UNIVERSIDADE FEDERAL DE SANTA CATARINA CENTRO DE CIÊNCIAS, TECNOLOGIAS E SAÚDE DO CAMPUS ARARANGUÁ CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Trabalho Final de Banco de Dados

UFSCarona

Carlos Eduardo Vitorino Gomes- 20102086

Enrico Caliolo - 20104594

Matheus Rocha Jacks - 20103932

Araranguá,

2022

Uma breve descrição **3**

2. Problema a ser abordado **3**
2.1 Contextualização 3
2.2 Problema 3

3. Modelagem conceitual **5**

4. Modelo lógico **6**

5. Script DDL **6**

6. Consultas **6**
6.1 - Quantidade de carros por campus da UFSC 7
6.2 - POIs mais usados em viagens 8
6.3 - Número de viagens por campus da UFSC 9
6.4 - Método de pagamento mais usado em viagens. 10
6.5 - Consulta para login 11
6.6 - Buscar viagens 11
6.7 - Buscar informações do usuário 12
6.8 - Pegar id do POI 13
6.9 - Buscar histórico de viagem do usuário 14

Uma breve descrição

Pensando na locomoção dos estudantes da UFSC do campus onde estudam para suas cidades natais e vice-versa, criaremos um banco de dados para um aplicativo, onde estudantes poderão oferecer caronas para outros com localizações próximas.

2. Problema a ser abordado

2.1 Contextualização

Grande parte dos discentes da Universidade Federal de Santa Catarina (UFSC) são originários de diversas cidades no Brasil. Alguns desses alunos são de regiões próximas ao campus da UFSC onde estudam, ocasionando com que visitem seus familiares e/ou amigos com mais frequência do que alunos que são nativos de cidades mais distantes.

As passagens de ônibus são normalmente a opção mais simples para viagens intermunicipais, são seguras e podem ser compradas online, porém normalmente são caras e a viagem costuma demorar muito, por causa dos famosos ônibus ‘pinga-pinga’. Uma solução já existente para isso são as caronas, como a aplicativo Blablacar, por exemplo, entretanto esse aplicativo é aberto para qualquer pessoa que queira oferecer uma carona, causando uma certa insegurança ao pegar uma carona.

Outro problema que acontece com uma certa frequência ao pegar uma carona com o Blablacar é que as pessoas não gostam de entrar nas cidades para buscar o passageiro, o que na cidade de Araranguá acaba sendo um grande problema, visto que a BR-101, estrada normalmente utilizada pelos caroneiros, fica longe da cidade, causando um custo enorme de locomoção até o ponto de encontro da carona.

