# EjerciciosTema3-Resueltos.pdf

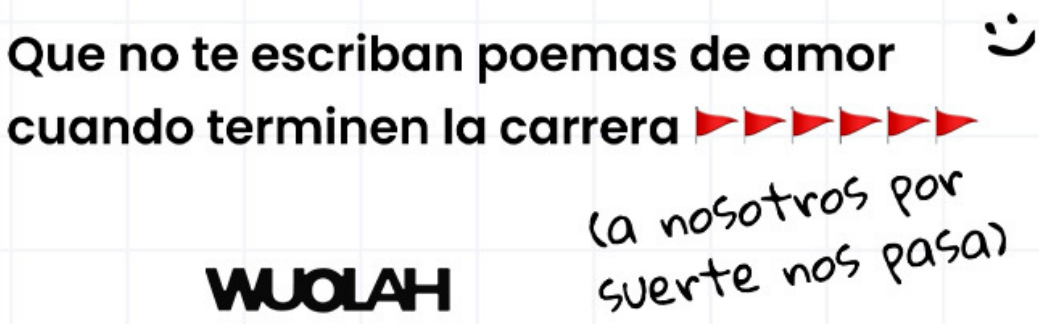

Anónimo

Sistemas Informaticos I

3º Grado en Ingeniería Informática

Escuela Politécnica Superior
Universidad Autónoma de Madrid



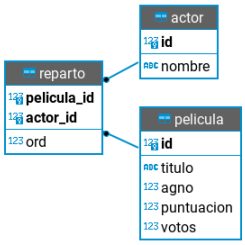


# Sistemas Informáticos I

## Tema 3: Hoja de ejercicios

**1)** Dado el script SQL que podrás encontrar en Moodle, relacionado con una BD de sobre películas:

a) Crea la BD en PostgreSQL a partir del script proporcionado.




b) ¿Qué relaciones se establecen entre cada entidad (cardinalidad)?


c) ¿Por qué la tabla REPARTO tiene una clave primaria compuesta por ACTOR_ID y PELICULA_ID?

Porque un actor puede estar en varias películas, es necesario saber en qué película está participando.

d) ¿Por qué el orden de creación de las tablas debe ser: ¿ACTOR o PELICULA, y luego REPARTO?

Porque reparto utiliza los ids de los actores y las películas, por lo tanto, deben crearse primero en sus respectivas tablas.

e) ¿Por qué el orden de borrado de las tablas debe ser: ¿REPARTO, y luego ACTOR o PELICULA?

El orden de borrado debe ser al revés de la creación ya que no te permite borrar las tablas actor o película si hay otra tabla que está utilizando sus campos. Esto se produce por la *Integridad Referencial* (La integridad referencial es un tipo de restricción de integridad en SQL que se utiliza para garantizar la consistencia de los datos con relación a las tablas relacionadas. Esta restricción se aplica a las claves foráneas en una tabla y se utiliza para asegurar que los valores en esas claves corresponden a valores existentes en la tabla relacionada (clave primaria).).

**2)** Se tiene la siguiente BD (cuyo Script puedes encontrar en Moodle):

```
create table sucursal
   (nombre_sucursal  varchar(15) not null unique,
    ciudad_sucursal  varchar(15) not null,
    capital   numeric  not null,
    primary key(nombre_sucursal));

create table cuenta
   (numero_cuenta  varchar(15),-- not null unique,
    nombre_sucursal  varchar(15) not null,
    saldo   numeric  not null,
    primary key(numero_cuenta),
    foreign key(nombre_sucursal) references
    sucursal);


create table cliente
   (id              varchar(15),
    nombre_cliente  varchar(15) not null,-- unique,
    calle_cliente  varchar(12) not null,
    ciudad_cliente  varchar(15) not null,
    primary key(id));
```

```
create table prestamo
   (numero_prestamo  varchar(15) not null unique,
    nombre_sucursal  varchar(15) not null
    references sucursal,
    cantidad   numeric  not null,
    primary key(numero_prestamo),
    foreign key(nombre_sucursal) references
    sucursal);

create table cliente_cuenta
   (id_cliente  varchar(15) not null,
    numero_cuenta  varchar(15) not null,
    primary key(id_cliente, numero_cuenta),
    foreign key(numero_cuenta) references
    cuenta(numero_cuenta),
    foreign key(id_cliente) references
    cliente(id));

create table cliente_prestamo
   (id_cliente  varchar(15) not null,
    numero_prestamo  varchar(15) not null,
    primary key(id_cliente, numero_prestamo),
    foreign key(id_cliente) references cliente(id),
    foreign key(numero_prestamo) references
    prestamo(numero_prestamo));
```

