Treinamento em SQL

Nível: Básico
Horas: 4

Índice

Rua Lauro Vannucci, 1020
Jardim Santa Cândida – Campinas – SP
CEP 13087-548
Tel (19) 3256-5433
Fax (19) 3256-5433
conceptia@conceptia.com.br

Introdução

O que são Banco de Dados

Bancos de dados são entidades computacionais responsáveis por armazenar informações para que elas não se percam com o tempo ou quando o computador seja desligado. Esses dados são armazenados num disco rígido, normalmente com uma alta capacidade para que não existam problemas de falta de espaço.

Dessa forma, toda aplicação que de alguma forma precise manter algum tipo de informação armazenada através do tempo e não possa fazer por meio de arquivos textos (por uma série de limitações de crescimento, segurança e consistência) deve ter uma conexão com banco de dados para que esse armazenamento, bem como consulta aos dados, possa ocorrer.

Bancos de Dados Relacionais

O conceito de bancos de dados relacionais está na forma em que eles são implementados, que estabelece uma relação lógica entre os dados, para que a repetição de dados (redundância) seja a menor possível, economizando espaço em disco e aumentando a velocidade de consulta dos dados.

Um banco de dados relacional possui como entidade central tabelas, onde as colunas armazenam os tipos de dados e as linhas um caso específico de dados, sendo chamada de tupla ou registro. Também é importante o conceito de chave, que identifica unicamente um registro.

Estudo de Caso: PostgreSQL | MySQL

Existem muitos bancos de dados sendo utilizados atualmente, porém, segundo uma visão de mercado, eles podem ser divididos em dois grandes grupos: os comerciais, que cobram uma valor de licença para serem usados e os livres, que possuem distribuição livre, inclusive do código fonte, não implicando em nenhum custo adicional.

Dentre os bancos de dados comerciais, três possuem um destaque maior devido a sua relevância: o Oracle, que é um dos mais famosos do mundo, sendo indicado principalmente para aplicações médias e grandes, onde a possibilidade de crescimento é muito importante; o Microsoft SQLServer, que vêm ganhando importância; e o DB2 da IBM, que começa a ganhar destaque.

Nos bancos de dados de distribuição livre se destacam o MySQL, uma das primeiras iniciativas na área, mas que possui uma séria de limitações técnicas; e o PostGreSQL, que pode ser utilizado desde aplicações pequenas até médias, com uma série de recursos interessantes (inclusive Triggers) que permite um crescimento razoável para soluções que a adotam.

O que é SQL

A sigla SQL significa “Structured Query Language”, ou seja, Linguagem Estrutura de Consulta, e ela foi a forma encontrada para que a comunicação com um banco de dados pudesse ser feita de uma maneira descomplicada, ágil e que pudesse ser facilmente entendida e aprendida pelos desenvolvedores.

Ela possui uma sintaxe padrão que é implementada pela maioria dos bancos de dados, porém cada banco de dados a aumenta da maneira que lhe é mais interessante ou que forneça um maior número de recursos para os usuários, o que leva à situação de não ser tão fácil mudar de um banco de dados para outro, principalmente por causa de detalhes ou funções pré-definidas.

Como utilizar

Cada linguagem de programação implementa isso de uma forma específica, mas todas elas possuem alguns pontos semelhantes. De modo geral, os passos a seguir são comuns, todos implementados dentro da própria aplicação:

- O comando SQL é escrito como uma cadeia de caracteres da linguagem e armazenada numa variável;
- Uma conexão com o banco de dados da aplicação é criada (através de uma rede local ou da Internet) e a referência fica armazenada numa outra variável;
- A variável de conexão deve possuir alguma função que dado um comando SQL (que neste momento está armazenado numa variável) o manda para o banco e fica esperando o resultado;
- Uma terceira variável recebe o resultado da consulta, sendo que essas informações podem ser trabalhadas da maneira que a aplicação quiser/puder para se exibir ou então realizar alguma computação;
- Ao final do processo, é interessante finalizar a conexão com o banco de dados, que normalmente não está no mesmo computador que executou a operação.

