Resumen para final Base de Datos 2022

Esquemas:

- **Películas**
- **Voluntarios**

Trabajo Práctico N4 parte 1:

1. Seleccione el identificador y nombre de todas las instituciones que son Fundaciones. (V)

```
SELECT id_institucion, nombre_institucion
FROM unc_esq_voluntario.institucion;
```

2. Seleccione el identificador de distribuidor, identificador de departamento y nombre de todos los departamentos. (P)

```
SELECT id_distribuidor, id_departamento, nombre_departamento
FROM unc_esq_pelicula;
```

3. Muestre el nombre, apellido y el teléfono de todos los empleados cuyo id_tarea sea 7231, ordenados por apellido y nombre. (P)

```
SELECT nombre, apellido, telefono
FROM unc_esq_pelicula.empleado
WHERE id_tarea LIKE 7231
ORDER BY apellido, nombre;
```

4. Muestre el apellido e identificador de todos los empleados que no cobran porcentaje de comisión. (P)

```
SELECT apellido, id_empleado
FROM unc_esq_pelicula.empleado
WHERE porc_comision IS NULL;
```

5. Muestre el apellido y el identificador de la tarea de todos los voluntarios que no tienen coordinador. (V)

```
SELECT apellido, id_tarea
FROM unc_esq_voluntario.voluntario
WHERE id_coordinador IS NULL;
```

6. Muestre los datos de los distribuidores internacionales que no tienen registrado teléfono. (P)

```
SELECT *
```

```
FROM unc_esq_pelicula.distribuidor
WHERE tipo LIKE 'I'
AND telefono IS NULL;
```

7. Muestre los apellidos, nombres y mails de los empleados con cuentas de gmail y cuyo sueldo sea superior a $ 1000. (P)

```
SELECT apellido, nombre, e_mail
FROM unc_esq_pelicula.empleado
WHERE e_mail LIKE '%@gmail.com'
AND sueldo > 1000;
```

8.- Seleccione los diferentes identificadores de tareas que se utilizan en la tabla empleado. (P)

```
SELECT DISTINCT id_tarea
FROM unc_esq_empleado.empleado;
```

9. Muestre el apellido, nombre y mail de todos los voluntarios cuyo teléfono comienza con +51. Coloque el encabezado de las columnas de los títulos Apellido y Nombre y Dirección de mail. (V)

```
SELECT apellido || ',' || nombre AS 'Apellido y Nombre', e_mail AS 'Dirección de mail'
FROM unc_esq_voluntario.voluntario
WHERE telefono LIKE '+51%'
```

10. Hacer un listado de los cumpleaños de todos los empleados donde se muestre el nombre y el apellido (concatenados y separados por una coma) y su fecha de cumpleaños (solo el día y el mes), ordenado de acuerdo al mes y día de cumpleaños en forma ascendente. (P)

```
SELECT nombre || ',' || apellido AS 'Nombre y apellido', EXTRACT(DAY FROM
fecha_nacimiento) AS dia, EXTRACT(MONTH FROM fecha_nacimiento) AS mes
FROM unc_esq_pelicula.empleado
ORDER BY mes, dia;
```

11. Recupere la cantidad mínima, máxima y promedio de horas aportadas por los voluntarios nacidos desde 1990. (V)

```
SELECT MIN(horas_aportadas), MAX(horas_aportadas), AVG(horas_aportadas)
FROM unc_esq_voluntario.voluntario
WHERE EXTRACT(YEAR FROM fecha_nacimiento) > 1989;
```

12. Listar la cantidad de películas que hay por cada idioma. (P)

```
SELECT COUNT(codigo_pelicula)
FROM unc_esq_pelicula.pelicula
GROUP BY idioma;
```

13. Calcular la cantidad de empleados por departamento. (P)

```
SELECT COUNT(id_empleado)
FROM unc_esq_pelicula.empleado
GROUP BY id_departamento;
```

14. Mostrar los códigos de películas que han recibido entre 3 y 5 entregas. (veces entregadas, NO cantidad de películas entregadas). (P)

PROBAR

```
SELECT codigo_pelicula
FROM unc_esq_pelicula.renglon_entrega
GROUP BY codigo_pelicula
HAVING COUNT(nro_entrega) BETWEEN 3 AND 5;
```

15. ¿Cuántos cumpleaños de voluntarios hay cada mes? (V)

```
SELECT COUNT(nro_voluntario)
FROM unc_esq_voluntario.voluntario
GROUP BY EXTRACT(MONTH FROM fecha_nacimiento);
```

16. ¿Cuáles son las 2 instituciones que más voluntarios tienen? (V)

```
SELECT id_institucion, COUNT(nro_voluntario) AS cantidad
FROM unc_esq_voluntario.voluntario
GROUP BY id_institucion
ORDER BY cantidad DESC
LIMIT 2;
```

17. ¿Cuáles son los id de ciudades que tienen más de un departamento? (P)

```
SELECT id_ciudad
FROM unc_esq_pelicula.departamento
GROUP BY id_ciudad HAVING COUNT(id_departamento) > 1;
```

Trabajo Práctico N4 parte 2:

Consultas con anidamiento (usando IN, NOT IN, EXISTS, NOT EXISTS):

1. Listar todas las películas que poseen entregas de películas de idioma inglés durante el año 2006. (P)

```
SELECT *
FROM unc_esq_pelicula.pelicula p
WHERE idioma LIKE 2006
```

```
AND EXISTS (SELECT 1 #2
            FROM unc_esq_pelicula.renglon_entrega re
            WHERE p.codigo_pelicula = re.codigo_pelicula
            AND EXISTS (SELECT 1 #1
                        FROM unc_esq_pelicula.entrega e
                        WHERE re.nro_entrega = e.nro_entrega
                        AND fecha_entrega LIKE 2006));
```

#1 Se fija si existe un registro donde coincidan el renglon_entrega y entrega y donde la fecha sea 2006.
#2 Se fija si existe un registro donde coincidan la película y el renglon_entrega

2. Indicar la cantidad de películas que han sido entregadas en 2006 por un distribuidor nacional. Trate de resolverlo utilizando ensambles. (P)

```
SELECT COUNT(p.codigo_pelicula)
FROM unc_esq_pelicula.pelicula p
JOIN unc_esq_pelicula.renglon_entrega re ON (p.codigo_pelicula = re.codigo_pelicula)
JOIN unc_esq_pelicula.entrega e ON (re.nro_entrega = e.nro_entrega)
JOIN unc_esq_pelicula.distribuidor d ON (e.id_distribuidor = d.id_distribuidor)
WHERE e.fecha_entrega LIKE 2006
AND d.tipo LIKE 'N';
```

3. Indicar los departamentos que no posean empleados cuya diferencia de sueldo máximo y mínimo (asociado a la tarea que realiza) no supere el 40% del sueldo máximo. (P) (Probar con 10% para que retorne valores)

```
SELECT d.id_departamento
FROM unc_esq_pelicula.departamento d
WHERE NOT EXISTS (SELECT 1
                  FROM unc_esq_pelicula.tarea t
                  WHERE d.id_tarea = t.id_tarea
                  AND (t.sueldo_maximo - t.sueldo_minimo) < (t.sueldo_maximo * 0.4);
```

4.- Liste las películas que nunca han sido entregadas por un distribuidor nacional. (P)

```
SELECT p.*
FROM unc_esq_pelicula.pelicula p
WHERE p.codigo_pelicula IN (SELECT re.codigo_pelicula #3
    FROM unc_esq_pelicula.renglon_entrega re
    WHERE re.nro_entrega IN (SELECT e.nro_entrega #2
        FROM unc_esq_pelicula.entrega e
        WHERE re.id_distribuidor IN (SELECT d.id_distribuidor #1
            FROM unc_esq_pelicula.distribuidor d
            WHERE d.tipo NOT IN ('N'))));
```

#1 Selecciona los distribuidores que no sean nacionales.
#2 Selecciona las entregas que no pertenezcan a distribuidores nacionales.

#3 Selecciona las películas donde las entregas no pertenezcan a distribuidores nacionales.

5. Determinar los jefes que poseen personal a cargo y cuyos departamentos (los del jefe) se encuentren en la Argentina. (P)

```
SELECT e.*
FROM unc_esq_pelicula.empleado e
WHERE e.id_empleado IN (SELECT e2.id_jefe #3
        FROM unc_esq_pelicula.empleado e2
        WHERE e2.id_jefe IS NOT NULL
        AND e2.id_jefe, e2.id_distribuidor IN (SELECT d.jefe_departamento, d.distribuidor#2
                FROM unc_esq_pelicula.departamento d
                WHERE d.id_ciudad IN (SELECT c.id_ciudad #1
                                FROM unc_esq_pelicula.ciudad c
                                WHERE c.id_pais LIKE 'AR'))));
```

#1 Selecciona las ciudades de Argentina.
#2 Selecciona los jefes de departamento que residen en una ciudad de Argentina.
#3 Selecciona a los jefes de empleados.

6. Liste el apellido y nombre de los empleados que pertenecen a aquellos departamentos de Argentina y donde el jefe de departamento posee una comisión de más del 10% de la que posee su empleado a cargo. (P)

```
SELECT e.apellido, e.nombre
FROM unc_esq_pelicula.empleado e
WHERE e.id_departamento, e.id_distribuidor IN (SELECT d.id_departamento,
d.id_distribuidor #3
        FROM unc_esq_pelicula.departamento d
        WHERE d.id_ciudad IN (SELECT c.id_ciudad #2
                FROM unc_esq_pelicula.ciudad c
                WHERE c.id_pais LIKE 'AR')
        AND d.jefe_departamento IN (SELECT ej.id_empleado #1
                FROM unc_esq_pelicula.empleado ej
                WHERE ej.porc_comision > 10));
```

#1 Selecciona los empleados que tiene una comisión mayor al 10%
#2 Selecciona las ciudades que pertenecen a Argentina.
#3 Selecciona los departamentos que pertenecen a una ciudad de Argentina y donde el jefe del departamento cobra una comisión mayor al 10%

7. Indicar la cantidad de películas entregadas a partir del 2010, por género. (P)

```
SELECT p.genero, COUNT(p.codigo_pelicula) * re.cantidad
FROM unc_esq_pelicula.pelicula p
WHERE p.codigo_pelicula IN (SELECT re.codigo_pelicula #2
        FROM unc_esq_pelicula.renglon_entrega re
        WHERE re.nro_entrega IN (SELECT e.nro_entrega #1
```

```
        FROM unc_esq_pelicula.entrega e
        WHERE EXTRACT(YEAR FROM e.fecha_entrega) > 2009))
GROUP BY p.genero;
);
```

#1 Selecciona las entregas a partir del año 2010.
#2 Selecciona las películas con entregas a partir del año 2010.

