PROYECTO FINAL CURSO SQL

Pasajes y usuarios en el sistema de transporte de Argentina adheridos a la Red SUBE

Profesor: Rodas Miguel
Tutor: Bevilacqua Flavio
Comisión: 31270

Lorenzo Alliot
lorenzo-alliot

Contenido

1. INTRODUCCIÓN

En este proyecto se abarcará la temática de los usos de tarjetas SUBE. Se desarrollará una base de datos SQL desde el SGBD MySQL, se tomarán como base los archivos de datos abiertos proporcionados por el Ministerio de Transporte de la Nación.

2. OBJETIVOS

El objetivo de este proyecto será construir una base de datos normalizada y estructurada, que sea totalmente robusta en cuanto a la integridad de sus datos, pero así también sea lo suficientemente flexible en su uso y performance.

3. SITUACIÓN PROBLEMÁTICA

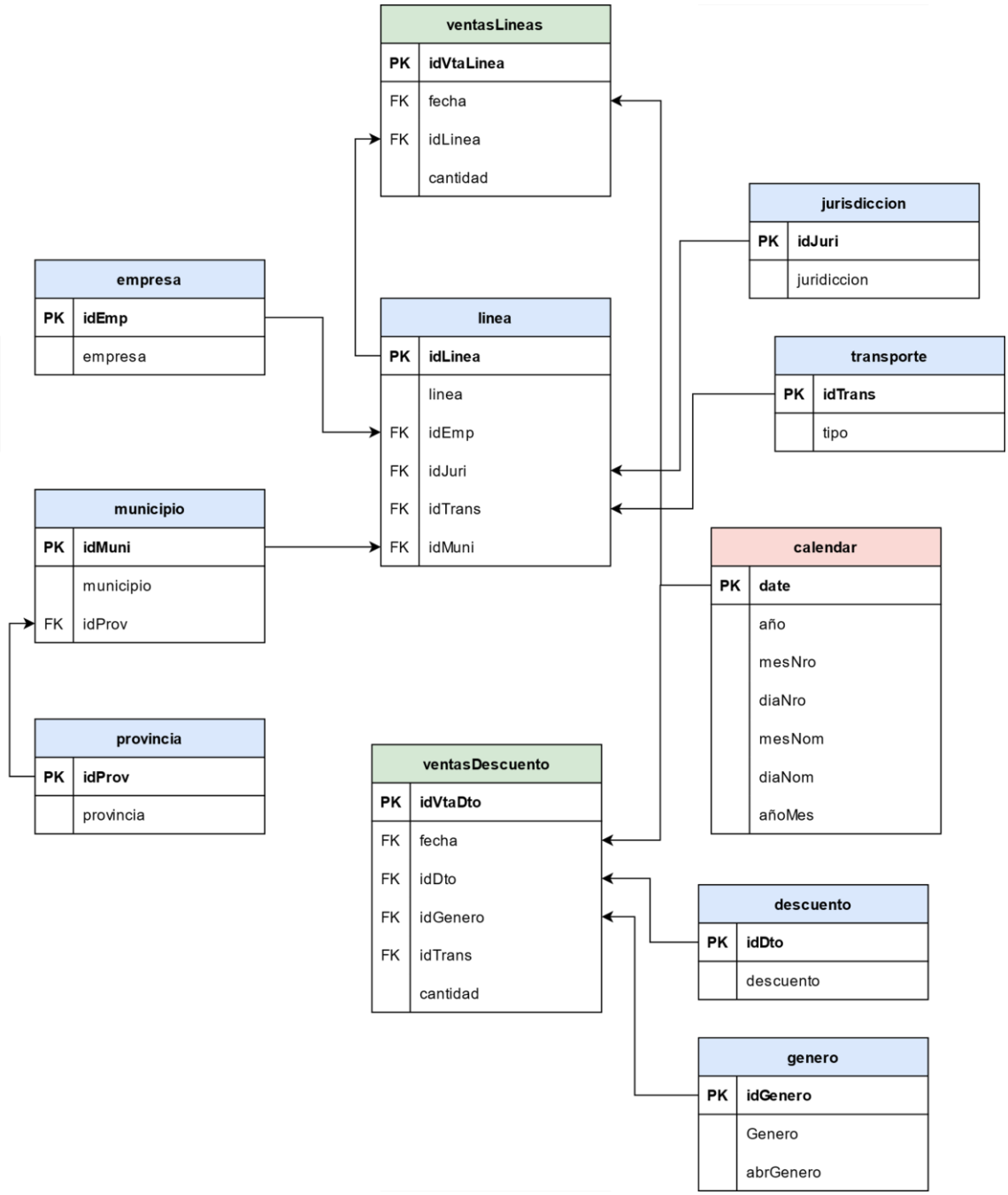
Desde los datos abiertos del Gobierno de la Nación se presentan varios dataset con gran cantidad de información que si bien están estructurados no están normalizados, lo cual complica sus análisis, además de la baja performance consecuente por su gran volumen. Con este proyecto lo que se busca que organizar y optimizar la información para maximizar su rendimiento.

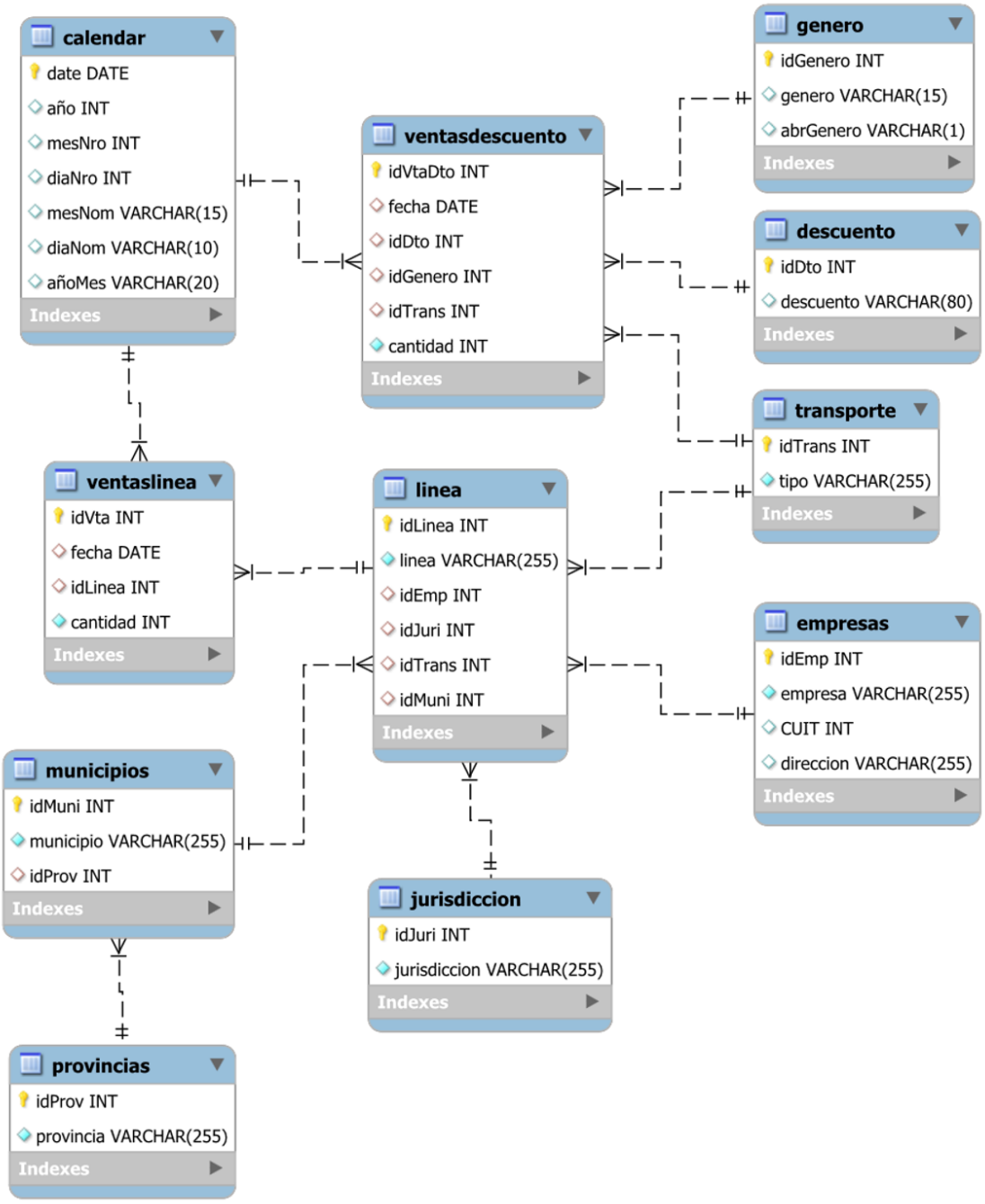
4. MODELO DE NEGOCIO

El Sistema Único de Boleto Electrónico, también conocido como SUBE, es una metodología de pago implementada en Argentina a partir del año 2011 que permite a cada usuario con su respectiva tarjeta inteligente, abonar los viajes en colectivos, subtes, trenes y lanchas (este último desde 2019), adheridas a la Red SUBE.

Es un servicio simple y práctico, similar al utilizado en varios partes del mundo para unificar los medios de pagos de varios tipos de transportes dentro de un mismo territorio nacional. En Argentina este sistema es administrado por el Gobierno Nacional, lo que simplifica también la facilidad para el otorgamiento de descuentos o beneficios en el sistema de transporte.

5. DIAGRAMA E-R




calendar
date DATE
año INT
mesNro INT
diaNro INT
mesNom VARCHAR(15)
diaNom VARCHAR(10)
añoMes VARCHAR(20)
Indexes
ventasdescuento
idVtaDto INT
fecha DATE
idDto INT
idGenero INT
idTrans INT
cantidad INT
Indexes
genero
idGenero INT
genero VARCHAR(15)
abrGenero VARCHAR(1)
Indexes
descuento
idDto INT
descuento VARCHAR(80)
Indexes
transporte
idTrans INT
tipo VARCHAR(255)
Indexes
ventaslinea
idVta INT
fecha DATE
idLinea INT
cantidad INT
Indexes
linea
idLinea INT
linea VARCHAR(255)
idEmp INT
idJuri INT
idTrans INT
idMuni INT
Indexes
empresas
idEmp INT
empresa VARCHAR(255)
CUIT INT
direccion VARCHAR(255)
Indexes
municipios
idMuni INT
municipio VARCHAR(255)
idProv INT
Indexes
jurisdiccion
idJuri INT
jurisdiccion VARCHAR(255)
Indexes
provincias
idProv INT
provincia VARCHAR(255)
Indexes

6. DESCRIPCIÓN DE LA BASE DE DATOS

Temática de la base de datos

Esta base almacena la cantidad de usos de tarjetas SUBE (cantidad de pasajes) agrupados por día en las distintas líneas pertenecientes a sus empresas, con los datos complementarios de a qué provincia y municipio pertenecen, a que jurisdicción y tipo de transporte corresponden. Por otra parte, también se presentan los usuarios de las tarjetas SUBE registradas, por género y tipo de descuento otorgado.

