Uso de bases de datos

Práctica 2: El lenguaje SQL II

En la primera parte de la práctica hemos realizado consultas y modificaciones sobre los datos utilizando SQL. En esta segunda parte, sobre la misma base de datos, añadiremos lógica dentro de la base de datos utilizando procedimientos almacenados y disparadores.

Para la correcta ejecución de la segunda parte de la práctica, **es necesario volver a crear la base de datos de nuevo e insertar otra vez los datos iniciales utilizando los *scripts* que se proporcionan** junto a este enunciado (create_db.sql e inserts_db.sql, respectivamente). Este paso es necesario debido a la introducción de los siguientes cambios en el esquema de la BD respecto a la primera parte de la práctica

- Creación de una nueva tabla llamada *REPORT_DOG*, con las columnas *id_dog*, *name_dog* y *birth*, que son el identificador, el nombre y la fecha de nacimiento del perro, *name_owner* y *phone*, que son el nombre y el teléfono del propietario, *num_visits* el número de visitas que ha tenido el perro, *num_dif_vaccines* y *date_last_vaccine* que son el número de vacunas diferentes que se le han administrado al perro, y la fecha de la última vacuna, *num_drugs*, el número total de medicamentos que se le han prescrito al perro, y finalmente *num_tests*, el número total de pruebas que se le han hecho al perro. Las columnas se calculan en relación con el perro.
- En la tabla *DOG* se ha añadido la columna *num_vaccines,* que representa **el número total de vacunas que se le han administrado al perro en cuestión.**

Nota importante: El SQL implementado en PostgreSQL puede aceptar diferentes variantes de sintaxis, que además pueden diferir según la versión que instaléis, y que pueden ser o no SQL estándar. Evitad (excepto que se indique lo contrario) utilizar sentencias de este tipo, y concentraros en las que se explican en los módulos didácticos. Si usáis sentencias SQL estándar vuestro código funcionará en cualquier SGBD.

Pregunta 1 (50 % puntuación)

Enunciado

Se pide crear un procedimiento almacenado que dado el nombre de un perro, nos dé, si está vivo, los datos que hacen referencia al mismo (o mismos) indicando su identificador (*id_dog*), el nombre, la fecha de nacimiento, el nombre y el teléfono del propietario, el número de visitas, el número de vacunas diferentes que se le han administrado, la fecha de la última vacunación, el número de medicamentos que se le han prescrito, y el número de tests que se le han hecho, y los almacene en la tabla *REPORT_DOG*. La tabla en cuestión se creará con la ejecución del archivo *create_db.sql*, **que se deberá ejecutar en primer término**. Si ya existen filas en la tabla *REPORT_DOG* para el perro (o perros), ésta deberá modificarse con los nuevos valores. Además de guardarlo en la tabla, el procedimiento devolverá el resultado del *report*.

Habrá que informar al usuario con un mensaje específico cuando no exista ningún perro con el nombre que se ha indicado.

Si el perro está muerto, se deberá dar un mensaje diciendo que está muerto y si se encontrara en la tabla *REPORT_DOG*, deberá suprimirse.

Si el perro ya existe en la tabla (y no está muerto), se actualizarán los datos. Si no está en la tabla, se insertará.

La signatura del procedimiento pedido y el tipo que devolverá son los siguientes:

```
CREATE OR REPLACE FUNCTION update_report_dog(p_name_dog VARCHAR(255))
RETURNS SETOF REPORT_DOG_TYPE
```

donde *REPORT_DOG_TYPE* es de tipo:

```
CREATE TYPE REPORT_DOG_TYPE AS (
      t_id_dog SMALLINT,
      t_name_dog VARCHAR(255),
      t_birth DATE,
      t_name_owner VARCHAR(255),
      t_phone INTEGER,
      t_num_visits SMALLINT,
      t_num_dif_vaccines SMALLINT,
      t_last_vaccine date,
      t_num_drugs SMALLINT,
      t_num_tests SMALLINT
);
```

Nota: Adicionalmente, en el siguiente enlace, encontraréis información sobre errores y mensajes en PL/PostgreSQL: https://www.postgresql.org/docs/current/plpgsql-errors-and-messages.html.

