```java
package stakashka.ssm;

import stakashka.ssm.core.model.Controller;
import stakashka.ssm.core.model.Controller.Databases;

public class Main {

    public static void main(String[] args) {
        String urlFrom = "jdbc:oracle:thin:@localhost:1521:orcl";
        String user = "ewa";
        String password = "ewapas";
        String schemaFrom = "ewa";
        Databases databaseFrom = Databases.Oracle;
        String schemaTo = "ewa";
        Databases databaseTo = Databases.Postgres;

        Controller controller = new Controller(urlFrom, user, password, schemaFrom, databaseFrom,
schemaTo, databaseTo);
        controller.process();
    }
}

```

```java
package stakashka.ssm.api.db;

public interface TypesTemplates {
    String getTypeTemplate(String[] args);
}

```

```java
package stakashka.ssm.api.db;

import stakashka.ssm.core.data.*;
import stakashka.ssm.core.data.Constraint.ConstraintType;
import stakashka.ssm.core.data.DataTypes.NormalizedTypes;

import java.sql.*;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

abstract public class AbstractDatabase {

    protected String schemaFrom;
    protected String schemaTo;
    protected Connection connection = null;
    protected Statement statement = null;
    protected ResultSet resultSet = null;
    // TODO : unique values and think about it more
    protected Map<TypesTemplates, NormalizedTypes> typesMapping = null;
    protected Map<NormalizedTypes, TypesTemplates> inverseTypesMapping = null;

    public AbstractDatabase(String schemaTo) throws Exception {
        this.schemaTo = schemaTo;
        initTypesMapping();
        initInverseTypesMapping();
    }

    public AbstractDatabase(String url, String user, String password, String schemaFrom) throws
SQLException {
        this.schemaFrom = schemaFrom;
        initTypesMapping();
        initDriver();
        try {
            this.connection = DriverManager.getConnection(url, user, password);
        } catch (SQLException e) {
            e.printStackTrace();
            try {
                if (connection != null) connection.close();
            } catch (Exception ee) {
                ee.printStackTrace();
            }
            throw e;
        }
    }

    protected abstract void initTypesMapping();

    protected void initInverseTypesMapping() throws Exception {
        inverseTypesMapping = new HashMap<>();
        for (TypesTemplates key : typesMapping.keySet()) {
            inverseTypesMapping.put(typesMapping.get(key), key);
        }
        if (inverseTypesMapping.size() != typesMapping.size()) {
            throw new Exception();
        }
    }

    public Schema getSchema() throws SQLException {
        try {
            Schema schema = new Schema();
            schema.setTablesList(getTablesList());
            schema.setColumnsList(getColumnsList());
            schema.setConstraintsList(getConstraintsList());
            schema.setConstraintColumnsList(getConstraintColumnsList());
            return schema;
        } catch (SQLException e) {
            e.printStackTrace();
            throw e;
        } finally {
            try {
                if (connection != null) connection.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            try {
```

```
77                 if (statement != null) statement.close();
78             } catch (Exception e) {
79                 e.printStackTrace();
80             }
81             try {
82                 if (resultSet != null) resultSet.close();
83             } catch (Exception e) {
84                 e.printStackTrace();
85             }
86         }
87     }
88
89     public String crateDDL(Schema schema) throws Exception {
90         String resultDDL = "";
91
92         Map<String, List<Column>> groupByTables = schema.getColumnsList()
93                 .stream().collect(Collectors.groupingBy(Column::getTableName));
94         for (String table : groupByTables.keySet()) {
95             resultDDL += createTableDDL(table, groupByTables.get(table));
96         }
97
98         // foreign key must be created last
99         List<Constraint> sortedConstraintList = schema.getConstraintsList().stream()
100                .sorted((c1, c2) -> ConstraintType.compare(c1.getConstraintType(), c2.
