

Base de Datos II



TEMAS DE BASE DE DATOS II

Profundizar Modelo Relacional-SQL

Administración SGBD

Procesamiento de Consultas y optimización

Transacciones (Tema central)

Concurrencia

Recuperación

Programación con acceso a base de datos



EVALUACIÓN Y CALIFICACIÓN DEL CURSO

Primera evaluación (Parcial 1) = 25%

Segunda evaluación (Parcial 2) = 35%

Evaluación Continua (Laboratorios+Defensa) = 40% I



Temario:

- Profundización en Modelo Relacional
 - Consultas avanzadas en SQL.
 - Vistas.
 - Sentencias DDL y DCL.
- Conceptos relacionados con la administración de un SGBD relacional.
 - Catálogo del sistema.
 - Seguridad y autorización.
 - Arquitectura de manejadores de bases de datos
- Procesamiento de consultas y optimización.
 - Índices.
- Transacciones
 - Conceptos de transacciones.
 - Concurrency.
 - Recuperación.
- Programación con acceso a base de datos.
 - Procedimientos almacenados y triggers.
 - SQL embebido en un lenguaje de programación.

08-03-21

Sentencia de definición DDL Posgresql

- CREATE (crea)
CREATE TABLE
CREATE VIEW
- ALTER (modifica)
ALTER TABLE

ALTER VIWN
• DROP (elimina)

DROP TABLE

DROP VIWN

ALGUNAS COSITAS A TENER ENCUENTA:

- Doble guion -- comentario
- Serial significa autoincrementar. (no es un tipo de datos, el tipo de datos serias INT. Es una particularidad de postgresql).
- UNIQUE hace que el campo, aunque no sea clave primaria no se pueda repetir.
- Por ejemplo, en un campo “sueldo money NOT NULL”, no lo puede dejar nulo.
- ; (llego al final la sentencia)

Se puede crear una table sin clave primaria según el motor, pero no es lo que dice el teórico. Problemas: repetición de datos, no tenemos forma segura de recuperar una tupla de forma única.

Sentencia para agregarle una clave primaria:

```
ALTER TABLE empleados ADD CONSTRAINT id_pkey PRIMARY KEY (id);
```

EJEMPLO 2

EJEMPLO 2

```
CREATE TABLE cuenta (
    nombre VARCHAR(20),
    activa CHAR(2) DEFAULT 'SI',
    balance NUMERIC(16,2) DEFAULT 0,
    creada TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

COMENTARIOS

DEFAULT siempre pone “SI” por defecto, si quisiéramos guardar otra información lo especificamos en el INSERT.

TIMESTAMAP te guarda fecha y hora.

¿Cuál es la diferencia entre VARCHAR y CHAR?

- VARCHAR te da un máximo, pero no un mínimo y CHAR tiene que tener exactamente esa cantidad específica.
- CHAR sirve para guarda datos que sabemos el largo exacto, la ci o números de la credencial.

TAREA EN GRUPO CLASE

Tabla departamento código (entero clave primaria) y nombre (varchar(50)).

Segunda tabla Profesor ci (char (8)), nombre (varchar(50)) y codigodpto.

FOREIGN KEY (codigodpto)REFERENCES departamento(código)

```
CREATE TABLE departamento (
    código serial PRIMARY KEY,
    nombre VARCHAR(50),
);

CREATE TABLE profesor (
    ci CHAR(8) PRIMARY KEY,
    nombre VARCHAR(50),
    codigodpto INT,
    FOREIGN KEY (codigodpto) REFERENCES departamento(código)
);
```

POR EL PROFE

```
CREATE TABLE departamento (
    código INT PRIMARY KEY,
    nombre VARCHAR(50)
);

--sin cascada
CREATE TABLE profesor (
    cedula INT PRIMARY KEY,
    nombre VARCHAR(50),
    sueldo NUMERIC,
    codigodpto INT,
    FOREIGN KEY (codigodpto) REFERENCES departamento(código)
);
DROP TABLE profesor;
```

Ahora la crea nuevamente con cascada que actualice en cascada y borre en cascada:

```
--con cascada
CREATE TABLE profesor (
    cedula INT PRIMARY KEY,
    nombre VARCHAR(50),
    sueldo NUMERIC,
    codigodpto INT,
    FOREIGN KEY (codigodpto) REFERENCES departamento(código)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

FOREING

Son 5 las opciones: RESTRICT, NO ACTION, CASCADE, SET NULL, SET DEFAULT.

RESTRICT	No permite la acción sobre la fila en cuestión (Pero no genera ningún error)
NO ACTION	No permite la acción sobre la fila en cuestión y genera un error. Esta es la opción por defecto
CASCADE	Realiza la eliminación o los cambios en cascada
SET NULL	Asigna el valor NULL a las filas que apunten a la fila modificada (Genera un error si esto no es posible o viola alguna restricción)
SET DEFAULT	Asigna el valor por defecto a las filas que apunten a la fila modificada (Genera un error si esto no es posible o viola alguna restricción)

Realizar cambios en la tabla profesor, agrega un campo, luego lo elimina y lo vuelve a agregar con un valor por defecto.

```
ALTER TABLE profesor ADD COLUMN seguro NUMERIC;
ALTER TABLE profesor DROP COLUMN seguro;
ALTER TABLE profesor ADD COLUMN seguro NUMERIC DEFAULT 0.3;
```

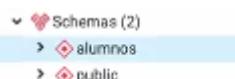
Para crear una base de datos con nombre prueba01.

```
CREATE DATABASE prueba01;
```

ESQUEMAS

Por defecto crea las tablas en el público. Los esquemas permiten organizar la base de datos.

CREATE SCHEMA alumnos;



Ahora tenemos un esquema que sirve para: para los permisos de la base de datos, más eficiencia de la seguridad, porque se le puede agregar permisos al esquema.

Organizadores lógicos de objeto.

AHORA UN EJEMPLO

Ahora creamos una tabla en el esquema que creamos, así como hacemos tabla.campo. Se hace igual con los esquemas.

TAREA

Un esquema alumno con una tabla materia (código PRIMARY KEY y nombre VARCHAR(20))
INSERT en materia para ver como se inserta en un esquema.

```
CREATE SCHEMA alumno;  
  
CREATE TABLE alumno.materia(  
    codigo INT PRIMARY KEY,  
    nombre VARCHAR(20)  
);
```

```
INSERT INTO alumnos.materia (código, nombre) VALUE (111, 'pepe');
```

Por último, para terminar la clase de hoy, consola de postgresql, el intérprete de comando.

En el EVA, primeros pasos con la consola.

PRIMEROS PASOS CON LA CONSOLA DE POSTGRES

Lista de los comandos más comunes para gestionar bases de datos en postgres desde la terminal:

```
\l  --> lista las bases de datos existentes  
\c  --> cambiar a otra base de datos ( \c nueva_base ) por ejemplo  
\d  --> ver definiciones de una tabla ( \d tabla) por ejemplo  
\dn  --> listar los esquemas  
\dn+ --> listar los esquemas con más detalles de privilegios y la descripción  
\df  --> listar definiciones  
\dv  --> listar las vistas  
\du  --> listas los usuarios  
\q  --> salir del gestor  
Para conocer una lista más amplia de los comandos en psql, ejecutamos:  
postgres# \?
```

TAREAS:

- Practico 2 y 3
- Leer teórico DDL-EJEMPLO

EJERCICIO PRÁCTICO 02 - BASES DE DATOS 2

ACTIVIDADES QUE SE REALIZAN:

1. Crea una base de datos llamada TIENDA
2. Genera las siguientes tablas:

FABRICANTES		
	NOMBRE DE COLUMNA	TIPO DE DATO
Clave Principal	Clave_fabricante	Int
	Nombre	Varchar(30)

```
CREATE TABLE FABRICANTES (Clave_fabricante  
INT NOT NULL, Nombre VARCHAR(30), PRIMARY  
KEY (Clave_fabricante));
```

ARTICULOS		
	NOMBRE DE COLUMNA	TIPO DE DATO
Clave Principal	Clave_articulo	Int
	Nombre	Varchar(30)
	Precio	Int
Clave Foránea	Clave_fabricante	Int

```
CREATE TABLE ARTICULOS (Clave_articulo INT NOT  
NULL, Nombre VARCHAR(30), Precio INT,  
Clave_fabricante INT, PRIMARY KEY (Clave_articulo),  
FOREING KEY(Clave_fabricante) REFERENCES  
FABRICANTES (Clave_fabricante));
```

SE PIDE:

1. Genere las sentencias SQL para introducir los siguientes datos en las tablas:

TABLA: FABRICANTES	
CLAVE FABRICANTE	NOMBRE
1	Kingston
2	Adata

TABLA: ARTICULOS			
CLAVE_ARTICULO	NOMBRE	PRECIO	CLAVE_FABRICANTE
1	Teclado	\$ 100	2
2	Disco duro 300 Gb	\$ 500	1

2. Genera las siguientes consultas SQL:

- a. Obtener los nombres de todos los productos de la tienda.
- b. Obtener los nombres y precio de los productos de la tienda
- c. Obtener todos los datos del artículo cuya clave de producto es '2'
- d. Obtener todos los datos del artículo cuyo nombre del producto es "Teclado"
- e. Obtener todos los datos de los artículos que empiezan con 'M'
- f. Obtener todos los datos de los artículos cuyo precio este entre \$100 y \$350
- g. Obtener el precio medio de todos los productos
- h. Obtener todos los datos de los productos ordenados descendenteamente por Precio
- i. Obtener la clave de producto, nombre del producto y nombre del fabricante de todos los productos en venta
- j. Cambia el nombre del producto 2 a 'Impresora Laser'
- k. Aplicar un descuento de \$ 10 a todos los productos cuyo precio sea mayor o igual a \$ 300
- l. Borra el producto numero 2

```
CREATE DATABASE tienda;
```

```
CREATE TABLE fabricante (  
clave_fabricante INT PRIMARY KEY,  
nombre VARCHAR(30)  
);  
  
CREATE TABLE articulo (  
clave_articulo INT PRIMARY KEY,  
nombre VARCHAR(30),
```

```
precio INT,  
clave_fabricante INT,  
FOREING KEY (clave_fabricante) REFERENCES fabricante (clave_fabricante));
```

INSERT PARTE 1:

```
INSERT INTO fabricante (codigo_fabricante, nombre) VALUE (1,'Kingston');  
INSERT INTO fabricante (codigo_fabricante, nombre) VALUE (2,'Adata');  
  
INSERT INTO articulo (clave_articulo, nombre, precio, clave_fabricante) VALUES (1, 'Teclado', 100, 2);  
INSERT INTO articulo (clave_articulo, nombre, precio, clave_fabricante) VALUES (2, 'Disco', 320, 1);
```

CONSULTAS PARTE 2:

```
1-SELECT nombre FROM articulo;  
2-SELECT nombre, precio FROM articulo;  
3-SELECT * FROM clave_articulo = 2;  
4-SELECT * FROM nombre= 'Teclado';  
5-SELECT * FROM articulo WHERE nombre LIKE 'M%';  
6-SELECT * FROM articulo WHERE precio BETWEEN 100 AND 350;  
7-SELECT AVG(precio) FROM articulo;  
8-SELECT * FROM articulo ORDEN BY precio DESC;  
9-SELECT articulo.nombre, articulo.precio, fabricante.nombre FROM articulo, fabricante  
WHERE articulo.clave_fabricante = fabricante.clave_fabricante;  
SELECT a.nombre, a.precio, f.nombre FROM articulo A JOIN fabricante F ON a.clave_fabricante  
= f.clave_fabricante;  
10-UPDATE articulo SET nombre = "Impresora Láser" WHERE código=2;  
11-UPDATE articulo SET precio = precio – 10 WHERE precio >= 300;  
12-DELETE FROM articulo WHERE clave_articulo = 2;
```

Práctico 03

1) Crear una base de datos denominada **biblioteca**.

2) Creamos la tabla estableciendo como clave primaria y serial el campo "codigo":

```
create table libros(
    codigo serial,
    titulo varchar(30),
    autor varchar(30),
    editorial varchar(15),
    primary key (codigo)
);
```

```
postgres=# CREATE DATABASE biblioteca;
CREATE DATABASE
postgres=# \l
```

Con \l vemos que esta creada con las demás.

Ahora nos movemos a ella con \c biblioteca para podés hacer el paso 2.

```
postgres=# \c biblioteca
Ahora está conectado a la base de datos «biblioteca» con el usuario «postgres».
biblioteca=# create table libros(
biblioteca(#   codigo serial,
biblioteca(#   titulo varchar(30),
biblioteca(#   autor varchar(30),
biblioteca(#   editorial varchar(15),
biblioteca(#   primary key (codigo)
biblioteca(# );
CREATE TABLE
biblioteca=#
```

Ingresamos datos. Note que al detallar los campos para los cuales ingresaremos valores hemos omitido "codigo"; **cuando un campo es serial no es necesario ingresar valor para él, porque se genera automáticamente**. Recuerde que si es obligatorio ingresar los datos de todos los campos que se detallan y en el mismo orden.

```
biblioteca=# insert into libros (titulo,autor,editorial)
biblioteca-#   values('El aleph','Borges','Planeta');
INSERT 0 1
biblioteca=#
```

Hacemos un SELECT, vemos que este primer registro ingresado guardó el valor 1 en el campo correspondiente al código, comenzó la secuencia en 1.

```
biblioteca=# select * from libros;
codigo | titulo | autor | editorial
-----+-----+-----+
      1 | El aleph | Borges | Planeta
(1 fila)

biblioteca=#
```

Ingresamos mas datos

```
biblioteca=# insert into libros (titulo,autor,editorial)
biblioteca-#   values('Martin Fierro','Jose Hernandez','Emece');
INSERT 0 1
biblioteca=# insert into libros (titulo,autor,editorial)
biblioteca-#   values('Aprenda PHP','Mario Molina','Emece');
INSERT 0 1
biblioteca=# insert into libros (titulo,autor,editorial)
biblioteca-#   values('Cervantes y el quijote','Borges','Paidos');
INSERT 0 1
biblioteca=# insert into libros (titulo,autor,editorial)
biblioteca-#   values('Matematica estas ahi', 'Paenza', 'Paidos');
INSERT 0 1
biblioteca=#
```

Si hacemos un SELECT de los libros, Vemos que el código, dato que no ingresamos, **se cargó automáticamente siguiendo la secuencia de autoincremento.**

Borramos el libro de código 1.

```
biblioteca=# delete from libros where codigo=1;
DELETE 1
```

Luego si insertamos otro registro sin indicar el valor del campo serial el valor generado es el siguiente del último generado:

```
biblioteca=# insert into libros (titulo,autor,editorial)
biblioteca-#   values('Java Ya', 'Nelson', 'Paidos');
INSERT 0 1
```

Por último SELECT de la tabla libros.

```

biblioteca=# SELECT * FROM libros;
codigo |         titulo        |     autor    | editorial
-----+---------------------+-----+-----+
  2 | Martin Fierro      | Jose Hernandez | Emece
  3 | Aprenda PHP         | Mario Molina  | Emece
  4 | Cervantes y el quijote | Borges       | Paidos
  5 | Matematica estas ahi | Paenza      | Paidos
  6 | Java Ya             | Nelson      | Paidos
(5 filas)

biblioteca=#

```

CLASE DE MIERCOLES 10/03/2021

El scrip ejecuta una serie de acción una tras otra.

En el EVA podemos descargar un scrip de la base de datos Pedidos.

```

--CREACION DE TABLAS
CREATE TABLE EMPLEADOS(
    EMPLEADOID int NOT NULL,
    NOMBRE char(30) NULL,
    APELLIDO char(30) NULL,
    FECHA_NAC timestamp NULL,
    REPORTA_A int NULL,
    EXTENSION int NULL,
    CONSTRAINT PK_EMPLEADOS PRIMARY KEY (EMPLEADOID));

```

CONSTRAINT es una restricción, nombra la clave primaria eso sirve para identificarlas.

Por convención:

- PK CLAVE PRIMARIA
- FK FORANEA
- CK es para restricciones.

Una especie en programación es un error.

EJERCICIO CLASE MIÉRCOLES 10/3

Crear una base de datos GESTION

Crear una tabla CLIENTES con los campos id serial, nombre varchar 20, apellido varchar 20. La clave primaria debe llamarse pk_cliente y debe ser el campo id.

Crear una tabla LIBROS con los campos ISBN varchar 10, titulo varchar 50 y stock INT. Stock por defecto debe ser cero, y titulo no puede aceptar nulos. La clave primaria debe llamarse pk_libro y debe ser el campo ISBN.

Crear una tabla PRESTAMOS con los campos idPrestamos serial, idCliente INT, ISBN varchar 10 y fecha_prestamo date. La clave primaria debe llamarse pk_prestamo y debe ser el campo isPrestamo. idCliente debe ser foranea de clientes (debe llamarse fk_cliente) e ISBN debe ser foranea de libros (debe llamarse fk_libros) En el caso de fk_cliente no debe permitir eliminarse los datos relacionados y en el caso de fk_libros debe permitir actualizaciones y eliminaciones en cascada. Los campos idCliente e ISBN no deben permitir nulos.

SOLUCION

```
postgres=# CREATE DATABASE gestion
postgres-# ;
CREATE DATABASE
postgres=# \c gestion
Ahora está conectado a la base de datos «gestion» con el usuario «postgres».
gestion=# CREATE TABLE clientes(
gestion(# id serial,
gestion(# nombre varchar(30),
gestion(# apellido varchar(30),
gestion(# CONSTRAINT PK_CLIENTE PRIMARY KEY (id));
CREATE TABLE

gestion=# CREATE TABLE libros(
gestion(# isbn varchar(10),
gestion(# titulo varchar(50) NOT NULL,
gestion(# stock int DEFAULT 0,
gestion(# CONSTRAINT PK_LIBRO PRIMARY KEY (isbn));
CREATE TABLE

CREATE TABLE prestamos(
idprestamo serial,
idcliente int NOT NULL,
isbn varchar(10) NOT NULL,
fecha_prestamo date,
CONSTRAINT PK_PRESTAMO PRIMARY KEY (idprestamo),
CONSTRAINT FK_CLIENTE FOREIGN KEY (idcliente) REFERENCES clientes(id) ON DELETE RESTRICT,
CONSTRAINT FK_LIBROS FOREIGN KEY (isbn) REFERENCES libros(isbn) ON DELETE CASCADE));
```

Sobra un).

NOTAS DE REPASO:

Primero crea dos esquemas.

```
CREATE SCHEMA Persona;
CREATE SCHEMA Admision;
```

Creamos una tabla de prueba

```
create table prueba (
    id serial primary key
)
```

Se creo eso:

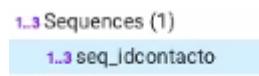


Eso pasa cuando definimos un serial. Posgre crea una secuencia y cada vez que creamos un dato él se encarga de asignarles el número que corresponde.

AHORA:

```
CREATE SEQUENCE Persona.seq_idcontacto START 1; --empieza desde 1
```

Se crea:



Ahora creamos una table, ahora utilizamos la secuencia que creamos recién.

```
create table Persona.Contacto
(
    IDContacto int default nextval('Persona.seq_idcontacto'),
    Nombres varchar(30) not null,
    Paterno varchar(30) not null,
    Materno varchar(30) not null,
    Genero char(1) default('0') not null,
    DNI varchar(10) null,
    FechaNac date null,
    FechaCreacion date not null default now()
);
```

AHORA:

Creamos una tabla en cada esquema. Tenemos una tabla constan que tiene varias columnas. Tiene un campo género que guarda 0 o 1 eso se hace:

```
ALTER TABLE Persona.Contacto
ADD CONSTRAINT ck_Genero
CHECK (Genero in ('0','1'))
;
```

CK es para restricciones y aparecen así:



En fin, podemos poner reglas y restricciones en la base de datos, para controlar la información.

TAREA

Al ejercicio de recién. Hacer una regla que el stock no sea negativo. Y que la fecha de préstamos no sea anterior a la fecha actual.

```
ALTER TABLE libros ADD ck_stock CHECK (stock>=0);  
ALTER TABLE libros ADD CONSTRAINT CK_STOCK check(stock>=0);  
ALTER TABLE prestamos ADD CONSTRAINT CK_FECHA check(fecha_prestamo>=current_date);
```

PRUEBA DEL PROFESOR

```
CREATE TABLE resultados_examen (  
  
    id serial primary key,  
    alumno varchar(20),  
    fecha date default current_date,  
    nota INT  
);  
  
alter table resultados_examen  
add constraint CK_notas  
check (nota between 1 and 12);  
  
insert into resultados_examen (alumno,nota) values ('alumno1',1);  
insert into resultados_examen (alumno,nota) values ('alumno2',12);  
insert into resultados_examen (alumno,nota) values('alumno3',13);
```

NOTAS:

Fecha por defecto current_date (fecha de hoy).

Creamos una restricción que las notas del examen sean del 1 a 12.

Ahora cuando hacemos los insert, el primer y segundo insert se ejecutan con éxito. Pero cuando insertamos el ultimo sale un error que dice que viola la restriccción ck_notas.

```
ERROR: new row for relation "resultados_examen" violates check constraint "ck_notas"  
DETAIL: Failing row contains (3, alumno3, 2021-03-10, 13).  
SQL state: 23514
```

CONSTRUIR UNA NUEVA TABLA CON LOS DATOS DE OTRA TABLA YA EXISTENTE

```
CREATE TABLE tablaNueva AS SELECT * FROM resultados_examen WHERE nota<6;
```

Arma una nueva tabla con la salida del SELECT.

PRACTICOS 4

Tabla: vino

ID	VINO	PRODUCTOR	AÑO	BOTELLAS	LISTO
2	Chardonnay	Buena Vista	1997	1	1999
3	Chardonnay	Geyser Peak	1997	5	1999
6	Chardonnay	Simi	1996	4	1998
12	Joh. Riesling	Jekel	1998	1	1999
21	Fumé Blanc	Ch. St. Jean	1997	4	1999
22	Fumé Blanc	Robt. Mondavi	1996	2	1998
30	Gewurztraminer	Ch. St. Jean	1998	3	1999
43	Cab. Sauvignon	Windsor	1991	12	2000
45	Cab. Sauvignon	Geyser Peak	1994	12	2002
48	Cab. Sauvignon	Robt. Mondavi	1993	12	2004
50	Pinot Noir	Gary Farrel	1996	3	1999
51	Pinot Noir	Fetzer	1993	3	2000
52	Pinot Noir	Dehlinger	1995	2	1998
58	Merlot	Clos du Bois	1994	9	2000
64	Zinfandel	Cline	1994	9	2003
72	Zinfandel	Rafanelli	1995	2	2003

1. Obtener el id, nombre del vino y número de botellas de todos los vinos de Mirassou.
2. Obtener el id y el nombre de todos los vinos con más de cinco botellas en existencia.
3. Obtener el id de todos los vinos tintos.
4. Agregar tres botellas al id número 30.
5. Eliminar todo el Chardonnay de las existencias.
6. Agregar un registro para una caja nueva (12 botellas) de Mirassou Chardonnay: id número 5, año 2012, listo en 2011, con "1er premio" como comentario.

SOLUCION

- 1- SELECT id, vino, botellas FROM vino WHERE vino='Mirassou';
- 2- SELECT id, nombre FROM vino WHERE botellas >5;
- 3- SELECT id FROM vino WHERE vino='Tinto';
- 4- UPDATE vino SET botellas= botellas+3 WHERE id=30;
- 5- DELETE FROM vino WHERE vino='Chardonnay';
- 6- INSERT INTO vino (id, vino, productor, año, botellas, listo) values (5, 'Mirassou Chardonnay', '1er premio', 2012, 12, 2011);

PRÁCTICO 06 – SQL – DML

Se disponen de la siguientes tablas con los siguientes datos.

TABLA EMPLE

■ emple : Tabla							
emp_no	apellido	oficio	dir	fecha_alt	salario	comision	dept_no
7369	SÁNCHEZ	EMPLEADO	7902	17/12/1980	104000		20
7499	ARROYO	VENDEDOR	7698	20/02/1980	208000	39000	30
7521	SALA	VENDEDOR	7698	22/02/1981	162500	162500	30
7566	JIMÉNEZ	DIRECTOR	7839	02/04/1981	386750		20
7654	MARTÍN	VENDEDOR	7698	29/09/1981	162500	182000	30
7698	NEGRO	DIRECTOR	7839	01/05/1981	370500		30
7788	GIL	ANALISTA	7566	09/11/1981	390000		20
7839	REY	PRESIDENTE		17/11/1981	650000		10
7844	TOVAR	VENDEDOR	7698	08/09/1981	195000	0	30
7876	ALONSO	EMPLEADO	7788	23/09/1981	143000		20
7900	JIMENO	EMPLEADO	7698	03/12/1981	1235000		30
7902	FERNÁNDEZ	ANALISTA	7566	03/12/1981	390000		20
7934	MUÑOZ	EMPLEADO	7782	23/01/1982	169000		10

TABLA DEPART.

■ depart : Tabla			
	dept_no	dnombre	loc
► +	10	CONTABILIDAD	SEVILLA
+	20	INVESTIGACIÓN	MADRID
+	30	VENTAS	BARCELONA
+	40	PRODUCCIÓN	BILBAO

Realizar las siguientes consultas:

1. Mostrar el apellido, oficio y número de departamento de cada empleado.
2. Datos de los empleados ordenados por apellidos.
3. Datos de los empleados ordenados por número de departamento descendente.
4. Datos de los empleados ordenados por número de departamento descendente y dentro de cada departamento ordenados por apellido ascendente.
5. Mostrar los datos de los empleados cuyo salario sea mayor que 2000000.
6. Mostrar los datos de los empleados cuyo oficio sea 'ANALISTA'.
7. Seleccionar el apellido y oficio de los empleados del departamento número 20.
8. Mostrar los empleados cuyo departamento sea 10 y cuyo oficio sea 'ANALISTA'. Ordenar el resultado por apellido.
9. Mostrar los empleados que tengan un salario mayor que 200000 o que pertenezcan al departamento número 20.
10. Seleccionar de la tabla EMPLE los empleados cuyo apellido empiece por 'A'.
11. Seleccionar de la tabla EMPLE los empleados cuyo apellido termine por 'Z'.
12. Seleccionar de la tabla EMPLE aquellas filas cuyo APELLIDO empiece por 'A' y el OFICIO tenga una 'E' en cualquier posición.
13. Seleccionar los empleados cuyo salario esté entre 100000 y 200000. Utilizar el operador BETWEEN.
14. Obtener los empleados cuyo oficio sea 'VENDEDOR' y tengan una comisión superior a 100000.
15. Datos de los departamentos cuya localización empiece por 'B'.

16. Datos de los empleados cuyo oficio sea 'EMPLEADO', tengan un salario superior a 100000 y pertenezcan al departamento número 10.
17. Mostrar los apellidos de los empleados cuyo oficio sea 'VENDEDOR', 'ANALISTA' o 'EMPLEADO'.
18. Mostrar los apellidos de los empleados cuyo oficio no sea ni 'ANALISTA' ni 'EMPLEADO', y además tengan un salario mayor de 200000.
19. Lista los nombres y fecha de contratación de aquellos empleados que no son vendedores.
20. ¿Cuántos empleados tienen comisión?
21. ¿Cuántos empleados tiene el departamento 20?
22. Halla la suma de salarios de la empresa.
23. Obtén todos los departamentos sin empleados.
24. Obtén los empleados que no son supervisados por ningún otro.
25. Para los empleados que tengan como jefe a un empleado con código mayor que el suyo, obtén los que reciben de salario más de 1000 y menos de 2000, o que están en el departamento 30.

SOLUCION:

- 1-SELECT apellido,oficio,dept_no FROM emple;
- 2-SELECT * FROM emple ORDER BY apellido;
- 3- SELECT * FROM EMPLE ORDER BY dept_no desc;
- 4- SELECT * FROM EMPLE ORDER BY dept_no desc, apellido asc;
- 5- SELECT * FROM EMPLE WHERE salario > 2000000;
- 6- SELECT * FROM EMPLE WHERE oficio='ANALISTA';
- 7- SELECT apellido,oficio FROM EMPLE WHERE dept_no=20;
- 8- SELECT * FROM EMPLE WHERE dept_no = 10 AND oficio='ANALISTA' ORDER BY apellido;
- 9- SELECT * FROM EMPLE WHERE salario > 200000 OR dept_no = 20;
- 10- SELECT * FROM EMPLE WHERE apellido LIKE 'A%';
- 11- SELECT * FROM EMPLE WHERE apellido LIKE '%Z';
- 12- SELECT * FROM EMPLE WHERE apellido LIKE 'A%' AND oficio LIKE '%E%';
- 13- SELECT * FROM EMPLE WHERE salario BETWEEN 100000 AND 200000;
- 14- SELECT * FROM EMPLE WHERE oficio='VENDEDOR' AND comision >100000;
- 15- SELECT * FROM DEPART WHERE loc LIKE 'B%';
- 16- SELECT * FROM EMPLE WHERE oficio='EMPLEADO' AND salario > 100000 AND dept_no = 10;
- 17- SELECT apellido FROM EMPLE WHERE oficio='VENDEDOR' OR oficio='ANALISTA' OR oficio='EMPLEADO'

18- SELECT apellido FROM EMPLE WHERE oficio <>'ANALISTA' AND oficio <>'EMPLEADO' AND salario > 200000;

19- SELECT apellido, fecha_alt FROM EMPLE WHERE oficio <> 'VENDEDOR';

20- SELECT COUNT(comision) FROM emple; //4 empleados

20- SELECT COUNT(comision) FROM emple WHERE comision IS NOT NULL AND comision > 0;

21- SELECT COUNT(*) FROM emple WHERE dept_no = 20 AND oficio='EMPLEADO';

21- SELECT COUNT(*) FROM emple WHERE dept_no = 20;

22- SELECT SUM(salario) FROM emple; // 4566250

23-SELECT dnombre FROM depart WHERE NOT EXISTS (SELECT dept_no FROM emple WHERE emple.dept_no = depart.dept_no);

24- SELECT * FROM EMPLE WHERE dir IS NULL;

25- SELECT * FROM EMPLE WHERE emp_no > dir AND salario BETWEEN 1000 AND 200 OR dept_no = 30;

PRACTICO 5

1. Obtener los detalles completos de todos los proyectos.
2. Obtener los detalles completos de todos los proyectos de Londres.
3. Obtener los números de los proveedores que suministran partes al proyecto J1, ordenados por número de proveedor.
4. Obtener todos los envíos en los cuales la cantidad está en el intervalo de 300 a 750 inclusive.
5. Obtener todas las tripletas número de proveedor / número de parte / número de proyecto tales que el proveedor, la parte y el proyecto indicados estén todos en la misma ciudad.
6. Obtener los números de las partes suministradas por algún proveedor de Londres.
7. Obtener los números de las partes suministradas por algún proveedor de Londres a un proyecto en Londres.
8. Obtener el número total de proyectos a los cuales suministra partes el proveedor S1.
9. Obtener la cantidad total de la parte P1 suministrada por el proveedor S1.
10. Para cada parte suministrada a un proyecto, obtener el número de parte, el número de proyecto y la cantidad total correspondiente.
11. Obtener los números de partes suministradas a algún proyecto tales que la cantidad promedio suministrada sea mayor que 320.
12. Obtener todos los envíos para los cuales la cantidad no sea nula.
13. Obtener números de proyectos y ciudades en los cuales la segunda letra del nombre de la ciudad sea una "o".
14. Obtener los nombres de los proyectos a los cuales suministra partes el proveedor S1.
15. Mostrar el resultado de la siguiente selección:

```
SELECT P.COLOR FROM P
UNION
SELECT P.COLOR FROM P
```

16. Construir una lista ordenada de todas las ciudades en las cuales esté situado por lo menos un proveedor, una parte o un Proyecto.
17. Obtener los números de los proveedores cuya situación sea inferior a la del proveedor S1.

