



Universidade do Porto
Faculdade de Engenharia
FEUP

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA E
COMPUTAÇÃO

TECNOLOGIAS DE BASES DE DADOS

Tema A

Otimização de consultas - Serviço docente

ANDRÉ GOMES FERREIRA ARAÚJO CORREIA - up200706629@fe.up.pt

ARTUR SOUSA FERREIRA - ei12168@fe.up.pt

JOSÉ FRANCISCO CAGIGAL DA SILVA GOMES - up201305016@fe.up.pt

9 de Abril de 2018

Conteúdo

1	Introdução	3
2	Explicação das restrições e índices criados	4
2.1	Restrições	4
2.1.1	Definição de restrições - Caso de estudo	4
2.2	Índices	5
2.2.1	Principais tipos de índices	6
2.2.1.1	<i>B-Tree</i>	6
2.2.1.2	<i>Hashing</i>	7
2.2.1.3	<i>Bitmap</i>	8
2.2.2	Processo de definição dos índices	8
2.2.3	Definição dos índices - Caso de estudo	9
3	Perguntas	11
3.1	Pergunta 1	11
3.1.1	SQL query	11
3.1.2	Resposta	11
3.1.3	Análise do plano de execução	11
3.1.4	Tempos	13
3.2	Pergunta 2	13
3.2.1	SQL query	13
3.2.2	Resposta	13
3.2.3	Análise do plano de execução	14
3.2.4	Tempos	15
3.3	Pergunta 3a	15
3.3.1	SQL query	15
3.3.2	Resposta	15
3.3.3	Análise do plano de execução	16
3.3.4	Tempos	17
3.4	Pergunta 3b	17
3.4.1	SQL query	17
3.4.2	Resposta	18
3.4.3	Análise do plano de execução	18
3.4.4	Tempos	19
3.5	Pergunta 4	20
3.5.1	SQL query	20
3.5.2	Resposta	20

3.5.3	Análise do plano de execução	20
3.5.4	Tempos	23
3.6	Pergunta 5	23
3.6.1	SQL query	23
3.6.2	Resposta	23
3.6.3	Análise do plano de execução	23
3.6.4	Tempos	26
3.7	Pergunta 6	26
3.7.1	SQL query	26
3.7.2	Resposta	26
3.7.3	Análise do plano de execução	26
3.7.4	Tempos	28
4	Conclusão	29

1 Introdução

Este trabalho foi desenvolvido no âmbito da unidade curricular Tecnologias de Bases de Dados do Mestrado Integrado em Engenharia Informática e Computação, e tem como principal objetivo perceber até que ponto a utilização de índices afeta o desempenho de consultas a bases de dados, através da análise dos diferentes planos de execução.

No decorrer deste relatório vão ser analisadas 7 perguntas, no que toca aos custos da consulta e ao tempo médio de resposta em 3 diferentes tipos de tabela. As tabelas X não têm qualquer tipo de restrições ou índices. As tabelas Y têm chaves primárias e estrangeiras. Finalmente, as tabelas Z são compostas por chaves primárias e estrangeiras e, ainda, índices escolhidos pelos autores.

2 Explicação das restrições e índices criados

2.1 Restrições

A definição de restrições de integridade condiciona os valores dos tuplos na base de dados. Um primeiro grupo de restrições está associado ao Modelo Relacional, impondo, por exemplo, que não haja tuplos duplicados. Um segundo grupo de restrições pode ser imposto no esquema da relação, e inclui restrições de domínio e chaves. Um terceiro grupo depende das regras do negócio e terá de ser imposto através das construções que a linguagem de definição de dados permitir.

As chaves constituem restrições de integridade associadas ao esquema e permitem ao SGBD manter a coerência dos dados, não permitindo tuplos que violem as restrições.

A restrição PRIMARY KEY é equivalente às cláusulas NOT NULL + UNIQUE juntas, isto é, o conteúdo da coluna ou colunas não pode ser nulo e não pode admitir repetições. Numa tabela apenas pode existir uma PRIMARY KEY.

A escolha da chave primária não é totalmente arbitrária, visto que devem seguir-se boas práticas, por exemplo: escolher uma chave pequena, com poucos atributos ou apenas um.

Uma chave estrangeira de uma relação assume valores da chave primária, geralmente de outra relação, e permite manter a integridade referencial.

A restrição REFERENCES permite fazer a validação das chaves estrangeiras. Isto é, não se podem introduzir nos campos referenciados como chave estrangeira, valores que não existam na tabela onde os campos são chave primária.

2.1.1 Definição de restrições - Caso de estudo

Relativamente ao caso em análise, foram definidas as seguintes chaves primárias e estrangeiras para as tabelas com prefixo Y e Z:

Tabela DOCENTES		
Restrição	Nome	Atributo
PK	PK_DOCENTES	NR

Tabela 1: Restrições na tabela DOCENTES

Tabela DSD			
Restrição	Nome	Atributos	Referência
PK	PK_DSD	NR,ID	-
FK	FK_DOCENTE_NR	NR	DOCENTES(NR)
FK	FK_TIPOSAULA_ID	ID	TIPOSAULA(ID)

Tabela 2: Restrições na tabela DSD

Tabela TIPOSAULA			
Restrição	Nome	Atributos	Referência
PK	PK_TIPOSAULA	ID	-
FK	FK_OCORRENCIAS	ANO_LET,PER,COD	OCORRENCIAS(ANO_LET,PER,COD)

Tabela 3: Restrições na tabela TIPOSAULA

É de salientar que para a criação da FK_OCORRENCIAS foi necessário recorrer ao ENABLE NOVALIDATE, uma vez que existem tuplos na tabela TIPOSAULA que não têm correspondência na tabela OCORRENCIAS. Conforme mencionado anteriormente, as chaves estrangeiras servem exatamente para evitar que situações deste tipo possam ocorrer, pelo que se pressupõe que as chaves estrangeiras não existiram desde o início da criação das duas tabelas.

Tabela OCORRENCIAS			
Restrição	Nome	Atributos	Referência
PK	PK_OCORRENCIAS	ANO_LETIVO,PERIODO,CODIGO	-
FK	FK_UCS_CODIGO	CODIGO	UCS(CODIGO)

Tabela 4: Restrições na tabela OCORRENCIAS

Tabela UCS		
Restrição	Nome	Atributo
PK	PK_UCS	CODIGO

Tabela 5: Restrições na tabela UCS

2.2 Índices

Em muitas situações, as consultas à base de dados dizem respeito apenas a registos com valores específicos em alguns dos seus atributos, pelo que não se justifica o seu varrimento sequencial. Neste caso, é útil usar estruturas auxiliares de acesso, chamadas índices. Um índice funciona como num livro: sendo dado um valor, permite um acesso quase direto ao item pretendido. Deste modo, os índices têm entradas que permitem o acesso aleatório através de uma chave, que pode ser qualquer conjunto de atributos de uma relação.

Em alguns casos, a utilização do índice pode dispensar o acesso aos registos, por exemplo, quando se pretende saber apenas se um dado valor de atributo existe. Esta é uma situação comum quando se verifica a integridade referencial e se pretende saber apenas se um determinado valor da chave primária existe.

A utilização de um índice como uma estrutura auxiliar de melhoria do desempenho pressupõe:

- Que a existência do índice não piora as escolhas feitas pelo componente de otimização de consultas (a existência de muitos índices, fornecendo várias alternativas de caminhos

de acesso, pode piorar o processo de otimização de consultas, quer em tempo, quer em qualidade);

- Que o custo de aceder aos registos usando o índice não piora o desempenho, quando comparado com um acesso sem índice;
- Que o custo de efetuar alterações nas colunas indexadas não é agravado de forma substancial pelo custo das alterações que implicam nos respetivos índices.

2.2.1 Principais tipos de índices

Existem essencialmente três tipos de índices:

- Arborescentes ou ordenados;
- *Hashing*;
- *Bitmap*.

Os índices arborescentes utilizam vários tipos de estruturas arborescentes, essencialmente variações de árvores binárias, baseadas numa dada ordenação dos valores da chave do índice. Os índices de *hashing* repartem uniformemente os valores da chave por um conjunto de blocos aplicando uma função de *hashing*.

Em seguida, é feita uma descrição sumária de cada um.

2.2.1.1 *B-Tree*

Uma *B-Tree* é uma estrutura arborescente que apresenta as seguintes características genéricas:

- Os nós podem conter 2 k -chaves, ao invés das árvores binárias, onde um nó contém uma única chave. Diz-se que a ordem da árvore é k ;
- Os nós nunca são totalmente preenchidos, usando geralmente apenas metade da sua capacidade para ficar com espaço livre para inserções posteriores. Desta forma, a inserção posterior de muitas chaves não implica forçosamente reorganizar os nós;
- Podem ser acrescentados e removidos nós conforme necessário, com um custo reduzido;
- A ordenação da chave é sempre mantida, o que facilita as operações de encontrar antecessores e sucessores de um dado valor. Desta forma, um índice pode ser usado para ordenar resultados;
- Qualquer folha está à mesma distância da raiz, o que significa que a árvore é equilibrada.

