# Sistemas de Gerenciamento de Banco de Dados

Tradução da
Terceira Edição

Mc Graw Hill

Ramakrishnan • Gehrke

**Sistemas de Gerenciamento de Banco de Dados**
ISBN 978-85-7726-027-0



Tradução da terceira edição do original em inglês Database Management Systems.
© 2003, 2000, 1998 de The McGraw-Hill Companies, Inc.
ISBN da obra original: 0-07-246563-8

Diretor-Geral: *Adilson Pereira*
Editora: *Gisélia Costa*
Supervisora de Produção: *Guacira Simonelli*
Preparação de Texto: *Lucrécia Freitas e Mônica de Aguiar*
Design da Capa: *Mick Wiggins*
Editoração Eletrônica: *Crontec Ltda*.

---

R165s Ramarkrishnan, Raghu.
Sistemas de gerenciamento de banco de dados [recurso eletrônico] / Raghu Ramarkrishnan,Johannes Gehrke ; tradução: Célia Taniwake. – 3. ed. – Dados eletrônicos. – Porto Alegre : AMGH, 2011.

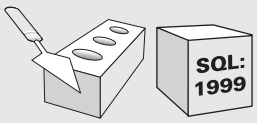
Editado também como livro impresso em 2008.
ISBN 978-85-63308-77-1

1. Ciência da computação. 2. Bases de dados – Gerenciamento. I. Gehrke, Johannes. II. Título.

CDU 004.658

---

Catalogação na publicação: Ana Paula Magnus – CRB 10/2052




# 5

# SQL: CONSULTAS, RESTRIÇÕES, GATILHOS

- O que está incluído na linguagem SQL? O que é SQL:1999?
- Como as consultas são expressas em SQL? Como é o significado de uma consulta especificada no padrão SQL?
- Como a SQL cria e estende a álgebra e o cálculo relacional?
- O que é agrupamento? Como ele é utilizado com operações agregadas?
- O que são consultas aninhadas?
- O que são valores *nulos*?
- Como podemos usar as consultas para escrever restrições de integridade complexas?
- O que são gatilhos, e por que eles são úteis? Como eles estão relacionados às restrições de integridade?
- **Conceitos-chave:** consultas SQL, conexão com a álgebra e o cálculo relacional; recursos além da álgebra, cláusula `DISTINCT` e semântica de multiconjuntos, agrupamento e agregação; consultas aninhadas, correlação; operadores de comparação de conjuntos; valores *nulos*, junções externas; restrições de integridade especificadas usando consultas; gatilhos e bancos de dados ativos, regras evento-condição-ação.

Que homens ou deuses são estes? Que virgens relutantes?
Que busca exasperada? Que luta para se libertar?
Que flautas e adufes? Que arrebatamento extraordinário?

— John Keats, *Ode a um Túmulo Grego*

A Linguagem de Consulta Estruturada (SQL — Structured Query Language) é a linguagem de banco de dados relacional comercial mais amplamente utilizada. Ela foi originalmente desenvolvida pela IBM nos projetos SEQUEL-XRM e System-R (1974-1977). Quase imediatamente, outros fabricantes introduziram produtos SGBD baseados em SQL, que agora é um padrão de fato. A SQL continua a evoluir em resposta às necessidades evolutivas na área de banco de dados. O padrão ANSI/ISO atual da SQL é chamado SQL:1999. Embora nem todos os produtos SGBD suportem completamente

**Conformidade com os Padrões SQL:** O SQL:1999 tem uma coleção de recursos chamada Core SQL, que um fabricante deve implementar para alegar conformidade com o padrão SQL:1999. É estimado que todos os principais fabricantes podem se ajustar ao Core SQL com pouco esforço. Muitos dos recursos restantes estão organizados em **pacotes**.

Por exemplo, os pacotes tratam cada um dos seguintes recursos (com os capítulos relevantes entre parênteses): *data e horário melhorados*, *gerenciamento de integridade melhorado e banco de dados ativos* (neste capítulo), *interfaces de linguagem externa* (Capítulo 6), *OLAP* (Capítulo 25), e *recursos de objeto* (Capítulo 23). O padrão SQL/MM complementa o SQL:1999 definindo pacotes adicionais que suportam a *mineração de dados* (Capítulo 26), *dados espaciais* (Capítulo 28) e *documentos de texto* (Capítulo 27). O suporte aos dados e consultas XML está por vir.

o padrão SQL:1999, os fabricantes estão trabalhando em direção a esse objetivo, e a maioria dos produtos já suporta os recursos principais. O padrão SQL:1999 é bem próximo ao padrão anterior, SQL-92, com relação aos recursos discutidos neste capítulo. Nossa apresentação é consistente com ambos — SQL-92 e SQL:1999 —, e observamos explicitamente quaisquer aspectos que sejam diferentes nas duas versões do padrão.