8. Realizar un resumen de entregas por día, indicando el video club al cual se le realizó la entrega y la cantidad entregada. Ordenar el resultado por fecha. (P)

```
SELECT e.id_video, e.fecha_entrega, COUNT(re.codigo_pelicula) * re.cantidad
FROM unc_esq_pelicula.entrega e, unc_esq_pelicula.renglon_entrega re
WHERE e.nro_entrega = re.nro_entrega
GROUP BY e.fecha_entrega
ORDER BY e.fecha_entrega;
```

9. Listar, para cada ciudad, el nombre de la ciudad y la cantidad de empleados mayores de edad que desempeñan tareas en departamentos de la misma y que posean al menos 30 empleados. (P)

```
SELECT c.nombre_ciudad, COUNT(e.id_empleado)
FROM unc_esq_pelicula.ciudad c, unc_esq_pelicula.departamento d,
unc_esq_pelicula.empleado e
WHERE c.id_ciudad = d.id_ciudad
AND d.id_departamento = e.id_departamento
AND d.id_distribuidor = e.id_distribuidor
AND AGE(e.fecha_nacimiento) > 18
GROUP BY c.nombre_ciudad HAVING COUNT(e.id_empleado) > 29;
```

10. Muestre, para cada institución, su nombre y la cantidad de voluntarios que realizan aportes. Ordene el resultado por nombre de institución. (V)

```
SELECT i.nombre_institucion, COUNT(v.nro_voluntario)
FROM unc_esq_voluntario.institucion i JOIN unc_esq_voluntario.voluntario v ON
(i.id_institucion = v.id_institucion)
GROUP BY i.nombre_institucion
ORDER BY i.nombre_institucion;
```

11. Determine la cantidad de coordinadores en cada país, agrupados por nombre de país y nombre de continente. Etiquete la primer columna como Número de coordinadores. (V)

```
SELECT p.nombre_pais, c.nombre_continente, COUNT(v.id_coordinador) AS ‘Número de
coordinadores’
FROM unc_esq_voluntario.continente c JOIN unc_esq_voluntario.pais p ON
(c.id_continente = p.id_continente)
JOIN unc_esq_voluntario.direccion d ON (p.id_pais = d.id_pais)
```

```
JOIN unc_esq_voluntario.institucion i ON (i.id_direccion = d.id_direccion)
JOIN unc_esq_voluntario.voluntario v ON (v.id_institucion = i.id_institucion)
WHERE v.id_coordinador IS NOT NULL
GROUP BY p.nombre_pais, c.nombre_continente;
```

12. Escriba una consulta para mostrar el apellido, nombre y fecha de nacimiento de cualquier voluntario que trabaje en la misma institución que el Sr. de apellido Zlotkey. Excluya del resultado a Zlotkey.

```
SELECT v.apellido, v.nombre, v.fecha_nacimiento
FROM unc_esq_voluntario.voluntario v
WHERE v.id_institucion IN (SELECT v2.id_institucion #1
        FROM unc_esq_voluntario.v2
        WHERE v2.apellido LIKE 'Zlotkey'
AND v.apellido NOT LIKE 'Zlotkey';
```

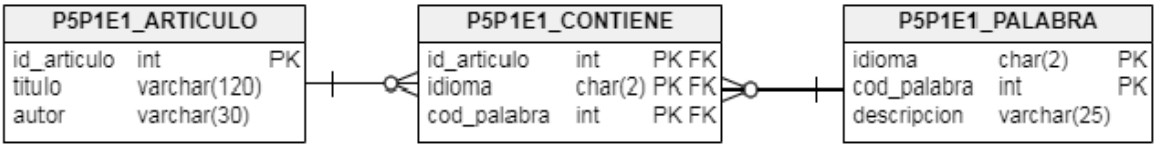
#1 Selecciona la institución donde se encuentra el Sr. Zlotkey.

13. Cree una consulta para mostrar los números de voluntarios y los apellidos de todos los voluntarios cuya cantidad de horas aportadas sea mayor que la media de las horas aportadas. Ordene los resultados por horas aportadas en orden ascendente.

```
SELECT v.nro_voluntario, v.apellido
FROM unc_esq_voluntario.voluntario v
WHERE v.horas_aportadas > AVG(v.horas_aportadas)
ORDER BY v.horas_aportadas;
```

Trabajo Práctico N5 parte 1:

1. Cómo debería implementar las Restricciones de Integridad Referencial (RIR) si se desea que cada vez que se elimine un registro de la tabla PALABRA , también se eliminen los artículos que la referencian en la tabla CONTIENE.




```
ALTER TABLE P5P1E1_CONTIENE
ADD CONSTRAINT FK_cod_palabra FOREING KEY (cod_palabra)
REFERENCES P5P1E1_PALABRA (cod_palabra)
ON DELETE CASCADE;
ALTER TABLE P5P1E1_CONTIENE
```

ADD CONSTRAINT FK_idioma FOREING KEY (idioma)
REFERENCES P5P1E1_PALABRA (idioma)
ON DELETE CASCADE;

2. Verifique qué sucede con las palabras contenidas en cada artículo, al eliminar una palabra, si definen la Acción Referencial para las bajas (ON DELETE) de la RIR correspondiente como:

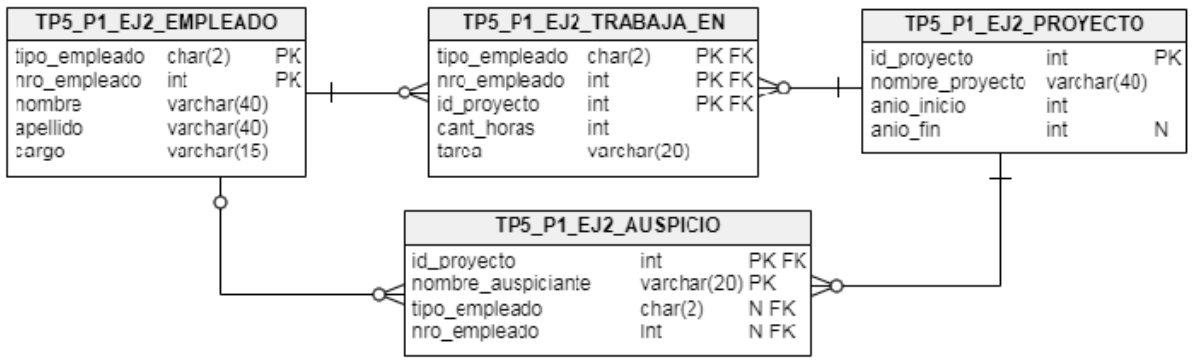
A) Restrict

B) Es posible para éste ejemplo colocar SET NULL o SET DEFAULT para ON DELETE y ON UPDATE?

2A) Si queremos eliminar una palabra, usaremos la tabla P5P1E1_PALABRA y si la acción referencial para el DELETE es RESTRICT, que es por defecto, podrá borrar la palabra si no está siendo referenciada en la tabla P5P1E1_CONTIENE, en el caso contrario, no podrá borrarla.

2B) No es posible para ninguno de los dos casos poner SET NULL porque en la tabla P5P1E1_CONTIENE, la columna cod_palabra e idioma no tienen la opción de NULL.
En el caso de que tengan un DEFAULT establecido, sí se podría y establecerán ese valor por defecto.

3. Considere el siguiente esquema de una base de datos de un Centro de Desarrollo, cuyo script de creación lo podes descargar de aquí



Teniendo en cuenta las cláusulas ON UPDATE y ON DELETE las FOREIGN KEY y siguientes los siguientes registros en las respectivas tablas cuyos inserts los podes descargar de aquí

AUSPICIO

id_proyecto	nombre_auspiciante	tipo_empleado	nro_empleado
2	McDonald	A	2

PROYECTO

id_proyecto	nombre_proyecto	anio_inicio	anio_fin
1	Proy 1	2019	*NULL*
2	Proy 2	2018	2019
3	Proy 3	2020	*NULL*

EMPLEADO

tipo_empleado	nro_empleado	nombre	apellido	cargo
A	1	Juan	Garcia	Jefe
B	1	Luis	Lopez	Adm
A	2	Maria	Casio	CIO

TRABAJA_EN

tipo_empleado	nro_empleado	id_proyecto	cant_horas	tarea
A	1	1	35	T1
A	2	2	25	T3

A. **Indique el resultado de las siguientes operaciones, teniendo en cuenta las acciones referenciales e instancias dadas. En caso de que la operación no se pueda realizar, indicar qué regla/s entra/n en conflicto y cuál es la causa. En caso de que sea aceptada, comente el resultado que produciría (NOTA: en cada caso considere el efecto sobre la instancia original de la BD, los resultados no son acumulativos).**

   A.1) delete from tp5_p1_ej2_proyecto where id_proyecto = 3;
   A.2) update tp5_p1_ej2_proyecto set id_proyecto = 7 where id_proyecto = 3;
   A.3) delete from tp5_p1_ej2_proyecto where id_proyecto = 1;
   A.4) delete from tp5_p1_ej2_empleado where tipo_empleado = 'A' and nro_empleado = 2;
   A.5) update tp5_p1_ej2_trabaja_en set id_proyecto = 3 where id_proyecto =1;
   A.6) update tp5_p1_ej2_proyecto set id_proyecto = 5 where id_proyecto = 2;

A.1) El registro se borra porque no está haciendo referencia a ningún otro registro, por lo que no hay ninguna restricción implicada.
A.2) El registro se actualiza con el id_proyecto = 7 ya que no está haciendo referencia a ningún otro registro, ese id_proyecto = 7 no existe y por último, no hay ninguna restricción implicada.
A.3) No se podrá borrar el registro ya que está haciendo referencia a otro registro de la tabla TRABAJA_EN que tiene el ON CASCADE en RESTRICT.
A.4) En la tabla AUSPICIO tiene el ON DELETE SET NULL por lo que tipo_empleado y nro_empleado se modificarán a NULL. En la tabla TRABAJA_EN está el ON DELETE CASCADE por lo que se borrará el registro de la tabla EMPLEADO y en TRABAJA_EN.
A.5) Se modificará el registro de id_proyecto = 1 a id_proyecto = 3 ya que ese registro no está siendo referenciado en ningún lado.
A.6) No se va a poder hacer esa actualización ya que en la tabla AUSPICIO el ON UPDATE es RESTRICT y esta acción referencial se ejecuta antes que todas las demás, por lo tanto aunque la tabla TRABAJA_EN tenga el ON UPDATE en CASCADE, no se modificará.

