Object-relational databases - exercises

(Lecture 3 - Object-oriented and object-relational databases)

From the course pages (Exercises\ ORDBStruct.sql) download SQL commands to create a relation (and filling relations with appropriate content) required for this exercise.

Connect to the PostgreSQL DBMS. Choose existing or create a new database and execute statements from ORDBStruct.sql.

Excercise 1.

a) Define following two data types:
   - ***GPSPointDeg*** suitable for storing GPS coordinate on the Earth's surface: *latitude* $\in$ [-180, 180], specifies north-south and *longitude* $\in$ [-90, 90], specifies east-west position on the Earth's surface. To store the decimal part of the value of *latitude* and *longitude* use 6 places..
   - ***GPSPointRad*** similar to ***GPSPointDeg*** the difference being that the coordinates are expressed in radians rather than in degrees.

b) Write a function ***DegToRad*** that receives ***GPSPointDeg*** data type as an argument**,** returns a ***GPSPointRad*** data type as a result and converts *latitude* and *longitude* values from degrees to radians. **Help**: see function *radians*.

c) Write a command that will create all the objects necessary to carry out conversion (CAST) from ***GPSPointDeg*** in ***GPSPointRad*** data type. Use the function ***DegToRad***.

d) Write a function ***GPSdistance*** that receives two ***GPSPointDeg*** values and returns a shortest distance between that two points on the Earth's surface in km. To calculate shortest distance use Haversin formula:

$$\mathrm{a} = \sin^2\left(\frac{lat2-lat1}{2}\right) + \cos(lat1) * \cos(lat2) * \sin^2\left(\frac{long2-long1}{2}\right)$$

$$\mathrm{d} = R * 2 * atan2\left(\sqrt{a}, \sqrt{1-a}\right) \qquad R = 6371 \text{ km} \text{ prosječna vrijednost radijusa Zemlje}$$

Help: in case you have to declare variables or use procedural commands in PostgreSQL function you should use `LANGUAGE 'plpgsql'`.

e) Using function ***GPSdistance*** calculate distance between
   Zagreb lat: 45.81497 long: 15.97851
   and
   Split lat: 43.50692 long:16.44245
   Distance is about 259km.

f) The names and arguments of mathematical functions implemented in PostgreSQL you can see here: http://www.postgresql.org/docs/9.4/static/functions-math.html

Structure and content of the relations used in excercises 2., 3. and 4. can be found in ORDBStruct.sql.

Excercise 2.

tramRoute		
tramRouteId	tramRouteAbrev	tramRouteName

tramStation	
tramStationId	tramStationName

tramRouteStation		
tramRouteId	TramStationId	tramStationOrd

tramRouteSchedule	
tramRouteId	departureTime

Using relations ***tramRoute***, ***tramStation***, ***tramRouteStation*** i ***tramRouteSchedule*** write:

a) an SQL statement to create relation ***tramRouteOR*** with the following scheme:

tramRoute				
tramRouteId	tramRouteAbrev	tramRouteName	stationsOnTramRoute	departuresOnTramRoute
1	1	Zapadni kolodvor - Borongaj	{"(1,\"Zapadni kolodvor\",1)", "(2,Talovčeva,2)",...}	{04:16:52, 04:42:00, ...}
...	...	...	...	...

Attribute ***stationsOnTramRoute*** is a collection containing elements composed of the following attributes: tramStationId, tramStationName i tramStationOrd.
Attribute ***departuresOnTramRoute*** is a collection consisting of all departure times for particular tram route.

b) one INSERT statement to fill ***tramRouteOR*** relation with the content in accordance with the content of the ralations ***tramRoute***, ***tramStation***, ***tramRouteStation*** and ***tramRouteSchedule***.

Excercise 3.

Using ***tramRouteOR*** relation only, list the names of the tram stations contained at least two tram routes.

Excercise 4.

Using ***tramRouteOR*** relation only, list the names of tram routes and the number of departures in each hour. Additionaly, list the cumulative number of departures - for example, the cumulative number of departures from 5-6 hours includes departures from 4-5 and so on.
Help: to extract the houre from the ***source*** attribute of type TIME you can use the EXTRACT function EXTRACT (HOUR FROM source)

The query result should look as shown below:

	tramroutename character varying(120)	hourfromto text	noofdepartures bigint	noofdeparturescumul numeric
1	Črnomerec - Savišće	0-1	2	2
2	Črnomerec - Savišće	4-5	5	7
3	Črnomerec - Savišće	5-6	7	14
4	Črnomerec - Savišće	6-7	8	22

Excercise 5.

