DATABASE SYSTEMS

Coursework 1

Dr Paolo Guagliardo

Released: Thu 13 October 2015 – **Due: Fri 28 October 2016 at 16:00**

Before you begin, please read carefully the policy on Late coursework and Academic misconduct and related links, in particular the rules for the publication of solutions to coursework.

Database schema. For this assignment we will use the schema available at:

`<dbshome>/assignment1/schema.sql` (MD5: `e73e2831f60dd4b8a7ef03b6b75840e5`)

where `<dbshome>` is `/afs/inf.ed.ac.uk/group/teaching/dbs`. This is a slight variation of the schema we have already seen in tutorial 1. You can create the schema in your PostgreSQL database on DICE with the command: `psql -h pgteach -1 -f <dbshome>/assignment1/schema.sql`. The schema file is annotated with comments consisting of directives for `datafiller` to generate more realistic-looking test data. These directives will be completely ignored by the DBMS and do not in any way constrain the database.

Assignment. Write the following queries in SQL.

(01) Invoices issued after their due date. Return all attributes.

(02) Invoices that were issued before the date in which the order they refer to was placed. Return the ID of the invoice, the date it was issued, the ID of the order associated with it and the date the order was placed.

(03) Orders that do not have a detail and were placed before 6 September 2016. Return all attributes.

(04) Customers who have not placed any orders in 2016. Return all attributes.

(05) ID and name of customers and the date of their last order. For customers who did not place any orders, no rows must be returned. For each customer who placed more than one order on the date of their most recent order, only one row must be returned.

(06) Invoices that have been overpaid. Observe that there may be more than one payment referring to the same invoice. Return the invoice number and the amount that should be reimbursed.

(07) Products that were ordered more than 10 times in total, by taking into account the quantities in which they appear in the order details. Return the product code and the total number of times it was ordered.

(08) Products that are usually ordered in bulk: whenever one of these products is ordered, it is ordered in a quantity that *on average* is equal to or greater than 8. For each such product, return product code and price.

(09) Total number of orders placed in 2016 by customers of each country. If all customers from a specific country did not place any orders in 2016, the country will not appear in the output.

(10) For each order without invoice, list its ID, the date it was placed and the total price of the products in its detail, taking into account the quantity of each ordered product and its unit price. Orders without detail must not be included in the answers.

Submission instructions.

- Each query must be written in a text file named `<xx>.sql` where `<xx>` is the two-digit number in the list of queries above. For example, the first query will be written in file `01.sql` and the last one in file `10.sql`.
- Each file consists of a single SQL statement, terminated by semicolon (`;`). Submitted files that do not contain *exactly one semicolon* will be discarded when the submission is processed and consequently they will not be assessed (as if they were not submitted). Please pay attention to this; even if it looks like a trivial detail, it is not.
- Submission is via the `submit` command on DICE.[1] You can submit all your files as follows:

  ```
  submit dbs 1 01.sql 02.sql 03.sql 04.sql 05.sql 06.sql 07.sql 08.sql 09.sql 10.sql
  ```

  You can also submit your files one by one, or in smaller groups, and in any order. For example:

  ```
  submit dbs 1 03.sql
  submit dbs 1 01.sql
  submit dbs 1 04.sql 02.sql
  ```

  Files can be submitted more than once, in which case the previously submitted version will be overwritten (`submit` will ask you whether you want to proceed).
- Before submitting your files, you should check that they run smoothly in PostgreSQL on DICE[2] with the following command:

  ```
  psql -h pgteach -f <file>.sql
  ```

- The final deadline for the submission is on **Friday 28 October 2016 at 16:00**. As file history is not recorded, files whose latest version is submitted after the deadline (as reported in the log) will not be considered for assessment, unless you have been granted a coursework extension.

Assessment. Your queries will be executed on 5 database instances *without nulls*, generated by datafiller using the annotated schema. Each query is worth 10 marks, which are allocated as follows:

- 2 marks are given for each test database on which the query returns *all and only* the correct answers.
- 1 mark is given for each test database on which the query returns *at least 50%* of the correct answers (but not all of them) and *no wrong answers.*

The queries will not be assessed for their performance, as long as they terminate after a "reasonable" time; each query should not take more than a few seconds to run.[3] Style will not be assessed either: you could write a query on single line, but this is not recommended for your own sake.

