

LCP - Aula 08

Prof. Lucas Guerreiro

Aulas anteriores

- GUI Swing
- Componentes básicos (JFrame, JLabel, JButton, etc)
- Tratamento de eventos (mouse/teclado)
- Aplicações com Swing

Aula 08

Objetivos: Conceitos de Bancos de Dados. SQL: fundamentos e overview. JDBC.

Banco de Dados (Conceitos)

- Até este momento, trabalhamos com dados em memória e em arquivos.
- Agora vamos ver como trabalhar com dados em bancos de dados, os quais permitem comunicação entre sistemas e servidores de maneira distribuída.
- Banco de dados = **coleção organizada de dados**
- Sistema de Gerenciamento de Banco de Dados (**SGBD**): sistema que fornece mecanismos para armazenar, organizar, recuperar e modificar de diversos usuários e aplicações que integram um sistema de banco de dados.
- Existem diversas classificações/tipos de Banco de Dados, sendo o mais popular o **Banco de Dados Relacional**.

BD Relacional

BD Relacional = representação lógica que **armazena dados em tabelas** e que permite o acesso aos dados sem conhecer sua estrutura física.

Relações são expressas em **tabelas** que caracterizam a semântica e tipo de dados em **colunas**, em que registros são expressos por **linhas** em tais tabelas.

Tabela de **Alunos**

Nome	Idade	Curso
João	23	BCC
Maria	30	Pedagogia
José	26	Biologia

Coluna

Linha

BD Relacional

- **Bancos Relacionais** abordam, ainda, alguns conceitos, como o de **chave primária** que é uma das colunas (ou combinação de colunas) que identifica cada linha de forma única.
- **Chaves estrangeiras** são colunas da tabela que fazem relacionamento com outras tabelas.
- Podemos fazer inserções, remoções, atualizações e consultas de linhas e colunas das tabelas. Estas manipulações de dados das tabelas ou delas em si, podem ser feitas através de operações **SQL**, que é a linguagem padrão de manipulação de BDs Relacionais.
- Em Java, temos pacotes/APIs para que os programas possam manipular BDs, veremos o uso da conexão com o Banco de Dados **MySQL** e ainda o uso do driver **JDBC** (Java Database Connectivity) para conexão e integração com o banco.

SQL

- Em SQL, podemos ter diversos tipos de comandos, mas podemos destacar os comandos de **manipulação da tabela** em si (como as "colunas") e dos **dados** (como as "linhas").
- Em relação às tabelas, podemos, por exemplo, criar tabelas através do comando `CREATE TABLE`, a sintaxe geral do comando pode ser resumida como:

```
CREATE TABLE nome (  
    atributo1      tipo_de_dado_dominio  <restrição_atributo>,  
    ...  
    <restrições_da_tabela>  
)
```

