

BBDD

Bases de Datos – Programación de bases de datos II



Objetivos

- Identificar y diferenciar los distintos tipos de encapsulamiento de código y su aplicación: bloques anónimos, procedimientos almacenados, funciones de usuario, eventos, Triggers.
- Desarrollar los distintos tipos de eventos, evaluando y utilizando las sentencias del lenguaje incorporado en el SGBD.
- Desarrollar los distintos tipos de disparadores, evaluando y utilizando las sentencias del lenguaje incorporado en el SGBD.
- Utilizar los recursos de programación en las encapsulaciones de código: cursores, transacciones y la gestión de excepciones.



Contextualización

La **programación de Bases de Datos** está planificada para desarrollarse durante la 3ª evaluación y no antes dada la interdisciplinaridad del módulo con otro módulo (**Programación**).

Se considera que el alumnado ha podido adquirir ya los fundamentos básicos de programación impartidos en el módulo **Programación** tales como:

- Encapsulamiento de código en clases y métodos
- Gestión de variables, operadores y tipos elementales de datos



Interdisciplinaridad

Durante el desarrollo de las UDs 12 y 13, **programación de Bases de Datos**, realizaremos prácticas transversales a los módulos de Programación y Bases de Datos, que serán tenidas en cuenta en la evaluación de ambos módulos.



Programación



Bases de Datos



Encapsulamiento de código (repaso)

- Los SGBDs relacionales más utilizados, tales como ORACLE, SQL Server, PostgreSQL o MySQL incorporan una extensión del lenguaje SQL para añadir la opción de realizar **programación procedimental**.

ORACLE → PL-SQL: Procedural Language – Structured Query Language

SQL SERVER → T-SQL: Transact-SQL

PostgreSQL → PgSQL: Postgress SQL

MYSQL → Conjunto de instrucciones propias 🤖



- Con la **programación procedimental** podemos:
 - Encapsular bloques de código
 - Utilizar variables
 - Utilizar estructuras condicionales
 - Utilizar estructuras repetitivas (bucles)
 - Controlar errores y excepciones
- En general, con la programación procedimental disponible en las bases de datos se puede implementar:
 - Bloques anónimos
 - Procedimientos almacenados**
 - Funciones de usuario**
 - Triggers**
 - Eventos**

Bloques anónimos

- Los **bloques anónimos** son secciones de código de programación de bases de datos que NO se almacenan en la BD sino en un buffer y que además no se les asigna un nombre. Estos bloques se construyen de forma dinámica (en tiempo de ejecución) y se ejecutan una sola vez.
- Basta con utilizar las etiquetas BEGIN y END para crear el bloque anónimo que queramos ejecutar:

```
BEGIN  
    INSTRUCCIONES;  
END
```

- En MySQL NO están habilitado los **bloques anónimos** tal y como sí lo está en otros SGBDs como ORACLE.



- Si queremos ejecutar código de programación en MySQL, SIEMPRE tendremos que encapsularlo en procedimientos almacenados, funciones, triggers o eventos.

Triggers

- Los **triggers** o *disparadores* son objetos que se crean con la sentencia **CREATE TRIGGER** y tienen que estar asociados siempre a una tabla.
- Un trigger se activa cuando ocurre un evento de **inserción** (INSERT), **actualización** (UPDATE) o **borrado** (DELETE), sobre la tabla a la que está asociado.
- Es posible definir también cuando se activará el trigger: si se activará **antes** (BEFORE) o **después** (AFTER) de que se produzca el evento. Los primeros se usan para actualizar o validar registros antes de ser guardados en la BD. Los segundos se usan para acceder a los valores de los campos en modo lectura y para disparar cambios en otros registros.
- Las FK con “*on update cascade*” y “*on delete cascade*” NO disparan la ejecución de los triggers sobre la tabla hija.
- Las variables especiales **OLD** y **NEW** permiten acceder, dentro de un trigger, a los valores del registro afectado antes y después de la modificación respectivamente:

OLD.<nombre_columna> (disponible en delete y update)

NEW.<nombre_columna> (disponible en insert y update)

- Los Triggers pueden ser borrados usando la sentencia:
DROP <IF EXISTS> TRIGGER <nom_trigger>

```
CREATE
  [DEFINER = user]
  TRIGGER [IF NOT EXISTS] trigger_name
  trigger_time trigger_event
  ON tbl_name FOR EACH ROW
  [trigger_order]
  trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

<https://dev.mysql.com/doc/refman/8.0/en/create-trigger.html>

- La sintaxis básica para definir un trigger es:

```
delimiter $$
CREATE TRIGGER nombre_trigger
{BEFORE | AFTER}
{INSERT | UPDATE | DELETE}
ON nombre_tabla
FOR EACH ROW
BEGIN
    ...SENTENCIAS...
END$$
delimiter ;
```

- No se puede invocar un trigger.

SHOW TRIGGERS;

SHOW CREATE TRIGGER xxxx;

Triggers

- La modificación de triggers NO es posible: es necesario borrar y volver a crear el trigger (igual que sucede con los procedimientos almacenados y funciones si queremos modificar el cuerpo o parámetros)
- Si se borra la tabla asociada, todos sus triggers se borran automáticamente (sin confirmación).
- Desde un trigger NO podemos iniciar ni finalizar una transacción (START TRANSACTION, COMMIT o ROLLBACK) pero SI invocar procedimientos almacenados.
- Los **triggers** afectan negativamente al rendimiento de la BD durante las sentencias DML. No se debe por tanto abusar de su uso.
- Algunos SGBDs no soportan la activación de más de un trigger para un mismo evento y momento (ej. insert + before). MySQL sí y se puede ordenar su ejecución (por defecto, orden de creación)
- Suelen implementarse para realizar un **control de cambios** (traceando los cambios y a sus responsables) pero también para ejecutar **reglas del negocio** (ej: cuando el stock de productos descienda de cierta cantidad, que se registre la petición de más productos de forma automática)
- La lógica de los Triggers se ejecuta de forma transparente al usuario. Es decir, este último no toma conciencia siquiera de su existencia.
- Los Triggers SOLO se activan cuando se ejecutan sentencias SQL, NO cuando las acciones se llevan a cabo mediante APIs.
- Si necesitamos impedir la acción asociada al trigger (INSERT, UPDATE o DELETE) podemos desencadenar un error genérico usando la sentencia:

SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'xxxxxxxxxx';