Scoops Ahoy — CITS1402 Project

Gordon Royle

2019 Semester Two

Version 2.0 (14 Oct 2019) - Final RELEASE

Overview

The purpose of this project is to develop and test your ability to write both simple and advanced MySQL without the time-pressure of an exam and where you have access to a computer and the MySQL documentation to write your code.

However, the questions are not meant to be fiendishly difficult or to investigate extremely advanced or intricate MySQL. Students who are fully up-to-date with the labs and lecture material should be able to do most of the project within the specified lab times.

This is an *individual* project and so you may not submit any code that was not developed by you alone. You may discuss the project in general terms with your friends or ask the lab facilitators. You may wonder where to draw the line between legitimate discussion and unacceptable collaboration — as a rule-of-thumb, you should not look at anything *written* (on paper) or *displayed* (on a screen) by someone else. If you ask for assistance on `help1402` then try to narrow down the exact portion of a query or routine that is causing problems and only post that portion. In any case, avoid posting an entire query or stored routine.

All submitted code will be checked by automated tools for excessive similarity. Students may be asked to verbally explain their code or reproduce their code. In any confirmed case of plagiarism, *both parties* involved are deemed equally liable for the breach and will receive the same penalty. (In the past, students who have been given zero for the project because they copied from a friend seem to have been most upset by the fact that friend was *also* given zero.)

Please note that your lab facilitators are only permitted to give *general guidance*, help you identify *syntax errors* and refer you to lectures notes or other documentation.

Deliverables

We will be using `cssubmit` once again for the project submissions, but rather than one file containing all of the answers, the submission will consist of *multiple small files.*

You *must name* your files, functions and procedures *exactly* as specified — most of your code will be tested by an automated script, and so if the names are incorrect, then the script will not be able to find and run them.

You will be supplied with a small test-version of the database in order to test your code, but the actual marking will be performed on a different and larger version.

Project Setting

Scoops Ahoy is a chain of ice-cream stores that sells ice-cream in cones.Their database manager has moved interstate, but they feel that with only a limited range of queries ever used, it is not necessary to have a new permanent database administrator. As a result, you have been hired as a consultant to write some stored routines for the most commonly-used queries.

You are given a file `ScoopsAhoy.sql` that will construct and populate a test version of the real database. It contains sales data for the nine-month period from January 2019 to September 2019. The database contains tables with the names `Cone`, `conesInPurchase`, `Customer`, `customerPurchases`, `Purchase`, `Scoop` and `scoopsInCone`.

You are told some of the business rules:

1. A `Purchase` consists of one or more `Cones` purchased on a particular *day* at one of the three *stores*.
2. A `Cone` is made of 1–3 `Scoops` of (possibly different) flavours on either a wafer cone, a sugar cone or a waffle cone.
3. Each `Purchase` is made by a `Customer` who possibly uses a *discount coupon* for a certain percentage reduction in price.

In order to work with stored routines, it is useful to know the following commands to list and delete them!

```
SHOW PROCEDURE status WHERE DB = 'ScoopsAhoy';
SHOW FUNCTION status WHERE DB = 'ScoopsAhoy';
DROP PROCEDURE <procedurename> ;
DROP FUNCTION <procedurename> ;
```

Creating and verifying the sample `ScoopsAhoy` database

Using MySQL Workbench, choose `Run SQL Script` from the `File` menu, then navigate to the location of the `CreateScoopsAhoy.sql` file on your computer, and choose `Open`.

A second window will open up showing the first few lines of the file — just choose the `Run` option and the entire database should be created. Then change to that database using `USE ScoopsAhoy;` and *verify* the database using MySQL's checksum function.

```
CHECKSUM TABLE Cone;
+-----------------+-----------+
| Table           | Checksum  |
+-----------------+-----------+
| scoopsahoy.cone | 556540323 |
+-----------------+-----------+
```

The other tables have checksums 2520800458, 3746423535, 255292472, 2615166047, 205518241 and 1669916532 (in alphabetical order of table name).

Basic Questions

The first set of questions are basic SQL queries — for each of these you should write your query into a separate file with *exactly* the specified file name.

1. Using `ERDplus.com` prepare an ER Diagram describing the *current structure* of the database. Include only the entities, relationships and attributes that are *currently present* in the database. Include suitable cardinality constraints for each relationship, using common sense about how ice-cream shops work where it is not apparent from the database.
   (a) Upload to `cssubmit` an image file that has been produced by the "Export Image" function of ERDplus, and which you should *rename* to `ScoopsAhoy.png`.
   (b) PNG, or Portable Network Graphics is a file format for raster graphics images.
   (c) ERDPlus will save the file under some default name (on the Mac it is `image.png`)
2. Write a single SQL query that lists the total number of purchases made at Scoops Ahoy.
   (a) Upload to `cssubmit` a file with the name `Q2.sql` that contains *only* your query.
   (b) For the supplied sample database, the answer should be `3859` purchases.
3. Write a single SQL query that lists the month (by name) and the number of purchases made in that month for all the data in the database. The columns of the output table should be called `month` and `numPurchases`.
   (a) Upload to `cssubmit` a file with the name `Q3.sql` that contains *only* your query.
   (b) See `https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html`
   (c) The output should contain the following rows, not necessarily in any particular order

```
+-----------+--------------+
| month     | numPurchases |
+-----------+--------------+
| January   |          429 |
| February  |          432 |
```

4. Write a single SQL query that lists, for each customer in the database, their customer id and the total number of cones bought by this customer over the whole year, with the biggest buyers first.
   (a) Upload to `cssubmit` a file with the name `Q4.sql` that contains *only* your query.
   (b) For the supplied sample database, Customer #1 bought 473 cones.
5. Write the code necessary to create a *stored function* `numFlavours(coneNumber INT)` that returns the number of distinct scoop flavours in a particular cone.
   (a) Upload to `cssubmit` a file with the name `Q5.sql` that contains *only* the code required to create the function.
   (b) Your answer must start with `CREATE FUNCTION numFlavours(coneNumber INT)` (copy and paste this to be absolutely sure you do not misspell the function name).
   (c) Test your function by typing `SELECT numFlavours(1000);` (the answer is 2).
   (d) Do not include the testing code in your file.
   (e) This can be done in a single query, and so you do not need to change delimiters for this question.

