Atividade - Indexação - Banco de Dados II
Aluno: Luiz Ricardo Brumati De Lima
RA: 2155184

I. Exercício 1

Crie as constraints que julgar necessário após as tabelas forem criadas. Popule as tabelas com várias tuplas, de preferêcia acima de mil tuplas, até que os índices sejam usados em suas consultas. Não se esqueça de atualizar as métricas das tabelas após a inserção (analyze).

Código para criação das tabelas:

```
CREATE TABLE Aluno(
    Nome VARCHAR(50) NOT NULL,
    RA DECIMAL(8) NOT NULL,
    DataNasc DATE NOT NULL,
    Idade DECIMAL(3),
    NomeMae VARCHAR(50) NOT NULL,
    Cidade VARCHAR(30),
    Estado CHAR(2),
    Curso VARCHAR(50),
    periodo integer
);

CREATE TABLE Discip(
    Sigla CHAR(7) NOT NULL,
    Nome VARCHAR(25) NOT NULL,
    SiglaPreReq CHAR(7),
    NNCred DECIMAL(2) NOT NULL,
    Monitor DECIMAL(8),
    Depto CHAR(8)
);

CREATE TABLE Matricula(
    RA DECIMAL(8) NOT NULL,
    Sigla CHAR(7) NOT NULL,
    Ano CHAR(4) NOT NULL,
    Semestre CHAR(1) NOT NULL,
    CodTurma DECIMAL(4) NOT NULL,
    NotaP1 NUMERIC(3,1),
    NotaP2 NUMERIC(3,1),
    NotaTrab NUMERIC(3,1),
    NotaFIM NUMERIC(3,1),
    Frequencia DECIMAL(3)
)
```

Definição das constraints:

```
-- Primary Keys:

alter table aluno add constraint
ra_aluno_pk primary key (ra);

alter table discip add constraint
sigla_discip_pk primary key (sigla);

alter table matricula add constraint
ra_sigla_ano_semestre_matricula_pk
primary key (ra, sigla, ano,
semestre);

-- Foreign Keys:

alter table discip add constraint
monitor_discip_fk foreign key
(monitor) references aluno (ra);

alter table matricula add constraint
aluno_matricula_fk foreign key (ra)
references aluno (ra);

alter table matricula add constraint
matricula_discip_fk foreign key
(sigla) references discip (sigla);
```

Os códigos aqui apresentados estão disponíveis em *atividade_indexacao.sql.*

Cerca de 1000 tuplas foram geradas de forma randômica para cada tabela com o uso do código disponível em *gerar_dados.py*. Em seguida, os comandos de inserção criados foram transferidos para um único arquivo (*tables_population.sql*) e as tabelas foram populadas com uma única execução do script completo.

Após o devido preenchimento, as métricas dos dados recém inseridos foram atualizadas por meio dos comandos:

```
analyze aluno;
analyze discip;
analyze matricula;
```

II. EXERCÍCIO 2

Suponha que o seguinte índice foi criado para a relação Aluno:

```
CREATE UNIQUE INDEX IdxAlunoNNI ON
Aluno (Nome, NomeMae, Idade);
```

a. **Escreva uma consulta que utilize esse índice.**
b. **Mostre um exemplo onde o índice não é usado mesmo utilizando algum campo indexado na cláusula where.**

```
-- Consulta que utiza o índice criado:
explain analyze select nome, nomemae
from aluno where nome = 'Luiz' and
nomemae = 'Marcieli';
```

	QUERY PLAN text
1	Index Only Scan using idxalunonni on aluno (cost=0.28..8.29 rows=1 v
2	Index Cond: ((nome = 'Luiz'::text) AND (nomemae = 'Marcieli'::text))
3	Heap Fetches: 1
4	Planning Time: 0.066 ms
5	Execution Time: 0.040 ms

A busca foi realizada como *Index Only*, visto que a cláusula *where* especifica dois atributos já indexados.

```
-- Consulta que não utiliza o índice
criado:
explain analyze select nome, nomemae
from aluno where nomemae = 'Marcieli';
```

	QUERY PLAN text
1	Seq Scan on aluno (cost=0.00..24.50 rows=14 width=14) (actual time=
2	Filter: ((nomemae)::text = 'Marcieli'::text)
3	Rows Removed by Filter: 986
4	Planning Time: 0.097 ms
5	Execution Time: 0.249 ms

No segundo exemplo, o índice não é utilizado pois a busca não obedece a ordem de prioridade definida na criação do índice *(Nome > NomeMae > Idade)*. Dessa forma, é necessário que o nome do aluno também seja utilizado como critério de seleção na pesquisa.

