INSTITUTO TECNOLÓGICO DE TEPIC

TALLER DE BASES DE DATOS

UNIDAD 5

INTRODUCCIONA AL LENGUAJE PROCEDURAL

COMPETENCIA

Implementar el lenguajes procedural para automatizar las reglas de negocios y garantizar la integridad , consistencia e integridad de los datos mediante el uso de procedimientos almacenados y disparadores

Participación y tareas (15%)
Memoria de Prácticas (Rúbrica 35%)
Examen práctico (50%).

D.E. Verónica Ramírez Jáuregui

PROCEDIMIENTOS ALMACENADOS (SP)

SP

TIPOD

CREATE PROCEDURE

SENTENCIAS VALIDAS

DROP PROCEDURE

EXECUTE

VARIABLES

PARAMETROS

- Un procedimiento almacenado es un conjunto de sentencias SQL y de control de flujo (SP- Store Procedure)
- Beneficios de los procedimientos almacenados:
  - Simplifican la ejecución de tareas repetitivas
  - Corren más rápido que las mismas instrucciones ejecutadas en forma interactiva
  - Reducen el tráfico a través de la red
  - Pueden capturar errores antes que ellos puedan entrar a la base de datos
  - Establece consistencia porque ejecuta las tareas de la misma forma
  - Permite el desarrollo modular de aplicaciones
  - Ayuda a proveer seguridad
  - Puede forzar reglas y defaults complejos de los negocios

TIPOS DE SP

- **Procedimientos almacenados definidos por el usuario**
  - Son procedimientos definidos por el usuario que se deben llamar explícitamente.

- **Triggers**
  - Son procedimientos definidos por el usuario que se ejecutan automáticamente cuando se modifica un dato en una tabla.

- **Procedimientos del sistema**
  - Procedimientos suministrados por el sistema.

- **Procedimientos extendidos**
  - Procedimientos que hacen llamadas al sistema operativo y ejecutan tareas a ese nivel.

TIPOS DE SP

Sintaxis simplificada para create:

```
create procedure  NOMBRE_PROCED
        as
                COMANDOS.....
        return
```

Crear un procedimiento almacenado sencillo:

```
CREATE PROC proc_holamundo
AS
    PRINT "Hola Mundo!!!!"
RETURN
```

Otro Ejemplo:

```
CREATE PROC sp_actualiza_titulos
AS
   UPDATE titles
   SET price = price * $0.95
   WHERE total_sales < 3000
RETURN
```

SENTENCIAS VÁLIDAS E INVÁLIDAS

- Un procedimiento almacenado puede:
  - ✔ **Seleccionar y modificar datos**
  - ✔ **Crear tablas temporales y permanentes**
  - ✔ **Llamar otros procedimientos almacenados**
  - ✔ **Referenciar objetos de bases de datos**
- Un procedimiento almacenado no puede ejecutar:
  - ✘ **use *database***
  - ✘ **create view**
  - ✘ **create default**
  - ✘ **create rule**
  - ✘ **create procedure**
  - ✘ **create trigger**

ELIMINANDO SP

Sintaxis simplificada para drop

drop procedure *NOMBRE_PROCED*

Ejemplo:

drop procedure *SP_ACTUALIZA_TITULOS*

Ver el código fuente de un procedimiento:

EXEC sp_helptext proc_hello

EJECUTAR SP

Sintaxis simplificada para drop

```
execute NOMBRE_PROCED
exec  NOMBRE_PROCED
```

Ejemplo:

```
exec SP_ACTUALIZA_TITULOS
```

VARIABLES

Los procedimientos almacenados pueden crear y usar variables locales:

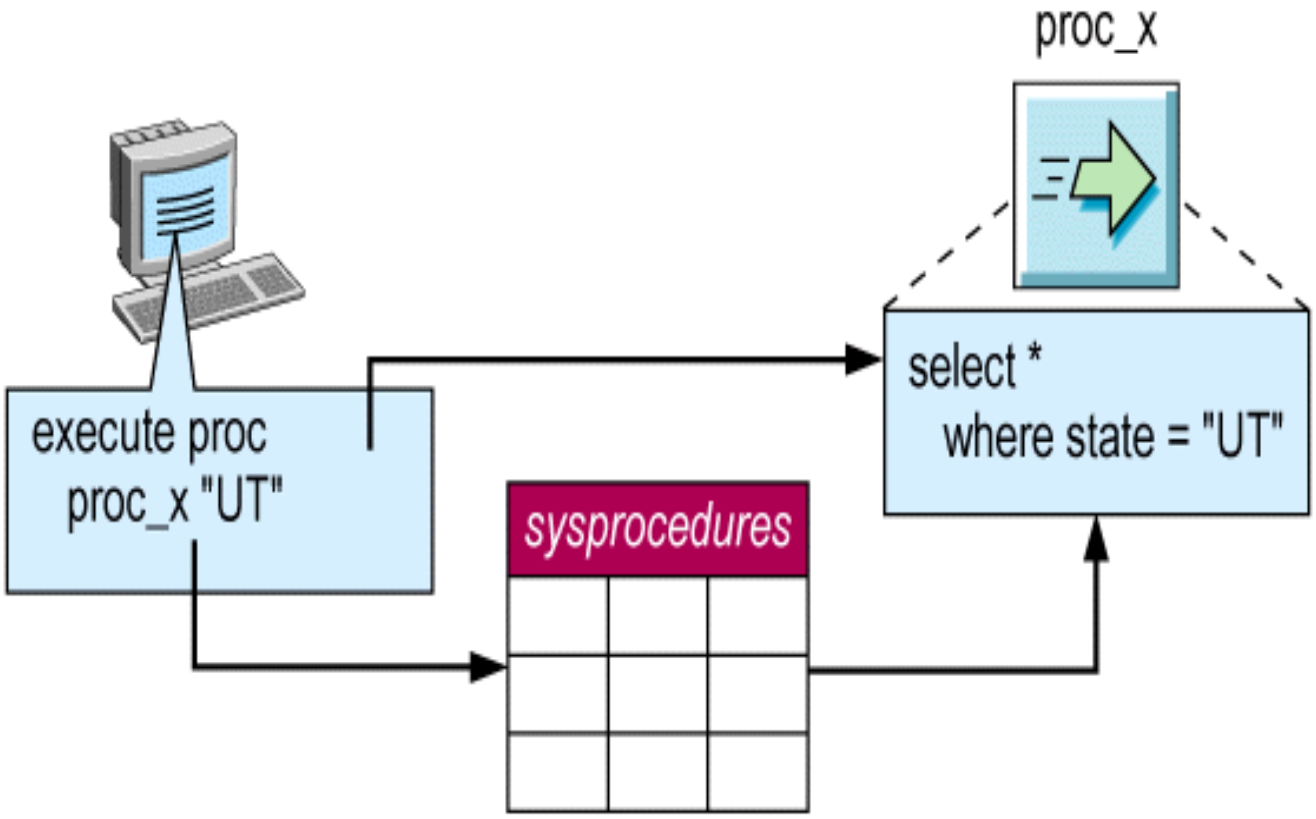
- Las variables sólo existen mientras exista el procedimiento
- Las variables no las puede usar otro proceso

Ejemplo:

```
CREATE PROC sp_act_maxtit
AS
    DECLARE @max_venta int, @mitad_max real

    SELECT @max_venta = MAX(total_sales)
    FROM titles

    SET @mitad_max = @max_venta / 2

    SELECT title, total_sales   FROM titles
    WHERE total_sales < @mitad_max

    UPDATE titles  SET price = price * $0.95
    WHERE total_sales < @mitad_max
RETURN
```

PARÁMETROS DE ENTRADA

Un parámetro de entrada es una variable local para un procedimiento almacenado que puede recibir un valor para su ejecución



PARÁMETROS DE ENTRADA

Definir parámetros de entrada

Sintaxis:

```
CREATE PROC NOMBRE_PROCED (param1  tipo_dato  default,
      param2  tipo_dato  default,.....
      paramN  tipo_dato  default)
AS
      sentencias

RETURN
```

Ejemplo:

```
CREATE PROC sp_info_autor (@apellido varchar(40),
                           @nombre varchar(20) )
AS
    -- LISTA DE  LIBROS DEL AUTOR

    SELECT au_lname, au_fname, title
    FROM  titleauthor ta
    INNER JOIN authors a ON (a.au_id = ta.au_id)
    INNER JOIN  titles t ON (t.title_id = ta.title_id)
    WHERE au_fname =  @nombre
          and au_lname = @apellido
RETURN
```

