Universidad Carlos III

Arquitectura de Datos

Curso 2024-25

Práctica 2

Migración de una base de datos a `Cassandra`

Ingeniería Informática, Cuarto curso

Adrián Fernández Galán (NIA: 100472182, e-mail: 100472182@alumnos.uc3m.es)
César López Mantecón (NIA: 100472092, e-mail: 100472092@alumnos.uc3m.es)
Manuel Gómez-Plana Rodríguez (NIA: 100472310, e-mail: 100472310@alumnos.uc3m.es)

Prof . Lourdes Moreno López
Grupo: 81

Índice

1. Introducción

En este documento se recoge la parte de diseño para el desarrollo de la práctica 2 de la asignatura *Arquitectura de Datos*. En esta práctica se tratará de completar una migración de una base de datos desde `MongoDB` a `Cassandra`. Además, se computarán nuevas tablas con el fin de permitir el análisis estadístico, aprovechando las cualidades de `Cassandra` para el análisis de datos gracias a su capacidad para la consulta masiva de datos de una misma columna.

La metodología usada para el desarrollo de este proyecto ha sido la siguiente: análisis de datos y casos de uso, realización del diseño lógico y físico orientado a `Cassandra` e implementación de casos de uso en forma e consulta. Adicionalmente se empleará la herramienta `PySpark` para realizar el primer volcado de datos.

En el diseño de consultas se tratará de aprovechar al máximo las capacidades de `Cassandra` en la lectura y escritura, dejando a la aplicación otra clase de operaciones. De esta forma, ambos sistemas trabajarán en conjunto, garantizando la eficiencia de la aplicación.

2. Modelo de Información del nuevo sistema

Para completar la migración es necesario realizar un estudio de los datos almacenados en el antiguo modelo. Como la anterior gestión de los datos se realizaba en el gestor de bases de datos *MongoDB* se tiene un `json` con todos los datos.

Observando las características del `json` con la información del anterior sistema podemos sacar las siguientes conclusiones:

- El antiguo sistema almacena registro sobre los vehículos que han circulado por diferentes autovías, este registro se ha hecho a través de grabaciones realizadas por los diferentes radares situados por las autovías.
- El sistema también gestiona sanciones ya emitidas como multas por velocidad (*speed ticket*), cargos administrativos (*clearance ticket*) y multas en zonas de radar de tramo (*stretch ticket*).
- Para cada uno de los registros se pueden identificar los siguientes elementos:
  - Una autovía o carretera
  - Un radar en un kilometro determinado de la carretera
  - Un vehículo que ha cruzado el radar
  - El vehículo consta de información del conductor en ese momento y del propietario del vehículo
  - Una grabación realizada por el radar sobre el vehículo en cuestión

Con el objetivo de entender el dominio del problema se ha creado un diagrama de clases en formato UML que nos permita entender los elementos que se tienen en el sistema y cómo estos se relacionan entre sí.

