GRADED LAB

CITS1402 Relational Database Management Systems – Lab 4

Released on 29 April
Due date 20 May , Monday at 1800 hrs (6pm)
How to submit: On LMS. The questions are listed on page 4.
Marks: 5% of the total grade
Use of AI: You are allowed to consult each other or get help from SQL tutorials. AI (LLMs) such as ChatGPT are not allowed.

Learning Aims
This lab is concerned with
1. Using subqueries to retrieve data, create tables, create/drop views
2. Examining the table and its entries
3. More advanced use of SELECT, WHERE and use of JOIN, NATURAL JOIN, VIEWS, CREATE, INSERT INTO, UPDATE

Case study:

We are using the same case study that was provided to you for labs 2&3.

SafePrint Solutions is a small printing company that works for book publishers (tracked in the **publishers** table). SafePrint Solutions' jobs consist of printing books or parts of books. These jobs are recorded in the **bookjobs** table. A printing job requires the use of materials, such as paper and ink, which are assigned to a job via purchase orders (PO) kept in the **pos** table. Each printing job may have several POs assigned to it. Likewise, each PO may contain several PO items which are recorded in a separate **po_items** table. The one-to-many relationship between **pos** and **po_items** is implemented by the composite foreign key (*job_id, po_id*) in the **po_items** table. The materials which appear in **po_items** are tracked in the **items** table, which records the material description, the quantity on-hand in the warehouse, and the price.

The many-to-many (*:*) relationship between **pos** and **items** is decomposed into two one-to-many (1:*) relationships by means of the intersection relation **po_items**

SafePrint Solutions Relation Structures

publishers (cust_id, name, city, phone, creditcode)

bookjobs (job_id, cust_id, job_date, descr, jobtype)

pos (job_id, po_id, po_date, vendor_id)
items (item_id, descr, on_hand, price)

po_items (job_id, po_id, item_id, quantity)

Creating the database

You are provided with 3 files, that will help you load the database.

There are multiple ways to load a database in your sqlite. Let's look at those ways.

1. Loading a .db file
   You need to open a windows or a mac terminal. Now navigate to the folder where you have downloaded the files that we have provided. You can do that using the cd command. In order to display the contents of the folder please type dir (on windows) or ls (on mac). Once you are in the right folder on the terminal please type sqlite3. Then load the database using the **.open** command as shown below.

```
mehwishnasim@Mehwishs-Mac-Studio Downloads % sqlite3
SQLite version 3.37.0 2021-12-09 01:34:53
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .database
main: "" r/w
sqlite> .open printing.db
sqlite> .database
main: /Users/mehwishnasim/Downloads/printing.db r/w
sqlite>
```

2. You can also load the database when you are opening your sqlite3.
   First you need to quit sqlite.
   Now write **sqlite3 printing.db**
   It will load the database.

```
mehwishnasim@Mehwishs-Mac-Studio Downloads % sqlite3 printing.db
SQLite version 3.37.0 2021-12-09 01:34:53
Enter ".help" for usage hints.
sqlite> .tables
bookjobs    items       po_items    pos         publishers
```

3. There is also a third way. It is useful in situations where you have played around with the database and you want to load a fresh copy without quitting sqlite. Also, at times .db file is not available.

- You can construct the database through files that have a ".sql" or a ".txt" extension.
- The files that are provided to you are:
  - load-tables-sqlite.sql : this file will create tables. If you open this using a textpad (right-click and open with textpad or notepad), you will see multiple drop table statements. Those statements will drop any existing tables that have the same name. If you load this file in a fresh instance of sqlite, it will give you 5 errors. Why? Because the tables that you are trying to delete (DROP) do not exist. Ignore those errors.
  - Load it using the command **.read load-tables-sqlite.sql**

```
Error: near line 1: in prepare, no such table: po_items (1)
Error: near line 2: in prepare, no such table: items (1)
Error: near line 3: in prepare, no such table: pos (1)
Error: near line 4: in prepare, no such table: bookjobs (1)
Error: near line 5: in prepare, no such table: publishers (1)
sqlite> .tables
```

- Next load the data using **.read load-data.sql**
- Now type the **.schema** command to see your relations and attributes.
- To retrieve data from a table type SELECT * FROM *tablename*;

Once you have the tables and data loaded, you can attempt the lab. Please note you can also save the database through the following command:
.save mydatabase.db (or any other name of your liking)

How should I submit my lab

For every question you need to create a .txt file. For example for Q1, create a file called Q1.txt. The file should only contain the relevant query and nothing else. For instance, see the following example:

How am I going to create a .txt file.
Windows:
You can create a .txt file on notepad. Make sure not to save it as .rtf or a .doc or a .pdf file. Also, a file format that is like "Q1.txt.rtf" is wrong. The reason is the files that have an extension other than .txt may introduce characters (invisible to you). For example, for a newline the character is "\n". You may not see it, but it is there. If the auto-grader sees those characters, it will give you zero points as it will not be able to run the query.

Mac/linux
Go to command prompt and type *vi query1.txt* (or Q1.txt etc.). Type your query. When finished, press the Esc key on the keyboard and then write :wq!

