Universidad de Sonora
División de Ciencias Exactas y Naturales
Maestría en Ciencia de Datos

Manejo de base de datos con SQL

Dr. Juan Pablo Soto Barrera
Introducción a la Ciencia de Datos y sus Metodologías

PRESENTA
Rodrigo Iván González Valenzuela

Hermosillo, Sonora. Noviembre de 2022

I. Introducción

Como científico de datos, resulta indispensable saber manejar conjuntos de datos y prepararse para toda clase de escenario al momento de obtenerlos; ya sea trabajando junto a una empresa con un departamento que administre las bases de datos, o bien desempeñarse como ingeniero o arquitecto de datos para conseguir esa información verdaderamente útil.

SQL es un lenguaje que permite la organización, manipulación y consulta de datos —especialmente grandes cantidades— que tiene una sintaxis sencilla de interpretar, es de código abierto y cuenta con una gran comunidad de soporte [1].

Este proyecto presenta una aplicación de SQL para la gestión de una gran cantidad de registros a través de MySQL Workbench 8.0.31, para después ser consultados con Python y generar una visualización de los datos.

Decidí utilizar los archivos de datos abiertos provistos por el Secretariado Ejecutivo del Sistema Nacional de Seguridad Pública con tema de incidencia delictiva por municipio de la república mexicana [2], además de hacer uso de los datos georreferenciados que se encuentran en el sistema de consultas del INEGI [3].

Objetivo general: Importar, comprender y normalizar el conjunto de datos crudos de incidencia delictiva en México en el año 2015 haciendo uso de MySQL, y realizar una consulta a la base de datos obtenida con lenguaje Python para generar un mapa descriptivo de los datos correspondientes a Sonora.

Objetivos específicos:

i. Importar el archivo CSV de los datos crudos a una tabla en una base de datos.
ii. Explorar los datos: cantidad de registros, nombre de columnas, tipos de variables y valores presentes en estas.
iii. Diseñar un modelo de entidad-relación para la base de datos.
iv. Generar tablas, columnas y relaciones de acuerdo al modelo propuesto.
v. Crear funciones almacenadas.
vi. Crear una vista con la consulta de los datos de Sonora.
vii. Conectarse al servidor de MySQL con Python para almacenar la vista en un dataframe.
viii. Presentar un mapa con los datos de la consulta.

II. Importación de los datos

Al estar el dataset en formato CSV, la metodología que seguí para incluirlo a SQL fue a través de la herramienta **Import Wizard** presente en *Workbench*, una opción bastante intuitiva que facilita darle formato a cada variable en la tabla.

1. Anterior al uso de la herramienta, fue necesario crear un DATABASE donde sería almacenada la importación. Aunque opcional, consideré importante incluir un comando que verificara —en caso de existir— las bases de datos definidas con el mismo nombre antes de proceder a crearla.

```
CREATE DATABASE IF NOT EXISTS crimen_mexico;
```

2. A **Import Wizard** se accede dando clic derecho en la sección de *Tables* del esquema deseado.

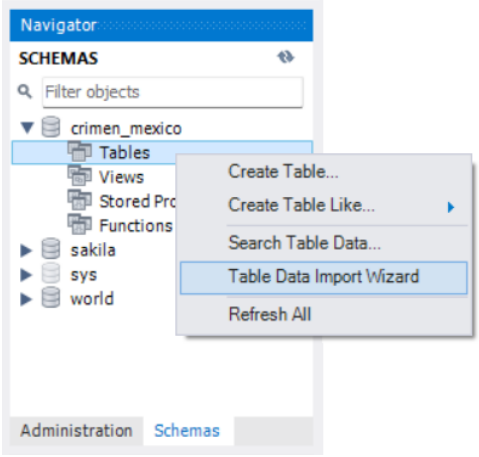

Figura 1. Captura de MySQL Workbench.

3. Después de seleccionar la ubicación de mi archivo, configuré el destino de la tabla creando una nueva de nombre **raw_delitos**, con la opción de eliminar alguna existente con ese nombre.

