

REPASO QUERIES

Víctor Manuel Peiró Martínez
FIIS 3A Ampliación de Bases de Datos

Contenido

Queries Mongo 2

 Normales..... 2

 Agregadas 7

Queries Redis..... 20

Queries Neo4j 25

Queries Mongo

Normales

1. **Insertar un nuevo usuario:** Insertar usuario en la colección users

```
exercises> db.users.insertOne({
...   firstName: "Juan Felipe",
...   lastName: "Rodriguez Cordoba",
...   age: 20,
...   email: "juan.rodriguez@example.com"
... })
{
  acknowledged: true,
  insertedId: ObjectId('695cee190dea41a0499dc29d')
```

2. **Insertar varios documentos:** Insertar varios documentos de productos en una colección de products.

```
exercises> db.products.insertMany([
...   {name: "p1", description: "product 1", price: 20, category: "Test Products"},
...   {name: "p2", description: "product 2", price: 10, category: "Test Products"}
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('695cef550dea41a0499dc29e'),
    '1': ObjectId('695cef550dea41a0499dc29f')
  }
}
```

3. **Buscar un documento por ID:** Recupere un documento de usuario por su _id de la colección de users

```
exercises> db.users.find({_id: 7})
[
  {
    _id: 7,
    firstName: 'Eve',
    lastName: 'Miller',
    email: 'eve.miller@example.com',
    age: 27,
    username: 'evem',
    lastLogin: '2024-09-05'
  }
]
```

4. **Actualizar un documento:** actualice la dirección de correo electrónico de un usuario en la colección de users.

```
exercises> db.users.updateOne({name: "Juan Felipe"}, {$set: {email: "juanfelipe.rodriguezcordoba@example.com", username: "juanfeliperc"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
```

5. **Eliminar un documento:** elimina un producto de la colección de products por su `_id`.

```
exercises> db.products.deleteOne({_id:ObjectId('695cef550dea41a0499dc29f')})
{ acknowledged: true, deletedCount: 1 }
```

6. **Buscar documentos con una condición:** recupere todos los usuarios mayores de 30 años de la colección de users.

```
exercises> db.users.find({age: {$gt: 30}})
[
  {
    _id: 5,
    firstName: 'Charlie',
    lastName: 'Davis',
    email: 'charliedavis@example.com',
    age: 31,
    username: 'charlied',
    lastLogin: '2024-07-22'
  },
]
```

7. **Actualizar varios documentos:** Aumente el precio de todos los productos de la colección de products en un 10%

```
exercises> db.products.updateMany({}, {$mul: {price: 1.1}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 12,
  modifiedCount: 12,
  upsertedCount: 0
}
```

8. **Eliminar varios documentos:** elimine todos los usuarios que no hayan iniciado sesión durante más de seis meses de la colección de users

```
exercises> db.users.deleteMany({lastLogin: {$lte : '2024-03-22'}})
{ acknowledged: true, deletedCount: 4 }
```

9. **Operación Upsert:** Inserte un nuevo usuario si no existe, o actualice la información del usuario existente.

```
exercises> db.users.updateOne(
... {firstName: "Victor"},
... {$set: {firstName: "Victor Manuel", lastName: "Peiro Martinez"}},
... {upsert: true}
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

10. **Expresión regular:** Encuentre places que tengan la palabra "of" en sus nombres usando una expresión regular.

```
exercises> db.places.find({name: {$regex: "\\sof\\s"}})
[
  {
    _id: 4,
    name: 'Statue of Liberty',
    location: { type: 'Point', coordinates: [ -74.0445, 40.6892 ] },
    rating: 4.6
  },
  {
    _id: 6,
    name: 'Great Wall of China',
    location: { type: 'Point', coordinates: [ 116.5704, 40.4319 ] },
    rating: 4.8
  }
]
```

11. **Índice compuesto:** cree un índice compuesto en firstName y lastName en la colección de users.

```
exercises> db.users.createIndex({firstName: "text", lastName: "text"})
```

12. **Índice de texto:** cree un índice de texto en el campo de description de la colección de products para la búsqueda de texto completo.

```
exercises> db.products.createIndex({description: "text"})
description_text
```

13. **Índice único:** asegúrese de que el campo de nombre de usuario de la colección de users sea único.

```
exercises> db.users.createIndex({username: "text"}, {unique: true})
username_text
exercises> db.users.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  {
    v: 2,
    key: { firstName: 1, lastName: 1 },
    name: 'firstName_1_lastName_1'
  },
  {
    v: 2,
    key: { _fts: 'text', _ftsx: 1 },
    name: 'username_text',
    unique: true,
    weights: { username: 1 },
    default_language: 'english',
    language_override: 'language',
    textIndexVersion: 3
  }
]
```

14. **Buscar texto:** Usando el índice de texto, busque los products con la palabra "laptop" en su descripción

```
exercises> db.products.find({description: {$regex: "laptop"}})
[
  {
    _id: 1,
    name: 'Laptop',
    description: 'A high-performance laptop',
    price: 1452.0000000000002,
    category: 'Electronics'
  }
]
```

15. **Índice geoespacial:** cree un índice 2dsphere en el campo de location de la colección places.

```
exercises> db.places.createIndex({location:"2dsphere"})
location_2dsphere
exercises> db.places.getIndex
db.places.getIndexes      db.places.getIndexSpecs  db.places.getIndexKeys

exercises> db.places.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  {
    v: 2,
    key: { location: '2dsphere' },
    name: 'location_2dsphere',
    '2dsphereIndexVersion': 3
  }
]
```

16. **Búsqueda indexada:** Recupera todos los productos con precios superiores a 1000 y con la palabra "laptop" en la descripción.

```
exercises> db.products.find({$and : [
... {description: {$regex: "laptop"}}},
... {price: {$gte: 1000}}
... ]})
[
  {
    _id: 1,
    name: 'Laptop',
    description: 'A high-performance laptop',
    price: 1452.0000000000002,
    category: 'Electronics'
  }
]
```

17. **Buscar documentos cerca de un punto:** recupere todos los places dentro de los 5 kilómetros de un punto determinado.

```
exercises> db.places.find(
... {location: {$near: {$geometry: {type: "Point", coordinates: [-74.0, 40.71]},
... $maxDistance:5000}}}
... )
[
  {
    _id: 4,
    name: 'Statue of Liberty',
    location: { type: 'Point', coordinates: [ -74.0445, 40.6892 ] },
    rating: 4.6
  }
]
```

- 18. Clasificación geoespacial:** ordena los places por su distancia a un punto determinado.

```
exercises> db.places.find(
... {location: {$near: {$geometry: {$type: "Point", coordinates: [0,0]},
... $minDistance: 0}}
... }).sort()
[
  {
    _id: 5,
    name: 'Colosseum',
    location: { type: 'Point', coordinates: [ 12.4922, 41.8902 ] },
    rating: 4.7
  },
  {
```

- 19. Buscar documentos usando \$or :** Busque places con la palabra "the" o "of" en sus nombres.

```
exercises> db.places.find(
... {$or: [
... {name: {$regex: "of"}},
... {name: {$regex: "the"}}
... ]})
[
  {
    _id: 4,
    name: 'Statue of Liberty',
    location: { type: 'Point', coordinates: [ -74.0445, 40.6892 ] },
    rating: 4.6
  },
  {
    _id: 6,
    name: 'Great Wall of China',
    location: { type: 'Point', coordinates: [ 116.5704, 40.4319 ] },
    rating: 4.8
  },
  {
    _id: 10,
    name: 'Christ the Redeemer',
    location: { type: 'Point', coordinates: [ -43.2105, -22.9519 ] },
    rating: 4.7
  }
]
```

- 20. Buscar documentos mediante \$in :** Buscar products de tipo “Electronics” y “Furniture”.

```
exercises> db.products.find({category: {$in: ["Electronics","Furniture"]}})
[
  {
    _id: 1,
    name: 'Laptop',
    description: 'A high-performance laptop',
    price: 1452.0000000000002,
    category: 'Electronics'
  },
  {
    _id: 2,
    name: 'Smartphone',
    description: 'A latest model smartphone',
    price: 968.0000000000002,
    category: 'Electronics'
  },
  {
```

- 21. Buscar documentos por valor de matriz:** busque places con ambas coordenadas menores que 0.

