

UD 5

TRATAMIENTO DE DATOS

Continuamos estudiando el **DML** de **SQL** para aprender el uso de las sentencias:

- **INSERT**: para insertar filas en tablas
- **DELETE**: para eliminar filas de tablas
- **UPDATE**: para modificar columnas de filas en tablas

Base de datos de prueba

Vamos a crear una **BDD** llamada **centro_educativo** para probar algunas sentencias DML de inserción, modificación y borrado de filas de tablas



The screenshot shows the MySQL Workbench interface. In the top query editor window, titled "Query 1", the command `1. create database centro_educativo;` is entered. Below it, the "Action Output" pane displays a table with one row of data:

#	Time	Action	Message
✓	1	03:15:33	create database centro_educativo 1 row(s) affected

1. Inserción de registros

La sentencia que se usa para insertar filas en una tabla es la sentencia **INSERT**

Vamos a crear la tabla **Asignatura** para poder practicar las distintas opciones de la sentencia **INSERT**

Query 1 ✖

The screenshot shows the MySQL Workbench interface. At the top is a toolbar with various icons. Below it is a query editor window titled "Query 1" containing the following SQL code:

```
1 • use centro_educativo;
2
3 • create table Asignatura (
4     CodAsig      numeric(3),
5     CodCF       numeric(3) not null,
6     NumHoras    numeric(3),
7     Nombre       varchar(50) not null,
8     primary key (CodAsig),
9     constraint ck_asig check(NumHoras between 10 and 900)
10 );
```

Below the query editor is an "Action Output" table with the following data:

Action	#	Time	Message
use centro_educativo	1	03:19:20	0 row(s) affected
create table Asignatura (CodAsig numeric(3), ...	2	03:24:01	0 row(s) affected

Ejemplo 1: Introducimos una fila en la tabla Asignatura

Query 1 ✖

```
1 • use centro_educativo;
2
3 • insert into Asignatura values (1, 1, 165, 'Bases de datos');
```

Action Output
Time Action Message
1 12:15:28 insert into Asignatura values (1, 1, 165, 'Bases de datos') 1 row(s) affected

Vemos que se insertan los datos en cada columna, en el orden en el que se crearon las columnas de la tabla:

1. **CodAsig** → **1**
2. **CodCF** → **1**
3. **NumHoras** → **165**
4. **Nombre** → **Bases de datos**

Los datos no numéricos se introducen entrecomillados (**'Bases de datos'**)

Query 1 ✖

```
1 • use centro_educativo;
2
3 • select *
4     from Asignatura;
```

Result Grid				
#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	165	Bases de datos
*	NULL	NULL	NULL	NULL

Ejemplo 2: Introducimos otra fila en la tabla **Asignatura** pero ahora **escribimos explícitamente** (en orden de creación) **las columnas de la tabla**

The screenshot shows the MySQL Workbench interface. In the top panel, there is a toolbar with various icons and a dropdown menu set to "Limit to 1000 rows". Below the toolbar, a query editor window titled "Query 1" contains the following SQL code:

```
1 • use centro_educativo;
2
3 • insert into Asignatura (CodAsig, CodCF, NumHoras, Nombre)
4     values (2, 1, 120, 'Lenguaje de marcas');
```

In the bottom panel, there is an "Action Output" section with a table showing the results of the last query:

#	Time	Action	Message
1	12:40:48	insert into Asignatura (CodAsig, CodCF, NumHoras, Nom...	1 row(s) affected

Vemos que se inserta la fila:

The screenshot shows the MySQL Workbench interface. In the top panel, there is a toolbar with various icons and a dropdown menu set to "Limit to". Below the toolbar, a query editor window titled "Query 1" contains the following SQL code:

```
1 • use centro_educativo;
2
3 • select *
4     from Asignatura;
5
```

In the bottom panel, there is a "Result Grid" showing the data from the "Asignatura" table:

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	165	Bases de datos
2	2	1	120	Lenguaje de marcas
*	NULL	NULL	NULL	NULL

Ejemplo 3: Introducimos otra fila en la tabla **Asignatura** pero ahora **escribimos desordenadas las columnas de la tabla**

Query 1 X

```
1 • use centro_educativo;
2
3 • insert into Asignatura (CodAsig, Nombre, CodCF, NumHoras)
4     values (3, 'Seguridad informática', 2, 90);
5
```

Action Output ▼

#	Time	Action	Message
✓ 1	12:49:08	insert into Asignatura (CodAsig, Nombre, CodCF, NumH... 1 row(s) affected	

Vemos que se inserta la fila:

Query 1 X

```
1 • use centro_educativo;
2
3 • select *
4     from Asignatura;
5
```

Result Grid ▼ Filter Rows: A Edit: Pencil

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	165	Bases de datos
2	2	1	120	Lenguaje de marcas
3	3	2	90	Seguridad informática
*	NULL	NULL	NULL	NULL

Ejemplo 4: Como la columna **NumHoras** puede ser **nula**, podemos omitir el nombre de esa columna en la sentencia INSERT

The screenshot shows the MySQL Workbench interface. The top section is a query editor titled "Query 1" containing the following SQL code:

```
1 • use centro_educativo;
2
3 • insert into Asignatura (CodAsig, CodCF, Nombre)
4     values (4, 1, 'Despliegue aplicaciones web');
5
```

The bottom section is the "Action Output" pane, which displays a log entry:

#	Time	Action	Message
1	13:01:10	insert into Asignatura (CodAsig, CodCF, Nombre) values ...	1 row(s) affected

Vemos que se inserta la fila con la columna **NumHoras** a **NULL**:

The screenshot shows the MySQL Workbench interface. The top section is a query editor titled "Query 1" containing the following SQL code:

```
1 • use centro_educativo;
2
3 • select *
4     from Asignatura;
5
```

The bottom section is the "Result Grid" pane, displaying the following data:

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	165	Bases de datos
2	2	1	120	Lenguaje de marcas
3	3	2	90	Seguridad informática
4	4	1	NULL	Despliegue aplicaciones web
*	NULL	NULL	NULL	NULL

Ejemplo 5: Como la columna **NumHoras** puede ser **nula**, podemos usar el valor **NULL** para dar ese valor a dicha columna

Query 1 x

```
1 • use centro_educativo;
2
3 • insert into Asignatura values (5, 2, NULL, 'Fundamentos de hardware');|
```

Action Output

#	Time	Action	Message
1	13:12:15	insert into Asignatura values (5, 2, NULL, 'Fundamentos ...	1 row(s) affected

Vemos que se inserta la fila con la columna **NumHoras** a **NULL**:

Query 1 x

```
1 • use centro_educativo;
2
3 • select *
4     from Asignatura;|
```

Result Grid Filter Rows: Edit: Export

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	165	Bases de datos
2	2	1	120	Lenguaje de marcas
3	3	2	90	Seguridad informática
4	4	1	NULL	Despliegue aplicaciones web
5	5	2	NULL	Fundamentos de hardware
*	NULL	NULL	NULL	NULL

Ejemplo 6: Como la columna **Planta** de la tabla **Departamento** tiene el valor **1 por defecto** omitir el valor al insertar una fila

Query 1 X

```
1 • use centro_educativo;
2
3 • create table Departamento (
4     CodDep      numeric(2),
5     Nombre      varchar(40) not null,
6     Planta      numeric(2) default 1,
7     DNI         varchar(9) not null,
8     primary key (CodDep)
9 );
```

Action Output ▼

#	Time	Action	Message
✓ 1	02:20:11	create table Departamento (CodDep numeric...)	0 row(s) affected

Insertamos una fila sin dar valor a la columna **Planta**

Query 1 X

```
1 • use centro_educativo;
2
3 • insert into Departamento (CodDep, Nombre, DNI)
4   values (1, 'Informática y Comunicaciones', '48103100');
5
```

Action Output ▼

#	Time	Action	Message
✓ 1	02:21:22	insert into Departamento (CodDep, Nombre, DNI) values...	1 row(s) affected

Vemos que la columna **Planta** de la fila insertada toma el valor por defecto 1:

Query 1 X

```
1 • use centro_educativo;
2
3 • select *
4   from Departamento
5   where CodDep = 1;
```

Result Grid ▼ Filter Rows: A Edit: Pencil New Delete Export

#	CodDep	Nombre	Planta	DNI
1	1	Informática y Comunicaciones	1	48103100

Ejercicio 1: Inserta en la tabla Asignatura una fila con los siguientes valores

1. **CodAsig** → **6**
2. **CodCF** → **1**
3. **NumHoras** → **180**
4. **Nombre** → **Acceso a datos**

Solución: La solución más simple sería

Query 1 ×

The screenshot shows the MySQL Workbench interface. At the top is a toolbar with various icons. Below it is a query editor window titled "Query 1" containing the following SQL code:

```
1 • use centro_educativo;
2
3 • insert into Asignatura values (6, 1, 180, 'Acceso a datos');
4
```

Below the query editor is an "Action Output" panel with a table showing the results of the last query:

#	Time	Action	Message
1	13:24:57	insert into Asignatura values (6, 1, 180, 'Acceso a datos')	1 row(s) affected

Vemos que se inserta la fila:

Query 1 ×

The screenshot shows the MySQL Workbench interface. At the top is a toolbar with various icons. Below it is a query editor window titled "Query 1" containing the following SQL code:

```
1 • use centro_educativo;
2
3 • select *
4      from Asignatura;
```

Below the query editor is a "Result Grid" table showing the data from the "Asignatura" table:

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	165	Bases de datos
2	2	1	120	Lenguaje de marcas
3	3	2	90	Seguridad informática
4	4	1	NULL	Despliegue aplicaciones web
5	5	2	NULL	Fundamentos de hardware
6	6	1	180	Acceso a datos

Ejemplo 7: Se pueden insertar varias filas con una única sentencia INSERT

Eliminamos primero todas las filas de la tabla Asignatura para volver a insertar filas en ella

Query 1 ×

```
1 • use centro_educativo;
2
3 • truncate table Asignatura;
```

Action Output ▼

#	Time	Action
1	19:44:18	truncate table Asignatura

Vemos que la tabla queda vacía:

Query 1 ×

```
1 • use centro_educativo;
2
3 • select *
4     from Asignatura;
```

Result Grid ▼ Filter Rows: Ⓐ

#	CodAsig	CodCF	NumHoras	Nombre
*	NULL	NULL	NULL	NULL

Insertamos tres filas con un solo **insert**:

Query 1 ×

1 • `use centro_educativo;`
2
3 • `insert into Asignatura values`
4 `(1, 1, 165, 'Bases de datos'),`
5 `(2, 1, 120, 'Lenguaje de marcas'),`
6 `(3, 2, 90, 'Seguridad informática');`

Action Output ▼

#	Time	Action
1	19:58:00	insert into Asignatura values (1, 1, 165, 'Bases de datos')...

Observamos que:

1. Cada fila aparece entre **paréntesis**
2. Las filas se separan entre ellas por una **coma**
3. La última fila va seguida de un **punto y coma**

Vemos que se han insertado las tres filas:

Query 1 ×

1 • `use centro_educativo;`
2
3 • `select *`
4 `from Asignatura;`

Result Grid ▼ Filter Rows: A Edit: P E D Export

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	165	Bases de datos
2	2	1	120	Lenguaje de marcas
3	3	2	90	Seguridad informática
*	NULL	NULL	NULL	NULL

Ejercicio 2: Inserta en una única sentencia las tres filas que faltan para tener todas las asignaturas en la tabla

Query 1 ×

use centro_educativo;

select * from Asignatura;

Result Grid Filter Rows: A Edit: Edit Insert Delete Update Export

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	165	Bases de datos
2	2	1	120	Lenguaje de marcas
3	3	2	90	Seguridad informática
4	4	1	110	Despliegue aplicaciones web
5	5	2	90	Fundamentos de hardware
6	6	1	180	Acceso a datos
*	NULL	NULL	NULL	NULL

Solución: Una solución (no es la única) podría ser

Query 1 ×

use centro_educativo;

insert into Asignatura values (4, 1, 110, 'Despliegue aplicaciones web'),
(5, 2, 90, 'Fundamentos de hardware'),
(6, 1, 180, 'Acceso a datos');

Action Output ▼

#	Time	Action
1	20:27:47	insert into Asignatura values (4, 1, 110, 'Despliegue aplic...')

Campos automuméricos

Es muy habitual que los campos clave primaria sigan una **secuencia autonumérica**.

Sin embargo SQL no define el tratamiento de dichos valores autonuméricos, por lo cual cada SGBD relacional utiliza un mecanismo diferente.

Por ejemplo:

1. **Oracle** utiliza unos objetos llamados **secuencias** (SEQUENCE) que se definen independientemente de las tablas
2. **SQL Server** utiliza **IDENTITY**
3. **MySQL** utiliza **AUTO_INCREMENT** en la definición de una columna de una tabla

Ejemplo 1: creamos (en MySQL) la tabla **Ciclo** con la columna **CodCF** definida como **AUTO_INCREMENT**

The screenshot shows the MySQL Workbench interface. The query editor window is titled "Query 1". It contains the following SQL code:

```
1 • use centro_educativo;
2
3 • create table Ciclo (
4     CodCF    int not null auto_increment primary key,
5     Nombre   varchar(255) not null,
6     Siglas   varchar(4) not null,
7     Tipo     varchar(1)  not null
8 );
```

Below the query editor is an "Action Output" table showing the execution results:

#	Time	Action	Message
1	14:30:03	create table Ciclo (CodCF int not null auto_increment...	0 row(s) affected

NOTA: la columna **CodCF** tiene que ser de tipo **int** (número entero) para que se le pueda asignar **AUTO_INCREMENT**

Podemos ver con **desc** que se ha creado la columna autoincremental

Query 1 x

```
1 • use centro_educativo;
2
3 • desc Ciclo;
```

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

#	Field	Type	Null	Key	Default	Extra
1	CodCF	int	NO	PRI	NULL	auto_increment
2	Nombre	varchar(255)	NO		NULL	
3	Siglas	varchar(4)	NO		NULL	
4	Tipo	varchar(1)	NO		NULL	

Insertamos tres filas sin referirnos a la columna **CodCF** porque como es autoincremental tomará un valor aunque nosotros no se lo demos

Query 1 x

```
1 • use centro_educativo;
2
3 • insert into Ciclo (Nombre, Siglas, Tipo) values
4 ('Desarrollo de aplicaciones web', 'DAW', 'S'),
5 ('Administración de sistemas informáticos en red', 'ASIR', 'S'),
6 ('Desarrollo de aplicaciones multiplataforma', 'DAM', 'S');
```

Action Output ▼

#	Time	Action	Message
1	14:33:35	insert into Ciclo (Nombre, Siglas, Tipo) values ('Desarrol...')	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0

Vemos que se insertan las tres filas y observamos los valores consecutivos de la columna **CodCF**:

Query 1 x

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it is a query editor window containing the following SQL code:

```
1 • use centro_educativo;
2
3 • select *
4     from Ciclo;
```

Below the query editor is a "Result Grid" window. It has a header row with columns: #, CodCF, Nombre, Siglas, and Tipo. The data is as follows:

#	CodCF	Nombre	Siglas	Tipo
1	1	Desarrollo de aplicaciones web	DAW	S
2	2	Administración de sistemas informá...	ASIR	S
3	3	Desarrollo de aplicaciones multiplat...	DAM	S
*	NULL	NULL	NULL	NULL

Ejemplo 2: Veamos que si le **asigno manualmente unos valores** concretos a la columna **CodCF** entonces **no se aplica el autoincremento**

Eliminamos primero todas las filas de la tabla **Ciclo** para hacer la prueba

Query 1 ×

```
1 • use centro_educativo;
2
3 • truncate table Ciclo;
```

Action Output ▾

#	Time	Action
1	14:53:09	truncate table Ciclo

Vemos que la tabla queda vacía:

Query 1 ×

```
1 • use centro_educativo;
2
3 • select *
   from Ciclo;
```

Result Grid [] Filter Rows: A

#	CodCF	Nombre	Siglas	Tipo
*	NULL	NULL	NULL	NULL

Insertamos ahora dos filas indicando explícitamente el valor de **CodCF** (11 para la primera fila y 12 para la segunda fila):

