
Projecto de Bases de Dados, Parte 4

Grupo: 66

Turno: Quinta-feira, 11h00-12h30

Professor: Gabriel Pestana



82304 - André Gonçalo Brandão Mendonça (11 horas de esforço estimado)

82303 - Gonçalo Castanheira Ribeiro (11 horas de esforço estimado)

81183 - Alexandre da Silva Machado (11 horas de esforço estimado)

Índices

- a) Indique, justificando, que tipo de índice(s), sobre que atributo(s) e sobre que tabela(s) faria sentido criar de modo a acelerar a execução destas interrogações.
- b) Crie o(s) índice(s) em SQL, se necessário. Examine o plano de execução obtido para cada uma das queries e justifique.

É possível verificar através do mysql que existem algumas tabelas das queries referidas no enunciado do projeto (1 e 2), que contêm índices por omissão, pois cada vez que é adicionada uma primary key, é automaticamente criado um índice sobre essas colunas (BTREE por omissão), tal como podemos observar neste exemplo referente à tabela "User".

A versão do MySQL no sigma não suporta índices Hash, logo a melhor solução para ambas as

```
mysql> show index from user;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| user  | 0          | PRIMARY | 1            | nif         | A         | 11          | NULL    | NULL   | NULL | BTREE      |          |
| user  | 0          | nif     | 1            | nif         | A         | 11          | NULL    | NULL   | NULL | BTREE      |          |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> describe user;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| nif   | varchar(9)    | NO   | PRI | NULL    |       |
| nome  | varchar(80)   | NO   |     | NULL    |       |
| telefone | varchar(26) | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

queries será a utilização do índice B Tree.

Query 1)

Descrição: Quais utilizadores cujos espaços foram fiscalizados sempre pelo mesmo fiscal?

SQL: `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`

Índices a aplicar em atributos de tabelas: Utilizar Btree e aplicar índice primário e ordenado (clustered) em A.nif, para que assim não existam duplicados sendo que os índices aplicar índices secundários seriam A.morada/codigo e F.morada/codigo para que possa existir duplicados.

Plano de execução: Após a implementação dos índices e inúmeros testes, concluímos que esta seria a melhor maneira de melhorar a performance, sendo que os tempos registados foram:

- 0.51511900 (antes da introdução de índices e sem chaves primárias)
- 0.00017400 (após introdução de índices e chaves primárias)

Query 2)

Descrição: Quais os espaços com postos que nunca foram alugados?

SQL: `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')`

Índices a aplicar em atributos de tabelas: Utilizar Btree agrupada, utilizando índice composto <P.morada,P.codigo>, sendo que a ordem não interessa para os dois atributos. Para o estado utilizamos da tabela Posto, dado que as condições das mesmas são seletivas. É necessário criar também um índice Btree agrupado em estado, pois é pouco seletiva mas a interrogação é frequente, conseguindo assim uma maior rapidez quando queremos saber se o espaço x com posto y nunca foi alugado.

Plano de execução: Após a implementação dos índices e inúmeros testes, concluímos que esta seria a melhor maneira de melhorar a performance, sendo que os tempos registados foram:

- 0.02934600 (antes da introdução de índices e sem chaves primárias)
- 0.00031600 (após introdução de índices e chaves primárias)

Data Warehouse

1. Crie na base de dados o esquema de uma estrela com informação sobre reservas (montante pago e duração em dias) tendo como dimensões:

`DROP TABLE IF EXISTS facts; DROP TABLE IF EXISTS dim_date; DROP TABLE IF EXISTS dim_time;
DROP TABLE IF EXISTS dim_user; DROP TABLE IF EXISTS dim_location; DROP PROCEDURE IF EXISTS fillDates;
DROP PROCEDURE IF EXISTS fillTimes; DROP PROCEDURE IF EXISTS fillLocations;
DROP PROCEDURE IF EXISTS fillFacts; DROP FUNCTION IF EXISTS calcAverage;`

a. Utilizador que reservou

`CREATE TABLE IF NOT EXISTS `dim_user` (
`id` INT NOT NULL UNIQUE AUTO_INCREMENT,
`nif` VARCHAR(9) NOT NULL UNIQUE,
PRIMARY KEY (`id`)) AS SELECT `nif` FROM `user`;`

b. Localização (com a hierarquia posto, espaço, edifício)

`CREATE TABLE IF NOT EXISTS `dim_location` (
`id` INT NOT NULL UNIQUE AUTO_INCREMENT,`

