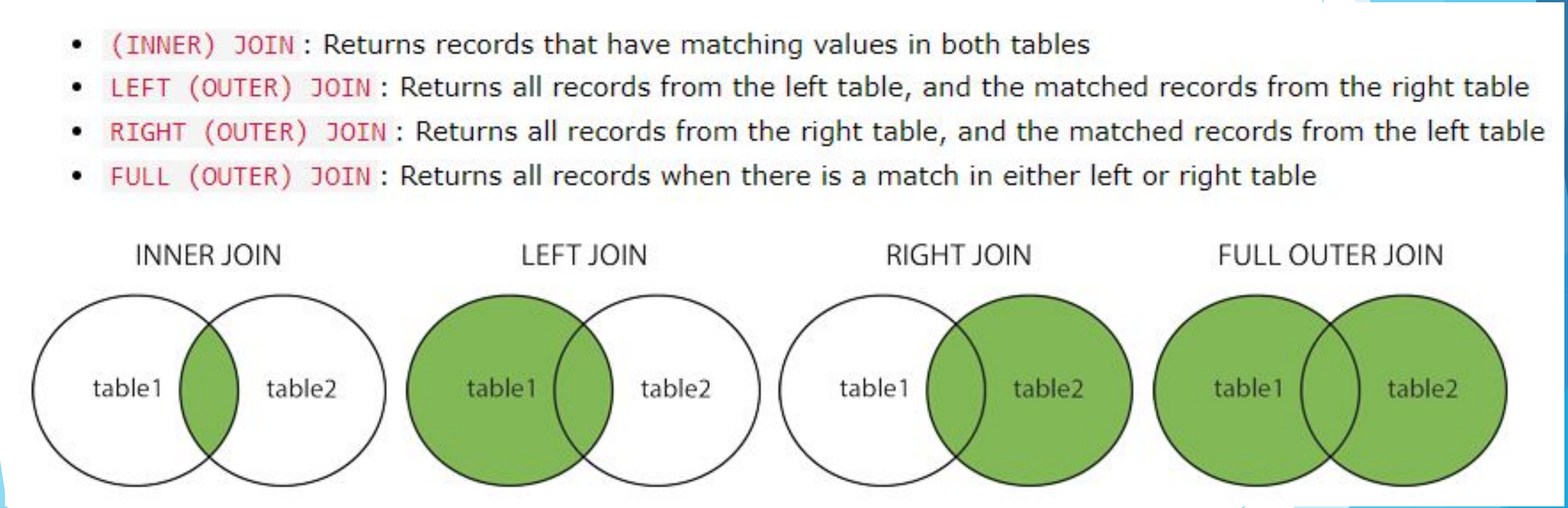
More Complex SQL Retrieval Queries

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



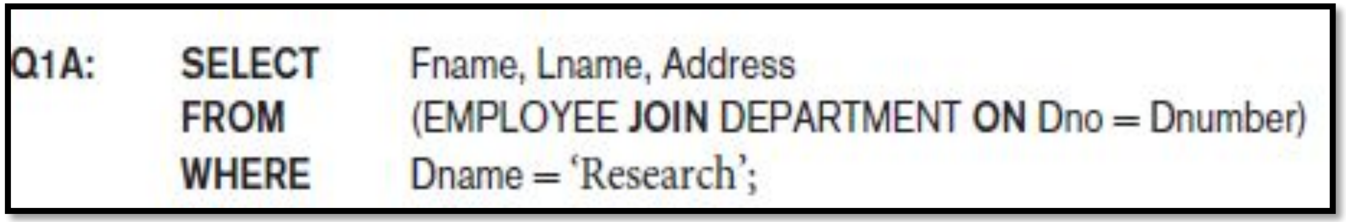
- `(INNER) JOIN` : Returns records that have matching values in both tables
- `LEFT (OUTER) JOIN` : Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN` : Returns all records from the right table, and the matched records from the left table
- `FULL (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***
- 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***
- 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 statements 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.

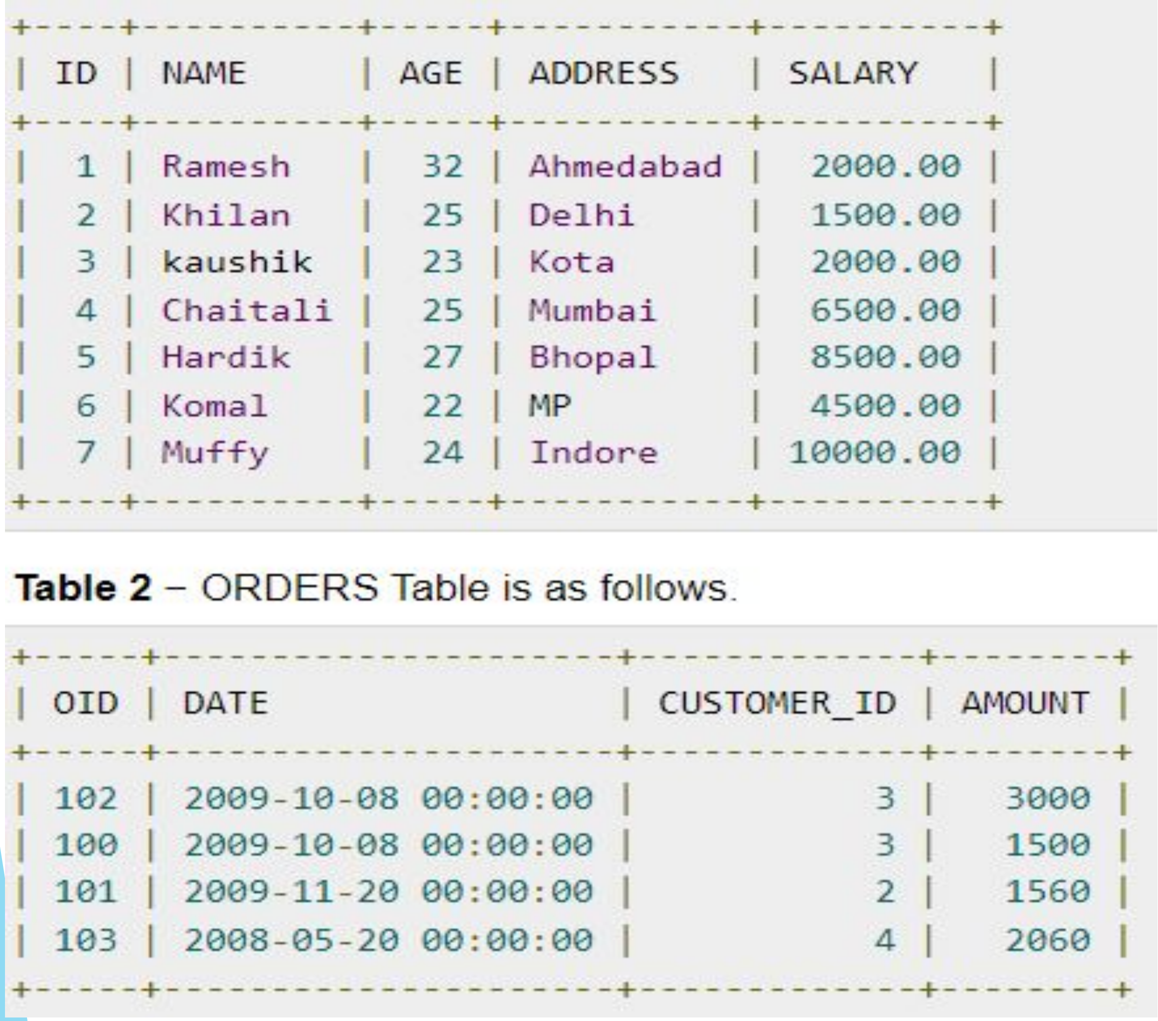


```
Q1A:    SELECT    Fname, Lname, Address
        FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno = Dnumber)
        WHERE     Dname = 'Research';
```

