



Universidad Técnica Particular de Loja

Integrantes:

Miguel Ángel Caraguay Correa

Adriana Sofía Jaramillo Ochoa

Mateo Sebastián Martínez Velásquez

Tema:

Proyecto Integrador

Docentes:

Prácticum: Ing. Danilo Jaramillo

Bases de Datos: Ing. Nelson Piedra

Programación Funcional: Ing. Jorge López

Grupo: A

Proyecto Integrador

Tabla de contenido

3. Introducción	7
4. Objetivos	8
4.1 Objetivo general.....	8
4.2 Objetivos específicos	8
5. Marco Teórico:.....	9
5.1 Programación funcional	9
5.2 Programación Reactiva	11
5.3 Base de datos.....	12
5.4 MySQL	14
5.5 Vega lite	15
5.6 Scala.....	16
5.7 Slick	17
5 Procedimiento de programación funcional y reactiva	18
5.6 Proceso consulta uno.....	19
6.2 Proceso consulta dos	21
6.3 Proceso consulta tres.....	23
6.4 Proceso consulta cuatro.....	25
6.5 Proceso consulta cinco.....	27
6.6 Proceso consulta seis	28
6 Procedimiento para realizar las visualizaciones en vega-lite:.....	30
6.6 Proceso consulta uno.....	31

6.7 Proceso consulta dos	34
6.8 Proceso consulta tres	36
7 Procedimiento de base de datos	44
7.6 Proceso de creación del script de datos matriculación vehicular.....	44
7.7 Proceso consulta uno.....	49
7.8 Proceso consulta dos	50
7.9 Proceso consulta tres	51
8 Conclusiones	52
9 Bibliografía	53
10 Anexos	55
Anexo 1	55
Anexo 2	56
Anexo 3	57
Anexo 4	58

Ilustración 1 Importaciones para leer y escribir archivos csv	18
Ilustración 2 Código para ejecutar la consulta uno	20
Ilustración 3 Código para la creación del archivo correspondiente a la consulta uno	20
Ilustración 6 Código para ejecutar la consulta tres	23
Ilustración 7 Código para la creación del archivo correspondiente a la consulta tres	24
Ilustración 4 Código para ejecutar la consulta dos	21
Ilustración 5 Código para la creación del archivo correspondiente a la consulta dos	21
Ilustración 8 Scripts para trabajar vega en HTML.....	30
Ilustración 9 Contenedores de los gráficos de vega.....	30
Ilustración 10 Código para realizar la visualización de la consulta uno	32
Ilustración 11 Visualización de la consulta uno	33
Ilustración 12 Código para realizar la visualización de la consulta dos	34
Ilustración 13 Visualización de la consulta dos	35
Ilustración 14 Código para realizar la visualización de la consulta tres	36
Ilustración 15 Visualización de la consulta tres.....	37
Ilustración 16 Creación de tablas en el script	44
Ilustración 17 Importación de datos de matriculación.....	45
Ilustración 18 Proceso de carga de datos de matriculación	46
Ilustración 19 Datos importados	47
Ilustración 20 Modelo entidad relación	48
Ilustración 21 Sentencias para realizar la consulta uno en MySQL	49
Ilustración 22 Resultado de la consulta uno	49
Ilustración 23 Sentencias para realizar la consulta dos en MySQL.....	50
Ilustración 24 Resultado de la consulta dos	50
Ilustración 25 Sentencias para realizar la consulta tres en MySQL.....	51

Ilustración 26 Resultado de la consulta tres.....	51
---	----

3. Introducción

El presente proyecto consta en la integración de diversas materias enfocadas en el ámbito de la computación, haciendo uso de estas se demostrará todos los conocimientos adquiridos en el presente ciclo universitario tanto en las ramas de programación funcional y reactiva como en base de datos.

Para lo cual vamos a integrar estos conocimientos para resolver consultas solicitadas por el usuario utilizando los datos de matriculación de automóviles del año 2008 en Ecuador, adquiridos del Instituto Ecuatoriano de Normalización (INEN).

Para este informe se pondrá información de todas las herramientas y métodos que se van a usar para el desarrollo de las consultas, toda esta información se colocará en el marco teórico

Consultas requeridas por el usuario:

- ¿Cuál es la distribución (el número) de los vehículos clasificados según el tipo de combustible que usan?
- ¿Cuál es la distribución (el número) de los vehículos según el servicio que presentan?
- ¿Cuál es la distribución por provincia (el número) de los tipos de servicios para vehículos que pueden considerarse de trabajo pesado (Camión, tanquero, tráiler y volqueta)
- ¿Cuántas camionetas hay en LOJA con relación a su servicio?
- ¿Cuántos vehículos son a gasolina y son de marca TOYOTA, con respecto a la provincia?
- Consultar número de vehículos según el tipo de marca que no superen los 4 asientos y sean automóviles a gasolina

4. Objetivos

4.1 Objetivo general

Integrar las asignaturas seleccionadas en el presente ciclo académico, con el propósito de plasmar las habilidades adquiridas mediante la resolución de consultas.

4.2 Objetivos específicos

- Identificar las herramientas a usar para llegar a la resolución de las consultas.
- Realizar las consultas con los conocimientos adquiridos en el lenguaje de programación Scala aplicando el paradigma funcional para obtener los datos requeridos en las consultas.
- Graficar los resultados obtenidos de las consultas utilizando los conocimientos adquiridos de vega-lite.
- Diseñar una estructura de base de datos necesaria para el desarrollo de las consultas definiendo relaciones entre los datos.
- Aplicar conocimientos de MySQL para obtener los datos requeridos en las consultas directamente de la base de datos.

5. Marco Teórico:

5.1 Programación funcional

La programación funcional es un paradigma de programación en el que tratamos de representar funciones matemáticas puras, la forma en la que se analiza los problemas es de tipo declarativo puesto que se enfoca en el “qué necesito” y no en el “cómo tengo que hacerlo” que es como se lo analizaría en un tipo de programación imperativa. (Phares, 2018)

Las bases de la programación funcional son:

- **Inmutabilidad:** Es una de las bases fundamentales de la programación funcional y se basa en prevenir la mutación o el cambio de cualquier dato en nuestro programa. (Phares, 2018)
- **Funciones Puras:** Las funciones puras son, en pocas palabras funciones que no tienen ningún “efecto colateral” y el resultado que estas devuelven únicamente depende de los argumentos que esta recibe (Phares, 2018)
- **Recursión:** La recursión es como un tipo de bucle en el que la función se llama a sí misma hasta que se cumple cierta condición. (Phares, 2018)
- **Currying:** El currying consiste en “cargar” los parámetros de una función a otra función hasta que esta tenga la información necesaria para ser ejecutada. (Phares, 2018)
- **High Order Functions (Lambdas):** Las funciones de orden superior consisten en una función que es pasada como argumento a otra función. (Phares, 2018)

Según Digital Guide IONOS algunas ventajas y desventajas de la programación funcional son:

Ventajas:

- Código más preciso y corto
- Fácil de combinar con la programación imperativa y orientada a objetos
- Código fácilmente verificable, incluso las funciones sin estado se pueden verificar
- El código se puede testar fácilmente
- Muy adecuados para la paralelización
- Los programas no tienen estados (1&1 IONOS España S.L.U., 2020)

Desventajas:

- Los datos no se pueden modificar, son inmutables
- No se permite el acceso eficiente a grandes cantidades de datos
- No se recomienda para conexiones a bases de datos y servidores
- No es adecuado para muchas recursiones de la misma pila
- La programación recurrente puede dar lugar a errores graves (1&1 IONOS España S.L.U., 2020)

5.2 Programación Reactiva

La programación reactiva se enfoca en el trabajo con flujos de datos finitos o infinitos de manera asíncrona. El objetivo de la programación reactiva es conseguir una aplicación con capacidad de respuesta y una fácil escalabilidad. (Íñigo, 2020)

Las bases de los sistemas Reactivos publicadas por el Manifiesto Reactivo son:

- **Responsivos:** aseguran la calidad del servicio cumpliendo unos tiempos de respuesta establecidos. (Íñigo, 2020)
- **Resilientes:** se mantienen responsivos incluso cuando se enfrentan a situaciones de error. (Íñigo, 2020)
- **Elásticos:** se mantienen responsivos incluso ante aumentos en la carga de trabajo. (Íñigo, 2020)
- **Orientados a mensajes:** minimizan el acoplamiento entre componentes al establecer interacciones basadas en el intercambio de mensajes de manera asíncrona. (Íñigo, 2020)

Los beneficios de la programación reactiva son:

- **Escalabilidad:** se obtiene una implementación débilmente acoplada escalable y que tiende a aislar los fallos. La escalabilidad se refiere a la capacidad de escalar horizontalmente y de forma rápida, anticipándose al manejo del número de eventos asociados con los datos. (Á.Martín, 2018)
- **Ahorro:** se utiliza eficientemente los recursos, se puede procesar cargas de trabajo más altas con menos recursos. (Á.Martín, 2018)

5.3 Base de datos

Un sistema gestor de bases de datos (SGBD) consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos datos. La colección de datos, normalmente denominada base de datos contiene información relevante. Su objetivo principal es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto práctica como eficiente. Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la información.

Un sistema de bases de datos es una colección de archivos interrelacionados y un conjunto de programas que permitan a los usuarios acceder y modificar estos archivos. Uno de los propósitos principales de un sistema de bases de datos es proporcionar a los usuarios una visión abstracta de los datos. Es decir, el sistema esconde ciertos detalles de cómo se almacenan y mantienen los datos. (Silberschatz et al., 2002)

- **Abstracción de datos:**

- **Nivel físico:** El nivel más bajo de abstracción describe cómo se almacenan realmente los datos. (Silberschatz et al., 2002)
- **Nivel lógico:** El siguiente nivel más alto de abstracción describe qué datos se almacenan en la base de datos y qué relaciones existen entre esos datos. La base de datos completa se describe así en términos de un número pequeño de estructuras relativamente simples. Aunque la implementación de estructuras simples en el nivel lógico puede involucrar estructuras complejas del nivel físico, los usuarios del nivel lógico no necesitan preocuparse de esta complejidad. (Silberschatz et al., 2002)

- **Nivel de vistas:** El nivel más alto de abstracción describe sólo parte de la base de datos completa. A pesar del uso de estructuras más simples en el nivel lógico, queda algo de complejidad, debido a la variedad de información almacenada en una gran base de datos. Muchos usuarios del sistema de base de datos no necesitan toda esta información. En su lugar, tales usuarios necesitan acceder sólo a una parte de la base de datos. El sistema puede proporcionar muchas vistas para la misma base de datos. (Silberschatz et al., 2002)
- **Modelo de datos:**
 - **Modelo entidad-relación:** Una entidad o atributo es una «cosa» u «objeto», Por lo cual una relación es una asociación entre varias entidades. La estructura lógica general de una base de datos se puede expresar gráficamente mediante un diagrama ER, que consta de los siguientes componentes: Rectángulos, representan conjuntos de entidades. Elipses, que representan atributos. Rombos, que representan relaciones entre conjuntos de entidades. Líneas, que unen los atributos con los conjuntos de entidades y los conjuntos de entidades con las relaciones. (Silberschatz et al., 2002)

5.4 MySQL

Es un sistema de gestión de bases de datos relacionales de código abierto con un modelo cliente-servidor, es un software o servicio utilizado para crear y administrar bases de datos basadas en un modelo relacional, es decir, utiliza tablas múltiples que se interconectan entre sí para almacenar la información y organizarla correctamente. (B., 2020)

Los procesos Principales que tiene el entorno MySQL son:

- MySQL crea una base de datos para almacenar y manipular datos, definiendo la relación de cada tabla.
- Los clientes pueden realizar solicitudes escribiendo instrucciones SQL específicas en MySQL.
- La aplicación del servidor responderá con la información solicitada y esta aparecerá frente a los clientes.

