Chapter 11

GROUP BY / HAVING

Question

- Write the SQL SELECT statement to return the highest salary of all employees in the employees table?

```
SELECT MAX(salary) FROM employees;
```

Question

- Write the SQL statement to return the highest salary in each department?

SELECT Statements

- Up to now, you would have to write a number of different SQL statements to accomplish this:

```
SELECT MAX(salary) FROM employees WHERE dept_id = 50;
SELECT MAX(salary) FROM employees WHERE dept_id = 60;
SELECT MAX(salary) FROM employees WHERE dept_id = 90;
```

- And so on

GROUP BY and HAVING Clauses

- GROUP BY and HAVING clauses make this easier

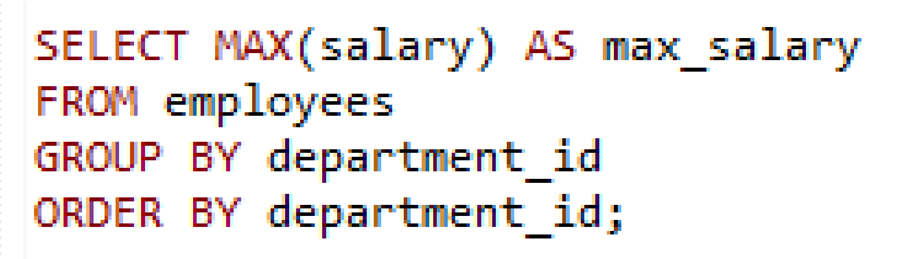
GROUP BY and HAVING Clauses

- The GROUP BY clause works with group (aggregate) functions (SUM, AVG, MIN, MAX, COUNT) to group data in the result set by columns

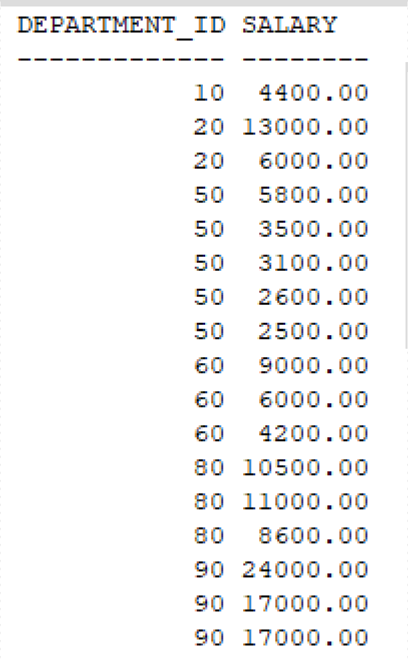
- The HAVING clause is used to restrict rows in the result set after group (aggregate) functions have been applied to the grouped rows

GROUP BY Example

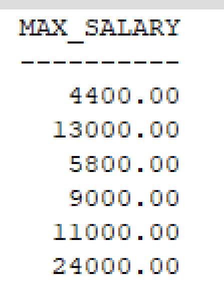
- Return the highest salary in each department
  - Rows are grouped by department_id
  - The MAX function is then applied to each group



```
SELECT MAX(salary) AS max_salary
FROM employees
GROUP BY department_id
ORDER BY department_id;
```



```
DEPARTMENT_ID SALARY
------------- --------
           10  4400.00
           20 13000.00
           20  6000.00
           50  5800.00
           50  3500.00
           50  3100.00
           50  2600.00
           50  2500.00
           60  9000.00
           60  6000.00
           60  4200.00
           80 10500.00
           80 11000.00
           80  8600.00
           90 24000.00
           90 17000.00
           90 17000.00
```

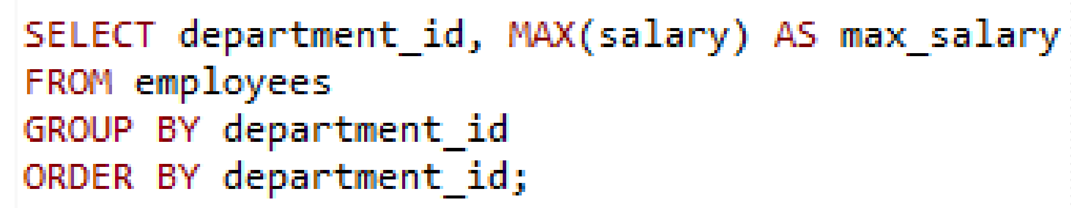


```
MAX_SALARY
----------
   4400.00
  13000.00
   5800.00
   9000.00
  11000.00
  24000.00
```

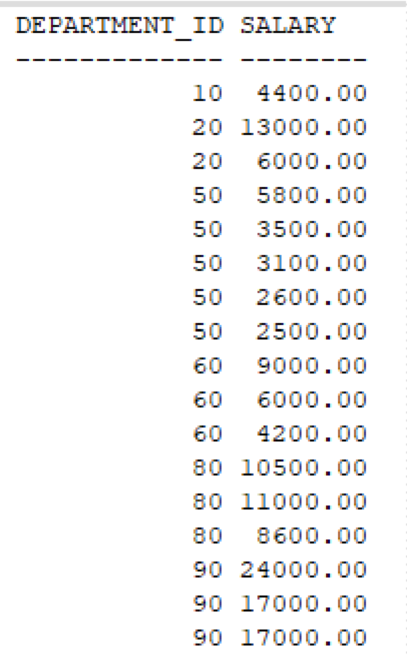
- Which MAX salary belongs to which department?

GROUP BY Example

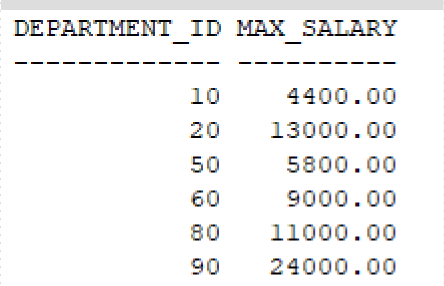
- Include the GROUP BY column in the SELECT column-list
- Cannot use a column alias in the GROUP BY clause



```
SELECT department_id, MAX(salary) AS max_salary
FROM employees
GROUP BY department_id
ORDER BY department_id;
```



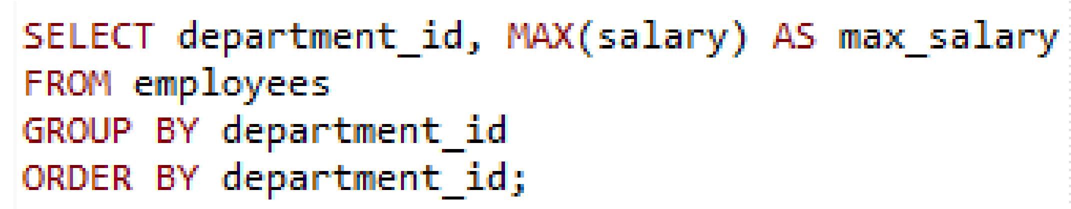
```
DEPARTMENT_ID SALARY
------------- --------
           10  4400.00
           20 13000.00
           20  6000.00
           50  5800.00
           50  3500.00
           50  3100.00
           50  2600.00
           50  2500.00
           60  9000.00
           60  6000.00
           60  4200.00
           80 10500.00
           80 11000.00
           80  8600.00
           90 24000.00
           90 17000.00
           90 17000.00
```



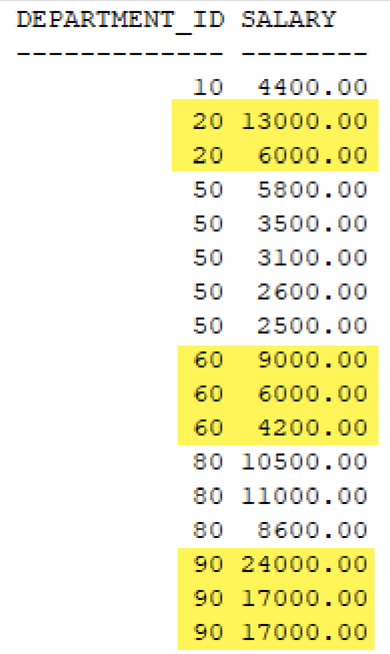
```
DEPARTMENT_ID MAX_SALARY
------------- ----------
           10    4400.00
           20   13000.00
           50    5800.00
           60    9000.00
           80   11000.00
           90   24000.00
```

GROUP BY

- The GROUP BY clause divides the rows into smaller groups by department_id



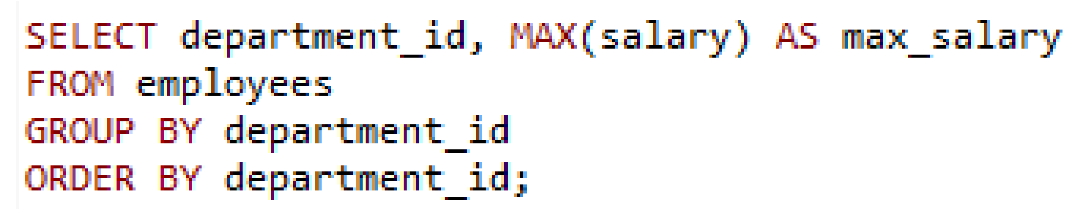
```
SELECT department_id, MAX(salary) AS max_salary
FROM employees
GROUP BY department_id
ORDER BY department_id;
```



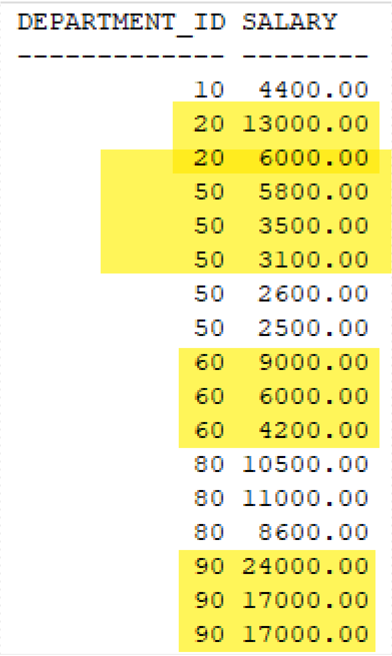
```
DEPARTMENT_ID SALARY
------------- --------
           10  4400.00
           20 13000.00
           20  6000.00
           50  5800.00
           50  3500.00
           50  3100.00
           50  2600.00
           50  2500.00
           60  9000.00
           60  6000.00
           60  4200.00
           80 10500.00
           80 11000.00
           80  8600.00
           90 24000.00
           90 17000.00
           90 17000.00
```

GROUP BY

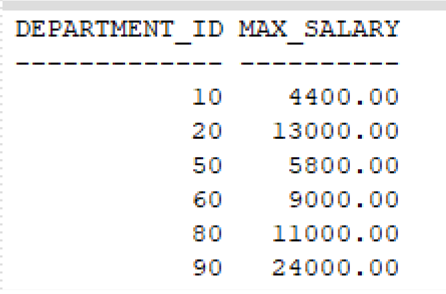
- The MAX group function is applied to each group, returning the max (highest) salary for each group