PARÁMETROS DE ENTRADA

Paso de parámetros por posición

Sintaxis para paso por posición:

EXEC *NOMBRE_PROCED* ***value1, value2.... valueN***

Ejemplo:

EXEC sp_info_autor "Ringer", "Albert"

- Los parámetros se deben pasar en el mismo orden en que ellos aparecen en la sentencia **create procedure**
- Como este método es más propenso a errores, se aconseja el paso por nombre

PARÁMETROS DE ENTRADA

Paso de parámetros por nombre

Sintaxis para paso por posición:

EXEC *NOMBRE_PROCED* ***parametro_nomb1 = value1, parametro_nomb2 = value2.... Parametro_nombN valueN***

Ejemplo:

EXEC sp_info_autor @apellido ="Ringer", @nombe = "Albert"

- Los nombres de los parámetros en la sentencia **exec** deben concordar con los nombres de los parámetros usados en la sentencia **create procedure**
- Los parámetros pueden pasar en cualquier orden

PARÁMETROS DE ENTRADA

Valores por default

- Se puede asignar un valor por default a un parámetro cuando él no se indica en la sentencia **exec**

Ejemplo:

```
CREATE PROC  sp_estado_autor (@state char(2) = "CA")
AS
   SELECT au_lname, au_fname, state
   FROM authors
    WHERE state = @state
RETURN


EXEC sp_estado_autor              -- NO SE PASA EL VALOR
EXEC sp_estado autor    “UT”
```

IF -ELSE

IF - ELSE

Impone condiciones en la ejecución de una instrucción Transact-SQL. La instrucción Transact-SQL que sigue a una palabra clave IF y a su condición se ejecuta si la condición se cumple: la expresión booleana devuelve TRUE. La palabra clave opcional ELSE introduce otra instrucción Transact-SQL que se ejecuta cuando la condición IF no se cumple: la expresión booleana devuelve FALSE.

```
IF Boolean_expression
 BEGIN
   SETENCIAS SQL....
 END

ELSE
 BEGIN
  SETENCIAS SQL....
 END
```

DISPARADORES (TRIGGERS)

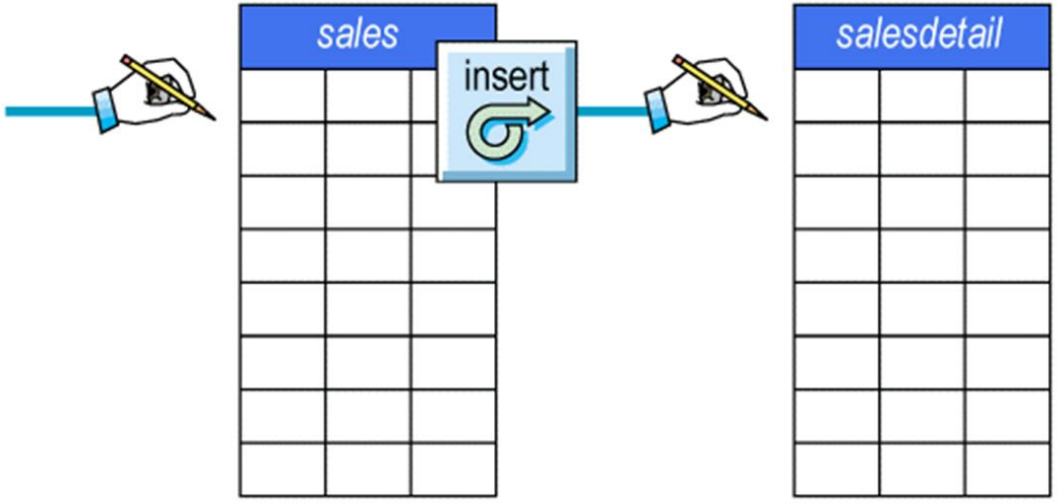
Un trigger es un procedimiento almacenado asociado con una tabla, el cual se ejecuta automáticamente cuando se modifica un dato de esa tabla

