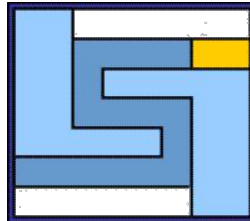




Escuela Técnica Superior de  
**Ingeniería Informática**



---

## Lab 04: SQL Instructions I

---

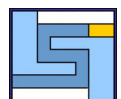
Introduction to Software Engineering and Information  
Systems I

Course 2021/22

David Ruiz, Inma Hernández, Agustín Borrego, Daniel Ayala

## Index

1 Objective .....	1
2 Environment setup. ....	1
3 INSERT .....	2
4 UPDATE .....	3
5 DELETE .....	4
6 SELECT .....	4
7 Views .....	6
8 Example queries .....	7
9 Tasks .....	9



## 1. Objective

The objective of this session is to handle the data stored in the database through the execution of SQL scripts. The student will learn to:

- Use the INSERT statement to introduce records.
- Use the UPDATE statement to edit records.
- Use the DELETE statement to remove records.
- Use the SELECT statement to query records.

So far we have created the tables that will support the data, but we have not inserted data in them, nor have we worked with them. In this practice we will use instructions to insert, alter, delete and query the rows of each table.

During the session the following relational model will be used:

### Modelo Entidad-Relación

Introducción a la Ingeniería del Software y Sistemas de Información I  
Proyecto Curso 2019-20

Grado: Degrees(degreeId (PK), name, duration)  
Espacio::Despacho: Offices(officeId (PK), name, floor, capacity)  
Espacio::Aula: Classrooms(classroomId (PK), name, floor, capacity, hasProjector, hasLoudspeakers)  
Departamento: Departments(departmentId (PK), name)  
Persona::Alumno: Students(studentId (PK), accessMethod, dni, firstName, surname, birthDate, email)  
Persona::Profesor: Professors(professorId (PK), officeId(FK), departmentId (FK), category, dni, firstName, surname, birthDate, email)  
Tutoría: TutoringHours(tutoringHoursId (PK), professorId(FK), dayOfWeek, startHour, endHour)  
Carga: TeachingLoads(teachingLoadId (PK), professorId(FK), groupId (FK), credits)  
Cita: Appointments(appointmentId (PK), tutoringHoursId (FK), studentId (FK), hour, date)  
Asignatura: Subjects(subjectId (PK), departmentId (FK), gradeId (FK), name, acronym, credits, year, type)  
Nota: Grades(gradeId (PK), studentId (FK), groupId (FK), value, call, withHonours)  
Grupo: Groups(groupId (PK), subjectId (FK), name, activity, academicYear)  
matriculadoEn: StudentsGroups(studentGroupId (PK), studentId (FK), groupId (FK))

Septiembre de 2019

## 2 Environment setup

Connect to the database and run the tables.sql file against the degrees database. Create a file queries.sql in which we will write the statements. Before each query execution, it is recommended to run the table creation script, so that the tables are in a controlled state before the retrieval or modification of any data.



### 3. INSERT

To insert rows into a table, we use the INSERT statement as follows:

```
14 INSERT INTO Degrees VALUES (NULL, 'Tecnologías Informáticas', 4);
```

Observe the following:

- The table name is indicated first, and then the row values separated by commas and between parentheses.

By default, the values must be written in the same order as the related columns within

- the table structure.

We give the ID a NULL value, so that it is given an automatic value with the pre-

- configured auto-increment. If given a number manually, that number would be used instead of the automatically generated one.
- When writing text strings, single quotes should be used. Although MariaDB technically allows the use of double quotes, they are not recommended since in similar languages and DBMS they cause errors.

Let's add some rows in the tables:

```
1 INSERT INTO Degrees (name, years) VALUES
2   ('Ingeniería del Software', 4),
3   ('Ingeniería del Computadores', 4),
4   ('Tecnologías Informáticas', 4);
5
6 INSERT INTO Subjects (name, acronym, credits, year, type, degreeId) VALUES
7   ('Fundamentos de Programación', 'FP', 12, 1, 'Formacion Basica', 3),
8   ('Lógica Informatica', 'LI', 6, 2, 'Optativa', 3);
9
10 INSERT INTO Groups (name, activity, year, subjectId) VALUES
11   ('T1', 'Teoria', 2019, 1),
12   ('L1', 'Laboratorio', 2019, 1),
13   ('L2', 'Laboratorio', 2019, 1);
14
15 INSERT INTO Students (accessMethod, dni, firstname, surname, birthdate, email) VALUES
16   ('Selectividad', '12345678A', 'Daniel', 'Pérez', '1991-01-01', 'daniel@alum.us.es'),
17   ('Selectividad', '22345678A', 'Rafael', 'Ramírez', '1992-01-01', 'rafael@alum.us.es'),
18   ('Selectividad', '32345678A', 'Gabriel', 'Hernández', '1993-01-01', 'gabriel@alum.us.es');
19
20 INSERT INTO GroupsStudents (groupId, studentId) VALUES
21   (1, 1),
22   (3, 1);
23
24 INSERT INTO Grades (value, gradeCall, withHonours, studentId, groupId) VALUES
25   (4.50, 1, 0, 1, 1);
```



Observe the following:

- The related column names were written before the assignment of the values. Columns not indicated are given a NULL value (or the default value if any). The order of the specified columns does not have to match the one given in the table creation, but it must be consistent with the values written afterwards.