The screenshot shows the MySQL Workbench interface. In the top query editor, the following SQL code is written:

```
1 • use centro_educativo;
2
3 • insert into Ciclo (CodCF, Nombre, Siglas, Tipo) values
4     (11, 'Desarrollo de aplicaciones web', 'DAW', 'S'),
5     (22, 'Administración de sistemas informáticos en red', 'ASIR', 'S');
```

In the bottom "Action Output" section, there is a table with the following data:

#	Time	Action	Message
1	14:56:51	insert into Ciclo (CodCF, Nombre, Siglas, Tipo) values (11... 2 row(s) affected	Records: 2 Duplicates: 0 Warnings: 0

Observamos que se insertan esos valores (11 y 22) en lugar de los autoincrementales (1 y 2):

The screenshot shows the MySQL Workbench interface. In the top query editor, the following SQL code is written:

```
1 • use centro_educativo;
2
3 • select *
4     from Ciclo;
```

In the bottom "Result Grid" section, the data is displayed as follows:

#	CodCF	Nombre	Siglas	Tipo
1	22	Administración de sistemas informáticos en red	ASIR	S
2	11	Desarrollo de aplicaciones web	DAW	S
*	NULL	NULL	NULL	NULL

Si insertamos una nueva fila sin explicitar CodCF usará el valor autoincremental:

The screenshot shows the MySQL Workbench interface. In the top query editor, the following SQL code is written:

```
1 • use centro_educativo;
2
3 • insert into Ciclo (Nombre, Siglas, Tipo) values
4     ('Desarrollo de aplicaciones multiplataforma', 'DAM', 'S');
```

In the bottom "Action Output" section, there is a table with the following data:

#	Time	Action	Message
1	15:02:57	insert into Ciclo (Nombre, Siglas, Tipo) values ('Desarroll... 1 row(s) affected	

Como podemos comprobar la nueva fila dará el valor **23** a la columna **CodCF**:

Query 1 ×

```
1 • use centro_educativo;
2
3 • select *
4     from Ciclo;
```

Result Grid Filter Rows: Edit: A Export

#	CodCF	Nombre	Siglas	Tipo
1	11	Desarrollo de aplicaciones web	DAW	S
2	22	Administración de sistemas informáticos en red	ASIR	S
3	23	Desarrollo de aplicaciones multiplataforma	DAM	S
*	NULL	NULL	NULL	NULL

Nota: si necesitamos **resetear** el autoincremento de una tabla lo podemos hacer con **alter table**

Ejemplo: si truncamos la tabla Ciclo nos interesa que el autoincremento se reinicie a 1

Query 1 × Administration - Status and System Variables ×

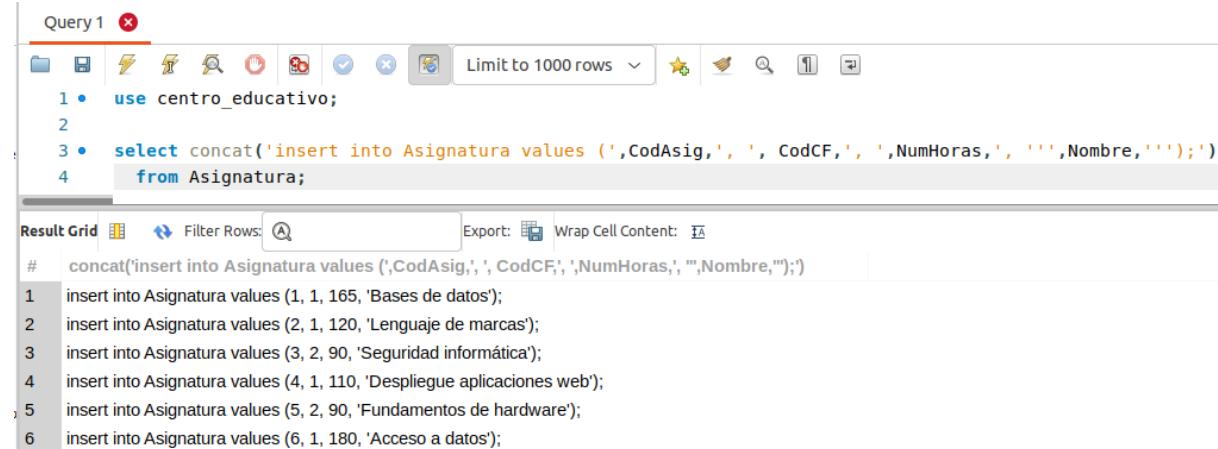
```
1 • use centro_educativo;
2
3 • truncate table Ciclo;
4
5 • alter table Ciclo auto_increment = 1;
6
```

Action Output ▼

#	Time	Action	Message
1	14:56:05	truncate table Ciclo	0 row(s) affected
2	14:56:09	alter table Ciclo auto_increment = 1	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Crear sentencias insert a partir de consultas

Ejemplo 1: si queremos copiar la filas de la tabla Asignatura en otra base de datos podemos usar la sentencia SELECT con la función concat (para concatenar cadenas):



The screenshot shows the MySQL Workbench interface. In the Query Editor (Query 1), the following SQL code is written:

```
1 • use centro_educativo;
2
3 • select concat('insert into Asignatura values (' ,CodAsig,' , ', CodCF,' , ',NumHoras,' , ''',Nombre,'''');'
4     from Asignatura;
```

The Result Grid displays the output of the query, which is a list of six insert statements for the Asignatura table:

```
#   concat('insert into Asignatura values (' ,CodAsig,' , ', CodCF,' , ',NumHoras,' , ''',Nombre,'''');'
1 insert into Asignatura values (1, 1, 165, 'Bases de datos');
2 insert into Asignatura values (2, 1, 120, 'Lenguaje de marcas');
3 insert into Asignatura values (3, 2, 90, 'Seguridad informática');
4 insert into Asignatura values (4, 1, 110, 'Despliegue aplicaciones web');
5 insert into Asignatura values (5, 2, 90, 'Fundamentos de hardware');
6 insert into Asignatura values (6, 1, 180, 'Acceso a datos');
```

NOTA: es necesario usar tres comillas simples consecutivas para obtener una

Podríamos usarlo para generar un fichero con todos los inserts:

```
mysql>
mysql> select database();
+-----+
| database()      |
+-----+
| centro_educativo |
+-----+
1 row in set (0,00 sec)

mysql> select concat('insert into Asignatura values (' ,CodAsig,' , ', CodCF,' , ',NumHoras,' , ''',Nombre,'''');'
->     from Asignatura
->     into outfile '/var/lib/mysql-files/varios_inserts_Asignatura.sql';
Query OK, 6 rows affected (0,00 sec)
```

Comprobamos que se ha creado el fichero correctamente:

```
profe@profe-VirtualBox:~/Desktop$ 
profe@profe-VirtualBox:~/Desktop$ sudo more /var/lib/mysql-files/varios_inserts_Asignatura.sql
insert into Asignatura values (1, 1, 165, 'Bases de datos');
insert into Asignatura values (2, 1, 120, 'Lenguaje de marcas');
insert into Asignatura values (3, 2, 90, 'Seguridad informática');
insert into Asignatura values (4, 1, 110, 'Despliegue aplicaciones web');
insert into Asignatura values (5, 2, 90, 'Fundamentos de hardware');
insert into Asignatura values (6, 1, 180, 'Acceso a datos');
profe@profe-VirtualBox:~/Desktop$
```

Ejemplo 2: podemos conseguirlo también con una sola sentencia insert, por ejemplo así

The screenshot shows the MySQL Workbench interface. The query editor contains the following SQL code:

```
1 • select * from (
2   select 'insert into Asignatura values'
3   union
4   select if (CodAsig= (select max(CodAsig) from Asignatura),
5             concat('(',CodAsig,', ', CodCF,', ', NumHoras,', ''', Nombre, ''');'),
6             concat('(',CodAsig,', ', CodCF,', ', NumHoras,', ''', Nombre, '''),'))
7   from Asignatura) as resultado
8   order by 1;
```

The result grid below the editor shows the output of the query:

insert into Asignatura values
insert into Asignatura values
(1, 1, 165, 'Bases de datos'),
(2, 1, 120, 'Lenguaje de marcas'),
(3, 2, 90, 'Seguridad informática'),
(4, 1, 110, 'Despliegue aplicaciones web'),
(5, 2, 90, 'Fundamentos de hardware'),
(6, 1, 180, 'Acceso a datos');

Podríamos usarlo para generar un fichero con el insert:

```
mysql>
mysql> select * from (select 'insert into Asignatura values' union select if (CodAsig= (select max(CodAsig) from Asignatura), concat('(',CodAsig,', ', CodCF,', ', NumHoras,', ''', Nombre, ''');'), concat('(',CodAsig,', ', CodCF,', ', NumHoras,', ''', Nombre, '''),')) from Asignatura) as resultado order by 1 into outfile 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/insert_unico_Asignatura.sql';
Query OK, 7 rows affected (0.00 sec)

mysql>
```

Comprobamos que se ha creado el fichero correctamente:

```
C:\ProgramData\MySQL\MySQL Server 8.0\Uploads>type varios_inserts_Asignatura.sql
insert into Asignatura values(1, 1, 165, 'Bases de datos');
insert into Asignatura values(2, 1, 120, 'Lenguaje de marcas');
insert into Asignatura values(3, 2, 90, 'Seguridad informática');
insert into Asignatura values(4, 1, 110, 'Despliegue aplicaciones web');
insert into Asignatura values(5, 2, 90, 'Fundamentos de hardware');
insert into Asignatura values(6, 1, 180, 'Acceso a datos');

C:\ProgramData\MySQL\MySQL Server 8.0\Uploads>
```

2. Borrado de registros. Modificación de registros

Borrado de registros

La sentencia que se usa para insertar filas en una tabla es la sentencia **DELETE**

Ejemplo 1: podemos **eliminar todas las filas de una tabla** usando DELETE sin cláusula WHERE

The screenshot shows the MySQL Workbench interface. In the top-left, it says "Query 1". To the right, there's a toolbar with various icons. Below the toolbar, the query text is displayed in four numbered lines:

- 1 • `use centro_educativo;`
- 2
- 3 • `delete from Ciclo;`
- 4

Below the query text is a "Action Output" section. It contains a table with three columns: "#", "Time", and "Message". There is one row in the table:

#	Time	Action
1	12:53:02	delete from Ciclo

To the right of the table, the message column shows "3 row(s) affected".

Vemos que se han eliminado todas (tres) las filas de la tabla

Nos aseguramos que se han eliminado todas las filas:

The screenshot shows the MySQL Workbench interface. In the top-left, it says "Query 1". To the right, there's a toolbar with various icons. Below the toolbar, the query text is displayed in five numbered lines:

- 1 • `use centro_educativo;`
- 2
- 3 • `select count(*)`
- 4 `from Ciclo;`
- 5

Below the query text is a "Result Grid" section. It contains a table with two columns: "#" and "count(*)". There is one row in the table:

#	count(*)
1	0

NOTA: Para que MySQL permita **eliminar y modificar filas** sin cláusula WHERE en la sentencia DELETE es necesario tener la variable **sql_safe_updates** con valor 0 (OFF)

Vemos que si **sql_safe_updates** vale 1 (ON)

The screenshot shows the MySQL Workbench interface with the following details:

- Query 1:** Administration - Status and System Variables
- SQL Editor:** Contains the following code:

```
1 • use centro_educativo;
2
3 • show variables like 'sql_safe_updates';
4
```
- Result Grid:** Shows the output of the `show variables` command:

#	Variable_name	Value
1	sql_safe_updates	ON

entonces no permite hacer el DELETE sin WHERE

The screenshot shows the MySQL Workbench interface with the following details:

- Query 1:** Administration - Status and System Variables
- SQL Editor:** Contains the following code:

```
1 • use centro_educativo;
2
3 • delete from Ciclo;
4
```
- Action Output:** Shows the error message:

#	Time	Action	Message
*	1	13:03:31 delete from Ciclo	Error Code: 1175. You To disable safe mode,

Details of the error message:
Action: delete from Ciclo
Response: Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column. To disable safe mode, toggle the option in Preferences -> SQL Editor and reconnect.
Duration: 0,00021 sec

Podemos cambiar el valor de la variable **sql_safe_updates** con el comando **set**:

Query 1 Administration - Status and System Variables

```

1 • use centro_educativo;
2
3 • set sql_safe_updates = 0;
4
5 • show variables like 'sql_safe_updates';

```

Result Grid Filter Rows: Export: Wrap Cell Content:

#	Variable_name	Value
1	sql_safe_updates	OFF

Action Output

#	Time	Action	Message
1	13:09:52	set sql_safe_updates = 0	0 row(s) affected
2	13:09:55	show variables like 'sql_safe_updates'	1 row(s) returned

Ejemplo 2: podemos **eliminar algunas las filas de una tabla** usando DELETE con cláusula WHERE

Veamos qué filas tenemos en la tabla **Asignatura**

Query 1 Administration - Status and System Variables

```

1 • use centro_educativo;
2
3 • select *
4     from Asignatura;
5

```

Result Grid Filter Rows: Edit: Export:

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	165	Bases de datos
2	2	1	120	Lenguaje de marcas
3	3	2	90	Seguridad informática
4	4	1	110	Despliegue aplicaciones web
5	5	2	90	Fundamentos de hardware
6	6	1	180	Acceso a datos

Si eliminamos aquellas cuyo CodAsig sea mayor o igual que 5

Query 1 Administration - Status and System Variables

```

1 • use centro_educativo;
2
3 • delete from Asignatura
4     where CodAsig >= 5;
5

```

Action Output

#	Time	Action	Message
✓ 1	13:31:10	delete from Asignatura where CodAsig >= 5	2 row(s) affected

Vemos que se eliminan solamente esas dos filas:

Query 1 Administration - Status and System Variables

```

1 • use centro_educativo;
2
3 • select *
4     from Asignatura;
5

```

Result Grid

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	165	Bases de datos
2	2	1	120	Lenguaje de marcas
3	3	2	90	Seguridad informática
4	4	1	110	Despliegue aplicaciones web
*	NULL	NULL	NULL	NULL

Ejemplo 3: eliminamos más filas particulares usando la cláusula WHERE dentro de DELETE

Query 1 Administration - Status and System Variables

```
1 • use centro_educativo;
2
3 • delete from Asignatura
4     where Nombre like 'B%'
5         or Nombre like 'S%';
6
```

Action Output

#	Time	Action	Message
1	13:37:12	delete from Asignatura where Nombre like 'B%' or No...	2 row(s) affected

Comprobamos el resultado:

Query 1 Administration - Status and System Variables

```
1 • use centro_educativo;
2
3 • select *
4     from Asignatura;
5
```

Result Grid Filter Rows: Edit: Export/Import:

#	CodAsig	CodCF	NumHoras	Nombre
1	2	1	120	Lenguaje de marcas
2	4	1	110	Despliegue aplicaciones web
*	NULL	NULL	NULL	NULL

Vemos que se han eliminado solamente las dos filas que cumplían las condiciones en la cláusula WHERE

Modificación de registros

La sentencia que se usa para insertar filas en una tabla es la sentencia **UPDATE**

Vamos a modificar filas de la tabla **Asignatura**

The screenshot shows the MySQL Workbench interface. In the top-left, there's a 'Query 1' tab and an 'Administration - Status and System Variables' tab. Below the tabs is a toolbar with various icons. A dropdown menu says 'Limit to 1000 rows'. The main area contains the following SQL code:

```
1 • use centro_educativo;
2
3 • select *
4     from Asignatura;
5
```

Below the code is a 'Result Grid' table with the following data:

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	165	Bases de datos
2	2	1	120	Lenguaje de marcas
3	3	2	90	Seguridad informática
4	4	1	110	Despliegue aplicaciones web
5	5	2	90	Fundamentos de hardware
6	6	1	180	Acceso a datos
*	NULL	NULL	NULL	NULL

Ejemplo 1: podemos modificar la columna **NumHoras** de la asignatura de “Bases de datos” así

The screenshot shows the MySQL Workbench interface. In the top-left, there's a 'Query 1' tab and an 'Administration - Status and System Variables' tab. Below the tabs is a toolbar with various icons. A dropdown menu says 'Limit to 1000 rows'. The main area contains the following SQL code:

```
1 • use centro_educativo;
2
3 • update Asignatura
4     set NumHoras = 243
5     where CodAsig = 1;
```

