

Projecto de Base de Dados

Instituto Superior Técnico

2016/2017

4ª Parte

Gonçalo Rodrigo, nº 81543 esforço semanal: 10 horas
Filipe Azevedo, nº 82468 esforço semanal: 10 horas
Martim Zanatti, nº 82517 esforço semanal: 10 horas
Número do grupo: AL42

Turno: BD225179L07

Índices

1.

Interrogação 1: Quais utilizadores cujos espaços foram fiscalizados sempre pelo mesmo fiscal?

Query 1:

```
SELECT A.nif
FROM Arrenda A
      INNER JOIN Fiscaliza F
      ON A.morada = F.morada
      AND A.codigo = F.codigo
GROUP BY A.nif
HAVING COUNT(DISTINCT F.id) = 1;
```

a)

Faz sentido fazer um índice com **chave de procura composta** sobre <morada, código> sobre a tabela Fiscaliza, visto que na *query* utilizamos um *inner join* onde a morada e código da tabela Arrenda é igual aos mesmos atributos na tabela Fiscaliza, desta forma são filtrados os espaços que foram fiscalizados. Não criámos o índice composto sobre morada e código da tabela arrenda, pois este já é criado automaticamente quando a própria tabela é criada, pois <morada,código> é chave primária da tabela de Arrenda, ou contrário da tabela Fiscaliza.

Como morada e código não é chave primária de Fiscaliza o índice criado sobre esta tabela é **secundário**, logo este último índice muito provavelmente contém duplicados, isto é existem entradas de dados no índice que tem o mesmo valor para a chave de pesquisa associado ao índice (Ex: morada e código igual em Fiscaliza com id (Fiscal) diferente.)

Como fiscaliza-se entre o Fiscal e um Alugável arrendado por um User, todos os conjuntos <morada,código> de Fiscaliza estão na tabela Arrenda, por outro lado nem todos os conjuntos <morada,código> de Arrenda estão na tabela Fiscaliza. (Ex: Um Alugável que foi arrendado mas ainda não foi fiscalizado.) Por este motivo, o índice da tabela Fiscaliza é **denso** e **não agrupado**, pois utilizamos uma hash como dispersão.

O índice de **dispersão dinâmica** (*dynamic hashing*), visto que estamos a fazer um inner join onde são filtrados os valores com base na igualdade entre a morada e código entre a tabela Arrenda e o Fiscaliza, em caso de igualdades a Hash encontra em tempo constante $O(1)$ enquanto a Btree encontra em $O(\log(n))$.

Achamos também que ter um índice de dispersão dinâmica sobre o nif na tabela reserva poderia ajudar a realizar o group by. No entanto não temos a certeza e portanto não foi implementado no trabalho final.

b)

-Tabela Fiscaliza sem Índice:

```
-----+
| fiscaliza | CREATE TABLE `fiscaliza` (
| `id` int(11) NOT NULL,
| `morada` varchar(255) NOT NULL,
| `codigo` varchar(255) NOT NULL,
| PRIMARY KEY (`id`,`morada`,`codigo`),
| CONSTRAINT `fiscaliza_ibfk_1` FOREIGN KEY (`morada`,`codigo`) REFERENCES `arrenda` (`morada`,`codigo`),
| CONSTRAINT `fiscaliza_ibfk_2` FOREIGN KEY (`id`) REFERENCES `fiscal` (`id`)
| ) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+
```

-Query antes de criar índice:

```
mysql> show profiles;
+-----+
| Query_ID | Duration | Query |
+-----+
| 1 | 0.06146200 | select a.nif from arrenda a inner join fiscaliza f on a.morada=f.morada and a.codigo=f.codigo group by a.nif having count(f.id)=1 |
| 2 | 0.00022400 | select a.nif from arrenda a inner join fiscaliza f on a.morada=f.morada and a.codigo=f.codigo group by a.nif having count(f.id)=1 |
+-----+
2 rows in set (0.00 sec)
```

