



High performance. Delivered.

## Application Delivery Fundamentals: Java

Module 13: Core Java API – JDBC

## Module Objectives

- **No final deste módulo, você será capaz de:**
  - Use a tecnologia JDBC e descreva seus recursos.
  - Use drivers de banco de dados e descreva seus recursos.
  - Estabeleça uma conexão com um banco de dados usando o objeto Connection.
  - Crie e execute instruções SQL em Java e manipule dados resultantes de instruções SQL executadas.
  - Defina o conceito de objetos de acesso ao banco de dados.



# JDBC

---

**“JDBC”**



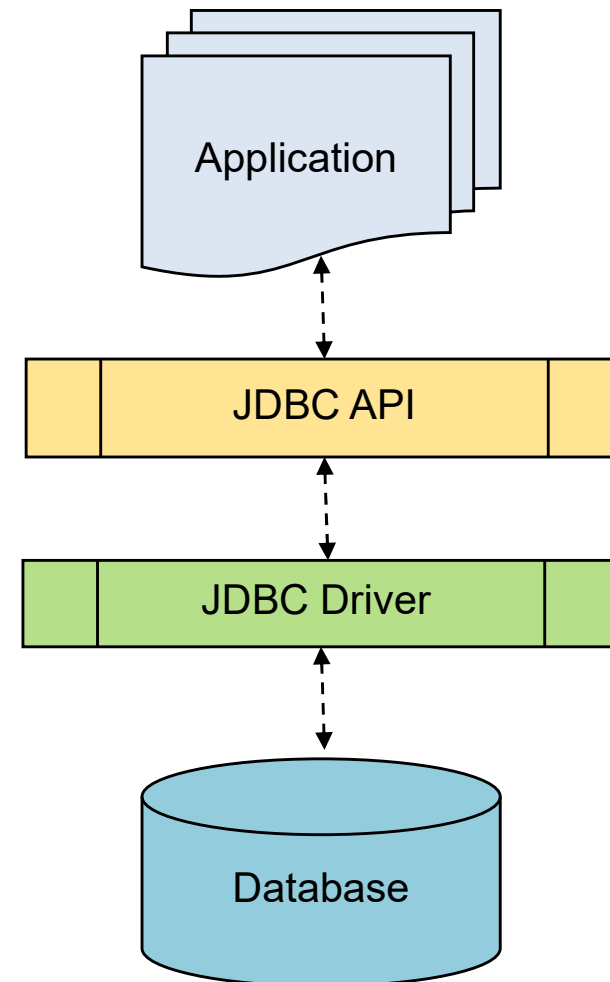
# JDBC Overview

- JDBC (Java DataBase Connectivity) é o padrão do setor Java para conectividade independente de banco de dados entre a linguagem Java e outros bancos de dados.
- O JDBC fornece uma API abrangente que fornece ao aplicativo a capacidade de conectar-se a bancos de dados, enviar instruções SQL e resultados do processo.