## 5.1 VISÃO GERAL

A linguagem SQL inclui diversos aspectos:

- **A Linguagem de Manipulação de Dados (DML — Data Manipulation Language):** Este subconjunto da SQL permite que os usuários formulem consultas e insiram, excluam e modifiquem tuplas. As consultas são o principal foco deste capítulo. Cobrimos os comandos da DML para inserir, excluir e modificar tuplas no Capítulo 3.
- **A Linguagem de Definição de Dados (DDL — Data Definition Language):** Este subconjunto da SQL suporta a criação, exclusão e modificação das definições das tabelas e visões. As *restrições de integridade* podem ser definidas nas tabelas, tanto quando a tabela é criada como posteriormente. Cobrimos os recursos da DDL da SQL no Capítulo 3. Embora o padrão não discuta índices, as implementações comerciais também fornecem comandos para criação e exclusão de índices.
- **Gatilhos e Restrições de Integridade Avançadas:** O novo padrão SQL:1999 inclui suporte a *gatilhos (triggers)*, que são ações executadas pelo SGBD sempre que alterações no banco de dados satisfazem condições especificadas no gatilho. Cobriremos gatilhos neste capítulo. A SQL permite o uso de consultas para criar especificações de restrição de integridade complexas. Também discutiremos tais restrições neste capítulo.
- **SQL Embutida e Dinâmica:** Os recursos de SQL embutida permitem que o código SQL seja chamado por meio de uma linguagem hospedeira como C ou COBOL. Os recursos de SQL dinâmica permitem que uma consulta seja construída (e executada) em tempo de execução. Trataremos desses recursos no Capítulo 6.
- **Execução Cliente-Servidor e Acesso a Banco de Dados Remoto:** Estes comandos controlam como um programa de aplicativo *cliente* pode se conectar a um *servidor* de banco de dados SQL, ou acessar dados de um banco de dados através de uma rede. Trataremos esses comandos no Capítulo 7.

- **Gerenciamento de Transação:** Diversos comandos permitem que um usuário controle explicitamente os aspectos da execução de uma transação. Trataremos esses comandos no Capítulo 16.
- **Segurança:** A SQL provê mecanismos para controlar o acesso dos usuários aos objetos de dados, tais como tabelas e visões. Trataremos desses mecanismos no Capítulo 21.
- **Recursos avançados:** O padrão SQL:1999 inclui recursos de orientação a objetos (Capítulo 23), consultas recursivas (Capítulo 24), consultas de apoio à decisão (Capítulo 25), e também trata áreas emergentes como mineração de dados (Capítulo 26), dados espaciais (Capítulo 28) e gerenciamento de texto e de dados XML (Capítulo 27).

### 5.1.1 Organização do Capítulo

O restante deste capítulo está organizado da seguinte maneira: apresentaremos as consultas SQL básicas na Seção 5.2 e introduziremos os operadores de conjunto da SQL na Seção 5.3. Discutiremos as consultas aninhadas, nas quais uma relação referenciada numa consulta é ela própria definida dentro da consulta, na Seção 5.4. Cobriremos os operadores agregados, que nos permitem escrever consultas SQL que não podem ser expressas na álgebra relacional, na Seção 5.5. Discutiremos os valores *nulos*, que são valores especiais usados para indicar valores de campo não existentes ou desconhecidos, na Seção 5.6. Discutiremos as restrições de integridade complexas que podem ser especificadas usando a DDL da SQL na Seção 5.7, estendendo a discussão sobre SQL DDL iniciada no Capítulo 3; as novas especificações de restrições nos possibilitam utilizar completamente os recursos de linguagem de consulta da SQL.

