Assignment Guidance and Front Sheet

Student ID or IDs for group work	**U1921983**

Module Title & Code	*WM264 Smart solutions development II*
Module Owner	*Jianhua Yang*
Module Tutor	
Module Marker	
Assesment type	*Portfolio assignment*
Date Set	*28 September 2020*
Submission Date (excluding extensions)	*9 November 2020 (12 noon) via Tabula*
Marks return date (excluding extensions)	*9 December 2020*
Weighting of mark	*40%*

Assessment Detail

The assessment consists of solutions to three questions, known as Challenges. Each solution and the final report carries an equal weight of 25% of the assessment.

Additional details

Given a scenario where a database system is required in a company, the three Challenges are:

1. *Describing the conceptual design considerations.*
2. *Setting up the database using SQL and designing queries.*
3. *Manipulating the data in a batch manner using programming languages.*

More details about the scenario and specific requirements can be found in the assignment brief.

Module learning outcomes

LO1: Know the role of data management systems in managing organisational data and information, and use a high-level language to create user interface for accessing data from DBMS.
LO2: Describe the logical and conceptual data modelling, make entity relationship model for incorporating system and user requirements.
LO3: Use SQL to perform simple queries including adding, updating and deleting queries.
LO4: Identify the data redundancy problems and update anomalies, apply data normalization techniques to combat the data redundancy problem.
LO5: Use analytical and critical thinking skills to technology solutions development, analyse and apply structured problem-solving techniques to complex systems and situations

Learning outcomes assessed in this assessment

LO1/3

Marking guidelines

0-40%

Challenge No1: Limited understanding of the role of data management. Business process missing or completely unsuitable.
Challenge No2: Not attempted, or very limited attempts.
Challenge No3: Not attempted, or very limited attempts.
Report and Coding Style: The report contains lots of grammar mistakes. The logic behind the code is difficult to follow.

41 – 60%

Challenge No1: Most aspects of the challenge are addressed, however, there is a lack of in-depth theoretical analysis. Limited understanding of the business process.
Challenge No2: Fulfil basic functionality requirements. Demonstrate a good understanding of basic SQL commands.

Challenge No3: A solution is provided. However, it shows a limited understanding of structures and functions available in the particular programming language.
Report and Coding Style: The report makes use of illustrations. The code is highlighted. however, it contains some errors.

61 – 80%
Challenge No1: Good command of database terminology. Able to explain the design choice using database theory and provide high-level summative justification. Good business process.
Challenge No2: Good use of advanced SQL features. Some successful attempts to implement additional features.
Challenge No3: The code uses various functions and structures in the chosen programming language to solve the challenge.
Report and Coding Style: The report is clearly structured and well written with little mistakes. The code is well commented.

81 – 100%
Challenge No1: Very good understanding of existing theory. Successful attempts are made linking the database theory with application practice. Well thought out and completely appropriate business process.
Challenge No2: Functionality is optimized to achieve better performance.
Challenge No3: Make use of 3rd party libraries that are not explicitly taught in the module.
Report and Coding Style: Advanced use of diagrams for illustration. The coding style is consistent and follows the convention.

Submission guidance

You should submit your solutions to all three challenges in a single Word or PDF file:

- *The document should be clearly structured using Challenge titles and query aims as headings.*
- *SQL queries should be properly formatted and highlighted. It should contain necessary comments for anyone without in-depth background knowledge to understand, and images of your queries before and after their execution.*
- *In case you use Python for Challenge No3, your Python code should be embedded into the word file. It should be color-highlighted and contain necessary instructions e.g. dependencies to reproduce the results including appropriate comments.*

The submission link is on Tabula.

Resubmission details

The University policy is that students should be given the opportunity to remedy any failure at the earliest opportunity. What that "earliest opportunity" means in terms of timing and other arrangements is different depending on Programme (i.e. Undergraduate, Full Time Masters, Part Time Postgraduate, or Overseas). Students are advised to consult your Programme Team or intranet for clarity.

Late submission details

If work is submitted late, penalties will be applied at the rate of **5 marks per University working day** after the due date, up to a **maximum of 10 working days** late. After this period the mark for the work will be reduced to 0 (which is the maximum penalty). "Late" means **after the submission deadline time as well as the date** – work submitted after the given time even on the same day is counted as 1 day late, as per University Regulation 36.3

Contents

Challenge 1:....4

1.1. User Definition....4

1.2. Components and Architecture....4

Jobs Table....4

Branch Table....5

Shop Table....5

Employee Table....6

Transactions Table....6

Product Table....7

Audit Table....7

Keys....8

EER Diagram....8

1.3. Business Process....9

1.4. Advantages....9

1.5. Disadvantages....9

Challenge 2:....9

1.6. Creating Database & Tables....9

1.7. Inserting Data....12

1.8. Triggers & Procedures....14

1.9. Views....21

1.10. Challenge Query Proof....23

The discount that is given to all employees.....23

Retrieve the number of Managers who purchase Nike products....24

Retrieve the names of the employee who purchase more than two items....25

Retrieve the items that are sold to two employees....25

The total discount that is given....26

Challenge 3....26

1.11. SQL Query to merge all tables into one to be exported:....26

1.12. Python Code....27

References:....40

Challenge 1:

1.1. User Definition

In this section I will define who will use the database and how the database will be used. Firstly, the main users of this database will be Customer Advisors, Supervisors, Managers, the CEO, HR, and any other relevant employees. However, others will not be specified in the database. Branch Managers, the CEO and HR will have access to all functions of the database as database administrators. The Duty Managers will have the ability view and edit all tables in the database while supervisors and customer advisors will have limited access to the database, with their privileges only allowing 'INSERT' SQL function into certain tables. No user will be able to access the 'AUDIT' table. The Supervisors and Customer Advisors will be defined as 'End users' as their job requires access to query and update the database. Customer advisors will be 'Parametric End Users' and supervisors will be 'Casual end users' as they may require different information each time of use (Elmasri and Navathe, 2016). The casual end users and parametric end users will be inserting and updating records in the transactions and product tables. This database will have the functionality to include multiple shops and branches. E.g. it could contain branches from Sports Redirect and JS Sports along with their corresponding branches.

1.2. Components and Architecture

In this section I will define the design of database along with the components used and how is connected. For this assignment I will be using a MySQL database hosted locally on my laptop, however in the real world this would be hosted on a server. A user interface will be developed in Python and PyQT5. Users will need hardware capable of connecting to the company intranet and running an executable application to access the database. It will be split into seven tables where six of them are linked with foreign keys (Harrington, 2016a). The database will only involve employee details and employee transactions. The data required will be as follows:

Jobs Table

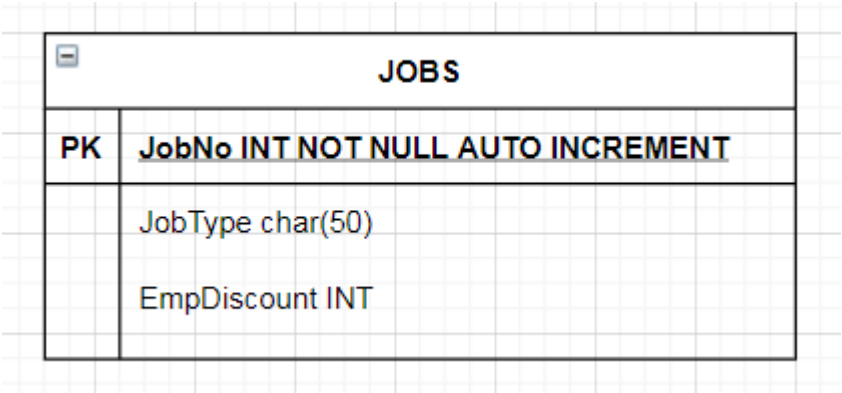

Figure 1: Job Table

The 'jobs' table will have a PRIMARY KEY 'JobNo' stored in the form of an integer and will store the 'JobType' and their corresponding employee discount. It will be used to keep track of different job roles within the company.

Branch Table

BRANCH	
PK	**BranchNo INT NOT NULL**
FK1	ShopNo INT NOT NULL
	BranchLocation char(50)

Figure 2: Branch Table

The 'branch' table will contain a foreign key linked to which shop it is. It will also contain the location of the branch. It will be used to distinguish between branches in a chain.

Shop Table

SHOP	
PK	**ShopNo INT NOT NULL**
	ShopName char(20) NOT NULL
	ShopAddress char(50) NOT NULL

Figure 3: Shop Table

The 'shop' table will define which shop it is using 'ShopNo' as the primary key. 'ShopNo' will be defined as a foreign key in the branch table. It will be used to distinguish between shop chains.

Employee Table

EMPLOYEE	
PK	EmpNo INT NOT NULL AUTO INCREMENT
FK1	FName char(15) NOT NULL
	SName char(20) NOT NULL
	PhoneNo char(15) NOT NULL
	EmailAddr char(50) NOT NULL
	DOB DATE NOT NULL
FK1	JobNo INT NOT NULL
FK2	BranchNo INT NOT NULL

Figure 4: Employee Table

Figure 4 shows the planned implementation for an employee table with 1 PRIMARY KEY and 2 FOREIGN KEYS linked to tables stated in Figure 1 & 2. It will be used to track the employees in the company, their job role, where they work and how to contact them.

Transactions Table

TRANSACTIONS	
PK	TransNo INT NOT NULL AUTO INCREMENT
	DelAddress char(200) NOT NULL
	PurchaseDate DATE NOT NULL
FK1	ProductNo INT NOT NULL
FK2	EmpNo INT NOT NULL

Figure 5: Transactions Table

Figure 5 shows the implementation for the transactions table using a PRIMARY KEY and 2 FOREIGN KEYS linked to tables stated above in Figure 4 and below in Figure 6. It will be used to track all purchases made by employees.

