

# Functions, Procedures, Triggers, and Views in SQL

Base de Datos 2022

# Funciones y Procedimientos

- Un store procedure/function es un conjunto de sentencias SQL con un nombre asignado y puede ser reutilizado cuantas veces se desee.
- Generalización de SQL añadiendo la estructura de un lenguaje de programación al lenguaje SQL

Estructura típicamente disponibles en procedimientos almacenados:

- Variables Locales
- Sentencias IF
- Sentencias LOOP

# Funciones

```
create function dept_count(dept_name varchar(20))  
  returns integer  
  begin  
    declare d_count integer;  
    select count(*) into d_count  
    from instructor  
    where instructor.dept_name= dept_name  
  return d_count;  
  end
```

```
select dept_name, budget  
from instructor  
where dept_count(dept_name) > 12;
```

# Procedimientos

```
create procedure dept_count_proc(in dept_name varchar(20),  
                                out d_count integer)  
begin  
    select count(*) into d_count  
    from instructor  
    where instructor.dept_name= dept_count_proc.dept_name  
end  
  
declare d_count integer;  
call dept_count_proc('Physics', d_count);
```

# Procedimientos

Modos de Pasajes de Parámetros:

- IN: parámetro pasado por valor
- OUT: parámetro pasado por referencia
- INOUT: parámetro pasado por referencia y puede ser modificado pero se asume que ha sido inicializado

# Sentencias **While** y **Repeat**

```
while boolean expression do  
    sequence of statements;  
end while
```

```
repeat  
    sequence of statements;  
until boolean expression  
end repeat
```

## For loop...

```
declare n integer default 0;  
for r as  
    select budget from department  
    where dept_name = 'Music'  
do  
    set n = n - r.budget  
end for
```

# Sentencia if-then-else

```
if boolean expression  
    then statement or compound statement  
elseif boolean expression  
    then statement or compound statement  
else statement or compound statement  
end if
```



## Ejemplo: registrar un estudiante en un curso

```
create function registerStudent(  
    in s_id varchar(5),  
    in s_courseid varchar (8),  
    in s_secid varchar (8),  
    in s_semester varchar (6),  
    in s_year numeric (4,0),  
    out errorMsg varchar(100)  
returns integer  
begin  
    declare currEnrol int;  
    select count(*) into currEnrol  
        from takes  
        where course_id = s_courseid and sec_id = s_secid  
            and semester = s_semester and year = s_year;  
    declare limit int;  
    select capacity into limit  
        from classroom natural join section  
        where course_id = s_courseid and sec_id = s_secid  
            and semester = s_semester and year = s_year;
```

## Ejemplo: registrar un estudiante en un curso...

```
if (currEnrol < limit)
    begin
        insert into takes values
            (s_id, s_courseid, s_secid, s_semester, s_year, null);
        return(0);
    end
-- Otherwise, section capacity limit already reached
set errorMsg = 'Enrollment limit reached for course ' || s_courseid
    || ' section ' || s_secid;
return(-1);
end;
```

# Triggers...

- Un disparador o **trigger** es una funcionalidad que la base de datos ejecuta de forma automática cuando se realiza una operación de tipo **INSERT**, **UPDATE**, o **DELETE** que impacta sobre los registros en la tabla asociada.
- Se puede configurar un trigger para que se active o bien antes (**BEFORE**) o después (**AFTER**) del evento del trigger

# Triggers (Sintaxis)

**CREATE TRIGGER** trigger\_name **trigger\_time** trigger\_event

**ON** table\_name **FOR EACH ROW**

**BEGIN**

**[trigger\_order]**

trigger\_body

**END;**

**trigger\_time:** {BEFORE | AFTER}

**trigger\_event:** {INSERT | UPDATE | DELETE}

**trigger\_order:** {FOLLOWS| PRECEDES} other\_trigger\_name

## Ejemplo

```
create trigger credits_earned after update of takes on (grade)  
referencing new row as nrow  
referencing old row as orow  
for each row  
when nrow.grade <> 'F' and nrow.grade is not null  
  and (orow.grade = 'F' or orow.grade is null)  
begin atomic  
  update student  
  set tot_cred = tot_cred +  
    (select credits  
     from course  
     where course.course_id = nrow.course_id)  
  where student.id = nrow.id;  
end;
```

# Views

- Las relaciones de vista se pueden definir como relaciones que contienen el resultado de consultas.
- Las vistas son útiles para ocultar información innecesaria y para recopilar información de más de una relación en una sola vista.
- Cuando se define una vista, normalmente la base de datos almacena solo la consulta que define la vista.

# Views

Definición:

```
CREATE VIEW v AS <query expression>;
```

# Ejemplo

```
create view faculty as  
select ID, name, dept_name  
from instructor;
```



## Otro Ejemplo

```
create view physics_fall_2009 as  
  select course.course_id, sec.id, building, room.number  
  from course, section  
  where course.course_id = section.course_id  
        and course.dept_name = 'Physics'  
        and section.semester = 'Fall'  
        and section.year = '2009';
```

```
select course_id  
from physics_fall_2009  
where building = 'Watson';
```

# Materialized Views

- Ciertos sistemas de bases de datos permiten almacenar las relaciones de vista, pero se aseguran de que, si las relaciones reales utilizadas en la definición de la vista cambian, la vista se mantenga actualizada.
- Cuando se define una vista, normalmente la base de datos almacena solo la consulta que define la vista.
- Por el contrario, una vista materializada es una vista cuyos contenidos se calculan y almacenan.

# Materialized Views

Definición:

```
CREATE MATERIALIZED VIEW v AS <query expression>;
```

# Materialized Views

- Las vistas materializadas constituyen datos redundantes
- Sin embargo, en muchos casos es mucho más barato leer el contenido de una vista materializada que calcular el contenido de la vista ejecutando la consulta que define la vista.

# Materialized Views

Las vistas materializadas son importantes para mejorar el rendimiento en algunas aplicaciones.

```
create view department_total_salary(dept_name, total_salary) as  
select dept_name, sum (salary)  
from instructor  
group by dept_name;
```

# Materialized Views

- Suponga que el monto de salario total en un departamento se requiere con frecuencia.
- Calcular la vista requiere leer cada tupla de instructor perteneciente a un departamento y resumir los montos de los salarios, lo que puede llevar mucho tiempo.
- Por el contrario, si la definición de la vista del monto del salario total se **materializara**, el monto del salario total podría encontrarse buscando una tupla en la vista materializada