Contexto de tablas

- **Empresa**: se listarán todos los nombres y contendrá los datos de las distintas empresas que realicen ventas con tarjeta SUBE
- **Provincia**: se listarán todas las provincias de Argentina donde haya habido al menos una venta con tarjeta SUBE
  Las provincias JN corresponden a las empresas que tienen Jurisdicción Nacional
  Las provincias en blanco corresponden a los subtes
- **Municipio**: se listarán todos los municipios de las provincias de Argentina donde haya habido al menos una venta con tarjeta SUBE.
  Tendrá relación con la/s tabla/s:
  - provincia
- **Transporte**: se listarán los distintos tipos de transporte que hayan realizado al menos una venta con tarjeta SUBE
- **Juridiccion**: se listarán los distintos tipos de jurisdicción a las que puedan pertenecer las distintas líneas de transporte.
- **Linea**: se listarán todas las líneas de transporte pertenecientes a las empresas donde al menos se haya realizado una venta con tarjeta SUBE.
  Tendrá relación con la/s tabla/s:
  - empresa
  - municipio
  - transporte
  - jurisdicción
  - calendar
- **VentasLinea**: se listarán la cantidad de ventas agrupadas por día de cada línea de transporte que haya realizado al menos una venta con tarjeta SUBE
- **Descuento:** se listarán los distintos tipos de descuentos otorgados para las tarjetas SUBE.
- **Genero**: se listarán los géneros de los titulares de las tarjetas SUBE
- **Calendar**: tabla calendario donde se listarán todas las fechas, desde la más antigua a la más reciente contenida en la tabla de VentasLinea o VentasDescuento
- **VentasDescuento**: se listarán la cantidad de ventas agrupadas por día y segmentada por el tipo de descuento, el género y el transporte utilizado.
  Tendrá relación con la/s tabla/s:
  - descuento
  - genero
  - calendar

7. LISTADOS DE OBJETOS DE LA BASE DE DATOS

7.1. Tablas

Las tablas almacenan la información en forma de registros o tuplas. Para ello, respetan la estructura de cada dato de un registro, el cual condice con la definición del campo que lo almacena.

Este proyecto cuenta con 13 tablas en total, de ellas, 8 son tablas de dimensiones (encabezado celeste), 2 son tablas de hechos (encabezados verdes), 1 calendar (encabezado rojo) y 2 tablas de logs que no se listan porque el tema se abarcará más adelante.

provincia		
CAMPO	TIPO DATO	PROPIEDAD
idProv	INT	PK
provincia	varchar(25)	-

municipio		
CAMPO	TIPO DATO	PROPIEDAD
idMuni	INT	PK
municipio	varchar(80)	-
idProv	INT	FK

empresa		
CAMPO	TIPO DATO	PROPIEDAD
idEmp	INT	PK
empresa	varchar(100)	-

descuento		
CAMPO	TIPO DATO	PROPIEDAD
idDto	INT	PK
descuento	varchar(80)	-

jurisdiccion		
CAMPO	TIPO DATO	PROPIEDAD
idJuri	INT	PK
jurisdiccion	varchar(85)	-

transporte		
CAMPO	TIPO DATO	PROPIEDAD
idTrans	INT	PK
tipo	varchar(15)	-

linea		
CAMPO	TIPO DATO	PROPIEDAD
idLinea	INT	PK
linea	varchar(65)	-
idEmp	INT	FK
idJuri	INT	FK
idTrans	INT	FK
transporte	INT	FK

ventasLinea		
CAMPO	TIPO DATO	PROPIEDAD
idVta	INT	PK
fecha	date	FK
idLinea	INT	FK
cantidad	INT	-

genero		
CAMPO	TIPO DATO	PROPIEDAD
idGenero	INT	PK
genero	varchar(15)	-
abrGenero	varchar(1)	-

calendar		
CAMPO	TIPO DATO	PROPIEDAD
date	DATE	PK
año	INT	-
mesNro	INT	-
diaNro	INT	-
mesNom	varchar(20)	-
diaNom	varchar(15)	-
añoMes	varchar(30)	-

ventasDescuento		
CAMPO	TIPO DATO	PROPIEDAD
idVtaDto	INT	PK
Fecha	date	FK
idDto	INT	FK
idGenero	INT	FK
idTrans	INT	FK
cantidad	INT	-

7.2. Vistas

Vista SQL es básicamente una tabla virtual que se genera a partir de la ejecución de una o más consultas SQL, aplicada sobre una o más tablas. Su estructura corresponde a una serie de filas y columnas tal como encontramos en las tablas SQL, que almacenan la vista de la información tal como la definimos al crearla.

En el proyecto se crearán 5 vistas distintas:

7.2.1. **PaxPorDia**:

a. Descripción: VISTA de pasajeros total y promedio AGRUPADOS por día ORDENADO de mayor a menor.
b. Objetivo: identificar cuáles son los días de la semana con más pasajeros.
c. Tablas que la componen: VentasLinea, Calendar.

```sql
CREATE OR REPLACE VIEW PasajerosPorDia AS
    (SELECT
        C.diaNom,
        SUM(vl.cantidad) AS PasajerosTotales,
        ROUND((SUM(vl.cantidad) / COUNT(DISTINCT (c.date))),0) AS PasajerosProm
    FROM
        calendar C
            JOIN
        ventaslinea VL ON (C.date = vl.fecha)
    GROUP BY diaNom
    ORDER BY PasajerosProm desc);
```

7.2.2. **PasajerosPorEmpresa**:

a. Descripción: VISTA de cantidad de pasajeros en tabla ventaslinea AGRUPADOS POR empresa DONDE tipo de transporte es colectivo y provincia es Santa Fe ORDENADOS de mayor a menor.
b. Objetivo: Saber cuáles son las empresas de colectivo en la provincia de Santa Fe son las de mayor cantidad de pasajeros.
c. Tablas que la componen: Empresa, Linea, VentasLinea, Transporte, Municipios, Provincias, Tipo.

```sql
CREATE OR REPLACE VIEW PasajerosPorEmpresa AS
    (SELECT
        E.Empresa, T.tipo AS Transporte, SUM(cantidad) AS Pasajeros
    FROM
        empresas E
            JOIN
        linea L ON (e.idEmp = l.idEmp)
            JOIN
        ventaslinea VL ON (l.idLinea = vl.idLinea)
            JOIN
        transporte t ON (l.idTrans = t.idTrans)
            JOIN
        municipios m ON (l.idMuni = m.idMuni)
            JOIN
        provincias p ON (p.idprov = m.idprov)
    WHERE
        t.tipo = 'Colectivo'
            AND p.provincia = 'Santa Fe'
    GROUP BY e.empresa
    ORDER BY Pasajeros DESC);
```

7.2.3. PaxPorGenero:

a. Descripción: VISTA de cantidad y porcentaje de pasajeros de la tabla VentasDescuento AGRUPADOS por el género.
b. Objetivo: saber la cantidad de pasajeros por cada género y que porcentaje representan al total de pasajeros.
c. Tablas que la componen: VentasDescuento, Genero.

```
/*
Creo la funcion TotalPaxVD para obtener la cantidad e pasajeros totales de la tabla VentasDescuento,
a este numero lo puedo usar de divisor para obtener el porcentaje de pasajeros en cada genero.
*/

CREATE DEFINER=`root`@`localhost` FUNCTION `TotalPaxVD`() RETURNS int
    DETERMINISTIC
RETURN (select sum(cantidad)as pasajeros from ventasDescuento);

-- Creo la vista PaxPorGenero

CREATE OR REPLACE VIEW PaxPorGenero AS
    (SELECT
        g.genero,
        SUM(vd.cantidad) AS Pasajeros,
        (SUM(vd.cantidad) / TOTALPAXVD()) * 100 AS "%Pasajeros"
    FROM
        genero g
            JOIN
        ventasdescuento vd ON (g.idGenero = vd.idGenero)
    GROUP BY g.genero);
```

7.2.4. PaxPorProvincias:

a. Descripción: VISTA de cantidad de pasajeros AGRUPADOS por provincias DONDE el día de la semana sea "lunes".
b. Objetivo: Conocer cuál es la cantidad de pasajeros los días lunes en cada provincia.
c. Tablas que la componen: VentasLinea, Linea, Municipios, Provincias, Calendar.

```
CREATE OR REPLACE VIEW pasajerosporprovincia AS
    (SELECT
        SUM(vl.cantidad) AS pasajeros, p.provincia
    FROM
        ventaslinea vl
            JOIN
        linea l ON (l.idLinea = vl.idLinea)
            JOIN
        municipios m ON (l.idMuni = m.idMuni)
            JOIN
        provincias p ON (p.idprov = m.idprov)
    WHERE
        vl.fecha IN (SELECT
                c.date
            FROM
                calendar c
            WHERE
                c.diaNom = 'lunes')
    GROUP BY p.provincia);
```

7.2.5. CantidadesPorProvincias:

a. Descripción: VISTA de suma de pasajeros, cantidades de líneas, municipio y empresas AGRUPADOS por provincias.

b. Objetivo: Visualizar la composición de las provincias en el sistema SUBE.

c. Tablas que la componen: VentasLinea, Linea, Municipios, Provincias, Empresas.

```
CREATE OR REPLACE VIEW CantidadesPorProvincias AS
    (SELECT
        p.Provincia,
        sum(vl.cantidad) as Cantidad_Pasajeros,
        COUNT(DISTINCT (l.idlinea)) AS Cantidad_Lineas,
        COUNT(DISTINCT (m.idmuni)) AS Cantidad_Municipios,
        COUNT(DISTINCT (e.idEmp)) AS Cantidad_Empresas
    FROM
        ventaslinea vl
            JOIN
        linea l ON (vl.idlinea=l.idlinea)
            JOIN
        empresas e ON (l.idemp = e.idemp)
            JOIN
        municipios m ON (l.idmuni = m.idmuni)
            JOIN
        provincias p ON (m.idprov = p.idprov)
    GROUP BY p.provincia);
```

7.3. Funciones

Las funciones customizadas 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 retornando un único resultado.

Se crearán 4 funciones, la primera, TotalPaxVD fue creada en la creación de la vista PaxPorGenero para hacer uso de ella.