```
exercises> db.places.find({$and: [{location.coordinates.0:{$lt:0}}, {location.coordinates.1:{$lt:0}}]})
[
  {
    _id: 9,
    name: 'Machu Picchu',
    location: { type: 'Point', coordinates: [ -72.545, -13.1631 ] },
    rating: 4.8
  },
  {
    _id: 10,
    name: 'Christ the Redeemer',
    location: { type: 'Point', coordinates: [ -43.2105, -22.9519 ] },
    rating: 4.7
  }
]
```

Agregadas

1. **Edad media de los usuarios:** Calcula la edad media de todos los usuarios.

```
exercises> db.users.aggregate([
... {$group:
... {_id: null, avg_age: {$avg: "$age"}}
... })
... ])
[ { _id: null, avg_age: 28.88888888888889 } ]
```

2. **Los 5 usuarios más antiguos:** Encuentre los 5 usuarios más antiguos

```
exercises> db.users.aggregate([
... {$sort: {age: -1}},
... {$limit: 5}
... ])
[
  {
    _id: 10,
    firstName: 'Hank',
    lastName: 'Anderson',
    email: 'hank.anderson@example.com',
    age: 36,
    username: 'hanka',
    lastLogin: '2024-03-25'
  },
```

3. **Usuarios con correos electrónicos de dominio específico:** cuente el número de usuarios con direcciones de correo electrónico de un dominio específico (por ejemplo, example.com).

```
exercises> db.users.aggregate([
... {$match: {email: {$regex: "example.com$"}}},
... {$group: {_id: null, num_users: {$sum: 1}}}
... ])
[ { _id: null, num_users: 8 } ]
```

4. **Nombres de usuario que comienzan con una letra:** Encuentre todos los nombres de usuario que comiencen con una letra específica.

```
exercises> db.users.aggregate([
... {$match: {username: {$regex: "^j"}}}
... ])
[
  {
    _id: 1,
    firstName: 'John',
    lastName: 'Doe',
    email: 'john.doe@example.com',
    age: 28,
    username: 'johndoe',
    lastLogin: '2024-09-01'
  },
  {
    _id: 2,
    firstName: 'Jane',
    lastName: 'Smith',
    email: 'jane.smith@example.com',
    age: 34,
    username: 'janesmith',
    lastLogin: '2024-08-15'
  }
]
```

5. **Inicios de sesión recientes:** recupere los usuarios que han iniciado sesión en los últimos 30 días.

```
exercises> db.users.aggregate([
... {$match: {lastLogin: {$gte: "2025-12-06"}}}}
... ])
```

6. **Usuarios por último año de inicio de sesión:** agrupe a los usuarios por el año de su último inicio de sesión y cuente el número de usuarios de cada año.

```
exercises> db.users.aggregate([
... {$group: {
... _id: {year: {$year: {$toDate: "$lastLogin"}}}},
... num_users: {$sum: 1}
... }}})
[
  { _id: { year: 2024 }, num_users: 6 },
  { _id: { year: 2025 }, num_users: 2 },
  { _id: { year: null }, num_users: 2 }
]
```

7. **Usuarios con campos faltantes:** identifique a los usuarios que no tienen una dirección de correo electrónico.

```
exercises> db.users.aggregate([
... {$match: {email: null}}
... ])
[
  { _id: ObjectId('68d11a5fd63e940bafce5f47') },
  {
    _id: ObjectId('68d67b3d982b6f61fb247938'),
    name: 'Janet',
    lastName: 'Smith',
    age: 35,
    lastLogin: '2025-07-22'
  }
]
```

8. **Edad media por nombre inicial:** Calcula la edad media de los usuarios agrupados por la primera letra de su nombre.

```
exercises> db.users.aggregate([
... {$group: {
... _id: {firstLetter: {$substr: ["$firstName", 0, 1]}},
... avg_age: {$avg: "$age"}
... }}})
[
  { _id: { firstLetter: 'E' }, avg_age: 27 },
  { _id: { firstLetter: 'B' }, avg_age: 29 },
  { _id: { firstLetter: 'C' }, avg_age: 31 },
  { _id: { firstLetter: 'J' }, avg_age: 27.333333333333332 },
  { _id: { firstLetter: 'H' }, avg_age: 36 },
  { _id: { firstLetter: 'I' }, avg_age: 35 },
  { _id: { firstLetter: 'V' }, avg_age: 20 }
]
```

9. **Usuarios con varias condiciones:** busque usuarios que tengan más de 30 años y que hayan iniciado sesión en los últimos 6 meses

```
exercises> db.users.aggregate([
... {$match: {$and: [
... {age: {$gte: 30}},
... {lastLogin: {$gte: "2025-03-06"}}
... ]}}}
[
  {
    _id: ObjectId('68d67b3d982b6f61fb247938'),
    name: 'Janet',
    lastName: 'Smith',
    age: 35,
    lastLogin: '2025-07-22'
  }
]
```

10. **Precio medio por categoría:** Calcula el precio medio de los productos de cada categoría.

```
exercises> db.products.aggregate([
... {$group: {
... _id: "$category",
... avg_price: {$avg: "$price"}
... }}}]
[
  { _id: 'Furniture', avg_price: 332.75 },
  { _id: 'Test Products', avg_price: 22 },
  { _id: 'Accessories', avg_price: 423.50000000000001 },
  { _id: 'Home Appliances', avg_price: 151.25000000000003 },
  { _id: 'Sportswear', avg_price: 139.15 },
  { _id: 'Electronics', avg_price: 887.3333333333335 }
]
```

11. **Valor total del inventario:** Calcule el valor total de todos los productos.

```
exercises> db.products.aggregate([
... {$group: {
... _id: null,
... total_price: {$sum: "$price"}
... }}}]
[ { _id: null, total_price: 4492.9500000000001 } ]
```

12. **Los 3 productos más caros:** Encuentra los 3 productos más caros.

```
exercises> db.products.aggregate([
... {$sort: {price: -1}},
... {$limit: 3}
... ])
[
  {
    _id: 1,
    name: 'Laptop',
    description: 'A high-performance laptop',
    price: 1452.0000000000002,
    category: 'Electronics'
  },
  {
    _id: 2,
    name: 'Smartwatch',
    description: 'A smartwatch with health monitoring',
    price: 129.99,
    category: 'Electronics'
  },
  {
    _id: 3,
    name: 'Wireless Earbuds',
    description: 'A pair of wireless earbuds',
    price: 99.99,
    category: 'Electronics'
  }
]
```

13. **Productos con una palabra clave específica:** busque productos cuya descripción contenga una palabra clave específica.

```
exercises> db.products.aggregate([
... {$match: {description: {$regex: "phone"}}}
... ])
[
  {
    _id: 2,
    name: 'Smartphone',
    description: 'A latest model smartphone',
    price: 968.0000000000002,
    category: 'Electronics'
  },
  {
    _id: 3,
    name: 'Headphones',
    description: 'Noise-cancelling headphones',
    price: 242.00000000000006,
    category: 'Electronics'
  }
]
```

14. **Productos con una palabra clave específica:** busque productos cuya descripción contenga una palabra clave específica.

```
exercises> db.products.aggregate([
... {$match: {description: {$regex: "phone"}}}
... ])
[
  {
    _id: 2,
    name: 'Smartphone',
    description: 'A latest model smartphone',
    price: 968.0000000000002,
    category: 'Electronics'
  },
  {
    _id: 3,
    name: 'Headphones',
    description: 'Noise-cancelling headphones',
    price: 242.00000000000006,
    category: 'Electronics'
  }
]
```

15. **Recuento de productos por categoría:** cuente el número de productos de cada categoría.

```
exercises> db.products.aggregate([
... {$group: {
... _id: "$category",
... num_prods: {$sum: 1}
... }}}]
[
  { _id: 'Furniture', num_prods: 2 },
  { _id: 'Test Products', num_prods: 1 },
  { _id: 'Accessories', num_prods: 1 },
  { _id: 'Home Appliances', num_prods: 2 },
  { _id: 'Sportswear', num_prods: 3 },
  { _id: 'Electronics', num_prods: 3 }
]
```

- 16. Simulación de aumento de precios:** Calcule los nuevos precios si los precios de todos los productos se incrementaron en un 10%.

