

**DATABASES**  
**SQL**  
**Data Manipulation Language**  
**Part III**

TECNOLOGIAS E SISTEMAS DE INFORMAÇÃO PARA A  
WEB

# Agenda

## SQL – Structured Query Language – Data Manipulation Language

### ❖ SELECT with *SubQueries*

#### ❖ *Subqueries*

#### ❖ Operator IN

#### ❖ Operator EXISTS

### ❖ INSERT

### ❖ UPDATE

### ❖ DELETE

#### Relações

Relações na 1FN

Relações na 2FN

Relações na 3FN

Relações na 4FN

Relações na 5FN



# SQL

# SQL

---

## ▶ SELECT

- ▶ *SubQuery* - Consists of one SELECT within another SELECT.

It results from the ability to integrate chained queries into each other.

- ▶ A SELECT can be placed:

- Within another SELECT in clauses WHERE or HAVING
- Within a SubQuery, under the above conditions
- Within an INSERT, UPDATE, or DELETE
- In the definition of a VIEW

also in FROM

# SQL

---

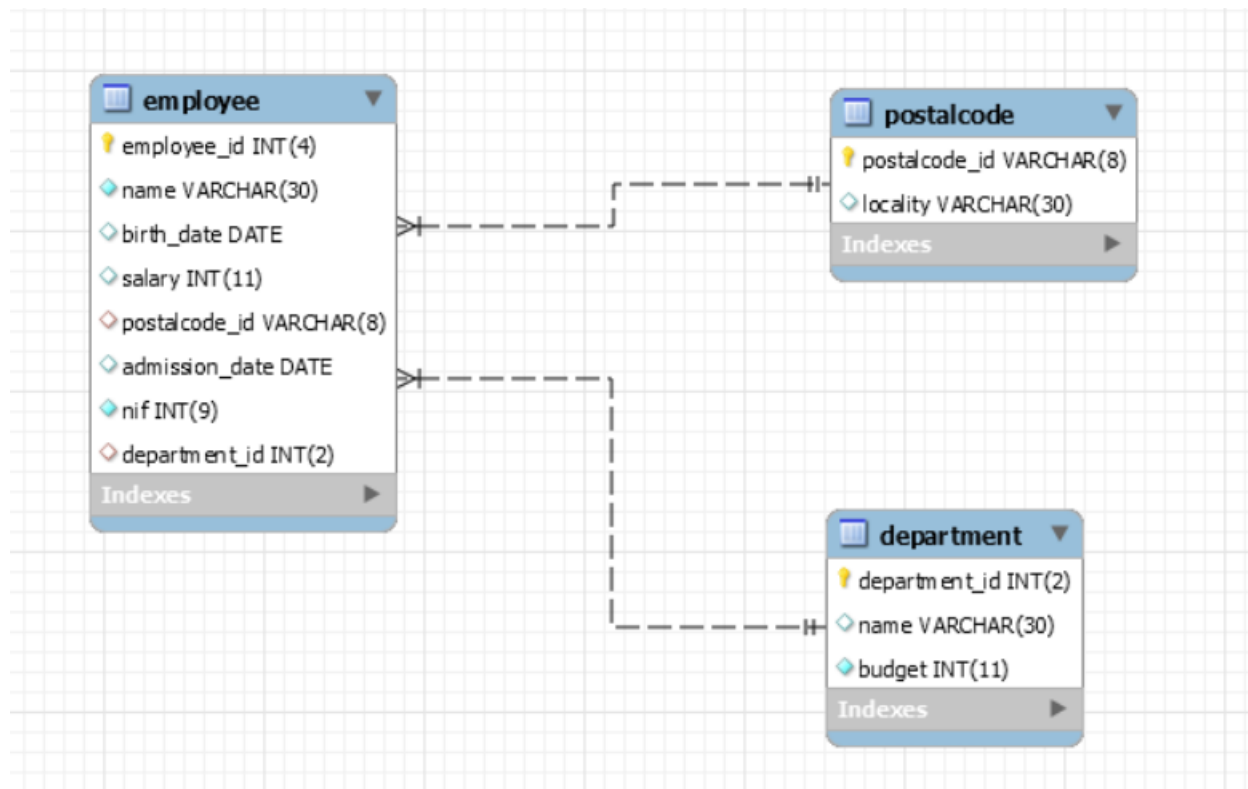
## ▶ SELECT

- ▶ **SubQuery** - There is no defined limit for the presence of chained queries in a SQL statement.
- ▶ However, it is not common to observe more than 2 or 3 queries chained.

