BANCO DE DADOS 2

T08-Comando pré-compilados

Railson Da Silva Martins - 11811BSI208

Wothon Mateus de Araujo - 12111BSI262

1 - Criando método insertMyData1 na classe MyQueries e executando.

Realizada chamada função na main e executado código a baixo:

MyQueries.java

```
283
284  public static void insertMyData1(Connection con) throws SQLException {
285      Statement stmt = null;
286      String query = null;
287      query = "insert into debito (numero_debito, valor_debito, motivo_debito, data_debito, numero_conta, nome_agencia, nome_cliente) " +
288              "values (3000,3000,5,'2014-02-06',36593,'UFU','Pedro Alvares Sousa');";
289      try {
290          stmt = con.createStatement();
291          stmt.executeUpdate(query);
292          if (stmt != null) {
293              stmt.close();
294          }
295          System.out.println("Debitos da Instituicao Bancaria atualizados.");
296      } catch (SQLException e) {
297          JDBCUtilities.printSQLException(e);
298      }
299  }
300
```

Execução:

```
railson@railson-virtual-machine:~/mydir/JDBCTutorial$ ./comp MyQueries propertie
s/postgres-properties_ib2.xml
Set the following properties:
dbms: postgresql
driver: com.postgresql.cj.jdbc.Driver
dbName: IB2
userName: postgres
serverName: localhost
portNumber: 5432
Connected to database
Debitos da Instituicao Bancaria atualizados.
Releasing all open resources ...
```

Consulta no banco:

```
1  -- checar se registro da classe insertMyData1 foi inserido
2  select * from debito
3  where numero_debito = 3000
4
5
```

Data Output | Messages | Notifications

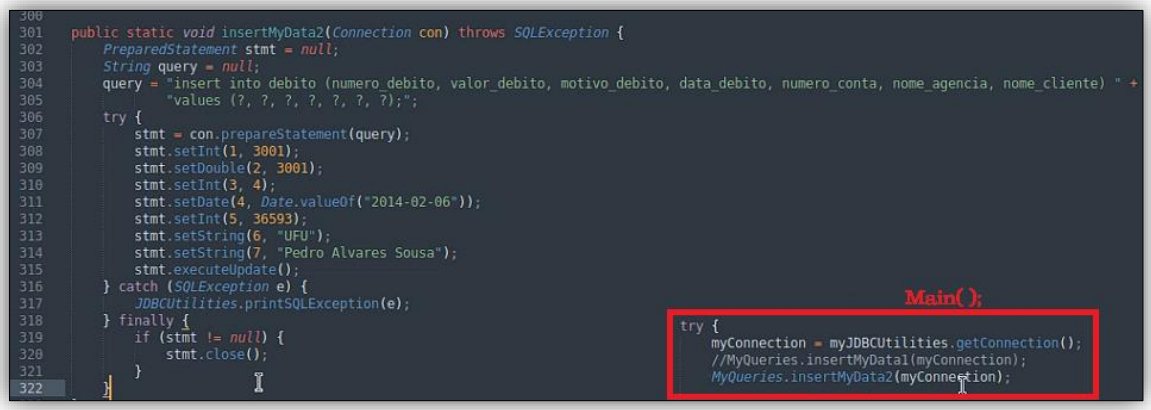
	numero_debito [PK] integer	valor_debito double precision	motivo_debito smallint	data_debito date	numero_conta integer	nome_agencia character varying (50)	nome_cliente character varying (80)
1	3000	3000	5	2014-02-06	36593	UFU	Pedro Alvares Sousa

BANCO DE DADOS 2

T08-Comando pré-compilados

2 – Realizando cópia do método para utilização de comandos pré-compilados.

Realizada cópia do método insertMyData1 gerando o insertMyData2 implementando os comandos pré-compilados, para execução, foi necessário importação da classe: "**import java.sql.PreparedStatement"**, conforme código abaixo:

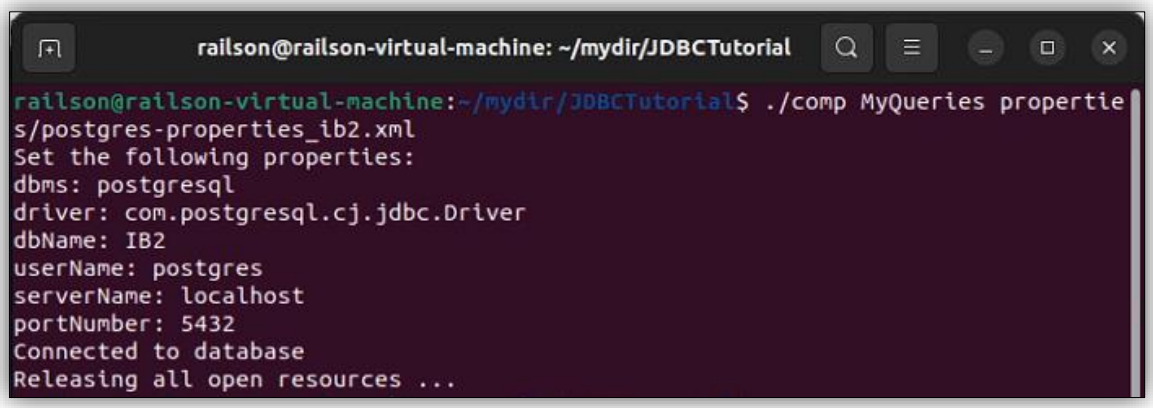


```
public static void insertMyData2(Connection con) throws SQLException {
    PreparedStatement stmt = null;
    String query = null;
    query = "insert into debito (numero_debito, valor_debito, motivo_debito, data_debito, numero_conta, nome_agencia, nome_cliente) " +
            "values (?, ?, ?, ?, ?, ?, ?);";
    try {
        stmt = con.prepareStatement(query);
        stmt.setInt(1, 3001);
        stmt.setDouble(2, 3001);
        stmt.setInt(3, 4);
        stmt.setDate(4, Date.valueOf("2014-02-06"));
        stmt.setInt(5, 36593);
        stmt.setString(6, "UFU");
        stmt.setString(7, "Pedro Alvares Sousa");
        stmt.executeUpdate();
    } catch (SQLException e) {
        JDBCUtilities.printSQLException(e);
    } finally {
        if (stmt != null) {
            stmt.close();
        }
    }
```

Main( );

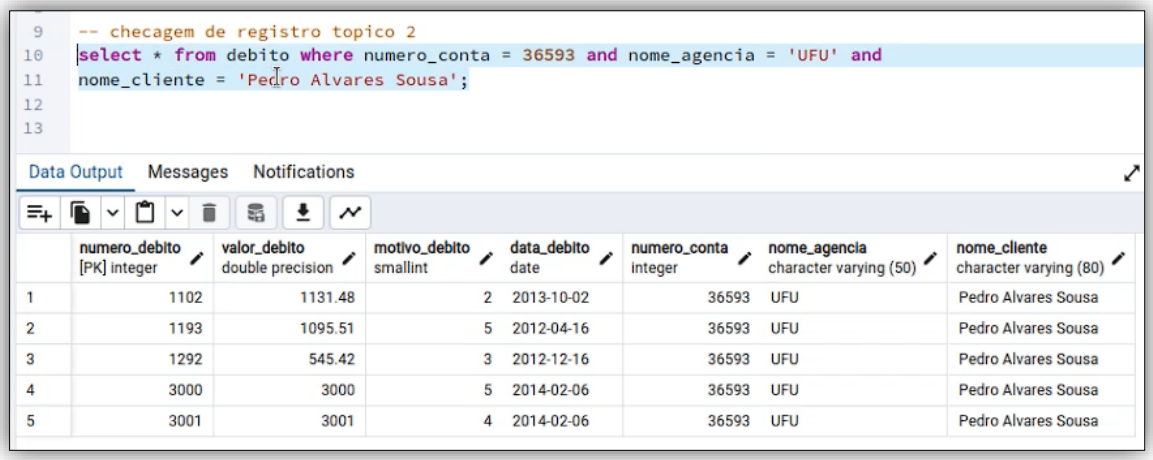
```
try {
    myConnection = myJDBCUtilities.getConnection();
    //MyQueries.insertMyData1(myConnection);
    MyQueries.insertMyData2(myConnection);
```

Execução:



```
railson@railson-virtual-machine: ~/mydir/JDBCTutorial
railson@railson-virtual-machine:~/mydir/JDBCTutorial$ ./comp MyQueries propertie
s/postgres-properties_ib2.xml
Set the following properties:
dbms: postgresql
driver: com.postgresql.cj.jdbc.Driver
dbName: IB2
userName: postgres
serverName: localhost
portNumber: 5432
Connected to database
Releasing all open resources ...
```

