Base de datos: Justificación

Integrantes:

Jaime Cuartas Granada

Emily Esmeralda Carvajal Camelo

TABLA DE CONTENIDO

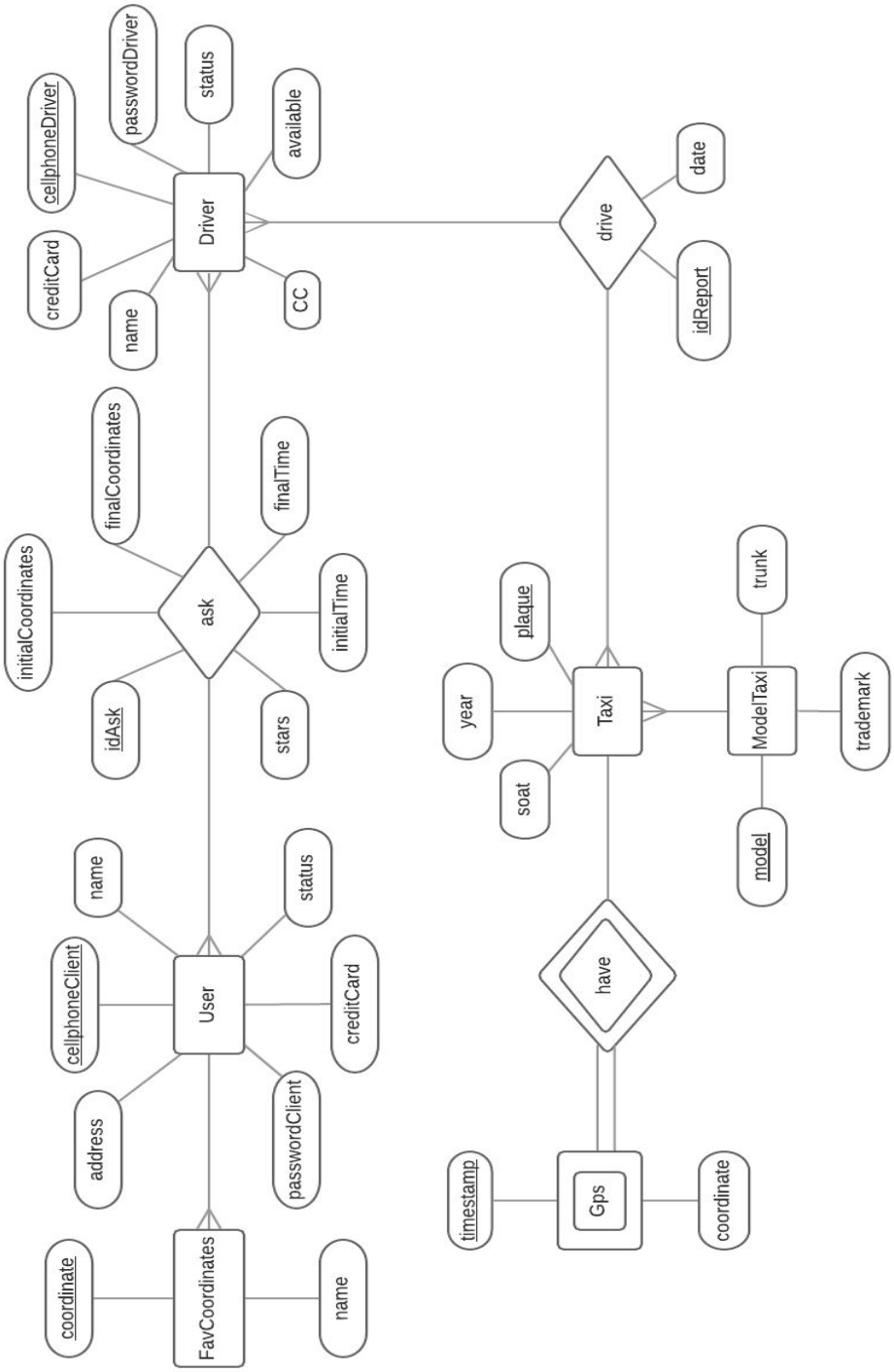
1. Objetivos

Se entiende la creación de una base de datos como un trabajo arduo para su correcta ejecución, en el que se crea un esquema con una colección de datos que contiene información relevante para un proyecto, en este caso en particular para el software “Not so easy Taxi”.

