**Title Page**

**Unit Title:** Introduction to Databases (COM417)

**Final Assessment**

**Student Name:** *Will Bladon-Whittam*

**Date:** *28/04/2024*

**Contents page**

Contents

[PART 1 - Retrieving Data using SQL 4](#_Toc165214746)

[QUESTION A: 4](#_Toc165214747)

[SQL CODE: 4](#_Toc165214748)

[EXPLANATION: 4](#_Toc165214749)

[SCREENSHOT: 5](#_Toc165214750)

[TESTING: 5](#_Toc165214751)

[QUESTION B: 6](#_Toc165214752)

[SQL CODE: 6](#_Toc165214753)

[EXPLANATION: 7](#_Toc165214754)

[SCREENSHOT: 7](#_Toc165214755)

[TESTING: 8](#_Toc165214756)

[QUESTION C: 9](#_Toc165214757)

[SQL CODE: 9](#_Toc165214758)

[EXPLANATION: 10](#_Toc165214759)

[SCREENSHOT: 10](#_Toc165214760)

[TESTING: 11](#_Toc165214761)

[QUESTION D: 13](#_Toc165214762)

[SQL CODE: 13](#_Toc165214763)

[EXPLANATION: 13](#_Toc165214764)

[SCREENSHOT: 14](#_Toc165214765)

[TESTING: 15](#_Toc165214766)

[PART 2 – Database Design, Implementation and Integrity 17](#_Toc165214767)

[QUESTION A: 17](#_Toc165214768)

[Conceptual Design: 17](#_Toc165214769)

[Logical Design: 18](#_Toc165214770)

[Physical Design: 19](#_Toc165214771)

[QUESTION B: 20](#_Toc165214772)

[Parana Entity Relationship Diagram: 20](#_Toc165214773)

[QUESTION C: 21](#_Toc165214774)

[Create Tables: 21](#_Toc165214775)

[Insert Data: 22](#_Toc165214776)

[QUESTION D: 23](#_Toc165214777)

[Testing: 23](#_Toc165214778)

[PART 3 – Programming for Databases 29](#_Toc165214779)

[Code Overview: 29](#_Toc165214780)

[Requirements Description / Testing: 31](#_Toc165214781)

[Full Python Code: 55](#_Toc165214782)

[Main.py: 55](#_Toc165214783)

[sql.py 64](#_Toc165214784)

[Bibliography 66](#_Toc165214785)

# PART 1 - Retrieving Data using SQL

## QUESTION A:

Text

Description automatically generated

### SQL CODE:

SELECT shopper\_first\_name As [Shopper First Name],

shopper\_surname AS [Shopper Surname],

shopper\_email\_address AS [Email Address],

IFNULL(gender, "Not Known") AS [Gender],

STRFTIME("%d-%m-%Y",date\_joined) AS [Date Joined],

CAST((JULIANDAY(DATE()) - JULIANDAY(date\_of\_birth)) / 365 AS INTEGER) AS [Current Age]

FROM Shoppers

WHERE date\_joined >= "2020-01-01" OR [Gender] = "F"

ORDER BY [Gender], [Current Age] DESC

### EXPLANATION:

Used Column alias’s to display a more readable column header for all coulmns selected.

Used the IFNULL function, to update NULL values in the Gender field to display as “Not Known”

Used STRFTIME function to change the format of date joined to “dd-mm-yyyy”.

To calculate the age, I used the DATE function to get the current date, then I used the JULIANDAY function to convert to the number of days from Nov 24, 4714 BC. I then took away the JULIANDAY value of the date of birth. To remove the decimal places, I used the CAST function, as apposed to the ROUND function as a person’s age has to reach the whole number, it can’t be rounded up.

I filtered out the shoppers that joined after “2020-01-01” and females using a WHERE clause.

I then ordered by Gender, then current age in Descending order.

### A screenshot of a computer Description automatically generatedSCREENSHOT:

### TESTING:

To test the age I created a new query, and added the date of birth field.

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generatedI can now manually calculate the age, and all the ages are outputted correctly.

To test the correct rows were outputted,I first checked that all the males that are present in the query, that the date that they joined is later than 2020-01-01, which they are. Next I made a new query to output all the female shoppers, and make sure all of them are present in the initial query.

This shows that there are 8 female shoppers and they are all present in the initial query.

## QUESTION B:

Text

Description automatically generated

### SQL CODE:

SELECT s.shopper\_first\_name AS [Shopper First Name],

s.shopper\_surname AS [Shopper Surname],

so.order\_id AS [Order ID],

STRFTIME("%d-%m-%Y", so.order\_date) AS [Order Date],

p.product\_description AS [Product Description],

se.seller\_name AS [Seller Name],

op.quantity AS [Quantity],

PRINTF("£%.2f",op.price) AS [Price],

ordered\_product\_status AS [Order Status]

FROM shoppers as s

INNER JOIN shopper\_orders so ON s.shopper\_id = so.shopper\_id

INNER JOIN ordered\_products op ON so.order\_id = op.order\_id

INNER JOIN product\_sellers ps ON op.product\_id = ps.product\_id

AND op.seller\_id = ps.seller\_id

INNER JOIN sellers se ON ps.seller\_id = se.seller\_id

INNER JOIN products p ON ps.product\_id = p.product\_id

WHERE s.shopper\_id=@shopper

ORDER BY so.order\_date DESC

### EXPLANATION:

Selected all the necessary columns using column and table alias’s.

When selecting the order date I used STRFTIME to format the date as “dd-mm-yyyy”

When selecting the price, I used PRINTF to output the price with a “£” symbol and to 2 decimal places.

I then used 5 INNER JOINS to join together the shoppers, shopper\_orders, product\_sellers, sellers and products tables together. When joining the product sellers table, there are compound keys used, so to join this table both of the Primary/Foreign keys need to be matched against.

Then I accepted a parameter in the WHERE clause to only output the results of the given shopper id.

I then ordered by the order date in descending order.

### SCREENSHOT:

Shopper ID: 10000

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedShopper ID: 10019

### TESTING:

First thing to test is to make sure that only one shopper’s results are displayed, and that the order id entered matches that shopper, to do this I created a new query to also select the shopper\_id.

A screenshot of a computer

Description automatically generated

Then I checked that the order dates are in descending order.

A screenshot of a computer

Description automatically generated

## QUESTION C:

Text

Description automatically generated

### SQL CODE:

SELECT se.seller\_account\_ref AS [Seller Account Ref],

se.seller\_name AS [Seller Name],

IFNULL(p.product\_code,"N/A") AS [Product Code],

IFNULL(p.product\_description,"N/A") AS [Product Description],

COUNT(op.order\_id) AS [No. of Orders],

IFNULL(SUM(op.quantity), 0) AS [Total quantity sold],

PRINTF("£%.2f", SUM(op.quantity) \* op.price) AS [Total Value of Sales]

FROM sellers se

OUTER LEFT JOIN product\_sellers ps ON se.seller\_id = ps.seller\_id

OUTER LEFT JOIN products p ON ps.product\_id = p.product\_id

OUTER LEFT JOIN ordered\_products op ON ps.product\_id = op.product\_id

AND ps.seller\_id = op.seller\_id

GROUP BY se.seller\_id, p.product\_code

ORDER BY [Total quantity sold] ASC

### EXPLANATION:

Selected all the necessary columns using column and table alias’s.

To calculate the number of orders, I used the COUNT function to calculate the total number of orders.

To calculate the total quantity sold, I used the SUM function to add up all the items in all the orders, and IFNULL if nothing had been sold to display 0.

I also used IFNULL on the product\_code and product\_description columns. I set these to display “N/A” if NULL, as this is more representative than just “0”.

To calculate the total value of the sales, I used the SUM function to calculate the total quantity sold, then multiplied that against the price. I formatted this value with the PRINTF function to add a “£” symbol to the start and to 2 decimal places.

I used 3 OUTER LEFT JOINs to join together product\_sellers, products and order\_products tables. This is because I still want the products that haven’t been ordered yet, and sellers that don’t sell any products to be displayed (I assume this as the question asks for all sellers and all products). When joining the product sellers table, there are compound keys used, so to join this table both of the Primary/Foreign keys need to be matched against.

