCTURED QUERY LANGUAGE

- LEFT JOIN:
  - SQL LEFT join fetches a complete set of records from table1, with the matching records (depending upon the availability) in table2.
  - The result is NULL in the right side when no matching will take place.



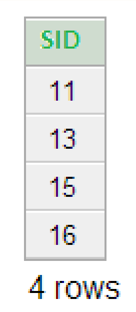
STRUCTURED QUERY LANGUAGE

- LEFT JOIN
- TAB1 TAB2



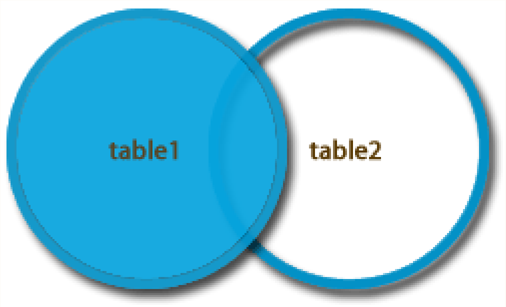
ID
10
11
12
13

4 row:

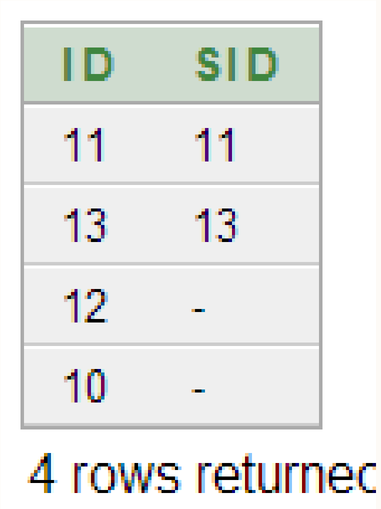


SID
11
13
15
16

4 rows



SELECT * FROM TAB1 LEFT OUTER JOIN TAB2 ON TAB1.ID=TAB2.SID



ID	SID
11	11
13	13
12	-
10	-

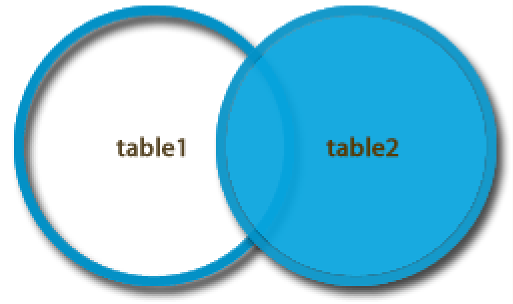
4 rows returnec

STRUCTURED QUERY LANGUAGE

- RIGHT JOIN
- joins two tables and fetches rows based on a condition, which are matching in both the tables, and the unmatched rows will also be available from the table written after the JOIN clause
- Syntax
    - Select * FROM table1 RIGHT OUTER JOIN table2 ON table1.column_name=table2.column_name ;

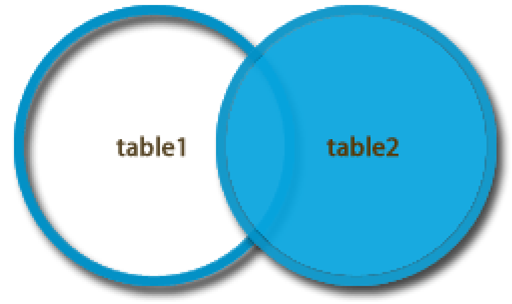
STRUCTURED QUERY LANGUAGE

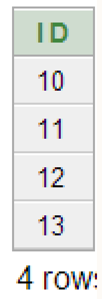
- RIGHT
- join fetches a complete set of records from table2, i.e. the rightmost table after JOIN clause, with the matching records (depending upon the availability) in table1.
- The result is NULL in the left side when no matching will take place.



