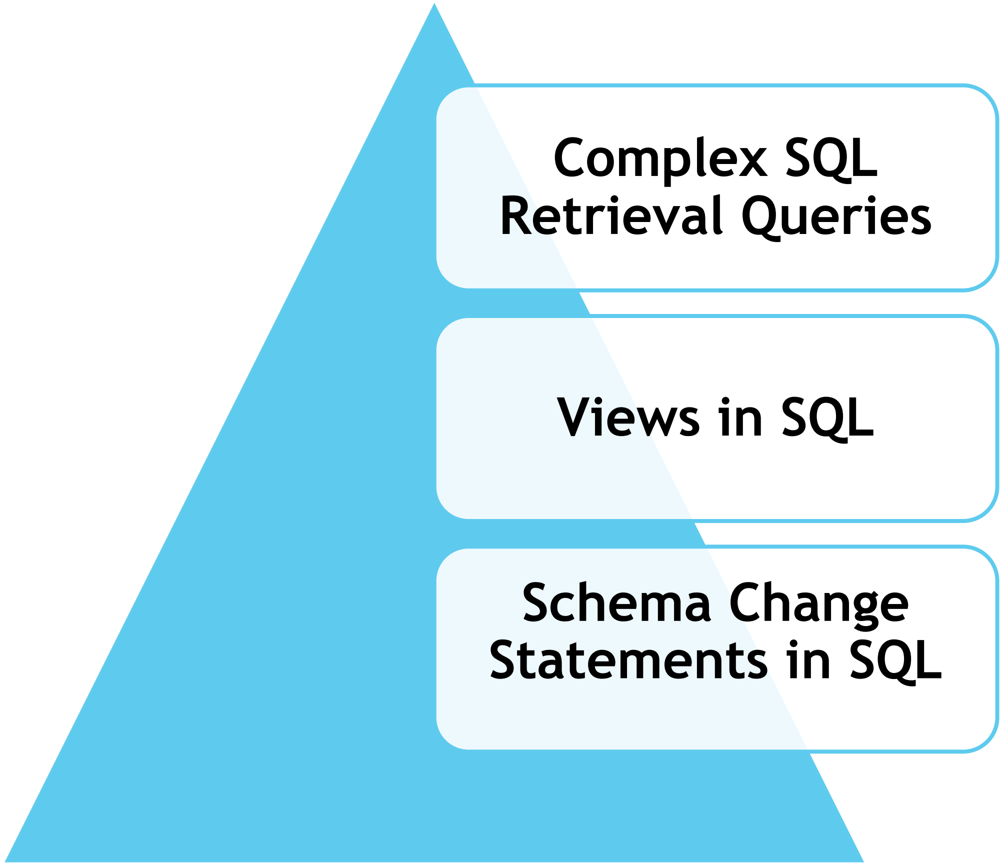
Chapter 7
More SQL: Complex Queries

Content



Complex SQL Retrieval Queries

Views in SQL

Schema Change Statements in SQL

More Complex SQL Retrieval Queries

- ***Comparisons Involving NULL and Three-Valued Logic***
- NULL is used to represent a **missing value**, but it usually has one of three different interpretations:
  - 1. Value unknown (value exists but is not known, or it is not known whether or not the value exists)
  - 2. Value not available (value exists but is purposely withheld),
  - 3. Value not applicable (the attribute does not apply to this tuple or is undefined for this tuple).

Unknown value

- A person’s date of birth is not known, so it is represented by NULL in the database.
- An example of the other case of unknown would be NULL for a person’s home phone because it is not known whether or not the person has a home phone.

Not applicable attribute

- An attribute Last College Degree would be NULL for a person who has no college degrees because it does not apply to that person.

Unavailable or withheld value

- A person has a home phone but does not want it to be listed, so it is withheld and represented as NULL in the database.

More Complex SQL Retrieval Queries

- ***Comparisons Involving NULL and Three-Valued Logic***
- When a record with NULL in one of its attributes is involved in a comparison operation, the result is considered to be **UNKNOWN** (it may be TRUE or it may be FALSE).
- SQL uses a three-valued logic with values TRUE, FALSE, and UNKNOWN instead of the standard two-valued (Boolean) logic with values TRUE or FALSE.
- It is therefore necessary to define the results (or truth values) of three-valued logical expressions when the logical connectives AND, OR, and NOT are used.

More Complex SQL Retrieval Queries

- ***Comparisons Involving NULL and Three-Valued Logic***

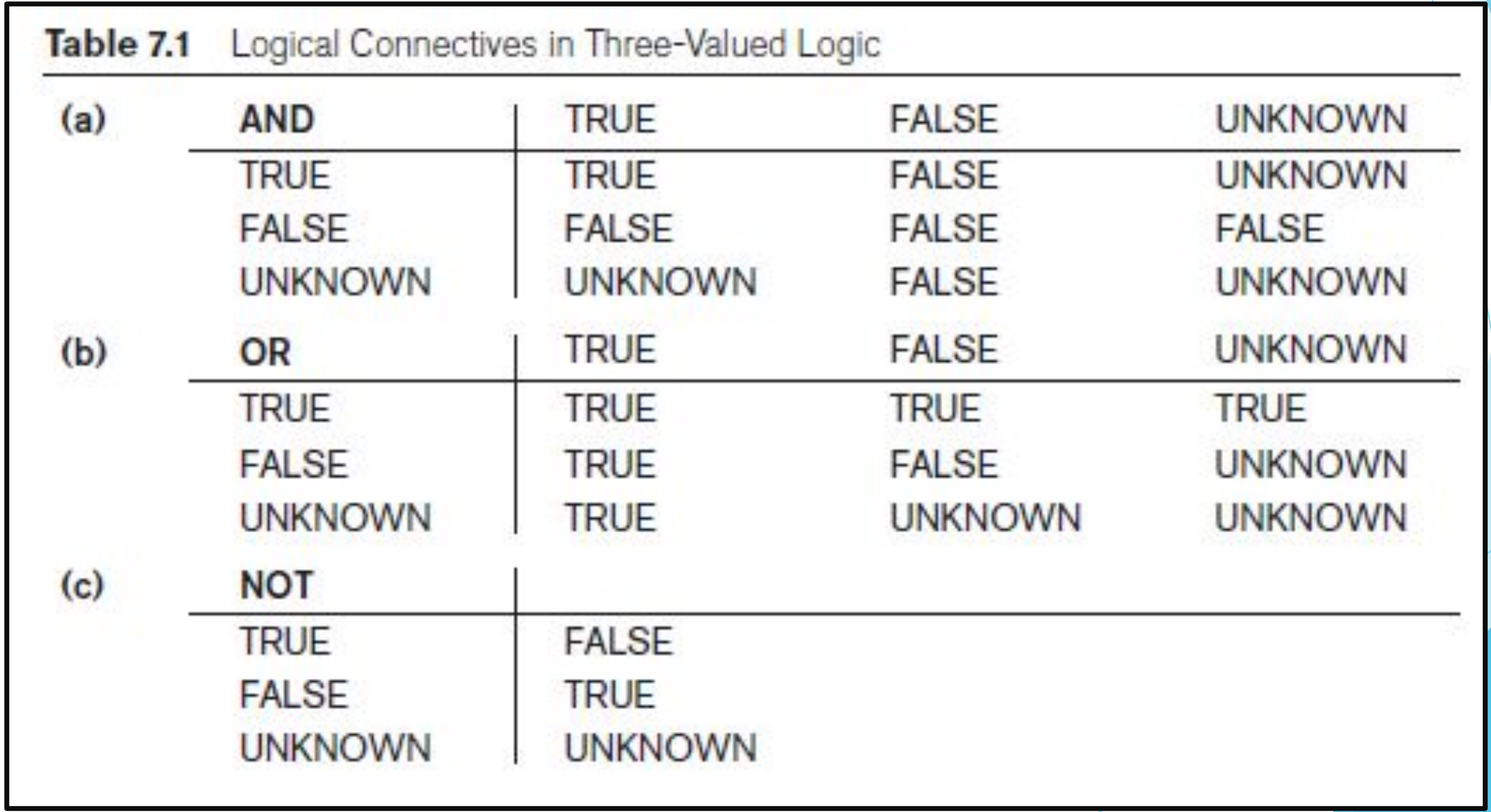


Table 7.1 Logical Connectives in Three-Valued Logic

(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

More Complex SQL Retrieval Queries

- ***Comparisons Involving NULL and Three-Valued Logic***
- SQL allows queries that check whether an attribute value is **NULL**.
- Rather than using = or <> to compare an attribute value to NULL, SQL uses the comparison operators ***"IS" or "IS NOT".***
- This is because SQL considers each NULL value as being distinct from every other NULL value, so equality comparison is not appropriate.

More Complex SQL Retrieval Queries

- ***Comparisons Involving NULL and Three-Valued Logic***
- Query 18. Retrieve the names of all employees who do not have supervisors.
- Q18: SELECT Fname, Lname

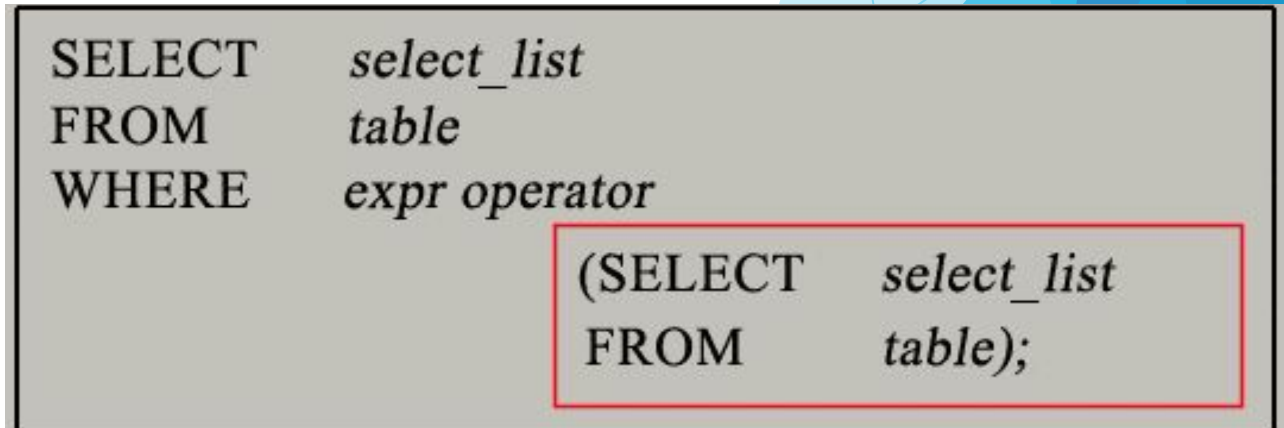
```
FROM EMPLOYEE
WHERE Super_ssn IS NULL;
```

More Complex SQL Retrieval Queries

- ***Subqueries***
- A **Subquery or Inner query or a Nested query** is a query within another SQL query and embedded within the WHERE clause.
- Subqueries can be used with the **SELECT, FROM, WHERE, INSERT, UPDATE, and DELETE** statements
- They are used along with the operators like **=, <, >, >=, <=, IN, ANY, SOME etc.**
- Subqueries must be enclosed within parentheses.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.

More Complex SQL Retrieval Queries

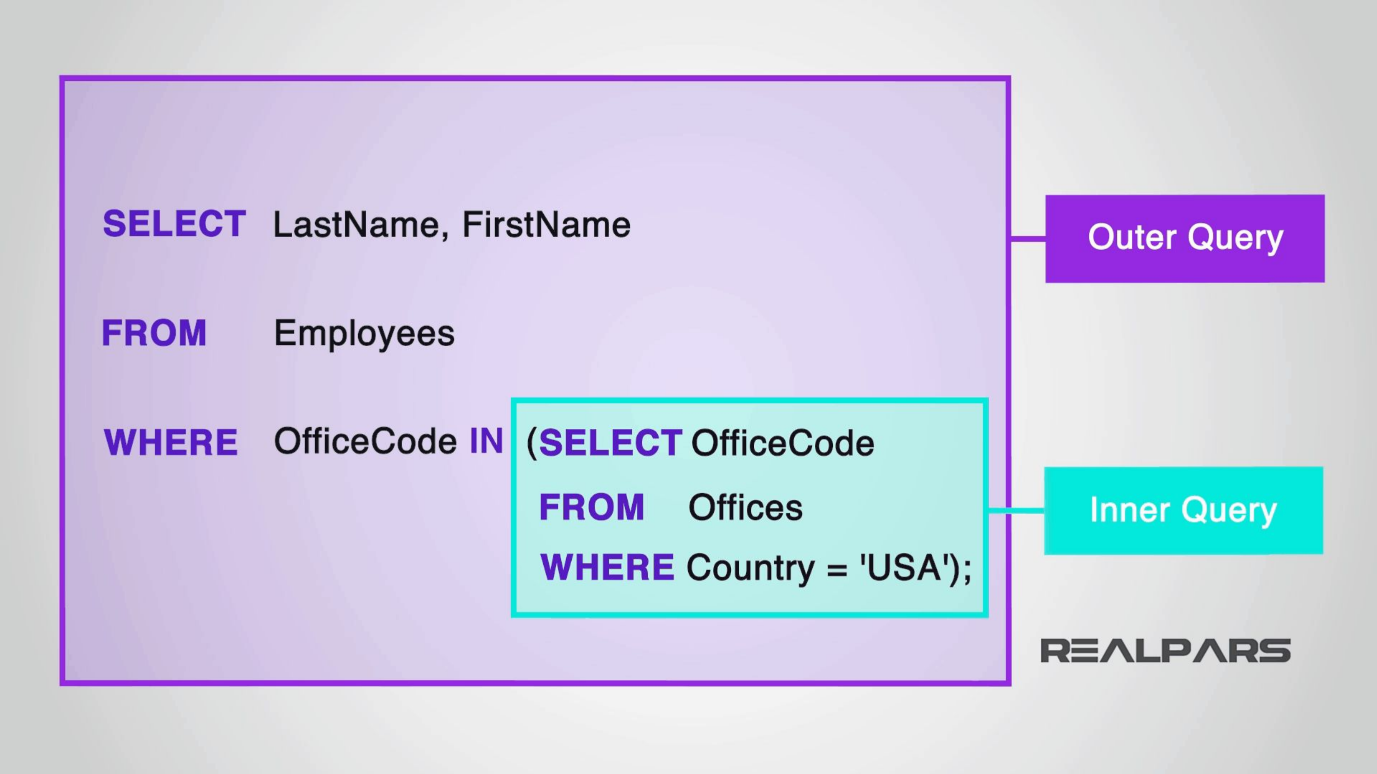
- ***Subqueries***
- Why use subqueries?
  - Compare an expression to the result of the query.
  - Determine if an expression is included in the results of the query.
- Execution follows Bottom – Top Approach i.e. inner query executes first.
- The main query (outer query) use the subquery result.



