

Otimização de Interrogações SQL

Alexandre Santos
up202108671@edu.fe.up.pt
Faculdade de Engenharia da
Universidade do Porto

Emanuel Maia
up202107486@edu.fe.up.pt
Faculdade de Engenharia da
Universidade do Porto

8 de abril de 2025

Resumo

Este relatório analisa os planos de execução de diferentes interrogações SQL sobre uma base de dados de eleições. O impacto da existência de índices e de estatísticas, bem como a utilização de diferentes estratégias de estruturação das perguntas, são avaliados.

1 Introdução

Este relatório documenta a análise de planos de execução para diversas interrogações SQL realizadas sobre a base de dados fornecida. São explorados três contextos distintos de execução, com e sem índices e restrições de integridade.

2 Descrição da Base de Dados

A base de dados contém informações sobre as eleições legislativas de 1999 em Portugal, organizadas em sete tabelas:

- **DISTRITOS** (codigo, nome, regioao)
- **CONCELHOS** (codigo, nome, distrito)
- **FREGUESIAS** (codigo, nome, concelho)
- **PARTIDOS** (sigla, designacao)
- **VOTACOES** (partido, freguesia, votos)
- **LISTAS** (distrito, partido, mandatos)
- **PARTICIPACOES** (distrito, inscritos, votantes, abstenções, brancos, nulos)

3 Metodologia

As interrogações foram executadas em três contextos distintos:

- **X**: Sem índices nem restrições de integridade.
- **Y**: Com restrições de integridade (chave primária e chave externa).
- **Z**: Com restrições de integridade e índices adicionais.

4 Análise das Interrogações

4.1 Seleção e Junção

4.1.1 Query 1a - Quais os códigos e nomes de freguesias do concelho 1103? E do concelho “Azambuja”?

	CODIGO	NOME
1	110308	Vila Nova de São Pedro
2	110309	Maçussa
3	110301	Alcoentre
4	110302	Aveiras de Baixo
5	110303	Aveiras de Cima
6	110304	Azambuja
7	110305	Manique do Intendente
8	110306	Vale do Paraíso
9	110307	Vila Nova da Rainha

Figura 1: Resultados da query 1a

```
SELECT codigo, nome
FROM xfreguesias
WHERE concelho = 1103;
```

Plano de Execução: A query procura as entradas com o código 1103 na tabela `xfreguesias` e retorna o código e nome das freguesias correspondentes.

- **Contexto X:** Realiza uma varredura completa da tabela (*full table scan*) devido à ausência de índices, tendo 7 de custo.
- **Contexto Y:** Apesar da chave primária em `codigo`, mantém a varredura completa devido à falta de índice em `concelho`, tendo também 7 de custo.
- **Contexto Z:** Com um índice em `concelho`, realiza uma busca por índice (*index range scan*), reduzindo o custo de 7 para apenas 2.

Contexto	Custo
X	7
Y	7
Z	2

Tabela 1: Custos da query 1a (pesquisa por código)

Conclusões: Verifica-se que a criação de um índice na coluna `concelho` da tabela `xfreguesias` tem um impacto significativo na eficiência da query, reduzindo o custo de execução de 7 para 2. O índice permite aceder diretamente às entradas relevantes sem ter de percorrer todos os registos da tabela, o que poupa tempo e recursos computacionais.

```
SELECT f.codigo, f.nome
FROM xfreguesias f, xconcelhos c
WHERE f.concelho = c.codigo
      AND c.nome = 'Azambuja';
```

Plano de Execução: A query realiza uma junção entre `xfreguesias` e `xconcelhos` para aceder aos nomes dos concelhos, filtrando pelo nome do concelho "Azambuja", retornando também o código e nome das freguesias desse concelho.

- **Contexto X:** Junta as tabelas sem índices, varrendo ambas completamente, com custo 10.
- **Contexto Y:** Apesar da chave externa em `f.concelho`, o facto de não ter índice ainda implica varredura completa em `yfreguesias` e `yconcelhos`, tendo também custo 10.
- **Contexto Z:** Índices em `zconcelhos.nome` e `zfreguesias.concelho` permitem busca rápida e junção eficiente (*index join*), reduzindo o custo para apenas 4.

Contexto	Custo
X	10
Y	10
Z	4

Tabela 2: Custos da query 1a (pesquisa por nome)

Conclusões: O índice em `zconcelhos.nome` no contexto Z otimiza a filtragem inicial, reduzindo o número de linhas juntadas. Além disso, o índice em `zfreguesias.concelho` permite que a junção entre as tabelas seja feita de forma eficiente, evitando varreduras completas e permitindo acesso direto aos registos relevantes. Isto demonstra como a presença de índices nas colunas usadas em filtros e junções pode ter um impacto significativo na redução do custo da execução da query.

4.1.2 Query 1b - Indique as siglas e designações dos partidos e o respetivo número de mandatos obtidos no distrito de Lisboa.

