School of Information Technology  
Department of Computer Science

COS326 Database Systems: Practical 3 PostgreSQL ORDBMS  
Handout date: 7 August 2015   Due date:  14 August 2015        Marks:  50

Objectives  
1. Practice the use of advanced features of PostgreSQL ORDBMS.  
2. Learn how to implement PL/pgSQL functions, triggers, and database constraints using triggers.

When you are done:  
1. You must submit files, named:  
   a. **EERD.pdf** which contains the database design as an EER diagram. Your name and student number must appear in this document.  
   b. **CreateStatements.sql** which contains all statements necessary to create the database ‘objects’ i.e. user defined types, sequences, tables, functions and triggers.  
   c. **InsertQueries.sql** which contains all statements that add to the content of the database (INSERT statements).  
   d. **TestTriggerQueries.sql** which contains all statements for testing the triggers (INSERT, UPDATE, and DELETE statements).  
   e. **SelectQueries.sql** which contains all statements that provide reports from the database (SELECT statements)  
   f. Compress the above documents into an archive and upload it to the CS website before the due date/time. The file name for the archive must have your student number as part of the file name, e.g. xxxxx-prac3.pdf where xxxxx is your student number.

2. Book a marking timeslot session in the practical session of Friday 14 August.

Task 1: Practice the implementation of PL/pgSQL functions  
(not for marks)

In practical exercise 2, you created SQL functions with the following syntax:

```
CREATE FUNCTION functionname(argtype1, argtype2, .. ) RETURNS returntype AS $$
SELECT expression to compute AS functionname;$$
LANGUAGE SQL;
```

In this practical exercise you will practice the use of PL/pgSQL functions. PL/pgSQL is the procedural language for the PostgreSQL DBMS (refer to chapter 39 of the PostgreSQL reference document). PL/pgSQL provides a subset of the features specified for PL/SQL which is the standard procedural language for SQL (refer to the textbook chapter 8, section 8.1).
The syntax of a PL/pgSQL function is as follows:

```
CREATE FUNCTION functionname(argtype1, argtype2, .. ) RETURNS returntype AS $$
DECLARE   --comment 1: declare vars only if necessary
    variable1 vartype1, variable2 vartype2,..... ;
BEGIN
    -- comment 2: code to compute returnvalue
    RETURN returnvalue ;
    --comment 3: returnvalue must be of returntype
END;
$$ LANGUAGE plpgsql;
```

To be done:

a. Read chapter 39, sections 39.1 to 39.6 of the PostgreSQL 9.2 reference document to find out the details of all the programming constructs that one can use to write the code for comment 2.

b. Select any three of the functions you implemented for practical exercise 2 (as SQL functions) and convert them to PL/pgSQL functions.

**Task 2: Practice the implementation of PL/pgSQL triggers and trigger functions/procedures (not for marks)**

A trigger defines an action that the DBMS should take when some event occurs. Most commonly the event arises due to an operation (INSERT, UPDATE, or DELETE) on a database table. A trigger may be defined as a BEFORE trigger or an AFTER trigger. A BEFORE trigger executes before the specified operation while an AFTER trigger executes after the specified operation. A trigger may be used to enforce some referential constraints, to enforce complex constraints (business rules) or to audit changes to data (refer to chapter 8, section 8.3 of the textbook). The textbook section 8.3 gives the syntax of the CREATE TRIGGER statement as defined in the SQL standard. Specific to PostgreSQL, a function (trigger procedure) is associated with a trigger (refer to the CREATE TRIGGER statement description in the PostgreSQL reference document). A simple syntax of a BEFORE trigger that is executed for each row of a table is as follows:

```
CREATE TRIGGER triggername
    BEFORE tableoperation(s) ON tablename --- table operation(s) is/are INSERT or UPDATE or
        INSERT OR UPDATE, or DELETE
    FOR EACH ROW --- enables access to the NEW and OLD table records
    EXECUTE PROCEDURE triggerfunction( );
```