Esta última propriedade é particularmente importante, garantindo que o custo de acesso a qualquer folha é uniforme. O custo máximo de encontrar uma chave, em termos de acesso ao disco, é sempre igual à altura da árvore.

As operações de pesquisa, inserção e remoção têm um custo $O(\log(N))$, em que N é o número de chaves.

Ao longo dos anos foram sendo desenvolvidas variantes da *B-Tree* original, tendo em vista colmatar algumas das suas limitações - como a ineficiência em acessos sequenciais. A *B+-Tree* é, assim, uma variante da *B-Tree* original e veio introduzir as seguintes propriedades:

- Os nós intermédios não permitem acesso aos registos, apenas contêm chaves. Somente as folhas permitem o acesso aos registos;
- Por essa razão, as chaves dos nós intermédios, que servem apenas de separadores, repetem-se nas folhas, correspondendo assim as folhas a todos os valores existentes nos registos;
- As folhas estão ligadas entre si, nos dois sentidos, o que favorece o varrimento sequencial do índice.

As folhas contêm apontadores para as páginas onde se situam os registos.

As *B+-Tree* podem organizar um elevado número de chaves num número de níveis reduzido. Como o número de níveis da árvore corresponde ao número máximo de acessos ao disco para encontrar uma chave, esta propriedade é interessante.

Se considerarmos que uma entrada do índice tem 20 *bytes* (12 para a chave e 8 para o apontador) e que as páginas são de 8 K, temos cerca de 400 entradas por página. Se cada página estiver semipreenchida e contiver em média cerca de 250 entradas, temos a seguintes estimativa do número total de chaves e do espaço necessário, conforme o número de níveis (Tabela 1):

Nível	N.º de chaves	Tamanho (<i>bytes</i>)
1	250	8 K
2	$250^2 = 62\ 500$	2 MB
3	$250^3 = 15\ 625\ 000$	500 MB

Tabela 6: Número de chaves e tamanho da árvore

O nível 1 só contém a raiz, ou seja, uma página de 8 K com 250 chaves. Se o nível 2 estiver preenchido, temos 250 nós descendentes da raiz, com uma média de 250 chaves em cada um, ocupando 2 MB no total. Ou seja, uma árvore de nível 3 pode conter mais de 15 milhões de chaves e ocupar 500 MB. Para a grande maioria das aplicações, não são necessários mais do que estes três níveis. Estando a raiz em memória - o que parece razoável, dado que é apenas uma página - isto significa que qualquer chave pode ser encontrada em apenas dois acessos ao disco.

2.2.1.2 *Hashing*

Uma função de *hashing* opera um mapeamento entre um grande conjunto de entrada e um conjunto finito de saída. As funções de *hashing* são extensivamente usadas em várias áreas da informática. No contexto da indexação, a função de *hashing* mapeia uma chave K para um de n blocos, que são tipicamente as páginas do disco. Assim, várias chaves são mapeadas para o mesmo bloco, originando uma colisão, e uma das propriedades mais desejáveis de uma função de *hashing*

é o mapeamento uniforme dos valores da chave por todos os n blocos. Se um bloco for uma página do disco que contém os registos indexados, na prática, este tipo de funções permite um acesso direto aos dados a partir da chave.

Assim, uma função de *hashing* deve distribuir uniformemente os registos por todos os blocos, e essa distribuição deve ser aleatória, isto é, deve ser independente da distribuição das chaves, resultando no mesmo número de chaves por bloco.

Os índices que usam funções de *hashing* apresentam as seguintes limitações:

- Como não existe nenhuma relação de ordem entre $h(k)$, não é possível percorrer de forma eficiente e ordenada as chaves. Deste modo, os índices que usam *hashing* não podem ser agrupados, exceto se a organização da tabela estiver também de acordo com a função de *hashing*;
- Não é possível utilizar funções de *hashing* para consultas com intervalos de valor, apenas para condições de igualdade.

No entanto, as funções de *hashing* são muito eficazes para testes de igualdade, pelo que são bastante usadas no processamento de equijunções.

2.2.1.3 Bitmap

Um índice mapa de *bits* usa um mapa de *bits*, em que cada *bit* corresponde a um valor possível do atributo indexado. Para um determinado tuplo e atributo indexado, o *bit* k é 1 se o k -ésimo valor estiver no atributo desse tuplo, caso contrário é 0.

O mapa de *bits* permite uma representação compacta dos valores presentes no atributo, e é tanto mais eficaz quanto menos valores únicos existirem para esse atributo. Este tipo de índice permite também a representação de valores nulos, o que não acontece com os índices arborescentes (*B-Tree*).

Estes índices implicam geralmente um acesso aos tuplos no disco, enquanto uma *B-Tree* pode permitir acessos apenas ao índice.

Este tipo de índices é essencialmente atrativo para consultas muito variadas e grandes quantidades de tuplos - situações em que a utilização de diversos índices *B-Tree* muito grandes seria ineficaz.

2.2.2 Processo de definição dos índices

A correta utilização de índices é essencial para um bom desempenho da base de dados e a sua definição deve ser considerada uma etapa essencial da conceção física. Assim, a definição dos índices deve ser feita durante a conceção e não após a entrada em produção do sistema, onde apenas pequenos ajustes deverão ser feitos.

Assim, a definição dos índices deve ser baseada em dois processos elementares:

- Detetar consultas que sejam muito lentas devido à falta de indexação apropriada;

- Definir índices que tornem todas as consultas suficientemente rápidas.

O desempenho da utilização de índices pode variar, dependendo de como o otimizador de consultas os utiliza. Existem essencialmente três operações efetuadas através de índices:

- **Consulta apenas do índice:** o acesso é muito rápido, dispensando o acesso às páginas da tabela. Acessos nesta categoria incluem consultas por igualdade de chave primária ou de outra coluna única;
- **Consulta por intervalo no índice:** o acesso percorre todas as folhas do índice que respeitam o critério de pesquisa no intervalo de valores. O resultado é um conjunto de endereços de registos;
- **Consulta da tabela usando entrada do índice:** a página que contém o endereço encontrado no índice é lida. Esta situação é comum nos casos em que se acede a colunas que não estão no índice. Uma situação limite desta operação consulta todos os endereços da tabela, pelo que o varrimento sequencial pode ser a opção mais económica.

A terceira operação determina fortemente o desempenho. Se muitas páginas tiverem de ser lidas, os tempos de resposta aumentam, principalmente se estas não forem sequenciais. Isto pode acontecer se esta operação for precedida da segunda operação e esta resultar num conjunto muito grande de registos.

2.2.3 Definição dos índices - Caso de estudo

Relativamente ao caso em análise, foram definidas os seguintes índices para as tabelas com o prefixo Z:

Tabela DOCENTES		
Tipo	Nome	Atributos
B-Tree (UNIQUE)	IDX_ZDOCENTES_NR_NOME	NR,NOME

Tabela 7: Índices na tabela DOCENTES

O índice `IDX_ZDOCENTES_NR_NOME` na tabela `ZDOCENTES`, para os atributos `nr` e `nome`, revela-se de enorme importância, dado que, a maioria dos acessos a esta tabela é feito através do atributo `nr`, mas o atributo que se pretende ver projetado é o nome do docente. Assim, com a criação deste índice, evita-se, na maior parte das consultas, o acesso à tabela, sendo que apenas é realizado o acesso ao índice.

Tabela TIPOSAULA		
Tipo	Nome	Atributos
B-Tree (NONUNIQUE)	IDX_ZTIPOSAULA_ANOLETIVO	ANO_LETIVO
B-Tree (NONUNIQUE)	IDX_ZTIPOSAULA_ANOLETIVO_COD	ANO_LETIVO,CODIGO
Bitmap	IDX_ZTIPOSAULA_BITMAP_TIPO	TIPO

Tabela 8: Índices na tabela TIPOSAULA

Conforme se comprovará mais à frente, o índice `IDX_ZTIPOSAULA_ANOLETIVO` na tabela `ZTIPOSAULA`, para o atributo `ano_letivo`, reveste-se de enorme importância, uma vez que a grande maioria dos acessos a esta tabela são feitos através do ano letivo. Com efeito, a maioria das consultas são para um determinado ano letivo.

Também o índice `IDX_ZTIPOSAULA_ANOLETIVO_COD` na tabela `ZTIPOSAULA`, para os atributos `ano_letivo` e `codigo`, se reveste de enorme importância, dado que, e conforme mencionado anteriormente, um grande número de acessos a esta tabela é feito através do ano letivo, mas o atributo que se pretende ver projetado é o código.

Relativamente ao índice `IDX_ZTIPOSAULA_BITMAP_TIPO`, com o atributo `tipo` na tabela `ZTIPOSAULA`, este assume grande importância uma vez que algumas das consultas recorrem ao agrupamento pelo atributo `tipo`. Neste caso optou-se pela criação de um índice *bitmap* devido à baixa cardinalidade do atributo `tipo` - 5.