    getConstraintType()))
101                .collect(Collectors.toList());
102        Map<String, List<ConstraintColumn>> columnsMap = schema.getConstraintColumnsList()
103                .stream().collect(Collectors.groupingBy(ConstraintColumn::getConstraintName));
104        for (Constraint constraint : sortedConstraintList) {
105            Constraint refConstraint = null;
106            List<ConstraintColumn> refCol = null;
107            if (constraint.getRefConstraintName() != null) {
108                refConstraint = schema.getConstraintsList().stream()
109                        .filter(con -> con.getConstraintName().equals(constraint.getRefConstraintName
    ()))
110                        .findFirst().get();
111                refCol = columnsMap.get(refConstraint.getConstraintName());
112            }
113            resultDDL += createConstraintDDL(constraint, columnsMap.get(constraint.getConstraintName(
    )), refConstraint, refCol);
114        }
115
116        System.out.println(resultDDL);
117        return resultDDL;
118    }
119
120    protected abstract void initDriver();
121
122    protected abstract List<Table> getTablesList() throws SQLException;
123
124    protected abstract List<Column> getColumnsList() throws SQLException;
125
126    protected abstract List<Constraint> getConstraintsList() throws SQLException;
127
128    protected abstract List<ConstraintColumn> getConstraintColumnsList() throws SQLException;
129
130    protected NormalizedTypes getNormalizedType(TypesTemplates type) {
131        NormalizedTypes res = typesMapping.get(type);
132        return (res == null) ? NormalizedTypes.VARCHAR : res;
133    }
134
135    protected TypesTemplates getSpecificType(NormalizedTypes type) throws Exception {
136        TypesTemplates res = inverseTypesMapping.get(type);
137        if (res == null) {
138            throw new Exception();
139        }
140        return res;
141    }
142
143    protected abstract String createTableDDL(String tableName, List<Column> columnsList) throws
    Exception;
144
145    protected abstract String createConstraintDDL(Constraint constraint, List<ConstraintColumn>
    columns,
146                                                  Constraint refConstraint, List<ConstraintColumn>
    refColumns);
147 }
```

```java
package stakashka.ssm.api.db.oracle;

import stakashka.ssm.api.db.AbstractDatabase;
import stakashka.ssm.api.db.TypesTemplates;
import stakashka.ssm.core.data.*;
import stakashka.ssm.core.data.Constraint.ConstraintType;

import java.sql.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class OracleDatabase extends AbstractDatabase {

    private enum ColumnTypes implements TypesTemplates {
        VARCHAR2,
        NUMBER;

        @Override
        public String getTypeTemplate(String[] args) {
            return null;
        }
    }

    public OracleDatabase(String schemaTo) throws Exception {
        super(schemaTo);
    }

    public OracleDatabase(String url, String user, String password) throws SQLException {
        super(url, user, password, null);
        schemaFrom = connection.getMetaData().getUserName();
    }

    @Override
    protected void initTypesMapping() {
        typesMapping = new HashMap<TypesTemplates, DataTypes.NormalizedTypes>() {{
            put(ColumnTypes.NUMBER, DataTypes.NormalizedTypes.NUMBER);
            put(ColumnTypes.VARCHAR2, DataTypes.NormalizedTypes.VARCHAR);
        }};
    }

    @Override
    protected void initDriver() {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    @Override
    public List<Table> getTablesList() throws SQLException {
        List<Table> tablesList = new ArrayList<Table>();
        statement = connection.createStatement();
        resultSet = statement.executeQuery("select TABLE_NAME from SYS.ALL_TABLES where OWNER = '" +
schemaFrom + "'");
        while (resultSet.next()) {
            tablesList.add(new Table(resultSet.getString(1)));
        }
        return tablesList;
    }

    @Override
    protected List<Column> getColumnsList() throws SQLException {
        List<Column> columnsList = new ArrayList<>();
        statement = connection.createStatement();
        resultSet = statement.executeQuery(
                "select COLUMN_NAME, TABLE_NAME, DATA_TYPE, DATA_LENGTH, DATA_PRECISION, DATA_SCALE,
NULLABLE " +
                        "from SYS.ALL_TAB_COLS where OWNER = '" + schemaFrom + "' order by COLUMN_ID"
        );
        while (resultSet.next()) {
            columnsList.add(new Column(
                    resultSet.getString(1), resultSet.getString(2),
                    getNormalizedType(ColumnTypes.valueOf(resultSet.getString(3))),
                    resultSet.getInt(4), resultSet.getInt(5), resultSet.getInt(6),
                    resultSet.getString(7).equals("Y")
```

```
76              ));
77          }
78          return columnsList;
79      }
80
81      @Override
82      protected List<Constraint> getConstraintsList() throws SQLException {
83          List<Constraint> constraintsList = new ArrayList<>();
84          statement = connection.createStatement();
85          resultSet = statement.executeQuery(
86                  "select CONSTRAINT_NAME, GENERATED, CONSTRAINT_TYPE, TABLE_NAME, " +
87                          "SEARCH_CONDITION_VC, R_CONSTRAINT_NAME " +