a) Crea una consulta que muestre los nombres de los clientes que tienen un préstamo en la sucursal Perryridge, pero que no tengan una cuenta en dicha sucursal.

```
select c.id,nombre_cliente  from cliente c,
      (select id_cliente  from cliente_prestamo cp join prestamo p on cp.numero_prestamo =p.numero_prestamo
      where p.nombre_sucursal ='Perryridge') as t
      where c.id = t.id_cliente and not c.id in (select id_cliente  from cliente_cuenta cc join cuenta cu on cc.numero_cuenta =cu.numero_cuenta
      where cu.nombre_sucursal ='Perryridge')
```

b) Crea una consulta que muestre, en franjas de 1000 en 1000, el número de préstamos cuyo monto (cantidad) esté en cada una de las franjas.

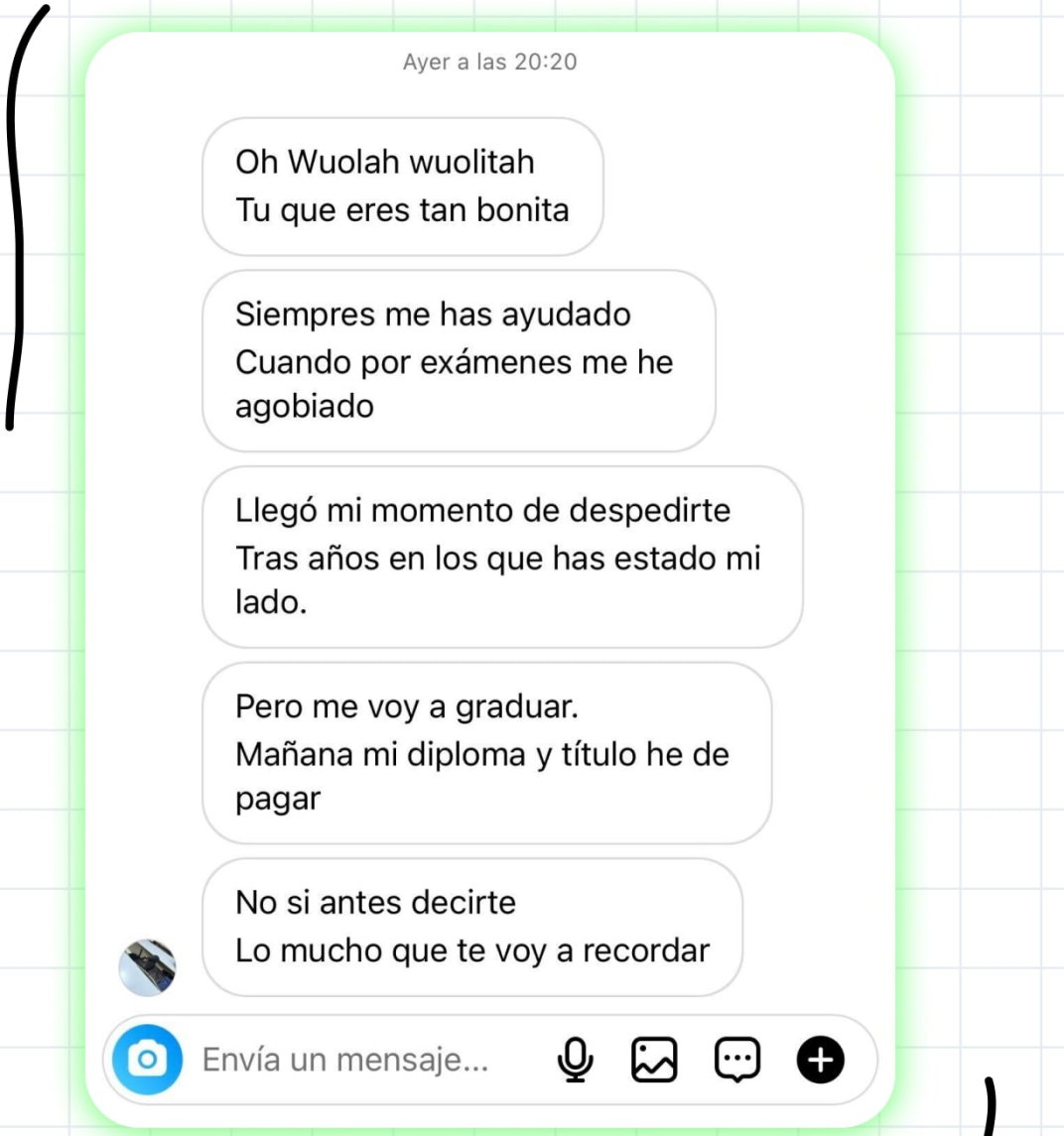
```
select floor (cantidad/1000)*1000 as rango, count(*) as n_prestamos
  from prestamo p
       group by rango
       order by rango
```