Finalmente, discutiremos o conceito de um *banco de dados ativo* nas seções 5.8 e 5.9. Um **banco de dados ativo** tem uma coleção de **gatilhos**, que são especificados pelo DBA. Um gatilho descreve ações a serem tomadas quando ocorrem determinadas situações. O SGBD monitora o banco de dados, detecta essas situações, e dispara o gatilho. O padrão SQL:1999 requer suporte aos gatilhos, e diversos produtos SGBD relacionais já suportam alguma forma de gatilho.

## Sobre os Exemplos

Apresentaremos diversas consultas de exemplo usando as seguintes definições de tabela:

Marinheiros(*id-marin*: `integer`, *nome-marin*: `string`, *avaliação*: `integer`, *idade*: `real`)
Barcos(*id-barco*: `integer`, *nome-barco:* `string`, *cor:* `string`)
Reservas(*id-marin*: `integer`, *id-barco*: `integer`, *dia*: `date`)

Damos a cada consulta um número único, continuando com o esquema de numeração utilizado no Capítulo 4. A primeira consulta nova deste capítulo tem o número C15. As consultas C1 a C14 foram introduzidas no Capítulo 4.[1] Ilustramos as consultas usando as instâncias $M3$ de Marinheiros, $R2$ de Reservas e $B1$ de Barcos introduzidas no Capítulo 4, as quais reproduziremos nas Figuras 5.1, 5.2 e 5.3, respectivamente.

Todas as tabelas e consultas de exemplo que aparecem neste capítulo encontram-se disponíveis on-line na página web do livro em

`http://www.cs.wisc.edu/~dbbook`

[1] Todas as referências a uma consulta podem ser encontradas no índice de tópicos do livro.

O material on-line inclui instruções de como configurar o Oracle, o IBM DB2, o Microsoft SQL Server, e o MySQL, e os scripts para criação de tabelas e consultas de exemplo.

## 5.2 O FORMATO DE UMA CONSULTA SQL BÁSICA

Esta seção apresenta a sintaxe de uma consulta SQL simples e explica seu significado através de uma *estratégia de avaliação conceitual*, que é uma forma de avaliar a consulta, intencionalmente mais fácil de entender, mas não tão eficiente. Um SGBD executaria normalmente uma consulta de uma forma diferente e mais eficiente.

O formato básico de uma consulta SQL é:

```
SELECT [ DISTINCT ] lista-seleção
FROM     lista-from
WHERE    qualificação
```

| *id-marin* | *nome-marin* | *avaliação* | *idade* |
|---|---|---|---|
| 22 | Dustin | 7 | 45,0 |
| 29 | Brutus | 1 | 33,0 |
| 31 | Lubber | 8 | 55,5 |
| 32 | Andy | 8 | 25,5 |
| 58 | Rusty | 10 | 35,0 |
| 64 | Horatio | 7 | 35,0 |
| 71 | Zorba | 10 | 16,0 |
| 74 | Horatio | 9 | 35,0 |
| 85 | Art | 3 | 25,5 |
| 95 | Bob | 3 | 63,5 |

**Figura 5.1** Uma instância $M3$ de Marinheiros.

| *id-marin* | *id-barco* | *dia* |
|---|---|---|
| 22 | 101 | 10/10/98 |
| 22 | 102 | 10/10/98 |
| 22 | 103 | 10/8/98 |
| 22 | 104 | 10/7/98 |
| 31 | 102 | 11/10/98 |
| 31 | 103 | 11/6/98 |
| 31 | 104 | 11/12/98 |
| 64 | 101 | 9/5/98 |
| 64 | 102 | 9/5/98 |
| 74 | 103 | 9/8/98 |

**Figura 5.2** Uma instância $R2$ de Reservas.

| *id-barco* | *nome-barco* | *cor* |
|---|---|---|
| 101 | Interlake | azul |
| 102 | Interlake | vermelho |
| 103 | Clipper | verde |
| 104 | Marine | vermelho |

**Figura 5.3** Uma instância $B1$ de Barcos.

Toda consulta deve ter uma cláusula `SELECT`, que especifica as colunas a serem mantidas no resultado, e uma cláusula `FROM`, que especifica um produto cartesiano de tabelas. A cláusula opcional `WHERE` especifica as condições de seleção nas tabelas mencionadas na cláusula `FROM`.

Uma consulta como essa corresponde intuitivamente a uma expressão de álgebra relacional envolvendo seleções, projeções e produtos cartesianos. A íntima relação entre a SQL e a álgebra relacional é a base para a otimização de consulta em um SGBD relacional, conforme veremos nos Capítulos 12 e 15. De fato, os planos de execução

das consultas SQL são representados usando uma variação das expressões de álgebra relacional (Seção 15.1).