Exemplo

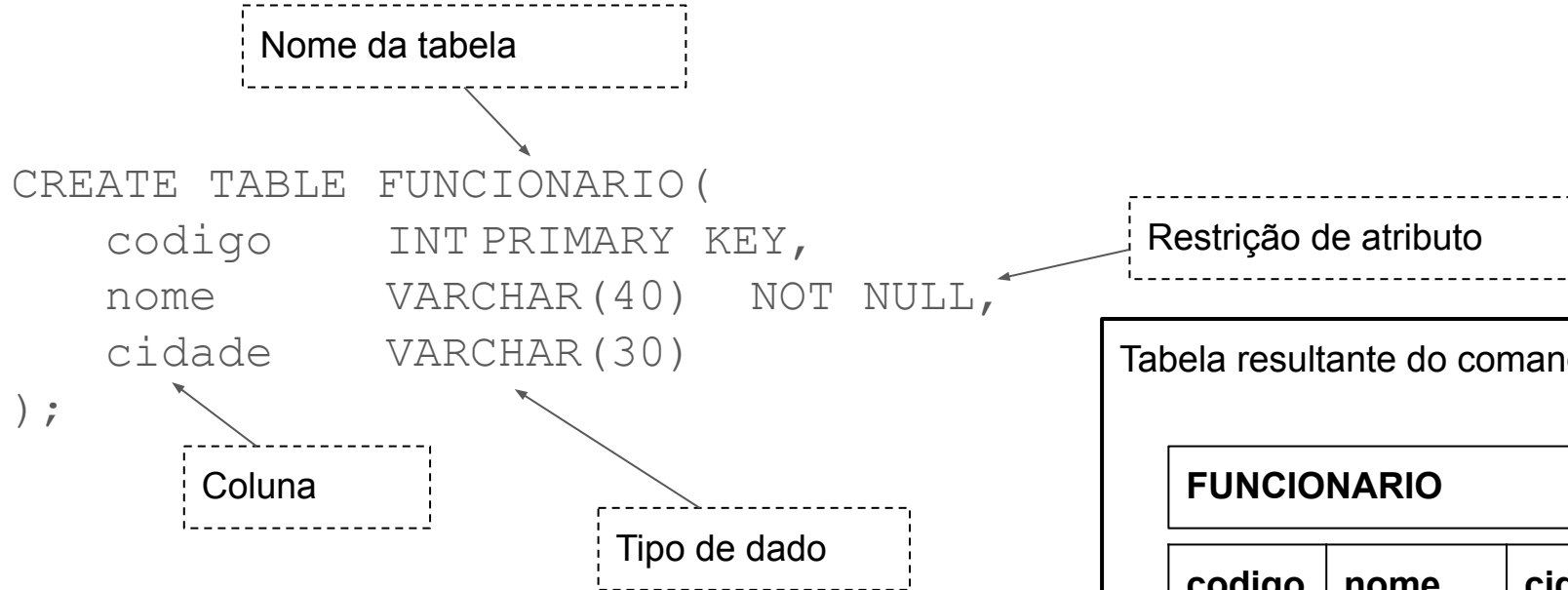


Tabela resultante do comando:

FUNCIONARIO		
codigo	nome	cidade

SQL

Além dos comandos de criação de tabela, temos comandos de manipulação dos dados, sendo os principais:

INSERT: utilizado para inserir dados (linhas) na tabela

UPDATE: utilizado para atualizar valores de linhas

DELETE: utilizado para remover linhas

SELECT: utilizado para consultar dados de tabelas


Insertão

```
INSERT INTO Funcionario  
VALUES (1, "Fulano", "Rio Claro");  
  
INSERT INTO Funcionario  
VALUES (2, "Ciclana", "São Paulo");  
  
INSERT INTO Funcionario  
VALUES (3, "Beltrano", "Rio Claro");
```

codigo	nome	cidade
1	Fulano	Rio Claro
2	Ciclana	São Paulo
3	Beltrano	Rio Claro

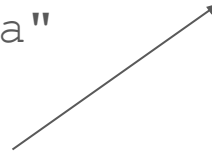
Atualização

```
UPDATE Funcionario  
SET Nome = "Beltrana"  
WHERE codigo = 3;
```



codigo	nome	cidade
1	Fulano	Rio Claro
2	Ciclana	São Paulo
3	Beltrana	Rio Claro

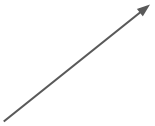
```
UPDATE Funcionario  
SET cidade = "Piracicaba"  
WHERE nome = "Fulano";
```



codigo	nome	cidade
1	Fulano	Piracicaba
2	Ciclana	São Paulo
3	Beltrana	Rio Claro


Remoção

```
DELETE FROM Funcionario  
WHERE nome = "Beltrana";
```



codigo	nome	cidade
1	Fulano	Rio Claro
2	Ciclana	São Paulo

```
DELETE FROM Funcionario  
WHERE nome = "Fulano" OR  
       cidade = "São Paulo";
```



codigo	nome	cidade

Consulta

Considerar a tabela Aluno:

codigo	nome	sexo	idade	cidade
1	Alberto	M	23	Rio Claro
2	Bernardo	M	25	São Paulo
3	Carlos	NULL	22	Piracicaba
4	Debora	F	27	Rio Claro
5	Eliane	F	30	Rio de Janeiro
6	Flávia	F	30	Rio de Janeiro
7	Gabriel	M	34	Campinas

