

MEMORIA PRÁCTICA 2 ABD



Gonzalo Puebla Holguín

David Ramos Archilla

Jesús Calzas Bedoya

19-12-2021

Administración y gestión de bases de datos

PARTE 1: CREACIÓN DE USUARIOS Y ASIGNACIÓN DE PRIVILEGIOS	3
Crear roles y usuarios.....	3
Visualización de permisos	5
Consultas.....	8
Eliminar permisos Santiago y Miriam	14
PARTE 2: CREACIÓN DE VISTAS Y ASIGNACIÓN DE PRIVILEGIOS	15
Con el usuario 'root'	15
Ejecución de operaciones	15
Creación de vistas	17
Creación de usuario suplencia y concesión de permisos.....	17
Con el usuario 'suplencia'	18
Ejecución de consultas tipo 'Obtener'	18
Ejecución de consultas tipo 'Insertar'	19
Comprobación de acceso a tablas	19
Comparativa de métodos.....	20
PARTE 3: TRANSACCIONES, CONEXIONES REMOTAS Y BACKUPS.....	20
1. Ejecución de 2 transacciones simultáneas:.....	20
A) Lectura – lectura:.....	21
B) Lectura – escritura:	22
C) Escritura – escritura:.....	23
2. Lectura sucia	25
3. Conectarse a la BD	27
4. Backup.....	28
MYSQL Workbench	28
Consola.....	31
5. Servidor externo	35
A) Lectura – Lectura:	35
B) Lectura- Escritura.....	36
C) Escritura – Escritura:.....	37
D) Lectura Sucia.....	38

PARTE 1: CREACIÓN DE USUARIOS Y ASIGNACIÓN DE PRIVILEGIOS

Crear roles y usuarios

Comenzaremos creando el rol gestor desde el usuario *root*

```
CREATE ROLE IF NOT EXISTS 'gestor'@'localhost';
```

Y le asignaremos los permisos mínimos indicados a las tablas. A diferencia del resto de roles, estos permisos de gestor vienen acompañados de “WITH GRANT OPTION”. Esto significa que puede replicar los permisos que se ven afectados, sobre los roles y usuarios que tenga acceso.

```
GRANT INSERT, DELETE, SELECT, UPDATE ON pracabd1.personas TO 'gestor'@'localhost' WITH GRANT OPTION;  
GRANT INSERT, DELETE, SELECT, UPDATE ON pracabd1.cursos TO 'gestor'@'localhost' WITH GRANT OPTION;  
GRANT INSERT, DELETE, SELECT, UPDATE ON pracabd1.matriculados TO 'gestor'@'localhost' WITH GRANT OPTION;
```

Los roles secretario y comercial deben ser creados por un gestor, en este caso, Javier será nuestro gestor. Procedemos a crearlo y asignarle tanto el rol como los permisos de este para que pueda realizarlo.

```
CREATE USER IF NOT EXISTS 'Javier'@'localhost' DEFAULT ROLE 'gestor'@'localhost';  
GRANT 'gestor'@'localhost' TO 'Javier'@'localhost';
```

Antes de seguir, debemos activar los roles

```
SET PERSIST activate_all_roles_on_login = ON;
```

Y debemos asignarle al rol gestor los permisos suficientes para crear roles y usuarios, además de poder asociarlos

```
GRANT CREATE USER ON *.* TO 'gestor'@'localhost';  
GRANT CREATE ROLE ON *.* TO 'gestor'@'localhost';  
GRANT ROLE_ADMIN ON *.* TO 'gestor'@'localhost';
```

GRANT ROLE_ADMIN le da los permisos suficientes como para poder asociar roles a usuarios sobre los que tenga permisos

Desde el usuario de Javier, creamos el rol comercial

```
CREATE ROLE IF NOT EXISTS 'comercial'@'localhost';
```

Y le asignamos al rol los permisos para poder leer todas las tablas, insertar y actualizar personas y matriculados

```
GRANT SELECT, INSERT, UPDATE ON pracabd1.personas TO 'comercial'@'localhost';
GRANT SELECT ON pracabd1.cursos TO 'comercial'@'localhost';
GRANT SELECT, INSERT, UPDATE ON pracabd1.matriculados TO 'comercial'@'localhost';
```

Creamos tanto el usuario “Paco” como “Santiago” asignándoles por defecto el rol de comercial, como comerciales que son. Además, les asignamos los permisos relacionados con su rol a cada uno.

```
CREATE USER IF NOT EXISTS 'Paco'@'localhost' DEFAULT ROLE 'comercial'@'localhost';
GRANT 'comercial'@'localhost' TO 'Paco'@'localhost';

CREATE USER IF NOT EXISTS 'Santiago'@'localhost' DEFAULT ROLE 'comercial'@'localhost';
GRANT 'comercial'@'localhost' TO 'Santiago'@'localhost';
```

Javier, además de crear el rol comercial y sus respectivos usuarios, también ha de crear el rol “secretario”.

```
CREATE ROLE IF NOT EXISTS 'secretario'@'localhost';
```

Y le asigna los permisos correspondientes al rol

```
GRANT SELECT ON pracabd1.personas TO 'secretario'@'localhost';
GRANT SELECT, INSERT ON pracabd1.cursos TO 'secretario'@'localhost';
GRANT SELECT, UPDATE (matriculado) ON pracabd1.matriculados TO 'secretario'@'localhost';
```

Javier crea los usuarios de Miriam y Maribel, quienes serán los usuarios que operen con el rol de secretario. Aprovechamos su creación para asignarle por defecto el rol que van a utilizar, y también le asignamos los permisos del rol

```
CREATE USER IF NOT EXISTS 'Maribel'@'localhost' DEFAULT ROLE 'secretario'@'localhost';
GRANT 'secretario'@'localhost' TO 'Maribel'@'localhost';

CREATE USER IF NOT EXISTS 'Miriam'@'localhost' DEFAULT ROLE 'secretario'@'localhost';
GRANT 'secretario'@'localhost' TO 'Miriam'@'localhost';
```

A diferencia del rol gestor, no hemos tenido que activar los roles ya que estos ya se encontraban activados de cuando lo hicimos con gestor.

Para todos los roles y usuarios, hemos creado además unos scripts que nos permiten borrarlos o quitarle algún permiso en particular por si en algún momento fuese necesario. Quitar los permisos solo es para los roles ya que no se puede quitar un permiso a un usuario en particular. Para quitarle los permisos a un usuario debemos quitarle los permisos del rol

Borrar el rol gestor

```
DROP ROLE gestor;
```

Quitar los permisos de gestor

```
REVOKE INSERT, DELETE, SELECT, UPDATE ON pracabd1.personas FROM 'gestor'@'localhost';
REVOKE INSERT, DELETE, SELECT, UPDATE ON pracabd1.cursos FROM 'gestor'@'localhost';
REVOKE INSERT, DELETE, SELECT, UPDATE ON pracabd1.matriculados FROM 'gestor'@'localhost';
REVOKE CREATE USER ON *.* FROM 'gestor'@'localhost';
REVOKE CREATE ROLE ON *.* FROM 'gestor'@'localhost';
REVOKE ROLE_ADMIN ON *.* FROM 'gestor'@'localhost';
```

Borrar el usuario de Javier

```
DROP USER 'Javier'@'localhost';
```

Le quitamos los permisos al usuario Javier

```
REVOKE 'gestor'@'localhost' FROM 'Javier'@'localhost';
```

Borrar el rol comercial

```
DROP ROLE 'comercial'@'localhost';
```

Quitar los permisos de comercial

```
REVOKE SELECT, INSERT, UPDATE ON pracabd1.personas FROM 'comercial'@'localhost';  
REVOKE SELECT ON pracabd1.cursos FROM 'comercial'@'localhost';  
REVOKE SELECT, INSERT, UPDATE ON pracabd1.matriculados FROM 'comercial'@'localhost';
```

Borrar los usuarios de Paco y Santiago

```
DROP USER 'Paco'@'localhost';  
DROP USER 'Santiago'@'localhost';
```

Quitar los permisos a los usuarios de Paco y Santiago

```
REVOKE 'comercial'@'localhost' FROM 'Santiago'@'localhost';  
REVOKE 'comercial'@'localhost' FROM 'Paco'@'localhost';
```

Borrar el rol de secretario

```
DROP ROLE 'secretario'@'localhost';
```

Quitar los permisos a secretario

```
REVOKE SELECT ON pracabd1.personas FROM 'secretario'@'localhost';  
REVOKE SELECT, INSERT ON pracabd1.cursos FROM 'secretario'@'localhost';  
REVOKE SELECT, UPDATE (matriculado) ON pracabd1.matriculados FROM 'secretario'@'localhost';
```

Borrar los usuarios de Miriam y Maribel

```
DROP USER 'Maribel'@'localhost';  
DROP USER 'Miriam'@'localhost';
```

Quitar los permisos a los usuarios de Miriam y Maribel

```
REVOKE 'secretario'@'localhost' FROM 'Miriam'@'localhost';  
REVOKE 'secretario'@'localhost' FROM 'Maribel'@'localhost';
```

Visualización de permisos

Comenzamos visualizando los permisos de forma manual. Root tiene todos los permisos, para que no estorben a la hora de visualizar los permisos de los usuarios y roles que hemos creado, hemos decidido que lo mejor es que no nos los imprima. Esto lo conseguimos gracias al “NOT LIKE ‘\root\’@‘\localhost\’”

Con la siguiente sentencia podremos observar los privilegios asociados a los roles

```
SELECT *  
FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES  
WHERE GRANTEE NOT LIKE '\root\'@'\localhost\';
```

Nos da como resultado

	GRANTEE	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	PRIVILEGE_TYPE	IS_GRANTABLE
►	'gestor'@'localhost'	def	pracabd1	personas	SELECT	YES
	'gestor'@'localhost'	def	pracabd1	personas	INSERT	YES
	'gestor'@'localhost'	def	pracabd1	personas	UPDATE	YES
	'gestor'@'localhost'	def	pracabd1	personas	DELETE	YES
	'mysql.session'@'localhost'	def	mysql	user	SELECT	NO
	'mysql.sys'@'localhost'	def	sys	sys_config	SELECT	NO
	'secretario'@'localhost'	def	pracabd1	cursos	SELECT	NO
	'secretario'@'localhost'	def	pracabd1	cursos	INSERT	NO
	'secretario'@'localhost'	def	pracabd1	matriculados	SELECT	NO
	'secretario'@'localhost'	def	pracabd1	personas	SELECT	NO
	'comercial'@'localhost'	def	pracabd1	personas	SELECT	NO
	'comercial'@'localhost'	def	pracabd1	personas	INSERT	NO
	'comercial'@'localhost'	def	pracabd1	personas	UPDATE	NO
	'comercial'@'localhost'	def	pracabd1	cursos	SELECT	NO
	'comercial'@'localhost'	def	pracabd1	matriculados	SELECT	NO
	'comercial'@'localhost'	def	pracabd1	matriculados	INSERT	NO
	'comercial'@'localhost'	def	pracabd1	matriculados	UPDATE	NO
	'comercial'@'localhost'	def	pracabd1	matriculados	DELETE	NO
	'gestor'@'localhost'	def	pracabd1	cursos	SELECT	YES
	'gestor'@'localhost'	def	pracabd1	cursos	INSERT	YES
	'gestor'@'localhost'	def	pracabd1	cursos	UPDATE	YES
	'gestor'@'localhost'	def	pracabd1	cursos	DELETE	YES
	'gestor'@'localhost'	def	pracabd1	matriculados	SELECT	YES
	'gestor'@'localhost'	def	pracabd1	matriculados	INSERT	YES
	'gestor'@'localhost'	def	pracabd1	matriculados	UPDATE	YES
	'gestor'@'localhost'	def	pracabd1	matriculados	DELETE	YES

Con la siguiente sentencia comprobamos los privilegios de los usuarios y de los roles relacionados con los usuarios

```
SELECT *
FROM INFORMATION_SCHEMA.USER_PRIVILEGES
WHERE GRANTEE NOT LIKE '\\'root\\'@'localhost\\';
```

Nos da como resultado

	GRANTEE	TABLE_CATALOG	PRIVILEGE_TYPE	IS_GRANTABLE
►	'Javier'@'localhost'	def	USAGE	NO
	'Maribel'@'localhost'	def	USAGE	NO
	'Miriam'@'localhost'	def	USAGE	NO
	'Paco'@'localhost'	def	USAGE	NO
	'Santiago'@'localhost'	def	USAGE	NO
	'comercial'@'localhost'	def	USAGE	NO
	'gestor'@'localhost'	def	CREATE USER	YES
	'gestor'@'localhost'	def	CREATE ROLE	YES
	'gestor'@'localhost'	def	ROLE_ADMIN	NO
	'mysql.infoschema'@'localhost'	def	SELECT	NO
	'mysql.infoschema'@'localhost'	def	SYSTEM_USER	NO
	'mysql.session'@'localhost'	def	SHUTDOWN	NO
	'mysql.session'@'localhost'	def	SUPER	NO
	'mysql.session'@'localhost'	def	BACKUP_ADMIN	NO

'mysql.session'@'localhost'	def	CLONE_ADMIN	NO
'mysql.session'@'localhost'	def	CONNECTION_...	NO
'mysql.session'@'localhost'	def	PERSIST_RO_V...	NO
'mysql.session'@'localhost'	def	SESSION_VARI...	NO
'mysql.session'@'localhost'	def	SYSTEM_USER	NO
'mysql.session'@'localhost'	def	SYSTEM_VARIA...	NO
'mysql.sys'@'localhost'	def	USAGE	NO
'mysql.sys'@'localhost'	def	SYSTEM_USER	NO
'secretario'@'localhost'	def	USAGE	NO

Hay algún rol (secretario) que tiene permisos sobre una columna de una tabla en particular, esos permisos los podemos visualizar con la siguiente sentencia

```
SELECT *
FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES
WHERE GRANTEE NOT LIKE '\\'root\\'@'localhost\\';
```

Nos da como resultado

GRANTEE	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	PRIVILEGE_TYPE	IS_GRANTABLE
'secretario'@'localhost'	def	pracabd1	matriculados	matriculado	UPDATE	NO

Todo ello lo podemos comprobar con la sentencia SHOW GRANTS FOR 'user_role' donde user_role indicaremos el rol

13 • SHOW GRANTS FOR 'gestor'@'localhost';

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
Grants for gestor@localhost			
GRANT CREATE USER, CREATE ROLE ON *.* TO `gestor`@`localhost` WITH GRANT OPTION			
GRANT ROLE_ADMIN ON *.* TO `gestor`@`localhost`			
GRANT SELECT, INSERT, UPDATE, DELETE ON `pracabd1`.`cursos` TO `gestor`@`localhost` WITH GRANT OPTION			
GRANT SELECT, INSERT, UPDATE, DELETE ON `pracabd1`.`matriculados` TO `gestor`@`localhost` WITH GRANT OPTION			
GRANT SELECT, INSERT, UPDATE, DELETE ON `pracabd1`.`personas` TO `gestor`@`localhost` WITH GRANT OPTION			

14 • SHOW GRANTS FOR 'secretario'@'localhost';

Result Grid	Filter Rows:	Export:	Wrap
Grants for secretario@localhost			
GRANT USAGE ON *.* TO `secretario`@`localh...			
GRANT SELECT, INSERT ON `pracabd1`.`curso...			
GRANT SELECT, UPDATE (`matriculado`) ON `p...			
GRANT SELECT ON `pracabd1`.`personas` TO...			

