# Práctica 2. Programación Web y Bases de Datos

*Versión 1.0 26/10/2020*

## Objetivos docentes

Tras la realización de la Práctica 2, el estudiante debe ser capaz de:

- Entender los conceptos involucrados en el diseño de una base de datos relacional, en especial las relaciones que se deben establecer entre las distintas tablas.
- Realizar consultas de diferente complejidad sobre bases de datos relacionales.
- Entender y manejar con agilidad los triggers en una base datos.
- Entender y ser capaz de implementar Funciones y Procedimientos Almacenados.
- Acceder a bases de datos desde el lenguaje Python con la interfaz SQLAlchemy, realizando actualizaciones y consultas bajo demanda del usuario.

## Base de Datos

Para facilitar la tarea, se proporciona una base de datos volcada en un fichero de texto (fichero incluido en **dump_v1.3.sql.gz**) que servirá de punto de partida para implementar lo solicitado en la práctica. Para cargar este volcado, suponiendo que el usuario alumnodb ya existe en PostgreSQL (consultar las instrucciones de la plataforma), podemos ejecutar los siguientes comandos:

```
createdb -U alumnodb si1
gunzip -c dump_v1.3.sql.gz | psql -U alumnodb si1
```

En los anteriores comandos, **si1** es el nombre que se le pone a la base de datos. **Debe utilizarse siempre este nombre para facilitar la corrección del código entregado**. El primer comando crea la propia base de datos, mientras que el segundo descomprime el volcado y rellena la base de datos a partir de lo especificado en él.

Grosso modo, la base de datos está organizada en las siguientes tablas:

- *imdb_movies*: representa las distintas películas que se pueden comprar.
- *products*: almacena los distintos productos que pueden existir de la misma película con sus precios.
- *inventory*: indica las ventas y el stock de cada producto.
- *customers*: guarda la información relativa a los diferentes clientes.
- *orderdetail*: detalla las compras concretas que han ido realizando los clientes a través de la web.
- *orders*: agrupa los productos de *orderdetail* en pedidos (carritos finalizados, es decir, que se han pagado) propios de cada usuario.

Algunas características adicionales de esta base de datos son:

- Un pedido en curso (cesta o carrito), se caracteriza por tener un valor NULL (valor reservado de SQL, no una cadena de caracteres) en la columna *status* de la tabla *orders*. Por ello, sólo puede haber como máximo un registro de la tabla *orders* con *status* NULL para un cliente dado.
- El pedido pasa sucesivamente por los siguientes valores de *status*: NULL, '*Paid*', '*Processed*', '*Shipped*'.

- La columna *sales* de la tabla *inventory* contiene el número acumulado de artículos vendidos de un producto.

# Tareas de la práctica

Se solicitan a continuación una serie de tareas e informes sobre la base de datos. Si se encuentra que los datos existentes no son suficientes para dar respuesta a estas peticiones, el alumno deberá incorporar a la misma la información extra necesaria.

## *Diseño de la BD*

a) En primer lugar, se deberá efectuar un proceso de ingeniería inversa sobre la base de datos suministrada. En particular, se deberá:

- Obtener el diagrama entidad-relación correspondiente. Se puede realizar manualmente o utilizando las herramientas que sean necesarias.
- Discutir el diseño de la base de datos, ventajas y desventajas de las decisiones implícitas en el mismo.
- Completar aquellos aspectos que se consideren necesarios, tales como restricciones, claves extranjeras, cambios en cascada, etc. Todos estos cambios se incorporarán en un único script, **actualiza.sql**.
- Se deberá entregar, al menos, el diagrama E-R final, tras los cambios del script actualiza.sql. A la hora de elaborar este diagrama conviene revisar:
  - claves primarias
  - claves externas o extranjeras
  - qué tablas son entidades, cuáles relaciones y cuáles atributos
  - cardinalidad
  - entidades débiles
  - atributos multivaluados
  - atributos derivados
  - participación total

b) Sabiendo que los precios de las películas se han ido incrementando un 2% anualmente, elaborar la consulta **setPrice.sql** que complete la columna 'price' de la tabla 'orderdetail', sabiendo que el precio actual es el de la tabla 'products'.

c) Una vez se disponga de esta información, realizar un procedimiento almacenado, **setOrderAmount**, que complete las columnas 'netamount' (suma de los precios de las películas del pedido) y 'totalamount' ('netamount' más impuestos) de la tabla 'orders' cuando éstas no contengan ningún valor. Invocadlo para realizar una carga inicial.

