Implementação de Banco de Dados

Aula 5: LINGUAGEM SQL – Funções de Grupo

Apresentação

Na aula passada, você aprendeu a consultar os dados de uma tabela, a definir as colunas que retornam e a filtrar as linhas. Nesta aula, você aprenderá a gerar dados agregados utilizando funções de grupo e a ordenar a saída da consulta.

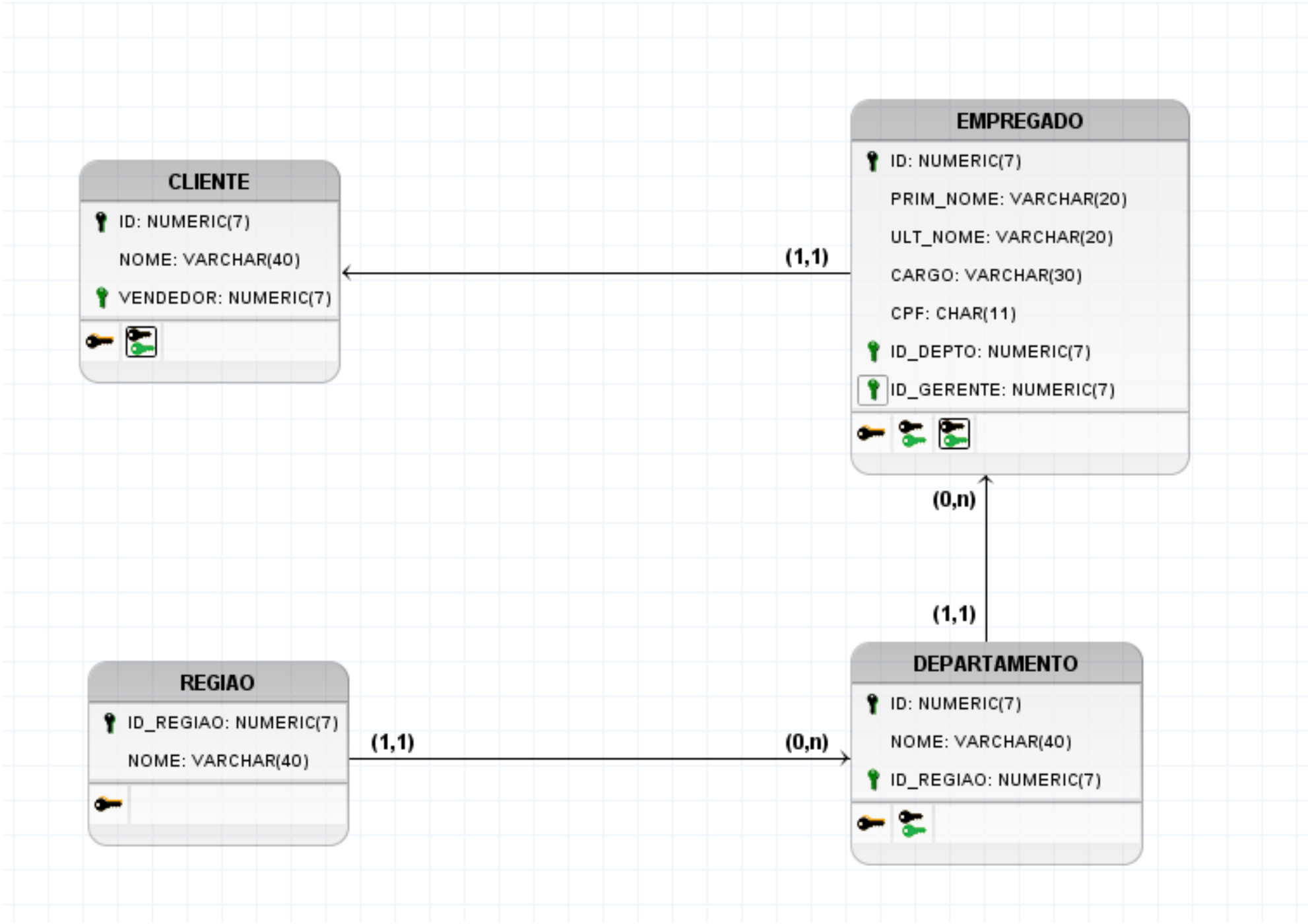
Objetivos

- Reproduzir funções de grupo;
- Ordenar o resultado das consultas.