SOLUCION

```
1-SELECT * FROM j;
2-SELECT * FROM j WHERE ciudad='LONDRES';
3-SELECT s_s FROM spj WHERE j_j='J1' ORDER BY s_s;
4-SELECT * FROM spj WHERE cant BETWEEN 300 AND 750;
5-SELECT * FROM spj JOIN s ON s_s = s.s JOIN j ON j_j = j.j JOIN p ON p_p = p.p WHERE j.ciudad
= s.ciudad AND s.ciudad = p.ciudad
5.2- SELECT s.s, j.j, p.p, p.ciudad, s.ciudad, j.ciudad FROM s, j, p WHERE p.ciudad = s.ciudad
AND j.ciudad = s.ciudad
6- SELECT DISTINCT p_p FROM spj JOIN s ON spj.s_s = s.s WHERE s.ciudad ='LONDRES'
7-SELECT s_s FROM spj JOIN s ON s_s = s.s JOIN j ON j_j = j.j WHERE j.ciudad = 'LONDRES' AND
s.ciudad = 'LONDRES'
8- SELECT COUNT(DISTINCT j_j) FROM spj WHERE s_s = 'S1';
9- SELECT COUNT(cant) FROM spj WHERE p_p = 'P1' AND s_s = 'S1' GROUP BY p_p;
10- SELECT count(cant) FROM spj JOIN j ON j_j = j.j JOIN p ON p_p = p.p;
13- SELECT j_ciudad FROM j WHERE j_ciudad LIKE '_O%';
14- SELECT jnombre FROM j WHERE j IN(SELECT j_j FROM spj WHERE s_s = 'S1');
```

15-

```
1 SELECT p.color FROM p UNION SELECT p.color FROM p
```

Data Output		Explain	Messages	Notifications
	color character (10)			
1	AZUL			
2	GRIS			
3	VERDE			

16-

Luego del practico 5 la parte 2 podemos hacerla hasta el punto 3 sin problemas.

El punto 4, Forma de resolverla usando el ej anterior de las notas del examen:

```
CREATE TABLE tablaNueva AS SELECT * FROM resultados_examen WHERE nota<6;
```

Arma una nueva tabla con la salida del SELECT.

PRÁCTICO 5.2

SQL – DML

Todos estos ejercicios utilizan la base de datos de proveedores, partes y proyectos.

1. Cambiar a color gris el color de todas las partes rojas.
2. Eliminar todos los proyectos para los cuales no haya envíos.
3. Insertar un nuevo proveedor (S10) en la tabla S. El nombre y la ciudad son 'Salazar' y 'Nueva York'. La situación es 25.
4. Construir una tabla con los números de las partes suministradas por un proveedor de Londres.

SOLUCION

- 1- UPDATE p SET color='GRIS' WHERE color='ROJO'
- 2- DELETE FROM j WHERE j = (SELECT j FROM j WHERE j NOT IN (SELECT j.j FROM SPJ JOIN J ON j.j = j.j))
- 3- INSERT INTO s (s, snombre, situacion, ciudad) VALUES ('S10', 'Salazar', '25', 'NUEVO YORK');
- 4- CREATE TABLE tablaNueva AS SELECT p_p FROM spj JOIN s ON s_s = s.s WHERE s.ciudad = 'LONDRES'

PRÁCTICO 5.3

SQL – DML (EXISTS)

Todos estos ejercicios utilizan la base de datos de proveedores, partes y proyectos.

1. Obtener los nombres de los proyectos a los cuales suministra partes el proveedor S1 utilizando EXISTS en la solución.
 2. Obtener los números de los proyectos donde se utilice al menos una de las partes suministradas por el proveedor S1, utilizando EXISTS en la solución.
 3. Obtener los números de los proyectos para los cuales S1 es el único proveedor, utilizando EXISTS en la solución.
-
1. SELECT jnombre FROM j j1 WHERE EXISTS (SELECT * FROM spj WHERE s_s = 'S1' AND j1.j = spj.j_j);
 2. SELECT j FROM j WHERE EXISTS (SELECT * FROM spj WHERE s_s = 'S1');
 3. SELECT j FROM j WHERE EXISTS (SELECT * FROM spj WHERE s_s = 'S1' AND s_s != 'S1');

CLASE 15/3 LUNES

Pregunta 1 Incorrecta Puntuación 0,00 sobre 1,00 Marcar pregunta

Eliminar de la tabla tab1 el campo col3.

Seleccione una:

- a. DROP COLUMN col3 FROM TABLE tab1
- b. ALTER TABLE tab1 DELETE COLUMN col3 X
- c. ALTER TABLE tab1 DROP COLUMN col3

Respuesta incorrecta.
La respuesta correcta es: ALTER TABLE tab1 DROP COLUMN col3

Pregunta 2 Incorrecta Puntuación 0,00 sobre 1,00 Marcar pregunta

Hacer que no puedan haber dos empleados con el mismo nombre. La tabla se llama **empleados**, el campo **nombre** y la restricción debe llamarse **u_nombre**.

Respuesta: ALTER TABLE empleados ADD u_nombre CHECK (nombre=nombre) X

La respuesta correcta es: ALTER TABLE empleados ADD CONSTRAINT u_nombre UNIQUE (nombre);

Pregunta 3 Incorrecta Puntuación 0,00 sobre 1,00 Marcar pregunta

La sentencia **DROP TABLE** sirve para **eliminar una tabla**, y la podemos eliminar aunque el borrado infringe las reglas de integridad referencial (si interviene como tabla padre en una relación y tiene registros relacionados).

Seleccione una:

- Verdadero ✓
- Falso

La respuesta correcta es 'Falso'

Pregunta 4 Correcta Puntuación 1,00 sobre 1,00 Marcar pregunta

La cláusula **CONSTRAINT** sirve para definir una **restricción** que se podrá eliminar cuando queramos sin tener que borrar la columna. A cada restricción se le asigna un nombre que se utiliza para identificarla y para poder eliminarla cuando se quiera.

Seleccione una:

- Verdadero ✓
- Falso

La respuesta correcta es 'Verdadero'

Pregunta 5 Correcta Puntuación 1,00 sobre 1,00 Marcar pregunta

¿Al crear la clave primaria de una tabla debemos especificar NOT NULL?

Ejemplo:

CREATE TABLE A (a INT PRIMARY KEY NOT NULL);

Seleccione una:

- a. NO ✓
- b. SI

Respuesta correcta
La respuesta correcta es: NO

Pregunta 6 Correcta Puntuación 1,00 sobre 1,00 Marcar pregunta

En una tabla **no pueden haber varias claves principales**, por lo que no podemos incluir la cláusula **PRIMARY KEY** más de una vez, en caso contrario la sentencia da un error.

Seleccione una:

- Verdadero ✓
- Falso

La respuesta correcta es 'Verdadero'

Pregunta 7 Incorrecta Puntuación 0,00 sobre 1,00 Marcar pregunta

Agregar a la tabla clientes la columna limitecredito de tipo MONEY.

Respuesta: ALTER TABLE clientes ADD CONSTRAINT MONEY X

La respuesta correcta es: ALTER TABLE clientes ADD COLUMN limitecredito MONEY;

Restricciones de campo o columna y restricciones de tabla:

```
prueba=# CREATE TABLE producto1 (
    codigo INT,
    nombre VARCHAR(100),
    precio_ini NUMERIC CHECK (precio_ini > 0),
    precio_des NUMERIC CHECK (precio_des > 0),
    CHECK (precio_des < precio_ini),
    PRIMARY KEY(codigo)
);
CREATE TABLE
prueba=#

```

Estas son restricciones de columna

Esta es una restricción de tabla

Campos NOT NULL:

```
prueba=# CREATE TABLE persona (
    cedula INT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    apellido VARCHAR(50) NOT NULL,
    fecha_nac DATE
);
NOTICE: CREATE TABLE / PRIMARY KEY will create
        implicit index "persona_pkey" for table "persona"
CREATE TABLE
prueba=#

```

UNIQUE no permite que existe un duplicado

```
prueba=# CREATE TABLE carro (
    placa VARCHAR(6) PRIMARY KEY,
    serial_motor VARCHAR(50) UNIQUE,
    serial_carro VARCHAR(50) UNIQUE,
    color VARCHAR(10)
);
NOTICE: CREATE TABLE / PRIMARY KEY will create
        implicit index "carro_pkey" for table "carro"
NOTICE: CREATE TABLE / UNIQUE will create
        implicit index "carro_serial_motor_key" for table "carro"
NOTICE: CREATE TABLE / UNIQUE will create
        implicit index "carro_serial_carro_key" for table "carro"
CREATE TABLE
prueba=#

```

UNIQUE NOT NULL no me repita la credencial y que no sea NULL

Valores por Defecto:

```
prueba=# CREATE TABLE producto (
    codigo INT PRIMARY KEY,
    nombre VARCHAR(50),
    descripcion TEXT DEFAULT 'N/A',
    precio NUMERIC DEFAULT 100
);
NOTICE: CREATE TABLE / PRIMARY KEY will create
        implicit index "producto_pkey" for table "producto"
CREATE TABLE
prueba=#

```

Si no se especifica, el valor de descripción y de precio por defecto es 'N/A' y 100 respectivamente. Adicionalmente, es posible actualizar una fila y asignar nuevamente el valor por defecto sin conocerlo

Dos tablas relacionadas, queremos eliminar un registro.

Partiendo de la definición de las tablas **prueba1** y **prueba2** sabemos que están relacionadas. ¿Qué sucede si quiero eliminar la tabla **prueba1**?

```
prueba=# DROP TABLE prueba1;
NOTICE: constraint prueba2_a_fkey on table prueba2
depends on table prueba1
ERROR: cannot drop table prueba1 because other objects
depend on it
HINT: Use DROP ... CASCADE to drop the dependent
objects too.
```

```
prueba=# DROP TABLE prueba1 CASCADE;
NOTICE: drop cascades to constraint prueba2_a_fkey
on table prueba2
DROP TABLE
```

Elimino la tabla, según lo que dice ahí también elimina la clave foránea(la relación).

Leer Teórico

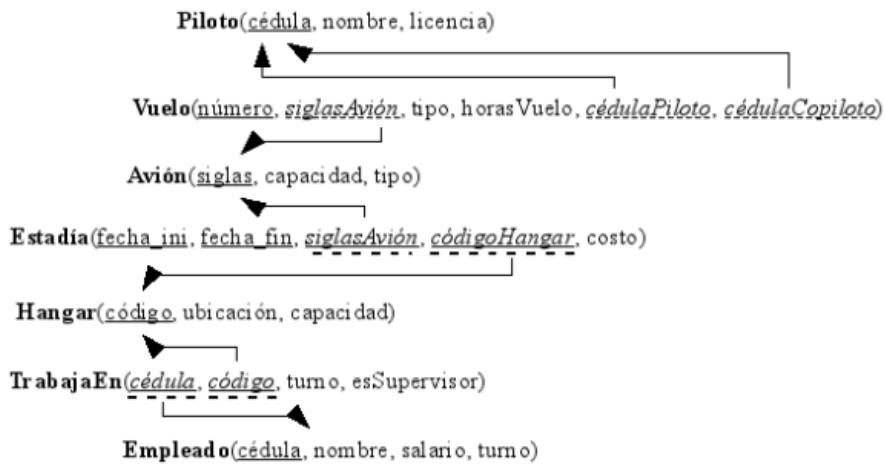
TAREA GRUPAL

BASES DE DATOS II

PRÁCTICO SQL “AEROPUERTO”

Aeropuerto

Dado el siguiente esquema relacional (Claves primarias subrayadas de forma continua y claves foráneas subrayadas de forma alternada), responder las preguntas de la página siguiente:



A.- Escribir las instrucciones en un **archivo aeropuerto_ddl.sql** en SQL para crear las tablas de la base de datos. Asuma que el salario de un empleado no puede ser menor o igual que 0, y que los nombres de los pilotos y de los empleados no pueden ser nulos.

B.- Agregar al archivo de la base de datos creada en 2.a, posteriormente a la creación de las tablas, una restricción tal que la *capacidad* de un hangar permita valores únicamente comprendidos entre 0 y 2000.

```

create table piloto (cedula int primary key, nombre varchar[20] not null, licencia varchar[20]);
create table avion (siglas varchar[10] primary key, capacidad int, tipo varchar[15]);
create table empleado (cedula int primary key, nombre varchar[20] not null, salario int, turno varchar[10]);
create table hangar (codigo int primary key, ubicacion varchar[15], capacidad int);
create table vuelo (numero int primary key, siglasAvion varchar[10], tipo varchar[15], horasVuelo int, cedulaPiloto int, cedulaCopiloto int, foreign key(siglasAvion) references avion(siglas), foreign key(cedulaPiloto) references piloto(cedula), foreign key (cedulaCopiloto) references piloto(cedula));
create table estadia (fecha_ini date, fecha_fin date, siglasAvion varchar[10], codigoHangar int, costo money, primary key (fecha_ini, fecha_fin), foreign key(siglasAvion) references avion(siglas), foreign key (codigohangar) references hangar(codigo));
create table trabajaEn (cedula int, codigo int, turno varchar[10], esSupervisor bool, primary key (cedula, codigo), foreign key(cedula) references empleado(cedula), foreign key(codigo) references hangar(codigo));

alter table empleado add constraint salario_min check(salario > 0);
alter table hangar add constraint lim_capacidad check(capacidad between 0 and 200);

```

EMPEZAMOS CON UN TEMA EXIST

Los operadores “exist” y “not exist” se emplean para determinar si hay o no datos en una lista de valores.

```

select numero,fecha from facturas f
where exists
    (select * from Detalles d
     where f.numero=d.numerofactura
     and d.articulo='lapiz');

```

Retorna verdadero o falso, no hace falta poner el valor afuera como con el IN.

Nos da una tabla para usar en la base de datos.

```

select ALIASdeTABLADERIVADA.CAMPO
from (TABLADERIVADA) as ALIAS;
La tabla derivada es una subconsulta.

select f.*,
       (select sum(d.precio*cantidad)
        from Detalles as d
        where f.numero=d.numerofactura) as total
       from facturas as f;

select td.numero,c.nombre,td.total
       from clientes as c
      join (select f.*,
                  (select sum(d.precio*cantidad)
                   from Detalles as d
                   where f.numero=d.numerofactura) as total
                  from facturas f) td
      on td.codigocliente=c.codigo;

```

	numero [PK] integer	fecha date	cliente character varying (30)	total numeric
1	1200	2018-01-...	Juan Lopez	175.00
1	1201	2018-01-...	Luis Torres	580.00
1	1202	2018-01-...	Ana Garcia	400.00
1	1300	2018-01-...	Juan Lopez	300.00

Calcula el total multiplica precio por cantidad de cada artículo y los suma

Subconsultas utiliza más recursos

Ahora otro ejemplo de subconsulta:

Select * from facturas where numero IN (Select numerofactura from detalles)

	numero [PK] integer	fecha date	cliente character varying (30)
1	1200	2018-01-...	Juan Lopez
2	1201	2018-01-...	Luis Torres
3	1202	2018-01-...	Ana Garcia
4	1300	2018-01-...	Juan Lopez

En el IN, comparamos el campo numero con lo de después del IN

AHORA:

```
| -- 1) subconsulta como expresion
| select numerofactura,articulo, precio
|   from detalles
|   where precio=
|     (select max(precio) from detalles);
|
|   --Note que el campo del "where" de la consulta exterior es compatible con el valor retornado por la expresión de la subconsulta.
|
| -- 2) Actualizamos el precio del articulo con maximo valor:
| update detalles set precio=40
| where precio=
|   (select max(precio) from detalles);
|
| -- 3) Eliminamos los articulos con precio menor:
| delete from detalles
| where precio=
|   (select min(precio) from detalles);
```

17/3 CLASE DE PRACTICOS

SELECT ci, count(nroChasis) FROM interes GROUP BY ci;

```
SELECT i.nroChasis, COUNT(i.ci) FROM interes as i
JOIN automoviles as a ON a.nroChasis=i.nroChasis
WHERE a.marca= 'Fiat' AND a.modelo= 'Uno'
AND NOT EXISTS (SELECT * FROM ventas as v WHERE v.nroChasis=i.nroChasis)
GROUP BY i.nroChasis HAVING COUNT(i.ci) > 10;
```

corrección:

```
Select ci, count(ci) Cantidad_Automoviles
From Interes
Group by ci;

Select Interes.nroChasis
From Interes
Inner Join Automoviles On Interes.nroChasis=Automoviles.nroChasis
where not exists(Select Ventas.nroChasis From Ventas) and Automoviles.Marca='Fiat' and Automoviles.modelo='Uno'
Group by Interes.nroChasis
Having count(Interes.nroChasis)>=10;
```

```
SELECT interes.nrochasis FROM interes
JOIN automoviles ON interes.nrochasis = automoviles.nrochasis
WHERE automoviles.marca = 'FIAT' AND automoviles.modelo = 'UNO'
AND interes.nrochasis NOT IN (SELECT nrochasis FROM ventas)
GROUP BY interes.nrochasis
HAVING COUNT(interes.nrochasis) > 10
```

Convierte todo en mayúsculas y compara en mayúsculas

```
UPPER(Automoviles.Marca)=UPPER('Fiat')
```

tarea 2

```
--1
select m.marca , count(m."cod-servicio") from "marca-rep" as m
join servicios as s on s."cod-servicio"=m."cod-servicio"
where s.fecha between '2017-01-01' and '2017-12-31'
group by m.marca

select c.nombre,c.tel_c,c.depto,c.ciudad,c.direccion from clientes as c
join vclientes as vc on vc.ci_cli=c.ci_cli join vbonados as va on va.mat=vc.mat
where va.mat not in(select mat from servicios where "cod-servicio" = 72 )
```

```
--1
select m.marca , count(m."cod-servicio") from "marca-rep" as m
join servicios as s on s."cod-servicio"=m."cod-servicio"
where s.fecha between '2017-01-01' and '2017-12-31'
group by m.marca
```

```
--2
select c.nombre,c.tel_c,c.depto,c.ciudad,c.direccion from clientes as c
join vclientes as vc on vc.ci_cli=c.ci_cli join vbonados as va on va.mat=vc.mat
where va.mat not in(select mat from servicios where "cod-servicio" = 72 )
```

SOLUCION

```
Select nombre,tel_c,
depto,ciudad,direccion
From clientes C,
vclientes V,
vabonados A
Where
V.ci_cli=C.ci_cli
and A.mat=V.mat and A.mat not in
( select mat From
servicios Where
"cod-servicio"=3)
```

Tenemos tablas, una tabla cliente con dos registros.

```
select ci, nombre, NOW() as fechaHoraActual from clientes
```

NOW()

CREATE VIEW

```
create view clientesprueba as
select ci, nombre, NOW() as fechaHoraActual from clientes
```

Ahora vemos



```
▼ [Views (2)]
  > [clientesprueba]
  > [misclientes]
```

Si la abrimos la view

	ci character (8)	nombre character (30)	fechahoraactual timestamp with time zone
1	1	juan	2021-03-17 20:09:55.252396-03
2	2	ana	2021-03-17 20:09:55.252396-03

Utilidad: Es como que la consulta queda guardada, al ejecutar la vista es lo mismo que ejecutar la consulta. A las vista se le puede dar seguridad. No se pueden hacer operaciones de inser o borrado a una vista.

PRACTICO 10 Y 8 DE DEBERES

PRACTICO 8

https://ev1.utec.edu.uy/moodle/pluginfile.php/326764/mod_resource/content/1/PR%C3%81CTICO%2008.pdf

Nos da una tabla y unos insert.

```
CREATE TABLE cliente
(
    id_cliente INT NOT NULL,
    nombre VARCHAR(30),
    PRIMARY KEY (id_cliente)
);

CREATE TABLE venta
(
    id_factura INT NOT NULL,
    id_cliente INT NOT NULL,
    cantidad INT,
    PRIMARY KEY(id_factura),
    FOREIGN KEY (id_cliente) REFERENCES cliente(id_cliente)
);
```

El usuario ingresa los siguientes registros:

- a) INSERT INTO cliente VALUES(1,'Juan Penas');
- b) INSERT INTO cliente VALUES(2,'Pepe el toro');
- c) INSERT INTO venta VALUES(1,1,23);
- d) INSERT INTO venta VALUES(3,2,81);
- e) INSERT INTO venta VALUES(2,3,39);

Todas las sentencias anteriores se ejecutaron con éxito **excepto una**, cuya salida fue:

ERROR: insert or update on table "venta" violates foreign key constraint "venta_id_cliente_fkey"

Se pide:

- 1) ¿Cuál de las sentencias anteriores desplegó el error? (10p)

- a.
- b.
- c.
- d.
- e.

- 2) ¿Por qué la sentencia mencionada anteriormente dio error? (15p)

- 3) El usuario de la base de datos ejecuta la siguiente sentencia SQL: **DELETE FROM cliente WHERE id_cliente=2;** y obtiene un error que no le permite eliminar el registro. A raíz de esta situación nos solicita -para poder eliminar- cambiar la restricción para que cuando se intente eliminar un registro de la tabla CLIENTE se guarde el valor NULL en el campo asociado en la tabla VENTA. ¿es posible según la definición de la estructura actual de la tabla VENTA establecer en la clave foránea **ON DELETE SET NULL**? En caso que su respuesta sea "NO" debe justificar la misma. (10p)

SI

NO

- 1) La sentencia insert que no se ejecuta es la e.

- 2) Viola la clave foránea. En la tabla venta está queriendo insertar un cliente que no existe.

3)

PARTE 2 DEL PRACTICO 8

A partir de una base de datos que tiene las siguientes tablas:

TABLA CURSOS

ID_CURSO	TITULO
1	Programación PHP
2	Modelos abstracto de datos
3	SQL desde cero
4	Dibujo técnico
5	SQL avanzado

TABLA PROFESORES

ID_PROFE	NOMBRE	APELLIDOS	F_NACIMIENTO
1	Federico	Gasco Daza	1975-04-23
2	Ana	Saura Trenzo	1969-08-02
3	Rosa	Honrosa Pérez	1980-09-05

TABLA ALUMNOS

ID_ALUMNO	NOMBRE	APELLIDOS	F_NACIMIENTO
1	Pablo	Hernandez Mata	1995-03-14
2	Jeremias	Santo Lote	1993-07-12
3	Teresa	Lomas Trillo	1989-06-19
4	Marta	Fuego García	1992-11-23
5	Sergio	Ot Dimet	1991-04-21
6	Carmen	Dilma Perna	1987-12-04

Se Pide:

- 1) La clave primaria de la tabla CURSOS es ID_CURSO, y el usuario de la base de datos nos informa que no pueden existir en la misma dos valores iguales en el campo TITULO.
a. ¿Qué solución propone? (10p)

- b. Escriba la sentencia SQL correspondiente a la solución propuesta (10p):

- 2) ¿Qué sentencia o sentencias SQL escribiría para realizar una copia de respaldo de la tabla ALUMNOS con los registros de los alumnos nacidos entre 1990 y 1993? (10p)

- 3) Escribir la sentencia SQL correspondiente para eliminar de la tabla CURSOS el registro cuyo ID_CURSO es igual a 3. (5p)

1) A-En la creación podemos especificar que el campo “titulo” sea UNIQUE

B-titulo VARCHAR(30) UNIQUE;

2) CREATE VIEW respaldoalumnos AS SELECT * FROM alumnos WHERE f_nacimiento

BETWEEN 1990-01-01 AND 1993-12-31

3) DELETE FROM cursos WHERE id_cursos = 3;

EJERCICIO 3

VERDADERO O FALSO

1) Las claves foráneas de una tabla permiten establecer relaciones con otras tablas, puesto que contienen valores que encontramos como clave primaria en la tabla con la que se relaciona. (5p)

- a. V
- b. F

2) El campo o campos que forman una clave foránea nunca podrán contener valores nulos. (5p)

- a. V
- b. F

3) La instrucción TRUNCATE permite eliminar registros de una tabla, solo debemos indicar que registros deseamos eliminar mediante la cláusula WHERE. (5p)

- a. V
- b. F

4) Los índices -como los índices de los libros- sirven para agilizar las consultas a las tablas, evitando que el SGBD tenga que revisar todos los datos disponibles para devolver el resultado. (5p)

- a. V
- b. F

CLASE DEL 22/03

SQL AVANZADO

EMPEZAMOS CON UN REPASO pdf

Sistema de gestión de base de datos

¿Qué es un SGBD?

- Permite a los usuarios crear otras bases de datos y especificar su esquema por medio de algún lenguaje de definición
- Ofrece a los usuarios la capacidad de consultar los datos y modificarlos, usando para ello un lenguaje de consulta y manipulación.
- Brinde soporte al almacenamiento de cantidades voluminosas de datos durante un largo período, protegiéndolo contra accidentes o utilización no autorizada.
- Controle el acceso concurrente

SGBD: Sistema de gestión de base de datos, son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos y el usuario.

Acceso concurrente, buscar que significa.

Lenguajes de Bases de Datos

- Además, un SGBD debe tener un lenguaje con el que poder trabajar con él, el más conocido es el SQL (**Structured Query Language**) en él se integra un DDL y un DML.
- **Lenguajes de definición de datos (DDL):** Creación de esquemas, modificación de los mismos, etc.
- **Lenguaje de manipulación de datos (DML):** Creación, Modificación, Eliminación y Obtención de Datos.
- **Lenguaje de Control de Datos (DCL):** Estos comandos permiten al Administrador del sistema gestor de base de datos, controlar el acceso a los objetos, es decir, podemos otorgar o denegar permisos a uno o más roles para realizar determinadas tareas.

Si ami me dice, que el lenguaje sql se clasifica en 3 tipos, DCL gran y rebok no la emos dado.

Para un examen es necesario saber explicar las 3 y dar ejemplos de cada una.

Esquemas

- Las bases de datos en PostgreSQL se organizan mediante **esquemas, contenedores lógicos de objetos de base de datos** (tablas, vistas, procedimientos, etc.), básicamente son un espacio de nombres. Es característico de los esquemas:
- Tienen un propietario.
- Permiten el uso de la base de datos por múltiples usuarios sin interferencias.
- Permiten que se puedan instalar aplicaciones realizadas por terceros si que existan colisiones en los nombres de los objetos.
- Para poder ver objetos de un esquema, debe usarse la notación: **esquema.objeto**

NOTA:

- En esquema publico podemos Tenes una tabla cliente y en una con otro nombre también podemos tener una tabla cliente.

Secuencias

- Son contadores que se crean con:
CREATE SEQUENCE nombreseq;
- Se manipulan con:
 - nextval(): retorna un valor y lo incrementa
 - curval(): retorna un valor
 - setval(): establece un valor
- Se suelen usar en campos de clave primaria secuenciales
 - de este modo, creamos la tabla haciendo uso de una **secuencia que ya existe**:

```
CREATE TABLE cliente (
    ident INTEGER DEFAULT nextval('nombre_secuencia')
    [otros_campos]);
```

o bien, más cómodo, pero crea la tabla la secuencia (con nombre aleatorio):

```
CREATE TABLE cliente (ident serial,
    ...
);
```

Vistas

- Es una estructura lógica que permite visualizar un grupo de datos que provienen de una o varias tablas u otras vistas.
- No contiene datos propios, estos provienen de otras tablas con datos reales.
- No permite la inserción, actualización y eliminación de datos, solo la lectura.
- Su utilización es como el de una tabla, podemos usar cualquier sentencia de tipo SELECT sobre ellas.
- Prácticamente es una tabla virtual que proviene de la instrucción SELECT.

Una consulta que queda guardada con un nombre.

Vistas

Ventajas:

- Siempre se muestran los datos actualizados.
- Simplifica el uso de consultas complejas.
- Simplifica la representación de los datos ofreciendo mas sentido lógico.
- Define un nivel mas de seguridad.
- Aísla las aplicaciones de la Base de Datos.

Desventajas:

- No se pueden utilizar las sentencias INSERT, UPDATE y DELETE sobre la vista para alterar los datos.
- No mejora el rendimiento.

Ventajas y desventajas, eso puede ser una **pregunta de parcial**.

Un ejemplo de vista:

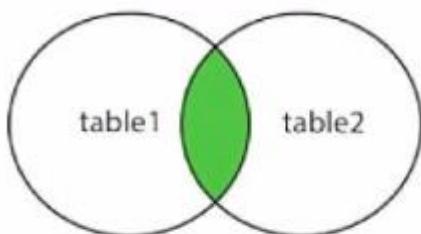
```
postgres=# CREATE VIEW profesores_departamentos AS
  SELECT profesor.cedula, profesor.nombre AS nombreProf,
         departamento.codigo, departamento.nombre AS nombreDpto
    FROM profesor, departamento
   WHERE profesor.codigodpto = departamento.codigo;
CREATE VIEW
postgres=# SELECT * FROM profesores_departamentos;
      cedula |      nombreprof |   codigo | nombreredpto
-----+-----+-----+-----+
 12489778 | Pedro Perez   |       1 | Computacion
 13449543 | Juan Rojas    |       1 | Computacion
 15669442 | Jose Fuentes   |       2 | Control
 11639337 | Miguel Redondo |       2 | Control
 10274448 | Andres Silva   |       3 | Investigacion
 12539445 | Luis Cardenas  |       3 | Investigacion
 14556333 | Hector Mora    |       1 | Computacion
 15889332 | Felipe Redondo |       3 | Investigacion
(8 rows)
```

Del manual de postgres: Currently, views are read only: the system will not allow an insert, update, or delete on a view.

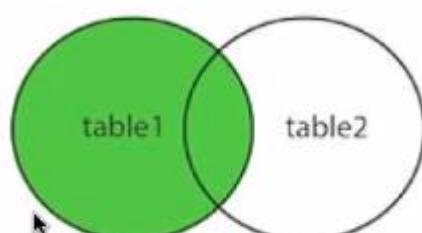
Vemos la diferencia, oculta la lógica de la consulta.

