

DATABASES
SQL
Data Manipulation Language
Part I

TECNOLOGIAS E SISTEMAS DE INFORMAÇÃO PARA A
WEB

Agenda

❖ SQL – Structured Query Language – Data Manipulation Language

❖ SELECT

❖ WHERE clause

❖ WHERE - Relational, logical, other operators

❖ Aggregation functions

❖ Grouping data

❖ Ordering data



SQL

SQL

▶ SELECT

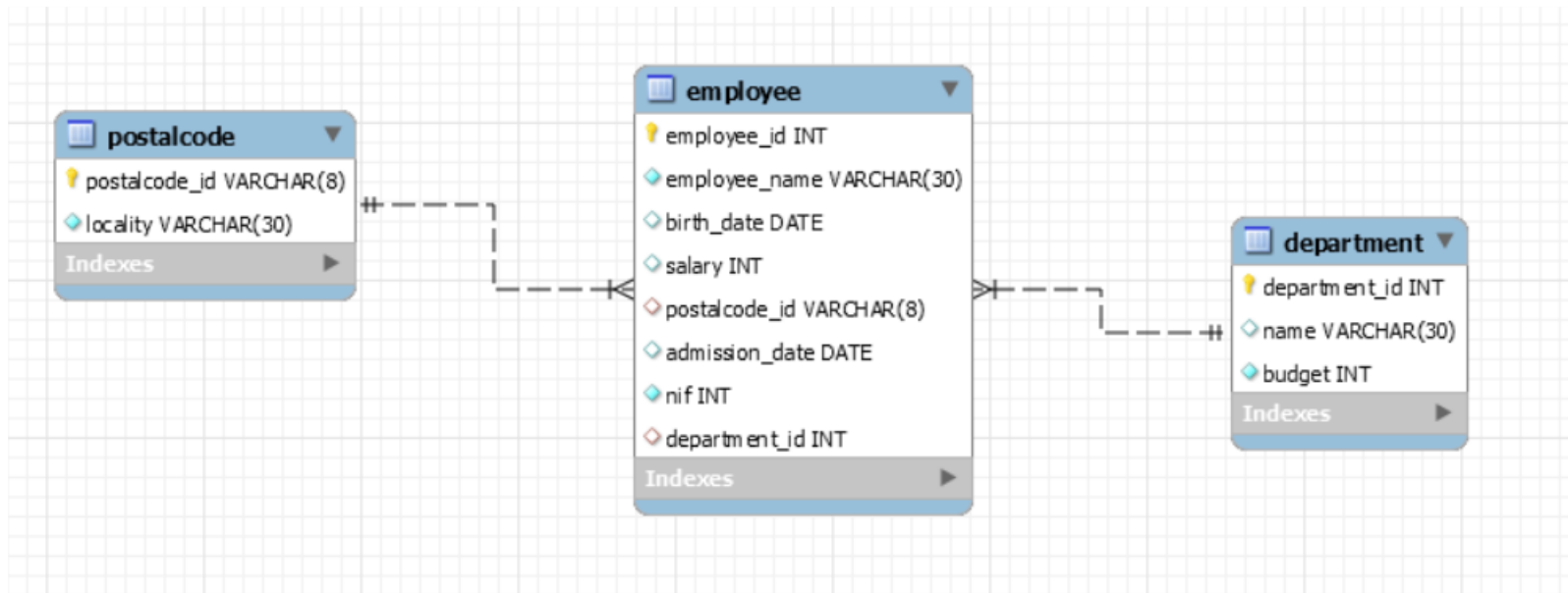
Select, query data from a database

```
SELECT [distinct/all] field, field2, ..., fieldn [*]  
FROM tables  
[WHERE conditions]  
[GROUP BY fields]  
[HAVING conditions]  
[ORDER BY fields]
```

- ▶ SELECT : specify the fields that we want to get
- ▶ FROM : specifies the source of the data
- ▶ WHERE: specifies data query conditions
- ▶ GROUP BY : groups rows of the same value in a given field
- ▶ HAVING : specifies a condition for the data groups
- ▶ ORDER BY : specifies the order (ordering) of the results obtained

SQL

- Consider the following database schema :



SQL

- ▶ With the following sample data:

	employee_id	employee_name	birth_date	salary	postalcode_id	admission_date	nif	department_id
▶	1	Manuel Santos	1972-01-01	1550	4000-100	2007-03-03	123456789	1
	2	Paulo Fosneca	1973-03-03	1820	4000-100	2008-01-01	234567890	1
	3	Carla Carolina	1974-04-04	1550	4100-050	2008-01-01	345678901	1
	4	Isabel Antunes	1975-05-05	2100	4400-100	2009-04-04	456789012	3
	5	Maria Costa	1976-06-06	1950	4480-876	2010-01-01	567890123	2
	6	Ricardo Rocha	1977-07-07	2150	4480-876	2012-07-01	678901234	4
	7	José Silva	1978-08-08	1420	4100-050	2012-07-01	789012345	5
	8	Maria Andrade	1979-09-09	1420	4000-100	2014-09-01	890123456	5
	9	Liliana Lousada	1980-10-10	2350	4460-100	2014-09-01	901234567	4
	10	Diogo Dionísio	1981-11-11	2100	4460-100	2014-09-01	213456789	3

	department_id	name	budget
	1	Production	5000
	2	Accounting	3500
	3	Computing	7000
	4	Sales	2500
▶	5	Logistics	3000
*	NULL	NULL	NULL

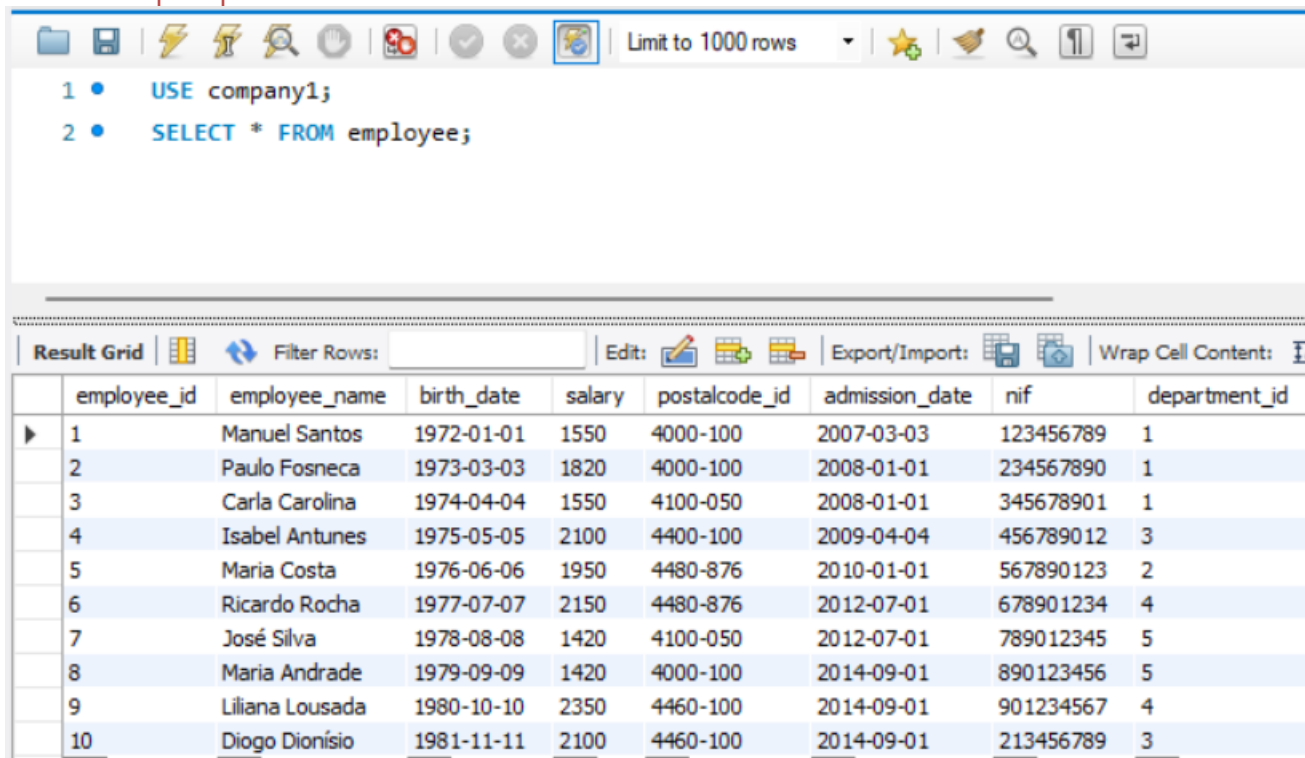
	postalcode_id	locality
▶	4000-100	Porto
	4100-050	Porto
	4400-100	V.N.Gaia
	4400-150	V.N. Gaia
	4460-100	Matosinhos
	4460-205	Matosinhos
	4480-876	Vila do Conde
*	NULL	NULL

SQL

▶ SELECTS ALL RECORDS IN THE EMPLOYEE TABLE

Execute the selected portion of the script or everything, if there is no selection

Execute the statement under the keyboard cursor

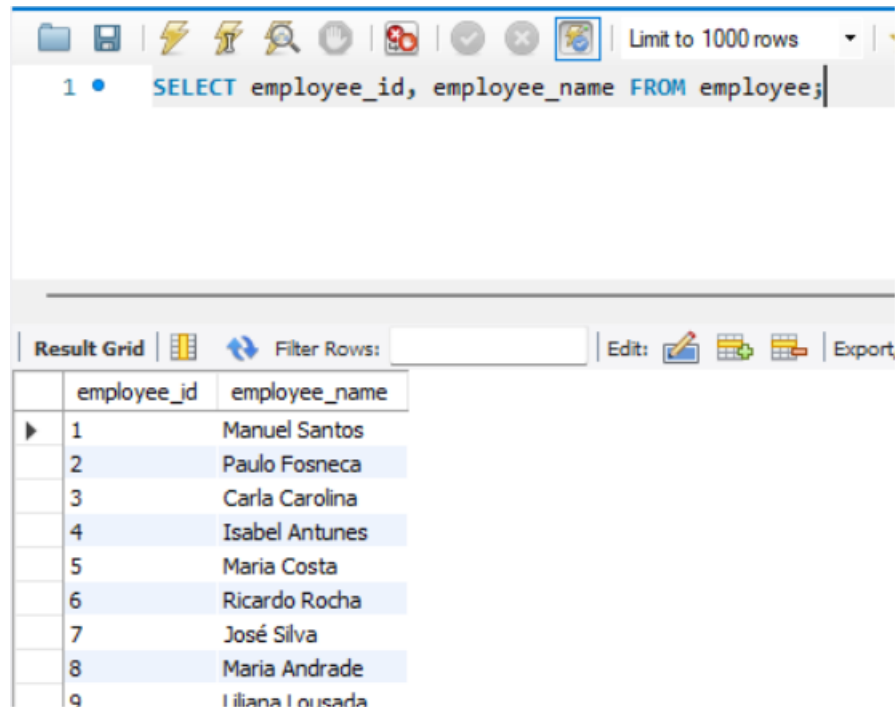


The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and search. The script editor contains two lines of SQL code: `USE company1;` and `SELECT * FROM employee;`. The result grid below displays the output of the `SELECT` statement, showing 10 rows of employee data. The grid has columns for `employee_id`, `employee_name`, `birth_date`, `salary`, `postalcode_id`, `admission_date`, `nif`, and `department_id`.

	employee_id	employee_name	birth_date	salary	postalcode_id	admission_date	nif	department_id
▶	1	Manuel Santos	1972-01-01	1550	4000-100	2007-03-03	123456789	1
	2	Paulo Fosneca	1973-03-03	1820	4000-100	2008-01-01	234567890	1
	3	Carla Carolina	1974-04-04	1550	4100-050	2008-01-01	345678901	1
	4	Isabel Antunes	1975-05-05	2100	4400-100	2009-04-04	456789012	3
	5	Maria Costa	1976-06-06	1950	4480-876	2010-01-01	567890123	2
	6	Ricardo Rocha	1977-07-07	2150	4480-876	2012-07-01	678901234	4
	7	José Silva	1978-08-08	1420	4100-050	2012-07-01	789012345	5
	8	Maria Andrade	1979-09-09	1420	4000-100	2014-09-01	890123456	5
	9	Liliana Lousada	1980-10-10	2350	4460-100	2014-09-01	901234567	4
	10	Diogo Dionísio	1981-11-11	2100	4460-100	2014-09-01	213456789	3