Vale destacar ainda que todas as essas operações que foram descritas como sendo realizadas de dentro de uma aplicação, podem ser realizadas diretamente de um terminal que forneça acesso ao banco de dados.

Criando um Banco de Dados

Criando um Banco de Dados

Existem duas formas de se criar um banco de dados, diretamente do terminal ou então de dentro de um outro banco de dados já existente, sendo em seguida necessário se conectar ao outro banco de dados. Utilizaremos como exemplo para o primeiro caso um banco de dado PostGreSQL, mas funções similares são fornecidas para outros bancos. Admitindo que o PostGreSQL está corretamente instalado, o comando para a criação de um novo banco de dados seria:

```
$ createdb –E Latin1 bdCurso
```

No segundo caso, de dentro de algum banco de dados já criado e que se tem acesso, o seguinte comando deveria ser escrito:

```
$ CREATE DATABASE “bdCurso” WITH ENCODING = ‘LATIN1’;
```

Vale destacar ainda que, a sintaxe do SQL não faz diferenciação entre maiúsculas e minúsculas, sendo o seu uso indiscrimado.

Tipos de dados

Da mesma forma que outras linguagens de programação, existe a definição de tipos de dados em SQL e a sua obrigatoriedade de definição durante o momento em que o banco de dados está sendo criado. Da mesma forma, caso o sistema tente inserir um dado incompatível numa determinada coluna, o banco de dados deverá retornar uma mensagem de erro.

A SQL padrão possui um grupo de dados básico e cada banco de dados implementa uma série de variações delas, para permitir uma maior flexibilidade para o usuário. Tomando novamente como exemplo o caso do PostGreSQL, os tipos de dados existentes mais comuns são:

Tipo de Dado	Descrição
BIGINT	Inteiro com precisão de 8 bytes
BOOLEAN	Valor de verdadeiro ou falso
VARCHAR(n)	Cadeia de caracteres de tamanho variável
CHAR(n)	Cadeia de caracteres de tamanho fixo
DATE	Data
FLOAT8	Ponto flutuante de precisão dupla
INTEGER	Inteiro com 4 bytes
DECIMAL(p, s)	Ponto flutuante de precisão variável
REAL	Ponto flutuante de precisão simples
SMALLINT	Inteiro com precisão de 2 bytes
SERIAL	Inteiro auto-incremental
TEXT	Cadeia de caracteres de tamanho variável
TIME	Hora
TIMESTAMP	Data e hora

Definindo Tabelas

Definindo campos

Para cada atributo que deverá existir numa tabela, deve-se dar um nome, um atributo e opcionalmente um modificador, como por exemplo, que esse atributo criado não possa ser vazio quando for inserido algum dado. Não é interessante que esses campos tenham um nome muito grande, pois dificulta a sua posterior utilização.

Criando uma tabela

Como já mencionado, uma tabela é uma dos conceitos mais importantes existentes no modelo de banco de dados relacional. São nas tabelas que os dados são armazenados e acessados por diferentes aplicações que precisam ter informações a respeito delas. Normalmente as tabelas são definidas inicialmente no projeto, antes que dados sejam existentes. Uma tabela pode ser criada como:

```
CREATE TABLE pessoa (

     cpf            VARCHAR(12) NOT NULL,
     nome           VARCHAR(80) NOT NULL,
     nascimento     DATE,
     altura         REAL,
     filhos         INTEGER,

     PRIMARY KEY (cpf)

);
```

Neste exemplo, é criada uma tabela que representa uma pessoa, com alguns atributos que fazem referência aos dados de uma pessoa. O item de chave primária será explicado abaixo, mas indica que o atributo não poderá ser repetido em nenhum outro registro desta tabela, caso contrário um erro será devolvido e por fim, o modificador NOT NULL, indica que esses campos são de preenchimento obrigatório.

Ao se executar esse comando, uma tabela sem dados com essa definição é criada e está pronta para receber dados, ou seja, pronta para inserir novas pessoas no cadastro.

