CONSULTAS EM BANCO DE DADOS

Matheus da Silva Serpa

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

União de conjuntos em SQL

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Descrever a operação de união em álgebra relacional.
- Comparar as cláusulas `UNION` e `UNION ALL` em SQL.
- Implementar a operação de união em SQL.

Introdução

A álgebra relacional é uma linguagem de consulta procedural que recebe um conjunto de relações como entrada e retorna um novo conjunto como saída. Um conjunto de operadores é definido e aplicado a tais conjuntos de relações com o objetivo de produzir os resultados buscados.

A operação de união compreende uma dessas operações, que, resumidamente, tem o comportamento equivalente ao da operação de união da teoria de conjuntos. Uma das únicas restrições para o uso dessa operação reside no fato de que, para a união de duas relações, ambas devem ter o mesmo conjunto de atributos ou colunas da tabela.

Neste capítulo, você conhecerá mais sobre a operação de união aplicada na álgebra relacional, compreenderá na prática o uso das cláusulas `UNION` e `UNION ALL` na linguagem SQL (Structured Query Language), e, por fim, estudará a implementação da operação de união em PostgreSQL.

1 Operação de união na álgebra relacional

Entre as operações básicas da álgebra relacional, podemos citar as operações sobre conjuntos, como a união, que, representada por U, retorna uma relação de resposta contendo todas as tuplas que ocorrem, por exemplo, na instância de uma relação A ou em uma relação B. Na Figura 1, podemos ver o diagrama de Venn dessa operação (MENEZES, 2013), em que os conjuntos A e B foram

unidos em um novo conjunto resposta: como observamos, as tuplas também podem existir em ambas as instâncias, parte representada pela região mais escura no centro. E vale ressaltar um detalhe importante — na álgebra relacional, os atributos de cada relação devem ser compatíveis, o que, segundo Ramakrishnan e Gehrke (2008, p. 112), acontece se as duas instâncias da relação têm o mesmo número de campos e se os campos da esquerda para a direita apresentam o mesmo domínio.

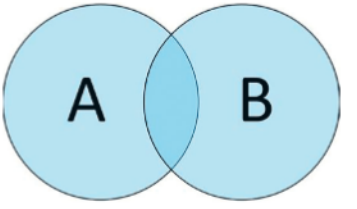

Figura 1. Representação gráfica da união dos conjuntos A e B utilizando o diagrama de Venn.

Antes de observarmos alguns exemplos, definiremos duas tabelas: a tabela `aluno`, com 4 alunos, contendo suas informações de nome, telefone e idade:

Nome	Telefone	Idade
Anthony Thomas Theo da Costa	(51) 995371754	29
Kaique Daniel Melo	(51) 983266773	24
Liz Eliane Antônia Aparício	(51) 984385050	21
Mariana Evelyn Porto	(51) 993324695	17
Yago Ricardo Nascimento	(51) 994909159	25

E a tabela `funcionario`, que também apresenta nome, telefone e idade.

Nome	Telefone	Idade
Andrea Sebastiana Mirella Moreira	(51) 989307088	47
Anthony Thomas Theo da Costa	(51) 995371754	29
Matheus da Silva Serpa	(51) 999764528	25
Rita Silvana Rocha	(51) 993946251	32

Outro ponto importante é que a operação de união em álgebra relacional não leva em consideração o nome dos atributos, e sim seus tipos de dados. Na literatura, considera-se que o resultado da operação de união (U) herda os nomes dos atributos da primeira tabela utilizada no operador. Como exemplo, vamos efetuar a operação `aluno` U `funcionario`, cujo resultado é ilustrado na tabela a seguir.

Nome	Telefone	Idade
Anthony Thomas Theo da Costa	(51) 995371754	29
Kaique Daniel Melo	(51) 983266773	24
Liz Eliane Antônia Aparício	(51) 984385050	21
Mariana Evelyn Porto	(51) 993324695	17
Yago Ricardo Nascimento	(51) 994909159	25
Andrea Sebastiana Mirella Moreira	(51) 989307088	47
Matheus da Silva Serpa	(51) 999764528	25
Rita Silvana Rocha	(51) 993946251	32

Nessa tabela, os campos são listados na mesma ordem como acontece na tabela `aluno`. Em geral, os campos da tabela que é a segunda no operador de união (nesse caso, a tabela `funcionario`) podem ter nomes de campos diferentes. Aqui, o importante é o tipo de cada atributo, e não o nome deles. Como resultado da operação, temos todas as tuplas que aparecem em alguma das tabelas. Caso haja repetições, a operação de união retorna apenas uma instância desta.