	⚡ SIGLA	⚡ DESIGNACAO	⚡ MANDATOS
1	PSN	Partido Solidariedade Nacional	0
2	PS	Partido Socialista	23
3	PPM	Partido Popular Monárquico	0
4	POUS	Partido Operário de Unidade Socialista	0
5	PH	Partido Humanista	0
6	PPDPSD	Partido Social Democrata	14
7	PCTPMRPP	Partido Comunista dos Trabalhadores Portugueses	0
8	PCPPEV	Partido Comunista Português	6
9	CDSPP	Partido Popular	4
10	BE	Bloco de Esquerda	2
11	MPT	Movimento Partido da Terra	0

Figura 2: Resultados da query 1b

```
SELECT p.sigla, p.designacao, l.mandatos
FROM xlistas l, xdistritos d, xpartidos p
WHERE l.distrito = d.codigo AND l.partido = p.sigla
      AND d.nome = 'Lisboa';
```

Plano de Execução: Esta query retorna as siglas, designações e número de mandatos dos partidos que concorreram no distrito de Lisboa. Para isso, realiza uma junção entre três tabelas: `xlistas`, que contém os mandatos por distrito e partido; `xdistritos`, que associa códigos a

nomes de distritos; e `xpartidos`, que fornece as siglas e designações dos partidos. A condição `d.nome = 'Lisboa'` filtra os resultados apenas para o distrito de Lisboa, enquanto as condições de junção garantem a correspondência entre distritos, partidos e mandatos.

- **Contexto X:** Varredura completa nas três tabelas com junção *hash*, custo 9.
- **Contexto Y:** Chaves primárias ajudam na validação, conseguindo, mesmo sem índices em `ydistritos.nome` e nas chaves estrangeiras, reduzir o custo da interrogação de 9 para 8.
- **Contexto Z:** Índice Bitmap em `zdistritos.nome` permite procura rápida, seguido de junções otimizadas por índices em `zlistas.distrito` e `zlistas.partido`, permitem diminuir o custo da interrogação para apenas 6.

Contexto	Custo
X	9
Y	8
Z	6

Tabela 3: Custos da query 1b

Conclusões: A presença de índices nas chaves estrangeiras no contexto Z reduz drasticamente o custo da junção tripla. Para além disso, a baixa cardinalidade da coluna `zdistritos.nome` (20 distritos diferentes) permite-nos usufruir da eficiência de um índice Bitmap, facilitando assim a filtragem por nome de distrito.

4.1.3 Query 1c - Indique o número de votos obtido pelo BE nas freguesias do distrito de Lisboa.

Nome	Votos
1 A Sto Estevão	71
2 A Triana	55
3 Abrigada	29
4 Aldeia Galega da Merceana	5
5 Aldeia Gavinha	7
6 Cabanas de Torres	3
7 Cadafais	16
8 Carnota	14
9 Carregado	96
10 Meca	3

Figura 3: Resultados da query 1c (10 primeiros)

```
SELECT f.nome, v.votos
FROM xfreguesias f, xconcelhos c, xdistritos d, xvotacoes v
WHERE f.concelho = c.codigo AND c.distrito = d.codigo AND f.codigo = v.freguesia
AND d.nome = 'Lisboa' AND v.partido = 'BE';
```

Plano de Execução: Esta query lista os nomes das freguesias do distrito de Lisboa e o número de votos obtidos pelo partido Bloco de Esquerda (BE) em cada uma delas. Envolve uma junção entre quatro tabelas: `xfreguesias` (nomes das freguesias), `xconcelhos` (ligação

freguesias-concelhos), `xdistritos` (ligação concelhos-distritos), e `xvotacoes` (votos por freguesia e partido). As condições `d.nome = 'Lisboa'` e `v.partido = 'BE'` restringem os resultados ao distrito de Lisboa e ao partido BE, respetivamente.

- **Contexto X:** Varredura completa em todas as tabelas com junção lenta, custo 45.
- **Contexto Y:** Chaves primárias ajudam na consistência e permitem reduzir ligeiramente o custo da interrogação para 43, mas sem índices específicos a varredura continua a ser extensa.
- **Contexto Z:** Índices em `zdistritos.nome`, `zvotacoes.partido`, `zvotacoes.freguesia`, `zfreguesias.concelho` e `zconcelhos.distrito` otimizam a filtragem e junção, reduzindo o custo para 37.

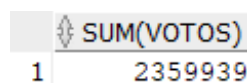
Contexto	Custo
X	45
Y	43
Z	37

Tabela 4: Custos da query 1c

Conclusões: O contexto Z melhora o desempenho com o uso de índices para junções como `zvotacoes.freguesia`, `zfreguesias.concelho` e `zconcelhos.distrito`. A filtragem também é melhorada usando um índice Bitmap em `zdistritos.nome` e um índice B-tree (padrão) em `zvotacoes.partido`. Assim a quantidade de varrimentos completos é reduzida, melhorando a eficiência da query.

