Fábio Burch Salvador

Linguagem SQL
Aprendendo a falar a língua dos bancos de dados

editora
VIENA
1ª Edição
Bauru/SP
Editora Viena
2013

Sumário

Lista de Siglas e Abreviaturas

API _____ *Application Programming Interface.*
BD _____ *Banco de Dados.*
ER _____ *Entidade-Relacionamento.*
IBM _____ *International Business Machines.*
MVCC _____ *Multiversion Concurrency Control.*
SGBD _____ *Sistema Gerenciador de Banco de Dados.*
SQL _____ *Structured Query Language.*
UML _____ *Unified Modeling Language.*

1
Conceitos Básicos

1. Conceitos Básicos

Antes de começar é importante ver algumas noções básicas, sem as quais não será possível entender o conteúdo deste livro. Este capítulo é eminentemente teórico, e embora pareça perda de tempo ler um monte de teoria, peço que o façam com atenção, caso contrário, não será possível entender como funciona um banco de dados.

1.1. O Que São Dados e Para Que Servem?

Dados são informações. Simples assim. Considere, por exemplo, um cadastro de funcionários no qual existem alguns dados, como nome, endereço, data de nascimento, cargo, salário, e assim por diante. Para registrar esses dados em uma estrutura lógica, é preciso criar uma **tabela**, com alguns **campos**.

Aqui, cabe explicar que uma tabela é uma grade que contém registros, cada um deles contendo dados diferentes em seus campos.

Para tornar mais fácil o entendimento destes conceitos, veja um exemplo prático:

1. Crie uma tabela chamada **Empregados**, com os campos **Nome**, **Nascimento**, **Função** e **Salário**;

NOME	NASCIMENTO	FUNÇÃO	SALÁRIO

2. Esta tabela já possui, portanto, uma série de campos previstos, mas ainda não tem nenhum registro. Crie um registro com um empregado;

NOME	NASCIMENTO	FUNÇÃO	SALÁRIO
Severino Quebra-Galho	25/10/1956	Serviços Gerais	R$ 800,00

3. Agora, adicione mais um registro.

NOME	NASCIMENTO	FUNÇÃO	SALÁRIO
Severino Quebra-Galho	25/10/1956	Serviços Gerais	R$ 800,00
João Cannabrava	14/09/1974	Contador	R$ 3000,00

Pode-se inserir os dados de 300, 900, meio milhão de empregados, e cada um será um registro na tabela de dados. Com isso, a empresa terá um conjunto de informações contido nos campos desta tabela.

Estas informações são passíveis, futuramente, de:

- **Alteração:** Pode-se mudar informações, para corrigir um dado que foi digitado incorretamente ou, por exemplo, se quiser dar um aumento ao Severino, mudar o salário dele de R$ 800 para R$ 900.
- **Exclusão:** Pode-se apagar um funcionário, caso seja demitido, por exemplo.
- **Inclusão:** Inserir novos empregados conforme a necessidade surgir.

Mas além de alterar, excluir e incluir registros, o acúmulo de informações em uma tabela de dados permitirá que estes dados sejam utilizados por um programa de computador, que poderá, inclusive, ser construído para fazer cálculos em cima destes dados. Pode-se, por exemplo, construir um programa que, todos os dias, às 8h da

manhã, busque na tabela de dados os nomes dos funcionários que fazem aniversário no dia, imprimindo um bilhetinho de parabéns, a ser entregue a cada um deles. Ou fazer um programa que pegue todas as datas de nascimento que estão na tabela, e calcule a idade média da força de trabalho da empresa.

Um programa também pode ser construído para todo final de mês percorrer a tabela de dados, emitindo os contracheques nos valores dos salários dos empregados, para o setor financeiro pagar. Esta é, aliás, a ideia básica dos sistemas de folha de pagamento, um filão bastante rentável do campo do desenvolvimento de sistemas empresariais.