CONSULTAS DE REUNION JOIN

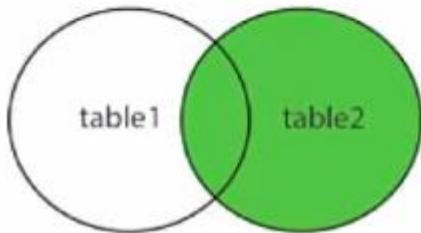
INNER JOIN



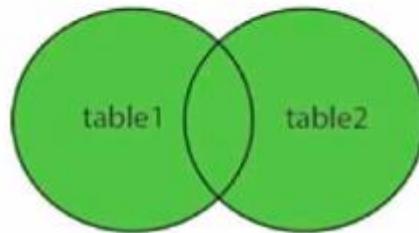
LEFT JOIN



RIGHT JOIN



FULL OUTER JOIN



EJEMPLO:

cedula	nombre	codigo	nombre
12889883	Juan Guerra	2	Control
11389998	Felipe Fernandez	3	Investigacion
14556334	Hector Loma	1	Computacion
14443895	Jose Fuentes		

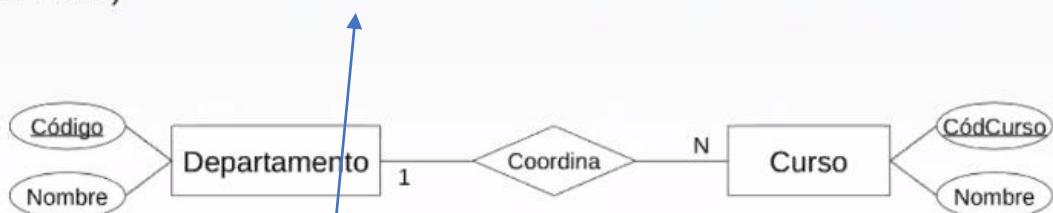
codigo	nombre	codigodpto
1	Prog. Digital 1	1
2	Prog. Digital 2	1
3	Control	2
4	Instrumentacion	2
5	Astrofisica	

Tenemos 3 tablas.

```
postgres=# SELECT *
  FROM departamento
  JOIN curso
    ON departamento.codigo = curso.codigoDpto;
```

codigo	nombre	codigo	nombre	codigodpto
1	Computacion	1	Prog. Digital 1	1
1	Computacion	2	Prog. Digital 2	1
2	Control	3	Control	2
2	Control	4	Instrumentacion	2

(4 rows)



¿Y el departamento de investigación? ¿Qué pasa si pregunto por los departamentos que **no** tienen cursos?

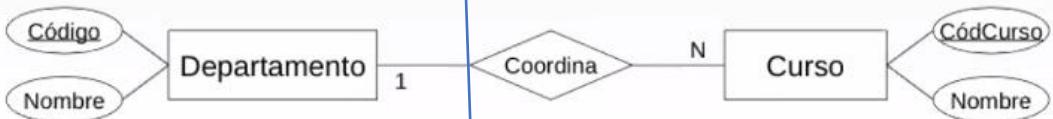
Astrofisica no tiene un código departamento.

Por eso no sale en resultado de la consulta.

Ahora si queremos la los departamento que no tiene cursos:

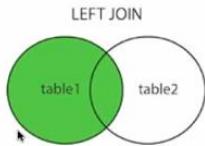
```
postgres=# SELECT *  
      FROM departamento  
      LEFT JOIN curso  
      ON departamento.codigo = curso.codigoDpto;
```

codigo	nombre	codigo	nombre	codigodpto
2	Control	4	Instrumentacion	2
2	Control	3	Control	2
3	Investigacion	NULL	NULL	NULL
1	Computacion	2	Prog. Digital 2	1
1	Computacion	1	Prog. Digital 1	1



¡Aquí está!

Ahora le agregamos el operador LEFT y aparece la tabla investigación.

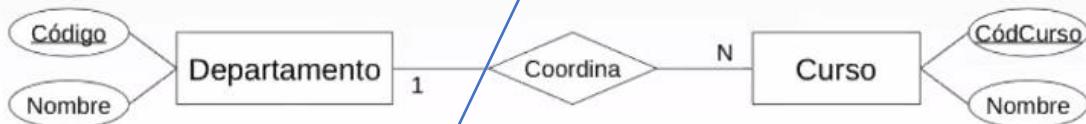


La tabla de la izquierda es la que esta en el FROM.

```
postgres=# SELECT departamento.codigo, departamento.nombre  
      FROM departamento  
      LEFT JOIN curso  
        ON departamento.codigo = curso.codigoDpto  
 WHERE curso.codigo IS NULL;
```

```
;  
codigo |     nombre  
-----+-----  
      3 | Investigacion  
(1 row)
```

IS NULL se usa para comparar atributos con valores nulos



¿Qué pasa si pregunto por los departamentos que *no* tienen cursos? (*¡La Respuesta!*)

Estamos evitando hacer una subconsulta.

RIGHT JOIN

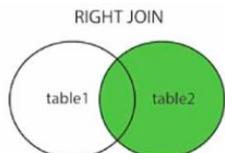
```
postgres=# SELECT *
  FROM curso
    RIGHT JOIN departamento
      ON curso.codigoDpto = departamento.codigo;
```

codigo	nombre	codigodpto	codigo	nombre
4	Instrumentacion		2	Control
3	Control		2	Control
NULL	NULL	NULL	3	Investigacion
2	Prog. Digital 2		1	Computacion
1	Prog. Digital 1		1	Computacion

(5 rows)



¡Aquí está! (Nuevamente)

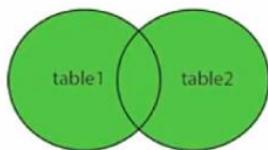


FULL JOIN

```
postgres=# SELECT *
  FROM curso
    FULL JOIN departamento
      ON curso.codigoDpto = departamento.codigo;
```

codigo	nombre	codigodpto	codigo	nombre
1	Prog. Digital 1	1	1	Computacion
2	Prog. Digital 2	1	1	Computacion
3	Control	2	2	Control
4	Instrumentacion	2	2	Control
5	Astrofisica	NULL	NULL	NULL
NULL	NULL	NULL	3	Investigacion

FULL OUTER JOIN



Admite NULOS.

LIMIT Y OFFSET

BESALCO SA	11/03/2021	e-Factura A 345040	Aceptado	UYU 1.500,00	
Administración Nacional de Telecomunicaciones	10/03/2021	e-Factura D 7271555	Aceptado	UYU 1.729,88	
Administración Nacional de Telecomunicaciones	10/03/2021	e-Factura D 7272805	Aceptado	UYU 496,63	
BESALCO SA	10/03/2021	e-Factura A 345009	Aceptado	UYU 1.000,00	
BESALCO SA	10/03/2021	e-Factura A 344966	Aceptado	UYU 1.000,00	
Casa Elio Ocampos SC	09/03/2021	e-Factura A 1326521	Aceptado	UYU 1.000,00	
Fabian J.C. y R. D. SRL	08/03/2021	e-Factura A 629534	Aceptado	UYU 1.000,10	
Fernández Orsi Sofía	08/03/2021	e-Factura A 566	Aceptado	UYU 24,00	
BESALCO SA	08/03/2021	e-Factura A 344809	Aceptado	UYU 1.899,90	
AUTOSERVICE LA SUPER GRANJA SRL	07/03/2021	e-Factura A 35116	Aceptado	UYU 1.073,00	

Mostrando registros del 21 al 30 de un total de 48 registros

Anterior 1 2 3 4 5 Siguiente

Muestra 10 datos y si quieres ver 10 mas tienes que darle siguiente o cambiar de página.

Si hacemos una consulta común nos tira los 500 registros, como hago para traer 10 registros por tan dadas:

```
postgres=# (SELECT cedula, nombre FROM estudiante)
UNION ALL
(SELECT cedula, nombre FROM profesor)
ORDER BY cedula;
cedula | nombre
+-----+
10274448 | Andres Silva
11389998 | Felipe Fernandez
11639337 | Miguel Redondo
12489778 | Pedro Perez
12539445 | Luis Cardenas
12889883 | Juan Guerra
13449543 | Juan Rojas
14443895 | Jose Fuentes
14556334 | Hector Loma
15669442 | Jose Fuentes
(10 rows)
```

1er Grupo de 3 registros

2do Grupo de 3 registros

3er Grupo de 3 registros

4to Grupo de 3 registros
(Pero sólo hay 1)

LIMIT y OFFSET se utilizan para limitar la cantidad de registros que se muestran, y desplazar la posición de inicio a partir de la cual se comienzan a mostrar los registros. Muy útil para **paginar** registros.

FUNCIONA ASI:

```

postgres=# (SELECT cedula, nombre FROM estudiante)
UNION ALL
(SELECT cedula, nombre FROM profesor)
ORDER BY cedula LIMIT 3 OFFSET 0;
cedula | nombre
-----+
10274448 | Andres Silva
11389998 | Felipe Fernandez
11639337 | Miguel Redondo

postgres=# (SELECT cedula, nombre FROM estudiante)
UNION ALL
(SELECT cedula, nombre FROM profesor)
ORDER BY cedula LIMIT 3 OFFSET 3;
cedula | nombre
-----+
12489778 | Pedro Perez
12539445 | Luis Cardenas
12889883 | Juan Guerra

```

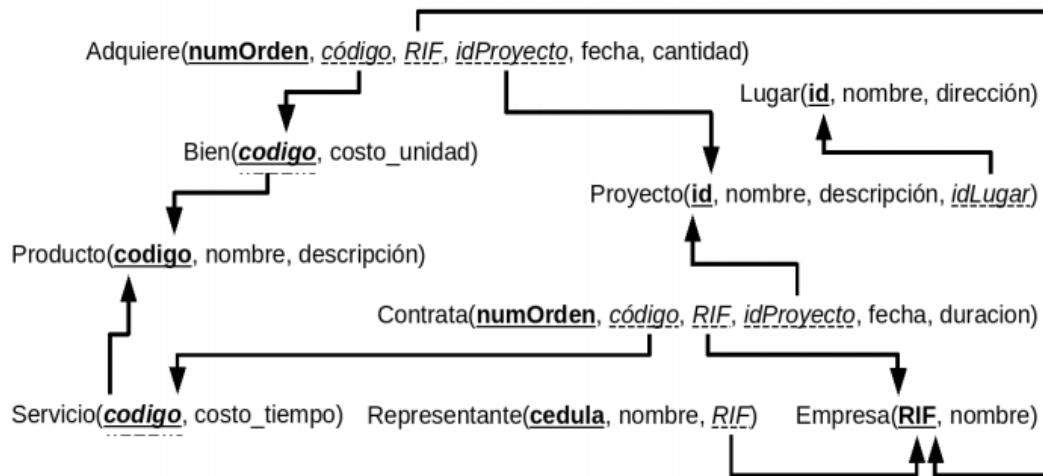
La cláusula ORDER BY es importante para predecir el orden en que aparecen los registros y mantener la consistencia de las distintas páginas

En la segunda consulta pide el segundo grupo de la imagen anterior.

PRACTICO

Compras y Contrataciones

(Asuma que la herencia entre producto (padre), bien y servicio (hijos) es **disjunta y total**)



SELECT

SELECT MAX(A.Phone) AS Phone

Se Pide:

1. Retornar una lista con el nombre de todos los productos adquiridos por una empresa particular. De la empresa se conoce su RIF y los productos deben salir sólo una vez en la lista.
2. Escribir una consulta que retorne el costo total que ha invertido una empresa (dada por su RIF) tanto en la contratación de servicios como en la adquisición de bienes. El costo de una adquisición es igual a la “cantidad adquirida” x “costo unidad” del bien, y el costo de una contratación es igual a la “duración contratación” x “costo tiempo” del bien. **Tip:** puede utilizar subconsultas.
3. Crear una vista denominada **ContratacionMasLarga** que al ejecutarla muestre para cada empresa (mostrando RIF y nombre), cual fue la duración de contratación más larga realizada a partir del 1ero de Enero del 2020.
4. En relación a las tablas Producto, Bien y Servicio, escriba una consultas, que debe probar que los datos existentes en las tablas cumplen con la restricción de que la herencia sea disjunta.
5. En relación a las tablas Producto, Bien y Servicio, se escribe una consulta para probar que cumplen con la restricción de que la herencia sea total. La misma es:

```
SELECT COUNT(*) FROM (
    SELECT producto.codigo
    FROM producto LEFT JOIN bien
    ON producto.codigo = bien.codigo
    WHERE bien.codigo IS NULL
)
INTERSECT
(SELECT producto.codigo
    FROM producto LEFT JOIN serv
```

```
    ON producto.codigo = serv.codigo
    WHERE serv.codigo IS NULL
) AS x
```

- a) ¿Cuál es la función del operador **INTERSECT** en SQL?
- b) ¿Qué es lo que hace la consulta? ¿Cómo demuestra a través de la misma que la herencia es total?

2- `(select e.empresa, SUM(b.costo * a.cantidad) as total from empresa as e, bien as b)`
`union all (select e.empresa, SUM(s.costo_tiempo * c.duracion) as total from empresa`
`as e, servicio as s) join adquiere as a on a.codigo = b.codigo join contrata as c on`
`c.codigo= s.codigo order by e.empresa group by e.empresa`

3- CREATE VIEW ContratacionMasLarga as SELECT e.RIF, e.nombre,
MAX(c.duracion) as Duración FROM empresa as e JOIN contrata as c ON e.RIF =
c.RIF WHERE fecha > '2020-01-01';

CORRECCION

El operador INTERSECT SQL se utiliza para devolver los resultados de 2 o más instrucciones SELECT. Sin embargo, sólo devuelve las filas seleccionadas por todas las consultas o conjuntos de datos. Si un registro existe en una consulta y no en el otro, se omite de los resultados se cruzan.

Operación de conjunto que devuelve filas que provienen de dos expresiones de consulta. Las filas que no se devuelven en las dos expresiones se descartan.

Total es cuando todo elemento de la super entidad debe estar al menos en una.
 Puede estar a lo sumo en una cuando es disyunta.

productos		bien		servicio	
id	nombre	codigo	costo_unidad	codigo	costo_tiempo
1	computadora	1	8000	3	200
2	celular	2	12000	4	350
3	plomería				
4	albañil				

subconsulta 1		suboncsulta 2			
producto.id	bien.codigo	producto.id	servicio.codigo		
1	1		1 NULL		
2	2		2 NULL		
3 NULL			3	3	
4 NULL			4	4	

SELECT producto.codigo FROM producto LEFT JOIN bien ON producto.codigo = bien.codigo WHERE bien.codigo IS NULL	INTERSECT	SELECT producto.codigo FROM producto LEFT JOIN serv ON producto.codigo = serv.codigo WHERE serv.codigo IS NULL	
-------------------------------------------------------------------------------------------------------------------------	-----------	-------------------------------------------------------------------------------------------------------------------------	--

subconsulta 1		suboncsulta 2			
producto.id	bien.codigo	producto.id	servicio.codigo		
1	1		1 NULL		
2	2		2 NULL		
3 NULL			3	3	
4 NULL			4	4	

ME DEBERIA DAR 0 EL INTERSET

solucion del 4

- En relación a las tablas Producto, Bien y Servicio, escriba una consulta que devuelva los datos existentes en las tablas cumplen con la restricción de que la herencia es disyunta.

SELECT COUNT(*)

FROM bien, serv

WHERE bien.codigo = serv.codigo

(Si el resultado es 0 entonces es disyunta, en caso contrario no lo es).

solución del 2

```

SELECT SUM(costo)
FROM (
    (SELECT SUM(adquiere.cantidad * bien.costo_unidad) AS costo
     FROM adquiere, bien
     WHERE adquiere.codigo = bien.codigo AND
           RIF = 'XXX')
UNION
    (SELECT SUM(contrata.duracion * serv.costo_tiempo) AS costo
     FROM contrata, serv
     WHERE contrata.codigo = serv.codigo AND
           RIF = 'XXX')
) AS x

```

hacer ejercicio el que descarge sobre join

MIERCOLES 24/3

CORRECCION DEL PRACTICO 5C

```

1.Obtener los nombres de los proyectos a los cuales suministra partes el proveedor S1 utilizando EXISTS en la solución.

SELECT j.nombre FROM j
WHERE exists (
SELECT * FROM spj
WHERE spj.j_j = j.j AND spj.s_s = 'S1'
);

2.Obtener los números de los proyectos donde se utilice al menos una de las partes suministradas por el proveedor S1, utilizando NOT EXISTS en la solución.

SELECT j.j FROM j
WHERE exists (
SELECT * FROM spj
WHERE spj.j_j = j.j AND spj.s_s = 'S1'
);

3.Obtener los números de los proyectos para los cuales S1 es el único proveedor, utilizando EXISTS en la solución.

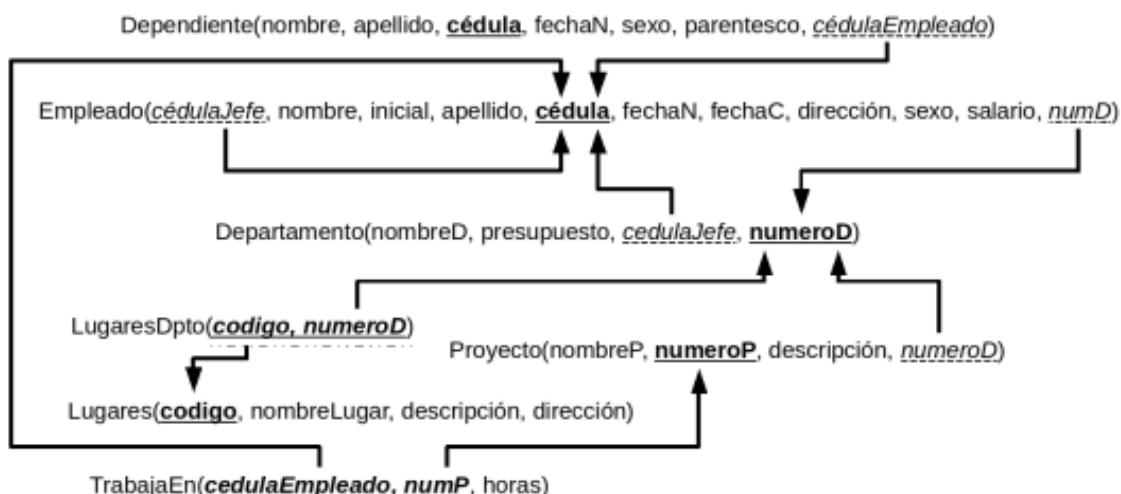
SELECT distinct j.j FROM j
WHERE NOT exists (
SELECT * FROM spj
WHERE spj.j_j = j.j AND spj.s_s <> 'S1'
);

```

TAREA EN CLASE

Empresa y Empleados

Dado el siguiente esquema relacional:



Se Pide:

1. Genere las instrucciones DDL correspondientes en SQL para crear la tabla "**Empleado**" (con todas las claves), asumiendo que el salario de un empleado debe estar comprendido entre 20.000 y 100.000 pesos, que los nombres de los empleados y de los departamentos no pueden ser nulos y que la fecha de nacimiento de un empleado tiene que ser anterior a su fecha de contratación.
2. Escriba una consulta SQL que permita obtener el nombre y el número de todos los departamentos con presupuesto menor de 500.000 pesos, ordenador por nombre ascendente.
3. Sin utilizar sub-consultas genere una consulta en SQL que devuelva una lista de los proyectos en los que aún no trabajan empleados. Para cada proyecto muestre el nombre del proyecto, el nombre del departamento al que pertenece y la cédula y el nombre del jefe del departamento al que pertenece.
4. Genere una consulta en SQL, para saber la cantidad de departamentos que están distribuidos en 1 lugar, la cantidad que están distribuidos en 2 lugares y así sucesivamente. Por ejemplo, en el caso de que el Dpto. de IT tenga sedes en Mérida y en Maracaibo (2 lugares), el Dpto. de Ventas tenga sedes en Mérida y Caracas (también 2 lugares), el Dpto. de Mercadeo tenga sedes en Caracas, Valencia y Maracaibo (3 lugares) y la Gerencia General tenga sede sólo en Caracas, el resultado será:

lugares		cant. de dptos
1		1
2		2
3		1

Tip: puede utilizar subconsultas.

5. Crear una vista **ProyectosPorDepartamento** que genere una lista que muestre el numero, el nombre, la cédula del jefe y la cantidad de proyectos asociados a cada departamento.

PARTE 1

```
CREATE TABLE Empleado(
    cedulaJefe INT NOT NULL,
    nombre VARCHAR(50) NOT NULL,
    inicial VARCHAR(5),
    apellido VARCHAR(50),
    cedula INT PRIMARY KEY NOT NULL,
    fechaN DATE,
    fechaC DATE,
    direccion VARCHAR(50),
    sexo VARCHAR(10),
    salario FLOAT CHECK (salario BETWEEN 20.000 AND 100.000),
    numD INT NOT NULL,
    FOREIGN KEY (cedulaJefe) REFERENCES Empleado(cedula),
    FOREIGN KEY (numD) REFERENCES Departamento(numeroD),
    CHECK(fechaN < FechaC)
);
```

UN EJEMPLO

```
CREATE TABLE empleado(
    cedula int PRIMARY KEY,
    nombre varchar(20) not null,
    apellido varchar(20) not null,
    inicial int,
    fechaNac date,
    fechaC date,
    direccion varchar(50),
    sexo char,
    salario numeric,
    numD int not null,
    CHECK (fechaNac < fechaC),
    CHECK (salario BETWEEN 20000 AND 100000)
--FOREIGN KEY (numD) REFERENCES departamento(numeroD),
```

parte 3

```
SELECT p.numeroP,p.nombreP,d.nombreD,d.cedulaJefe,e.nombre
|FROM Proyecto AS p
|JOIN Departamento AS d ON p.numeroD = d.numeroD
|JOIN Empleados AS e ON d.cedulaJefe = e.cedula
|LEFT JOIN TrabajaEn AS t ON p.numeroP = t.numP
|WHERE t.cedulaEmpleado IS NULL
```

Parte 4

```
SELECT lugares, COUNT(*) FROM  
  
(SELECT nombreD, count(*) AS lugares  
  
FROM departamento, lugaresDpto  
  
WHERE departamento.numeroD = lugaresDpto.numeroD  
  
GROUP BY departamento.numeroD) AS x  
  
GROUP BY lugares  
  
ORDER BY lugares|
```

VOLVEMOS DE SEMANA DE TURISMO

Computación en la Nube

Servicios conocidos: amazonAw

Ayur

Google

Al final de la clase todos vamos a tener una instancia en la nube.

Nos vamos a crear una cuenta de estudiante, creamos una instancia de servidor posgrett.

TAREA

-Hacer los pasos y tener un servidor

-Hacer ejercicio

9. En la instancia de servidor creada anteriormente realizar las siguientes tareas:
 - a. Crear un usuario denominado "utec_admin" con password "2020utec".
 - b. Crear un usuario denominado "utec_op" con password "2020op".
 - c. Crear una base de datos "practico_utec" y el dueño de la base de datos debe ser el usuario "utec_admin".
 - d. Crear en la base de datos "practico_utec" un esquema "laboratorio".
10. Las instrucciones para continuar con el práctico se encuentran en el [siguiente link](#) de acuerdo a su grupo.
 - a. **Debe realizar solamente las que corresponden a su grupo y no otro.**
 - b. Todos los puntos deben ser resueltos escribiendo las sentencias SQL y registrando la misma en el documento de la letra correspondiente a su grupo.
11. Al finalizar debe subir a la tarea de EVA el documento correspondiente a su grupo e incluir el acceso a su instancia de servidor (nombre o ip + puerto).

Tareas Grupo 01

ATENCIÓN: GUARDE TODAS LAS SENTENCIAS SQL DE TODOS LOS PUNTOS YA QUE DEBERÁ ENTREGARLAS EN EVA.

1. Crear una tabla “**personas**” en el esquema “**laboratorio** ”con los campos:
 - a. **ID**: como PK y autoincremental
 - b. **Nombre**: no puede tener valores duplicados.
2. Insertar 5 registros en la tabla creada anteriormente.
3. Otorgar privilegios totales para el usuario “**utec_admin**” sobre esta tabla.
4. Otorgar privilegios INSERT y UPDATE para el usuario “**utec_op**” sobre esta tabla.
5. Crear una vista **v_personas** que muestre solamente el nombre de las personas de la tabla ordenado alfabéticamente.
6. Otorgar privilegios sobre la vista al usuario “**utec_op**” para que la pueda ejecutar.

```
-- DROP table laboratorio.personas;
CREATE TABLE laboratorio.personas(
    ident serial PRIMARY KEY,
    nombre varchar(50) UNIQUE
);

insert into laboratorio.personas values ('Leandro');
insert into laboratorio.personas values ('Nicolas');
insert into laboratorio.personas values ('Cintia');
insert into laboratorio.personas values ('a');
insert into laboratorio.personas values ('l');
```

```
CREATE VIEW v_personas AS(
    SELECT laboratorio.personas.nombre FROM laboratorio.personas ORDER BY nombre ASC
)
GRANT ALL ON laboratorio.personas TO utec_admin
```

Waiting for the query to complete...

```
GRANT INSERT, UPDATE ON laboratorio.personas TO utec_op
```

Output Explain Notifications Messages

La seguridad informática consiste en asegurar que los **recursos de sistemas y de información** (material informático o programas, bases de datos, etc.) de una organización sean **utilizados de la manera en que se decidió** y que el acceso a la información allí contenida, así como su modificación, **sólo sea posible a las personas que se encuentren acreditadas y dentro de los límites de su autorización**

Físico:

Los servidores (o nodos de cómputo) donde está el SGBD deben estar protegidos frente al acceso de extraños

Humano:

El personal encargado de los servidores debe ser calificado y de confianza (sobornos / ingeniería social)

Red:

La red en la que opera el SGBD debe tener las protecciones correspondientes, protección en el envío de datos, firewalls, cifrado, entre otras

Sistema Operativo:

El sistema operativo debe estar actualizado y se deben realizar todos los esfuerzos necesarios para que no sea vulnerable

Software / Aplicación:

Las aplicaciones cliente de los SGBD deben ser **diseñadas y desarrolladas** con los niveles de seguridad adecuados

SGBD:

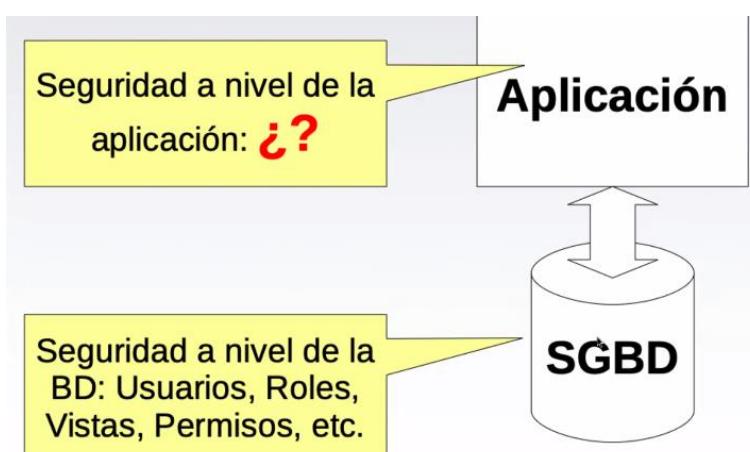
La base de datos debe ser configurada con roles, privilegios y permisos adecuados para evitar los accesos malintencionados

¿Dónde implementar la seguridad?

**¿Seguridad a nivel del SGBD o
seguridad a nivel de la Aplicación?**

**Recuerden el dilema:
¿Validar a nivel del SGBD, a nivel de la
aplicación, a nivel del cliente?**

1-Se refiere vamos a decir que hace un programa de facturación usado por 30 usuarios, seguridad en la base de datos o en la aplicación.



NOTA: sqlinjection buscar que es.

Manejo la seguridad de la aplicación.

En los SGBD, el concepto de seguridad se refiere a la **protección de los datos** ante **usuarios no autorizados**, es decir, definir estrategias que permitan establecer que usuarios pueden acceder a qué datos

- Tipos de seguridad en los SGBD
 - Seguridad Discrecional
 - Seguridad Obligatoria
 - Seguridad en Sistemas Estadísticos

(¿Qué creen que sea esto?)

Si un usuario puede ser una persona o un dispositivo de hardware, no tiene porque acceder a información que no sea relevante. Le damos información a lo juntos y necesario.