Vamos considerar um exemplo simples.

*(C15) Encontre os nomes e as idades de todos os marinheiros.*

```
SELECT DISTINCT M.nome-marin, M.idade
FROM    Marinheiros M
```

A resposta é um *conjunto* de linhas, sendo cada linha um par ⟨*nome-marin, idade*⟩. Se dois ou mais marinheiros tiverem o mesmo nome e a mesma idade, a resposta conterá apenas um par com esse nome e idade. Essa consulta é equivalente a aplicar o operador projeção da álgebra relacional.

Se omitíssemos a palavra reservada `DISTINCT`, obteríamos uma cópia da linha ⟨*m, i*⟩ para cada marinheiro com nome *m* e idade *i*; a resposta seria um multiconjunto de linhas. Um **multiconjunto** é semelhante a um conjunto por ser uma coleção desordenada de elementos, mas pode conter diversas cópias de cada elemento, e o número de cópias é significativo — dois multiconjuntos poderiam ter os mesmos elementos e ainda assim ser diferentes em razão do número de cópias ser diferente para alguns elementos. Por exemplo, {a, b, b} e {b, a, b} denotam o mesmo multiconjunto, e diferem do multiconjunto {a, a, b}.

A resposta a essa consulta com e sem a palavra reservada `DISTINCT` na instância *M*3 de Marinheiros é ilustrada nas Figuras 5.4 e 5.5. A única diferença é que a tupla de Horatio aparecerá duas vezes se `DISTINCT` for omitido; isso ocorre porque há dois marinheiros chamados Horatio, de idade 35.

| *nome-marin* | *idade* |
|---|---|
| Dustin | 45,0 |
| Brutus | 33,0 |
| Lubber | 55,5 |
| Andy | 25,5 |
| Rusty | 35,0 |
| Horatio | 35,0 |
| Zorba | 16,0 |
| Art | 25,5 |
| Bob | 63,5 |

**Figura 5.4** Resposta a C15.

| *nome-marin* | *idade* |
|---|---|
| Dustin | 45,0 |
| Brutus | 33,0 |
| Lubber | 55,5 |
| Andy | 25,5 |
| Rusty | 35,0 |
| Horatio | 35,0 |
| Zorba | 16,0 |
| Horatio | 35,0 |
| Art | 25,5 |
| Bob | 63,5 |

**Figura 5.5** Resposta a C15 sem `DISTINCT`.

Nossa próxima consulta é equivalente a uma aplicação do operador seleção da álgebra relacional.

*(C11) Encontre todos os marinheiros com uma avaliação acima de 7.*

```
SELECT  M.id-marin, M.nome-marin, M.avaliação, M.idade
FROM    Marinheiros AS M
WHERE   M.avaliação > 7
```

Essa consulta utiliza a palavra reservada opcional `AS` para introduzir uma variável de intervalo. Aliás, quando desejamos recuperar todas as colunas, como nessa consulta, a SQL provê um atalho conveniente: Podemos escrever simplesmente `SELECT *`. Esta notação é útil para consultas interativas, mas é um estilo pobre para as consultas que deverão ser reutilizadas e mantidas porque o esquema do resultado não é claro a partir da consulta propriamente dita; devemos referenciar ao esquema da tabela subjacente Marinheiros.

Conforme esses dois exemplos ilustram, a cláusula `SELECT` é usada realmente para fazer *projeção*, enquanto as *seleções*, no sentido da álgebra relacional, são expressas usando a cláusula `WHERE`! Essa confusão entre a nomeação dos operadores de seleção e projeção na álgebra relacional e na sintaxe da SQL é um acidente histórico infeliz.

Consideraremos agora a sintaxe de uma consulta SQL básica com mais detalhes.