I then used the GROUP BY clause, to group together all the sellers, to summarise by each sellers, and then I also grouped by the products code, to summarise by each product.

Then I ordered by the total quantity sold in ascesnding order.

### SCREENSHOT:

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### TESTING:

To test this I made sure that all the sellers and products are displayed. I created these queries to show all the sellers and all products. All the sellers and products are included in the query.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedTo test the calculations are correct. I created a query to output the non-summarised data of the ordered\_products table, to check that the values in my initial query add up to match the value outputted by this query.

I used as an example the Samsung Galaxy M30s SIM Free Smartphone (product\_id 3006033) sold by IntelliWorld (seller\_id 200009).

A screenshot of a computer screen

Description automatically generatedIn this example, IntelliWorld had 2 orders for this product.

One order selling 1, and another order selling 5.

At a price of £232.76, the told revenue from this is £ 1,396.56.

This matches what is displayed in out initial query.



## QUESTION D:

Text

Description automatically generated

### SQL CODE:

SELECT c.category\_description AS [Category Description],

p.product\_code AS [Product Code],

p.product\_description AS [Product Description],

PRINTF("%.2f", IFNULL(AVG(op.quantity), 0)) AS [Avg Qty Sold],

PRINTF("%.2f", subq.[Avg Qty Sold for Category]) AS [Avg Qty Sold for Category]

FROM products p

LEFT OUTER JOIN product\_sellers ps ON ps.product\_id = p.product\_id

LEFT OUTER JOIN ordered\_products op ON ps.product\_id = op.product\_id

AND ps.seller\_id = op.seller\_id

INNER JOIN categories c ON c.category\_id = p.category\_id

INNER JOIN (

SELECT c.category\_description,

AVG(op.quantity) AS [Avg Qty Sold for Category]

FROM ordered\_products op

INNER JOIN product\_sellers ps ON ps.product\_id = op.product\_id

AND ps.seller\_id = op.seller\_id

INNER JOIN products p ON p.product\_id = ps.product\_id

INNER JOIN categories c ON c.category\_id = p.category\_id

WHERE op.ordered\_product\_status != "Cancelled"

GROUP BY c.category\_id

) subq ON subq.category\_description = c.category\_description

GROUP BY p.product\_id

HAVING IFNULL(AVG(op.quantity), 0) < subq.[Avg Qty Sold for Category]

### EXPLANATION:

**Sub-query (Average of quantity sold in each category):**

Starting of with the sub-query. I obtain the average quantity sold for each category.

To do this I use an INNER JOIN to join together the order\_products, products, product\_sellers and categories tables.

I then SELECT the average of the quantity sold, using the AVG function. I also select the category\_description, so I can join to this table in the main query. (can be any unique field from the categories table)

I then add a WHERE clause to remove any ordered that have been cancelled.

I then GROUP BY the category\_id. This shows the average quantity sold for each category

**Main Query:**

In the main query, I use the sub-query as a pseudo table.

So I first use LEFT OUTER JOIN on the product\_sellers and ordered\_products tables, as all products needs to be selected even if the products has had no orders. Its worth noting also that I have to join from the products table to the product\_sellers table then to the ordered\_products, as SQLite doesn’t suppoer RIGHT OUTER JOINs.

I then use INNER JOIN to join the categores table.

I also, use an INNER JOIN to join to the sub-query. Matching on the category\_description column.

I then SELECT the required fields. To select the average quantity sold for each product I use IFNULL to convert NULL values to 0, then use PRINTF to add 2 decimal places to the end.

I now use a GROUP BY the product\_id, to summarise all the averages for each product.

I then use a HAVING, to filter out all the Average quantity sold for the products that are more than the average quantity sold for the category. (Have to use HAVING instead of WHERE as summarised data is being used).

### A screenshot of a computer Description automatically generatedSCREENSHOT:

### TESTING:

A screenshot of a computer

Description automatically generatedTo test the average quantity sold for each product, I used the Samsung Galaxy M30s SIM Free Smartphone product (product\_id 3006033) as an example. Using the following query, I calculated the average quantity sold to be:

(4 + 1 + 1 + 1 + 1+ 4 + 5 + 2) / 8 = 2.375 (2.38 2 s.f)

This matches what the query found.

A screenshot of a computer

Description automatically generatedTo test the average quantity sold for each category. I used the Mobile phones and accessories (category id 1) as an example. There are 2 products in category 1, Samsung Galaxy M30s SIM Free Smartphone and Huawei Y6 32GB 6.09 inch Smartphone. Using the query below finds the quantitys for both products.

(4 + 1 + 1 + 1 + 1 + 4 + 5 +5 +4 + 2 + 2 + 10 + 1 + 12) / 14 = 3.7857.. (3.78 2 s.f)

This matches the query

Samsung Galaxy M30s SIM Free Smartphone had a order cancelled, so that order needs to be filtered out.

To test that the correct products are showing , the HAVING statement can be removed to show all the products, or swap the symbol to > to show the products that have a greater average quantity sold than the average for the category, or = to show any products that A screenshot of a computer

Description automatically generatedhave the same averages.

A screenshot of a product list

Description automatically generatedThis now shows all 18 products, using another query we can find all the products

This means that all the products are considered in the original query, and the correct products are filtered.

# PART 2 – Database Design, Implementation and Integrity

## Text Description automatically generatedQUESTION A:

### Conceptual Design:

A diagram of products and seller reviews

Description automatically generated

My conceptual design adds 3 tables:

* Reviews
* Product reviews
* Seller Reviews

I decided to store all the reviews in one table, instead of having 2 tables storing reviews. I thought it made more sense to do this as the 2 tables would have the same column headers, this would make the database more concise.

A shopper can have many reviews, but they are not required to leave a review, so the

relationship between the shoppers table and the reviews table is optional one to many.

I have linked the Reviews table to the product reviews and seller reviews tables. These relationships are also optional one to many, as the review can be a product review, seller review or both (See assumptions).

A product can have many product reviews, but a product doesn’t have to have any reviews, so this relationship is also optional one to many.

A Seller can have many seller reviews, but a seller doesn’t have to have any reviews, so this relationship is also optional one to many.

### Logical Design:

A diagram of a product

Description automatically generated

In the logical design I have implemented the table columns.

The reviews table includes a Review\_ID as the primary key, and uses the Shopper\_ID from the shoppers table as a foreign key. The reviews table also includes, review\_star\_rated for how many stars the review is, review\_comment for any comments, and review\_date for when the review was created.

The product reviews, and seller reviews table are used to link together the products/sellers to the revires table. They use a compound primary/foreign keys from the reviews and products/sellers tables to link together the review with the product/seller.

### Physical Design:

A diagram of a product

Description automatically generated

For the physical design I have added in the column data types that will be used.

Also there are some constraints for some of the columns:

* The Review\_star\_rated column must be 1-5 stars
* The date must be in date format

Assumptions:

* The same review can be used for products and sellers, it doesn’t have to, it is just an option in the design. This could be a feature of the online shop, where a review can be created and used as a seller and product review.
* Assume a shopper can write an anonymous review
* Assume that a shopper can leave multiple reviews on the same product or seller
* Assume an empty comment can be left (Stored as a empty string, not NULL)
* Any shopper can review a product or seller even if they haven’t purchased/ordered anything from that seller or product. An alternative design would be joining to the ordered products table instead of the sellers and products tables.

## QUESTION B:

### Parana Entity Relationship Diagram:

A screenshot of a computer screen

Description automatically generated

I have rewritten the paraná ERD in drawio and have labelled the new tables and relationships in green.

## QUESTION C:

### Create Tables:

A screenshot of a computer program

Description automatically generated

When creating the reviews table, I set the review fields (review\_star\_rated, review\_date) to NOT NULL, as all of these fields are required to create a review. Also, I have made the assumption that review\_comments can be empty, but they should not be stored as a NULL values, they are stored as an empty string. Shopper\_id can be NULL, as anonymous reviews can be written.