III. EXERCÍCIO 3

Crie índices e mostre exemplos de consultas (resultados e explain) que usam os seguinte tipos de acessos:

a) Sequential Scan

b) Bitmap Index Scan

c) Index Scan

d) Index-Only Scan

e) Multi-Index Scan

O primeiro passo foi a criação dos índices utilizados em cada busca. No entanto, para a utilização da busca com *Bitmap Index,* foi necessária a criação da extensão adicional *btree_gin*.

```
create extension btree_gin;
create unique index IdxAlunoRaNome ON
Aluno (RA, nome);
create index IdxSiglaBitmap on
Matricula using gin (Sigla);
```

Para a busca sequencial, basta definir como parâmetro de busca qualquer campo não indexado:

```
-- a) Sequential Scan
explain analyze
select ra, nome, cidade from aluno
where cidade = 'Curitiba';
```

	QUERY PLAN text
1	Seq Scan on aluno (cost=0.00..24.50 rows=23 width=24) (actual time=
2	Filter: ((cidade)::text = 'Curitiba'::text)
3	Rows Removed by Filter: 977
4	Planning Time: 0.049 ms
5	Execution Time: 0.109 ms

Como o índice *IdxSiglaBitmap* foi criado com o método gin, o resultado é a utilização de um índice bitmap quando realizada uma busca indexada:

```
-- b) Bitmap Index Scan
```

```
explain analyze
select sigla from matricula where
sigla = 'ZCRAEPG';
```

	QUERY PLAN text
1	Bitmap Heap Scan on matricula (cost=8.01..12.02 rows=1 width=8) (a
2	Recheck Cond: (sigla = 'ZCRAEPG'::bpchar)
3	Heap Blocks: exact=2
4	-> Bitmap Index Scan on idxsiglabitmap (cost=0.00..8.01 rows=1 widt
5	Index Cond: (sigla = 'ZCRAEPG'::bpchar)
6	Planning Time: 0.081 ms
7	Execution Time: 0.041 ms

Para o uso de *Index Scan*, é necessário que o parâmetro da busca seja um atributo indexado. No entanto, também requere-se que a projeção seja realizada em pelo menos um atributo não indexado, já que, dessa forma, o banco de dados é obrigado a consultar a tabela de origem.

```
-- c) Index Scan
explain analyze
select ra, cidade from aluno where ra
= 72435763
```

	QUERY PLAN text
1	Index Scan using idxalunoranome on aluno (cost=0.28..8.29 rows=1 w
2	Index Cond: (ra = '72435763'::numeric)
3	Planning Time: 0.063 ms
4	Execution Time: 0.022 ms

De forma semelhantate ao item anterior, o *Index-Only Scan* é utilizado quando a projeção é realizada apenas sobre itens já indexados, eliminando a necessidade de consulta na tabela original.

```
-- d) Index-Only Scan
explain analyze
select ra, nome from aluno where ra =
72435763 and nome = 'Camila';
```

	QUERY PLAN text
1	Index Only Scan using idxalunoranome on aluno (cost=0.28..8.29 rows
2	Index Cond: ((ra = '72435763'::numeric) AND (nome = 'Camila'::text))
3	Heap Fetches: 1
4	Planning Time: 0.148 ms
5	Execution Time: 0.040 ms

Por fim, para o uso do *Multi-Index Scan*, basta realizar uma consulta que contenha, pelo menos, dois atributos indexados por índices diferentes.

```
-- e) Multi-Index Scan
explain analyze
select matricula.sigla, aluno.ra from
matricula, aluno
where matricula.sigla = 'ZCRAEPG' and
aluno.ra = 72435763;
```