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

More Complex SQL Retrieval Queries

- ***Subqueries***



```
SELECT  LastName, FirstName
FROM    Employees
WHERE   OfficeCode IN (SELECT OfficeCode
                        FROM   Offices
                        WHERE Country = 'USA');
```

More Complex SQL Retrieval Queries

- ***Subqueries***

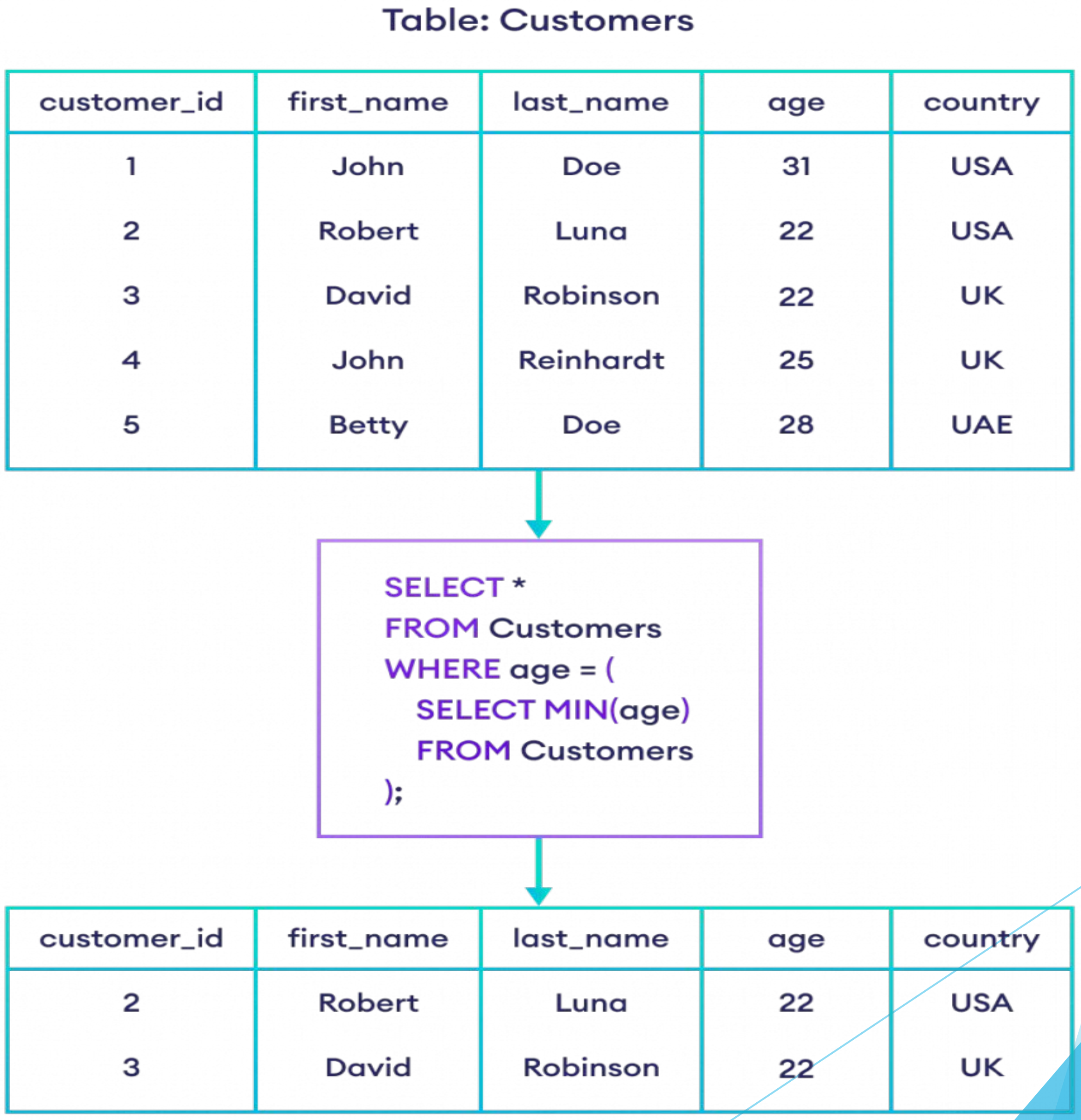


Table: Customers

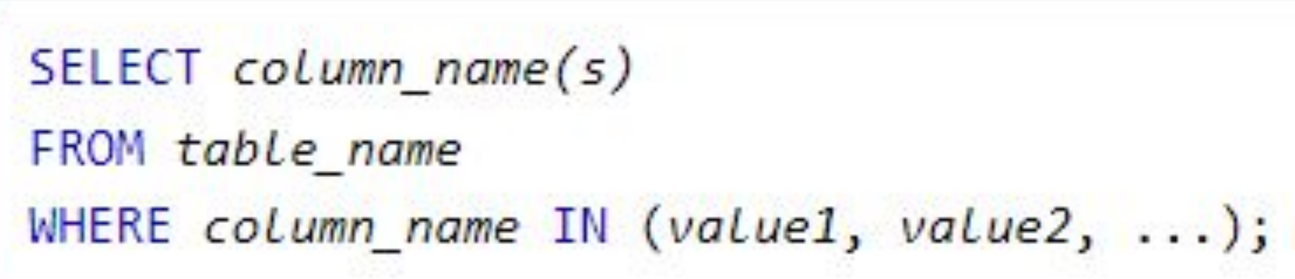
customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
WHERE age = (
  SELECT MIN(age)
  FROM Customers
);
```

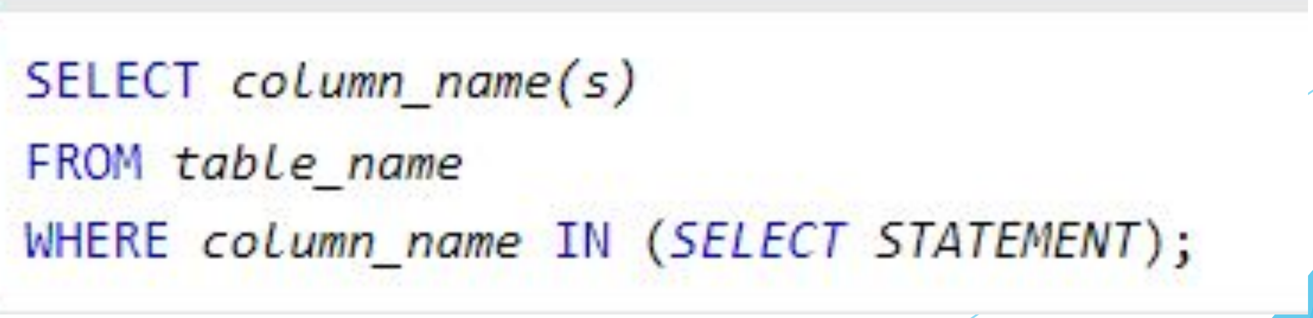
customer_id	first_name	last_name	age	country
2	Robert	Luna	22	USA
3	David	Robinson	22	UK

More Complex SQL Retrieval Queries

- *Nested Queries, Tuples, and Set/Multiset Comparisons*
- IN Operator (comparison operator)
- which **compares a value v1 with a set (or multiset) of values V and evaluates to TRUE if v1 is one of the elements in V.**
- The IN operator allows you to specify **multiple values** in a WHERE clause.
- The IN operator is a shorthand for multiple **OR** conditions.



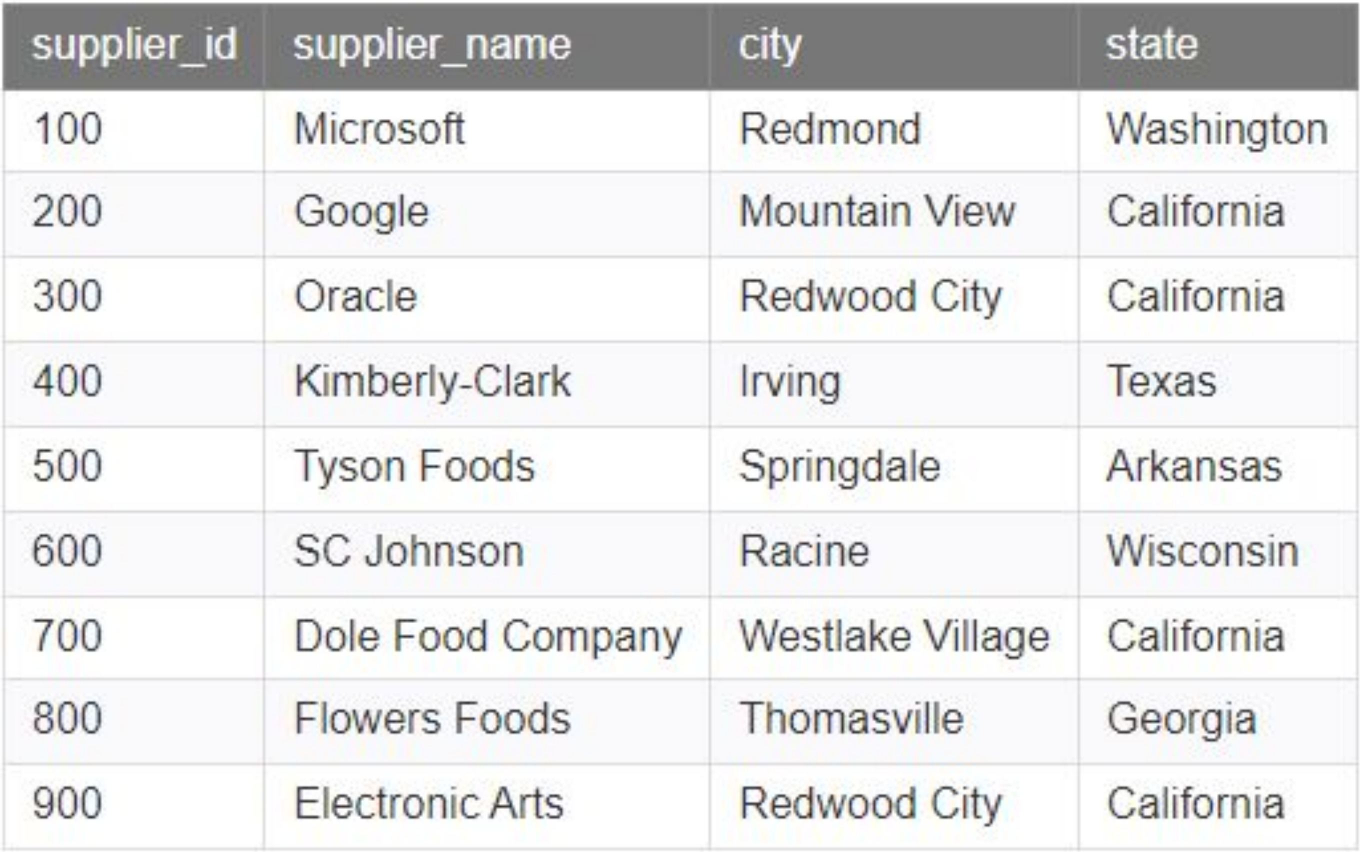
```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```



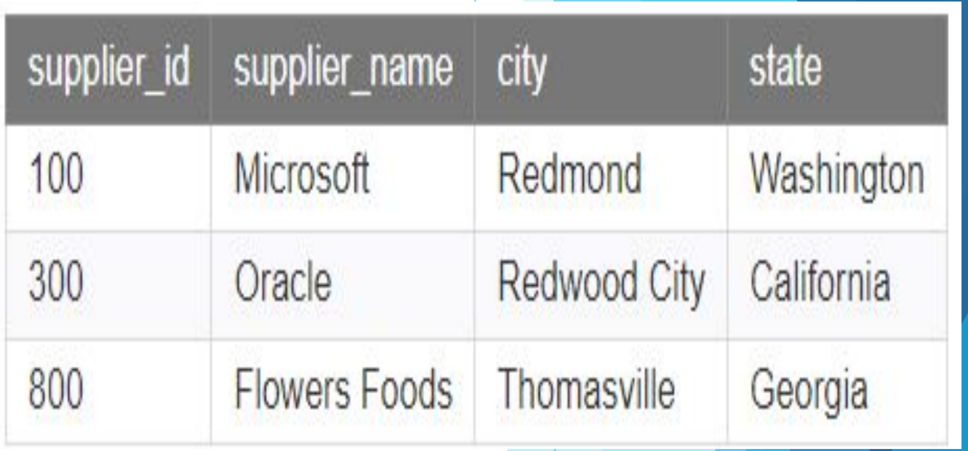
```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

More Complex SQL Retrieval Queries

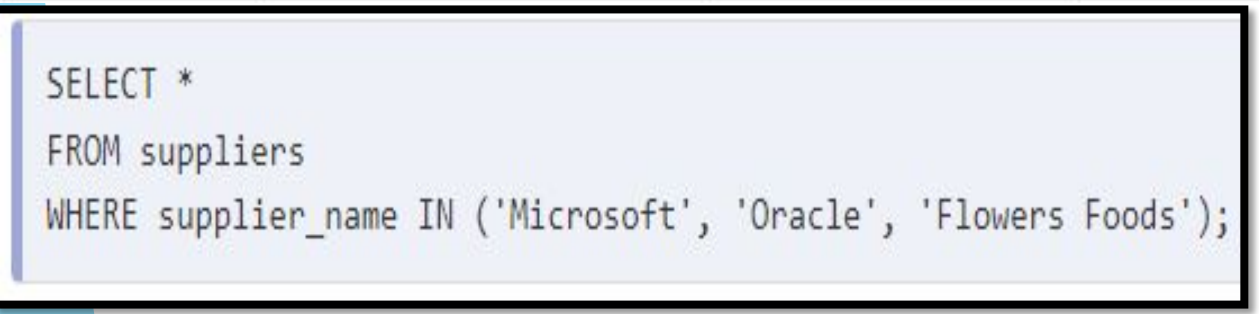
- ***Nested Queries, Tuples, and Set/Multiset Comparisons***
- **IN Operator**



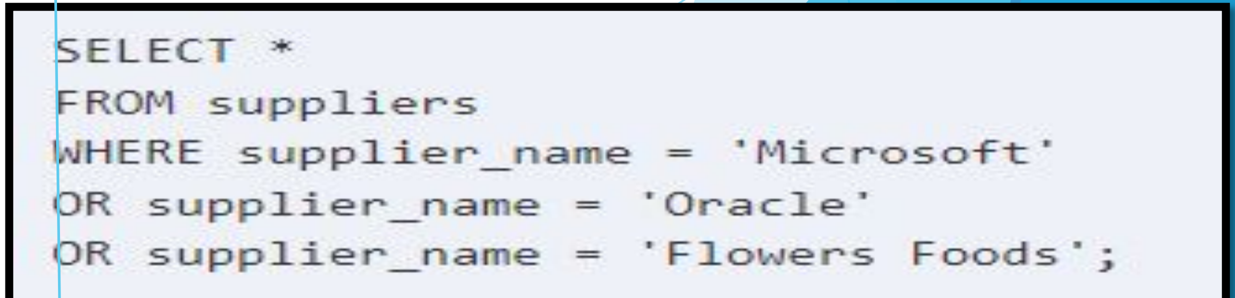
supplier_id	supplier_name	city	state
100	Microsoft	Redmond	Washington
200	Google	Mountain View	California
300	Oracle	Redwood City	California
400	Kimberly-Clark	Irving	Texas
500	Tyson Foods	Springdale	Arkansas
600	SC Johnson	Racine	Wisconsin
700	Dole Food Company	Westlake Village	California
800	Flowers Foods	Thomasville	Georgia
900	Electronic Arts	Redwood City	California



supplier_id	supplier_name	city	state
100	Microsoft	Redmond	Washington
300	Oracle	Redwood City	California
800	Flowers Foods	Thomasville	Georgia



```
SELECT *
FROM suppliers
WHERE supplier_name IN ('Microsoft', 'Oracle', 'Flowers Foods');
```