Alternatively the CREATE TRIGGER statement may specify the condition under which the trigger function (procedure) should be executed as follows:

```
CREATE TRIGGER triggername
    BEFORE tableoperation ON tablename --- table operation is INSERT or DELETE or UPDATE
    FOR EACH ROW --- enables access to NEW and OLD
    WHEN (condition for function execution)
    EXECUTE PROCEDURE triggerfunction( );
```
A simple syntax of a trigger function (procedure) is as follows:

```sql
CREATE FUNCTION triggerfunction() RETURNS TRIGGER AS $$
DECLARE
  -- comment 1: declare vars only if necessary
  variable1 vartype1, variable2 vartype2, ...... ;
BEGIN
  -- comment 2: code to compute any required values goes here
  -- in the code, use NEW.attribute1, NEW.attribute2,... to access values of new input row
  -- e.g for INSERT, UPDATE
  -- use OLD.attribute1, OLD.attribute2,... to access values of existing row in table
  -- e.g for DELETE
  IF (condition to test for unwanted situation) THEN
    RAISE EXCEPTION 'error message for exception';
  END IF;
END;
RETURN NEW;
$$ LANGUAGE plpgsql;

To be done:

Practice the creation of a trigger by doing the following:

a. Create a simple table without specifying a primary key, e.g.
   CREATE TABLE City (name text, population real, areaSqKm int);

b. Insert some records into the table, ensuring that there are no duplicate records with the same (city) name.

c. Create a trigger that will enforce the (not-null and unique) primary key constraint on the City table.

d. Test the trigger using INSERT statements that attempt to violate the primary key constraint.

Task 3: Creation of the database ‘objects’ [30 marks]

Problem statement:

The Administration department at the Hatfield campus of the university has to keep records of all teaching venues: lecture rooms and laboratories. For every teaching venue the building code, floor number, room number, size (length, width), and sitting capacity are recorded. For a laboratory, the details of the laboratory manager and laboratory type are additionally recorded. For a lecture room, the available equipment is additionally recorded.
In this practical exercise, the specification is extended as follows: **Bookings** for activities in the **Teaching venues** need to be recorded. For every **booking**, the activity name (lecture, practical, test, exam), activity room (booked lecture room or laboratory which is-a teaching venue), activity date, start time, end time and contact person are recorded. It is necessary to ensure that the activity room corresponds to a teaching venue already recorded in the Teaching venue table (referential integrity). It is also necessary to ensure that a booking for a ‘lecture’ activity can only be made for a **lecture room**, and a booking for a ‘practical’ activity, can only be made for a laboratory.

A straightforward object-relational database design for the above specification would be to implement the tables: **TeachingVenue**, **LectureRoom** (inherits TeachingVenue), **Laboratory** (inherits TeachingVenue), and **Booking** with a foreign key that references a row in the **TeachingVenue** table (by specifying REFERENCES TeachingVenue). Unfortunately, in PostgreSQL 9.2 inheritance and foreign keys are not compatible. (refer to chapter 5 section 5.8.1 of the PostgreSQL 9.2 reference document and the blog [http://ledgersmbdev.blogspot.com/2012/08/postgresql-or-modelling-part-3-table.html](http://ledgersmbdev.blogspot.com/2012/08/postgresql-or-modelling-part-3-table.html) for details). So, for the database implementation in this practical exercise we will use triggers to enforce the referential integrity between the tables **TeachingVenue** and **Booking**. The following table summarises the above aspects of the required database design for this practical exercise. Some of these aspects are similar to practical exercise 2.

<table>
<thead>
<tr>
<th>Database ‘object’ type</th>
<th>Database ‘objects’</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQUENCE</td>
<td>venueSeq</td>
<td>generates surrogate (primary) keys for VENUE table starting at 101</td>
</tr>
<tr>
<td></td>
<td>bookingSeq</td>
<td>generates surrogate (primary) keys for the BOOKING table starting at 1001</td>
</tr>
<tr>
<td>TYPE</td>
<td>BuildingCodeType</td>
<td>codes for the buildings IT, EMS, HSB, ENG</td>
</tr>
<tr>
<td></td>
<td>RoomType</td>
<td>ROW type holding (building code, floor, room) building code is of type BuildingCodeType</td>
</tr>
<tr>
<td></td>
<td>EquipmentType</td>
<td>description of equipment with values: projector, PA system, safe.</td>
</tr>
<tr>
<td></td>
<td>LaboratoryType</td>
<td>description of laboratory with values: computer, chemistry, physics</td>
</tr>
<tr>
<td></td>
<td>ActivityType</td>
<td>description of activity with values: lecture, practical, test, exam.</td>
</tr>
<tr>
<td></td>
<td>TitleType</td>
<td>description of titles with values: Ms, Msc, Miss, Mr, Mnr, Dr, Prof</td>
</tr>
<tr>
<td></td>
<td>PersonType</td>