- 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

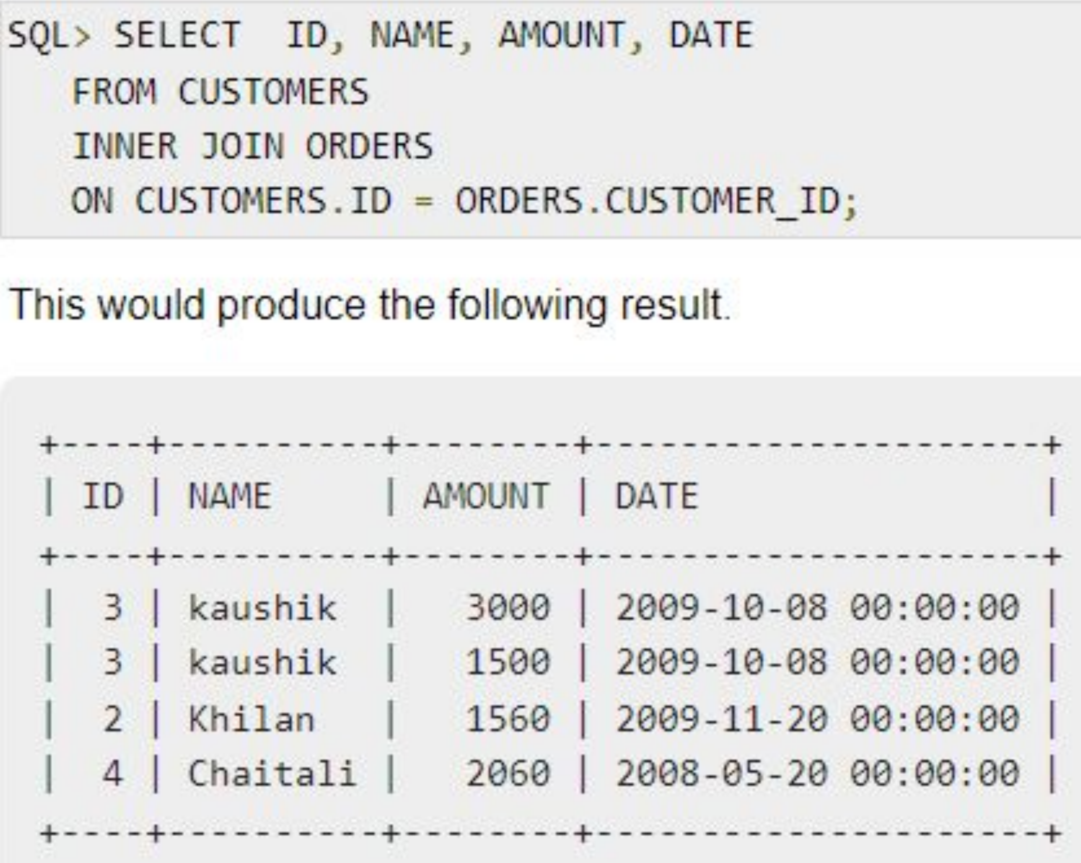
- ***Joined Tables in SQL and Outer Joins - INNER JOIN***



```
+----+----------+-----+-----------+----------+
| 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.

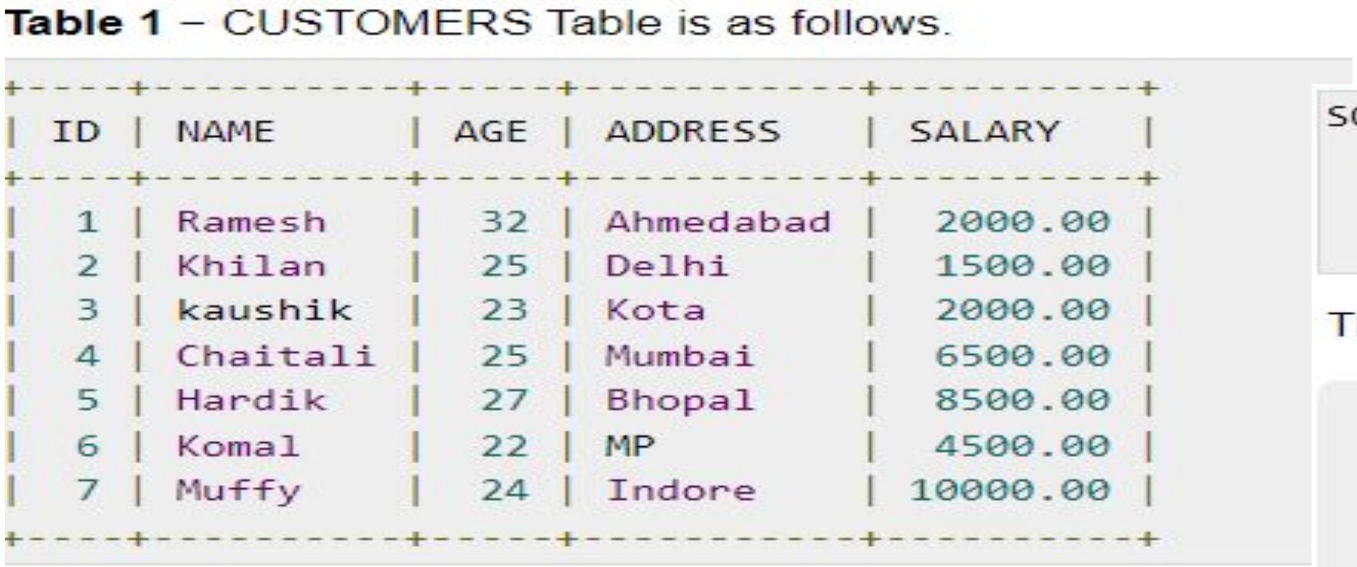


Table 1 – CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

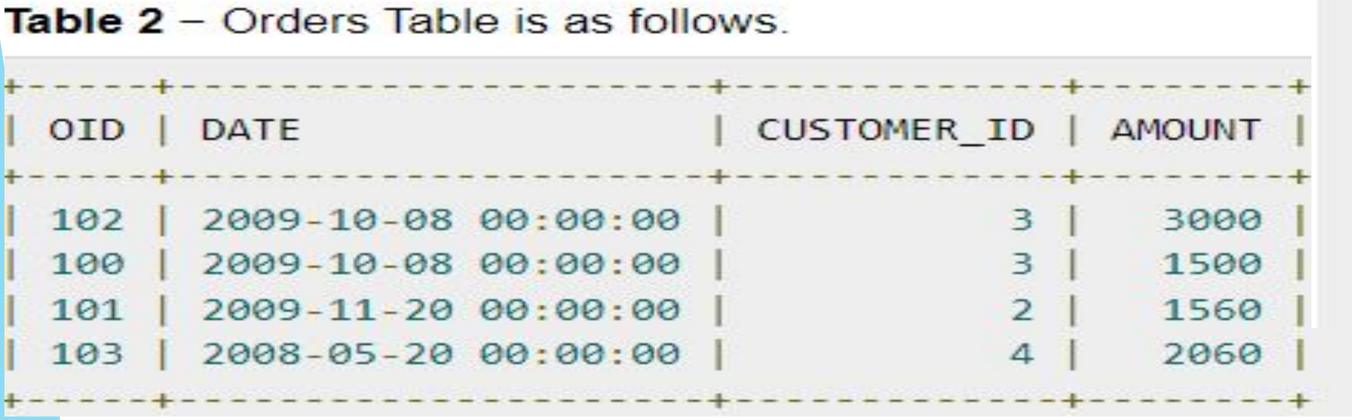
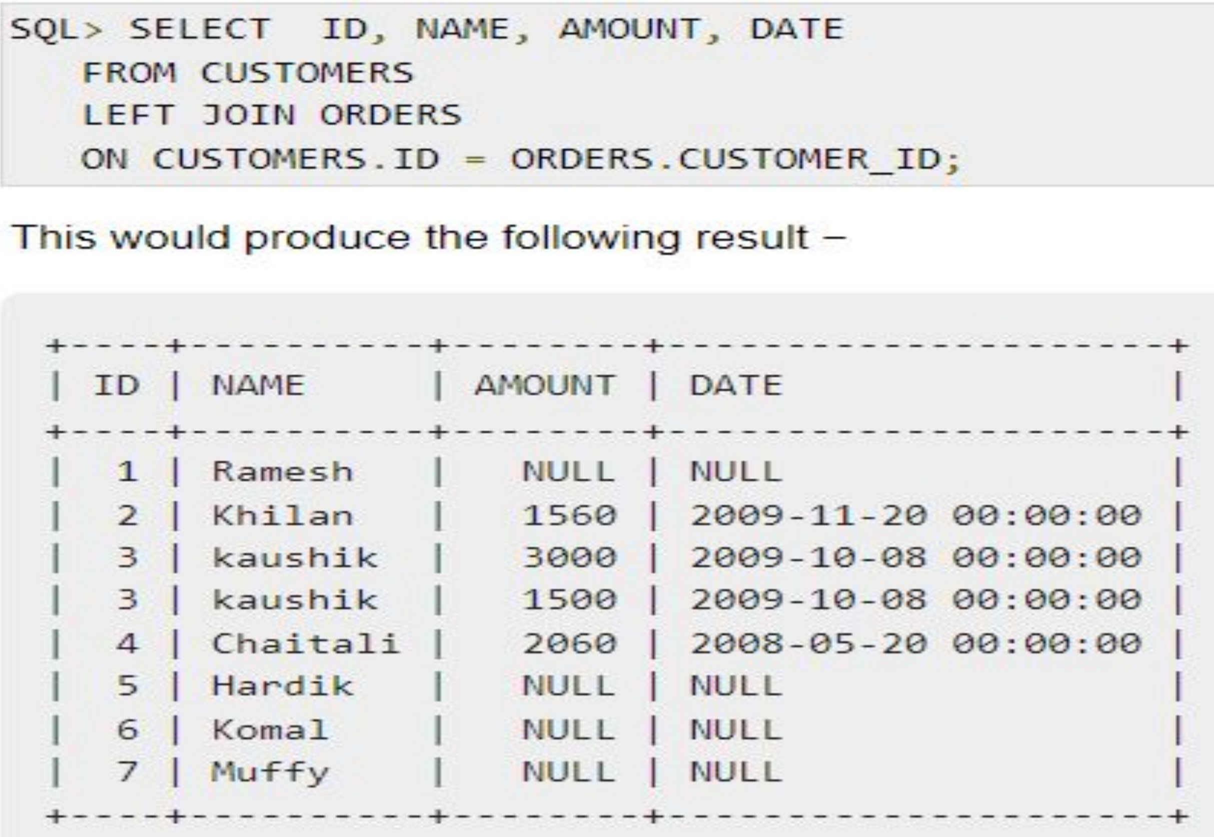


Table 2 – Orders Table is as follows.