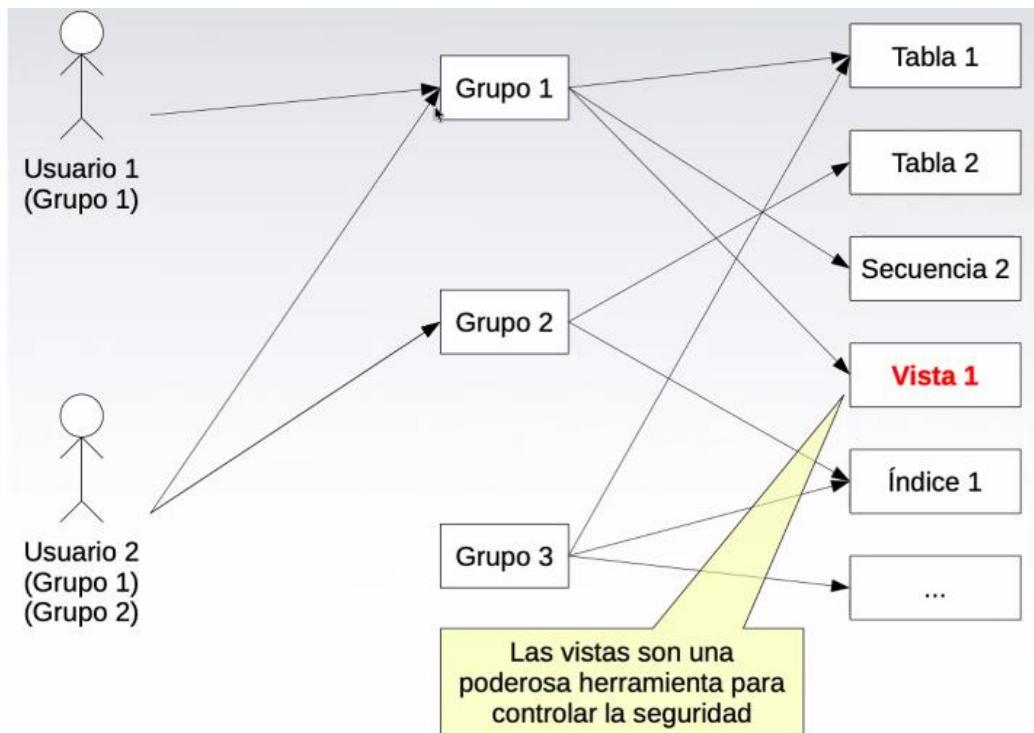
Se basa en otorgar privilegios a usuarios (o grupos de usuarios), en los que se incluye la capacidad de tener acceso tablas, registros o campos específicos con un determinado modo (para leer, insertar o actualizar)

- Autorizar al usuario X a realizar consultas en filas de la tabla A
- Autorizar al usuario X a utilizar un procedimiento almacenado B

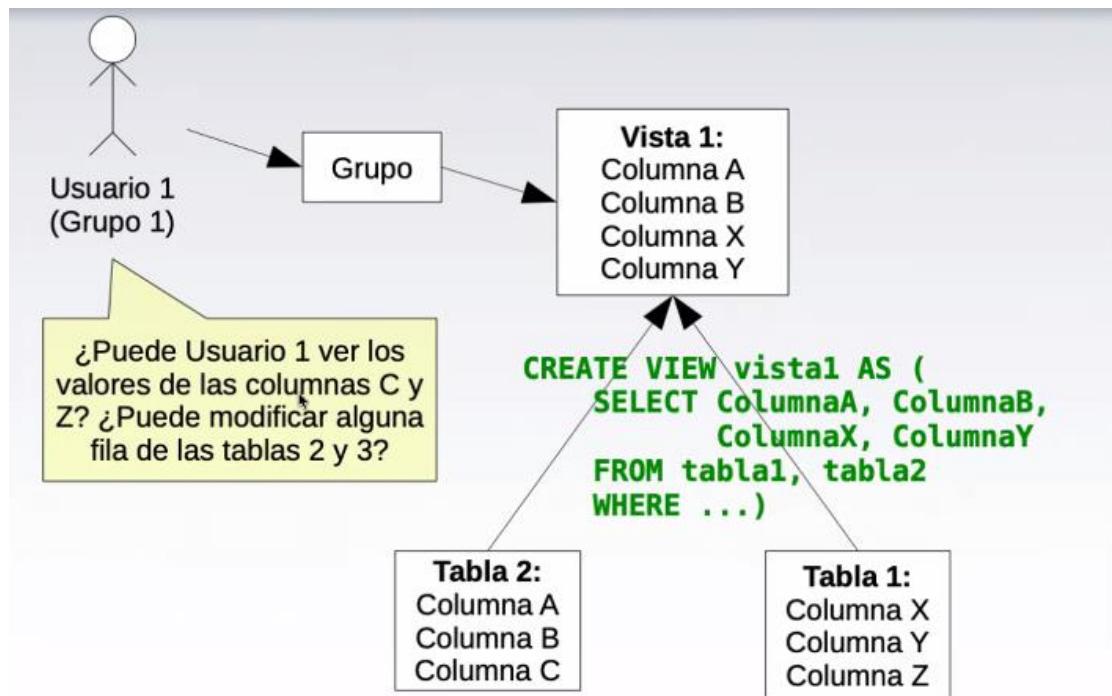
Un programa que se ejecuta dentro de la base de datos



SEGURIDAD DISCRECIONAL



Si tengo 200 empleados y necesito restringir accesos, lo mejor es crear grupos y a los usuarios asignarle un grupo.



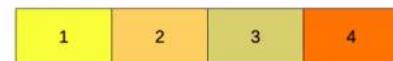
¿Cómo evitan que Usuario 1 vea ciertas filas específicas?

Creo una vista con lo que necesita, no le doy acceso a todo. solo le dejo ver al grupo lo que necesita ver. Ya que en una vista solo puede ver, no puede borrar, agregar ni nada. Con eso nos ahorraremos problemas de seguridad.

SEGURIDAD OBLIGATORIA

Consiste en imponer seguridad de múltiples niveles, clasificando los datos y los usuarios en varias clases (o niveles) de seguridad, de manera que los usuarios puedan acceder a los datos según tengan o no el nivel necesario para el dato que desean acceder

- Las filas (o los objetos) tienen un nivel F_i de seguridad, que solo se pueden leer si el usuario tiene un nivel $U_i \geq F_i$ de seguridad...



Clasifica la información, confidencial, secreta y super secreta.

Si tenemos acceso a información super secreta para abajo podemos ver toda la otra información.

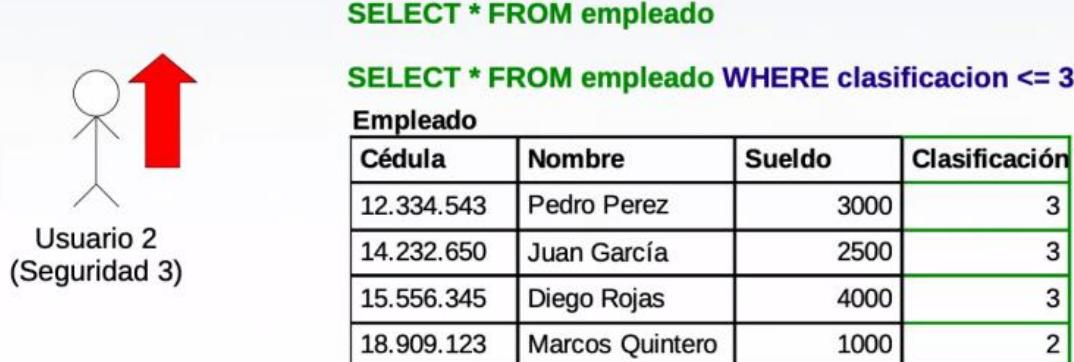
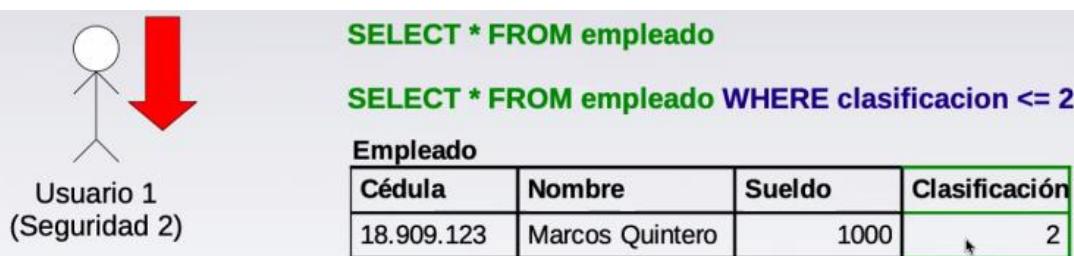
- El usuario **X** puede recuperar (leer) el objeto **Y** sólo si el nivel de acreditación (seguridad) de **X** (**Ui**) es mayor o igual que el nivel de clasificación de **Y** (**Fi**) (“propiedad de seguridad simple”)
- El usuario **X** puede actualizar el objeto **Y** sólo si el nivel de acreditación de **X** (**Ui**) es **igual** al nivel de clasificación de **Y** (**Fi**) (“propiedad estrella”). **¿Ideas de por qué esto es así?**

Vamos a suponer que tengo (Daniel) un nivel de seguridad super secreto. Y romina tiene secreto, y franco también el nivel secreto. Entonces esto nos dice que, Daniel puede ver las cosas de romina y franco pero no puede cambiarlo, pero porque no puede cambiarlo? Para evitar fuga de información, por eso no pude cambiar. Daniel solo puede hacer cambios en su nivel.

La segunda regla anterior evita que existan filtraciones de seguridad hacia abajo, o que un usuario escriba datos que luego no pueda leer... (hacia arriba)

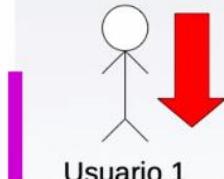


Clave primaria Cedula.



```
INSERT INTO empleado
VALUES(16.343.222, 'Luis Silva', 2000)
```

```
INSERT INTO empleado
VALUES(16.343.222, 'Luis Silva', 2000, 2)
```



Empleado			
Cédula	Nombre	Sueldo	Clasificación
12.334.543	Pedro Perez	3000	3
14.232.650	Juan García	2500	3
15.556.345	Diego Rojas	4000	3
16.343.222	Luis Silva	2000	2
16.343.222	Luis Silva	10000	4
18.909.123	Marcos Quintero	1000	2

En este modelo, el campo clasificación también cuenta como clave primaria.



```
SELECT * FROM empleado
SELECT * FROM empleado WHERE clasificacion <= 2
```

Empleado			
Cédula	Nombre	Sueldo	Clasificación
18.909.123	Marcos Quintero	1000	2
16.343.222	Luis Silva	2000	2



```
SELECT * FROM empleado
SELECT * FROM empleado WHERE clasificacion <= 4
```

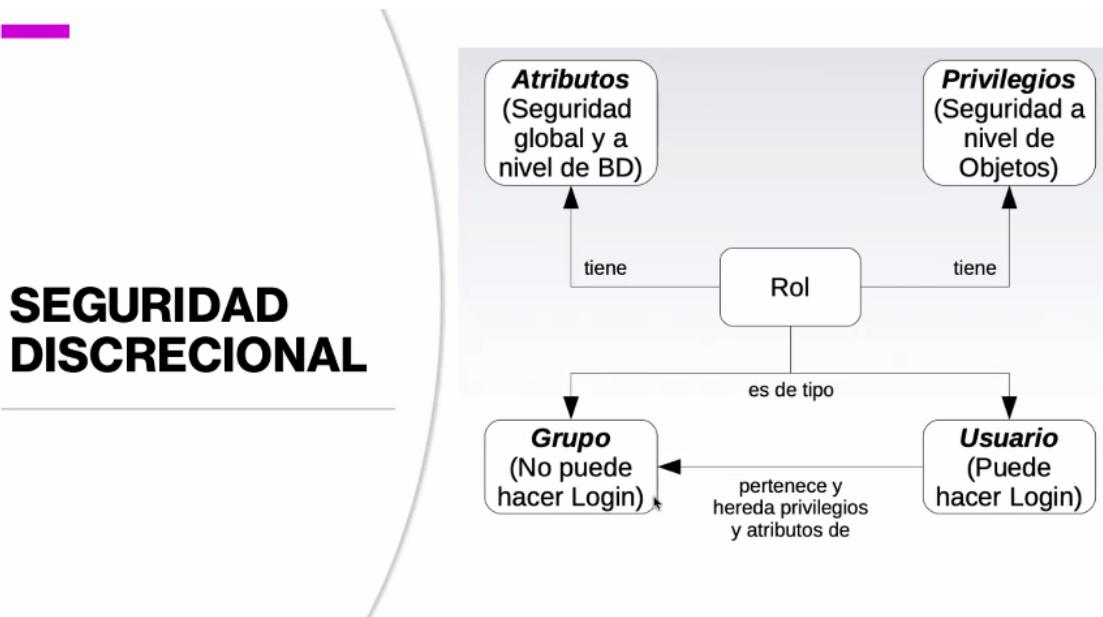
Empleado			
Cédula	Nombre	Sueldo	Clasificación
12.334.543	Pedro Perez	3000	3
14.232.650	Juan García	2500	3
15.556.345	Diego Rojas	4000	3
16.343.222	Luis Silva	2000	2
16.343.222	Luis Silva	10000	4
18.909.123	Marcos Quintero	1000	2

Repetición de datos, cuando pasa esto se toma el valor del de información que tenga clasificación más alta.

SEGURIDAD DISCRECIONAL

PostgreSQL manages database access permissions using the concept of roles. A role can be thought of as **either a database user, or a group of database users**, depending on how the role is set up. **Roles can own database objects** (for example, tables) and can assign privileges on those objects to other roles to control who has access to which objects. Furthermore, it is possible to grant membership in a role to another role, thus allowing the member role use of privileges assigned to the role it is a member of.

FUENTE: <http://www.postgresql.org/docs/8.4/interactive/user-manag.html>



Vamos a ver en la practica con que sentencias sql se hace.

Aplicado a posgret

Los roles de la BD son “usuarios”, son globales, es decir, no existen por cada base de datos

CREATE ROLE nombre_rol;

DROP ROLE nombre_rol;

SELECT * FROM pg_roles;

rolname	super	inherit	createrole	createdb	catupdate	canlogin	connlimit	pass	...
postgres	t	t	t	t	t	t		-1	**** ...

Los roles de la BD son “usuarios”, son globales, es decir, no existen por cada base de datos

CREATE ROLE nombre_rol;

DROP ROLE nombre_rol;

SELECT * FROM pg_roles;

rolname	super	inherit	createrole	createdb	catupdate	canlogin	connlimit	pass	...
postgres	t	t	t	t	t	t		-1	**** ...

Hace una consulta a una tabla que tiene los roles creados.

- **LOGIN**: La posibilidad de hacer login
- **SUPERUSER**: Permisos de superusuario (usar con cuidado)
- **CREATEDB**: La posibilidad de crear bases de datos
- **CREATEROLE**: La posibilidad de crear otros roles
- **PASSWORD 'string'**: Para asignar una contraseña (String es el password)

Cuando un “objeto” (tabla, secuencia, índice, etc) es creado usualmente se le asigna un dueño. Por defecto, el dueño es el rol (usuario) que crea el objeto. Es decir, para la mayor parte de los objetos, inicialmente, sólo el dueño y el superusuario pueden hacer algo con el objeto

Para permitir a otros usuarios hacer algo con esos objetos es necesario asignar privilegios, de los cuales hay de distintos tipos:

(Lamina siguiente)

***SELECT, INSERT, UPDATE, DELETE,
REFERENCES, TRIGGER, CREATE, CONNECT,
TEMPORARY, EXECUTE, USAGE***

Los privilegios se asignan con el comando
GRANT:

GRANT UPDATE ON tabla TO rol;

y se eliminan con REVOKE:

REVOKE ALL ON tabla FROM rol;

Leer teorico. Seguridad y autorización.

CREATE USER

Sintaxis

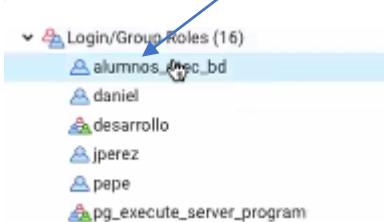
CREATE USER user_name [WITH PASSWORD 'password_value'];

Ejemplos:

- ***CREATE USER techonthenet;***
- ***CREATE USER techonthenet WITH PASSWORD 'fantastic';***

Si ejecutamos en el pgAdmin la sentencia sql, aparece.

`CREATE USER alumnos_utec_bd WITH PASSWORD 'fantastic';`



Ahora creamos un grupo.

```
create group estudiantes;
```

The screenshot shows the pgAdmin interface. A blue arrow points from the text 'create group estudiantes;' to the 'estudiantes' entry in the 'Login/Group Roles (7)' list. The list includes: alumnos_utec_db, daniel, desarrollo, estudiantes (highlighted by the arrow), jperez, pepe, and pg_execute_server_program.

```
create group estudiantes;
```

- ▼ Login/Group Roles (7)
 - alumnos_utec_db
 - daniel
 - desarrollo
 - estudiantes
 - jperez
 - pepe
 - pg_execute_server_program

La sintaxis completa para el comando CREATE USER es:

```
CREATE USER nombre_usuario
[ [WITH] option]...
option := SYSID user-id-number
| [NO] CREATEDB
| [NO] CREATEUSER
| IN GROUP groupname[,...]
| [ [UN] ENCRYPTED ] PASSWORD 'password'
| VALID UNTIL 'expiration'
```

CAMBIAR PASSWORD USUARIO

Sintaxis



```
ALTER USER user_name WITH PASSWORD 'new_password';
```

Ejemplos:

- **ALTER USER techonthenet WITH PASSWORD 'fantastic';**

OTROS

ALTER USER user_name RENAME TO new_name;	Cambiar nombre de usuario
DROP USER user_name;	Eliminar usuario
SELECT DISTINCT username FROM pg_stat_activity;	Usuarios logueados en posgres
SELECT username FROM pg_user;	Ver Usuarios de posgres

Ahora para ver quien esta logueado en la base de datos en ese momento.

```
SELECT DISTINCT username FROM pg_stat_activity;
```

Para ver usuarios logueados

```
0
7   SELECT username
8   FROM pg_user;
```

The screenshot shows a PostgreSQL terminal window. At the top, there is a command-line interface with the following text:

```
0
7   SELECT username
8   FROM pg_user;
```

Below the command line, there is a table titled "Data Output". The table has two columns: "username" and "name". The data rows are:

	username	name
1	postgres	
2	daniel	
3	jperez	
4	pepe	
5	alumnos_utec..	

En posgret existe un catálogo de sistema:

CATÁLOGO DEL SISTEMA

- Contiene una descripción completa de la estructura de la base de datos y sus restricciones.
- La información almacenada en el catalogo se denomina meta-datos.
- Su principal utilidad es la de conocer que datos existen sin acceder a ellos.

CATÁLOGO DEL SISTEMA DE POSTGRES

- Todos los catálogos del sistema tienen un nombre que empieza por pg....

Nombre del Catálogo	Descripción
pg_database	base de datos en el sistema
pg_class	contiene las tablas, indices (primarios), secuencias, vistas ("relaciones")
pg_attribute	atributos de cada "clase" que aparece en pg_class
pg_index	indices secundarios
pg_proc	procedimientos existentes para programar funciones, etc.
pg_type	tipos de datos usados: base y compuesto
pg_operator	operadores (+ - * / < > = ~ ! @ # % ^ & ` ?) -> y nuevos tipos!!!!
pg_aggregate	Almacena "funciones de agregación". Son funciones que operan sobre un conjunto de valores, por ejemplo: sum, count y max.
pg_am	que "método de acceso por indices" existen
pg_authid	contiene información sobre identificadores de autorización de acceso a la base de datos (roles).

- Más sobre catálogo de Postgres: <http://www.postgresql.org/docs/8.2/static/catalogs-overview.html>

Si yo ejecuto en el pgAdmin, nos devuelve todas las bases de datos que tenemos en el servidos.

```
select * from pg_database;
```

	oid	datname	datdba	encoding	datcollate	datctype	datistemplate	dataallowconn	datconnlimit	datas
	oid	name	oid	integer	name	name	boolean	boolean	integer	oid
2	1	template1	10	6	C	C	true	true	-1	
3	13634	template0	10	6	C	C	true	false	-1	
4	33818	world	10	6	C	C	false	true	-1	
5	34353	peraza_corta..	10	6	C	C	false	true	-1	
6	34776	marraga_galareto	10	6	C	C	false	true	-1	
7	34951	lacava_castro	10	6	C	C	false	true	-1	
8	35083	cuesta_lopez	10	6	C	C	false	true	-1	
9	35251	olivera	10	6	C	C	false	true	-1	
10	35430	clase_ddl	10	6	C	C	false	true	-1	
11	35541	prueba01	10	6	C	C	false	true	-1	
12	35673	clase_ddl_pa..	10	6	C	C	false	true	-1	
13	35740	automóviles	10	6	C	C	Successfully run. Total query runtime: 341 msec. 13 rows affected.			

BASES DE DATOS 2

GRANT / REVOKE (PRIVILEGIOS)

GRANT

Sintaxis:

GRANT privileges ON object TO user;

Ejemplos:

- **GRANT SELECT, INSERT, UPDATE, DELETE ON products TO techonthenet;**
- **GRANT ALL ON products TO techonthenet;**
- **GRANT SELECT ON products TO PUBLIC;** ————— Todos los usuarios

REVOKE

Sintaxis:

REVOKE privileges ON object FROM user;

- **REVOKE DELETE, UPDATE ON products FROM techonthenet;**
- **REVOKE ALL ON products FROM techonthenet;**
- **REVOKE SELECT ON products FROM PUBLIC;**

TEORICOS POR LEER SOBRE ADMINISTRACIÓN

PRIMERO []

SEGUNDO []

TERCERO []

CUARTO []

QUINTO []

TARDE DE PRACTICOS

PRACTICO 12

PRÁCTICO 12

Archivos necesarios para la práctica:

- Academia.sql
- Academia_Datos.sql

Problemática que resolver

El ejercicio consiste en un proyecto que describe el problema de una empresa dedicada a la prestación de servicios educativos.

En "Academia", se ofrecen dos tipos de cursos en el periodo especial de verano, en que se imparten cursos de verano y cursos extracurriculares. Los primeros son materias que un alumno regular que estudia una carrera cursa en este periodo, se le permite adelantar hasta dos materias; mientras que los segundos son cursos especiales de capacitación que se ofrecen a alumnos regulares como estudiantes o profesionistas externos.

Los docentes de la "Academia", son los únicos a los que se les permite impartir estos cursos, por los cuales reciben un pago adicional, se les paga de acuerdo con un tabulador que indica el costo de la hora de estos cursos de acuerdo con el nivel académico del docente. El pago se genera a partir del alta del curso y sólo se permite expedir un cheque por cada curso. Además, los estudiantes deben acudir a pagar adicionalmente al costo del semestre por asistir a ellos.

"Academia" tiene dos departamentos que intervienen en la administración de los cursos:

A) Departamento de Administración (DA) y B) Departamento de Control Escolar (DCE). Corresponde al DA, efectuar el pago a los docentes y los cobros a los alumnos. El DA es dirigido por el C.P. Ávila y es auxiliado por el Sr. Cancino. Mientras que el DCE, es dirigido por el Lic. Barroso y auxiliado por los Sras. Tirado, Martínez, Aquino y Ramos y es en éste donde se decide qué cursos se imparten en el periodo, quién los imparte, y se aceptan las solicitudes de los alumnos. Un caso especial, es el de los Profesores, ya que el DA es quién les puede modificar el sueldo quincenal, mientras que el DCE ni siquiera puede visualizar este. Lo curioso es que el DCE es quien acepta los docentes y los registra en el sistema, pero es el DA donde se captura el sueldo. Importante es para la administración de la "Academia" que esta política se aplique al pie de la letra, y que sea implementado directamente sobre la DB.

Crear una base de datos denominada "Academia", crear las tablas e insertar los datos.

Administración de usuarios

Resolviendo el problema de la administración de una academia. Se han identificado dos grupos de usuarios en este sistema: **Administración y Escolar**. Dentro de cada uno de estos grupos encontramos a los usuarios, los que se clasifican así:

Administración:

- Jefe del departamento: C. P. Ávila
- Auxiliar de Administración: Sr. Cancino

Escolar:

- Jefe del departamento: Lic. Barroso
- Auxiliar Escolar: Sra. Tirado
- Auxiliar Escolar: Sra. Martínez

-
- Auxiliar Escolar: Sra. Aquino
 - Auxiliar Escolar: Sra. Ramos

Se Pide:

1. Crear dos grupos **Admin** para la DA y **Escolar** para el DCE.
2. Crear todos los usuarios asociados a cada grupo, con la contraseña igual al nombre del grupo por defecto. Utilizar como nombre de usuario el apellido de cada persona.
3. Al grupo "Administración" la dirección de la academia la ha otorgado acceso a las siguientes tablas:
 - a. CuentaCheques
 - b. Cheque
 - c. Tabulador
 - d. Profesores
 - e. Concepto
 - f. Recibo
 - g. DetalleRecibo

Como casos especiales este departamento podrá acceder a consultar las tablas de Cursos Especiales, Cursos Especiales Verano, Cursos Especiales Extracurriculares, Cursos Extracurriculares y Materias. Explicitamente no se les permite modificar ningún campo o registro.

4. Tablas a las que se le permite el acceso al grupo Escolar:
 - a. CursosEspeciales
 - b. CursosExtracurricular
 - c. Materias
 - d. CEVerano
 - e. CEEExtracurricula
 - f. Alumnos
 - g. Bimestre
 - h. Faltas
 - i. CalendarioEscolar
5. Caso especial de profesores para el grupo Escolar: La forma en que evitaremos que el grupo escolar vea el sueldo del docente es creando una vista y asignando privilegios por separado.
6. Conectado a la BD "Academia" con el usuario **barroso** e intente los siguientes comandos:
 - a. Select * from Profesores;
 - b. Select * from VistaProfesoresEscolar;
 - c. Select * from CuentaCheques;
 - d. Select * from Cheque;
7. Conectado a la BD "Academia" con el usuario **avila**, intente los siguientes comandos:
 - a. Select * from Profesores;
 - b. Select * from VistaProfesoresEscolar;
 - c. Select * from CuentaCheques;
 - d. Select * from Cheque;
 - e. update Profesores set sueldo = 6000 where idprofe = 7;
 - f. insert into Profesores values(8, 30, 'Salvador', 'Maestria', 20000);

SOLUCION:

1- CREATE GROUP Adminstrador;

CREATE GROUP Escolar;

2-

CREATE USER Ávila WITH PASSWORD 'Administrador' IN GROUP Adminstrador;

CREATE USER Cancino WITH PASSWORD 'Administrador' IN GROUP Adminstrador;

CREATE USER Barroso WITH PASSWORD 'Escolar' IN GROUP Escolar;

```
CREATE USER Tirado WITH PASSWORD 'Escolar' IN GROUP Escolar;  
CREATE USER Martínez WITH PASSWORD 'Escolar' IN GROUP Escolar;  
CREATE USER Aquino WITH PASSWORD 'Escolar' IN GROUP Escolar;  
CREATE USER Ramos WITH PASSWORD 'Escolar' IN GROUP Escolar;
```

3-

```
GRANT ALL ON cuentacheques TO Adminstrador;  
GRANT ALL ON cheque TO Adminstrador;  
GRANT ALL ON tabulador TO Adminstrador;  
GRANT ALL ON profesores TO Adminstrador;  
GRANT ALL ON concepto TO Adminstrador;  
GRANT ALL ON recibo TO Adminstrador;  
GRANT ALL ON detallerecibo TO Adminstrador;  
  
GRANT SELECT ON cursosespeciales TO Adminstrador;  
GRANT SELECT ON ceverano TO Adminstrador;  
GRANT SELECT ON ceextracurricula TO Adminstrador;  
GRANT SELECT ON cursoextracurricular TO Adminstrador;  
GRANT SELECT ON materias TO Adminstrador;
```

4-

```
GRANT ALL ON cursosespeciales TO Escolar;  
GRANT ALL ON cursoextracurricular TO Adminstrador;  
GRANT ALL ON materias TO Adminstrador;  
GRANT ALL ON ceverano TO Adminstrador;  
GRANT ALL ON ceextracurricula TO Adminstrador;  
GRANT ALL ON alumnos TO Adminstrador;  
GRANT ALL ON bimestre TO Adminstrador;  
GRANT ALL ON faltas TO Adminstrador;  
GRANT ALL ON calendarioescolar TO Adminstrador;
```

5-

```
CREATE VIEW profesoresvista AS SELECT idprofe,idtab,nombre,maximo FROM profesores;  
GRANT ALL ON profesoresvista TO Escolar;
```

6-

```
Server [localhost]:  
Database [postgres]: Academia  
Port [5432]:  
Username [postgres]: barroso  
Contraseña para usuario barroso:  
pgsql (11.11)  
ADVERTENCIA: El código de página de la consola (850) difiere del código  
de página de Windows (1252).  
Los caracteres de 8 bits pueden funcionar incorrectamente.  
Vea la página de referencia de psql «Notes for Windows users»  
para obtener más detalles.  
Digite «help» para obtener ayuda.  
Academia=>
```

Ya dentro de la base de datos como el usuario Barroso ejecutamos una serie de consulta de pruebas:

a-

```
Academia=> SELECT * FROM Profesores;  
ERROR: permiso denegado a la tabla profesores
```

b-

```
Academia=> SELECT * FROM profesoresvista;  
 idprofe | idtab | nombre | maximo  
-----+-----+-----+-----  
      1 |    40 | Roberto | Maestria  
      2 |    70 | Carlos  | Doctorado  
      3 |    20 | Luis    | Licenciatura  
      4 |    30 | Yunuan | Maestria  
      5 |    10 | Julio   | Licenciatura  
      6 |    20 | Samuel  | Licenciatura  
(6 filas)
```

c y d-

```
Academia=> SELECT * FROM cuentacheques;  
ERROR: permiso denegado a la tabla cuentacheques  
Academia=> SELECT * FROM cheque;  
ERROR: permiso denegado a la tabla cheque
```

ANALISIS DE LETRA

A) Departamento de Administración (DA)

efectuar el pago a los docentes

os cobros a los alumnos

Dirigido: C.P. Ávila

Auxiliado: Sr. Cancino

B) Departamento de Control Escolar (DCE)

Dirigido: Lic. Barroso

Auxiliado: Sras. Tirado

Martínez

Aquino

Ramos

en éste donde se decide qué cursos se imparten en el periodo

quién los imparte

y se aceptan las solicitudes de los alumnos

Un caso especial) Profesores

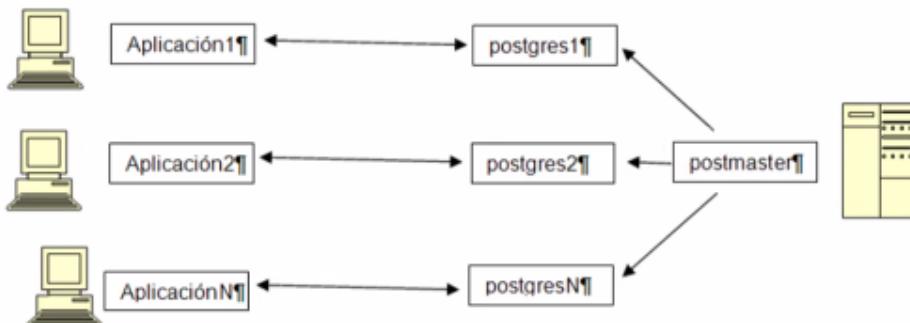
El DA es quién les puede modificar el sueldo quincenal, mientras que el DCE ni siquiera puede visualizar este.

Lo curioso es que el DCE es quien acepta los docentes y los registra en el sistema.

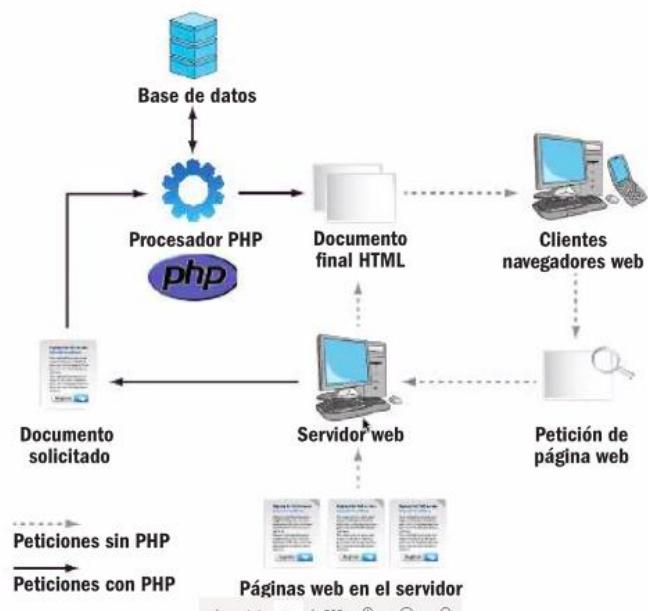
Pero es el DA donde se captura el sueldo.

Introducción: Arquitectura de PostgreSQL

PostgreSQL funciona con una arquitectura Cliente/Servidor, un proceso servidor (**postmaster**) y una serie de aplicaciones cliente que realizan solicitudes de acciones contra la base de datos a su proceso servidor. Por cada una de estas aplicaciones cliente, el proceso postmaster crea un proceso **postgres**.



