



Universidad de Ingeniería y Tecnología

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

SISTEMA DE RESERVA DE ESTACIONAMIENTOS

Base de datos I

Integrantes:

Luis Fernando Méndez Lázaro (202110216)
Nicolás Diego Castañeda Quichís (201910104)
Juaquin Estefano Remon Flores (202110302)
Mariajulia Romina Romani Tafur (202110320)

Junio del 2022

Índice

1. Requisitos	3
1.1. Introducción	3
1.2. Descripción general del problema/organización/empresa	3
1.3. Necesidad/usos de la base de datos	3
1.4. ¿Cómo resuelve el problema hoy?	3
1.4.1. ¿Cómo se almacenan/procesa los datos hoy?	3
1.4.2. Flujo de datos	4
1.5. Descripción detallada del sistema	4
1.5.1. Objetos de información actuales	4
1.5.2. Características y funcionalidades esperadas	4
1.5.3. Tipos de usuarios existentes/necesarios	4
1.5.4. Tipos de consulta, actualizaciones	5
1.5.5. Tamaño estimado de la base de datos	5
1.6. Objetivos del Proyecto	6
1.7. Referencias del Proyecto	6
1.8. Eventualidades	7
1.8.1. Problemas que pudieran encontrarse en el proyecto	7
1.8.2. Límites y alcances del proyecto	7
2. Modelo Entidad-Relación	8
2.1. Reglas semánticas	8
2.2. Modelo Entidad-Relación	9
2.3. Especificaciones y consideraciones sobre el modelo	9
3. Modelo Relacional	11
3.1. Modelo Relacional	11
3.2. Especificaciones de transformación	12
3.2.1. Entidades	12
3.2.2. Entidades débiles	12
3.2.3. Entidades superclase/subclase	13
3.2.4. Relaciones Binarias	13
3.3. Diccionario de datos	14
4. Implementación de la base de datos	19
4.1. Creación de la Base de Datos	19
4.2. Carga de datos	21
4.3. Simulación de datos	21
4.4. Descripción de Hardware y Software utilizado en la implementación	22

5. Optimización y Experimentación	23
5.1. Consultas SQL para la experimentación	23
5.1.1. Descripción de las consultas	23
5.1.2. Implementación de las consultas SQL	23
5.2. Metodología del experimento	25
5.3. Optimización de consultas	26
5.3.1. Índices para consulta 1	26
5.3.2. Índices para consulta 2	26
5.3.3. Índices para consulta 3	26
5.3.4. Identificación de índices implementados por Postgresql	27
5.4. Medición de tiempos	27
5.4.1. Ejecución de consulta 1	27
5.4.2. Ejecución de consulta 2	37
5.4.3. Ejecución de consulta 3	43
5.5. Medición de tiempos	52
5.5.1. Sin Índices	52
5.5.2. Con Índices	53
5.6. Resultados y análisis	54
5.6.1. Gráfica Consulta 1	55
5.6.2. Gráfica Consulta 2	56
5.6.3. Gráfica Consulta 3	57
6. Stored Procedures	57
7. Conclusiones	59
8. Anexos	60
8.1. Dumps de tablas, archivos CSV externos y vídeos de ejecución de consultas	60
8.2. Diagrama del modelo E-R	60
8.3. Diagrama del modelo relacional	61

1. Requisitos

1.1. Introducción

Presentamos esta base de datos como propuesta para la creación de un sistema de reserva de estacionamiento en línea dirigida hacia empresas dedicadas netamente al servicio de estacionamientos existentes en el Perú, tales como Fundo Don Felman o centros comerciales como lo es Real Plaza, la cuál será disponible para toda la comunidad.

1.2. Descripción general del problema/organización/empresa

En el Perú es notable la escasez de sistemas de estacionamiento en línea, los cuales optimizarían de forma efectiva el servicio al cliente, dado que reduce el tiempo de espera y búsqueda de un sitio dentro de un estacionamiento público, en su mayoría en días festivos o feriados. Para solucionar este problema se dará uso a la base de datos, la cual será visible para el usuario a través de una interfaz y podrá ser editada, pero de forma restringida.

1.3. Necesidad/usos de la base de datos

Se requiere el uso de una base de datos para poder administrar de manera eficiente los datos de cada usuario al reservar un estacionamiento, así como manejar eficazmente los espacios en la reserva.

1.4. ¿Cómo resuelve el problema hoy?

1.4.1. ¿Cómo se almacenan/procesa los datos hoy?

El procesamiento y almacenamiento de la data se divide en tres aspectos:

1. Captura de Datos

La información es obtenida mediante el envío de data que realiza el programa de reserva en línea, el cuál estará disponible para toda la población de interés. Este programa va a insertar todos los datos que han sido leídos y validados con anterioridad a la base de datos. Por consiguiente, no es posible enviar data incorrecta o innecesaria.

2. Procesamiento de datos:

El procesamiento de los datos es realizado por el usuario, debido a que puede realizar una reserva tanto como editarla o cancelarla, manipulando la data obtenida.

3. Análisis y resultados:

La información puede ser visualizada, editada y eliminada por cada usuario de forma restringida.

1.4.2. Flujo de datos

El flujo de datos se maneja mediante una aplicación web disponible para los usuarios. A través de este se podrá enviar, editar, eliminar y realizar consultas sobre la información contenida en la base de datos.

1.5. Descripción detallada del sistema

1.5.1. Objetos de información actuales

La fuente de información proviene de los registros vehiculares de la SUNARP (Superintendencia Nacional de los Registros Públicos) con los cuales se realizará la generación de usuarios y vehículos en el sistema. Para efectos de nuestro proyecto, la información será almacenada en la base de datos y se realizarán consultas que involucren a 1000 (mil) datos, 10 000 (diez mil) datos, 100 000 (cien mil) datos y 1 000 000 (un millón) de datos.

1.5.2. Características y funcionalidades esperadas

La empresa busca implementar un sistema donde se pueda reservar un espacio de estacionamiento desde tu celular o aplicativo web. La aplicación te solicitará los datos del vehículo y del conductor para realizar una reserva. Asimismo, para asegurar una reserva responsable, se realizará un pago mediante la aplicación al momento de reservar, equivalente a un porcentaje de la tarifa total, la cual depende de las horas de reserva. Una vez que tengas un espacio reservado, solo podrás cancelar o modificar tu reserva hasta antes que esta empiece.

Por otro lado, los usuarios podrán ser VIP y tener la posibilidad de acceder a algunos beneficios como reservar sin hacer pagos previos, servicio de lavado de vehículo y entre otros. Adicionalmente, los usuarios Empresariales dispondrán de todos los beneficios que se ofrecen.

1.5.3. Tipos de usuarios existentes/necesarios

- **Usuario:** Persona registrada en el sistema y con acceso a los estacionamientos de la compañía. Dicha entidad es responsable de proporcionar flujo de datos a otras entidades.
- **Vehículo:** Entidad perteneciente a un usuario. La compañía debe asegurar su existencia para proceder con diferentes servicios que se ofrecen como el estacionamiento.
- **Estacionamiento:** Entidad que permite almacenar los aparcamientos donde se realizarán las reservas en el sistema.

1.5.4. Tipos de consulta, actualizaciones

Entre las principales consultas se tiene:

- Consulta 1: ¿Cuál es el tiempo promedio que dura una reserva?
- Consulta 2: ¿Cuál es la cantidad de vehículos estacionados por cada estacionamiento?
- Consulta 3: ¿En qué mes se registra la mayor cantidad de reservas?
- Consulta 4: ¿Cuál es la cantidad de vehículos registrados por cada usuario?
- Consulta 5: ¿Cuánto es el ingreso generado por las cuentas de tipo VIP?
- Consulta 6: ¿Cuántas reservas mensuales realizan los usuarios de tipo VIP?
- Consulta 7: ¿Cuál es costo promedio de reservas que duran menos de 5 horas?

Además, está permitida la actualización de algunos recursos:

- Inhabilitación de cuenta propia.
- Atributos principales del vehículo.
- Cambiar el estado de reserva.

1.5.5. Tamaño estimado de la base de datos

- Las tablas de la base de datos estarán en constante crecimiento, debido a que se añaden nuevos registros a partir de la creación de nuevos usuarios, vehículos y reservas.
- Referente a tablas relacionadas a los estacionamientos, se tiene la capacidad de crear nuevos establecimientos y/o aumentar la cantidad de aparcamientos disponibles dentro de un estacionamiento existente.
- Inicialmente la tabla Usuarios poseerá mayor actividad, puesto que a partir de sus registros se realizarán las funcionalidades principales de la base de datos. Posteriormente, la tabla Reserva contará con un mayor flujo de datos ya que el servicio de reserva centra sus actividades en base a este.

A partir de lo mencionado, se va a estimar el tamaño en bytes de la base de datos, tomando en consideración cada una de sus tablas y el tipo de sus respectivos atributos:

Nombre de la tabla	Longitud de atributos (Bytes)	Longitud del registro (Bytes)
Usuario	9+51+51+10+257+1	379
Personal	9	9
VIP	9+8	17
Empresarial	9+21	30
Vehículo	9+11+21+51+21+4	117
Reserva	4+4+9+11+8+8+4	48
Aparcamiento	4+4+4	12
Estacionamiento	4+51+101+8+8	172
Beneficio	21+51	72
posee_b_vip	21+9	30
posee_b_emp	21+9	30

Cuadro 1: Estimación de la base de datos

Nombre de la tabla	Longitud de atributos (Bytes)	Datos Estimados	Tamaño total (bytes)
Usuario	379	1000000	379000000
Personal	9	900000	8100000
VIP	17	50000	850000
Empresarial	30	50000	1500000
Vehículo	117	500000	58500000
Reserva	48	200000	9600000
Aparcamiento	12	10000	120000
Estacionamiento	172	100	17200
Beneficio	72	10	720
posee_b_vip	30	50000	1500000
posee_b_emp	30	50000	1500000
Total			460687920

Cuadro 2: Estimación del tamaño total de la base de datos

1.6. Objetivos del Proyecto

- Implementar el uso de base de datos para optimizar el proceso de búsqueda de un espacio de estacionamiento para un vehículo.
- Optimizar la experiencia del usuario al evitar largos tiempo de espera para encontrar un estacionamiento.
- Habilitar la posibilidad de que los aparcamientos dentro de un estacionamiento sean gestionados automáticamente.

1.7. Referencias del Proyecto

Para el actual proyecto, tomamos en cuenta distintos aspectos de diferentes compañías que actualmente funcionan de manera amigable con el usuario como la gestión de reserva de un espacio en un autocine, el sistema de gestión de seguridad y control de estacionamientos de Los Portales y el sistema de pago con tarjetas simple y rápido de Rappi. Junto a dichas características, añadimos funcionalidades útiles y necesarias para el buen desempeño del proyecto.

1.8. Eventualidades

1.8.1. Problemas que pudieran encontrarse en el proyecto

Un problema que surge de las interacciones con los usuarios es la posible falsificación de datos que nos proporcionan, como la placa de su auto, sus nombres, apellidos o celular. Además, al utilizar datos como el número celular y el correo electrónico para la creación de usuario, se puede dar por perdida una cuenta debido a que el usuario puede olvidar su contraseña, perder su celular, etc. Para ello se tiene que añadir autenticaciones que permitan recuperar la información de un usuario.

1.8.2. Límites y alcances del proyecto

- El límite más importante es la ubicación geográfica, al ser una aplicación pensada para clientes y vehículos peruanos, estas restricciones e información que se tomará se tienen que dar en territorio nacional.
- La incapacidad de abarcar todos los posibles estacionamientos, con riesgos de no poder ubicar por lo menos un establecimiento en una zona residencial. Por lo tanto, no podría ser útil para cualquier tipo de usuario por su ubicación.
- El proyecto incluye modelar una base de datos confiable y segura para la modernización de los estacionamientos automovilísticos. Además, propone un alcance nacional como forma de implementación comercial.

2. Modelo Entidad-Relación

2.1. Reglas semánticas

- Cada Usuario posee un DNI único para identificarlo. Además, posee un celular, nombres, apellidos, contraseña y un estado de registro. Pertenece necesariamente al tipo Personal, VIP o Empresarial.
- El estado de registro de cada usuario solo puede ser True para activo o False para no activo.
- El celular de cada usuario es único, inicia con 9 y debe tener 9 dígitos.
- La contraseña debe contener por lo menos un dígito, una mayúscula, una minúscula y 8 caracteres como mínimo.
- Cada Usuario del tipo VIP solo tiene acceso a tres beneficios.
- Cada Usuario del tipo Empresarial tiene acceso a todos los beneficios.
- El beneficio es identificado por un título y una descripción. El título es único.
- Un usuario puede tener cero o más vehículos.
- Un vehículo posee una placa única. El vehículo tiene asociado a él una marca, modelo, color y estado de registro.
- Un vehículo pertenece necesariamente a un usuario.
- El estado de registro de cada vehículo solo puede ser Disponible (DIS) o No disponible (NOD).
- Un vehículo puede registrar 0 o más reservas y la reserva es de solo un vehículo.
- Cada reserva debe ser hecha en un aparcamiento de un estacionamiento y a su vez debe ser registrada por un vehículo perteneciente a un usuario.
- Una reserva debe realizarse por lo menos con 30 minutos de anticipación.
- Una reserva debe durar por lo menos 1 hora.
- El estado de registro de cada reserva solo puede ser Cancelado (CAN), Pendiente (PEN) o En proceso (ENP).
- Un aparcamiento tiene 0 o más reservas y la reserva es de exactamente un aparcamiento.
- Un aparcamiento perteneciente a un estacionamiento tiene un código único identificador.

- El estado de registro de cada aparcamiento solo puede ser Reservado (RES), Ocupado (OCU) o Disponible (DIS).
- Un estacionamiento posee un código único identificador. Además, posee un nombre y sus coordenadas geográficas latitud y longitud, las cuales pueden ser cero si el estacionamiento no posee acceso a esa información.
- Un estacionamiento tiene al menos un aparcamiento y un aparcamiento pertenece a un estacionamiento.

2.2. Modelo Entidad-Relación

Referirse al Anexo 8.2 para visualizar el modelo Entidad-Relación

2.3. Especificaciones y consideraciones sobre el modelo

En esta sección se describirán todas las entidades de la siguiente manera:

1. Entidad Usuario

Especificaciones: Almacena la información referente a un usuario registrado en el sistema. Su llave primaria es el DNI.

Consideraciones: La llave primaria es DNI debido a que este es un identificador único para los ciudadanos del Perú. Usuario es una superclase y hereda atributos a las subclases Personal, VIP y Empresarial. Un usuario puede tener una cuenta personal y empresarial simultáneamente por lo que existe solapamiento. Hay cobertura debido a que cada usuario debe pertenecer necesariamente a una subclase.

2. Entidad Personal

Especificaciones: Almacena información pertinente al Usuario tipo Personal. Su llave primaria la hereda de Usuario.

Consideraciones: Es una subclase de Usuario, por lo tanto, hereda su llave primaria y atributos de este explicados en su sección.

3. Entidad VIP

Especificaciones: Almacena información pertinente al Usuario tipo VIP. Su llave primaria la hereda de Usuario.

Consideraciones: Es una subclase de Usuario, por lo tanto, hereda su llave primaria y atributos de este explicados en su sección.

4. Entidad Empresarial

Especificaciones: Almacena información pertinente al Usuario tipo Empresarial. Su llave primaria la hereda de Usuario.

Consideraciones: Es una subclase de Usuario, por lo tanto, hereda su llave primaria y atributos de este explicados en su sección.

5. Entidad Beneficio

Especificaciones: Almacena información sobre un beneficio el cual puede relacionarse a un usuario VIP o Empresarial. Su llave primaria es Título.

Consideraciones: Esta entidad no depende de la existencia de VIP y Empresarial, pues los beneficios pueden existir sin la necesidad de que un usuario se relacione a dicho beneficio.

6. Entidad Vehículo

Especificaciones: Entidad débil de Usuario, debido a que todo vehículo tiene que ser registrado y relacionado a dicho usuario. Sus llaves primarias son placa y DNI, la llave foránea es DNI de la entidad Usuario. El atributo placa es único debido a que este es un identificador para cada vehículo.

Consideraciones: La entidad débil vehículo es necesaria para poder registrar una reserva. Cada vehículo depende de la existencia de un usuario por lo que es una entidad débil.

7. Entidad Reserva

Especificaciones: Se trata de una entidad que requiere de un vehículo y un lugar de aparcamiento de un estacionamiento para registrarla en la base de datos. Sus llaves son sus fechas de inicio, fin, el código del aparcamiento y del estacionamiento.

Consideraciones: Al igual que con la entidad vehículo, cuenta con una participación total con aparcamiento.

8. Entidad Aparcamiento

Especificaciones: Esta entidad es débil pues depende de la entidad estacionamiento para su registro. En tal sentido, no existe un aparcamiento que no pertenezca a alguna playa de estacionamiento de la compañía. Su llave parcial es el código.

Consideraciones: Esta entidad es necesaria para manejar correctamente los datos del lugar y estado de todos los aparcamientos existentes en cada una de las playas de estacionamiento.

9. Entidad Estacionamiento

Especificaciones: Se almacenan principalmente datos relacionados a la geolocalización del estacionamiento. Se elige como llave primaria el código. Además, la dirección debe ser única puesto que no pueden existir dos o más estacionamientos en una misma localización.

Consideraciones: Cuenta con participación total con aparcamiento, esto es, que cada playa de estacionamiento de la compañía posee al menos un aparcamiento. Asimismo, la latitud y longitud de dicho establecimiento no son llaves pues en ciertos registros no se cuenta con información al respecto y su valor predeterminado es 0 para ambos casos.