d) Realizar una función postgreSQL, **getTopVentas**, que reciba como argumentos dos años diferentes y devuelva las películas que más se han vendido entre esos dos años, una por año, ordenadas de mayor a menor por número de ventas.

| AÑO | PELICULA | VENTAS |
|---|---|---|
| 1897 | El Gran Dictador | 12 |
| 1912 | Superman | 4 |
| .... | | |

(Sólo es un ejemplo, no son resultados reales)

e) Realizar una función postgreSQL, **getTopMonths**, que reciba unos umbrales de número de productos y de importe ('totalamount') acumulados, y devuelva los meses (en realidad la pareja año-mes) en los que se ha superado alguno de los dos umbrales, junto con su importe y productos. Probadla con umbrales de 19.000 artículos y 320.000 euros.

| Año | Mes | Importe | Productos |
|---|---|---|---|
| 2016 | 3 | 250000 | 25000 |
| 2017 | 5 | 340000 | 10000 |
| | | | |

(Sólo es un ejemplo, no son resultados reales)

## *Integridad de los datos*

f) Para garantizar la integridad de los datos (los valores posibles de las columnas), crear las tablas correspondientes y convertir los atributos multivaluados 'moviecountries', 'moviegenres' y 'movielanguages' en relaciones entre la tabla 'movies' y las tablas creadas. Estos cambios también se incorporarán al script **actualiza.sql**.

g) Realizar un trigger, **updOrders**, que actualice la información de la tabla 'orders' cuando se añada, actualice o elimine un artículo al carrito.

h) Realizar un trigger, **updInventory**, que actualice las tablas 'inventory' y 'orders' cuando se finalice la compra. El trigger también deberá crear una alerta en una nueva tabla llamada '*alertas*' si la cantidad en stock llega a cero. Realizar los cambios necesarios en la base de datos para incluir dicha tabla, incorporándolos al script actualiza.sql.

## *Integración en el portal*

i) Incorporar la tabla resultante en el apartado d) anterior, para los últimos tres años, a la página inicial de bienvenida.

j) Implementar en el sitio web el registro y la validación de usuario (login) usando la tabla 'customers'. No es preciso cifrar las contraseñas.

k) Implementar el resto de páginas con contenidos de películas. Para ello, tened en cuenta que las películas que se muestran y compran deben ser un subconjunto de las que existen en la BD para que se cumpla la integridad referencial al añadirlas al carrito (es decir, la estructura JSON con la que el sitio web juega para mostrar la información debe estar creada con la información de la BD, aunque tenga un número limitado de películas).

l) Implementar en el sitio web la funcionalidad de carrito, usando las tablas de la base de datos apropiadas. Se puede mantener la funcionalidad de carrito por sesión de la práctica 1 para el caso de navegación por el sitio web sin haber hecho login.
(***Nota***: Se debe usar la BD para almacenar el carrito desde el primer artículo que se introduzca en él; no es válido ir guardando el carrito en la sesión y volcarlo a la BD al hacer *logout*, etc.)

El código Python de acceso a la base de datos deberá usar la interfaz implementada en SQLAlchemy (https://www.sqlalchemy.org/).

# Entregables

Como **resultado** de la práctica se entregará:

- Código fuente:
    - Parte SQL: como se ha especificado más arriba, los ficheros SQL que hay que entregar son (los nombres de los archivos se corresponden con los de los triggers, funciones, etc. que contienen):
        - **actualiza.sql:** script que transforma la base de datos suministrada en la de trabajo de la práctica, tras corregir errores, incorporar mejoras, etc.
        - **setPrice.sql**
        - **setOrderAmount.sql**
        - **getTopVentas.sql**
        - **getTopMonths.sql**
        - **updOrders.sql**
        - **updInventory.sql**
    - En el caso de implementar algún ejercicio optativo, todos los ficheros adicionales que resulten.
    - **La base de datos como tal (dump) no debe entregarse, debido a su tamaño.** Se entiende que el script 'actualiza.sql' es el que transforma el dump suministrado originalmente en la base de datos propia de cada pareja.
    - Parte web: Python y auxiliares (CSS, JS, HTML…). Recordar **no entregar el entorno de python si1pyenv**
- Memoria en la que se incluirá:
    - El diagrama entidad-relación de la base de datos resultante tras aplicar el script 'actualiza.sql' y los cambios introducidos, con su justificación.
    - El análisis de la solución dada a las consultas, triggers y funciones solicitados.
    - Las evidencias de los resultados obtenidos.
    - Cualquier mejora o propuesta de implantación que el alumno considere.