Consulta no banco:



```
-- checagem de registro topico 2
select * from debito where numero_conta = 36593 and nome_agencia = 'UFU' and
nome_cliente = 'Pedro Alvares Sousa';
```

Data Output | Messages | Notifications

	numero_debito [PK] integer	valor_debito double precision	motivo_debito smallint	data_debito date	numero_conta integer	nome_agencia character varying (50)	nome_cliente character varying (80)
1	1102	1131.48	2	2013-10-02	36593	UFU	Pedro Alvares Sousa
2	1193	1095.51	5	2012-04-16	36593	UFU	Pedro Alvares Sousa
3	1292	545.42	3	2012-12-16	36593	UFU	Pedro Alvares Sousa
4	3000	3000	5	2014-02-06	36593	UFU	Pedro Alvares Sousa
5	3001	3001	4	2014-02-06	36593	UFU	Pedro Alvares Sousa

BANCO DE DADOS 2

T08-Comando pré-compilados

3 – Medindo tempo de execução dos métodos insertMyData1 e insertMyData2.

Classes alteradas e adicionado a informação de milissegundos em seu corpo, necessário excluir informações inseridas no passo anterior para a execução:

```java
public static void insertMyData1(Connection con) throws SQLException {
    Statement stmt = null;
    String query = null;
    query = "insert into debito (numero_debito, valor_debito, motivo_debito, data_debito, numero_conta, nome_agencia,
    nome_cliente) " +
            "values (3000,3000,5,'2014-02-06',36593,'UFU','Pedro Alvares Sousa');";
    try {
        long startTime = System.currentTimeMillis();

        stmt = con.createStatement();
        stmt.executeUpdate(query);

        long endTime = System.currentTimeMillis();
        System.out.println("Débito inserido pela classe insertMyData1 no banco em " + (endTime - startTime) + "
        milissegundos");

    } catch (SQLException e) {
        JDBCUtilities.printSQLException(e);
    } finally {
        if (stmt != null) {
            stmt.close();
        }
    }
}
```

```java
public static void insertMyData2(Connection con) throws SQLException {
    PreparedStatement stmt = null;
    String query = null;
    query = "insert into debito (numero_debito, valor_debito, motivo_debito, data_debito, numero_conta, nome_agencia,
    nome_cliente) " +
            "values (?, ?, ?, ?, ?, ?, ?);";
    try {
        long startTime = System.currentTimeMillis();

        stmt = con.prepareStatement(query);
        stmt.setInt(1, 3001);
        stmt.setDouble(2, 3001);
        stmt.setInt(3, 4);
        stmt.setDate(4, Date.valueOf("2014-02-06"));
        stmt.setInt(5, 36593);
        stmt.setString(6, "UFU");
        stmt.setString(7, "Pedro Alvares Sousa");
        stmt.executeUpdate();

        long endTime = System.currentTimeMillis();
        System.out.println("Débito inserido pela classe insertMyData2 no banco em " + (endTime - startTime) + "
        milissegundos");

    } catch (SQLException e) {
        JDBCUtilities.printSQLException(e);
    } finally {
        if (stmt != null) {
            stmt.close();
        }
    }
}
```

Excluindo valores para impedir erros de duplicação de chaves:

```sql
-- deletando informaçoes para execuçao do passo 3
delete from debito where numero_debito in (3000, 3001)
```

```
DELETE 2

Query returned successfully in 94 msec.
```

BANCO DE DADOS 2

T08-Comando pré-compilados

Execução:

```
railson@railson-virtual-machine:~/mydir/JDBCTutorial$ ./comp MyQueries propertie
s/postgres-properties_ib2.xml
Set the following properties:
dbms: postgresql
driver: com.postgresql.cj.jdbc.Driver
dbName: IB2
userName: postgres
serverName: localhost
portNumber: 5432
Connected to database
Débito inserido pela classe insertMyData1 no banco em 16 milissegundos
Débito inserido pela classe insertMyData2 no banco em 9 milissegundos
Releasing all open resources ...
```

Diferença de 5 milissegundos.

4 –Criação dos métodos insertMyData1000 e insertMyData2000, e sua execução.

Realizada criação de loop para incremento dos valores gerados pela execução do laço.

