

Apresentação

A *Structured Query Language* (SQL) e as demais linguagens utilizadas em sistemas de gerenciamento de banco de dados (SGBD) estão presentes em aplicativos para *smartphones* ou sistemas de informação utilizados diariamente. A SQL é uma linguagem de consulta estruturada, pela qual é possível realizar diferentes tipos e formas de consultas, sejam básicas ou complexas, para recuperar informações em um banco de dados.

Nesta Unidade de Aprendizagem, você estudará as funções de agregação e agrupamento em SQL. Além disso, verá exemplos da criação de consultas em SQL utilizando junção, agregação e ordenação. Por fim, aprenderá sobre consultas avançadas empregando aninhamento e correlacionamento.

Bons estudos.

Ao final desta Unidade de Aprendizagem, você deve apresentar os seguintes aprendizados:

- Descrever funções de agregação e agrupamento na linguagem SQL.
- Desenvolver consultas avançadas em banco de dados usando junção, agregação e ordenação.
- Produzir consultas avançadas empregando conceitos de aninhamento e correlacionamento.

Infográfico

Em uma consulta em SQL somente as cláusulas SELECT e FROM são obrigatórias. A cláusula SELECT lista os atributos a serem recuperados, enquanto a cláusula FROM especifica todas as tabelas úteis para a consulta, bem como suas relações. Já a cláusula WHERE é utilizada para especificar as condições para selecionar as tuplas dessas relações, bem como condições de junção, quando necessário. A cláusula GROUP BY especifica os atributos para o agrupamento, enquanto que a cláusula ORDER BY define uma ordem para a exibição do resultado de uma consulta.

As funções de agregação COUNT, SUM, MIN, MAX e AVG são empregadas em conjunto com o agrupamento. A função COUNT conta o número de tuplas ou valores, conforme for definido na consulta e a COUNT DISTINCT conta somente elementos distintos, enquanto a função SUM possibilita a soma de valores; MAX é empregada para verificar um valor máximo; MIN para um valor mínimo e a AVG para uma média de valores.

Neste Infográfico, você vai ver uma representação gráfica de algumas destas cláusulas e funções para facilitar sua compreensão.

CLÁUSULAS E FUNÇÕES DE AGREGAÇÃO SQL

O objetivo deste Infográfico é tornar o conteúdo abordado na disciplina mais visível e compreensível com imagens e exemplos simples, além dos textos e exemplos empregados previamente.

Veja a descrição e os exemplos práticos das principais cláusulas e funções de agregação utilizadas para a criação de consultas em SQL.



SELECT

Seleciona dados em um banco de dados.

Exemplo:
SELECT nome FROM alunos;

FROM

Especifica todas as tabelas e/ou relações úteis para a consulta.

Exemplo:
SELECT nome FROM alunos;



WHERE

Especifica condições na consulta com base em operações específicas.

Exemplo:
SELECT nome FROM alunos
WHERE idade=20;

ORDER BY

Ordena o resultado da consulta pela coluna especificada.

Exemplo:
SELECT * FROM alunos
ORDER BY nome;



GROUP BY

Agrupa o resultado de funções agregadas no SELECT pela coluna que for especificada.

Exemplo:
SELECT faculdade, COUNT (*) FROM alunos
GROUP BY faculdade;

AND E OR

O operador AND determina que mais de uma condição especificada na operação seja verdadeira e o OR que ao menos uma das condições especificadas na operação seja verdadeira.

Exemplo:
SELECT CPF FROM alunos
WHERE matricula=1234 AND idade=22

SELECT CPF FROM alunos
WHERE matricula=1234 OR idade=22



COUNT

Conta o número de linhas em uma consulta.

Exemplo:
SELECT COUNT (*) FROM faculdade;

SUM

Soma os valores de uma seleção.

Exemplo:
SELECT SUM (valor) FROM produto;



AVG

Retorna a média dos valores de uma seleção.

Exemplo:
SELECT AVG (valor) FROM produto;

MAX E MIN

Retornam o maior (MAX) e o menor (MIN) valor de uma seleção.

Exemplo:
SELECT MAX (valor) FROM produto;

SELECT MIN (valor) FROM produto;



Essas são apenas algumas das possibilidades de utilização de cláusulas e funções, existem ainda muitas outras possíveis.



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Conteúdo do Livro

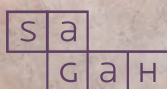
Bancos de dados são muito úteis para empresas, pois garantem organização, controle e segurança de suas informações. Por isso, é evidente sua importância para o bom funcionamento das atividades dessas organizações e para que suas missões e visões sejam alcançadas. A fim de que isso seja possível, é necessário que você, profissional de Tecnologia da Informação, tenha conhecimento sobre este tipo de ferramenta, trabalhando nestas questões como analista contratado ou como proprietário de sua própria empresa.