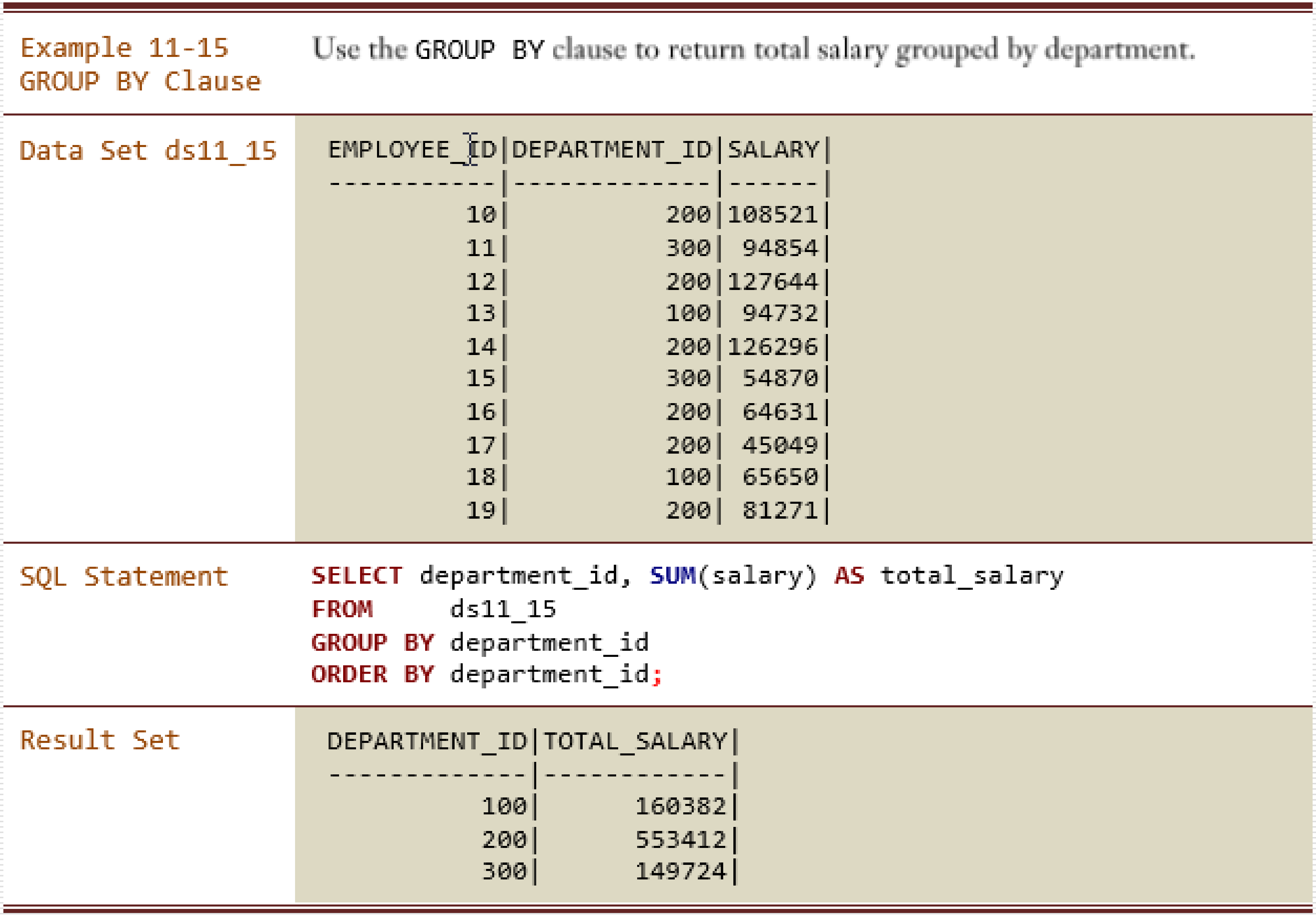
```sql
SELECT department_id, MAX(salary) AS max_salary
FROM employees
GROUP BY department_id
ORDER BY department_id;
```



```
DEPARTMENT_ID SALARY
------------- --------
           10  4400.00
           20 13000.00
           20  6000.00
           50  5800.00
           50  3500.00
           50  3100.00
           50  2600.00
           50  2500.00
           60  9000.00
           60  6000.00
           60  4200.00
           80 10500.00
           80 11000.00
           80  8600.00
           90 24000.00
           90 17000.00
           90 17000.00
```



```
DEPARTMENT_ID MAX_SALARY
------------- ----------
           10    4400.00
           20   13000.00
           50    5800.00
           60    9000.00
           80   11000.00
           90   24000.00
```



Example 11-15
GROUP BY Clause

Use the GROUP BY clause to return total salary grouped by department.

Data Set ds11_15

```
EMPLOYEE_ID|DEPARTMENT_ID|SALARY|
-----------|-------------|------|
         10|          200|108521|
         11|          300| 94854|
         12|          200|127644|
         13|          100| 94732|
         14|          200|126296|
         15|          300| 54870|
         16|          200| 64631|
         17|          200| 45049|
         18|          100| 65650|
         19|          200| 81271|
```

SQL Statement

```
SELECT department_id, SUM(salary) AS total_salary
FROM     ds11_15
GROUP BY department_id
ORDER BY department_id;
```

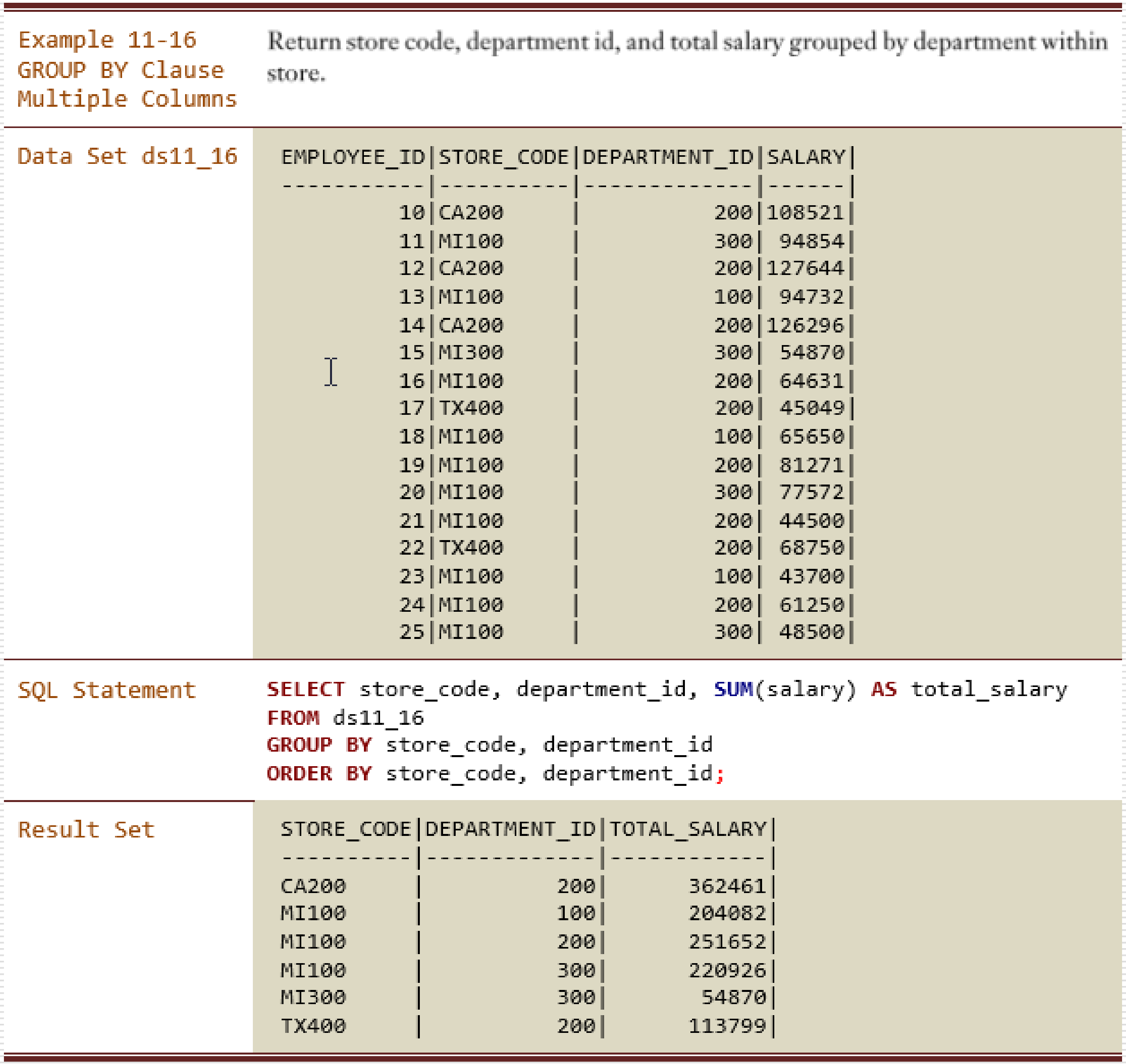
Result Set

```
DEPARTMENT_ID|TOTAL_SALARY|
-------------|------------|
          100|      160382|
          200|      553412|
          300|      149724|
```

GROUP BY Clause on Multiple Columns

- May need to divide groups into smaller groups

- Total salary by department within each store



Example 11-16
GROUP BY Clause
Multiple Columns

Return store code, department id, and total salary grouped by department within store.

Data Set ds11_16

```
EMPLOYEE_ID|STORE_CODE|DEPARTMENT_ID|SALARY|
-----------|----------|-------------|------|
         10|CA200     |          200|108521|
         11|MI100     |          300| 94854|
         12|CA200     |          200|127644|
         13|MI100     |          100| 94732|
         14|CA200     |          200|126296|
         15|MI300     |          300| 54870|
         16|MI100     |          200| 64631|
         17|TX400     |          200| 45049|
         18|MI100     |          100| 65650|
         19|MI100     |          200| 81271|
         20|MI100     |          300| 77572|
         21|MI100     |          200| 44500|
         22|TX400     |          200| 68750|
         23|MI100     |          100| 43700|
         24|MI100     |          200| 61250|
         25|MI100     |          300| 48500|
```

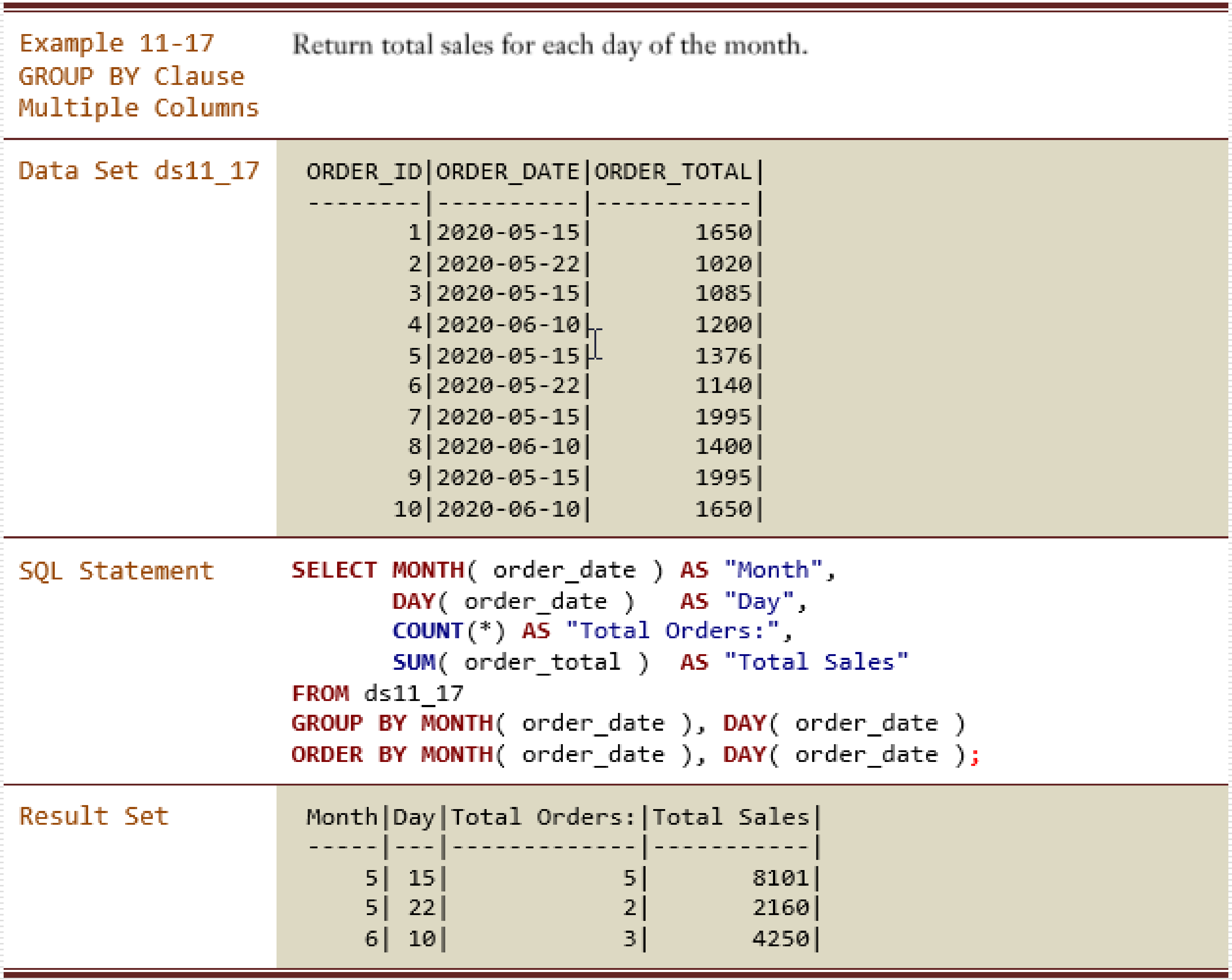
SQL Statement

```sql
SELECT store_code, department_id, SUM(salary) AS total_salary
FROM ds11_16
GROUP BY store_code, department_id
ORDER BY store_code, department_id;
```

Result Set

```
STORE_CODE|DEPARTMENT_ID|TOTAL_SALARY|
----------|-------------|------------|
CA200     |          200|      362461|
MI100     |          100|      204082|
MI100     |          200|      251652|
MI100     |          300|      220926|
MI300     |          300|       54870|
TX400     |          200|      113799|
```

GROUP BY Clause with Dates



Example 11-17
GROUP BY Clause
Multiple Columns

Return total sales for each day of the month.

