TÉCNICO LISBOA

Base de Dados

Licenciatura em Engenharia Telecomunicações e Informática

Relatório

Projeto de Base de Dados, Parte 4

sexta-feira, 16 de Dezembro de 2016

Grupo 14
BD8179L04

João Freitas - 81950 (7 horas)
João Carlos Costa – 82528 (7 horas)
Mariana Cruz – 82553 (7 horas)

Índices

1. **a)** Neste exercício são usados dois índices, um para a tabela Arrenda e outro para a tabela Fiscaliza, sendo que aquele que mais interessa é o índice da tabela Fiscaliza sobre o atributo morada.

   b) Não criamos índices para este exercício pois já estavam a ser usados, os quais achamos os melhores em termos de otimização.

   Plano de execução obtido:

```
mysql> EXPLAIN SELECT A.nif FROM arrenda A inner join fiscaliza F ON A.morada = F.morada
AND A.codigo = F.codigo GROUP BY A.nif HAVING COUNT(distinct F.id) = 1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: A
         type: index
possible_keys: PRIMARY
          key: nif
      key_len: 4
          ref: NULL
         rows: 14
        Extra: Using index
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: F
         type: ref
possible_keys: morada
          key: morada
      key_len: 261
          ref: ist181950.A.morada,ist181950.A.codigo
         rows: 1
        Extra: Using index
2 rows in set (0.00 sec)
```

2. **a)** Neste exercício os índices existentes são, morada e código_espaco, usados na tabela Posto após o select distinct (using índex group by), é usado ainda outro índice para a condição where not in (using where) que verifica se a morada, código_espaco existe na sub query, os índices seguintes são usados nos natural join entre as tabelas (posto,aluga) e (aluga,estado) o último é utilizado no where na tabela Estado para verificar se o estado está aceite.
   Estes são os índices retirados através do comando explain da query, como visto na imagem em baixo.
   Faria sentido substituir o ultimo índice (Estado) usando uma Hash. Para uma melhor otimização essa deverá ser a alteração feita aos índices.

   b) CREATE INDEX e_idx ON Estado(estado) USING HASH;

Nota: MySQL não suporta HASH.

Plano de execução obtido:

```
[mysql> EXPLAIN SELECT distinct P.morada, P.codigo_espaco FROM posto P WHERE (P.morada, P.]
codigo_espaco) not in (SELECT P.morada, P.codigo_espaco FROM posto P NATURAL JOIN aluga A
 NATURAL JOIN estado E WHERE E.estado = 'Aceite')\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: P
         type: range
possible_keys: NULL
          key: morada
      key_len: 261
          ref: NULL
         rows: 13
        Extra: Using where; Using index for group-by
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: P
         type: ref
possible_keys: PRIMARY,morada
          key: morada
      key_len: 261
          ref: func,func
         rows: 3
        Extra: Using where; Using index
*************************** 3. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: A
         type: ref
possible_keys: PRIMARY,numero
          key: PRIMARY
      key_len: 261
          ref: ist181950.P.morada,ist181950.P.codigo
         rows: 1
        Extra: Using index
*************************** 4. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: E
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: ist181950.A.numero
         rows: 1
        Extra: Using where
```

Data Warehouse

1.

a) Em vez de criarmos uma dimensão nova para o Utilizador que reservou, utilizamos uma tabela já existente (a *user*), utilizando o *nif* como id da tabela reserva_reading.

b) Como os registos da dimensão Localização não devem ser gerados automaticamente recorremos a utilização de Triggers, usando um para cada tabela (posto, espaço e edifício). Optámos também por usar *after insert* para obter os valores de cada entrada nova da dimensão.

```
create table location_dimension
    (location_id int auto_increment,
     morada varchar(255) not null,
     codigo_espaco int,
     codigo int,
     primary key(location_id)
     constraint foreign key(morada) references edificio(morada) on delete cascade);
```

```
CREATE TRIGGER populate_location_dimension_posto AFTER INSERT ON posto
FOR EACH ROW
BEGIN
insert into location_dimension (morada, codigo_espaco, codigo) values (NEW.morada, NEW.codigo_espaco, NEW.codigo);
END //

CREATE TRIGGER populate_location_dimension_espaco AFTER INSERT ON espaco
FOR EACH ROW
BEGIN
insert into location_dimension (morada, codigo_espaco, codigo) values (NEW.morada, NEW.codigo, NEW.codigo);
END //

CREATE TRIGGER populate_location_dimension_edificio AFTER INSERT ON edificio
FOR EACH ROW
BEGIN
insert into location_dimension (morada) values (NEW.morada);
END //
```