Banco de dados de exemplo

Continuaremos utilizando o Banco de Dados da Empresa para os exemplos.

Modelo Lógico



As tabelas possuem os seguintes dados:

	id_regiao numeric (7)	nome character varying (40)
1	1	Norte
2	2	Sul

Região

	id numeric (7)	nome character varying (40)	id_regiao numeric (7)
1	10	Administrativo	1
2	20	Vendas	1
3	30	Compras	2

Departamento

	id numeric (7)	ult_nome character varying (20)	prim_nome character varying (20)	cargo character varying (30)	salario numeric (7,2)	dt_admissao date	cpf character (11)	id_depto numeric (7)	id_gerente numeric (7)
1	1	Velasques	Carmen	Presidente	29500.00	2009-05-05	34567890125	10	[null]
2	2	Neves	Lauro	Diretor de Compras	19500.00	2009-03-03	23456789012	30	1
3	3	Nogueira	Ernane	Diretor de Vendas	18000.00	2010-04-07	34567890123	20	1
4	4	Queiroz	Mark	Gerente de Compras	8000.00	2010-11-11	12345432123	30	2
5	5	Rodrigues	Alberto	Vendedor	4000.00	2008-10-10	87965432123	20	3
6	6	Ugarte	Marlene	Vendedor	3500.00	2009-03-03	87654345678	20	3

Empregado

	id numeric (7)	nome character varying (40)	vendedor numeric (7)
1	110	Ponto Quente	5
2	120	Casa Supimpa	6
3	130	Coisas e Tralhas	5
4	140	Casa Desconto	[null]

Cliente

Tendo este Banco em mente, é altamente recomendável que você execute os comandos de exemplo no PostGreSql.:

Comentário

Foi escolhido como base o PostgreSql, por ser um SGBD mais leve e fácil de instalar, porém, se for possível usar o SqlServer ou o Oracle quando houver diferença entre os SGBD's, você será informado.

Eliminando valores duplicados (DISTINCT)

Analise o conteúdo da tabela Empregados

	id numeric (7)	ult_nome character varying (20)	prim_nome character varying (20)	cargo character varying (30)	salario numeric (7,2)	dt_admissao date	cpf character (11)	id_depto numeric (7)	id_gerente numeric (7)
1	1	Velasques	Carmen	Presidente	29500.00	2009-05-05	34567890125	10	[null]
2	2	Neves	Lauro	Diretor de Compras	19500.00	2009-03-03	23456789012	30	1
3	3	Nogueira	Ernane	Diretor de Vendas	18000.00	2010-04-07	34567890123	20	1
4	4	Queiroz	Mark	Gerente de Compras	8000.00	2010-11-11	12345432123	30	2
5	5	Rodrigues	Alberto	Vendedor	4000.00	2008-10-10	87965432123	20	3
6	6	Ugarte	Marlene	Vendedor	3500.00	2009-03-03	87654345678	20	3

Observe que na tabela Empregados, RODRIGUES e UGARTE possuem o mesmo cargo. E se desejássemos ver os diferentes cargos? Se comandássemos SELECT CARGO FROM EMPREGADO, teríamos o resultado desejado?

Em termos de dados, até poderíamos dizer que sim, que todos os cargos aparecem no resultado. Mas, em termos de facilidade para o usuário, isso é suficiente?

Imagine que a tabela tivesse milhares de linhas com dezenas de cargos diferentes.

O usuário ficaria confuso, pois teria muita dificuldade de isolar todos os cargos existentes ali. Para resolver esse tipo de caso, podemos, no comando, eliminar os valores duplicados, retornando apenas uma vez cada cargo.