```
exercises> db.products.aggregate([
... { $project: {
... _id: 0,
... name: 1,
... new_price: { $multiply: [ "$price", 1.1 ] }
... } } ] )
[
  { name: 'Laptop', new_price: 1597.2000000000003 },
  { name: 'Smartphone', new_price: 1064.8000000000004 },
  { name: 'Watch', new_price: 465.85000000000014 },
  { name: 'Blender', new_price: 199.65000000000003 },
  { name: 'Headphones', new_price: 266.20000000000001 },
  { name: 'Coffee Maker', new_price: 133.10000000000005 },
  { name: 'Running Shoes', new_price: 159.72000000000003 },
  { name: 'Desk Chair', new_price: 332.75 },
  { name: 'Jacket', new_price: 239.58000000000007 },
  { name: 'Bookshelf', new_price: 399.30000000000007 },
  { name: 'T-Shirt', new_price: 59.89500000000002 },
  { name: 'p1', new_price: 24.200000000000003 }
]
```

- 17. Productos con varias condiciones:** busca productos que pertenezcan a una categoría específica y que tengan un precio superior a un determinado umbral.

```
exercises> db.products.aggregate([
... { $match: {
... $and: [
... { category: "Electronics" },
... { price: { $gt: 400 } }
... ] } } ] )
[
  {
    _id: 1,
    name: 'Laptop',
    description: 'A high-performance laptop',
    price: 1452.0000000000002,
    category: 'Electronics'
  },
  {
    _id: 2,
    name: 'Smartphone',
    description: 'A latest model smartphone',
    price: 968.0000000000002,
    category: 'Electronics'
  }
]
```

- 18. Calificación promedio de los lugares:** Calcule la calificación promedio de todos los lugares

```
exercises> db.places.aggregate([
... { $group: {
... _id: null,
... avg_rating: { $avg: "$rating" }
... } } ] )
[ { _id: null, avg_rating: 4.76 } ]
```

- 19. Lugares cerca de un punto:** Encuentre todos los lugares dentro de un radio de 10 kilómetros de un punto determinado.

```
exercises> db.places.aggregate([
... {$geoNear:
... {near: {type: "Point", coordinates: [-73.8650, 40.7820]},
... maxDistance: 10000,
... spherical: true,
... distanceField: "distance"}
... }])
[
  {
    _id: 1,
    name: 'Central Park',
    location: { type: 'Point', coordinates: [ -73.9654, 40.7829 ] },
    rating: 4.7,
    distance: 8463.317728147455
  }
]
```

- 20. Los 5 lugares mejor valorados:** Encuentra los 5 lugares mejor valorados

```
exercises> db.places.aggregate([ {$sort: {rating: -1}}, {$limit: 5} ])
[
  {
    _id: 3,
    name: 'Louvre Museum',
    location: { type: 'Point', coordinates: [ 2.3364, 48.8606 ] },
    rating: 4.9
  },
  {
    _id: 8,
    name: 'Taj Mahal',
    location: { type: 'Point', coordinates: [ 78.0421, 27.1751 ] },
    rating: 4.9
  },
  {
    _id: 2,
    name: 'Eiffel Tower',
    location: { type: 'Point', coordinates: [ 2.2945, 48.8584 ] },
    rating: 4.8
  },
]
```

- 21. Lugares por proximidad:** ordene los lugares por su distancia a un punto determinado.

```
exercises> db.places.aggregate([
... {$geoNear: {near: {type: "Point", coordinates: [0,0]}, spherical: true, distanceField: "distance"}}
... ])
[
  {
    _id: 5,
    name: 'Colosseum',
    location: { type: 'Point', coordinates: [ 12.4922, 41.8902 ] },
    rating: 4.7,
    distance: 4829128.852056703
  },
  {
    _id: 10,
    name: 'Christ the Redeemer',
    location: { type: 'Point', coordinates: [ -43.2105, -22.9519 ] },
    rating: 4.7,
    distance: 5326016.119249308
  },
]
```

- 22. Lugares con una palabra clave de nombre específico:** busque lugares cuyo nombre contenga una palabra clave específica.

```
exercises> db.places.aggregate([
... {$match: {name: {$regex: "of"}}}
... ])
[
  {
    _id: 4,
    name: 'Statue of Liberty',
    location: { type: 'Point', coordinates: [ -74.0445, 40.6892 ] },
    rating: 4.6
  },
  {
    _id: 6,
    name: 'Great Wall of China',
    location: { type: 'Point', coordinates: [ 116.5704, 40.4319 ] },
    rating: 4.8
  }
]
```

- 23. Lugares por recuento de calificación:** cuente el número de lugares para cada valor de calificación

```
exercises> db.places.aggregate([
... {$group: {
... _id: "$rating",
... num_places: {$sum: 1}
... }}}]
[
  { _id: 4.6, num_places: 1 },
  { _id: 4.7, num_places: 4 },
  { _id: 4.8, num_places: 3 },
  { _id: 4.9, num_places: 2 }
]
```

- 24. Lugares con múltiples condiciones:** Encuentre lugares que tengan una calificación superior a 4.8 y estén a una cierta distancia de un punto determinado.

```
exercises> db.places.aggregate([
... {$geoNear: {
... near: {type: "Point", coordinates: [2.2945,48.8584]},
... maxDistance: 10000,
... spherical: true,
... distanceField: "distance"
... }},
... {$match: {rating: {$gte: 4.8}}}
... ])
[
  {
    _id: 2,
    name: 'Eiffel Tower',
    location: { type: 'Point', coordinates: [ 2.2945, 48.8584 ] },
    rating: 4.8,
    distance: 0
  },
  {
    _id: 3,
    name: 'Louvre Museum',
    location: { type: 'Point', coordinates: [ 2.3364, 48.8606 ] },
    rating: 4.9,
    distance: 3078.409450014034
  }
]
exercises> |
```

25. Usuarios “problemáticos”: sin email O email no válido.

```
test> db.users.aggregate([
...  {$match: {$or: [
...  {email: {$exists: false}},
...  {email: {$not: {$regex: "@.+\\.+.$"}}}}
... ]}}])
```

26. “Limpieza” automática: si falta lastLogin, ponlo a NOW; si falta email, crea uno placeholder.

```
exercices> db.users.updateMany(
...  {$or: [{email: null},{email: {$exists: false}}},{lastLogin: null},{lastLogin: {$exists: false}}],
...  [{set: {email: {$ifNull: ["$email",{$concat: ["$firstName",".", "$lastName","@example.com"]}},lastLogin: {$ifNull: ["$lastLogin","$NOW"]}}}
...  ])
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
```

27. Productos con price > media global de Price.

```
exercices> db.products.aggregate([
...  {$group: {_id: null, avg_price: {$avg: "$price"}, prods: {$push: {name: "$name", price: "$price"}}}},
...  {$unwind: "$prods"},
...  {$match: {$expr: {$gt: ["$prods.price", "$avg_price"]}}}
... ])
[
  {
    _id: null,
    avg_price: 374.41250000000001,
    prods: { name: 'Laptop', price: 1452.0000000000002 }
  },
  {
    _id: null,
    avg_price: 374.41250000000001,
    prods: { name: 'Smartphone', price: 968.0000000000002 }
  },
  {
    _id: null,
    avg_price: 374.41250000000001,
    prods: { name: 'Watch', price: 423.5000000000001 }
  }
]
```

28. Subir precio 10% solo a Electronics, pero capando precio máximo a 2000.

```
db.products.updateMany( { category: "Electronics" }, [ { $set: { price: { $min: [2000, { $multiply: ["$price", 1.1] } ] } } } ] )
```

29. Productos con price > media de la cateogria de Price.

```
exercices> db.products.aggregate([
...  {$group: {_id: "$category", avg_price: {$avg: "$price"}, prods: {$push: {name: "$name", price: "$price"}}}},
...  {$unwind: "$prods"},
...  {$match: {$expr: {$gt: ["$prods.price", "$avg_price"]}}}
... ])
```

30. Devuelve los productos cuyo precio sea superior al precio del producto más barato de su categoría multiplicado por 3.

