**Fundamentos teóricos y prácticos**

**1. Procesamiento y representación de la información**

Procedimiento en el análisis descarga y procesamiento de los datos antes de ser estructurados para cada tipo de modelo.

**2. Estructura de datos**

Explicación de como es la estructura teórica de como deben ir conformados los datos para cada modelo.

Introduccion (tecnologia)

Pros/contras

Estructura de la información

Arquitectura

Configuración e instalación

Preprocesamiento e importación de datos

Modelado de datos

**3. Representación de la información**

Introducción (como son y cuales son permitidas por tipo de tecnologia)

Consultas

**Procesamiento y representación de la información**

Una vez localizada y decidida de cúal versión de la fuente de datos partiremos, se procede a su **descarga**, **preprocesamiento** o limpieza de datos, para ellao añadiremos fundamentos necesarios para realizar todo el proceso para cada tipo de representación o modelador de datos: Cassandra, Neo4j y/o Mongodb.

* **Análisis de la fuente**. Elección de la versión de la fuente de datos, siguiendo los criterios:
  + Por separación de columnas: .tsv para Cassandra, y .csv para Neo4j y Mongodb
  + Sin filtros añadidos.
* **Descarga**. Por línea de comandos o directa desde el navegador.
* **Preprocesamiento** o limpieza de datos. Mediante *bash scripting* (Cassandra) o mediante scripts de *python*(Mongodb) y *R* (Neo4j).
* **Importación al modelo de datos**. Dependerá del motor que encapsule el modelo de datos en cada caso: Neo4j, Cassandra, Y Mongodb.

**Análisis de la fuente y descarga**. Toda la información que se muestra en el mapa de incidencias, si desactivamos los filtros por defecto, se puede descargar mediante línea de comandos gracias al portal Socrata, lo haremos de la siguiente manera:

wget -O incidents.raw.tsv "https://data.sfgov.org/api/views/tmnf-yvry/rows.tsv?accessType=DOWNLOAD&api\_foundry=true"

**Preprocesamiento o limpieza de datos**. La limpieza de datos se ha hecho de acuerdo al modelo de datos requerido para cada motor de modelado de datos: *Cassandra*, *Neo4j* o *Mongodb*; pero en general se ha seguido el mismo patrón de limpieza y han recuperado la mayoría de las columnas.

[A. Cassandra](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/cassandra/readme.md#preprocesamiento-e-importaci%C3%B3n-de-datos)

B. [Neo4j](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/neo4j/readme.md#preprocesamiento-e-importaci%C3%B3n-de-datos)

C. [Mongodb](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/mongodb/readme.md#preprocesamiento-e-importaci%C3%B3n-de-datos)

D. [Processorcpp](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/processorcpp/readme.md#preprocesamiento-e-importaci%C3%B3n-de-datos)

A. Cassandra

Cassandra es una base de datos dependiente del caso de uso. En la mayoría de los casos, una simple instancia de MySQL o PostgreSQL haría mejor el trabajo.

La idea principal es que encontremos que Cassandra pueda ofrecer facilidades con respecto a la organización de los **atributos** en nuestro modelo de datos.

El uso de Cassandra tiene muchos pros y contras diferentes, muchos de los cuales dependen de lo que se quiera hacer con él.

**Pros**

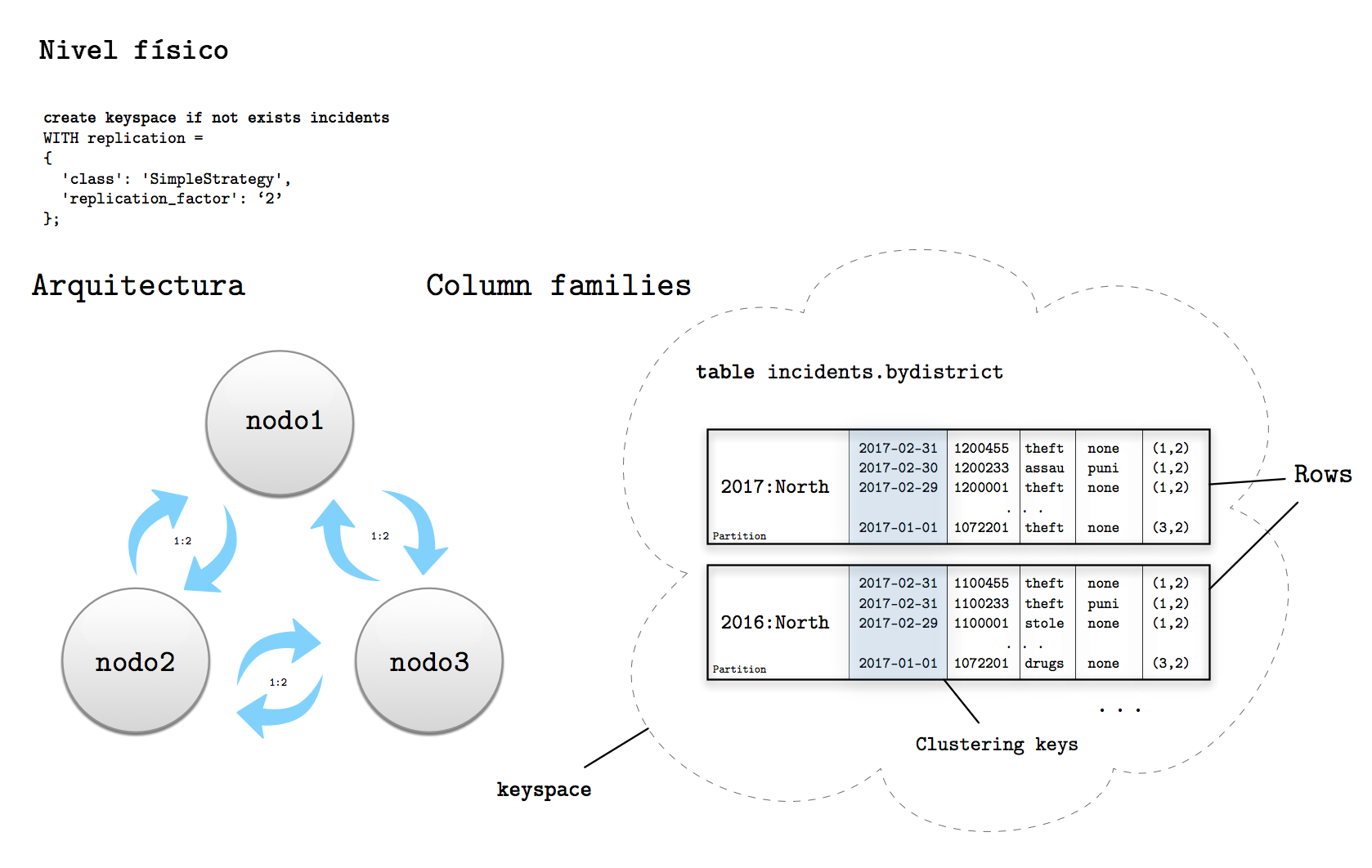
* **Consistencia estable y replicación de datos**. La relación entre la **estructura lógica y la física** permite que la información quede estructurada y organizada entre los nodos, con el fin de optimizar y almacenar la información entre ellos.
* **Redundancia de información**. La información se encuentra organizada entre los nodos de forma redundante a su vez.
* CQL Lenguaje de consulta de Cassandra es una forma bastante familiar para hacer consultas sobre Cassandra. Es un subconjunto de SQL y tiene muchas de las mismas características, haciendo que la transición de un RDBMS basado en SQL a Cassandra sea menos discordante.