Principales sentencias de MySQL utilizadas en este proyecto

- SELECT es usada para consultar datos.
- DISTINCT Sirve para eliminar los duplicados de las consultas de datos.
- WHERE Es usada para incluir las condiciones de los datos que queremos consultar.
- AND y OR es usada para incluir 2 o más condiciones a una consulta.
- ORDER BY Es usada para ordenar los resultados de una consulta.
- INSERT Es usada para insertar datos.
- UPDATE Es usada actualizar o modificar datos ya existentes.
- DELETE Es usada borrar datos. (Robledano, 2020)

5.5 Vega lite

Vega-Lite, consta con un lenguaje de gramática de alto nivel que permite una rápida especificación de visualizaciones de datos interactivas. Vega-Lite combina una gramática tradicional de gráficos, proporcionando reglas de codificación visual y un álgebra de composición para pantallas en capas y multivista. (Satyanarayan et al., 2017)

A su vez Vega-Lite permite que una selección la cual es una abstracción que define el procesamiento de eventos de entrada, los puntos de interés y una función de predicado para las pruebas de inclusión, usando el lenguaje de JSON. (Satyanarayan et al., 2017)

Las selecciones parametrizan las codificaciones visuales sirviendo como datos de entrada, definiendo extensiones de escala o impulsando la lógica condicional. El compilador Vega-Lite sintetiza automáticamente el flujo de datos requerido y la lógica de manejo de eventos, que los usuarios pueden anular para una mayor personalización. En contraste con las especificaciones reactivas existentes, las selecciones de Vega-Lite descomponen un diseño de interacción en unidades semánticas enumerables y concisas. A través de una variedad de ejemplos se evalúa, que demuestran una especificación precisa tanto de métodos de interacción personalizados como de técnicas comunes como panorámica, zoom y selección vinculada. (Satyanarayan et al., 2017)

5.6 Scala

Scala es un lenguaje de programación que se podría decir que es relativamente nuevo puesto que su primer lanzamiento se dio en el año 2003, a pesar de ello hasta el día de hoy tiene una comunidad la cual va creciendo debido a la popularidad del lenguaje. (RecluiT, 2020)

El diseño de Scala no solo nos brinda un lenguaje de programación que nos permite expresar código de una forma sencilla y elegante sino que además de esto Scala es un lenguaje que soporta múltiples paradigmas de programación así como lo son la programación orientada a objetos y la programación funcional, algo de igual forma importante a tener en cuenta es que Scala permite la interoperabilidad con Java puesto que el código de Scala se ejecuta en la máquina virtual de Java o JVM por sus siglas en inglés. (Eugene Yokota, s.f.)

A pesar de los múltiples paradigmas que soporta Scala en este proyecto nos enfocaremos en su mayoría al ámbito de la programación funcional, siendo así que al momento de tener que procesar datos para hacer consultas se utilizara todos los recursos y conceptos aprendidos en el ámbito funcional, como lo son las funciones anónimas, funciones puras y diversos métodos aprendidos para el tratamiento de conjuntos de datos.

Herramientas complementarias para Scala

Algunas de las herramientas y librerías adicionales utilizadas en el presente proyecto son:

- **SBT:** Simple Build Tool es una herramienta de construcción de proyectos para gestionar dependencias y los recursos que utilizaran en todo el proyecto, el SBT tiene la posibilidad de trabajar con Java o Scala. (Gracia, 2013)
- **kantan.csv:** Según el propio repositorio de kantan en GitHub escrito por Nicolas Rinaudo “kantan es una biblioteca para el análisis y la serialización de CSV escrita en el lenguaje de programación Scala”. (Rinaudo, 2015)


Kantan será de gran ayuda en el proyecto en la parte de programación funcional pues esta librería nos permitirá realizar la lectura y escritura de archivo con extensión .csv.

5.7 Slick

Slick es una biblioteca moderna de consulta y acceso a bases de datos para Scala. Le permite trabajar con datos almacenados casi como si estuviera usando colecciones de Scala y al mismo tiempo le brinda control total sobre cuándo ocurre un acceso a la base de datos y qué datos se transfieren. Puede escribir las consultas de su base de datos en Scala en lugar de SQL, beneficiándose así de la comprobación estática, la seguridad en tiempo de compilación y la composicionalidad de Scala. Slick presenta un compilador de consultas extensible que puede generar código para diferentes backends.

6. Procedimiento de programación funcional y reactiva

Para llevar a cabo las consultas debemos realizar las importaciones de algunas librerías como lo son `java.io.file` y `kantan` las cual nos permitirán trabajar con los archivos con extensión `csv`, además de algunas otras librerías como el `scala.io.Codec` que nos va a permitir definir el tipo de codificación con la que vamos a trabajar, estas importaciones se las puede observar en la ilustración 1.



```
1 import java.io.File
2 import kantan.csv._
3 import kantan.csv.ops._
4 import kantan.csv.generic._
5
6 import scala.collection.immutable.ListMap
7 import scala.io.Codec
```

Ilustración 1 Importaciones para leer y escribir archivos csv

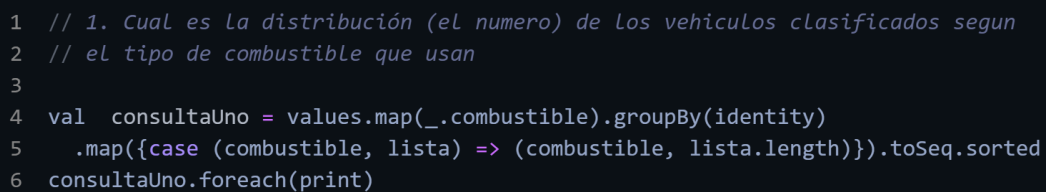
6.1 Proceso consulta uno

Antes de explicar el proceso de la primera consulta debemos aclarar la función de algunos métodos que nos ayudaran para la misma.

- Map: recorre una lista y mediante la condición que le demos nos devuelve una nueva con el mismo número de elementos.
- GroupBy: agrupa secuencias mediante el valor que le especifiquemos y nos devuelve un mapa, clave-valor
- Map-Case: permite alterar valores, guardando en nuevas tuplas
- ToSeq: permite transformar una estructura de datos en una secuencia
- Sorted: ordena los datos de una secuencia en orden ascendente

La palabra reservada `val` nos permite declarar una variable mutable, la función `map` actúa como un buscador de alguna columna especifica que en este caso seria sobre el combustible, el `groupBy(identity)` cumple la función de agrupar los datos según el tipo de combustible y por otro lado tenemos el `map-case` que nos va a permitir formar una tupla con datos específicos que necesitamos y sobre todo calcular el número de vehículos y por último usamos la función. `toSeq.sorted` que nos permite transformar la tupla en una secuencia de datos para así poder ordenarlos.

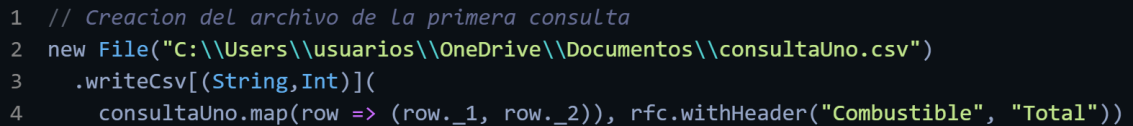
En la ilustración 2 podemos observar el código en scala que se utilizó para realizare la consulta uno.



```
1 // 1. Cual es la distribución (el numero) de los vehiculos clasificados segun
2 // el tipo de combustible que usan
3
4 val consultaUno = values.map(_.combustible).groupBy(identity)
5   .map({case (combustible, lista) => (combustible, lista.length)}).toSeq.sorted
6 consultaUno.foreach(print)
```

Ilustración 2 Código para ejecutar la consulta uno

Para realizar la creación del archivo csv directamente desde scala lo podemos hacer con el código de la ilustración 3



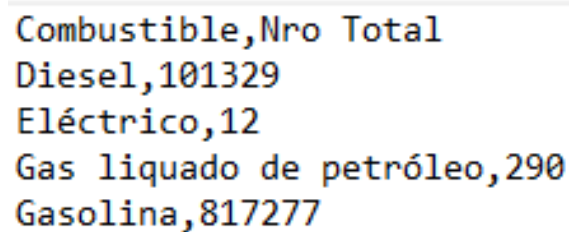
```

1 // Creacion del archivo de la primera consulta
2 new File("C:\\Users\\usuarios\\OneDrive\\Documentos\\consultaUno.csv")
3   .writeCsv[(String,Int)](
4     consultaUno.map(row => (row._1, row._2)), rfc.withHeader("Combustible", "Total"))

```

Ilustración 3 Código para la creación del archivo correspondiente a la consulta uno

Los resultados que nos arroja esta consulta en el archivo .csv son los mostrados en la ilustración 4



```

Combustible,Nro Total
Diesel,101329
Eléctrico,12
Gas liquado de petróleo,290
Gasolina,817277

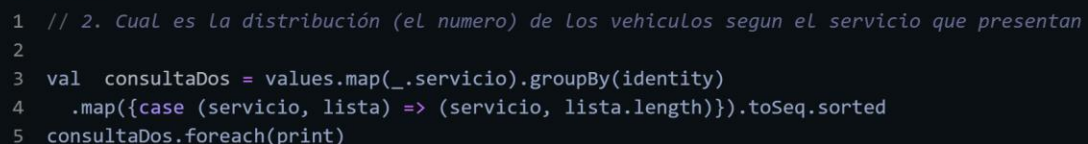
```

Ilustración 4 Resultados Consulta Uno

6.2 Proceso consulta dos

- **Filter:** permite evaluar de manera específica un tipo de parámetro, en este caso podemos visualizar que estamos buscando solo las motocicletas.
- **GroupBy:** funciona para agrupar según un parámetro específico, en la imagen podemos analizar que lo usamos para ordenar según las provincias.
- **Map-Case:** recibe valores para luego atribuir otros en función de tuplas, analizamos que se dividen los datos según provincia y se calcula el total de motocicletas por provincia.

En la ilustración 4 podemos observar el código en scala que se utilizó para realizare la consulta dos.



```

1 // 2. Cual es la distribución (el numero) de Los vehiculos segun el servicio que presentan
2
3 val consultaDos = values.map(_.servicio).groupBy(identity)
4   .map({case (servicio, lista) => (servicio, lista.length)}).toSeq.sorted
5 consultaDos.foreach(print)

```

Ilustración 5 Código para ejecutar la consulta dos

Para realizar la creación del archivo csv directamente desde scala lo podemos hacer con el código de la ilustración 5.



```

1 // Creacion del archivo de la segunda consulta
2 new File("C:\\Users\\usuarios\\OneDrive\\Documentos\\consultaDos.csv")
3   .writeCsv[(String,Int)](
4     consultaDos.map(row => (row._1, row._2)), rfc.withHeader("Servicio", "Total"))

```

Ilustración 6 Código para la creación del archivo correspondiente a la consulta dos

Los resultados que nos arroja esta consulta en el archivo .csv son los mostrados en la ilustración 7

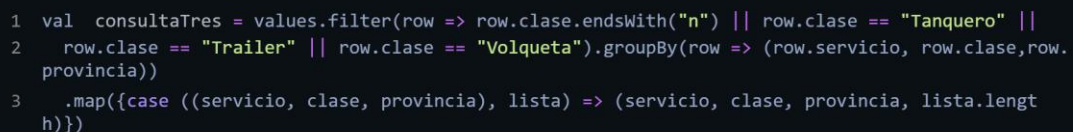
Servicio	Total
ALQUILER	35738
ESTADO	6800
MUNICIPIO	2453
OTROS	156
PARTICULAR	873725
SECTORES SECCIONALES	36

Ilustración 7 Resultados consulta Dos

6.3 Proceso consulta tres

- **Filter:** devuelve una nueva lista con los elementos de la lista original que cumplen el predicado.
- **Map:** aplica una función a todos los elementos de la lista, devolviendo una nueva lista con el resultado.
- **GroupBy:** Agrupa secuencias y recopila en tuplas.
- **Map-Case:** Permite alterar valores, guardando en nuevas tuplas
- **Filter** es usado para obtener únicamente la información de tanquero, trailer, volqueta para ser agrupados con un groupBy según su servicio, clase y provincia, utilizando un map case que nos ayudara a obtener la información específica calculando el número de servicios que existen por provincia.