```
SELECT *
FROM suppliers
WHERE supplier_name = 'Microsoft'
OR supplier_name = 'Oracle'
OR supplier_name = 'Flowers Foods';
```

More Complex SQL Retrieval Queries

- ***Nested Queries, Tuples, and Set/Multiset Comparisons***
- **IN Operator**
- Query: Find the name of employees who are working on a project.
- _So, first it'll fetch employees who are working on a project
- Select Eid From Project

Employee Table

Eid	Ename	Address
1	Brown	New York
2	Sam	Houston
3	Smith	Seattle
4	John	Denver
5	Mary	San Francisco

Project Table

Eid	Pid	Pname	Location
1	P1	IoT	Ohio
5	P2	AI	Los Angeles
3	P3	Cloud	Chicago
4	P4	Android	Texas

More Complex SQL Retrieval Queries

- ***Nested Queries, Tuples, and Set/Multiset Comparisons***
- **IN Operator**
- Query: Find the name of employees who are working on a project.
- **Select Ename**

  From Employee

  Where Eid IN (Select Eid From Project)

Compare Eid from Employee table with Project Table. Check if that Eid exists IN Project Table.
Comparison is done one by one for the complete table.

Employee Table

Eid	Ename	Address
1	Brown	New York
2	Sam	Houston
3	Smith	Seattle
4	John	Denver
5	Mary	San Francisco

Project Table

Eid	Pid	Pname	Location
1	P1	IoT	Ohio
5	P2	AI	Los Angeles
3	P3	Cloud	Chicago
4	P4	Android	Texas

More Complex SQL Retrieval Queries

- ***Nested Queries, Tuples, and Set/Multiset Comparisons***
- NOT IN Operator
- Query: Find the name of employees **who are not working on a project.**
- Select Ename

  From Employee

  Where Eid NOT IN (Select Eid From Project)

Compare Eid from Employee table with Project Table. Check if that Eid exists IN Project Table.

Comparison is done one by one for the complete table.

Employee Table

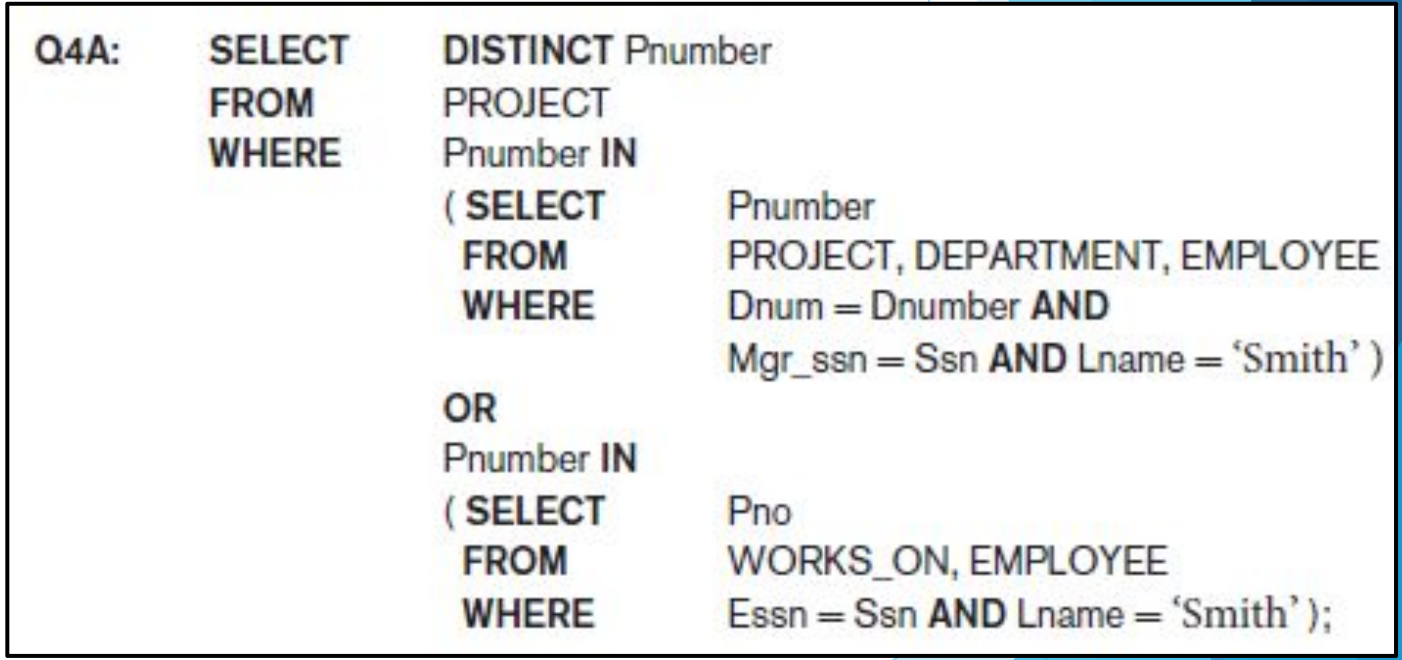
Eid	Ename	Address
1	Brown	New York
2	Sam	Houston
3	Smith	Seattle
4	John	Denver
5	Mary	San Francisco

Project Table

Eid	Pid	Pname	Location
1	P1	IoT	Ohio
5	P2	AI	Los Angeles
3	P3	Cloud	Chicago
4	P4	Android	Texas

More Complex SQL Retrieval Queries

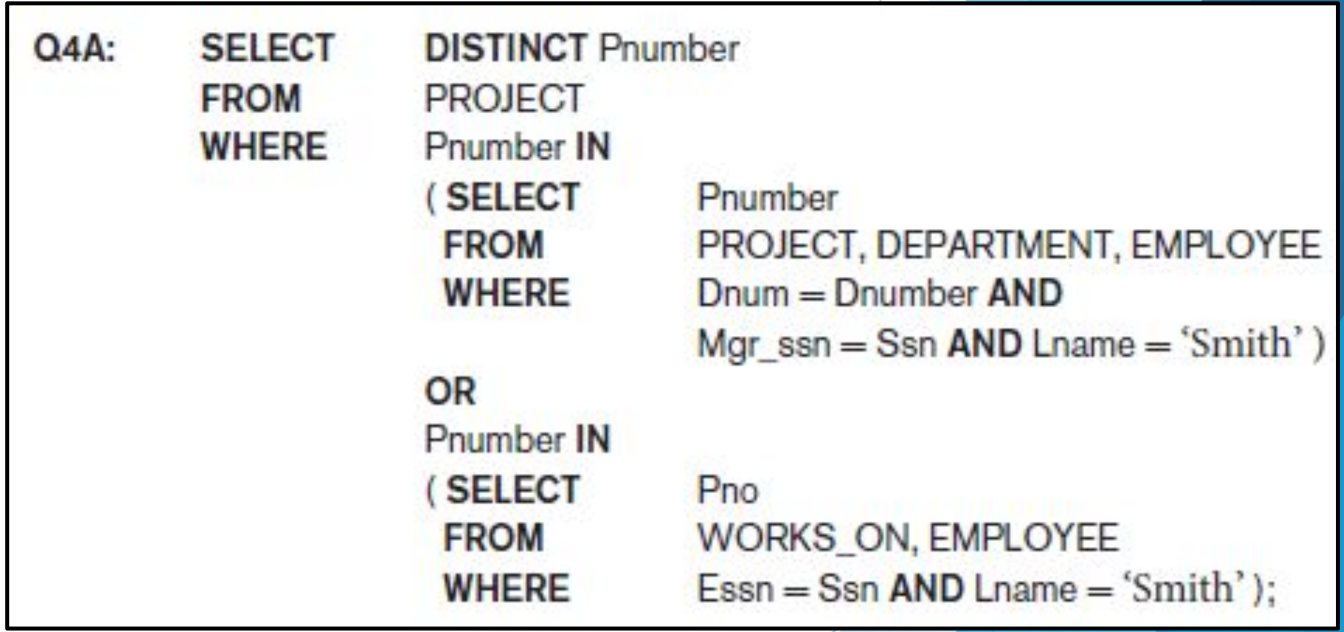
- *Nested Queries, Tuples, and Set/Multiset Comparisons*
- In Q4A:
- the first nested query **selects the project numbers of projects that have an employee with last name ‘Smith’ involved as manager,**
- whereas the second nested query **selects the project numbers of projects that have an employee with last name ‘Smith’ involved as worker.**
- In the outer query, we use the OR logical connective to retrieve a PROJECT tuple if the PNUMBER value of that tuple is in the result of either nested query.



```
Q4A:    SELECT    DISTINCT Pnumber
        FROM      PROJECT
        WHERE     Pnumber IN
                  ( SELECT    Pnumber
                    FROM      PROJECT, DEPARTMENT, EMPLOYEE
                    WHERE     Dnum = Dnumber AND
                              Mgr_ssn = Ssn AND Lname = ‘Smith’ )
                  OR
                  Pnumber IN
                  ( SELECT    Pno
                    FROM      WORKS_ON, EMPLOYEE
                    WHERE     Essn = Ssn AND Lname = ‘Smith’ );
```

More Complex SQL Retrieval Queries

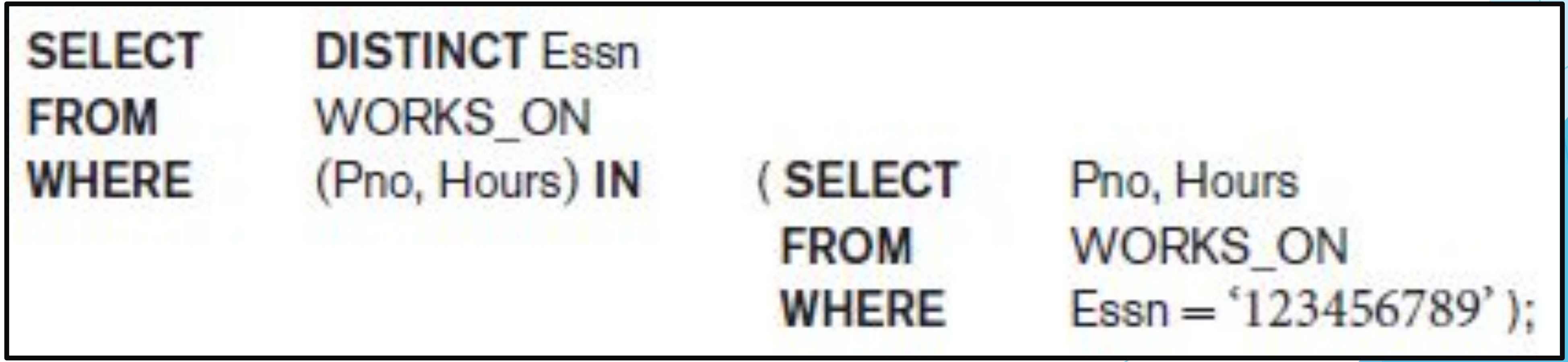
- ***Nested Queries, Tuples, and Set/Multiset Comparisons***
- If a nested query returns a **single attribute and a single tuple, the query result will be a single (scalar) value.**
- In such cases, it is **permissible to use = instead of IN** for the comparison operator.