Conceito de Chaves

Chave primária

Chave primária é um conceito muito importante que está diretamente ligado à possibilidade de se consultar dados de tabelas do banco de dados. Uma chave primária é um atributo ou conjunto de atributos que identifica unicamente um registro na tabela e que por isso não pode ser repetido, pois caso contrário não teríamos como diferenciar um registro de outro quando fossemos consulta-los.

Podemos definir um atributo como sendo chave primária explicitamente, ou seja, incluímos um campo contador que é incrementado em um a cada novo registro inserido ou então algum atributo que temos certeza que identifica unicamente um registro, como o número do cpf num cadastro de pessoa.

Chave estrangeira

Muitas vezes desejamos dividir os dados de uma tabela em várias outras para que não aconteça a duplicação de dados no sistema, e uma maneira de se fazer isso é utilizando chaves estrangeiras. Por exemplo, imagine que uma pessoa possa ter de um a três número de telefone, mas só gostaríamos de colocar um atributo telefone na tabela principal, pessoa.

Se quiséssemos inserir mais de um número de telefone, teríamos que ou separar os diferentes números por algum caractere especial, como vírgulas ou então inserir novos registros mudando apenas o número de telefone, só que para isso teríamos que definir a chave primária da tabela como sendo o cpf e o número do telefone, pois poderia ocorrer telefones diferentes com o mesmo número de cpf.

Uma solução para este caso é manter a tabela pessoa como ela está e criar uma nova tabela, chamada telefones, como segue:

```
CREATE TABLE telefones (

     cpf            VARCHAR(12),
     telefone       VARCHAR(10)

     PRIMARY KEY (cpf, telefone),
     FOREIGN KEY (cpf) REFERENCES pessoa

);
```

Neste exemplo, o conjunto cpf e telefone compõe a chave primária da tabela, portanto não podem se repetir e a chave estrangeira é o cpf, que indica que deve existir um número de cpf na outra tabela, pessoa, antes de inserir um registro nesta tabela. Para alguns tipos de computação e uma consulta mais ágil esse tipo de modelagem é muito importante.

Rua Lauro Vannucci, 1020
Jardim Santa Cândida – Campinas – SP
CEP 13087-548

Tel (19) 3256-5433
Fax (19) 3256-5433
conceptia@conceptia.com.br

Inserindo Dados

Inserindo dados numa tabela

A inserção de dados numa tabela é uma das operações mais simples que podem ser realizadas utilizando a SQL. Um exemplo para a tabela acima seria:

```
INSERT INTO pessoa(cpf, nome, nascimento, altura, filhos)
VALUES ('123456789012', 'Fulano de Tal Silva', '1982-10-10', 1.82, 2);
```

Alguns pontos merecem destaque neste comando: deve-se declarar qual a tabela em que estão sendo inseridos os dados, bem como a ordem em que os atributos serão colocados (a parte dos atributos é opcional, mas se não for colocada a ordem dos dados deve seguir a ordem dos campos na tabela). Em seguida vêm os dados propriamente ditos, sendo que caracteres devem vir entre aspas, o formato de data é Ano-Mês-Dia e o valor decimal utiliza ponto e não vírgula. Além disso, todos os valores são separados por vírgula.

Selecionando Dados

Selecionando dados de uma tabela

Outra operação muito importante de se realizar num banco de dados é o resgate de informações uma vez armazenadas nele. Isso é feito utilizando a cláusula SELECT, que como o próprio nome indica, retorna registros de uma tabela. Um exemplo seria:

```
SELECT * FROM pessoa;
```

Neste exemplo, todos os registros já inseridos na tabela pessoa são retornados para uma variável, dependendo do programa ou então exibidos na tela, caso esteja sendo utilizado algum ambiente de acesso aos dados diretamente.

Esse tipo de consulta é muito simples, pois não limita quais dados serão retornados, o que pode, para casos em que a tabela possua muitos registros, retornar uma quantidade de dados impossível de ser tratada pelo programa que executou a consulta.

