CITS1402 Week 5 Lab

Gordon Royle

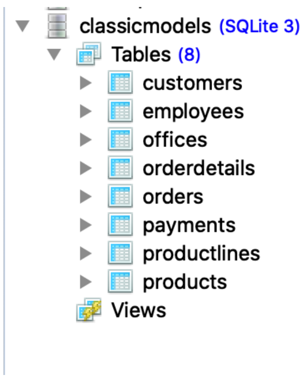
2023 Semester Two

Learning Aims

This lab is concerned with combining all that we have learned so far by constructing queries to run against the supplied database `classicmodels.db`.

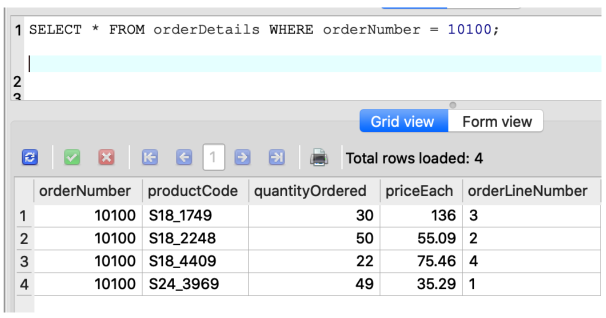
This is one of the original test databases distributed by MySQL, and I have modified it as little as possible while converting it into SQLite. (Most of the changes are to do with the table creation statements, where MySQL and SQLite differ in a number of ways.)

First download `classicmodels.dbs` and use SQLiteStudio to attach and open the database files and then explore the tables.



This describes the activities of a *wholesaler* selling scale models of classic trains, planes, cars etc. to various retail outlets such as toy shops. The database includes customers, employees, products, orders and offices.

It is more complicated than other databases that we have seen, and so you should spend a little time just examining the data to see how aspects of the business are encoded in the database.



```
SELECT * FROM orderDetails WHERE orderNumber = 10100;
```

Total rows loaded: 4

	orderNumber	productCode	quantityOrdered	priceEach	orderLineNumber
1	10100	S18_1749	30	136	3
2	10100	S18_2248	50	55.09	2
3	10100	S18_4409	22	75.46	4
4	10100	S24_3969	49	35.29	1

Figure 1: An order is a bunch of order details

Some features of `ClassicModels`

Most features of Classic Models are straightforward, but there are a few things that may be confusing for someone encountering the database model for the first time.

- The table `customers`

  A customer is identified by a unique `customerNumber` and has a name (such as "La Rochelle Gifts") and a variety of other contact information.

  Remember that Classic Models is a *wholesaler* and so its customers are not individuals who might buy one or two different model trains, but *retail outlets* who might buy 25 units of the same model to sell in their store.

- The table `orders`

  An order is identified by a unique `orderNumber` which identifies the customer who made the order, and a variety of other information (date etc) related to that particular order.

  This table does not contain any information about the *actual products* in the order.

- The table `orderdetails`

  This is the most critical table to understand. A single order can contain a number of different products; for example a toy shop might order 10 model Ferraris, 5 model Porsches and 6 model Lamborghinis. Each row of `orderdetails` contains the information about *one of the products* in a particular order. This includes the unique code for the product, the number of units, and the price per unit. Notice: the `priceEach` column is the price *charged* by Classic Models for each unit.

  (Sometimes these are called *line items* because they usually appear on a receipt on a single line.)

  Figure 1 shows that order number 10100 has 4 line items – the first line item is 30 units of product `S18_1749` at a unit price of 136.

- The table `products`

This lists the actual products, each identified by a unique *product code*, along with descriptive information such as its name, and what *type* of product it is (as listed in `productLine` field). In addition, this table lists the quantity in stock, the actual price paid by the wholesaler and the MSRP (manufacturer's recommended price).

- *IMPORTANT* Prices

  There are multiple fields in the database related to *prices* and it is important not to confuse them. So read this very carefully.

  - The column `orderdetails.priceEach`

    This is the amount *charged by* Classic Models *per unit* for this particular line item. Classic Models may charge a different price for each customer.
  - The column `products.buyPrice`

    This is the unit price *paid by* Classic Models when they purchase bulk lots direct from the manufacturer. To remain in business, Classic Models will need to charge their customers substantially more than this price when they sell the same product.
  - The column `products.MSRP`

    This is the manufacturer's suggested retail price, which is the price that the manufacturer suggests as the selling price in a retail store selling to individuals. (For this database, the MSRP is almost irrelevant, because the database does not contain any information about retail sales to individuals.)
  - The column `payments.amount`

    This is an actual amount received when a customer makes a payment (cheque or bank transfer) to Classic Models. Customers may pay for orders in advance, or immediately, or they may delay payments until the end of the month or quarter and pay for several orders at once.

    Therefore the real value of a customer's business with Classic Models can only be determined by the *cost of the orders* they have made, and not by how much they have actually paid.