Multiple records separated by commas can be inserted within the same INSERT

- statement.
- Dates are entered as strings, using the YYYY-MM-DD format.
- Booleans are entered as numeric values 0 or 1.

## 4. UPDATE

To modify one or more rows, we need to employ the UPDATE statement as follows:

```
42 UPDATE Students
43     SET birthdate='1998-01-01', surname='Fernández'
44     WHERE studentId=3;
```

Notice the following:

- Multiple attributes can be updated at the same time.
- The query has three parts: the affected table, the attributes to be modified, and a condition limiting the affected rows.
- If we omit the WHERE clause, all the rows will be updated. Watch out.

Updates can also use old values when giving new values, for example, the following divide by 2 the current value of the credits column for all the subjects:

```
46 UPDATE Subjects
47     SET credits = credits/2;
```



## 5. DELETE

To delete rows from a table, use the DELETE FROM statement. We can delete rows as follows:

```
49 DELETE FROM Grades
50 WHERE gradeId = 1;
```

Observe the following:

- The query is divided into two parts: the affected table, and a condition constraining the deleted rows.

- Watch out! If we omit the WHERE clause, all the rows in the table will be deleted.

By default (i.e., when used the RESTRICT/NO ACTION strategy when creating the foreign key), you cannot delete a row that is referenced by another using a foreign

- key. We would first have to remove the record that references the one that we want to remove.

If we want the records with a reference to the record we want to delete to be removed/update automatically, instead of producing an error, we have to set the cascade strategy to the deletion clause of the foreign key through setting ON DELETE CASCADE, as explained in the previous session. Note that, in any case, the referential integrity of the database is maintained, preventing references to rows that do not exist.

## 6. SELECT

Querying data from a database is one of the essential operations. The result of this type of query is always a table with rows and columns determined by the query.

For example, we could select the names and surnames of “Selectividad” access students as follows:

```
55 SELECT firstname, surname
56 FROM Students
57 WHERE accessMethod = 'Selectividad';
```

students (2x3)	
firstname	surname
Daniel	Pérez
Rafael	Ramírez
Gabriel	Fernández



Observe the following:

- The query has three parts: the columns to be obtained, the tables involved (there can be several), and a condition (optional) constraining the selected rows.
- If we want to get all the columns present in the selected table, we can use \*: SELECT \* FROM ....


The selected columns do not need to exist in the tables. Derived computations can be defined as new columns to be retrieved. For instance:

```
59 SELECT credits > 3
60 FROM Subjects;
```

In that case, the returned value will be a boolean, indicating, for each row, whether the number of credits is greater than 3.

We can also compute aggregate values (sums, averages, etc.). Querying these values implies that only one row will be returned. If also only one column is requested, a unique value will be returned:

```
62 SELECT AVG(credits)
63 FROM Subjects;
```

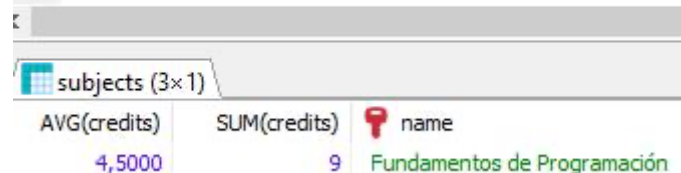


Resultado #1 (1x1)

AVG(credits)
4,5000

Notice what happens when aggregated values are requested as columns along with a column from the table:

```
65 SELECT AVG(credits), SUM(credits), name
66 FROM Subjects;
```



subjects (3x1)


AVG(credits)	SUM(credits)	name
4,5000	9	Fundamentos de Programación



When requesting aggregated values, only one row is returned, which corresponds to the value computed using all the data present in the table. The returned name is simply the name of the first of the rows.

One of the most useful aggregate values is COUNT. In its most common variant, it counts the number of rows returned:

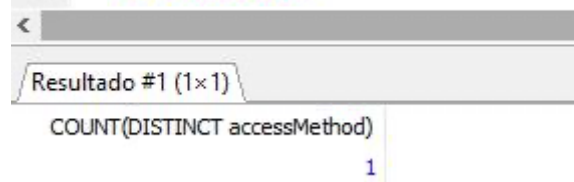
```
65 SELECT COUNT(*)
66 FROM Subjects
67 WHERE credits > 4;
```



The screenshot shows a SQL query result in a window titled 'Resultado #1 (1x1)'. The query is: `SELECT COUNT(*) FROM Subjects WHERE credits > 4;`. The result is a single row with the value 1.

