BASES DE DATOS ACTIVAS

1. Motivación: comportamientos activos

- Las BD tradicionales se consideran pasivas
  - No pueden reaccionar automáticamente ante situaciones
    - Ej.: actualizar automáticamente la ruta del autobús de la escuela cuando se suscriban nuevos alumnos/as a dicho servicio → comprobaciones periódicas
- Encapsular comportamiento en las aplicaciones que acceden a la BD
  - Poca modularidad, comportamiento distribuido
- O en una aplicación de monitorización
  - Más modular, pero ¿con qué frecuencia de monitorización?
- Mejor si podemos encapsularlo en la propia BD → BD activas
  - Mejor modularidad
  - No comprobaciones periódicas → reacción inmediata ante cambios

Ejemplos

- Gestión de inventarios
  - Solicitar un producto al proveedor cuando baje su nivel de existencias por debajo de cierto umbral
- Gestión de viajes
  - Si el autobús está lleno y falta bastante tiempo para el viaje, asignar un autobús adicional
- Estrategias de precios en supermercados
  - Si a un producto perecedero le queda poco para caducar, bajar su precio

2. Reglas ECA

- Reglas ECA:
  - Evento: qué dispara la acción, suceso a vigilar
  - Condición: situación que debe darse para ejecutar la acción asociada
  - Acción: el comportamiento que se dispara (qué se hace) cuando se produce el evento y la condición es cierta
- Disparador (*trigger*) = procedimiento invocado automáticamente por el SGBD como respuesta a ciertos cambios en la BD (≈ demonio que monitoriza la BD)

Utilización

- Utilización interna para la explotación o administración de la BD:
  - Implementación de restricciones de integridad
  - Actualización de vistas materializadas (no soportadas de forma transparente por el SGBD) o atributos derivados
  - Seguridad y auditoría: registro de acciones sobre objetos de la BD
  - Gestión de versiones, mantenimiento de datos históricos
  - Realización automática de copias de los datos
- Utilización externa para la implementación de reglas de negocio:
  - Ej.: si pasan 7 días sin que el cliente haya pagado una factura, se le envía un email, se le bloquea la posibilidad de realizar más compras, y se registra como moroso

Eventos (1)

- La fuente del evento puede ser:
  - Una sentencia del LMD:
    - *Insert*, *Delete*, *Update*
    - Antes o después
  - La finalización de una transacción:
    - *Commit*, *rollback*
  - Una excepción:
    - Violación de permisos de acceso
    - Bloqueos
    - …
  - La aplicación
  - El reloj del sistema

Eventos (2)

- Granularidad de los cambios
  - Disparadores a nivel de tupla → un cambio en una sola tupla se considera una ocurrencia del evento
  - Disparadores a nivel de sentencia → los cambios en todas las tuplas se consideran una ocurrencia del evento

Condiciones y acciones

- Condición → un predicado sobre la BD
- Acción:
  - Comandos SQL
  - Comandos PL/SQL (SQL extendido)
  - Llamadas externas
  - Abortar la transacción → *rollback*
    - Ej.: si ningún empleado puede ganar más que su jefe, podemos definir una regla ECA donde el evento sea la inserción en la tabla empleados, la condición que el salario introducido sea mayor que el del jefe, y la acción abortar la transacción

Modos de acoplamiento

- *Modo de acoplamiento inmediato*:
  - La regla ECA se evalúa tan pronto como se detecta el evento
- *Modo de acoplamiento diferido*:
  - La regla ECA se evalúa al finalizar la transacción donde se ha activado (justo antes de hacer *commit*)
- *Modo de acoplamiento desacoplado*:
  - La evaluación de la condición y la ejecución de la acción se ejecutan en transacciones separadas

3. Disparadores (Oracle)

- Tres tipos en Oracle:
  - DML (*Data Manipulation Language*):
    - Disparados por sentencias DML: *INSERT*, *UPDATE*, *DELETE*
    - Condición adicional con la cláusula *WHEN*
  - *INSTEAD OF*:
    - Definidos sobre vistas
    - Permiten transformar una actualización sobre vistas no directamente actualizables en actualizaciones adecuadas sobre las tablas base
  - *Triggers* sobre eventos de sistema o de usuario:
    - Definidos sobre un esquema (*SCHEMA*) o sobre la BD (*DATABASE*)
    - Eventos de sistema: arranque y parada, transacciones, errores
    - Eventos de usuarios: entrada en el sistema, salida del sistema, sentencias DDL (*Data Definition Language*): *CREATE*, *ALTER*, *DROP*
- Están almacenados en la BD