15 • SHOW GRANTS FOR 'comercial'@'localhost';

Result Grid	Filter Rows:	Export:	Wi
Grants for comercial@localhost			
GRANT USAGE ON *.* TO `comercial`@`localh...			
GRANT SELECT ON `pracabd1`.`cursos` TO `c...			
GRANT SELECT, INSERT, UPDATE ON `pracabd...			
GRANT SELECT, INSERT, UPDATE ON `pracabd...			

Consultas

A continuación, hemos realizado 4 consultas con Javier, el usuario relacionado con el rol gestor. Habrá 2 consultas con el resultado esperado

- Visualizar toda la tabla cursos

The screenshot shows the SQL Server Enterprise Manager interface. The query editor contains the following SQL code:

```
1 USE pracabd1;
2
3 SELECT *
4 FROM cursos;
5
6
```

The Results pane displays the following data:

curso_id	nombre	area	edicion
2	Curso2	Ciberseguridad	2015
3	Curso3	Ciberseguridad	2016
4	Curso4	Ciberseguridad	2011
5	Curso5	Desarrollo web	2011
6	Curso6	Big Data	2014

The Output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
71	18:22:30	Error loading schema content	Error Code: 1142 SELECT command denied to user 'Javier'@'localhost' for table 'user_variab...	
72	18:25:49	SELECT * FROM cursos	Error Code: 1046. No database selected Select the default DB to be used by double-clicking ...	0.000 sec
73	18:25:51	USE pracabd1	0 row(s) affected	0.000 sec
74	18:25:54	SELECT * FROM cursos	9999 row(s) returned	0.016 sec / 0.015 sec

- Seleccionar todas las personas cuya provincia sea Lugo

The screenshot shows the SQL Server Enterprise Manager interface. The query editor contains the following SQL code:

```
1 USE pracabd1;
2
3 SELECT *
4 FROM cursos;
5
6
7 SELECT *
8 FROM pracabd1.personas
9 WHERE pracabd1.personas.provincia LIKE "Lugo";
```

The Results pane displays the following data:

persona_id	dni	nombre	apellidos	genero	direccion	localidad	provincia	cod_postal	telefono	email
2815	99183003W	Luis	del Horno Poveda	H	CALLE DEL DIALOGO	VALADOURO, O	LUGO	27063	995782070	LuisdelHornoPoveda1@gmail.com
2874	99402823R	Fidas	Guzmán Cruelo	H	CALLE DE POBLACION DE CAMPOS	MEIRA	LUGO	27029	990582451	FidasGuzmánCruelo@gmail.com
3066	99800684P	Vicente	Rosado Frutos	H	TRAVESIA DE BRINGAS	NAVIA DE SUARNA	LUGO	27034		VicenteRosadoFrutos4@gmail.com
9134	99131937W	Laura	de la Iglesia Piñonero	M	CALLE DE LORETO Y CHICOTE	TRIACASTELA	LUGO	27062	996846710	LauradelatIglesiaPiñonero@gmail.com

The Output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
72	18:25:49	SELECT * FROM cursos	Error Code: 1046. No database selected Select the default DB to be used by double-clicking ...	0.000 sec
73	18:25:51	USE pracabd1	0 row(s) affected	0.000 sec
74	18:25:54	SELECT * FROM cursos	9999 row(s) returned	0.016 sec / 0.015 sec
75	18:27:01	SELECT * FROM pracabd1.personas WHERE pracabd1.personas.provincia LIKE "Lugo"	20 row(s) returned	0.203 sec / 0.000 sec

Y 2 consultas que no podrá realizar debido a que no tiene los permisos necesarios para ello

Visualizar todos los permisos relacionados con los usuarios creados


```
1 • USE pracabd1;
2
3 • SELECT *
4   FROM cursos;
5
6
7 • SELECT *
8   FROM pracabd1.personas
9   WHERE pracabd1.personas.provincia LIKE "Lugo";
10
11 • SELECT *
12   FROM mysql.user;
```

#	Time	Action	Message	Duration / Fetch
73	18:25:51	USE pracabd1	0 row(s) affected	0.000 sec
74	18:25:54	SELECT * FROM cursos	9999 row(s) returned	0.016 sec / 0.015 sec
75	18:27:01	SELECT * FROM pracabd1.personas WHERE pracabd1.personas.provincia LIKE "Lugo"	20 row(s) returned	0.203 sec / 0.000 sec
76	18:27:30	SELECT * FROM mysql.user	Error Code: 1142: SELECT command denied to user 'Javier'@'localhost' for table 'user'	0.016 sec

Crear un nuevo esquema

```
1 • USE pracabd1;
2
3 • SELECT *
4   FROM cursos;
5
6
7 • SELECT *
8   FROM pracabd1.personas
9   WHERE pracabd1.personas.provincia LIKE "Lugo";
10
11 • SELECT *
12   FROM mysql.user;
13
14 • CREATE SCHEMA aux;
15
```

#	Time	Action	Message	Duration / Fetch
74	18:25:54	SELECT * FROM cursos	9999 row(s) returned	0.016 sec / 0.015 sec
75	18:27:01	SELECT * FROM pracabd1.personas WHERE pracabd1.personas.provincia LIKE "Lugo"	20 row(s) returned	0.203 sec / 0.000 sec
76	18:27:30	SELECT * FROM mysql.user	Error Code: 1142: SELECT command denied to user 'Javier'@'localhost' for table 'user'	0.016 sec
77	18:27:57	CREATE SCHEMA aux	Error Code: 1044: Access denied for user 'Javier'@'localhost' to database 'aux'	0.000 sec

Paco tiene relacionado el perfil de comercial, por lo que solo podrá hacer consultas con los permisos que tiene asociados.

Las 2 consultas que puede realizar son:

- Visualizar toda la información de la tabla personas

Paco x

Server Tools Scripting Help

consultas_paco

```

1 • use pracabd1;
2
3 • SELECT *
4   FROM personas;
5
6

```

person_id	dni	nombre	apellidos	genero	direccion	localidad	provincia	cod_postal	telefono	email
1	99383036W	Mario	Dieza Castillo	H	CALLE DE POZOHALCON	MADRID	MADRID	28053	990257168	MarioDiezaCastillo2@gmail.com
2	99317727A	Guillermo	Heredia	H	CALLE DE LOS GALLEGOS	GETAFE	MADRID	28904		Guillermoheredia1@yahoo.com
3	99309930Y	Miguel Angel	Olivares Cuenca	H	CALLE DE LA PEÑA DEL VELADO	MADRID	MADRID	28023		MiguelAngelOlivaresCuenca4@gmail.com
4	99462506Y	Micela	Noria Conde	M	CALLE DE VERDE VIENTO	MADRID	MADRID	28021		MicelaNoriaConde4@gmail.com
5	99470303X	María Antonia	Romero Preciado	M	CALLE DE AURELIO DE LA TORRE	MADRID	MADRID	28023		MaríaAntoniaRomeroPreciado4@gmail.com

personas 1 x

Output

#	Time	Action	Message	Duration / Fetch
1	18:29:33	Error loading schema content	Error Code: 1142 SELECT command denied to user 'Paco'@'localhost' for table 'user_variables...	
2	18:29:46	use pracabd1	0 row(s) affected	0.000 sec
3	18:29:54	SELECT * FROM personas	87032 row(s) returned	0.000 sec / 0.516 sec

- Visualizar toda la información de la tabla matriculados cuya persona tenga el persona_id=2

Paco x

Server Tools Scripting Help

consultas_paco

```

1 • use pracabd1;
2
3 • SELECT *
4   FROM personas;
5
6 • SELECT *
7   FROM matriculados
8  WHERE persona_id = 2;

```

person_id	curso_id	matriculado	comentarios
2	1240	1	Lorem ipsum dolor sit amet, consectetur adipiscing...

matriculados 3 x

Output

#	Time	Action	Message	Duration / Fetch
2	18:29:46	use pracabd1	0 row(s) affected	0.000 sec
3	18:29:54	SELECT * FROM personas	87032 row(s) returned	0.000 sec / 0.516 sec
4	18:32:20	SELECT * FROM matriculados WHERE persona_id = 1	1 row(s) returned	0.187 sec / 0.000 sec
5	18:32:29	SELECT * FROM matriculados WHERE persona_id = 2	1 row(s) returned	0.172 sec / 0.000 sec

Pero no tiene permisos para realizar cualquier sentencia

- Borrar la base de datos pracabd1

```
1 • use pracabd1;
2
3 • SELECT *
4   FROM personas;
5
6 • SELECT *
7   FROM matriculados
8   WHERE persona_id = 2;
9
10 • DROP SCHEMA pracabd1;
```

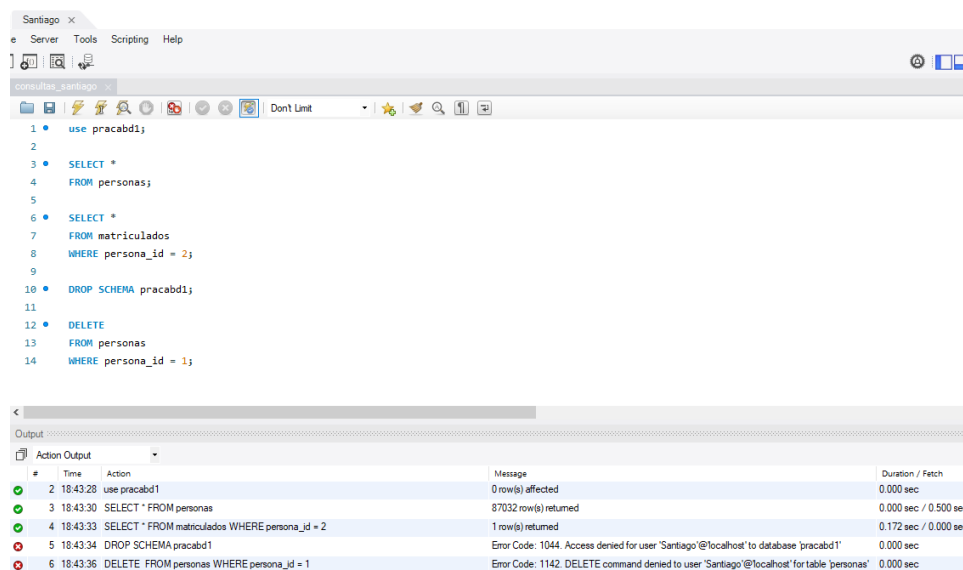
#	Time	Action	Message	Duration / Fetch
✓ 3	18:29:54	SELECT * FROM personas	87032 row(s) returned	0.000 sec / 0.516 sec
✓ 4	18:32:20	SELECT * FROM matriculados WHERE persona_id = 1	1 row(s) returned	0.187 sec / 0.000 sec
✓ 5	18:32:29	SELECT * FROM matriculados WHERE persona_id = 2	1 row(s) returned	0.172 sec / 0.000 sec
✗ 6	18:36:49	DROP SCHEMA pracabd1	Error Code: 1044. Access denied for user 'Paco'@'localhost' to database 'pracabd1'	0.000 sec

- Borrar la información relacionada con la persona cuyo id sea 1

```
1 • use pracabd1;
2
3 • SELECT *
4   FROM personas;
5
6 • SELECT *
7   FROM matriculados
8   WHERE persona_id = 2;
9
10 • DROP SCHEMA pracabd1;
11
12 • DELETE
13   FROM personas
14   WHERE persona_id = 1;
```

#	Time	Action	Message	Duration / Fetch
✓ 4	18:32:20	SELECT * FROM matriculados WHERE persona_id = 1	1 row(s) returned	0.187 sec / 0.000 sec
✓ 5	18:32:29	SELECT * FROM matriculados WHERE persona_id = 2	1 row(s) returned	0.172 sec / 0.000 sec
✗ 6	18:36:49	DROP SCHEMA pracabd1	Error Code: 1044. Access denied for user 'Paco'@'localhost' to database 'pracabd1'	0.000 sec
✗ 7	18:39:10	DELETE FROM personas WHERE persona_id = 1	Error Code: 1142. DELETE command denied to user 'Paco'@'localhost' for table 'personas'	0.000 sec

