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 Estado** |
| 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 primário para join com a tabela aluga |
| None | None | None | None | Usa uma tabela temporária |  | Usa um join buffer (Block Nested Loop) | Usa um join buffer (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 buffer (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 primário 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

**date\_dimension(DT)**

date\_id(PK)

date\_year

date\_semester

date\_month\_number

date\_week

date\_day

**location\_dimension(DT)**

location\_id(PK)

location\_morada

location\_codigo\_espaco

location\_codigo\_posto

**reservasolap(FT)**

location\_id(FK, PK)

time\_id(FK, PK)

date\_id(FK, PK)

time\_id(FK, PK)

montante

duracao

**time\_dimension(DT)**

time\_id(PK)

time\_hour

time\_minute

**user\_dimension(DT)**

user\_id(PK)

user\_nif

user\_nome

user\_telefone

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;