Sintaxis

- Estructura:

```
CREATE [OR REPLACE] TRIGGER nombre
{BEFORE | AFTER | INSTEAD OF}
{INSERT | DELETE | UPDATE [OF <atributos>]} ON <tabla>
[REFERENCING  {NEW AS …, OLD AS …}+]
[FOR EACH ROW]
[WHEN condición]
BEGIN
  cuerpo del disparador (bloque PL/SQL)
END;
```

Condiciones → requieren *FOR EACH ROW*; no pueden especificarse para *triggers INSTEAD OF*.
INSTEAD OF ←→ triggers sobre vistas.
Warning: Trigger created with compilation errors. → SHOW ERROR TRIGGER *nombreDelTrigger*;
o simplemente SHOW ERRORS

Disparadores a nivel de fila vs. a nivel de sentencia: *FOR EACH ROW/STATEMENT*

- Disparadores a nivel de fila (*row triggers*):
  - *FOR EACH ROW*
  - Ejecutan la acción una vez por cada fila que se vea afectada por la sentencia que dispara el *trigger*
  - Lógicamente, no se dispara si no se ve afectada ninguna fila
  - Ejemplo: cuando se borre un cliente de la tabla de clientes, borrarlo también de la tabla de personas
- Disparadores a nivel de sentencia (*statement triggers*):
  - Por defecto (no "*FOR EACH STATEMENT*")
  - Ejecutan una sola vez la acción asociada, con independencia del número de filas afectadas por la sentencia que dispara el *trigger*
  - Ejemplo: cuando se borre en la tabla de clientes mostrar un mensaje de error indicando que no se pueden borrar clientes

Disparadores *BEFORE* vs. *AFTER* vs. *INSTEAD OF*

- Disparadores *BEFORE*:
  - Ejecutan la acción asociada antes de la ejecución de la sentencia correspondiente
  - Por tanto, pueden utilizarse para decidir si la sentencia debe ejecutarse o no o para cambiar la ejecución de la misma (fijar valores alternativos antes de que se escriban en disco)
- Disparadores *AFTER*:
  - Ejecutan la acción asociada después de la ejecución de la sentencia
- Disparadores *INSTEAD OF*:
  - Ejecutan la acción asociada en lugar de la sentencia correspondiente
  - Muy útil para vistas
- El uso de uno u otro tipo de disparador determina la temporalidad del evento considerado

Orden de ejecución de disparadores

- Una misma sentencia SQL puede disparar varios *triggers*
- Además, la activación de un *trigger* también puede disparar la activación de otros *triggers* (*cascading triggers*)
- Orden de ejecución:
  - *BEFORE statement triggers*
  - *BEFORE row triggers*
  - Ejecución de la sentencia
  - *AFTER row triggers*
  - *AFTER statement triggers*

Ojo: todas después de ejecutar la sentencia, aunque sea tupla a tupla

- Disparadores del mismo tipo se ejecutan en un orden arbitrario
  - En Oracle 11g, cláusula *FOLLOWS*

Pseudo-registros: *OLD* y *NEW* (1)

- NEW → tupla nueva
- OLD → tupla vieja

Definidos para disparadores a nivel de fila (se les puede asignar alias con la cláusula REFERENCING)

- De tipo *nombreTabla%ROWTYPE*, donde *nombreTabla* es el nombre de la tabla sobre la que se define el disparador
- *OLD.nombreColumna*:
  - Valor de la columna antes de su cambio por un *UPDATE*
  - Valor de la columna antes de su borrado por un *DELETE*
  - *NULL* en el caso de su inserción por un *INSERT*
- *NEW.nombreColumna*:
  - Valor de la columna después de su cambio por un *UPDATE*
  - Valor de la columna después de su inserción por un *INSERT*
  - *NULL* en el caso de su borrado por un *DELETE*

Pseudo-registros: *OLD* y *NEW* (2)

- Disponibles tanto en disparadores *BEFORE* como *AFTER*:
  - Podemos cambiar valores en *NEW* en el caso de disparadores a nivel de tupla *BEFORE*, pero no en el caso de los AFTER
- Sintaxis:
  - En el cuerpo del disparador → *:NEW*, *:OLD*
  - En la cláusula *WHEN* → *NEW*, *OLD* (sin :)
- Prohibiciones (son pseudo-registros):
  - No se puede cambiar el valor de *OLD*, pero sí el valor de *NEW*
  - No se puede pasar *OLD* o *NEW* como parámetro de tipo registro a un subprograma llamado desde el disparador, pero sí es posible pasar atributos de *OLD* y *NEW*
  - No se pueden realizar operaciones a nivel de registro con *OLD* y *NEW* (p.ej., *:NEW := NULL*)