4.2 Agregação

4.2.1 Query 2a - Quantos votos teve o 'PS' a nível nacional?



	SUM(VOTOS)
1	2359939

Figura 4: Resultados da query 2a

```
SELECT SUM(votos)
FROM xvotacoes
WHERE partido = 'PS';
```

Plano de Execução: Esta query calcula o total de votos obtidos pelo Partido Socialista (PS) em todas as freguesias de Portugal. Utiliza a tabela `xvotacoes`, que regista os votos por partido e freguesia, e aplica a função de agregação `SUM(votos)` para somar os votos, filtrando apenas as entradas onde `partido = 'PS'`. É uma operação simples que não envolve junções, mas depende da eficiência do acesso aos dados relevantes.

- **Contexto X:** Varredura completa para somar os votos, 32 de custo.
- **Contexto Y:** Igual a X, mas o conjunto de chaves primárias/estrangeiras (`freguesia`, `partido`) permite reduzir o custo para 30.
- **Contexto Z:** Índice em `partido` permite busca seletiva antes da agregação, reduzindo o custo para 27.

Contexto	Custo
X	32
Y	30
Z	27

Tabela 5: Custos da query 2a

Conclusões: O índice associado ao conjunto de chaves primárias (*freguesia*, *partido*) e o índice em *partido* no contexto Z facilita a agregação e evita um varrimento completo na filtragem por partido, daí a melhoria no desempenho.

4.2.2 Query 2b - Quantos votos teve cada partido, em cada distrito?

NOME	PARTIDO	SUM(V.VOTOS)
1 Setúbal	MPT	1711
2 Braga	MPT	973
3 Viana do Castelo	MPT	448
4 Vila Real	CDSPP	8599
5 Madeira	MPT	475
6 Évora	PSN	196
7 Portalegre	PSN	119
8 Guarda	PPDPSD	40024
9 Lisboa	PS	480410
10 Bragança	PS	32588

Figura 5: Resultados da query 2b (10 primeiros)

```
SELECT d.nome, v.partido, SUM(v.votos)
FROM xfreguesias f, xconcelhos c, xdistritos d, xvotacoes v
WHERE f.concelho = c.codigo AND c.distrito = d.codigo AND f.codigo = v.freguesia
GROUP BY d.nome, v.partido;
```

Plano de Execução: Esta query calcula o total de votos por partido em cada distrito, agregando os dados ao nível distrital. Junta quatro tabelas: *xfreguesias* (freguesias), *xconcelhos* (concelhos), *xdistritos* (distritos), e *xvotacoes* (votos por partido em cada freguesia). As condições de junção ligam freguesias a concelhos, concelhos a distritos e freguesias a votos, enquanto *GROUP BY d.nome, v.partido* organiza os resultados por nome do distrito e sigla do partido, somando os votos com *SUM(v.votos)*.

- **Contexto X:** Varredura completa em todas as tabelas e junção lenta antes da agregação, com custo 47.
- **Contexto Y:** Similar a X, mas com verificação de integridade devido às chaves externas e primárias, que também permitem reduzir ligeiramente o custo para 45.
- **Contexto Z:** Índices em chaves estrangeiras aceleram o processo de junção e juntamente com o índice *Bitmap* em *zdistritos.nome* a agregação é feita em menos linhas, reduzindo o custo da interrogação para 44.

Conclusões: Índices em chaves primárias e estrangeiras bem como o índice *Bitmap* em *zdistritos.nome* em Z diminuem o custo da junção e agrupamento. Ainda que não evitando completamente os varrimentos completos, permitem tipos de junção mais eficientes como *nested loops* que melhoram o desempenho da query.

Contexto	Custo
X	47
Y	45
Z	44

Tabela 6: Custos da query 2b

4.2.3 Query 2c - Qual o partido que, ao nível de freguesia, registou o maior número de votos? Indique a sigla do partido, o nome da freguesia e os votos correspondentes.

	PARTIDO	NOME	VOTOS
1	PS	Agualva-Cacem	15188

Figura 6: Resultados da query 2c

```
SELECT v.partido, f.nome, v.votos
FROM xfreguesias f, xvotacoes v
WHERE f.codigo = v.freguesia
      AND v.votos = (SELECT MAX(votos) FROM xvotacoes);
```

Plano de Execução: Esta query identifica o partido com o maior número de votos ao nível de freguesia, retornando o nome da freguesia, a sigla do partido vencedor e o número de votos. Junta *xfreguesias* (nomes das freguesias) com *xvotacoes* (votos por freguesia e partido), usando uma subconsulta correlacionada para comparar os votos de cada partido na mesma freguesia com o valor máximo obtido, determinado por *MAX(votos)*.

