Lab 8: PL/SQL

Due date: Friday, June 5 11:59. There is also a grace period for this lab, which extends until 10:00am, June 6.

Assignment Preparation

The main part of this assignment is to be done in teams. You can organize yourselves in teams of 2-3 people each.

Lab Assignment

You are asked to form groups of 2-3 people. The goal of each group is to build simple PL/SQL packages for two course databases and create some PL/SQL and SQL code that tests your package.

To test functions from your package you will write SQL queries that use them. To test procedures, you will write anonymous PL/SQL blocks.

The list of tasks is outlined below.

Note: We might not get to talking about packages until Tuesday, June 1 class. If that is the case, work on creating individual procedures and functions to specifications below for now. You will then turn them into packages next week.

BAKERY database

Create a package `bakery` which contains the following functions and procedures.

Please Note: I use anchored types in the specifications below. These anchor types reference attribute names from **my implementation of the databases**. If in your implementation, attribute names are **different** you will use your attribute names in such situations. E.g., if the `Food` column of the goods.csv file is named FType in your database, then you will reference goods.FType%type rather than goods.Food%type. (Same thing applies to the table names).

- function item_revenue() takes as input the food type, and the food flavor, as well as a pair of dates, and outputs the total revenue generated by this specified item in the given time interval (the dates are inclusive). The full spec is:

```
function item_revenue(myFood in goods.food%type,
                      myFlavor in goods.flavor%type,
                      stDate in receipts.SaleDate%type,
                      endDate in receipts.SaleDate%type)
                 return goods.price\%type;
```

- function customer_revenue() takes as input the last name of the customer and a pair of dates and outputs the total revenue generated by the specified customer in the given time interval (the dates are inclusive). The full spec is:

```
function customer_revenue(myLname in customer.LastName%type,
                          stDate in receipts.SaleDate%type,
                          endDate in receipts.SaleDate%type)
                  return goods.price\%type;
```

- procedure purchase() takes as input the last name of the customer, a date, the description of a bakery item (food and flavor information). The procedure creates a database record of the specified customer's purchase of the given bakery item. The receipt number is generated as follows: find the largest receipt number in the database, and add 10 to it. The full spec is:

```
procedure purchase(myLname in customer.LastName%type,
                   myDate in receipts.SaleDate%type,
                   myFood in goods.food%type,
                   myFlavor in goods.flavor%type);
```

  Note: this procedure results in creation of a new purchase of a single item in the BAKERY database. This is NOT a generic "create a new receipt for a multi-item purchase" procedure.

Testing. To test your two functions, write a number of SQL queries that use them. For example, finding the customer who spent the most at the bakery can be done using customer_revenue() function. Your test file should contain comment blocks specifying what each query is designed to return. Please use queries that limit the output to only a few rows. To test your procedure, you can write an anonymous block that calls the procedure and have a collection of SQL queries that establish that the database has, indeed been changed.

AIRLINES database

Create a package `airlines` which contains the following functions and procedures.

- procedure `direct_connect()` takes as input a pair of airport codes (source and destination). If at least one direct flight between the two airports exists, for each such flight the procedure outputs the flight number and the name of the airline servicing the flight. If no direct flights are found, the procedure reports `"No direct flights found."` The full spec is:

```
procedure direct_connect( mySource in flights.source%type,
                          myDest   in flights.destination%type);
```

- procedure `one_stop_connect()` takes as input a pair codes (source and destination). If at least one direct flight between the two airports exists, for each such flight the procedure outputs the flight number and the name of the airline servicing the flight. If there are no direct flights, the procedure checks if it is possible to get from the source airport to the destination airport via a single transfer point (i.e., via two flights). For each such transition, the procedure shall report the source airport, the flight number of the first flight and the airline servicing it, the intermediate airport, the flight number of the second flight and the airline servicing it and, finally, the destination airport. The full spec is:

```
procedure one_stop_connect( mySource in flights.source%type,
                          myDest   in flights.destination%type);
```

- function `num_flights()` takes as input an airport code and returns back the number of flights to/from the airport (note that the database is organized so that the number of incoming flights is the same as the number of the outgoing flights). The full spec is:

```
function num_flights( myAirport in flights.source%type)
                    return binary_integer;
```

Testing. The two procedures can be tested in a straightforward manner — create one or more anonymous blocks that call them. Both procedures are supposed to print their results to screen. To test the function, write a few SQL queries that use it. For example, you can output all airports sorted by the number of outgoing flights, or output the airports with exactly 57 outgoing flights. Please include detailed comments before every query and every anonymous block explaining what you intended them to do.

Extra Credit

Extra credit will be given for implementation of the following procedure in the package `airlines`:

- procedure `reachable()` takes as input an airport code and an integer number and outputs the list of all airports *reachable from the given airport in the num of flights that is less than or equal to the input number*. If the input integer is 0, then the procedure outputs **all airports** reachable from the given airport in any number of steps.

  Notes: Each airport name shall be reported exactly once. The input airport shall not appear in the output.

  The full spec is:

```
procedure reachable(myAirport in flights.source%type,
                    numSteps  in binary_integer);
```

Submission Instructions

You have to prepare two .sql package files: `BAKERY-package.sql` and `AIRLINES-package.sql`. For each of the two datasets, also submit the `-setup.sql`, `-insert.sql` and `-cleanup.sql` files. For your test files use the `<DATASET>-testXX.sql` syntax, e.g., `AIRLINES-test01.sql`, `AIRLINES-test02.sql`, etc., and provide a `<DATASET>-test.sql` file which will essentially contain a sequence of

```
start <DATASET>-testXX.sql
```

commands to run your entire test (you should assume that your dataset is created and your package is loaded by the time the test file runs).

Each dataset is to be submitted in a separate directory (BAKERY and AIRLINES respectively). Create an archive of your entire submission and use `handin` as shown below to submit it. Name your archive `lab08.zip` or `lab08.tar.gz`. Submit as follows:

```
handin dekhtyar-grader lab08-365 lab08.ext
```

Each file must contain a comment block at the top listing all members of the group.

If you have implemented the extra credit assignment (which is worth 1% point towards the final grade), include a file titled `EXTRA_CREDIT` in your submission. Put any information concerning the implementation of the extra credit in this file. Note, that it is possible to earn partial credit as long as the submitted code compiles without errors and passes some of the tests. Note also, that extra credit will be given to the entire group, regardless of who actually implemented the extra credit functionality.