Primeira consulta, não
está na cache

-Criação do índice:

```
CREATE INDEX Index_fiscaliza
ON Fiscaliza(morada, código)
USING BTREE;
```

Segunda consulta, com dados
situados na cache

-Query apos criação de índice:

```
mysql> show profiles;
+-----+
| Query_ID | Duration | Query |
+-----+
| 1 | 0.06146200 | select a.nif from arrenda a inner join fiscaliza f on a.morada=f.morada and a.codigo=f.codigo group by a.nif having count(f.id)=1 |
| 2 | 0.00022400 | select a.nif from arrenda a inner join fiscaliza f on a.morada=f.morada and a.codigo=f.codigo group by a.nif having count(f.id)=1 |
| 3 | 0.08333100 | create index index_fiscaliza on fiscaliza(morada,codigo) using btree |
| 4 | 0.00050600 | create index index_fiscaliza on fiscaliza(morada,codigo) using btree |
| 5 | 0.02922500 | select a.nif from arrenda a inner join fiscaliza f on a.morada=f.morada and a.codigo=f.codigo group by a.nif having count(f.id)=1 |
| 6 | 0.00012700 | select a.nif from arrenda a inner join fiscaliza f on a.morada=f.morada and a.codigo=f.codigo group by a.nif having count(f.id)=1 |
+-----+
```

0.000127 s < 0.000224 s

2.

Interrogação 2: Quais os espaços com postos que nunca foram alugados?

Query 2:

```
SELECT DISTINCT P.morada, P.codigo_espaco
FROM Posto P
WHERE (P.morada, P.codigo_espaco) NOT IN (
    SELECT P.morada, P.codigo_espaco
    FROM posto P
    NATURAL JOIN aluga A
    NATURAL JOIN estado E
    WHERE E.estado = "aceite");
```

a)

Neste caso criamos um índice com **chave de procura simples** sobre o atributo <estado> sobre a tabela Estado. Também criamos um índice com **chave de procura composta** sobre o atributo <morada, código espaço> sobre a tabela Posto, pois desejamos espaços com postos que nunca foram alugados, ou seja, todos os postos que foram alugados e o seu estado não é aceite.

Como estado não é uma chave primária da Tabela Estado e morada e código_espaço os índices criados sobre estas tabelas é **secundário**, logo provavelmente estes índices contêm duplicados, isto é, existem entradas de dados no índice que tem o mesmo valor para a chave de pesquisa associado ao índice (Ex: Pode haver vários alugáveis com o mesmo estado, na verdade os estados possíveis são “Pendente”, “Aceite”, “Declinada” ou “Cancelada”; Diferentes postos no mesmo espaço e edifício).

Como o estado na tabela é reduzido e limitado (apenas 4 possibilidades), cada número de Reserva possui um dos quatro estados, logo este índice é **esparso**, logo **agrupado**. O índice no Posto são igualmente esparso e **agrupado**.

O índice Estado é um índice **bitmap**, visto que os valores das chaves de pesquisa (estado) são segmentados para formar conjunto de critérios booleanos, isto é cada valor distinto do atributo (“Pendente”, “Aceite”, “Declinada” ou “Cancelada”) tem um bitmap, um número binário que identifica esse estado, neste caso há 4 bitmaps diferentes. O índice da tabela Posto utiliza Hash pois verificamos os postos que não estão dentro de um conjunto de postos que foram alugados, ou seja uma igualdade entre os dos conjuntos, esta indexação encontra o valor em tempo constante $O(1)$ enquanto a Btree encontra em $O(\log(n))$.

b)