Below the code is an 'Action Output' panel with the following data:

#	Time	Action	Message
1	15:41:28	update Asignatura set NumHoras = 243 where CodAsig...	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

Vemos que, como esperábamos, solamente una fila se ve afectada

Comprobamos que efectivamente se ha realizado la modificación de esa columna:

Query 1 x

```
1 • use centro_educativo;
2
3 • select NumHoras
4     from Asignatura
5     where Nombre = 'Bases de datos';
6
```

Result Grid Filter Rows: Export: Wrap Cell Content:

#	NumHoras
1	243

Ejemplo 2: aumentamos un 20% las horas de las asignaturas que tengan menos de 100 horas

Vemos que hay dos asignaturas de menos de 100 horas

Query 1 x Administration - Status and System Variables x

```
1 • use centro_educativo;
2
3 • select *
4     from Asignatura;
5
```

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

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	165	Bases de datos
2	2	1	120	Lenguaje de marcas
3	3	2	90	Seguridad informática
4	4	1	110	Despliegue aplicaciones web
5	5	2	90	Fundamentos de hardware
6	6	1	180	Acceso a datos
*	NULL	NULL	NULL	NULL

Aumentamos un 20% las horas de las asignaturas que tengan menos de 100 horas (vemos que se cambian dos filas “Rows matched: 2 Changed: 2”)

Query 1 x

```
1 • use centro_educativo;
2
3 • update Asignatura
4     set NumHoras = NumHoras*1.20
5     where NumHoras < 100;
```

Action Output v

#	Time	Action	Message
✓ 1	21:02:37	update Asignatura set NumHoras = NumHoras*1.20 w...	2 row(s) affected Rows matched: 2 Changed: 2 Warnings: 0

Vemos que los cambios se aplican para la columna NumHoras de ambas filas:

Query 1 x

```
1 • use centro_educativo;
2
3 • select *
4     from Asignatura;
5
```

Result Grid Filter Rows: A Edit: A Export

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	243	Bases de datos
2	2	1	120	Lenguaje de marcas
3	3	2	108	Seguridad informática
4	4	1	110	Despliegue aplicaciones web
5	5	2	108	Fundamentos de hardware
6	6	1	180	Acceso a datos

Ejemplo 3: ponemos a **null** la **columna NumHoras** de todas las filas de la **tabla Asignatura** (no necesitamos escribir cláusula WHERE)

Query 1 x

```
1 • use centro_educativo;
2
3 • update Asignatura
4     set NumHoras = null;
5
```

Action Output

#	Time	Action	Message
✓	1	21:11:13 update Asignatura set NumHoras = null	6 row(s) affected Rows matched: 6 Changed: 6 Warnings: 0

Vemos que se modifican 6 filas

Comprobamos que se ha realizado el cambio para todas las filas

Query 1 x

```
1 • use centro_educativo;
2
3 • select *
4     from Asignatura;
```

Result Grid Filter Rows: A Edit: Export

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	NULL	Bases de datos
2	2	1	NULL	Lenguaje de marcas
3	3	2	NULL	Seguridad informática
4	4	1	NULL	Despliegue aplicaciones web
5	5	2	NULL	Fundamentos de hardware
6	6	1	NULL	Acceso a datos
*	NULL	NULL	NULL	NULL

Ejemplo 4: se pueden modificar varias columnas separándolas con comas

Vamos a modificar el número de horas y el nombre de la asignatura de Base de datos:

The screenshot shows the MySQL Workbench interface. In the top tab bar, 'Query 1' is active, followed by 'SQL File 5*', 'SQL File 4*', and 'SQL File 5*'. Below the tabs is a toolbar with various icons. A dropdown menu says 'Limit to 1000 rows'. The main area contains the following SQL code:

```
1 • use centro_educativo;
2
3 • select *
4     from Asignatura;
5
```

Below the code is a 'Result Grid' table with the following data:

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	NULL	Bases de datos
2	2	1	NULL	Lenguaje de marcas
3	3	2	NULL	Seguridad informática
4	4	1	NULL	Despliegue aplicaciones web
5	5	2	NULL	Fundamentos de hardware
6	6	1	NULL	Acceso a datos
*	NULL	NULL	NULL	NULL

Realizamos el update de ambas columnas y vemos que **una fila se ve afectada y que una fila se ha cambiado**:

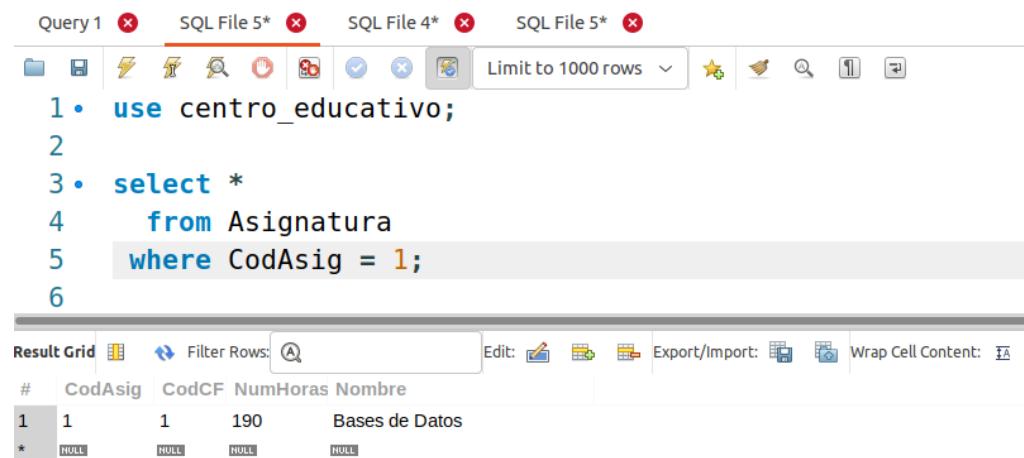
The screenshot shows the MySQL Workbench interface. In the top tab bar, 'Query 1' is active, followed by 'SQL File 5*', 'SQL File 4*', and 'SQL File 5*'. Below the tabs is a toolbar with various icons. A dropdown menu says 'Limit to 1000 rows'. The main area contains the following SQL code:

```
1 • use centro_educativo;
2
3 • update Asignatura
4     set NumHoras = 190,
5         Nombre = 'Bases de Datos'
6     where CodAsig = 1;
7
```

Below the code is an 'Action Output' table with the following data:

#	Time	Action	Message
1	01:41:11	update Asignatura set NumHoras = 190, Nombre = 'Bases de Datos' where CodAsig = 1;	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

Podemos comprobarlo:



The screenshot shows the MySQL Workbench interface. At the top, there are four tabs: 'Query 1' (selected), 'SQL File 5*', 'SQL File 4*', and 'SQL File 5*'. Below the tabs is a toolbar with various icons for file operations, search, and navigation. A dropdown menu says 'Limit to 1000 rows'. The main area contains a numbered SQL script:

```
1 • use centro_educativo;
2
3 • select *
4     from Asignatura
5     where CodAsig = 1;
6
```

Below the script is a 'Result Grid' section with the following data:

#	CodAsig	CodCF	NumHoras	Nombre
1	1	1	190	Bases de Datos
*	NULL	NULL	NULL	NULL

Nota: Se puede realizar la operación update para **dejar las filas y columnas de una tabla igual que ya estaban** (no tiene sentido hacerlo pero es sintácticamente aceptable)

Ejemplo 1: Vemos que una fila se ve afectada pero no se modifica ninguna fila

The screenshot shows the MySQL Workbench interface. At the top, there are four tabs: Query 1, SQL File 5*, SQL File 4*, and SQL File 5*. Below the tabs is a toolbar with various icons. A dropdown menu says "Limit to 1000 rows". The main area contains the following SQL code:

```
1 • use centro_educativo;
2
3 • update Asignatura
4     set Nombre = 'Lenguaje de marcas'
5     where CodAsig = 2;
6
```

Below the code is a table titled "Action Output" with columns: #, Time, Action, and Message. The table shows one row of data:

#	Time	Action	Message
1	01:49:00	update Asignatura set Nombre = 'Lenguaje de marcas' ...	0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0

Ejemplo 2: Vemos que seis filas se ven afectadas pero no se modifica ninguna fila

The screenshot shows the MySQL Workbench interface. At the top, there are four tabs: Query 1, SQL File 5*, SQL File 4*, and SQL File 5*. Below the tabs is a toolbar with various icons. A dropdown menu says "Limit to 1000 rows". The main area contains the following SQL code:

```
1 • use centro_educativo;
2 • set sql_safe_updates = OFF;
3
4 • update Asignatura
5     set Nombre = Nombre,
6         NumHoras = NumHoras;
7
```

Below the code is a table titled "Action Output" with columns: #, Time, Action, and Message. The table shows two rows of data:

#	Time	Action	Message
1	01:54:36	set sql_safe_updates = OFF	0 row(s) affected
2	01:54:47	update Asignatura set Nombre = Nombre, NumHor...	0 row(s) affected Rows matched: 6 Changed: 0 Warnings: 0

3. Borrados y modificaciones e integridad referencial. Subconsultas y composiciones en órdenes de edición.

Borrados y modificaciones e integridad referencial

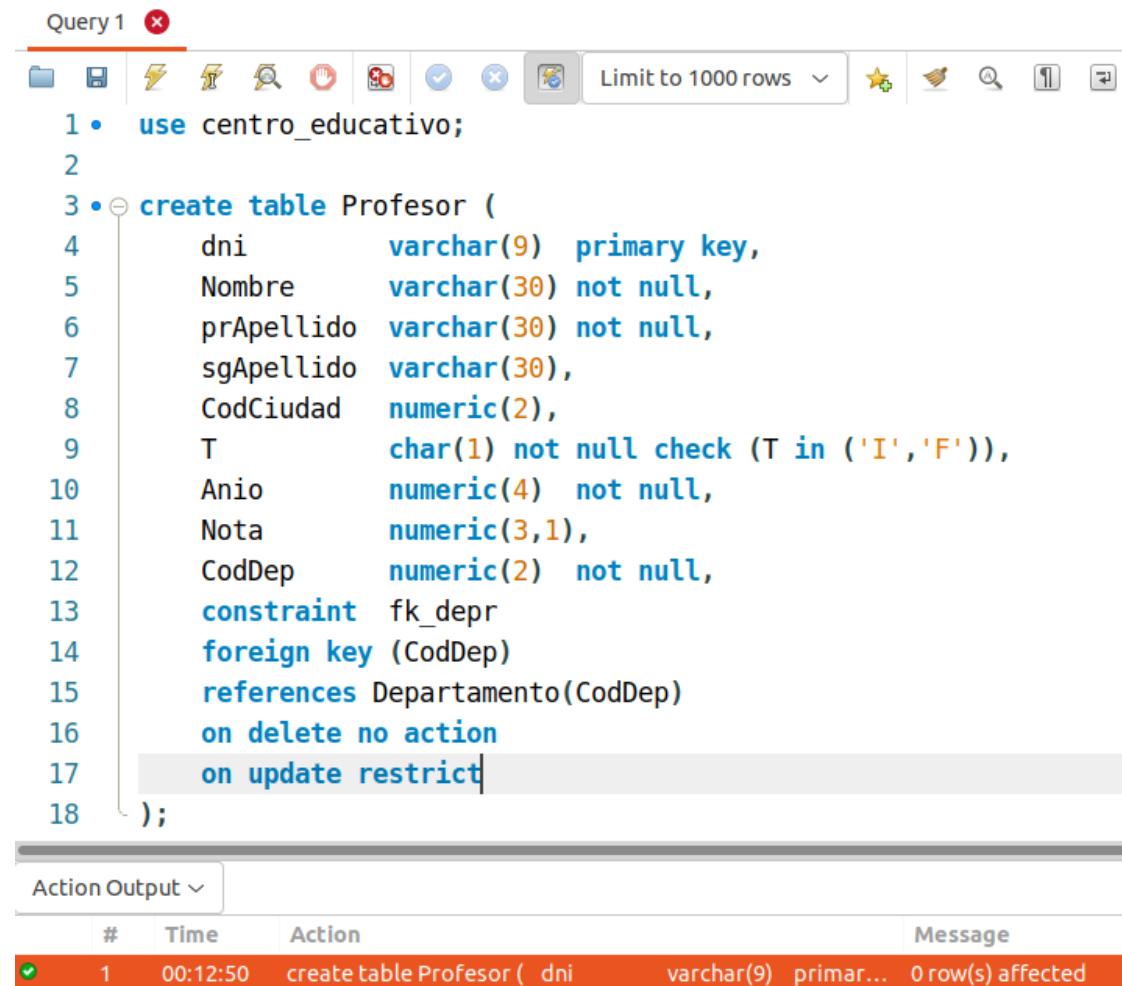
Recordemos las tres modalidades de creación de claves ajenas:

1. **NO ACTION** (es lo mismo que **RESTRICT**): es la **opción por defecto**, no permite eliminar ni modificar los registros que son referenciados desde otra tabla
2. **SET NULL**: se eliminan o modifican los registros referenciados y los que hacen referencia a ellos en otra se dejan como NULL (para lo cual hace falta que las columnas afectadas tengan la opción de ser NULL)
3. **CASCADE** (eliminación o modificación en cascada): se eliminan o modifican los registros referenciados y los que hacen referencia a ellos en otra tabla

NO ACTION

Creamos la tabla **Profesor** con una clave ajena con la opción **NO ACTION** (**RESTRICT**) tanto para **delete** como para **update**

Query 1 



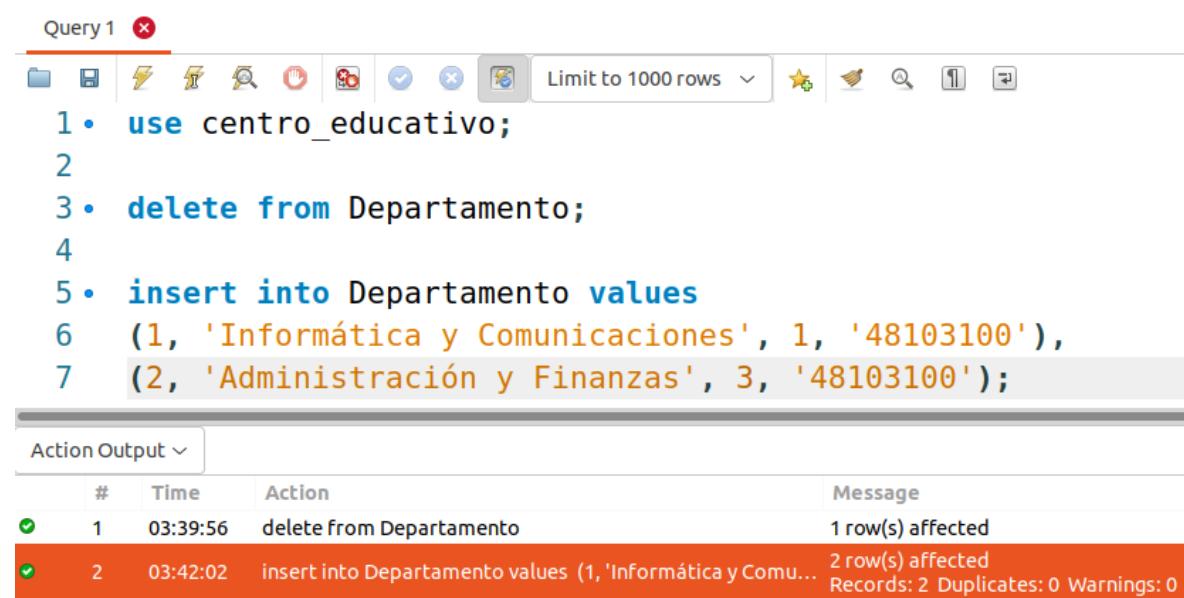
```
1 • use centro_educativo;
2
3 • create table Profesor (
4     dni      varchar(9) primary key,
5     Nombre    varchar(30) not null,
6     prApellido varchar(30) not null,
7     sgApellido varchar(30),
8     CodCiudad numeric(2),
9     T          char(1) not null check (T in ('I','F')),
10    Anio       numeric(4) not null,
11    Nota       numeric(3,1),
12    CodDep     numeric(2) not null,
13    constraint fk_depr
14    foreign key (CodDep)
15    references Departamento(CodDep)
16    on delete no action
17    on update restrict
18 );
```