2.2 Problema

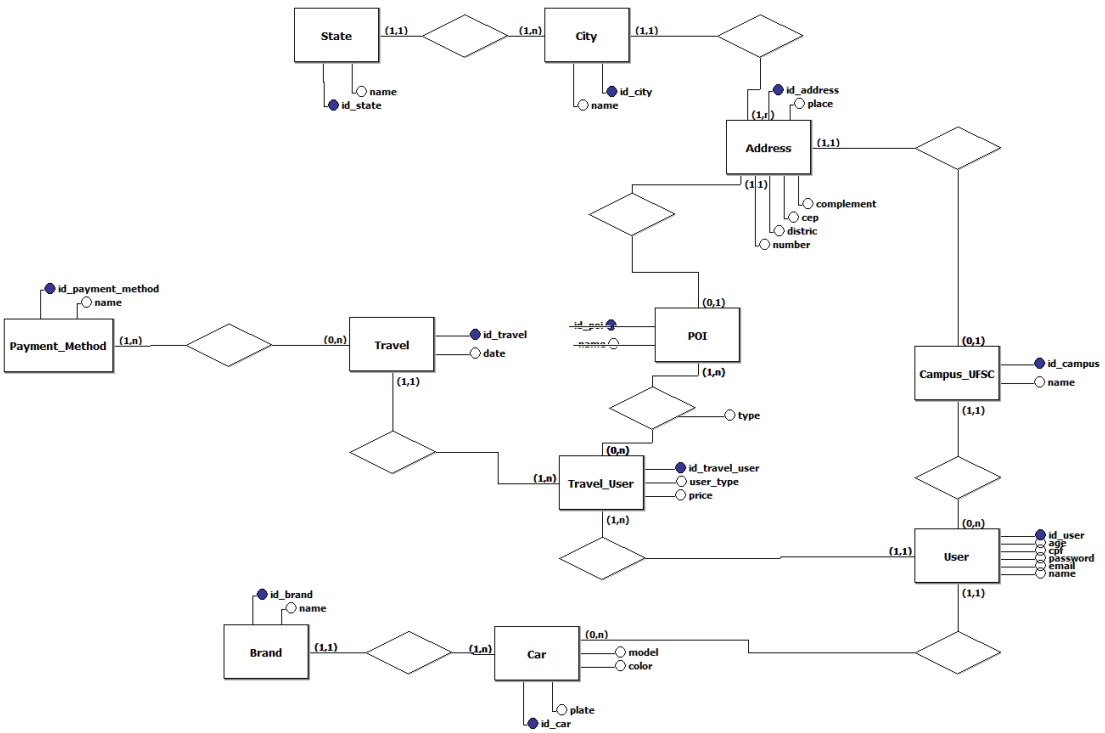
A ideia do aplicativo é que um discente da UFSC consiga disponibilizar caronas para outros discentes da UFSC, que tem o destino como localização próxima do motorista.

Um discente que queira disponibilizar carona de um ponto para o outro criará uma Viagem, que terá alguns POIs (Pontos de Interesse) no trajeto. Um outro discente que esteja saindo de um local e indo para outro local, e ambos estão no trajeto da Viagem, poderá ver os detalhes da viagem e solicitar carona, e após isso realizar o pagamento, que será especificado pelo discente que disponibilizou a carona.

O usuário terá uma opção de adicionar carros seus, porque para oferecer uma carona, o usuário deve ter pelo menos um carro. Ele também deverá marcar seu campus.

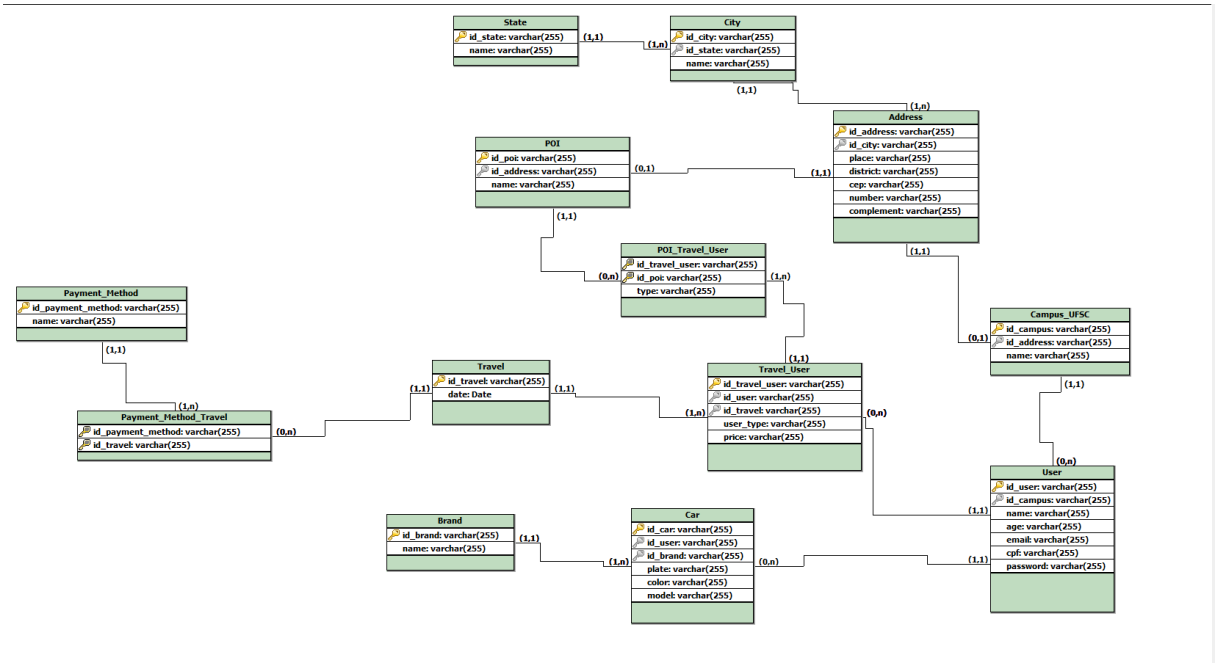
O usuário poderá ver seu histórico de viagem, tanto das caronas como passageiro como quando as disponibilizou.

