

```
1 package stakashka.ssm;
2
3 import stakashka.ssm.core.model.Controller;
4 import stakashka.ssm.core.model.Controller.Databases;
5
6 public class Main {
7
8     public static void main(String[] args) {
9         String urlFrom = "jdbc:oracle:thin:@localhost:1521:orcl";
10        String user = "ewa";
11        String password = "ewapas";
12        String schemaFrom = "ewa";
13        Databases databaseFrom = Databases.Oracle;
14        String schemaTo = "ewa";
15        Databases databaseTo = Databases.Postgres;
16
17        Controller controller = new Controller(urlFrom, user, password, schemaFrom, databaseFrom,
18            schemaTo, databaseTo);
18        controller.process();
19    }
20 }
21
```

```
1 package stakashka.ssm.api.db;  
2  
3 public interface TypesTemplates {  
4     String getTypeTemplate(String[] args);  
5 }  
6
```

```

1 package stakashka.ssm.api.db;
2
3 import stakashka.ssm.core.data.*;
4 import stakashka.ssm.core.data.Constraint.ConstraintType;
5 import stakashka.ssm.core.data.DataTypes.NormalizedTypes;
6
7 import java.sql.*;
8 import java.util.HashMap;
9 import java.util.List;
10 import java.util.Map;
11 import java.util.stream.Collectors;
12
13 abstract public class AbstractDatabase {
14
15     protected String schemaFrom;
16     protected String schemaTo;
17     protected Connection connection = null;
18     protected Statement statement = null;
19     protected ResultSet resultSet = null;
20     // TODO : unique values and think about it more
21     protected Map<TypesTemplates, NormalizedTypes> typesMapping = null;
22     protected Map<NormalizedTypes, TypesTemplates> inverseTypesMapping = null;
23
24     public AbstractDatabase(String schemaTo) throws Exception {
25         this.schemaTo = schemaTo;
26         initTypesMapping();
27         initInverseTypesMapping();
28     }
29
30     public AbstractDatabase(String url, String user, String password, String schemaFrom) throws
SQLException {
31         this.schemaFrom = schemaFrom;
32         initTypesMapping();
33         initDriver();
34         try {
35             this.connection = DriverManager.getConnection(url, user, password);
36         } catch (SQLException e) {
37             e.printStackTrace();
38             try {
39                 if (connection != null) connection.close();
40             } catch (Exception ee) {
41                 ee.printStackTrace();
42             }
43             throw e;
44         }
45     }
46
47     protected abstract void initTypesMapping();
48
49     protected void initInverseTypesMapping() throws Exception {
50         inverseTypesMapping = new HashMap<>();
51         for (TypesTemplates key : typesMapping.keySet()) {
52             inverseTypesMapping.put(typesMapping.get(key), key);
53         }
54         if (inverseTypesMapping.size() != typesMapping.size()) {
55             throw new Exception();
56         }
57     }
58
59     public Schema getSchema() throws SQLException {
60         try {
61             Schema schema = new Schema();
62             schema.setTablesList(getTablesList());
63             schema.setColumnsList(getColumnsList());
64             schema.setConstraintsList(getConstraintsList());
65             schema.setConstraintColumnsList(getConstraintColumnsList());
66             return schema;
67         } catch (SQLException e) {
68             e.printStackTrace();
69             throw e;
70         } finally {
71             try {
72                 if (connection != null) connection.close();
73             } catch (Exception e) {
74                 e.printStackTrace();
75             }
76         }
77     }

```

```

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 createDDL(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 }

```

```

1 package stakashka.ssm.api.db.oracle;
2
3 import stakashka.ssm.api.db.AbstractDatabase;