No capítulo Consultas complexas em SQL, do livro *Banco de Dados*, base teórica desta Unidade de Aprendizagem, você vai estudar sobre consultas complexas em SQL. Além disso, vai ver como os blocos com SELECT, FROM e WHERE completos dentro de uma cláusula WHERE podem ser utilizados ao longo de outra consulta, a chamado de consulta aninhada.

Boa leitura.

BANCO DE DADOS

Roni Francisco Pichetti



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS



Consultas complexas em SQL

Objetivos de aprendizagem

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

- Descrever funções de agregação e agrupamento na linguagem SQL.
- Desenvolver consultas avançadas em banco de dados usando junção, agregação e ordenação.
- Produzir consultas avançadas empregando conceitos de aninhamento e correlacionamento.

Introdução

Os bancos de dados podem armazenar informações sobre diferentes áreas do conhecimento, como no controle de estoques ou no cadastro de colaboradores em uma organização. Por exemplo, considerando os dados dos colaboradores, esses bancos abrangem suas informações pessoais, seu relacionamento com a empresa (setor, departamento, supervisão), bem como salários, benefícios e auxílios. Muitos tipos e formas de consultas podem ser realizadas em um banco de dados que utiliza a linguagem de consulta estruturada (SQL, do inglês *structured query language*) com essas informações. Algumas funções podem agilizar o trabalho do usuário final, como realizar somas de valores, calcular médias ou contar registros de colaboradores pela própria linguagem de consulta.

Neste capítulo, você vai estudar as funções de agregação e agrupamento em SQL. Você também vai conferir exemplos da criação de consultas em SQL utilizando junção, agregação e ordenação e, ainda, vai aprender sobre consultas avançadas empregando aninhamento e correlacionamento.

1 Funções de agregação e agrupamento em SQL

As funções em SQL são recursos que facilitam diferentes tarefas consideradas corriqueiras. Elas são divididas em grupos de características operacionais, como manipulação de datas, operações matemáticas, entre outras. As **funções de agregação** são utilizadas para resumir informações de diferentes tuplas em uma síntese de tuplas única. Conforme lecionam Ramakrishnan e Gehrke (2011), além de obter dados simples, muitas vezes, é necessário aplicar pesquisas com resumos ou computações em grupos de dados. Assim, o agrupamento é empregado para criar subgrupos de tuplas antes do resumo.

O agrupamento e a agregação são necessários em diferentes aplicações de bancos de dados. Entre as funções de agregação embutidas, estão: `COUNT`, `COUNT DISTINCT`, `SUM`, `MAX`, `MIN` e `AVG`. De acordo com Manzano (2017), a sintaxe da definição e do uso das funções de agregação é a seguinte:

```
NOME_FUNÇÃO ([ALL / DISTINCT] / [expressão]);
```

onde o `NOME_FUNÇÃO` se refere à definição de uma função de agregação, e `ALL` é responsável por aplicar a função de agregação em todos os valores da expressão. Já `DISTINCT` aplica a ação da função uma única vez em cada instância, e `expressão` trata do tipo de valor obtido no campo de um registro, que pode ser numérico, por exemplo (MANZANO, 2017).

A função `COUNT` conta o número de tuplas ou valores, conforme for definido na consulta, e a função `COUNT DISTINCT` conta somente elementos distintos. Já a função `SUM` possibilita a soma de valores, `MAX` é empregada para verificar um valor máximo, `MIN` é utilizada para verificar um valor mínimo, e `AVG` é usada para verificar uma média de valores. Essas funções podem ser aplicadas tanto em um conjunto quanto em um multiconjunto de valores. As funções `MAX` e `MIN` também podem ser utilizadas com atributos que possuem domínios não numéricos, desde que esses valores estejam em uma ordenação total entre si, como os domínios `DATE` e `TIME` (ELMASRI; NAVATHE, 2011).



Exemplo

A seguir, você pode conferir um exemplo de consulta com funções de agregação.

```
SELECT SUM (Salario), MAX (Salario), MIN (Salario), AVG  
(Salario)  
FROM COLABORADOR;
```

Com a consulta do exemplo, verifica-se a soma, os valores máximo e mínimo e a média dos salários dos colaboradores cadastrados no banco de dados. Caso seja necessário obter os valores separados de um departamento específico, pode ser utilizada a cláusula `WHERE`. Assim, os resultados da consulta sobre os salários dos colaboradores serão restringidos pelo departamento que for descrito na cláusula `WHERE` (ELMASRI; NAVATHE, 2011).

