Projeto de Bases de Dados, Parte 4

Grupo 1, BD225179L08 – Prof. Gabriel Pestana
Daniel Correia, 80697
Carolina Inês Xavier, 81172
Inês Leite, 81328

Esforço: 14 horas

Índices

a)

Na query 1, escolhemos índices hash sobre os atributos morada e código das tabelas Arrenda e Fiscaliza com função de dispersão dinâmica porque não temos testes de intervalos (para estes B+ seria mais adequado) mas sim testes de igualdade na operação de join em:

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

Na query 2, escolhemos índices hash sobre os atributos morada e código das tabelas Posto e Aluga pelas mesmas razões da query anterior aplicados agora sobre a operação de natural join em:

Posto P natural join Aluga A natural join Estado E

Os índices hash são os melhores para seleção por igualdade porque cada chave (gerada pela função de dispersão do 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 seria dinâmica (ou seja, número de *buckets* a variar ao longo do ciclo de vida do índice) porque, quando adicionarmos novas entidades às tabelas em que estes índices são aplicados serão gerados novos buckets, evitando situações de overflow em algum bucket e reduzindo o tempo de procura dos dados dentro de cada bucket.

Para além dos índices hash na query 2, também escolhemos criar um índice bitmap sobre o atributo estado da tabela Estado porque este atributo tem um número reduzido de valores possíveis e a query faz um teste de igualdade sobre este atributo (**where E.estado='aceite'**) .

Neste caso, o uso de índices bitmap é mais vantajoso porque, sendo um array de bits em que o estado para cada registo é identificado por 1 bit, o índice bitmap será mais compacto (menor espaço em disco/memória) e os testes de igualdade realizados sobre este índice serão mais

rápidos porque passa a ser apenas necessário verificar quais os registos com o bit de estado aceite a 1.

b) Em MySQL não é possível implementar bitmap e hash-indexes por isso não podemos testar mas sabemos que os índices da alínea anterior seriam os mais indicados para as queries analisadas. Como não é possível implementar os índices desejados, comparamos o desempenho das queries em três situações: com os indices disponiveis por default (índices de Foreign Key e de Primary Key), com índices default excepto os índices das Foreign Keys e com índices default acrescentando índices B+ feitos por nós.

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:

	Com índices nas Primary keys + FKs	Com índices nas Primary keys	Com índices default + B+ acrescentados
Query 1	0.058285	0.092446	0.632257
Query 2	0.014319	0.015586	0.014473

Observando os tempos para correr as queries sem índices em relação às restantes situações, podemos concluir que os índices existentes por default para as Primary Keys e Foreign Keys das tabelas são os que permitem realizar as queries de maneira mais eficiente.

No caso dos índices acrescentados aos já existentes por default, estes não provocaram melhorias de desempenho porque são índices B+ criados sobre atributos de chaves primárias ou estrangeiras.

O uso dos índices B+ default levam a uma maior eficiência na realização das queries porque, sem qualquer índice implementado, para aceder a qualquer registo é necessário percorrer toda a tabela para se encontrar o registo desejado, enquanto que, com índices

implementados, sabendo os registos a que queremos aceder, é possível aceder-lhes com menor número de comparações/testes através do seu índice.

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

Com base nestas observações, os índices default estão implementados sobre os mesmos atributos para os quais gostaríamos de criar os índices hash (para ambas as queries). Como nestas queries são realizadas operações de join com estes atributos, um índice de hash sobre estes atributos seria mais eficiente do que o índice B+ existente por default, porque B+ é mais adequado

para testes de intervalo, enquanto que o índice hash é mais adequado aos testes de igualdade realizados nas operações do tipo join.

Data Warehouse

User dimension

```
DROP TABLE IF EXISTS user_dimension;
CREATE TABLE user_dimension (
  user_id varchar(13),
  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),
  morada varchar(255) NOT NULL,
  codigo_espaco varchar(255) NOT NULL,
  codigo_posto varchar(255) NOT NULL,
  PRIMARY KEY (location_id)
);

INSERT INTO location_dimension
SELECT
  concat(morada,codigo) as location_id,
    morada,
    codigo,
    ' '
FROM espaco
UNION
SELECT
  concat(morada,codigo_espaco,codigo),
  morada,
  codigo_espaco,
  codigo
FROM posto;
```

Date dimension

```
DROP TABLE IF EXISTS date_dimension;
CREATE TABLE date_dimension (
  date_id    int(11),
  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 = @d0;

    WHILE @date <= @d1 DO
      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)
      );
      SET @date = date_add(@date, INTERVAL 1 DAY);
    END WHILE;
  END //
CALL populate_date_dimension();
```

Time dimension

```
DROP TABLE IF EXISTS time_dimension;
CREATE TABLE time_dimension (
  time_id int(4),
  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 = @t0;
    WHILE @time <= @t1 DO
      SET @minuteofday = ( hour(@time) * 60 ) + minute(@time) + 1;
      INSERT INTO time_dimension VALUES(
        date_format(@time, "%H%i"),
        @time,
        hour(@time),
        @minuteofday,
        minute(@time)
      );
      SET @time = date_add(@time, INTERVAL 1 MINUTE);
    END WHILE;
  END //
CALL populate_time_dimension();
```

Reservas (Tabela de Factos)

```
DROP TABLE IF EXISTS reservas_info;
CREATE TABLE reservas_info (
  user_id varchar(13),
  location_id varchar(510),
  time_id int(4),
  date_id int(11),
  montante_pago int NOT NULL,
  duracao int NOT NULL,
  PRIMARY KEY (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
  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
  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 NATURAL JOIN paga;
```

Cube query

```
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;
```