

# Otimização de interrogações



Mestrado Integrado em Engenharia Informática e Computação

Tecnologias de Bases de Dados

Ano Letivo 2018/19, 2º Semestre

Francisco Tuna de Andrade - 201503481

João Paulo Damas - 201504088

Faculdade de Engenharia da Universidade do Porto  
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

27 de Março de 2019

# Conteúdo

<b>1</b>	<b>Sumário</b>	<b>3</b>
<b>2</b>	<b>Restrições</b>	<b>3</b>
<b>3</b>	<b>Índices</b>	<b>4</b>
<b>4</b>	<b>Perguntas</b>	<b>5</b>
4.1	Pergunta 1 . . . . .	5
4.1.1	<i>Query</i> SQL de resposta . . . . .	5
4.1.2	Tempos de execução . . . . .	6
4.1.3	Planos de execução . . . . .	6
4.1.4	Resultado da <i>query</i> . . . . .	7
4.1.5	Comentários . . . . .	7
4.2	Pergunta 2 . . . . .	8
4.2.1	<i>Query</i> SQL de resposta . . . . .	8
4.2.2	Tempos de execução . . . . .	8
4.2.3	Planos de execução . . . . .	9
4.2.4	Resultado da <i>query</i> . . . . .	10
4.2.5	Comentários . . . . .	10
4.3	Pergunta 3 . . . . .	11
4.3.1	<i>Query</i> SQL de resposta . . . . .	11
4.3.2	Tempos de execução . . . . .	11
4.3.3	Planos de execução . . . . .	12
4.3.4	Resultado da <i>query</i> . . . . .	14
4.3.5	Comentários . . . . .	14
4.4	Pergunta 4 . . . . .	15
4.4.1	<i>Query</i> SQL de resposta . . . . .	15
4.4.2	Tempos de execução . . . . .	16
4.4.3	Planos de execução . . . . .	16
4.4.4	Resultado da <i>query</i> . . . . .	20
4.4.5	Comentários . . . . .	20
4.5	Pergunta 5 . . . . .	21
4.5.1	<i>Query</i> SQL de resposta . . . . .	21
4.5.2	Tempos de execução . . . . .	21
4.5.3	Planos de execução . . . . .	21
4.5.4	Resultado da <i>query</i> . . . . .	22
4.5.5	Comentários . . . . .	22
4.6	Pergunta 6 . . . . .	22
4.6.1	<i>Query</i> SQL de resposta . . . . .	23
4.6.2	Tempos de execução . . . . .	23
4.6.3	Planos de execução . . . . .	24
4.6.4	Resultado da <i>query</i> . . . . .	27
4.6.5	Comentários . . . . .	27
<b>5</b>	<b>Conclusões</b>	<b>28</b>

# 1 Sumário

No âmbito da cadeira de Tecnologia de Bases de Dados do Mestrado Integrado em Engenharia Informática e Computação, no ano letivo 2018/2019, este trabalho tem como objetivo analisar diferentes planos de execução de interrogações SQL decorrentes da presença de índices e diversas estruturas das interrogações a executar.

Começaremos por apresentar uma proposta de um conjunto de restrições e de um conjunto de índices que cremos irão melhorar a performance das queries em análise. De seguida, para cada questão, vamos mostrar o código SQL (com os nomes das tabelas originais, para evitar repetição) que cremos ser o mais adequado para responder à query, e, também, verificar se as nossas propostas de restrições e de índices melhoraram a sua performance, através da análise dos tempos de execução (a serem medidos 5x por teste, para evitar medições singulares falaciosas) e do custo e estratégia do plano de execução calculado pelo otimizador.

## 2 Restrições

Relativamente a restrições, aplicadas nas cópias Y e Z das tabelas originais, foram implementadas as chaves primárias e estrangeiras descritas no enunciado. O código abaixo representa o script de criação desta restrições, sendo que as chaves primárias seguem uma convenção de nomenclatura `<table_name>_pk`, enquanto que as chaves estrangeiras seguem o formato `<table_name>_fk_<referenced_table>`.

```
-- primary keys
ALTER TABLE ylics ADD CONSTRAINT ylics_pk PRIMARY KEY (codigo);
ALTER TABLE ycands ADD CONSTRAINT ycands_pk PRIMARY KEY (bi,curso,ano_lectivo);
ALTER TABLE yalus ADD CONSTRAINT yalus_pk PRIMARY KEY (numero);

ALTER TABLE zlics ADD CONSTRAINT zlics_pk PRIMARY KEY (codigo);
ALTER TABLE zcands ADD CONSTRAINT zcands_pk PRIMARY KEY (bi,curso,ano_lectivo);
ALTER TABLE zalus ADD CONSTRAINT zalus_pk PRIMARY KEY (numero);

-- foreign keys
ALTER TABLE ycands ADD CONSTRAINT ycands_fk_curso
    FOREIGN KEY (curso) REFERENCES ylics (codigo);
ALTER TABLE yalus ADD CONSTRAINT yalus_fk_curso
    FOREIGN KEY (curso) REFERENCES ylics(codigo);
ALTER TABLE yalus ADD CONSTRAINT yalus_fk_cands
    FOREIGN KEY (bi, curso, a_lect_matricula) REFERENCES ycands(bi, curso, ano_lectivo);

ALTER TABLE zcands ADD CONSTRAINT zcands_fk_curso
    FOREIGN KEY (curso) REFERENCES zlics (codigo);
ALTER TABLE zalus ADD CONSTRAINT zalus_fk_curso
    FOREIGN KEY (curso) REFERENCES zlics(codigo);
ALTER TABLE zalus ADD CONSTRAINT zalus_fk_cands
    FOREIGN KEY (bi, curso, a_lect_matricula) REFERENCES zcands(bi, curso, ano_lectivo);
```

### 3 Índices

De forma a tentar otimizar as *queries* efetuadas, foram decididos os seguintes índices:

#### Pergunta 1

- Índice do tipo B-tree na tabela *zalus* (estado, a\_lect\_conclusao - a\_lect\_matricula, curso)
- Índice unique do tipo B-tree na tabela *zlics* (codigo, sigla)

Justificação: Para esta query foram criados índices em todos os campos acedidos por ela através de expressões *join* e *where*. Note-se que a ordem dos campos no índice é importante e contribui para o baixo custo da query na opção Z. A query começa por procurar uma instância de *zalus* onde *estado* = 'EIC' e *a\_lect\_conclusao* - *a\_lect\_matricula* < 5 e, de seguida, uma instância de *zlics* onde *zlics.codigo* = *zalus.curso* e onde *zlics.sigla* = 'EIC'. Ou seja a ordem dos campos nestes índices mutlicolunas foi pensada de forma a corresponder à sua ordem de acesso pela query.

Por último, optou-se por um índice B-tree ao invés de um Bitmap, porque todos os campos usados nos índices têm uma variabilidade alta.

#### Pergunta 2

- Índice unique do tipo B-tree na tabela *zalus* (curso, a\_lect\_matricula, bi)

Justificação: A chave estrangeira é um conjunto único na tabela de alunos (um aluno entrou graças a uma candidatura para aquela ocorrência), pelo que o índice corresponde quase à criação dessa restrição. Para além disso, a junção *alus-cands* é realizada frequentemente (com *curso* e *a\_lect\_matricula* a serem as colunas referenciadas mais frequentemente).