O exemplo a seguir traz uma consulta em que é utilizada a função `COUNT`:

```
SELECT COUNT (*)  
FROM COLABORADOR, DEPARTAMENTO  
WHERE Dnr=dnumero  
AND Dnome='Faturamento';
```

Nesse caso, a consulta trata de verificar e contar o número de colaboradores cadastrados no banco de dados que fazem parte do departamento `'Faturamento'`. Veja que o asterisco (*) se refere a todas as tuplas da tabela, de modo que `COUNT (*)` resulta no número total de linhas com a informação solicitada, e não de seus atributos. Caso seja necessário utilizar a função `COUNT` em valores de uma única coluna, ela pode ser especificada na função. Se empregada aliada ao `DISTINCT`, os valores duplicados serão eliminados.

Cabe ressaltar que quaisquer tuplas com valor nulo (`NULL`), ao se utilizar essa função, não serão contadas, visto que, geralmente, os valores `NULL` são descartados quando as funções de agregação são aplicadas a uma coluna. Portanto, pode-se compreender que o uso das funções de agregação gera um resumo da parte do banco de dados que for consultada, conforme lecionam Elmasri e Navathe (2011).

Existe a possibilidade de se aplicar funções de agregação a subgrupos de tuplas em uma relação, como o salário médio dos colaboradores de cada departamento da empresa ou o número de colaboradores envolvidos no trabalho de cada projeto. Para isso, é preciso dividir a relação em subconjuntos de tuplas não sobrepostos. Cada partição será formada pelas tuplas que possuírem o mesmo valor em um ou mais atributos, os quais são chamados de **atributos de agrupamento**. Assim, a função pode ser aplicada a cada grupo de forma independente, para se produzir um resumo. A SQL possui a cláusula `GROUP BY`, útil nesses casos (ELMASRI; NAVATHE, 2011).

De acordo com Ramakrishnan e Gehrke (2011), a cláusula `GROUP BY` é empregada nos casos em que é preciso aplicar operações agregadas a cada um dos vários grupos de tuplas em uma relação. Dessa forma, essa cláusula especifica os atributos de agrupamento, para que o valor resultante do emprego de cada função de agregação a um grupo apareça junto com o valor dos seus atributos.



Exemplo

Veja um exemplo do uso da cláusula `GROUP BY` (ELMASRI; NAVATHE, 2011):

```
SELECT Dnr, COUNT (*), AVG (Salario)
FROM COLABORADOR
GROUP BY Dnr;
```

A consulta do exemplo procura recuperar o número do departamento, o número de colaboradores no departamento e o salário médio em cada departamento. Para isso, as tuplas `COLABORADOR` são divididas em grupos pelo valor do atributo `Dnr`, que é o número do departamento. Assim, cada grupo contém os colaboradores que trabalham no departamento determinado. As funções `COUNT` e `AVG` são aplicadas separadamente em cada grupo de tuplas. Caso houver um ou mais valores `NULL` no atributo que será utilizado para o agrupamento, é criado um grupo separado para todas essas tuplas (ELMASRI; NAVATHE, 2011). Para facilitar a compreensão, nos quadros a seguir é apresentado o resultado da consulta do exemplo com `GROUP BY` em forma de tabela.

Pnome	...	Unome	Cpf	Salario	Dnr
Diego		Silva	12345678966	3.000	5
Paulo		Souza	33344455566	2.000	5
Ana		Freitas	66677788899	10.000	5
Lia		Martins	98765432199	7.500	4
Jorge		Lima	98798798798	4.500	4
João		Brito	87687687676	6.000	4
Regina		Pereira	54545454545	15.000	1

Fonte: Adaptado de Elmasri e Navathe (2011).

Dnr	COUNT (*)	AVG (Salario)
5	3	5.000
4	3	6.000
1	1	15.000

Fonte: Adaptado de Elmasri e Navathe (2011).

De acordo com os dados presentes nos quadros, é possível verificar que a consulta criou três grupos distintos, de acordo com o número do departamento, e aplicou as funções COUNT e AVG às respectivas tuplas componentes dos grupos. A coluna sem informações mostra que a tabela COLABORADORES possui mais dados, que não estão visíveis no exemplo.

O próximo exemplo trata de uma consulta sobre o número e o nome de projetos e o número de colaboradores que trabalham em cada projeto, utilizando agrupamento.



Exemplo

Veja o código a seguir (ELMASRI; NAVATHE, 2011):

```
SELECT Projnumero, Projnome, COUNT (*)  
FROM PROJETO, TRABALHA_EM  
WHERE Projnumero=Pnr  
GROUP BY Projnumero, Projnome;
```

Neste exemplo, fica evidente a possibilidade de utilizar uma condição de junção em um conjunto `GROUP BY`, visto que o agrupamento e as funções são aplicadas depois da junção das duas relações. Pode ser necessário aplicar funções somente a grupos que satisfazem determinadas condições. Levando-se em conta o exemplo dos números de projetos, pode ser necessário pesquisar apenas os projetos que contam com mais de dois colaboradores. Para isso, a SQL possui a cláusula `HAVING`, que pode ser empregada junto à cláusula `GROUP BY`. A cláusula `HAVING` possibilita incluir uma condição sobre a informação de resumo referente ao grupo de tuplas criado pela cláusula `GROUP BY`. Dessa forma, somente os grupos que contemplarem a condição serão recuperados no resultado da consulta (ELMASRI; NAVATHE, 2011).