```
SELECT [distinct/all] field1, field2, ..., fieldn [*]  
FROM table1, table2...tablen  
[WHERE condition] [= SubQuery]  
[GROUP BY fields]  
[HAVING conditions] [= SubQuery]  
[ORDER BY fields]
```

# SQL

- Consider the following database schema :



# SQL

- ▶ With the following sample data:

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

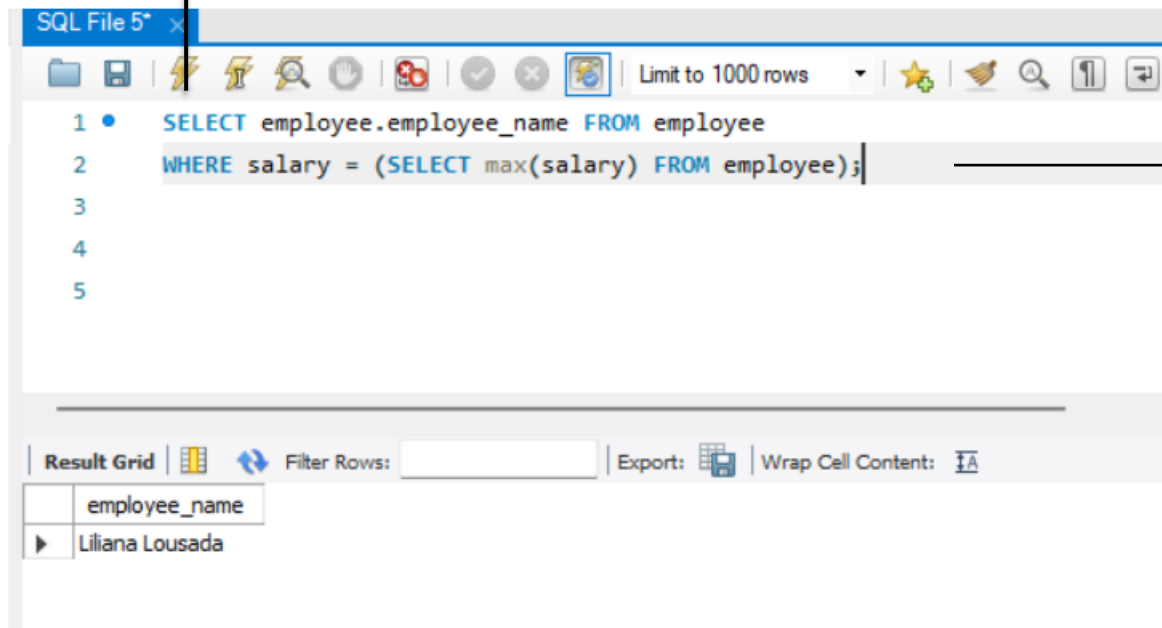
	employee_id	name	birth_date	salary	postalcode_id	admission_date	nif	department_id
	1	Manuel Santos	1972-01-01	1550	4000-100	2007-03-03	123456789	1
	2	Paulo Fosneca	1973-03-03	1820	4000-100	2008-01-01	234567890	1
	3	Carla Carolina	1974-04-04	1550	4100-050	2008-01-01	345678901	1
	4	Isabel Antunes	1975-05-05	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	Dioogo Dionísio	1981-11-11	2100	4460-100	2014-09-01	213456789	3
	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Result Grid	Filter Rows:	Edit:
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

Main Query



SubQuery

**Note:**

SubQuery runs first! The result enters as input in the main Query!

