

ENTREGA FINAL

SQL

Alumno: Nicolás Cardozo

Año: 2022

INDICE

1. Introducción.....	3
2. Objetivo.....	3
3. Situación Problemática.....	3
4. Modelo de negocio.....	4
5. Diagrama de Entidad-Relación.....	5
6. Listado de tablas con descripción de su estructura.....	6
7. Paso a paso para importación de datos.....	8
8. Paso a paso para la creación de los INSERT INTO.....	13
9. Creación de vistas	19
10. Creación de funciones.....	21
11. Creación de stored procedures.....	23
12. Creación de triggers.....	25
13. Sublenguaje TCL.....	29
14. Backup.....	30
15. Herramientas y tecnologías.....	32

1) Introducción

Creación de mi propia base de datos, en la cual se implementó el modelo relacional para representar procesos basados en un modelo de negocio propio. El modelo de negocios elegido fue el de una planta de producción de neumáticos. Tomándose como ejemplo los datos de dos subplantas, planta materiales y planta toberas. Ya que en total la planta tiene más de 400 máquinas distintas que reportan datos y no venía al caso hacer la bb.dd. tan extensa.

Script SQL

https://github.com/nicolasmcardozo/SQL_Coder

2) Objetivo

- Objetivo 1 Crear una base de datos relacional, basada en un modelo de negocio.
- Objetivo 2 Desarrollar objetos que permitan el mantenimiento de la base de datos.
- Objetivo 3 Implementar consultas SQL que permitan la generación de informes.
- Objetivo 4 Presentar a la compañía indicadores claves del negocio para la toma de decisiones.

3) Situación Problemática

Como ya fue mencionado, la temática se basa en el análisis y la obtención de indicadores en el área de producción de una empresa, la problemática es que no se cuenta con información ordenada y confiable para la toma de decisiones.

La información con la que contamos es:

- La cantidad programada de cada elemento por subcentro, máquina y fecha con el legajo del programador de turno.
- La cantidad producida de cada elemento por subcentro, máquina y fecha con el legajo del operario de turno.
- Tiempo de duración de producción de elementos por subcentro, máquina y fecha.
- Código de porqué la máquina estuvo parada con su correspondiente tiempo por subcentro, máquina y fecha.
- Detalle de nombre y apellido de programador.
- Detalle de nombre y apellido de operador.
- Detalle de elementos con sus descripciones completas.
- Detalle de máquinas con sus características.
- Detalle con la descripción de cada código de máquina parada.
- Detalle con descripción de cada subcentro.

Algunos indicadores que se buscan obtener:

- Cumplimiento de programa por subcentro, máquina y/o fecha

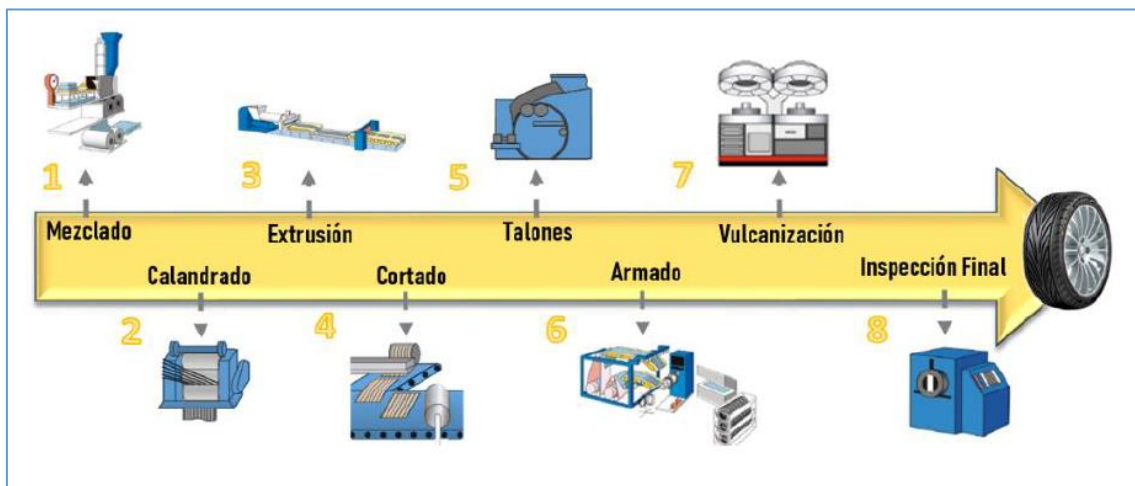
- Cumplimiento de programa por operador de máquina.
- Resumen de cumplimiento por máquina mes a mes.
- Código de interrupción de máquina con más cantidad de minutos cuando el programa no se cumple.
- Etc.

4) Modelo de negocio

Empresa argentina líder en fabricación y exportación de neumáticos. Ocupa un predio de 40 hectáreas y abarca una superficie cubierta de más de 157.513 m². La capacidad productiva supera los cinco millones de neumáticos por año. Se realiza internamente todo el proceso de diseño de sus productos, desde la compleja ingeniería de su estructura y la formulación de sus compuestos, hasta los elaborados ensayos requeridos para su homologación. Esto permite obtener los neumáticos más aptos para todos los caminos de la Argentina. Los productos incluyen neumáticos diagonales y radiales para automóviles, camionetas, camiones y colectivos. También incluyen diagonales para tractores y maquinaria vial, así como una completa línea de cámaras de aire y otros productos para la preservación y reparación de neumáticos.

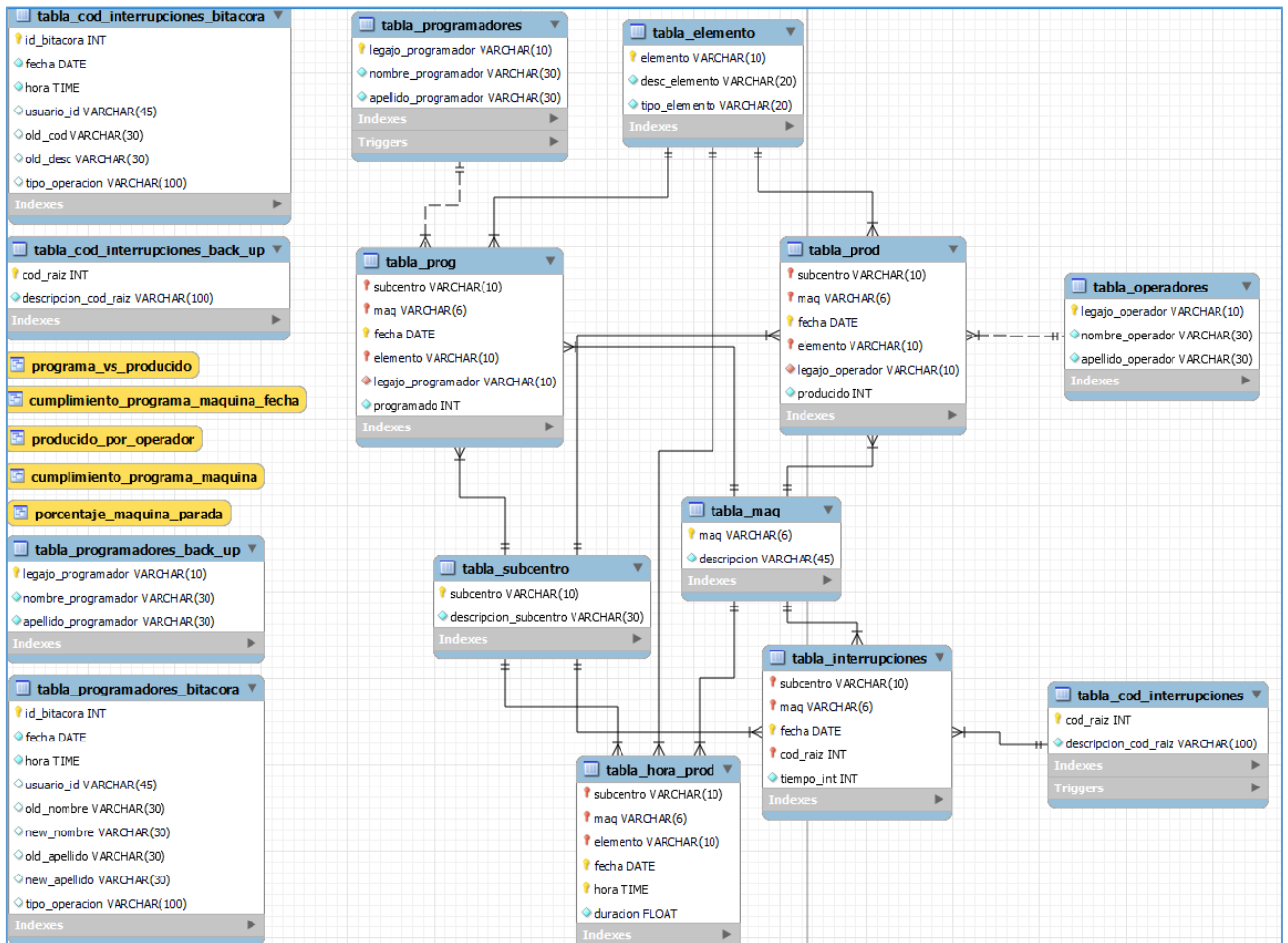
Mediante un elaborado sistema logístico, abastece en la Argentina un poco más de la cuarta parte de la demanda de neumáticos de reposición. Este exitoso desempeño se asienta en la disposición y fortaleza de una red de más de 250 Comercios Autorizados.

Flujo de proceso:



En la base de datos se estarán abordado datos de Mezclado y Extrusionado.