STRUCTURED QUERY LANGUAGE

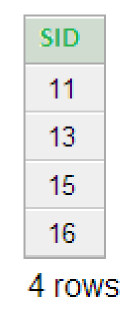
- RIGHT JOIN
- TAB1 TAB2





ID
10
11
12
13

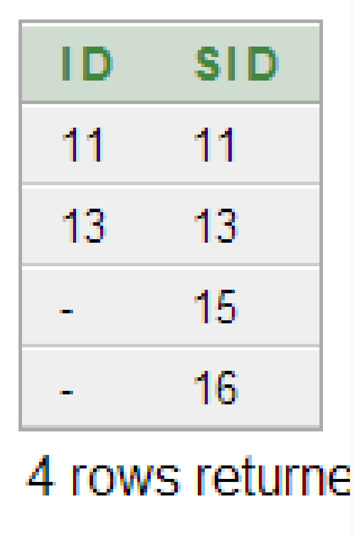
4 row:



SID
11
13
15
16

4 rows

SELECT * FROM TAB1 RIGHT OUTER JOIN TAB2 ON TAB1.ID=TAB2.SID

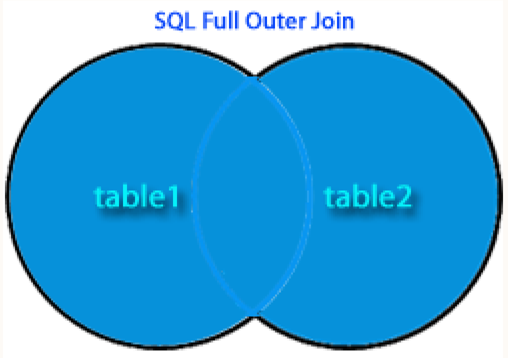


ID	SID
11	11
13	13
-	15
-	16

4 rows returne

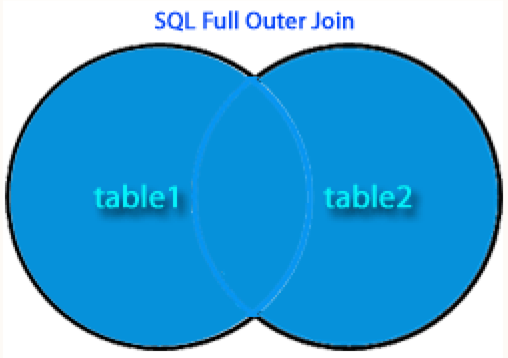
STRUCTURED QUERY LANGUAGE

- FULL OUTER JOIN
- combines the results of both left and right outer joins and returns all (matched or unmatched) rows from the tables on both sides of the join clause.
- Syntax
- SELECT * FROM table1 FULL OUTER JOIN table2 ON table1.column_name=table2.column_name ;

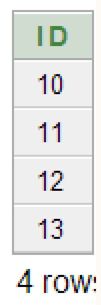


STRUCTURED QUERY LANGUAGE

- FULL OUTER JOIN
- TAB1 TAB2



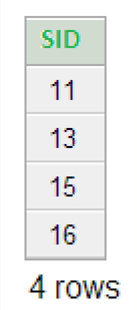
TAB1



ID
10
11
12
13

4 rows

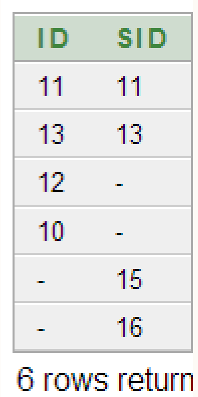
TAB2



SID
11
13
15
16

4 rows

SELECT * FROM TAB1 FULL OUTER JOIN TAB2 ON TAB1.ID=TAB2.SID



ID	SID
11	11
13	13
12	-
10	-
-	15
-	16

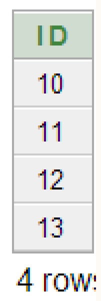
6 rows return

STRUCTURED QUERY LANGUAGE

- A SELF JOIN
  - which is used to join a table to itself
  - In this join, the table appears twice after the FROM clause and is followed by aliases for the tables that qualify column names in the join condition
  - In this join those rows are returned from the table which are satisfying the conditions.

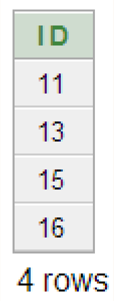
STRUCTURED QUERY LANGUAGE

- SELF JOIN
- TAB1 TAB2



ID
10
11
12
13

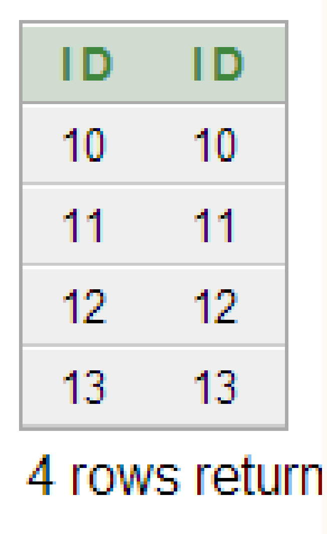
4 row:



ID
11
13
15
16

4 rows

SELECT A.ID, B.ID FROM TAB1 A,TAB1 B WHERE A.ID=B.ID;



ID	ID
10	10
11	11
12	12
13	13

4 rows return

NESTED JOIN

Write a query which identify the full name of each employee (concatenated FIRST_NAME and LAST_NAME)., The name of the department the employee belongs to. The city where the department is located.

```
SELECT E.FIRST_NAME || ' ' || E.LAST_NAME AS Employee_Name,
       D.DEPARTMENT_NAME AS Department_Name,
       L.CITY AS Location_City
FROM EMPLOYEES E
JOIN DEPARTMENTS D ON E.DEPARTMENT_ID = D.DEPARTMENT_ID
JOIN LOCATIONS L ON D.LOCATION_ID = L.LOCATION_ID;
```

- Retrieve Employee Name, Job Title, and Department Name

```
SELECT E.FIRST_NAME || ' ' || E.LAST_NAME AS Employee_Name,
       J.JOB_TITLE AS Job_Title,
       D.DEPARTMENT_NAME AS Department_Name
FROM EMPLOYEES E
JOIN JOBS J ON E.JOB_ID = J.JOB_ID
JOIN DEPARTMENTS D ON E.DEPARTMENT_ID = D.DEPARTMENT_ID;
```

- Retrieve Department Name, Manager Name, and Location Country

```
SELECT D.DEPARTMENT_NAME AS Department_Name,
M.FIRST_NAME || ' ' || M.LAST_NAME AS Manager_Name,
C.COUNTRY_NAME AS CountryFROM DEPARTMENTS DJOIN
EMPLOYEES M ON D.MANAGER_ID = M.EMPLOYEE_IDJOIN
LOCATIONS L ON D.LOCATION_ID = L.LOCATION_IDJOIN COUNTRIES
C ON L.COUNTRY_ID = C.COUNTRY_ID;
```

- Retrieve Employee, Department, and Region Name

```
SELECT E.FIRST_NAME || ' ' || E.LAST_NAME AS Employee_Name,
D.DEPARTMENT_NAME AS Department_Name,       R.REGION_NAME AS
RegionFROM EMPLOYEES EJOIN DEPARTMENTS D ON
E.DEPARTMENT_ID = D.DEPARTMENT_IDJOIN LOCATIONS L ON
D.LOCATION_ID = L.LOCATION_IDJOIN COUNTRIES C ON
L.COUNTRY_ID = C.COUNTRY_IDJOIN REGIONS R ON C.REGION_ID =
R.REGION_ID;ccc
```

AGGREGATE FUNCTIONS

- Aggregate functions are used to summarize information from multiple tuples into a single-tuple summary.
- Grouping is used to create subgroups of tuples before summarization.
- The functions SUM, MAX, MIN, and AVG can be applied to a set or multiset of numeric values and return, respectively, the sum, maximum value, minimum value, and average (mean) of those values.
- A number of built-in aggregate functions exist:
  - **Count**
  - **Sum**
  - **Max**
  - **Min**

```
SELECT SUM (SALARY), MAX (SALARY), MIN (SALARY), AVG (SALARY)
FROM EMPLOYEES;
```

COUNT FUNCTION

- The **COUNT** function returns the number of tuples or values as specified in a query.
- Count the number of employees in each department, but show only departments with more than 5 employees.

```
SELECT Department_Name, Employee_Count
FROM (
    SELECT D.DEPARTMENT_NAME, COUNT(E.EMPLOYEE_ID) AS Employee_Count
    FROM DEPARTMENTS D
    JOIN EMPLOYEES E ON D.DEPARTMENT_ID = E.DEPARTMENT_ID
    GROUP BY D.DEPARTMENT_NAME
)
WHERE Employee_Count > 5;
```

- Retrieve the total number of employees in the company and the number of employees in the ‘Research’ department

```
SELECT COUNT (*) FROM EMPLOYEE, DEPARTMENT WHERE DNO =
DNUMBER AND DNAME = ‘Research’;
```

