UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

TRABALHO FINAL

INTRODUÇÃO A BANCO DE DADOS

2019/1

Tema: Gerenciador de peladas

Participantes:

Breno Poggiali de Sousa

Gabriela Peres Neme

Guilherme de Abreu Lima Buitrago Miranda

Vinicius Julião Ramos [líder]

Introdução

O objetivo do presente trabalho é desenvolver um sistema web responsável pelo gerenciamento e organização de partidas de futebol entre amigos, popularmente conhecido como peladas. Nesse sistema existem grupos de peladas, nos quais os integrantes organizam as partidas, com data, custo e local, bem como podem organizar vaquinhas para arrecadação de dinheiro para eventos, confraternizações e itens personalizados, como camisas.

Ademais, é possível avaliar o desempenho de jogadores, jogos e grupos de pelada através de métricas registradas para cada jogador, em cada partida. Para isso são armazenados no banco de dados número de gols do jogador, faltas cometidas, assistências, defesas, etc.

Para atingir esse objetivo, foram criados um banco de dados, uma interface gráfica através de sistema web composto por *front end* e *back end*, consultas em SQL, consultas do tipo relatório, análise de eficiência das consultas e utilização de dados do tipo não convencional para exibição de fotos dos jogadores e logos de grupos de pelada.

Banco de dados

Descrição textual

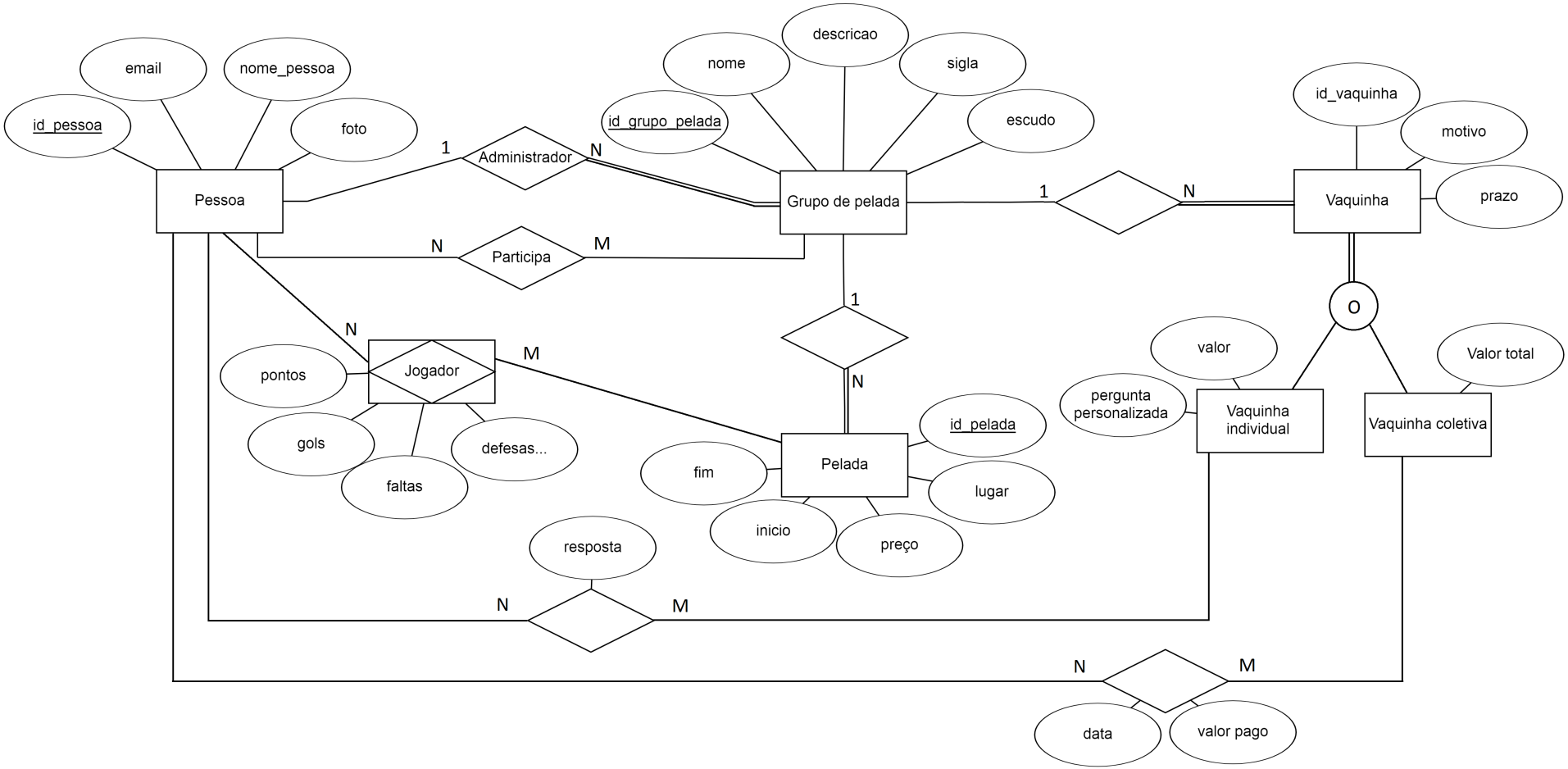
O banco de dados é composto por pessoas, com nome, e-mail, foto e um identificador. Cada pessoa pode participar de diversos grupos de pelada, e cada grupo de pelada deve ter um código identificador, um nome e um administrador, podendo ter, também, descrição, sigla e escudo.