2 Consultas avançadas em banco de dados usando junção, agregação e ordenação

Uma consulta em SQL pode possuir até seis cláusulas, mas somente as cláusulas `SELECT` e `FROM` são obrigatórias. A consulta pode ser dividida em várias linhas e deve terminar com um sinal de ponto e vírgula. Os termos de uma consulta são separados por espaços, sendo que parênteses podem ser utilizados para agrupar partes relevantes de uma consulta.

A cláusula `SELECT` lista os atributos a serem recuperados, enquanto a cláusula `FROM` especifica todas as tabelas úteis para a consulta, bem como suas relações. Já a cláusula `WHERE` é utilizada para especificar as condições para selecionar as tuplas dessas relações, bem como condições de junção, quando necessário. A cláusula `GROUP BY` especifica os atributos para o agrupamento, e a cláusula `HAVING`, uma condição sobre os grupos selecionados. As funções de agregação `COUNT`, `SUM`, `MIN`, `MAX` e `AVG` são empregadas em conjunto

com o agrupamento e também podem ser aplicadas em uma consulta sem agrupamento. A cláusula `ORDER BY`, por sua vez, define uma ordem para a exibição do resultado de uma consulta (ELMASRI; NAVATHE, 2011).

Para formular consultas de maneira correta, é útil considerar as etapas que definem o significado ou a semântica de cada consulta. Uma consulta é avaliada conceitualmente aplicando primeiro a cláusula `FROM` (para identificar todas as tabelas envolvidas na consulta ou materializar quaisquer tabelas de junção), seguida pela cláusula `WHERE` para selecionar e juntar tuplas, e depois por `GROUP BY` e `HAVING`. Conceitualmente, `ORDER BY` é aplicado no final para classificar o resultado da consulta. Se nenhuma das três últimas cláusulas (`GROUP BY`, `HAVING` e `ORDER BY`) for especificada, podemos pensar conceitualmente em uma consulta como sendo executada da seguinte forma: para cada combinação de tuplas, [...] avaliar a cláusula `WHERE`; se ela for avaliada como `TRUE`, colocar os valores dos atributos especificados na cláusula `SELECT` dessa combinação de tuplas no resultado da consulta. (ELMASRI; NAVATHE, 2011, p. 86).

Existem várias formas de especificar a mesma consulta em SQL. Uma vantagem dessa flexibilidade é que os usuários podem escolher a técnica com a qual estão mais acostumados para definir uma consulta, enquanto, para o programador do sistema, a fim de otimizar a consulta, é preferível que uma consulta tenha o mínimo de ordenação possível, por exemplo. Uma desvantagem de se ter diferentes maneiras de escrever a mesma consulta é que isso pode criar confusão para o usuário, que pode não saber qual é a técnica correta de consulta para cada situação. Além disso, ainda que seja mais eficiente executar uma consulta de uma forma específica, essa forma pode não ser empregada. De qualquer forma, o ideal é que o usuário se preocupe em especificar a consulta corretamente, enquanto o sistema de gerenciamento de banco de dados determina como colocar em prática a consulta de maneira eficiente, conforme lecionam Elmasri e Navathe (2011).

É necessário ter cuidado quando duas condições diferentes se aplicam em uma consulta, para que uma não se sobreponha à outra, como quando utilizadas as cláusulas `WHERE` e `HAVING` em sequência (ELMASRI; NAVATHE, 2011). De acordo com Ramakrishnan e Gehrke (2011), `HAVING` é uma cláusula opcional, que pode ser empregada para especificar qualificações nos grupos definidos pelo uso da cláusula `GROUP BY`.



Exemplo

Verifique a seguir um exemplo incorreto e um exemplo correto das consultas com agregação propostas (ELMASRI; NAVATHE, 2011).

