# Fundamentos teóricos / prácticos (II)

[Fundamentos teóricos / prácticos (II) 1](#_Toc516770979)

[1. Acceso a la información: Conectores 3](#_Toc516770980)

[¿Para qué sirven, cómo los usaremos? 3](#_Toc516770981)

[Comparativa de los drivers 3](#_Toc516770982)

[Requerimientos comunes de los conectores 4](#_Toc516770983)

[1.1 Cassandra 4](#_Toc516770984)

[Introducción 4](#_Toc516770985)

[Falta una mini introducción aquí de la librería 4](#_Toc516770986)

[Instalación y configuración 4](#_Toc516770987)

[Librerías adicionales 4](#_Toc516770988)

[Configuración y creación de la sesión 4](#_Toc516770989)

[Caso de uso de sesión: ejemplo de consulta 5](#_Toc516770990)

[1.2 Mongodb 5](#_Toc516770991)

[Introducción 5](#_Toc516770992)

[Instalación y configuración 5](#_Toc516770993)

[Librerías adicionales 6](#_Toc516770994)

[Configuración y creación de la sesión 6](#_Toc516770995)

[Caso de uso de sesión: ejemplo de consulta 6](#_Toc516770996)

[1.3 Neo4j 7](#_Toc516770997)

[Introducción 7](#_Toc516770998)

[Instalación y configuración 7](#_Toc516770999)

[Librerías adicionales 7](#_Toc516771000)

[Configuración y creación de la sesión 7](#_Toc516771001)

[Caso de uso de sesión: ejemplo de consulta 8](#_Toc516771002)

[2. Representación de datos 8](#_Toc516771003)

[2.1 Vistas 8](#_Toc516771004)

[Cassandra 8](#_Toc516771005)

[Mongodb 8](#_Toc516771006)

[Neo4j 8](#_Toc516771007)

[2.2 Gráficos 8](#_Toc516771008)

[4. Bibliografía 8](#_Toc516771009)

Acceso a la información: Conectores

#### ¿Para qué sirven, cómo los usaremos?

Actualmente, toda estructura de almacenamiento de datos posee algún tipo de software o interfaz de usuario oficial para el mantenimiento y gestión de la información que posee.

Esta aplicación, por lo general, está construida bajo uno de los lenguajes de programación comunes ¿Cómo consigue dicho lenguaje conectarse con la base de datos e interactuar con esta? Mediante el uso de driver y conectores. Hoy en día, el término “driver”, en este contexto, quiere referirse a ambas funcionalidades de conexión e interacción, mientras que conector es el módulo software que permite interacción. Esto implica que un conector está implícitamente contenido en un driver de bases de datos.

Dichos drivers nos permitirán por tanto recuperar la información almacenada mediante el lenguaje de consulta característico de cada sistema. Tener acceso a las funciones de los conectores nos ofrece la posibilidad de diseñar funciones auxiliares basadas en estas, las cuales estructurarán y recuperarán los datos de manera que resulten útiles para el ámbito de la aplicación, dando lugar a nuestros conectores personalizados.

#### Comparativa de los drivers

|  |  |  |
| --- | --- | --- |
|  | Ventajas | Desventajas |
| Cassandra | Permite balanceo de carga entre los nodos disponibles.  Pool de threads para conexiones.  Permite añadir una capa de seguridad.  Conexión única con acceso al clúster de nodos en su totalidad. | <a rellenar> |
| Mongodb | Capacidad de manejo de datos en formatos de documentos, con capacidad dinámica.  Capacidad de transformación de tipos de datos, sobre todo fechas.  Permite añadir una capa de seguridad.  Añade geo espacialidad para los datos relacionados con las coordenadas. | <a rellenar> |
| Neo4j | Permite cambios de configuración de usuario desde el conector.  Pool de conexiones que se sirven a las sesiones.  Posee comunicación mediante TLS y certificados.  Transacciones con simples o auto-commit, y explícitas, con capacidad de auto-reintento. | Driver inmutable, necesidad de crearlo de nuevo cuando hay cambios en la conexión. |

#### Requerimientos comunes de los conectores

* Python 3.5
* Librerías de estructuración de datos: *numpy*, *pandas, …*
* Para la representación gráfica: *matplotlib*, *seaborn, tqdm, …*

%config IPCompleter.greedy=**True**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

### Cassandra

#### Introducción

### Falta una mini introducción aquí de la librería

***cassandra.cluster*** - Contiene una clase principal denominada ***Cluster*** que se conecta a un clúster de Cassandra estableciendo una conexión encapsulada en el objeto *Session*. Se pueden añadir ciertas configuraciones en acorde a la arquitectura definida o a la forma de conexión.