c) Crea un procedimiento almacenado, llamado *busqueda*, que dado el nombre de una ciudad muestre los clientes que tienen cuenta en alguna sucursal de dicha ciudad, pero que vivan (los clientes) en otra ciudad distinta.

```
create or replace function busqueda(pnombre_ciudad VARCHAR) returns table (nombre_cliente varchar) as $$

declare

begin

   return query

    select c2.nombre_cliente  from sucursal s,cuenta c,cliente_cuenta cc,cliente c2  where s.nombre_sucursal=c.nombre_sucursal
       and cc.numero_cuenta= c.numero_cuenta and c2.id =cc.id_cliente and s.ciudad_sucursal=pnombre_ciudad and c2.ciudad_cliente
       <> pnombre_ciudad;

end;
$$ language plpgsql ;

--PRUEBA DE LA FUNCIÓN
select * from busqueda('Brooklyn')
```

**Que no te escriban poemas de amor cuando terminen la carrera** 🚩🚩🚩🚩🚩🚩

(a nosotros por suerte nos pasa)




WUOLAH

**3)** Utilizando la BD de películas ya presentada con anterioridad, realiza los siguientes ejercicios

a) Crear una tabla auxiliar que contenga los 10 actores que más papeles como protagonista (valor ORD=1 en la tabla REPARTO) tengan entre los 80's y 90's definiendo para ello una vista que implemente el comportamiento y un trigger que se encargue de mantener la tabla actualizada.

```
--SOLUCION
create or replace view top_actores as select a.nombre , count(*) as n_protagonismo from actor a,reparto r ,pelicula p
where a.id =r.actor_id  and r.pelicula_id =p.id and r.ord = 1 and p.agno between 1980 and 1999
group by a.nombre
order by n_protagonismo desc
limit 10
```

```
drop table top_actores_protagonista;

CREATE TABLE top_actores_protagonista
                (nombre_actor  varchar,
                num_protagonismo INTEGER
                );


create or replace function updateTopActores() returns trigger as $$
begin
    truncate table  top_actores_protagonista;
    insert into top_actores_protagonista select a.nombre , count(*) as n_protagonismo from actor a,reparto r ,pelicula p
                                            where a.id =r.actor_id  and r.pelicula_id =p.id and r.ord = 1 and p.agno
                                            between 1980 and 1999
                                            group by a.nombre
                                            order by n_protagonismo desc
                                            limit 10;
    return null;
end;
$$ language plpgsql;


create trigger updTopActores
    after insert on reparto
        for each row
            execute procedure updateTopActores()
```

b) Obtener un listado de los actores (nombres) que tienen más papeles como protagonista que como secundario, para seguidamente proporcionar para estos mismos actores el año en el que tienen más papeles como protagonista. Utiliza para ello un procedimiento almacenado.

**4)** Estudia las **diapositivas** denominadas Explain Plan en PostgreSQL que podrás encontrar en Moodle, intentando ejecutar los ejemplos que se muestran en las diapositivas hasta donde sea posible, y observando los resultados obtenidos.

**5)** Partiendo de la BD sobre películas ya presentada, diseña una consulta que permita obtener, ordenados alfabéticamente, todos los actores que no han actuado en ninguna película con Tom Hanks, pero que sí lo han hecho con actores que han sido sus compañeros de reparto. Diseña distintas soluciones y compara su rendimiento.

```
SELECT * FROM actor WHERE
NOT EXISTS
    (SELECT NULL FROM
        (SELECT pelicula_id FROM reparto WHERE reparto.actor_id = actor.id) peliId
      NATURAL JOIN
        (SELECT pelicula_id FROM reparto WHERE actor_id = (SELECT id FROM actor WHERE nombre = 'Tom Hanks'))
    peliTH)
AND EXISTS
    (SELECT NULL FROM
        (SELECT pelicula_id FROM reparto WHERE reparto.actor_id = actor.id) peliId
      NATURAL JOIN
        (SELECT pelicula_id FROM
            (SELECT id FROM actor WHERE EXISTS
                (SELECT NULL FROM
                    (SELECT pelicula_id FROM reparto WHERE reparto.actor_id = actor.id) peliId NATURAL JOIN
                    (SELECT pelicula_id FROM reparto WHERE actor_id =
                        (SELECT id FROM actor WHERE nombre = 'Tom Hanks')) peliTH)) compisTH,
            reparto WHERE compisTH.id = reparto.actor_id) pelisCompisTH);
```