En la ilustración 6 podemos observar el código en scala que se utilizó para realizare la consulta



```
1 val consultaTres = values.filter(row => row.clase.endsWith("n") || row.clase == "Tanquero" ||  
2   row.clase == "Trailer" || row.clase == "Volqueta").groupBy(row => (row.servicio, row.clase,row.  
   provincia))  
3   .map({case ((servicio, clase, provincia), lista) => (servicio, clase, provincia, lista.length  
   h)})}
```

Ilustración 8 Código para ejecutar la consulta tres

Para realizar la creación del archivo csv directamente desde scala lo podemos hacer con el código de la ilustración 7



```

1 new File("C:\\Users\\usuarios\\OneDrive\\Documentos\\consultaCincoParteDos.csv")
2   .writeCsv[(String,String,String,Int)](
3     consultaTres.map(row => (row._1, row._2, row._3, row._4)), rfc.withHeader("Se
    rvicio", "Clase", "Servicion", "Total"))

```

Ilustración 9 Código para la creación del archivo correspondiente a la consulta tres

Los resultados que nos arroja esta consulta en el archivo .csv son los mostrados en la ilustración 10

```

Servicio,Clase,Provincia,Total
ESTADO,Volqueta,LOS RÍOS,31
MUNNICIPIO,Volqueta,IMBABURA,14
ALQUILER,Trailer,PICHINCHA,318
MUNNICIPIO,Camión,SANTO DOMINGO DE LOS TSÁCHILAS,18
PARTICULAR,Camión,GUAYAS,10963
ALQUILER,Tanquero,CAÑAR,7
ESTADO,Camión,MORONA SANTIAGO,2
ESTADO,Camión,LOJA,9
ALQUILER,Camión,ZAMORA CHINCHIPE,44
ALQUILER,Volqueta,SANTA ELENA,1
MUNNICIPIO,Volqueta,SUCUMBÍOS,7
PARTICULAR,Tanquero,COTOPAXI,58
MUNNICIPIO,Volqueta,COTOPAXI,18
PARTICULAR,Camión,SANTA ELENA,245
PARTICULAR,Camión,LOS RÍOS,3470
ALQUILER,Tanquero,ZAMORA CHINCHIPE,2
ALQUILER,Tanquero,PICHINCHA,75
MUNNICIPIO,Volqueta,CARCHI,13
MUNNICIPIO,Tanquero,GUAYAS,21
ESTADO,Volqueta,CAÑAR,7
ALQUILER,Trailer,IMBABURA,61
ALQUILER,Camión,CHIMBORAZO,214
SECTORES SECCIONALES,Volqueta,PASTAZA,4
PARTICULAR,Tanquero,AZUAY,61

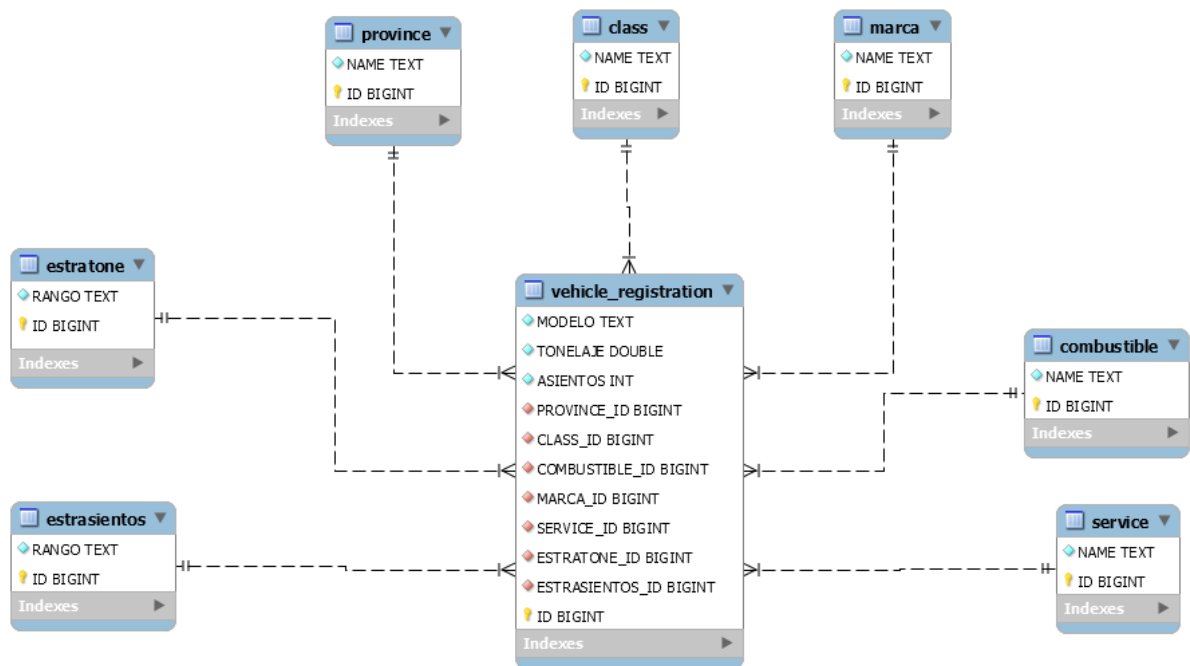
```

Ilustración 10 Resultados Consulta Tres

6.4 Proceso consulta cuatro

Para realizar esta consulta utilizamos los conocimientos adquiridos de Slick para obtener los datos desde una base de datos previamente cargada. En la cual tenemos un modelo de entidad relación de las tablas con las cuales se encuentran los datos de matriculación vehicular.

El modelo de entidad relación es el siguiente:



En la ilustración 11 se presenta el procedimiento para realizar la consulta Cuatro.

```

1 val consultaCuatro = registrationQry.
2 filter(k => k.classId === getClassIdByName("Camioneta") && k.provinceId === getProvinceIdByName(
  "LOJA")).
3   groupBy(_.serviceId).map({case(service, list) => (service,list.size)})
4
5 val resultadoCuatro = exec(consultaCuatro.result).map({case(service, total) => (getServiceNameBy
  Id(service),total)}).sorted
6
7 new File("C:\\Users\\Usuario iTC\\OneDrive\\Documentos\\ConsultaCuatro.csv") writeCsv[(String, I
  nt)](resultadoCuatro.map(row => (row._1, row._2)), rfc.withHeader("Service", "Total"))
  
```

Ilustración 11 Consulta Cuatro

La variable `registrationQry` es con la cual se puede acceder a los datos de la tabla `vehicle_registration`.

Para realizar esta consulta utilizamos:

- **Filter:** devuelve una nueva lista con los elementos de la lista original que cumplen las condiciones:
 - La clase debe ser Camioneta
 - La provincia debe ser Loja
- **GroupBy:** Agrupa secuencias y recopila en tuplas. En este caso nos agrupará por el id del servicio
- **Map-Case:** Permite alterar valores, guardando en nuevas tuplas, en este caso se guarda como (Servicio, Total de vehículos de la provincia de Loja que tienen como clase Camioneta.)

Para ejecutar esta consulta utilizamos la función `exec` previamente definida para que la consulta se la realice con la base de datos y le hacemos un `map-case` para cambiar el id de servicio por su nombre para luego realizar la creación del archivo.

6.5 Proceso consulta cinco

Al igual que la consulta cuatro, para realizar la consulta cinco, lo que vamos a utilizar es la variable `registrationQry`, la cual nos va a dar la estructura de nuestra tabla de registro de vehículos.



Ilustración 12 Consulta Uno en Scala

Para realizar esta consulta utilizamos:

- **Filter:** devuelve una nueva lista con los elementos de la lista original que cumplen las condiciones:
 - El combustible sea Gasolina
 - La marca sea TOYOTA
- **GroupBy:** Agrupa secuencias y recopila en tuplas. En este caso nos agrupará por el id de la provincia
- **Map-Case:** Permite alterar valores, guardando en nuevas tuplas, en este caso se guarda como (Provincia, Total de vehículos de la provincia.)

Para ejecutar esta consulta utilizamos la función `exec` previamente definida para que la consulta se la realice con la base de datos y le hacemos un `map-case` para cambiar el id de provincia por su nombre para luego realizar la creación del archivo

6.6 Proceso consulta seis

Esta consulta utiliza la variable de registrationQry que nos va a permitir usar nuestra tabla de que contiene los datos del registro de vehículos.

```

1 val consultaSeis =registrationQry.filter(k => k.asientos === 4 && k.classId === getClassIdByName(
  "Automóvil")
2 && k.combustibleId === getCombustibleIdByName("Gasolina")).groupBy(.marcaId)
3   .map({ case (marca, list) => (marca, list.size)})
4
5 val resultadoSeis = exec(consultaSeis.result).map({ case (marca, total) => (getMarcaNameById(marc
  a), total) }).sortBy(_.2)
6
7 new File("C:\Users\Usuario iTC\OneDrive\Documentos\ConsultaSeis.csv") writeCsv[(String, Int)](
8   resultadoSeis.map(row => (row._1, row._2)), rfc.withHeader("Marca", "Total"))

```

- **Filter:** devuelve una nueva lista con los elementos de la lista original que cumplen el predicado.
 - Asientos sean igual a 4
 - Class sea “Automovil”
 - Combustible sea “Gasolina”
- **GroupBy:** Agrupa secuencias y recopila en tuplas.
 - Se agrupa según la marca
- **Map-Case:** Permite alterar valores, guardando en nuevas tuplas
 - Se crea una tupla de la marca y la cantidad de vehículosEn la ilustración 7 podemos observar el código en scala que se utilizó para realizare la consulta

Usamos el exec para que se pueda conectar con la base de datos y un map-case que permitiría obtener el nombre de la marca y el total de vehículos que era de 4 asientos con gasolina en una tupla para luego usar un sortBy para ordenarlo según su marca.

7. Procedimiento para realizar las visualizaciones en vega-lite:

Primeramente, para lograr una visualización de los datos mediante gráficos en una página web debemos hacer el uso de un documento HTML en el que importaremos las scripts necesarias para poder trabajar con vega-lite, este proceso lo realizaremos en el head del HTML, los scripts que importaremos son los que se pueden observar en la ilustración 8.



Ilustración 13 Scripts para trabajar vega en HTML

Luego de esto en la parte del body del HTML definiremos contenedores en los cuales se encontrará cada uno de los gráficos que deseamos representar, en este caso habrá un gráfico por cada consulta por lo que definiremos 3 contenedores, como se lo puede visualizar en la ilustración 9.



