**CONSTRAINT / ASSERTION/CHEK**

/\* no puede haber vol de mas de 70\*/  
alter table voluntariov  
add constraint ck\_edad\_voluntariov  
check ( *extract*(year from fecha\_nacimiento)>1954);  
  
alter table voluntariov drop constraint ck\_edad\_voluntariov;  
/\*B. Ningún voluntario puede aportar más horas que las de su coordinador. \*/  
create assertion add constraint ck\_horas\_aportadas\_voluntariov  
check ( not exists  
 (select 1  
 from voluntario jefe  
 join voluntario empleado on jefe.id\_coordinador = empleado.nro\_voluntario  
 where empleado.horas\_aportadas > jefe.horas\_aportadas  
) );  
  
/\*C. Las horas aportadas por los voluntarios deben estar dentro de los valores máximos y  
mínimos consignados en la tarea.\*/  
create assertion  
add constraint ck\_horas\_max\_min\_vol\_tarea  
check ( not exists  
 (select 1  
 from voluntario v join tarea t on v.id\_tarea = t.id\_tarea  
 where v.horas\_aportadas between t.min\_horas and t.max\_horas  
 ) );  
alter table voluntario drop constraint ck\_horas\_max\_min\_vol\_tarea;  
  
/\*D. Todos los voluntarios deben realizar la misma tarea que su coordinador.\*/  
create assertion  
add constraint ck\_tarea\_coodinador  
check ( not exists(  
 select 1  
 from voluntario c join voluntario v on c.id\_coordinador = v.nro\_voluntario  
 where v.id\_tarea = c.id\_tarea  
) );  
  
/\*E. Los voluntarios no pueden cambiar de institución más de tres veces al año.\*/  
create assertion  
add constraint ck\_vol\_historico  
check ( not exists  
 (select h.id\_institucion,v.nro\_voluntario, count()  
 from voluntario v join historico h on v.nro\_voluntario = h.nro\_voluntario  
 where extract(year from fecha\_fin) = extract(year from fecha\_inicio)  
 group by h.id\_institucion,v.nro\_voluntario  
 having count(\*) > 3  
) );  
/\*F. En el histórico, la fecha de inicio debe ser siempre menor que la fecha de finalización. \*/  
alter table historico  
add constraint ck\_his\_fecha\_inicio  
check (historico.fecha\_fin > historico.fecha\_inicio)  
  
/\*Considere las siguientes restricciones que debe definir sobre el esquema de la BD de Películas:  
A. Para cada tarea el sueldo máximo debe ser mayor que el sueldo mínimo.  
B. No puede haber más de 70 empleados en cada departamento.  
 \*/  
set search\_path = unc\_esq\_peliculas;  
alter table tarea  
add constraint tarea\_fecha  
check ( not *exists*(  
select t.id\_tarea  
from tarea t  
where t.sueldo\_maximo > t.sueldo\_minimo ))  
  
alter table empleado  
add constraint ck\_empleado\_departamento\_70  
check ( not *exists*(  
 select e.id\_departamento,*count*(e.id\_empleado)  
from empleado e  
group by e.id\_departamento  
having *count*(e.id\_empleado) < 70  
))  
  
/\*C. Los empleados deben tener jefes que pertenezcan al mismo departamento. \*/  
create assertion  
add constraint ck\_empleado\_jefe  
check ( not exists(  
 select e.id\_empleado  
 from empleado e join empleado j on e.id\_jefe = j.id\_empleado  
 where e.id\_departamento != j.id\_departamento  
) )  
/\*Restricciones de Integridad: Domain, check y assertions  
D. Todas las entregas, tienen que ser de películas de un mismo idioma.  
 \*/  
--create assertion entregas\_idioma  
create assertion add constraint entregas\_idioma  
check(not exists(  
 select 1  
 from entrega e join renglon\_entrega r on e.nro\_entrega = r.nro\_entrega  
 join pelicula p on r.codigo\_pelicula = p.codigo\_pelicula  
 group by e.nro\_entrega  
 having count(distinct p.idioma) > 1));  
  
