

### Vistas

Una vista es una relación o "tabla" derivada de otras relaciones, ya sean "básicas" o también derivadas

- ☛ es transparente para el usuario

- ☛ se genera a partir de una **consulta relacional**

- ☛ son actualizables, pero no deben contener subselects, joins o agregaciones. Si la vista es compleja es posible que existan varias maneras de llegar a la misma conclusión.

WITH	Limita los UPDATES que pueden hacerse y que
CHECK	afectan a las vistas creadas.
OPTION	

### FUNCIONES

- ☛ Los procedimientos almacenados son funciones definidas que se almacenan en la Base de datos. Se tratan como un objeto más de la BD

- » Simplifican el desarrollo de aplicaciones

- » Mejoran el rendimiento de la BD

- » Controlan las operaciones que se realizan

### Disparadores (TRIGGERS)

- ☛ Proveen al DBMS con un recurso para participar de forma más activa en los procesos. Es capaz de reaccionar ante eventos.

- ☛ es muy importante la documentación de los TRIGGERS

- » Monitorización/ alerta ante situaciones adversas (presupuesto inferior a n)

- » Comprobación de restricciones de integridad. (no departamentos de más de n empleados)

- » Mantenimiento automático: atributos derivados

- ☛ El trigger se invoca cuando una operación altera o es susceptible de aleterar el estado correcto del sistema

### TRANSACCIONES y CONCURRENCIA

- ☛ Un DBMS permite el acceso simultaneo de diferentes usuarios a la misma base de datos, preservando su integridad y consistencia

Una transacción es un conjunto de operaciones de lectura y/o modificación de la base de datos que deben realizarse como una sola operación, y se debe acabar confirmando (COMMIT) o cancelando (ROLLBACK).

Toda transacción debe cumplir las propiedades ACID:	Atomicidad, Consistencia, Aislamiento, Definitividad
---	--

- ☑ ATOMICIDAD Las operaciones se tienen que ejecutar completamente o en absoluto. TODAS O NINGUNA

- ☑ CONSISTENCIA Las transacciones deben cumplir los requisitos de los usuarios

- ☑ ISOLATION El acceso simultáneo de dos usuarios no debe producir resultados anómalos

- ☑ DEFINITIVIDAD Han de existir mecanismos para evitar pérdida de datos, tanto antiguos como recientes

### Transacciones: INTERFERENCIAS

- ☹ Actualización perdida Cuando se pierde un cambio realizado por una operación de escritura. READ UNCOMMITTED, READ COMMITTED, REPETEABLE READ, SERIALIZABLE

- ☹ Lectura no confirmada Se lee un dato resultante de una modificación realizada por una transacción que aborta. READ COMMITTED, REPETEABLE READ, SERIALIZABLE

- ☹ Lectura no repetible REPETEABLE READ, SERIALIZABLE



### Transacciones: INTERFERENCIAS (cont)

⊕ Análisis Interferencia respecto a la visión que varias transacciones tienen del conjunto de datos. **REPETEABLE** **READ**, **SERIALIZABLE**

⊕ Fantasma Una transacción lee un conjunto de datos mientras otra añade nuevos elementos al conjunto. **SERIALIZABLE**

Estas interferencias se solucionan con las transacciones. Las transacciones sólo serán necesarias para evitar las interferencias a toda costa. No es recomendable usarlas siempre ya que puede perjudicar el rendimiento del sistema. Existen diversos niveles de posibles transacciones para equilibrar seguridad y rendimiento. **SERIALIZABLE** protege de cualquier interferencia pero imposibilita paralelizar. Esto se llama horario serial.

```
SELECT ID, nombre, sueldo, proyecto
FROM empleados
WHERE proyecto IN ('IoT2', 'UDT4');
```

↓

```
CREATE VIEW EMPLEA DOS _IO T_UDT AS
(SELECT ID, nombre, sueldo, proyecto
FROM empleados
WHERE proyecto IN ('IoT2', 'UDT4'));
```

```
CREATE FUNCTION encontrar_peliculas RETURN TABLE
(tconst char(10))
AS $conte nga _pal$
BEGIN
    RETURN QUERY
        SELECT t.tconst
        FROM title_basics t
        WHERE t.primary title
        LIKE CONCAT ('%', palabra, '%');
END; $conte nga _pal$
LANGUAGE plpgsql;
SELECT * FROM encontrar_peliculas ('AVATAR');
```

### Ejemplos TRIGGERS

**BEFORE & FOR EACH STATEMENT:** Se ejecuta una sola vez, antes de la ejecución de la sentencia que dispara el trigger

**BEFORE & FOR EACH ROW** se ejecuta una sola vez por cada tupla afectada, y justo antes de que la fila se inserte, modifique o borre

**AFTER & FOR EACH ROW** Se ejecuta una sola vez por cada fila afectada, y justo después de la ejecución de la sentencia que dispara el trigger

**AFTER & FOR EACH STATEMENT** se ejecuta una sola vez después de la ejecución de la sentencia que dispara el trigger

**Variables accesibles dentro de la función que ejecuta el trigger:**

**TG\_OP:** contiene qué provocó el salto del disparador: **INSERT**, **DELETE** o **UPDATE**

**NEW:** Valores de la tupla después del **UPDATE** o **INSERT**. No se define para **DELETE**.

**OLD:** valores de la tupla antes de la ejecución del **UPDATE** o **DELETE**. No se define para **INSERT**.

```
CREATE TRIGGER nombre_trigger
{BEFORE || AFTER}
{INSERT || DELETE || UPDATE}
ON
nombre_relacion
FOR EACH {ROW || STATEMENT}
EXECUTE PROCEDURE nombreFunción(argumentos);
```

```
CREATE FUNCTION nombreFunción()
RETURNS trigger AS $$
BEGIN
[...]
```

**RETURN {NULL || NEW || OLD};**

```
END;
$$ LANGUAGE plpgsql;
```



```
-  
UPDATE productos  
SET stock = stock + 10  
WHERE ID = '1123S'  
OLD.ID = '1 123S'  
OLD.stock = #stock  
NEW.ID = '1 123S'  
NEW.stock = #stock + 10
```

```
-  
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
    CREATE TABLE tabla_nueva (ID INTEGER  
PRIMARY KEY);  
COMMIT;
```



By **asourv**

[cheatography.com/asourv/](https://cheatography.com/asourv/)

Published 2nd February, 2023.

Last updated 2nd February, 2023.

Page 3 of 3.

Sponsored by **ApolloPad.com**

Everyone has a novel in them. Finish Yours!

<https://apollopad.com>