Ilustración 14 Contenedores de los gráficos de vega

7.1 Proceso consulta uno

En la parte del body del HTML realizamos la creación de un script en el que definiremos las constantes en las que estarán las características de las gráficas, como vamos a definir inicialmente la primera consulta creamos una constante llamada `consultaUno` y dentro de esta los siguientes parámetros:

- **\$schema:** Dentro de este parámetro se ingresa el enlace de GitHub de la versión actual de vega-lite.
- **data:** Dentro de este parámetro podemos escribir los datos con los que queremos trabajar o mediante el uso de otro parámetro llamado “url” la dirección en la cual se encuentra el archivo con los datos que queremos leer.
- **width:** Permite definir el ancho que ocupara la gráfica.
- **height:** Permite definir el largo que ocupara la gráfica.
- **mark:** Permite definir el tipo de elemento que se va a representar en la gráfica, ya sean barras, puntos, u otros.
- **encoding:** Permite codificar los parámetros que queremos que representen a la parte “x” y a la parte “y” de la gráfica, así como diferentes propiedades para cada uno de estos como lo son:
 - **field;** Se escribe cual es el nombre de la columna con la que trabajaremos.
 - **type:** Define si los datos de la columna son cuantitativos o nominales.
 - **scale:** Permite transformar el dominio de un conjunto de datos para que podamos trabajar con un rango de valores visuales óptimos.

Finalmente, para que la gráfica se pueda visualizar, luego de la creación y definición de la constante definiremos `vegaEmbed` y dentro de los paréntesis de continuación haremos una relación entre el contenedor que definimos anteriormente y separado por una coma el nombre de la constante, todo el proceso descrito lo podemos visualizar en la ilustración 10.



```
1  const consultaUno = {
2    $schema: 'https://vega.github.io/schema/vega-lite/v4.json',
3    data: {
4      url: "data/consultaUno.csv"
5    },
6    width: 320,
7    height: 200,
8    mark: 'bar',
9    encoding: {
10      x: {
11        field: 'Combustible',
12        type: 'nominal'},
13      y: {
14        field: 'Nro Total',
15        type: 'quantitative',
16        scale: {type: 'symlog'}
17      },
18    }
19 };
20 vegaEmbed('#vis1', consultaUno);
```

Ilustración 15 Código para realizar la visualización de la consulta uno

Como podemos observar y dando un recuento de lo descrito anteriormente primeramente creamos la constante con la que vamos a trabajar , dentro de esta en la parte de “\$schema” se encuentra el enlace a la librería de vega-lite, en la parte de “data” tenemos descrito el directorio en el que se encuentra el archivo con el que queremos trabajar, la gráfica ocupara un ancho de 320 pixeles por un alto de 200 pixeles, en la parte de “mark” le decimos que haga la representación mediante barras, seguido en la parte de “encoding” definimos que “x” tome como datos los que se encuentran en la columna de Combustible y determinamos a este mismo como un valor nominal, en la parte de “y” hacemos que esta tome los datos de la columna de Nro Total y lo determinamos como cuantitativo, en la parte de “scale” le decimos que queremos algo parecido a una escala logarítmica, luego de esto fuera de la definición de la constante colocamos el vegaEmbed relacionando el contenedor con la constante, finamente obtenemos el gráfico de la ilustración 11 :

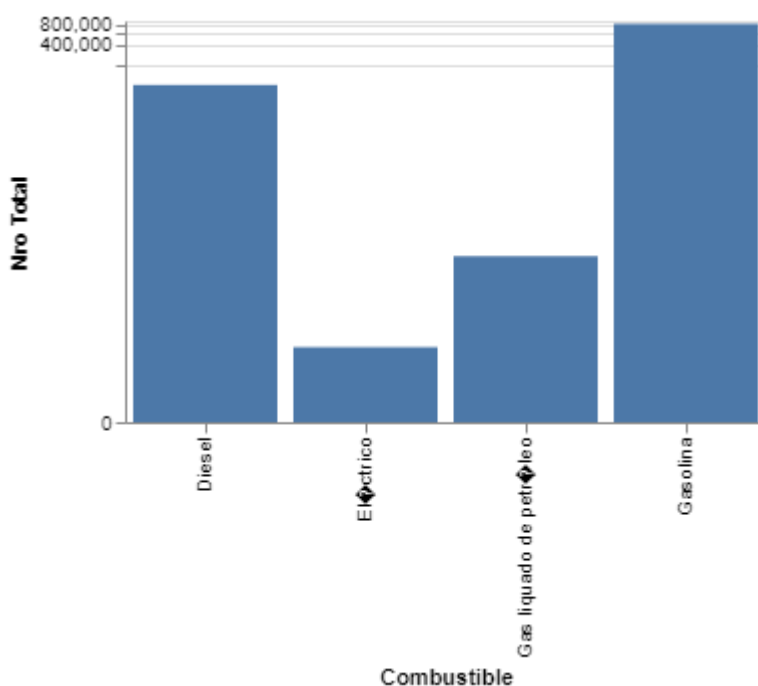


Ilustración 16 Visualización de la consulta uno

7.2 Proceso consulta dos

El proceso es sumamente similar al descrito en la consulta uno solamente que en este caso en la parte de “data” hacemos referencia a un archivo diferente, en la parte de “mark” le decimos que queremos hacer las visualizaciones de forma de áreas y le damos un dato adicional que es “color” para darle un color diferente al que viene por defecto, en la parte de “encoding” al definir “x” le decimos que tome los datos de la columna Servicio y lo definimos como nominal, mientras que para “y” le decimos que tome la columna Total y lo definimos como cuantitativo, eso se puede observar de una mejor forma en la ilustración

```
1  const consultaDos = {
2    $schema: 'https://vega.github.io/schema/vega-lite/v4.json',
3    data: {
4      "url": "data/consultasDos.csv"
5    },
6    "width": 320,
7    "height": 200,
8    mark: {type: 'area', color: '#7c8500'},
9    encoding: {
10      x: {
11        field: 'Servicio',
12        type: 'nominal'},
13      y: {
14        field: 'Total',
15        type: 'quantitative',
16        scale: {type: 'symlog'}
17      },
18    }
19 };
20 vegaEmbed('#vis2', consultaDos);
```

Ilustración 17 Código para realizar la visualización de la consulta dos

La visualización que obtenemos de la consulta dos es la que podemos observar en la ilustración 18:

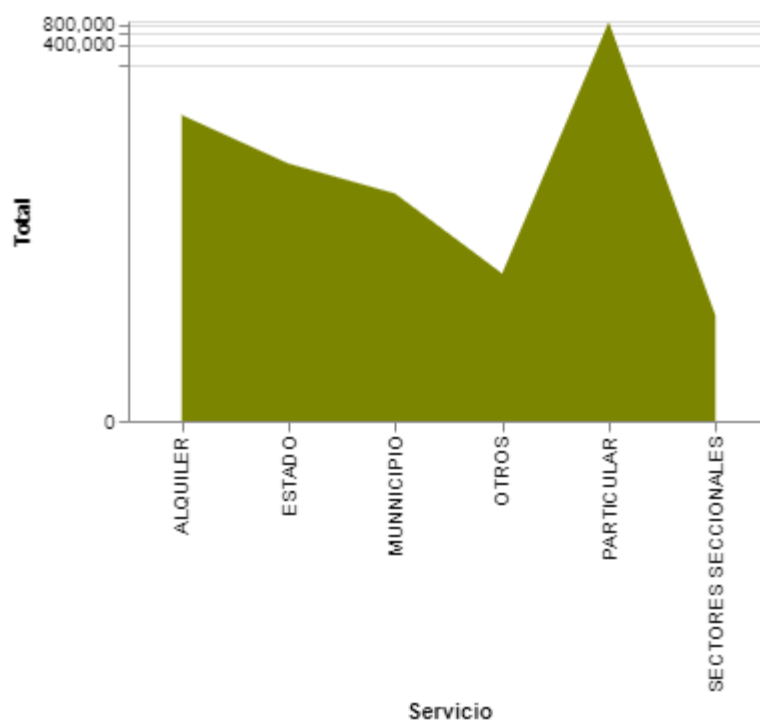


Ilustración 18 Visualización de la consulta dos

7.3 Proceso consulta tres

En la parte del código no hay muchas cosas nuevas pero si algunas importantes a resaltar como que vamos a trabajar con varias tablas puesto que tenemos varias provincias es por esto que en la parte de “encoding” agregamos un nuevo parámetro llamado “row” donde le decimos que estas tablas se mostraran cada una abajo de la otra, a forma de filas y lo harán en relación a las Provincias que haya, también le decimos que este dato es nominal, otro parámetro que agregamos de igual manera en “encoding” es “color” que en este caso nos sirve para decirle en relación a que va a colorear las barras, en este caso le decimos que lo va a hacer en relación a la columna de Clase, y le decimos que es de tipo nominal, todo esto se puede observar en la ilustración 14 .

A screenshot of a code editor with a dark background and light-colored text. The code is written in JavaScript and defines a Vega-Lite visualization configuration. It includes a schema URL, data source, mark type, and encoding for row, x, y, and color. The code is as follows:

```
1  const consultaTres = {
2    $schema: 'https://vega.github.io/schema/vega-lite/v4.json',
3    data: {"url": 'data/consultaTres.csv'},
4    mark: 'bar',
5    encoding: {
6      row: {"field": 'Provincia', "type": 'nominal'},
7      x: {"field": 'Total', "type": 'quantitative'},
8      y: {"field": 'Servicio', "type": 'nominal'},
9      color: {"field": 'Clase', "type": 'nominal'}
10   }
11 };
12 vegaEmbed('#vis3', consultaTres);
```

Ilustración 19 Código para realizar la visualización de la consulta tres

Una muestra de la visualización que podemos obtener de la consulta tres es la que se observa en la ilustración 15.

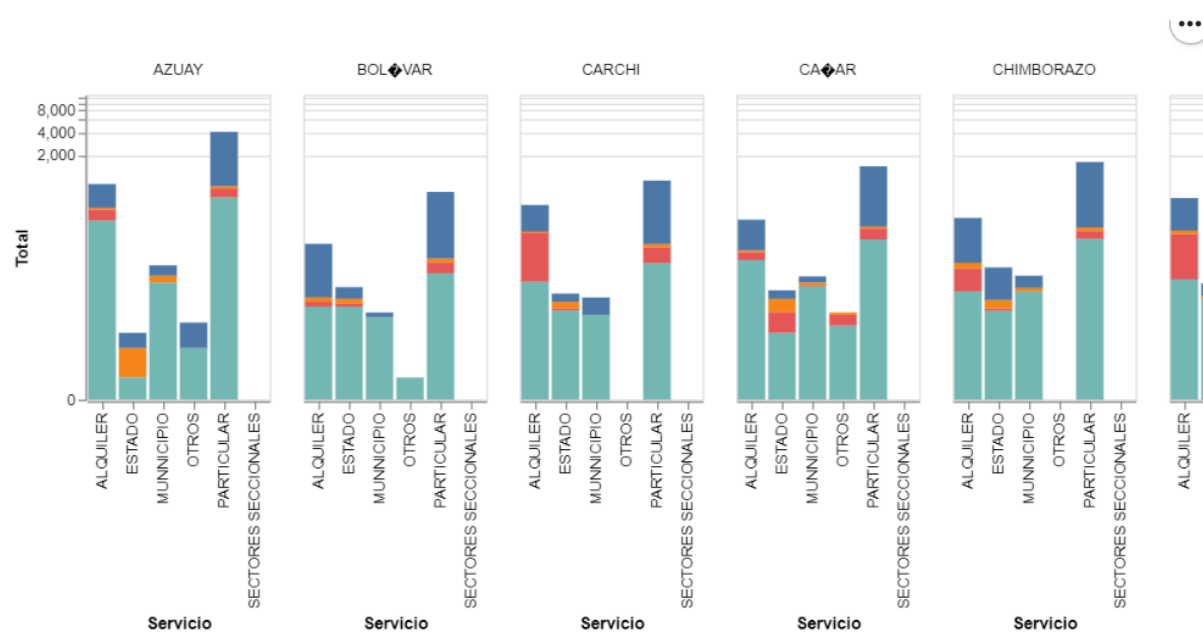


Ilustración 20 Visualización de la consulta tres

7.4 Proceso consulta cuatro

```

1  const consultaCuatro = {
2    $schema: 'https://vega.github.io/schema/vega-lite/v4.json',
3    data: {
4      'url': 'data/ConsultaCuatro.csv'
5    },
6    'width': 500,
7    'height': 500,
8    layer:[
9      {mark:{type: 'arc', innerRadius: 100, stroke: 'black'}},
10     {mark: {type:'text', radiusOffset: 20},
11      encoding: {text: {field: 'Total', type: 'quantitative'}}
12    }
13  ],
14  encoding: {
15    theta: {field: 'Total', type: 'quantitative', stack: true},
16    radius: {
17      field: 'Total',
18      scale: {type: 'sqrt', zero: true, rangeMin:240, domain: {unionWith: [0, 5]}}
19    },
20    color: {field: 'Service', type: 'nominal'}
21  },
22  view: {stroke: null}
23
24 };
25 vegaEmbed('#vis4', consultaCuatro);