88                          "from sys.all_constraints where owner = '" + schemaFrom + "'"
89          );
90          while (resultSet.next()) {
91              constraintsList.add(new Constraint(
92                      resultSet.getString(1), resultSet.getString(2).equals("GENERATED NAME"),
93                      ConstraintType.valueOf(resultSet.getString(3)), resultSet.getString(4),
94                      resultSet.getString(5), resultSet.getString(6)
95              ));
96          }
97          return constraintsList;
98      }
99
100     @Override
101     protected List<ConstraintColumn> getConstraintColumnsList() throws SQLException {
102         List<ConstraintColumn> columnsList = new ArrayList<>();
103         statement = connection.createStatement();
104         resultSet = statement.executeQuery(
105             "select CONSTRAINT_NAME, TABLE_NAME, COLUMN_NAME, POSITION " +
106                     "from SYS.ALL_CONS_COLUMNS where OWNER = '" + schemaFrom + "'"
107         );
108         while (resultSet.next()) {
109             columnsList.add(new ConstraintColumn(
110                     resultSet.getString(1), resultSet.getString(2),
111                     resultSet.getString(3), resultSet.getInt(4)
112             ));
113         }
114         return columnsList;
115     }
116
117     @Override
118     protected String createTableDDL(String tableName, List<Column> columnsList) {
119         return null;
120     }
121
122     @Override
123     protected String createConstraintDDL(Constraint constraint, List<ConstraintColumn> columns,
124                                          Constraint refConstraint, List<ConstraintColumn> refColumns)
    {
125         return null;
126     }
127
128     public static void main(String[] args) throws Exception {
129         Connection connection;
130         Class.forName("oracle.jdbc.driver.OracleDriver");
131         connection = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl", "ewa", "
    ewapas");
132         System.out.println(connection.getMetaData().getDatabaseProductName());
133         Statement statement = connection.createStatement();
134         ResultSet resultSet = statement.executeQuery("select TABLE_NAME from SYS.ALL_TABLES where
    OWNER = 'EWA'");
135         while (resultSet.next()) {
136             System.out.println(resultSet.getString(1));
137         }
138         connection.close();
139     }
140
141
142 }
143
```

```java
package stakashka.ssm.api.db.postgres;

import stakashka.ssm.api.db.AbstractDatabase;
import stakashka.ssm.api.db.TypesTemplates;
import stakashka.ssm.core.data.*;

import java.sql.SQLException;
import java.util.HashMap;
import java.util.List;
import java.util.stream.Collectors;

public class PostgresDatabase extends AbstractDatabase {

    private enum ColumnTypes implements TypesTemplates {
        CHARACTER,
        NUMERIC;

        @Override
        public String getTypeTemplate(String[] args) {
            switch (this) {
                case CHARACTER:
                    return String.format("CHARACTER(%s)", args[0]);
                case NUMERIC:
                    return String.format("NUMERIC(%s,%s)", args[1], args[2]);
                default:
                    return this.name();
            }
        }
    }

    public PostgresDatabase(String schemaTo) throws Exception {
        super(schemaTo);
    }

    public PostgresDatabase(String url, String user, String password, String schemaFrom) throws
SQLException {
        super(url, user, password, schemaFrom);
    }

        @Override
        protected void initTypesMapping() {
            typesMapping = new HashMap<TypesTemplates, DataTypes.NormalizedTypes>() {{
                put(ColumnTypes.NUMERIC, DataTypes.NormalizedTypes.NUMBER);
                put(ColumnTypes.CHARACTER, DataTypes.NormalizedTypes.VARCHAR);
            }};
        }

    @Override
    protected void initDriver() {

    }

    @Override
    protected List<Table> getTablesList() throws SQLException {
        return null;
    }

    @Override
    protected List<Column> getColumnsList() throws SQLException {
        return null;
    }

    @Override
    protected List<Constraint> getConstraintsList() throws SQLException {
        return null;
    }

    @Override
    protected List<ConstraintColumn> getConstraintColumnsList() throws SQLException {
        return null;
    }

    @Override
    protected String createTableDDL(String tableName, List<Column> columnsList) throws Exception {
        String res = "";
        res += "CREATE TABLE " + ((schemaTo == null) ? "" : schemaTo + ".") + tableName + " (";
        for (Column column : columnsList) {
```

```
77              String type = getSpecificType(column.getColumnType())
78                      .getTypeTemplate(new String[]{column.getLength().toString(), column.getPrecision(
   ).toString(), column.getScale().toString()});
79              res += "\n    " + column.getColumnName() + " " + type + ((column.isNullable()) ? "" : "
   not null") + ",";
80          }
81          res = res.substring(0, res.length() - 1);
82          res += "\n);\n";
83          return res;
84      }
85
86      @Override
87      protected String createConstraintDDL(Constraint constraint, List<ConstraintColumn> columns,