Santiago también tiene asociado el perfil de comercial, por lo que tiene los mismos permisos que Paco, lo podemos comprobar ejecutando las mismas sentencias.



```

1 • use pracabd1;
2
3 • SELECT *
4   FROM personas;
5
6 • SELECT *
7   FROM matriculados
8   WHERE persona_id = 2;
9
10 • DROP SCHEMA pracabd1;
11
12 • DELETE
13   FROM personas
14   WHERE persona_id = 1;

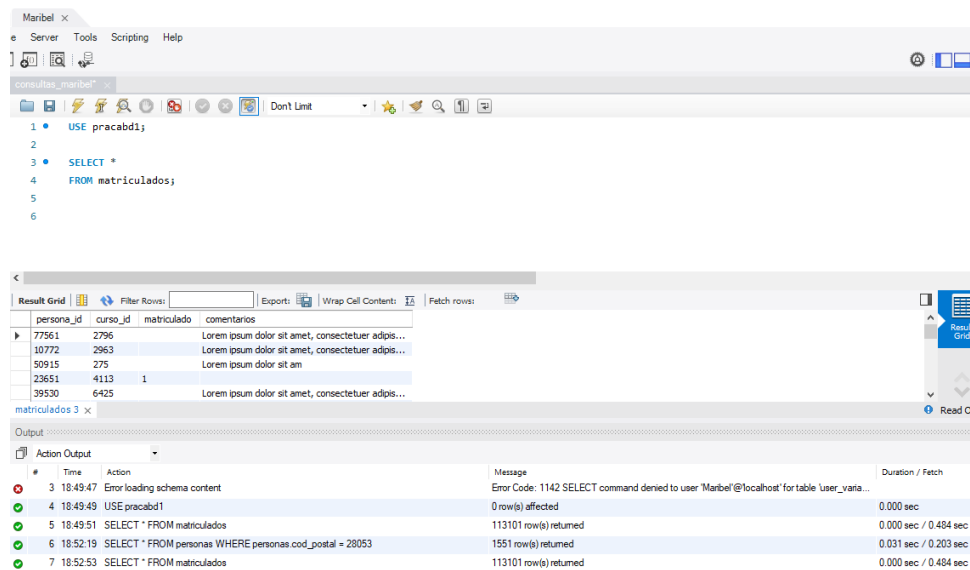
```

#	Time	Action	Message	Duration / Fetch
2	18:43:28	use pracabd1	0 row(s) affected	0.000 sec
3	18:43:30	SELECT * FROM personas	87032 row(s) returned	0.000 sec / 0.500 sec
4	18:43:33	SELECT * FROM matriculados WHERE persona_id = 2	1 row(s) returned	0.172 sec / 0.000 sec
5	18:43:34	DROP SCHEMA pracabd1	Error Code: 1044. Access denied for user 'Santiago'@'localhost' to database 'pracabd1'	0.000 sec
6	18:43:36	DELETE FROM personas WHERE persona_id = 1	Error Code: 1142. DELETE command denied to user 'Santiago'@'localhost' for table 'personas'	0.000 sec

Como podemos observar, el resultado de las sentencias ejecutadas es el mismo que antes.

Maribel tiene asociado el perfil de secretario y sus respectivos permisos. Para comprobarlo ejecutaremos 2 sentencias con sus permisos

- Visualizar toda la información de la tabla matriculados



```

1 • USE pracabd1;
2
3 • SELECT *
4   FROM matriculados;
5
6

```

persona_id	curso_id	matriculado	comentarios
77561	2796		Lorem ipsum dolor sit amet, consectetur adipis...
10772	2963		Lorem ipsum dolor sit amet, consectetur adipis...
50915	275		Lorem ipsum dolor sit am
23651	4113	1	
39530	6425		Lorem ipsum dolor sit amet, consectetur adipis...

#	Time	Action	Message	Duration / Fetch
3	18:49:47	Error loading schema content	Error Code: 1142 SELECT command denied to user 'Maribel'@'localhost' for table 'User_varia...'	0.000 sec
4	18:49:49	USE pracabd1	0 row(s) affected	0.000 sec / 0.484 sec
5	18:49:51	SELECT * FROM matriculados	113101 row(s) returned	0.000 sec / 0.484 sec
6	18:52:19	SELECT * FROM personas WHERE personas.cod_postal = 28053	1551 row(s) returned	0.031 sec / 0.203 sec
7	18:52:53	SELECT * FROM matriculados	113101 row(s) returned	0.000 sec / 0.484 sec

- Visualizar la información de las personas cuyo código postal es 28053

Maribel x

Server Tools Scripting Help

consultas_maribel

```

1 • USE pracabd1;
2
3 • SELECT *
4   FROM matriculados;
5
6 • SELECT *
7   FROM personas
8  WHERE personas.cod_postal = 28053;

```

Result Grid

persona_id	dni	nombre	apellidos	genero	direccion	localidad	provincia	cod_postal	telefono	email
1	99383036W	Mario	Dieza Castillo	H	CALLE DE POZOHALCON	MADRID	MADRID	28053	990257168	MarioDiezaCastillo2@gmail.com
9	99824736C	Julán	Hernán Mesón	H	CALLE DE BLANCA LUNA	MADRID	MADRID	28053	990188107	JulánHernánMesón@gmail.com
22	99586538N	Gabriel	Villaverde	H	CALLE DE LA SIERRA DE MONCHIQUE	MADRID	MADRID	28053	990188107	GabrielVillaverde4@gmail.com
88	99122311O	Gonzalo	Illescas Ramírez	H		MADRID	MADRID	28053	990188107	GonzaloIllescasRamirez3@gmail.com

personas 2 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
2	18:49:47	USE pracabd1	0 row(s) affected	0.000 sec
3	18:49:47	Error loading schema content	Error Code: 1142 SELECT command denied to user 'Maribel'@'localhost' for table 'user_varia...	
4	18:49:49	USE pracabd1	0 row(s) affected	0.000 sec
5	18:49:51	SELECT * FROM matriculados	113101 row(s) returned	0.000 sec / 0.484 sec
6	18:52:19	SELECT * FROM personas WHERE personas.cod_postal = 28053	1551 row(s) returned	0.031 sec / 0.203 sec

Y 2 sentencias a las cuales no tiene permisos

- Borrar la información de la tabla matriculados que esté relacionada con la persona cuyo id sea 1

Maribel x

Server Tools Scripting Help

consultas_maribel

```

1 • USE pracabd1;
2
3 • SELECT *
4   FROM matriculados;
5
6 • SELECT *
7   FROM personas
8  WHERE personas.cod_postal = 28053;
9
10 • DELETE
11   FROM matriculados
12  WHERE matriculados.persona_id = 1;

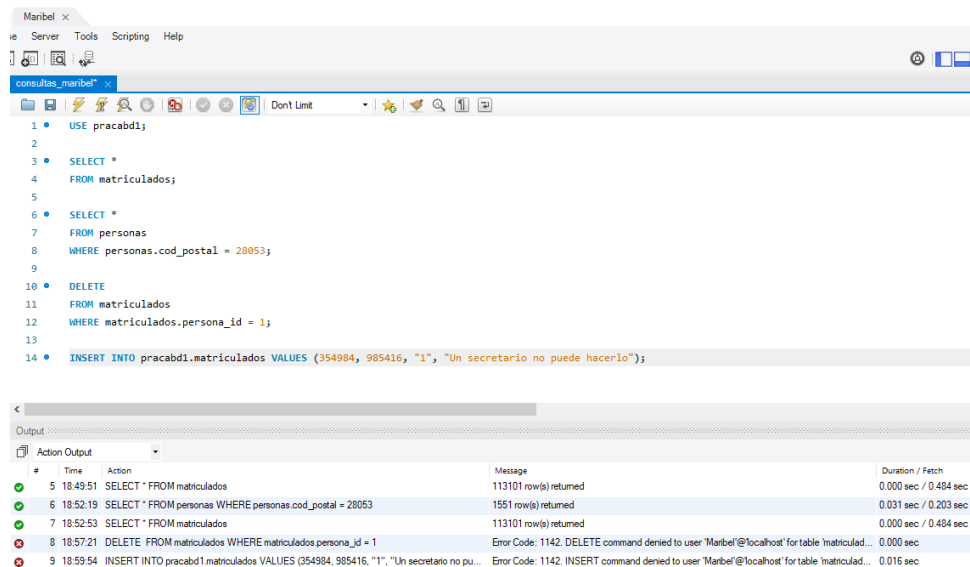
```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
4	18:49:49	USE pracabd1	0 row(s) affected	0.000 sec
5	18:49:51	SELECT * FROM matriculados	113101 row(s) returned	0.000 sec / 0.484 sec
6	18:52:19	SELECT * FROM personas WHERE personas.cod_postal = 28053	1551 row(s) returned	0.031 sec / 0.203 sec
7	18:52:53	SELECT * FROM matriculados	113101 row(s) returned	0.000 sec / 0.484 sec
8	18:57:21	DELETE FROM matriculados WHERE matriculados.persona_id = 1	Error Code: 1142. DELETE command denied to user 'Maribel'@'localhost' for table 'matriculad...	0.000 sec

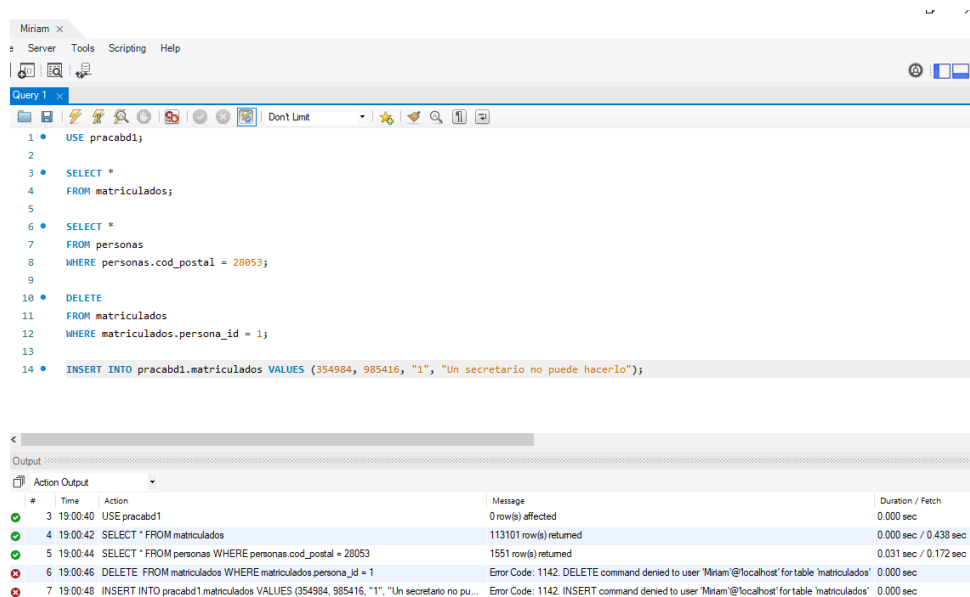
- Actualizar la tabla matriculados añadiendo una nueva entrada



```
1 • USE pracabd1;
2
3 • SELECT *
4   FROM matriculados;
5
6 • SELECT *
7   FROM personas
8   WHERE personas.cod_postal = 28053;
9
10 • DELETE
11   FROM matriculados
12   WHERE matriculados.persona_id = 1;
13
14 • INSERT INTO pracabd1.matriculados VALUES (354984, 985416, "1", "Un secretario no puede hacerlo");
```

#	Time	Action	Message	Duration / Fetch
✓ 5	18:49:51	SELECT * FROM matriculados	113101 row(s) returned	0.000 sec / 0.484 sec
✓ 6	18:52:19	SELECT * FROM personas WHERE personas.cod_postal = 28053	1551 row(s) returned	0.031 sec / 0.203 sec
✓ 7	18:52:53	SELECT * FROM matriculados	113101 row(s) returned	0.000 sec / 0.484 sec
✗ 8	18:57:21	DELETE FROM matriculados WHERE matriculados.persona_id = 1	Error Code: 1142. DELETE command denied to user 'Maribel'@'localhost' for table 'matriculad...	0.000 sec
✗ 9	18:59:54	INSERT INTO pracabd1.matriculados VALUES (354984, 985416, "1", "Un secretario no pu...	Error Code: 1142. INSERT command denied to user 'Maribel'@'localhost' for table 'matriculad...	0.016 sec

Miriam también está relacionada con el rol secretario, por lo que puede hacer lo mismo que Maribel. Lo podemos ver al ejecutar las mismas sentencias que con Maribel



```
1 • USE pracabd1;
2
3 • SELECT *
4   FROM matriculados;
5
6 • SELECT *
7   FROM personas
8   WHERE personas.cod_postal = 28053;
9
10 • DELETE
11   FROM matriculados
12   WHERE matriculados.persona_id = 1;
13
14 • INSERT INTO pracabd1.matriculados VALUES (354984, 985416, "1", "Un secretario no puede hacerlo");
```

#	Time	Action	Message	Duration / Fetch
✓ 3	19:00:40	USE pracabd1	0 row(s) affected	0.000 sec
✓ 4	19:00:42	SELECT * FROM matriculados	113101 row(s) returned	0.000 sec / 0.438 sec
✓ 5	19:00:44	SELECT * FROM personas WHERE personas.cod_postal = 28053	1551 row(s) returned	0.031 sec / 0.172 sec
✗ 6	19:00:46	DELETE FROM matriculados WHERE matriculados.persona_id = 1	Error Code: 1142. DELETE command denied to user 'Miriam'@'localhost' for table 'matriculados'	0.000 sec
✗ 7	19:00:48	INSERT INTO pracabd1.matriculados VALUES (354984, 985416, "1", "Un secretario no pu...	Error Code: 1142. INSERT command denied to user 'Miriam'@'localhost' for table 'matriculados'	0.000 sec

Como podemos observar, el resultado de las sentencias ejecutadas es el mismo.