Action Output 

#	Time	Action	Message
1	00:12:50	create table Profesor (dni varchar(9) primar...)	0 row(s) affected

Necesitamos tener estas dos filas en la tabla Departamento:

Query 1 



```
1 • use centro_educativo;
2
3 • delete from Departamento;
4
5 • insert into Departamento values
6     (1, 'Informática y Comunicaciones', 1, '48103100'),
7     (2, 'Administración y Finanzas', 3, '48103100');
```

Action Output 

#	Time	Action	Message
1	03:39:56	delete from Departamento	1 row(s) affected
2	03:42:02	insert into Departamento values (1, 'Informática y Comu...', 2)	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0

Necesitamos insertar estas filas en la tabla **Profesor**

Query 1 ×

```
1 • use centro_educativo;
2
3 • insert into Profesor values
4 ('44102321', 'Antonio',      'Martínez', 'González', 3, 'I', 2013, null, 1),
5 ('29600501', 'Dolores',     'Ramos',    'Cabrera',   4, 'F', 2019, 5.9, 1),
6 ('48300100', 'Ian',        'Oxley',    null,       5, 'I', 2004, null, 2),
7 ('84501495', 'Iván',       'Sánchez',  'Muñoz',    2, 'F', 2018, 9.1, 1),
8 ('48103100', 'Miguel Ángel', 'Martínez', 'Marín',    1, 'F', 2011, 7.3, 1),
9 ('90100200', 'Alejandro',   'Marín',    'Cantos',   1, 'F', 2019, 5.9, 1),
10 ('28900194', 'David',     'Negro',    'Catalán',  6, 'I', 2001, null, 1);
11
```

Action Output ▼

#	Time	Action	Message
✓	12:45:09	insert into Profesor values ('44102321','Antonio',...)	7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0

Ejemplo 1 (delete): Si intento eliminar una fila en la tabla **Departamento** vemos que la clave ajena **NO ACTION** no me lo permite

Query 1 ×

```
1 • use centro_educativo;
2
3 • delete from Departamento
4 where CodDep = 2; |
```

Action Output ▼

#	Time	Action	Message
✗	12:55:49	delete from Departamento where CodDep = 2	Error Code: 1451. Car

Action: delete from Departamento where CodDep = 2
Response: Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails (`centro_educativo`.`Profesor`, CONSTRAINT `fk_depr` FOREIGN KEY (`CodDep`) REFERENCES `Departamento` (`CodDep`) ON UPDATE RESTRICT)
Duration: 0,025 sec

Ejemplo 2 (update): Tampoco la clave ajena **NO ACTION** no me lo permite si intento modificar el valor de esa columna en la tabla **Departamento**

Query 1 X

The screenshot shows the MySQL Workbench interface. In the top bar, there are several icons for file operations, a search function, and a refresh button. A dropdown menu says "Limit to 1000 rows". Below the toolbar is a query editor window containing the following SQL code:

```
1 • use centro_educativo;
2
3 • update Departamento
4     set CodDep = 22
5     where CodDep = 2; |
```

Below the query editor is an "Action Output" table with the following data:

#	Time	Action	Message
x 1	13:01:39	update Departamento set CodDep = 22 where CodDep ...	Error Code: 1451. Ca...

A tooltip box is overlaid on the right side of the table, displaying the following information:

Action: update Departamento set CodDep = 22 where CodDep = 2
Response: Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails ('centro_educativo`.`Profesor`, CONSTRAINT `fk_depr` FOREIGN KEY (`CodDep`) REFERENCES `Departamento`(`CodDep`) ON UPDATE RESTRICT)
Duration: 0,0013 sec

SET NULL

Necesitamos crear la tabla **Ciudad**

Query 1 x

The screenshot shows the MySQL Workbench interface. At the top is a toolbar with various icons. Below it is a query editor window titled "Query 1" containing the following SQL code:

```
1 • use centro_educativo;
2
3 • create table Ciudad (
4     CodCiudad    numeric(2) primary key,
5     Nombre       varchar(15) not null
6 );
7
```

Below the query editor is an "Action Output" table with the following data:

#	Time	Action	Message
✓ 1	14:22:05	create table Ciudad (CodCiudad numeric(2) primar... 0 row(s) affected	

Y necesitamos insertar estas filas en la tabla **Ciudad**

Query 1 x

The screenshot shows the MySQL Workbench interface. At the top is a toolbar with various icons. Below it is a query editor window titled "Query 1" containing the following SQL code:

```
1 • use centro_educativo;
2
3 • insert into Ciudad values
4     (1, 'Madrid'),
5     (2, 'Barcelona'),
6     (3, 'Sevilla'),
7     (4, 'Málaga'),
8     (5, 'Londres'),
9     (6, 'Bilbao'),
10    (7, 'Granada'),
11    (8, 'Valencia'),
12    (9, 'Pamplona'),
13    (10,'Ciudad Real'),
14    (11,'Tarragona');
```

Below the query editor is an "Action Output" table with the following data:

#	Time	Action	Message
✓ 1	14:26:14	insert into Ciudad values (1, 'Madrid'), (2, 'Barcelona')... 11 row(s) affected Records: 11 Duplicates: 0 Warnings: 0	

Creamos una nueva clave ajena en la tabla **Profesor** con la opción **SET NULL** tanto para **delete** como para **update** que refuerce el **CodCiudad** de **Ciudades**

Query 1 X

The screenshot shows the MySQL Workbench interface. The top bar has tabs for 'Query 1' and 'Results'. Below the tabs is a toolbar with various icons for database management. A dropdown menu says 'Limit to 1000 rows'. The main area contains a numbered list of SQL statements. The bottom part shows an 'Action Output' pane with a table of log entries.

```
1 • use centro_educativo;
2
3 • alter table Profesor
4   add constraint Profesor_Ciudad_FK
5   foreign key (CodCiudad)
6   references Ciudad (CodCiudad)
7   on delete set null
8   on update set null;
9
```

Action Output

#	Time	Action	Message
1	00:22:29	alter table Profesor add constraint Profesor_Ciudad_F...	7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0

Ejemplo 1 (delete): Miramos primero qué profesores tienen como ciudad **Madrid** (CodCiudad = 1)

Si intento eliminar la fila de Madrid en la tabla Ciudad vemos que sí puedo

Query 1 x

The screenshot shows a MySQL Workbench interface. The title bar says "Query 1". Below it is a toolbar with various icons: folder, save, lightning bolt, trophy, magnifying glass, hand, redo, undo, checkmark, cross, and a globe. To the right of the toolbar is a dropdown menu "Limit to 1000 rows". Further right are icons for star, envelope, magnifying glass, and a question mark. Below the toolbar is a search bar with "Find" selected, followed by "Replace Found match", "set", and two delete buttons. The main area contains the following SQL code:

```
1 • use centro_educativo;  
2  
3 • delete from Ciudad  
   where Nombre = 'Madrid';  
4  
5
```

Action Output ▼

#	Time	Action	Message
1	00:32:00	delete from Ciudad where Nombre = 'Madrid'	1 row(s) affected

Si miramos de nuevo qué profesores tienen el CodCiudad **nulo** vemos que los son los que antes tenían a **Madrid**

Ejemplo 2 (update): Miramos primero qué profesores tienen como ciudad Barcelona (CodCiudad = 2)

```
Query 1 •
1 • use centro_educativo;
2
3 • select * from Profesor
4     where CodCiudad = 2;
5

Result Grid Filter Rows: A Edit: E Export/Import: E Wrap Cell Content: E
#   dni      Nombre    prApellido sgApellido CodCiudad T Anio Nota CodDep
1   84501495  Iván       Sánchez    Muñoz        2       F  2018  9.1   1
*   NULL      NULL       NULL       NULL        NULL      NULL  NULL  NULL  NULL
```

Si intento eliminar cambiar en la tabla **Ciudad** el CodCiudad = 2 por CodCiudad = 22 veo que sí puedo

Query 1 X

Limit to 1000 rows

```
1 • use centro_educativo;
2
3 • update Ciudad
   set CodCiudad = 22
5 where CodCiudad = 2;
6
```

Action Output ▾

#	Time	Action	Message
1	01:00:33	update Ciudad set CodCiudad = 22 where CodCiudad = 2	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

Pero si veo que de nuevo qué profesores tienen el CodCiudad **nulo** vemos que aparece ahora el profesor que estaba asignado a Barcelona (Iván Sánchez Muñoz):

CASCADE

Vamos a modificar la tabla **Profesor** para eliminar la clave ajena Profesor_Ciudad_FK (SET NULL para delete y update) y para que cree una nueva clave ajena sobre la misma columna que borre y actualice en cascada:

Query 1 X



Limit to 1000 rows ▼

1 • **use centro_educativo;**
2
3 • **alter table Profesor**
4 **drop constraint Profesor_Ciudad_FK,**
5 **add constraint Profesor_Ciudad_FK_Cascade**
6 **foreign key (CodCiudad)**
7 **references Ciudad(CodCiudad)**
8 **on delete cascade**
9 **on update cascade;**
10

Action Output ▼

#	Time	Action	Message
✓	1	02:50:53	alter table Profesor drop constraint Profesor_Ciudad_F...
			7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0

Ejemplo 1 (delete): Miramos primero qué profesores tienen como ciudad **Londres** (CodCiudad = 5)

```
Query 1 X
Limit to 1000 rows ▼
1 • use centro_educativo;
2
3 • select *
   from Profesor
5   where CodCiudad = 5;
6

Result Grid Filter Rows: A Edit: A C D Export/Import: E F Wrap Cell Content: G
# dni Nombre prApellido sgApellido CodCiudad T Anio Nota CodDep
1 48300100 Ian Oxley NULL 5 I 2004 NULL 2
* NULL NULL NULL NULL NULL NULL NULL NULL NULL
```

Si intentamos eliminar la fila de **Londres** en la tabla **Ciudad** vemos que efectivamente se elimina una fila:

Query 1 X

1 • `use centro_educativo;`
2
3 • `delete from Ciudad`
4 `where CodCiudad = 5;`

Action Output ▼

#	Time	Action	Message
1	02:58:33	delete from Ciudad where CodCiudad = 5	1 row(s) affected

Miramos de nuevo qué profesores tienen como ciudad **Londres** (`CodCiudad = 5`) y vemos que ya no hay ninguno (se ha eliminado en cascada):

Query 1 X

1 • `use centro_educativo;`
2
3 • `select *`
4 `from Profesor`
5 `where CodCiudad = 5;`

Result Grid ▼

#	dni	Nombre	prApellido	sgApellido	CodCiudad	T	Anio	Nota	CodDep
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Action Output ▼

#	Time	Action	Message
1	02:58:33	delete from Ciudad where CodCiudad = 5	1 row(s) affected
2	03:01:02	select * from Profesor where CodCiudad = 5 LIMIT 0, 1000	0 row(s) returned

Ejemplo 2 (update): Miramos primero qué profesores tienen como ciudad **Málaga** (CodCiudad = 4)

Query 1 ✖

```
1 • use centro_educativo;
2
3 • select *
4     from Profesor
5     where CodCiudad = 4;
6
```

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

#	dni	Nombre	prApellido	sgApellido	CodCiudad	T	Anio	Nota	CodDep	
1	29600501	Dolores	Ramos		Cabrera	4	F	2019	5.9	1
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	

Si intentamos cambiar el código de **Málaga** en la tabla **Ciudad** vemos que efectivamente se modifica una fila:

Query 1 ✖

```
1 • use centro_educativo;
2
3 • update Ciudad
4     set CodCiudad = 44
5     where CodCiudad = 4;
6
```

Action Output ✚

#	Time	Action	Message
1	03:06:10	update Ciudad set CodCiudad = 44 where CodCiudad = 4	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

Finalmente, si comprobamos todos los profesores, vemos que a **Dolores Ramos Cabrera** se le ha modificado el código de su ciudad en cascada (ha pasado de ser el 4 a ser el 44):

```
Query 1 •
1 • use centro_educativo;
2
3 • select *
   from Profesor;
4
5

Result Grid Filter Rows: (A) Edit: Export/Import: Wrap Cell Content: (A)
# dni Nombre prApellido sgApellido CodCiudad T Anio Nota CodDep
1 28900194 David Negro Catalán 6 I 2001 NULL 1
2 29600501 Dolores Ramos Cabrera 44 F 2019 5.9 1
3 44102321 Antonio Martínez González 3 I 2013 NULL 1
4 48103100 Miguel Ángel Martínez Marín NULL F 2011 7.3 1
5 84501495 Iván Sánchez Muñoz NULL F 2018 9.1 1
6 90100200 Alejandro Marín Cantos NULL F 2019 5.9 1
* NULL NULL NULL NULL NULL NULL NULL NULL NULL
```

Subconsultas y composiciones en órdenes de edición

Sentencias **INSERT** con subconsultas

Ejemplo: En la base de datos **employees** creamos una copia vacía (limit 0) de la tabla de **salarios** llamada **5_mejores_sueldos**:

Query 1 ✖

```
1 • use employees;
2
3 • create table 5_mejores_sueldos
4     as (select *
5          from salaries limit 0);
6
```

Action Output ▼

#	Time	Action	Message
1	03:58:56	create table 5_mejores_sueldos as (select * from ...)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Vemos que efectivamente está vacía

Query 1 ✖

```
1 • use employees;
2
3 • select count(*)
4     from 5_mejores_sueldos;
5
```

Result Grid ▼ Filter Rows: Ⓐ Export: ⎙ Wrap Cell Content: ⤒

#	count(*)
1	0

Insertamos en la tabla **5_mejores_sueldos** algunos salarios usando una select dentro de la sentencia insert:

Query 1 x

The screenshot shows the MySQL Workbench interface with a query window titled "Query 1". The query itself is:

```
1 • use employees;
2
3 • insert into 5_mejores_sueldos
4   select *
5     from salaries
6   where salary > 155377;
7
```

Below the query window is the "Action Output" pane, which displays the results of the execution:

Action	Message
insert into 5_mejores_sueldos select * from salaries w...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0

Y comprobamos que se han insertado las cinco filas:

Query 1 x

The screenshot shows the MySQL Workbench interface with a query window titled "Query 1". The query is:

```
1 • use employees;
2
3 • select *
4   from 5_mejores_sueldos;
5
```

Below the query window is the "Result Grid" pane, which displays the data from the table:

#	emp_no	salary	from_date	to_date
1	43624	157821	2001-03-22	2002-03-22
2	43624	158220	2002-03-22	9999-01-01
3	47978	155709	2002-07-14	9999-01-01
4	253939	155513	2002-04-11	9999-01-01
5	254466	156286	2001-08-04	9999-01-01

NOTA: Se podría haber hecho también habiendo incluido los nombres de las columnas en el insert y/o en el select

Query 1 



```
1 • use employees;
2
3 • truncate table 5_mejores_sueldos;
4
5 • insert into 5_mejores_sueldos (salary, emp_no, from_date, to_date)
6   select salary, emp_no, from_date, to_date
7     from salaries
8   where salary > 155377;
```

Action Output ▾

#	Time	Action	Message
1	12:15:23	truncate table 5_mejores_sueldos	0 row(s) affected
2	12:15:25	insert into 5_mejores_sueldos (salary, emp_no, from_da...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0

Sentencias **DELETE** con subconsultas

Ejemplo: vamos a usar una subconsulta para seleccionar qué filas vamos a eliminar de la tabla **5_mejores_sueldos**

Query 1 ×

The screenshot shows the MySQL Workbench interface. At the top is a toolbar with various icons. Below it is a query editor window titled "Query 1" containing the following SQL code:

```
1 • use employees;
2
3 • select *
4     from 5_mejores_sueldos;
```

Below the query editor is a "Result Grid" window displaying the results of the SELECT query. The grid has columns: #, emp_no, salary, from_date, and to_date. The data is as follows:

#	emp_no	salary	from_date	to_date
1	43624	157821	2001-03-22	2002-03-22
2	43624	158220	2002-03-22	9999-01-01
3	47978	155709	2002-07-14	9999-01-01
4	253939	155513	2002-04-11	9999-01-01
5	254466	156286	2001-08-04	9999-01-01

Ejecutamos la sentencia y vemos que se elimina una fila: “*1 row(s) affected*”

Query 1 ×

The screenshot shows the MySQL Workbench interface. At the top is a toolbar with various icons. Below it is a query editor window titled "Query 1" containing the following SQL code:

```
1 • use employees;
2
3 • delete from 5_mejores_sueldos a
4   where emp_no = (select emp_no
5           from employees b
6           where a.emp_no = b.emp_no
7               and first_name = 'Honesty');
```

Below the query editor is an "Action Output" table showing the results of the DELETE operation. The table has columns: #, Time, Action, and Message.