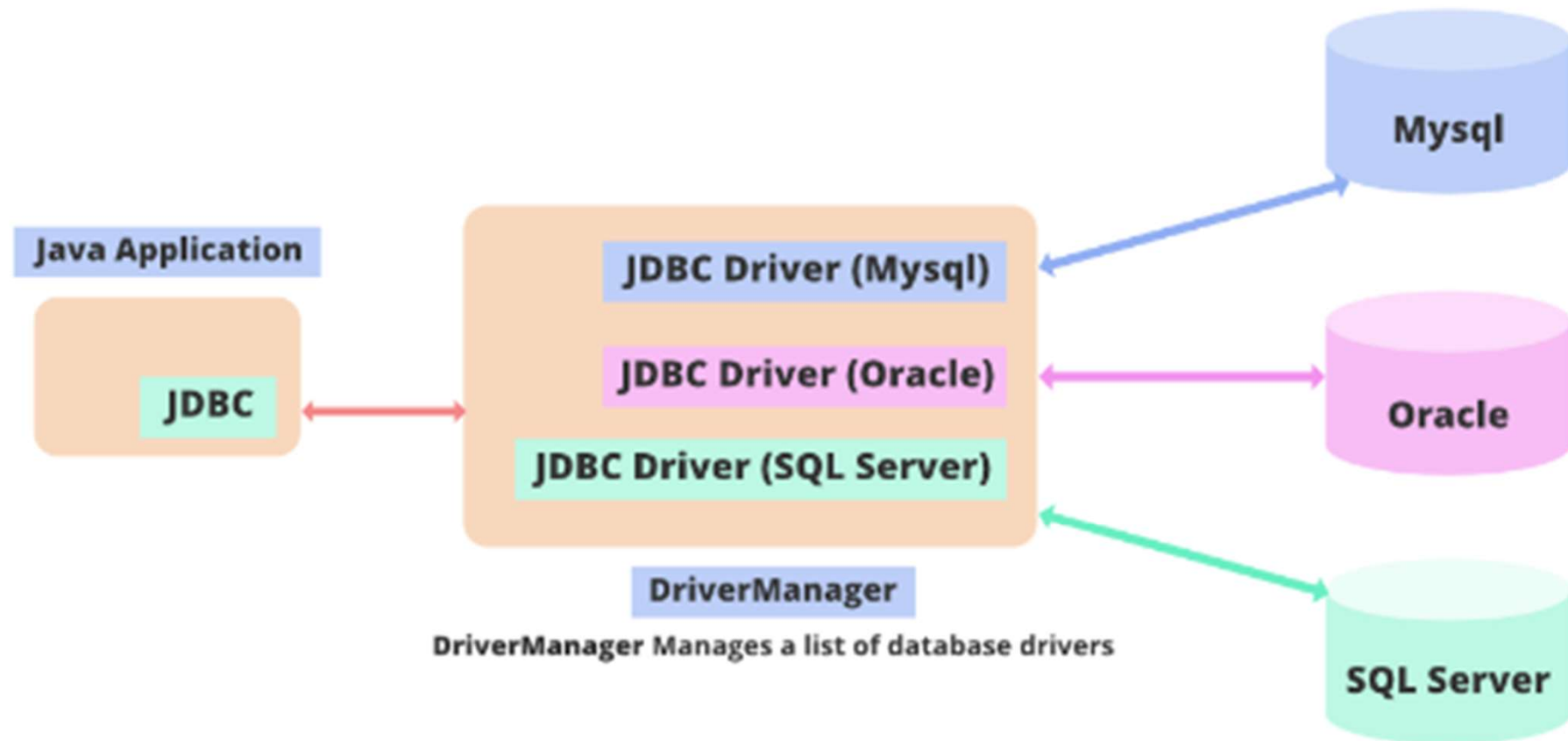
## Database Drivers

- Drivers de banco de dados ou drivers JDBC são necessários para conectar-se a diferentes bancos de dados.
- O JDBC requer drivers diferentes para diferentes bancos de dados.
- Drivers JDBC:
- Forneça a conexão com o banco de dados.
- Implemente o protocolo necessário para enviar consultas e recuperar resultados.



## Database Drivers

- Drivers de banco de dados ou drivers JDBC são necessários para conectar-se a diferentes bancos de dados.



## Basic SELECT Statement (1 of 3)

- Use a instrução SELECT para recuperar dados de uma ou mais tabelas :

```
SELECT <column(s) >  
FROM <table(s)>  
[WHERE <condition>]  
[ORDER BY <column(s) [ASC|DESC]>]
```

- **(select) column** – os nomes das colunas na tabela a serem selecionadas.
- **Table** – os nomes das tabelas onde os dados devem ser selecionados.
- **condition** – identifica as linhas a serem selecionadas que contêm expressões, restrições, subconsultas e operadores de comparação.
- **(order by) column** – os nomes das colunas usadas para classificar.

