



**An-Najah National University**

**Faculty of Engineering and Information Technology**

**Computer Engineering Department**

## **Distributed Operating System**

**Student's Name :**

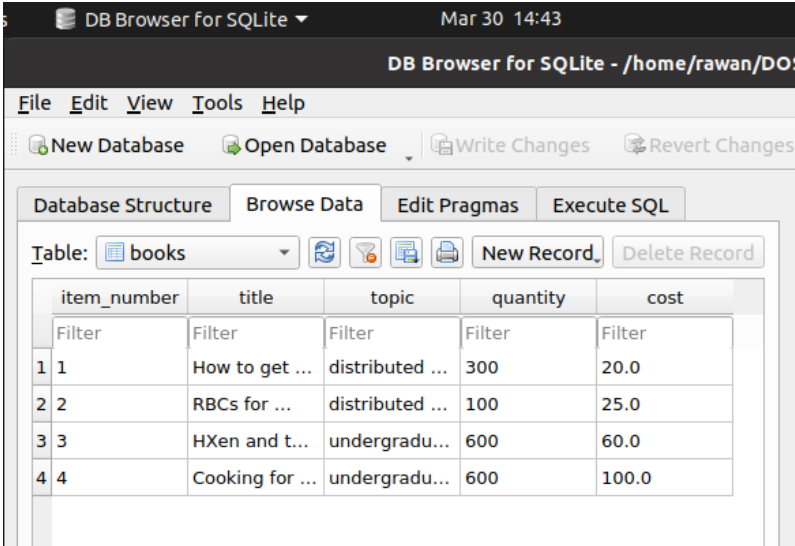
**Rawan Abo-Dyak**

**Sabah Tartir**

**Bazar.com: A Multi-tier Online Book Store Report**

The Program is a two-tier web design - a front-end and a backend - and uses microservices at each tier designed with **Flask Framework**. The front-end tier is composed of one microservice, while the back end tier consists of two servers: order and catalog servers.

In the catalog server, all the books and related information is stored in a **sqlite database**, as shown below :

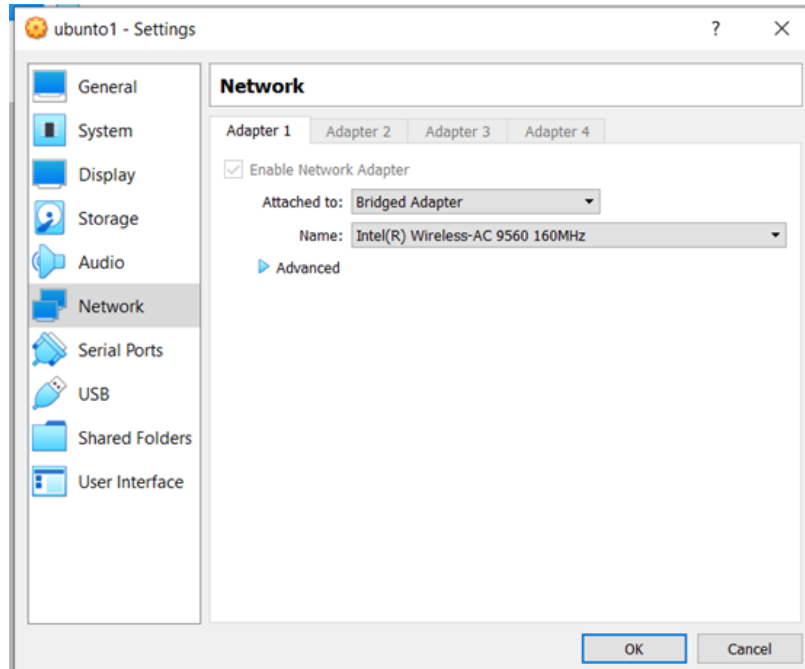


The screenshot shows the DB Browser for SQLite application. The title bar indicates the file path is /home/rawan/DOS. The interface includes a menu bar (File, Edit, View, Tools, Help) and a toolbar with buttons for New Database, Open Database, Write Changes, and Revert Changes. Below the toolbar, there are tabs for Database Structure, Browse Data, Edit Pragmas, and Execute SQL. The 'Browse Data' tab is active, showing a table named 'books'. The table has five columns: item\_number, title, topic, quantity, and cost. There are four data rows, each with an index number in the first column.

	item_number	title	topic	quantity	cost
	Filter	Filter	Filter	Filter	Filter
1	1	How to get ...	distributed ...	300	20.0
2	2	RBCs for ...	distributed ...	100	25.0
3	3	HXen and t...	undergradu...	600	60.0
4	4	Cooking for ...	undergradu...	600	100.0

Each server was on a different machine (the front server was running on Windows, the other two were running on Ubuntu), and they communicated over the network using the HTTP protocol.

