Projeto Bases de Dados

2016/2017

**4ª feira 8h30 - Grupo 9**

**81900** – Nuno Anselmo

**81936** – Liliana Oliveira

**82047** – André Mendes

Parte 4

Esforço dedicado:

**81900 – 8h**

**81936 – 8h**

**82047 – 8h**

Índices

1. **Primeira query :**

O índice mais indicado seria um índice B-Tree sobre os campos morada e código nas tabelas fiscaliza e arrenda, para acelerar o *INNER JOIN*.

**Segunda query:**

Para a tabela estado, um índice nos campos *estado* e *numero*, de tipo B-Tree, pois um índice do tipo Hash não permite matches parciais da chave. Este índice permitiria realizar uma match parcial do número, e com isso encontrar o estado, sendo possível realizar o *NATURAL JOIN* (e depois o *WHERE* com o estado) apenas com o índice.

Para a tabela aluga, um índice sobre os campos *morada*, *codigo* e *numero*, novamente B-Tree e pela mesma razão: permitir que a query “qual o número a que está associado um tuplo (morada, código)” seja realizada apenas sobre o índice.

Para a tabela posto, um índice nos campos *morada*, *codigo*, e *codigo\_espaco*, novamente B-Tree, para acelerar o join e permitir que toda a query seja feita apenas sobre os índices.

Com estes três índices, toda a query pode ser realizada sobre os índices, sem aceder a blocos em disco.

1. Não será necessário criar index para a primeira query porque o MySQL cria índices por omissão sobre a chave primária de cada tabela, que foi a proposta de índice indicada acima. Executando a query com *EXPLAIN*, os índices estão a ser usados.

Para a segunda query, os seguintes índices devem ser criados:

CREATE INDEX estado\_numero ON estado(estado,numero);

CREATE INDEX morada\_codigo\_numero ON aluga(morada,codigo,numero);

CREATE UNIQUE INDEX morada\_codigo\_cespaco ON posto(morada,codigo,codigo\_espaco);

Executando a query com *EXPLAIN* os índices não estão a ser usados mas aparecem como “*possible\_keys*”, indicando que foi feita uma escolha por parte do SGBD de não os usar apesar de estes estarem presentes e serem relevantes para a query. Suspeitamos que isto seja devido a uma baixa cardinalidade no nosso ambiente de testes, em que operar sobre a tabela é tão ou mais rápido do que operar sobre os índices.

Data Warehouse

1.

DELIMITER //

# Parses a day, month and year to a mysql date type.

DROP FUNCTION IF EXISTS to\_date //

CREATE FUNCTION to\_date(day INT, month INT, year INT)

RETURNS DATE

BEGIN

RETURN DATE(CONCAT(year, '-', month, '-', day));

END //

# Generates the date dimension, with all the days of the years 2016 and 2017.

DROP PROCEDURE IF EXISTS load\_date\_dimension //

CREATE PROCEDURE load\_date\_dimension()

BEGIN

DECLARE full\_date DATETIME;

SET full\_date = '2016-01-01 00:00:00';

WHILE full\_date < '2018-01-01 00:00:00' DO

INSERT INTO date\_dimension (date\_id, dia, semana, mes, semestre, ano) VALUES (

YEAR(full\_date) \* 10000 + MONTH(full\_date) \* 100 + DAY(full\_date),

DAY(full\_date),

# Week starting with a sunday and range from 1-52

WEEK(full\_date, 2),

MONTH(full\_date),

# If the month is lesser then the 7th month, its first semester, otherwise second.

IF(MONTH(full\_date) < 7, 1, 2),

YEAR(full\_date)

);

SET full\_date = DATE\_ADD(full\_date, INTERVAL 1 DAY);

END WHILE;

END;

//

# Generates the time dimension, with all minutes of the day, since 00:00 until 23:59

DROP PROCEDURE IF EXISTS load\_time\_dimension //

CREATE PROCEDURE load\_time\_dimension()

BEGIN

DECLARE full\_day DATETIME;

SET full\_day = '2016-01-01 00:00:00';

WHILE full\_day < '2016-01-01 23:59:59' DO

INSERT INTO time\_dimension (time\_id, hora, minuto) VALUES (

HOUR(full\_day) \* 100 + MINUTE(full\_day),

HOUR(full\_day),

MINUTE(full\_day)

);

SET full\_day = DATE\_ADD(full\_day, INTERVAL 1 MINUTE);

END WHILE;

END //

# Loads all users into the user dimension.

DROP PROCEDURE IF EXISTS load\_user\_dimension //

CREATE PROCEDURE load\_user\_dimension()

BEGIN

INSERT INTO user

SELECT

nif,

nome,

telefone

FROM proj.user;

END //

# Loads all the locations into the local dimension.

# All workspaces will have the cod\_posto as null.

DROP PROCEDURE IF EXISTS load\_local\_dimension //

CREATE PROCEDURE load\_local\_dimension()

BEGIN

# Unions all workspaces and workstations.

INSERT INTO local\_dimension

SELECT

CONCAT(morada, codigo\_espaco, IFNULL(codigo\_posto, '')) AS local\_id,

codigo\_espaco AS cod\_espaco,

codigo\_posto AS cod\_posto,

morada AS cod\_edificio

FROM ((SELECT

morada,

codigo AS codigo\_espaco,

NULL codigo\_posto

FROM proj.espaco)

UNION ALL (SELECT

morada,

codigo\_espaco,

codigo AS codigo\_posto

FROM proj.posto)) AS local;