**Contras**

* **Sin Consultas Ad-Hoc**: La capa de almacenamiento de datos de Cassandra es básicamente un sistema de almacenamiento de clave-valor. Esto significa que debe "modelar" sus datos en torno a las consultas que desea que surjan, en lugar de en torno a la estructura de los datos en sí. Esto puede llevar a almacenar los datos varias veces de diferentes maneras para poder satisfacer los requisitos de su aplicación.
* **Sin agregaciones**, aunque en esta versión si las usaremos (3.11): las versiones más nuevas de Cassandra tendrán soporte limitado para agregaciones con una sola partición. **Esto es de uso muy limitado. Debido a que Cassandra es una tienda de valores clave, hacer cosas como SUM, MIN, MAX, AVG y otras agregaciones requieren una gran cantidad de recursos si es posible**. Si hacer un análisis ad-hoc es un requisito para su aplicación, entonces Cassandra puede no ser para usted.
* CQL: Es fácil para alguien que proviene de SQL confundirse acerca de qué es o no compatible. Esto significa una frustración adicional (costos de lectura) para los desarrolladores que no conocen las limitaciones de Cassandra.

### **A.1. Estructura de la información (Nivel lógico vs Nivel físico)**

Unas de las características principales de Cassandra es la separación entre nivel físico y lógico.

[](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/docs/cassandra/nivel_fisico.png)

* **Column families** son abstracciones lógicas unitarias que conforman las tablas fisicas de CQL. Es la forma en la que disponemos la información, el esquema de organización de los campos y valores, mediante distintos tipos de columnas.
* **Particiones**. Define la estructura de compactación que engloban parte de la información de una tabla en concreto, y que las filas que la componen tienen en común el mismo criterio lógico dado por una/s clave/s de partición. Las filas que componen cada partición pueden estar ordenadas dependiendo del criterio de ordenación mediante definición de columnas como claves clustering.
* **Rows** Son las unidades de forman las column families y son instancias de las particiones. Una partición puede contener una gran cantidad de rows. Por cada partición, se pueden definir la ordenación de las filas que lo componen.

En nuestro caso, las particiones van a ser tipo **multi-row** ya que contendrán una cantidad masiva de filas ordenadas (por timestamp), y tendrán en común un determinado criterio, dado por una composite key (clave de partición compuesta): por distrito o por zona a parte del año, conformando claves de distinto tipo dependiendo del modelo de consulta que se requeiera por la aplicación:

* Por año: 2017
* Por zona/año: 2017:north
* Por tipo de delito/año: 2017:theft

La definición de las tablas necesarias para las consultas las podemos encontrar bajo el fichero: .

Con respecto a las ordenación de las filas las claves de ordenación o clustering han sido fijadas sobre las columnas claves para seleccionar filas y definir cierto criterio en la realización de cualquier consulta. Generalmente la columna time ha sido seleccionada en todas las opciones y de forma DESCENDENTE, con el fin de que a nivel físico sean ordenadas de más reciente a menos reciente; pues por usabilidad no es cuestionable dicho orden.

A **nivel de arquitectura**, solo mencionar que los datos de las tablas son alojadas como **column families** y dicha información es compartida de forma redundante entre los nodos que compongan la red Cassandra y del grado de replicación de como se configure el keyspace (Véase figura anterior.)

### **Instalación y configuración**

### Requerimientos técnicos

* Java 1.8
* Cassandra 3.11.2
* DevCenter

### Instalación

* Descargar cassandra server

$ wget http://apache.rediris.es/cassandra/3.11.2/apache-cassandra-3.11.2-bin.tar.gz

$ tar -xvf apache-cassandra-3.11.2-bin.tar.gz

* Running cassandra server:

./apache-cassandra-3.11.2/bin/cassandra

* Running CQL shell:

./apache-cassandra-3.11.2/bin/cqlsh

### **Troubleshooting**

DevCenter closes bceause of java virtual machine Edit DevCenter.app/Contents/devcenter.ini and add the line

-vm

/Library/Java/JavaVirtualMachines/jdk1.8.0\_51.jdk/Contents/Home/bin

**A.2 Preprocesamiento e importación de datos**

El mapa de incidencias se puede descargar mediante línea de comandos:

wget -O incidents.raw.tsv \

"https://data.sfgov.org/api/views/tmnf-yvry/rows.tsv?accessType=DOWNLOAD&api\_foundry=true"

La información en CSV (con tabulador como separador) se encuentra estructurada en bruto de la siguiente manera:

IncidntNum Category Descript DayOfWeek Date Time PdDistrict Resolution Address X Y Location PdId

150060275 NON-CRIMINAL LOST PROPERTY Monday 01/19/2015 14:00 MISSION NONE 18TH ST / VALENCIA ST -122.42158168137 37.7617007179518 (37.7617007179518, -122.42158168137) 15006027571000

### **Limpieza de datos**

Una vez creados los esquemas de tablas necesarias para el modelo de datos procedemos al preprocesamiento de los datos descargados desde el portal de información de incidencias para trabajar con datos limpios y en acorde al formato de los atributos que utilizaremos para dichas tablas. Se realizarán los siguientes cambios con respecto a los datos en bruto:

* Formato timestamp para definir el campo 'time', a partir de las columnas 'Date' y 'Time' ('01/19/2015' y '14:00') para obtener '2015-01-19 14:00:00'.
* Campos de agrupación relacionados con periodos de tiempo: year, month, day, hour, basados en los campos anteriores. Los cuales serán necesarios para las particiones de datos por año y/o búsqueda por horas.

