Clase 4 - sql

Transacciones

Bloque de código que enlaza las operaciones que están dentro del él. La principal ventaja es la posibilidad de hacer “rollback” cuando alguna de las instrucciones internas del bloque falla, de esta forma no se guardan los cambios en la DB.

Ejemplo: respecto del sistema de voluntariado, imaginemos que se quiere hacer la carga masiva de las instituciones y sus respectivas emergencias, entonces cada institución envía un archivo con la información, la transacción vendría dada por la carga de datos en las tablas “institucion” y “emergencia”. El archivo que llega es de la institución “TECHO” y quiere reportar 3 emergencias.

```
begin;
 insert into institucion (id, nombre)  values (0,'TECHO');
 insert into emergencia(id,id_institucion,nombre) values (0,0,'em0');
 insert into emergencia(id,id_institucion,nombre) values (1,0,'em1');
 insert into emergencia(id,id_institucion,nombre) values (2,0,'em2');
commit;
```

Es conveniente realizar los bloques de transacción contemplando la captura de excepciones y realizando rollback si existen fallas.

- Funciona correctamente

```
do $$
begin
 insert into institucion (id, nombre,descrip)  values (0,'TECHO','descrip0');
 insert into emergencia(id,id_institucion,nombre) values (0,0,'em0');
 insert into emergencia(id,id_institucion,nombre) values (1,0,'em1');
 insert into emergencia(id,id_institucion,nombre) values (2,0,'em2');
exception when others then
 rollback;
 raise;
end;
$$ language plpgsql;
```

- Error Capturado (1).

```
do $$
begin
 insert into institucion (id, nombre,descrip)  values (10,'TECHO 10','descrip10');
 insert into emergencia(id,id_institucion,nombre) values (10,10,'em10');
 insert into emergencia(id,id_institucion,nombre) values (11,10,'em11');
 insert into emergencia(id,id_institucion,nombre) values (2,10,'em12');
exception when others then
 rollback;
 raise;
end;
$$ language plpgsql;
```

- Error Capturado (2), se puede establecer un mensaje de error propio.

```
do $$
begin
 insert into institucion (id, nombre,descrip)  values (10,'TECHO 10','descrip10');
 insert into emergencia(id,id_institucion,nombre) values (10,10,'em10');
 insert into emergencia(id,id_institucion,nombre) values (11,10,'em11');
 insert into emergencia(id,id_institucion,nombre) values (12,40,'em12');
exception when others then
 rollback;
 raise notice 'ERROR CAPTURADO EN LA TRANSACCION';
end;
$$ language plpgsql;
```

- Corrección de Error

```
do $$
begin
 insert into institucion (id, nombre,descrip)  values (10,'TECHO 10','descrip10');
 insert into emergencia(id,id_institucion,nombre) values (10,10,'em10');
 insert into emergencia(id,id_institucion,nombre) values (11,10,'em11');
 insert into emergencia(id,id_institucion,nombre) values (12,10,'em12');
exception when others then
 rollback;
 raise;
end;
$$ language plpgsql;
```

- Verificación

```
delete from emergencia;
delete from institucion;
```

```
SELECT id, nombre, descrip FROM public.institucion;
SELECT id,id_institucion, nombre FROM public.emergencia;
```

Procedimientos Almacenados

Es un procedimiento que realiza acciones específicas sobre los datos al momento de ser llamado.

Ejemplo: queremos cambiar el ESTADO de una tarea a CERRADO (en la db corresponde al valor 2), esto siempre y cuando detectemos que la tarea es muy antigua, suponiendo que las tareas antiguas son todas aquellas que son anteriores “2000-01-01”

- primero verifiquemos cuantas cumplen con este criterio

```
select count(*) from tarea
where finicio < '2000-01-01'
and id_estado=2;
```

- luego creamos el procedimiento almacenado (estos se realizan para labores mucho más complejas que la planteada en este ejemplo)

```
CREATE OR REPLACE PROCEDURE cerrar_tareas_antiguas(fcierre date )
LANGUAGE SQL
AS $$
        UPDATE tarea SET id_estado = 2 WHERE finicio < fcierre;
$$;
```

- Realizamos la llamada

```
CALL public.cerrar_tareas_antiguas('2000-01-01');
```

- Nuevamente verificamos, para ver cuales cambiaron de estado

```
select count(*) from tarea
where finicio < '2000-01-01'
and id_estado=2;
```

Funciones

Las funciones permiten crear acciones especificar que trabajan sobre los datos de la db, la ventaja es pueden ser llamada en cualquier momento desde un bloque o instrucción sql, ejemplo: una sentencia de selección.

