

# ***CODER HOUSE***

**Curso: Lenguaje SQL**

***Proyecto: Confección de una base de datos  
en SQL para AVANTI MAYORISTA***



**Alumno: Juan Ignacio Rabazzi**

**Docente: Miguel Rodas**

**Tutor: David Canchi**

## Contenido

1. Introducción .....	3
2. Objetivo.....	3
3. Modelo de negocio .....	3
4. Situación problemática.....	3
5. Descripción de Tablas.....	3
6. Diagrama Entidad – Relación .....	6
7. Creación de tablas en SQL.....	8
8. Inserción de datos en SQL .....	9
9. Vistas en SQL.....	10
10. Funciones en SQL .....	11
11. Procedimientos almacenados en SQL .....	11
12. Triggers en SQL.....	13
13. Data Control Language.....	14
14. Transaction Control Language .....	15
15. Backup y restauración .....	17
16. Herramientas y tecnologías utilizadas .....	18

## 1. Introducción

El presente informe corresponde al Proyecto Final del curso de SQL dictado en CoderHouse. En el mismo se desarrollará la confección de una base de datos de una empresa que se dedica a la venta de diversos productos alimenticios.

## 2. Objetivo

El objetivo del proyecto final será la confección de una base de datos relacional en SQL para la Empresa: Avanti Mayorista.

## 3. Modelo de negocio

Avanti Mayorista es una Empresa que se dedica a la venta de alimentos, principalmente lácteos, fiambres y delicatessen. Se encuentra situada en la ciudad de Santa Fe, Argentina y cuenta con 2 sucursales. Sus principales clientes son bares y comercios de la región.

El dataset fue brindado por la Organización, pero se realizaron modificaciones sobre los registros para proteger la información de esta. Actualmente quien realiza este informe desempeña tareas contables y de administración en la compañía.

## 4. Situación problemática

Avanti Mayorista es una Organización que actualmente genera una gran cantidad de datos de diversas fuentes que no se encuentran estructurados. Con este proyecto se busca poder organizar su información en una base de datos relacional con el objetivo de poder extraer información de los datos.

## 5. Descripción de Tablas

A continuación, se muestran las tablas que serán desarrolladas luego en SQL, con una breve introducción de los datos que se alojarán en ellas.

Tabla: <u><b>Clientes</b></u>				
<i><u>Descripción:</u> esta tabla contiene información personal de los clientes.</i>				
Nombre del campo	Tipo de campo	NOT NULL	PK	FK
ID_Cliente	INT	X	X	
Nombre	VARCHAR(255)	X		
CUIT	INT	X		
Domicilio	VARCHAR(255)	X		
Cod_postal	INT	X		

Tabla: <b><u>Proveedores</u></b>				
<i>Descripción: esta tabla contiene información personal de los proveedores.</i>				
Nombre del campo	Tipo de campo	NOT NULL	PK	FK
ID_Proveedor	INT	X	X	
Nombre	VARCHAR(255)	X		
CUIT	INT	X		
Domicilio	VARCHAR(255)	X		
Cod_postal	INT	X		

Tabla: <b><u>Clase Articulos</u></b>				
<i>Descripción: esta tabla contiene las diferentes clases de artículos que la Empresa comercializa.</i>				
Nombre del campo	Tipo de campo	NOT NULL	PK	FK
ID_Clase	INT	X	X	
Nombre	VARCHAR(255)	X		

Tabla: <b><u>Marca Articulos</u></b>				
<i>Descripción: esta tabla contiene todas las marcas de los artículos que la Empresa comercializa.</i>				
Nombre del campo	Tipo de campo	NOT NULL	PK	FK
ID_Marca	INT	X	X	
Nombre	VARCHAR(255)	X		

Tabla: <b><u>Articulos</u></b>				
<i>Descripción : esta tabla contiene toda la información de los artículos que la Empresa comercializa, también se detalla si el producto es de venta al peso (por kg) o por unidad y se encuentra activo o suspendido.</i>				
Nombre del campo	Tipo de campo	NOT NULL	PK	FK
ID_Articulo	INT	X	X	
Descripcion	VARCHAR(255)	X		
Unidad	ENUM("kg","un")	X		
Estado	ENUM("ACTIVO","SUSPENDIDO")	X		
Cod_barra	INT			
Proveedor_id	INT	X		X
Marca_id	INT	X		X
Clase_id	INT	X		X