Cada grupo de pelada pode organizar diversas partidas (peladas), e essas devem ter um identificador, lugar, preço, data de início e de fim.

Todos os integrantes do grupo de pelada podem participar das partidas (peladas) criadas no grupo. Assim, para cada integrante que participar de uma partida específica, serão salvos os pontos do jogador na partida, sua nota, roubadas de bola, faltas cometidas, gols contra, cartões amarelos, cartão vermelho, jogo sem sofrer gol, defesa difícil, defesa de pênalti, gol sofrido, falta sofrida, passe errado, assistência, finalização na trave, finalização defendida, finalização para fora, gol, impedimento e pênalti perdido.

Além disso, cada grupo de peladas pode organizar vaquinhas, que devem ter um motivo e um prazo. As vaquinhas podem ser coletivas ou individuais. As coletivas possuem um valor total, o valor pago por cada integrante da vaquinha e a data do pagamento de cada um. As vaquinhas individuais, por sua vez, possuem uma pergunta personalizada, o valor que cada participante deve pagar e a resposta à pergunta personalizada para cada um dos integrantes dessa vaquinha.

Diagrama ER


email
nome_pessoa
id_pessoa
foto
Pessoa
1
Administrador
N
descricao
nome
sigla
id_grupo_pelada
escudo
Grupo de pelada
1
N
id_vaquinha
motivo
prazo
Vaquinha
N
Participa
M
1
O
valor
Valor total
pergunta personalizada
Vaquinha individual
Vaquinha coletiva
N
Jogador
M
pontos
gols
faltas
defesas...
N
id_pelada
fim
Pelada
lugar
inicio
preço
resposta
N
M
N
M
data
valor pago

Esquema relacional

Pessoa (id_pessoa, email, nome_pessoa, foto)

Grupo_de_pelada (id_grupo_de_pelada, nome, descrição, sigla, escudo, id_adm)

id_adm REFERENCIA Pessoa

Participa_grupo_pelada (id_grupo_de_pelada, id_pessoa)

id_grupo_de_pelada REFERENCIA Grupo_de_pelada

id_pessoa REFERENCIA Pessoa

Pelada (id_pelada, lugar, preco, inicio, fim, id_grupo_de_pelada)

id_grupo_de_pelada REFERENCIA Grupo_de_pelada

Jogador (id_pessoa, id_pelada, Pontos, Partida, Nota, FS, PE, A , FT, FD, FF, G, I, PP, RB, FC, GC, CA, CV, SG, DD, DP, GS)

id_pessoa REFERENCIA Pessoa

id_pelada REFERENCIA Pelada

Vaquinha (id_vaquinha, motivo, id_grupo_de_pelada, prazo)

id_grupo_de_pelada REFERENCIA Grupo_de_pelada

Vaquinha_Coletiva (id_vaquinha_coletiva, valor_total)

id_vaquinha_coletiva REFERENCIA Vaquinha

Vaquinha_Coletiva_Pessoa (id_vaquinha_coletiva, id_pessoa, valor_pago, data)

id_vaquinha_coletiva REFERENCIA Vaquinha_Coletiva

id_pessoa REFERENCIA Pessoa

Vaquinha_Individual (id_vaquinha_individual, pergunta_personalizada, valor)

id_vaquinha_individual REFERENCIA Vaquinha

id_pessoa REFERENCIA Pessoa

Vaquinha_Individual_Pessoa (id_vaquinha_individual, id_pessoa, resposta)

id_vaquinha_individual REFERENCIA Vaquinha_Individual

id_pessoa REFERENCIA Pessoa

Para o povoamento do banco de dados foi escolhida a base do jogo eletrônico Cartola, famoso *Fantasy Game* de futebol, disponíveis em formato csv no repositório público do GitHub “caRtola”. Dessa forma, as tabelas “Pessoa”, “Jogador”, “Grupo_de_Pelada” e “Pelada” foram alimentadas com dados dos jogadores e das partidas do Brasileirão Série A entre 2014 e 2017.

Além disso, para normalização e transformação dos arquivos csvs para inserts em linguagem SQL, diversos scripts Python foram desenvolvidos, utilizando sobretudo, a biblioteca Pandas para manipulação dos dados.

Consultas realizadas

Consulta 1

Lista, para uma vaquinha coletiva específica, todas as pessoas que estão participando e o valor já pago por cada um dos participantes.