- A **lista-from** da cláusula `FROM` é uma lista de nomes de tabela. Um nome de tabela pode ser seguido por uma **variável de intervalo** (*range viariable*), que é particularmente útil quando o mesmo nome de tabela aparece mais do que uma vez na lista-from.
- A **lista-seleção** é uma lista de (expressões envolvendo) nomes de coluna das tabelas nomeadas na lista-from. Os nomes de coluna podem ser prefixados por uma variável de intervalo.
- A **qualificação** da cláusula `FROM` é uma combinação booleana (isto é, uma expressão usando os conectivos lógicos `AND`, `OR` e `NOT`) de condições no formato *expressão* `op` *expressão*, onde `op` é um dos operadores de comparação $\{<, <=, =, <>, >=, >\}$.[2] Uma *expressão* é um nome de *coluna*, uma *constante* ou uma expressão (aritmética ou de string).
- A palavra reservada `DISTINCT` é opcional. Ela indica que a tabela computada como uma resposta a essa consulta não deve conter *duplicatas*, ou seja, duas cópias da mesma linha. O padrão é que as duplicatas não sejam eliminadas.

Embora as regras precedentes descrevam (informalmente) a sintaxe de uma consulta SQL básica, elas não nos informam o *significado* de uma consulta. A resposta a uma consulta é por si só uma relação — que em SQL! é um *multiconjunto* de linhas — cujo conteúdo pode ser compreendido considerando-se a seguinte estratégia de avaliação conceitual:

1. Compute o produto cartesiano das tabelas da **lista-from**.
2. Exclua as linhas no produto cartesiano que não satisfazem as condições de **qualificação**.
3. Exclua as colunas que não aparecem na **lista-seleção**.
4. Se `DISTINCT` for especificado, elimine as linhas duplicatas.

Essa estratégia direta de avaliação conceitual torna explícitas as linhas que devem ser apresentadas na resposta à consulta. Entretanto, é bem provável que seja bem ineficiente. Consideraremos como um SGBD realmente avalia as consultas em capítulos posteriores; por ora, nosso propósito é simplesmente explicar o significado de uma consulta. Ilustraremos a estratégia de avaliação conceitual usando a seguinte consulta:

*(C1) Encontre os nomes de marinheiros que reservaram o barco 103.*

[2] As expressões com `NOT` sempre podem ser substituídas por expressões equivalentes sem `NOT` dado o conjunto de operadores de comparação listados.

*C1* pode ser expressa em SQL assim:

```
SELECT  M.nome-marin
FROM    Marinheiros M, Reservas R
WHERE   M.id-marin = R.id-marin AND R.id-barco= 103
```

Vamos computar a resposta a essa consulta nas instâncias $R3$ de Reservas e $M4$ de Marinheiros ilustradas nas Figuras 5.6 e 5.7, uma vez que a computação em nossas instâncias de exemplo usuais ($R2$ e $M3$) seriam desnecessariamente tediosas.

| *id-marin* | *id-barco* | *dia* |
|---|---|---|
| 22 | 101 | 10/10/96 |
| 58 | 103 | 11/12/96 |

**Figura 5.6** Instância $R3$ de Reservas.

| *id-marin* | *nome-marin* | *avaliação* | *idade* |
|---|---|---|---|
| 22 | dustin | 7 | 45,0 |
| 31 | lubber | 8 | 55,5 |
| 58 | rusty | 10 | 35,0 |

**Figura 5.7** Instância $M4$ de Marinheiros.

A primeira etapa é construir o produto cartesiano $M4 \times R3$, que é ilustrado na Figura 5.8.

| *id-marin* | *nome-marin* | *avaliação* | *idade* | *id-marin* | *id-barco* | *dia* |
|---|---|---|---|---|---|---|
| 22 | dustin | 7 | 45,0 | 22 | 101 | 10/10/96 |
| 22 | dustin | 7 | 45,0 | 58 | 103 | 11/12/96 |
| 31 | lubber | 8 | 55,5 | 22 | 101 | 10/10/96 |
| 31 | lubber | 8 | 55,5 | 58 | 103 | 11/12/96 |
| 58 | rusty | 10 | 35,0 | 22 | 101 | 10/10/96 |
| 58 | rusty | 10 | 35,0 | 58 | 103 | 11/12/96 |

**Figura 5.8** $M4 \times R3$.

A segunda etapa é aplicar a qualificação *M.id-marin = R.id-marin* `AND` *R.id-barco = 103*. (Observe que a primeira parte dessa qualificação requer uma operação junção.) Essa etapa elimina todas, exceto a última linha da instância ilustrada na Figura 5.8. A terceira etapa é eliminar as colunas não desejadas; apenas *nome-marin* aparece na cláusula `SELECT`. Essa etapa nos fornece o resultado ilustrado na Figura 5.9, que é uma tabela com uma única coluna e, por coincidência, apenas uma linha.