I created 2 constraints, one for the shopper\_id as this is a foreign key from the shoppers table, and a constraint for the review\_star\_rated field using CHECK, as this field should only store “\*”, “\*\*”, “\*\*\*”, “\*\*\*\*”, “\*\*\*\*\*”.

When creating the seller\_review and product review tables, I used 3 constraints. 1 constrint was the review\_id and product\_id/seller\_id is a compound key, then the other 2 constraints are that both review\_id and product\_id/seller\_id are foreign keys. Also the review\_id and seller\_id/product\_id have NOT NULL contrainsts.

### Insert Data:

A screenshot of a computer program

Description automatically generated

I have created data that test the following requirements:

* Test that a product can store multiple reviews.
* Test that a seller can store multiple reviews.

This also tests some assumptions:

* Comments can be empty
* A shopper can leave multiple reviews of products/sellers
* A shopper can leave an anonymous review

## QUESTION D:

### Testing:

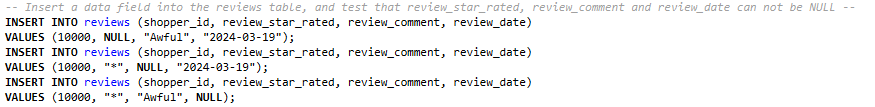
**Test the review star field can only enter 1-5 stars:**

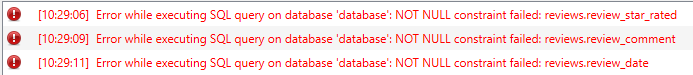




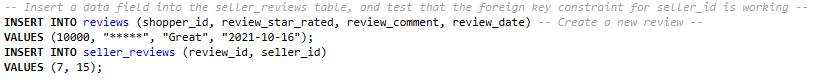
**Test foreign key constraints for shopper\_id in reviews table:**

**Test NOT NULL constraints for review\_start\_rated, review\_comment, review\_date**





**Test foreign key constraints for seller\_id in seller\_reviews:**





**Test foreign key constraints for review\_id in seller\_reviews:**

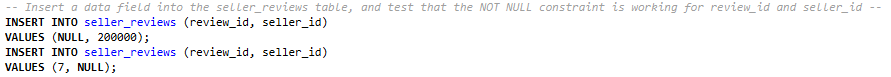
**Test unique review\_id in seller\_reviews:**

A close up of a computer screen

Description automatically generated

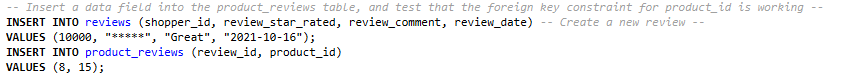


**Test NOT NULL constraint working for review\_id and seller\_id in seller\_reviews**





**Test foreign key constraints for product\_id in product\_reviews:**





**Test foreign key constraints for review\_id in product\_reviews:**

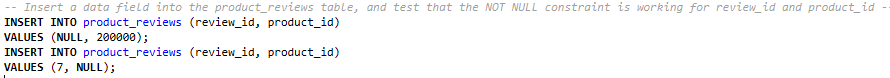
**Test unique review\_id in product\_reviews:**

A close up of a computer screen

Description automatically generated



**Test NOT NULL constraint working for review\_id and product\_id in product\_reviews**





**Full SQL Code:**

PRAGMA foreign\_keys = ON;

-- Drop tables if they exists --

DROP TABLE IF EXISTS seller\_reviews;

DROP TABLE IF EXISTS product\_reviews;

DROP TABLE IF EXISTS reviews;

-- Create tables --

CREATE TABLE reviews

(review\_id INTEGER PRIMARY KEY AUTOINCREMENT,

shopper\_id TEXT,

review\_star\_rated TEXT NOT NULL,

review\_comment TEXT NOT NULL,

review\_date TEXT NOT NULL,

CONSTRAINT shopper\_id\_fk FOREIGN KEY (shopper\_id)

REFERENCES shoppers(shopper\_id)

CONSTRAINT review\_star\_rated\_check CHECK (review\_star\_rated IN ("\*", "\*\*", "\*\*\*", "\*\*\*\*", "\*\*\*\*\*"))

);

CREATE TABLE seller\_reviews

(review\_id INTEGER NOT NULL UNIQUE,

seller\_id INTEGER NOT NULL,

CONSTRAINT seller\_reviews\_pk PRIMARY KEY (review\_id, seller\_id),

CONSTRAINT review\_id\_fk FOREIGN KEY (review\_id)

REFERENCES reviews(review\_id)

CONSTRAINT seller\_id\_fk FOREIGN KEY (seller\_id)

REFERENCES sellers(seller\_id)

);

CREATE TABLE product\_reviews

(review\_id INTEGER NOT NULL UNIQUE,

product\_id INTEGER NOT NULL,

CONSTRAINT product\_reviews\_pk PRIMARY KEY (review\_id, product\_id),

CONSTRAINT review\_id\_fk FOREIGN KEY (review\_id)

REFERENCES reviews(review\_id)

CONSTRAINT product\_id\_fk FOREIGN KEY (product\_id)

REFERENCES products(product\_id)

);

-- Insert Data --

-- Insert Review --

INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date)

VALUES (10000, "\*", "Awful", "2024-03-19");

INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date)

VALUES (10001, "\*\*\*", "Okay", "2020-01-02");

INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date)

VALUES (10003, "\*\*\*\*\*", "Amazing!", "2022-07-07");

INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date)

VALUES (10004, "\*\*", "Not Great", "2021-01-19");

INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date)

VALUES (10005, "\*\*\*\*", "Decent", "2023-11-09");

INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date)

VALUES (10000, "\*\*\*\*\*", "", "2021-10-16");

-- Insert product reviews --

INSERT INTO product\_reviews (review\_id, product\_id)

VALUES (1, 3000000);

INSERT INTO product\_reviews (review\_id, product\_id)

VALUES (2, 3000000);

INSERT INTO product\_reviews (review\_id, product\_id)

VALUES (3, 3000021);

-- Insert seller reviews --

INSERT INTO seller\_reviews (review\_id, seller\_id)

VALUES (4, 200000);

INSERT INTO seller\_reviews (review\_id, seller\_id)

VALUES (5, 200000);

INSERT INTO seller\_reviews (review\_id, seller\_id)

VALUES (6, 200001);

-- Insert an anonymous review --

INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date)

VALUES (NULL, "\*", "Awful", "2024-03-19");

INSERT INTO product\_reviews (review\_id, product\_id)

VALUES (7, 3000000);

---- Test constraints ----

-- Insert a data field into the review table, and test that the review star field can only enter stars --

/\*INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date)

VALUES (10000, "Invalid Text", "Awful", "2024-03-19");\*/

-- Insert a data field into the reviews table, and test that the foreign key constraint for shopper\_id is working --

/\*INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date)

VALUES (1, "\*", "Awful", "2024-03-19");\*/

-- Insert a data field into the reviews table, and test that review\_star\_rated, review\_comment and review\_date can not be NULL --

/\*INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date)

VALUES (10000, NULL, "Awful", "2024-03-19");

INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date)

VALUES (10000, "\*", NULL, "2024-03-19");

INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date)

VALUES (10000, "\*", "Awful", NULL);\*/

---- Test seller\_review\_table ----

-- Insert a data field into the seller\_reviews table, and test that the foreign key constraint for seller\_id is working --

/\*INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date) -- Create a new review --

VALUES (10000, "\*\*\*\*\*", "Great", "2021-10-16");

INSERT INTO seller\_reviews (review\_id, seller\_id)

VALUES (7, 15);\*/

-- Insert a data field into the seller\_reviews table, and test that the foreign key constraint for review\_id is working --

/\*INSERT INTO seller\_reviews (review\_id, seller\_id)

VALUES (15, 200000);\*/

-- Insert a data field into the seller\_reviews table, and test that the unique constraint for review\_id is working --

/\*INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date) -- Create a new review --

VALUES (10000, "\*\*\*\*\*", "Great", "2021-10-16");

