BANCO DE DADOS

Funções Agregadas

A característica marcante das funções agregadas é que produzem um único valor a partir de uma coluna inteira de dados. Portanto, enquanto qualquer outro tipo de expressão retorna um valor para cada linha, as funções agregadas retornam um valor que representa um agregado dos valores referentes às várias linhas. Por esta razão, são também chamadas de funções de colunas.

Existem cinco funções agregadas. São elas:

- **AVG** (argumento) – Retorna a média dos valores do argumento
- **MAX** (argumento) – Retorna o maior valor do argumento
- **MIN** (argumento) – Retorna o menor valor do argumento
- **SUM** (argumento) – Retorna o somatório dos valores do argumento
- **COUNT** (argumento) – Retorna a número de linhas do argumento

As funções agregadas normalmente usam como argumento um nome de coluna ou uma expressão que tenha um nome de coluna como componente, mas podemos usá-las com qualquer expressão numérica ou de datas.
A consulta a seguir lê todos os valores da coluna DESCONTO e fornece os percentuais médio, máximo e mínimo de desconto oferecidos aos hóspedes do **Visual Spa**. Como a função **AVG**, **MAX**, **MIN** e **SUM** ignoram valores nulos, o valor **AVG** (média) é realmente o desconto médio apenas daqueles hóspedes que obtiveram algum desconto:

```
SELECT AVG(desconto), MAX(desconto), MIN(desconto) FROM quarto
```

Outro exemplo:

```
SELECT MIN(desconto) * AVG(desconto) FROM quarto
```

Vejamos alguns exemplos:

- Mostrar o menor e o maior desconto para o quarto:

```
SELECT MIN(desconto), MAX(desconto) FROM quarto
```

- Mostrar a quantidade total de descontos para um determinado quarto:

```
SELECT SUM(desconto) FROM quarto WHERE nome = 'MIGUEL'
```

- Qual a média dos descontos dos quartos:

```
SELECT AVG(desconto) FROM quarto
```

- Quantos quartos apresentam um desconto superior a 10%:

```
SELECT COUNT(*) FROM quarto WHERE desconto > 0.10
```

Agregação com a cláusula GROUP BY

A cláusula **GROUP BY** reúne diferentes linhas do resultado de uma consulta em conjuntos de acordo com as colunas nela mencionadas, chamadas colunas formadoras de grupos. As linhas são "agrupadas" de duas formas, ou em dois aspectos.

A consulta abaixo exemplifica a primeira forma, na qual todas as linhas que contêm o mesmo valor na primeira coluna especificada são exibidas em grupos no resultado. Neste caso, a coluna formadora de grupo é QUARTO, e todas as linhas que tenham o mesmo valor aparecerão juntas no resultado:

```
SELECT * FROM quarto GROUP BY quarto
```

Caso existam linhas em branco para a coluna QUARTO, ou seja, com um valor nulo, também seriam agrupadas. E as linhas são agrupadas da mesma maneira para cada coluna formadora de grupos subsequente, embora isto não esteja aparente no exemplo dados pois só contém duas colunas. Existem as seguintes funções agregadas — **AVG**, **SUM**, **MAX**, **MIN** e **COUNT** — que devem ser usadas com a cláusula **GROUP BY** pois geram um único valor.

A cláusula **GROUP BY** também pode ser usada em consultas contendo uma cláusula **WHERE**. Neste caso, a **GROUP BY** é codificada depois da cláusula **WHERE**. Por exemplo, a consulta abaixo exibe quantos hóspedes ocuparam os quartos depois do dia 01 de janeiro por quarto:

```
SELECT SUM(quarto) FROM quarto WHERE chegada > '2010-01-01' GROUP BY quarto
```

Agregação com a cláusula HAVING

A cláusula **HAVING** nos permite estreitar a área de atuação da cláusula **GROUP BY** da mesma forma que a cláusula **WHERE** estreita a área de atuação da cláusula **SELECT**, ou seja, através de uma condição de pesquisa.

Com o exemplo abaixo, criaremos uma nova tabela, PESO que será usada nos exemplos deste tópico. Criamos uma tabela através da declaração CREATE TABLE abaixo. Esta tabela se destina a registrar as pesagens periódicas dos hóspedes do **Visual Spa**.

```
CREATE TABLE peso (
 nome VARCHAR(25) NOT NULL,
 quando DATE,
 peso DECIMAL(4,1),
 PRIMARY KEY (nome, quando))
```

Após a inclusão dos dados, a tabela PESO terá a seguinte aparência:

NOME	QUANDO	PESO
MIGUEL	01-01-2010	66.4
MIGUEL	02-01-2010	66.2
RAQUEL	01-01-2010	86.5
RAQUEL	02-01-2010	86.1
RAQUEL	03-01-2010	85.8
JOANA	01-01-2010	66.7
JOANA	02-01-2010	66.3
LUCIANA	01-01-2010	67.5

A seguinte tabela contém os registros de peso dos hóspedes por dia. A consulta a seguir indica os hóspedes que tiveram uma diferença entre seu peso mínimo e seu peso máximo.

```
SELECT nome, MAX(peso) - MIN(peso) diferenca FROM peso
GROUP BY nome
HAVING MAX(peso) - MIN(peso) > 0.0
```

Passemos para um outro exemplo. Esta consulta mostra os hospedes que realizaram mais de duas pesagens:

```
SELECT nome, COUNT(nome) FROM peso GROUP BY nome HAVING COUNT(nome) > 2
```

Ordenando os Dados com a cláusula ORDER BY

A cláusula **ORDER BY** permite classificar as linhas do resultado alfabética e numericamente, em ordem crescente ou decrescente. O padrão é a ordem crescente. A cláusula **ORDER BY** deve ser sempre a última cláusula da consulta.

No exemplo a seguir, obtemos os nomes dos hóspedes classificados em ordem decrescente através da palavra-chave **DESC** na cláusula **ORDER BY** depois do nome da coluna a ser ordenada. Usamos a palavra-chave **DISTINCT** para suprimir as linhas duplicadas:

```
SELECT DISTINCT nome FROM quarto ORDER BY nome DESC
```

As classificações são, por padrão, efetuadas em ordem crescente, a não ser que seja indicada a palavra-chave **DESC** após o nome da coluna. Entretanto, podemos explicitar a ordem crescente para uma determinada coluna, usando-se a palavra-chave **ASC** após o nome da coluna na cláusula **ORDER BY**.

Produto Cartesiano

Podemos utilizar os seguintes Operadores Lógicos:

- **AND** para realizar uma conjunção de instruções
- **OR** para realizar uma disjunção de instruções

- **NOT** para realizar uma negação de instruções

Por exemplo. Quais são os hóspedes que têm o biótipo igual a 1 e o gênero igual a 'M'

```
SELECT nome FROM hospede WHERE biotipo = 1 AND genero = 'M'
```

Outro exemplo. Quais são os hóspedes que têm o biótipo igual a 1 ou o gênero igual a 'M'

```
SELECT nome FROM hospede WHERE biotipo = 1 OR genero = 'M'
```

Outro exemplo. Quais são os hóspedes que não têm o gênero igual a 'M'

```
SELECT nome FROM hospede WHERE NOT (genero = 'M')
```

Composição (JOIN)

Este é o comando mais comum utilizado e sua característica é só trazer registros que contenham correspondência nas outras tabelas utilizadas. Podemos listar todos os hóspedes e os quartos aos quais estes pertencem, com o seguinte código:

```
SELECT h.nome, q.quarto FROM hospede h
      INNER JOIN quarto q ON h.nome = q.nome
```

Outro modo que podemos realizar é mostrar todos os hóspedes, mesmo que não existam quartos associados a esta:

```
SELECT h.nome, q.quarto FROM hospede h
      LEFT JOIN quarto q ON h.nome = q.nome
```

Ou o inverso, queremos ver todos os quartos, mesmo que estes não contenham nenhum hóspede:

```
SELECT h.nome, q.quarto FROM hospede h
      RIGHT JOIN quarto q ON h.nome = q.nome
```

Concluímos então que as instruções **LEFT JOIN** e **RIGHT JOIN** são utilizadas para resolver a não correspondência de dados entre diferentes tabelas.
O **NATURAL JOIN** e o **STRAIGHT_JOIN** fazem exatamente a mesma coisa que o **INNER JOIN** em questão de resultado, porém com algumas particularidades:

- **NATURAL JOIN:** não será necessário identificar quais colunas serão comparadas, pois será realizada uma comparação entre os campos com mesmo nome.

```
SELECT h.nome, h.altura, p.peso FROM hospede h NATURAL JOIN peso p
```

- **STRAIGHT_JOIN:** faz com que a tabela a esquerda seja lida primeiro, isso é utilizado quando o otimizador do **JOIN** coloca as tabelas em ordem errada. Isto é muito pouco utilizado.

```
SELECT h.nome, h.altura, p.peso FROM hospede h
   STRAIGHT_JOIN peso p ON h.nome = p.nome
```

Observação:

1. É possível substituir o ON por USING quando o nome das colunas nas duas tabelas for idêntico.

```
SELECT h.nome, h.altura, p.peso FROM hospede h INNER JOIN peso p USING(nome)
```

2. O uso do INNER é opcional.

```
SELECT h.nome, h.altura, p.peso FROM hospede h JOIN peso p USING(nome)
```