## Basic SELECT Statement (2 of 3)

---

- Nota: WHERE e ORDERBY são opcionais. Opcional é representado por[ ... ]
  - Ordem usando implicitamente ascendente(ASC)
    - SELECT firstname FROM user;
    - SELECT firstname, lastname FROM user WHERE firstname = 'Luis' ORDER BY lastname;
    - SELECT firstname FROM user ORDER BY firstname DESC



## Basic SELECT Statement (3 of 3)

- Para recuperar dados de duas ou mais tabelas, o nome da coluna deve ser anexado pelo nome da tabela.

### **Syntax:**

```
<table name>.<column name>
```

### **Example:**

```
SELECT patients.name, meds.name FROM patients,meds  
WHERE patients.med_id = meds.med_id;
```

- O alias é usado para encurtar as instruções.

### **Syntax:**

```
<table name> <alias>  
<alias>.<column name>
```

### **Example:**

```
SELECT a.name, b.name FROM patients a, meds b  
WHERE a.med_id = b.med_id;
```

## Basic INSERT Statement

- Use a instrução INSERT para adicionar dados a uma tabela específica :

```
INSERT INTO <table>  
          [ (<column(s)> ) ]  
VALUES    (<value(s)>)
```

- **table** - o nome da tabela onde os dados devem ser inseridos.
- **column** - os nomes das colunas na tabela a serem preenchidas.
- **value** - os valores correspondentes a serem inseridos para cada coluna especificada.
- Note: column is optional. Optional is represented by [ ... ]
  - INSERT INTO user VALUES ('Luis', 'Chua')
  - INSERT INTO user (firstname, lastname) VALUES ('Luis', 'Chua')

## Basic UPDATE Statement

- Use a instrução UPDATE para alterar os valores da coluna em uma tabela específica :

```
UPDATE <table>  
SET    <column> = <value>  
        [, <column> = <value>, ...]  
[WHERE <condition>]
```

- **table** - O nome da tabela.
- **column** - O nome da coluna onde o novo valor deve ser atribuído.
- **value** – o novo valor para a coluna.
- **condition** - identifica as linhas a serem atualizadas que contêm expressões, restrições, subconsultas e operadores de comparação.

## Basic UPDATE Statement (cont.)

---

- Note: condição é opcional. Opcional é representado por [ ... ]
  - UPDATE user SET firstname = 'Aleli';
  - UPDATE user SET firstname = 'Aleli', lastname = 'Zapanta'  
WHERE firstname = 'Manny';



## Basic DELETE Statement

- Use DELETE para excluir linhas em uma tabela específica :

```
DELETE FROM <table>  
[WHERE <condition>] ;
```

- **table** - O nome da tabela.
- **condition** - identifica as linhas a serem excluídas, que contêm expressões, restrições, subconsultas e operadores de comparação.
  - DELETE FROM user WHERE firstname = 'Manny';

## Retrieving a Connection Object

---

- O objeto **Connection** define uma conexão ou sessão com um banco de dados específico.
- É onde as instruções SQL são executadas e os resultados são retornados.
- Antes que o objeto **Connection** possa ser usado, a classe `java.sql.Connection` deve ser importada.