III. Exploración de la base de datos

Para conocer mejor la tabla, comencé con una consulta de la cantidad de registros con las siguientes líneas de código:

```
USE crimen_mexico;
SELECT COUNT(*) AS NUM_REGISTROS
FROM raw_delitos;
```

A continuación, accedí a la tabla **INFORMATION_SCHEMA** que provee la metadata de nuestros esquemas, esto para obtener información como el nombre o tipo de variable de cada columna.

```
SELECT COLUMN_NAME, DATA_TYPE, COLUMN_KEY,CHARACTER_MAXIMUM_LENGTH,
```

```
       NUMERIC_PRECISION
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'raw_delitos';
```

Finalmente, un ejemplo con la columna «Tipo de delito», de cómo aplicar el comando **DISTINCT** para obtener los valores únicos.

```
SELECT DISTINCT `Tipo de delito`
FROM raw_delitos;
```

IV. Modelo entidad-relación

Para tener una mejor organización y estructura de los datos, diseñé con la herramienta **MySQL Model** un modelo entidad-relación con tres tablas:

- **delitos_2015**: tabla principal que contiene los registros de cada crimen realizado y su conteo por mes correspondiente al 2015.
- **delitos**: se relaciona a **delitos_2015** por la columna ID_DELITO e incluye la clasificación del crimen y el bien jurídico que este afecta.
- **locaciones**: contiene el nombre del municipio y la entidad mexicana a la que pertenece.

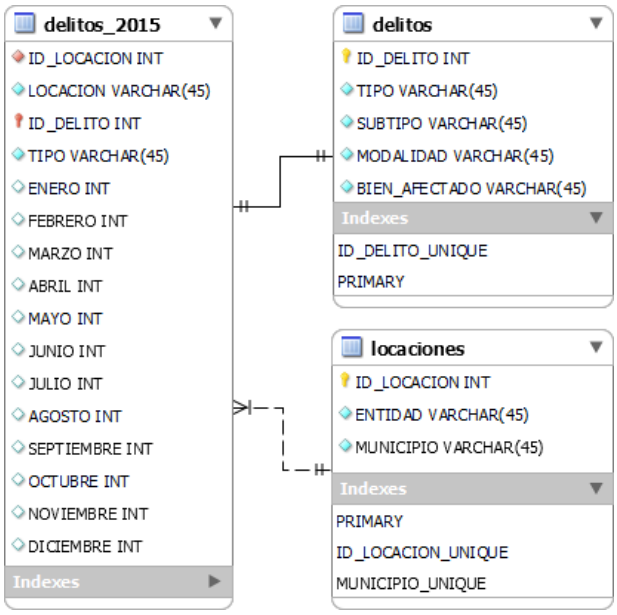

Figura 2. Diagrama entidad-relación generado en MySQL Workbench

V. Normalización de tablas

Para organizar la base de datos conforme al modelo propuesto lo primero que realicé fue renombrar las columnas a un formato más manejable por el lenguaje:

```
ALTER TABLE raw_delitos
RENAME COLUMN Entidad TO ENTIDAD,
RENAME COLUMN `Cve. Municipio` TO ID_LOCACION,
RENAME COLUMN Municipio TO MUNICIPIO,
RENAME COLUMN `Bien jurídico afectado` TO BIEN_AFECTADO,
RENAME COLUMN `Tipo de delito` TO TIPO,
RENAME COLUMN `Subtipo de delito` TO SUBTIPO,
RENAME COLUMN Modalidad TO MODALIDAD,
RENAME COLUMN Enero TO ENERO,
RENAME COLUMN Febrero TO FEBRERO,
RENAME COLUMN Marzo TO MARZO,
RENAME COLUMN Abril TO ABRIL,
RENAME COLUMN Mayo TO MAYO,
RENAME COLUMN Junio TO JUNIO,
RENAME COLUMN Julio TO JULIO,
RENAME COLUMN Agosto TO AGOSTO,
RENAME COLUMN Septiembre TO SEPTIEMBRE,
RENAME COLUMN Octubre TO OCTUBRE,
RENAME COLUMN Noviembre TO NOVIEMBRE,
RENAME COLUMN Diciembre TO DICIEMBRE;
```

Posteriormente añadí una columna con valores únicos que sirvieran de *ID* para cada registro, a la cual le asigna la característica **UNIQUE** y **AUTO_INCREMENT** para asegurar que no hubiese valores vacíos o repetidos.

```
ALTER TABLE raw_delitos
ADD ID_DELITO INT NOT NULL UNIQUE AUTO_INCREMENT;
```

