Co-funded by the
Erasmus+ Programme
of the European Union

SQL Detective Game Manual

Author: Angel Marchev, Jr.

Contents

Instructions for Teachers

Overview

The SQL Detective Game is an excellent tool for motivating students to exercise their SQL skills. As the game is suitable for online and offline teaching it also includes elements for effective teamwork.

Objectives

Main objective of the game is for students to practice their SQL skills. Secondary objective is for them to get used using particular online tools. As a general universal skill is working together on a case to obtain and share knowledge.

Group Size

Each student team should consist of up to 3 students. There are no general limitations to how many student teams there could be, apart from time for presenting their solution to the others and time for grading their solutions.

Time requirements

This is a game which could be given as an out-of-class or in-class activity. The typical successful solution by a team takes about several hours in which time they might need some consultation time. Initial instructions and team appointment takes about 30 minutes. Final presentation takes about 5-7 minutes per team. Grading takes about 20-30 minutes per team. Final debrief takes about 30 minutes.

Set Up

For a successful participation each student has to have access to a personal computer (not mobile device) and internet connection. They also need to have received the file

sql-murder-mystery-no-sol.db which holds the database which stores the records for the game.

Procedure

1. Distribute the students in teams of up to 3.
2. Supply the teams with the student instructions and the file *sql-murder-mystery-no-sol.db*.
3. Brief explanation of the game.
4. Give them about 3-4 hours to make their solution, while being available for their questions, if needed.
5. After they deliver, make each team present their solution in front of the others. You may credit additional points for presentation skills.
6. Grade and review their solutions (it would take some time to do so).
7. Debrief

Directions

For the best results make them work with less known partners and try to distribute uniformly the students with better SQL skills.

Explain to them only a little bit about the activities of the game as a teaser. Everything they need to know is already written in the instructions. Give instructions of how and when to deliver and ways to communicate with you in the meantime for consultations. You may decide to give them the Technical assistance pack or leave it up to them to figure it out. Also, you may make them use the automatic tool for the Entity Relationship Diagram (so learn this tool) or leave them extract table headers and make a diagram on their own (more realistic case scenario). In the latter case you should modify step 6 of the student instructions.

Debrief

Make the debrief explaining not so much the case (as it is most probably solved correctly by everyone) as on their approach and their SQL code. There is not one single approach to find the solution and additionally there is several ways to code the SQL for each approach.

Variations

There are two conditions that may vary the game – if you have a local SQL server setup and if it is distance learning game. So, there would be three variants of the game:

- Local server, in-class game
- Internet IDE, in-class game
- Internet IDE, distance learning game.

In you decide to use the internet IDE variant, the students should use some online SQL IDE, the most preferable one is https://sqliteonline.com/.
Alternatively, a local SQL server could be setup for the class, and in that case step 4 of the student instructions should be altered.
If the game is played online, one would need establish a way for the teams to communicate with the teacher.
The in-class game tends to be quicker, as the students often consult among various teams.

Instructions for Students

1) Who are you?

After graduation you just joined the **FBI Data Analytics Division** in a team of **up to three** colleagues. So together with your teammates you are about to receive your first task. Before that you need to introduce yourselves to each other within the team. Also start a report .docx file named fak_nom1_fak_nom2_fak_nom3_SQL_murder.docx and write down **three common things for everyone in the team**. Based on that create your **team logo**, and **team motto** and add them to the report file.

2) Team issues

Next set of things you have to decide on is **general team issues** such as:

- what are going to be your communicational channels?
- who is in charge of writing down the important discoveries?
- who would present at the end in front of the audience?
- etc.

3) Setup

In order to **setup your database environment**, you should then upload the attached file *sql-murder-mystery-no-sol.db* in the IDE (for example https://sqliteonline.com/).

4) The Case

Next comes **the case**. The FBI wanted your team to help them solve a crime by reading through the crime scene report, but the report got scrambled beyond reading during your party for the new job. Still being smart ones as you are, you partly remember that it was a murder that occurred sometime on **Jan.15, 2018**. Also, you remember it took place in SQL City.

5) Queries

So, you should start by retrieving the corresponding **crime scene report** from the police department's database. and continue doing **SQL queries** to solve the crime itself.

6) Relationships

Working with the SQL database you should first construct its graphical scheme representation (also known as **entity relationship diagram**) which shows the relations within the database tables. You may use https://dbdiagram.io/ or any graphical tools to do that. Once you have done it, add it to the report file.

7) Main task

Main task is to correctly find the **name of the killer** through properly worded SQL queries. **TRIPPLE CHECK** your solution through various logic ways and various queries - after all you do not want to accuse an innocent person but also you do not want to leave a murderer free on the streets. Write down the name it in the report file.