4 import stakashka.ssm.api.db.TypesTemplates;
5 import stakashka.ssm.core.data.*;
6 import stakashka.ssm.core.data.Constraint.ConstraintType;
7
8 import java.sql.*;
9 import java.util.ArrayList;
10 import java.util.HashMap;
11 import java.util.List;
12
13 public class OracleDatabase extends AbstractDatabase {
14
15     private enum ColumnTypes implements TypesTemplates {
16         VARCHAR2,
17         NUMBER;
18
19         @Override
20         public String getTypeTemplate(String[] args) {
21             return null;
22         }
23     }
24
25     public OracleDatabase(String schemaTo) throws Exception {
26         super(schemaTo);
27     }
28
29     public OracleDatabase(String url, String user, String password) throws SQLException {
30         super(url, user, password, null);
31         schemaFrom = connection.getMetaData().getUserName();
32     }
33
34     @Override
35     protected void initTypesMapping() {
36         typesMapping = new HashMap<TypesTemplates, DataTypes.NormalizedTypes>() {{
37             put(ColumnTypes.NUMBER, DataTypes.NormalizedTypes.NUMBER);
38             put(ColumnTypes.VARCHAR2, DataTypes.NormalizedTypes.VARCHAR);
39         }};
40     }
41
42     @Override
43     protected void initDriver() {
44         try {
45             Class.forName("oracle.jdbc.driver.OracleDriver");
46         } catch (ClassNotFoundException e) {
47             e.printStackTrace();
48         }
49     }
50
51     @Override
52     public List<Table> getTablesList() throws SQLException {
53         List<Table> tablesList = new ArrayList<Table>();
54         statement = connection.createStatement();
55         resultSet = statement.executeQuery("select TABLE_NAME from SYS.ALL_TABLES where OWNER = '" +
schemaFrom + "'");
56         while (resultSet.next()) {
57             tablesList.add(new Table(resultSet.getString(1)));
58         }
59         return tablesList;
60     }
61
62     @Override
63     protected List<Column> getColumnsList() throws SQLException {
64         List<Column> columnsList = new ArrayList<Column>();
65         statement = connection.createStatement();
66         resultSet = statement.executeQuery(
67             "select COLUMN_NAME, TABLE_NAME, DATA_TYPE, DATA_LENGTH, DATA_PRECISION, DATA_SCALE,
NULLABLE " +
68                     "from SYS.ALL_TAB_COLS where OWNER = '" + schemaFrom + "' order by COLUMN_ID"
69         );
70         while (resultSet.next()) {
71             columnsList.add(new Column(
72                 resultSet.getString(1), resultSet.getString(2),
73                 getNormalizedType(ColumnTypes.valueOf(resultSet.getString(3))),
74                 resultSet.getInt(4), resultSet.getInt(5), resultSet.getInt(6),
75                 resultSet.getString(7).equals("Y"))
76         );
77     }
78 }

```

```

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 {
126     return null;
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

```

```

1 package stakashka.ssm.api.db.postgres;
2
3 import stakashka.ssm.api.db.AbstractDatabase;
4 import stakashka.ssm.api.db.TypesTemplates;
5 import stakashka.ssm.core.data.*;
6
7 import java.sql.SQLException;
8 import java.util.HashMap;
9 import java.util.List;
10 import java.util.stream.Collectors;
11
12 public class PostgresDatabase extends AbstractDatabase {
13
14     private enum ColumnTypes implements TypesTemplates {
15         CHARACTER,
16         NUMERIC;
17
18         @Override
19         public String getTypeTemplate(String[] args) {
20             switch (this) {
21                 case CHARACTER:
22                     return String.format("CHARACTER(%s)", args[0]);
23                 case NUMERIC:
24                     return String.format("NUMERIC(%s,%s)", args[1], args[2]);
25                 default:
26                     return this.name();
27             }
28         }