```
+-----+---------------------+-------------+--------+
| 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***

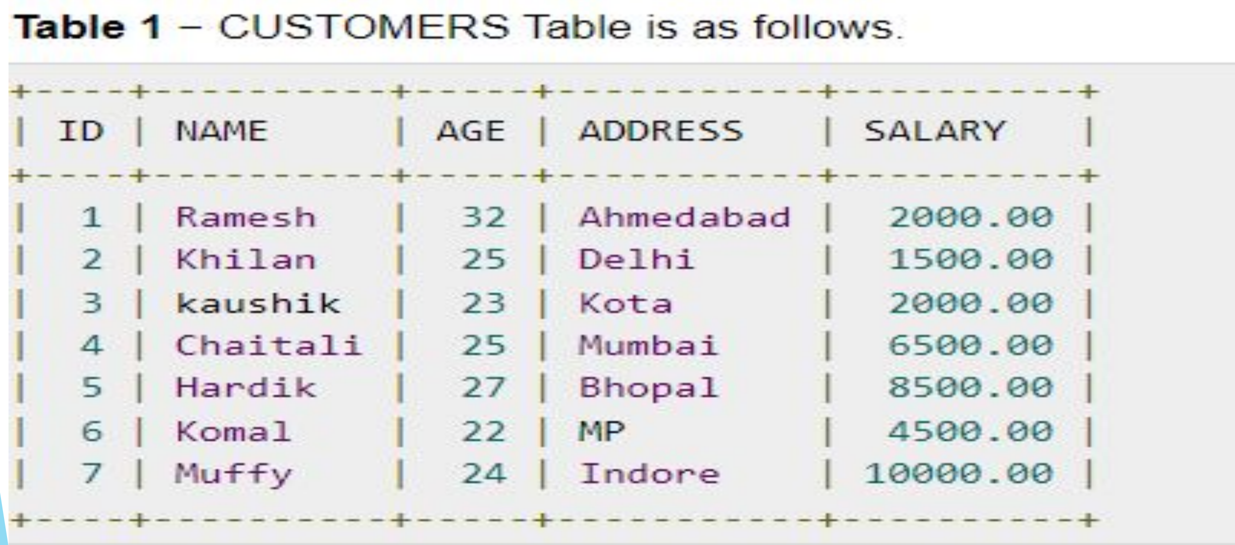


Table 1 – CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

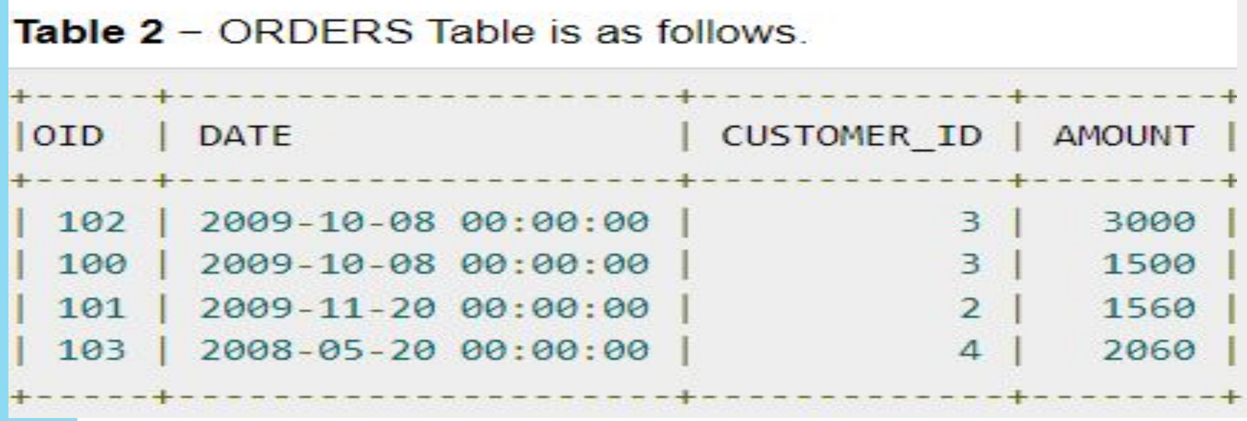
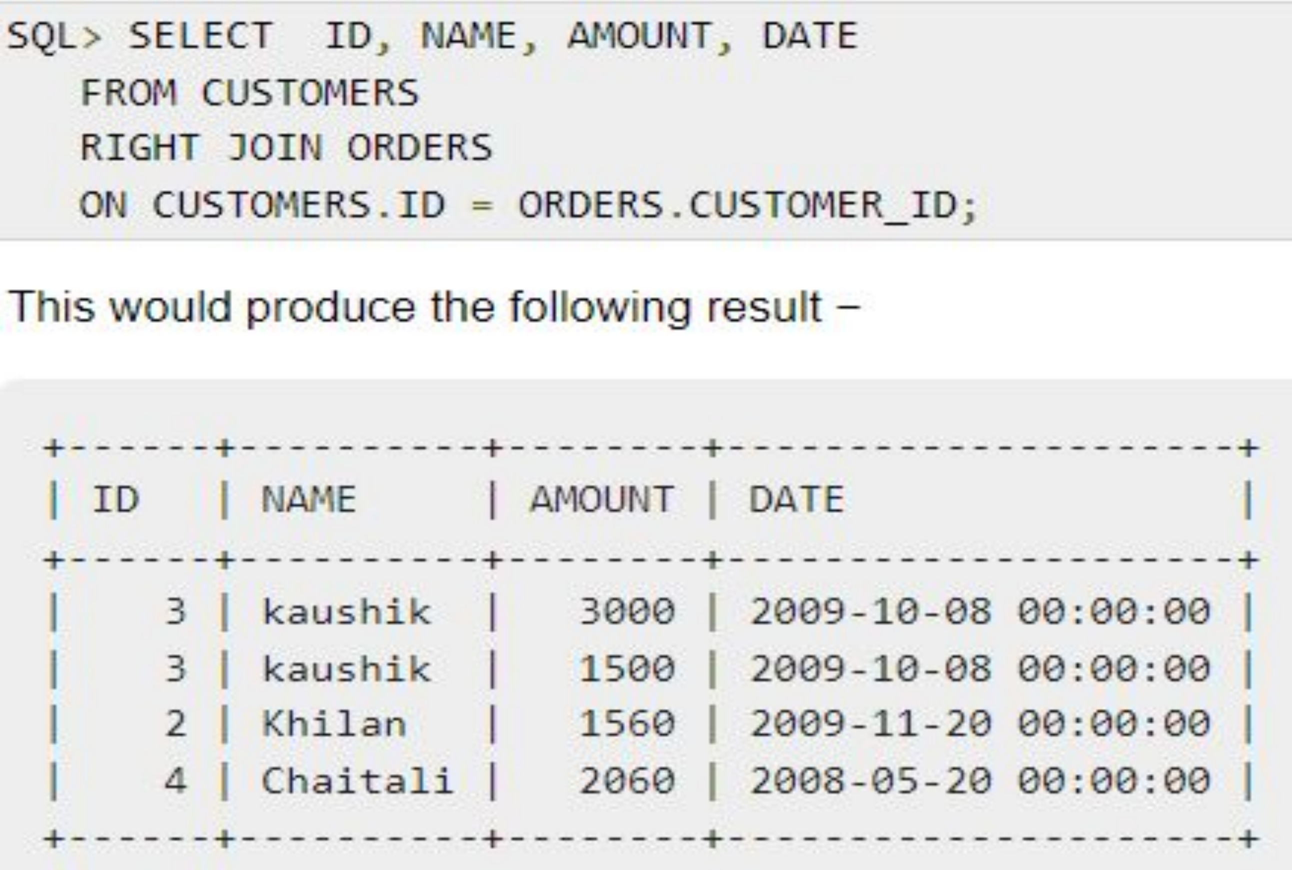


Table 2 – ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+
|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***