```

Ilustración 21 Código de gráfica cuatro

En relación con las gráficas anteriores, ahora tenemos la novedad de que la gráfica tendrá una forma circular.

Algo que nos sirve para hacer la gráfica más descriptiva es tener los datos totales a lado de su respetivo pedazo de gráfica, para lograr esto debemos definir un layer, esta es como una capa en la cual se encontrara la estructura de la gráfica para que así podamos definir los totales en su respectivo sitio, en el layer primero definimos dos marks, el primero con la estructura del circulo, y el segundo con la del texto, en la parte de encodign definimos en función de que se creara la figura, en este caso en función de la columna Total de nuestro documento csv.

En la parte de encoding es donde definiremos nuestro circulo principal, primeramente, definimos theta, en esta establecemos en función de qué necesitamos las longitudes del arco del circulo, en este caso en función de la columna de Total de nuestro documento csv, luego

de esto definimos un radio para el circulo en la parte de radius, de igual forma le establecemos en función de que columna trabajará y definimos scale, para poder darle una escala a los datos, en este apartado definimos algunos parametros que nos ayudaran a que la gráfica no esté tan dispareja.

Finalmente, definimos color y la columna con la cual trabajara para representar los datos, si es requerido definimos view que nos sirve para darle un color al margen de la gráfica, para concluir obtenemos la gráfica de la siguiente visualización.

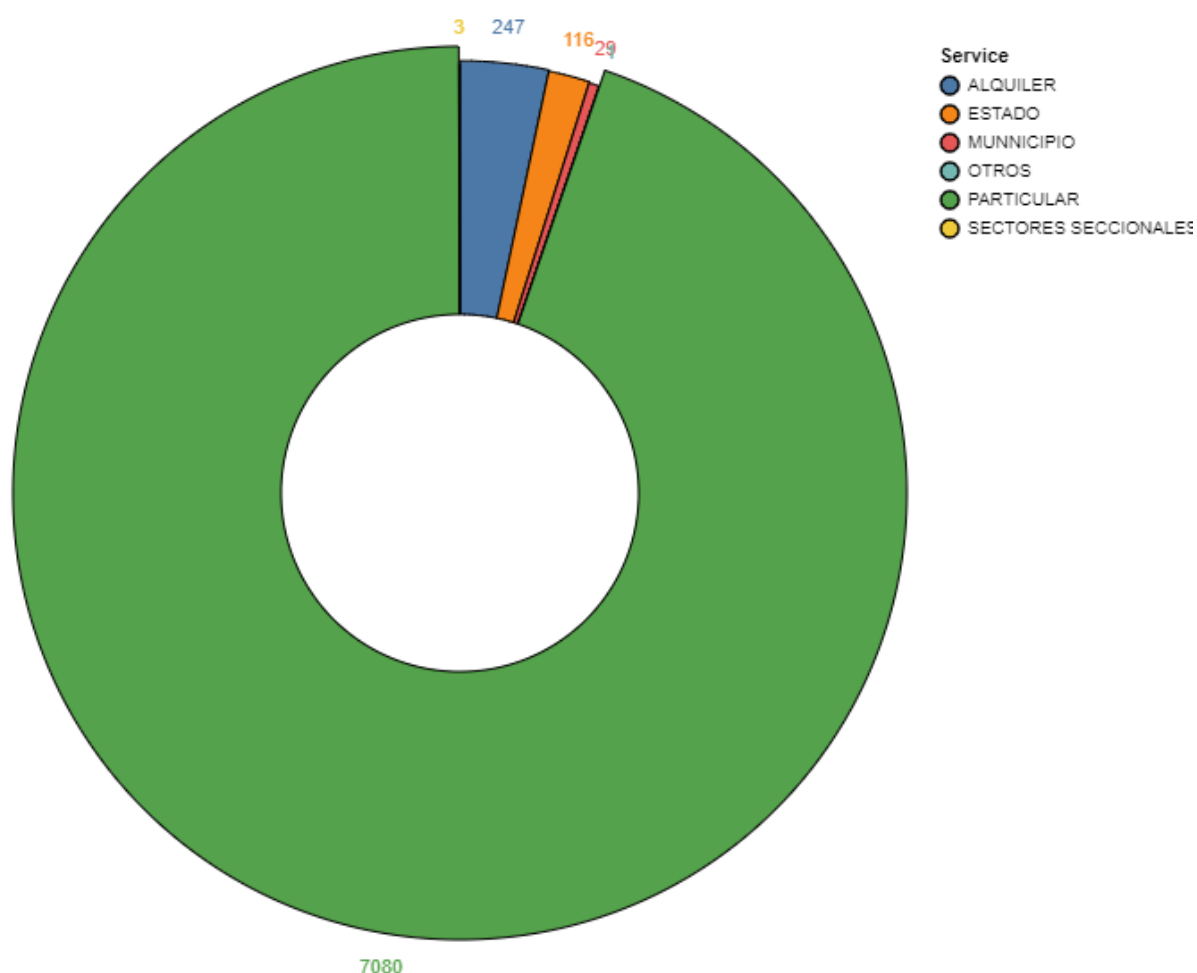


Ilustración 22 Resultado de consulta cuatro

7.5 Proceso consulta cinco

```

1  const consultaCinco = {
2    data: {
3      'url': 'data/ConsultaCinco.csv'
4    },
5    'width': 500,
6    'height': 500,
7    encoding: {
8      theta: {field: 'Total', type: 'quantitative', stack: true},
9      color: {field: 'Province', type: 'nominal'}
10   },
11   layer: [{
12     mark: {type: "arc", outerRadius: 200, innerRadius: 50}
13   },
14   {
15     mark: {type: 'text', radius: 220},
16     encoding: {text: {field: 'Total', "type": 'quantitative'}}
17   }
18  ],
19   "view": {"stroke": null}
20 };
21
22 vegaEmbed('#vis5', consultaCinco);

```

Ilustración 23 Código de la gráfica cinco

Para lograr realizar esta grafica en la parte de encoding es donde definiremos nuestro circulo principal, primeramente, definimos theta, en esta establecemos en función de qué necesitamos las longitudes del arco del circulo, en este caso en función de la columna de Total de nuestro documento csv, luego de esto definimos un radio para el circulo en la parte de radius, de igual forma le establecemos en función de que columna trabajará.

Finalmente, definimos color y la columna con la cual trabajara para representar los datos, si es requerido definimos view que nos sirve para darle un color al margen de la gráfica.

Debemos definir un layer, esta es una capa en la cual se encontrará la estructura de la gráfica para que así podamos definir los totales en su respectivo sitio, en el layer primero definimos

un mark, con la estructura del círculo, segundo con la del texto, en la parte de encoding definimos en función de que se creara la figura, en este caso en función de la columna Total de nuestro documento csv.

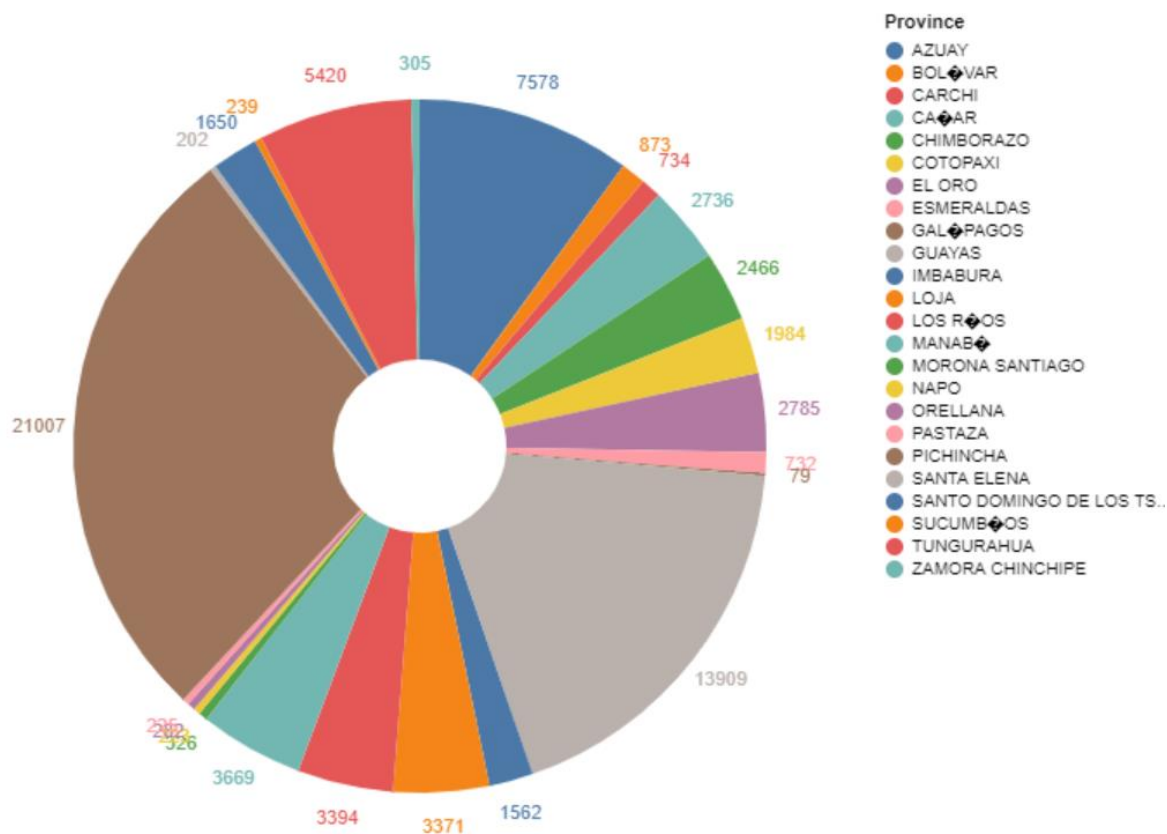


Ilustración 24 Resultado de la consulta cinco

7.6 Proceso consulta seis

```

1  const consultaSeis = {
2    $schema: 'https://vega.github.io/schema/vega-lite/v4.json',
3    data: {
4      'url': 'data/consultaSeis.csv'
5    },
6    'width': 500,
7    'height': 500,
8    layer:[
9      {mark:{type: 'arc', innerRadius: 20, stroke: 'blue'}}, //inner radius es el radio del circulo
10     o de adentro
11     {mark: {type:'text', radiusOffset: 20}, //radiusOffset es para el radio de los nombres
12       encoding: {text: {field: 'Total', type: 'quantitative'}}
13     },
14     encoding: {
15       theta: {field: 'Total', type: 'quantitative', stack: true},
16       radius: {
17         field: 'Total',
18         scale: {type: 'sqrt', zero: true, rangeMin:240, domain: {unionWith: [0, 5]}}
19       },
20       color: {field: 'Marca', type: 'nominal'}
21     },
22     view: {stroke: null}
23   };
24   vegaEmbed('#vis6', consultaSeis);

```

Ilustración 25 Código de gráfica seis

Tenemos una gráfica en forma circular similar a una dona.