88                                           Constraint refConstraint, List<ConstraintColumn> refColumns)
    {
89          String res = "ALTER TABLE " + ((schemaTo == null) ? "" : schemaTo + ".") + constraint.
   getTableName() + " ADD"
90                  + (constraint.isGeneratedName() ? "" : " CONSTRAINT " + constraint.getConstraintName(
   )) + " ";
91          switch (constraint.getConstraintType()) {
92              case C:
93                  if (!constraint.getSearchCondition().contains("IS NOT NULL")) {
94                      res += "CHECK (" + constraint.getSearchCondition() + ");\n";
95                  } else {
96                      return "";
97                  }
98                  break;
99              case O:
100                 break;
101             case P:
102                 res += "PRIMARY KEY (";
103                 columns = columns.stream().sorted((c1, c2) -> Integer.compare(c1.getPosition(), c2.
    getPosition()))
104                         .collect(Collectors.toList());
105                 for (ConstraintColumn column: columns) {
106                     res += column.getColumnName() + ", ";
107                 }
108                 res = res.substring(0, res.length() - 2) + ");\n";
109                 break;
110             case U:
111                 res += "UNIQUE (";
112                 columns = columns.stream().sorted((c1, c2) -> Integer.compare(c1.getPosition(), c2.
    getPosition()))
113                         .collect(Collectors.toList());
114                 for (ConstraintColumn column: columns) {
115                     res += column.getColumnName() + ", ";
116                 }
117                 res = res.substring(0, res.length() - 2) + ");\n";
118             case V:
119                 break;
120             case R:
121                 res += "FOREIGN KEY (" + columns.get(0).getColumnName() + ") REFERENCES "
122                         + ((schemaTo == null) ? "" : schemaTo + ".") + refConstraint.getTableName()
123                         + " (" + refColumns.get(0).getColumnName() + ");\n";
124                 break;
125         }
126         return res;
127     }
128 }
129
```

```java
package stakashka.ssm.core.data;

public class Table {
    private String tableName;

    public Table(String tableName) {
        this.tableName = tableName;
    }

    public String getTableName() {
        return tableName;
    }

    public void setTableName(String tableName) {
        this.tableName = tableName;
    }

    @Override
    public String toString() {
        return "Table{" +
                "tableName='" + tableName + '\'' +
                '}';
    }
}

```

```java
package stakashka.ssm.core.data;

public class Column {

    private String columnName;
    private String tableName;
    private DataTypes.NormalizedTypes columnType;
    private Integer length;
    private Integer precision;
    private Integer scale;
    private boolean nullable;

    public Column(String columnName, String tableName, DataTypes.NormalizedTypes columnType,
                  Integer length, Integer precision, Integer scale, boolean nullable) {
        this.columnName = columnName;
        this.tableName = tableName;
        this.columnType = columnType;
        this.length = length;
        this.precision = precision;
        this.scale = scale;
        this.nullable = nullable;
    }

    public String getColumnName() {
        return columnName;
    }

    public void setColumnName(String columnName) {
        this.columnName = columnName;
    }

    public String getTableName() {
        return tableName;
    }

    public void setTableName(String tableName) {
        this.tableName = tableName;
    }

    public DataTypes.NormalizedTypes getColumnType() {
        return columnType;
    }

    public void setColumnType(DataTypes.NormalizedTypes columnType) {
        this.columnType = columnType;
    }

    public Integer getLength() {
        return length;
    }

    public void setLength(Integer length) {
        this.length = length;
    }

    public Integer getPrecision() {
        return precision;
    }

    public void setPrecision(Integer precision) {
        this.precision = precision;
    }

    public Integer getScale() {
        return scale;
    }

    public void setScale(Integer scale) {
        this.scale = scale;
    }

    public boolean isNullable() {
        return nullable;
    }

    public void setNullable(boolean nullable) {
        this.nullable = nullable;