Criterios de evaluación

- Las propuestas de solución que no se puedan ejecutar, es decir, las que den error de sintaxis, no serán evaluadas.
- Se valorará positivamente el uso de sentencias SQL estándar, al margen de otros elementos que se puedan indicar en el enunciado.
- Para obtener la máxima nota, el código SQL de vuestra solución tiene que ser eficiente. Por ejemplo, se valorará negativamente realizar más *joins* de los necesarios.
- Para obtener la máxima nota, la propuesta de solución tiene que incluir pruebas que cubran todas las posibles situaciones descritas en el enunciado. Por ejemplo, se deberían cubrir todas las posibles situaciones de error.
- Para obtener la máxima nota, la propuesta de solución tiene que incluir los resultados, mediante el uso de capturas de pantalla o cualquier otro mecanismo similar.

Solución

```
SET search_path TO ubd_20222;

CREATE OR REPLACE FUNCTION update_report_dog(p_name_dog VARCHAR(255))
RETURNS SETOF REPORT_DOG_TYPE AS $$

DECLARE
      var_return_data REPORT_DOG_TYPE;
BEGIN

IF ((SELECT COUNT(*)
      FROM DOG d
      WHERE d.name_dog LIKE p_name_dog) = 0)
THEN

      RAISE EXCEPTION 'ERROR: There is no dog called %', p_name_dog;

END IF;

FOR var_return_data in
      SELECT d.id_dog, d.name_dog, d.birth
      FROM DOG d
      WHERE name_dog LIKE p_name_dog

LOOP
      IF EXISTS (SELECT d.death
            FROM  DOG d
            WHERE  d.id_dog = var_return_data.t_id_dog AND d.death IS NOT NULL)
      THEN
         RAISE NOTICE ' % with id % is dead', p_name_dog,var_return_data.t_id_dog;
         IF EXISTS (     SELECT rd.id_dog
                         FROM  REPORT_DOG rd
                         WHERE  rd.id_dog = var_return_data.t_id_dog)
```

```
    THEN
            DELETE FROM REPORT_DOG rd
            WHERE rd.id_dog = var_return_data.t_id_dog;
    END IF;

ELSE
      SELECT o.name_owner, o.phone
      INTO var_return_data.t_name_owner, var_return_data.t_phone
      FROM OWNER o NATURAL JOIN DOG d
      WHERE d.id_dog = var_return_data.t_id_dog;

      SELECT COUNT(*)
      INTO var_return_data.t_num_visits
      FROM VISIT vi
      WHERE vi.id_dog = var_return_data.t_id_dog;

      SELECT COUNT(DISTINCT(va.id_vaccine)), MAX(vi.date)
      INTO var_return_data.t_num_dif_vaccines, var_return_data.t_last_vaccine
      FROM DOG d NATURAL LEFT JOIN VISIT vi  NATURAL JOIN VACCINATION va
      WHERE d.id_dog = var_return_data.t_id_dog AND vi.reason ='vaccination';

      SELECT COUNT(pr.id_drug)
      INTO var_return_data.t_num_drugs
      FROM DOG d NATURAL LEFT JOIN VISIT vi NATURAL LEFT JOIN PRESCRIPTION pr
      WHERE d.id_dog = var_return_data.t_id_dog;

      SELECT COUNT(dt.id_test)
      INTO var_return_data.t_num_tests
      FROM DOG d NATURAL LEFT JOIN VISIT vi  NATURAL LEFT JOIN DOG_TEST dt
      WHERE d.id_dog = var_return_data.t_id_dog;


      IF NOT EXISTS(
                  SELECT rd.id_dog
                  FROM REPORT_DOG rd
                  WHERE rd.id_dog = var_return_data.t_id_dog)

      THEN

            INSERT INTO REPORT_DOG
            VALUES (
                  var_return_data.t_id_dog,
                  var_return_data.t_name_dog,
                  var_return_data.t_birth,
                  var_return_data.t_name_owner,
                  var_return_data.t_phone,
                  var_return_data.t_num_visits,
                  var_return_data.t_num_dif_vaccines,
                  var_return_data.t_last_vaccine,
                  var_return_data.t_num_drugs,
                  var_return_data.t_num_tests
            );

      ELSE
            UPDATE REPORT_DOG
            SET   name_owner = var_return_data.t_name_owner,
                  phone = var_return_data.t_phone,
                  num_visits = var_return_data.t_num_visits,
```

```
                        num_dif_vaccines = var_return_data.t_num_dif_vaccines,
                        date_last_vaccine = var_return_data.t_last_vaccine,
                        num_drugs = var_return_data.t_num_drugs,
                        num_tests = var_return_data.t_num_tests
                    WHERE id_dog = var_return_data.t_id_dog;
            END IF;

      END IF;
      RETURN NEXT var_return_data;
END LOOP;
END;

$$LANGUAGE plpgsql;
```

