Recursos Educacionais / Educational Resources

Armadilhas em consultas SQL: em Bases de Dados Relacionais

Luís Cavique
DCeT, Univ. Aberta
Luis.Cavique@uab.pt

Lisboa, março 2021

Resumo

Este documento pretende complementar a bibliografia da UC de Sistemas de Gestão de Base de Dados oferecida no 1º semestre do 3º ano, na Licenciatura em Engenharia Informática.

Apesar da simplicidade do SQL, os estudantes estar cientes de algumas particularidades nas consultas do SQL.

Neste trabalho pretende-se detalhar três armadilhas nas consultas SQL em bases de dados relacionais:
(A) consultas em caminhos múltiplos,
(B) armadilha do leque ou "fan trap",
(C) armadilha do abismo ou "chasm trap".

O conhecimento destas armadilhas é importante no processo de desnormalização das bases de dados relacionais, quando da construção e manutenção dos "data warehouses", bem como no processo de ETL ("extract, transform, load").

São incluídos exemplos para cada uma das três referidas armadilhas.

A. Caminhos Múltiplos

Consideremos o esquema de base de dados da Figura 1, onde existe mais de um caminho entre as tabelas scientificAreas e evaluations:
- caminho 1: scientificAreas -> projects -> evaluations,
- caminho 2: scientificAreas -> referees -> evaluations.

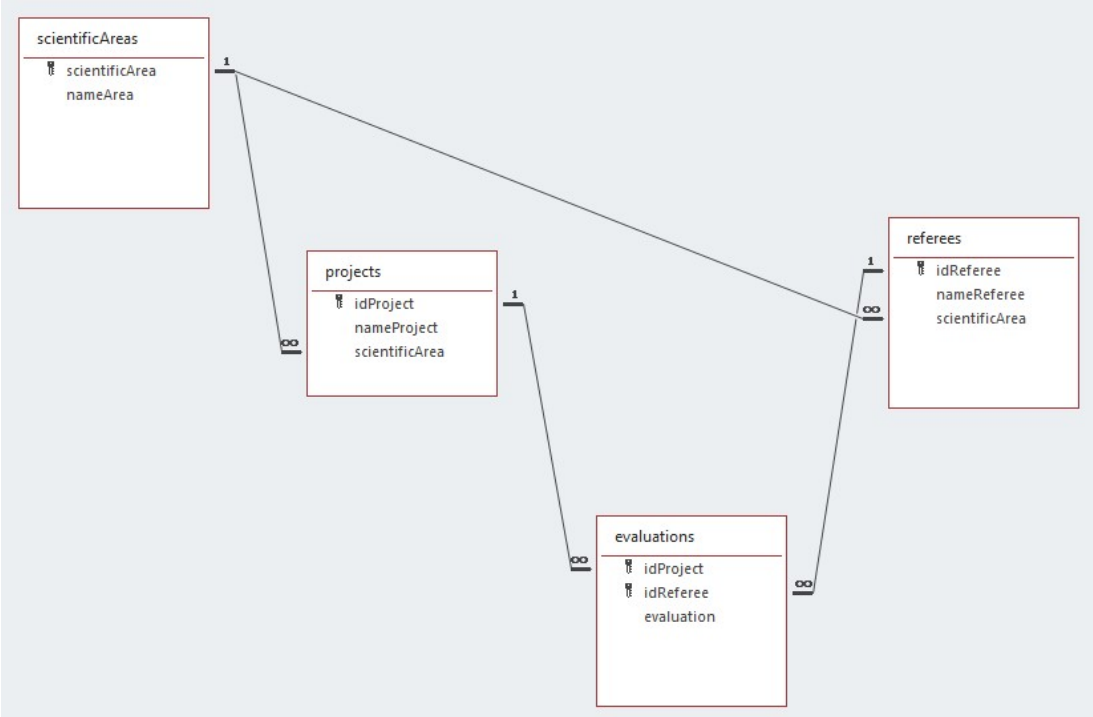


Fig.1 – Esquema da base de dados com caminhos múltiplos

Consideremos ainda os dados de exemplo da Figura 2 da base de dados com dois caminhos.