Para **descargar y realizar** este limpiado de datos in-situ lo podemos realizar desde la línea de comandos de la siguiente manera:

wget -O- "https://data.sfgov.org/api/views/tmnf-yvry/rows.tsv?accessType=DOWNLOAD&api\_foundry=true" |tail -n +2 | tr '\t' ';' | sed -E 's/([0-9]+)\/([0-9]+)\/([0-9]+);([0-9]+):([0-9]+)/\3-\1-\2 \4:\5:00;\2;\3;\1;\4/g' > incidents.raw\_data.csv

o una vez descargado el dataset en 'incidents.raw.tsv' con las 2 millones de entradas, realizamos el mismo tipo de procesamiento:

$ nice cat incidents.raw.tsv |wc -l

2185964

$ nice cat incidents.raw.tsv |tail -n +2 | tr '\t' ';' \

| sed -E 's/([0-9]+)\/([0-9]+)\/([0-9]+);([0-9]+):([0-9]+)/\3-\1-\2 \4:\5:00;\2;\3;\1;\4/g' \

> incidents.dataset.csv

Una muestra del resultado la podemos encontrar en el fichero [incidents.dataset.sample.100.tsv](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/dataset/incidents.dataset.sample.100.tsv) generado a partir del original [incidents.raw.sample.100.tsv](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/dataset/incidents.raw.sample.100.tsv).

150060275;NON-CRIMINAL;LOST PROPERTY;Monday;2015-01-19 14:00:00;19;2015;01;14;MISSION;NONE;18TH ST / VALENCIA ST;-122.42158168137;37.7617007179518;(37.7617007179518, -122.42158168137);15006027571000

150098210;ROBBERY;ROBBERY, BODILY FORCE;Sunday;2015-02-01 15:45:00;01;2015;02;15;TENDERLOIN;NONE;300 Block of LEAVENWORTH ST;-122.414406029855;37.7841907151119;(37.7841907151119, -122.414406029855);15009821003074

A partir de aquí se procede al volcado de datos a una tabla de índole general

### **Importación de datos**

Para realizar el volcado e importación de datos es imprescindible qye se haya creado: el keyspace de trabajo y los esquemas y tablas físicas sobre la consola CSQL o desde el DevCenter. El comando utilizado para realizar el volcado se denomina COPY. Es un comando bastante simple de usar, poco flexible y dúctil para poder realizar transformaciones o selecciones específicas.

COPY incidents.overall(incidentId, category, description, dayoftheweek, time, day,year,month, hour, district, resolution, address,x,y,location, subid)

FROM 'dataset/incidents.dataset.sample.100.csv'

WITH DELIMITER=';' and HEADER=false and DATETIMEFORMAT='%Y-%m-%d %H:%M:%S';

Para la importación de datos a los otros esquemas se sigue un proceso de importación en cadena, de manera que entre las tablas con la información volcada son tomadas como base para realizar una exportación-importación de campos selectivos.

Tabla 1 -> Export -> solo los campos necesarios para la Tabla2 (script cql temporal) <- Import <- Tabla 2

### **A.3 Estructura de datos**

### **Diseño**

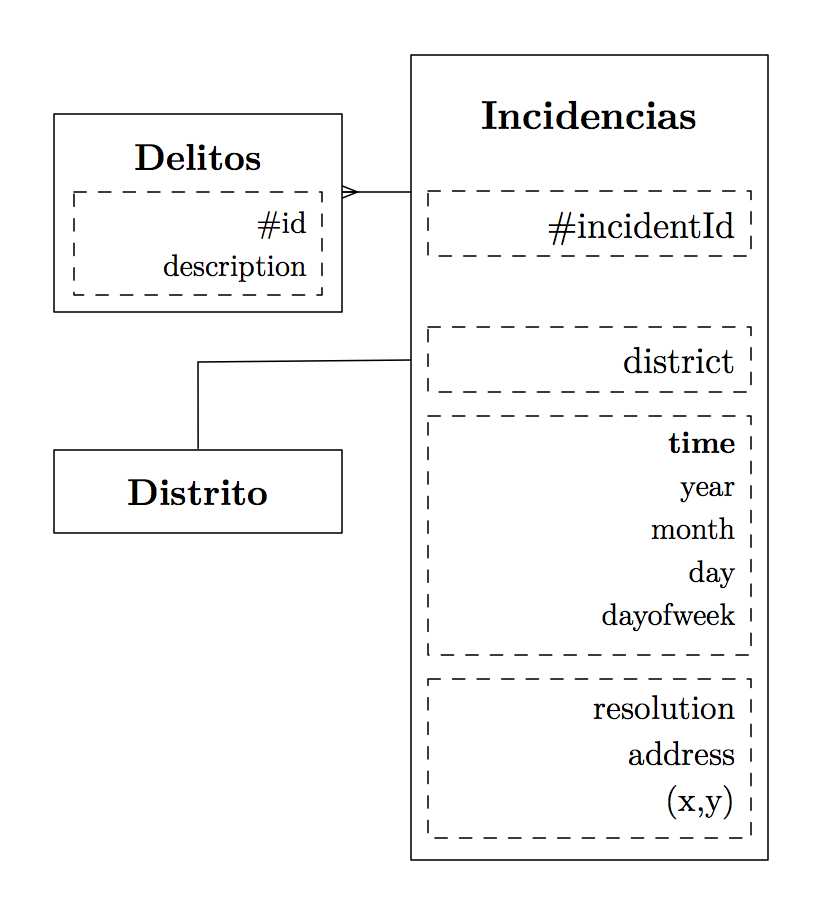
En Cassandra, el diseño y definición de un modelo de datos se procede una vez conocidas las metas y sentencias necesarias para la visualización final de la información a analizar. Se sugiere seguir una series de pautas para conseguir un modelado de datos idóneo para el análisis y el procesamiento masivo de datos.

Hay dos metas importantes a tener en cuenta para el modelado de datos en Cassandra:

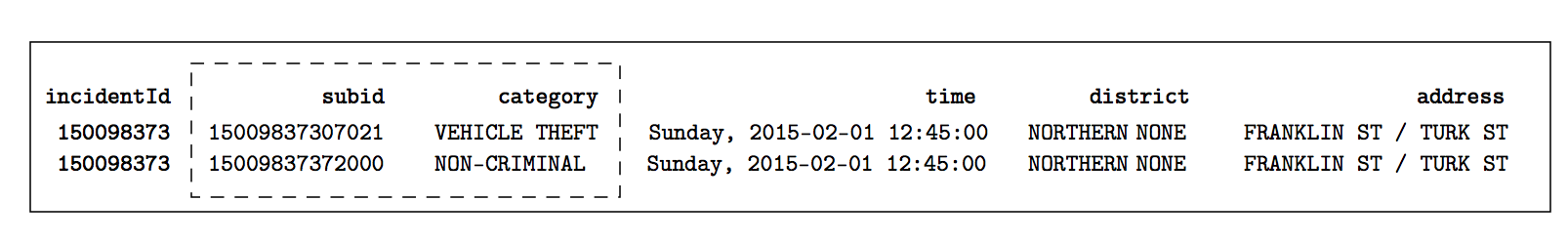
* Definir un procedimiento para repartir la información de forma equitativa y horizontal en los clusters definidos.
* Minimizar el numero de lectura de particiones.