Con las siguientes pruebas validamos el comportamiento en las diferentes casuísticas, partiendo del fichero *inserts_db.sql* que os damos en el enunciado, el cual **deberéis ejecutar antes de realizar cualquier prueba.**

Primero comprobaremos el control del error. En este caso introduciremos un nombre de perro que no exista en la base de datos.

```
SELECT * FROM update_report_dog('Laoka');
```

Y el resultado debe ser:

ERROR: ERROR: There is no dog called Laoka
CONTEXT: PL/pgSQL function update_report_dog(character varying) line 12 at RAISE

A continuación probamos con un nombre que solo tenga un perro:

```
SELECT * FROM update_report_dog('Tom');
```

En la tabla *REPORT_DOG* encontramos

```
idd  | name_dog |   birth    |    owner       |   phone   | nvi | ndv | date_vacc  | ndr | nte
-----+----------+------------+----------------+-----------+-----+-----+------------+-----+-----
   1 | Tom      | 2019-03-21 | Laura Gonzalez | 656734829 |   2 |   1 | 2022-02-02 |   0 |   1
(1 row)
```

Si volvemos a ejecutar la sentencia vemos que no da ningún error:

```
SELECT * FROM update_report_dog('Tom');
```

En la tabla *REPORT_DOG* encontramos

```
idd  | name_dog |   birth    |    owner       |   phone   | nvi | ndv | date_vacc  | ndr | nte
-----+----------+------------+----------------+-----------+-----+-----+------------+-----+-----
   1 | Tom      | 2019-03-21 | Laura Gonzalez | 656734829 |   2 |   1 | 2022-02-02 |   0 |   1
(1 row)
```

Ahora lo intentaremos con un nombre que tenga más de un perro:

```
SELECT * FROM update_report_dog('Max');
```

En la tabla *REPORT_DOG* encontramos

```
idd  | name_dog |   birth    |    owner       |   phone   | nvi | ndv | date_vacc  | ndr | nte
-----+----------+------------+----------------+-----------+-----+-----+------------+-----+-----
   1 | Tom      | 2019-03-21 | Laura Gonzalez | 656734829 |   2 |   1 | 2022-02-02 |   0 |   1
   6 | Max      | 2015-03-12 | Sergio Ortiz   | 655729481 |   6 |   1 | 2021-01-23 |   2 |   7
  36 | Max      | 2018-02-28 | Marta Torres   | 692481937 |   2 |   0 |            |   0 |   1
(3 rows)
```

Ahora haremos modificaciones que afecten a las diferentes columnas de la tabla *REPORT_DOG*.

Lo haremos para el perro con `id_dog = 6` añadiendo la visita `id_visit= 69`

```
INSERT INTO VISIT VALUES (69,6,'06-03-2023', 'vaccination','3476','Doing
some tests.');
```

Modificando el teléfono del propietario

```
UPDATE OWNER SET phone = 666666666 WHERE id_owner = 12;
```

Añadiendo Vacuna

```
INSERT INTO VACCINATION VALUES (69, 8);
```

Añadiendo un nuevo medicamento prescrito

```
INSERT INTO PRESCRIPTION VALUES (69,  3, 2, 7);
```