	QUERY PLAN text
1	Nested Loop (cost=8.28..20.32 rows=1 width=15) (actual time=0.030.
2	-> Bitmap Heap Scan on matricula (cost=8.01..12.02 rows=1 width=8)
3	Recheck Cond: (sigla = 'ZCRAEPG'::bpchar)
4	Heap Blocks: exact=2
5	-> Bitmap Index Scan on idxsiglabitmap (cost=0.00..8.01 rows=1 widt
6	Index Cond: (sigla = 'ZCRAEPG'::bpchar)
7	-> Index Only Scan using idxalunoranome on aluno (cost=0.28..8.29 ro
8	Index Cond: (ra = '72435763'::numeric)
9	Heap Fetches: 2
10	Planning Time: 0.122 ms
11	Execution Time: 0.058 ms

IV. Exercício 4

Faça consultas com junções entre as tabelas e mostre o desempenho criando-se índices para cada chave estrangeira.

Ao realizar a operação do exemplo 1, é possível observar que, por meio da busca sequencial na tabela matrícula, o tempo de execução é prolongado.

```
-- Exemplo 1:
explain analyze
select nome, notaFIM
from matricula natural join discip
where sigla = 'ZYZZCDQ';
```

	QUERY PLAN text
1	Nested Loop (cost=0.28..32.83 rows=4 width=21) (actual time=0.025..
2	-> Index Scan using sigla_discip_pk on discip (cost=0.28..8.29 rows=1
3	Index Cond: (sigla = 'ZYZZCDQ'::bpchar)
4	-> Seq Scan on matricula (cost=0.00..24.50 rows=4 width=14) (actual
5	Filter: (sigla = 'ZYZZCDQ'::bpchar)
6	Rows Removed by Filter: 996
7	Planning Time: 0.085 ms
8	Execution Time: 0.125 ms

No entanto, com a criação do índice *IdxMatriculaSiglaFK*, a busca sequencial é substituída por uma busca indexada que reduz o tempo de execução para 0.050ms.

```
create index IdxMatriculaSiglaFK on
Matricula (sigla);
```

	QUERY PLAN text
1	Nested Loop (cost=4.58..21.76 rows=4 width=21) (actual time=0.027..
2	-> Index Scan using sigla_discip_pk on discip (cost=0.28..8.29 rows=1
3	Index Cond: (sigla = 'ZYZZCDQ'::bpchar)
4	-> Bitmap Heap Scan on matricula (cost=4.31..13.43 rows=4 width=14
5	Recheck Cond: (sigla = 'ZYZZCDQ'::bpchar)
6	Heap Blocks: exact=2
7	-> Bitmap Index Scan on idxmatriculasiglafk (cost=0.00..4.31 rows=4
8	Index Cond: (sigla = 'ZYZZCDQ'::bpchar)
9	Planning Time: 0.091 ms
10	Execution Time: 0.050 ms

Ao mesmo modo, a criação do índice *IdxDiscipMonitorFK* reduz de maneira considerável o tempo de execução ao realizar a operação do exemplo 2, visto que este sai de 0.316ms para 0.165ms, caindo aproximadamente pela metade:

```
-- Exemplo 2:
explain analyze
select nome, sigla, periodo
from discip natural join aluno
where monitor = 46953510;
```

	QUERY PLAN text
1	Hash Join (cost=23.51..49.27 rows=1 width=44) (actual time=0.302..(
2	Hash Cond: ((aluno.nome)::text = (discip.nome)::text)
3	-> Seq Scan on aluno (cost=0.00..22.00 rows=1000 width=11) (actual
4	-> Hash (cost=23.50..23.50 rows=1 width=23) (actual time=0.160..0.1
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on discip (cost=0.00..23.50 rows=1 width=23) (actual tim
7	Filter: (monitor = '46953510'::numeric)
8	Rows Removed by Filter: 999
9	Planning Time: 0.305 ms
10	Execution Time: 0.316 ms

```
create index IdxDiscipMonitorFK on
Discip (monitor);
```