Ejemplo: crear una función que realice un cálculo aproximado de la edad de una persona basado en el año de nacimiento y el año actual.

```
CREATE FUNCTION calc_edad_aprox(fnacimiento date)
RETURNS integer AS $$
	BEGIN
		RETURN ( EXTRACT(YEAR FROM CURRENT_DATE) - EXTRACT(YEAR FROM fnacimiento));
	END; $$
LANGUAGE PLPGSQL;
```

- Verifiquemos el funcionamiento de la función

```
select id, nombre, fnacimiento,calc_edad_aprox(fnacimiento) edad
from voluntario;
```

Triggers

Es una acción que se desencadena antes o después de un evento sobre una tabla de la base de datos. Para crear un trigger se deben crear 2 elementos, una función que realiza una acción determinada y el trigger que llama la función creada al momento de realizar una acción en una tablas (insert, update, delete)

Ejemplo: pensemos que queremos dejar un log de las acciones realizadas sobre el sistema de voluntariado, y que dicha acción puede ser la creación de una emergencia o una tarea de una emergencia

- primero creamos la tabla log_accion
    - SCRIPT DE CREACIÓN

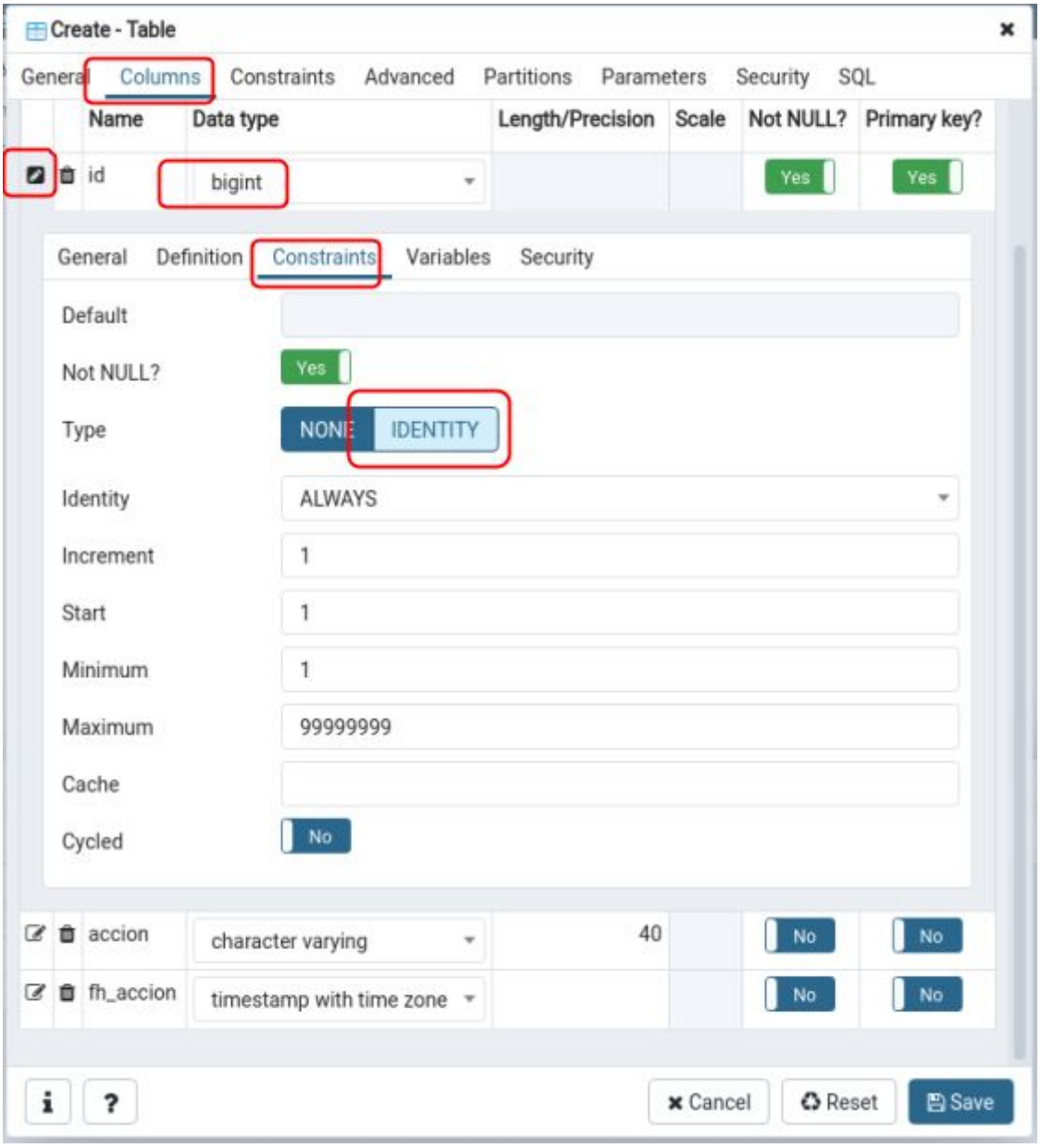
```
-- Table: public.log_accion

-- DROP TABLE public.log_accion;

CREATE TABLE public.log_accion
(
id bigint NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 99999999 CACHE 1 ),
accion character varying(100) COLLATE pg_catalog."default",
fh_accion timestamp with time zone,
CONSTRAINT log_accion_pkey PRIMARY KEY (id)
)
WITH (
OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.log_accion
OWNER to postgres;
```