SQL

- ▶ SELECTS EMPLOYEE_ID AND NAME, ALL RECORDS

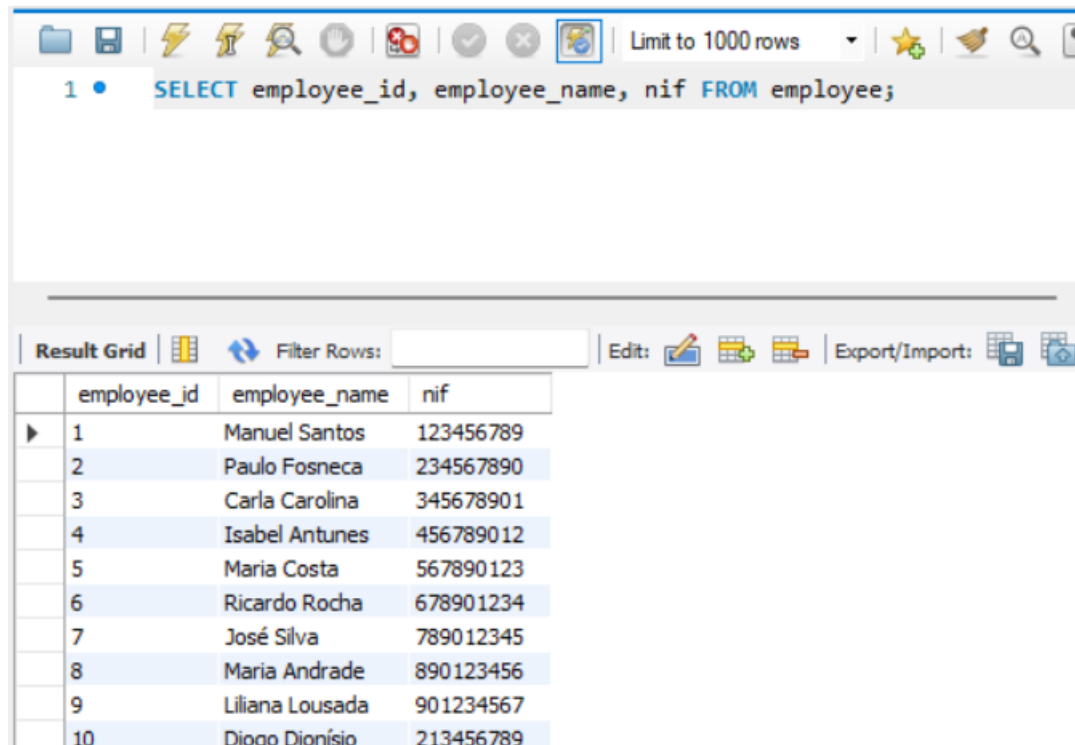


The screenshot shows a SQL query execution interface. The query entered is `SELECT employee_id, employee_name FROM employee;`. The results are displayed in a table with two columns: `employee_id` and `employee_name`. The table contains 9 rows of data. The interface includes a toolbar with various icons and a 'Limit to 1000 rows' dropdown menu.

	employee_id	employee_name
▶	1	Manuel Santos
	2	Paulo Fosneca
	3	Carla Carolina
	4	Isabel Antunes
	5	Maria Costa
	6	Ricardo Rocha
	7	José Silva
	8	Maria Andrade
	9	Liliana Louzada

SQL

- ▶ SELECTS EMPLOYEE_ID, NAME AND NIF, ALL RECORDS

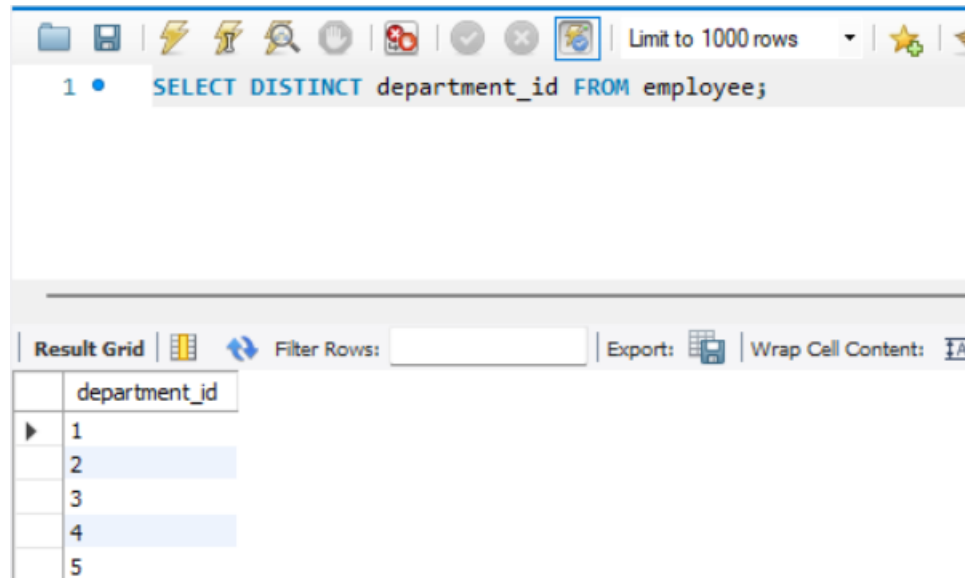


The screenshot shows a SQL query execution window. At the top, there is a toolbar with various icons and a dropdown menu set to "Limit to 1000 rows". Below the toolbar, the SQL query is displayed: `1 • SELECT employee_id, employee_name, nif FROM employee;`. The results are shown in a table with the following columns: `employee_id`, `employee_name`, and `nif`. The table contains 10 rows of data.

	employee_id	employee_name	nif
▶	1	Manuel Santos	123456789
	2	Paulo Foseca	234567890
	3	Carla Carolina	345678901
	4	Isabel Antunes	456789012
	5	Maria Costa	567890123
	6	Ricardo Rocha	678901234
	7	José Silva	789012345
	8	Maria Andrade	890123456
	9	Liliana Lousada	901234567
	10	Dioqo Dionísio	213456789

SQL

- ▶ WHEN AN ENTITY ATTRIBUTE CONTAINS (OR MAY CONTAIN) DUPLICATE VALUES, THE
- ▶ **DISTINCT** KEYWORD **RETURNS** ONLY DISTINCT, **DIFFERENT VALUES**.



The screenshot shows a SQL query editor interface. At the top, there is a toolbar with various icons for file operations, execution, and viewing. Below the toolbar, the SQL query is entered: `1 • SELECT DISTINCT department_id FROM employee;`. The query is highlighted in blue. Below the query, there is a section for the results. The results are displayed in a table with the column header `department_id`. The table contains five rows with values 1, 2, 3, 4, and 5. The first row is highlighted in blue. The interface also includes a 'Limit to 1000 rows' dropdown, a 'Filter Rows' input field, and an 'Export' button.

department_id
1
2
3
4
5

SQL

Restrictions: **WHERE** keyword

- ▶ Lets you impose constraints / conditions on data selection

- ▶ Operators:

```
SELECT [distinct/all] field1, field2, ..., fieldn [*]  
FROM tables  
[WHERE condition]  
[GROUP BY fields]  
[HAVING conditions]  
[ORDER BY fields]
```

Relational
=
>
<
>=
<=
<> Ou !=

Logical
AND ou &&
OR ou
NOT ou !
XOR (disjunção exclusiva)



condition 1 AND NOT condition 2

Others
BETWEEN
IN
IS NULL, NOT NULL
LIKE

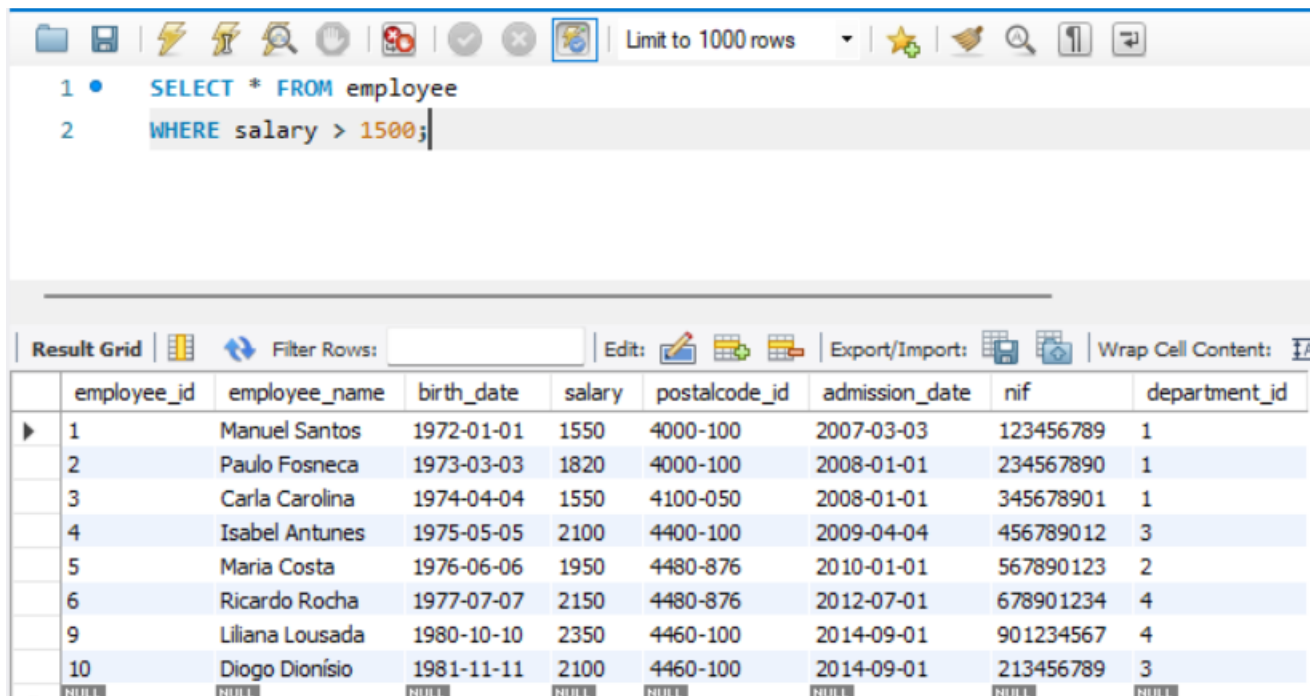


Wildcards: %, _

SQL

Restrictions: **WHERE** keyword

- ▶ SELECTS ALL RECORDS OF THE EMPLOYEE'S ENTITY WHERE THE SALARY IS > 1500



Limit to 1000 rows

```
1 • SELECT * FROM employee
2 WHERE salary > 1500;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: I

	employee_id	employee_name	birth_date	salary	postalcode_id	admission_date	nif	department_id
▶	1	Manuel Santos	1972-01-01	1550	4000-100	2007-03-03	123456789	1
	2	Paulo Fosneca	1973-03-03	1820	4000-100	2008-01-01	234567890	1
	3	Carla Carolina	1974-04-04	1550	4100-050	2008-01-01	345678901	1
	4	Isabel Antunes	1975-05-05	2100	4400-100	2009-04-04	456789012	3
	5	Maria Costa	1976-06-06	1950	4480-876	2010-01-01	567890123	2
	6	Ricardo Rocha	1977-07-07	2150	4480-876	2012-07-01	678901234	4
	9	Liliana Lousada	1980-10-10	2350	4460-100	2014-09-01	901234567	4
	10	Diogo Dionísio	1981-11-11	2100	4460-100	2014-09-01	213456789	3

Restrictions: **WHERE** keyword

- [illegible]

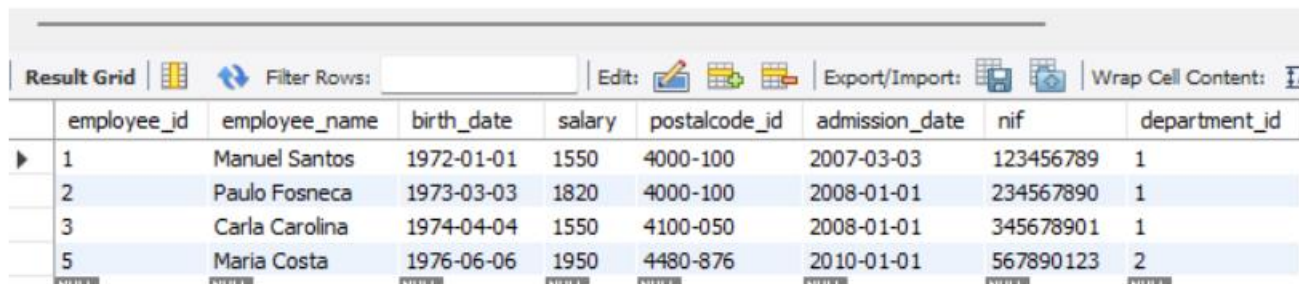
SQL

Restrictions: **WHERE** keyword

- ▶ SELECT THE EMPLOYEES WHERE THE SALARY IS BETWEEN 1500 AND 2000