--E. No pueden haber más de 10 empresas productoras por ciudad.  
create assertion empresa\_productora add constraint entrega\_ciudad  
check ( not exists( select 1  
from empresa\_productora ep join ciudad c on ep.id\_ciudad = c.id\_ciudad  
group by c.id\_ciudad,c.nombre\_ciudad  
having count(ep.nombre\_productora) > 10) )  
  
--F. Para cada película, si el formato es 8mm, el idioma tiene que ser francés.  
alter table pelicula  
add constraint pelicula\_formato\_idioma  
 check ( not *exists*(  
 select 1  
from pelicula  
where formato != 'formato 8' or idioma = 'Francés'  
group by titulo  
 ) )  
  
--G. El teléfono de los distribuidores Nacionales debe tener la misma característica que la de su  
--distribuidor mayorista.  
select telefono  
from distribuidor  
where tipo = 'N'

**TRIGGERS/TABLA AUDITORIA**

-- TP 5.2 Ej3,4,5  
/\*A. Controlar que las nacionalidades sean argetina, español, Inglés, Alemán O Chilena;. \*/  
alter table p5p1e1\_articulo  
add constraint ck\_nacionalidad\_especifica  
check ( nacionalidad IN('Argentina','Espaniol','Ingles','Aleman','Chilena'));  
  
--B. Para las fechas de publicaciones se debe considerar que sean fechas  
-- posteriores o iguales al 2010.  
  
alter table p5p1e1\_articulo  
add constraint ck\_publicaciones\_posteriores  
check ( *extract*(year from fecha\_publicacion) >= 2010 );  
  
--C. Cada palabra clave puede aparecer como máximo en 5 artículos.  
alter table p5p1e1\_contiene  
add constraint ck\_palabra\_clave\_art  
check ( not *exists*(  
 select 1  
 from p5p1e1\_contiene  
 group by idioma,cod\_palabra  
 having *count*(\*) >5  
) );  
CREATE TRIGGER tr\_max\_pal\_art  
BEFORE INSERT OR UPDATE OF idioma, cod\_palabra ON p5p1e1\_contiene  
 FOR EACH ROW EXECUTE PROCEDURE *fn\_maxArticulosxPalabra*();  
  
create or replace function *fn\_maxArticulosxPalabra*()  
RETURNS Trigger AS  
 $$ declare cant integer;  
 begin  
 SELECT *count*(id\_articulo) INTO cant  
 FROM p5p1e1\_contiene  
 WHERE idioma=NEW.idioma AND cod\_palabra=NEW.cod\_palabra;  
 IF (cant > 4) THEN  
 RAISE EXCEPTION 'Esta palabra esta contenida en mas de 5 articulos: ';  
 END IF;  
 RETURN NEW;  
 end  
 $$  
language 'plpgsql';  
  
/\*D. Sólo los autores argentinos pueden publicar artículos que contengan  
más de 10 palabras claves, pero con un tope de 15 palabras, el resto  
de los autores sólo pueden publicar artículos que contengan hasta  
10 palabras claves.\*/  
--create assetion  
alter table p5p1e1\_contiene  
add constraint ck\_clave\_arg\_autores  
check ( not *exists*(  
 select  
 from p5p1e1\_articulo join p5p1e1\_contiene  
 on p5p1e1\_articulo.id\_articulo = p5p1e1\_contiene.id\_articulo  
 where nacionalidad = 'Argentina'  
 group by p5p1e1\_contiene.id\_articulo  
 having *count*(\*) > 15  
) )  
  
create or replace function *fn\_articulos\_argentinos*()  
returns Trigger as  
 $$ declare cantidad integer;  
 declare autores\_arg p5p1e1\_articulo.nacionalidad%type;  
 begin  
 select nacionalidad as autores\_arg  
 from p5p1e1\_articulo  
 where id\_articulo=new.id\_articulo;  
 select *count*(\*) into cantidad  
 from p5p1e1\_contiene  
 where id\_articulo=new.id\_articulo; --and cantidad < 9??  
 if (autores\_arg = 'Argentina' and cantidad > 15) or (autores\_arg != 'Argentina' and cantidad>10)  
 then  
 raise exception 'tiene mas de 15';  
 end if;  
 return new;  
 end  
 $$  
 language 'plpgsql';  
  