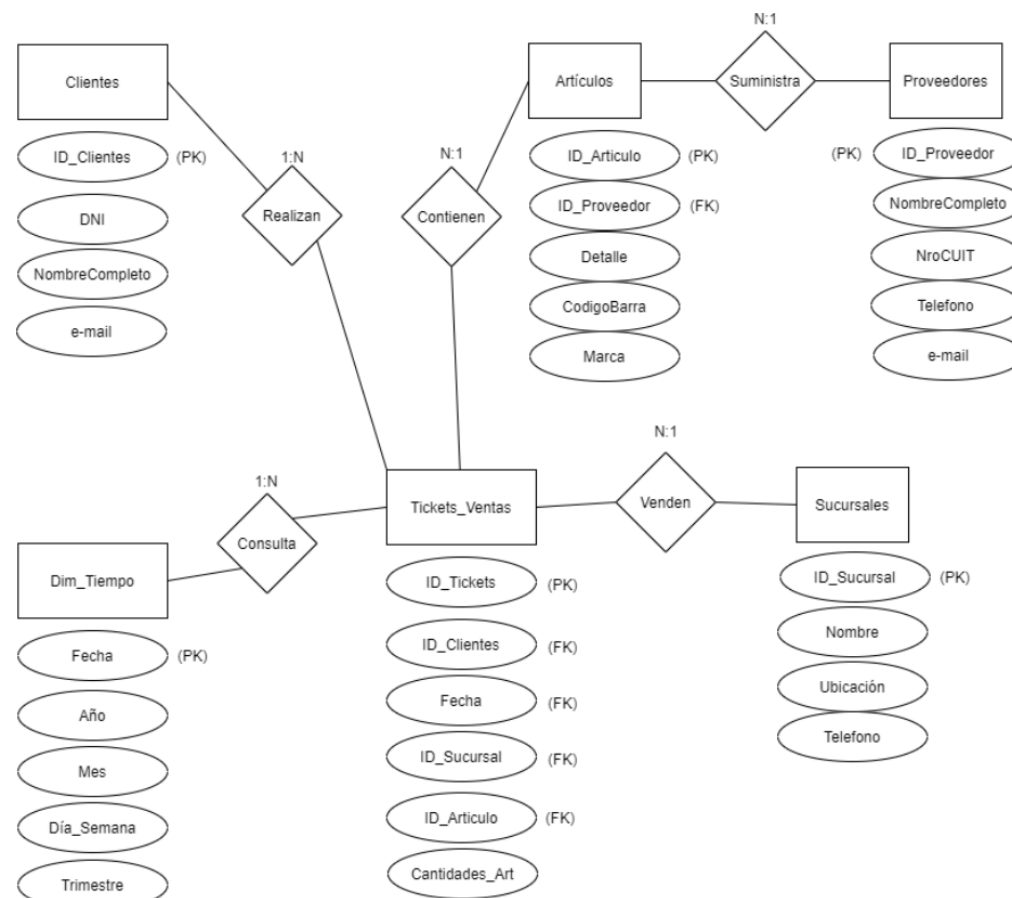
Tabla: <b>Dim Tiempo</b>				
<i>Descripción: esta tabla contiene fechas que serán de utilidad para analizar y agrupar los tickets de venta de la Empresa.</i>				
Nombre del campo	Tipo de campo	NOT NULL	PK	FK
Fecha	DATE	X	X	
Año	INT	X		
Mes	VARCHAR(255)	X		
Dia_semana	VARCHAR(255)	X		
Trimestre	INT	X		

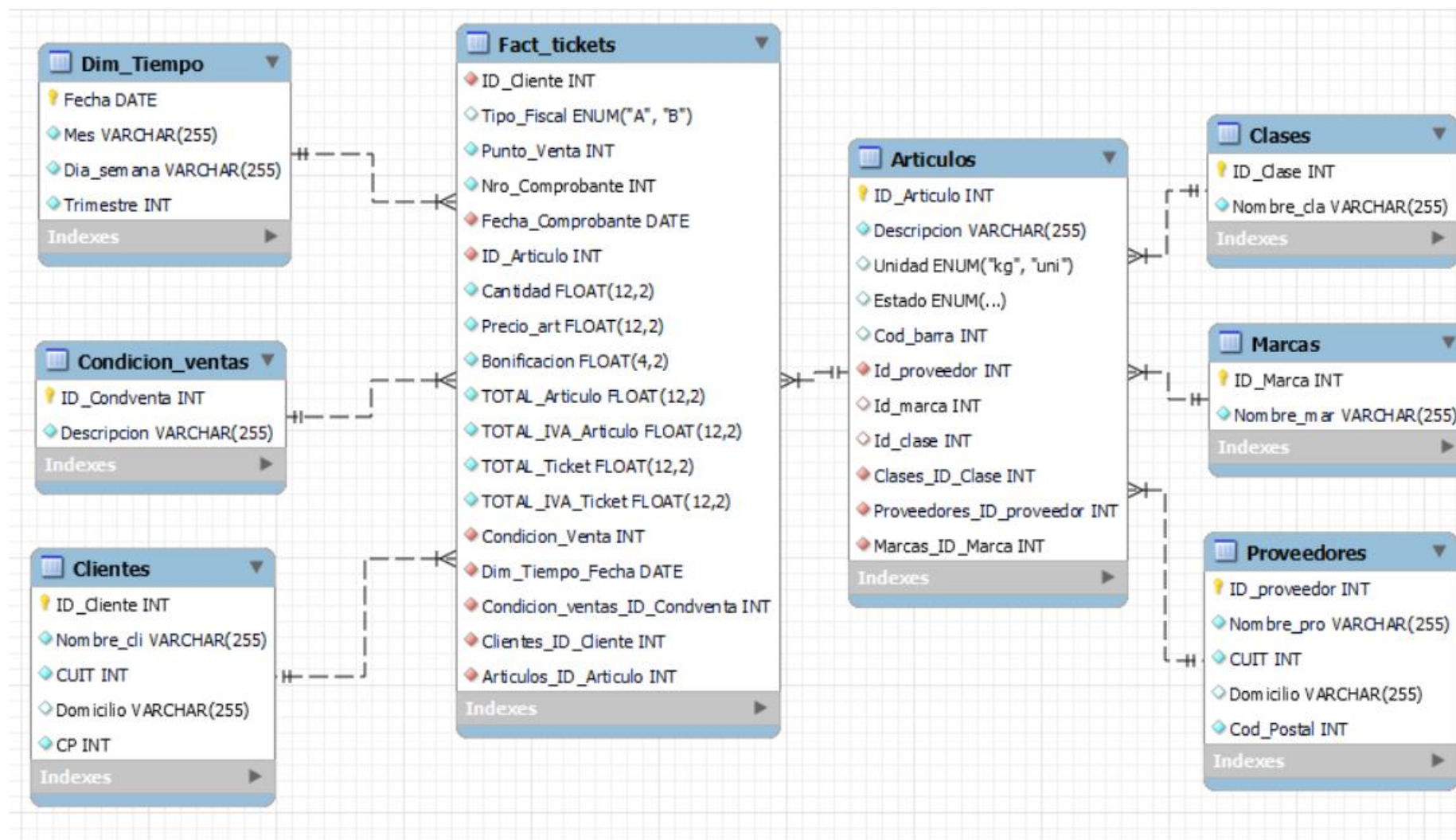
Tabla: <b>Condicion_Venta</b>				
<i>Descripción: esta tabla contiene información de la condición de pago del ticket (Contado, Cuenta corriente).</i>				
Nombre del campo	Tipo de campo	NOT NULL	PK	FK
Codigo	INT	X	X	
Descripcion	INT	X		