- **Contexto X:** Subconsulta executada repetidamente com varredura completa, com custo total de 72.
- **Contexto Y:** Índices das chaves primárias permitem acessos mais eficientes, fazendo uso de *nested loops* e reduzindo o custo da operação para 68.
- **Contexto Z:** O índice em *zvotacoes.votos* permite um aumento substancial na eficiência da query na comparação dos votos de cada partido na mesma freguesia com o valor máximo obtido, reduzindo o custo de 68 para 17.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				68
HASH JOIN				37
Access Predicates				
F.CODIGO=V.FREGUESIA				
NESTED LOOPS			22	37
NESTED LOOPS				
STATISTICS COLLECTOR				
TABLE ACCESS	YVOTACOES	FULL	22	30
Filter Predicates				
V.VOTOS= (SELECT MAX(VOTOS) FROM YVOTACOES YVOTACOES)				
SORT		AGGREGATE	1	
TABLE ACCESS	YVOTACOES	FULL	39874	30
INDEX	PK_YFREGUESIAS	UNIQUE SCAN		
Access Predicates				
F.CODIGO=V.FREGUESIA				
TABLE ACCESS	YFREGUESIAS	BY INDEX ROWID	1	7
TABLE ACCESS	YFREGUESIAS	FULL	4241	7

Figura 7: Plano de execução da query 2c no contexto Y

Conclusões: O índice em *zvotacoes.votos* no contexto Z melhora a eficiência da comparação com a subconsulta, evitando um varrimento completo em *zvotacoes* e assim melhorando consideravelmente o desempenho da query.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				17
HASH JOIN			22	15
Access Predicates F.CODIGO=V.FREGUESIA				
NESTED LOOPS			22	15
NESTED LOOPS				
STATISTICS COLLECTOR				
TABLE ACCESS	ZVOTACOES	BY INDEX ROWID BATCHED	22	8
INDEX	IDX_ZVOTACOES_VOTOS	RANGE SCAN	22	1
Access Predicates V.VOTOS=(SELECT MAX(VOTOS) FROM ZVOTACOES ZVOTACOES)				
SORT		AGGREGATE	1	
INDEX	IDX_ZVOTACOES_VOTOS	FULL SCAN (MIN/MAX)	1	2
INDEX	PK_ZFREGUESIAS	UNIQUE SCAN		
Access Predicates F.CODIGO=V.FREGUESIA				
TABLE ACCESS	ZFREGUESIAS	BY INDEX ROWID	1	7
TABLE ACCESS	ZFREGUESIAS	FULL	4241	7

Figura 8: Plano de execução da query 2c no contexto Z

Contexto	Custo
X	72
Y	68
Z	17

Tabela 7: Custos da query 2c

4.2.4 Query 2d - Para cada distrito indique qual o seu nome e a designação e número de votos do partido que nele teve melhor votação.

NOME	DESIGNACAO	VOTOS_MAX
1 Aveiro	Partido Socialista	145575
2 Beja	Partido Socialista	39728
3 Braga	Partido Socialista	195602
4 Bragança	Partido Social Democrata	36841
5 Castelo Branco	Partido Socialista	63398
6 Coimbra	Partido Socialista	109956
7 Évora	Partido Socialista	42257
8 Faro	Partido Socialista	87162
9 Guarda	Partido Socialista	44254
10 Leiria	Partido Social Democrata	99091
11 Lisboa	Partido Socialista	480410
12 Portalegre	Partido Socialista	36545
13 Porto	Partido Socialista	440162
14 Santarém	Partido Socialista	110326
15 Setúbal	Partido Socialista	170193
16 Viana do Castelo	Partido Socialista	55132
17 Vila Real	Partido Social Democrata	56507
18 Viseu	Partido Social Democrata	90116
19 Madeira	Partido Social Democrata	56302
20 Açores	Partido Socialista	49947

Figura 9: Resultados da query 2d

```

SELECT d.nome, p.designacao, m.votos_max
FROM xpartidos p
  INNER JOIN
    (SELECT c.distrito, v.partido, SUM(v.votos) AS total_votos
     FROM xfreguesias f, xconcelhos c, xvotacoes v
     WHERE f.concelho=c.codigo AND f.codigo=v.freguesia
     GROUP BY c.distrito, v.partido) s
  ON s.partido=p.sigla

```



```

INNER JOIN
  (SELECT distrito, MAX(total_votos) AS votos_max
   FROM (SELECT c.distrito, v.partido, SUM(v.votos) AS total_votos
         FROM xfreguesias f, xconcelhos c, xvotacoes v
         WHERE f.concelho=c.codigo AND f.codigo=v.freguesia
         GROUP BY c.distrito, v.partido)
   GROUP BY distrito) m
ON m.distrito=s.distrito AND m.votos_max=s.total_votos
INNER JOIN xdistritos d ON m.distrito=d.codigo;

```

Plano de Execução: Esta query determina o partido com o maior número total de votos em cada distrito, retornando o nome do distrito, a designação do partido vencedor e o total de votos. Usa duas subqueries: uma calcula os votos totais por partido e distrito, juntando três tabelas (*xfreguesias*, *xconcelhos* e *xvotacoes*); a outra usa a primeira novamente e encontra o valor máximo de votos por distrito. A query final junta essas subconsultas com *xpartidos* e *xdistritos* para obter a designação do partido e o nome do distrito.