Na consulta abaixo se considerou a vaquinha coletiva de id igual a 2.

```
SELECT nome_pessoa, valor_pago
FROM Vaquinha_Coletiva_Pessoa NATURAL JOIN Pessoa
WHERE id_vaquinha_coletiva = 2
ORDER BY nome_pessoa;
```

Consulta 2

Lista o número de pontos, a nota, o número de gols e o número de gols contra de todos os jogadores que participaram de uma determinada partida, ordenando alfabeticamente.

Na consulta abaixo se considerou a pelada de id igual a 9.

```
SELECT nome_pessoa, Pontos, Nota, G, GC
FROM Jogador NATURAL JOIN Pessoa
WHERE Jogador.id_pelada = 9
ORDER BY nome_pessoa;
```

Consulta 3

Lista o nome e a descrição de todos os grupos dos quais determinado usuário é administrador.

Na consulta abaixo se considerou a usuária cujo e-mail é helena@gmail.com.

```
SELECT Grupo_de_pelada.nome, Grupo_de_pelada.descricao
FROM Pessoa JOIN Grupo_de_Pelada ON Pessoa.id_pessoa =
       Grupo_de_Pelada.id_adm
WHERE Pessoa.email = 'helena@gmail.com' ;
```

Consulta 4

Lista todas as peladas de um grupo que estão compreendidas em um intervalo de tempo.

Na consulta abaixo se considerou o grupo de pelada cujo id é 1, a data início igual a 04/06/2019 e a data fim igual a 30/2/2019.

```
SELECT lugar, preco, inicio, fim
FROM Grupo_de_pelada NATURAL JOIN Pelada
```

```
WHERE id_grupo_de_pelada = 1 AND
     (inicio BETWEEN '2019-06-04 00:00:00' AND '2019-12-30
      00:00:00')
ORDER BY inicio;
```

Consulta 5

Lista motivo e prazo de todas as vaquinhas pertencentes a um grupo que estão compreendidas em um intervalo de tempo.

Na consulta abaixo se considerou o grupo de pelada cujo id é 1, a data início igual a 04/06/2019 e a data fim igual a 30/2/2019.

```
SELECT motivo, prazo
FROM Grupo_de_pelada NATURAL JOIN Vaquinha
WHERE id_grupo_de_pelada = 1 AND
     (prazo BETWEEN '2019-06-04 00:00:00' AND '2019-12-30
     00:00:00')
ORDER BY prazo;
```

Consulta 6

Lista nome e descrição de todos os grupos de pelada a que uma pessoa pertence.

Na consulta abaixo se considerou o usuário cujo e-mail é matheus@gmail.com.

```
SELECT Grupo_de_pelada.nome, Grupo_de_pelada.descricao
FROM Pessoa NATURAL JOIN Participa_grupo_pelada NATURAL JOIN
      Grupo_de_pelada
WHERE Pessoa.email = 'matheus@gmail.com';
```

Consulta 7

Lista as últimas partidas que o jogador participou, o grupo de pelada em que a partida foi jogada, a data e a hora da partida, e as métricas do jogador na partida em questão, mostrando a mais recentes primeiro.

Na consulta abaixo se considerou a usuária cujo e-mail é manuela@gmail.com.

```
SELECT Grupo_de_pelada.nome, Pelada.lugar, Pelada.preco,
     Pelada.inicio, Pontos, Nota, G, GC
FROM Pessoa NATURAL JOIN Jogador NATURAL JOIN Pelada NATURAL
     JOIN Grupo_de_pelada
WHERE Pessoa.email = 'manuela@gmail.com'
ORDER BY inicio desc;
```

Consulta 8

Obtém todas as pessoas participantes de uma vaquinha individual e a coluna de personalização (resposta) correspondente.

Na consulta abaixo se considerou a vaquinha individual de id igual a 1.

```
SELECT nome_pessoa, pergunta_personalizada, valor, resposta
FROM Pessoa NATURAL JOIN Vaquinha_Individual_Pessoa NATURAL
     JOIN Vaquinha_Individual
WHERE id_vaquinha_individual = 1
ORDER BY nome_pessoa;
```

Consulta 9

Exibe o nome da pelada, o lugar em que foi feita, a data de início e o número total de gols na partida da pelada com maior número total de gols feitos, dentre as quais determinado jogador participou.

Na consulta abaixo se considerou o usuário cujo e-mail é ricardo@gmail.com.

```
SELECT Grupo_de_Pelada.nome, lugar, inicio, sum(G) as
       Gols_na_partida
FROM Jogador NATURAL JOIN Pelada NATURAL JOIN Grupo_de_Pelada
WHERE id_pelada in (SELECT id_pelada FROM Jogador NATURAL JOIN
      Pessoa WHERE email = ' ricardo@gmail.com ')
GROUP BY id_pelada
ORDER BY Gols_na_partida desc limit(1);
```

Consulta 10

Exibe o nome da vaquinha, o prazo, o total já arrecadado e o valor total de uma vaquinha coletiva.

