EXCERCISE 1

-- **** Databases, Tables and CRUD Operations ****
-- 1. Create a database called coursedb (DONE)

-- 2. In the coursedb database create a table called suppliers with these fields:
-- supplierid int (Primary Key), company varchar(40), country varchar(30), city varchar(30)

-- 3. Execute this statement to populate the suppliers table

```
insert into suppliers(supplierid, company, country, city)
select SupplierID, CompanyName, Country, City from northwind.Suppliers;
```

-- 4. Create another table called products with these fields:
-- productid int auto_increment (Primary Key), product varchar(40), price decimal(6,2),
-- supplierid int (Foreign key that references supplierid in the suppliers table)

-- 5. Execute this statement to populate the table

```
insert into products(company, price, supplierid)
select productname, unitprice, supplierid from northwind.products;
```

-- 6. Try some insert, update and delete statements, you should not be able to delete existing suppliers due
-- to the foreign key constraint but you can delete newly inserted suppliers and any existing products. The foreign
-- key constraint will also prevent you from changing the supplierid of a product to one that does not exist in the
-- suppliers table (DONE)

EXCERCISE 2

-- **** Select Statements Exercise ****
-- 1 Show the ProductName and UnitPrice of all products in the price range 20 to 30 (DONE)

-- 2 Show the countries in the customers table with no duplicates (DONE)

-- 3 Show the CompanyName, Country and City of all customers that are restaurants (the word restaurant should be in the CompanyName) (DONE)

-- 4 Show The CompanyName, ContactName, Country,and City of all suppliers from Germany, France, Italy and Spain (DONE)

-- 5 Show the OrderID, OrderDate and CustomerID for all orders in July of 2017

-- 6 Create a report that shows each ProductName, UnitPrice, UnitsInStock and ReorderLevel and a message
-- for each product that will be either 'Order Stock' (UnitsInStock is equal to or below ReorderLevel) or 'Sufficient Stock'
-- (UnitsInStock is above the ReorderLevel) give the report headings such as Product, Price, Stock, 'Reorder Level' and
-- 'Stock Alert'. Discontinued products should be excluded from the report. (DONE)

```
SELECT CompanyName, OrderID, OrderDate FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
WHERE Country = 'Germany';

SELECT CompanyName, OrderID, OrderDate FROM Customers c, Orders o
WHERE c.CustomerID = o.CustomerID
AND Country = 'Germany';

SELECT s.CompanyName Supplier, s.Country `Supplier Country`, c.CompanyName Customer,
c.Country `Customer Country` FROM Suppliers s RIGHT JOIN Customers c
ON s.Country = c.Country;
```

The Where filters the raw data before the group by is executed

In this sequence can I use >where >group by >having >order by ?

-- ***** Joins and Aggregates *****
-- 1. List the CompanyName (customers), OrderID (orders) and OrderDate (orders) for all customers from Germany who placed an order
-- in 2018, you need to join the customers and orders tables to do this. (DONE)

-- 2. Write a SQL Statement that displays a list showing the Category (CategoryName), Supplier (CompanyName), Product (ProductName) and
-- Price (UnitPrice) of all products. To do this you will need to Join the Categories, Products and Suppliers tables (DONE)

-- 4 List the CategoryName (categories), ProductName (products) and UnitPrice (products) for the 5
-- most expensive products. To do this you will need to Join the Categories and Products tables

-- 5 Write a SQL Statement that displays the total revenue for each category. The revenue can be calculated using the expression
-- (unitprice * quantity) * (1 â€“ discount). You will need to join the Categories, Products and Order_Details tables to do this.
-- Make sure you use the UnitPrice in the Order_Details table not the Products table. Sort the list in descending order of revenue.

-- 6 Write a SQL Statement that displays the total revenue generated by each customer country. The revenue can be calculated
-- using the expression (unitprice * quantity) * (1 â€“ discount) in a sum aggregate. You will need to join the Customers, Orders and
-- Order_Details tables to do this. Only show rows where the revenue is over 100,000

-- ***** Subqueries and Views *****
-- 1. Use a subquery to generate a list of customers (CompanyName, ContactName, Country, City) that are in the same country as
-- the employees.

-- 2. Execute this statement:
SELECT ProductName, DATE_FORMAT(OrderDate, "%D %b %Y") `Order Date`,
Quantity FROM Products p JOIN Order_Details od ON p.ProductID = od.ProductID
JOIN Orders o ON o.OrderID = od.OrderID ORDER BY ProductName, Quantity DESC;
-- Can you modify it so that we only see the row or rows (in the case of a tie) for the best sale for each product?

-- 3. Create a view vCountryFromTo that lists the OrderID(Orders), OrderDate(Orders), ProductName(Products), Country(Suppliers) with an alias
-- 'Supplier Country' and Country(Customers) with an alias 'Customer Country'. You will need to join the Suppliers, Products, Order_Details,
-- Orders and Customers tables to do this. When querying the view you can limit the results to a single month of your choice.

```
SELECT * FROM vCountryFromTo WHERE MONTH(OrderDate) = 4 AND YEAR(OrderDate) =
2017;

CREATE View vDemo AS
SELECT ProductName, DATE_FORMAT(OrderDate, "%D %b %Y") `Order Date`,
Quantity FROM Products p JOIN Order_Details od ON p.ProductID = od.ProductID
JOIN Orders o ON o.OrderID = od.OrderID ORDER BY ProductName, Quantity DESC;
-- Can you modify it so that we only see the row or rows (in the case of a tie) for the best sale
for each product?
ALTER VIEW vDemo AS
SELECT ProductName, DATE_FORMAT(OrderDate, "%D %b %Y") `Order Date`,
Quantity FROM Products p JOIN Order_Details od ON p.ProductID = od.ProductID
JOIN Orders o ON o.OrderID = od.OrderID WHERE Quantity =
(
   SELECT MAX(Quantity) FROM order_details Where ProductID = p.ProductID
);
```