B. **Indique el resultado de la siguiente operaciones justificando su elección:**
   update auspicio set id_proyecto= 66, nro_empleado = 10
   where id_proyecto = 22
   and tipo_empleado = 'A';
   and nro_empleado = 5;
   (suponga que existe la tupla asociada)

   B.1) Realiza la modificación si existe el proyecto 22 y el empleado TipoE = 'A'; ,NroE = 5
   B.2) Realiza la modificación si existe el proyecto 22 sin importar si existe el empleado TipoE =
   'A'; ,NroE = 5
   B.3) Se modifican los valores, dando de alta el proyecto 66 en caso de que no exista (si no se violan restricciones de nulidad), sin importar si existe el empleado

B.4) Se modifican los valores, y se da de alta el proyecto 66 y el empleado correspondiente (si no se violan restricciones de nulidad)
B.5) No permite en ningún caso la actualización debido a la modalidad de la restricción entre la tabla empleado y auspicio.
B.6) Ninguna de las anteriores, cuál?

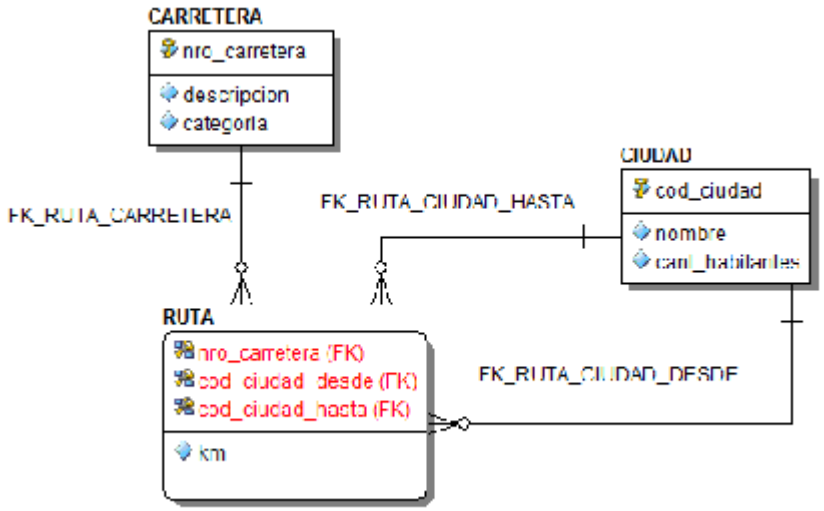
B.1 es la respuesta ya que suponiendo que todos los datos existen, id_proyecto y nro_empleado son FK que hacen referencia a datos de otra tabla.

C. **Indique cuáles de las siguientes operaciones serán aceptadas/rechazadas, según se considere para las relaciones AUSPICIO-EMPLEADO y AUSPICIO-PROYECTO match: i) simple, ii) parcial, o iii) full:**

C.1) insert into Auspicio values (1, Dell , B, null);
C.2) insert into Auspicio values (2, Oracle, null, null);
C.3) insert into Auspicio values (3, Google, A, 3);
C.4) insert into Auspicio values (1, HP, null, 3);

C.1) Será aceptada ya que las PK no se repiten y porque id_proyecto = 1 existe en la tabla PROYECTO. El tipo de match es PARCIAL.
C.2) Será aceptada porque por más que el registro que quiero insertar tenga id_proyecto = 2, el nombre_auspiciante que es la otra PK no existe en la tabla AUSPICIO. El tipo de match es FULL.
C.3) Será rechazada porque el nro_empleado = 3 no existe en la tabla EMPLEADO. El tipo de match es PARCIAL.
C.4) Será rechazada porque no existe un nro_empleado = 3. El tipo de match es SIMPLE.

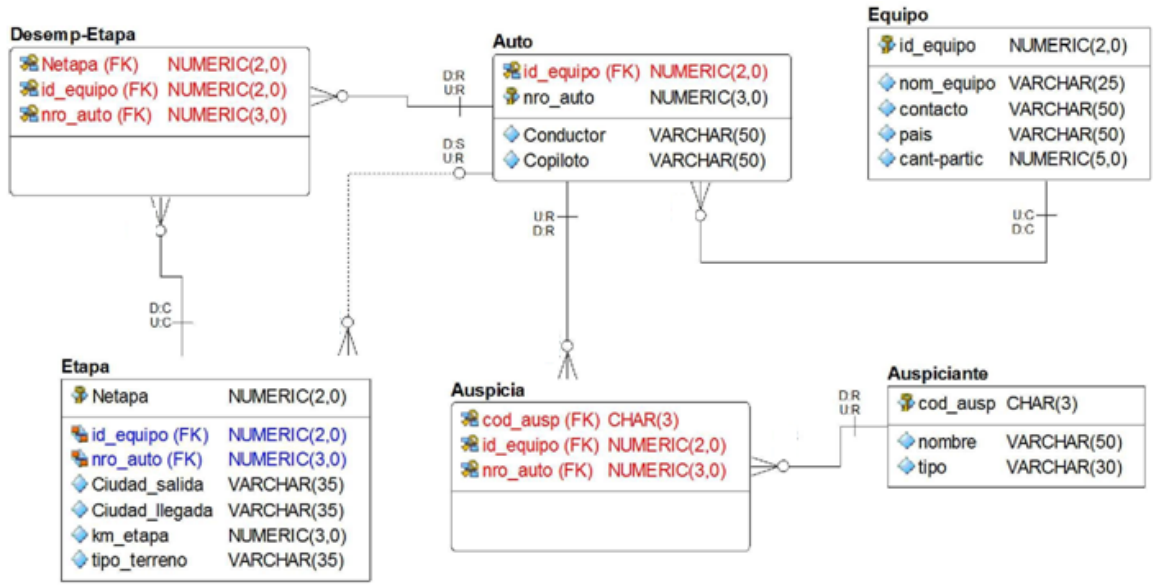
4. Sea el siguiente DERExt:



A. **Se podrá declarar como acción referencial de la (RIR) FK_Ruta_ciudad_desde DELETE CASCADE y para la RIR FK_Ruta_ciudad_hasta DELETE RESTRICT ?**
B. **Es posible colocar DELETE SET NULL o UPDATE SET NULL como acción referencial de la RIR FK_Ruta_Carretera ?**

4.A) No se va a poder porque ambas FK van a hacer referencia a un mismo registro y en el caso de DELETE CASCADE le estamos diciendo que borre todo lo que esté relacionado con ese registro y en su contraparte con el DELETE RESTRICT le decimos lo contrario y este último se chequea antes que las otras restricciones así que no va a borrar nada.
4.B) En el DELETE SET NULL va a borrar el registro en la tabla CARRETERA en el caso de que la FK acepte NULL. En el UPDATE SET NULL no se va a poder llevar a cabo porque nro_carretera en la tabla CARRETERA es PK y las PK de destacan por ser únicas y no NULL.

5. Es posible definir las siguientes RIRs tal como se declaran en cada punto? Indique V o F según corresponda y justifique.



5.A) ALTER TABLE Desemp-Etapa
ADD CONSTRAINT FK_DesempEtapa_Auto FOREIGN KEY (id_equipo)
REFERENCES Auto (id_equipo);

5.B) ALTER TABLE Equipo
ADD CONSTRAINT FK_Equipo_Auto FOREIGN KEY (id_equipo, contacto)
REFERENCES Auto (id_equipo, conductor);

5.C) ALTER TABLE Desemp-Etapa
ADD CONSTRAINT FK_DesempEtapa_Etapa FOREIGN KEY (Netapa, id_equipo)
REFERENCES Etapa (etapa, id_equipo);

5.D) ALTER TABLE Auspicia
ADD CONSTRAINT FK_Auspicia_Etapa FOREIGN KEY (nro_auto)
REFERENCES Etapa (nro_auto);

5.E) ALTER TABLE Auto

ADD CONSTRAINT FK_Auto_Equipo FOREIGN KEY (id_equipo)
REFERENCES Equipo (id_equipo);

5.F) ALTER TABLE Desemp-Etapa
ADD CONSTRAINT FK_DesempEtapa_Auto FOREIGN KEY (id_equipo, nro_auto)
REFERENCES Auto (id_equipo, nro_auto);

5.A) Falso, le falta relacionar nro_auto ya que es PK en la tabla Auto.
5.B) Falso, id_equipo es la PK en Equipo y en todo caso esa FK debería hacerse en la tabla Auto.
5.C) Falso, id_equipo en la tabla Etapa no es una PK y principalmente el atributo de etapa no existe en la tabla.
5.D) Falso, nro_auto en la tabla Etapa no es una PK por lo que no puede hacer una FK.
5.E) Verdadero, id_equipo en la tabla Equipo es PK por lo que puede hacer una FK con Auto.
5.F) Verdadero, id_equipo y nro_auto en la tabla Auto son PK por lo que pueden hacer una FK con la tabla Desemp-Etapa.

Trabajo Práctico N5 parte 2:

1. Primero completa la TABLA DE RESTRICCIONES que se encuentra debajo con lo siguiente:

1.A) clasifica según corresponda cada una de las restricciones de integridad enunciadas (de dominio o atributo, de tupla, de tabla o general).
1.B) identificá el recurso declarativo más apropiado para implementar la restricción de acuerdo al estándar SQL3. (DOMAIN, CHECK o ASSERTION)

2) Segundo plantea la/las sentencia/s en SQL3 estándar para realizar cada uno de los controles solicitados (ALTER TABLE o CREATE ASSERTION).

3) Tercero, crea cada esquema correspondiente y plantea las sentencias de alteración de tablas de las restricciones que puedan ser soportadas en PostgreSQL.