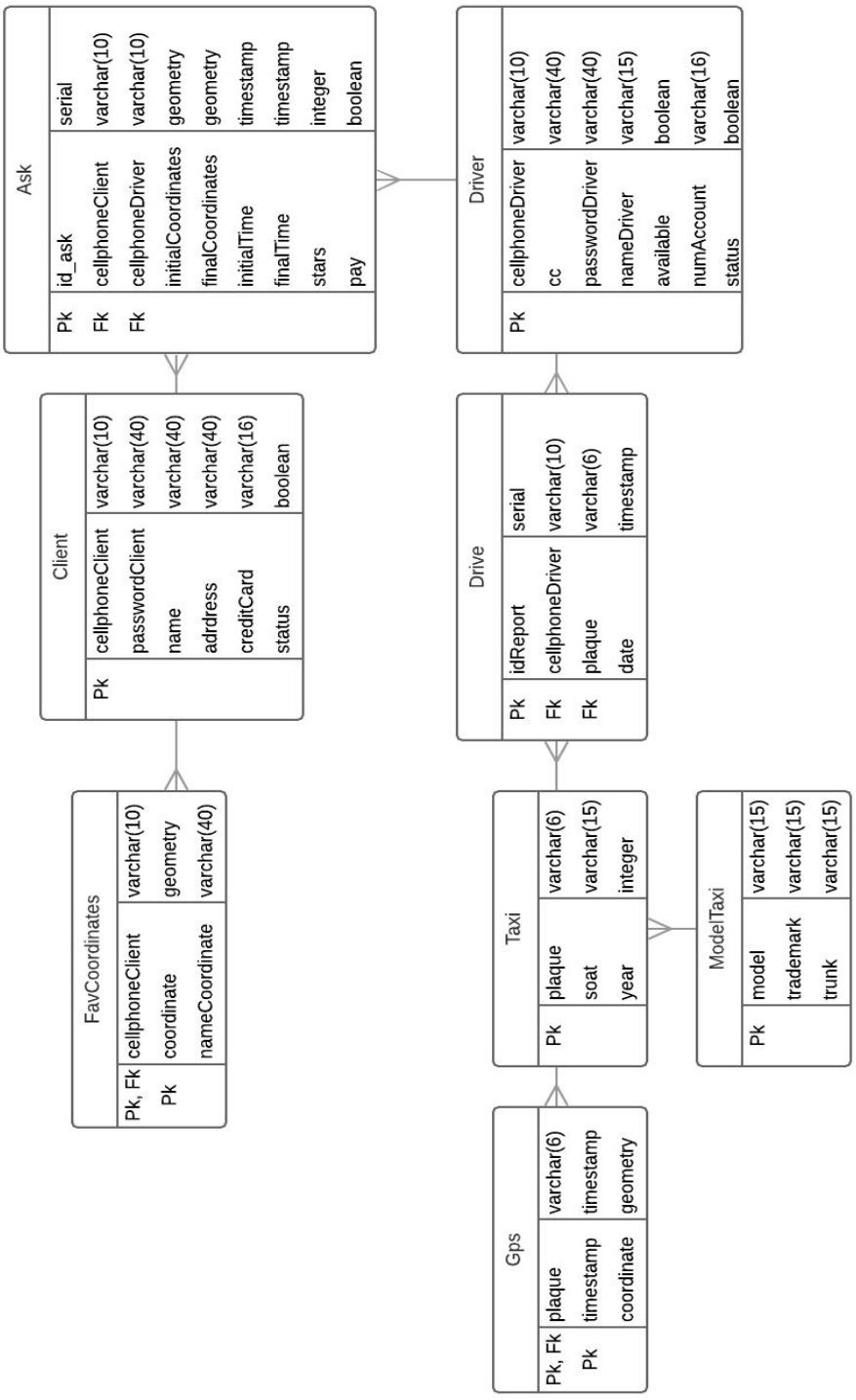
El software “Not so easy taxi” debe proporcionar una forma de almacenar y recuperar información de manera practica y eficiente, además de la fiabilidad de esta información. Por estas razones la necesidad de un sistema gestor de bases de datos.

En este trabajo se justifican las acciones y decisiones tomadas para el correcto almacenamiento de los datos utilizando PostgreSQL usado por el software creado por Emily Esmeralda Carvajal y Jaime Cuartas como proyecto final de Bases de datos 1, Universidad del Valle.