CREATE TRIGGER th\_art\_arg  
BEFORE INSERT OR UPDATE OF id\_articulo, cod\_palabra ON p5p1e1\_contiene  
 FOR EACH ROW EXECUTE PROCEDURE *fn\_articulos\_argentinos*();  
  
--4\_A. La modalidad de la imagen médica puede tomar los siguientes valores RADIOLOGIA  
--CONVENCIONAL, FLUOROSCOPIA, ESTUDIOS RADIOGRAFICOS CON  
--FLUOROSCOPIA, MAMOGRAFIA, SONOGRAFIA,  
alter table p5p2e4\_imagen\_medica  
add constraint ck\_imagen\_med\_modalidad  
check ( modalidad IN ('CONVENCIONAL', 'FLUROSCOPIA', 'ESTUDIOS RADIOGRAFICOS CON FLUROSCOPIA',  
 'MAMOGRAFIA', 'SONOGRAFIA') )  
  
--B. Cada imagen no debe tener más de 5 procesamientos.  
alter table p5p2e4\_procesamiento  
add constraint ck\_imagen\_procesamiento\_5  
check ( not *exists*(  
 select 1  
 from p5p2e4\_procesamiento  
 group by id\_paciente,id\_imagen  
 having *count*(\*) > 5  
) )  
  
create or replace function *fn\_imagen\_procesamiento*() returns trigger as  
 $$declare cant int;  
 begin  
 select *count*(\*) into cant  
 from p5p2e4\_procesamiento  
 where id\_imagen=new.id\_imagen and id\_paciente=new.id\_paciente;  
 if(cant>4) then  
 raise exception 'No puede haber mas de 5 proc para cada img';  
 end if;  
 return new;  
 end;  
 $$ language plpgsql;  
  
create or replace trigger tg\_img\_proc  
 before insert or update of id\_paciente,id\_imagen on p5p2e4\_procesamiento  
 for each row execute procedure *fn\_imagen\_procesamiento*();  
  
  
--C. Agregue dos atributos de tipo fecha a las tablas Imagen\_medica y Procesamiento, una  
--indica la fecha de la imagen y la otra la fecha de procesamiento de la imagen y controle  
--que la segunda no sea menor que la primera.  
alter table p5p2e4\_imagen\_medica add fecha\_imagen date;  
alter table p5p2e4\_procesamiento add fecha\_procesamiento date;  
--create assertion  
alter table p5p2e4\_procesamiento  
add constraint ck\_fecha\_imagen\_proc  
check ( not *exists*(  
 select 1  
 from p5p2e4\_procesamiento p join p5p2e4\_imagen\_medica i on  
 p.id\_paciente = i.id\_paciente and p.id\_imagen = i.id\_imagen  
 where p.fecha\_procesamiento < i.fecha\_imagen  
) )  
--NECESITO DOS TRIGGERS UNO PARA CADA TABLA, EN PROC POR INSERTAR O MOD  
--Y EN IMAGEN POR MOD LA FECHA  
create or replace function *fn\_fecha\_proc\_fecha\_img*() returns trigger as  
 $$ declare fecha\_img p5p2e4\_imagen\_medica.fecha\_imagen%type;  
 declare fecha\_proc date;  
 begin  
 SELECT fecha\_imagen INTO fecha\_img  
 FROM p5p2e4\_imagen\_medica i  
 WHERE new.id\_paciente = i.id\_paciente AND new.id\_imagen = i.id\_imagen;  
 fecha\_proc := new.fecha\_procesamiento;  
 if(fecha\_proc > fecha\_img) then  
 raise exception 'la fecha de img es mayor a fecha del proc';  
  
 end if;  
 return new;  
 end;  
 $$ language plpgsql;  
  
create trigger tg\_fecha\_img\_fecha\_proc  
 before insert or update of id\_imagen,id\_paciente,fecha\_procesamiento  
 on p5p2e4\_procesamiento for each row execute procedure *fn\_fecha\_proc\_fecha\_img*();  
  