```
Q4A:    SELECT      DISTINCT Pnumber
        FROM        PROJECT
        WHERE       Pnumber IN
                    ( SELECT     Pnumber
                      FROM       PROJECT, DEPARTMENT, EMPLOYEE
                      WHERE      Dnum = Dnumber AND
                                 Mgr_ssn = Ssn AND Lname = ‘Smith’ )
                    OR
                    Pnumber IN
                    ( SELECT     Pno
                      FROM       WORKS_ON, EMPLOYEE
                      WHERE      Essn = Ssn AND Lname = ‘Smith’ );
```

More Complex SQL Retrieval Queries

- ***Nested Queries, Tuples, and Set/Multiset Comparisons***
- SQL allows the **use of tuples of values in comparisons by placing them within parentheses**.
- This query will **select the Essns of all employees who work the same (project, hours) combination on some project that employee ‘John Smith’ (whose Ssn = ‘123456789’) works on.**
- In this example, the IN operator compares the sub tuple of values in parentheses (Pno, Hours) within each tuple in WORKS_ON with the set of **type-compatible tuples produced by the nested query.**



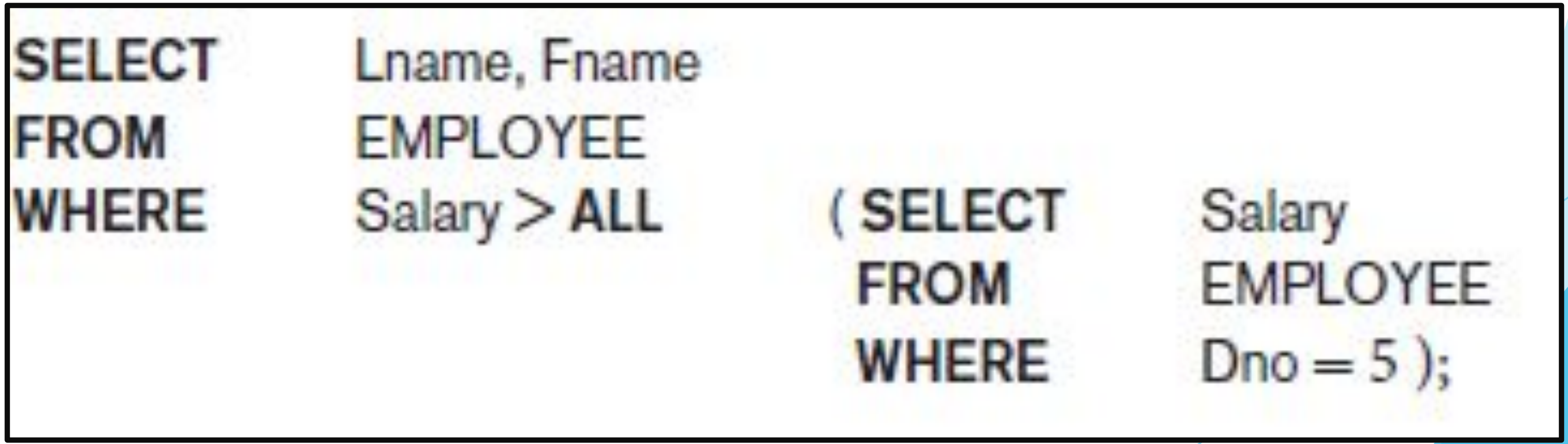
```
SELECT      DISTINCT Essn
FROM        WORKS_ON
WHERE       (Pno, Hours) IN     ( SELECT    Pno, Hours
                                  FROM      WORKS_ON
                                  WHERE     Essn = ‘123456789’ );
```

More Complex SQL Retrieval Queries

- ***Nested Queries, Tuples, and Set/Multiset Comparisons***
- Several other comparison operators can be used to compare a single value v1 to a set or multiset V (typically a nested query).
- The **= ANY (or = SOME)** operator returns TRUE if the value v1 is equal to some value in the set V and is hence **equivalent to IN**.
- Other operators that can be combined with ANY (or SOME) include >, >=, <, <=, and <>.

More Complex SQL Retrieval Queries

- ***Nested Queries, Tuples, and Set/Multiset Comparisons***
- The keyword **ALL** can also be combined with each of these operators.
- For example, **the comparison condition (v > ALL V) returns TRUE if the value v is greater than all the values in the set (or multiset) V.**
- An example is the query, which returns the names of employees whose salary is greater than the salary of all the employees in department 5.



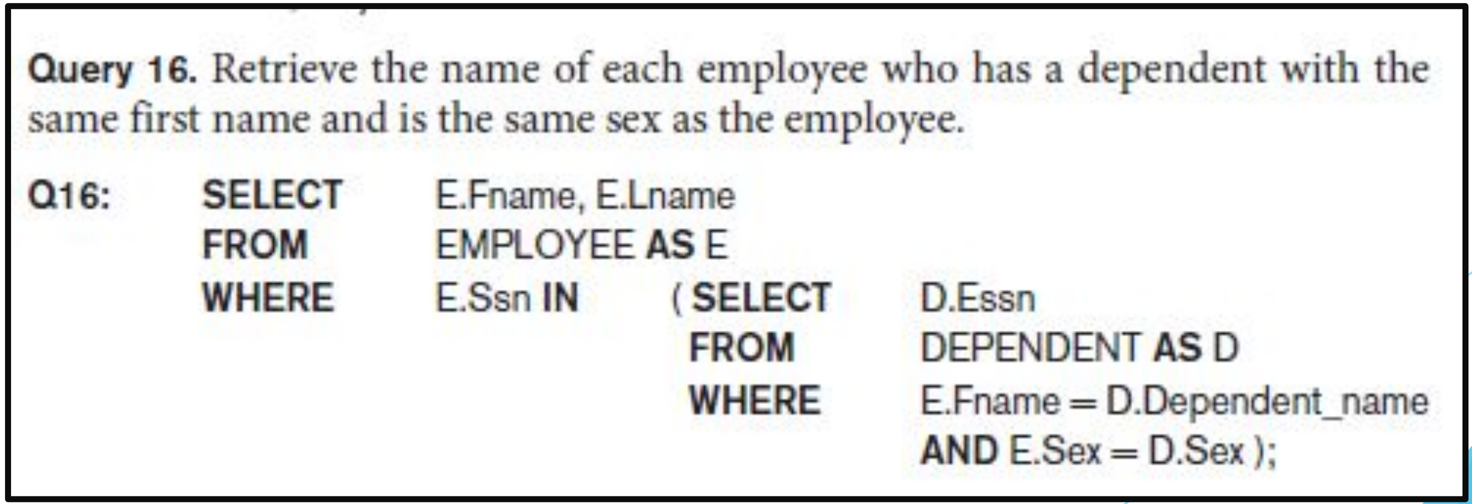
```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     Salary > ALL      ( SELECT    Salary
                              FROM      EMPLOYEE
                              WHERE     Dno = 5 );
```

More Complex SQL Retrieval Queries

- ***Nested Queries, Tuples, and Set/Multiset Comparisons***
- **AMBIGUITY ISSUE**
  - Ambiguity among attribute names if attributes of the same name exist—one in a relation in the FROM clause of the outer query, and another in a relation in the FROM clause of the nested query.
- To refer to an attribute of the relation specified in the outer query, we specify and **refer to an alias** (tuple variable) for that relation.

More Complex SQL Retrieval Queries

- ***Nested Queries, Tuples, and Set/Multiset Comparisons***
- **AMBIGUITY ISSUE**
- In the nested query of Q16, we must qualify **E.Sex because it refers to the Sex attribute of EMPLOYEE from the outer query, and DEPENDENT also has an attribute called Sex**.
- If there were any unqualified references to Sex in the nested query, they would refer to the Sex attribute of DEPENDENT.
- However, we would not have to qualify the attributes **Fname and Ssn of EMPLOYEE if they appeared in the nested query because the DEPENDENT relation does not have attributes called Fname and Ssn, so there is no ambiguity.**

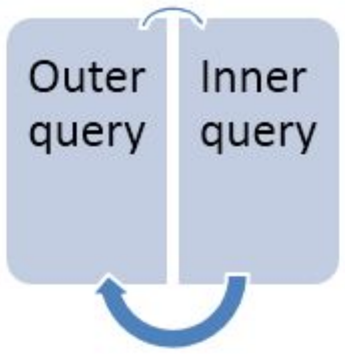


Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

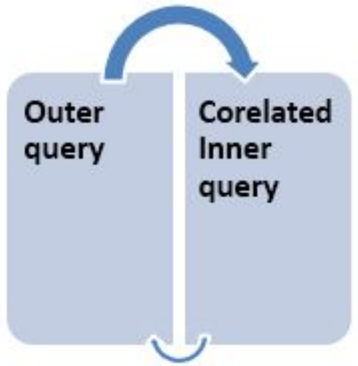
```
Q16:    SELECT   E.Fname, E.Lname
        FROM     EMPLOYEE AS E
        WHERE    E.Ssn IN    ( SELECT   D.Essn
                               FROM     DEPENDENT AS D
                               WHERE    E.Fname = D.Dependent_name
                                        AND E.Sex = D.Sex );
```

More Complex SQL Retrieval Queries

- *Correlated Nested Queries*
- Correlated subqueries are used for row-by-row processing.
- Each subquery is executed once for every row of the outer query.
- The parent statement can be a SELECT, UPDATE, or DELETE statement.



Ordinary Sub Query



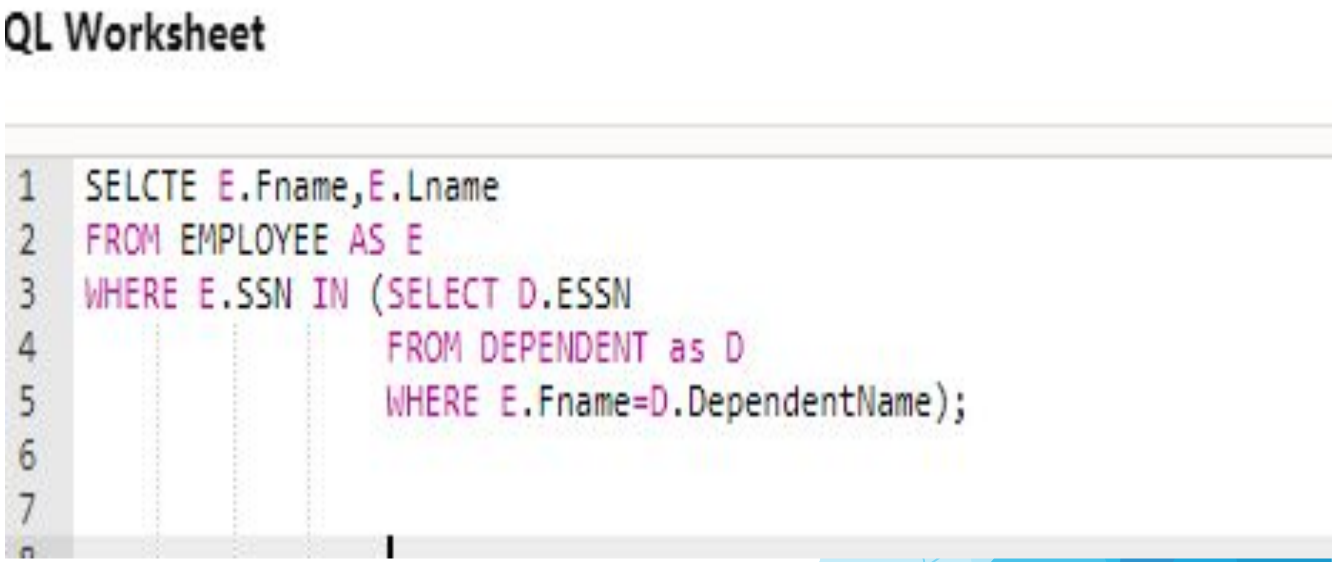
Correlated Sub Query

More Complex SQL Retrieval Queries

- ***Correlated Nested Queries***
- A correlated subquery is one way of reading every row in a table and comparing values in each row against related data.
- **Nested Subqueries Versus Correlated Subqueries :**
- With a normal nested subquery, the inner SELECT query runs first and executes once, returning values to be used by the main query.
- A correlated subquery, however, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query.
- **NOTE : You can also use the ANY and ALL operator in a correlated subquery.**

More Complex SQL Retrieval Queries

- ***Correlated Nested Queries***
- **Whenever a condition in the WHERE clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be correlated.**
- For example, we can think of Q16 as follows:
- For each EMPLOYEE tuple, evaluate the nested query, which retrieves the Essn values for all DEPENDENT tuples with the same name as that EMPLOYEE tuple; if the Ssn value of the EMPLOYEE tuple is in the result of the nested query, then select that EMPLOYEE tuple.



```
SELCTE E.Fname,E.Lname
FROM EMPLOYEE AS E
WHERE E.SSN IN (SELECT D.ESSN
                FROM DEPENDENT as D
                WHERE E.Fname=D.DependentName);
```

More Complex SQL Retrieval Queries

- ***The EXISTS Function in SQL***
- EXISTS is a Boolean functions that return **TRUE** or **FALSE**.
- Can be used in a WHERE clause condition.
- **EXISTS function:** used to check whether the result of a nested query is empty (contains no tuples) or not.
- The result of:
  - EXISTS = TRUE; if the nested query result contains at least one tuple,
  - EXISTS= FALSE if the nested query result contains no tuples.
- EXISTS and NOT EXISTS are typically used in conjunction with a correlated nested query.

More Complex SQL Retrieval Queries

- *The EXISTS Function in SQL*
- EXISTS Operator
- Query: Find the detail of employee who is atleast working on a project.
- Select * From Employee

```
 Where EXISTS (Select Eid from Project
                    Where Employee.Eid = ProjectEid)
```

Eid in employee table compared with employee id in project table.
If match is found then TRUE, select that row.

Project Table

Eid	Pid	Pname	Location
1	P1	IoT	Ohio
5	P2	AI	Los Angeles
3	P3	Cloud	Chicago
4	P4	Android	Texas

Employee Table

Eid	Ename	Address
1	Brown	New York
2	Sam	Houston
3	Smith	Seattle
4	John	Denver
5	Mary	San Francisco

More Complex SQL Retrieval Queries

- ***The EXISTS Function in SQL***
- **NOT EXISTS Operator**
- Query: Find the detail of employee who is not working on a project.
- **Select * From Employee**

  Where NOT EXISTS (Select Eid from Project

  Where Employee.Eid = Project.Eid)

Eid in employee table compared with employee id in project table.
If match is not found then TRUE, select that row.

Project Table

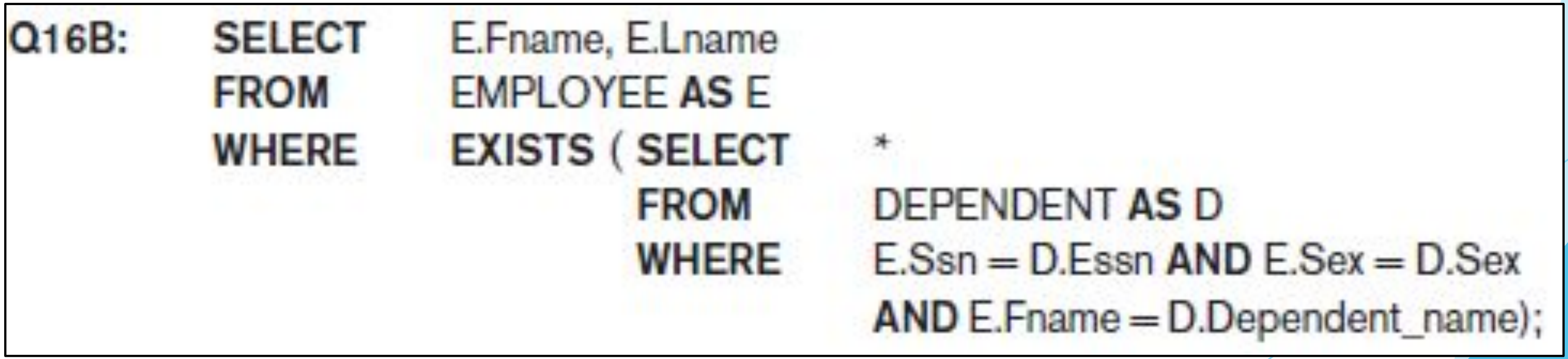
Eid	Pid	Pname	Location
1	P1	IoT	Ohio
5	P2	AI	Los Angeles
3	P3	Cloud	Chicago
4	P4	Android	Texas

Employee Table

Eid	Ename	Address
1	Brown	New York
2	Sam	Houston
3	Smith	Seattle
4	John	Denver
5	Mary	San Francisco

More Complex SQL Retrieval Queries

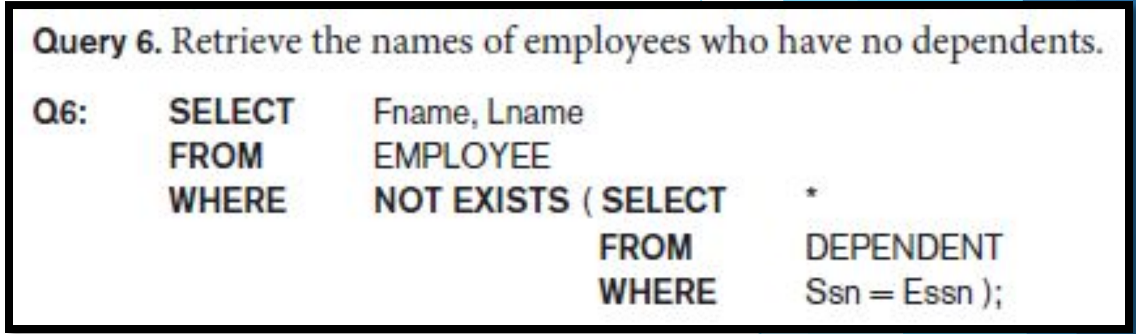
- ***The EXISTS Function in SQL***
- In Q16B, the nested query references the **Ssn, Fname, and Sex attributes of the EMPLOYEE relation from the outer query.**
- **For each EMPLOYEE tuple, evaluate the nested query, which retrieves all DEPENDENT tuples with the same Essn, Sex, and Dependent_name as the EMPLOYEE tuple; if at least one tuple EXISTS in the result of the nested query, then select that EMPLOYEE tuple.**
- EXISTS(Q) returns TRUE if there is at least one tuple in the result of the nested query, and returns FALSE otherwise.



```
Q16B:   SELECT   E.Fname, E.Lname
        FROM     EMPLOYEE AS E
        WHERE    EXISTS ( SELECT   *
                          FROM     DEPENDENT AS D
                          WHERE    E.Ssn = D.Essn AND E.Sex = D.Sex
                                   AND E.Fname = D.Dependent_name);
```

More Complex SQL Retrieval Queries

- ***The EXISTS Function in SQL***
- NOT EXISTS(Q) returns TRUE if there are no tuples in the result of nested query Q, and returns FALSE otherwise.
- In Q6, the correlated nested query retrieves all DEPENDENT tuples related to a particular EMPLOYEE tuple.
- If none exist, the EMPLOYEE tuple is selected because the WHERE-clause condition will evaluate to TRUE in this case.
- Q6 as follows: For each EMPLOYEE tuple, the correlated nested query selects all DEPENDENT tuples whose Essn value matches the EMPLOYEE Ssn; if the result is empty, no dependents are related to the employee, so we select that EMPLOYEE tuple and retrieve its Fname and Lname.

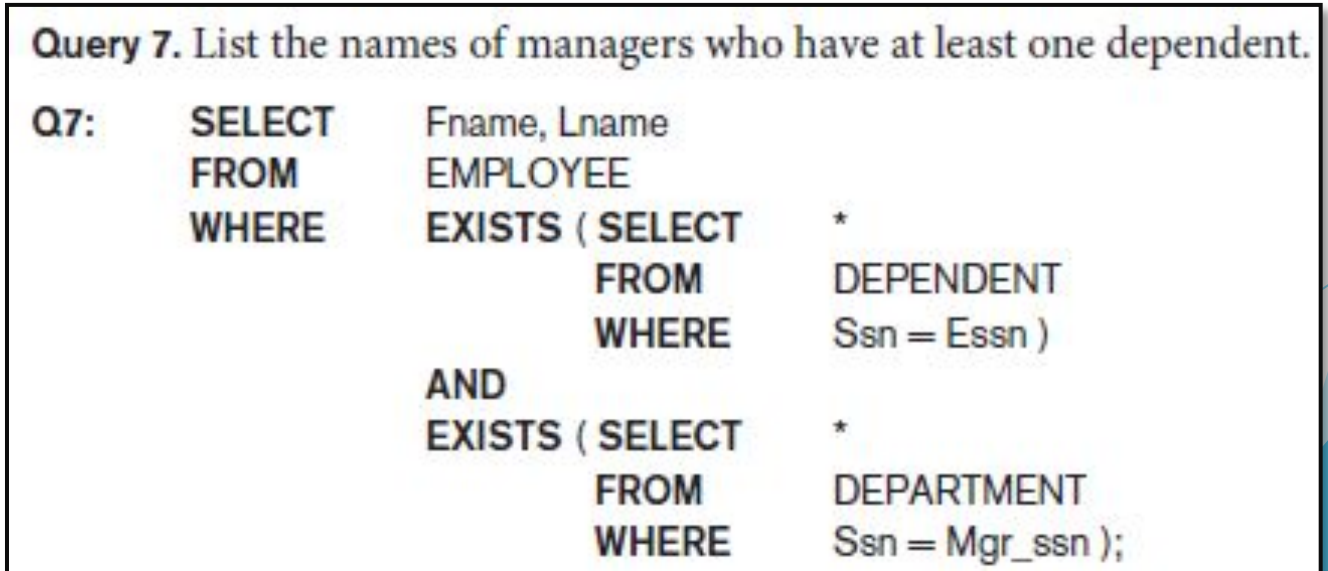


Query 6. Retrieve the names of employees who have no dependents.

