Reportes de Personas - Consultas GraphQL y Cypher

Introducción

Este documento describe las consultas GraphQL y Cypher disponibles para generar reportes detallados sobre personas y sus relaciones en el sistema. Las consultas están diseñadas para proporcionar información valiosa sobre la red de conexiones, amistades y relaciones familiares.

1. Lista de Amigos de una Persona Específica

Descripción

Obtiene todos los amigos de una persona específica (personas relacionadas con status `FRIEND`).

Consulta GraphQL

```
query GetFriends($personId: ID!) {
  people(where: { id: $personId }) {
    id
    name
    nickname
    email
    friends {
      id
      name
      nickname
      email
    }
  }
}
```

Variables de Ejemplo

```
{
  "personId": "person-id-1"
}
```

Código Cypher Subyacente

```
MATCH (p:Person {id: $personId})
MATCH (p)-[rel:HAS_RELATIONSHIP {status: "FRIEND"}]->(friend:Person)
RETURN {
  id: friend.id,
  name: friend.name,
  nickname: friend.nickname,
  email: friend.email
} as friend
```

Respuesta de Ejemplo


```
{
  "data": {
    "people": [
      {
        "id": "person-1",
        "name": "Juan García",
        "nickname": "juangar",
        "email": "juan@example.com",
        "friends": [
          {
            "id": "person-2",
            "name": "María López",
            "nickname": "mariaL",
            "email": "maria@example.com"
          },
          {
            "id": "person-3",
            "name": "Carlos Martínez",
            "nickname": "carlosM",
            "email": "carlos@example.com"
          }
        ]
      }
    ]
  }
}
```


2. Lista de Familiares de una Persona Específica

Descripción

Obtiene todos los familiares de una persona específica (personas relacionadas con status `FAMILY`).

Consulta GraphQL

```graphql
query GetFamilyMembers($personId: ID!) {
  people(where: { id: $personId }) {
    id
    name
    nickname
    email
    familyMembers {
      id
      name
      nickname
      email
    }
  }
}
```

Variables de Ejemplo

```json
{
  "personId": "person-id-1"
}
```

Código Cypher Subyacente

```cypher
MATCH (p:Person {id: $personId})
MATCH (p)-[rel:HAS_RELATIONSHIP {status: "FAMILY"}]->(family:Person)
RETURN {
  id: family.id,
  name: family.name,
  nickname: family.nickname,
  email: family.email
} as family
```

Respuesta de Ejemplo


```
{
  "data": {
    "people": [
      {
        "id": "person-1",
        "name": "Juan García",
        "nickname": "juangar",
        "email": "juan@example.com",
        "familyMembers": [
          {
            "id": "person-5",
            "name": "Ana García",
            "nickname": "anaG",
            "email": "ana@example.com"
          },
          {
            "id": "person-6",
            "name": "Roberto García",
            "nickname": "robertoG",
            "email": "roberto@example.com"
          }
        ]
      }
    ]
  }
}
```


3. Amigos Comunes entre Dos Personas

Descripción

Obtiene los amigos comunes entre dos personas específicas. Identifica las personas que son amigas de ambas.

Consulta GraphQL

```graphql
query GetMutualFriends($personId1: ID!, $personId2: ID!) {
  mutualFriendsQuery(personId1: $personId1, personId2: $personId2) {
    id
    name
    nickname
    email
    mutualFriendsCount
  }
}
```

Variables de Ejemplo

```json
{
  "personId1": "person-1",
  "personId2": "person-2"
}
```

Código Cypher Subyacente

```cypher
MATCH (p1:Person {id: $personId1})-[rel1:HAS_RELATIONSHIP {status: "FRIEND"}]->(friend:Person)
MATCH (p2:Person {id: $personId2})-[rel2:HAS_RELATIONSHIP {status: "FRIEND"}]->(friend)
RETURN {
  id: friend.id,
  name: friend.name,
  nickname: friend.nickname,
  email: friend.email,
  mutualFriendsCount: 1
} as result
```

Respuesta de Ejemplo


```
{
  "data": {
    "mutualFriendsQuery": [
      {
        "id": "person-3",
        "name": "Carlos Martínez",
        "nickname": "carlosM",
        "email": "carlos@example.com",
        "mutualFriendsCount": 1
      },
      {
        "id": "person-4",
        "name": "Laura Fernández",
        "nickname": "lauraF",
        "email": "laura@example.com",
        "mutualFriendsCount": 1
      }
    ]
  }
}
```


4. Persona con Mayor Número de Relacionados

Descripción

Identifica la persona con el mayor número de relaciones activas en la red (amigos, familiares, colegas, etc.).

