

# ***FACEC***

## ***F a c u l d a d e***

# Cont. Linguagem SQL

Professor: Yuri Ferreira

- Revisão aula anterior:
  - Conectar no Banco;
  - Criar Tabela;
  - Inserir dados;
  - Atualizar dados;

## ➤ Conteúdo:

- **CallableStatement**: funções e procedimentos;
- Percorrer resultado de uma consulta;
- **ResultSet**: Atualizar dados e mover cursor;
- Análise de **Metadados**;
- Movimentação e **Transações**;
- **Encerrando** a conexão;
- Exercícios;

- Projeto Java:
- 2) Executando instruções SQL
  - A Interface **CallableStatement**;
  - Permite chamar **procedimentos/funções** armazenados;
  - Ex:

```
String functionCall = "{?= call tam_departamento(?)}";  
CallableStatement cstmt = con.prepareStatement(functionCall);  
cstmt.setInt(2, 1);  
cstmt.registerOutParameter(1, Types.VARCHAR);  
cstmt.execute();  
String tamDepartamento = cstmt.getString(1);
```

- Projeto Java:
- 3) Analisando resultados da Consulta
  - O **resultado** após efetuar uma consulta é representado pelo objeto **ResultSet**;
  - ResultSet possui uma estrutura semelhante de uma **tabela com linhas e colunas**;
  - **ResultSet** utiliza o método **next()** para ir para a próxima linha;

- Projeto Java:
- 3) Analisando resultados da Consulta
  - Criar uma **Classe** Java Funcionário e preenche-la;
  - **Percorrendo** o ResultSet:

```
ResultSet res = stm.executeQuery("Select * from funcionario");

while (res.next()) {
    Funcionario fun = new Funcionario();
    fun.setCpf(res.getString("cpf"));
    ...
}
```

- Projeto Java:
- 4) Utilizando o **ResultSet** para atualizar e percorrer os dados nos 2 sentidos;

```
Statement stm2 = con.createStatement(  
    ResultSet.TYPE_SCROLL_INSENSITIVE,  
    ResultSet.CONCUR_UPDATABLE);
```

- Após isso é feito uma **consulta** sobre os dados para obter o **ResultSet**:

```
String selectSql = "select *" +  
    " FROM public.funcionario";  
ResultSet res2 = stm2.executeQuery(selectSql);
```



- Projeto Java:
- 4) Utilizando o ResultSet para atualizar e percorrer os dados nos 2 sentidos;
- Para **percorrer** sobre as linhas, pode se utilizar os seguintes **métodos**:

```
res2.first();  
// ou estes  
// last(), beforeFirst(), beforeLast(),  
// next(), previous(),  
// moveToInsertRow(), moveToCurrentRow(),  
// absolute(int row), relative(int nrRows);
```

- Projeto Java:
- 4) Utilizando o ResultSet para atualizar e percorrer os dados nos 2 sentidos;
- Após posicionar o **cursor** na linha desejada a ser atualizada, basta utilizar o método **UpdateX**, onde X é o tipo do dado: String, Date, Double, Int, etc;
- Ex:

```
res2.updateDouble("salario", 744444);  
res2.updateString("primeiro_nome", "Joao");  
...
```

- Projeto Java:
- 4) Utilizando o ResultSet para atualizar e percorrer os dados nos 2 sentidos;
- Para **persistir** as modificações no ResultSet pode utilizar um destes métodos:
  - **updateRow()** - para persistir as alterações na linha atual para o banco de dados;
  - **insertRow(), deleteRow()** - para adicionar uma nova linha ou excluir a atual do banco de dados;
  - **refreshRow()** - para atualizar o *ResultSet* com quaisquer alterações no banco de dados;
  - **cancelRowUpdates()** - para cancelar as alterações feitas na linha atual;

## ➤ Projeto Java:

➤ 4) Utilizando o ResultSet para atualizar e percorrer os dados nos 2 sentidos;

➤ Ex:

```
public void atualizaNovoSalarioFuncionario(String cpf,  
double novoSalario) throws SQLException {
```

```
Statement stm2 = con.createStatement(  
ResultSet.TYPE_SCROLL_INSENSITIVE,  
ResultSet.CONCUR_UPDATABLE);
```

```
String selectSql = "select *" +  
" FROM public.funcionario" +  
" WHERE cpf='" + cpf + "'";
```

```
ResultSet res2 = stm2.executeQuery(selectSql);  
res2.next();  
res2.updateDouble("salario", novoSalario);  
res2.updateRow();  
}
```