```
Q6:   SELECT   Fname, Lname
      FROM     EMPLOYEE
      WHERE    NOT EXISTS ( SELECT   *
                            FROM     DEPENDENT
                            WHERE    Ssn = Essn );
```

More Complex SQL Retrieval Queries

- ***The EXISTS Function in SQL***
- One way to write this query is shown in Q7, where we specify two nested correlated queries.
- The first selects all DEPENDENT tuples related to an EMPLOYEE, and the second selects all DEPARTMENT tuples managed by the EMPLOYEE.
- If at least one of the first and at least one of the second exists, we select the EMPLOYEE tuple. **First checks the dependents are NULL & then check the person is manager**

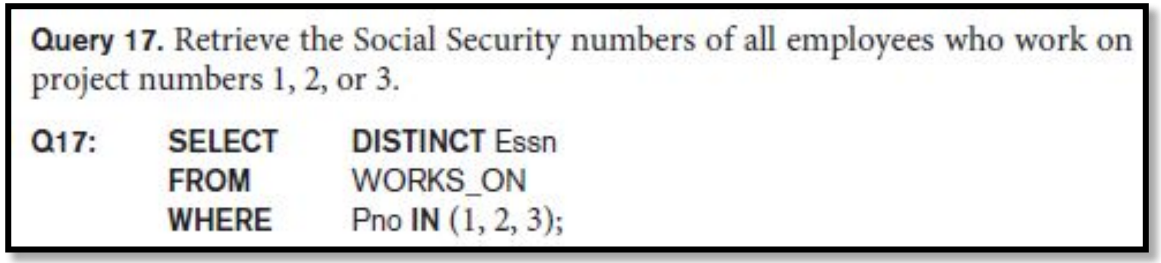


Query 7. List the names of managers who have at least one dependent.

```
Q7:    SELECT    Fname, Lname
       FROM      EMPLOYEE
       WHERE     EXISTS ( SELECT    *
                          FROM      DEPENDENT
                          WHERE     Ssn = Essn )
                 AND
                 EXISTS ( SELECT    *
                          FROM      DEPARTMENT
                          WHERE     Ssn = Mgr_ssn );
```

More Complex SQL Retrieval Queries

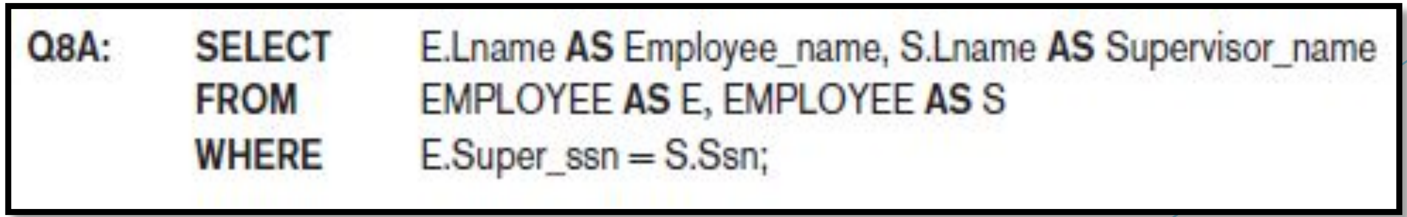
- ***Explicit Sets and Renaming in SQL***
- It is also possible to use **an explicit set of values in the WHERE clause,** rather than a nested query. Such a set is enclosed in parentheses in SQL.



Query 17. Retrieve the Social Security numbers of all employees who work on project numbers 1, 2, or 3.

```
Q17:    SELECT    DISTINCT Essn
        FROM      WORKS_ON
        WHERE     Pno IN (1, 2, 3);
```

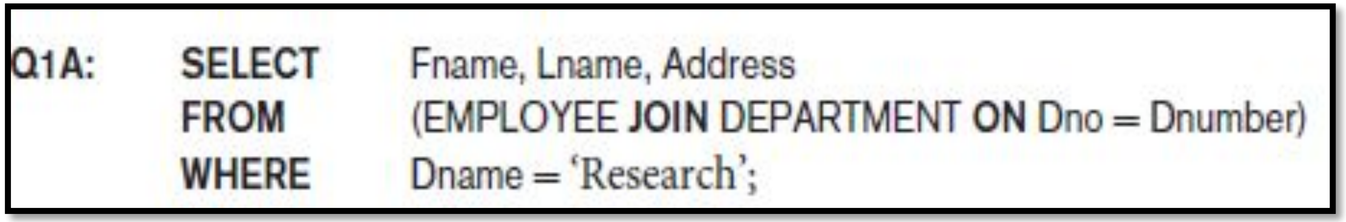
- In SQL, it is possible to rename any attribute that appears in the result of a query by adding the qualifier **AS** followed by the desired new name.
- Hence, **the AS construct can be used to alias both attribute and relation names** in general, and it can be used in appropriate parts of a query.



```
Q8A:    SELECT    E.Lname AS Employee_name, S.Lname AS Supervisor_name
        FROM      EMPLOYEE AS E, EMPLOYEE AS S
        WHERE     E.Super_ssn = S.Ssn;
```

More Complex SQL Retrieval Queries

- *Joined Tables in SQL and Outer Joins*
- The concept of a joined table (or joined relation) was incorporated into SQL to **permit users to specify a table resulting from a join operation in the FROM clause of a query.**
- **Join construct is easy to implement rather than mixing all statemsnts in where clause**
- For example, consider query, which retrieves the name and address of every employee who works for the ‘Research’ department. It may be easier to specify the join of the EMPLOYEE and DEPARTMENT relations in the WHERE clause, and then to select the desired tuples and attributes.



```
Q1A:    SELECT   Fname, Lname, Address
        FROM     (EMPLOYEE JOIN DEPARTMENT ON Dno = Dnumber)
        WHERE    Dname = ‘Research’;
```

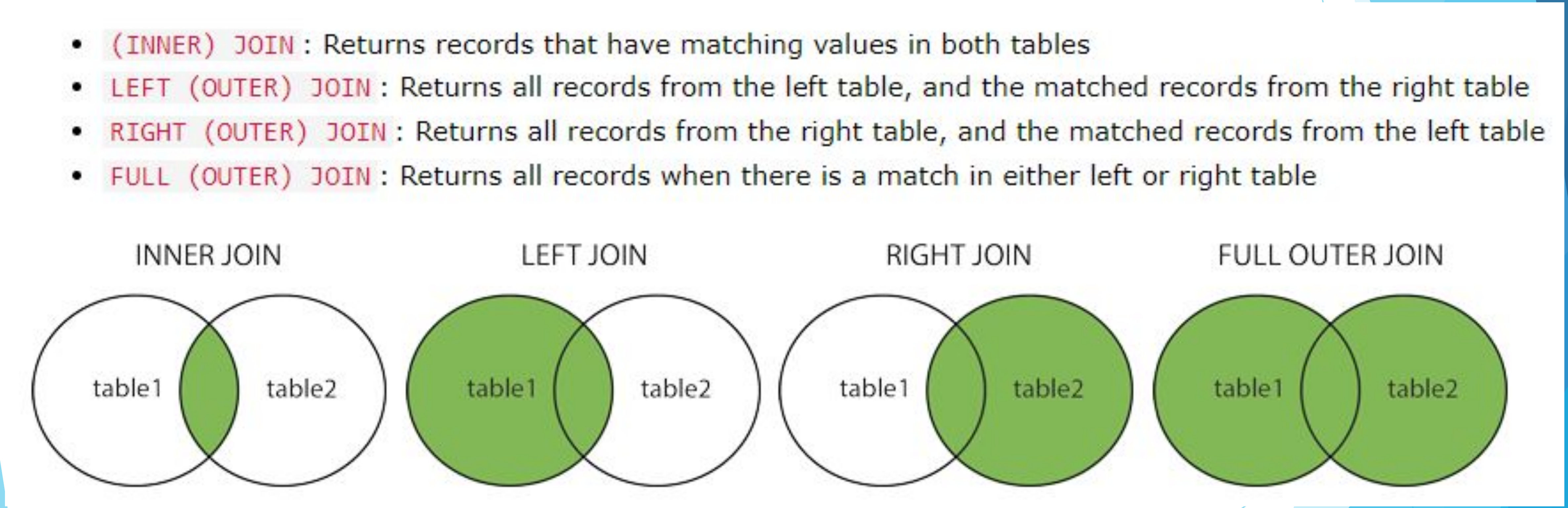
- The FROM clause in Q1A contains a single joined table. The attributes of such a table are **all the attributes of the first table, EMPLOYEE, followed by all the attributes of the second table, DEPARTMENT.**
- The concept of a joined table also allows the user to specify different types of join, such as NATURAL JOIN and various types of OUTER JOIN. In a NATURAL JOIN on two relations R and S, no join condition is specified.

More Complex SQL Retrieval Queries

- ***Joined Tables in SQL and Outer Joins***
- The default type of join in a joined table is called an **inner join, where a tuple is included in the result only if a matching tuple exists in the other relation.**
- In SQL, the options available for specifying joined tables include:
  - **INNER JOIN** (only pairs of tuples that match the join condition are retrieved, same as JOIN),
  - **LEFT OUTER JOIN** (every tuple in the left table must appear in the result; if it does not have a matching tuple, it is padded with NULL values for the attributes of the right table),
  - **RIGHT OUTER JOIN** (every tuple in the right table must appear in the result; if it does not have a matching tuple, it is padded with NULL values for the attributes of the lefttable), and FULL OUTER JOIN. In the latter three options, the keyword OUTER may be omitted.
  - If the join attributes have the same name, one can also specify the natural join variation of outer joins by using the keyword **NATURAL** before the operation (for example, NATURAL LEFT OUTER JOIN).
  - The keyword **CROSS JOIN** is used to specify the CARTESIAN PRODUCT although this should be used only with the utmost care because it generates all possible tuple combinations.

More Complex SQL Retrieval Queries

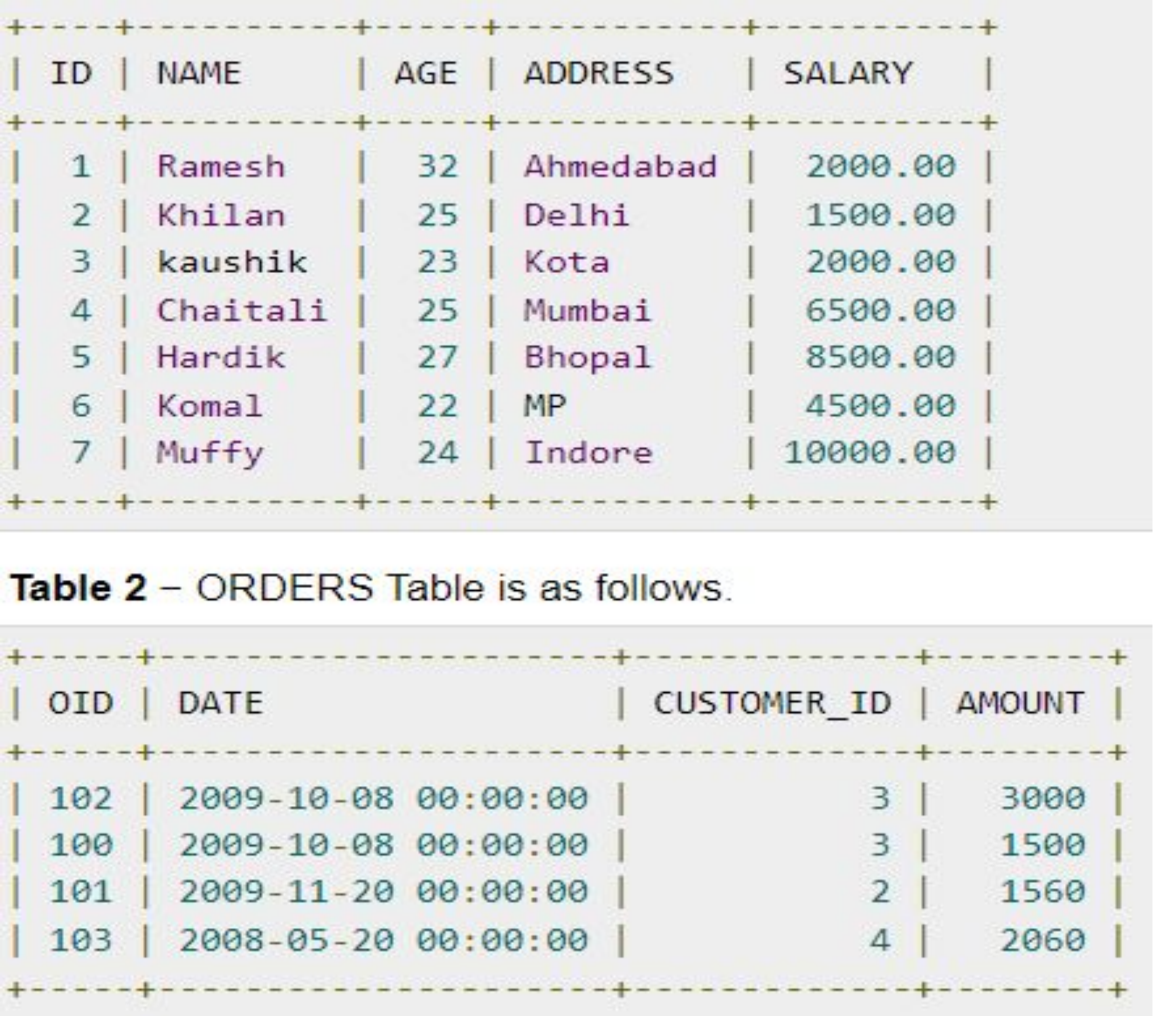
- ***Joined Tables in SQL and Outer Joins***



- `(INNER) JOIN` : Returns records that have matching values in both tables
- `LEFT (OUTER) JOIN` : Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN` : Returns all records from the right table, and the matched records from the left table
- `FULL (OUTER) JOIN` : Returns all records when there is a match in either left or right table

More Complex SQL Retrieval Queries

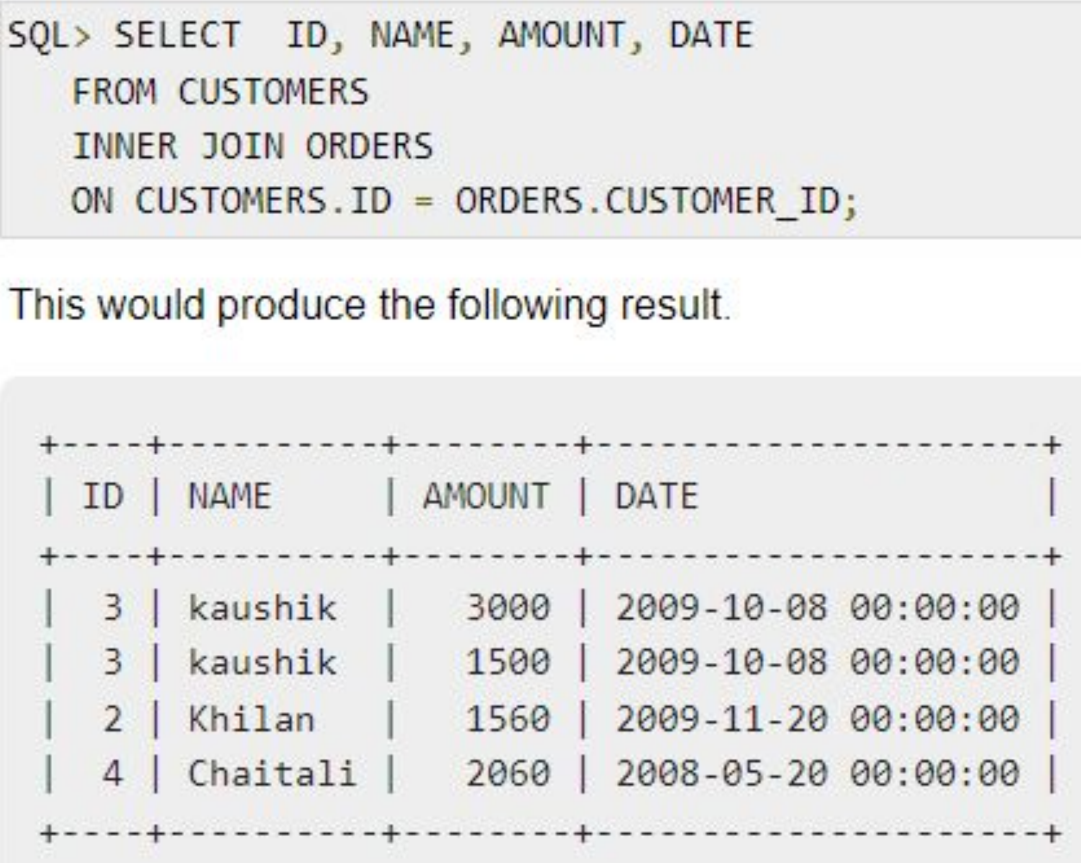
- ***Joined Tables in SQL and Outer Joins - INNER JOIN***



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 – ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+
| OID | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```



```
SQL> SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   INNER JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result.