5) Diagrama de entidad-relación de la base de datos



6) Listado de tablas con descripción de su estructura

nombre_tablas	nombre_campo	claves	tipo_de_dato	descripcion_tabla	observaciones
tabla_prog	subcentro	pk , fk	varchar(10)	Tabla obtenida del área de programación: cantidad programada de cada elemento por subcentro, maquina y fecha	Para que mi pk sea único, tengo que unir subcentro, máquina, elemento y fecha (clave compuesta) ya que la tabla como viene dada no tiene id
	maq	pk , fk	varchar(6)		
	fecha	pk	date		
	elemento	pk, fk	varchar(10)		
	legajo_programador	fk	varchar(10)		
	programado		int		
tabla_prod	subcentro	pk , fk	varchar(10)	Tabla obtenida del área de producción: cantidad producida de cada elemento por subcentro, maquina y fecha	Para que mi pk sea único, tengo que unir subcentro, máquina, elemento y fecha (clave compuesta) ya que la tabla como viene dada no tiene id
	maq	pk , fk	varchar(6)		
	fecha	pk	date		
	elemento	pk, fk	varchar(10)		
	legajo_operador	fk	varchar(10)		
	producido		int		
tabla_maq	maq	pk	varchar(6)	tabla que se obtiene de las especificaciones de las maquinas, probablemente tenga mas campos pero puse algunos a modo ejemplo	mi pk será campo maq que uno con campo maq de tablas tabla_prog , tabla_pro , tabla_interrupciones y tabla_hora_prod
	descripcion		varchar(45)		
tabla_subcentro	subcentro	pk	varchar(10)	tabla que se obtiene de las especificaciones de subcentros	mi pk será campo subcentro que uno con campo subcentro de tablas tabla_prog , tabla_prod , tabla_interrupciones y tabla_hora_prod
	descripcion_subcentro		varchar(30)		
tabla_elemento	elemento	pk	varchar(10)	tabla que se obtiene de las especificaciones de los elementos, probablemente tenga mas campos pero puse algunos a modo ejemplo	mi pk será elemento que uno con campo elemento de tablas tabla_prog , tabla_prod y tabla_hora_prod
	desc_elemento		varchar(20)		
	tipo_elemento		varchar(20)		
tabla_programadores	legajo_programador	pk	varchar(10)	tabla que se obtiene del payroll	mi pk será legajo_programador que uno con campo legajo_programador de tabla tabla_prog
	nombre_programador		varchar(30)		
	apellido_programador		varchar(30)		
tabla_operadores	legajo_operador	pk	varchar(10)	tabla que se obtiene del payroll	mi pk será legajo_operador que uno con campo legajo_operador de tabla tabla_prod
	nombre_operador		varchar(30)		
	apellido_operador		varchar(30)		
tabla_interrupciones	subcentro	pk, fk	varchar(10)	Tabla obtenida del área de producción: tiempo que para la máquina con su identificación de parada por subcentro, maquina y fecha	Para que mi pk sea único, tengo que unir subcentro, maq, cod_raiz y fecha (clave compuesta) ya que la tabla como viene dada no tiene id
	maq	pk, fk	varchar(6)		
	fecha	pk	date		
	cod_raiz	pk, fk	int		
	tiempo_int		int		

tabla_cod_interrupciones	cod_raiz	pk	int	tabla que se obtiene de las especificaciones de las interrupciones	mi pk será <i>cod_raiz</i> que uno con <i>cod_raiz</i> de tabla <i>tabla_interrupciones</i>
	descripcion_cod_raiz		varchar(100)		
tabla_hora_prod	subcentro	pk, fk	varchar(10)	Tabla obtenida del área de producción: duración del ciclo de cada elemento por subcentro, máquina, fecha	Para que mi pk sea único, tengo que unir <i>subcentro</i> , <i>maq</i> , <i>elemento</i> , <i>fecha y hora</i> (clave compuesta) ya que la tabla como viene dada no tiene id
	maq	pk, fk	varchar(6)		
	elemento	pk, fk	varchar(10)		
	fecha	pk	date		
	hora	pk	time		
	duracion		float		
tabla_programadores_back_up	legajo_programador	pk	varchar(10)	Tabla backup de tabla_programadores	
	nombre_programador		varchar(30)		
	apellido_programador		varchar(30)		
tabla_cod_interrupciones_back_up	cod_raiz	pk	int	Tabla backup de tabla_programadores	
	descripcion_cod_raiz		varchar(100)		
tabla_programadores_bitacora	id_bitacora	pk	int	Tabla bitacora al actualizar tabla_programadores	
	fecha		date		
	hora		time		
	usuario_id		varchar(45)		
	old_nombre		varchar(30)		
	new_nombre		varchar(30)		
	old_apellido		varchar(30)		
	new_apellido		varchar(30)		
	tipo_operacion		varchar(100)		
tabla_cod_interrupciones_bitacora	id_bitacora	pk	int	Tabla bitacora al eliminar un registro en tabla_cod_interrupciones	
	fecha		date		
	hora		time		
	usuario_id		varchar(45)		
	old_cod		varchar(30)		
	old_desc		varchar(30)		
	tipo_operacion		varchar(100)		

7) Paso a paso para importación de datos

Se ha decidido realizar la importación de cada una de las tablas mediante el asistente de importación de Workbench, mostraré el paso a paso con la importación de una de las tablas, siendo luego el mismo proceso para cada una de ellas. Finalmente, mostraré la evidencia de que cada una de las tablas fueron cargadas según corresponde.

Las tablas a importar serán un total de 10 nombradas a continuación

- Tabla_prog
- Tabla_prod
- Tabla_maq
- Tabla_subcentro
- Tabla_elemento
- Tabla_programadores
- Tabla_operadores
- Tabla_interrupciones
- Tabla_cod_interrupciones
- Tabla_hora_prod

Se mostrará el ejemplo para la carga de la tabla **Tabla_cod_interrupciones**

Paso 1. Ejecuto el uso del proyecto, en este caso: **use proyecto;**

The screenshot shows the SQL Workbench interface with a query editor titled 'Query 1' and 'primer_entrega_proyecto_SQL*'. The query editor contains the following SQL code:

```
1 • create database proyecto;  
2  
3 • use proyecto;  
4
```

The 'use proyecto;' statement on line 3 is highlighted with a red box. The toolbar above the editor includes icons for file operations, execution (lightning bolt), and other database functions.

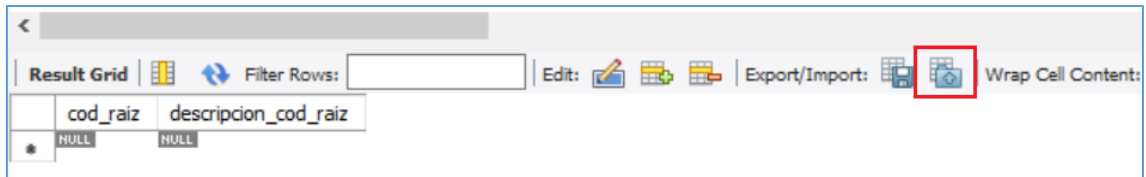
Paso 2. Verifico los datos que tiene las tablas mediante **select * from tabla_cod_interrupciones;** en este caso, todavía está vacía.

The screenshot shows the SQL Workbench interface with a query editor titled 'Query 1' and 'primer_entrega_proyecto_SQL*'. The query editor contains the following SQL code:

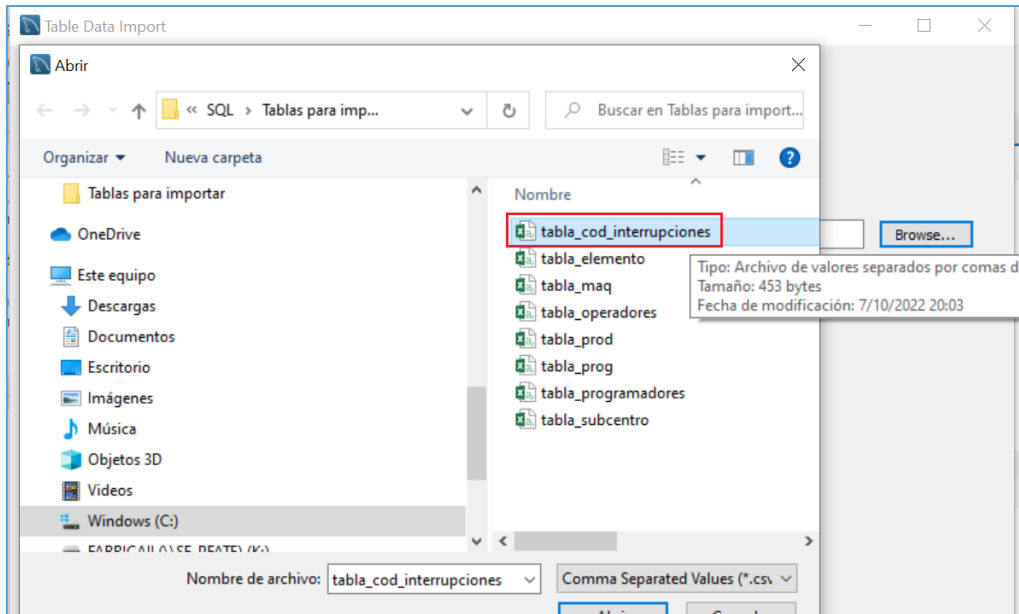
```
100 • );  
101  
102 • create table tabla_cod_interrupciones(  
103     cod_raiz int NOT NULL,  
104     descripcion_cod_raiz VARCHAR(100) not null,  
105     primary key (cod_raiz)  
106 • );  
107  
108 • select * from tabla_cod_interrupciones;  
109
```

The 'select * from tabla_cod_interrupciones;' statement on line 108 is highlighted with a red box. The toolbar above the editor includes icons for file operations, execution (lightning bolt), and other database functions.