```
test> db.products.aggregate([
...  {$group: {_id: "$category", min_price: {$min: "$price"}, prods: {$push: {name: "$name", price: "$price"}}}},
...  {$unwind: "$prods"},
...  {$match: {$expr: {$gt: ["$prods.price", {$multiply: ["$min_price", 3]}}}}}
... ])
```

31. Devuelve los productos cuyo precio sea más de 500 euros superior al precio mínimo de su categoría.

```
test> db.products.aggregate([
...  {$group: {_id: "$category", min_price: {$min: "$price"}, prods: {$push: {name: "$name", price: "$price"}}}},
...  {$unwind: "$prods"},
...  {$match: {$expr: {$gt: ["$prods.price", {$add: ["$min_price", 500]}}}}}
... ])
```

32. Devuelve los usuarios que hayan iniciado sesión en los últimos 30 días.

```
exercises> db.users.aggregate([
... {$addFields: {lastLogin: {$toDate: "$lastLogin"}}},
... {$addFields: {lastLoginDays: {$dateDiff: {startDate: "$lastLogin", endDate: "$$NOW", unit: "day"}}}},
... {$match: {lastLoginDays: {$lt: 30}}}
... ])
[
  {
    _id: 3,
    firstName: 'Alice',
    lastName: 'Johnson',
    email: 'alice.johnson@example.com',
    age: 45,
    username: 'alicej',
    lastLogin: ISODate('2026-01-07T00:00:00.000Z'),
    lastLoginDays: Long('0')
  }
]
```

33. Encontrar usuarios inactivos > 6 meses

```
exercises> db.users.aggregate([
... {$addFields: {lastLogin: {$toDate: "$lastLogin"}}},
... {$addFields: {lastLoginMonths: {$dateDiff: {startDate: "$lastLogin", endDate: "$$NOW", unit: "month"}}}},
... {$match: {lastLoginMonths: {$gt: 6}}}
... ])
[
  {
    _id: 1,
    firstName: 'John',
    lastName: 'Doe',
    email: 'john.doe@example.com',
    age: 28,
    username: 'johndoe',
    lastLogin: ISODate('2024-09-01T00:00:00.000Z'),
    lastLoginMonths: Long('16')
  },
]
```

34. Agrupa usuarios por el año de su último inicio de sesión y cuenta cuántos hay en cada año.

```
exercises> db.users.aggregate([
... {$addFields: {lastLogin: {$toDate: "$lastLogin"}}},
... {$addFields: {lastLoginYear: {$year: "$lastLogin"}}},
... {$group: {
... _id: "$lastLoginYear",
... num_users: {$sum: 1}
... }}}]
[
  { _id: 2024, num_users: 8 },
  { _id: 2023, num_users: 1 },
  { _id: 2026, num_users: 1 }
]
```

35. Usuarios > 30 años y login en los últimos 6 meses.

```
exercises> db.users.aggregate([
... {$addFields: {lastLogin: {$toDate: "$lastLogin"}}},
... {$addFields: {lastLoginMonths: {$dateDiff: {startDate: "$lastLogin", endDate: "$$NOW", unit: "month"}}}},
... {$match: {$and: [{age: {$gt: 30}}, {lastLoginMonths: {$lt: 6}}]}}
... ])
[
  {
    _id: 3,
    firstName: 'Alice',
    lastName: 'Johnson',
    email: 'alice.johnson@example.com',
    age: 45,
    username: 'alicej',
    lastLogin: ISODate('2026-01-07T00:00:00.000Z'),
    lastLoginMonths: Long('0')
  }
]
```

36. Devuelve lugares a menos de 10 km de un punto y cuya distancia sea menor que el doble de su rating multiplicado por 1000.

```
exercises> db.places.aggregate([
...   {$geoNear:
...     {near: {type: "Point", coordinates: [2.29, 48.45]},
...       maxDistance: 10000,
...       spherical: true,
...       distanceField: "dist"
...     }},
...   {$match: {$expr: {$lt: ["$dist", {$multiply: [2, "$rating", 1000]}]}}}
... ])
```

37. Lugares a menos de 100 km con rating >= 4.8 y ordenada la lista de menor a mayor

```
exercises> db.places.aggregate([
...   {
...     $geoNear: {
...       near: { type: "Point", coordinates: [2.29, 48.45] },
...       maxDistance: 100000,
...       spherical: true,
...       distanceField: "dist"
...     }
...   },
...   {
...     $match: { rating: { $gte: 4.8 } }
...   },
...   { $sort: { dist: 1 } }
... ])
```

38. Devuelve los productos cuyo precio sea al menos el triple del mínimo de su categoría

```
exercises> db.products.aggregate([
...   {$group: {
...     _id: "$category",
...     minPrice: {$min: "$price"},
...     prods: {$push: {name: "$name", price: "$price"}}
...   }},
...   {$unwind: "$prods"},
...   {$match: {$expr: {$gte: ["$prods.price", {$multiply: ["$minPrice", 3]}]}}}
... ])
[
  {
    _id: 'Electronics',
    minPrice: 200,
    prods: { name: 'Smartphone', price: 800 }
  },
  {
    _id: 'Electronics',
    minPrice: 200,
    prods: { name: 'Laptop', price: 1200 }
  }
]
```

39. Devuelve los productos cuya diferencia entre price y la media de su categoría sea mayor que 100

```
exercises> db.products.aggregate([
...   {$group: {
...     _id: "$category",
...     avg_price: {$avg: "$price"},
...     prods: {$push: {name: "$name", price: "$price"}}
...   }},
...   {$unwind: "$prods"},
...   {$match: {$expr: {$gt: [{$abs: {$subtract: ["$prods.price", "$avg_price"]}}, 100]}}}
... ])
[
  {
    _id: 'Electronics',
    avg_price: 733.3333333333334,
    prods: { name: 'Headphones', price: 200 }
  },
  {
    _id: 'Electronics',
    avg_price: 733.3333333333334,
    prods: { name: 'Laptop', price: 1200 }
  }
]
```

40. Devuelve los 10 usuarios más inactivos

```
exercises> db.users.aggregate([
...   {$addFields: {lastLogin: {$toDate: "$lastLogin"}}},
...   {$addFields: {lastLoginDays: {$dateDiff: {startDate: "$lastLogin", endDate: "$$NOW", unit: "day"}}}},
...   {$sort: {lastLoginDays: -1}},
...   {$limit: 10}
... ])
[
  {
    _id: 8,
    firstName: 'Frank',
    lastName: 'Moore',
    email: 'frank.moore@example.com',
    age: 50,
    username: 'frankm',
    lastLogin: ISODate('2023-11-30T00:00:00.000Z'),
    lastLoginDays: Long('770')
  },
  {
    _id: 6,
    firstName: 'David',
    lastName: 'Wilson',
    email: 'david.wilson@example.com',
    age: 38,
    username: 'davidw',
    lastLogin: ISODate('2024-01-15T00:00:00.000Z'),
    lastLoginDays: Long('724')
  },
]
```

41. Devuelve los usuarios registrados entre los últimos 10 y 480 días

```
exercises> db.users.aggregate([
...   {$addFields: {lastLogin: {$toDate: "$lastLogin"}}},
...   {$addFields: {lastLoginDays: {$dateDiff: {startDate: "$lastLogin", endDate: "$$NOW", unit: "day"}}}},
...   {$match: {lastLoginDays: {$gte: 10, $lte: 480}}}
... ])
[
]
```

42. Devuelve lugares cuya distancia sea menor que $2 * \text{rating} * 10000$

```
exercises> db.places.aggregate([
...   {$geoNear:
...     {near: {type: "Point", coordinates: [2.29, 48.85]},
...     distanceField: "dist",
...     spherical: true
...   }},
...   {$match: {$expr: {$lt: ["$dist", {$multiply: [2, 10000, "$rating"]}]}}}
... ])
[
  {
    _id: 2,
    name: 'Eiffel Tower',
    location: { type: 'Point', coordinates: [ 2.2945, 48.8584 ] },
    rating: 4.8,
    dist: 991.4686457071444
  },
  {
    _id: 3,
    name: 'Louvre Museum',
    location: { type: 'Point', coordinates: [ 2.3364, 48.8606 ] },
    rating: 4.9,
    dist: 3597.5267043352774
  }
]
```

43. Devuelve los 2 productos más caros por categoría

