COMP3311 21T1 Database Systems

Week 05
Stored Functions in SQL and PLpgSQL

[Show with no answers] [Show with all answers]

1. Write a simple PLpgSQL function that returns the square of its argument value. It is used as follows:

```
mydb=> select sqr(4);
 sqr
-----
  16
(1 row)

mydb=> select sqr(1000);
   sqr
---------
 1000000
(1 row)
```

   Could we use this function in any of the following ways?

```
select sqr(5.0);
select(5.0::integer);
select sqr('5');
```

   If not, how could we write a function to achieve this?

   [show answer]

2. Write a PLpgSQL function that "spreads" the letters in some text. It is used as follows:

```
mydb=> select spread('My Text');
     spread
----------------
 M y   T e x t
(1 row)
```

   [show answer]

3. Write a PLpgSQL function to return a table of the first *n* positive integers.

   The fuction has the following signature:

```
create or replace function seq(n integer) returns setof integer
```

and is used as follows:

```
mydb=> select * from seq(5);
 seq
-----
   1
   2
   3
   4
   5
(5 rows)
```

[show answer]

4. Generalise the previous function so that it returns a table of integers, starting from *lo* up to at most *hi*, with an increment of *inc*. The function should also be able to count down from *lo* to *hi* if the value of *inc* is negative. An *inc* value of 0 should produce an empty table. Use the following function header:

```
create or replace function seq(lo int, hi int, inc int) returns setof integer
```

and the function would be used as follows:

```
mydb=> select * from seq(2,7,2);
 val
-----
   2
   4
   6
(3 rows)
```

Some other examples, in a more compact representation:

```
seq(1,5,1)   gives   1  2  3  4  5
seq(5,1,-1)  gives   5  4  3  2  1
seq(9,2,-3)  gives   9  6  3
seq(2,9,-1)  gives   empty
seq(1,5,0)   gives   empty
```

[show answer]

5. Re-implement the `seq(int)` function from above as an **SQL function**, and making use of the generic `seq(int,int,int)` function defined above.

   [show answer]

6. Create a factorial function based on the above sequence returning functions.

   ```
   create function fac(n int) returns integer
   ```

   Implement it as an **SQL function** (not a PLpgSQL function). The obvious solution to this problem requires a `product` aggregate, analogous to the `sum` aggregate. PostgreSQL does not actually have a `product` aggregate, but for the purposes of this question, you can assume that it does, and has the following interface:

   ```
   product(list of integers) returns integer
   ```

   [show answer]

Use the old Beers/Bars/Drinkers database in answering the following questions. A summary schema for this database:

```
Beers(name:string, manufacturer:string)
Bars(name:string, address:string, license#:integer)
Drinkers(name:string, address:string, phone:string)
Likes(drinker:string, beer:string)
Sells(bar:string, beer:string, price:real)
Frequents(drinker:string, bar:string)
```

Primary key attributes are in **bold**. Foreign key attributes are in ***bold italic***.

The examples below assume that the user is connected to a database called `beer` containing an instance of the above schema.

7. Write a PLpgSQL function called `hotelsIn()` that takes a single argument giving the name of a suburb, and returns a text string containing the names of all hotels in that suburb, one per line.

   ```
   create function hotelsIn(_addr text) returns text
   ```

   The function is used as follows:

   ```
   beer=> select hotelsIn('The Rocks');
       hotelsin
   -----------------
    Australia Hotel+
   ```

```
 Lord Nelson    +

(1 row)
```

Can you explain what the `'+'` at the end of each line is? And why it says `(1 row)`?

Note that the output from functions returning a single `text` string and looks better if you turn off `psql`'s output alignment (via `psql`'s **`\a`** command) and column headings (via `psql`'s **`\t`** command).

Compare the aligned output above to the unaligned output below:

```
beer=> \a
Output format is unaligned.
beer=> \t
Showing only tuples.
beer=> select hotelsIn('The Rocks');
Australia Hotel
Lord Nelson
```

From now on, sample outputs for functions returning `text` will assume that we have used **`\a`** and **`\t`**.