Na consulta abaixo se considerou a vaquinha coletiva de id igual a 2.

```
SELECT motivo, prazo, sum(valor_pago) as total_pago,
valor_total
FROM (Vaquinha JOIN Vaquinha_Coletiva ON Vaquinha.id_vaquinha
      =  Vaquinha_Coletiva.id_vaquinha_coletiva) NATURAL JOIN
      Vaquinha_Coletiva_Pessoa
WHERE Vaquinha_Coletiva.id_vaquinha_coletiva = 2
GROUP BY id_vaquinha_coletiva;
```

Consultas do tipo relatório

Relatório 1

Relatório que contém todos os grupos de pelada e suas médias de pontos, gols e gols contra, apurado para todas as partidas realizadas por cada um dos grupos.

```
SELECT nome as Nome_Grupo, count(*) as Num_Peladas, avg(P) as
       Media_pontos,  avg(G) as Media_gols, avg(GC) as
       media_gols_contra
FROM Grupo_de_Pelada NATURAL JOIN
```

```
     (SELECT id_grupo_de_pelada, id_pelada, sum(Pontos) as P,
       sum (G) as G, sum(GC) as GC
     FROM Pelada NATURAL JOIN Jogador
     GROUP BY id_pelada)
GROUP BY id_grupo_de_pelada;
```

Relatório 2

Relatório que contém todos os participantes de um grupo e suas médias de pontos, gols e gols contra. Tal média é apurada através das partidas organizadas somente pelo grupo em questão.

Na consulta abaixo se considerou o grupo de pelada cujo id é 5.

```
SELECT nome_pessoa, avg(Pontos) as Media_Pontos, avg(G) as
       Media_gols, avg(GC) as Media_gols_contra
FROM Grupo_de_Pelada NATURAL JOIN Pelada NATURAL JOIN Jogador
      NATURAL JOIN Pessoa
WHERE  id_grupo_de_pelada = 5
GROUP BY id_pessoa
ORDER BY nome_pessoa;
```

Relatório 3

Relatório que mostra a evolução da média de pontos, de gols e de gols contra para um determinado jogador, mensalmente.

Na consulta abaixo se considerou o usuário cujo e-mail é manuela@gmail.com.

```
SELECT strftime('%m/%Y', inicio) as Mes, avg(Pontos) as
       Media_Pontos, avg(G) as Media_gols, avg(GC) as
       Media_gols_contra
FROM Jogador NATURAL JOIN Pessoa NATURAL JOIN Pelada
WHERE  email = ' manuela@gmail.com '
GROUP BY strftime('%m/%Y', inicio)
ORDER BY strftime('%Y', inicio), strftime('%m', inicio);
```

Análise de eficiência das consultas

Cada uma das 10 consultas realizadas anteriormente podem ser formuladas de pelo menos duas maneiras. Nesse sentido, é possível avaliá-las com o intuito de descobrir as formas mais eficientes. Para isso, as consultas serão analisadas considerando cinco critérios comparativos, sendo eles:

- NATURAL JOIN × JOIN ON
- NATURAL JOIN × produto cartesiano + condição WHERE

- JOIN × produto cartesiano + condição WHERE
- FROM a JOIN b WHERE b =... × FROM a JOIN (SELECT * FROM b WHERE b =...)
- LIMIT(1) × MAX

A metodologia aplicada na avaliação consiste em separar as consultas realizadas nos critérios estabelecidos nos itens acima, de forma que cada uma das 10 consultas é usada para avaliar um dos critérios. Em seguida, a consulta é escrita da outra maneira, considerando o critério na qual foi enquadrada, e então é computado o tempo levado para a execução de cada formulação.

Cada consulta foi executada 100 vezes, e o tempo registrado é a média de tempo das 100 execuções. Em seguida, foram feitos gráficos comparativos de tempo × forma da consulta para cada um dos critérios estabelecidos, a partir do qual foi possível concluir sobre a eficiência das formulações.

Análise 1 – NATURAL JOIN × JOIN

Serão utilizadas as consultas 1, 2 e 4 para essa avaliação.

Consulta 1

A consulta 1 realizada utilizando o NATURAL JOIN pode ser vista no tópico .

A consulta 1 realizada utilizando o JOIN foi apresentada abaixo.

```
SELECT nome_pessoa, valor_pago
FROM Vaquinha_Coletiva_Pessoa JOIN Pessoa ON
        Vaquinha_Coletiva_Pessoa.id_pessoa = Pessoa.id_pessoa
WHERE id_vaquinha_coletiva = 2
RDER BY nome_pessoa;
```

Consulta 2

A consulta 2 realizada utilizando o NATURAL JOIN pode ser vista no tópico .