- La otra opción es por el panel de pgAdmin4



Notar que la tabla tiene un id automático que va de 1 hasta 999999999, esto es preferible cuando se crean tablas tipo log, para solo insertar la información relevante y no preocuparse el id de la tabla.

- Verificamos que se ha creado la tabla (insertemos y luego seleccionemos)

```
INSERT INTO public.log_accion( accion, fh_accion)
        VALUES ('emergencia', CURRENT_TIMESTAMP);

SELECT id, accion, fh_accion
        FROM public.log_accion;
```

Ahora que tenemos la tabla podemos crear las funciones que gatillarán acciones sobre ella, en este caso insertar un registro en la tabla “log_accion“ cuando se inserte un registro en la tabla “emergencia” o en la tabla “tarea”. ()

- Creamos la función que se lanzará cuando se inserte el registro en la tabla emergencia

```
CREATE OR REPLACE FUNCTION insert_emergencia_logaccion()
  RETURNS trigger AS
$$
BEGIN
        INSERT INTO log_accion(accion,fh_accion)
        VALUES('EM-'||NEW.id||'-'||NEW.nombre,CURRENT_TIMESTAMP);

    RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';
```

- Creamos el trigger que gatilla la ejecución acción sobre log_accion luego de insertar un registro en la tabla emergencia

```
CREATE TRIGGER trg_insert_emergencia_log
  AFTER INSERT
  ON emergencia
  FOR EACH ROW
  EXECUTE PROCEDURE insert_emergencia_logaccion();
```

- verifiquemos que funciona correctamente

```
INSERT INTO public.emergencia(
            id, nombre, descrip, finicio, ffin, id_institucion)
            VALUES (501, 'em', 'descrip', '2020-01-01', '2020-01-01', 1);

select * from public.log_accion;
```

- Creamos la función que se lanzará cuando se inserte el registro en la tabla tarea

```
CREATE OR REPLACE FUNCTION insert_tarea_logaccion()
  RETURNS trigger AS
$$
BEGIN
        INSERT INTO log_accion(accion,fh_accion)
    VALUES('EM-'||NEW.id_emergencia||'-TA-'||NEW.id||'-'||NEW.nombre,CURRENT_TIMESTAMP);

    RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';
```

- Creamos el trigger que gatilla la ejecución acción sobre log_accion luego de insertar un registro en la tabla tarea

```
CREATE TRIGGER trg_insert_tarea_log
  AFTER INSERT
  ON tarea
  FOR EACH ROW
  EXECUTE PROCEDURE insert_tarea_logaccion();
```

- verifiquemos que funciona correctamente

```
INSERT INTO public.tarea(
        id, id_emergencia,nombre, descrip, cant_vol_requeridos, cant_vol_inscritos, finicio, ffin, id_estado)
        VALUES ('10000','501' , 'tarea-aa','descrip-aa', 10, 8, CURRENT_DATE, NULL, 1);

select * from public.log_accion;
```

Carga de la DB

Cargar el backup de la db desde el archivo **voluntariadodb_backup** de la carpeta clase4, ver documento clase3 de la carpeta clase3 que contiene la especificación de la db, tablas y columnas

