

Projeto de Bases de Dados

Parte 4

Grupo 19 – Turno BD225179L09

Prof. Gabriel Pestana

81082 – Nuno Gonçalves (12 horas)

81205 – Alice Dourado (12 horas)

81500 – Rodrigo Rato (12 horas)

1. Índices

Query #1 - A)

Índices na tabela **arrenda**:

- Índice do tipo **HASH** para a coluna **morada**;
- Índice do tipo **HASH** para a coluna **codigo**;
- Índice do tipo **BTREE** para a coluna **nif**;

Índices na tabela **fiscaliza**:

- Índice do tipo **HASH** para a coluna **morada**;
- Índice do tipo **HASH** para a coluna **codigo**;

Os índices hash são colocados devido à comparação **a.morada = f.morada and a.codigo = f.codigo**, pois este tipo de índices são mais eficientes para operações de verificação de igualdade entre valores.

O índice BTREE sobre nif na tabela arrenda deve-se ao **group by a.nif**, que fica mais rápido ao percorrer a tabela por ordem, algo que é possível quando se tem este tipo de índice mas que é impossível de fazer num índice em hash onde determinar uma ordem de iteração da hash table é impossível.

Query #1 - B)

Para criar o índice BTREE do arrenda sobre a coluna nif usou-se a seguinte instrução:

```
CREATE INDEX nif_idx ON arrenda(nif) USING BTREE;
```

Os índices HASH seriam criados com as seguintes instruções:

```
CREATE INDEX morada_idx ON arrenda(morada) USING HASH;
```

```
CREATE INDEX codigo_idx ON arrenda(codigo) USING HASH;
```

```
CREATE INDEX morada_idx ON fiscaliza(morada) USING HASH;
```

```
CREATE INDEX codigo_idx ON fiscaliza(codigo) USING HASH;
```

No entanto devido a limitações do *software* MySQL não é possível criar índices hash uma vez que o storage engine usado nestas tabelas não suporta este tipo de índices. Também não é necessário criar o índice `nif_idx` pois está associado a uma primary key da tabela logo é criado um índice automaticamente.

O plano de execução desta query foi o seguinte:

id	select_type	table	type	possible_keys	key	key_len
1	1	f	index	morada	morada	514
2	1	a	eq_ref	PRIMARY	PRIMARY	514

ref	rows	Extra
1 <null>	9	Using index; Using temporary; Using filesort
2 ist181205.f.morada, ist181205.f.codigo	1	

É possível observar que o índice btree que criámos não foi usado neste caso porque no resultado obtido há apenas um `nif` distinto com o conteúdo que está nas tabelas, mas possivelmente com um conteúdo diferente seria usado se fosse obtida uma maior quantidade de `nifs` a partir da query.

A primeira linha mostra que é usado um índice chamado `morada` que foi criado automaticamente na tabela `fiscaliza`, se fosse possível criar os índices hash que indicámos esses seriam usados em vez deste.

Query #2 – A)

Índice na tabela **estado**:

- Índice do tipo **HASH** para a coluna `estado`;

Índice na tabela **posto**:

- Índice do tipo **HASH** para as colunas `morada`, `codigo_espaco`;
- Índice do tipo **HASH** para as colunas `morada`, `codigo`;

Índice na tabela **aluga**:

- Índice do tipo **HASH** para a coluna `numero`;

O índice do estado é criado porque, na query, é usada no **where** a condição `e.estado = 'aceite'` pelo que um índice do tipo hash será a melhor escolha para este caso pois este tipo de índice é mais eficiente para operações de verificação de igualdade entre valores.

O índice da tabela `posto` é devido ao facto de, na cláusula **where** da query efetuada, é usada a condição `(p.morada, p.codigo_espaco) not in (...)`, o que implica que o valor irá ser comparado com todos os

valores da tabela em segundo argumento e como isto é uma operação de verificação de igualdade o melhor índice para a situação é do tipo hash.

Os restantes índices são utilizados para melhorar o desempenho do **natural join** dentro da cláusula **not in**, que é efetuado através de comparações de valores de colunas com o mesmo nome, algo que é melhorado pelos hash indexes. Só é criado um índice numa das tabelas que possuem colunas correspondentes pois o **natural join** irá sempre percorrer uma das tabelas para verificar se existe algum correspondente.