Tabla: <b>Fact Tickets</b>				
<i>Descripción: esta tabla contiene información de los tickets de venta emitidos por la Empresa, tiene como datos la fecha en que se efectuó la venta, el punto de venta, el tipo fiscal, el nro de comprobante, el cliente, los artículos y cantidades. También tiene información del importe total pagado por artículo y por el total del ticket. La tabla cuenta con una primary key que se forma de la combinación de las columnas Tipo_Fiscal, PuntoVenta, Nro_Comprobante, ID_Articulo.</i>				
Nombre del campo	Tipo de campo	NOT NULL	PK	FK
ID_Cliente	INT	X		X
Tipo_Fiscal	ENUM("A","B")	X		
Nro_Comprobante	INT	X		
Fecha_Comprobante	DATE	X		
ID_Articulo	INT	X		X
Cantidad	FLOAT(12,2)	X		
Precio_unitario	FLOAT(12,2)	X		
TOTAL_Articulo	FLOAT(12,2)	X		
TOTAL_Ticket	FLOAT(12,2)	X		
Condicion_Venta	INT	X		X
PRIMARY KEY: CONCAT(Tipo_Fiscal, PuntoVenta, Nro_Comprobante, ID_Articulo)				

## 6. Diagrama Entidad – Relación

A continuación, se muestran dos imágenes del diagrama de Entidad – Relación del proyecto. La primera imagen fue realizada en la web diagrams.net y se realizó al comienzo del curso (luego se realizaron modificaciones sobre los campos de las tablas). La segunda imagen contiene el Diagrama E – R final y se realizó a partir de un Script en SQL en MySQL Workbench.