Análisis de requisitos. Se requiere obtener y visualizar la actividad criminal para periodos de tiempo de ciertas zonasde la ciudad.

Identificar entidades y relaciones. Cada línea del documento contiene información de la relación incidencia-delito. Cada incidencia se puede desglosar en varios causas delictivas o delitos. La relación natural entre las entidades mencionadas queda representada en el siguiente diagrama:

[](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/docs/modelodatos.png)

Dado que el modelo de datos se hace en acorde a las sentencias y no sobre las entidades o relaciones, no se requiere de un diseño alejado de la representación natural. En tal caso, se tendrá en cuenta la representación actual de las incidencias pero se hará para distintos escenarios (esquemas) en acorde a cada sentencia requerida.

[](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/docs/modelodatosraw.png)

### **Identificación de consultas**

Identificar sentencias. La mejor manera de particionar las lecturas es modelar los datos en acorde a las sentencias, que son dadas por los requisitos. De primera instancia hay que considerar los siguientes puntos antes de realizar la definición de las sentencias:

* Agrupación por un atributo: por distrito, por tipo de actividad criminal
* Ordenación por un atributo: por tiempo
* Filtro basado en un conjunto de condiciones.

A partir de aquí, desde otra perspectiva de alto nivel, nuestro objetivo consistirá en clasificar las requeridas incidencias y recuperar la información de la actividad criminal (cantidad/frecuencia), en periodos de tiempo, de forma ordenada, y dada ciertas condiciones definidas: por distrito o tipo de delito.

La visualización de la actividad criminal se consigue mediante la representación de las incidencias producidas para cada atributo: periodo de tiempo, por cada zona ó tipo de delito. Para una perspectiva de análisis un poco más precisa, se necesitaría representar la cuantificación mediante uso de proporciones o frecuencias para un determinado grupo de atributos o condiciones.

* Frecuencia criminal por periodos de tiempo: horas o día de la semana para cierto año.
* Frecuencia criminal por cada zona.
* Proporción de la incidencia de diferentes actividades delíctivas.

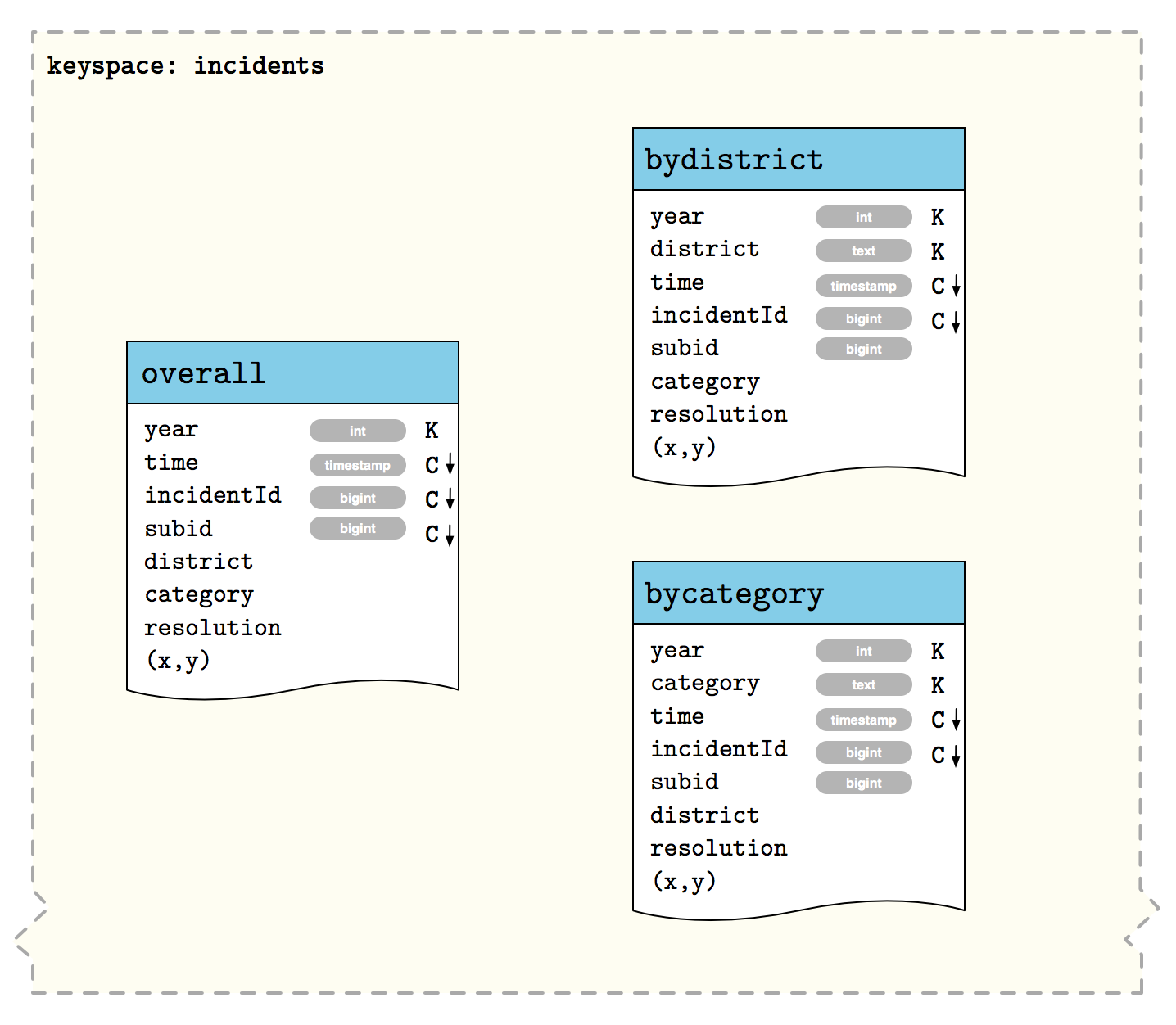
Especificación del esquema. La especificación del esquema vendrá determinado por la consulta o tipo de consulta que la requiera. Todos los esquemas se pueden generar obtenidos desde [schema.cql](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/cassandra/cql/schema.cql)

### A.4. Modelado de datos

**Modelo de datos físico**. Ahora, pasemos nuestra atención en las tablas de incidencias diseñadas. Nuestro modelo lógico contiene tres tablas desnormalizadas para admitir consultas de incidencias por año, zona, tipo de delito y fecha. A medida que trabajamos para implementar estos diseños por consultas, querremos considerar si debemos realizar la administración con la desnormalización de forma manual o usar la capacidad de extrapolar las consultas dentro de vistas en Cassandra, con ayuda de las consultas de agregación.

El diseño que se muestra para el espacio de incidencias en la siguiente figura utiliza ambos enfoques. Elegimos implementar **incidents.overall** y **incidents.bydistrict** y **incidents.bycategory** como **tablas regulares**.