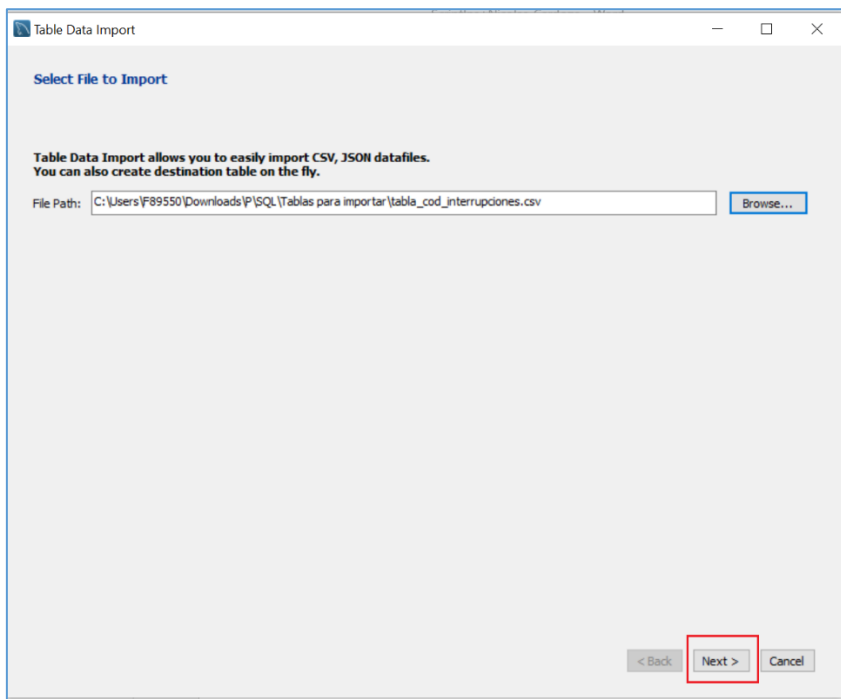
Paso 3. Utilizo el asistente de importación de Workbench



Paso 4. Busco mi tabla en formato .csv donde se encuentran mis datos



Paso 5. Next



Paso 6. Selecciono la opción de usar tabla existente y selecciono **proyecto.tabla_cod_interrupciones**, en este caso no uso Truncate table before import ya que habíamos corroborado que la tabla estaba vacía. Luego presiono en **Next**

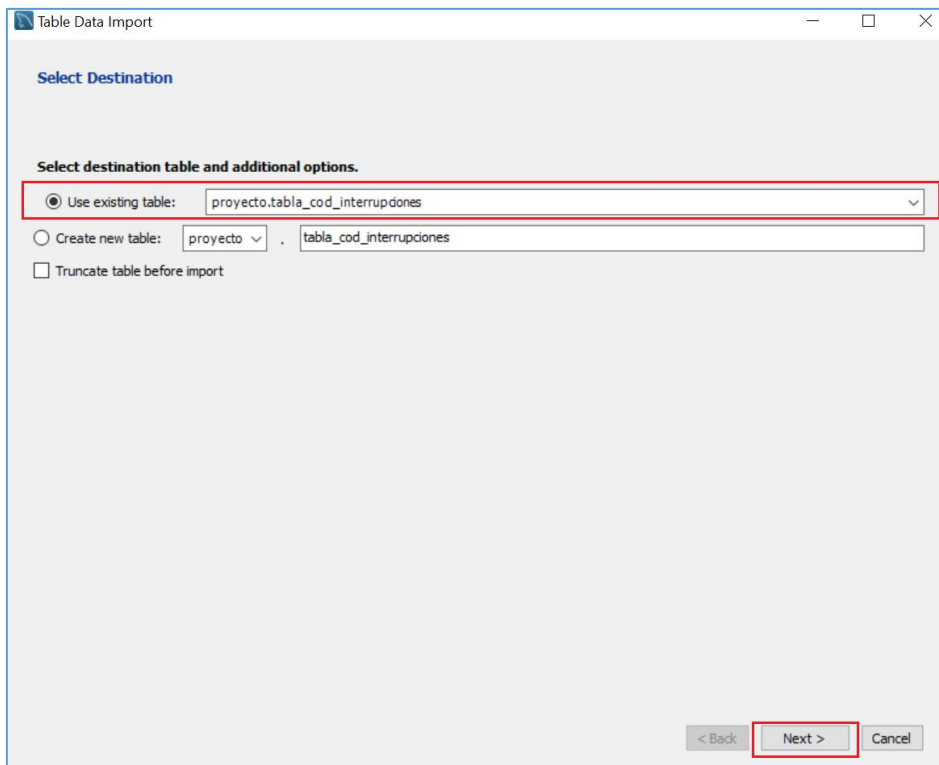


Table Data Import

Select Destination

Select destination table and additional options.

☒ Use existing table: proyecto.tabla_cod_interrupciones

☐ Create new table: proyecto . tabla_cod_interrupciones

☐ Truncate table before import

< Back Next > Cancel

Paso 7. Verifico la configuración de la importación, en este caso, no hace falta modificar nada. Luego presiono Next.

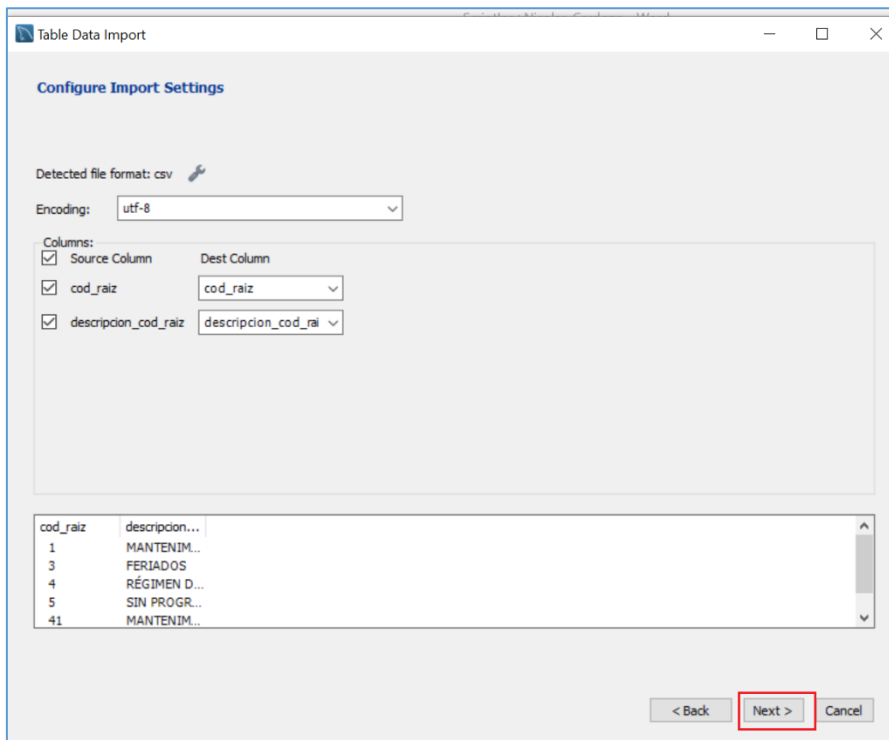


Table Data Import

Configure Import Settings

Detected file format: csv

Encoding: utf-8

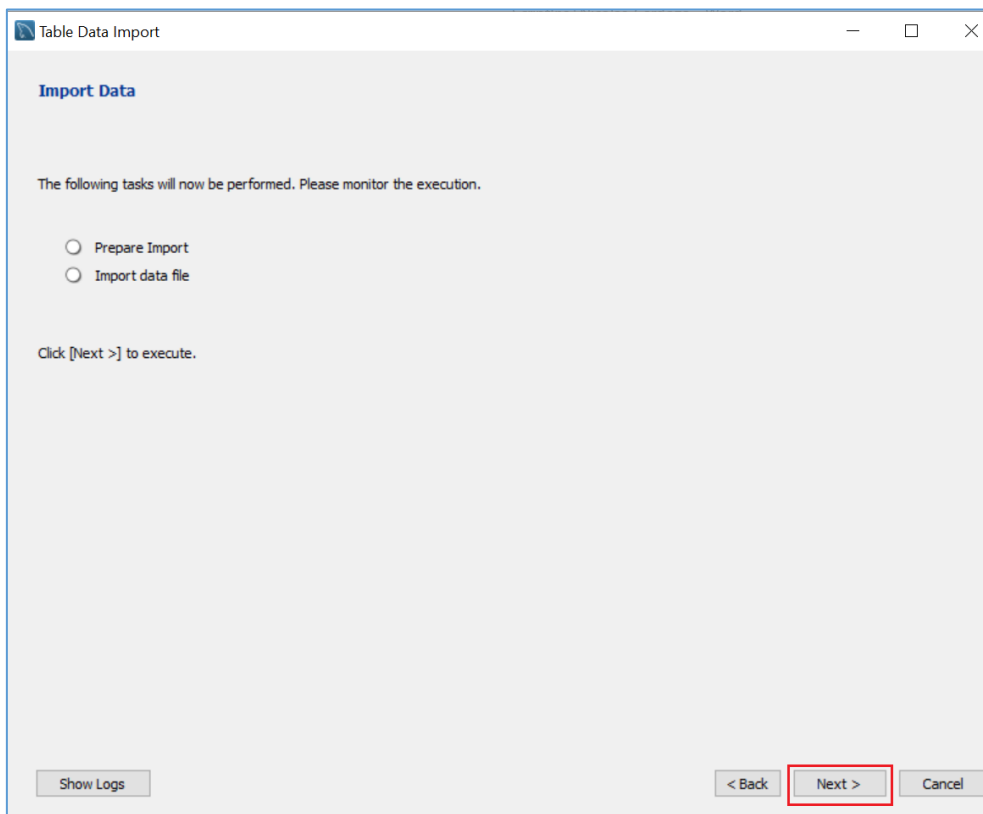
Columns:

Source Column	Dest Column
<input checked="" type="checkbox"/> cod_raiz	cod_raiz
<input checked="" type="checkbox"/> descripcion_cod_raiz	descripcion_cod_rai

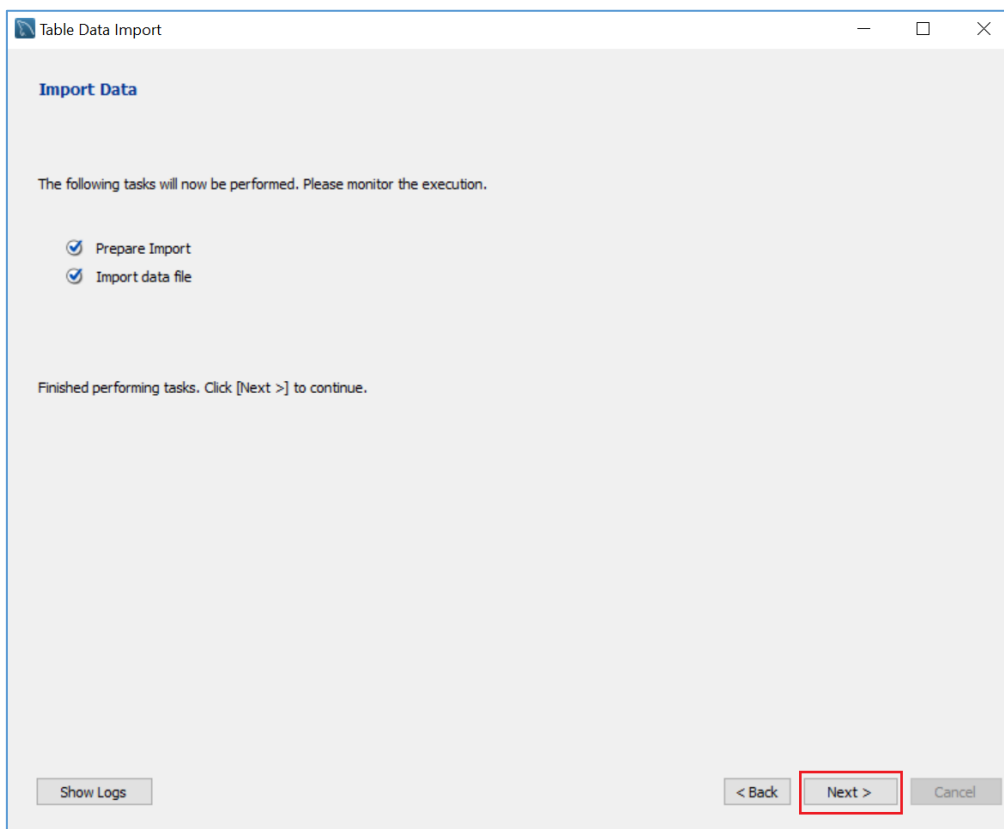
cod_raiz	descripcion...
1	MANTENIM...
3	FERIADOS
4	RÉGIMEN D...
5	SIN PROGR...
41	MANTENIM...

< Back Next > Cancel

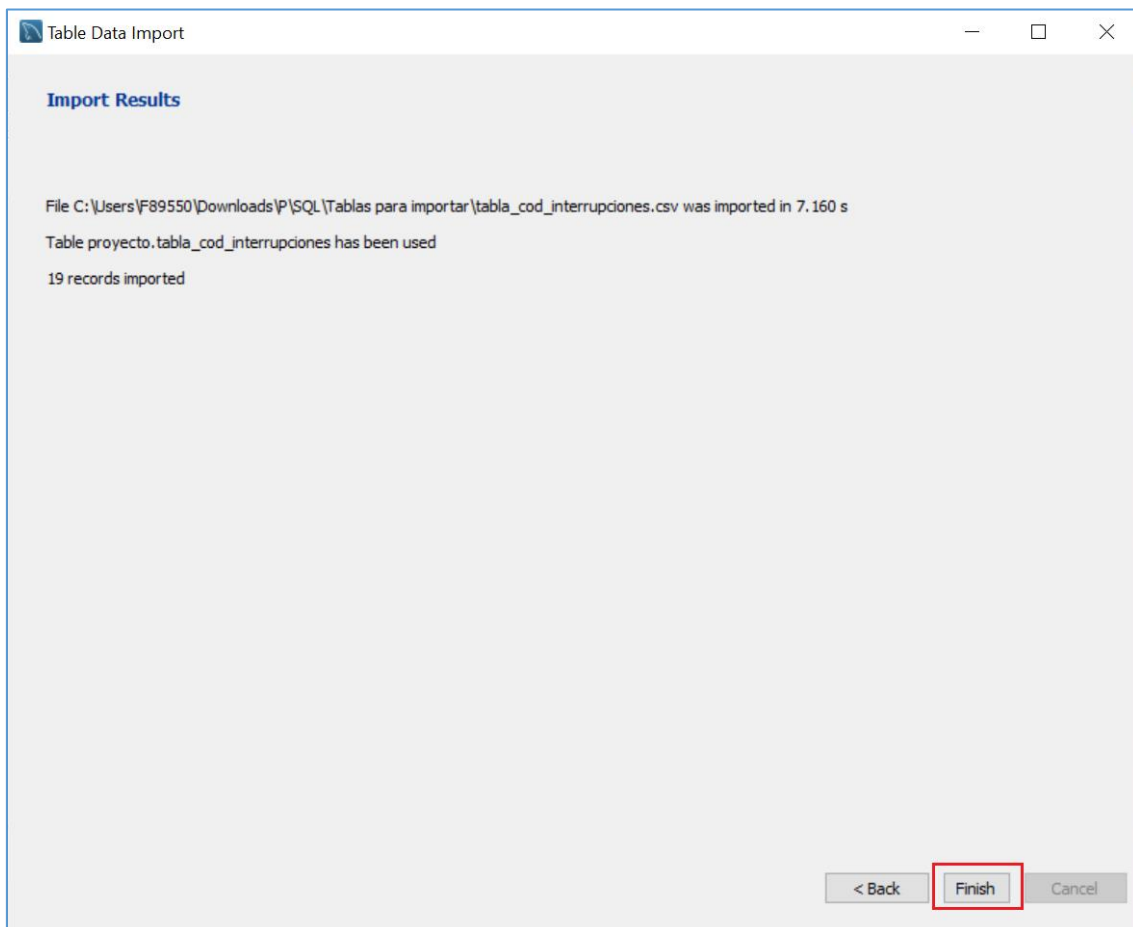
Paso 8. Nuevamente next para comenzar la importación



Paso 9. Finished Performed. Next.



Paso 10- Finish



A continuación se muestra evidencia de todas las tablas cargadas:

- Tabla_prog

	subcentro	maq	fecha	elemento	legajo_programador	programado
▶	BANBY	BY01	2022-07-01	LIMP	F20089001	3
	BANBY	BY01	2022-07-01	MB6210	F20089002	40
	BANBY	BY01	2022-07-01	MH2000	F20089003	16
	BANBY	BY01	2022-07-01	MH249	F20089001	25
	BANBY	BY01	2022-07-01	MR55	F20089002	1

- Tabla_prod

	subcentro	maq	fecha	elemento	legajo_operador	producido
	BANBY	BY01	2022-07-01	LIMP	F30089001	6
	BANBY	BY01	2022-07-01	MB6210	F30089002	40
	BANBY	BY01	2022-07-01	MH2000	F30089003	16
	BANBY	BY01	2022-07-01	MH249	F30089001	7
	BANBY	BY01	2022-07-01	XMH9200	F30089001	3
	-

- Tabla_maq

maq	descripcion
BY01	Banbury 01
BY02	Banbury 02
BY03	Banbury 03
TC02	Tobera Dual Farrel

- Tabla_subcentro

subcentro	descripcion_subcentro
BANBY	Zona de mezclado Banburys
TOBDIA	Tobera Tractor Diagonal

- Tabla_elemento

elemento	desc_elemento	tipo_elemento
C117	Elemento C117	C1
C1209	Elemento C1209	C1
C207	Elemento C207	C2
C259	Elemento C259	C2
CT223	Elemento CT223	CT

- Tabla_programadores

legajo_programador	nombre_programador	apellido_programador
F20089001	Carlos <small>RACA</small>	Perez
F20089002	Cristian <small>RACA</small>	Gonzalez
F20089003	Pedro <small>RACA</small>	Cardozo

- Tabla_operadores

legajo_operador	nombre_operador	apellido_operador
F30089001	Sebastian	Sabaleta
F30089002	Gonzalo	Riquelme
F30089003	Franco	Ortiz
NULL	NULL	NULL

- Tabla_interrupciones

subcentro	maq	fecha	cod_raiz	tiempo_int
BANBY	BY01	2022-07-01	42	25
BANBY	BY01	2022-07-01	43	15
BANBY	BY01	2022-07-01	45	49
BANBY	BY01	2022-07-01	48	27
BANBY	BY01	2022-07-01	49	6

- Tabla_cod_interrupciones

cod_raiz	descripcion_cod_raiz	autor	apellido_d
1	MANTENIMIENTO ANUAL		Sabaleta
3	FERIADOS		Riquelme
4	RÉGIMEN DE TURNOS		Ortiz
5	SIN PROGRAMA		
41	MANTENIMIENTO Y PROYECTOS		