INSERT INTO seller\_reviews (review\_id, seller\_id)

VALUES (7, 200000);

INSERT INTO seller\_reviews (review\_id, seller\_id)

VALUES (7, 200000);\*/

-- Insert a data field into the seller\_reviews table, and test that the NOT NULL constraint is working for review\_id and seller\_id --

/\*INSERT INTO seller\_reviews (review\_id, seller\_id)

VALUES (NULL, 200000);

INSERT INTO seller\_reviews (review\_id, seller\_id)

VALUES (7, NULL);\*/

---- Test product\_review\_table ----

-- Insert a data field into the product\_reviews table, and test that the foreign key constraint for product\_id is working --

/\*INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date) -- Create a new review --

VALUES (10000, "\*\*\*\*\*", "Great", "2021-10-16");

INSERT INTO product\_reviews (review\_id, product\_id)

VALUES (8, 15);\*/

-- Insert a data field into the product\_reviews table, and test that the foreign key constraint for review\_id is working --

/\*INSERT INTO product\_reviews (review\_id, product\_id)

VALUES (15, 200000);\*/

-- Insert a data field into the product\_reviews table, and test that the unique constraint for review\_id is working --

/\*INSERT INTO reviews (shopper\_id, review\_star\_rated, review\_comment, review\_date) -- Create a new review --

VALUES (10000, "\*\*\*\*\*", "Great", "2021-10-16");

INSERT INTO product\_reviews (review\_id, product\_id)

VALUES (8, 3000000);

INSERT INTO product\_reviews (review\_id, product\_id)

VALUES (8, 3000000);\*/

-- Insert a data field into the product\_reviews table, and test that the NOT NULL constraint is working for review\_id and product\_id --

--INSERT INTO product\_reviews (review\_id, product\_id)

--VALUES (NULL, 200000);

--INSERT INTO product\_reviews (review\_id, product\_id)

--VALUES (7, NULL);

# PART 3 – Programming for Databases

Python code is attached to the submission as a zip file, or alternatively can be cloned from the following git repo:

<https://github.com/WillBladon-Whittam/Parana-Database>

The full code will also be included at the bottom of this document.

Follow the steps in README.md to setup the python environment.

## Code Overview:

Before matching against any requirements, I will first explain the design of the project. There are only 2 python files used, “main.py” and “sql.py”. main.py is where practically the whole project is, where sql.py is a wrapper created to simplify SQL querys. This wrapper automatically handles the connection the database, and simplifies querys. I will explain the sql wrapper here, as it isn’t in the requirements, and explain the main code in the requirements description.

A screen shot of a computer code

Description automatically generatedInside the \_\_init\_\_ function, the db\_file location is parsed through, or will look for the database in the directory where the python file is run from. It then connects to the database, sets up the cursor and executes “PRAGMA foreign\_keys=ON”, to enforce foreign key constrains.

The methods in this wrapper include a method to select a query, update a table and execute a query.

A screenshot of a computer program

Description automatically generated

The “select\_query” is used to minimise effort required to create and execute an sql select query. Everything required to execute an SQL select query can be configured through the parameters.

A screen shot of a computer program

Description automatically generated

The “update\_table” method is used to handle transactions and manage errors with the query. If any errors occur when updating a table, it automatically rollsback and raised an error.

There are also methods for executing a query, and closing the database.



## Requirements Description / Testing:

**Code Description:**

“a. i. Prompt for the entry of a shopper\_id which will be used to test all the menu options. If the shopper\_id entered is found, print a welcome message including the name of the shopper. If the shopper\_id is not found in the database, print an error message and exit the program otherwise print the main menu below.”

Firstly, run the program “python main.py”. Then the session will be created, and run the ParanaShopperSession class.

A screen shot of a computer program

Description automatically generatedThe \_\_init\_\_ function is ran, where the sql wrapper is created, the user is asked for there shopper\_id, a welcome message is displayed, and there basket is retrieved. If they don’t already have a basket from today, one will be created (requirement a .iii). The main\_loop is then called, which is where the main menu is displayed.

A screen shot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated

Here the user is prompt for there shopper ID. For most user prompts throughout the project I use static methods to handle the input to remove any duplicate code, but in this case the program needs to close when an invalid input is entered, and the sql database needs to be correctly closed down.

So, if the shopper\_id is not a valid integer or isn’t in the shoppers table, a error message is displayed and closes the program.

I return the shopper\_id to store as a attribute as this used used throughout many of the methods, so the same query doesn’t need to continue to be executed.

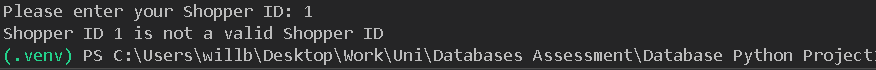
A screenshot of a computer

Description automatically generated  
**Testing:**

Here shows entering a valid shopper id, displaying the welcome message, and the main menu

A screenshot of a computer

Description automatically generatedThis shows entering text for the shopper id prints an error message and closes the program.



This shows entering an invalid shopper id prints an error message and closes the program.

“a. ii. Print a text-based menu as follows:”

A screenshot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated

When the main loop is ran at the end of the \_\_init\_\_ method, an infinite loop is created to continue the interface after different functions have been ran. I display the options, and prompt the user for a number, also validating the responses from within a specific range, and ensuring that the values entered a valid integers with the prompt\_number static method.

A screen shot of a computer program

Description automatically generated  
  
This static method is used all throughout the program. It can ensure that a specific range of numbers is entered, handles incorrect types entered and allows specific prompts and error messages to be displayed. I first check if a range is defined, if not I use a while loop to continue prompting the user for a value until there option is an integer greater than 0, displaying errors messages for each incorrect attempt. The same is applied for if there is a minimum value defined, the user must enter a value greater than a minimum specific value, and for when a minimum and a maximum value are defined, the user must enter a value between these ranges. I haven’t implemented for a scenario where only a maximum value is used, as this wasn’t required for the program.

“a. iii. As shoppers should be able to resume a basket previously created from a previous execution of the program on the same day, check if there is a row in the shopper\_baskets table created today for the selected shopper and, if so, make this the current basket. If there is more than one basket created today for the shopper, use the most recent one.”

**Code Description:**

A computer screen with text

Description automatically generated

A screen shot of a computer program

Description automatically generated

Inside the \_\_init\_\_ function, after the welcome message. The basket ID is retrieved. If the basket\_id from today is not found, one will be created.

**Testing:**There is no change in what the user sees for this. So I waited a couple days, ran the program, and checked SQLiteStudio to check the shopper\_baskets table had a new entry in for today. I then reran the program and added a temporary print statement to show the basket id, and made sure that is matched the basket created for today.

A number on a white background

Description automatically generated

A screen shot of a computer

Description automatically generated This shows the correct basket is being selected.

“Option 1 – Display your order history

i. For each order that the customer has placed, display the order id and order date together with the product description, seller name, price, quantity ordered and status of each product on that order. You can use the query you wrote for Question 1b of this assessment as a basis for the SQL query for this option.

ii. Sort orders by order date (most recent first)

iii. If no orders are found for the shopper\_id that you are testing with, print the message “No orders placed by this customer”

iv. Display the data in the format shown below (which is for shopper\_id 10010)”

v. Return to the main menu

**A screenshot of a computer program

Description automatically generatedCode Description:**

When option 1 is pressed, the order history is displayed. The SQL query is explained in Question 1b of the assessment. If there is no order history, a message “No orders placed by this customer” is displayed.

A screenshot of a computer

Description automatically generated

If orders have been found, the results are printed into a neat table. To do this is used an external library called tabulate. This takes the result of an SQL query and displays it in a neat table.

**Testing:**To test this, I displayed pressed option 1 to display the order history, and all of the orders should be displayed, then return to the menu. The SQL query has been explained and tested in Question 1b.