Para hacer más descriptiva la gráfica debemos obtener datos totales a lado de su respectivo pedazo de gráfica, para lograr esto debemos definir un layer, esta es como una capa en la cual se encontrara la estructura de la gráfica para que así podamos definir los totales en su respectivo sitio, en el layer primero definimos dos marks, el primero con la estructura del circulo, y el segundo con la estructura del texto, en la parte de encoding definimos en función de que se creara la figura, en este caso en función de la columna Total de nuestro documento csv.

En la parte de encoding es donde definiremos nuestro circulo principal, primeramente, definimos theta, en esta establecemos en función de qué necesitamos las longitudes del arco del circulo, en este caso en función de la columna de Total de nuestro documento csv, luego de esto definimos un radio para el circulo en la parte de radius y sea el innerRadius para

dar el radio del círculo interno y radiusOffset para el radio de los nombres, de igual forma le establecemos en función de que columna trabajará y definimos scale, para poder darle una escala a los datos, en este apartado definimos algunos parametros que nos ayudaran a que la gráfica no esté tan dispareja.

Finalmente, definimos color y la columna con la cual trabajara para representar los datos, si es requerido definimos view que nos sirve para darle un color al margen de la gráfica.

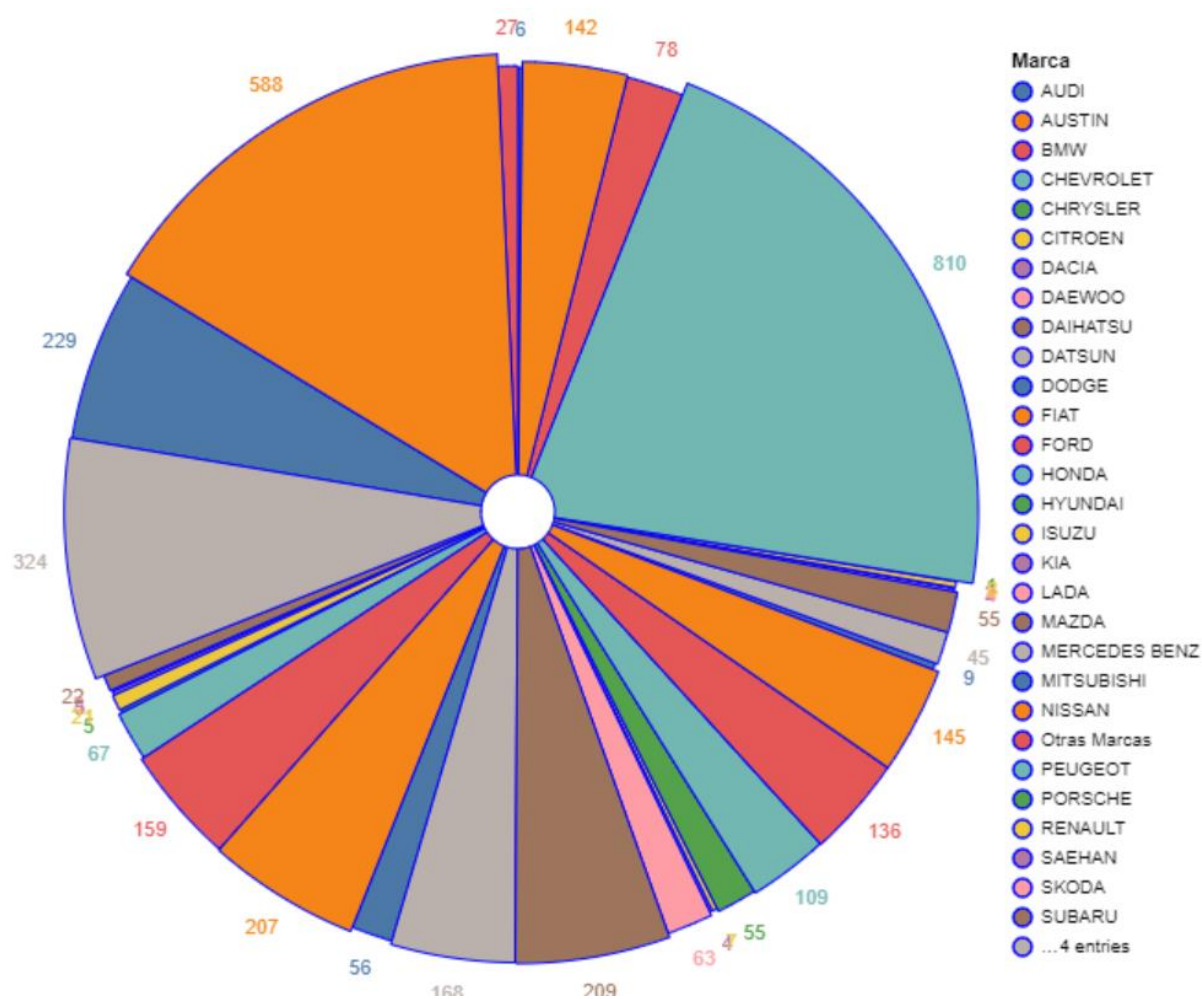


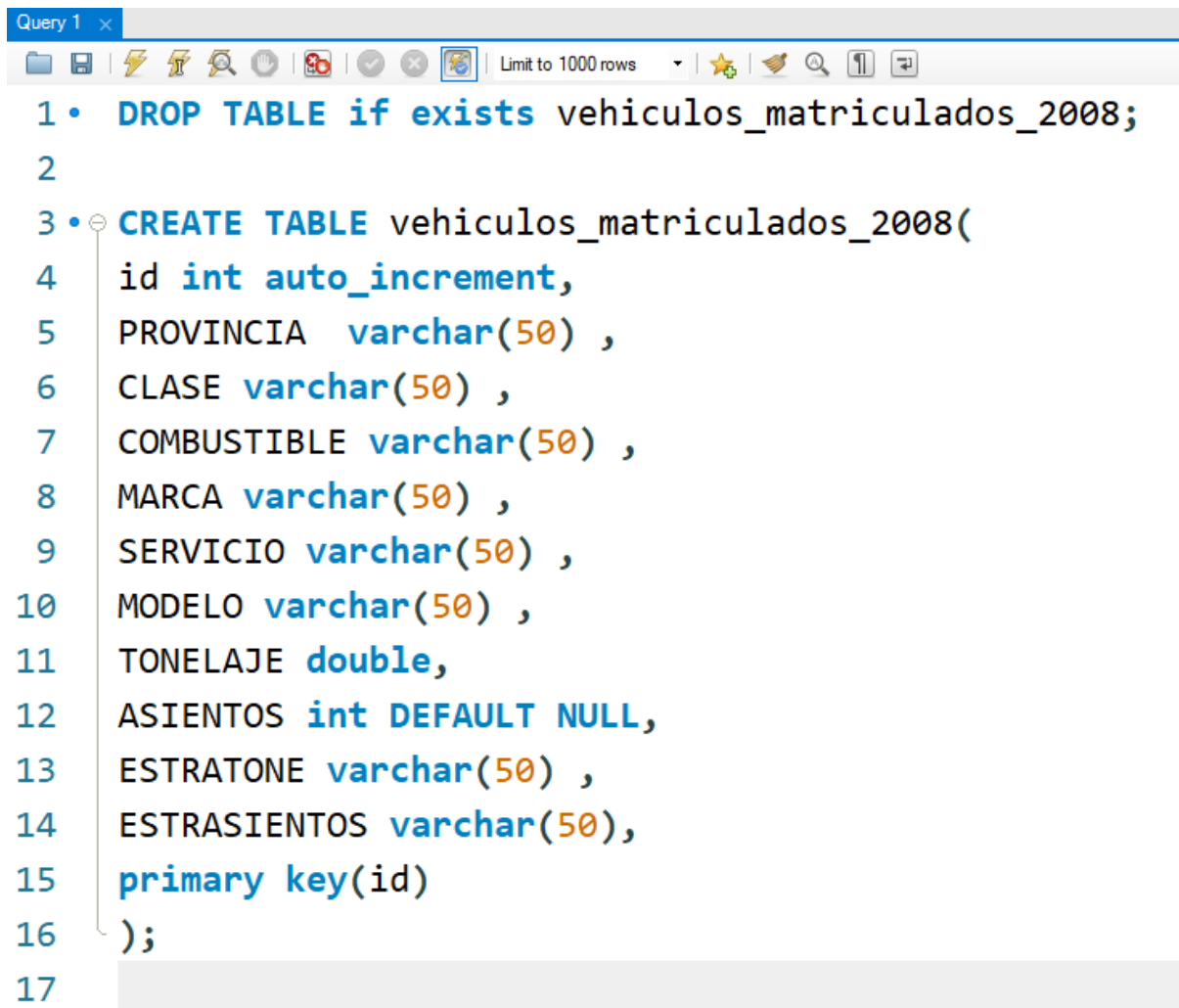
Ilustración 26 Grafica de consulta seis

8. Procedimiento de base de datos

8.1 Proceso de creación del script de datos matriculación vehicular

Creamos una base de datos llamada `vehículos_matriculados_2008` con sus respectivos atributos, añadiéndole una clave primaria (`id`) con la propiedad de auto incremento para que la importación de los datos sea más rápida.

Como se puede observar en la Ilustración 16

The image shows a screenshot of a SQL query editor window titled 'Query 1'. The editor has a toolbar with various icons for file operations, execution, and navigation. The SQL script is as follows:

```
1 • DROP TABLE if exists vehiculos_matriculados_2008;
2
3 • CREATE TABLE vehiculos_matriculados_2008(
4   id int auto_increment,
5   PROVINCIA varchar(50) ,
6   CLASE varchar(50) ,
7   COMBUSTIBLE varchar(50) ,
8   MARCA varchar(50) ,
9   SERVICIO varchar(50) ,
10  MODELO varchar(50) ,
11  TONELAJE double,
12  ASIENTOS int DEFAULT NULL,
13  ESTRATONE varchar(50) ,
14  ESTRASIENTOS varchar(50),
15  primary key(id)
16 );
17
```

Ilustración 27 Creación de tablas en el script

Importamos los datos del archivo .csv que adquirimos de la base de datos del INEC sobre matriculación de vehículos en el año 2008.