```

```
78      }
79
80      @Override
81      public String toString() {
82          return "Column{" +
83                  "columnName='" + columnName + '\'' +
84                  ", tableName='" + tableName + '\'' +
85                  ", columnType=" + columnType +
86                  ", length=" + length +
87                  ", precision=" + precision +
88                  ", scale=" + scale +
89                  ", nullable=" + nullable +
90                  '}';
91      }
92  }
93
```

```java
package stakashka.ssm.core.data;

import java.util.ArrayList;
import java.util.List;

public class Schema {

    private List<Table> tablesList;
    private List<Column> columnsList;
    private List<Constraint> constraintsList;
    private List<ConstraintColumn> constraintColumnsList;


    public Schema() {
        tablesList = new ArrayList<>();
        columnsList = new ArrayList<>();
        constraintsList = new ArrayList<>();
        constraintColumnsList = new ArrayList<>();
    }

    public List<Table> getTablesList() {
        return tablesList;
    }

    public void setTablesList(List<Table> tablesList) {
        this.tablesList = tablesList;
    }

    public List<Column> getColumnsList() {
        return columnsList;
    }

    public void setColumnsList(List<Column> columnsList) {
        this.columnsList = columnsList;
    }

    public List<Constraint> getConstraintsList() {
        return constraintsList;
    }

    public void setConstraintsList(List<Constraint> constraintsList) {
        this.constraintsList = constraintsList;
    }

    public List<ConstraintColumn> getConstraintColumnsList() {
        return constraintColumnsList;
    }

    public void setConstraintColumnsList(List<ConstraintColumn> constraintColumnsList) {
        this.constraintColumnsList = constraintColumnsList;
    }

    @Override
    public String toString() {
        return "Schema{" +
                "tablesList=" + tablesList +
                ", columnsList=" + columnsList +
                ", constraintsList=" + constraintsList +
                ", constraintColumnsList=" + constraintColumnsList +
                '}';
    }
}

```

```java
package stakashka.ssm.core.data;

public class DataTypes {
    public enum NormalizedTypes {
        NUMBER,
        VARCHAR
    }
}

```

```java
package stakashka.ssm.core.data;

public class Constraint {

    public enum ConstraintType {
        C,
        O,
        P,
        U,
        V,
        R;
        // foreign keys must be created last
        public static int compare(ConstraintType ct1, ConstraintType ct2) {
            if (ct1 == ct2 || (ct1 != R && ct2 != R)) {
                return ct1.name().compareTo(ct2.name());
            } else if (ct1 == R) {
                return 1;
            } else {
                return -1;
            }
        }
    }

    private String constraintName;
    private boolean generatedName;
    private ConstraintType constraintType;
    private String tableName;
    private String searchCondition;
    private String refConstraintName;

    public Constraint(String constraintName, boolean generatedName, ConstraintType constraintType,
                      String tableName, String searchCondition, String refConstraintName) {
        this.constraintName = constraintName;
        this.generatedName = generatedName;
        this.constraintType = constraintType;
        this.tableName = tableName;
        this.searchCondition = searchCondition;
        this.refConstraintName = refConstraintName;
    }

    public String getConstraintName() {
        return constraintName;
    }

    public void setConstraintName(String constraintName) {
        this.constraintName = constraintName;
    }

    public boolean isGeneratedName() {
        return generatedName;
    }

    public void setGeneratedName(boolean generatedName) {
        this.generatedName = generatedName;
    }

    public ConstraintType getConstraintType() {
        return constraintType;
    }

    public void setConstraintType(ConstraintType constraintType) {
        this.constraintType = constraintType;
    }

    public String getTableName() {
        return tableName;
    }

    public void setTableName(String tableName) {
        this.tableName = tableName;
    }

    public String getSearchCondition() {
        return searchCondition;
    }

    public void setSearchCondition(String searchCondition) {
```

```
78          this.searchCondition = searchCondition;