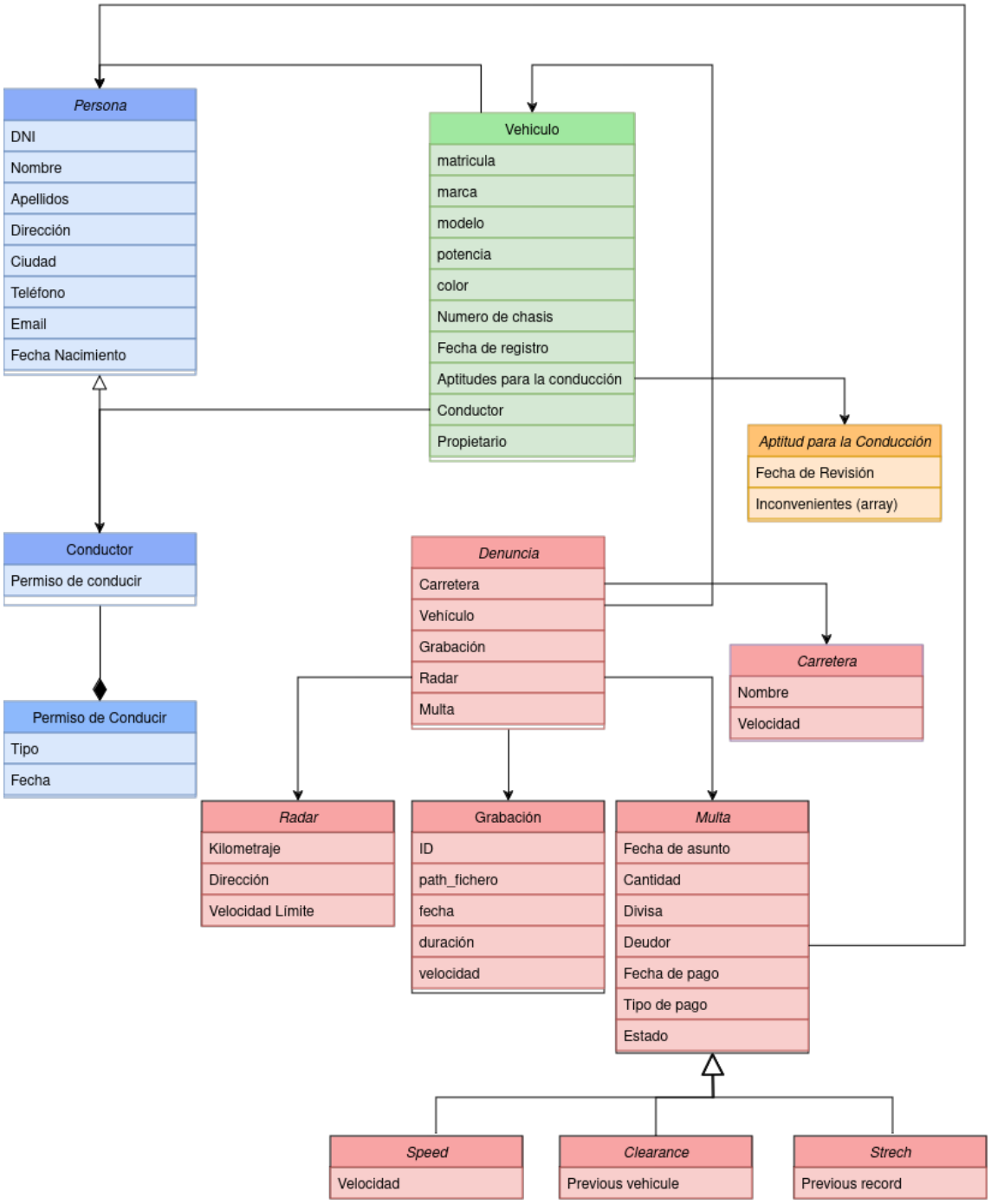

Figura 1: Diagrama de clases

3. Análisis de los Casos de Uso y diseño de consultas

Para poder desarrollar el nuevo sistema es necesario entender a la perfección los casos de uso que se nos plantean. A continuación se analizan los distintos casos de uso que se presentan.

Los casos de uso se pueden dividir según su finalidad:

- **Por funciones operativas**: Aquellos casos de uso que tienen como objetivo generar nuevas funcionalidades.
- **Para análisis estadístico**: Aquellos casos de uso que tienen como objetivo organizar la información de tal manera que pueda realizarse un análisis estadístico sobre ella.

3.1. Funciones Operativas

Encontramos dos casos de uso enfocados a las funciones operativas.

- **Emisión de sanciones:**

  Este caso de uso busca generar nuevas sanciones a aquellos conductores que no cumplen con una serie de condiciones relacionadas con la velocidad del coche durante el tramo de radar, la situación personal del conductor identificado, el estado de las revisiones técnicas del vehículos y otros factores. A continuación se realizará descripción de las condiciones que generarán una nueva sanción:

  - *Discrepancia en el carné de conducir del conductor*: En aquellos casos en los que la fecha de obtención del permiso de conducir no sea superior a la fecha de nacimiento del conductor en 18 años.
  - *Conducción con el vehículo deficiente*: Un coche se considera deficiente si en su última revisión se ha identificado algún defecto.
  - *Impago de sanciones emitidas*: Se generará una nueva sanción a aquellos conductores no hayan realizado el pago durante la fecha prevista, dado que no se tiene el conocimiento de esta fecha se tomará una fecha que corresponda con la mediana de todas las fechas existentes en la base de datos.

  Con el objetivo de cumplir este caso de uso se ha generado una tabla llamada `sanciones`. En esta tabla se recopilarán todas las sanciones de cualquier tipo, tanto generadas como las emitidas. Para ello se generará la siguiente tabla:

Sanciones		
dni_deudor	text	K
tipo	text	C↑
fecha_grabacion	date	C↓
cantidad	int	
dni_conductor	text	
dni_propietario	text	
estado	text	
matrícula	text	

Cuadro 1: Tabla SANCIONES

Tal y como funciona *Cassandra* es necesario crear una tabla vaya de la mano de la consulta. Para satisfacer el caso de uso se realizará a través de la siguiente consulta:

```
SELECT * from Sanciones WHERE dni_deudor = '12345678X' AND tipo
    IN ('discrepancia carne', 'desperfectos', 'impago')
```

Listing 1: Consulta de todas las sanciones

Como se puede observar para poder realizar esta consulta en *Cassandra* es necesario que el *DNI* y *Tipo* formen parte de la clave primaria de la tabla. Además, la elección de *Fecha* como clave de *clustering* permite ordenar todas las sanciones generadas y concede control en la unicidad de los registros.

- **Apertura de un proceso ejecutivo para expedientes activos sin plazo de pago cerrado:**

  Este caso de uso busca identificar aquellos expedientes que continuan activos pero su plazo de pago no ha sido cerrado, para posteriormente abrir un proceso ejecutivo realizado por otro sistema de gestión de la DGT que permita avisar a los conductores para que realicen su pago. Esta funcionalidad se contempla en la siguiente consulta sobre la tabla anterior:

```
SELECT * from Sanciones WHERE estado='stand_by' ALLOW FILTERING
    ;
```

Listing 2: Consulta de las sanciones pendientes de pago

En este caso es necesario incluir la *flag* `ALLOW FILTERING` debido a que las claves primaria y de clustering no forman parte de la condición de búsqueda. Sin embargo, ya que es necesario comprobar todos los registros para extraer aquellos con el estado adecuado no presenta un problema.