```
1 • SELECT * FROM employee
2 WHERE salary >=1500 AND salary <=2000;
```

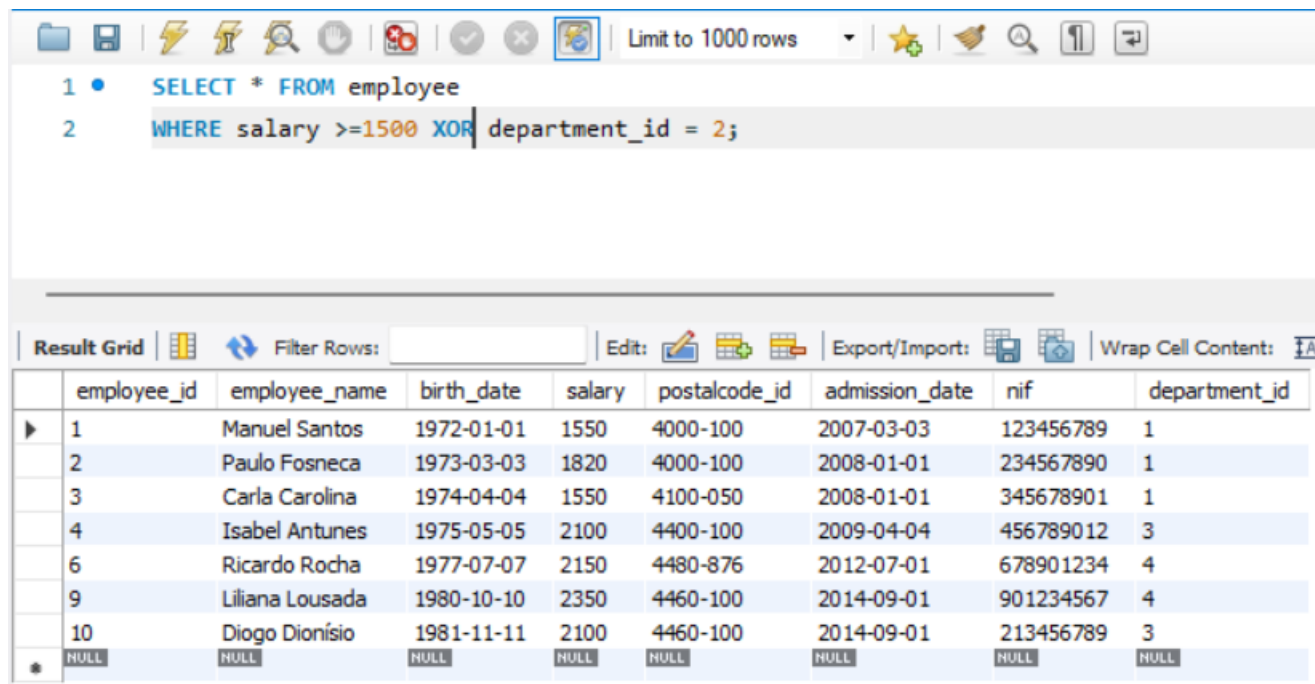


	employee_id	employee_name	birth_date	salary	postalcode_id	admission_date	nif	department_id
▶	1	Manuel Santos	1972-01-01	1550	4000-100	2007-03-03	123456789	1
	2	Paulo Fosneca	1973-03-03	1820	4000-100	2008-01-01	234567890	1
	3	Carla Carolina	1974-04-04	1550	4100-050	2008-01-01	345678901	1
	5	Maria Costa	1976-06-06	1950	4480-876	2010-01-01	567890123	2

SQL

Restrictions: **WHERE** keyword

- ▶ SELECT THE EMPLOYEES WHERE THE SALARY IS ≥ 1500 AND THE DEPARTMENT IS NOT 2



Limit to 1000 rows

```
1 • SELECT * FROM employee
2 WHERE salary >=1500 XOR department_id = 2;
```

Result Grid

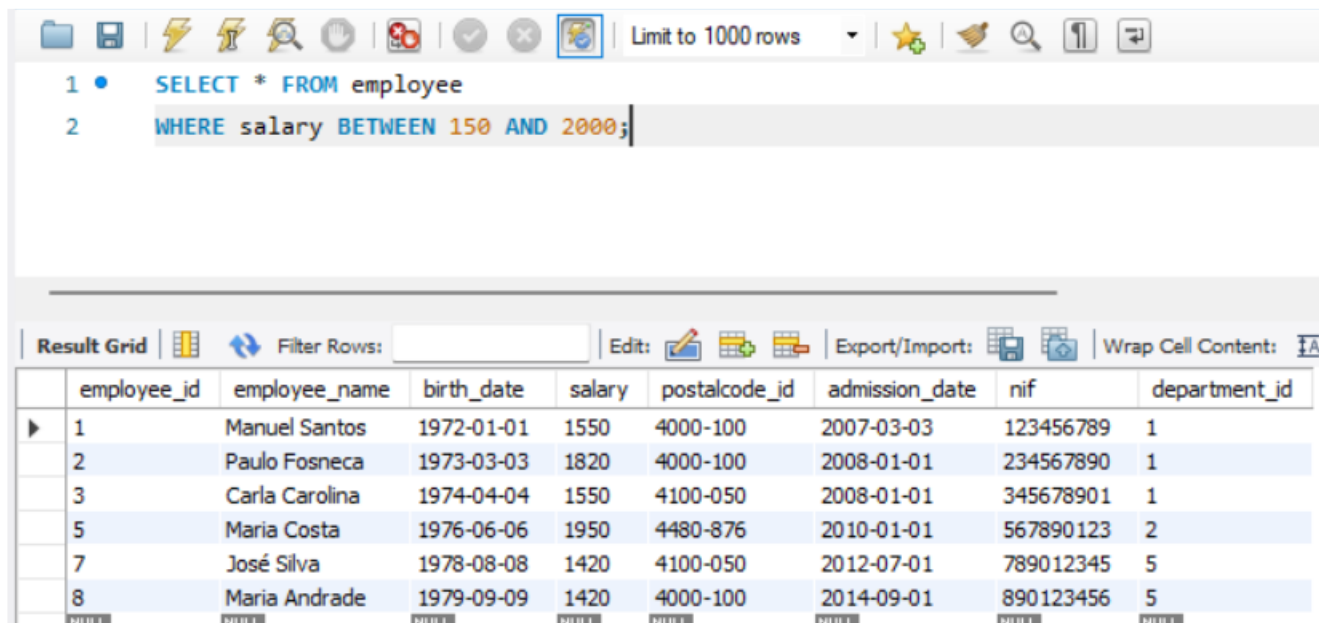
	employee_id	employee_name	birth_date	salary	postalcode_id	admission_date	nif	department_id
▶	1	Manuel Santos	1972-01-01	1550	4000-100	2007-03-03	123456789	1
	2	Paulo Fosneca	1973-03-03	1820	4000-100	2008-01-01	234567890	1
	3	Carla Carolina	1974-04-04	1550	4100-050	2008-01-01	345678901	1
	4	Isabel Antunes	1975-05-05	2100	4400-100	2009-04-04	456789012	3
	6	Ricardo Rocha	1977-07-07	2150	4480-876	2012-07-01	678901234	4
	9	Liliana Lousada	1980-10-10	2350	4460-100	2014-09-01	901234567	4
	10	Diogo Dionísio	1981-11-11	2100	4460-100	2014-09-01	213456789	3
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

XOR: denial of condition2

SQL

Restrictions: **WHERE** keyword

- ▶ SELECT THE EMPLOYEES WHERE THE SALARY IS BETWEEN 1500 AND 2000



The screenshot shows a SQL query editor interface. The query entered is:

```
1 • SELECT * FROM employee
2 WHERE salary BETWEEN 1500 AND 2000;
```

Below the query editor, the results are displayed in a table with the following columns: employee_id, employee_name, birth_date, salary, postalcode_id, admission_date, nif, and department_id. The results show 8 rows of data.

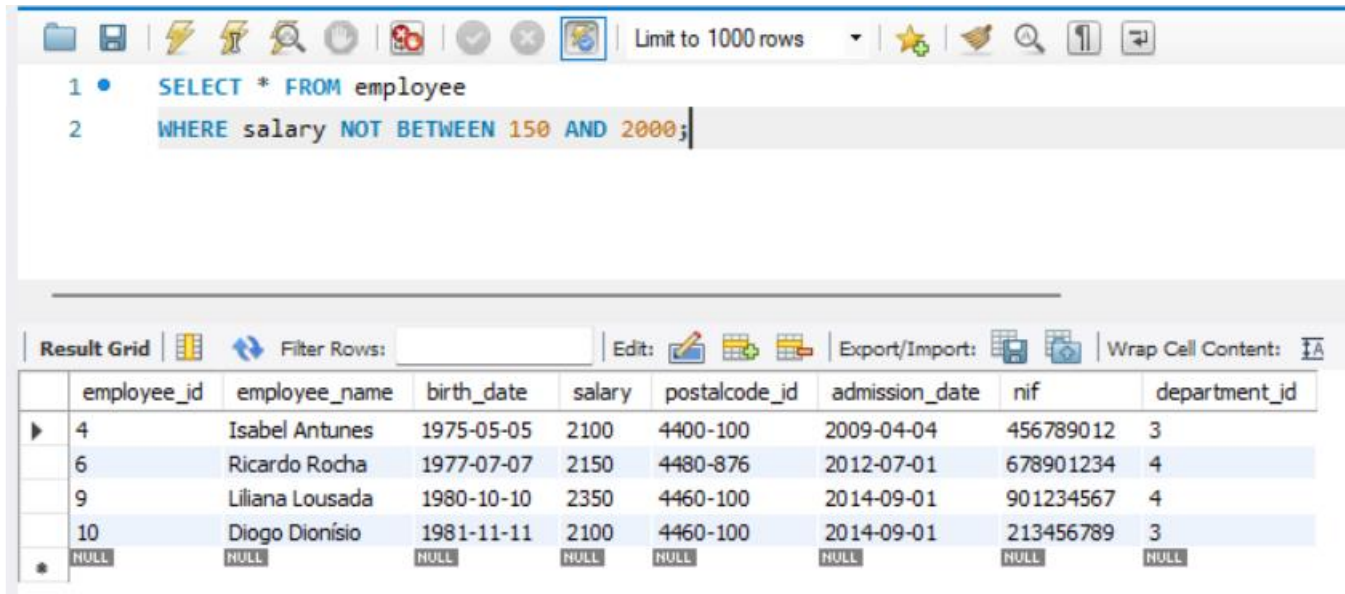
	employee_id	employee_name	birth_date	salary	postalcode_id	admission_date	nif	department_id
▶	1	Manuel Santos	1972-01-01	1550	4000-100	2007-03-03	123456789	1
	2	Paulo Fosneca	1973-03-03	1820	4000-100	2008-01-01	234567890	1
	3	Carla Carolina	1974-04-04	1550	4100-050	2008-01-01	345678901	1
	5	Maria Costa	1976-06-06	1950	4480-876	2010-01-01	567890123	2
	7	José Silva	1978-08-08	1420	4100-050	2012-07-01	789012345	5
	8	Maria Andrade	1979-09-09	1420	4000-100	2014-09-01	890123456	5

BETWEEN: allows you to specify a range of values

SQL

Restrictions: **WHERE** keyword

- ▶ SELECT THE EMPLOYEES WHERE THE SALARY IS NOT BETWEEN 1500 AND 2000



The screenshot shows a SQL query editor with a toolbar at the top. The query is: `SELECT * FROM employee WHERE salary NOT BETWEEN 1500 AND 2000;`. Below the query, the results are displayed in a table with 9 columns: employee_id, employee_name, birth_date, salary, postalcode_id, admission_date, nif, and department_id. The results show 5 rows, with the first 4 rows highlighted in blue. The 5th row is a NULL row.

	employee_id	employee_name	birth_date	salary	postalcode_id	admission_date	nif	department_id
▶	4	Isabel Antunes	1975-05-05	2100	4400-100	2009-04-04	456789012	3
	6	Ricardo Rocha	1977-07-07	2150	4480-876	2012-07-01	678901234	4
	9	Liliana Lousada	1980-10-10	2350	4460-100	2014-09-01	901234567	4
	10	Diogo Dionísio	1981-11-11	2100	4460-100	2014-09-01	213456789	3
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

BETWEEN: allows you to specify a range of values

Restrictions: **WHERE** keyword

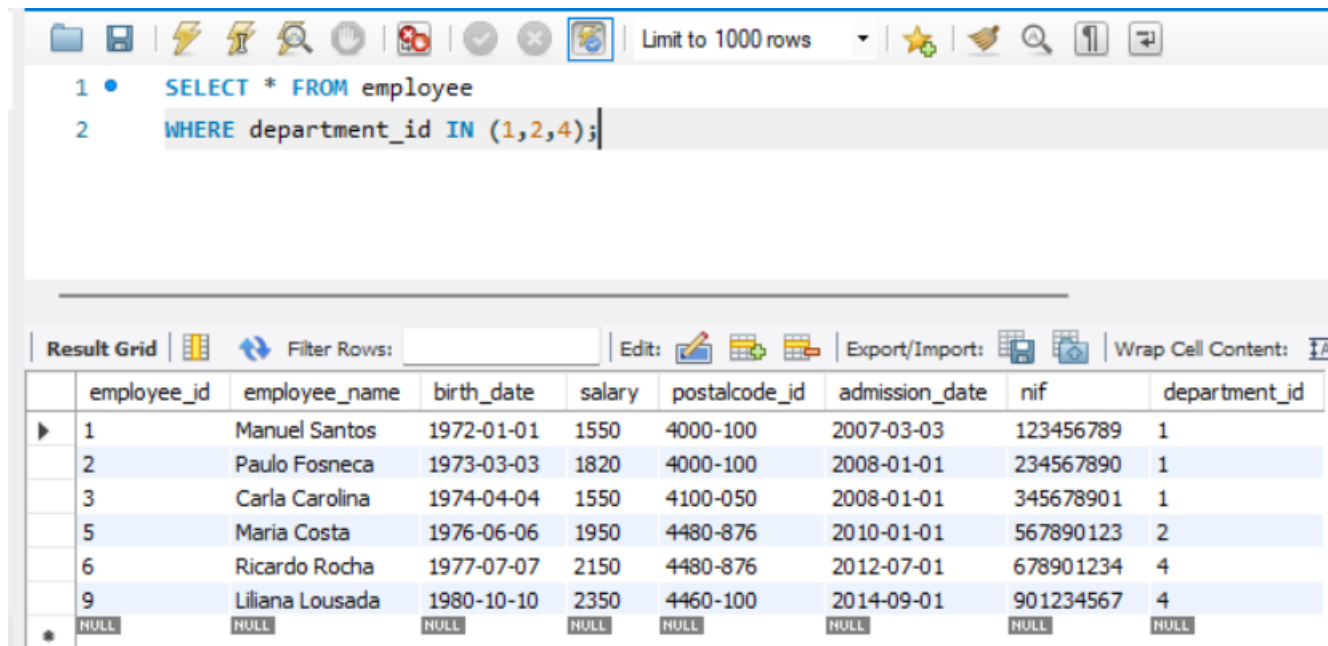
- ▶ SELECT THE EMPLOYEES WHERE THE DEPARTMENT IS BETWEEN 2 AND 4

[illegible]

SQL

Restrictions: **WHERE** keyword

- ▶ SELECT THE EMPLOYEES WHERE THE DEPARTMENT_ID IS ONE OF THE LIST: 1,2,4



The screenshot shows a SQL query editor with a toolbar at the top. The query is as follows:

```
1 • SELECT * FROM employee
2 WHERE department_id IN (1,2,4);
```

Below the query editor is a 'Result Grid' showing the results of the query. The grid has columns for employee_id, employee_name, birth_date, salary, postalcode_id, admission_date, nif, and department_id. The results are as follows:

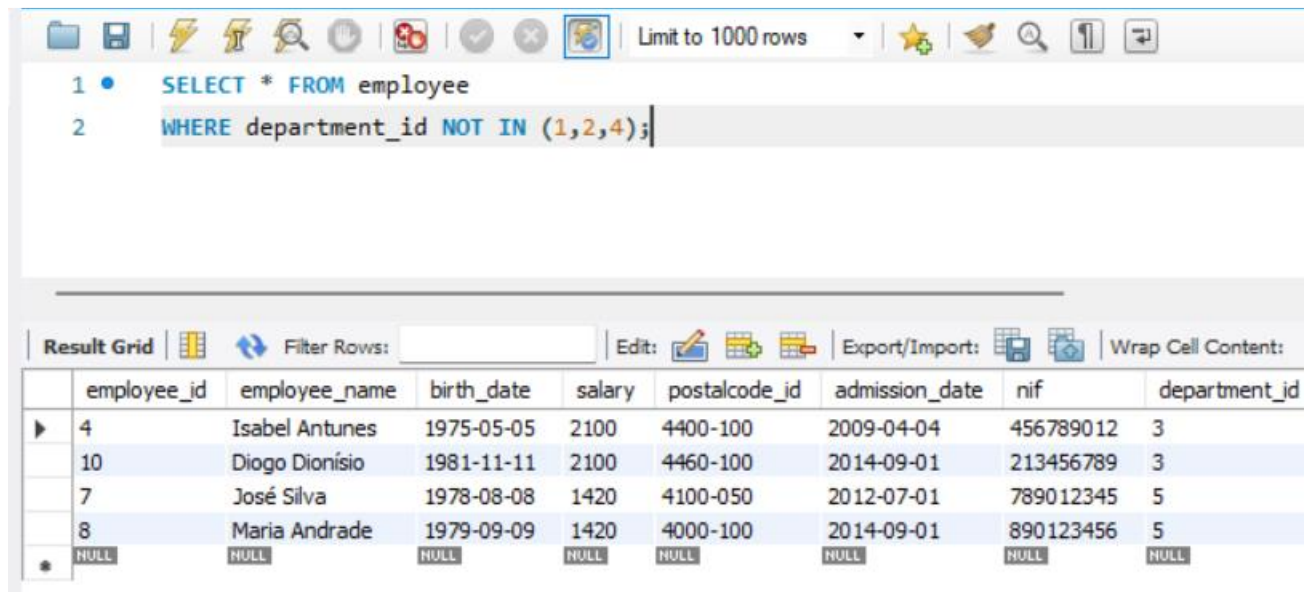
	employee_id	employee_name	birth_date	salary	postalcode_id	admission_date	nif	department_id
▶	1	Manuel Santos	1972-01-01	1550	4000-100	2007-03-03	123456789	1
	2	Paulo Foseca	1973-03-03	1820	4000-100	2008-01-01	234567890	1
	3	Carla Carolina	1974-04-04	1550	4100-050	2008-01-01	345678901	1
	5	Maria Costa	1976-06-06	1950	4480-876	2010-01-01	567890123	2
	6	Ricardo Rocha	1977-07-07	2150	4480-876	2012-07-01	678901234	4
	9	Liliana Lousada	1980-10-10	2350	4460-100	2014-09-01	901234567	4
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

IN : allows you to specify a list of values

SQL

Restrictions: **WHERE** keyword

- ▶ SELECT THE EMPLOYEES WHERE THE DEPARTMENT_ID IS **NOT ON THE LIST: 1,2,4**



```
1 • SELECT * FROM employee
2 WHERE department_id NOT IN (1,2,4);
```

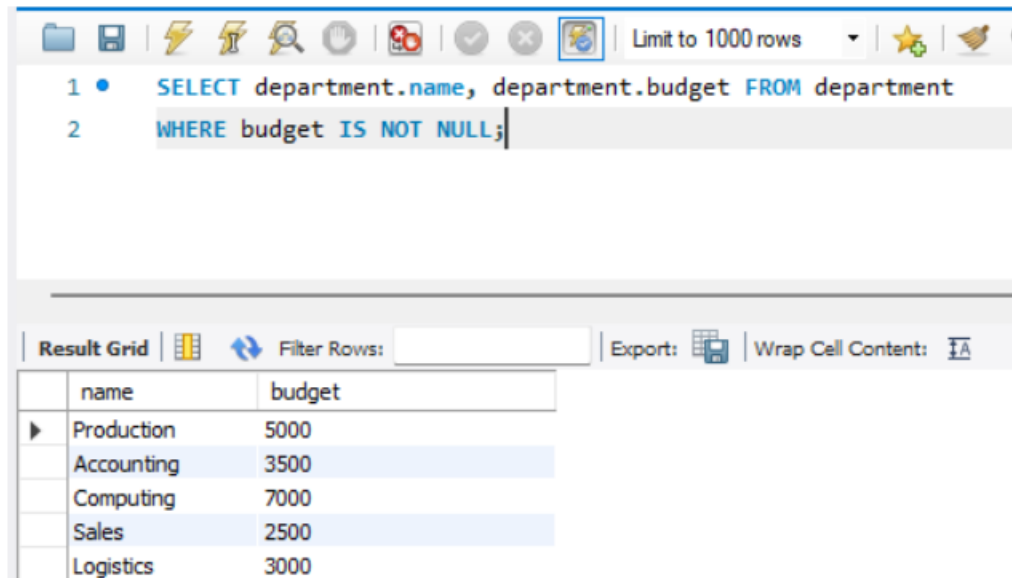
	employee_id	employee_name	birth_date	salary	postalcode_id	admission_date	nif	department_id
▶	4	Isabel Antunes	1975-05-05	2100	4400-100	2009-04-04	456789012	3
	10	Diogo Dionísio	1981-11-11	2100	4460-100	2014-09-01	213456789	3
	7	José Silva	1978-08-08	1420	4100-050	2012-07-01	789012345	5
	8	Maria Andrade	1979-09-09	1420	4000-100	2014-09-01	890123456	5
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

IN : allows you to specify a list of values

SQL

Restrictions: **WHERE** keyword

- ▶ SELECT DEPARTMENTS WITH BUDGET NOT NULL, NOT EMPTY



```
1 • SELECT department.name, department.budget FROM department
2   WHERE budget IS NOT NULL;
```

	name	budget
▶	Production	5000
	Accounting	3500
	Computing	7000
	Sales	2500
	Logistics	3000

NOT NULL : field does not contain null values, is not empty

NULL : field with no value, empty

Notes:

☐ Comparison with null values requires the **IS** operator

☐ **NULL** is not 0, it is a empty field

SQL

Restrictions: **WHERE** keyword

► Operator LIKE

- ❑ Comparing strings with relational operators always results in the comparison of the entire string;
- ❑ Operator Like is used to compare parts of Strings;
- ❑ To do so, we can use two *Wildcards*:
 - ❑ **%** : indicates any character set (0 or more);
 - ❑ **_** : indicates one and only one character

Restrictions: **WHERE** keyword

- [illegible]

Restrictions: **WHERE** keyword

- [illegible]

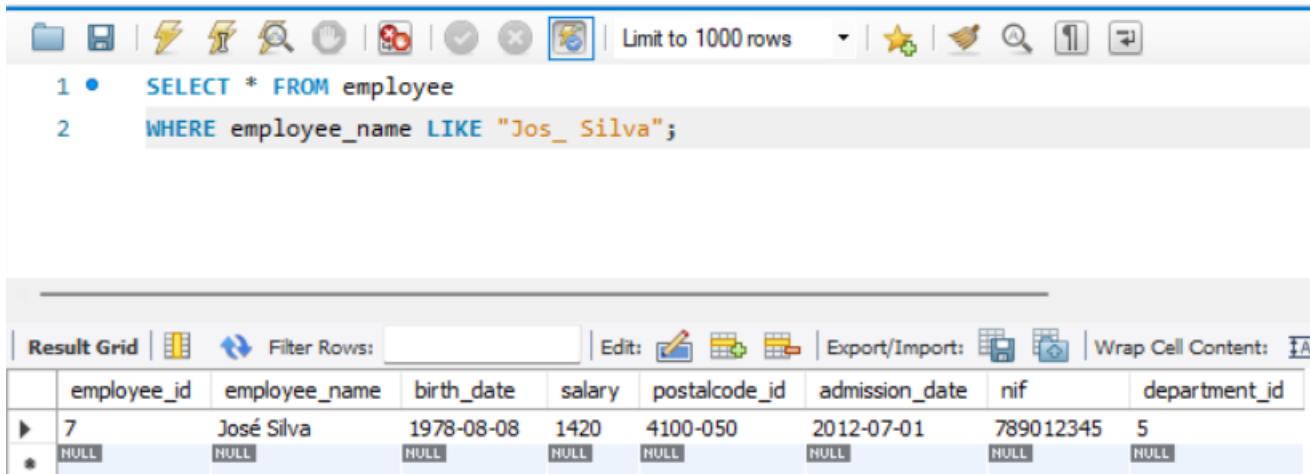
Restrictions: **WHERE** keyword

- ▶ SELECT EMPLOYEES WHERE POSTALCODE INCLUDES "4000"

[illegible]

Restrictions: **WHERE** keyword

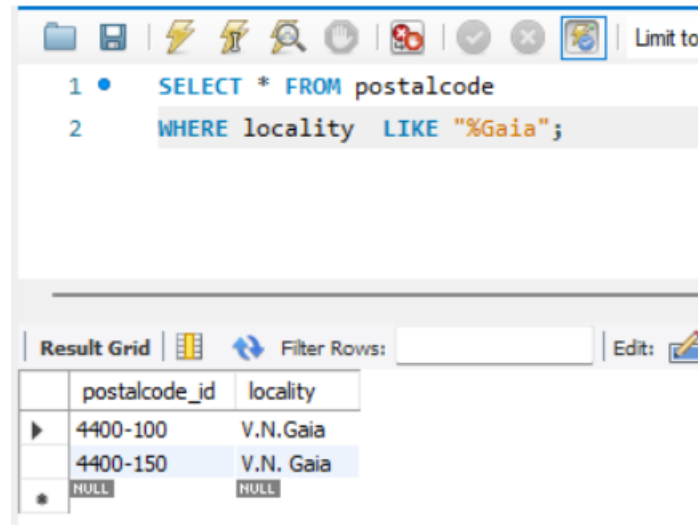
- ▶ SELECT EMPLOYEES WHERE NAME INCLUDES A CHARACTER, WE DON'T KNOW WHICH



SQL

Restrictions: **WHERE** keyword

- ▶ SELECT LOCALITIES THAT ENDS WITH GAIA



SQL

SQL - Aggregation Functions

- ❑ Statistical functions provided by SQL
- ❑ They allow to obtain statistical information about sets of records specified in a SELECT, namely in the WHERE clause or on data groups - GROUP BY clause

Functions	Description
COUNT(*)	returns the number of rows/records obtained in a select
MAX	returns the highest value
MIN	returns the lowest value
SUM	returns the sum of a column
AVERAGE	returns the average of the values of a column

SQL

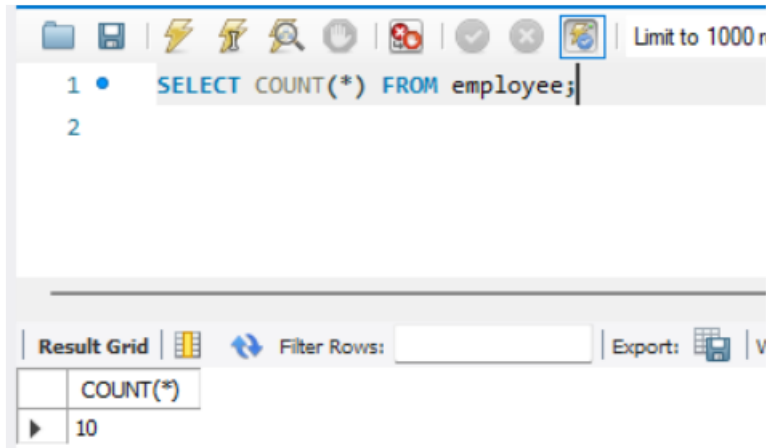
SQL – COUNT function

- ❑ COUNT(*) number of rows that results from a select
- ❑ COUNT(coluna) number of rows in the indicated column,
provided that different from Null
- ❑ COUNT (DISTINCT coluna) number of distinct lines in the indicated column

SQL

SQL – COUNT

SOME EXAMPLES...

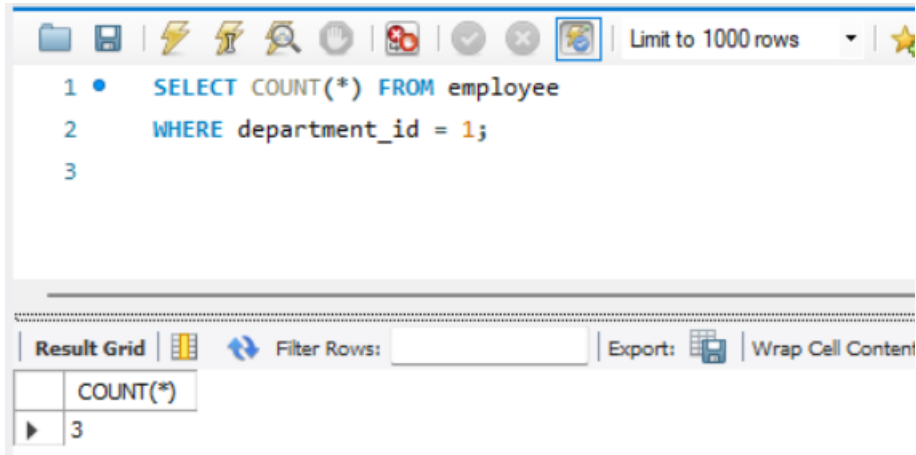


The screenshot shows the SQL Developer interface. The query editor contains the following SQL statement:

```
1 • SELECT COUNT(*) FROM employee;  
2
```

The toolbar at the top includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. Below the query editor, the 'Result Grid' tab is active, displaying the following result:

COUNT(*)
10



The screenshot shows the SQL Developer interface. The query editor contains the following SQL statement:

```
1 • SELECT COUNT(*) FROM employee  
2 WHERE department_id = 1;  
3
```

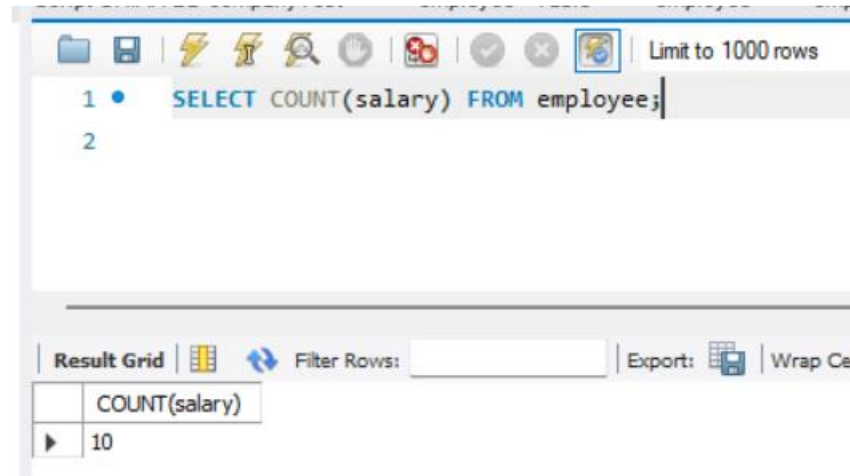
The toolbar at the top includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. Below the query editor, the 'Result Grid' tab is active, displaying the following result:

COUNT(*)
3

SQL

SQL – COUNT

SOME EXAMPLES...

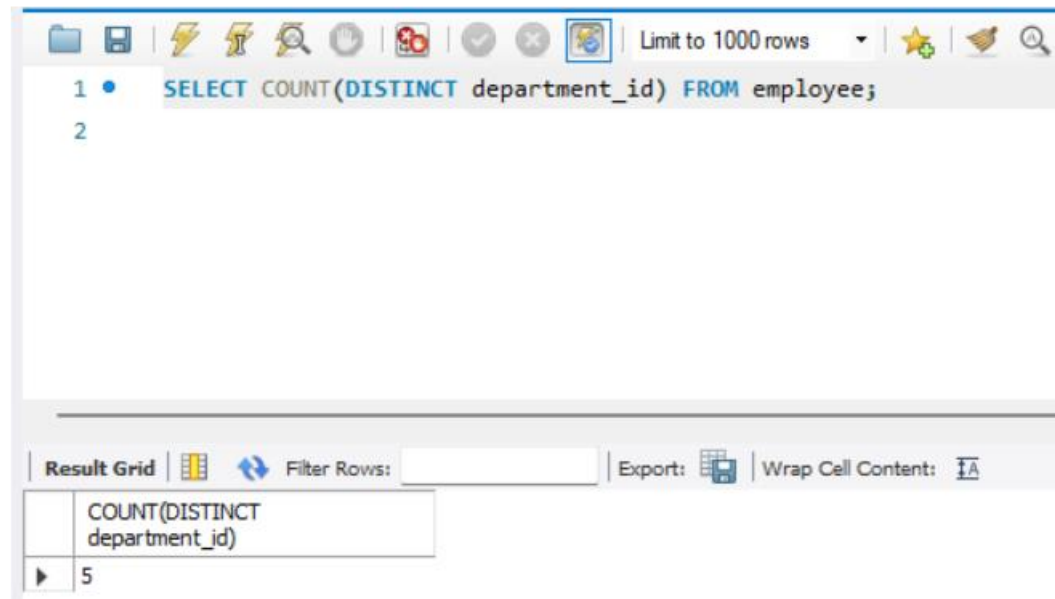


The screenshot shows a SQL IDE window with a toolbar at the top. The query editor contains the following SQL statement:

```
1 • SELECT COUNT(salary) FROM employee;
2
```

Below the query editor, the 'Result Grid' tab is active. It displays the following result:

COUNT(salary)
10



The screenshot shows a SQL IDE window with a toolbar at the top. The query editor contains the following SQL statement:

```
1 • SELECT COUNT(DISTINCT department_id) FROM employee;
2
```

Below the query editor, the 'Result Grid' tab is active. It displays the following result:

COUNT(DISTINCT department_id)
5

SQL

SQL – COUNT

SOME EXAMPLES...

The screenshot shows a SQL query editor interface. The query is: `SELECT COUNT(DISTINCT department_id) as numberdeparts FROM employee;`. The text `as numberdeparts` is highlighted with a red box. Below the query, the result grid is displayed, showing a single row with the column header `numberdeparts` and the value `5`. The result grid is also highlighted with a red box. The interface includes a toolbar at the top with icons for file operations, a 'Limit to 1000 rows' dropdown, and a 'Result Grid' tab at the bottom.

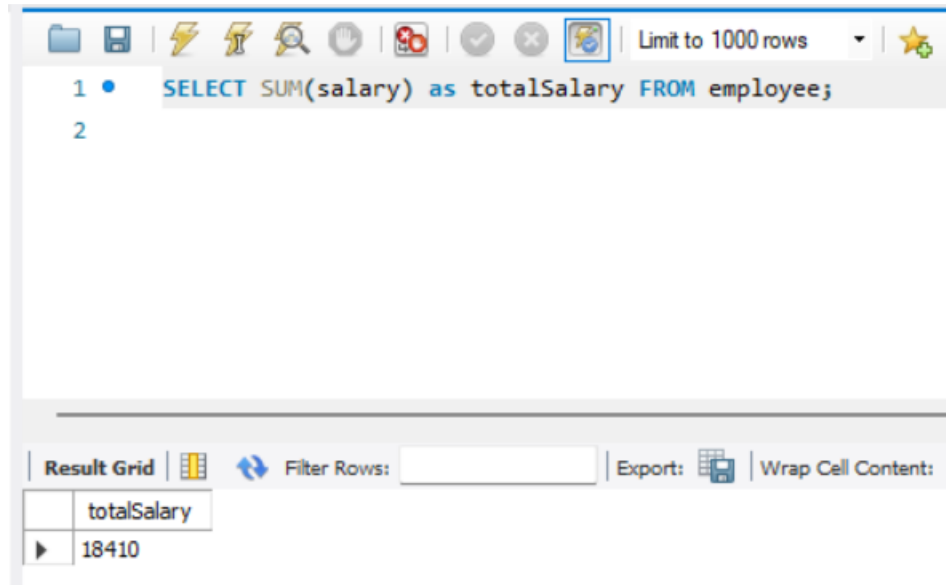
```
1 • SELECT COUNT(DISTINCT department_id) as numberdeparts FROM employee;
2
```

numberdeparts
5

SQL

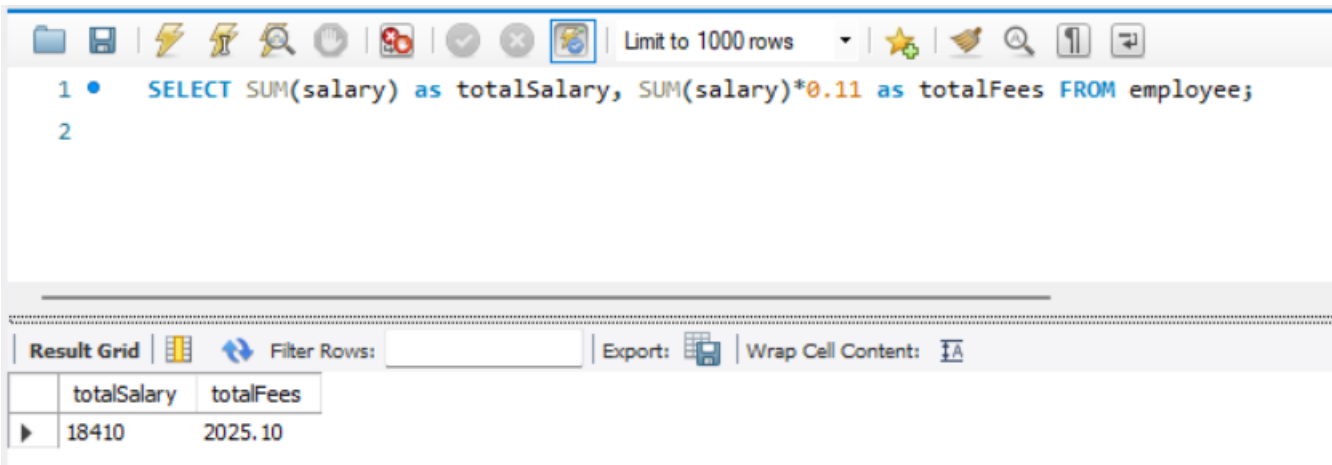
SQL – SUM

SOME EXAMPLES...



The screenshot shows a SQL query editor with a toolbar at the top. The query is: `SELECT SUM(salary) as totalSalary FROM employee;`. Below the query, the results are displayed in a table with one column, `totalSalary`, and one row with the value `18410`.

totalSalary
18410



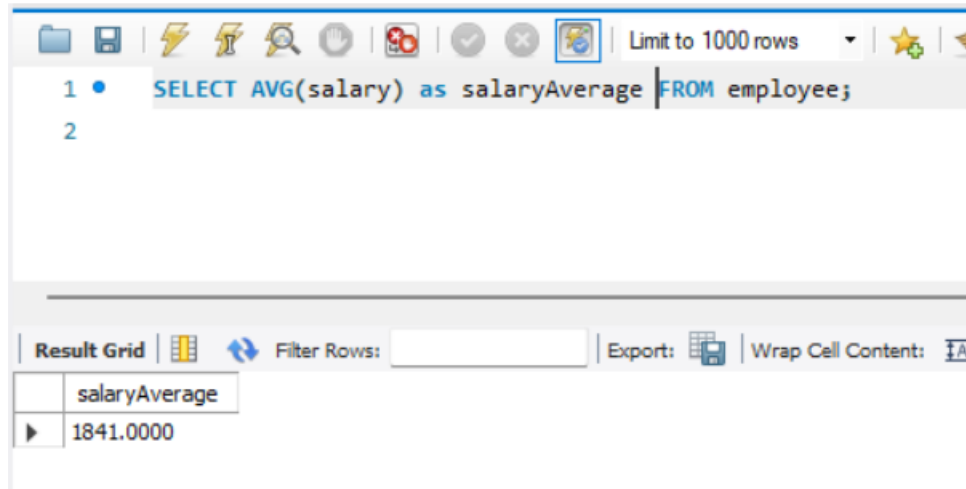
The screenshot shows a SQL query editor with a toolbar at the top. The query is: `SELECT SUM(salary) as totalSalary, SUM(salary)*0.11 as totalFees FROM employee;`. Below the query, the results are displayed in a table with two columns, `totalSalary` and `totalFees`, and one row with the values `18410` and `2025.10`.

totalSalary	totalFees
18410	2025.10

SQL

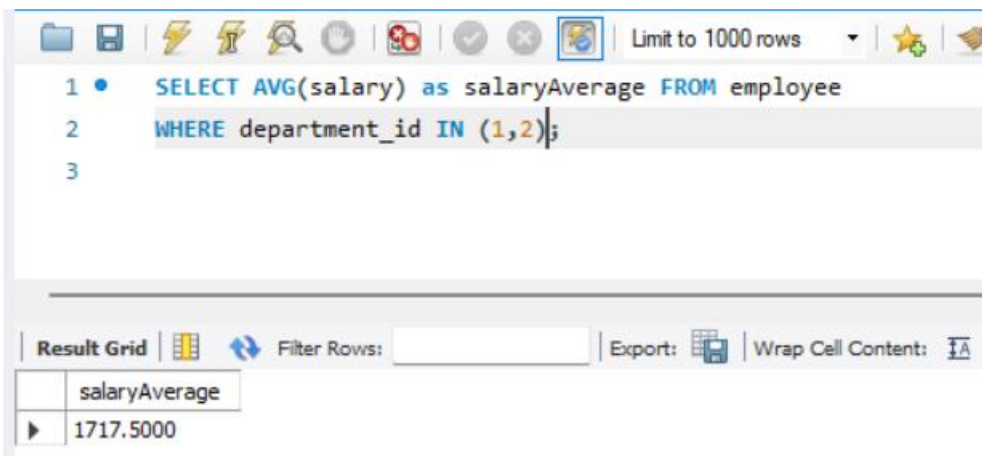
SQL – AVG

SOME EXAMPLES...



The screenshot shows the SQL Developer interface. The SQL Editor contains the query: `SELECT AVG(salary) as salaryAverage FROM employee;`. The Results window displays the following data:

salaryAverage
1841.0000



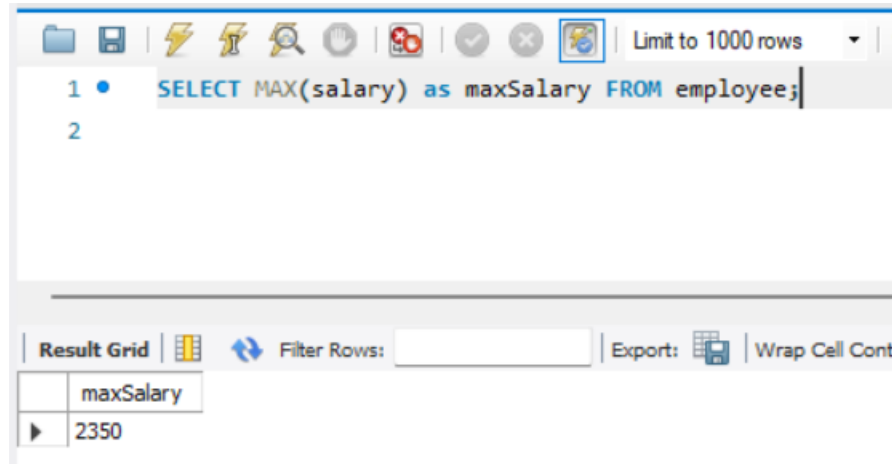
The screenshot shows the SQL Developer interface. The SQL Editor contains the query: `SELECT AVG(salary) as salaryAverage FROM employee WHERE department_id IN (1,2);`. The Results window displays the following data:

salaryAverage
1717.5000

SQL

SQL – MAX and MIN functions

SOME EXAMPLES...

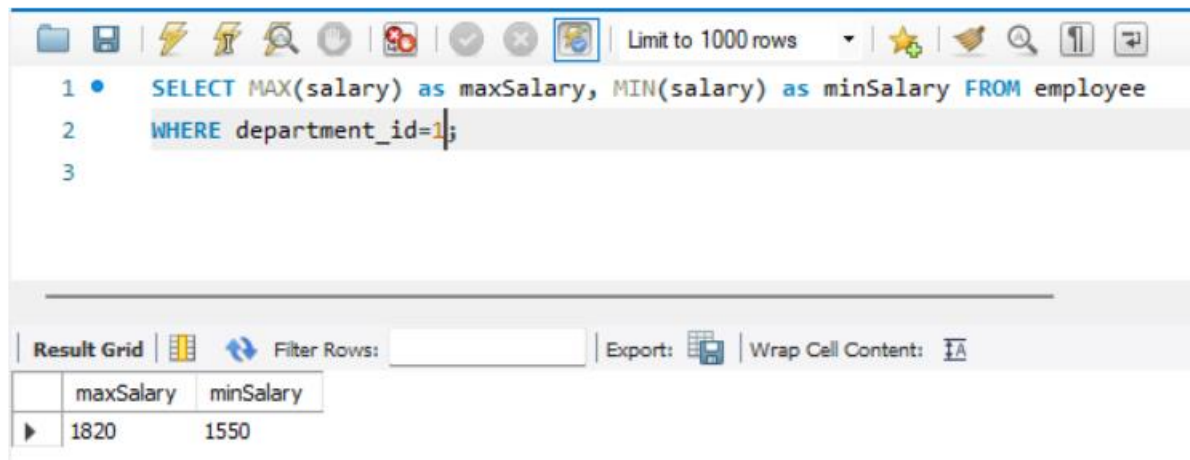


The screenshot shows a SQL IDE interface. The query editor contains the following SQL statement:

```
1 • SELECT MAX(salary) as maxSalary FROM employee;
```

The interface includes a toolbar with various icons and a dropdown menu set to "Limit to 1000 rows". Below the query editor, there is a "Result Grid" section with a table showing the result of the query:

	maxSalary
▶	2350



The screenshot shows a SQL IDE interface. The query editor contains the following SQL statement:

```
1 • SELECT MAX(salary) as maxSalary, MIN(salary) as minSalary FROM employee  
2 WHERE department_id=1;  
3
```

The interface includes a toolbar with various icons and a dropdown menu set to "Limit to 1000 rows". Below the query editor, there is a "Result Grid" section with a table showing the result of the query:

	maxSalary	minSalary
▶	1820	1550

SQL

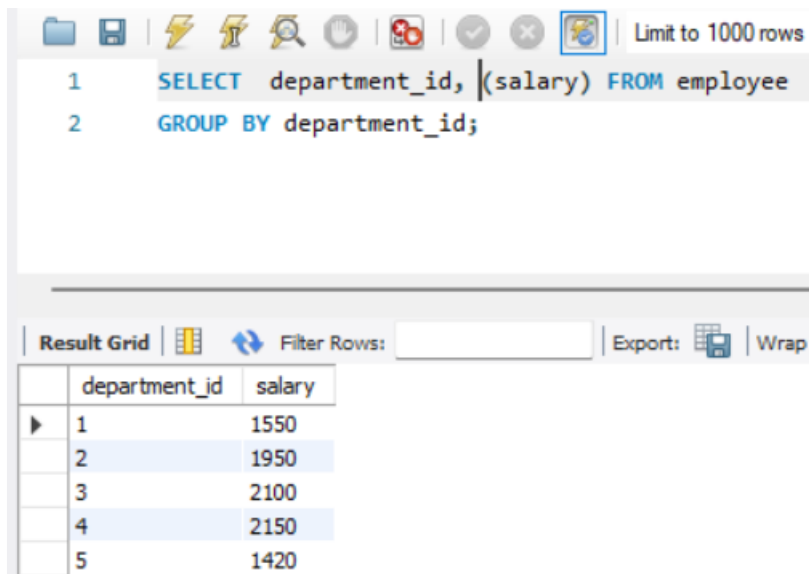
SQL – GROUP BY

- ❑ Let's you group data that results from a select
- ❑ Data Grouping is particularly useful when coupled with aggregation functions
- ❑ GROUP BY Clause:
 - ❑ It is used to group data
 - ❑ The records are processed in groups of similar characteristics
 - ❑ By associating aggregation functions (SUM,AVG, COUNT,...) we can obtain statistical data about each data group

```
SELECT [distinct/all] field1, field2, ..., fieldn [*]  
FROM tables  
[WHERE conditions]  
[GROUP BY fields]  
[HAVING conditions]  
[ORDER BY fields]
```

SQL

SQL – GROUP BY

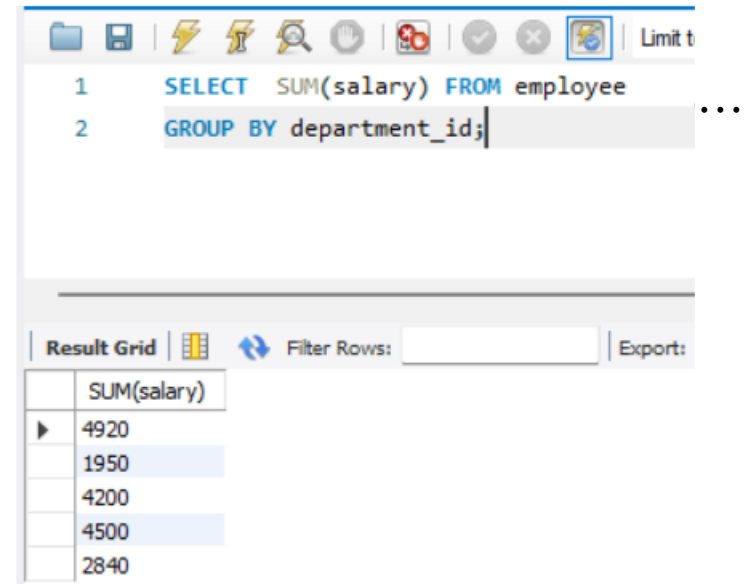


The screenshot shows a SQL IDE interface. The query editor contains the following SQL code:

```
1 SELECT department_id, SUM(salary) FROM employee
2 GROUP BY department_id;
```

Below the query editor is the 'Result Grid' tab. It displays the results of the query in a table with two columns: 'department_id' and 'salary'. The table has 5 rows of data.

	department_id	salary
▶	1	1550
	2	1950
	3	2100
	4	2150
	5	1420



The screenshot shows a SQL IDE interface. The query editor contains the following SQL code:

```
1 SELECT SUM(salary) FROM employee
2 GROUP BY department_id;
```

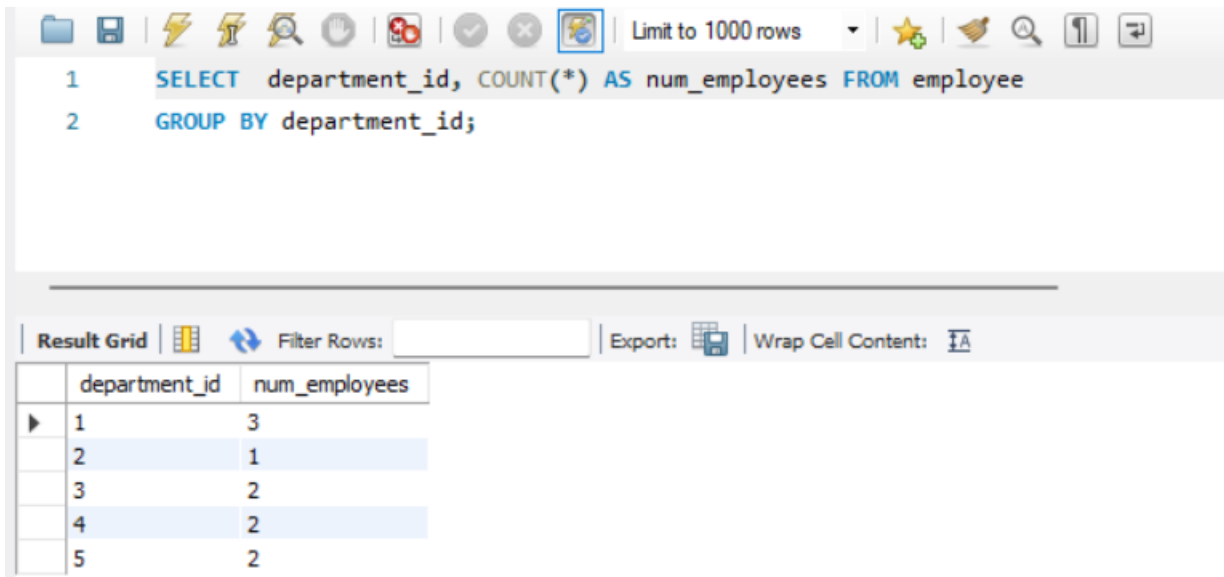
Below the query editor is the 'Result Grid' tab. It displays the results of the query in a table with one column: 'SUM(salary)'. The table has 5 rows of data.

	SUM(salary)
▶	4920
	1950
	4200
	4500
	2840

SQL

SQL – GROUP BY

SOME EXAMPLES...



The screenshot shows a SQL query editor interface. The query is as follows:

```
1 SELECT department_id, COUNT(*) AS num_employees FROM employee
2 GROUP BY department_id;
```

Below the query, the results are displayed in a table with the following data:

department_id	num_employees
1	3
2	1
3	2
4	2
5	2

SQL

SQL – GROUP BY ... HAVING

□ HAVING Clause

- Used to enforce constraints at the data grouping level
- Acts on data groups
- The **WHERE** clause imposes constraints on the level of data selection;
- The **HAVING** clause allows you to restrict data at the groups that are formed, after applying data selection

```
SELECT [distinct/all] field1, field2, ..., fieldn [*]  
FROM tables  
[WHERE conditions]  
[GROUP BY fields]  
[HAVING conditions]  
[ORDER BY fields]
```

SQL

SQL – HAVING

```
1 SELECT department_id, AVG(salary) as salary_average, SUM(salary) AS sum_salary FROM employee
2 GROUP BY department_id HAVING department_id <> 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	department_id	salary_average	sum_salary
▶	2	1950.0000	1950
	3	2100.0000	4200
	4	2250.0000	4500
	5	1420.0000	2840

```
1 SELECT department_id, AVG(salary) as salary_average, SUM(salary) AS sum_salary FROM employee
2 GROUP BY department_id HAVING SUM(salary) >3000;
```

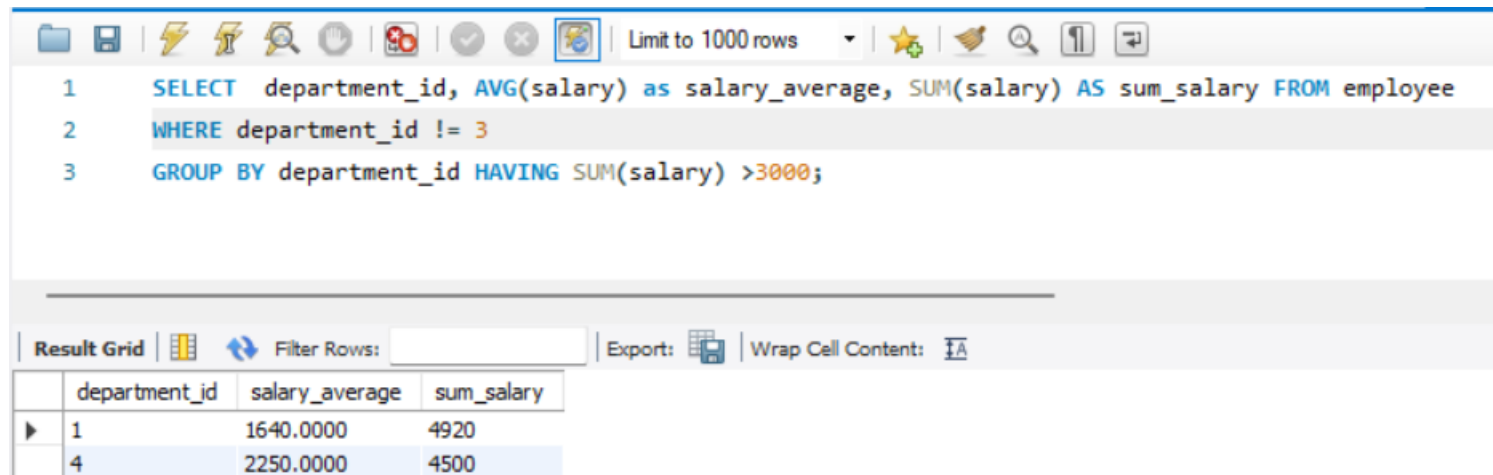
Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	department_id	salary_average	sum_salary
▶	1	1640.0000	4920
	3	2100.0000	4200
	4	2250.0000	4500

SQL

SQL – HAVING

SOME EXAMPLES...



The screenshot shows a SQL query editor interface. The query is as follows:

```
1 SELECT department_id, AVG(salary) as salary_average, SUM(salary) AS sum_salary FROM employee
2 WHERE department_id != 3
3 GROUP BY department_id HAVING SUM(salary) >3000;
```

Below the query, the results are displayed in a table. The table has three columns: department_id, salary_average, and sum_salary. The results are as follows:

	department_id	salary_average	sum_salary
▶	1	1640.0000	4920
	4	2250.0000	4500

SQL

SQL- ORDER BY

- ❑ Let's you sort the resulting data from a select
- ❑ The ordering is based on the ASCII table
- ❑ Digits appear before alphabetic characters; uppercase to lowercase letters
- ❑ Some DBMS are case sensitive, others are not ... (MySQL is not case sensitive)
- ❑ ORDER BY clause
 - ❑ The ordering can be Ascending - ASC or Descending - DESC
 - ❑ By default, the sort is ascending

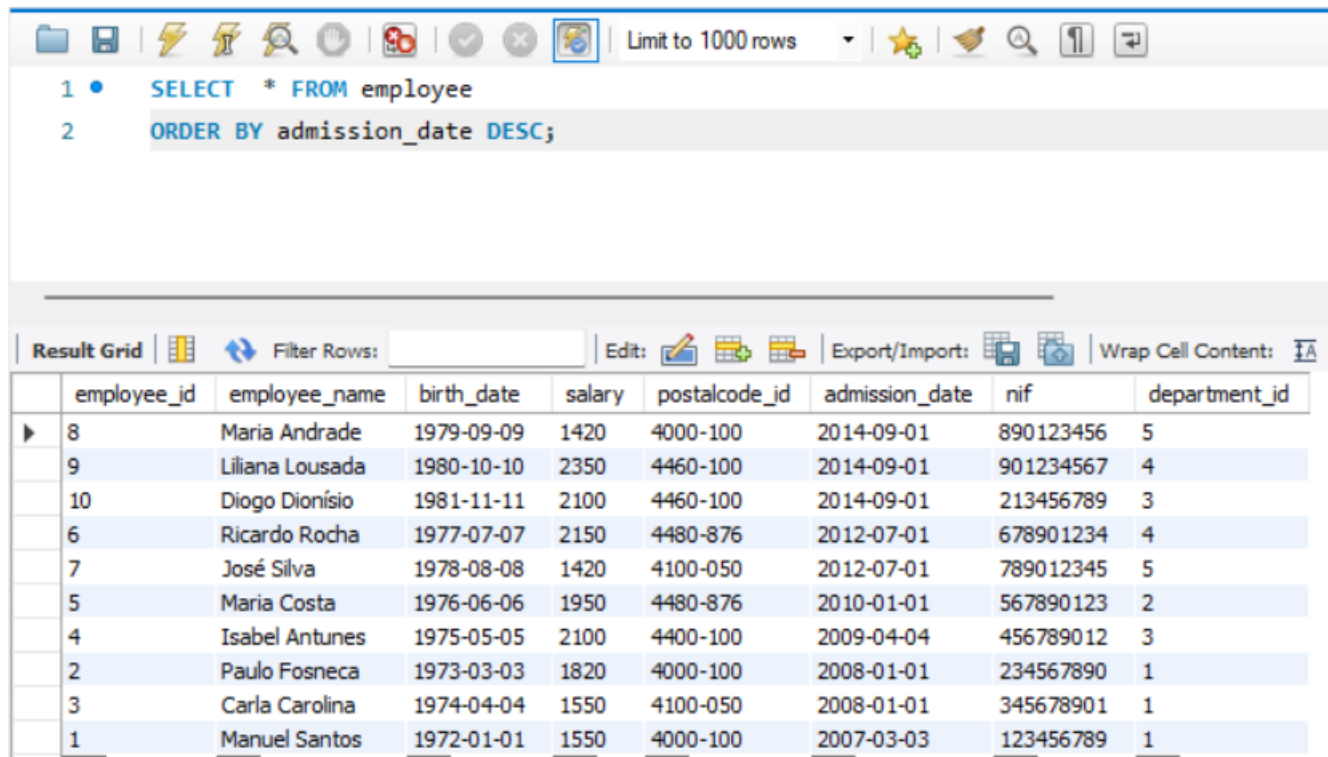
```
SELECT [distinct/all] field1, field2, ..., fieldn [*]  
FROM tables  
[WHERE conditions]  
[GROUP BY fields]  
[HAVING conditions]  
[ORDER BY fields] [ASC | DESC]
```

SQL

SQL – ORDER BY

- ▶ Sorting ASC ou DESC

SOME EXAMPLES...



The screenshot shows a SQL query editor interface. At the top, there is a toolbar with various icons and a dropdown menu set to 'Limit to 1000 rows'. Below the toolbar, the SQL query is entered in two lines:

```
1 • SELECT * FROM employee
2 ORDER BY admission_date DESC;
```

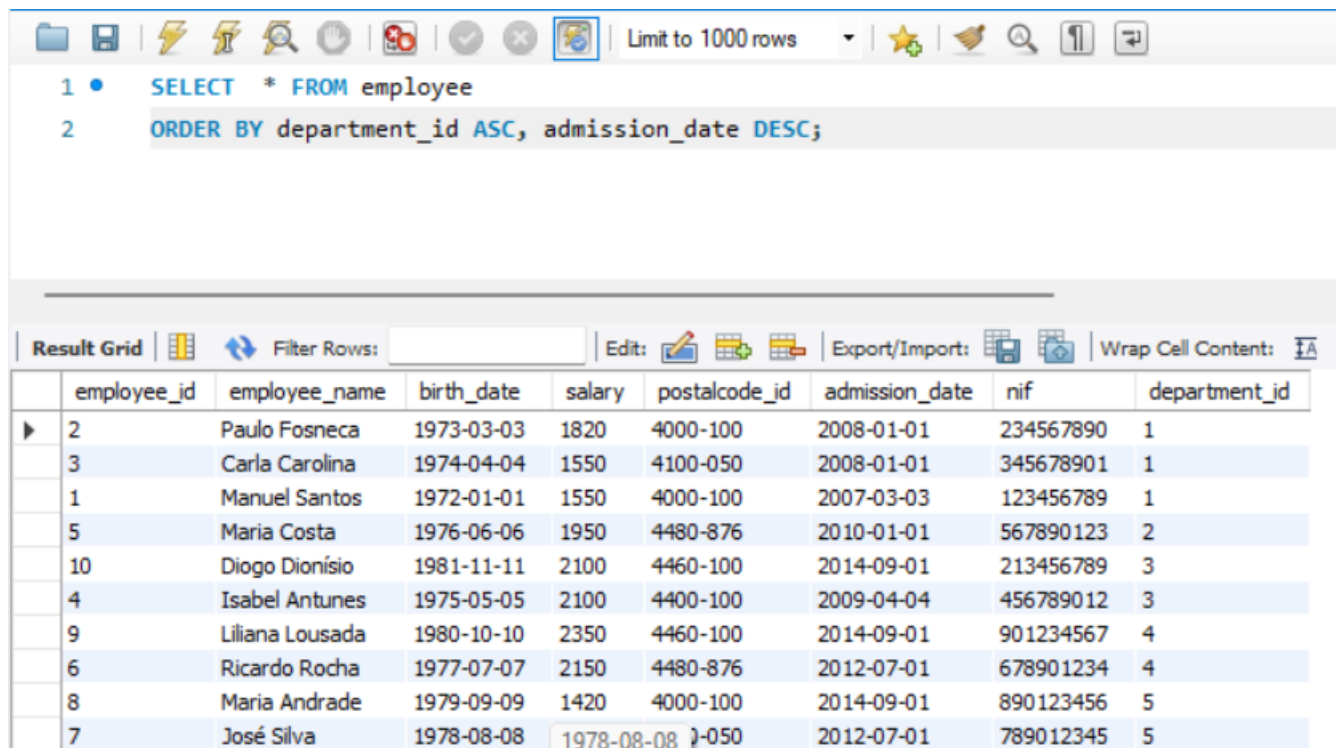
Below the query editor, there is a 'Result Grid' section. It includes a 'Filter Rows:' input field, an 'Edit:' button, and an 'Export/Import:' button. The result grid displays the following data:

	employee_id	employee_name	birth_date	salary	postalcode_id	admission_date	nif	department_id
▶	8	Maria Andrade	1979-09-09	1420	4000-100	2014-09-01	890123456	5
	9	Liliana Lousada	1980-10-10	2350	4460-100	2014-09-01	901234567	4
	10	Diogo Dionísio	1981-11-11	2100	4460-100	2014-09-01	213456789	3
	6	Ricardo Rocha	1977-07-07	2150	4480-876	2012-07-01	678901234	4
	7	José Silva	1978-08-08	1420	4100-050	2012-07-01	789012345	5
	5	Maria Costa	1976-06-06	1950	4480-876	2010-01-01	567890123	2
	4	Isabel Antunes	1975-05-05	2100	4400-100	2009-04-04	456789012	3
	2	Paulo Fosneca	1973-03-03	1820	4000-100	2008-01-01	234567890	1
	3	Carla Carolina	1974-04-04	1550	4100-050	2008-01-01	345678901	1
	1	Manuel Santos	1972-01-01	1550	4000-100	2007-03-03	123456789	1