3. Modelagem conceitual



Como podemos ver na figura acima, temos a tabela Travel, que vai armazenar as viagens que ocorreram. A tabela Travel_User vai ser onde ficará armazenada as informações de um usuário referente a uma viagem, já que em uma viagem podem haver quantas pessoas o carro couber. Cada usuário vai informar seus POI de entrada e saída, enquanto o motorista (discente que disponibiliza a carona) irá informar quais POIs ele vai passar. Cada POI tem um endereço associado, que está numa cidade em um estado. O usuário pode ou não ter um carro associado, e esse carro tem uma única marca, além do campus que está associado a um endereço.

4. Modelo lógico



A figura acima mostra a modelagem lógica do banco de dados. A peculiaridade foi a inclusão de duas chaves candidatas (id_user, id_travel) na tabela Travel_User, para evitar a propagação delas no POI_Travel_User.

5. Script DDL

O script DDL poderá ser encontrado no caminho ‘backend/brModelo/DDL.sql’.

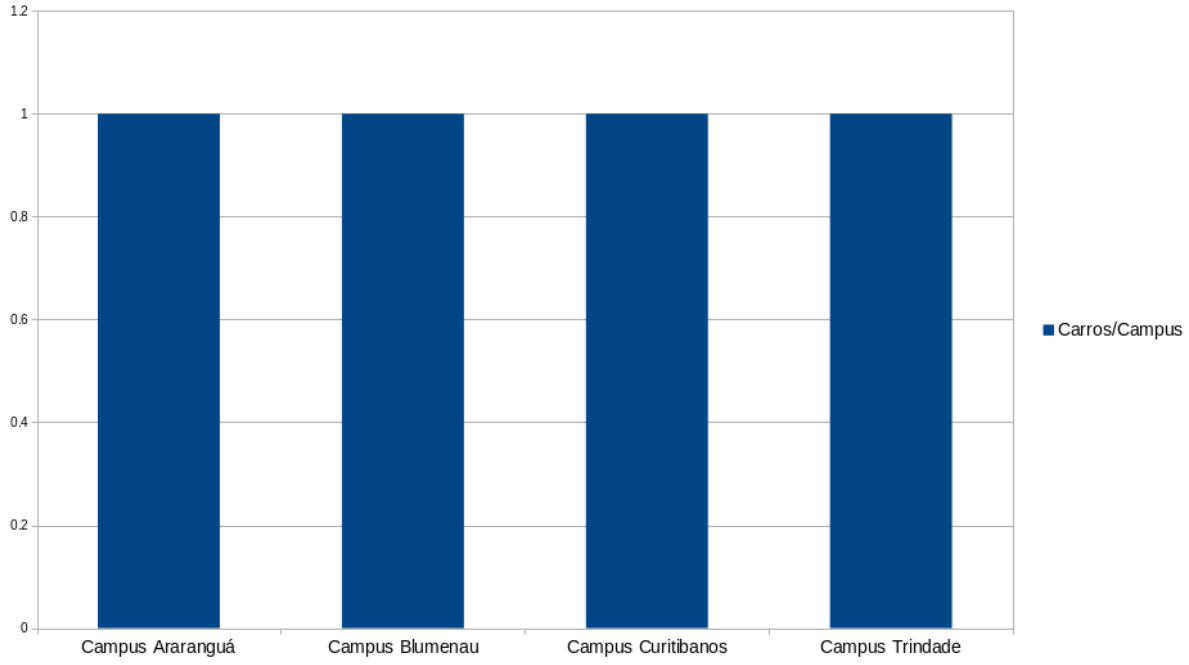
6. Consultas

