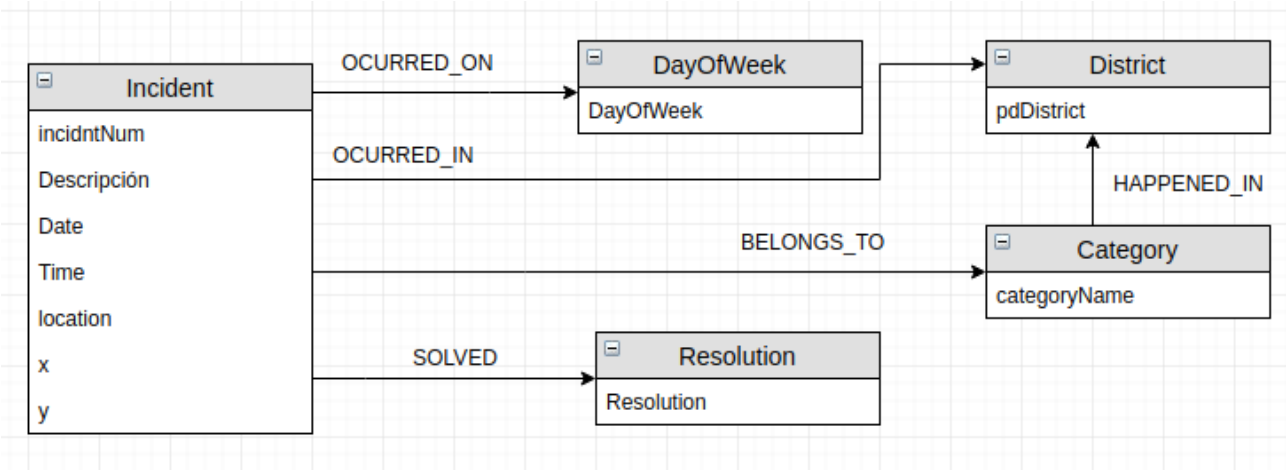
Neo4J

Planteamiento

Antes de comenzar, se esboza un esquema de la estructura que tendrá la base de datos. En este caso, se ha decidido definir 5 nodos unidos por 5 arcos de la forma que se muestra en la siguiente imagen:



Importar datos

Empezamos creando un nuevo proyecto, y dentro del mismo una nueva base de datos. En primer lugar, se importan los datos a partir de un fichero CSV y se crean los nodos 'Incident' mediante la siguiente sentencia:

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///incidents.csv" AS row
CREATE (:Incident {incidntNum: row.IncidntNum, descript: row.Descript, date: row.Date, time:
row.Time, x: row.X, y: row.Y, location: row.Location})
```

Se carga el fichero CSV creando un nodo por cada fila con los atributos definidos, en este caso incidntNum, descript, dayOfWeek, date, time y resolution. Gracias a la primera línea 'USING PERIODIC COMMIT' que almacena los datos cada 1000 registros.

La importación de las categorías se realiza con una sentencia similar a la anterior. Sin embargo, se utiliza la cláusula MERGE en lugar de CREATE para evitar importar en la base de datos categorías repetidas. Puesto que MERGE busca nodos con las mismas propiedades para evitar la repetición de datos,la sentencia se vuelve más lenta.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///incidents.csv" AS row
MERGE (:Category {categoryName: row.Category})
```

A continuación, se crean los nodos Resolution y DayOfWeek, que se importarán de forma similar a las categorías ya que no queremos almacenar registros repetidos. Estos nodos contienen las resoluciones de los incidentes y los días de la semana respectivamente.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///incidents.csv" AS row
MERGE (:Resolution {resolution: row.Resolution})
```

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///incidents.csv" AS row
MERGE (:DayOfWeek {dayOfWeek: row.DayOfWeek})
```

