04. Controle de transação

Date	@12/12/2022
Categoria	Java e persistência
Curso	Java e JDBC: trabalhando com um banco de dados

Tópicos

- Projeto da aula anterior
- Assumindo o controle
- JDBC e transações
- Lidando com commit e rollback
- Auto-Commit
- Usando o try-with-resources
- Try com recursos e o close
- Faça que eu fiz
- O que aprendemos?

JDBC e transações

Qual o padrão do JDBC (ou seja do *driver)* para lidar com transações e o banco de dados?

- *Auto-Commit*
  - Esse é o padrão, que pode ser alterado pelo método `setAutoCommit`, da interface `Connection`.

Lidando com commit e rollback

O *commit será executado se não acontecer nenhuma exceção, caso aconteça uma exceção o commit* não será executado e sim o rollback, visto que entraria no catch.

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

public class TestaInsercaoComParametro {
  public static void main(String[] args) throws SQLException{

    ConnectionFactory factory = new ConnectionFactory();
    Connection connection = factory.recuperarConexao();

    connection.setAutoCommit(false);

    try{
      PreparedStatement stm = connection.prepareStatement("INSERT INTO PRODUTO (nome, descricao) VALUES (?,?)", Statement.RETURN_GENER

      adicionarVariavel("SmartTV","SEMP",stm);
      adicionarVariavel("Mouse","Mouse sem fio",stm);

      connection.commit();

      stm.close();
      connection.close();
    } catch (Exception e) {
      e.printStackTrace();
      System.out.println("ROLLBACK EXECUTADO");
      connection.rollback();
```

```
    }

  }

  private static void adicionarVariavel(String nome, String descricao, PreparedStatement stm) throws SQLException{
    stm.setString(1,nome);
        stm.setString(2,descricao);

    if(nome.equals("Mouse")){
      throw new RunTimeException("Não foi possível adicionar o produto");
    }

        stm.execute();

    ResultSet rst = stm.getGeneratedKeys();
    while(rst.next()){
      Integer id = rst.getInt(1);
      System.out.println("Id criado: " + id);
    }
  }
}
```

Auto-Commit

Renata decidiu que não queria mais que o JDBC controlasse as suas transações e por isso *setou* o *Auto-Commit* para `false`. O que mais precisa ser feito para Renata ter o controle total de suas transações?

- É necessário explicitar o *commit* e o *rollback*
    - Caso a transação não dê problema, Renata precisará *commitá-la*, explicitando o *commit*, assim como em caso de erro, o *rollback* precisará estar explícito.

Usando o try-with-resources

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

public class TestaInsercaoComparando {
  public static void main(String[] args) throws SQLException{

    ConnectionFactory factory = new ConnectionFactory();
    try(Connection connection = factory.recuperarConexao()){

      connection.setAutoCommit(false);

      try(
        PreparedStatement stm =
        connection.prepareStatement("INSERT INTO PRODUTO (nome, descricao) VALUES (?,?)", Statement.RETURN_GENERATED_KEYS);
        ){

        adicionarVariavel("SmartTV","SEMP",stm);
        adicionarVariavel("Mouse","Mouse sem fio",stm);

        connection.commit();
      } catch (Exception e) {
        e.printStackTrace();
        System.out.println("ROLLBACK EXECUTADO");
        connection.rollback();
      }
    }
  }

  private static void adicionarVariavel(String nome, String descricao, PreparedStatement stm) throws SQLException{
    stm.setString(1,nome);
        stm.setString(2,descricao);

    if(nome.equals("Mouse")){
      throw new RunTimeException("Não foi possível adicionar o produto");
    }

        stm.execute();

    try(ResultSet rst = stm.getGeneratedKeys()){
      while(rst.next()){
```

```
        Integer id = rst.getInt(1);
        System.out.println("Id criado: " + id);
      }
    }
  }
}
```

Try com recursos e o close

Por que, ao utilizar o *try-with-resources*, não é mais necessário explicitar o `close` para fechar o *statements* (`ResultSet`, `Connection`, `PreparedStatement`)?

- Pelo fato dos *statements* estenderem `AutoCloseable`
  - Estendendo o `AutoCloseable`, o *try-with-resources* executa o método `close` sem que precise estar explícito.

O que aprendemos?

Nesta aula, aprendemos que:

- O banco de dados oferece um recurso chamado de **transação**, para juntar várias alterações como unidade de trabalho
  - Se uma alteração falha, nenhuma alteração é aplicada (é feito um *rollback* da transação)
  - Todas as alterações precisam funcionar para serem aceitas (é feito um `commit`)
- `commit` e `rollback` são operações clássicas de transações
- Para garantir o fechamento dos recursos, existe no Java uma cláusula *try-with-resources*
  - O recurso em questão deve usar a interface `Autoclosable`