Para esse projeto, pensamos em consultas que seriam para o funcionamento do aplicativo, que não englobam os requisitos pedidos pelo professor. Dessa forma, realizamos três consultas adicionais que atendem aos requisitos pedidos. Das consultas 6.1 a 6.4, temos as consultas que atendem aos requisitos. Dali em diante, são consultas que realizamos pensando no protótipo do aplicativo.

6.1 - Quantidade de carros por campus da UFSC

O objetivo da primeira consulta foi a quantidade de carros por campus da ufsc. Isso é, achar os usuários que possuem carros, e checar os campus de cada um.

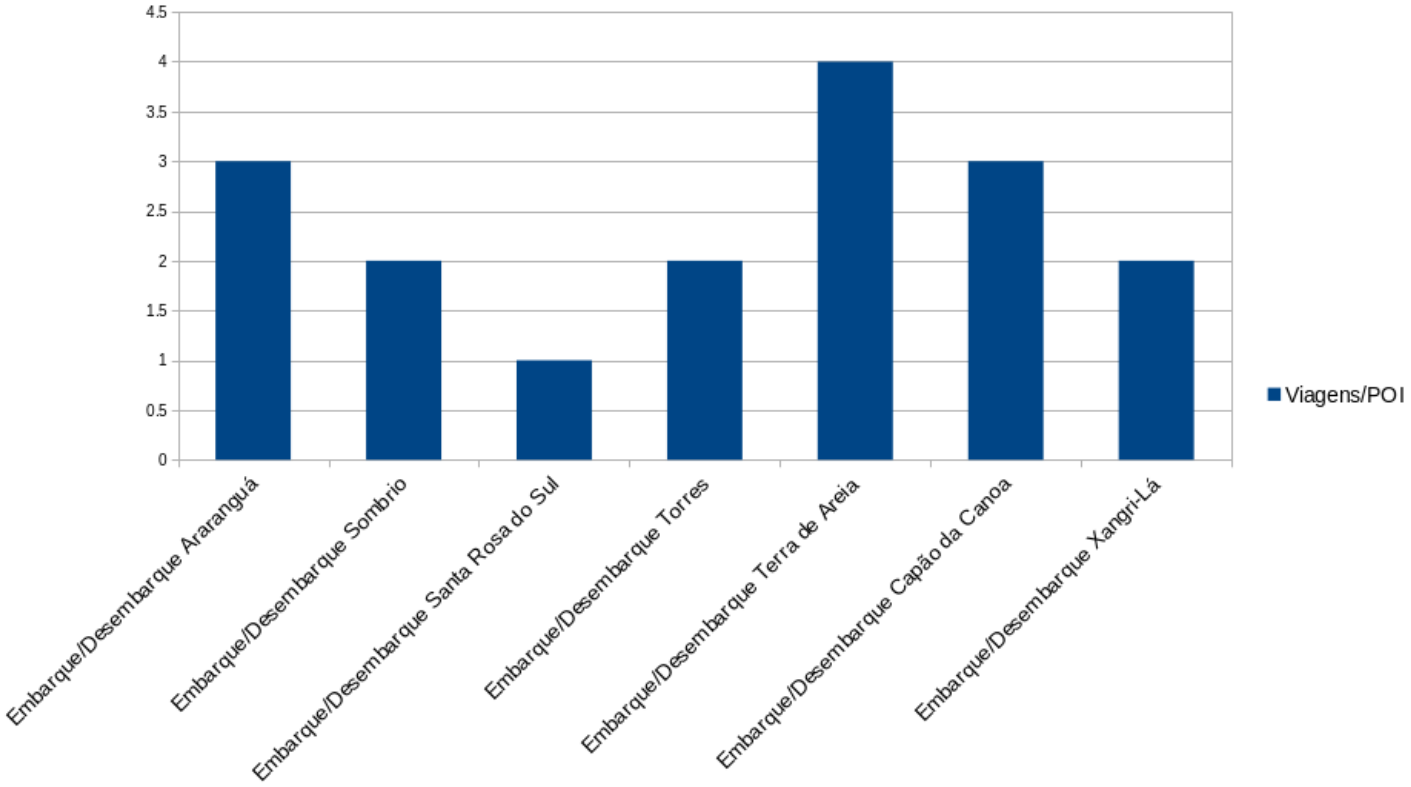
```
export const getCarsByCampus: Query<string> = async (db) => {
  try {
    const sql = `
    SELECT Campus_UFSC.name, count(*) AS total
    FROM User
    INNER JOIN Car ON Car.id_user = User.id_user
    INNER JOIN Campus_UFSC ON User.id_campus = Campus_UFSC.id_campus
    GROUP BY Campus_UFSC.name
    `;

    const [result] = await db.execute(sql);
    logQueryResult(result);

    return result as any;
  } catch (err) {
    console.log(`Error searching user: ${(err as Error).toString()}`);
  }
};
```



