Projeto Bases De Dados,

Parte 4

Realizado por:

Francisco Duarte, Nº 78682

João Silvestre, Nº80996

Gonçalo Soares, Nº81418

Grupo Nº7, Turno Segunda 8:00h

O esforço, em horas, despendido na realização

deste trabalho por aluno foi: 10 horas

Índices

A)

Query 1:

Para esta query faria sentido criar dois índices:

1. Índice composto sobre a tabela Arrenda com os atributos morada e código.
2. Índice composto sobre a tabela Fiscaliza com os atributos morada e código.

Criamos estes índices pois os atributos morada e código são acedidos bastantes vezes durante a execução da query. Estes acessos são feitos para comparar as moradas e códigos das tabelas Arrenda e Fiscaliza, e para este tipo de comparações, o tipo ideal para o índice seria Hash. No entanto, este tipo não é possível implementar na versão MySQL utilizada na cadeira, sendo que os índices têm então de ser do tipo BTree.

Como comprovativo é usada a função explain do mysql:

```
mysql> explain 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;
+----+-------------+-------+--------+---------------+---------+---------+-----------------------------------+------+-----------------------------------------------+
| id | select_type | table | type   | possible_keys | key     | key_len | ref                               | rows | Extra                                         |
+----+-------------+-------+--------+---------------+---------+---------+-----------------------------------+------+-----------------------------------------------+
|  1 | SIMPLE      | F     | index  | morada        | morada  | 514     | NULL                              |    9 | Using index; Using temporary; Using filesort |
|  1 | SIMPLE      | A     | eq_ref | PRIMARY       | PRIMARY | 514     | ist180996.F.morada,ist180996.F.codigo |    1 |                                               |
+----+-------------+-------+--------+---------------+---------+---------+-----------------------------------+------+-----------------------------------------------+
2 rows in set (0.05 sec)
```

Query 2:

Para esta query faria sentido criar dois índices:

1. Um índice composto sobre a morada e codigo_espaco da tabela Posto.
2. Um índice sobre o estado da tabela Estado.

Como na alínea anterior, escolhemos os índices sobre estes atributos pois são os mais acedidos durante a execução da query. Para o segundo índice o tipo deveria ser Hash, pois é só usado para fazer comparações de igualdade, no entanto, como referido acima, isto não é possível implementar. Para o primeiro índice, o melhor tipo é BTree, e portanto ambos os índices ficam do tipo BTree.

Como comprovativo é usada a função explain do mysql:

```
mysql> explain 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 natu
ral join estado E where E.estado = 'aceite');
+----+--------------------+-------+--------+---------------+---------+---------+-----------------------+------+-----------------------------------------+
| id | select_type        | table | type   | possible_keys | key     | key_len | ref                   | rows | Extra                                   |
+----+--------------------+-------+--------+---------------+---------+---------+-----------------------+------+-----------------------------------------+
|  1 | PRIMARY            | P     | range  | NULL          | morada  | 514     | NULL                  |    3 | Using where; Using index for group-by |
|  2 | DEPENDENT SUBQUERY | A     | index  | PRIMARY,numero | nif     | 7       | NULL                  |   13 | Using index                             |
|  2 | DEPENDENT SUBQUERY | P     | eq_ref | PRIMARY,morada | PRIMARY | 514     | func,ist180996.A.codigo |    1 | Using where                             |
|  2 | DEPENDENT SUBQUERY | E     | ref    | PRIMARY       | PRIMARY | 257     | ist180996.A.numero    |    1 | Using where                             |
+----+--------------------+-------+--------+---------------+---------+---------+-----------------------+------+-----------------------------------------+
4 rows in set (0.00 sec)
```

B)

Query 1:

Nesta query não implementamos índices pois aqueles que pretendíamos criar são automaticamente criados pelo MySQL, que cria índices BTree sobre todas as chaves primárias das tabelas.

Os índices que queremos criar estão portanto já criados.

Após uma execução da query com e sem os índices, sobre o script de população fornecido pela cadeira, obtemos os seguintes resultados (média de 7 testes):

Execução	Tempo
Sem Índices	0.018300
Com Índices	0.001267

Query2:

Nesta query implementamos um índice sobre o estado da tabela Estado:

```
CREATE INDEX estadoIdx ON estado(estado);
```

O segundo índice é criado é criado automaticamente pelo MySQL numa BTree, visto que a morada e o codigo_espaco são chaves primarias da tabela posto.

Após uma execução da query com e sem os índices, sobre um script de população com um número de entradas bastante superior, obtemos os seguintes resultados (média de 7 testes):

Execução	Tempo
Sem Índice	17.68156
Com Índice	15.14811

Data Warehouse

Foi inserido na base de dados referente à entrega 3 um esquema em estrela com informação sobre reservas, tendo como dimensões: o utilizador que reservou, a localização, o tempo e a data.

As instruções sql necessárias para carregar o esquema em estrela na base de dados são as seguintes:

1. Dimensão Utilizador:

```
create table user_dimension
  (nif numeric(15,0) not null,
  nome varchar(255) not null,
  telefone numeric(9,0) not null,
  primary key(nif),
  foreign key(nif) references user(nif) ON DELETE CASCADE);
```

2. Dimensão Localização:

```
create table place_dimension
  (place_dim_id char(255) not null unique,
  morada varchar(255) not null,
  codigo_posto varchar(255),
  codigo_espaco varchar(255) not null,
  primary key(place_dim_id),
  foreign key(morada, codigo_espaco) references espaco(morada, codigo) ON DELETE CASCADE,
  foreign key(morada, codigo_posto) references posto(morada, codigo) ON DELETE CASCADE);
```

3. Dimensão Tempo:

```
create table time_dimension
  (time_dim_id int not null unique,
  time_hour int not null,
  time_minutes int not null,
  time_total_minutes int not null,
  primary key(time_dim_id));
```

4. Dimensão Data:

```
create table date_dimension
  (date_dim_id date not null unique,
  date_day int not null,
  date_week int not null,
  date_month int not null,
  date_semester int not null,
  date_year int not null,
  primary key(date_dim_id));
```

A tabela de factos, que é o centro do esquema em estrela, contem as informações relativas ao montante pago, duração em dias e identificador da reserva, para além disso contem os identificadores para as tabelas das quatro dimensões que pertencem ao esquema em estrela.

```
create table reserves
  (reserves_id varchar(255) not null unique,
  user_dim_id numeric(15,0) not null,
  place_dim_id char(255) not null,
  time_dim_id int not null,
  date_dim_id date not null,
  payed_sum numeric(15,2) not null,
  duration_time int not null,
  primary key(reserves_id),
  foreign key(user_dim_id) references user_dimension(nif) ON DELETE CASCADE,
  foreign key(place_dim_id) references place_dimension(place_dim_id) ON DELETE CASCADE,
  foreign key(time_dim_id) references time_dimension(time_dim_id) ON DELETE CASCADE,
  foreign key(date_dim_id) references date_dimension(date_dim_id) ON DELETE CASCADE);
```

Depois de ser criado o esquema em estrela, foi necessário carregar a informação presente na base de dados para este. Isto foi conseguido com o auxilio de procedures em que são chamados logo a seguir ao script de população fornecido pela cadeira.

1. Carregamento da informação relativa ao utilizador:

```
delimiter //
CREATE PROCEDURE load_user_dim()
BEGIN
   INSERT INTO user_dimension(nif, nome, telefone)
        SELECT nif, nome, telefone FROM user;
END; //
```

2. Carregamento da informação relativa à localização:

```
delimiter //
CREATE PROCEDURE load_place_dim()
BEGIN
   INSERT INTO place_dimension(place_dim_id, morada, codigo_espaco, codigo_posto)
   SELECT concat(morada, codigo), morada, codigo_espaco, codigo FROM posto
   union
   SELECT concat(morada, codigo), morada, codigo, null FROM espaco;
END; //
```

3. Carregamento da informação relativa ao tempo:

```
delimiter //
CREATE PROCEDURE load_time_dim()
BEGIN
   DECLARE v_full_date TIME;
   SET v_full_date = '00:00:00';
   WHILE v_full_date < '24:00:00' DO
       INSERT INTO time_dimension(time_dim_id, time_hour, time_minutes, time_total_minutes)
       VALUES (HOUR(v_full_date) * 100 + MINUTE(v_full_date),
           HOUR(v_full_date),
           MINUTE(v_full_date),
           HOUR(v_full_date)*60 + MINUTE(v_full_date));
       SET v_full_date = ADDTIME(v_full_date, '00:01:00');
   END WHILE;
END;
//
```

4. Carregamento da informação relativa à data:

```
delimiter //
CREATE PROCEDURE load_date_dim()
BEGIN
    DECLARE v_full_date DATE;
    DECLARE semester int;
    SET v_full_date = '2016-01-01';
    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 date_dimension(
          date_dim_id,
          date_day,
          date_week,
          date_month,
          date_semester,
          date_year
         ) VALUES (
             v_full_date,
              DAY(v_full_date),
              WEEKOFYEAR(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;//
```

5. Carregamento da informação relativa às reservas e às dimensões:

```
delimiter //
CREATE PROCEDURE load_reserves()
BEGIN
   INSERT INTO reserves(
     reserves_id,
     user_dim_id,
     place_dim_id,
     time_dim_id,
     date_dim_id,
     payed_sum,
     duration_time)
   select numero, nif, concat(morada, codigo), HOUR(data)*100 + MINUTE(data), date_format(data, '%Y-%m-%d'),
   tarifa*datediff(data_fim,data_inicio), datediff(data_fim,data_inicio)
   from paga natural join oferta natural join aluga;
END; //
```

Para realizar a consulta OLAP, foi criado um procedure que executa um query cujo o objetivo é obter um cubo com valor médio pago sobre as dimensões localização e data.

Visto que o mysql não é possível executar a instrução cube, a consulta foi elaborado com uma sucessão de unions.

```
delimiter //
CREATE PROCEDURE OLAP_search()
BEGIN
    select codigo_espaco, codigo_posto, date_day, date_month, avg(payed_sum)
    from reserves natural join place_dimension natural join date_dimension
    group by codigo_espaco, codigo_posto, date_day, date_month with rollup
    union
    select codigo_espaco, codigo_posto, date_day, date_month, avg(payed_sum)
    from reserves natural join place_dimension natural join date_dimension
    group by codigo_posto, date_day, date_month, codigo_espaco with rollup
    union
    select codigo_espaco, codigo_posto, date_day, date_month, avg(payed_sum)
    from reserves natural join place_dimension natural join date_dimension
    group by date_day, date_month, codigo_espaco, codigo_posto with rollup
    union
    select codigo_espaco, codigo_posto, date_day, date_month, avg(payed_sum)
    from reserves natural join place_dimension natural join date_dimension
    group by date_month, codigo_espaco, codigo_posto, date_day with rollup
    union
    select codigo_espaco, codigo_posto, date_day, date_month, avg(payed_sum)
    from reserves natural join place_dimension natural join date_dimension
    group by date_month, codigo_posto, codigo_espaco, date_day with rollup
    union
    select codigo_espaco, codigo_posto, date_day, date_month, avg(payed_sum)
    from reserves natural join place_dimension natural join date_dimension
    group by codigo_espaco, date_day, date_month, codigo_posto with rollup;
END; //
```