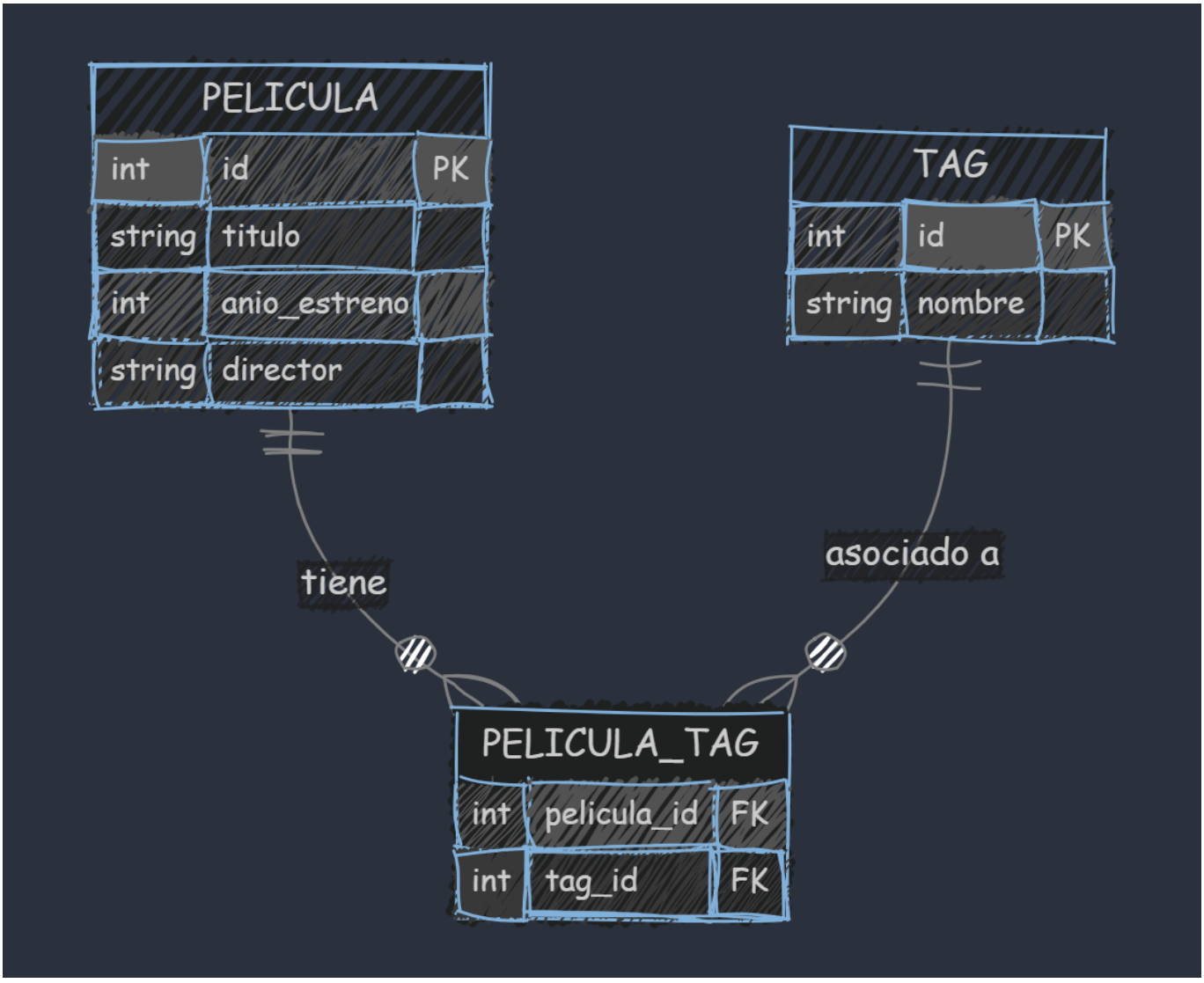
Prueba de Fundamentos de Bases de Datos Relacionales

Paso 1: Crear el modelo de Películas y Tags

Primero, se crean las tablas necesarias para el modelo de "Películas" y "Tags". El diagrama de relaciones será N:N, lo que significa que una película puede tener muchos tags, y un tag puede estar asociado con muchas películas.