Carga masiva de datos para pruebas

Realice pruebas de carga de datos utilizando el programa **load_data.py**, es fácil de manipular (modificar arreglo llamado **tables**) y modificar por si quiere hacer carga para otras bases de datos postgresql

SQL complejos

- listado de tareas por emergencia

```
select b.* from emergencia a, tarea b
where a.id=b.id_emergencia;
```

- institución que ha participado en más emergencias

```
select a.id,a.nombre, count(b.id) cant_eme
   from institucion a, emergencia b
   where a.id=b.id_institucion
   group by a.id,a.nombre
   having count(b.id)=(
                        select max(cant_eme) max_cant_eme
                        from (
                        select a.id, count(b.id) cant_eme
                        from institucion a, emergencia b
                        where a.id=b.id_institucion
                        group by a.id
                        ) a
            );
```

- máxima cantidad de voluntarios por emergencia

```
select max(cant_vol) from (
   select a.id_emergencia, count(c.id) cant_vol
   from tarea a, ranking b, voluntario c
   where a.id = b.id_tarea
   and b.id_voluntario=c.id
   group by a.id_emergencia
   ) a
```

- mínima cantidad de voluntarios por tarea

```
select min(cant_vol) from (
   select a.id, count(c.id) cant_vol
   from tarea a, ranking b, voluntario c
   where a.id = b.id_tarea
   and b.id_voluntario=c.id
   group by a.id
   ) a
```

- listar tareas finalizadas entre 2016 y 2020 (estado = 2)

```
select * from tarea
where id_estado=2
and EXTRACT(YEAR FROM ffin) between 2016 and 2020;
```

- listar tareas que no lograron la cantidad de participantes requeridos

```
select a.* from tarea a
where a.cant_vol_requeridos > a.cant_vol_inscritos;
```

- listar las 3 habilidades más comunes dentro de los voluntarios

```
select a.id_habilidad,count(a.id_habilidad) cant
from vol_habilidad a
group by a.id_habilidad
order by  count(a.id_habilidad) desc
limit 3;
```

- listar las 3 habilidades más solicitadas por emergencia

```
select a.id_habilidad,count(a.id_habilidad) cant
from eme_habilidad a
group by a.id_habilidad
order by  count(a.id_habilidad) desc
limit 3;
```

- detectar al voluntario que ha participado en más emergencias

```
select id,nombre, count(id_emergencia) cant_eme
from (
   select c.id,c.nombre,a.id_emergencia
   from tarea a, ranking b, voluntario c
   where a.id = b.id_tarea
   and b.id_voluntario=c.id
   group by c.id,c.nombre,a.id_emergencia
   ) a
   group by id, nombre
   having count(id_emergencia) = (

                        select max(cant_eme) max_cant_eme
                        from (
                                select id, count(id_emergencia) cant_eme
                                from (
                                        select c.id,a.id_emergencia
                                        from tarea a, ranking b, voluntario c
                                        where a.id = b.id_tarea
                                        and b.id_voluntario=c.id
                                        group by c.id,a.id_emergencia
                                        ) a
                                group by id
                        ) a

   );
```

- voluntario con máximo puntaje para participar por tarea

```
select a.id,a.nombre, c.id_tarea, c.nombre_tarea, b.puntaje
from voluntario a, ranking b, (
                                select a.id id_tarea,a.nombre nombre_tarea, max(b.puntaje) max_puntaje
                                from tarea a, ranking b
                                where a.id=b.id_tarea
                                group by  a.id,a.nombre) c
where
a.id=b.id_voluntario
and b.id_tarea = c.id_tarea
and b.puntaje = c.max_puntaje
order by  c.id_tarea;
```

- voluntario con puntaje más bajo por tarea ejecutadas entre el 2016 y 2019

```
select a.id,a.nombre, c.id_tarea, c.nombre_tarea, b.puntaje
from voluntario a, ranking b, (
                                select a.id id_tarea,a.nombre nombre_tarea, min(b.puntaje) min_puntaje
                                from tarea a, ranking b
                                where a.id=b.id_tarea
                                and  EXTRACT(YEAR FROM a.ffin) between 2016 and 2019
                                group by  a.id,a.nombre) c
where
a.id=b.id_voluntario
and b.id_tarea = c.id_tarea
and b.puntaje = c.min_puntaje
order by  c.id_tarea;
```