Consulta

```
SELECT *
```

```
FROM Funcionario
```

```
WHERE idade > 28;
```

codigo	nome	sexo	idade	cidade
5	Eliane	F	30	Rio de Janeiro
6	Flávia	F	30	Rio de Janeiro
7	Gabriel	M	34	Campinas

Consulta

```
SELECT nome, idade  
FROM Funcionario  
WHERE sexo = 'M';
```

nome	idade
Alberto	23
Bernardo	25
Gabriel	34

```
SELECT codigo, nome  
FROM Funcionario  
WHERE cidade = 'Rio Claro';
```

codigo	nome
1	Alberto
4	Debora

JDBC

- **API**/Conjunto de drivers que permite a conexão ao Banco de Dados
- JDBC faz a implementação da linguagem **SQL** para acesso, manipulação e consulta ao BD.
- JDBC permite conexões com diferentes tipos de Bancos de Dados. Com isso, devemos especificar o BD utilizado, bem como a String/URL de conexão quando instanciando o JDBC.

RDBMS	Formato de URL de banco de dados
MySQL	<code>jdbc:mysql:// nomeDoHost: númeroDePorta /nomeDoBancoDeDados</code>
ORACLE	<code>jdbc:oracle:thin:@ nomeDoHost: númeroDePorta :nomeDoBancoDeDados</code>
DB2	<code>jdbc:db2: nomeDoHost: númeroDePorta /nomeDoBancoDeDados</code>
PostgreSQL	<code>jdbc:postgresql:// nomeDoHost: númeroDePorta /nomeDoBancoDeDados</code>
Java DB/Apache Derby	<code>jdbc:derby:nomeDoBancoDados</code> (incorporado) <code>jdbc:derby://nomeDoHost: númeroDaPorta /nomeDoBancoDeDados</code> (rede)
Microsoft SQL Server	<code>jdbc:sqlserver://nomeDoHost: númeroDaPorta ;databaseName=nomeDoBancoDeDados</code>
Sybase	<code>jdbc:sybase:Tds: nomeDoHost: númeroDePorta /nomeDoBancoDeDados</code>

JDBC

- Com isso, JDBC pode se conectar ao BD definido.
- Para utilizarmos o JDBC, devemos ter uma **instância de BD** (qual base de dados iremos nos conectar), bem como o respectivo Driver no computador. Carregamos o driver com a classe `java.sql.DriverManager` e no método `Class.forName()`.
- Para testes, podemos instalar o **MySQL Server** na máquina e configurar o conector `connector/J` ou qualquer outra base de dados e realizar as devidas configurações. Conferir a documentação para realizar a configuração correta de acordo com o ambiente escolhido.