A consulta 2 realizada utilizando o JOIN foi apresentada abaixo.

```
SELECT nome_pessoa, Pontos, Nota, G, GC
FROM Jogador JOIN Pessoa ON Jogador.id_pessoa = Pessoa.id_
WHERE Jogador.id_pelada = 9
ORDER BY nome_pessoa;
```

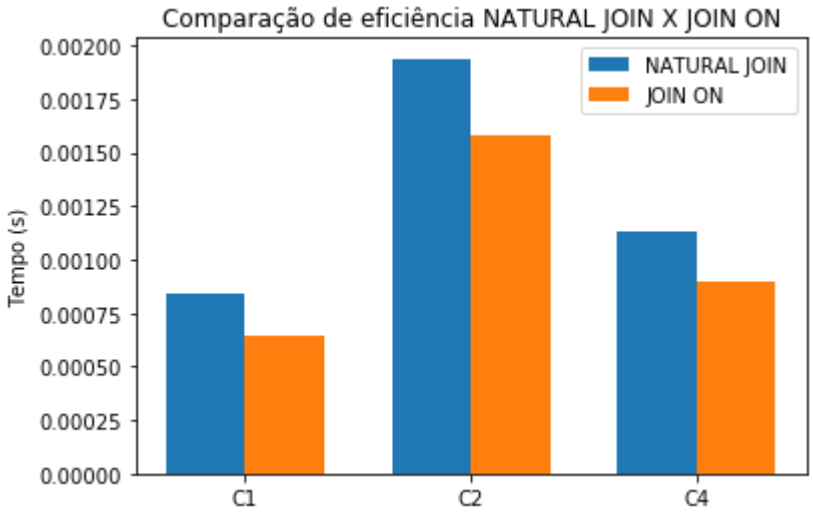
Consulta 4

A consulta 4 realizada utilizando o NATURAL JOIN pode ser vista no tópico .

A consulta 4 realizada utilizando o JOIN foi apresentada abaixo.

```
SELECT lugar, preco, inicio, fim
FROM Grupo_de_pelada JOIN Pelada ON
     Grupo_de_pelada.id_grupo_de_pelada =
     Pelada.id_grupo_de_pelada
WHERE Pelada.id_grupo_de_pelada = 1 AND
     (inicio BETWEEN '2019-06-04 00:00:00' AND '2019-12-31
     00:00:00')
ORDER BY inicio;
```

Comparação de resultados



Conclusão da análise 1

Apesar de haver variações significativas do tempo obtido a cada iteração, em todas as simulações o tempo de execução da operação NATURAL JOIN foi maior que da operação JOIN ON. Tal fato evidencia que a operação de JOIN ON é mais eficiente que a operação de NATURAL JOIN.

Uma explicação plausível para essa diferença de desempenho é que, ao utilizar NATURAL JOIN, é necessário que o SGBD realize uma busca entre os nomes das colunas, à procura daqueles que são coincidentes. Como, para a operação JOIN ON, a informação da coluna em que será feita a junção é fornecida no comando, o algoritmo de busca de colunas com nomes coincidentes não é feito, o que torna sua execução mais rápida e, consequentemente, mais eficiente.

Análise 2 - NATURAL JOIN × produto cartesiano + condição WHERE

Serão utilizadas as consultas 5 e 6 para essa avaliação.

Consulta 5

A consulta 5 realizada utilizando o NATURAL JOIN pode ser vista no tópico .

A consulta 5 realizada utilizando produto cartesiano + condição WHERE foi apresentada abaixo.

```
SELECT motivo, prazo
FROM Grupo_de_pelada, Vaquinha
WHERE id_grupo_de_pelada = 1 AND
     (prazo BETWEEN '2019-06-04 00:00:00' AND '2019-12-30
     00:00:00') AND Vaquinha.id_grupo_de_pelada =
     Grupo_de_pelada.id_grupo_de_pelada
ORDER BY prazo;
```

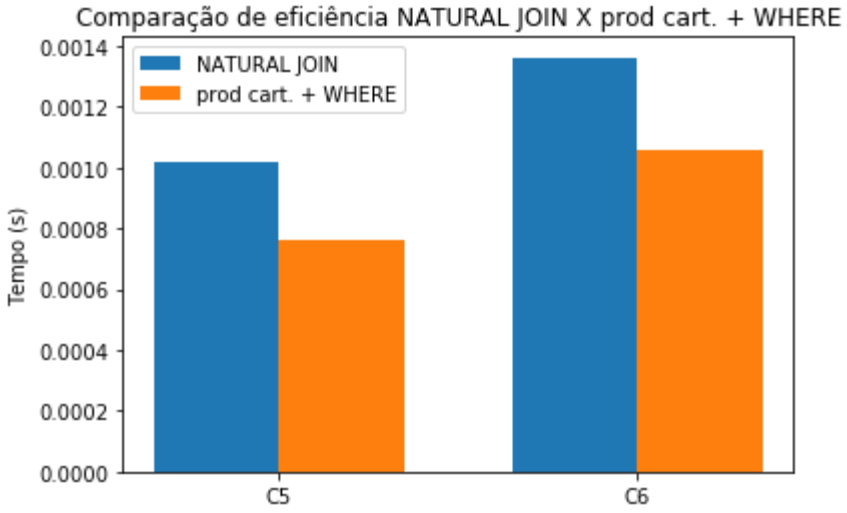
Consulta 6

A consulta 6 realizada utilizando o NATURAL JOIN pode ser vista no tópico .