Para tal, devemos acrescentar a cláusula distinct ao comando de Select. Dessa forma o comando seria:

```
SELECT DISTINCT CARGO

FROM EMPREGADO
```

Agora conseguimos o resultado que queríamos. Observe agora o seguinte comando:

```
SELECT DISTINCT CARGO, ULT_NOME

FROM EMPREGADO
```

Note que voltaram os dois vendedores. Por que isso ocorreu se o DISTINCT continua antes do cargo? O Distinct, na realidade, filtra as linhas diferentes, não os valores da coluna. Como os dois vendedores possuem nomes diferentes, as linhas são distintas, portanto as duas retornam.

Atividade

Utilizando o nosso banco de dados de exemplo para fazer alguns exercícios.

1. Mostrar uma única vez os cargos dos empregados como o cabeçalho Cargos Diferentes. O retorno esperado é exibido na figura.

Retorno do comando

Agregando dados

Até agora todos os comandos que demos retornavam uma linha para cada linha da tabela ou uma linha para cada linha da tabela que atendesse a condição da cláusula Where. Veja os exemplos abaixo.

No primeiro comando, retornaram três linhas, o total de linhas da tabela.

No segundo comando, duas linhas, o total de linhas que atendem a condição da cláusula Where.

Observe agora os próximos dois comandos.

Note que, com os dois comandos, retornou apenas uma linha.

No primeiro, com valor três (total de linhas da tabela) e no segundo, com valor dois (quantidade de linhas que atendem a cláusula Where).

O que mudou? Foi acrescentado COUNT na claúsula Select que, como você já deve ter percebido, contou as linhas que a consulta retornaria e exibiu esse valor. O que fizemos foi agregar dados, ou seja, derivamos um dado sumarizado a partir dos dados da tabela.

Saiba mais

Para fazer isso, você deverá utilizar as FUNÇÕES DE GRUPO, das quais COUNT é um exemplo.

Funções de grupos

Uma função de grupo atua em uma instância da tabela, ou seja, no conjunto de suas linhas.

As funções agregam os dados a partir de todas a linhas da tabela ou de grupos em que as linhas possam ser enquadradas.

A princípio, a tabela forma um único grupo, e a consulta com função de grupo retornará uma única linha.

Ao utilizarmos a cláusula GROUP BY, podemos dividir a tabela em grupo, sendo que a consulta, então, retornará uma linha para cada grupo. Na linguagem SQL, possuímos as seguintes funções de grupo:

Clique nos botões para ver as informações.

AVG (coluna)

- Retorna a MÉDIA aritmética dos valores da coluna solicitada;
- Exemplo: AVG (valor_segurado);
- Se na coluna existirem nulos, ela os desconsidera no cálculo,
- Os valores na coluna devem ser numéricos.

MAX (coluna)

- Retorna o MAIOR valor existente na coluna solicitada;
- Exemplo: MAX (CPF);
- Se na coluna existirem nulos, ela os desconsidera.
- Os valores na coluna podem ser numéricos, alfanuméricos ou datas

MIN (coluna)

- Retorna o MENOR valor existente na coluna solicitada;
- Exemplo: MIN (CPF);
- Se na coluna existirem nulos, ela os desconsidera;
- Os valores na coluna podem ser numéricos, alfanuméricos ou datas

SUM (coluna)

- Retorna a soma aritmética dos valores da coluna solicitada;
- Exemplo: SUM (valor_segurado);
- Se na coluna existirem nulos, ela os desconsidera no cálculo;
- Os valores na coluna devem ser numéricos.

COUNT (coluna)

- Retorna o número de valores não nulos da coluna solicitada;
- Exemplo: COUNT (CPF);
- Os valores na coluna podem ser numéricos, alfanuméricos ou datas

COUNT (*)

- Retorna o número de linhas que a consulta retornaria;
- Exemplo: COUNT (*);
- Considera os valores nulos.

Vejamos alguns exemplos.

Acesse o PostgreSQL e digite o seguinte comando: SELECT * FROM EMPREGADO.

Esse comando seleciona dados de todos os empregados

Vamos supor que você deseja recuperar apenas o valor médio dos salários e a soma dos salários. Para essa consulta, você comandaria:

SELECT AVG(SALARIO), SUM(SALARIO)

FROM EMPREGADO

Atenção

Apesar de já ter sido dito, lembre-se sempre de que as funções AVG e SUM são numéricas, ou seja, exigem que os seus argumentos (colunas que passamos a função; no exemplo, a coluna SALÁRIO) sejam números.