#### Pergunta 3

- Índice unique do tipo B-tree na tabela *zalus* (curso, a\_lect\_matricula, bi)

Justificação: Índice correspondente a uma restrição que um dado aluno só pode, num determinado ano, inscrever-se num determinado curso uma vez. Este índice diminui substancialmente tanto o custo da query com subquery constante como o custo da query com subquery variável, devido ao facto de permitir procurar por uma instância da tabela *zalus* pelos campos do índice (*curso*, *a\_lect\_matricula* e *bi*) de forma muito mais eficiente.

Optou-se por um índice B-tree, ao invés de um Bitmap, porque todos os campos usados no índice têm uma variabilidade alta.

#### Pergunta 4

- Índice do tipo Bitmap na tabela *zalus* (estado, curso, a\_lect\_conclusao, med\_final)

Justificação: Estado é uma coluna com pouca variabilidade (apenas 2 valores distintos) e, por isso, bom candidato para um índice bitmap. As restantes colunas são frequentemente seleccionadas em *queries* envolventes, logo, o índice evita a necessidade de recorrer à tabela principal (e, por outro lado, um índice apenas na coluna estado não estava a ser considerado pelo otimizador).

## Pergunta 5

- Índice do tipo B-tree na tabela *zcands* (resultado)
- Índice do tipo Bitmap na tabela *zcands* (resultado)

Justificação: Nesta pergunta usámos os índices que foram sugeridos no guião. A análise de qual deles é mais eficaz será feita na secção de Comentários à questão 5.

## Pergunta 6

- Índice do tipo Bitmap na tabela *zcands* (resultado, curso, ano\_lectivo).

Justificação: Tal como na pergunta 4, resultado é uma columnna com baixa cardinalidade e curso e ano\_lectivo são acedidos frequentemente ao filtrar por resultado, perfazendo então uma versão *light* da tabela.

## Código

Abaixo encontra-se o código para (re)criar os índices:

```
-- Question 1
CREATE INDEX q1_alus ON zalus(estado, a_lect_conclusao - a_lect_matricula, curso);
CREATE UNIQUE INDEX q1_lics ON zlics(codigo, sigla);

-- Question 2 / 3
CREATE UNIQUE INDEX q2q3_alus ON zalus (curso, a_lect_matricula, bi);

-- Question 4
CREATE BITMAP INDEX q4 ON zalus (estado, curso, a_lect_conclusao, med_final);

-- Question 5
CREATE INDEX q5_btree_cands ON zcands(resultado);
CREATE BITMAP INDEX g5_bitmap_cands ON zcands(resultado);

-- Question 6
CREATE BITMAP INDEX q6 ON zcands (resultado, curso, ano_lectivo);
```

## 4 Perguntas

### 4.1 Pergunta 1

*Qual a lista dos números dos alunos especiais, que terminaram o curso com sigla EIC em menos de cinco anos, e quantos anos demoraram?*

#### 4.1.1 Query SQL de resposta

```
SELECT a.numero AS "Aluno (BI)", a.a_lect_conclusao - a.a_lect_matricula AS "Anos Demorados"
FROM alus a JOIN lics l ON a.curso = l.codigo
WHERE a.estado = 'C' AND a.a_lect_conclusao - a.a_lect_matricula < 5 AND l.sigla = 'EIC';
```

### 4.1.2 Tempos de execução

	X	Y	Z
	0.016	0.016	0.008
	0.031	0.015	0.015
	0.016	0.012	0.0
	0.016	0.0	0.015
	0.0	0.016	0.016
$\mu$	0.016	0.012	0.011

Tabela 1: Tempos (em segundos) da resposta da pergunta 1 nos diferentes ambientes.

### 4.1.3 Planos de execução

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			86	19
HASH JOIN			86	19
Access Predicates A.CURSO=L.CODIGO				
TABLE ACCESS	XLICS	FULL	1	3
Filter Predicates L.SIGLA='EIC'				
TABLE ACCESS	XALUS	FULL	942	16
Filter Predicates AND A.ESTADO='C' A.A_LECT_CONCLUSAO-A.A_LECT_MATRICULA<5				

Figura 1: Plano de execução da query para a pergunta 1, na cópia X.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			86	19
HASH JOIN			86	19
Access Predicates A.CURSO=L.CODIGO				
TABLE ACCESS	YLICS	FULL	1	3
Filter Predicates L.SIGLA='EIC'				
TABLE ACCESS	YALUS	FULL	942	16
Filter Predicates AND A.ESTADO='C' A.A_LECT_CONCLUSAO-A.A_LECT_MATRICULA<5				

Figura 2: Plano de execução da query para a pergunta 1, na cópia Y.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			21	3
NESTED LOOPS			21	3
INDEX	Q1_LICS	SKIP SCAN	1	1
Access Predicates				
L.SIGLA='EIC'				
Filter Predicates				
L.SIGLA='EIC'				
TABLE ACCESS	ZALUS	BY INDEX ROWID BATCHED	21	2
INDEX	Q1_ALUS	RANGE SCAN	4	1
Access Predicates				
AND				
A.ESTADO='C'				
A.CURSO=L.CODIGO				
A_LECT_CONCLUSAO-A_LECT_MATRICULA<5				
Filter Predicates				
A.CURSO=L.CODIGO				

Figura 3: Plano de execução da query para a pergunta 1, na cópia Z.

#### 4.1.4 Resultado da query

	Aluno (BI)	Anos Demorados
1	980509039	2
2	970509050	2
3	950509002	4
4	950509003	4
5	950509006	4
6	950509009	4
7	950509011	4
8	950509012	4
9	950509013	4
10	950509014	4
11	950509025	4
12	950509029	4
13	950509031	4
14	950509032	4
15	950509033	4
16	950509034	4
17	950509038	4
18	950509039	4
19	950509042	4
20	950509045	4

Figura 4: Parte do resultado da query de resposta da pergunta 1 (no total, o resultado contém 72 linhas).

#### 4.1.5 Comentários

Query simples, onde o plano de execução nas cópias X e Y é feito, sem nenhuma otimização, através de um FULL SCAN às duas tabelas em análise, *alus* e *lics*, seguido de um HASH JOIN para fazer *join*. Não se verificou um custo menor na cópia X do que na cópia Y, o que se deve ao facto de esta query não usar nenhuma das chaves primárias definidas.

No plano de execução na cópia Z, verificou-se uma redução substancial do custo em relação

às outras cópias, de 19 para 3. Pode-se constatar, no plano de execução para a cópia Z, que o otimizador seguiu uma estratégia de NESTED LOOPS para fazer o *join* entre as duas tabelas, devendo, no entanto, a diminuição do custo da query ter ficado a dever-se ao facto de o acesso à tabelas ser feito através dos índices, utilizando as estratégias de SKIP SCAN e RANGE SCAN.

Quanto aos tempos de execução, apesar da média dos tempos de execução na cópia Z ser inferior aos tempos de execução nas outras cópias, esta diferença não é significativa. Tal deve-se, provavelmente, ao facto de a quantidade de dados não ser grande o suficiente para se registarem diferenças nos tempos de execução.

## 4.2 Pergunta 2

*Qual a média mínima de candidatura em cada curso, em cada ano, dos alunos matriculados? Nem todas as candidaturas têm a média preenchida.*

### 4.2.1 Query SQL de resposta

```
SELECT c.curso, c.ano_lectivo AS ano, MIN(c.media) AS media
FROM cands c INNER JOIN alus a
ON c.bi = a.bi AND c.ano_lectivo = a.a_lect_matricula AND c.curso = a.curso
WHERE c.media IS NOT NULL
GROUP BY c.curso, c.ano_lectivo;
```

### 4.2.2 Tempos de execução

	X	Y	Z
	0.03	0.035	0.027
	0.026	0.028	0.021
	0.025	0.027	0.022
	0.026	0.025	0.025
	0.024	0.027	0.023
$\mu$	0.026	0.028	0.024

Tabela 2: Tempos (em segundos) da resposta da pergunta 2 nos diferentes ambientes.