A consulta 6 realizada utilizando produto cartesiano + condição WHERE foi apresentada abaixo.

```
SELECT Grupo_de_pelada.nome, Grupo_de_pelada.descricao
FROM Pessoa, Participa_grupo_pelada, Grupo_de_pelada
WHERE Pessoa.email = 'matheus@gmail.com' AND
      Pessoa.id_pessoa = Participa_grupo_pelada.id_pessoa AND
      Participa_grupo_pelada.id_grupo_de_pelada =
      Grupo_de_pelada.id_grupo_de_pelada;
```

Comparação de resultados



Conclusão da análise 2

Apesar das variações no tempo de execução em ambas as consultas, o produto cartesiano com cláusula WHERE demonstrou-se mais rápido do que o NATURAL JOIN. A explicação para o resultado observado deve-se ao fato de que, devido à maneira como foi projetado, o SGBD tem uma maior velocidade de execução em operações simples em sequência (produto cartesiano seguido da cláusula WHERE). Por outro lado, ao utilizar o NATURAL JOIN, é preciso realizar uma busca entre

os nomes das colunas à procura daqueles que são coincidentes e, posteriormente, compara-los, tornando a operação mais trabalhosa.

Dessa forma, devido à maior dificuldade da segunda operação, vê-se que o produto cartesiano é mais eficiente nos testes.

Análise 3 – JOIN × produto cartesiano + condição WHERE

Serão utilizadas as consultas 3 e 10 para essa avaliação.

Consulta 3

A consulta 3 realizada utilizando o JOIN pode ser vista no tópico .

A consulta 3 realizada utilizando produto cartesiano + condição WHERE foi apresentada abaixo.

```
SELECT Grupo_de_pelada.nome, Grupo_de_pelada.descricao
FROM Pessoa, Grupo_de_Pelada
WHERE Pessoa.email = 'helena@gmail.com'  AND
     Pessoa.id_pessoa = Grupo_de_Pelada.id_adm ;
```

Consulta 10

Consulta 10 realizada utilizando o JOIN:

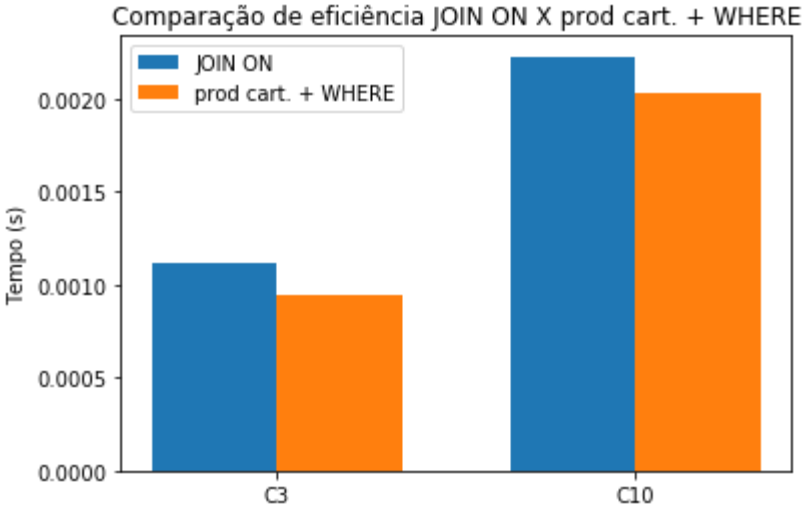
```
SELECT motivo, prazo, sum(valor_pago) as total_pago,
valor_total
FROM (Vaquinha JOIN Vaquinha_Coletiva ON Vaquinha.id_vaquinha
     =  Vaquinha_Coletiva.id_vaquinha_coletiva) JOIN
     Vaquinha_Coletiva_Pessoa ON
     Vaquinha_Coletiva.id_vaquinha_coletiva =
     Vaquinha_Coletiva_Pessoa.id_vaquinha_coletiva

WHERE Vaquinha_Coletiva.id_vaquinha_coletiva = 2
GROUP BY Vaquinha_Coletiva.id_vaquinha_coletiva;
```

Consulta 10 realizada utilizando produto cartesiano + condição WHERE:

```
SELECT motivo, prazo, sum(valor_pago) as total_pago,
valor_total
FROM Vaquinha, Vaquinha_Coletiva , Vaquinha_Coletiva_Pessoa
WHERE Vaquinha.id_vaquinha =
      Vaquinha_Coletiva.id_vaquinha_coletiva AND
      Vaquinha_Coletiva.id_vaquinha_coletiva =
      Vaquinha_Coletiva_Pessoa.id_vaquinha_coletiva AND
      Vaquinha_Coletiva.id_vaquinha_coletiva = 2
GROUP BY Vaquinha_Coletiva.id_vaquinha_coletiva;
```

Comparação de resultados



Conclusão análise 3

Embora o tempo de execução de ambas as consultas tenha sido muito parecido, a utilização do produto cartesiano seguido pela cláusula WHERE ainda foi mais eficiente que o uso do JOIN ON. Isso aconteceu, provavelmente, devido à forma com que o SGBD é construído, sendo mais eficiente a realização de consultas elementares em sequência, em comparação à consultas de junção.