29     }
30
31     public PostgresDatabase(String schemaTo) throws Exception {
32         super(schemaTo);
33     }
34
35     public PostgresDatabase(String url, String user, String password, String schemaFrom) throws
SQLException {
36         super(url, user, password, schemaFrom);
37     }
38
39     @Override
40     protected void initTypesMapping() {
41         typesMapping = new HashMap<TypesTemplates, DataTypes.NormalizedTypes>() {{
42             put(ColumnTypes.NUMERIC, DataTypes.NormalizedTypes.NUMBER);
43             put(ColumnTypes.CHARACTER, DataTypes.NormalizedTypes.VARCHAR);
44         }};
45     }
46
47     @Override
48     protected void initDriver() {
49
50     }
51
52     @Override
53     protected List<Table> getTablesList() throws SQLException {
54         return null;
55     }
56
57     @Override
58     protected List<Column> getColumnsList() throws SQLException {
59         return null;
60     }
61
62     @Override
63     protected List<Constraint> getConstraintsList() throws SQLException {
64         return null;
65     }
66
67     @Override
68     protected List<ConstraintColumn> getConstraintColumnsList() throws SQLException {
69         return null;
70     }
71
72     @Override
73     protected String createTableDDL(String tableName, List<Column> columnsList) throws Exception {
74         String res = "";
75         res += "CREATE TABLE " + ((schemaTo == null) ? "" : schemaTo + ".") + tableName + " (" +
76         for (Column column : columnsList) {

```

```

77         String type = getSpecificType(column.getColumnType())
78             .getTypeTemplate(new String[]{column.getLength().toString(), column.getPrecision()
79             .toString(), column.getScale().toString()}));
80         res += "\n    " + column.getColumnName() + " " + type + ((column.isNullable()) ? "" : "
81     not null") + ",";
82     }
83     res = res.substring(0, res.length() - 1);
84     res += "\n);\n";
85     return res;
86 }
87
88 @Override
89 protected String createConstraintDDL(Constraint constraint, List<ConstraintColumn> columns,
90                                     Constraint refConstraint, List<ConstraintColumn> refColumns)
91 {
92     String res = "ALTER TABLE " + ((schemaTo == null) ? "" : schemaTo + ".") + constraint.
93     getTableName() + " ADD"
94         + (constraint.isGeneratedName() ? "" : " CONSTRAINT " + constraint.getConstraintName(
95     ) + " ");
96     switch (constraint.getConstraintType()) {
97     case C:
98         if (!constraint.getSearchCondition().contains("IS NOT NULL")) {
99             res += "CHECK (" + constraint.getSearchCondition() + ");\n";
100        } else {
101            return "";
102        }
103        break;
104    case O:
105        break;
106    case P:
107        res += "PRIMARY KEY (";
108        columns = columns.stream().sorted((c1, c2) -> Integer.compare(c1.getPosition(), c2.
109        getPosition()))
110            .collect(Collectors.toList());
111        for (ConstraintColumn column: columns) {
112            res += column.getColumnName() + ", ";
113        }
114        res = res.substring(0, res.length() - 2) + ");\n";
115        break;
116    case U:
117        res += "UNIQUE (";
118        columns = columns.stream().sorted((c1, c2) -> Integer.compare(c1.getPosition(), c2.
119        getPosition()))
120            .collect(Collectors.toList());
121        for (ConstraintColumn column: columns) {
122            res += column.getColumnName() + ", ";
123        }
124        res = res.substring(0, res.length() - 2) + ");\n";
125    case V:
126        break;
127    case R:
128        res += "FOREIGN KEY (" + columns.get(0).getColumnName() + ") REFERENCES "
129            + ((schemaTo == null) ? "" : schemaTo + ".") + refConstraint.getTableName()
130            + " (" + refColumns.get(0).getColumnName() + ");\n";
131        break;