Eliminar permisos Santiago y Miriam

Como no se puede quitar permisos a un usuario directamente, primero debemos quitarle el rol con los permisos que deben ser eliminados

```
REVOKE 'comercial'@'localhost' FROM 'Santiago'@'localhost';
REVOKE 'secretario'@'localhost' FROM 'Miriam'@'localhost';
```

Al quitar los permisos de comercial y secretario respectivamente, se quitan todos los permisos, hasta los de visualizar la tabla. Para ello, hemos decidido que lo mejor es asignarles un rol “basico”, que es creado y asignado por Javier, que les permita hacer esta función, ya que, si les asignamos

directamente el permiso de lectura, luego no podremos quitárselo directamente, deberíamos borrar el usuario, por ejemplo.

```
CREATE ROLE IF NOT EXISTS 'basico'@'localhost';
```

```
GRANT SELECT ON pracabd1.personas TO 'basico'@'localhost';
```

```
GRANT SELECT ON pracabd1.cursos TO 'basico'@'localhost';
```

```
GRANT SELECT ON pracabd1.matriculados TO 'basico'@'localhost';
```

Con las siguientes sentencias asignamos los permisos y el rol tanto a Santiago como a miriam

```
GRANT 'basico'@'localhost' TO 'Santiago'@'localhost';
```

```
GRANT 'basico'@'localhost' TO 'Miriam'@'localhost';
```

```
SET DEFAULT ROLE 'basico'@'localhost' TO 'Santiago'@'localhost';
```

```
SET DEFAULT ROLE 'basico'@'localhost' TO 'Miriam'@'localhost';
```

Para comprobarlo, realizaremos estas consultas

```
3 • SELECT *
4   FROM matriculados
5  WHERE persona_id = 2;
6
7 • INSERT INTO cursos VALUES (164641,'a','a',234);
8
```

Result Grid	Filter Rows:	Exports	Wrap Cell Content:
persona_id	curso_id	matriculado	comentarios
2	1240	1	Lorem ipsum dolor sit amet, consectetur adipiscing...

#	Time	Action	Message
4	10:05:01	SELECT * FROM pracabd1.personas	87032 row(s) returned
5	10:06:52	INSERT INTO cursos VALUES (164641,'a','a',234)	Error Code: 1142. INSERT command denied to user 'Santiago'@'localhost' for table 'cursos'

PARTE 2: CREACIÓN DE VISTAS Y ASIGNACIÓN DE PRIVILEGIOS

Con el usuario 'root'

Ejecución de operaciones

Los tiempos de las consultas son medidos en la máquina de Gonzalo Puebla Holguín

Operación A1:

```
SELECT personas.nombre, personas.apellidos
FROM pracabd1.personas
ORDER BY personas.provincia ASC;
```

Esta operación tarda de media 0,094 segundos.

Operación A2:

```
SELECT COUNT(personas.persona_id)
FROM pracabd1.personas
WHERE personas.provincia = 'Sevilla';
```

Esta operación tarda de media 0,063 segundos.

Operación A3:

```
SELECT personas.email
FROM pracabd1.personas
WHERE personas.provincia = 'Barcelona';
```

Esta operación tarda de media 0,078 segundos.

Operación A4:

```
INSERT INTO pracabd1.personas (persona_id, dni, nombre, apellidos, genero, direccion, localidad, provincia, telefono, email, en_paro, canal, fecha, cod_postal)
VALUES(100000, '00044561R', 'Gonzalo', 'Puebla Holguín', 'H', 'CALLE DE LA BASE DE DATOS', 'MADRID', 'MADRID', '914532187', 'gonzalopueblaholguin@gmail.com', 1, 2, '1954-11-02', 28018);
```

Esta operación tarda de media 0,000 segundos.

Operación B1:

```
SELECT cursos.curso_id, cursos.nombre, cursos.area
FROM pracabd1.cursos
WHERE cursos.edicion = 2020
ORDER BY cursos.area ASC;
```

Esta operación tarda de media 0,016 segundos.

Operación B2:

```
SELECT COUNT(cursos.curso_id)
FROM pracabd1.cursos
WHERE cursos.area = 'Big Data'
AND cursos.edicion = 2020;
```

Esta operación tarda de media 0,015 segundos.

Operación B3:


```
SELECT cursos.nombre
FROM pracabd1.cursos
WHERE cursos.area = 'Realidad Virtual'
AND cursos.edicion = 2020;
```

Esta operación tarda de media 0,016 segundos.

Operación B4:

```
INSERT INTO pracabd1.cursos(curso_id, nombre, area, edicion)
VALUES(10001, 'CursoTest', 'PracticaABD', 2021);
```

Esta operación tarda de media 0,015 segundos.

Creación de vistas

Vista para el conjunto de datos A:

```
CREATE VIEW A AS
SELECT p.persona_id, p.nombre, p.apellidos, p.provincia, p.telefono, p.email
FROM pracabd1.personas AS p;
```

Vista para el conjunto de datos B:

```
CREATE VIEW B AS
SELECT c.curso_id, c.nombre, c.area
FROM pracabd1.cursos AS c
WHERE c.edicion = 2020;
```

Creación de usuario suplencia y concesión de permisos

Creamos el rol de suplencia y al usuario con estas sentencias.

```
CREATE ROLE IF NOT EXISTS 'suplencia'@'localhost';

GRANT SELECT, INSERT ON pracabd1.A TO 'suplencia'@'localhost';
GRANT SELECT, INSERT ON pracabd1.B TO 'suplencia'@'localhost';

CREATE USER IF NOT EXISTS 'Suplente'@'localhost' DEFAULT ROLE 'suplencia'@'localhost';
GRANT 'suplencia'@'localhost' TO 'Suplente'@'localhost';
```

Primero creamos el rol, después le asignamos permisos al rol y por último creamos el usuario y le asignamos dicho rol.

Con el usuario '*suplencia*'

Ejecución de consultas tipo 'Obtener'

Consulta A1:

```
SELECT nombre, apellidos  
FROM pracabd1.A  
ORDER BY provincia;
```

Esta operación tarda de media 0,109 segundos.

Consulta A2:

```
SELECT COUNT(persona_id)  
FROM pracabd1.A  
WHERE provincia LIKE 'MADRID';
```

Esta operación tarda de media 0,063 segundos.

Consulta A3:

```
SELECT email  
FROM pracabd1.A  
WHERE provincia LIKE 'BARCELONA';
```

Esta operación tarda de media 0,078 segundos.

Consulta B1:

```
SELECT *  
FROM pracabd1.B  
ORDER BY area;
```

Esta operación tarda de media 0,016 segundos.

Consulta B2:

```
SELECT COUNT(curso_id)  
FROM pracabd1.B  
WHERE area LIKE 'Big Data';
```

Esta operación tarda de media 0,016 segundos.

Consulta B3:

```
SELECT nombre
FROM pracabd1.B
WHERE area LIKE 'Realidad Virtual';
```

Esta operación tarda de media 0,016 segundos.

Ejecución de consultas tipo 'Insertar'

Consulta A4:

```
INSERT INTO pracabd1.A VALUES
(123456789, 'dniPrueba', 'nombrePrueba', 'apellidoPruebaInsert', 'H', 'direccionPrueba', 'localidadPrueba',
'MADRID', 12345, 987654321, 1, 2, 0000-00-00, 'emailPruebaInsert@gmail.com');
```

Esta operación tarda de media 0,015 segundos.

Consulta B4:

```
INSERT INTO pracabd1.B VALUES
(123546789, 'nombrePrueba', 'areaPruebaInsert', 2020)
```

Esta operación tarda de media 0,015 segundos.

Comprobación de acceso a tablas

Para comprobar si el usuario Suplente puede acceder a las tablas vamos a realizar la siguiente consulta.

```
SELECT *
FROM pracabd1.cursos;
```

El resultado es el siguiente:

57 17:47:26 SELECT * FROM pracabd1.cursos Error Code: 1142. SELECT command denied to user 'Suplente'@'localhost' for table 'cursos' 0.000 sec

Esto nos indica que, en este caso el usuario Suplente no puede acceder a la tabla cursos porque el rol que se le asignó no tiene los permisos adecuados.

Con el usuario 'secretario'

Al intentar acceder a las vistas creadas para el usuario "suplencia" desde el usuario Miriam(secretaria) aparece el siguiente error:

```
6 • SELECT *
7 FROM pracabd1.A
```


Output

#	Time	Action	Message	Duration / Fetch
1	17:41:31	SELECT * FROM pracabd1.A LIMIT 0, 200	Error Code: 1142. SELECT command denied to user 'Miriam'@'localhost' for table 'a'	0.015 sec
2	17:43:39	SELECT * FROM pracabd1.A LIMIT 0, 200	Error Code: 1142. SELECT command denied to user 'Miriam'@'localhost' for table 'a'	0.000 sec


```
6 • SELECT *
7 FROM pracabd1.B
8
```

Output

#	Time	Action	Message	Duration / Fetch
4	17:48:20	SELECT * FROM pracabd1.B LIMIT 0, 200	Error Code: 1142. SELECT command denied to user 'Miriam'@'localhost' for table 'b'	0.000 sec

Esto se debe a que el rol secretario no tiene permisos de acceso a las vistas.

Comparativa de métodos

Por lo visto en los resultados, ninguno de los métodos afecta al rendimiento pues las consultas e inserciones se ejecutan más o menos en los mismos tiempos. En cuanto a seguridad, es más seguro el método que hace uso de las vistas dado que al tener una mayor limitación, es menos probable que, en este caso, el usuario con el rol de suplencia no puede romper la base de datos puesto que tiene ciertos permisos. Solo puede acceder a las vistas con las limitaciones que tenga. Esto significa que no podría, por ejemplo, acceder a las filas de la tabla cursos cuya edición no fuera 2020 dado que la vista que contiene parte de la tabla cursos está limitada.

PARTE 3: TRANSACCIONES, CONEXIONES REMOTAS Y BACKUPS

1. Ejecución de 2 transacciones simultáneas:

Para la realización de las transacciones, utilizaremos 2 conexiones locales simultáneas con el usuario root. Para cada subapartado utilizaremos las sentencias que iremos viendo.

Una vez tenemos las 2 sesiones abiertas ejecutamos “SET AUTOCOMMIT = 0;” para que no se ejecuten automáticamente los commits.

A) Lectura – lectura:

Con la conexión 1 de root, realizaremos la siguiente transacción:

```
START TRANSACTION;
SELECT *
FROM cursos
WHERE curso_id = 2 FOR UPDATE;
SELECT *
FROM cursos
WHERE curso_id = 3 FOR UPDATE;
COMMIT;
```

Dando como resultado:

✓	1	10:19:47	SET AUTOCOMMIT = 0	0 row(s) affected	0.000 sec
✓	2	10:21:51	START TRANSACTION	0 row(s) affected	0.000 sec
✓	3	10:21:56	SELECT * FROM cursos WHERE curso_id = 2 FOR UPDATE	1 row(s) returned	0.015 sec / 0.000 sec
✓	4	10:22:08	SELECT * FROM cursos WHERE curso_id = 3 FOR UPDATE	1 row(s) returned	4.547 sec / 0.000 sec

Con la conexión 2 de root, realizaremos la siguiente transacción:

```
START TRANSACTION;
SELECT *
FROM cursos
WHERE curso_id = 3 FOR UPDATE;
SELECT *
FROM cursos
WHERE curso_id = 2 FOR UPDATE;
COMMIT;
```

Dando como resultado:

✓	1	10:19:52	SET AUTOCOMMIT = 0	0 row(s) affected	0.000 sec
✓	2	10:22:00	START TRANSACTION	0 row(s) affected	0.000 sec
✓	3	10:22:01	SELECT * FROM cursos WHERE curso_id = 3 FOR UPDATE	1 row(s) returned	0.000 sec / 0.000 sec
✗	4	10:22:13	SELECT * FROM cursos WHERE curso_id = 2 FOR UPDATE	Error Code: 1213. Deadlock found when trying to get lock; try restarting transaction	0.016 sec

La secuencia de las operaciones ha sido:

- Iniciar transacción de root1
- Ejecutar sentencia1 de root1
- Iniciar transacción de root2
- Ejecutar sentencia1 de root2
- Ejecutar sentencia 2 de root1
- Ejecutar sentencia 2 de root2
- Deadlock
- Commit root1

Al ejecutar las sentencias de esta forma, se produce un interbloqueo ya que los recursos se bloquean por cada transacción. Cuando se ejecuta la segunda sentencia de la sesión 1, se encuentra con el recurso bloqueado, por lo que se encola en la cola de espera

#	Time	Action	Message	Duration / Fetch
✓	5	10:24:34	COMMIT	0 row(s) affected 0.000 sec
✓	6	10:24:37	START TRANSACTION	0 row(s) affected 0.000 sec
✓	7	10:24:38	SELECT * FROM cursos WHERE curso_id = 2 FOR UPDATE	1 row(s) returned 0.000 sec / 0.000 sec
4	8	10:24:47	SELECT * FROM cursos WHERE curso_id = 3 FOR UPDATE	Running... ? / ?