8) Secondary task

Secondary task is to find correctly who the real villain is through properly worded SQL queries. **TRIPPLE CHECK** your solution through various logic ways and various queries - after all you do not want to accuse an innocent person but also you do not want to leave a **criminal-mastermind** free on the streets. Write down the name it in the report file.

9) The Story

Next write in the report with regular words what is **the full story** of the crime - exactly what happened, how it happened, and why it happened.

10) The Code

Copy your **SQL queries as code** that led to the successful result to the report file in sequential **execution order**.

11) The Report

Take some time to **format your report file aesthetically** so that it is most informative and at the same time most readable. If you wish you **may rearrange** it in a way that makes most sense to you to improve its understanding.

12) The Delivery

Before the deadline go to the link deliver your report to the given link.

Technical Assistance

SQL

SQL is **Structured Query Language**, which is a computer language for storing, manipulating and retrieving data stored in a relational database. **SQL** is the standard language for Relational Database System. For more detailed documentation on SQLite, you may refer to https://www.sqlitetutorial.net/ .

SQL query structure:

- SELECT statement is used to extract data
- Clauses: JOIN, WHERE, ORDER BY, LIMIT, GROUP BY, HAVING, etc.
- Good Practice is each clause starts from new line

```
SELECT DISTINCT column_list
FROM table_list
  JOIN table ON join_condition
WHERE row_filter
ORDER BY column
LIMIT count OFFSET offset
GROUP BY column
HAVING group_filter;
```

SELECT Statement:

Different columns should be separated by comma! The SQL Keywords are case-insensitive (SELECT, FROM, WHERE, etc.), but are often written in all caps as a good practice. Usually, the column and table name are also case-insensitive

*SELECT * FROM table*
SELECT column name FROM table
SELECT Column1, Column2, Column3 FROM table
Comments (-- , /* ….. */)

ORDER BY clause

ORDER BY clause is sorting the result based on one or more columns in different order. Column name by which you want to sort after the ORDER BY clause followed by the ASC or DESC keyword. The ASC keyword means ascending. And the DESC keyword means descending. If not specified ASC order is by default. You can sort the result set using a column that does not appear in the select list of the SELECT clause. Use a comma (,) to separate multiple order columns

```
SELECT
   select_list
FROM
   table
ORDER BY
    column_1 ASC,
    column_2 DESC;
```

LIMIT clause

The LIMIT clause is an optional part of the SELECT statement. You use the LIMIT clause to constrain the number of rows returned by the query. We can retrieve 10 rows instead of 1 MLN rows. The **row_count** is a positive integer that specifies the number of rows returned (5, 10, 20, 1000, etc.)

```
SELECT
        column_list
FROM
        table
LIMIT row_count;
```

WHERE clause

The WHERE clause is an optional clause of the SELECT statement. It appears after the FROM clause as the following statement. When evaluating a SELECT statement with a WHERE clause, SQLite uses the following steps:

- First, check the table in the FROM clause.
- Second, evaluate the conditions in the WHERE clause to get the rows that met these conditions.
- Third, make the final result set based on the rows in the previous step with columns in the SELECT clause.

```
SELECT
        column_list
FROM
        table
WHERE
        search_condition;
```

For example, you can form a search condition as follows:

- WHERE column_1 **=** 100;
- WHERE column_2 **IN** (1,2,3); **IN** (‘Canada’, ‘UK’); **NOT IN** (‘Canada’, ‘Bulgaria’)
- WHERE column_3 **LIKE** 'An%';
- ‘%text%’ (will look for ‘text’ anywhere in the text); **NOT LIKE**
- WHERE column_4 **BETWEEN** 10 AND 20;
- WHERE column_5 **IS NULL**; **NOT NULL**
- WHERE column_6 <> “”
- Use **AND** or **OR** operator for multiple conditions

SQL Comparison Operators

A comparison operator tests if two expressions are the same. The following table illustrates the comparison operators that you can use to construct expressions:

Operator	Meaning
=	Equal to
<> or !=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

DISTINCT clause

The DISTINCT clause allows you to remove the duplicate rows in the result set.

SELECT DISTINCT (*)

SELECT DISTINCT(column1)

SELECT COUNT (DISTINCT column1)

SELECT COUNT (DISTINCT *)

```
SELECT DISTINCT select_list
FROM table;
```

CASE clause

CASE expression evaluates a list of conditions and returns an expression based on the result of the evaluation. The CASE expression is similar to the IF-THEN-ELSE statement in other programming languages.

```
CASE case_expression
     WHEN when_expression_1 THEN result_1
     WHEN when_expression_2 THEN result_2
     ...
     [ ELSE result_else ]
END
```