Código do método **insertMyData1000:**

```
337
338   public static void insertMyData1000(Connection con) throws SQLException {
339       Statement stmt = null;
340       String query = null;
341       try {
342           long startTime = System.currentTimeMillis();
343           System.out.println("Iniciando InsertMyData1.\n\n ");
344           for (int numdeb = 3002; numdeb < 4002; numdeb++) {
345               query = "insert into debito (numero_debito, valor_debito, motivo_debito, data_debito, numero_conta,
                  nome_agencia, nome_cliente) " +
346                       "values (" + numdeb + "," + numdeb + ",5,'2014-02-06',36593,'UFU','Pedro Alvares Sousa');";
347
348               stmt = con.createStatement();
349               stmt.executeUpdate(query);
350
351               if ((numdeb % 50) == 0) {
352                   long endTime = System.currentTimeMillis();
353                   System.out.println(numdeb - 3000 + "\t" + (endTime - startTime));
354               }
355           }
356           System.out.println("\n\nInsertMyData1 concluido.");
357
358       } catch (SQLException e) {
359           JDBCUtilities.printSQLException(e);
360       } finally {
361           if (stmt != null) {
362               stmt.close();
363           }
364       }
365   }
```

BANCO DE DADOS 2

T08-Comando pré-compilados

Código do método **insertMyData2000:**

```java
public static void insertMyData2000(Connection con) throws SQLException {
    PreparedStatement stmt = null;
    String query = null;
    try {
        long startTime = System.currentTimeMillis();

        System.out.println("Iniciando InsertMyData2.\n\n ");
        for (int numdeb = 5002; numdeb < 6002; numdeb++) {
            query = "insert into debito (numero_debito, valor_debito, motivo_debito, data_debito, numero_conta,
            nome_agencia, nome_cliente) " +
                    "values (?, ?, 5, '2014-02-06', 36593, 'UFU', 'Pedro Alvares Sousa');";

            stmt = con.prepareStatement(query);
            stmt.setInt(1, numdeb);
            stmt.setDouble(2, numdeb);
            stmt.executeUpdate();

            if ((numdeb % 50) == 0) {
                long endTime = System.currentTimeMillis();
                System.out.println(numdeb - 5000 + "\t" + (endTime - startTime));
            }
        }
        System.out.println("\n\nInsertMyData2 concluido.");

    } catch (SQLException e) {
        JDBCUtilities.printSQLException(e);
    } finally {
        if (stmt != null) {
            stmt.close();
        }
    }
}
```

Execução:

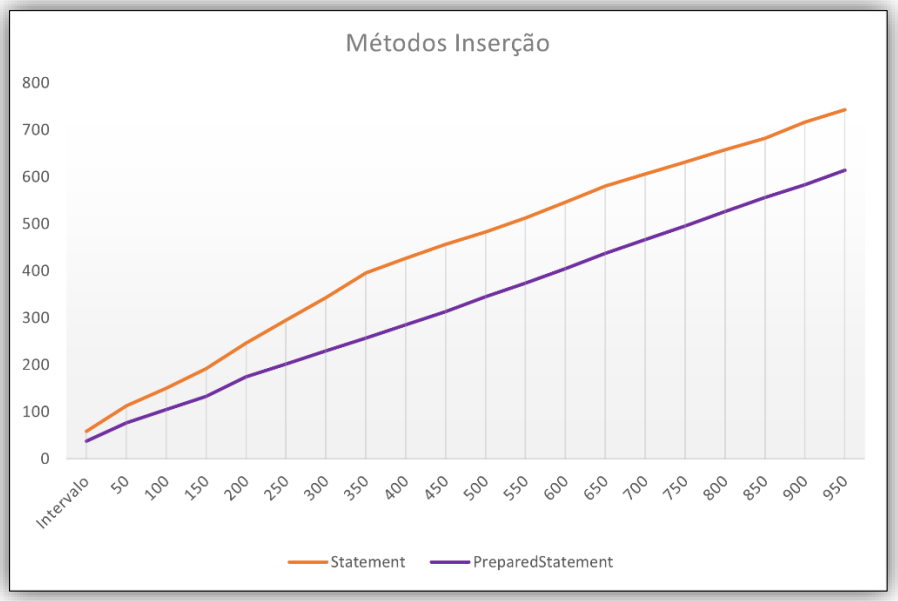
```
railson@railson-vir
Connected to database
Iniciando InsertMyData1.


50      59
100     113
150     150
200     192
250     246
300     295
350     343
400     396
450     427
500     457
550     483
600     513
650     546
700     581
750     606
800     632
850     658
900     682
950     717
1000    743


InsertMyData1 concluido.
```