Diagrama:



Creación de tablas en SQL:

```sql
CREATE TABLE pelicula (
    id SERIAL PRIMARY KEY,
    titulo VARCHAR(255),
    anio_estreno INT,
    director VARCHAR(255)
);

CREATE TABLE tag (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100)
);

CREATE TABLE pelicula_tag (
    pelicula_id INT REFERENCES pelicula(id),
    tag_id INT REFERENCES tag(id),
    PRIMARY KEY (pelicula_id, tag_id)
);
```

```
postgres=# \c peliculas_y_tags
You are now connected to database "peliculas_y_tags" as user "postgres".
peliculas_y_tags=# CREATE TABLE pelicula (
peliculas_y_tags(#     id SERIAL PRIMARY KEY,
peliculas_y_tags(#     titulo VARCHAR(255),
peliculas_y_tags(#     anio_estreno INT,
peliculas_y_tags(#     director VARCHAR(255)
peliculas_y_tags(# );
CREATE TABLE
peliculas_y_tags=# CREATE TABLE tag (
peliculas_y_tags(#     id SERIAL PRIMARY KEY,
peliculas_y_tags(#     nombre VARCHAR(100)
peliculas_y_tags(# );
CREATE TABLE
peliculas_y_tags=# CREATE TABLE pelicula_tag (
peliculas_y_tags(#     pelicula_id INT REFERENCES pelicula(id),
peliculas_y_tags(#     tag_id INT REFERENCES tag(id),
peliculas_y_tags(#     PRIMARY KEY (pelicula_id, tag_id)
peliculas_y_tags(# );
CREATE TABLE
peliculas_y_tags=# \dt
            List of relations
 Schema |     Name     | Type  |  Owner
--------+--------------+-------+----------
 public | pelicula     | table | postgres
 public | pelicula_tag | table | postgres
 public | tag          | table | postgres
(3 rows)
```

Paso 2: Insertar películas y tags

Ahora insertaremos 5 películas y 5 tags. La primera película tendrá 3 tags asociados, y la segunda tendrá 2 tags.

Inserción de datos en SQL:

```sql
-- Insertar películas
INSERT INTO pelicula (titulo, anio_estreno, director) VALUES
('Forest Gump', 1994, 'Robert Zemeckis'),
('Titanic', 1997, 'James Cameron'),
('El Padrino', 1972, 'Francis Ford Coppola'),
('Gladiator', 2000, 'Ridley Scott'),
('El Señor de los Anillos: El Retorno del Rey', 2003, 'Peter Jackson');

-- Insertar tags
INSERT INTO tag (nombre) VALUES
('Drama'),
('Acción'),
('Aventura'),
('Histórico'),
('Fantasía');

-- Asociar tags a películas
INSERT INTO pelicula_tag (pelicula_id, tag_id) VALUES
(1, 1), (1, 2), (1, 3),  -- Forest Gump tiene 3 tags
(2, 1), (2, 4);          -- Titanic tiene 2 tags
```

```
peliculas_y_tags=# INSERT INTO pelicula (titulo, anio_estreno, director) VALUES
peliculas_y_tags-# ('Forest Gump', 1994, 'Robert Zemeckis'),
peliculas_y_tags-# ('Titanic', 1997, 'James Cameron'),
peliculas_y_tags-# ('El Padrino', 1972, 'Francis Ford Coppola'),
peliculas_y_tags-# ('Gladiator', 2000, 'Ridley Scott'),
peliculas_y_tags-# ('El Señor de los Anillos: El Retorno del Rey', 2003, 'Peter Jackson');
INSERT 0 5
peliculas_y_tags=# SELECT * FROM pelicula;
 id |                  titulo                  | anio_estreno |       director
----+------------------------------------------+--------------+----------------------
  1 | Forest Gump                              |         1994 | Robert Zemeckis
  2 | Titanic                                  |         1997 | James Cameron
  3 | El Padrino                               |         1972 | Francis Ford Coppola
  4 | Gladiator                                |         2000 | Ridley Scott
  5 | El Señor de los Anillos: El Retorno del Rey |         2003 | Peter Jackson
(5 rows)
```

```
peliculas_y_tags=#
peliculas_y_tags=# INSERT INTO tag (nombre) VALUES
peliculas_y_tags-# ('Drama'),
peliculas_y_tags-# ('Acción'),
peliculas_y_tags-# ('Aventura'),
peliculas_y_tags-# ('Histórico'),
peliculas_y_tags-# ('Fantasía');
INSERT 0 5
peliculas_y_tags=# SELECT * FROM tag;
 id |  nombre
----+-----------
  1 | Drama
  2 | Acción
  3 | Aventura
  4 | Histórico
  5 | Fantasía
(5 rows)
```

```
peliculas_y_tags=# INSERT INTO pelicula_tag (pelicula_id, tag_id) VALUES
peliculas_y_tags-# (1, 1), (1, 2), (1, 3),  -- Forest Gump tiene 3 tags
peliculas_y_tags-# (2, 1), (2, 4);          -- Titanic tiene 2 tags
INSERT 0 5
peliculas_y_tags=# SELECT * FROM pelicula_tag;
 pelicula_id | tag_id
-------------+--------
           1 |      1
           1 |      2
           1 |      3
           2 |      1
           2 |      4
(5 rows)
```

Paso 3: Contar la cantidad de tags por película

Vamos a contar cuántos tags tiene cada película, incluyendo aquellas que no tienen ninguno.