#	Time	Action	Message
1	14:21:49	delete from 5_mejores_sueldos a where emp_no = (sele...	1 row(s) affected

Comprobamos que efectivamente se ha eliminado una fila en la tabla **5_mejores_sueldos**

Query 1 X

The screenshot shows a MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it is a query editor window titled 'Query 1' containing the following SQL code:

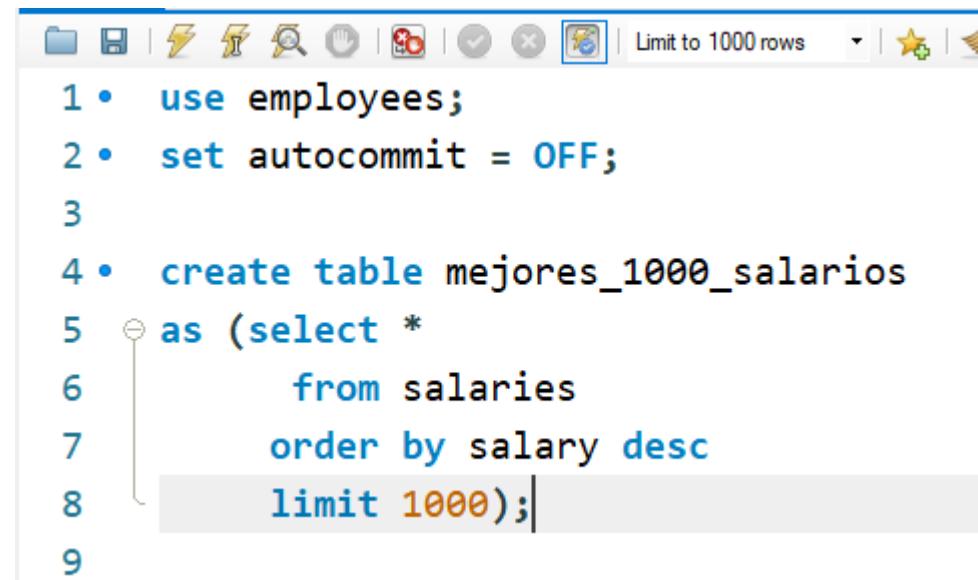
```
1 • use employees;
2
3 • select *
4     from 5_mejores_sueldos;
5
```

Below the query editor is a 'Result Grid' table. The table has four columns: '#', 'emp_no', 'salary', 'from_date', and 'to_date'. It contains four rows of data:

#	emp_no	salary	from_date	to_date
1	43624	157821	2001-03-22	2002-03-22
2	43624	158220	2002-03-22	9999-01-01
3	47978	155709	2002-07-14	9999-01-01
4	253939	155513	2002-04-11	9999-01-01

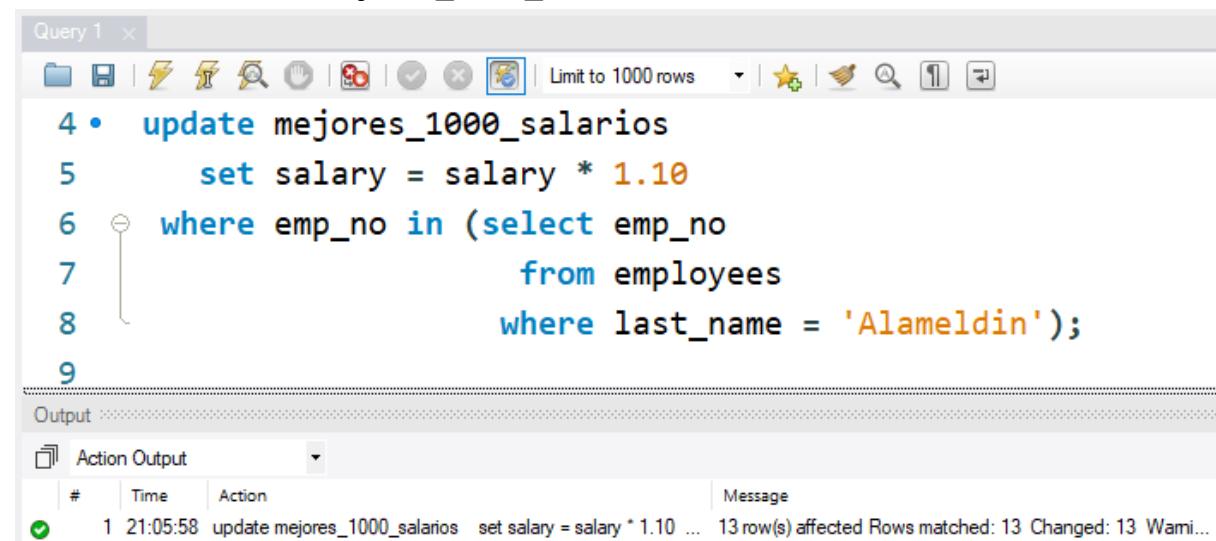
Sentencias UPDATE con subconsultas

Vamos a practicar en la base de datos **employees** con una copia de la tabla salaries que llamaremos **mejores_1000_salarios**:



```
1 • use employees;
2 • set autocommit = OFF;
3
4 • create table mejores_1000_salarios
5   as (select *
6        from salaries
7        order by salary desc
8        limit 1000);
9
```

Ejemplo 1: vamos a usar una subconsulta para seleccionar qué filas vamos a actualizar de la tabla **mejores_1000_salarios**



```
Query 1
4 • update mejores_1000_salarios
5   set salary = salary * 1.10
6   where emp_no in (select emp_no
7                     from employees
8                     where last_name = 'Alameldin');
9
```

Output

Action Output
Time Action Message
1 21:05:58 update mejores_1000_salarios set salary = salary * 1.10 ... 13 row(s) affected Rows matched: 13 Changed: 13 Wami...

Vemos que se modifican 13 filas (hemos aumentado esos salarios un 10%)

Ejemplo 2: añadimos una subconsulta y otros filtros para seleccionar qué filas vamos a actualizar de la tabla **mejores_1000_salarios**

The screenshot shows the Oracle SQL Developer interface. In the top panel, there is a toolbar with various icons and a "Query 1" tab. Below the toolbar is a code editor containing the following SQL script:

```
4 • update mejores_1000_salarios
5      set salary = salary * 1.10
6  where emp_no in (select emp_no
7                      from employees
8                     where last_name = 'Portugali')
9              and from_date >= '199-01-01';
10
```

In the bottom panel, there is an "Output" window titled "Action Output". It contains a table with the following data:

#	Time	Action	Message
1	21:08:14	update mejores_1000_salarios set salary = salary * 1.10 ...	2 row(s) affected Rows matched: 2 Changed: 2 Warnings...

Vemos que se modifican 2 filas (hemos aumentado esos salarios un 5%)

Ejemplo 3: añadimos dos subconsultas anidadas para seleccionar qué filas vamos a actualizar de la tabla **mejores_1000_salarios**

The screenshot shows the Oracle SQL Developer interface. In the top panel, there is a toolbar with various icons and a "Query 1" tab. Below the toolbar is a code editor containing the following SQL script:

```
4 • update mejores_1000_salarios
5      set salary = salary * 1.15
6  where emp_no in (select emp_no
7                      from dept_emp
8                     where dept_no = (select dept_no
9                           from departments
10                          where dept_name = 'Finance')));
11
```

In the bottom panel, there is an "Output" window titled "Action Output". It contains a table with the following data:

#	Time	Action	Message
1	21:12:32	update mejores_1000_salarios set salary = salary * 1.15 where e... where dept_no = (select dept_no from departments where dept_name = 'Finance'));	29 row(s) affected Rows matched: 29 Changed: 29 Warnings: 0

Vemos que se modifican 29 filas (hemos aumentado esos salarios un 15%)

4. Transacciones

Una **transacción** es un **conjunto de sentencias SQL que se tratan como una única instrucción atómica**.

Es decir:

- Si alguna sentencia no se puede ejecutar, entonces se anulan todas las demás (o funciona **todo** o no se cambia **nada**)
- Se valida la transacción solamente si todas las sentencias se pueden ejecutar (o funciona **todo** o no se cambia **nada**)

Una **transacción termina cuando**:

- **Se confirma** (commit): si todas las operaciones individuales se ejecutaron correctamente
 - Por ejemplo, para realizar un pedido será necesario modificar e insertar filas en varias tablas:
 - modificar la cantidad de unidades disponibles de cada artículo en la tabla de artículos
 - insertar una fila en la tabla de pedidos pendientes
 - insertar una fila en la tabla de envíos

Si todo ha ido bien, se confirmarán ambas operaciones solamente después de insertar el nuevo pedido (**todo** o **nada**)

- **Se aborta** (rollback): si a mitad de su ejecución ha habido algún problema
 - Por ejemplo, si un producto del pedido no está disponible, entonces no te puede generar el envío), entonces:
 - hay que anular las modificaciones hechas a la cantidad de unidades disponibles de cada artículo en la tabla de artículos
 - hay que eliminar la fila en la tabla de pedidos pendientes

Abortar una transacción implica deshacer todas las operaciones individuales desde el principio de la transacción (**todo** o **nada**).

Propiedades de las transacciones

Las transacciones deben cumplir una serie de propiedades:

- **Aislamiento**: el resultado de una transacción no es visto por otras transacciones hasta que termina en su totalidad.

- **Atomicidad:** todas las operaciones de una transacción se realizan correctamente, sin ningún error, o bien no se realiza ninguna.
- **Consistencia:** cuando las acciones asociadas a una transacción comienzan se tiene la certeza de que, antes de su inicio, la base de datos estaba en un estado consistente. Cuando finaliza la transacción, el nuevo estado también debe ser totalmente consistente.
- **Durabilidad:** a partir de que una transacción finaliza con éxito, no se puede volver al estado anterior, es decir, el nuevo estado continúa en el tiempo hasta que se ejecute una nueva transacción.

El **TCL** (Lenguaje de Control de Transacciones) de SQL es muy simple, básicamente se compone de dos sentencias:

- **COMMIT** valida una transacción
- **ROLLBACK** deshace una transacción

Nota: por defecto MySQL funciona en modo **AUTOCOMMIT** lo cual significa que se realiza siempre un COMMIT después de cada sentencia INSERT, DELETE o UPDATE con lo cual no tiene sentido usar ROLLBACK nunca

```
profe@profe-VirtualBox:~$ 
profe@profe-VirtualBox:~$ mysql -u root -pASIR2
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 8.0.27-0ubuntu0.20.04.1 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show variables like 'autocommit';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| autocommit    | ON   |
+-----+-----+
1 row in set (0,00 sec)

mysql>
mysql>
```

El usuario **root** puede deshabilitar el **AUTOCOMMIT** con el comando:

Query 1 X

```
1 • use centro_educativo;
2
3 • set autocommit = 0;
4 • show variables like 'autocommit';|
```

Result Grid Filter Rows: A Export: Wrap Cell Content: IA

#	Variable_name	Value
1	autocommit	OFF

Result 1 X

Action Output

#	Time	Action	Message
1	13:00:00	set autocommit = 0	0 row(s) affected
2	13:00:36	show variables like 'autocommit'	1 row(s) returned

Ejemplo 1: Para a ver cómo funcionan las sentencias y **COMMIT** y **ROLLBACK**

Comprobamos primero qué filas hay en la tabla **Ciudad**

Query 1 X

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • select * from Ciudad;|
```

Result Grid Filter Rows: A Edit: A Export/Import: A Wrap Cell Content: IA

#	CodCiudad	Nombre
1	3	Sevilla
2	6	Bilbao
3	7	Granada
4	8	Barcelona
5	9	Pamplona
6	10	Ciudad Real
7	11	Tarragona
8	22	Barcelona
9	44	Málaga
*	NULL	NULL

A continuación modifco el código de **Barcelona** y realizo un **COMMIT**

Query 1 ✖

The screenshot shows the MySQL Workbench interface. In the top panel, there is a query editor window titled "Query 1" containing the following SQL code:

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • update Ciudad
5      set CodCiudad = 2
6      where CodCiudad = 22;
7
8 • commit;
```

In the bottom panel, there is an "Action Output" table with the following data:

Action	Message
update Ciudad set CodCiudad = 2 where CodCiudad = 22	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
commit	0 row(s) affected

A continuación modifco el código de **Málaga** y realizo un **ROLLBACK**

Query 1 ✖

The screenshot shows the MySQL Workbench interface. In the top panel, there is a query editor window titled "Query 1" containing the following SQL code:

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • update Ciudad
5      set CodCiudad = 4
6      where CodCiudad = 44;
7
8 • rollback;
```

In the bottom panel, there is an "Action Output" table with the following data:

Action	Message
update Ciudad set CodCiudad = 4 where CodCiudad = 44	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
rollback	0 row(s) affected

Comprobamos de nuevo las filas de la tabla Ciudad

Query 1 

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • select *
5     from Ciudad;
6
```

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

#	CodCiudad	Nombre
1	2	Barcelona
2	3	Sevilla
3	6	Bilbao
4	7	Granada
5	8	Barcelona
6	9	Pamplona
7	10	Ciudad Real
8	11	Tarragona
9	44	Málaga
*	NULL	NULL

Observamos que Barcelona se ha modificado (porque se hizo **commit**) pero Málaga no (porque se hizo **rollback**)

Ejemplo 2: Para a ver cómo funcionan las sentencias y **COMMIT** y **ROLLBACK**

Comprobamos de nuevo qué filas hay en la tabla **Ciudad**

The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query editor contains the following SQL code:

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • select *
   from Ciudad;
```

The result grid displays the following data:

#	CodCiudad	Nombre
1	2	Barcelona
2	3	Sevilla
3	6	Bilbao
4	7	Granada
5	8	Barcelona
6	9	Pamplona
7	10	Ciudad Real
8	11	Tarragona
9	44	Málaga
*	HULL	HULL

Insertamos una fila para **Madrid**

The screenshot shows the MySQL Workbench interface with a query editor and an action output log. The query editor contains the following SQL code:

```
2 • set autocommit = 0;
3
4 • insert into Ciudad (CodCiudad, Nombre)
  values (1, 'Madrid');
```

The action output log shows the following entry:

Action Output			
#	Time	Action	Message
1	03:01:27	insert into Ciudad (CodCiudad, Nombre) values (1, 'Madrid')	1 row(s) affected

A continuación modificamos el CodCiudad de Málaga y validamos la transacción (hacemos **COMMIT**)

Query 1 ✖

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • update Ciudad
5     set CodCiudad = 4
6     where CodCiudad = 44;
7
8 • commit;
9
```

Action Output ▼

#	Time	Action	Message
1	03:07:17	update Ciudad set CodCiudad = 4 where CodCiudad = 44	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
2	03:07:20	commit	0 row(s) affected

Seguidamente insertamos una fila para **Zaragoza** y deshacemos la transacción (hacemos rollback)