3. Modelo Relacional

3.1. Modelo Relacional

1. Estacionamiento(codigo:INTEGER, nombre:VARCHAR(50), direccion:VARCHAR(100), latitud:DOUBLE PRECISION, longitud:DOUBLE PRECISION)
2. Usuario(DNI:VARCHAR(8), nombres:VARCHAR(50), apellidos:VARCHAR(50), celular:VARCHAR(9), contraseña:VARCHAR(256), estadoRegistro:BOOLEAN)
3. Vehiculo(Usuario.DNI:VARCHAR(8), placa:VARCHAR(10), marca:VARCHAR(20), modelo:VARCHAR(50), color:VARCHAR(20), estadoRegistro:VARCHAR(3))
4. Aparcamiento(Estacionamiento.codigo:INTEGER, codigo:INTEGER, estadoRegistro:VARCHAR(3))
5. Reserva(inicioReserva:TIMESTAMP, finReserva:TIMESTAMP, Estacionamiento.codigo:INTEGER, Aparcamiento.codigo:INTEGER, Usuario.DNI:VARCHAR(8), Vehiculo.placa:VARCHAR(10), estadoRegistro:VARCHAR(3))
6. Personal(Usuario.DNI:VARCHAR(8))
7. VIP(Usuario.DNI:VARCHAR(8), costo:REAL)
8. Empresarial(Usuario.DNI:VARCHAR(8), RUC:VARCHAR(20))
9. Beneficio(titulo:VARCHAR(20), descripción:TEXT)
10. posee_b_vip(Beneficio.titulo:VARCHAR(20), VIP.DNI:VARCHAR(8))
11. posee_b_emp(Beneficio.titulo:VARCHAR(20), Empresarial.DNI:VARCHAR(8))

3.2. Especificaciones de transformación

3.2.1. Entidades

Tabla	Estacionamiento
Clave Primaria	codigo
Clave Foránea	

Cuadro 3: Estacionamiento

Tabla	Beneficio
Clave Primaria	titulo
Clave Foránea	

Cuadro 4: Beneficio

3.2.2. Entidades débiles

Tabla	Reserva
Entidad Débil	Reserva
Entidad Fuerte	Aparcamiento
Clave Primaria	inicioReserva,finReserva,Estacionamiento.codigo, Aparcamiento.codigo
Clave Foránea	Usuario.DNI, Vehiculo.placa

Cuadro 5: Reserva

Tabla	Vehiculo
Entidad Débil	Vehiculo
Entidad Fuerte	Usuario
Clave Primaria	placa,U.DNI
Clave Foránea	Usuario.DNI

Cuadro 6: Vehiculo

Tabla	Aparcamiento
Entidad Débil	Aparcamiento
Entidad Fuerte	Estacionamiento
Clave Primaria	codigo , E.codigo
Clave Foránea	Estacionamiento.codigo

Cuadro 7: Aparcamiento

3.2.3. Entidades superclase/subclase

Tabla	Usuario
Clave Primaria	DNI
Clave Foránea	

Cuadro 8: Usuario

Tabla	Personal
Superclase	Usuario
Clave Primaria	U.DNI
Clave Foránea	Usuario.DNI

Cuadro 9: Personal

Tabla	VIP
Superclase	Usuario
Clave Primaria	U.DNI
Clave Foránea	Usuario.DNI

Cuadro 10: VIP

Tabla	Empresarial
Superclase	Usuario
Clave Primaria	U.DNI
Clave Foránea	Usuario.DNI

Cuadro 11: Empresarial

3.2.4. Relaciones Binarias

Tabla	Posee_b_vip
Entidades	VIP, beneficio
Nombre Relación	Posee_b_vip
Atributos Relación	
Clave Primaria	Beneficio.titulo, VIP.DNI
Clave Foránea VIP	VIP.DNI
Clave Foránea Beneficio	Beneficio.titulo

Cuadro 12: Posee_b_vip

Tabla	Posee_b_emp
Entidades	empresarial, beneficio
Nombre Relación	Posee_b_emp
Atributos Relación	
Clave Primaria	Beneficio.titulo, Empresarial.DNI
Clave Foránea Empresarial	Empresarial.DNI
Clave Foránea Beneficio	Beneficio.titulo

Cuadro 13: Posee_b.emp

3.3. Diccionario de datos

Estacionamiento

Nombre del campo	Tipo de dato	PK	FK	Descripción
codigo	INTEGER	✓		Código único que identifica al estacionamiento
nombre	VARCHAR(200)			Nombre del estacionamiento
latitud	DOUBLE PRECISION			Latitud en grados de la coordenada geográfica
longitud	DOUBLE PRECISION			Longitud en grados de la coordenada geográfica

Cuadro 14: Estacionamiento

Usuario

Nombre del campo	Tipo de dato	PK	FK	Descripción
DNI	VARCHAR(8)	✓		Identificador único de los ciudadanos del Perú
nombres	VARCHAR(50)			Nombres del usuario registrado
apellidos	VARCHAR(50)			Apellidos del usuario registrado
celular	VARCHAR(9)			Número de celular del usuario registrado
contrasena	VARCHAR(256)			Contraseña para inicio de sesión del usuario
estadoRegistro	BOOLEAN			Indica el estado en el que se encuentra el registro

Cuadro 15: Usuario

Vehículo

Nombre del campo	Tipo de dato	PK	FK	Descripción
Usuario.DNI	VARCHAR(8)	✓	✓	Identificador único del ciudadano dueño del vehículo
placa	VARCHAR(10)	✓		Identificador único del vehículo
marca	VARCHAR(50)			Marca del vehículo registrado
modelo	VARCHAR(50)			Modelo del vehículo registrado
color	VARCHAR(50)			Color específico del vehículo registrado
estadoRegistro	VARCHAR(3)			Indica el estado en el que se encuentra el registro

Cuadro 16: Vehículo

Aparcamiento

Nombre del campo	Tipo de dato	PK	FK	Descripción
Estacionamiento.codigo	INTEGER	✓	✓	Código único que identifica al estacionamiento
codigo	INTEGER	✓		Código que identifica al aparcamiento en un estacionamiento
estadoRegistro	VARCHAR(3)			Indica el estado en el que se encuentra el registro

Cuadro 17: Aparcamiento

Reserva

Nombre del campo	Tipo de dato	PK	FK	Descripción
inicioReserva	TIMESTAMP	✓		Momento en el que se inicia la reserva
finReserva	TIMESTAMP	✓		Momento en el que se finaliza la reserva
Estacionamiento.codigo	INTEGER	✓	✓	Código único que identifica al estacionamiento
Aparcamiento.codigo	INTEGER	✓	✓	Código que identifica al aparcamiento en un estacionamiento
Usuario.DNI	VARCHAR(8)		✓	Identificador único del ciudadano dueño del vehículo
Vehiculo.placa	VARCHAR(10)		✓	Identificador único del vehículo
estadoRegistro	VARCHAR(3)			Estado en el que se encuentra la reserva

Cuadro 18: Reserva

Personal

Nombre del campo	Tipo de dato	PK	FK	Descripción
usuario.DNI	VARCHAR(8)	✓	✓	DNI único del personal

Cuadro 19: Personal

VIP

Nombre del campo	Tipo de dato	PK	FK	Descripción
usuario.DNI	VARCHAR(8)	✓	✓	DNI único del usuario VIP
costo	REAL			Precio de la cuenta VIP que el usuario pagó

Cuadro 20: VIP

Empresarial

Nombre del campo	Tipo de dato	PK	FK	Descripción
usuario.DNI	VARCHAR(8)	✓	✓	DNI único del personal
RUC	VARCHAR(20)			RUC que se ingresó para acceder al empresarial

Cuadro 21: Empresarial

Beneficio

Nombre del campo	Tipo de dato	PK	FK	Descripción
titulo	VARCHAR(20)	✓		Nombre único de el beneficio
descripcion	TEXT			Descripción específica del beneficio

Cuadro 22: Beneficio

posee_b_vip

Nombre del campo	Tipo de dato	PK	FK	Descripción
Beneficio.titulo	VARCHAR(20)	✓	✓	Nombre único de el beneficio
VIP.DNI	VARCHAR(8)	✓	✓	DNI único del usuario VIP

Cuadro 23: posee_b_vip

posee_b.emp

Nombre del campo	Tipo de dato	PK	FK	Descripción
Beneficio.titulo	VARCHAR(20)	✓	✓	Nombre único de el beneficio
empresarial.DNI	VARCHAR(8)	✓	✓	DNI único del usuario con cuenta empresarial

Cuadro 24: posee_b.emp

4. Implementación de la base de datos

4.1. Creación de la Base de Datos

A continuación, en el presente apartado se muestra el código fuente SQL encargado de la creación de las tablas de nuestra base de datos. De acuerdo con nuestro modelo relacional normalizado se incluyen además las restricciones de integridad para su adecuado funcionamiento.

```
1 -- Estacionamiento
2 CREATE TABLE Estacionamiento (
3     codigo INTEGER,
4     nombre VARCHAR(200) NOT NULL,
5     latitud DOUBLE PRECISION NOT NULL,
6     longitud DOUBLE PRECISION NOT NULL,
7
8     PRIMARY KEY(codigo)
9 );
10
11 ALTER TABLE Estacionamiento ADD CONSTRAINT direccion_unique
12 UNIQUE (latitud, longitud);
13
14 ALTER TABLE Estacionamiento ADD CONSTRAINT latitud_check
15 CHECK (latitud >= -90 AND latitud <= 90);
16
17 ALTER TABLE Estacionamiento ADD CONSTRAINT longitud_check
18 CHECK (longitud >= -90 AND longitud <= 90);
19
20 -- Usuario
21 CREATE TABLE Usuario(
22     DNI VARCHAR(8),
23     nombres VARCHAR(50) NOT NULL,
24     apellidos VARCHAR(50) NOT NULL,
25     celular VARCHAR(9),
26     contrasena VARCHAR(256) NOT NULL,
27     estadoRegistro BOOLEAN NOT NULL,
28
29     PRIMARY KEY (DNI)
30 );
31
32 ALTER TABLE Usuario ADD CONSTRAINT dni_len_check CHECK (char_length
(DNI) = 8);
33
34 ALTER TABLE Usuario ADD CONSTRAINT celular_len_check CHECK (
char_length(celular) = 9);
35
36
37 -- Vehiculo
38 CREATE TABLE Vehiculo(
39     placa VARCHAR(10),
40     DNI VARCHAR(8),
41     marca VARCHAR(50),
42     modelo VARCHAR(50),
43     color VARCHAR(50),
44     estadoRegistro VARCHAR(3),
```

```

46     PRIMARY KEY (placa, dni),
47     FOREIGN KEY (DNI) REFERENCES Usuario(DNI)
48 );
49
50 ALTER TABLE Vehiculo ADD CONSTRAINT placa_check CHECK (char_length(
51     placa) = 6);
51 ALTER TABLE Vehiculo ADD CONSTRAINT Estado_registro_check CHECK (
52     estadoRegistro in ('REG', 'NOR'));
53
53 -- Aparcamiento
54 CREATE TABLE Aparcamiento(
55     codigo INTEGER,
56     codigo_e INTEGER,
57     estadoRegistro VARCHAR(3),
58
59     PRIMARY KEY(codigo, codigo_e),
60     FOREIGN KEY (codigo_e) REFERENCES Estacionamiento(codigo)
61 );
62
63 ALTER TABLE Aparcamiento ADD CONSTRAINT Estado_registro_check CHECK
64     (estadoRegistro in ('OCU', 'RES', 'DIS'));
65
65 -- Reserva
66 CREATE TABLE Reserva(
67     codigo_e INTEGER,
68     codigo_a INTEGER,
69     inicioReserva TIMESTAMP,
70     finReserva TIMESTAMP,
71     DNI VARCHAR(8),
72     placa VARCHAR(10),
73     estadoRegistro VARCHAR(3),
74
75     PRIMARY KEY (codigo_e, codigo_a, inicioReserva , finReserva),
76     FOREIGN KEY (placa, DNI) REFERENCES Vehiculo(placa, DNI),
77     FOREIGN KEY (codigo_a, codigo_e) REFERENCES Aparcamiento(codigo
78     , codigo_e)
79 );
80
80 ALTER TABLE Reserva ADD CONSTRAINT check_estado CHECK (
81     estadoRegistro in ('PEN', 'ENP', 'CAN'));
82
82 -- Personal, vip, empresario
83 CREATE TABLE Personal(
84     DNI VARCHAR(8),
85
86     PRIMARY KEY(DNI),
87     FOREIGN KEY (DNI) REFERENCES Usuario(DNI)
88 );
89
90 CREATE TABLE VIP(
91     DNI VARCHAR(8),
92     costo REAL,
93
94     PRIMARY KEY(DNI),
95     FOREIGN KEY (DNI) REFERENCES Usuario(DNI)
96 );
97

```

```

98  ALTER TABLE VIP ADD CONSTRAINT check_costo CHECK (costo > 0);
99
100 CREATE TABLE Empresarial(
101     DNI VARCHAR(8),
102     RUC VARCHAR(20),
103
104     PRIMARY KEY(DNI),
105     FOREIGN KEY (DNI) REFERENCES Usuario(DNI)
106 );
107
108 ALTER TABLE Empresarial ADD CONSTRAINT check_ruc CHECK (RUC ~ $$\d
109     {11}$$);
110
111 -- Beneficio
112 CREATE TABLE Beneficio(
113     titulo VARCHAR(20),
114     descripcion TEXT NOT NULL,
115
116     PRIMARY KEY(titulo)
117 );
118
119 CREATE TABLE Posee_b_vip(
120     titulo VARCHAR(20),
121     DNI VARCHAR(8),
122
123     PRIMARY KEY (titulo, DNI),
124     FOREIGN KEY (titulo) REFERENCES Beneficio(titulo),
125     FOREIGN KEY (DNI) REFERENCES VIP(DNI)
126 );
127
128 CREATE TABLE Posee_b_emp(
129     titulo VARCHAR(20),
130     DNI VARCHAR(8),
131
132     PRIMARY KEY (titulo, DNI),
133     FOREIGN KEY (titulo) REFERENCES Beneficio(titulo),
134     FOREIGN KEY (DNI) REFERENCES Empresarial(DNI)
135 );

```

4.2. Carga de datos

La carga de datos fue llevada a cabo por generadores de datos online y scripts, que se explicarán en el siguiente apartado, y mediante la técnica de web scraping en donde se utilizó data de Google Maps para la obtención de información referente a los estacionamientos. Los resultados fueron almacenados en archivos csv para luego ser cargados en la base de datos a través del comando COPY.

4.3. Simulación de datos

La simulación de la datos de las tablas se consiguieron a través de generadores de datos online como 'online data generator' y 'mockaroo'. La información

obtenido era complementada a través archivos R y python en los cuales se utilizaron las librerías disponibles en estas herramientas para conseguir registros acordes a las restricciones de cada tabla.

4.4. Descripción de Hardware y Software utilizado en la implementación

Sistema operativo	Windows 10 Home Single Language 19041.630 x64
Procesador	Intel(R) Core (TM) i9-10900 10-Core 2.8 GHz
RAM	16 GB DDR4 3200 MHz
PostgreSQL	14.4
pgAdmin	6.4
Python	3.9.7
R	4.2.1

5. Optimización y Experimentación

En la presente sección, se desarrollarán tres consultas con un grado de complejidad adecuado con el fin de realizar varias pruebas que permitan evidenciar la eficiencia de la base de datos. En el transcurso de la experimentación, se utilizarán dichas consultas sobre distintos volúmenes de datos (1k, 10k, 100k y 1M), habilitando y deshabilitando los índices, de modo que se pueda evidenciar el impacto de los índices en el proceso de optimización.

5.1. Consultas SQL para la experimentación

5.1.1. Descripción de las consultas

Consulta 1 Se desea conocer las matrículas automovilísticas de cuyos usuarios propietarios han hecho uso de los estacionamientos con mayor número de reservas iniciadas desde 2019.

Justificación: Por medio de los datos obtenidos de la consulta, la empresa puede comenzar un estudio de mercado sobre qué tipo de autos y marcas son los más frecuentes en estos estacionamientos. Esto último alienta a priorizar y brindar mayores servicios a estos para adaptarse a las necesidades de los clientes. Se priorizan los estacionamientos con mayor cantidad de reservas desde un año específico dado que estos son los que generan más ingresos a la empresa.

Consulta 2 ¿ Cuáles son los estacionamientos que han tenido usuarios que durante determinado periodo de tiempo han realizado reservas en más de un estacionamiento?

Justificación: La consulta se basa en el uso de un intervalo de tiempo arbitrario con el fin de conocer específicamente los estacionamientos que tienen clientes recurrentes en varios otros estacionamientos. Esto permite que la empresa pueda reconocer a clientes frecuentes y capturar las características de aquellos estacionamientos a los que estos clientes suelen ir durante ciertos periodos del año.

Consulta 3

¿Cuales son los DNIs de los vehículos que reservaron una cantidad de horas pertenecientes al 1 % con más horas?

Justificación: Esta consulta ayuda a mapear los usuarios que han reservado sus autos más horas en la aplicación. Con esta información la empresa puede beneficiar a ellos o considerar alguna característica destacada.

5.1.2. Implementación de las consultas SQL

Consulta 1

```

1  -- Creacion de una vista para la evitar la repeticion de codigo
2  -- Numero de reservas iniciadas desde 2019 por cada estacionamiento
3  CREATE OR REPLACE VIEW reservas_por_estacionamiento_desde_2019 AS
4  SELECT codigo_e, COUNT(codigo_e) AS reservas
5  FROM reserva
6  WHERE EXTRACT(YEAR FROM inicioReserva) >= 2019
7  GROUP BY codigo_e;
8
9  -- Consulta
10 SELECT placa
11 FROM vehiculo
12 WHERE dni IN (
13     -- Dnis de los usuarios que han registrado una reserva en
14     -- dichos estacionamientos
15     SELECT dni
16     FROM estacionamiento E
17     INNER JOIN reserva R
18     ON E.codigo = R.codigo_e
19     WHERE E.codigo IN (
20         -- Codigo de los estacionamientos con mayor numero de
21         -- reservas iniciadas desde 2019
22         SELECT codigo_e
23         FROM reservas_por_estacionamiento_desde_2019
24         WHERE reservas = (
25             SELECT MAX(reservas)
26             FROM reservas_por_estacionamiento_desde_2019
27         )
28     )
29 );

```

Consulta 2

```

1  SELECT codigo, nombre, latitud, longitud
2  FROM estacionamiento
3  INNER JOIN (
4      SELECT codigo_e
5      FROM reserva
6      INNER JOIN (
7          SELECT dni
8          FROM reserva
9          WHERE inicioreserva >= '2018-12-02' AND inicioreserva <= ,
10              2019-03-01,
11              OR finreserva >= '2018-12-02' AND finreserva <= '2019-03-01'
12          ,
13          GROUP BY dni
14          HAVING COUNT(codigo_e) > 1
15      ) dnis
16      ON reserva.dni = dnis.dni
17  ) est_cod
18  ON estacionamiento.codigo = est_cod.codigo_e;

```

Consulta 3

```
1 SELECT dni
2 FROM vehiculo
3   NATURAL JOIN (SELECT placa
4     FROM (
5       SELECT placa,
6         sum(finreserva - inicioreserva) AS total_reservado
7       FROM reserva
8       GROUP BY placa) b
9     WHERE total_reservado IN (SELECT total_tiempo
10      FROM (
11        SELECT total_tiempo, row_number()
12        OVER (ORDER BY total_tiempo DESC)
13      FROM (
14        SELECT sum(finreserva - inicioreserva)
15          AS total_tiempo
16        FROM reserva
17        GROUP BY placa
18        ORDER BY total_tiempo DESC) n) b
19     WHERE row_number < (
20       SELECT count(placa)
21         FROM reserva) / 100
22   ) c;
```

5.2. Metodología del experimento

Para el experimento se trabajará con distintos esquemas de datos sobre los cuales se van a analizar la eficiencia del uso de índices. Para las pruebas se ejecutarán los siguientes pasos:

1. Se creará una base de datos la cual contendrá ocho esquemas basados en el modelo relacional del Anexo 8.3, los cuales presentan 1000, 10 000, 100 000 y 1 000 000 tuplas
2. Se utilizó scripts en R y Python para la generación de datos acordes a los constraints de las tablas.
3. Para que cada consulta se realice en igualdad de condiciones se utilizará VACUUM FULL con el fin de limpiar el caché de PostgreSQL.
4. Se desactivan los índices por defecto de PostgreSQL. De esta manera se puede analizar la diferencia de tiempos de cada consulta al utilizar el comando EXPLAIN ANALYZE.

```
1 SET enable_mergejoin TO OFF;
2 SET enable_hashjoin TO OFF;
3 SET enable_bitmapsScan TO OFF;
4 SET enable_sort TO OFF;
```

5. Al obtener los datos, se procede a guardar los resultados para su posterior análisis.

6. Se crean nuevos índices para las consultas, los cuales se especificarán en la sección 5.3 Optimización de consultas, y se reactivan los índices por defecto de PostgreSQL. De igual modo, se ejecuta EXPLAIN ANALYZE sobre cada consulta.

```

1 SET enable_mergejoin TO ON;
2 SET enable_hashjoin TO ON;
3 SET enable_bitmapscan TO ON;
4 SET enable_sort TO ON;
```

7. Los resultados son guardados y se procede a sus respectivos análisis y gráficos.

5.3. Optimización de consultas

Se crean los siguientes índices con el fin de identificar la utilidad e impacto del uso de índices en el tiempo de ejecución de las consultas.