132    }
133 }
134
135 return res;
136 }
137
138 }
139
140 }
141
142 }
143
144 }
145
146 }
147
148 }
149
150 }
151
152 }
153
154 }
155
156 }
157
158 }
159
160 }
161
162 }
163
164 }
165
166 }
167
168 }
169
170 }
171
172 }
173
174 }
175
176 }
177
178 }
179
180 }
181
182 }
183
184 }
185
186 }
187
188 }
189
190 }
191
192 }
193
194 }
195
196 }
197
198 }
199
200 }
201
202 }
203
204 }
205
206 }
207
208 }
209
210 }
211
212 }
213
214 }
215
216 }
217
218 }
219
220 }
221
222 }
223
224 }
225
226 }
227
228 }
229
230 }
231
232 }
233
234 }
235
236 }
237
238 }
239
240 }
241
242 }
243
244 }
245
246 }
247
248 }
249
250 }
251
252 }
253
254 }
255
256 }
257
258 }
259
260 }
261
262 }
263
264 }
265
266 }
267
268 }
269
270 }
271
272 }
273
274 }
275
276 }
277
278 }
279
280 }
281
282 }
283
284 }
285
286 }
287
288 }
289
290 }
291
292 }
293
294 }
295
296 }
297
298 }
299
300 }
301
302 }
303
304 }
305
306 }
307
308 }
309
310 }
311
312 }
313
314 }
315
316 }
317
318 }
319
320 }
321
322 }
323
324 }
325
326 }
327
328 }
329
330 }
331
332 }
333
334 }
335
336 }
337
338 }
339
340 }
341
342 }
343
344 }
345
346 }
347
348 }
349
350 }
351
352 }
353
354 }
355
356 }
357
358 }
359
360 }
361
362 }
363
364 }
365
366 }
367
368 }
369
370 }
371
372 }
373
374 }
375
376 }
377
378 }
379
380 }
381
382 }
383
384 }
385
386 }
387
388 }
389
390 }
391
392 }
393
394 }
395
396 }
397
398 }
399
400 }
401
402 }
403
404 }
405
406 }
407
408 }
409
410 }
411
412 }
413
414 }
415
416 }
417
418 }
419
420 }
421
422 }
423
424 }
425
426 }
427
428 }
429
430 }
431
432 }
433
434 }
435
436 }
437
438 }
439
440 }
441
442 }
443
444 }
445
446 }
447
448 }
449
450 }
451
452 }
453
454 }
455
456 }
457
458 }
459
460 }
461
462 }
463
464 }
465
466 }
467
468 }
469
470 }
471
472 }
473
474 }
475
476 }
477
478 }
479
480 }
481
482 }
483
484 }
485
486 }
487
488 }
489
490 }
491
492 }
493
494 }
495
496 }
497
498 }
499
500 }
501
502 }
503
504 }
505
506 }
507
508 }
509
510 }
511
512 }
513
514 }
515
516 }
517
518 }
519
520 }
521
522 }
523
524 }
525
526 }
527
528 }
529
530 }
531
532 }
533
534 }
535
536 }
537
538 }
539
540 }
541
542 }
543
544 }
545
546 }
547
548 }
549
550 }
551
552 }
553
554 }
555
556 }
557
558 }
559
560 }
561
562 }
563
564 }
565
566 }
567
568 }
569
570 }
571
572 }
573
574 }
575
576 }
577
578 }
579
580 }
581
582 }
583
584 }
585
586 }
587
588 }
589
590 }
591
592 }
593
594 }
595
596 }
597
598 }
599
600 }
601
602 }
603
604 }
605
606 }
607
608 }
609
610 }
611
612 }
613
614 }
615
616 }
617
618 }
619
620 }
621
622 }
623
624 }
625
626 }
627
628 }
629
630 }
631
632 }
633
634 }
635
636 }
637
638 }
639
640 }
641
642 }
643
644 }
645
646 }
647
648 }
649
650 }
651
652 }
653
654 }
655
656 }
657
658 }
659
660 }
661
662 }
663
664 }
665
666 }
667
668 }
669
670 }
671
672 }
673
674 }
675
676 }
677
678 }
679
680 }
681
682 }
683
684 }
685
686 }
687
688 }
689
690 }
691
692 }
693
694 }
695
696 }
697
698 }
699
700 }
701
702 }
703
704 }
705
706 }
707
708 }
709
710 }
711
712 }
713
714 }
715
716 }
717
718 }
719
720 }
721
722 }
723
724 }
725
726 }
727
728 }
729
730 }
731
732 }
733
734 }
735
736 }
737
738 }
739
740 }
741
742 }
743
744 }
745
746 }
747
748 }
749
750 }
751
752 }
753
754 }
755
756 }
757
758 }
759
759 }
760
761 }
762
763 }
764
765 }
766
767 }
768
769 }
770
771 }
772
773 }
774
775 }
776
777 }
778
779 }
779 }
780
781 }
782
783 }
784
785 }
786
787 }
788
789 }
789 }
790
791 }
792
793 }
794
795 }
796
797 }
798
799 }
799 }
800
801 }
802
803 }
804
805 }
806
807 }
808
809 }
809 }
810
811 }
812
813 }
814
815 }
816
817 }
818
819 }
819 }
820
821 }
822
823 }
824
825 }
826
827 }
828
829 }
829 }
830
831 }
832
833 }
834
835 }
836
837 }
838
839 }
839 }
840
841 }
842
843 }
844
845 }
846
847 }
848
849 }
849 }
850
851 }
852
853 }
854
855 }
856
857 }
858
859 }
859 }
860
861 }
862
863 }
864
865 }
866
867 }
868
869 }
869 }
870
871 }
872
873 }
874
875 }
876
877 }
878
879 }
879 }
880
881 }
882
883 }