Tenemos aquí que se ejecutan una serie de aplicaciones cliente (FrontEnd) y una serie de procesos en el servidor (BackEnd), de modo que, por ejemplo, la interacción entre un programa que se ejecuta en un cliente, por ejemplo, pgadmin3, una aplicación PHP, etc y el servidor de base de datos sería:



Concepto de proceso en informática: un programa en ejecución.

¿Si yo en una sola computadora ejecuto el pgAdmin y hago una consulta sql y hago lo mismo en la consola, cuantos clientes se ejecutan al servidor?

En realidad, son dos clientes. Un cliente es el pgAdmin y el otro es la consola.

El proceso Postmaster:

- es el proceso inicial
- gestiona los accesos multiusuario y multiconexión
- levanta la memoria compartida
- está al tanto de solicitudes de nuevas conexiones
- lanza procesos de atención de demanda, realizando las operaciones sobre la base de datos a solicitud de los clientes
- realiza el enlazado con los archivos de datos, gestionando estos ficheros, donde los ficheros pueden pertenecer a varias bases de datos

La comunicación entre BackEnd/FrontEnd se realiza mediante **TCP/IP**, normalmente por el puerto **5432**, creando un **socket** (Socket designa un concepto abstracto por el cual dos **programas** (posiblemente situados en **computadoras** distintas) pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada.).

De esta manera podemos definir el concepto de **CLUSTER DE BASE DE DATOS**, como una instancia de PostgreSQL, donde se lanza un proceso postmaster, que puede gestionar varias bases de datos. En un servidor, pueden haber varios cluster, cada uno tendrá su postmaster, y cada postmaster escuchará por un puerto diferente.

CLUSTER

Definicion de un Cluster

Un cluster de alta disponibilidad es un conjunto de dos o más máquinas que se caracterizan por mantener una serie de servicios compartidos y por estar constantemente monitorizándose entre sí.

Un cluster es una colección de bases de datos que están administradas por una sola instancia del servidor.

Crear un clúster de base de datos consiste en crear los directorios donde se almacenarán las bases de datos, la generación de las tablas compartidas del catálogo (tablas que pertenecen al grupo en su totalidad y no a una determinada base de datos), y crear las bases de datos template1 y postgres.

CLASE 14/4/2021

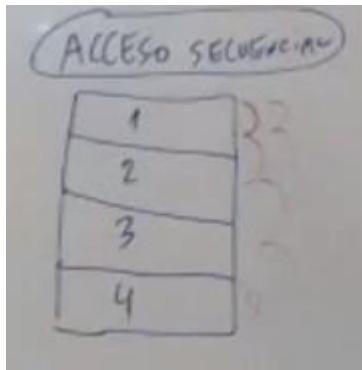
Parcial 26 de abril.

Retomamos el 5 de mayo que es miércoles.

Tema: Optimización de consultas. Índices

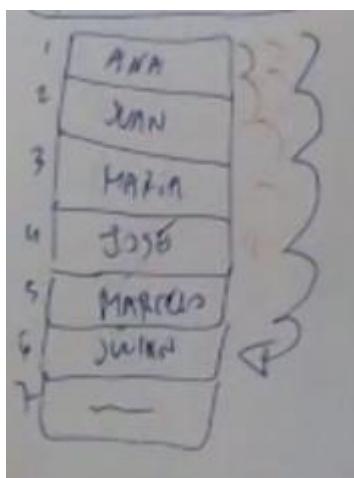
Concepto de acceso secuencial

Tenemos datos en cada parte del disco, cuando decimos de acceder secuencialmente a cada dato. Es lo que hacemos: si queremos acceder al segundo, tenemos que parar primero por el primero y así secuencialmente. No se puede acceder directamente a una ubicación.



Si esto fuera una tabla, y tenemos 1000000 registros obesa 1000000 cuadraditos. Tenemos que ir uno por uno secuencialmente.

Tenemos una tabla con registro y un numero identificador.



Ahora si manejamos el concepto índice, decimos que jose esta en la ubicación 4. Ana en la 1 y maría en la 3.

El acceso ideas do es como que tiene un punto e indica en que lugar del disco se encuentra el registro. En lugar de ir de forma secuencial buscando a algún registro vamos directo.

ÍNDICE	
SOM	4
AMP	1
MARIA	3

Falencias de índices:

Consume mucha memoria, ya que tenemos una tabla cliente y digo bueno para que ande más rápido le hago un índice a cada campo de esa tabla. Si tenemos 3 tablas va a crear 3 tablas más, eso consumiría más memoria. Si bien índice es para acelerar las búsquedas una mala práctica como crear un índice para cada campo eso nos aria el efecto secundario del que queremos.

¿Cuál es la estrategia para crear índice? ¿A qué campos?

La clave primaria es un índice por definición. Sería los campos que más consulto, los que busco más. Los que más pongo en el WHERE.

Las consultas funcionan lentas ¿?

Lo primero como accesorios sería: mirar como se hace la consulta.

HERRAMIENTAS

El motor planifica como ejecutar las consultas.

Con una palabra antes de la consulta, dice como planea hacerlo. EXPLAIN

ORGANIZACIÓN Y ACCESO DATOS

Un índice es una estructura diferente dentro de la base de datos; creado con el comando **create index**. Requiere su propio espacio en disco y contiene una copia de los datos de la tabla. Eso significa que un índice es una redundancia. Crear un índice no cambia los datos de la tabla; solamente establece una nueva estructura de datos que hace referencia a la tabla. De hecho, un índice de base de datos se parece mucho a un índice de un libro: ocupa su propio espacio, es redundante y hace referencia a la información actual almacenada en otro lugar.

Primary index

Primary index: es creado por defecto en las tablas al usar la sentencia PRIMARY KEY

```
create table libro(
    id_libro int not null PRIMARY KEY,
    titulo varchar(40),
    autor varchar(30),
    editorial varchar(15),
    precio decimal(6,2)
);
```

Common index

Common index: acelera la búsqueda para el campo/s sobre el que se crea el índice, en este campo los valores no necesariamente son únicos y aceptan valores nulos.

```
CREATE INDEX idx_libros_editorial ON libro(editorial);
```

notas: clave primaria no acepta nulos y ya es índice.

Unique index

Unique index: acelera la búsqueda para el campo/s sobre el que se crea el índice, en este campo/s los valores deben ser únicos y diferentes, aunque permite valores nulos y pueden definirse varios por tabla.

```
CREATE UNIQUE INDEX idx_libros_titulo_editorial on libro(titulo,editorial);
```

Nota: lo que hace es que tiene la particularidad que no aceptan duplicados.

No permite que se vuelva a repetir la misma duplicidad de título y editorial.

Ahora EXPLAIN la herramienta que muestra como se hace la consultas.

En el ejemplo dos usa un index y nos avisa.

Leer material del eva sobre EXPLAIN.

BASES DE DATOS 2

PRÁCTICO 13

Utilizar PGAdmin y Sentencias SQL.

1. Crear un usuario “jperez” con la password “jp123”.
2. Crear un usuario “mdiaz” con la password “md123”.
3. Eliminar el usuario “jperez”.
4. Crear una nueva base de datos “dbfacturas” y que el dueño de esta sea el usuario “mdiaz”.
5. Cambiar la contraseña del usuario “mdiaz” por “987654321”.
6. Crear en la base de datos “dbfacturas” una tabla “ventas” con los siguientes campos:
 - a. Idventa (serial y clave primaria).
 - b. Monto (numeric).
7. Crear un usuario “crodriguez” con la password “c543”.
8. Asignarle al usuario “mdiaz” privilegios totales sobre las tablas de la base de datos.
9. Asignarle al usuario “crodriguez” permiso de lectura sobre la tabla “ventas”.
10. Crear en la base de datos “dbfacturas” una tabla “clientes” con los siguientes campos:
 - a. IdCliente (serial y clave primaria).
 - b. Nombre (varchar).
11. Insertar un registro en la tabla “clientes”.
12. ¿El usuario “mdiaz” tiene permisos totales sobre la nueva tabla “clientes”?

1-CREATE USER jperez WITH PASSWORD 'jp123';

2-CREATE USER mdiaz WITH PASSWORD 'md123';

3-DROP USER jperez;

4-CREATE DATABASE dbfacturas WITH OWNER mdiaz;

5-ALTER USER mdiaz WITH PASSWORD '987654321';

6-CREATE TABLE ventas(

 idVenta serial PRIMARY KEY,

 Monto numeric

);

7-CREATE USER crodriguez WITH PASSWORD 'c543';

8-

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO mdiaz;
```

```
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO mdiaz;
```

9- GRANT SELECT ON ventas TO crodriguez;

10-

```
CREATE TABLE clientes(
```

```
    idCliente serial PRIMARY KEY,
```

```
    Nombre varchar
```

```
);
```

11- INSERT INTO clientes(Nombre) VALUES ('Ana');

12-Si deja porque le dimos permisos (ALL) sobre todo el Schema Public.

BASES DE DATOS 2

PRÁCTICO 14 (Seguridad y Optimización).

Se pide escribir las correspondientes sentencias SQL para:

1. Crear un grupo "*administradores*".
2. Crear un usuario "*adiaz*" con la password "123456" para el grupo "*administradores*".
3. Crear una base de datos denominada "*Club*" y que el dueño de la misma sea el usuario "*adiaz*".
4. Crear la tabla **Afiliados** en la base de datos "*Club*" con los campos: **Socioid** (serializable como clave primaria), **SocioNombre** (varchar 30), **SocioDocumento** (varchar 8 not null), **SocioActivo** (int valor por defecto 0 y not null) y **SocioFechNac** (date not null).
5. Crear un índice de tipo UNIQUE sobre el campo **SocioDocumento**.
6. Otorgar al grupo "*administradores*" todos los privilegios sobre la tabla "*Afiliados*".
7. Crear el usuario **aperez** con la password "987654321", y otorgar los siguientes privilegios: **selección, inserción, actualización y borrado** de registros en la tabla **Afiliados**.
8. Quitar todos los privilegios al grupo "*administradores*" sobre la tabla "*Afiliados*".
9. Crear una vista "**SoloNombres**" que lista el campo SocioNombre de todos los afiliados.
10. Otorgar a la vista "**SoloNombres**" privilegio de selección para todos los usuarios.

1.CREATE GROUP Administradores;

2.CREATE USER adiaz WHITE PASSWORD 123456 IN GROUP Administradores;

3. CREATE DATABASE Club WITH OWNER adiaz;

4.CREATE TABLE Club.Afiliados (

```
    Socioid serial PRIMARY KEY,
```

```
    SocioNombre varchar(30),
```

```
    SocioDocumento varchar(8) NOT NULL,
```

```
    SocioActivo int DEFAULT 0 NOT NULL,
```

```
    SocioFechNac date NOT NULL
```

```
);

5. CREATE UNIQUE INDEX idx_Afiliados_SocioDocumento ON Afiliados (SocioDocumento);

6. GRANT ALL ON Afiliados TO Admininstradores;

7.CREATE USER aperez PASSWORD 987654321;

GRANT SELECT, INSERT, UPDATE, DELETE ON Afiliados TO aperez;

8.REVOKE ALL ON Afiliados FROM Administradores;

9.CREATE VIEW SoloNombre AS SELECT SocioNombre FROM Afiliados;

10. GRANT SELECT ON SoloNombre TO PUBLIC;
```

CLASE DESPUES DEL PRIMER PARCIAL

3-5-2021 tema nuevo

¿Qué es una Transacción?

Es una **unidad lógica de trabajo** (procesamiento) de la base de datos que incluye una o más operaciones de acceso a la base de datos, que pueden ser de inserción, modificación o recuperación.

Las transacciones pueden delimitarse de forma explícita con sentencias de tipo “**iniciar transacción**” y “**terminar transacción**”

Un eje de transacción: Vamos a comprar un boleto de avión, entramos a comprar el pasaje a buenos aires, buscamos por fecha, y el programa nos devuelve los asientos que tiene disponible. Ahora llega otro usuario y enseguida se compra el siento 10. Ahora al usuario que estaba mirando los asientos disponibles se quedó sin ese lugar.

Ahora el programa a quien le da el lugar a quien lo paga primero, es decir lo selecciona pasa a la página de pago y quien lo paga primero es quien se lo queda.

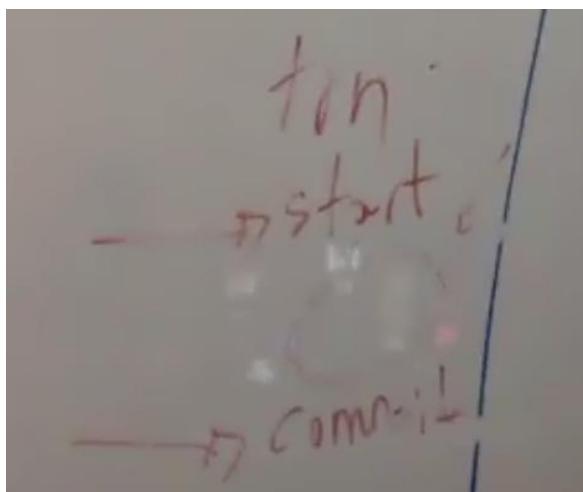
Consistencia de los datos ej: Somos una tienda de ropa que vende a crédito, un usuario compra un par de zapatos que vale 4.000 pesos, plan de pago a partir de hoy a pagar en los próximos 10 meses.

Cuántas tablas necesitamos: 3 tablas, cliente, ventas, cuotas.

Hago el insert de cliente y ventas, y ahora tantos como cuotas existan es decir 10 insert en la tabla cuotas. Estoy haciendo los insert y se me corta la luz justo cuando iba por insertar las cuotas. Cuando queramos ver los datos nos van a quedar inconsistentes.

¿Qué podemos hacer para que no ocurra eso?

Lo solucionamos con una transacción, después de lograr ingresar todos los datos realizamos comit. En el círculo están las 12 insert.



Existe el rollback, que si no le damos todos los datos consistentes tira todo para atrás. No queda nada guardados y nos ahorraremos los datos inconsistentes.

PROPIEDADES ACID:

I

La idea es que dada una BD en un estado consistente, luego de ejecutarse las transacciones la BD quede también en un estado consistente.

El SGBD debe garantizar esto último haciendo que las transacciones cumplan con las propiedades ACID.

Estas propiedades son:

- **Atomicidad:** T se ejecuta completamente o no se ejecuta por completo (todo o nada)
- **Consistencia:** T transforma un estado consistente de la BD en otro estado consistente (los programas deben ser correctos)
- **Aislamiento:** Las Ti se ejecutan sin interferencias.
- **Durabilidad:** Las actualizaciones a la BD serán durables y públicas.

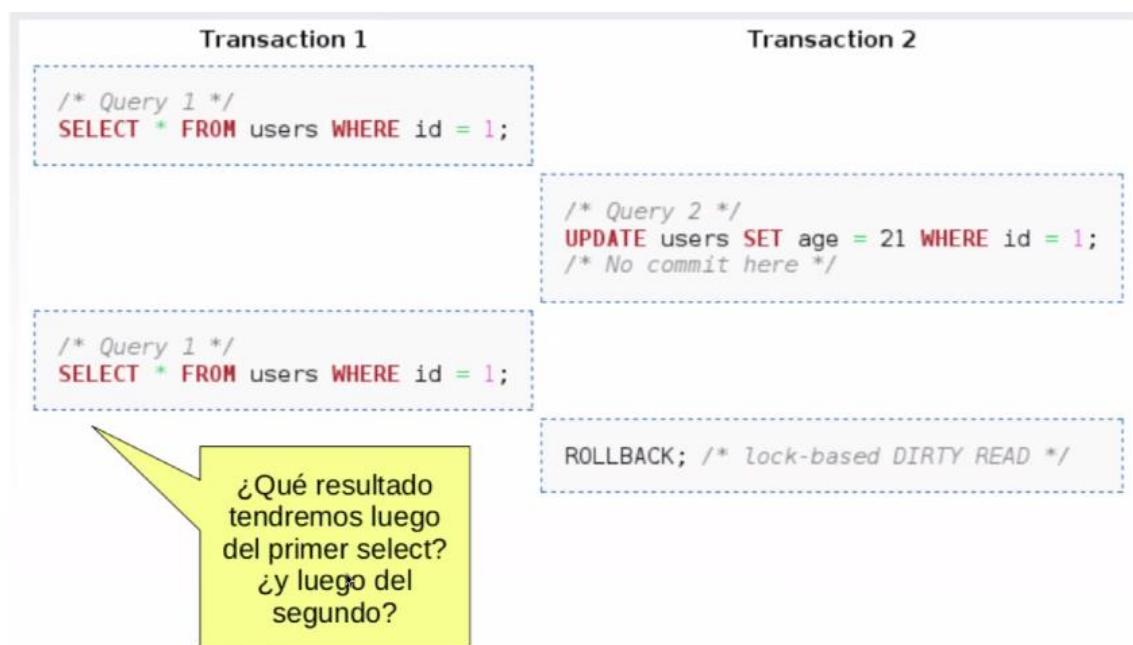
→ Importante para parciales y exámenes saber de memoria.

EJ aislamiento:

Niveles de Aislamiento

READ UNCOMMITTED:

Apenas transaccional, permite hacer lecturas sucias (dirty reads), donde las consultas dentro de una transacción son afectadas por cambios no confirmados (not committed) de otras transacciones

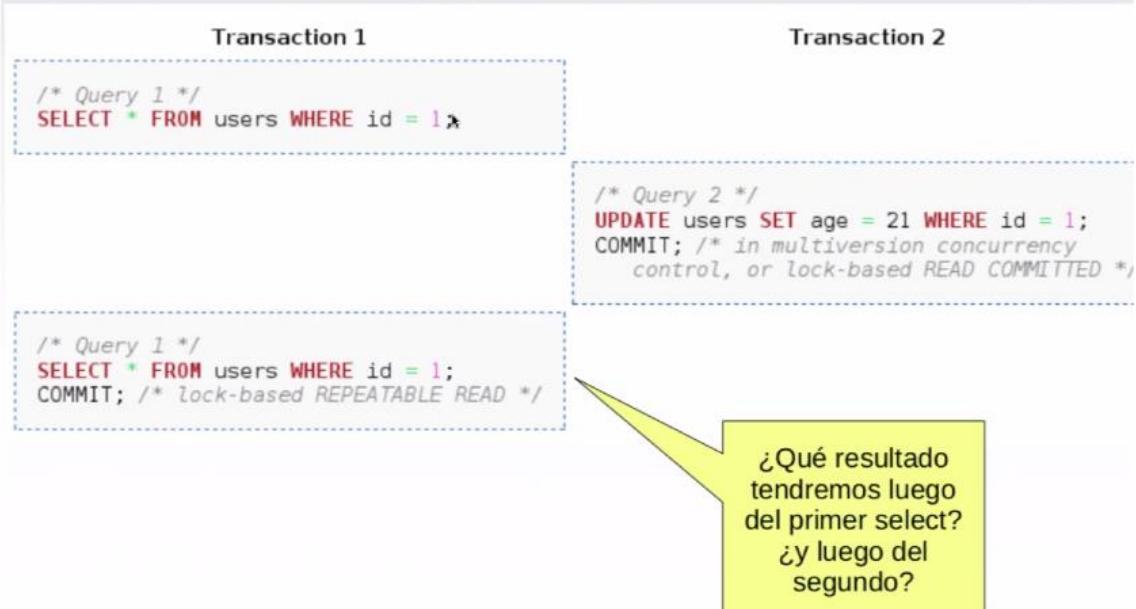


Que se quede viendo el 21 age seria una lectura sucia.

RED COMMITTE

Los cambios confirmados son visibles dentro de otra transacción esto significa que dos consultas dentro de una misma transacción pueden retornar diferentes resultados (Generalmente este es el comportamiento por defecto en los SGBD)

READ COMMITTED (Non-repeatable reads)



El usuario id empieza con 20 años.

SERIALIZABLE:

No se permiten las actualizaciones en otras transacciones si una transacción ha realizado una consulta sobre ciertos datos (las distintas transacciones no se afectan entre si)

Para garantizar ACID, las transacciones tiene que pasar por determinados estados:

Diagrama de Estados de la Ejecución de una Transacción

Figure 21.4
State transition diagram illustrating the states for transaction execution.

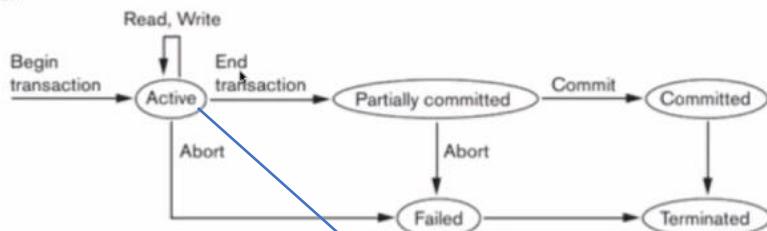


Figura de Elmasri/Navathe-*Fundamentals of DB Systems*, 6th Ed., Pearson, 2011

Una cosa son los estados de una transacción y otra son los comitt y rollback, eso son lo mismo.

Commit y Rollback es cómo van a terminar las transacciones. Pero tiene estados para garantizar eso, activo, parcialmente comiteado, comiteado, fallado, terminado (Commit-RollBack).

Concepto niveles de aislamiento, lo importante es decidir que nivel de aislamiento elegimos para cada transacción. Tenemos transacciones que se ejecutan entrelazadas, tenemos que ver que tan aisladas están una transacción de la otra.

Es una **unidad lógica de trabajo** (procesamiento) de la base de datos que incluye una o más operaciones de acceso a la base de datos, que pueden ser de inserción, modificación o recuperación

Las transacciones pueden delimitarse de forma explícita con sentencias de tipo “**iniciar transacción**” y “**terminar transacción**”

```
iniciar T0
... operaciones ...
terminar T0
```

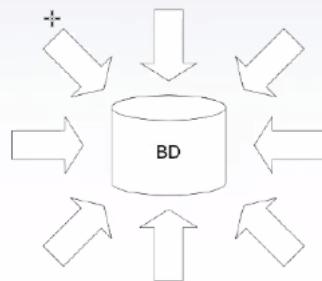
Además, en las transacciones tenemos operaciones básicas (“**leer elemento**”, “**escribir elemento**”), y cálculos sobre los datos leídos.

Las transacciones tienen otras propiedades “**deseables**” que veremos más adelante

```
iniciar T0
leer(A)
leer(B)
A = A + B
B = B * 1.1
escribir(A)
escribir(B)
terminar T0
```

CONCURRENCIA

Es cuando muchas transacciones acceden a la misma Base de Datos al mismo tiempo. Especialmente, cuando acceden a los mismos datos de la misma Base de Datos al mismo tiempo



Sean T0 y T1 dos transacciones:

T0:

```
leer(A)  
A = A - 50  
escribir(A)  
leer(B)  
B = B + 50  
escribir(B)
```

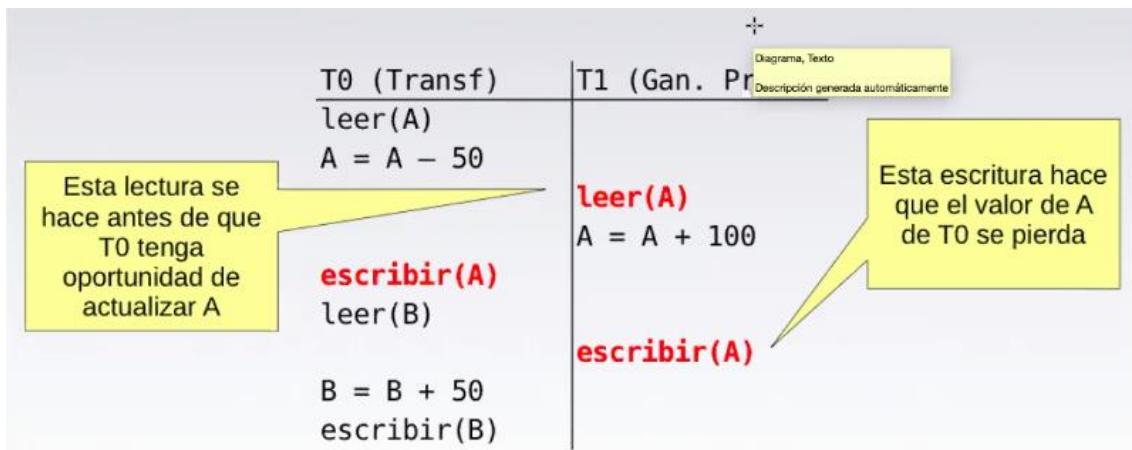
T1:

```
leer(A)  
temp = A * 0.1  
A = A - temp  
escribir(A)  
leer(B)  
B = B + temp  
escribir(B)
```

Donde A y B son saldos de dos cuentas bancarias diferentes con valores de 1000 y 2000 respectivamente

$$A + B = 3000$$

Acá no das 'problemas'



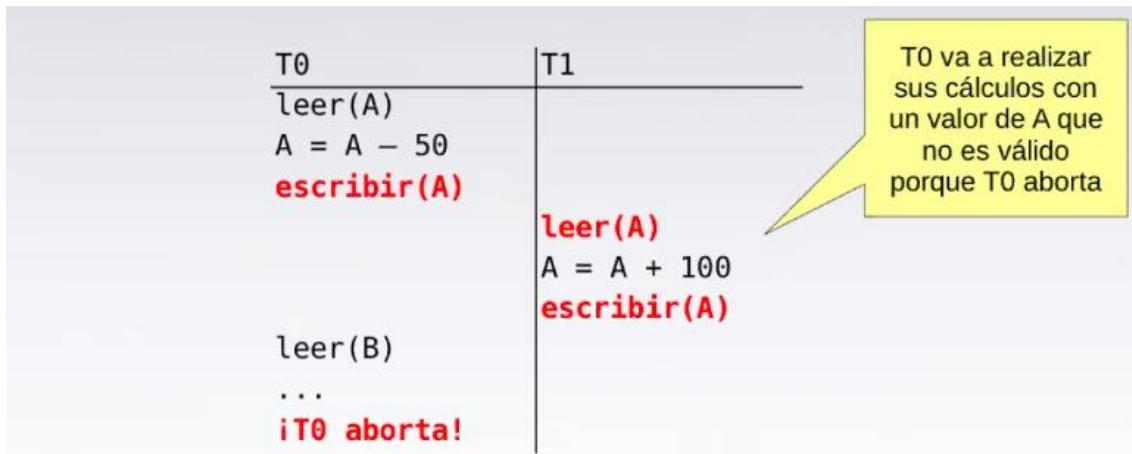
Actualización Perdida

T1 lee el valor de A antes de que T0 lo pueda actualizar, luego T0 escribe A, pero después T1 sobre escribe A con un valor incorrecto.
Como consecuencia la actualización que hizo T0 se pierde.

7

Acá si da problemas, si bien los ejemplos no son el mismo, el problema mayor es que escribe en la derecha que se estaba usando en la izquierda. Una inconsistencia, se llama actualización perdida, perdimos datos.

¿Por qué tenemos un problema? porque evidentemente, si sacas de un lado y pones en el otro, ahora tenemos un problema. Una inconsistencia.



Actualización Temporal (Lectura Sucia)

T1 lee el valor de A luego de que T0 lo escribió, y realiza cálculos en base a dicho valor de A. Sin embargo, luego T0 aborta, por lo tanto el valor de A leído por T1 ya no es válido

8

Tenemos 2 transacciones la primera va corriendo la segunda se mete en medio, pero la primera aborta. ¿Qué pasa? Ya que empezamos con 1.000 en leer (a) del lado izquierdo, termina con 1.000.

T0	T1	
<pre> leer(A) A = A - 50 escribir(A) leer(B) B = B + 50 escribir(B) </pre>	<pre> suma = 0 leer(A) suma = suma + A leer(B) suma = suma + B </pre>	<p>La suma se debería hacer completamente antes o después de T0, pero no en el medio</p>

Diagrama, Texto
Descripción generada automáticamente

Resumen Incorrecto

T1 está calculando la suma (o cualquier otra función agregada) con un valor correcto de A, pero con un valor incorrecto (anterior) de B

9

PROPIEDADES
DESEABLES DE
LAS
TRANSACCIONES

ACID:
Atomicity (Atomicidad)
Consistency (Consistencia)
Isolation (Aislamiento)
Durability (Durabilidad / Permanencia)



EJEMPLO PRÁCTICO

```
postgres=# BEGIN TRANSACTION;
BEGIN
postgres=# SELECT * FROM departamento WHERE codigo=1
codigo | nombre
-----+
1 | Informatica
(1 row)

postgres=# UPDATE Departamento SET nombre='Computacion';
UPDATE 3
postgres=# ROLLBACK
ROLLBACK
postgres=# SELECT * FROM departamento WHERE codigo=1;
codigo | nombre
-----+
1 | Informatica
(1 row)
postgres=#

```

El dato no actualizó porque abortamos (ROLLBACK) la transacción

Ej en posgres

```
1 begin transaction;
```

Nos dice Begin que empezamos

```
1 begin transaction;
2
3 select * from clientes where ci='1';
```

Data Output Explain Messages Notifications					
	ci [PK] character (8)	nombre character (30)	apellido character (30)	direccion character (30)	telefono character (10)
1	1	juan	[null]	[null]	[null]

Ahora nos dice que tenemos ese cliente

```
update clientes set nombre='maria' where ci='1';
```

le cambiamos el nombre

```
2
3 select * from clientes where ci='1';
```

Al ejecutar nos dice que cambio el nombre

Data Output Explain Messages Notifications					
	ci [PK] character (8)	nombre character (30)	apellido character (30)	direccion character (30)	telefono character (10)
1	1	maria	[null]	[null]	[null]

Ahora