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

I also tested with a different shopper (shopper id 10023). This shopper has no previous order history. So a message is displayed saying “no orders placed by this customer”, then return to the menu

b. Option 7 – Exit the program

i. Exit the program

**Code Description:**

A computer screen shot of text

Description automatically generatedWhen option 7 is pressed, the close methos is called, this will close the SQL database and exit the program  
**Testing:**

A screenshot of a computer

Description automatically generatedTo test this I just entered option 7, and the program successfully closed.

b. Option 2 – Add an item to your basket

i. Display a numbered list of product categories in alphabetical order

ii. Prompt the user to enter the number of the product category they want to choose from and store the category\_id for the selected category

**Code Description:**

A computer code with many colorful text

Description automatically generated with medium confidence

When Option 2 is pressed, an item can be added to the basket. Firstly in this method, all the product categories are retrieved in alphabetical order. These are then displayed to the user in a number list, with the display options static method.

A screenshot of a computer program

Description automatically generatedThis will number each row of an SQL query result and print it out.

After the options have been displayed, the user is prompted to enter the number they would like to choose. The prompt is validated by the prompt\_number static method.

After that, the selected category\_id is retrieved using the selected number mapped against the product description. Its important not to just use the number entered by the user. As this is an auto generated number NOT the category\_id.

b. Option 2 – Add an item to your basket

iii. Display a numbered list of the available products in the category selected in alphabetical order.

iv. Prompt the user to enter the number of the product they want to purchase and store the product\_id for the selected product

A computer screen shot of a program

Description automatically generatedUsing the category id that the user selected, all the products are retrieved in alphabetical order. These are then displayed to the user in a number list, with the display options static method.

After the options have been displayed, the user is prompted to enter the number they would like to choose. The prompt is validated by the prompt\_number static method.

After that, the selected product\_id is retrieved using the selected number mapped against the product description. Its important not to just use the number entered by the user. As this is an auto generated number NOT the product\_id.

b. Option 2 – Add an item to your basket

v. Display a numbered list of sellers who sell the product they have selected and the price they are selling that product at in alphabetical order by seller name

vi. Prompt the user to enter the seller they wish to buy the product from and store the seller\_id for the selected seller

A screen shot of a computer code

Description automatically generated

Using the product id that the user selected, all the sellers are retrieved in alphabetical order. The price is all retrieved to show the different prices from each seller in brackets. (requirement b. vii). These are then displayed to the user in a number list, with the display options static method.

After the options have been displayed, the user is prompted to enter the number they would like to choose. The prompt is validated by the prompt\_number static method.

After that, the selected seller\_id is retrieved using the selected number mapped against the seller description. Its important not to just use the number entered by the user. As this is an auto generated number NOT the seller\_id.

b. Option 2 – Add an item to your basket

vii. Prompt the user to enter the quantity of the selected product they want to order. Display ‘The quantity must be greater than 0’ if the quantity is <=0 and re-prompt the user to enter it again



Using the prompt\_number static method, the user enters there selected quantity.

b. Option 2 – Add an item to your basket

viii. Get the price of the selected product from the selected supplier

This has already been retrieved where querying the available sellers.

b. Option 2 – Add an item to your basket

ix. If there is no current basket, get the next basket id by selecting from the sqlite\_sequence table and insert a new row into the shopper\_baskets table using the next basket\_id

This has also already been covered when initiating the session. The basket\_id is stored as an

attribute in the class.

b. Option 2 – Add an item to your basket

x.Insert a new row into the basket\_contents table for the product they’ve chosen to purchase using the basket id selected in stage ix. All items added to the basket should have the same basket\_id in the basket\_contents table.

xi.Commit the transaction

xii.Print “Item added to your basket”

xiii.Return to the main menu

A black screen with text

Description automatically generated

Using all of the information entered from the user, a new row is inserted into the basket\_contents table.

A screenshot of a computer screen

Description automatically generated

I use the remove\_money\_format, as the format that was returned from the SQL query displayed the format with a pound sign and brackets. This is not how is should be stored in the database, so I remove these from the value.

The transaction is then committed by the sql wrapper, and the item is added to the basket, and a message is printed to the user.

Also, the SQL database returns an integrity error when trying to add multiple of the same items to the basket. So I also print a message telling the user to change the quantity or delete the item instead.

**Testing:**

A screenshot of a computer program

Description automatically generatedThis shows all the prompts are correct, also testing that the quantity entered has to be an integer greater than 0. The main menu is displayed after.

A close-up of a number

Description automatically generatedChecking SQLiteStudio also shows the order has been correctly added to the table.

b. Option 3 – Display your basket

i.If the basket is empty, display ‘Your basket is empty’ otherwise display all rows from the basket\_contents table for the current basket, labelling each item with a basket item no. starting at 1. Also display a total basket cost.

ii. Return to the main menu

**Code Description:**

A screenshot of a computer program

Description automatically generated

To display the contents of the basket, I use the ROW\_NUMBER() OVER() SQL functions to

label each row returned with an incremented number. I then select the rest of the values

from the requirements and join the tables needed, also returning only the values with the

shoppers basket\_id.

If the basket contents are empty, a message is printed and then return to the main menu.

If the basket has items in they a printed to the screen in a nice table.

To display the totals to the screen I added an extra tuple to the list that the SQL query

returns to display the basket total underneath where all the totals for the items are.

I also return the basket contents from the method, as this can be reused in other options to

Get the basket contents.

**Testing:**

A screenshot of a computer

Description automatically generated

This shows the basket items, with the incremented back item number, all the values from the requirements and the basket total. Then returns to the main menu

A screenshot of a computer

Description automatically generatedI also tested that with an empty basket a message is displayed and it returns back to the main menu.

b. Option 4 – Change the quantity of an item in your basket

i.If the basket is empty, display ‘Your basket is empty’ and return to the main menu otherwise display the current basket and the basket total (as per option 3.

ii.If there is more than one item in the basket, prompt the user to enter the basket item no. of the item they want to update. If they enter an invalid basket item no., display ‘The basket item no. you have entered is invalid’ and re-prompt the user to enter it again.

If there is only one item in the basket, this will obviously be the one the user wants to change.

iii.Prompt the user to enter the new quantity for the item selected. If they enter a quantity <= 0, display ‘The quantity must be greater than 0’ and re-prompt the user to enter it again.

iv.Update the basket\_contents table with the new quantity for the current basket and item that has been changed.

v.Display the current basket with a re-calculated total.

vi.Return to the main menu

**Code Description:**

A computer screen with text

Description automatically generated

Here I display and get the basket contents, if there is nothing in the basket, I just exit out

the method. The empty basket message will have already been displayed in the

display\_basket method if the basket is empty.

If the length if the basket is greater than 1 (2, as I have added an extra column with the

basket totals), then the user is prompted to pick a basket item number. Displaying an error

message and reprompting if incorrect.

If there is only 1 item item in the basket, then they must want to change that item.

Then the user is prompted to enter there new quantity. Displaying an error

message and reprompting if incorrect.

I then use an SQL UPDATE to update the quantity values.

I use an embedded SQL Query to get the product\_id from the product\_description.

**Testing:**

A screenshot of a computer program

Description automatically generated

This shows that both prompts for the basket\_id and the quantity are correctly validated,

all the basket contents are display before and after, and the user is returned to the main

menu.

A screenshot of a number

Description automatically generated

Checking in SQLiteStudio shows the updated quantity value.

A screenshot of a computer

Description automatically generated

This shows that if ther is only 1 item in the basket, the prompt to enter the basket item number is skipped, and so just the new quantity prompt shows.

A screenshot of a computer screen

Description automatically generatedThis also shows that id the user has an empty basket, a empty basket message is displayed, and the user is returned to the main menu.

b. Option 5 – Remove an item from your basket

i.If the basket is empty, display ‘Your basket is empty’ otherwise display the current basket and the basket total as per option 3.

ii.If there is more than one item in the basket, prompt the user to enter the basket item no. of the item they want to remove. If they enter an invalid basket item no., display ‘The basket item no. you have entered is invalid’ and re-prompt the user to enter it again.