A sintaxe nomeTabela.nomeCampo

Uma forma alternativa de se realizar consultas a determinados campos de uma tabela é através do seu nome, o operador ponto "." e o nome do atributo. Então, a consulta anterior ficaria sendo:

```
SELECT pessoa.* FROM pessoa;
```

Ou ainda:

```
SELECT pessoa.cpf, pessoa.nome, pessoa.nascimento,
pessoa.altura, pessoa.filhos FROM pessoa;
```

O segundo exemplo permite que os atributos sejam arranjados da melhor maneira possível para uma eventual exibição na tela.

A cláusula WHERE

Em conjunto com o operador SELECT, pode ser utilizada a palavra WHERE, que permite fazer uma restrição dos registros a serem retornados pela consulta. Após o WHERE, podem ser colocadas várias condições que serão operadas sobre os registros, só retornando os

dados que realmente satisfação essas condições. Por exemplo, retornar todas as pessoas com altura maior que 1,60m:

```
SELECT * FROM pessoa WHERE altura > 1.60;
```

Se nenhum registro obedecer a esta restrição, será retornado um resultado vazio, indicando que nada foi encontrado. Outro exemplos seria retornar apenas o nome das pessoas que tenham três filhos:

```
SELECT pessoa.nome FROM pessoa WHERE filhos = 3;
```

Sendo que o resultado conterá apenas o nome das pessoas que obedeçam ao critério de igualdade desejado.

Operadores lógicos: AND, OR, NOT

Junto com a cláusula WHERE podem vir um número qualquer de condições para se satisfazer a consulta dos registros, sendo que essas condições devem vir separadas por operadores lógicos. Assim, para retornar algum registro, ambos os lados da expressão (expr1 AND expr2) devem ser satisfeitos, ao passo que apenas um dos lados da expressão (expr1 OR expr2) precisa. Juntando os exemplos acima, teríamos algo como:

```
SELECT * FROM pessoa WHERE (altura > 1.60) AND (filhos = 3);
```

Essa consulta só retornará algum resultado se ambas as condições forem verdadeiras, caso contrário, nada será retornado.

Atualizando Dados

Atualizando dados de uma tabela

Existem situações em que é desejado que um registro já inserido seja atualizado, com alguns de seus valores modificados. Para esta situação existe o operador UPDATE, que atualiza um determinado registro ou grupo de registros de uma tabela. É importante ressaltar que esta operação irá atuar em todos os registros caso nenhuma restrição usando WHERE seja definida. Assim, se quisermos modificar o número de filhos de todas as pessoas para 1, escreveríamos:

```
UPDATE pessoa SET filhos = 1;
```

Agora, se quiséssemos que apenas aqueles que tenham um determinado cpf tivessem o número de filhos modificados, escreveríamos:

```
UPDATE pessoa SET filhos = 1 WHERE cpf = '123456789012';
```

Neste exemplo, apenas um registro será modificado, pois estamos utilizando uma chave primária para restringir os registros que serão modificados.

Uso da palavra SET

Pode ser alterado qualquer número de atributos de uma tabela numa operação de UPDATE. Para tanto, devemos utilizar SET seguido pelos parâmetros a serem modificados separados por vírgula, sendo que cada parâmetro deve ser colocado com o valor de igual para o novo valor que este irá receber.

Removendo Dados

Removendo dados de uma tabela

A sintaxe para a remoção de registros de uma tabela é muito parecida com a atualização de dados da tabela, a diferença está que o registro será removido definitivamente do sistema. Daí é de extrema importância a utilização da cláusula WHERE, caso contrário, todos os registros da tabela serão apagados, o que pode ser desastroso para uma aplicação. A sintaxe básica é:

```
DELETE FROM pessoa;
```

Nesse comando, todos registros da tabela pessoa seriam removidos, o que dificilmente é interessante para uma aplicação real. O seguinte comando é bem mais útil:

```
DELETE FROM pessoa WHERE cpf = ‘123456789012’;
```

Nesse caso, apenas o registro que tenha como cpf o valor especificado seria removido da tabela. Essa outra consulta poderia ser utilizada para remover determinados registros da tabela telefones:

```
DELETE FROM telefones WHERE cpf = ‘123456789012’;
```

Como na tabela de telephones pode existir mais de um telefone para o mesmo cpf, vários registros podem ser removidos com a mesma consulta. Um item importante para ressaltar com esse comando é a importância de manter a consistência dos dados, ou seja, não podem ser mantidas numa tabela dados de chave estrangeira se os dados da chave primária correspondente na tabela inicial tenham sido removidos, causando um erro na consulta.