Tabela UCS		
Tipo	Nome	Atributos
B-Tree (UNIQUE)	IDX_ZUCS_CURSO_DESG_COD	CURSO,DESIGNACAO,CODIGO

Tabela 9: Índices na tabela UCS

A criação do índice `IDX_ZUCS_CURSO_DESG_COD` na tabela `UCS`, para os atributos `curso`, `designacao` e `codigo`, reveste-se, igualmente, de enorme importância uma vez que a maioria das consultas são para um determinado curso. No entanto, e apesar de o acesso maioritário ser realizado através do atributo `curso`, o atributo que é necessário ser projetado mais vezes é o código e, por vezes, também a designação. Assim, com a criação de um índice com estes três atributos evita-se, na maior parte das consultas, o acesso à tabela, sendo consultado apenas o índice.

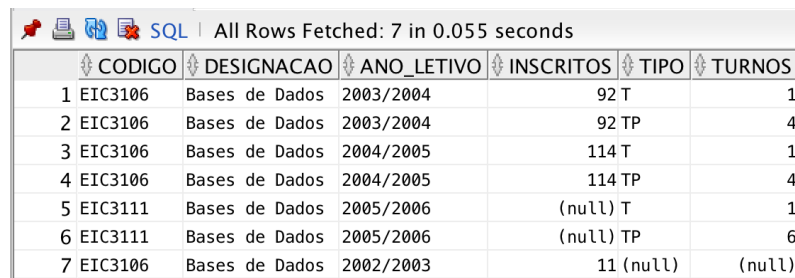
3 Perguntas

3.1 Pergunta 1

3.1.1 SQL query

```
SELECT xucs.codigo , xucs.designacao , xocorrencias.ano_letivo , xocorrencias.
    inscritos , xtiposaula.tipo , xtiposaula.turnos
FROM xucs JOIN xocorrencias ON xucs.codigo = xocorrencias.codigo
left outer JOIN xtiposaula ON xtiposaula.codigo = xocorrencias.codigo and
    xtiposaula.ano_letivo = xocorrencias.ano_letivo
WHERE xucs.designacao = 'Bases de Dados' AND xucs.curso = 275;
```

3.1.2 Resposta



	CODIGO	DESIGNACAO	ANO_LETIVO	INSCRITOS	TIPO	TURNOS
1	EIC3106	Bases de Dados	2003/2004	92	T	1
2	EIC3106	Bases de Dados	2003/2004	92	TP	4
3	EIC3106	Bases de Dados	2004/2005	114	T	1
4	EIC3106	Bases de Dados	2004/2005	114	TP	4
5	EIC3111	Bases de Dados	2005/2006	(null)	T	1
6	EIC3111	Bases de Dados	2005/2006	(null)	TP	6
7	EIC3106	Bases de Dados	2002/2003	11 (null)	(null)	(null)

Figura 1: Resultados da execução da consulta.

É de salientar que, após uma análise cuidada e atenta dos dados, se verificou que relativamente à unidade curricular EIC3106, apesar de existir registo dela na tabela OCORRENCIAS, para o ano letivo 2002/2003, verificou-se não existir correspondência na tabela TIPOSAULA para esse mesmo ano letivo. Dado existirem alunos inscritos (e aprovados) na UC nesse ano letivo, e apesar de não haver registo da existência de aulas, decidiu-se apresentar o respetivo tuplo no resultado com recurso ao OUTER JOIN. Para tal decisão, teve influência o facto de no enunciado do trabalho ser solicitada a apresentação do ano letivo e do número de inscritos.

3.1.3 Análise do plano de execução

- x)



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			4	642
HASH JOIN		OUTER	4	642
Access Predicates				
AND				
XTIPOSADULA.CODIGO(+) = XOCORRENCIAS.CODIGO				
XTIPOSADULA.ANO_LETIVO(+) = XOCORRENCIAS.ANO_LETIVO				
HASH JOIN			4	606
Access Predicates				
XUCS.CODIGO = XOCORRENCIAS.CODIGO				
TABLE ACCESS	XUCS	FULL	1	13
Filter Predicates				
AND				
XUCS.DESIGNACAO = 'Bases de Dados'				
XUCS.CURSO = 275				
TABLE ACCESS	XOCORRENCIAS	FULL	21747	593
TABLE ACCESS	XTIPOSADULA	FULL	21206	36

Figura 2: Plano de execução da *query* 1, nas tabelas com prefixo X.

Através da análise do plano de execução, ilustrado na Figura 2, é possível constatar que a primeira operação realizada é o varrimento completo da tabela XUCS, aplicando os filtros presentes na cláusula WHERE aos respetivos tuplos. Daqui resultam apenas os tuplos que cumprem as condições presentes na cláusula (designacao = 'Bases de Dados' e curso = 275). Em seguida, é feito um varrimento completo da tabela XOCORRENCIAS, selecionando apenas os tuplos cujo código seja igual ao código dos tuplos resultantes da seleção anterior (na tabela XUCS). Posteriormente, é efetuada a junção através de um HASH JOIN. Finalmente, é efetuado um varrimento completo da tabela XTIPOSAULA selecionando os tuplos cujo código e ano letivo sejam iguais aos tuplos da tabela OCORRENCIAS, resultantes da seleção anterior. Posto isto, é feita a junção através de um HASH JOIN.

• y)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			10	60
HASH JOIN		OUTER	10	60
Access Predicates				
AND				
YTIPOSAULA.CODIGO(+) = YOCORRENCIAS.CODIGO				
YTIPOSAULA.ANO_LETIVO(+) = YOCORRENCIAS.ANO_LETIVO				
HASH JOIN			10	23
Access Predicates				
YUCS.CODIGO = YOCORRENCIAS.CODIGO				
NESTED LOOPS			10	23
NESTED LOOPS			10	23
STATISTICS COLLECTOR				
TABLE ACCESS	YUCS	FULL	2	13
Filter Predicates				
AND				
YUCS.DESIGNACAO = 'Bases de Dados'				
YUCS.CURSO = 275				
INDEX	PK_OCORRENCIAS	RANGE SCAN	5	1
Access Predicates				
YUCS.CODIGO = YOCORRENCIAS.CODIGO				
TABLE ACCESS	YOCORRENCIAS	BY INDEX ROWID	5	6
TABLE ACCESS	YOCORRENCIAS	FULL	5	6
TABLE ACCESS	YTIPOSAULA	FULL	21206	36

Figura 3: Plano de execução da *query* 1, nas tabelas com prefixo Y.

Como é possível verificar através da análise do plano de execução ilustrado na Figura 3, a principal diferença, com a introdução das restrições de chaves primárias e estrangeiras nas diversas tabelas, reflete-se no facto de já não ser necessário o recurso ao varrimento completo da tabela YOCORRENCIAS. Assim, com a introdução das chaves primárias e estrangeiras, após a seleção dos tuplos da tabela YUCS que cumprem as condições da cláusula WHERE, os respetivos tuplos na tabela YOCORRENCIAS são acedidos através do índice criado com a chave primária.

• z)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	29
HASH JOIN		OUTER	1	29
Access Predicates				
AND				
ZTIPOSAULA.CODIGO(+) = ZOCORRENCIAS.CODIGO				
ZTIPOSAULA.ANO_LETIVO(+) = ZOCORRENCIAS.ANO_LETIVO				
NESTED LOOPS		OUTER	1	29
STATISTICS COLLECTOR				
HASH JOIN			1	7
Access Predicates				
ZUCS.CODIGO = ZOCORRENCIAS.CODIGO				
NESTED LOOPS			1	7
STATISTICS COLLECTOR				
INDEX	IDX_ZUCS_CURSO_DESG_COD	RANGE SCAN	1	1
Access Predicates				
AND				
ZUCS.CURSO = 275				
ZUCS.DESIGNACAO = 'Bases de Dados'				
TABLE ACCESS	ZOCORRENCIAS	BY INDEX ROWID BATCHED	5	6
INDEX	PK_ZOCORRENCIAS	RANGE SCAN	5	1
Access Predicates				
ZUCS.CODIGO = ZOCORRENCIAS.CODIGO				
TABLE ACCESS	ZOCORRENCIAS	FULL	5	6
TABLE ACCESS	ZTIPOSAULA	BY INDEX ROWID BATCHED	1	22
Access Predicates				
ZTIPOSAULA.CODIGO(+) = ZOCORRENCIAS.CODIGO				
INDEX	IDX_ZTIPOSAULA_ANOLETIVO	RANGE SCAN	1116	3
Access Predicates				
ZTIPOSAULA.ANO_LETIVO(+) = ZOCORRENCIAS.ANO_LETIVO				
TABLE ACCESS	ZTIPOSAULA	FULL	1	22

Figura 4: Plano de execução da *query* 1, nas tabelas com prefixo Z.