Data Set ds11_17

```
ORDER_ID|ORDER_DATE|ORDER_TOTAL|
--------|----------|-----------|
       1|2020-05-15|       1650|
       2|2020-05-22|       1020|
       3|2020-05-15|       1085|
       4|2020-06-10|       1200|
       5|2020-05-15|       1376|
       6|2020-05-22|       1140|
       7|2020-05-15|       1995|
       8|2020-06-10|       1400|
       9|2020-05-15|       1995|
      10|2020-06-10|       1650|
```

SQL Statement

```
SELECT MONTH( order_date ) AS "Month",
       DAY( order_date )   AS "Day",
       COUNT(*) AS "Total Orders:",
       SUM( order_total )  AS "Total Sales"
FROM ds11_17
GROUP BY MONTH( order_date ), DAY( order_date )
ORDER BY MONTH( order_date ), DAY( order_date );
```

Result Set

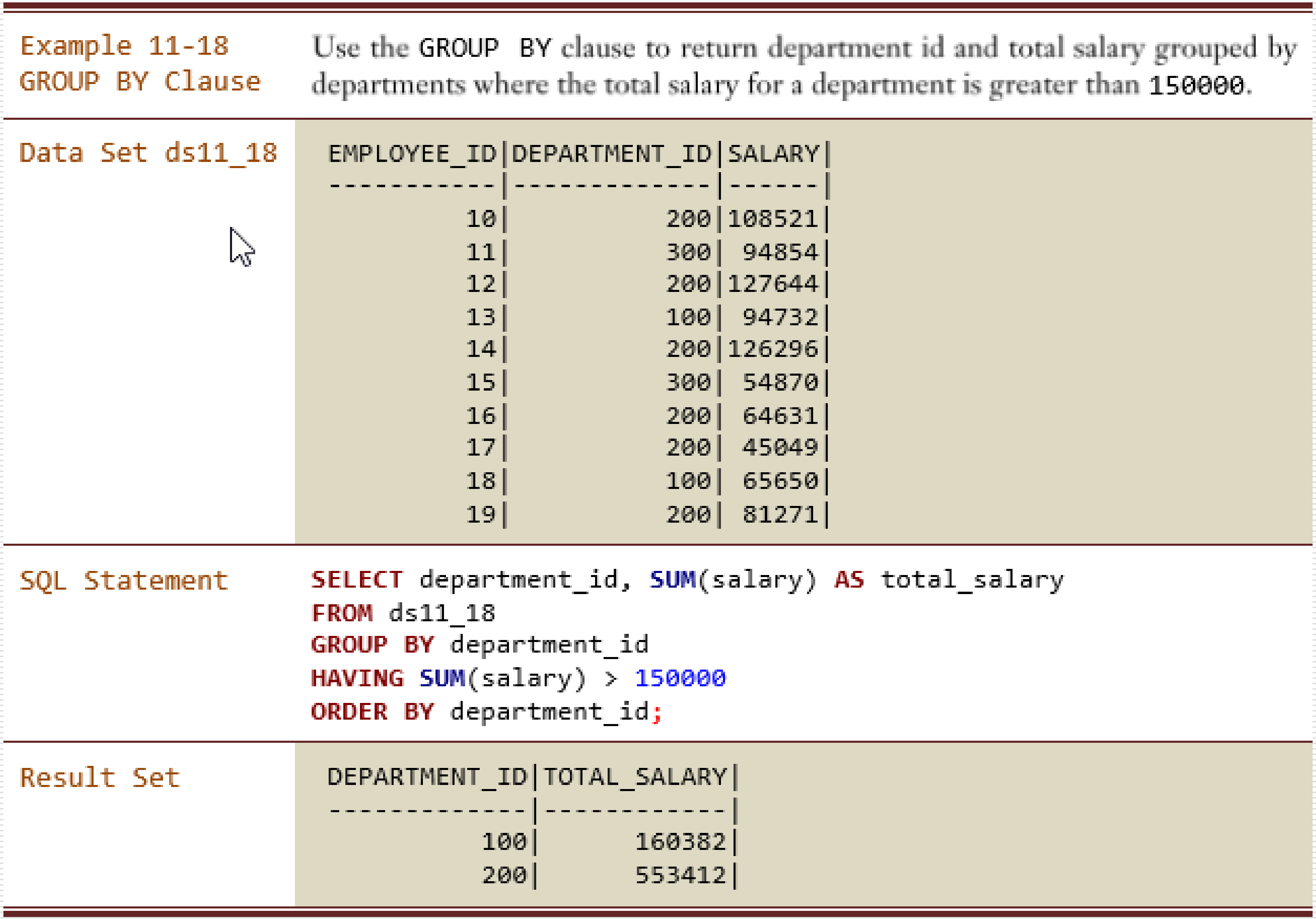
```
Month|Day|Total Orders:|Total Sales|
-----|---|-------------|-----------|
    5| 15|            5|       8101|
    5| 22|            2|       2160|
    6| 10|            3|       4250|
```

GROUP BY & HAVING Clauses

HAVING Clause

- WHERE clause – Restricts rows
- HAVING clause – Restricts groups

- In a query using a GROUP BY and HAVING clause:
  - The rows are first grouped
  - Group functions are applied
  - Only those groups matching the HAVING clause are displayed



Example 11-18 GROUP BY Clause

Use the GROUP BY clause to return department id and total salary grouped by departments where the total salary for a department is greater than 150000.

Data Set ds11_18

```
EMPLOYEE_ID|DEPARTMENT_ID|SALARY|
-----------|-------------|------|
         10|          200|108521|
         11|          300| 94854|
         12|          200|127644|
         13|          100| 94732|
         14|          200|126296|
         15|          300| 54870|
         16|          200| 64631|
         17|          200| 45049|
         18|          100| 65650|
         19|          200| 81271|
```

SQL Statement

```
SELECT department_id, SUM(salary) AS total_salary
FROM ds11_18
GROUP BY department_id
HAVING SUM(salary) > 150000
ORDER BY department_id;
```

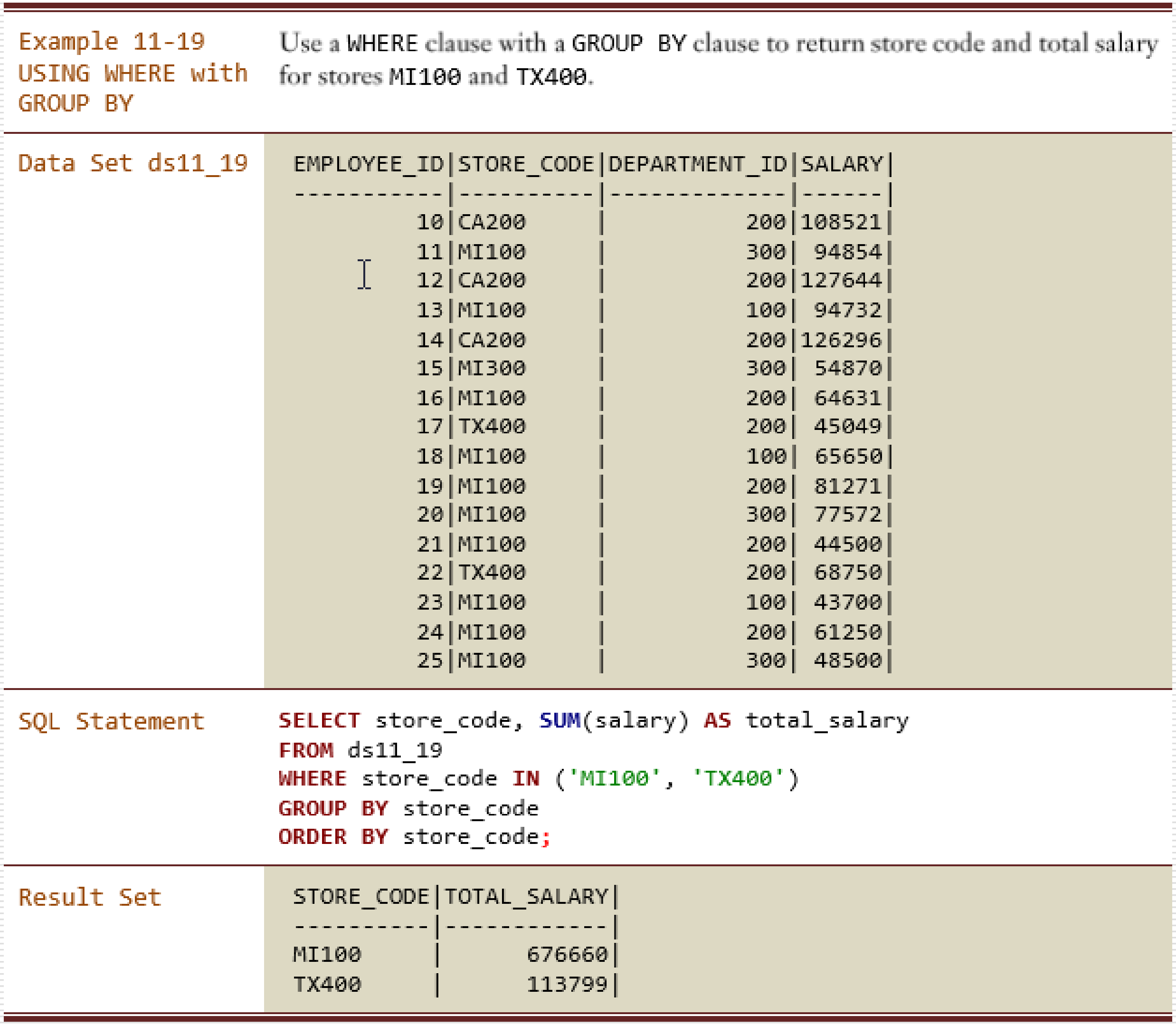
Result Set

```
DEPARTMENT_ID|TOTAL_SALARY|
-------------|------------|
          100|      160382|
          200|      553412|
```

WHERE CLAUSE versus HAVING Clause

- WHERE clause excludes rows before the rows are formed into groups

- HAVING clause excludes groups after the rows are formed into groups



Example 11-19 USING WHERE with GROUP BY

Use a WHERE clause with a GROUP BY clause to return store code and total salary for stores MI100 and TX400.