```
exercises> db.products.aggregate([
...   {$sort: {price: -1}},
...   {$group: {
...     _id: "$category",
...     topProds: {$push: {name: "$name", price: "$price"}}
...   }},
...   {$project: {
...     category: "$_id",
...     topProducts: {$slice: ["$topProds", 2]}
...   }}])
[
  {
    _id: 'Electronics',
    category: 'Electronics',
    topProducts: [
      { name: 'Laptop', price: 1200 },
      { name: 'Smartphone', price: 800 }
    ]
  },
  {
    _id: 'Clothing',
    category: 'Clothing',
    topProducts: [
      { name: 'T-shirt', price: 1500 },
      { name: 'Jeans', price: 1000 }
    ]
  }
]
```

44. Devuelve los 5 productos más caros

```
exercises> db.products.aggregate([
...   {$sort: {price: -1}},
...   {$limit: 5}
... ])
[
  {
    _id: 1,
    name: 'Laptop',
    description: 'A high-performance laptop',
    price: 1200,
    category: 'Electronics'
  },
  {
    _id: 2,
    name: 'Smartphone',
    description: 'A latest model smartphone',
    price: 800,
    category: 'Electronics'
  },
  {
    _id: 3,
    name: 'T-shirt',
    description: 'A comfortable cotton t-shirt',
    price: 1500,
    category: 'Clothing'
  },
  {
    _id: 4,
    name: 'Jeans',
    description: 'A pair of stylish jeans',
    price: 1000,
    category: 'Clothing'
  },
  {
    _id: 5,
    name: 'Sneakers',
    description: 'A pair of comfortable sneakers',
    price: 900,
    category: 'Footwear'
  }
]
```

45. Encuentra los lugares que estén entre 1 km y 10 km del centro de Paris que contengan la palabra Tower en el nombre.

```
exercises> db.places.aggregate([
...   {$geoNear: {
...     near: {type: "Point", coordinates: [2.34, 48.85]},
...     minDistance: 1000,
...     maxDistance: 10000,
...     spherical: true,
...     distanceField: "dist"
...   }},
...   {$match: {name: {$regex: "Tower"}}}
... ])
[
  {
    _id: 2,
    name: 'Eiffel Tower',
    location: { type: 'Point', coordinates: [ 2.2945, 48.8584 ] },
    rating: 4.8,
    dist: 3461.357639603731
  }
]
```

46. Calcula la distancia media de todos los lugares respecto al centro de Paris

```
exercises> db.places.aggregate([
...   {$geoNear: {
...     near: {type: "Point", coordinates: [2.34,48.85]},
...     spherical: true,
...     distanceField: "dist"
...   }},
...   {$group: {
...     _id: null,
...     avg_dist: {$avg: "$dist"}
...   }}
... ])
[ { _id: null, avg_dist: 6395869.882103038 } ]
```

47. Lugares a ≤ 5 km cuyo nombre empiece por una vocal.

```
exercises> db.places.aggregate([
...   {$geoNear: {
...     near: {type: "Point", coordinates: [2.34,48.85]},
...     maxDistance: 50000,
...     spherical: true,
...     distanceField: "dist"
...   }},
...   {$match: {name: {$regex: "^[aeiouáéíóú]", $options:"i"}}}
... ])
[
  {
    _id: 2,
    name: 'Eiffel Tower',
    location: { type: 'Point', coordinates: [ 2.2945, 48.8584 ] },
    rating: 4.8,
    dist: 3461.357639603731
  }
]
```

48. Los 5 lugares más lejanos al centro de Paris

```
exercises> db.places.aggregate([
...   {$geoNear: {
...     near: {type: "Point", coordinates: [2.34,48.85]},
...     spherical: true,
...     distanceField: "dist"
...   }},
...   {$sort: {dist: -1}},
...   {$limit: 5}
... ])
[
  {
    _id: 7,
    name: 'Sydney Opera House',
    location: { type: 'Point', coordinates: [ 151.2153, -33.8568 ] },
    rating: 4.7,
    dist: 16980005.923296284
  },
  {
    _id: 9,
    name: 'Machu Picchu',
    location: { type: 'Point', coordinates: [ -72.545, -13.1631 ] },
    rating: 4.8,
    dist: 10046728.899198016
  },
]
```

49.

Queries Redis

1. **Conexión a Redis:** Conéctate a un servidor Redis y verifica la conexión con PING

```
127.0.0.1:6379> PING
PONG
```

2. **Operaciones Básicas de Clave-Valor:** Guarda, recupera y elimina un valor usando SET, GET y DEL

```
127.0.0.1:6379> SET c1 "v1"
OK
127.0.0.1:6379> GET c1
"v1"
127.0.0.1:6379> DEL c1
(integer) 1
```

3. **Expiración de Claves:** Configura una expiración de 10 segundos para una clave con EXPIRE.

```
127.0.0.1:6379> SET c1 "v1" EX 10
OK
127.0.0.1:6379> TTL c1
(integer) 4
```

4. **Listas:** Añade elementos a una lista con LPUSH, recupéralos con LRANGE y elimina el primero con LPOP.

```
127.0.0.1:6379> LRANGE l1 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379> LPOP l1
"5"
127.0.0.1:6379> LRANGE l1 0 -1
1) "4"
2) "3"
3) "2"
4) "1"
```

5. **Conjuntos:** Añade elementos a un conjunto con SADD, recupéralos con SMEMBERS y verifica pertenencia con SISMEMBER.

```
127.0.0.1:6379> SADD s1 1 2 3 4 5
(integer) 5
127.0.0.1:6379> SMEMBERS s1
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
127.0.0.1:6379> SISMEMBER s1 2
(integer) 1
```

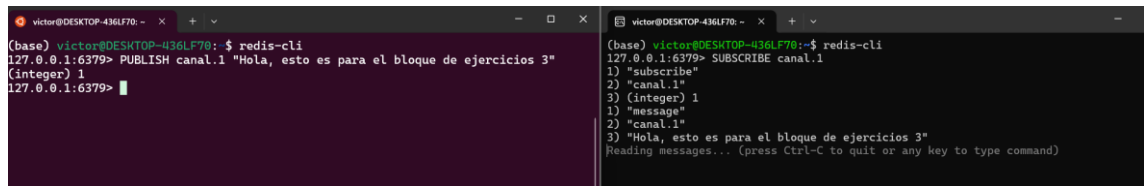
6. **Hashes:** Añade campos y valores a un hash con HSET, recupéralos con HGETALL y un campo específico con HGET

```
127.0.0.1:6379> HSET h1 c1 "v1" c2 "v2"
(integer) 2
127.0.0.1:6379> HGETALL h1
1) "c1"
2) "v1"
3) "c2"
4) "v2"
127.0.0.1:6379> HGET h1 c1
"v1"
```

7. **Conjuntos Ordenados (ZSET):** Añade elementos con puntuaciones a un conjunto ordenado con ZADD y recupéralos con ZRANGE

```
127.0.0.1:6379> ZADD ss1 1 5 2 7 5 1
(integer) 3
127.0.0.1:6379> ZRANGE ss1 0 -1
1) "5"
2) "7"
3) "1"
```

8. **Publicar y Suscribirse (Pub/Sub):** Suscríbete a un canal con SUBSCRIBE y publica un mensaje con PUBLISH.