Product Table

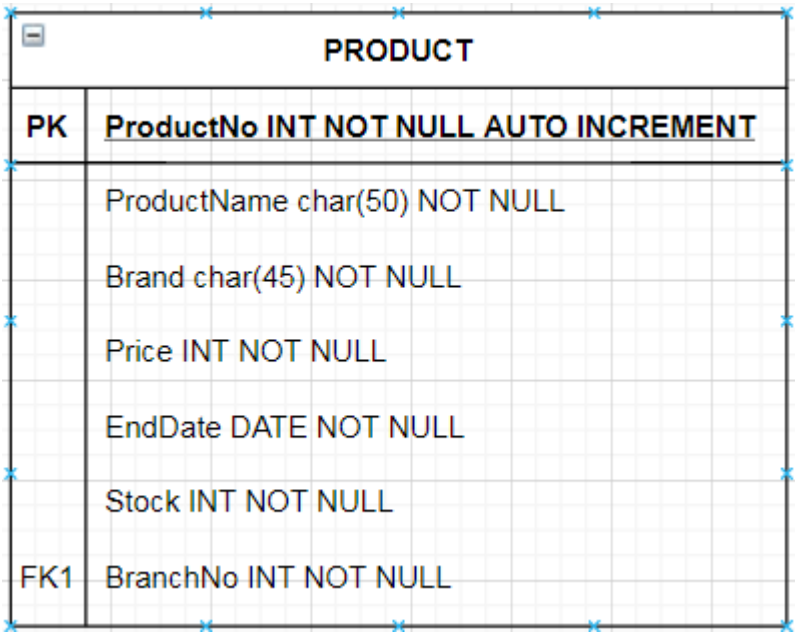

Figure 6: Product Table

Figure 6 shows how the product table will be defined using a FOREIGN KEY linked to the branch table to show where this product is stored. It will be used to track the type of products each store has, how much stock there is, what they are and how much they cost.

Audit Table

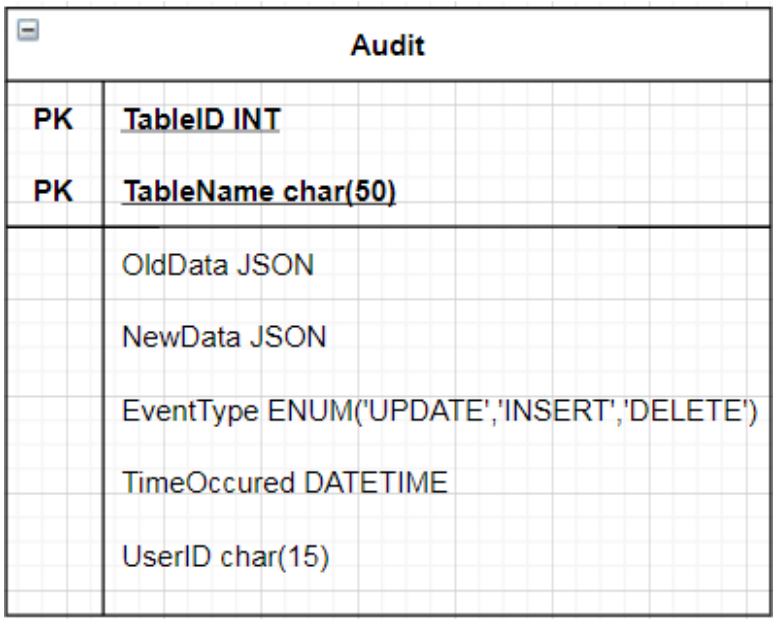

Figure 7: Audit Table

Figure 7 shows how the audit table will be defined. It will be used to store records of all database transactions in the employee and transactions table to keep track of what is added, updated, and deleted.

Keys

Each table will have one primary key except for the Audit table which has a composite key of TableID and TableName and TimeOccured. Each table, except for the audit table, will have either a foreign key linked to it or its own primary key used as a foreign key in another table. This allows the database administrators to design the database is such a way that there is no data redundancy and allows for normalisation into 3rd normal form. This, in turn, will remove any many to many relationships leaving only one to many relations.

EER Diagram

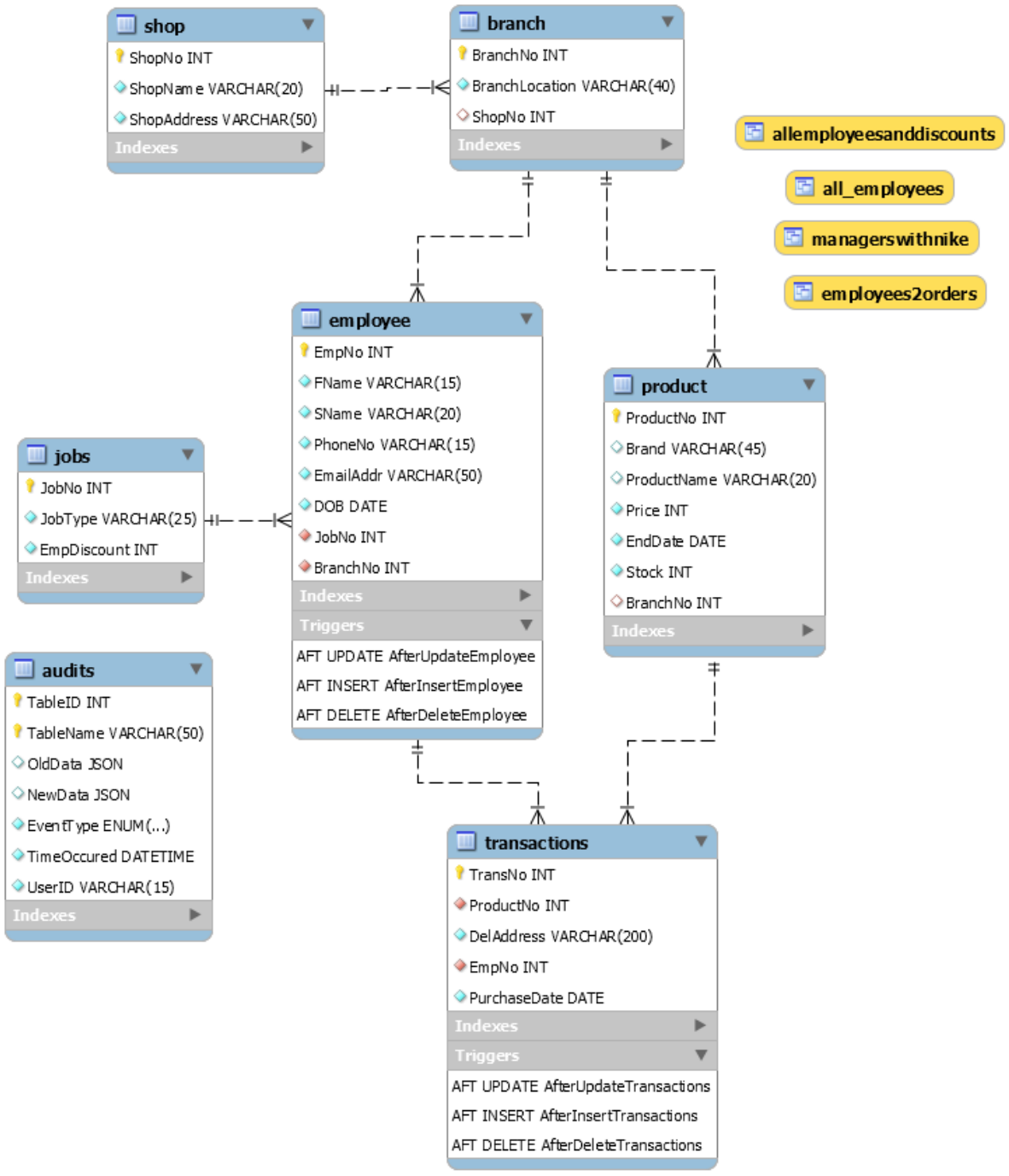

Figure 8: EER Diagram with relationships

In figure 8 I have illustrated my database in the form of an EER diagram. The relationships shown are only one to many. This meets the requirements for 3rd normal form as the attributes are solely dependent on the primary key. Using Foreign keys will also enforce referential integrity as the only values they can contain are null or values from a parent table's primary key. This could also be enforced by using a trigger.

1.3. Business Process

The business process followed in this database only involves employees and products, it will not involve external customers, however employees can be seen as customers when purchasing a product. Firstly, an employee will have an account created for them by either a branch manager or HR. An employee can then purchase a product or sell a product to another employee. When the product is purchased the stock level should be decreased by one for that product. The transaction would be created and then either shipped to an address or collected by the customer. Transactions records can be stored and then queried to keep track of which employees have made purchases.

1.4. Advantages

Firstly, a big advantage of a computerized database is that it makes it easier to enter, store and search for data. Databases allow automation of data storage and manipulation. If you invest time in designing and configuring a database, you can save money and time in the future. Organising of data in a database makes information more readily accessible and users can simply enter queries to find a record instead of searching 2000 square feet of a library (Hill, 2020).

1.5. Disadvantages

There are many disadvantages of using a computer-based database. Firstly, lots of costs are incurred, both excepted and unexpected. The hardware and infrastructure needed to run a database a quite costly and once the company has implemented a database, they need to spend time training employees on company time or paying for instructors to teach them. Both models will bring a significant cost to the business (Harrington, 2016b).