Conforme se constata através da análise do plano de execução ilustrado na Figura 4, a criação de um índice *B-Tree* nos dois atributos presentes na cláusula *WHERE* (designacao e curso) e no atributo código - *IDX_ZUCS_CURSO_DESG_COD* - evita o acesso à tabela *ZUCS*, sendo o acesso realizado apenas através do índice. Como o índice contém todos os atributos da tabela *ZUCS* que são necessários à projeção (código, designação e curso), não é necessário o acesso à tabela *ZUCS*, sendo realizado apenas o acesso ao índice. Com a criação do índice *IDX_ZTIPOSAULA_ANOLETIVO* na tabela *ZTIPOSAULA*, com o atributo *ano_letivo*, evita-se o varrimento completo da tabela *ZTIPOSAULA*, sendo que agora o acesso aos tuplos é feito através do índice. Os tuplos são acedidos através do índice, pesquisando apenas pelo ano letivo igual aos tuplos resultantes da seleção anterior na tabela *ZOCORRENCIAS*.

Assim, e através da análise das Figuras acima, é possível constatar que existe uma enorme diferença entre os custos das consultas às tabelas *X* e *Y* (diferença de 642 para 60). Conforme referido anteriormente, esta diferença deve-se ao facto de nas tabelas *Y* estarem presentes restrições tipo chaves primárias que automaticamente criam índices únicos.

Comparando as tabelas *Y* às *Z*, verifica-se a existência de uma diferença de custos significativa (60 para 29), embora menor do que a verificada anteriormente.

3.1.4 Tempos

Tabela	Tempo (s)
X	0.055
Y	0.046
Z	0.037

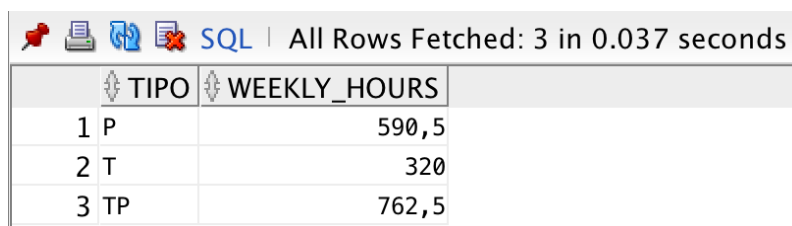
Tabela 10: Tempo de execução da *query* 1, nas tabelas com prefixo *X*, *Y* e *Z*.

3.2 Pergunta 2

3.2.1 SQL query

```
SELECT tipo , SUM(turnos*horas_turno) AS weekly_hours
FROM xucs JOIN xtiposaula ON xtiposaula.codigo = xucs.codigo
WHERE xucs.curso = 233 AND xtiposaula.ano_letivo = '2004/2005'
GROUP BY tipo;
```

3.2.2 Resposta



TIPO	WEEKLY_HOURS
1 P	590,5
2 T	320
3 TP	762,5

Figura 5: Resultados da execução da consulta.

3.2.3 Análise do plano de execução

- x)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				50
HASH		GROUP BY	5	50
HASH JOIN			236	49
Access Predicates				
XTIPOSAULA.CODIGO=XUCS.CODIGO				
TABLE ACCESS	XUCS	FULL	504	13
Filter Predicates				
XUCS.CURSO=233				
TABLE ACCESS	XTIPOSAULA	FULL	1690	36
Filter Predicates				
XTIPOSAULA.ANO_LETIVO='2004/2005'				

Figura 6: Plano de execução da *query* 2, nas tabelas com prefixo X.

Através da análise do plano de execução presente na Figura 6, é possível constatar que é feito o varrimento completo, tanto da tabela XUCS como da tabela XTIPOSAULA, aplicando em cada uma os respectivos filtros (curso e ano letivo, respetivamente). Em seguida, é realizada a junção (HASH JOIN) através do atributo *codigo*.

- y)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				50
HASH		GROUP BY	5	50
HASH JOIN			54	49
Access Predicates				
YTIPOSAULA.CODIGO=YUCS.CODIGO				
TABLE ACCESS	YUCS	FULL	47	13
Filter Predicates				
YUCS.CURSO=233				
TABLE ACCESS	YTIPOSAULA	FULL	1116	36
Filter Predicates				
YTIPOSAULA.ANO_LETIVO='2004/2005'				

Figura 7: Plano de execução da *query* 2, nas tabelas com prefixo Y.

Da análise do plano de execução presente na Figura 7, é possível constatar que o mesmo permanece inalterado relativamente ao anterior, uma vez que os índices criados, com a introdução das chaves primárias, não contemplam nenhum dos atributos pelos quais são realizados os acessos às tabelas nesta consulta (YUCS.CURSO e YTIPOSAULA.ANO_LETIVO).

- z)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				40
HASH		GROUP BY	3	40
HASH JOIN			236	39
Access Predicates				
ZTIPOSAULA.CODIGO=ZUCS.CODIGO				
NESTED LOOPS			236	39
STATISTICS COLLECTOR				
INDEX	IDX_ZUCS_CURSO_DESG_COD	RANGE SCAN	503	5
Access Predicates				
ZUCS.CURSO=233				
INDEX	IDX_ZTIPOSAULA_ANOLETIVO	RANGE SCAN	1690	6
Access Predicates				
ZTIPOSAULA.ANO_LETIVO='2004/2005'				
TABLE ACCESS	ZTIPOSAULA	BY INDEX ROWID	1	34
Filter Predicates				
ZTIPOSAULA.CODIGO=ZUCS.CODIGO				
TABLE ACCESS	ZTIPOSAULA	BY INDEX ROWID BATCHED	1690	34
Filter Predicates				
ZTIPOSAULA.ANO_LETIVO='2004/2005'				

Figura 8: Plano de execução da *query* 2, nas tabelas com prefixo Z.

Conforme se constata pela análise da Figura 8, com a criação do índice *IDX_ZUCS_CURSO_DESG_COD* na tabela ZUCS e do índice *IDX_ZTIPOSAULA_ANOLETIVO* na tabela ZTIPOSAULA,

evita-se o varrimento completo de ambas as tabelas. É consultado o índice da tabela ZUCS para obter os codigos dos tuplos com curso = 233 e, posteriormente, esses codigos são utilizados para seleccionar os tuplos da tabela ZTIPOSAULA. A consulta à tabela ZTIPOSAULA é feita com recurso ao índice no atributo ano_letivo.

Assim, embora não se tenha verificado redução do custo, entre as consultas às tabelas com prefixo X e Y, já entre as consultas às tabelas com prefixo Y e Z a redução é de assinalar (50 para 40).

3.2.4 Tempos

Tabela	Tempo (s)
X	0.037
Y	0.033
Z	0.029

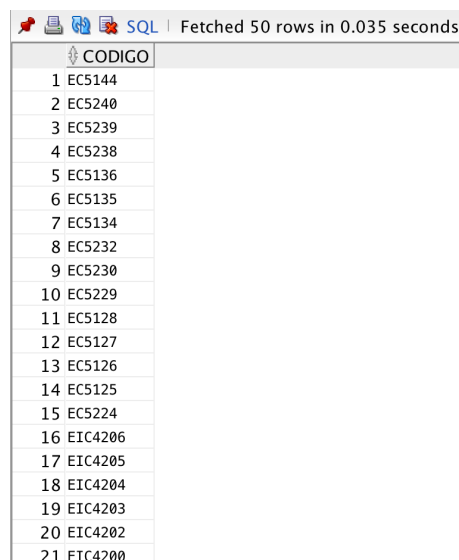
Tabela 11: Tempo de execução da *query* 2, nas tabelas com prefixo X, Y e Z.

3.3 Pergunta 3a

3.3.1 SQL query

```
SELECT codigo
FROM xucs
WHERE codigo NOT IN
(SELECT DISTINCT codigo
FROM xtiposaula
WHERE ano_letivo = '2003/2004');
```

3.3.2 Resposta



CODIGO
1 EC5144
2 EC5240
3 EC5239
4 EC5238
5 EC5136
6 EC5135
7 EC5134
8 EC5232
9 EC5230
10 EC5229
11 EC5128
12 EC5127
13 EC5126
14 EC5125
15 EC5224
16 EIC4206
17 EIC4205
18 EIC4204
19 EIC4203
20 EIC4202
21 EIC4200

Figura 9: Resultados da execução da consulta.

3.3.3 Análise do plano de execução

- x)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5396	49
HASH JOIN		RIGHT ANTI	5396	49
Access Predicates				
CODIGO=CODIGO				
TABLE ACCESS	XTIPOSAULA	FULL	1610	36
Filter Predicates				
ANO_LETIVO='2003/2004'				
TABLE ACCESS	XUICS	FULL	5396	13

Figura 10: Plano de execução da *query* 3a., nas tabelas com prefixo X.

Conforme se constata através da análise do plano de execução ilustrado na Figura 10, é feito um varrimento completo à tabela XTIPOSAULA, aplicando o filtro correspondente à condição presente na cláusula WHERE (`ano_letivo = '2003/2004'`). Verifica-se, igualmente através da análise da Figura 10, que é feito um HASH JOIN RIGHT ANTI. Quer isto dizer que, o resultado da seleção dos tuplos da tabela XTIPOSAULA é guardado em memória, sendo aplicada uma função de *hashing* e, posteriormente, é realizado um varrimento completo à tabela XUICS, seleccionando apenas aqueles tuplos cujo código não se encontra nessa tabela em memória.