```
(base) victor@DESKTOP-436LF70:~$ redis-cli
127.0.0.1:6379> PUBLISH canal.1 "Hola, esto es para el bloque de ejercicios 3"
(integer) 1
127.0.0.1:6379>

(base) victor@DESKTOP-436LF70:~$ redis-cli
127.0.0.1:6379> SUBSCRIBE canal.1
1) "subscribe"
2) "canal.1"
3) (integer) 1
1) "message"
2) "canal.1"
3) "Hola, esto es para el bloque de ejercicios 3"
Reading messages... (press Ctrl-C to quit or any key to type command)
```

9. **Transacciones Básicas:** Inicia una transacción con MULTI, añade comandos y ejecuta con EXEC.

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379(TX)> INCR bar
QUEUED
127.0.0.1:6379(TX)> INCR bar
QUEUED
127.0.0.1:6379(TX)> INCR bar
QUEUED
127.0.0.1:6379(TX)> DECR bar
QUEUED
127.0.0.1:6379(TX)> EXEC
1) (integer) 4
2) (integer) 5
3) (integer) 6
4) (integer) 5
```

10. **Transacciones con Descartar:** Inicia una transacción con MULTI, añade comandos y descarta con DISCARD.

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379(TX)> INCR bar
QUEUED
127.0.0.1:6379(TX)> INCR bar
QUEUED
127.0.0.1:6379(TX)> INCR bar
QUEUED
127.0.0.1:6379(TX)> DISCARD
OK
127.0.0.1:6379> |
```

11. **Transacciones con WATCH:** Usa WATCH para monitorear claves, inicia una transacción con MULTI y ejecuta con EXEC

```
127.0.0.1:6379> WATCH saldo
OK
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379(TX)> SET saldo 100
QUEUED
127.0.0.1:6379(TX)> GET saldo
QUEUED
127.0.0.1:6379(TX)> EXEC
1) OK
2) "100"
```

12. **Transacciones con Unwatch:** Usa WATCH para monitorear claves, desactiva la monitorización con UNWATCH y ejecuta con EXEC.

```
127.0.0.1:6379> WATCH saldo
OK
127.0.0.1:6379> UNWATCH
OK
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379(TX)> SET saldo 150
QUEUED
127.0.0.1:6379(TX)> GET saldo
QUEUED
127.0.0.1:6379(TX)> EXEC
1) OK
2) "150"
```

13. **Conjuntos Ordenados (ZSET) - Rango y Eliminación:** Recupera elementos dentro de un rango de puntuaciones con ZRANGEBYSCORE y elimina elementos con ZREM.

```
127.0.0.1:6379> ZRANGEBYSCORE ss1 2 7
1) "7"
2) "2"
3) "1"
4) "3"
127.0.0.1:6379> ZREM ss1 1
(integer) 1
127.0.0.1:6379> ZRANGEBYSCORE ss1 2 7
1) "7"
2) "2"
3) "3"
```

14. **Conjuntos Ordenados (ZSET) - Contar y Obtener Rango:** Cuenta elementos dentro de un rango de puntuaciones con ZCOUNT y obtén el rango de un elemento con ZRANK

```
127.0.0.1:6379> ZCOUNT ss1 2 6
(integer) 3
127.0.0.1:6379> ZRANK ss1 7
(integer) 1
127.0.0.1:6379> |
```

15. **Conjuntos Ordenados (ZSET) - Incrementar Puntuación:** Incrementa la puntuación de un elemento con ZINCRBY y verifica la nueva puntuación con ZSCORE.

```
127.0.0.1:6379> ZINCRBY ss1 7 2
"10"
127.0.0.1:6379> ZSCORE ss1 7
"2"
127.0.0.1:6379> ZSCORE ss1 2
"10"
127.0.0.1:6379> |
```

16. **Configuración de Cachés - Listas de Caché:** Añade valores a una lista con LPUSH, recupéralos con LRANGE y configura una expiración con EXPIRE.

```
127.0.0.1:6379> LPUSH l1 5 EX 10
(integer) 7
127.0.0.1:6379> LRANGE l1 0 -1
1) "10"
2) "EX"
3) "5"
4) "4"
5) "3"
6) "2"
7) "1"
127.0.0.1:6379> EXPIRE l1 20
(integer) 1
```

17. **Configuración de Cachés - Caché de Hashes:** Guarda campos y valores en un hash con HSET, recupéralos con HGETALL y configura una expiración con EXPIRE.

```
127.0.0.1:6379> HSET h1 c3 "v3"
(integer) 1
127.0.0.1:6379> HGETALL h1
1) "c1"
2) "v1"
3) "c2"
4) "v2"
5) "c3"
6) "v3"
127.0.0.1:6379> EXPIRE h1 20
(integer) 1
```

18. **Configuración de Cachés - Caché de Conjuntos:** Añade valores a un conjunto con SADD, recupéralos con SMEMBERS y configura una expiración con EXPIRE.

```
127.0.0.1:6379> SADD s1 10 19 15
(integer) 3
127.0.0.1:6379> SMEMBERS s1
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
7) "7"
8) "9"
9) "10"
10) "15"
11) "19"
127.0.0.1:6379> EXPIRE s1 20
(integer) 1
```

19. **Configuración de Cachés - Caché de Conjuntos Ordenados:** Añade valores con puntuaciones a un conjunto ordenado con ZADD, recupéralos con ZRANGE y configura una expiración con EXPIRE.

```
127.0.0.1:6379> ZADD z1 19 k
(integer) 1
127.0.0.1:6379> ZRANGE z1 0 -1
1) "k"
127.0.0.1:6379> EXPIRE z1 20
(integer) 1
```

20. **Operaciones de Incremento:** Guarda un valor numérico con SET, incrementa con INCR y decrementa con DECR

```
127.0.0.1:6379> SET c2 5
OK
127.0.0.1:6379> INCR c2
(integer) 6
127.0.0.1:6379> DECR c2
(integer) 5
```

Queries Neo4j

1. **Agregar más personas:** Crea nodos para dos nuevas personas, Dave y Eve, con sus respectivas edades.

```
neo4j$ CREATE (d:Person {name: 'Dave', age: 60}),(e:Person {name: 'Eve', age: 70})
```



Added 2 labels, created 2 nodes, set 4 properties, completed after 35 ms.

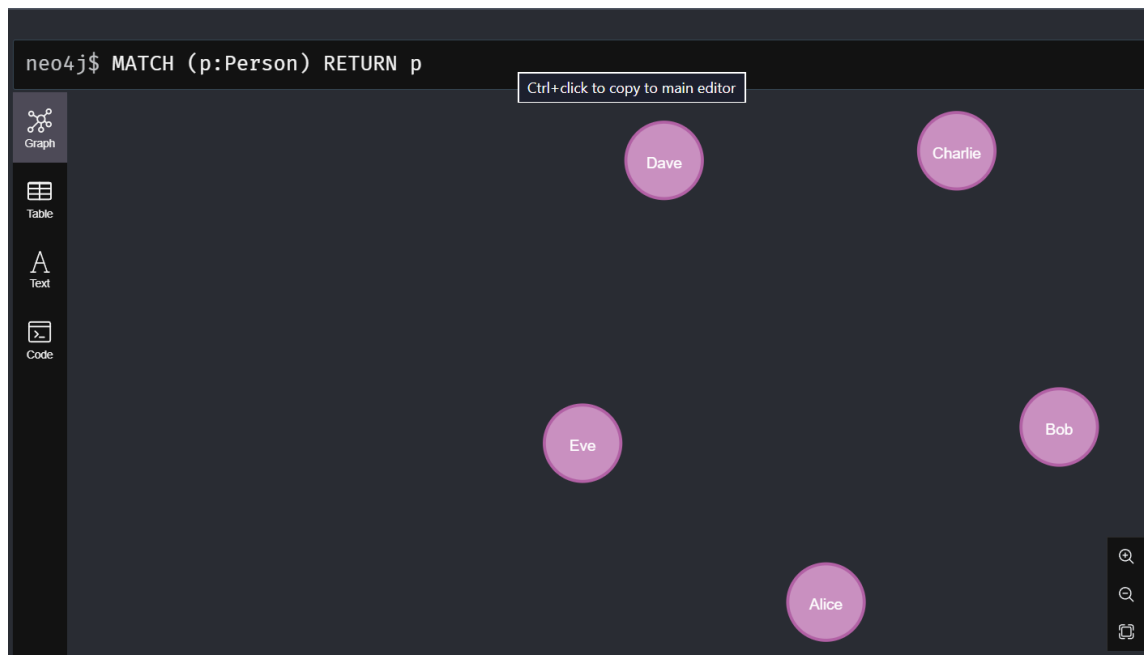
2. **Relacionar personas con ciudades:** Relaciona a Dave con la ciudad de New York (LIVES_IN).

