Projeto de BD - UBER, fase III

Guilherme Moreira Castilho - B51987,
Maria Eduarda Mesquita Magalhães - B51085,
Vitor Matheus do Nascimento Moreira - B51092

Novembro de 2023

Banco de Dados Uber

1 Sobre

A presente etapa do do trabalho trabalho envolve a implementação dos dados gerados nas etapas anteriores no SGBD no postgresql, além de carregar os registros já gerados, vamos também apresentar algumas instruções SQL úteis em nosso projeto.

2 Instruções DML e DDL

Nessa etapa, criamos as nossas tabelas e convertemos os dados que havíamos criados no Excel para inserção via SQL.

1. instruções DDL: DLL create tables UBER
2. instruções DDL: DML insert tables UBER

2.1 Desafios

Nessa etapa, um de nossos desafios foi durante a criação das tabelas, criar cada coluna com o tipo correto para posterior inserção de dados demanda um trabalho cuidadoso, e foi necessário refazer algumas colunas pois os tipos não batiam com os necessários. Inicialmente estávamos criando todas tabelas para depois inserir os dados, mas notamos que o trabalho de criação de tabelas e inserção de dados deveria ser algo em conjunto, e mudamos nossa metodologia para criar as tabelas/colunas e em seguida inserir os respectivos registros. Por fim, juntamos todas instruções nos arquivos disponiveis acima.

3 Comandos SQL úteis

Nessa etapa, criamos alguns comandos SQL que podem ser úteis para nosso tema.

3.1 Listando o preço e os nomes do clientes e motorista de um pedido

```sql
SELECT
    cliente.nome AS "nome cliente",
    motorista.nome AS "nome motorista",
    pedido.preco AS "preco pedido"
FROM pedido
INNER JOIN cliente ON pedido.idcliente = cliente.idcliente
INNER JOIN motorista ON pedido.idmotorista = motorista.idmotorista
ORDER BY pedido.preco DESC;
```

nome cliente	nome motorista	preco pedido
Gabriela Lopes Moreira	Till Lindemann	$66.08
João Vitor Manuel Sousa	Yago Marreta Lopes	$57.30
Guilherme Juliano Castilho	Felipe Juan Pereira	$45.31
Eduarda Valentim Moura	Marilia Rezende	$37.49
Gabriel Santos Biancardi	Isaac Brock	$23.80
Joana Mesquita Magalhães	Felipe Juan Pereira	$16.97
Luisa Sonza Quadros	Chaves Teixeira	$14.50
Milena Justino Tavares	Adriana Costeleta	$11.23
Natalia Fontenelle Araujo	Lavínia Emilly Trovão	$9.74
Ruana Oliveira Fernandes	Thiago Manoel Lima	$5.50

No comando SQL acima, fazemos uma consulta que envolve três tabelas, "pedido", "cliente" e "motorista", usando INNER JOIN para combinar registros da tabela cliente na tabela pedido com base nos ids dos clientes, e outro INNER JOIN para combinar registros da tabela motorista na tabela pedido com base nos ids dos motoristas. Como selecionamos as colunas "nome" da tabela "cliente", a coluna "nome" da tabela "motorista" e a coluna "preco" da tabela "pedido", a consulta retorna uma tabela de nomes de clientes, nomes de motoristas e os preços de seus pedidos correspondentes, ordenados do maior para o menor preço. A tabela "pedido" é usada como a tabela principal, e as tabelas "cliente" e "motorista são usadas para obter informações adicionais sobre os clientes e motoristas associados ao pedido. Acreditamos que esse comando é útil pois podemos querer saber quais pessoas estão envolvidas nos pedidos de maior ou menos valor.

3.2 Mostrando um ranking de clientes conforme seus saldos

```sql
SELECT
    nome AS "nome cliente",
    saldo,
    RANK() OVER (ORDER BY saldo DESC) AS ranking
FROM cliente
ORDER BY saldo DESC;
```

nome cliente	saldo	ranking
João Vitor Manuel Sousa	1500.02	1
Ruana Oliveira Fernandes	600.00	2
Gabriel Santos Biancardi	562.33	3
Milena Justino Tavares	231.47	4
Guilherme Juliano Castilho	230.15	5
Gabriela Lopes Moreira	170.90	6
Luisa Sonza Quadros	57.99	7
Eduarda Valentim Moura	18.78	8
Natalia Fontenelle Araujo	16.89	9
Joana Mesquita Magalhães	10.32	10

Este comando faz um ranking de clientes conforme seus saldos no aplicativo. Primeiro selecionamos as colunas úteis da tabela cliente, e criamos uma nova coluna chamada "ranking" a partir do ranking dos saldos dos clientes, depois exibimos a tabela de forma descrescente em relação ao saldo. Esse comando pode ser útil em casos onde se quer saber quais clientes tem o maior valor em saldo, para distribuição de descontos por exemplo (clientes com saldo maior gastam mais, portanto podem receber mais descontos).

3.3 Mostrando o valor total transacionado por Estado

```sql
SELECT
    localidade.estado,
    SUM(pedido.preco) AS "valor total transacionado"
FROM viagem
JOIN pedido ON viagem.idpedido = pedido.idpedido
JOIN localidade ON localidade.idlocal = viagem.idlocaldestino
GROUP BY localidade.estado
ORDER BY "valor total transacionado" DESC;
```

estado	valor total transacionado
Minas Gerais	$145.60
São Paulo	$99.62
Rio de Janeiro	$31.47
Parana	$11.23

No comando acima, listamos o valor total transacionado por estado de acordo com o local de destino, isto é, quais estados de destino transacionam o maior valor. Primeiro selecionamos a coluna estado da tabela "localidade", depois criamos uma coluna que ira somar os preços das viagens. Fazemos essa seleção na tabela viagem, e depois juntamos a tabela pedido conforme o id do pedido, e juntamos a localidade conforme o idlocal da tabela "localidade" e o "idlocaldestino" da tabela "viagem", por fim agrupamos nossos registros conforme o estado, e exibimos as colunas selecionadas coforme orda descrescente do valor total transacionado. Esse comando pode ser útil em uma situação na qual se quer saber quais estados atraem mais clientes, para um possivel investimento por parte da UBER, por exemplo.

3.4 Mostrando o valor total rotacionado por cada motorista

```sql
SELECT
    motorista.nome AS "nome motorista",
    SUM(pedido.preco) AS "valor rotacionado total"
FROM pedido
INNER JOIN motorista ON pedido.idmotorista = motorista.idmotorista
GROUP BY motorista.idmotorista
ORDER BY "valor rotacionado total" DESC;
```

Nome Motorista	Valor Rotacionado Total
Till Lindemann	$66.08
Felipe Juan Pereira	$62.28
Yago Marreta Lopes	$57.30
Marilia Rezende	$37.49
Isaac Brock	$23.80
Chaves Teixeira	$14.50
Adriana Costeleta	$11.23
Lavínia Emilly Trovão	$9.74
Thiago Manoel Lima	$5.50

Neste comando, verificamos o total rotacionado por cada motorista. Primeiro selecionamos as colunas "nome" da tabela motorista e criamos uma coluna que ira somar os preços, fazemos tal seleção na tabela pedido, e concatenamos as partes de interesse da tabela motorista conforme o id do motorista. Depois agrupamos por motorista (o que faz também a soma da coluna de valor rotacionado), e retornamos a tabela em ordem descrescente de valor. Esse comando pode ser interessante quando se quer equilibrar o valor rotacionado por cada motorista por exemplo, (sugerir mais corridas para os que estão rodando pouco valor).