-Tabela estado antes da criação do índice:

```
posto | CREATE TABLE `posto` (  
  `morada` varchar(255) NOT NULL,  
  `codigo` varchar(255) NOT NULL,  
  `codigo_espaço` varchar(255) NOT NULL,  
  PRIMARY KEY (`morada`,`codigo`),  
  KEY `morada` (`morada`,`codigo_espaço`),  
  CONSTRAINT `posto_ibfk_1` FOREIGN KEY (`morada`,`codigo`) REFERENCES `alugavel` (`morada`,`codigo`),  
  CONSTRAINT `posto_ibfk_2` FOREIGN KEY (`morada`,`codigo_espaço`) REFERENCES `espaco` (`morada`,`codigo`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
```

```
-----+  
estado | CREATE TABLE `estado` (  
  `numero` varchar(255) NOT NULL,  
  `time_stamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
  `estado` varchar(255) NOT NULL,  
  PRIMARY KEY (`numero`,`time_stamp`),  
  CONSTRAINT `estado_ibfk_1` FOREIGN KEY (`numero`) REFERENCES `reserva` (`numero`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
```

-Query antes da criar o índice:

```
-----+  
Query_ID | Duration | Query  
-----+-----+-----  
9 | 0.00022500 | set profiling_history_size =100  
10 | 0.00025400 | set profiling =1  
11 | 0.00024300 | Select p.morada,p.codigo_espaço from posto p where (p.morada, p.codigo_espaço) not in (select p.morada, p.codigo_espaço from posto p natural join aluga a natural join estado e where e.estado='aceite') |  
12 | 0.00017400 | Select p.morada,p.codigo_espaço from posto p where (p.morada, p.codigo_espaço) not in (select p.morada, p.codigo_espaço from posto p natural join aluga a natural join estado e where e.estado='aceite') |
```

-Criação do índice:

```
CREATE INDEX Index_estado  
ON Estado(estado)  
USING BTREE;
```

```
CREATE INDEX index_morada_codigoespaco  
ON Posto (morada,código_espaço)  
USING BTREE;
```

-Query depois da criação dos índices:

0.000137 s < 0.000174 s

```
0.000993800 | Select p.morada,p.codigo_espaço from posto p where (p.morada, p.codigo_espaço) not in (select p.morada, p.codigo_espaço from posto p natural join aluga a natural join estado e where e.estado='aceite') |  
0.00013700 | Select p.morada,p.codigo_espaço from posto p where (p.morada, p.codigo_espaço) not in (select p.morada, p.codigo_espaço from posto p natural join aluga a natural join estado e where e.estado='aceite') |
```

Data Warehouse

De seguida encontra-se o código para criar as tabelas da Data Warehouse. Criámos 5 novas tabelas: `dimensao_user`, `dimensao_tempo`, `dimensao_localizacao`, `dimensao_data` e `dimensao_reserva`. Cada uma destas tabelas guarda respetivamente a informação sobre os utilizadores do sistema (semelhante à tabela `user`), todos os minutos e todas as horas de um dia, a informação sobre as moradas, códigos do espaços e códigos dos postos existentes na base de dados, todas as datas entre o ano de 2016 e 2017 ("2016-01-01" – "2017-12-31"), e por fim a tabela `dimensao_reserva` correlaciona a informação proveniente das tabelas anteriores e a informação das tabelas da entrega anterior e guarda essa informação.

```
drop table if exists dimensao_reserva;
drop table if exists dimensao_user;
drop table if exists dimensao_localizacao;
drop table if exists dimensao_tempo;
drop table if exists dimensao_data;
DROP PROCEDURE IF EXISTS load_time_dim;
DROP PROCEDURE IF EXISTS load_date_dim;

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

create table dimensao_localizacao(
  local_id varchar(255) not null unique,
  morada varchar(255) not null,
  codigo_espaco varchar(255) not null,
  codigo varchar(255),
  primary key(local_id));

create table dimensao_tempo(
  tempo_id varchar(5) not null unique,
  hora varchar(3) not null,
  minutos varchar(3) not null,
  primary key(tempo_id));

create table dimensao_data(
  data_id varchar(20) not null unique,
  dia varchar(3) not null,
  semana varchar(3) not null,
  mes varchar(3) not null,
  semestre varchar(10) not null,
  ano varchar(5) not null,
  primary key(data_id));

create table dimensao_reserva(
  numero varchar(255) not null unique,
  montante numeric(19,4) not null,
  duracao numeric(19,0) not null,
  tarifa numeric(19,4) not null,
  nif varchar(9) not null,
  local_id varchar(255) not null,
  tempo_id varchar(5) not null,
  data_id varchar(20) not null,
  primary key(numero,nif, tempo_id, local_id, data_id),
  foreign key(nif) references dimensao_user(nif),
  foreign key(tempo_id) references dimensao_tempo(tempo_id),
  foreign key(local_id) references dimensao_localizacao(local_id),
  foreign key(data_id) references dimensao_data(data_id));
```