884
885 }
886
887 }
888
889 }
889 }
890
891 }
892
893 }
894
895 }
896
897 }
898
899 }
899 }
900
901 }
902
903 }
904
905 }
906
907 }
908
909 }
909 }
910
911 }
912
913 }
914
915 }
916
917 }
918
919 }
919 }
920
921 }
922
923 }
924
925 }
926
927 }
928
929 }
929 }
930
931 }
932
933 }
934
935 }
936
937 }
938
939 }
939 }
940
941 }
942
943 }
944
945 }
946
947 }
948
949 }
949 }
950
951 }
952
953 }
954
955 }
956
957 }
958
959 }
959 }
960
961 }
962
963 }
964
965 }
966
967 }
968
969 }
969 }
970
971 }
972
973 }
974
975 }
976
977 }
978
979 }
979 }
980
981 }
982
983 }
984
985 }
986
987 }
988
989 }
989 }
990
991 }
992
993 }
994
995 }
996
997 }
998
999 }
999 }

```

```
1 package stakashka.ssm.core.data;
2
3 public class Table {
4     private String tableName;
5
6     public Table(String tableName) {
7         this.tableName = tableName;
8     }
9
10    public String getTableName() {
11        return tableName;
12    }
13
14    public void setTableName(String tableName) {
15        this.tableName = tableName;
16    }
17
18    @Override
19    public String toString() {
20        return "Table{" +
21            "tableName='" + tableName + '\'' +
22            '}';
23    }
24}
25
```

```

1 package stakashka.ssm.core.data;
2
3 public class Column {
4
5     private String columnName;
6     private String tableName;
7     private DataTypes.NormalizedTypes columnType;
8     private Integer length;
9     private Integer precision;
10    private Integer scale;
11    private boolean nullable;
12
13    public Column(String columnName, String tableName, DataTypes.NormalizedTypes columnType,
14                  Integer length, Integer precision, Integer scale, boolean nullable) {
15        this.columnName = columnName;
16        this.tableName = tableName;
17        this.columnType = columnType;
18        this.length = length;
19        this.precision = precision;
20        this.scale = scale;
21        this.nullable = nullable;
22    }
23
24    public String getColumnName() {
25        return columnName;
26    }
27
28    public void setColumnName(String columnName) {
29        this.columnName = columnName;
30    }
31
32    public String getTableName() {
33        return tableName;
34    }
35
36    public void setTableName(String tableName) {
37        this.tableName = tableName;
38    }
39
40    public DataTypes.NormalizedTypes getColumnType() {
41        return columnType;
42    }
43
44    public void setColumnType(DataTypes.NormalizedTypes columnType) {
45        this.columnType = columnType;
46    }
47
48    public Integer getLength() {
49        return length;
50    }
51
52    public void setLength(Integer length) {
53        this.length = length;
54    }
55
56    public Integer getPrecision() {
57        return precision;
58    }
59
60    public void setPrecision(Integer precision) {
61        this.precision = precision;
62    }
63
64    public Integer getScale() {
65        return scale;
66    }
67
68    public void setScale(Integer scale) {
69        this.scale = scale;
70    }
71
72    public boolean isNullable() {
73        return nullable;
74    }
75
76    public void setNullable(boolean nullable) {
77        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
```

```

1 package stakashka.ssm.core.data;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Schema {
7
8     private List<Table> tablesList;
9     private List<Column> columnsList;
10    private List<Constraint> constraintsList;
11    private List<ConstraintColumn> constraintColumnsList;
12
13
14    public Schema() {
15        tablesList = new ArrayList<>();
16        columnsList = new ArrayList<>();
17        constraintsList = new ArrayList<>();
18        constraintColumnsList = new ArrayList<>();
19    }
20
21    public List<Table> getTablesList() {
22        return tablesList;
23    }
24
25    public void setTablesList(List<Table> tablesList) {