# SQL

SubQuery

UNCORRELATED QUERY

```
1 • SELECT employee.employee_id, employee.employee_name FROM employee
2 WHERE employee.department_id = (SELECT department_id FROM department WHERE department.name = "Production");
3
4
5
```

Result Grid

	employee_id	employee_name
▶	1	Manuel Santos
	2	Paulo Fosneca
	3	Carla Carolina
*	NULL	NULL

SubQuery must return only one value!

**Note:**

SubQuery runs first! The result enters as input in the main Query!



# SQL

## OPERATOR IN



Allows you to specify a  
list of values

The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The SQL editor contains the following query:

```
1 • SELECT employee.employee_id, employee.employee_name FROM employee
2 WHERE employee.department_id IN
3 (SELECT department.department_id FROM department WHERE department.name = "Production" OR department.name= "Sales");
4
5
6
```

The word 'IN' in the WHERE clause is highlighted with a red box. Below the editor is the 'Result Grid' toolbar with options like 'Filter Rows', 'Edit', 'Export/Import', and 'Wrap Cell Content'. The result grid displays the following data:

	employee_id	employee_name
▶	1	Manuel Santos
	2	Paulo Fosneca
	3	Carla Carolina
	6	Ricardo Rocha
	9	Liliana Lousada

### Note:

SubQuery runs first! The result enters as input in the main Query!

# SQL

When SubQuery returns (or can return) multiple rows

OPERATOR IN



Allows you to specify a list of values

The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The SQL editor contains the following query:

```
1 • SELECT employee.employee_id, employee.employee_name FROM employee
2 WHERE employee.department_id IN
3 (SELECT department.department_id FROM department WHERE department.name IN ("Production", "Sales", "Computing"));
4
5
6
```

Below the editor is the 'Result Grid' section, which includes a 'Filter Rows' input and buttons for 'Edit', 'Export/Import', and 'Wrap Cell Content'. The result grid displays the following data:

	employee_id	employee_name
▶	1	Manuel Santos
	2	Paulo Fosneca
	3	Carla Carolina
	4	Isabel Antunes
	10	Diogo Dionísio
	6	Ricardo Rocha
	9	Liliana Lousada
*	NULL	NULL

# SQL

## CORRELATED QUERY



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The SQL editor contains the following query:

```
1 • SELECT employee.employee_id, employee.employee_name FROM employee
2 WHERE employee.salary < (SELECT AVG(salary) FROM employee);
3
4
5
6
```

Below the editor is a 'Result Grid' section with a 'Filter Rows' input field and various action buttons (Edit, Export/Import, etc.). The result grid displays the following data:

	employee_id	employee_name
▶	1	Manuel Santos
	2	Paulo Foseca
	3	Carla Carolina
	7	José Silva
	8	Maria Andrade
*	NULL	NULL

# SQL

---

## ► UNCORRELATED QUERY

The inner SELECT does not depend on the external SELECT.

SELECT inner is executed first, and only once, and its result serves as input to the external SELECT.

## ► CORRELATED QUERY

The internal SELECT depends on the data provided by the external SELECT.

External SELECT is executed first, and the SELECT inside executes as many times as the ones executed from the outside.

`Correlated subqueries are used for row-by-row processing.`

`A correlated subquery is evaluated once for each row processed by the parent statement.`