Restricción	Tabla/s	Atributo/s	Tipo de restricción
Check	Voluntario	fecha_nacimiento	Atributo
Assertion	(Voluntario, Insitucion)	(horas_aportadas nro_voluntario id_coordinador id_institucion)	Global
Assertion	(Voluntario,	(horas_aportadas,	Global

	Tarea)	min_horas, max_horas)	
Assertion	Voluntario	(id_tarea, id_coordinador, id_institucion, nro_voluntario)	Global
Assertion	(Voluntario Historico)	(nro_voluntario, id_institucion, fecha_inicio, fecha_fin)	Global
Check	Historico	(fecha_inicio, fecha_fin)	Tupla

4. Considere las siguientes restricciones que debe definir sobre el esquema de la BD de Voluntarios:

4.A) No puede haber voluntarios de más de 70 años.
4.B) Ningún voluntario puede aportar más horas que las de su coordinador.
4.C) Las horas aportadas por los voluntarios deben estar dentro de los valores máximos y mínimos consignados en la tarea.
4.D) Todos los voluntarios deben realizar la misma tarea que su coordinador.
4.E) Los voluntarios no pueden cambiar de institución más de tres veces en el año.
4.F) En el histórico, la fecha de inicio debe ser siempre menor que la fecha de finalización.

```
4.A) ALTER TABLE unc_esq.voluntario.voluntario
ADD CONSTRAINT ck_edad_voluntarios
CHECK (age(fecha_nacimiento) < 70);

4.B) CREATE ASSERTION horas_aportadas
CHECK (NOT EXISTS (
                SELECT 1
                FROM unc_esq_voluntario.voluntario v JOIN unc_esq_voluntario.voluntario c
                ON (v.nro_voluntario = c.id coordinador)
                JOIN unc_esq_voluntario.institucion i ON (v.id_institucion = i.id_institucion)
                GROUP BY c.id_institucion HAVING (v.horas_aportadas >
        c.horas_aportadas)));

4.C) CREATE ASSERTION horas_aportadas_min_max
CHECK (NOT EXISTS (
                SELECT 1
```

```
        FROM unc_esq_voluntario.voluntario v JOIN unc_esq_voluntario.tarea t ON
        (v.id_tarea = t.id_tarea)
        WHERE v.horas_aportadas NOT BETWEEN t.min_horas AND t.max_horas));
```

4.D)
```
CREATE ASSERTION tarea_coordinador
CHECK (NOT EXISTS (
        SELECT 1
        FROM unc_esq_voluntario.voluntario v JOIN unc_esq_voluntario.voluntario c
        ON (v.nro_voluntario = c.id_coordinador)
        WHERE v.id_institucion = c.id_institucion
        AND v.id_tarea NOT LIKE c.id_tarea));
```

4.E)
```
CREATE ASSERTION cambio_institucion
CHECK (NOT EXISTS (
        SELECT 1
        FROM unc_esq_voluntario.voluntario v JOIN unc_esq_voluntario.historico h
        ON (v.nro_voluntario = h.nro_voluntario)
        GROUP BY h.nro_voluntario
        HAVING COUNT(h.id_insitucion) > 3
        BETWEEN h.fecha_inicio AND h.fecha_fin));
```

4.F)
```
ALTER TABLE unc_esq_voluntario.historico
ADD CONSTRAINT ck_fecha_inicio_menor
CHECK (fecha_inicio < fecha_fin);
```

5. Considere las siguientes restricciones que debe definir sobre el esquema de la BD de Películas:

5.A) Para cada tarea el sueldo máximo debe ser mayor que el sueldo mínimo.
5.B) No puede haber más de 70 empleados en cada departamento.
5.C) Los empleados deben tener jefes que pertenezcan al mismo departamento.
5.D) Todas las entregas, tienen que ser de películas de un mismo idioma.
5.E) No pueden haber más de 10 empresas productoras por ciudad.
5.F) Para cada película, si el formato es 8mm, el idioma tiene que ser francés.
5.G) El teléfono de los distribuidores Nacionales debe tener la misma característica que la de su distribuidor mayorista.
5.H) Las fechas de entrega de las películas no debe ser menor a “hoy”.

5.A)
```
ALTER TABLE unc_esq_pelicula.tarea
ADD CONSTRAINT ck_sueldo_maximo_mayor
CHECK (sueldo_maximo > sueldo_minimo);
```

5.B)
```
ALTER TABLE unc_esq_pelicula.empleado
ADD CONSTRAINT ck_maximo_empleados
CHECK (NOT EXISTS (
        SELECT 1
        FROM unc_esq_pelicula.empleado
        GROUP BY id_departamento, id_distribuidor
```

```
        HAVING COUNT(id_empleado) > 70));
```

5.C)
```
CREATE ASSERTION mismo_departamento
CHECK (NOT EXISTS(
        SELECT 1
        FROM unc_esq_pelicula.empleado e JOIN unc_esq_pelicula.empleado j
        ON (e.id_empleado = j.id_jefe)
        WHERE e.id_departamento NOT LIKE j.id_departamento));
```

5.D)

Todas las entregas, tienen que ser de películas de un mismo idioma.

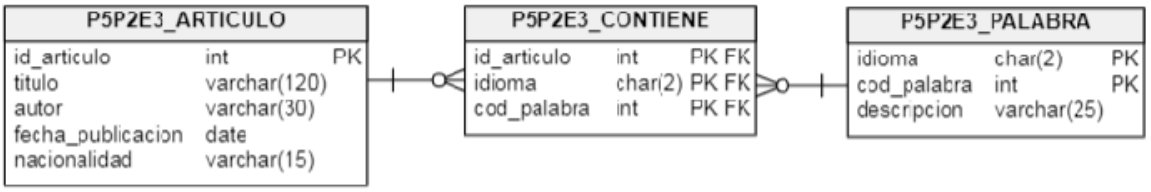
5.E)
```
ALTER TABLE unc_esq_pelicula.empresa_productora
ADD CONSTRAINT ck_max_empresas_ciudad
CHECK (NOT EXISTS (
        SELECT 1
        FROM unc_esq_pelicula.empresa_productora
        GROUP BY id_ciudad
        HAVING COUNT(codigo_productora) > 10));
```

5.F)
```
ALTER TABLE unc_esq_pelicula.pelicula
ADD CONSTRAINT ck_formato_idioma
CHECK ((formato LIKE ‘8mm’ AND idioma LIKE ‘Fracés’) OR formato NOT LIKE ‘8mm’);
```

5.G)
```
CREATE ASSERTION misma_caracteristica
CHECK NOT EXISTS(
        SELECT 1
        FROM unc_esq_pelicula.distribuidor d JOIN unc_esq_pelicula.nacional n
        ON (d.id_distribuidor = n.id_distribuidor)
        JOIN unc_esq_pelicula.internacional i
        ON (n.id_distrib_mayorista = i.id_distribuidor)
        JOIN unc_esq_pelicula.distribuidor di
        ON (i.id_distribuidor = di.id_distribuidor)
        WHERE LEFT(n.telefono, 3) NOT LIKE LEFT(di.telefono, 3));
```

5.H)
```
ALTER TABLE unc_esq_pelicula.entrega
ADD CONSTRAINT ck_fecha_menor_hoy
CHECK (fecha_entrega > NOW());
```

6. Para el esquema de la figura que corresponde al ejercicio 1 del Trabajo Práctico 5 cuyo script de creación de tablas lo podes descargar de aquí.



6.A) Controlar que las nacionalidades sean ‘Argentina’, ‘Español’, ‘Inglés’, ‘Alemán’ o ‘Chilena’.
6.B) Para las fechas de publicaciones se debe considerar que sean fechas posteriores o iguales al 2010.
6.C) Cada palabra clave puede aparecer como máximo en 5 artículos.
6.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.

6.A) ALTER TABLE P5P2E3_ARTICULO
ADD CONSTRAINT ck_nacionalidad
CHECK (nacionalidad IN (‘Argentina’, ‘Español’, ‘Inglés’, ‘Alemán’, ‘Chilena’));

6.B) ALTER TABLE P5P2E3_ARTICULO
ADD CONSTRAINT ck_fechas_publicaciones
CHECK (EXTRACT(YEAR FROM fecha_publicacion) > 2009);

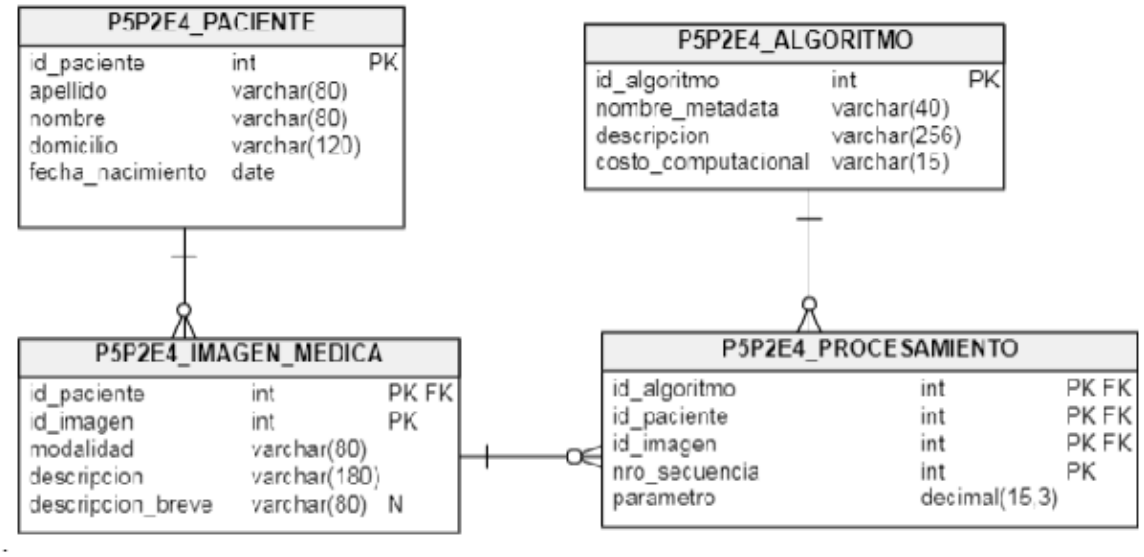
6.C) ALTER TABLE P5P2E3_CONTIENE
ADD CONSTRAINT ck_max_palabras_articulo
CHECK (NOT EXISTS (
SELECT 1
FROM P5P2E3_CONTIENE
GROUP BY id_articulo, idioma
HAVING COUNT(cod_palabra) > 5));

6.D) CREATE ASSERTION palabras_articulo_autor
CHECK (NOT EXISTS (
SELECT 1
FROM P5P2E3_ARTICULO
WHERE (nacionalidad LIKE ‘Argentina’
AND id_articulo IN (**#2**
SELECT id_articulo
FROM P5P2E3_CONTIENE
GROUP BY id_articulo
HAVING COUNT(cod_palabra) > 15)
OR (nacionalidad NOT LIKE ‘Argentina’
AND id_articulo IN (**#1**
SELECT id_articulo
FROM P5P2E3_CONTIENE
GROUP BY id_articulo

HAVING COUNT(cod_palabra) > 10))));

#1 Selecciona el artículo que tenga más de 10 palabras.
#2 Selecciona el artículo que tiene más de 15 palabras.