## 7. Creación de tablas en SQL

En este apartado se procedió a la creación de las tablas del proyecto. En este caso fueron 8 tablas. Se adjunta en el documento un Script SQL con la creación de estas.

```
CREATE TABLE IF NOT EXISTS Clientes(  
    ID_Cliente INT PRIMARY KEY NOT NULL AUTO_INCREMENT,  
    Nombre_cli VARCHAR(255) NOT NULL,  
    CUIT DOUBLE NOT NULL,  
    Domicilio VARCHAR(255),  
    CP INT NOT NULL);  
  
CREATE TABLE IF NOT EXISTS Marcas(  
    ID_Marca INT PRIMARY KEY NOT NULL AUTO_INCREMENT,  
    Nombre_mar VARCHAR(255) NOT NULL);  
  
CREATE TABLE IF NOT EXISTS Clases(  
    ID_Clase INT PRIMARY KEY NOT NULL AUTO_INCREMENT,  
    Nombre_cla VARCHAR(255) NOT NULL);  
  
CREATE TABLE IF NOT EXISTS Proveedores(  
    ID_proveedor INT PRIMARY KEY NOT NULL AUTO_INCREMENT,  
    Nombre_pro VARCHAR(255) NOT NULL,  
    CUIT DOUBLE NOT NULL,  
    Domicilio VARCHAR(255),  
    Cod_Postal INT NOT NULL);  
  
CREATE TABLE IF NOT EXISTS Articulos(  
    ID_Articulo INT NOT NULL AUTO_INCREMENT,  
    Descripcion VARCHAR(255) NOT NULL,  
    Unidad ENUM("kg", "uni"),  
    Estado ENUM("ACTIVO", "SUSPENDIDO"),  
    Cod_barra DOUBLE,  
    Id_proveedor INT NOT NULL,  
    Id_marca INT,  
    Id_clase INT,  
    PRIMARY KEY (ID_Articulo),  
    FOREIGN KEY (Id_proveedor) REFERENCES avanti_coder.Proveedores(ID_Proveedor),  
    FOREIGN KEY (Id_marca) REFERENCES avanti_coder.Marcas(ID_Marca),  
    FOREIGN KEY (Id_clase) REFERENCES avanti_coder.Clases(ID_Clase));  
  
CREATE TABLE IF NOT EXISTS Dim_Tiempo(  
    Fecha DATE NOT NULL PRIMARY KEY,  
    Año VARCHAR(255) NOT NULL,  
    Mes VARCHAR(255) NOT NULL,  
    Dia_semana VARCHAR(255) NOT NULL,  
    Trimestre INT NOT NULL);
```



```
CREATE TABLE IF NOT EXISTS Condicion_ventas(  
    ID_Condventa INT PRIMARY KEY NOT NULL AUTO_INCREMENT,  
    Descripcion VARCHAR(255) NOT NULL);  
  
CREATE TABLE IF NOT EXISTS Fact_tickets(  
    ID_Cliente INT NOT NULL,  
    Tipo_Fiscal ENUM("A","B"),  
    Punto_Venta INT NOT NULL,  
    Nro_Comprobante INT NOT NULL,  
    Fecha_Comprobante DATE NOT NULL,  
    ID_Articulo INT NOT NULL,  
    Cantidad FLOAT(12,2) NOT NULL,  
    Precio_unitario FLOAT(12,2) NOT NULL,  
    TOTAL_Articulo FLOAT(12,2) NOT NULL,  
    TOTAL_Ticket FLOAT(12,2) NOT NULL,  
    Condicion_Venta INT NOT NULL,  
    FOREIGN KEY (ID_Cliente) REFERENCES avanti_coder.Clientes(ID_Cliente),  
    FOREIGN KEY (Fecha_Comprobante) REFERENCES avanti_coder.Dim_Tiempo(Fecha),  
    FOREIGN KEY (ID_Articulo) REFERENCES avanti_coder.Articulos(ID_Articulo),  
    FOREIGN KEY (Condicion_Venta) REFERENCES avanti_coder.Condicion_Ventas(ID_Condventa),  
    PRIMARY KEY CONCAT(ID_Cliente,Tipo_Fiscal,Punto_Venta,Nro_Comprobante,ID_Articulo));
```

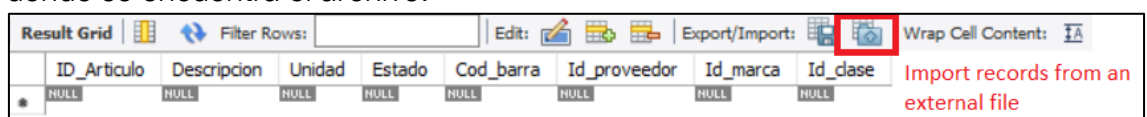
## 8. Inserción de datos en SQL

Se procedió a realizar la inserción de datos dentro de las tablas. Algunos registros fueron realizados manualmente otros mediante importación de archivos CSV. Se adjunta en el documento un Script SQL con la inserción de los registros en las tablas.

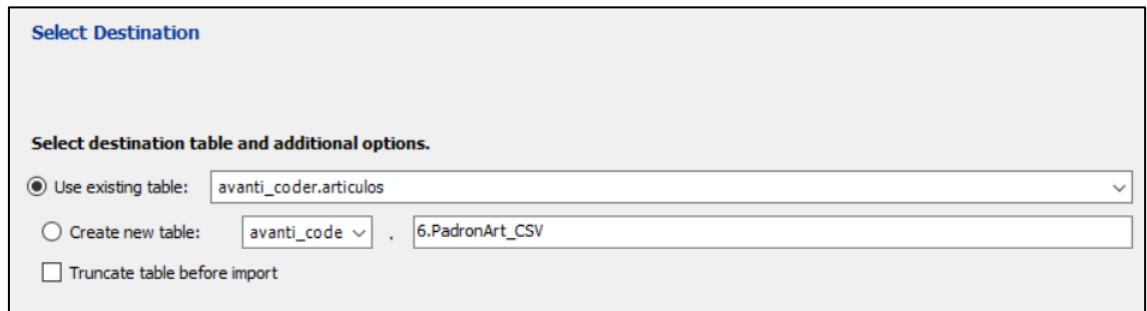
Los datos que se incorporan al proyecto corresponden a los artículos, clientes y proveedores actuales de la Empresa cuyos datos fueron modificados ya que son información sensible de la Organización. Por otro lado, los registros importados en la tabla “Fact\_tickets” corresponden a las facturas emitidas de enero a diciembre de 2021.

En las tablas “Articulos”, “Clientes”, “Marcas”, “Proveedores”, “Dim\_tiempo” y “Fact\_tickets” se utilizó la importación de datos desde archivos .CSV. El proceso que se realizó fue el siguiente:

1. Creación de la tabla en SQL
2. En MySQL se ejecuta: SELECT \* FROM tabla\_a\_importar;
3. En el Result Grid se selecciona la opción de importar datos y se elige la ruta donde se encuentra el archivo.