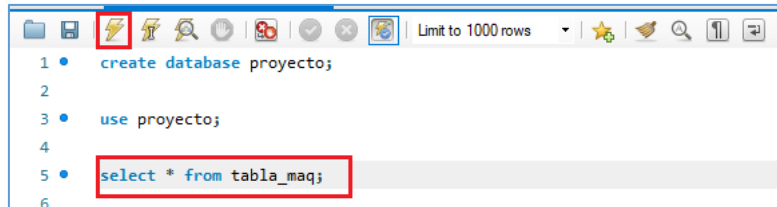
- Tabla_hora_prod

subcentro	maq	elemento	fecha	hora	duracion
BANBY	BY01	LIMP	2022-08-29	06:36:05	419
BANBY	BY01	MB223	2022-07-05	06:29:36	1132.83
BANBY	BY01	MB223	2022-07-05	06:33:39	4.05
BANBY	BY01	MB223	2022-07-05	06:36:08	2.48
BANBY	BY01	MB6210	2022-08-12	06:33:59	1194.97

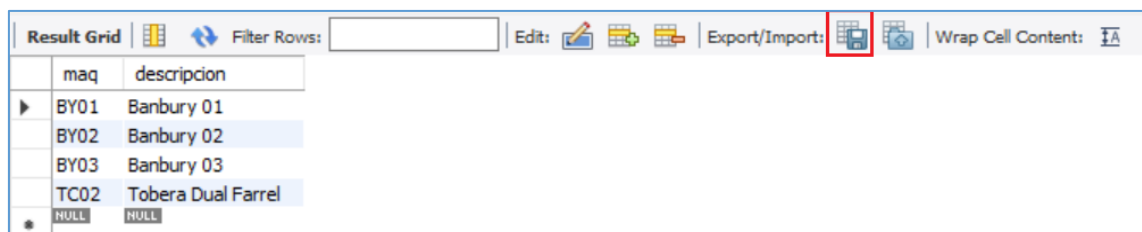
8) Paso a paso para la creación de los INSERT INTO de cada tabla y ejecutar todo desde un mismo script.

Mostraré el paso a paso con la importación de una de las tablas, siendo luego el mismo proceso para cada una de ellas. Se tomará de ejemplo tabla: **tabla_maq**.

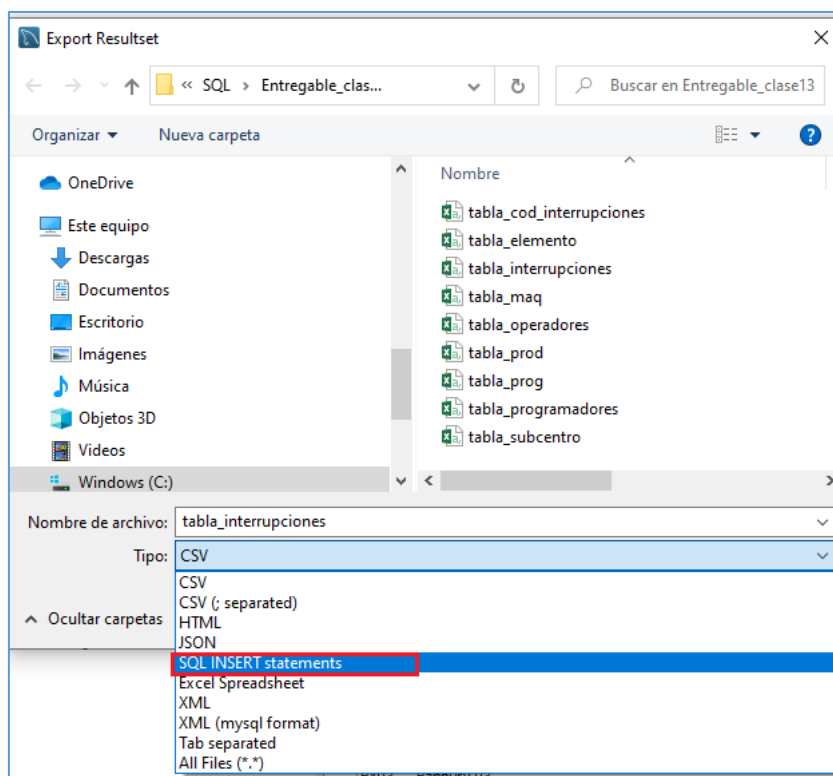
Paso 1. Ejecuto `SELECT * FROM tabla_maq;`



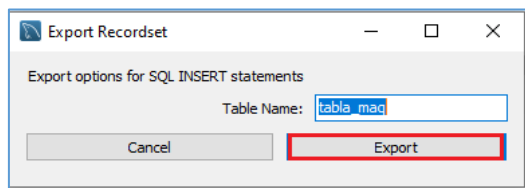
Paso 2. Export recorsset to an external file



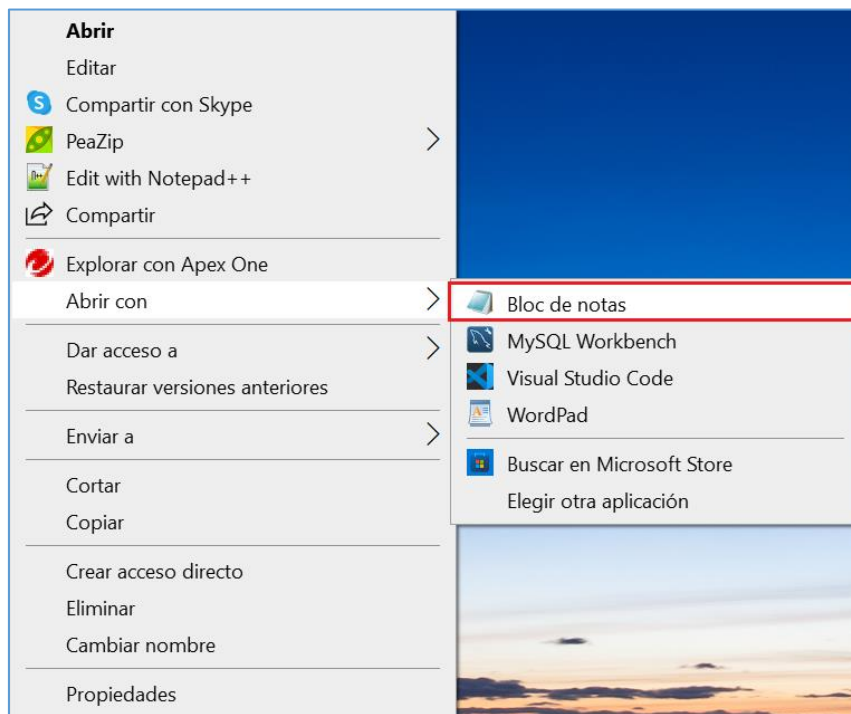
Paso 3. Guardo como SQL INSERT statements



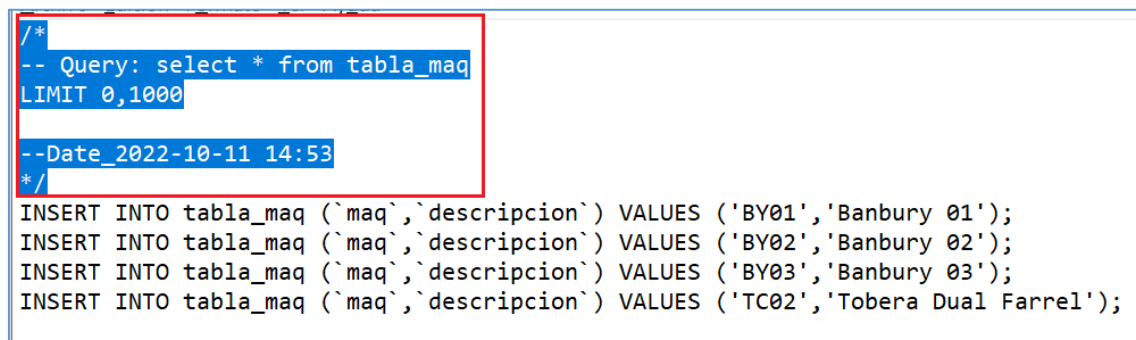
Paso 4. Export



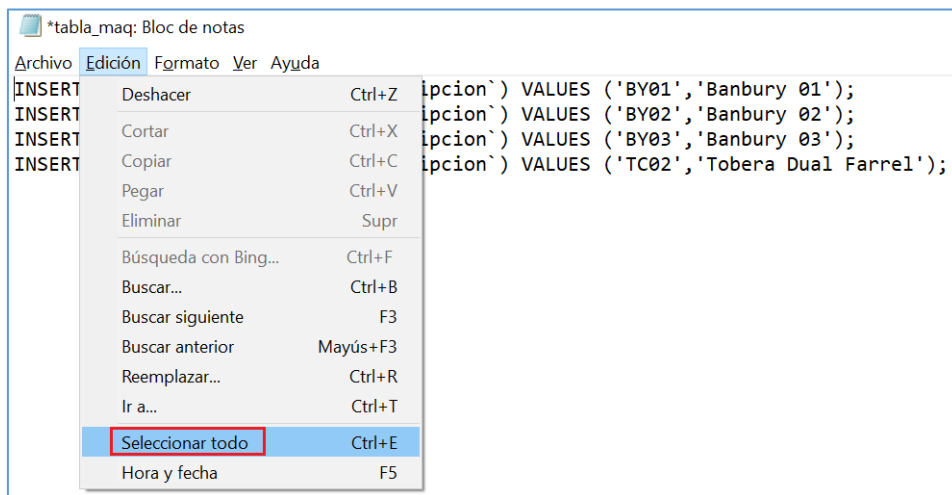
Paso 5. Voy a la carpeta donde lo guardé y abro con bloc de notas