1. Diagrama Entidad-Relación (Link)



2. Diagrama Relacional (Link)



Links de los diagramas:

3. Normalización de la base de datos

Criterios para determinar si una relación se encuentra en forma normal de Boyce-Codd

Un esquema de relación R está en FNBC respecto a un conjunto de dependencias funcionales F si,para todas las dependencias funcionales de F^+ de la forma $\alpha \rightarrow \beta$, donde $\alpha \subseteq R$ y $\beta \subseteq R$, se cumple al menos una de las siguientes condiciones:

- $\alpha \rightarrow \beta$ es una dependencia funcional trivial (es decir, $\beta \subseteq \alpha$)
- α es una superclave del esquema R.

Teniendo las siguientes relaciones se mostrarán las dependencias funcionales no triviales ($\alpha \rightarrow \alpha$) de cada Esquema en el que se basa la base de datos implementada en Postgresql:

- En Esquema-FavCoordinates:

Siendo los atributos cellphoneClient, coordinate y nameCoordinate los atributos de la relación. cellphoneClient y corrdinate son la clave primaria pues un cellphoneClient puede relacionarse con más de una coordinate, y un corrdinate puede relacionarse en varios cellphoneClient, entonces solo se identifica el atributo nameCoordinate que es el único no primo con los atributos cellphoneClient y coordinate de manera completa y sin transitividad.

  - cellphoneClient coordinate → cellphoneClient coordinate nameCoordinate

  (cellphoneClient coordinate)$^+$ = cellphoneClient coordinate nameCoordinate

  cellphoneClient y coordinate son clave primaria

  Esquema-favCoordinates esta en forma normal de Boyce-Codd

- En Esquema-Client:

Es una abstracción del usuario que solicita los servicios. Los atributos de la relación son cellphoneClient, passwordClient, nameClient, address, creditCard y status, y es el atributo cellphoneClient el único que identifica de manera inequívoca al resto de los demás atributos. Entonces cada atributo depende funcionalmente de cellphoneClient, por ser una clave primaria no compuesta tambien dependen evidentemente de manera completa de dicha clave y en la relación se alcanzan todos los atributos sin ninguna transitividad se concluye lo siguiente.

- cellphoneClient → cellphoneClient passwordClient nameClient address creditCard status

(cellphoneClient)$^+$ = cellphoneClient passwordClient nameClient address creditCard status

cellphoneClient es clave primaria

Esquema-FavCoordinates esta en forma normal de Boyce-Codd

- En Esquema-Ask:

Es la abstracción de las solicitudes de viaje realizadas, donde se guarda cada solicitud con una identificación generada de manera automática, donde se guarda información relevante del viaje, la clave primaria del Esquema-Client y la clave primaria del Esquema-Driver. Los atributos de la relación son idAsk, cellPhoneClient, cellphoneDriver, initialCoordinates, finalCoordinates, initialTime, finalTime y stars, donde ningún atributo fuera de idAsk depende funcionalmente de otro, esto debido a que es posible que un usuario con un determinado cellPhoneClient pueda realizar múltiples viajes iguales y que deben estar registrados en la base de datos, cada conductor con un determinado cellPhoneDriver puede realizar de la misma manera múltiples viajes y pueden relacionarse estos mismos en viajes distintos. Por estas razones el atributo auto-generado idAsk existe e identifica cada viaje solicitado de manera inequívoca.

- IdAsk → idAsk cellPhoneClient cellphoneDriver initialCoordinates finalCoordinates initialTime finalTime stars

(IdAsk)$^+$ = idAsk cellPhoneClient cellphoneDriver initialCoordinates finalCoordinates initialTime finalTime stars

idAsk es la clave primaria

Esquema-Ask esta en forma normal de Boyce-Codd

- En Esquema-Driver:

Es una abstracción del usuario que toma los servicios, quien conduce el taxi para llevar a su destino al usuario Cliente. Los atributos de la relación son cellphoneDriver, cc, passwordDriver, nameDriver, available, numAccount y status. El atributo cellphnoeDriver y cc identifican de manera inequívoca al resto de los demás atributos en la relación, entonces se tienen dos claves candidatas para esta relación y es posible la elección de cualquiera de las dos. Para esta implementación se eligió cellphone como clave primaria y cc como atributo no primo unique. Como ninguna de las dos claves candidatas es compuesta, los atributos dependen completamente de la clave y en la relación se alcanzan todos los atributos sin ninguna transitividad se concluye lo siguiente.