4. Se importan los datos dentro de la tabla ya previamente creada.



5. Se verifica que los tipos de datos de la tabla coincidan con los datos importados.
6. Se realiza la importación.

## 9. Vistas en SQL

Una vista es una tabla virtual que se genera a partir de la ejecución de una o más consultas SQL, aplicada sobre una o más tablas.

En este apartado se realizaron diversas Vistas de consultas a la base de datos del proyecto. Se adjunta en el documento un Script SQL con las siguientes vistas:

- Vista Nro. 1: trae todos los artículos donde la clase sea "Queso". Para obtener el ID de la clase Queso se ejecuta una subconsulta.

```
CREATE VIEW Categoria_Quesos AS
SELECT * FROM avanti_coder.Articulos
WHERE Id_clase = (SELECT ID_Clase FROM avanti_coder.Clases WHERE Nombre_cla = "QUESOS");
```

- Vista Nro. 2: trae todos los artículos donde la marca sea "Lario". Para obtener el ID de la marca Lario se ejecuta una subconsulta.

```
CREATE VIEW Marca_Lario AS
SELECT * FROM avanti_coder.Articulos
WHERE Id_marca = (SELECT ID_Marca FROM avanti_coder.Marcas WHERE Nombre_mar = "LARIO");
```

- Vista Nro. 3: trae todos los clientes donde el nombre empiece con "A" y su código postal sea "3000".

```
CREATE VIEW Cliente_ConA_stafe AS
SELECT * FROM avanti_coder.clientes
WHERE Nombre_cli LIKE "A%" AND CP = 3000;
```

- Vista Nro. 4: trae todos los artículos donde el estado sea SUSPENDIDO.

```
CREATE VIEW Articulos_Suspendidos AS
SELECT * FROM avanti_coder.Articulos
WHERE Estado = "SUSPENDIDO";
```

- Vista Nro. 5: trae todos los clientes con facturas mayores a \$25000.

```
CREATE VIEW Facturacion_Clientes AS
SELECT * FROM avanti_coder.fact_tickets
WHERE TOTAL_Ticket > 25000
GROUP BY ID_Cliente;
```

## 10. Funciones en SQL

Las funciones personalizadas o almacenadas de Mysql permiten procesar y manipular datos de forma procedural y eficiente. Dichos datos son enviados a través de uno o más parámetros, al momento de invocar la función, y devueltos como resultado por esta misma.

Se adjunta en el documento un Script SQL con las siguientes funciones.

- Función Nro. 1: El usuario define un idarticulo y la función devuelve el proveedor que más ha comprado dicho artículo.

```
CREATE DEFINER='root'@'localhost' FUNCTION `maxcliente`(idarticulo INT) RETURNS varchar(255) CHARSET utf8mb4
READS SQL DATA
BEGIN
    DECLARE Max FLOAT(12,2);
    DECLARE Cliente VARCHAR(255);

    SET Max = (SELECT MAX(Cantidad) FROM avanti_coder.fact_tickets WHERE ID_Articulo = idarticulo);
    SET Cliente = (SELECT ID_Cliente FROM avanti_coder.fact_tickets WHERE Cantidad = Max AND ID_Articulo = idarticulo);

    RETURN Cliente;
END
```

- Función Nro. 2: El usuario define un idarticulo y la función devuelve el nombre de ese artículo.

```
CREATE DEFINER='root'@'localhost' FUNCTION `nombreakiculo`(idarticulo INT) RETURNS varchar(255) CHARSET utf8mb4
READS SQL DATA
BEGIN
    DECLARE Articulo VARCHAR(255);
    SET Articulo = (SELECT Descripcion FROM avanti_coder.articulos WHERE ID_Articulo = idarticulo);
    RETURN Articulo;
END
```

## 11. Procedimientos almacenados en SQL

Un Procedimiento Almacenado o Stored Procedure es un programa almacenado físicamente en una base de datos, creado para cumplir tareas específicas. Permite también establecer niveles de seguridad y manipular operaciones complejas o extensas del lado del servidor, evitando un ida y vuelta de datos que termine sobrecargando una red o servidor.

Se adjunta en el documento un Script SQL con los siguientes Procesos almacenados que se ejecutan sobre la base de datos del proyecto.

- Procedimiento almacenado Nro. 1: Cuando se ejecuta el PS se agregan 3 nuevos artículos.

```
CREATE DEFINER='root'@'localhost' PROCEDURE `Agregar articulos`(IN idproducto INT)
BEGIN
    INSERT INTO avanti_coder.articuloS (Descripcion,Unidad,Estado,Cod_barra,Id_proveedor,Id_marca,Id_clase)
    VALUES
    ('QUESO GRUYERE PREMIUM', 'KG', 'ACTIVO', 77123456789, 10, 14, 1),
    ('QUESO PROVOLONE PREMIUM', 'KG', 'ACTIVO', 77123456790, 10, 14, 1),
    ('JAMON IBERICO PREMIUM', 'KG', 'ACTIVO', 77123480755, 20, 32, 2);
END
```

