**Procedimientos almacenados y disparadores, ¿para qué son necesarios?**

**NOMBRE Y APELLIDOS: \_Alan David Coila Bustinza**

Los *Juegos Atléticos Olímpicos UOC*, ahora que dispone de la base de datos que hemos construido en la UOC como parte de la asignatura **Bases de datos para *Data Warehousing***, ha obtenido el presupuesto necesario para implementar una serie de mejoras. De nuevo, se ha puesto en contacto con nosotros para que implementéis los requisitos que nos han propuesto.

Para la implementación de esta PEC, debéis de crear una base de datos nueva denominada **dbdw\_pec3** y ejecutar el script adjunto **DB\_olympic\_structure.sql**. El segundo script proporcionado, **DB\_olympic\_data.sql,** os dará un conjunto de datos que os permitirá implementar los componentes requeridos en los diferentes apartados de esta PEC.

Consideraciones para la entrega y realización de la PEC:

* Todo lo que se pide en esta PEC está explicado en los bloques didácticos 2 y 3 (salvo que se trate de un ejercicio de investigación, cuyo enunciado lo especificará). No es necesario adelantar el estudio del material de otros bloques didácticos para la realización de esta PEC.
* En esta PEC trabajaremos procedimientos/funciones y disparadores. Al tratarse de objetos más complejos, debéis de asegurar una correcta ejecución de estos. Se recomienda que probéis de forma exhaustiva los componentes programados y os creéis vuestros casos de prueba utilizando la base de datos proporcionada.
* Se recomienda la utilización de **pgAdmin** para la implementación de toda la PEC. Existe otra alternativa que es ***psql*** (línea de comandos), pero es preferible que utilicéis pgAdmin ya que es una interfaz gráfica que os permitirá editar y crear sentencias SQL (así como mostrar los resultados) de forma más sencilla que *psql*.
* Tal y como se indica en el enunciado, cada respuesta a los ejercicios ha de entregarse en un fichero *.sql* diferente, con el nombre correspondiente. Se evaluará el código entregado en estos ficheros *.sql* y **NO el código que aparezca en el documento o en los pantallazos adjuntos**.
* Las capturas de pantalla de los ejercicios (y explicaciones pertinentes) han de proporcionarse en un documento aparte (se proporciona una plantilla para el caso, **indicad vuestro nombre en el documento**, por favor).
* Se debe de realizar la entrega de todos los ficheros de la PEC (tanto los ficheros .*sql* como el documento con explicaciones y capturas de pantalla) en un fichero comprimido *.zip*.

Consideraciones para la evaluación del ejercicio:

* Se tendrá en cuenta la aplicación de las buenas prácticas de codificación en SQL, de consultas y de programación de procedimientos y disparadores. Es decir: código con sangrado, uso de cláusulas SQL de forma correcta, comentarios, cabeceras en el procedimiento, etc.
* Los *scripts* proporcionados por el estudiante con las soluciones de los ejercicios han de ejecutarse correctamente. El estudiante ha de asegurarse de que lanzando el *script* completo de cada ejercicio no produzca ningún error.
* **Importante**: Las sentencias SQL proporcionadas en los *scripts* han de ser creadas de forma manual y no mediante asistentes que PostgreSQL/pgAdmin puedan proporcionar. Se pretende aprender SQL y no la utilización de asistentes.
* Las sentencias SQL proporcionadas en los ejercicios han de ser **solamente** aquellas que pide el enunciado y ninguna otra más. Cualquier sentencia añadida a mayores, si está mal o provoca que el *script* no se ejecute correctamente a la hora de corregirlo, penalizará el ejercicio.