É possível usar tabelas de dados para tomar decisões gerenciais. Se a tabela do exemplo anterior tivesse, por exemplo, um campo para registrar a cidade onde os empregados moram, e fosse preciso executar uma consulta para ver quantos moram em municípios vizinhos ao da empresa, poderia descobrir que vários empregados vêm de uma mesma cidade, de um mesmo bairro, e que seria mais barato ter uma van para pegar estes empregados em casa ao invés de dar vale-transporte.

Enfim, uma tabela de dados bem projetada é a base para um sistema útil, eficiente e funcional.

1.2. Chave Primária

Ao construir uma tabela para armazenar dados, não se sabe o que as pessoas vão gravar nela, quais serão os seus registros. Mesmo no exemplo anterior, é possível que alguma coisa saia errada: os usuários poderiam registrar dois empregados com um nome muito comum (tipo João da Silva), que tenham nascido no mesmo dia, e trabalhem ambos numa mesma função recebendo, portanto, o mesmo salário. É difícil, mas não impossível.

Esquecendo agora o cadastro de empregados, existem várias aplicações para tabelas de dados que podem gerar registros idênticos, ou com apenas um dos campos preenchido de forma diferente, com consequências desastrosas.

Poderiam haver dois João da Silva trabalhando na empresa, mas cada um com um cargo e cada um com um salário. Como saber quem é quem? A resposta é: teríamos que ter um campo cujo valor jamais pudesse se repetir, como, por exemplo, uma Matrícula de Empregado.

Se tivéssemos na tabela o campo **Matrícula**, o João da Silva que é faxineiro poderia ser o empregado de matrícula 5, e o João da Silva que é auxiliar administrativo, o empregado de matrícula 12. Poderíamos também criar o campo CPF, e como não existem dois brasileiros com CPFs iguais, este poderia ser o campo realmente capaz de diferenciar dois empregados homônimos.

O importante é que exista um campo na tabela que sirva como identificador de um registro, de forma única e que jamais se repita em dois ou mais registros. Ao definir um campo que tenha essas características, ele poderá ser usado como **chave primaria**.

O que é uma **chave primária**? É um campo (ou mais de um, em conjunto, como será visto adiante), que serve de identificador daquele registro e que não se repete em nenhum outro registro. Se acaso o usuário tentar inserir um registro com um dado no campo que é chave primária, mas este dado já existir em outro registro, o novo registro não é gravado.

As chaves primárias podem ser simples ou compostas:

- **Um exemplo de chave primária simples:** É o nosso pequeno cadastro de empregados, adicionando o campo CPF à tabela.
- **Um exemplo de chave primária composta:** É quando dois campos servem como chave primária. Veja um exemplo: uma tabela de CARROS, outra de COMPONENTES e uma terceira de CARRO-COMPONENTE.

CARROS	
CÓDIGO	**MODELO**
1	Ford Del Rey
2	VW Fusca

COMPONENTES	
CÓDIGO	**MODELO**
1	Distribuidor
2	Carburador

CARRO-COMPONENTE		
CARRO	**COMPONENTE**	**DESCRIÇÃO**
1	2	Weber 460
2	2	Solex H30

Este exemplo serve para ilustrar o seguinte: a chave primária da última tabela é formada pelos dois primeiros campos (CARRO e COMPONENTE). Podem existir dois ou mais registros com o campo CARRO, contendo o código do Del Rey (1), e podem existir dois ou mais registros com o campo COMPONENTE contendo o valor correspondente ao Carburador (2), mas o registro do carburador Solex tem como identificadores ambos os dados (2,2). Não poderá existir nenhum outro registro com ambos os valores iguais a estes.

Entendidos estes conceitos, há mais alguns itens a esclarecer.

O campo que é chave primária da tabela, sempre será **NOT NULL**, ou seja, não aceitará um registro no qual fique sem valor algum (nulo).

A maioria dos sistemas gerenciadores de bancos de dados, hoje em dia, aceita determinar o campo que é chave primária, como **AUTO_INCREMENT**. Esta função permite determinar, por exemplo, que num cadastro de empregados, haja um campo chamado MATRÍCULA, que é chave primária com AUTO_INCREMENT. Ao inserir os dados do primeiro empregado, a matrícula atribuída a ele terá como número o 1. O segundo receberá o 2, e assim por diante.