26        this.tablesList = tablesList;
27    }
28
29    public List<Column> getColumnsList() {
30        return columnsList;
31    }
32
33    public void setColumnsList(List<Column> columnsList) {
34        this.columnsList = columnsList;
35    }
36
37    public List<Constraint> getConstraintsList() {
38        return constraintsList;
39    }
40
41    public void setConstraintsList(List<Constraint> constraintsList) {
42        this.constraintsList = constraintsList;
43    }
44
45    public List<ConstraintColumn> getConstraintColumnsList() {
46        return constraintColumnsList;
47    }
48
49    public void setConstraintColumnsList(List<ConstraintColumn> constraintColumnsList) {
50        this.constraintColumnsList = constraintColumnsList;
51    }
52
53    @Override
54    public String toString() {
55        return "Schema{" +
56                "tablesList=" + tablesList +
57                ", columnsList=" + columnsList +
58                ", constraintsList=" + constraintsList +
59                ", constraintColumnsList=" + constraintColumnsList +
60                '}';
61    }
62}
63

```

```
1 package stakashka.ssm.core.data;  
2  
3 public class DataTypes {  
4     public enum NormalizedTypes {  
5         NUMBER,  
6         VARCHAR  
7     }  
8 }  
9
```

```

1 package stakashka.ssm.core.data;
2
3 public class Constraint {
4
5     public enum ConstraintType {
6         C,
7         O,
8         P,
9         U,
10        V,
11        R;
12        // foreign keys must be created last
13        public static int compare(ConstraintType ct1, ConstraintType ct2) {
14            if (ct1 == ct2 || (ct1 != R && ct2 != R)) {
15                return ct1.name().compareTo(ct2.name());
16            } else if (ct1 == R) {
17                return 1;
18            } else {
19                return -1;
20            }
21        }
22    }
23
24    private String constraintName;
25    private boolean generatedName;
26    private ConstraintType constraintType;
27    private String tableName;
28    private String searchCondition;
29    private String refConstraintName;
30
31    public Constraint(String constraintName, boolean generatedName, ConstraintType constraintType,
32                      String tableName, String searchCondition, String refConstraintName) {
33        this.constraintName = constraintName;
34        this.generatedName = generatedName;
35        this.constraintType = constraintType;
36        this.tableName = tableName;
37        this.searchCondition = searchCondition;
38        this.refConstraintName = refConstraintName;
39    }
40
41    public String getConstraintName() {
42        return constraintName;
43    }
44
45    public void setConstraintName(String constraintName) {
46        this.constraintName = constraintName;
47    }
48
49    public boolean isGeneratedName() {
50        return generatedName;
51    }
52
53    public void setGeneratedName(boolean generatedName) {
54        this.generatedName = generatedName;
55    }
56
57    public ConstraintType getConstraintType() {
58        return constraintType;
59    }
60
61    public void setConstraintType(ConstraintType constraintType) {
62        this.constraintType = constraintType;
63    }
64
65    public String getTableName() {
66        return tableName;
67    }
68
69    public void setTableName(String tableName) {
70        this.tableName = tableName;
71    }
72
73    public String getSearchCondition() {
74        return searchCondition;
75    }
76
77    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
```

```

1 package stakashka.ssm.core.data;
2
3 public class ConstraintColumn {
4
5     private String constraintName;
6     private String tableName;
7     private String columnName;
8     private Integer position;
9
10    public ConstraintColumn(String constraintName, String tableName, String columnName, Integer