Data Set ds11_19

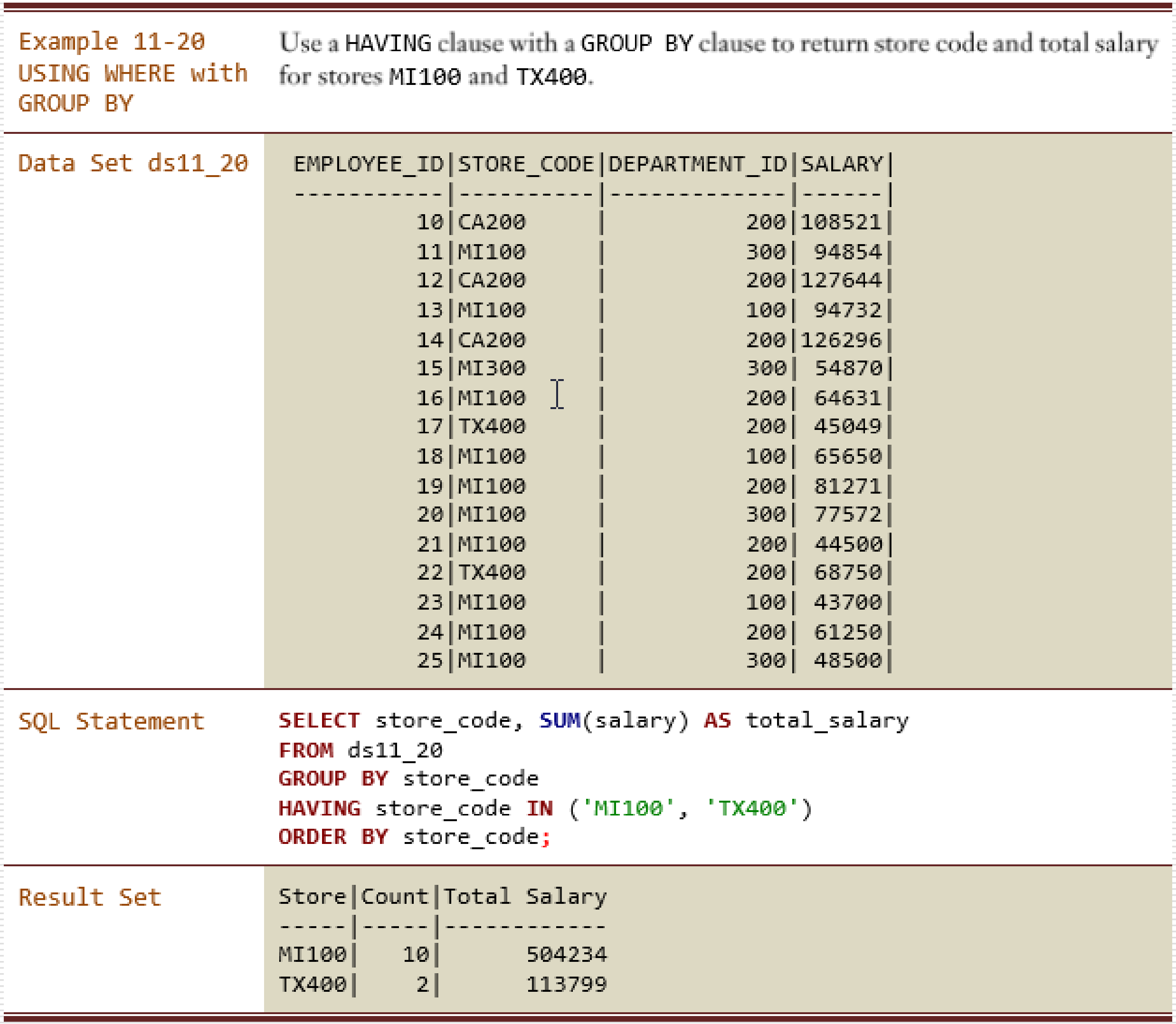
```
EMPLOYEE_ID|STORE_CODE|DEPARTMENT_ID|SALARY|
-----------|----------|-------------|------|
         10|CA200     |          200|108521|
         11|MI100     |          300| 94854|
         12|CA200     |          200|127644|
         13|MI100     |          100| 94732|
         14|CA200     |          200|126296|
         15|MI300     |          300| 54870|
         16|MI100     |          200| 64631|
         17|TX400     |          200| 45049|
         18|MI100     |          100| 65650|
         19|MI100     |          200| 81271|
         20|MI100     |          300| 77572|
         21|MI100     |          200| 44500|
         22|TX400     |          200| 68750|
         23|MI100     |          100| 43700|
         24|MI100     |          200| 61250|
         25|MI100     |          300| 48500|
```

SQL Statement

```sql
SELECT store_code, SUM(salary) AS total_salary
FROM ds11_19
WHERE store_code IN ('MI100', 'TX400')
GROUP BY store_code
ORDER BY store_code;
```

Result Set

```
STORE_CODE|TOTAL_SALARY|
----------|------------|
MI100     |      676660|
TX400     |      113799|
```



Example 11-20 USING WHERE with GROUP BY

Use a HAVING clause with a GROUP BY clause to return store code and total salary for stores MI100 and TX400.

Data Set ds11_20

```
EMPLOYEE_ID|STORE_CODE|DEPARTMENT_ID|SALARY|
-----------|----------|-------------|------|
         10|CA200     |          200|108521|
         11|MI100     |          300| 94854|
         12|CA200     |          200|127644|
         13|MI100     |          100| 94732|
         14|CA200     |          200|126296|
         15|MI300     |          300| 54870|
         16|MI100     |          200| 64631|
         17|TX400     |          200| 45049|
         18|MI100     |          100| 65650|
         19|MI100     |          200| 81271|
         20|MI100     |          300| 77572|
         21|MI100     |          200| 44500|
         22|TX400     |          200| 68750|
         23|MI100     |          100| 43700|
         24|MI100     |          200| 61250|
         25|MI100     |          300| 48500|
```

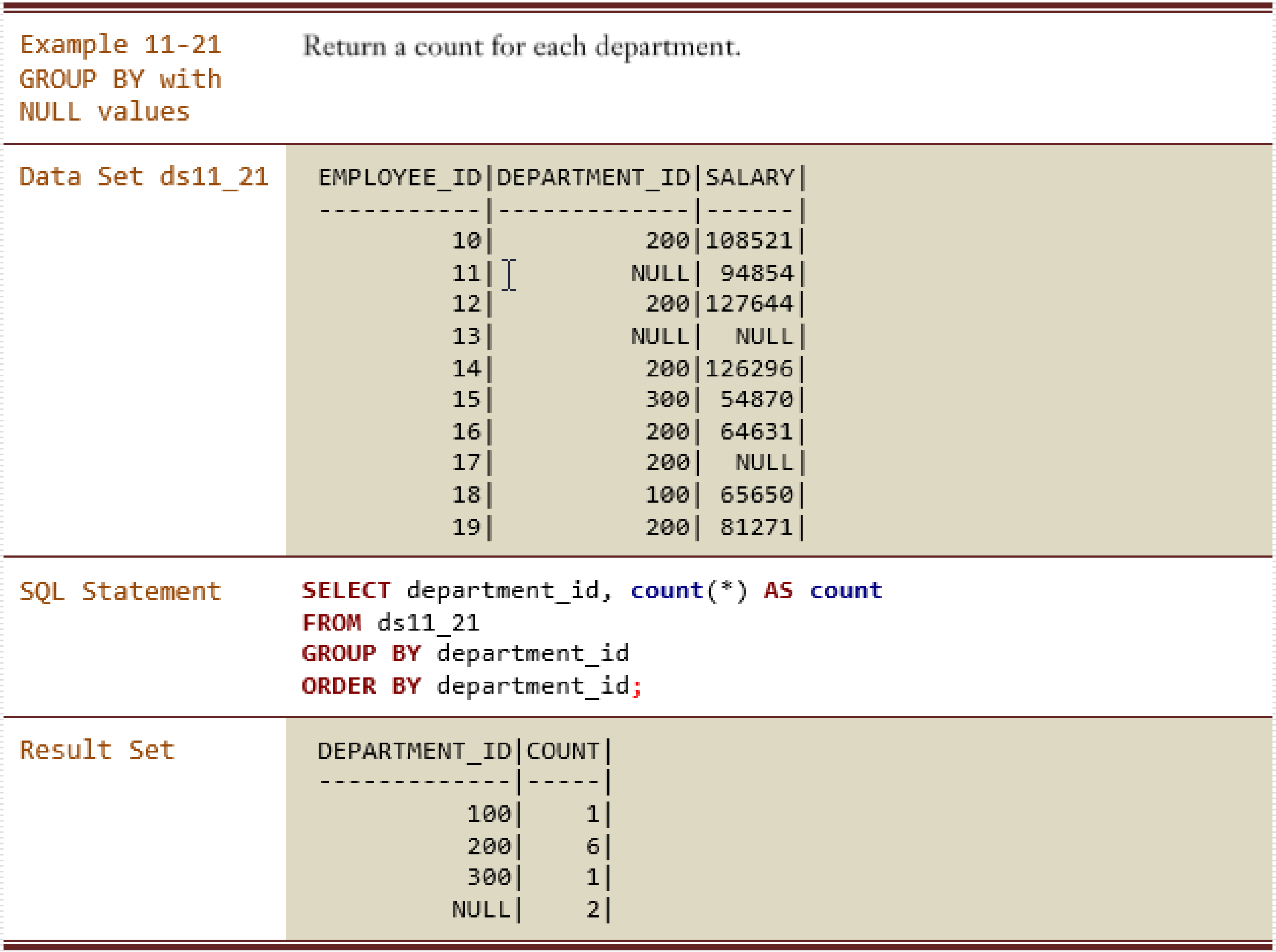
SQL Statement

```
SELECT store_code, SUM(salary) AS total_salary
FROM ds11_20
GROUP BY store_code
HAVING store_code IN ('MI100', 'TX400')
ORDER BY store_code;
```

Result Set

```
Store|Count|Total Salary
-----|-----|------------
MI100|   10|      504234
TX400|    2|      113799
```

GROUP BY with NULL Values



Example 11-21 GROUP BY with NULL values

Return a count for each department.