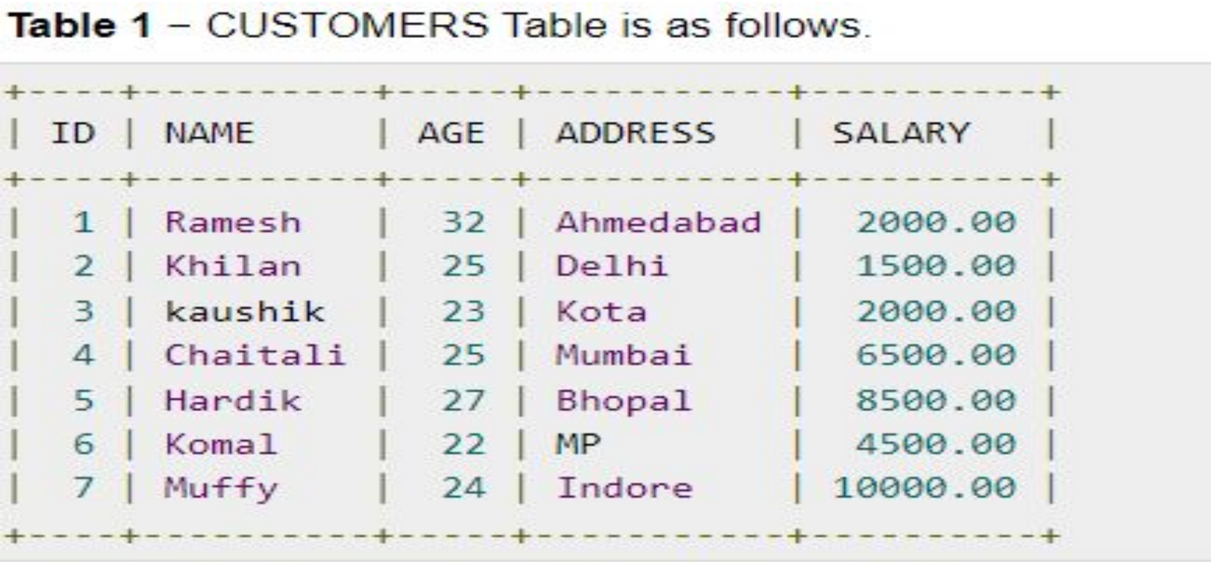


Table 1 – CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

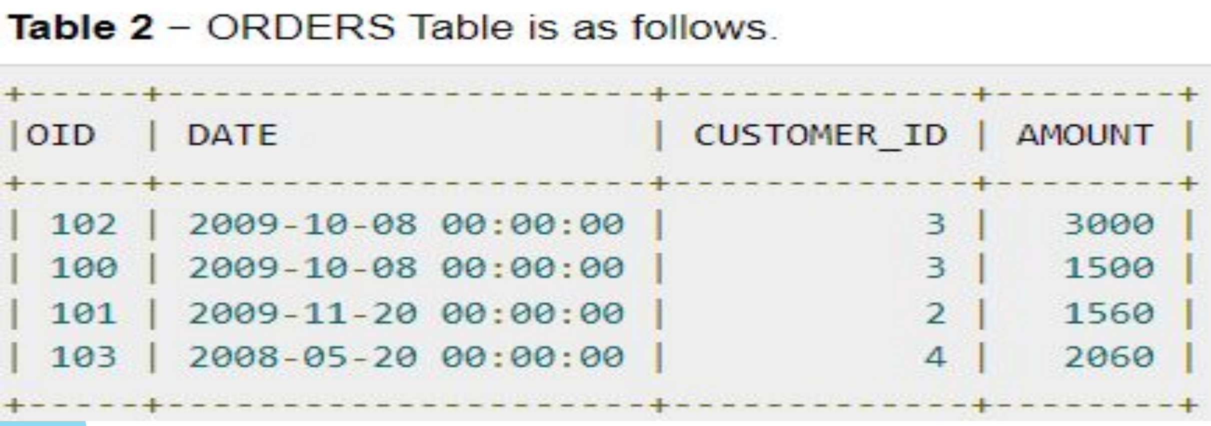
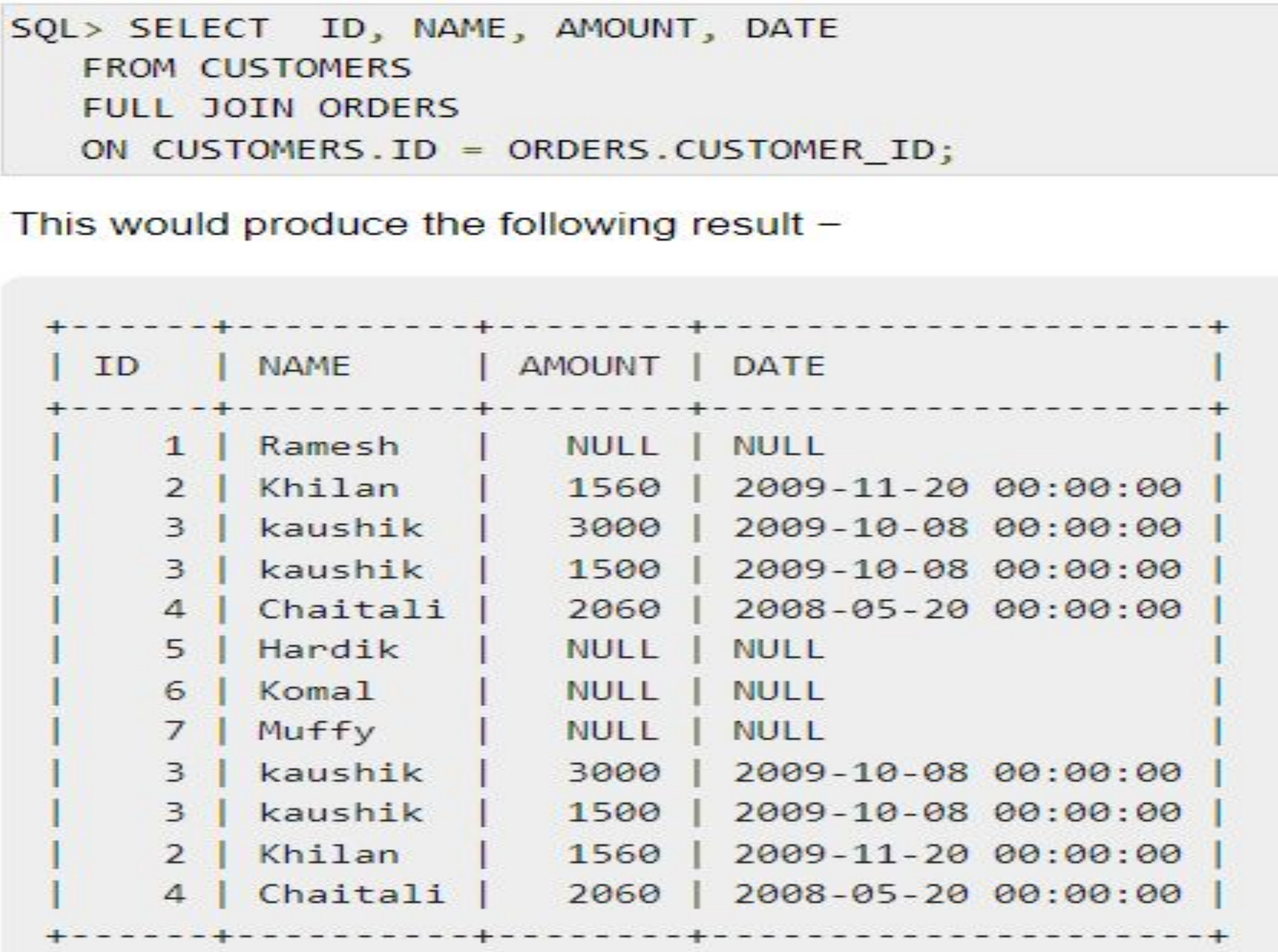


Table 2 – ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+
|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 |
+-----+---------------------+-------------+--------+
```



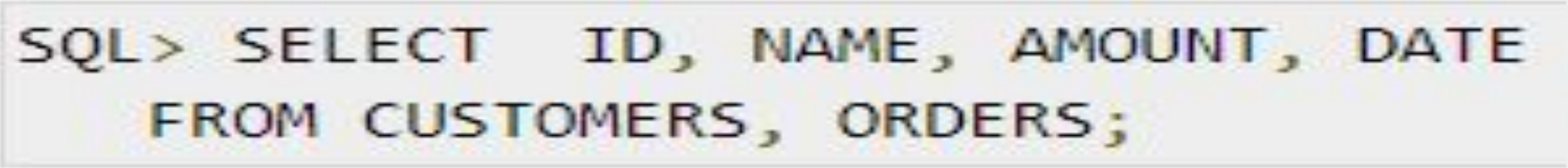
```
SQL> SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   FULL JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

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 |
+------+-----------+--------+---------------------+
```

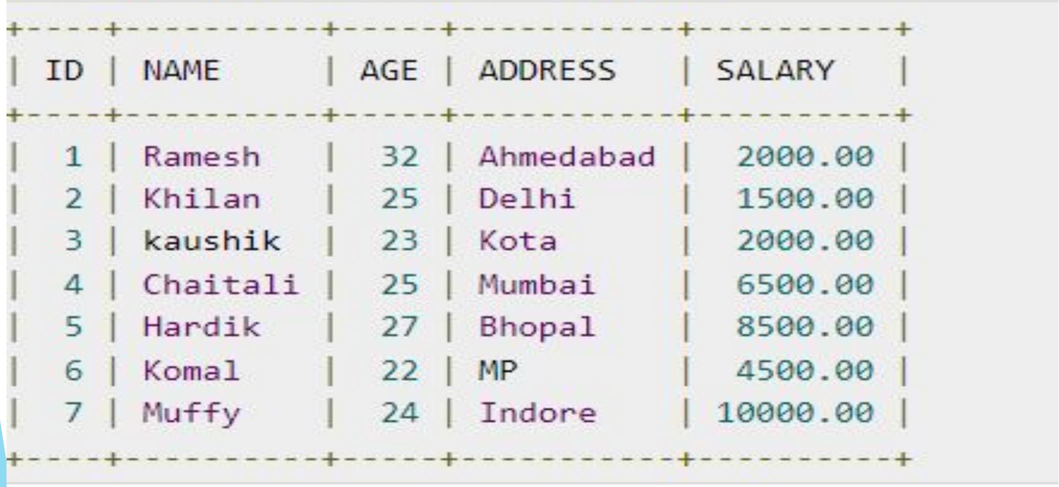
More Complex SQL Retrieval Queries

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



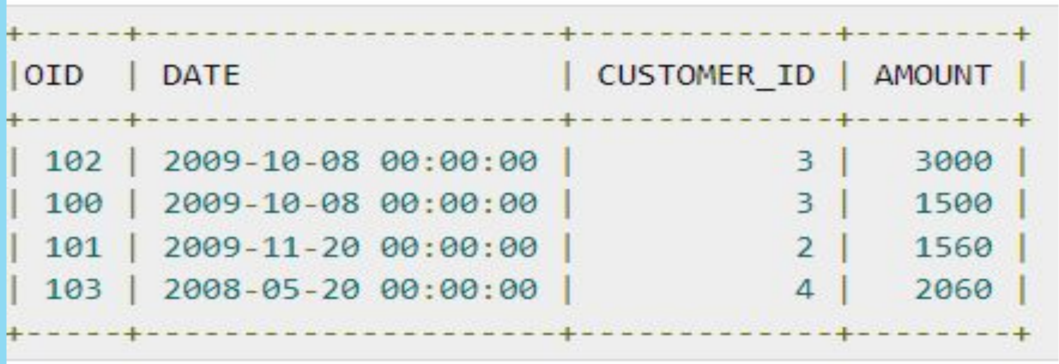
```
SQL> SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS, ORDERS;
```

Table 1 – CUSTOMERS table is as follows.

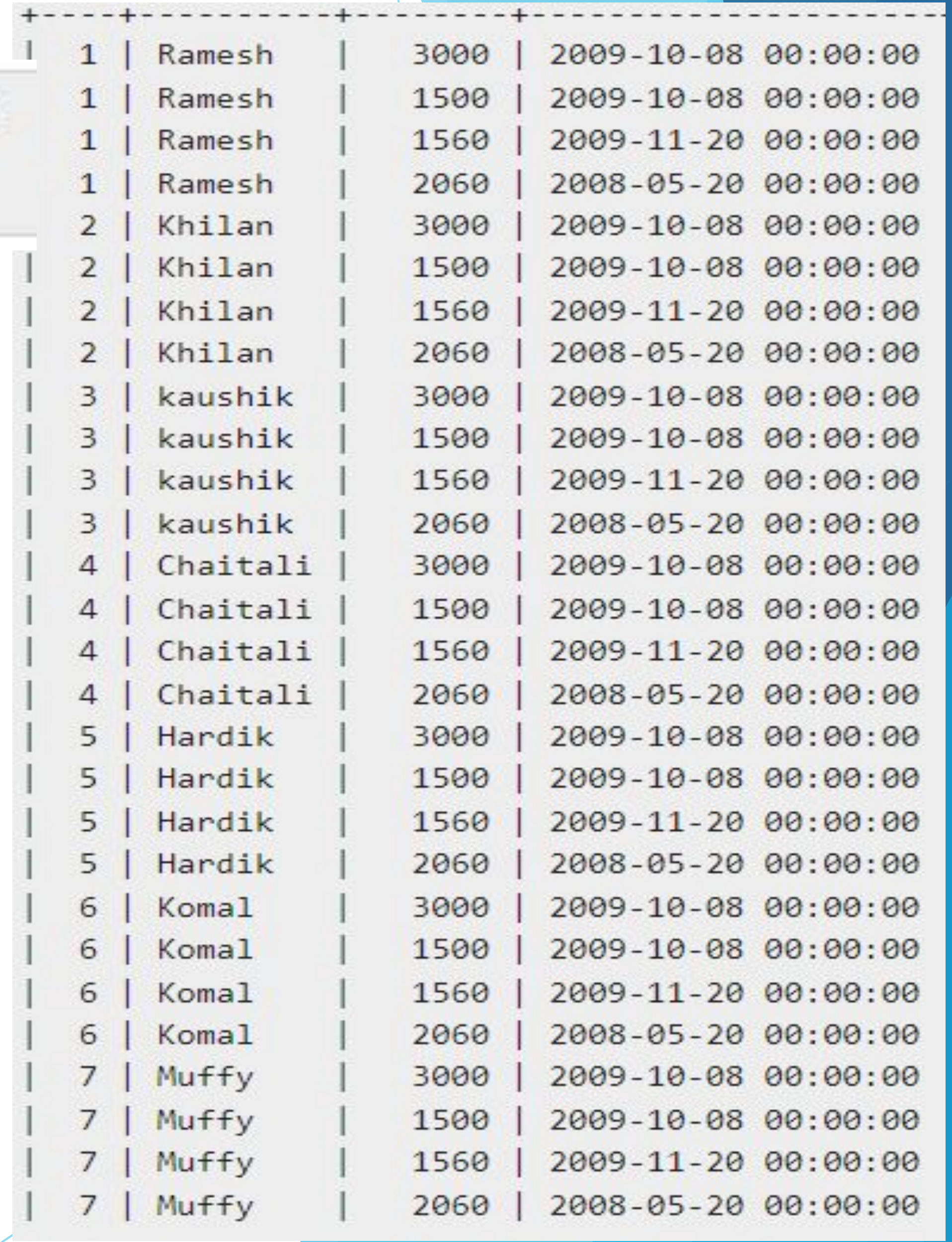