GROUP BY clause

The GROUP BY clause a selected group of rows into summary rows by values of one or more columns. For each group, you can apply an aggregate function such as MIN, MAX, SUM, COUNT, or AVG to provide more information about each group.

```
SELECT
    column_1,
    aggregate_function(column_2)
FROM
    table
GROUP BY
    column_1,
    column_2;
```

Working with dates

The date and time functions use a subset of IS0-8601 date and time formats. The date() function returns the date in this format: YYYY-MM-DD. The time() function returns the time as HH:MM:SS. The datetime() function returns "YYYY-MM-DD HH:MM:SS". The julianday() function returns the Julian day - the number of days since noon in Greenwich on November 24, 4714 B.C. (Proleptic Gregorian calendar). The **strftime()** routine returns the date formatted according to the format string specified as the first argument

HAVING clause

The HAVING clause specifies a search condition for a group. You often use the HAVING clause with the GROUP BY clause. The GROUP BY clause groups a set of rows into a set of summary rows or groups. Then the HAVING clause filters groups based on a specified condition. If you use the HAVING clause, you must include the GROUP BY clause; otherwise, you will get the following error: Error: a GROUP BY clause is required before HAVING.

Note: that the HAVING clause is applied after GROUP BY clause, whereas the WHERE clause is applied before the GROUP BY clause.

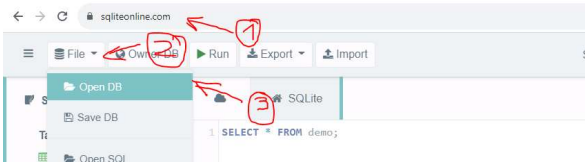
INNER JOIN

INNER JOIN or JOIN returns records that match in **BOTH** right and left tables. INNER JOIN clause matches each row from the albums table with every row from the artists table based on the join condition (artists.ArtistId = albums.ArtistId) specified after the ON keyword.

```
SELECT
    Title,
    Name
FROM
    albums
INNER JOIN artists
    ON artists.ArtistId = albums.ArtistId;
```

Loading a database to SQLite Online

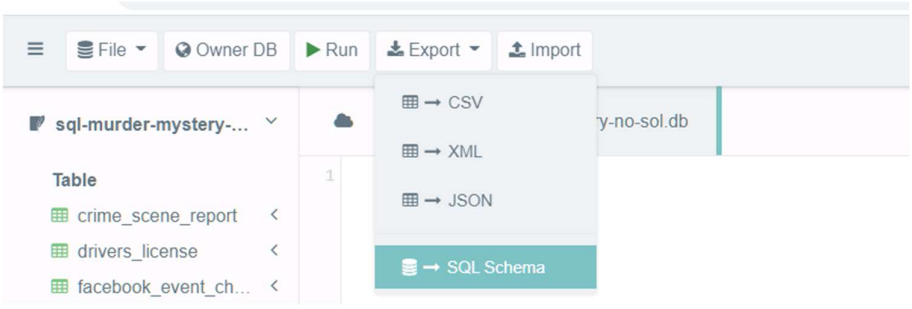
To open and work with a database, go to https://sqliteonline.com/ and open the .db file



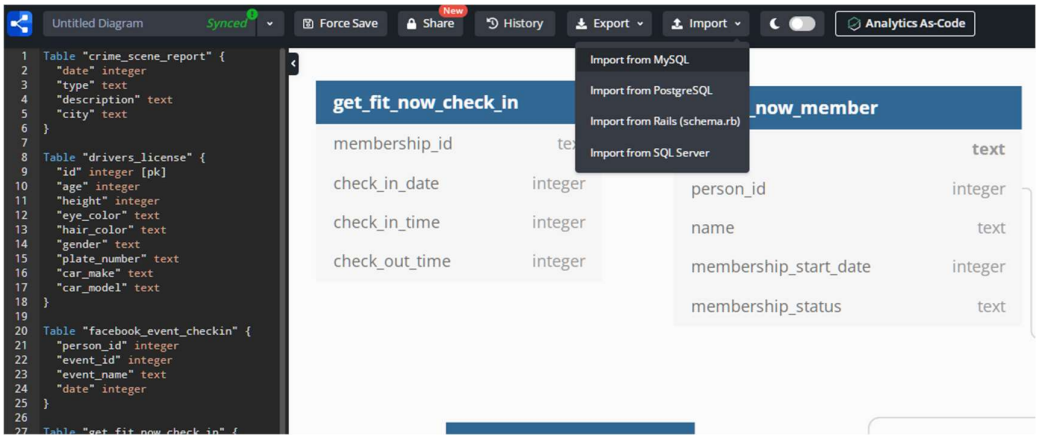
Creating DB schema diagrams

For easy creation of DB schema diagrams, you must do the following:

1) Export the schema of the database from the IDE as an SQL Schema :



2) Go to https://dbdiagram.io/ and choose import form MySQL:



3) Select the schema file and submit