```
`edificio` VARCHAR(255) NOT NULL, `espaco` VARCHAR(255) NOT NULL,
`posto` VARCHAR(255) NOT NULL,
PRIMARY KEY(`id`));
```

- c. Tempo, com a hora e minuto a que foi efetuado o pagamento (esta dimensão deve conter todos os minutos de um dia)**

```
CREATE TABLE IF NOT EXISTS `dim_time` (
  `id` INT NOT NULL UNIQUE AUTO_INCREMENT,
  `hour` INT NOT NULL, `minute` INT NOT NULL,
  PRIMARY KEY (`id`));
```

- d. Data, com a data a que foi efetuado o pagamento (esta dimensão deve conter todos os dias dos anos 2016 e 2017 com a hierarquia, dia, semana, mês, semestre e ano)**

```
CREATE TABLE IF NOT EXISTS `dim_date` (
  `id` INT NOT NULL UNIQUE AUTO_INCREMENT,
  `day` INT NOT NULL,
  `month` INT NOT NULL, `year` INT NOT NULL, `week` INT NOT NULL, `semester` INT NOT NULL,
  PRIMARY KEY (`id`));
```

```
-- Tabela de factos com as dimensoes definidas --
```

```
CREATE TABLE IF NOT EXISTS `facts` (
  `id` INT UNIQUE NOT NULL AUTO_INCREMENT,
  `uid` INT NOT NULL, `lid` INT NOT NULL, `tid` INT NOT NULL, `did` INT NOT NULL,
  `paid` NUMERIC(19, 4) NOT NULL, `leaseTime` INT NOT NULL,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`uid`) REFERENCES `dim_user` (`id`),
  FOREIGN KEY (`lid`) REFERENCES `dim_location` (`id`),
  FOREIGN KEY (`tid`) REFERENCES `dim_time` (`id`),
  FOREIGN KEY (`did`) REFERENCES `dim_date` (`id`));
```

```
DELIMITER $
```

```
CREATE PROCEDURE fillDates(dateStart DATE, dateEnd DATE)
```

```
BEGIN
```

```
  DECLARE semesterNum INT; DECLARE weekNum INT; DECLARE cnt INT;
```

```
  WHILE dateStart <= dateEnd DO
```

```
    IF MONTH(dateStart) = 1 AND DAY(dateStart) = 1 THEN SET cnt = 0; SET weekNum = 1; END IF;
```

```
    IF cnt > 0 AND cnt % 7 = 0 THEN SET weekNum = weekNum + 1; END IF;
```

```
    IF(MONTH(dateStart) < 7) THEN SET semesterNum = 1; ELSE SET semesterNum = 2; END IF;
```

```
    INSERT INTO `dim_date` (`day`, `month`, `year`, `week`, `semester`)
```

```
    VALUES (DAY(dateStart), MONTH(dateStart), YEAR(dateStart), weekNum, semesterNum);
```

```
    SET dateStart = date_add(dateStart, INTERVAL 1 DAY);
```

```
    SET cnt = cnt + 1;
```

```
  END WHILE;
```

```
END;
```

```
$
```

```
CREATE PROCEDURE fillTimes(timeStart DATETIME, timeEnd DATETIME)
```

```
BEGIN
```

```
  WHILE timeStart <= timeEnd DO
```

```
    INSERT INTO `dim_time` (`hour`, `minute`) VALUES (HOUR(timeStart), MINUTE(timeStart));
```

```
    SET timeStart = date_add(timeStart, INTERVAL 1 MINUTE);
```

```
  END WHILE;
```

```

END;
$

CREATE PROCEDURE fillLocations()
BEGIN
    INSERT INTO `dim_location` (`edificio`, `espaco`, `posto` SELECT `morada`, `codigo`, `codigo` FROM `espaco`;

    INSERT INTO `dim_location` (`edificio`, `espaco`, `posto`) SELECT `morada`, `codigo_espaco`, `codigo`
    FROM `posto`;
END;
$