#### Instalación y configuración

Configuración e instalación del conector / compatibilidad con Django

pip install cassandra-driver

pip install django-cassandra-engine

MacOS

brew install libev

#### Librerías adicionales

Comunes: panda, numpy.

#### Configuración y creación de la sesión

Para permitir realizar conexiones y consultas de forma genérica hemos implementado un conjunto de funciones auxiliares.

* La función **get\_session** encapsula la conexión sobre nuestro "keyspace" (equivalente a un objeto de base de datos en SQL) generando el objeto Session característico del conector de Cassandra.
* La función q es una abreviatura de “query” y permite acondicionar la realización de consultas y parametrizar valores sobre ellas. El resultado es un objeto tipo **data frame**, por lo cual estaría totalmente adaptado al entorno y sin necesidad de ser característico para el tipo de conector o base de datos.

**def** get\_session(keyspace):

*"""Obtiene el conector con la sesión actual al keyspace indicado."""*

**return** (Cluster(['127.0.0.1']).connect(keyspace))

**def** q(session, query, \*\*kwargs):

*"""Función auxiliar para encapsular las queries producidas por cassandra en formato dataframe."""*

**return** pd.DataFrame([row **for** row **in** session.execute(query.format(\*\*kwargs))])

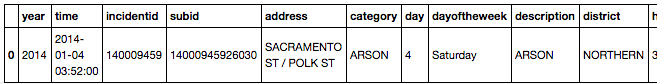
conn = get\_session("incidents")

#### Caso de uso de sesión: ejemplo de consulta

De esta manera podemos obtener una sesión para la realización de las consultas:

**def** getIncidents(session, limit = 100):

**return** q(session, "select \* from incidents.overall limit **{limit}**",limit=limit)



### Mongodb

#### Introducción

El paquete ***pymongo*** contiene herramientas para trabajar desde Python con Mongodb. La estructura de los datos en documentos tipo *JSON* con un esquema dinámico llamado *BSON*, lo que implica que no existe un esquema predefinido. Los elementos de los datos se denominan documentos y se guardan en colecciones.

**pymongo** – Mediante su clase principal, **MongoClient**, da acceso a una instancia de MongoDB, un conjunto de replicaciones o un set de “*mongoses*”. Dicho objeto es seguro sobre hilo y da acceso a un pool de conexiones. Soporta múltiples enlaces de mongoDB y provee de las configuraciones de cliente necesarias para establecer conexiones persistentes en segundo plano, además de poder recuperar datos en una clase de documento preestablecida.

#### Instalación y configuración

Instalación mediante pip

pip install pymongo

#### Librerías adicionales

Comunes: *panda*, *numpy*.

Específicas: *json*, *prettypandas*.

#### Configuración y creación de la sesión

Una vez se realiza la conexión con el cliente de Mongo mediante la URL y el puerto, o el enlace específico de MongoDB, es posible acceder a la base de datos y a continuación, recuperar la información.

Las bases de datos tienen nombres únicos, por lo que la librería permite, mediante el cliente, acceder a estas como **propiedades** de un objeto. Dicha propiedad ofrece la posibilidad de adquirir los datos, que están almacenados en forma de documentos. Para ello vuelve a utilizarse una propiedad, en este caso del **objeto de base de datos**, que retorna una colección de dichos documentos.

client = MongoClient('localhost', 27017)

db = client.datascience # Base de datos llamada datascience

collection = db.incidents # Colección de documentos

#### Caso de uso de sesión: ejemplo de consulta

Mediante la función ***aggregation***, se pueden aplicar ciertos comandos sobre la colección que retuvimos en la variable. En este caso, queremos los incidentes cuya categoría sea “robo”, e imprimimos la primera.