Análise 4 – FROM a JOIN b WHERE b = ... × FROM a JOIN (SELECT * FROM b WHERE b = …)

Serão utilizadas as consultas 7 e 8 para essa avaliação.

Consulta 7

A consulta 7 realizada utilizando FROM a JOIN b WHERE b = ... pode ser vista no tópico .

A consulta 7 realizada utilizando FROM a JOIN (SELECT * FROM b WHERE b = ...) foi apresentada abaixo.

```
SELECT Grupo_de_pelada.nome, Pelada.lugar, Pelada.preco,
     Pelada.inicio, Pontos, Nota, G, GC
FROM Jogador NATURAL JOIN Pelada NATURAL JOIN Grupo_de_pelada
     NATURAL JOIN (SELECT * FROM Pessoa
WHERE Pessoa.email = 'manuela@gmail.com')
ORDER BY inicio desc;
```

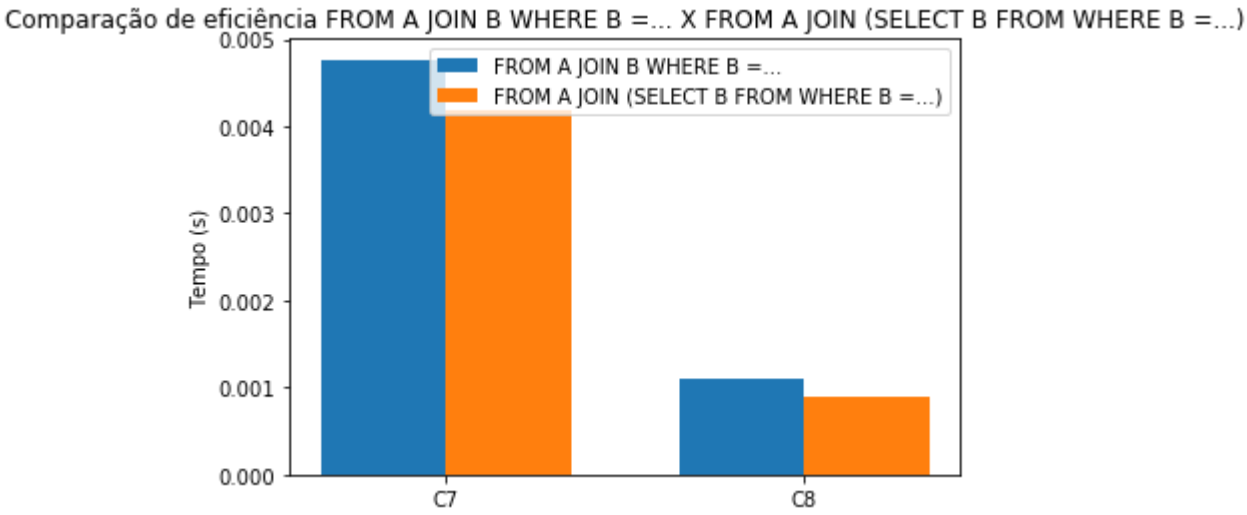
Consulta 8

A consulta 8 realizada utilizando FROM a JOIN b WHERE b = ... pode ser vista no tópico .

A consulta 8 realizada utilizando FROM a JOIN (SELECT * FROM b WHERE b = ...) foi apresentada abaixo.

```
SELECT nome_pessoa, pergunta_personalizada, valor, resposta
FROM Pessoa NATURAL JOIN Vaquinha_Individual_Pessoa NATURAL
      JOIN (SELECT * FROM Vaquinha_Individual WHERE
      id_vaquinha_individual = 1)
ORDER BY nome_pessoa;
```

Comparação de resultados



Conclusão análise 4

Apesar do tempo de execução de ambas as consultas terem sido bastante similares, a segunda abordagem mostrou-se mais otimizada pois, como o SELECT interno separa apenas as tuplas que satisfazem a condição necessária, é feito um join com uma tabela de menores proporções.

Análise 5 – LIMIT(1) × MAX()

Será utilizada a consulta 9 para essa avaliação.

Consulta 9

A consulta 9 realizada utilizando LIMIT(1) pode ser vista no tópico .

A consulta 9 realizada utilizando MAX() foi apresentada abaixo.