**EJERCICIO 1** **(30%)**

**A)**

SET SEARCH\_PATH TO olympic;

ALTER TABLE tb\_register

    ALTER COLUMN register\_date SET DATA TYPE timestamp;

ALTER TABLE tb\_register

    RENAME COLUMN register\_date TO register\_ts;

ALTER TABLE tb\_register

    ADD COLUMN register\_updated timestamp;

B)

CREATE OR REPLACE FUNCTION fn\_register\_inserted()

RETURNS TRIGGER AS $$

BEGIN

    new.register\_updated = new.register\_ts;

RETURN NEW;

END

$$

LANGUAGE plpgsql;

CREATE TRIGGER tg\_register\_inserted

BEFORE INSERT ON tb\_register

FOR EACH ROW

EXECUTE PROCEDURE fn\_register\_inserted();

C)

CREATE FUNCTION fn\_register\_updated()

RETURNS TRIGGER AS $$

BEGIN

    new.register\_updated = now();

RETURN NEW;

END

$$

LANGUAGE plpgsql;

CREATE TRIGGER tg\_register\_updated

BEFORE UPDATE ON tb\_register

FOR EACH ROW

EXECUTE PROCEDURE fn\_register\_updated();

**EJERCICIO 2** **(50%)**

**A)**

CREATE DOMAIN email\_type AS TEXT

  CHECK ( value ~ '^[a-zA-Z0-9.!#$%&''\*+/=?^\_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)\*$' );

ALTER TABLE tb\_sponsor

    ADD COLUMN email email\_type;

ALTER TABLE tb\_collaborator

    ADD COLUMN email email\_type;

**B)**

CREATE TABLE olympic.tb\_athletes\_info\_log(

    athlete\_id CHAR (7) NOT NULL,

    discipline\_id INT NOT NULL,

    round\_number INT NOT NULL,

    athlete\_name VARCHAR (50) NOT NULL,

    discipline\_name VARCHAR (50) NOT NULL,

    mark VARCHAR (12) NOT NULL,

    rating INT NOT NULL,

    info\_log\_dt DATE,

    CONSTRAINT pk\_athlete\_info\_log PRIMARY KEY (athlete\_id,discipline\_id,round\_number),

    CONSTRAINT fk\_athlete\_info\_log FOREIGN KEY (athlete\_id,discipline\_id,round\_number) REFERENCES olympic.tb\_register (athlete\_id,discipline\_id,round\_number)

);

**C)**

ALTER TABLE olympic.tb\_athletes\_info\_log

drop CONSTRAINT fk\_athlete\_info\_log;

ALTER TABLE olympic.tb\_athletes\_info\_log

ADD CONSTRAINT fk\_athlete\_info\_log

   FOREIGN KEY (round\_number, athlete\_id, discipline\_id)

   REFERENCES olympic.tb\_register (round\_number, athlete\_id, discipline\_id)

   ON DELETE CASCADE ON UPDATE CASCADE;

CREATE OR REPLACE FUNCTION fn\_athletes\_info()

RETURNS TRIGGER AS $$

DECLARE

    name\_2 varchar (50);

    discipline\_2 varchar (50);

    marck\_athlete varchar (12);

    fecha DATE;

BEGIN

        name\_2 = (

                SELECT name

                FROM tb\_athlete a

                WHERE a.athlete\_id = new.athlete\_id

            );

        discipline\_2 = (

            SELECT name

            FROM tb\_discipline a

            WHERE a.discipline\_id = new.discipline\_id

        );

        fecha = now();

        IF new.register\_time  isnull then

            marck\_athlete = new.register\_measure;

        ELSE

            marck\_athlete = new.register\_time;

        END IF;

        IF (TG\_OP = 'INSERT') THEN

            INSERT INTO tb\_athletes\_info\_log values (

                new.athlete\_id,

                new.discipline\_id,

                new.round\_number,

                name\_2,

                discipline\_2,

                marck\_athlete,

                new.register\_position,

                fecha

            );

        ELSIF (TG\_OP = 'DELETE') THEN

            DELETE FROM  tb\_athletes\_info\_log

                WHERE athlete\_id = new.athlete\_id

                AND discipline\_id = new.discipline\_id

                AND round\_number = new.round\_number;

        ELSIF (TG\_OP = 'UPDATE') THEN

            UPDATE tb\_athletes\_info\_log

            SET (mark , rating) = ( marck\_athlete ,new.register\_position)

            WHERE athlete\_id = new.athlete\_id

                AND discipline\_id = new.discipline\_id

                AND round\_number = new.round\_number;

        END IF;

RETURN new;

END;

$$

LANGUAGE plpgsql;

CREATE TRIGGER tg\_athletes\_info

AFTER INSERT OR DELETE OR UPDATE ON tb\_register

FOR EACH ROW

EXECUTE PROCEDURE fn\_athletes\_info();