- y)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5396	42
HASH JOIN		RIGHT ANTI	5396	42
Access Predicates				
CODIGO=CODIGO				
TABLE ACCESS	YTIPOSAULA	FULL	1116	36
Filter Predicates				
ANO_LETIVO='2003/2004'				
INDEX	PK_UCS	FAST FULL SCAN	5396	6

Figura 11: Plano de execução da *query* 3a., nas tabelas com prefixo Y.

Como se verifica da análise do plano de execução presente na Figura 11, a única alteração face ao plano de execução anterior reside no facto de deixar de ser realizado o varrimento completo à tabela YUCS. Na verdade, com criação do índice associado à introdução da restrição de chave primária, o acesso à tabela YUCS deixou de ser necessário passando a ser feito apenas o acesso ao índice. Isto é possível uma vez que, o único atributo que se projeta na consulta (codigo) faz parte do índice. Caso fosse necessário projetar outro atributo que não fosse parte integrante do índice, o acesso à tabela continuaria a ser necessário, embora fosse realizado através do índice (evitando o varrimento completo).

- z)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5403	14
HASH JOIN		RIGHT ANTI	5403	14
Access Predicates				
CODIGO=CODIGO				
INDEX	IDX_XTIPOSAULA_ANOLETIVO_COD	RANGE SCAN	1610	8
Access Predicates				
ANO_LETIVO='2003/2004'				
INDEX	PK_ZUCS	FAST FULL SCAN	5403	6

Figura 12: Plano de execução da *query* 3a., nas tabelas com prefixo Z.

Analisando o plano de execução da Figura 12, é possível verificar que com a criação do índice `IDX_ZTIPOSAULA_ANOLETIVO_COD`, com os atributos `ano_letivo` e `codigo` da tabela `ZTIPOSAULA`, se evita o acesso à tabela `ZTIPOSAULA`, sendo apenas consultado o índice.

Assim, embora a redução do custo, entre as consultas às tabelas com prefixo X e Y, tenha sido reduzido (49 para 42), verifica-se que a diminuição do custo entre as consultas às tabelas com prefixo Y e Z é bastante significativo (42 para 14).

3.3.4 Tempos

Tabela	Tempo (s)
X	0.035
Y	0.034
Z	0.033

Tabela 12: Tempo de execução da *query* 3a., nas tabelas com prefixo X, Y e Z.

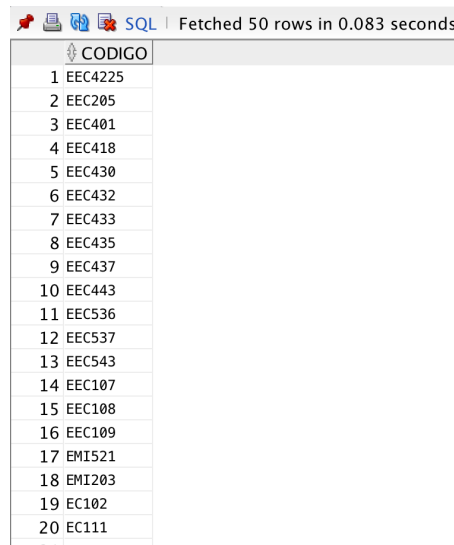
3.4 Pergunta 3b

3.4.1 SQL query

```
CREATE VIEW courses_not_sa_2003_2004
AS
SELECT *
FROM xtiposaula
WHERE codigo NOT IN (SELECT DISTINCT codigo
FROM xtiposaula
WHERE ano_letivo='2003/2004');

SELECT DISTINCT xucs.codigo
FROM xucs LEFT OUTER JOIN courses_not_sa_2003_2004 ON xucs.codigo=
courses_not_sa_2003_2004.codigo
WHERE ano_letivo IS NOT NULL;
```

3.4.2 Resposta



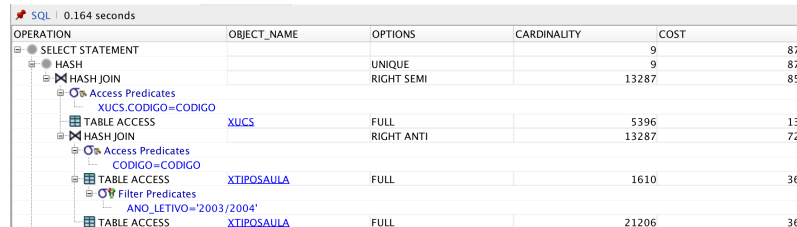
SQL | Fetched 50 rows in 0.083 seconds

	CODIGO
1	EEC4225
2	EEC205
3	EEC401
4	EEC418
5	EEC430
6	EEC432
7	EEC433
8	EEC435
9	EEC437
10	EEC443
11	EEC536
12	EEC537
13	EEC543
14	EEC107
15	EEC108
16	EEC109
17	EMI521
18	EMI203
19	EC102
20	EC111
21	...

Figura 13: Resultados da execução da consulta.

3.4.3 Análise do plano de execução

- x)



SQL | 0.164 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			9	87
HASH		UNIQUE	9	87
HASH JOIN		RIGHT SEMI	13287	85
Access Predicates				
XUCS.CODIGO=CODIGO				
TABLE ACCESS	XUCS	FULL	5396	13
HASH JOIN		RIGHT ANTI	13287	72
Access Predicates				
CODIGO=CODIGO				
TABLE ACCESS	XTIPOSAULA	FULL	1610	36
Filter Predicates				
ANO_LETIVO='2003/2004'				
TABLE ACCESS	XTIPOSAULA	FULL	21206	36

Figura 14: Plano de execução da *query* 3b., nas tabelas com prefixo X.

Conforme se constata através da análise do plano de execução ilustrado na Figura 14, é feito um varrimento completo à tabela XTIPOSAULA, aplicando o filtro correspondente à condição presente na cláusula WHERE ($\text{ano_letivo} = '2003/2004'$). Verifica-se, igualmente através da análise da Figura 14, que é feito um HASH JOIN RIGHT ANTI. Quer isto dizer que, o resultado da seleção dos tuplos da tabela XTIPOSAULA é guardado em memória, sendo aplicada uma função de *hashing* e, posteriormente, é realizado um novo varrimento completo à tabela XTIPOSAULA, selecionando apenas aqueles tuplos cujo código não se encontra nessa tabela em memória. Em seguida, é realizado um HASH JOIN RIGHT SEMI, querendo isto dizer que o resultado do join anterior é guardado em memória, sendo aplicada uma função de *hashing* e, posteriormente, é realizado um varrimento completo da tabela XUCS, selecionando apenas aqueles tuplos cujo código se encontram, pelo menos uma vez, nessa tabela em memória. Finalmente, é realizado um HASH UNIQUE que é uma implementação mais eficiente do algoritmo de ordenação para o statement DISTINCT.

- y)

SQL 0.156 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				74
HASH			13	74
NESTED LOOPS		UNIQUE	13	74
HASH JOIN		SEMI	15394	73
		RIGHT ANTI	15394	72
Access Predicates				
CODIGO=CODIGO				
TABLE ACCESS	YTIPOSAULA	FULL	1116	36
Filter Predicates				
ANO_LETIVO='2003/2004'				
TABLE ACCESS	YTIPOSAULA	FULL	21206	36
INDEX	PK_YUCS	UNIQUE SCAN	5396	0
Access Predicates				
YUCS.CODIGO=CODIGO				

Figura 15: Plano de execução da *query* 3b., nas tabelas com prefixo Y.

Como se verifica da análise do plano de execução presente na Figura 15, a única alteração face ao plano de execução anterior reside no facto de deixar de ser realizado o varrimento completo à tabela YUCS. Na verdade, com criação do índice associado à introdução da restrição de chave primária, o acesso à tabela YUCS deixou de ser necessário passando a ser feito apenas o acesso ao índice. Isto é possível uma vez que, o único atributo que se projeta na consulta (codigo) faz parte do índice.

- z)

SQL 0.808 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				35
HASH			9	35
NESTED LOOPS		UNIQUE	9	35
HASH JOIN		SEMI	13287	34
		RIGHT ANTI	13287	33
Access Predicates				
CODIGO=CODIGO				
INDEX	IDX_ZTIPOSAULA_ANOLETIVO_COD	RANGE SCAN	1610	8
Access Predicates				
ANO_LETIVO='2003/2004'				
INDEX	IDX_ZTIPOSAULA_ANOLETIVO_COD	FAST FULL SCAN	21206	25
INDEX	PK_ZUCS	UNIQUE SCAN	5403	0
Access Predicates				
ZUCS.CODIGO=CODIGO				

Figura 16: Plano de execução da *query* 3b., nas tabelas com prefixo Z.

Analisando o plano de execução da Figura 16, é possível verificar que com a criação do índice IDX_ZTIPOSAULA_ANOLETIVO_COD, com os atributos ano_letivo e codigo da tabela ZTIPOSAULA, se evita o acesso à tabela ZTIPOSAULA, sendo apenas consultado o índice.