If there is only one item in the basket, this will obviously be the one the user wants to remove.

iii.Prompt the user to confirm they definitely want to remove the selected item from their basket by entering Y or N.

iv.If the user confirms they definitely want to remove the selected item, delete the item from the current basket in the basket\_contents table.

v.Check if the basket is now empty and if so, display ‘Your basket is empty’ otherwise display the current basket with a re-calculated total.

vi.Return to the main menu

**Code Description:**

A screenshot of a computer program

Description automatically generated

Here I display and get the basket contents, if there is nothing in the basket, I just exit out

the method. The empty basket message will have already been displayed in the

display\_basket method if the basket is empty.

If the length if the basket is greater than 1 (2, as I have added an extra column with the

basket totals), then the user is prompted to pick a basket item number. Displaying an error

message and reprompting if incorrect.

If there is only 1 item item in the basket, then they must want to delete that item.

Then the user is prompt if they definitely want to delete that item from there basket.

A screen shot of a computer program

Description automatically generatedI use a static method prompt\_yes\_no to ensure the user can only enter “y” or “Nb (can be capitalised), it will continue to reprompt the user with an error message until the user has entered a valid entry. A boolean will be returned.

If the user says “n”, then nothing is deleted and the user is returned to the main menu. If

The user says “y”, then an SQL DELETE is used to remove that product from there basket.

I use an embedded SQL Query to get the product\_id from the product\_description.

**Testing:**

A screenshot of a computer

Description automatically generated

To test this I added 2 items to the basket, then tested that it only accepts integers of item

Of the item baskets that are shown. When a valid basket item number has been entered,

the user is prompt with a Y/N prompt. When Y is pressed the item is removed and the new

basket is displayed.

A close-up of a person

Description automatically generatedChecking the SQLiteStudio database, the item has been successfully removed.

A screenshot of a computer program

Description automatically generated

Now as there is only 1 item in the basket. There is only a Y/N prompt as the user must want

to remove the only item in the basket. When the item is removed, the message “Your

basket is now empty” is displayed.

A screenshot of a computer screen

Description automatically generatedIf there are no items in the basket and the user tries to remove an item they will be prompt with a “Your basket is now empty” message.

A screenshot of a computer

Description automatically generated

If the user says “n” to deleting the item, the basket will be displayed with the item still

There an return to the main menu.

b. Option 6 – Checkout your basket

i.If the basket is empty, display a suitable message and return to the main menu

ii.Display the current basket and the basket total (the same as option 3) and ask the user if they wish to proceed with the checkout (Y or N). If they enter N, return to the main menu. If they enter Y, continue as follows:

iii.Insert a new row into the shopper\_orders table for the basket with a status of ‘Placed’

iv.Insert a new row into the ordered\_products table for each item in the basket with a status of ‘Placed’

v.Delete the rows from the basket\_contents and shopper\_baskets tables for this basket

vi.Print the message ‘Checkout complete, your order has been placed’

vii.Return to the main menu

**Code Description:**

A screenshot of a computer program

Description automatically generated

Here I display and get the basket contents, if there is nothing in the basket, I just exit out

the method. The empty basket message will have already been displayed in the

display\_basket method if the basket is empty.

The user is then prompt with a Y/N if they definitely want to checkout. If they say N, they

Are returned to the main menu. If they say yes the tables are updated to reflect this.

Firstly, the order is added to the shoppers\_orders tables. I make sure not to commit the

Changes to the database until all tables have been successfully updated to keep the

Integrity of the database. I store the order\_id, as this will be used in the next table.

After adding a row to the shoppers\_orders table, I create a row for each item in the basket.

I then delete the contents of the basket, and the basket itself. The table is the committed

after all transactions have been successful.

Then the user is prompt with a checkout complete message and is returned to the main

menu.

**Testing:**

A screenshot of a computer

Description automatically generated

To test this works, I added some items to my basket, pressed option 6 to checkout, tested

that only “Y” and “N” can be entered. When “Y” is entered. The use is displayed with a

checkout complete message and returned to the main menu.

I checked all the database changed in SQLiteStudio and all the tables have been correctly

changed.

A close up of numbers

Description automatically generated

Shoppers\_order has been placed

A screenshot of a computer

Description automatically generated

Both the products are in the ordered\_products table.

A close-up of a word

Description automatically generated

The shopper\_contetns table is empty

The A screenshot of a computer code

Description automatically generatedshopper\_baskets table does not show the basket from today (28/04/2024)

A screenshot of a computer screen

Description automatically generatedChecking out with an empty basket also displays a message to the user and then returns to the main menu.

A screenshot of a computer

Description automatically generated

Saying “N” at checkout then returns to the main menu. Checking the basket contents the basket items are still there so nothing has been checked out.

## Full Python Code:

### Main.py:

from sql import SqlWrapper

from tabulate import tabulate

import datetime

from typing import Tuple, Union, List

*"""*

*Store database in the root directory as 'database' or specify when calling SqlWrapper*

*Links Used:*

*https://learnpython.com/blog/print-table-in-python/ -> Pretty print a SQL query results in a table*

*https://learnsql.com/cookbook/how-to-number-rows-in-sql/ -> Number rows returned from an SQL Query*

*https://www.sqlite.org/lang\_datefunc.html -> DATE('now'), was returung the incorrect date at midnight. Needs to use local timezone.*

*"""*

class ParanaShopperSession:

*"""*

*Creates a Shopper session based on the Parana database*

*"""*

    def \_\_init\_\_(self) -> None:

        self.sql = SqlWrapper()

        self.shopper\_id = self.get\_shopper\_id()

        self.welcome()

        self.basket\_id = self.get\_basket\_id()

        if not self.basket\_id:

            self.basket\_id = self.create\_basket()

        else:

            self.basket\_id = self.basket\_id[0]

        self.main\_loop()

    @staticmethod

    def pretty\_print(results: Union[List[Tuple], Tuple], headers: List[str] = []) -> None:

*"""*

*Pretty print a table based on the results from an SQL table*

*"""*

        print(tabulate(results, headers), "\n")

    @staticmethod

    def display\_options(options: List[Tuple[str]]) -> None:

*"""*

*Displays the options returned from an SQL query as a numbered list*

*"""*

        for i, option in enumerate(options, start=1):

            print(f"{i}.\t{"  ".join(str(row) for row in option)}")

        print("\n")

    @staticmethod

    def prompt\_yes\_no(prompt: str) -> bool:

*"""*

*Prompts the user for a Y/N response*

*"""*

        selected\_option = None

        while selected\_option not in ["Y", "N", "y", "n"]:

            selected\_option = input(prompt)

            if selected\_option not in ["Y", "N", "y", "n"]:

                print("Please enter Y/N\n")

        if selected\_option.upper() == "Y":

            return True

        elif selected\_option.upper() == "N":

            return False

    @staticmethod

    def prompt\_number(prompt: str, \_range: Tuple[int, Union[int, None]] = None, error\_message: str = "Invalid Value!") -> int:

*"""*

*Prompts the user for a number between a specific range.*

*"""*

        selected\_option = -1

        if \_range is None:  # If no range is specified

            while not selected\_option > 0:

                try:

                    selected\_option = int(input(prompt))

                except ValueError:

                    print(f"{error\_message}\n")

                    continue

                if selected\_option <= 0:

                    print(f"{error\_message}\n")

            return selected\_option

        elif \_range[1] is None:  # If there is a minimum value

            while not \_range[0] <= selected\_option:

                try:

                    selected\_option = int(input(prompt))

                except ValueError:

                    print(f"{error\_message}\n")

                    continue

                if selected\_option <= \_range[0]:

                    print(f"{error\_message}\n")

        else:  # If there is a range of 2 values

            while selected\_option not in range(\_range[0], \_range[1]+1):

                try:

                    selected\_option = int(input(prompt))

                except ValueError:

                    print(f"{error\_message}\n")

                    continue

                if selected\_option not in range(\_range[0], \_range[1]+1):

                    print(f"{error\_message}\n")

        return selected\_option

    @staticmethod

    def remove\_money\_format(value: str) -> List[Tuple[int, str]]:

*"""*

*Remove strings that are formatted with £ and/or ().*

*e.g. convert (£5.32) -> 5.32*

*"""*

        return value.replace("£", "").replace(")", "").replace("(", "")

    def get\_shopper\_id(self) -> int:

*"""*

*Return the shopper ID, validating that the shopper ID is in the database*

*"""*

        try:

            shopper\_id = int(input("Please enter your Shopper ID: "))

        except ValueError:

            print("Shopper ID is not a valid Shopper ID")

            self.close()

        shoppers = self.sql.select\_query("SELECT shopper\_id "

                                         "FROM shoppers")

        if shopper\_id not in [shopper[0] for shopper in shoppers]:

            print(f"Shopper ID {shopper\_id} is not a valid Shopper ID")

            self.close()

        return shopper\_id

    def get\_basket\_id(self) -> int:

*"""*

*If there is a basket created from today, use that basket and return the basket\_id*

*"""*

        # Must use localtime, otherwise breaks at midnight

        return self.sql.select\_query("SELECT basket\_id "

                                     "FROM shopper\_baskets "

                                     "WHERE shopper\_id = ? AND DATE(basket\_created\_date\_time) = DATE('now', 'localtime') "

                                     "ORDER BY basket\_created\_date\_time DESC "

                                     "LIMIT 1", sql\_parameters=(self.shopper\_id,), fetch="one")

    def create\_basket(self) -> int:

*"""*

*If a basket is not already created from today, then create a new basket*

*"""*

        self.sql.update\_table("INSERT INTO shopper\_baskets (shopper\_id, basket\_created\_date\_time) "

                              "VALUES (?, ?)", sql\_parameters=(self.shopper\_id, datetime.datetime.now().strftime("%Y-%m-%d"),))

        return self.sql.cursor.lastrowid

    def welcome(self) -> None:

*"""*

*Prints a welcome message to the shopper*

*"""*

        shopper\_first\_name, shopper\_surname = self.sql.select\_query("SELECT shopper\_first\_name, shopper\_surname "

                                                                    "FROM shoppers "

                                                                    "WHERE shopper\_id = ? ",

                                                                    sql\_parameters=(self.shopper\_id,),

                                                                    fetch="one")

        print(f"Welcome {shopper\_first\_name} {shopper\_surname}!\n")

    def display\_order\_history(self) -> None:

*"""*

*Display the order history of the shopper (Option 1)*

*"""*

        order\_history = self.sql.select\_query("SELECT so.order\_id, "

                                              "STRFTIME('%d-%m-%Y', so.order\_date), "

                                              "p.product\_description, "

                                              "se.seller\_name, "

                                              "PRINTF('£%.2f',op.price), "

                                              "op.quantity, "

                                              "ordered\_product\_status "

                                              "FROM shoppers as s "

                                              "INNER JOIN shopper\_orders so ON s.shopper\_id = so.shopper\_id "

                                              "INNER JOIN ordered\_products op ON so.order\_id = op.order\_id "

                                              "INNER JOIN product\_sellers ps ON op.product\_id = ps.product\_id "

                                              "AND op.seller\_id = ps.seller\_id "

                                              "INNER JOIN sellers se ON ps.seller\_id = se.seller\_id "

                                              "INNER JOIN products p ON ps.product\_id = p.product\_id "

                                              "WHERE s.shopper\_id= ? "

                                              "ORDER BY so.order\_date DESC", sql\_parameters=(self.shopper\_id,))

        if not order\_history:

            print("No orders placed by this customer\n")

        else:

            self.pretty\_print(results=order\_history, headers=[

                              "Order ID", "Order Date", "Product Description", "Seller", "Price", "Qty", "Status"])

    def add\_item(self) -> None:

*"""*

*Add item to the shoppers basket. (Option 2)*

*Choose product categories -> products -> sellers -> quantity*

*"""*

        product\_categories = self.sql.select\_query("SELECT category\_description "

                                                   "FROM categories "

                                                   "ORDER BY category\_description ASC ")

        self.display\_options(product\_categories)

        selected\_category = self.prompt\_number(prompt="Enter the number against the product category you want to choose: ",

                                               \_range=(1, len(product\_categories)))

        selected\_category\_id = self.sql.select\_query("SELECT category\_id "

                                                     "FROM categories "

                                                     "WHERE category\_description = ?", sql\_parameters=product\_categories[selected\_category-1], fetch="one")

        products = self.sql.select\_query("SELECT product\_description "

                                         "FROM products "

                                         "WHERE category\_id = ? "

                                         "ORDER BY product\_description ASC ", sql\_parameters=selected\_category\_id)

        self.display\_options(products)

        selected\_product = self.prompt\_number(prompt="Enter the number against the product you want to choose: ",

                                              \_range=(1, len(products)))

        selected\_product\_id = self.sql.select\_query("SELECT product\_id "

                                                    "FROM products "

                                                    "WHERE product\_description = ?", sql\_parameters=products[selected\_product-1], fetch="one")

        sellers = self.sql.select\_query("SELECT s.seller\_name, PRINTF('(£%.2f)', ps.price) "

                                        "FROM sellers s "

                                        "INNER JOIN product\_sellers ps ON s.seller\_id = ps.seller\_id "

                                        "WHERE ps.product\_id = ? "

                                        "ORDER BY s.seller\_name ASC ", sql\_parameters=selected\_product\_id)

        self.display\_options(sellers)

        selected\_seller = self.prompt\_number(prompt="Enter the number against the seller you want to choose: ",

                                             \_range=(1, len(sellers)))

        selected\_seller\_id = self.sql.select\_query("SELECT seller\_id "

                                                   "FROM sellers "

                                                   "WHERE seller\_name = ?", sql\_parameters=sellers[selected\_seller-1][0], fetch="one")

        quantity = self.prompt\_number(prompt="Enter the quantity of the selected product you want to buy: ", \_range=(1, None),

                                      error\_message="The quantity must be greater than 0")

        # NOTE: ix. in the brief says to create a new basket here, if there is not already one. This is done when initialising the session.

        query\_status = self.sql.update\_table("INSERT INTO basket\_contents (basket\_id, product\_id, seller\_id, quantity, price) "

                                             "VALUES (?, ?, ?, ?, ?)",

                                             sql\_parameters=(self.basket\_id, selected\_product\_id[0], selected\_seller\_id[0], quantity,

                                                             self.remove\_money\_format(sellers[selected\_seller-1][1])))

        if query\_status is None:

            print("Item added to your basket\n")

        else:

            print("That item is already in your basket. Please edit the quantity of the item in Option 4, or delete the item in Option 5\n")

    def display\_basket(self) -> None:

*"""*

*Display the contents of the shoppers basket (Option 3)*

*"""*

        basket\_contents = self.sql.select\_query("SELECT ROW\_NUMBER() OVER(), p.product\_description, s.seller\_name, bc.quantity, PRINTF('£%.2f', bc.price), PRINTF('£%.2f', bc.price \* bc.quantity) "

                                                "FROM basket\_contents bc "

                                                "INNER JOIN product\_sellers ps ON bc.product\_id = ps.product\_id and bc.seller\_id = ps.seller\_id "

                                                "INNER JOIN products p ON p.product\_id = ps.product\_id "

                                                "INNER JOIN sellers s ON s.seller\_id = ps.seller\_id "

                                                "WHERE bc.basket\_id = ?", sql\_parameters=self.basket\_id)

        if not basket\_contents:

            print("Your basket is empty\n")

            return basket\_contents

        else:

            # Abit of a botched way... looks okay so oh well.

            basket\_contents.append((None, None, None, None, "Basket Total", f'£{sum(

                [float(self.remove\_money\_format(item[5])) for item in basket\_contents])}'))

            self.pretty\_print(basket\_contents, headers=[

                              "Basket Item", "Product Description", "Seller Name", "Qty", "Price", "Total"])

        return basket\_contents

    def change\_quantity(self) -> None:

*"""*

*Change the quantity of an item in the basket. (Option 4)*

*The basket\_contents table has 2 primary keys. basket\_id and product\_id.*

*So there can only be 1 of a product added to a basket, even if they are from different sellers.*

