Instituto Superior Técnico

Projeto de Bases de Dados

Parte 4

Eduardo Janicas: 78974 (8H)

Diana Antunes: 82448 (6H)

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 operacoes 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 (ate 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 optimização 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.

Data Warehouse

Table structure for table `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)

);

Dumping data for table `location\_dimension`

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

;

Table structure for table `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

);

Dumping data for table `user\_dimension`

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;

Table structure for table `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

);

Dumping data for table `date\_dimension`

DELIMITER //

DROP PROCEDURE IF EXISTS createdatedimension;

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();

Table structure for table `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

);

Dumping data for table `time\_dimension`

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();

Table structure for table `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

);

Dumping data for table `time\_dimension`

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.morada AND

      (aluga.codigo = location\_dimension.codigo\_posto OR

      aluga.codigo = location\_dimension.codigo\_espaco)

      INNER JOIN user\_dimension ON

      aluga.nif = user\_dimension.nif;

Cube

SELECT codigo\_espaco, codigo\_posto, date\_month\_number, date\_day,

AVG(montante) as average

   FROM reservasolap NATURAL JOIN location\_dimension NATURAL JOIN date\_dimension

   GROUP BY codigo\_espaco, codigo\_posto, date\_month\_number, date\_day WITH ROLLUP

UNION

SELECT codigo\_espaco, codigo\_posto, date\_month\_number, date\_day,

AVG(montante) as average

   FROM reservasolap NATURAL JOIN location\_dimension NATURAL JOIN date\_dimension

   GROUP BY codigo\_posto, date\_month\_number, date\_day, codigo\_espaco WITH ROLLUP

UNION

SELECT codigo\_espaco, 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, codigo\_espaco, codigo\_posto WITH ROLLUP

UNION

SELECT codigo\_espaco, 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, codigo\_espaco, codigo\_posto, date\_month\_number WITH ROLLUP;