- cellphoneDriver → cellphoneDriver cc passwordDriver nameDriver available numAccount status
- cc → cellphoneDriver cc passwordDriver nameDriver available numAccount status

(cellphoneClient)$^+$ = cellphoneDriver cc passwordDriver nameDriver available numAccount status

cellphoneClient es clave primaria

Esquema-FavCoordinates esta en forma normal de Boyce-Codd

- En Esquema-Drive:

En esta relación se busca guardar todos los taxis que un conductor ha manejado, pues un conductor podría tener un taxi y cambiarlo por uno diferente para la empresa, pero desear ver cuales han sido sus carros desde que trabaja en el lugar. La relación tiene los atributos idReport, cellphoneDriver, plaque y date, donde se guarda y asocia la clave primaria de cada conductor con la placa del vehículo, junto con el momento en que se hizo este cambio de taxi y un id autogenerado donde se identifican todos los cambios de cada conductor, debido a que un conductor podría cambiar su carro por otro y luego cambiarlo nuevamente por el anterior, asi que no es posible identificar un registro en particular sin un atributo como idReport que tenga identificaciones únicas para cada cambio. Entonces, cada atributo depende funcionalmente de idReport de manera completa y sin transitividad.

- idReport → idReport cellphoneDriver plaque date

(idReport)$^+$ = idReport cellphoneDriver plaque date

idReport es clave primaria

Esquema- Drive está en forma normal de Boyce-Codd

- En Esquema-Taxi:

Es una abstracción del Taxi. Los atributos de la relación son model, year y plaque, donde plaque y soat identifican de manera inequívoca al resto de los atributos, entonces se tienen dos claves candidatas para esta relación y es posible la elección de cualquiera de las dos. Para esta implementación se eligió plaque como clave primaria y soat como atributo no primo unique. Como ninguna de las dos claves candidatas es compuesta (se cumple condición de Fnbc), los atributos dependen completamente de la clave y en la relación se alcanzan todos los atributos sin ninguna transitividad se concluye lo siguiente.

  - plaque → plaque soat model year
  - soat → soat plaque model year

(plaque)$^+$ = plaque model year

plaque es clave primaria

Esquema-Taxi esta en forma normal de Boyce-Codd

- En Esquema-ModelTaxi

Es una relación con los atributos model, tradeMark y trunk, dónde model es el único que identifica de manera inequívoca al resto de los atributos, pues cada modelo tiene una única marca que lo fabrica y el tamaño de su baúl depende exclusivamente del modelo. Cada atributo depende funcionalmente de model de manera completa y sin transitividad.

  - model → model tradeMark trunk

(model)$^+$ = model tradeMark trunk

mode es clave primaria

Esquema-ModelTaxi esta en forma normal de Boyce-Codd

- Esquema-Gps

Relación donde se guarda cada determinado tiempo la placa de un taxi, el momento en el que se realiza la inserción de la tupla y las corrdenadas de la ubicación del taxi en ese momento, con los atributos plaque, timestamp y coordinate, donde la clave primaria es plaque y timestamp pues identifican de manera inequívoca una coordenada que es el atributo no primo.

- plaque timestamp → plaque timestamp coordinate

(plaque timestamp)$^+$ = plaque timestamp coordinate

plaque y timestamp son la clave primaria

Esquema-Gps está en forma normal de Boyce-Codd

Un diseño de base de datos está en FNBC si cada miembro del conjunto de esquemas de relación que constituye el diseño está en FNBC.
Por lo tanto el diseño de la base de datos está en FNBC.

4. Índices

Para la tabla Client

CREATE TABLE Client (
cellphoneClient VARCHAR(10)**,**
passwordClient VARCHAR(40)**,**
nameClient VARCHAR(40)**,**
address GEOMETRY**,**
creditCard VARCHAR(16)**,**
status BOOLEAN**,**

PRIMARY KEY (cellphoneClient)
);

Se analiza la consulta más recurrente para esta tabla que es para el login:

SELECT * FROM Client WHERE cellphone = ? AND password = ? AND status = true

Se usa el índice en cellphone para encontrar todos lo usuarios con el cellphone indicado (es único por tratarse de un atributo primario en la tabla) y luego examinar si coinciden los atributos password y status. Se descartan estrategias en las que se usen más indices u otros índices, pues podrían haber muchos registros con status = true o contraseña igual en muchos usuarios.