Finalmente, se importan los distritos. En este caso, se evita importar registros repetidos al igual que en los casos anteriores. Sin embargo, también hay que controlar la aparición de campos en blanco ya que en este caso fallaría la ejecución de la sentencia.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///incidents.csv" AS row
WITH row
WHERE NOT row.PdDistrict IS NULL
MERGE (:District {PdDistrict: row.PdDistrict})
```

Creación de relaciones

Ya que los datos están importados, es necesario conectarlos entre ellos mediante relaciones. Estas relaciones se crearán siguiendo el esquema anterior. En primer lugar, se crea la relación entre incidentes y categorías. Para ello, se vuelve a utilizar el documento CSV de forma que se utiliza un parámetro (incidntNum y categoryName) de cada etiqueta (Incident y Category) para enlazar los nodos.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///incidents.csv" AS row
MATCH (i:Incident {incidntNum: row.IncidntNum })
MATCH (c:Category {categoryName: row.Category})
MERGE (i)-[:BELONGS_TO]->(c);
```

De la misma forma se crean las otras relaciones District-Category, Incident-District, Incident-DayOfWeek e Incident-Resolution.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///incidents.csv" AS row
MATCH (d:District {PdDistrict: row.PdDistrict })
MATCH (c:Category {categoryName: row.Category})
MERGE (c)-[:HAPPENED_IN]->(d);
```

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///incidents.csv" AS row
MATCH (i:Incident {incidntNum: row.IncidntNum })
MATCH (d:District {PdDistrict: row.PdDistrict })
MERGE (i)-[:OCURRED_IN]->(d);

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///incidents.csv" AS row
MATCH (i:Incident {incidntNum: row.IncidntNum })
MATCH (d:DayOfWeek {dayOfWeek: row.DayOfWeek })
MERGE (i)-[:OCURRED_ON]->(d);

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///bdejemplo.csv" AS row
MATCH (i:Incident {incidntNum: row.IncidntNum })
MATCH (r:Resolution {Resolution: row.Resolution })
MERGE (i)-[:SOLVED]->(r);
```

Queries

Con la base de datos en funcionamiento, el siguiente paso es comenzar a crear consultas. Comenzamos con algunas simples como, por ejemplo todos los incidentes y las categorías a las que pertenecen. En la cláusula RETURN se definen los datos que se quieren mostrar como un nodo o una propiedad. En este caso sería conveniente utilizar la cláusula LIMIT para limitar los resultados y evitar que se ralentice demasiado la consulta.

```
MATCH (i:Incident)-[:BELONGS_TO]-(c:Category)
RETURN i, c
```

Si en la consulta anterior añadimos la cláusula WHERE, es posible obtener los incidentes que pertenecen a cierta categoría:

```
MATCH (i:Incident)-[:BELONGS_TO]-(c:Category)
WHERE c.categoryName="ROBBERY"
RETURN i, c
```

En la siguiente consulta se utiliza CONTAINS para encontrar aquellos incidentes que contienen la palabra 'DOG' en su descripción. El tiempo de ejecución de esta consulta es de 3454 ms.

```
MATCH (i:Incident)
WHERE i.descript CONTAINS 'DOG'
RETURN i
```

La siguiente consulta resulta algo más compleja ya que enlazamos tres tipos de nodos: Incident, Category y DayOfWeek. La finalidad de esta consulta es de obtener aquellos incidentes que pertenecen a la categoría 'RUNAWAY' y ocurrieron en Domingo. El tiempo de ejecución de esta consulta es de 20592 ms.

```
MATCH (c:Category)-[:BELONGS_TO]-(i1:Incident),(d:DayOfWeek)-[:OCURRED_ON]-(i2:Incident)
WHERE c.categoryName="RUNAWAY" AND d.dayOfWeek="Sunday" AND
i1.incidntNum=i2.incidntNum
RETURN c.categoryName,i1.descript,d.dayOfWeek
```

Siguiendo la estructura de esta consulta, es posible obtener diferentes variaciones que sean útiles para extraer datos de interes. En el siguiente caso, se utiliza la función COUNT, que cuenta los nodos. Esta consulta devuelve el número de incidentes que pertenecen a la categoría 'RUNAWAY' y que ocurrieron en Domingo.

```
MATCH (c:Category)-[:BELONGS_TO]-(i1:Incident),(d:DayOfWeek)-[:OCURRED_ON]-(i2:Incident)
WHERE c.categoryName="RUNAWAY" AND d.dayOfWeek="Sunday" AND
i1.incidntNum=i2.incidntNum
RETURN c.categoryName,count(i1),d.dayOfWeek
```