Saiba mais

Para aprender mais sobre álgebra relacional, incluindo a operação de união, pesquise na internet o artigo "Álgebra Relacional – Parte I", de autoria de Ricardo Rezende.

Um segundo exemplo que podemos analisar refere-se ao uso da união em conjunto com outros operadores de álgebra relacional. Por exemplo, dada uma lista de funcionários com seus respectivos departamentos e seus supervisores, desejamos retornar a quantidade de funcionários que trabalham no departamento 5 ou que supervisionam alguém que trabalha no departamento 5. Para tanto, podemos utilizar a operação de união como no exemplo a seguir.

Operações de álgebra relacional
funcionarios_dep5 ← σcodigo_departamento=5(funcionarios) resultado1 ← πcodigo_funcionario(funcionarios_dep5) resultado2 ← πcodigo_supervisor(funcionarios_dep5) resultado ← resultado1 ∪ resultado2

Primeiro, utilizando o operador de seleção, armazenamos as tuplas de todos funcionários que pertencem ao departamento de código igual a `5`. Depois, utilizando a variável na qual armazenamos a lista de funcionários, retornamos a coluna com o código de todos funcionários e a armazenamos em `resultado1`. Em seguida, armazenamos a coluna que tem o código de todos supervisores desses funcionários em `resultado2`. Por fim, utilizando o operador de união, retornamos todos códigos de funcionários que ou são do departamento `5` ou supervisionam alguém desse departamento.

No que diz a respeito à cardinalidade do resultado da operação de união, a operação R1 U R2 retorna todas as linhas da relação 1 seguida de todas linhas da relação 2, eliminando as linhas duplicadas (ou seja, linhas comuns aparecem apenas uma vez). No caso das colunas, o número de colunas retornado é o mesmo de ambas as relações.

Por fim, como já mencionado, duas relações são incompatíveis se o número e o domínio dos atributos são diferentes. Por exemplo, uma relação `aluno` com `id`, `nome`, `idade` e `curso` é compatível com uma relação `professor` com `id`, `nome`, `idade` e `setor`, mas não com uma relação `funcionario` com `id`, `nome`, `setor` e `idade`. Vale lembrar ainda que a operação de união é comutativa, ou seja, R1 U R2 é equivalente a R2 U R1.

Saiba mais

Leia mais sobre o operador de união no capítulo 5 do livro "Sistemas de Gerenciamento de Bancos de Dados", de Raghu Ramakrishnan e Johannes Gehrke.

2 Cláusulas `UNION` e `UNION ALL` em SQL

A linguagem de consulta estruturada SQL é utilizada principalmente para realização de consultas em bancos de dados, responsável pela implementação da maior parte das operações de álgebra relacional. A operação de união de conjuntos é implementada por meio de duas cláusulas condicionais, a `UNION` e a `UNION ALL`.

Já surgiram diversos casos em que se torna necessário combinar dois comandos `SELECT` para gerar um resultado para uma aplicação ou um usuário final — para isso, utilizamos a união. Como exemplo do uso de `UNION`, podemos assumir que existem duas tabelas: aquela que armazena todos professores e a que armazena todos os alunos de uma universidade, por exemplo, as tabelas `aluno` e `funcionario`, definidas anteriormente. Para mostrarmos o `nome` e a `idade` de todos os professores e alunos sem repetição, podemos utilizar o seguinte comando SQL:

```
SELECT nome, idade FROM professor
  UNION
SELECT nome, idade FROM aluno
```

Esse comando resultará em uma tabela como a do quadro do resultado da operação `aluno` U `funcionario` apresentada anteriormente, que mostra o resultado da operação de união de conjuntos, ainda que apenas com as colunas nome e idade. Por padrão, as linhas duplicadas são removidas dos resultados das instruções SQL com a palavra-chave `UNION` (MANNINO, 2008), ou seja, a cláusula `UNION` combina duas consultas SQL sem a repetição de dados, caso ocorra. Para manter as repetições, se elas existirem, adicionamos o comando `ALL`, resultando em `UNION ALL`.