Se usa además la organización por asociación (hash) pues para este tipo de consultas el tiempo medio de una búsqueda es una constante que no depende del tamaño de la base de datos.

- create index searchCellphoneClient on Client using hash (cellphoneClient);

Para la tabla FavCoordinates

CREATE TABLE FavCoordinates (
cellphoneClient VARCHAR(10)**,**
coordinate GEOMETRY**,**
nameCoordinate VARCHAR(40)**,**

PRIMARY KEY (cellphoneClient, coordinate)**,**
FOREIGN KEY (cellphoneClient) REFERENCES Client(cellphoneClient) ON DELETE CASCADE
);

Se usa el índice en cellphone, suponiendo una distribución de datos en las que hay muchos más usuarios clientes que usan la aplicación que la cantidad de lugares favoritos que tienen cada uno de estos clientes en caso de que se hagan consultas por toda la llave primaria.

Las consultas hechas en la aplicación buscan todas las coordenadas de un mismo cliente, son de la forma:

SELECT * FROM FavCoordinates WHERE cellphone = ?

Por estas razones se opta por la utilización de una organización por asociación (hash)

- create index searchNameFavCoordinate on FavCoordinates using hash (cellphoneClient);

Para la tabla Driver

CREATE TABLE Driver (
cellphoneDriver VARCHAR(10)**,**
cc VARCHAR(15) UNIQUE**,**
passwordDriver VARCHAR(40)**,**
nameDriver VARCHAR(40)**,**
available BOOLEAN**,**
numAccount VARCHAR(16)**,**

status BOOLEAN**,**

PRIMARY KEY (cellphoneDriver)
);

Se usan los mismos criterios que en la tabla **Client** para la elección de una organización por asociación (hash) y para elegir el atributo cellphoneDriver

- create index searchCellphoneDriver on Driver using hash (cellphoneDriver);

Para las tablas modelTaxi, Taxi y Drive se realizan las siguiente consultas implementadas respectivamente, donde se consulta usualmente por la primary key de cada tabla exceptuando la tabla Drive:

SELECT * FROM modelTaxi WHERE model = ?

SELECT * FROM Taxi WHERE plaque = ?

SELECT * FROM Drive WHERE cellphoneDriver = ?

las busquedas de la aplicación son de la forma descrita en el libro:

select A1, A2, ..., An
from r
where Ai= c

Por lo que se usa la organización por asociación (hash).

- create index searchModel on modelTaxi using hash (model);
- create index searchTaxi on Taxi using hash (plaque);
- create index searchTaxiDrive on Drive using hash (cellPhoneDriver);

Para la tabla Ask

CREATE TABLE Ask (
idAsk SERIAL**,**
cellphoneClient VARCHAR(10)**,**
cellphoneDriver VARCHAR(10)**,**
initialCooridnates GEOMETRY**,**

finalCooridnates GEOMETRY,
initialTime TIMESTAMP,
finalTime TIMESTAMP,
stars INTEGER,

PRIMARY KEY (idAsk),
FOREIGN KEY (cellphoneClient) REFERENCES Client(cellphoneClient) ON DELETE CASCADE,
FOREIGN KEY (cellphoneDriver) REFERENCES Driver(cellphoneDriver) ON DELETE CASCADE
);

Se realizan dos consultas en la implementación, una para ver el historial de los usuarios clientes y otra para los viajes hechos por los usuarios conductores

SELECT * FROM Ask WHERE cellphoneClient = ?

SELECT * FROM Ask WHERE cellphoneDriver = ?

Para la realización de cada busqueda se crean dos indices usando organización por asociación (hash)

- create index searcAskClient on Ask using hash (cellphoneClient);
- create index searcAskDriver on Ask using hash (cellphoneDriver);

Para la tabla Gps

CREATE TABLE Gps (
plaque VARCHAR(6),
timestamp TIMESTAMP,
coordinate GEOMETRY,

PRIMARY KEY (plaque, timestamp),
FOREIGN KEY (plaque) REFERENCES Taxi(plaque)
);

Para simular un gps, se deben almacenar demasiados registros con puntos de su ubicación en intervalos de tiempo muy pequeños para hacer la ruta. Por esta razón, aunque en el proyecto no se generen tantos registros, una consulta muy usada seria:

SELECT * FROM Gps WHERE plaque = ? and timestamp >= ? and timestamp <= ?