Predicados condicionales: *DELETING, INSERTING, UPDATING*

```
CREATE OR REPLACE TRIGGER ejemploPredicadosCond
BEFORE INSERT OR UPDATE OR DELETE ON tabla
BEGIN
IF DELETING THEN
  Acciones en caso de borrado
ELSIF INSERTING THEN
  Acciones en caso de inserción
ELSIF UPDATING[('columna1')]
  Acciones en caso de modificación de columna1
ELSIF UPDATING[('columna2')]
  Acciones en caso de modificación de columna2
END IF;
END ejemploPredicadosCond;
/
```

Cuando más de un tipo de operación DML puede disparar el trigger

Para reducir la sobrecarga si no se actualiza la columna de interés (de otro modo, se puede simplemente usar el predicado UPDATING sin indicar la columna)

Lanzamiento de excepciones: *RAISE_APPLICATION_ERROR*

```
CREATE OR REPLACE TRIGGER ejemploExcepcion
BEFORE INSERT ON tabla
FOR EACH ROW
BEGIN
  IF … THEN
    RAISE_APPLICATION_ERROR(-20001,
      'No se permite insertar porque…');
END IF;
...
END ejemploExcepcion;
/
```

Condición que representa una inserción no permitida

Valor en el rango -20000 .. -20999

Zona de declaración de variables

- Antes del *BEGIN* del disparador:
  - *DECLARE*
    declaración de variables
- Sintaxis de declaración de variables:
  - nombreVariable TIPO; → TIPO = *NUMBER, INTEGER, CHAR(n), DATE, …*
  - nombreConstante *CONSTANT* TIPO:=valor;
  - nombreVariable nombreTabla.nombreColumna*%TYPE*
  - nombreVariable nombreTabla*%ROWTYPE*

Habilitación, deshabilitación y eliminación de disparadores

ALTER TABLE *nombreTabla* **ENABLE ALL TRIGGERS**
ALTER TABLE *nombreTabla* **DISABLE** ALL TRIGGERS

ALTER TRIGGER *nombreDisparador* **ENABLE**
ALTER TRIGGER *nombreDisparador* **DISABLE**

Al crearse, los disparadores quedan habilitados por defecto

DROP TRIGGER *nombreDisparador* *Eliminación del disparador*

Obtener información sobre los disparadores definidos (1)

- Vistas del diccionario de datos con información sobre disparadores: *USER_TRIGGERS , ALL_TRIGGERS, DBA_TRIGGERS*
- Algunos atributos interesantes:
  - *TRIGGER_NAME*: nombre del disparador
  - *TRIGGER_TYPE*: *BEFORE STATEMENT*, *BEFORE EACH ROW*, *BEFORE EVENT*, *AFTER STATEMENT*, *AFTER EACH ROW*, *AFTER EVENT*
  - *BASE_OBJECT_TYPE*: *TABLE*, *VIEW*, *DATABASE*, *SCHEMA*
  - *TABLE_NAME* (nulo si el objeto base no es *TABLE* o *VIEW*)
  - *TRIGGERING_EVENT*: evento que dispara el *trigger*
  - *TRIGGER_BODY*: cuerpo del disparador
  - *WHEN_CLAUSE*: cláusula WHEN definida para el disparador
  - *STATUS*: *ENABLED*, *DISABLED*
  - *DESCRIPTION*: descripción del disparador (útil para recrearlo)

Obtener información sobre los disparadores definidos (2)

Algunos ejemplos

```
SELECT TRIGGER_NAME, STATUS FROM
USER_TRIGGERS;

SELECT TRIGGER_BODY
FROM USER_TRIGGERS
WHERE TRIGGER_NAME='nombreDeMiDisparador';

SELECT TRIGGER_TYPE, TRIGGERING_EVENT, TABLE_NAME
FROM USER_TRIGGERS
WHERE TRIGGER_NAME = 'nombreDeMiDisparador ';
```


Disparadores y transacciones (1)

- En general, cuando se aborta la ejecución de un disparador (ej., con *RAISE_APPLICATION_ERROR*)
  - Se deshacen los efectos tanto del disparador como de la sentencia que activó el disparador
- Alternativamente, es posible definir disparadores que contienen transacciones autónomas (para disparadores y procedimientos almacenados):
  - *DECLARE*
    PRAGMA AUTONOMOUS_TRANSACTION;
  - En este caso tenemos una transacción independiente arrancada desde una *transacción principal*
  - Hace *commit* y *rollback* sin afectar a la transacción principal