We used two virtual machines and a real OS to implement the program. Afterward, we set up the communication between the machines by assigning each a static IP address (using a bridge network )



And Use Postman to send requests between the three servers to test their functionality.

All the requests start from front end server as its the point that the client interact with it , three operations are supported :

Search (topic):

URL: "<http://192.168.1.135:5000/search/topic> ,

verb :Get

Request will be sent from the front server to Catalog server and response as Json file returning item\_number and title of each book related to the specified topic .

info(item\_number)

URL :"[http://192.168.1.135:5000/info/item\\_number](http://192.168.1.135:5000/info/item_number)

Verb : Get

Return all the information about the book of specified item number ,

Request will be sent from the front server to Catalog server ,catalog server will response with Json file .

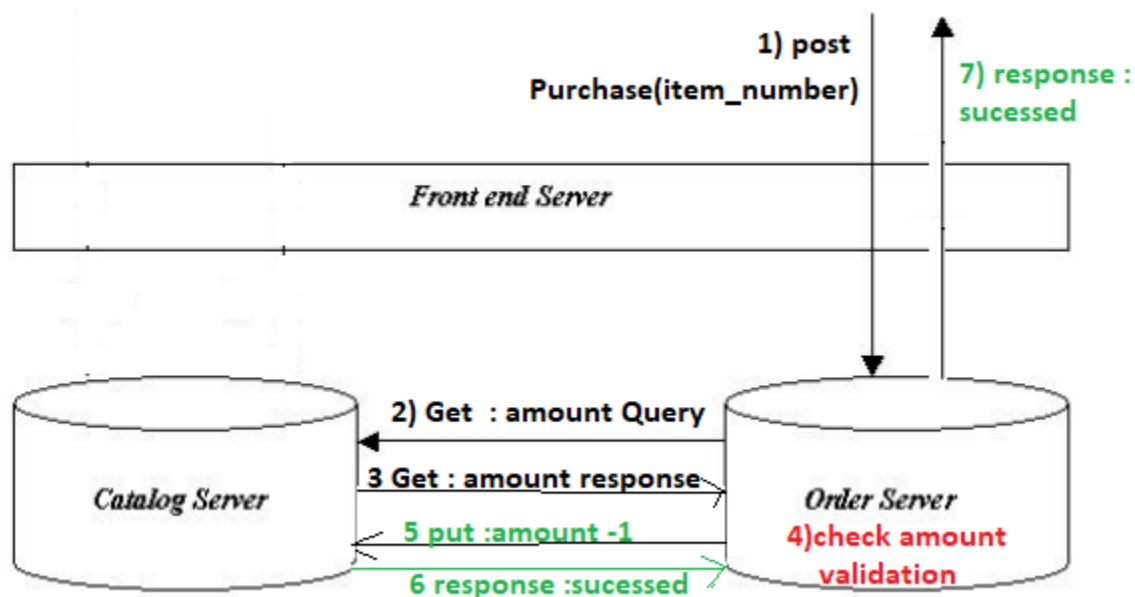
`purchase(item_number)`

URL :” [http://192.168.1.135:5000/info/item\\_number](http://192.168.1.135:5000/info/item_number)

Verb :post

Create a new purchase operation, which will result in a JSON message that describes whether the purchase succeeded or failed. Every successful purchase decreases the amount in the database by 1.

Clarification of success purchase operation :



Also, there are additional function which is updating the price of a specific book . request sent from front to catalog server to apply changes in books DB .

URL :” [http://192.168.1.135:5000/purchase/item\\_number](http://192.168.1.135:5000/purchase/item_number)

Price value sent as json in body

Verb :put

**Possible improvement :**

Presently, the purchase operation works to purchase a single item. As an improvement, the user should be able to determine the number of copies they wish to purchase. We can do this with keeping the same verbs and flow between servers but adding the number of copies in the body of the request and the order server can change the logic of validation accordingly .

**Addition :**

many operations that help users and make the app more interactive can be done Like :

Search (price) :

description : search according to specific price and return all items with this price .

Verb :Get

Need communication between Front and catalog servers

Search\_Above (price) : return all books with specific price or above

AddBook : create new book and add it to catalog server .

Verb :post

This can be done in backend tier by sending request from order server to catalog server