  **IMPORTANTE:** Si es necesario incorporar datos adicionales para completar las tablas del modelo y para hacer pruebas, estos cambios deben ser documentados debidamente en la memoria entregada.

# Fechas

Esta práctica se dividirá en dos entregas. La primera entrega, si bien obligatoria, no será evaluada y servirá para ver el progreso del proyecto. El contenido de las entregas será:

- Semana del 9/11: Se deberá entregar el diagrama entidad relación, las consultas implicadas, las funciones postgreSQL, triggers y procedimientos almacenados.
- Semana del 23/11: Entrega final con la práctica completa: la funcionalidad completa de la práctica 2 con la integración adecuada entre el código Python y la base de datos desarrollada.

La fecha concreta y normas de entrega de las prácticas se encuentran en Moodle.

# Evaluación

Se valorará muy positivamente cualquier mejora o añadido adicional que aporte el alumno, discusión sobre la eficiencia del método empleado, etc.

**Revisad que el formato de las sentencias SQL de consulta se adecúe a las Normas de Prácticas**.

# Consejos

## *SQL*

Si se hace '*update mi_tabla set mi_columna = (select …* ' revisar si se puede evitar vía '*update …. from …*'. De igual forma, si se hace un '*delete*' similar (que use un '*select*') revisar '*delete …. using …*'.

Si se hace ' *… from (select …*' revisar si de verdad es necesario ese select o se puede sustituir con un join.

En las funciones de base de datos, intentar evitar los bucles y usar las capacidades de las sentencias SQL: por ejemplo, una sola sentencia '*update*' puede cambiar muchos registros, no hace falta un bucle que los recorra todos.

Si la función de un trigger no hace referencia a '*NEW*' ni a '*OLD*', casi seguro que está mal, es muy ineficiente (en esta práctica, seguro que es así).

Al programar la función de un trigger hay que tener siempre cuidado de no ejecutar código cuando no se debe, por ejemplo, porque se ha actualizado sólo una columna que no es la relevante de cara a la ejecución del trigger.

Recordad que, en un trigger, la variable TG_OP indica si el trigger se disparó por un INSERT, UPDATE o DELETE.

Recordad que un diagrama E-R no es un diagrama de diseño de una base de datos, con sus tablas y relaciones.

El uso de *pgadmin3* puede dar problemas al guardar los ficheros. En particular, añade una cabecera de la UTF8 (‘BOM’) que afecta a la posterior ejecución de los scripts de BD. Se puede utilizar un editor como *geany* para eliminarlos (opción Document->uncheck ‘Write Unicode BOM’.

# SQLAlchemy

Se suministra un ejemplo de recuperación de datos de la BD en example-web-alchemy-v<x.y>.zip. Se recomienda revisarlo detenidamente.

SQLAlchemy provee dos interfaces de acceso a la BD: vía objetos (ORM) o vía funciones (Core). Se

recomienda usar SQLAlchemy Core.

Para ejecutar sentencias SQL se puede usar el método *execute* de una *connection* (ver https://docs.sqlalchemy.org/en/latest/core/connections.html#sqlalchemy.engine.Connection).

El resultado de una consulta se devuelve vía el objeto *ResultProxy* (https://docs.sqlalchemy.org/en/latest/core/connections.html#sqlalchemy.engine.ResultProxy), que contiene las filas resultantes.

Cada fila del resultado es un objeto *RowProxy* (https://docs.sqlalchemy.org/en/13/core/connections.html#sqlalchemy.engine.RowProxy). Los valores de cada columna se obtienen vía *items()*, que devuelve un array de tuplas clave-valor.

También se puede recuperar la información a través de los métodos *first()*, *fetchone()*, *fetchall()*, … del objeto *ResultProxy*, que funcionan de forma análoga a los cursores.

# Bibliografía

[1] Documentación de postgreSQL: https://www.postgresql.org/docs/10/index.html
[2] Acceso a bases de datos en Python usando SQLAlchemy: https://www.sqlalchemy.org/
[3] PgAdmin: http://www.pgadmin.org/