	name	total
	Campus Araranguá	1
▶	Campus Blumenau	1
	Campus Curitibanos	1
	Campus Trindade	1

6.2 - POIs mais usados em viagens

O objetivo da segunda consulta foi o de obter os POIs mais usados em viagens.



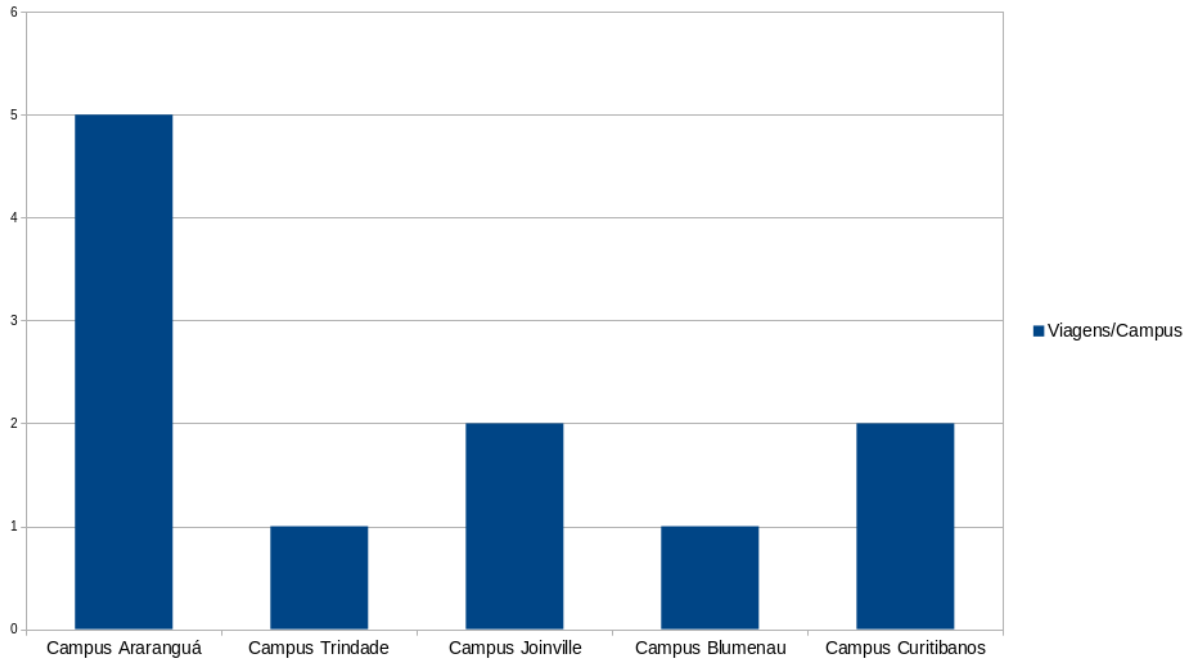

```typescript
export const getPOIsMostBadalados: Query<string> = async (db) => {
  try {
    const sql = `
    SELECT POI.name, count(*) AS total
    FROM Travel_User
    INNER JOIN POI_Travel_User ON Travel_User.id_travel_user = POI_Travel_User.id_travel_user
    INNER JOIN POI ON POI_Travel_User.id_POI = POI.id_POI
    GROUP BY POI.name
    `;

    const [result] = await db.execute(sql);
    logQueryResult(result);

    return result as any;
  } catch (err) {
    console.log(`Error searching user: ${(err as Error).toString()}`);
  }
};
```