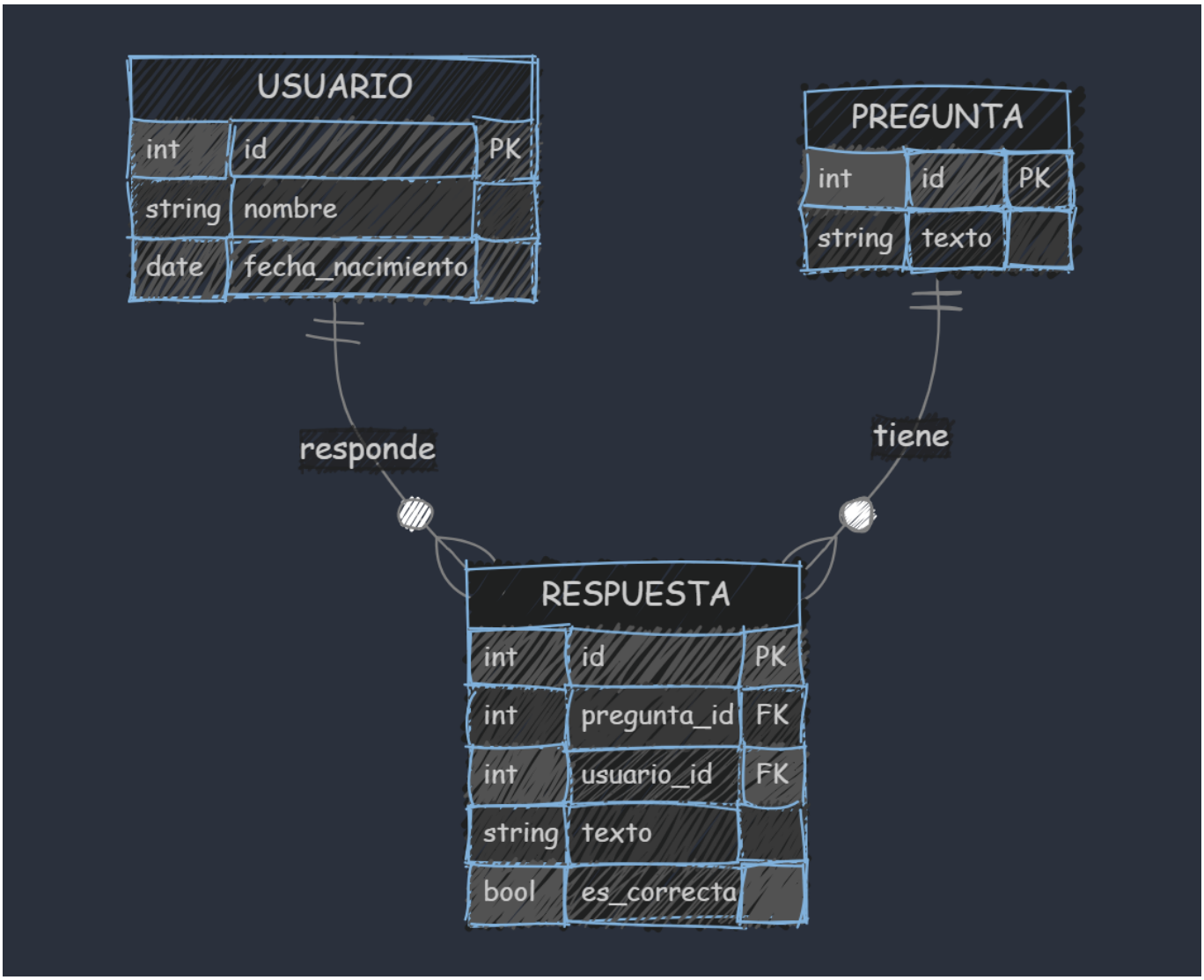
Consulta en SQL:

```sql
SELECT p.titulo, COUNT(pt.tag_id) AS num_tags
FROM pelicula p
LEFT JOIN pelicula_tag pt ON p.id = pt.pelicula_id
GROUP BY p.titulo;
```

```
peliculas_y_tags=# SELECT p.titulo, COUNT(pt.tag_id) AS num_tags
peliculas_y_tags-# FROM pelicula p
peliculas_y_tags-# LEFT JOIN pelicula_tag pt ON p.id = pt.pelicula_id
peliculas_y_tags-# GROUP BY p.titulo;
                titulo                 | num_tags
---------------------------------------+----------
 El Padrino                            |        0
 Titanic                               |        2
 Forest Gump                           |        3
 El Señor de los Anillos: El Retorno del Rey |        0
 Gladiator                             |        0
(5 rows)
```