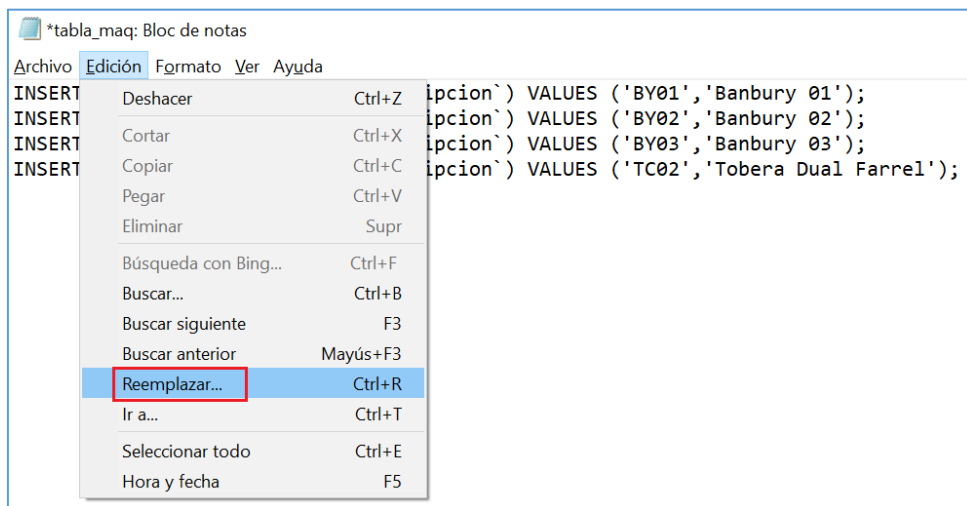
Paso 6. Elimino las primeras filas



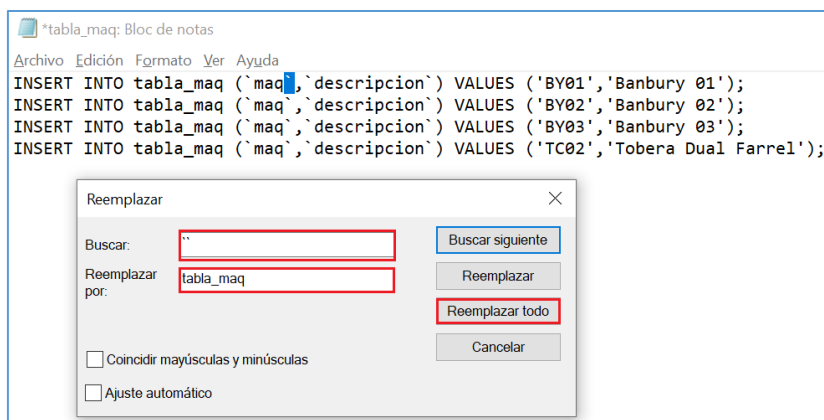
Paso 7. Edición -> Seleccionar todo



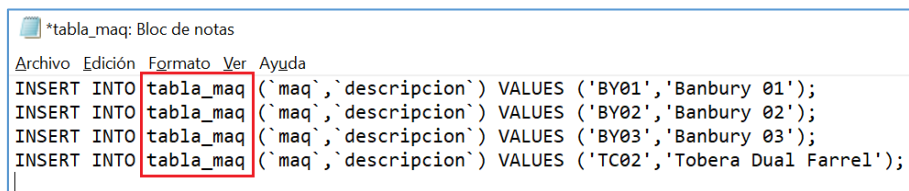
Paso 8. Edición -> Reemplazar



Paso 9. Reemplazo las dobles comillas por el nombre de mi tabla y hago click en el botón **Reemplazar todos**

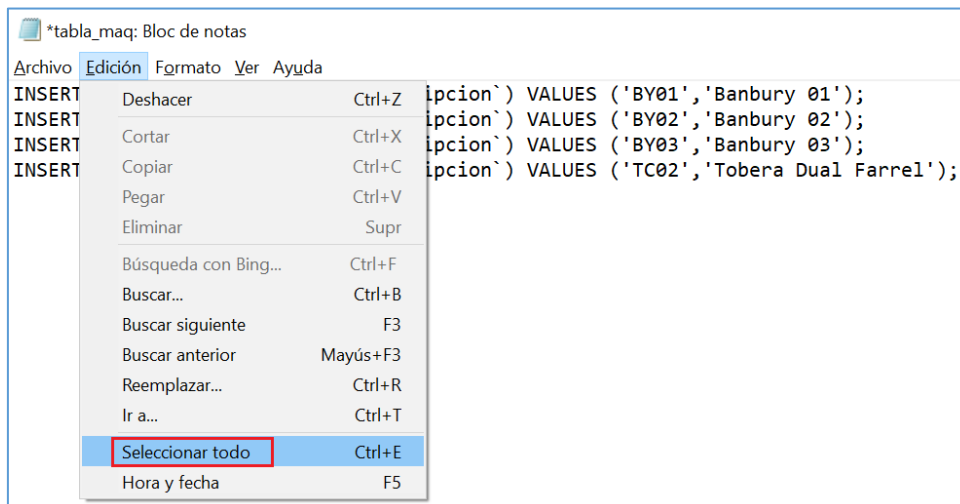


Paso 10. Verifico que se reemplazó correctamente



```
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('BY01','Banbury 01');
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('BY02','Banbury 02');
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('BY03','Banbury 03');
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('TC02','Tobera Dual Farrel');
```

Paso 11. Edición -> Seleccionar todo



```
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('BY01','Banbury 01');
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('BY02','Banbury 02');
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('BY03','Banbury 03');
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('TC02','Tobera Dual Farrel');
```

Paso 12. Copio y pego en mi script

```
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('BY01','Banbury 01');
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('BY02','Banbury 02');
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('BY03','Banbury 03');
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('TC02','Tobera Dual Farrel');
```

Paso 13. Repito para cada una de mis tablas.

9) Creación de vistas

- Creación de vista ***programa_vs_producido***. Contempla JOIN de tablas: ***tabla_prod*** y ***tabla_prog***

La misma nos mostrará por máquina y por fecha la suma de la cantidad programada de cada elemento para esa máquina y fecha y la suma de la cantidad producida de cada elemento para esa máquina y fecha.

maq	fecha	programado	producido
BY01	2022-07-01	87	72
BY01	2022-07-02	52	67
BY01	2022-07-03	74	54
BY01	2022-07-04	49	45
BY01	2022-07-05	87	53
BY01	2022-07-06	25	24
BY01	2022-07-07	89	62
BY01	2022-07-08	68	73

En la imagen son todos de maquina **BY01**, pero luego va pasando por otras máquinas.

- Creación de vista ***cumplimiento_programa_maquina_fecha***. Contempla JOIN de tablas: ***tabla_prod*** y ***tabla_prog***

Nos mostrará el cumplimiento del programa en [%] por cada máquina y fecha

maq	fecha	cump_programa
BY01	2022-07-01	82.7586
BY01	2022-07-02	128.8462
BY01	2022-07-03	72.9730
BY01	2022-07-04	91.8367
BY01	2022-07-05	60.9195

- Creación de vista ***cumplimiento_programa_maquina***. Contempla JOIN de tablas: ***tabla_prod*** y ***tabla_prog***

Nos mostrará el cumplimiento del programa en [%] por máquina en todo el período contemplado, sin discriminar por fecha.

	maq	cump_programa
►	BY01	93.9497
	BY03	87.7842
	TC02	103.2087

- Creación de vista ***producido_por_operador***. Contempla JOIN de tablas: ***tabla_prod*** y ***tabla_opreadores***

Nos muestra la cantidad producida por cada operador.

legajo_operador	nombre_operador	apellido_operador	cantidad_producida
F30089001	Sebastian	Sabaleta	9757
F30089002	Gonzalo	Riquelme	12213
F30089003	Franco	Ortiz	11027

- Creación de vista ***Porcentaje_maquina_parada***. Utiliza tabla ***tabla_interrupciones***

Nos muestra el tiempo calendario que tiene cada máquina entre las fechas contempladas, luego para el mismo período, los minutos que la máquina estuvo parada. Y finalmente en [%], la relación del tiempo de máquina parada y tiempo calendario.

subcentro	maq	tiempo_calendario	tiempo_maquina_parada	porcentaje_maquina_parada
BANBY	BY01	29760	18577	62.4227
BANBY	BY02	29760	29760	100.0000
BANBY	BY03	29760	17586	59.0927
BANBY	BY04	29760	14567	48.9483
BANBY	BY05	29760	14714	49.4422

10) Creación de funciones

- Creación de función ***cumplimiento_programa***.

La función es pasarle los parámetros de producido y programado y que calcule el % de cumplimiento de programa.

Es utilizada en la vista 'cumplimiento_programa_maquina_fecha'.

```
DELIMITER //
```

```
CREATE FUNCTION cumplimiento_programa(producido INT,programado INT) RETURNS FLOAT
```

```
DETERMINISTIC
```

```
BEGIN
```

```
    DECLARE relacion_programado_producido FLOAT;
```

```
    SET relacion_programado_producido = ((producido/programado) * 100 );
```

```
    RETURN relacion_programado_producido;
```

```
END
```

```
//
```

Uso de la función en la creación de una vista:

```
CREATE VIEW cumplimiento_programa_maquina_fecha AS(
```

```
SELECT coalesce(tabla_prog.maq,prod.maq) maq,
```

```
coalesce(tabla_prog.fecha,prod.fecha) fecha,
```

```
cumplimiento_programa(sum(producido),sum(programado)) cumplimiento_programa
```

```
FROM tabla_prog
```

```
RIGHT JOIN (select * from tabla_prod) PROD ON tabla_prog.maq=prod.maq
```

```
AND tabla_prog.fecha=prod.fecha AND tabla_prog.elemento = prod.elemento
```

```
group by maq,fecha
```

```
);
```

Resultado:

maq	fecha	cumplimiento_programa
BY01	2022-07-01	82.7586
BY01	2022-07-02	128.846
BY01	2022-07-03	72.973
BY01	2022-07-04	91.8367
BY01	2022-07-05	60.9195
BY01	2022-07-06	96
BY01	2022-07-07	69.6629
BY01	2022-07-08	107.353
BY01	2022-07-10	126.984

- Creación de funciones **obtener_detalle_nombre** y **obtener_detalle_apellido**

Las funciones obtienen valores de otra tabla, reemplazando a un JOIN.