7. Para el esquema de la figura cuyo script de creación de tablas lo podes descargar de aquí.



7.A) La modalidad de la imagen médica puede tomar los siguientes valores RADIOLOGIA CONVENCIONAL, FLUOROSCOPIA, ESTUDIOS RADIOGRAFICOS CON FLUOROSCOPIA, MAMOGRAFIA, SONOGRAFIA.
7.B) Cada imagen no debe tener más de 5 procesamientos.
7.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.
7.D) Cada paciente sólo puede realizar dos FLUOROSCOPIA anuales.
7.E) No se pueden aplicar algoritmos de costo computacional "O(n)" a imágenes de FLUOROSCOPIA

7.A) ALTER TABLE P5P2E4_IMAGEN_MEDICA
ADD CONSTRAINT ck_modalidades
CHECK (modalidad IN ('RADIOLOGIA CONVENCIONAL', 'FLUOROSCOPIA', 'ESTUDIOS RADIOGRAFICOS CON FLUOROSCOPIA', 'MAMOGRAFIA', 'SONOGRAFIA'));

7.B) ALTER TABLE P5P2E4_PROCESAMIENTO
ADD CONSTRAINT ck_max_procesamientos
CHECK (NOT EXISTS (
SELECT 1
FROM P5P2E4_PROCESAMIENTO
GROUP BY id_paciente, id_imagen
HAVING COUNT(*) > 5));

7.C) ALTER TABLE P5P2E4_IMAGEN_MEDICA
ADD COLUMN fecha_img date;

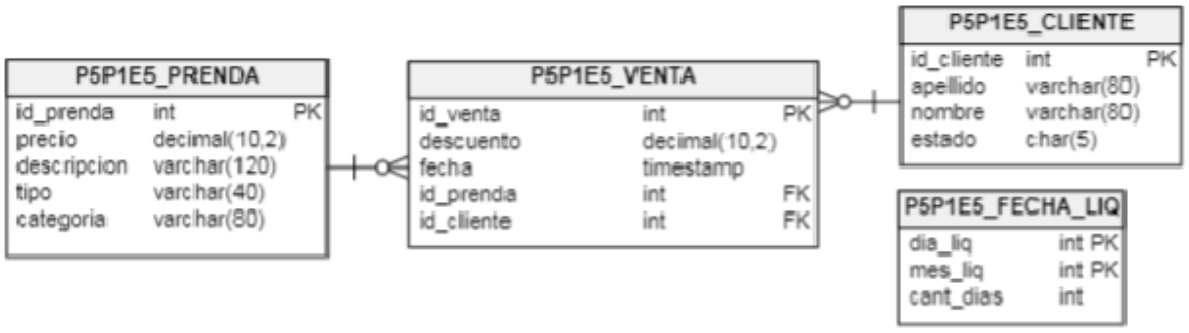
ALTER TABLE P5P2E4_PROCESAMIENTO
ADD COLUMN fecha_proc date;

CREATE ASSERTION fecha_menor
CHECK (NOT EXISTS (
SELECT 1
FROM P5P2E4_PROCESAMIENTO p JOIN P5P2E4_IMAGEN_MEDICA im
ON (p.id_paciente = im.id_paciente AND p.id_imagen = im.id_imagen)
WHERE p.fecha_proc < im.fecha_img));

7.D) ALTER TABLE P5P2E4_IMAGEN_MEDICA
ADD CONSTRAINT ck_max_fluoroscopia_anual
CHECK (NOT EXISTS (
SELECT 1
FROM P5P2E4_IMAGEN_MEDICA
WHERE modalidad LIKE 'FLUOROSCOPIA'
GROUP BY id_paciente, EXTRACT(YEAR FROM fecha_img)
HAVING COUNT(id_imagen) > 2));

7.E) CREATE ASSERTION costo_computacional_fluoroscopia
CHECK (NOT EXISTS (
SELECT 1
FROM P5P2E4_IMAGEN_MEDICA im JOIN P5P2E4_PROCESAMIENTO p
ON (im.id_imagen = p.id_imagen AND im.id_paciente = p.id_paciente)
JOIN P5P2E4_ALGORITMO a
ON (p.id_algoritmo = a.id_algoritmo)
WHERE im.modalidad LIKE 'FLUOROSCOPIA'
AND a.costo_computacional LIKE 'O(n)'
));

8. Para el esquema de la figura cuyo script de creación de tablas lo podes descargar de aquí.

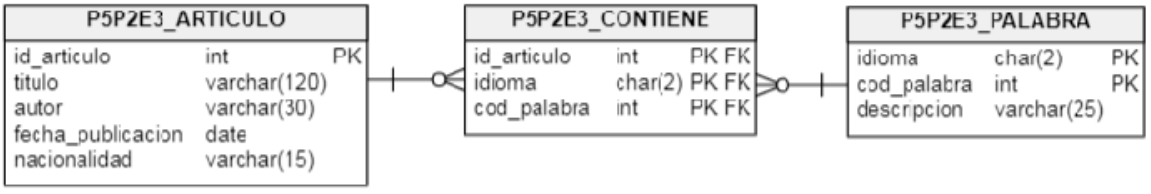


8.A) Los descuentos en las ventas son porcentajes y deben estar entre 0 y 100.
8.B) Los descuentos realizados en fechas de liquidación deben superar el 30%.
8.C) Las liquidaciones de Julio y Diciembre no deben superar los 5 días.
8.D) Las prendas de categoría 'oferta' no tienen descuentos.

```
8.A) ALTER TABLE P5P1E5_VENTA
ADD CONSTRAINT ck_porcentaje_venta
CHECK (descuento BETWEEN 0 AND 100);
8.B) CREATE ASSERTION descuentos_liquidacion
CHECK (NOT EXISTS (
                SELECT 1
                FROM P5P1E5_VENTA, P5P1E5_FECHA_LIQ
                WHERE EXTRACT(DAY FROM fecha)
                BETWEEN dia_liq AND (dia_liq + cant_dias)
                AND EXTRACT(MONTH FROM fecha) LIKE mes_liq)
                AND descuento > 30);

8.C) ALTER TABLE P5P1E5_FECHA_LIQ
ADD CONSTRAINT ck_liquidacion
CHECK (NOT EXISTS (
                SELECT 1
                FROM P5P1E5_FECHA_LIQ
                WHERE mes_liq LIKE 07 OR mes_liq LIKE 12
                AND cant_dias > 5));

8.D) CREATE ASSERTION prendas_oferta
CHECK (NOT EXISTS (
                SELECT 1
                FROM P5P1E5_PRENDA p JOIN P5P1E5_VENTA v
                ON (p.id_prenda = v.id_prenda)
                WHERE p.categoria LIKE 'oferta'
                AND v.descuento <> 0));
```

Trabajo Práctico N6 parte 1:

1. Implemente de manera procedural las restricciones que no pudo realizar de manera declarativa en el ejercicio 3 del Práctico 5 Parte 2; cuyo script de creación del esquema se encuentra aquí. Ayuda: las restricciones que no se pudieron realizar de manera declarativa fueron las de los items C y D; la solución declarativa está aquí.



1.C) Cada palabra clave puede aparecer como máximo en 5 artículos.

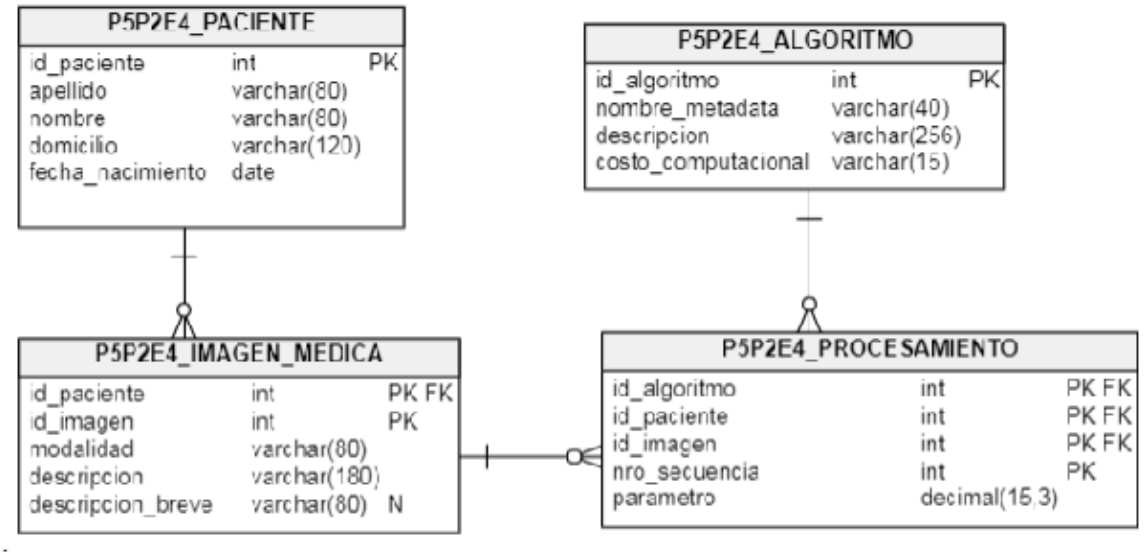
```
1.C) CREATE OR REPLACE FUNCTION FN_MAXIMO_PALABRAS_CLAVE() RETURNS
Trigger AS $$
BEGIN
        IF ((SELECT COUNT(*)
                FROM P5P2E3_CONTIENE
                WHERE id_articulo = NEW.id_articulo) > 4 ) THEN
                RAISE EXCEPTION 'Superó la cantidad de palabras claves en el artículo %',
                NEW.id_articulo;
        END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_MAXIMO_PALABRAS_CLAVE
BEFORE INSERT OR UPDATE OF id_articulo
ON CONTIENE
FOR EACH ROW EXECUTE PROCEDURE FN_MAXIMO_PALABRAS_CLAVE();
```

1.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.

```
1.D) CREATE OR REPLACE FUNCTION
FN_MAXIMO_PALABRAS_NACIONALIDAD_ART() RETURNS Trigger AS $$
DECLARE cant integer;
BEGIN
        SELECT COUNT(*) INTO cant
        FROM P5P2E3_CONTIENE
        WHERE id_articulo = NEW.id_articulo;

        IF ((NEW.nacionalidad = ‘Argentina’ AND cant > 14) OR (NEW.nacionalidad !=
                ‘Argentina’ AND cant > 9)) THEN
                RAISE EXCEPTION 'Superó la cantidad de palabras claves en el artículo';
        END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_MAXIMO_PALABRAS_NACIONALIDAD_ART
BEFORE UPDATE OF nacionalidad
ON P5P2E3_ARTICULO
FOR EACH ROW
EXECUTE PROCEDURE FN_MAXIMO_PALABRAS_NACIONALIDAD_ART();
```

```
CREATE OR REPLACE FUNCTION
FN_MAXIMO_PALABRAS_NACIONALIDAD_CONTIENE() RETURNS Trigger AS $$
DECLARE
        cant integer;
        nac P5P2E3_ARTICULO.NACIONALIDAD%type;
BEGIN
        SELECT COUNT(*) INTO cant
        FROM P5P2E3_CONTIENE
        WHERE id_articulo = NEW.id_articulo;

        SELECT nacionalidad INTO nac
        FROM P5P2E3_ARTICULO
        WHERE id_articulo = NEW.id_articulo;

        IF ((nac = 'Argentina' AND cant > 14) OR (nac != 'Argentina' AND cant > 9)) THEN
                RAISE EXCEPTION 'Superó la cantidad de palabras claves en el artículo';
        END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_MAXIMO_PALABRAS_NACIONALIDAD_CONTIENE
BEFORE INSERT OR UPDATE OF id_articulo
ON P5P2E3_CONTIENE
FOR EACH ROW
EXECUTE PROCEDURE FN_MAXIMO_PALABRAS_NACIONALIDAD_CONTIENE();
```

2. Implemente de manera procedural las restricciones que no pudo realizar de manera declarativa en el ejercicio 4 del Práctico 5 Parte 2; cuyo script de creación del esquema se encuentra aquí. Ayuda: las restricciones que no se pudieron realizar de manera declarativa fueron las de los items B, C, D, E; la solución declarativa está aquí.