3.2. Análisis Estadístico

Para el análisis estadístico la DGT está interesada en los siguientes estudios:

- Estudio por marcas y modelos

  Este caso de uso busca realizar un estudio sobre las marcas y modelos de coches que más infracciones cometen. Es por ello que se exigen tres análisis:

  1. Número de multas por marca y modelo del vehículo
  2. Los tres colores más multados de coches
  3. Las marcas y modelos de los vehículos con más infracciones por velocidad

  Se han elaborado dos tablas distintas sobre las que hacer las consultas en `Cassandra` que permitan obtener cada una de estas estadísticas. Las tablas tienen la siguiente forma:

Multas por marca y modelo		
marca	text	K
modelo	text	K
matricula	text	C↓
fecha_grabacion	text	C↓
tipo	text	

Cuadro 2: Tabla MULTAS_MARCA_MODELO

Multas por color		
color	text	K
matricula	text	C↓
fecha_grabacion	date	C↓

Cuadro 3: Tabla MULTAS_COLOR

La primera tabla recoge todas las sanciones con la marca y el modelo como claves de partición. De esta forma la consulta relativa a este caso de uso será más eficiente.

A continuación se incluyen las consultas realizadas sobre estas tablas:

```
// Query para el numero de multas por marca y modelo
SELECT marca, modelo, COUNT(*) as total_multas FROM
   multas_marca_modelo GROUP BY marca, modelo;
```

```
// Query para el numero de multas por color
SELECT color, COUNT(*) as total_multas from multas_color_coche
    GROUP BY color;

// Query para sacar las marcas y modelos de los vehiculos con
    mas infracciones por velocidad
SELECT marca, modelo, COUNT(*) as total_multas FROM
    multas_marca_modelo WHERE tipo = 'velocidad' GROUP BY marca,
     modelo ALLOW FILTERING;
```

Listing 3: Querys para el caso de uso 1

Para la primera y tercera consulta, se obtienen directamente los datos que se piden para el caso de uso. En el caso de la consulta 2, se obtienen todas las multas para cada color y se deja como responsabilidad de la aplicación obtener los tres registros con mayor número de multas. Esto último es necesario ya que `Cassandra` sólo soporta ordenar en una única partición. No obstante, los datos de esta consulta podrían estar distribuidos en varios nodos.

- Estudio por carreteras

  Este caso de uso busca realizar un estudio sobre las carreteras más conflictivas de España, haciendo un análisis del exceso de velocidad medio en carreteras, así como los tramos donde más infracciones se han cometido. Es por ello que se pide lo siguiente:

  1. Exceso de velocidad medio para una carretera determinada
  2. Tramo y sentido más conflictivo de una carretera

  Se han elaborado dos tablas distintas sobre las que hacer las consultas en `Cassandra` que permitan obtener cada una de estadísticas. Las tablas tienen la siguiente forma:

Exceso de Velocidad Medio Por Carretera		
carretera	text	K
fecha_grabacion	date	C↓
velocidad_registrada	int	
velocidad_limite_radar	int	

Cuadro 4: Tabla EXCESO_VELOCIDAD_CARRETERA

Tramo y Sentido más Conflicto por Carretera		
carretera	text	K
fecha_grabacion	date	C↓
kilometro	int	
sentido	text	

Cuadro 5: Tabla CONFLICTO_TRAMO_SENTIDO

Las consultas para obtener las estadísticas son las siguientes:

```
// Query para el exceso de velocidad medio para una carretera
    determinada
SELECT carretera, AVG(velocidad_registrada) as
    media_velocidad_registrada, AVG(velocidad_limite_radar) as
    media_velocidad_radar FROM exceso_velocidad_carretera GROUP
    BY carretera;

// Query para el tramo y sentido mas conflictivo de una
    carretera
```

```
SELECT carretera, kilometro, sentido, COUNT(*) as
    infracciones_tramo FROM conflictos_tramo_sentido GROUP BY
    carretera, kilometro, sentido;
```

Listing 4: Consultas para el caso de uso 2

La primera consulta devuele el promedio de las velocidades esperadas en un tramo y las registradas para conches infractores. La segunda consulta permite conocer el número de infracciones para cada tramo de cada carretera, de forma que la aplicación pueda seleccionar aquellos registros con mayor número de multas.

Adicionalmente, se puede modificar ligeramente la consulta en caso de querer seleccionar los registros para una única carretera. En este caso sería necesario usar la *flag* `ALLOW FILTERING` debido a que la condición where no agrupa todos los atributos de la clave de partición.

```
// Query para el tramo y sentido mas conflictivo de una
    carretera concreta
SELECT carretera, kilometro, sentido, COUNT(*) as
    infracciones_tramo FROM conflictos_tramo_sentido  where
    carretera = 'A2' GROUP BY carretera, kilometro, sentido
    ALLOW FILTERING;
```

Listing 5: Querys modificada para el caso de uso 2

- Estudio por conductores

  El último caso de uso busca realizar un estudio sobre los conductores, concretamente, buscan determinar los conductores más infractores, así como la probabilidad de que se cometa una infracción cuando un vehículo no es conducido por el dueño del mismo.