- Procedimiento almacenado Nro. 2: Cuando se ejecuta permite al usuario seleccionar la columna por la que quiere realizar el ordenamiento, el orden (ASCENDENTE O DESCENDENTE) y finalmente la tabla que desea consultar.

```
CREATE DEFINER='root'@'localhost' PROCEDURE `Ordenamiento productos por tabla`(IN campo VARCHAR(255), IN tipo_ordenamiento ENUM('ASC','DESC'), IN tabla VARCHAR(255))
BEGIN
    IF campo <> '' THEN
        SET @ordenar = CONCAT(' ORDER BY ', campo);
    ELSE
        SET @ordenar = '';
    END IF;
    IF tipo_ordenamiento <> '' THEN
        SET @tipo = CONCAT(' ', tipo_ordenamiento);
    ELSE
        SET @tipo = '';
    END IF;
    SET @clausula = CONCAT('SELECT * FROM ',Tabla,@ordenar, @tipo);
    PREPARE ejecutarSQL FROM @clausula;
    EXECUTE ejecutarSQL;
    DEALLOCATE PREPARE ejecutarSQL;
END
```

## 12. Triggers en SQL

Un Trigger puede definirse como una aplicación o programa almacenado en el servidor (de base de datos) creado para ejecutarse de forma automática, cuando uno o más eventos específicos ocurren en la base de datos.

Se adjunta en el documento un Script SQL con los siguientes Triggers.

- Trigger Nro. 1: Este trigger genera un registro luego de que se insertan datos correspondientes a la tabla fact\_tickets.

```
CREATE TABLE IF NOT EXISTS Auditoria_fact(  
    id_auditoria INT AUTO_INCREMENT PRIMARY KEY,  
    fecha DATETIME NOT NULL,  
    usuario VARCHAR(255),  
    Punto_Venta INT NOT NULL,  
    Nro_Comprobante INT NOT NULL,  
    Tipo_operacion VARCHAR(255));  
  
CREATE TRIGGER `Aft_Ins_Fact` AFTER INSERT ON `fact_tickets` FOR EACH ROW  
INSERT INTO `Auditoria_fact` (fecha, usuario, Punto_Venta, Nro_Comprobante, Tipo_operacion)  
VALUES (NOW(), USER(), NEW.Punto_Venta, NEW.Nro_Comprobante, 'INSERT');
```

- Trigger Nro. 2: El siguiente trigger guarda en una tabla "auditoria\_clientes" los nuevos clientes que se ingresen al sistema, guardando también la fecha y usuario encargado de dar de alta al cliente.

```
CREATE TABLE IF NOT EXISTS Auditoria_Clientes(  
    ID_Cliente INT NOT NULL,  
    Nombre VARCHAR(255) NOT NULL,  
    Fecha DATETIME NOT NULL,  
    Usuario VARCHAR(255));  
  
CREATE TRIGGER `Bef_Ins_Cliente`  
BEFORE INSERT ON `Clientes` FOR EACH ROW  
INSERT INTO `Auditoria_Clientes` (ID_Cliente, Nombre, Fecha, Usuario)  
VALUES (NEW.ID_Cliente, NEW.Nombre_cli, NOW(), USER());
```

- Trigger Nro. 3: Los siguientes dos triggers registran en una tabla LOG, la fecha, hora y usuario. Antes de que se realice un UPDATE de un artículo y después de que se ejecute un DELETE de un artículo.

```
CREATE TABLE IF NOT EXISTS avanti_coder.LOG(  
    fecha DATE,  
    hora VARCHAR(255),  
    usuario VARCHAR(255),  
    id_articulo INT,  
    tipo_movimiento VARCHAR(255));
```

```
CREATE TRIGGER `Bef_Upd_articulos1`  
BEFORE UPDATE ON `articulos` FOR EACH ROW  
INSERT INTO `LOG` (fecha, hora, usuario, id_articulo, tipo_movimiento)  
VALUES (CURDATE(), CURTIME(), USER(), NEW.id_articulo, "UPDATE");
```

```
CREATE TRIGGER `Aft_Del_articulos`  
AFTER DELETE ON `articulos` FOR EACH ROW  
INSERT INTO `LOG` (fecha, hora, usuario, id_articulo, tipo_movimiento)  
VALUES (CURDATE(), CURTIME(), USER(), articulos.ID_Articulo, "DELETE");
```

## 13. Data Control Language

El Lenguaje de Control de Datos (DCL) permite definir diferentes usuarios dentro del motor de base de datos Mysql, y establecer para cada uno de ellos, permisos totales, parciales, o negar el acceso sobre los diferentes Objetos que conforman la Base de Datos.

Para el proyecto se utilizó este lenguaje para la creación de dos usuarios, con diferentes permisos:

- Otorgar permisos de solo lectura a todas las tablas al Usuario1.
- Otorgar permisos de lectura, inserción y modificación a todas las tablas al Usuario2.
- Ambos usuarios no pueden eliminar registros.