```
rollback;
```

Si ahora ejecutamos la consulta

```
select * from clientes where ci='1';
```

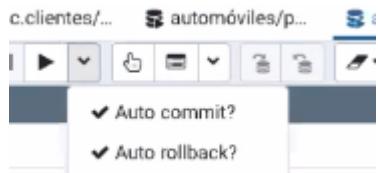
Data Output Explain Messages Notifications					
	ci [PK] character (8)	nombre character (30)	apellido character (30)	direccion character (30)	telefono character (10)
1	1	juan	[null]	[null]	[null]

No cambio nada porque tiramos Rollback.

¿Como veríamos al usuario 1 desde otra transacción?



Antes usábamos



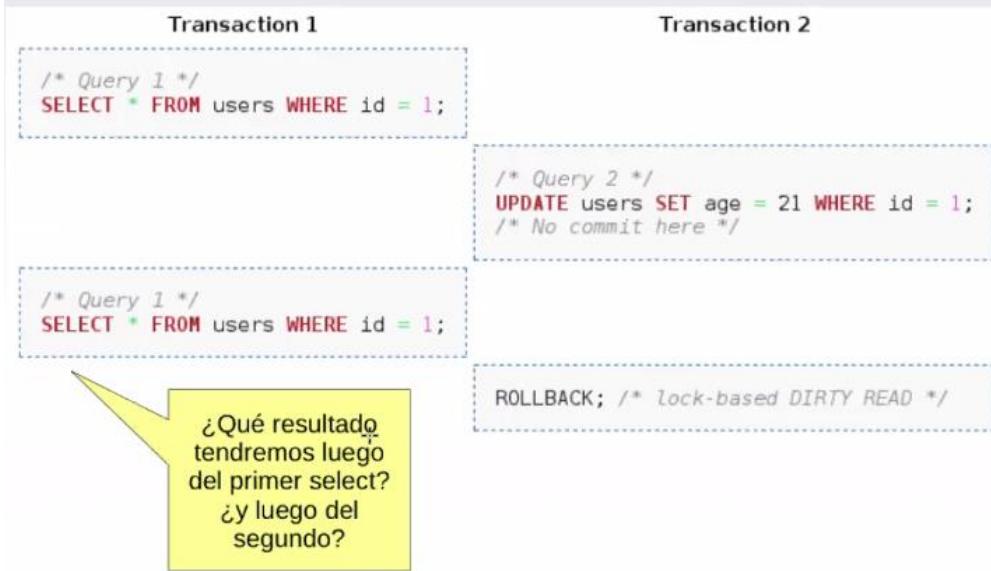
Si teníamos un error no lo hacia y si lo hacíamos bien lo guardaba.

NIVELES DE AISLAMIENTOS

READ UNCOMMITTED:

Apenas transaccional, permite hacer lecturas sucias (dirty reads), donde las consultas dentro de una transacción son afectadas por cambios no confirmados (not committed) de otras transacciones

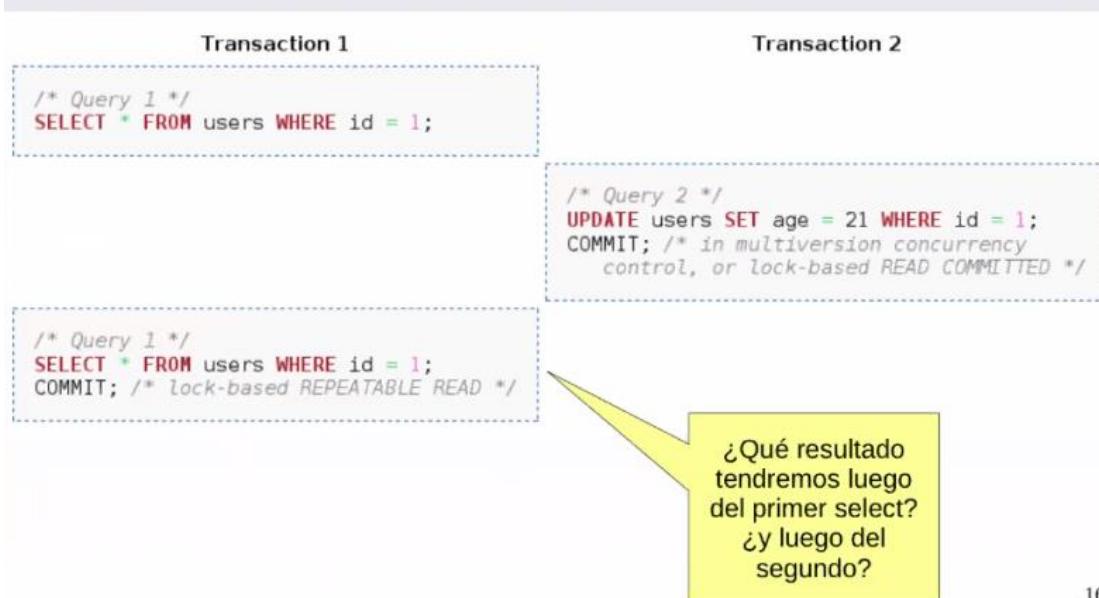
READ UNCOMMITTED (dirty reads)



READ COMMITTED:

Los cambios confirmados son visibles dentro de otra transacción esto significa que dos consultas dentro de una misma transacción pueden retornar diferentes resultados (Generalmente este es el comportamiento por defecto en los SGBD)

READ COMMITTED (Non-repeatable reads)

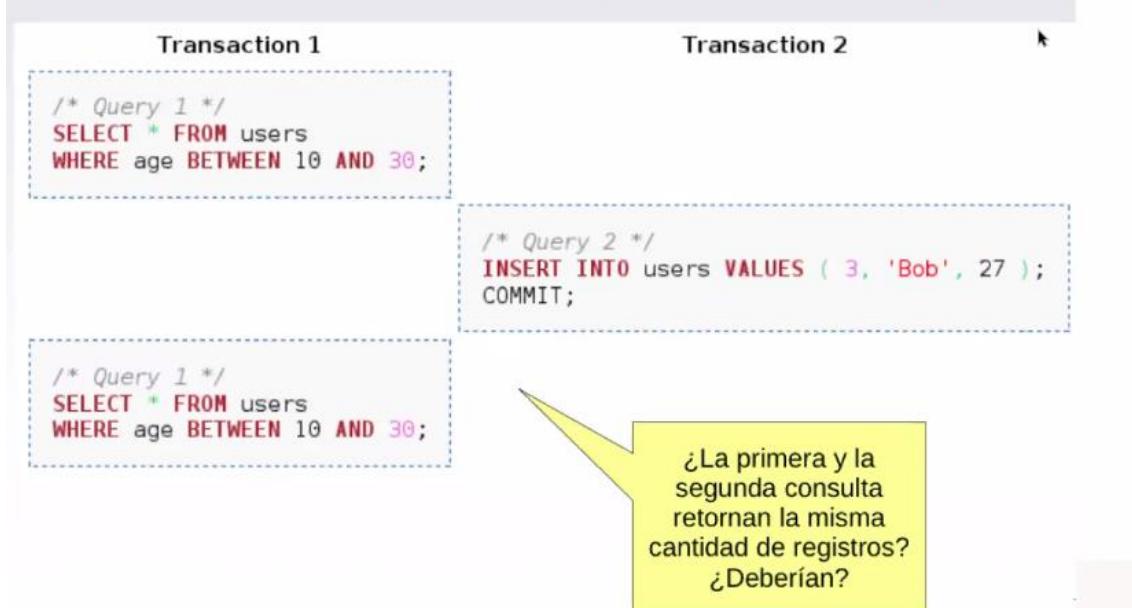


Lee los que ya tuvieron un commit. Lee lecturas no repetidas, se lee de otro lado se hace un cambio y cuando vuelve a leer el dato esta cambiado.

REPEATABLE READ:

Dentro de una transacción todas las lecturas son consistentes (Esto es el comportamiento por defecto en MySQL usando tablas en InnoDB)

REPEATABLE READS (Phantom Reads)



Lectura fantasma, en el primer select hace un listado. Pero en medio ocurre un insert. Ahora cuando se hace el segundo select el resultado no cambia, por eso lectura fantasma.

SERIALIZABLE:

No se permiten las actualizaciones en otras transacciones si una transacción ha realizado una consulta sobre ciertos datos (las distintas transacciones no se afectan entre si)

***Las transacciones están completamente aisladas entre si
(Esto tiene un costo asociado...)***

T0:

leer(A)
A = A - 50
escribir(A)
leer(B)
B = B + 50
escribir(B)

T1:

leer(A)
temp = A * 0.1
A = A - temp
escribir(A)
leer(B)
B = B + temp
escribir(B)

ocurre EL MUNDO PERFECTO, eso hace la serializarle. Va a ser lo ideal.