No exemplo a seguir, assumiremos a existência de uma tabela `inscritos`, que apresenta como atributos o `codigo_canal`, o `nome` e o `sobrenome`. Assim, a consulta utilizará o comando `UNION ALL` para retornar uma lista de inscritos em dois canais do YouTube diferentes. O resultado será utilizado para um sorteio, em que a pessoa inscrita nos dois canais terá mais vantagem, pois será listada duas vezes. Essa operação executará mais rapidamente em comparação a uma que utilize dois `OR` na cláusula `WHERE`, resultado que explicaremos logo a seguir:

```
SELECT codigo_canal, nome, sobrenome FROM inscritos
    WHERE codigo_canal = 14
  UNION ALL
SELECT codigo_canal, nome, sobrenome FROM inscritos
    WHERE codigo_canal = 18
```

Um exemplo de resultado dessa consulta está no quadro a seguir.

codigo_canal	nome	sobrenome
14	Letícia	Correia
14	Julia	Castro
14	Matheus	Serpa
14	Gustavo	Dias
18	Luis	Costa
18	Mariana	Cavalcanti
18	Gustavo	Dias
18	Diego	Goncalves
18	Matheus	Serpa

Na tabela de resultado da consulta, temos que os usuários `Gustavo Dias` e `Matheus Serpa` seguem os dois canais e, portanto, têm mais chances de ganhar o sorteio que será realizado pelo aplicativo. Caso o operador `UNION` fosse utilizado no lugar de `UNION ALL`, esse usuário apareceria apenas uma vez.

Fique atento

Sempre que possível, utilize a cláusula `UNION` ou `UNION ALL` no lugar de uma cláusula `WHERE` com múltiplos `OR`. Mesmo que isso pareça contraintuitivo, o desempenho de duas consultas `SELECT` com `UNION ALL` é maior do que a utilização de vários `OR` no mesmo `WHERE`.

Para aprender mais sobre otimização de consultas SQL, pesquise na internet o artigo "Otimização de Consultas SQL", de Fábio Sarturi Prass.

No quesito desempenho, diversos autores, como Price (2009), ressaltam que o desempenho do `UNION ALL` é maior que o do `UNION`; então, quando está claro que não existe a possibilidade de repetições nos dados, devemos utilizar o operador `UNION ALL`, pois este não executa o comando `SELECT DISTINCT` no final, o qual causa lentidão mesmo quando não há repetições. Outro detalhe importante reside no fato de que não faz sentido utilizar o operador `UNION` quando alguma das consultas utilizar o comando `SELECT DISTINCT`, já que, nesse caso, o resultado será exatamente o mesmo, e, no nível de SQL, a operação será executada duas vezes desnecessariamente.

Exemplo

Confira um exemplo que mostra na prática a otimização de desempenho utilizada por especialistas em banco de dados SQL: a utilização da cláusula `UNION ALL` em substituição de cláusulas `OR`.
Por exemplo, a consulta

```
SELECT email FROM funcionarios WHERE cidade = "Porto Ale-
gre" or disp_viajar = "sim"
```

tem por objetivo listar o *e-mail* de todos os funcionários que ou são de Porto Alegre ou têm disponibilidade para viajar. Essa operação pode ser reescrita utilizando o operador `UNION ALL`, como a seguir:

```
SELECT email FROM funcionarios WHERE cidade = "Porto Alegre"
  UNION ALL
SELECT email FROM funcionarios WHERE disp_viajar = "sim"
```

Ambas as consultas retornarão o mesmo resultado, entretanto, utilizando `UNION ALL`, caso exista um índice armazenado para cidade, e não para `disp_viajar`, na primeira consulta a linha da tabela será verificada, enquanto, na segunda, o índice será utilizado, motivo pelo qual a consulta será executada mais rapidamente.

3 Operação de união no PostgreSQL

O PostgreSQL é um poderoso sistema gerenciador de banco de dados objeto relacional (SGDB) para aplicações modernas (CARVALHO, 2017), além de multiplataforma e de código aberto. Nos exemplos a seguir, utilizaremos os comandos `SQL UNION` e `UNION ALL` para efetuar consultas no banco de dados PostgreSQL.

O primeiro passo será criar algumas tabelas para efetuar as consultas. Criaremos a tabela `cliente`, com código, nome da empresa, nome do cliente e cidade, e uma tabela `fornecedor`, com código, nome da empresa e cidade. Os seguintes comandos realizam essa tarefa:

```
CREATE TABLE cliente(
    cod_cliente serial PRIMARY KEY,
    nome_empresa VARCHAR (200) NOT NULL,
    nome_cliente VARCHAR (100) NOT NULL,
    cidade VARCHAR (100) NOT NULL
);
CREATE TABLE fornecedor(
    cod_fornecedor serial PRIMARY KEY,
    nome_empresa VARCHAR (200) NOT NULL,
    cidade VARCHAR (100) NOT NULL
);
```