Query 1 ✖

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • insert into Ciudad (CodCiudad, Nombre)
5     values (12, 'Zaragoza');
6
7 • rollback;
```

Action Output ▼

#	Time	Action	Message
1	03:17:20	insert into Ciudad (CodCiudad, Nombre) values (12, 'Zara...')	1 row(s) affected
2	03:17:23	rollback	0 row(s) affected

Finalmente comprobamos qué filas hay en la tabla de **Ciudades**

Query 1 ×

1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • select *
from Ciudad;ll

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

#	CodCiudad	Nombre
1	1	Madrid
2	2	Barcelona
3	3	Sevilla
4	4	Málaga
5	6	Bilbao
6	7	Granada
7	8	Barcelona
8	9	Pamplona
9	10	Ciudad Real
10	11	Tarragona
*	NULL	NULL

Ciudad 9 ×

Action Output ▼

#	Time	Action	Message
1	03:20:50	insert into Ciudad (CodCiudad, Nombre) values (12, 'Zara...')	1 row(s) affected
2	03:20:53	rollback	0 row(s) affected
3	03:20:57	select * from Ciudad LIMIT 0, 1000	10 row(s) returned

Vemos que aparecen los cambios validados con el commit (aparece Madrid y Málaga tiene el código 4) pero no los cambios deshechos con el rollback (no aparece Zaragoza)

Comando SET TRANSACTION

En MySQL si no queremos deshabilitar la opción autocommit se pueden iniciar transacciones manualmente con el comando **START TRANSACTION** (que terminará cuando se haga rollback o commit)

Ejemplo 1: Con el **autocommit habilitado** podemos hacer **rollback** de un insert si usamos **start transaction** antes. En ese momento **termina la transacción** y como el autocommit está habilitado al insert siguiente ya no se le podrá hacer rollback

The screenshot shows the MySQL Workbench interface. The top bar has tabs for 'Query 1', 'SQL File 3*', and 'SQL File 4*'. The 'SQL File 4*' tab is active, showing the following SQL code:

```
1 • set autocommit = 1;
2 • start transaction;
3 • insert into Ciudad (CodCiudad, Nombre) values (13,'Valladolid');
4 • rollback;
5 • insert into Ciudad (CodCiudad, Nombre) values (14,'La Coruña');
6 • rollback;
7
8 • select *
  from Ciudad
9   where CodCiudad in (13, 14);
11
```

The results grid below shows the output of the final select statement:

#	CodCiudad	Nombre
1	14	La Coruña
*	NULL	NULL

The 'Action Output' section at the bottom displays the following log entries:

#	Time	Action	Message
1	15:29:27	set autocommit = 1	0 row(s) affected
2	15:29:29	start transaction	0 row(s) affected
3	15:29:31	insert into Ciudad (CodCiudad, Nombre) values (13,'Valladolid')	1 row(s) affected
4	15:29:34	rollback	0 row(s) affected
5	15:29:36	insert into Ciudad (CodCiudad, Nombre) values (14,'La Coruña')	1 row(s) affected
6	15:29:38	rollback	0 row(s) affected
7	15:29:40	select * from Ciudad where CodCiudad in (13, 14) LIMIT ...	1 row(s) returned

Ejemplo 2: Con el **autocommit habilitado** usamos **start transaction**. Pero no podemos hacer **rollback** de un insert si ya antes hemos hecho commit (porque en ese momento termina la transacción el ya vuelve a funcionar el autocommit)

The screenshot shows the MySQL Workbench interface. In the top tab bar, 'Query 1' and 'SQL File 3*' are closed, while 'SQL File 4*' is active. Below the tabs is a toolbar with various icons. A dropdown menu says 'Limit to 1000 rows'. The main area contains the following SQL code:

```

1 • set autocommit = 1;
2 • start transaction;
3 • insert into Ciudad (CodCiudad, Nombre) values (13, 'Valladolid');
4 • commit;
5 • insert into Ciudad (CodCiudad, Nombre) values (15, 'Valencia');
6 • rollback;
7
8 • select *
9     from Ciudad
10    where CodCiudad in (13, 15);

```

Below the code is a 'Result Grid' table:

#	CodCiudad	Nombre
1	13	Valladolid
2	15	Valencia
*	NULL	NULL

At the bottom, there's an 'Action Output' section with a table:

#	Time	Action	Message
1	14:54:31	select * from Ciudad LIMIT 0, 1000	12 row(s) returned
2	14:55:14	set autocommit = 1	0 row(s) affected
3	14:55:27	start transaction	0 row(s) affected
4	14:55:30	insert into Ciudad (CodCiudad, Nombre) values (13,'Valladolid')	1 row(s) affected
5	14:55:33	commit	0 row(s) affected
6	14:55:35	insert into Ciudad (CodCiudad, Nombre) values (15,'Valencia')	1 row(s) affected
7	14:55:38	rollback	0 row(s) affected
8	14:55:42	select * from Ciudad where CodCiudad in (13, 15) LIMIT ...	2 row(s) returned

Nota: la operación **rollback** solamente vale para las sentencias **DML** (insert, delete y update)

Ejemplo 1: No se puede hacer **rollback** de las sentencias **DDL**

Hacemos un **desc** de la tabla **Ciudades** y vemos que **Nombre** tiene un tamaño de 15

Query 1

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • desc Ciudad;
5
```

Result Grid

#	Field	Type	Null	Key	Default	Extra
1	CodCiudad	decimal(2,0)	NO	PRI	NULL	
2	Nombre	varchar(15)	NO		NULL	

Primero quitamos el auto commit y a continuación alteramos la tabla Ciudades para aumentar a 20 el tamaño de la columna Nombre, seguidamente hacemos rollback

Query 1

```
1 • use centro_educativo;
2
3 • set autocommit = 0;
4
5 • alter table Ciudad
6   modify column Nombre varchar(20) not null;
7
8 • rollback;
9
```

Result Grid

#	Field	Type	Null	Key	Default	Extra
1	CodCiudad	decimal(2,0)	NO	PRI	NULL	
2	Nombre	varchar(20)	NO		NULL	

Action Output

#	Time	Action	Message
1	03:37:03	set autocommit = 0	0 row(s) affected
2	03:37:07	alter table Ciudad modify column Nombre varchar(20) n...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
3	03:37:09	rollback	0 row(s) affected
4	03:37:27	desc Ciudad	2 row(s) returned

Vemos que el tamaño de **Nombre** no ha cambiado con **rollback**

Ejemplo 2: No se puede hacer rollback de las sentencias DCL

Query 1 X

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • create user 'director'@'localhost'
   identified by 'vistaalegre';
5
6
7 • rollback;
8
9 • select user, host
  from mysql.user
11   where user = 'director';
12
```

Result Grid Filter Rows: A Export: Wrap Cell Content: IA

#	user	host
1	director	localhost

user 15 X

Action Output ▼

#	Time	Action	Message
✓ 1	03:45:08	set autocommit = 0	0 row(s) affected
✓ 2	03:45:11	create user 'director'@'localhost' identified by 'vistaale...'	0 row(s) affected
✓ 3	03:45:14	rollback	0 row(s) affected
✓ 4	03:45:18	select user, host from mysql.user where user = 'direct...'	1 row(s) returned

Vemos que ejecutar rollback no elimina el usuario director@localhost

Consecuencias:

1. **No sirve de nada hacer rollback** inmediatamente **después de ejecutar sentencias DDL o DCL**
2. Cada sentencia **DDL o DCL** hace un **commit implícito** siempre
3. Si ejecutamos una sentencia **DDL o DCL después de** una sentencia **DML**, entonces **se ha hecho commit** de la sentencia DML y ya **no se puede hacer rollback** de esa sentencia DML

Ejemplo: Ejecutamos una sentencia DCL después de una sentencia DML y vemos que no podemos hacer rollback

Primero comprobamos primero que **Zaragoza** no está en la tabla **Ciudad**:

Query 1 X

File New Open Save Run Stop Cancel Reset Help Limit to 1000 rows More Star Copy Search Print Close

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • select *
5     from Ciudad;
```

Result Grid New Open Filter Rows: A Edit: Edit Insert Delete Update Export/Import: Export Import Wrap Cell Content: Wrap

#	CodCiudad	Nombre
1	1	Madrid
2	2	Barcelona
3	3	Sevilla
4	4	Málaga
5	6	Bilbao
6	7	Granada
7	8	Barcelona
8	9	Pamplona
9	10	Ciudad Real
10	11	Tarragona
*	NULL	NULL

Y comprobamos que con **autocommit deshabilitado**, si ejecutamos una **sentencia DCL (drop user)** ya **no podemos hacer rollback** de la sentencia DML (insert) y por tanto Zaragoza no desaparece de la tabla **Ciudad**

Query 1 X

The screenshot shows the MySQL Workbench interface. In the top-left, there's a code editor window titled "Query 1" containing the following SQL script:

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • insert into Ciudad (CodCiudad, Nombre)
5     values (12, 'Zaragoza');
6
7 • drop user 'director'@'localhost';
8
9 • rollback;
10
11 • select *
12     from Ciudad
13     where CodCiudad = 12;
```

Below the code editor is a "Result Grid" table with two columns: "#", "CodCiudad" and "Nombre". It contains one row with values 1 and Zaragoza.

#	CodCiudad	Nombre
1	12	Zaragoza

At the bottom of the interface is an "Action Output" table showing the log of actions:

#	Time	Action	Message
1	11:10:16	set autocommit = 0	0 row(s) affected
2	11:11:08	insert into Ciudad (CodCiudad, Nombre) values (12, 'Zaragoza')	1 row(s) affected
3	11:11:30	drop user 'director'@'localhost'	0 row(s) affected
4	11:12:13	rollback	0 row(s) affected
5	11:12:33	select * from Ciudad where CodCiudad = 12 LIMIT 0, 1000	1 row(s) returned

Diferencias entre DELETE y TRUNCATE TABLE

- **DELETE:** es una sentencia **DML**, permite **cláusula where** y permite **rollback**
- **TRUNCATE TABLE:** es una sentencia **DDL**, **no** permite **cláusula where** y **no** permite **rollback** (como es DDL hace commit implícito)

Ejemplo 1: Vemos que podemos hacer un DELETE de una fila de la tabla Ciudad usando where y que si hacemos rollback se deshace la eliminación de la fila

The screenshot shows the MySQL Workbench interface. In the top-left, there's a 'Query 1' tab with a red error icon. Below it is a toolbar with various icons. A dropdown menu says 'Limit to 1000 rows'. The main area contains the following SQL code:

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • delete from Ciudad
5     where CodCiudad = 12;
6
7 • rollback;
8
9 • select *
10    from Ciudad
11   where CodCiudad = 12;
12
```

Below the code is a 'Result Grid' section with a table:

#	CodCiudad	Nombre
1	12	Zaragoza
*	HULL	HULL

At the bottom, there's an 'Action Output' section with a table:

#	Time	Action	Message
1	11:25:54	set autocommit = 0	0 row(s) affected
2	11:26:25	delete from Ciudad where CodCiudad = 12	1 row(s) affected
3	11:26:39	rollback	0 row(s) affected
4	11:26:59	select * from Ciudad where CodCiudad = 12 LIMIT 0, 1000	1 row(s) returned

Ejemplo 2: Vemos que no podemos hacer un TRUNCATE con cláusula WHERE

The screenshot shows the MySQL Workbench interface. In the top-left, there's a 'Query 1' tab with a red error icon. Below it is a toolbar with various icons. A dropdown menu says 'Limit to 1000 rows'. The main area contains the following SQL code:

```
1 • use centro_educativo;
2 • set autocommit = OFF;
3
4 • truncate table Ciudad
5 •     where CodCiudad = 12;
6
```

At the bottom, there's an 'Output' section with an 'Action Output' table:

#	Time	Action	Message
1	20:47:16	truncate table Ciudad where CodCiudad = 12	Error Code: 1064. You have an error in your SQL syntax; c...

Ejemplo 3: Vemos que no sirve de nada hacer ROLLBACK después de una sentencia TRUNCATE TABLE

Primero hacemos una copia de la tabla Ciudad y vemos cuántas filas tiene:

Query 1

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • create table Copia_Ciudad as (select * from Ciudad);
5
6 • select count(*)
7     from Copia_Ciudad;
8
```

Result Grid

#	count(*)
1	11

Action Output

#	Time	Action	Message
1	11:44:20	create table Copia_Ciudad as (select * from Ciudad)	11 row(s) affected Records: 11 Duplicates: 0 Warnings: 0
2	11:44:41	select count(*) from Copia_Ciudad LIMIT 0, 1000	1 row(s) returned

A continuación, con el autocommit deshabilitado intentamos hacer **rollback** del TRUNCATE TABLE pero vemos que **no funciona**:

Query 1

```
1 • use centro_educativo;
2 • set autocommit = 0;
3
4 • truncate table Copia_Ciudad;
5
6 • rollback;
7
8 • select count(*)
9     from Copia_Ciudad;
10
```

Result Grid

#	count(*)
1	0

Action Output

#	Time	Action	Message
1	11:46:56	set autocommit = 0	0 row(s) affected
2	11:47:13	truncate table Copia_Ciudad	0 row(s) affected
3	11:47:28	rollback	0 row(s) affected
4	11:47:33	select count(*) from Copia_Ciudad LIMIT 0, 1000	1 row(s) returned

5. Políticas de bloqueo

Pueden suceder problemas de concurrencia en el acceso a datos cuando dos transacciones distintas acceden al mismo dato (misma fila de la misma tabla).

Estos problemas están contemplados por el estándar SQL y son los siguientes:

- **Lectura Sucia:** cuando una transacción está leyendo datos de otra transacción que no ha hecho commit
- **Lectura No Repetible:** cuando una transacción intenta volver a leer datos que leyó antes pero han sido modificados y validados por otra transacción
- **Lectura Fantasma:** cuando una transacción está leyendo datos que no existían cuando se inició la transacción (porque han sido creados y validados por otra transacción)

Nota: la diferencia entre la **Lectura No Repetible** y la **Lectura Fantasma** es que si se lanza la misma consulta dos veces en una misma transacción:

- La **Lectura No Repetible** devuelve el **mismo número de filas** pero con datos diferentes
- La **Lectura Fantasma** devuelve **más filas** pero los datos de las filas no nuevas no han cambiado

Interbloqueos (deadlocks)

Puede ocurrir que dos transacciones se interbloqueen: ambas esperan a que la otra termine para poder terminar.

El SGBD tiene que detectar los interbloqueos y abortar una de las dos transacciones (hacer rollback) para deshacer el interbloqueo.

Niveles de aislamiento

Para evitar estos problemas el SGBD podrá bloquear conjuntos de datos aplicando uno de los siguientes cuatro niveles de aislamiento:

1. **Lectura no acometida:** no se realiza ningún bloqueo y por tanto las transacciones concurrentes podrán tener los tres tipos de problemas.
2. **Lectura acometida:** una transacción no podrá leer datos no validados por otras transacciones. Se evita la Lectura Sucia, pero no se evitan la Lectura No Repetible ni la Lectura Fantasma.
3. **Lectura repetible:** una transacción no podrá leer datos diferentes a los que ha leído anteriormente. Se evitan la Lectura Sucia y la Lectura No Repetible pero no se evita la Lectura Fantasma.

4. **Serialización**: cada transacción bloquea todas las filas modificadas y no las libera hasta que finalice la transacción. Las demás transacciones tendrán que esperar a que esa transacción termine.

Nota: En MySQL el nivel **Lectura repetible** tampoco permite la **Lectura Fantasma**

La diferencia entre el nivel **Lectura repetible** y el nivel **Serializable** en MySQL es:

- En el nivel **Lectura repetible** no se bloquean las filas que afectan a las lecturas de otra transacción
- En el nivel **Serializable** se bloquean las filas que afectan a las lecturas de otra transacción

Por defecto MySQL tiene establecido el nivel **Lectura repetible**:

The screenshot shows the MySQL Workbench interface with a single query window titled "Query 1". The query is:

```
1 • show variables like 'transaction_isolation';  
2
```

The results grid shows one row:

#	Variable_name	Value
1	transaction_isolation	REPEATABLE-READ

Pero se puede cambiar a cualquiera de los otros tres niveles usando el comando **SET TRANSACTION ISOLATION LEVEL**:

Ejemplo 1: el usuario **root** puede establecer el nivel de aislamiento a **Lectura no acometida** para la sesión actual

The screenshot shows the MySQL Workbench interface with two windows: "Query 1" and "Result 7".

Query 1:

```
1 • set session transaction isolation level read uncommitted;  
2  
3 • show variables like 'transaction_isolation';  
4
```

Result Grid:

#	Variable_name	Value
1	transaction_isolation	READ-UNCOMMITTED

Result 7:

Action Output

#	Time	Action	Message
1	13:28:36	set session transaction isolation level read uncommitted	0 row(s) affected
2	13:28:39	show variables like 'transaction_isolation'	1 row(s) returned

Ejemplo 2: el usuario **root** puede establecer el nivel de aislamiento a **Lectura acometida** para la sesión actual

Query 1 x

1 • **set session transaction isolation level read committed;**
2
3 • **show variables like 'transaction_isolation';**
4

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

#	Variable_name	Value
1	transaction_isolation	READ-COMMITTED

Result 8 x

Action Output Action

#	Time	Action	Message
1	13:30:12	set session transaction isolation level read committed	0 row(s) affected
2	13:30:14	show variables like 'transaction_isolation'	1 row(s) returned

Ejemplo 3: el usuario **root** puede establecer el nivel de aislamiento a **Serializable** para la sesión actual

Query 1 x

1 • **set session transaction isolation level serializable;**
2
3 • **show variables like 'transaction_isolation';**
4

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

#	Variable_name	Value
1	transaction_isolation	SERIALIZABLE

Result 9 x

Action Output Action

#	Time	Action	Message
1	13:32:01	set session transaction isolation level serializable	0 row(s) affected
2	13:32:03	show variables like 'transaction_isolation'	1 row(s) returned

Ejemplo 4: el usuario **root** puede restablecer el nivel de aislamiento al nivel por defecto **Lectura repetible** para la sesión actual

Query 1 ✖

```
1 • set session transaction isolation level repeatable read;
2
3 • show variables like 'transaction_isolation';
```

Result Grid Filter Rows: A Export: Wrap Cell Content: T

#	Variable_name	Value
1	transaction_isolation	REPEATABLE-READ

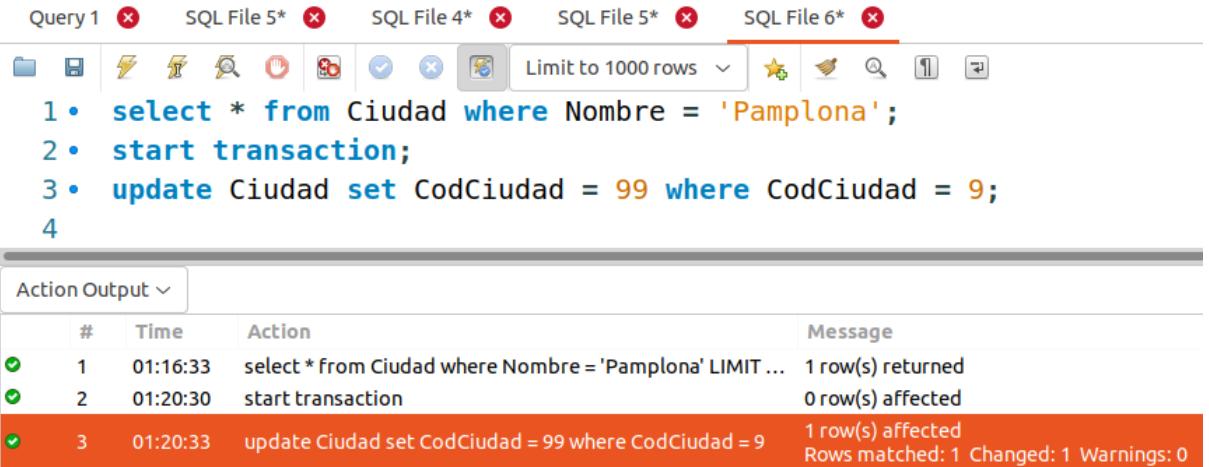
Result 11 ✖

Action Output ▼

#	Time	Action	Message
1	13:39:22	set session transaction isolation level repeatable read	0 row(s) affected
2	13:39:24	show variables like 'transaction_isolation'	1 row(s) returned

Ejemplo 5.1: lectura sucia

Abrimos en Workbench una ventana e iniciamos una transacción:



The screenshot shows the MySQL Workbench interface with multiple tabs at the top. The 'Query 1' tab is active, containing the following SQL code:

```
1 • select * from Ciudad where Nombre = 'Pamplona';
2 • start transaction;
3 • update Ciudad set CodCiudad = 99 where CodCiudad = 9;
4
```

Below the code, the 'Action Output' pane displays the execution log:

#	Time	Action	Message
1	01:16:33	select * from Ciudad where Nombre = 'Pamplona' LIMIT ...	1 row(s) returned
2	01:20:30	start transaction	0 row(s) affected
3	01:20:33	update Ciudad set CodCiudad = 99 where CodCiudad = 9	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

Abrimos una sesión desde el símbolo del sistema y fijamos el nivel para **lectura no acometida** y leemos en la tabla modificada en la transacción de la otra sesión:

```
mysql>
mysql> use centro_educativo
Database changed
mysql>
mysql> set session transaction isolation level read uncommitted;
Query OK, 0 rows affected (0,00 sec)

mysql> select * from Ciudad where Nombre = 'Pamplona';
+-----+-----+
| CodCiudad | Nombre   |
+-----+-----+
|      99 | Pamplona |
+-----+-----+
1 row in set (0,00 sec)

mysql>
mysql>
```

Hemos hecho una **lectura sucia** (porque la transacción en Workbench no se ha terminado aún)

Terminamos la transacción en Workbench con **rollback**:

The screenshot shows the MySQL Workbench interface with multiple tabs at the top: Query 1, SQL File 5*, SQL File 4*, SQL File 5*, and SQL File 6*. The SQL File 6* tab is selected and highlighted with a red border. Below the tabs is a toolbar with various icons. The main area contains the following SQL code:

```
1 • select * from Ciudad where Nombre = 'Pamplona';
2 • start transaction;
3 • update Ciudad set CodCiudad = 99 where CodCiudad = 9;
4
5 • rollback;
```

Below the code is a table titled "Action Output" with columns: #, Time, Action, and Message. The entries are:

#	Time	Action	Message
1	01:16:33	select * from Ciudad where Nombre = 'Pamplona' LIMIT ...	1 row(s) returned
2	01:20:30	start transaction	0 row(s) affected
3	01:20:33	update Ciudad set CodCiudad = 99 where CodCiudad = 9	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
4	01:24:00	rollback	0 row(s) affected

Volvemos al símbolo del sistema y vemos que si volvemos a leer la fila afectada cambia el resultado:

```
mysql>
mysql> use centro_educativo
Database changed
mysql>
mysql> set session transaction isolation level read  uncommitted;
Query OK, 0 rows affected (0,00 sec)

mysql> select * from Ciudad where Nombre = 'Pamplona';
+-----+-----+
| CodCiudad | Nombre   |
+-----+-----+
|      99 | Pamplona |
+-----+-----+
1 row in set (0,00 sec)

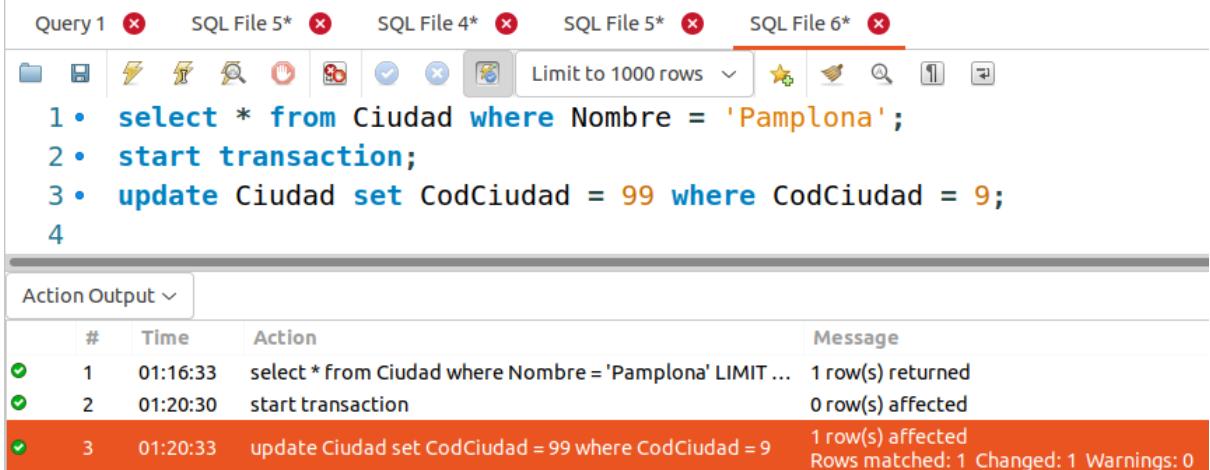
mysql>
mysql>
mysql> select * from Ciudad where Nombre = 'Pamplona';
+-----+-----+
| CodCiudad | Nombre   |
+-----+-----+
|        9 | Pamplona |
+-----+-----+
1 row in set (0,00 sec)

mysql>
mysql>
```

Ejemplo 5.2: evitamos la lectura sucia

Repetimos el Ejemplo 5.1 pero ahora con el **nivel de lectura acometida** para evitar la lectura sucia

Abrimos en Workbench una ventana e iniciamos una transacción:



The screenshot shows the MySQL Workbench interface with multiple tabs at the top. The 'SQL File 6*' tab is selected, indicated by a red underline. Below the tabs is a toolbar with various icons. The main area contains four numbered SQL statements:

- 1 • `select * from Ciudad where Nombre = 'Pamplona';`
- 2 • `start transaction;`
- 3 • `update Ciudad set CodCiudad = 99 where CodCiudad = 9;`
- 4

Below the statements is a table titled 'Action Output' with three rows of log entries:

#	Time	Action	Message
1	01:16:33	<code>select * from Ciudad where Nombre = 'Pamplona' LIMIT ...</code>	1 row(s) returned
2	01:20:30	<code>start transaction</code>	0 row(s) affected
3	01:20:33	<code>update Ciudad set CodCiudad = 99 where CodCiudad = 9</code>	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

Abrimos una sesión desde el símbolo del sistema y fijamos el modo para **lectura acometida** y leemos en la tabla modificada en la transacción de la otra sesión:

```
mysql>
mysql> use centro_educativo
Database changed
mysql>
mysql> set session transaction isolation level read committed;
Query OK, 0 rows affected (0,00 sec)

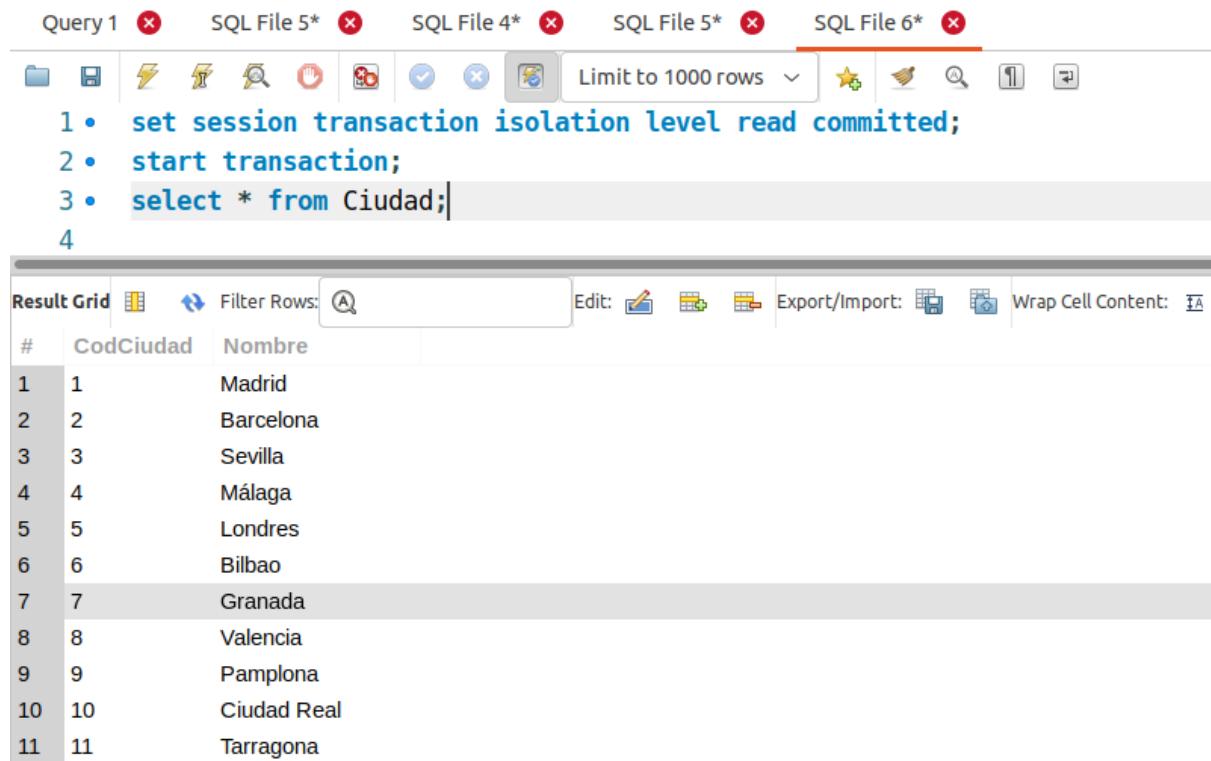
mysql> select * from Ciudad where Nombre = 'Pamplona';
+-----+-----+
| CodCiudad | Nombre   |
+-----+-----+
|      9    | Pamplona |
+-----+-----+
1 row in set (0,00 sec)

mysql>
mysql>
```

Vemos que **ya no hemos hecho una lectura sucia** (vemos el resultado anterior al inicio de la transacción en la otra sesión)

Ejemplo 6.1: lectura no repetible

Realizamos una lectura después de iniciar una transacción en nivel de **lectura acometida**:



The screenshot shows the MySQL Workbench interface. At the top, there are tabs for 'Query 1', 'SQL File 5*', 'SQL File 4*', 'SQL File 5*', and 'SQL File 6*'. The 'SQL File 6*' tab is active. Below the tabs is a toolbar with various icons. A dropdown menu says 'Limit to 1000 rows'. The main area contains the following SQL code:

```
1 • set session transaction isolation level read committed;
2 • start transaction;
3 • select * from Ciudad;
4
```

Below the code is a 'Result Grid' table with the following data:

#	CodCiudad	Nombre
1	1	Madrid
2	2	Barcelona
3	3	Sevilla
4	4	Málaga
5	5	Londres
6	6	Bilbao
7	7	Granada
8	8	Valencia
9	9	Pamplona
10	10	Ciudad Real
11	11	Tarragona

Abrimos una sesión desde el símbolo del sistema, e iniciamos y validamos una transacción:

```
mysql>
mysql> use centro_educativo
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0,00 sec)

mysql> update Ciudad set Nombre = 'San Sebastián' where CodCiudad = 6;
Query OK, 1 row affected (0,00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0,00 sec)

mysql>
```

Repetimos la lectura en Workbench y vemos que no podemos repetir la lectura (ya no aparece Bilbao)

The screenshot shows the MySQL Workbench interface. At the top, there are five tabs: 'Query 1', 'SQL File 5*', 'SQL File 4*', 'SQL File 5*', and 'SQL File 6*'. The fifth tab is highlighted with a red border. Below the tabs is a toolbar with various icons for file operations, search, and database management. A dropdown menu 'Limit to 1000 rows' is open. The main area contains the following SQL code:

```
1 • set session transaction isolation level read committed;
2 • start transaction;
3 • select * from Ciudad;
4 • select * from Ciudad;|
```

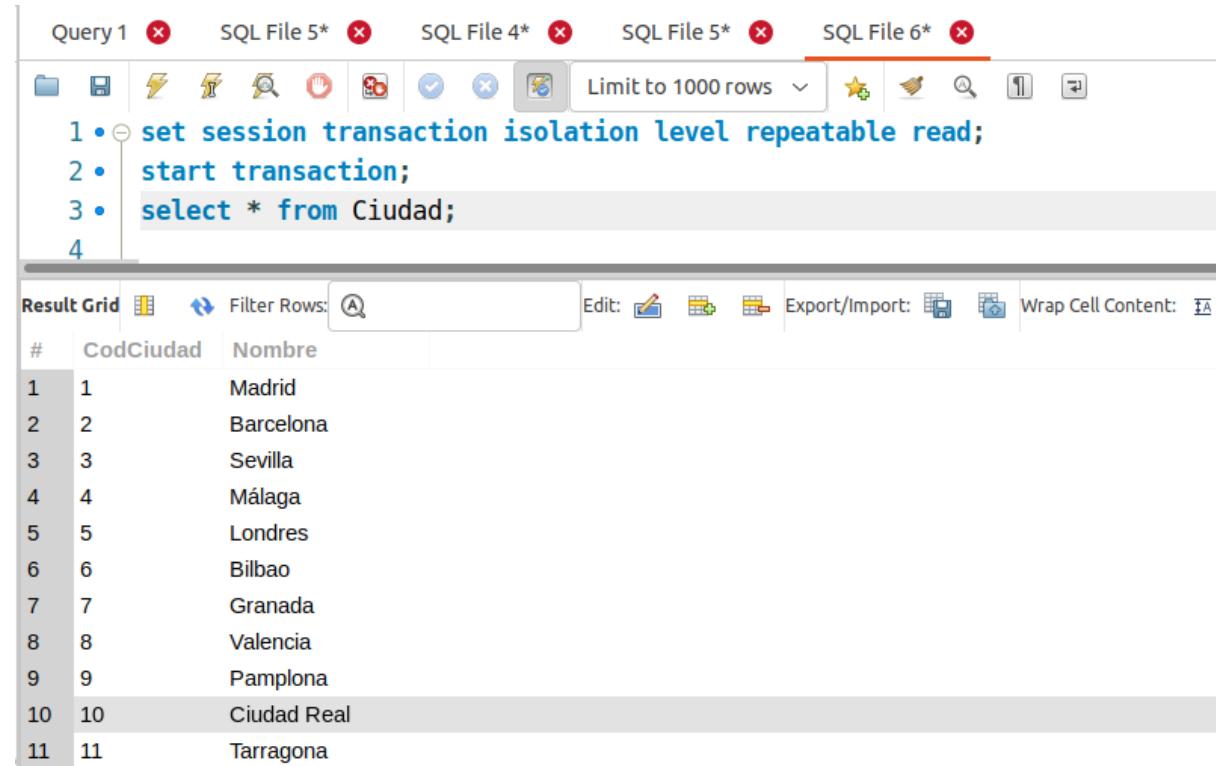
The number '5' is also present below the fourth line. Below the code is a results grid titled 'Result Grid' with columns '#', 'CodCiudad', and 'Nombre'. The data is as follows:

#	CodCiudad	Nombre
1	1	Madrid
2	2	Barcelona
3	3	Sevilla
4	4	Málaga
5	5	Londres
6	6	San Sebastián
7	7	Granada
8	8	Valencia
9	9	Pamplona
10	10	Ciudad Real
11	11	Tarragona

Ejemplo 6.2: evitamos la lectura no repetible

Repetimos el Ejemplo 6.1 pero ahora con el **nivel lectura repetible** para evitar la lectura no repetible

Realizamos en una sesión en Workbench una lectura después de iniciar una transacción en nivel de **lectura repetible**:



The screenshot shows the MySQL Workbench interface with a session titled "SQL File 6*". The session contains the following SQL code:

```
1 • set session transaction isolation level repeatable read;
2 • start transaction;
3 • select * from Ciudad;
4
```

The result grid displays the following data:

#	CodCiudad	Nombre
1	1	Madrid
2	2	Barcelona
3	3	Sevilla
4	4	Málaga
5	5	Londres
6	6	Bilbao
7	7	Granada
8	8	Valencia
9	9	Pamplona
10	10	Ciudad Real
11	11	Tarragona

Abrimos una sesión desde el símbolo del sistema, e iniciamos y validamos una transacción:

```
mysql>
mysql> use centro_educativo
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0,00 sec)

mysql> update Ciudad set Nombre = 'San Sebastián' where CodCiudad = 6;
Query OK, 1 row affected (0,00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0,00 sec)

mysql>
```

Repetimos la lectura en la sesión en Workbench (vemos que **se evita la lectura no repetible**):

The screenshot shows the MySQL Workbench interface with multiple tabs at the top: Query 1, SQL File 5*, SQL File 4*, SQL File 5*, and SQL File 6*. The SQL File 6* tab is active. Below the tabs is a toolbar with various icons. A dropdown menu labeled "Limit to 1000 rows" is open. The main area contains the following SQL code:

```
1 • set session transaction isolation level repeatable read;
2 • start transaction;
3 • select * from Ciudad;
4 • select * from Ciudad;
```

The number 5 is visible below the fourth line of code. Below the code is a "Result Grid" table with the following data:

#	CodCiudad	Nombre
1	1	Madrid
2	2	Barcelona
3	3	Sevilla
4	4	Málaga
5	5	Londres
6	6	Bilbao
7	7	Granada
8	8	Valencia
9	9	Pamplona
10	10	Ciudad Real
11	11	Tarragona

Por último, si finalizamos la transacción en la sesión Workbench vemos que ya se puede ver el dato modificado (ya aparece **San Sebastián**):

The screenshot shows the MySQL Workbench interface with the same tabs and toolbar as the previous screenshot. The SQL File 6* tab is active. The main area contains the following SQL code:

```
1 • set session transaction isolation level repeatable read;
2 • start transaction;
3 • select * from Ciudad;
4 • select * from Ciudad;
5 • rollback;
6 • select * from Ciudad;
```

The number 7 is visible below the sixth line of code. Below the code is a "Result Grid" table with the following data:

#	CodCiudad	Nombre
1	1	Madrid
2	2	Barcelona
3	3	Sevilla
4	4	Málaga
5	5	Londres
6	6	San Sebastián
7	7	Granada
8	8	Valencia
9	9	Pamplona
10	10	Ciudad Real
11	11	Tarragona

Ejemplo 7.1: lectura fantasma

En una sesión en Workbench iniciamos una transacción en nivel lectura acometida y realizamos una lectura:

The screenshot shows the MySQL Workbench interface with multiple tabs at the top: Query 1, SQL File 5*, SQL File 4*, SQL File 5*, and SQL File 6*. The SQL File 6* tab is active, containing the following SQL code:

```
1 •  set session transaction isolation level read committed;
2 •  start transaction;
3 •  select * from Ciudad;
4
```

Below the code is a Result Grid table with the following data:

#	CodCiudad	Nombre
1	1	Madrid
2	2	Barcelona
3	3	Sevilla
4	4	Málaga
5	5	Londres
6	6	San Sebastián
7	7	Granada
8	8	Valencia
9	9	Pamplona
10	10	Ciudad Real
11	11	Tarragona

At the bottom, the Action Output section shows the following log entries:

#	Time	Action	Message
1	12:21:11	set session transaction isolation level read committed	0 row(s) affected
2	12:21:13	start transaction	0 row(s) affected
3	12:21:39	select * from Ciudad LIMIT 0, 1000	11 row(s) returned

Abrimos una sesión desde el símbolo del sistema, iniciamos una transacción e insertamos un nuevo datos:

```
mysql> use centro_educativo;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>
mysql> start transaction;
Query OK, 0 rows affected (0,00 sec)

mysql> insert into Ciudad values (12, 'Zaragoza');
Query OK, 1 row affected (0,00 sec)

mysql> commit;
Query OK, 0 rows affected (0,00 sec)

mysql>
```

Repetimos la lectura en Workbench y vemos que aparecen los datos introducidos en la otra transacción (aparece la fila de Zaragoza):

The screenshot shows the MySQL Workbench interface with the following details:

- Query List:** Shows tabs for "Query 1" and several "SQL File" tabs (5*, 4*, 5*, 6*).
- Toolbar:** Includes icons for file operations, search, and database management.
- SQL Editor:** Contains the following SQL code:

```
1 • set session transaction isolation level read committed;
2 • start transaction;
3 • select * from Ciudad;
4 • select * from Ciudad;
```
- Result Grid:** Displays a table with columns "#", "CodCiudad", and "Nombre". The data is:

#	CodCiudad	Nombre
1	1	Madrid
2	2	Barcelona
3	3	Sevilla
4	4	Málaga
5	5	Londres
6	6	San Sebastián
7	7	Granada
8	8	Valencia
9	9	Pamplona
10	10	Ciudad Real
11	11	Tarragona
12	12	Zaragoza
- Action Output:** Shows the session history:

#	Time	Action	Message
1	12:21:11	set session transaction isolation level read committed	0 row(s) affected
2	12:21:13	start transaction	0 row(s) affected
3	12:21:39	select * from Ciudad LIMIT 0, 1000	11 row(s) returned
4	12:24:47	select * from Ciudad LIMIT 0, 1000	12 row(s) returned

Ejemplo 7.2: evitamos la lectura fantasma

Repetimos el Ejemplo 7.1 pero ahora con el **nivel lectura repetible** para evitar la lectura fantasma

En una sesión en Workbench iniciamos una transacción en nivel **lectura repetible** y realizamos una lectura:

The screenshot shows the MySQL Workbench interface. At the top, there are tabs for Query 1, SQL File 5*, SQL File 4*, SQL File 5*, and SQL File 6*. The SQL File 6* tab is active, containing the following SQL code:

```
1 • 1 set session transaction isolation level repeatable read;
2 • 2 start transaction;
3 • 3 select * from Ciudad;
```

Below the code is a Result Grid showing the following data:

#	CodCiudad	Nombre
1	1	Madrid
2	2	Barcelona
3	3	Sevilla
4	4	Málaga
5	5	Londres
6	6	San Sebastián
7	7	Granada
8	8	Valencia
9	9	Pamplona
10	10	Ciudad Real
11	11	Tarragona
*	NULL	NULL

At the bottom, the Action Output pane shows the following log entries:

#	Time	Action	Message
1	12:38:20	set session transaction isolation level repeatable read	0 row(s) affected
2	12:38:22	start transaction	0 row(s) affected
3	12:38:25	select * from Ciudad LIMIT 0, 1000	11 row(s) returned

Abrimos una sesión desde el símbolo del sistema, iniciamos una transacción e insertamos nuevos datos:

```
mysql>
mysql> use centro_educativo
Database changed
mysql>
mysql> insert into Ciudad values (12,'Zaragoza');
Query OK, 1 row affected (0,00 sec)

mysql> commit;
Query OK, 0 rows affected (0,00 sec)

mysql>
```

Repetimos la lectura en Workbench y vemos que no aparece la fila introducida en la otra transacción (no aparece la fila de Zaragoza):

The screenshot shows the MySQL Workbench interface with multiple tabs at the top: Query 1, SQL File 5*, SQL File 4*, SQL File 5*, and SQL File 6*. The SQL File 6* tab is active, containing the following SQL code:

```
1 • 1 set session transaction isolation level repeatable read;
2 • 2 start transaction;
3 • 3 select * from Ciudad;
4 • 4 select * from Ciudad;|
```

The Result Grid below displays the following data:

#	CodCiudad	Nombre
1	1	Madrid
2	2	Barcelona
3	3	Sevilla
4	4	Málaga
5	5	Londres
6	6	San Sebastián
7	7	Granada
8	8	Valencia
9	9	Pamplona
10	10	Ciudad Real
11	11	Tarragona
*	NULL	NULL

The Action Output section shows the following log entries:

#	Time	Action	Message
1	12:38:20	set session transaction isolation level repeatable read	0 row(s) affected
2	12:38:22	start transaction	0 row(s) affected
3	12:38:25	select * from Ciudad LIMIT 0, 1000	11 row(s) returned
4	12:40:24	select * from Ciudad LIMIT 0, 1000	11 row(s) returned

Ejemplo 8.1: interbloqueo

En una sesión en Workbench iniciamos una transacción y realizamos una lectura en nivel **Serializable** para **bloquear las filas de la consulta**

The screenshot shows the MySQL Workbench interface with several tabs at the top: Query 1, SQL File 5*, SQL File 4*, SQL File 5*, and SQL File 6*. The SQL File 6* tab is active, displaying the following SQL code:

```
1 • set session transaction isolation level serializable;
2 • start transaction;
3 • select * from Ciudad;
4
```

Below the code is a Result Grid table with columns #, CodCiudad, and Nombre. The data is as follows:

#	CodCiudad	Nombre
1	1	Madrid
2	2	Barcelona
3	3	Sevilla
4	4	Málaga
5	5	Londres
6	6	San Sebastián
7	7	Granada
8	8	Valencia
9	9	Pamplona
10	10	Ciudad Real
11	11	Tarragona
*	NULL	NULL

At the bottom, there is an Action Output section showing the following log entries:

#	Time	Action	Message
1	12:27:59	set session transaction isolation level serializable	0 row(s) affected
2	12:28:02	start transaction	0 row(s) affected
3	12:28:04	select * from Ciudad LIMIT 0, 400	11 row(s) returned

Abrimos una sesión desde el símbolo del sistema, iniciamos una transacción e insertamos nuevos datos:

```
mysql> use centro_educativo;
Database changed
mysql>
mysql> start transaction;
Query OK, 0 rows affected (0,00 sec)

mysql> insert into Ciudad values (12, 'Zaragoza');
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql>
```

Vemos que la fila que quiero insertar está bloqueada por la consulta en la transacción en nivel Serializable en Workbench. El SGBD detecta que es un **interbloqueo** y aborta la transacción (hace rollback).

Ejemplo 8.2: evitamos el interbloqueo terminando la transacción en nivel serializable antes de que el SGBD aborte la otra transacción

En una sesión en Workbench iniciamos una transacción en y realizamos una lectura en nivel **serializable** para **bloquear las filas de la consulta**

The screenshot shows the MySQL Workbench interface. At the top, there are tabs for 'Query 1' and several SQL files. Below the tabs is a toolbar with various icons. A dropdown menu says 'Limit to 400 rows'. The main area contains the following SQL code:

```
1 • set session transaction isolation level serializable;
2 • start transaction;
3 • select * from Ciudad;
```

Below the code is a 'Result Grid' showing the following data:

#	CodCiudad	Nombre
1	1	Madrid
2	2	Barcelona
3	3	Sevilla
4	4	Málaga
5	5	Londres
6	6	San Sebastián
7	7	Granada
8	8	Valencia
9	9	Pamplona
10	10	Ciudad Real
11	11	Tarragona
*	HULL	NULL

At the bottom, there is a 'Ciudad 3' tab and an 'Action Output' section with the following log:

#	Time	Action	Message
1	12:27:59	set session transaction isolation level serializable	0 row(s) affected
2	12:28:02	start transaction	0 row(s) affected
3	12:28:04	select * from Ciudad LIMIT 0, 400	11 row(s) returned

Abrimos una sesión desde el símbolo del sistema, iniciamos una transacción e insertamos nuevos datos:

```
mysql>
mysql>
mysql> use centro_educativo;
Database changed
mysql>
mysql> start transaction;
Query OK, 0 rows affected (0,00 sec)

mysql> insert into Ciudad values (12, 'Zaragoza');
```

Vemos que la tabla Ciudad está bloqueada por la transacción en nivel serializable en Workbench.

Hacemos rollback en Workbench de la transacción en nivel Serializable:

The screenshot shows the MySQL Workbench interface. At the top, there are tabs for 'Query 1' and several 'SQL File' tabs. Below the tabs is a toolbar with various icons. A dropdown menu says 'Limit to 1000 rows'. The main area contains the following SQL code:

```
1 •  set session transaction isolation level serializable;
2 •  start transaction;
3 •  select * from Ciudad;
4 •  rollback;
```

Below the code is a table titled 'Action Output' with the following data:

#	Time	Action	Message
1	12:57:56	set session transaction isolation level serializable	0 row(s) affected
2	12:57:58	start transaction	0 row(s) affected
3	12:58:01	select * from Ciudad LIMIT 0, 1000	11 row(s) returned
4	12:58:46	rollback	0 row(s) affected

Vemos que entonces el insert se desbloquea:

```
mysql>
mysql> use centro_educativo;
Database changed
mysql>
mysql> start transaction;
Query OK, 0 rows affected (0,00 sec)

mysql> insert into Ciudad values (12, 'Zaragoza');
Query OK, 1 row affected (17,89 sec)

mysql>
mysql> █
```