Mesmo que, depois de colocar 40 empregados, eu exclua o registro do empregado número 14, o próximo cadastrado terá a matrícula 41. O sistema gerenciador de banco de dados grava qual o último código atribuído àquele campo no último registro gravado, e atribui mais um número ao próximo registro.

1.3. Várias Tabelas e Relacionamentos

Tabelas de dados são muito mais úteis quando utilizadas em um sistema na qual uma se relaciona com a outra. Existem basicamente três tipos de relação que podem estabelecer-se entre tabelas de dados:

- **Relação um para um (1..1):** Quando um registro de uma tabela relaciona-se com um registro da outra. Um bom exemplo: crie duas tabelas de dados para cadastrar os participantes de uma corrida de carros. Então, cadastre pilotos e carros.

PILOTOS		
CÓDIGO	**NOME**	**IDADE**
1	Airton	49
2	Michael	42
3	Rubinho	38

CARROS			
CÓDIGO	**PILOTO**	**MODELO**	**EQUIPE**
1	2	Opalão Veneno	Red Bucho
2	3	Brasília Amarela	Ferrada
3	1	Passatão Rebaixado	MacMala

Note que, embora tenha cadastrado ambas as tabelas com seus respectivos campos CÓDIGO em sequência (1, 2 e 3), os carros não foram cadastrados na mesma ordem dos pilotos: o carro número 2 (Brasília) é do piloto número 3 (Rubinho), o carro 1 é do piloto 2, e assim por diante. Se, por exemplo, a equipe Ferrada comprar um outro carro para Rubinho, pode-se excluir o registro número 2 e incluir um novo carro, colocando o código de corredor do Rubinho neste novo carro.

- **Relação um para vários (1..n):** É uma relação na qual um registro em uma tabela é equivalente a dois ou mais em outra. Um exemplo bastante simples e bem clássico é o do cadastro de empregados e de cargos e salários.

CARGOS		
CÓDIGO	**CARGO**	**SALÁRIO**
1	Serviços Gerais	R$ 800,00
2	Auxiliar Administrativo	R$ 1.200,00
3	Supervisor	R$ 2.000,00

EMPREGADOS		
CÓDIGO	**CARGO**	**NOME**
1	1	Zé das Couves
2	2	Tonho da Lua
3	2	Seu Saraiva

EMPREGADOS		
CÓDIGO	**CARGO**	**NOME**
4	1	Galeão Cumbica
5	2	Seu Boneco
6	3	Estive Jobes
7	2	Epaminondas Rodézio

Fazendo um cruzamento de dados entre as duas tabelas, dá para ver que Tonho, Saraiva, Boneco e Epaminondas são auxiliares administrativos. Estive é supervisor, enquanto Zé e Galeão fazem a limpeza do prédio.

Este tipo de construção das tabelas de dados permite uma série de operações que seriam impossíveis de se fazer se houvesse uma única tabela de dados de empregados, cada um com seu campo CARGO e seu campo SALÁRIO. Só como exemplo, se a gerência da empresa resolvesse aumentar o salário dos faxineiros para mil reais, bastaria alterar o campo SALÁRIO do registro número 1 da tabela CARGOS, e ao pedir ao sistema o salário do Galeão, ele seria mil reais. O mesmo ocorreria com Zé.

- **Relação vários para vários (n..n):** Este tipo de relação é mais complexa, e ocorre quando um registro da tabela A pode ter relação com dois ou mais registros da tabela B, e o contrário também acontece. Quando se configura este tipo de relação há duas maneiras diferentes para estabelecer a relação dos registros de uma tabela com os da outra. A primeira maneira, e mais simples (porém mais limitada) é criar, em uma das tabelas, vários campos para registrar esta relação. A outra, mais flexível e mais complexa, é a criação de uma terceira tabela, apenas para cuidar desta relação. Veja a seguir dois exemplos, cada um com uma destas construções.