Data Set ds11_21

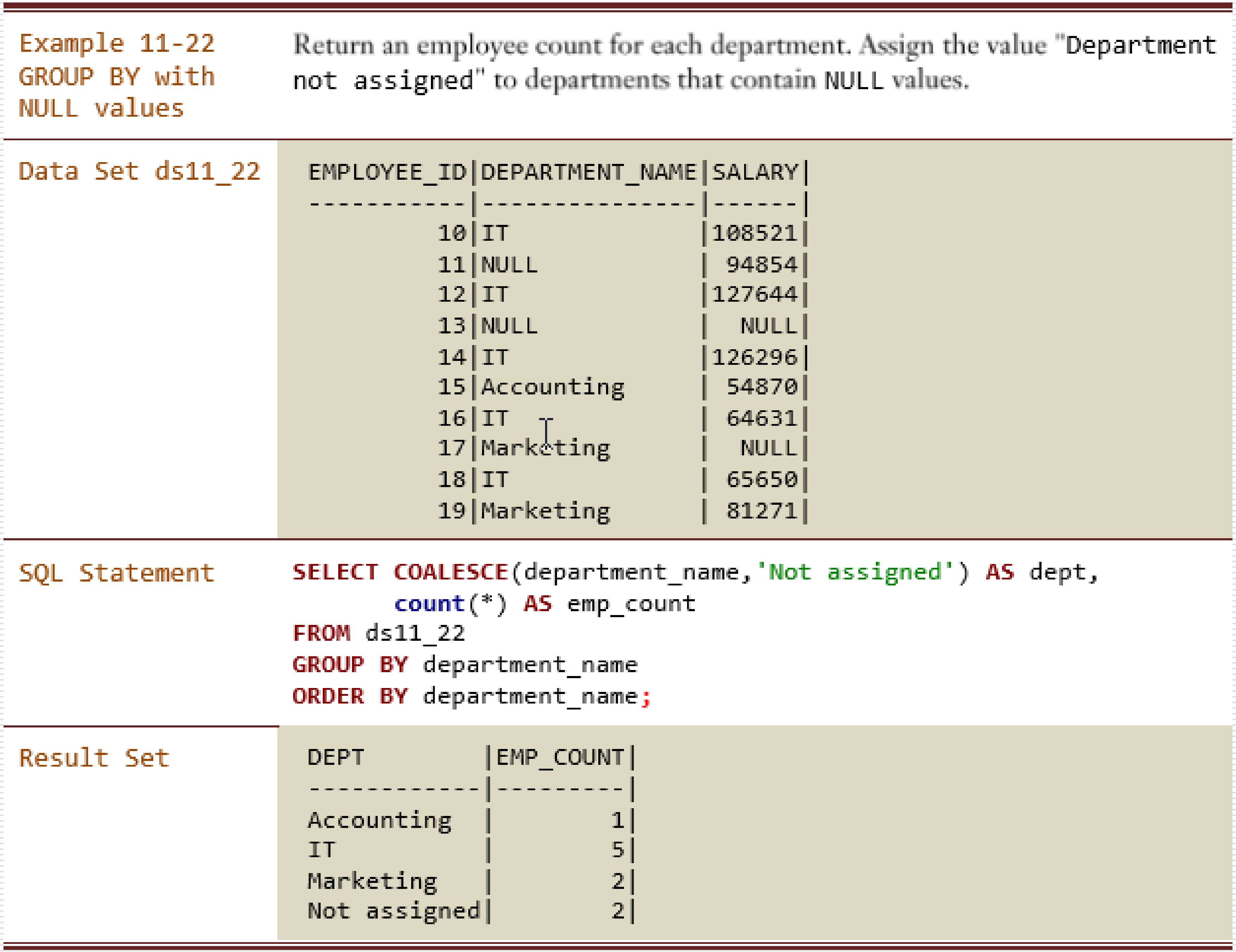
```
EMPLOYEE_ID|DEPARTMENT_ID|SALARY|
-----------|-------------|------|
         10|          200|108521|
         11|         NULL| 94854|
         12|          200|127644|
         13|         NULL|  NULL|
         14|          200|126296|
         15|          300| 54870|
         16|          200| 64631|
         17|          200|  NULL|
         18|          100| 65650|
         19|          200| 81271|
```

SQL Statement

```
SELECT department_id, count(*) AS count
FROM ds11_21
GROUP BY department_id
ORDER BY department_id;
```

Result Set

```
DEPARTMENT_ID|COUNT|
-------------|-----|
          100|    1|
          200|    6|
          300|    1|
         NULL|    2|
```



Example 11-22 GROUP BY with NULL values

Return an employee count for each department. Assign the value "Department not assigned" to departments that contain NULL values.

Data Set ds11_22

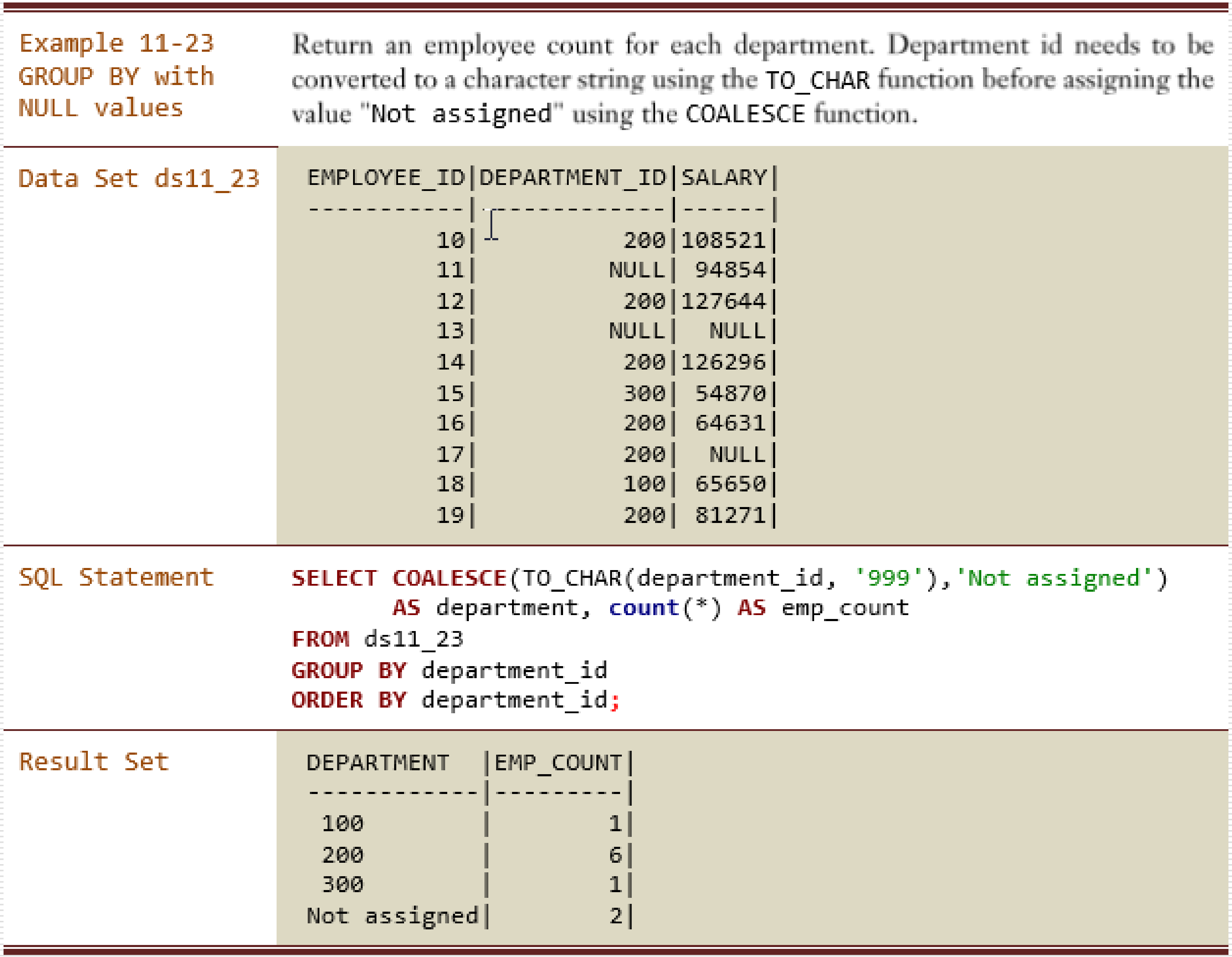
```
EMPLOYEE_ID|DEPARTMENT_NAME|SALARY|
-----------|---------------|------|
         10|IT             |108521|
         11|NULL           | 94854|
         12|IT             |127644|
         13|NULL           |  NULL|
         14|IT             |126296|
         15|Accounting     | 54870|
         16|IT             | 64631|
         17|Marketing      |  NULL|
         18|IT             | 65650|
         19|Marketing      | 81271|
```

SQL Statement

```
SELECT COALESCE(department_name,'Not assigned') AS dept,
       count(*) AS emp_count
FROM ds11_22
GROUP BY department_name
ORDER BY department_name;
```

Result Set

```
DEPT        |EMP_COUNT|
------------|---------|
Accounting  |        1|
IT          |        5|
Marketing   |        2|
Not assigned|        2|
```



Example 11-23 GROUP BY with NULL values

Return an employee count for each department. Department id needs to be converted to a character string using the TO_CHAR function before assigning the value "Not assigned" using the COALESCE function.

Data Set ds11_23

```
EMPLOYEE_ID|DEPARTMENT_ID|SALARY|
-----------|-------------|------|
         10|          200|108521|
         11|         NULL| 94854|
         12|          200|127644|
         13|         NULL|  NULL|
         14|          200|126296|
         15|          300| 54870|
         16|          200| 64631|
         17|          200|  NULL|
         18|          100| 65650|
         19|          200| 81271|
```

SQL Statement

```
SELECT COALESCE(TO_CHAR(department_id, '999'),'Not assigned')
       AS department, count(*) AS emp_count
FROM ds11_23
GROUP BY department_id
ORDER BY department_id;
```

Result Set

```
DEPARTMENT  |EMP_COUNT|
------------|---------|
 100        |        1|
 200        |        6|
 300        |        1|
Not assigned|        2|
```

GROUP BY Rule

- GROUP BY requires that any column listed in the SELECT column-list that is not part of a group function (SUM, AVG, MIN, MAX, COUNT) must be listed in a GROUP BY clause
- What is wrong with this example?

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

GROUP BY Rule – No alias

- You cannot use a column alias in the GROUP BY clause
- What is wrong with this example?

```
SELECT job_id, AVG(salary) AS AVG_SAL
  FROM employees
  GROUP BY AVG_SAL;
```

Logical Processing Order of the SELECT Statement

Order	Clause	Function
1	FROM	Joins to tables
2	WHERE	Filters the non-joined rows
3	GROUP BY	Groups the data
4	HAVING	Filters the group rows
5	SELECT	Filters the columns and aggregations in the data set
6	ORDER BY	Sorts the final data
7	LIMIT	Limits the number of rows in the result set

Logical Processing Order of the SELECT Statement

- The order determines when the objects defined in one step are made available to the clauses in subsequent steps
  - When the query processor binds to the tables or views defined in the FROM clause, these objects and their columns are made available to all subsequent steps

Logical Processing Order of the SELECT Statement

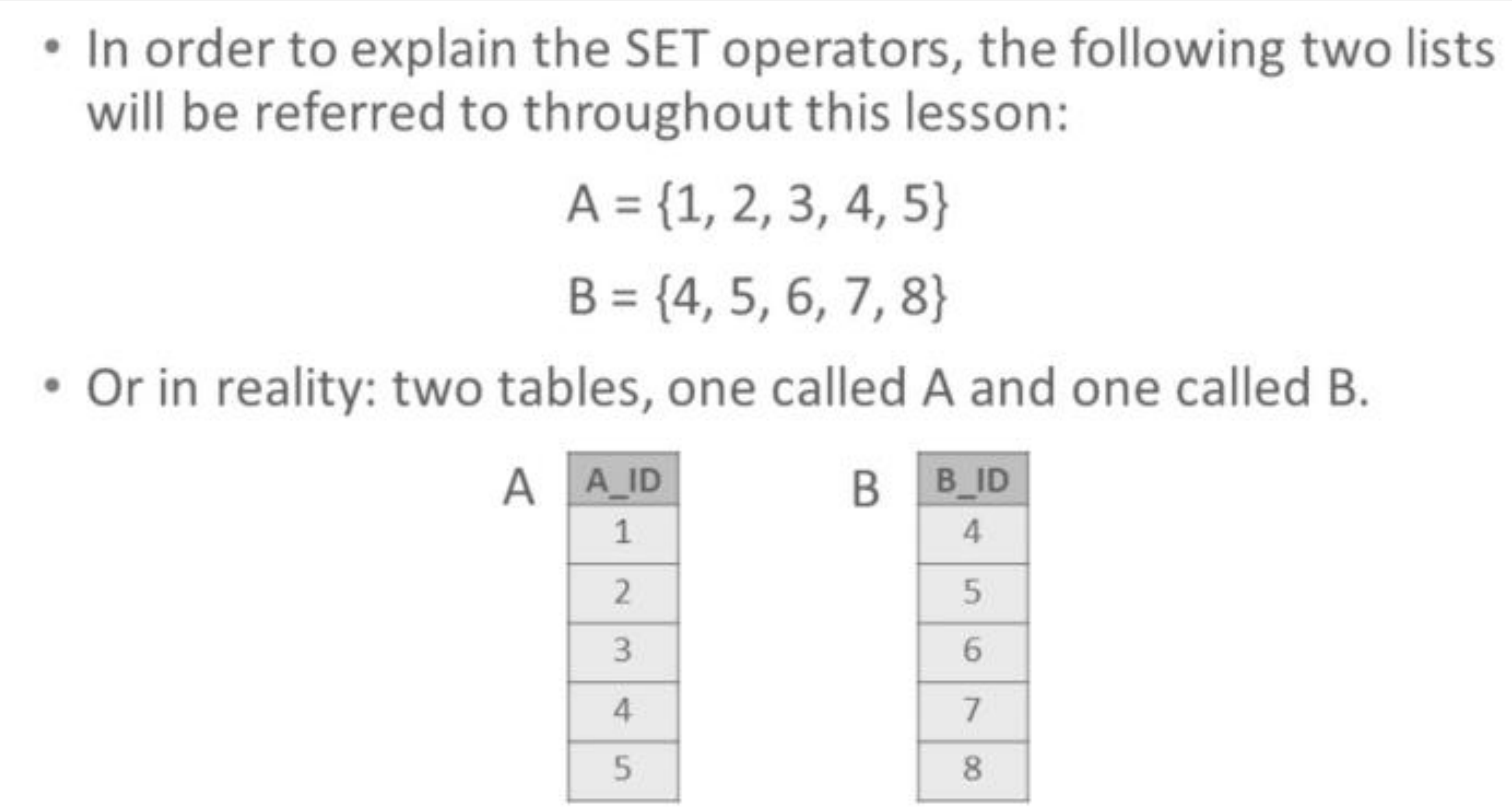
- The SELECT clause is step 5
  - Any column aliases or derived columns defined in that clause cannot be referenced by preceding clauses
  - However, they can be referenced by subsequent clauses such as the ORDER BY clause

Set Operators

- Set Operators
  - Combine the results of two or more SELECT statements

- UNION
- UNION ALL
- INTERSECT
- MINUS

Tables for Set Operators



- In order to explain the SET operators, the following two lists will be referred to throughout this lesson:

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

- Or in reality: two tables, one called A and one called B.

