Sesión 12
Características Avanzadas

Laboratorio de Bases de Datos I

Josefa Gómez
Hamid Tayebi
Daniel Rodríguez
Ignacio Olmeda
Iván González Diego
Dept. Ciencias de la Computación
Universidad de Alcalá

Creación de Secuencias

- *Secuencia: 1,2,3,4,5, ...*
- *Se pueden asignar a los campos de las tablas:*

```
CREATE SEQUENCE tablename_colname_seq;
CREATE TABLE tablename ( colname integer DEFAULT
                nextval('tablename_colname_seq') NOT NULL );
```

- *Es equivalente a:*

```
CREATE TABLE tablename ( colname SERIAL );
```

- *Funciones asociadas: nextval('sequencia')*
  currval('sequencia')
  lastval()

Ejemplo de Secuencias

Create sequence codigo_cliente_seq;

CREATE TABLE cliente (codigo_cliente integer DEFAULT nextval('codigo_cliente_seq') NOT NULL PRIMARY KEY, nombre text NOT NULL, Apellidos text NOT NULL);

Insertar el cliente cuyo nombre es "Angel" y Apellidos "Gutierrez Herrero" sin introducir el campo codigo_cliente

Insertar el cliente cuyo nombre es "Oscar" y Apellidos "Gutierrez Blanco" sin introducir el campo codigo_cliente

PL/pgSQL

- *Lenguaje procedimental para ser utilizado en el servidor:*
  - *Mayor rendimiento*
  - *Soporte SQL*
  - *Portabilidad ⇒ Reusar código en el servidor Postgres*
  - *Crear funciones y disparadores (triggers)*
  - *Tipos creados por el usuario, funciones y operadores*
  - *Añadir estructuras de control al lenguaje SQL*
  - *Poder realizar computaciones complejas*
  - *Sencillo de utilizar*
- *Estructura: Lenguaje estructurado por bloques*

```
[ <<etiqueta>>
   ]
[ DECLARE
declaraciones
   ]
BEGIN
estamentos
```

PL/pgSQL - Funciones

```
CREATE FUNCTION algunafuncion() RETURNS INTEGER AS '
DECLARE
cantidad INTEGER := 30;
BEGIN
RAISE NOTICE ''La cantidad aquí es %'',cantidad; --
   La cantidad aquí es 30
cantidad := 50;
DECLARE
cantidad INTEGER := 80;
BEGIN
RAISE NOTICE ''La cantidad aquí es %'',cantidad; --
   La cantidad aquí es 80
END;
RAISE NOTICE ''La cantidad aquí es %'',cantidad; --
   La cantidad aquí es 50
RETURN cantidad;
END;
' LANGUAGE 'plpgsql';
```

PL/pgSQL - Declaraciones

- Declarar variables, filas y registros.
- Cualquier tipo de datos SQL (Integer, Varchar, char, etc)

```
user_id INTEGER;
cantidad NUMERIC(5);
url VARCHAR;
myrow nombretabla%ROWTYPE; -- Tipo fila
myfield nombretabla.nombrecampo%TYPE; -- tipo atributos
unafila RECORD; -- Registros sin formato especificado
```

- Asignación de variables:
  - Identificador := expresión. Ejemplo: user_id:=20;
  - SELECT INTO destino expresiones FROM …;
    - Select into my_rec * from emp where

PL/pgSQL – Estructuras de Control

- RETURN expresion; -- Devuelve resultado desde una función.

- Condicionales
  - IF … THEN … ELSE END IF;

- Bucles
  - LOOP ….. END LOOP;
  - EXIT , para salir del bucle si cumple condición.
  - WHILE expresion LOOP ….. END LOOP;
  - FOR nombre IN expresion LOOP … END LOOP (Solo Integer)
  - FOR registro | fila IN select_query LOOP .. END LOOP (Bucles a través de los resultados de una consulta)

PL/pgSQL – Cursores

- Para evitar sobrecargas de memoria, se pueden definir cursores.
- Se leen unos cuantos resultados a la vez y no todos.
- Declaración:

```
DECLARE
curs1 refcursor;
curs2 CURSOR FOR SELECT * from tenk1;
curs3 CURSOR (key int) IS SELECT * from tenk1 where
unique1 = key;
```

- Abrir cursores:

```
OPEN curs1 FOR SELECT * FROM foo WHERE key = mykey;
        OPEN curs2;
        OPEN curs3(42);
```

- Usar cursores:

```
FETCH curs1 INTO rowvar;
FETCH curs2 INTO foo,bar,baz;
```

- Cerrar cursores:

```
CLOSE curs1;
```

PL/pgSQL – Ejemplo Cursores

```
CREATE OR REPLACE FUNCTION cursor() RETURNS void AS '
DECLARE
   curs1 refcursor;
   count integer;
   fila cliente%rowtype;
BEGIN
   OPEN curs1 FOR SELECT * FROM cliente;
   fetch curs1 into fila ;
   raise notice ''codigo :%, Nombre:%,
   Apellidos:%'',fila.codigo_cliente,fila.nombre,fila.apellidos;
   WHILE true LOOP
     fetch curs1 into fila ;
                 if not found then
              exit;
     end if;
     raise notice ''codigo :%, Nombre:%,
   Apellidos:%'',fila.codigo_cliente,
                                      fila.nombre,fila.apellidos;
   END LOOP;
END;
' LANGUAGE 'plpgsql';
```