Challenge 2:

2.1. Creating Database & Tables

```
-- Creating Database
CREATE database sportsredirect use sportsredirect
-- Creating Shop Table
CREATE TABLE shop(
      ShopNo INT NOT NULL PRIMARY key AUTO_INCREMENT UNIQUE,
      ShopName VARCHAR(20) NOT NULL,
      ShopAddress VARCHAR(50) NOT NULL)
-- Creating Branch Table
```

```
CREATE TABLE branch(
      BranchNo INT NOT NULL PRIMARY key AUTO_INCREMENT UNIQUE,
      BranchLocation VARCHAR(40) NOT NULL,
      ShopNo INT NOT NULL,
      FOREIGN KEY (ShopNo) REFERENCES shop(ShopNo))
-- Creating Job Table
CREATE TABLE jobs(
      JobNo INT NOT NULL PRIMARY key AUTO_INCREMENT UNIQUE,
      JobType VARCHAR(25) NOT NULL,
      EmpDiscount INT NOT NULL)
-- Creating Employee Table
CREATE TABLE employee(
      EmpNo INT NOT NULL PRIMARY key AUTO_INCREMENT UNIQUE,
      FName VARCHAR(15) NOT NULL,
      SName VARCHAR(20) NOT NULL,
      PhoneNo VARCHAR(15) NOT NULL,
      EmailAddr VARCHAR(50) NOT NULL,
      DOB DATE NOT NULL,
      JobNo INT NOT NULL,
      BranchNo INT NOT NULL,
      FOREIGN KEY (JobNo) REFERENCES jobs(JobNo),
      FOREIGN KEY (BranchNo) REFERENCES Branch(BranchNo))
-- Creating Product Table
CREATE TABLE product(
      ProductNo INT NOT NULL PRIMARY key AUTO_INCREMENT UNIQUE,
      Brand VARCHAR(20) NOT NULL,
      ProductName VARCHAR(20) NOT NULL,
      Price INT NOT NULL,
      Stock INT NOT NULL,
      EndDate DATE NOT NULL,
      BranchNo INT NOT NULL,
      FOREIGN KEY (BranchNo) REFERENCES Branch(BranchNo))
```

```
-- Creating Transactions Table
CREATE TABLE transactions(
      TransNo INT NOT NULL PRIMARY key AUTO_INCREMENT UNIQUE,
      EmpNo INT NOT NULL,
      ProductNo INT NOT NULL,
      DelAddress VARCHAR(50) NOT NULL,
      PurchaseDate DATE NOT NULL,
      FOREIGN KEY (ProductNo) REFERENCES product(ProductNo),
      FOREIGN KEY (EmpNo) REFERENCES employee(EmpNo))
-- Creating Audit Table
CREATE TABLE audits(
      TableID INT NOT NULL,
      TableName VARCHAR(50) NOT NULL,
      OldData JSON,
      NewData JSON,
      EventType ENUM('INSERT', 'UPDATE', 'DELETE') NOT NULL,
      TimeOccured DATETIME NOT NULL,
      UserID VARCHAR(15) NOT NULL,
      PRIMARY KEY (TableID, TableName, TimeOccured))
-- Creating some logins to be used for testing
CREATE USER 'HR'@'%' IDENTIFIED BY 'HR101';
CREATE USER 'CEO'@'%' IDENTIFIED BY 'CEOIsBest';
CREATE USER 'FrankLampard'@'%' IDENTIFIED BY 'Lampard';
CREATE USER 'JackYoung'@'%' IDENTIFIED BY 'Young';
-- Granting Privileges - the employees that aren't managers don't
get access to anything other than product and transactions and their
own record in employee
GRANT SELECT , INSERT, DELETE, UPDATE ON sportredirect.product TO
'FrankLampard'@'localhost';
GRANT SELECT , INSERT ON sportredirect.transactions TO
'FrankLampard'@'localhost';
GRANT SELECT , UPDATE ON sportredirect.employee TO
'FrankLampard'@'localhost';
-- Granting Privileges to super users
```

```
GRANT ALL PRIVILEGES ON *.* TO 'CEO'@'localhost';
GRANT ALL PRIVILEGES ON *.* TO 'HR'@'localhost' WITH GRANT OPTION;
GRANT ALL PRIVILEGES ON *.* TO 'JackYoung'@'localhost';
```

2.2. Inserting Data

```
-- Inserting shop data
INSERT INTO shop (ShopName, ShopAddress)
VALUES
      ( 'Sports Redirect', 'London' )
-- Inserting branch data
INSERT INTO branch (BranchLocation, ShopNo)
VALUES
      ( 'London', 1 ) ,
      ( 'Cheltenham', 1 ) ,
      ( 'Leamington Spa', 1 ) ,
      ( 'Bath Spa', 1 ) ,
      ( 'Aberdeen', 1 ) ,
      ( 'Birmingham', 1 ) ,
      ( 'Sheffield', '1' ) ,
      ( 'Cirencester', '1' ) ,
      ( 'Warwick', '1' ) ,
      ('Liverpool', '1');
-- Inserting job data
INSERT INTO jobs (JobType, EmpDiscount)
VALUES
      ( "CEO", 20 ) ,
      ( "Branch Manager", 20 ) ,
      ( "Duty Manager", 20 ) ,
      ( "Supervisor", 10 ) ,
      ( "Customer Advisor", 10 );
-- Inserting employee data
INSERT INTO employee (FName, SName, PhoneNo, EmailAddr, DOB, JobNo,
BranchNo)
```

```
VALUES
      ( 'Fred', 'Smith', '07563 326610', 'fred.smith@hotmail.com',
'1996/10/01', 7, 6 ) ,
      ( 'John', 'Legend', '07821 806490', 'john.legend@hotmail.com',
'1989/10/31', 10, 2 ) ,
      ( 'Michael', 'Buble', '07533 2164430',
'michael.buble@gmail.com', '1992/07/01', 9, 2 ) ,
      ( 'Robert', 'Williams', '07493 406630',
'robbie.williams@hotmail.com', '1998/05/28', 9, 3 ) ,
      ( 'Jack', 'Young', '07993 606630', 'jaxkyoung@icloud.com',
'2000/11/18', 6, 1 ) ,
      ( 'Jianhua', 'Yang', '07452 685734',
'jianhua.yang@warwick.ac.uk', '1970/03/12', 8, 3 ) ,
      ( 'Brendon', 'Urie', '07678 109674',
'brendon.urie@hotmail.com', '1978/04/15', 10, 5 ) ,
      ( 'Thierry', 'Henry', '07286 967384',
'Thierry.henry@hotmail.com', '1998/02/07', 10, 5 ) ,
      ( 'John', 'Terry', '07986 896732', 'john.terry@chelseafc.com',
'1999/10/30', 7, 3 ) ,
      ( 'Frank', 'Lampard', '07778 675433',
'frank.lampard@yahoo.com', '1998/12/02', 10, 2 ) ,
      ( 'Steve', 'Gerrard', '07993 407754', 'stevieg@liverpool.com',
'1967/09/10', 10, 4 ) ,
      ( 'Ed', 'Sheeran', '07932 230789', 'ed@sheeran.me', '1989-10-
12', 10, 4 ) ,
      ( 'Freddie', 'Mercury', '07493 231890', 'fred@queen.co.uk',
'1956-04-30', 10, 1 ) ;



-- Inserting Product data
INSERT INTO product (Brand, ProductName, Price, Stock, EndDate)
VALUES
      ( 'Nike', 'Air Force Trainers', 100, 10, '2019/10/01' ) ,
      ( 'Reebok', 'Blue T Shirt', 65, 22, '2020/05/25' ) ,
      ( 'Adidas', 'Trainers', 32, 5, '2020/10/15' ) ,
      ( 'Tommy Hilfiger', 'White T Shirt', 100, 1, '2020/09/10' ) ,
      ( 'Nike', 'Boxers', 10, 2, '2018/01/01' ) ,
```

```
    ( 'Adidas', 'Sports Shorts', 200, 7, '2019/10/01' ) ,
    ( 'Reebok', 'Socks', 3, 9, '2017/10/01' ) ,
    ( 'Nike', 'Necklace', 5, 3, '2016/12/30' ) ,
    ( 'Tommy Jeans', 'Ripped Jeans', 125, 7, '2015/03/10' ) ,
    ( 'Vans', 'Air Force Trainers', 119, 8, '2020/11/18' ) ,
    ( 'Nike', 'SB Trainers', 87, 10, '2014/06/12' ) ,
    ( 'Gul', 'Wetsuit', 225, 5, '2014/02/12' ) ;

-- Inserting Transaction Data
INSERT INTO transactions (`ProductNo`, `DelAddress`, `EmpNo`,
`PurchaseDate`)
VALUES
    ( '12', '157 Prestbury Road', '1', '2020/10/10' ) ,
    ( '10', '50 Chatham Avenue', '2', '2020/05/01' ) ,
    ( '5', '9 Southbourne', '1', '2018/05/01' ) ,
    ( '6', '2 Warwick Street', '10', '2020/07/15' ) ,
    ( '6', '7 Oxford Street', '5', '2020/02/17' ) ,
    ( '11', '27 Bromley Common', '2', '2020/10/15' ) ,
    ( '4', '22 Regent Street', '8', '2020/03/30' ) ,
    ( '9', '31 Parade', '6', '2019/12/25' ) ,
    ( '2', '89 Gloucester Road', '8', '2019/10/30' ) ,
    ( '1', '25 Pitville Crescent', '4', '2017/04/16' ) ,
    ( '7', '1 Connor Place', '9', '2020/09/29' ) ,
    ( '3', '12 Gaydon Road', '12', '2019-12-13' ) ,
    ( '4', '12 Gaydon Road', '12', '2019-12-14' ) ;
```