	QUERY PLAN text
1	Hash Join (cost=8.30..34.06 rows=1 width=44) (actual time=0.148..0.'
2	Hash Cond: ((aluno.nome)::text = (discip.nome)::text)
3	-> Seq Scan on aluno (cost=0.00..22.00 rows=1000 width=11) (actual '
4	-> Hash (cost=8.29..8.29 rows=1 width=23) (actual time=0.013..0.013
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Index Scan using idxdiscipmonitorfk on discip (cost=0.28..8.29 row
7	Index Cond: (monitor = '46953510'::numeric)
8	Planning Time: 0.326 ms
9	Execution Time: 0.165 ms

V. Exercício 5

Utilize um índice bitmap para período e mostre-o em uso nas consultas.

A seguinte consulta foi realizada sem a criação de nenhum índice específico:

```
explain analyze
select periodo from aluno
where periodo = 9;
```

	QUERY PLAN text
1	Seq Scan on aluno (cost=0.00..24.50 rows=126 width=4) (actual time=
2	Filter: (periodo = 9)
3	Rows Removed by Filter: 874
4	Planning Time: 0.055 ms
5	Execution Time: 0.135 ms

Em seguidam, criou-se o índice *idxAlunoPeriodo*. Como já explicado previamente, o índice bitmap requer a criação da extensão *btree_gin*, já realizada.

```
create index IdxAlunoPeriodo on Aluno
using gin (periodo);
```

	QUERY PLAN text
1	Bitmap Heap Scan on aluno (cost=8.98..22.55 rows=126 width=4) (act
2	Recheck Cond: (periodo = 9)
3	Heap Blocks: exact=12
4	-> Bitmap Index Scan on idxalunoperiodo (cost=0.00..8.95 rows=126 w
5	Index Cond: (periodo = 9)
6	Planning Time: 0.063 ms
7	Execution Time: 0.067 ms

Com o uso do índice, é possível notar uma pequena redução no tempo de planejamento e uma alteração expressiva no tempo de execução.

VI. Exercício 6

Compare na prática o custo de executar uma consulta com e sem um índice clusterizado na tabela aluno. Ou seja, faça uma consulta sobre algum dado indexado, clusterize a tabela naquele índice e refaça a consulta. Mostre os comandos e os resultados do explain analyze.

A princípio, foi criado o índice *IdxAlunoNomeIdade*:

```
create index IdxAlunoNomeIdade ON
Aluno (nome, idade);
```

Ao realizar a operação abaixo, o seguinte resultado foi obtido:

```
explain analyze
select nome, idade from aluno
where nome = 'Luiz';
```

	QUERY PLAN text
1	Bitmap Heap Scan on aluno (cost=4.35..16.88 rows=10 width=12) (act
2	Recheck Cond: ((nome)::text = 'Luiz'::text)
3	Heap Blocks: exact=7
4	-> Bitmap Index Scan on idxalunonomeidade (cost=0.00..4.35 rows=1(
5	Index Cond: ((nome)::text = 'Luiz'::text)
6	Planning Time: 0.069 ms
7	Execution Time: 0.042 ms

Em seguida, a criação do cluster reordenou a tabela fisicamente com base no índice especificado. Ao repetir a operação anterior, os seguintes resultados foram obtidos:

```
cluster aluno using IdxAlunoNomeIdade;
```

	QUERY PLAN text
1	Bitmap Heap Scan on aluno (cost=4.35..16.88 rows=10 width=12) (act
2	Recheck Cond: ((nome)::text = 'Luiz'::text)
3	Heap Blocks: exact=1
4	-> Bitmap Index Scan on idxalunonomeidade (cost=0.00..4.35 rows=1(
5	Index Cond: ((nome)::text = 'Luiz'::text)
6	Planning Time: 0.086 ms
7	Execution Time: 0.028 ms

VII. Exercício 7

Acrescente um campo adicional na tabela de Aluno, chamado de *informacoesExtras*, do tipo JSON. Insira dados diferentes telefônicos e de times de futebol que o aluno torce para cada aluno neste JSON. Crie índices para o JSON e mostre consultas que o utiliza (explain analyze). *Exemplo: retorne os alunos que torcem para o Internacional.*