### 4.2.3 Planos de execução

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			42	34
HASH				
HASH JOIN		GROUP BY	42	34
Access Predicates		SEMI	42	33
AND				
C.BI=A.BI				
C.ANO_LECTIVO=A.A_LECT_MATRICULA				
C.CURSO=A.CURSO				
TABLE ACCESS	XCANDS	FULL	3211	17
Filter Predicates				
C.MEDIA IS NOT NULL				
TABLE ACCESS	XALUS	FULL	9214	16

Figura 5: Plano de execução da query para a pergunta 2, na cópia X.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			42	34
HASH				
HASH JOIN		GROUP BY	42	34
Access Predicates		SEMI	42	33
AND				
C.BI=A.BI				
C.ANO_LECTIVO=A.A_LECT_MATRICULA				
C.CURSO=A.CURSO				
TABLE ACCESS	YCANDS	FULL	3211	17
Filter Predicates				
C.MEDIA IS NOT NULL				
TABLE ACCESS	YALUS	FULL	9214	16

Figura 6: Plano de execução da query para a pergunta 2, na cópia Y.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	18
HASH				
NESTED LOOPS		GROUP BY	1	18
TABLE ACCESS	ZCANDS	SEMI	1	17
Filter Predicates		FULL	3211	17
C.MEDIA IS NOT NULL				
INDEX	Q2_ALUS	UNIQUE SCAN	1	0
Access Predicates				
AND				
C.CURSO=A.CURSO				
C.ANO_LECTIVO=A.A_LECT_MATRICULA				
C.BI=A.BI				

Figura 7: Plano de execução da query para a pergunta 2, na cópia Z.

#### 4.2.4 Resultado da query

	CURSO	ANO	MEDIA
1	649	2000	11.33
2	433	2001	14.6
3	304	1993	12
4	433	2000	14.45
5	315	2000	10.83
6	233	2000	13.28
7	275	2001	10.2
8	255	1989	5.5
9	275	1998	10
10	233	1998	11
11	304	2000	10.9
12	233	2001	12
13	304	2002	12.23
14	255	1997	13
15	315	2001	11.15
16	233	2002	12.75
17	331	2002	14.33
18	1093	2002	12.68
19	255	1994	13
20	649	1998	13
21	304	1998	12

Figura 8: Parte do resultado da query de resposta da pergunta 2 (no total, o resultado contém 42 linhas).

#### 4.2.5 Comentários

A pergunta em questão era relativamente simples e os respectivos planos de execução refletem isso. Quer para a cópia X, quer para a Y, o otimizador opta por um realizar um HASH JOIN, visto que vê a necessidade de realizar um FULL SCAN a duas tabelas com alguma cardinalidade (aparentemente, as restrições de integridade em Y não influenciam a sua escolha). É visível o uso de um SEMI JOIN, apesar da inexistência de uma cláusula do tipo *(not) exists/in*, provavelmente devido aos resultados não conterem tuplos repetidos (graças ao agrupamento por curso/ano).

Na cópia Z, é de notar o reduzido custo (para quase metade) graças à adição do índice unique na chave estrangeira na tabela *zalus*. No entanto, a mudança mais relevante será a estratégia de junção: o otimizador passou a adotar uma estratégia de NESTED LOOPS. Na verdade, este é o comportamento esperado: a tabela *zalus*, com um índice unique, pode agora servir de (pequena!) tabela interna na junção e, ao ler cada tuplo na tabela externa (*zcands*), o lookup na tabela interna (base da estratégia nested loop) é imediato, pelo que esta abordagem se torna mais vantajosa.

Relativamente a tempos de execução, nota-se alguma melhoria em Z comparativamente às outras cópias, no entanto não parece ser extremamente significativa. Porém, a expectativa é que, com tabelas com mais dados, a diferença fosse mais notável.

### 4.3 Pergunta 3

Considere a questão de saber quantos candidatos aceites não se matricularam nesse ano lectivo. Compare uma formulação que use uma subpergunta constante com a equivalente que use uma subpergunta variável.

#### 4.3.1 Query SQL de resposta

```
/*
Question 3 (constant subquery)
*/
SELECT c.ano_lectivo AS "Ano Letivo", COUNT(*) AS "Número de Alunos não Matriculados"
FROM cands c WHERE c.resultado = 'C'
AND (c.bi, c.curso, c.ano_lectivo) NOT IN
    (SELECT a.bi, a.curso, a.a_lect_matricula FROM alus a)
GROUP BY c.ano_lectivo
ORDER BY c.ano_lectivo;
```

```
/*
Question 3 (variable subquery)
*/
/* X*/
SELECT c.ano_lectivo AS "Ano Letivo", COUNT(*) AS "Número de Alunos não Matriculados"
FROM cands c WHERE c.resultado = 'C'
AND NOT EXISTS (
    SELECT * FROM alus a
    WHERE a.bi = c.bi AND a.curso = c.curso AND a.a_lect_matricula = c.ano_lectivo
)
GROUP BY c.ano_lectivo
ORDER BY c.ano_lectivo;
```

#### 4.3.2 Tempos de execução

	X	Y	Z
	0.048	0.078	0.06
	0.047	0.043	0.047
	0.047	0.031	0.047
	0.047	0.039	0.047
	0.032	0.031	0.047
$\mu$	0.044	0.044	0.050

Tabela 3: Tempos (em segundos) da resposta da pergunta 3 nos diferentes ambientes (estratégia com subquery constante).

	X	Y	Z
	0.031	0.034	0.046
	0.016	0.031	0.047
	0.025	0.031	0.044
	0.023	0.016	0.062
	0.031	0.027	0.043
$\mu$	0.025	0.028	0.048

Tabela 4: Tempos (em segundos) da resposta da pergunta 3 nos diferentes ambientes (estratégia com subquery variável).

### 4.3.3 Planos de execução

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			3	36
SORT		GROUP BY	3	36
MERGE JOIN		ANTI NA	49	35
SORT		JOIN	4855	18
TABLE ACCESS	XCANDS	FULL	4855	17
Filter Predicates				
C.RESULTADO='C'				
SORT		UNIQUE	9214	17
Access Predicates				
AND				
INTERNAL_FUNCTION(C.BI)=INTERNAL_FUNCTION(A.BI)				
INTERNAL_FUNCTION(C.CURSO)=INTERNAL_FUNCTION(A.CURSO)				
INTERNAL_FUNCTION(C.ANO_LECTIVO)=INTERNAL_FUNCTION(A.A_LECT_MATRICULA)				
Filter Predicates				
AND				
INTERNAL_FUNCTION(C.ANO_LECTIVO)=INTERNAL_FUNCTION(A.A_LECT_MATRICULA)				
INTERNAL_FUNCTION(C.CURSO)=INTERNAL_FUNCTION(A.CURSO)				
INTERNAL_FUNCTION(C.BI)=INTERNAL_FUNCTION(A.BI)				
TABLE ACCESS	XALUS	FULL	9214	16