SQL

SQL – ORDER BY

- ▶ Sort with sub-ordering



The screenshot shows a SQL query editor interface. The query is as follows:

```
1 • SELECT * FROM employee
2 ORDER BY department_id ASC, admission_date DESC;
```

Below the query, the 'Result Grid' displays the results of the query. The grid has 9 columns: employee_id, employee_name, birth_date, salary, postalcode_id, admission_date, nif, and department_id. The results are sorted by department_id in ascending order, and for identical values of department_id, they are sorted by admission_date in descending order.

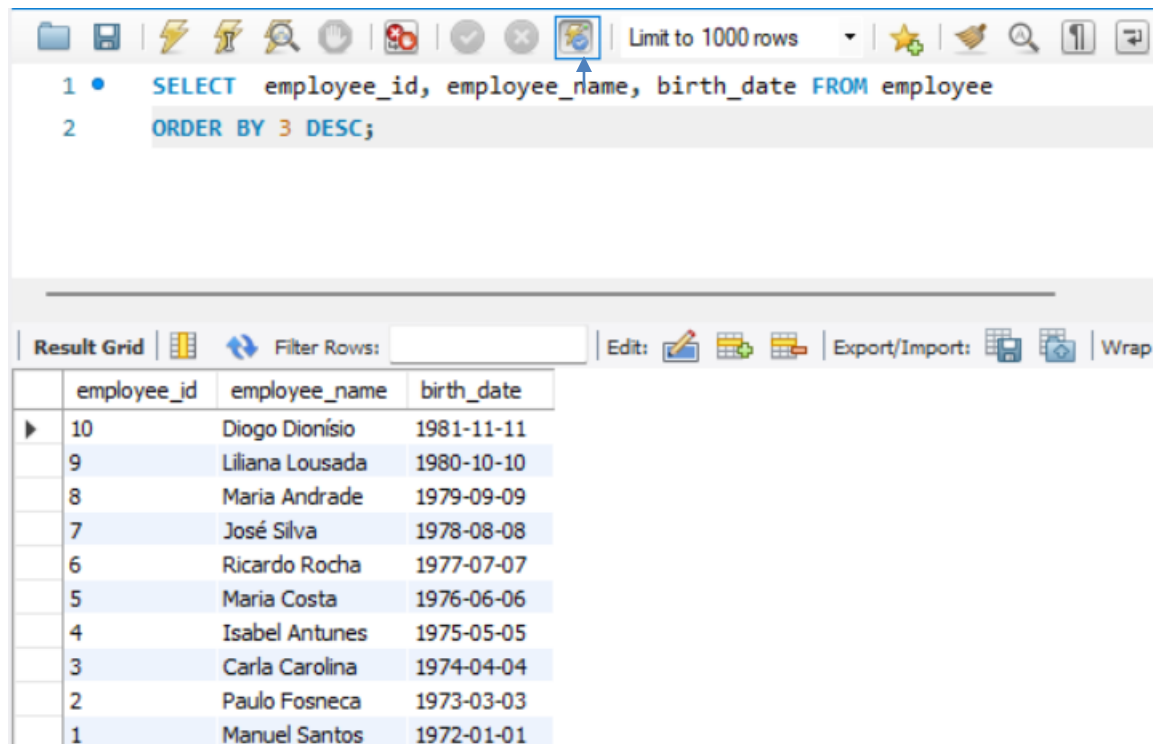
	employee_id	employee_name	birth_date	salary	postalcode_id	admission_date	nif	department_id
▶	2	Paulo Fosneca	1973-03-03	1820	4000-100	2008-01-01	234567890	1
	3	Carla Carolina	1974-04-04	1550	4100-050	2008-01-01	345678901	1
	1	Manuel Santos	1972-01-01	1550	4000-100	2007-03-03	123456789	1
	5	Maria Costa	1976-06-06	1950	4480-876	2010-01-01	567890123	2
	10	Diogo Dionísio	1981-11-11	2100	4460-100	2014-09-01	213456789	3
	4	Isabel Antunes	1975-05-05	2100	4400-100	2009-04-04	456789012	3
	9	Liliana Lousada	1980-10-10	2350	4460-100	2014-09-01	901234567	4
	6	Ricardo Rocha	1977-07-07	2150	4480-876	2012-07-01	678901234	4
	8	Maria Andrade	1979-09-09	1420	4000-100	2014-09-01	890123456	5
	7	José Silva	1978-08-08	1978-08-08	4100-050	2012-07-01	789012345	5

Sorting is performed by the indicated first column; for identical values column, it orders by the second column (sub-ordering)

SQL

SQL – ORDER BY

- ▶ Sorting by column position in the query



The screenshot shows a SQL query editor interface. The query is as follows:

```
1 • SELECT employee_id, employee_name, birth_date FROM employee
2 ORDER BY 3 DESC;
```

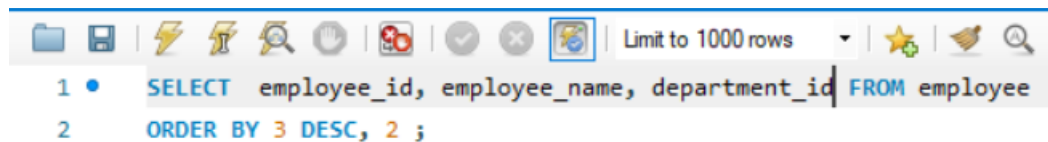
The query is executed, and the result is displayed in a table below the query editor. The table has four columns: `employee_id`, `employee_name`, and `birth_date`. The results are sorted by `birth_date` in descending order.

	employee_id	employee_name	birth_date
▶	10	Diogo Dionísio	1981-11-11
	9	Liliana Lousada	1980-10-10
	8	Maria Andrade	1979-09-09
	7	José Silva	1978-08-08
	6	Ricardo Rocha	1977-07-07
	5	Maria Costa	1976-06-06
	4	Isabel Antunes	1975-05-05
	3	Carla Carolina	1974-04-04
	2	Paulo Fosneca	1973-03-03
	1	Manuel Santos	1972-01-01

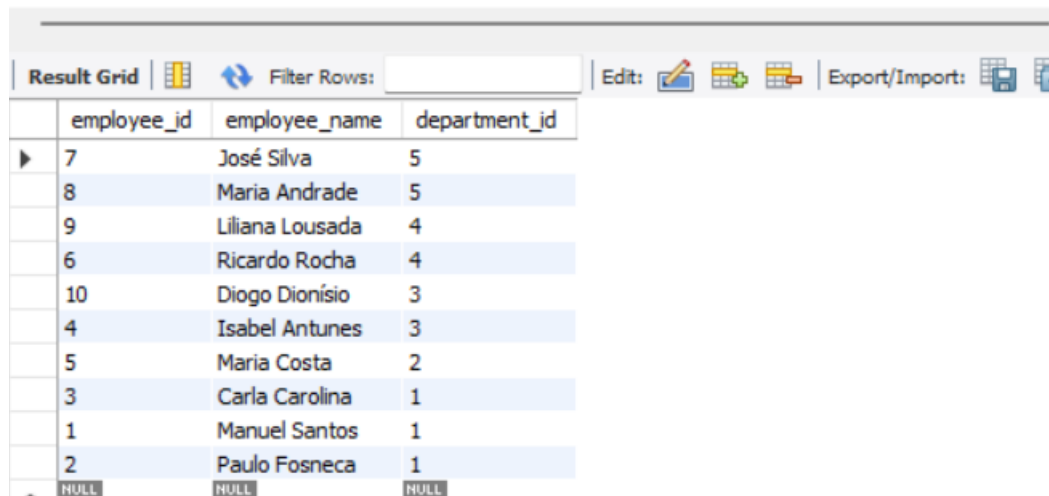
SQL

SQL – ORDER BY

- ▶ Sorting by column position in the query



```
1 • SELECT employee_id, employee_name, department_id FROM employee
2 ORDER BY 3 DESC, 2 ;
```

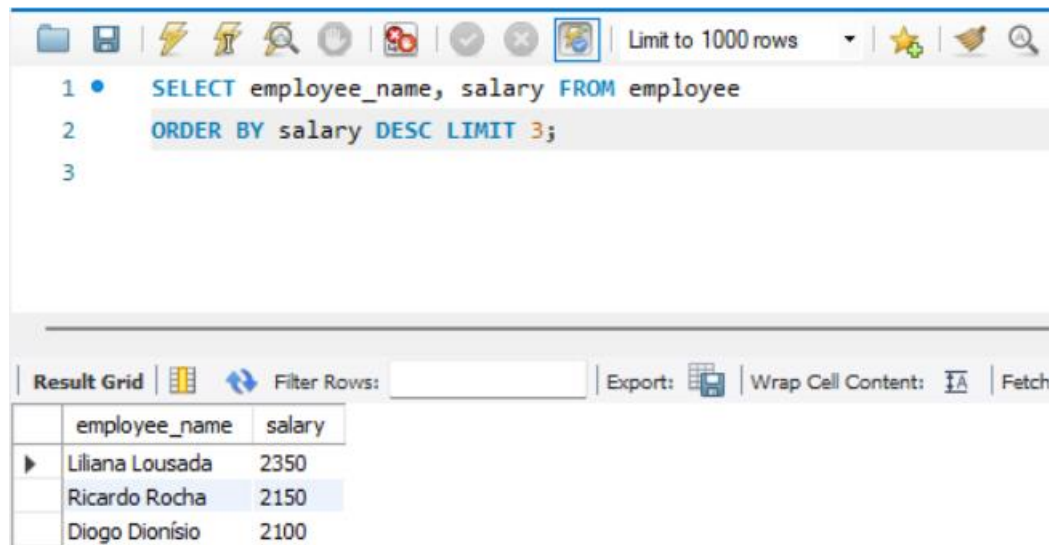


	employee_id	employee_name	department_id
▶	7	José Silva	5
	8	Maria Andrade	5
	9	Liliana Lousada	4
	6	Ricardo Rocha	4
	10	Diogo Dionísio	3
	4	Isabel Antunes	3
	5	Maria Costa	2
	3	Carla Carolina	1
	1	Manuel Santos	1
	2	Paulo Fosneca	1
▲	NULL	NULL	NULL

SQL

SQL – LIMIT

- ❑ It limits the number of records returned in the query



The screenshot shows a SQL query editor with a toolbar at the top. The query text is:

```
1 • SELECT employee_name, salary FROM employee
2 ORDER BY salary DESC LIMIT 3;
3
```

Below the query editor is the 'Result Grid' section. It includes a 'Filter Rows' input field, an 'Export' button, a 'Wrap Cell Content' checkbox, and a 'Fetch' button. The result grid displays the following data:

	employee_name	salary
▶	Liliana Lousada	2350
	Ricardo Rocha	2150
	Diogo Dionísio	2100

It returns the top 3

SQL

SQL – LIMIT

- ❑ It limits the number of records returned in the query

The screenshot shows a SQL query editor with the following query:

```
1 • SELECT employee_name, salary FROM employee
2 ORDER BY salary DESC LIMIT 1;
3
```

The result grid below the query shows one record:

employee_name	salary
Liliana Lousada	2350

It returns the top 1

It returns the 3 first records
(without sorted data)

The screenshot shows a SQL query editor with the following query:

```
1 • SELECT employee_name, salary FROM employee
2 LIMIT 3;
3
```

The result grid below the query shows three records:

employee_name	salary
Manuel Santos	1550
Paulo Fosneca	1820
Carla Carolina	1550