Using SQL commands below create relations ***person*** and ***student***. Create all objects in the database, which will provide a uniqueness of the attribute value ***person.personId*** regardless of whether the tuple is INSERTED into relation person or student.

```
CREATE TABLE person(
  personId INTEGER CONSTRAINT pkPerson
                   PRIMARY KEY,
  FName    VARCHAR(25),
  Lname    VARCHAR(25)) ;
```

```
CREATE TABLE student (
  JMBAG      CHAR(20),
  enrolDate  VARCHAR(100))
 INHERITS (person);
```

SOLUTIONS:

Excercise 1.

a)

```
CREATE TYPE GPSPointDeg AS (
lat DECIMAl (10,6),
long DECIMAl (10,6))

CREATE TYPE GPSPointRad AS (
lat DECIMAl (10,6),
long DECIMAl (10,6))
```

b)

```
CREATE OR REPLACE FUNCTION DegToRad (GPSPointDeg)
RETURNS GPSPointRad AS $$
    SELECT CAST( ROW (radians($1.lat), radians($1.long)) AS GPSPointRad);
$$ LANGUAGE SQL;
```

c)

```
CREATE CAST (GPSPointDeg AS GPSPointRad)
  WITH FUNCTION DegToRad (GPSPointDeg) ;
```

All trigonometric functions used in the function below, as arguments expect the value in radians. On the other hand, it is common (in this part of Earth) to expressed GPS coordinates in degrees. Because of that, we have to enable easy conversion of GPS coordinates in degrees to GPS coordinates in radians.

d)

```
CREATE OR REPLACE FUNCTION GPSdistance (GPSPointDeg, GPSPointDeg)
RETURNS DECIMAl(10,6) AS $$
   DECLARE
   a DECIMAL(10,6);
   DECLARE
   point1 GPSPointRad;
   DECLARE
   point2 GPSPointRad;

   BEGIN
     point1:= $1::GPSPointRad;
     point2:= $2::GPSPointRad;
     a:= sin((point2.lat-point1.lat)/2)^2 +
cos(point1.lat)*cos(point2.lat)*sin((point2.long-point1.long)/2)^2;

     RETURN 6371 *2*atan2((a ^0.5), ((1-a)^0.5));
   END
 $$ LANGUAGE 'plpgsql';
```

We can not use "ordinary" SQL function in PostgreSQL, because in that case it is not possible to declare variables or use commands to assign values to variables.

e)

```
SELECT GPSDistance ((45.81497, 15.97851), (43.50692, 16.44245))

Result: 259.279238301364
```

Excercise 2.

a)

```
CREATE TYPE stationOnTramRoute AS
  (tramStationId          INT,
   tramStationName        VARCHAR(120),
   tramStationOrd         SMALLINT);

CREATE TABLE tramRouteOR
  (tramRouteId            INTEGER CONSTRAINT pkTramRouteOR PRIMARY KEY,
   tramRouteAbrev         CHAR(3) NOT NULL UNIQUE,
   tramRouteName          VARCHAR(120) NOT NULL UNIQUE,
   stationsOnTramRoute    stationOnTramRoute[],
   departuresOnTramRoute  TIME[]
   );
```

b)

```
INSERT INTO tramRouteOR
 SELECT tramRoute.*
      , (SELECT array_agg(CAST( ROW(tramStation.*, tramRouteStation.tramStationOrd)
AS StationOnTramRoute))
           FROM tramRouteStation, tramStation
          WHERE tramRouteStation.tramStationId = tramStation.tramStationId
            AND tramRouteStation.tramRouteId = tramRoute.tramRouteId)
      , (SELECT array_agg(departureTime)
           FROM tramRouteSchedule
          WHERE tramRouteSchedule.tramRouteId = tramRoute.tramRouteId)
FROM tramRoute
```

Consider the expression:

```
array_agg(CAST( ROW(tramStation.*, tramRouteStation.tramStationOrd)  AS
StationOnTramRoute))
```

1. Do we need `CAST (…) AS StationOnTramRoute`?

   Attempt to INSERT without CAST:

   ```
   INSERT INTO tramRouteOR
    SELECT tramRoute.*
         , (SELECT array_agg(ROW(tramStation.*, tramRouteStation.tramStationOrd))
              FROM tramRouteStation, tramStation
                ...
   ```

   ends up with an error:

   ```
   ERROR:  column "stationsOnTramRoute" is of type stationOnTramRoute[] but expression
   is of type record[]
   LINE 4:       , (SELECT array_agg(ROW(tramStation.*, tramRouteStation.r...
                    ^
   HINT:  You will need to rewrite or cast the expression.
   ```

`array_agg(ROW(tramStation.*, tramRouteStation.tramStationOrd))` is a collection whose elements are of "unknown" type ie. the result of this expression is of the type **record []**. TramRouteOR.stationsOnTramRoute attribute is the collection whose elements are of type ***stationOnTramRoute***. We need to perform explicit CAST - otherwise, due to incompatible data types, INSERT command will end with an error.

