Sistemas de Bancos de Dados
Aula 10 – Criando e Populando Tabelas

Nesta atividade você trabalhará como criar e popular uma tabela em um banco de dados. A criação de uma tabela em um banco pode ser feita utilizando comandos DDL embutidos na linguagem java, bem como a povoamento da tabela. Proponho aqui que a criação de uma nova tabela ocorra utilizando a interface gráfica do programa pgadmin3 por dois motivos: (i) a tabela que criaremos é pequena e será criada apenas uma vez. (ii) Os tipos de dados possíveis para colunas de uma tabela existem em uma variedade enorme, bem como é enorme a quantidade de configurações adicionais que podem ser feitas nos campos de uma tabela (não aceitar valores nulos, preencher com valor *default*, …, entre outros). Dessa forma, se quisermos utilizar um comando DDL dentro do java para criar a tabela será inevitável consultar a interface gráfica do pgadmin3 para ver as possibilidades de criação da tabela. Entretanto, caso você tenha a necessidade que um programa crie tabelas automaticamente no computador de um usuário, então não há como evitar a criação de um código java responsável em criar a tabela. Para popular os dados na tabela, também poderíamos utilizar um comando *psql* passando o arquivo de dados pelo parâmetro *-f* com as instruções de inserção de dados em uma tabela recém-criada. Na atividade de hoje faremos a utilização de um código DML, embutido em um programa java, para efeitos de treinar a possibilidade de fazer com que um programa que está sendo instalado na máquina de um usuário seja o responsável por criar e popular as tabelas automaticamente durante, por exemplo, a instalação do Sistema de Banco de Dados na máquina desse usuário. Isso mesmo, a mesma metodologia para criar a tabela também é utilizada para popular a tabela. A principal diferença do método utilizado nesta atividade para criar e popular tabelas, quando comparado à última atividade, é a utilização de métodos JDBC que não necessitam retornar tuplas como resultado da execução de uma ''consulta'' na SQL. Antes de embutirmos o código java para popular uma tabela, faremos a execução deste código ''isolado'' das classes do tutorial, isso mesmo, faremos um teste de leitura dos dados a serem inseridos no banco a partir de um arquivo-texto e para isso não vamos utilizar nenhuma classe pré-definida no tutorial do JDBC, mas tão somente a pura linguagem java. Uma vez que seu código tenha funcionado então partimos para modificá-lo a fim de que possa ser embutido junto das classes do tutorial JDBC. Bem, então vamos ao trabalho. Um lembrete muito importante: se você deixar todo este tutorial para ser executado no último dia permitido para envio do relatório da atividade, eu receio que talvez você não vá conseguir terminar a tempo.

1) Nesta atividade vamos utilizar uma nova versão do banco da Instituição Bancária (IB). Não adianta tentar inserir estes dados no banco das aulas anteriores, visto que acontecerão erros de integridade referencial para todas as linhas da nova tabela que vamos criar e popular. Portanto, baixe o arquivo *IB2.dump*. Na sequência você criará um banco vazio para importar o *dump* para o novo BD de nome IB2:
    1. https://www.dropbox.com/s/6zuffauqzvd4fde/IB2.dump
2) Criar um banco de dados vazio na interface do pgadmin3 de nome *IB2*;
3) Executar a carga do arquivo *IB2.dump* no banco IB2;
    1. *psql -h localhost -p 5432 -U postgres IB2 -f IB2.dump*
4) Baixar o arquivo *debito-populate-table.txt* do link abaixo para a pasta raiz do tutorial do JDBC:
    1. https://www.dropbox.com/s/3nommkws4zqgvgq/debito-populate-table.txt
    2. Listar o conteúdo do arquivo baixado com *cat* ou editando-o com *gedit*;

Perceba que este arquivo possui centenas de linhas separadas por tabulação. A próxima tarefa compreende a leitura desses dados em um *loop* e execução de um comando *insert into* para cada linha e assim popular a tabela com estes dados. Para tanto disponibilizei um código de exemplo para ler dados tabulados de um arquivo e imprimir o resultado na tela. Façamos uma execução deste código para vê-lo em ação:

5) Crie um arquivo vazio com o *gedit*, insira o seguinte código no arquivo vazio e salve-o com o nome de *ReadFile.java*, na mesma pasta na qual o arquivo *debito-populate-table.txt* foi salvo:

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.Scanner;

public class ReadFile {
public static void main(String[] args) throws IOException {
BufferedReader inputStream = null;
Scanner scanned_line = null;
String line;
String[] value;
value = new String[7];
int countv;
try {
       inputStream = new BufferedReader(new FileReader("debito-populate-table.txt"));
       while ((line = inputStream.readLine()) != null) {
              countv=0;
              System.out.println("<<");
              //split fields separated by tab delimiters
              scanned_line = new Scanner(line);
              scanned_line.useDelimiter("\t");
              while (scanned_line.hasNext()) {
                     System.out.println(value[countv++]=scanned_line.next());
              } //while
              if (scanned_line != null) { scanned_line.close(); }
              System.out.println(">>");
              System.out.println("insert into debito (numero_debito, valor_debito,
motivo_debito, data_debito, numero_conta, nome_agencia, nome_cliente) " + "values (" +
value[0] +", "+ value[1] +", "+ value[2] +", '"+ value[3] +"', "+ value[4] +", '"+ value[5]
+"', '"+ value[6] + "');" );
       } //while
} finally { if (inputStream != null) { inputStream.close(); } } //if & finally
} //main
}//class
```

6) Compile e execute o arquivo *ReadFile.java:*

   1. *javac ReadFile.java*
   2. *java ReadFile*

7) Utilize a interface do *pgadmin3* para criar a seguinte tabela no banco:

Nome da tabela: debito

Nome da Coluna	Tipo	Modificadores
numero_debito	integer	not null
valor_debito	double precision	not null
motivo_debito	smallint	
data_debito	date	
numero_conta	integer	
nome_agencia	character varying(50)	
nome_cliente	character varying(80)	

Índices:

Nome: pk_debito, **Tipo:** PRIMARY KEY, **Sobre:** (numero_debito)

Restrições de chave estrangeira:

Nome: fk_debito, **Tipo:** FOREIGN KEY **Sobre:** (numero_conta, nome_agencia, nome_cliente), **Referenciando:** conta(numero_conta, nome_agencia, nome_cliente)

Perceba que ao final temos apenas uma tabela sem dado algum e será necessário realizar um carga de dados na tabela.

8) Após ter criado manualmente a tabela, agora é necessário populá-la com os dados do arquivo *debito-populate.txt*. Crie um método na classe *MyQueries* para executar tal operação. Utilize o arcabouço do código abaixo para este propósito.

```
public static void populateTable(Connection con) throws SQLException {
  Statement stmt = null;
  String create = "";

  try {
    stmt = con.createStatement();
    System.out.println("Executando DDL/DML:");
    stmt.executeUpdate(create);
  } catch (SQLException e) {
    JDBCUtilities.printSQLException(e);
  } finally {
    if (stmt != null) { stmt.close(); }
  }
}
```

Perceba que o código aqui tem um comando de execução diferente dos anteriores. A explicação está abaixo na Figura 1, em resumo: o método *executeQuery()* retorna um objeto do tipo *ResultSet*, mas nem todos os comandos executados no SGBD vão retornar algum valor a ser tratado, como é o caso dos comandos DDL e DML. Para comandos que não retornam parâmetros utilizamos o método *executeUpdate()*.

Method Detail

executeQuery

```
ResultSet executeQuery(String sql)
                       throws SQLException
```

Executes the given SQL statement, which returns a single `ResultSet` object.

Note: This method cannot be called on a `PreparedStatement` or `CallableStatement`.

Parameters:

`sql` - an SQL statement to be sent to the database, typically a static SQL `SELECT` statement

Returns:

a `ResultSet` object that contains the data produced by the given query; never `null`

Throws:

`SQLException` - if a database access error occurs, this method is called on a closed `Statement`, the given SQL statement produces anything other than a single `ResultSet` object, the method is called on a `PreparedStatement` or `CallableStatement`

`SQLTimeoutException` - when the driver has determined that the timeout value that was specified by the `setQueryTimeout` method has been exceeded and has at least attempted to cancel the currently running `Statement`

executeUpdate