	name	total
▶	Embarque/Desembarque Araranguá	3
	Embarque/Desembarque Sombrio	2
	Embarque/Desembarque Santa Rosa do Sul	1
	Embarque/Desembarque Torres	2
	Embarque/Desembarque Terra de Areia	4
	Embarque/Desembarque Capão da Canoa	3
	Embarque/Desembarque Xangri-Lá	2

6.3 - Número de viagens por campus da UFSC

O objetivo da terceira consulta foi buscar o número de viagens por campus. Isso é, buscar o campus de cada usuário que participou de uma viagem.

```
export const getTravelsByCampus: Query<string> = async (db) => {
  try {
    const sql = `
    SELECT Campus_UFSC.name, count(*) as total from Travel
    INNER JOIN Travel_User ON Travel.id_travel = Travel_User.id_travel
    INNER JOIN User ON Travel_User.id_user = User.id_user
    INNER JOIN Campus_UFSC ON User.id_campus = Campus_UFSC.id_Campus
    GROUP BY Campus_UFSC.name
    `;

    const [result] = await db.execute(sql);
    logQueryResult(result);

    return result as any;
  } catch (err) {
    console.log(`Error searching user: ${(err as Error).toString()}`);
  }
};
```

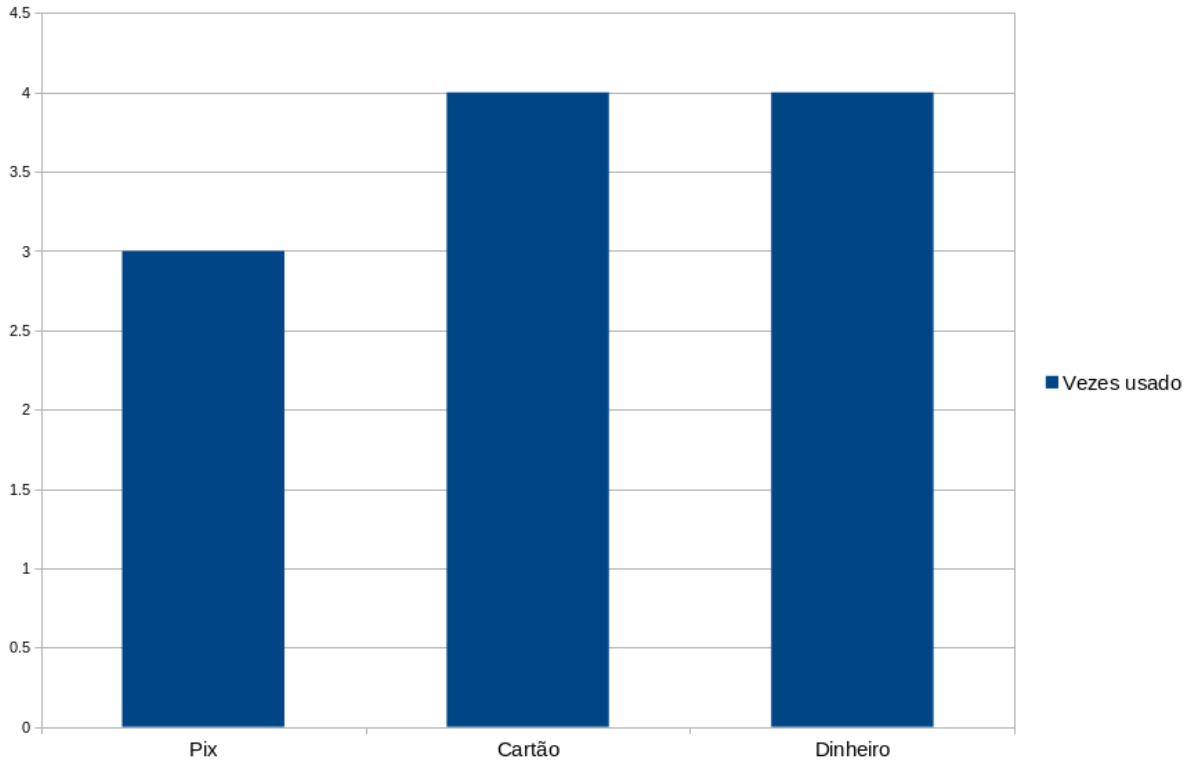


	name	total
▶	Campus Araranguá	5
	Campus Trindade	1
	Campus Joinville	2
	Campus Blumenau	1
	Campus Curitibanos	2

6.4 - Método de pagamento mais usado em viagens.

O objetivo da quarta consulta foi o de achar o método de pagamento mais usado nas viagens.

```
export const getPaymentMethodUses: Query<string> = async (db) => {
  try {
    const sql = `
    select PM.name, count(*)
    from Travel T
    inner join Payment_Method_Travel PMT on PMT.id_travel = T.id_travel
    inner join Payment_Method PM on PM.id_payment_method = PMT.id_payment_method
    group by PM.name
    `;

    const [result] = await db.execute(sql);
    logQueryResult(result);

    return result as any;
  } catch (err) {
    console.log(`Error searching user: ${(err as Error).toString()}`);
  }
};
```



	name	count(*)
▶	Pix	3
	Cartão	4
	Dinheiro	4

6.5 - Consulta para login

Essa primeira query, pensando no aplicativo, seria pro usuário fazer o login na aplicação.

```
export const loginUser: Query<{ email: string; password: string }> = async (