Paso 4: Crear el modelo de Preguntas, Respuestas y Usuarios

Diagrama:



Creación de tablas en SQL:

```
CREATE TABLE usuario (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100),
    fecha_nacimiento DATE
);

CREATE TABLE pregunta (
    id SERIAL PRIMARY KEY,
    texto VARCHAR(255)
);

CREATE TABLE respuesta (
    id SERIAL PRIMARY KEY,
    pregunta_id INT REFERENCES pregunta(id),
    usuario_id INT REFERENCES usuario(id),
    texto VARCHAR(255),
```

```
    es_correcta BOOLEAN
);
```

```
examen_db=# CREATE TABLE usuario (
examen_db(#     id SERIAL PRIMARY KEY,
examen_db(#     nombre VARCHAR(100),
examen_db(#     fecha_nacimiento DATE
examen_db(#     fecha_nacimiento DATE
examen_db(# );
CREATE TABLE
examen_db=# CREATE TABLE pregunta (
examen_db(#     id SERIAL PRIMARY KEY,
examen_db(#     texto VARCHAR(255)
examen_db(# );
CREATE TABLE
examen_db=#
examen_db=# CREATE TABLE respuesta (
examen_db(#     id SERIAL PRIMARY KEY,
examen_db(#     pregunta_id INT REFERENCES pregunta(id),
examen_db(#     usuario_id INT REFERENCES usuario(id),
examen_db(#     texto VARCHAR(255),
examen_db(#     es_correcta BOOLEAN
examen_db(# );
CREATE TABLE
examen_db=# \dt
           List of relations
 Schema |   Name    | Type  |  Owner
--------+-----------+-------+----------
 public | pregunta  | table | postgres
 public | respuesta | table | postgres
 public | usuario   | table | postgres
(3 rows)
```

Paso 5: Insertar registros en Preguntas y Respuestas

Inserción de datos en SQL:

```
-- Insertar usuarios
INSERT INTO usuario (nombre, fecha_nacimiento) VALUES
('Usuario1', '1990-01-01'),
('Usuario2', '1985-05-15');

-- Insertar preguntas
INSERT INTO pregunta (texto) VALUES
('¿Cuál es la capital de Francia?'),
('¿Qué es 2 + 2?');

-- Insertar respuestas
INSERT INTO respuesta (pregunta_id, usuario_id, texto, es_correcta) VALUES
(1, 1, 'París', TRUE),
```

```
  (1, 2, 'París', TRUE),
  (2, 1, '4', TRUE),
  (2, 2, '5', FALSE);
```

```
examen_db=# INSERT INTO usuario (nombre, fecha_nacimiento) VALUES
examen_db-# ('Usuario1', '1990-01-01'),
examen_db-# ('Usuario2', '1985-05-15');
INSERT 0 2
examen_db=# INSERT INTO pregunta (texto) VALUES
examen_db-# ('¿Cuál es la capital de Francia?'),
examen_db-# ('¿Qué es 2 + 2?');
INSERT 0 2
examen_db=# INSERT INTO respuesta (pregunta_id, usuario_id, texto, es_correcta) VALUES
examen_db-# (1, 1, 'París', TRUE),
examen_db-# (1, 2, 'París', TRUE),
examen_db-# (2, 1, '4', TRUE),
examen_db-# (2, 2, '5', FALSE);
INSERT 0 4
```

Paso 6: Contar respuestas correctas por usuario

Consulta en SQL:

```
  SELECT u.nombre, COUNT(r.id) AS num_respuestas_correctas
  FROM usuario u
  JOIN respuesta r ON u.id = r.usuario_id
  WHERE r.es_correcta = TRUE
  GROUP BY u.nombre;
```

```
examen_db=# SELECT u.nombre, COUNT(r.id) AS num_respuestas_correctas
examen_db-# FROM usuario u
examen_db-# JOIN respuesta r ON u.id = r.usuario_id
examen_db-# WHERE r.es_correcta = TRUE
examen_db-# GROUP BY u.nombre;
  nombre  | num_respuestas_correctas
----------+-------------------------
 Usuario1 |                       2
 Usuario2 |                       1
(2 rows)
```