  Para determinar qué conductores son los más infractores se ha diseñado la siguiente consulta sobre la tabla sanciones:

```
SELECT dni_deudor, COUNT(*) as num_multas FROM sanciones GROUP
    BY dni_deudor;
```

Listing 6: Consulta para obtener los conductores más infractores

Esta consulta devuelve el número de infracciones para cada conductor, de forma que la aplicación pueda seleccionar fácilmente el valor más alto del registro.

Para obtener la probabilidad de que un infractor sea el propietario del coche se ha diseñado una tabla adicional con la siguiente forma:

Probabilidad de infracción		
conductor_igual_propietario	boolean	K
matricula	text	C↓
fecha_grabacion	timestamp	C↓

Cuadro 6: Tabla CONDUCTORES_MAS_INFRACTORES

El valor requerido es el resultado de la siguiente fórmula:

$$P = \frac{\text{num\_conductor\_eq\_propietario}}{\text{num\_multas}}$$

Estos dos valores se pueden obtener fácilmente mediante dos consultas:

```
// Obtencion del numero de multas totales
SELECT COUNT(*) FROM sanciones;
// Obtencion del numero de infracciones con conductor =
    propietario
```

```
select conductor_igual_propietario, count(*) from
    probabilidad_infraccion GROUP BY conductor_igual_propietario
    ;
```

Listing 7: Querys para el caso de uso 3

Con ambos valores la aplicación puede calcular el valor requerido, garantizando la resolución eficiente de este caso de uso.

4. Modelo Lógico y Físico

Tras presentar los casos de uso y cómo estos casos de uso derivan en sus respectivas consultas y tablas, estudiaremos cómo se ha realizado el modelo completo del sistema para conseguir que dichas tablas se carguen con los datos necesarios.

4.1. Modelo Lógico

4.1.1. Sanciones

El primer enfoque que se ha considerado es conseguir alimentar la tabla **Sanciones** con aquellas personas que tenga algún tipo de sanción. Por lo que para alimentar esta tabla tenemos que tratar dos grupos de sanciones: sanciones generadas y sanciones emitidas previamente.

- **Sanciones ya generadas**: En los datos originales encontramos sanciones ya emitidas con anterioridad como lo son *Speed Ticket*, *Clearance Ticket* y *Stretch Ticket*, por lo que ha creado un *dataframe* en *Pyspark* para cada una de estas multas.
- **Nuevas sanciones**: Para generar nuevas sanciones se han contemplado las condiciones descritas en el punto 3.1. Para cada una de ellas se ha generado un *dataframe* en *Pyspark* que parte de los datos del *JSON*, excepto en el caso de *impago de sanciones* que utilizará los dataframes de las multas para conocer qué conductores deben de recibir otra infracción. En cada uno de los *dataframes* se insertarán los datos que cumplan la condición, de manera que solo contengan los datos de aquellos conductores infractores. Finalmente los *dataframes* creados han sido los siguientes: `Discrepancia_Carnet`, `Vehiculo_Deficiente`, `Impago_Sanciones`

Cada de uno de los *dataframes* mencionados tendrá únicamente los atributos necesarios para alimentar la tabla de *Sanciones* y cualquier otra tabla que satisfaga a un caso de uso. La tabla *Sanciones* será la unión de algunos atributos de cada una de los *dataframes* mencionados. Tal como se ha indicado en el anterior punto, *Sanciones* cuenta con los atributos `dni_deudor`, `dni_propietario`, `dni_conductor`, `fecha_grabacion`, `matricula`, `cantidad`, y `estado`; por lo que será necesario que los *dataframes* tengan esta información, o al menos pueda ser obtenida de algún modo. Esta nueva tabla será insertada en `Cassandra`.

Para ello los *dataframes* anteriores contarán con `dni_deudor`, `fecha_grabación`, `matricula`; y si son sanciones emitidas previamente se usará su estado, tipo y cantidad, en cambio para las sanciones generadas se les dotará de una cantidad, el estado de `stand_by` y el tipo según la procedencia de la multa.

A continuación se puede ver un diagrama que recoge el proceso de migración para la creación de la tabla *Sanciones*.