É possível concluir que, embora a redução do custo, entre as consultas às tabelas com prefixo X e Y, tenha sido reduzido (87 para 74), verifica-se que a diminuição do custo entre as consultas às tabelas com prefixo Y e Z é bastante significativo (mais de metade) - de 74 para 35.

3.4.4 Tempos

Tabela	Tempo (s)
X	0.083
Y	0.065
Z	0.053

Tabela 13: Tempo de execução da *query* 3b., nas tabelas com prefixo X, Y e Z.

3.5 Pergunta 4

3.5.1 SQL query

```
CREATE VIEW horas_docentes
AS
SELECT xdocentes.nr, xdocentes.nome, tipo, SUM(horas*fator) AS nr_horas
FROM xdocentes JOIN xdsd ON xdsd.nr = xdocentes.nr JOIN xtiposaula ON xtiposaula.ID
    = xdsd.ID
WHERE ano_letivo='2003/2004'
GROUP BY xdocentes.nr, xdocentes.nome, tipo;

CREATE VIEW max_horas_docentes
AS
SELECT tipo, MAX(nr_horas) AS max_nr_horas
FROM horas_docentes
GROUP BY tipo;

SELECT nr, nome, max_horas_docentes.tipo, max_nr_horas
FROM horas_docentes join max_horas_docentes on horas_docentes.tipo =
    max_horas_docentes.tipo and horas_docentes.nr_horas = max_horas_docentes.
    max_nr_horas;
```

3.5.2 Resposta



NR	NOME	TIPO	MAX_NR_HORAS
1 210006	João Carlos Pascoal de Faria	OT	3,5
2 249564	Cecília do Carmo Ferreira da Silva	TP	26
3 208187	Antônio Almerindo Pinheiro Vieira	P	30
4 207638	Fernando Francisco Machado Veloso Gomes	T	30,67

Figura 17: Resultados da execução da consulta.

3.5.3 Análise do plano de execução

- x)

SQL 0.234 seconds					
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				4	141
HASH JOIN				4	141
Access Predicates					
AND					
HORAS_DOCENTES.TIPO=MAX_HORAS_DOCENTES.TIPO					
HORAS_DOCENTES.NR_HORAS=MAX_HORAS_DOCENTES.MAX_NR_HORAS					
MAX_HORAS_DOCENTES				5	70
VIEW				5	70
HASH		GROUP BY		1068	70
VIEW				1068	70
HASH		GROUP BY		2621	69
Access Predicates					
XDS.D.NR=XDOCENTES.NR					
TABLE ACCESS XDOCENTES		FULL		939	5
HASH JOIN				2621	64
Access Predicates					
XTIPOS.AULA.ID=XDS.D.ID					
TABLE ACCESS XTIPOS.AULA		FULL		1610	36
Filter Predicates					
XTIPOS.AULA.ANO_LETIVO='2003/2004'					
TABLE ACCESS XDS.D		FULL		27765	28
VIEW				1068	70
HASH		GROUP BY		1068	70
HASH JOIN				2621	69
Access Predicates					
XDS.D.NR=XDOCENTES.NR					
TABLE ACCESS XDOCENTES		FULL		939	5
HASH JOIN				2621	64
Access Predicates					
XTIPOS.AULA.ID=XDS.D.ID					
TABLE ACCESS XTIPOS.AULA		FULL		1610	36
Filter Predicates					
XTIPOS.AULA.ANO_LETIVO='2003/2004'					
TABLE ACCESS XDS.D		FULL		27765	28

Figura 18: Plano de execução da *query* 4, nas tabelas com prefixo X.

Conforme é possível verificar através da análise do plano de execução ilustrado na Figura 18, é realizado, em primeiro lugar, um varrimento completo da tabela XTIPOS.AULA, selecionando os tuplos que cumpram a condição presente na cláusula WHERE (ano_letivo = '2003/2004'). É, igualmente, realizado um varrimento completo da tabela XDS.D. Em seguida, é realizada a junção das duas tabelas, com recurso ao HASH JOIN, através do atributo ID. Posteriormente, a tabela XDOCENTES é varrida por completo e, em seguida, é realizada a junção desta tabela com o resultado da junção anterior, através do atributo NR e com recurso a um HASH JOIN. Finalmente é realizado o agrupamento (GROUP BY) do resultado da junção anterior, pelos atributos NR, NOME e TIPO, e o resultado é guardado numa view (HORAS_DOCENTES). Em seguida, o resultado dessa view é agrupado pelo atributo tipo, projetando o atributo tipo e o MAX(nr_horas), sendo o respetivo resultado guardado noutra view (MAX_HORAS_DOCENTES). Finalmente, é feita a seleção dos tuplos que cumpram simultaneamente as condições de junção, com recurso ao operador AND, e é efetuada a junção com recurso a um HASH JOIN.

- y)

SQL 0.309 seconds					
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				4	141
HASH JOIN				4	141
Access Predicates					
AND					
HORAS_DOCENTES.TIPO=MAX_HORAS_DOCENTES.TIPO					
HORAS_DOCENTES.NR_HORAS=MAX_HORAS_DOCENTES.MAX_NR_HORAS					
MAX_HORAS_DOCENTES			5	70	
VIEW					
HASH		GROUP BY	5	70	
VIEW			1068	70	
HASH		GROUP BY	1068	70	
HASH JOIN			2621	69	
Access Predicates					
YDSD.NR=YDOCENTES.NR					
TABLE ACCESS YDOCENTES		FULL	939	5	
HASH JOIN			2621	64	
Access Predicates					
YTIPOSAULA.ID=YDSD.ID					
TABLE ACCESS YTIPOSAULA		FULL	1610	36	
Filter Predicates					
YTIPOSAULA.ANO_LETIVO='2003/2004'					
TABLE ACCESS YDSD		FULL	27765	28	
VIEW					
HASH		GROUP BY	1068	70	
HASH JOIN			1068	70	
HASH JOIN			2621	69	
Access Predicates					
YDSD.NR=YDOCENTES.NR					
TABLE ACCESS YDOCENTES		FULL	939	5	
HASH JOIN			2621	64	
Access Predicates					
YTIPOSAULA.ID=YDSD.ID					
TABLE ACCESS YTIPOSAULA		FULL	1610	36	
Filter Predicates					
YTIPOSAULA.ANO_LETIVO='2003/2004'					
TABLE ACCESS YDSD		FULL	27765	28	

Figura 19: Plano de execução da *query* 4, nas tabelas com prefixo Y.

Conforme se constata da análise do plano de execução ilustrado na Figura 19, não existem alterações relativamente ao plano de execução anterior. Apesar da criação dos índices, resultantes da introdução das restrições de chave primária, como o acesso nesta consulta não é realizado pelos atributos indexados, os mesmos não surtem qualquer efeito.

• z)

SQL 0.675 seconds					
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT			19	131	
HASH JOIN				19	131
Access Predicates					
AND					
HORAS_DOCENTES.TIPO=MAX_HORAS_DOCENTES.TIPO					
HORAS_DOCENTES.NR_HORAS=MAX_HORAS_DOCENTES.MAX_NR_HORAS					
MAX_HORAS_DOCENTES			5	65	
VIEW					
HASH		GROUP BY	5	65	
VIEW			1068	65	
HASH		GROUP BY	1068	65	
HASH JOIN			2621	64	
Access Predicates					
ZDSD.NR=ZDOCENTES.NR					
INDEX		FAST FULL SCAN	939	4	
HASH JOIN			2621	60	
Access Predicates					
ZTIPOSAULA.ID=ZDSD.ID					
TABLE ACCESS ZTIPOSAULA		BY INDEX ROWID BATCHED	1610	32	
INDEX		RANGE SCAN	1610	5	
Access Predicates					
ZTIPOSAULA.ANO_LETIVO='2003/2004'					
TABLE ACCESS ZDSD		FULL	27765	28	
VIEW					
HASH		GROUP BY	1068	65	
HASH JOIN			1068	65	
HASH JOIN			2621	64	
Access Predicates					
ZDSD.NR=ZDOCENTES.NR					
INDEX		FAST FULL SCAN	939	4	
HASH JOIN			2621	60	
Access Predicates					
ZTIPOSAULA.ID=ZDSD.ID					
TABLE ACCESS ZTIPOSAULA		BY INDEX ROWID BATCHED	1610	32	
INDEX		RANGE SCAN	1610	5	
Access Predicates					
ZTIPOSAULA.ANO_LETIVO='2003/2004'					
TABLE ACCESS ZDSD		FULL	27765	28	

Figura 20: Plano de execução da *query* 4, nas tabelas com prefixo Z.