Las mismas son utilizadas en vista 'producido_por_operador'.

```
DELIMITER //
CREATE FUNCTION obtener_detalle_nombre(dato varchar(10)) RETURNS VARCHAR(24)
DETERMINISTIC
BEGIN
    DECLARE dato_devuelto VARCHAR(24);
    SET dato_devuelto = (SELECT nombre_operador FROM proyecto.tabla_operadores WHERE legajo_operador=dato);
    RETURN dato_devuelto;
END
//

DELIMITER //
CREATE FUNCTION obtener_detalle_apellido(dato1 varchar(10)) RETURNS VARCHAR(24)
DETERMINISTIC
BEGIN
    DECLARE dato_devuelto1 VARCHAR(24);
    SET dato_devuelto1 = (SELECT apellido_operador FROM proyecto.tabla_operadores WHERE legajo_operador=dato1);
    RETURN dato_devuelto1;
END
//
```

Uso de las funciones en la creación de una vista:

```
CREATE VIEW producido_por_operador AS(
SELECT
legajo_operador,
obtener_detalle_nombre(legajo_operador) nombre_operador,
obtener_detalle_apellido(legajo_operador) apellido_operador,
SUM(producido) cantidad_producida
FROM tabla_prod
group by legajo_operador
);
```

El dato **legajo_operador** y la **cantidad_producida** son datos que obtengo de la tabla **tabla_prod**, ahora bien, no solo quiero ver el legajo del operador sino que también quiero ver su **nombre y apellido**, estos datos están en otra tabla llamada **tabla_operador**, entonces en vez de realizar el JOIN, llamo a las funciones mencionadas.

Resultado

legajo_operador	nombre_operador	apellido_operador	cantidad_producida
F30089001	Sebastian	Sabaleta	9757
F30089002	Gonzalo	Riquelme	12213
F30089003	Franco	Ortiz	11027

11) Creación de stored procedures

- Creación de stored procedure **ingreso_registro**.

El objetivo es pasarle al SP como parámetros los valores de un INSERT INTO para la creación de un nuevo registro en la tabla **tabla_programadores**, con la salvedad que si el legajo del programador ya existe, te devuelva la leyenda **“Legajo ya existe”**, caso contrario, el registro se ingresa correctamente.

```
DELIMITER //
CREATE PROCEDURE ingreso_registro(IN legajo varchar(10),IN nombre varchar(10),IN apellido varchar(10))
BEGIN
IF (SELECT count(legajo_programador) from tabla_programadores where legajo_programador=legajo)>0 THEN
SELECT 'Legajo ya existe';
ELSE
INSERT INTO tabla_programadores VALUES(legajo,nombre,apellido);
END IF;
END
//
```

Probamos llamando el SP con un legajo que no existe previamente:

```
call ingreso_registro ('F20089004','Sebastian','Afonso');
```

Realizamos el SELECT * FROM tabla_programadores;

```
select * from tabla_programadores;
```

Observamos como el registro se ingresó correctamente

legajo_programador	nombre_programador	apellido_programador
F20089001	Carlos	Perez
F20089002	Cristian	Gonzalez
F20089003	Pedro	Cardozo
F20089004	Sebastian	Afonso
NULL	NULL	NULL

Ahora bien, volvemos a llamar al SP con los mismos valores:

Observamos cómo nos devuelve que el legajo ya existe:

```
► Legajo ya existe
```

- Creación de stored procedure **cant_producida_ordenado**.

El objetivo es pasarle al SP un primer parámetro que representa el campo por el cual quiero ordenar una tabla, en este caso ejemplo, campo **"producido"**. Luego, mediante un segundo parámetro le defino si es ASC o DESC, es decir, obtendré mi tabla **"tabla_prod"** ordenada por cuando se produjo más o cuando se produjo menos.

```
DELIMITER //
CREATE PROCEDURE cant_producida_ordenado(IN columna VARCHAR(20), IN orden VARCHAR(10))
BEGIN
SET @columna=columna;
SET @orden=orden;
SET @q = 'SELECT * FROM tabla_prod ORDER BY';
SET @qfinal = concat(@q, ' ', @columna, ' ', @orden, ';');
PREPARE ejecutar FROM @qfinal;
EXECUTE ejecutar;
DEALLOCATE PREPARE ejecutar;
END
//
```

Probamos llamando el SP utilizando como primer parámetro el campo **"producido"** y como segundo parámetro **ASC**

```
call cant_producida_ordenado('producido','asc');
```

Efectivamente me ordena la tabla **tabla_prod** según el campo **producido** de manera **ASC**.

subcentro	maq	fecha	elemento	legajo_operador	producido
BANBY	BY01	2022-07-08	MT8221	F30089003	1
BANBY	BY01	2022-07-20	MT34	F30089001	1
BANBY	BY01	2022-07-21	MC11	F30089001	1
BANBY	BY01	2022-07-21	MC19	F30089002	1
BANBY	BY01	2022-08-12	C207	F30089002	1
BANBY	BY01	2022-08-16	C259	F30089002	1
BANBY	BY01	2022-08-25	MS619	F30089001	1
BANBY	BY03	2022-07-05	FT810	F30089003	1
BANBY	BY03	2022-07-17	FS618	F30089003	1
BANBY	BY04	2022-08-27	MT277	F30089002	1
BANBY	BY04	2022-08-29	MB1038	F30089002	1

12) Creación de triggers

- Creación de trigger **back_up_tabla_programadores**.

La función del trigger es crear una tabla espejo luego de la inserción de un registro en **tabla_programadores**. EVENT: **INSERT**. TIMING: **AFTER**

Paso 1. Creo la tabla espejo

```
create table tabla_programadores(  
legajo_programador varchar(10) NOT NULL,  
nombre_programador varchar(30) NOT NULL,  
apellido_programador varchar(30) NOT NULL,  
primary key (legajo_programador)  
);
```

Paso 2. Creo el trigger

```
CREATE TRIGGER back_up_tabla_programadores  
AFTER INSERT ON tabla_programadores  
FOR EACH ROW  
INSERT INTO tabla_programadores_back_up(legajo_programador,nombre_programador,apellido_programador)  
VALUES(NEW.legajo_programador,NEW.nombre_programador,NEW.apellido_programador);
```

Paso 3. Pruebo

Inserto un nuevo registro en tabla **tabla_programadores**

```
INSERT INTO tabla_programadores(legajo_programador,nombre_programador,apellido_programador) VALUES('F20089004','nombrePrueba','apellidoPrueba');
```

Reviso tabla original:

```
SELECT * FROM tabla_programadores;
```

Se agregó el registro correctamente:

legajo_programador	nombre_programador	apellido_programador
F20089001	Carlos	Perez
F20089002	Cristian	Gonzalez
F20089003	Pedro	Cardozo
F20089004	nombrePrueba	apellidoPrueba
NULL	NULL	NULL

Ahora revisemos tabla espejo:

```
SELECT * FROM tabla_programadores_back_up;
```

Observamos que también se agregó el registro correctamente siendo una copia de la tabla **tabla_programadores**

legajo_programador	nombre_programador	apellido_programador
F20089001	Carlos	Perez
F20089002	Cristian	Gonzalez
F20089003	Pedro	Cardozo
F20089004	nombrePrueba	apellidoPrueba
NULL	NULL	NULL

- Creación de trigger **back_up_tabla_cod_interrupciones**.

La función del trigger es crear una tabla espejo luego de la inserción de un registro en **tabla_cod_interrupciones**. EVENT: **INSERT**. TIMING: **AFTER**

Paso 1. Creo la tabla espejo

```
create table tabla_cod_interrupciones_back_up(
cod_raiz int NOT NULL,
descripcion_cod_raiz VARCHAR(100) not null,
primary key (cod_raiz)
);
```

Paso 2. Creo el trigger

```
CREATE TRIGGER back_up_tabla_cod_interrupciones
AFTER INSERT ON tabla_cod_interrupciones
FOR EACH ROW
INSERT INTO tabla_cod_interrupciones_back_up(cod_raiz,descripcion_cod_raiz)
VALUES(NEW.cod_raiz,NEW.descripcion_cod_raiz);
```

Paso 3. Pruebo

Inserto un nuevo registro en tabla **tabla_cod_interrupciones**

```
INSERT INTO tabla_cod_interrupciones(cod_raiz,descripcion_cod_raiz) VALUES('150','codPrueba');
```

Reviso tabla original:

```
select * from tabla_cod_interrupciones;
```

Se agregó el registro correctamente:

cod_raiz	descripcion_cod_raiz
100	VELOCIDAD REDUCIDA
101	SIN REGISTRO
110	MICROINTERRUPCIONES
150	codPrueba

Ahora revisemos tabla espejo:

```
select * from tabla_cod_interrupciones_back_up;
```

Observamos que también se agregó el registro correctamente siendo una copia de la tabla **tabla_cod_interrupciones**

cod_raiz	descripcion_cod_raiz
100	VELOCIDAD REDUCIDA
101	SIN REGISTRO
110	MICROINTERRUPCIONES
150	codPrueba

- Creación de trigger **upload_log_tabla_programadores**

La función del trigger es crear una tabla bitácora antes de la actualización de un registro en **tabla_programadores**. EVENT: **UPDATE**. TIMING: **BEFORE**

Paso 1. Creo la tabla bitácora

```
CREATE TABLE tabla_programadores_bitacora(
id_bitacora INT NOT NULL AUTO_INCREMENT,
fecha DATE NOT NULL,
hora TIME NOT NULL,
usuario_id VARCHAR(45),
old_nombre VARCHAR(30),
new_nombre VARCHAR(30),
old_apellido VARCHAR(30),
new_apellido VARCHAR(30),
tipo_operacion varchar(100),
primary key(id_bitacora));
```