However, we can include an expression that limits the rows that to be counted:

```
70 SELECT COUNT(DISTINCT accessMethod)
71 FROM Students;
```



The screenshot shows a SQL query result in a window titled 'Resultado #1 (1x1)'. The query is: `SELECT COUNT(DISTINCT accessMethod) FROM Students;`. The result is a single row with the value 1.

## 7. Views

Sometimes it is useful to save the results of a query and then use them in other queries as if they were just another table. This greatly simplifies the creation of complex nested queries. This can be done through views, in which we name a SELECT query that can then be used as if it were a table.

For example, we could create a view that contains the notes of the group with ID 18:

```
123 CREATE OR REPLACE VIEW ViewGradesGroup18 AS
124 SELECT * FROM Grades WHERE groupId = 18;
```

And then use it in different queries:





```
126 SELECT MAX(value) FROM ViewGradesGroup18;
127 SELECT COUNT(*) FROM ViewGradesGroup18;
128 SELECT * FROM ViewGradesGroup18 WHERE gradeCall = 2;
```

We can also use one view within another:

```
130 CREATE OR REPLACE VIEW ViewGradesGroup18Call1 AS
131     SELECT * FROM ViewGradesGroup18 WHERE gradeCall = 1;
132
133 SELECT * FROM ViewGradesGroup18Call1;
```

## 8. Example queries

The possibilities when composing query are vast. In this section we'll run a set of statements covering many of the possible cases. Before running the proposed queries, execute the populate.sql file, which inserts a larger amount of data into the tables.

- All the asubjects.

```
SELECT * FROM Subjects;
```

- Subjects with acronym 'FP'.

```
SELECT * FROM Subjects WHERE acronym='FP';
```

- Names and acronyms of all subjects.

```
SELECT name, acronym FROM Subjects;
```

- Average of the grades of the group with ID 18.

```
SELECT AVG(value) FROM Grades WHERE groupId=18;
```

- Total credits for the subjects of the Computer Technologies degree (ID 3).

```
SELECT SUM(credits) FROM Subjects WHERE degreeId=3;
```

- Debrees with a value less than 4 or greater than 6.



```
SELECT * FROM Grades WHERE value < 4 OR value > 6;
```

- Names of different groups.

```
SELECT DISTINCT name FROM Groups;
```

- Maximum grade of the student with ID 1.

```
SELECT MAX(VALUE) FROM Grades WHERE studentId=1;
```

- Students with a surname equal to the acronym of some subject.

```
SELECT * FROM Students WHERE surname IN (SELECT acronym FROM Subjects);
```

- Student IDs for the 2019 course.

```
SELECT DISTINCT(StudentId)
FROM GroupsStudents
WHERE groupId IN (SELECT groupId FROM Groups WHERE year = 2019);
```

- Students with a DNI (ID number) with the character C as the last character. Notice how % represents any number of previous characters.

```
SELECT *
FROM Students
WHERE dni LIKE('%C')
```

- Students with a 6-letter name. Notice how \_ represents any character.

```
SELECT *
FROM Students
WHERE firstName LIKE('_____') -- 6 guiones bajos
```

- Students born before 1995.

```
SELECT *
FROM Students
WHERE YEAR(birthdate) < 1995
```

- Students born between January and February.

```
SELECT *
FROM Students
WHERE (MONTH(birthdate) >= 1 AND MONTH(birthdate) <= 2)
```

Notice how in the last two queries it is possible to include another query within the condition of the first, instead of using hand-typed values.



## 9. Tasks

To the previous queries, add others that get the following:

- Name of the subjects that are mandatory.
- Average of the grades of the group with ID 19, using the AVG aggregator.
- The same query above, without using AVG.
- Number of different group names.
- Notes between 4 and 6, inclusive.
- Notes with a value equal to or greater than 9, but which are not honors. Create a view for your honors degree notes.

Create an SQL script that generates the tutoring management system tables and inserts, modifies, deletes, and queries them. It must include:

- At least three rows inserted in each table.
- Update multiple attributes for a specific row.
- Update a single attribute on multiple rows at the same time.
- Deletion of a specific row.
- Deletion of several rows at the same time (without removing all the rows from the table).

In order to complete this task, create a copy of the repository for this session with your GitHub user. Make the appropriate modifications to the supplied files, creating new files if necessary, and upload those changes to your copy on GitHub. Remember to set the privacy of your repository as "Private" and add dfernandez6 as a collaborator.