## Retrieving a Connection Object (cont.)

- Um driver de banco de dados pode ser carregado usando o **Class.forName()** method.

```
Syntax :      Class.forName("JDBCDriver_Name");  
Example :      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

- O objeto **Connection** possui três (3) partes importantes :
  - The URL or location of the data source
  - The username
  - The password

## Retrieving a Connection Object (cont.)

- Uma conexão pode ser estabelecida usando o **getConnection()** method.
- Criando uma conexão usando a URL, nome de usuário e senha como parâmetros.



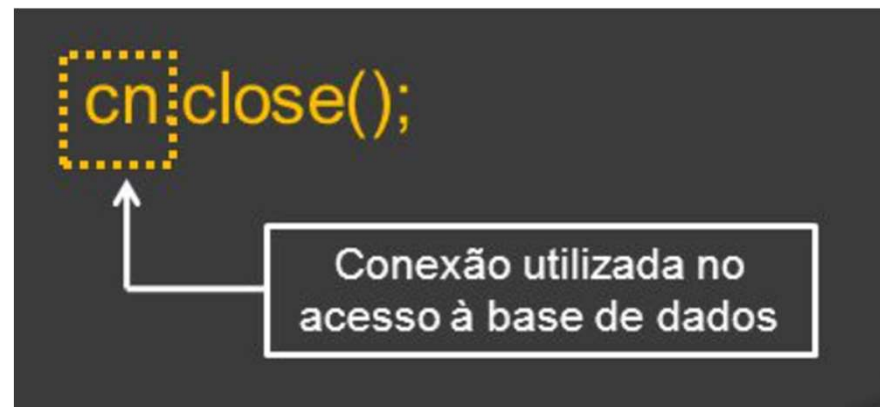
```
Connection cn = DriverManager.getConnection
( URL, username, password );
```



## Retrieving a Connection Object (cont.)

- Criando uma conexão usando a URL; Nesse caso, o URL já inclui o nome de usuário e a senha.

```
Connection cn = DriverManager.getConnection ( URL );
```



Refer to the ConnectionSample.java sample code.

## Creating Query Statements

---

- Cada operação na base de dados é definida por uma instrução na linguagem SQL, chamada e **Statement**.
- Um **Statement** é a mensagem que o JDBC envia à fonte de dados para manipular ou solicitar dados da fonte de dados.
- Antes que um objeto **Statement** possa ser usado, a classe `java.sql.Statement` deve ser importada.

## Creating Query Statements (cont.)

- Uma instrução é criada usando a seguinte sintaxe:

- Utilizado para execução de simples instruções SQL

Conexão utilizada no acesso à base de dados

```
Statement st = cn.createStatement();
```

```
st.executeUpdate("UPDATE tab_func SET salario = salario + 200");
```

```
st.executeUpdate("DELETE FROM tab_func");
```

```
st.executeUpdate("INSERT INTO tab_cargo (nome) VALUES ('RH')");
```

```
st.executeUpdate("CREATE TABLE tab_produto (...");
```

```
st.close();
```

## Creating Query Statements (cont.)

- Uma instrução precisa usar um objeto **Connection** para identificar a qual conexão à instrução será associada.

### Syntax:

```
Statement <identifier> = <Connection_Object>.createStatement();
```

### Sample:

```
Connection myConn = DriverManager.getConnection(...);  
Statement myStatement = myConn.createStatement();
```



## Creating Query Statements (cont.)

- A **Statement** pode ser executado utilizando tanto **execute()**, **executeQuery()** e **executeUpdate()** metodos.

### Syntax:

```
<Statement_Object>.executeQuery("SQL Statement goes here");
```

### Sample:

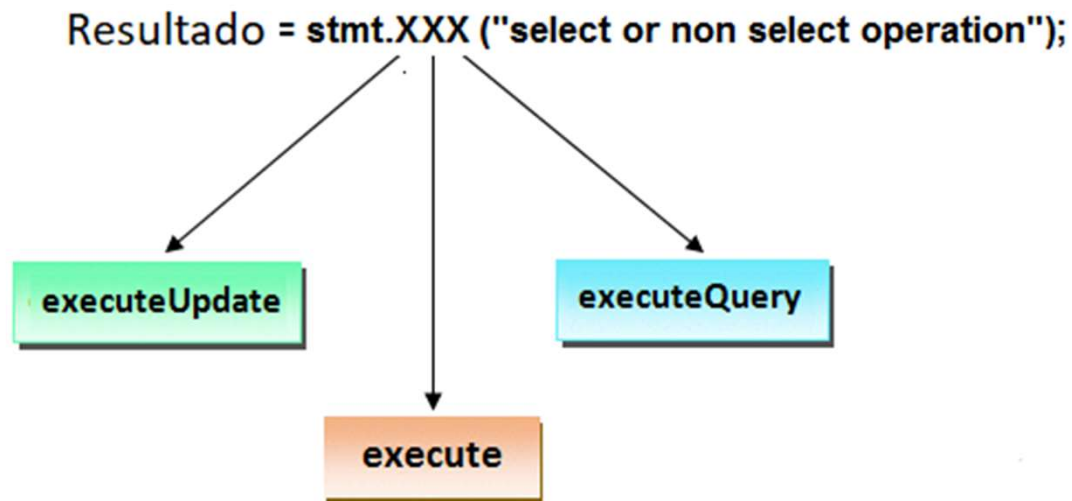
```
Connection myConn = DriverManager.getConnection(...);  
Statement myStatement = myConn.createStatement();  
myStatement.executeQuery("Select * from aTable");
```



Refer to the StatementSample.java sample code.

## Creating Query Statements (cont.)

- Após à execução de uma instrução, ela retorna diferentes tipos de dados, dependendo do tipo de execução usado.



Comando	Retorno
execute ()	booleano
executeUpdate ()	Inteiro
executeQuery ()	ResultSet



Refer to the InsertSample.java, UpdateSample.java and DeleteSample.java sample codes.

## Creating Query Statements (cont.)

- Após à execução de uma instrução, ela retorna diferentes tipos de dados, dependendo do tipo de execução usado.
- **execute ()** retorna um valor booleano e é usado para executar instruções SQL gravadas como um objeto String.
- **executeUpdate ()** retorna um valor int referente ao número de linhas afetadas e é usado para executar instruções SQL DML.
- **executeQuery ()** retorna um objeto ResultSet e é usado para executar instruções SELECT SQL.



Refer to the InsertSample.java, UpdateSample.java and DeleteSample.java sample codes.

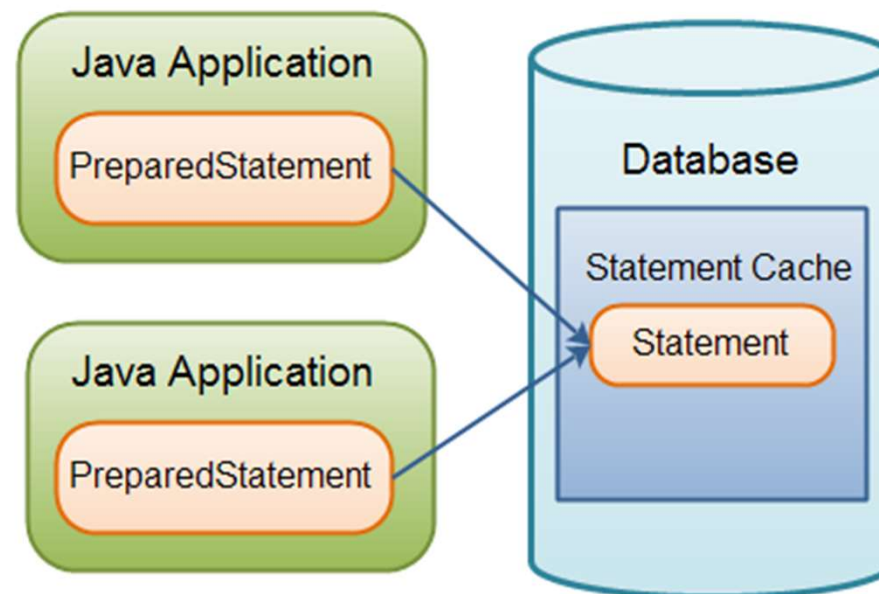
## Tabela de relação Tipo/Método

Tipo	Método
Tinyint	getByte
SmallInt	getShort
Integer	getInt
BigInt	getLong
Real	getFloat
Float	getDouble
Double	getDouble
VarChar	getString



## Using PreparedStatement

- **PreparedStatement** representa uma instrução SQL parametrizada, onde alguns de seus valores são dinamicamente assinalados pela aplicação.
- **PreparedStatement** é um objeto Java, que representa uma instrução SQL pré-compilada.
- Antes que um objeto **Statement** possa ser usado, a classe **java.sql.PreparedStatement** deve ser importada.



## Using PreparedStatement (cont.)

### Exemplo 1

Conexão utilizada no acesso à base de dados

