INF-2700: Database system

Assignment 1

Iver Mortensen

UiT id: imo059@uit.no

GitHub user: Solcelle

September 24, 2023

1 Part 1

1.1 Task 1

These are the tables used to create the database:

CREATE TABLE Customers (customerNumber INTEGER PRIMARY KEY, customerName TEXT NOT NULL, contactLastName TEXT NOT NULL, contactFirstName TEXT NOT NULL, phone TEXT NOT NULL, addressLine1 TEXT NOT NULL, addressLine2 TEXT NULL, city TEXT NOT NULL, state TEXT NULL, postalCode TEXT NULL, country TEXT NOT NULL, salesRepEmployeeNumber INTEGER NULL, creditLimit REAL NULL)

CREATE TABLE Employees (employeeNumber INTEGER PRIMARY KEY, lastName TEXT NOT NULL, firstName TEXT NOT NULL, extension TEXT NOT NULL, email TEXT NOT NULL, officeCode TEXT NOT NULL, reportsTo INTEGER NULL, jobTitle TEXT NOT NULL, FOREIGN KEY (reportsTo) REFERENCES Employees(employeeNumber))

CREATE TABLE Offices (officeCode TEXT PRIMARY KEY, city TEXT NOT NULL, phone TEXT NOT NULL, addressLine1 TEXT NOT NULL, addressLine2 TEXT NULL, state TEXT NULL, country TEXT NOT NULL, postalCode TEXT NOT NULL, territory TEXT NOT NULL)

CREATE TABLE OrderDetails (orderNumber INTEGER NOT NULL, productCode TEXT NOT NULL, quantityOrdered INTEGER NOT NULL, priceEach REAL NOT NULL, orderLineNumber INTEGER NOT NULL, PRIMARY KEY (orderNumber, productCode), FOREIGN KEY (productCode) REFERENCES Products)

CREATE TABLE Orders (orderNumber INTEGER PRIMARY KEY, orderDate TEXT NOT NULL, requiredDate TEXT NOT NULL, shippedDate TEXT NULL, status TEXT NOT NULL, comments TEXT NULL, customerNumber INTEGER NOT NULL, FOREIGN KEY (customerNumber) REFERENCES Customers)

CREATE TABLE Payments (customerNumber INTEGER NOT NULL, checkNumber TEXT NOT NULL, paymentDate TEXT NOT NULL, amount REAL NOT NULL, PRIMARY KEY (customerNumber, checkNumber), FOREIGN KEY (customerNumber) REFERENCES Customers)

CREATE TABLE ProductLines(productLine TEXT PRIMARY KEY, description TEXT NULL)

CREATE TABLE Products (productCode TEXT PRIMARY KEY, productName TEXT NOT NULL, productLine TEXT NOT NULL, productScale TEXT NOT NULL, productVendor TEXT NOT NULL, productDescription TEXT NOT NULL, quantityInStock INTEGER NOT NULL, buyPrice REAL NOT NULL, MSRP REAL NOT NULL, FOREIGN KEY (productLine) REFERENCES Productlines)

1.2 Task 2

a)
SELECT customerName, contactLastName, contactFirstName
FROM Customers;

Selects customer name, contact last name and contact first name from the Customers table.

b)
SELECT *
FROM Orders
WHERE shippedDate IS NULL;

Selects all from the Orders table where shipped date is empty.

c)
SELECT C.customerName AS Customer, SUM(OD.quantityOrdered) AS Total
FROM Orders O, Customers C, OrderDetails OD
WHERE O.customerNumber = C.customerNumber
AND O.orderNumber = OD.orderNumber
GROUP BY O.customerNumber
ORDER BY Total DESC;

Selects customer name and their total quantity of ordered products ordered by the quantity.

d)
SELECT P.productName, T.totalQuantityOrdered
FROM Products P NATURAL JOIN
(SELECT productCode, SUM(quantityOrdered) AS totalQuantityOrdered
FROM OrderDetails GROUP BY productCode) AS T
WHERE T.totalQuantityOrdered = 1000;

Select product name and total quantity ordered of that product where quantity is above or equal to 1000.

1.3 Task 3

In the two first queries, the selected columns in each query is from the same table. This means the data can be selected directly from the table. The country column in Customers have some variation in white space. Using LIKE here instead of IS removes these inconsistencies. The same is done in the second query.

In the third query the selected data is from three different tables. The required columns are selected from the tables they reside in. Only the orders with the correct order status is chosen. The selected columns are aligned with the orders with the correct order status.

In the forth query the Orders and Orderdetails tables are combined on order number. This is used to get customer number from Orders and the summarized ordered price from Orderdetails. This is then joined with Customers on customer number to get customer name and credit limit. The sum of total payments and customer number is selected from Payments. This is then also joined with Customers on customer number. A WHERE is used to only select rows where customers have exceeded their credit limit.

– Select customer names from customer numbers SELECT C.customerName FROM Customers C WHERE C.customerNumber IN (– Select all customer numbers that have ordered the same product codes SELECT O.customerNumber FROM OrderDetails OD JOIN Orders O ON O.orderNumber = OD.orderNumber WHERE OD.productCode IN (– Select all product codes orderd by 219 SELECT OD.productCode FROM OrderDetails OD JOIN Orders O ON O.orderNumber = OD.orderNumber WHERE O.customerNumber = 219)) ORDER BY C.customerName;

When I made the fifth query I misunderstood the task and got all customers that have ordered one of the products that 219 have ordered. The task was to get the customers that have ordered all the products that 219 have ordered.
The query selects the product codes of the products that 219 have ordered. After that it selects the customer numbers that have ordered the same products. The customer numbers are used to get the customer names, which is the final list. The list is ordered by customer name to get the results in a alphabetical order.

2 Part 2

The first part was to open the database. This is done with the sqlite3 open function. This function has a return code which is used to check if the opening of the database was successful.
The second part was to get the result from the query and format it for the result test. The sqlite3 prepare function is used to compile the current query. This function also has a return code used to check if it was successful. Sqlite3's step function is used on the query statement to get each row of the query result. Each column within the current row is added to the result with the correct format. The statement is freed using sqlite3's finalize function.
The last part is was to close the database. This is done using sqlite3's close function.