Table Data Import

Configure Import Settings

Detected file format: csv

Encoding: latin2 (iso8859-2)

Columns:

- ☒ MARCA text
- ☒ SERVICIO text
- ☒ MODELO text
- ☒ TONELAJE text
- ☒ ASIENTOS int
- ☒ ESTRATONE text
- ☒ ESTRASIENTOS text

PROVINCIA	CLASE	COMBUSTI...	MARCA	SERVICIO	MODELO	TONELAJE	ASIENTOS	ESTRATONE	ESTRAS
AZUAY	Camioneta	Gasolina	CHEVROLET	ESTADO	2003 Y ANT...	1	3	1/4 A 3	1 A 10
BOLÍVAR	Jeep	Gasolina	CHEVROLET	ESTADO	2005	.8	5	1/4 A 3	1 A 10
BOLÍVAR	Jeep	Gasolina	CHEVROLET	ESTADO	2005	.8	5	1/4 A 3	1 A 10
BOLÍVAR	Jeep	Gasolina	CHEVROLET	ESTADO	2005	.8	5	1/4 A 3	1 A 10

< Back Next > Cancel

Ilustración 28 Importación de datos de matriculación

Esperamos que los datos se importen correctamente, para luego realizar las consultas requeridas con estos datos.

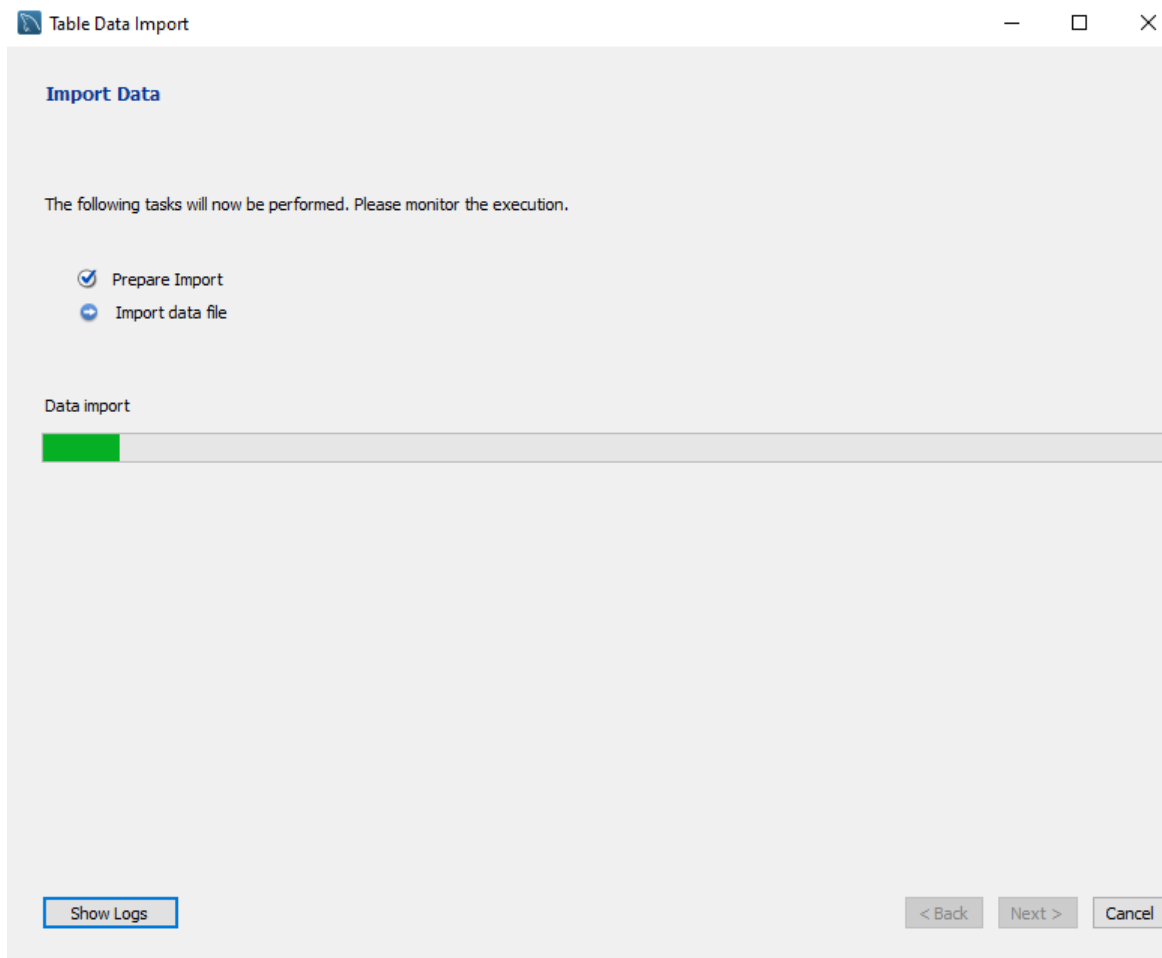


Ilustración 29 Proceso de carga de datos de matriculación

Comprobamos que los datos se hayan importado correctamente haciendo una visualización de estos en MySQL.

	id	PROVINCIA	CLASE	COMBUSTIBLE	MARCA	SERVICIO	MODELO	TONELAJE	ASIENTOS	ESTRATONE	ESTRASIENTOS
▶	1	AZUAY	NULL	Gasolina	CHEVROLET	ESTADO	2003 Y ANTERIORES	1	3	1/4 A 3	1 A 10
	2	BOLÍVAR	NULL	Gasolina	CHEVROLET	ESTADO	2005	0.8	5	1/4 A 3	1 A 10
	3	BOLÍVAR	NULL	Gasolina	CHEVROLET	ESTADO	2005	0.8	5	1/4 A 3	1 A 10
	4	BOLÍVAR	NULL	Gasolina	CHEVROLET	ESTADO	2005	0.8	5	1/4 A 3	1 A 10
	5	CARCHI	NULL	Gasolina	DAIHATSU	ESTADO	2003 Y ANTERIORES	0.8	5	1/4 A 3	1 A 10
	6	ESMERALDAS	NULL	Gasolina	CHEVROLET	ESTADO	2008	1	3	1/4 A 3	1 A 10
	7	ESMERALDAS	NULL	Gasolina	CHEVROLET	ESTADO	2008	1	3	1/4 A 3	1 A 10
	8	CHIMBORAZO	NULL	Diesel	HYUNDAI	ESTADO	2006	1	2	1/4 A 3	1 A 10
	9	IMBABURA	NULL	Gasolina	CHEVROLET	ESTADO	2003 Y ANTERIORES	0.8	5	1/4 A 3	1 A 10
	10	SUCUMBÍOS	NULL	Gasolina	MITSUBISHI	ESTADO	2003 Y ANTERIORES	1	3	1/4 A 3	1 A 10
	11	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2003 Y ANTERIORES	0.8	5	1/4 A 3	1 A 10
	12	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2003 Y ANTERIORES	1	3	1/4 A 3	1 A 10
	13	LOJA	NULL	Gasolina	CHEVROLET	ESTADO	2003 Y ANTERIORES	0.8	5	1/4 A 3	1 A 10
	14	NAPO	NULL	Gasolina	TOYOTA	MUNNICI...	2007	0.8	5	1/4 A 3	1 A 10
	15	EL ORO	NULL	Gasolina	CHEVROLET	ESTADO	2005	0.8	5	1/4 A 3	1 A 10
	16	PICHINCHA	NULL	Gasolina	FORD	ESTADO	2003 Y ANTERIORES	0.8	5	1/4 A 3	1 A 10
	17	SUCUMBÍOS	NULL	Gasolina	CHEVROLET	ESTADO	2003 Y ANTERIORES	1	3	1/4 A 3	1 A 10
	18	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2003 Y ANTERIORES	1	3	1/4 A 3	1 A 10
	19	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2003 Y ANTERIORES	1	3	1/4 A 3	1 A 10
	20	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2003 Y ANTERIORES	0.8	5	1/4 A 3	1 A 10
	21	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2003 Y ANTERIORES	0.8	5	1/4 A 3	1 A 10
	22	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2003 Y ANTERIORES	0.8	5	1/4 A 3	1 A 10
	23	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2003 Y ANTERIORES	0.8	5	1/4 A 3	1 A 10
	24	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2003 Y ANTERIORES	1	3	1/4 A 3	1 A 10
	25	GUAYAS	NULL	Gasolina	CHEVROLET	ESTADO	2004	0.8	5	1/4 A 3	1 A 10
	26	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2005	0.8	5	1/4 A 3	1 A 10
	27	SANTO DOMI...	NULL	Gasolina	CHEVROLET	ESTADO	2005	0.8	5	1/4 A 3	1 A 10
	28	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2005	1	3	1/4 A 3	1 A 10
	29	PICHINCHA	NULL	Gasolina	BMW	ESTADO	2003 Y ANTERIORES	0.75	5	1/4 A 3	1 A 10
	30	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2007	0.8	5	1/4 A 3	1 A 10
	31	MANABÍ	NULL	Gasolina	CHEVROLET	ESTADO	2007	0.8	5	1/4 A 3	1 A 10
	32	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2008	0.8	5	1/4 A 3	1 A 10
	33	PICHINCHA	NULL	Gasolina	CHEVROLET	ESTADO	2008	1	3	1/4 A 3	1 A 10

Ilustración 30 Datos importados

Se realizó un análisis de los datos de la matriculación vehicular para poder realizar un diagrama de entidad relación y obtuvimos el modelo que se encuentra en la ilustración 20:

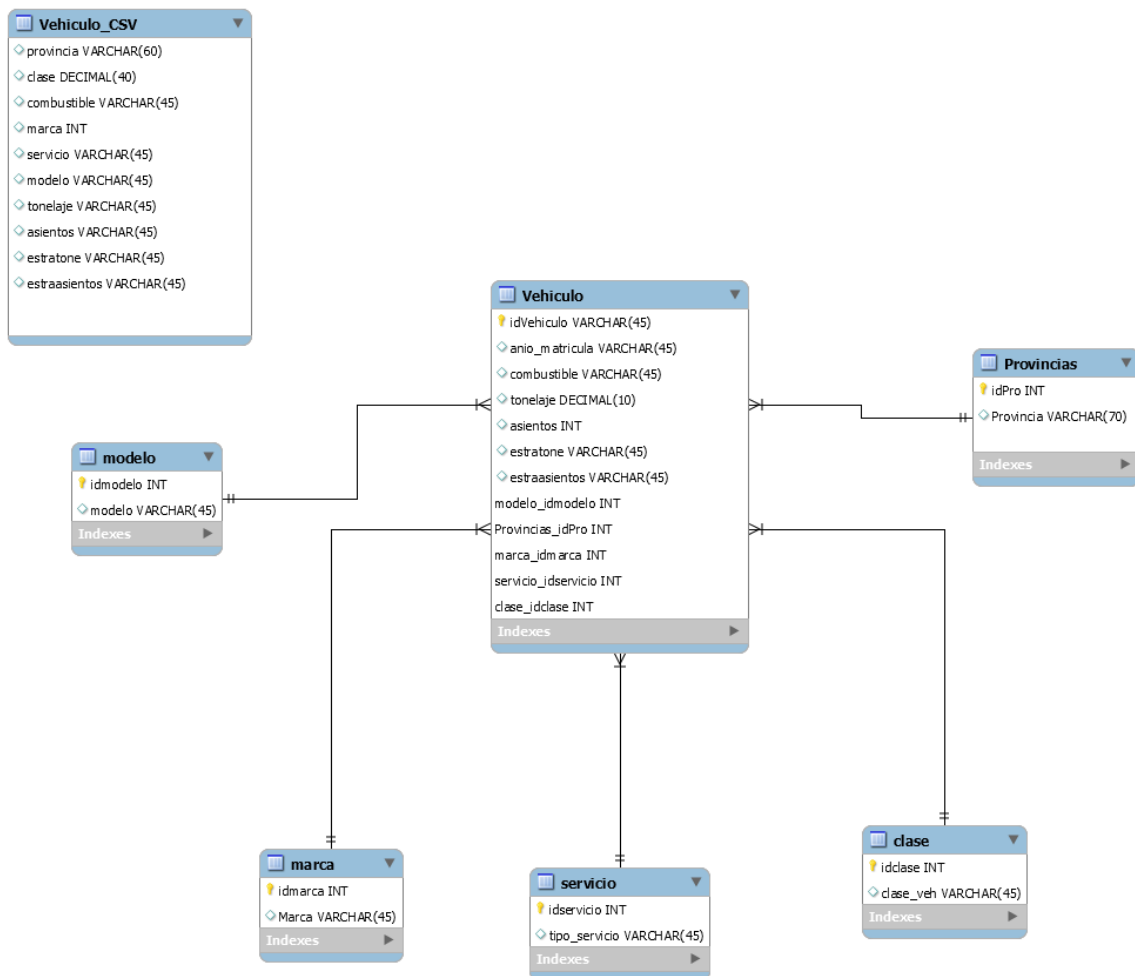


Ilustración 31 Modelo entidad relación

8.2 Proceso consulta uno

Cuando ya tenemos los datos importados en nuestra base de datos de MySQL, lo siguiente que vamos a realizar es la primera consulta.