Lab Questions

Question 1. Write an SQL query that will determine how many orders have been made.

```sql
SELECT COUNT(*)
FROM orders;
```

Question 2. Write an SQL query whose output lists the `officeCode` and the `city` for each of the offices run by ClassicModels.

```
SELECT officeCode, city
FROM   offices;
```

QUESTION 3. Write a single SQL query whose output is a 1-column table containing the *last names* of the employees working from the Sydney office. Your query should not *hard code* the current code for the Sydney office, because Classic Models is considering changing its office codes, and your query must continue to work even if they are changed.

```
SELECT E.lastname
FROM   employees E, offices O
WHERE  E.officecode = O.officecode
       AND O.city = 'Sydney';
```

QUESTION 4. Using only the `orders` table, write a single SQL query to count the number of unique customers who have ever made an order.

```
SELECT COUNT(DISTINCT customerNumber) FROM orders;
```

You may need to look up the syntax for `COUNT (DISTINCT .. )`

QUESTION 5. Some of the customers in the table `Customers` have *never made* an order (according to the table `orders`. Look up, and use, the `EXCEPT` statement to find the *customerNumber* for all these customers.

```
SELECT customerNumber FROM customers
EXCEPT
SELECT customerNumber FROM orders;
```

(The `EXCEPT` statement is the SQL equivalent of the *set difference* operator, which finds the values that are in one set and *not* in another, usually expressed as $S \setminus T$ or just $S - T$.)

QUESTION 6. Write a single SQL query that calculates the *total price* of order 10122. (Your output should be a table with 1 row and 1 column.)

Your query must calculate the cost of each line item (which depends on *quantityOrdered* and *priceEach*) and then add them up to get the total cost of all of the line items.

The aggregate function `SUM` is likely to be helpful here.

```
SELECT
       SUM(quantityOrdered * priceEach)
```

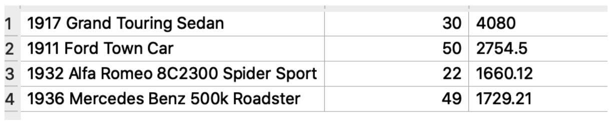
```
FROM   orderdetails
WHERE orderNumber = 10122;
```

Question 7. Write a SQL query to list the order numbers of all the orders that contain *planes* (use the `productLine` column of `product` to decide if something is a plane). Make sure you list each order number once only.

```
SELECT DISTINCT orderNumber
FROM   orderdetails NATURAL JOIN products
WHERE  productLine = "Planes";
```

Question 8. Write a SQL query to produce a "human-readable" summary of order 10122. This should be a list containing the *product name*, *quantity* and *total price* for each of the line items in the order.

If you were to test your code by changing 10122 to 10100 then you should get the following output.



1	1917 Grand Touring Sedan	30	4080
2	1911 Ford Town Car	50	2754.5
3	1932 Alfa Romeo 8C2300 Spider Sport	22	1660.12
4	1936 Mercedes Benz 500k Roadster	49	1729.21

```
SELECT products.productName,
       orderdetails.quantityOrdered,
       orderdetails.quantityOrdered * orderdetails.priceEach
  FROM orderdetails
       JOIN
       products ON orderDetails.productCode = products.productCode
 WHERE orderNumber = 10122;
```

Question 9. Other than the CEO (President), each employee reports to another employee (his or her immediate supervisor) with this information contained the `Employees` table.

Write an SQL query that lists the names (first and last) of the employees who report to employee 1102. (Test: One of these employees is called Pamela Castillo.)

```
SELECT E.firstName, E.lastName
FROM Employees E
WHERE E.reportsTo = 1102;
```

QUESTION 10. The operator used to *concatenate* two strings is `||`.

For example, `"John" || "Smith"` is the string `"JohnSmith"`.

Write a SQL query that lists the names of the employees in the format `Last, First` (so that John Smith would be listed as `Smith, John`). Ensure that the listing is in *alphabetical order*.

Don't forget the *comma* after the first name and the *space* before the last name.

```
SELECT lastName || ", " || firstName AS name
FROM Employees ORDER BY name;
```

QUESTION 11. The CEO is the only employee with no supervisor. Decide what would be a good way to indicate this in a database and thereby write a query that returns the first and last name of the CEO.

```
SELECT firstName, lastName
FROM   employees
WHERE  reportsTo IS NULL;
```

This produces Diane Murphy as President.

QUESTION 12. (Challenge) Write a SQL query that produces the order numbers (once each) of all orders that include both a plane *and* a ship. (This needs multiple copies of the same table(s) or a subquery.)

```
SELECT DISTINCT O1.orderNumber
  FROM orderDetails O1,
       orderDetails O2,
       products P1,
       products P2
 WHERE O1.productCode = P1.productCode AND
       O2.productCode = P2.productCode AND
       O1.orderNumber = O2.orderNumber AND
       P1.productLine = 'Planes' AND
       P2.productLine = 'Ships';
```