- **Contexto X:** Varredura completa e junções custosas nas subconsultas, custando um total de 95.
- **Contexto Y:** Similar a X, mas as chaves primárias permitem usufruir de *nested loops* e evitar varrimentos completos em alguns casos, reduzindo o custo para 89.
- **Contexto Z:** Similar a Y, mas índices em chaves estrangeiras permitem acelerar junções e agregações nas subqueries.

Contexto	Custo
X	95
Y	89
Z	87

Tabela 8: Custos da query 2d

Conclusões: O uso de índices das chaves primárias e índices em chaves estrangeiras como *zvotacoes.partido*, *zvotacoes.freguesia* e *zfreguesias.concelho* no contexto Z otimiza as subqueries, reduzindo o custo total.

Ainda assim, é notória uma grande ineficiência na interrogação visto que, a mesma subconsulta está a ser escrita em duas ocasiões na query. Visto isto, fazer uso de estruturas como vistas materializadas seria bastante mais recomendável (o caso descrito abaixo, sem índices, obteria um custo de 11):

```

CREATE MATERIALIZED VIEW Sum_votos AS
  (SELECT c.distrito, v.partido, SUM(v.votos) AS total_votos
   FROM zfreguesias f, zconcelhos c, zvotacoes v
   WHERE f.concelho=c.codigo AND f.codigo=v.freguesia
   GROUP BY c.distrito, v.partido
  );
CREATE MATERIALIZED VIEW Max_votos AS
  (SELECT distrito, MAX(total_votos) AS votos_max
   FROM Sum_votos
   GROUP BY distrito
  );

SELECT d.nome, p.designacao, m.votos_max
FROM Max_votos m, Sum_votos s, zpartidos p, zdistritos d

```

```
WHERE m.districto=s.districto AND m.votos_max=s.total_votos AND s.partido
=p.sigla AND m.districto=d.codigo;
```

4.3 Negação

4.3.1 Query 3 - Quais os partidos que não concorreram no distrito de Lisboa.

	SIGLA	DESIGNACAO
1	PDA	Partido Democrático do Atlântico

Figura 10: Resultados da query 3

```
SELECT sigla, designacao
FROM xpartidos
WHERE sigla NOT IN (
  SELECT l.partido
  FROM xlistas l, xdistritos d
  WHERE l.districto = d.codigo
    AND d.nome = 'Lisboa'
);
```

Plano de Execução: Esta query identifica os partidos que não apresentaram listas no distrito de Lisboa, retornando as suas siglas e designações. Utiliza a tabela `xpartidos` e uma subconsulta com `NOT IN` que junta `xlistas` (listas por distrito e partido) e `xdistritos` (nomes dos distritos), filtrando para `d.nome = 'Lisboa'`. A condição `NOT IN` exclui os partidos que aparecem na subconsulta, ou seja, aqueles que concorreram em Lisboa.

- **Contexto X:** Sub-interrogação executada com varredura completa, custo 9.
- **Contexto Y:** Chaves primárias permitem junções mais eficientes, fazendo uso da ordenação para realizar *merge join*, reduzindo o custo para apenas 7.
- **Contexto Z:** Índice Bitmap em `zdistritos.nome` evita o varrimento completo da tabela `zdistritos`, otimizando a subconsulta e reduzindo o custo para 6.

Contexto	Custo
X	9
Y	7
Z	6

Tabela 9: Custos da query 3

Conclusões: Os índices das chaves primárias facilitam o processo de junção de tabelas enquanto que o índice Bitmap na coluna `zdistritos.nome` em Z melhora a performance da subconsulta `NOT IN` ao otimizar a filtragem em `d.nome = 'Lisboa'`.

4.4 Comparação de Estratégias

Query 4 - Houve algum partido a vencer em todos as freguesias de um concelho do distrito do Porto? Indique código do concelho e sigla do partido.

	CODIGO	PARTIDO
1	1304	PS
2	1308	PS

Figura 11: Resultados da query 4

4.4.1 a - Sem Vista (Contagem)

```
SELECT c.codigo, v.partido
FROM zconcelhos c
INNER JOIN zdistritos d ON c.districto=d.codigo
INNER JOIN zfreguesias f ON c.codigo=f.concelho
INNER JOIN zvotacoes v ON f.codigo=v.freguesia
WHERE d.nome='Porto' AND v.votos=(SELECT MAX(votos) FROM zvotacoes
    WHERE freguesia=v.freguesia)
GROUP BY c.codigo, v.partido
HAVING COUNT(*) = (
    SELECT COUNT(*)
    FROM zfreguesias f2
    WHERE f2.concelho = c.codigo
);
```

Plano de Execução: Esta query identifica os partidos que venceram em todas as freguesias de cada concelho do distrito do Porto. Junta as tabelas `zconcelhos`, `zdistritos`, `zfreguesias` e `zvotacoes` para associar concelhos ao distrito, freguesias ao concelho e votos às freguesias. A condição `v.votos = (SELECT MAX(...))` seleciona o partido mais votado em cada freguesia. A cláusula `HAVING COUNT(*) = (...)` compara o número de freguesias onde um partido venceu com o total de freguesias do concelho, garantindo que o partido venceu em todas as freguesias desse concelho. A filtragem inicial por `d.nome = 'Porto'` restringe a análise apenas ao distrito do Porto.