El razonamiento detrás de esta elección de diseño momentáneamente, es que gracias a las claves de partición compuestas entre zona o tipo de delito con el año, podemos dividir y dimensionar mejor las consultas sobre todo cuando existe una base de información masiva.

[](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/docs/cassandra/model_tablas.png)

**A.5 Representación de la información: consultas CQL**

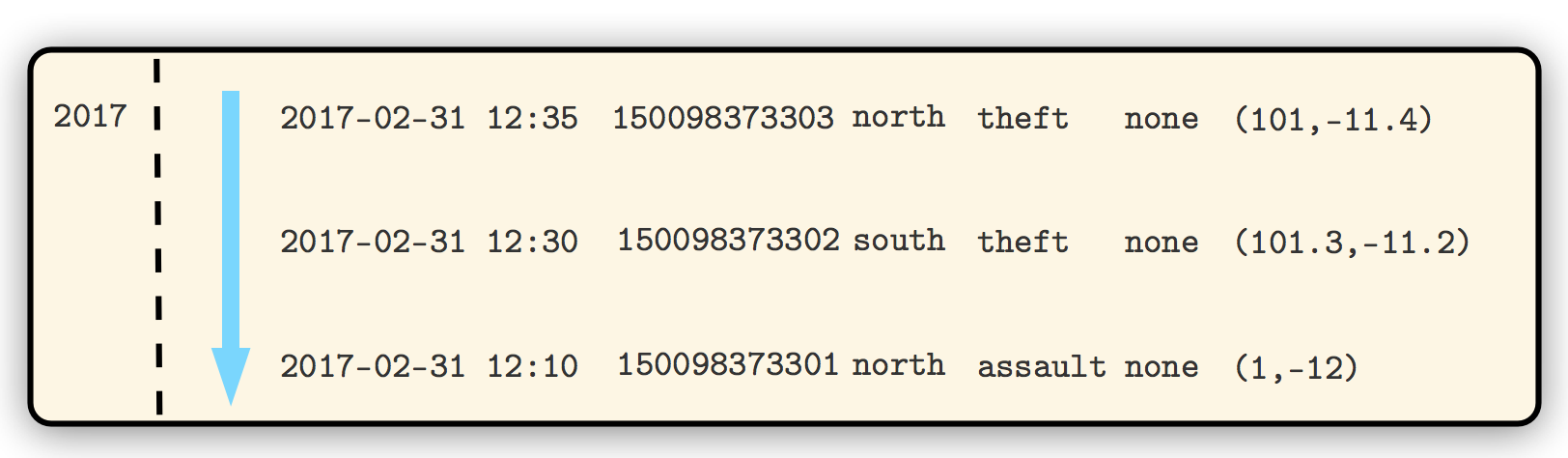
Todas las consultas se pueden encuentrar en el script CQL [cql/queries.cql](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/cassandra/cql/queries.cql), pero para realizarlas es necesario crear el esquema de la estructura en la base de datos de Cassandra desde la consola a partir de los esquemas en [cql/schema.cql](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/cassandra/cql/schema.cql).

1. Crear keyspace
2. Crear tablas e volcar los dataset mediante el comando COPY

### **Actividad criminal para un periodo de tiempo**

La actividad es ofrecida por la tabla: incidents.overall, con la estructura:

|  |  |
| --- | --- |
| Partition keys | year |
| Clustering keys | time,incidentid |

[](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/docs/cassandra/diagrama_table1.png)

La información se encuentra particionada por una clave primaria compuesta de las claves: de partición año; pero al añadir el campo time como clave de clusterización podemos realizar una búsqueda por periodo. No se podría considerar una consulta muy eficiente ya que no se aprovecha las ventajas de particionamiento con respecto a la condición de búsqueda.

CREATE TABLE incidents.overall (

year int,

time timestamp,

...

PRIMARY KEY ((year), time, incidentId, subid)

) WITH CLUSTERING ORDER BY (time DESC, incident ASC, subId ASC);

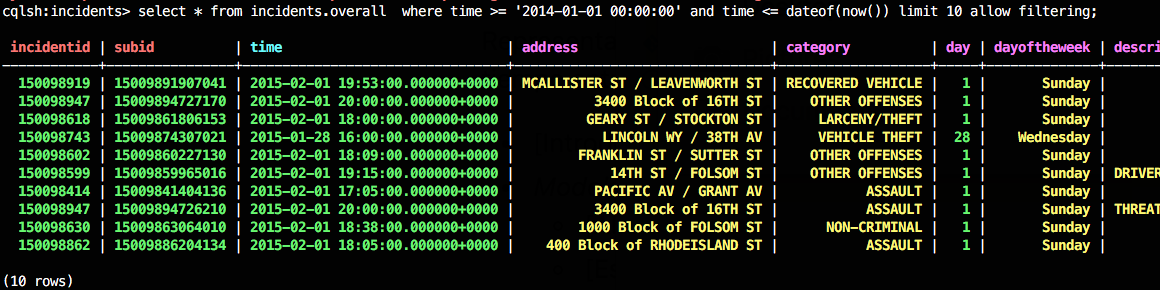
* Obtener toda las incidencias para un periodo de tiempo.

select \* from incidents.overall

where time >= '2014-01-01 00:00:00' and time <= dateof(now())

and yearh = 2017

allow filtering;

[](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/docs/cassandra/queries/query_overall_periodtime.png)

* Actividad criminal por zona

Para esta sentencia si se realiza una partición de datos adecuada, con respecto a la zona y año: district:year.

La actividad es ofrecida por la tabla: incidents.overall, con la estructura:

|  |  |
| --- | --- |
| Partition keys | year, district |
| Clustering keys | time,... |

* Obtener información de incidencias por zonas (para un determinado año)

select district, year, incidentid, category, time, location

from incidents.bydistrict

where year = ?

[](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/docs/cassandra/queries/query_bydistrict_incidents_year_2015.png)

Si queremos añadir condicion de periodo de tiempo, necesitamos añadir filtering:

select district, year, incidentid, category, time, location

from incidents.bydistrict

where year = 2015 and time >= '2015-02-01 00:00:00' and time <= dateof(now())

allow filtering;

* Actividad criminal por tipo de delito

Para esta sentencia si se realiza una partición de datos adecuada, con respecto al tipo de incidencia y año: category:year.

|  |  |
| --- | --- |
| Partition keys | year, category |
| Clustering keys | time,... |

* Obtener información de incidencias por categorias (para un determinado año)

select category, year, incidentid, category, time, location

from incidents.bycategory

where year = ?