Figura 9: Plano de execução da query para a pergunta 3, na cópia X, na variante com subquery constante.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			9	36
SORT		GROUP BY	9	36
MERGE JOIN		ANTI NA	152	35
SORT		JOIN	4855	18
TABLE ACCESS	YCANDS	FULL	4855	17
Filter Predicates				
C.RESULTADO='C'				
SORT		UNIQUE	9214	17
Access Predicates				
AND				
INTERNAL_FUNCTION(C.BI)=INTERNAL_FUNCTION(A.BI)				
INTERNAL_FUNCTION(C.CURSO)=INTERNAL_FUNCTION(A.CURSO)				
INTERNAL_FUNCTION(C.ANO_LECTIVO)=INTERNAL_FUNCTION(A.A_LECT_MATRICULA)				
Filter Predicates				
AND				
INTERNAL_FUNCTION(C.ANO_LECTIVO)=INTERNAL_FUNCTION(A.A_LECT_MATRICULA)				
INTERNAL_FUNCTION(C.CURSO)=INTERNAL_FUNCTION(A.CURSO)				
INTERNAL_FUNCTION(C.BI)=INTERNAL_FUNCTION(A.BI)				
TABLE ACCESS	YALUS	FULL	9214	16

Figura 10: Plano de execução da query para a pergunta 3, na cópia Y, na variante com subquery constante.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			3	31
SORT		GROUP BY	3	31
MERGE JOIN		ANTI NA	49	30
TABLE ACCESS	ZCANDS	JOIN	4855	18
Filter Predicates		FULL	4855	17
C.RESULTADO='C'				
SORT		UNIQUE	9214	12
Access Predicates				
AND				
INTERNAL_FUNCTION(C.BI)=INTERNAL_FUNCTION(A.BI)				
INTERNAL_FUNCTION(C.CURSO)=INTERNAL_FUNCTION(A.CURSO)				
INTERNAL_FUNCTION(C.ANO_LECTIVO)=INTERNAL_FUNCTION(A.A_LECT_MATRICULA)				
Filter Predicates				
AND				
INTERNAL_FUNCTION(C.ANO_LECTIVO)=INTERNAL_FUNCTION(A.A_LECT_MATRICULA)				
INTERNAL_FUNCTION(C.CURSO)=INTERNAL_FUNCTION(A.CURSO)				
INTERNAL_FUNCTION(C.BI)=INTERNAL_FUNCTION(A.BI)				
INDEX	Q2Q3_ALUS	FAST FULL SCAN	9214	11

Figura 11: Plano de execução da query para a pergunta 3, na cópia Z, na variante com subquery constante.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			3	34
SORT		GROUP BY	3	34
HASH JOIN		ANTI	49	33
Access Predicates				
AND				
A.BI=C.BI				
A.CURSO=C.CURSO				
A.A_LECT_MATRICULA=C.ANO_LECTIVO				
TABLE ACCESS	XCANDS	FULL	4855	17
Filter Predicates				
C.RESULTADO='C'				
TABLE ACCESS	XALUS	FULL	9214	16

Figura 12: Plano de execução da query para a pergunta 3, na cópia X, na variante com subquery variável.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			9	34
SORT		GROUP BY	9	34
HASH JOIN		ANTI	152	33
Access Predicates				
AND				
A.BI=C.BI				
A.CURSO=C.CURSO				
A.A_LECT_MATRICULA=C.ANO_LECTIVO				
TABLE ACCESS	YCANDS	FULL	4855	17
Filter Predicates				
C.RESULTADO='C'				
TABLE ACCESS	YALUS	FULL	9214	16

Figura 13: Plano de execução da query para a pergunta 3, na cópia Y, na variante com subquery variável.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			3	18
SORT		GROUP BY	3	18
NESTED LOOPS		ANTI	49	17
TABLE ACCESS	ZCANDS	FULL	4855	17
Filter Predicates C.RESULTADO='C'				
INDEX	Q2Q3_ALUS	UNIQUE SCAN	9214	0
Access Predicates AND A.CURSO=C.CURSO A.A_LECT_MATRICULA=C.ANO_LECTIVO A.BI=C.BI				

Figura 14: Plano de execução da query para a pergunta 3, na cópia Z, na variante com subquery variável.

#### 4.3.4 Resultado da query

	Ano Letivo	Número de Alunos não Matriculados
1	1972	2
2	1973	2
3	1974	3
4	1975	8
5	1976	14
6	1977	3
7	1978	8
8	1979	12
9	1980	10
10	1981	31
11	1982	21
12	1983	16
13	1984	26
14	1985	29
15	1986	46
16	1987	60
17	1988	96
18	1989	145
19	1990	143
20	1991	222

Figura 15: Parte do resultado da query de resposta da pergunta 3 (no total, o resultado contém 31 linhas).

#### 4.3.5 Comentários

Nesta query, foi usada a expressão *not in* para a versão com subquery constante e a expressão *not exists* para a versão com subquery variável. Em ambas as versões, as cópias X e Y apresentam o mesmo custo, o que se deve ao facto de o otimizador não fazer nenhum acesso pelas chaves primárias definidas.

Em todas as cópias, a versão com subquery constante apresenta um maior custo de execução do que a versão com subquery variável. Atentando aos diferentes planos de execução recolhidos,

podemos constatar que a razão que contribui mais para esta diferença nos custos é a estratégia de *join* das tabelas *alus* e *cands*. No caso da subquery constante, esta estratégia é um MERGE JOIN para as 3 diferentes cópias, enquanto que, no caso da subquery variável, é usado o HASH JOIN nas cópias X e Y e o NESTED LOOPS na cópia Z.

A cópia Z apresenta custos menores nas duas versões. É de atentar, na versão com subquery variável, na redução do custo de 34 para 18 na cópia Z. Tal deve-se aos acessos às tabelas usando os índices definidos com estratégias de FAST FULL SCAN e UNIQUE SCAN.

Relativamente aos tempos de execução, as únicas diferenças significativas que se notou foi o facto de este tempo na versão com subquery variável ser nas cópias X e Y inferior aos restantes. Como esta pergunta incide sobre um número pequeno de dados, retornando apenas 31 linhas, podemos concluir que esta diferença se deve principalmente ao facto de, nas cópias X e Y, a query não ter de gastar tempo a lidar com índices. Acreditamos que se o número de dados fosse maior, os tempos de execução seguiriam a mesma tendência que o custo.

## 4.4 Pergunta 4

*Estude as tentativas de resposta à questão “Qual o curso com a melhor média de conclusão em cada ano lectivo”. Comente-as.*

### 4.4.1 Query SQL de resposta

```
-- primeira forma (utilizando uma subquery para filtrar as linhas onde o valor
↳ corresponde ao maximo)
WITH media_curso_ano AS (
SELECT curso, a_lect_conclusao AS ano, ROUND(AVG(med_final),2) AS media
FROM alus
WHERE estado = 'C'
GROUP BY curso, a_lect_conclusao)
SELECT *
FROM media_curso_ano t1
WHERE media = (SELECT MAX(media) FROM media_curso_ano t2 WHERE t2.ano = t1.ano);

-- segunda forma (utilizando left outer join com media inferior na juncao e filtrando
↳ por linhas com valores nulos, que corresponderao a linhas com o valor maximo)
WITH media_curso_ano AS (
SELECT curso, a_lect_conclusao AS ano, ROUND(AVG(med_final),2) AS media
FROM alus
WHERE estado = 'C'
GROUP BY curso, a_lect_conclusao)
SELECT t1.*
FROM media_curso_ano t1 LEFT OUTER JOIN media_curso_ano t2
ON t1.ano = t2.ano AND t1.media < t2.media
WHERE t2.ano IS NULL;
```

#### 4.4.2 Tempos de execução

	X	Y	Z
	0.024	0.023	0.02
	0.022	0.024	0.022
	0.025	0.023	0.025
	0.022	0.023	0.023
	0.023	0.023	0.022
$\mu$	0.023	0.023	0.022

Tabela 5: Tempos (em segundos) da resposta da pergunta 4 nos diferentes ambientes (estratégia subquery).