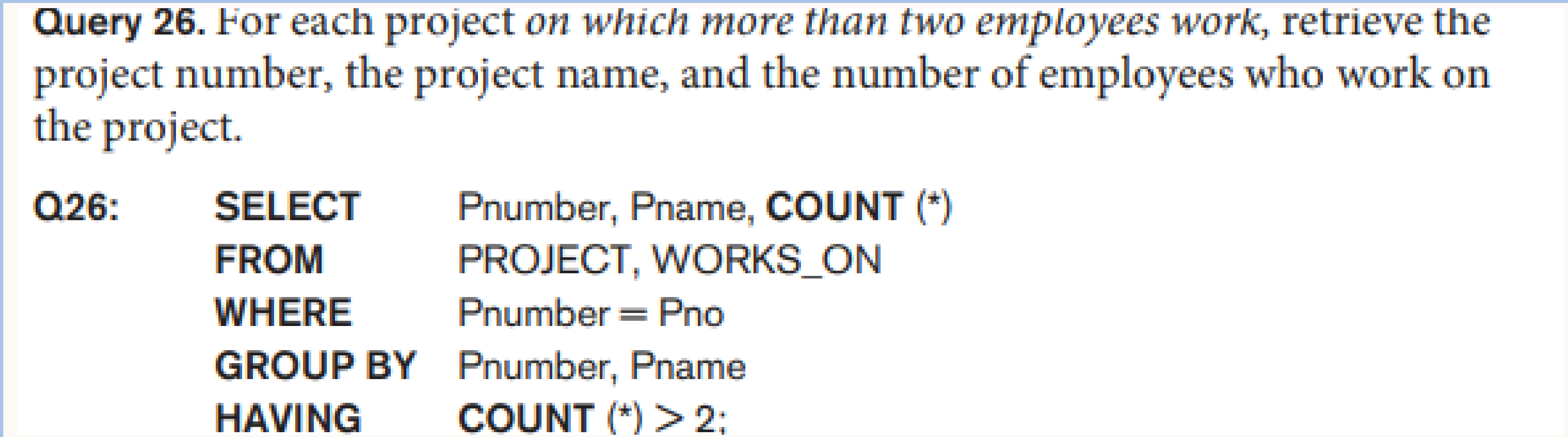
GROUP BY

- In many cases we want to apply the aggregate functions to subgroups of tuples in a relation, where the subgroups are based on some attribute values.
- Each group (partition) will consist of the tuples that have the same value of some attribute(s), called the **grouping attribute(s).**
- The GROUP BY clause specifies the grouping attributes, which should also appear in the SELECT clause, so that the value resulting from applying each aggregate function to a group of tuples appears along with the value of the grouping attribute(s).

```
SELECT Dno, COUNT (*), AVG (Salary) FROM EMPLOYEE GROUP BY Dno;
```