Disparadores y transacciones (2)

```
CREATE TABLE cambiosSalarios(idCambio NUMBER(6), fechaCambio
   DATE, nuevoSalario NUMBER(8,2), viejoSalario NUMBER(8,2));
CREATE OR REPLACE TRIGGER auditarCambioSalario
AFTER UPDATE OF salario ON Empleados
FOR EACH ROW
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  INSERT INTO cambiosSalarios VALUES(:old.idEmpleado, SYSDATE,
    :new.salario, :old.salario);
COMMIT;
END;
/
```

Tablas mutantes

- Tabla mutante: tabla que está siendo modificada por
  - una sentencia SQL (*INSERT*, *UPDATE*, *DELETE*)
  - o por el efecto de un *DELETE CASCADE* (integridad referencial)
- Restricciones sobre tablas mutantes (*ORA-04091* → *rollback*):
  - En el cuerpo de un disparador no se puede consultar o actualizar una tabla mutante (tabla cuya modificación está activando el disparador) → excepción: si la sentencia disparadora es un *INSERT* de una única tupla
  - Aplicable a disparadores *FOR EACH ROW*
  - Aplicable a disparadores a nivel de sentencia que se disparan como resultado de un borrado en cascada (en principio, sólo hasta Oracle 8i…)
  - Vistas que están siendo modificadas en disparadores *INSTEAD OF* no se consideran mutantes
  - Estas restricciones pretenden evitar que los disparadores puedan ver datos inconsistentes o que se produzca una cascada de ejecución de disparadores

Tablas mutantes: ejemplo

```
CREATE OR REPLACE TRIGGER ContarEmpleados
AFTER DELETE ON Empleado
FOR EACH ROW
DECLARE
  numEmpleados INTEGER;
BEGIN
  SELECT COUNT(*) INTO numEmpleados FROM Empleado;
  DBMS_OUTPUT.PUT_LINE('Hay ' || numEmpleados || '
    empleados.');
END;
```

DELETE FROM emp WHERE empno = 7499;

→

ORA-04091: table Empleado is mutating, trigger/function might not see it

Tablas mutantes: otro ejemplo (1)

```
CREATE TABLE Matriculas(
  nombreAlumno VARCHAR2(100),
  nombreAsignatura VARCHAR2(100),
  constraint pk_matricula PRIMARY KEY(nombreAlumno, nombreAsignatura)
);

CREATE OR REPLACE TRIGGER triggerMatriculas
BEFORE INSERT OR UPDATE ON Matriculas
FOR EACH ROW
DECLARE
  numAsignaturas NUMBER;
BEGIN
  SELECT COUNT(*) INTO numAsignaturas FROM Matriculas
  WHERE Matriculas.nombreAlumno = :NEW.nombreAlumno;
  IF numAsignaturas >= 10 THEN
    RAISE_APPLICATION_ERROR (-20000, 'El alumno ' || :NEW.nombreAlumno || ', ya tiene demasiadas asignaturas');
  END IF;
END;
/
```


Tablas mutantes: otro ejemplo (1)

```
UPDATE Matriculas
SET nombreAlumno = 'Juancho Rancho'
WHERE nombreAlumno  = 'Lucas White';
```

→

```
UPDATE Matriculas
*
ERROR at line 1:
ORA-04091: table MATRICULAS is mutating, trigger/function
    may not see it
ORA-06512: at "TRIGGERMATRICULAS", line 6
ORA-04088: error during execution of trigger
    'TRIGGERMATRICULAS'
```

Tablas mutantes: otro ejemplo (2)

- Solución:
  - Creamos un disparador a nivel de fila (*FOR EACH ROW*) para almacenar los datos que queremos consultar
    - Podemos utilizar una tabla auxiliar (o una tabla PL/SQL o un paquete con variables)
  - Creamos un disparador a nivel de sentencia y *AFTER*, donde realizamos la consulta utilizando los datos almacenados previamente

Otra solución (sólo desde Oracle 11g):

Disparadores compuestos (*compound triggers*)

- Se pueden disparar en más de un momento
- Simplifican la compartición de datos entre las acciones implementadas para esos diversos momentos