| *nome-marin* |
|---|
| Rusty |

**Figura 5.9** Resposta à consulta C1 sobre $R3$ e $M4$.

### 5.2.1 Exemplos de Consultas SQL Básicas

Apresentaremos agora diversas consultas de exemplo, muitas das quais foram expressas anteriormente em álgebra e cálculo relacional (Capítulo 4). Nosso primeiro exemplo

ilustra que o uso de variáveis de intervalo é opcional, a menos que elas sejam necessárias para resolver uma ambigüidade. A Consulta C1, que discutimos na seção anterior, também pode ser expressa assim:

```
SELECT  nome-marin
FROM    Marinheiros M, Reservas R
WHERE   M.id-marin = R.id-marin AND R.id-barco= 103
```

Apenas as ocorrências de *id-marin* devem ser qualificadas, uma vez que essa coluna aparece em ambas as tabelas Marinheiros e Reservas. Uma forma equivalente de escrever esta consulta é:

```
SELECT  nome-marin
FROM    Marinheiros, Reservas
WHERE   Marinheiros.id-marin = Reservas.id-marin AND id-barco= 103
```

Essa consulta mostra que os nomes de tabelas podem ser usados implicitamente como variáveis de linha. As variáveis de intervalo precisam ser introduzidas explicitamente apenas quando a cláusula `FROM` contiver mais que uma ocorrência de uma relação.[3] Entretanto, recomendamos o uso explícito de variáveis de intervalo e a qualificação completa de todas as ocorrências das colunas com uma variável de intervalo para melhorar a legibilidade de suas consultas. Adotaremos essa convenção em todos os nossos exemplos.

*(C16) Encontre os* id-marin*s dos marinheiros que reservaram um barco vermelho.*

```
SELECT  R.id-marin
FROM    Barcos B, Reservas R
WHERE   B.id-barco = R.id-barco AND B.cor = 'vermelho'
```

Essa consulta contém uma junção de duas tabelas, seguidas por uma seleção na cor dos barcos. Podemos considerar B e R como as linhas nas tabelas correspondentes que 'demonstram' que um marinheiro com id-marin R.id-marin reservou um barco vermelho B.id-barco. Em nossas instâncias de exemplo $R2$ e $M3$ (Figuras 5.1 e 5.2), a resposta consiste nos *id-marin*s 22, 31 e 64. Se desejarmos os nomes dos marinheiros no resultado, deveremos também considerar a relação Marinheiros, uma vez que Reservas não contém essa informação, como ilustra o próximo exemplo.

*(C2) Encontre os nomes dos marinheiros que reservaram um barco vermelho.*

```
SELECT  M.nome-marin
FROM    Marinheiros M, Reservas R, Barcos B
WHERE   M.id-marin = R.id-marin AND R.id-barco = B.id-barco AND B.cor = 'vermelho'
```

Essa consulta contém uma junção das três tabelas seguidas por uma seleção na cor dos barcos. A junção com Marinheiros nos permite encontrar o nome do marinheiro que, de acordo com a tupla R de Reservas, reservou um barco vermelho descrito pela tupla B.

*(C3) Encontre as cores dos barcos reservados por Lubber.*

[3] O nome da tabela não pode ser usado como uma variável de intervalo implícita, uma vez que uma variável de intervalo é introduzida para a relação.

```
SELECT   B.cor
FROM     Marinheiros M, Reservas R, Barcos B
WHERE    M.id-marin = R.id-marin AND R.id-barco = B.id-barco AND
         M.nome-marin =  'Lubber'
```

Essa consulta é muito semelhante à anterior. Observe que, em geral, pode haver mais do que um marinheiro chamado Lubber (uma vez que *nome-marin* não é uma chave de Marinheiros); essa consulta ainda é correta, pois ela retornará as cores dos barcos reservados por *algum* Lubber, se houver diversos marinheiros chamados Lubber.

*(C4) Encontre os nomes dos marinheiros que reservaram pelo menos um barco.*

```
SELECT   M.nome-marin
FROM     Marinheiros M, Reservas R
WHERE    M.id-marin = R.id-marin
```

A junção de Marinheiros e Reservas assegura que, para cada *nome-marin* selecionado, o marinheiro tenha feito alguma reserva. (Se um marinheiro não tivesse feito uma reserva, a segunda etapa na estratégia de avaliação conceitual eliminaria todas as linhas no produto cartesiano que envolvessem esse marinheiro.)