Al hacer lo mismo la sesión 2, el motor innodb tiene un sistema que detecta los interbloqueos, por lo que salta la excepción de interbloqueo y obliga a que la transacción termine y libere sus recursos. Al liberarse el recurso que pertenecía a la primera sesión, la segunda puede avanzar y termina

B) Lectura – escritura:

Para los 2 siguientes apartados, hemos insertado 3 filas para no modificar las anteriores y no tener problemas de integridad referencial con la tabla matriculados, pero que aun así siguen teniendo la misma funcionalidad.

11001	BDA	informatica	1
11002	PP	informatica	1
11003	ABD	informatica	1

Con la conexión de root 1 realizaremos la transacción de lectura:

```
START TRANSACTION;
SELECT *
FROM cursos
WHERE curso_id = 11001 FOR UPDATE;
SELECT *
FROM cursos
WHERE curso_id = 11002 FOR UPDATE;
COMMIT;
```

Dando como resultado:

26	10:46:31	START TRANSACTION	0 row(s) affected	0.000 sec
27	10:46:32	SELECT * FROM cursos WHERE curso_id = 11001 FOR UPDATE	1 row(s) returned	4.906 sec / 0.000 sec
28	10:46:46	SELECT * FROM cursos WHERE curso_id = 11002 FOR UPDATE	1 row(s) returned	24.281 sec / 0.000 sec
29	10:47:16	COMMIT	0 row(s) affected	0.000 sec

Con la conexión de root 2 realizaremos la transacción de escritura:

```
START TRANSACTION;
SELECT * FROM cursos WHERE curso_id = 11002 FOR UPDATE;
UPDATE pracabd1.cursos SET curso_id = 11021 WHERE curso_id = 11001;
COMMIT;
```

Dando como resultado:

28	10:46:37	START TRANSACTION	0 row(s) affected	0.000 sec
29	10:46:41	SELECT * FROM cursos WHERE curso_id = 11002 FOR UPDATE	1 row(s) returned	0.000 sec / 0.000 sec
30	10:47:10	UPDATE pracabd1.cursos SET curso_id = 11021 WHERE curso_id = 11001	Error Code: 1213. Deadlock found when trying to get lock; try restarting transaction	0.000 sec

La secuencia de las operaciones ha sido:

- Iniciar transacción de root1
- Iniciar transacción de root2
- Ejecutar sentencia de lectura1 de root1
- Ejecutar sentencia de lectura de root2
- Ejecutar sentencia de lectura2 de root1
- Ejecutar sentencia de escritura de root2
- Deadlock
- Commit root1

Como podemos observar, la transacción de la primera sesión bloquea el recurso con id 11001 y la segunda al recurso cuyo id es 11002. Cuando la primera sesión intenta acceder al recurso de la sesión 2, se encola en la cola de espera ya que es un recurso bloqueado.

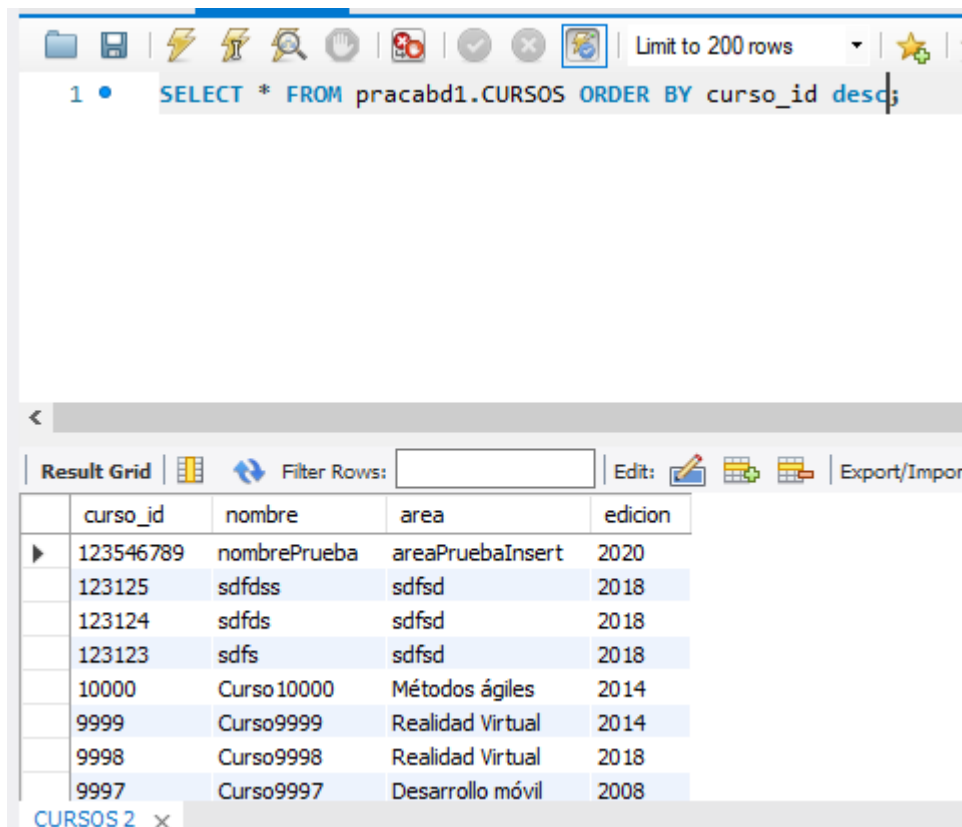
25	10:45:58	COMMIT	0 row(s) affected	0.000 sec
26	10:46:31	START TRANSACTION	0 row(s) affected	0.000 sec
27	10:46:32	SELECT * FROM cursos WHERE curso_id = 11001 FOR UPDATE	1 row(s) returned	4.906 sec / 0.000 sec
4	28	10:46:46	SELECT * FROM cursos WHERE curso_id = 11002 FOR UPDATE	Running...

Al intentar hacer lo mismo la sesión 2 con el recurso de la primera, innodb detecta el interbloqueo y lanza la excepción finalizando la transacción y liberando recursos. Finalmente, con el recurso liberado, puede terminar la primera sesión su transacción con un commit sin ningún problema.

C) Escritura – escritura:

Para realizar esta parte hemos forzado que se produzca un interbloqueo siguiendo los siguientes pasos:

Primero hemos añadido en la tabla cursos 2 filas adicionales para no modificar los datos reales y evitar problemas de integridad referencial:



The screenshot shows a database management tool interface. At the top, there is a toolbar with various icons. Below the toolbar, a SQL query is entered in a text area: `SELECT * FROM pracabd1.CURSOS ORDER BY curso_id desc;`. Below the query, there is a "Result Grid" section. The grid has a header row with columns: `curso_id`, `nombre`, `area`, and `edicion`. The grid contains several rows of data, including rows with `curso_id` values 123546789, 123125, 123124, 123123, 10000, 9999, 9998, and 9997. The row with `curso_id` 123123 is highlighted.

curso_id	nombre	area	edicion
123546789	nombrePrueba	areaPruebaInsert	2020
123125	sdfdss	sfdsd	2018
123124	sfdss	sfdsd	2018
123123	sdfs	sfdsd	2018
10000	Curso10000	Métodos ágiles	2014
9999	Curso9999	Realidad Virtual	2014
9998	Curso9998	Realidad Virtual	2018
9997	Curso9997	Desarrollo móvil	2008

Las filas que vamos a usar para lograr el interbloqueo son aquellas con clave 123123 y 123124.

A continuación, ejecutamos las 2 sentencias siguientes en una de las sesiones:

12	
13	-- Escritura
14	START TRANSACTION;
15	UPDATE pracabd1.cursos SET edicion = 2021 WHERE curso_id = 123124;
16	DELETE FROM pracabd1.cursos WHERE curso_id = 123123;
17	COMMIT;
18	

#	Time	Action	Message	Duration / Fetch
✓ 26	16:49:57	SET AUTOCOMMIT = 0	0 row(s) affected	0.000 sec
✓ 27	16:49:57	SET AUTOCOMMIT = 0	0 row(s) affected	0.000 sec
✓ 28	16:50:49	START TRANSACTION	0 row(s) affected	0.000 sec
✓ 29	16:50:50	UPDATE pracabd1.cursos SET edicion = 2021 WHERE curso_id = 123124	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec

Luego, en la siguiente sesión iniciamos una transacción y ejecutamos las siguientes sentencias de UPDATE y DELETE:

7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	-- Escritura
18	START TRANSACTION;
19	UPDATE pracabd1.cursos SET edicion = 2022 WHERE curso_id = 123123;
20	DELETE FROM pracabd1.cursos WHERE curso_id = 123124;
21	
22	COMMIT;
23	

#	Time	Action	Message	Duration / Fetch
✓ 33	16:50:05	SET AUTOCOMMIT = 0	0 row(s) affected	0.000 sec
✓ 34	16:53:46	START TRANSACTION	0 row(s) affected	0.000 sec
✓ 35	16:53:47	UPDATE pracabd1.cursos SET edicion = 2022 WHERE curso_id = 123123	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec
4 36	16:53:49	DELETE FROM pracabd1.cursos WHERE curso_id = 123124	Running...	?

Como se puede apreciar, esta sesión se ha quedado bloqueada. Esto se debe a que está a la espera de que la otra sesión libere el recurso sobre el que se intenta "escribir" (en este caso borrar).

Si ahora en la otra sesión se ejecuta lo siguiente:

11	SET AUTOCOMMIT = 0;
12	
13	-- Escritura
14	START TRANSACTION;
15	UPDATE pracabd1.cursos SET edicion = 2021 WHERE curso_id = 123124;
16	DELETE FROM pracabd1.cursos WHERE curso_id = 123123;
17	COMMIT;
18	

#	Time	Action	Message	Duration / Fetch
✓ 27	16:49:57	SET AUTOCOMMIT = 0	0 row(s) affected	0.000 sec
✓ 28	16:50:49	START TRANSACTION	0 row(s) affected	0.000 sec
✓ 29	16:50:50	UPDATE pracabd1.cursos SET edicion = 2021 WHERE curso_id = 123124	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec
✗ 30	16:54:22	DELETE FROM pracabd1.cursos WHERE curso_id = 123123	Error Code: 1213. Deadlock found when trying to get lock; try restarting transaction	0.000 sec

Se produce un interbloqueo. Esto se debe a que ambas sesiones están esperando por un recurso que la otra sesión tiene bloqueado mientras cada sesión mantiene bloqueado recurso que la otra sesión

necesita. En particular, lo que está ocurriendo es que la primera sesión (sesion1) primero bloquea la fila con id 123124, después la segunda sesión (sesion2) bloquea la fila con id 123123 y se pone a esperar a que la sesion1 libere la fila con id 123124. Sin embargo, la sesion1 en lugar de liberar dicha fila se pone a esperar por la fila con id 123123 que está bloqueada por la sesion2. En este escenario, se produce un interbloqueo al no poder ninguna de las sesiones continuar con su ejecución.

2. Lectura sucia

Para realizar esta parte, haremos uso de la siguiente sentencia

```
SET GLOBAL TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

Al establecerlo como global, afecta a todas las conexiones, no solo en la que se realiza

Read uncommitted permite leer datos que no son confirmados

Para comprobarlo vamos a realizar 2 operaciones, la primera será de lectura en la conexión 1 del usuario root y la segunda será de escritura en la segunda conexión con el usuario root

Con la conexión de root 1 realizaremos la transacción de escritura:

```
START TRANSACTION;
SELECT * FROM cursos WHERE curso_id = 1;
UPDATE pracabd1.cursos SET area = 'Curso de transacciones' WHERE curso_id = 1;
ROLLBACK;
```

La primera sentencia es para comprobar que se ha hecho el update correctamente.

Con la conexión de root 2 ejecutaremos la siguiente transacción:

```
START TRANSACTION;
SELECT * FROM pracabd1.cursos WHERE curso_id = 1;
ROLLBACK;
```

El orden de ejecución es el siguiente:

Root 1 y root 2 comienzan sus transacciones

Root 2 ejecuta su consulta.

```
1 • SET AUTOCOMMIT = 0;
2
3 • SET GLOBAL TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
4
1 • SET AUTOCOMMIT = 0;
2
3 • SET GLOBAL TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
4
5 • START TRANSACTION;
6 • SELECT * FROM cursos WHERE curso_id = 1;
7 • UPDATE pracabd1.cursos SET area = 'Curso de transacciones' WHERE curso_id = 1;
```

curso_id	nombre	area	edicion
1	Curso1	Curso de transacciones	2011

Root 1 ejecuta su modificación. Además, ejecuta el SELECT para comprobar que lo ha cambiado bien.

Root 2 vuelve a ejecutar su consulta. En este momento se da el caso de lectura sucia.

```
1 • SET AUTOCOMMIT = 0;
2
3 • SET GLOBAL TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
4
5 • START TRANSACTION;
6 • SELECT * FROM pracabd1.cursos WHERE curso_id = 1;
7 • ROLLBACK;
```

curso_id	nombre	area	edicion
1	Curso1	Curso de transacciones	2011

Se puede observar que root 2 obtiene el área que no corresponde.

Por último, para volver al estado inicial, root 1 hace rollback y root 2 vuelve a ejecutar su consulta.