Consulta GraphQL - Opción 1 (Estándar)

```
query GetMostConnectedPerson {
  people(options: { limit: 1, sort: [{ relationships: { totalCount: DESC } }] }) {
    id
    name
    nickname
    email
  }
}
```

Consulta GraphQL - Opción 2 (Cypher Personalizado)

```
query GetMostConnectedPerson {
  mostConnectedPerson {
    id
    name
    nickname
    email
  }
}
```

Código Cypher Subyacente

```
MATCH (p:Person)-[rel:HAS_RELATIONSHIP]->()
WITH p, COUNT(rel) as connectionCount
ORDER BY connectionCount DESC
RETURN p
LIMIT 1
```

Respuesta de Ejemplo


```
{
  "data": {
    "mostConnectedPerson": {
      "id": "person-10",
      "name": "Pedro Sánchez",
      "nickname": "pedroS",
      "email": "pedro@example.com"
    }
  }
}
```


5. Personas Influyentes

Descripción

Obtiene las personas más influyentes de la red, ordenadas por el promedio de importancia en sus relaciones. La influencia se calcula como el promedio del valor de importancia de todas las relaciones de una persona.

Consulta GraphQL

```
query GetInfluentialPeople($limit: Int = 10) {
  influentialPeople(limit: $limit) {
    id
    name
    nickname
    email
    averageImportance
  }
}
```

Variables de Ejemplo

```
{
  "limit": 10
}
```

Código Cypher Subyacente

```
MATCH (p:Person)-[rel:HAS_RELATIONSHIP]->()
WITH p, CASE WHEN COUNT(rel) > 0 THEN toFloat(SUM(rel.importance)) / COUNT(rel) ELSE 0.0 END as
ORDER BY avgImportance DESC
RETURN p
LIMIT $limit
```

Cálculo de Influencia

Para cada persona, se calcula:

$$\text{AverageImportance} = \frac{\sum_{i=1}^{n} \text{importance}_i}{n}$$

Donde:

- n = número total de relaciones
- importance_i = valor de importancia de cada relación (rango 1-10)

Respuesta de Ejemplo


```
{
  "data": {
    "influentialPeople": [
      {
        "id": "person-8",
        "name": "Sofía González",
        "nickname": "sofiaG",
        "email": "sofia@example.com",
        "averageImportance": 8.5
      },
      {
        "id": "person-9",
        "name": "Miguel Rodríguez",
        "nickname": "miguelR",
        "email": "miguel@example.com",
        "averageImportance": 8.2
      },
      {
        "id": "person-1",
        "name": "Juan García",
        "nickname": "juangar",
        "email": "juan@example.com",
        "averageImportance": 7.8
      }
    ]
  }
}
```


Estructura de Datos - Propiedades de Relaciones

Todas las relaciones `HAS_RELATIONSHIP` contienen las siguientes propiedades:

Propiedad	Tipo	Descripción	Rango
`status`	Enum	Tipo de relación (FRIEND, FAMILY, COLLEAGUE)	-
`frequency`	Integer	Frecuencia de interacción	1-10

Propiedad	Tipo	Descripción	Rango
`importance`	Integer	Nivel de importancia de la relación	1-10

Tipos de Relaciones (Status)

- **FRIEND**: Relación de amistad
- **FAMILY**: Relación familiar
- **COLLEAGUE**: Relación laboral o de compañerismo

Notas de Implementación

1. **Directiva @cypher**: Las consultas personalizadas utilizan la directiva `@cypher` de Neo4j GraphQL para ejecutar código Cypher personalizado.
2. **Parámetros de Variables**: Los parámetros se pasan como variables GraphQL (`$personId` , `$limit` , etc.) y se convierten automáticamente a parámetros Cypher.
3. **Rendimiento**: Para bases de datos grandes, considere agregar índices en los campos `id` , `name` y `status` de las relaciones.
4. **Filtrado Adicional**: Todas las consultas pueden extenderse para filtrar por `frequency` , `importance` o rango de fechas según sea necesario.

Casos de Uso Comunes

Análisis de Red Social

Utilice las consultas de amigos comunes y personas influyentes para identificar usuarios clave en la red.

Gestión de Relaciones

Use la lista de amigos y familiares para personalizar comunicaciones y recomendaciones.

Detección de Comunidades

Combine las consultas para identificar grupos cerrados o comunidades dentro de la red.

Recomendaciones

Basándose en amigos comunes, sugiera nuevas conexiones a los usuarios.

Integración en el Frontend

Las consultas están disponibles en el endpoint GraphQL:

```
http://localhost:4000/graphql
```

Ejemplo de uso en el frontend con Apollo Client:

```
import { ApolloClient, gql } from '@apollo/client';

const GET_FRIENDS = gql`
  query GetFriends($personId: ID!) {
    people(where: { id: $personId }) {
      id
      name
      friends {
        id
        name
        nickname
        email
      }
    }
  }
`;

// Uso
client.query({
  query: GET_FRIENDS,
  variables: { personId: 'person-1' }
});
```

Versionado

- **Versión**: 1.0
- **Última actualización**: 4 de febrero de 2026
- **Compatibilidad**: Neo4j GraphQL 5.x+