<td>ROW type holding (title, firstName, surname)</td>
</tr>
<tr>
<td>TABLE</td>
<td>TeachingVenue</td>
<td>with attributes: (venueKey, venueCode, length, width, seats) venueCode is of type RoomType</td>
</tr>
<tr>
<td></td>
<td>LectureRoom</td>
<td>inherits TeachingVenue and additionally has attribute: equipmentList which is an array of type EquipmentType</td>
</tr>
<tr>
<td></td>
<td>Laboratory</td>
<td>inherits TeachingVenue and additionally has attributes: (manager, labType)</td>
</tr>
<tr>
<td></td>
<td>Booking</td>
<td>the bookings for the activities in the lecture rooms. Attributes are: (bookingKey, activityName, activityRoom, activityDate, startTime, endTime, contactName) activityName is of type ActivityType, activityRoom is of type RoomType contactName is of type PersonType</td>
</tr>
<tr>
<td></td>
<td>DeletedBooking</td>
<td>When a booking is deleted, the deleted record is copied to this table. Attributes are: (all the attributes of Booking, plus date-and-time of deletion, userid of the user who deleted the record)</td>
</tr>
</tbody>
</table>
To be done:

1. Create an Enhanced Entity Relationship model (EER) diagram to show the above relationships. Refer to textbook chapter 13. (2 marks)

2. Write SQL statements and PL/pgSQL code to create all the database ‘objects’ in the above table and any other ‘objects’ you consider to be necessary.

3. Create a database in PostgreSQL called VenuesDBprac3 and run all the SQL statements in (2) above to create the database ‘objects’. Note: marks for part (2) will only be awarded if the database ‘objects’ actually get created. Marks will be awarded as follows:
   a. Sequences and types: 2 marks
   b. Tables: 4 marks
   c. Re-implementation of prac 2 functions using PL/pgSQL: 4 marks
   d. Triggers: 8 marks
   e. Functions for triggers: 10 marks
Task 4: Inserting data into the Database tables  [4 marks]

Use the `INSERT INTO` SQL statement to add the following data into the database. Execute some `SELECT` statements to confirm that you entered the data correctly.

<table>
<thead>
<tr>
<th>Teaching Venue Type</th>
<th>Attribute values: note that the values of attribute venue key are generated by the SEQUENCE you created</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>venue key</td>
</tr>
<tr>
<td>laboratory</td>
<td>('IT', 2, 10)</td>
</tr>
<tr>
<td>laboratory</td>
<td>('IT', 2, 15)</td>
</tr>
<tr>
<td>lecture room</td>
<td>('IT', 4, 2)</td>
</tr>
<tr>
<td>lecture room</td>
<td>('IT', 2, 24)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bookings: note that the values of attribute bookingKey are generated by the SEQUENCE you created</th>
</tr>
</thead>
<tbody>
<tr>
<td>booking Key</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>lecture</td>
</tr>
<tr>
<td>test</td>
</tr>
<tr>
<td>practical</td>
</tr>
<tr>
<td>exam</td>
</tr>
</tbody>
</table>

Task 5: Insert, update and delete operations to test the triggers  (10 marks)

Write and test SQL statements to do the operations listed in the table below. Marks will only be awarded if the SQL statement produces the correct output.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Table</th>
<th>values (… means you are free to choose values for the remaining attributes)</th>
<th>Testing trigger (for)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>Booking</td>
<td>‘lecture’, ('IT', 4, 20), 2015-08-4, 12:30, 14:30, ……</td>
<td>valid_room (referential integrity)</td>
</tr>
<tr>
<td>INSERT</td>
<td>Booking</td>
<td>‘test’, ('IT', 2, 24), 2015-08-5, 08:00, 10:00, ……</td>
<td>valid_insert_update (double booking)</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Booking</td>
<td>change the <code>startTime</code> value for the exam booking of the table above (from 10:30) to 09:30</td>
<td>valid_insert_update (double booking)</td>
</tr>
<tr>
<td>DELETE</td>
<td>Booking</td>
<td>delete the exam booking</td>
<td>audit_delete_booking (auditing)</td>
</tr>
<tr>
<td>INSERT or UPDATE</td>
<td>Booking</td>
<td>values of your choice</td>
<td>'one other trigger’ that you implemented for Task 3</td>
</tr>
</tbody>
</table>
Task 6: Accessing data from two tables (6 marks)

1. Write a query that will list the details of all laboratories. The query result should show the venueKey, venueCode, roomarea(…), seats, personName(…), labtype. (1 mark)

2. Write a query that will list the details of all lecture rooms. The query result should show the venueKey, venueCode, roomarea(length,width), seats, hasProjector(…), hasPAsystem(…), hasSafe(…). (1 mark)

3. Write a query that will list the details of all bookings for all lecture rooms that have a projector. The query result should display the following columns: activityName, activityRoom, activityDate, startTime, endTime, contactName. (2 marks)

4. Write a query to report on the deletion of bookings in the IT building. The result should show the database user name, date and time (of deletion of booking), and all the booking details as they originally appeared in the Booking table. (2 marks)