```
postgres=# BEGIN TRANSACTION;  
BEGIN  
postgres=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;  
SET  
postgres=# -- Otras operaciones... ---  
UPDATE 3  
postgres=# COMMIT;  
COMMIT  
postgres=#  
SET TRANSACTION
```

Diagrama
Descripción generada automáticamente

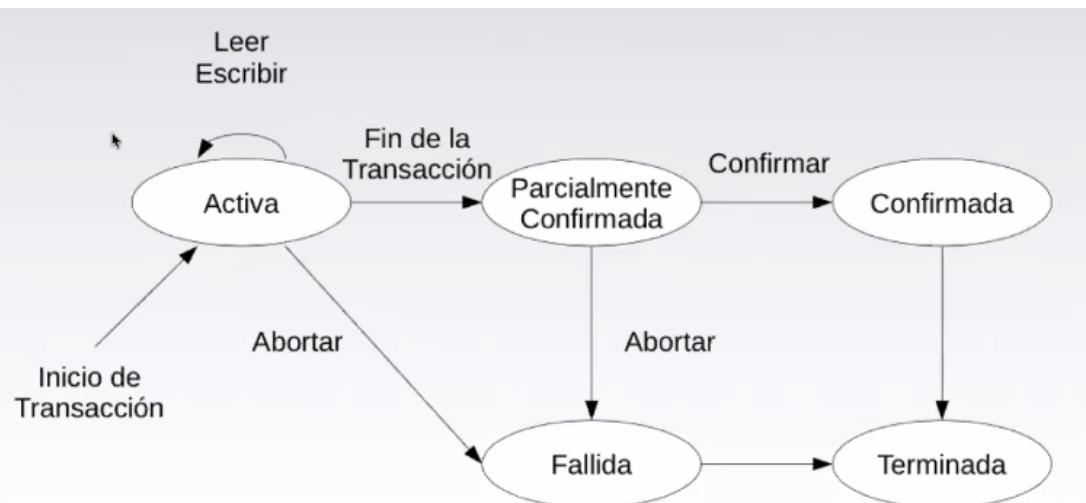
O ROLLBACK, según sea necesario

ejemplo de como se pone el nivel de aislamiento, la contesta como SET, luego hacemos las operaciones y luego el commit. El nivel de aislamiento lo sitiamos para cada operación según lo que necesitamos.

¿Cómo Logran los SGDB esto?

En general, las transacciones tienen una serie de estados, que permiten hacerle un adecuado seguimiento

Estados de las transacciones, es la forma que tiene de garantizar que funcione bien.



Confirmar = Commit
Abortar = Rollback

¿Qué es una Planificación?

Una planificación representa el orden cronológico en que se ejecutan o se han ejecutado las instrucciones de un conjunto de transacciones en el SGBD

Una planificación para una transacción debe conservar todas las instrucciones de la transacción. Además, se debe conservar el orden de las instrucciones dentro de la transacción

Plan:

{T1, L(A)}, {T0, L(A)}, {T0, E(A)}, {T0, L(B)}
{T1, E(A)}, {T0, E(B)}, {T1, L(B)}, {T1, E(B)}

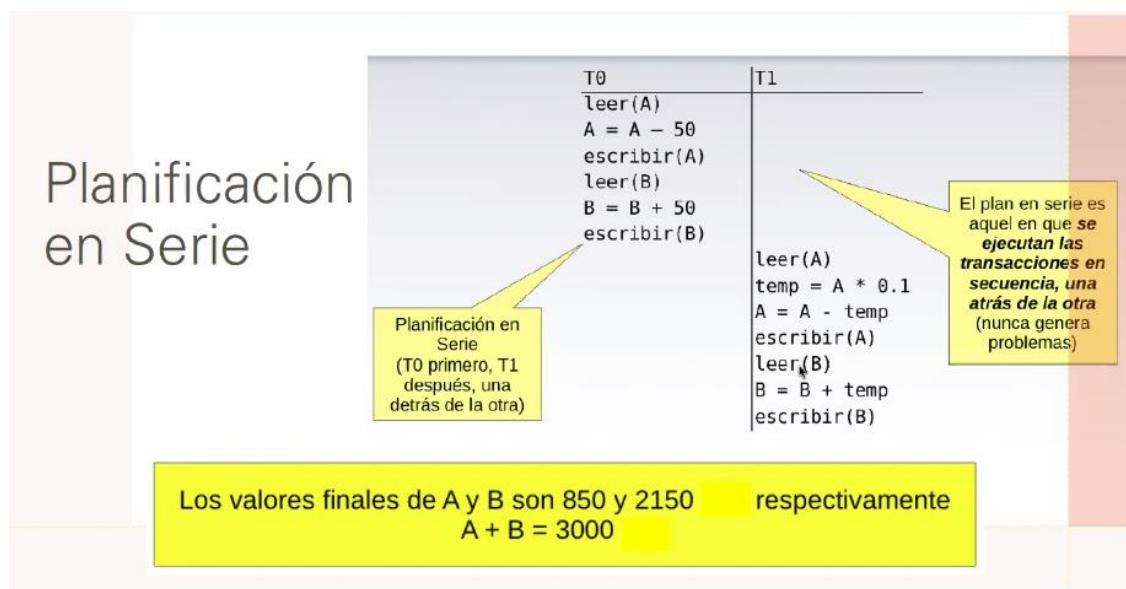
Un plan en serie (serial) es aquel en que las transacciones se ejecutan completas en secuencia una detrás de otra

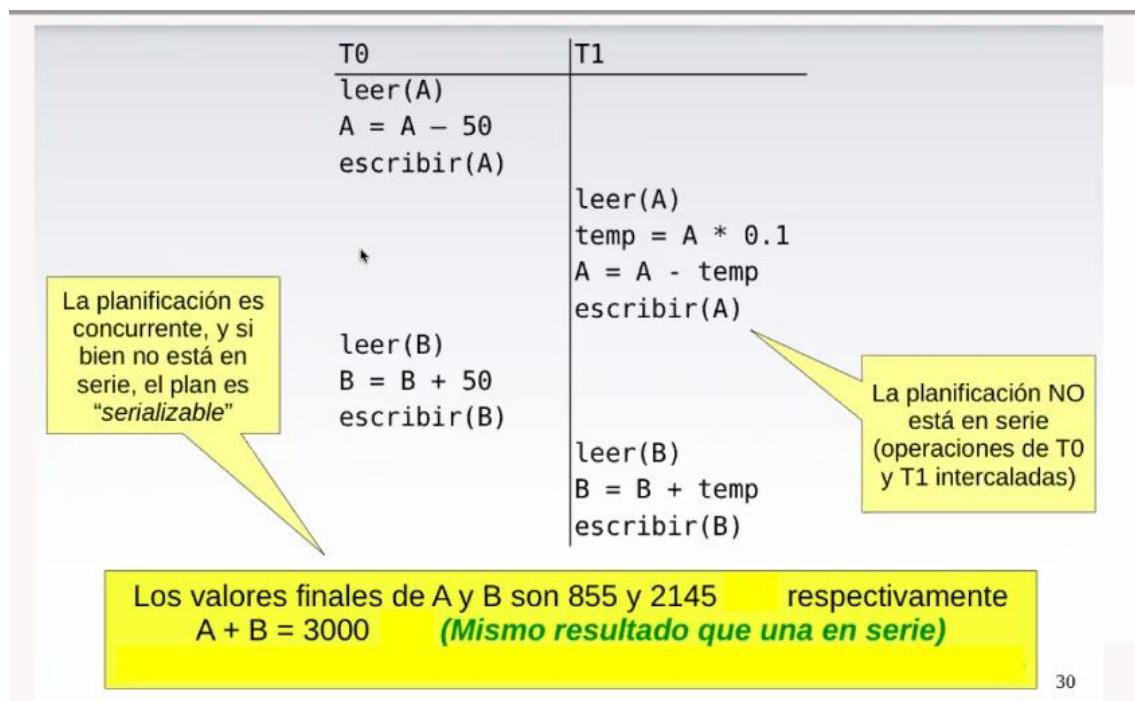
Un plan no en serie (no serial, o plan intercalado) es aquel en el que se intercalan de forma simultánea instrucciones de distintas transacciones



Planificación es lo de historia del a clase pasada

Dado el primer ejemplo de hoy

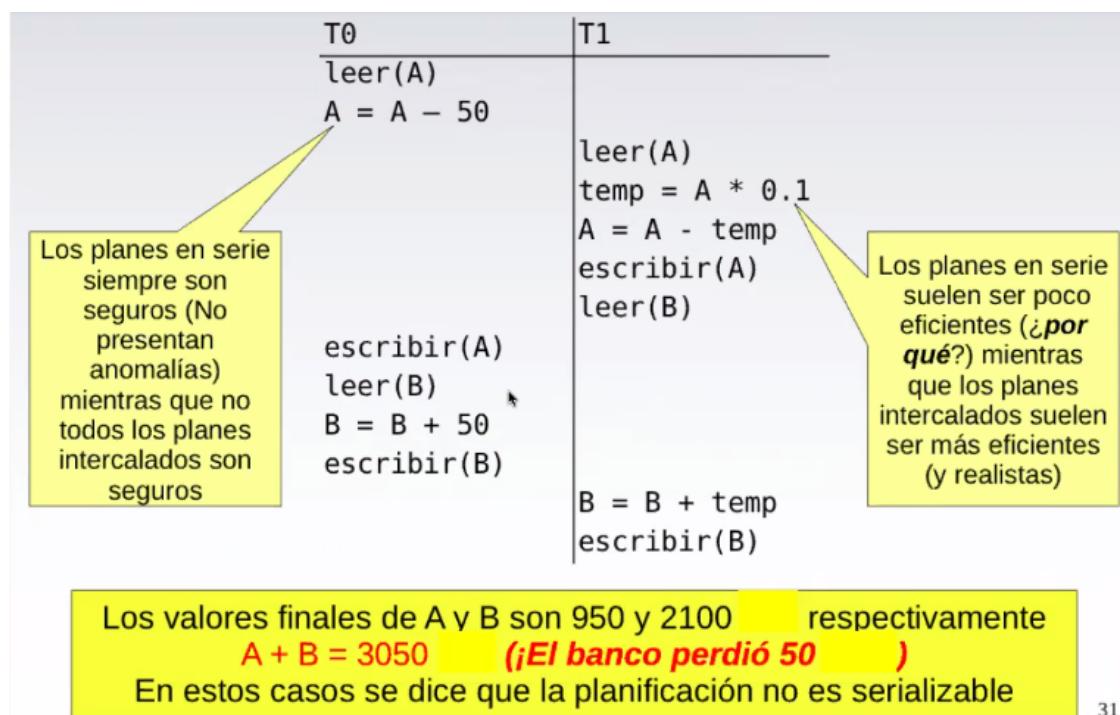




30

A pesar de estar enlazadas no genera error, nos da un resultado correcto.

No da problema porque dice que el plan es serializable.



31

Ese caso si tiene problemas, es que lee a y le resta y en T1 lo lee de nuevo y el escribir recién esta abajo en T0.

OPERACIONES CONFLICTIVAS

- **{T0, leer(A)}, {T1, escribir(A)}**: El orden si importa, porque T0 está leyendo un valor que deja de ser válido cuando T1 realiza la escritura.
- **{T0, escribir(A)}, {T1, leer(A)}**: Aplican las mismas consideraciones que en el caso anterior, aun cuando esta combinación en concreto no genera anomalías.

Para que exista un conflicto debe existir al menos 2 transacciones.

Se dice que dos operaciones I_i e I_j de un plan S están en conflicto si operan sobre el mismo dato, pertenecen a transacciones distintas y al menos una de ellas es escribir

Si dos operaciones NO están en conflicto entonces **es posible** intercambiar el orden en que se ejecutan sin generar ningún tipo de anomalía

PRACTICO TRANSACCION

PRÁCTICA 2

Entrar a la consola de PostgreSQL

1. Crear una base de datos "**Pruebas**".
2. Crear una tabla "**foo**" con dos campos: "**id**" (int, clave primaria) y "**edad**" (int).
3. Insertar un registro con los valores (1,10).
4. Iniciar una transacción.
5. Hacer un Select de la tabla foo.
6. Abrir otra consola y actualizar la edad del registro 1 a 11.
7. Volver a la consola inicial, y hacer un Select de la tabla foo y observar si la edad se mantiene en 10 o cambió a 11.
8. Hacer commit en la consola incial.
9. Iniciar una transacción.
10. Abrir otra consola:
 - o Iniciar transacción.
 - o Actualizar la edad del registro 1 a 12 y **no hacer commit**.
11. Volver a la consola inicial, y hacer un Select de la tabla foo y observar si la edad se mantiene en 11 o cambió a 12.
12. Volver a la consola donde cambió la edad a 12 y hacer commit.
13. Volver a la consola inicial, y hacer un Select de la tabla foo y observar si la edad se mantiene en 11 o cambió a 12.
14. Hacer commit.
15. Iniciar una transacción.
16. Abrir otra consola:
 - o Iniciar transacción.
 - o Insertar un nuevo registro con id=2 y la edad 20 y **no hacer commit**.
17. Volver a la consola inicial, y hacer un Select de la tabla foo y observar cuantos registros nos devuelve.
18. Volver a la consola donde inserto el registro y hacer commit.
19. Volver a la consola inicial, y hacer un Select de la tabla foo y observar cuantos registros nos devuelve.
20. Hacer commit.
21. Iniciar una transacción.
22. SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE;
23. Abrir otra consola:
 - o Iniciar transacción.
 - o Insertar un nuevo registro con id=3 y la edad 30 y **HACER commit**.
24. Volver a la consola inicial, y hacer un Select de la tabla foo y observar cuantos registros nos devuelve.

2. CREATE TABLE foo (

 id INT PRIMARY KEY,

 edad INT

);

3. INSERT INTO foo(id,edad) VALUES (1,10);

4. BEGIN TRANSACTION;

5.SELECT * FROM foo;

6.

 Pruebas/postgr...  Pruebas/postgres@PostgreSQL 11 *

UPDATE foo SET edad = 10 WHERE id=1 ;

7. EFECTIVAMENTE

	id [PK] integer	edad integer
1	1	10

8. COMMIT;

9. BEGIN TRANSACTION;

10.

 Pruebas/postgr...  Pruebas/postgres@PostgreSQL 11 *

BEGIN TRANSACTION;

UPDATE foo SET edad = 12 WHERE id=1;

11. En algo falle porque me dice edad 10

	id [PK] integer	edad integer
1	1	10

12. COMMIT;

13. Cambio ahora la edad es 12

	id [PK] integer	edad integer
1	1	12

14. COMMIT;

15. BEGIN TRANSACTION;

16.

BEGIN TRANSACTION;

INSERT INTO foo(id,edad) VALUES (2,20);

17. Nos devuelve uno solo

	id [PK] integer	edad integer
1	1	12

18. COMMIT;

19. Nos devuelve 2 registros

	id [PK] integer	edad integer
1	1	12
2	2	20

20. COMMIT;

21. BEGIN TRANSACTION;

22. SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE;

23.

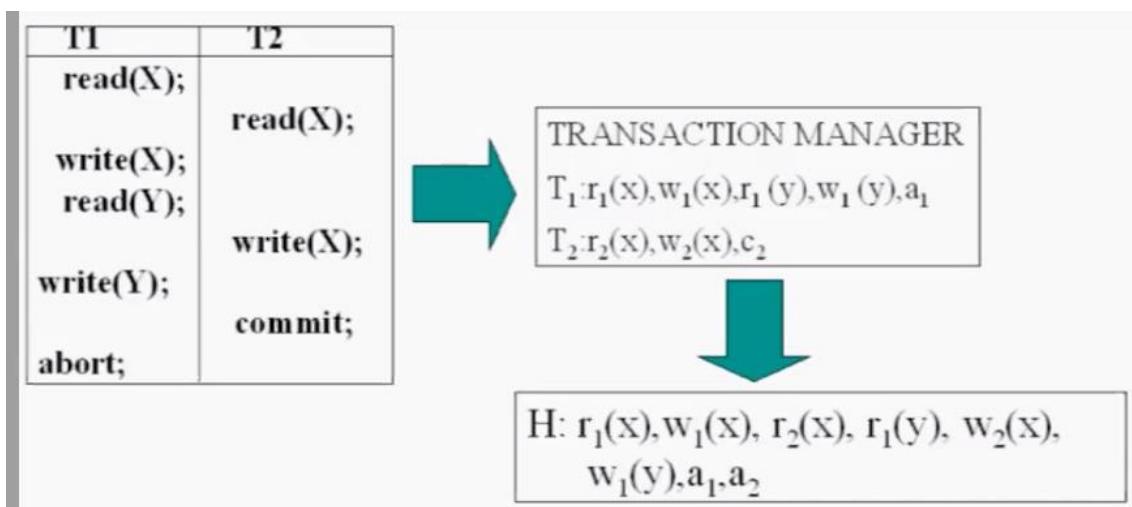
BEGIN TRANSACTION;

INSERT INTO foo(id,edad) VALUES (3,30);

COMMIT;

24. Si volvemos a la consola inicial y hacemos un SELECT nos devuelve 2 registros

	id [PK] integer	edad integer
1	1	12
2	2	20



Termina entrelazándolas y decide que la T2 tiene que abortar también, para que T2 no tenga una lectura errónea.

El Manejador de Transacciones (Scheduler o Transaction Manager)

- Se encarga de la administración de las transacciones en la Base de Datos.
- Para eso recibe las instrucciones que los programas pretenden ejecutar y se toma la libertad de reordenarlas, sin cambiar nunca el orden relativo de los Read y Write.
- Puede llegar a agregar instrucciones (nunca R/W) por su cuenta.
- Puede llegar a demorar la ejecución de las instrucciones.

LEER TEORICO DEL EVA

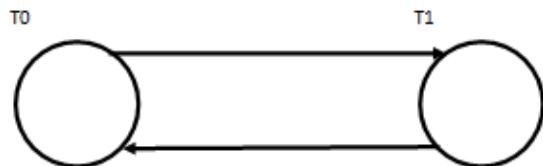
EJERCICIO 3 (5p)

Dada la siguiente historia S:

S:r0(A);r1(A);w0(A);r0(B);w0(B);w1(A);r1(C);w1(C)

Se pide:

1. Construir el SG (S) (grafo de precedencia).
2. Indicar si S es SR (serializable)



T0 T1

R(A)

2.

No es serializable tiene un ciclo, tenemos el caso en que T1 R(A) y T0 W(A) y luego T1 W(A).

W(A)

R(B)

W(B)

W(A)

R(C)

W(C)

EJERCICIO 2 (5p)

Dadas las siguientes transacciones:

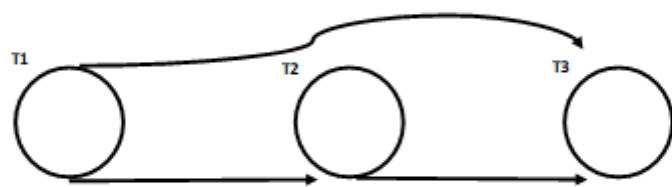
- T1= r(A); w(A); r(B); w(B); c
- T2= r(A); w(A); c
- T3= r(B); r(A); w(A); w(B); c

Y la historia:

- H1= r1(A); w1(A); r2(A); r1(B); w1(B); c1 r3(B); w2(A); r3(A); c2 w3(A); w3(B); c3

Se pide:

1. Construir el SG (H1) (grafo de precedencia).
2. Indicar si H1 es SR (~~serializable~~) y en caso afirmativo indicar las historias seriales equivalentes.



El grafo tiene ciclo?

T1 T2 T3

No tiene.

R(A)

2.

W(A)

Como no tiene ciclo si es SR.

R(A)

R(B)

Sacamos T1 junto con las aristas. (Ya que de entrada es al que no le llegan aristas)

W(B)

Ahora a cual no llegan aristas ?

C

R(B)

T2

W(A)

Me queda solo T3.

R(A)

T2

C

W(A)

En teoría vamos quitando al que no le llegan aristas y eso los vamos anotando, nos quedaría un orden T1, T2, T3

W(B)

C

Práctico

BASES DE DATOS

PRÁCTICO TRANSACCIONES Y CONCURRENCIA

EJERCICIO 1

1. Indicar si las siguientes historias son serializables.
2. Construir los grafos de serialización para cada historia.

- $H_1: r_1(x), r_2(x), w_1(x), r_1(y), w_1(y), w_2(x)$
- $H_2: r_1(x), w_1(x), r_2(x), w_2(x), r_1(y), w_1(y)$

R(E)

W(C)

CLASE DEL DIA 19/5/2021

¿Pregunta de parcial?

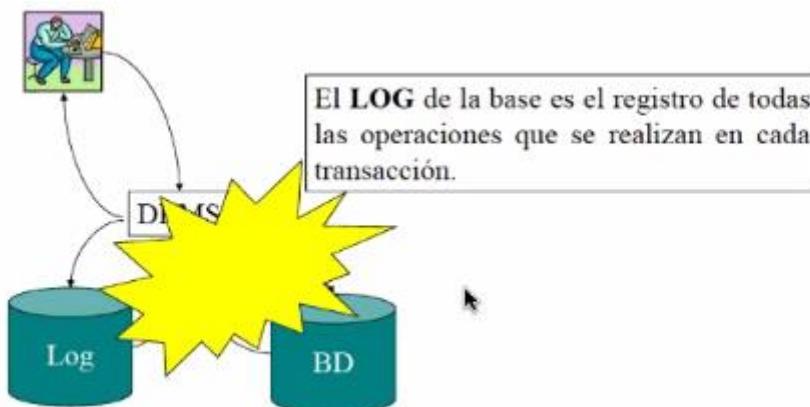
Que tipo de fallas pueden pasar:

Corte de luz

Corte de internet

Error en la aplicación

¿Que hace la base de datos cuando vuelve a funcionar?



La base de datos genera un archivo Log que almacena todas las operaciones que realizamos, todos los select, todo los insert.

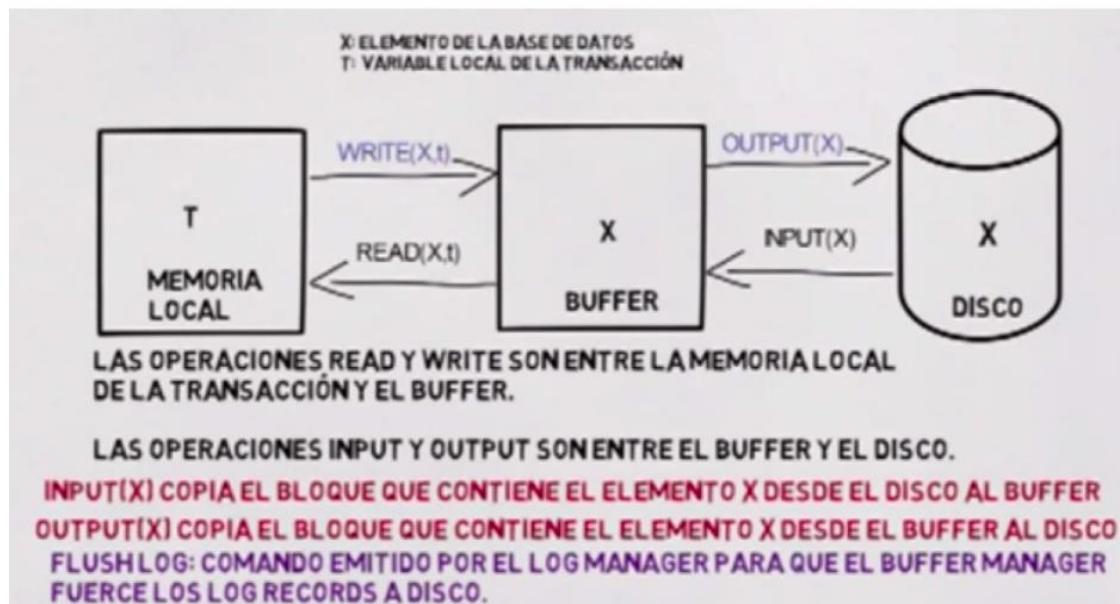
Políticas de recuperación

Algoritmos de Recuperación

- Si hubo un desastre (se pierde el disco, etc.) entonces:
 - Se debe recuperar el último respaldo conocido de la base.
 - Se debe recuperar el LOG hasta donde se pueda
 - Se deben rehacer (redo) todas las operaciones indicadas en el log desde el momento en que se hizo el respaldo a la base.
- Si la falla fue menor (corte de energía, falla en la red, etc):
 - puede ser que haya que deshacer (undo) cambios ya realizados.
 - puede ser que haya que rehacer cambios que no se hayan confirmado.

- Es un registro de la actividad sobre la base, típicamente con entradas como las siguientes:
- **[start_transaction,T]**: Comenzó la transacción T
- **[write_item,T,X,v_ant,v_actual]**: La transacción T cambió el ítem X del valor v_ant al valor v_act.
- **[read_item,T,X]**: La transacción T leyó el valor del ítem X
- **[commit,T]**: La transacción T confirma que todos los efectos deben ser permanentes en la base.
- **[abort, T]**: La transacción T abortó, por lo que ningún cambio realizado por T debe ser permanente en la base

Algunas de las operaciones que se encuentran en un log, un detalle del log es que tiene por ejemplo el dato anterior y el dato nuevo, muestra que se cambió.

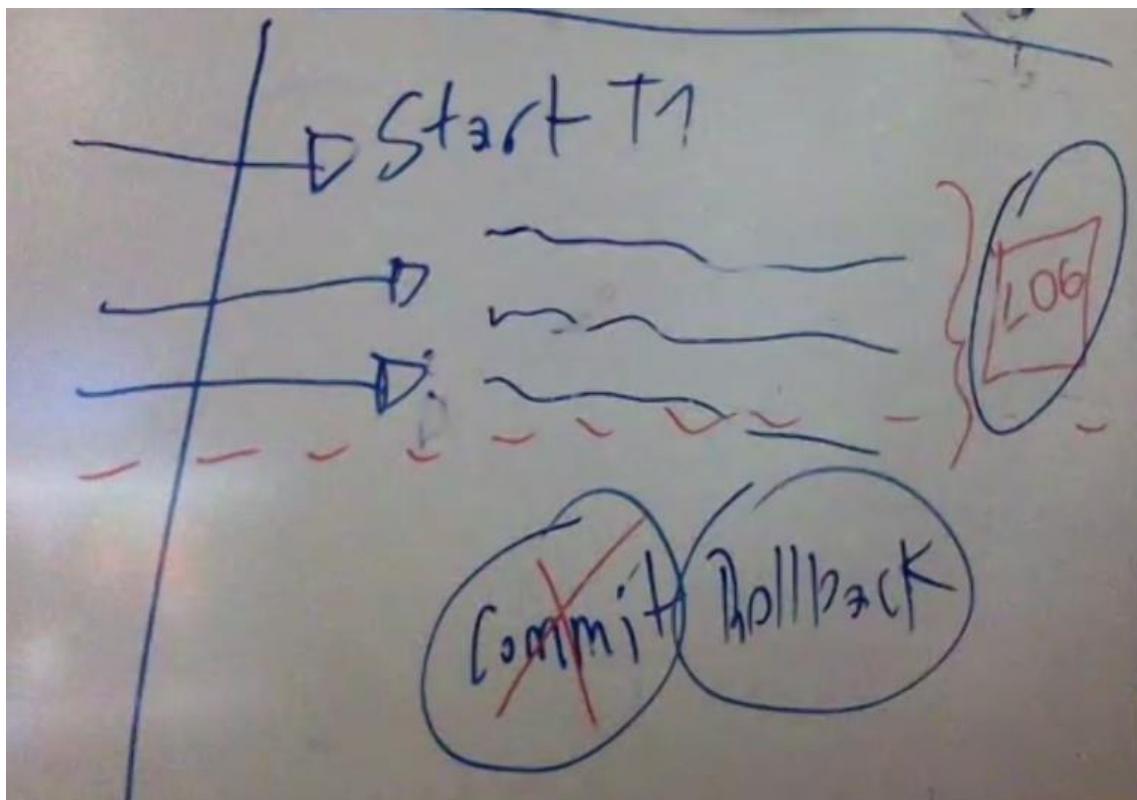


Buffer memoria temporal.

Transacciones Completas e Incompletas

- **Transacciones completas:** son las transacciones comiteadas (terminaron exitosamente con COMMIT) y las transacciones abortadas (terminaron con ABORT).
- **Transacciones incompletas:** son las transacciones con registro de START, y sin registro de COMMIT ni ABORT.

Volvemos a la pizarra podemos decir que es incompleta.



Un error común es pensar que esta incompleta porque tiene un rollback.

Algoritmos de Recuperación: Actualización Diferida e Inmediata

- Si la falla no fue catastrófica, hay dos técnicas básicas:
- **Actualización Diferida:** Cada transacción trabaja en un área local de disco o memoria y recién se baja al disco después que la transacción alcanza el commit.
 - Si hay un abort o una falla, no es necesario deshacer ninguna operación (NO-Undo/Redo).
- **Actualización Inmediata:** La base es actualizada antes de que la transacción alcance el commit.
 - Si hay un abort o falla, se deben deshacer las operaciones de la transacción.

En realidad, es algo de la bd el buffer.

- Siempre se graba primero el Log para garantizar la recuperación.

Checkpoint

- **[checkpoint]:** Registro del Log que indica que todos los buffers modificados de la base fueron actualizados al disco.
- Ninguna transacción T tal que [commit,T] aparece en el Log antes que [checkpoint] necesita Redo.
- El checkpoint consiste de los siguientes pasos:
 - Suspender la ejecución de todas las transacciones.
 - Grabar todos los buffers modificados en el disco
 - Registrar el [checkpoint] en el log y grabar el Log en disco.

Checkpointing

Objetivo

Limitar la cantidad de registros del Log que deben ser examinados en la recuperación.

Que significa idempotente: Significa que no importa cuantas veces ejecute una operación, siempre va a quedar una vez sola. Si yo hago un delete 10 veces, de todas maneras, solo se ejecuta una vez.

- Es fundamental que la operación de **Redo** sea idempotente.

Tres políticas de recuperación y son:

Undo

Deshace las transacciones incompletas

Redo

Rehace las transacciones que hicieron **COMMIT**

Undo/Redo

Deshace las transacciones incompletas y rehace las que hicieron **COMMIT**

- Log:

- [start_transaction,T1]
- [write_item,T1,D,20]
- [commit,T1]
- [checkpoint]
- [start_transaction,T4]
- [write_item,T4,B,15]
- [write_item,T4,A,20]
- [commit,T4]
- [start_transaction,T2]
- [write_item,T2,B,12]
- [start_transaction,T3]
- [write_item,T3,A,30]
- [write_item,T2,D,25]

CT={T₄}

AT={T₂, T₃}

T₁: Ignorar porque terminó antes del CHKP

T₂, T₃: Ignorar en Redo, relanzar al final.

T₄: Aplicar Redo_w.

Notas: 2 y 3 incompletas.

4 completa, le puedo aplicar redo porque esta completa y es idempotente, como no tenemos un checkpoint para saber que llego bien a dc vamos a decir que le hacemos un redo para comprobar, eso no afecta en nada.

No se le puede hacer redo a operaciones incompletas, nunca llegaron a disco, las ignoro.

Esta mal decir que nos deshacemos de 2 y 3, esta muy mal, en diferida no tenemos un deshacer, en inmediata si se puede.

EJERCICIO 1

Un banco desea elegir qué manejador de base de datos es más conveniente en el contexto de trabajo que se les presenta. Ante todo, tienen muchas transacciones pequeñas y concurrentes. Para tomar la decisión, instalaron en máquinas exactamente iguales los manejadores A y B, y estudian el funcionamiento de los mismos frente a las ejecuciones repetidas de las mismas transacciones en los dos sistemas, en situaciones normales y de falla (alguien desconecta la energía...) y luego analizan los logs. Los registros del log analizados son los siguientes:

Manejador A	Manejador B
[start_transaction, T ₁] [read, X, T ₁] [write, X, 5, 7, T ₁] [start_transaction, T ₂] [commit, T ₁] [checkpoint] [write, X, 7, 20, T ₂] [start_transaction, T ₃] [commit, T ₂] [write, X, 20, 45, T ₃] FALLA !	[start_transaction, T ₁] [start_transaction, T ₂] [read, X, T ₁] [write, X, 5, 7, T ₁] [commit, T ₁] [write, X, 7, 20, T ₂] [checkpoint] [start_transaction, T ₃] [commit, T ₂] [write, X, 20, 45, T ₃] FALLA !
Antes de la recuperación : X=7 Después de la recuperación: X=20	Antes de la recuperación: X=45 Después de la recuperación: X=20

Se pide:

1. ¿Cuál es la estrategia de recuperación que sigue el manejador A? Justifique su respuesta.
2. ¿Cuál es la estrategia de recuperación que sigue el manejador B? Justifique su respuesta.

1.Diferido

Redo T2, ignora T3 porque es incompleta.

2.Inmediata

Primer el ckeckpoint arranca la transacción 3 y commit a la 2. La 3 es incompleta.

Undo T2 y T3

Corrección:

Cuando pregunta la estrategia de recuperación nos pregunta si es diferida o inmediata.

- El método **Undo** requiere que los items sean grabados inmediatamente después de que la transacción terminó, quizás incrementando el número de I/O.
- El método **Redo** requiere mantener todos los bloques modificados en el Buffer hasta que la transacción commitea y los registros de log fueron flusheados, quizás incrementando el promedio de bloques en buffer requeridos por las transacciones.
- El tercer método de Logging, llamado **Undo/Redo**, provee mayor flexibilidad para ordenar las acciones pero a expensas de mantener más información en el Log.

Política: Undo (Recovery)

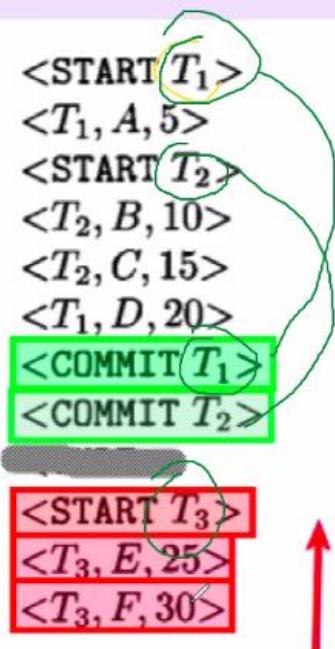


Figure 17.4: An undo log

Cuando aplicamos el undo arranca de abajo a arriba. Ve que T3 no tiene commit entonces la desase

Política: Redo (Recovery)

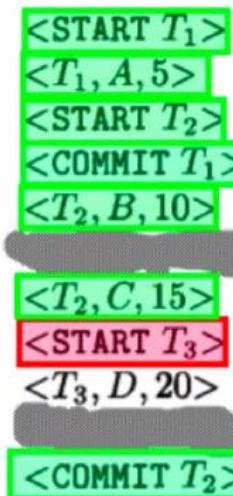


Figure 17.8: A redo log

De arriba abajo, veo que T3 esta incompleta, solo ejecuta nuevamente T1 y T2, las vuelve a ejecutar para quedarse tranquila e ignora T3.

Política: Undo/Redo (Recovery)

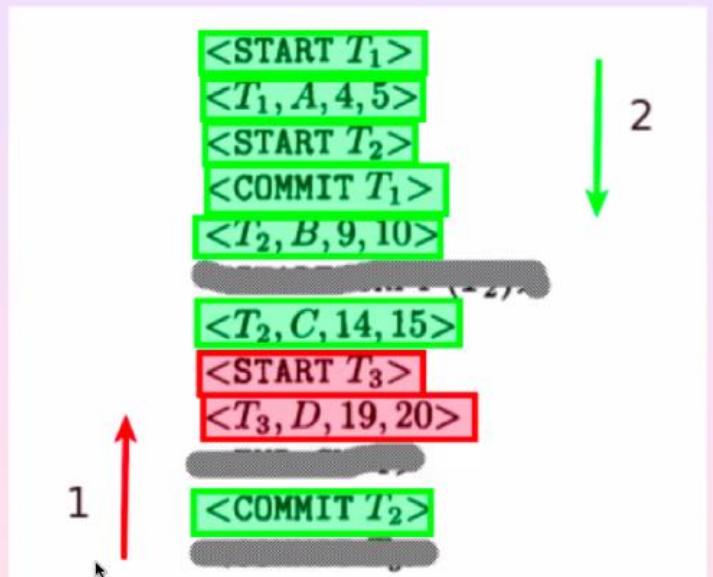


Figure 17.10: An undo/redo log

Primero undo y después redu.

EJERCICIO 2

Si la situación es la siguiente:

r1(X) w1(X) r2(X) r1(Y) w2(X) w1(Y) c1 r2(Y) caída del sistema

¿Cómo recuperará el sistema si trabaja con Actualización Inmediata?

Hacer undu y redu que hago con cada una de las transacciones

EJERCICIO 3

En los diferentes casos de fallas, decir en qué pasos consistiría el proceso de recuperación.

1. Se trabaja en modo de Actualización Inmediata, y ocurre una caída del sistema junto con falla física (ROTURA DE DISCO).
2. Se trabaja en modo de Actualización Diferida y ocurre una caída del sistema.^[OBJ]

1. Contestar que haríamos nosotros en el proceso de recuperación. Ya sabemos que tenemos una rotura de disco, primero tenemos que hacer un backup es fundamental. Como tenemos actualización inmediata, tenemos que hacer el algoritmo undu/redu. Como es inmediata, puede que no tengamos operaciones correctas, por eso usamos el undu, redu o undu/redu.

2. Levantar el sistema y hacer redu.

Clase del 24/5/2021

Resolución del ejercicio de práctico de recurrencia.

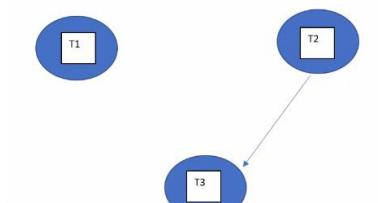
Primero separamos en formato tabular.

T1	T2	T3
	Leer elemento(Z)	
	Leer elemento(Y)	
	Escribir elemento(Y)	
		Leer elemento(Y)
		Leer elemento(Z)
Leer elemento(X)		
Escribir elemento(X)		
		Escribir elemento(Y)
		Escribir elemento(Z)
	Leer elemento(X)	
Leer elemento(Y)		
Escribir elemento(Y)		
		Escribir elemento(X)

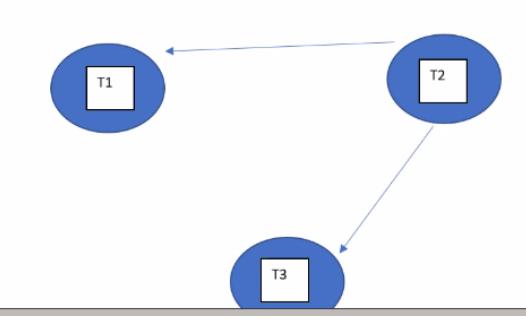
Hacemos las 3 'Elipses', arranco con t2 que es donde arranca la historia.

Si en t2 leo z y en t3 escribo z, tenemos un conflicto y lo representamos con una flecha.

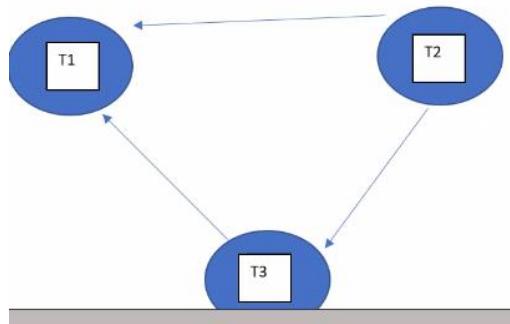
Ahora todo lo Y, me voy fijando y como ya tengo representada el conflicto no es necesario volver a representarlo.



Ahora vemos que en T2 escribo en Y. Podemos ver que T2 tiene conflicto con T1. Y representamos.

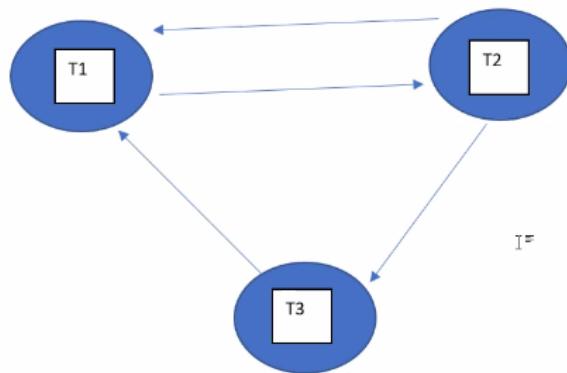


Ahora continuamos Con T3 en Lee elemento Y, y nos damos cuenta que tiene un conflicto con T1 y lo representamos.



Ahora en T1 tenemos un conflicto en T2 porque T1 lee y T2 escribe. Lo representamos

| Escribir_elemento(X) |

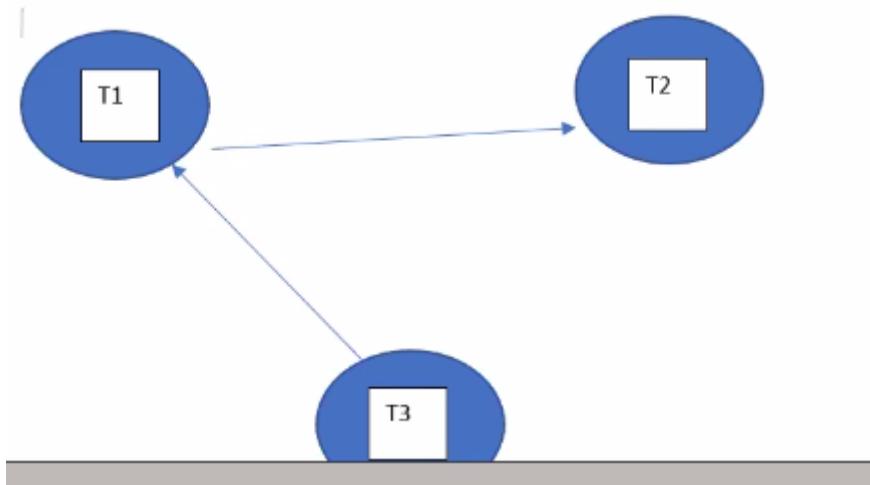


Ya se ve claramente que no es serializable, se puede apreciar el circulo en el grafo.

Si la pregunta es si es serializable, cuando ya encontramos una circularidad no es necesario seguir con el ejercicio, ya abrimos demostrado que no lo es y no es necesario continuar.

EJERCICIO 2

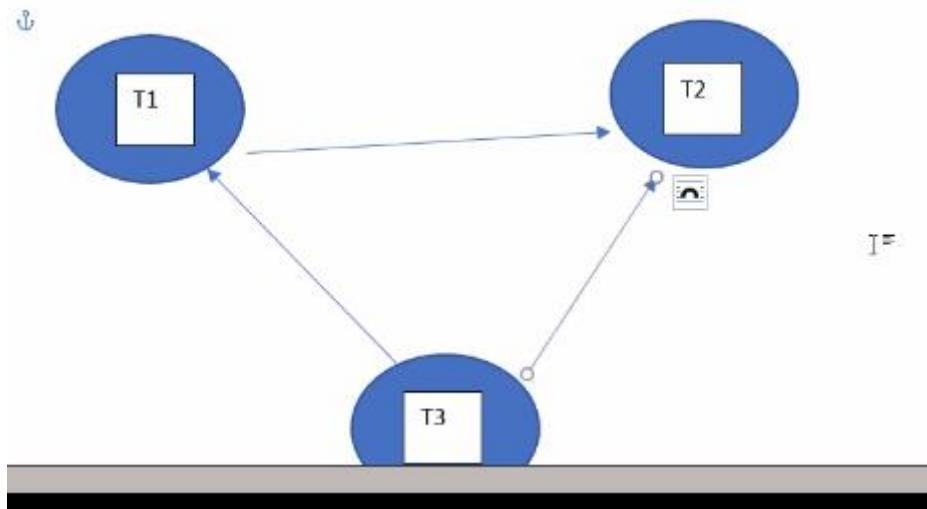
T1	T2	T3
		LEER_ELEMENTO(Y)
		LEER_ELEMENTO(Z)
LEER_ELEMENTO(X)		
ESCRIBIR_ELEMENTO(X)		
		ESCRIBIR_ELEMENTO(Y)
		ESCRIBIR_ELEMENTO(Z)
	LEER_ELEMENTO(Z)	
LEER_ELEMENTO(Y)		
ESCRIBIR_ELEMENTO(Y)		
	LEER_ELEMENTO(Y)	
	ESCRIBIR_ELEMENTO(Y)	
	LEER_ELEMENTO(X)	
	ESCRIBIR_ELEMENTO(X)	



Ahora voy:

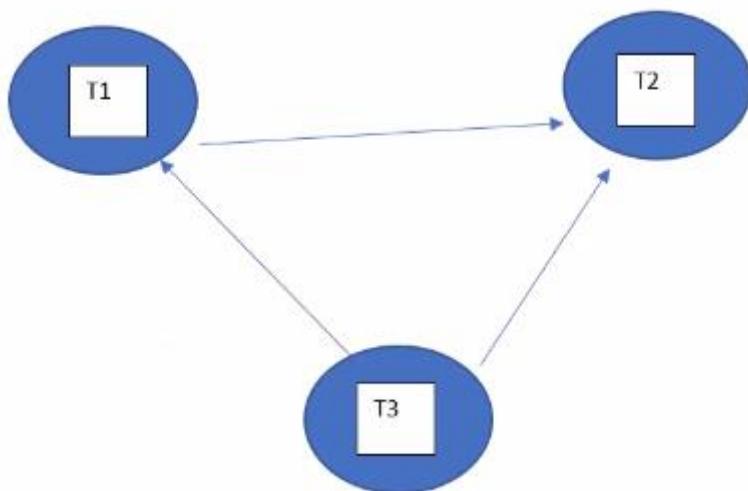
T3
LEER_ELEMENTO(Y)
LEER_ELEMENTO(Z)
ESCRIBIR_ELEMENTO(Y)
ESCRIBIR_ELEMENTO(Z)

Y analizo, vemos que en T2 tiene conflicto porque lee y lo representamos



Ahora en t3 con escribir z, pero como ya esta presentada no hace falta.

Finalmente vemos todos los casos y nos queda el grafo:



Si es serializable porque no tiene ningún ciclo.

Plan serial equivalente:

Ahora sacamos la arista a la cual no llegan flechas, esa es la T3. Nos llevamos también las flechas que salen de ella.

Ahora nos llevamos T1 y luego T2.



Plan serie equivalente: T3, T1, T2

EJERCICIO EN GRUPO:

PRACTICO DE RECUPERACION ANTE FALLAS III

Sean las transacciones:

T1 : r1(x), w1(y), w1(u), c1

T2 : r2(y), w2(z), r2(w), w2(w), c2

T3 : r3(z), w3(x), c3

y la siguiente historia que finaliza con la caída del sistema:

H: r1(x), w1(y), r2(y), w2(z), r3(z), w3(x), r2(w), w2(w), <Falla sistema >

Se pide:

- a. Decir que transacciones deberían revertirse y por qué.
- b. Dar una historia de T1, T2 y T3 en la cual haya que revertir solo T2. Explicarlo.
- c. Sea la historia:

H: r1(x), w1(y), r2(y), w2(z), r3(z), w3(x), c3, r2(w), w2(w), <Falla sistema>

- I. Decir que transacciones deberían revertirse y por qué.
- II. ¿Que fenómeno se da en este caso y por qué?
- III. Decir si la base de datos corre riesgo de quedar en un estado inconsistente.

A. Se revierten todas porque están todas incompletas (no tiene commit), porque no se logró el commit en ninguna antes de la falla del sistema.

B.

H: r1(x), w1(y), w1(u), c1, r2(y), ...

C.

- I. Tenemos T1 completa y T2 incompletas. T1 y T2 tienen que revertirse. Si yo Abortan T1, T2 que están incompletas vemos que T3 leyó de T2. Entonces aborta las 3.
- II. El fenómeno se llama un aborto en cascada.
- III. Si yo no aborto T3, la base quería inconsistente.

EJERCICIO EQUIPO



Ejercicio en Equipo



Suponga la siguiente situación: en el día de hoy (**24 de Mayo 2021**) durante el funcionamiento normal de un sistema de base de datos ocurre una falla física del sistema rompiéndose el disco donde se almacenaba la base de datos. La base de datos se pierde, pero no se pierden los archivos de log, que se encontraban en otro disco. Existe respaldo de la base de datos hasta el día 23/5/2021 incluido. **El sistema de base de datos trabaja con el mecanismo de Actualización Diferida.**

1. Describir los pasos que se deben ejecutar para recuperar la base de datos dejándola al día y en un estado consistente.
2. ¿Qué diferencia habría en la recuperación (resuelta en la parte a) si el sistema trabajara con Actualización Inmediata?

- 1.Backup, nos fijamos el log, redu.
- 2.No tenemos diferencias, es lo mismo.

CLASE DEL 26/5/2021

Tenemos clase hasta 16 de junio.

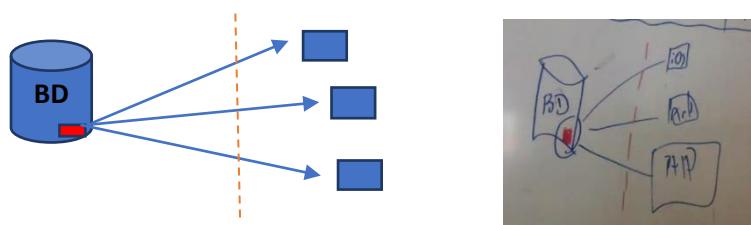
Laboratorio: semana que viene correcciones.

¿Cuál es la razón para escribir una función en programación?

Una de las razones es para no repetir código.

TEMA NUEVO

Procedimientos almacenados y triggers.



Lo que yo resuelvo dentro de la base de datos ya no tengo que resolverlo afuera.

Disparadores o triggers

BASES DE DATOS II

PROCEDIMIENTOS ALMACENADOS

En PostgreSQL a diferencia de otros motores de bases de datos, no existe una distinción explícita entre una función y un procedimiento almacenado. En PostgreSQL solo existen funciones, claro que estas pueden ser usadas a modo de una función o de un procedimiento almacenado. Además de esta diferencia con otros motores de bases de datos, es importante mencionar que **PostgreSQL nos ofrece más de un lenguaje para crear nuestros procedimientos almacenados**, pudiendo elegir entre los siguientes lenguajes:

- PL/PgSQL
- C.
- C++.
- Java PL/Java web.
- PL/Perl.
- plPHP.
- PL/Python.
- PL/Ruby.
- PL/sh.
- PL/Tcl.
- PL/Scheme.

En el curso de Bases de Datos II usaremos **PL/PgSQL** pues es un lenguaje procedural basado en SQL y que sigue el estándar ANSI SQL.

Lo tenemos que especificar:

```
CREATE FUNCTION suma (p1 integer default 0, p2 integer default 0)
returns integer
AS $$
begin
    return p1+p2;
end
$$
language 'plpgsql'
```

retorna la suma de los dos parámetros. En la última línea específica.

En este **ejemplo** si no existe la función la crea: con OR REPLACE FUNCION

```
CREATE OR REPLACE FUNCTION sp_indepyear(IN character)
RETURNS integer AS $$

DECLARE
    iNombre integer;
BEGIN
    select indepyear FROM country where code=$1 INTO iNombre;
    if (iNombre>0) then
        return iNombre;
    else
        return 0;
    end if;
END;
$$ LANGUAGE plpgsql
```

IN significa que el parámetro no tiene nombre, dice que el parámetro es de entrada, por defecto son de entrada, por buena práctica se pone.

Declara una variable, la se antepone el i antes del nombre de la variable ese i significa int.

\$1 hace referencia al IN.

Lo guarda en la variable creada.

TEORICO

Procedimientos Almacenados (Stored Procedures)

- Es una porción de código, que se almacena en el catálogo de la base de datos y se puede invocar mediante una sentencia SQL.
- Se ejecuta en el servidor de base de datos. Reduce el número de idas y vueltas entre aplicaciones y el servidor de base de datos.
- Encapsulan reglas de negocio fuertemente relacionadas con los datos de la BD y sin interacción con el usuario
- No es obligatorio que estén escritos en SQL (Java, PL/SQL, PL/pgSQL, Transact SQL)
- Cada RDBMS tiene su propio lenguaje de Stored Procedure, los cuales incluyen sentencias de control, manejo de variables, etc.
- Los Stored Procedures, tienen un nombre, reciben parámetros y pueden devolver resultados
- Permite reutilizar código entre diversas aplicaciones

Un metadato.

Gana en optimización.

Procedimientos Almacenados (Stored Procedures)

- Velocidad de desarrollo más lenta.
- Requiere habilidades especiales
- Es difícil manejar versiones y es más complejo depurar
- No son portables entre diferentes sistemas de bases de datos.

Sintaxis

```
CREATE OR REPLACE FUNCTION suma(p1 integer DEFAULT 0, p2 integer
                               DEFAULT 0) RETURNS integer
LANGUAGE 'plpgsql' AS
$BODY$
BEGIN
    return p1 + p2;
END
$BODY$;
```

Llamado

```
select suma(2,2);
```

Se hace un select del proceso almacenado con los parámetros que queremos.

EJEMPLO:

¿Qué hace ese proceso almacenado cuando se ejecuta?

```
CREATE PROCEDURE AUDITORIA
AS
DECLARE VARIABLE TABLA VARCHAR(80);
BEGIN
  FOR SELECT RDB$RELATION_NAME FROM RDB$RELATIONS
    WHERE RDB$VIEW_SOURCE IS NULL
      AND RDB$RELATION_NAME NOT LIKE 'RDB$%'
      into :tabla DO
    BEGIN
      EXECUTE STATEMENT 'ALTER TABLE ' ||
        :tabla ||' ADD FECHA_AUD DATE, ADD HORA_AUD TIME';
    END
  END^
SET TERM ; ^ ]
```

Para cada tabla en la base de datos me guarda el nombre de esa tabla en el campo tabla.

Lo único que hace es crear dos campos en todas las tablas de la base de datos.

En pgAdmin se ven así:

The screenshot shows the pgAdmin interface. On the left, the schema browser tree is visible, with the 'Functions' node under the 'public' schema highlighted in blue. Below the tree, there is a list of functions. One function, 'suma', is also highlighted in blue. At the bottom of the tree, there is a 'Materialized Views' node. In the center, there is a 'query Editor' tab with the following SQL query:

```
1 select suma(20,50);
```

A blue arrow points from the 'suma' function in the tree to the 'suma' function in the query editor. Another blue arrow points from the 'suma' function in the query editor to the result table below. The result table has one row with two columns: 'suma' and 'integer'. The value '70' is displayed in the 'suma' column.

suma	integer
1	70

Un proceso almacenado que nos devuelve una tabla.