A

A_ID
1
2
3
4
5

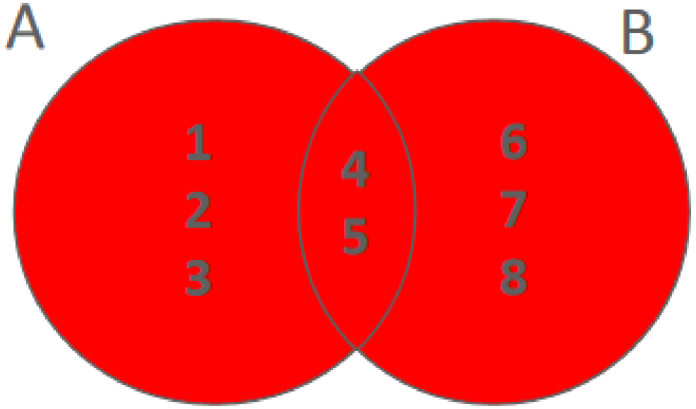
B

B_ID
4
5
6
7
8

UNION Operator

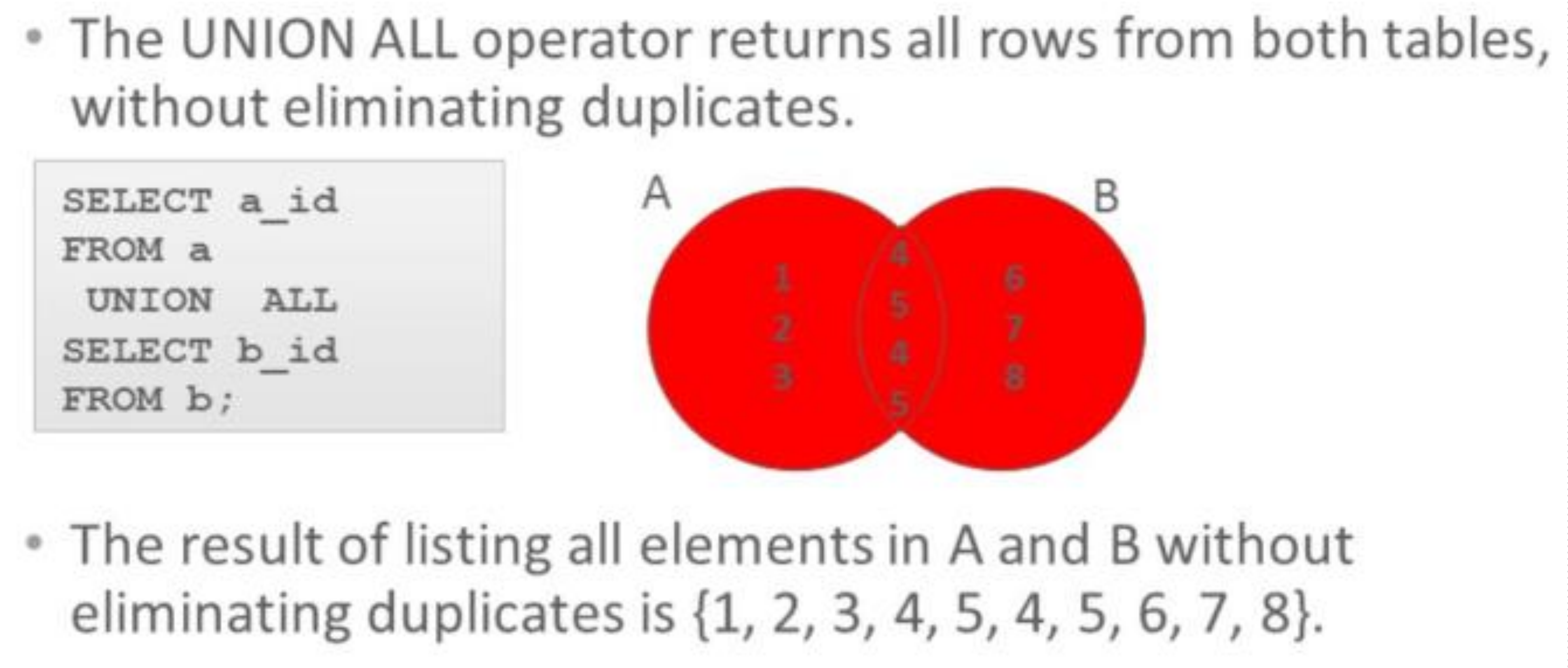
- The UNION operator returns all rows from both tables, after eliminating duplicates.

```
SELECT a_id
FROM a
 UNION
SELECT b_id
FROM b;
```



- The result of listing all elements in A and B eliminating duplicates is {1, 2, 3, 4, 5, 6, 7, 8}.

UNION ALL Operator

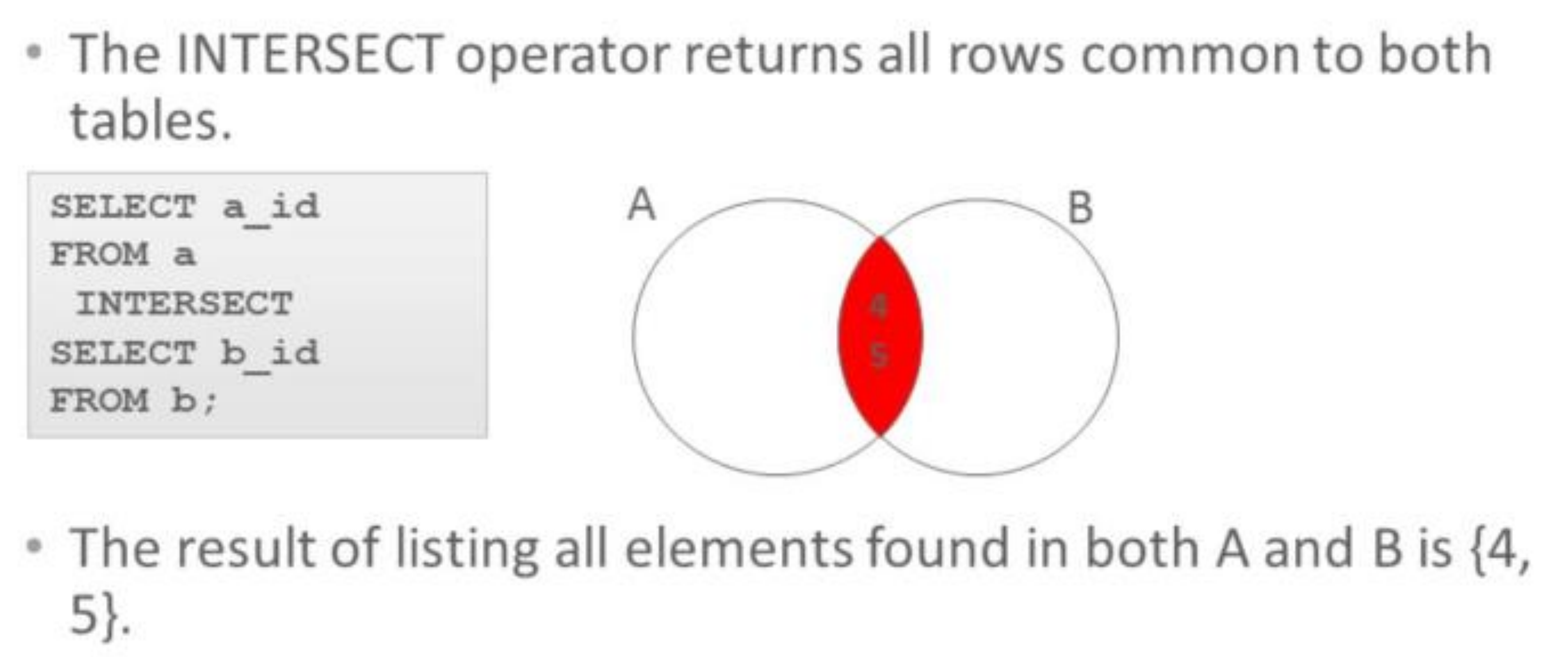


- The UNION ALL operator returns all rows from both tables, without eliminating duplicates.

```
SELECT a_id
FROM a
 UNION  ALL
SELECT b_id
FROM b;
```

- The result of listing all elements in A and B without eliminating duplicates is {1, 2, 3, 4, 5, 4, 5, 6, 7, 8}.

INTERSECT Operator

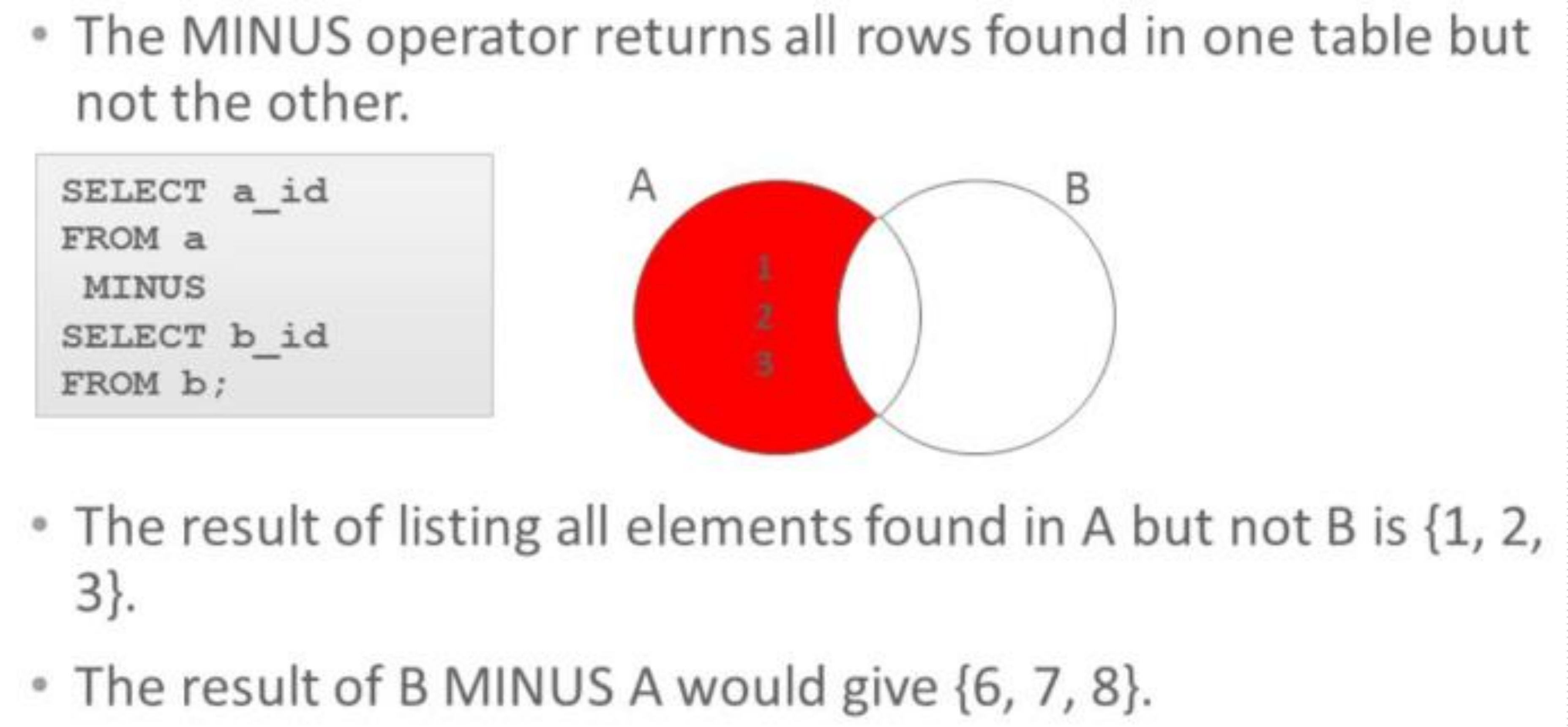


- The INTERSECT operator returns all rows common to both tables.

```
SELECT a_id
FROM a
 INTERSECT
SELECT b_id
FROM b;
```

- The result of listing all elements found in both A and B is {4, 5}.