Medium Questions

6. Write a single SQL query that lists the most expensive flavours in the `Scoop` table.
   (a) Upload to `cssubmit` a file with the name `Q6.sql` that contains *only* your query.
   (b) Check your query by simply looking at the table `Scoop`
   (c) The output should be a one-column table listing the flavour (or flavours) that cost the most

7. Write the code to create a *stored function* `isNutFree(coneNumber INT)` that returns `TRUE` if every part of that cone is nut-free, and `FALSE` otherwise. At the moment, the waffle cones are made with peanut oil, while the wafer and sugar cones are both nut-free, and Coconut and Macadamia are the only scoop flavours that are made with nuts.
   (a) Upload to `cssubmit` a file `Q7.sql` that contains *only* the code needed to create the function.
   (b) You must check that *each scoop of ice-cream* is nut-free and that the cone containing those scoops is nut-free.
   (c) You may find `https://dev.mysql.com/doc/refman/8.0/en/if.html` useful.
   (d) You may assume that the real database will not involve any more products containing nuts.

8. Write a single SQL query that lists the *customer email*, *store location* and *number of purchases* for each customer/store combination along with the total number of purchases by the customer over all stores.
   (a) Upload to `cssubmit` a file with the name `Q8.sql` that contains *only* your query.
   (b) Your output should contain the following rows

```
| esokullu@hotmail.com   | Cottesloe  |       70 |
| esokullu@hotmail.com   | Fremantle  |       67 |
| esokullu@hotmail.com   | City Beach |       59 |
| esokullu@hotmail.com   | NULL       |      196 |
```

Advanced Questions

9. First, adapt your code from Question 5 to make a *stored function* `numScoops(coneNumber INT)` that returns the *number of scoops* in the cone with the specified cone number.

   Next, write the code necessary to create a *stored procedure*

```
purchaseSummary(purchaseNum INT,
                OUT oneScoop INT,
                OUT twoScoop INT,
                OUT threeScoop INT)
```

   that analyses the cones in a single purchase, and sets the values of the three `OUT` variables to the number of one-scoop, two-scoop and three-scoop cones in that purchase.
   (a) Upload to `cssubmit` a file with the name `Q9.sql` that contains *only* the code required to create the two stored routines (the procedure and the supporting function).
   (b) Notice that `purchaseNum` is an `IN` parameter, while the other three are all `OUT` parameters.
   (c) Do not worry too hard about making this efficient, if you need to use three very similar SQL queries, then do so.
   (d) You may find it useful to use the stored function you created at the beginning of this question.
   (e) You should test your code by calling

```
CALL purchaseSummary(120,@one,@two,@three);
SELECT @one, @two, @three;
```

and you should get the output

```
+------+------+--------+
| @one | @two | @three |
+------+------+--------+
|    1 |    1 |      1 |
+------+------+--------+
```

10. Each month, Scoops Ahoy presents a "store-of-the-month" award to the store where the most purchases were made (just by *number* of purchases). Write a *stored procedure* `createMonthlyWinners()` that *creates a table* `MonthlyWinners` which has three columns, namely the month, the winning store and the number of purchases made in that month in that particular store, for all the months in the database.

    (a) Upload to `cssubmit` a file with the name `Q10.sql` that contains *only* the code necessary to create this stored procedure.

    (b) The *procedure itself* will not return anything, but will just *create a new table* in the same database that has the correct values in it.

    (c) Make sure your procedure works even if you call it several times in succession (mentally imagine this procedure being called once every month).

    (d) Use all the data in the database, ignoring the fact that the database may only have data for part of the final month.

    (e) Make sure you take into account the fact that the real database may have data covering *several years*, and so the first column should contain values like `'December 2018'`, `'January 2019'`.

Bonus Question

This is a bonus question for those enjoying the challenge of writing these queries, functions and procedures. The marks allocated to this can only be used to offset marks *lost* in the previous questions, so you can view it as an "insurance policy" in case you make a mistake earlier.

If you do not have time or inclination to tackle this question, then you can simply omit it and still get full marks if all of the other questions are correct.

11. Write the code necessary to create a *stored function* `costOfPurchase(purchaseNumber INT)` that will calculate the cost of a purchase according to the following rather complicated business rules.

    Each *cone* in a purchase is costed as:

    - A *base cost* which is 50c on Monday-Friday, 100c on Saturday, and 150c on Sunday.
    - The *cone cost* which is found in the `Cone` table.
    - The sum of the *scoop costs* for the individual scoops (found in `Scoop`).
    - A *multiscoop discount* of 50c for 2-scoop cones 150c for 3-scoop cones.

    The final *purchase cost* is then obtained by summing the cost of all of the cones, and reducing this value according to the specific customer/purchase *discount* found in the `customerPurchases` table, and rounding to the nearest cent.

    - Upload to `cssubmit` a file with the name `Q11.sql` that contains *only* the code necessary to create this stored procedure.