```
CREATE OR REPLACE FUNCTION DatosPaises4(IN character)
RETURNS TABLE (codigo character(3), nombre character(52)) AS
$$
BEGIN
RETURN QUERY SELECT country.code , country.name FROM country WHERE country.code = $1;
END;
$$
LANGUAGE 'plpgsql';
```

Un ejemplo por parámetros out de salida

```
CREATE or replace FUNCTION listar_paises_OUT
(OUT out_id character, OUT out_nombre character)
RETURNS SETOF RECORD AS
$BODY$
DECLARE
    reg RECORD;
BEGIN
    FOR REG IN SELECT * FROM country LOOP
        out_id      := reg.code;
        out_nombre := reg.name;
        RETURN NEXT;
    END LOOP;
    RETURN;
END
$BODY$ LANGUAGE 'plpgsql'
```

EJERCICIOS

1. Crear una función llamada 'media' que reciba 3 números enteros y devuelva el promedio de los mismos.
2. Ahora crearemos una función que retorne el código de un país que se pasa por parámetro.

1.

```
CREATE OR REPLACE FUNCTION promedio (p1 integer default 0, p2 integer default 0, p3  
integer default 0)
```

```
RETURNS INTEGER
```

```
AS $$
```

```
BEGIN
```

```
RETURN (p1+p2+p3)/3;
```

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

```
SELECT promedio (2,2,2);
```

Ventajas de los procedimientos almacenados

VENTAJAS DE USAR SP

- Compilación: La primera vez que se invoca un SP, el motor lo compila y a partir de ahí, se sigue usando la versión compilada del mismo, hasta que se modifique. Esto significa que se tendrá un mejor rendimiento que las consultas directas que usan cadenas con las instrucciones , que se compilan cada vez que se invocan.

Es decir, va a funcionar más rápido.



CLASE DEL 02/06

EJEMPLO DE CONDICIONAL

```
CREATE FUNCTION espositivo (int4)
RETURNS boolean
AS $$ 
DECLARE
    a int4;
    res boolean := false;
BEGIN
    a := $1;
    IF (a > 0) THEN
        res := true;
    ELSE
        IF (a = 0) THEN
            res := true;
        END IF;
    END IF;
    RETURN res;
END;
$$
Language 'plpgsql';
```

EJERCICIO FIBONACCI

```
create function fib2(n integer)
```

```
    returns integer
```

```
        as $$
```

```
begin
```

```
    if n = 0 then
```

```
        return 0;
```

```
    elsif n = 1 then
```

```
        return 1;
```

```
    else
```

```
        return fib2(n - 1) + fib2(n - 2);
```

```
    end if;
```

```
end
```

```
$$
```

```
language 'plpgsql';
```

```
SELECT fib2(4);
```

Pero lo que nos interesa es trabajar con los datos de una tabla. Entonces veamos un ejemplo más práctico.

Supongamos que tenemos una tabla **item** que almacena la información de piezas de hardware vendidas por una tienda de computadoras.

```
CREATE TABLE item (
    item_id      serial NOT NULL,
    nombre        varchar(150) NOT NULL,
    tipo          varchar(100) NOT NULL,
    cantidad      int4 NOT NULL DEFAULT 0,
    precio_compra numeric(7,2) NOT NULL,
    precio_venta numeric(7,2) NOT NULL,
    CONSTRAINT item_id_pk PRIMARY KEY (item_id)
);
```

```
CREATE TABLE item(
    item_id serial NOT NULL,
    nombre varchar(150) NOT NULL,
    tipo varchar(100) NOT NULL,
    cantidad int4 NOT NULL,
    precio_compra numeric(7,2) NOT NULL,
    precio_venta numeric(7,2) NOT NULL,
    CONSTRAINT item_id_pk PRIMARY KEY (item_id)
);
```

Probemos insertando algunas filas en la tabla item:

```
INSERT INTO item (nombre, tipo, cantidad, precio_compra, precio_venta)
VALUES ('Switch Dlink 8 puertos', 'Switch', 2, 32.5, 45.6);
INSERT INTO item (nombre, tipo, cantidad, precio_compra, precio_venta)
VALUES ('Dlink 56K', 'Modem', 0, 10.5, 15.5);
INSERT INTO item (nombre, tipo, cantidad, precio_compra, precio_venta)
VALUES ('Samsung 17', 'Monitor', 0, 100.0, 120.0);
INSERT INTO item (nombre, tipo, cantidad, precio_compra, precio_venta)
VALUES ('DDR2 512/533', 'RAM', 15, 40.0, 45.0);
INSERT INTO item (nombre, tipo, cantidad, precio_compra, precio_venta)
VALUES ('NVIDIA GEFORCE FX 5200', 'Tarjeta de Video', 7, 40.0, 48.5);
INSERT INTO item (nombre, tipo, cantidad, precio_compra, precio_venta)
VALUES ('Pentium 4 3.2 GHZ 800/2MB', 'Procesador', 7, 200.0, 230.0);
```

```
INSERT INTO item (nombre, tipo, cantidad, precio_compra, precio_venta)
VALUES ('Switch Dlink 8 puertos', 'Switch', 2, 32.5, 45.6);
INSERT INTO item (nombre, tipo, cantidad, precio_compra, precio_venta)
VALUES ('Dlink 56K', 'Modem', 0, 10.5, 15.5);
INSERT INTO item (nombre, tipo, cantidad, precio_compra, precio_venta)
```

```
VALUES ('Samsung 17','Monitor', 0, 100.0, 120.0);
INSERT INTO item (nombre, tipo, cantidad, precio_compra, precio_venta)
VALUES ('DDR2 521/533', 'RAM', 15, 40.0, 45.0);
INSERT INTO item (nombre, tipo, cantidad, precio_compra, precio_venta)
VALUES ('NVIDIA GEFORCE FX 5200', 'Tarjeta de Video', 7, 40.0, 48.5);
INSERT INTO item (nombre, tipo, cantidad, precio_compra, precio_venta)
VALUES ('Pentium 4 3.2 GHZ 800/2MB', 'Procesador', 7, 200.0, 230.0);
```

ALGO NUEVO DE PROCESOS ALMACENADO.

Empezamos a definir la función:

```
CREATE OR REPLACE FUNCTION comprar_item()
RETURNS int4
AS $$
DECLARE+
    cont_item int4;
    fila_item item%ROWTYPE;
BEGIN
    ...
END;
$$
Language 'plpgsql';
```

Declara una variable que solo va a contar.

Declara una variable del tipo de la tabla (item) y ROWTYPE significa.

Podemos declarar variables compuestas que puedan almacenar los campos de una determinada fila de una determinada tabla.

```
fila_item item%ROWTYPE ;
```

Indica que la variable **fila_item** podrá almacenar los campos de cualquier fila de la tabla item.

Para acceder a los valores de la variable **fila_item** usamos una notación ya conocida:

```
fila_item.nombre
fila_item.cantidad
fila_item.precio_compra
```

Ahora:

```

CREATE OR REPLACE FUNCTION comprar_item()
RETURNS int4
AS $$ 
DECLARE
    cont_item int4;
    fila_item item%ROWTYPE;
BEGIN
    DROP TABLE item_por_comprar; +
    CREATE TABLE item_por_comprar (item_id int4 NOT NULL,
                                    nombre varchar(150));
END;
$$
Language 'plpgsql';

```

Cada vez que vayamos a ejecutar la función borraremos la anterior tabla y crearemos una nueva con datos actualizados. No nos interesa almacenar la información que se genera cada vez que ejecutamos la función, solo la información más reciente.

Ahora usamos el recurso de la clase anterior, selecciono los item que tiene cantidad cero y los cuenta y los guarda en la cont_item.

```

CREATE OR REPLACE FUNCTION comprar_item()
RETURNS int4
AS $$ 
DECLARE
    cont_item int4;
    fila_item item%ROWTYPE;
BEGIN
    DROP TABLE item_por_comprar;
    CREATE TABLE item_por_comprar (item_id int4 NOT NULL,
                                    nombre varchar(150));
    SELECT COUNT(*) INTO cont_item FROM item WHERE cantidad = 0;
END;
$$
Language 'plpgsql';

```

Ahora:

Vimos que podíamos definir un bucle con la sentencia FOR de esta forma:

```
FOR variable IN rango
```

Si trabajamos con variables compuestas y tablas, podemos colocar en **variable** una variable compuesta y en **rango** una tabla cualquiera (o un subconjunto), la variable compuesta recorrerá toda la tabla tomando los valores de cada una de las filas, si tienen los mismos atributos por supuesto.

```
FOR fila_item IN SELECT * FROM item WHERE cantidad = 0;
```

Ahora:

Usamos un FOR, estamos usando la segunda variable que declaramos la item.

Es parecido a un FOREACH dice el profe.

En cada vuelta del for, para cada fila que haya en la table item cuando cantidad es cero, hace un insert en item_por_comprar

Accede a los valores de la table.

```
CREATE OR REPLACE FUNCTION comprar_item()
RETURNS int4
AS $$ 
DECLARE
    cont_item int4;
    fila_item item%ROWTYPE;
BEGIN
    DROP TABLE item_por_comprar;
    CREATE TABLE item_por_comprar (item_id int4 NOT NULL,
                                    nombre varchar(150));
    SELECT COUNT(*) INTO cont_item FROM item WHERE cantidad = 0;
    FOR fila_item IN SELECT * FROM item WHERE cantidad = 0
        LOOP
            INSERT INTO item_por_comprar
                VALUES (fila_item.item_id, fila_item.nombre);
        END LOOP;
    END;
$$
Language 'plpgsql';
```

Ahora, solo nos queda agregarle el RETURN:

```
CREATE OR REPLACE FUNCTION comprar_item()
RETURNS int4
AS $$ 
DECLARE
    cont_item int4;
    fila_item item%ROWTYPE;
BEGIN
    DROP TABLE item_por_comprar;
    CREATE TABLE item_por_comprar (item_id int4 NOT NULL,
                                    nombre varchar(150));
    SELECT COUNT(*) INTO cont_item FROM item WHERE cantidad = 0;
    FOR fila_item IN SELECT * FROM item WHERE cantidad = 0
        LOOP
            INSERT INTO item_por_comprar
                VALUES (fila_item.item_id, fila_item.nombre);
        END LOOP;
    RETURN cont_item; 
END;
$$
Language 'plpgsql';
```

Nos devuelve cont pero por dentras hace mas cosas.

CONTINUACION DEL EJERCICIO

Esta parte result de lo que vimos antes, fuimos viendo el teorico y haciendolo.

Queremos que haya un función **comprar_item()** que haga lo siguiente:

1. Cree una tabla nueva llamada **item_por_comprar** que almacene los datos de los items que tengan **cantidad = 0**, o sea los items que hay que renovar para luego poner a la venta.
2. Busque en la tabla **item** todos los items que ya no estén en stock y los inserte en la tabla **item_por_comprar**
3. Nos devuelva la cantidad de items que ya no se encuentran en stock

```
CREATE OR REPLACE FUNCTION comprar_item()
```

```
RETURNS int4
```

```
AS $$
```

```
DECLARE
```

```
    cont_item int4;
```

```
    fila_item item%ROWTYPE;
```

```
BEGIN
```

```
    DROP TABLE item_por_comprar;
```

```
    CREATE TABLE item_por_comprar (
```

```
        item_id int4 NOT NULL,
```

```
        nombre varchar(150)
```

```
);
```

```
    SELECT COUNT(*) INTO cont_item FROM item WHERE cantidad = 0;
```

```
    FOR fila_item IN SELECT * FROM item WHERE cantidad = 0
```

```
        LOOP
```

```
            INSERT INTO item_por_comprar
```

```
                VALUES(fila_.item_id, fila_item.nombre);
```

```
        END LOOP;
```

```
    RETURN cont_item;
```

```
END;
```

```
$$
```

```
LANGUAGE 'plpgsql';
```

Primero crear la table por las dudas:

```
CREATE TABLE item_por_comprar (
```

```
    item_id int4 NOT NULL,  
    nombre varchar(150)  
);
```

La prueba:

```
select comprar_item();
```

ULTIMO EJERICICO DEL DIA:

Utilicemos la misma tabla **item** y hagamos una función **patrimonio()** que nos retorne:

- El valor total de todos los items disponibles en la tienda.

o sea :

+ cantidad x precio_venta (de cada item)

```
CREATE OR REPLACE FUNCTION patrimonio()  
RETURNS int4  
AS $$  
DECLARE  
    fila_item item%ROWTYPE;  
    resultado int4;  
BEGIN  
    resultado := 0;  
    FOR fila_item IN SELECT * FROM item  
  
        LOOP  
            resultado = resultado + (fila_item.cantidad * fila_item.precio_venta);  
        END LOOP;  
    RETURN resultado;  
END;  
$$  
LANGUAGE 'plpgsql';  
SELECT patrimonio();
```

SOLUCION

```
CREATE OR REPLACE FUNCTION patrimonio()
RETURNS numeric(7,2)
AS $$ 
DECLARE
    total numeric(7,2);
    fila_item item%ROWTYPE;
BEGIN
    total := 0.0;
    FOR fila_item IN SELECT * FROM item WHERE cantidad != 0
    LOOP
        total := total + fila_item.cantidad * fila_item.precio_venta ;
    END LOOP;
    RETURN total;
END;
$$
Language 'plpgsql';
```

Veamos como el tipo de retorno de la funcion ha cambiado, ya no es **int4** sino **numeric**.

HACER PRACTICO:

VEMOS UNAS ESTRUCTURAS:

WHILE

```
CREATE FUNCTION adicion()
RETURNS int4
AS $$ 
DECLARE
    cont int4;
    res int4;
BEGIN
    cont := 1;
    res := 0;
    WHILE (cont <= 10)
        LOOP
            res := res + cont;
            cont := cont + 1;
        END LOOP;
    RETURN res;
END;
$$
Language 'plpgsql';
```

FOR

```
CREATE FUNCTION adicion2()
RETURNS int4
AS $$ 
DECLARE
    cont int4;
    res int4;
BEGIN
    res := 0;
    FOR cont IN 1 .. 10
        LOOP
            res := res + cont;
        END LOOP;
    RETURN res;
END;
$$
Language 'plpgsql';
```

CLASE DEL 7/6/21 LUNES

Notas del practico:

El 5, da problemas.

Laboratorio:

Documentación, del proceso almacenador, objetivos que parámetros recibe que retorna y de que tipo. Decir que hace, que recibe y que devuelve.

EJEMPLO:

1. Procedimiento registrar

Este procedimiento permite registrar una nueva cuenta free. Las validaciones que tiene este procedimiento son que los campos no estén vacíos, y que el email recibido no pertenezca a una cuenta existente.

Nota: La contraseña se recibe codificada en base64

URL para llamar: [/SatWeb-Backend/Index_controller/registrar](#)

Método: POST

Codificación: JSON

Ejemplo codificado en JSON:

Valor enviado:

`{"username": "fede@fede.com", "password": "Mltz"}`

Valor Devuelto:

`{"status": "ok", "message": "Cuenta registrada correctamente"}`

`{"status": "error", "message": "Email ya registrado"}`

TEMA NUEVO

Crear y observar la salida del siguiente store procedure (base de datos World.sql):

```
CREATE or Replace FUNCTION sp_incrementar_population(pid integer) RETURNS numeric AS $
$<<principal>>
DECLARE
    incremento numeric := (SELECT population FROM city WHERE id = $1) * 0.3;
BEGIN
    RAISE NOTICE 'Aumento de %', incremento;
    <<excepcional>>
    DECLARE
        incremento numeric := (SELECT population FROM city WHERE id = $1) * 0.5
    ;
    BEGIN
        RAISE NOTICE 'Aumento excepcional de %', incremento;
        RAISE NOTICE 'Aumento principal de %', principal.incremento;
    END;
    RAISE NOTICE 'El aumento después de incremento será de %', incremento;
    RETURN incremento;
END;
$$ LANGUAGE plpgsql;
```

RAISE: Manera de dar mensaje

```
NOTICE: Aumento de 56040.0
NOTICE: Aumento excepcional de 93400.0
NOTICE: Aumento principal de 56040.0
NOTICE: El aumento después de incremento será de 56040.0

Successfully run. Total query runtime: 233 msec.
1 rows affected.
```

En zona principal declara una variable que es.

Y manda un mensaje que dice que le va a aumentar, y es el primer mensaje que vemos.

Otra zona que tiene una variable con otro valor.

La variable incremento se declara en principal con un valor, y en la zona excepcional tiene otro valor. Pero si yo la emboco con **principal.incremento** nos da la que creamos en principal.

Scope, ámbito. Si declaramos un variable fuera y otra dentro con distinto valor. <>>

MENSAJES

Mensajes

En ejemplos analizados previamente se ha hecho uso de mensajes utilizando la cláusula `RAISE` con la opción `NOTICE`. Además de esta opción, la cláusula `RAISE` permite las opciones `DEBUG`, `LOG`, `INFO`, `WARNING` y `EXCEPTION`, esta última utilizada por defecto:

- **`NOTICE`**: es utilizado para hacer notificaciones.
- **`WARNING`**: es utilizado para hacer advertencias.
- **`LOG`**: es utilizado para dejar constancia en los logs de PostgreSQL del mensaje o error.
- **`EXCEPTION`**: es utilizado para lanzar una excepción, cancelándose todas las operaciones realizadas previamente en la función.

Un ejemplo de exception, en una función cuando una condición no se cumple, tira una excepción. Otro ejemplo dividir por cero.

Ejemplo:

```
CREATE or replace FUNCTION sp_pais(character) RETURNS character AS $$  
DECLARE  
    sPais character(52);  
BEGIN  
    SELECT country.name INTO sPais FROM country WHERE code = $1;  
    IF not found THEN  
        RAISE EXCEPTION 'País % no encontrado', $1;  
    END IF;  
    RETURN sPais;  
END;  
$$ LANGUAGE plpgsql;
```

Retorno de Valores

1. Uno de los tipos de datos definidos en el estándar SQL o por el usuario, por ejemplo, `VARCHAR` para devolver la ciudad en que vive el cliente "Brian Daniel Vázquez López", `INTEGER` para devolver la edad del cliente "Lillian Pozo Ortiz", **VOID para aquellas funciones que no retornen un valor usable, un tipo de dato compuesto** (ejemplo `PRODUCTS` para devolver una tupla de la tabla `products` de la base de datos), `RECORD` para retornar una fila resultante de un subconjunto de las columnas de una tabla o de concatenaciones entre tablas, etc.
2. Un conjunto de tuplas: mediante la especificación del tipo de retorno "`SETOF algún_tipo`", o de forma equivalente declarando el tipo de retorno "`TABLE (columnas)`", en cuyo caso todas las tuplas de la última consulta son retornadas.
3. El valor nulo: en caso de que la consulta no retorne ninguna fila.

Tipo de dato RECORD

PL/pgSQL soporta el tipo de dato `RECORD`, similar a `ROWTYPE` pero sin estructura predefinida, la que toma de la fila actual asignada durante la ejecución del comando `SELECT` o `FOR`.

`RECORD` no es un tipo de dato verdadero sino un contenedor. Este tipo de dato no es el mismo concepto que cuando se declara una función para que retorne un tipo `RECORD`; en ambos casos la estructura de la fila actual es desconocida cuando la función está siendo escrita, pero para el retorno de una función la estructura actual es determinada cuando la llamada es revisada por el analizador sintáctico, mientras que la de la variable puede ser cambiada en tiempo de ejecución.

Puede ser utilizado para devolver un valor del que no se conoce tipo de dato, pero sí se debe conocer su estructura cuando se quiere acceder a un valor dentro de él, por ejemplo para acceder a un valor de una variable de tipo `RECORD` se debe conocer previamente el nombre del atributo para poder calificarlo y acceder al mismo.

Triggers

Evento al que está asociado un trigger:

- **ON INSERT**: Se dispara el trigger al ejecutarse una instrucción INSERT sobre la tabla a la cual está asociado.
- **ON UPDATE**: Se dispara el trigger al ejecutarse una instrucción UPDATE sobre la tabla a la cual está asociado.
- **ON DELETE**: Se dispara el trigger al ejecutarse una instrucción DELETE sobre la tabla a la cual está asociado.

El alcance del trigger:

- **POR FILA**: El trigger se ejecuta para cada una de las filas afectadas por la sentencia por la que fue disparado.
- **POR SENTENCIA**: El trigger se ejecuta una sola vez para la sentencia que lo disparó (sin importar cuantas filas sean afectadas).

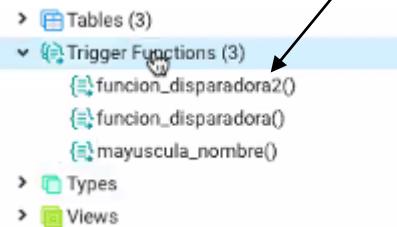
El momento en que se dispara el trigger:

- **ANTES**: Los triggers de este tipo se ejecutan antes de que la sentencia se ejecute o de que esta afecte una fila en particular (dependiendo de si es por sentencia o por fila).
- **DESPUÉS**: Los triggers de este tipo se ejecutan después de que la sentencia se haya ejecutado o de que haya afectado una fila en particular (dependiendo si es por sentencia o por fila).

EJEMPLO DE TRIGGERS

creamos la función disparadora

```
create or replace function funcion_disparadora2()
RETURNS trigger AS
$$
BEGIN
    RAISE NOTICE 'HOLA MUNDO SOY UN TRIGGER';
    RETURN null;
END;
$$
LANGUAGE 'plpgsql';
```



En postgres primero hacemos la una función disparadora, y después:

```
CREATE TRIGGER trigger_clase
AFTER UPDATE
ON city
FOR EACH ROW
EXECUTE PROCEDURE funcion_disparadora2();
```

Evento
Alcance

Para probar una actualización simple:

```
update city set name='pepe' WHERE id=3;
```

Miramos los mensajes:

```
NOTICE: HOLA MUNDO SOY UN TRIGGER
NOTICE: Un disparador ha sido invocado
UPDATE 1

Query returned successfully in 331 msec. [I]
```

Cada vez que alguien hace una actualización, ahí un trigger que invoca a función_diparadora2, ahora solo muestra un mensaje, pero podría hacer otras cosas.

OTRO EJEMPLO:

```
CREATE TRIGGER trigger_dos
AFTER UPDATE or DELETE
ON city
FOR EACH ROW
EXECUTE PROCEDURE funcion_disparadora();
```

El evento puede ser mas de uno combinados, ahora tenemos el ejemplo de cuando actualice o elimine.

CLASE DEL 9/6/21

TRABAJO EN GRUPO

Entonces cuales son las acciones que debe realizar nuestro trigger?

- Necesitamos llevar un registro de todos los cambios relevantes en nuestra tabla **item**, para empezar queremos tener un seguimiento de los cambios en los precios de los items.
- También queremos almacenar en algún lugar todos aquellos items que hayan sido eliminados, también queremos guardar sus respectivos historiales de cambios.

Una función disparadora que, ante un cambio en los precios, guarde el registro del precio venta que está cambiando. Lo guarde en otra tabla, la tabla que paso el profe.

```
CREATE TABLE item_actualizado (
    item_id          int4 NOT NULL,
    nombre            varchar(150) NOT NULL,
    precio_anterior   numeric(7,2) NOT NULL,
    precio_actualizado numeric(7,2) NOT NULL,
    autor             varchar(100) NOT NULL,
    fecha_cambio      date NOT NULL,
    CONSTRAINT item_id_fk FOREIGN KEY (item_id) REFERENCES item(item_id)
);
```

Solo en el update.

SOLUCIÓN:

Usando las tablas de la clase anterior.

Primero hacemos la tabla:

```
CREATE TABLE item_actualizado (
    item_id int4 NOT NULL,
    nombre varchar(150) NOT NULL,
    precio_anterior numeric(7,2) NOT NULL,
    precio_actualizado numeric(7,2) NOT NULL,
    autor varchar(100) NOT NULL,
    fecha_cambio date NOT NULL,
    CONSTRAINT item_id_fk FOREIGN KEY (item_id) REFERENCES item(item_id)
);
```

Hacemos la función:

```
CREATE OR REPLACE FUNCTION modificar_item2()
RETURNS trigger AS
$$
DECLARE
    file_item item%ROWTYPE;
BEGIN
    RAISE NOTICE 'Se realizo un cambio';

    INSERT INTO item_actualizado VALUES (OLD.item_id, OLD.nombre,
                                          OLD.precio_venta, NEW.precio_venta,
                                          current_user, current_date);
    RETURN null;
END;
$$
LANGUAGE 'plpgsql';
```

Hacemos el disparador:

```
CREATE TRIGGER trigger_modifica2
AFTER UPDATE
on item
FOR EACH ROW
EXECUTE PROCEDURE modificar_item2();
```

Probamos:

```
UPDATE item set precio_venta = 50 WHERE item_id=1;
```

Nos fijamos en la tabla item_actualizado:

	item_id integer	nombre character varying (150)	precio_anterior numeric (7,2)	precio_actualizado numeric (7,2)	autor character varying (100)	fecha_cambio date
1	1	Hola???	45.60	50.00	postgres	2021-06-09

SOLUCION DEL PROFESOR:

Contemplo que, si el precio nuevo es distinto al anterior, si es así hace el insert, de lo contrario no lo hace.

```
CREATE FUNCTION item_actualizado_tri()
RETURNS trigger
AS $$
BEGIN
    IF (TG_OP = 'UPDATE') THEN
        IF (OLD.precio_venta != NEW.precio_venta) THEN
            INSERT INTO item_actualizado VALUES (
                OLD.item_id, OLD.nombre, OLD.precio_venta,
                NEW.precio_venta, current_user, current_date);
        END IF;
    END IF;
    RETURN NULL;
END;
$$
Language 'plpgsql';
```

Vamos a la definicion formal del trigger:

```
CREATE TRIGGER actualizar_item
AFTER UPDATE ON item
FOR EACH ROW
EXECUTE PROCEDURE item_actualizado_tri();
```

Listo, cada vez que se actualice una tupla en la tabla **item**, se reflejará el cambio en la tabla **item_actualizado**. Pero era eso lo que queríamos?, no totalmente, volvamos a la función del trigger.

EJERCICIO 2:

Usando lo anterior.

Ejercicios:

- A la tabla **item_actualizado** añadir un atributo **hora_cambio** que almacene la hora de la modificación.
- Hacer otro trigger que haga el mismo tratamiento pero con los items eliminados de la tabla **item**.

Parte 1 Modificar la tabla y el trigger.

```
ALTER TABLE item_actualizado ADD COLUMN hora_cambio time;

CREATE OR REPLACE FUNCTION modificar_item2()
RETURNS trigger AS
$$
DECLARE
    file_item item%ROWTYPE;
BEGIN
    RAISE NOTICE 'Se realizo un cambio';
    IF(TG_OP = 'UPDATE') THEN
        IF (OLD.precio_venta != NEW.precio_venta)THEN
            INSERT INTO item_actualizado VALUES (OLD.item_id, OLD.nombre,
                                                OLD.precio_compra, NEW.precio_venta,
                                                current_user, current_date, current_time);
        END IF;
        END IF;
        RETURN null;
    END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER trigger_modifica2
AFTER UPDATE
on item
FOR EACH ROW
EXECUTE PROCEDURE modificar_item2();
```

Parte 2

Primero creamos la tabla item_eliminado

```
CREATE TABLE item_eliminado (
    item_id      int4 NOT NULL,
    nombre       varchar(150) NOT NULL,
    precio_compra numeric(7,2) NOT NULL,
    precio_venta numeric(7,2) NOT NULL,
    autor        varchar(100) NOT NULL,
    fecha_eliminación date NOT NULL,
    hora_eliminacion timetz NOT NULL
);
```

```
CREATE FUNCTION item_eliminado_tri()
RETURNS trigger
AS $$
BEGIN
    IF (TG_OP = 'DELETE') THEN
        INSERT INTO item_eliminado VALUES (
            OLD.item_id, OLD.nombre, OLD.precio_compra, OLD.precio_venta,
            current_user, current_date, current_time);
    END IF;
    RETURN NULL;
END;
$$
Language 'plpgsql';
```

```
CREATE TRIGGER eliminar_item
AFTER DELETE ON item
FOR EACH ROW
EXECUTE PROCEDURE item_eliminado_tri();
```

ULTIMO EJERCICIO:

Me cree una nuevo BD llamada Ejerci

Nos dio un .sql:

```
CREATE TABLE numeros(
    numero bigint NOT NULL,
    cuadrado bigint,
    cubo bigint,
    raiz2 real,
    PRIMARY KEY (numero) );|
```

Probar:

power(x,y): retorna el valor de "x" elevado a la "y" potencia. Ejemplo:

```
select power(2,3)
```

sqrt(x): devuelve la raíz cuadrada del valor enviado como argumento. Ejemplo:

```
select sqrt(9)
```

SOLUCION



Hacemos la tabla, y hacemos la función:

```
CREATE FUNCTION calcular()
RETURNS trigger
AS $$ 
BEGIN
    IF(tg_op = 'INSERT') then
        UPDATE numeros SET cuadrado = power(new.numero,2), cubo = power(new.numero,3),
                           raiz2 = sqrt(new.numero) WHERE new.numero = numero;

    END IF;
    RETURN null;
END;
$$
Language 'plpgsql';
```

Hacemos el disparador:

```
CREATE TRIGGER calcula
AFTER INSERT ON numeros
FOR EACH ROW
EXECUTE PROCEDURE calcular();
```

Probamos:

29	INSERT INTO numeros(numero) VALUES (9);
30	
31	select * from numeros;

Data Output Explain Messages Notifications

	numero [PK] bigint	cuadrado bigint	cubo bigint	raiz2 real	
1	2	4	8	1.41421	
2	9	81	729	3	

SOLUCION DEL PROFESOR:

Función:

```
CREATE OR REPLACE FUNCTION proteger_y_rellenar_datos()
RETURNS TRIGGER AS
$$
DECLARE
BEGIN
    IF (TG_OP = 'INSERT' OR TG_OP = 'UPDATE' ) THEN
        NEW.cuadrado := power(NEW.numero,2);
        NEW.cubo := power(NEW.numero,3);
        NEW.raiz2 := sqrt(NEW.numero);

        RETURN NEW;
    ELSEIF (TG_OP = 'DELETE') THEN
        RETURN NULL;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Disparador:

```
CREATE TRIGGER proteger_y_rellenar_datos
BEFORE INSERT OR UPDATE OR DELETE ON numeros
FOR EACH ROW
EXECUTE PROCEDURE proteger_y_rellenar_datos();
```