MINUS Operator



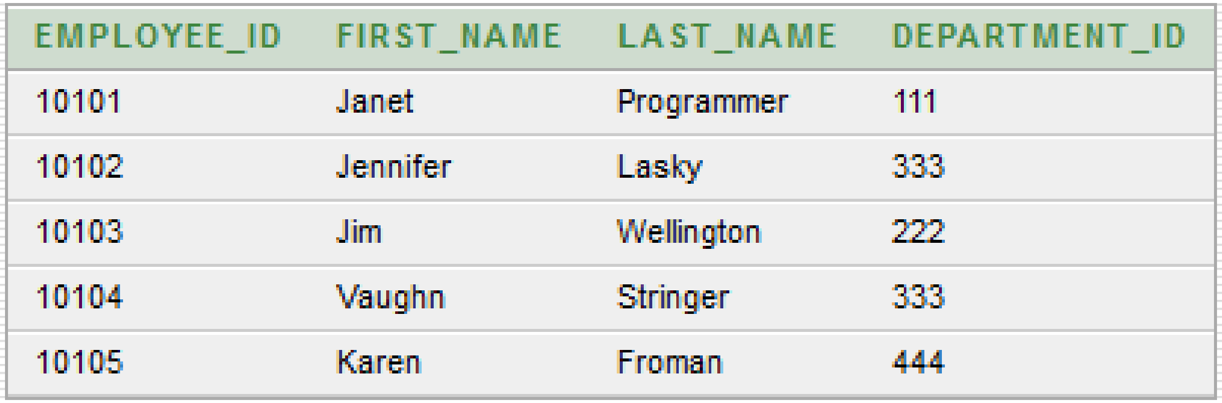
- The MINUS operator returns all rows found in one table but not the other.

```
SELECT a_id
FROM a
 MINUS
SELECT b_id
FROM b;
```

- The result of listing all elements found in A but not B is {1, 2, 3}.
- The result of B MINUS A would give {6, 7, 8}.

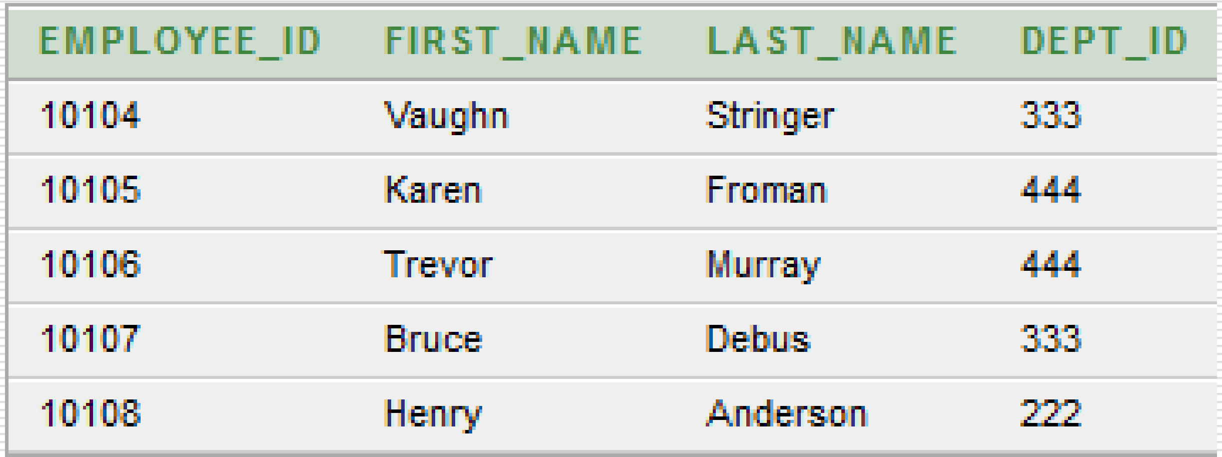
Tables for Set Operators

```
SELECT * FROM s_employees;
```



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
10101	Janet	Programmer	111
10102	Jennifer	Lasky	333
10103	Jim	Wellington	222
10104	Vaughn	Stringer	333
10105	Karen	Froman	444

```
SELECT * FROM s_employees_retired;
```



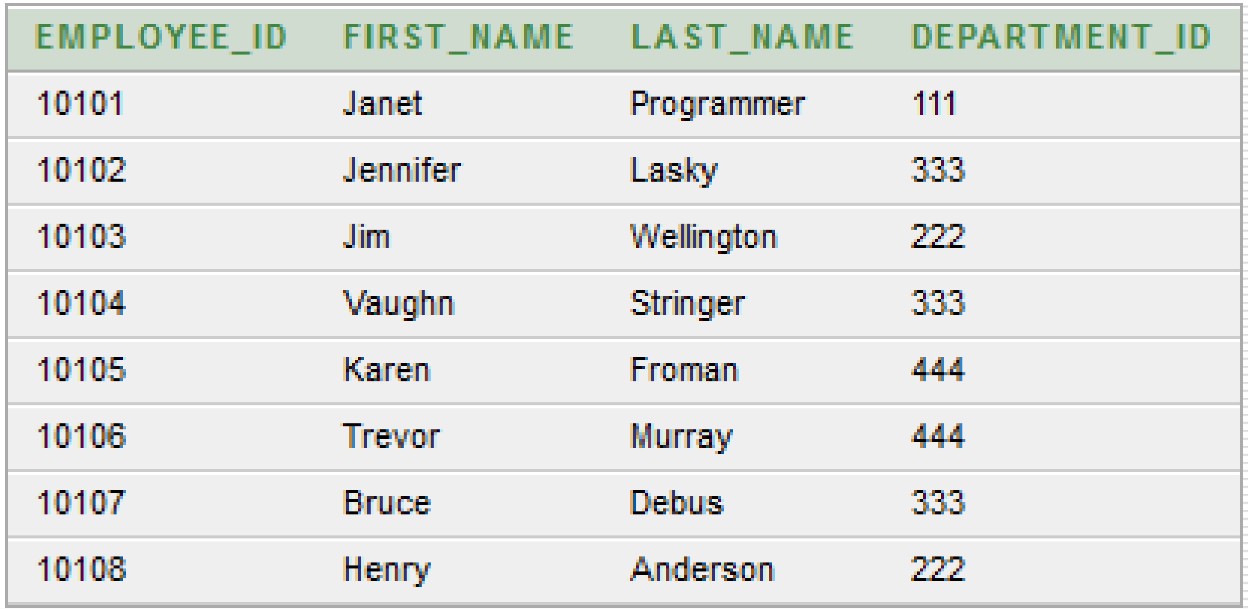
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPT_ID
10104	Vaughn	Stringer	333
10105	Karen	Froman	444
10106	Trevor	Murray	444
10107	Bruce	Debus	333
10108	Henry	Anderson	222

UNION Operator

- The UNION operator returns all rows from both tables, after eliminating duplicates

UNION Operator

```
SELECT employee_id, first_name, last_name, department_id
  FROM s_employees
UNION
SELECT employee_id, first_name, last_name, dept_id
  FROM s_employees_retired;
```



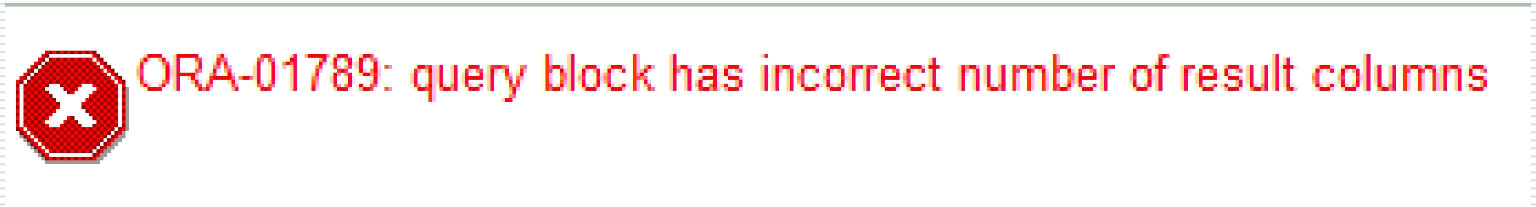
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
10101	Janet	Programmer	111
10102	Jennifer	Lasky	333
10103	Jim	Wellington	222
10104	Vaughn	Stringer	333
10105	Karen	Froman	444
10106	Trevor	Murray	444
10107	Bruce	Debus	333
10108	Henry	Anderson	222

Set Operator Rules

- The number of columns and the data types of the columns must be identical in all of the SELECT statements used in the query
- The names of the columns need not be identical
- Column names in the output are taken from the column names in the first SELECT statement
  - So any column aliases should be entered in the first statement as you would want to see them in the finished report

UNION Operator

```
SELECT employee_id, first_name
  FROM s_employees
UNION
SELECT employee_id, first_name, last_name
  FROM s_employees_retired;
```



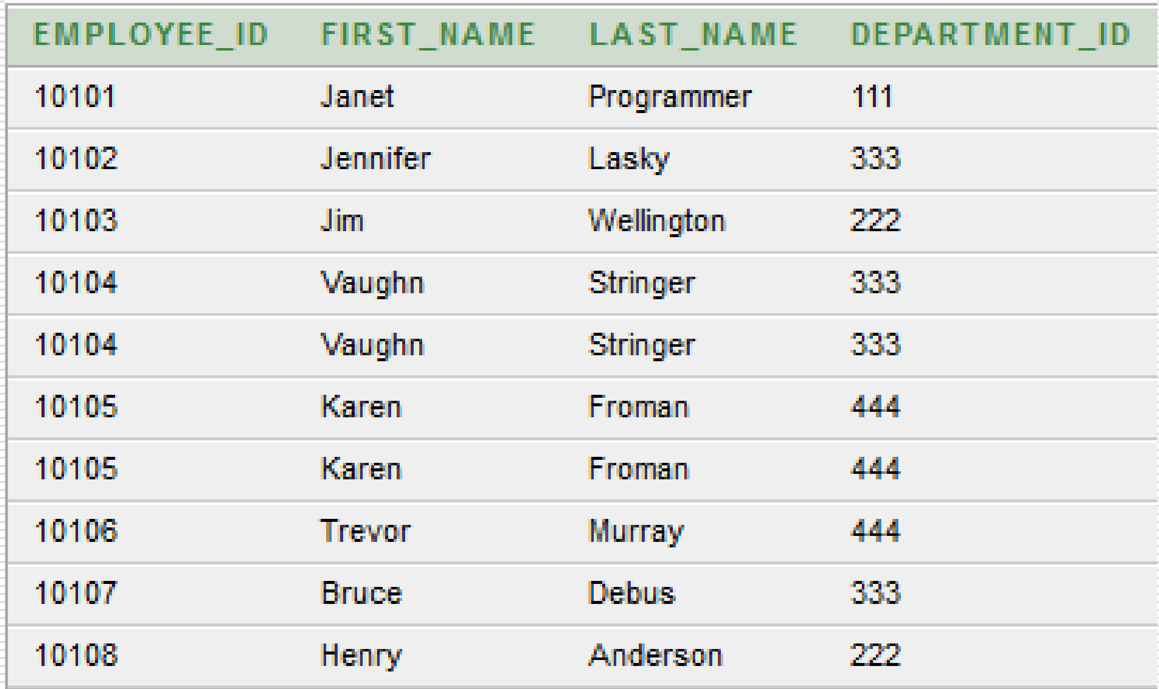
ORA-01789: query block has incorrect number of result columns