2.3. Triggers & Procedures

```
-- Creating a trigger to log all transactions in the employee table
that will then insert into my audit table that is no accessible by
any employees except CEO and high-level management
CREATE TRIGGER AfterDeleteEmployee
    AFTER DELETE
    ON employee FOR EACH ROW
```

```
BEGIN
	DECLARE currentdate DATETIME;
	DECLARE currentuser VARCHAR(15);
	select sysdate(), CURRENT_USER() into currentdate, currentuser;
	INSERT INTO audits (TableID, TableName, OldData, NewData,
EventType, TimeOccured, UserID)
	VALUES (
		OLD.EmpNo,
		'Employee',
		JSON_OBJECT(
			"Emp ID", OLD.EmpNo,
			"First Name", OLD.FName,
			"Last Name", OLD.SName,
			"Phone No", OLD.PhoneNo,
			"Email", OLD.EmailAddr,
			"DOB", OLD.DOB,
			"Job No", OLD.JobNo ),
		null,
		'UPDATE',
		currentdate,
		currentuser
	) ;
END


CREATE TRIGGER AfterInsertEmployee
	AFTER INSERT
	ON employee FOR EACH ROW
BEGIN
	DECLARE currentdate DATETIME;
	DECLARE currentuser VARCHAR(15);
	select sysdate(), CURRENT_USER() into currentdate, currentuser;
	INSERT INTO audits (TableID, TableName, OldData, NewData,
EventType, TimeOccured, UserID)
```

```
	VALUES (
		NEW.EmpNo,
		'Employee',
		null,
		JSON_OBJECT(
			"Emp ID", NEW.EmpNo,
			"First Name", NEW.FName,
			"Last Name", NEW.SName,
			"Phone No", NEW.PhoneNo,
			"Email", NEW.EmailAddr,
			"DOB", NEW.DOB,
			"Job No", NEW.JobNo ),
		'INSERT',
		currentdate,
		currentuser
	) ;
END

CREATE TRIGGER AfterUpdateEmployee
	AFTER UPDATE
	ON employee FOR EACH ROW
BEGIN
	DECLARE currentdate DATETIME;
	DECLARE currentuser VARCHAR(15);
	select sysdate(), CURRENT_USER() into currentdate, currentuser;
	INSERT INTO audits (TableID, TableName, OldData, NewData,
EventType, TimeOccured, UserID)
	VALUES (
		NEW.EmpNo,
		'Employee',
		JSON_OBJECT(
			"Emp ID", OLD.EmpNo,
			"First Name", OLD.FName,
```

```
                "Last Name", OLD.SName,
                "Phone No", OLD.PhoneNo,
                "Email", OLD.EmailAddr,
                "DOB", OLD.DOB,
                "Job No", OLD.JobNo ),
            JSON_OBJECT(
                "Emp ID", NEW.EmpNo,
                "First Name", NEW.FName,
                "Last Name", NEW.SName,
                "Phone No", NEW.PhoneNo,
                "Email", NEW.EmailAddr,
                "DOB", NEW.DOB,
                "Job No", NEW.JobNo ),
            'UPDATE',
            currentdate,
            currentuser
      ) ;
END


-- Creating a triggers to log all transactions in the transactions table that will then insert into my audit table that is no accessible by any employees except CEO and high level management
CREATE TRIGGER AfterDeleteTransactions
      AFTER DELETE
      ON transactions FOR EACH ROW
BEGIN
      DECLARE currentdate DATETIME;
      DECLARE currentuser VARCHAR(15);
      select sysdate(), CURRENT_USER() into currentdate, currentuser;
      INSERT INTO audits (TableID, TableName, OldData, NewData, EventType, TimeOccured, UserID)
      VALUES (
            NEW.TransNo,
            'Transactions',
```

```
            null,
            JSON_OBJECT(
                  "Trans ID", NEW.TransNo,
                  "Product No", NEW.ProductNo,
                  "Delivery Address", NEW.DelAddress,
                  "EmpNo", NEW.EmpNo,
                  "Purchase Data", NEW.PurchaseDate ),
            'INSERT',
            currentdate,
            currentuser
      ) ;
END

CREATE TRIGGER AfterInsertTransactions
      AFTER DELETE
      ON transactions FOR EACH ROW
BEGIN
      DECLARE currentdate DATETIME;
      DECLARE currentuser VARCHAR(15);
      select sysdate(), CURRENT_USER() into currentdate, currentuser;
      INSERT INTO audits (TableID, TableName, OldData, NewData,
      EventType, TimeOccured, UserID)
      VALUES (
            NEW.TransNo,
            'Transactions',
            null,
            JSON_OBJECT(
                  "Trans ID", NEW.TransNo,
                  "Product No", NEW.ProductNo,
                  "Delivery Address", NEW.DelAddress,
                  "EmpNo", NEW.EmpNo,
                  "Purchase Data", NEW.PurchaseDate ),
            'INSERT',
```

```
            currentdate,
            currentuser
      ) ;
END

CREATE TRIGGER AfterUpdateTransactions
      AFTER DELETE
      ON transactions FOR EACH ROW
BEGIN
      DECLARE currentdate DATETIME;
      DECLARE currentuser VARCHAR(15);
      select sysdate(), CURRENT_USER() into currentdate, currentuser;
      INSERT INTO audits (TableID, TableName, OldData, NewData,
      EventType, TimeOccured, UserID)
      VALUES (
            NEW.TransNo,
            'Transactions',
            JSON_OBJECT(
                  "Trans ID", OLD.TransNo,
                  "Product No", OLD.ProductNo,
                  "Delivery Address", OLD.DelAddress,
                  "EmpNo", OLD.EmpNo,
                  "Purchase Data", OLD.PurchaseDate
            ),
            JSON_OBJECT(
                  "Trans ID", NEW.TransNo,
                  "Product No", NEW.ProductNo,
                  "Delivery Address", NEW.DelAddress,
                  "EmpNo", NEW.EmpNo,
                  "Purchase Data", NEW.PurchaseDate
            ),
            'UPDATE',
            currentdate,
```

```
            currentuser
      ) ;
END


-- Procedure called when employee permissions need to be checked
CREATE PROCEDURE errorCheckEmpPermissions ()
BEGIN
      IF current_user() <> 'CEO@localhost' or current_user() <>
      'HR@localhost' or current_user() <> 'JackYoung@localhost' or
      current_user() <> concat(FName, SName)
      then
      signal sqlstate '45000' set message_text = 'You do not have
      access to this!'; end if;
END
-- Triggers to prevent access to employees who aren't allowed access
CREATE TRIGGER PreventDeleteTrigger_Employee
      BEFORE DELETE
      ON employee FOR EACH ROW
BEGIN
      call errorCheckEmpPermissions();
END


CREATE TRIGGER PreventChangeTrigger_Employee
      BEFORE UPDATE
      ON employee FOR EACH ROW
BEGIN
      call errorCheckEmpPermissions();
END


-- Trigger to stop employees from inserting data
CREATE TRIGGER PreventInsertTrigger_Employee
      BEFORE INSERT
      ON employee FOR EACH ROW
BEGIN
```

```
      IF current_user() <> 'CEO@localhost'
      or current_user() <> 'HR@localhost'
      or current_user() <> 'JackYoung@localhost'
      then
      signal sqlstate '45000' set message_text = 'My Error Message';
      end if;
END
```

2.4. Views

```
-- Create view to show all employees and more relevant information
CREATE VIEW All_Employees AS
      SELECT
            employee.EmpNo AS 'Employee ID',
            employee.FName AS 'First Name',
            employee.SName AS 'Surname',
            jobs.JobType AS 'Job',
            branch.BranchLocation AS 'Location'
      FROM
            employee
      INNER JOIN jobs ON employee.JobNo = jobs.JobNo
      INNER JOIN branch ON employee.BranchNo = branch.BranchNo;

-- Create view to show all employees and their discounts
CREATE view AllEmployeesAndDiscounts AS
      SELECT
            employee.EmpNo,
            employee.SName AS Surname,
            jobs.JobType AS Job,
            concat(jobs.EmpDiscount, '%') AS Discount
      FROM
            employee
      INNER JOIN jobs ON employee.JobNo = jobs.JobNo
```

```sql
-- Create view to show all managers that have purchased nike products
CREATE View ManagersWithNike AS
	SELECT
		product.ProductNo,
		product.Brand,
		transactions.EmpNo,
		jobs.JobType
	FROM
		product
	INNER JOIN transactions ON product.ProductNo = transactions.ProductNo
	INNER JOIN employee ON transactions.EmpNo = employee.EmpNo
	INNER JOIN jobs ON employee.JobNo = jobs.JobNo
	WHERE product.Brand = 'Nike' AND employee.JobNo < 9



-- Create view to show all transactions and their prices discounted and user-friendly information
CREATE VIEW TransactionsWithDiscountPrice AS
	SELECT
		transactions.TransNo,
		transactions.PurchaseDate,
		employee.SName,
		product.ProductName,
		product.Price,
	CASE
		WHEN TIMESTAMPDIFF(MONTH, product.EndDate, transactions.PurchaseDate) >= 18 THEN CONCAT(jobs.EmpDiscount + 10, '%')
		WHEN TIMESTAMPDIFF(MONTH, product.EndDate, transactions.PurchaseDate) < 18 THEN CONCAT(jobs.EmpDiscount, '%')
	END AS Discount,
```

```
CASE
      WHEN TIMESTAMPDIFF(MONTH, product.EndDate,
      transactions.PurchaseDate) >= 18 THEN product.Price -
      (product.Price * ((jobs.EmpDiscount*0.01) + 0.1))
      WHEN TIMESTAMPDIFF(MONTH, product.EndDate,
      transactions.PurchaseDate) < 18 THEN product.Price -
      (product.Price * (jobs.EmpDiscount*0.01))
END AS DiscountedPrice
FROM
      transactions
INNER JOIN product ON transactions.ProductNo = product.ProductNo
INNER JOIN employee ON transactions.EmpNo = employee.EmpNo
INNER JOIN jobs ON employee.JobNo = jobs.JobNo
```

2.5. Challenge Query Proof

The discount that is given to all employees.

```
SELECT
      transactions.TransNo,
      transactions.PurchaseDate,
      employee.SName,
      product.ProductName,
      product.Price,
CASE
      WHEN TIMESTAMPDIFF(MONTH, product.EndDate,
      transactions.PurchaseDate) >= 18 THEN
      CONCAT(jobs.EmpDiscount + 10, '%')
      WHEN TIMESTAMPDIFF(MONTH, product.EndDate,
      transactions.PurchaseDate) < 18 THEN
      CONCAT(jobs.EmpDiscount, '%')
END AS Discount,
CASE
      WHEN TIMESTAMPDIFF(MONTH, product.EndDate,
      transactions.PurchaseDate) >= 18 THEN product.Price -
      (product.Price * ((jobs.EmpDiscount*0.01) + 0.1))
```

```
        WHEN TIMESTAMPDIFF(MONTH, product.EndDate,
        transactions.PurchaseDate) < 18 THEN product.Price -
        (product.Price * (jobs.EmpDiscount*0.01))
    END AS DiscountedPrice
    FROM
        transactions
    INNER JOIN product ON transactions.ProductNo = product.ProductNo
    INNER JOIN employee ON transactions.EmpNo = employee.EmpNo
    INNER JOIN jobs ON employee.JobNo = jobs.JobNo
```