7.3.1. **TotalPaxVD**:

a) Descripción: función para obtener la cantidad de pasajeros totales de la tabla VentasDescuento.

b) Objetivo: usar este número como divisor para obtener el porcentaje de pasajeros en distintas querys.

c) Datos y/o tablas manipuladas: VentasDescuento[cantidad]

```
CREATE DEFINER=`root`@`localhost` FUNCTION `TotalPaxVD`() RETURNS int
    DETERMINISTIC
RETURN (select sum(cantidad)as pasajeros from ventasDescuento);
```

7.3.2. **pax_fecha_transporte_empresa:**

a) Descripción: Función que devuelve la cantidad de pasajeros en una fecha, para un tipo de transporte y una empresa determinada de la tabla ventasLinea según los parámetros establecidos.
En caso de error por valores incorrectos en los parámetros la función devuelve una sentencia indicando en donde está la falla.

b) Objetivo: Saber que cantidad de pasajeros transportó una empresa en una fecha determinada.

c) Datos y/o tablas manipuladas: VentasLinea[cantidad], Empresas, Linea, Transporte, Calendar, Transporte

d) Parámetros:

* Fecha
* Tipo de transporte [idTrans]
* Empresa [idEmp]

```
DROP function IF EXISTS `pax_fecha_transporte_empresa`;

DELIMITER $$
$$
CREATE DEFINER=`root`@`localhost` FUNCTION `pax_fecha_transporte_empresa`(p_fecha date, p_idTrans int, p_idEmp int) RETURNS char(255)
    READS SQL DATA
BEGIN
declare v_pasajeros int;
    set v_pasajeros =
    (SELECT
       SUM(vl.cantidad) AS Pasajeros
    FROM
        empresas E
            JOIN
        linea L ON (e.idEmp = l.idEmp)
            JOIN
        ventaslinea VL ON (l.idLinea = vl.idLinea)
            JOIN
        transporte t ON (l.idTrans = t.idTrans)
    WHERE
        t.idTrans = p_idTrans
        and e.idEmp=p_idEmp
        and vl.fecha=p_fecha);
        IF (p_fecha) NOT between '2022-01-01' and '2022-01-10' then
            RETURN "Seleccione una fecha entre '2022-01-01' y '2022-01-10'";
            ELSE
                IF (ISNULL(v_pasajeros)) then RETURN "La empresa seleccionada no tiene relación con el tipo de transporte";
                ELSE RETURN v_pasajeros;
            END IF;
        END IF;
END$$
```

7.3.3. PorcentajePorGenero:

a) Descripción: Función que devuelve el porcentaje de pasajeros para un género especificado en los parámetros, en la tabla VentasDescuento.
En caso de error por valores incorrectos en los parámetros la función devuelve una sentencia indicando en donde está la falla.

b) Objetivo: ver qué número de pasajeros y que porcentaje del total representa un género especifico en el transporte de pasajeros.

c) Datos y/o tablas manipuladas: VentasDescuento [Cantidad], Genero[idGenero]

d) Parámetros:

* Genero [idGenero]

```
DROP function IF EXISTS `PorcentajePorGenero`;

DELIMITER $$
$$
CREATE DEFINER=`root`@`localhost` FUNCTION `PorcentajePorGenero`(p_idGenero int) RETURNS CHAR(255)
    DETERMINISTIC
BEGIN
    declare v_pasajerosVD int;
    DECLARE v_paxgen int;
    set v_pasajerosVD =(select sum(cantidad)from ventasDescuento vd);
    set v_paxgen = (select sum(cantidad) from ventasDescuento vd where p_idGenero=vd.idGenero);
    if (p_idGenero) > 3 then RETURN "Parámetro incorrecto, seleccione un número del 1 al 3";
    ELSE
RETURN v_paxgen/v_pasajerosVD*100;
END IF;
END$$
```

7.3.4. **PaxPromedioPorDescuento**:

a) Descripción: Función que devuelve el promedio de pasajeros para un tipo de descuento establecido en los parámetros, en la tabla VentasDescuento.
En caso de error por valores incorrectos en los parámetros la función devuelve una sentencia indicando en donde está la falla.

b) Objetivo: obtener el valor promedio de pasajeros en un tipo de descuento para analizar su impacto en la totalidad de los pasajes.

c) Datos y/o tablas manipuladas: VentasDescuento [Cantidad], Descuentos [idDescuento]

d) Parámetros:

* Descuento [idDescuento]

```
DROP function IF EXISTS `PaxPromedioPorDescuento`;

DELIMITER $$
$$
CREATE DEFINER=`root`@`localhost` FUNCTION `PaxPromedioPorDescuento`(p_idDto int) RETURNS char(255)
    DETERMINISTIC
BEGIN
    DECLARE v_paxdto int;
        set v_paxdto = (select avg(vd.cantidad) from ventasDescuento vd where p_idDto=vd.idDto);
        if (p_idDto) > 8 then
        RETURN "Parámetro incorrecto, seleccione un número del 1 al 8"; else
RETURN v_paxdto;
end if;
END$$
```

7.4. Stores Procedures

Un Stored Procedure o Procedimiento Almacenado representa un conjunto de sentencias almacenado físicamente en una DB, creado para cumplir tareas específicas. Su objetivo es resolver desde una operación simple hasta operaciones complejas que requieran modificar varias tablas y/o datos almacenados en una DB

Crearemos 2 Stores Procedures, uno para inserción en una tabla y otra para ordenar la tabla

7.4.1. **insertProvincia**:

a) Descripción: SP que inserta una provincia en la tabla provincias, haciendo que el id incremente automáticamente +1 y con un upper busque si ya existe esa provincia. Si ya existe no la debe dejar crear enviando una sentencia con error, si no existe se ejecuta el SP insertando la nueva provincia.

b) Objetivo o beneficio: El beneficio que otorga al proyecto es que se puede agregar una provincia y que el SP automáticamente le otorga el id consecutivo y evita el error de duplicar un registro por escribir con mayúsculas o minúsculas ya que reporta error

c) Datos y/o tablas manipuladas: Provincias[idProv, Provincia]

d) Parámetros: IN p_provincia

```
DROP procedure IF EXISTS `insertProvincia`;

DELIMITER $$
$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `insertProvincia`( in p_provincia char(255))
BEGIN
    SET @idProv = (SELECT max(idProv) FROM provincias)+1; -- Buscamos el último id y le agregamos uno
    SET @Provincia= upper(p_provincia); -- ponemos todo el texto en mayúsculas
    SET @cantidad = (SELECT count(*) FROM provincias p WHERE p.provincia=@provincia); -- Buscamos el texto ingresado, si es > 1 es porque ya existe

    IF @cantidad > 0 THEN
    SELECT '"La provincia ya existe"';
    ELSEIF @cantidad = 0 THEN
    INSERT INTO provincias(idProv,provincia) VALUES(@idProv,@provincia);
    END IF;
END$$
```

7.4.2. **order_tables**:

a) Descripción: SP donde se seleccione una tabla de la bd, se elija una columna de la tabla y se ordene asc o desc según se determine. En caso de que la tabla o la columna no exista devolverá error.

b) Objetivo o beneficio: El beneficio que tiene este SP es la flexibilidad que presenta al poder seleccionar cualquier tabla de la bd y ordenarla por el campo que se desee.

c) Datos y/o tablas manipuladas: La tabla y el campo que se elija en los parámetros.

d) Parámetros:

* IN p_table = como la tabla a utilizar
* IN p_field= como el campo por el cual se ordenará
* IN p_order= como el orden que se otorgará

```sql
DROP procedure IF EXISTS `order_tables`;
DELIMITER $$
$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `order_tables`(in p_table char(50), in p_field char(50), in p_order char(5))
BEGIN
    /*LOS PARAMETROS A DEFINIR SON:
    p_table = COMO LA TABLA A UTILIZAR
    p_field= COMO EL CAMPO POR EL CUAL SE ORDENARÁ
    p_order= COMO EL ORDEN QUE SE OTORGARÁ
    */

    set @tabla = lower(p_table); -- ponemos en minuscula el nombre de la tabla a ordenar
    set @campo = lower(p_field); -- ponemos en minuscula el nombre del campo por el cual se ordenará
    set @v_order= (p_order);
    set @Clausulaok = concat("Select * from ",@tabla," order by ",@campo," ",@v_order); -- generamos la consulta a ejecutar si no hay errores

 -- hacemos un if donde se ejecuta una consulta donde si coincide el @campo con alguna columna de la tabla se continua con el procedimiento
 -- ejecutamos una subconsulta para traer los nomres de los campos de la @tabla
 -- si hay alguna coincidencia el count(*) será mayor a 0 y se ejecutara @Clausulaok
 -- si no hay coincidencia el conteo será 0 y arrojará el error establecido

     if (select count(*) as coincidencia from (SELECT
            COLUMN_NAME
        FROM
            INFORMATION_SCHEMA.COLUMNS
        WHERE
            TABLE_SCHEMA = DATABASE()
                AND TABLE_NAME = @tabla) as c where c.COLUMN_NAME =  @campo) > 0
        then
        prepare runSQLok from @clausulaok;
        execute runSQLok;
        else select "Tabla o columna inexistente";
    end if ;
END$$
```

7.5. Triggers

Los triggers son un conjunto de sentencias o programa almacenado en el servidor (de DB) creado para ejecutarse (dispararse) de forma automática, cuando uno o más eventos de DML específicos ocurren en la DB.

Crearemos triggers para las dos tablas más usadas del proyecto, cada una tendrá un trigger de inserción y otro de eliminación.

7.5.1. TRIGGERS TABLA VENTASLINEA

- Se crea una tabla log para los delete y update de la tabla ventaslinea.
- La tabla contendrá los archivos antes de eliminarse, antes de actualizarse y luego de actualizarse.
- Se diferenciará la acción realizada por la columna acción.

```
CREATE TABLE IF NOT EXISTS `logVentasLinea` (`idVta` INT NOT NULL,
                                             `fecha` DATE NULL,
                                             `idLinea` INT NULL,
                                             `cantidad` INT NULL,
                                             `user` VARCHAR(255) NOT NULL,
                                             `FechaAccion` DATE NOT NULL,
                                             `HoraAccion` TIME NOT NULL,
                                             `motivo` VARCHAR(255) DEFAULT NULL,
                                             `accion` VARCHAR(15)
                                             );
```

- Se crea un SP para colocar el motivo por el cual se elimina algún registro.

```
DROP PROCEDURE IF EXISTS `motivo_delete_VL`;

DELIMITER $$
 $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `motivo_delete_VL`(in p_motivo varchar (255))
BEGIN
UPDATE `logVentasLinea`
SET `motivo` = p_motivo WHERE motivo IS NULL ;
END$$
```

7.5.1.1. TRIGGER bef_del_ventasLineas_logVentasLinea

En la tabla logVentasLinea se registrará con un trigger before delete: todos los campos de la tabla ventaslinea que se eliminan, el usuario, la fecha, la hora, el motivo por el que lo elimina (llamando al SP "motivo_delete") y la acción realizada.

```
DROP TRIGGER IF EXISTS `bef_del_ventasLineas_logVentasLinea`;

DELIMITER $$
$$
CREATE TRIGGER `bef_del_ventasLineas_logVentasLinea`
BEFORE
DELETE ON ventaslinea
FOR EACH ROW BEGIN
INSERT INTO logVentasLinea
VALUES (OLD.idVta,
        OLD.fecha,
        OLD.idLinea,
        OLD.cantidad,
        USER(),
        curdate(),
        curtime(),
        NULL,
        "Delete"
        );
    CALL motivo_delete_VL("algun motivo");
END $$
```

7.5.1.2. TRIGGER aft_upd_ventasLineas_logVentasLinea

Este trigger ejecutará dos sentencias, una para registrar los datos antes de actualizarse y otra para registrarlos ya actualizados.