pipeline = [

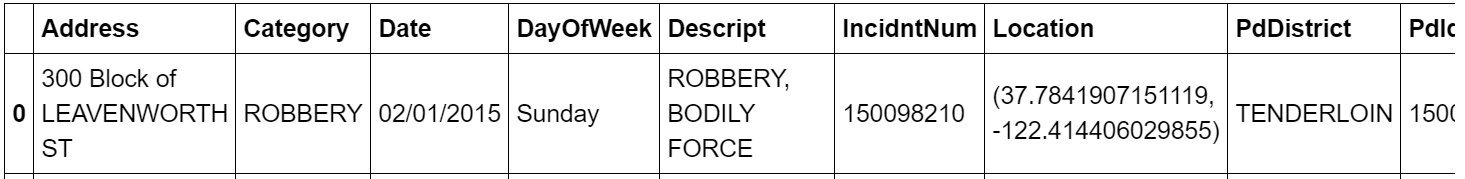
{"$match": {"Category":"ROBBERY"}},

]

aggResult = collection.aggregate(pipeline)

robbery = pd.DataFrame(list(aggResult))

robbery.head(1)



### Neo4j

#### Introducción

El conector ***neo4j-driver*** está basado en el [driver oficial de Neo4J](https://neo4j.com/docs/api/python-driver/current/) para Python. Basándose en este último, se ha creado un **nuevo conector** más avanzado preparado para la recolección específica de datos con el objetivo de ofrecer una mayor transparencia acerca de los objetos con los que se trabaja y mejorar la facilidad de acceso a la información consultada.

**neo4jConnector - GraphDatabase –** La clase neo4JConnector permite la creación de un objeto que encapsula todas las funciones que ofrece el driver de **GraphDatabase**, el cual procede de **neo4j.v1**, la librería que contiene las funciones para realizar la conexión. Se ha elegido encapsular dicho conector para abstraer los detalles sobre el procedimiento de autenticación y así simplificar el proceso. De esta forma se reduce el ámbito a la sección de real importancia: La recuperación de datos en un formato útil.

#### Instalación y configuración

Instalación mediante pip

pip install neo4j-driver

import **neo4jConnector**

#### Librerías adicionales

Comunes: *Pandas*, *matplotlib*.

Personalizadas: *Neo4JConnector*.

#### Configuración y creación de la sesión

Gracias a la nueva clase, no es necesario especificar los parámetros, y con la URI específica de neo4j se reducen a tres argumentos por llamada de conexión.

En cada función de *Neo4JConnector* se utiliza el driver para obtener una sesión y ejecutar las llamadas a la base de datos. La existencia de este objeto nos libera de la necesidad de re-declarar funciones estáticas para las distintas consultas que se deseen hacer, y pone énfasis en qué datos se han de adquirir.

*# Conectamos con la base de datos*

cliente = Neo4JConnector('bolt://localhost:7687','test','4321')

#### Caso de uso de sesión: ejemplo de consulta

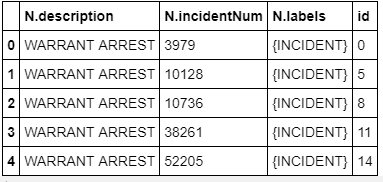
Debido al wrapper, las consultas pueden no requerir de conocer el lenguaje Cypher en el que están basadas estas últimas para Neo4j. Sin embargo, tal y como se especificó en la documentación, es posible realizar consultas especializadas también.

Para este caso, se seleccionan incidentes con una limitación en cantidad sin ningún tipo de filtro, lo que nos devuelve los primeros cinco nodos que poseen dicha etiqueta.