Consulta incorreta:

```
SELECT Dnome, COUNT (*)
FROM DEPARTAMENTO, COLABORADOR
WHERE Dnumero=Dnr AND Salario>5.000
GROUP BY Dnome;
HAVING COUNT (*) > 5;
```

Consulta correta:

```
SELECT Dnumero, COUNT (*)
FROM DEPARTAMENTO, COLABORADOR
WHERE Dnumero=Dnr AND Salario>5.000 AND
    (SELECT Dnr IN
     FROM COLABORADOR
     GROUP BY Dnr
     HAVING COUNT (*) > 5);
```

A primeira consulta do exemplo está incorreta, pois vai selecionar somente os departamentos que tenham mais de cinco funcionários, cada um com um salário de mais de R\$ 5.000. Sabe-se que a cláusula **WHERE** é executada primeiro, a fim de selecionar as tuplas individuais ou tuplas de junção; assim, a cláusula **HAVING** é aplicada depois, para selecionar os grupos individuais de tuplas. Dessa forma, as tuplas já estarão restritas a colaboradores que recebem mais de R\$ 5.000 antes que a cláusula **HAVING** possa ser aplicada, e o resultado obtido não será o esperado. Dito isso, uma das formas corretas de realizar essa mesma consulta é com o uso de consulta aninhada, que será detalhada na sequência. O que foi feito para que a cláusula fosse aplicada integralmente foi alterar a ordem da pesquisa. Dessa forma, é possível consultar primeiro o departamento que possui mais de cinco funcionários e, depois, o número do departamento e de colaboradores com salário maior do que R\$ 5.000 (ELMASRI; NAVATHE, 2011).

Consultas avançadas podem ser realizadas em SQL por meio de junções. A SQL possui suporte para operações de junção de diferentes tipos, entre elas:

- a `INNER JOIN`, que trata de uma junção interna de duas ou mais tabelas, na qual uma tupla somente é incluída no resultado se combinar com a outra relação; e
- a `NATURAL JOIN`, por meio da qual cada par de atributos é incluído apenas uma vez na relação resultante.

Caso os nomes dos atributos de junção não sejam os mesmos nas relações iniciais, é possível alterar seus nomes para que combinem e aplicar a `NATURAL JOIN` somente depois. Se o resultado for nulo para uma das relações, a tupla é excluída da tabela de junção.

Existem diferentes opções disponíveis para distinguir tabelas de junção; entre elas, estão:

- `LEFT OUTER JOIN`, na qual toda tupla da tabela da esquerda precisa estar presente no resultado, caso contrário, é preenchida com valores nulos;
- `RIGHT OUTER JOIN`, em que toda tupla da tabela da direita precisa estar presente no resultado, caso contrário, é preenchida com valores nulos.

Os próximos exemplos apresentam consultas utilizando junções internas e ordenação.



Exemplo

Veja, a seguir, uma consulta sobre o nome e o endereço de todos os colaboradores que trabalham no departamento de Compras utilizando `JOIN` (ELMASRI; NAVATHE, 2011).

```
SELECT Pnome, Unome, Endereco
FROM (COLABORADOR JOIN DEPARTAMENTO ON Dnr=Dnumero)
WHERE Dnome= 'Compras'
ORDER BY Pnome, Unome;
```

A seguir, é apresentada uma consulta em que a relação DEPARTAMENTO é renomeada como DEP e seus atributos são renomeados para combinar com os da tabela Colaborador, utilizando NATURAL JOIN.

```
SELECT Pnome, Unome, Endereco
FROM (COLABORADOR NATURAL JOIN
      (DEPARTAMENTO AS DEP
        (Dnome, Dnr, Cpf_gerente)))
WHERE Dnome= 'Compras'
ORDER BY Pnome, Unome;
```

Nesses exemplos, pode ser observado que os resultados das consultas com junções serão apresentados de forma ordenada, pelo primeiro nome e pelo último nome, respectivamente, por conta do uso da cláusula ORDER BY.

As operações de junção também podem ser utilizadas combinadas às funções de agrupamento e agregação, conforme pode ser observado no próximo exemplo.



Exemplo

Veja, a seguir, uma consulta sobre a soma dos salários de todos os colaboradores vinculados ao departamento de Compras, assim como os salários máximo e mínimo e a média dos salários desse departamento (ELMASRI; NAVATHE, 2011).

```
SELECT SUM (Salario), MAX (Salario), MIN (Salario), AVG
(Salario)
FROM (COLABORADOR JOIN DEPARTAMENTO ON Dnr=Dnumero)
WHERE Dnome= 'Compras';
```

3 Consultas avançadas empregando conceitos de aninhamento e correlacionamento

Determinadas consultas necessitam que os valores presentes no banco de dados sejam buscados e também comparados. Esse tipo de consulta pode ser formulado utilizando-se **consultas aninhadas**, que são blocos com `SELECT`, `FROM` e `WHERE` completos dentro de uma cláusula `WHERE` de outra consulta. A outra consulta é chamada, nesse caso, de **consulta externa** (ELMASRI; NAVATHE, 2011). De acordo com Ramarkrishnan e Gehrke (2011), as consultas aninhadas são um dos recursos mais poderosos da SQL. Os autores consideram a consulta aninhada aquela que possui outra ou outras consultas embutidas dentro dela, sendo que a consulta embutida é chamada de **subconsulta**.