```

1 • SET AUTOCOMMIT = 0;
2
3 • SET GLOBAL TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
4
5 • START TRANSACTION;
6 • SELECT * FROM cursos WHERE curso_id = 1;
7 • UPDATE pracabd1.cursos SET area = 'Curso de transacciones' WHERE curso_id = 1;
8 • ROLLBACK;

```

Output

#	Time	Action	Message	Duration / Fetch
3	10:53:27	START TRANSACTION	0 row(s) affected	0.000 sec
4	10:56:49	UPDATE pracabd1.cursos SET area = 'Curso de transacciones' WHERE curso_id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec
5	10:56:51	SELECT * FROM cursos WHERE curso_id = 1	1 row(s) returned	0.000 sec / 0.000 sec
6	11:01:32	ROLLBACK	0 row(s) affected	0.000 sec

```

1 • SET AUTOCOMMIT = 0;
2
3 • SET GLOBAL TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
4
5 • START TRANSACTION;
6 • SELECT * FROM pracabd1.cursos WHERE curso_id = 1;
7 • ROLLBACK;

```

Result Grid

curso_id	nombre	area	edicion
1	Curso1	Big Data	2011

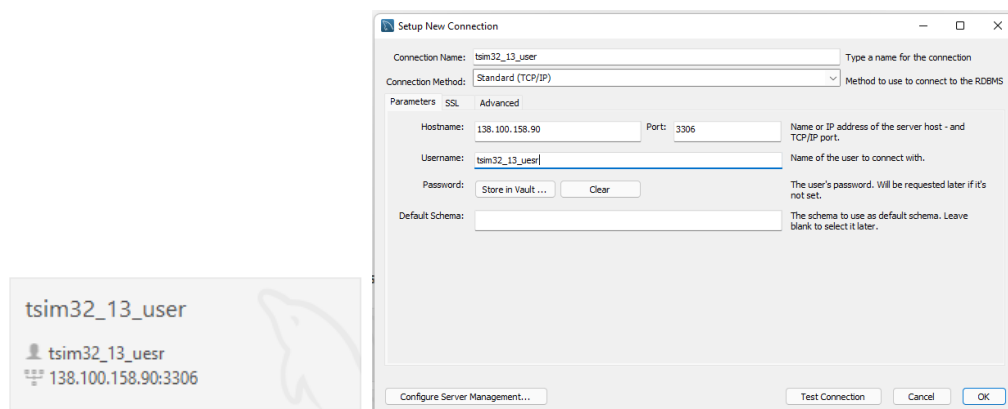
cursos 3

Output

#	Time	Action	Message	Duration / Fetch
3	10:53:31	START TRANSACTION	0 row(s) affected	0.000 sec
4	10:53:33	SELECT * FROM pracabd1.cursos WHERE curso_id = 1	1 row(s) returned	0.000 sec / 0.000 sec
5	10:59:13	SELECT * FROM pracabd1.cursos WHERE curso_id = 1	1 row(s) returned	0.000 sec / 0.000 sec
6	11:01:51	SELECT * FROM pracabd1.cursos WHERE curso_id = 1	1 row(s) returned	0.000 sec / 0.000 sec


3. Conectarse a la BD

Para conectarnos a la base de datos alojada en el servidor cuya dirección IP es 138.100.158.90, creamos una nueva conexión en el MySQLWorkbench



En ella introducimos la dirección IP dada y el puerto que usaremos el puerto por defecto 3006.
También introduciremos la contraseña establecida en clase

Al entrar nos encontraremos con la base de datos creada que nuestro usuario tiene permiso

 **tsim32_13_db**

4. Backup

MYSQL Workbench

Primero exportamos los datos de la base de datos en local a través de MYSQL WB con la opción
server > Data Export.

Para que no haya problemas, hay que cambiar en los ficheros generados el nombre de la base de
datos y eliminar cualquier ocurrencia de los tablespaces.

pracad1_cursos.sql: Bloc de notas

Archivo Edición Formato Ver Ayuda

-- MySQL dump 10.13 Distrib 8.0.26, for Win64 (x86_64)
--
-- Host: localhost Database: tsim32_13_db
-- -----
-- Server version 8.0.26

pracad1_cursos.sql: Bloc de notas

Archivo Edición Formato Ver Ayuda

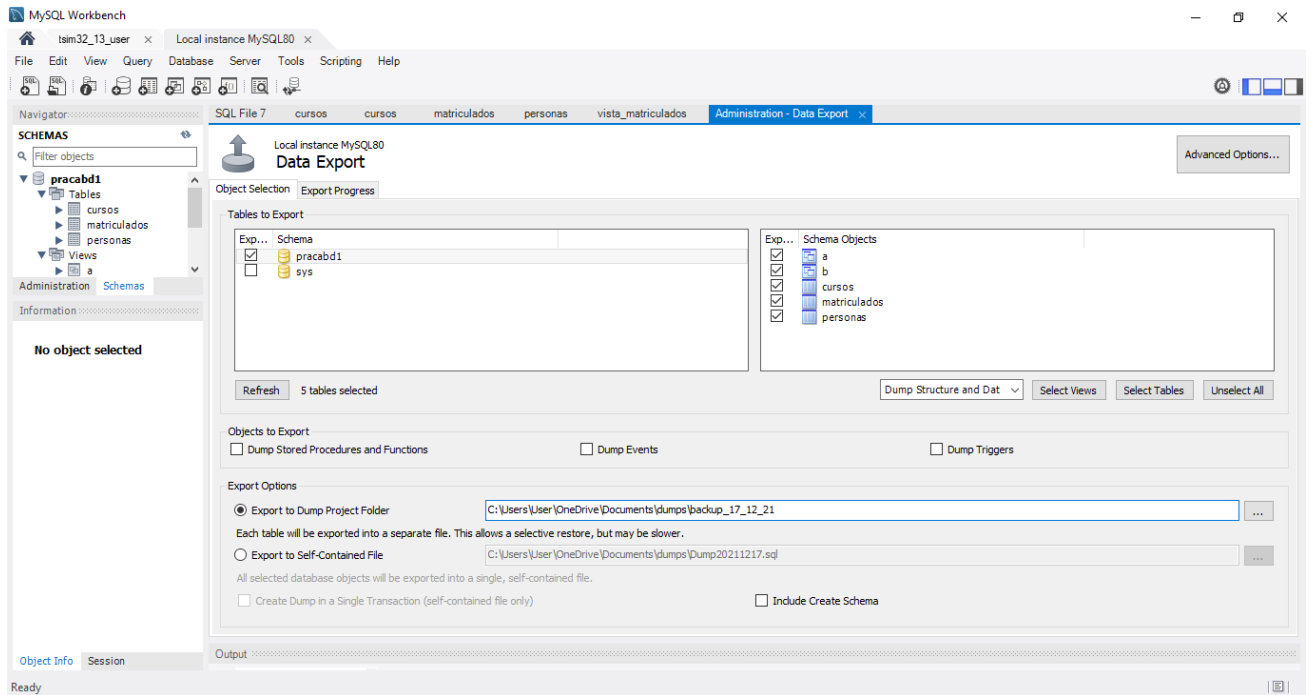
/*!40111 SET @OLD_SQL_NOTES=@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `cursos`
--

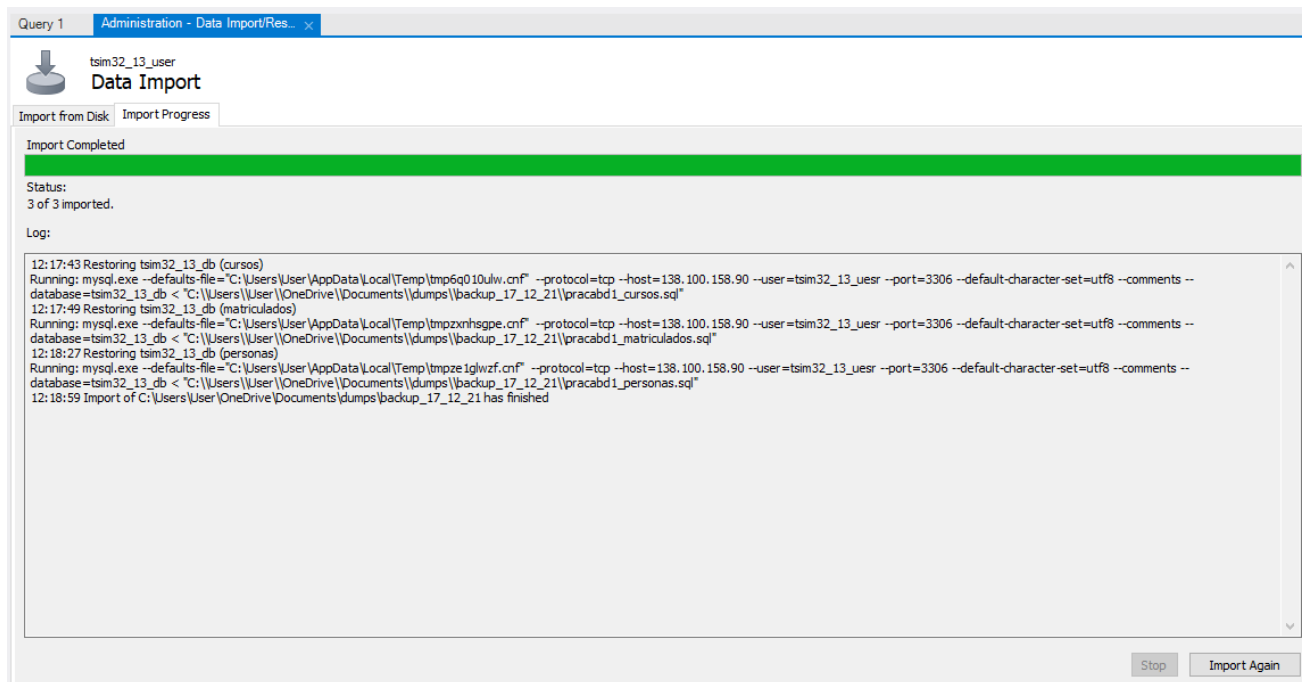
DROP TABLE IF EXISTS `cursos`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `cursos` (
 `curso_id` int NOT NULL,
 `nombre` varchar(15) COLLATE utf8_spanish2_ci NOT NULL,
 `area` varchar(30) COLLATE utf8_spanish2_ci DEFAULT NULL,
 `edicion` int NOT NULL,
 PRIMARY KEY (`curso_id`),
 UNIQUE KEY `curso_id` (`curso_id`),
 UNIQUE KEY `nombre` (`nombre`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8_spanish2_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `cursos`
--

< Línea 32, columna 1 100% Windows (CRLF) UTF-8

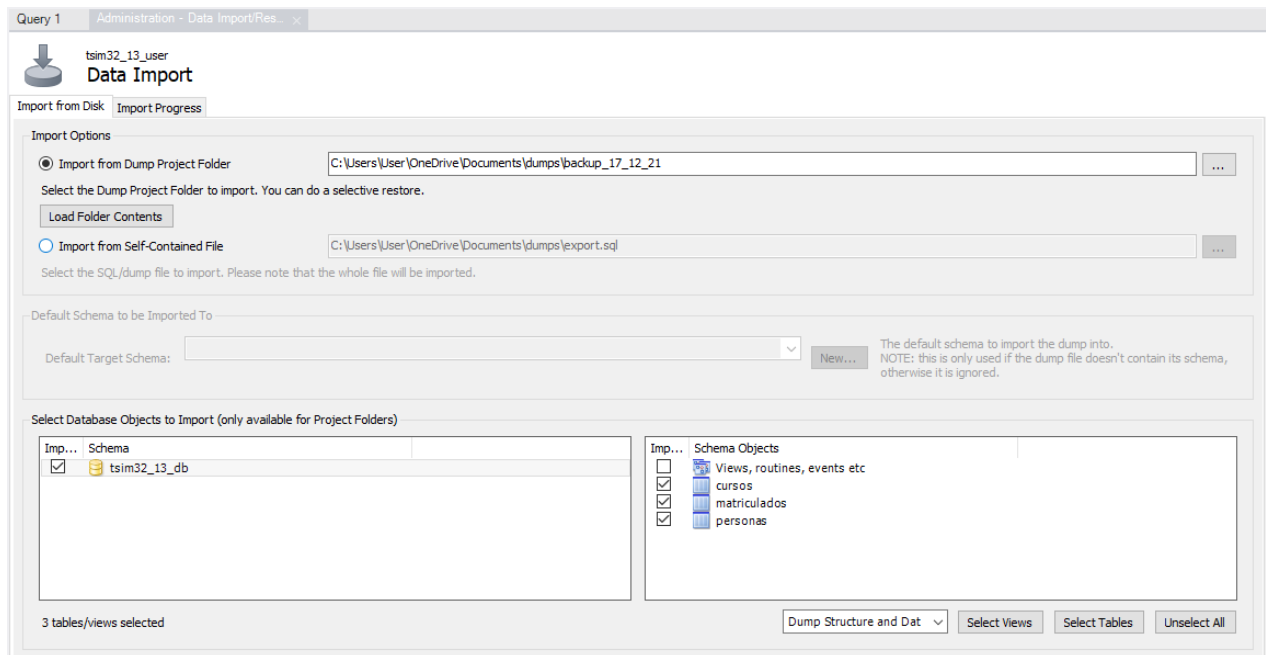


Tras hacer esto vemos como todo funciona correctamente.



Una vez tenemos el backup, lo cargamos en la base de datos alojada en el servidor.

Para ello usamos la opción server > Data Import y seleccionamos el backup que acabamos de hacer:



Importamos solamente las tablas ya que si seleccionábamos la opción que hemos dejado sin seleccionar (views, routines, events, etc.) nos salían errores. No hay ningún inconveniente en hacer esto que las tablas se han importado correctamente y las vistas se pueden crear en otro momento.

Consola

Para poder realizar la copia de la base de datos, ejecutamos la siguiente sentencia en la consola.

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysqldump -u root -p pracabd1 > backup_fecha.sql
Enter password: *****

C:\Program Files\MySQL\MySQL Server 8.0\bin>_
```