[](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/docs/cassandra/queries/query_bydistrict_incidents_year_2015.png)

* Nº incidencias agrupadas por año

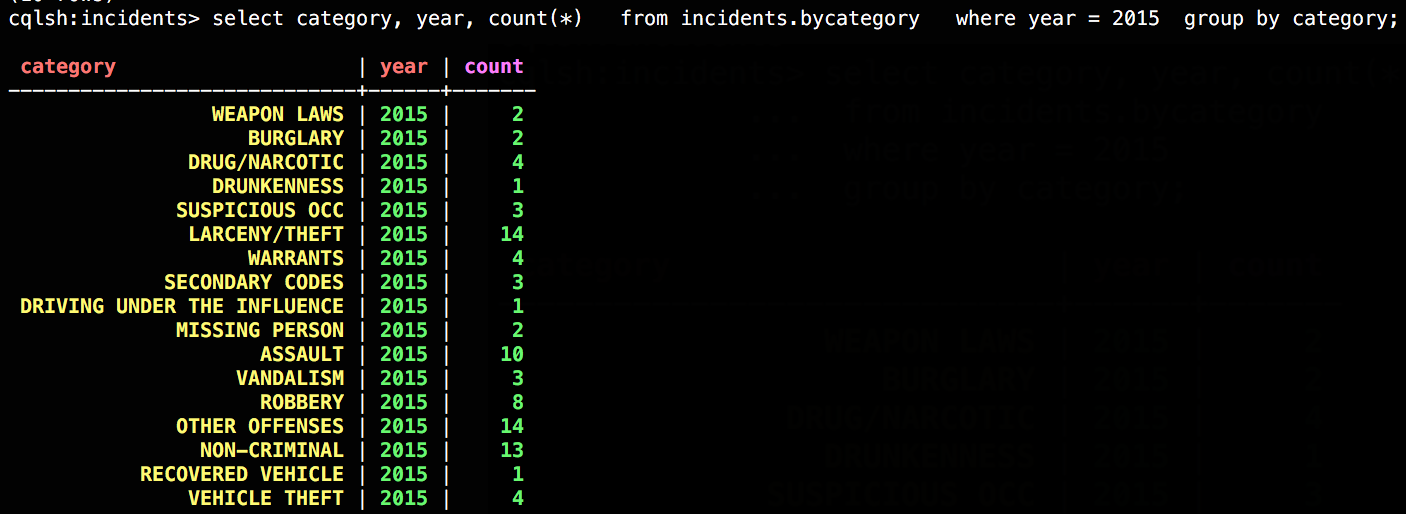
Para obtener número de incidencias producidas de cierta categoria (por un determinado año). Sin añadir filtro, más eficiente.

select category, year, count(\*)

from incidents.bycategory

where year = ?

group by category;

[](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/docs/cassandra/queries/query_bycategory_groupby_withoutfilter2.png)

* Nº incidencias agrupadas por año

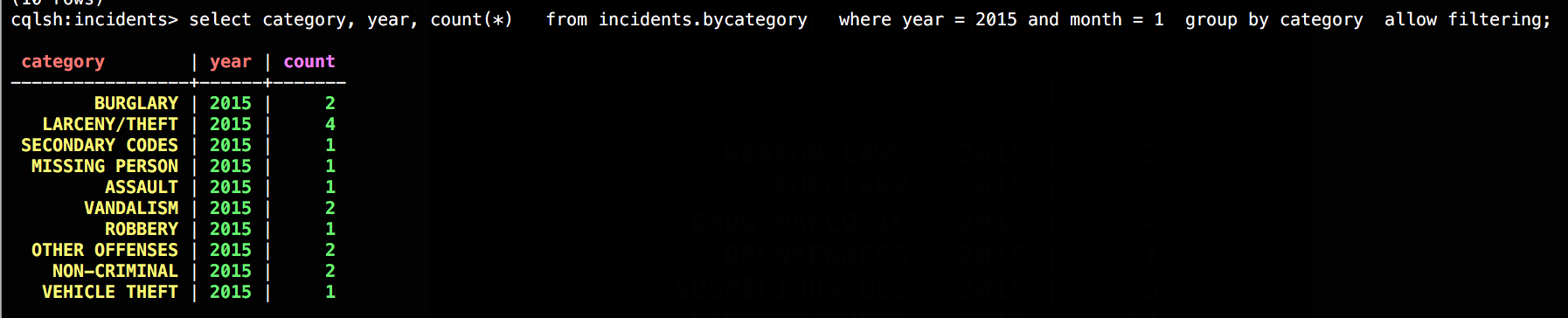
select category, year, count(\*)

from incidents.bycategory

where year = 2015 and month = 1

group by category

allow filtering;

[](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/docs/cassandra/queries/query_bycategory_groupby_with_filter2.png)

B. Neo4j

Representación y modelado de datos de actividades criminales con respecto a localizaciones o por periodos de tiempo entre los años 2003 y 2018 mediante Neo4j.

**B.1 Instalación y configuración**

Requerimientos técnicos

* Neo4j Desktop/Server
* R and RStudio

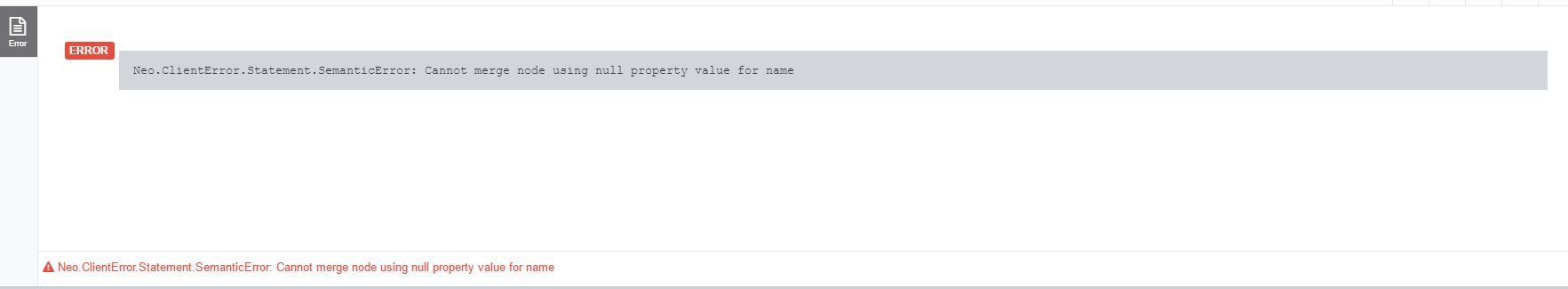
Instalación

* Download Neo4j Desktop
* Download R
* Download RStudio
* Run Neo4jDesktop
* Run RSTudio

**B.2 Preprocesamiento e importación de datos**

### Limpieza y preprocesamiento de datos

Preprocesamiento mediante R. Aunque en primer lugar no era necesario, tras causar cierto error en **Cypher**relacionado con la ausencia de una propiedad en una operación **MERGE**, se procedió a buscar el motivo de este fallo en el archivo dataset.raw.csv descargado directamente de la fuente.