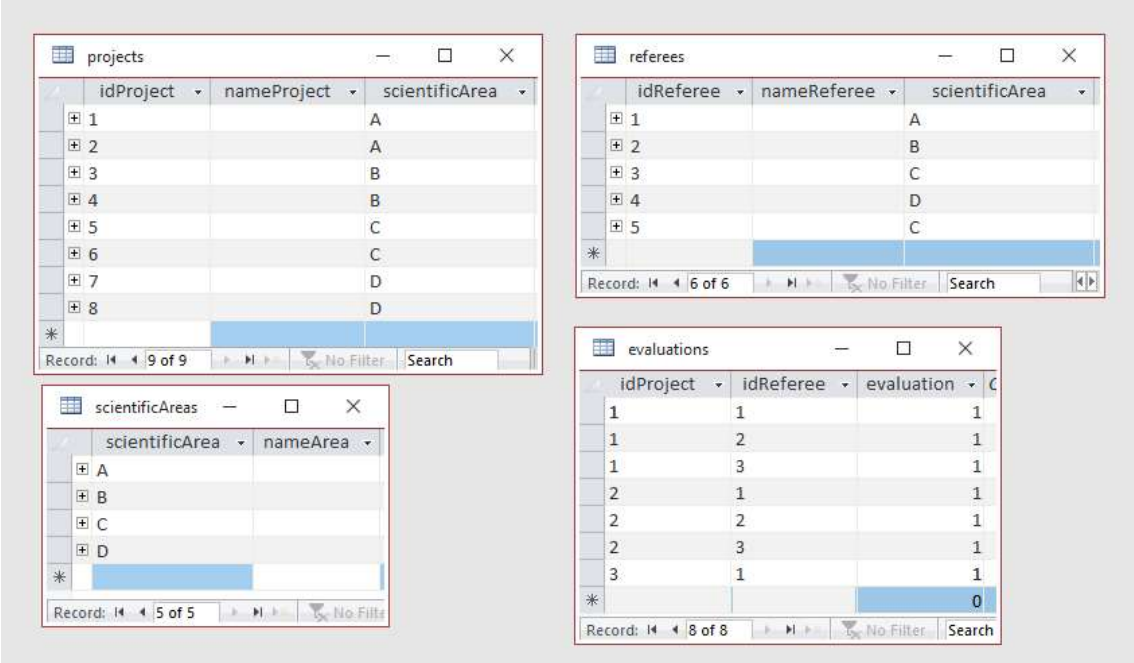


Fig.2 – Dados para o exemplo dos caminhos múltiplos

Pretende-se saber quantas avaliações foram realizadas por área científica. Sendo assim, podemos criar duas consultas diferentes, utilizando os caminhos já definidos. Cada consulta envolve 3 tabelas usando 2 junções. As duas consultas terão as seguintes junções de tabelas:
Consulta 1: scientificAreas |><| projects |><| evaluations
Consulta 2: scientificAreas |><| referees |><| evaluations

Em SQL teremos:

```
-- Consulta 1
SELECT S.scientificArea, Count(E.evaluation) AS SubTotal
FROM scientificAreas  S, projects  P, evaluations  E
WHERE P.idProject=E.idProject
AND  S.scientificArea=P.scientificArea
GROUP BY S.scientificArea;
--
-- Consulta 2
SELECT S.scientificArea, Count(E.evaluation) AS SubTotal
FROM scientificAreas S, referees  R, evaluations  E
WHERE R.idReferee=E.idReferee
AND  S.scientificArea=R.scientificArea
GROUP BY S.scientificArea;
```

A Figura 3 apresenta os resultados das duas consultas, onde se constata que na base de dados com caminhos múltiplos, para caminhos diferentes é possível encontrar resultados também diferentes.

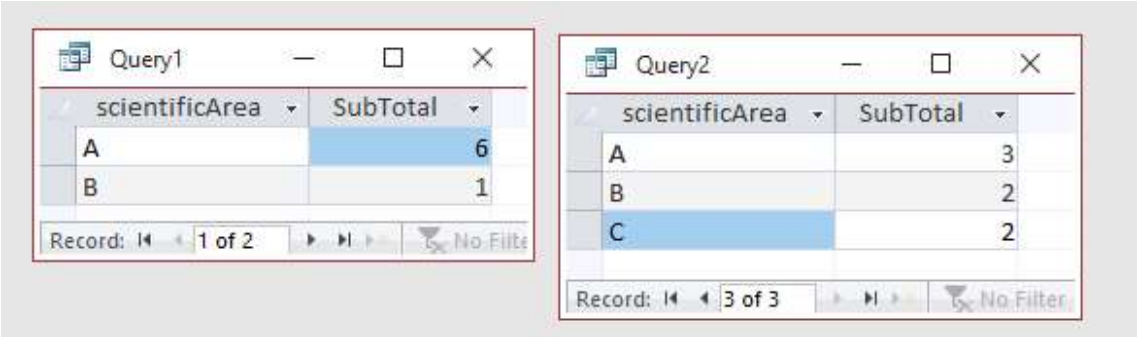


Fig. 3 - Os caminhos múltiplos apresentam resultados diferentes nas consultas

Esse problema não ocorre em bases de dados onde há um único caminho entre duas tabelas. Quando dois ou mais caminhos ocorrem entre duas tabelas, as consultas correspondentes podem retornar soluções diferentes.