Exemplo 1: Considere uma tabela ESPORTES e uma tabela PESSOAS. As pessoas cadastradas podem escolher seus três esportes favoritos.

ESPORTES	
CÓDIGO	**ESPORTE**
1	Futebol
2	Vôlei
3	Atletismo
4	Boxe

PESSOAS				
CÓDIGO	**NOME**	**ESPORTE1**	**ESPORTE2**	**ESPORTE3**
1	Lineu da Silva	1	2	3
2	Pedro de Souza	3	4	1
3	Antônio da Lua	4	3	1
4	João do Pulo	1	3	4
5	Alan Delonge	2	3	1

Neste exemplo, há uma clara relação vários para vários, pois o esporte Futebol está relacionado com Pedro, Antônio, Alan e João, mas ao mesmo tempo, Alan está relacionado com Futebol, Atletismo e Vôlei.

Exemplo 2: Agora, considere duas tabelas que, nesse exemplo, servem para o cadastro das matérias de um site de notícias. Uma das tabelas será a de ASSUNTOS, e a outra, a de MATÉRIAS. Será criada uma tabela chamada ASSUNTO-MATÉRIA, que servirá apenas para estabelecer a relação entre os registros das outras duas.

ASSUNTOS	
CodAssunto	**Assunto**
1	Política
2	Esporte
3	Policial
4	Engraçadas

MATÉRIAS	
CodMateria	**Título**
1	Ministro tinha envolvimento com o tráfico.
2	Deputados saem no tapa no plenário.
3	Popó, ex-lutador, foi eleito para o Congresso.
4	Ladrão fica entalado na janela da casa.
5	Tenista famoso é assassinado a sapatadas.

ASSUNTO-MATÉRIA	
CodMatéria	**CodAssunto**
1	1
1	3
2	1
2	4
3	2
3	1
4	4
4	3
5	2
5	3
5	4

Para entender como funciona esta relação, primeiro temos que identificar que o relacionamento entre MATÉRIAS e ASSUNTOS é de vários para vários: várias matérias têm o mesmo assunto, e vários assuntos estão em várias matérias.

Mais interessante é ver a tabela ASSUNTO-MATÉRIA.

Nela, por exemplo, a matéria número 5, "Tenista famoso é assassinado a sapatadas", tem relação com os assuntos 2 ("Esporte", porque o cara era um esportista famoso), 3 ("Policial", porque foi um assassinato) e 4 ("Engraçadas", porque há um elemento bizarro, o cara foi morto a sapatadas).

A matéria número 2, "Deputados saem no tapa no plenário", está relacionada com os assuntos 1 ("Política", porque é uma matéria sobre deputados) e 4 ("Engraçadas", porque é uma daquelas bizarrices que fazem as pessoas rirem).

Embora seja mais complicado estabelecer uma relação vários para vários usando uma terceira tabela contendo essas relações, este é o modelo mais versátil e capaz de servir aos sistemas mais complexos.

1.4. Bancos de Dados

Agora que já definimos o que é um dado, e como são armazenados em uma tabela, formando registros (ou "linhas", ou "tuplas"), ao preencher campos definidos para aquela tabela.

Agora que já vimos como os dados são identificados por chaves, como uma tabela de dados pode ter uma relação estabelecida com outra tabela, vamos definir o que é um banco de dados: um banco de dados é um repositório de dados, armazenado em algum lugar (normalmente um computador com grande capacidade de armazenamento).

Cabe explicar, agora, que tudo o que vimos até aqui refere-se a um tipo de banco de dados chamado de **modelo relacional**, caracterizado pela organização em tabelas, também chamadas de "relações", daí o nome do modelo. Existem outros tipos de bancos de dados, outras formas de armazenar dados.

1.4.1. Sistema Gerenciador de Banco de Dados

De forma simplificada, os dados armazenados em um banco de dados são apenas informações gravadas em arquivos no computador que os hospeda. Mas é possível construir bancos de dados absolutamente funcionais, usando até coisas simples, como arquivos de texto.