- Projeto Java:
- 5) **Análise de Metadados:**
- A interface fornece informações gerais sobre o banco de dados, como **tabelas, procedimentos armazenados**, visões, entre outros;
- Ex: Como obter as tabelas do banco:

```
public void listGetTables() throws SQLException {  
    DatabaseMetaData dbmd = con.getMetaData();  
  
    String[] types = {"TABLE"};  
  
    ResultSet tablesResultSet = dbmd.getTables("empresa", "", "%", types);  
    while (tablesResultSet.next()) {  
        System.out.println(tablesResultSet.getString("TABLE_NAME"));  
    }  
}
```

- Projeto Java:
- 5) **Análise de Metadados:**
- Pode ser utilizado a interface **ResultSetMetaData** para encontrar informações sobre determinado **ResultSet**;
- Ex: 

```
public void imprimeAtributosResultSet() throws SQLException {  
    ResultSet res = stm.executeQuery("Select * from funcionario");  
  
    ResultSetMetaData rsmd = res.getMetaData();  
    int qtdeCols = rsmd.getColumnCount();  
  
    for (int i=1; i <= qtdeCols; i++) {  
        System.out.println(i + " " + rsmd.getColumnName(i));  
    }  
}
```

- Projeto Java:
- 6) **Movimentação de Transações:**
  - Por padrão cada instrução SQL é confirmada logo após ser concluída;
  - As vezes se faz necessário ter o controle para confirmar a transação programaticamente;
  - Em casos que deseja se manter a **consistência dos dados**, por exemplo, confirmar uma transação se uma transação anterior tiver sido concluída com êxito;
  - Métodos Interface Connection:
    - SetAutoCommit(false), Commit() e Rollback();

➤ Projeto Java:

➤ 6) Movimentação de Transações:

➤ Ex:

```
String updateDepto = "update funcionario" +  
    " set num_depto=?" +  
    " where cpf=? ";  
Update 1 PreparedStatement pstmt = con.prepareStatement(updateDepto);  
pstmt.setInt(1, numDepto);  
pstmt.setString(2, cpf);
```

```
String updateSalario = "update funcionario" +  
    " set salario= ? " +  
    " where cpf=? ";  
Update 2 PreparedStatement pstmt2 = con.prepareStatement(updateSalario);  
pstmt2.setDouble(1, salario);  
pstmt2.setString(2, cpf);
```



- Projeto Java:
- 6) Movimentação de Transações:
- Ex:

Controle de Execução  
das Transações

```
boolean autoCommit = con.getAutoCommit();
try {
    con.setAutoCommit(false);
    pstmt.executeUpdate();
    pstmt2.executeUpdate();
    con.commit();
} catch (SQLException exc) {
    con.rollback();
} finally {
    con.setAutoCommit(autoCommit);
}
```

- Projeto Java:
- 7) **Encerrando a Conexão:**
- Quando não estiver sendo utilizado a conexão, deve ser encerrada para liberação de recursos do banco de dados;
- Evitar consumo de memória no BD;
- O Encerramento pode ser feito através do método:

```
// Objeto Connection  
con.close();
```

## ➤ Exercícios:

- 1) Crie uma Classe **Departamento** que represente os dados da tabela departamento e em seguida crie um método Java na Classe Service que retorne uma lista de todos departamentos do banco;
- 2) Criar um método Java que **atualiza** o gerente da tabela Departamento alterando o cpf e a data de início. O método deve receber 2 parâmetros, o número do departamento a ser atualizado, um objeto do tipo Departamento;
- 3) Criar um método para **excluir** o departamento com o nro\_depto passado como parâmetro;

## ➤ Referências:

- SILBERSCHATZ, A.; KORTH, F.; SUDARSHA, S. Database System Concepts. 6. ed. Nova York: MC Graw Hill, 2011.
- ELMASRI, R.; NAVATHE B. Sistemas de banco de dados. 6. Ed. São Paulo, SP: Pearson Addison-Wesley, 2011.