Exemplo

Veja, a seguir, um exemplo de consulta aninhada (ELMASRI; NAVATHE, 2011).

```
SELECT DISTINCT Projnumero
FROM PROJETO
WHERE Projnumero IN
    (SELECT Projnumero
     FROM PROJETO,
          DEPARTAMENTO,
          FUNCIONARIO
          Dnum=Dnumero AND
     WHERE Cpf _ gerente=Cpf
          AND Unome='Silva')
OR Projnumero IN
    (SELECT Pnr
     FROM TRABALHA _ EM,
          FUNCIONARIO
     WHERE Fcpf=Cpf AND
          Unome='Silva');
```


A primeira consulta aninhada seleciona números de projetos que possuem um funcionário cujo último nome (sobrenome) é igual a Silva e que seja gerente. Já na segunda consulta aninhada, são selecionados os números dos projetos com um funcionário também de sobrenome Silva, mas que tenha o vínculo de trabalhador. Na consulta externa, é empregado o conectivo lógico OR, a fim de que o resultado possa apresentar um PROJETO cujo PROJNUMERO esteja em qualquer uma das consultas aninhadas. De forma geral, a consulta aninhada resulta em uma tabela que é um conjunto ou multiconjunto de tuplas (ELMASRI; NAVATHE, 2011).

Quando uma consulta aninhada retorna um único atributo e uma tupla única, o resultado é um único valor. Nesse caso, pode-se utilizar o sinal = em vez do IN para o operador de comparação. De qualquer forma, geralmente a consulta aninhada resulta em uma tabela, que é formada por um conjunto de tuplas. As consultas podem estar aninhadas em vários níveis. Por conta disso, é necessário lidar com a possibilidade de ambiguidade entre nomes de atributos que possuam a mesma descrição. A regra, nesses casos, é que uma referência a um atributo não qualificado se refere à relação declarada na consulta mais interna. Essa regra é semelhante às regras de escopo para variáveis de programa em diferentes linguagens de programação, conforme apontam Elmasri e Navathe (2011).



Exemplo

O próximo exemplo trata da ambiguidade em potencial de nomes de atributo nas consultas aninhadas (ELMASRI; NAVATHE, 2011).

```
SELECT C.Pnome, C.Unome
FROM COLABORADOR AS C
WHERE C.Cpf IN
      (SELECT D.Ccpf
       FROM DEPENDENTE AS D
       WHERE C.Pnome=D.Niome_dependente
          AND C.Sexo=D.Sexo);
```

A consulta do exemplo busca recuperar o nome de cada colaborador que possui um dependente com o mesmo nome e com o mesmo sexo do funcionário. Na consulta aninhada do exemplo, foi necessário qualificar `C.Sexo`, pois se refere ao atributo `Sexo` de `COLABORADOR` na consulta externa, visto que `DEPENDENTE` também possui o mesmo atributo, com a mesma nomenclatura. Caso houvesse alguma referência não qualificada ao atributo `Sexo` na consulta aninhada, eles se refeririam ao atributo de `DEPENDENTE`. Já a qualificação dos atributos `Pnome` e `Cpf` de `COLABORADOR` não seria necessária, já que a relação `DEPENDENTE` não possui atributos com os mesmos nomes. Para evitar erros e ambiguidade, é recomendável criar variáveis de tupla para todas as referências em uma consulta, conforme realizado na consulta do exemplo (ELMASRI; NAVATHE, 2011).

As consultas aninhadas podem ser consideradas **correlacionadas**, quando uma condição da cláusula `WHERE` de uma consulta aninhada faz referência a algum atributo de uma relação declarada na consulta externa. Para compreender essa correlação, pense que a consulta aninhada é avaliada uma vez para cada linha (ou tupla) na consulta externa. Levando-se em conta o exemplo anterior, para cada tupla de `COLABORADOR`, a consulta aninhada recupera os valores de `Ccpf` para todas as tuplas de `DEPENDENTE` com o mesmo sexo e nome da tupla da tabela de colaboradores. Caso o valor do `Ccpf` estiver no resultado da consulta aninhada, então, é selecionada essa tupla em `COLABORADOR`. De forma geral, uma consulta escrita com blocos aninhados, utilizando as cláusulas `SELECT`, `FROM` e `WHERE` e usando operadores de comparação `=` ou `IN`, sempre pode ser definida com uma única consulta em bloco (ELMASRI; NAVATHE, 2011).