Após a criação das tabelas, inseriremos 6 clientes na tabela. Uma vez que colocamos o código do cliente como serial, o PostgreSQL o gerará automaticamente. Nesse sentido, utilizamos o comando `INSERT INTO` para inserir o nome da empresa, o nome do cliente e a cidade. O comando `VALUES` indica que um conjunto de valores será indicado a seguir. Depois, empregamos o mesmo comando para inserir 7 fornecedores. O código também é gerado de forma automática pelo SGDB, enquanto os outros dados são inseridos pelo `INSERT INTO`. O comando SQL utilizado foi:

```
INSERT INTO
    cliente (nome_empresa, nome_cliente, cidade)
VALUES
    ('Ponto Shop', 'Renata Rezende', 'Porto Alegre'),
   ('Sarah Estética Capilar', 'Cláudia Lima', 'Porto Alegre'),
    ('Segredos de Minas', 'André Drumond', 'Alegrete'),
    ('Sabor Lusitano', 'Juan das Neves', 'Rio de Janeiro'),
    ('Banco Verde', 'Marcos de Paula', 'Rio de Janeiro'),
    ('Pais e Filhos', 'Leonardo Duarte', 'São Paulo');
INSERT INTO
     fornecedor (nome_empresa, cidade)
VALUES
    ('Reffatti', 'São Paulo'),
    ('Soberania das Cadeiras', 'Belo Horizonte'),
    ('Armazém das Cadeiras', 'São Paulo'),
    ('Cia das Cadeiras', 'Rio de Janeiro'),
    ('mT Mesas e Cadeiras', 'Porto Alegre'),
    ('Dendezeiros Cadeiras', 'Salvador'),
    ('Point Móveis para Escritório', 'Porto Alegre');
```

Agora, dispomos de um banco de dados com duas tabelas que já têm dados. Em nosso primeiro exemplo, desejamos listar as cidades nas quais dispomos de clientes ou fornecedores, sem repeti-las:

```
SELECT cidade FROM cliente
    UNION
SELECT cidade FROM fornecedor
    ORDER BY cidade
```

Este comando trará como resultado:

	cidade
1	Alegrete
2	Belo Horizonte
3	Porto Alegre
4	Rio de Janeiro
5	Salvador
6	São Paulo

Para tanto, podemos utilizar a cláusula `UNION`: primeiro, selecionamos a lista de cidades da tabela `cliente` e, depois, as cidades dos fornecedores; por fim, com `UNION`, unimos ambos resultados, removemos os duplicados e ordenamos por cidade utilizando a cláusula `ORDER BY`. Como resultado, temos as cidades `Alegrete`, `Belo Horizonte`, `Porto Alegre`, `Rio de Janeiro`, `Salvador` e `São Paulo`. Note que os resultados duplicados foram removidos.

Exemplo

Veja mais exemplos do uso de `UNION` e `UNION ALL`, além de outras cláusulas, no "Guia do desenvolvedor de banco de dados" da plataforma Amazon Redshift.

Em um segundo exemplo, mais complexo, temos como objetivo listar a quantidade de clientes ou fornecedores em cada cidade, em ordem decrescente da quantidade de clientes ou fornecedores na cidade. Para tanto, nosso primeiro passo foi listar todas as cidades por `cliente` e por `fornecedor`. Depois, utilizamos o comando `UNION ALL` para unir ambos os resultados mantendo os duplicados, o que foi armazenado como `cidades`. Nesse momento, temos a lista de todas as cidades de clientes e fornecedores mantendo as duplicadas. A partir desse novo conjunto de dados, listaremos as cidades e a quantidade de vezes que cada uma aparece na lista, as agruparemos por cidade e, por fim, ordenaremos pela quantidade de forma decrescente. O comando SQL para realizar essa operação pode ser visto a seguir.

```
SELECT COUNT(cidade), cidade FROM
    (
    SELECT cidade FROM cliente
        UNION ALL
    SELECT cidade FROM fornecedor
    )
    AS cidades
GROUP BY cidade
ORDER BY COUNT(cidade) DESC
```

Após a execução desse comando SQL, o PostgreSQL retorna os seguintes dados:

count	cidade
4	Porto Alegre
3	Rio de Janeiro
3	São Paulo
1	Belo Horizonte
1	Salvador
1	Alegrete

A cidade de `Porto Alegre` se destaca com `4` aparições, `Rio de Janeiro` e `São Paulo` vêm em seguida com `3`, e as outras cidades têm apenas `1`. Além do uso das cláusulas, é importante entender o seu funcionamento e desempenho.