É de referir ainda que podem existir situações onde existindo dois caminhos entre duas tabelas elas têm significados claros e distintos. Quando para dois ou mais caminhos a semântica é clara e diferente, não existe qualquer problema em adotar os vários caminhos.

Na Figura 4 é apresentado um esquema de base de dados com caminhos múltiplos entre as tabelas Balcão e Cliente. Contudo, neste esquema existem semânticas (ou significados) diferentes para cada um dos caminhos. Um dos caminhos refere-se aos depósitos dos clientes e o outro aos empréstimos.

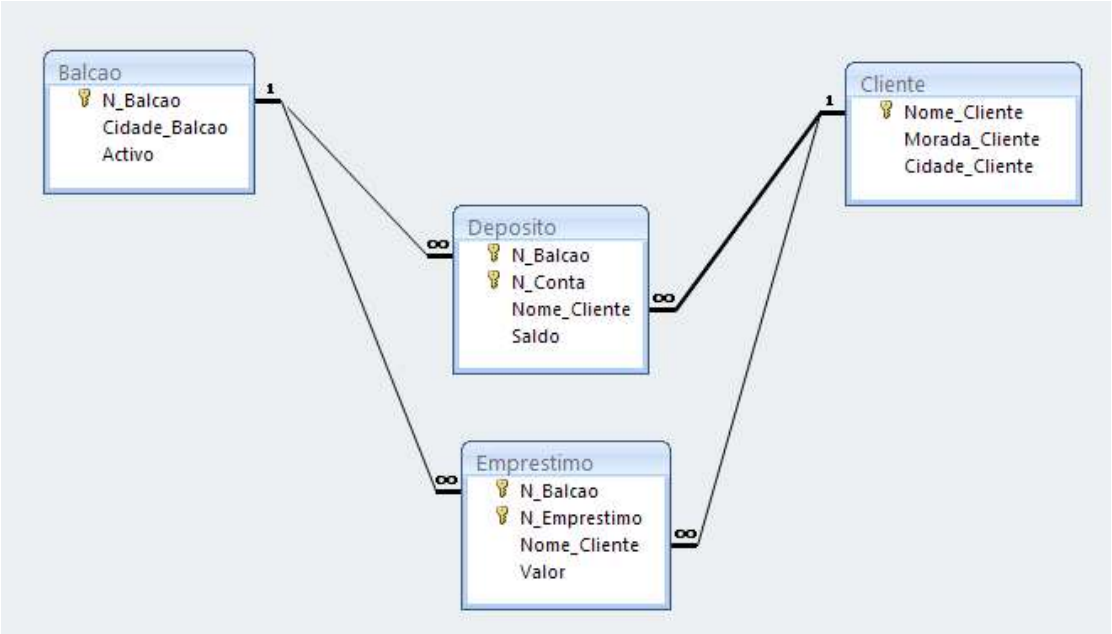


Fig. 4 - Os caminhos múltiplos entre Balcão e Cliente com semânticas distintas

Podemos concluir que se não existirem semânticas claras entre dois ou mais caminhos de um esquema de base de dados, podemos estar a gerar resultados ambíguos.

Voltando ao exemplo inicial, podemos ainda clarificar o significado das consultas e remover a ambiguidade gerada. Em vez de "pretender saber quantas avaliações foram realizadas por área científica", vamos detalhar cada consulta em linguagem natural da seguinte forma:

Consulta 1: "quantas avaliações de projetos existem em cada área científica?"

Π (scientificArea, count(evaluation))
(scientificAreas |><| projects |><| evaluations)

Consulta 2: "quantas avaliações foram realizadas pelos revisores em cada área científica?"

Π (scientificArea, count(evaluation))
(scientificAreas |><| referees |><| evaluations)

B. Armadilha do leque ('Fan trap')

Consideremos o esquema de base de dados da Figura 5, com três tabelas: clientes, faturas e linhas-da-fatura.

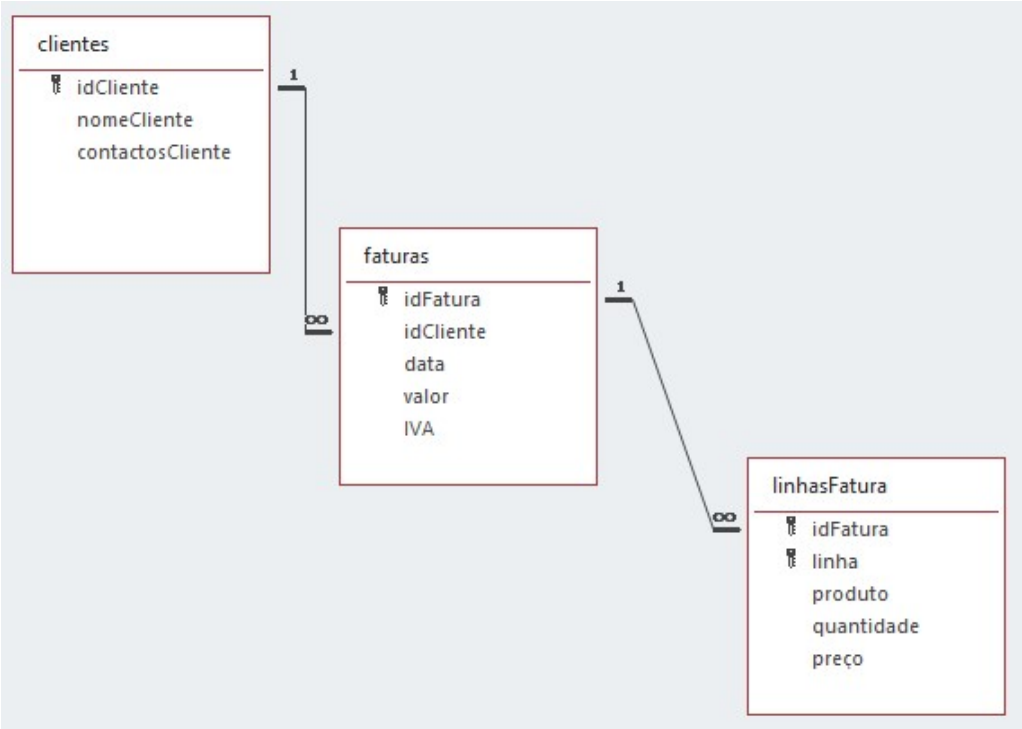


Fig. 5 – Esquema da base de dados

Consideremos ainda os dados de exemplo da Figura 6 da base de dados com três tabelas.

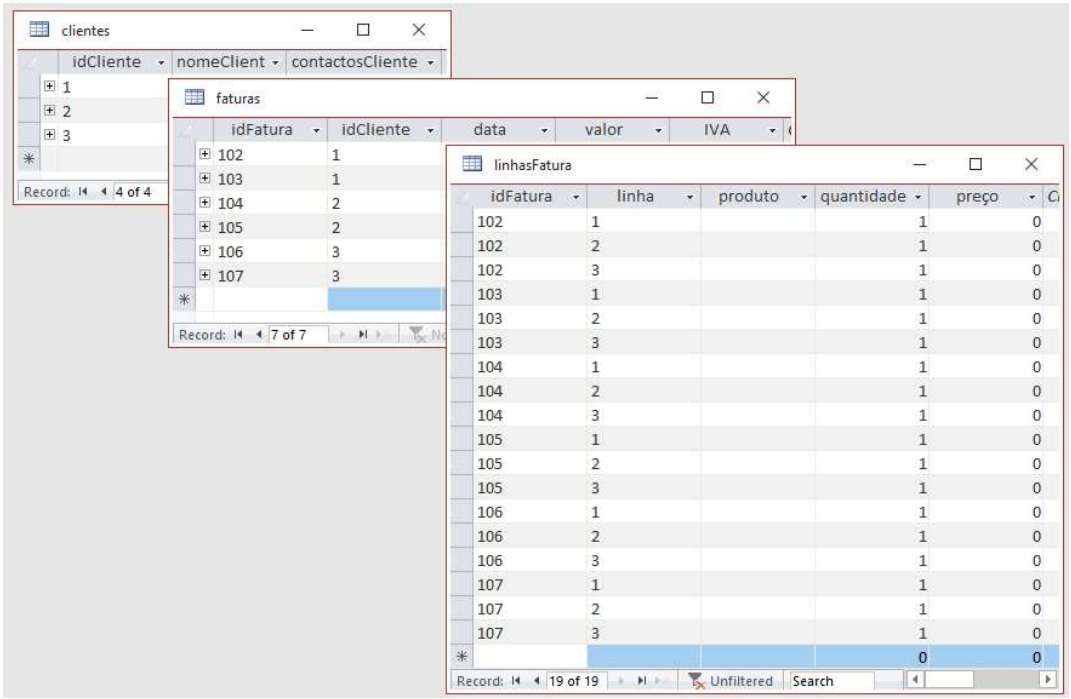


Fig. 6 - Dados para o exemplo

De seguida, vamos criar duas consultas diferentes com o objetivo de saber qual a soma do valor das faturas por cliente. A primeira consulta envolve 3 tabelas usando 2 junções. A segunda envolve 3 tabelas. As duas consultas serão:
Consulta 1: clientes |><| faturas |><| linhasFatura
Consulta 2: clientes |><| faturas

A primeira consulta ao incluir a tabela linhasFatura vai multiplicar sem necessidade o número de linhas do conceito de fatura. A este tipo de erro dá-se o nome de armadilha do leque ('fan trap'). A 'fan trap' e a consulta correta sem a utilização da tabela linhasFatura são apresentadas de seguida em SQL.

```
-- 'Fan trap'
SELECT C.idCliente, SUM(F.valor) AS SubTotal
FROM clientes AS C, faturas AS F, linhasFatura AS L
WHERE C.idCliente = F.idCliente
AND F.idFatura = L.idFatura
GROUP BY C.idCliente
--
-- Right query
SELECT C.idCliente, SUM(F.valor) AS SubTotal
FROM clientes AS C, faturas AS F
WHERE C.idCliente = F.idCliente
GROUP BY C.idCliente;
```

Os resultados são apresentados na Figura 7. Visto que cada fatura tem 3 linhas os resultados vêm triplicados.

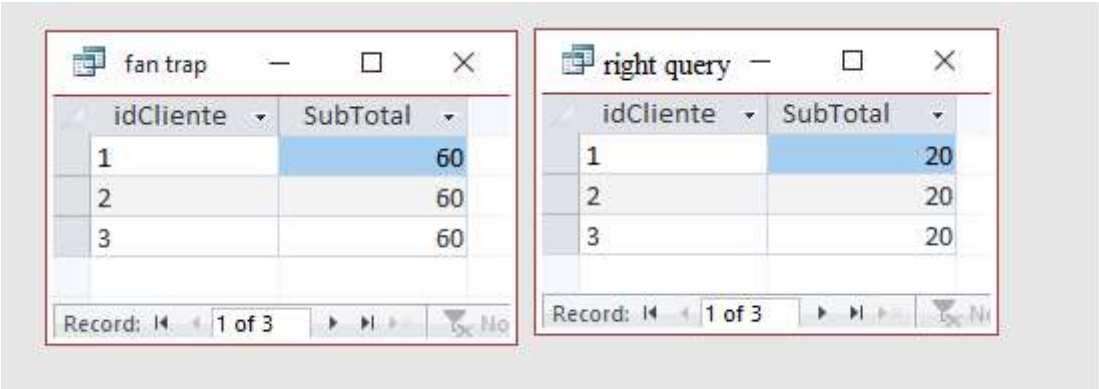


Fig. 7 – O resultado da 'fan trap' triplica o valor do resultado correto

Este tipo de armadilha é simples de evitar, desde que se controlem as tabelas que estão a ser usadas.

C. Armadilha do abismo ('Chasm trap')

Consideremos o esquema de base de dados da Figura 8, com três tabelas: clientes, encomendas e faturas.

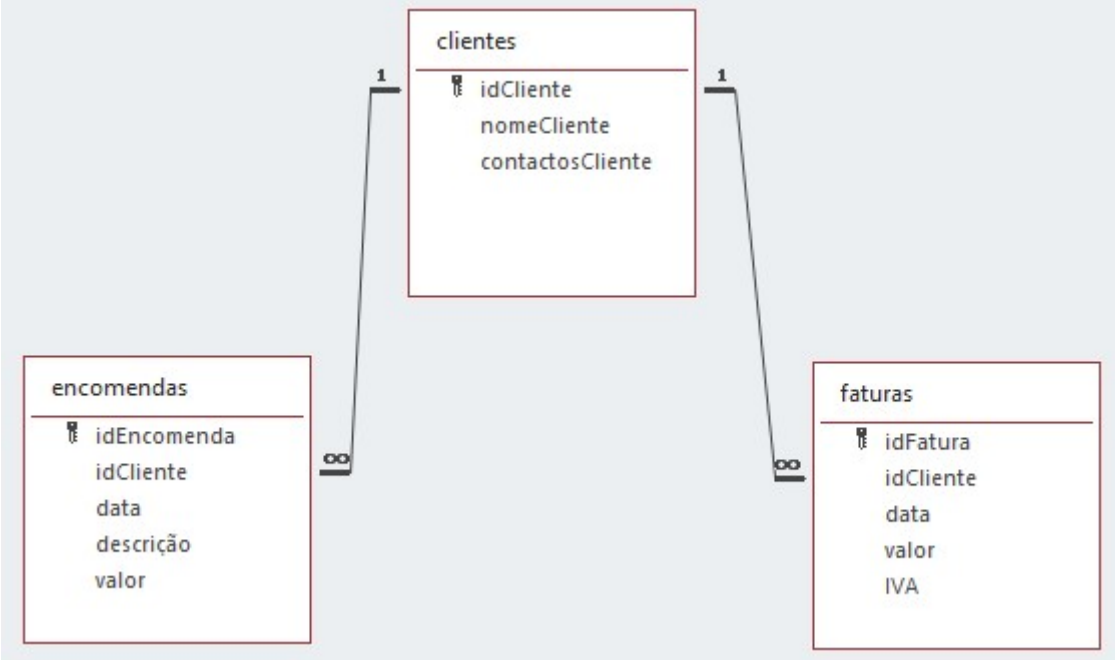


Fig. 8 – Esquema da base de dados

Consideremos ainda os dados de exemplo da Figura 9 da base de dados com três tabelas.

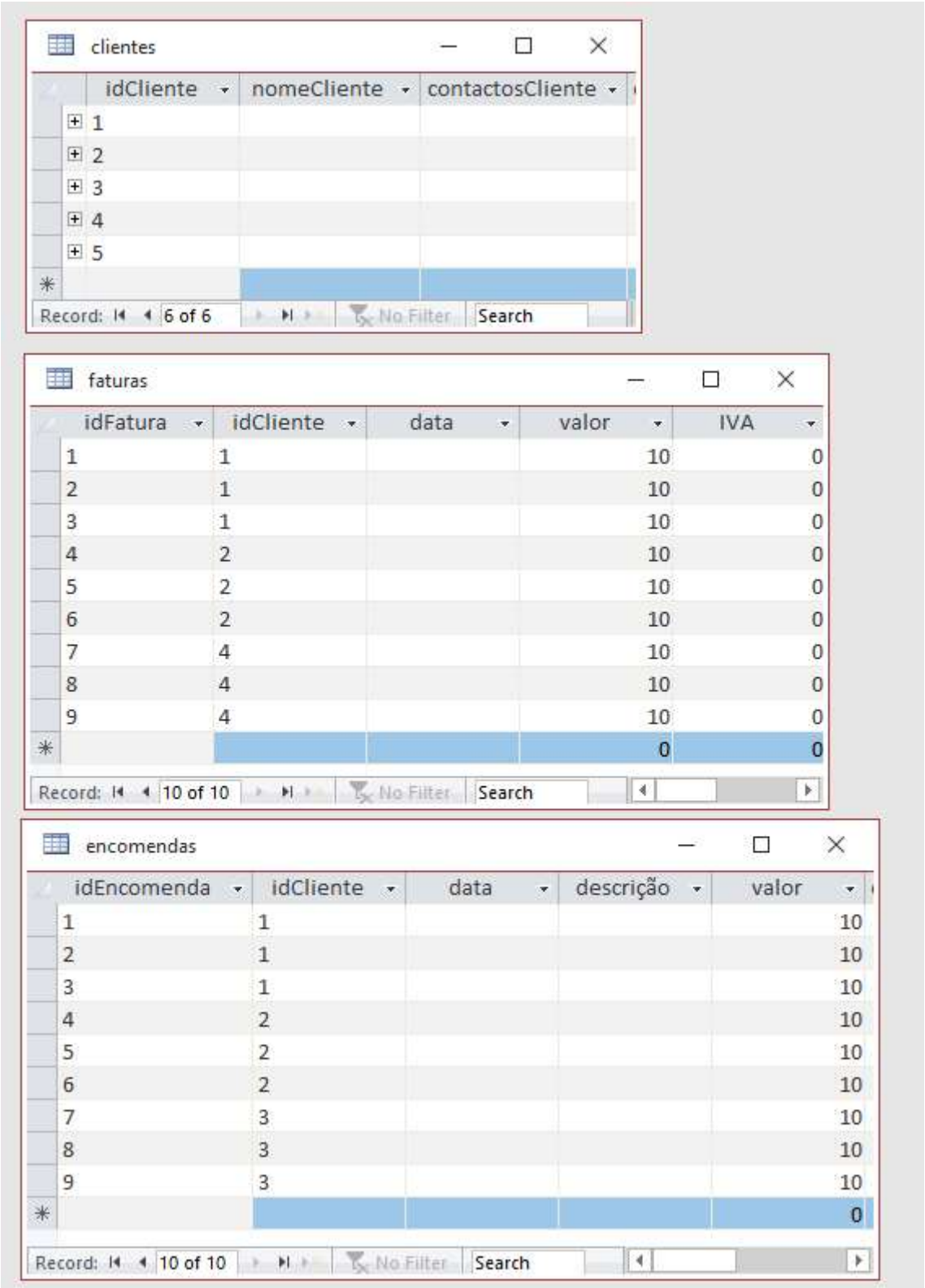
De seguida, vamos criar uma consulta com o objetivo de juntar a informação do cliente relativamente às encomendas e faturas.

A dita consulta, denominada Consulta 1, envolve 3 tabelas (clientes, encomenda e faturas) usando 2 junções:
Consulta 1: clientes |><| encomendas |><| faturas

O cliente 1, aparentemente com 30 euros de encomendas e 30 euros de faturação aparece com subtotais de 90 euros respetivamente. O mesmo acontece para o cliente 2, como indicado na Figura 10.

A tabela Faturas vai multiplicar o número de linhas com a tabela Encomendas, triplicando o valor. A este tipo de erro dá-se o nome de armadilha do abismo ('chasm trap').

Entre as tabelas Faturas e Encomendas, existe realmente um abismo, no sentido de não se pode simplesmente associá-las com o operador de junção.



clientes

idCliente	nomeCliente	contactosCliente
1		
2		
3		
4		
5		

Record: 6 of 6 No Filter Search

faturas

idFatura	idCliente	data	valor	IVA
1	1		10	0
2	1		10	0
3	1		10	0
4	2		10	0
5	2		10	0
6	2		10	0
7	4		10	0
8	4		10	0
9	4		10	0
			0	0

Record: 10 of 10 No Filter Search

encomendas

idEncomenda	idCliente	data	descrição	valor
1	1			10
2	1			10
3	1			10
4	2			10
5	2			10
6	2			10
7	3			10
8	3			10
9	3			10
				0

Record: 10 of 10 No Filter Search

Fig. 9 - Dados para o exemplo

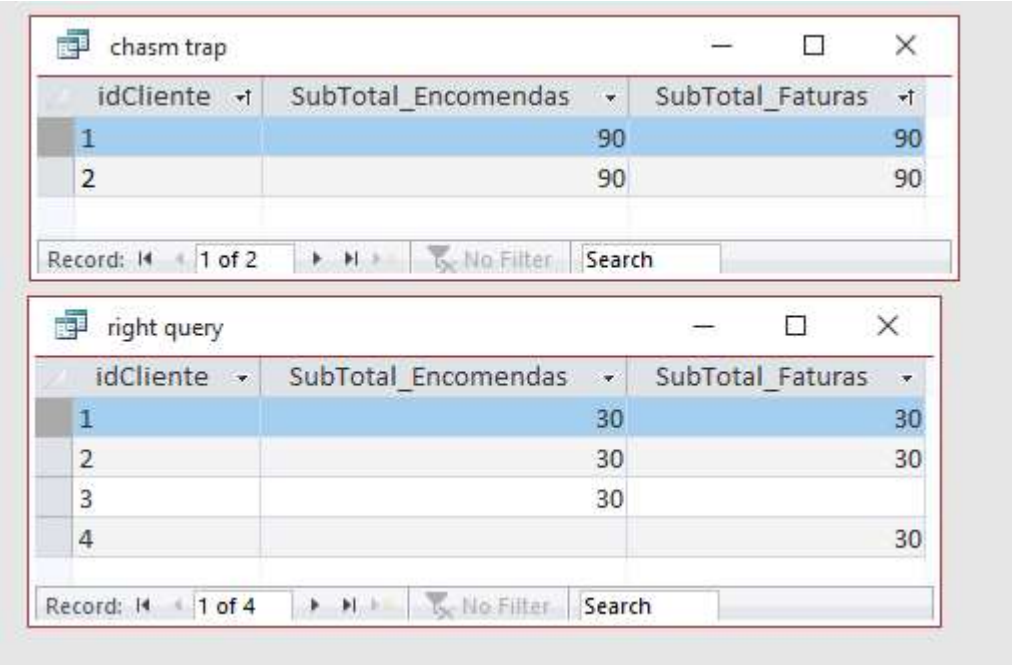


Fig. 10 – Resultado da 'chasm trap' e o resultado correto

A 'chasm trap' é apresentada de seguida em SQL.

```
-- 'Chasm trap'
SELECT C.idCliente, SUM(E.valor) AS SubTotal_Encomendas,
                    SUM(F.valor) AS SubTotal_Faturas
FROM clientes C, encomendas E, faturas F
WHERE C.idCliente = E.idCliente
AND       C.idCliente = F.idCliente
GROUP BY C.idCliente;
```

Se à consulta anterior retirármos a cláusula Group-By e os operadores de soma, para o idCliente=1 obtemos o resultado inflacionado da Figura 11, onde se constata que para cada idEncomenda se juntam três IdFatura.

Fig. 11 – Detalhe da 'chasm trap' para o idCliente = 1

Se pretendemos juntar a informação do cliente relativamente às encomendas e faturas, teremos de usar outro processo.

Em primeiro lugar calculamos os subtotais dos clientes por faturas e encomendas e em segundo lugar usar o 'full join' das tabelas agregadas. No exemplo a seguir o 'full join' é obtido com a união de dois 'left joins'.

```
-- 'Full join' das agreagações valorE e valorF
WITH valorE AS ( SELECT C.idCliente, Sum(E.valor) AS SumOfvalor
                    FROM clientes AS C,  encomendas AS E ON
                    WHERE C.idCliente = E.idCliente
                    GROUP BY C.idCliente)
        valor F AS (SELECT C.idCliente, Sum(F.valor) AS SumOfvalor
                    FROM clientes AS C, faturas AS F ON
                    WHERE C.idCliente = F.idCliente
                    GROUP BY C.idCliente)

(SELECT E.idCliente, E.SumOfvalor AS SubTotal_Encomendas,
                        F.SumOfvalor AS SubTotal_Faturas
FROM  valorE AS E
        LEFT JOIN ValorF AS F
ON  E.idCliente=F.idCliente)

UNION

(SELECT F.idCliente, E.SumOfvalor, F.SumOfvalor
FROM  valorF AS F
        LEFT JOIN ValorE AS E
ON  E.idCliente=F.idCliente);
```

Finalmente, convém reforçar que não é possível associar as tabelas Faturas e Encomendas com um grau de granularidade menor, existindo realmente um abismo entre as tabelas Encomendas e Faturas. A associação das duas tabelas só pode ser realizada com valores agregados.

Das armadilhas das junções em SQL a 'chasm trap' esta é a mais complicada e para a qual devemos estar mais atentos no processo de ETL ("extract, transform, load").

Notas Bibliográficas

Neste trabalho incluímos o 'Multiple Access Path Problem' (MAPP) como a primeira armadilha das consultas em SQL. As 'connection traps' dividem-se em 'fan trap' e 'chasm trap' [Oracle 2021]. A 'fan trap' que têm uma fácil resolução. Por outro lado, a 'chasm trap' é uma armadilha de resolução mais complexa.

O Multiple Access Path Problem (MAPP) é apresentado em [Hall 1986] que procura traduzir frases em linguagem natural em consultas de bases de dados removendo a ambiguidade. Soluções para o MAPP podem ser encontrados em [Wald, Sorenson 1984] e desenvolvimentos em [Wu, Ichikawa, Cercone 1996].

As bases de dados com caminhos múltiplos são também conhecidas por bases de dados cíclicas, tendo sido objeto de vários trabalhos referidos em Ullman (1982) e Maier (1983).

O termo 'connection traps' é referido pelo próprio criador do modelo relacional, E.F. Codd [1970], num trabalho onde clarifica várias noções sobre redundância e consistência de bases de dados.

As 'connection traps' são estudadas por Feng [Feng, Crowe 1999] [Qin, Feng 2020], e referidas como 'falsa inferência' por [Date 1995] ou de forma mais explicita, por 'uma deficiência intrínseca do modelo relacional' segundo [ter-Bekke 1992].

Referências

Codd E.F. (1970), A relational model of data for large shared data banks, Communications of the ACM, vol. 13, n. 6, pp. 377- 387.

Feng J., M. Crowe (1999), The notion of 'classes of a path' in ER schemas, Advances in Databases and Systems Information, Third East European Conference, ADBIS'99, Slovenia, Proceedings, Johann Eder, Ivan Rozman, Tatjana Welzer (Eds.), Lecture Notes in Computer Science, Springer, pp. 218-232.

Hall G.W. (1986), Querying cyclic databases in natural language, Master of Science in the Scholl of Computer Science, Simon Fraser University.

Maier D. (1983) The Theory of Relational Databases, Computer Science Press, Rockville, ISBN-13: 978-091-48-9442-1.

Oracle (2021), Business Intelligence design traps, Everything Oracle, acesso ao site em 09-03-2021, http://everything-oracle.com/obieefanch.htm.

Qin L., J. Feng (2020), The Concept of Representation Capability of Databases and its Application in IS Development, Journal of International Technology and Information Management, vol. 29, n. 2, article 1.

ter-Bekke, J.H. (1992), Semantic Data Modeling, Prentice Hall, New York, ISBN: 0138060509.

Ullman J.D. (1982), Principles of database systems, Second edition, Computer software engineering series, Computer Science Press, Rockville.

Wald J.A., P.G. Sorenson (1984), Resolving the query inference problem using Steiner trees, ACM Transactions on Database Systems (TODS), vol.9 (3), pp. 348-368.

Wu X., T. Ichikawa, N. Cercone (1996), Knowledge-Base Assisted Database Retrieval Systems, World Scientific, ISBN: 978-981-02-1850-8.