```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  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 –



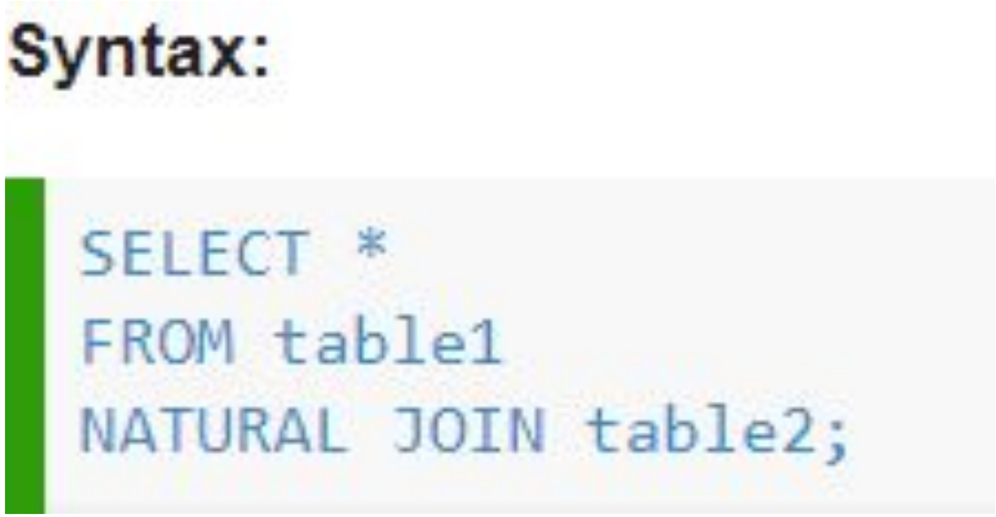
```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           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

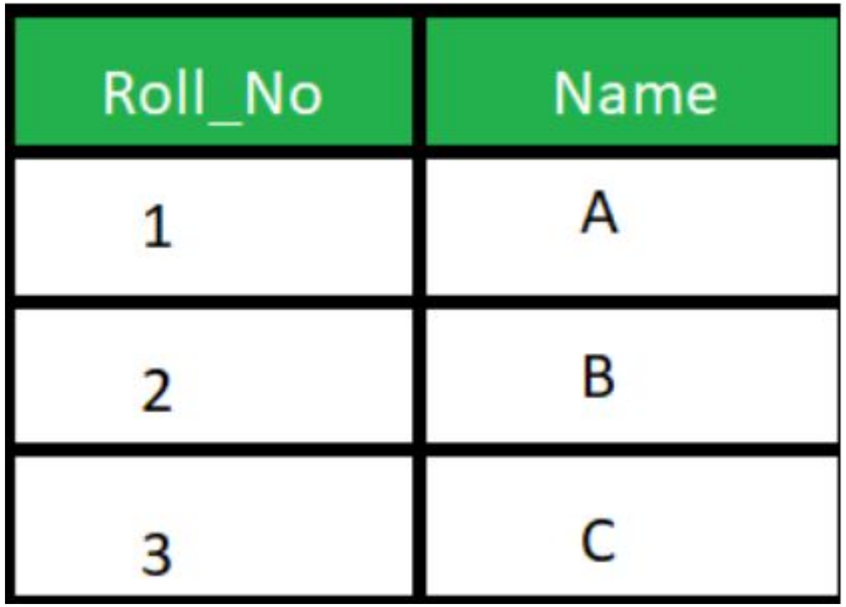


Syntax:

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

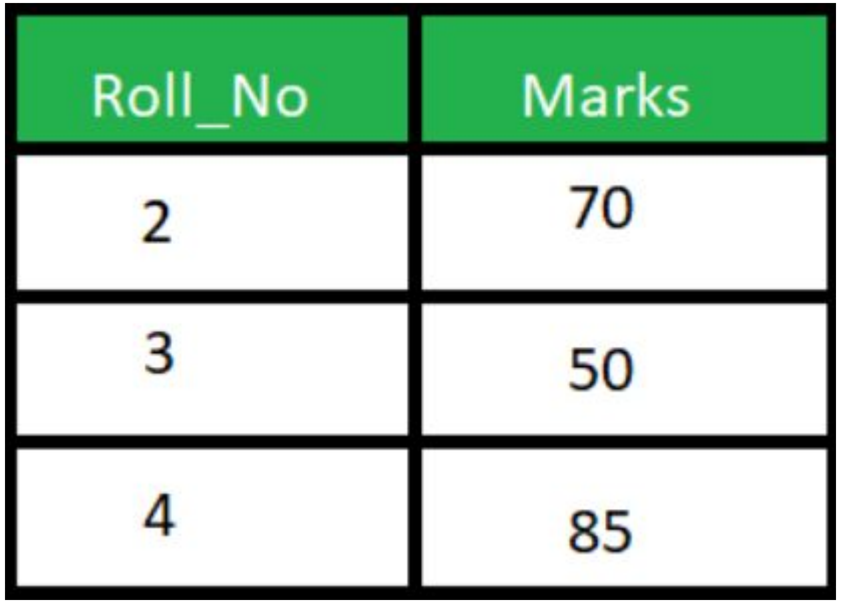
More Complex SQL Retrieval Queries

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



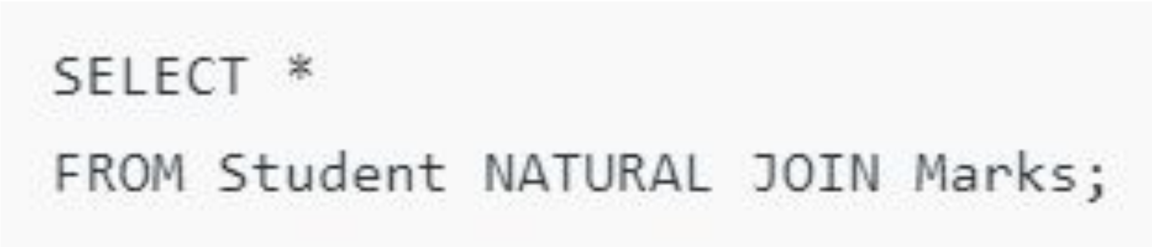
Roll_No	Name
1	A
2	B
3	C

Student table

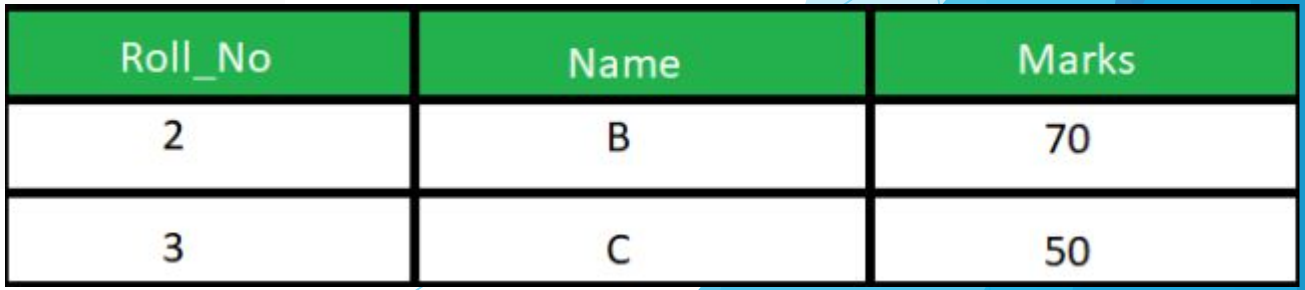


Roll_No	Marks
2	70
3	50
4	85

Marks table



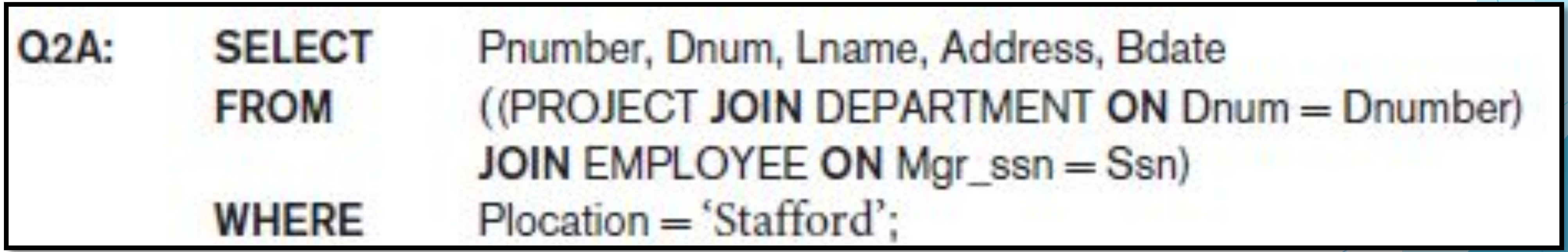
```
SELECT *
FROM Student NATURAL JOIN Marks;
```



Roll_No	Name	Marks
2	B	70
3	C	50

More Complex SQL Retrieval Queries

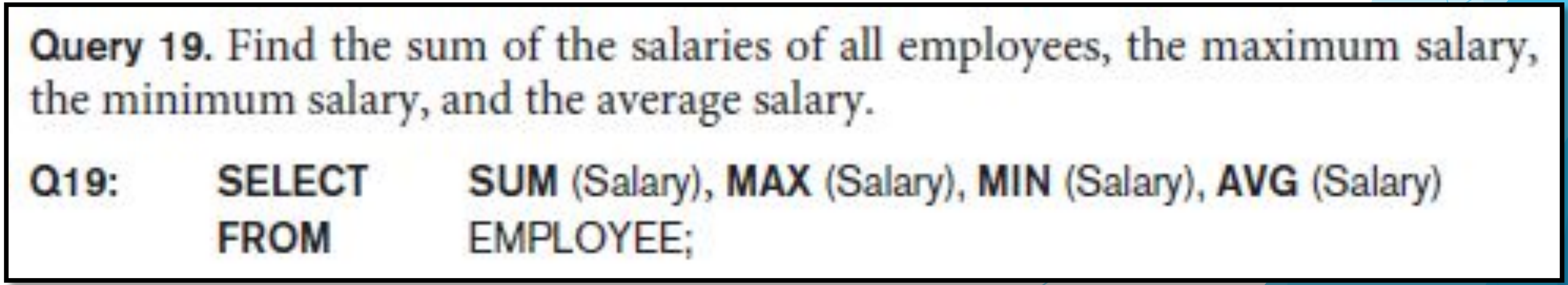
- *Joined Tables in SQL and Outer Joins*
- It is also possible to nest join specifications; that is, one of the tables in a join may itself be a joined table.
- This allows the specification of the join of three or more tables as a single joined table, which is called a **multiway join**.
- For example, 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';
```

More Complex SQL Retrieval Queries

- *Aggregate Functions in SQL*
- **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.
- A number of built-in aggregate functions exist: **COUNT, SUM, MAX, MIN, and AVG.**
  - The **COUNT** function returns the number of tuples or values as specified in a query.
  - 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.

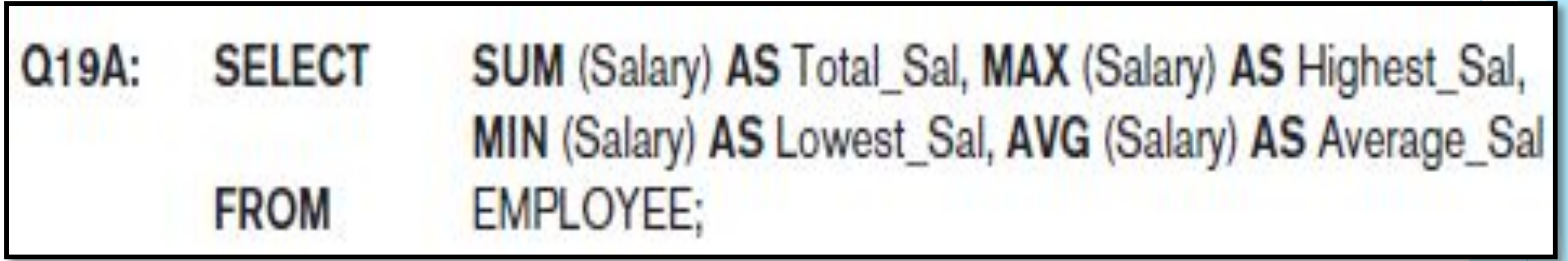


Query 19. Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

```
Q19:    SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
        FROM      EMPLOYEE;
```

More Complex SQL Retrieval Queries

- ***Aggregate Functions in SQL***
- We could use AS to rename the column names in the resulting single-row table; for example, as in Q19A.

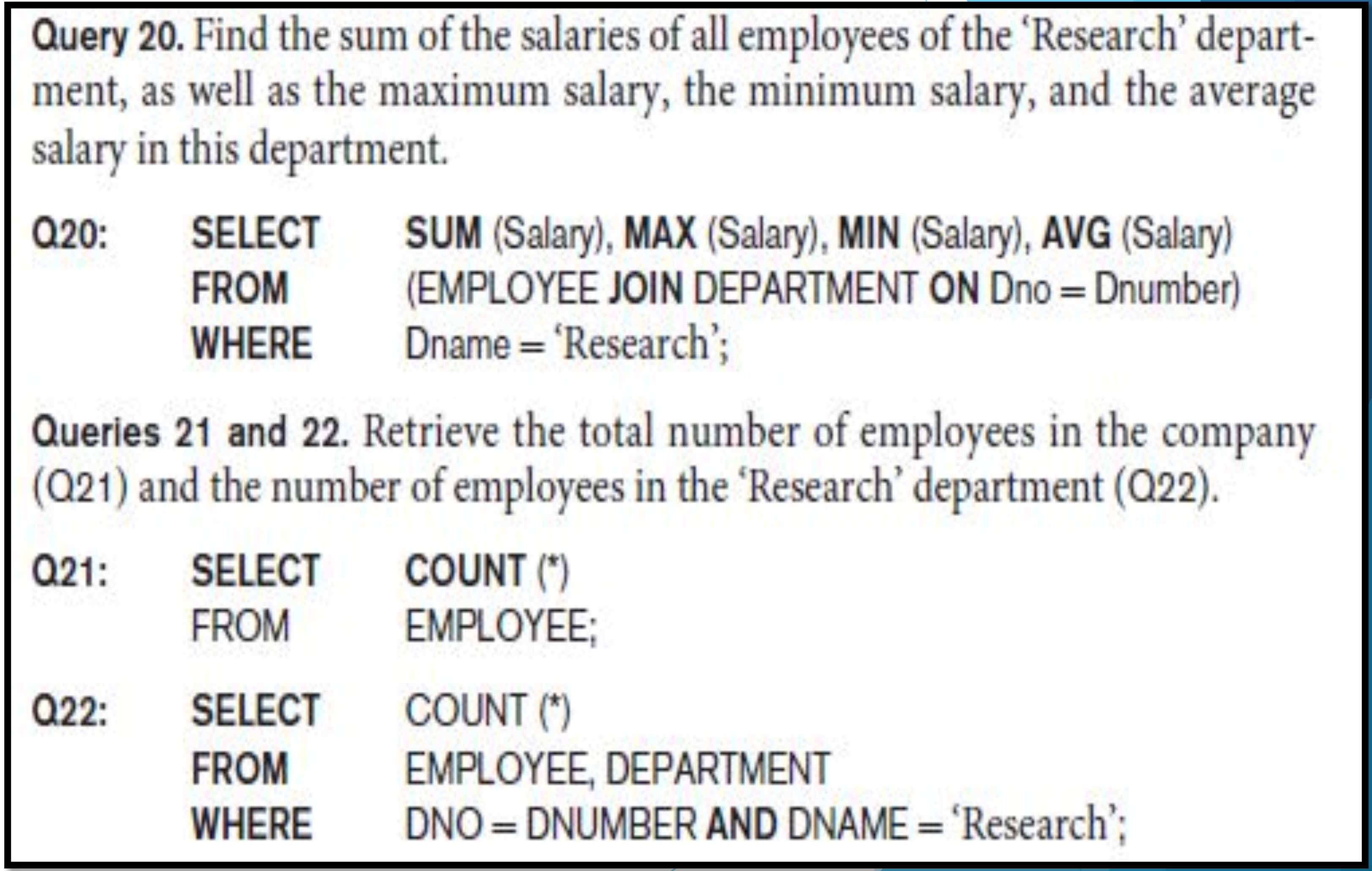


```
Q19A:   SELECT   SUM (Salary) AS Total_Sal, MAX (Salary) AS Highest_Sal,
                 MIN (Salary) AS Lowest_Sal, AVG (Salary) AS Average_Sal
        FROM     EMPLOYEE;
```

More Complex SQL Retrieval Queries

Here the asterisk (*) refers to the rows (tuples), so COUNT (*) returns the number of
rows in the result of the query. This includes NULL values and duplicates.

- ***Aggregate Functions in SQL***
- If we want to get the preceding aggregate function values for employees of a specific department—say, the 'Research' department—we can write Query 20, where the EMPLOYEE tuples are restricted by the WHERE clause to those employees who work for the 'Research' department.