```
1 MATCH (d:Person {name: "Dave"}),(ny:City {name: "New York"})
2 CREATE (d)-[:LIVES_IN]->(ny)
```

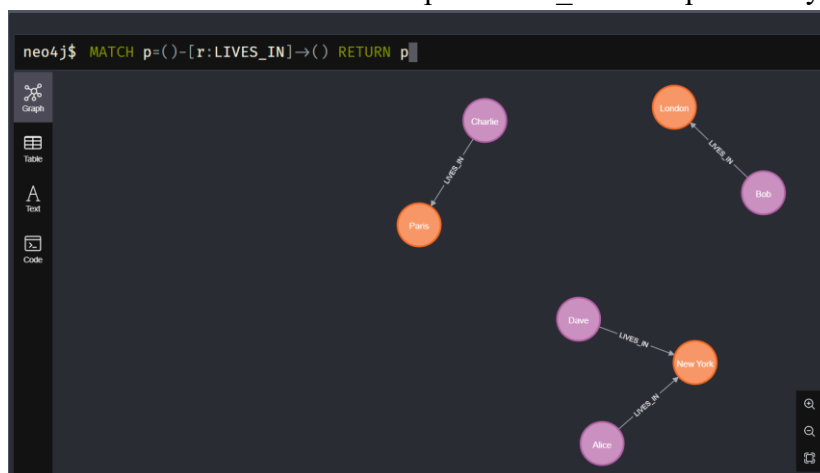


Created 1 relationship, completed after 11 ms.

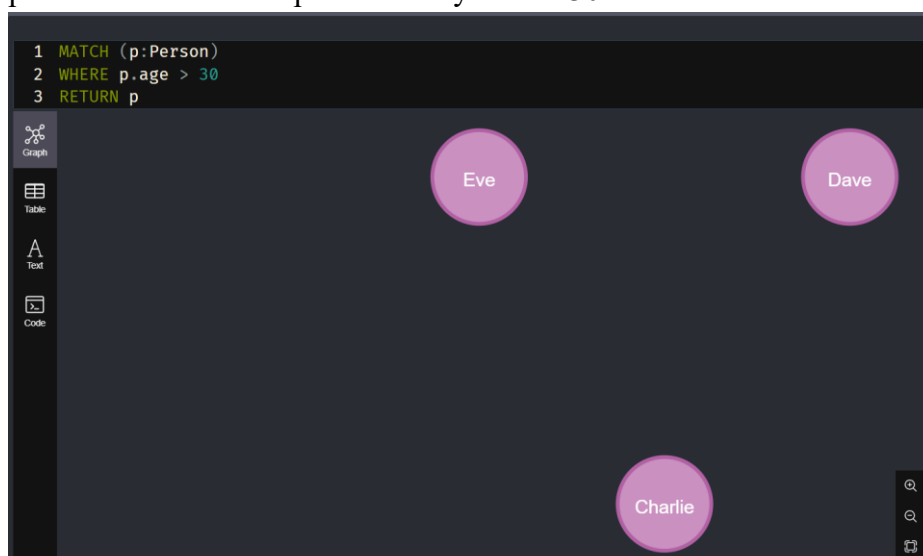
3. **Consultar todas las personas:** Realiza una consulta para obtener todos los nodos de tipo Persona



- Consultar relaciones de residencia:** Realiza una consulta para obtener todas las relaciones de tipo `LIVES_IN` entre personas y ciudades.



- Consultar personas mayores de 30 años:** Realiza una consulta para obtener todas las personas mayores de 30 años.



6. **Actualizar la edad de Eve:** Actualiza la edad de Eve a 23 años

```
neo4j$ MATCH (p:Person {name: 'Eve'}) SET p.age = 23
```



Table

Set 1 property, completed after 8 ms.

7. **Eliminar la relación de residencia de Bob:** Elimina la relación de residencia entre Bob y la ciudad de London

```
1 MATCH (b:Person {name: 'Bob'})-[r:LIVES_IN]-(l:City {name: 'London'})
2 DELETE r
```



Deleted 1 relationship, completed after 20 ms.

8. **Eliminar el nodo de Dave:** Elimina el nodo correspondiente a Dave.

```
neo4j$ MATCH (d:Person {name: 'Dave'})-[r:LIVES_IN]-(ny:City {name: 'New York'}) DELETE r,d
```



Table

Deleted 1 node, deleted 1 relationship, completed after 11 ms.

9. **Crear un nodo de empresa y relacionarlo con Alice:** Crea un nodo para una empresa llamada TechCorp y relaciona a Alice con esta empresa.

```
1 MATCH (a:Person {name: 'Alice'})
2 CREATE (t:Business {name: "TechCorp"})
3 CREATE (a)-[:WORKS_FOR]-(t)
```



Table

Added 1 label, created 1 node, set 1 property, created 1 relationship, completed after 14 ms.

10. **Consultar personas y las empresas donde trabajan:** Realiza una consulta para obtener los nombres de las personas y las empresas donde trabajan

```
1 MATCH (p:Person)-[:WORKS_FOR]-(b:Business)
2 RETURN p.name, b.name
```



Table



Table

p.name

b.name

1

"Alice"

"TechCorp"

11. **Amigos del usuario con id 1.**

```
1 MATCH (p:Persona {id:1})-[:FRIEND]-(p2:Persona)
2 RETURN p2
```

12. **Amigos o familiares del usuario con id 8.**

```
MATCH (p:Persona {id:8})-[:FRIEND|FAMILY]-(p2:Persona)
RETURN p2
```

13. Familiares de familiares de 8, excluyendo al propio usuario y sin duplicados.

```
MATCH (p1:Persona {id:8})-[:FAMILY]→(p2:Persona)-[:FAMILY]→(p3:Persona)
WHERE p3 <> p1
RETURN DISTINCT p3
```

14. Mostrar la conversación completa entre los usuarios 5 y 3, ordenada correctamente por fecha.

```
MATCH (p1:Persona {id: 5})-[:SENT]→(m:Mensaje)-[:RECEIVED]→(p2:Persona {id: 3})
RETURN m
ORDER BY m.fecha ASC
```

15. Obtener el mensaje más reciente enviado por el usuario 7, ordenado de más antiguo a más reciente

```
1 MATCH (p:Persona {id: 7})-[:SENT]→(m:Mensaje)
2 RETURN m
3 ORDER BY m.fecha ASC
4 LIMIT 1
```

16. Obtener todos los usuarios mencionados en publicaciones realizadas por el usuario con id 0, sin repetir usuarios.

```
MATCH (p1:Persona {id:0})-[:POSTED]→(p:Publicacion)-[:MENTIONS]→(p2:Persona)
RETURN DISTINCT p2
```

17. Obtener el número total de mensajes enviados por cada usuario

```
1 MATCH (p1:Persona)-[:SENT]→(m:Mensaje)
2 WITH p1.name as usuario, count(s) as numMensajes
3 RETURN usuario, numMensajes
```

18. Obtener los usuarios que han enviado más de 4 mensajes

```
1 MATCH (p1:Persona)-[:SENT]→(m:Mensaje)
2 WITH p1.name as usuario, count(s) as numMensajes
3 WHERE numMensajes > 4
4 RETURN usuario, numMensajes
```

19. Obtener los usuarios mencionados en publicaciones del usuario y cuántas veces se menciona a cada uno.

```
1 MATCH (p1:Persona {id:0})-[:POSTED]→(p:Publicacion)-[:MENTIONS]→(p2:Persona)
2 WITH p2.name as usuarioMencionado, count(p) as numMenciones
3 RETURN usuarioMencionado, numMenciones
```

20. Obtener los usuarios conectados al usuario 1 en exactamente 2 saltos, excluyendo al propio usuario.

```
1 MATCH (p1:Persona {id: 1})-[*2]-(p2:Persona)
2 WHERE p2 <> p1
3 RETURN p2
```

21. Obtener los usuarios conectados a 1 en hasta 4 saltos, mostrando el número de hops, ordenados por cercanía.