2. Do we need `ROW(tramStation.*, tramRouteStation.tramStationOrd)`? Is it enough to write just this:
   `CAST( (tramStation.*, tramRouteStation.tramStationOrd)  AS StationOnTramRoute)`

We don’t need ROW – sintax is correct without ROW.

Excercise 3.

To be able to group tuples by tram station name contained in the collection ***tramRouteOr.stationsOnTramRoute*** we have to unnest collection. The result of the following query

```
SELECT tramRouteOR.tramRouteId
     , UNNEST (tramRouteOR.stationsOnTramRoute) AS stationsOnTramRoute
  FROM tramRouteOR
```

is:

	tramrouteid integer	stationsontramroute stationontramroute
1	1	(1,"Zapadni kolodvor",1)
2	1	(2,Talovčeva,2)
3	1	(3,Reljkovićeva,3)
4	1	(4,"Trg dr. Franje Tuđmana",4)
5	1	(5,"Britanski trg",5)
6	1	(6,Frankopanska,6)

Attribute ***stationsOnTramRoute*** is of complex type (`TYPE stationOnTramRoute`). To access attributes of a complex type we can use attribute names. However, care should be taken when approaching the complex type attributes.
Eg. the following query ends up in error:

```
SELECT stationsOnTramRoute.tramStationId,
       stationsOnTramRoute.tramStationName,
       stationsOnTramRoute.tramStationOrd
  FROM (
        SELECT tramRouteOR.tramRouteId
             , UNNEST (tramRouteOR.stationsOnTramRoute) AS stationsOnTramRoute
         FROM tramRouteOR)    AS stationsOnAllTramRoutes
ERROR:  missing FROM-clause entry for table "stationsontramroute"
LINE 1: SELECT stationsOnTramRoute.tramStationId,
               ^
```

According to the SQL syntax ***stationsOnTramRoute*** represents the name of the table (while that's just a complex attribute in the "Table" stationsOnAllTramRoutes). Properly written SELECT list of the previous query is:

```
SELECT (stationsOnTramRoute).tramStationId,
       (stationsOnTramRoute).tramStationName,
       (stationsOnTramRoute).tramStationOrd
```

or if we want to include the name opf the table:

```
SELECT (stationsOnAllTramRoutes.stationsOnTramRoute).tramStationId,
       (stationsOnAllTramRoutes.stationsOnTramRoute).tramStationName,
       (stationsOnAllTramRoutes.stationsOnTramRoute).tramStationOrd
```

Object in parentheses is interpreted as a reference to the object ***stationsOnTramRoute***, and then it’s attributes can be accessed.

Complete solution:

```
SELECT (stationsOnAllTramRoutes).stationsOnTramRoute.tramStationName,
       COUNT((stationsOnAllTramRoutes).tramRouteId)
  FROM (
        SELECT tramRouteOR.tramRouteId
             , UNNEST (tramRouteOR.stationsOnTramRoute) AS stationsOnTramRoute
         FROM tramRouteOR)    AS stationsOnAllTramRoutes
 GROUP BY (stationsOnAllTramRoutes).stationsOnTramRoute.tramStationName
 HAVING COUNT(*) >= 2
```

Excercise 4.

To be able to group tuples by hour of departure, which is contained in the collection ***tramRouteOR.departuresOnTramRoute*** we hawe to unnest that collection. The result of the query

```
SELECT tramRouteOR.tramRouteId, tramRouteOR.tramRouteName
            , UNNEST (tramRouteOR.departuresOnTramRoute) AS departuresOnTramRoute
         FROM tramRouteOR
```

is of the form:

	tramrouteid integer	tramroutename character varying(120)	departuresontramroute time without time zone
1	1	Zapadni kolodvor - Borongaj	04:16:52
2	1	Zapadni kolodvor - Borongaj	04:42:00
3	1	Zapadni kolodvor - Borongaj	05:00:22

To calculate cumulative number of departures per hour we should:

1. create a partition for each tram route (PARTITION BY tramRouteName; or PARTITION BY tramRouteId or PARTITION BY tramRouteName, tramRouteId with adjusted GROUP BY part)
2. sort rows in partition properly - `ORDER BY (EXTRACT(HOUR FROM departuresOnTramRoute))`. Since the frame of current row should contain all rows in it's partition preceding current row and current row, without correct ORDER BY part, cumulative sums will not be correct.

```
SELECT tramRouteName,
        EXTRACT(HOUR FROM departuresOnTramRoute) || '-' ||
        EXTRACT(HOUR FROM departuresOnTramRoute)+1 hourFromTo,
        COUNT(*) noOfdepartures,
        SUM(count(*)) OVER (PARTITION BY tramRouteName
                            ORDER BY (EXTRACT(HOUR FROM departuresOnTramRoute)))
                    AS noOfdeparturesCumul
  FROM (
        SELECT tramRouteOR.tramRouteId, tramRouteOR.tramRouteName
             , UNNEST (tramRouteOR.departuresOnTramRoute) AS departuresOnTramRoute
         FROM tramRouteOR)    AS stationsOnAllTramRoutes
 GROUP BY tramRouteName,
          EXTRACT(HOUR FROM departuresOnTramRoute) || '-' ||
          EXTRACT(HOUR FROM departuresOnTramRoute)+1,
          EXTRACT(HOUR FROM departuresOnTramRoute)
 ORDER BY tramRouteName, EXTRACT(HOUR FROM departuresOnTramRoute)
```

In this exercise it is important to write correct GROUP BY part. It is obivous that GROUP BY should contain

```
tramRouteName
and
EXTRACT(HOUR FROM departuresOnTramRoute) || '-' || EXTRACT(HOUR FROM
departuresOnTramRoute)+1
```

since each of them appears in the SELECT list besides `COUNT(*)`.

The less obvious is that the GROUP BY must contain the following expression
`EXTRACT (HOUR FROM departuresOnTramRoute).`
The reason for that? It is used while calculating the SUM (COUNT(*)) for partition. Since the expressions calculated over rows contained in the window (partition or frame) are evaluated after GROUP BY (and eventual HAVING, not present in this query), if we omit `EXTRACT (HOUR FROM departuresOnTramRoute)` we will not be able to sort rows according to this value.

Excercise 5.

Performing following SQL statements we can see that primary key constraint is not preserved.

```
set DateStyle ='German, DMY';
INSERT INTO person     VALUES (1, 'Ana'   , 'Ban');
INSERT INTO person     VALUES (1, 'Tena'  , 'Pale');
ERROR:  duplicate key value violates unique constraint "pkperson"
DETAIL:  Key (personId)=(1) already exists.

INSERT INTO student   VALUES (1, 'Mia'  , 'Nel', '0036000001', '01.07.2013');
Query returned successfully: one row affected, 12 ms execution time.
```

```
SELECT *
FROM person;
```

	personid integer	fname character varying(25)	lname character varying(25)
1	1	Ana	Ban
2	1	Mia	Nel

In ***person*** the key constraint is no longer preserved. Let us try to preserve the key constraint with the following statement:

```
ALTER TABLE student ADD CONSTRAINT studentPk PRIMARY KEY (personId);
```

The fact that the above SQL command completed without error tells us that we havent protected the integrity of the key in ***person***.

Also, the following sequence of instructions shows that the uniqueness of the attribute personId is preserved only for tuples inserted in the ***student*** table, but not at the level of union of tuples in ***person*** and ***student***.

```
INSERT INTO person    VALUES (2, 'Tena'  , 'Pale');
INSERT INTO student   VALUES (2, 'Ivo'  , 'Puž',  '0036000002', '12.07.2014');
Query returned successfully: one row affected, 12 ms execution time.

INSERT INTO student   VALUES (2, 'Ivo'  , 'Puž',  '0036000002', '12.07.2014');
ERROR:  duplicate key value violates unique constraint "studentpk"
DETAIL:  Key (personId)=(2) already exists.
```

The integrity of key at the level of union of tuples from ***person*** and ***student*** can be achieved by using the procedure that will report error while trying to enter the tuples with the same key in relation ***person*** and triggers that will activate whenever INSERT in ***person*** and ***student*** occurs.

```
CREATE FUNCTION
  personPkChk() RETURNS TRIGGER AS $$
  BEGIN
     IF (EXISTS (SELECT * FROM person
                  WHERE person.personId = NEW.personId)) THEN
        RAISE EXCEPTION 'Already exists record with id %.', NEW.personId;
     END IF;
     RETURN NEW;
   END
 $$ LANGUAGE plpgsql;

CREATE TRIGGER studentPkChk
BEFORE INSERT ON student
  FOR EACH ROW
      EXECUTE PROCEDURE personPkChk();

CREATE TRIGGER personPkChk
BEFORE INSERT ON person
  FOR EACH ROW
```

```
EXECUTE PROCEDURE personPkChk();
```

With following SQL statements we can test solution:

```
INSERT INTO person     VALUES (1, 'Ana'   , 'Ban');                              --ok
INSERT INTO student   VALUES (1, 'Mia'  , 'Nel', '0036000001', '01.07.2013');
Already exists record with id 1.

INSERT INTO student   VALUES (2, 'Mia'  , 'Nel', '0036000001', '01.07.2013');  --ok
INSERT INTO person    VALUES (2, 'Tena'  , 'Pale');
Already exists record with id 2.
```