Paso 2. Creo el trigger

```
CREATE TRIGGER update_log_tabla_programadores
BEFORE UPDATE ON tabla_programadores
FOR EACH ROW
INSERT INTO tabla_programadores_bitacora
(fecha,hora,usuario_id,old_nombre,new_nombre,old_apellido,new_apellido,tipo_operacion)
VALUES
(curdate(),curtime(),session_user(),OLD.nombre_programador,NEW.nombre_programador,OLD.apellido_programador,NEW.apellido_programador,'Actualizacion');
```

Paso 3. Pruebo

Actualizo un registro en tabla **tabla_programadores**

```
update tabla_programadores set apellido_programador='DiMarzio' where legajo_programador='F20089003';
```

Reviso tabla original:

```
select * from tabla_programadores;
```

Se actualizó el registro correctamente:

legajo_programador	nombre_programador	apellido_programador
F20089001	Carlos	Perez
F20089002	Cristian	Gonzalez
F20089003	Pedro	DiMarzio
NULL	NULL	NULL

Ahora revisemos tabla bitácora:

```
select * from tabla_programadores_bitacora;
```

Observamos cómo se completa la tabla bitácora **tabla_programadores_bitacora**

id_bitacora	fecha	hora	usuario_id	old_nombre	new_nombre	old_apellido	new_apellido	tipo_operacion
1	2022-10-21	10:19:11	root@localhost	Pedro	Pedro	Cardozo	DiMarzio	Actualizacion
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Creación de trigger ***delete_log_tabla_cod_interrupciones***

La función del trigger es crear una tabla bitácora antes de la eliminación de un registro en ***tabla_cod_interrupciones*** EVENT: ***DELETE***. TIMING: ***BEFORE***

Paso 1. Creo la tabla bitácora

```
CREATE TABLE tabla_cod_interrupciones_bitacora(
id_bitacora INT NOT NULL AUTO_INCREMENT,
fecha DATE NOT NULL,
hora TIME NOT NULL,
usuario_id VARCHAR(45),
old_cod VARCHAR(30),
old_desc VARCHAR(30),
tipo_operacion varchar(100),
primary key(id_bitacora));
```

Paso 2. Creo el trigger

```
CREATE TRIGGER delete_log_tabla_cod_interrupciones
BEFORE DELETE ON tabla_cod_interrupciones
FOR EACH ROW
INSERT INTO tabla_cod_interrupciones_bitacora
(fecha,hora,usuario_id,old_cod,old_desc,tipo_operacion)
VALUES
(curdate(),curtime(),session_user(),OLD.cod_raiz,OLD.descripcion_cod_raiz,'Delete');
```

Paso 3. Pruebo

Elimino un registro en tabla ***tabla_cod_interrupciones***

```
DELETE FROM tabla_cod_interrupciones WHERE cod_raiz=160;
```

Reviso tabla original:

```
select * from tabla_programadores;
```

Se eliminó el registro correctamente:

cod_raiz	descripcion_cod_raiz
99	OTROS
100	VELOCIDAD REDUCIDA
101	SIN REGISTRO
110	MICROINTERRUPCIONES
NULL	NULL

Ahora revisemos tabla bitácora:

```
select * from tabla_cod_interrupciones_bitacora;
```

Observamos cómo se completa la tabla bitácora ***tabla_cod_interrupciones_bitacora***

id_bitacora	fecha	hora	usuario_id	old_cod	old_desc	tipo_operacion
1	2022-10-21	10:14:48	root@localhost	150	codPrueba	Delete
2	2022-10-21	10:25:36	root@localhost	160	Presion	Delete
NULL	NULL	NULL	NULL	NULL	NULL	NULL

13) Sentencias TCL

- Inicio la transacción antes de los INSERT INTO

```
START TRANSACTION;
```

Creo 1 savepoint antes de iniciar los INSERT INTO en cada tabla.

Ejemplo de SAVEPOINT ***pre_tabla_maq*** antes de tabla ***tabla_maq***

```
START TRANSACTION;

SAVEPOINT pre_tabla_maq;
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('BY01','Banbury 01');
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('BY02','Banbury 02');
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('BY03','Banbury 03');
INSERT INTO tabla_maq (`maq`,`descripcion`) VALUES ('TC02','Tobera Dual Farrel');
```

Luego de los INSERT INTO de cada tabla coloco sentencia COMMIT;

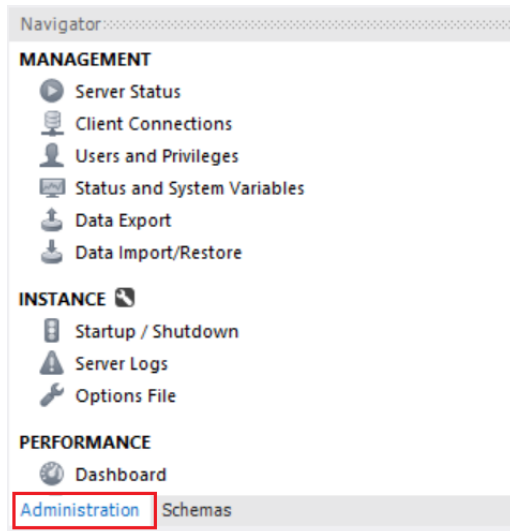
```
COMMIT;
```

14) Backup

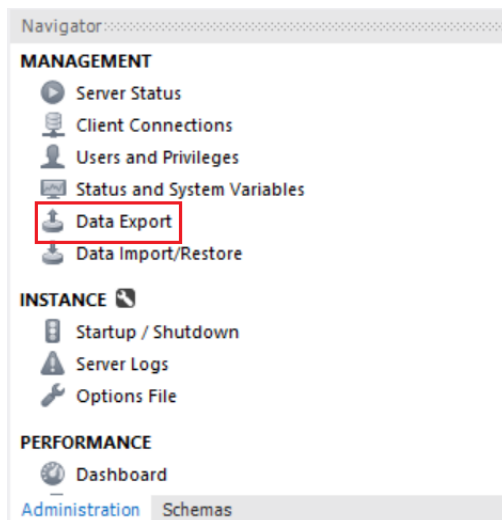
Se realizará Backup de las tablas más importantes de mi base de datos, siendo estas:

- Tabla_interrupciones
- Tabla_prod
- Tabla_prog

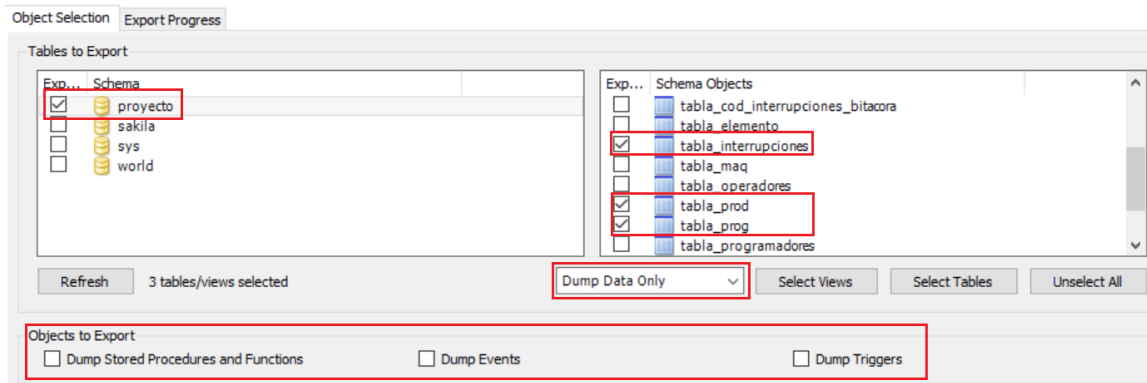
Paso 1. Solapa **Administration**



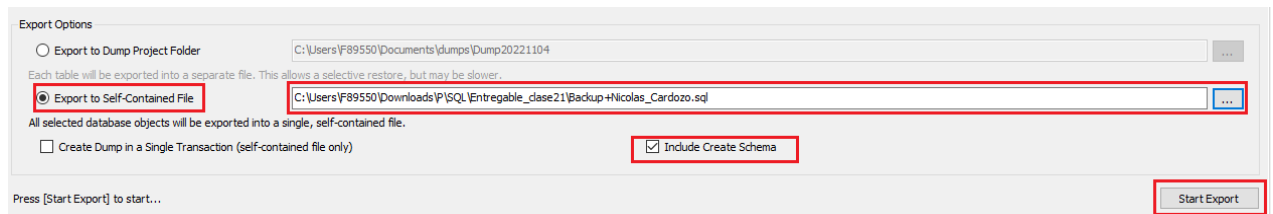
Paso 2. Data Export



Paso 3. Selecciono la base de datos que quiero hacer Backup, en este caso, **proyecto**, luego las tablas a las que deseo realizar Backup, **tabla_interrupciones**, **tabla_prog**, **tabla_prod**. Dentro de las opciones selecciono **Dump Data Only**. Y para este caso particular no selecciono **Dump Stored Procedures and Functions**, ni **Dump Events**, ni **Dump Triggers**



Paso 4. En la sección Export Options selecciono **Export to Self-Contained File**, selecciono la carpeta donde quiero que me guarde el Backup, selecciono **Include Create Schema**, y finalmente **Start Export**



15) Herramientas y tecnologías

Las herramientas utilizadas han sido:

- Google Drive.
- Archivos Excel descargados de la compañía para la obtención de los datos.
- Archivos .csv utilizados para la importación de los datos.
- MySQL Workbench
- Plataforma Coderhouse
- Microsoft Word
- PDF
- Zoom