  db,
  data
) => {
  try {
    const sql = `
    SELECT User.id_User, User.name, User.age, User.email, User.cpf, User.id_campus
    FROM User
    INNER JOIN Campus_UFSC ON Campus_UFSC.id_campus = User.id_campus
    WHERE User.email = :email AND User.password = :password;
    `;

    const [result] = await db.execute(sql, data);
    logQueryResult(result);

    return result as any;
  } catch (err) {
    console.log(`Error searching user: ${(err as Error).toString()}`);
  }
};
```

6.6 - Buscar viagens

Essa query é feita para o usuário buscar viagens. Como mecanismo de busca, ele usa o ID do POI que escolheu (há uma query para retorno do id dos POIs abaixo), o aplicativo envia a data também, e a query retorna as viagens disponíveis.

```
export const getTravels: Query<{
  date: string;
  origin: string;
  destination: string;
}> = async (db, data) => {
  try {
    const sql = `SELECT * FROM Travel WHERE
    Travel.id_travel IN
        (SELECT Travel.id_travel FROM Travel
          INNER join Travel_User on Travel_User.id_travel = Travel.id_travel
          INNER join POI_Travel_User on POI_Travel_User.id_travel_user = Travel_User.id_travel_user
            INNER join POI on POI.id_poi = POI_Travel_User.id_poi
            WHERE POI_Travel_User.id_poi = :origin AND Travel_User.user_type = 'driver' AND NOT POI_Travel_User.type = 'destination')
    AND Travel.id_travel IN
        (SELECT Travel.id_travel FROM Travel
          INNER join Travel_User on Travel_User.id_travel = Travel.id_travel
          INNER join POI_Travel_User on POI_Travel_User.id_travel_user = Travel_User.id_travel_user
            INNER join POI on POI.id_poi = POI_Travel_User.id_poi
            WHERE POI_Travel_User.id_poi = :destination AND Travel_User.user_type = 'driver' AND NOT POI_Travel_User.type = 'origin')
    AND
    Travel.date = :date`;

    const [result] = await db.execute(sql, data);
    logQueryResult(result);

    return result as any;
  } catch (err) {
    console.log(`Error searching travels: ${(err as Error).toString()}`);
  }
};
```