Tablas mutantes: otro ejemplo (3)

```
CREATE TABLE tempAlumnos(
  nombreAlumno VARCHAR2(100)
);

CREATE OR REPLACE TRIGGER triggerMatriculas
BEFORE INSERT OR UPDATE ON Matriculas
FOR EACH ROW
DECLARE
  numAsignaturas NUMBER;
BEGIN
IF :NEW.nombreAlumno IS NOT NULL THEN
    BEGIN
      INSERT INTO tempAlumnos VALUES(:NEW.nombreAlumno);
      --dbms_output.put_line('Guardo el alumno: ' || :NEW.nombreAlumno);
    END;
  END IF;
END;
/
```

SET SERVEROUTPUT ON;

Tablas mutantes: otro ejemplo (4)

```
CREATE OR REPLACE TRIGGER triggerMatriculasSentencia
AFTER INSERT OR UPDATE ON Matriculas
DECLARE
  numAsignaturas NUMBER;
  nombreAlumnoLocal Matriculas.nombreAlumno%TYPE;
  CURSOR cursorNombresAlumnos IS SELECT nombreAlumno FROM tempAlumnos;
BEGIN
OPEN cursorNombresAlumnos;
  LOOP
    FETCH cursorNombresAlumnos INTO nombreAlumnoLocal;
    EXIT WHEN cursorNombresAlumnos%NOTFOUND;
    SELECT COUNT(*) INTO numAsignaturas FROM Matriculas WHERE Matriculas.nombreAlumno = nombreAlumnoLocal
    IF numAsignaturas >= 10 THEN
      RAISE_APPLICATION_ERROR (-20000, 'El alumno ' || nombreAlumnoLocal || ', ya tiene demasiadas asignaturas');
    END IF;
  END LOOP;
  CLOSE cursorNombresAlumnos;
  DELETE FROM tempAlumnos;
END;
/
```


Tablas mutantes: otro ejemplo (5)

- Si intentamos una actualización que hace que el número de matrículas de un estudiante pase de 9:

```
UPDATE Matriculas
SET nombreAlumno = 'Juancho Rancho'
WHERE nombreAlumno  = 'Lucas White';
→
UPDATE Matriculas
    *
ERROR at line 1:
ORA-20000: El alumno Juancho Rancho, ya tiene demasiadas asignaturas
ORA-06512: at "TRIGGERMATRICULASSENTENCIA", line 13
ORA-04088: error during execution of trigger 'TRIGGERMATRICULASSENTENCIA'
```

Se hace *rollback* de la transacción

Otros ejemplos de disparadores (1)

Añade al proyecto las horas trabajadas por un empleado

```
CREATE or REPLACE TRIGGER ContabHorasProy
AFTER UPDATE ON Participar
FOR EACH ROW
WHEN (NEW.numHoras > 0)
 begin
  UPDATE Proyecto
  SET    horas = horas + :NEW.numHoras - :OLD.numHoras
  WHERE  numProy = :NEW.numProy;
 end ContabHorasProy;
/
```


Otros ejemplos de disparadores (2)

Refleja en HistorialSalario el cambio de sueldo de un empleado

```
CREATE OR REPLACE TRIGGER GuardarHistorialSalario
BEFORE UPDATE ON Empleado
FOR EACH ROW
 begin
  if (:OLD.Salario <> :NEW.salario)
   then INSERT INTO HistorialSalario VALUES
    (:OLD.codEmp, :OLD.salario, sysdate);
 end if;
 end GuardarHistorialSalario;
/
```


Otros ejemplos de disparadores (3)

Problemas con la actualización de vistas (1)

```
CREATE TABLE Persona(
  nombre VARCHAR2(20),
  descripcion VARCHAR2(20),
  profesion VARCHAR(15),
  constraint pk_persona PRIMARY KEY(nombre)
);

CREATE VIEW Estudiante AS
  (SELECT nombre, descripcion FROM Persona WHERE
   profesion='Estudiante');
```


Otros ejemplos de disparadores (4)

Problemas con la actualización de vistas (2)

```
INSERT INTO Estudiante VALUES('Juan', 'Muy trabajador...');
```

Curiosamente, si ahora consultamos los estudiantes:

```
SELECT * FROM Estudiante;
```

no sale Juan, ya que se adoptó un valor *NULL* para *"profesion"* al insertarlo en *Persona*

Otros ejemplos de disparadores (5)

Problemas con la actualización de vistas (3): Solución

```
CREATE OR REPLACE TRIGGER triggerEstudiantes
INSTEAD OF INSERT ON Estudiante
FOR EACH ROW
BEGIN
  INSERT INTO Persona VALUES(:NEW.nombre,
   :NEW.descripcion, 'Estudiante');
END;
/
```

La actualización de algunas vistas sólo es posible mediante *triggers INSTEAD OF*