Teniendo el formato deseado en la tabla de partida, cree cada una de las tablas con base a lo propuesto en la figura 2.

```
-- Creación de la tabla delitos:
CREATE TABLE delitos AS
SELECT ID_DELITO, TIPO, SUBTIPO, MODALIDAD, BIEN_AFECTADO
FROM raw_delitos
ORDER BY ID_DELITO;
-- Configuración de ID_DELITO como PRIMARY KEY:
ALTER TABLE delitos
```

```
ADD PRIMARY KEY (ID_DELITO)

-- Creación de la tabla locaciones:
CREATE TABLE locaciones AS
SELECT DISTINCT ID_LOCACION, ENTIDAD, MUNICIPIO
FROM raw_delitos
ORDER BY ENTIDAD;
-- Configuración de ID_LOCACION como PRIMARY KEY:
ALTER TABLE locaciones
ADD PRIMARY KEY (ID_LOCACION);

-- Creación de la tabla delitos_2015:
CREATE TABLE delitos_2015 AS
SELECT ID_LOCACION, MUNICIPIO, ID_DELITO, TIPO,
    ENERO, FEBRERO, MARZO, ABRIL, MAYO, JUNIO, JULIO,
    AGOSTO, SEPTIEMBRE, OCTUBRE, NOVIEMBRE, DICIEMBRE
FROM raw_delitos;
-- Configuración de FOREIGN KEYs, ID_DELITO como PRIMARY KEY:
ALTER TABLE delitos_2015
RENAME COLUMN MUNICIPIO TO LOCACION,
ADD PRIMARY KEY (ID_DELITO),
ADD FOREIGN KEY (ID_DELITO) REFERENCES delitos(ID_DELITO),
ADD FOREIGN KEY (ID_LOCACION) REFERENCES locaciones(ID_LOCACION);
```

VI. Creación de funciones

Usar funciones almacenadas representa una ventaja al manejar bases de datos donde se quiere reusabilidad de procesos y cálculos. Un ejemplo muy sencillo aplicado al esquema de **crimen_mexico** es poder obtener el subtipo almacenado en la tabla **delitos** a través de una consulta realizada desde la tabla principal —la cual no cuenta con esta información—, para ello se almacena una función que recibe el *ID* del registro y regresa el subtipo de crimen correspondiente.

```
DELIMITER //
DROP FUNCTION IF EXISTS DELITO_SUBTIPO //
CREATE FUNCTION DELITO_SUBTIPO(id_del INT)
RETURNS VARCHAR(45)
READS SQL DATA
BEGIN
    DECLARE subtipo_del VARCHAR(45);
```

```
    SET subtipo_del = (SELECT SUBTIPO
    FROM delitos WHERE ID_DELITO=id_del);
    RETURN subtipo_del;
END; //
DELIMITER ;

SELECT LOCACION, ID_DELITO, DELITO_SUBTIPO(4500)
FROM delitos_2015
WHERE ID_DELITO=4500;
```

Otro ejemplo es el caso de querer obtener el conteo total de meses, por tipo de crimen y municipio, por lo que se define una función con dos variables de entrada que permite obtener dicha suma agrupada por el tipo de delito sin distinción de cada registro.

```
DELIMITER //
DROP FUNCTION IF EXISTS DELITOS_TOTAL //
CREATE FUNCTION DELITOS_TOTAL(municipio VARCHAR(45), crimen TEXT)
RETURNS INT
READS SQL DATA
BEGIN
    DECLARE total INT;
    SELECT ENERO+FEBRERO+MARZO+ABRIL+MAYO+JUNIO+JULIO
           +AGOSTO+SEPTIEMBRE+OCTUBRE+NOVIEMBRE+DICIEMBRE
    INTO total
    FROM delitos_2015
    WHERE LOCACION=municipio AND TIPO=crimen
    GROUP BY TIPO;
    RETURN total;
END; //
DELIMITER ;

SELECT LOCACION, TIPO, DELITOS_TOTAL(LOCACION, TIPO) AS TOTAL
FROM delitos_2015
WHERE LOCACION='Hermosillo'
GROUP BY TIPO;
```

VII. Creación de vistas

Como científico de datos es importante saber generar vistas con el propósito de explotar la principal ventaja de estas: el hecho de no exponer los datos reales. Entre

usuarios, accesos y manipulación de los datos, puede resultar comprometida la integridad de la base, por lo que una vista puede facilitar compartir consultas de forma segura.

Para poder usar libremente una porción de **crimen_mexico**, creé una vista combinando información de todas sus tablas pero sólo para aquellos registros que hubiesen ocurrido en el estado de Sonora. Esto no sólo mantiene las tablas originales intactas, sino que agiliza la lectura de los datos excluyendo aquellos que no son de interés a mi aplicación.

```
CREATE VIEW SONORA_DELITOS AS
SELECT delitos_2015.ID_LOCACION, locaciones.ENTIDAD,
     locaciones.MUNICIPIO, delitos_2015.ID_DELITO, delitos.TIPO,
    delitos.SUBTIPO, delitos.MODALIDAD,
    ENERO+FEBRERO+MARZO+ABRIL+MAYO+JUNIO+JULIO+AGOSTO+
    SEPTIEMBRE+OCTUBRE+NOVIEMBRE+DICIEMBRE AS TOTAL
FROM delitos_2015
LEFT JOIN locaciones
ON delitos_2015.ID_LOCACION=locaciones.ID_LOCACION
LEFT JOIN delitos
ON delitos_2015.ID_DELITO=delitos.ID_DELITO
WHERE ENTIDAD='Sonora';
```