En la tabla logVentasLinea se registrará con un trigger after update: todos los campos de la tabla ventaslinea (en un registro antes de que se actualicen y en otro consecutivo, ya actualizados), el usuario, la fecha, la hora, el motivo por el que lo elimina (será null en este caso), la acción realizada

```
DROP TRIGGER IF EXISTS `aft_upd_ventasLineas_logVentasLinea`;

DELIMITER $$
$$
CREATE TRIGGER `aft_upd_ventasLineas_logVentasLinea`
AFTER UPDATE ON ventaslinea
FOR EACH ROW
BEGIN
INSERT INTO logVentasLinea
VALUES  (OLD.idVta,
         OLD.fecha,
         OLD.idLinea,
         OLD.cantidad,
         USER(),
         curdate(),
         curtime(),
         NULL,
         "Update");
INSERT INTO logVentasLinea
VALUES  (NEW.idVta,
         NEW.fecha,
         NEW.idLinea,
         NEW.cantidad,
         USER(),
         curdate(),
         curtime(),
         NULL,
         "Update");
END $$
```

7.5.2. TRIGGERS TABLA VENTASDESCUENTO

- Se crea una tabla log para los delete y update de la tabla ventasdescuento
- La tabla contendrá los archivos antes de eliminarse, antes de actualizarse y luego de actualizarse
- Se diferenciará la acción realizada por la columna acción.

```
CREATE TABLE IF NOT EXISTS `logVentasDescuento` (`idVtaDto` INT NOT NULL,
                                                 `fecha` DATE NULL,
                                                 `idDto` INT NULL,
                                                 `idGenero` INT NULL,
                                                 `idTrans` INT NULL,
                                                 `cantidad` INT NULL,
                                                 `user` VARCHAR (255) NOT NULL,
                                                 `FechaDelete` DATE NOT NULL,
                                                 `HoraDelete` TIME NOT NULL,
                                                 `motivo` VARCHAR(255) DEFAULT NULL,
                                                 `accion` VARCHAR (15));
```

- Se crea un SP para colocar el motivo por el cual se elimina algún registro

```
DROP PROCEDURE IF EXISTS `motivo_delete_VD`;

DELIMITER $$
 $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `motivo_delete_VD`(in p_motivo varchar (255)) BEGIN
UPDATE `logVentasDescuento`
SET `motivo` = p_motivo WHERE motivo IS NULL ;
END$$
```

7.5.2.1. TRIGGER bef_del_ventasDescuento_logVentasLinea

En la tabla logVentasDescuento se registrará con un trigger before delete: todos los campos de la tabla ventasdescuento que se eliminan, el usuario, la fecha, la hora, el motivo por el que lo elimina (llamando al SP "motivo_delete_VD"), la acción realizada

```
DROP TRIGGER IF EXISTS `bef_del_ventasdescuento_logVentasdescuento`;

DELIMITER $$
$$
CREATE TRIGGER `bef_del_ventasdescuento_logVentasdescuento`
BEFORE
DELETE ON ventasdescuento
FOR EACH ROW BEGIN
INSERT INTO logVentasDescuento
VALUES (OLD.idVtaDto,
        OLD.fecha,
        OLD.idDto,
        OLD.idGenero,
        OLD.idTrans,
        OLD.cantidad,
        USER(),
        curdate(),
        curtime(),
        NULL,
        "Delete"
        );
    CALL motivo_delete_VD("algun motivo");
END $$
```

7.5.2.2. TRIGGER aft_upd_ventasDescuento_logVentasDescuento

Este trigger ejecutará dos sentencias, una para registrar los datos antes de actualizarse y otra para registrarlos ya actualizados.

En la tabla logVentasDescuento se registrará con un trigger after update: todos los campos de la tabla ventasdescuento (en un registro antes de que se actualicen y en otro consecutivo, ya actualizados), el usuario, la fecha, la hora, el motivo por el que lo elimina (será null en este caso), la acción realizada.

```
 DROP TRIGGER IF EXISTS `aft_upd_ventasDescuento_logVentasDescuento`;

DELIMITER $$
$$
CREATE TRIGGER `aft_upd_ventasDescuento_logVentasDescuento`
AFTER UPDATE ON ventasDescuento
FOR EACH ROW
BEGIN
INSERT INTO logVentasDescuento VALUES  (OLD.idVtaDto,
                                        OLD.fecha,
                                        OLD.idDto,
                                        OLD.idGenero,
                                        OLD.idTrans,
                                        OLD.cantidad,
                                        USER(),
                                        curdate(),
                                        curtime(),
                                        NULL,
                                        "Update"
                                        );

INSERT INTO logVentasDescuento VALUES  (NEW.idVtaDto,
                                        NEW.fecha,
                                        NEW.idDto,
                                        NEW.idGenero,
                                        NEW.idTrans,
                                        NEW.cantidad,
                                        USER(),
                                        curdate(),
                                        curtime(),
                                        NULL,
                                        "Update"
                                        );
END $$
```

7.6. Usuarios (Data Control Language)

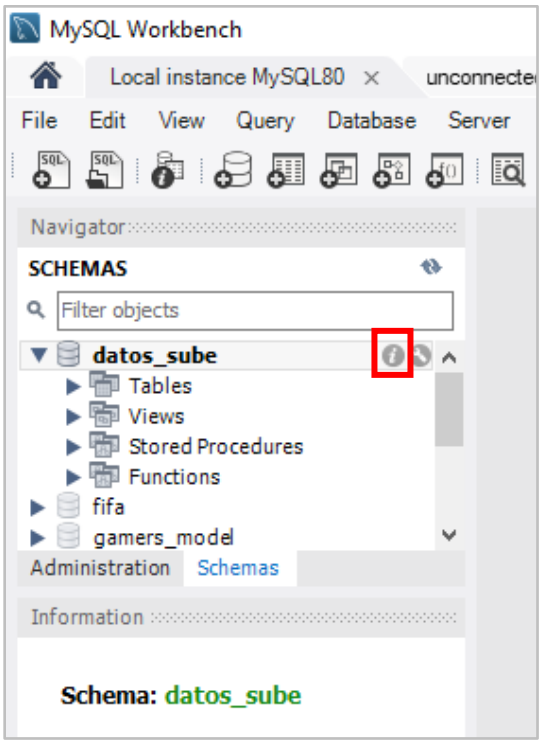
El Lenguaje de Control de Datos (DCL) permite definir diferentes usuarios dentro del motor de base de datos MySQL, y establecer para cada uno de ellos, permisos totales, parciales, o negar el acceso sobre los diferentes Objetos que conforman la Base de Datos.

Para el proyecto realizaremos lo siguiente:

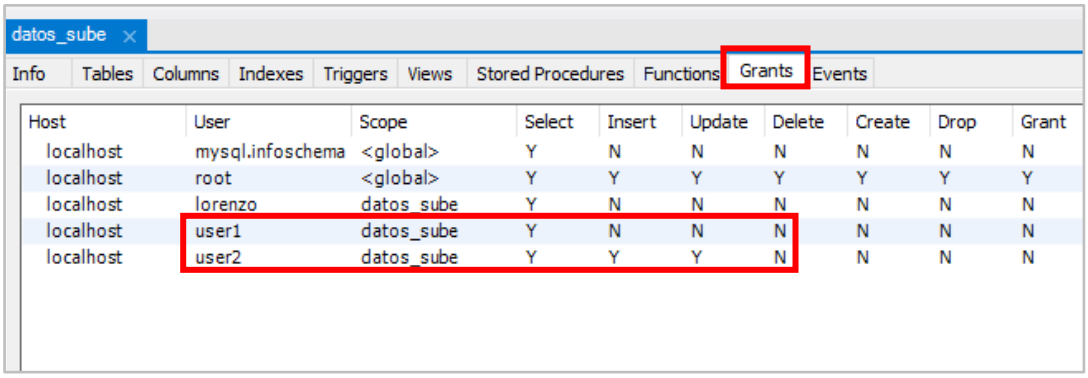
- Se crearán dos usuarios con contraseñas para cada uno y se les otorgarán distintos permisos.
- A un usuario se le otorgará permiso solo de lectura, a otro usuario se le otorgarán permisos de lectura, inserción y modificación, y ninguno deberá tener permisos de eliminación.

```
/* Creamos los usuarios user1 y user2 con las clausulas CREATE USER
y les asignamos sus repectivas contraseñas con la clausula IDENTIFIED BY */
CREATE USER user1@localhost IDENTIFIED BY "clave1";
CREATE USER user2@localhost IDENTIFIED BY "clave2";

/* El usuario user1 tendrá permiso de solo lectura a todos los objetos de la base
el usuario user2 tendrá permisos de insertar y modificar registros*/

/* Con la clausula GRANT otorgamos los permisos deseados y
con la clausula TO indicamos al usuario que se le otorga dichos permisos*/

-- Se le otrorga permisos solo de lectura a user1 sobre todos los objetos de la db datos_sube
GRANT SELECT ON datos_sube.* TO user1@localhost;

-- Se le otroga permisos de lectura, inserción y modificación a user2 sore todos los objetos de la db datos_sube
GRANT SELECT, INSERT, UPDATE ON datos_sube.* TO user2@localhost;

-- Para estar seguros que no pueden eliminar quitamos los permisos de ambos usuarios
REVOKE DELETE ON datos_sube.* FROM user1@localhost,user2@localhost;
```

Si hacemos click en la información del schema



Y luego en la ventana de grants, podremos ver todos los usuarios generados para esa base de datos y los permisos que cada uno posee



Host	User	Scope	Select	Insert	Update	Delete	Create	Drop	Grant
localhost	mysql.infoschema	<global>	Y	N	N	N	N	N	N
localhost	root	<global>	Y	Y	Y	Y	Y	Y	Y
localhost	lorenzo	datos_sube	Y	N	N	N	N	N	N
localhost	user1	datos_sube	Y	N	N	N	N	N	N
localhost	user2	datos_sube	Y	Y	Y	N	N	N	N

7.7. Transacciones (Transaction Control Language)

Se conoce como Transaction Control Language (o TCL) al grupo de sentencias que se utilizan para gestionar los cambios realizados por las sentencias DML y agruparlas en transacciones lógicas. El papel de TCL es fundamental ya que, a través del mismo, controlamos las cláusulas u operaciones DML.

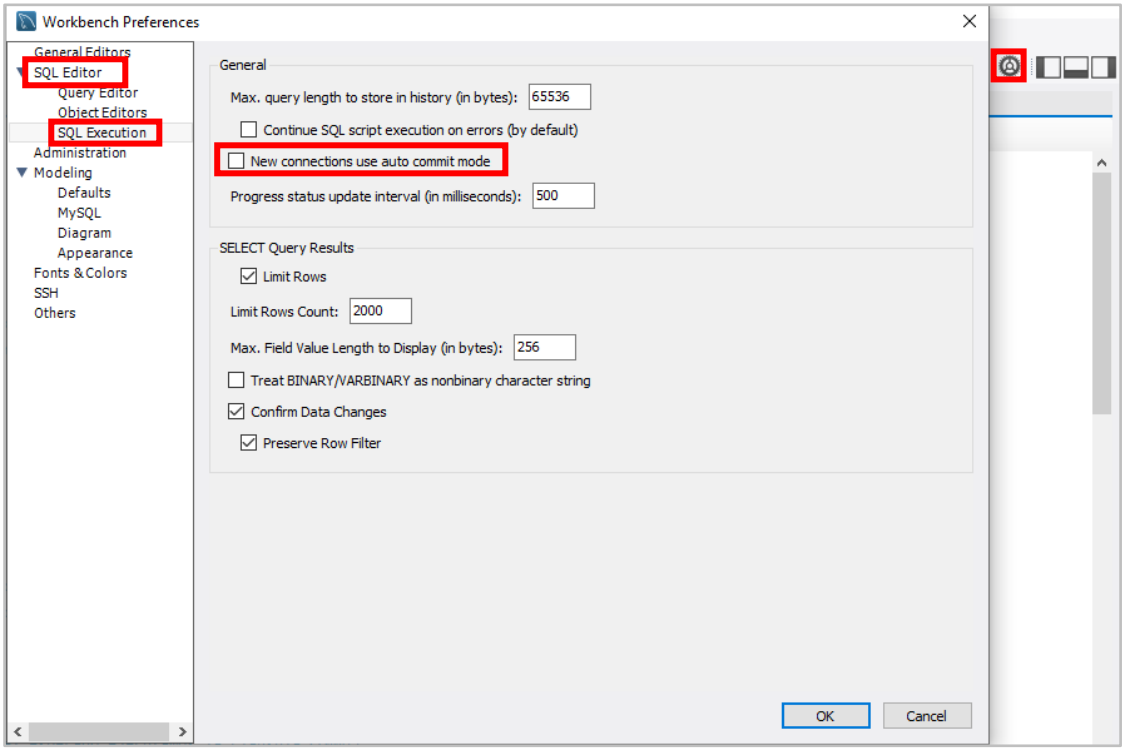
Transaction Control Language incluye tres sentencias claves:

- COMMIT es un comando que sirve para "confirmar" las operaciones realizadas sobre una o más tablas. Una vez ejecutados los cambios son permanentes y se ponen a disposición de los demás usuarios.
- ROLLBACK se ocupa de "deshacer" los cambios, o volver al estado permanente anterior. Debemos tener presente que, el mismo, solo funciona si no se ejecutó antes el comando COMMIT.
- SAVEPOINT nos permite establecer un punto de recuperación dentro de la transacción, utilizando un identificador. Podemos hacer un ROLLBACK deshaciendo sólo las instrucciones que se han ejecutado hasta un determinado SAVEPOINT que se indique. Al ejecutar el COMMIT, todo SAVEPOINT que hayamos establecido se perderá dado que, en este punto, ROLLBACK, no puede volver a ejecutarse.