	X	Y	Z
	0.021	0.024	0.022
	0.022	0.027	0.021
	0.023	0.023	0.019
	0.026	0.022	0.021
	0.021	0.022	0.023
$\mu$	0.023	0.024	0.021

Tabela 6: Tempos (em segundos) da resposta da pergunta 4 nos diferentes ambientes (estratégia left join).

#### 4.4.3 Planos de execução

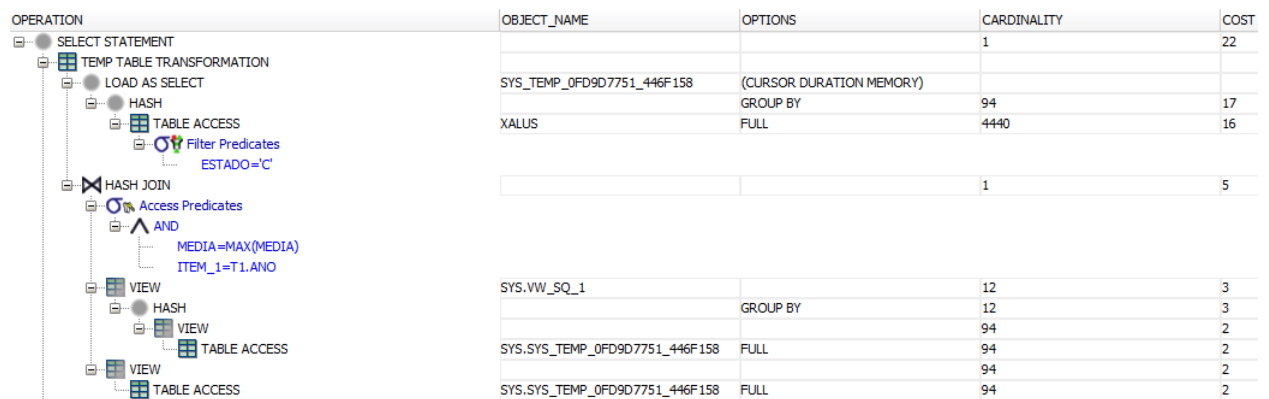


Figura 16: Plano de execução da query para a pergunta 4, na cópia X, na variante com subquery.

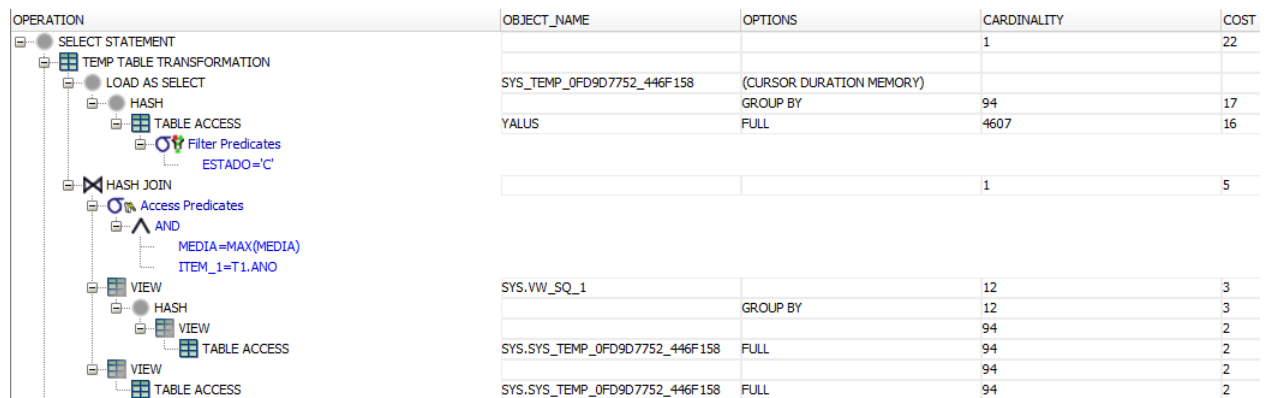


Figura 17: Plano de execução da query para a pergunta 4, na cópia Y, na variante com subquery.



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	10
TEMP TABLE TRANSFORMATION				
LOAD AS SELECT	SYS_TEMP_0FD9D7753_446F158	(CURSOR DURATION MEMORY)		
SORT		GROUP BY NOSORT	94	5
BITMAP CONVERSION		TO ROWIDS	4440	5
BITMAP INDEX	Q4	RANGE SCAN		
Access Predicates				
ESTADO='C'				
Filter Predicates				
ESTADO='C'				
HASH JOIN			1	5
Access Predicates				
AND				
MEDIA=MAX(MEDIA)				
ITEM_1=T1.ANO				
VIEW	SYS.VW_SQ_1		12	3
HASH		GROUP BY	12	3
VIEW			94	2
TABLE ACCESS	SYS.SYS_TEMP_0FD9D7753_446F158	FULL	94	2
VIEW			94	2
TABLE ACCESS	SYS.SYS_TEMP_0FD9D7753_446F158	FULL	94	2

Figura 18: Plano de execução da query para a pergunta 4, na cópia Z, na variante com subquery (utilizando CTE).

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	5
FILTER				
Filter Predicates				
ROUND(SUM(MED_FINAL)/COUNT(MED_FINAL),2) = (SELECT MAX(MEDIA) FROM (SELECT CURSO CURSO,A_LECT_CONCLUSAO ANO,ROUND(SUM(MED_FINAL)/COUNT(MED_FINAL),2) MEDIA FROM ZALUS ZALUS WHERE A_LECT_				
SORT		GROUP BY NOSORT	1	5
BITMAP CONVERSION		TO ROWIDS	4440	5
BITMAP INDEX	Q4	RANGE SCAN		
Access Predicates				
ESTADO='C'				
Filter Predicates				
ESTADO='C'				
SORT		AGGREGATE	1	
VIEW			8	5
SORT		GROUP BY	8	5
BITMAP CONVERSION		TO ROWIDS	370	5
BITMAP INDEX	Q4	RANGE SCAN		
Access Predicates				
ESTADO='C'				
Filter Predicates				
AND				
A_LECT_CONCLUSAO=:B1				
ESTADO='C'				

Figura 19: Plano de execução da query para a pergunta 4, na cópia Z, na variante com subquery (sem utilizar CTE).