Paso 7: Contar usuarios con respuestas correctas por pregunta

Consulta en SQL:

```sql
SELECT p.texto, COUNT(r.usuario_id) AS num_usuarios_correctos
FROM pregunta p
JOIN respuesta r ON p.id = r.pregunta_id
WHERE r.es_correcta = TRUE
GROUP BY p.texto;
```

```
examen_db=# SELECT p.texto, COUNT(r.usuario_id) AS num_usuarios_correctos
examen_db-# FROM pregunta p
examen_db-# JOIN respuesta r ON p.id = r.pregunta_id
examen_db-# WHERE r.es_correcta = TRUE
examen_db-# GROUP BY p.texto;
             texto              | num_usuarios_correctos
--------------------------------+------------------------
 ¿Cuál es la capital de Francia? |                      2
 ¿Qué es 2 + 2?                 |                      1
(2 rows)
```

Paso 8: Implementar borrado en cascada

SQL para borrado en cascada:

```sql
ALTER TABLE respuesta
DROP CONSTRAINT respuesta_usuario_id_fkey,
ADD CONSTRAINT respuesta_usuario_id_fkey FOREIGN KEY (usuario_id)
REFERENCES usuario(id) ON DELETE CASCADE;

-- Probar borrado en cascada
DELETE FROM usuario WHERE id = 1;
```

```
examen_db=# DELETE FROM usuario WHERE id = 1;
DELETE 1
examen_db=# SELECT * FROM usuario;
 id |  nombre  | fecha_nacimiento
----+----------+------------------
  2 | Usuario2 | 1985-05-15
(1 row)
```

Paso 9: Crear restricción para impedir usuarios menores de 18 años

SQL para restricción:

```
ALTER TABLE usuario
ADD CONSTRAINT chk_mayor_de_edad CHECK (fecha_nacimiento <= CURRENT_DATE -
INTERVAL '18 years');
```

```
examen_db-# ADD CONSTRAINT chk_mayor_de_edad CHECK (fecha_nacimiento <= CURRENT_DATE - INTERVAL '18 years');
ALTER TABLE
examen_db=# \d usuario
                                        Table "public.usuario"
      Column      |          Type          | Collation | Nullable |               Default
------------------+------------------------+-----------+----------+-------------------------------------
 id               | integer                |           | not null | nextval('usuario_id_seq'::regclass)
 nombre           | character varying(100) |           |          |
 fecha_nacimiento | date                   |           |          |
Indexes:
    "usuario_pkey" PRIMARY KEY, btree (id)
Check constraints:
    "chk_mayor_de_edad" CHECK (fecha_nacimiento <= (CURRENT_DATE - '18 years'::interval))
Referenced by:
    TABLE "respuesta" CONSTRAINT "respuesta_usuario_id_fkey" FOREIGN KEY (usuario_id) REFERENCES usuario(id)
```

Paso 10: Alterar tabla usuarios para agregar email con restricción única

SQL para agregar campo:

```
ALTER TABLE usuario
ADD COLUMN email VARCHAR(255) UNIQUE;
```

```
examen_db=# ALTER TABLE usuario
examen_db-# ADD COLUMN email VARCHAR(255) UNIQUE;
ALTER TABLE
examen_db=# \d usuario
examen_db=# \d usuario
                                        Table "public.usuario"
      Column      |          Type          | Collation | Nullable |               Default
------------------+------------------------+-----------+----------+-------------------------------------
 id               | integer                |           | not null | nextval('usuario_id_seq'::regclass)
 nombre           | character varying(100) |           |          |
 fecha_nacimiento | date                   |           |          |
 email            | character varying(255) |           |          |
Indexes:
    "usuario_pkey" PRIMARY KEY, btree (id)
    "usuario_email_key" UNIQUE CONSTRAINT, btree (email)
Check constraints:
    "chk_mayor_de_edad" CHECK (fecha_nacimiento <= (CURRENT_DATE - '18 years'::interval))
Referenced by:
    TABLE "respuesta" CONSTRAINT "respuesta_usuario_id_fkey" FOREIGN KEY (usuario_id) REFERENCES usuario(id) ON DELETE CASCADE
```