Es importante en este destacar en este punto que se debe desactivar el autocommit que el SGDB trae por defecto.

Esto lo podemos hacer de dos maneras:

- Ejecutando el comando set autocommit = 0; (esto no es permanente)
- Desde las preferencias del SGBD seleccionar en el apartado de SQL Editor la pestaña de SQL Execution y destildar la opción "New connections use auto commit mode" (luego de reiniciar el SGBD el cambio es permanente)



Para el proyecto se realizarán las siguientes consignas:

- Se eliminarán registros de la tabla ventaslinea
- Se crean 2 SAVEPOINT en esta sentencia, uno antes de ejecutar el DELETE y otro después de ejecutarlo

```
use datos_sube;
set autocommit = 0;

START TRANSACTION;

SAVEPOINT Inicioeliminacion;
DELETE FROM ventaslinea
WHERE
    idvta BETWEEN 10 AND 15;
SAVEPOINT Eliminacion;
```

- El comando ROLLBACK hará referencia al SAVEPOINT anterior a ejecutar el DELETE

```
SAVEPOINT Inicioeliminacion;
DELETE FROM ventaslinea
WHERE
    idvta BETWEEN 10 AND 15;
SAVEPOINT Eliminacion;

/*
En este punto los cmabios estan generados pero no aplicados,
es decir podemos usar el ROLLBACK para realizar alguna modificación en la sentencia sin afectar la tabla
*/
-- ROLLBACK TO Inicioeliminacion;

/*
Si tenemos la seguridad de que el resultado devuelto es el esperado ejecutamos la clausula COMMIT
donde confirmamos las modificaciones en la db y ya no tenemoos opcion de utilizar los SAVEPOINT anteriores.
En este caso la tabla VentasLineas contiene un TRIGGER que copia todos los registros eliminados a la tabla logventasdescuento
el cual no se disparará hasta que se ejecute la clausula COMMIT
*/
-- COMMIT;
```

- Se insertarán registros en la tabla VentasDescuento
- Se crean 3 SAVEPOINT en esta sentencia, uno antes de ejecutar el INSERT, otro después de los primero 4 registros y otro cuando finaliza.

```
START TRANSACTION;
 SAVEPOINT InicioInsercion;
INSERT INTO VentasDescuento VALUES (null,'2022-01-07',6,1,3,792);
INSERT INTO VentasDescuento VALUES (null,'2022-01-03',1,2,4,1052);
INSERT INTO VentasDescuento VALUES (null,'2022-01-04',7,2,4,870);
INSERT INTO VentasDescuento VALUES (null,'2022-07-05',7,3,3,841);
SAVEPOINT lote1;
INSERT INTO VentasDescuento VALUES (null,'2022-01-02',7,1,4,1090);
INSERT INTO VentasDescuento VALUES (null,'2022-01-08',2,2,4,784);
INSERT INTO VentasDescuento VALUES (null,'2022-01-03',2,3,2,232);
INSERT INTO VentasDescuento VALUES (null,'2022-01-06',6,2,1,1031);
SAVEPOINT lote2;

/*
En este punto estan insertados los nuevos registros en la tabla VentasDescuento, pero no confirmados y
podríamos ejecutar un ROLLBACK a cualquiera de los SAVEPOINT generados.
```

- Se agregará una sentencia de eliminación del SAVEPOINT de los primeros cuatro registros insertados.

En el caso de ejecutar la sentencia RELESE SAVEPOINT liberaríamos de la memoria el SAVEPOINT especificado y no podríamos volver a utilizarlo.

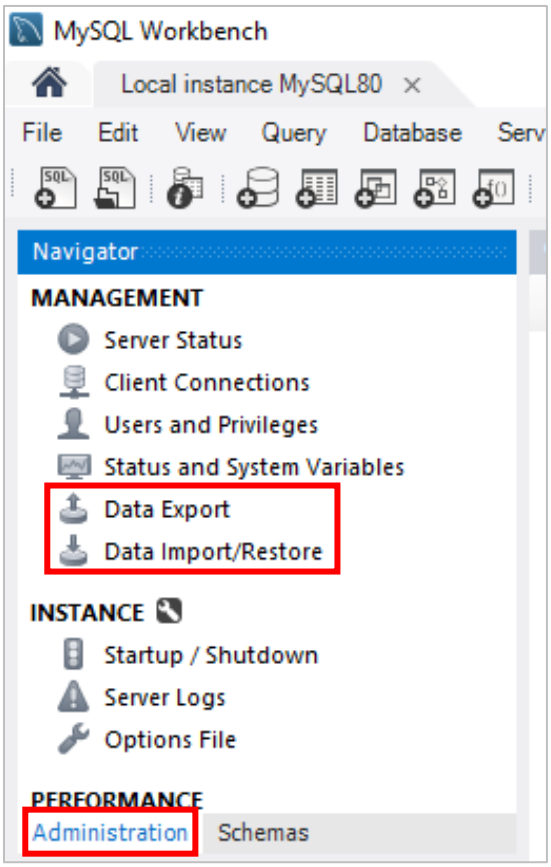
```
 RELEASE SAVEPOINT lote1;

/*
Luego de ejecutar esta sentencia solo podriamos realizar un ROLLBACK a lote2 o InicioInsercion,
lote1 ya no existiria como SAVEPOINT
*/
```

7.8. Backup

Los backup o copias de seguridad son claves para el mantenimiento de las bases de datos. Es importante realizarlas de forma periódica para evitar pérdida de datos en eventuales fallas.

El SGBD MySQL nos ofrece dos maneras de realizar Backups para respaldar nuestros datos

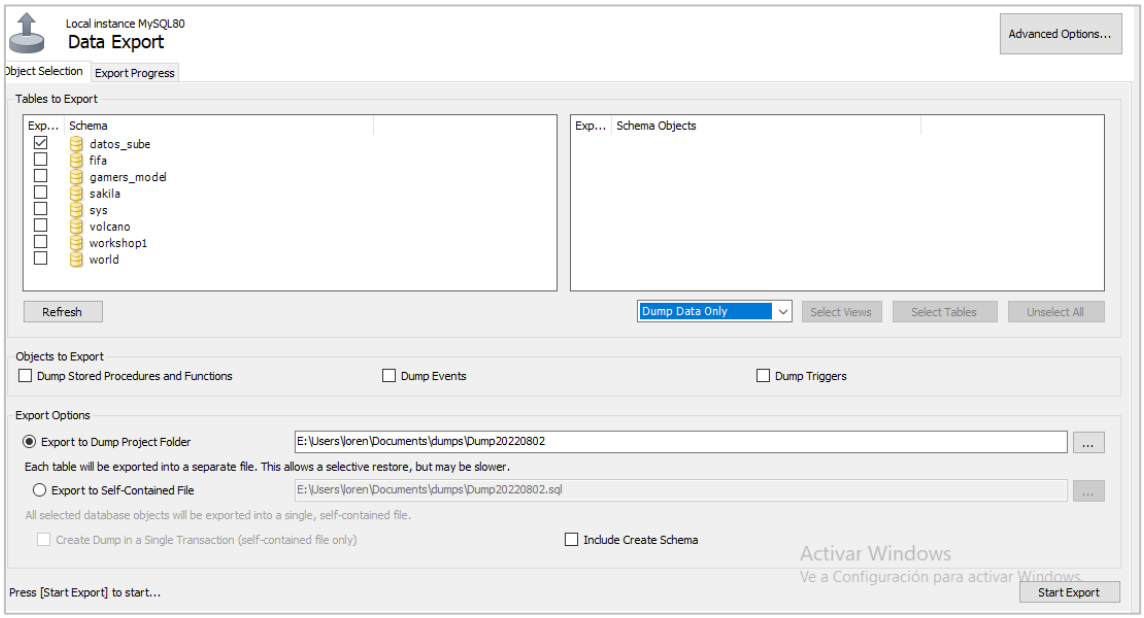
- MySQL incluye la herramienta mysqldump dentro del motor de base de datos para gestionar las copias de seguridad. Debemos utilizarla a través de la Línea de Comandos o Terminal, de nuestro sistema operativo.
- Mysql Workbench cuenta también con un apartado denominado Administration, donde tenemos un set de herramientas varias. Dentro de este panel encontramos las opciones Data Export y Data Import/Restore las cuales nos permitirán realizar el backup y posteriormente su recuperación.



Dentro del Data Export podremos personalizar nuestra exportación.

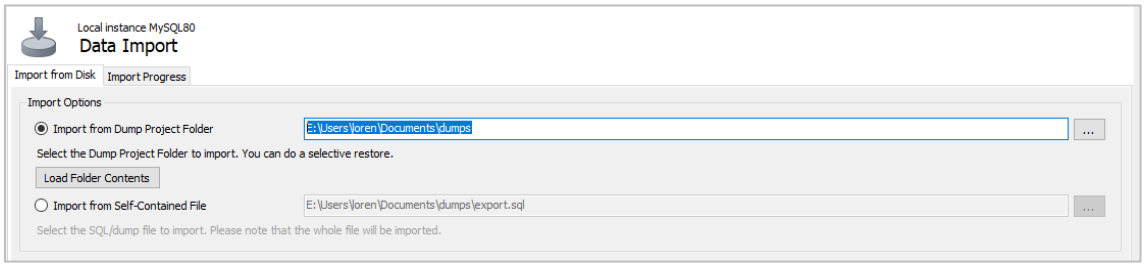
- Podemos seleccionar los schemas deseados.
- Al seleccionar algún schema podremos elegir que tablas y vistas queremos exportar.
- Podemos elegir las opciones:
    - Dump data only: para exportar solo los datos almacenados.
    - Dump structure only: para exportar solo la estructura de los objetos seleccionados.
    - Dump data and structure: exporta tanto los datos como la estructura de los objetos seleccionados.
- Podemos seleccionar si queremos exportar eventos, funciones y procedimientos almacenados y/o Triggers

- También se puede elegir qué tipo de backup realizar:
    - Export to Dump Project Folder: que vuelca a una carpeta de proyecto a la cual le podemos indicar la ruta donde se guardará la o las bases de datos generado un archivo .sql para cada objeto de la base.
    - Export to Self-Contained File: que genera un archivo .sql de la información seleccionada. Esta opción también permite seleccionar la carpeta donde se almacenará, y especificar el nombre del archivo.