Um bom exemplo seria o seguinte: se eu salvasse uma série de dados em um arquivo chamado **dados.txt**, eles poderiam estar armazenados de forma que cada registro ficasse em uma linha, com os campos separados por um sinal qualquer, como um ponto-e-vírgula. Veja como cadastrar alguns músicos, as bandas e o gênero musical.

```
NEY MATOGROSSO;SECOS E MOLHADOS;MPB
AXL ROSE;GUNS N ROSES;ROCK
BRUCE DICKINSON;IRON MAIDEN;METAL
JOEY RAMONE;RAMONES;PUNK_
```

Como é que um programa de computador poderia usar estes dados, de alguma forma? Fácil: para salvar um novo registro, o programa teria que percorrer a lista de dados do arquivo, até achar uma linha em branco e, então, preencher aquela linha com os novos dados. Para buscar um dado qualquer, o programa teria que percorrer a lista, lendo o primeiro dado (nome do músico), e comparando com o nome procurado (se estivesse procurando o Axl Rose, teria sucesso na procura ao chegar à linha 2). Uma

vez encontrado o dado certo, poderia usá-lo no programa, carregando os 3 valores dos 3 campos em variáveis. Ou apagar aquela linha.

Gerenciar dados dentro de arquivos-texto exige um grande esforço de programação, um tempo muito grande de processamento, especialmente se tiver que relacionar dados de um arquivo com dados de outro, e uma noção muito precisa de lógica da programação, e da organização dos dados.

Embora pareça algo absurdo, o armazenamento de dados em arquivos-texto já foi prática muito comum, no princípio dos sistemas de gravação de dados em discos, no fim da era dos cartões perfurados. Neste, e em outros sistemas igualmente arcaicos de armazenamento de dados, as informações são jogadas nos arquivos, nos quais ficarão gravadas, e quem faz todo o trabalho "inteligente" em cima destes depósitos "mortos" de dados é o programa. O que compromete muito tempo de desenvolvimento dos sistemas, dá margem a diversos erros e toma, na execução do programa, um grande tempo de processamento.

Esta problemática foi sentida, desde os anos 1960 e, então, passaram a ser criados programas específicos para fazer os processos de pesquisa, inserção de dados, exclusão e alteração, que passariam a funcionar de forma integrada a outros programas, estes sim, voltados para a interface com o usuário e outras tarefas de cálculo e processamento das informações.

Com o tempo, e a evolução destes Sistemas Gerenciadores de Bancos de Dados (SGBD), estes passaram a incorporar cada vez mais funções, e a diversificarem. Temos hoje SGBDs de grande sucesso, outros nem tanto, alguns que são de uso livre, de código aberto, outros proprietários, pertencentes a empresas, cujo uso é pago e o código-fonte do programa, secreto. Uns são conhecidos pela capacidade de lidar com acervos gigantescos de dados, outros pela velocidade, outros pela simplicidade, pela estabilidade, etc.

1.5. Linguagem SQL

Inicialmente, um SGBD era um utilitário desenvolvido pela própria empresa que faria uso dele, em seus sistemas, ou até por desenvolvedores que criavam seus próprios SGBDs e comercializavam, de forma independente, e muitas vezes sem conhecer os demais sistemas rivais. De modo que, nos primórdios, não havia uma mínima preocupação com padronizações, e um grande especialista, capaz de trabalhar perfeitamente com um sistema, e que tornava-se absolutamente novato diante de outro diferente.

Nos anos 1970, finalmente, a **IBM** passou a criar uma linguagem estruturada para um sistema que estava desenvolvendo. Este trabalho daria origem à linguagem **SQL Structured Query Language** ou **Linguagem de Consulta Estruturada**. A rápida disseminação desta nova linguagem deveu-se, não apenas ao enorme tamanho que a **IBM** tinha no mercado naquela época (era uma espécie de **Microsoft** na época, criava padrões mundiais), mas também ao fato de que muitos empregados envolvidos no projeto, mais tarde, fundaram suas próprias empresas, desenvolvendo SGBDs, usando a mesma linguagem e ideia básica, ou sistemas que funcionavam integrados ao SGBD da **IBM**.