Para permitir a inserção dos novos dados na tabela, a coluna adicional *informacoesExtras* foi criada:

```
alter table aluno
add column informacoesExtras jsonb;
```

Em seguida, o código abaixo foi desenvolvido com a própria linguagem estrutural do SQL para gerar de forma randômica os novos dados:

```
do $$
declare
  table_row aluno%rowtype;
  j numeric;
  times varchar[] =
'{"Chapecoense","Palmeiras","São
Paulo","Internacional", "Flamengo",
"Santos","Atlético-MG","Fluminense",
"Ceará", "Corinthians", "Botafogo"}';
begin
  for table_row in select * from aluno
  loop
    j = round(random() * 10)+1;
    update aluno set informacoesExtras
=
    (('{
        "telefone" : ' ||
round(random()*100000000)||',
        "time" : "'|| times[j] ||'"
        }')::json)
    where ra = table_row.ra;
  end loop;
end $$
language plpgsql;
```

Com os campos devidamente populados, a análise de desempenho foi realizada:

```
-- Ex. 1 de utilização de índices com JSON:

explain analyze
select nome, informacoesExtras from aluno
where informacoesExtras->>'time' = 'Internacional';
```

	QUERY PLAN text
1	Seq Scan on aluno (cost=0.00..27.00 rows=5 width=39) (actual time=(
2	Filter: ((informacoesextras ->> 'time'::text) = 'Internacional'::text)
3	Rows Removed by Filter: 1000
4	Planning Time: 0.552 ms
5	Execution Time: 0.111 ms

Em seguida, foi criado o índice *idxAlunoJSONTime* e a operação repetida:

```
create index idxAlunoJSONTime
on aluno using BTREE
((informacoesExtras->>'time'));
```

	QUERY PLAN text
1	Bitmap Heap Scan on aluno (cost=4.19..14.58 rows=5 width=39) (actu
2	Recheck Cond: ((informacoesextras ->> 'time'::text) = 'Internacional'::te
3	-> Bitmap Index Scan on idxalunojsontime (cost=0.00..4.19 rows=5 wi
4	Index Cond: ((informacoesextras ->> 'time'::text) = 'Internacional'::text)
5	Planning Time: 0.071 ms
6	Execution Time: 0.024 ms

É possível visualizar a diferença notória de desempenho com a presença do índice, visto que o tempo de planejamento foi reduzido de 0.552ms para 0.071ms. Ao mesmo passo, o tempo de execução foi de 0.111ms para 0.024ms.

O mesmo procedimento foi realizado para o exemplo 2:

```
-- Ex. 2 de utilização de índices com JSON:

explain analyze
select nome, informacoesExtras from aluno
where informacoesExtras->>'telefone' = '84581151';
```

	QUERY PLAN text
1	Seq Scan on aluno (cost=0.00..27.00 rows=5 width=39) (actual time=(
2	Filter: ((informacoesextras ->> 'telefone'::text) = '84581151'::text)
3	Rows Removed by Filter: 1000
4	Planning Time: 0.045 ms
5	Execution Time: 0.115 ms

```
create index idxJSON on aluno using gin
((informacoesExtras->>'telefone'));
```

	QUERY PLAN text
1	Bitmap Heap Scan on aluno (cost=8.04..18.43 rows=5 width=39) (actu
2	Recheck Cond: ((informacoesextras ->> 'telefone'::text) = '84581151'::t
3	-> Bitmap Index Scan on idxjson (cost=0.00..8.04 rows=5 width=0) (ac
4	Index Cond: ((informacoesextras ->> 'telefone'::text) = '84581151'::text
5	Planning Time: 0.064 ms
6	Execution Time: 0.023 ms

Vale destacar que, embora não se note uma grande diferença prática para operações com duração na casa de milissegundos, tais procedimentos tem a capacidade de reduzir grandes períodos de processamento em bancos com uma população mais expressiva.

Uma diferença menor que 1 segundo em uma tabela com mil tuplas pode resultar em uma economia de horas para bancos com milhões de dados armazenados. Portanto, o impacto positivo do uso correto da indexação jamais deve ser descartado.