`The parent statement can be a SELECT, UPDATE, or DELETE statement.`

# SQL

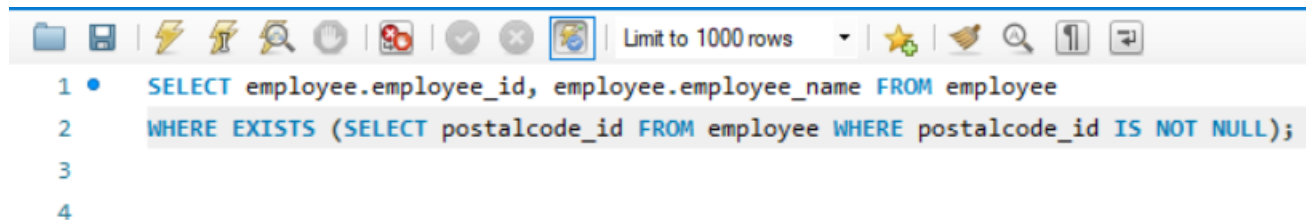
---

## OPERATOR EXISTS

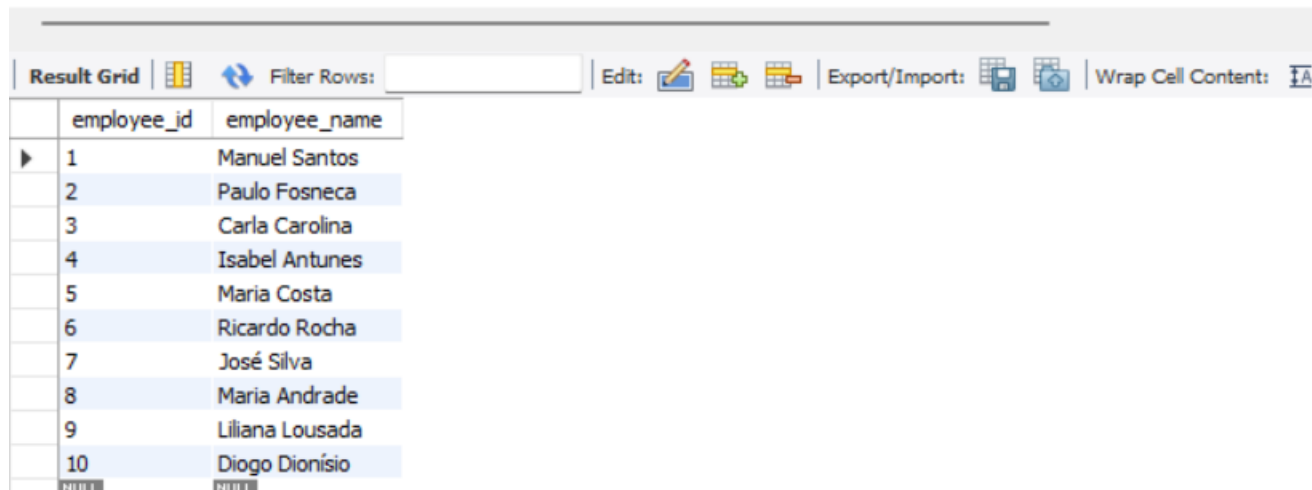
- ▶ Allows you to check whether the result of the SubQuery returns or not some results
- ▶ It is an operator that returns a logical value
- ▶ EXISTS operator is never preceded by any column or expression