A linguagem SQL possui um grande número de recursos para que seus usuários possam especificar recuperações complexas do banco de dados. Nesse sentido, vale ressaltar a importância das regras para se lidar com valores `NULL` (nulos). A expressão `NULL` é utilizada para identificar valores que estão faltando, os quais podem ser desconhecidos, não disponíveis ou, ainda, não aplicáveis. Geralmente, cada `NULL` é considerado diferente de qualquer outro presente nos registros do banco de dados. Assim, quando um valor `NULL` passa por uma operação de comparação, o resultado acaba sendo considerado `UNKNOWN` (desconhecido), `TRUE` (verdadeiro) ou `FALSE` (falso). Portanto, a linguagem SQL utiliza uma **lógica de três valores**, com `TRUE`, `FALSE` e `UNKNOWN`, em vez da lógica booleana padrão, que emprega apenas dois valores. Dessa forma, é necessário definir os três valores verdadeiros das expressões

que utilizarem os conectivos lógicos AND, OR e NOT. O Quadro 1 apresenta os valores resultantes possíveis nos três casos (ELMASRI; NAVATHE, 2011).

Quadro 1. Conectivos lógicos na lógica de três valores

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT	—	—	—
TRUE	FALSE	—	—
FALSE	TRUE	—	—
UNKNOWN	UNKNOWN	—	—

Fonte: Adaptado de Elmasri e Navathe (2011).

No Quadro 1, as linhas e colunas representam os valores dos resultados em condições de comparação que geralmente são utilizados na cláusula WHERE de uma consulta SQL. A combinação de dois valores utilizando o conectivo AND é mostrada nas primeiras quatro linhas. Em sequência, é apresentado o resultado do uso do conectivo OR. Por exemplo, o resultado de (TRUE AND FALSE) é FALSE, enquanto o resultado de (TRUE OR FALSE) é TRUE. Já no caso dos resultados da operação lógica NOT, apresentados ao final do primeiro grupo do Quadro 1, apenas há mudança no TRUE e no FALSE, enquanto o UNKNOWN permanece igual.

A SQL possibilitar realizar consultas que verificam se o valor de um atributo é NULL utilizando os operadores de comparação `IS` ou `IS NOT`. Isso se aplica porque se compreende cada valor nulo como diferente de outro valor nulo. Assim, a comparação da igualdade não se torna apropriada (ELMASRI; NAVATHE, 2011).



Link

Você viu, ao longo deste capítulo, diferentes conceitos e exemplos em SQL de consultas utilizando junção, agregação e ordenação. Acessando o *link* a seguir, você pode conferir conceitos adicionais sobre funções de agregação, que podem ser empregadas em diferentes versões de bancos de dados em SQL.

<https://qrqo.page.link/P115X>



Referências

ELMASRI, R.; NAVATHE, S. B. *Sistemas de banco de dados*. 6. ed. São Paulo: Pearson, 2011.

MANZANO, J. A. N. G. *Microsoft SQL Server 2016: express edition interativo*. São Paulo: Érica, 2017.

RAMAKRISHNAN, R.; GEHRKE, J. *Sistemas de gerenciamento de banco de dados*. 3. ed. Porto Alegre: AMGH, 2011.



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*.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS

Dica do Professor

As consultas em um banco de dados auxiliam na realização das tarefas tanto dos setores de Tecnologia de Informação, muitas vezes responsáveis pela garantia de sua integridade, quanto dos demais setores de uma empresa.

Nesta Dica do Professor, você vai verificar alguns exemplos de consultas avançadas em SQL.



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Exercícios

- 1) As funções de agregação permitem contar ou realizar operações de soma entre os valores presentes em uma tabela, entre outras opções.

Entre as opções a seguir, a que utiliza corretamente a função utilizada para verificar o valor máximo de um produto em uma consulta SQL é:

- A) `SELECT SUM(Valor) FROM PRODUTO;`
- B) `SELECT MIN(Salario) FROM FUNCIONARIO;`
- C) `SELECT MAX(Valor) FROM PRODUTO;`
- D) `SELECT AVG(Valor) FROM PRODUTO;`
- E) `SELECT COUNT(Valor) FROM PRODUTO;`

- 2) A cláusula **GROUP BY** é utilizada para formar grupos de acordo com atributos definidos na consulta, para que assim possa ser aplicada determinada função de agregação.

Entre as opções a seguir, assinale a que utiliza corretamente a cláusula **GROUP BY** e uma função de agregação para verificar a quantidade e os nomes dos produtos em estoque em uma consulta SQL:

- A) `SELECT NProduto SUM(Valor) FROM PRODUTOS GROUP BY NProduto.`
- B) `SELECT NProduto, SUM(Quantidade) FROM PRODUTOS GROUP BY NProduto;`
- C) `SELECT SUM(Valor) FROM PRODUTOS GROUP BY NProduto;`
- D) `SELECT SUM(Valor) GROUP BY Nproduto FROM PRODUTOS;`
- E) `SELECT NProduto, SUM FROM PRODUTOS GROUP BY NProduto;`

- 3) Ao realizar consultas em um banco de dados em SQL, diferentes cláusulas podem ser utilizadas para encontrar o resultado esperado.