Query #2 – B)

Para criar se criarem os índices usaram-se as seguintes instruções:

```
CREATE INDEX estado_idx ON estado(estado) USING HASH;
```

```
CREATE INDEX posto_idx1 ON posto(morada, codigo_espaco) USING HASH;
```

```
CREATE INDEX posto_idx2 ON posto(morada, codigo) USING HASH;
```

```
CREATE INDEX aluga_idx ON aluga(numero) USING HASH;
```

Não é possível criar estes índices em MySQL devido às limitações mencionadas anteriormente.

O plano de execução desta query foi o seguinte:

	id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	1	PRIMARY	p	index	<null>	morada	514	<null>	9	Using where; Using index
2	2	DEPENDENT SUBQUERY	p	ref	PRIMARY, morada	PRIMARY	257	func	1	Using where
3	2	DEPENDENT SUBQUERY	a	ref	PRIMARY, numero	PRIMARY	514	ist181205.p.morada, ist181205.p.codigo	1	Using index
4	2	DEPENDENT SUBQUERY	e	ref	PRIMARY	PRIMARY	257	ist181205.a.numero	1	Using where

Para além dos índices das primary keys o outro índice usado é o índice ‘morada’ que foi criado automaticamente na tabela posto, mas se fosse possível criar/usar índices hash o nosso índice iria ser escolhido em detrimento do índice predefinido em btree.

2. Data Warehouse

Schema para a base de dados OLAP:

```
drop table if exists reserva_estrela;  
drop table if exists tempo_dim;  
drop table if exists local_dim;  
drop table if exists user_dim;  
drop table if exists data_dim;
```

```
create table tempo_dim(  
    tempo int not null unique,  
    hora int not null,  
    minuto int not null,  
  
    primary key(tempo)  
);
```

```
create table data_dim(  
    data int not null unique,  
    dia int not null,  
    semana int not null,  
    mes int not null,  
    semestre int not null,  
    ano int not null,  
    primary key(data)  
);
```

```
create table local_dim(  
    morada_codigo varchar(510) not null,  
    edificio varchar(255) not null,  
    espaco varchar(255) not null,  
    posto varchar (255),  
    primary key(morada_codigo)  
);
```

```
create table local_dim(  
    morada_codigo varchar(510) not null,  
    edificio varchar(255) not null,  
    espaco varchar(255) not null,  
    posto varchar (255),  
    primary key(morada_codigo)  
);
```

```
create table user_dim(  
    nif varchar(9) not null unique,  
    nome varchar(80) not null,  
    telefone varchar(26) not null,  
    primary key(nif)  
);
```

```
create table reserva_estrela(  
    numero varchar(255) not null unique,  
    montante_pago numeric(19,4) not null,  
    duracao_dias int not null,
```

```
    nif varchar(9) not null,  
    morada_codigo varchar(510) not null,  
    tempo int not null,  
    data int not null,
```

```
    foreign key(nif) references user_dim(nif),  
    foreign key(morada_codigo) references  
local_dim(morada_codigo),  
    foreign key(tempo) references tempo_dim(tempo),  
    foreign key(data) references data_dim(data),  
    primary key(numero, nif, morada_codigo, tempo, data)  
);
```

