

# InstantOffice

Projecto de Bases de Dados – LEIC-A 2016/2017

## Parte 4 – Índices & *Data Warehouse*

Prof<sup>o</sup> Responsável: Gabriel Pestana

Turno: BD225179L09 – Grupo 2

81045	Rui Ventura	Tempo dedicado: 10h
81586	Diogo Freitas	Tempo dedicado: 3h
81700	Sara Azinhal	Tempo dedicado: 10h

# 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.

1. Quais os utilizadores cujos espaços foram fiscalizados sempre pelo mesmo fiscal?

Morada, Código – HASH: Desagrupado / Denso em Fiscaliza e Arrenda

Estes índices têm como objectivo acelerar a execução do **INNER JOIN** entre as tabelas Fiscaliza e Arrenda. São do tipo HASH de modo a permitirem um *scan* dos índices em vez de percorrer as tabelas com uma complexidade temporal constante.

nif – BTREE: Agrupado / Esparso em Arrenda

Estes índices têm como objectivo acelerar a execução do **GROUP BY**, pelo que faz sentido o índice ser do tipo BTREE, especialmente quando a diversidade de nifs é pequena, sendo usado índice para encontrar o alugável arrendado pelo utilizador com o nif correspondente.

2. Quais os espaços com postos que nunca foram alugados?

Morada, Código – HASH: Desagrupado / Denso em Posto e Aluga

Número – HASH: Desagrupado / Denso em Aluga e Estado

Estes índices têm como objectivo acelerar a execução dos **NATURAL JOINS** entre Posto, Aluga e Estado, por motivos similares aos expostos acima acerca do índice sobre Morada e Código nas tabelas Arrenda e Fiscaliza.

Estado – HASH: Desagrupado / Denso em Estado

Criamos este índice de modo a acelerar a execução do **WHERE E.estado = 'Aceite'**, pois é uma *point query*, ou seja, uma pesquisa pontual, encontrando-se na cláusula **WHERE** uma igualdade.

Morada, Código\_Espaço – HASH: Desagrupado / Denso em Posto

Criamos este índice de modo a acelerar a execução do **WHERE (morada, código\_espaco) NOT IN (...)**, tornando mais eficiente a correspondência dos tuplos <morada,código\_espaco> com os resultados da query aninhada.

- b) Crie o(s) índice(s) em SQL, se necessário. Examine o plano de execução obtido para cada uma das queries e justifique.

O MySQL cria automaticamente índices referentes a *Foreign* e *Primary Keys*. De forma a comparar os planos de execução com e sem índices, primeiro é necessário remover os índices de FKs:

Começamos por listar os nomes das restrições das chaves, usando “**SHOW CREATE TABLE <Table Name>**” (Nota: diferente de “**SHOW INDEX**”, onde aparece o nome do índice e não da restrição).

De seguida, eliminámos as retrições das FKs, usando “**ALTER TABLE <Table Name> DROP FOREIGN KEY <Foreign Key Constraint Name>**” e eliminámos os índices das FK, usando “**ALTER TABLE <Table Name> DROP KEY <Key Name>**”.

Consecutivamente, obtemos os planos e tempos de execução das duas queries. “**EXPLAIN <Query>**” para ambas, de forma a obter os seus planos de execução, “**SET PROFILING = 1**”, executamos as queries, “**SET PROFILING = 0**” e “**SHOW PROFILES \G**”, de forma a obter os seus tempos.

Finalmente, criámos, dos índices escolhidos na alínea anterior, aqueles que não estão associados a *Primary Keys*, usando “**CREATE INDEX <Index Name> ON <Table Name>(<Atributo(s)>)**”.

E obtemos os novos tempos e planos de execução das queries. (profiling.sql)

```
***** 1. row *****
Query_ID: 1
Duration: 0.00174900
Query: SELECT A.nif FROM Arrenda A INNER JOIN Fiscaliza F ON A.morada = F.morada
***** 2. row *****
Query_ID: 2
Duration: 0.00238800
Query: SELECT DISTINCT P.morada, P.codigo_espaco FROM Posto P WHERE P.estado = 'Aceite'
***** 3. row *****
Query_ID: 3
Duration: 0.00171600
Query: SELECT A.nif FROM Arrenda A INNER JOIN Fiscaliza F ON A.morada = F.morada
***** 4. row *****
Query_ID: 4
Duration: 0.00179200
Query: SELECT DISTINCT P.morada, P.codigo_espaco FROM Posto P WHERE P.estado = 'Aceite'
```

Tempos de execução

Query\_ID:

1. 1ª Query sem índices
2. 2ª Query sem índices
3. 1ª Query com índices
4. 2ª Query com índices

Planos de execução:

1ª Query (sem índices)

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	F	index	NULL	PRIMARY	518	NULL	9	Using index; Using temporary; Using filesort
1	SIMPLE	A	eq_ref	PRIMARY	PRIMARY	514	ist181045.F.morada, ist181045.F.codigo	1	

2ª Query (sem índices)

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	P	ALL	NULL	NULL	NULL	NULL	9	Using where; Using temporary
2	DEPENDENT SUBQUERY	A	index	PRIMARY	PRIMARY	785	NULL	13	Using index
2	DEPENDENT SUBQUERY	P	eq_ref	PRIMARY	PRIMARY	514	func, ist181045.A.codigo	1	Using where
2	DEPENDENT SUBQUERY	E	ref	PRIMARY	PRIMARY	257	ist181045.A.numero	1	Using where

### 1ª Query (com índices)

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	F	index	idx_fiscaliza_mc	idx_fiscaliza_mc	514	NULL	9	Using index; Using temporary; Using filesort
1	SIMPLE	A	eq_ref	PRIMARY	PRIMARY	514	ist181045.F.morada, ist181045.F.codigo	1	

### 2ª Query (com índices)

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	P	range	NULL	idx_posto_mce	514	NULL	3	Using where; Using index for group-by
2	DEPENDENT SUBQUERY	A	index	PRIMARY, idx_aluga_numero	idx_aluga_numero	257	NULL	13	Using index
2	DEPENDENT SUBQUERY	E	ref	PRIMARY, idx_estado_estado	PRIMARY	257	ist181045.A.numero	1	Using where
2	DEPENDENT SUBQUERY	P	eq_ref	PRIMARY, idx_posto_mce	PRIMARY	514	func, ist181045.A.codigo	1	Using where

É importante salientar que na versão do MySQL utilizada (v5.1), não existe a possibilidade de criar índices que não do tipo BTREE, pelo que não é possível criar índices HASH ou BITMAP.

Como podemos verificar, analisando os **EXPLAINs** acima:

1. O SGBD, após a criação do índice composto <morada, codigo> sobre Fiscaliza, toma a decisão de utilizar o mesmo, ao contrário do que acontece no caso em que não tem índice ou FK, considerando a chave primária. O índice sobre o nif não é utilizado e julgamos ser devido à cadência de registos que existem nas tabelas, uma vez que se encontram sub-populados. No entanto, se a tabela Arrenda contivesse imensos nif's, potencialmente diferentes, o índice proposto (BTREE Agrupado esparso) facilitaria e tornaria mais eficiente a execução da query.
2. Podemos verificar que dois dos três índices criados foram preferencialmente utilizados pelo SGBD, não tendo sido utilizado o índice sobre o estado da tabela Estado, julgando mais uma vez que se deve ao facto das tabelas conterem poucos registos.

## Data Warehouse

1. Escreva as instruções SQL necessárias para carregar o esquema em estrela a partir das tabelas existentes. Os registos das dimensões data e tempo devem ser gerados automaticamente.

Esquema em estrela – Anexo 1 (reservations\_star.sql)

População das dimensões data e tempo – Anexo 2 (populate\_procs.sql)

2. Considerando o esquema em estrela criado em (1), escreva uma consola OLAP em SQL para obter o cubo com valor médio pago sobre as dimensões localização e data, considerando apenas os níveis “espaço” e “posto” da dimensão localização e “dia” e “mês” da dimensão data.

Anexo 3 (olap\_query.sql)

# Anexo 1

```
CREATE TABLE olap_User_dim (  
    user_id INTEGER(9) NOT NULL UNIQUE AUTO_INCREMENT,  
    nif      VARCHAR(9) NOT NULL,  
    name     VARCHAR(80) NOT NULL,  
    phone    VARCHAR(26) NOT NULL,  
    PRIMARY KEY(user_id)  
);  
  
CREATE TABLE olap_Location_dim (  
    location_id INTEGER(9) NOT NULL UNIQUE AUTO_INCREMENT,  
    code_office VARCHAR(255),  
    code_space  VARCHAR(255),  
    address_building VARCHAR(255) NOT NULL,  
    PRIMARY KEY(location_id)  
);  
  
CREATE TABLE olap_Date_dim (  
    date_id      INTEGER(3) NOT NULL UNIQUE AUTO_INCREMENT,  
    date_day     INTEGER(2) NOT NULL,  
    date_week    INTEGER(2) NOT NULL,  
    date_month   INTEGER(2) NOT NULL,  
    date_semester INTEGER(1) NOT NULL,  
    date_year    INTEGER(4) NOT NULL,  
    PRIMARY KEY(date_id)  
);  
  
CREATE TABLE olap_Time_dim (  
    time_id      INTEGER(3) NOT NULL UNIQUE AUTO_INCREMENT,  
    time_minute  INTEGER(2) NOT NULL,  
    time_hour    INTEGER(2) NOT NULL,  
    PRIMARY KEY(time_id)  
);  
  
CREATE TABLE olap_Reservations (  
    location_id INTEGER(9) NOT NULL,  
    time_id      INTEGER(3) NOT NULL,  
    date_id      INTEGER(3) NOT NULL,  
    user_id      INTEGER(9) NOT NULL,  
    amount       DECIMAL(19,4) NOT NULL,  
    time_period  INTEGER(3) NOT NULL,  
    PRIMARY KEY(location_id, time_id, date_id),  
    FOREIGN KEY(location_id) REFERENCES olap_Location_dim(location_id),  
    FOREIGN KEY(time_id) REFERENCES olap_Time_dim(time_id),  
    FOREIGN KEY(date_id) REFERENCES olap_Date_dim(date_id),  
    FOREIGN KEY(user_id) REFERENCES olap_User_dim(user_id)  
);
```

## Anexo 2

```
DELIMITER $$
CREATE PROCEDURE load_date_dim()
BEGIN
    DECLARE v_full_date DATETIME;
    SET v_full_date = '2016-01-01 00:00:00';
    WHILE v_full_date < '2018-01-01 00:00:00' DO
        INSERT INTO olap_Date_dim (
            date_day,
            date_week,
            date_month,
            date_semester,
            date_year
        ) VALUES (
            DAY(v_full_date),
            WEEK(v_full_date),
            MONTH(v_full_date),
            CEIL(MONTH(v_full_date) / 6),
            YEAR(v_full_date)
        );
        SET v_full_date = DATE_ADD(v_full_date, INTERVAL 1 DAY);
    END WHILE;
END;
$$

CREATE PROCEDURE load_time_dim()
BEGIN
    DECLARE v_full_time TIME;
    SET v_full_time = '00:00:00.0000';
    WHILE v_full_time <= '23:59:00.0000' DO
        INSERT INTO olap_Time_dim (
            time_minute,
            time_hour
        ) VALUES (
            MINUTE(v_full_time),
            HOUR(v_full_time)
        );
        SET v_full_time = ADDTIME(v_full_time, '00:01:0.0000');
    END WHILE;
END;
$$
```

## Anexo 3

```
SELECT date_month, date_day, code_space, code_office, AVG(amount)
FROM olap_Reservations
    NATURAL JOIN olap_Date_dim
    NATURAL JOIN olap_Location_dim
GROUP BY date_month, date_day, code_space, code_office WITH ROLLUP
UNION
SELECT date_month, date_day, code_space, code_office, AVG(amount)
FROM olap_Reservations
    NATURAL JOIN olap_Date_dim
    NATURAL JOIN olap_Location_dim
GROUP BY date_day, code_space, code_office, date_month WITH ROLLUP
UNION
SELECT date_month, date_day, code_space, code_office, AVG(amount)
FROM olap_Reservations
    NATURAL JOIN olap_Date_dim
    NATURAL JOIN olap_Location_dim
GROUP BY code_space, code_office, date_month, date_day WITH ROLLUP
UNION
SELECT date_month, date_day, code_space, code_office, AVG(amount)
FROM olap_Reservations
    NATURAL JOIN olap_Date_dim
    NATURAL JOIN olap_Location_dim
GROUP BY code_office, date_month, date_day, code_space WITH ROLLUP
UNION
SELECT date_month, date_day, code_space, code_office, AVG(amount)
FROM olap_Reservations
    NATURAL JOIN olap_Date_dim
    NATURAL JOIN olap_Location_dim
GROUP BY date_month, code_space
UNION
SELECT date_month, date_day, code_space, code_office, AVG(amount)
FROM olap_Reservations
    NATURAL JOIN olap_Date_dim
    NATURAL JOIN olap_Location_dim
GROUP BY date_day, code_office
ORDER BY date_month, date_day, code_space, code_office;
```