[](https://camo.githubusercontent.com/f7e1f655c1e9d19663a12e303d519b908938589c/68747470733a2f2f696d6167652e6962622e636f2f627a697148782f6572726f722e6a7067)

El proceso llevado a cabo fue el siguiente:

* En **RStudio** se importó el dataset al completo, reemplazando los valores nulos con NA. Sin embargo, al iterar sobre el dataframe, el número de filas que [complete.cases](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/neo4j) devolvía era el mismo que el número de líneas del dataset original.

# R

SF\_Crime\_Heat\_Map <- read.csv("C:/Users/Julio/Desktop/SF\_Crime\_Heat\_Map.csv")

View(SF\_Crime\_Heat\_Map)

missing <- SF\_Crime\_Heat\_Map[complete.cases(SF\_Crime\_Heat\_Map),]

nrow(SF\_Crime\_Heat\_Map) == nrow(missing)

# [1] TRUE

* Por tanto se decidió exportar estos mismos datos pero aprovechando el tipado que se había producido en la importación para ordenar el dataframe mediante el incidentNum. Este sería exportado a un nuevo archivo dataset.data.csv.

# R

SF\_data <- SF\_Crime\_Heat\_Map[order(SF\_Crime\_Heat\_Map$IncidntNum),]

View(SF\_data)

write.csv(SF\_data,"dataset.data.csv.csv")

Este fichero resultante sería el utilizado para realizar el volcado al modelo de datos.

#### Actualización

Sin embargo, encontramos un problemas con el formato actual. Este consiste en que el campo Date no nos proporciona acceso individual a cada variable (Día, mes y año). Obtener estas dividiendo la columna en una sentencia **cipher** aumenta de forma exponencial el tiempo de carga del CSV a la base de datos, por lo tanto es recomendable realizar esta operación anteriormente y modificar el archivo existente:

# Transformamos las columas del data.frame para poder operar sobre ellas.

SF\_Crime\_Ordered\_Map\_2 <- data.frame(lapply(SF\_Crime\_Ordered\_Map, as.character), stringsAsFactors=FALSE)

# Usamos la librería stringr para mayor facilidad a la hora de separar la columna

install.packages("stringr")

library("stringr")

# Almacenamos los tres vectores resultantes

fechaSeparada <- str\_split\_fixed(SF\_Crime\_Ordered\_Map\_2$Date,'/',3)

# Añadimos al data.frame original las nuevas columnas y lo exportamos

SF\_Crime\_Ordered\_Map$Day = fechaSeparada[,2]

SF\_Crime\_Ordered\_Map$Month <- fechaSeparada[,1]

SF\_Crime\_Ordered\_Map$Year <- fechaSeparada[,3]

write.csv(SF\_Crime\_Ordered\_Map, "dataset.ordered.data.csv", row.names = FALSE)

### **B.3 Importación de datos**

Una vez obtenido el fichero preformateado dataset.ordered.data.csv se procede a la importación desde el motor de Neo4j. Es necesario utilizar la sentencia LOAD CSV.

Pero a continuación describimos el proceso que llevamos a cabo para realizar la importación de datos y la cadena de problemas con los que nos encontramos hasta la solución final.

Importamos con la sentencia LOAD CSV básica en su defecto.

LOAD CSV WITH HEADERS FROM 'file:///dataset.ordered.data.csv' as line return line

Para importar los datos de CSV a la base de datos, sin embargo esto no genera nodos ni relaciones.

Para realizar la importación de datos tuvimos que seguir una serie de pasos hasta llegar con la solución final, a partir del data set dataset.ordered.data.csv:

#### Primera solución

Esta fue la primera sentencia **Cypher** utilizada para cargar los datos.

* El tag **WITH HEADERS** de la instrucción **LOAD CSV** nos permite referirnos a las columnas por los nombres originales situados al comienzo del csv.
* Donde **MATCH** es similar a **SELECT** y **CREATE** a **INSERT**, respectivamente, **MERGE** es una mezcla entre ambos que busca el elemento especificado, y si no lo encuentra, lo crea, lo que conlleva una búsqueda previa a la potencial inserción.
* Los elementos como nodos y relaciones se especifican con **LABELS** y **Properties**, donde los primeros actuan como tipo principal del elemento y sirven para filtrarlos más fácilmente, y los segundos almacenan datos más específicos por lo que requieren sentencias con **WHERE** para su filtrado.

LOAD CSV WITH HEADERS FROM 'file:///SF\_Crime\_Ordered\_Map.csv' AS line

MERGE (i:INCIDENT { incidentNum:toInt(line.IncidntNum)})

SET ON CREATE i.description = line.Descript

MERGE (c:CATEGORY { name: line.Category})

MERGE (f:DATE { date: line.Date, diaSemana: line.DayOfWeek})

MERGE (r:RESOLUTION { name: line.Resolution})

MERGE (d:DISTRICT { name:line.PdDistrict})

CREATE (i)-[t:TYPE]->(c),

(i)-[p:PLACE { address:line.Address, x:line.X, y:line.Y}]->(d),

(i)-[ti:TIME {time: line.Time}]->(f),

(i)-[s:STATUS]->(r);

Problema de esta sentencia

* Las sentencias MERGE buscan nodos con las mismas propiedades y labels antes de insertar lo que ralentiza la carga.
* La sentencia completa intenta realizarse con la memoria RAM actual, con 2M de datos por procesar.

Esto resultaba en errores de memoria o tiempos demasiado largos para la importación (una hora y media).

Solución

* Se añadió la instrucción USING PERIODIC COMMIT la cual almacena los cambios realizados cada 1000 líneas, por defecto.
* Se añadió la instrucción ON CREATE SET que condiciona una propiedad de un nodo a su creación en Merge, mejorando la eficiencia.
* Se incluyeron índices antes de la carga para cada nodo, mejorando la velocidad de búsqueda:
* CREATE INDEX ON :INCIDENT(incidentNum);
* CREATE INDEX ON :CATEGORY(name);
* CREATE INDEX ON :DATE(day);
* CREATE INDEX ON :DATE(month);
* CREATE INDEX ON :DATE(year);
* CREATE INDEX ON :RESOLUTION(name);
* CREATE INDEX ON :DISTRICT(name);
  + Se aumentó la memoria modificando el archivo neo4j.conf con estos valores:
  + dbms.memory.heap.initial\_size=2G
  + dbms.memory.heap.max\_size=4G
  + dbms.memory.pagecache.size=4G

Lo cual aumentaba la memoría máxima utilizable por **Neo4j** hasta 8Gb de RAM.