CREATE PROCEDURE fillFacts()
BEGIN
    DECLARE num VARCHAR(255); DECLARE address VARCHAR(255); DECLARE identifier VARCHAR(255);
    DECLARE startDate TIMESTAMP; DECLARE endDate TIMESTAMP; DECLARE price NUMERIC(19, 4);
    DECLARE nif VARCHAR(9); DECLARE payDate TIMESTAMP; DECLARE method VARCHAR(255);
    DECLARE uid INT; DECLARE lid INT; DECLARE tid INT; DECLARE did INT; DECLARE leaseTime INT;
    DECLARE done INT DEFAULT FALSE;
    DECLARE cur CURSOR FOR SELECT * FROM `oferta` NATURAL JOIN `aluga` NATURAL JOIN `paga`;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;
    dateLoop: LOOP
        FETCH cur INTO num, address, identifier, startDate, endDate, price, nif, payDate, method;

        IF done THEN LEAVE dateLoop; END IF;

        SET uid = (SELECT du.`id` FROM `dim_user` du WHERE du.`nif` = nif);
        SET lid = (SELECT dl.`id` FROM `dim_location` dl WHERE dl.`edificio` = address
        AND ((dl.`espaco` = identifier AND dl.`posto` = identifier) OR (dl.`espaco` != identifier AND dl.`posto` = identifier)));

        SET tid = (SELECT dt.`id` FROM `dim_time` dt WHERE dt.`hour` = HOUR(payDate) AND dt.`minute` =
        MINUTE(payDate));

        SET did = (SELECT dd.`id` FROM `dim_date` dd WHERE dd.`day` = DAY(payDate) AND dd.`month` =
        MONTH(payDate) AND dd.`year` = YEAR(payDate));

        SET leaseTime = datediff(endDate, startDate);
        SET price = leaseTime * price;

        INSERT INTO `facts` (`uid`, `lid`, `tid`, `did`, `paid`, `leaseTime`)
        VALUES (uid, lid, tid, did, price, leaseTime);
    END LOOP;
    CLOSE cur;
END;
$

```

2. Considerando o esquema da estrela criado em (1), escreva uma consulta OLAP em SQL para obter o cubo com valor médio pago sobre as dimensões localização e data.

```

CREATE FUNCTION calcAverage (location INT, endDate INT) RETURNS DECIMAL(19, 4)
BEGIN
    DECLARE done INT DEFAULT FALSE; DECLARE newDate INT; DECLARE cnt DECIMAL(19, 4) DEFAULT 0;
    DECLARE val DECIMAL(19, 4); DECLARE average DECIMAL(19, 4) DEFAULT 0;
    DECLARE cur CURSOR FOR SELECT paid, did FROM facts WHERE lid = location;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

```

```

OPEN cur;
  fetchLoop: LOOP
    FETCH cur INTO val, newDate;
    IF done THEN LEAVE fetchLoop; END IF;

    IF newDate <= endDate THEN SET average = average + val; SET cnt = cnt + 1; END IF;
  END LOOP;
CLOSE cur;

RETURN average / cnt;
END;
$
DELIMITER ;
CALL fillDates('2016-01-01','2017-12-31'); -- fills date dimension
CALL fillTimes('1970-01-01 00:00:00', '1970-01-01 23:59:59'); -- fills time dimension date is irrelevant but convenient
CALL fillLocations(); -- fills space dimension with currently available spaces and workstations
CALL fillFacts(); -- fills facts tables with currently paid reservations

SET @olapSql = NULL;
SELECT GROUP_CONCAT(DISTINCT concat("CASE WHEN did = \"", did, "\" THEN paid ELSE calcAverage(f.lid, ", did, ")
END AS \"", (SELECT DATE(concat(d.year, "-", d.month, "-", d.day)) FROM dim_date d WHERE d.id = did), "\""))
INTO @olapSql
FROM facts;

SET @olapSql = concat("SELECT CASE WHEN l.espaco = l.posto THEN concat(l.edificio, \" - \", l.espaco) ELSE
concat(l.edificio, \" - \", l.espaco, \" - \", l.posto) END AS location, \"\", @sql, \" FROM facts f
INNER JOIN dim_location l ON f.lid = l.id
INNER JOIN dim_date d ON f.did = d.id
GROUP BY lid;");

SELECT @olapSql; PREPARE stmt FROM @olapSql; EXECUTE stmt; DEALLOCATE PREPARE stmt;

```