Aplicaciones Típicas de triggers

- Hacer modificaciones en cascada sobre tablas relacionadas
- Deshacer cambios que violan la integridad de los datos
- Forzar restricciones que son muy complejas para reglas y restricciones
- Mantener datos duplicados
- Mantener columnas con datos derivados
- Hacer ajustes de registros
- Bitácoras de Eventos

DISPARADORES (TRIGGERS)

- Un trigger se define asociado con una tabla para una o más sentencias de manipulación de datos. (insert, update, delete).
- Cuando se modifica un dato en una tabla que tiene declarado un trigger para esa sentencia, el trigger se "dispara"
  - El trigger se dispara una vez, independientemente del número de filas afectadas
  - El trigger se dispara aunque no hayan filas afectadas



REGLAS DE LOS TRIGGERS

- Los triggers pueden:
  - Declarar variables locales
  - Invocar procedimientos almacenados
- Los triggers no pueden:
  - Llamarse directamente
  - Usar parámetros
  - Definirse sobre tablas temporales o vistas
  - Crear objetos permanentes de base de datos
- Las operaciones con registro mínimo (como **select into**) no disparan los triggers

CREAR TRIGGERS

Sintaxis

```
CREATE TRIGGER  nomb_ trigger
ON  nomb_tabla
FOR {insert | update | delete} [, {insert | update | delete} ...]
AS
        SETENCIAS.....
```

Ejemplo

```
CREATE TRIGGER trg_i_sales
ON  sales
FOR  INSERT
AS
 IF datename (dd,getdate()) = 'Sun'
   BEGIN
   raiserror ('La Venta no puede ser procesada en
                Domingo',1,1)
 END
```

BORRAR TRIGGERS

Sintaxis

drop trigger *nomb_trigger*

Ejemplo

drop trigger *trg_i_sales*

Procedimientos almacenados:

- **sp_depends** {*table_name* | *trigger_name*}
  - Cuando se da el nombre de tabla, lista todos los objetos (incluyendo triggers) de la misma base de dtos
  - Cuando se da el nombre de trigger, lista todas las tablas referencias
- **sp_help** *trigger_name*
  - Muestra información del trigger
- **sp_helptext** *trigger_name*
  - Muestra el código usado para crear el trigger
- **sp_rename** *old_trigger_name*, *new_trigger_name*
  - Cambia el nombre del trigger

EJEMPLO

Crear DOS tablas:

```
SELECT * INTO myauthors  from pubs2..authors
CREATE TABLE miRecord
(
   miFecha datetime,
   miRegisros int
)
```

Crear un trigger que guarde la fecha y número de filas afectadas por cada **delete**:

```
CREATE TRIGGER trg_d_myauthors
ON myauthors
FOR delete
AS
        INSERT INTO myrecord
        VALUES (getdate(), @@rowcount)
RETURN
```

EJEMPLO

- Ejecutar un **delete** y ver la tabla *myrecords*:

```
DELETE FROM myauthors  WHERE state = "CA"
SELECT * FROM miRecord
```