11        position) {
12        this.constraintName = constraintName;
13        this.tableName = tableName;
14        this.columnName = columnName;
15        this.position = position;
16    }
17
18    public String getConstraintName() {
19        return constraintName;
20    }
21
22    public void setConstraintName(String constraintName) {
23        this.constraintName = constraintName;
24    }
25
26    public String getTableName() {
27        return tableName;
28    }
29
30    public void setTableName(String tableName) {
31        this.tableName = tableName;
32    }
33
34    public String getColumnName() {
35        return columnName;
36    }
37
38    public void setColumnName(String columnName) {
39        this.columnName = columnName;
40    }
41
42    public Integer getPosition() {
43        return position;
44    }
45
46    public void setPosition(Integer position) {
47        this.position = position;
48    }
49
50    @Override
51    public String toString() {
52        return "ConstraintColumn{" +
53            "constraintName='" + constraintName + '\'' +
54            ", tableName='" + tableName + '\'' +
55            ", columnName='" + columnName + '\'' +
56            ", position=" + position +
57            '}';
58    }
59

```

```

1 package stakashka.ssm.core.model;
2
3 import stakashka.ssm.api.db.AbstractDatabase;
4 import stakashka.ssm.api.db.oracle.OracleDatabase;
5 import stakashka.ssm.api.db.postgres.PostgresDatabase;
6 import stakashka.ssm.core.data.Schema;
7
8 import java.io.FileWriter;
9 import java.io.IOException;
10
11 public class Controller {
12
13     public enum Databases {
14         Oracle,
15         Postgres
16     }
17
18     private String urlFrom;
19     private String dbUser;
20     private String dbPassword;
21     private Databases dbFrom;
22     private Databases dbTo;
23     private String schemaFrom;
24     private String schemaTo;
25
26     public Controller(String urlFrom, String dbUser, String dbPassword, String schemaFrom,
27                       Databases dbFrom, String schemaTo, Databases dbTo) {
28         this.urlFrom = urlFrom;
29         this.dbUser = dbUser;
30         this.dbPassword = dbPassword;
31         this.dbFrom = dbFrom;
32         this.dbTo = dbTo;
33         this.schemaFrom = schemaFrom;
34         this.schemaTo = schemaTo;
35     }
36
37     public void process() {
38         try {
39             AbstractDatabase databaseFrom = null;
40             AbstractDatabase databaseTo = null;
41             Schema schema = null;
42             switch (dbFrom) {
43                 case Oracle:
44                     databaseFrom = new OracleDatabase(urlFrom, dbUser, dbPassword);
45                     break;
46             }
47             if (databaseFrom != null) {
48                 schema = databaseFrom.getSchema();
49                 System.out.println(schema.getTablesList());
50                 System.out.println(schema.getColumnsList());
51                 System.out.println(schema.getConstraintsList());
52                 System.out.println(schema.getConstraintColumnsList());
53             }
54
55             switch (dbTo) {
56                 case Postgres:
57                     databaseTo = new PostgresDatabase(schemaTo);
58                     break;
59             }
60             if (databaseTo != null && schema != null) {
61                 writeToFile(databaseTo.createDDL(schema));
62             }
63
64         } catch (Exception e) {
65             e.printStackTrace();
66         }
67     }
68
69     private void writeToFile(String text) {
70         try(FileWriter writer = new FileWriter("createbase.sql", false)) {
71             writer.write(text);
72             writer.flush();
73         } catch (IOException e) {
74             e.printStackTrace();
75         }
76     }
77 }

```