Añadiendo nuevo test

```
INSERT INTO DOG_TEST VALUES ( 2,  69);
```

Volvemos a ejecutar la función:

```
SELECT * FROM update_report_dog('Max');
```

En la tabla REPORT_DOG encontramos

```
idd  | name_dog |   birth    |    owner       |   phone   | nvi | ndv | date_vacc  | ndr | nte
-----+----------+------------+----------------+-----------+-----+-----+------------+-----+-----
   1 | Tom      | 2019-03-21 | Laura Gonzalez | 656734829 |   2 |   1 | 2022-02-02 |   0 |   1
   6 | Max      | 2015-03-12 | Sergio Ortiz   | 666666666 |   7 |   2 | 2023-03-06 |   3 |   8
  36 | Max      | 2018-02-28 | Marta Torres   | 692481937 |   2 |   0 |            |   0 |   1
(3 rows)
```

A continuación pondremos fecha de defunción a un perro que tengamos en la tabla *REPORT_DOG*:

Añadiendo fecha de defunción al perro con `id_dog = 36`

```
UPDATE DOG SET death ='07-02-2023' WHERE id_dog = 36;
```

Volvemos a ejecutar la función:

```
SELECT * FROM update_report_dog('Max');

NOTICE:   Max with id 36 is dead
```

En la tabla *REPORT_DOG* ya no se encuentra el perro que tiene fecha de defunción:

```
idd  | name_dog |   birth    |    owner       |   phone   | nvi | ndv | date_vacc  | ndr | nte
-----+----------+------------+----------------+-----------+-----+-----+------------+-----+-----
   1 | Tom      | 2019-03-21 | Laura Gonzalez | 656734829 |   2 |   1 | 2022-02-02 |   0 |   1
   6 | Max      | 2015-03-12 | Sergio Ortiz   | 666666666 |   7 |   2 | 2023-03-06 |   3 |   8
```

```
(2 rows)
```

Pregunta 2 (50 % puntuación)

Enunciado

En la tabla *DOG* tenemos la columna *num_vaccines* con el objetivo de almacenar el número de dosis de vacuna que ha recibido el perro (se quieren todas, aunque sean repetidas).

Cread un disparador o disparadores, sobre la tabla o tablas que sean necesarias, de forma que se mantenga correctamente actualizada la columna *num_vaccines* de la tabla *DOG*.

En concreto, se quiere que esta columna siempre refleje los valores solicitados y estos valores siempre deben mantenerse actualizados en función de los cambios.

Podemos suponer que los usuarios o programas nunca actualizarán directamente la columna *num_vaccines* de la tabla *DOG*, y que, en el momento de insertar un nuevo perro, el valor de la columna *num_vaccines* será cero.

Criterios de evaluación

- Las propuestas de solución que no se puedan ejecutar, es decir, las que den error de sintaxis, no serán evaluadas.
- Se valorará positivamente el uso de sentencias SQL estándar, al margen de otros elementos que se puedan indicar en el enunciado.
- Para obtener la máxima nota, el código SQL de vuestra solución tiene que ser eficiente. Por ejemplo, se valorará negativamente realizar más *joins* de los necesarios.
- Para obtener la máxima nota, la propuesta de solución tiene que incluir pruebas que cubran todas las posibles situaciones descritas en el enunciado.
- Para obtener la máxima nota, la propuesta de solución tiene que incluir los resultados, mediante capturas de pantalla o de alguna forma similar.
- Para obtener la máxima puntuación, el código que actualice *num_vaccines* tiene que pertenecer a una única función.