A existência de índice Bitmap em `zdistritos.nome` e índice B-tree em `zvotacoes.votos` otimizam a filtragem nas condições `WHERE` enquanto que os índices em chaves primárias e estrangeiras como `zconcelhos.districto`, `zfreguesias.concelho` e `zvotacoes.freguesia` facilitam o processo de junção permitindo maior eficiência com *nested loops*. O custo total é 73.

4.4.2 a - Sem Vista (Dupla Negação)

```
SELECT c.codigo, v.partido
FROM zconcelhos c
INNER JOIN zfreguesias f ON f.concelho=c.codigo
INNER JOIN zdistritos d ON c.districto=d.codigo
INNER JOIN zvotacoes v ON v.freguesia=f.codigo
WHERE d.nome='Porto'
AND NOT EXISTS (
    SELECT f2.concelho
    FROM zfreguesias f2
    WHERE f2.concelho = c.codigo
    AND NOT EXISTS (
        SELECT v2.freguesia, v2.partido
        FROM zvotacoes v2
        WHERE v2.freguesia = f2.codigo
            AND v2.partido = v.partido
            AND v2.votos >= ALL (
                SELECT votos
```

```

        FROM zvotacoes
        WHERE freguesia = f2.codigo
    )
)
)
GROUP BY c.codigo, v.partido;

```

Plano de Execução: Esta query tem o mesmo objetivo que a anterior, fazendo as mesmas junções entre tabelas e usando a mesma restrição de distrito, apenas muda a forma de verificar a condição principal - se o partido venceu em todas as freguesias de um concelho - usando a estratégia de Dupla Negação. Para isso, faz uso de duas operações NOT EXISTS() aninhadas em que:

- A condição interna NOT EXISTS (...) verifica, para cada freguesia f2 do c.concelho, se existe pelo menos uma votação onde o partido em análise (v.partido) tenha sido o mais votado (v2.votos >= ALL (...)).
- A condição NOT EXISTS (...) externa verifica que não existe nenhuma freguesia no concelho atual (c.codigo) que viole a condição interna.

Se essa verificação falhar para alguma freguesia (ou seja, se o partido não vencer em alguma), a subquery interna será verdadeira e o NOT EXISTS externo irá falhar, excluindo o partido da lista de vencedores do concelho. A cláusula GROUP BY agrupa os pares (concelho, partido) onde o partido venceu em todas as freguesias do concelho, de forma a não retornar resultados repetidos.

A query é custosa (5027) devido às subqueries aninhadas com NOT EXISTS, que forçam muitas comparações. No entanto, o índice composto em zvotacoes(freguesia, partido, votos) ajudou a reduzir o custo, permitindo acesso rápido aos votos por freguesia e partido e evitando varreduras completas repetidas.

4.4.3 b - Com Vista (Contagem)

```

CREATE VIEW count_view_vencedores_freguesia AS
SELECT f.concelho, v1.partido
FROM zfreguesias f
INNER JOIN
    (SELECT v2.partido, v2.freguesia, v2.votos
     FROM zvotacoes v2
     WHERE v2.votos=(SELECT MAX(votos) FROM zvotacoes WHERE freguesia=v2
                     .freguesia)) v1
ON f.codigo=v1.freguesia;

SELECT c.codigo, v.partido
FROM zconcelhos c
INNER JOIN zdistritos d ON c.distrito=d.codigo
INNER JOIN count_view_vencedores_freguesia v ON c.codigo=v.concelho
WHERE d.nome='Porto'
GROUP BY c.codigo, v.partido
HAVING COUNT(*) = (
    SELECT COUNT(*)
    FROM zfreguesias f2
    WHERE f2.concelho = c.codigo
);

```

Plano de Execução: Esta query é equivalente à interrogação com estratégia de Contagem da situação a, tendo como única alteração a introdução de uma vista com a condição que verifica que um partido foi o mais votado numa freguesia (VIEW count_view_vencedores_freguesia).

Esta modificação não altera o desempenho da query, apenas melhora a organização e compreensibilidade do código, visto que, a vista é uma estrutura que não armazena os resultados de uma query mas sim a query em si, logo, a cada vez que é chamada, a interrogação da vista é executada e portanto não tem influência no desempenho (o custo mantém-se a 73).