UNION ALL Operator

- The UNION ALL operator returns all rows from both tables, without eliminating duplicates

UNION ALL Operator

```
SELECT employee_id, first_name, last_name, department_id
  FROM s_employees
UNION ALL
SELECT employee_id, first_name, last_name, dept_id
  FROM s_employees_retired
ORDER BY employee_id;
```



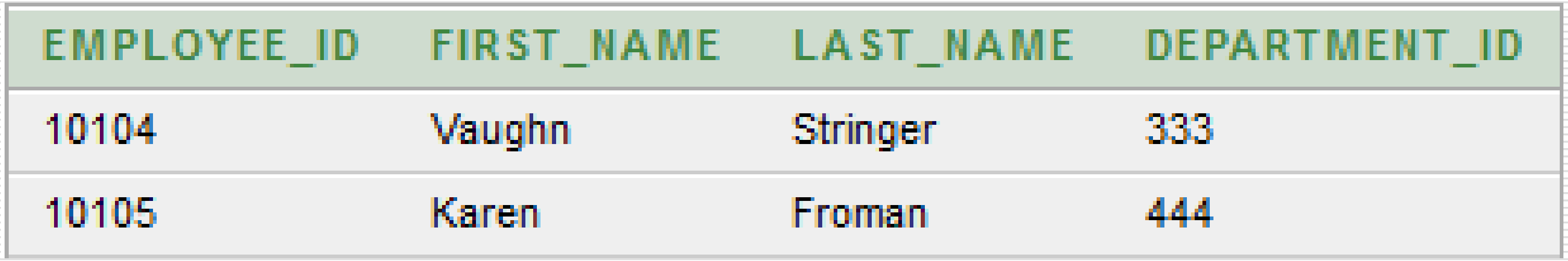
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
10101	Janet	Programmer	111
10102	Jennifer	Lasky	333
10103	Jim	Wellington	222
10104	Vaughn	Stringer	333
10104	Vaughn	Stringer	333
10105	Karen	Froman	444
10105	Karen	Froman	444
10106	Trevor	Murray	444
10107	Bruce	Debus	333
10108	Henry	Anderson	222

INTERSECT Operator

- The INTERSECT operator returns all rows common to both tables

INTERSECT Operator

```
SELECT employee_id, first_name, last_name, department_id
  FROM s_employees
INTERSECT
SELECT employee_id, first_name, last_name, dept_id
  FROM s_employees_retired
ORDER BY employee_id;
```



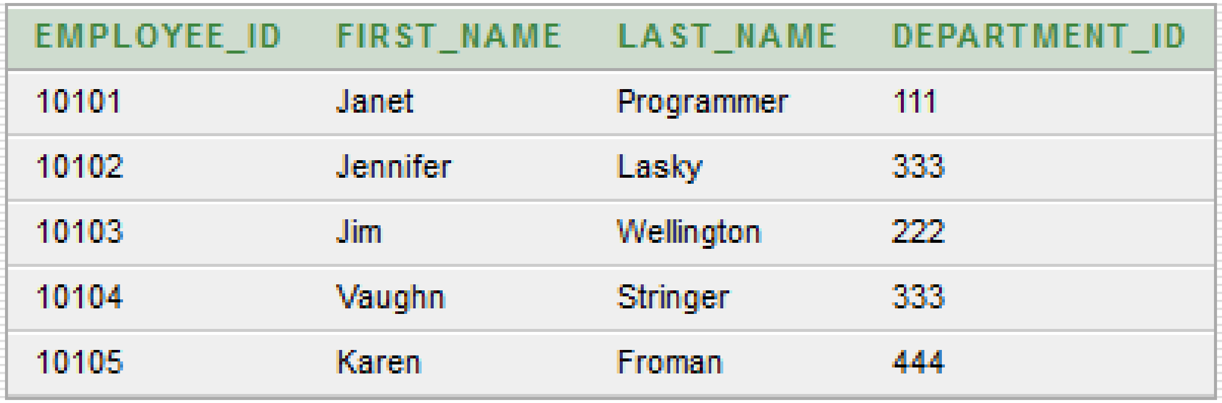
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
10104	Vaughn	Stringer	333
10105	Karen	Froman	444

MINUS Operator

- The MINUS operator returns all rows found in one table but not the other

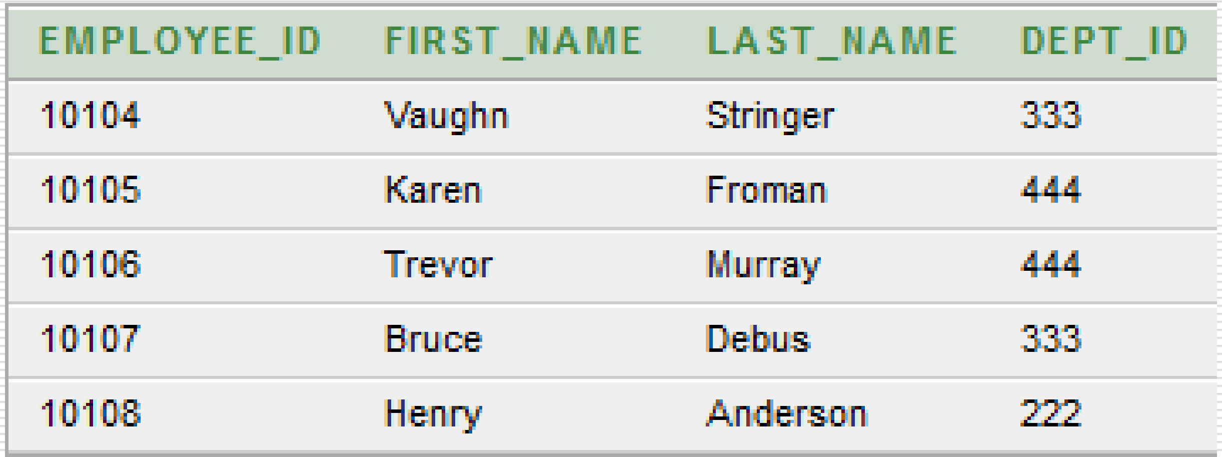
Tables for Set Operators

```
SELECT * FROM s_employees;
```



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
10101	Janet	Programmer	111
10102	Jennifer	Lasky	333
10103	Jim	Wellington	222
10104	Vaughn	Stringer	333
10105	Karen	Froman	444

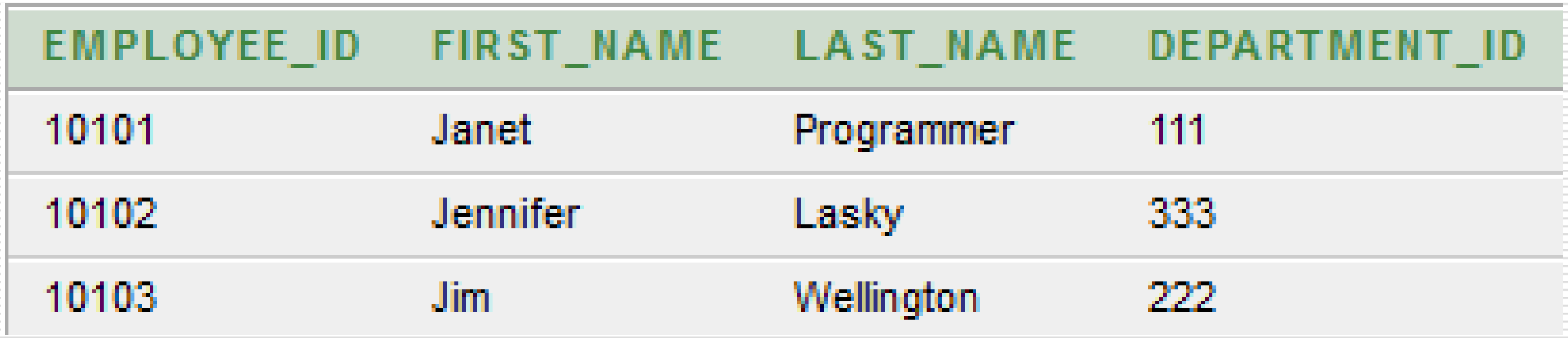
```
SELECT * FROM s_employees_retired;
```



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPT_ID
10104	Vaughn	Stringer	333
10105	Karen	Froman	444
10106	Trevor	Murray	444
10107	Bruce	Debus	333
10108	Henry	Anderson	222

MINUS Operator 1/2

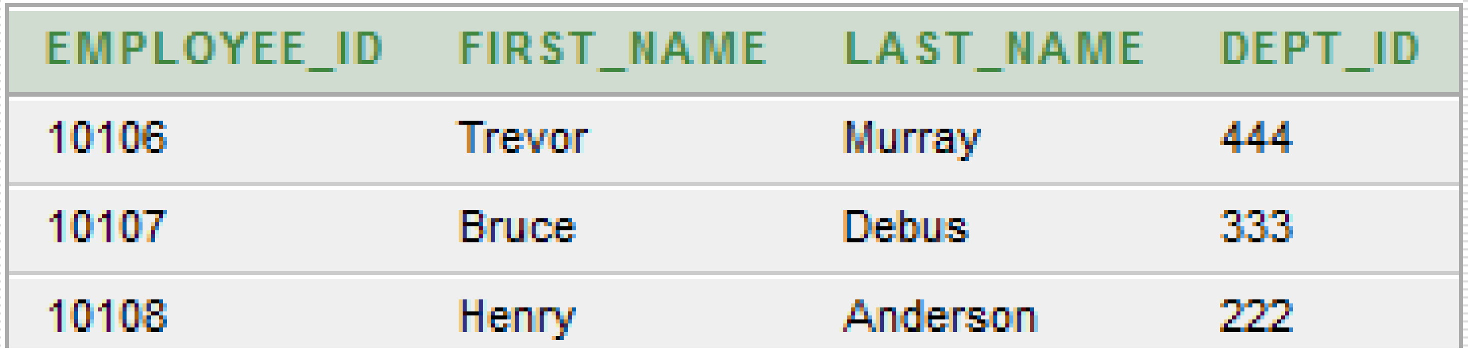
```
SELECT employee_id, first_name, last_name, department_id
  FROM s_employees
MINUS
SELECT employee_id, first_name, last_name, dept_id
  FROM s_employees_retired
ORDER BY employee_id;
```



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
10101	Janet	Programmer	111
10102	Jennifer	Lasky	333
10103	Jim	Wellington	222

MINUS Operator 2/2

```
SELECT employee_id, first_name, last_name, dept_id
  FROM s_employees_retired
MINUS
SELECT employee_id, first_name, last_name, department_id
  FROM s_employees
ORDER BY employee_id;
```



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPT_ID
10106	Trevor	Murray	444
10107	Bruce	Debus	333
10108	Henry	Anderson	222

Q
&
A
Questions &
Answers