Es importante abrir la consola en modo administrador porque no abriéndose en modo administrador surge lo siguiente.

```
Microsoft Windows [Versión 10.0.19044.1415]
(c) Microsoft Corporation. Todos los derechos reservados.

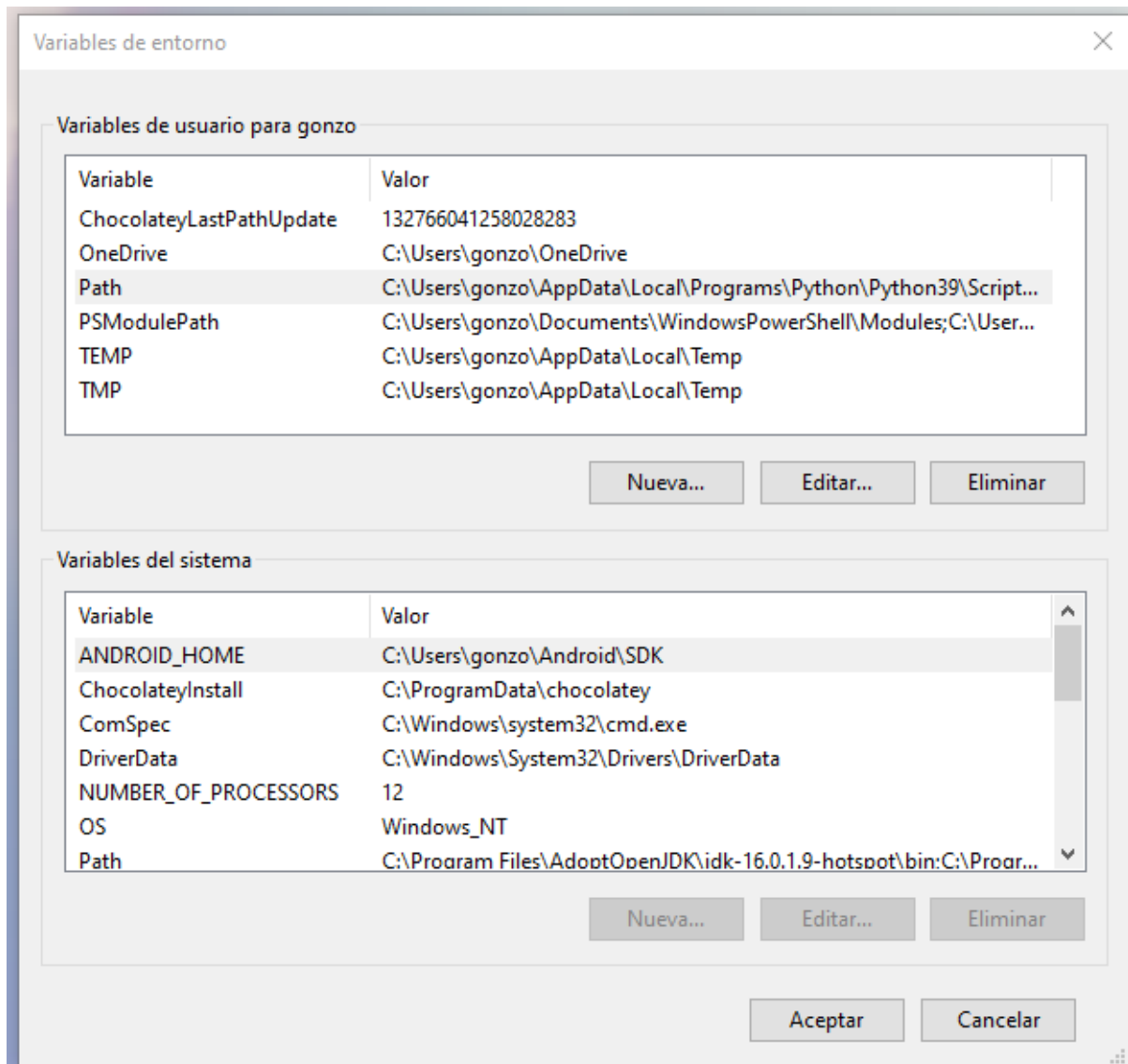
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysqldump -u root -p pracabd1 > backup_fecha2.sql
Acceso denegado.

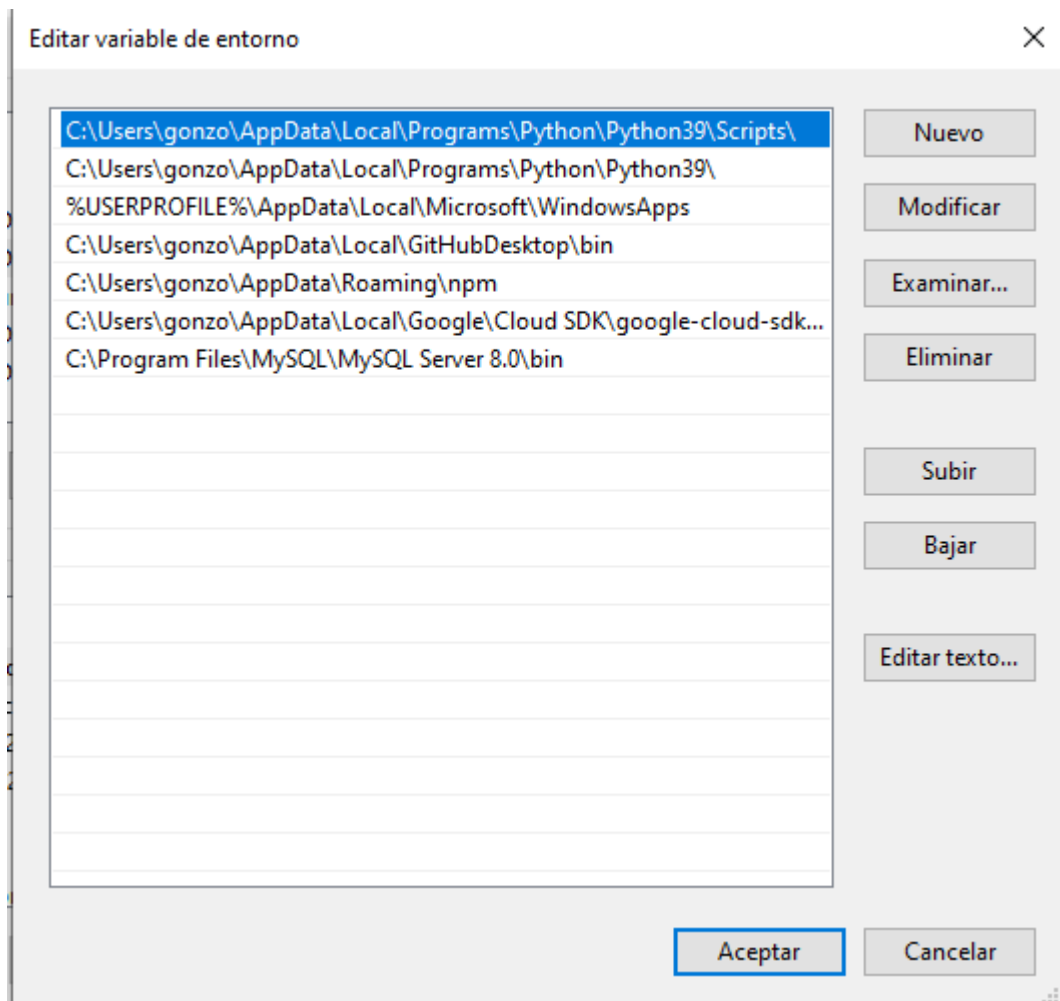
C:\Program Files\MySQL\MySQL Server 8.0\bin>_
```

Nos aparece acceso denegado, sin darnos la opción de introducir la contraseña. Esto es porque se realiza la acción en una carpeta en la que se necesita permisos de administrador para poder crear, modificar o eliminar ficheros en ella.

Para hacerlo fuera de la carpeta bin de MySQL, por ejemplo, en el escritorio, debemos asegurarnos de que, en las variables del entorno, en la variable path tenemos introducida la carpeta bin para poder usar el comando mysqldump sin problema. Para ello buscamos variables del entorno en la

barra de búsqueda de Windows, seleccionamos la opción editar las variables de entorno de esta cuenta, seleccionamos Path, después editar, y añadimos la carpeta bin donde está mysqldump.exe si no está ahí.





Tras realizar este paso se podrá crear backup_fecha desde el escritorio, por ejemplo.

```
C:\Users\gonzo\Desktop>mysqldump -u root -p pracabd1 > backup_fecha.sql
Enter password: *****

C:\Users\gonzo\Desktop>_
```

Tras exportar los datos de la base de datos local, para evitar problemas hay que cambiar el nombre de la base de datos del fichero de backup y eliminar los tablespaces como cuando se realiza con MySQL Workbench. Tras este paso, pasamos a importar los datos a la base de datos que se encuentra en el servidor.

```
C:\Users\gonzo\Desktop>mysql -h 138.100.158.90 -u tsim32_13_uesr -p tsim32_13_db < backup_fecha.sql
Enter password: *****
```

Hemos tenido que eliminar la parte de las vistas creadas en la parte 2 porque no teníamos los permisos adecuados para poder crear las vistas y, por lo tanto, al hacer el import en el servidor, no podíamos meter las vistas.

Para comprobar que el import se realiza adecuadamente, hacemos las siguientes consultas y comprobamos si se han introducido los datos adecuadamente.

```
1 • SELECT * FROM tsim32_13_db.cursos;
```

curso_id	nombre	area	edicion
1	Curso 1	Big Data	2011
2	Curso2	Ciberseguridad	2015
3	Curso3	Ciberseguridad	2016
4	Curso4	Ciberseguridad	2011
5	Curso5	Desarrollo web	2011
6	Curso6	Big Data	2014
7	Curso7	Desarrollo web	2017
8	Curso8	Desarrollo móvil	2020

cursos 1 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	20:18:33	SELECT * FROM tsim32_13_db.personas	87034 row(s) returned	0.109 sec / 6.953 sec
✓ 2	20:18:44	SELECT * FROM tsim32_13_db.matriculados	113096 row(s) returned	0.125 sec / 8.829 sec
✓ 3	20:18:56	SELECT * FROM tsim32_13_db.cursos	10002 row(s) returned	0.094 sec / 0.234 sec

```
1 • SELECT * FROM tsim32_13_db.matriculados;
```

persona_id	curso_id	comentarios	matriculado
1	817	Lorem ipsum dolor sit amet, consectetur adipisicing...	0
2	1240	Lorem ipsum dolor sit amet, consectetur adipisicing...	1
3	31	Lorem ipsum dolor sit amet, consectetur adipisicing...	1
3	719	Lorem ipsum dolor sit amet, consectetur adipisicing...	0
3	2224	Lorem ipsum dolor sit amet, consectetur adipisicing...	1
4	7779		1
5	385	Lorem ipsum dolor sit amet, consectetur adipisicing...	0
6	3903		1

matriculados 1 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	20:18:33	SELECT * FROM tsim32_13_db.personas	87034 row(s) returned	0.109 sec / 6.953 sec
✓ 2	20:18:44	SELECT * FROM tsim32_13_db.matriculados	113096 row(s) returned	0.125 sec / 8.829 sec
✓ 3	20:18:56	SELECT * FROM tsim32_13_db.cursos	10002 row(s) returned	0.094 sec / 0.234 sec

```
1 • SELECT * FROM tsim32_13_db.personas;
```

persona_id	dni	nombre	apellidos	genero	direccion	localidad	provincia	telefono	email	en_
1	99383036W	Mario	Dieza Castillo	H	CALLE DE POZOHALCON	MADRID	MADRID	990257168	MarioDiezaCastillo2@gmail.com	1
2	99317727A	Guillermo	Heredia	H	CALLE DE LOS GALLEGOS	GETAFE	MADRID		GuillermoHeredia1@yahoo.com	1
3	99309930Y	Miguel Angel	Oliveras Cuenca	H	CALLE DE LA PEÑA DEL YELMO	MADRID	MADRID		MiguelAngelOliverasCuenca4@gmail.com	0
4	99462506Y	Micaela	Noria Conde	M	CALLE DE VERDE VIENTO	MADRID	MADRID		MicaelaNoriaConde4@gmail.com	0
5	99470303X	Maria Antonia	Romero Preciado	M	CALLE DE AURELIO DE LA TORRE	MADRID	MADRID		MariaAntoniaRomeroPreciado4@gmail.com	1
6	99317471X	Maria Carmen	Rubio Pulido	M	CALLE DE PIEDRAHITA	MADRID	MADRID		MariaCarmenRubioPulido4@gmail.com	1
7	99462762H	Gabriel	Hontecillas Lozano	H	CALLE DEL HOSPITAL	PARLA	MADRID		GabrielHontecillasLozano3@gmail.com	0

personas 1 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	20:18:33	SELECT * FROM tsim32_13_db.personas	87034 row(s) returned	0.109 sec / 6.953 sec
✓ 2	20:18:44	SELECT * FROM tsim32_13_db.matriculados	113096 row(s) returned	0.125 sec / 8.829 sec
✓ 3	20:18:56	SELECT * FROM tsim32_13_db.cursos	10002 row(s) returned	0.094 sec / 0.234 sec

Podemos observar que los datos se insertan correctamente.

