Instituto Superior Técnico

Projeto de Bases de Dados

Parte 4

Eduardo Janicas: 78974 (10H)

Diana Antunes: 82448 (8H)

Nuno Fernandes: 80774 (4H)

Número de grupo: 44

Turno: BD225179L06

Índices

Na interrogação 1, o MySQL utiliza os índices BTREE criados automaticamente quando as tabelas *arrenda* e *fiscaliza* são criadas. Utilizando o comando *show index*, observa-se que já existem dois índices para a tabela *fiscaliza* – o índice *PRIMARY*, referente à chave primária, constituído pelos seus três atributos (id, morada, código) e um índice secundário, constituído pelos atributos (morada, código) da foreign key, designado *morada*. Na tabela arrenda, existem também dois índices pré-criados: o índice *PRIMARY*, sobre a morada e código, e o índice *nif*, sobre o atributo foreign key *nif*. Para esta query, o MySQL usa os índices BTREE morada da tabela *fiscaliza* e PRIMARY da *arrenda*, fazendo a devida correspondência.

Neste caso, o ideal seria manter a tabela *arrenda* organizada por *nif* e a tabela *fiscaliza* por *id*, uma vez que a principal carga computacional vem das operações group by e having, após os joins já optimizados pelos indices BTREE primários. No entanto, uma ordenação total seria cara de se manter. Como tal, uma BTREE Clustered sobre esses atributos seria a opção ideal a escolher.

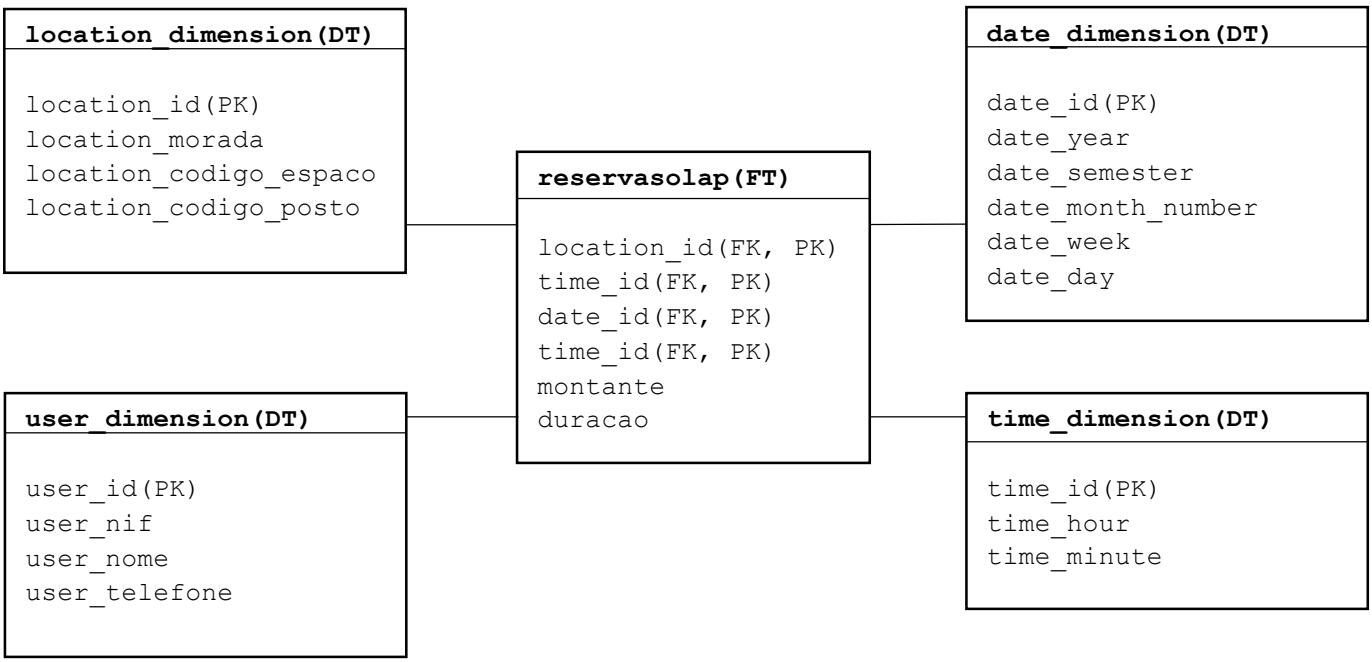
Na interrogação 2, além dos índices BTREE pré-criados das tabelas *posto*, *aluga* e *estado*, faria sentido criar um índice de HASH sobre os atributos (*morada*, *codigo_espaco*) da tabela *posto*, devido à comparação feita durante o *not in* da subquery em relação à query primária, durante o *where*. Note-se que a utilidade do índice de HASH provém de se supor que esta igualdade acontece uma pequena percentagem das vezes (até 10%). No entanto, como o engine InnoDB do MySQL não suporta este tipo de índices, não é possível implementá-los. Em relação ao *distinct* no *select*, o MySQL utiliza o índice BTREE *morada* gerado automaticamente sobre os atributos (*morada, codigo_espaco*) da tabela *posto*, não necessitando de otimização adicional. O mesmo se aplica em relação aos *joins*, uma vez que o SGBD utiliza os índices BTREE *PRIMARY* da tabela *aluga* e *PRIMARY* da tabela *estado* para juntar as tabelas, comparando apenas com os atributos-chave iniciais necessários (*morada*, *codigo*) no caso da junção *aluga-posto*, e *numero* no caso da junção *estado-aluga*. No caso do último *where*, como vai ser bastante frequente a igualdade verificar-se, não é necessário qualquer índice.

Arrenda	Fiscaliza	Execução Arrenda	Execução Fiscaliza
BTREE on (morada, código)	BTREE on (morada, código)	Usa índice (morada, código) para join com fiscaliza	Usa índice (morada, código) para join com arrenda. Usa uma tabela temporária e filesort
BTREE on (morada, código); BTREE on nif	None	Usa índice (morada, código) para join com fiscaliza	Usa uma tabela temporária e filesort
None	BTREE on (morada, código)	Usa um join buffer (Block Nested Loop)	Usa índice (morada, código) para join com arrenda. Usa uma tabela temporária e filesort
None	None	Usa um join buffer (Block Nested Loop)	Usa uma tabela temporária e filesort

Posto (Primário)	Posto (Subquery)	Aluga	Estado	Execução Posto (P)	Execução Posto (S)	Execução Aluga	Execução Esta
BTREE on (morada, código_espaco);	BTREE on (morada, código)	BTREE on (morada, código, data_inicio, nif, numero)	BTREE on (numero, timestamp)	Usa índice (morada, código_espaço) para o not in e distinct	Usa índice primário para comparar com a tabela aluga	Usa índice primário para join com a tabela posto	Usa índice primá para join com a tabela aluga
None	None	None	None	Usa uma tabela temporária		Usa um join buffer (Block Nested Loop)	Usa um join buffe (Block Nested Loop)
BTREE on (morada, código_espaco);	None	None	None	Usa índice (morada, código_espaço) para o not in e distinct		Usa um join buffer (Block Nested Loop)	Usa um join buffe (Block Nested Loop)
None	BTREE on (morada, código)	BTREE on (morada, código, data_inicio, nif, numero)	BTREE on (numero, timestamp)	Usa uma tabela temporária	Usa índice primário para comparar com a tabela aluga	Usa índice primário para join com a tabela posto	Usa índice primá para join com a tabela aluga

Os comandos *explain* usados para simular as execuções com e sem índices encontram-se no ficheiro index.sql. Para não usar índices numa determinada tabela esta foi importada na clausula from seguida de *use index()*.

Data Warehouse



Estrutura da tabela *location_dimension*:

```
DROP TABLE IF EXISTS location_dimension;
CREATE TABLE location_dimension (
      location_id             int(11)           NOT NULL,
      location_morada         varchar(255)      NOT NULL,
      location_codigo_espaco  varchar(255)      NOT NULL,
      location_codigo_posto   varchar(255)
);
```

Popular a tabela *location_dimension*. O identificador é um contador sequêncial na tabela:

```
SET @count = 0;
INSERT INTO location_dimension
      SELECT  @count:=@count+1 AS location_id,
                  espaco.morada AS location_morada,
                  espaco.codigo AS location_codigo_espaco,
                  posto.codigo AS location_codigo_posto FROM
      espaco LEFT JOIN posto
      ON espaco.codigo = posto.codigo_espaco
;
```

Estrutura da tabela *user_dimension*:

```
DROP TABLE IF EXISTS user_dimension;
CREATE TABLE user_dimension (
      user_id           int(11)           NOT NULL,
      user_nif          varchar(9)        NOT NULL,
      user_nome         varchar(80)       NOT NULL,
      user_telefone     varchar(26)       NOT NULL
);
```

Popular a tabela *user_dimension*. O identificador é um contador sequêncial na tabela:

```
SET @count = 0;
INSERT INTO user_dimension
      SELECT @count:=@count+1 AS user_id,
             nif as user_nif,
             nome as user_nome,
             telefone as user_telefone
      FROM user;
```

Estrutura da tabela *date_dimension*:

```
DROP TABLE IF EXISTS date_dimension;
CREATE TABLE date_dimension (
  date_id               int(11)          NOT NULL,
  date_year             int(11)          NOT NULL,
  date_semester         int(11)          NOT NULL,
  date_month_number     int(11)          NOT NULL,
  date_week             int(11)          NOT NULL,
  date_day              int(11)          NOT NULL
);
```

Popular a tabela *date_dimension*. Foi escolhido como id um valor lógico onde se multiplica o ano por 10000, o mês por 100 e se somam os dias (conforme exemplo disponibilizado pelos professores):

```
DROP PROCEDURE IF EXISTS createdatedimension;
DELIMITER //
CREATE PROCEDURE createdatedimension()
BEGIN
   DECLARE v_full_date DATETIME;
   SET v_full_date = '2016-01-01 00:00:00';
   WHILE v_full_date < '2018-01-01 00:00:00' DO
       INSERT INTO date_dimension(
          date_id,
          date_year,
          date_semester,
          date_month_number,
          date_week,
          date_day
       ) VALUES (
           YEAR(v_full_date) * 10000 + MONTH(v_full_date)*100 + DAY(v_full_date),
           YEAR(v_full_date),
           CEIL(QUARTER(v_full_date) / 2),
           MONTH(v_full_date),
           WEEKOFYEAR(v_full_date),
           DAY(v_full_date)
       );
       SET v_full_date = DATE_ADD(v_full_date, INTERVAL 1 DAY);
   END WHILE;
END; //
DELIMITER ;
CALL createdatedimension();
```

Estrutura da tabela *time_dimension*:

```
DROP TABLE IF EXISTS time_dimension;
CREATE TABLE time_dimension (
  time_id         int(11)     NOT NULL,
  time_hour       int(11)     NOT NULL,
  time_minute     int(11)     NOT NULL
);
```

Popular a tabela *time_dimension*. Foi escolhido como id um valor lógico onde se multiplica a hora por 100 e se somam os minutos:

```
DROP PROCEDURE IF EXISTS createtimedimension;
DELIMITER //
CREATE PROCEDURE createtimedimension()
BEGIN
   DECLARE v_full_date DATETIME;
   SET v_full_date = '2016-01-01 00:00:00';
   WHILE v_full_date <= '2016-01-01 23:59:59' DO
       INSERT INTO time_dimension(
          time_id,
          time_hour,
          time_minute)
       VALUES (
           HOUR(v_full_date)*100 + MINUTE(v_full_date),
           HOUR(v_full_date),
           MINUTE(v_full_date)
       );
       SET v_full_date = DATE_ADD(v_full_date, INTERVAL 1 MINUTE);
   END WHILE;
END; //
DELIMITER ;
CALL createtimedimension();
```

Estrutura da tabela *reservasolap*:

```
DROP TABLE IF EXISTS reservasolap;
CREATE TABLE reservasolap (
      location_id       int(11)           NOT NULL,
      user_id           int(11)           NOT NULL,
      date_id           int(11)           NOT NULL,
      time_id           int(11)           NOT NULL,
      montante          varchar(255)      NOT NULL,
      duracao           varchar(255)      NOT NULL
);
```

Popular a tabela *reservasolap*. É criada a partir das tabelas reserva, aluga, oferta e paga já existentes. Para os valores de location_id e user_id é necessários comparer os dados recebidos com as dimensões user e location, uma vez que os ids não tem qualquer significado lógico (são apenas um contador sequencial nas respetivas tabelas):

```
INSERT INTO reservasolap
      SELECT location_id,
             user_id,
                 YEAR(data) * 10000 + MONTH(data)*100 + DAY(data) AS date_id,
                 HOUR(data)*100 + MINUTE(data) AS time_id,
                 datediff(data_fim, data_inicio)*tarifa as montante,
                 datediff(data_fim, data_inicio) as duracao
      FROM reserva NATURAL JOIN aluga NATURAL JOIN oferta NATURAL JOIN paga
      INNER JOIN location_dimension ON
      aluga.morada = location_dimension.location_morada AND
      (aluga.codigo = location_dimension.location_codigo_posto OR
      aluga.codigo = location_dimension.location_codigo_espaco)
      INNER JOIN user_dimension ON
      aluga.nif = user_dimension.user_nif;
```

Cubo (Note-se que uma vez que o MySQL não suporta esta operação, é necessário fazer uma união de Rollups onde a ordem dos mesmos é variada):

```
SELECT location_codigo_espaco, location_codigo_posto, date_month_number, date_day,
       AVG(montante) as average
       FROM reservasolap NATURAL JOIN location_dimension NATURAL JOIN date_dimension
       GROUP BY location_codigo_espaco, location_codigo_posto, date_month_number,
       date_day WITH ROLLUP
UNION
SELECT location_codigo_espaco, location_codigo_posto, date_month_number, date_day,
       AVG(montante) as average
       FROM reservasolap NATURAL JOIN location_dimension NATURAL JOIN date_dimension
       GROUP BY location_codigo_posto, date_month_number, date_day,
       location_codigo_espaco WITH ROLLUP
UNION
SELECT location_codigo_espaco, location_codigo_posto, date_month_number, date_day,
       AVG(montante) as average
       FROM reservasolap NATURAL JOIN location_dimension NATURAL JOIN date_dimension
       GROUP BY date_month_number, date_day, location_codigo_espaco,
       location_codigo_posto WITH ROLLUP
UNION
SELECT location_codigo_espaco, location_codigo_posto, date_month_number, date_day,
       AVG(montante) as average
       FROM reservasolap NATURAL JOIN location_dimension NATURAL JOIN date_dimension
       GROUP BY date_day, location_codigo_espaco, location_codigo_posto,
       date_month_number WITH ROLLUP;
```