[show answer]

8. Write a new PLpgSQL function called `hotelsIn()` that takes a single argument giving the name of a suburb and returns the names of all hotels in that suburb. The hotel names should all appear on a single line, as in the following examples:

```
beer=> select hotelsIn('The Rocks');
Hotels in The Rocks:  Australia Hotel  Lord Nelson

beer=> select hotelsIn('Randwick');
Hotels in Randwick:  Royal Hotel

beer=> select hotelsIn('Rendwik');
There are no hotels in Rendwik
```

[show answer]

9. Write a PLpgSQL procedure `happyHourPrice` that accepts the name of a hotel, the name of a beer and the number of dollars to deduct from the price, and returns a new price. The procedure should check for the following errors:

   - non-existent hotel (invalid hotel name)
   - non-existent beer (invalid beer name)
   - beer not available at the specified hotel

- invalid price reduction (e.g. making reduced price negative)

Use `to_char(price,'$9.99')` to format the prices.

```
beer=> select happyHourPrice('Oz Hotel','New',0.50);
There is no hotel called 'Oz Hotel'

beer=> select happyHourPrice('Australia Hotel','Newer',0.50);
There is no beer called 'Newer'

beer=> select happyHourPrice('Australia Hotel','New',0.50);
The Australia Hotel does not serve New

beer=> select happyHourPrice('Australia Hotel','Burragorang Bock',4.50);
Price reduction is too large; Burragorang Bock only costs $ 3.50

beer=> select happyHourPrice('Australia Hotel','Burragorang Bock',1.50);
Happy hour price for Burragorang Bock at Australia Hotel is $ 2.00
```

[show answer]

10. The `hotelsIn` function above returns a formatted string giving details of the bars in a suburb. If we wanted to return a table of records for the bars in a suburb, we could use a view as follows:

```
beer=> create or replace view HotelsInTheRocks as
    -> select * from Bars where addr = 'The Rocks';
CREATE VIEW
beer=> select * from HotelsInTheRocks;
      name       |   addr    | license
-----------------+-----------+---------
 Australia Hotel | The Rocks |  123456
 Lord Nelson     | The Rocks |  123888
(2 rows)
```

Unfortunately, we need to specify a suburb in the view definition. It would be more useful if we could define a "parameterised view" which we could use to generate a table for any suburb, e.g.

```
beer=> select * from HotelsIn('The Rocks');
      name       |   addr    | license
-----------------+-----------+---------
 Australia Hotel | The Rocks |  123456
 Lord Nelson     | The Rocks |  123888
(2 rows)
```

```
beer=> select * from hotelsIn('Coogee');
       name       |  addr   | license
------------------+---------+---------
 Coogee Bay Hotel | Coogee |  966500
(1 row)
```

Such a parameterised view can be implemented via an SQL function, defined as:

```
create or replace function hotelsIn(text) returns setof Bars
as $$ ... $$ language sql;
```

Complete the definition of the SQL function.

[show answer]

11. The function for the previous question can also be implemented in PLpgSQL. Give the PLpgSQL definition. It would be used in the same way as the above.

[show answer]

Use the Bank Database in answering the following questions. A summary schema for this database:

```
Branches(location:text, address:text, assets:real)
Accounts(holder:text, branch:text, balance:real)
Customers(name:text, address:text)
Employees(id:integer, name:text, salary:real)
```

The examples below assume that the user is connected to a database called `bank` containing an instance of the above schema.

12. For each of the following, write both an SQL and a PLpgSQL function to return the result:

    a. salary of a specified employee

    [show answer]

    b. all details of a particular branch

    [show answer]

    c. names of all employees earning more than $*sal*

    [show answer]

    d. all details of highly-paid employees

[show answer]

13. Write a PLpgSQL function to produce a report giving details of branches:

    - name and address of branch
    - list of customers who hold accounts at that branch
    - total amount in accounts held at that branch

    Use the following format for each branch:

```
Branch: Clovelly, Clovelly Rd.
Customers:  Chuck Ian James
Total deposits: $   8860.00
```

[show answer]

Use the following database schema, which is somehwat similar to the schema for Assignment 2. The schema is too large to give a complete summary here, but we provide some details for some tables:

```
Term(id:integer, year:integer, session:('S1','S2','X1','X2'), ...)
Subject(id:integer, code:text, ..., name:text, ... uoc:integer, ...)
Course(id:integer, subject:integer, term:integer, lic:integer, ...)
OrgUnit(id, utype, name, longname, ...)
OrgUnitType(id, name)
Person(id:integer, ..., name:text, ...)
Student(id:integer, sid:integer, stype:('local','intl'))
Staff(id:integer, sid:integer, office:integer, ...)
StaffRole(id, descript)
Affiliation(staff, orgunit, role, fraction)
```

Note that there is an example database unsw.dump (3.5MB) that you could load into a newly created database to help with these problems, although you should be able to solve them without reference to a specific database instance. Note that all of the people data in this database is synthetic and the various enrolment tables have been cleared to save space.