*# Un ejemplo de llamada: Queremos 5 nodos de incidentes*

record = h.select\_limit('INCIDENT',5)

pd.DataFrame([row **for** row **in** record])



## Representación de datos

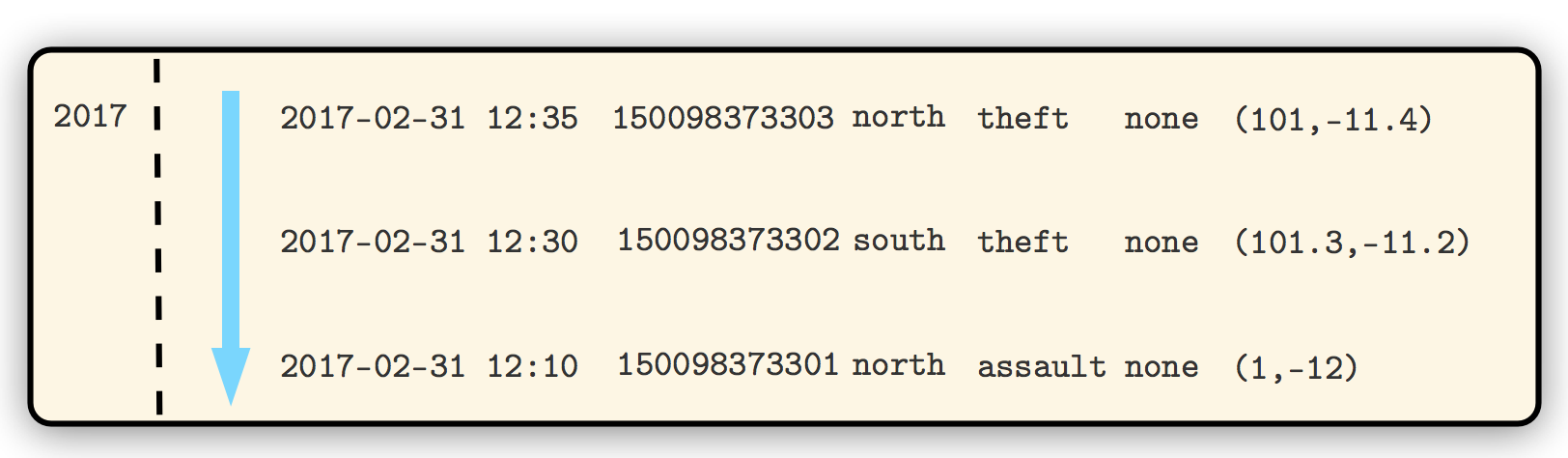
### Consultas

#### Cassandra

##### 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);

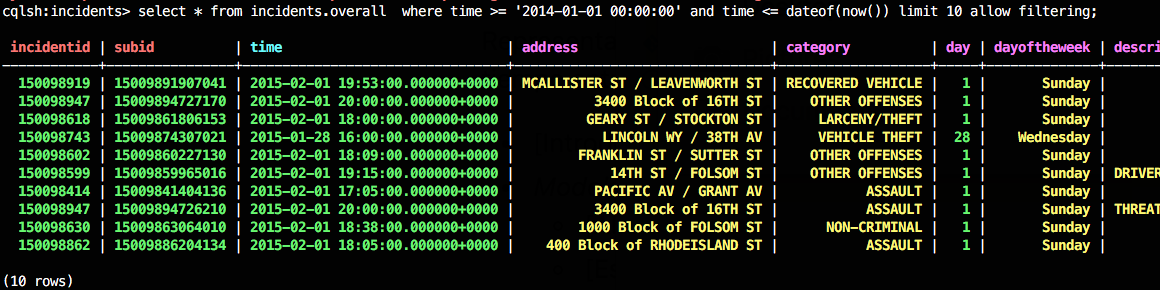
* Obtener toda las incidencias para un periodo de tiempo.

