Dr Joseph Vella
Department of Computer Information Systems, FICT

Distributed Databases Assignment (2019-2020)
Asynchronous query and stored procedure execution

Submit a written and PDF versions of this assignment with scripts, indicative output (e.g. of scripts) and explanation in a pretty formatted report. You are allowed to work in pairs. Keep your workings concise and examples as simple as possible. You must write a query or function or trigger (and response if appropriate) when an '*' appears next the problem number. You are allowed to use any schema for any instance (if you are using the scott database for every instance it is best to change some data e.g. department location and employees number in one database to make your data distinct per instance).

Caution: Please note scripts and techniques are for teaching purposes and specifically for starting, executing and managing synchronous and asynchronous queries. Many features and necessary code have been avoided for simplicity of illustration. That is, these are not production quality code.

Use case: Our application needs to schedule meetings; this action requires sending request for quotes to various suppliers (e.g. restaurants, transport, etc.) and then co-ordinate selection. We need to know, for our interested date interval and cost we are willing to pay, if these suppliers can provide the service. We expect them to respond with a set of dates that are possible for them. Once we have a response from all of our suppliers we can then choose a day (i.e. that must be one that is commonly available from all of our solicited suppliers).

For illustrative purposes, my examples use company name '**We**' for requester and company name '**Grecco**' for supplier.

Revise synchronous and asynchronous communications between PostgreSQL databases.

Create two databases; one each for requester and supplier. Also create a schema in each database where tables, views and stored procedures for this exercise are stored. See code snippets w1 and g1 for relative companies.
Create an SQL shell for each database (e.g. one for We and one for Grecco). Henceforth execute scripts/snippets at the specified shell. E.g. execute w2 in We's shell.
In the We database, create a table in which we can store responses from our suppliers, e.g. Grecco. See for example code snippet w2.
In the Grecco database, create a table and populate it with test data. Revise script g2 as a method to generate reasonable data. In script g3 find details of a table created and how generated data is inserted into the table.
'We' is insisting that their requests are to be answered by three fields for each possibility: Available date (date datatype), cost (double precision data type), and name of supplier / sender of data (character varying data type). This structure is easy to create – see g4 (and applicable to 'we' too).
Create a function/view in Grecco that can retrieve availability rows that match a date interval and a cost. See g5 for an example.
In We database enable connectivity, connect to Grecco database, send a synchronous request for availability (see code snippet w3), and then send the same request asynchronously (see code snippet w4). Convert the asynchronous wait and get into a procedure (see code snippet w5).

* Introduce another supplier (e.g. Bertu) and add database, schema, table, test data, retrieval stored procedure.
* Create on We's database a stored procedure (as shown in snippet w4) to now combine requests sent to both Grecco and Bertu asynchronousely.

WE Code Snippet **w1**: Database and schema creation

```
CREATE DATABASE we
WITH ENCODING = 'UTF8'
OWNER = postgres
TEMPLATE = template1
CONNECTION LIMIT = - 1;

CREATE SCHEMA requests
AUTHORIZATION postgres;
```

GRECCO Code Snippet **g1**: Database and schema creation

```
CREATE DATABASE Greco
WITH ENCODING = 'UTF8'
TEMPLATE = template1
CONNECTION LIMIT = - 1;

CREATE SCHEMA bookings
AUTHORIZATION postgres;
```

WE Code Snippet **w2**: Create request table (his table holds responses from suppliers).

```
CREATE TABLE requests.got_back (
    i smallserial,
    supplier character varying(99),
    cost double precision,
    available date,
    CONSTRAINT got_back_pk
    PRIMARY KEY (i)
) WITH ( OIDS = FALSE );

ALTER TABLE requests.got_back
OWNER TO postgres;
```

GRECCO Code Snippet **g2**: Try the following queries and understand their meaning

```
-- today's date is
SELECT to_char(now(), 'YYYYMMDD Dy');

-- date sequence/range generation
SELECT to_char(now()::date + i.i, 'YYYYMMDD Day D')
  FROM generate_series(0, 200) AS i (i);

-- set a mask for working days
-- if today is a Sunday then say we have
-- 0110111
-- which is on (i.e. 1) on 2,3,5,6 & 7

-- i.e. working on M,T,T,F and S
-- date sequence generation - omitting off days

SELECT now()::date + i.i, (random() * 1000)::integer
    -- to_char(now()::date+i.i,'YYYYMMDD Day D')
  FROM generate_series(0, 200) AS i (i)
 WHERE to_char(now()::date + i.i, 'D')
       IN ('2', '3', '5', '6', '7');
```