79      }
80
81      public String getRefConstraintName() {
82          return refConstraintName;
83      }
84
85      public void setRefConstraintName(String refConstraintName) {
86          this.refConstraintName = refConstraintName;
87      }
88
89      @Override
90      public String toString() {
91          return "Constraint{" +
92                  "constraintName='" + constraintName + '\'' +
93                  ", generatedName=" + generatedName +
94                  ", constraintType=" + constraintType +
95                  ", tableName='" + tableName + '\'' +
96                  ", searchCondition='" + searchCondition + '\'' +
97                  ", refConstraintName='" + refConstraintName + '\'' +
98                  '}';
99      }
100 }
101
```

```java
package stakashka.ssm.core.data;

public class ConstraintColumn {

    private String constraintName;
    private String tableName;
    private String columnName;
    private Integer position;

    public ConstraintColumn(String constraintName, String tableName, String columnName, Integer position) {
        this.constraintName = constraintName;
        this.tableName = tableName;
        this.columnName = columnName;
        this.position = position;
    }

    public String getConstraintName() {
        return constraintName;
    }

    public void setConstraintName(String constraintName) {
        this.constraintName = constraintName;
    }

    public String getTableName() {
        return tableName;
    }

    public void setTableName(String tableName) {
        this.tableName = tableName;
    }

    public String getColumnName() {
        return columnName;
    }

    public void setColumnName(String columnName) {
        this.columnName = columnName;
    }

    public Integer getPosition() {
        return position;
    }

    public void setPosition(Integer position) {
        this.position = position;
    }

    @Override
    public String toString() {
        return "ConstraintColumn{" +
                "constraintName='" + constraintName + '\'' +
                ", tableName='" + tableName + '\'' +
                ", columnName='" + columnName + '\'' +
                ", position=" + position +
                '}';
    }
}

```

```java
package stakashka.ssm.core.model;

import stakashka.ssm.api.db.AbstractDatabase;
import stakashka.ssm.api.db.oracle.OracleDatabase;
import stakashka.ssm.api.db.postgres.PostgresDatabase;
import stakashka.ssm.core.data.Schema;

import java.io.FileWriter;
import java.io.IOException;

public class Controller {

    public enum Databases {
        Oracle,
        Postgres
    }

    private String urlFrom;
    private String dbUser;
    private String dbPassword;
    private Databases dbFrom;
    private Databases dbTo;
    private String schemaFrom;
    private String schemaTo;

    public Controller(String urlFrom, String dbUser, String dbPassword, String schemaFrom,
                      Databases dbFrom, String schemaTo, Databases dbTo) {
        this.urlFrom = urlFrom;
        this.dbUser = dbUser;
        this.dbPassword = dbPassword;
        this.dbFrom = dbFrom;
        this.dbTo = dbTo;
        this.schemaFrom = schemaFrom;
        this.schemaTo = schemaTo;
    }

    public void process() {
        try {
            AbstractDatabase databaseFrom = null;
            AbstractDatabase databaseTo = null;
            Schema schema = null;
            switch (dbFrom) {
                case Oracle:
                    databaseFrom = new OracleDatabase(urlFrom, dbUser, dbPassword);
                    break;
            }
            if (databaseFrom != null) {
                schema = databaseFrom.getSchema();
                System.out.println(schema.getTablesList());
                System.out.println(schema.getColumnsList());
                System.out.println(schema.getConstraintsList());
                System.out.println(schema.getConstraintColumnsList());
            }

            switch (dbTo) {
                case Postgres:
                    databaseTo = new PostgresDatabase(schemaTo);
                    break;
            }
            if (databaseTo != null && schema != null) {
                writeToFile(databaseTo.crateDDL(schema));
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void writeToFile(String text) {
        try(FileWriter writer = new FileWriter("createbase.sql", false)) {
            writer.write(text);
            writer.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```