5.3.1. Índices para consulta 1

```

1 DROP INDEX IF EXISTS index_estacionamiento;
2
3 CREATE INDEX index_estacionamiento
4 ON estacionamiento USING HASH (codigo);
5
6 DROP INDEX IF EXISTS index_inicioReserva;
7
8 CREATE INDEX index_inicioReserva
9 ON reserva USING btree (EXTRACT(YEAR FROM inicioReserva));
```

5.3.2. Índices para consulta 2

```

1 DROP INDEX IF EXISTS index_inicio_reserva;
2
3 CREATE INDEX index_inicio_reserva
4 ON reserva USING btree(inicioreserva);
5
6 DROP INDEX IF EXISTS index_fin_reserva;
7
8 CREATE INDEX index_fin_reserva
9 ON reserva USING btree(finreserva);
10
11 DROP INDEX IF EXISTS index_estacionamiento_reserva;
12
13 CREATE INDEX index_estacionamiento_reserva
14 ON reserva USING hash(codigo_e);
```

5.3.3. Índices para consulta 3

```

1 DROP INDEX IF EXISTS index_placa;
2
3 CREATE INDEX index_placa
4 ON vehiculo USING hash(placa);

```

5.3.4. Identificación de índices implementados por Postgresql

Se ejecutó el siguiente comando con el fin de conocer los índices que genera Postgresql, en base a los resultados se determina los índices adicionales que implementar.

```

1 SELECT * FROM pg_indexes
2 WHERE tablename NOT LIKE 'pg%';

```

El resultado es el siguiente:

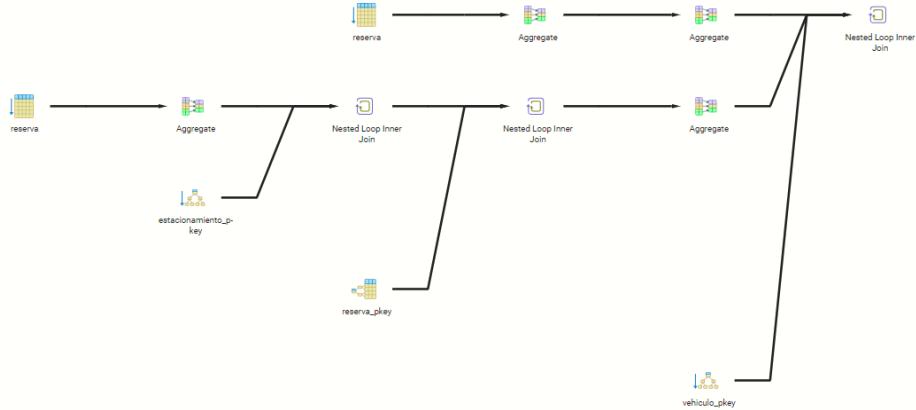
The screenshot shows a PostgreSQL query results table titled 'Input To Search Data' with 13 rows of data. The columns are: row#, schemaname, name, tablename, indexname, tablespace, and indexdef. The table lists 13 unique indices (sre_sin_indice_1000 through sre_sin_indice_1000) across various tables like estacionamiento, usuario, vehiculo, etc., with their respective index definitions.

row#	schemaname	name	tablename	indexname	tablespace	indexdef
1	sre_sin	sre_sin_indice_1000	estacionamiento	estacionamiento_pkey	(NULL)	CREATE UNIQUE INDEX estacionamiento_pkey ON sre_sin.estacionamiento
2	sre_sin	sre_sin_indice_1000	estacionamiento	direccion_unique	(NULL)	CREATE UNIQUE INDEX direccion_unique ON sre_sin.estacionamiento
3	sre_sin	sre_sin_indice_1000	usuario	usuario_pkey	(NULL)	CREATE UNIQUE INDEX usuario_pkey ON sre_sin.usuario
4	sre_sin	sre_sin_indice_1000	vehiculo	vehiculo_pkey	(NULL)	CREATE UNIQUE INDEX vehiculo_pkey ON sre_sin.vehiculo
5	sre_sin	sre_sin_indice_1000	aparcamiento	aparcamiento_pkey	(NULL)	CREATE UNIQUE INDEX aparcamiento_pkey ON sre_sin.aparcamiento
6	sre_sin	sre_sin_indice_1000	reserva	reserva_pkey	(NULL)	CREATE UNIQUE INDEX reserva_pkey ON sre_sin.reserva
7	sre_sin	sre_sin_indice_1000	personal	personal_pkey	(NULL)	CREATE UNIQUE INDEX personal_pkey ON sre_sin.personal
8	sre_sin	sre_sin_indice_1000	vip	vip_pkey	(NULL)	CREATE UNIQUE INDEX vip_pkey ON sre_sin.vip
9	sre_sin	sre_sin_indice_1000	empresarial	empresarial_pkey	(NULL)	CREATE UNIQUE INDEX empresarial_pkey ON sre_sin.empresarial
10	sre_sin	sre_sin_indice_1000	beneficio	beneficio_pkey	(NULL)	CREATE UNIQUE INDEX beneficio_pkey ON sre_sin.beneficio
11	sre_sin	sre_sin_indice_1000	posee_b_vip	posee_b_vip_pkey	(NULL)	CREATE UNIQUE INDEX posee_b_vip_pkey ON sre_sin.posee_b_vip
12	sre_sin	sre_sin_indice_1000	posee_b_emp	posee_b_emp_pkey	(NULL)	CREATE UNIQUE INDEX posee_b_emp_pkey ON sre_sin.posee_b_emp
13	sre_sin	sre_sin_indice_1000	usuario	index_cellular	(NULL)	CREATE INDEX index_cellular ON sre_sin.usuario

5.4. Medición de tiempos

5.4.1. Ejecución de consulta 1

5.4.1.1. Ejecución sin índices para 1k datos

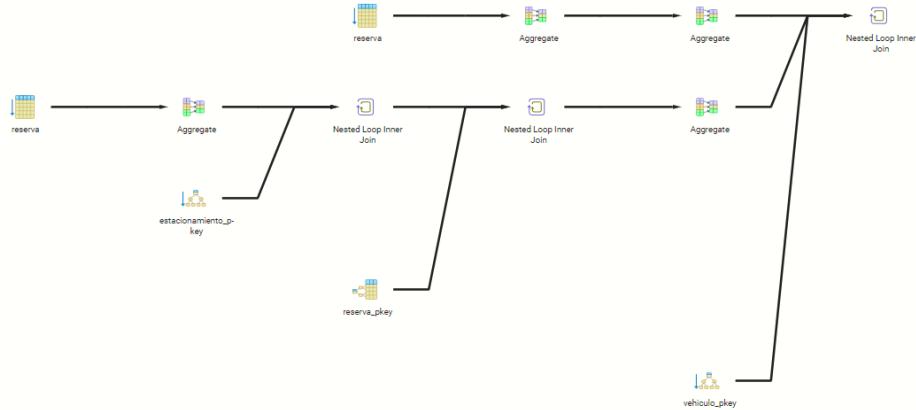


```

1 | Nested Loop (cost=65.98..73.61 rows=1 width=7) (actual time=7.054..23.463 rows=111 loops=1)
2 | [...] InitPlan 1 (returns $0)
3 | [...]-> Aggregate (cost=28.87..28.88 rows=1 width=8) (actual time=0.438..0.439 rows=1 loops=1)
4 | [...]-> HashAggregate (cost=26.66..27.64 rows=98 width=12) (actual time=0.419..0.432 rows=100 loops=1)
5 | [...] Group Key: reserva_1.codigo_e
6 | [...] Batches: 1 Memory Usage: 32kB
7 | [...]-> Seq Scan on reserva reserva_1 (cost=0.00..25.00 rows=333 width=4) (actual time=0.010..0.286 rows=786 loops=1)
8 | [...] Filter: (EXTRACT(year FROM inicioreserva) >= 2019::numeric)
9 | [...] Rows Removed by Filter: 214
10 | [...]-> HashAggregate (cost=36.83..36.84 rows=1 width=9) (actual time=5.526..5.594 rows=110 loops=1)
11 | [...] Group Key: (r.dni)::text
12 | [...] Batches: 1 Memory Usage: 40kB
13 | [...]-> Nested Loop (cost=27.21..36.83 rows=1 width=9) (actual time=2.372..5.463 rows=110 loops=1)
14 | [...]-> Nested Loop (cost=26.94..36.21 rows=1 width=8) (actual time=1.154..1.219 rows=11 loops=1)
15 | [...]-> HashAggregate (cost=26.66..27.89 rows=1 width=12) (actual time=0.934..0.956 rows=11 loops=1)
16 | [...] Group Key: reserva.codigo_e
17 | [...] Filter: (count(reserva.codigo_e) = $0)
18 | [...] Batches: 1 Memory Usage: 32kB
19 | [...] Rows Removed by Filter: 89
20 | [...]-> Seq Scan on reserva (cost=0.00..25.00 rows=333 width=4) (actual time=0.030..0.345 rows=786 loops=1)
21 | [...] Filter: (EXTRACT(year FROM inicioreserva) >= '2019'::numeric)
22 | [...] Rows Removed by Filter: 214
23 | [...]-> Index Only Scan using estacionamiento_pkey on estacionamiento_e (cost=0.28..8.29 rows=1 width=4) (actual time=0.023..0.023 rows=1 loops=11)
24 | [...] Index Cond: (codigo = reserva.codigo_e)
25 | [...] Heap Fetches: 11
26 | [...]-> Index Scan using reserva_pkey on reserva r (cost=0.28..0.52 rows=10 width=13) (actual time=0.381..0.383 rows=10 loops=11)
27 | [...] Index Cond: (codigo_e = e.codigo)
28 | [...]-> Index Only Scan using vehiculo_pkey on vehiculo (cost=0.28..7.88 rows=1 width=16) (actual time=0.141..0.161 rows=1 loops=110)
29 | [...] Index Cond: (dni = (r.dni)::text)
30 | [...] Heap Fetches: 111
31 | Planning Time: 9.666 ms
32 | Execution Time: 23.613 ms

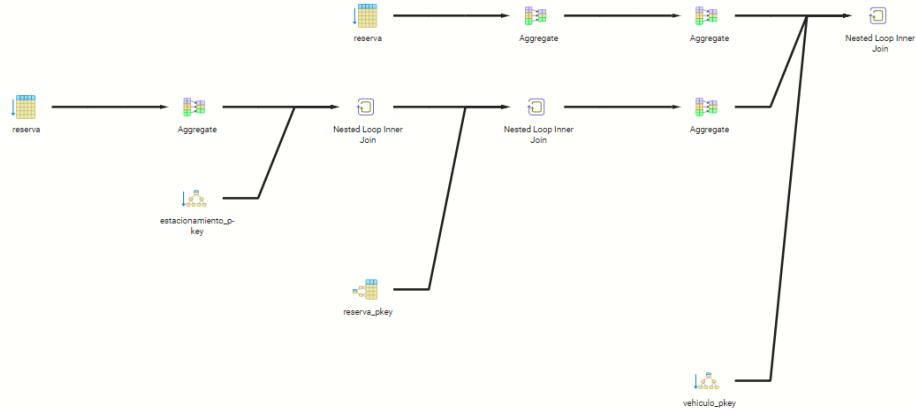
```

5.4.1.2. Ejecución sin índices para 10k datos



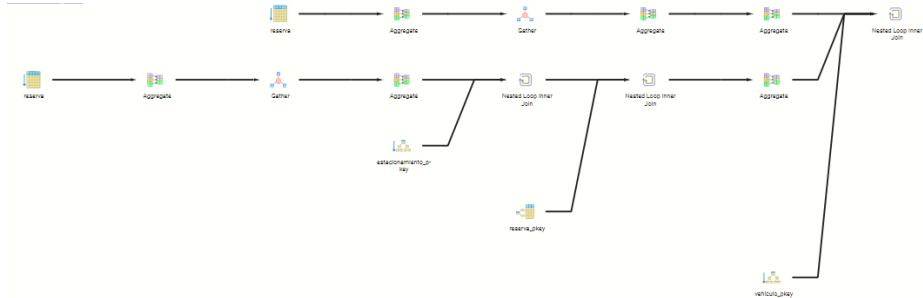
1	Nested Loop (cost=580.70..957.11 rows=5 width=7) (actual time=94.822..505.304 rows=1092 loops=1)
2	[...] InitPlan 1 (returns \$0)
3	[...] >> Aggregate (cost=282.78..282.79 rows=1 width=8) (actual time=3.890..3.891 rows=1 loops=1)
4	[...] >> HashAggregate (cost=260.67..270.50 rows=983 width=12) (actual time=3.728..3.848 rows=1000 loops=1)
5	[...] Group Key: <code>reserva_1.codigo_e</code>
6	[...] Batches: 1 Memory Usage: 193kB
7	[...] >> Seq Scan on <code>reserva_1</code> (cost=0.00..244.00 rows=3333 width=4) (actual time=0.021..2.482 rows=7900 loops=1)
8	[...] Filter: (EXTRACT(year FROM <code>inicioreserva</code>) >= '2019'::numeric)
9	[...] Rows Removed by Filter: 2100
10	[...] >> HashAggregate (cost=297.62..297.67 rows=5 width=9) (actual time=77.182..77.724 rows=1004 loops=1)
11	[...] Group Key: <code>(r.dni)::text</code>
12	[...] Batches: 1 Memory Usage: 217kB
13	[...] >> Nested Loop (cost=261.24..297.61 rows=5 width=9) (actual time=63.462..75.812 rows=1010 loops=1)
14	[...] >> Nested Loop (cost=260.95..294.52 rows=5 width=8) (actual time=62.462..65.460 rows=101 loops=1)
15	[...] >> HashAggregate (cost=260.67..272.95 rows=5 width=12) (actual time=61.860..62.056 rows=101 loops=1)
16	[...] Group Key: <code>reserva.codigo_e</code>
17	[...] Filter: (count(<code>reserva.codigo_e</code>) = \$0)
18	[...] Batches: 1 Memory Usage: 193kB
19	[...] Rows Removed by Filter: 899
20	[...] >> Seq Scan on <code>reserva</code> (cost=0.00..244.00 rows=3333 width=4) (actual time=3.334..56.221 rows=7900 loops=1)
21	[...] Filter: (EXTRACT(year FROM <code>inicioreserva</code>) >= '2019'::numeric)
22	[...] Rows Removed by Filter: 2100
23	[...] >> Index Only Scan using <code>estacionamiento_pkey</code> on <code>estacionamiento_e</code> (cost=0.29..4.30 rows=1 width=4) (actual time=0.033..0.033 rows=1 loops=101)
24	[...] Index Cond: (<code>codigo</code> = <code>reserva.codigo_e</code>)
25	[...] Heap Fetches: 0
26	[...] >> Index Scan using <code>reserva_pkey</code> on <code>reserva r</code> (cost=0.29..0.52 rows=10 width=13) (actual time=0.099..0.101 rows=10 loops=101)
27	[...] Index Cond: (<code>codigo_e</code> = <code>e.codigo</code>)
28	[...] >> Index Only Scan using <code>vehiculo_pkey</code> on <code>vehiculo</code> (cost=0.29..75.32 rows=1 width=16) (actual time=0.204..0.425 rows=1 loops=1004)
29	[...] Index Cond: (<code>dni</code> = <code>(r.dni)::text</code>)
30	[...] Heap Fetches: 0
31	Planning Time: 5.042 ms
32	Execution Time: 505.615 ms

5.4.1.3. Ejecución sin índices para 100k datos



1	Nested Loop (cost=5276.50..9028.61 rows=9 width=7) (actual time=80.709..492.644 rows=195 loops=1)
2	[...] InitPlan 1 (returns \$0)
3	[...] >> Aggregate (cost=2624.16..2624.18 rows=1 width=8) (actual time=34.146..34.147 rows=1 loops=1)
4	[...] >> HashAggregate (cost=2601.66..2611.66 rows=1000 width=12) (actual time=33.977..34.099 rows=1000 loops=1)
5	[...] Group Key: reserva_1.codigo_e
6	[...] Batches: 1 Memory Usage: 193kB
7	[...] >> Seq Scan on reserva reserva_1 (cost=0.00..2435.00 rows=33333 width=4) (actual time=0.019..23.255 rows=78675 loops=1)
8	[...] Filter: (EXTRACT(year FROM inicioreserva) >= '2019':numeric)
9	[...] Rows Removed by Filter: 21325
10	[...] >> HashAggregate (cost=2651.90..2651.95 rows=5 width=9) (actual time=77.476..77.656 rows=100 loops=1)
11	[...] Group Key: (r.dni):text
12	[...] Batches: 1 Memory Usage: 40kB
13	[...] >> Nested Loop (cost=2602.38..2651.89 rows=5 width=9) (actual time=77.362..77.450 rows=100 loops=1)
14	[...] >> Nested Loop (cost=2601.96..2635.77 rows=5 width=8) (actual time=77.328..77.394 rows=1 loops=1)
15	[...] >> HashAggregate (cost=2601.66..2614.16 rows=5 width=12) (actual time=76.811..76.876 rows=1 loops=1)
16	[...] Group Key: reserva.codigo_e
17	[...] Filter: (count(reserva.codigo_e) = \$0)
18	[...] Batches: 1 Memory Usage: 193kB
19	[...] Rows Removed by Filter: 999
20	[...] >> Seq Scan on reserva (cost=0.00..2435.00 rows=33333 width=4) (actual time=0.148..30.138 rows=78675 loops=1)
21	[...] Filter: (EXTRACT(year FROM inicioreserva) >= '2019':numeric)
22	[...] Rows Removed by Filter: 21325
23	[...] >> Index Only Scan using estacionamiento_pkey on estacionamiento e (cost=0.29..4.31 rows=1 width=4) (actual time=0.513..0.513 rows=1 loops=1)
24	[...] Index Cond: (codigo = reserva.codigo_e)
25	[...] Heap Fetches: 0
26	[...] >> Index Scan using reserva_pkey on reserva r (cost=0.42..2.23 rows=100 width=13) (actual time=0.031..0.042 rows=100 loops=1)
27	[...] Index Cond: (codigo_e = e.codigo)
28	[...] >> Index Only Scan using vehiculo_pkey on vehiculo (cost=0.42..750.48 rows=2 width=16) (actual time=1.415..4.146 rows=2 loops=100)
29	[...] Index Cond: (dni = (r.dni):text)
30	[...] Heap Fetches: 0
31	Planning Time: 5.440 ms
32	Execution Time: 493.495 ms