c) Sendo que a dimensão Tempo deve conter todos os minutos de um dia criamos um procedimento em que os minutos são incrementados por uma unidade de minuto até ao final de um dia.

```
create table time_dimension
    (time_id int not null,
     time_of_day time not null,
     hour_of_day int not null,
     minute_of_day int not null,
     minute_of_hour int not null,
     primary key(time_id));
```

```
CREATE PROCEDURE populate_time_dimension()
BEGIN
    DECLARE time_id INT;
    DECLARE time_of_day time;
    DECLARE hour_of_day INT;
    DECLARE minute_of_day INT;
    DECLARE minute_of_hour INT;
    SET @time_id=0;
    SET @time_of_day='00:00:00';
    SET @hour_of_day=0;
    SET @minute_of_day=0;
    SET @minute_of_hour=0;
    WHILE @hour_of_day < 24 DO
    WHILE @minute_of_hour < 60 DO
        insert into time_dimension values (@time_id, @time_of_day, @hour_of_day, @minute_of_day, @minute_of_hour);
            SET @time_id = @time_id + 1;
            SET @minute_of_hour = @minute_of_hour + 1;
            SET @minute_of_day = @minute_of_day + 1;
            SET @time_of_day = ADDTIME(@time_of_day, '0:1');
    END WHILE;
    SET @minute_of_hour = 0; SET @hour_of_day = @hour_of_day + 1;
    END WHILE;
END //
```

d) Como a dimensão Data deve conter todos os dias dos anos 2016 e 2017 criamos também um procedimento em que o dia é incrementado por uma unidade.

```
CREATE PROCEDURE populate_date_dimension()
BEGIN
    DECLARE date_id int;
    DECLARE date_time date;
    DECLARE date_year int;
    DECLARE date_semester int;
    DECLARE date_month_number int;
    DECLARE date_month_name varchar(255);
    DECLARE date_week_number int;
    DECLARE date_week_day_number int;
    DECLARE date_week_day_name varchar(255);
    DECLARE date_day_of_month_number int;
    DECLARE date_day_number int;
    DECLARE var int;

    SET @date_id=1;
    SET @date_time='2016-01-01';
    SET @date_year=YEAR(@date_time);
    SET @date_semester=1;
    SET @date_month_number=MONTH(@date_time);
    SET @date_month_name=MONTHNAME(@date_time);
    SET @date_week_number=1;
    SET @date_week_day_number=DAYOFWEEK(@date_time) - 1;
    IF @date_week_day_number = 0 THEN
    SET @date_week_day_number = 7;
    END IF;
    SET @date_week_day_name=DAYNAME(@date_time);
    SET @date_day_of_month_number=DAYOFMONTH(@date_time);
    SET @date_day_of_year_number=DAYOFYEAR(@date_time);
    SET var = @date_year;
    WHILE @date_year < 2018 DO
        insert into date_dimension values (@date_id, @date_time, @date_year, @date_semester, @date_month_number, @date_week_number, @date_day_of_month_number);
            SET @date_id = @date_id + 1;
            SET @date_time=ADDDATE(@date_time, INTERVAL 1 DAY);
            SET @date_year=YEAR(@date_time);
            SET @date_month_number=MONTH(@date_time);
            SET @date_month_name=MONTHNAME(@date_time);
            SET @date_week_day_number=DAYOFWEEK(@date_time) - 1;
            SET @date_week_day_name=DAYNAME(@date_time);
            SET @date_day_of_month_number=DAYOFMONTH(@date_time);
            SET @date_day_of_year_number=DAYOFYEAR(@date_time);
            IF @date_month_number < 7 THEN
            SET @date_semester=1;
            ELSE
            SET @date_semester=2;
            END IF;
            IF @date_week_day_number = 0 THEN
            SET @date_week_day_number = 7;
            END IF;
            IF @date_week_day_number = 1 THEN
            SET @date_week_number=@date_week_number + 1;
            END IF;
            IF @date_year <> var THEN
            SET @date_week_number=1;
            SET var = @date_year;
            END IF;
    END WHILE;
```

```
create table date_dimension
    (date_id int not null,
     date_time date not null,
     date_year int not null,
     date_semester int not null,
     date_month_number int not null,
     date_week_number int not null,
     date_day_of_month_number int not null,
     primary key(date_id));
```