Apresentamos também as queries de inserção de valores nestas tabelas:

```

insert into dimensao_user(nif, nome, telefone) SELECT nif, nome, telefone FROM user;

insert into dimensao_localizacao(local_id, morada, codigo_espaco, codigo)
(
    SELECT concat(morada, "-", codigo_espaco) as local_id, morada, codigo_espaco, codigo
    from (SELECT morada, codigo as codigo_espaco, NULL as codigo from espaco) as E
);

insert into dimensao_localizacao(local_id, morada, codigo_espaco, codigo)
(
    SELECT concat(morada, "-", codigo_espaco, "-", codigo) as local_id, morada, codigo_espaco, codigo
    from (SELECT morada, codigo_espaco, codigo from posto) as P
);

delimiter //
CREATE PROCEDURE load_date_dim()
BEGIN
    DECLARE v_full_date DATE;
    DECLARE semester NUMERIC(1,0);
    SET v_full_date = '2016-01-01';
    SET semester = 1;
    WHILE v_full_date < '2018-01-01' DO
        IF month(v_full_date) > 6 THEN
            SET semester = 2;
        ELSE SET semester = 1;
        END IF;
        INSERT INTO dimensao_data(
            data_id,
            dia,
            semana,
            mes,
            semestre,
            ano
        ) VALUES (
            DATE_FORMAT(v_full_date, '%Y-%m-%d'),
            DAY(v_full_date),
            WEEK(v_full_date),
            MONTH(v_full_date),
            semester,
            YEAR(v_full_date)
        );
        SET v_full_date = DATE_ADD(v_full_date, INTERVAL 1 DAY);
    END WHILE;
END;
//

CREATE PROCEDURE load_time_dim()
BEGIN
    SET @v_full_time = '2015-01-01 00:00:00';
    WHILE (DAY(@v_full_time) < 2) DO
        INSERT INTO dimensao_tempo(
            tempo_id,
            hora,
            minutos
        ) VALUES (
            DATE_FORMAT(@v_full_time, '%H:%i'),
            hour(@v_full_time),
            minute(@v_full_time)
        );
        SET @v_full_time = DATE_ADD(@v_full_time, INTERVAL 1 MINUTE);
    END WHILE;
END; //

delimiter ;

call load_time_dim;

call load_date_dim;

insert into dimensao_reserva(numero, montante, duracao, tarifa, nif, local_id, tempo_id, data_id)
select
    numero,
    tarifa * datediff(data_fim,data_inicio) as montante,
    datediff(data_fim,data_inicio) as duracao,
    tarifa,
    nif,
    concat(morada, '-', codigo) as local_id,
    DATE_FORMAT(data, '%H:%i') as tempo_id,
    DATE_FORMAT(data, '%Y-%m-%d') as data_id
from
    reserva natural join
    paga natural join
    oferta natural join
    aluga natural join
    espaco
;

insert into dimensao_reserva(numero, montante, duracao, tarifa, nif, local_id, tempo_id, data_id)
select
    numero,
    tarifa * datediff(data_fim,data_inicio) as montante,
    datediff(data_fim,data_inicio) as duracao,
    tarifa,
    nif,
    concat(morada, '-', codigo_espaco, '-', codigo) as local_id,
    DATE_FORMAT(data, '%H:%i') as tempo_id,
    DATE_FORMAT(data, '%Y-%m-%d') as data_id
from
    reserva natural join
    paga natural join
    oferta natural join
    aluga natural join
    posto
;

```

Note que a inserção de valores nas tabelas dimensão_data e dimensão_tempo é feita de forma automática recorrendo a stored procedures.