```
PreparedStatement ps = cn.prepareStatement(
    "INSERT INTO tab_func (matricula, nome, salario) VALUES (?, ?, ?)");
```

```
ps.setInt(1, 1001);
ps.setString(2, "José Souza");
ps.setDouble(3, 3200.45);
```

```
ps.executeUpdate();
ps.close();
```



## Using PreparedStatement (cont.)

### Exemplo 2

Conexão utilizada no  
acesso à base de dados

```
PreparedStatement ps = cn.prepareStatement(
    "UPDATE tab_func SET salario = ? WHERE matricula = ?");
```

```
ps.setDouble(1, 3200.45);
```

```
ps.setInt(2, 1001);
```

```
ps.executeUpdate();
```

```
ps.setDouble(1, 7350.92);
```

```
ps.setInt(2, 1005);
```

```
ps.executeUpdate();
```

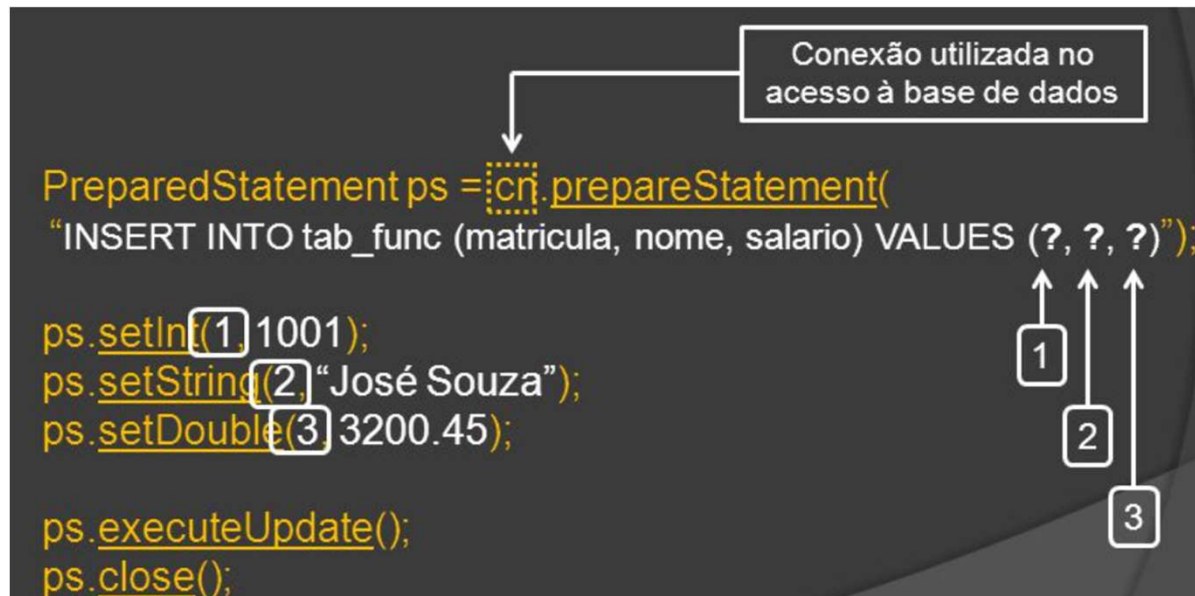
```
ps.close();
```

1

2

## Using PreparedStatement (cont.)

- O ponto de interrogação (?) Serve como curinga ou parâmetro e pode ser substituído por um valor.
- O curinga pode ser substituído usando os métodos apropriados para um tipo de dados específico, incluindo, entre outros:
  - `setString()`
  - `setInt()`
  - `setDouble()`
  - `setDate()`