create or replace function *fn\_fecha\_img*() returns trigger as  
 $$declare fecha\_proc date;  
 begin  
 select *max*(fecha\_procesamiento) into fecha\_proc  
 from p5p2e4\_procesamiento  
 where id\_paciente=new.id\_paciente and id\_imagen=new.id\_imagen;  
 if (new.fecha\_imagen>fecha\_proc) then  
 raise exception 'la img es mayor';  
 end if;  
 return new;  
 end;  
 $$ language plpgsql;  
  
create trigger tg\_fecha\_img  
 before update of fecha\_imagen  
 on p5p2e4\_imagen\_medica for each row execute procedure *fn\_fecha\_img*();  
  
--D. Cada paciente sólo puede realizar dos FLUOROSCOPIA anuales.  
--tabla imagen  
alter table p5p2e4\_imagen\_medica  
add constraint ck\_imagen\_anual  
check ( not *exists*(  
 select 1  
 from p5p2e4\_imagen\_medica  
 where modalidad = 'Fluroscopia'  
 group by id\_paciente,*extract*(year from fecha\_imagen)  
 having *count*(\*) > 2  
) );  
create or replace function *fn\_img\_fluroscopia*() returns trigger as  
 $$declare cantidad int;  
 begin  
 select *count*(\*) into cantidad  
 from p5p2e4\_imagen\_medica  
 where id\_paciente=new.id\_paciente and modalidad = 'Fluroscopia'  
 and *extract*(year from (fecha\_imagen))=*extract*(year from (new.fecha\_imagen));  
 if (cantidad>1) then--podria tambien cantidad<=2  
 raise exception 'ya tiene dos fluroscopias';  
 end if;  
 return new;  
 end  
 $$ language plpgsql;  
  
create trigger tg\_imagen\_fluroscopia\_anual  
 before insert or update of fecha\_imagen,modalidad on p5p2e4\_imagen\_medica  
for each row execute procedure *fn\_img\_fluroscopia*();  
  
--E. No se pueden aplicar algoritmos de costo computacional “O(n)” a imágenes de  
--FLUOROSCOPIA  
create assertion  
check( not exists(  
 select 1  
 from p5p2e4\_imagen\_medica i join p5p2e4\_procesamiento p  
 on i.id\_paciente = p.id\_paciente and i.id\_imagen = p.id\_imagen  
 join p5p2e4\_algoritmo a on p.id\_algoritmo = a.id\_algoritmo  
 where i.modalidad = 'Fluroscopia' and a.costo\_computacional = '0(n)'  
));  
--EJ2-E--|TABLAS | UPDATE | INSERT | DELETE  
-- | imagen\_medica | SI id\_paciente,modalidad | NO | NO  
-- | algoritmo | SI costo\_computacional,id\_algoritmo | NO | NO  
-- | procesamiento | SI id\_algorit/id\_paciente/id\_imagen | SI id\_algor/id\_pacient/id\_img| NO  
--E. No se pueden aplicar algoritmos de costo computacional “O(n)” a imágenes de FLUOROSCOPIA  
  
create or replace trigger tg\_proc\_img\_algoritmos  
 before insert or update of id\_paciente,id\_imagen,id\_algoritmo on p5p2e4\_procesamiento  
 for each row execute function *fn\_proc\_img\_algoritmos*();  
  
create or replace function *fn\_proc\_img\_algoritmos*() returns trigger as  
 $$declare modalidad p5p2e4\_imagen\_medica.modalidad%type;  
 declare costo\_com p5p2e4\_algoritmo.id\_algoritmo%type;  
 begin  
 select modalidad into modalidad  
 from p5p2e4\_imagen\_medica i join p5p2e4\_procesamiento on  
 i.id\_imagen=new.id\_imagen and i.id\_paciente=new.id\_paciente;  
 if(modalidad = 'Fluroscopia') then  
 select costo\_computacional into costo\_com  
 from p5p2e4\_algoritmo a join p5p2e4\_procesamiento on  
 a.id\_algoritmo=new.id\_algoritmo;  
 if(costo\_com = '0(n)') then  
 raise exception 'La mod fluroscopia no puede tener costo 0(n)';  
 end if;  
 end if;  
 return new;  
 end;  
 $$ language plpgsql;  
  