Solución final Por tanto la sentencia final fue esta:

USING PERIODIC COMMIT 5000

LOAD CSV WITH HEADERS FROM 'file:///SF\_Crime\_Ordered\_Map.csv' AS line

MERGE (i:INCIDENT { incidentNum:toInt(line.IncidntNum)})

SET ON CREATE i.description = line.Descript

MERGE (c:CATEGORY { name: line.Category})

MERGE (f:DATE { date: line.Date})

ON CREATE SET f.dayofweek = line.DayOfWeek

MERGE (r:RESOLUTION { name: line.Resolution})

MERGE (d:DISTRICT { name:line.PdDistrict})

CREATE (i)-[t:TYPE]->(c),

(i)-[p:PLACE { address:line.Address, x:line.X, y:line.Y}]->(d),

(i)-[ti:TIME {time: line.Time}]->(f),

(i)-[s:STATUS]->(r);

Que importaba los datos, creaba los nodos y sus relaciones en un tiempo de **3m20s**.

#### Mejora definitiva

Para cargar los nuevos datos de día, mes y año, la query tuvo que cambiar a:

USING PERIODIC COMMIT 5000

LOAD CSV WITH HEADERS FROM 'file:///SF\_Crime\_Ordered\_Date\_Map.csv' AS line

MERGE (i:INCIDENT { incidentNum:toInt(line.IncidntNum)})

SET ON CREATE i.description=line.Descript

MERGE (c:CATEGORY { name: line.Category})

MERGE (f:DATE { day:toInteger(line.Day), month:toInteger(line.Month),year:toInteger(line.Year)})

ON CREATE SET f.dayofweek = line.DayOfWeek

MERGE (r:RESOLUTION { name: line.Resolution})

MERGE (d:DISTRICT { name:line.PdDistrict})

CREATE (i)-[t:TYPE]->(c),

(i)-[p:PLACE { address:line.Address, x:line.X, y:line.Y}]->(d),

(i)-[ti:TIME {time: line.Time}]->(f),

(i)-[s:STATUS]->(r);

La cual es ciertamente más lenta, debido a que se han añadido tipados en **:INCIDENT** y **:DATE**, además de que el **MERGE** realizado en este segundo tipo de nodo debe comparar dos variables adicionales para no causar duplicidad, aumentando la duración de la carga a **12m49s**.

¿Motivo principal?. La subida de datos a servidor es algo que sólo se debe realizar una vez, por tanto es absolutamente más eficiente que tarde más este paso, que ralentizar las búsquedas teniendo que filtrar la fecha con comparadores de cadenas de caracteres en cada una.

#### Configuración necesaria antes del import

Fue necesario tener en cuenta una serie de requisitos de configuracioón, por lo que también es importante recordar que el fichero ha de estar en la carpeta:

Application\neo4jDatabases\database-(numero)\installation-3.3.3\import

Pero se puede cambiar modificando el archivo neo4j.conf en la línea:

dbms.directories.import=import

Siendo "import" el directorio por defecto.

También se puede acceder a esta configuración desde la aplicación Desktop mediante:

My project->(Nombre de BD)->Settings

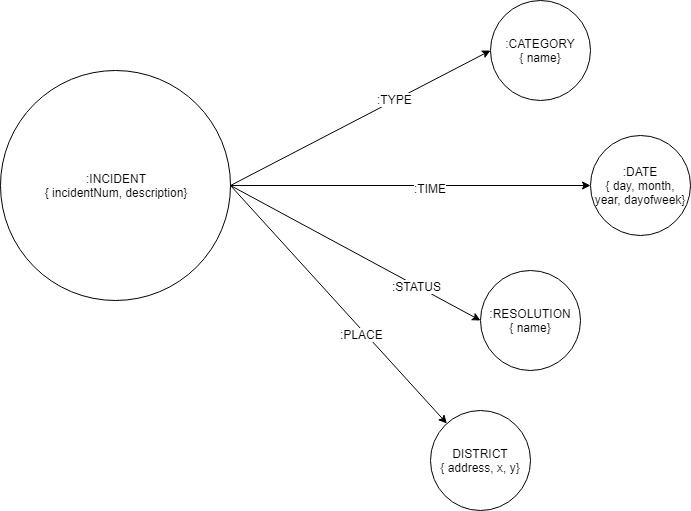
Que provee de un editor propio.

Estructura de datos

Dada la estructura del archivo .CSV proporcionado, titulado con los siguientes **headers**:

IncidntNum, Category, Descript, DayOfWeek, Date, Time, PdDistrict, Resolution, Address, X, Y, Location.

Se ha decidido realizar este diseño de nodos y relaciones:

[](https://camo.githubusercontent.com/c0aa6d56d9091360794caf24a23066d3b616a9ce/68747470733a2f2f696d6167652e6962622e636f2f6e63724263782f4e6f646f732e6a7067)

Dado que, a excepción de los incidentes, un gran número de elementos estaban duplicados, se ha procedido a crear sus propios nodos y a evitar su inserción múltiple. Los datos que podían causar la aparición doble de un mismo valor, como un distrito con mismo nombre pero coordenadas X e Y diferentes a las de otro mismo distrito, se han añadido a las relaciones entre los nodos de incidentes y el resto. Esto aumenta la eficiencia y rendimiento final tanto de la inserción como de las búsquedas futuras.

Formato de fechas.Mientras que en el archivo sólo se guarda la fecha como cadena de caracteres en la columna Date, en el esquema aparecen tres campos: Day,Month y Year, los cuales serán obtenidos durante el apartado de preprocesamiento [aquí](https://github.com/orial/grupo_Spark/blob/master/Procesamiento/neo4j/readme.md#limpieza-y-preprocesamiento-de-dato).

**B.4 Consultas**

### **Consultas comunes**

* Actividad criminal para un periodo de tiempo En este caso se obtendría el número de incidencias por día. **Count**es una función agregativa que reúne todas las entradas de un mismo tipo.

MATCH (n:INCIDENT)-[r:TIME]->(d:DATE) return d,count(n)

* Nº incidencias agrupadas por día

MATCH (n:INCIDENT)-->(s:DATE) return s,count(n); // Incidentes por día

* Nº incidencias agrupadas por año

MATCH (n:INCIDENT)-->(s:DATE) return s.year,count(n) // Incidentes por año

* Actividad criminal por zona

MATCH (n:INCIDENT)-[r:PLACE]->(d:DISTRICT) return distinct d,count(n)

* Actividad criminal por tipo de delito
* Nº incidencias agrupadas por día

MATCH (c:CATEGORY)<--(n:INCIDENT)-->(s:DATE) return s.year,c.name,count(n) order by c.name // Incidentes y su tipo, por año

* Nº incidencias agrupadas por año

MATCH (c:CATEGORY)<--(n:INCIDENT)-->(s:DATE) return s,c.name,count(n) order by c.name // Incidentes y su tipo, por día

* Actividad criminal por dia de la semana

MATCH (n:INCIDENT)-->(s:DATE) return s.dayofweek,s.year,count(n) order by s.dayofweek,s.year // Totales de incidentes los días de la semana de cada año

**Bibliografia**