2.B) Cada imagen no debe tener más de 5 procesamientos.

```
2.B)  CREATE OR REPLACE FUNCTION FN_MAXIMO_PROCESAMIENTOS() RETURNS
Trigger AS $$
DECLARE
        cant integer;
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 'Superó la cantidad de procesamientos por imagen'
        END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_MAXIMO_PROCESAMIENTOS
BEFORE INSERT OR UPDATE OF id_imagen, id_paciente
ON P5P2E4_PROCESAMIENTO
FOR EACH ROW
EXECUTE PROCEDURE FN_MAXIMO_PROCESAMIENTOS();
```

2.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.

```
2.C) CREATE OR REPLACE FUNCTION FN_FECHAS_PROCESAMIENTO() RETURNS
Trigger AS $$
```

```
DECLARE
	fechaImg P5P2E3_IMAGEN_MEDICA.fecha_img%type;
BEGIN
	SELECT fecha_img INTO fechaImg
	FROM P5P2E3_IMAGEN_MEDICA
	WHERE id_paciente = NEW.id_paciente
	AND id_imagen = NEW.id_imagen;

	IF (NEW.fecha_proc > fechaImg) THEN
		RAISE EXCEPTION 'La fecha de procesamiento no puede ser mayor que la
		fecha de la imagen';
	END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_FECHAS_PROCESAMIENTO
BEFORE INSERT OR UPDATE OF fecha_proc, id_imagen, id_paciente
ON P5P2E4_PROCESAMIENTO
FOR EACH ROW
EXECUTE PROCEDURE FN_FECHAS_PROCESAMIENTO();

CREATE OR REPLACE FUNCTION FN_FECHAS_IMG_MEDICA() RETURNS Trigger AS
$$
DECLARE
	fechaProc P5P2E3_PROCESAMIENTO.fecha_proc%type;
BEGIN
	SELECT fecha_proc INTO fechaProc
	FROM P5P2E3_PROCESAMIENTO
	WHERE id_paciente = NEW.id_paciente
	AND id_imagen = NEW.id_imagen;

	IF (fechaProc > NEW.fecha_img) THEN
		RAISE EXCEPTION 'La fecha de procesamiento no puede ser mayor que la
		fecha de la imagen';
	END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_FECHAS_IMG_MEDICA
BEFORE UPDATE OF fecha_img
ON P5P2E4_IMAGEN_MEDICA
FOR EACH ROW
EXECUTE PROCEDURE FN_FECHAS_IMG_MEDICA();
```

2.D) Cada paciente sólo puede realizar dos FLUOROSCOPIA anuales.

2.D)
```
CREATE OR REPLACE FUNCTION FN_MAXIMO_FLUOROSCOPIA_ANUAL()
RETURNS Trigger AS $$
DECLARE
	cant integer;
BEGIN
	SELECT COUNT(*) INTO cant
	FROM P5P2E4_IMAGEN_MEDICA
	WHERE id_paciente = NEW.id_paciente
	AND EXTRACT(YEAR FROM fecha_img) = EXTRACT(YEAR
	FROM NEW.fecha_img);

	IF (cant > 1)
		THEN RAISE EXCEPTION ‘El paciente ya tiene 2 fluoroscopias este año’;
	END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_MAXIMO_FLUOROSCOPIA_ANUAL
BEFORE INSERT OR UPDATE OF fecha_img, modalidad
ON P5P2E4_IMAGEN_MEDICA
FOR EACH ROW
WHEN (NEW.modalidad LIKE ‘FLUOROSCOPIA’)
EXECUTE PROCEDURE FN_MAXIMO_FLUOROSCOPIA_ANUAL();
```

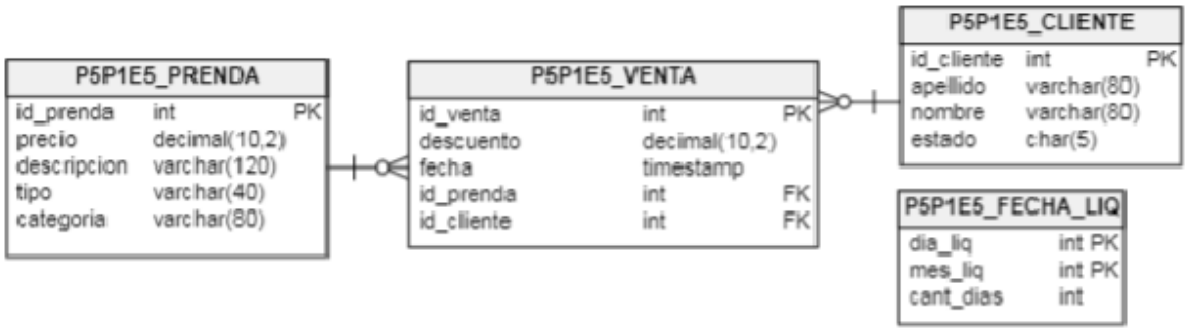
2.E) No se pueden aplicar algoritmos de costo computacional “O(n)” a imágenes de FLUOROSCOPIA.

2.E)
```
CREATE OR REPLACE FUNCTION FN_ALGORITMO_FLUOROSCOPIA_PROC()
RETURNS Trigger AS $$
DECLARE
	modalidad P5P2E3_IMAGEN_MEDICA.modalidad%type;
	costo P5P2E3_ALGORITMO.costo_computacional%type;
BEGIN
	SELECT im.modalidad INTO modalidad
	FROM P5P2E4_PROCESAMIENTO p JOIN P5P2E4_IMAGEN_MEDICA im
	ON (NEW.id_imagen = im.id_image AND NEW.id_paciente = im.id_paciente);

	SELECT a.costo_computacional INTO costo
	FROM P5P2E4_PROCESAMIENTO p JOIN P5P2E4_ALGORITMO a
	ON (NEW.id_algoritmo = a.id_algoritmo);

	IF (modalidad LIKE ‘FLUOROSCOPIA’ AND costo LIKE ‘O(n)’) THEN
		RAISE EXCEPTION ‘No puede tener costo O(n) en FLUOROSCOPIA’;
	END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER TR_ALGORITMO_FLUOROSCOPIA_PROC
BEFORE INSERT OR UPDATE OF id_imagen, id_paciente, id_algoritmo
ON P5P2E4_PROCESAMIENTO
FOR EACH ROW
EXECUTE PROCEDURE FN_ALGORITMO_FLUOROSCOPIA_PROC();

CREATE OR REPLACE FUNCTION FN_ALGORITMO_FLUOROSCOPIA_ALG()
RETURNS Trigger AS $$
DECLARE
	modalidad P5P2E3_IMAGEN_MEDICA.modalidad%type;
BEGIN
	SELECT im.modalidad INTO modalidad
	FROM P5P2E4_ALGORITMO a JOIN P5P2E4_PROCESAMIENTO p
	ON (NEW.id_algoritmo = p.id_algoritmo) JOIN P5P2E4_IMAGEN_MEDICA im
	ON (p.id_imagen = im.id_imagen);

	IF (modalidad LIKE ‘FLUOROSCOPIA’)
		THEN RAISE EXCEPTION ‘No puede tener costo O(n) en FLUOROSCOPIA’;
	END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_ALGORITMO_FLUOROSCOPIA_ALG
BEFORE UPDATE OF costo_computacional
ON P5P2E4_ALGORITMO
FOR EACH ROW
WHEN (NEW.costo_computacional LIKE ‘O(n)’)
EXECUTE PROCEDURE FN_ALGORITMO_FLUOROSCOPIA_ALG();

CREATE OR REPLACE FUNCTION FN_ALGORITMO_FLUOROSCOPIA_IMG() RETURNS
Trigger AS $$
DECLARE
	costo P5P2E3_ALGORITMO.costo_computacional%type;
BEGIN
	SELECT a.costo_computacional INTO costo
	FROM P5P2E3_IMAGEN_MEDICA im JOIN P5P2E3_PROCESAMIENTO p
	ON (NEW.id_imagen = p.id_imagen) JOIN P5P2E3_ALGORITMO a
	ON (p.id_algoritmo = a.id_algoritmo);

	IF (costo LIKE ‘O(n)’)
		THEN RAISE EXCEPTION ‘No puede tener costo O(n) en FLUOROSCOPIA’;
	END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER TR_ALGORITMO_FLUOROSCOPIA_IMG
BEFORE UPDATE OF modalidad
ON P5P2E4_IMAGEN_MEDICA
FOR EACH ROW
WHEN (NEW.modalidad LIKE 'FLUOROSCOPIA')
EXECUTE PROCEDURE FN_ALGORITMO_FLUOROSCOPIA_IMG();
```

3. Implemente de manera procedural las restricciones que no pudo realizar de manera declarativa en el ejercicio 5 del Práctico 5 Parte 2; cuyo script de creación del esquema se encuentra aquí. Ayuda: las restricciones que no se pudieron realizar de manera declarativa fueron B, C, D, E



3.B) Los descuentos realizados en fechas de liquidación deben superar el 30%.

```
3.B) CREATE OR REPLACE FUNCTION FN_DESCUENTO_LIQUIDACION_VENTA()
RETURNS Trigger AS $$
DECLARE
        dia integer;
        mes integer;
        cant integer;
BEGIN
        SELECT dia_liq INTO dia
        FROM P5P1E5_FECHA_LIQ;

        SELECT mes_liq INTO mes
        FROM P5P1E5_FECHA_LIQ;

        SELECT cant_dias INTO cant
        FROM P5P1E5_FECHA_LIQ;

        IF ((EXTRACT(DAY FROM NEW.fecha) BETWEEN dia
                AND (dia + cant) AND EXTRACT(MONTH FROM NEW.fecha) = mes))
                        THEN RAISE EXCEPTION 'El descuento no puede ser mayor que 30
                        por ciento';
        END IF;
RETURN NEW;
END $$
```

```
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_DESCUENTO_LIQUIDACION_VENTA
BEFORE INSERT OR UPDATE OF descuento, fecha
ON P5P1E5_VENTA
FOR EACH ROW
WHEN (NEW.descuento > 30)
EXECUTE PROCEDURE FN_DESCUENTO_LIQUIDACION_VENTA();

CREATE OR REPLACE FUNCTION FN_DESCUENTO_LIQUIDACION_FECHA_LIQ()
RETURNS Trigger AS $$
DECLARE
        descuento P5P1E5_VENTA.descuento%type;
BEGIN
        SELECT v.descuento INTO descuento
        FROM P5P1E5_VENTA v
        WHERE EXTRACT(DAY FROM v.fecha) BETWEEN NEW.dia_liq AND (NEW.dia_liq
        + NEW.cant_dias)
        AND EXTRACT(MONTH FROM v.fecha) = NEW.mes_liq;

        IF (descuento > 30)
                THEN RAISE EXCEPTION ‘El descuento no puede ser mayor que 30
                por ciento’;
        END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_DESCUENTO_LIQUIDACION_FECHA_LIQ
BEFORE INSERT OR UPDATE OF dia_liq, mes_liq, cant_dias
ON P5P1E5_FECHA_LIQ
FOR EACH ROW
EXECUTE PROCEDURE FN_DESCUENTO_LIQUIDACION_FECHA_LIQ();
```