O comando `EXPLAIN` do PostgreSQL analisa uma *query* (consulta) e retorna o plano de execução para ela. Nesse plano, de acordo com o SGDB, podemos ver quais estruturas de dados, índices e algoritmos são utilizadas, além de analisar o tempo de execução total e o tempo consumido por cada etapa da *query*, dados que podem ser utilizados para otimizar as consultas.

Saiba mais

Acesse a documentação oficial do PostgreSQL para entender melhor o funcionamento do `EXPLAIN`.

Na Figura 2, temos um exemplo de saída do comando `EXPLAIN`. A *query* analisada foi a do exemplo anterior: o primeiro passo consiste em listar os dados das tabelas `cliente` e `fornecedor`; depois, faz-se um acréscimo após a chamada do `UNION`; em seguida, realiza-se uma chamada de agregação com o intuito de agrupar as cidades pela quantidade; e, finalmente, chama-se uma classificação para ordenar (`ORDER BY`) por cidades em ordem decrescente (`DESC`). Além dessa informação, na segunda parte da Figura 2, podemos analisar o tempo gasto e o custo de cada operação. É possível observar que o maior custo (51% do tempo) se deu na etapa Seq Scan. Essa operação refere-se a percorrer as tabelas `cliente` e `fornecedor` do início ao fim. Depois, temos os custos da agregação e da classificação. Por fim, o menor custo foi o da operação de acréscimo (*append*), basicamente refletida pela cláusula `UNION ALL`.

Statistics per Node Type

Node type	Count	Time spent	%% of query
Aggregate	1	0.016 ms	31.38%
Append	1	0.003 ms	5.89%
Seq Scan	2	0.026 ms	50.99%
Sort	1	0.007 ms	13.73%

Figura 2. Exemplo de plano de execução de um *query* no PostgreSQL.

Neste capítulo, estudamos a utilização da operação de união de conjuntos em álgebra relacional e as cláusulas `UNION` e `UNION ALL` da linguagem SQL. Por fim, aplicamos vários comandos SQL, incluindo o `UNION` e o `UNION ALL`, no gerenciador de banco de dados PostgreSQL.

Referências

CARVALHO, V. *PostgreSQL*: banco de dados para aplicações web modernas. São Paulo: Casa do Código, 2017. 220 p.

MANNINO, M. V. *Projeto, desenvolvimento de aplicações e administração de banco de dados*. 3. ed. Porto Alegre: AMGH; 2008. 717 p.

MENEZES, P. B. *Matemática discreta para computação e informática*. 4. ed. Porto Alegre: Bookman, 2013. 370 p. (Série Livros Didáticos Informática UFRGS, 16).

PRICE, J. *Oracle Database 11g SQL*: domine SQL e PL/SQL no banco de dados Oracle. Porto Alegre: Bookman; Oracle Press, 2009. 684 p.

RAMAKRISHNAN, R..; GEHRKE, J. *Sistemas de gerenciamento de banco de dados*. 3. ed. Porto Alegre: AMGH; Bookman, 2008. 905 p.

Leituras recomendadas

EXEMPLOS de consultas UNION. *In*: Amazon Redshift: Guia do desenvolvedor de banco de dados. *AWS*, Seattle, 2020. Disponível em: https://docs.aws.amazon.com/pt_br/redshift/latest/dg/c_example_union_query.html. Acesso em: 2 jun. 2020.

EXEMPLOS de consultas UNION ALL. *In*: Amazon Redshift: Guia do desenvolvedor de banco de dados. *AWS*, Seattle, 2020. Disponível em: https://docs.aws.amazon.com/pt_br/redshift/latest/dg/c_example_unionall_query.html. Acesso em: 2 jun. 2020.

PRASS, F. S. Otimização de Consultas SQL. *DevMedia*, Rio de Janeiro, 2015. Disponível em: https://www.devmedia.com.br/algebra-relacional-parte-i/2663. Acesso em: 2 jun. 2020.

REZENDE, R. Álgebra Relacional – Parte I. *DevMedia*, Rio de Janeiro, 2006. Disponível em: https://www.devmedia.com.br/algebra-relacional-parte-i/2663. Acesso em: 2 jun. 2020.

UTILIZAÇÃO do comando EXPLAIN. *In*: Documentação do PostgreSQL 8.0.0: Projeto de Tradução para o Português do Brasil. *The PostgreSQL Global Development Group*, [*S. l.*], 2005. Disponível em: http://pgdocptbr.sourceforge.net/pg80/performance-tips.html#USING-EXPLAIN. Acesso em: 2 jun. 2020.

Fique atento

Os *links* para *sites* da *web* fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declaram não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS