Projeto de Bases de Dados, Parte 4

Grupo 1, BD225179L08
Daniel Correia, 80697
Carolina Inês Xavier, 81172
Inês Leite, 81328
Esforço: 14 horas

Índices

a)

Em MySQL não é possível implementar bitmap e hash-indexes por isso não podemos testar mas sabemos que hash-indexes para a primeira interrogação e bitmap para a segunda seriam melhores opções.

Índices hash sobre os atributos morada e código das tabelas Arrenda, Fiscaliza, Posto e Aluga com função de dispersão dinâmica.

É uma melhor opção porque não temos testes de intervalos (para estes B+ seria mais adequado) mas sim testes de igualdade na 1ª interrogação em:

Arrenda A
inner join Fiscaliza F
on A.morada = F.morada
and A.codigo = F.codigo

e na 2ª interrogação em

Posto P
natural join Aluga A
natural join Estado E

Índices hash são as melhores para seleção por igualdade porque cada índice corresponde a um bucket que armazena um conjunto de dados, logo a função de dispersão aponta logo para o bucket onde os dados procurados se encontram.

A função de dispersão é dinâmica - número de *buckets* varia ao longo do ciclo de vida do índice - porque podemos adicionar novas entidades às tabelas Arrenda, Fiscaliza, Posto ou Aluga da base de dados. Com a função de dispersão dinâmica evita-se um aumento no tempo de procura de dados, porque quando se acrescentam novas entidades são criados novos buckets, poupando-se assim no tempo de procura dos dados dentro do bucket.

Índices bitmaps sobre o atributo estado da tabela Estado, porque estes são mais compactos que índices B+.

Sabemos que o atributo estado da tabela Estado tem um número reduzido de valores possíveis, e nesta query é feito um teste de igualdade para este atributo em

where E.estado='aceite'

Neste caso é mais vantajoso o uso de índices bitmap porque é um array de bits, em que o estado para cada registo é identificado por bits, logo nesta operação passa a ser apenas necessário verificar quais os registos com o bit de estado aceite a 1.

b)

Observando os tempos para correr as query sem índices e depois implementando índices nos atributos das tabelas 'visitadas' em cada uma das queries conclui-se que para se obter uma maior eficiência na realização da 1ª query os índices a criar são: índices B+ sobre os atributos primários das tabelas Aluga e Fiscaliza e sobre o conjunto dos atributos morada e código destas tabelas.

E para se obter uma maior eficiência na realização da 2ª query os índices que necessitamos de criar são: índices B+ sobre os atributos primários das tabelas Posto, Aluga e Estado e sobre o conjunto dos atributos morada e código das tabelas Posto e Aluga.

É possível verificar o benefício destes índices comparando os tempos de execução das queries sem índices das Foreign Keys e os tempos com os índices *default* gerados pelo MySQL para as chaves primárias.

Os tempos obtidos foram:

	Query 1	Query 2
Sem indices nas FKs	0.092446	0.088322
Com índices *default*	0.058285	0.012372

A criação destes índices é benéfica porque sem os índices implementados, para aceder a qualquer registo é necessário percorrer todos os registos até se encontrar o desejado, enquanto que com os índices implementados, sabendo os registos a que queremos aceder, é possível aceder-lhes diretamente através do seu índice.

Os índices para os conjuntos morada e código são benéficos porque nas queries estes são os atributos acedidos, logo com um índice conjunto é possível aceder aos dois.

Data Warehouse

User dimension

```
DROP TABLE IF EXISTS user_dimension;
CREATE TABLE user_dimension (
  user_id varchar(13) NOT NULL,
  user_nif varchar(9) NOT NULL,
  user_nome varchar(80) NOT NULL,
  user_telefone varchar(26) NOT NULL,
  PRIMARY KEY (user_id)
);

INSERT INTO user_dimension
SELECT
  concat('user',nif) as user_id,
  nif as user_nif,
  nome as user_nome,
  telefone as user_telefone
FROM user;
```

Location dimension

```
DROP TABLE IF EXISTS location_dimension;
CREATE TABLE location_dimension (
  location_id varchar(765) NOT NULL,
  morada varchar(255) NOT NULL,
  codigo_espaco varchar(255) NOT NULL,
  codigo_posto varchar(255),
  location_foto varchar(255),
PRIMARY KEY (location_id)
);

INSERT INTO location_dimension
SELECT
  concat(morada,codigo) as location_id,
  morada,
  codigo,
  NULL,
  foto
FROM espaco natural join alugavel;

INSERT INTO location_dimension
SELECT
  concat(morada,codigo_espaco,codigo),
  morada,
  codigo_espaco,
  codigo,
  foto
FROM posto natural join alugavel;
```

Date dimension

```
DROP TABLE IF EXISTS date_dimension;
CREATE TABLE date_dimension (
  date_id    int(11) NOT NULL,
  date_time date DEFAULT NULL,

  date_year    int(11) DEFAULT NULL,
  semester    int(11) DEFAULT NULL,

  month_number   int(11) DEFAULT NULL,
  month_name  char(10) DEFAULT NULL,

  week_number    int(11) DEFAULT NULL,
  week_day_number    int(11) DEFAULT NULL,
  week_day_name char(10) DEFAULT NULL,
  PRIMARY KEY (date_id)
);

DROP PROCEDURE IF EXISTS populate_date_dimension;
DELIMITER //
CREATE PROCEDURE populate_date_dimension()
  BEGIN
    SET @d0 = '2016-01-01';
    SET @d1 = '2017-12-31';
    SET @date = date_sub(@d0, INTERVAL 1 DAY);

    WHILE date_add(@date, INTERVAL 1 DAY) <= @d1 DO
      SET @date = date_add(@date, INTERVAL 1 DAY);
      IF quarter(@date) <= 2 THEN
        SET @semester = 1;
      ELSE
        SET @semester = 2;
      END IF;
      INSERT INTO date_dimension VALUES(
        date_format(@date, "%Y%m%d"),
        @date,

        year(@date),
        @semester,

        month(@date),
        monthname(@date),

        week(@date),
        day(@date),
        dayname(@date)
      );
    END WHILE;
  END //

CALL populate_date_dimension();
```

Time dimension

```
DROP TABLE IF EXISTS time_dimension;
CREATE TABLE time_dimension (
  time_id int(4) NOT NULL,
  time_of_day time NOT NULL,
  hour_of_day int(2) NOT NULL,

  minute_of_day int(4) NOT NULL,
  minute_of_hour int(2) NOT NULL,
  PRIMARY KEY (time_id)
);

DROP PROCEDURE IF EXISTS populate_time_dimension;
DELIMITER //
CREATE PROCEDURE populate_time_dimension()
  BEGIN
    SET @t0 = '2016-11-11 00:00:00';
    SET @t1 = '2016-11-11 23:59:59';
    SET @time = date_sub(@t0, INTERVAL 1 MINUTE);
    WHILE date_add(@time, INTERVAL 1 MINUTE) <= @t1 DO
      SET @time = date_add(@time, INTERVAL 1 MINUTE);
      SET @minuteofday = ( hour(@time) * 60 ) + minute(@time) + 1;
      INSERT INTO time_dimension VALUES(
            date_format(@time, "%H%i"),
            @time,
            hour(@time),
            @minuteofday,
            minute(@time)
      );
    END WHILE;
  END //

CALL populate_time_dimension();
```

Reservas info

```
ATURDROP TABLE IF EXISTS reservas_info;
CREATE TABLE reservas_info (
  reserva_id varchar(255) NOT NULL,
  user_id varchar(13) NOT NULL,
  location_id varchar(510) NOT NULL,
  time_id int(4) NOT NULL,
  date_id int(11) NOT NULL,
  montante_pago int NOT NULL,
  duracao int NOT NULL,
  PRIMARY KEY (reserva_id,time_id,date_id),
  FOREIGN KEY(time_id) REFERENCES time_dimension(time_id),
  FOREIGN KEY(date_id) REFERENCES date_dimension(date_id),
  FOREIGN KEY(location_id) REFERENCES location_dimension(location_id),
  FOREIGN KEY(user_id) REFERENCES user_dimension(user_id)
);
INSERT INTO reservas_info
SELECT
  numero as reserva_id,
  concat('user',nif) as user_id,
  concat(morada,codigo) as location_id,
  date_format(data, "%H%i") as time_id,
  date_format(data, "%Y%m%d") as date_id,
  tarifa * (data_fim - data_inicio) as montante_pago,
  data_fim - data_inicio as duracao
FROM aluga NATURAL JOIN oferta NATURAL JOIN espaco NATURAL JOIN paga
```

```
UNION

SELECT
  numero as reserva_id,
  concat('user',nif) as user_id,
  concat(morada, codigo_espaco, codigo) as location_id,
  date_format(data, "%H%i") as time_id,
  date_format(data, "%Y%m%d") as date_id,
  tarifa * (data_fim - data_inicio) as montante_pago,
  data_fim - data_inicio as duracao
FROM aluga NATURAL JOIN oferta NATURAL JOIN posto NAL JOIN paga;
```

Cube

```
SELECT codigo_espaco, codigo_posto, month_number, week_day_number, avg(montante_pago)
FROM reservas_info
  NATURAL JOIN location_dimension
  NATURAL JOIN date_dimension
GROUP BY codigo_espaco, codigo_posto, month_number, week_day_number WITH ROLLUP

UNION

SELECT codigo_espaco, codigo_posto, month_number, week_day_number, avg(montante_pago)
FROM reservas_info
  NATURAL JOIN location_dimension
  NATURAL JOIN date_dimension
GROUP BY codigo_posto, month_number, week_day_number, codigo_espaco  WITH ROLLUP

UNION

SELECT codigo_espaco, codigo_posto, month_number, week_day_number, avg(montante_pago)
FROM reservas_info
  NATURAL JOIN location_dimension
  NATURAL JOIN date_dimension
GROUP BY month_number, week_day_number, codigo_espaco, codigo_posto  WITH ROLLUP

UNION

SELECT codigo_espaco, codigo_posto, month_number, week_day_number, avg(montante_pago)
FROM reservas_info
  NATURAL JOIN location_dimension
  NATURAL JOIN date_dimension
GROUP BY week_day_number, codigo_espaco, codigo_posto, month_number WITH ROLLUP

UNION

SELECT codigo_espaco, codigo_posto, month_number, week_day_number, avg(montante_pago)
FROM reservas_info
  NATURAL JOIN location_dimension
  NATURAL JOIN date_dimension
GROUP BY codigo_espaco, month_number, codigo_posto, week_day_number WITH ROLLUP

UNION

SELECT codigo_espaco, codigo_posto, month_number, week_day_number, avg(montante_pago)
FROM reservas_info
  NATURAL JOIN location_dimension
  NATURAL JOIN date_dimension
GROUP BY codigo_posto, week_day_number, codigo_espaco, month_number WITH ROLLUP;
```