```

PreparedStatement ps = cn.prepareStatement(
    "INSERT INTO tab_func (matricula, nome, salario) VALUES (?, ?, ?)");

ps.setInt(1, 1001);
ps.setString(2, "José Souza");
ps.setDouble(3, 3200.45);

ps.executeUpdate();
ps.close();
  
```

Conexão utilizada no acesso à base de dados

1 2 3



Refer to the PreparedStatementSample.java sample code.

## Using PreparedStatement (cont.)

- Assinala um parâmetro do tipo texto.

```
ps = cn.prepareStatement(
    "DELETE FROM tab_func WHERE cargo LIKE ?");
ps.setString(1, "Manuel");
ps.executeUpdate();
ps.close();
```

- Assinala um parâmetro numérico sem parte fracionária.

```
ps = cn.prepareStatement(
    "DELETE FROM tab_func WHERE matricula = ?");
ps.setInt(1, 7012);
ps.executeUpdate();
ps.close();
```



## Using PreparedStatement (cont.)

- Assinala um parâmetro booleano e armazena na base como 0 (false) ou 1 (true)

```
ps = cn.prepareStatement("UPDATE tab_func SET ativo = ?");
ps.setBoolean(1, true);
ps.executeUpdate();
ps.close();
```

- Assinala um parâmetro com o valor NULL

```
ps = cn.prepareStatement(
    "INSERT INTO tab_func (nome, salario) VALUES (?, ?)");

ps.setString(1, "Manuel silva");
ps.setNull(2);
ps.executeUpdate();
ps.close();
```

## Using PreparedStatement (cont.)

- Assinala um parâmetro do tipo `java.sql.Date`, que representa uma data (dia, mês e ano)

```
ps = cn.prepareStatement("UPDATE tab_func SET nasc = ?");
```

```
Calendar calendar = Calendar.getInstance();
```

```
Calendar.set(1992, 0, 25); /* dia 25/01/1992 */
```

```
java.sql.Date date = new java.sql.Date(calendar.getTimeInMillis());
```

```
ps.setDate(1, date);
```

```
ps.executeUpdate();
```

```
ps.close();
```



## Using PreparedStatement (cont.)

- Assinala um parâmetro do tipo `java.sql.Time`, que representa um horário (hora, minuto e segundo)

```
ps = cn.prepareStatement("UPDATE tab_func SET hr_entr = ?");
```

```
Calendar calendar = Calendar.getInstance();
```

```
Calendar.set(0, 0, 0, 12, 15, 30); /* 12:15:30 */
```

```
java.sql.Time time = new java.sql.Time(calendar.getTimeInMillis());
```

```
ps.setTime(1, time);
```

```
ps.executeUpdate();
```

```
ps.close();
```

## Using PreparedStatement (cont.)

- Assinala um parâmetro do tipo `java.sql.Timestamp`, que representa um instante no tempo (ano, mês, dia, hora, minuto e segundo)

```
ps = cn.prepareStatement("UPDATE tab_func SET nasc = ?");
```

```
Calendar calendar = Calendar.getInstance();
```

```
Calendar.set(1992, 0, 25, 12, 15, 30); /* 25/01/1992 12:15:30 */
```

```
Timestamp dateTime = new Timestamp(calendar.getTimeInMillis());
```

```
ps.setTimestamp(1, dateTime);
```

```
ps.executeUpdate();
```

```
ps.close();
```

## Using PreparedStatement (cont.)

- Assinala um parâmetro do tipo `java.io.InputStream`, que representa alguma informação binária, como uma foto, mp3, doc ou outro.