Vejamos mais exemplos.

Agora você deseja listar o maior e o menor valor de salário de um empregado, os sobrenomes que aparecem como último e como primeiro na ordem alfabética crescente e as datas do empregado contratado há mais tempo e há menos tempo. Para isso basta comandar:

Observe que:

- Podem ser utilizadas várias funções de grupo em paralelo, na mesma coluna ou em colunas distintas;
- Quando o argumento da função é alfanumérico (como a coluna ult_nome), o valor que aparece por último na ordem alfabética crescente é o maior e o que aperece primeiro é o menor;
- Quando o argumento da função é numérico (como a coluna salário), menor e maior se referem à posição dos números no sistema de numeração, respeitado o fato de serem positivos ou negativos;
- Quando o argumento da função é data (como a coluna dt_admissao), menor e maior se referem à posição da data na linha do tempo, ou seja, a menor aparece primeiro na linha do tempo e a maior aparece por último.

Mais um exemplo:

Acesse o PostgreSql e digite o seguinte comando: SELECT * FROM EMPREGADO.

Agora você deseja saber a quantidade de empregados e a quantidade de empregados que possuem gerentes. Como seria o comando?

O comando seria:

SELECT COUNT(*), COUNT(ID), COUNT(ID_GERENTE)

FROM EMPREGADO

E obteríamos um retorno similar ao da figura abaixo.

Observe que:

- Count(*) conta a quantidade de linhas retornadas, independentemente de seu conteúdo;
- Count(coluna) conta a quantidade de linhas não nulas naquela coluna;
- Se compararmos o resultado de count(id) com count(id_gerente), podemos notar que a primeira expressão retornou o valor 6, total de linhas da tabela já que ID é chave primária, não podendo possuir valor nulo. Já a segunda retornou 5, pois o empregado de ID 1 não possui gerente, de forma que somente cinco linhas possuem valor não nulo nesta coluna.

Se acrescentar a função de grupo DISTINCT, essa passará a ignorar os valores duplicados, computando cada um deles apenas uma vez.

Observe a figura abaixo que nos mostra o conteúdo das colunas CARGO e ID_DEPTO na tabela EMPREGADO.

Podemos observar que, na coluna cargo, dentre seis linhas há duas com o valor vendedor, e na coluna id_depto, dentre seis linhas há quatro com o valor 20.

Se desejássemos listar a quantidade de cargo e quantidade de cargos diferentes a quantidade de departamentos e a quantidade de departamentos diferentes, qual seria o comando?

O comando seria:

SELECT COUNT(CARGO), COUNT(DISTINCT CARGO),

COUNT(ID_DEPTO), COUNT(DISTINCT ID_DEPTO)

FROM EMPREGADO

Obteríamos um retorno similar ao da figura abaixo.

Observe o seguinte:

- Quando executamos o comando sem o distinct, (count(cargo) ou count(id_depto), são contadas todas as linhas não nulas;
- Quando executamos o comando com o distinct, (count(distinct cargo) ou count(distinct id_depto), cada valor discreto existente na coluna é computado apenas uma vez.

Atividade

Utilizando o nosso banco de dados de exemplo para fazer alguns exercícios.

2. Mostrar salário médio, maior salário, menor salário e total dos salários de todos os empregados. O retorno esperado é exibido na figura.

Retorno do comando

3. Mostrar o sobrenome que é o primeiro em ordem alfabética com o alias.

Primeiro Ordem Alfabética.

O retorno esperado é exibido na figura.

Retorno do comando

4. Mostrar o sobrenome que é o ÚLTIMO, em ordem alfabética com o alias.

– Último Ordem Alfabética.

O retorno esperado é exibido na figura.

Retorno do comando

5. Mostrar a data de admissão do empregado que possui mais tempo de casa.

O retorno esperado é exibido na figura.

Retorno do comando

Criando grupos nas tabelas

Podemos agrupar os dados de nossa tabela utilizando a cláusula GROUP BY.