Query 20. Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
Q20:    SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
        FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno = Dnumber)
        WHERE     Dname = 'Research';
```

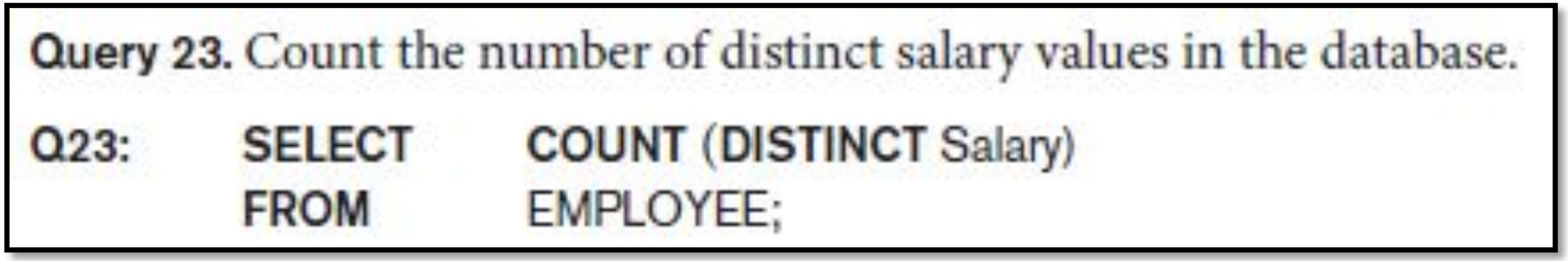
Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Q21:    SELECT    COUNT (*)
        FROM      EMPLOYEE;

Q22:    SELECT    COUNT (*)
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     DNO = DNUMBER AND DNAME = 'Research';
```

More Complex SQL Retrieval Queries

- *Aggregate Functions in SQL*
- We may also use the COUNT function to count values in a column rather than tuples.



Query 23. Count the number of distinct salary values in the database.

```
Q23:    SELECT    COUNT (DISTINCT Salary)
        FROM      EMPLOYEE;
```

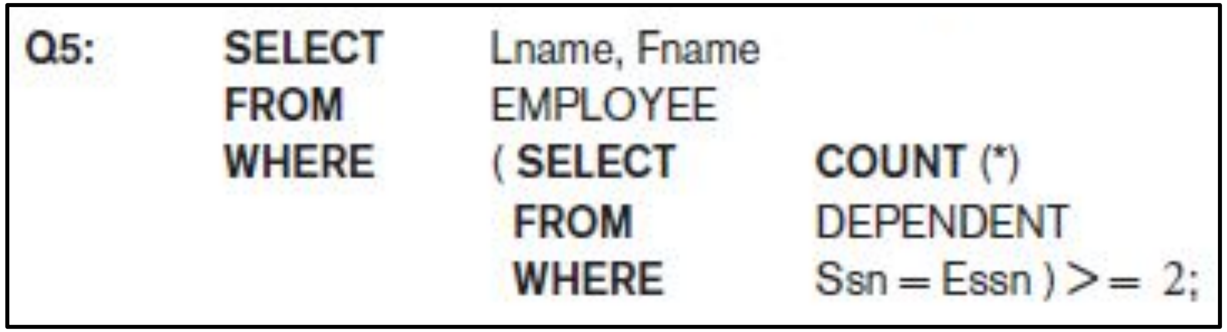
- COUNT(SALARY) – duplicate values won’t eliminate
- COUNT(DISTINCT SALARY) - duplicate values eliminated.
- However, any tuples with NULL for SALARY will not be counted.
- **COUNT(expression) does not count NULL values.**

More Complex SQL Retrieval Queries

- *Aggregate Functions in SQL*
- **MAX and MIN do not count NULL in their data evaluation.** If we have a column containing only dates for instance and there is a NULL date, **MAX and MIN will both ignore that value.**
- **Aggregate functions such as SUM, COUNT, AVG, MAX, and MIN exclude NULL values.**
- In general, NULL values are discarded when aggregate functions are applied to a particular column (attribute); **the only exception is for COUNT(*) because tuples instead of values are counted.**
- The general rule is as follows:
  - aggregate function is applied to a collection of values,
  - NULLs are removed from the collection before the calculation;
  - if the collection becomes empty because all values are NULL,
  - the aggregate function will return NULL (except in the case of COUNT, where it will return 0 for an empty collection of values).

More Complex SQL Retrieval Queries

- ***Aggregate Functions in SQL***
- We can specify a correlated nested query with an aggregate function, and then use the nested query in the WHERE clause of an outer query.
- For example, to retrieve the names of all employees who have two or more dependents (Query 5), we can write the following:



```
Q5:     SELECT    Lname, Fname
        FROM      EMPLOYEE
        WHERE     ( SELECT    COUNT (*)
                    FROM      DEPENDENT
                    WHERE     Ssn = Essn ) >= 2;
```

- The correlated nested query counts the number of dependents that each employee has; if this is greater than or equal to two, the employee tuple is selected.
- SQL also has aggregate functions SOME and ALL that can be applied to a collection of Boolean values; SOME returns TRUE if at least one element in the collection is TRUE, whereas ALL returns TRUE if all elements in the collection are TRUE.

More Complex SQL Retrieval Queries

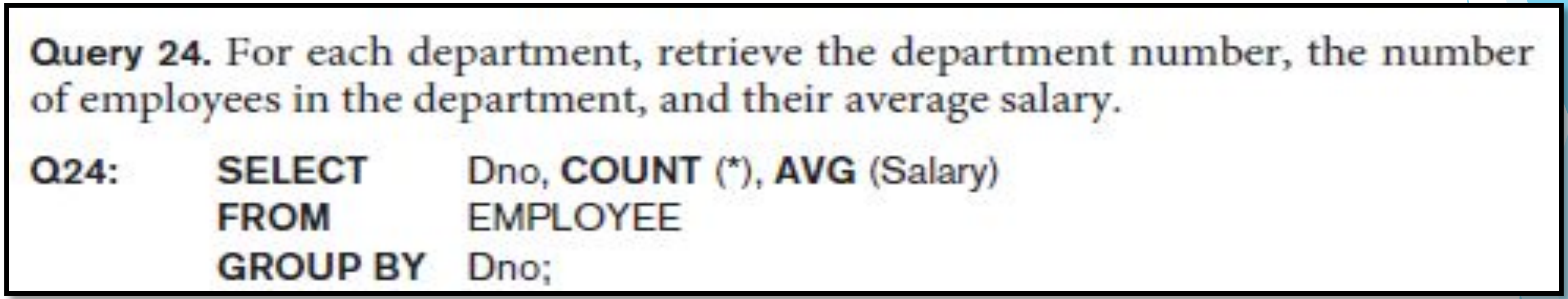
- ***Grouping: The GROUP BY and HAVING Clauses***
- Sometimes we want to apply the aggregate functions to subgroups of tuples in a relation, where the subgrouping is done on some attribute values.
  - For example, we may want to find the average salary of employees in each department or the number of employees who work on each project.
- In these cases, do following:
  - partition the relation into subsets (or groups) of tuples.
  - Each group will consist of the tuples that have the same value of some attribute(s), called the grouping attribute(s).
  - then apply the function to each such group independently
  - to produce summary information about each group.

More Complex SQL Retrieval Queries

- ***Grouping: The GROUP BY and HAVING Clauses***
- 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).
- **The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.**
- **GROUP BY clause in the SELECT statement without using aggregate functions will behave like DISTINCT clause.**