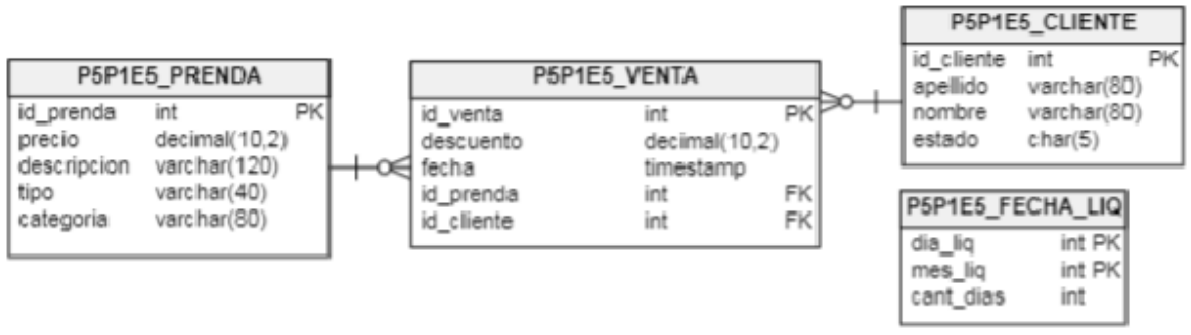
3.C) Las liquidaciones de Julio y Diciembre no deben superar los 5 días.

```
8.C) CREATE OR REPLACE FUNCTION FN_LIQUIDACIONES_JULIO_DICIEMBRE()
RETURNS Trigger AS $$
BEGIN
        RAISE EXCEPTION ‘Las liquidaciones de Julio y diciembre no deben
        superar los 5 días’;
END $$
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_LIQUIDACIONES_JULIO_DICIEMBRE
BEFORE INSERT OR UPDATE OF mes_liq, cant_dias
ON P5P1E5_FECHA_LIQ
FOR EACH ROW
```

```
WHEN (NEW.cant_dias > 5 AND (NEW.mes_liq = 07 OR NEW.mes_liq = 12))
EXECUTE PROCEDURE FN_LIQUIDACIONES_JULIO_DICIEMBRE();
```

3.D) Las prendas de categoría ‘oferta’ no tienen descuentos.

```
8.D) CREATE OR REPLACE FUNCTION FN_OFERTA_DESCUENTO_PRENDA()
RETURNS Trigger AS $$
DECLARE
        descuento P5P1E5_VENTA.descuento%type;
BEGIN
        SELECT v.descuento INTO descuento
        FROM P5P1E5_VENTA v
        WHERE NEW.id_prenda = v.id_prenda;

        IF (descuento > 0)
                THEN RAISE EXCEPTION ‘Las prendas con categoría oferta no tienen
                descuento’;
        END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_OFERTA_DESCUENTO_PRENDA
BEFORE UPDATE OF categoria
ON P5P1E5_PRENDA
FOR EACH ROW
WHEN (NEW.categoria LIKE ‘oferta’)
EXECUTE PROCEDURE FN_OFERTA_DESCUENTO_PRENDA();
```




```
CREATE OR REPLACE FUNCTION FN_OFERTA_DESCUENTO_VENTA() RETURNS
Trigger AS $$
DECLARE
        categoria P5P1E5_PRENDA.categoria%type;
BEGIN
        SELECT p.categoria INTO categoria
        FROM P5P1E5_PRENDA p
        WHERE NEW.id_prenda = p.id_prenda;
```

```
        IF (categoria LIKE ‘oferta’)
                THEN RAISE EXCEPTION ‘Las prendas con categoría oferta no tienen
                descuento’;
        END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_OFERTA_DESCUENTO_VENTA
BEFORE INSERT OR UPDATE OF descuento
ON P5P1E5_VENTA
FOR EACH ROW
WHEN (NEW.descuento > 0)
EXECUTE PROCEDURE FN_OFERTA_DESCUENTO_VENTA();
```

4.A) Copie en su esquema la estructura de la tabla PELICULA del esquema unc_peliculas:
CREATE TABLE Pelicula AS SELECT * FROM unc_esq_peliculas.pelicula;

4.B) Cree la tabla ESTADISTICA con la siguiente sentencia:
CREATE TABLE estadistica AS SELECT genero, COUNT(*) total_peliculas, count (distinct idioma) cantidad_idiomas FROM Pelicula GROUP BY genero;

4.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: La granularidad del trigger, eventos ante los cuales se debe disparar y analice si conviene modificar por cada operación de actualización o reconstruirla de cero.

La granularidad la hice FOR EACH STATEMENT y se dispara por INSERT, DELETE y UPDATE OF genero, idioma ya que así es más fácil poder mantener la información de la tabla estadistica ya que por cada modificación que se hace en la tabla pelicula, se va a ver reflejada en la tabla estadistica porque vamos a copiar la consulta sql. Con FOR EACH ROW está el problema de que debemos detectar si estamos queriendo hacer un INSERT, DELETE o UPDATE y plantear demasiada lógica en cada una y se hace inmantenible.

```
4.C) CREATE OR REPLACE FUNCTION FN_ACTUALIZAR_ESTADISTICA() RETURNS
Trigger AS $$
BEGIN
        DELETE FROM unc_esq_pelicula.estadistica;

        INSERT INTO unc_esq_estadistica AS
        (SELECT genero, COUNT(*) AS total_peliculas, COUNT(DISTINCT idioma) AS
        cantidad_idiomas
        FROM unc_esq_pelicula.pelicula
        GROUP BY genero);
END $$
LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER TR_ACTUALIZAR_ESTADISTICA
AFTER INSERT OR DELETE OR UPDATE OF genero, idioma
ON unc_esq_pelicula.pelicula
FOR EACH STATEMENT
EXECUTE PROCEDURE FN_ACTUALIZAR_ESTADISTICA();
```

Trabajo Práctico N6 parte 2:

1. Para el esquema unc_voluntarios considere que se quiere mantener un registro de quién y cuándo realizó actualizaciones sobre la tabla TAREA en la tabla HIS_TAREA. Dicha tabla tiene la siguiente estructura: HIS_TAREA(nro_registro, fecha, operación, cant_reg_afectados, usuario)

1.A) Provea el/los trigger/s necesario/s para mantener en forma automática la tabla HIS_TAREA cuando se realizan actualizaciones (insert, update o delete) en la tabla TAREA.
1.B) Muestre los resultados de las tablas si se ejecuta la operación:

```
DELETE FROM TAREA
WHERE id_tarea like 'AD%';
```

Según el o los triggers definidos sean FOR EACH ROW o FOR EACH STATEMENT. Evalúe la diferencia entre ambos tipos de granularidad.

```
1.A) CREATE TABLE IF NOT EXISTS HIS_TAREA (
        nro_registro SERIAL NOT NULL,
        fecha date NOT NULL,
        operacion varchar(6)) NOT NULL,
        cant_reg_afectados integer NOT NULL,
        usuario varchar(100) NOT NULL,
        CONSTRAINT pk_nro_registro PRIMARY KEY (nro_registro));

CREATE OR REPLACE FUNCTION FN_ACTUALIZAR_HIS_TAREA() RETURNS Trigger
AS $$
DECLARE
        usuario varchar := TG_TABLE_SCHEMA;
        cant := 0;
        registro integer := 0;
BEGIN
        INSERT INTO HIS_TAREA (fecha, operacion, cant_reg_afectados, usuario)
        VALUES (NOW(), TG_OP, 0, usuario) RETURNING nro_registro INTO registro;
        cant = @@ROWCOUNT;
        UPDATE HIS_TAREA SET cant_reg_afectados = cant WHERE nro_registro =
        registro;
        RETURN NEW;
END $$
```

```
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_ACTUALIZAR_HIS_TAREA
AFTER INSERT OR DELETE OR UPDATE
ON unc_esq_voluntario.tarea
FOR EACH ROW
EXECUTE PROCEDURE FN_ACTUALIZAR_HIS_TAREA();
```

Está pensado para que sea FOR EACH ROW porque por cada operación voy a disparar a la función para que deje un registro de cada uno.

2. A partir del esquema unc_peliculas, realice procedimientos para:

2.A) Completar una tabla denominada MAS_ENTREGADAS con los datos de las 20 películas más entregadas en los últimos seis meses desde la ejecución del procedimiento. Esta tabla por lo menos debe tener las columnas código_pelicula, nombre, cantidad_de_entregas (en caso de coincidir en cantidad de entrega ordenar por código de película).
2.B) Generar los datos para una tabla denominada SUELDOS, con los datos de los empleados cuyas comisiones superen a la media del departamento en el que trabajan. Esta tabla debe tener las columnas id_empleado, apellido, nombre, sueldo, porc_comision.
2.C) Cambiar el distribuidor de las entregas sucedidas a partir de una fecha dada, siendo que el par de valores de distribuidor viejo y distribuidor nuevo es variable.

```
2.A) CREATE TABLE IF NOT EXISTS MAS_ENTREGADAS(
        codigo_pelicula SERIAL NOT NULL,
        nombre varchar(100) NOT NULL,
        cantidad_de_entregas integer NOT NULL,
        ADD CONSTRAINT pk_codigo_pelicula PRIMARY KEY (codigo_pelicula));

CREATE OR REPLACE FUNCTION FN_ACTUALIZAR_MAS_ENTREGADAS() RETURNS
Trigger AS $$
BEGIN
        DELETE FROM MAS_ENTREGADAS;

        INSERT INTO MAS_ENTREGADAS AS
        (SELECT re.codigo_pelicula, p.nombre, re.cantidad
        FROM unc_esq_pelicula.renglon_entrega re JOIN unc_esq_pelicula.pelicula p
        ON (re.codigo_pelicula = p.codigo_pelicula)
        JOIN unc_esq_pelicula.entrega e
        WHERE e.fecha_entrega - INTERVAL ‘6 months’
        GROUP BY re.codigo_pelicula
        ORDER BY COUNT(re.nro_entrega) DESC);

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