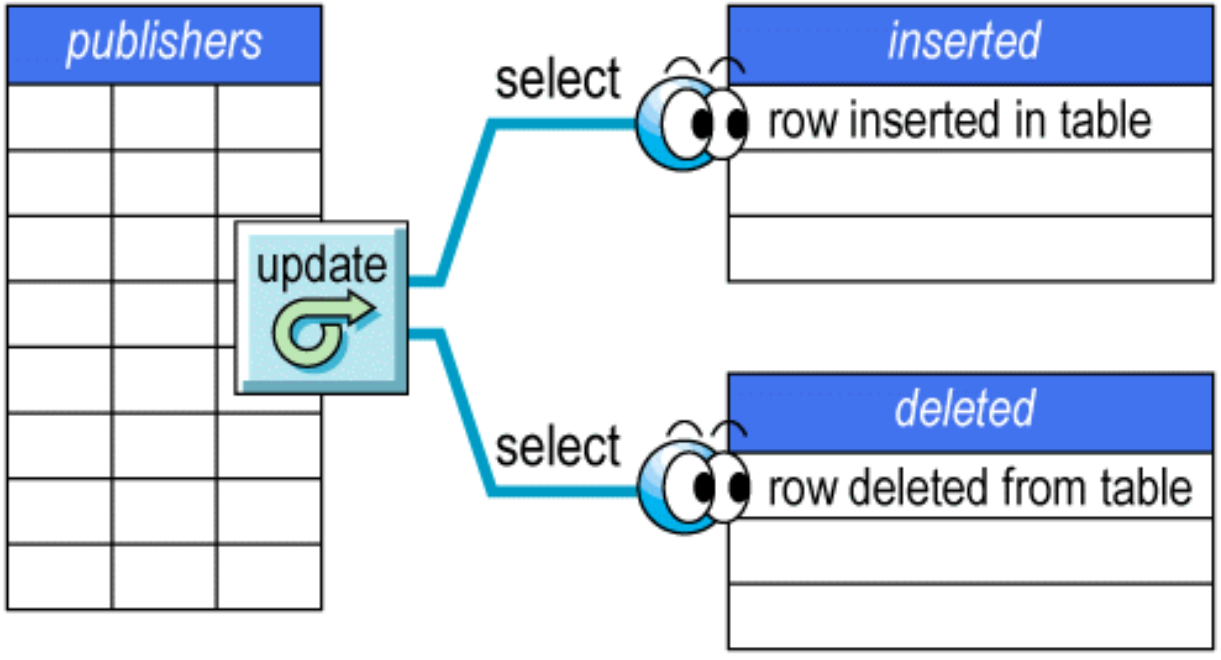
- Ejecutar un **delete** que no afecta filas y ver la tabla *miRecords* :

```
DELETE FROM myauthors WHERE 1 = 2
SELECT * FROM miRecord
```

EJEMPLO

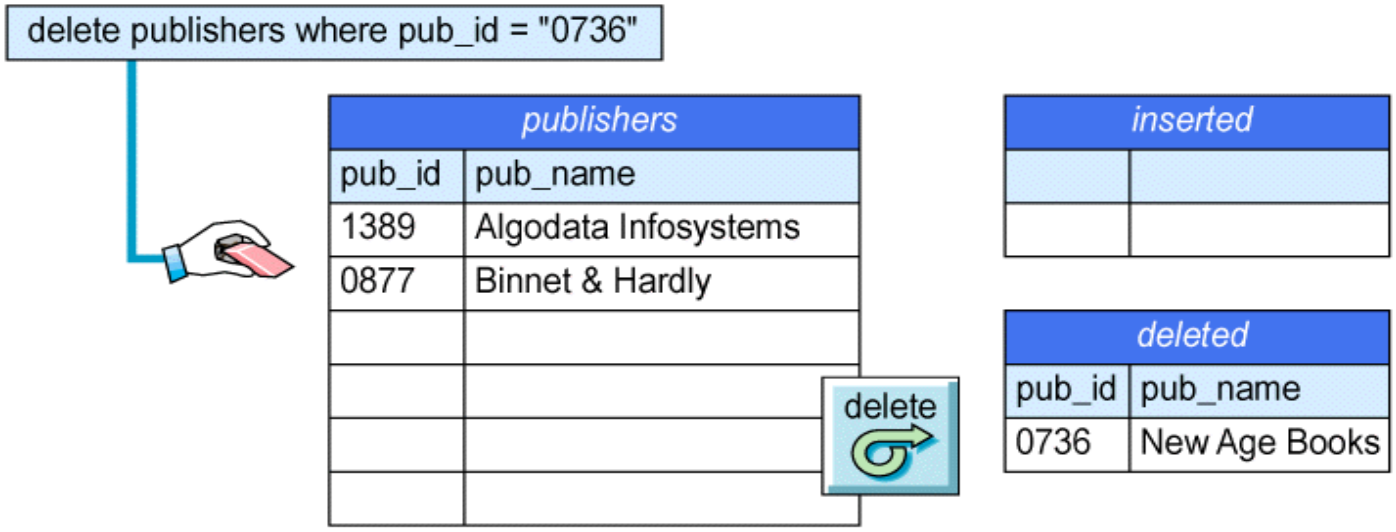
INSERTED y ***DELETED*** son dos tablas que se crean automáticamente cada vez que se dispara un trigger.