JDBC

- **Statement**: classe utilizada para tratar as operações SQL, é ela quem irá trafegar as operações pela conexão. Instanciamos um objeto `Statement` a partir de `objConexao.createStatement()`
- **Statement.executeQuery(<operacao>)**: operação é o comando SQL que queremos aplicar. Este método retorna um `ResultSet` do resultado da solicitação de consulta
- **Statement.executeUpdate(<operacao>)**: utilizado para executar operação de manipulação da estrutura (`CREATE TABLE`, por exemplo) ou execuções que não têm um retorno (`ResultSet`).
- **ResultSet**: resultado de operações. Tipicamente, temos os resultados das consultas em `ResultSets` e iteramos sobre ele para visualizar e manipular os resultados das consultas no BD.
- **PreparedStatement**: similar ao `Statement`, porém é utilizado para comunicações parametrizadas/pré-compiladas, sendo mais eficiente para estes casos.

```
private static void criaTabela() {  
    try {  
        st.executeUpdate("CREATE TABLE  
Aluno(codigo int PRIMARY KEY, nome  
VARCHAR(30));");  
        System.out.println("Tabela Aluno criada  
com sucesso");  
    } catch (SQLException e) {  
        System.err.println("Erro na criação da  
tabela Aluno");  
        e.printStackTrace();  
    }  
}
```

```
private static void insereAluno(int chave, String nome) {  
    try {  
        st.executeUpdate(String.format("INSERT INTO  
Aluno VALUES('%d', '%s')", chave, nome));  
        System.out.println("Dados inseridos com  
sucesso");  
    } catch (SQLException e) {  
        System.err.println("Erro na inserção de dados");  
        e.printStackTrace();  
    }  
}
```

```
private static void selectAlunos() {  
    try {  
        ResultSet rs = st.executeQuery('SELECT * FROM Aluno');  
        while (rs.next()) {  
            int codigo = rs.getInt("codigo");  
            String nome = rs.getString("nome");  
            System.out.printf("Codigo = %d, Nome = %s%n", codigo,  
nome);  
        }  
        System.out.println("Fim da consulta de alunos");  
    } catch (SQLException e) {  
        System.err.println("Erro na seleção de todos os alunos");  
        e.printStackTrace();  
    }  
}
```

```
private static void selectAlunoPorChave(int chave) {  
    try {  
        ResultSet rs = st.executeQuery(String.format("SELECT *  
FROM Aluno WHERE codigo = " + chave));  
        while (rs.next()) {  
            int codigo = rs.getInt("codigo");  
            String nome = rs.getString("nome");  
            System.out.printf("Codigo = %d, Nome = %s%n", codigo,  
nome);  
        }  
        System.out.println("Fim da consulta de alunos pela chave "  
+ chave);  
    } catch (SQLException e) {  
        System.err.println("Erro na consulta de dados");  
        e.printStackTrace();  
    }  
}
```

```
private static void PSSelectAlunoPorChave(int chave) {  
    try {  
        PreparedStatement ps = db.prepareStatement("SELECT * FROM  
Aluno WHERE codigo = ?");  
        ps.setInt(1, chave);  
        ResultSet rs = ps.executeQuery();  
        while (rs.next()) {  
            int codigo = rs.getInt("codigo");  
            String nome = rs.getString("nome");  
            System.out.printf("Codigo = %d, Nome = %s\n", codigo,  
nome);  
        }  
        System.out.println("Fim da consulta de alunos com  
preparedStatement pela chave " + chave);  
    } catch (SQLException e) {  
        System.err.println("Erro na consulta de dados");  
        e.printStackTrace();  
    }  
}
```

```
private static void atualizaNome(int chave, String novoNome)
{
    try {
        st.executeUpdate(String.format("UPDATE Aluno SET
nome='%s' WHERE codigo = '%d'", novoNome, chave));
        System.out.println("Dados atualizados com
sucesso");
    } catch (SQLException e) {
        System.err.println("Erro na atualização de
dados");
        e.printStackTrace();
    }
}
```



```
private static void excluirAluno(int chave) {  
    try {  
        st.executeUpdate(String.format("DELETE FROM Aluno  
WHERE codigo = '%d'", chave));  
        System.out.println("Comando de exclusão executado  
com sucesso");  
    } catch (SQLException e) {  
        System.err.println("Erro na exclusão de dados");  
        e.printStackTrace();  
    }  
}
```

```
public static void main(String[] args) {
    String url = "jdbc:mysql://localhost:3306/aula08";
    String driverName = "com.mysql.cj.jdbc.Driver";
    try {
        Class.forName(driverName);
    } catch (ClassNotFoundException e1) {
        e1.printStackTrace();
    }
    String usuario = "USUARIO";
    String senha = "SENHA";
    try {
        db = DriverManager.getConnection(url, usuario, senha);
    } catch (SQLException e) {
        e.printStackTrace();
    }
    try {
        st = db.createStatement();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    criaTabela();
    insereAluno(1, "marcos");
    insereAluno(2, "nair");
    insereAluno(3, "orlando");
    selectAlunos();
    selectAlunoPorChave(2);
    atualizaNome(2, "NATALIA");
    selectAlunos();
    excluirAluno(2);
    selectAlunos();
}
```

Exemplo

Objetivo: interface visual para manipular a base de dados de livros

- Criar Banco de Dados **books**
- Rodar **books.sql** (seleção de comandos para criar as tabelas e alimentá-las)
- Classes ResultSetTableModel e **DisplayQueryResults**