# SQL

## OPERATOR EXISTS



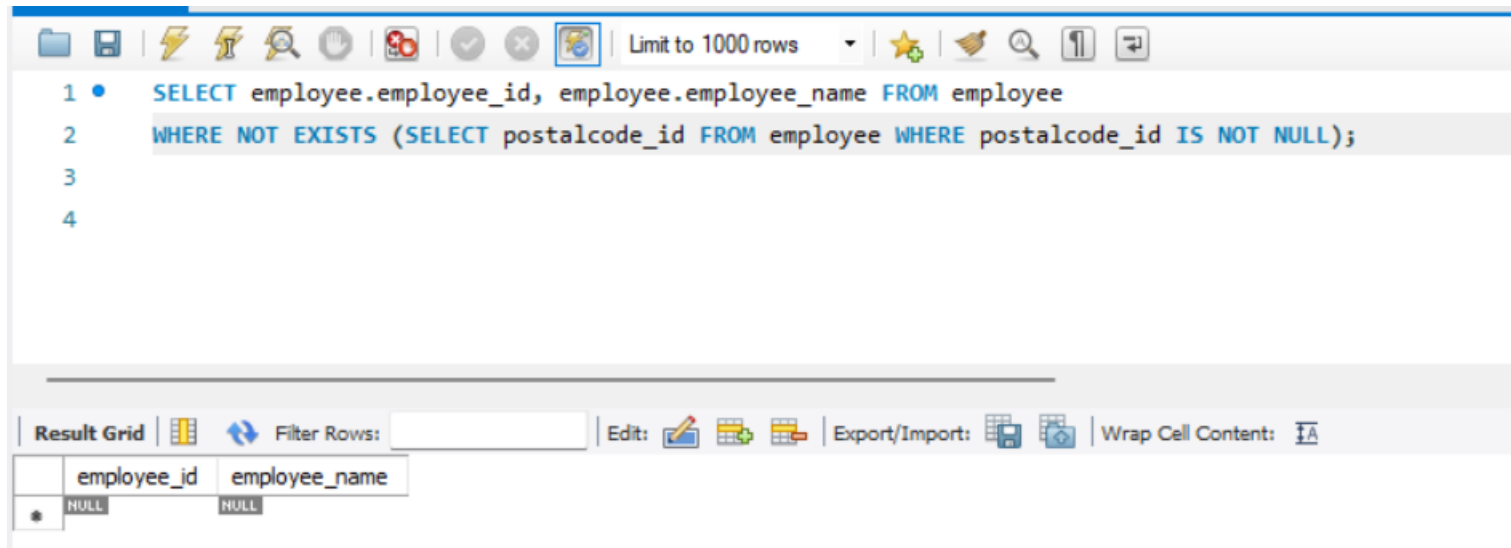
```
1 • SELECT employee.employee_id, employee.employee_name FROM employee
2   WHERE EXISTS (SELECT postalcode_id FROM employee WHERE postalcode_id IS NOT NULL);
3
4
```



	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 Lousada
	10	Diogo Dionísio

# SQL

OPERATOR EXISTS  
NOT EXISTS



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The SQL editor contains the following query:

```
1 • SELECT employee.employee_id, employee.employee_name FROM employee
2 WHERE NOT EXISTS (SELECT postalcode_id FROM employee WHERE postalcode_id IS NOT NULL);
3
4
```

Below the editor is the 'Result Grid' section. It includes a 'Filter Rows' input field and buttons for 'Edit', 'Export/Import', and 'Wrap Cell Content'. The result grid displays the following data:

	employee_id	employee_name
*	NULL	NULL

# SQL

## ► INSERT

Insert data into a database entity

INSERT INTO *table* VALUES (*values list*)

The screenshot displays a SQL IDE interface. The top panel shows two SQL statements: an INSERT statement and a SELECT statement. The bottom panel shows the result grid of the SELECT statement.

SQL Statements:

```
1 • INSERT INTO department VALUES(6, "Quality", 5500);
2
1 • SELECT * FROM company1.department;
```

Result Grid:

	department_id	name	budget
▶	1	Production	5000
	2	Accounting	3500
	3	Computing	7000
	4	Sales	2500
	5	Logistics	3000
	6	Quality	5500