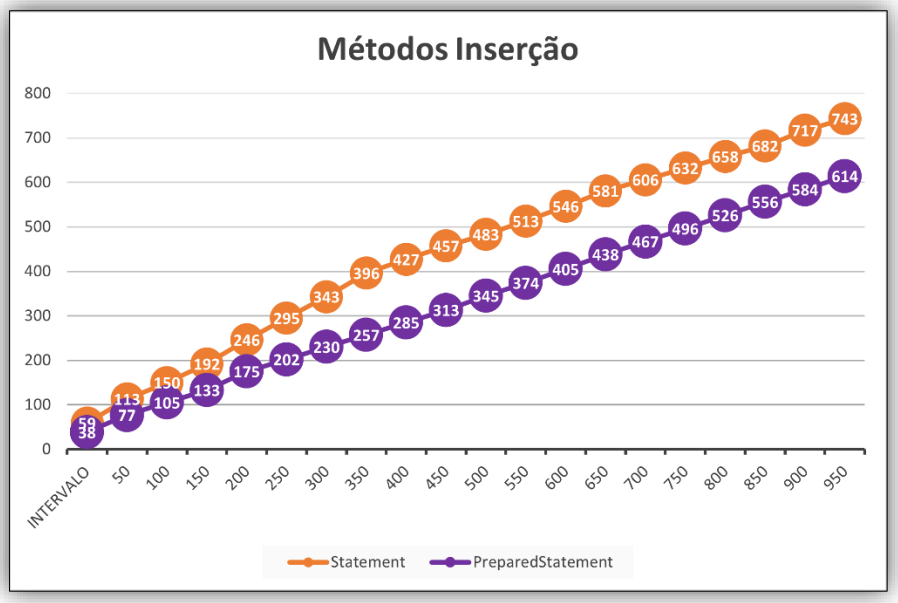
```
railson@railson-virtual-machi
Iniciando InsertMyData2.


50      38
100     77
150     105
200     133
250     175
300     202
350     230
400     257
450     285
500     313
550     345
600     374
650     405
700     438
750     467
800     496
850     526
900     556
950     584
1000    614


InsertMyData2 concluido.
Releasing all open resources ...
```

BANCO DE DADOS 2

T08-Comando pré-compilados

Gráficos de comparações Statement e PreparedStatement.





Após olhar para o gráfico, fica claro que o método InsertMyData2, usando PreparedStatement, é mais eficiente que o InsertMyData1, que usa Statement. Isso é evidenciado pelos tempos de execução mais curtos em cada método. Por exemplo, ao inserir 1000 linhas, o InsertMyData1 levou cerca de 743 milissegundos, enquanto o InsertMyData2 levou apenas cerca de 614 milissegundos. Essa diferença se deve ao fato de que o PreparedStatement pré-compila a consulta SQL, tornando-a mais eficiente do que o Statement, que não faz essa pré-compilação. Portanto, usar PreparedStatement é uma boa prática ao lidar com grandes quantidades de dados em um banco de dados, resultando em tempos de execução mais curtos e melhor desempenho do sistema.

BANCO DE DADOS 2

T08-Comando pré-compilados

5 – Criação do método insertMyData3000 com uso do processamento em BATCH.

Realizado a cópia do método insertMyData2000 para criação do insertMyData3000 e alterado o código para utilização do processamento em batch:

Código do método **insertMyData3000:**

```java
public static void insertMyData3000(Connection con) throws SQLException {
    PreparedStatement stmt = null;
    String query = null;
    try {
        con.setAutoCommit(false);

        query = "insert into debito (numero_debito, valor_debito, motivo_debito, data_debito, numero_conta, nome_agencia, nome_cliente) " +
                "values (?, ?, 5, '2014-02-06', 36593, 'UFU', 'Pedro Alvares Sousa');";
        stmt = con.prepareStatement(query);

        con.setAutoCommit(false);
        int numdeb;
        long startTime = System.currentTimeMillis();

        for (numdeb = 5002; numdeb < 6002; numdeb++) {
            stmt.setInt(1, numdeb);
            stmt.setDouble(2, numdeb);
            stmt.addBatch();

            if ((numdeb % 50) == 0) {
                long endTime = System.currentTimeMillis();
                System.out.println(numdeb - 5000 + "\t" + (endTime - startTime));
            }
        }

        stmt.executeBatch();

        con.commit();
        long endTime = System.currentTimeMillis();
        System.out.println("\n\nCommit concluido em: " + (endTime - startTime)  + " milissegundos \n\n");

    } catch (SQLException e) {
        JDBCUtilities.printSQLException(e);
        if (con != null) {
            con.rollback();
        }
    } finally {
        if (stmt != null) {
            stmt.close();
        }
        con.setAutoCommit(true);
    }
}
```