Juntando Dados

Juntando dados de duas ou mais tabelas

Em determinadas situações, precisamos juntar os dados de duas tabelas diferentes que de alguma forma se relacionam, pois o modelo relacional foi concebido desta forma, então um dos seus pontos fortes está em não duplicar os dados à partir do momento que se usa relacionamentos baseados em chaves para separar os dados em locais diferentes.

Porém, é preciso fazer o caminho inverso, ou seja, dadas duas tabelas, de alguma forma pegar seus dados e montar um relacionamento que faça sentido naquele contexto. Como exemplo, podemos tomar a tabela de pessoas e de telefones mencionadas acima. Caso queiramos pegar todos os telefones de uma determinada pessoa como deveríamos proceder?

Simplesmente fazendo uma consulta à tabela de telefone não seria muito útil, pois ao invés de termos os nomes das pessoas, teríamos apenas seus números de cpf, que de modo geral são ruins para os seres humanos identificarem-se uns aos outros. O ideal seria colocar o nome da pessoa ao lado do respectivo número de telefone, mas o problema reside no fato dos nomes estarem em outra tabela.

O jeito mais simples de fazer isso é utilizar o chamado “produto cartesiano”, que consistem em pegar cada um dos registros de uma tabela e relacionar com cada uma das linhas da outra tabela. Vale destacar ainda que o resultado de uma junção é a união de todos os atributos das tabelas em questão. A sintaxe é muito simples e seria:

```
SELECT * FROM pessoa JOIN telefones;
```

Ou de uma forma mais simples:

```
SELECT * FROM pessoa, telefones;
```

Em ambos o caso o resultado produzido será o mesmo, porém não muito útil nem com um sentido muito claro. Pois se numa tabela tivermos quatro registros e na outra oito, o resultado produzido será de 32 registros (uma multiplicação simples), mas alguns telefones

estarão associados a pessoas erradas. Nesse ponto que entra a utilização do conceito de chaves. Um exemplo modificado seria:

```
SELECT * FROM pessoa, telefones WHERE pessoa.cpf = telefones.cpf;
```

No exemplo acima, só serão retornados os registros onde as chaves forem iguais, ou seja, o cpf de uma pessoa deve aparecer na tabela de telefones para que a linha seja incluída no resultado final. A união de todos os atributos continua ocorrendo da mesma forma, porém os dados passam a fazer sentido.

É necessário discriminar de onde vem cada parâmetro da condição WHERE porque eles possuem o mesmo nome, o que geraria um erro na execução da consulta. Esse tipo de operação pode conter quantas tabelas forem necessárias, mas é uma boa prática sempre colocar os relacionamentos das chaves para limitar o número de dados retornado. E por fim, dizer que como essa operação manuseia uma imensa quantidade de registro e comparações, ela tende a ser mais lenta que as demais, então minimizar a sua utilizar é aconselhável.

Referência

http://dev.mysql.com/

Site do banco de dados mySQL, de onde é possível fazer o download deste, bem como obter uma grande quantidade de documentação, inclusive de instalação, além de poder contribuir com o projeto em si. É um dos bancos de dados mais fáceis de serem instalados em ambiente Microsoft, uma vez que possui código executável direto para esta plataforma.

http://www.postgresql.org/

Site do banco de dados postGreSQL, um banco de dados livre, que vem ganhando muito destaque no mundo, tanto entre desenvolvedores como entre clientes, por sua estabilidade, eficiência e alta capacidade de armazenamento de dados, além de implementar diversos recursos interessantes, como controle de transação, chave estrangeira e linguagens procedurais.