```
CREATE TRIGGER TR_ACTUALIZAR_MAS_ENTREGADAS
AFTER INSERT OR DELETE OR UPDATE
ON unc_esq_pelicula.renglon_entrega
FOR EACH STATEMENT
EXECUTE PROCEDURE FN_ACTUALIZAR_MAS_ENTREGADAS()
```

2.B)
```
CREATE TABLE IF NOT EXISTS SUELDOS (
        id_empleado SERIAL NOT NULL,
        apellido varchar(100) NOT NULL,
        nombre varchar(100) NOT NULL,
        sueldo decimal NOT NULL,
        porc_comision decimal NOT NULL);
```

Generar los datos para una tabla denominada SUELDOS, con los datos de los empleados cuyas comisiones superen a la media del departamento en el que trabajan. Esta tabla debe tener las columnas id_empleado, apellido, nombre, sueldo, porc_comision.

Trabajo Práctico N7:

1. Considere las siguientes sentencias de creación de vistas en el esquema de Películas. Nota: el planteo es sólo teórico porque no podrá insertar registros en unc_esq_peliculas por los permisos

```
CREATE VIEW Distribuidor_200 AS
SELECT id_distribuidor, nombre, tipo
FROM unc_esq_peliculas.distribuidor
WHERE id_distribuidor > 200;
```

```
CREATE VIEW Departamento_dist_200 AS
SELECT id_departamento, nombre, id_ciudad, jefe_departamento
FROM unc_esq_peliculas.departamento
WHERE id_distribuidor > 200;
```

1.A) Discuta si las vistas son actualizables o no y justifique.
1.B) Considere que algunos registros de la tabla Distribuidor son:

id_distribuidor	nombre	direccion	telefono	tipo
1049	Distribuidor 1049	Doro	7372214-6352	N
1050	Distribuidor 1050	Lakhagarh	569842-2643	N

y se ha creado la vista:

```
CREATE VIEW Distribuidor_1000 AS
SELECT *
FROM unc_esq_peliculas.distribuidor d
WHERE id_distribuidor > 1000;
```

Si se ejecuta la siguiente sentencia:

```
INSERT INTO Distribuidor_1000 VALUES (1050, 'NuevoDistribuidor 1050', 'Montiel 340', '569842-2643', 'N');
```

Indique y justifique la opción correcta:

1.C) Falla porque la vista no es actualizable.
1.D) Falla porque si bien la vista es actualizable viola una restricción de foreign key.
1.E) Falla porque si bien la vista es actualizable viola una restricción de primary key.
1.F) Procede exitosamente.

1.A) La primer vista es actualizable ya que no cuenta con subconsultas en el SELECT y tampoco un DISTINCT, contiene todas las claves primarias, en este caso 1 clave y no tiene funciones de agregación ni información derivada.

La segunda vista no es actualizable ya que lo único que le falta una clave primaria que es id_distribuidor. Lo demás está bien.

1.E) Falla porque la PK con el id_distribuidor 1050 ya existe.

2. Considere el esquema de la BD unc_esq_peliculas:
Escriba las sentencias de creación de cada una de las vistas solicitadas en cada caso. Indique si para el estandar SQL y/o Postgresql dicha vista es actualizable o no, si es de Proyección-Selección (una tabla) o Proyección-Selección-Ensamble (más de una tabla). Justifique cada respuesta.

2.A) Cree una vista EMPLEADO_DIST que liste el nombre, apellido, sueldo, y fecha_nacimiento de los empleados que pertenecen al distribuidor cuyo identificador es 20.
2.B) Sobre la vista anterior defina otra vista EMPLEADO_DIST_2000 con el nombre, apellido y sueldo de los empleados que cobran más de 2000.
2.C) Sobre la vista EMPLEADO_DIST cree la vista EMPLEADO_DIST_20_70 con aquellos empleados que han nacido en la década del 70 (entre los años 1970 y 1979).

2.D) Cree una vista PELICULAS_ENTREGADA que contenga el código de la película y la cantidad de unidades entregadas.
2.E) Cree una vista ACCION_2000 con el código, el titulo el idioma y el formato de las películas del género 'Acción' entregadas en el año 2006.
2.F) Cree una vista DISTRIBUIDORAS_ARGENTINA con los datos completos de las distribuidoras nacionales y sus respectivos departamentos.
2.G) De la vista anterior cree la vista Distribuidoras_mas_2_emp con los datos completos de las distribuidoras cuyos departamentos tengan más de 2 empleados.
2.H) Cree la vista PELI_ARGENTINA con los datos completos de las productoras y las películas que fueron producidas por empresas productoras de nuestro país.
2.I) De la vista anterior cree la vista ARGENTINAS_NO_ENTREGADA para las películas producidas por empresas argentinas pero que no han sido entregadas.
2.J) Cree una vista PRODUCTORA_MARKETINERA con las empresas productoras que hayan entregado películas a TODOS los distribuidores.

2.A) CREATE VIEW EMPLEADO_DIST AS
SELECT nombre, apellido, sueldo, fecha_nacimiento
FROM unc_esq_pelicula.empleado
WHERE id_distribuidor = 20

Es de proyección-selección porque usa la tabla empleado. No es actualizable ya que me falta en el SELECT la pk que es id_empleado.

2.B) CREATE VIEW EMPLEADO_DIST_2000 AS
SELECT nombre, apellido, sueldo
FROM EMPLEADO_DIST
WHERE sueldo > 2000;

Es de proyección-selección porque usa la vista de EMPLEADO_DIST. No es actualizable ya que me falta la pk en el SELECT que es id_empleado.

2.C) CREATE VIEW EMPLEADO_DIST_20_70 AS
SELECT *
FROM EMPLEADO_DIST
WHERE EXTRACT(YEAR FROM fecha_nacimiento) BETWEEN 1970 AND 1979;

Es de proyección-selección porque usa la vista EMPLEADO_DIST. No es actualizable ya que me falta la pk en el SELECT que es id_empleado.

2.D) CREATE VIEW PELICULAS_ENTREGADAS AS
SELECT codigo_pelicula, sum(cantidad)
FROM unc_esq_pelicula.renglon_entrega
GROUP BY codigo_pelicula;

Es de proyección-selección porque usa la tabla renglon_entrega. No es actualizable ya que me falta una pk en el SELECT que es nro_entrega, porque tengo un GROUP BY y además una suma en el SELECT.

2.E) CREATE VIEW ACCION_2000 AS
SELECT codigo_pelicula, titulo, idioma, formato
FROM unc_esq_pelicula.pelicula
WHERE genero LIKE ‘Acción’
AND codigo_pelicula = (SELECT codigo_pelicula
FROM unc_esq_pelicula.renglon_entrega
WHERE nro_entrega = (SELECT nro_entrega
FROM unc_esq_pelicula.entrega
WHERE EXTRACT(YEAR FROM fecha_entrega = 2006)));

Es de proyección-selección-ensamble porque usa las tablas pelicula, renglon_entrega y entrega. Es actualizable ya que tengo la pk de codigo_pelicula en el SELECT, no tengo subconsultas ni DISTINCT en el SELECT y no tiene funciones de agregación ni información derivada.

2.F) CREATE VIEW DISTRIBUIDORAS_ARGENTINA AS
SELECT d.id_distribuidor, d.nombre, d.direccion, d.telefono, d.tipo, n.nro_inscripcion, n.encargado, n.id_distrib_mayorista, depto.id_departamento
FROM unc_esq_pelicula.distribuidor d JOIN unc_esq_pelicula.departamento depto
ON (d.id_distribuidor = depto.id_distribuidor)
JOIN unc_esq_pelicula.nacional n ON (d.id_distribuidor = n.id_distribuidor)
WHERE tipo LIKE ‘N’;

Es de proyección-selección-ensamble porque usa las tablas distribuidor, departamento y nacional. No es actualizable porque estoy haciendo ensambles con JOIN.

2.G) CREATE VIEW Distribuidoras_mas_2_emp AS
SELECT *
FROM DISTRIBUIDORAS_ARGENTINA
WHERE (id_distribuidor, id_departamento) IN (SELECT id_distribuidor, id_departamento
FROM unc_esq_pelicula.empleado
GROUP BY id_departamento, id_distribuidor
HAVING COUNT(id_empleado) > 2);

Es de proyección-selección-ensamble porque usa la vista Distribuidoras_mas_2_emp y la tabla empleado. Es actualizable ya que no rompo ninguna regla.

2.H) CREATE VIEW PELI_ARGENTINA AS
SELECT p.codigo_pelicula, p.titulo, p.idioma, p.formato, p.genero, ep.codigo_productora, ep.nombre_productora, c.id_ciudad
FROM unc_esq_pelicula.pelicula p JOIN unc_esq_pelicula.empresa_productora ep
ON (p.codigo_productora = ep.codigo_productora)
JOIN unc_esq_pelicula.ciudad c ON (ep.id_ciudad = c.ciudad)
WHERE c.id_pais LIKE ‘AR’;

Es de proyección-selección-ensamble porque usa las tablas pelicula, empresa_productora y ciudad. No es actualizable porque estoy haciendo ensambles con JOIN.

2.I) CREATE VIEW ARGENTINAS_NO_ENTREGADA AS
SELECT *
FROM PELI_ARGENTINA
WHERE codigo_pelicula NOT IN (SELECT codigo_pelicula
FROM unc_esq_pelicula.renglon_entrega);

Es de proyección-selección-ensamble porque usa la vista de PELI_ARGENTINA y la tabla de renglon_entrega. Es actualizable porque no rompe ninguna regla.

2.J)

3. Analice cuáles serían los controles y el comportamiento ante actualizaciones sobre las vistas EMPLEADO_DIST, EMPLEADO_DIST_2000 y EMPLEADO_DIST_20_70 creadas en el ej. 2, si las mismas están definidas con WITH CHECK OPTION LOCAL o CASCADE en cada una de ellas. Evalúe todas las alternativas.

3. EMPLEADO_DIST WITH LOCAL CHECK OPTION: Va a verificar que el id_distribuidor sea 20.
EMPLEADO_DIST WITH CASCADE CHECK OPTION: Va a verificar la vista que se encuentra en su FROM.

EMPLEADO_DIST_2000 WITH LOCAL CHECK OPTION: Va a verificar que el sueldo sea mayor a 200.
EMPLEADO_DIST_2000 WITH CASCADE CHECK OPTION: Va a verificar la vista que se encuentre en su FROM, en este caso EMPLEADO_DIST y verificará que el sueldo sea mayor a 2000 y que el id_distribuidor sea 20.

EMPLEADO_DIST_20_70 WITH LOCAL CHECK OPTION: Va a verificar que la fecha de nacimiento se encuentre entre 1970 y 1979 inclusive.

EMPLEADO_DIST_20_70 WITH CASCADE CHECK OPTION: Va a verificar que la fecha de nacimiento se encuentre entre 1970 y 1979 inclusive y que el id_distribuidor sea 20.