create or replace trigger tg\_img\_modalidad  
 before update of id\_paciente,id\_imagen,modalidad on p5p2e4\_imagen\_medica  
 for each row execute function *fc\_img\_modalidad*();  
  
create or replace function *fc\_img\_modalidad*() returns trigger as  
 $$--si mod en imagen med primero controlo que sea fluros y luego que pertenezca a ese paciente  
 declare costo p5p2e4\_algoritmo.costo\_computacional%type;  
 begin  
 if(new.modalidad = 'Fluroscopia') then  
 select a.costo\_computacional into costo  
 from p5p2e4\_algoritmo a join p5p2e4\_procesamiento p on a.id\_algoritmo = p.id\_algoritmo  
 join p5p2e4\_imagen\_medica i on p.id\_paciente = new.id\_paciente and p.id\_imagen = new.id\_imagen  
 if(costo = '0(n)') then  
 raise exception 'la mod fluroscopia no puede ser 0(n)';  
 end if;  
 end if;  
 return new;  
 end;  
 $$ language plpgsql;  
  
create trigger tr\_modalidad\_acept\_algoritmo  
 before update of id\_algoritmo, costo\_computacional on p5p2e4\_algoritmo  
 for each row execute function *fc\_modalidad\_acept\_algoritmo*();  
 create or replace function *fc\_modalidad\_acept\_algoritmo*() returns trigger as  
 $$  
 declare modalidad varchar(80);  
 begin  
 if(NEW.costo\_computacional='O(n)')then  
 select i.modalidad into modalidad  
 from p5p2e4\_imagen\_medica i  
 join p5p2e4\_procesamiento p on (i.id\_paciente=p.id\_paciente and i.id\_imagen=p.id\_imagen)  
 join p5p2e4\_algoritmo a on(p.id\_algoritmo=NEW.id\_algoritmo);  
 if(modalidad = 'FLUOROSCOPIA')then  
 raise exception 'costo computacional invalido para el tipo de modalidad';  
 end if;  
 end if;  
 return NEW;  
 end  
 $$language plpgsql;  
  
--EJ 5  
--A. Los descuentos en las ventas son porcentajes y deben estar entre 0 y 100.  
alter table p5p2e5\_venta  
add constraint ck\_venta\_descuento  
check ( descuento between 0 and 100)  
  
--B. Los descuentos realizados en fechas de liquidación deben superar el 30%.  
alter table p5p2e5\_venta  
add mes\_liq int not null,  
add dia\_liq int not null;  
  
alter table p5p2e5\_venta  
add constraint fk\_fecha\_liq\_venta  
foreign key (mes\_liq,dia\_liq)  
references p5p2e5\_fecha\_liq (mes\_liq,dia\_liq)  
NOT DEFERRABLE  
INITIALLY IMMEDIATE;  
  
alter table p5p2e5\_venta  
add constraint ck\_fecha\_liq\_30  
check ( not *exists*(  
 select 1  
 from p5p2e5\_venta  
 where descuento < 30 and *extract*(day from fecha) = dia\_liq and *extract*(month from fecha) = mes\_liq  
) );  
  
create or replace function *fc\_venta\_liq*() returns trigger as  
 $$  
 begin  
 SELECT  
 FROM p5p2e5\_venta  
 where *extract*(month from fecha) = new.mes\_liq  
 and *extract*(day from fecha) = new.dia\_liq;  
 if(new.descuento <= 30) then  
 raise exception 'El desc no supera el 30%';  
 end if;  
 return new;  
 end;  
 $$ language plpgsql;  
create or replace trigger tr\_fecha\_liq  
 before insert or update of id\_venta,descuento  
 on p5p2e5\_venta for each row execute function *fc\_venta\_liq*();  
  