### 5.2.2 Expressões e Strings no Comando SELECT

A SQL suporta uma versão mais genérica da **lista-seleção** do que apenas uma lista de colunas. Cada item em uma **lista-seleção** pode ser da forma *expressão* `AS` *nome-coluna*, onde *expressão* é qualquer expressão aritmética ou de string envolvendo os nomes de colunas (possivelmente prefixadas por variáveis de intervalo) e constantes, e *nome-coluna* é um novo nome para essa coluna na saída da consulta. A lista-seleção também pode conter *funções agregadas*, tais como *sum* e *count*, que discutiremos na Seção 5.5. O padrão SQL também inclui expressões envolvendo valores de data e hora, os quais não trataremos. Embora não seja parte do padrão SQL, muitas implementações também suportam o uso de funções embutidas como *sqrt*, *sin* e *mod*.

*(C17) Compute incrementos das avaliações de pessoas que manobraram dois barcos diferentes no mesmo dia.*

```
SELECT   M.nome-marin, M.avaliação+1 AS avaliação
FROM     Marinheiros M, Reservas R1, Reservas R2
WHERE    M.id-marin = R1.id-marin AND M.id-marin = R2.id-marin
         AND R1.dia = R2.dia AND R1.id-barco < > R2.id-barco
```

E, ainda, cada item em uma *qualificação* pode ser tão genérico quanto *expressão1 = expressão2*.

```
SELECT   M1.nome-marin AS nome1, M2.nome-marin AS nome2
FROM     Marinheiros M1, Marinheiros M2
WHERE    2*M1.avaliação = M2.avaliação-1
```

Para comparações de string, podemos usar os operadores de comparação (=, <, > etc.) com a ordem das strings determinada alfabeticamente como usual. Se necessitarmos ordenar strings em uma ordem que não seja alfabética (por exemplo, ordenar strings que denotam os nomes dos meses na ordem do calendário: Janeiro, Fevereiro, Março etc.), a SQL suporta um conceito genérico de **collation**, ou ordem de classificação, para um conjunto de caracteres. Uma *collation* permite que o usuário especifique

**Expressões Regulares em SQL:** Refletindo a importância crescente de dados de texto, o SQL:1999 inclui uma versão mais poderosa do operador `LIKE` chamado `SIMILAR`. Esse operador permite que um vasto conjunto de expressões regulares seja usado como padrão na busca de texto. As expressões regulares são semelhantes àquelas suportadas pelo sistema operacional Unix para busca de strings, embora a sintaxe seja um pouco diferente.

**Álgebra Relacional e SQL:** As operações de conjunto da SQL estão disponíveis em álgebra relacional. A principal diferença, naturalmente, é que elas são operações em multiconjuntos na SQL, já que as tabelas são multiconjuntos de tuplas.

quais caracteres são 'menores do que' outros e provê grande flexibilidade na manipulação de strings.

Além disso, a SQL fornece suporte para correspondência de padrão através do operador `LIKE`, juntamente com o uso dos símbolos curinga % (que quer dizer zero ou mais caracteres arbitrários) e _ (que quer dizer exatamente um caractere arbitrário). Assim, '_AB%' denota um padrão correspondente a toda string que contém no mínimo três caracteres, com o segundo e o terceiro caracteres sendo A e B, respectivamente. Observe que diferentemente dos demais operadores de comparação, os brancos podem ser significativos para o operador `LIKE` (dependendo da collation (ordem de classificação) do conjunto de caracteres subjacente). Assim, *'Jeff'* = *'Jeff'* é verdadeiro enquanto *'Jeff'* `LIKE` *'Jeff'* é falso. Um exemplo do uso de `LIKE` em uma consulta é dado a seguir.

*(C18) Encontre as idades dos marinheiros cujos nomes começam e terminam com B e têm no mínimo três caracteres.*

```
SELECT   M.idade
FROM     Marinheiros M
WHERE    M.nome-marin LIKE 'B_%B'
```

O único marinheiro que satisfaz essa consulta é Bob, e sua idade é 63,5.

[4] Observe que, embora o padrão SQL inclua essas operações, muitos sistemas atualmente suportam apenas `UNION`. Além disso, muitos sistemas reconhecem a palavra reservada `MINUS` como `EXCEPT`.