Query OLAP:

```
SELECT espaco, posto, dia, mes, AVG(montante_pago) AS
AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim
NATURAL JOIN local_dim
GROUP BY espaco, posto, data
UNION
SELECT espaco, posto, dia, NULL, AVG(montante_pago)
AS AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim
NATURAL JOIN local_dim
GROUP BY espaco, posto, dia
UNION
SELECT espaco, posto, NULL, mes, AVG(montante_pago)
AS AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim
NATURAL JOIN local_dim
GROUP BY espaco, posto, mes
UNION
SELECT espaco, NULL, dia, mes, AVG(montante_pago)
AS AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim
NATURAL JOIN local_dim
GROUP BY espaco, data
UNION
SELECT NULL, posto, dia, mes, AVG(montante_pago) AS
AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim
NATURAL JOIN local_dim
WHERE posto IS NOT NULL
GROUP BY posto, data
UNION
SELECT espaco, posto, NULL, NULL,
AVG(montante_pago) AS AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim
NATURAL JOIN local_dim
GROUP BY espaco, posto
UNION
SELECT NULL, NULL, dia, mes, AVG(montante_pago) AS
AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim
NATURAL JOIN local_dim
GROUP BY (data)
UNION
SELECT espaco, NULL, dia, NULL, AVG(montante_pago)
AS AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim
NATURAL JOIN local_dim
GROUP BY espaco, dia
UNION
```

```
SELECT espaco, NULL, NULL, mes, AVG(montante_pago) AS
AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim NATURAL
JOIN local_dim
GROUP BY espaco, mes
UNION
SELECT NULL, posto, NULL, mes, AVG(montante_pago) AS
AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim NATURAL
JOIN local_dim
WHERE posto IS NOT NULL
GROUP BY posto, mes
UNION
SELECT NULL, posto, dia, NULL, AVG(montante_pago) AS
AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim NATURAL
JOIN local_dim
WHERE posto IS NOT NULL
GROUP BY posto, dia
UNION
SELECT espaco, NULL, NULL, NULL, AVG(montante_pago)
AS AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim NATURAL
JOIN local_dim
GROUP BY espaco
UNION
SELECT NULL, posto, NULL, NULL, AVG(montante_pago)
AS AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim NATURAL
JOIN local_dim
WHERE posto IS NOT NULL
GROUP BY posto
UNION
SELECT NULL, NULL, NULL, mes, AVG(montante_pago) AS
AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim NATURAL
JOIN local_dim
GROUP BY mes
UNION
SELECT NULL, NULL, dia, NULL, AVG(montante_pago) AS
AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim NATURAL
JOIN local_dim
GROUP BY dia
UNION
SELECT NULL, NULL, NULL, NULL, AVG(montante_pago)
AS AVG_MONTANTE_PAGO
FROM reserva_estrela NATURAL JOIN data_dim NATURAL
JOIN local_dim;
```

Populate das tabelas do esquema em estrela:

```
delimiter //
drop procedure if exists buildtimedimension//
create procedure buildtimedimension ()
begin
  declare begin_time TIME;
  delete from tempo_dim;
  set begin_time = '00:00:00';
  while begin_time <= '23:59:00' do
    insert into
tempo_dim(`tempo`,`hora`,`minuto`)
      values (HOUR(begin_time)*10000 +
MINUTE(begin_time)*100,HOUR(begin_time),
MINUTE(begin_time));
    set begin_time = ADDTIME(begin_time,
'0:1:0.0');
  end while;
end//
delimiter ;

delimiter //
drop procedure if exists builddatedimension//
create procedure builddatedimension ()
begin
  declare begin_date DATE;
  delete from data_dim;
  set begin_date = '2016-1-1';
  while begin_date <= '2017-12-31' do
    insert into
data_dim(`data`,`dia`,`semana`,`mes`,`semestre`,`ano`)
      values (YEAR(begin_date) * 10000 +
MONTH(begin_date)*100 +
DAY(begin_date),DAY(begin_date),WEEK(begin
_date),MONTH(begin_date),IF(MONTH(begin_
date) < 7, 1, 2),YEAR(begin_date));
    set begin_date = DATE_ADD(begin_date,
INTERVAL 1 DAY);
  end while;
end//
delimiter ;
```

```
call buildtimedimension();

call builddatedimension();

INSERT INTO user_dim SELECT * from
user;

INSERT INTO
local_dim(`morada_codigo`,`edificio`,`espac
o`,`posto`)
SELECT
CONCAT(morada,codigo),morada,codigo_es
paco,codigo
FROM posto;

INSERT INTO
local_dim(`morada_codigo`,`edificio`,`espac
o`)
SELECT
CONCAT(morada,codigo),morada,codigo
FROM espaco;

INSERT INTO
reserva_estrela(`numero`,`nif`,`duracao_dias`
,`montante_pago`,`morada_codigo`,`data`,`te
mpo`)
SELECT
numero,nif,datediff(data_fim,data_inicio),dat
ediff(data_fim,data_inicio)*tarifa,
concat(morada,codigo),YEAR(data) * 10000
+ MONTH(data)*100 + DAY(data),
HOUR(data)*10000 + MINUTE(data)*100
FROM aluga NATURAL JOIN oferta
NATURAL JOIN paga;
```