5.4.1.4. Ejecución sin índices para 1000k datos



```

1 | Nested Loop (cost=39702.40..414746.11 rows=537 width=7) (actual time=438.835..4606.608 rows=1076 loops=1)
2 | [...] InitPlan 1 (returns $1)
3 | [...] -> Aggregate (cost=19708.66..19708.67 rows=1 width=8) (actual time=198.897..198.960 rows=1 loops=1)
4 | [...] -> Finalize HashAggregate (cost=19484.29..19584.01 rows=9972 width=12) (actual time=194.269..197.706 rows=10000 loops=1)
5 | [...] Group Key: reserva_1.codigo_e
6 | [...] Batches: 1 Memory Usage: 1425kB
7 | [...] -> Gather (cost=17290.44..19384.57 rows=19944 width=12) (actual time=187.969..190.458 rows=12147 loops=1)
8 | [...] Workers Planned: 2
9 | [...] Workers Launched: 2
10 | [...] -> Partial HashAggregate (cost=16290.44..16390.17 rows=9972 width=12) (actual time=157.230..158.032 rows=4049 loops=3)
11 | [...] Group Key: reserva_1.codigo_e
12 | [...] Batches: 1 Memory Usage: 913kB
13 | [...] Worker 0: Batches: 1 Memory Usage: 913kB
14 | [...] Worker 1: Batches: 1 Memory Usage: 913kB
15 | [...] -> Parallel Seq Scan on reserva reserva_1 (cost=0.00..15596.00 rows=138889 width=4) (actual time=1.023..112.989 rows=262585 loops=3)
16 | [...] Filter: (EXTRACT(year FROM inicioreserva) >= '2019'::numeric)
17 | [...] Rows Removed by Filter: 70749
18 | [...] -> HashAggregate (cost=19993.31..19993.81 rows=50 width=9) (actual time=435.553..435.875 rows=100 loops=1)
19 | [...] Group Key: (r.dni)::text
20 | [...] Batches: 1 Memory Usage: 40kB
21 | [...] -> Nested Loop (cost=19485.14..19993.18 rows=50 width=9) (actual time=434.060..435.462 rows=100 loops=1)
22 | [...] -> Nested Loop (cost=19484.71..19831.56 rows=50 width=8) (actual time=433.961..435.330 rows=1 loops=1)
23 | [...] -> Finalize HashAggregate (cost=19484.29..19608.94 rows=50 width=12) (actual time=433.304..434.672 rows=1 loops=1)
24 | [...] Group Key: reserva.codigo_e
25 | [...] Filter: (count(reserva.codigo_e) = $1)
26 | [...] Batches: 1 Memory Usage: 1425kB
27 | [...] Rows Removed by Filter: 9999
28 | [...] -> Gather (cost=17290.44..19384.57 rows=19944 width=12) (actual time=228.259..230.563 rows=11826 loops=1)
29 | [...] Workers Planned: 2
30 | [...] Workers Launched: 2

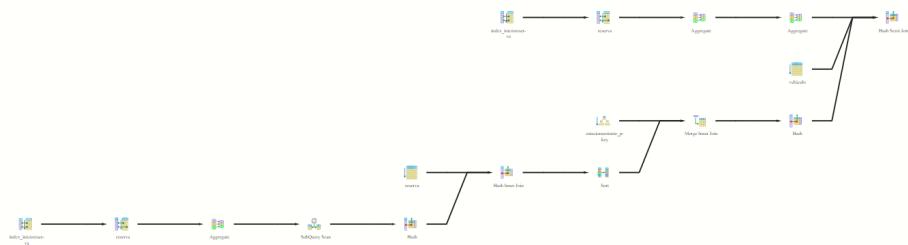
```

```

31 [...] -> Partial HashAggregate (cost=16290.44..16390.17 rows=9972 width=12) (actual time=156.163..156.835 rows=3942 loops=3)
32 [...] Group Key: reserva.codigo_e
33 [...] Batches: 1 Memory Usage: 913kB
34 [...] Worker 0: Batches: 1 Memory Usage: 657kB
35 [...] Worker 1: Batches: 1 Memory Usage: 913kB
36 [...] -> Parallel Seq Scan on reserva (cost=0.00..15596.00 rows=138889 width=4) (actual time=0.665..112.151 rows=262585 loops=3)
37 [...] Filter: (EXTRACT(year FROM inicioreserva) >= 2019::numeric)
38 [...] Rows Removed by Filter: 70749
39 [...] -> Index Only Scan using estacionamiento_pkey on estacionamiento_e (cost=0.42..4.44 rows=1 width=4) (actual time=0.651..0.652 rows=1 loops=1)
40 [...] Index Cond: (codigo_e = reserva.codigo_e)
41 [...] Heap Fetches: 0
42 [...] -> Index Scan using reserva_pkey on reserva_r (cost=0.42..2.23 rows=100 width=13) (actual time=0.093..0.115 rows=100 loops=1)
43 [...] Index Cond: (codigo_e = e.codigo)
44 [...] -> Index Only Scan using vehiculo_pkey on vehiculo (cost=0.42..7500.76 rows=11 width=16) (actual time=3.806..41.697 rows=11 loops=100)
45 [...] Index Cond: (dni = (r.dni)::text)
46 [...] Heap Fetches: 0
47 Planning Time: 4.570 ms
48 Execution Time: 4610.865 ms

```

5.4.1.5. Ejecución con índices para 1k datos

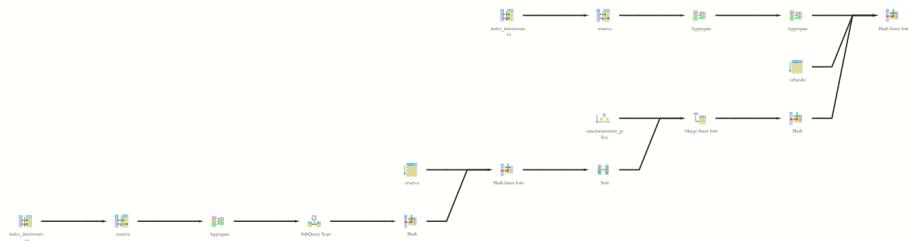


```

1 Hash Semi Join (cost=85.28..108.92 rows=1 width=7) (actual time=2.097..2.412 rows=111 loops=1)
2 [...] Hash Cond: ((vehiculo.dn)::text = (r.dni)::text)
3 [...] InitPlan 1 (returns $0)
4 [...]-> Aggregate (cost=28.87..28.88 rows=1 width=8) (actual time=0.766..0.767 rows=1 loops=1)
5 [...]-> HashAggregate (cost=26.66..27.64 rows=98 width=12) (actual time=0.729..0.751 rows=100 loops=1)
6 [...] Group Key: reserva_1.codigo_e
7 [...] Batches: 1 Memory Usage: 32kB
8 [...]-> Seq Scan on reserva reserva_1 (cost=0.00..25.00 rows=333 width=4) (actual time=0.017..0.498 rows=786 loops=1)
9 [...] Filter: (EXTRACT(year FROM inicioreserva) >= 2019::numeric)
10 [...] Rows Removed by Filter: 214
11 [...]-> Seq Scan on vehiculo (cost=0.00..21.00 rows=1000 width=16) (actual time=0.027..0.155 rows=1000 loops=1)
12 [...]-> Hash (cost=56.39..56.39 rows=1 width=9) (actual time=2.041..2.045 rows=110 loops=1)
13 [...] Buckets: 1024 Batches: 1 Memory Usage: 13kB
14 [...]-> Merge Join (cost=51.09..56.39 rows=1 width=9) (actual time=1.932..2.009 rows=110 loops=1)
15 [...] Merge Cond: (e.codigo_e = r.codigo_e)
16 [...]-> Index Only Scan using estacionamiento_pkkey on estacionamiento e (cost=0.28..49.27 rows=1000 width=4) (actual time=0.025..0.059 rows=101 loops=1)
17 [...] Heap Fetches: 101
18 [...]-> Sort (cost=50.81..50.84 rows=10 width=17) (actual time=1.898..1.909 rows=110 loops=1)
19 [...] Sort Key: r.codigo_e
20 [...] Sort Method: quicksort Memory: 33kB
21 [...]-> Hash Join (cost=27.91..50.65 rows=10 width=17) (actual time=1.600..1.868 rows=110 loops=1)
22 [...] Hash Cond: (r.codigo_e = reservas_por_estacionamiento_desde_2019.codigo_e)
23 [...]-> Seq Scan on reserva r (cost=0.00..20.00 rows=1000 width=13) (actual time=0.017..0.133 rows=1000 loops=1)
24 [...]-> Hash (cost=27.90..27.90 rows=1 width=4) (actual time=1.560..1.562 rows=11 loops=1)
25 [...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
26 [...]-> Subquery Scan on reservas_por_estacionamiento_desde_2019 (cost=26.66..27.90 rows=1 width=4) (actual time=1.533..1.552 rows=11 loops=1)
27 [...]-> HashAggregate (cost=26.66..27.89 rows=1 width=12) (actual time=1.532..1.549 rows=11 loops=1)
28 [...] Group Key: reserva.codigo_e
29 [...] Filter: (count(reserva.codigo_e) = $0)
30 [...] Batches: 1 Memory Usage: 32kB
31 [...] Rows Removed by Filter: 89
32 [...]-> Seq Scan on reserva (cost=0.00..25.00 rows=333 width=4) (actual time=0.018..0.520 rows=786 loops=1)
33 [...] Filter: (EXTRACT(year FROM inicioreserva) >= '2019'::numeric)
34 [...] Rows Removed by Filter: 214
35 Planning Time: 1.314 ms
36 Execution Time: 2.684 ms

```

5.4.1.6. Ejecución con índices para 10k datos

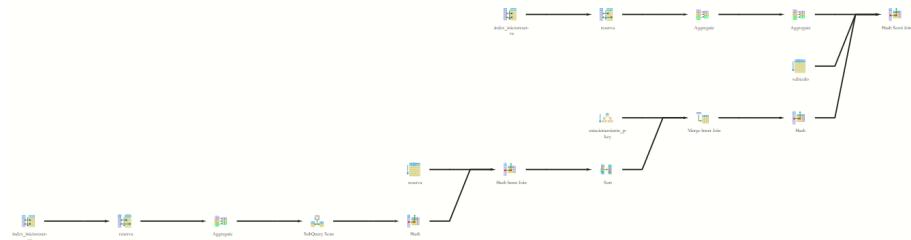


```

1 Hash Semi Join (cost=808.23..1037.53 rows=5 width=7) (actual time=18.829..22.013 rows=1092 loops=1)
2   [...] Hash Cond: ((vehiculo.dni)::text = (r.dni)::text)
3   [...] InitPlan 1 (returns $0)
4   [...]-> Aggregate (cost=282.78..282.79 rows=1 width=8) (actual time=7.487..7.489 rows=1 loops=1)
5   [...]-> HashAggregate (cost=260.67..270.50 rows=983 width=12) (actual time=7.159..7.402 rows=1000 loops=1)
6   [...] Group Key: reserva_1.codigo_e
7   [...] Batches: 1 Memory Usage: 193kB
8   [...]-> Seq Scan on reserva reserva_1 (cost=0.00..244.00 rows=3333 width=4) (actual time=0.026..4.776 rows=7900 loops=1)
9   [...] Filter: (EXTRACT(year FROM inicioreserva) >= 2019::numeric)
10  [...] Rows Removed by Filter: 2100
11  [...]-> Seq Scan on vehiculo (cost=0.00..203.00 rows=10000 width=16) (actual time=0.026..1.090 rows=10000 loops=1)
12  [...]-> Hash (cost=525.37..525.37 rows=5 width=9) (actual time=18.777..18.781 rows=1010 loops=1)
13  [...] Buckets: 1024 Batches: 1 Memory Usage: 49kB
14  [...]-> Merge Join (cost=495.12..525.37 rows=5 width=9) (actual time=18.003..18.530 rows=1010 loops=1)
15  [...] Merge Cond: (e.codigo = r.codigo_e)
16  [...]-> Index Only Scan using estacionamiento_pkkey on estacionamiento e (cost=0.29..270.29 rows=10000 width=4) (actual time=0.023..0.169 rows=974...)
17  [...] Heap Fetches: 0
18  [...]-> Sort (cost=494.84..494.96 rows=50 width=17) (actual time=17.971..18.043 rows=1010 loops=1)
19  [...] Sort Key: r.codigo_e
20  [...] Sort Method: quicksort Memory: 103kB
21  [...]-> Hash Join (cost=273.06..493.43 rows=50 width=17) (actual time=15.026..17.691 rows=1010 loops=1)
22  [...] Hash Cond: (r.codigo_e = reservas_por_estacionamiento_desde_2019.codigo_e)
23  [...]-> Seq Scan on reserva r (cost=0.00..194.00 rows=10000 width=13) (actual time=0.015..1.092 rows=10000 loops=1)
24  [...]-> Hash (cost=273.00..273.00 rows=5 width=4) (actual time=14.983..14.985 rows=101 loops=1)
25  [...] Buckets: 1024 Batches: 1 Memory Usage: 12kB
26  [...]-> Subquery Scan on reservas_por_estacionamiento_desde_2019 (cost=260.67..273.00 rows=5 width=4) (actual time=14.740..14.949 rows=101 loops=1)
27  [...]-> HashAggregate (cost=260.67..272.95 rows=5 width=12) (actual time=14.739..14.936 rows=101 loops=1)
28  [...] Group Key: reserva.codigo_e
29  [...] Filter: (count(reserva.codigo_e) = $0)
30  [...] Batches: 1 Memory Usage: 193kB
31  [...] Rows Removed by Filter: 899
32  [...]-> Seq Scan on reserva (cost=0.00..244.00 rows=3333 width=4) (actual time=0.020..4.800 rows=7900 loops=1)
33  [...] Filter: (EXTRACT(year FROM inicioreserva) >= 2019::numeric)
34  [...] Rows Removed by Filter: 2100
35  Planning Time: 1.792 ms
36  Execution Time: 22.496 ms

```

5.4.1.7. Ejecución con índices para 100k datos

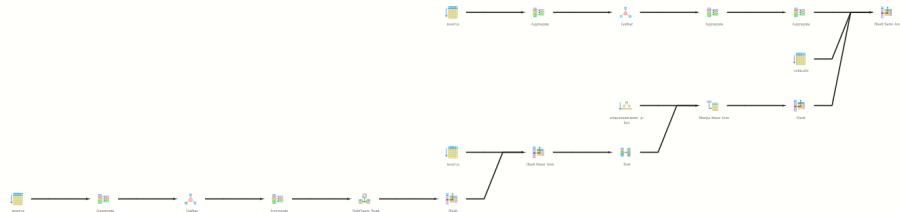


```

1 Hash Semi Join (cost=7494.96..9782.56 rows=9 width=7) (actual time=132.912..217.561 rows=195 loops=1)
2 [...] Hash Cond: ((vehiculo.dni)::text = (r.dni)::text)
3 [...] InitPlan 1 (returns $0)
4 [...]-> Aggregate (cost=2624.16..2624.18 rows=1 width=8) (actual time=47.118..47.120 rows=1 loops=1)
5 [...]-> HashAggregate (cost=2601.66..2611.66 rows=1000 width=12) (actual time=46.886..47.053 rows=1000 loops=1)
6 [...] Group Key: reserva_1.codigo_e
7 [...] Batches: 1 Memory Usage: 193kB
8 [...]-> Seq Scan on reserva reserva_1 (cost=0.00..2435.00 rows=33333 width=4) (actual time=0.025..32.576 rows=78675 loops=1)
9 [...] Filter: (EXTRACT(year FROM inicioreserva) >= '2019::numeric')
10 [...] Rows Removed by Filter: 21325
11 [...]-> Seq Scan on vehiculo (cost=0.00..2025.00 rows=100000 width=16) (actual time=0.490..72.216 rows=100000 loops=1)
12 [...]-> Hash (cost=4870.72..4870.72 rows=5 width=9) (actual time=132.350..132.354 rows=100 loops=1)
13 [...] Buckets: 1024 Batches: 1 Memory Usage: 13kB
14 [...]-> Merge Join (cost=4835.61..4870.72 rows=5 width=9) (actual time=132.303..132.327 rows=100 loops=1)
15 [...] Merge Cond: (e.codigo = r.codigo_e)
16 [...]-> Index Only Scan using estacionamiento_pkey on estacionamiento e (cost=0.29..2604.29 rows=100000 width=4) (actual time=0.034..0.119 rows=4...)
17 [...] Heap Fetches: 0
18 [...]-> Sort (cost=4835.30..4836.55 rows=500 width=17) (actual time=132.154..132.163 rows=100 loops=1)
19 [...] Sort Key: r.codigo_e
20 [...] Sort Method: quicksort Memory: 32kB
21 [...]-> Hash Join (cost=2614.28..4812.89 rows=500 width=17) (actual time=122.357..132.128 rows=100 loops=1)
22 [...] Hash Cond: (r.codigo_e = reservas_por_estacionamiento_desde_2019.codigo_e)
23 [...]-> Seq Scan on reserva r (cost=0.00..1935.00 rows=100000 width=13) (actual time=0.023..7.278 rows=100000 loops=1)
24 [...]-> Hash (cost=2614.22..2614.22 rows=5 width=4) (actual time=115.561..115.564 rows=1 loops=1)
25 [...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
26 [...]-> Subquery Scan on reservas_por_estacionamiento_desde_2019 (cost=2601.66..2614.22 rows=5 width=4) (actual time=115.486..115.557 rows=1 loops=1)
27 [...]-> HashAggregate (cost=2601.66..2614.16 rows=5 width=12) (actual time=115.485..115.555 rows=1 loops=1)
28 [...] Group Key: reserva.codigo_e
29 [...] Filter: (count(reserva.codigo_e) = $0)
30 [...] Batches: 1 Memory Usage: 193kB
31 [...] Rows Removed by Filter: 999
32 [...]-> Seq Scan on reserva (cost=0.00..2435.00 rows=33333 width=4) (actual time=0.017..46.818 rows=78675 loops=1)
33 [...] Filter: (EXTRACT(year FROM inicioreserva) >= '2019::numeric')
34 [...] Rows Removed by Filter: 21325
35 Planning Time: 20.049 ms
36 Execution Time: 217.956 ms

```

5.4.1.8. Ejecución con índices para 1000k datos



1	Hash Semi Join (cost=61959.91..84833.88 rows=537 width=7) (actual time=831.550..1351.096 rows=1076 loops=1)
2	[...] Hash Cond: ((vehiculo.dni)::text = (r.dni)::text)
3	[...] InitPlan 1 (returns \$1)
4	[...]-> Aggregate (cost=19708.66..19708.67 rows=1 width=8) (actual time=274.517..274.594 rows=1 loops=1)
5	[...]-> Finalize HashAggregate (cost=19484.29..19584.01 rows=9972 width=12) (actual time=272.355..274.014 rows=10000 loops=1)
6	[...] Group Key: reserva_1.codigo_e
7	[...] Batches: 1 Memory Usage: 1425kB
8	[...]-> Gather (cost=17290.44..19384.57 rows=19944 width=12) (actual time=261.597..265.869 rows=12140 loops=1)
9	[...] Workers Planned: 2
10	[...] Workers Launched: 2
11	[...]-> Partial HashAggregate (cost=16290.44..16390.17 rows=9972 width=12) (actual time=220.336..221.588 rows=4047 loops=3)
12	[...] Group Key: reserva_1.codigo_e
13	[...] Batches: 1 Memory Usage: 913kB
14	[...] Worker 0: Batches: 1 Memory Usage: 913kB
15	[...] Worker 1: Batches: 1 Memory Usage: 913kB
16	[...]-> Parallel Seq Scan on reserva reserva_1 (cost=0.00..15596.00 rows=138889 width=4) (actual time=1.187..157.796 rows=262585 loops=3)
17	[...] Filter: (EXTRACT(year FROM inicioreserva) >= '2019'::numeric)
18	[...] Rows Removed by Filter: 70749
19	[...]-> Seq Scan on vehiculo (cost=0.00..20243.00 rows=1000000 width=16) (actual time=0.546..380.142 rows=1000000 loops=1)
20	[...]-> Hash (cost=42250.62..42250.62 rows=50 width=9) (actual time=825.518..825.578 rows=100 loops=1)
21	[...] Buckets: 1024 Batches: 1 Memory Usage: 13kB
22	[...]-> Merge Join (cost=41890.80..42250.62 rows=50 width=9) (actual time=825.458..825.537 rows=100 loops=1)
23	[...] Merge Cond: (e.codigo = r.codigo_e)
24	[...]-> Index Only Scan using estacionamiento_pkey on estacionamiento e (cost=0.42..25980.45 rows=1000002 width=4) (actual time=0.039..0.376 rows=...)
25	[...] Heap Fetches: 0
26	[...]-> Sort (cost=41890.33..41902.86 rows=5014 width=17) (actual time=824.928..824.993 rows=100 loops=1)
27	[...] Sort Key: r.codigo_e
28	[...] Sort Method: quicksort Memory: 32kB
29	[...]-> Hash Join (cost=19610.06..41582.17 rows=5014 width=17) (actual time=670.017..824.936 rows=100 loops=1)
30	[...] Hash Cond: (r.codigo_e = reservas_por_estacionamiento_desde_2019.codigo_e)
31	[...]-> Seq Scan on reservas (cost=0.00..19346.00 rows=1000000 width=13) (actual time=0.072..119.149 rows=1000000 loops=1)
32	[...]-> Hash (cost=19609.44..19609.44 rows=50 width=4) (actual time=610.617..610.675 rows=1 loops=1)
33	[...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
34	[...]-> Subquery Scan on reservas_por_estacionamiento_desde_2019 (cost=19484.29..19609.44 rows=50 width=4) (actual time=609.181..610.668 rows=...)
35	[...]-> Finalize HashAggregate (cost=19484.29..19608.94 rows=50 width=12) (actual time=609.180..610.666 rows=1 loops=1)
36	[...] Group Key: reserva.codigo_e
37	[...] Filter: (count(reserva.codigo_e) = \$1)
38	[...] Batches: 1 Memory Usage: 1425kB
39	[...] Rows Removed by Filter: 9999
40	[...]-> Gather (cost=17290.44..19384.57 rows=19944 width=12) (actual time=327.256..330.044 rows=12139 loops=1)
41	[...] Workers Planned: 2
42	[...] Workers Launched: 2
43	[...]-> Partial HashAggregate (cost=16290.44..16390.17 rows=9972 width=12) (actual time=258.359..259.183 rows=4046 loops=3)
44	[...] Group Key: reserva.codigo_e
45	[...] Batches: 1 Memory Usage: 913kB
46	[...] Worker 0: Batches: 1 Memory Usage: 913kB
47	[...] Worker 1: Batches: 1 Memory Usage: 913kB
48	[...]-> Parallel Seq Scan on reserva (cost=0.00..15596.00 rows=138889 width=4) (actual time=1.780..185.409 rows=262585 loops=3)
49	[...] Filter: (EXTRACT(year FROM inicioreserva) >= '2019'::numeric)
50	[...] Rows Removed by Filter: 70749
51	Planning Time: 15.957 ms
52	Execution Time: 1353.593 ms

5.4.1.9. Explicación de consulta 1

Sin índices

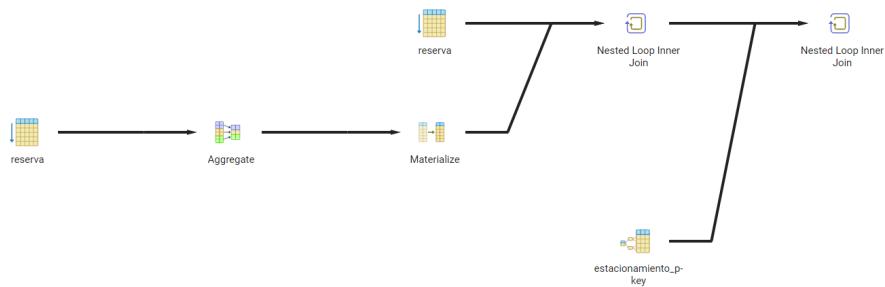
El plan de ejecución es similar para las tablas con una cantidad de tuplas 1k, 10k y 100k. Sobre la tabla reservas se crea una vista a partir de la función de agregación COUNT() agrupándolos por el código de estacionamiento y filtrando aquellas tuplas donde el año de la fecha de inicio reserva sea igual o mayor a 2019. A partir de esta vista, se utiliza la función de agregación MAX() y con la tabla resultante se realiza un nested loop con la tabla estacionamiento. Nuevamente, con la tabla resultante se realiza un nested loop con la tabla reserva. Finalmente, para obtener las matrículas automovilísticas, se llama a la tabla vehículos para realizar un nested loop con dichas tablas. Por otro lado, para las tablas con 1M de tuplas se añade al plan de ejecución el método gather.

Con índices

En este caso, el plan de ejecución hace uso de los índices index estacionamiento e index inicioReserva, además de permitir el uso Hash Inner Join y Merge Inner Join. Inicialmente, creamos la vista al igual que el plan de ejecución sin índices, pero esta vez filtramos los años con ayuda de un btree. Este mismo índice es utilizado también al momento de emplear la función de agregación MAX() para luego realizar un Hash Inner Join y encontrar los estacionamientos con mayores reservas. Se ordenan los datos de la tabla resultante para luego realizar un Merge Inner Join junto con la tabla de estacionamientos haciendo uso del índice index estacionamiento. Finalmente, se realiza un Hash Semi Join para obtener los datos de interés entre el par de tablas resultantes y la tabla de vehículos. Nuevamente, en el caso de las consultas con tablas de 1M de datos se incluye en el plan de ejecución el método gather.

5.4.2. Ejecución de consulta 2

5.4.2.1. Ejecución sin índices para 1k datos

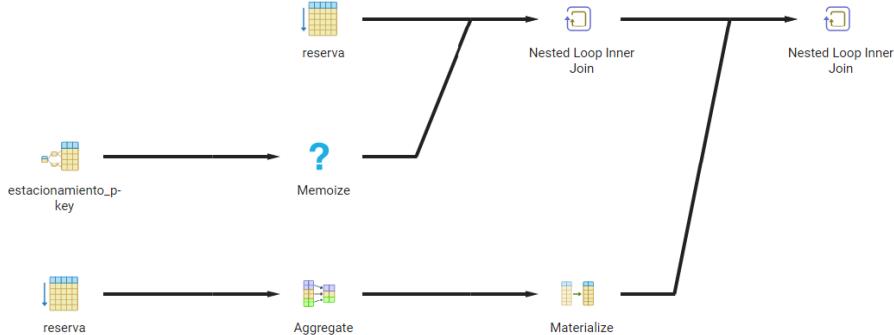


```

Nested Loop (cost=41.59..1394.41 rows=87 width=50) (actual time=0.312..0.312 rows=0 loops=1)
  . . .> Nested Loop (cost=31.31..1359.40 rows=87 width=50) (actual time=0.311..0.311 rows=0 loops=1)
    . . .> Join Filter ((reserva.mn).text = (reserva_1.dni).text)
    . . .> Seq Scan on reserva (cost=0.00..20.00 rows=1000 width=13) (actual time=0.020..0.056 rows=1000 loops=1)
      . . .> Materialize (cost=31.31..35.89 rows=87 width=9) (actual time=0.000..0.000 rows=0 loops=1)
      . . .> HashAggregate (cost=31.31..34.59 rows=87 width=9) (actual time=0.184..0.185 rows=0 loops=1)
        . . .> Group Key: reserva_1.dni
        . . .> Filter (count(reserva_1.codigo_e) > 1)
        . . .> Batches: 1 Memory Usage: 616B
      . . .> Rows Removed by Filter: 264
    . . .> Seq Scan on reserva reserva_1 (cost=0.00..30.00 rows=262 width=13) (actual time=0.126..rows=264 loops=1)
      . . .> Filter ((in(reserva => 2018-12-02 00:00:00 timestamp without time zone) AND (in(reserva => 2019-03-01 00:00:00 timestamp without time zone)) OR ((in(reserva => 2018-12-02 00:00:00 timestamp without time zone) AND (in(reserva => 2019-03-01 00:00:00 timestamp without time zone)))
      . . .> Rows Removed by Filter: 736
      . . .> Index Scan using estacionamiento_pkey on estacionamiento (cost=0.28..0.36 rows=1 width=50) (never executed)
      . . .> Index Cond: (codigo_e = reserva.codigo_e)
Planning Time: 0.820 ms
Execution Time: 0.347 ms

```

5.4.2.2. Ejecución sin índices para 10k datos

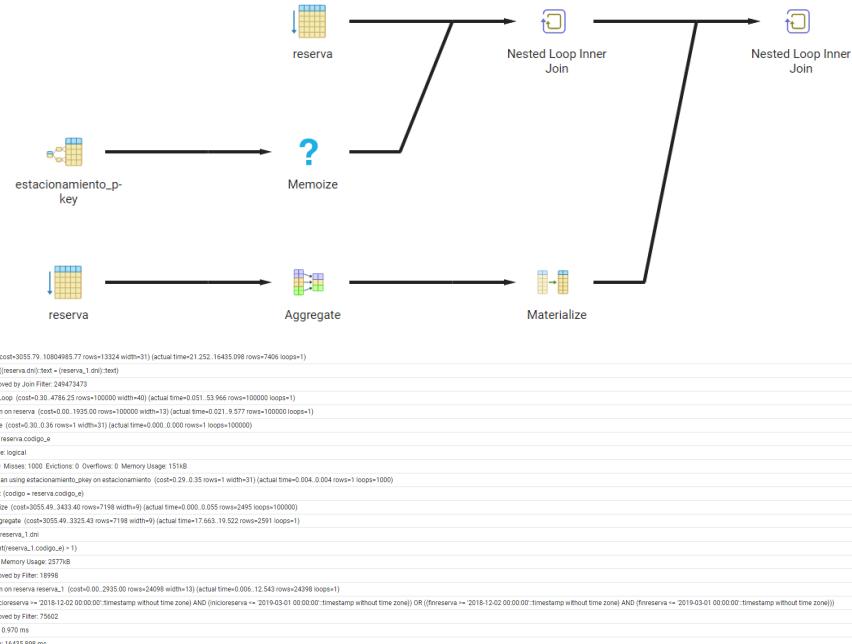


```

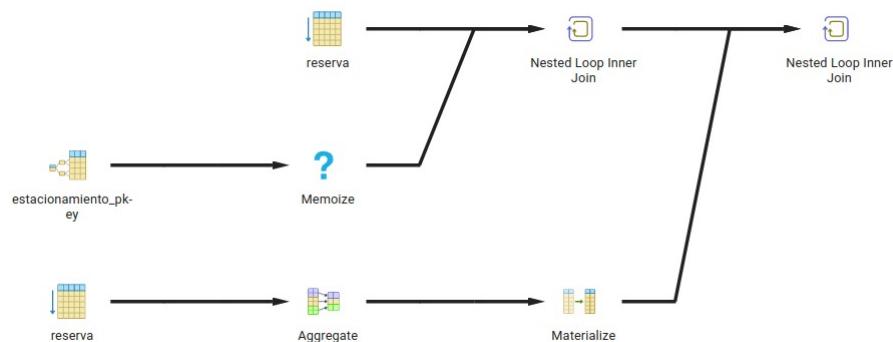
Nested Loop (cost=306.49..122335.48 rows=847 width=55) (actual time=1.957..24.723 rows=52 loops=1)
  . . .> Join Filter ((reserva.mn).text = (reserva_1.dni).text)
  . . .> Rows Removed by Filter: 249318
  . . .> Nested Loop (cost=0.30..801.99 rows=10000 width=64) (actual time=0.039..4.368 rows=10000 loops=1)
    . . .> Seq Scan on reserva (cost=0.00..194.00 rows=10000 width=13) (actual time=0.008..0.050 rows=10000 loops=1)
    . . .> Memoize (cost=0.30..37.37 rows=1 width=55) (actual time=0.000..0.000 rows=1 loops=10000)
    . . .> Cache Key: reserva.codigo_e
    . . .> Cache Mode: logical
    . . .> Hit: 9000 Misses: 1000 Evictions: 0 Memory Usage: 1518B
    . . .> Index Scan using estacionamiento_pkey on estacionamiento (cost=0.29..0.36 rows=1 width=55) (actual time=0.001..0.001 rows=1 loops=10000)
    . . .> Index Cond: (codigo_e = reserva.codigo_e)
    . . .> Materialize (cost=306.19..346.50 rows=898 width=9) (actual time=0.000..0.001 rows=25 loops=10000)
    . . .> HashAggregate (cost=306.19..336.48 rows=808 width=9) (actual time=1.357..1.514 rows=25 loops=1)
      . . .> Group Key: reserva_1.dni
      . . .> Filter (count(reserva_1.codigo_e) > 1)
      . . .> Batches: 1 Memory Usage: 3698B
      . . .> Rows Removed by Filter: 2397
      . . .> Seq Scan on reserva reserva_1 (cost=0.00..294.00 rows=2438 width=13) (actual time=0.005..0.932 rows=2447 loops=1)
      . . .> Filter ((in(reserva => 2018-12-02 00:00:00 timestamp without time zone) AND (in(reserva => 2019-03-01 00:00:00 timestamp without time zone)) OR ((in(reserva => 2018-12-02 00:00:00 timestamp without time zone) AND (in(reserva => 2019-03-01 00:00:00 timestamp without time zone)))
      . . .> Rows Removed by Filter: 7555
Planning Time: 0.680 ms
Execution Time: 24.843 ms

```

5.4.2.3. Ejecución sin índices para 100k datos



5.4.2.4. Ejecución sin índices para 1000k datos

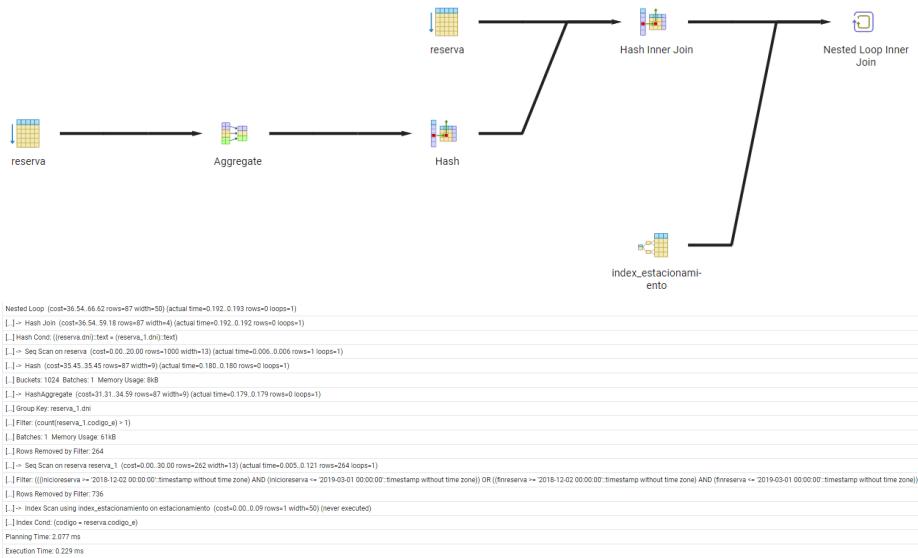


```

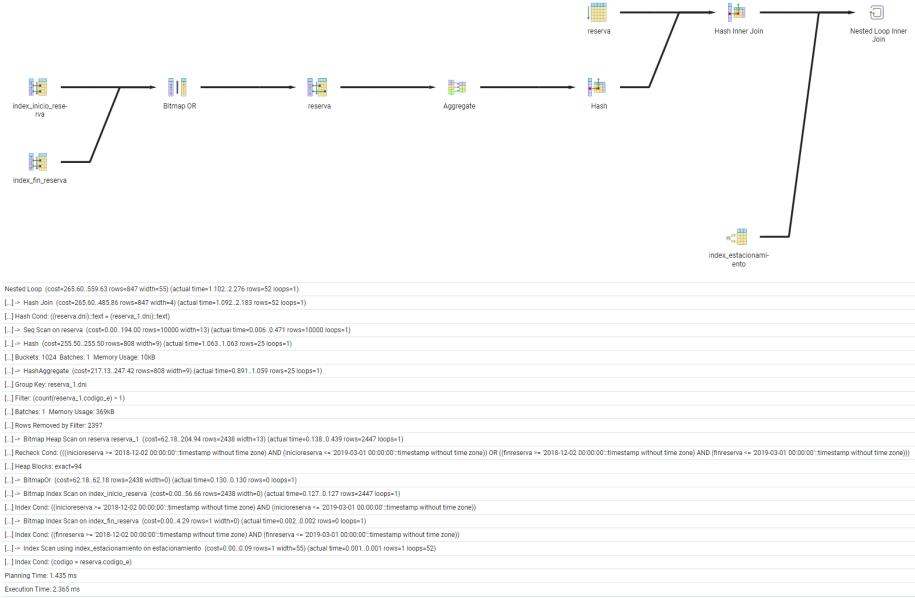
Nested Loop (cost=45995.05..442993251.39 rows=317259 width=23) (actual time=424.866..381.9924.767 rows=752016 loops=1)
  Join Filter: ((reserva.dn).text = (reserva_1.dn).text)
  Rows Removed by Join Filter: 43118586760
  > Nested Loop (cost=42..49150.52 rows=1000000 width=32) (actual time=159.450..7289.776 rows=1000000 loops=1)
    > Seq Scan on reserva (cost=0.00..19246.00 rows=1000000 width=13) (actual time=0.012..176.691 rows=1000000 loops=1)
    > MemSort (cost=0.42..0.49 rows=1 width=23) (actual time=0.002..0.002 rows=1 loops=1000000)
      Cache Key: reserva.codigo_e
      Cache Mode: logical
      Hits: 990000 Misses: 10000 Evictions: 0 Overflows: 0 Memory Usage: 1556kB
    > Index Scan using estacionamiento_pk on estacionamiento (cost=0..42..0.48 rows=1 width=23) (actual time=0.016..0.016 rows=1 loops=10000)
      Index Cond: (codigo_e = reserva.codigo_e)
    > Materialize (cost=45994.61..49045.13 rows=29520 width=9) (actual time=0.000..1.462 rows=43319 loops=100000)
    > HashAggregate (cost=45994.61..48602.33 rows=29520 width=9) (actual time=185.270..255.460 rows=69990 loops=1)
      Group Key: reserva_1.dn
      Filter: (count(reserva_1.codigo_e) > 1)
      Planned Partitions: 4 Batches: 5 Memory Usage: 41454kB Disk Usage: 7136kB
      Rows Removed by Filter: 2117
      > Seq Scan on reserva_1 (cost=0..0..29341.00 rows=245892 width=13) (actual time=0.008..76.590 rows=243778 loops=1)
      Filter: ((reserva >= '2018-12-02 00:00:00'::timestamp without time zone) AND (reserva <= '2019-03-01 00:00:00'::timestamp without time zone) OR ((reserva >= '2018-12-02 00:00:00'::timestamp without time zone) AND (reserva <= '2019-03-01 00:00:00'::timestamp without time zone))
      Rows Removed by Filter: 736222
      Planning Time: 0.209 ms
      JIT:
        Functions: 26
        Options: Inlining true, Optimization true, Expressions true, Deforming true
        Timing: Generation 1.304 ms, Inlining 16.248 ms, Optimization 110.722 ms, Emission 64.677 ms, Total 192.951 ms
      Execution Time: 3820229.259 ms

```

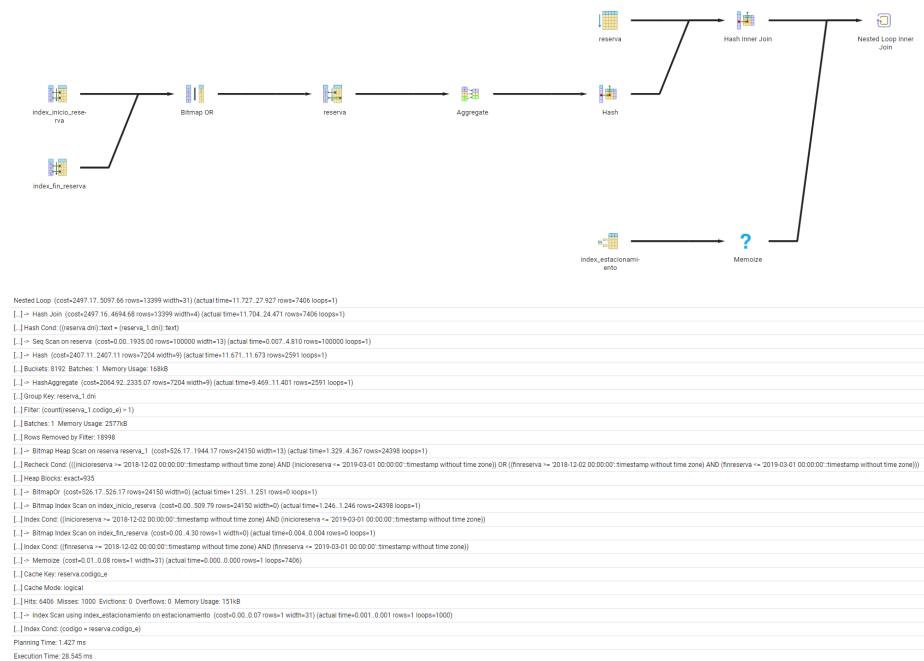
5.4.2.5. Ejecución con índices para 1k datos



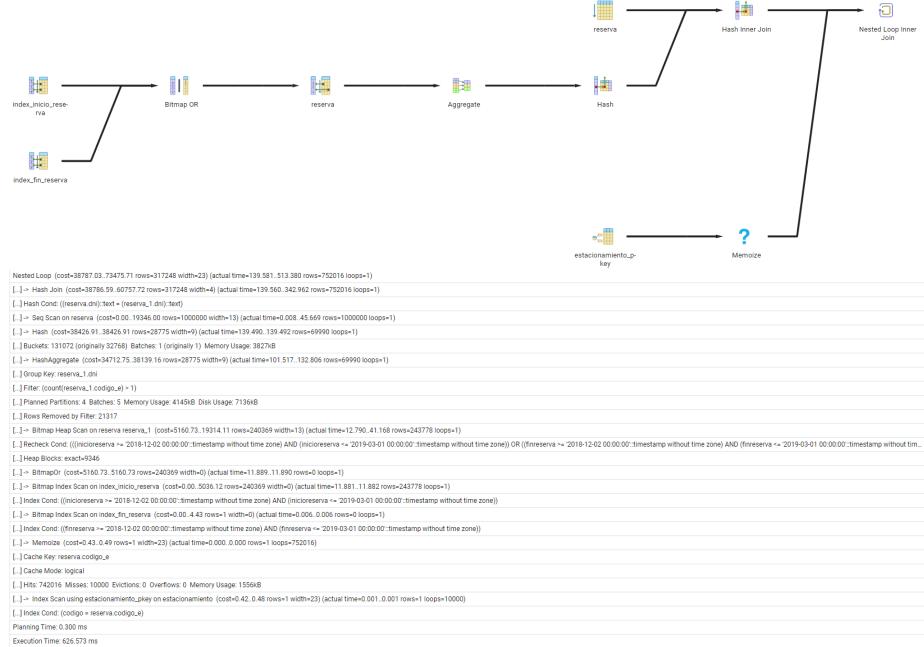
5.4.2.6. Ejecución con índices para 10k datos



5.4.2.7. Ejecución con índices para 100k datos



5.4.2.8. Ejecución con índices para 1000k datos



5.4.2.9. Explicación de consulta 2

Sin índices

Respecto a la consulta con 1k tuplas, primero se van a filtrar reservas cuyos usuarios tenían más de una reserva planeada en distintos estacionamientos, siendo obtenido a través de la función de agregación COUNT(), dentro de un intervalo de tiempo establecido. El resultado es unido a través de un Nested Loop Inner Join con reserva para obtener los códigos de los estacionamientos en donde participaron dichos usuarios. Finalmente, se une con estacionamientos mediante un Nested Loop Inner Join para obtener la información respectiva de cada uno de los estacionamientos. A partir de 10k, 100k y 1M de tuplas la principal diferencia es el uso de Memoize en estacionamientos, dicha técnica sirve para mantener valores previos encontrados en la tabla y referirse eficazmente a estos. El Nested Loop Inner Join aprovecha esta funcionalidad para obtener las reservas que coincidan con los códigos de estacionamiento. En la rama inferior se realiza el mismo proceso para el filtro de reservas. Finalmente, el resultado es unido a los estacionamientos a través de un Nested Loop Inner Join.

Con índices

En el caso de la consulta usando índices, para 1000 tuplas al igual que en el caso sin índices se utiliza la función de agregación COUNT() con el mismo propósito. Sin embargo, en este caso se hace uso de un Hash Inner Join para unir los resultados de las reservas a través del dni. Finalmente se realiza un Nested

Loop Inner Join entre el índice del estacionamiento junto a los códigos de los estacionamientos obtenidos previamente para obtener el resultado. Para las 10k, 100k y 1M de tuplas se aprecia el uso de los índices de inicio de reserva y fin de reserva para las reservas que cumplían con el intervalo de tiempo establecido. Los resultados obtenidos son unidos, al igual que en el caso de las 1000 tuplas con índices, por un Hash Inner Join. Finalmente a través de un Nested Loop Inner Join se unen con los estacionamientos a través de su código, en el cual para 1000k y 1M de tuplas se utiliza el Memoize para optimizar el loop.

5.4.3. Ejecución de consulta 3

5.4.3.1. Ejecución sin índices para 1k datos



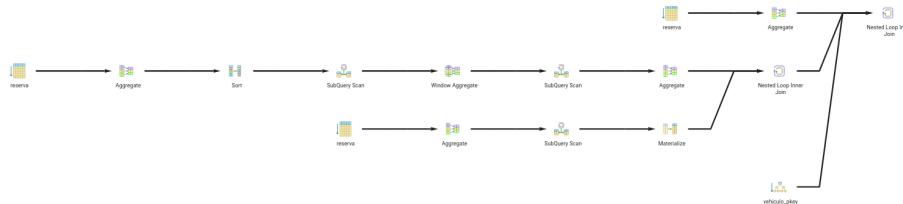
1	Nested Loop (cost=10000000180.95..1000000304.16 rows=500 width=9) (actual time=326.105..329.055 rows=9 loops=1)
2	InitPlan 1 (returns \$0)
3	-> Aggregate (cost=22.50..22.51 rows=1 width=8) (actual time=0.190..0.191 rows=1 loops=1)
4	-> Seq Scan on reserva reserva_2 (cost=0.00..20.00 rows=1000 width=7) (actual time=0.012..0.110 rows=1000 loops=1)
5	-> Nested Loop (cost=10000000158.16..10000002792.40 rows=500 width=7) (actual time=325.219..326.198 rows=9 loops=1)
6	Join Filter: (b.total_reservado = b_1.total_tiempo)
7	Rows Removed by Join Filter: 8991
8	-> HashAggregate (cost=10000000130.66..10000000132.40 rows=174 width=16) (actual time=324.454..324.463 rows=9 loops=1)
9	Group Key: b_1.total_tiempo
10	Batches: 1 Memory Usage: 40kB
11	-> Subquery Scan on b_1 (cost=10000000087.33..10000000129.83 rows=333 width=16) (actual time=323.884..324.427 rows=9 loops=1)
12	Filter: (b_1.row_number < (\$0 / 100))
13	Rows Removed by Filter: 991
14	-> WindowAgg (cost=10000000087.33..10000000114.83 rows=1000 width=24) (actual time=323.688..324.183 rows=1000 loops=1)
15	-> Subquery Scan on n (cost=10000000087.33..10000000099.83 rows=1000 width=16) (actual time=323.674..323.862 rows=1000 loops=1)
16	-> Sort (cost=10000000087.33..10000000089.83 rows=1000 width=23) (actual time=323.654..323.755 rows=1000 loops=1)
17	Sort Key: (sum((reserva.finreserva - reserva.inicioreserva))) DESC
18	Sort Method: quicksort Memory: 103kB
19	-> HashAggregate (cost=27.50..37.50 rows=1000 width=23) (actual time=323.124..323.353 rows=1000 loops=1)
20	Group Key: reserva.placa
21	Batches: 1 Memory Usage: 193kB
22	-> Seq Scan on reserva (cost=0.00..20.00 rows=1000 width=23) (actual time=0.534..1.078 rows=1000 loops=1)
23	-> Materialize (cost=27.50..52.50 rows=1000 width=23) (actual time=0.055..0.147 rows=1000 loops=9)
24	-> Subquery Scan on b (cost=27.50..47.50 rows=1000 width=23) (actual time=0.492..0.810 rows=1000 loops=1)
25	-> HashAggregate (cost=27.50..37.50 rows=1000 width=23) (actual time=0.489..0.719 rows=1000 loops=1)
26	Group Key: reserva_1.placa
27	Batches: 1 Memory Usage: 193kB
28	-> Seq Scan on reserva reserva_1 (cost=0.00..20.00 rows=1000 width=23) (actual time=0.010..0.094 rows=1000 loops=1)
29	-> Index Only Scan using vehiculo_pkey on vehiculo (cost=0.28..0.37 rows=1 width=16) (actual time=0.312..0.313 rows=1 loops=9)
30	Index Cond: (placa = (b.placa).text)
31	Heap Fetches: 9
32	Planning Time: 3.568 ms
33	JIT:
34	Functions: 36
35	Options: Inlining true, Optimization true, Expressions true, Deforming true
36	Timing: Generation 1.798 ms, Inlining 10.501 ms, Optimization 182.407 ms, Emission 128.788 ms, Total 323.495 ms
37	Execution Time: 331.023 ms

5.4.3.2. Ejecución sin índices para 10k datos



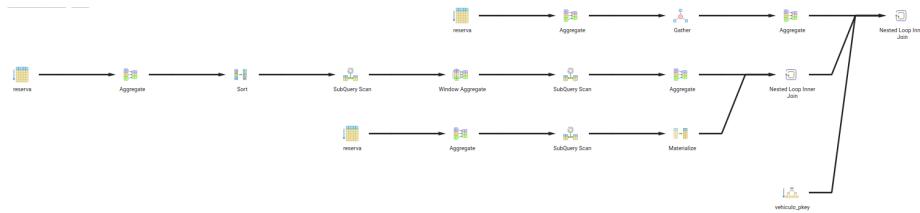
1	Nested Loop (cost=10000001955.01..10000033862.23 rows=5000 width=9) (actual time=634.533..3324.508 rows=99 loops=1)
2	InitPlan 1 (returns \$0)
3	-> Aggregate (cost=219.00..219.01 rows=1 width=8) (actual time=28.830..28.837 rows=1 loops=1)
4	-> Seq Scan on reserva reserva_2 (cost=0.00..194.00 rows=10000 width=7) (actual time=0.014..14.492 rows=10000 loops=1)
5	-> Nested Loop (cost=10000001735.72..10000031962.72 rows=5000 width=7) (actual time=633.930..3302.897 rows=99 loops=1)
6	Join Filter: (b.total_reservado = b_1.total_tiempo)
7	Rows Removed by Join Filter: 989901
8	-> HashAggregate (cost=10000001466.72..10000001468.72 rows=200 width=16) (actual time=540.311..540.794 rows=99 loops=1)
9	Group Key: b_1.total_tiempo
10	Batches: 1 Memory Usage: 40kB
11	-> Subquery Scan on b_1 (cost=10000001033.39..10000001458.39 rows=3333 width=16) (actual time=454.397..540.129 rows=99 loops=1)
12	Filter: (b_1.row_number < (\$0 / 100))
13	Rows Removed by Filter: 9901
14	-> WindowAgg (cost=10000001033.39..10000001308.39 rows=10000 width=24) (actual time=425.548..497.316 rows=10000 loops=1)
15	-> Subquery Scan on n (cost=10000001033.39..10000001158.39 rows=10000 width=16) (actual time=425.511..467.633 rows=10000 loops=1)
16	-> Sort (cost=10000001033.39..10000001058.39 rows=10000 width=23) (actual time=425.477..440.084 rows=10000 loops=1)
17	Sort Key: (sum(reserva.finreserva - reserva.inicioreserva))) DESC
18	Sort Method: quicksort Memory: 1166kB
19	-> HashAggregate (cost=269.00..369.00 rows=10000 width=23) (actual time=391.687..407.730 rows=10000 loops=1)
20	Group Key: reserva.placa
21	Batches: 1 Memory Usage: 1681kB
22	-> Seq Scan on reserva (cost=0.00..194.00 rows=10000 width=23) (actual time=0.571..17.507 rows=10000 loops=1)
23	-> Materialize (cost=269.00..519.00 rows=10000 width=23) (actual time=0.340..14.371 rows=10000 loops=99)
24	-> Subquery Scan on b (cost=269.00..469.00 rows=10000 width=23) (actual time=33.487..75.963 rows=10000 loops=1)
25	-> HashAggregate (cost=269.00..369.00 rows=10000 width=23) (actual time=33.478..48.871 rows=10000 loops=1)
26	Group Key: reserva_1.placa
27	Batches: 1 Memory Usage: 1681kB
28	-> Seq Scan on reserva reserva_1 (cost=0.00..194.00 rows=10000 width=23) (actual time=0.015..15.451 rows=10000 loops=1)
29	-> Index Only Scan using vehiculo_pkey on vehiculo (cost=0.29..0.33 rows=1 width=16) (actual time=0.201..0.204 rows=1 loops=99)
30	Index Cond: (placa = (b.placa)::text)
31	Heap Fetches: 0
32	Planning Time: 4.703 ms
33	JIT:
34	Functions: 36
35	Options: Inlining true, Optimization true, Expressions true, Deforming true
36	Timing: Generation 2.533 ms, Inlining 17.917 ms, Optimization 216.720 ms, Emission 121.215 ms, Total 358.383 ms
37	Execution Time: 3328.012 ms

5.4.3.3. Ejecución sin índices para 100k datos



1	Nested Loop (cost=10000037791.46..10000481732.91 rows=50000 width=9) (actual time=3450.561..286418.509 rows=999 loops=1)
2	InitPlan 1 (returns \$0)
3	-> Aggregate (cost=2185.00..2185.01 rows=1 width=8) (actual time=270.920..270.926 rows=1 loops=1)
4	-> Seq Scan on reserva reserva_2 (cost=0.00..1935.00 rows=100000 width=7) (actual time=0.010..135.759 rows=100000 loops=1)
5	-> Nested Loop (cost=10000035606.03..10000456229.90 rows=50000 width=7) (actual time=3450.519..285807.294 rows=999 loops=1)
6	Join Filter: (b.total_reservado = b_1.total_tiempo)
7	Rows Removed by Join Filter: 99799001
8	-> HashAggregate (cost=10000026233.53..10000026235.53 rows=200 width=16) (actual time=2305.362..2309.402 rows=998 loops=1)
9	Group Key: b_1.total_tiempo
10	Batches: 1 Memory Usage: 145kB
11	-> Subquery Scan on b_1 (cost=10000021900.20..10000026150.20 rows=33333 width=16) (actual time=1439.296..2303.718 rows=999 loops=1)
12	Filter: (b_1.row_number < (\$0 / 100))
13	Rows Removed by Filter: 99001
14	-> WindowAgg (cost=10000021900.20..10000024650.20 rows=100000 width=24) (actual time=1168.358..1892.385 rows=100000 loops=1)
15	-> Subquery Scan on n (cost=10000021900.20..10000023150.20 rows=100000 width=16) (actual time=1168.328..1590.868 rows=100000 loops=1)
16	-> Sort (cost=10000021900.20..10000022150.20 rows=100000 width=23) (actual time=1168.301..1314.779 rows=100000 loops=1)
17	Sort Key: (sum((reserva.finreserva - reserva.inicioreserva))) DESC
18	Sort Method: external merge Disk: 3240kB
19	-> HashAggregate (cost=9372.50..11544.38 rows=100000 width=23) (actual time=737.457..977.270 rows=100000 loops=1)
20	Group Key: reserva.placa
21	Planned Partitions: 8 Batches: 9 Memory Usage: 4689kB Disk Usage: 3520kB
22	-> Seq Scan on reserva (cost=0.00..1935.00 rows=100000 width=23) (actual time=0.010..147.779 rows=100000 loops=1)
23	-> Materialize (cost=9372.50..13630.38 rows=100000 width=23) (actual time=0.404..143.287 rows=100000 loops=998)
24	-> Subquery Scan on b (cost=9372.50..12544.38 rows=100000 width=23) (actual time=400.175..927.545 rows=100000 loops=1)
25	-> HashAggregate (cost=9372.50..11544.38 rows=100000 width=23) (actual time=400.163..648.508 rows=100000 loops=1)
26	Group Key: reserva_1.placa
27	Planned Partitions: 8 Batches: 9 Memory Usage: 4689kB Disk Usage: 3520kB
28	-> Seq Scan on reserva reserva_1 (cost=0.00..1935.00 rows=100000 width=23) (actual time=0.011..152.116 rows=100000 loops=1)
29	-> Index Only Scan using vehiculo_pkey on vehiculo (cost=0.42..0.46 rows=1 width=16) (actual time=0.598..0.600 rows=1 loops=999)
30	Index Cond: (placa = (b.placa)::text)
31	Heap Fetches: 0
32	Planning Time: 0.223 ms
33	JIT:
34	Functions: 44
35	Options: Inlining true, Optimization true, Expressions true, Deforming true
36	Timing: Generation 2.541 ms, Inlining 50.871 ms, Optimization 344.642 ms, Emission 188.295 ms, Total 586.348 ms
37	Execution Time: 286424.678 ms