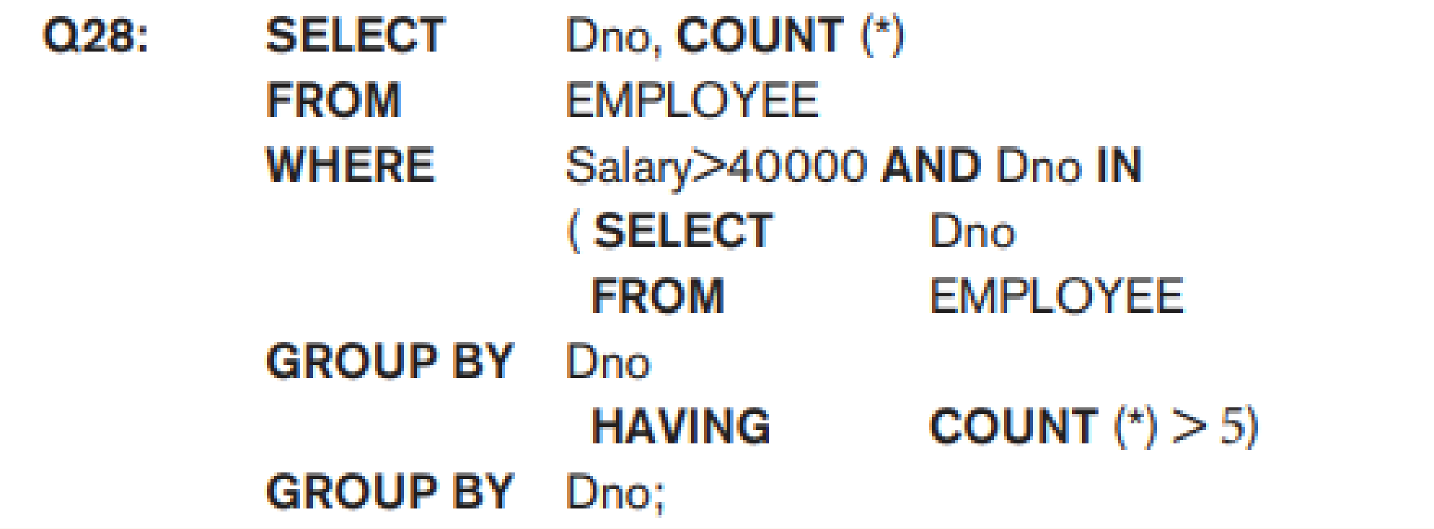
HAVING CLAUSE

- Sometimes we want to retrieve the values of these functions only for groups that satisfy certain conditions.
- SQL provides a HAVING clause, which can appear in conjunction with a GROUP BY clause, for this purpose.
- HAVING provides a condition on the summary information regarding the group of tuples associated with each value of the grouping attributes. Only the groups that satisfy the condition are retrieved in the result of the query.



Query 26. For each project *on which more than two employees work*, retrieve the project number, the project name, and the number of employees who work on the project.

```
Q26:    SELECT      Pnumber, Pname, COUNT (*)
        FROM        PROJECT, WORKS_ON
        WHERE       Pnumber = Pno
        GROUP BY    Pnumber, Pname
        HAVING      COUNT (*) > 2;
```



```
Q28:    SELECT      Dno, COUNT (*)
        FROM        EMPLOYEE
        WHERE       Salary>40000 AND Dno IN
                    ( SELECT        Dno
                      FROM          EMPLOYEE
        GROUP BY    Dno
                      HAVING        COUNT (*) > 5)
        GROUP BY    Dno;
```

EXAMPLES

- employees: Contains employee details.
- departments: Contains department details.
- projects: Contains project details.

```
CREATE TABLE employees (
employee_id INT PRIMARY KEY,
first_name VARCHAR2(50),
last_name VARCHAR2(50),
department_id INT,
salary DECIMAL(10, 2)
);
```

```
CREATE TABLE departments (
    department_id INT PRIMARY  KEY,
   department_name  VARCHAR2(50)
);
```

```
CREATE TABLE projects (
    project_id INT PRIMARY KEY,
    project_name VARCHAR2(50),
    department_id INT
);
```

AVERAGE SALARY BY DEPARTMENT

```
SELECT d.department_name, AVG(e.salary) AS
average_salary FROM employees e JOIN departments d
ON e.department_id = d.department_id GROUP BY
d.department_name;
```

TOTAL SALARY PER DEPARTMENT WITH MORE THAN ONE EMPLOYEE

```
SELECT d.department_name, SUM(e.salary) AS
total_salary FROM employees e JOIN
departments d ON e.department_id =
d.department_id GROUP BY d.department_name
HAVING COUNT(e.employee_id) > 1;
```

HIGHEST SALARY IN EACH DEPARTMENT

```
SELECT d.department_name, MAX(e.salary) AS
highest_salary FROM employees e JOIN departments d
ON e.department_id = d.department_id GROUP BY
d.department_name;
```

TOTAL NUMBER OF PROJECTS AND AVERAGE SALARY BY DEPARTMENT

```
SELECT d.department_name, COUNT(p.project_id) AS
total_projects, AVG(e.salary) AS average_salary FROM
departments d LEFT JOIN projects p ON
d.department_id = p.department_id LEFT JOIN
employees e ON d.department_id = e.department_id
GROUP BY d.department_name;
```

DEPARTMENTS WITH MORE THAN ONE PROJECT AND AVERAGE EMPLOYEE SALARY

```
SELECT d.department_name, AVG(e.salary) AS
average_salary FROM departments d JOIN projects p
ON d.department_id = p.department_id LEFT JOIN
employees e ON d.department_id = e.department_id
GROUP BY d.department_name HAVING
COUNT(p.project_id) > 1;
```