Para carregar o esquema em estrela a partir das tabelas existentes temos a tabela *reserva_reading* que por sua vez irá conter todos os ids das tabelas anteriores, a sua chave primária, o montante pago e a duração da reserva.

```
create table reserva_reading
    (nif int not null,
     location_id int not null default 0,
     time_id int not null default 0,
     date_id int not null default 0,
     montante_pago decimal(10,2) not null default 0,
     duracao_dias int not null default 0,
     primary key(nif,location_id,time_id,date_id),
     constraint foreign key(nif) references user(nif) on delete cascade,
     constraint foreign key(location_id) references location_dimension(location_id) on delete cascade,
     constraint foreign key(time_id) references time_dimension(time_id) on delete cascade,
     constraint foreign key(date_id) references date_dimension(date_id) on delete cascade);
CREATE TRIGGER populate_reserva_reading AFTER INSERT ON paga
FOR EACH ROW
BEGIN
DECLARE nif int;
DECLARE location_id int;
DECLARE time_id int;
DECLARE date_id int;
DECLARE montante_pago decimal(10,2);
DECLARE duracao int;

SET @nif = (select a.nif
            from aluga a
            where a.numero = NEW.numero);
SET @location_id = (SELECT l.location_id
                    FROM location_dimension l
                    NATURAL JOIN oferta
                    NATURAL JOIN aluga a
                    WHERE a.numero = NEW.numero);
SET @time_id = (select t.time_id
                from time_dimension t
                NATURAL JOIN (
                    select TIME(p.data) time
                    FROM paga p
                    WHERE p.numero = NEW.numero) x
                WHERE HOUR(x.time) = t.hour_of_day AND MINUTE(x.time) = t.minute_of_hour);
SET @date_id = (select d.date_id
                from date_dimension d
                NATURAL JOIN (
                    select DATE(p.data) data
                    FROM paga p
                    WHERE p.numero = NEW.numero) x
                WHERE x.data = d.date_time);
SET @montante_pago = (SELECT o.tarifa*(DATEDIFF(o.data_fim, o.data_inicio)+1)
                    FROM oferta o
                    NATURAL JOIN aluga a
                    WHERE a.numero = NEW.numero);
SET @duracao= (SELECT DATEDIFF(o.data_fim, o.data_inicio)+1
                FROM oferta o
                NATURAL JOIN aluga a
                WHERE a.numero = NEW.numero);
insert into reserva_reading (nif, location_id, time_id, date_id, montante_pago, duracao_dias)
    values (@nif, @location_id, @time_id, @date_id, @montante_pago, @duracao);

END //
```

2.

Para escrever a consulta em OLAP consideramos os níveis espaço e posto da dimensão Localização e dia e mês da dimensão Data. De seguida verificamos as diferentes combinações que poderíamos ter entre estes 4 elementos (16 no total). Como o SQL não suporta o CUBE optámos por efetuar 4 ROLLUPs para a maioria das combinações, ficando-nos a faltar apenas duas de 2 elementos, por isso realizamos 2 GROUP BY para essas duas últimas combinações. Por fim unimos tudo através do UNION.
Temos também numa coluna referente à média do valor pago sobre as dimensões localização e data.

```
SELECT codigo_espaco, codigo, date_month_number, date_day_of_month_number, AVG(montante_pago)
FROM location_dimension
NATURAL JOIN reserva_reading
NATURAL JOIN date_dimension
GROUP BY codigo_espaco, codigo, date_month_number, date_day_of_month_number with ROLLUP
UNION
SELECT codigo_espaco, codigo,
CASE WHEN date_month_number IS NOT NULL THEN NULL END date_month_number, date_day_of_month_number, AVG(montante_pago)
FROM location_dimension
NATURAL JOIN reserva_reading
NATURAL JOIN date_dimension
GROUP BY date_day_of_month_number, codigo, codigo_espaco with ROLLUP
UNION
SELECT codigo_espaco,
CASE WHEN codigo IS NOT NULL THEN NULL END codigo, date_month_number, date_day_of_month_number, AVG(montante_pago)
FROM location_dimension
NATURAL JOIN reserva_reading
NATURAL JOIN date_dimension
GROUP BY date_month_number, date_day_of_month_number, codigo_espaco with ROLLUP
UNION
SELECT
CASE WHEN codigo_espaco IS NOT NULL THEN NULL END codigo_espaco, codigo, date_month_number, date_day_of_month_number, AVG(montante_pago)
FROM location_dimension
NATURAL JOIN reserva_reading
NATURAL JOIN date_dimension
GROUP BY codigo, date_month_number, date_day_of_month_number with ROLLUP
UNION
SELECT codigo_espaco,
CASE WHEN codigo IS NOT NULL THEN NULL END codigo,
CASE WHEN date_month_number IS NOT NULL THEN NULL END date_month_number, date_day_of_month_number, AVG(montante_pago)
FROM location_dimension
NATURAL JOIN reserva_reading
NATURAL JOIN date_dimension
GROUP BY codigo_espaco, date_day_of_month_number with ROLLUP
UNION
SELECT codigo_espaco,
CASE WHEN codigo IS NOT NULL THEN NULL END codigo, date_month_number,
CASE WHEN date_day_of_month_number IS NOT NULL THEN NULL END date_day_of_month_number, AVG(montante_pago)
FROM location_dimension
NATURAL JOIN reserva_reading
NATURAL JOIN date_dimension
GROUP BY codigo_espaco, date_month_number with ROLLUP ORDER BY codigo_espaco, codigo;
```