The examples below assume that the user is connected to a database called `unsw` containing an instance of the above schema.

14. Write a PLpgSQL function to produce the complete name of an organisational unit (aka OrgUnit), given the OrgUnit's internal id:

```
function unitName(_ouid integer) returns text
```

This will need to make use of the `OrgUnit` and `OrgUnitType` tables. The `OrgUnitType` table contains a list of unit types (e.g. faculty, school, institute) via *(id,name)* tuples. The `OrgUnit` table has a foreign key to the `OrgUnitType` table to indicate what kind of unit it is. The attribute contains the useful name of the unit (the `name` attribute is a very abbreviated version of the unit's name). The

`longname` attribute for faculties already contains the words "Faculty of". For other kinds of `OrgUnit`, you need to prepend the name of its `OrgUnitType`.

The function returns the complete name using the rules:

- the university is denoted by UNSW
- a faculty is denoted using its base name (not all faculty names start with Faculty)
- a school is denoted School of XYZ
- a department is denoted Department of XYZ
- a centre is denoted Centre for XYZ
- an institute is denoted Institute of XYZ
- other kinds of OrgUnits are treated as having no name (i.e. return null)

Some examples of usage (assuming `\a` and `\t`):

```
unsw=> select unitName(0);
UNSW

unsw=> select unitName(2);
Faculty of Arts and Social Sciences

unsw=> select unitName(4);
Faculty of Law

unsw=> select unitName(9);
Faculty of Engineering

unsw=> select unitName(11);
Faculty of Science

unsw=> select unitName(36);
School of Chemistry

unsw=> select unitName(44);
School of Computer Science and Engineering

unsw=> select unitName(75);
Centre for Human Geography

unsw=> select unitName(92);
Department of Korean Studies

unsw=> select unitName(999);
ERROR:  No such unit: 999
```

[show answer]

15. In the previous question, you needed to know the internal ID of an `OrgUnit`. This is unlikely, so write a function that takes part of an `OrgUnit.longname` and returns the ID or `NULL` if there is no such unit. If there is more than one matching unit, return the ID of the first matching unit. Implement this as an SQL function, which allows case-insensitive matching:

```
create or replace function unitID(partName text) returns integer
as $$ ... $$ language sql;
```

Examples of usage:

```
unsw=> select unitName(unitID('law'));
Faculty of Law

unsw=> select unitName(unitID('arts'));
Faculty of Arts and Social Sciences

unsw=> select unitName(unitID('information'));
School of Information Management

unsw=> select unitName(unitID('information sys'));
School of Information Systems

unsw=> select unitName(unitID('chem'));
Department of Biochemistry

unsw=> select unitName(unitID('computer'));
School of Computer Science (ADFA)

unsw=> select unitName(unitID('comp%sci%eng'));
School of Computer Science and Engineering

unsw=> select unitName(unitID('korean'));
Department of Korean Studies
```

We use `unitName()` as a way of checking the result Note that such a simple text-based search can produce unexpected results.

[show answer]

16. Write a PLpgSQL function which takes the numeric identifier of a given OrgUnit and returns the numeric identifier of the parent faculty for the specified OrgUnit:

```
function facultyOf(_ouid integer) returns integer
```

Note that a faculty is treated as its own parent. Note also that some OrgUnits don't belong to any faculty; such OrgUnits should return a null result from the function.

Examples of use:

```
unsw=> select unitName(facultyof(2));
Faculty of Arts and Social Sciences

unsw=> select unitName(facultyof(9));
Faculty of Engineering

unsw=> select unitName(facultyof(36));
Faculty of Science

unsw=> select unitName(facultyof(44));
Faculty of Engineering

unsw=> select unitName(facultyof(75));
Faculty of Science

unsw=> select unitName(facultyof(92));
Faculty of Arts and Social Sciences

unsw=> select unitName(facultyof(999));
ERROR:  No such unit: 999
```

[show answer]