5. Servidor externo

Otorgar permisos al grupo tsim32_14_user

```
GRANT INSERT, DELETE, SELECT, UPDATE ON tsim32_13_db.personas TO 'tsim32_14_user'@'%';  
GRANT INSERT, DELETE, SELECT, UPDATE ON tsim32_13_db.cursos TO 'tsim32_14_user'@'%';  
GRANT INSERT, DELETE, SELECT, UPDATE ON tsim32_13_db.matriculados TO 'tsim32_14_user'@'%';
```

✓	2	10:55:09	GRANT INSERT, DELETE, SELECT, UPDATE ON tsim32_13_db.personas TO 'tsim32_14_us...	0 row(s) affected	0.141 sec
✓	3	10:55:10	GRANT INSERT, DELETE, SELECT, UPDATE ON tsim32_13_db.cursos TO 'tsim32_14_user'...	0 row(s) affected	0.125 sec
✓	4	10:55:13	GRANT INSERT, DELETE, SELECT, UPDATE ON tsim32_13_db.matriculados TO 'tsim32_14...	0 row(s) affected	0.235 sec

Vemos que se han otorgado los permisos adecuadamente.

Replicamos el apartado 3.1:

Antes de comenzar, como en los anteriores apartados, comenzaremos estableciendo el autocommit a 0 para evitar que cada sentencia sea una transacción en sí y así realizarlas correctamente sin ningún tipo de error

```
SET autocommit = 0;
```

A) Lectura – Lectura:

En la sesión 1:

```
SET autocommit = 0;
```

```
START TRANSACTION;
```

```
SELECT * FROM tsim32_13_db.personas WHERE persona_id = 3 FOR UPDATE;
```

En la sesión 2:

The screenshot shows a MySQL IDE with two sessions. Session 1 (top) has executed the following queries:
1. SET autocommit = 0;
2. START TRANSACTION;
3. SELECT * FROM tsim32_13_db.personas WHERE persona_id = 3 FOR UPDATE;
Session 2 (bottom) has executed the following queries:
1. SET autocommit = 0;
2. START TRANSACTION;
3. SELECT * FROM tsim32_13_db.personas WHERE persona_id = 3 FOR UPDATE;
The output window shows the results of these queries, including the list of rows in the 'personas' table and the execution status of the transactions.

Como podemos ver, en la segunda sesión se realiza la sentencia de forma correcta

#	Time	Action	Message	Duration / Fetch
5	11:12:20	SELECT * FROM tsim32_13_db.personas WHERE PersonalID = 3 FOR UPDATE	Error Code: 1054. Unknown column 'PersonalID' in 'where clause'	0.078 sec
6	11:12:33	SELECT * FROM tsim32_13_db.personas	87032 row(s) returned	0.125 sec / 5.703 sec
7	11:13:45	START TRANSACTION	0 row(s) affected	0.063 sec
8	11:13:49	SELECT * FROM tsim32_13_db.personas WHERE persona_id = 3 FOR UPDATE	Error Code: 2013. Lost connection to MySQL server during query	30.000 sec

Pero en la primera sesión se produce un error ya que el usuario de la segunda sesión bloquea el recurso con la cláusula FOR UPDATE, la cual bloquea un recurso tanto para lectura como para escritura (UPDATE, DELETE y SELECT) hasta que no se termina de ser usado. Si otro usuario (en este

caso la sesión 1) quiere acceder a dicho recurso, se pone en espera hasta que el recurso se libere. Finalmente, la espera termina porque se excede el tiempo de espera.

En estos casos no se produce un interbloqueo, pero si lo realizásemos como el apartado 3.1 en el que cada sesión bloquea primero un recurso y luego intentan acceder cada uno al recurso contrario, se produciría un interbloqueo ya que ninguno podría avanzar en espera de recursos y finalizarían con timeout.

B) Lectura- Escritura

En la sesión 1:

En esta sesión ejecutaremos la sentencia de lectura

```
SET autocommit = 0;

START TRANSACTION;

SELECT * FROM tsim32_13_db.personas WHERE persona_id = 3 FOR SHARE;

commit
```

Dando como resultado

9	11:19:19	commit	0 row(s) affected	0.063 sec
10	11:21:13	SET autocommit = 0	0 row(s) affected	0.062 sec
11	11:22:25	START TRANSACTION	0 row(s) affected	0.062 sec
12	11:22:26	SELECT * FROM tsim32_13_db.personas WHERE persona_id = 3 FOR SHARE	1 row(s) returned	5.234 sec / 0.000 sec

En la sesión 2:

En esta sesión ejecutaremos la sentencia de escritura “DELETE FROM tsim32_13_db.personas WHERE persona_id = 3;”

```
SET autocommit = 0;

start transaction;
select * from personas;
select * from tsim32_13_db.personas where persona_id = 3 for update;
delete from tsim32_13_db.personas where persona_id = 3;
rollback;
```

Dando como resultado:

#	Time	Action	Message	Duration / Fetch
1	11:09:08	SET autocommit = 0	0 row(s) affected	0.062 sec
2	11:09:14	start transaction	0 row(s) affected	0.062 sec
3	11:13:24	select * from personas LIMIT 0, 1000	1000 row(s) returned	0.094 sec / 0.062 sec
4	11:13:37	select * from tsim32_13_db.personas where persona_id = 3 LIMIT 0, 1000 for update	1 row(s) returned	0.062 sec / 0.000 sec
5	11:22:32	rollback	0 row(s) affected	0.062 sec
6	11:22:43	start transaction	0 row(s) affected	0.062 sec
7	11:22:59	select * from tsim32_13_db.personas where persona_id = 3 LIMIT 0, 1000 for update	Error Code: 2013. Last connection to MySQL server during query	30.000 sec

Como podemos observar, la sesión de lectura bloquea el recurso que intenta acceder la segunda sesión con la cláusula FOR SHARE. Es una cláusula muy parecida a FOR UPDATE ya que bloquea los recursos usados, pero solo para operaciones de escritura (UPDATE y DELETE) ya que si se quisiese hacer una de lectura (SELECT) si se podría hacer. El usuario de la sesión 2, al intentar acceder, se encuentra con el bloqueo y se encola para acceder a él. Finalmente, termina la ejecución por un timeout

C) Escritura – Escritura:

Primero añadimos 2 filas auxiliares para evitar problemas de integridad referencial (debido a la tabla matriculados). Una vez tenemos esto, ejecutamos en el orden que aparece en los comentarios las siguientes sentencias:

En la sesión 1 ejecutaremos las siguientes sentencias:

```
INSERT INTO cursos VALUES
(123124, 'nombrePrueba', 'areaPruebaInsert', 2020);

INSERT INTO cursos VALUES
(123123, 'nombrePrueba2', 'areaPruebaInsert2', 2020);

-- sesion 1
SET AUTOCOMMIT = 0;

-- Escritura
START TRANSACTION;
    UPDATE tsim32_13_db.cursos SET edicion = 2021 WHERE curso_id = 123124; -- 1
    DELETE FROM tsim32_13_db.cursos WHERE curso_id = 123123; -- 4 (se produce deadlock)
rollback;
COMMIT;
```

En la sesión 2:

```
-- sesion 2

SET AUTOCOMMIT = 0;

-- Escritura
START TRANSACTION;
    UPDATE tsim32_13_db.cursos SET edicion = 2022 WHERE curso_id = 123123; -- 2
    DELETE FROM tsim32_13_db.cursos WHERE curso_id = 123124; -- 3 (se queda en espera)
rollback;
COMMIT;
```

Tras las ejecuciones aparecen los siguientes outputs:

En la sesión 1:

43	11:49:35	START TRANSACTION	0 row(s) affected	0.063 sec
43	11:49:35	UPDATE tsim32_13_db.cursos SET edicion = 2021 WHERE curso_id = 123124	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.064 sec
44	11:50:35	DELETE FROM tsim32_13_db.cursos WHERE curso_id = 123123	Error Code: 1213. Deadlock found when trying to get lock; try restarting transaction	0.189 sec

En la sesión 2:

#	Time	Action	Message	Duration / Fetch
33	11:49:00	rollback	0 row(s) affected	0.125 sec
34	11:49:03	SET AUTOCOMMIT = 0	0 row(s) affected	0.062 sec
35	11:49:07	START TRANSACTION	0 row(s) affected	0.078 sec
36	11:49:13	UPDATE tsim32_13_db.cursos SET edicion = 2022 WHERE curso_id = 123123	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.078 sec
37	11:49:47	DELETE FROM tsim32_13_db.cursos WHERE curso_id = 123124	Error Code: 2013. Lost connection to MySQL server during query	30.000 sec
38	11:50:51	SET AUTOCOMMIT = 0	0 row(s) affected	0.062 sec
39	11:50:51	START TRANSACTION	0 row(s) affected	0.063 sec
40	11:50:51	UPDATE tsim32_13_db.cursos SET edicion = 2021 WHERE curso_id = 123124	Error Code: 1142. UPDATE command denied to user 'tsim32_14_user'@'188.25.223.158' for table 'cursos'	0.062 sec

Como está indicado en las imágenes, el orden de ejecución de las sentencias es:

- Iniciar la transacción 1
- Iniciar la transacción 2

- Ejecutamos el UPDATE en la sesión 1
- Ejecutamos el UPDATE en la sesión 2
- Ejecutamos el DELETE en la sesión 2
- Ejecutamos el DELETE en la sesión 1

Se produce un fallo debido al interbloqueo que fuerza la finalización de ambas transacciones

D) Lectura Sucia

Para comenzar, debemos cambiar el nivel de aislamiento para que se puedan hacer lecturas de datos sin estar confirmados, lo que puede generar problemas de lectura sucia.

```

SET autocommit = 0;

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

START TRANSACTION;
SELECT * FROM personas WHERE personas.persona_id = 3;
ROLLBACK;

set autocommit = 0;

set transaction isolation level read uncommitted;

start transaction;

select * from personas where personas.persona_id = 3;
update personas set personas.localidad = 'Avenida de los AGBD' where persona_id = 3;

rollback;

```

Se comienzan a ejecutar también las transacciones de cada sesión de tal forma que primero la sesión 2 actualiza la entrada que posteriormente la sesión 1 va a leer.

Primero observamos como se encuentra el dato antes de ser modificado

persona_id	dni	nombre	apellidos	genero	direccion	localidad	provincia	telefono	email	en_paro	canal	fecha
3	99309930Y	Miguel Angel	Oliveras Cuenca	H	CALLE DE LA PEÑA DEL YELMO	MADRID	MADRID		MiguelAngelOliverasCuenca4@gmail.com	0	4	1932-01-01

#	Time	Action	Message	Duration / Fetch
50	11:56:24	SELECT * FROM personas WHERE personas.persona_id = 3	1 row(s) returned	0.063 sec / 0.000 sec
51	11:56:30	ROLLBACK	0 row(s) affected	0.063 sec
52	11:57:21	START TRANSACTION	0 row(s) affected	0.079 sec
53	11:57:25	SELECT * FROM personas WHERE personas.persona_id = 3	1 row(s) returned	0.063 sec / 0.000 sec

Se ejecutan las sentencias de cambio de la sesión 2

#	Time	Action	Message	Duration / Fetch
51	11:56:20	rollback	0 row(s) affected	0.079 sec
52	11:57:21	set autocommit = 0	0 row(s) affected	0.079 sec
53	11:57:27	set transaction isolation level read uncommitted	0 row(s) affected	0.063 sec
54	11:57:29	start transaction	0 row(s) affected	0.062 sec
55	11:57:32	select * from personas where personas.persona_id = 3 LIMIT 0, 1000	1 row(s) returned	0.062 sec / 0.000 sec
56	11:57:30	update personas set personas.localidad = 'Avenida de los AGBD' where persona_id = 3	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.078 sec

Desde la sesión 1 podemos ver que los valores han cambiado y se están leyendo datos que no están confirmados

persona_id	dni	nombre	apellidos	genero	direccion	localidad	provincia	telefono	email	en_paro	canal	fecha
3	99309930Y	Miguel Angel	Oliveras Cuenca	H	CALLE DE LA PEÑA DEL YELMO	Avenida de los AGBD	MADRID		MiguelAngelOliverasCuenca4@gmail.com	0	4	

#	Time	Action	Message	Duration / Fetch
56	11:58:44	ROLLBACK	0 row(s) affected	0.063 sec
57	11:58:46	SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED	0 row(s) affected	0.078 sec
58	11:58:47	START TRANSACTION	0 row(s) affected	0.062 sec
59	11:58:49	SELECT * FROM personas WHERE personas.persona_id = 3	1 row(s) returned	0.078 sec / 0.000 sec

Finalmente se hace ROLLBACK de ambas transacciones para evitar cambios innecesarios en la BD

persona_id	dni	nombre	apellidos	genero	direccion	localidad	provincia	telefono	email	en_paro	canal	fecha
3	99309930Y	Miguel Angel	Oliveras Cuenca	H	CALLE DE LA PEÑA DEL YELMO	MADRID	MADRID		MiguelAngelOliverasCuenca4@gmail.com	0	4	1932-01-2

#	Time	Action	Message	Duration / Fetch
57	11:58:46	SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED	0 row(s) affected	0.078 sec
58	11:58:47	START TRANSACTION	0 row(s) affected	0.062 sec
59	11:58:49	SELECT * FROM personas WHERE personas.persona_id = 3	1 row(s) returned	0.078 sec / 0.000 sec
60	11:59:59	SELECT * FROM personas WHERE personas.persona_id = 3	1 row(s) returned	0.078 sec / 0.000 sec

Como podemos observar, la BD vuelve a estar como al principio sin haberse visto modificada.

CONCLUSIONES

Dadas las actividades realizadas en esta práctica hemos comprendido cómo se gestiona la seguridad en una base de datos y que supone de cara a los accesos de cada usuario en la misma. Además, nos ha servido esta práctica para ver que puede suceder si las transacciones se estorban unas a otras haciendo uso de los mismos recursos, así dando mucha importancia a la integridad de los datos y dando a conocer la facilidad de romper dicha integridad.