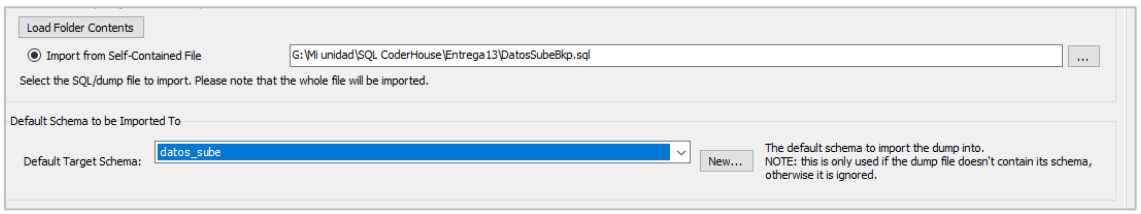


Restaurar los datos con el SGBD de MySQL debemos entrar a la opción Data Import/Restore en el apartado Administration y allí podemos seleccionar dos opciones:

- Import from Dump Proyect Folder: esta opción es válida si cuando hicimos la exportación elegimos el método Export to Dump Project Folder, por lo que seleccionaremos la carpeta que contiene los archivos de backup



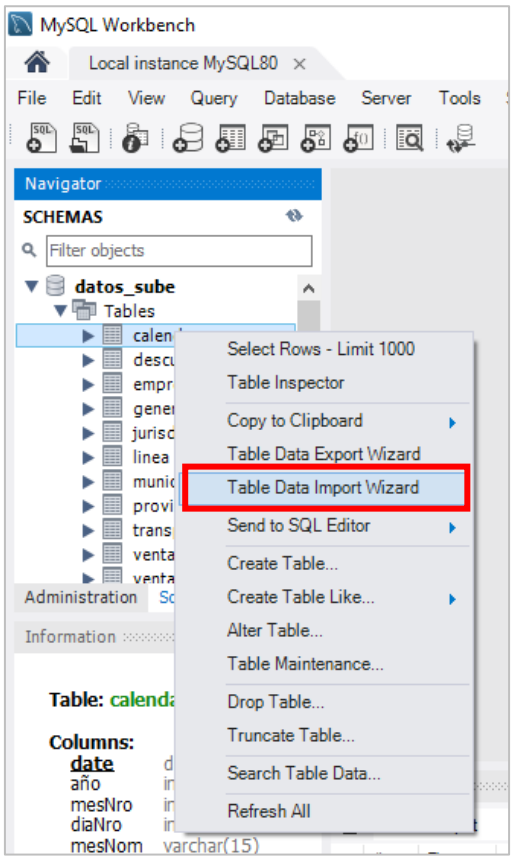
- Import from Self-Conteined File: en esta opción seleccionaremos el archivo .sql generado en el caso de haber realizado el backup con el método Export to Self-Contained File y posteriormente el schema donde queremos restaurarlo o elegimos un nombre distinto si queremos crear un nuevo schema para la restauración.



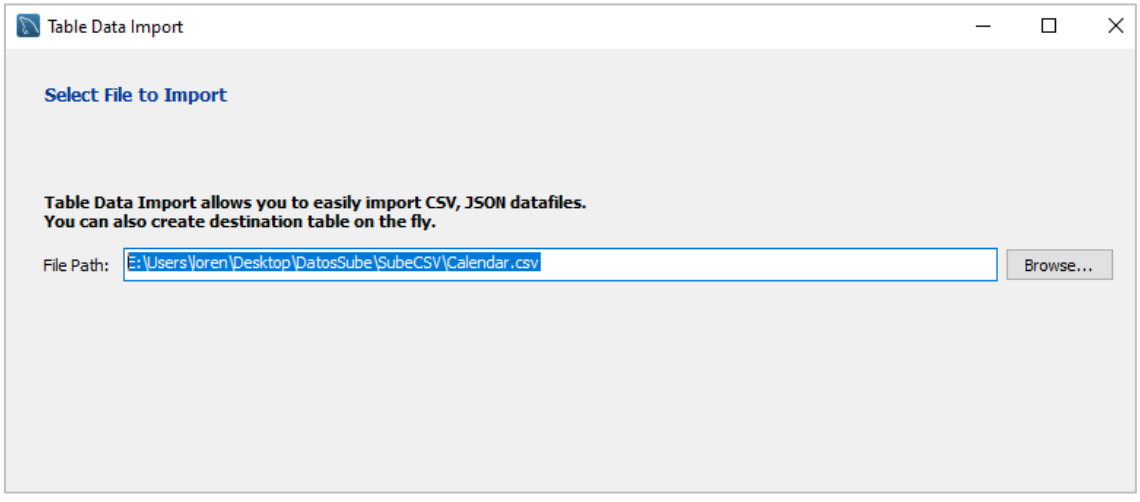
8. INSERCIÓN DE DATOS

8.1. Explicación importación .csv

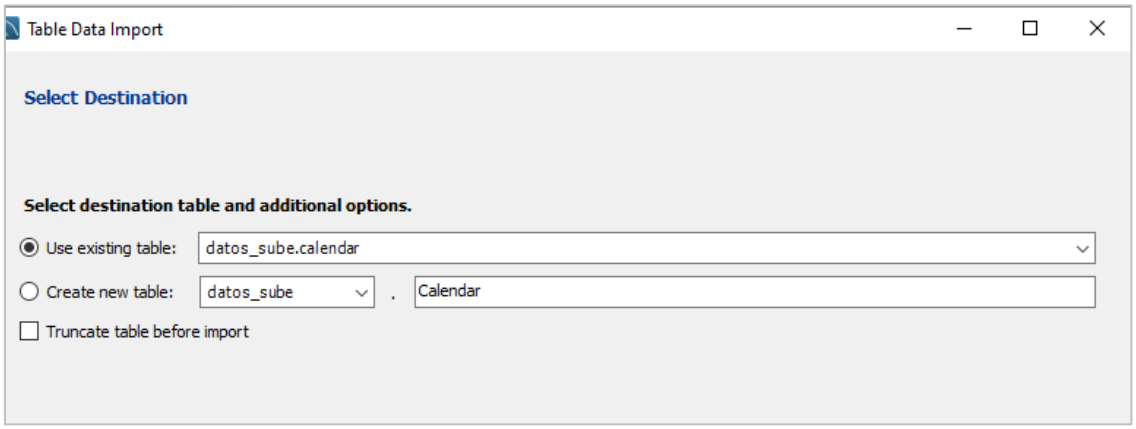
1) En la tabla deseada hacemos click derecho y seleccionamos el asistente para importación de datos



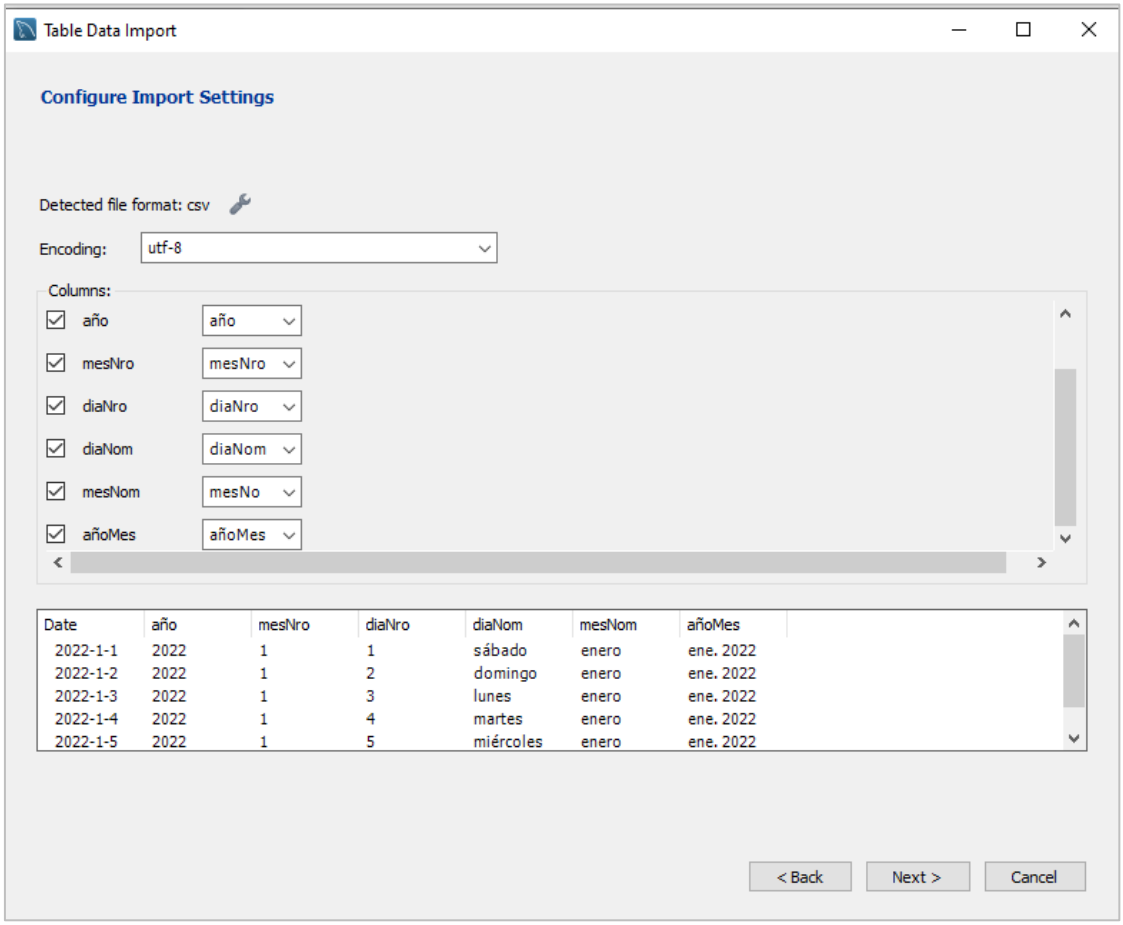
2) Seleccionamos el archivo (.csv en este caso)