Trans No	Purchase Date	Surname	Product Name	Price	Discount	Discounted Price
1	2020-10-10	Smith	Wetsuit	225.00	30%	157.50
3	2018-05-01	Smith	Boxers	10.00	20%	8.00
2	2020-05-01	Legend	Air Force Trainers	119.00	10%	107.10
6	2020-10-15	Legend	SB Trainers	87.00	20%	69.60
10	2017-04-16	Williams	Air Force Trainers	100.00	10%	90.00
5	2020-02-17	Young	Sports Shorts	200.00	20%	160.00
14	2020-10-27	Young	Air Force Trainers	119.00	20%	95.20
8	2019-12-25	Yang	Ripped Jeans	125.00	30%	87.50
7	2020-03-30	Henry	White T Shirt	100.00	10%	90.00
9	2019-10-30	Henry	Blue T Shirt	65.00	10%	58.50
11	2020-09-29	Terry	Socks	3.00	30%	2.10
4	2020-07-15	Lampard	Sports Shorts	200.00	10%	180.00
12	2019-12-13	Sheeran	Trainers	32.00	10%	28.80
13	2019-12-14	Sheeran	White T Shirt	100.00	10%	90.00

Retrieve the number of Managers who purchase Nike products

```
    SELECT
        product.ProductNo,
        product.Brand,
        transactions.EmpNo,
        jobs.JobType
    FROM
        product
    INNER JOIN transactions ON product.ProductNo =
    transactions.ProductNo
    INNER JOIN employee ON transactions.EmpNo = employee.EmpNo
    INNER JOIN jobs ON employee.JobNo = jobs.JobNo
    WHERE product.Brand = 'Nike' AND employee.JobNo < 9
```

ProductNo	Brand	EmpNo	JobType
Filter...	Filter...	Filter...	Filter...
5	Nike	1	Branch Manager

Retrieve the names of the employee who purchase more than two items

```
SELECT
	FName,
	SName,
	COUNT(0) AS `Number OF Orders`
FROM
	transactions
INNER JOIN employee ON employee.EmpNo = transactions.EmpNo
GROUP BY employee.FName
HAVING (COUNT(0) > 1)
```

FName	SName	Number of Orders
Filter...	Filter...	Filter...
Fred	Smith	2
John	Legend	3
Jack	Young	2
Thierry	Henry	2
Ed	Sheeran	2

Retrieve the items that are sold to two employees

```
SELECT
	employee.EmpNo,
	employee.SName,
	transactions.TransNo,
	product.ProductName,
```

```
      product.Price
FROM
      transactions
INNER JOIN employee ON employee.EmpNo = transactions.EmpNo
INNER JOIN product ON product.ProductNo = transactions.ProductNo
WHERE employee.EmpNo = 1 OR employee.EmpNo = 5;
```

EmpNo	SName	TransNo	ProductName	Price
1	Smith	1	Wetsuit	225.00
1	Smith	3	Boxers	10.00
5	Young	5	Sports Shorts	200.00
5	Young	14	Air Force Trainers	119.00

The total discount that is given

- This code selects the sum of price and discounted price and works out the discount given to all employees and all transactions.

```
SELECT format((SUM(price) - SUM(discountedprice)), 2) AS 'Money Lost'
FROM transactionswithdiscountprice
```

Money Lost
260.70

Challenge 3

3.1. SQL Query to merge all tables into one to be exported:

```
SELECT TransNo, DelAddress, PurchaseDate, product.ProductNo,
product.Brand, product.ProductName, product.Price, product.EndDate,
product.Stock, employee.EmpNo, employee.FName, employee.SName,
employee.PhoneNo, employee.EmailAddr, employee.DOB, jobs.JobNo,
jobs.JobType, jobs.EmpDiscount, branch.BranchNo, BranchLocation
FROM transactions
LEFT JOIN product ON transactions.ProductNo = product.ProductNo
LEFT JOIN employee ON employee.EmpNo = transactions.EmpNo
LEFT JOIN jobs ON employee.JobNo = jobs.JobNo
LEFT JOIN branch ON employee.BranchNo = branch.BranchNo
```

```
UNION

SELECT TransNo, DelAddress, PurchaseDate, product.ProductNo,
product.Brand, product.ProductName, product.Price, product.EndDate,
product.Stock, employee.EmpNo, employee.FName, employee.SName,
employee.PhoneNo, employee.EmailAddr, employee.DOB, jobs.JobNo,
jobs.JobType, jobs.EmpDiscount, branch.BranchNo, BranchLocation

FROM transactions

RIGHT JOIN product ON transactions.ProductNo = product.ProductNo

RIGHT JOIN employee ON employee.EmpNo = transactions.EmpNo

RIGHT JOIN jobs ON employee.JobNo = jobs.JobNo

RIGHT JOIN branch ON employee.BranchNo = branch.BranchNo
```

- The code shown above will be used in python and passed to a cursor to execute for exporting all tables.