4.4.4 b - Com Vista (Dupla Negação)

```
CREATE VIEW neg_view_vencedores_freguesia AS
SELECT f.codigo AS freguesia, f.concelho, v.partido
FROM zfreguesias f
INNER JOIN zvotacoes v
ON f.codigo = v.freguesia
WHERE v.votos >= ALL (
    SELECT v2.votos
    FROM zvotacoes v2
    WHERE v2.freguesia = v.freguesia
);

SELECT c.codigo, v.partido
FROM zconcelhos c
INNER JOIN zfreguesias f ON f.concelho=c.codigo
INNER JOIN zdistritos d ON c.districto=d.codigo
INNER JOIN zvotacoes v ON v.freguesia=f.codigo
WHERE d.nome='Porto'
AND NOT EXISTS (
    SELECT f2.concelho
    FROM zfreguesias f2
    WHERE f2.concelho = c.codigo
    AND NOT EXISTS (
        SELECT v2.freguesia
        FROM neg_view_vencedores_freguesia v2
        WHERE v2.freguesia = f2.codigo
        AND v2.partido = v.partido
    )
)
GROUP BY c.codigo, v.partido;
```

Plano de Execução: Esta query segue o mesmo processo da query anterior, consistindo na base da interrogação com estratégia de Dupla Negação da situação a, mas fazendo uso de uma vista (VIEW neg_view_vencedores_freguesia) para melhorar a estruturação do código. Tal como explicado no caso anterior, a vista não melhora o desempenho da interrogação, mantendo assim o custo em 5027.

4.4.5 c - Com Vista Materializada (Contagem)

```
CREATE MATERIALIZED VIEW mat_count_view_vencedores_freguesia AS
SELECT f.concelho, v1.partido
FROM zfreguesias f
INNER JOIN
    (SELECT v2.partido, v2.freguesia, v2.votos
     FROM zvotacoes v2
     WHERE v2.votos=(SELECT MAX(votos) FROM zvotacoes WHERE freguesia=v2
        .freguesia)) v1
ON f.codigo=v1.freguesia;

SELECT c.codigo, v.partido
```

```

FROM zconcelhos c
INNER JOIN zdistritos d ON c.distrito=d.codigo
INNER JOIN mat_count_view_vencedores_freguesia v ON c.codigo=v.concelho
WHERE d.nome='Porto'
GROUP BY c.codigo, v.partido
HAVING COUNT(*) = (
    SELECT COUNT(*)
    FROM zfreguesias f2
    WHERE f2.concelho = c.codigo
);

```

Plano de Execução: Nesta query altera-se a vista da situação **b** para vista materializada (`mat_count_view_vencedores_freguesia`). Ao contrário da vista (`VIEW`), a vista materializada (`MATERIALIZED VIEW`) guarda os resultados da query (como se fosse uma tabela) e não a query em si, o que permite deduzir o seu custo do custo total da query (visto que o seu custo já foi "pago" aquando da criação da vista), bastando apenas contabilizar o custo da leitura e da junção. Assim, o custo total da query pode ser reduzido significativamente de 73 para 9 com o uso de uma vista materializada.

4.4.6 c - Com Vista Materializada (Dupla Negação)

```

CREATE MATERIALIZED VIEW mat_neg_view_vencedores_freguesia AS
SELECT f.codigo AS freguesia, f.concelho, v.partido
FROM zfreguesias f
INNER JOIN zvotacoes v
ON f.codigo = v.freguesia
WHERE v.votos >= ALL (
    SELECT v2.votos
    FROM zvotacoes v2
    WHERE v2.freguesia = v.freguesia
);

CREATE INDEX idx_mat_neg_view_freguesia_partido ON
    mat_neg_view_vencedores_freguesia(freguesia, partido);

SELECT c.codigo, v.partido
FROM zconcelhos c
INNER JOIN zfreguesias f ON f.concelho=c.codigo
INNER JOIN zdistritos d ON c.distrito=d.codigo
INNER JOIN zvotacoes v ON v.freguesia=f.codigo
WHERE d.nome='Porto'
AND NOT EXISTS (
    SELECT f2.concelho
    FROM zfreguesias f2
    WHERE f2.concelho = c.codigo
    AND NOT EXISTS (
        SELECT v2.freguesia
        FROM mat_neg_view_vencedores_freguesia v2
        WHERE v2.freguesia = f2.codigo
        AND v2.partido = v.partido
    )
)
GROUP BY c.codigo, v.partido;

```

Plano de Execução: Tal como no caso anterior, nesta query transforma-se a vista (`VIEW`) da situação **b** em vista materializada (`MATERIALIZED VIEW`). Mantendo a semelhança com o funcionamento de uma tabela, podemos também adicionar índices a uma vista materializada.

Fazendo uso desta propriedade, `mat_neg_view_vencedores_freguesia(freguesia, partido)` foi usado como índice composto, permitindo evitar repetições desnecessárias da busca pelos vencedores de cada freguesia e facilitando a sua junção com a query principal. O desempenho da interrogação sofre então uma melhoria substancial com o custo a diminuir de 5027 para 3033.