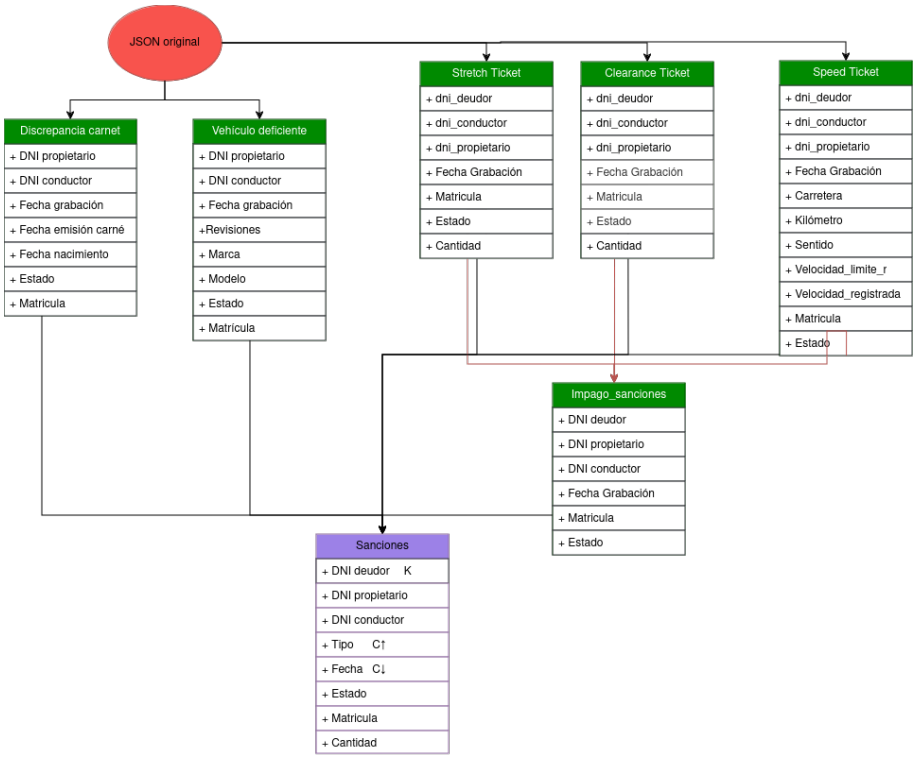

Figura 2: Modelo lógico de Sanciones

En verde se representan los *dataframes* auxiliares que se construyen en *Pyspark*.

4.1.2. Multas por marca y modelo o por color

Estas dos tablas se alimentan de datos de los vehículos y de las sanciones. Para cumplir con la funcionalidad descrita en los casos de uso y tal y cómo se expuso en el punto 3.2 es necesario que estas tablas contengan los siguientes datos:

- **Multas por marca y modelo**: `matricula`, `modelo`, `marca`, `fecha_grabacion` y `tipo_multa`.
- **Multas por color**: `color`, `fecha_grabacion` y `matricula`.

El proceso de migración que se ha seguido para alimentar estas tablas viene recogido en el siguiente diagrama:

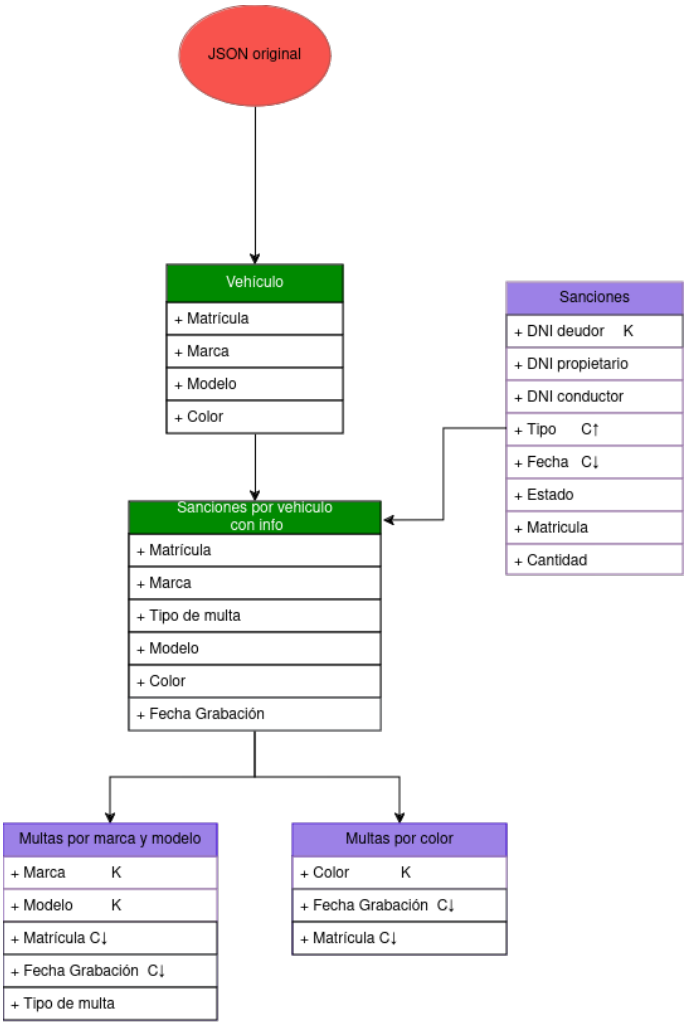

Figura 3: Modelo lógico de Multas por Color y Multas por marca y modelo

4.1.3. Tramo conflictivo y exceso de velocidad medio

Las tablas con información relativa al tramo más conflictivo y el exceso de velocidad medio contienen los siguientes datos:

- **Tramo conflictivo**: `carretera`, `kilometro`, `sentido` y `fecha_grabacion`.
- **Exceso de velocidad medio**: `carretera`, `velocidad_registrada`, `velocidad_limite_radar` y `fecha_grabacion`.