Test data. Two of the five instances on which your queries will be assessed are publicly available at:

- `<dbshome>/assignment1/data/db1.sql` (MD5: `7522068eea7c4e6ffb56f5a30af73d12`)
- `<dbshome>/assignment1/data/db2.sql` (MD5: `6b8d14691f991ac4bf7585157306247f`)

You can load these instances into your PostgreSQL database on DICE with the following command:

```
psql -h pgteach -1 -f <dbshome>/assignment1/data/db<n>.sql
```

[1] See how to submit a practical exercise.
[2] See how to use PostgreSQL on DICE machines.
[3] Executing all of the 10 queries, computing the answers and writing the CSV files took me 176 milliseconds in total.

where `<n>` is the instance number (1 or 2). Please note that the above command will delete existing tables you may already have in your database. Assuming the `datafiller` script is executable and it resides in your system's path, you can generate additional instances for your private tests by issuing the following command from the directory `<dbshome>/assignment1/`:

```
datafiller --drop schema.sql > <path/to/writable/file.sql>
```

Instead of writing the generated instance to a file that you will then have to import into PostgreSQL, you can directly load the database by piping the output of `datafiller` into `psql` as follows:

```
datafiller --drop schema.sql | psql -h pgteach
```

Query answers. The answers your queries are supposed to return on the public database instances are available in CSV format at:

- `<dbshome>/assignment1/answers/db1/`

  `01.sql` (MD5: `9e4b6f36376619e46cc6d8bbf1b8e59b`)
  `02.sql` (MD5: `a3f9fc2800bbf37a891c4a19bcf26e0f`)
  `03.sql` (MD5: `98fdebbc164d8cbb96a82e8e85929a06`)
  `04.sql` (MD5: `62f8fae73c91d14c7f24d1afdcd31445`)
  `05.sql` (MD5: `f1d6bfb7dbe57b8f7bb03e29ac7263b3`)
  `06.sql` (MD5: `7a44a7b4542c21bb1848a6250d23f2bf`)
  `07.sql` (MD5: `3e0e2580a84bd75f4d95b2cd998a7a4b`)
  `08.sql` (MD5: `15896db92ae420ecbe21051d67547b00`)
  `09.sql` (MD5: `c9ae0fafd17ca34271669e9840f40090`)
  `10.sql` (MD5: `65344caf2f9b03ec441c79578a1d949d`)

- `<dbshome>/assignment1/answers/db2/`

  `01.sql` (MD5: `143269c4fd085206f3b4fa23e5ad3501`)
  `02.sql` (MD5: `1c705c636253e1bec8d0f4b58e477c56`)
  `03.sql` (MD5: `0e0a63baf1e33825d831946e8cdd018d`)
  `04.sql` (MD5: `1462a196f1e92cf02b34e3caac64d670`)
  `05.sql` (MD5: `bede59d0f0e90e8b0615e05868db87a8`)
  `06.sql` (MD5: `c1d56a35e8fbbae752b2993d0f53594e`)
  `07.sql` (MD5: `62634dd3bb15699d672a14e218964ffc`)
  `08.sql` (MD5: `22400a46993b71483925b28a87c99498`)
  `09.sql` (MD5: `f5d5ea733ab25380deac753aa5948721`)
  `10.sql` (MD5: `a863217c0bb826d6bdacbe0ee0b346fc`)

The order in which the rows appear in the answer to your queries is irrelevant for this assignment (no ordering is enforced on the answers, so the DBMS will output rows in an arbitrary order). The names of the columns in the answers are also irrelevant (the above CSV files have no header) so they can be renamed as you wish in the `SELECT` clause. What is **important** is the number of columns and the order in which they appear in the output: $(1, 2)$ is not the same as $(2, 1, 1)$. Your query gives a fully correct answer if it outputs all and only the rows (with repetitions, if that's the case) listed in the corresponding CSV file, no matter on which line.

Other comments. All queries can be written using the constructs we will have seen in class by the end of week 4 (or the very beginning of week 5). Just keep it simple and all will be well. Good luck!