- ***inserted*** almacena cualquier fila que se vaya a añadir a la tabla
- ***deleted*** almacena cualquier fila que se vaya a borrar de la tabla



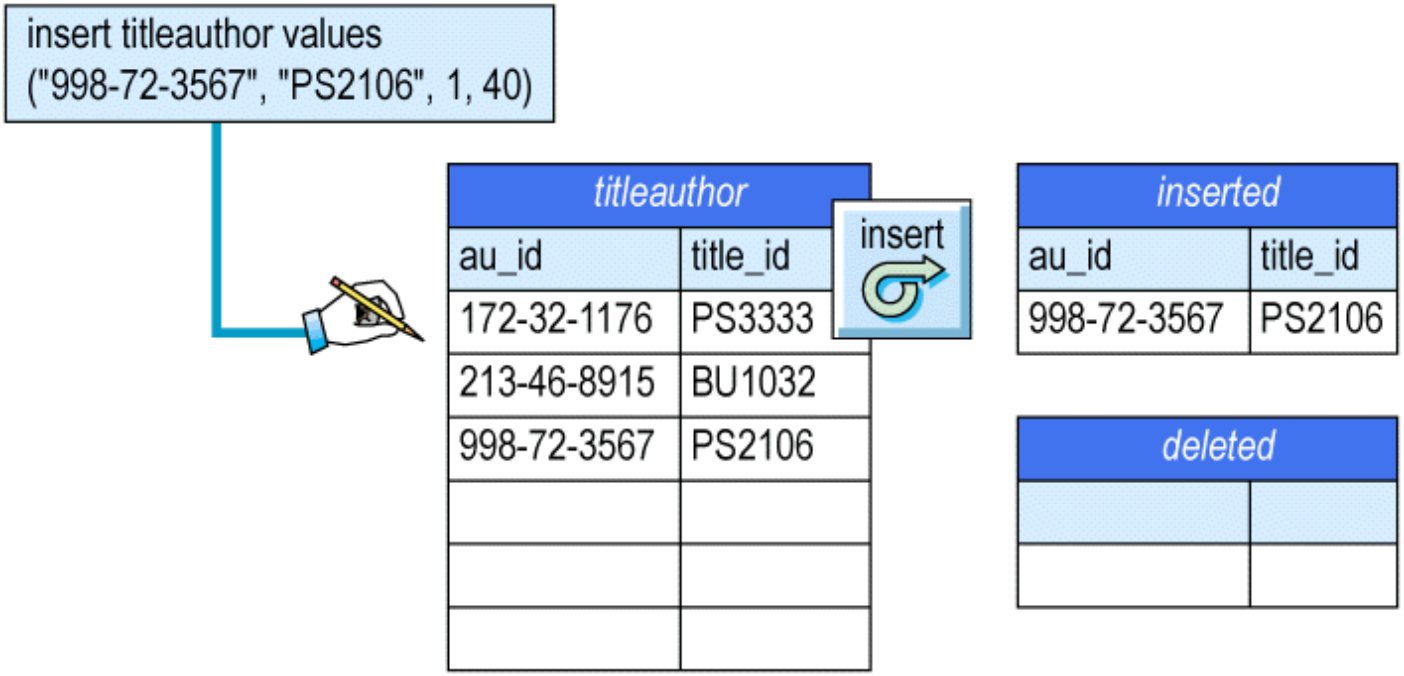
EJEMPLO

Borrados



EJEMPLO

Inserciones



EJEMPLO

Actualizaciones

```
update publishers set pub_id = "9988"
from publishers where pub_id = "9999"
```

publishers

pub_id	pub_name
1389	Algodata Infosystems
0877	Binnet & Hardly
9988	Tech Books

update

inserted

pub_id	pub_name
9988	Tech Books

deleted

pub_id	pub_name
9999	Tech Books

REGLAS PARA LAS TABLAS INSERTED Y DELETED

- Ambas tablas tienen las mismas columnas que la tabla asociada al trigger.
- El trigger puede consultar datos de las dos tablas
  - Otros procesos no pueden consultar datos de las dos tablas
- El trigger no puede modificar datos en las dos tablas
- Cada anidamiento de triggers tiene sus propias tablas ***inserted*** y ***deleted***
  - Si un trigger modifica datos de su tabla asociada, esos cambios no se reflejan en las tablas *inserted* and *deleted* de ese trigger

EJEMPLO

TRIGGER

REGLAS TRIGGER

CREATE TRIGGERS

DROP Y SP

EJEMPLO

INSERTED Y DELETED

REGLAS INSERTED Y DELETED