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

**NOMBRE Y APELLIDOS: _____**

La empresa "UOC Salud", 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 **BD_DW3** y ejecutar el script adjunto **BBDD_Clinical_structures.sql**. El segundo script proporcionado, **BBDD_Clinical_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 (20%)**

La empresa está teniendo problemas gestionando ciertas fechas en la base de datos. En concreto, han detectado que por error humano se llegan a introducir ciertas incoherencias que a toda costa deben ser evitadas:

1. En *tb_patient*, quieren evitar la posibilidad de registrar pacientes cuyo *birth_dt* tenga valor nulo.

2. En *tb_encounter*, quieren asegurarse de que *discharge_dt* sea siempre mayor o igual que *arrival_dt*.

3. En *tb_orders*, quieren evitar que los valores insertados en *created_dt* puedan ser modificados.

Para cada apartado del ejercicio debéis valorar si es necesario o no el uso de *triggers*, teniendo en cuenta que se debe evitar el uso de *triggers* si PostgreSQL permite realizar dicha tarea sin ellos.

**Toda modificación que se realice a la base de datos debe ser creada en el esquema *clinical* y se debe entregar en un fichero llamado pec3_ej1.sql.**

**EJERCICIO 2 (40%)**

La empresa nos pide ahora que implementemos un reporte en el esquema clinical en el que se analizarán las prestaciones anuales realizadas:

**a) (10%)** Se pide crear el procedimiento *catalog_yearly_orders,* de manera que:

- Tiene como parámetros el año y el código de la opción del catálogo (year, order_code).
- Devuelve el número de prestaciones realizadas para esa opción del catálogo y ese año.
- El procedimiento no debe devolver nunca NULL, por lo que en caso que no existan prestaciones realizadas para los parámetros de entrada, se debe devolver 0.

**b) (10%)** Se pide crear el procedimiento *yearly_orders,* de manera que:

- Tiene como parámetro el año (year).
- Debe devolver el número de prestaciones realizadas para ese año.
- El procedimiento no debe devolver nunca NULL, por lo que en caso que no existan prestaciones realizadas para los parámetros de entrada, se debe devolver 0.

**c) (20%)** Se pide crear el procedimiento *summary_orders*, mostrando los porcentajes de incidencia anuales de cada opción del catálogo, sobre el total de prestaciones realizadas en cada año. El procedimiento tiene las siguientes características:

- No tiene parámetros.
- Reutiliza los procedimientos del apartado a) y b)
- Debe devolver las siguientes columnas:
  - year: el año
  - order_desc: la descripción de la opción del catálogo
  - percentage: prestaciones para ese año y opción (se obtiene con *catalog_yearly_orders*) divididos por las prestaciones para ese año (se obtiene con *yearly_orders*).
- El resultado debe estar ordenado por el año y el código de la opción del catálogo, ascendentemente.

Por ejemplo una fila resultante del reporte podría ser **(2019, Consulta sucesiva cardiología, 0.06)**, que se interpreta de la siguiente forma: “durante el año 2019, las prestaciones realizadas para la opción “Consulta sucesiva cardiología” del catálogo supusieron un 6% de todas las prestaciones realizadas durante ese año”.

**Toda modificación que se realice a la base de datos debe ser creada en el esquema *clinical* y se debe entregar en un fichero llamado pec3_ej2.sql.**

## EJERCICIO 3 (30%)

La empresa nos ha pedido que implementemos una serie de novedades en nuestra base de datos para poder gestionar una serie de requisitos.

**a) (10%)** Crea un disparador que evite que las filas de *tb_orders* con status *'Cancelada'* puedan actualizarse a status *'Realizada'* y viceversa.

**b) (5%)** Crea una tabla *tb_orders_status_changelog* que capture las actualizaciones de estado de las órdenes. Nota: solo hay que capturar los cambios en tb_orders.status.

El formato de esta tabla ha de ser el siguiente:

- El id del order cuyo *tb_orders.status* ha sido alterado
- El estado antiguo
- El estado nuevo
- La fecha y hora de cuándo capturó el evento

| **Tabla:** tb orders status changelog **Esquema:** clinical | | | | |
|---|---|---|---|---|
| **Nombre columna** | **Tipo de datos** | **Acepta Nulos** | **Clave primaria** | **Clave foránea** |
| order_id | Numérico entero | No | No | tb_order |
| old_status | Cadena de 50 caracteres variable | No | No | |
| new_status | Cadena de 50 caracteres variable | No | No | |
| changelog_dt | Fecha con tiempo | No | No | |

**c) (15%)** Crea un disparador que capture cuando se ha cambiado una orden de estado. El disparador en cuestión debe insertar una nueva fila en *tb_orders_status_changelog* cada vez que detecte un cambio de estado en una orden.

**Toda modificación que se realice a la base de datos, debe ser creado en el esquema *clinical* y se debe entregar en un fichero llamado pec3_ej3.sql.**

**EJERCICIO 4 (10%)**

Desde la empresa nos piden que realicemos unas tareas de investigación y práctica sobre PostgreSQL, como nuevos expertos que sois.

En concreto se nos pide explicar a qué nos referimos cuando decimos que una función en PostgreSQL es volátil por defecto, y enumerar las posibles alternativas que tenemos.

La explicación debe acompañarse con ejemplos simples que ayuden a facilitar la comprensión de la respuesta.

Cuando se crea una FUNCTION por defecto es configurada como VOLATILE, lo que significa que cada vez que se ejecute puede que cambie el estado de la base de datos (realiza INSERT, UPDATE y/o DELETE) y que además su valor de retorno puede ser diferente para distintas invocaciones, incluso cuando llevan los mismos valores de entrada. Un ejemplo de este tipo de funciones es la función *tg_orders_status_changelog* (solución al ejercicio 3C) la cual modifica el estado de la base de datos cada vez que se ejecuta.

Las alternativas que tenemos a este modelo son IMMUTABLE Y STABLE:

- Las funciones IMMUTABLE son aquellas que aseguran que los valores que retornan serán siempre los mismos para las mismas entradas. Además nunca deben alterar el estado de la base de datos ni realizar consultas a tablas, y solo podrán llamar a otras funciones si son también inmutables. Un ejemplo de función inmutable es *md5(string)*.
- Las funciones STABLE son aquellas que aseguran que los valores que retornan serán los mismos para las mismas entradas durante la ejecución de un mismo statement (por ejemplo en un SELECT). Además nunca deben alterar el estado de la base de datos y solo podrán llamar a otras funciones sin son también inmutables o stables. Un ejemplo de función stable es *yearly_orders(integer)*, solución al ejercicio 3C.

La utilidad de estas dos es la de indicar a PostgreSQL cuándo realmente es necesario ejecutar más de una vez la misma función con los mismos parámetros. En el caso de las funciones inmutables con una sola vez será suficiente. En el caso de usar stable, no será necesaria evaluarla más de una vez dentro del mismo statement e.j. en el WHERE de una consulta.

## 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 **09/05/2021**.

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.