```
1 MATCH p=(u:Persona {id:1})-[*..4]-(x:Persona)
2 WHERE x < u
3 WITH x.name as usuario, min(length(p)) as numSaltos
4 RETURN usuario, numSaltos
5 ORDER BY numSaltos ASC
```

22. Obtener usuarios sin relación directa con el usuario 1, conectados a través de otros usuarios, mostrando el número de hops.

```
1 MATCH path=(p1:Persona {id:1})-[*..4]-(p2:Persona)
2 WHERE NOT (p1)-[]-(p2)
3 WITH p2.name AS usuario, min(length(path)) AS numSaltos
4 RETURN usuario, numSaltos
5 ORDER BY numSaltos DESC
```

23. Obtener nuevas conexiones del usuario 1, mostrando: el usuario tercero, el usuario intermedio por el que se llega, el número de hops

```
1 MATCH path=(p1:Persona{id: 1})-[]→(p2:Persona)-[*..3]→(p3:Persona)
2 WHERE p1 < p3 AND NOT (p1)-[:FAMILY|FRIEND]→(p3)
3 WITH p3.name AS tercero, p2.name AS intermedio, min(length(path)) as numSaltos
4 RETURN tercero, intermedio, numSaltos
5 ORDER BY numSaltos DESC
```

24. Obtener nuevas conexiones del usuario 1, conectadas vía usuarios intermedios con los que haya intercambiado más de 1 mensaje

```
MATCH path=(p1:Persona{id: 1})-[:FRIEND|FAMILY]→(p2:Persona)-[*1..3]→(p3:Persona)
WHERE NOT (p1)-[:FRIEND|FAMILY]→(p3) AND p1 < p3
MATCH msg_path=(p1)-[:SENT]→(m:Mensaje)-[:RECEIVED]→(p2)
WITH p3,p2, count(m) AS numMsgs
WHERE numMsgs >= 1
RETURN p3.name AS tercero, p2.name AS intermediario, numMsgs
```

25. Obtener todos los mensajes enviados de un usuario determinado a otro usuario determinado después de una fecha especificada.

```
1 MATCH (n:Persona {id: 11})-[:SENT]→(m:Mensaje)
2 WHERE m.fecha = "2026-01-09"
3 RETURN m
```

26. Obtener la conversación completa entre dos usuarios determinados.

```
1 MATCH (p1:Persona)-[:SENT]→(m:Mensaje)-[:RECEIVED]→(p2:Persona)
2 WHERE p1.id IN [11,12] AND p2.id IN [11,12]
3 RETURN p1.name, p2.name, m.texto
4 ORDER BY m.seq ASC
```

27. Obtener todos los usuarios mencionados por un usuario determinado los cuales tengan una relación laboral con el usuario que los mencionó.

```
1 MATCH (p1:Persona {id: 7})-[:POSTED]→(pub:Publicacion)-[:MENTIONS]→(p2:Persona)
2 WHERE p1 < p2 AND (p1)-[:WORKS_AT]→(:Empresa)←[:WORKS_AT]-(p2)
3 RETURN p2
```

Exámenes Anteriores

Extraordinaria 2025

Mongo

1. Realizar las operaciones necesarias para poder ejecutar las siguientes consultas de forma eficiente. Como máximo 4 consultas. Pueden ser necesarias menos de 4:

```
db.zips.createIndex({state: "text"})
db.zips.createIndex({population: 1})
db.zips.createIndex({location: "2dsphere"})
```

2. Estados (state) cuya población media sea mayor que 15000 habitantes a excepción del estado de California (CA) y Florida (FL).

```
exercises> db.zips.aggregate([
...   {$group: {
...     _id: "$state",
...     avg_pop: {$avg: "$population"}
...   }},
...   {$match: {$and: [
...     {$expr: {$gt: ["$avg_pop", 15000]}}
...     {_id: {$nin: ["CA", "FL"]}}
...   ]}}])
[ { _id: 'DC', avg_pop: 25287.5 } ]
```

3. Listado de las ciudades (city) agrupadas por estado (state) que estén a menos de 15km de las coordenadas con longitud -72.62 y latitud 42.07

```
exercises> db.zips.aggregate([
...   {$geoNear: {
...     near: {type: "Point", coordinates: [-72.62, 42.07]},
...     maxDistance: 15000,
...     spherical: true,
...     distanceField: "dist"
...   }},
...   {$group: {
...     _id: "$state",
...     cities: {$push: "$city"}
...   }
... ]})
```

Redis

1. Inicializar la BBDD con las reproducciones para dos usuarios:
 - a. La usuaria “Conchi” ha reproducido las canciones “Lola” 2 veces, y “Amor” 6 veces.

```
127.0.0.1:6379> ZADD Conchi 2 "Lola" 6 Amor  
(integer) 2
```

- b. El usuario “Adolfo” ha reproducido las canciones, “Adios” 30 veces, “Dolor” 2 veces y “Amor” 15 veces.

```
127.0.0.1:6379> ZADD Adolfo 30 Adios 2 Dolor 15 Amor  
(integer) 3
```

2. Indicar que la usuaria “Conchi” ha reproducido una vez más la canción “Amor”

```
127.0.0.1:6379> ZINCRBY Conchi 1 Amor  
"7"
```

3. Devolver las reproducciones totales de todas las canciones realizadas por los dos usuarios

```
127.0.0.1:6379> ZUNION 2 Conchi Adolfo WITHSCORES  
1) "Dolor"  
2) "2"  
3) "Lola"  
4) "2"  
5) "Amor"  
6) "22"  
7) "Adios"  
8) "30"
```

4. Crear un nuevo listado llamado “Conchi-Adolfo” de las canciones que han escuchado tanto “Adolfo” como “Conchi” con el valor de reproducciones más bajo realizado por cualquiera de los dos usuarios

```
127.0.0.1:6379> ZINTERSTORE Conchi-Adolfo 2 Conchi Adolfo AGGREGATE MIN  
(integer) 1
```

Neo4j

1. Crear cuatro nodos de tipo city con un atributo llamado type que siempre tendrá el valor "province" y otro atributo name con los siguientes valores: "Madrid", "Valencia", "Sevilla" y "Barcelona"

```
1 CREATE (m:City {name: "Madrid", type: "province"}),  
2 (v:City {name: "Valencia", type: "province"}),  
3 (s:City {name: "Sevilla", type: "province"}),  
4 (b:City {name: "Barcelona", type: "province"})
```

2. Crear una relación etiquetada como "road" en una sola dirección (cualquiera) entre cada dos ciudades diferentes. No puede haber relaciones de las ciudades consigo mismas

```
MATCH (v:City {name: "Valencia"}),(m:City {name: "Madrid"})
CREATE (v)-[:road]->(m)
```

3. Borrar todos los atributos "type" de todos los nodos la base de datos

```
1 MATCH (n)
2 REMOVE n.type
```

Extraordinaria 2023

Mongo

1. Misma que examen anterior
2. Devolver el listado de ciudades con las diferentes ciudades del estado de Nevada (NV)

```
exercises> db.zips.aggregate([
... {$match: {state: "NV"}},
... {$group: {_id: "$city"}}
... ])
```

3. Población media de las ciudades que estén a menos de 20km a la redonda de las coordenadas con longitud -68.45 y latitud 44.89

```
exercises> db.zips.aggregate([
... {$geoNear: {
... near: {type: "Point", coordinates: [-68.45,44.89]},
... maxDistance: 20000,
... spherical: true,
... distanceField: "dist"
... }},
... {$group: {
... _id: null,
... avg_pop: {$avg: "$population"}
... }}
... ])
[ { _id: null, avg_pop: 3848.714285714286 } ]
```

Redis

1. Almacenar la información

```
127.0.0.1:6379> HSET amazon "2018" 17.0 "2019" 18.2
(integer) 2
127.0.0.1:6379> HSET facebook "2018" 12.0 "2019" 11.8
(integer) 2
127.0.0.1:6379> HSET apple "2018" 16.1 "2019" 17.0
(integer) 2
127.0.0.1:6379> HSET microsoft "2018" 13.0 "2019" 13.5
(integer) 2
```

2.

Neo4j