Se realiza entonces una organización ordenada (btree) teniendo esto presente para un caso más real, usando los campos plaque y timestamp.

- create index intervalGps on Gps using btree (plaque, timestamp);

5. Vistas

Las vistas implementadas en este proyecto son vistas no materializadas, su función es reducir la complejidad a la hora de hacer querys.

5.1. Vista, ultimo taxi de cada conductor registrado

```
CREATE OR REPLACE VIEW lastPlaqueDriver AS (
   WITH
   driverLastRecord AS
   (select Driver.cellphoneDriver,  max(date) AS date
   from driver inner join drive on driver.cellphonedriver=drive.cellphonedriver
   group by (driver.cellphoneDriver)),

   driverLastPlaque AS
   (SELECT Drive.cellphoneDriver, Drive.plaque
   FROM driverLastRecord INNER JOIN drive
   ON driverLastRecord.cellphonedriver = drive.cellphoneDriver
   AND driverLastRecord.date = drive.date )

SELECT * FROM driverLastPlaque
);
```

En la tabla drive se encuentra la relación entre conductores y taxis con la fecha en la que se asociaron, la vista es una tabla con las columnas cellphoneDriver y plaque, correspondientes a la última placa registrada por cada conductor sin importar su disponibilidad.

5.2. Vista, última coordenada de un conductor

```
CREATE OR REPLACE VIEW lastCoordinatesPlaques AS (
   WITH
   driverLastRecord AS
   (select Driver.cellphoneDriver,  max(date) AS date
   from driver inner join drive on driver.cellphonedriver=drive.cellphonedriver
   where available = true and status = true
```

```
    group by (driver.cellphoneDriver)),

    driverLastPlaque AS
    (SELECT Drive.cellphoneDriver, Drive.plaque
    FROM driverLastRecord INNER JOIN drive
    ON driverLastRecord.cellphonedriver = drive.cellphoneDriver
    AND driverLastRecord.date = drive.date ),

    plaqueLastRecord AS
      (SELECT driverLastPlaque.cellphoneDriver, driverLastPlaque.plaque, max(timestamp) AS
timestamp
    FROM driverLastPlaque INNER JOIN Gps
    ON driverLastPlaque.plaque = Gps.plaque
    GROUP BY (driverLastPlaque.cellphoneDriver, driverLastPlaque.plaque)),

    plaqueLastCoordinate AS
    (SELECT plaqueLastRecord.cellphoneDriver, plaqueLastRecord.plaque, Gps.coordinate
    FROM plaqueLastRecord INNER JOIN Gps
    ON plaqueLastRecord.plaque = Gps.plaque
    AND plaqueLastRecord.timestamp = Gps.timestamp)

SELECT * FROM plaqueLastCoordinate
);
```

Esta es una vista que retorna las columnas cellphoneDriver, plaque y coordinate, correspondientes a cada conductor con el último carro registrado, es decir, su carro actual con la última coordenada registrada. Se entiende la última coordenada registrada por el carro como la posición actual del conductor.

5.3. HistoryClients / HistoryDrivers

```
CREATE OR REPLACE VIEW historyClients AS (
    SELECT cellphoneClient, POINT(initialCoordinates) AS initialPoint, POINT(finalCoordinates)
AS finalPoint, distance(initialCoordinates, finalCoordinates) AS distance, stars
    FROM Ask
);
CREATE OR REPLACE VIEW historyDrivers AS (
    SELECT cellphoneDriver, POINT(initialCoordinates) AS initialPoint, POINT(finalCoordinates)
AS finalPoint, distance(initialCoordinates, finalCoordinates) AS distance, stars
    FROM Ask
);
```

Son dos vistas pero con mucho en común, cada una ofrece una vista con las columnas ya sea cellphoneClient o cellphoneDriver con initialCoordinates, finalCoordinates, distance y stars. Relacionando cada teléfono ya sea de la tabla Client o de la tabla Driver con los servicios en los que se encuentran relacionados, con la posición inicial y final de cada viaje, la distancia calculada con una función implementada en plpgsql y el número de estrellas que dió cada cliente cuando finaliza dicho servicio.

6. Disparadores

La implementación de disparadores en el proyecto fue necesaria para realizar auditoría de los cambios realizados en cualquier tabla del proyecto (delete, update e insert).