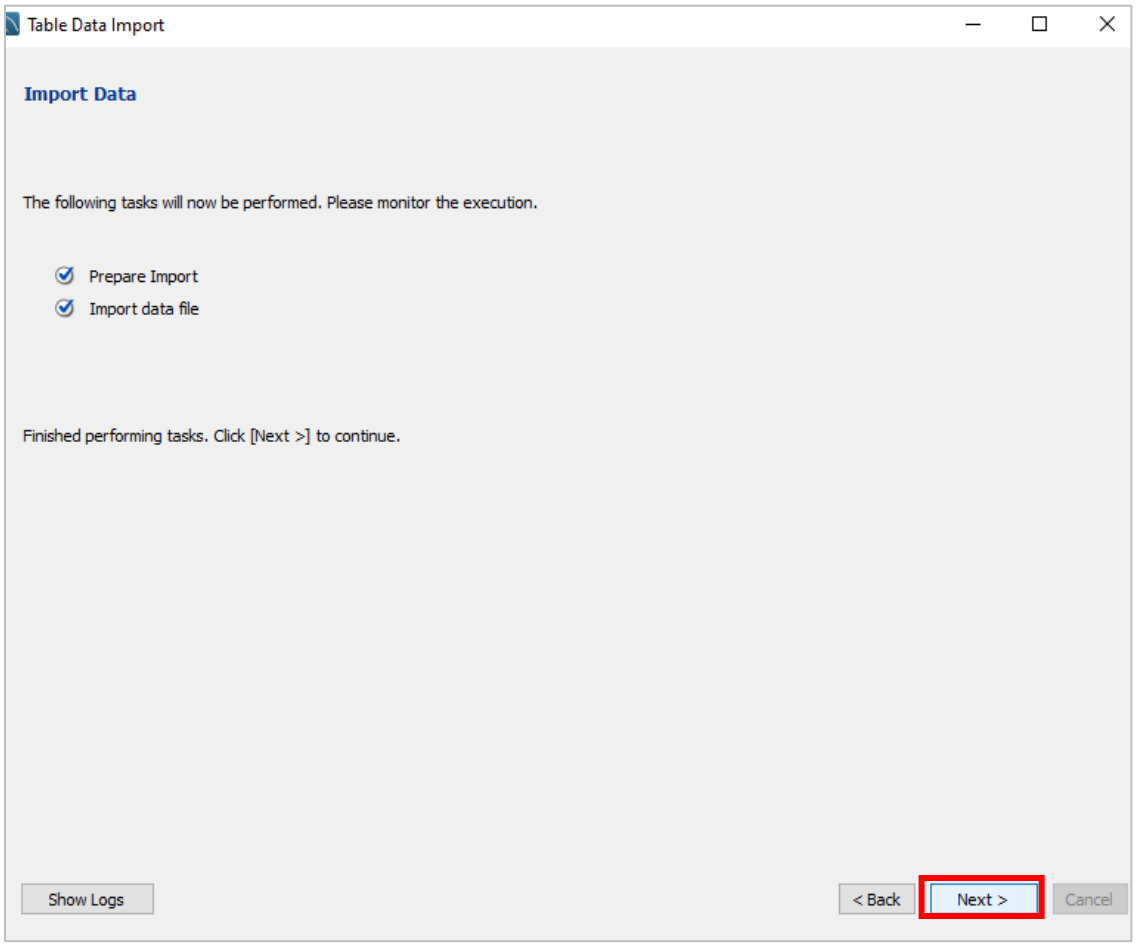
3) Seleccionamos si usamos una tabla existente, creamos una tabla nueva o si hacemos truncate de una tabla existente e importamos ahí.



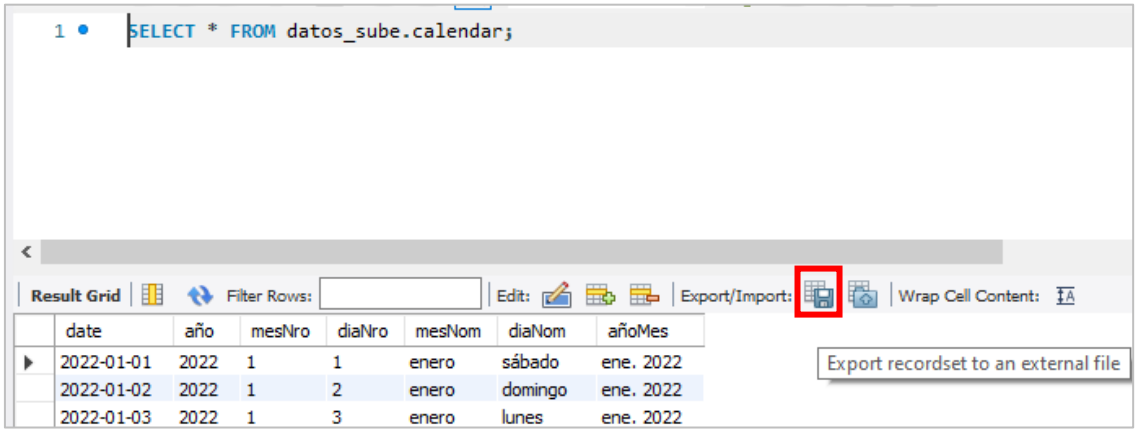
4) Corroboramos que los campos correspondan a sus columnas especificas



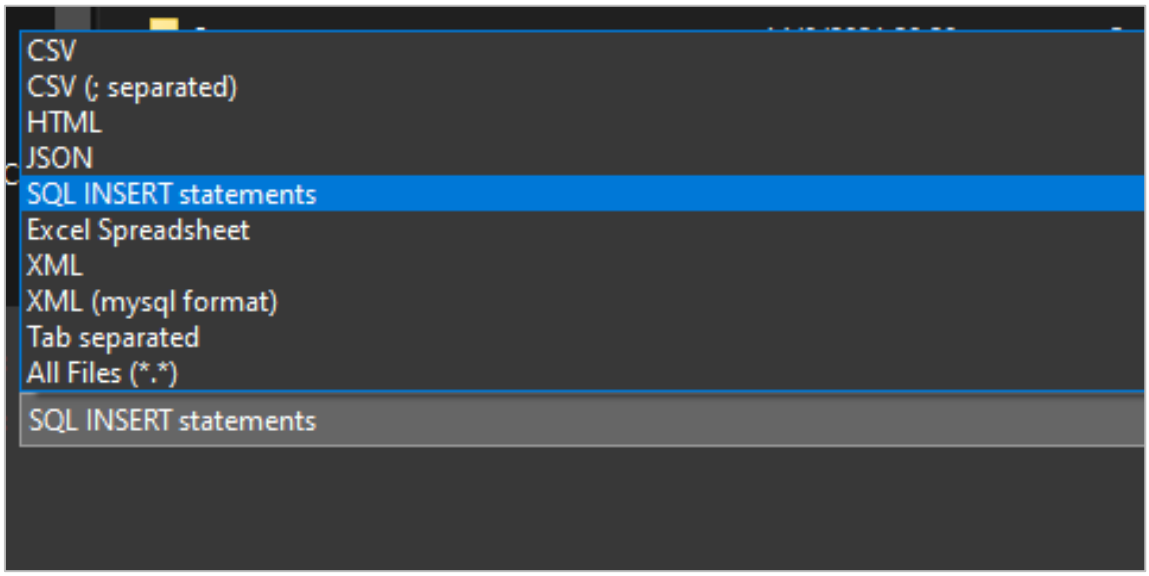
5) Importamos los datos apretando Next



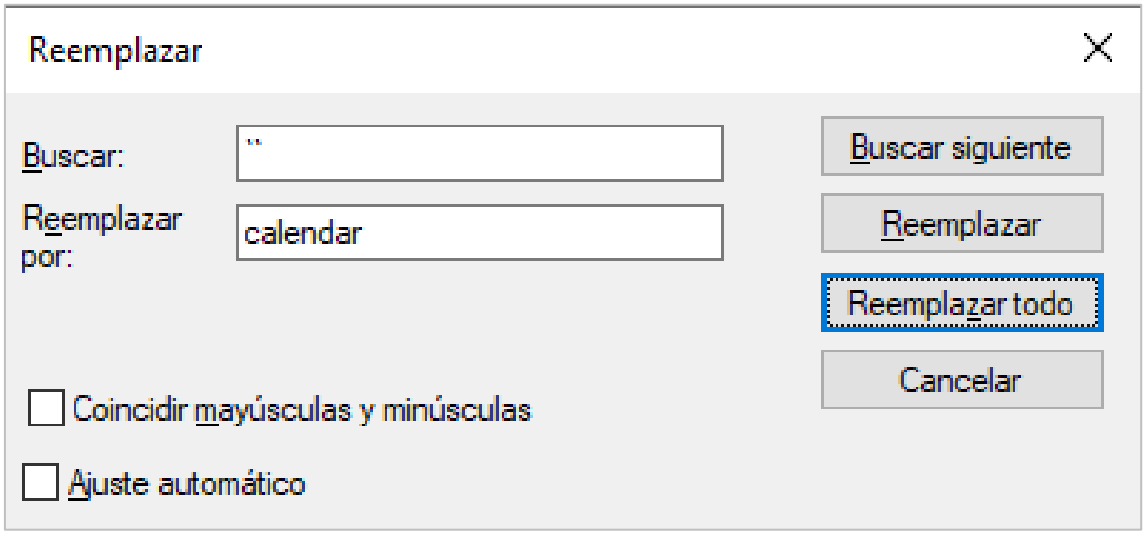
6) Seleccionamos la tabla para ver en la grilla del SGBD y hacemos click en "Export recordset on a external file"



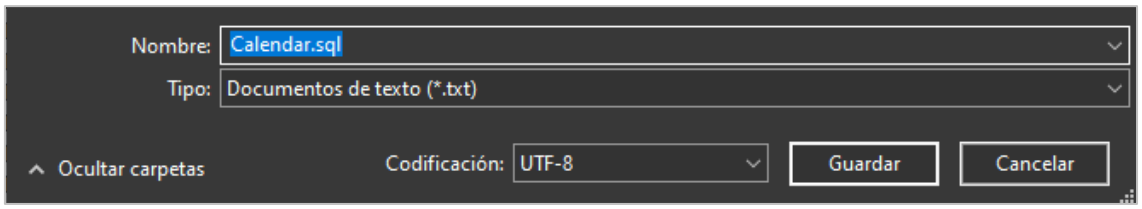
7) Seleccionamos el tipo de archivo SQL Inster Satement



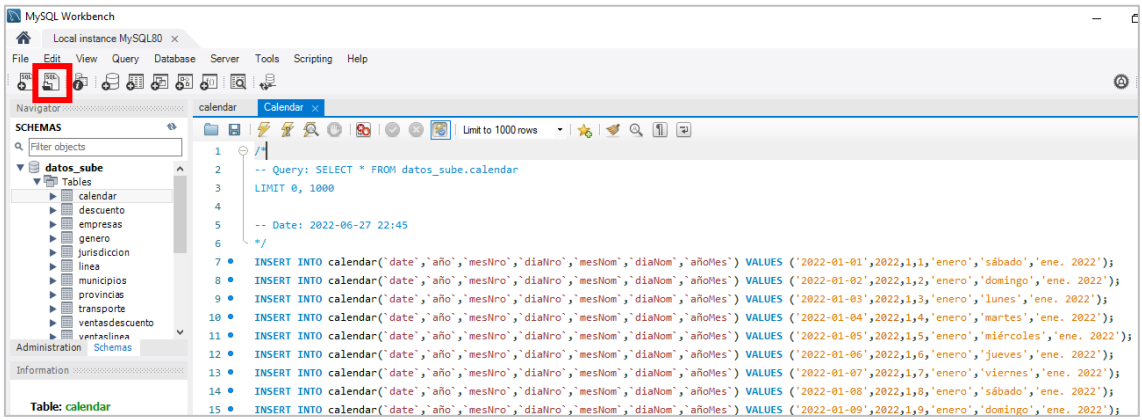
8) Abrimos el archivo con el bloc de notas o algun editor de texto y cambiamos las `` por el nombre de la tabla