GRECCO Code Snippet **g3**: Create table and generate test data (based on g2 example)

```
CREATE TABLE CREATE TABLE bookings.available (
    sn bigserial,
    day_avail date,
    day_cost double precision,
    CONSTRAINT avail_pk
    PRIMARY KEY (sn)
) WITH ( OIDS = FALSE );

ALTER TABLE bookings.available
OWNER TO postgres;


-- Insert some rows (set the number of generated integers to 400
--  for example)
-- And check them out (note each run will generate a different cost
--  per day)

INSERT INTO bookings.available (day_avail, day_cost)
     SELECT now()::date + i.i, (random() * 1000)::integer
       FROM generate_series(0, 400) AS i (i)
      WHERE to_char(now()::date + i.i, 'D')
         IN ('2', '3', '5', '6', '7');

-- check the data generated
-- everyone's data should be different because of the random() function

SELECT *
  FROM bookings.available;
```

GRECCO Code Snippet **g4**: Create a data type that has the structure of supplier response required by requester

```
-- use this in case you need to delete the type
-- or to reformat its structure
DROP TYPE broker;

CREATE TYPE broker AS (
    adate date,
    cost  double precision,
    weare character varying
);
```

GRECCO Code Snippet **g5**: Find available slots by date interval and a cost factor

```
-- Create function that returns dates within an interval of dates
-- and costs are less than indicated.

CREATE OR REPLACE FUNCTION
    bookings.availability (fd date, td date, dg numeric)
    RETURNS SETOF broker  AS $$

    SELECT pg_sleep(random() * 10);  -- this is a delay

    SELECT day_avail, day_cost, 'Greco'::character varying
      FROM bookings.available
     WHERE day_avail BETWEEN fd AND td
       AND day_cost < dg;
$$
LANGUAGE SQL;


-- test it
SELECT *
  FROM bookings.availability (
         to_date('20191201', 'YYYYMMDD'),
         to_date('20191231', 'YYYYMMDD'),
         500::numeric
        );
```

WE Code snippet **w3**: Create connection and send synch query

```
-- enable connection to supplier
-- we agree on structure of data to query & result type
-- (adate date, cost double precision, optionat character varying)

CREATE EXTENSION dblink;

-- create named connection
SELECT dblink_connect(
         'con2greco',
         'port=5432 dbname=greco user=postgres password=whatever');

-- try synch query mode from we to greco and back
-- note stored procedure has a programmed delay

SELECT *
  FROM dblink(
        'con2greco',
        'select * from bookings.availability (to_date(''20191201'',
''YYYYMMDD''), to_date(''20191231'', ''YYYYMMDD''), 500::numeric);',
'true') AS t (d date, c double precision, who character varying);
```

WE Code snippet **w4**: Asynchronous query example and connection management

```
-- try asynch query mode from we to greco
SELECT dblink_send_query('con2greco', 'select * from
bookings.availability (to_date(''20191201'', ''YYYYMMDD''),
to_date(''20191231'', ''YYYYMMDD''), 500::numeric);');

-- check if query is done at greco for us
SELECT dblink_is_busy('con2greco');

-- get query result from greco
SELECT *
  FROM dblink_get_result('con2greco') AS t (d date, c double precision,
who character varying);

-- disconnect from greco
SELECT dblink_disconnect('con2greco');
```

WE Code snippet **w5**: Asynchronous query example and connection management

```
-- check if greco connection is open, if not open a connection
SELECT TRUE
  FROM dblink_get_connections() AS c (t)
 WHERE 'con2greco' = ANY (c.t);


-- DROP TYPE we_broker;
CREATE TYPE we_broker AS (
 adate date, cost double precision, optionat character varying);

-- DROP FUNCTION requests.pooling_bookings();
CREATE OR REPLACE FUNCTION requests.pooling_bookings ()
    RETURNS TABLE (
        adate date,
        tcost double precision,
        optionat character varying
    ) AS $$
DECLARE
    grecoflag INTEGER;
    connections TEXT[];
    result we_broker;
BEGIN
    -- check connections are open
    SELECT * INTO connections
      FROM dblink_get_connections() AS c (t)
     WHERE 'con2greco' = ANY (c.t);
    IF NOT found THEN
        RAISE EXCEPTION 'Nonexistent dblink connection --> %',
                        'con2greco';
    END IF;
    -- check if asynch channel are busy
    grecoflag := 1;
    WHILE grecoflag = 1 LOOP
        PERFORM   SELECT pg_sleep(2);
                  -- sleep for 2 seconds (and ignore output)
        SELECT dblink_is_busy('con2greco') INTO grecoflag;
    END LOOP;
    -- get data now
    RETURN QUERY
    SELECT *
      FROM dblink_get_result('con2greco')
        AS t (adate date,
              tcost double precision,
              optionat character varying);
    RETURN;
END
$$ LANGUAGE 'plpgsql';

-- program / interactive mode
-- send asych query

SELECT dblink_send_query('con2greco', 'select * from
bookings.availability (to_date(''20191201'', ''YYYYMMDD''),
to_date(''20191231'', ''YYYYMMDD''), 500::numeric);');

-- pool for readiness and then retrieve
SELECT *
  FROM requests.pooling_bookings ();
```