Situação	Estratégia	Custo
a	Contagem	73
a	Dupla Negação	5027
b	Contagem	73
b	Dupla Negação	5027
c	Contagem	9
c	Dupla Negação	3033

Tabela 10: Custos da query 4

Conclusões: Neste contexto, a estratégia de **Contagem** é substancialmente mais eficiente que a estratégia de **Dupla Negação**. Isto acontece porque a **Contagem** depende de junções simples e filtros diretos, enquanto a **Dupla Negação** implica subqueries aninhadas com ciclos de **NOT EXISTS**, que são mais custosos em termos de leitura e comparação de dados devido à grande quantidade de repetição destas operações.

Além disso, também é possível verificar o efeito das vistas materializadas na otimização de queries, com ambas as estratégias a beneficiar significativamente do seu uso. Assim, a situação c verificou os melhores resultados e juntamente com a estratégia de **Contagem** é a resolução ideal da interrogação.

4.5 Comparação de Tipos de Índice

Query 5 - Quantos votos tiveram o PS e o PSD nos distritos 11, 15 e 17?

	NOME	PARTIDO	NUM_VOTOS
1	Lisboa	PS	480410
2	Vila Real	PPDPSD	56507
3	Setúbal	PPDPSD	70340
4	Lisboa	PPDPSD	307961
5	Vila Real	PS	50691
6	Setúbal	PS	170193

Figura 12: Resultados da query 5

```
SELECT d.nome, v.partido, SUM(v.votos) AS num_votos
FROM zfreguesias f, zconcelhos c, zdistritos d, zvotacoes v
WHERE f.concelho=c.codigo AND c.distrito=d.codigo AND f.codigo=v.
    freguesia
    AND (v.partido='PS' OR v.partido='PPDPSD')
    AND (c.distrito=11 OR c.distrito=15 OR c.distrito=17)
GROUP BY d.nome, v.partido;
```

Plano de Execução: Esta query calcula o número total de votos obtidos pelos partidos PS e PPDPSD nos distritos com código 11, 15 ou 17, agrupando os resultados por distrito e partido. Junta as tabelas `zconcelhos`, `zdistritos`, `zfreguesias` e `zvotacoes` para associar concelhos ao distrito, freguesias ao concelho e votos às freguesias, com condições de filtro nos campos `zvotacoes.partido` e `zconcelhos.distrito`.

4.5.1 Com índices B-tree em `zconcelhos.districto` e `zvotacoes.partido`

Quando aplicados índices B-tree nas colunas `zconcelhos.districto` e `zvotacoes.partido`, o custo da query é 42. O índice B-tree permitiu uma busca eficiente, evitando varrimentos completos nas tabelas `zconcelhos` e `zvotacoes`, o que melhora o desempenho da interrogação.

4.5.2 Com índices Bitmap em `zconcelhos.districto` e `zvotacoes.partido`

Apesar de índices Bitmap serem mais eficientes para condições com baixa cardinalidade como filtragem de poucos distritos e partidos, neste caso não se verificou melhoria no desempenho, com o custo a aumentar para 43. Isto porque ocorre um varrimento completo na tabela `zconcelhos` aquando da filtragem dos distritos, contrariamente ao que acontece com o índice B-tree, em que existe um *range scan*.

Tipo de Índice	Custo
B-tree	42
Bitmap	43

Tabela 11: Custos da query 5

Conclusões: Ainda que índices Bitmap sejam índices que funcionam particularmente bem em colunas com cardinalidade baixa como é o caso das colunas `zconcelhos.districto` (20 distritos diferentes) e `zvotacoes.partido` (12 partidos diferentes), neste contexto os índices B-tree superam Bitmap em termos de custo da interrogação. A diferença ocorre mais especificamente no índice em `zconcelhos.districto`, em que a natureza sequencial dos valores da coluna conferem uma vantagem para índices B-tree. Isto acontece porque os índices B-tree são muito eficientes para valores ordenáveis e consultas com "saltos" definidos, ou seja, quando a coluna tem valores numéricos bem distribuídos e sequenciais, como em `zconcelhos.districto` (1, 2, 3, ..., 20), o índice B-tree organiza essas entradas em ordem crescente, o que permite realizar um *range scan* e assim diminuir o custo. Por outro lado, índices Bitmap criam um bitmap por valor distinto, ou seja, para distritos 11, 15 e 17, seria necessário criar 3 bitmaps separados (um para cada distrito) para depois combiná-los e verificar quais linhas satisfazem a condição, não evitando um varrimento completo à tabela `zconcelhos`. Daí, para este caso específico, ser mais eficiente o uso de índices B-tree.

5 Conclusões

O trabalho desenvolvido permitiu compreender e avaliar o impacto de diferentes estratégias de otimização no desempenho de queries em SQL. Através da análise detalhada de planos de execução, verificou-se que a escolha entre índices B-tree e Bitmap, bem como o uso de vistas materializadas, pode ter um efeito significativo no custo das queries. Ficou evidente que não existe uma solução única ideal - a eficiência depende sempre do tipo de dados, da cardinalidade das colunas envolvidas e da natureza da interrogação. Assim, é fundamental conhecer bem a estrutura dos dados e o comportamento das consultas para tomar boas decisões na modelação e otimização de bases de dados relacionais.