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

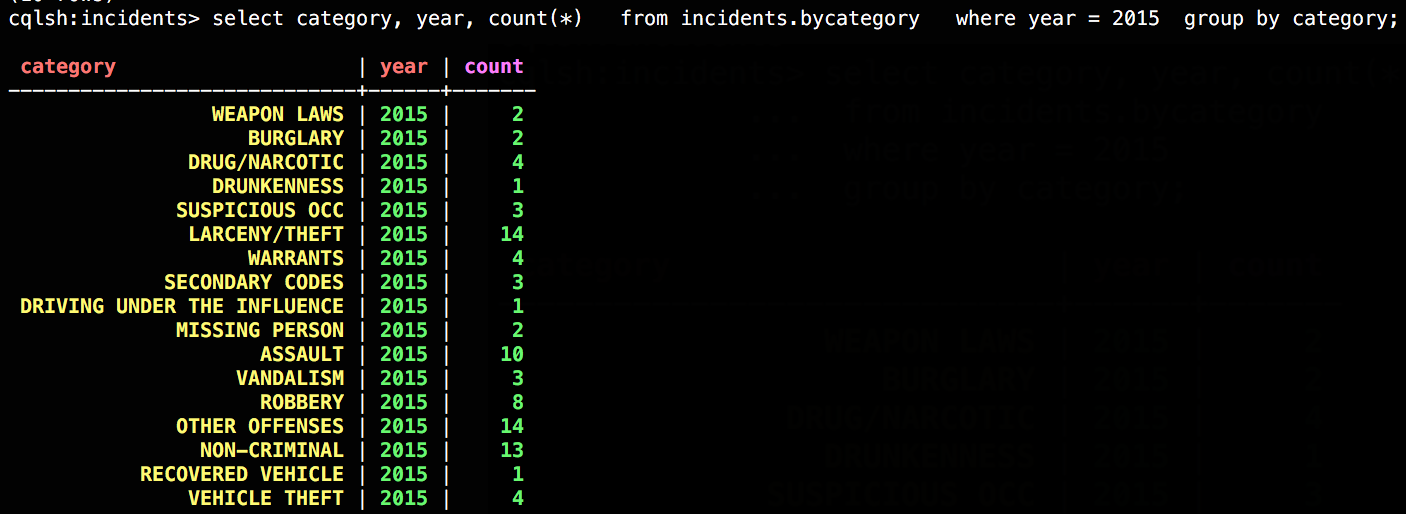
Para obtener número de incidencias producidas de cierta categoria (por un determinado año). Sin añadir filtro, más eficiente.

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

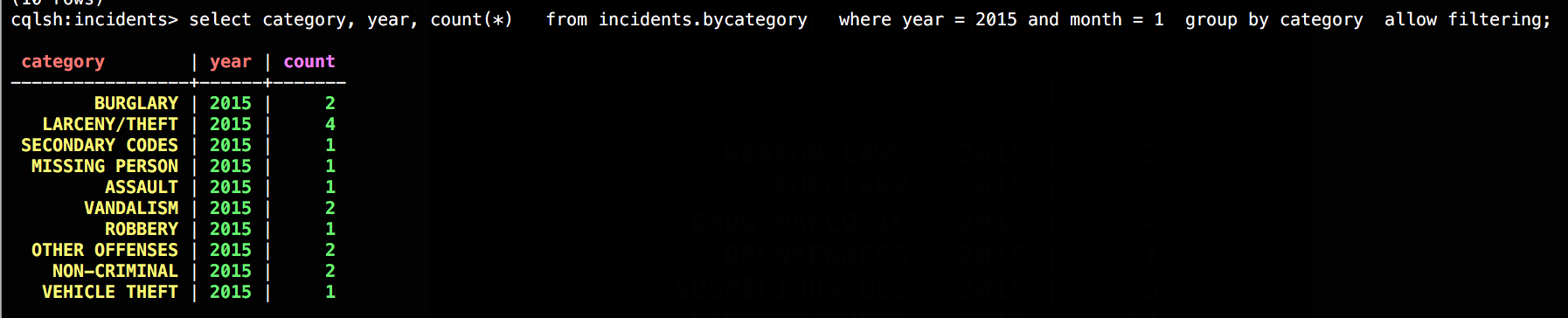
select category, year, count(\*)

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)

#### Mongodb

#### Neo4j

### Gráficos

#### Cassandra

#### Mongodb

#### Neo4j

### 

## 3. Bibliografía

* Python Cassandra Driver. 3.13.0 (2017). Datastax. <https://datastax.github.io/python-driver/>
* PyMongo. 3.6.1 (2018). MongoDB. <https://api.mongodb.com/python/current/index.html>
* Neo4j Bolt Driver for Python.1.6 (2018). Neo4J. <https://neo4j.com/docs/api/python-driver/current/>