Es por eso que necesitan alimentarse exclusivamente de información relativa a las multas por exceso de velocidad, tal y como se representa en el siguiente diagrama:

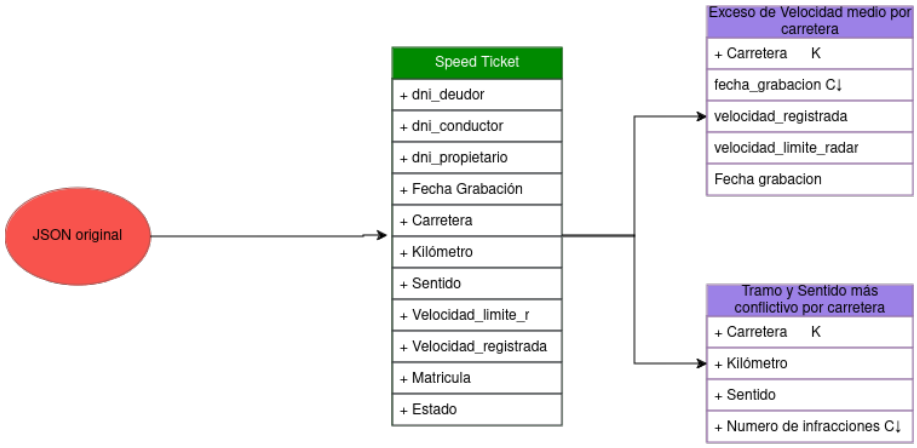

Figura 4: Modelo lógico de Tramos conflictivos y Exceso de velocidad medio

4.1.4. Probabilidad de infracción

La tabla con información relativa al caso de uso 3, es decir, el análisis de la probabilidad de infracción cuando el conductor de un vehículo multado es distinto al propietario, contiene los siguientes datos:

- **Conductor igual a propietario**: `conductor_igual_propietario`, `matricula`, y `fecha_grabacion`.

Es por ello que necesitan obtener estos datos de la tabla de sanciones, explicada anteriormente,tal y como se puede observar en la siguiente figura:

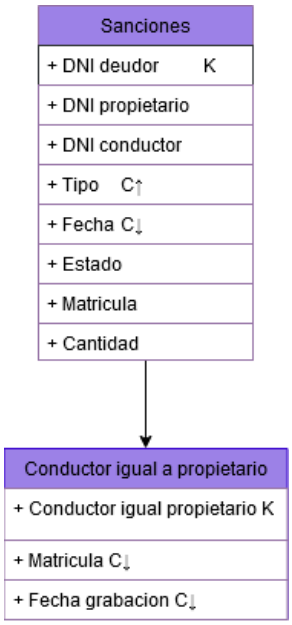

Figura 5: Modelo lógico de la Probabilidad de multa con conductor igual a propietario

4.1.5. Diagrama completo

Con todo lo anterior, el diagrama que representa la nueva base de datos, así como el proceso de migración es el siguiente:

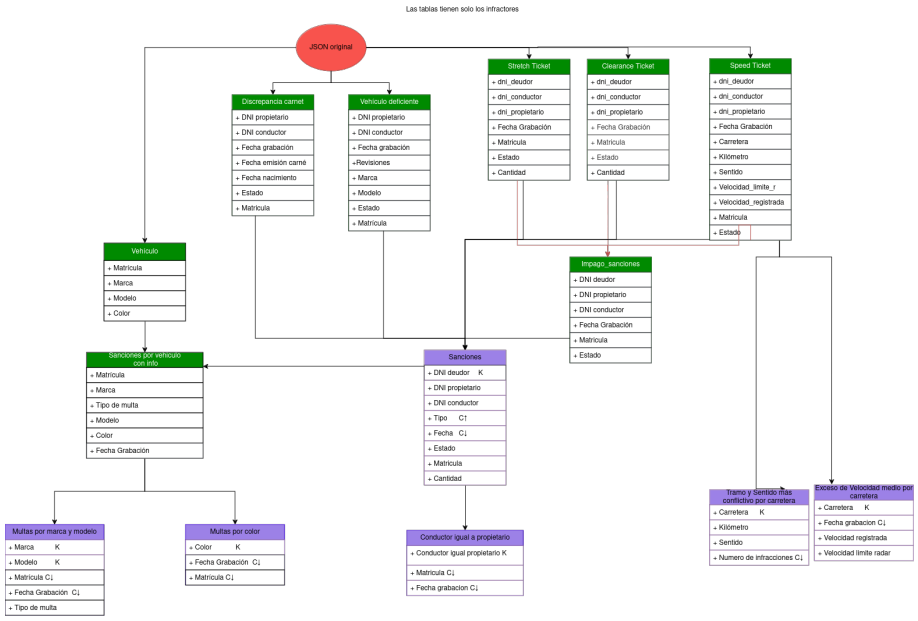

Figura 6: Modelo lógico completo

4.2. Modelo físico

4.2.1. Definición de keyspace

Pese a que pudiera parecer que existen dos contextos funcionales distintos (funciones operativas y análisis estadístico), dado que las tablas contienen información redundante en muchos casos y que existen relaciones claras entre todas ellas se ha decidido mantener toda la base de datos en un único *keyspace* llamado `practica2`. Este nombre, pese a no ser representativo de la base de datos, se ha elegido por ser muy fácilmente distinguible en la máquina virtual de otras bases de datos resultado de otros ejercicios. En un contexto laboral se debería cambiar por un nombre más representativo de la naturaleza de los datos como $DatosSancionesDGT$ o similares.