Se adjunta un Script SQL con la creación de los dos usuarios y los permisos correspondientes a cada uno:

```
USE mysql;
SELECT * FROM USER;

#Creación de los usuarios

CREATE USER Usuario1@localhost IDENTIFIED BY 'Test2021';
CREATE USER Usuario2@localhost IDENTIFIED BY 'Test2022';

#Otorgar permisos de solo lectura a todas las tablas al Usuario1:

GRANT SELECT ON *.* TO Usuario1@localhost;

#Otorgar permisos de lectura, inserción y modificación a todas las tablas al Usuario2:

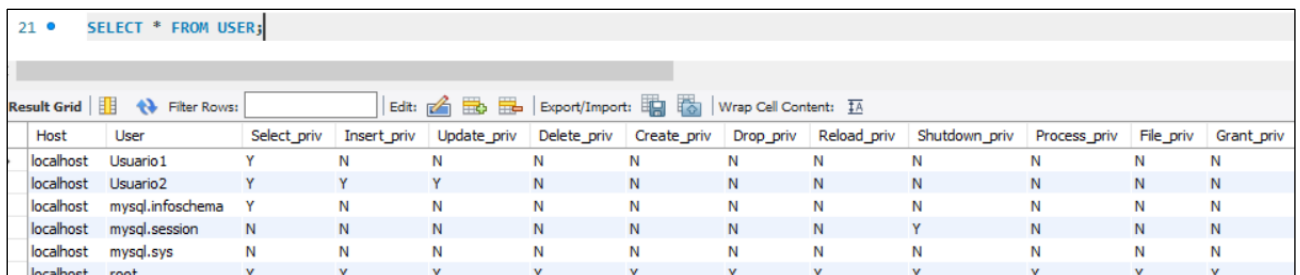
GRANT SELECT, INSERT, UPDATE ON *.* TO Usuario2@localhost;

#Aseguramos que ambos usuarios no pueden eliminar registros:

REVOKE DELETE ON *.* FROM Usuario1@localhost, Usuario2@localhost;

SELECT * FROM USER;
```

La sentencia `SELECT * FROM USER;` permite ver los distintos usuarios que tienen acceso a la base de datos y los permisos correspondientes.



Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Reload_priv	Shutdown_priv	Process_priv	File_priv	Grant_priv
localhost	Usuario1	Y	N	N	N	N	N	N	N	N	N	N
localhost	Usuario2	Y	Y	Y	N	N	N	N	N	N	N	N
localhost	mysql.infoschema	Y	N	N	N	N	N	N	N	N	N	N
localhost	mysql.session	N	N	N	N	N	N	N	Y	N	N	N
localhost	mysql.sys	N	N	N	N	N	N	N	N	N	N	N
localhost	root	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

## 14. Transaction Control Language

Se conoce como Transaction Control Language (o TCL) a una parte del lenguaje Transact-SQL que administra las transacciones en una base de datos.

TCL es utilizado para administrar cada uno de los cambios que se generan en una o más tablas mediante las cláusulas DML.

Para el proyecto se utilizó con los siguientes objetivos:

- Eliminar 5 proveedores, pero luego poder recuperarlos.
- Eliminar 5 proveedores, de forma definitiva.
- Insertar 8 nuevos registros y realizar 2 savepoints (luego realizar la eliminación de los registros del primer savepoint).



Se adjunta un archivo SQL que contiene las siguientes sentencias, correspondientes al lenguaje TCL:

```
START TRANSACTION;

DELETE FROM avanti_coder.proveedores WHERE ID_proveedor = 1;
DELETE FROM avanti_coder.proveedores WHERE ID_proveedor = 2;
DELETE FROM avanti_coder.proveedores WHERE ID_proveedor = 3;
DELETE FROM avanti_coder.proveedores WHERE ID_proveedor = 4;
DELETE FROM avanti_coder.proveedores WHERE ID_proveedor = 5;

#Si yo cierro MySQL Workbench las eliminaciones no se guardan
#Para deshacer las eliminaciones ejecuto:
ROLLBACK;

#Eliminar registros definitivamente

START TRANSACTION;

DELETE FROM avanti_coder.proveedores WHERE ID_proveedor = 1;
DELETE FROM avanti_coder.proveedores WHERE ID_proveedor = 2;
DELETE FROM avanti_coder.proveedores WHERE ID_proveedor = 3;
DELETE FROM avanti_coder.proveedores WHERE ID_proveedor = 4;
DELETE FROM avanti_coder.proveedores WHERE ID_proveedor = 5;

#Para confirmar definitivamente las eliminaciones ejecuto:
COMMIT;
```

```
#Inserción de 8 nuevos registros con savepoint
START TRANSACTION;

INSERT INTO avanti_coder.Articulos ('ID_Articulo','Descripcion','Unidad','Estado','Cod_barra','Id_proveedor','Id_marca','Id_clase')
VALUES (180,'MORTADELA PREMIUM','kg','ACTIVO',780000000001,17,20,2);
INSERT INTO avanti_coder.Articulos ('ID_Articulo','Descripcion','Unidad','Estado','Cod_barra','Id_proveedor','Id_marca','Id_clase')
VALUES (181,'JAMON PREMIUM','kg','ACTIVO',780000000002,17,20,2);
INSERT INTO avanti_coder.Articulos ('ID_Articulo','Descripcion','Unidad','Estado','Cod_barra','Id_proveedor','Id_marca','Id_clase')
VALUES (182,'SALAME PREMIUM','kg','ACTIVO',780000000003,17,20,2);
INSERT INTO avanti_coder.Articulos ('ID_Articulo','Descripcion','Unidad','Estado','Cod_barra','Id_proveedor','Id_marca','Id_clase')
VALUES (183,'JAMON CRUDO PREMIUM','kg','ACTIVO',780000000004,17,20,2);
SAVEPOINT CuatroRegistros;

INSERT INTO avanti_coder.Articulos ('ID_Articulo','Descripcion','Unidad','Estado','Cod_barra','Id_proveedor','Id_marca','Id_clase')
VALUES (184,'MATAMBRE PREMIUM','kg','ACTIVO',780000000005,17,20,2);
INSERT INTO avanti_coder.Articulos ('ID_Articulo','Descripcion','Unidad','Estado','Cod_barra','Id_proveedor','Id_marca','Id_clase')
VALUES (185,'FIAMBRE PREMIUM','kg','ACTIVO',780000000006,17,20,2);
INSERT INTO avanti_coder.Articulos ('ID_Articulo','Descripcion','Unidad','Estado','Cod_barra','Id_proveedor','Id_marca','Id_clase')
VALUES (186,'PALETA COCIDA PREMIUM','kg','ACTIVO',780000000007,17,20,2);
INSERT INTO avanti_coder.Articulos ('ID_Articulo','Descripcion','Unidad','Estado','Cod_barra','Id_proveedor','Id_marca','Id_clase')
VALUES (187,'SALAMIN PREMIUM','kg','ACTIVO',780000000008,17,20,2);
INSERT INTO avanti_coder.Articulos ('ID_Articulo','Descripcion','Unidad','Estado','Cod_barra','Id_proveedor','Id_marca','Id_clase')
VALUES (188,'SALAME MILAN PREMIUM','kg','ACTIVO',780000000009,17,20,2);
SAVEPOINT OchoRegistros;

#Eliminar savepoint:

RELEASE SAVEPOINT CuatroRegistros;
```