Verify as follows:

```
sqlite> .read query1.txt
P9|9KG PAPER|300|25.25
P12|12KG PAPER|700|49.99
P18|18KG PAPER|100|100
IRN|INK-RESIN|3|500
IWS|INK-WRSOL|5|350
CBD|CARDBOARD|47|15
```

You can verify all the answers, using the .read command one by one (for example ***.read Q2.txt*** or ***.read Q5.txt***)

Put all the queries in separate files: Q1.txt, Q2.txt … and so on, in a folder that is called "studentID_CITS1402_Lab2". Replace the student ID with your ID. For example, if your student ID is 123456, the folder name should look like: 123456_CITS1402_Lab4.

Zip the folder and submit it on LMS.

Your submission should run on our systems. If the file is corrupt or the naming structure is different, or if the file consists of irrelevant code or comments, the autograder will give you zero points.

Write SQL queries for the following questions.

1. Write an SQL query that retrieves a list of publishers along with any book jobs they may have. *the query should not use the keyword JOIN*. The result should include the publisher's customer ID, the publisher's name, each job's ID, and the job's description. *Hint: Join the publishers table with the bookjobs table.* Sorts the results by the publisher's customer ID.

```
cust_id|name|job_id|descr
A01|ART BOOKS|004|PAMPHLETS
A01|ART BOOKS|005|GOVT
D04|DIABLO CO|006|CAMPAIGN
E05|EASYPRINT|001|TEXT BOOKS
E05|EASYPRINT|002|BUS REPORT
E05|EASYPRINT|003|COMMERCIAL
```

2. Write an SQL *query using JOIN keyword* to retrieve a list of publishers and their associated book jobs. Your result should include the publisher's customer ID, name, each job's ID, and the job description. Ensure the results are ordered by the publisher's customer ID. *Hint: The tables publishers and bookjobs should be joined.*

```
cust_id|name|job_id|descr
A01|ART BOOKS|004|PAMPHLETS
A01|ART BOOKS|005|GOVT
D04|DIABLO CO|006|CAMPAIGN
E05|EASYPRINT|001|TEXT BOOKS
E05|EASYPRINT|002|BUS REPORT
E05|EASYPRINT|003|COMMERCIAL
sqlite>
```

3. Write an SQL query (*without using JOIN keyword*) to retrieve a detailed list of book jobs (job ID and description) along with their corresponding purchase order IDs and item ID and description. Your results should include the job ID, job description from the **bookjobs** table, purchase order ID from the **pos table**, item ID, and **item description** from the **items table**. Ensure that you join the bookjobs, pos, po_items, and items tables appropriately based on their relationships.
   Also, order by bookjobs ID, followed by po_id, and item id.

```
job_id|descr|po_id|item_id|descr
002|BUS REPORT|AAA|CBD|CARDBOARD
002|BUS REPORT|AAA|IWS|INK-WRSOL
002|BUS REPORT|AAA|P9|9KG PAPER
002|BUS REPORT|BBB|CBD|CARDBOARD
004|PAMPHLETS|CCC|IRN|INK-RESIN
004|PAMPHLETS|CCC|P9|9KG PAPER
004|PAMPHLETS|DDD|P18|18KG PAPER
006|CAMPAIGN|GGG|IRN|INK-RESIN
```

4. For the query in Q3, use the JOIN keyword to provide a solution.

5. Write an SQL query that calculates the total cost for each purchase order based on the quantity and price of items ordered. The query should utilize a **NATURAL JOIN** between the **respective** tables. Compute the total by multiplying the quantity of each item by its price. The results should be grouped by the purchase order ID (**po_id**) and display each **po_id** along with its respective total cost labeled as 'Total Cost'.

```
po_id|Total Cost
AAA|2217.5
BBB|255
CCC|5787.5
DDD|10000
GGG|1000
```

6. Write an SQL query that identifies unique publishers involved in book jobs by retrieving their customer ID and name. *You can use any approach to solve this question*. Ensure that each publisher's name appears only once in your result, even if they are linked to multiple book jobs.

```
cust_id|name
E05|EASYPRINT
A01|ART BOOKS
D04|DIABLO CO
```

7. Write an SQLite statement that creates a view named nc_jobs with all the columns as bookjobs table. It should only include the data from table bookjobs where job type is 'N' and the corresponding publisher has creditcode of 'C'.

Please **do not provide** code for retrieving this view.

8. Write the SQL command to drop the view named **nc_jobs** from your database. Ensure that the command correctly removes the view without affecting other database objects.
9. Write the SQLite DDL statement to create a new table named big_jobs which contains the job_id, cust_id, jobtype, and tot_pos for all bookjobs which have at

least $300.00 in total pos (i.e. sum of po_items.quantity * items.price). Also load the new table using an INSERT ...SELECT with a subquery that derives the information from other tables in the database (bookjobs, pos, po_items, and items).

*Note: tot_pos = sum(quantity * price)*

10. Write the SQLite DML statement to update big_jobs table. Set the jobtype to S for all rows with a non-null tot_pos value under $2000.00.

11. Write a single SQLite DML statement to delete all rows from big_jobs table where the value in the tot_pos column exceeds $3000.00.

12. Write a single SQLite DDL statement to alter the table big_jobs. Add a column to big_jobs table named "assigned" of type CHAR(1) that has a default value of "U" (meaning unassigned).