5.4.3.4. Ejecución sin índices para 1000k datos

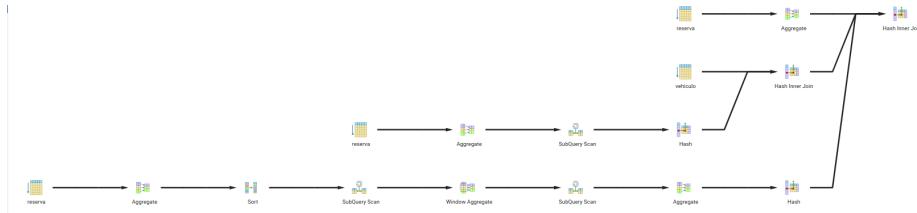


```

1 Nested Loop (cost=10000388216.91..10004830989.23 rows=500000 width=9) (actual time=26040.163..28022445.133 rows=9999 loops=1)
2 InitPlan 1 (returns $1)
3 -> Finalize Aggregate (cost=15554.55..15554.56 rows=1 width=8) (actual time=1066.816..1066.913 rows=1 loops=1)
4 -> Gather (cost=15554.33..15554.54 rows=2 width=8) (actual time=1066.605..1066.870 rows=3 loops=1)
5 Workers Planned: 2
6 Workers Launched: 2
7 -> Partial Aggregate (cost=14554.33..14554.34 rows=1 width=8) (actual time=1051.735..1051.742 rows=1 loops=3)
8 -> Parallel Seq Scan on reserva reserva_2 (cost=0.00..13512.67 rows=416667 width=7) (actual time=0.031..494.233 rows=333333 loops=3)
9 -> Nested Loop (cost=10000372661.93..10004578882.68 rows=500000 width=7) (actual time=26040.077..28018499.805 rows=9999 loops=1)
10 Join Filter: (b.total_reservado = b_.total_tiempo)
11 Rows Removed by Join Filter: 9984990001
12 -> HashAggregate (cost=10000278940.93..10000278942.93 rows=200 width=16) (actual time=16915.522..16964.944 rows=9985 loops=1)
13 Group Key: b_.total_tiempo
14 Batches: 1 Memory Usage: 929kB
15 -> Subquery Scan on b_1 (cost=10000235607.59..10000278107.59 rows=333333 width=16) (actual time=8313.310..16900.571 rows=9999 loops=1)
16 Filter: (b_1.row_number < ($1 / 100))
17 Rows Removed by Filter: 990001
18 -> WindowAgg (cost=10000235607.59..10000263107.59 rows=1000000 width=24) (actual time=7246.481..14397.632 rows=1000000 loops=1)
19 -> Subquery Scan on n (cost=10000235607.59..10000248107.59 rows=1000000 width=16) (actual time=7246.449..11529.632 rows=1000000 loops=1)
20 -> Sort (cost=10000235607.59..10000238107.59 rows=1000000 width=23) (actual time=7246.406..8691.516 rows=1000000 loops=1)
21 Sort Key: (sum((reserva.finreserva - reserva.inicioreserva))) DESC
22 Sort Method: external merge Disk: 32368kB
23 -> HashAggregate (cost=93721.00..115439.75 rows=1000000 width=23) (actual time=3512.604..5264.328 rows=1000000 loops=1)
24 Group Key: reserva.placa
25 Planned Partitions: 64 Batches: 65 Memory Usage: 4113kB Disk Usage: 61112kB
26 -> Seq Scan on reserva (cost=0.00..19346.00 rows=1000000 width=23) (actual time=0.051..1456.659 rows=1000000 loops=1)
27 -> Materialize (cost=93721.00..136299.75 rows=1000000 width=23) (actual time=0.308..1416.053 rows=1000000 loops=9985)
28 -> Subquery Scan on b (cost=93721.00..125439.75 rows=1000000 width=23) (actual time=3025.983..7551.321 rows=1000000 loops=1)
29 -> HashAggregate (cost=93721.00..115439.75 rows=1000000 width=23) (actual time=3025.976..4737.896 rows=1000000 loops=1)
30 Group Key: reserva_1.placa
31 Planned Partitions: 64 Batches: 65 Memory Usage: 4113kB Disk Usage: 61112kB
32 -> Seq Scan on reserva reserva_1 (cost=0.00..19346.00 rows=1000000 width=23) (actual time=0.051..1451.048 rows=1000000 loops=1)
33 -> Index Only Scan using vehiculo_pkey on vehiculo (cost=0.42..0.46 rows=1 width=16) (actual time=0.379..0.381 rows=1 loops=9999)
34 Index Cond: (placa = (b.placa)::text)
35 Heap Fetches: 0
36 Planning Time: 0.506 ms
37 JIT:
38 Functions: 52
39 Options: Inlining true, Optimization true, Expressions true, Deforming true
40 Timing: Generation 2.861 ms, Inlining 257.304 ms, Optimization 451.025 ms, Emission 201.239 ms, Total 912.428 ms
41 Execution Time: 28022502.076 ms

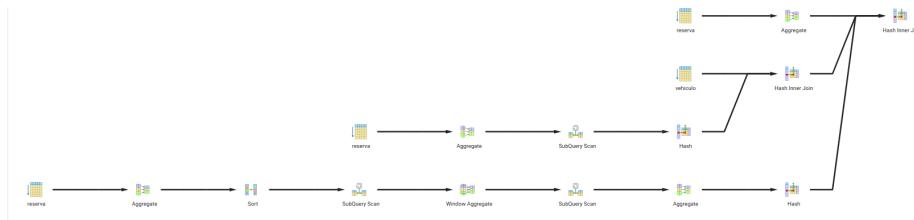
```

5.4.3.5. Ejecución con índices para 1k datos



1	Hash Join (cost=217.09..248.91 rows=500 width=9) (actual time=2.775..3.272 rows=9 loops=1)
2	Hash Cond: (b.total_reservado = b_1.total_tiempo)
3	InitPlan 1 (returns \$0)
4	-> Aggregate (cost=22.50..22.51 rows=1 width=8) (actual time=0.182..0.183 rows=1 loops=1)
5	-> Seq Scan on reserva reserva_2 (cost=0.00..20.00 rows=1000 width=7) (actual time=0.003..0.078 rows=1000 loops=1)
6	-> Hash Join (cost=60.00..83.64 rows=1000 width=25) (actual time=0.922..1.337 rows=1000 loops=1)
7	Hash Cond: ((vehiculo.placa)::text = (b.placa)::text)
8	-> Seq Scan on vehiculo (cost=0.00..21.00 rows=1000 width=16) (actual time=0.005..0.103 rows=1000 loops=1)
9	-> Hash (cost=47.50..47.50 rows=1000 width=23) (actual time=0.910..0.912 rows=1000 loops=1)
10	Buckets: 1024 Batches: 1 Memory Usage: 62kB
11	-> Subquery Scan on b (cost=27.50..47.50 rows=1000 width=23) (actual time=0.434..0.741 rows=1000 loops=1)
12	-> HashAggregate (cost=27.50..37.50 rows=1000 width=23) (actual time=0.434..0.641 rows=1000 loops=1)
13	Group Key: reserva.placa
14	Batches: 1 Memory Usage: 193kB
15	-> Seq Scan on reserva (cost=0.00..20.00 rows=1000 width=23) (actual time=0.002..0.080 rows=1000 loops=1)
16	-> Hash (cost=132.40..132.40 rows=174 width=16) (actual time=1.828..1.829 rows=9 loops=1)
17	Buckets: 1024 Batches: 1 Memory Usage: 9kB
18	-> HashAggregate (cost=130.66..132.40 rows=174 width=16) (actual time=1.823..1.826 rows=9 loops=1)
19	Group Key: b_1.total_tiempo
20	Batches: 1 Memory Usage: 40kB
21	-> Subquery Scan on b_1 (cost=87.33..129.83 rows=333 width=16) (actual time=1.137..1.819 rows=9 loops=1)
22	Filter: (b_1.row_number < (\$0 / 100))
23	Rows Removed by Filter: 991
24	-> WindowAgg (cost=87.33..114.83 rows=1000 width=24) (actual time=0.952..1.563 rows=1000 loops=1)
25	-> Subquery Scan on n (cost=87.33..99.83 rows=1000 width=16) (actual time=0.946..1.119 rows=1000 loops=1)
26	-> Sort (cost=87.33..89.83 rows=1000 width=23) (actual time=0.946..1.008 rows=1000 loops=1)
27	Sort Key: (sum((reserva_1.finreserva - reserva_1.inicioreserva))) DESC
28	Sort Method: quicksort Memory: 103kB
29	-> HashAggregate (cost=27.50..37.50 rows=1000 width=23) (actual time=0.426..0.612 rows=1000 loops=1)
30	Group Key: reserva_1.placa
31	Batches: 1 Memory Usage: 193kB
32	-> Seq Scan on reserva reserva_1 (cost=0.00..20.00 rows=1000 width=23) (actual time=0.002..0.075 rows=1000 loops=1)
33	Planning Time: 0.402 ms
34	Execution Time: 3.353 ms

5.4.3.6. Ejecución con índices para 10k datos



```

1 Hash Join (cost=2284.23..2595.36 rows=5000 width=9) (actual time=23.367..27.647 rows=99 loops=1)
2 Hash Cond: (b.total_reservado = b_1.total_tiempo)
3 InitPlan 1 (returns $0)
4 -> Aggregate (cost=219.00..219.01 rows=1 width=8) (actual time=1.642..1.643 rows=1 loops=1)
5 -> Seq Scan on reserva reserva_2 (cost=0.00..194.00 rows=10000 width=7) (actual time=0.010..0.722 rows=10000 loops=1)
6 -> Hash Join (cost=594.00..823.26 rows=10000 width=25) (actual time=8.001..11.629 rows=10000 loops=1)
7 Hash Cond: ((vehiculo.placa)::text = (b.placa)::text)
8 -> Seq Scan on vehiculo (cost=0.00..203.00 rows=10000 width=16) (actual time=0.004..0.608 rows=10000 loops=1)
9 -> Hash (cost=469.00..469.00 rows=10000 width=23) (actual time=7.981..7.984 rows=10000 loops=1)
10 Buckets: 16384 Batches: 1 Memory Usage: 666kB
11 -> Subquery Scan on b (cost=269.00..469.00 rows=10000 width=23) (actual time=3.327..6.288 rows=10000 loops=1)
12 -> HashAggregate (cost=269.00..369.00 rows=10000 width=23) (actual time=3.325..5.540 rows=10000 loops=1)
13 Group Key: reserva.placa
14 Batches: 1 Memory Usage: 1681kB
15 -> Seq Scan on reserva (cost=0.00..194.00 rows=10000 width=23) (actual time=0.001..0.511 rows=10000 loops=1)
16 -> Hash (cost=1468.72..1468.72 rows=200 width=16) (actual time=15.324..15.327 rows=99 loops=1)
17 Buckets: 1024 Batches: 1 Memory Usage: 13kB
18 -> HashAggregate (cost=1466.72..1468.72 rows=200 width=16) (actual time=15.274..15.291 rows=99 loops=1)
19 Group Key: b_1.total_tiempo
20 Batches: 1 Memory Usage: 40kB
21 -> Subquery Scan on b_1 (cost=1033.39..1458.39 rows=3333 width=16) (actual time=10.298..15.253 rows=99 loops=1)
22 Filter: (b_1.row_number < ($0 / 100))
23 Rows Removed by Filter: 9901
24 -> WindowAgg (cost=1033.39..1308.39 rows=10000 width=24) (actual time=8.650..13.142 rows=10000 loops=1)
25 -> Subquery Scan on n (cost=1033.39..1158.39 rows=10000 width=16) (actual time=8.640..10.523 rows=10000 loops=1)
26 -> Sort (cost=1033.39..1058.39 rows=10000 width=23) (actual time=8.638..9.771 rows=10000 loops=1)
27 Sort Key: (sum(reserva_1.finreserva - reserva_1.inicioreserva)) DESC
28 Sort Method: quicksort Memory: 1166kB
29 -> HashAggregate (cost=269.00..369.00 rows=10000 width=23) (actual time=3.503..5.478 rows=10000 loops=1)
30 Group Key: reserva_1.placa
31 Batches: 1 Memory Usage: 1681kB
32 -> Seq Scan on reserva reserva_1 (cost=0.00..194.00 rows=10000 width=23) (actual time=0.007..0.517 rows=10000 loops=1)
33 Planning Time: 0.304 ms
34 Execution Time: 27.949 ms

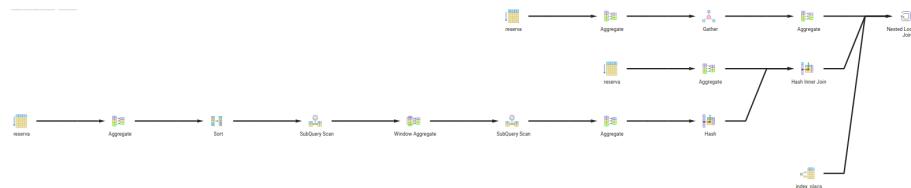
```

5.4.3.7. Ejecución con índices para 100k datos



1	Nested Loop (cost=37795.54..46239.16 rows=50000 width=9) (actual time=244.357..296.751 rows=999 loops=1)
2	InitPlan 1 (returns \$0)
3	-> Aggregate (cost=2185.00..2185.01 rows=1 width=8) (actual time=11.833..11.834 rows=1 loops=1)
4	-> Seq Scan on reserva reserva_2 (cost=0.00..1935.00 rows=100000 width=7) (actual time=0.009..4.735 rows=100000 loops=1)
5	-> Hash Join (cost=35610.53..39601.15 rows=50000 width=7) (actual time=244.329..293.989 rows=999 loops=1)
6	Hash Cond: ((sum((reserva.finreserva - reserva.inicioreserva))) = b.total_tiempo)
7	-> HashAggregate (cost=9372.50..11544.38 rows=100000 width=23) (actual time=42.621..84.672 rows=100000 loops=1)
8	Group Key: reserva.placa
9	Planned Partitions: 8 Batches: 9 Memory Usage: 4689kB Disk Usage: 3520kB
10	-> Seq Scan on reserva (cost=0.00..1935.00 rows=100000 width=23) (actual time=0.007..5.430 rows=100000 loops=1)
11	-> Hash (cost=26235.53..26235.53 rows=200 width=16) (actual time=201.692..201.696 rows=998 loops=1)
12	Buckets: 1024 Batches: 1 Memory Usage: 55kB
13	-> HashAggregate (cost=26233.53..26235.53 rows=200 width=16) (actual time=201.446..201.579 rows=998 loops=1)
14	Group Key: b.total_tiempo
15	Batches: 1 Memory Usage: 145kB
16	-> Subquery Scan on b (cost=21900.20..26150.20 rows=33333 width=16) (actual time=146.581..201.256 rows=999 loops=1)
17	Filter: (b.row_number < (\$0 / 100))
18	Rows Removed by Filter: 99001
19	-> WindowAgg (cost=21900.20..24650.20 rows=100000 width=24) (actual time=134.742..183.907 rows=100000 loops=1)
20	-> Subquery Scan on n (cost=21900.20..23150.20 rows=100000 width=16) (actual time=134.729..154.251 rows=100000 loops=1)
21	-> Sort (cost=21900.20..22150.20 rows=100000 width=23) (actual time=134.727..146.339 rows=100000 loops=1)
22	Sort Key: (sum((reserva_1.finreserva - reserva_1.inicioreserva))) DESC
23	Sort Method: external merge Disk: 3240kB
24	-> HashAggregate (cost=9372.50..11544.38 rows=100000 width=23) (actual time=40.296..81.098 rows=100000 loops=1)
25	Group Key: reserva_1.placa
26	Planned Partitions: 8 Batches: 9 Memory Usage: 4689kB Disk Usage: 3520kB
27	-> Seq Scan on reserva reserva_1 (cost=0.00..1935.00 rows=100000 width=23) (actual time=0.004..4.868 rows=100000 loops=1)
28	-> Index Scan using index_placa on vehiculo (cost=0.00..0.08 rows=1 width=16) (actual time=0.002..0.002 rows=1 loops=999)
29	Index Cond: ((placa)::text = (reserva.placa)::text)
30	Planning Time: 0.373 ms
31	Execution Time: 299.400 ms

5.4.3.8. Ejecución con índices para 1M datos



```

1 Nested Loop (cost=388220.98..470559.23 rows=500000 width=9) (actual time=2715.007..3267.543 rows=9999 loops=1)
2   InitPlan 1 (returns $1)
3     -> Finalize Aggregate (cost=15554.55..15554.56 rows=1 width=8) (actual time=64.884..64.951 rows=1 loops=1)
4     -> Gather (cost=15554.33..15554.54 rows=2 width=8) (actual time=64.793..64.941 rows=3 loops=1)
5   Workers Planned: 2
6   Workers Launched: 2
7     -> Partial Aggregate (cost=14554.33..14554.34 rows=1 width=8) (actual time=51.605..51.606 rows=1 loops=3)
8     -> Parallel Seq Scan on reserva reserva_2 (cost=0.00..13512.67 rows=416667 width=7) (actual time=0.030..22.028 rows=333333 loops=3)
9     -> Hash Join (cost=372666.43..412572.68 rows=500000 width=7) (actual time=2714.952..3219.487 rows=9999 loops=1)
10    Hash Cond: ((sum((reserva.finreserva - reserva.inicioreserva))) = b.total_tiempo)
11    -> HashAggregate (cost=93721.00..115439.75 rows=1000000 width=23) (actual time=637.428..1033.287 rows=1000000 loops=1)
12    Group Key: reserva.placa
13  Planned Partitions: 64 Batches: 65 Memory Usage: 4113kB Disk Usage: 61112kB
14    -> Seq Scan on reserva (cost=0.00..19346.00 rows=1000000 width=23) (actual time=0.035..95.639 rows=1000000 loops=1)
15    -> Hash (cost=278942.93..278942.93 rows=200 width=16) (actual time=2077.463..2077.466 rows=9985 loops=1)
16    Buckets: 16384 (originally 1024) Batches: 1 (originally 1) Memory Usage: 597kB
17    -> HashAggregate (cost=278940.93..278942.93 rows=200 width=16) (actual time=2075.474..2076.513 rows=9985 loops=1)
18    Group Key: b.total_tiempo
19    Batches: 1 Memory Usage: 929kB
20    -> Subquery Scan on b (cost=235607.59..278107.59 rows=333333 width=16) (actual time=1546.731..2073.592 rows=9999 loops=1)
21    Filter: (b.row_number < ($1 / 100))
22  Rows Removed by Filter: 990001
23  -> WindowAgg (cost=235607.59..263107.59 rows=1000000 width=24) (actual time=1481.840..1968.321 rows=1000000 loops=1)
24  -> Subquery Scan on n (cost=235607.59..248107.59 rows=1000000 width=16) (actual time=1481.827..1710.830 rows=1000000 loops=1)
25  -> Sort (cost=235607.59..238107.59 rows=1000000 width=23) (actual time=1481.809..1641.535 rows=1000000 loops=1)
26  Sort Key: (sum((reserva_1.finreserva - reserva_1.inicioreserva))) DESC
27  Sort Method: external merge Disk: 32368kB
28  -> HashAggregate (cost=93721.00..115439.75 rows=1000000 width=23) (actual time=492.074..917.669 rows=1000000 loops=1)
29  Group Key: reserva_1.placa
30  Planned Partitions: 64 Batches: 65 Memory Usage: 4113kB Disk Usage: 61112kB
31  -> Seq Scan on reserva reserva_1 (cost=0.00..19346.00 rows=1000000 width=23) (actual time=0.013..76.518 rows=1000000 loops=1)
32  -> Index Scan using index_placa on vehiculo (cost=0.00..0.07 rows=1 width=16) (actual time=0.004..0.004 rows=1 loops=9999)
33  Index Cond: ((placa).text = (reserva.placa).text)
34  Rows Removed by Index Recheck: 0
35  Planning Time: 0.428 ms
36  JIT:
37  Functions: 55
38  Options: Inlining false, Optimization false, Expressions true, Deforming true
39  Timing: Generation 3.632 ms, Inlining 0.000 ms, Optimization 2.062 ms, Emission 44.054 ms, Total 49.748 ms
40  Execution Time: 3288.989 ms

```

5.4.3.9. Explicación de consulta 3

Sin índices

En las mediciones de las consultas correspondientes a 1k, 10k y 100k de tuplas presentan un mismo plan de ejecución. Este consiste en agrupar por placa las reservas y realizar la función de agregación SUM() sobre la diferencia entre la hora final e inicial. En otra rama, se calcula la cantidad de horas total mediante la función SUM() y se ordena para obtener el 1 %, donde se calcula la cantidad de autos que reservaron mediante COUNT(). A estas dos ramas explicadas se les realiza un Nested Loop Inner Join previo a Nested Loop Inner Join final que junta el resultado anterior con la tabla vehículo mediante el índice de llave primaria de vehículo. El resultado final son 1 % de las tuplas con mayor tiempo gastado en el servicio. Para 1M de tuplas es similar a lo explicado anteriormente solo que usando el método Gather, este método paraleliza algún cálculo en la tabla reserva entre dos funciones de agregación.