```
int executeUpdate(String sql)
                  throws SQLException
```

Executes the given SQL statement, which may be an `INSERT`, `UPDATE`, or `DELETE` statement or an SQL statement that returns nothing, such as an SQL DDL statement.

Note: This method cannot be called on a `PreparedStatement` or `CallableStatement`.

Parameters:

`sql` - an SQL Data Manipulation Language (DML) statement, such as `INSERT`, `UPDATE` or `DELETE`; or an SQL statement that returns nothing, such as a DDL statement.

Returns:

either (1) the row count for SQL Data Manipulation Language (DML) statements or (2) 0 for SQL statements that return nothing

Throws:

`SQLException` - if a database access error occurs, this method is called on a closed `Statement`, the given SQL statement produces a `ResultSet` object, the method is called on a `PreparedStatement` or `CallableStatement`

`SQLTimeoutException` - when the driver has determined that the timeout value that was specified by the `setQueryTimeout` method has been exceeded and has at least attempted to cancel the currently running `Statement`

Figura 1: Captura de tela da API do tutorial do java relativo à classe Statement.

9) O código *ReadFile.java* deve ser reaproveitado para popular a tabela '*debito*'. Para tanto:
    1. Mescle-o com o código apresentado na classe *ReadFile.java*;
        1. As cláusulas *import* devem ser inseridas no início do *MyQueries.java*;
        2. O código destacado em negrito no *ReadFile.java* não fará parte do método *populatetable()*;
        3. O código com tipo de letra normal (sem negrito) deve ser inserido dentro da cláusula *try { … }*, antes do acionamento do comando "*stmt.executeUpdate(create)*". Neste caso, ao invés da impressão do resultado na tela, devemos armazenar o resultado da cadeia de caracteres criada (comando *insert into*) dentro da variável *create*.
        4. Atenção especial deve ser dada para o tratamento de erros. Antes o código *createtable()*, agora transformado em *populatetable()* precisava se resguardar da possibilidade de disparar (*throws*) mensagens de erro de comandos em SQL, por isso a classe tinha a clausula *throws SQLException* na definição do método e ao final da cláusula *try { … }* havia um "*catch (SQLException e)*", estrutura trabalha na última aula. Como o código incorporado possui a possibilidade de que ocorram erros de manipulação de arquivos, precisamos assegurar que a cláusula *try { … }* poderá ter a chance de tratar erros disparados na manipulação de arquivos. Para tanto, acrescente:
            1. "*throws SQLException**, IOException***", na definição do método;

2. "***catch (IOException e) { e.printStackTrace(); }***" após o *catch* existente;

5. Coloque antes do nome do arquivo *debito-populate-table.txt* o caminho completo de sua localização no disco;

2. Inclua a chamada ao método *populatetable()* no método main da classe *MyQueries*;

10) Modificar/criar o arquivo de propriedades de conexão para acessar o banco IB2. Recomendo que copie o arquivo de propriedades de conexão anterior ao PostgreSQL para fazer a conexão ao banco IB2;

11) Salve, compile e execute o *MyQueries*.

    1. Certifique-se de que o arquivo *properties/postgres-properties.xml* está configurado com os parâmetros de conexão ao banco de dados correto;

    2. *./comp MyQueries properties/postgres-properties.xml*

12) Sempre que você tentar executar a inserção destes dados mais de uma vez na tabela debito serão exibidas mensagens de erro como esta:

    org.postgresql.util.PSQLException: ERROR: duplicate key value violates unique constraint "pk_debito"

    Execute o código e comprove.

    ./comp MyQueries properties/postgres-properties.xml

    Isso ocorre porque a integridade da chave primária é policiada pelo SGBD. Para evitar que esta mensagem de erro ocorra, supondo que a tabela precisa ser recarregada periodicamente, podemos usar um comando para truncar a tabela antes de novas inserções. Insira o comando antes do *loop* de inserção de linhas lidas do arquivo, salve e execute novamente:

    1. *stmt.executeUpdate("truncate table debito;");*

    2. *./comp MyQueries properties/postgres-properties.xml*

13) Faça um relatório sobre esta atividade e entregue ao professor com os resultados das etapas principais.