More Complex SQL Retrieval Queries

- *Grouping: The GROUP BY and HAVING Clauses*



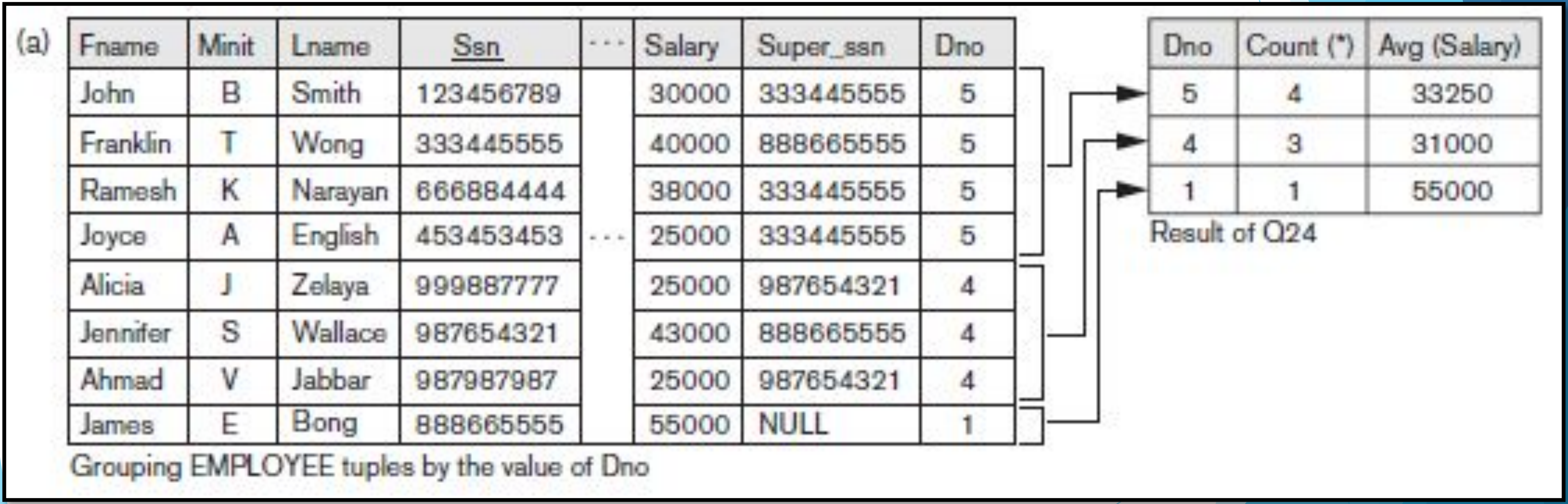
Query 24. For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q24:    SELECT      Dno, COUNT (*), AVG (Salary)
        FROM        EMPLOYEE
        GROUP BY    Dno;
```

- The SELECT clause includes only the grouping attribute and the aggregate functions to be applied on each group of tuples.

More Complex SQL Retrieval Queries

- *Grouping: The GROUP BY and HAVING Clauses*



(a)

Fname	Minit	Lname	Ssn	· · ·	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	· · ·	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

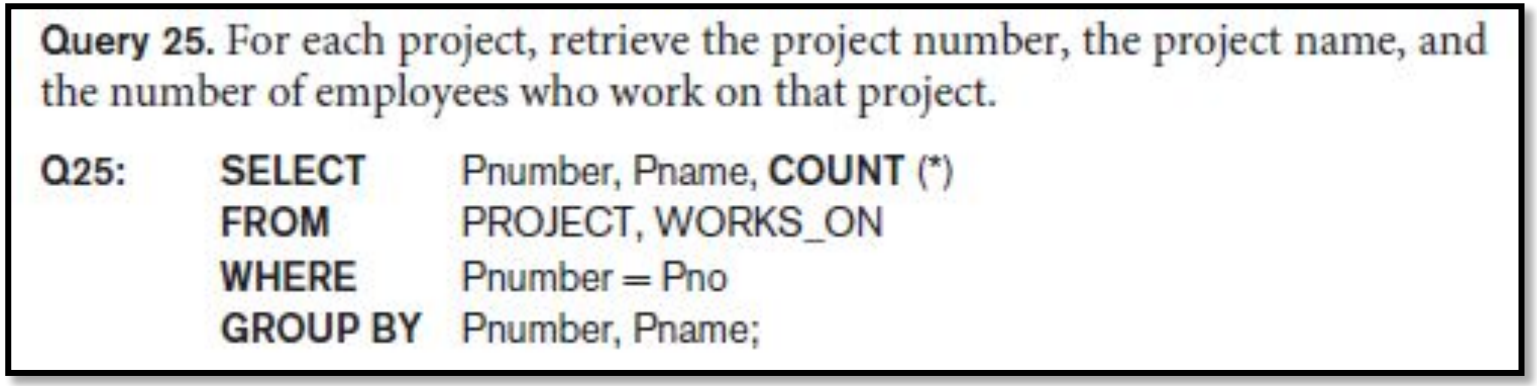
Grouping EMPLOYEE tuples by the value of Dno

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

More Complex SQL Retrieval Queries

- *Grouping: The GROUP BY and HAVING Clauses*
- If NULLs exist in the grouping attribute, then a separate group is created for all tuples with a NULL value in the grouping attribute.
- For example, if the EMPLOYEE table had some tuples that had NULL for the grouping attribute Dno, there would be a separate group for those tuples in the result of Q24.



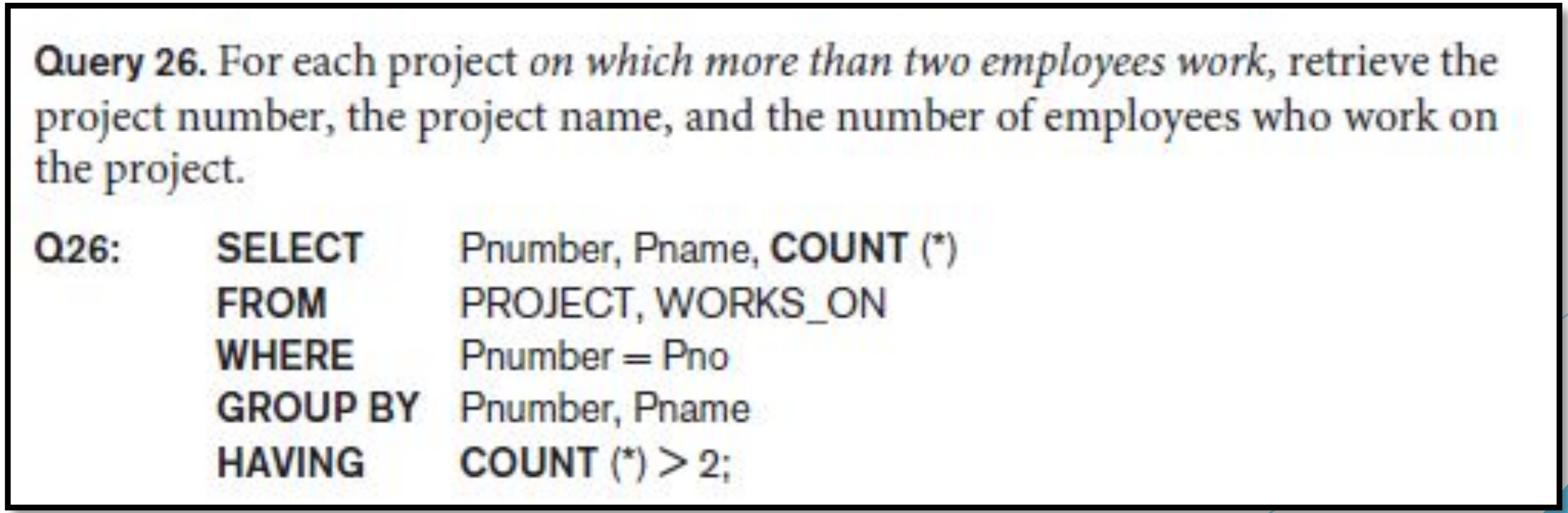
Query 25. For each project, retrieve the project number, the project name, and the number of employees who work on that project.

```
Q25:    SELECT      Pnumber, Pname, COUNT (*)
        FROM        PROJECT, WORKS_ON
        WHERE       Pnumber = Pno
        GROUP BY    Pnumber, Pname;
```

- Above query shows how we can use a join condition in conjunction with GROUP BY.
- In this case, the grouping and functions are applied after the joining of the two relations in the WHERE clause.

More Complex SQL Retrieval Queries

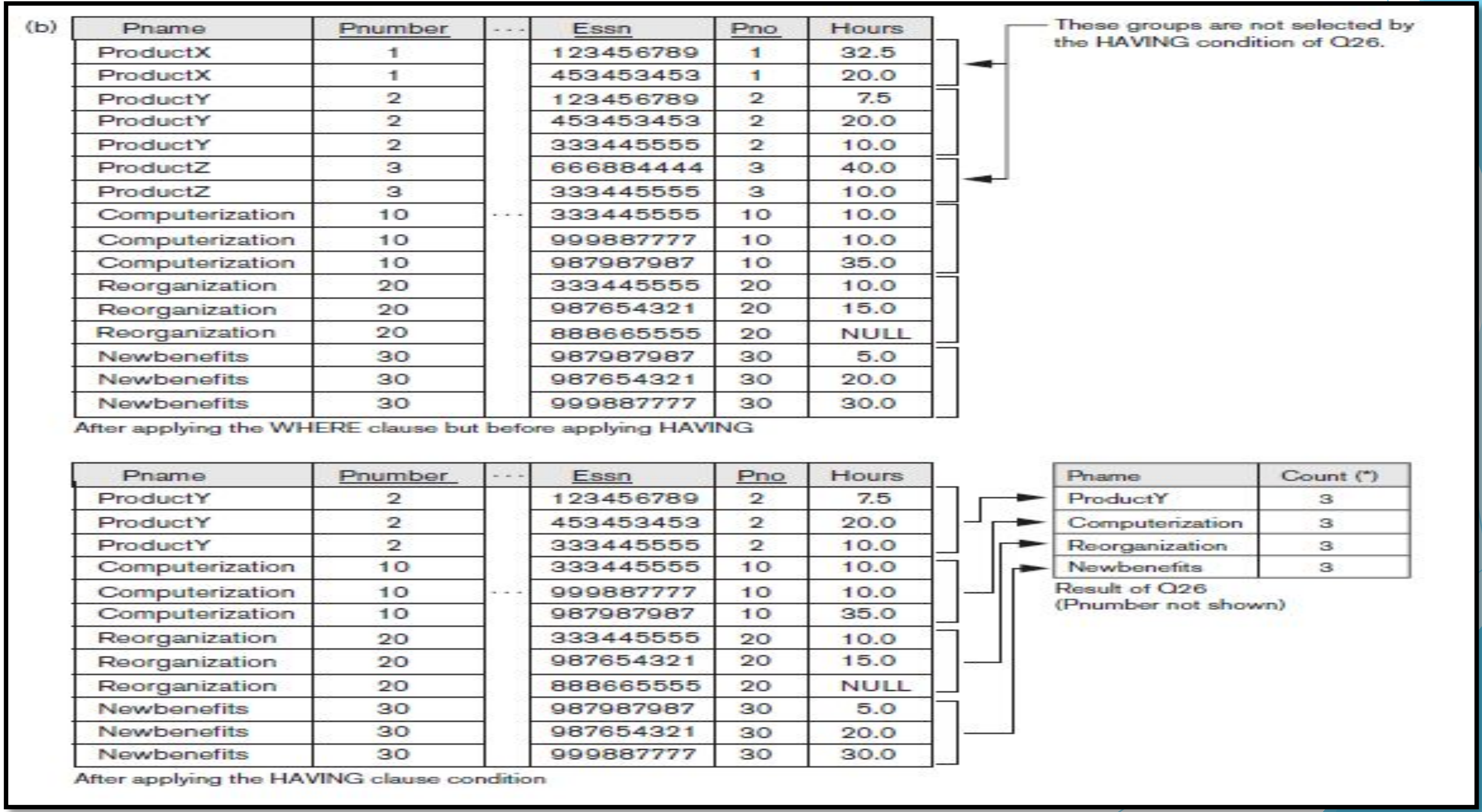
- ***Grouping: The GROUP BY and HAVING Clauses***
- SQL provides a HAVING clause, which can appear in conjunction with a GROUP BY clause.
- HAVING provides a condition on each value of the grouping attributes.
- **The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.**
- Only the groups that satisfy the condition are retrieved in the result of the query. This is illustrated by Query 26.



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;
```

More Complex SQL Retrieval Queries



(b)

Pname	Pnumber	· · ·	Essn	Pno	Hours
ProductX	1		123456789	1	32.5
ProductX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3		333445555	3	10.0
Computerization	10	· · ·	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

These groups are not selected by the HAVING condition of Q26.

After applying the WHERE clause but before applying HAVING

Pname	Pnumber	· · ·	Essn	Pno	Hours
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
Computerization	10		333445555	10	10.0
Computerization	10	· · ·	999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

After applying the HAVING clause condition

Pname	Count (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Result of Q26
(Pnumber not shown)