CALIFORNIA TRAFFIC COLLISION DATA FROM SWITRS

The Statewide Integrated Traffic Records System (SWITRS) is a database that serves as a means to collect and process data gathered from a collision scene. This system is used and updated by the CHP (California Highway Patrol) all over the state of California (USA).

Full SWITRS database dumps were requested to the CHP and posted on a Kaggle competition in December 2020, containing all the data collected by the CHP between the years 2001 and 2020.

After downloading the .SQLite database file from Kaggle, we started our Data Preparation process.

First, we inspected the database tables and columns of the .SQLite file, hoping to find some indications on how to build the relational schemas:

```
sqlite3
sqlite> .open switrs.sqlite
sqlite> PRAGMA table_info(table);
```

The code above shows the following kind of information, for any given table of the opened database:

```
cid      name          type     notnull  dflt_value  pk
-------  ------------  -------  -------  ----------  -------
0        case_id       text     1                    0
1        party_number  text     1                    0
```

In our case, no properties had been assigned to the columns, as shown in the output below:

```
0|case_id|TEXT|0||0
1|jurisdiction|INT|0||0
2|officer_id|TEXT|0||0
3|reporting_district|TEXT|0||0
4|chp_shift|TEXT|0||0
5|population|TEXT|0||0
6|county_city_location|TEXT|0||0
7|special_condition|TEXT|0||0
8|beat_type|TEXT|0||0
9|chp_beat_type|TEXT|0||0
...
```

We left this issue for later. To convert the database from SQLite to SQL, we could have chosen either between the python script file attached to this report (named “sqlite3-to-mysql.py”), or among one of the many database converter tools.

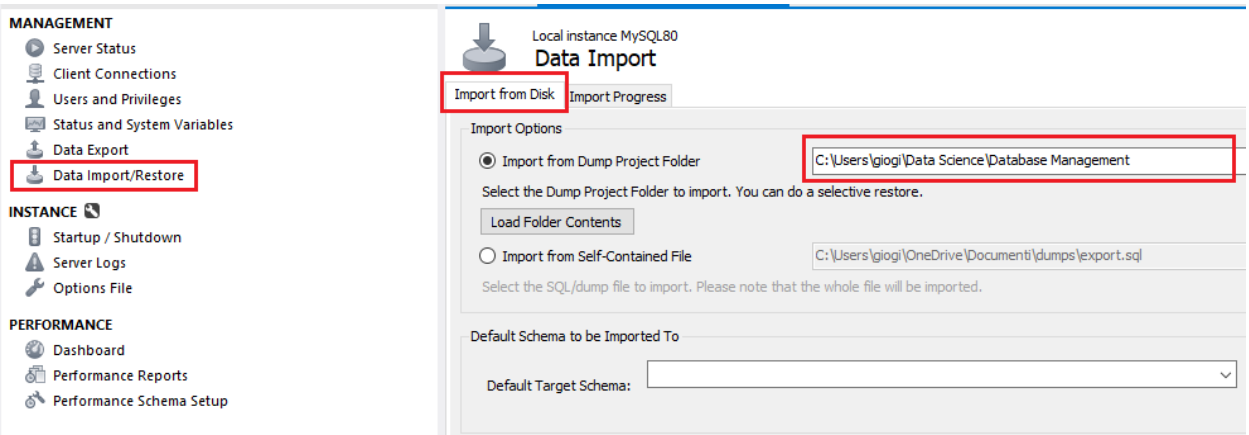
To use the python script, the following command line needs to be run from within the directory where the SQLite file is located:

```
➢  sqlite3 switrs.sqlite .dump | python sqlite3-to-mysql.py > dump.sql
```

Then, it is necessary to import the converted .sql file to MySQL using the following code:

```
➢  \sql
➢  \connect --mysql root@localhost:3306
➢  CREATE DATABASE traffic;
➢  mysql -u root@localhost:3306  --password=your_password traffic < dump.sql
```

Alternatively, it is possible to use the Data Import/Restore tool from within the MySQL Workbench Administration panel:



Before going any further, we scanned the engine behind the imported database tables:

```
➢  SHOW TABLE STATUS FROM `traffic`;
```

We found out that the engine is of ISAM type. To have a newer and better performing type of engine that is also compatible with all the features we wanted to implement to our database (such as the FOREIGN KEYS), we converted the tables to the INNOBD engine:

```
➢  ALTER TABLE table_name ENGINE=InnoDB;
```

To avoid errors when setting up the PRIMARY KEYS and the FOREIGN KEYS in the database, we also converted the “case_id” column, that is present in all the tables, from the TEXT type to the VARCHAR type.

First, we found the maximum length of characters that the case_id column can host by running the following query:

```
SELECT LENGTH(case_id)
FROM collisions
ORDER BY LENGTH(case_id) DESC
LIMIT 1;
```

Given the result, we knew that we could set VARCHAR(19) as case_id type. We could do so by running the following command:

```
ALTER TABLE table_name
CHANGE COLUMN `case_id` `case_id` VARCHAR(19) NOT NULL ;
```

All was ready to set the “case_id” column as PRIMARY KEY for the tables “Case_ids” and “Collisions”:

```
ALTER TABLE table_name ADD PRIMARY KEY (case_id);
```

The column “case_id”, however, was not unique in the tables Parties and Victims. The idea was then to use the columns “case_id” and “id” together to create a COMPOSITE PRIMARY KEY.

We then checked to see if this was feasible:

```
SELECT id, case_id
  FROM table_name
GROUP BY id, case_id
  HAVING count(*) > 1
```

We were good to go in both tables! We then created the COMPOSITE PRIMARY KEYS for both Parties and Victims:

```
ALTER TABLE table_name ADD PRIMARY KEY (id, case_id);
```

At this point we were only left with the creation of the appropriate FOREIGN KEYS. Since we were dealing with a huge database, with tables that weigh even more than 3 GBs, we needed to increase the size of our INNODB buffer pool size.

To find the optimal size to be given to our buffer pool, we ran the following command:

```
SELECT CEILING(Total_InnoDB_Bytes*1.3/POWER(1024,3)) RIBPS FROM
(SELECT SUM(data_length+index_length) Total_InnoDB_Bytes
FROM information_schema.tables WHERE engine='InnoDB') A;
```

Then, we logged-in with our Windows Administrator account, we accessed the MySQL **my.ini** file located at C:\ProgramData\MySQL\MySQL Server 8.0\my.ini, and set the option:

```
innodb_buffer_pool_size = 12G
```

That is, 12 Gigabytes. Next, we set the FOREIGN KEYS in the “Case_ids”, “Parties” and “Victims” tables:

```
ALTER TABLE table_name ADD FOREIGN KEY(case_id) REFERENCES collisions(case_id);
```

We were then ready to generate the ER diagram by using the “Reverse Engineer” function of MySQL:

As we can see, the “Collisions” table acts as an intermediate table between “Parties” and “Victims”. In fact, we have a one-to-many relationship between Collisions and Parties, and a one-to-many relationship between Collisions and Victims. For further information, you can refer to the attached file “traffic_diagram.mwb”.

We concluded the Data Preparation part of this project by converting the date columns “collision_date”, “collision_time” and “process_date” from the TEXT type to the DATE type. This will improve the performance of some of the queries we intend to run against the database.

```
UPDATE collisions
SET
    collision_date = STR_TO_DATE(`collision_date`, '%Y-%m-%d');
```

We can now start with the implementation of the database queries.