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+----+----------+--------+---------------------+
```

More Complex SQL Retrieval Queries

- ***Joined Tables in SQL and Outer Joins - LEFT JOIN***

3 occurs two times because 3 is matched twice in the orders table as the customer has placed order two times.

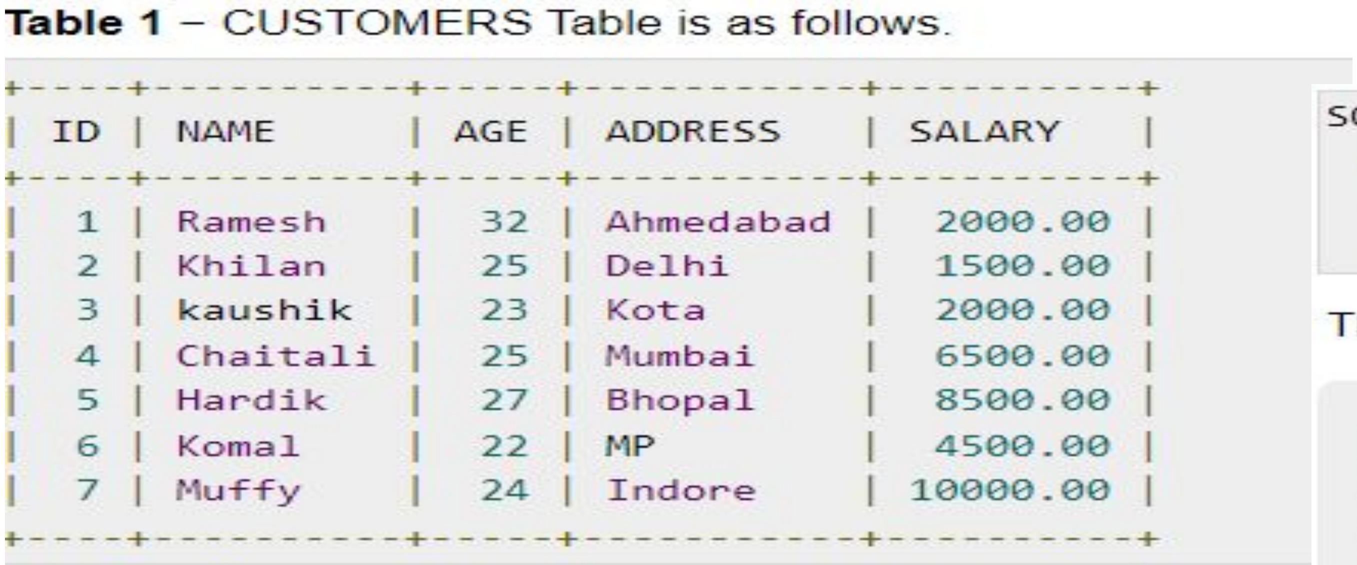


Table 1 – CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

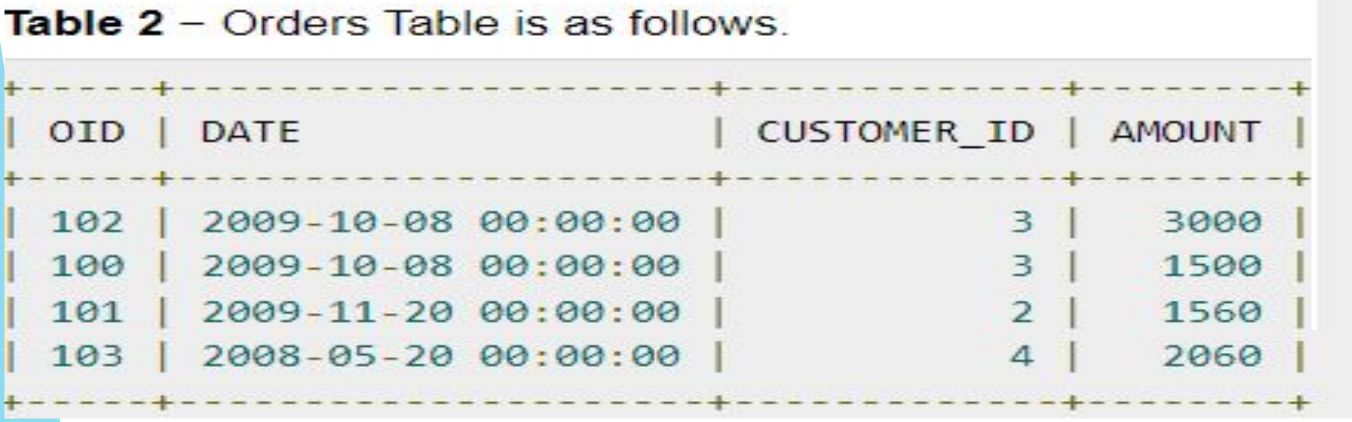
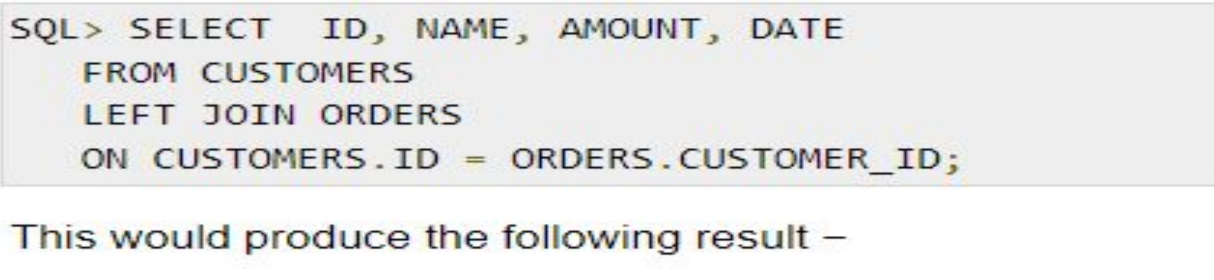


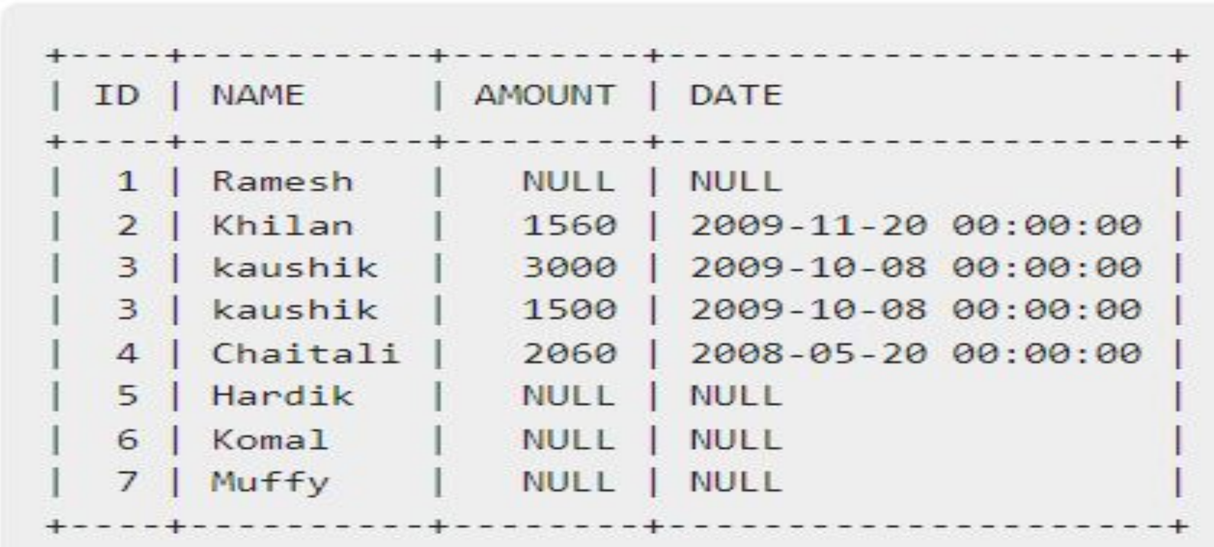
Table 2 – Orders Table is as follows.

```
+-----+---------------------+-------------+--------+
| OID | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```



```
SQL> SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   LEFT JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result –



```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  1 | Ramesh   |   NULL | NULL                |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|  5 | Hardik   |   NULL | NULL                |
|  6 | Komal    |   NULL | NULL                |
|  7 | Muffy    |   NULL | NULL                |
+----+----------+--------+---------------------+
```

More Complex SQL Retrieval Queries

- ***Joined Tables in SQL and Outer Joins - RIGHT JOIN***

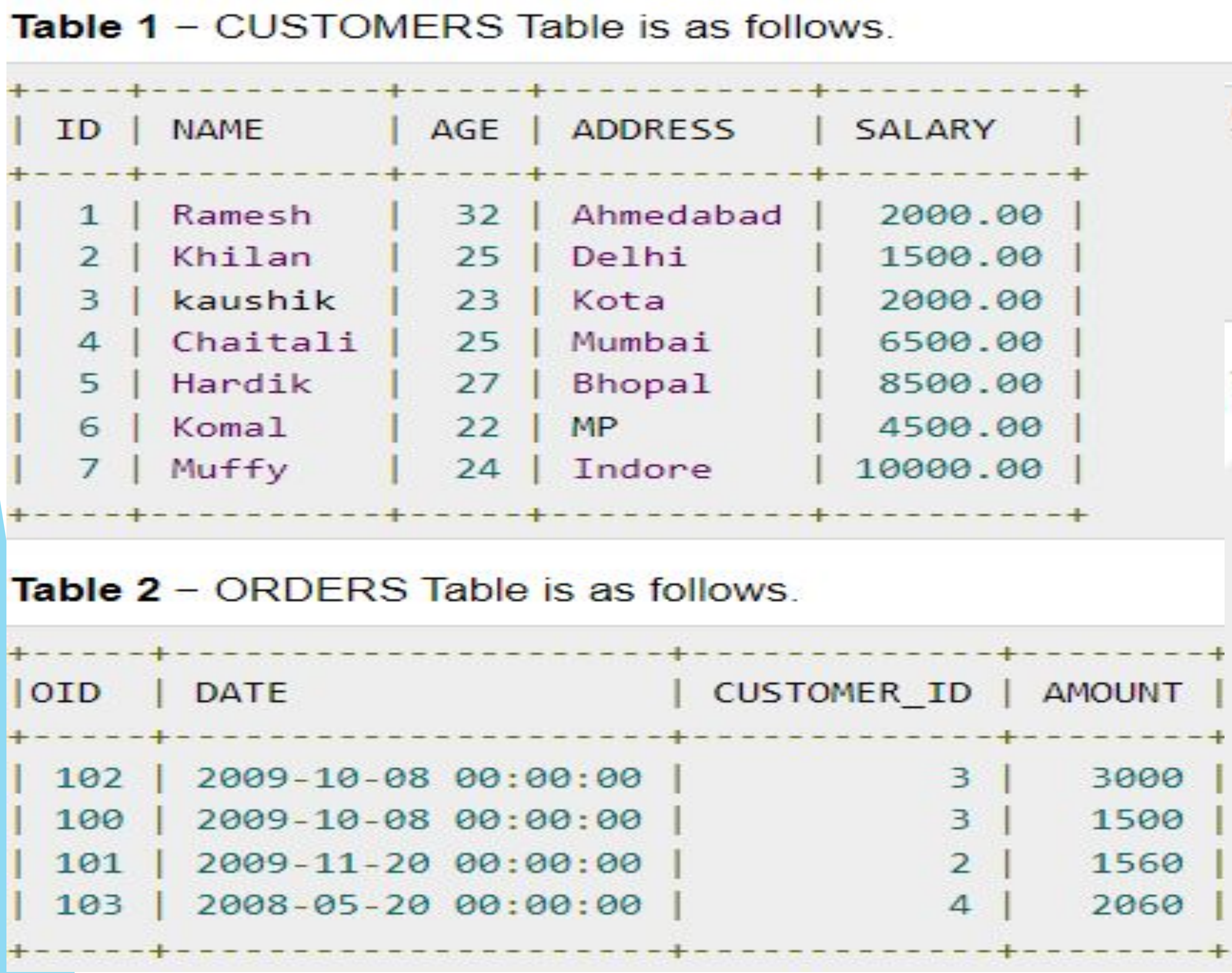
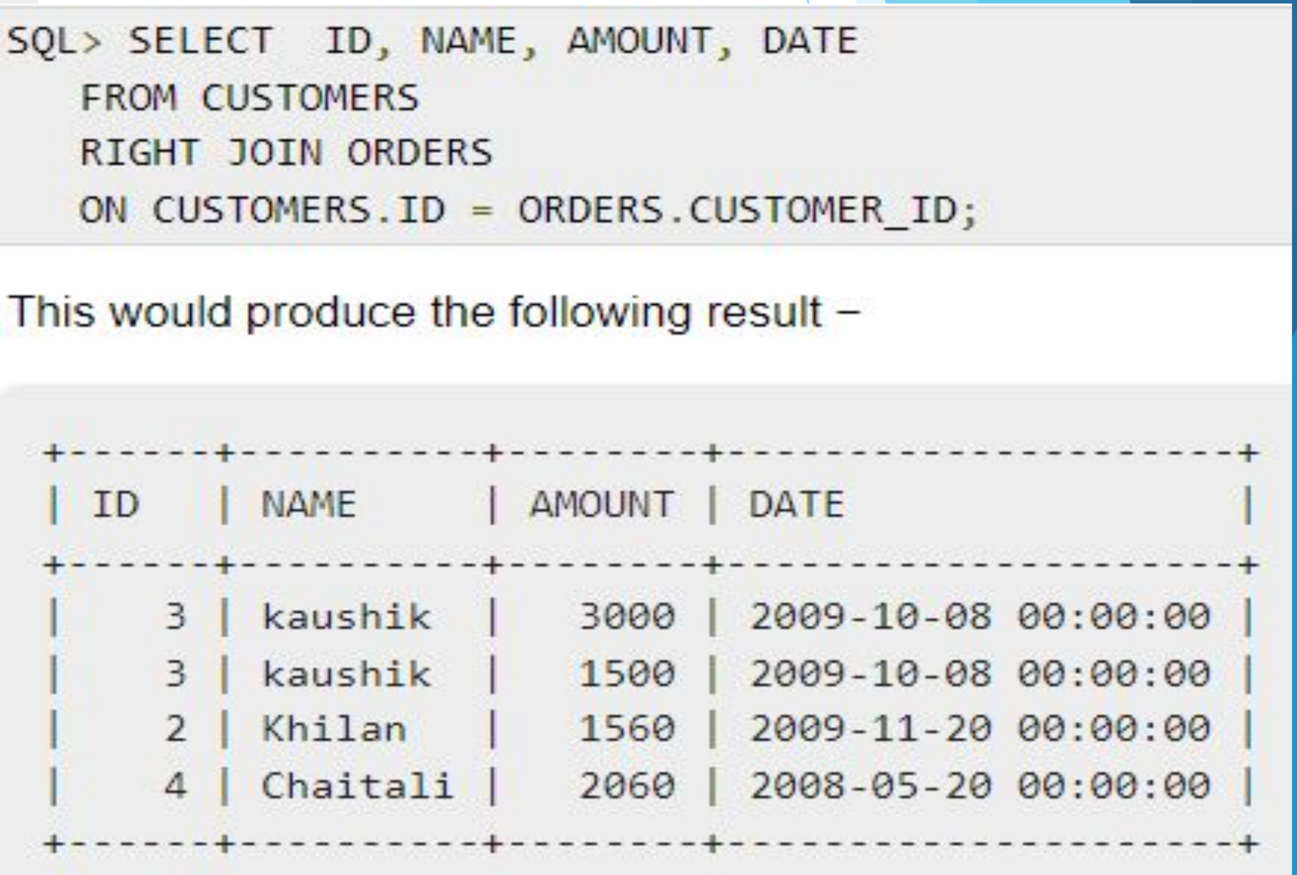


Table 1 – CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 – ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```