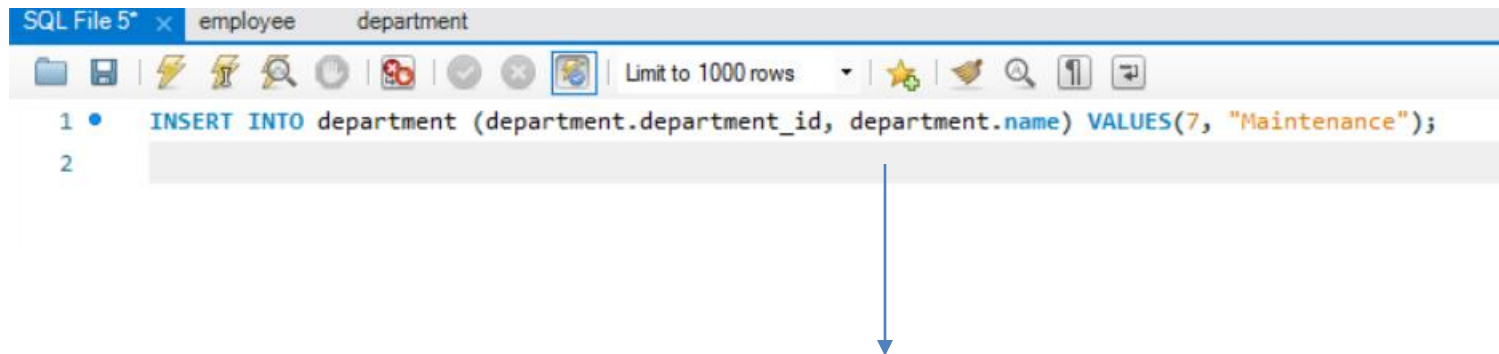


# SQL

## ► INSERT

Insert data into a database entity

```
INSERT INTO table [fields] VALUES (values list)
```



In this case, **if** budget field is required in the database:

```
Error Code: 1364. Field 'budget' doesn't have a default value
```

# SQL

## ► UPDATE

Edit data in a database entity

```
UPDATE table  
  SET field1 = value1,  
      field2 = value2  
[WHERE condition]
```

Attributes to edit in the database

If the WHERE clause does not exist, edit ALL the records of the entity !!!

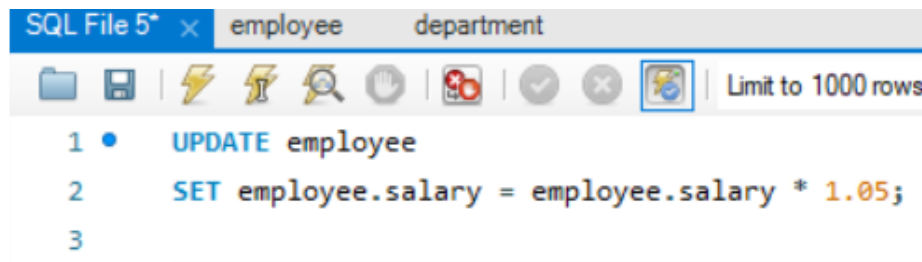


```
SQL File 5* x employee department  
1 UPDATE department  
2 SET department.budget= 6000  
3 WHERE department.department_id= 1;  
4
```

Result Grid			
Filter Rows: <input type="text"/>			
Edit:			
	department_id	name	budget
	1	Production	6000
	2	Accounting	3500
	3	Computing	7000
	4	Sales	2500
	5	Logistics	3000
	6	Quality	5500
	NULL	NULL	NULL

# SQL

## ► UPDATE



The screenshot shows a SQL IDE window with a tab labeled 'SQL File 5\*' and sub-tabs for 'employee' and 'department'. The toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' option. The SQL editor contains the following code:

```
1 • UPDATE employee
2   SET employee.salary = employee.salary * 1.05;
3
```

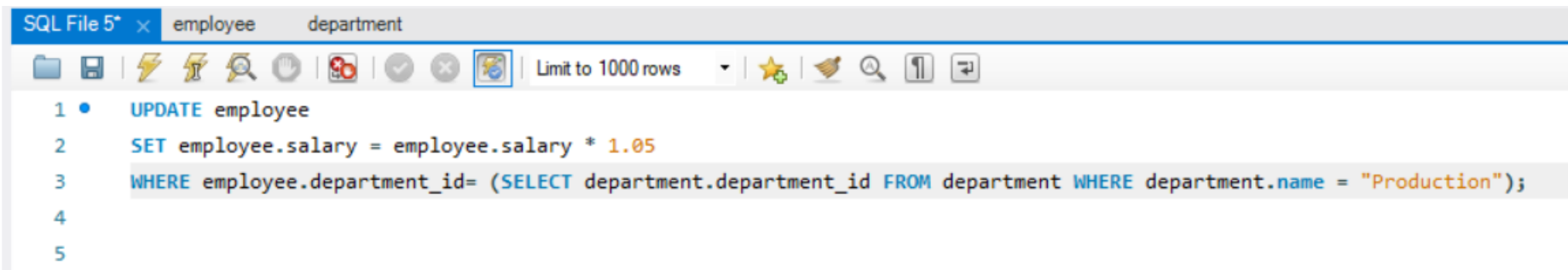


Example without WHERE, then edit ALL the records of the entity !!!