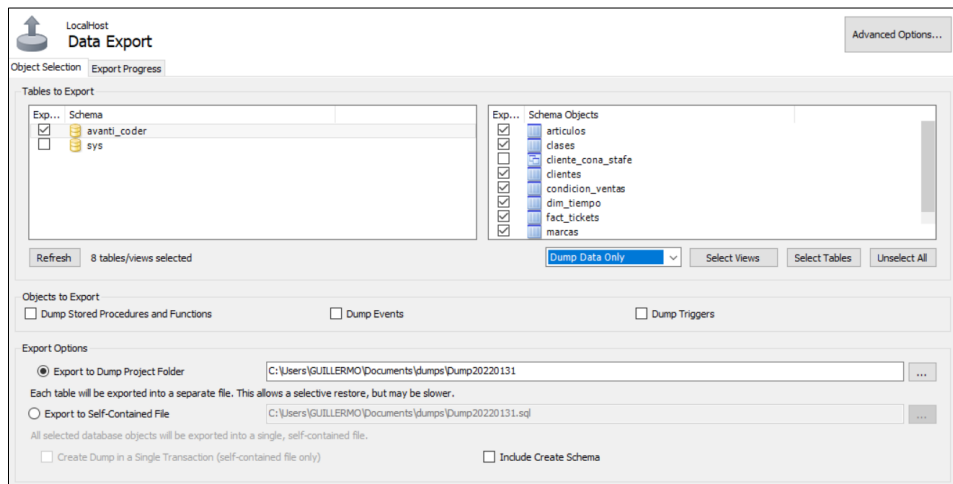
## 15. Backup y restauración

En este apartado lo que se realizó fue una copia de los datos originales de la base de datos que se realiza con el fin de disponer de un medio para recuperarlos en caso de una falla o pérdida.

El gestor de bases de datos MySQL permite dos métodos para realizar backups de la información:

- Mysql incluye la herramienta *mysqldump* dentro del motor de base de datos para gestionar las copias de seguridad. Se utiliza a través de la Línea de Comandos o Terminal, de nuestro S.O.
- Mysql Workbench cuenta con un apartado denominado **Administration.**, dentro de este panel encontramos las opciones Data Export y Data Import/Restore, que nos permiten realizar las tareas de crear los backups y realizar luego su recuperación.





Se adjunta un Script SQL con la creación de los backups correspondientes a las tablas "Articulos", "clases", "clientes", "condicion\_ventas", "fact\_tickets", "marcas", "proveedores" que se realizaron con el apartado Administration en MySQL Workbench. Se realizó una copia de seguridad únicamente de los registros (sin la estructura de las tablas).

## 16. Herramientas y tecnologías utilizadas

A continuación, se realiza un listado de las principales tecnologías empleadas para la realización de este proyecto:

- Software contable ALQUIMIA: este software de gestión de empresas es utilizado en la Organización para registrar artículos, proveedores y clientes. Como así también generar los tickets y facturas de ventas. Mediante este software se obtuvo el dataset del proyecto.
- Microsoft Excel – Power Query Editor: se empleó para realizar transformaciones a los datos, depurar y prepararlos para luego poder ser importados a la base de datos en MySQL.
- MySQL Workbench: se utilizó este gestor de bases de datos para la realización del Schema del proyecto, sus tablas y registros. Como así también las vistas, procesos almacenados y triggers entre otras aplicaciones que nos permitió desarrollar el programa a lo largo del proyecto.