6.1. Tabla auditoria

La tabla donde se auditan las modificaciones realizadas por todos los usuarios es creada de la siguiente manera:

```
CREATE TABLE Audit (
table_name  VARCHAR(30),
operation CHAR(1),
old_value VARCHAR(250),
new_value VARCHAR(250),
user_name VARCHAR(30),
date_oper TIMESTAMP
);
```

En esta tabla se guarda, el nombre, la operación (‘D’ para Delete, ‘U’ para Update e ‘I’ para Insert), el valor antiguo, el valor nuevo, el usuario que ejecuta el query y una fecha que debe corresponder a la del momento en el que se ejecutó.

6.2. Función auditoria / Función del disparador

A continuación, la implementación de la función encargada de activarse cuando se hace una modificación en alguna tabla del proyecto:

```
CREATE OR REPLACE FUNCTION fn_audit () RETURNS trigger AS
$$
BEGIN
        IF (TG_OP = 'DELETE') THEN
                INSERT INTO Audit (table_name, operation, old_value, new_value, user_name, date_oper)
                        VALUES (TG_TABLE_NAME, 'D', ROW(OLD.*), NULL, USER, now());
                RETURN OLD;
        ELSIF (TG_OP = 'UPDATE') THEN
                INSERT INTO Audit (table_name, operation, old_value, new_value, user_name, date_oper)
                        VALUES (TG_TABLE_NAME, 'U', ROW(OLD.*), ROW(NEW.*), USER, now());
                RETURN NEW;
        ELSIF (TG_OP = 'INSERT') THEN
                INSERT INTO Audit (table_name, operation, old_value, new_value, user_name, date_oper)
                        VALUES (TG_TABLE_NAME, 'I', NULL, ROW(NEW.*), USER, now());
                RETURN NEW;
        END IF;
        RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

Se identifica la operación básica del query y se hace un insert con todos los atributos posibles correspondientes al query dentro de la tabla Audit.

6.3. Creación de los disparadores

Se crea un disparador por cada tabla que hace parte del proyecto:

Tabla Cliente

CREATE TRIGGER t_aduit_client
AFTER INSERT OR UPDATE OR DELETE
ON Client
FOR EACH ROW
EXECUTE PROCEDURE fn_audit();

Tabla FavCoordinate

CREATE TRIGGER t_aduit_favCoordinates
AFTER INSERT OR UPDATE OR DELETE
ON FavCoordinates
FOR EACH ROW
EXECUTE PROCEDURE fn_audit();

Tabla Ask

```
CREATE TRIGGER t_aduit_ask
AFTER INSERT OR UPDATE OR DELETE
ON Ask
FOR EACH ROW
EXECUTE PROCEDURE fn_audit();
```

Tabla Driver

```
CREATE TRIGGER t_aduit_driver
AFTER INSERT OR UPDATE OR DELETE
ON Driver
FOR EACH ROW
EXECUTE PROCEDURE fn_audit();
```

Tabla Drive

```
CREATE TRIGGER t_aduit_drive
AFTER INSERT OR UPDATE OR DELETE
ON Drive
FOR EACH ROW
EXECUTE PROCEDURE fn_audit();
```

Tabla Taxi

```
CREATE TRIGGER t_aduit_taxi
AFTER INSERT OR UPDATE OR DELETE
ON Taxi
FOR EACH ROW
EXECUTE PROCEDURE fn_audit();
```

Tabla ModelTaxi

```
CREATE TRIGGER t_aduit_modeltaxi
AFTER INSERT OR UPDATE OR DELETE
ON ModelTaxi
FOR EACH ROW
EXECUTE PROCEDURE fn_audit();
```

Tabla Gps

```
CREATE TRIGGER t_aduit_gps
AFTER INSERT OR UPDATE OR DELETE
ON Gps
FOR EACH ROW
EXECUTE PROCEDURE fn_audit();
```

6.4. Permisos a los usuarios

En el software “Not so easy Taxi” se hace uso de 3 usuarios con roles diferentes que son:

1. Postgres, que tiene todos los permisos
2. clientRole
3. driverRole

Para la implementación de la auditoría es necesario entonces darle permisos a los usuarios clientRole y driverRole los permisos para insertar en la tabla Audit. Esto se consigue mediante las siguientes líneas:

Para clientRole:

```
GRANT INSERT ON TABLE Audit TO clientRole;
```

Para driverRole:

```
GRANT INSERT ON TABLE Audit TO driverRole;
```