3.2. Python Code

```
'''
This program is created in Python 3.8 and is dependent on PyQt5
being installed via pip.

Requirements:
    PyQt5 Python Library
    MySQL Connector Python Library
    Python 3.8
    MySQL Server with legacy authentication hosted on localhost
    Server must be configured as per Challenge 2 SQL Queries

Author: U191983
'''

# importing GUI library
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication, QTableWidget,
QTableWidgetItem, QFileDialog
# library to connect to mysql database
import mysql.connector
# library to export to csv
import csv

# global variables so all functions can connect to database
global conn
global cur
conn = 0
cur = 0
```

```
# this class contains all functions which directly affect the
appearance of my application and the setup code for the PyQt gui.
class Ui_DatabaseGUI(object):

    # function to setup gui
    def setupUi(self, DatabaseGUI):
        DatabaseGUI.setObjectName("DatabaseGUI")
        DatabaseGUI.resize(660, 540)
        # setting widgets on tab 1 to their location and type
        self.centralwidget = QtWidgets.QWidget(DatabaseGUI)
        self.centralwidget.setObjectName("centralwidget")
        self.tabWidget = QtWidgets.QTabWidget(self.centralwidget)
        self.tabWidget.setGeometry(QtCore.QRect(20, 10, 631, 481))
        self.tabWidget.setObjectName("tabWidget")
        self.queryTab = QtWidgets.QWidget()
        self.queryTab.setObjectName("queryTab")
        self.tableView = QtWidgets.QTableWidget(self.queryTab)
        self.tableView.setGeometry(QtCore.QRect(26, 90, 581, 351))
        self.tableView.setObjectName("tableView")
        self.queryViewButton = QtWidgets.QPushButton(self.queryTab)
        self.queryViewButton.setGeometry(QtCore.QRect(70, 40, 93,
28))
        self.queryViewButton.setObjectName("queryViewButton")

self.queryViewButton.clicked.connect(preMadeQueryTab.queryViewButton
Pressed)
        self.viewsComboBox = QtWidgets.QComboBox(self.queryTab)
        self.viewsComboBox.setGeometry(QtCore.QRect(10, 10, 191,
22))
        self.viewsComboBox.setObjectName("viewsComboBox")
        # adding items to drop down box
        self.viewsComboBox.addItem("")
        self.viewsComboBox.addItem("")
        self.viewsComboBox.addItem("")
        self.viewsComboBox.addItem("")
        self.viewsComboBox.addItem("")
        self.queryTableButton = QtWidgets.QPushButton(self.queryTab)
        self.queryTableButton.setGeometry(QtCore.QRect(260, 40, 93,
28))
        self.queryTableButton.setObjectName("queryTableButton")

self.queryTableButton.clicked.connect(preMadeQueryTab.queryTableButt
onPressed)
        self.viewTableComboBox = QtWidgets.QComboBox(self.queryTab)
        self.viewTableComboBox.setGeometry(QtCore.QRect(250, 10,
121, 22))
        self.viewTableComboBox.setObjectName("viewTableComboBox")
        # adding items to drop down box
        self.viewTableComboBox.addItem("")
        self.viewTableComboBox.addItem("")
        self.viewTableComboBox.addItem("")
        self.viewTableComboBox.addItem("")
        self.viewTableComboBox.addItem("")
        self.viewTableComboBox.addItem("")
        self.viewTableComboBox.addItem("")
        self.tabWidget.addTab(self.queryTab, "")
```

```
        # setting up widgets in tab 2
        self.exportTab = QtWidgets.QWidget()
        self.exportTab.setObjectName("exportTab")
        self.exportTableCombo = QtWidgets.QComboBox(self.exportTab)
        self.exportTableCombo.setGeometry(QtCore.QRect(140, 30, 121,
22))
        self.exportTableCombo.setObjectName("exportTableCombo")
        # adding items to drop down box
        self.exportTableCombo.addItem("")
        self.exportTableCombo.addItem("")
        self.exportTableCombo.addItem("")
        self.exportTableCombo.addItem("")
        self.exportTableCombo.addItem("")
        self.exportTableCombo.addItem("")
        self.exportTableCombo.addItem("")
        self.exportTableCombo.addItem("")
        self.pathToSaveEditBox = QtWidgets.QLineEdit(self.exportTab)
        self.pathToSaveEditBox.setGeometry(QtCore.QRect(120, 77,
181, 22))
        self.pathToSaveEditBox.setObjectName("pathToSaveEditBox")
        self.exportButton = QtWidgets.QPushButton(self.exportTab)
        self.exportButton.setGeometry(QtCore.QRect(150, 120, 93,
28))
        self.exportButton.setObjectName("exportButton")

self.exportButton.clicked.connect(exportTab.exportButtonPressed)
        self.selectTableLabel = QtWidgets.QLabel(self.exportTab)
        self.selectTableLabel.setGeometry(QtCore.QRect(20, 30, 81,
16))
        self.selectTableLabel.setObjectName("selectTableLabel")
        self.pathsSaveLabel = QtWidgets.QLabel(self.exportTab)
        self.pathsSaveLabel.setGeometry(QtCore.QRect(20, 80, 81,
16))
        self.pathsSaveLabel.setObjectName("pathsSaveLabel")
        self.selectPathButton =
QtWidgets.QPushButton(self.exportTab)
        self.selectPathButton.setGeometry(QtCore.QRect(300, 77, 21,
21))
        self.selectPathButton.setObjectName("selectPathButton")

self.selectPathButton.clicked.connect(exportTab.selectPathButtonPres
sed)
        self.tabWidget.addTab(self.exportTab, "")
        # setting up widgets in tab 3
        self.loginTab = QtWidgets.QWidget()
        self.loginTab.setObjectName("loginTab")
        self.userInput = QtWidgets.QLineEdit(self.loginTab)
        self.userInput.setGeometry(QtCore.QRect(90, 20, 113, 22))
        self.userInput.setObjectName("userInput")
        self.pwdInput = QtWidgets.QLineEdit(self.loginTab)
        self.pwdInput.setGeometry(QtCore.QRect(90, 70, 113, 22))
        self.pwdInput.setObjectName("pwdInput")
        self.pwdInput.setEchoMode(QtWidgets.QLineEdit.Password)
        self.userLabel = QtWidgets.QLabel(self.loginTab)
        self.userLabel.setGeometry(QtCore.QRect(20, 20, 55, 16))
        self.userLabel.setObjectName("userLabel")
```

```
        self.passwordLabel = QtWidgets.QLabel(self.loginTab)
        self.passwordLabel.setGeometry(QtCore.QRect(10, 74, 55, 16))
        self.passwordLabel.setObjectName("passwordLabel")
        self.connectButton = QtWidgets.QPushButton(self.loginTab)
        self.connectButton.setGeometry(QtCore.QRect(10, 130, 93,
28))
        self.connectButton.setObjectName("connectButton")
        self.connectButton.clicked.connect(loginTab.connectDbFunc)
        self.disconnectButton = QtWidgets.QPushButton(self.loginTab)
        self.disconnectButton.setGeometry(QtCore.QRect(120, 130, 93,
28))
        self.disconnectButton.setObjectName("disconnectButton")

self.disconnectButton.clicked.connect(loginTab.disconnectDbFunc)
        self.tabWidget.addTab(self.loginTab, "")
        DatabaseGUI.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(DatabaseGUI)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 26))
        self.menubar.setObjectName("menubar")
        DatabaseGUI.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(DatabaseGUI)
        self.statusbar.setObjectName("statusbar")
        DatabaseGUI.setStatusBar(self.statusbar)

        # calling translate function to write text to widgets
        self.retranslateUi(DatabaseGUI)
        self.tabWidget.setCurrentIndex(0)
        QtCore.QMetaObject.connectSlotsByName(DatabaseGUI)

    # function to set all text in gui
    def retranslateUi(self, DatabaseGUI):
        _translate = QtCore.QCoreApplication.translate
        # setting all text to its correct value ie labels and drop
down boxes and button labels.
        DatabaseGUI.setWindowTitle(_translate("DatabaseGUI",
"MainWindow"))
        self.queryViewButton.setText(_translate("DatabaseGUI",
"Query View"))
        self.viewsComboBox.setItemText(0, _translate("DatabaseGUI",
"All Employees"))
        self.viewsComboBox.setItemText(1, _translate("DatabaseGUI",
"Transactions"))
        self.viewsComboBox.setItemText(2, _translate("DatabaseGUI",
"Employees & Discounts"))
        self.viewsComboBox.setItemText(3, _translate("DatabaseGUI",
"Employees with > 2 Orders"))
        self.viewsComboBox.setItemText(4, _translate("DatabaseGUI",
"Managers with Nike"))
        self.queryTableButton.setText(_translate("DatabaseGUI",
"View Table"))
        self.viewTableComboBox.setItemText(0,
_translate("DatabaseGUI", "Employee"))
        self.viewTableComboBox.setItemText(1,
_translate("DatabaseGUI", "Branch"))
        self.viewTableComboBox.setItemText(2,
_translate("DatabaseGUI", "Jobs"))
```

```
        self.viewTableComboBox.setItemText(3,
_translate("DatabaseGUI", "Transactions"))
        self.viewTableComboBox.setItemText(4,
_translate("DatabaseGUI", "Product"))
        self.viewTableComboBox.setItemText(5,
_translate("DatabaseGUI", "Audits"))
        self.viewTableComboBox.setItemText(6,
_translate("DatabaseGUI", "Shop"))

self.tabWidget.setTabText(self.tabWidget.indexOf(self.queryTab),
_translate("DatabaseGUI", "Pre-Made Query"))
        self.exportTableCombo.setItemText(0,
_translate("DatabaseGUI", "Employee"))
        self.exportTableCombo.setItemText(1,
_translate("DatabaseGUI", "Branch"))
        self.exportTableCombo.setItemText(2,
_translate("DatabaseGUI", "Jobs"))
        self.exportTableCombo.setItemText(3,
_translate("DatabaseGUI", "Transactions"))
        self.exportTableCombo.setItemText(4,
_translate("DatabaseGUI", "Product"))
        self.exportTableCombo.setItemText(5,
_translate("DatabaseGUI", "Audits"))
        self.exportTableCombo.setItemText(6,
_translate("DatabaseGUI", "Shop"))
        self.exportTableCombo.setItemText(7,
_translate("DatabaseGUI", "All Tables"))
        self.exportButton.setText(_translate("DatabaseGUI",
"Export"))
        self.selectTableLabel.setText(_translate("DatabaseGUI",
"Select Table"))
        self.pathsSaveLabel.setText(_translate("DatabaseGUI", "Path
to save"))
        self.selectPathButton.setText(_translate("DatabaseGUI",
"..."))

self.tabWidget.setTabText(self.tabWidget.indexOf(self.exportTab),
_translate("DatabaseGUI", "Export DB"))
        self.userLabel.setText(_translate("DatabaseGUI", "User"))
        self.passwordLabel.setText(_translate("DatabaseGUI",
"Password"))
        self.connectButton.setText(_translate("DatabaseGUI",
"Connect"))
        self.disconnectButton.setText(_translate("DatabaseGUI",
"Disconnect"))

self.tabWidget.setTabText(self.tabWidget.indexOf(self.loginTab),
_translate("DatabaseGUI", "Login"))

# this class contains all functions which contain logic and output
code for the 1st tab shown in the gui
class preMadeQueryTab_Logic():

    # function to query views in database and return in form of
table on gui
    def queryViewButtonPressed(self):
```

```
        # getting which view is wanted from gui drop down box
        view = ui.viewsComboBox.currentText()
        # error handling - try except
        try:
            # setting columns names depending on which view is
selected using selection statements
            if view == 'All Employees':
                view = 'all_employees'
                columns = ['Employee ID','First Name', 'Surname',
'Job Title', 'Location']
            elif view ==  'Transactions':
                view = 'transactionswithdiscountprice'
                columns = ['Trans ID','Purchase Date', 'Surname',
'Product Name', 'Price', 'Discount', 'Discounted Price']
            elif view ==  'Employees & Discounts':
                view = 'allemployeesanddiscounts'
                columns = ['Employee ID', 'Surname', 'Job Title',
'Discount']
            elif view ==  'Employees with > 2 Orders':
                view = 'employees2orders'
                columns = ['First Name', 'Surname', 'Number of
Order']
            elif view ==  'Managers with Nike':
                view = 'managerswithnike'
                columns = ['Product No', 'Brand', 'Employee ID',
'Job Title']

            # array is 2d array to hold each record in an element
with each attribute an element inside it.
            array = []
            # query to select all records and columns from chosen
view.
            query = 'SELECT * FROM ' + view
            # cursor executing query passed to it
            cur.execute(query)
            # for every row in the query i will append it to the
array
            for row in cur:
                # appending to array
                array.append(row)
            # setting columns count int table to length of record
            ui.tableView.setColumnCount(len(array[0]))
            # setting row count to number of records
            ui.tableView.setRowCount(len(array))
            # setting names of columns with corresponding heading
decided in selection statement above
            ui.tableView.setHorizontalHeaderLabels(columns)
            # for every record
            for row in range(len(array)):
                # for every attribute in record
                for column in range(len(array[0])):
                    # set position row, column in table as data in
that position in array

ui.tableView.setItem(row,column,QTableWidgetItem(str((array[row][col
umn])))))
```

```
        except:
            # if error in this code then user is not logged in.
            QtWidgets.QMessageBox.warning(None, "Error", "You are
not logged in")
            # send user to login tab
            ui.tabWidget.setCurrentIndex(2)
            QtWidgets.QMessageBox.information(None, "Info", "Please
log in")

    # function to query tables in database and return in form of
table on gui
    def queryTableButtonPressed(self):
        # getting table and username from gui
        table = ui.viewTableComboBox.currentText()
        username = ui.userInput.text()
        # try except to catch error of logging in
        try:
            # setting columns to corresponding table selected
            if table == 'Employee':
                # column query gets info from the hidden
information_schema table
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'employee' order by
ordinal_position"
            elif table ==  'Transactions':
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'transactions' order
by ordinal_position"
            elif table ==  'Branch':
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'branch' order by
ordinal_position"
            elif table ==  'Shop':
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'shop' order by
ordinal_position"
            elif table ==  'Product':
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'product' order by
ordinal_position"
            elif table ==  'Jobs':
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'jobs' order by
ordinal_position"
            # audit table is only available to be accessed by some
users
            elif table == 'Audits' and (username == 'root' or
username == 'CEO' or username == 'JackYoung'):
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'audits' order by
ordinal_position"
            else:
                # error message
                QtWidgets.QMessageBox.warning(None, "Error", "You do
not have access to this table")
```

```
            # array for columns to be stored in
            columns = []
            # cursor executing chosen column query
            cur.execute(columnQuery)
            for row in cur:
                row = row[0]
                # appending columns from query into the columns
array
                columns.append(row)
            # 2d array to store records and attributes in them
            array = []
            # selecting all from chosen table
            query = 'SELECT * FROM ' + table
            cur.execute(query)
            for row in cur:
                # appending records to 2d array
                array.append(row)
            # setting gui table columns names and row and column
lengths to the length of array
            ui.tableView.setColumnCount(len(array[0]))
            ui.tableView.setRowCount(len(array))
            ui.tableView.setHorizontalHeaderLabels(columns)
            for row in range(len(array)):
                for column in range(len(array[0])):
                    # inserting data into table

ui.tableView.setItem(row,column,QTableWidgetItem(str((array[row][col
umn]))))
        except:
            # if user is not logged in, then send them to login tab
            QtWidgets.QMessageBox.warning(None, "Error", "You are
not logged in")
            ui.tabWidget.setCurrentIndex(3)
            QtWidgets.QMessageBox.information(None, "Info", "Please
log in")

# this class contains all functions which contain logic and output
code for the 2nd tab shown in the gui
class exportDBTab_Logic():
    # function is called when export button is pressed on export tab
    def exportButtonPressed(self):
        # getting username and table selected from the gui
        username = ui.userInput.text()
        table = ui.exportTableCombo.currentText()
        # try and except to catch error from login missing
        try:
            # setting columns to corresponding table selected
            if table == 'Employee':
                # column query gets info from the hidden
information_schema table
                # query to select all from selected table
                query = "SELECT * from employee"
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'employee' order by
ordinal_position"
            elif table ==  'Transactions':
```

```
                query = "SELECT * from transactions"
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'transactions' order
by ordinal_position"
            elif table ==  'Branch':
                query = "SELECT * from branch"
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'branch' order by
ordinal_position"
            elif table ==  'Shop':
                query = "SELECT * from shop"
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'shop' order by
ordinal_position"
            elif table ==  'Product':
                query = "SELECT * from product"
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'product' order by
ordinal_position"
            elif table ==  'Jobs':
                query = "SELECT * from jobs"
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'jobs' order by
ordinal_position"
            elif table == 'All Tables':
                query = "SELECT * FROM exportTable"
                columnQuery = "SELECT distinct column_name from
information_schema.columns where table_name = N'exporttable' order
by ordinal_position"
            elif table == 'Audits' and (username == 'root' or
username == 'CEO' or username == 'JackYoung'):
                query = "SELECT * from audits"
            else:
                QtWidgets.QMessageBox.warning(None, "Error", "You do
not have access to this table")

            # array to save columns and set table column names
            columns = []
            cur.execute(columnQuery)
            for row in cur:
                row = row[0]
                columns.append(row)

            # 2d array to store records and attributes
            array = []
            cur.execute(query)
            for row in cur:
                array.append(row)

            # if file path selected is nothing
            if self.filePath == '':
                # ask user if they are sure they want to save to
current working directory
                ans = QtWidgets.QMessageBox.question(None,
"Warning", "File Path is empty - file will be saved in current
working directory\n Do you want to continue?")
```

```
                # checks question answer and acts accordingly
                if ans == QtWidgets.QMessageBox.No:
                    # if answer is no then abort export
                    QtWidgets.QMessageBox.warning(None, "Aborted",
"Export Aborted")
                else:
                    # if answer is yes then export to cwd
                    with open("export.csv","w+") as my_csv:
                        csvWriter = csv.writer(my_csv,delimiter=',')
                        # write column names
                        csvWriter.writerow(columns)
                        # write records
                        csvWriter.writerows(array)
                    # message to show where export is saved
                    QtWidgets.QMessageBox.information(None,
"Success", "Export complete in CWD/export.csv")
            else:
                # if file path is not empty then save to file path
                with open(self.filePath + "/export.csv","w+") as
my_csv:
                    csvWriter = csv.writer(my_csv,delimiter=',')
                    # write column names
                    csvWriter.writerow(columns)
                    # write records
                    csvWriter.writerows(array)
                # message to show where file is saved
                QtWidgets.QMessageBox.information(None, "Success",
'Export complete in "' + self.filePath + '/export.csv"')
        except:
            # if error then user is not logged in - show error
message and send to login page
            QtWidgets.QMessageBox.warning(None, "Error", "You are
not logged in")
            ui.tabWidget.setCurrentIndex(3)
            QtWidgets.QMessageBox.information(None, "Info", "Please
log in")

    # function that is called when select path button is activated
    def selectPathButtonPressed(self):
        # file dialog to choose path
        self.filePath =
QtWidgets.QFileDialog.getExistingDirectory(None, 'Select Folder To
Save To')
        # set text box to path
        ui.pathToSaveEditBox.setText(self.filePath)

# this class contains all functions which contain logic and output
code for the 3rd tab shown in the gui
class loginTab_Logic():

    # function to connect to database -  called when connect button
is     pressed
    def connectDbFunc(self):
        # global variables to use cursor in other functions
        global conn
        global cur
```

```
        # get username and password from gui
        username = ui.userInput.text()
        pwd = ui.pwdInput.text()
        # try to login
        try:
            # db connection via local host using input password and
username
            conn = mysql.connector.connect(
                host="localhost",
                user=username,
                password=pwd,
                database="sportsredirect"
            )
            cur = conn.cursor()
            # when connected then show connection successful
            QtWidgets.QMessageBox.information(None, "Info",
"Connection Successful")
        except:
            # if error then password incorrect or user incorrect
            QtWidgets.QMessageBox.warning(None, "Error",
"User/Password Incorrect")

    # function to disconnect from database -  called when disconnect
button is pressed
    def disconnectDbFunc(self):
        # global variables
        global cur
        global conn
        try:
            # disconnecting from db
            cur.close()
            conn.close()
            QtWidgets.QMessageBox.information(None, "Info", "DB
Disconnect Successful")
        # catching error if db not connected
        except:
            QtWidgets.QMessageBox.warning(None, "Error", "DB Not
connected - disconnection impossible")

# runs when file is executed
if __name__ == "__main__":
    import sys
    # initialising classes with objects
    app = QtWidgets.QApplication(sys.argv)
    DatabaseGUI = QtWidgets.QMainWindow()
    loginTab = loginTab_Logic()
    preMadeQueryTab = preMadeQueryTab_Logic()
    exportTab = exportDBTab_Logic()
    exportTab.filePath = ''
    ui = Ui_DatabaseGUI()
    ui.setupUi(DatabaseGUI)
    # showing gui
    DatabaseGUI.show()
    sys.exit(app.exec_())
```