*"""*

        basket\_contents = self.display\_basket()

        if not basket\_contents:

            return

        # basket\_contents is returned with the botched basket total column, so 2 instead of 1.

        if len(basket\_contents) > 2:

            basket\_item\_number = self.prompt\_number("Enter the basket item no. of the item you want to change: ", \_range=(1, len(basket\_contents)-1),

                                                    error\_message="The basket item no. you have entered is invalid")

        else:

            basket\_item\_number = 1

        quantity = self.prompt\_number("Enter the new quantity of the selected product you want to buy: ", \_range=(1, None),

                                      error\_message="The quantity must be greater than 0")

        self.sql.update\_table("UPDATE basket\_contents "

                              "SET quantity = ? "

                              "WHERE product\_id = (SELECT product\_id "

                              "FROM products "

                              "WHERE product\_description = ?) and basket\_id = ? ", sql\_parameters=(quantity, basket\_contents[basket\_item\_number-1][1], self.basket\_id))

        self.display\_basket()

    def remove\_item(self) -> None:

*"""*

*Remove an item from the shoppers basket (Option 5)*

*"""*

        basket\_contents = self.display\_basket()

        if not basket\_contents:

            return

        # basket\_contents is returned with the botched basket total column, so 2 instead of 1.

        if len(basket\_contents) > 2:

            basket\_item\_number = self.prompt\_number("Enter the basket item no. of the item you want to change: ", \_range=(1, len(basket\_contents)-1),

                                                    error\_message="The basket item no. you have entered is invalid")

        else:

            basket\_item\_number = 1

        answer = self.prompt\_yes\_no(

            "Do you definitly want to delete this product from your basket (Y/N)? ")

        if answer:

            self.sql.update\_table("DELETE FROM basket\_contents "

                                  "WHERE product\_id = (SELECT product\_id "

                                  "FROM products WHERE "

                                  "product\_description = ?) and basket\_id = ?", sql\_parameters=(basket\_contents[basket\_item\_number-1][1], self.basket\_id))

            self.display\_basket()

        else:

            return

    def checkout(self) -> None:

*"""*

*Checkout the the shoppers basket. (Option 6)*

*Add order to shoppers\_orders and ordered\_products tables.*

*Remove the basket contents for the shopper and the basket from basket\_contents and shopper\_baskets*

*"""*

        basket\_contents = self.display\_basket()

        if not basket\_contents:

            return

        answer = self.prompt\_yes\_no(

            "Do you wish to proceed with the checkout (Y/N)? ")

        if answer:

            # Dont commit any changes untill the final query. To ensure all tables are updated correctly first.

            # Insert row into shoppers\_orders

            self.sql.update\_table("INSERT INTO shopper\_orders (shopper\_id, order\_date, order\_status) "

                                  "VALUES (?, ?, ?)", sql\_parameters=(self.shopper\_id, datetime.datetime.now().strftime("%Y-%m-%d"), "Placed"), commit=False)

            order\_id = self.sql.cursor.lastrowid

            # Insert row into ordered\_products

            for item in basket\_contents[:-1]:

                self.sql.update\_table("INSERT INTO ordered\_products (order\_id, product\_id, seller\_id, quantity, price, ordered\_product\_status) "

                                      "VALUES (?, (SELECT product\_id "

                                      "FROM products WHERE "

                                      "product\_description = ?), (SELECT seller\_id "

                                      "FROM sellers WHERE "

                                      "seller\_name = ?), ?, ?, ?)",

                                      sql\_parameters=(order\_id, item[1], item[2], item[3], self.remove\_money\_format(item[4]), "Placed"), commit=False)

            # Delete rows in basket\_contents associated with the basket

            self.sql.update\_table("DELETE FROM basket\_contents "

                                  "WHERE basket\_id = ?", sql\_parameters=(self.basket\_id), commit=False)

            # Delete basket in shopper\_baskets

            self.sql.update\_table("DELETE FROM shopper\_baskets "

                                  "WHERE basket\_id = ?", sql\_parameters=(self.basket\_id))

            print("Checkout complete, your order has been placed\n")

        else:

            return

    def close(self):

        self.sql.close()

        quit()

    def main\_menu(self) -> int:

*"""*

*Displays the main menu, returning the navigation integer*

*"""*

        print("PARANÁ – SHOPPER MAIN MENU\n"

              "----------------------------------------------------\n"

              "1.\tDisplay your order history\n"

              "2.\tAdd an item to your basket\n"

              "3.\tView your basket\n"

              "4.\tChange the quantity of an item in your basket\n"

              "5.\tRemove an item from your basket\n"

              "6.\tCheckout\n"

              "7.\tExit\n")

        return self.prompt\_number("Select an option: ", \_range=(1, 7))

    def main\_loop(self) -> None:

*"""*

*The main loop of the session*

*"""*

        while True:

            # NOTE: match/case statements are python 3.10+

            match self.main\_menu():

                case 1:

                    self.display\_order\_history()

                case 2:

                    self.add\_item()

                case 3:

                    self.display\_basket()

                case 4:

                    self.change\_quantity()

                case 5:

                    self.remove\_item()

                case 6:

                    self.checkout()

                case 7:

                    self.close()

if \_\_name\_\_ == "\_\_main\_\_":

    ParanaShopperSession()

### sql.py

import sqlite3

from typing import Literal, Union, Tuple

class SqlWrapper:

*"""*

*SQL Wrapper to create a connection to the database automatically and handle queries*

*"""*

    def \_\_init\_\_(self, db\_file: str = r".\database") -> None:

        self.db\_file = db\_file

        self.db = sqlite3.connect(self.db\_file)

        self.cursor = self.db.cursor()

        self.cursor.execute("PRAGMA foreign\_keys=ON")

    def \_\_str\_\_(self):

        return f"SQL Database wrapper for: {self.db\_file}"

    def execute(self, sql\_query: str, sql\_parameters: Tuple[str, int] = tuple()) -> None:

        self.cursor.execute(sql\_query, sql\_parameters)

    def select\_query(self, sql\_query, sql\_parameters: Tuple[str, int] = tuple(), fetch: Literal['all', 'many', 'one'] = "all", num\_fetch: int = 1):

*"""*

*Creates a SELECT query*

*Args:*

*sql\_query: An SQL Query to execute*

*sql\_parameters: Parameters for an SQL query*

*fetch: If set to "all", fetches all the rows returned,*

*If set to "one" returns only 1.*

*If set to "many" returns a set number of rows, specified by num\_fetch*

*"""*

        if not isinstance(sql\_parameters, tuple):

            sql\_parameters = (sql\_parameters,)

        self.execute(sql\_query, sql\_parameters)

        if fetch == "all":

            return self.cursor.fetchall()

        elif fetch == "many":

            return self.cursor.fetchmany(num\_fetch)

        elif fetch == "one":

            return self.cursor.fetchone()

    def update\_table(self, sql\_query, sql\_parameters: Tuple[str, int] = tuple(), commit=True) -> Union[None, Exception]:

*"""*

*Creates a INSERT/UPDATE/DELETE query*

*Args:*

*sql\_query: An SQL Query to execute*

*sql\_parameters: Parameters for an SQL query*

*commit: Commit changes to database immediatly*

*"""*

        if not isinstance(sql\_parameters, tuple):

            sql\_parameters = (sql\_parameters,)

        try:

            self.execute(sql\_query, sql\_parameters)

        except sqlite3.IntegrityError as e:

            self.db.rollback()

            return e

        except sqlite3.Error as e:

            self.db.rollback()

            print("Database Error!")

        if commit:

            self.db.commit()

    def close(self) -> None:

        self.db.close()

if \_\_name\_\_ == "\_\_main\_\_":

    sql = SqlWrapper(r"C:\Users\willb\Desktop\Work\Uni\Databases Assessment\database")

    print(sql)

## Bibliography

<https://learnpython.com/blog/print-table-in-python/> -> Pretty print a SQL query results in a table

<https://learnsql.com/cookbook/how-to-number-rows-in-sql/> -> Number rows returned from an SQL Query

<https://www.sqlite.org/lang_datefunc.html> -> DATE('now'), was returning the incorrect date at midnight. Needs to use local timezone.