De uma forma geral, **SQL** é uma linguagem só, mas os diversos SGBDs que existem por aí têm diferenças, normalmente pequenas, de sintaxe, sinais, funções disponíveis e outros aspectos.

A partir da disseminação de uma linguagem com a qual um programa (ou um operador, inserindo comandos diretamente) consegue "conversar" com um sistema de banco de dados de forma previsível, tornou-se mais simples para um programador preparar-se para trabalhar em empresas diferentes, com sistemas diferentes. Também ficou mais fácil migrar um programa, do uso de um SGBD, para outro, conforme as necessidades.

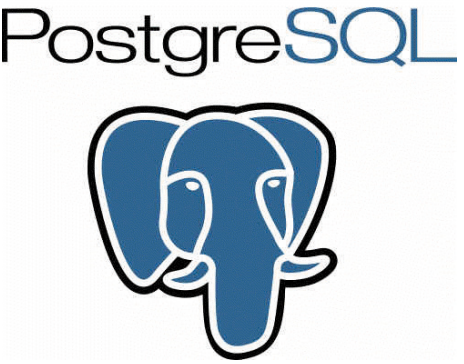
1.5.1. Como Funciona o SQL

O programa que faz o gerenciamento da base de dados (o SGBD) conversa com uma interface qualquer, que pode ser um outro programa, ou o prompt do próprio SGBD, através do qual o usuário poderá inserir comandos em **SQL** para execução. Este usuário escreve algo do tipo "Inserir tal dado em tal tabela", ou "apagar o registro cuja chave primária é tal", e o SGBD se encarrega de "obedecer" a este comando, dando normalmente um retorno se a operação foi bem sucedida, e quantos registros foram alterados. Também é possível inserir requisições de dados do tipo "buscar registros de pessoas que sejam do sexo masculino, maiores de 40 anos", e receber de volta uma lista com as pessoas que se enquadram neste perfil.

Quando a relação entre o banco de dados e um outro sistema se dá pelo mecanismo de interação com SGBD (este mecanismo é chamado de **API** (Application Programming Interface) e consiste em um protocolo que permite a comunicação de um programa com o outro), os comandos **SQL** não são enviados pelo usuário diretamente, e sim por uma linha de comando. O retorno de dados normalmente é carregado como uma variável, para ser manipulado também pelas linhas de comando do sistema. Por exemplo, se o programa requisita os dados do empregado número 412 da empresa e um destes dados é o nome, "Zé da Silva", este nome retorna como dado manipulável pelo programa, para exibir na tela, etc.

1.6. PostgreSQL

Neste livro, vamos optar pelo uso do sistema **PostgreSQL**, por ser o sistema de banco de dados open source mais completo existente hoje. A opção por este sistema deve-se ao fato de que ele possui funções variadas, como **Triggers**, **MVCC** (Multiversion Concurrency Control, ou Controle de Concorrência Multiversões), **Chaves Estrangeiras** (Foreign Keys), e muitas outras, presentes nos melhores sistemas proprietários (e aliás, ausentes em alguns deles), o que dá ao leitor a oportunidade de aprender a lidar com uma variada gama de funcionalidades. Ele até suporta a construção de código de programação procedural em diversas linguagens diferentes.



O **PostgreSQL** teve sua primeira versão publicada em 1996 e continua recebendo aprimoramentos até os dias atuais.

1.7. Servidor de Dados

Antes de prosseguir, vamos falar como funciona o acesso a dados em nível local e remoto. Uma vez que um computador tem, instalado em si e funcionando, um sistema gerenciador de banco de dados, programas poderão acessar estes dados, gravar novos, apagar e alterar. Caso o acesso à base de dados seja aberto a computadores conectados via rede ao computador que hospeda o banco de dados, então este torna-se um servidor de dados.

Anotações

Anotações

Anotações

2
Modelo Entidade-Relacionamento