```
ps = cn.prepareStatement("INSERT INTO tab_foto (foto) VALUES (?)");
```

```
ps.setBinaryStream(1, new FileInputStream("C:\\image\\foto.jpg"));  
ps.executeUpdate();  
ps.close();
```

## Using PreparedStatement (cont.)

### ● Exemplo

```
cn = DriverManager.getConnection(...);
st = cn.createStatement();

try {
    cn.setAutoCommit(false);
    st.executeUpdate("INSERT INTO tab (...) VALUES (...)");
    st.executeUpdate("INSERT INTO tab (...) VALUES (...)");
    st.executeUpdate("DELETE FROM tab WHERE ...");
    st.executeUpdate("UPDATE tab SET ....");
    cn.commit();
} catch (Exception e) {
    cn.rollback();
}
```



## Using PreparedStatement (cont.)

```
ResultSet rs = st.executeQuery("SELECT * FROM tab_func");
```

```
rs.next();
codigo = rs.getInt("f_code");
nome = rs.getString("f_name");
salario = rs.getDouble("f_rmnt");
```

...

```
rs.next();
codigo = rs.getInt("f_code");
nome = rs.getString("f_name");
salario = rs.getDouble("f_rmnt");
```

...

```
rs.next();
codigo = rs.getInt("f_code");
nome = rs.getString("f_name");
salario = rs.getDouble("f_rmnt");
```



f_code	f_name	f_rmnt
1003	Manuel	1.253,86
1004	Joaquim	950,60
1008	Maria	1.530,15
1012	Henrique	6.530,45
1039	João	4.350,12
1112	Priscila	843,00
1128	Ricardo	7.815,26

## Using PreparedStatement (cont.)

```
ResultSet rs = st.executeQuery("SELECT * FROM tab_func");
```

```
while (rs.next()) {
    codigo = rs.getInt("f_code");
    nome = rs.getString("f_name");
    salario = rs.getDouble("f_rmnt");
```

```
    ...
```

```
}
```

```
rs.close();
```



f_code	f_name	f_rmnt
1003	Manuel	1.253,86
1004	Joaquim	950,60
1008	Maria	1.530,15
1012	Henrique	6.530,45
1039	João	4.350,12
1112	Priscila	843,00
1128	Ricardo	7.815,26

# Perguntas:

---

## O que é statement ?

Um Statement é a mensagem que o JDBC envia à fonte de dados para manipular ou solicitar dados da fonte de dados

## O que é prepared statement ?

É um objeto Java, que representa uma instrução SQL pré-compilada.



# Atividade - 8



## Gerenciamento de Biblioteca com JDBC

- Desenvolver um aplicativo Java que se conecte ao MySQL para gerenciar uma biblioteca. O aplicativo deve permitir adicionar, visualizar, atualizar e deletar livros no banco de dados.
- **Configuração do Banco de Dados:**
- Crie um banco de dados chamado **biblioteca**.
- Crie uma tabela chamada **livros** com as seguintes colunas:
  - id (INT, chave primária, auto-incremento)
  - titulo (VARCHAR(255))
  - autor (VARCHAR(255))
  - ano\_publicacao (INT)



# Atividade - 8



## Gerenciamento de Biblioteca com JDBC

- **Operações CRUD:**

- **Adicionar Livro:**

- Adicionar novos livros ao banco de dados.

- **Visualizar Livros:**

- Exibir uma lista de todos os livros no banco de dados.

- **Atualizar Livro:**

- Atualizar as informações de um livro existente.

- **Deletar Livro:**

- Deletar um livro do banco de dados.

- **Conexão com o Banco de Dados:**

- Implemente uma classe **Conexao** para gerenciar a conexão com o banco de dados.



## Questions and Comments

- What questions or comments do you have?