Solución

La solución se divide en dos partes:

1. **Creación de la función del tratamiento de actualización**: implementación y creación de la función *update_dog_vaccines* que trata las casuísticas siguientes:
    - Inserción de una nueva fila (*VACCINATION*): aumentando en una unidad *num_vaccines* del perro a que corresponda la visita referenciada por *NEW.id_visit*.
    - Modificación en la tabla *VACCINATION* del identificador de la visita de un perro (*id_visit*) que identifica un perro (*id_dog*) de *VISIT*:
        - disminuiremos en una unidad el número de vacunas del perro referenciado por *OLD.id_visit*

        - aumentaremos en una unidad el número de vacunas del perro referenciado por *NEW.id_visit.*
    - Borrado de una fila (*VACCINATION*): disminuimos en una unidad el valor de la columna *num_vaccines* del perro referenciado por *OLD.id_visit*.
2. **Creación de los disparadores**: en segundo lugar, hay que crear el disparador que cubra las casuísticas pedidas en el enunciado. Nos hemos de preocupar de cubrir las inserciones, los borrados y modificaciones de la tabla *VACCINATION*, que es la que provoca la modificación del valor de *num_vaccines* de la tabla *DOG*. El disparador será *AFTER* dado que no queremos que actualice la tabla hasta realizar todas las comprobaciones.

Código SQL:

```
SET search_path TO ubd_20222;

CREATE OR REPLACE FUNCTION update_dog_vaccines()
RETURNS trigger AS $$
DECLARE
    dog_identifier DOG.id_dog%TYPE;
BEGIN
     IF (TG_OP = 'INSERT') THEN

          SELECT id_dog INTO dog_identifier
          FROM DOG NATURAL JOIN VISIT
          WHERE id_visit = NEW.id_visit;

          UPDATE DOG SET num_vaccines = num_vaccines + 1
          WHERE id_dog = dog_identifier;

     ELSIF (TG_OP = 'UPDATE') THEN

          SELECT id_dog INTO dog_identifier
          FROM DOG NATURAL JOIN VISIT
          WHERE id_visit = OLD.id_visit;

          UPDATE DOG SET num_vaccines = num_vaccines - 1
          WHERE id_dog = dog_identifier;

          SELECT id_dog INTO dog_identifier
          FROM DOG NATURAL JOIN VISIT
          WHERE id_visit = NEW.id_visit;

          UPDATE DOG SET num_vaccines = num_vaccines + 1
          WHERE id_dog = dog_identifier;

     ELSIF (TG_OP = 'DELETE') THEN

          SELECT id_dog INTO dog_identifier
          FROM DOG NATURAL JOIN VISIT
          WHERE id_visit = OLD.id_visit;
```

```
            UPDATE DOG SET num_vaccines = num_vaccines - 1
            WHERE id_dog = dog_identifier;

      END IF;
      RETURN NULL;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER trigger_update_dog_vaccines
AFTER INSERT OR DELETE OR UPDATE OF id_visit ON VACCINATION
FOR EACH ROW EXECUTE PROCEDURE update_dog_vaccines();
```

Con las siguientes pruebas podremos validar cada una de las casuísticas propuestas utilizando como base el fichero *inserts_db.sql* que os damos en el enunciado.

Es importante volver a crear la base de datos, de manera que el orden de ejecución de los *scripts* sea:

1. **create_db.sql**
2. **Creación del procedimiento y el disparador/disparadores**
3. **inserts_db.sql**