```
SQL> SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   RIGHT JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result –

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

More Complex SQL Retrieval Queries

3 occurs two times because 3 is matched twice in the orders table as the customer has placed order two times.

- *Joined Tables in SQL and Outer Joins - FULL JOIN*

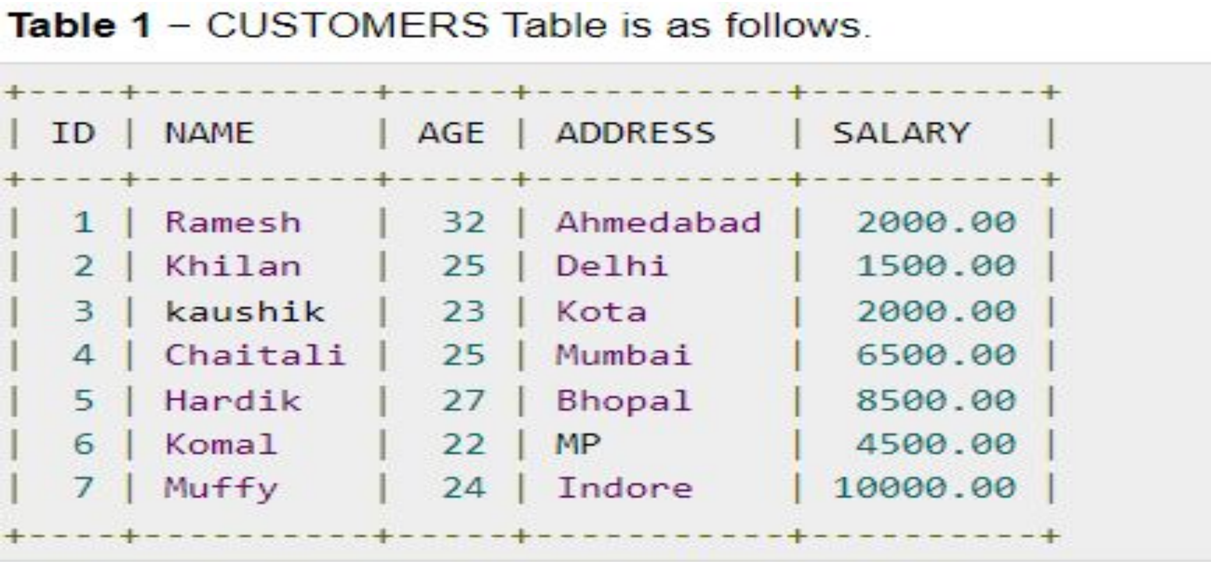


Table 1 – CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

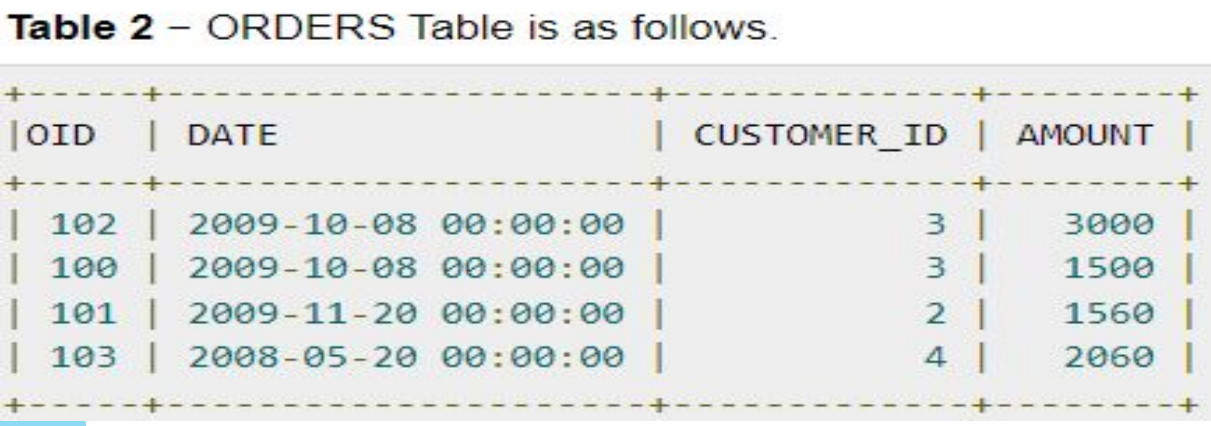
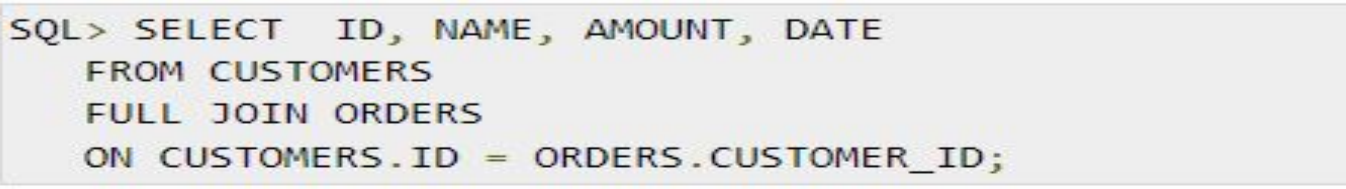
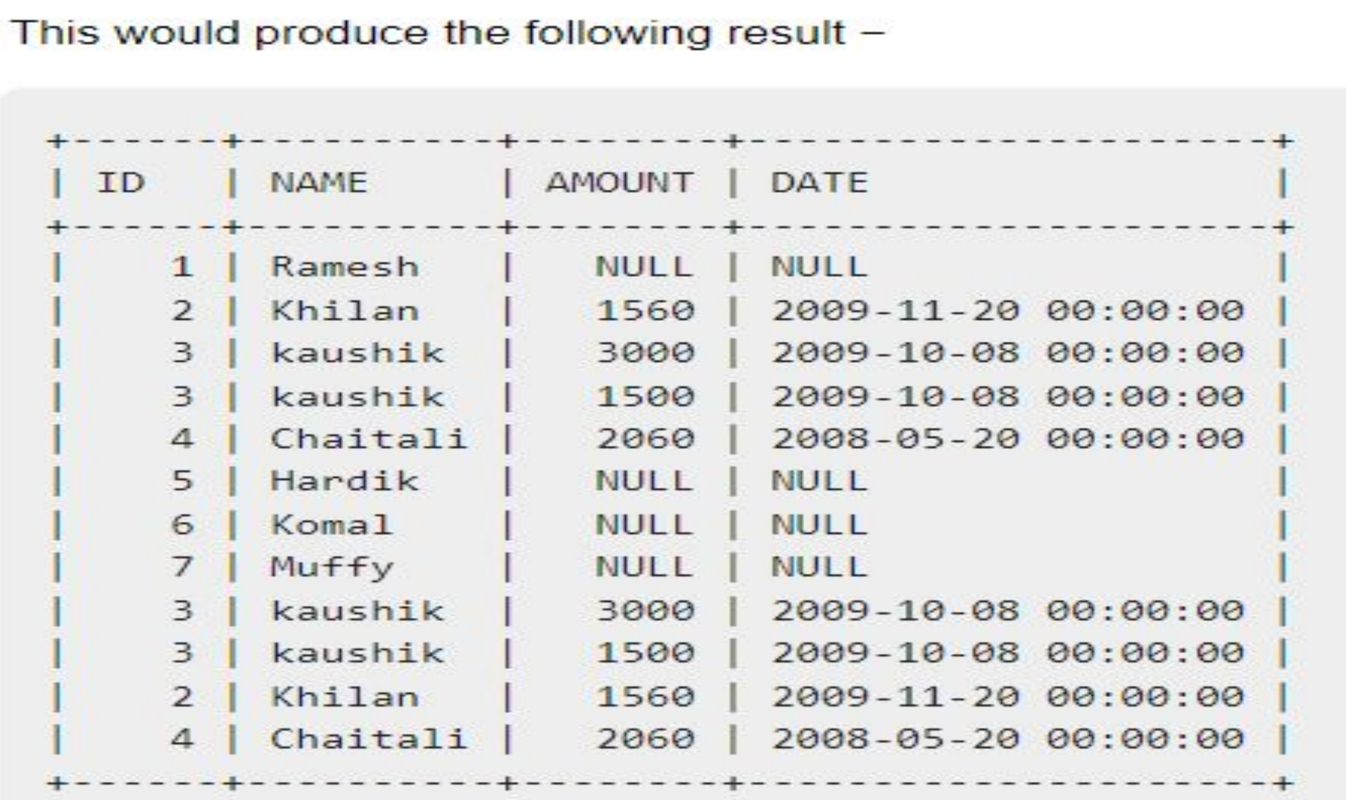


Table 2 – ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```



```
SQL> SELECT  ID, NAME, AMOUNT, DATE
     FROM CUSTOMERS
     FULL JOIN ORDERS
     ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

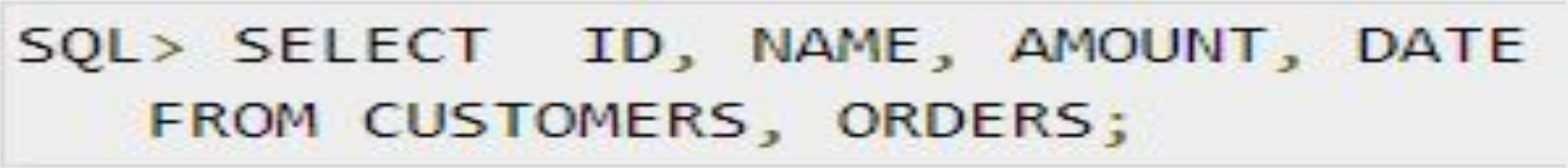


This would produce the following result –