Como se verifica através da análise do plano de execução presente na Figura 20, com a criação do índice `IDX_ZTIPOSAULA_ANOLETIVO`, no atributo ano letivo da tabela `ZTIPOSAULA`, evita-se o varrimento completo desta tabela, sendo que o acesso passa a ser feito através do índice. Por outro lado, com a criação do índice `IDX_ZDOCENTES_NR_NOME`, nos atributos `nr` e `nome` da tabela `ZDOCENTES`, evita-se não só o varrimento completo desta tabela, como o próprio acesso à tabela. O acesso é, assim, realizado exclusivamente ao índice, uma vez que os atributos que são necessários projetar são parte integrante do mesmo.

3.5.4 Tempos

Tabela	Tempo (s)
X	0.064
Y	0.06
Z	0.056

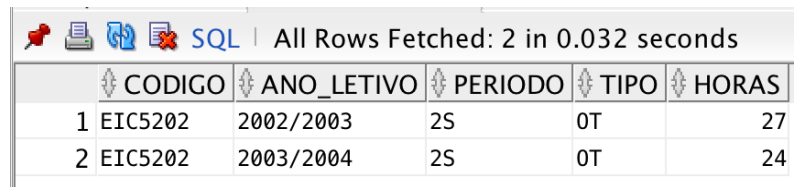
Tabela 14: Tempo de execução da *query* 4, nas tabelas com prefixo X, Y e Z.

3.6 Pergunta 5

3.6.1 SQL query

```
SELECT zucs.codigo, ztiposaula.ano_letivo, ztiposaula.periodo, tipo, SUM(turnos*
    horas_turno) AS horas
FROM zucs JOIN ztiposaula ON zucs.codigo = ztiposaula.codigo
WHERE tipo = 'OT' AND (ano_letivo = '2002/2003' OR ano_letivo = '2003/2004')
GROUP BY zucs.codigo, ztiposaula.ano_letivo, ztiposaula.periodo, tipo;
```

3.6.2 Resposta

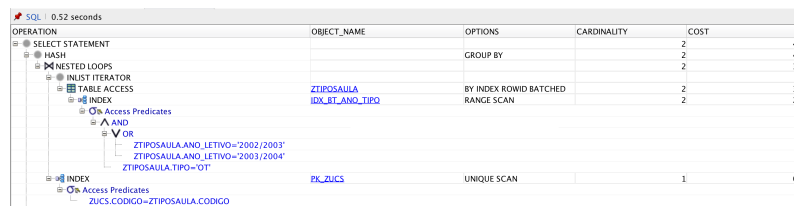


	CODIGO	ANO_LETIVO	PERIODO	TIPO	HORAS
1	EIC5202	2002/2003	2S	OT	27
2	EIC5202	2003/2004	2S	OT	24

Figura 21: Resultados da execução da consulta.

3.6.3 Análise do plano de execução

(a) Índice *B-Tree*)



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			2	4
HASH		GROUP BY	2	4
NESTED LOOPS			2	3
INSTEAD OF TABLE ACCESS	ZTIPOSAULA	BY INDEX ROWID BATCHED	2	3
INDEX	IDX_BT_ANO_TIPO	RANGE SCAN	2	2
Access Predicates				
AND				
OR				
ZTIPOSAULA.ANO_LETIVO='2002/2003'				
ZTIPOSAULA.ANO_LETIVO='2003/2004'				
ZTIPOSAULA.TIPO='OT'				
INDEX	PK_ZUCS	UNIQUE SCAN	1	0
Access Predicates				
ZUCS.CODIGO=ZTIPOSAULA.CODIGO				

Figura 22: Plano de execução da *query* 5a., nas tabelas com prefixo Z.

Como se verifica através da análise do plano de execução presente na Figura 22, o acesso à tabela ZTIPOSAULA é feito através do índice, selecionando os tuplos que cumpram as condições presentes na cláusula WHERE (tipo = 'OT' AND (ano_letivo = '2002/2003' OR ano_letivo = '2003/2004')). O acesso à tabela é realizado, dado que é necessário projetar mais atributos para além dos que fazem parte do índice. Em seguida, é feito o acesso ao índice da tabela ZUCS, criado com a introdução da restrição chave primária. É feita a

junção das tabelas com recurso aos NESTED LOOPS. Finalmente, é realizada a operação de agrupamento (GROUP BY) para os atributos codigo, ano_letivo, periodod e tipo.

Name	Value
1 OWNER	UP200706629
2 INDEX_NAME	IDX_BT_ANO_TIPO
3 TABLE_OWNER	UP200706629
4 TABLE_NAME	ZTIPOSAULA
5 PARTITION_NAME	(null)
6 PARTITION_POSITION	(null)
7 SUBPARTITION_NAME	(null)
8 SUBPARTITION_POSITION	(null)
9 OBJECT_TYPE	INDEX
10 BLEVEL	1
11 LEAF_BLOCKS	69
12 DISTINCT_KEYS	68
13 AVG_LEAF_BLOCKS_PER_KEY	1
14 AVG_DATA_BLOCKS_PER_KEY	12
15 CLUSTERING_FACTOR	839
16 NUM_ROWS	21206
17 AVG_CACHED_BLOCKS	(null)
18 AVG_CACHE_HIT_RATIO	(null)
19 SAMPLE_SIZE	21206
20 LAST_ANALYZED	18.04.07
21 GLOBAL_STATS	YES
22 USER_STATS	NO
23 STATTYPE_LOCKED	(null)
24 STALE_STATS	NO
25 SCOPE	SHARED

Figura 23: Estatísticas sobre o índice *B-Tree* IDX_BT_ANO_TIPO.

Conforme se pode verificar pela análise da Figura 23, o índice em questão é um índice *B-Tree* com apenas 2 níveis - o atributo BLEVEL indica o número de níveis intermédios, incluindo a raiz. Este índice ocupa 69 blocos - atributo LEAF_BLOCKS. O fator de agrupamento (CLUSTERING_FACTOR) dá uma estimativa do número de páginas a ler para efetuar um varrimento do índice. Se o fator de agrupamento for próximo do número de páginas da tabela, significa que esta está agrupada, por outro lado, se o fator de agrupamento for próximo do número de entradas no índice, significa que os registos estão espalhados, sem ordenação, e que o custo de os ler todos é muito elevado. Este valor é utilizado na escolha do otimizador de consultas. Ao dar uma estimativa do número de páginas a ler, o valor do fator de agrupamento é um critério importante na escolha do plano de consulta mais eficiente. Neste caso, como o valor do fator de agrupamento (CLUSTERING_FACTOR = 839) não está próximo do valor do número de entradas no índice (NUM_ROWS = 21206) pressupõe-se que os registos não se encontram espalhados e que estão ordenados.

(b) Índice *Bitmap*)

SQL 0.72 seconds				
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			2	3
HASH		GROUP BY	2	3
NESTED LOOPS			2	2
INLIST ITERATOR				
TABLE ACCESS	ZTIPOSAULA	BY INDEX ROWID BATCHED	2	2
BITMAP CONVERSION		TO ROWIDS		
BITMAP INDEX	IDX_BITMAP_ANO_TIPO	SINGLE VALUE		
Access Predicates				
AND				
OR				
ZTIPOSAULA.ANO_LETIVO=2002/2003				
ZTIPOSAULA.ANO_LETIVO=2003/2004				
ZTIPOSAULA.TIPO=OT				
INDEX	PK_ZUCS	UNIQUE SCAN	1	0
Access Predicates				
ZUCS.CODIGO=ZTIPOSAULA.CODIGO				

Figura 24: Plano de execução da *query* 5b., nas tabelas com prefixo Z.

Como é possível constatar através da análise do plano de execução ilustrado na Figura 24, a única diferença introduzida pelo índice *Bitmap* prende-se com a diminuição do custo de acesso ao índice (3 para 2).

Actions...	
Name	Value
1 OWNER	UP200706629
2 INDEX_NAME	IDX_BITMAP_ANO_TIPO
3 TABLE_OWNER	UP200706629
4 TABLE_NAME	ZTIPOSAULA
5 PARTITION_NAME	(null)
6 PARTITION_POSITION	(null)
7 SUBPARTITION_NAME	(null)
8 SUBPARTITION_POSITION	(null)
9 OBJECT_TYPE	INDEX
10 BLEVEL	1
11 LEAF_BLOCKS	2
12 DISTINCT_KEYS	68
13 AVG_LEAF_BLOCKS_PER_KEY	1
14 AVG_DATA_BLOCKS_PER_KEY	1
15 CLUSTERING_FACTOR	68
16 NUM_ROWS	68
17 AVG_CACHED_BLOCKS	(null)
18 AVG_CACHE_HIT_RATIO	(null)
19 SAMPLE_SIZE	68
20 LAST_ANALYZED	18.04.07
21 GLOBAL_STATS	YES
22 USER_STATS	NO
23 STATTYPE_LOCKED	(null)
24 STALE_STATS	NO
25 SCOPE	SHARED

Figura 25: Estatísticas sobre o índice *Bitmap* IDX_BITMAP_ANO_TIPO.