# SQL

## ► UPDATE

Update salary of employees from department *Production*



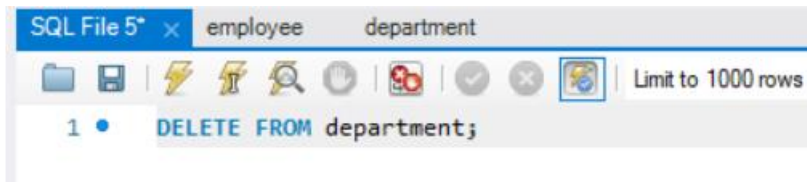
The screenshot shows a SQL IDE window with a tab labeled 'SQL File 5\*'. Below the tab bar, there are two database connections listed: 'employee' and 'department'. The main editor area displays an SQL query with line numbers 1 through 5 on the left. The query is: `UPDATE employee SET employee.salary = employee.salary * 1.05 WHERE employee.department_id= (SELECT department.department_id FROM department WHERE department.name = "Production");`. The IDE interface includes a toolbar with various icons for file operations, execution, and navigation. A dropdown menu shows 'Limit to 1000 rows'.

```
1 • UPDATE employee
2   SET employee.salary = employee.salary * 1.05
3   WHERE employee.department_id= (SELECT department.department_id FROM department WHERE department.name = "Production");
4
5
```

# SQL

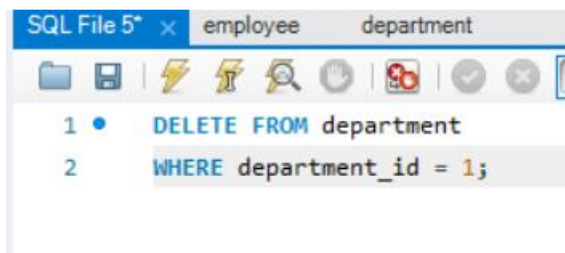
## ▶ DELETE

Remove data from an entity in the database



Delete all records  
from the entity

If the WHERE clause does not exist, remove ALL the records of the entity !!!



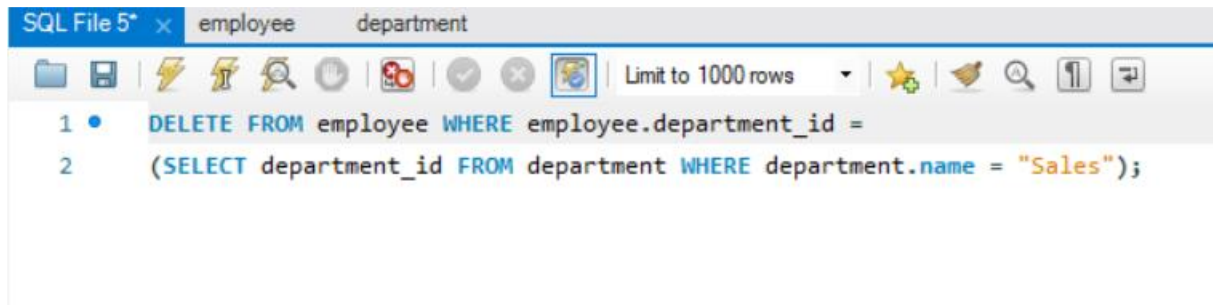
It removes the record related to department 1

# SQL

---

## ► DELETE

Remove data from an entity in the database



The screenshot shows a SQL IDE window with a tab labeled 'SQL File 5\*'. Below the tab bar, there are two database tables listed: 'employee' and 'department'. The main editor area contains a SQL query with two lines:

```
1 • DELETE FROM employee WHERE employee.department_id =  
2 (SELECT department_id FROM department WHERE department.name = "Sales");
```

The query is syntax-highlighted, with keywords in blue and string literals in orange. The IDE interface includes a toolbar with various icons for file operations, execution, and navigation, and a status bar at the bottom indicating 'Limit to 1000 rows'.

It removes all employees from department 'Sales'