VIII. Consulta con Python

Existen muchos recursos en Python que pueden ser usados para conectarse a un servidor de MySQL, yo utilicé la librería `pymysql` porque simplifica el código para realizar dicha conexión, como mi base de datos se encontraba almacenada en un servidor local, empleé el siguiente código:

```
import pandas as pd
import pymysql.cursors
connection = pymysql.connect(host='localhost',
                             user='root',
                             password=<PASSWORD>,
                             database='crimen_mexico',
                             cursorclass=pymysql.cursors.DictCursor)
query='SELECT * FROM sonora_delitos'
son_df=pd.read_sql(query, connection, index_col='ID_DELITO')
```

Como se puede apreciar en el código anterior, se utiliza el propio lenguaje SQL a modo de consulta para obtener la tabla. Con esta se puede realizar cualquier tipo de manipulación, o edición, y no se afectará lo contenido en el servidor.

Finalmente, generé un mapa interactivo en el que se puede visualizar la incidencia de homicidios por municipio de Sonora, y cada polígono despliega información correspondiente. El código completo de su elaboración puede ser encontrado en este repositorio [3].

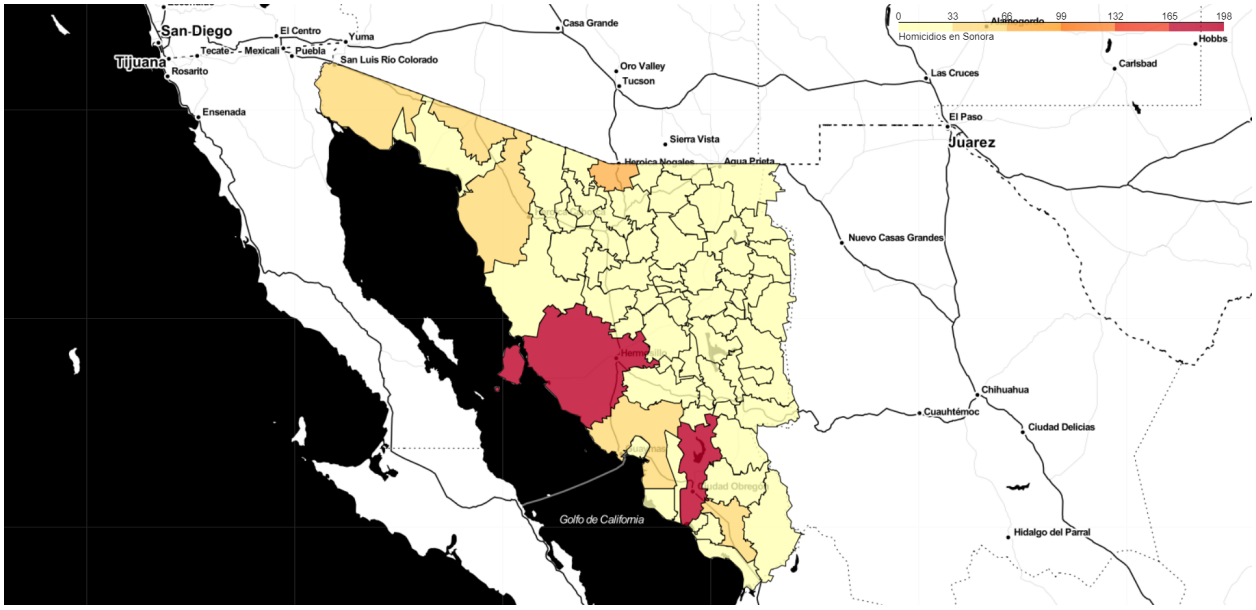

Figura 3. Mapa generado en Python a partir de la consulta a MySQL.

IX. Referencias

[1] What is SQL & Why is it important to learn it? (2022) CodeOp. Disponible en: https://codeop.tech/what-is-sql/ (Consultado: 24 de noviembre de 2022).

[2] Secretariado Ejecutivo del Sistema Nacional de Seguridad Pública (2022) Reportes de incidencia delictiva al mes de octubre 2022 (Nueva metodología). Disponible en: https://www.gob.mx/sesnsp/acciones-y-programas/datos-abiertos-de-incidencia-delictiva?state=published (Consultado: 23 de noviembre de 2022).

[3] Instituto Nacional de Estadística y Geografía (sin fecha) Descarga masiva. Disponible en: https://www.inegi.org.mx/app/descarga/ (Consultado: 24 de noviembre de 2022).

[4] MCD_ICD_SQL (2022) GitHub. Disponible en: https://github.com/roglz/MCD_ICD_SQL (Consultado: 1 de diciembre de 2022).