1. ¿Cuál es la distribución (el numero) de los vehículos clasificados según el tipo de combustible que usan?

La sentencia que se muestra en la Ilustración 21 nos daría la resolución de la consulta,

```
SELECT COMBUSTIBLE, count(COMBUSTIBLE)as Total
FROM vehiculos_matriculados_2008
GROUP BY COMBUSTIBLE
ORDER BY COMBUSTIBLE ;
```

Ilustración 32 Sentencias para realizar la consulta uno en MySQL

La descripción del desarrollo de esta consulta se la puede encontrar en el Anexo 2

Los Resultados obtenidos al ejecutar esta sentencia en MySQL se muestran en la Ilustración 22.

	COMBUSTIBLE	Total
►	Diesel	101329
	Eléctrico	12
	Gas liquado de petróleo	290
	Gasolina	817277

Ilustración 33 Resultado de la consulta uno

8.3 Proceso consulta dos

2. ¿Cuál es el número de motocicletas matriculadas en el 2008 por provincia?

La sentencia que se muestra en la Ilustración 23 nos daría la resolución de la consulta,

```
SELECT SERVICIO, count(SERVICIO)as Total
FROM vehiculos_matriculados_2008
WHERE CLASE = "Motocicleta"
GROUP BY SERVICIO
ORDER BY SERVICIO;
```

Ilustración 34 Sentencias para realizar la consulta dos en MySQL

La descripción del desarrollo de esta consulta se la puede encontrar en el Anexo 3

Los Resultados obtenidos al ejecutar esta sentencia en MySQL se muestra en la ilustración 24:

	SERVICIO	Total
►	ESTADO	149
	MUNICIPIO	29
	PARTICULAR	85739
	SECTORES SECCIONALES	1

Ilustración 35 Resultado de la consulta dos

8.4 Proceso consulta tres

3. ¿Cuál es la distribución por provincia (el numero) de los tipos de servicios para vehículos que pueden considerarse de trabajo pesado (Camión, tanquero, tráiler y volqueta)

La sentencia que se muestra en la Ilustración 25 nos daría la resolución de la consulta,

```
SELECT  SERVICIO, CLASE, PROVINCIA,
        count(SERVICIO)as Total
FROM vehiculos_matriculados_2008
WHERE CLASE Like("Cami%") OR CLASE = "Tanquero"
      OR CLASE = "Trailer" OR CLASE = "Volqueta"
GROUP BY SERVICIO
ORDER BY SERVICIO;
```

Ilustración 36 Sentencias para realizar la consulta tres en MySQL

La descripción del desarrollo de esta consulta se la puede encontrar en el Anexo 4

Los Resultados obtenidos al ejecutar esta sentencia en MySQL son los siguientes:

	SERVICIO	CLASE	PROVINCIA	Total
►	ALQUILER	Camión	AZUAY	14011
	ESTADO	Camioneta	AZUAY	4048
	MUNICIPIO	Camioneta	ZAMORA CHINCHIPE	1810
	OTROS	Camioneta	AZUAY	78
	PARTICULAR	Camioneta	AZUAY	285067
	SECTORES SECCIONALES	Camioneta	IMBABURA	29

Ilustración 37 Resultado de la consulta tres

9. Conclusiones

Con la realización de este proyecto se ha logrado cumplir con los objetivos que se establecieron al comienzo de este, obteniendo resultados satisfactorios que los validan, entre los cuales se encuentran:

- Lograr identificar e implementar cada una de las herramientas necesarias para la resolución de las consultas además de dar una breve descripción de estas.
- Conseguir obtener los datos requeridos al utilizar tanto el lenguaje de programación Scala como MySQL, a la vez que se redactó el proceso con el cual se obtuvo cada resultado.
- Se plasmó mediante gráficos generados por vega-lite los resultados que se obtuvieron en cada una de las consultas para obtener una mejor visualización de los datos.

10. Bibliografía

- Íñigo, J. A. (2020, 4 mayo). *¿Qué es la Programación Reactiva? Una introducción*. Profile Software Services. <https://profile.es/blog/que-es-la-programacion-reactiva-una-introduccion/>
- Martín, Á. (2018, 23 julio). *¿Qué es la programación reactiva?* BI Geek Blog. <https://blog.bi-geek.com/que-es-la-programacion-reactiva/>
- Phares D. (2018, 23 junio). Bases Programación Funcional. Diego Phares. Recuperado de: <https://medium.com/@pharesdiego/bases-de-la-programaci%C3%B3n-funcional-f25cea12040a>
- Digital Guide IONOS. (2020, 11 febrero). Programación funcional: ideal para algoritmos. OINOS. Recuperado de: <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/programacion-funcional/>
- Silberschatz, A., Korth, H. F., Sudarshan, S., Pérez, F. S., Santiago, A. I., & Sánchez, A. V. (2002). Fundamentos de bases de datos.
- S. A (2020, 2 abril) Un vistazo a la historia de Scala. Reclu IT. Recuperado de: <https://recluit.com/un-vistazo-a-la-historia-de-scala/>
- Scala (S.F) IINTRODUCCIÓN. Scala. Recuperado de: <https://docs.scala-lang.org/es/tour/tour-of-scala.html>
- Gracia L. (2013, 11 abril) ¿Qué es sbt?. Un poco de Java y +. Recuperado de: <https://unpocodejava.com/2013/04/11/que-es-sbt/#:~:text=sbt%20es%20una%20herramienta%20de,definir%20la%20construcci%C3%B3n%20del%20proyecto.>
- Rinaudo N. (2015, 22 julio) Getting started. Kantan.csv. Recuperado de: https://nrinaudo.github.io/kantan.csv/rows_as_case_classes.html

- B., G. (2020, 3 diciembre). ¿Qué es MySQL? Explicación detallada para principiantes. Tutoriales Hostinger. <https://www.hostinger.es/tutoriales/que-es-mysql/>
- Pulido, M. (2019, 23 septiembre). *Todo lo que deberías saber sobre programación reactiva*. SlashMobility | Soluciones mobile. <https://slashmobility.com/blog/2019/09/programacion-reactiva/>
- Robledano, Á. (2020, 3 julio). *Qué es MySQL: Características y ventajas*. OpenWebinars.net. <https://openwebinars.net/blog/que-es-mysql/>
- P. (2019, 11 febrero). *PatMartin/Dex*. GitHub. <https://github.com/PatMartin/Dex>
- Satyanarayan, A., Moritz, D., Wongsuphasawat, K., & Heer, J. (2016). Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1), 341-350.
- *Slick*. (2019). Slick. <http://scala-slick.org/>

11. Anexos

Anexo 1

Bitácora

Fecha	Participantes
21/12/2020	Miguel Angel Caraguay Correa
	Adriana Sofía Jaramillo Ochoa
	Mateo Sebastián Martínez Velásquez
04/01/2021	Miguel Angel Caraguay Correa
	Adriana Sofía Jaramillo Ochoa
	Mateo Sebastián Martínez Velásquez
07/01/2021	Miguel Angel Caraguay Correa
	Adriana Sofía Jaramillo Ochoa
	Mateo Sebastián Martínez Velásquez
14/01/2021	Miguel Angel Caraguay Correa
	Adriana Sofía Jaramillo Ochoa
	Mateo Sebastián Martínez Velásquez
18/01/2021	Miguel Angel Caraguay Correa
	Adriana Sofía Jaramillo Ochoa
	Mateo Sebastián Martínez Velásquez
21/01/2021	Miguel Angel Caraguay Correa
	Adriana Sofía Jaramillo Ochoa
	Mateo Sebastián Martínez Velásquez
24/01/2021	Miguel Angel Caraguay Correa
	Adriana Sofía Jaramillo Ochoa

	Mateo Sebastián Martínez Velásquez
28/01/2021	Miguel Angel Caraguay Correa
	Adriana Sofía Jaramillo Ochoa
	Mateo Sebastián Martínez Velásquez
02/02/2021	Miguel Angel Caraguay Correa
	Adriana Sofía Jaramillo Ochoa
	Mateo Sebastián Martínez Velásquez
08/02/2021	Miguel Angel Caraguay Correa
	Adriana Sofía Jaramillo Ochoa
	Mateo Sebastián Martínez Velásquez

Anexo 2

La sentencia se la puede leer de la siguiente manera:

- En SELECT:
 - Muestra los datos de la columna combustible.
 - Contar los datos de cada valor que tenga combustible (Gasolina ,1000) y nombra a la columna como “Total”.
- En FROM
 - Buscar estos datos de la tabla “vehículos_matriculados_2008”.
- En GROUP BY
 - Agrupar los datos que se repiten de la columna de combustible.
- En ORDER BY
 - Ordenar los datos de la columna combustible en orden alfabético.

Anexo 3

Esta sentencia se la puede leer de la siguiente manera:

- En SELECT:
 - Muestra los datos de la columna servicio.
 - Contar los datos de cada valor que tenga servicio (Publico,1000) y nombra a la columna como “Total”.
- En FROM
 - Buscar estos datos de la tabla “vehículos_matriculados_2008”.
- En WHERE
 - Filtra los que cumplan con la siguiente condición: Que el dato que se encuentra en la columna CLASE sea igual a “Motocicleta”
- En GROUP BY
 - Agrupar los datos que se repiten de la columna de servicio.
- En ORDER BY
 - Ordenar los datos de la columna servicio en orden alfabético.

Anexo 4

La sentencia se la puede leer de la siguiente manera:

- En SELECT:
 - Muestra los datos de la columna servicio.
 - Muéstrame los datos de la columna provincia
 - Muéstrame los datos de la columna Provincia
 - Contar los datos de cada valor que tenga la columna servicio (Publico, Tanquero, Loja ,1000) y nombra a la columna como “Total”.
- En FROM
 - Buscar estos datos de la tabla “vehículos_matriculados_2008”.
- En WHERE
 - Filtra los que cumplan con la siguiente condición: Que el dato que se encuentra en la columna CLASE sea igual a una palabra que comience con “Cami...”, o CLASE sea igual a “Tanquero, o CLASE sea igual a “Trailer” o Clase sea igual a “Volqueta”.
- En GROUP BY
 - Agrupar los datos que se repiten de la columna de servicio.
- En ORDER BY
 - Ordenar los datos de la columna servicio en orden alfabético.