--C. Las liquidaciones de Julio y Diciembre no deben superar los 5 días.  
alter table p5p2e5\_fecha\_liq  
add constraint ck\_liq\_julio\_diciembre  
check (not *exists*(  
 select 1  
 from p5p2e5\_fecha\_liq  
 where mes\_liq = '6' or mes\_liq = '12'  
 having *count*(dia\_liq) > 5  
));  
create or replace trigger tg\_liq\_julio\_dic  
 before insert or update of mes\_liq,cant\_dias on p5p2e5\_fecha\_liq  
 for each row execute function *fn\_liq\_julio\_diciembre*();  
  
create or replace function *fn\_liq\_julio\_diciembre*() returns trigger as  
 $$  
 begin  
 if(new.mes\_liq = 6 or new.mes\_liq = 12) then  
 if(new.cant\_dias > 5) then  
 raise exception 'Entre julio y diciembre la cantidad de dias no puede se mayor a 5';  
 end if;  
 end if;  
 return new;  
 end;  
 $$ language plpgsql;  
  
--D. Las prendas de categoría ‘oferta’ no tienen descuentos.  
--es un assertion  
alter table p5p2e5\_prenda  
add constraint ck\_prenda\_categ\_desc  
check ( not *exists*(  
 select 1  
 from p5p2e5\_prenda p join p5p2e5\_venta v on p.id\_prenda = v.id\_prenda  
 where categoria = 'oferta' and descuento in not null  
) )  
create or replace function *fn\_prenda\_oferta\_descuento*() returns trigger as  
 $$  
 declare descuento decimal(10,2);  
 begin  
 if(new.categoria = 'oferta') then  
 select descuento into descuento  
 from p5p2e5\_venta join p5p2e5\_prenda p on  
 new.id\_prenda=p.id\_prenda;  
 if(descuento !== null)then  
 raise exception 'Las prendas en oferta no tienen desc';  
 end if;  
 end if;  
 return new;  
 end;  
 $$ language plpgsql;  
  
create or replace trigger tg\_prenda\_oferta\_desc  
 before insert or update of id\_prenda,categoria on p5p2e5\_prenda  
 for each row execute function *fn\_prenda\_oferta\_descuento*();  
  
--PRACTICO 6 EJERCICIO 4 (ESTADISTICA)  
CREATE TABLE P6\_E4\_Pelicula AS  
SELECT \* FROM unc\_esq\_peliculas.pelicula;  
  
CREATE TABLE p6\_e4\_estadistica AS  
SELECT genero, *COUNT*(\*) total\_peliculas, *count* (distinct idioma) cantidad\_idiomas  
FROM p6\_e4\_Pelicula GROUP BY genero;  
  
/\*c) Cree un trigger que cada vez que se realice una modificación en la tabla película (la creada  
en su esquema) tiene que actualizar la tabla estadística.No se olvide de identificar:  
i) la granularidad del trigger.  
ii) Eventos ante los cuales se debe disparar.  
iii) Analice si conviene modificar por cada operación de actualización o reconstruirla de  
cero.  
\*/  
CREATE TABLE p6\_e4\_estadistica (  
 genero VARCHAR(255) PRIMARY KEY,  
 total\_peliculas INT,  
 cantidad\_idiomas INT  
);  
  
CREATE OR REPLACE FUNCTION *fn\_auditoria\_peliculas\_estadistica*() RETURNS TRIGGER AS  
$$  
BEGIN  
 -- Inserta los nuevos datos agrupados  
 INSERT INTO p6\_e4\_estadistica (genero, total\_peliculas, cantidad\_idiomas)  
 SELECT genero, *COUNT*(\*) total\_peliculas, *COUNT*(DISTINCT idioma) cantidad\_idiomas  
 FROM p6\_e4\_pelicula  
 WHERE genero = NEW.genero  
 GROUP BY genero;  
  
 RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
--GENERO NO PUEDE SER PRIMARI KEY  
  
create or replace trigger tg\_auditoria\_peliculas  
 after update of genero,idioma on p6\_e4\_pelicula  
 for each row execute function *fn\_auditoria\_peliculas\_estadistica*();  
  
select \* from p6\_e4\_estadistica;