**6)** Utilizando MongoDB, realiza los siguientes ejercicios:

a) Crea una colección de documentos de universidades públicas de la ciudad de Madrid, un documento para cada una de las universidades con los siguientes datos: nombre, dirección postal, teléfono, e-mail de contacto y página web. Se incluirá también el nombre y número total de estudiantes de los centros de cada una de las universidades. Los datos (centros, número de estudiantes, etc.) se pueden obtener de internet.

```
db.universidades.insertMany([
{"nombre": "Universidad Autónoma de Madrid","direccion_postal" : "Ciudad Universitaria de Cantoblanco, 28049, Madrid","telefono" : "914975000","email" : "informacion.general@uam.es","web" : "https://www.uam.es","nalumnos" : 30465 ,"centros" : [{“nombre”:"Derecho",”n_estudiantes”:1000},{“nombre”:"Ciencias",”n_estudiantes”:2500}
]},

{"nombre": "Universidad Complutense de Madrid","direccion_postal" : "Avda. de Séneca, 2, Ciudad Universitaria, 28040, Madrid","telefono" : "914520400","email" : "infocom@ucm.es","web" : "https://www.ucm.es","nalumnos" : 71 806 ,"centros" : [{“nombre”:"Derecho",”n_estudiantes”:3000},{“nombre”:"Ciencias",”n_estudia
```

ntes”:500},{“nombre”:"Filosofía",”n_estudiantes”:4000},{“nombre”:"Deporte",
”n_estudiantes”:2500}]},

{"nombre": "Universidad Rey Juan Carlos","direccion_postal" : "Av. del Alcalde de Móstoles, 28933 Móstoles, Madrid","telefono" : "914520400","email" : "info@urjc.es","web" : "https://www.urjc.es","nalumnos" : 37.939 ,"centros" : [{“nombre”:"Derecho",”n_estudiantes”:1500},{“nombre”:"Ciencias",”n_estudiantes”:3500}]}

])

b) Definir 3 consultas diferentes que permitan:
   - Mostrar todos los centros de la UAM.

db.universidades.find({“nombre”:"Universidad Complutense de Madrid"},{_id:0, centros:1})

   - Mostrar el total de estudiantes de cada una de las universidades.

db.universidades.aggregate([{$unwind: "$centros"}, {$group: {_id: "$nombre", estudiantes: { $sum: "$centros.estudiantes" }}}])

   - Mostrar sólo aquellas universidades que tengan facultades de Derecho.
     db.
     universidades.find({“centros.nombre”:{$in:["Derecho"]}},{nombre:1,_id:0})

**7)** Utilizando el lenguaje Cypher, crea una consulta que permita obtener el listado de actores que no actuaron con ‘Tom Hanks’ pero que sí actuaron con otros actores que actuaron con él.

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]
-(coActors),(coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cocoActors)
WHERE NOT (tom)-[:ACTED_IN]->()<-[:ACTED_IN]-(cocoActors) AND
   tom <> cocoActors
RETURN cocoActors.name
```

**8)** Se desea tener una BD NoSQL que recoja, en forma de grafo, la información de asignaturas, profesores y estudiantes de la EPS, así como de algunas relaciones específicas entre estas entidades de información. Implementa, utilizando el lenguaje Cypher, dicha BD y las siguientes consultas sobre la misma:

- Profesores cuyo despacho está en el edificio B.
- Asignaturas de "sistemas" que se imparten en el primer semestre.
- Nombres de los estudiantes cuyo NIA va de 1111 a 1114.
- Estudiantes que cursan asignaturas cuyas clases se imparten en el edificio A.
- Estudiantes que cursan, o profesores que imparten, Sistemas Informáticos I.
- Estudiantes a los que les imparte clase el profesor José A. Macías.