O SGBD ORACLE utiliza a estrutura dos índices *B-Tree* para guardar cada chave indexada pelos índices *Bitmap*. Assim, e conforme se pode verificar pela análise da Figura 25, o índice em questão é um índice *B-Tree* com apenas 2 níveis - o atributo BLEVEL indica o número de níveis intermédios, incluindo a raiz - e ocupa apenas 2 blocos (atributo LEAF_BLOCKS). Relativamente ao fator de agrupamento, o seu valor (CLUSTERING_FACTOR = 68) é igual ao número de entradas no índice (NUM_ROWS = 68) pelo que se pressupõe que os registos se encontram espalhados e não ordenados.

3.6.4 Tempos

Query	Tempo (s)
a.	0.032
b.	0.028

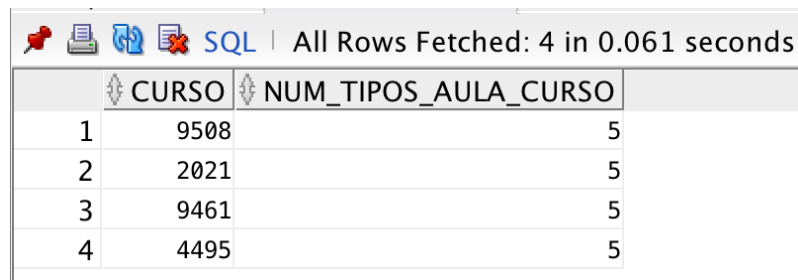
Tabela 15: Tempo de execução das *queries* 5a. e 5b., nas tabela com prefixo Z.

3.7 Pergunta 6

3.7.1 SQL query

```
SELECT curso , COUNT(curso) AS num_tipos_aula_curso
FROM (SELECT curso , tipo
FROM xucs JOIN xtiposaula ON xucs.codigo=xtiposaula.codigo
GROUP BY curso , tipo)
GROUP BY curso
HAVING COUNT(curso) = (SELECT COUNT (DISTINCT tipo) AS num_tipos_aula
FROM xtiposaula);
```

3.7.2 Resposta

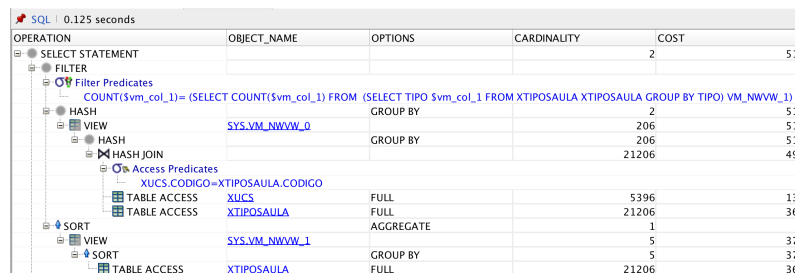


	CURSO	NUM_TIPOS_AULA_CURSO
1	9508	5
2	2021	5
3	9461	5
4	4495	5

Figura 26: Resultados da execução da consulta.

3.7.3 Análise do plano de execução

- x)



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			2	51
FILTER				
Filter Predicates				
COUNT(\$vm_col_1) = (SELECT COUNT(\$vm_col_1) FROM (SELECT TIPO \$vm_col_1 FROM XTIPOSAULA XTIPOSAULA GROUP BY TIPO) VM_NWWW_1)				
HASH		GROUP BY	2	51
VIEW	SYS.VM_NWWW_0		206	51
HASH		GROUP BY	206	51
HASH JOIN			21206	49
Access Predicates				
XUCS.CODIGO=XTIPOSAULA.CODIGO				
TABLE ACCESS	XUCS	FULL	5396	13
TABLE ACCESS	XTIPOSAULA	FULL	21206	36
SORT		AGGREGATE	1	
VIEW	SYS.VM_NWWW_1		5	37
SORT		GROUP BY	5	37
TABLE ACCESS	XTIPOSAULA	FULL	21206	36

Figura 27: Plano de execução da *query* 6, nas tabelas com prefixo X.

Através da análise do plano de execução presente na Figura 27, é possível constatar que em

primeiro lugar são realizados dois varrimentos completos - um à tabela XUCS e outro à tabela XTIPOSAULA - tendo em vista a realização da junção entre as duas tabelas, com recurso a um HASH JOIN no atributo codigo. Em seguida, é realizado o agrupamento (GROUP BY) pelos atributos curso e tipo, e o resultado é guardado numa vista (SYS.VM_NWVW_0). Posteriormente, é realizado novo varrimento completo da tabela XTIPOSAULA, sendo o resultado agrupado (GROUP BY) de modo a ser possível obter o número de tipos de aula diferentes (SELECT COUNT (DISTINCT tipo)). Esse resultado é guardado numa vista - SYS.VM_NWVW_1. Finalmente é aplicado o filtro referente à cláusula HAVING, por forma a serem projetados apenas os tuplos que respeitem essa cláusula.

• y)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			2	51
FILTER				
Filter Predicates				
COUNT(\$vm_col_1)= (SELECT COUNT(\$vm_col_1) FROM (SELECT TIPO \$vm_col_1 FROM YTIPOSAULA GROUP BY TIPO) VM_NWVW_1)				
HASH		GROUP BY	2	51
VIEW	SYS.VM_NWVW_0		206	51
HASH		GROUP BY	206	51
HASH JOIN			21206	49
Access Predicates				
YUCS.CODIGO=YTIPOSAULA.CODIGO				
TABLE ACCESS	YUCS	FULL	5396	13
TABLE ACCESS	YTIPOSAULA	FULL	21206	36
AGGREGATE			1	
VIEW	SYS.VM_NWVW_1		5	37
GROUP BY			5	37
TABLE ACCESS	YTIPOSAULA	FULL	21206	36

Figura 28: Plano de execução da *query* 6, nas tabelas com prefixo Y.

Conforme se constata da análise do plano de execução ilustrado na Figura 28, não existem alterações relativamente ao plano de execução anterior. Apesar da criação dos índices, resultantes da introdução das restrições de chave primária, como o acesso nesta consulta não é realizado pelos atributos indexados, os mesmos não surtem qualquer efeito.

• z)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			2	33
FILTER				
Filter Predicates				
COUNT(\$vm_col_1)= (SELECT COUNT(\$vm_col_1) FROM (SELECT TIPO \$vm_col_1 FROM ZTIPOSAULA GROUP BY TIPO) VM_NWVW_1)				
HASH		GROUP BY	2	33
VIEW	SYS.VM_NWVW_0		393	33
HASH		GROUP BY	393	33
HASH JOIN			21206	31
Access Predicates				
ZUCS.CODIGO=ZTIPOSAULA.CODIGO				
TABLE ACCESS	ZUCS	FULL	21206	31
INDEX	IDX_ZUCS_CURSO_DESG_COD	FAST FULL SCAN	5403	12
INDEX	IDX_ZTIPOSAULA_COD_TIPO	RANGE SCAN	4	19
TABLE ACCESS	ZTIPOSAULA	FULL	21206	31
VIEW	SYS.VM_NWVW_1		5	3
GROUP BY			5	3
TABLE ACCESS	ZTIPOSAULA	FULL	21206	31

Figura 29: Plano de execução da *query* 6, nas tabelas com prefixo Z.

Da análise do plano de execução presente na Figura 29, é possível verificar que com a criação do índice IDX_ZTIPOSAULA_COD_TIPO, com os atributos codigo e tipo da tabela ZTIPOSAULA, evita-se não só o varrimento completo da tabela, como o próprio acesso à tabela. O acesso é feito apenas ao índice. De igual forma, com a criação do índice IDX_ZUCS_CURSO_DESG_COD, com os atributos curso, designacao e codigo da tabela ZUCS, se evita não só o varrimento completo, como o acesso à tabela - passando

o acesso a ser realizado apenas ao índice. Por sua vez, com a criação do índice *bitmap* `IDX_ZTIPOSAULA_BITMAP_TIPO`, com o atributo `tipo` na tabela `ZTIPOSAULA`, evita-se o segundo varrimento completo da tabela `ZTIPOSAULA`. Conforme mencionado anteriormente, neste caso optou-se pela criação de um índice *bitmap* devido à baixa cardinalidade do atributo `tipo` (5).

Assim, embora não se tenha verificado redução do custo, entre as consultas às tabelas com prefixo X e Y, já entre as consultas às tabelas com prefixo Y e Z a redução é de assinalar (51 para 33).

3.7.4 Tempos

Tabela	Tempo (s)
X	0.061
Y	0.057
Z	0.045

Tabela 16: Tempo de execução da *query* 6, nas tabelas com prefixo X, Y e Z.

4 Conclusão

Após a realização do presente trabalho, é possível concluir que a correta utilização de índices é essencial para um bom desempenho da base de dados e que a sua definição deve ser considerada uma etapa essencial da concepção física. É de salientar, no entanto, que o desempenho da utilização de índices pode variar, dependendo de como o otimizador de consultas os utiliza.

Referências

- [F.14] Gouveia F. Fundamentos de bases de dados. *FCA - Lisboa*, 2014.
- [G.18] David G. Query optimization - teaching service. *Database Technology - Integrated Master in Informatics Engineering, Faculty of Engineering of the University of Porto*, 2018.