6.7 - Buscar informações do usuário

Essa query é para buscar as informações do usuário, como email, cpf, idade, nome, nome do campus, modelo do carro, cor do carro e placa do carro.

```
export const getUserInfo: Query<string> = async (db, id) => {
  try {
    const sql = `
    SELECT
    User.name, User.age, User.email, User.cpf,
    Campus_UFSC.name,
    Car.model, Car.color, Car.plate,
    Brand.name
    from User
    INNER JOIN Campus_UFSC on User.id_campus = Campus_UFSC.id_campus
    INNER JOIN Car on User.id_user = Car.id_user
    INNER JOIN Brand on Car.id_brand = Brand.id_brand
    where User.id_user = :id;
    `;

    const [result] = await db.execute(sql, { id });
    logQueryResult(result);

    return result as any;
  } catch (err) {
    console.log(`Error searching user: ${(err as Error).toString()}`);
  }
};
```

6.8 - Pegar id do POI

Essa query foi feita para complementar a query de buscar viagens. Ela retorna apenas o id do POI, recebendo o nome como parâmetro.

```
export const getIDPOI: Query<string> = async (db, name) => {
  try {
    const sql = `
    SELECT id_poi
    FROM POI
    INNER JOIN Address ON POI.id_address = Address.id_address
    INNER JOIN City ON Address.id_city = City.id_city
    WHERE City.name = :name;
    `;

    const [result] = await db.execute(sql, { name });
    logQueryResult(result);

    return result as any;
  } catch (err) {
    console.log(`Error searching user: ${(err as Error).toString()}`);
  }
};
```

6.9 - Buscar histórico de viagem do usuário

```
export const getUserHistory: Query<{ userId: string }> = async (
  db,
  data
) => {
  try {
    const sql = `
      select
      t.travelId,
      max(t.travelDate) as travelDate,
      max(t.travelPrice) as travelPrice,
      max(t.userType) as userType,
      max(t.poiOriginName) as poiOriginName,
      max(t.poiOriginId) as poiOriginId,
      max(t.poiDestinationName) as poiDestinationName,
      max(t.poiDestinationId) as poiDestinationId
      from (select
            Travel.id_travel as travelId,
            Travel.date as travelDate,
            Travel_User.price as travelPrice,
            Travel_User.user_type as userType,
            POI.name as poiOriginName,
            POI.id_poi as poiOriginId,
            '' as poiDestinationName,
            '' as poiDestinationId
            from User
            inner join Travel_User on Travel_User.id_user = User.id_user
            inner join Travel on Travel_User.id_travel = Travel.id_travel
            inner join POI_Travel_User on POI_Travel_User.id_travel_user = Travel_User.id_travel_user
            inner join POI on POI.id_poi = POI_Travel_User.id_poi
            where User.id_user = :userId and POI_Travel_User.type = 'origin'
            union
            select
            Travel.id_travel as travelId,
            Travel.date as travelDate,
            Travel_User.price as travelPrice,
            Travel_User.user_type as userType,
            '' as poiOriginName,
            '' as poiOriginId,
            POI.name as poiDestinationName,
            POI.id_poi as poiDestinationId
            from User
            inner join Travel_User on Travel_User.id_user = User.id_user
            inner join Travel on Travel_User.id_travel = Travel.id_travel
            inner join POI_Travel_User on POI_Travel_User.id_travel_user = Travel_User.id_travel_user
            inner join POI on POI.id_poi = POI_Travel_User.id_poi
            where User.id_user = :userId and POI_Travel_User.type = 'destination') t
      group by t.travelId;
    `;

    const [result] = await db.execute(sql, data);
    logQueryResult(result);

    return result as any;
  } catch (err) {
    console.log(`Error searching user: ${(err as Error).toString()}`);
  }
};
```