4.2.2. Elección de tipos

Los tipos de datos han sido contemplados en el modelo lógico. Aunque no sea una decisión común en ese ámbito se ha decidido seguir esa aproximación para permitirnos tener una primera estimación durante la fase de diseño de las necesidades de almacenamiento y particularidades del *pipeline* de migración, así como conversiones necesarias de tipos o posibles correcciones que hacer sobre los datos.

A continuación se incluye un resumen de las tablas:

Sanciones		
dni_deudor	text	K
tipo	text	C↑
fecha_grabacion	date	C↓
cantidad	int	
dni_conductor	text	
dni_propietario	text	
estado	text	
matrícula	text	

Multas por marca y modelo		
marca	text	K
modelo	text	K
matricula	text	C↓
fecha_grabacion	text	C↓
tipo	text	

Multas por color		
color	text	K
matricula	text	C↓
fecha_grabacion	date	C↓

Exceso de Velocidad Medio Por Carretera		
carretera	text	K
fecha_grabacion	date	C↓
velocidad_registrada	int	
velocidad_limite_radar	int	

Tramo y Sentido más Conflicto por Carretera		
carretera	text	K
fecha_grabacion	date	C↓
kilometro	int	
sentido	text	

Probabilidad de infracción		
conductor_igual_propietario	boolean	K
matricula	text	C↓
fecha_grabacion	timestamp	C↓

4.2.3. Dimensionamiento

Con el objetivo de entender y analizar la escalabilidad, el nivel de optimización y el balanceo de la carga de trabajo del sistema, se calcularán los tamaños de las particiones.

Para conocer el número de celdas (valores) que tiene cada partición se usará la siguiente fórmula:

$$N_v = N_r \cdot (N_c - N_{pk} - N_s) + N_s$$

- N_v: Número de valores en la partición
- N_s: Número de columnas estáticas
- N_r: Número de filas
- N_c: Número de columnas
- N_{pk}: Número de columnas en la pk

Con esta fórmula se calcula el número de valores de cada partición para cada una de las tablas. Para entender cómo se realizarán los cálculos se se desarrollará la fórmula para una tabla, en este caso se usará la tabla **Sanciones**:

- Nº de columnas: $N_c = 8$
- Nº de columnas en pk: $N_{pk} = 3$
- Nº de columnas estáticas: $N_s = 0$
- Nº de filas: $N_r = 114049$

Obtenidos los datos para la fórmula, sustituimos

$$114049 \cdot (8 - 3 - 0) + 0 = 570245 \text{filas por partición}$$

A continuación se muestran los valores por partición para cada tabla:

- **Sanciones**: 570245
- **Multas por Marca y Modelo**: 48725
- **Multas por Color**: 1

- **Conductor igual a propietario**: 1
- **Tramo y Sentido más conflictivo por carretera**: 88362
- **Exceso de Velocidad medio por carretera**: 88362

Es importante destacar que para las tablas donde todas las columnas forman parte de la clave primaria, aunque la formula resulte en 0, se ha tomado el valor 1, dado que cada fila insertada se ubicará en una partición.

Tras calcular el número de filas por partición se calculará el tamaño en disco de las tablas, esto se conseguirá a través de la siguiente fórmula:

$$S_t = \sum_i sizeOf(c_k) + \sum_j sizeOf(c_{s_j}) + N_r \cdot (\sum_k sizeOf(c_{r_k}) + \sum_l sizeOf(c_{c_l})) + N_v \cdot sizeOf(t_{avg})$$

- c_k: Cuenta las columnas que son clave de partición
- c_s: Cuenta las columnas estáticas
- c_r: Cuenta las columnas normales
- c_c: Cuenta las columnas que pertenecen a la clave de clusterización
- t_{avg}: Promedio de bytes utilizados por Cassandra para almacenar los metadatos de un valor, suele ser 8 bytes
- La función `sizeOf()` indica el tamaño en bytes de los tipos CQL utilizados en la definición de cada columna

Al igual que para el anterior cálculo, se desarrollará en profundidad el cálculo del tamaño de una partición de una tabla, de nuevo usaremos **Sanciones**:

- No tiene columnas estáticas ($c_s = 0$)
- Cálculo de bytes por columna
  - DNI deudor, es de tipo texto con 9 simbolos = 9 bytes
  - DNI propietario, 9 bytes
  - DNI conductor, 9 bytes
  - Tipo, es de tipo texto y la cadena más larga puede ser de 19 caracteres = 19 bytes
  - Fecha, timestamp = 8 bytes
  - Estado, es de tipo texto y la cadena más larga es "fullfied"que tiene 10 caracteres = 10 bytes
  - Matricula, 7 caracteres = 7 bytes
  - Cantidad, int = 4 bytes
- Suma de columnas de clave de partición = 9 bytes
- Suma de columnas de clave de clusterización = 27 bytes
- Suma de columnas normales = 29 bytes

Con los valores calculados sustituimos en la fórmula

$$S_t = 9 + 0 + 114049 \cdot (39 + 27) + 570245 \cdot 8 = 10948713 bytes = 12{,}09 MB$$

A continuación se muestran los tamaños de las particiones de las siguientes tablas:

- **Sanciones**: 12.09 MB
- **Multas por Marca y Modelo**: 2.05 MB
- **Multas por Color**: 0.73 MB

- **Conductor igual a propietario**: 0.73 MB
- **Tramo y Sentido más conflictivo por carretera**: 1.5 MB
- **Exceso de Velocidad medio por carretera**: 1.41 MB

Se ha supuesto que la marca y el modelo no superarán los 15 caracteres, que el color de los vehiculos no superará los 20 caracteres y que el nombre de la carretera no supera los 8 caracteres.

El límite de tamaño de una partición en Cassandra es de 2GB, pero se recomienda que el tamaño sea de 1MB. En nuestro caso todas las tablas tienen una gran holgura respecto al tamaño máximo de una partición. Sin embargo, únicamente 2 tablas no exceden el tamaño recomendado. Respecto a las demás, existen dos tablas que exceden el tamaño recomendado por, aproximadamente, 0.5 MB; y una tabla, **Sanciones**, que supera en un 1200 % el tamaño recomendado.

El tamaño de la tabla **Sanciones** es un problema a tener en cuenta y necesario de solucionar. La propuesta más sencilla es añadir una una nueva columna como clave de partición. Se sugiere usar *Matricula* con este fin, de forma que se optimicen consultas sobre sanciones que tiene una persona según los coches que tenga. Este cambio reduce el número de filas por partición de 570245 a **456196**, lo que supone que el tamaño de cada partición sea de **10.38 MB** y una reducción del **14 %** del tamaño con tan solo cambiar un atributo.

Aunque esté lejos del tamaño recomendado, se pueden seguir añadiendo atributos a la clave de partición según las necesidades emergentes de la base de datos para seguir reduciendo el tamaño.

4.2.4. Testeo de rendimiento

Se han ejecutado las consultas diseñadas en el apartado relativo a los casos de uso (ver 3), logrando obtener la totalidad de los datos en tiempos muy reducidos. Algunas de estas consultas devuelven grandes volúmenes de datos, destacando el buen rendimiento de `Cassandra` en lecturas. Es por esto que concluímos que el rendimiento de la base de datos es suficiente y no necesita de grandes optimizaciones.

5. Conclusiones

Esta práctica nos ha permitido introducirnos al diseño de bases de datos con la herramienta `Cassandra` y a la migración de bases de datos a través de herramientas como `PySpark`. Consideramos crucial la combinación de los lenguajes de programación con los sistemas de gestión de bases de datos para lograr un sistema eficiente y una migración completa que nos perita explotar las características de `Cassandra`.

También destacamos la importancia de crear un modelo adaptado al sistema gestor de base de datos. En el caso particular de esta práctica, `Cassandra` es un tipo de base de datos no relacional orientado a columnas optimizado para lecturas y escrituras masivas. Por lo que las tablas del diseño deben estar optimizadas para consultas específicas, estructuradas según las necesidades de acceso y los patrones de consulta previstos. Esto implica diseñar tablas que minimicen las uniones (joins) y aprovechen al máximo las características clave de `Cassandra`, como las claves de partición y las claves de agrupamiento para garantizar una distribución uniforme de los datos y un acceso eficiente.

Además, esta práctica resalta la importancia de comprender las diferencias entre los sistemas de gestión de bases de datos relacionales y no relacionales. En `Cassandra`, el modelado de datos se centra en las consultas, por lo que el diseño debe priorizar el rendimiento y la escalabilidad, sacrificando en algunos casos la normalización para evitar operaciones costosas.

En cuanto a la migración con `PySpark`, destacamos su capacidad para manejar grandes volúmenes de datos y transformarlos de manera eficiente, lo que resulta fundamental en entornos distribuidos. `PySpark` no solo facilita la extracción y transformación de datos desde fuentes heterogéneas, sino que también permite integrarlos de manera óptima en un sistema como `Cassandra`, asegurando consistencia y alta disponibilidad.

En conclusión, esta práctica nos ha permitido integrar nuestros conocimientos teóricos en bases de datos No SQL con conocimiento práctico en lenguajes de programación, comprendiendo la importancia de elegir las herramientas adecuadas para diseñar y migrar bases de datos que satisfagan las necesidades de escalabilidad y eficiencia de sistemas para la gestión de grandes volúmenes de información.