9) Guardamos el archivo como .sql



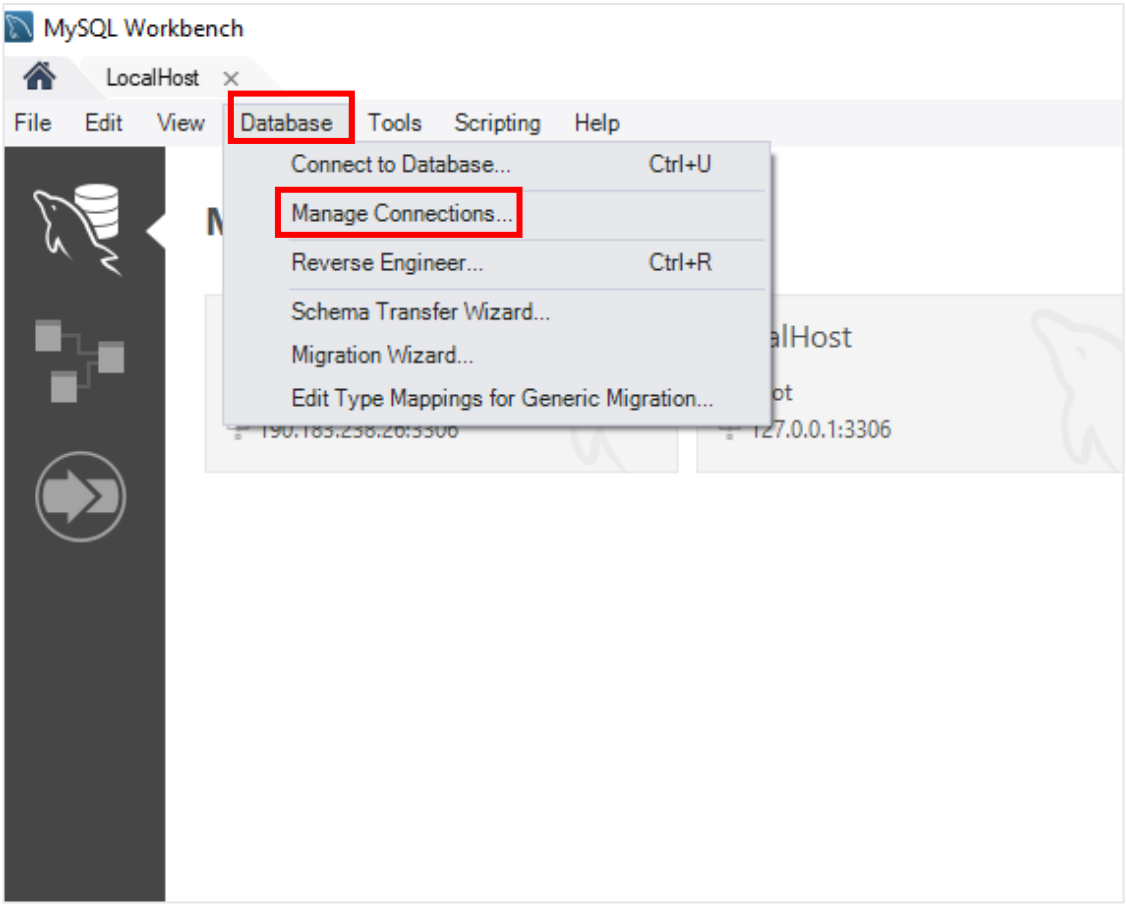
10) Abrimos el script .sql guardado y lo ejecutamos en el SGBD



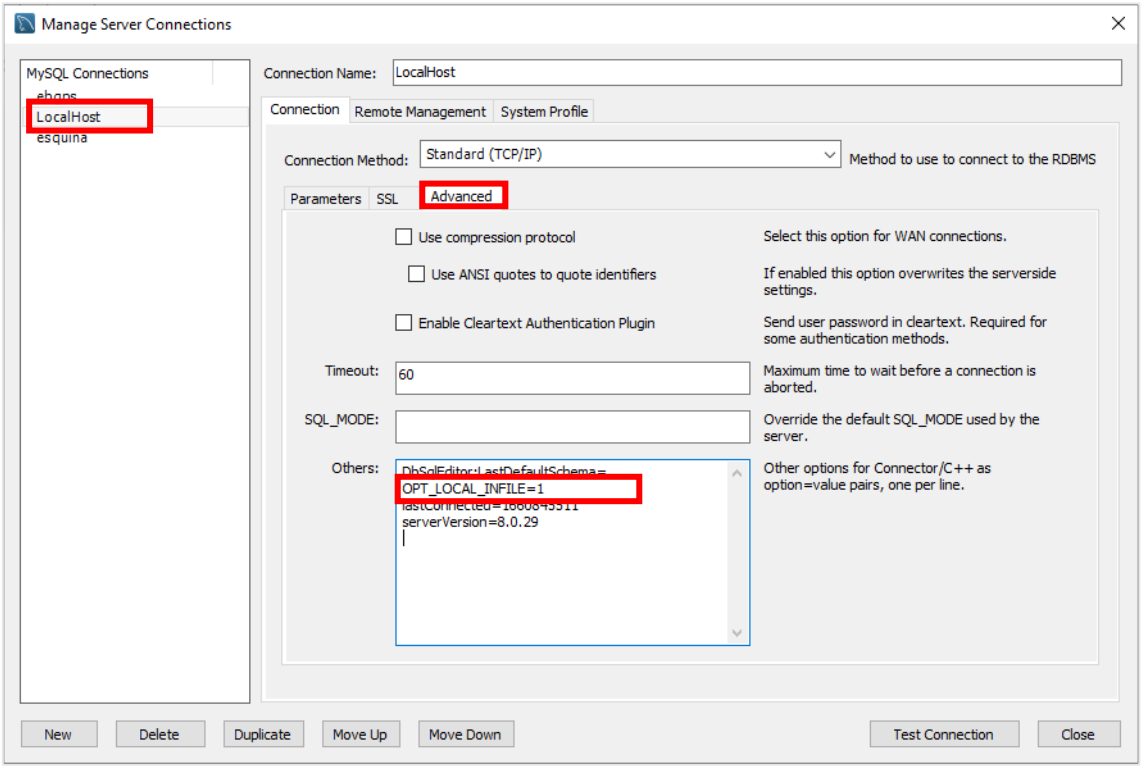
8.2. Explicación importación load data local infile

Otra opción es ejecutar la sentencia load data local infile, pero para eso antes debemos preparar el SGBD.

Primero debemos ir al inicio de MySQL Workbench y en el apartado Database seleccionar Manage Connections



Alli seleccionaremos nuestra conexión y haremos click en el apartado Advace, en la sección Others escribiremos la sentencia OPT_LOCAL_INFILE = 1 y le daremos close



Luego ya podemos volver a nuestro script, donde ejecutaremos la sentencia

SET GLOBAL local_infile=1;

Una vez realizados estos pasos por unica vez en el SGBD podremos continuar con la sentencia load data local infile.

Con la sentencia load data local infile especificamos la ruta del archivo

Con la sentencia insert into indicamos en que tabla se insertaran los datos

Con la sentencia charsert especificar la codificación del archivo

Con la sentencia fields terminated by especificar el delimitador de los campos

Con la sentencia terminated by si posee salto de linea,

Con la sentencia ignore ignorar alguna linea superior

y entre parentesis el orden de los campos a insertar.

```
use datos_sube;
load data local infile "E:/Users/loren/Desktop/DatosSube/SubeCSV/Calendar.CSV"  -- ubicación del archivo
into table calendar -- tabla a insertar
charset cp1250  -- codificación del archivo
fields terminated by ';' -- delimitador de los campos
terminated by '\r\n' -- salto de linea
ignore 1 lines -- ignora la primer linea
(date,año,mesNro,diaNro,diaNom,mesNom,CodAñoMes,añoMes); -- orden de los campos a insertar
```

Lo beneficioso de esta última opcion es que es más rápida en el tiempo de carga y que se pueden ejecutar varias sentencias en un solo script.

9. *INFORMES GENERADOS EN BASE A LA INFORMACIÓN ALMACENADA EN LAS TABLAS*

Los informes generados en base a la información contenida en las tablas pueden ser de carácter cuantitativo, estadístico, predictivo o preventivo. Como por ejemplo con las vistas generadas podemos determinar en qué provincia hay mayor concentración de empresas y líneas de transportes, que cantidad de pasajeros hay por día de la semana, cuáles son las provincias con mayor cantidad de pasajeros transportados, cual es la cantidad de pasajeros por género o por tipo de descuento.

Para hacer visible esta información he realizado un informe con la herramienta Power BI al que se puede acceder haciendo click aquí.

Descargable:
https://drive.google.com/file/d/125RzWgWtQB9FwzKTHFIx9myARmL8Uc3b/view?usp=sharing

10. HERRAMIENTAS Y TECNOLOGÍAS QUE UTILIZASTE

Las tecnologías utilizadas para el desarrollo de este proyecto fueron:

- MySQL Workbech: Herramienta utilizada como Sistema Gestor de Bases de Datos, la cual se utilizó para la creación de los schemas, tablas, vistas y demás objetos de la base de datos.
- Plataforma CoderHouse: Plataforma utilizada para la comunicación entre los docentes y alumnos del curso.
- Zoom: Plataforma utilizada para el dictado de las clases por los profesores de CoderHouse
- Notepad ++: Editor de texto utilizado para realizar manipulación de los datos, reemplazar texto o cambiar la codificación del mismo.
- Microsoft Excel: Herramienta utilizada para exploración de los datos contenidos en archivos .csv originados por el dataset utilizado.
- Power Query: Herramienta utilizada previamente a la carga de datos en el SGDB para la normalización de los datos provenientes del dataset.
- Power BI: Herramienta para visualización y análisis de datos con los informes generados durante el proyecto.
- Google Drive: Plataforma utilizada para subir los archivos presentables y compartirlos.
- ILovePDF: Página web que permite transformar archivos a PDF.

11. ORIGEN DE DATOS

En este caso la fuente de los archivos es tomada de un dataset público del Ministerio de Transporte de la Nación, el cual lo que permite descargar son .csv sin ninguna normalización.

https://datos.transporte.gob.ar/dataset/sube---cantidad-tarjetas-usuarios-por-fecha

https://datos.transporte.gob.ar/dataset/sube---cantidad-transacciones-usos-por-fecha

Adjunto links correspondientes de los archivos descargados.

https://drive.google.com/drive/folders/136HvPhrXRGuomxHyv3o3JU9wG4Ow-rpu?usp=sharing

Por lo que procedí a normalizar las distintas tablas según mi DER anteriormente presentado, generando los siguientes .csv

https://drive.google.com/drive/folders/15-iVrM3noWEKrkmog0n_DlERIy695DN_?usp=sharing

Generados los .csv y siguiendo los pasos anteriormente explicados logré crear los archivos de importación .sql (por el gran volumen de datos solo contienen una parte de los registros)

https://drive.google.com/drive/folders/13E5W1aEd1lHOe9usUMlZTLV7wNsdoihT?usp=sharing

Desde el archivo LoadDataLocalInFile se pueden ejecutar todos los registros completos.

https://drive.google.com/file/d/13pOv35kjfvDlznw-wEMRoQvtQCMaORUF/view?usp=sharing

12. SCRIPTS

12.1. Script creación Tablas

https://drive.google.com/file/d/10s2WiJf_iOc3E6XHPvRjNWs6hracEBxX/view?usp=sharing

12.2. Script inserción de datos

- Insert Into

https://drive.google.com/file/d/11uzXVNe5GIAuCTMMSuoW0u2uJym9DZA4/view?usp=sharing

- Load Data Local Infile

https://drive.google.com/file/d/13pOv35kjfvDlznw-wEMRoQvtQCMaORUF/view?usp=sharing

12.3. Script creación vistas, funciones, SP y Triggers

https://drive.google.com/file/d/11wwAnXb-FnCHJ1xszi084YqSkV7MMwU2/view?usp=sharing

12.4. Script creación Usuarios

https://drive.google.com/file/d/121A0-zjfemgC8unNZFV5qHAJUnxK7-38/view?usp=sharing

12.5. Script Backup

- Export from Self-Conteined File

https://drive.google.com/file/d/1317dKMap6l7qYZ0Q3oc5YQscaCyhmeNX/view?usp=sharing

- Export to Dump Project Folder

https://drive.google.com/drive/folders/12zI3a-LxUPbtvs3RKIk_2SgrtL4_35oo?usp=sharing

12.6. Script ejecutable

https://drive.google.com/file/d/134zLKxDrBXBOF6w5wR6kTw3A7MDoreYX/view?usp=sharing