Exclusão de registros antes da execução do código:

```
13  -- deletar debitos com numeros maiores que 2999
14  delete from debito where numero_conta = 36593 and nome_agencia = 'UFU' and nome_cliente =
15  'Pedro Alvares Sousa' and numero_debito >= 3000;
```

Data Output | Messages | Notifications

```
DELETE 2002

Query returned successfully in 99 msec.
```

BANCO DE DADOS 2

T08-Comando pré-compilados

Execução:

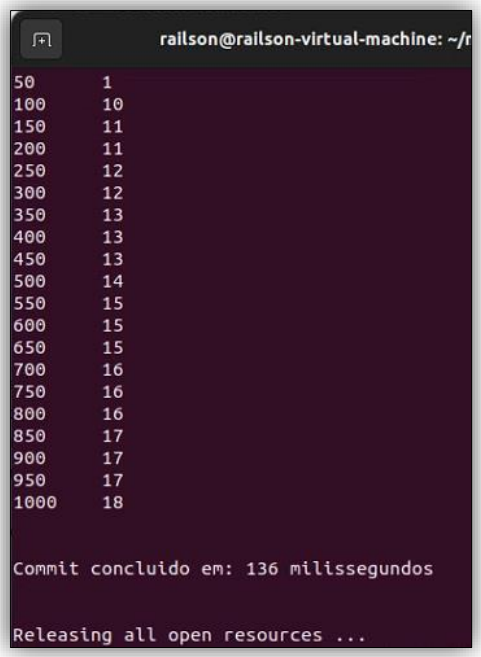
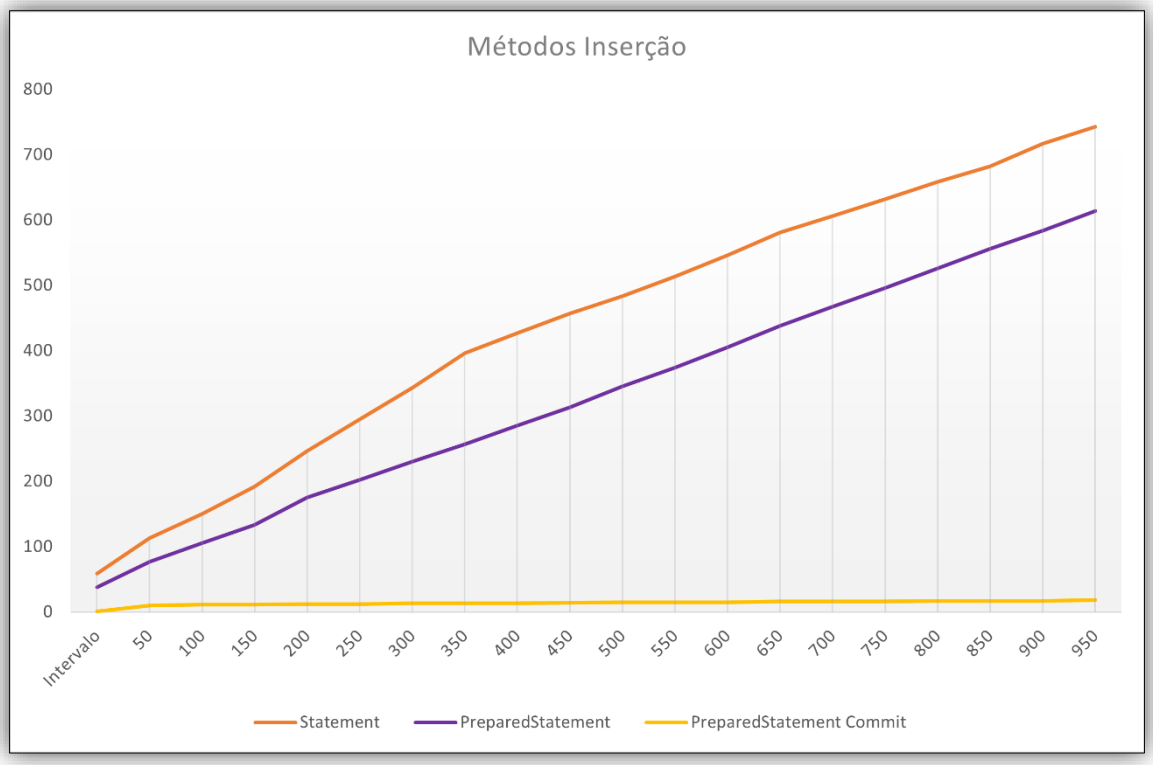
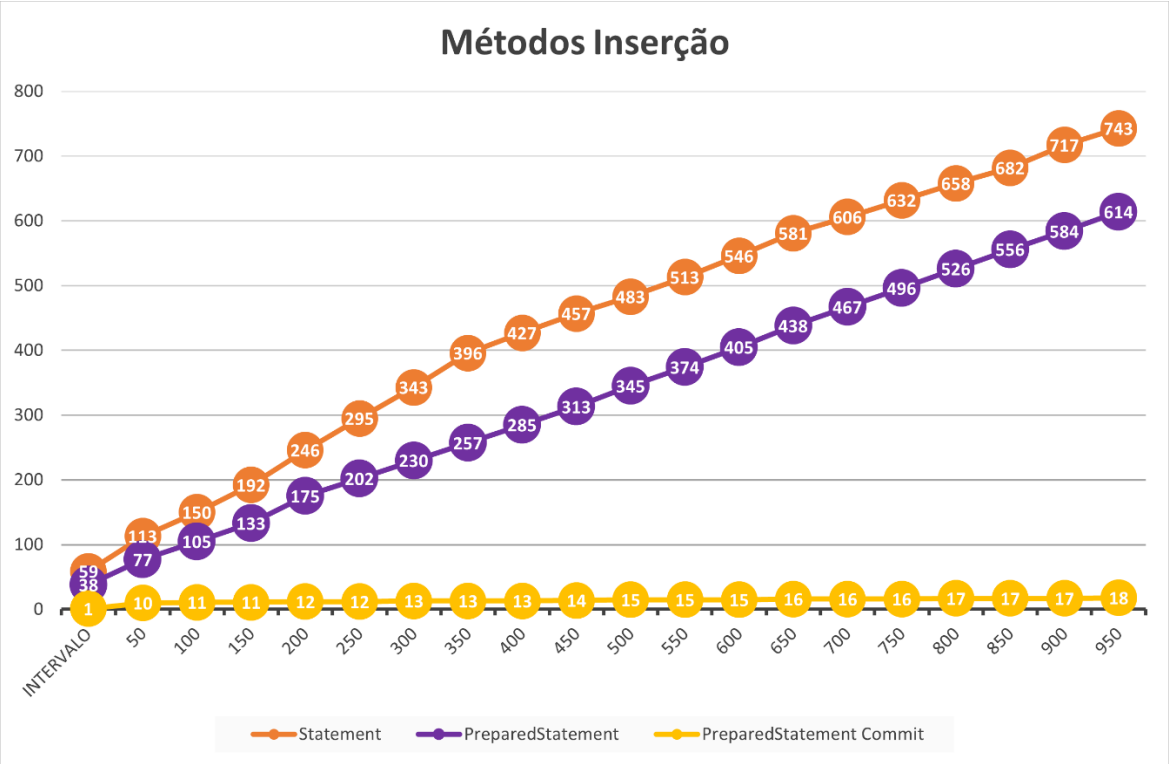


Gráfico de comparações Statement, PreparedStatement e PreparedStatment commit:



BANCO DE DADOS 2

T08-Comando pré-compilados



6 - O que acontecerá se utilizarmos o recurso de desabilitar o auto-commit para o caso de usarmos a classe Statement em vez de PreparedStatement?

Ao desabilitar o auto-commit ao usar a classe Statement, estamos assumindo o controle manual das transações no banco de dados, o que significa que as operações SQL não serão confirmadas automaticamente após sua execução.

Isso nos dá controle total sobre o gerenciamento de transações, permitindo-nos agrupar operações, garantir atomicidade e consistência, executar operações em lote e gerenciar exceções de forma eficaz. No entanto, essa abordagem também introduz riscos e complexidades adicionais, pois agora somos responsáveis por confirmar ou reverter transações manualmente, o que pode aumentar a possibilidade de erros e dificultar a manutenção do código.