Sintaxe:

SELECT nome da coluna [, nome da coluna]

FROM nome da tabela

WHERE condição

GROUP BY expressão

Dica

“Expressão” determina a(s) coluna(s) cujo(s) valor(es) vai servir de parâmetro para o agrupamento das linhas. Sobre os grupos criados serão aplicadas as funções de grupo, para gerar a sumarização desejada.

Por exemplo: se você desejasse saber a quantidade de empregados, a média salarial e o valor da folha salarial de cada departamento da empresa, qual seria o comando?

O comando seria:

SELECT COUNT(*), AVG(SALARIO), SUM(SALARIO), ID_DEPTO

FROM EMPREGADO

GROUP BY ID_DEPTO

Obteríamos um retorno similar ao da figura abaixo.

Observe o seguinte:

- Na cláusula Group By, você deve colocar a coluna cujos valores serão utilizados para criar os grupos, sendo que pertencem ao mesmo grupo todas as linhas que possuem valores iguais na coluna;
- Você também deve colocar na cláusula Select a coluna ou colunas utilizadas para criar os grupos, permitindo dessa forma que se saiba a qual grupo pertence o valor gerado pela função de grupo.

Saiba mais

No SELECT, só é permitido colocar COLUNAS que foram utilizadas no GROUP BY ou nas FUNÇÕES DE GRUPO.

Observe o exemplo da figura abaixo, similar ao comando anterior, com a diferença de que não temos a cláusula GROUP BY, gerando um erro com a presença da coluna id_depto no select.

Grupos dentro de grupos.

Clique no botão acima.

A cláusula GROUP BY permite que você crie grupos definidos por mais de uma coluna, ou seja, a agregação não será por uma coluna, mas por um conjunto de colunas.

Para recuperar a quantidade de empregados que temos em cada cargo em cada departamento, poderíamos dar o comando mostrado na figura a seguir.

Atividade

Utilizando o nosso banco de dados de exemplo para fazer alguns exercícios.

6. Mostrar a quantidade de empregados de cada cargo. O retorno esperado é exibido na figura.

Retorno do comando

7. Mostrar o ID do gerente e a quantidade de empregados que ele gerencia. O retorno esperado é exibido na figura.

Retorno do comando

8. Mostrar o ID do vendedor e a quantidade de clientes que ele atende. O retorno esperado é exibido na figura.

Retorno do comando

Filtrando dados agrupados

Ao utilizar a cláusula GROUP BY, pode-se filtrar a seleção de dados antes deles serem agrupados OU/E após o agrupamento. O momento de realizar essa restrição depende do desejo de eliminar linhas da tabela antes de aplicar as funções de grupo, criando uma condição na cláusula WHERE, ou se o seu intuito é eliminar GRUPOS que não atendam a uma determinada condição, utilizando nesse caso a cláusula HAVING.

Vejamos alguns exemplos:

Você deseja filtrar a consulta por dados que existem na tabela. Por exemplo, quer contar a quantidade de empregados, o salário médio e a soma dos salários dos empregados do departamento 20. Nesse caso devemos filtrar os dados antes de agrupá-los utilizando WHERE, já que a filtragem será realizada por um dado que existe na tabela, o ID_DEPTO.

```
SELECT  COUNT(*), AVG(SALARIO), SUM(SALARIO)

FROM EMPREGADO

WHERE ID_DEPTO = 20
```

E obteríamos um retorno similar ao da figura.

Em resumo, temos que:

- Quando se deseja filtrar antes do agrupamento, utiliza-se a cláusula WHERE para eliminar as linhas da tabela que não nos interessa agrupar, de forma similar ao que fazemos no comando select, quando não desejamos que as linhas retornem no resultado da consulta;
- A condição estabelecida no WHERE, as linhas que não a atendem, no caso do exemplo as linhas que não são do ID_DEPTO 20, não retornam. Dessa forma somente as linhas do departamento 20 estarão no conjunto a ser feito com a agregação de dados;
- Após a filtragem das linhas da tabela é que os dados são agrupados e as funções aplicadas, retornando, portanto, apenas os valores referentes ao departamento desejado.

A cláusula HAVING tem função semelhante à cláusula WHERE, mas é aplicada aos resultados das funções de grupo geradas para cada grupo, não nas linhas das tabelas. Em outras palavras, após gerarmos os resultados dos grupos, podemos filtrá-los retornando apenas aqueles que atendam às condições da Cláusula HAVING.

Considere agora o seguinte comando:

```
SELECT  ID_DEPTO,COUNT(*), AVG(SALARIO), SUM(SALARIO)

FROM EMPREGADO

GROUP BY ID_DEPTO
```

Note que os departamentos 20 e 30 possuem mais de um empregado. Se desejássemos listar os departamentos que possuem mais de um empregado, como seria o comando?

A primeira ideia que poderíamos ter seria utilizar a cláusula WHERE como no seguinte comando:

```
SELECT  COUNT(*), AVG(SALARIO), SUM(SALARIO), ID_DEPTO

FROM EMPREGADO

WHERE COUNT(*) > 1

GROUP BY ID_DEPTO
```

Esse comando gera um erro, conforme podemos ver na figura abaixo:

Não podemos utilizar funções de grupo desta forma na cláusula WHERE, somente em subconsulta, como veremos mais à frente no curso. Além disso, devemos lembrar que, na tabela original, não existe a contagem de linhas e a cláusula WHERE trabalha nos dados existentes na tabela.

Qual seria o comando então? Veja na figura abaixo.

Note o uso da cláusula HAVING para criar condições às quais os grupos devem obedecer para poderem retornar. Podemos dizer que a cláusula HAVING é similar à cláusula WHERE somente se aplicada aos dados gerados pelas funções de grupo.

Observe que você pode apenas usar WHERE para restringir linhas individuais. Para restringir grupos, usa-se a cláusula HAVING.

Funcionamento do Comando Group By.

Clique no botão acima.

Um comando com cláusulas WHERE e HAVING funciona obedecendo aos seguintes passos:

- Primeiro são selecionadas as linhas da tabela que satisfazem a condição da cláusula WHERE (se não houver, todas a linhas são selecionadas);
- As linhas são agrupadas;
- A função de grupo é aplicada ao Grupo;
- Os grupos que satisfazem a condição do HAVING são exibidos (se não houver, todos os grupos serão exibidos).

Atividade

Utilizando o nosso banco de dados de exemplo para fazer alguns exercícios.

9. Mostrar a quantidade de empregados, salário médio, maior salário, menor salário e total dos salários para os empregados do departamento 30. O retorno esperado é exibido na figura.

Retorno do comando

10. Mostrar o ID dos vendedores e quantidade de clientes para vendedores que atendem mais de um cliente. O retorno esperado é exibido na figura.

Retorno do comando

11. Mostrar a quantidade de empregados de cada cargo do departamento 20 para os cargos que possuem mais de um empregado. O retorno esperado é exibido na figura.

Retorno do comando

Ordenando consultas

Até agora, em todos os comandos que você viu, as linhas retornaram na ordem em que foram produzidas. Porém, muitas vezes, pode ser necessário que elas sejam ordenadas. Para permitir isso, o SQL possui a cláusula ORDER BY.

O resultado de uma consulta pode ser ordenado pelo valor de uma ou mais colunas, de forma crescente ou decrescente.

A sintaxe básica do ORDER By é:

SELECT { * | nome da coluna [, nome da coluna ...]}

FROM nome da tabela

WHERE condição {AND | OR} condição

ORDER BY nome da coluna [ASC | DESC] [, nome da coluna [ASC | DESC] .]

onde

ASC ordena as linhas de forma ascendente; é a ordenação default.

DESC ordena as linhas de forma descendente.

Saiba mais

A cláusula ORDER BY será sempre a última de um comando SQL.

Vejamos um exemplo: desejamos listar o ID, o sobrenome, a data de admissão, o cargo e o salário de todos os empregados em ordem crescente de salário.

Note que:

- O resultado da consulta voltou ordenado na forma solicitada. No exemplo, a ordenação definida na cláusula ORDER BY fez com que as linhas retornadas da tabela EMPREGADO sejam ordenadas por SALÁRIO e só então exibidas;
- O padrão default de ordenação da cláusula ORDER BY é crescente. Dessa forma, o uso de ASC para indicar a ordem crescente (ascendente) é opcional. Se for omitida, a ordenação ascendente será realizada, como foi o caso do exemplo.

Ordenação decrescente

Para a ordenação decrescente, basta utilizarmos a opção DESC (descendente) no lugar de ASC.

Ordenação múltipla

Podemos realizar a ordenação por várias colunas na mesma consulta. Podemos ordenar o resultado por cargo e, a partir da primeira ordenação, dentro de cada cargo por salário. Veja o exemplo abaixo.

Observe que o resultado é inicialmente ordenado pela CARGO de forma DESCENDENTE (VENDEDOR, PRESIDENTE, ETC.) e ao termos uma CARGO repetido (observe o VENDEDOR) as linhas de mesma CARGO são ordenadas entre si por SALARIO de forma ascendente.

Atenção

No comando, usamos o argumento ASC, mas como é forma padrão de ordenação, poderíamos omiti-lo.

Outra forma de comandarmos a ordenação é pela posição da coluna no resultado. Com isso o comando anterior poderia ser escrito da seguinte forma:

```
SELECT  *

FROM VEICULO

ORDER BY 2  ASC ,5 DESC
```

Onde os números 4 e 5 referenciam a quarta coluna (CARGO) e a quinta (SALÁRIO), produzindo, dessa forma, o mesmo resultado na consulta.

Atividades

Clique no botão acima.

Atividades

Utilizando o nosso banco de dados de exemplo para fazer alguns exercícios.

1. Mostrar ID, sobrenome, cargo e salário dos empregados ordenados pelo salário em ordem crescente (ordenar utilizando o nome da coluna). O retorno esperado é exibido na figura.

Retorno do comando

2. Mostrar ID, sobrenome, cargo e salário dos empregados ordenados pelo cargo em ordem decrescente (ordenar utilizando o nome da coluna). O retorno esperado é exibido na figura abaixo.

Retorno do comando

3. Mostrar ID, sobrenome, cargo e salário dos empregados ordenados pelo cargo em ordem decrescente dentro do cargo por salário em ordem crescente (ordenar utilizando o nome da coluna). O retorno esperado é exibido na figura.

Retorno do comando

4. Mostrar ID, sobrenome, cargo e salário dos empregados ordenados pelo cargo em ordem decrescente dentro do cargo por salário em ordem crescente (ordenar utilizando a posição da coluna). O retorno esperado é exibido na figura.

Retorno do comando

5. Mostrar ID, sobrenome, cargo e salário dos empregados ordenados pelo cargo em ordem decrescente dentro do cargo por salário em ordem crescente (ordenar utilizando a posição da coluna) para os empregados com ID maior que 3. O retorno esperado é exibido na figura.

Retorno do comando

Para as atividades abaixo, utilize o banco de dados da ESCOLA. Se você não fez a criação do banco da Escola, faça agora antes de continuar.

MODELO DE DADOS DA ESCOLA:

CONCEITUAL

LÓGICO

6. Dê os comandos abaixo filtrando dados das tabelas.

a) Selecione todos os dados de curso.

b) Selecione todos os dados do aluno de cpf 72718051337.

c). Selecione o nome dos alunos cujo CPF inicie com '09'.

d) Selecione o nome das disciplinas do curso de código 102.

e) Selecione o código das disciplinas cuja carga horária seja menor que 80 horas.

7. Dê os comandos abaixo utilizando funções de grupo.

a) Calcule a carga horária média de todas as disciplinas.

b) Calcule a média dos alunos aprovados.

c) Mostre a maior média de uma aluno reprovado por nota.

d) Mostre quantas reprovações por falta existem.

e) Mostre quantos alunos diferentes foram reprovados por falta.

f) Mostre quantos alunos foram reprovados.

g) Qual a menor média de um aluno aprovado em upo1003?

h) Quantos alunos cursaram upo1002 ou upo1001?

i) Quantos alunos tiveram média entre 7 e 8, inclusive?

8. Dê os comandos abaixo utilizando funções de grupo e ordenando o resultado conforme solicitado.

a) Mostre o nome dos alunos em ordem alfabética decrescente. Para os alunos que possuem a matrícula terminada em 27, comande a ordenação pelo nome da coluna.

b) Mostre o nome dos alunos em ordem alfabética decrescente. Para os alunos que possuem a matrícula terminada em 27, comande a ordenação pela posição da coluna.

c) Mostre o nome da disciplina, o seu código e a sua carga. A consulta deverá retornar na ordem crescente de carga horária e decrescente de nome da disciplina para as que possuem a mesma carga horária. Comande a ordenação pelo nome da coluna.

d). Mostre o nome da disciplina, o seu código e a sua carga. A consulta deverá retornar na ordem crescente de carga horária e decrescente de nome da disciplina para as que possuem a mesma carga horária. Comande a ordenação pela posição da coluna.

9. Dê os comandos abaixo criando grupos.

a) Mostre o código do curso e o total de disciplinas que cada um possui.

b) Mostre quantas vezes foram cursadas as disciplinas upo1010, upo1011, upo1012 ou upo1013.

c) Mostre o código das disciplinas e quantas vezes ela foi cursada para a disciplina cursada por menos de mil alunos.

d) Mostre o código da disciplina e quantas vezes ela foi cursada com aprovação para a disciplina que teve mais de 450 aprovações.

e) Mostre o código das disciplinas e quantas vezes elas tiveram reprovação por falta para as disciplinas com mais de 320 reprovações por falta. A consulta deverá retornar na ordem crescente de quantidade de reprovações e decrescente do código da disciplina. Comande a ordenação pelo nome da coluna.

f) Mostre o código das disciplinas e quantas vezes elas tiveram reprovação por falta para as disciplinas com mais de 320 reprovações por falta. A consulta deverá retornar na ordem crescente de quantidade de reprovações e decrescente do código da disciplina. Comande a ordenação pela posição da coluna.

10. Utilizando o banco de dados da Seguradora, emita os comandos abaixo utilizando funções de Grupo e Group By.

a) Recuperar apenas o valor médio segurado e a soma dos valores segurados;

b) Listar o maior e o menor valor segurado de um veículo e as cores que aparecem como última e como primeira na ordem alfabética crescente;

c) Listar a quantidade total de proprietários e a quantidade de proprietários que possuem CPF;

d) Recuperar quantas cores diferentes existem de veículos;

e) Recuperar a quantidade de veículos e o valor médio segurado dos veículos de cada cor;

f) Recuperar a quantidade de veículos segurados de cada modelo e de cada cor;

g)Listar a quantidade de veículos das cores vermelho e preto;

h) Retornar apenas as cores que possuem um único veículo.

a) Recuperar apenas o valor médio segurado e a soma dos valores segurados.

b) Listar o maior e o menor valor segurado de um veículo e as cores que aparecem como última e como primeira na ordem alfabética crescente.

c) Listar a quantidade total de proprietários e a quantidade de proprietários que possuem CPF.

d) Recuperar quantas cores diferentes existem de veículos.

e) Recuperar a quantidade de veículos e o valor médio segurado dos veículos de cada cor.

f) Recuperar a quantidade de veículos segurados de cada modelo e de cada cor.

g) Listar a quantidade de veículos das cores vermelho e preto.

h) Retornar apenas as cores que possuem um único veículo.

Notas

Case sensitive

Você deve considerar que o SqlServer, a princípio, não faz distinção entre maiúsculas e minúsculas, enquanto o Oracle e o PostGreSql fazem. Na realidade, o SqlServer pode ou não fazer. É uma configuração que o DBA faz no servidor, mas o padrão é não fazer tal distinção.

Referências

DATE, C. J. **Introdução a sistemas de banco de dados**. 7. ed. Rio de Janeiro: Campus, 2000.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. 7. ed. São Paulo: Pearson Addison Wesley, 2015.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistemas de banco de dados**. 5. ed. Rio de Janeiro: Campus, 2006.

Próxima aula

- Aplicação de comandos de junção.

- Funções de agregação
- Criando funções de agregação no PostgreSQL - Muito além de COUNT(), SUM() e AVG()
- SQL: Funções de agregação