Sobre as cláusulas em SQL em consultas, é correto afirmar que:

- A) Uma consulta pode ter até seis cláusulas, mas somente GROUP BY e ORDER BY são obrigatórias
 - B) uma consulta pode ter até quatro cláusulas, mas somente SELECT e GROUP BY são obrigatórias.
 - C) uma consulta pode ter até cinco cláusulas, mas somente SELECT e WHERE são obrigatórias.
 - D) Uma consulta pode ter até seis cláusulas, mas somente SELECT e FROM são obrigatórias.
 - E) uma consulta pode ter até sete cláusulas, mas somente FROM e HAVING são obrigatórias.
- 4) **Consultas em SQL podem ser escritas e executadas de diferentes formas. Por este motivo, existem vantagens e desvantagens dessa flexibilidade.**

Sobre as vantagens e/ou desvantagens da flexibilidade de especificação de consultas em SQL, é correto afirmar que:

- A) a disponibilidade de técnicas pode gerar desorganização do usuário, o que é uma vantagem.
- B) a possibilidade de a técnica escolhida pelo usuário ser menos eficiente é uma vantagem.
- C) o usuário poder escolher a técnica com a qual está acostumado é uma desvantagem.
- D) o usuário poder escolher a técnica com a qual está acostumado é uma vantagem.
- E) o programador do sistema pode receber a consulta organizada, o que é uma desvantagem.

- 5) **As consultas aninhadas têm subconsultas embutidas em seu texto.**

Analise as seguintes consultas e assinale a opção na qual uma subconsulta na cláusula WHERE em SQL é apresentada na sintaxe correta.

- A) SELECT F.Nome
FROM ALUNOS AS A
WHERE R.Cpf IN
(SELECT F.Cpf

FROM BY RESPONSAVEL AS R

WHERE R.Nome=F.Nome_responsavel);

B) SELECT SUM (Salario)

FROM (FUNCIONARIO JOIN DEPARTAMENTO ON Dnr=Dnumero)

WHERE D.Nome= 'Financeiro'

(SELECT D.Nome

FROM FUNCIONARIO AS A

WHERE D.Nome=F.Nome_dep;

C) SELECT A.Nome

FROM ALUNOS AS A

WHERE R.Cpf IN RESPONSAVEL AS R

WHERE R.Nome=A.Nome_responsavel;

D) SELECT A.Nome

FROM ALUNOS AS B

WHERE A.Cpf IN

(SELECT R.Cpf

FROM RESPONSAVEL AS A

WHERE R.Nome=A.Nome_responsavel);

E) SELECT A.Nome

FROM ALUNOS AS A

WHERE R.Cpf IN

(SELECT R.Cpf

FROM RESPONSAVEL AS R

WHERE R.Nome=A.Nome_responsavel);

Na prática

Consultas em SQL são úteis tanto para a manipulação quanto para realização de análises e relatórios sobre o que se tem armazenado, visto que as informações são um dos bens mais preciosos para as organizações nos dias de hoje, pois auxiliam no planejamento das ações e nas tomadas de decisão.

Nesta Na Prática, você vai conhecer o caso de uma empresa de saneamento que se utiliza de consultas com funções de agrupamento e agregação para realizar seu relatório orçamentário.

Conteúdo interativo disponível na plataforma de ensino!

Saiba mais

Para ampliar o seu conhecimento a respeito desse assunto, veja abaixo as sugestões do professor:

Conceitos de agregação e agrupamento

Confira no *link* a seguir a documentação oficial do MySQL sobre funções de agregação e agrupamento, que traz exemplos da sua aplicação em banco de dados.



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Funções agregadas

Para aprofundar seus conhecimentos sobre funções agregadas em SQL, assista ao vídeo a seguir, o qual aborda consultas na teoria e na prática em um banco de dados.



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Consultas avançadas com SQL

Para verificar mais exemplos sobre as consultas avançadas em SQL, consulte o seguinte livro. Dê atenção especial aos conceitos presentes entre as páginas 298 a 303, que tratam especificamente sobre junções internas e externas.

Conteúdo interativo disponível na plataforma de ensino!

Consultas aninhadas em SQL

Para verificar mais exemplos sobre as consultas aninhadas em SQL, consulte o seguinte livro. Dê atenção especial aos conceitos presentes entre as páginas 122 a 127, as quais tratam especificamente sobre consultas aninhadas.

Conteúdo interativo disponível na plataforma de ensino!