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			94	21
TEMP TABLE TRANSFORMATION				
LOAD AS SELECT	SYS_TEMP_0FD9D7755_446F158	(CURSOR DURATION MEMORY)		
HASH		GROUP BY	94	17
TABLE ACCESS	XALUS	FULL	4440	16
Filter Predicates				
ESTADO='C'				
FILTER				
Filter Predicates				
T2.ANO IS NULL				
HASH JOIN		OUTER	94	4
Access Predicates				
T1.ANO=T2.ANO(+)				
Filter Predicates				
T1.MEDIA<T2.MEDIA(+)				
VIEW			94	2
TABLE ACCESS	SYS.SYS_TEMP_0FD9D7755_446F158	FULL	94	2
VIEW			94	2
TABLE ACCESS	SYS.SYS_TEMP_0FD9D7755_446F158	FULL	94	2

Figura 20: Plano de execução da query para a pergunta 4, na cópia X, na variante com left join.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			94	21
TEMP TABLE TRANSFORMATION				
LOAD AS SELECT	SYS_TEMP_0FD9D7757_446F158	(CURSOR DURATION MEMORY)		
HASH		GROUP BY	94	17
TABLE ACCESS	YALUS	FULL	4607	16
Filter Predicates				
ESTADO='C'				
FILTER				
Filter Predicates				
T2.ANO IS NULL				
HASH JOIN		OUTER	94	4
Access Predicates				
T1.ANO=T2.ANO(+)				
Filter Predicates				
T1.MEDIA<T2.MEDIA(+)				
VIEW			94	2
TABLE ACCESS	SYS.SYS_TEMP_0FD9D7757_446F158	FULL	94	2
VIEW			94	2
TABLE ACCESS	SYS.SYS_TEMP_0FD9D7757_446F158	FULL	94	2

Figura 21: Plano de execução da query para a pergunta 4, na cópia Y, na variante com left join.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			94	9
TEMP TABLE TRANSFORMATION				
LOAD AS SELECT	SYS_TEMP_0FD9D7758_446F158	(CURSOR DURATION MEMORY)		
SORT		GROUP BY NOSORT	94	5
BITMAP CONVERSION		TO ROWIDS	4440	5
BITMAP INDEX	Q4	RANGE SCAN		
Access Predicates				
ESTADO='C'				
Filter Predicates				
ESTADO='C'				
FILTER				
Filter Predicates				
T2.ANO IS NULL				
HASH JOIN		OUTER	94	4
Access Predicates				
T1.ANO=T2.ANO(+)				
Filter Predicates				
T1.MEDIA<T2.MEDIA(+)				
VIEW			94	2
TABLE ACCESS	SYS.SYS_TEMP_0FD9D7758_446F158	FULL	94	2
VIEW			94	2
TABLE ACCESS	SYS.SYS_TEMP_0FD9D7758_446F158	FULL	94	2

Figura 22: Plano de execução da query para a pergunta 4, na cópia Z, na variante com left join (utilizando CTE).

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			92	10
FILTER				
Filter Predicates				
T2.ANO IS NULL				
HASH JOIN		OUTER	92	10
Access Predicates				
T1.ANO=T2.ANO(+)				
Filter Predicates				
T1.MEDIA<T2.MEDIA(+)				
VIEW			94	5
HASH		GROUP BY	94	5
BITMAP CONVERSION		TO ROWIDS	4440	5
BITMAP INDEX	Q4	RANGE SCAN		
Access Predicates				
ESTADO='C'				
Filter Predicates				
ESTADO='C'				
VIEW			94	5
HASH		GROUP BY	94	5
BITMAP CONVERSION		TO ROWIDS	4440	5
BITMAP INDEX	Q4	RANGE SCAN		
Access Predicates				
ESTADO='C'				
Filter Predicates				
ESTADO='C'				

Figura 23: Plano de execução da query para a pergunta 4, na cópia Z, na variante com left join (sem utilizar CTE).

#### 4.4.4 Resultado da *query*

	CURSO	ANO	MEDIA
1	433	1999	14.16
2	233	1990	12
3	304	1988	11
4	331	1993	13.43
5	275	1998	14.35
6	433	1996	13.91
7	433	2001	13.5
8	433	1995	14.23
9	255	1992	13.11
10	433	1994	13.91
11	433	1997	13.4
12	275	2000	14.02

Figura 24: Resultado da *query* de resposta da pergunta 4.

#### 4.4.5 Comentários

Para a pergunta 4, a operação relevante era, depois de agrupar um conjunto de valores por curso e ano, filtrar o valor máximo. Para tal, duas estratégias foram estudadas: uma **subquery** que filtra o(s) tuplo(s) cujo valor é igual ao máximo, e uma auto junção via **left join** com a condição de junção a ser uma média inferior à outra (e, por conseguinte, os tuplos cujas colunas da segunda tabela tiverem valores nulos corresponderão a valores máximos).

Relativamente à estratégia via **subquery**, não se verificam alterações entre as cópias X e Y: em ambos, o otimizador determina que o melhor plano é fazer FULL TABLE SCANS, seguidos de HASH JOINS, devido a não conseguir aproveitar nenhuma restrição de integridade standard (no caso de Y). No entanto, a situação é bastante mais curiosa em Z: o índice, como esperado, permitiu não ter de aceder à tabela diretamente, sendo necessário apenas perfazer um range scan da sua tabela, reduzindo o custo em mais de metade. Porém, ao reestruturar a *query*, eliminando a Common Table Expression (CTE) e colocando o respetivo excerto duas vezes repetido, o custo estimado de execução é ainda menor, já que o otimizador estima que pode realizar a mesma operação de range scan duas vezes em vez de uma com consequente FULL ACCESS à vista temporária. Considerando que CTE deveria, em teoria, funcionar como *syntactic sugar* e que a *query* presente na mesma, neste caso, não é de uma complexidade elevada, este resultado surgiu com alguma surpresa.

Já quanto à estratégia **left join**, verificou-se não ser extremamente distante da outra estratégia em termos de performance: nas cópias X e Y, a estratégia é em si bastante semelhante às versões anteriores, com um custo ligeiramente maior, provavelmente visto que se trata de um outer join. Na cópia Z, o índice ajuda a um decréscimo considerável no custo pelas mesmas razões que anteriormente. Curiosamente, desta vez, ao desintegrar a estrutura CTE em detrimento de duas *queries* repetidas, o custo já incrementa (embora muito ligeiramente), apesar de, novamente, seguir um caminho algo análogo ao da estratégia anterior (realizar o RANGE SCAN duas vezes e realizar, aqui, um HASH JOIN).

Finalmente, os tempos de execução, são relativamente parecidos entre as duas versões, com algumas melhorias na versão Z, o que corrobora a hipótese de que as estratégias são igualmente

eficientes na resolução deste tipo de interrogações.

## 4.5 Pergunta 5

Compare os planos de execução da pesquisa “Quantos candidatos tiveram como resultado algo diferente de “C” ou “E”, usando, no contexto Z, a) Com índice árvore-B em Resultado; b) Com índice bitmap em Resultado.

### 4.5.1 Query SQL de resposta

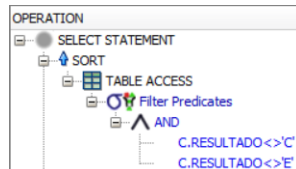
```
SELECT COUNT(*) AS "Número de Candidatos != (C | E)"
FROM cands c
WHERE c.resultado <> 'C' AND c.resultado <> 'E';
```

### 4.5.2 Tempos de execução

	X	Y	Z (btree)	Z (bitmap)
	0.03	0.049	0.025	0.02
	0.017	0.015	0.015	0.012
	0.021	0.018	0.017	0.014
	0.017	0.016	0.016	0.014
	0.016	0.015	0.016	0.013
$\mu$	0.020	0.023	0.018	0.015

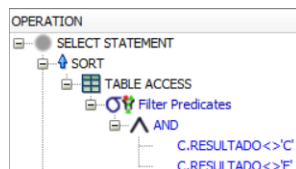
Tabela 7: Tempos (em segundos) da resposta da pergunta 5 nos diferentes ambientes.

### 4.5.3 Planos de execução



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	17
SORT		AGGREGATE	1	
TABLE ACCESS	XCANDS	FULL	1122	17
Filter Predicates				
AND				
C.RESULTADO <> 'C'				
C.RESULTADO <> 'E'				

Figura 25: Plano de execução da query para a pergunta 5, na cópia X.



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	17
SORT		AGGREGATE	1	
TABLE ACCESS	YCANDS	FULL	6474	17
Filter Predicates				
AND				
C.RESULTADO <> 'C'				
C.RESULTADO <> 'E'				

Figura 26: Plano de execução da query para a pergunta 5, na cópia Y.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	9
SORT		AGGREGATE	1	
INDEX	Q5_BTREE_CANDS	FAST FULL SCAN	6474	9
Filter Predicates				
AND				
C.RESULTADO<>'C'				
C.RESULTADO<>'E'				

Figura 27: Plano de execução da query para a pergunta 5, na cópia Z, usando um índice árvore-B.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	1
SORT		AGGREGATE	1	
BITMAP CONVERSION		COUNT	6474	1
BITMAP INDEX	G5_BITMAP_CANDS	FAST FULL SCAN		
Filter Predicates				
AND				
C.RESULTADO<>'C'				
C.RESULTADO<>'E'				

Figura 28: Plano de execução da query para a pergunta 5, na cópia Z, usando um índice bitmap.

#### 4.5.4 Resultado da query

Número de Candidatos != (C   E)	
1	618

Figura 29: Resultado da query de resposta da pergunta 5 (no total, o resultado contém 1).

#### 4.5.5 Comentários

Query simples, onde o plano de execução nas cópias X e Y é feito, sem nenhuma otimização, através de um FULL SCAN à tabela em análise, *cands*. Não se verificou um custo menor na cópia X do que na cópia Y, o que se deve ao facto de esta query não usar nenhuma das chaves primárias definidas.

No plano de execução na cópia Z, verificou-se uma redução substancial do custo em relação às outras cópias. No caso do índice B-tree, este custo passou a ser de 9, enquanto que no caso do índice Bitmap, este custo passou a ser de 1. Concluimos que o facto de o índice Bitmap apresentar melhores resultados do que o índice B-tree se deve ao facto de a coluna Resultado ter uma variabilidade muito baixa, com só 3 valores distintos.

Quanto aos tempos de execução, apesar de a sua média ser inferior na cópia Z, esta diferença não é significativa. Tal deve-se, provavelmente ao facto de a quantidade de dados não ser grande o suficiente para se registarem diferenças nos tempos de execução.

### 4.6 Pergunta 6

A pergunta “Há, em algum ano algum curso (ano\_lectivo, sigla e nome) que tenha todas as candidaturas aceites transformadas em matrículas, nesse mesmo ano?” é de natureza universal. Compare do ponto de vista temporal e de plano de execução as estratégias da dupla negação e da contagem.

#### 4.6.1 Query SQL de resposta

```
-- primeira forma (dupla negacao)
WITH aceites_ao_matriculados AS(
SELECT c.curso, c.ano_lectivo AS ano
FROM cands c
WHERE c.resultado='C' AND NOT EXISTS (SELECT 1 FROM alus a WHERE a.a_lect_matricula =
↪ c.ano_lectivo AND a.curso = c.curso AND c.bi = a.bi)
GROUP BY c.curso, c.ano_lectivo)
SELECT l.sigla, l.nome, c.ano_lectivo as ano
FROM lics l INNER JOIN cands c
ON l.codigo = c.curso
WHERE c.resultado = 'C' AND NOT EXISTS (SELECT 1 FROM aceites_ao_matriculados anm
↪ WHERE anm.curso = c.curso AND anm.ano = c.ano_lectivo)
GROUP BY l.sigla, l.nome, c.ano_lectivo;

-- segunda forma (contagem)
SELECT l.sigla, l.nome, c.curso, c.ano_lectivo as ano
FROM cands c INNER JOIN lics l
ON l.codigo = c.curso
WHERE c.resultado='C'
GROUP BY l.sigla, l.nome, c.curso, c.ano_lectivo
HAVING COUNT(*) = (SELECT COUNT(*) FROM alus a WHERE a.a_lect_matricula =
↪ c.ano_lectivo AND a.curso = c.curso);
```

#### 4.6.2 Tempos de execução

	X	Y	Z
	0.045	0.038	0.038
	0.036	0.033	0.039
	0.029	0.039	0.041
	0.033	0.033	0.035
	0.036	0.035	0.034
$\mu$	0.036	0.035	0.037

Tabela 8: Tempos (em segundos) da resposta da pergunta 6 nos diferentes ambientes (estratégia dupla negação).

	X	Y	Z
	0.133	0.137	0.025
	0.134	0.136	0.036
	0.129	0.125	0.03
	0.134	0.132	0.024
	0.13	0.125	0.025
$\mu$	0.132	0.131	0.028

Tabela 9: Tempos (em segundos) da resposta da pergunta 6 nos diferentes ambientes (estratégia contagem).

### 4.6.3 Planos de execução

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			61	56
HASH		GROUP BY	61	56
HASH JOIN			13172	54
Access Predicates				
L.CODIGO=C.CURSO				
TABLE ACCESS	XLICS	FULL	11	3
HASH JOIN		RIGHT ANTI	13172	51
Access Predicates				
AND				
ANM.CURSO=C.CURSO				
ANM.ANO=C.ANO_LECTIVO				
VIEW			1	34
HASH		GROUP BY	1	34
HASH JOIN		RIGHT ANTI	134	33
Access Predicates				
AND				
A.A_LECT_MATRICULA=C.ANO_LECTIVO				
A.CURSO=C.CURSO				
C.BI=A.BI				
TABLE ACCESS	XALUS	FULL	9214	16
TABLE ACCESS	XCANDS	FULL	13400	17
Filter Predicates				
C.RESULTADO='C'				
TABLE ACCESS	XCANDS	FULL	13400	17
Filter Predicates				
C.RESULTADO='C'				

Figura 30: Plano de execução da query de dupla negação para a pergunta 6, na cópia X.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	55
HASH		GROUP BY	1	55
HASH JOIN			1	54
Access Predicates				
L.CODIGO=ITEM_1				
NESTED LOOPS			1	54
NESTED LOOPS			1	54
STATISTICS COLLECTOR				
VIEW	SYS.VW_GBF_7		1	53
HASH		GROUP BY	1	53
HASH JOIN		RIGHT ANTI	13172	51
Access Predicates				
AND				
ANM.ANO=C.ANO_LECTIVO				
ANM.CURSO=C.CURSO				
VIEW			1	34
HASH		GROUP BY	1	34
HASH JOIN		RIGHT ANTI	134	33
Access Predicates				
AND				
A.A_LECT_MATRICULA=C.ANO_LECTIVO				
A.CURSO=C.CURSO				
C.BI=A.BI				
TABYALUS		FULL	9214	16
TABYCANDS		FULL	13400	17
Filter Predicates				
C.RESULTADO='C'				
TABLE ACCESS	YCANDS	FULL	13400	17
Filter Predicates				
C.RESULTADO='C'				
INDEX	YLICS_PK	UNIQUE SCAN	1	0
Access Predicates				
L.CODIGO=ITEM_1				
TABLE ACCESS	YLICS	BY INDEX ROWID	1	1
TABLE ACCESS	YLICS	FULL	1	1

Figura 31: Plano de execução da query de dupla negação para a pergunta 6, na cópia Y.



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	24
HASH		GROUP BY	1	24
HASH JOIN			1	23
Access Predicates L.CODIGO=ITEM_1				
NESTED LOOPS			1	23
NESTED LOOPS			1	23
STATISTICS COLLECTOR				
VIEW	SYS.VW_GBF_7		1	22
HASH		GROUP BY	1	22
HASH JOIN		RIGHT ANTI	13362	21
Access Predicates AND ANM.ANO=C.ANO_LECTIVO ANM.CURSO=C.CURSO				
VIEW			1	19
HASH		GROUP BY	1	19
NESTED LOOPS		ANTI	4186	18
TABLE ACCESS	ZCANDS	FULL	13400	17
Filter Predicates C.RESULTADO='C'				
INDEX	Q2	UNIQUE SCAN	6336	0
Access Predicates AND A.CURSO=C.CURSO A.A_LECT_MATRICULA=C.ANO_LECTIVO C.BI=A.BI				
BITMAP CONVERSION		TO ROWIDS	13400	2
BITMAP INDEX	Q6	FAST FULL SCAN		
Filter Predicates C.RESULTADO='C'				
INDEX	ZLICS_PK	UNIQUE SCAN	1	0
Access Predicates L.CODIGO=ITEM_1				
TABLE ACCESS	ZLICS	BY INDEX ROWID	1	1
TABLE ACCESS	ZLICS	FULL	1	1

Figura 32: Plano de execução da query de dupla negação para a pergunta 6, na cópia Z.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			134	21
FILTER				
Filter Predicates COUNT(*)= (SELECT COUNT(*) FROM XALUS A WHERE A.A_LECT_MATRICULA=:B1 AND A.CURSO=:B2)				
HASH		GROUP BY	134	21
HASH JOIN			13400	20
Access Predicates L.CODIGO=C.CURSO				
TABLE ACCESS	XLICS	FULL	11	3
TABLE ACCESS	XCANDS	FULL	13400	17
Filter Predicates C.RESULTADO='C'				
SORT		AGGREGATE	1	
TABLE ACCESS	XALUS	FULL	27	16
Filter Predicates AND A.A_LECT_MATRICULA=:B1 A.CURSO=:B2				

Figura 33: Plano de execução da query de contagem para a pergunta 6, na cópia X.

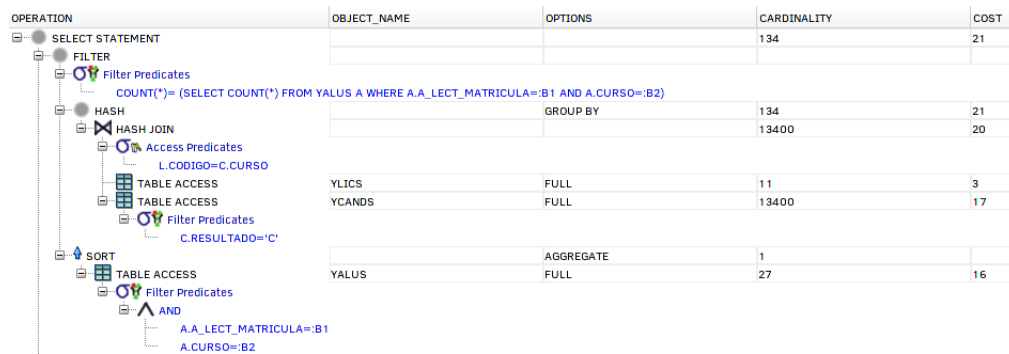


Figura 34: Plano de execução da query de contagem para a pergunta 6, na cópia Y.



Figura 35: Plano de execução da query de contagem para a pergunta 6, na cópia Z.

#### 4.6.4 Resultado da *query*

	SIGLA	NOME	ANO
1	EMM	Engenharia Metalúrgica e de Materiais	1982
2	EC	Engenharia Civil	1970
3	EMT	Engenharia Metalúrgica	1981
4	EMG	Engenharia de Minas e Geoambiente	1996
5	EM	Engenharia Mecânica	1977
6	EM	Engenharia Mecânica	1974
7	EMI	Engenharia de Minas	2001
8	EMM	Engenharia Metalúrgica e de Materiais	1983
9	EM	Engenharia Mecânica	1980
10	EMM	Engenharia Metalúrgica e de Materiais	1984
11	EC	Engenharia Civil	1972

Figura 36: Resultado da *query* de resposta da pergunta 6.

#### 4.6.5 Comentários

As duas estratégias desta última pergunta constituem duas formas bastante distintas de obter o mesmo resultado.

Como seria de esperar, a estratégia da **dupla negação** está sujeita a um plano de execução bastante mais complexo e dispendioso, visto que a própria formulação da *query* SQL assim o exige. Dada a necessidade de realizar FULL SCANS a todas as tabelas, são utilizados sucessivos HASH JOINS na junção das mesmas, com estratégia right anti dado o uso das cláusulas *not exists*. Na cópia Y, o otimizador tira partido da chave primária em *lics* para realizar as junções com essa tabela (que, com um índice unique, agora passam a ser mais vantajosas utilizando-a como tabela interna num NESTED LOOP, a nova escolha). No entanto, no geral, a melhoria de custo é quase insignificativa (apenas de 1 valor), talvez porque a tabela *lics* tem uma cardinalidade extremamente reduzida comparativamente às restantes tabelas e, por isso, o seu acesso não tem tanto impacto no custo total. A verdadeira melhoria no plano de execução verifica-se na cópia Z: para além de aproveitar a chave primária de *lics*, o otimizador reutiliza o índice unique definido para a chave estrangeira em *alus* (utilizado nas perguntas 2 e 3), permitindo transformar mais um HASH JOIN num NESTED LOOP (a junção entre *cands* e *alus* realizada no âmbito da primeira cláusula *not exists* presente na *query*). Para além disso, o índice Bitmap criado em *cands* permitiu ao sistema não ter de aceder à tabela diretamente, já que um (FAST) FULL SCAN ao índice é mais vantajoso e rápido. Em termos de custo, a melhoria é notável: a redução foi em mais de metade do valor "naïve".

Já a estratégia de **contagem** apresenta uma abordagem bem mais simples, comparando dois contadores apenas. Os planos de X e Y são equivalentes, visto que nenhuma chave primária é considerada útil pelo otimizador: é necessário percorrer duas tabelas via FULL SCAN, juntando-as com um HASH JOIN. Adicionalmente, outro FULL TABLE ACCESS com um "sort" aggregate (que na verdade não ordena nada) para calcular o valor do segundo contador para comparação. Na cópia Z, o uso dos dois índices já mencionados prova novamente trazer benefícios interessantes: os índices permitem uma consulta mais rápida e, estando as colunas necessárias todas presentes, evitam scans completos às tabelas originais. De notar que apesar de um HASH JOIN

ter sido convertido num HASH JOIN + NESTED LOOP, o custo total é bastante menor do que anteriormente.

Quanto a tempos de execução, os resultados podem ser considerados estranhos. Se, por um lado, na cópia Z, os resultados estão dentro do esperado (com uma vantagem palpável para a versão contagem), nas cópias X e Y, os tempos da versão contagem não fazem grande sentido. O plano da dupla negação é bastante mais dispensioso e, tirando um "sort" (que apenas agrega na realidade), inclui todo o tipo de operações presentes no plano de contagem. A única explicação que nos parece plausível é, de facto, um erro notável na estimativa do otimizador.

## 5 Conclusões

Com este trabalho é possível concluir que a adição de índices pode afetar significativamente a performance de *queries* a uma base de dados. Devendo ser sempre cuidadosamente analisados antes da sua criação, visto que trazem sempre um custo associado, representam, hoje em dia, um aumento importante no desempenho, já que, apesar de importantes na manutenção da consistência da base de dados, restrições de integridade *standard* (chaves primárias e estrangeiras) não satisfazem as necessidades contemporâneas. Apesar de representar frequentemente um processo incremental e até moroso, é fulcral para uma utilização eficiente por parte de terceiros posteriormente.