END //

DROP PROCEDURE IF EXISTS load\_reserva //

CREATE PROCEDURE load\_reserva()

BEGIN

# Declare all variables for the cursor fetching.

DECLARE fetched\_nif VARCHAR(9);

DECLARE fetched\_morada VARCHAR(255);

DECLARE fetched\_codigo\_espaco VARCHAR(255);

DECLARE fetched\_codigo\_posto VARCHAR(255);

DECLARE fetched\_data\_inicio DATE;

DECLARE fetched\_data\_fim DATE;

DECLARE fetched\_data\_pagamento DATETIME;

DECLARE fetched\_tarifa DECIMAL(19, 4);

# Declare all variables that will be used to create the reserva entry.

DECLARE fetched\_date\_id INT;

DECLARE fetched\_time\_id INT;

DECLARE fetched\_local\_id VARCHAR(765);

# Declare the cursor related variables.

# The query will give all the reservas that are rented by someone, even if they didn't pay

# for it yet.

DECLARE cursorDone INT DEFAULT FALSE;

DECLARE cursorReserva CURSOR FOR SELECT

nif,

morada,

codigo AS codigo\_espaco,

NULL AS codigo\_posto,

data\_inicio,

data\_fim,

data AS data\_pagamento,

tarifa

FROM proj.aluga

NATURAL JOIN proj.oferta

NATURAL JOIN proj.espaco

LEFT JOIN proj.paga ON paga.numero = aluga.numero

UNION ALL

SELECT

nif,

morada,

codigo\_espaco,

codigo AS codigo\_posto,

data\_inicio,

data\_fim,

data AS data\_pagamento,

tarifa

FROM proj.aluga

NATURAL JOIN proj.oferta

NATURAL JOIN proj.posto

LEFT JOIN proj.paga ON paga.numero = aluga.numero;

# Make the cursorDone variable go false once the cursor goes through all the records.

DECLARE CONTINUE HANDLER FOR NOT FOUND SET cursorDone = TRUE;

# Start looping.

OPEN cursorReserva;

reservaLoop: LOOP

# Set the cursor done false, since any SELECT INTO will activate the handler and turn the variable true.

SET cursorDone = FALSE;

# Fetch the next record.

FETCH cursorReserva

INTO fetched\_nif, fetched\_morada, fetched\_codigo\_espaco, fetched\_codigo\_posto, fetched\_data\_inicio, fetched\_data\_fim, fetched\_data\_pagamento, fetched\_tarifa;

# If there are no more records close the cursor and leave the loop.

IF cursorDone

THEN

CLOSE cursorReserva;

LEAVE reservaLoop;

END IF;

# Fetch the local dimension of the current record.

IF fetched\_codigo\_posto IS NULL

THEN

SELECT local\_id

INTO fetched\_local\_id

FROM local\_dimension

WHERE cod\_edificio = fetched\_morada AND cod\_espaco = fetched\_codigo\_espaco AND cod\_posto IS NULL;

ELSE

SELECT local\_id

INTO fetched\_local\_id

FROM local\_dimension

WHERE cod\_edificio = fetched\_morada AND cod\_espaco = fetched\_codigo\_espaco AND cod\_posto = fetched\_codigo\_posto;

END IF;

# Fill the whole date and time data for this reserva, since the start\_date until the end\_date.

# Also fill the remaining days and total\_pago as 0.

INSERT INTO reserva

SELECT

fetched\_nif AS nif,

date\_id,

time\_id,

fetched\_local\_id AS local\_id,

0 AS total\_pago,

(Datediff(fetched\_data\_fim, to\_date(dia, mes, ano))) AS duracao\_em\_dias

FROM time\_dimension, date\_dimension

WHERE to\_date(dia, mes, ano) BETWEEN fetched\_data\_inicio AND fetched\_data\_fim;

# If it was paid then update the total\_pago in the day and minute that it was paid.

IF fetched\_data\_pagamento IS NOT NULL

THEN

# Fetch the time dimension of when the reserva was paid.

SELECT time\_id

INTO fetched\_time\_id

FROM time\_dimension

WHERE hora = HOUR(fetched\_data\_pagamento) AND minuto = MINUTE(fetched\_data\_pagamento);

# Fetch the date dimension of when the reserva was paid.

SELECT date\_id

INTO fetched\_date\_id

FROM date\_dimension

WHERE

ano = YEAR(fetched\_data\_pagamento) AND mes = MONTH(fetched\_data\_pagamento) AND

dia = DAY(fetched\_data\_pagamento);

# Update the payment time, using the fields that we fetched.

UPDATE reserva

SET total\_pago = (Datediff(fetched\_data\_fim, fetched\_data\_inicio) + 1) \* fetched\_tarifa

WHERE

nif = fetched\_nif AND time\_id = fetched\_time\_id AND date\_id = fetched\_date\_id AND

local\_id = fetched\_local\_id;

END IF;

END LOOP;

END //

# Loads the data warehouse.

DROP PROCEDURE IF EXISTS load\_data\_warehouse //

CREATE PROCEDURE load\_data\_warehouse()

BEGIN

CALL load\_time\_dimension();

CALL load\_date\_dimension();

CALL load\_user\_dimension();

CALL load\_local\_dimension();

CALL load\_reserva();

END //

DELIMITER ;

2.

SELECT local\_id, date\_id , AVG(total\_pago)

FROM reserva

GROUP BY CUBE(local\_id,date\_id);