Primero hacemos la consulta sobre *DOG* par ver si se ha calculado correctamente el valor de *num_vaccines* después de insertar los datos.

```
SELECT d.id_dog, d.name_dog, d.breed, d.id_owner AS OWN, d.num_vaccines AS
VAC
FROM DOG d
ORDER BY VAC DESC;
```

```
 id_dog | name_dog |             breed              | own | vac
--------+----------+--------------------------------+-----+-----
     22 | Chloe    | Boxer                          |  12 |   5
      6 | Max      | German Shepherd                |  12 |   4
      2 | Rocky    | Bulldog                        |   1 |   4
      1 | Tom      | Pekingese                      |   4 |   2
     23 | Duke     | Bulldog                        |   8 |   2
     16 | Milo     | Shih Tzu                       |   1 |   1
     39 | Bentley  | Bernese Mountain Dog           |  10 |   1
      9 | Lucy     | Beagle                         |  24 |   1
     14 | Zeus     | Rottweiler                     |  18 |   1
     40 | Lucy     | Cocker Spaniel                 |  16 |   1
     20 | Toby     | Border Collie                  |  24 |   1
     13 | Luna     | Siberian Husky                 |  12 |   1
     41 | Gizmo    | Chihuahua                      |  24 |   1
     34 | Rocky    | Rottweiler                     |  21 |   1
     25 | Maggie   | Chihuahua                      |  21 |   0
     26 | Bear     | Chow Chow                      |  18 |   0
     27 | Gizmo    | Dachshund                      |  13 |   0
     28 | Coco     | French Bulldog                 |   7 |   0
```

```
     29 | Harley   | German Shepherd               |  10 |   0
     30 | Molly    | Golden Retriever              |  16 |   0
     31 | Rusty    | Great Dane                    |  24 |   0
     32 | Zeus     | Great Dane                    |   5 |   0
     33 | Bella    | Poodle                        |   3 |   0
     35 | Sadie    | Beagle                        |  18 |   0
     36 | Max      | Siberian Husky                |  13 |   0
     37 | Daisy    | Boxer                         |   7 |   0
     38 | Yago     | French Bulldog                |   5 |   0
      3 | Linda    | Labrador Retriever            |   3 |   0
     43 | Laika    | French Bulldog                |   5 |   0
     44 | Thor     | Great Dane                    |   5 |   0
     45 | Afrodita | Great Dane                    |   5 |   0
     42 | Roxy     | Great Dane                    |   5 |   0
      4 | Laika    | French Bulldog                |   5 |   0
      5 | Lassie   | Rough Collie                  |   8 |   0
      7 | Bella    | Pekingese                     |   4 |   0
      8 | Rocky    | Boxer                         |  15 |   0
     10 | Charlie  | Poodle                        |   9 |   0
     11 | Daisy    | Chihuahua                     |  20 |   0
     12 | Cooper   | Golden Retriever              |   7 |   0
     15 | Roxy     | Dachshund                     |   2 |   0
     17 | Sadie    | Australian Shepherd           |  22 |   0
     18 | Bear     | Bulldog                       |   9 |   0
     19 | Bailey   | Bichon Frise                  |   6 |   0
     21 | Sophie   | Boston Terrier                |  20 |   0
     24 | Lucky    | Cavalier King Charles Spaniel |   3 |   0
(45 rows)
```

Ahora eliminamos de la tabla *VACCINATION* la fila con *id_visit = 68* que corresponde al perro con *id_dog = 2*

Antes de eliminarla

```
SELECT d.id_dog, d.name_dog, d.breed, d.id_owner AS OWN, d.num_vaccines AS
VAC
FROM DOG d
WHERE id_dog = 2;
```

```
 id_dog | name_dog |  breed  | own | vac
--------+----------+---------+-----+-----
      2 | Rocky    | Bulldog |   1 |   4
(1 row)
```

```
DELETE FROM VACCINATION WHERE id_visit = 68;
```

Después de eliminarla

```
DELETE 1
```

```
 id_dog | name_dog |  breed   | own | vac
--------+----------+---------+-----+-----
      2 | Rocky    | Bulldog |   1 |   3
(1 row)
```

Finalmente, hacemos una modificación del proceso de vacunación cambiando *id_visit = 10* correspondiente al perro con *id_dog = 6* por la visita con *id_visit = 8* correspondiente al perro con *id_dog = 2* .

```
UPDATE VACCINATION SET id_visit = 8 WHERE id_visit = 10;
```

Antes de la modificación

```
SELECT d.id_dog, d.name_dog, d.breed, d.id_owner AS OWN, d.num_vaccines AS
VAC
FROM DOG d
WHERE id_dog IN (2,6);
```

```
 id_dog | name_dog |      breed      | own | vac
--------+----------+-----------------+-----+-----
      6 | Max      | German Shepherd |  12 |   4
      2 | Rocky    | Bulldog         |   1 |   3
(2 rows)
```

Después de la modificación

```
UPDATE 1
 id_dog | name_dog |      breed      | own | vac
--------+----------+-----------------+-----+-----
      6 | Max      | German Shepherd |  12 |   3
      2 | Rocky    | Bulldog         |   1 |   4
(2 rows)
```

Los perros con *id_dog = 6* y *id_dog = 2* pasan a tener una vacuna menos y una más, respectivamente.