```
+------+-----------+--------+---------------------+
| ID   | NAME      | AMOUNT | DATE                |
+------+-----------+--------+---------------------+
|    1 | Ramesh    |   NULL | NULL                |
|    2 | Khilan    |   1560 | 2009-11-20 00:00:00 |
|    3 | kaushik   |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik   |   1500 | 2009-10-08 00:00:00 |
|    4 | Chaitali  |   2060 | 2008-05-20 00:00:00 |
|    5 | Hardik    |   NULL | NULL                |
|    6 | Komal     |   NULL | NULL                |
|    7 | Muffy     |   NULL | NULL                |
|    3 | kaushik   |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik   |   1500 | 2009-10-08 00:00:00 |
|    2 | Khilan    |   1560 | 2009-11-20 00:00:00 |
|    4 | Chaitali  |   2060 | 2008-05-20 00:00:00 |
+------+-----------+--------+---------------------+
```

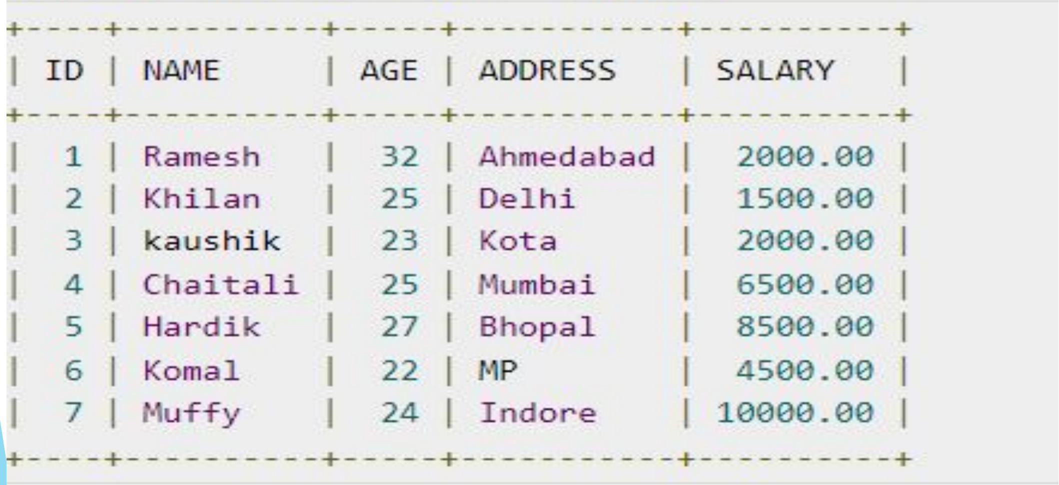
More Complex SQL Retrieval Queries

- ***Joined Tables in SQL and Outer Joins - CROSS JOIN***



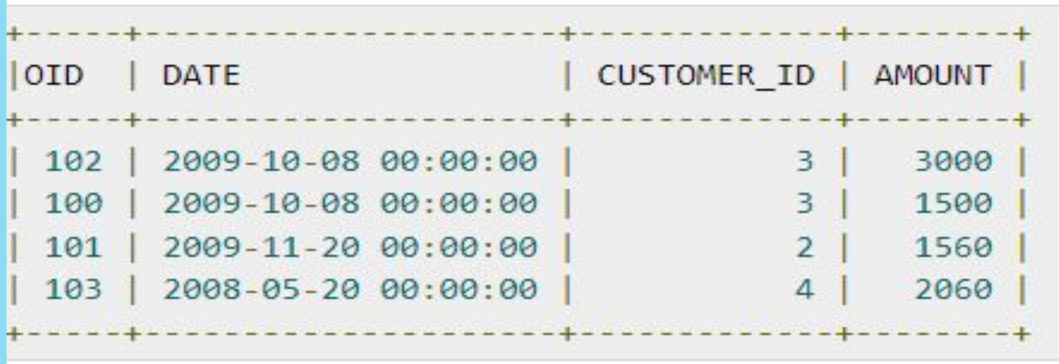
```
SQL> SELECT  ID, NAME, AMOUNT, DATE
     FROM CUSTOMERS, ORDERS;
```

Table 1 – CUSTOMERS table is as follows.

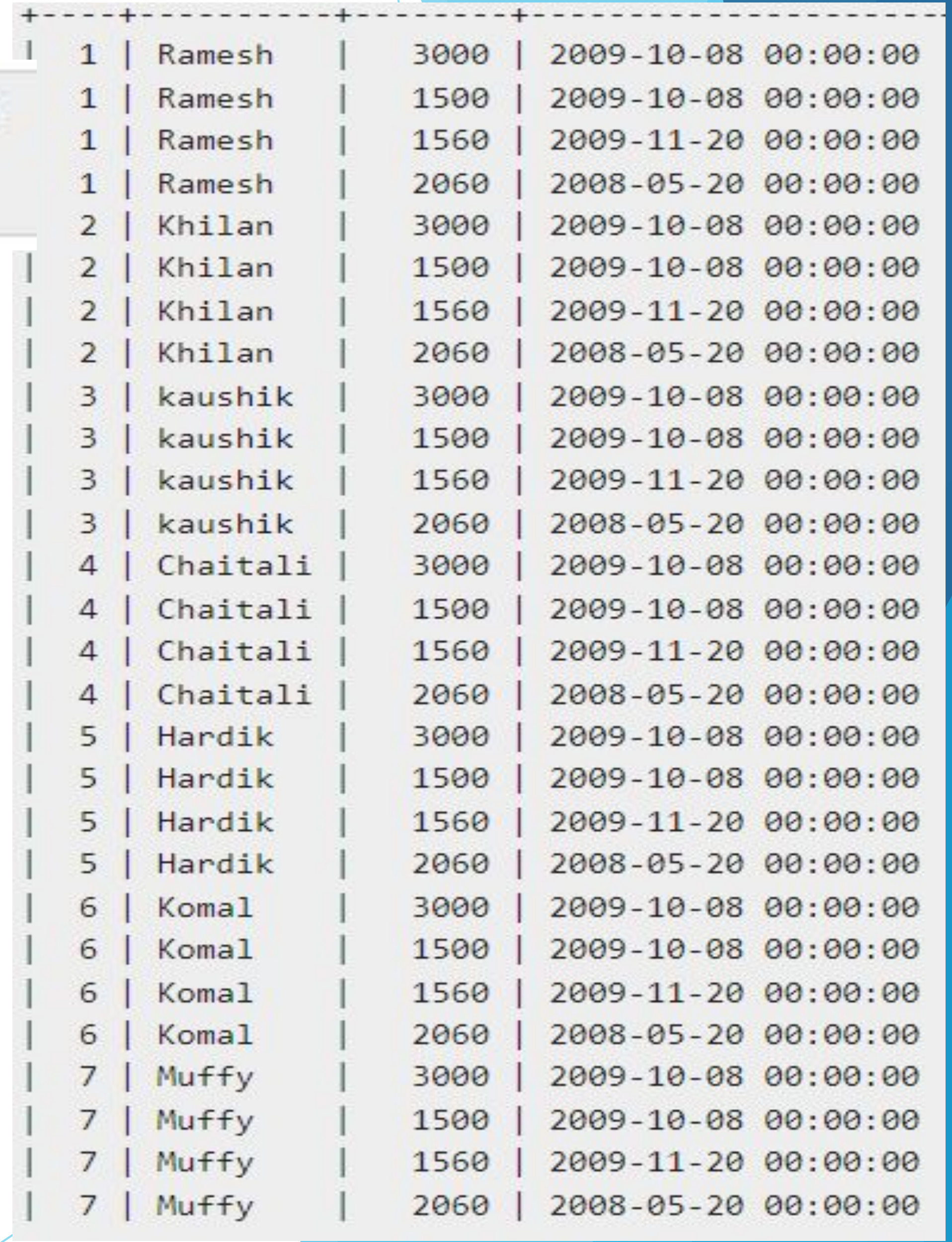


```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2: ORDERS Table is as follows –



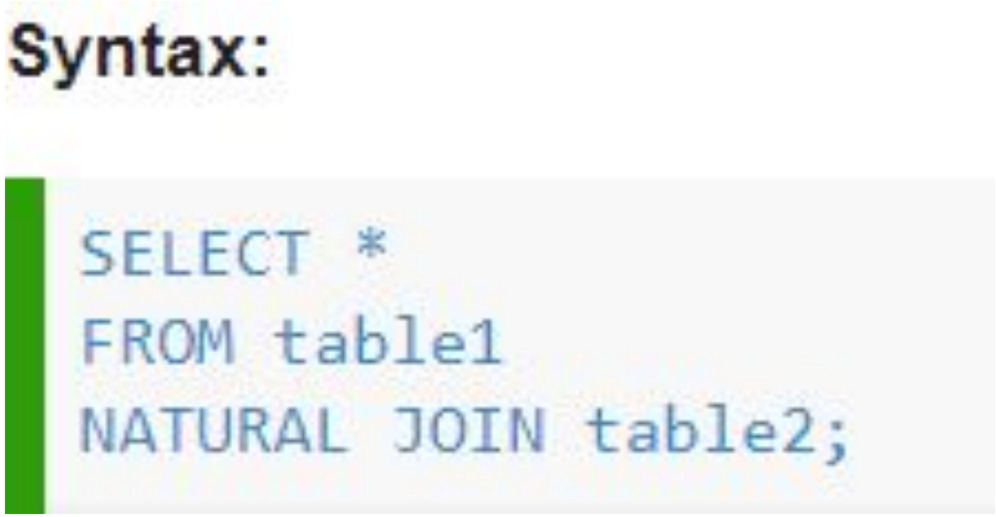
```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```



```
+----+----------+--------+---------------------
|  1 | Ramesh   |   3000 | 2009-10-08 00:00:00
   1 | Ramesh   |   1500 | 2009-10-08 00:00:00
   1 | Ramesh   |   1560 | 2009-11-20 00:00:00
   1 | Ramesh   |   2060 | 2008-05-20 00:00:00
   2 | Khilan   |   3000 | 2009-10-08 00:00:00
|  2 | Khilan   |   1500 | 2009-10-08 00:00:00
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00
|  2 | Khilan   |   2060 | 2008-05-20 00:00:00
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00
|  3 | kaushik  |   1560 | 2009-11-20 00:00:00
|  3 | kaushik  |   2060 | 2008-05-20 00:00:00
|  4 | Chaitali |   3000 | 2009-10-08 00:00:00
|  4 | Chaitali |   1500 | 2009-10-08 00:00:00
|  4 | Chaitali |   1560 | 2009-11-20 00:00:00
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00
|  5 | Hardik   |   3000 | 2009-10-08 00:00:00
|  5 | Hardik   |   1500 | 2009-10-08 00:00:00
|  5 | Hardik   |   1560 | 2009-11-20 00:00:00
|  5 | Hardik   |   2060 | 2008-05-20 00:00:00
|  6 | Komal    |   3000 | 2009-10-08 00:00:00
|  6 | Komal    |   1500 | 2009-10-08 00:00:00
|  6 | Komal    |   1560 | 2009-11-20 00:00:00
|  6 | Komal    |   2060 | 2008-05-20 00:00:00
|  7 | Muffy    |   3000 | 2009-10-08 00:00:00
|  7 | Muffy    |   1500 | 2009-10-08 00:00:00
|  7 | Muffy    |   1560 | 2009-11-20 00:00:00
|  7 | Muffy    |   2060 | 2008-05-20 00:00:00
```

More Complex SQL Retrieval Queries

- *Joined Tables in SQL and Outer Joins - NATURAL JOIN*
- Natural Join joins two tables based on same attribute name and data types.
- The resulting table will contain all the attributes of both the table but keep only one copy of each common column.
- Don’t use ON clause

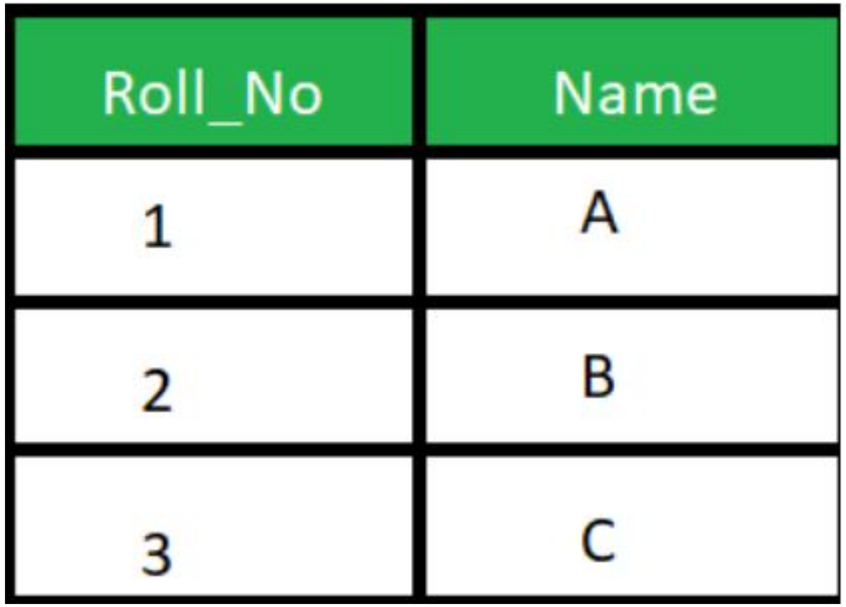


Syntax:

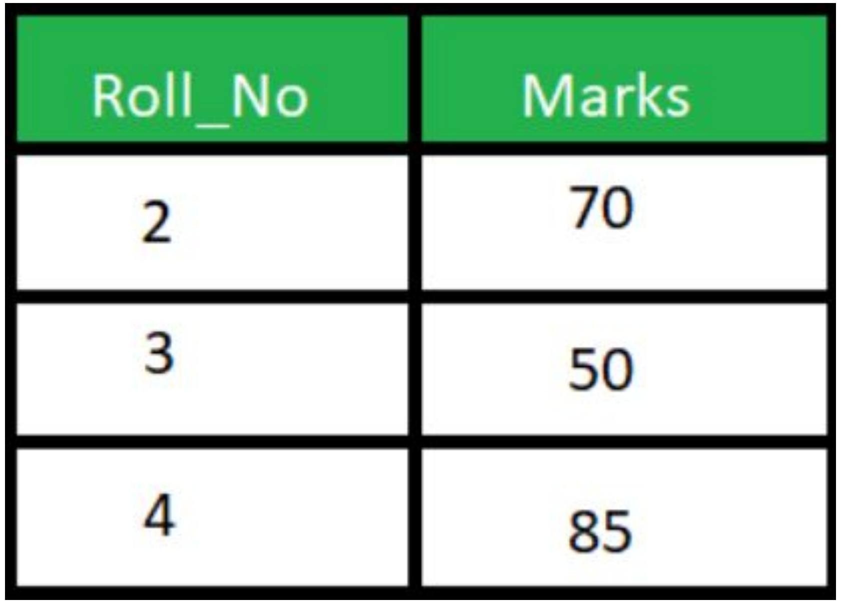
```
SELECT *
FROM table1
NATURAL JOIN table2;
```

More Complex SQL Retrieval Queries

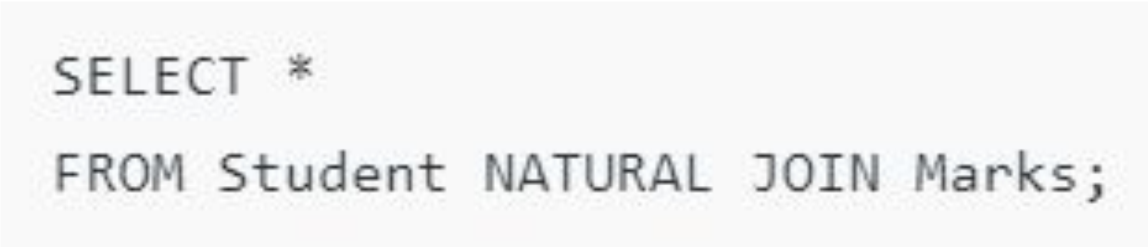
- *Joined Tables in SQL and Outer Joins - NATURAL JOIN*



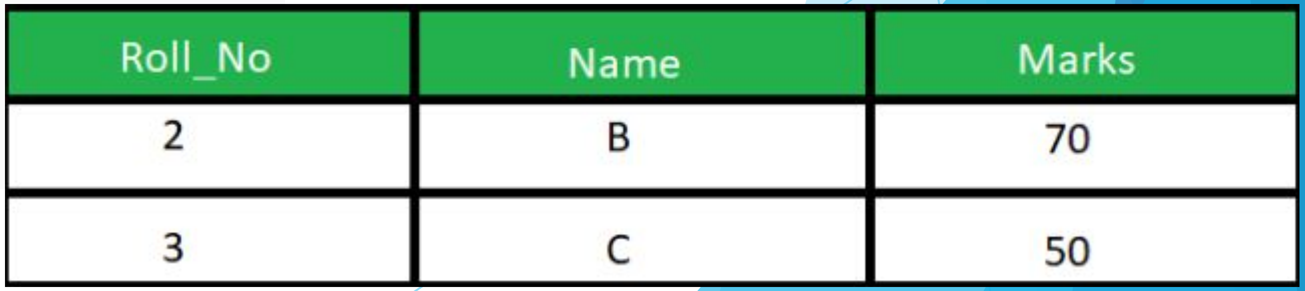
Roll_No	Name
1	A
2	B
3	C



Roll_No	Marks
2	70
3	50
4	85



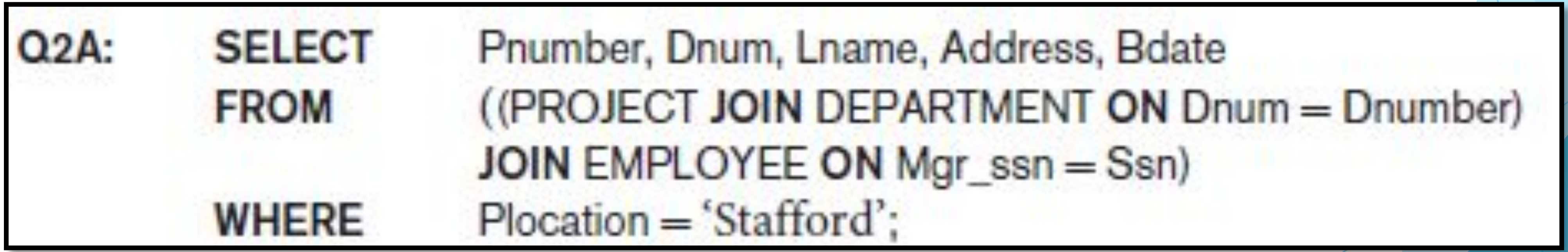
```
SELECT *
FROM Student NATURAL JOIN Marks;
```



Roll_No	Name	Marks
2	B	70
3	C	50

More Complex SQL Retrieval Queries

- ***Joined Tables in SQL and Outer Joins***
- It is also possible to nest join specifications; that is, one of the tables in a join may itself be a joined table.
- This allows the specification of the join of three or more tables as a single joined table, which is called a **multiway join**.
- For example, Q2A is a different way of specifying query Q2 from Section 6.3.1 using the concept of a joined table:



```
Q2A:    SELECT    Pnumber, Dnum, Lname, Address, Bdate
        FROM      ((PROJECT JOIN DEPARTMENT ON Dnum = Dnumber)
                  JOIN EMPLOYEE ON Mgr_ssn = Ssn)
        WHERE     Plocation = 'Stafford';
```