**D)**

CREATE OR REPLACE FUNCTION fn\_get\_info\_by\_sponsor(select\_date DATE ,sponsor character varying)

RETURNS TABLE (

    sponsor\_email olympic.email\_type,

    sponsor\_name VARCHAR (50),

    athlete\_n VARCHAR (50),

    discipline\_n VARCHAR (50),

    round\_n INT,

    athlete\_ma VARCHAR (12),

    register\_p INT,

    information\_d DATE

)

AS $$

BEGIN

RETURN QUERY

        SELECT c.email , c.name , a.athlete\_name, a.discipline\_name, a.round\_number,a.mark,a.rating,a.info\_log\_dt

        FROM olympic.tb\_athletes\_info\_log a

        LEFT JOIN olympic.tb\_finance b ON a.athlete\_id = b.athlete\_id

        LEFT JOIN olympic.tb\_sponsor c ON b.sponsor\_name = c.name

        WHERE sponsor = c.name

        AND a.info\_log\_dt = select\_date;

END; $$

LANGUAGE plpgsql

**EJERCICIO 3** **(20%)**

A)

¿Qué son los JSON?

El acrónimo JSON significa JavaScript Object Notation, es una extensión de archivo mas muy util para el intercambio de información debido a ser muy ligero, su facilidad de lectura, de análisis y de generación. Están conformados por las claves y los valores, en forma de parejas clave:valor

¿Qué son los arrays?

Son colección de elementos del mismo tipo que permiten implementar estructuras de datos mediante el uso de índices.

¿Qué diferencia hay entre arrays y JSON?

La principal diferencia es el que mientras JSON se trata de una extensión que puede incluir datos como tipo int , string o inclusive array; mientras Array es una estructura de datos que contiene un grupo de elementos

B)

CREATE OR REPLACE FUNCTION fn\_get\_info\_by\_sponsor\_json(select\_date DATE ,sponsor character varying)

RETURNS table (j json)

AS $$

BEGIN

RETURN QUERY

SELECT array\_to\_json(array\_agg(row\_to\_json(t)))

FROM (

    SELECT c.email , c.name , a.athlete\_name, a.discipline\_name, a.round\_number,a.mark,a.rating,a.info\_log\_dt

    FROM olympic.tb\_athletes\_info\_log a

    LEFT JOIN olympic.tb\_finance b ON a.athlete\_id = b.athlete\_id

    LEFT JOIN olympic.tb\_sponsor c ON b.sponsor\_name = c.name

    WHERE sponsor = c.name

    AND a.info\_log\_dt = select\_date

) t;

END; $$

LANGUAGE plpgsql

**Criterios de valoración**

En el enunciado se indica el peso/valoración de cada ejercicio.

Para conseguir la puntuación máxima en los ejercicios, es necesario explicar con claridad la solución que se propone.

**Formato y fecha de entrega**

Tenéis que enviar la PEC al buzón de Entrega y registro de EC disponible en el aula (apartado Evaluación). El formato del archivo que contiene vuestra solución puede ser **.pdf, .doc y .docx**. **Para otras opciones, por favor, contactar previamente con vuestro consultor**. El nombre del fichero debe contener el código de la asignatura, vuestro apellido y vuestro nombre, así como el número de actividad (PEC3).

La fecha límite para entregar la PEC3 es el **09/12/2021**.

|  |
| --- |
| Nota: **Propiedad intelectual**  Al presentar una práctica o PEC que haga uso de recursos ajenos, se tiene que presentar junto con ella un documento en que se detallen todos ellos, especificando el nombre de cada recurso, su autor, el lugar donde se obtuvo y su estatus legal: si la obra está protegida por el copyright o se acoge a alguna otra licencia de uso (Creative Commons, licencia GNU, GPL etc.). El estudiante tendrá que asegurarse que la licencia que sea no impide específicamente su uso en el marco de la práctica o PEC. En caso de no encontrar la información correspondiente tendrá que asumir que la obra está protegida por el copyright.  Será necesario, además, adjuntar los ficheros originales cuando las obras utilizadas sean digitales, y su código fuente, si así corresponde. |