PL/pgSQL – Errores y Mensajes

- Para mostrar mensajes en pantalla y lanzar errores.
  - RAISE level 'format' [, variable [...]];
    - INFO, NOTICE y WARNING.
    - EXCEPTION , lanza un error y aborta la ejecución.
- Ejemplos:
  - RAISE NOTICE ''Calling cs_create_job(%)'',v_job_id;
  - RAISE EXCEPTION ''Inexistent ID --> %'',user_id;

PL/pgSQL – Disparadores (Triggers)

- *Es un procedimiento que se ejecuta cuando se realiza una operación sobre una tabla (Insert, update o delete*

CREATE TRIGGER name { BEFORE | AFTER } { event [OR ...] } ON table [FOR [EACH] { ROW | STATEMENT }] EXECUTE PROCEDURE funcname (arguments)

DROP TRIGGER name ON table [CASCADE | RESTRICT]

PL/pgSQL – Disparadores (Triggers)

- *Variables asociadas al disparador:*

- **NEW** Tipo de datos RECORD; variable que almacena la nueva fila de base de datos para operaciones INSERT/UPDATE en triggers de nivel ROW.
- **OLD** Tipo de datos RECORD; variable que almacena la antigua fila de base de datos para operaciones UPDATE/DELETE en triggers de nivel ROW.
- **TG_NAME** Tipo de datos *name*; variable que contiene el nombre del trigger actualmente disparado.
- **TG_WHEN** Tipo de datos *text*; una cadena de BEFORE o AFTER dependiendo de la definición del trigger.
- **TG_LEVEL** Tipo de datos *text*; una cadena de ROW o STATEMENT dependiendo de la definición del trigger.
- **TG_OP** Tipo de datos *text*; una cadena de INSERT, UPDATE o DELETE indicando por cuál operación se disparó el trigger.
- **TG_RELID** Tipo de datos *oid*; el ID de objeto de la tabla que causó la invocación del trigger.

PL/pgSQL – Disparadores (Triggers)

Variables asociadas al disparador:

- **TG_RELNAME** Tipo de datos *name*; el nombre de la tabla que causó la invocación del trigger.
- **TG_NARGS** Tipo de datos *integer*; el número de argumentos proporcionado al procedimiento trigger en el estamento CREATE TRIGGER.
- **TG_ARGV[]** Tipo de datos *array* de *text*; los argumentos del estamento CREATE TRIGGER.

PL/pgSQL – Disparadores (Triggers)

- *Ejemplo:*

```
CREATE TABLE emp ( nombre_empleado text, salario integer, ultima_fecha
    timestamp, ultimo_usuario text );

CREATE FUNCTION emp_stamp () RETURNS TRIGGER AS '
BEGIN
-- Comprueba que se proporcionan nombre_empleado y salario
IF NEW.nombre_empleado ISNULL THEN
RAISE EXCEPTION ''El nombre del empleado no puede ser un valor NULO'';
END IF;
IF NEW.salario ISNULL THEN
RAISE EXCEPTION ''% no puede tener un salario NULO'', NEW.nombre_empleado;
END IF;
-- ¿Quién trabaja gratis?
IF NEW.salario < 0 THEN
RAISE EXCEPTION ''% no puede tener un salario negativo'', NEW.nombre_empleado;
END IF;
-- Recuerda quién y cuándo hizo el cambio
NEW.ultima_fecha := ''now'';
NEW.ultimo_usuario := current_user;
RETURN NEW;
END;
' LANGUAGE 'plpgsql';

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp
FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

Gestión de Usuarios / Permisos

- Usuarios: ***CREATE USER*** *name [[WITH] option [...]]*

  where option can be: SYSID uid | [ENCRYPTED | UNENCRYPTED] PASSWORD 'password' | CREATEDB | NOCREATEDB | CREATEUSER | NOCREATEUSER | IN GROUP groupname [, ...] | VALID UNTIL 'abstime'

- Grupos de usuarios:

  CREATE GROUP *name [[WITH] option [...]]*

  where option can be: SYSID gid | USER username [, ...]

- Permisos:

  GRANT *{ { SELECT | INSERT | UPDATE | DELETE | RULE | REFERENCES | TRIGGER } [,...] | ALL [PRIVILEGES] } ON [TABLE] tablename [, ...] TO { username | GROUP groupname | PUBLIC } [, ...] [WITH GRANT OPTION]*

  REVOKE *[GRANT OPTION FOR] { { SELECT | INSERT | UPDATE | DELETE | RULE | REFERENCES | TRIGGER } [,...] | ALL [PRIVILEGES] } ON [TABLE] tablename [, ...] FROM { username | GROUP groupname | PUBLIC } [, ...] [CASCADE | RESTRICT]*

Vistas / Indices / Catálogo del Sistema

- **Vistas**

CREATE [OR REPLACE] VIEW name [(column_name [, ...])] AS
query

- **Indices**

CREATE [UNIQUE] INDEX name ON table [USING method] ({ column | (expression) } [opclass] [, ...]) [TABLESPACE tablespace] [WHERE predicate]

- Hay tablas del sistema donde se guarda información referente al sistema, bases de datos, etc ⇒ **Catálogo del Sistema**

Ejemplo:

*Select * from pg_database;*

Herencia

- *PostgreSQL permite heredar atributos de otros tablas (INHERITS)*

Ejemplo:

```
CREATE TABLE cities ( name text, population float, altitude int -- (in ft)
);

CREATE TABLE capitals ( state char(2) ) INHERITS (cities);
```

- *Tabla capitals tiene 4 columnas: name, population, altitude y state*