3.3. GUI Design/Proof

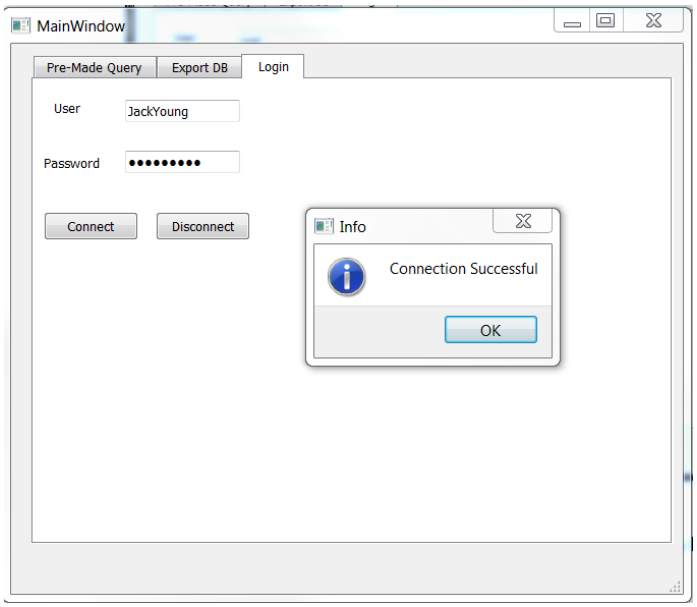

Figure 9: Login Window GUI with login proof

In figure 9 you can see the graphical user interface that contains a login window with login proof and error handling.

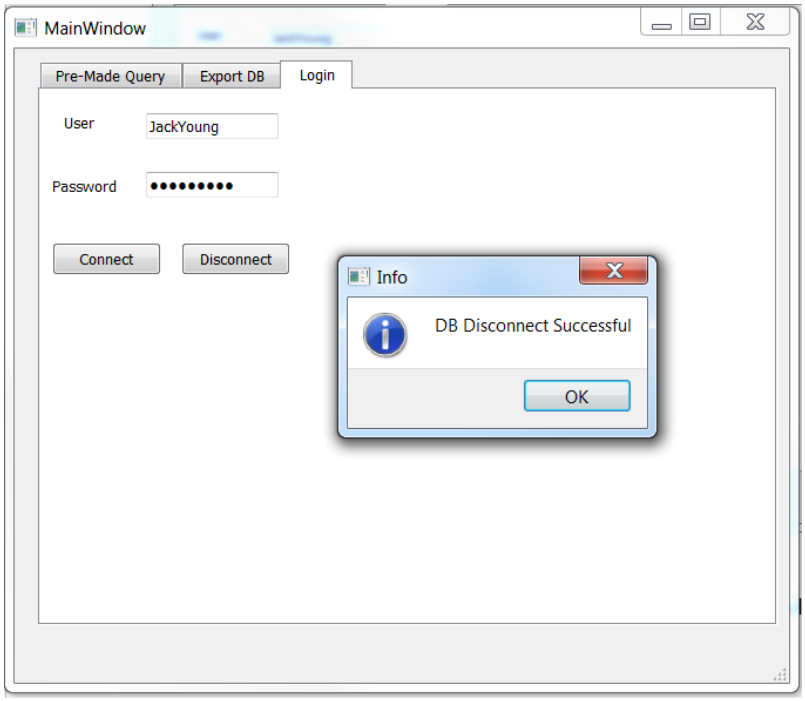

Figure 10: Login window disconnect proof

In figure 10 you can see the Login window of my GUI with proof of a disconnection from database and error handling.