Con índices

En esta consulta se ha creado el índice index_placa. El plan para las consultas de 1k y 10k de tuplas son iguales. Aquí se realizó un Hash Inner Join en primer lugar entre la tabla vehiculo y reservas agrupado por placas, para luego realizar otro Has Inner Join entre el resultado obtenido anteriormente, la cantidad de tuplas en reserva y lo obtenido con la agrupación de la tabla reserva y usar la función de agregación SUM() para obtener la cantidad de horas por placa. Para las consultas de 100k de tuplas es muy similar al proceso anterior, pero en el segundo Hash Join se reemplaza por un Nested Loop Inner Join y se usa el index_placa al final para remplazar las operaciones con la tabla vehiculo. Finalmente, para la consulta con 1M de tuplas solo se añade el método Gather para paralelizar la operación de obtener y calcular la cantidad de horas por placa.

5.5. Medición de tiempos

Para la medición de los tiempos se realizó 5 pruebas independientes para cada consulta. Cada una de ellas está incluida en un esquema con índices y en uno sin índices. Se ha utilizado el mismo método para todos explicado en la sección 5.2. Al termino de realizar todos las pruebas, se calculó su promedio y desviación estándar para graficarlo. La unidades de tiempo que se usó es milisegundos.

5.5.1. Sin Índices

▪ Consulta 1

Ejecución número:	1k	10k	100k	1000k
1	4.257	279.249	312.456	3031.471
2	4.152	272.862	319.154	3016.079
3	4.156	280.056	325.282	3047.466
4	4.108	282.972	319.726	3100.123
5	4.097	275.964	331.695	3197.227
Promedio	4.154	278.221	321.663	3076.473
Desviación Estándar	0.063	3.899	7.221	75.258

Cuadro 25: Consulta 1 sin índices

▪ Consulta 2

Ejecución número:	1k	10k	100k	1000k
1	0.283	23.719	16117.216	2798494.132
2	0.397	24.407	16514.358	3820229.259
3	0.347	23.627	16377.306	2798494.132
4	0.326	23.879	16430.775	3820229.259
5	0.309	24.843	16435.898	2798494.132
Promedio	0.332	24.095	16375.11	3207188
Desviación Estándar	0.043	0.516	152.232	559627.4

Cuadro 26: Consulta 2 sin índices

■ Consulta 3

Ejecución número:	1k	10k	100k	1000k
1	2.911	106.228	11413.271	47863
2	3.075	103.985	11486.332	48187
3	3.016	103.639	11777.645	47757
4	2.984	106.634	11752.833	48149
5	2.968	104.908	11686.482	47873
Promedio	2.991	105.079	11623.31	47965.6
Desviación Estándar	0.061	1.326	163.91	190.845

Cuadro 27: Consulta 3 sin índices

5.5.2. Con Índices

■ Consulta 1

Ejecución número:	1k	10k	100k	1000k
1	3.799	6.410	43.258	350.917
2	3.777	6.263	43.118	353.311
3	3.738	5.918	43.749	354.586
4	4.067	5.699	43.290	362.368
5	3.840	5.835	43.419	359.085
Promedio	3.844	6.025	43.367	356.053
Desviación Estándar	0.13	0.299	0.239	4.613

Cuadro 28: Consulta 1 con índices

■ Consulta 2

Ejecución número:	1k	10k	100k	1000k
1	0.214	2.419	27.010	648.146
2	0.177	2.330	27.759	652.860
3	0.229	2.227	26.973	612.735
4	0.211	2.236	27.225	615.780
5	0.197	2.365	28.545	626.573
Promedio	0.206	2.315	27.502	631.219
Desviación Estándar	0.02	0.083	0.662	18.415

Cuadro 29: Consulta 2 con índices

■ Consulta 3

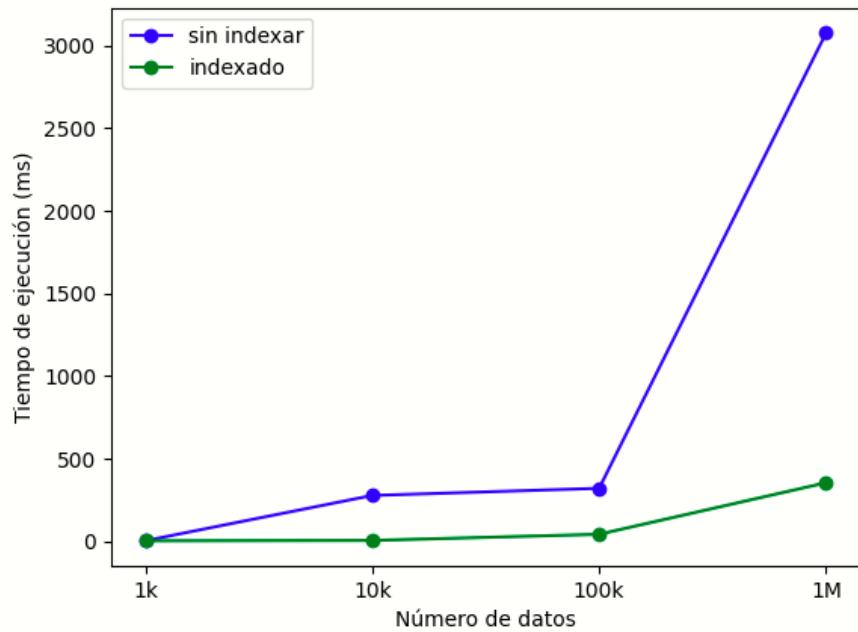
Ejecución número:	1k	10k	100k	1000k
1	2.342	24.153	364.458	4230.660
2	2.322	23.975	289.341	4152.593
3	2.127	23.894	303.527	4203.020
4	2.243	24.234	339.890	3945.725
5	2.248	24.698	339.655	4206.585
Promedio	2.256	24.191	327.374	4147.717
Desviación Estándar	0.085	0.314	30.405	116.434

Cuadro 30: Consulta 3 con índices

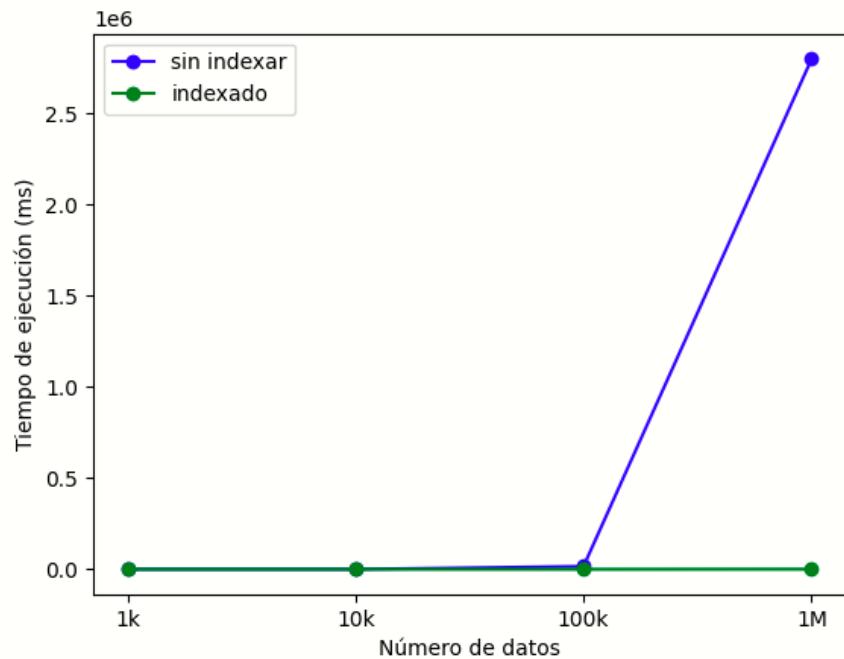
5.6. Resultados y análisis

A continuación, se visualizará por medio de gráficos el tiempo de ejecución de cada consulta vs la cantidad de datos en las tablas. Se ha tomado el promedio de tiempo para cada caso a partir de las tablas anteriores tanto para las consultas con índices y sin índices.

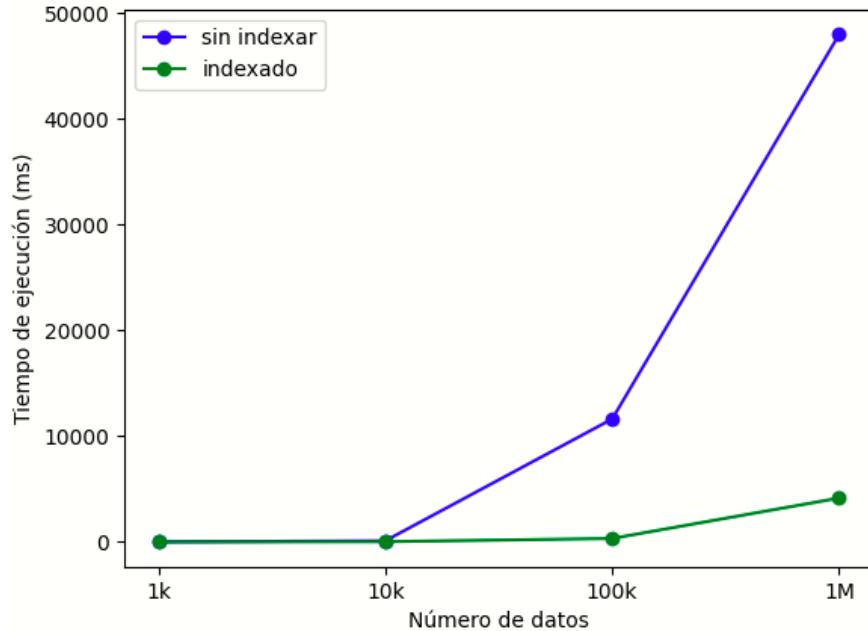
5.6.1. Gráfica Consulta 1



5.6.2. Gráfica Consulta 2



5.6.3. Gráfica Consulta 3



6. Stored Procedures

En cuanto a Stored Procedures, se ha incorporado una función muy útil para aquellos usuarios que deseen conocer rápidamente el estacionamiento más cercano con aparcamientos libres dentro de un intervalo de fechas. Dado las coordenadas actuales del usuario y las fechas especificadas de inicio y fin de las reservas se obtiene esta información. Esta función puede ser modificada también para ayudar también al usuario a encontrar estos estacionamientos en tiempo real.

```
1 CREATE OR REPLACE FUNCTION estacionamiento_con_aparcamientos_libres
2 (fechaInicio TIMESTAMP, fechaFin TIMESTAMP, x DOUBLE PRECISION, y
3  DOUBLE PRECISION)
4 RETURNS TABLE (
5   codigo INTEGER,
6   nombre VARCHAR(200),
7   latitud DOUBLE PRECISION,
8   longitud DOUBLE PRECISION,
9   distancia DOUBLE PRECISION
10 )
11 AS $$
```

12 BEGIN

```
13 RETURN QUERY
14 SELECT E.codigo, E.nombre, E.latitud, E.longitud, sqrt((E.latitud-x
15)*(E.latitud-x) + (E.longitud-y)*(E.longitud-y)) as distancia
```

```

14 FROM estacionamiento E
15 WHERE E.codigo IN (
16     SELECT codigo_e
17     FROM (
18         SELECT codigo_e, COUNT(lugares_libres.codigo) as
19             cantidad_liber
20             FROM (
21                 SELECT A.codigo, A.codigo_e
22                 FROM aparcamiento A
23                 EXCEPT
24                 SELECT R.codigo_a, R.codigo_e
25                 FROM reserva R
26                 WHERE inicioreserva >= fechaInicio AND inicioreserva <=
27                     fechaFin
28                     OR finreserva >= fechaInicio AND finreserva <= fechaFin
29                     ) lugares_liber
30                     GROUP BY codigo_e
31                     ) mas_liber
32                     WHERE cantidad_liber = (
33                         SELECT MAX(cantidad_liber)
34                         FROM (
35                             SELECT codigo_e, COUNT(lugares_libres.codigo) as
36             cantidad_liber
37             FROM (
38                 SELECT B.codigo, B.codigo_e
39                 FROM aparcamiento B
40                 EXCEPT
41                 SELECT R.codigo_a, R.codigo_e
42                 FROM reserva R
43                 WHERE inicioreserva >= fechaInicio AND inicioreserva <=
44                     fechaFin
45                     OR finreserva >= fechaInicio AND finreserva <= fechaFin
46                     ) lugares_liber
47                     GROUP BY codigo_e
48                     ) libres
49             )
50         )
51     AND sqrt((E.latitud-x)*(E.latitud-x) + (E.longitud-y)*(E.longitud-y))
52     ) = (
53     SELECT MIN(dist.distancia)
54     FROM (
55         SELECT sqrt((E1.latitud-x)*(E1.latitud-x) + (E1.longitud-y)
56             *(E1.longitud-y)) as distancia
57             FROM estacionamiento E1
58             WHERE E1.codigo IN (
59                 SELECT codigo_e
60                 FROM (
61                     SELECT codigo_e, COUNT(lugares_libres.codigo) as
62             cantidad_liber
63             FROM (
64                 SELECT C.codigo, C.codigo_e
65                 FROM aparcamiento C
66                 EXCEPT
67                 SELECT R.codigo_a, R.codigo_e
68                 FROM reserva R
69                 WHERE inicioreserva >= fechaInicio AND
70                     inicioreserva <= fechaFin

```

```

63          OR finreserva >= fechaInicio AND finreserva <=
fechaFin
64              ) lugares_libres
65                  GROUP BY codigo_e
66          ) mas_libres
67              WHERE cantidad_libres = (
68                  SELECT MAX(cantidad_libres)
69                  FROM (
70                      SELECT codigo_e, COUNT(lugares_libres.codigo)
as cantidad_libres
71                      FROM (
72                          SELECT D.codigo, D.codigo_e
73                          FROM aparcamiento D
74                          EXCEPT
75                          SELECT R.codigo_a, R.codigo_e
76                          FROM reserva R
77                          WHERE inicioreserva >= fechaInicio AND
inicioreserva <= fechaFin
78                                  OR finreserva >= fechaInicio AND finreserva
79                                  <= fechaFin
80                                      ) lugares_libres
81                                          GROUP BY codigo_e
82          ) libres
83      )
84  ) dist
85 );
86 END;
87 $$

88 LANGUAGE plpgsql;

```

7. Conclusiones

1. El modelo E/R diseñado cumple con las necesidades de las organizaciones que planean usarlo para el manejo de sus estacionamientos y de los usuarios. Además, en la estructura de la base de datos se usó vistas y *store procedure* para optimizar algunas consultas y tener un código más limpio y entendible.
 2. El modelo relacional y la distribución de tablas cumplen con la forma normal de Boyce-Codd, para asegurar una integridad en los datos y escalabilidad de la base de datos.
 3. Las consultas propuestas tienen una dificultad considerable por lo que en los gráficos se puede ver un crecimiento exponencial en tiempo con respecto a la cantidad de datos. Además, las consultas de gran cantidad de datos se ven favorecidas por procesos paralelos, por lo que podemos decir que el hardware es un factor clave con respecto a la ejecución de la consulta.
 4. Realizar un análisis en la planificación resulta muy útil cuando se está manejando gran cantidad de datos, y para ellos se debe realizar un gran

trabajo en la selección de índices para tener un modelo óptimo.

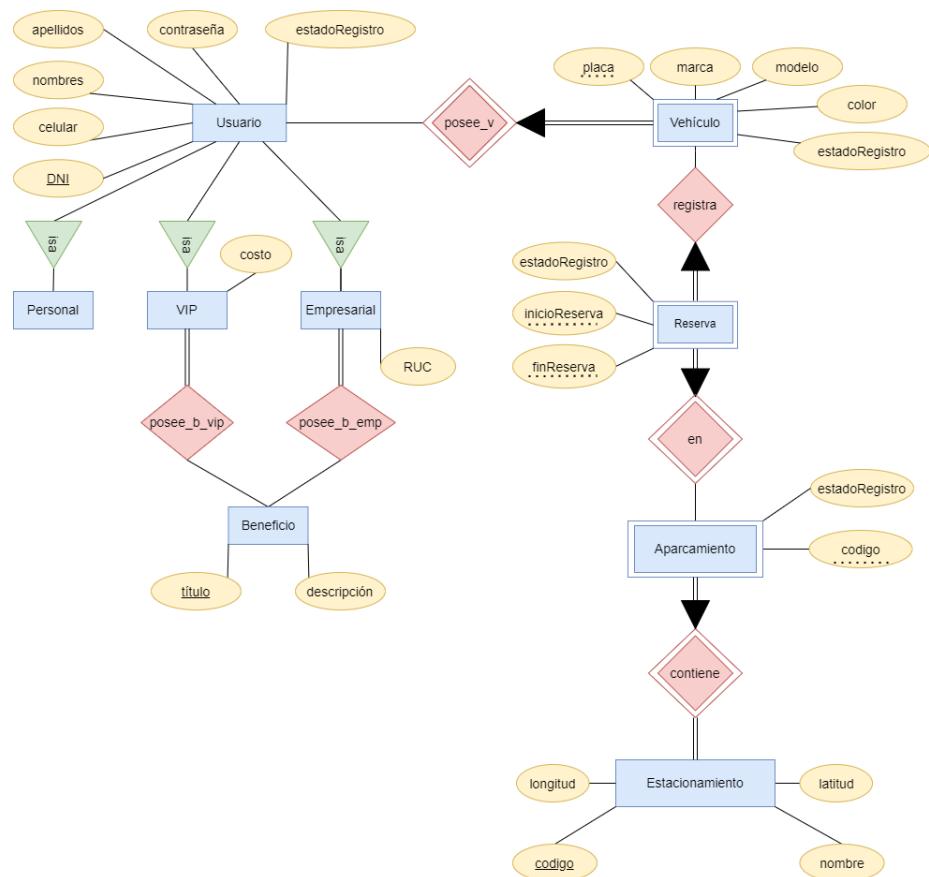
- Generalmente, los índices que se crean por defecto no son suficientes. Por ello, es necesario intervenir y crear los índices adecuados con respecto a un previo análisis de las tablas y atributos que se usan en una consulta, todo esto para lograr una mejora en la rapidez de la consulta.

8. Anexos

8.1. Dumps de tablas, archivos CSV externos y videos de ejecución de consultas

https://drive.google.com/drive/folders/18con-WvCPYJnKxaNyTuMcQkZEPZ6rcC_

8.2. Diagrama del modelo E-R



8.3. Diagrama del modelo relacional