SELECT nome, lugar, inicio, MAX(Gols_na_partida) FROM

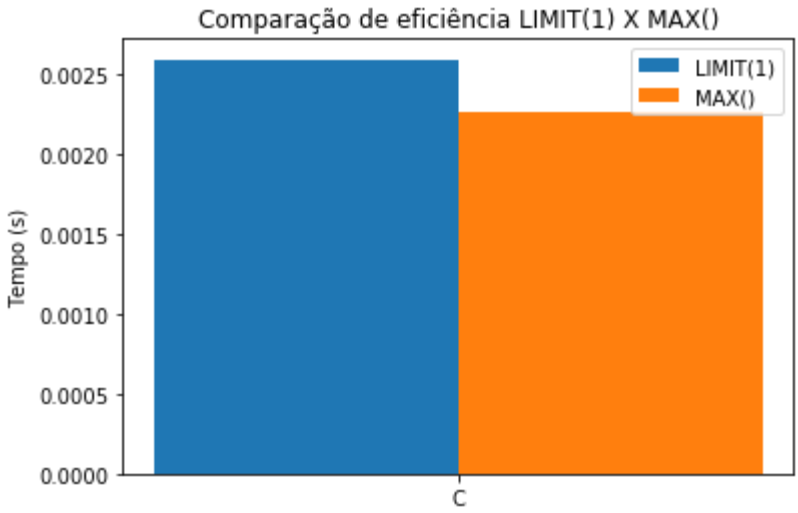
(SELECT Grupo_de_Pelada.nome, lugar, inicio, sum(G) as Gols_na_partida

FROM Jogador NATURAL JOIN Pelada NATURAL JOIN Grupo_de_Pelada

WHERE id_pelada in (SELECT id_pelada FROM Jogador NATURAL JOIN Pessoa
WHERE email = ' ricardo@gmail.com')

GROUP BY id_pelada);

Comparação de resultados



Conclusão análise 5

A consulta com a função MAX() é mais rápida que a primeira, que utiliza a ordenação decrescente e limitação do primeiro valor. Isso ocorre pois, para encontrar o maior valor, é necessário percorrer apenas uma vez toda a coluna, o que leva a uma complexidade O(n). Contudo, para ordenar os valores da coluna, é preciso utilizar algum método de ordenação (e.g. QuickSort), o que leva a maiores ordens de complexidade (no caso do QuickSort, o caso médio é O(n log n) e o pior caso $O(n^2)$).

Portanto, temos que, mesmo com a limitação de apenas um valor na ordenação decrescente, a utilização da função MAX() para retornar o maior valor da tabela é a opção mais veloz.

Descrição de atividades desenvolvidas

O presente tópico fará uma descrição breve das atividades desempenhadas por cada um dos participantes do grupo. Caso uma atividade tenha sido realizada por mais de um participante, esta constará na lista de atividades de ambos.

Breno Poggiali de Sousa

- Back end do desenvolvimento do sistema web;
- Integração do Back-End com o Front-End em Flask e Angular, respectivamente;
- Criação de rotas passando um JSON com o retorno das consultas SQL para o Front-End;
- Criação do repositório conjunto e estruturação de arquivos para o trabalho;

- Criação de páginas e componentes de login e register no Front-End;
- Auxílio no CSS de componentes no Front-End.

Gabriela Peres Neme

- Acertos e correções do esquema do banco de dados;
- Criação das 10 consultas em SQL, acompanhadas de descrição textual, o comando SQL e resultado;
- Criação das três consultas SQL do tipo relatório;
- Avaliação de eficiência das consultas;
- Documentação, incluindo formatação, conversão do diagrama ER para o formato digital, esquema relacional, introdução, conclusão e transcrição das consultas criadas e da avaliação de eficiência.

Guilherme de Abreu Lima Buitrago Miranda

- Criação do esquema do banco de dados;
- Inserção de dados não convencionais;
- Preenchimento do banco de dados com dados do Campeonato Brasileiro Série A;
- Criação de scripts Python para normalização dos dados obtidos do site cartola;
- Avaliação de eficiência das consultas.

Vinicius Julião Ramos

- Elaboração do diagrama ER do banco de dados;
- Definição das telas do programa;
- *Front end* do desenvolvimento web;
- Integração do Back-End com o Front-End em Flask e Angular, respectivamente;
- Criação do quadro, no aplicativo Trello, para definir a distribuição de tarefas entre os integrantes do grupo e os respectivos prazos;
- Manipulação da rotina de login integrada ao Firebase.

Conclusão

Através do presente trabalho foi possível aplicar os conceitos aprendidos na disciplina de Introdução a Banco de Dados, aprofundar os conhecimentos adquiridos e verificar eventuais

diferenças entre os aspectos conceituais e práticos, no que tange a implementação e estruturação de um banco de dados. Ademais, foi possível extrapolar o conteúdo da disciplina, ao criar um gerenciador de peladas com interface gráfica web e ao fazer análise de desempenho das consultas, que requer conteúdo de estatística e ciência de dados.

Desta maneira, o objetivo proposto foi atingido com sucesso, resultando em um gerenciador de peladas completo e com utilidade prática para os eventuais usuários.