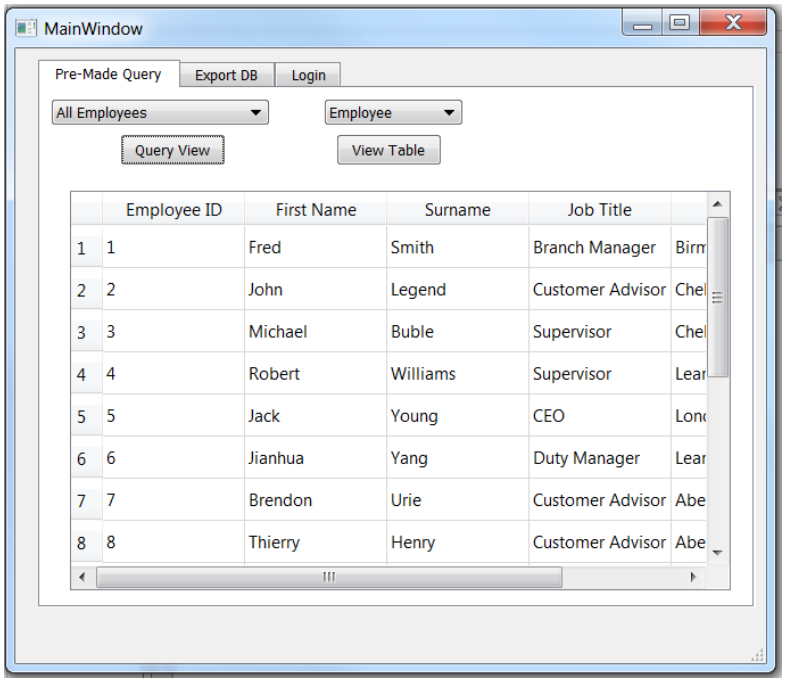

Figure 11: Pre-made query tab on gui

Figure 11 shows the GUI design for tab 1 (pre-made queries) with the all_employees view displayed in a table. This page has the option to show views or tables in the form of a table.

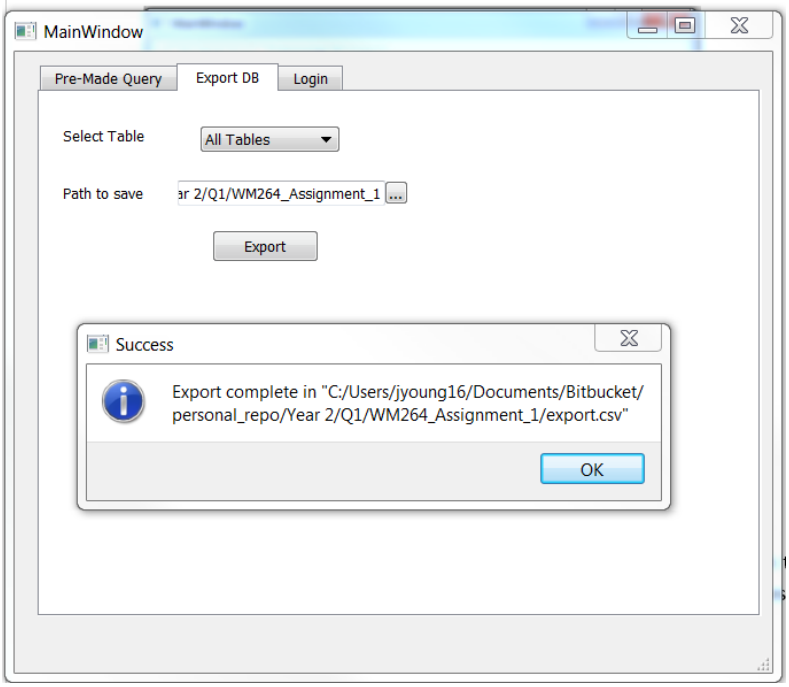

Figure 12: Shows export table with proof of export

Figure 12 shows how the GUI allows users to export a single table or all tables together to a file path of the user's choice

3.4. Export Proof

export.csv

```
TransNo,DelAddress,PurchaseDate,ProductNo,Brand,ProductName,Price,EndDate,Stock,Stock Location,EmpNo,FName,SName,PhoneNo,EmailAddr,DOB,JobNo,JobType,EmpDiscount,BranchNo,BranchLocation

1,157 Prestbury Road,2020-10-10,12,Gul,Wetsuit,225.00,2014-02-12,5,3,1,Fred,Smith,07563 326610,fred.smith@hotmail.com,1996-10-01,7,Branch Manager,20,6,Birmingham

2,50 Chatham Avenue,2020-05-01,10,Vans,Air Force Trainers,119.00,2020-11-18,8,6,2,John,Legend,07821 806490,john.legend@hotmail.com,1989-10-31,10,Customer Advisor,10,2,Cheltenham

3,9 Southbourne,2018-05-01,5,Nike,Boxers,10.00,2018-01-01,2,5,1,Fred,Smith,07563 326610,fred.smith@hotmail.com,1996-10-01,7,Branch Manager,20,6,Birmingham

4,2 Warwick Street,2020-07-15,6,Adidas,Sports Shorts,200.00,2019-10-01,7,5,10,Frank,Lampard,07778 675433,frank.lampard@yahoo.com,1998-12-02,10,Customer Advisor,10,2,Cheltenham

5,7 Oxford Street,2020-02-17,6,Adidas,Sports Shorts,200.00,2019-10-01,7,5,5,Jack,Young,07993 606630,jaxkyoung@icloud.com,2000-11-18,6,CEO,20,1,London

6,27 Bromley Common,2020-10-15,11,Nike,SB Trainers,87.00,2014-06-12,10,3,2,John,Legend,07821 806490,john.legend@hotmail.com,1989-10-31,10,Customer Advisor,10,2,Cheltenham

7,22 Regent Street,2020-03-30,4,Tommy Hilfiger,White T Shirt,100.00,2020-09-10,1,2,8,Thierry,Henry,07286 967384,Thierry.henry@hotmail.com,1998-02-07,10,Customer Advisor,10,5,Aberdeen

8,31 Parade,2019-12-25,9,Tommy Jeans,Ripped Jeans,125.00,2015-03-10,7,6,6,Jianhua,Yang,07452 685734,jianhua.yang@warwick.ac.uk,1970-03-12,8,Duty Manager,20,3,Leamington Spa

9,89 Gloucester Road,2019-10-30,2,Reebok,Blue T Shirt,65.00,2020-05-25,22,2,8,Thierry,Henry,07286 967384,Thierry.henry@hotmail.com,1998-02-07,10,Customer Advisor,10,5,Aberdeen

10,25 Pitville Crescent,2017-04-16,1,Nike,Air Force Trainers,100.00,2019-10-01,10,1,4,Robert,Williams,07493 406630,robbie.williams@hotmail.com,1998-05-28,9,Supervisor,10,3,Leamington Spa

11,1 Connor Place,2020-09-29,7,Reebok,Socks,3.00,2017-10-01,9,1,9,John,Terry,07986 896732,john.terry@chelseafc.com,1999-10-30,7,Branch Manager,20,3,Leamington Spa

12,12 Gaydon Road,2019-12-13,3,Adidas,Trainers,32.00,2020-10-15,5,2,12,Ed,Sheeran,07932 230789,ed@sheeran.me,1989-10-12,10,Customer Advisor,10,4,Bath Spa

13,12 Gaydon Road,2019-12-14,4,Tommy Hilfiger,White T Shirt,100.00,2020-09-10,1,2,12,Ed,Sheeran,07932 230789,ed@sheeran.me,1989-10-12,10,Customer Advisor,10,4,Bath Spa

14,157 Prestbury Road,2020-10-27,10,Vans,Air Force Trainers,119.00,2020-11-18,8,6,5,Jack,Young,07993 606630,jaxkyoung@icloud.com,2000-11-18,6,CEO,20,1,London

,,,,,,,,,13,Freddie,Mercury,07493 231890,fred@queen.co.uk,1956-04-30,10,Customer Advisor,10,1,London

,,,,,,,,,3,Michael,Buble,07533 2164430,michael.buble@gmail.com,1992-07-01,9,Supervisor,10,2,Cheltenham

,,,,,,,,,27,Claudia,Ellis,07932 188332,cells@jlr.com,1999-10-01,7,Branch Manager,20,2,Cheltenham

,,,,,,,,,29,Nicole,Minou,07777 120590,nminou@jlr.com,1985-04-10,9,Supervisor,10,2,Cheltenham

,,,,,,,,,11,Steve,Gerrard,07993 407754,stevieg@liverpool.com,1967-09-10,10,Customer Advisor,10,4,Bath Spa

,,,,,,,,,28,Dom,Taylor,07563 326610,dtayl@jlr.com,1982-02-21,12,HR Advisor,15,4,Bath Spa

,,,,,,,,,7,Brendon,Urie,07678 109674,brendon.urie@hotmail.com,1978-04-15,10,Customer Advisor,10,5,Aberdeen

,,,,,,,,,,,,,,,,,7,Sheffield

,,,,,,,,,,,,,,,,,8,Cirencester

,,,,,,,,,,,,,,,,,9,Warwick

,,,,,,,,,,,,,,,,,10,Liverpool
```

Figure 13: Exported table as CSV

Figure 13 shows all tables merged into one and exported as a CSV ready to be used elsewhere.

References:

Elmasri, R. & Navathe, S. B. 2016. *Fundamentals of Database Systems, Global Edition,* Harlow, United Kingdom, UNITED KINGDOM, Pearson Education Limited.

Harrington, J. L. 2016a. Chapter 3 - Why Good Design Matters. *In:* HARRINGTON, J. L. (ed.) *Relational Database Design and Implementation (Fourth Edition).* Boston: Morgan Kaufmann.

Harrington, J. L. 2016b. Chapter 5 - The Relational Data Model. *In:* HARRINGTON, J. L. (ed.) *Relational Database Design and Implementation (Fourth Edition).* Boston: Morgan Kaufmann.

Hill, S. 2020. *Advantages & Disadvantages of a Computerized Database* [Online]. Available: https://www.techwalla.com/articles/advantages-disadvantages-of-a-computerized-database [Accessed 29/10 2020].