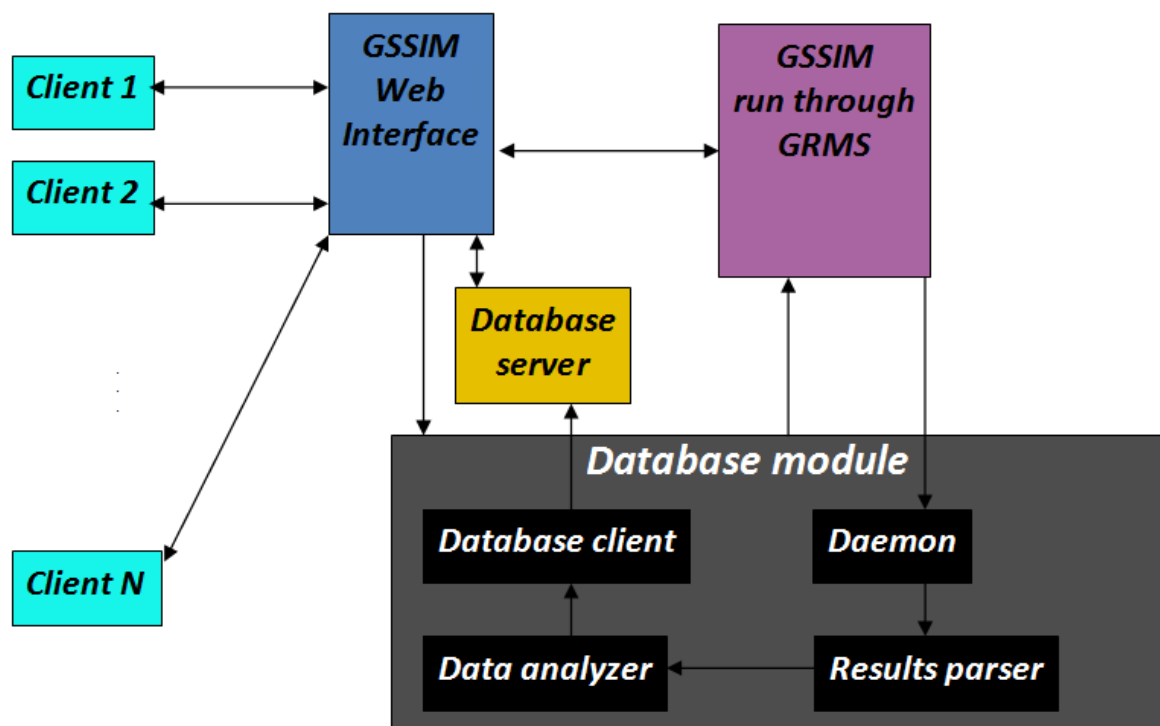


GSSIM database module documentation

Jarosław Szymczak <jarek.szymczak@gmail.com>

May 10, 2011

1 Structure of GSSIM service



Clients are visiting GSSIM website, which interface is written as a separate task implemented by another person and it's not included in this documentation. On the website clients are choosing specific options for experiment and then execute it. Execution is performed by GRMS webservice which is accessed by the web interface module.

Before job (GSSIM with specific simulation) is started some preparations must be done. It includes copying all the necessary files to data server. After that GRMS downloads these files, runs GSSIM and uploads results (all these actions are defined in GRMS configuration file).

2 Database module documentation

2.1 General usage

As depicted on first page, database module consist of four parts. They are strictly cooperating with each other. Daemon is running in background an monitoring file system (specified directory) for changes (new files transferred by GRMS after performing the experiment). When such changes are detected it looks for *.tar file with the GSSIM results. Once it finds the file the Data analyzer is run, it extracts files from result archive, uses Results parser to collect raw data and analyzes and puts the data into a database. To communicate with database Database client is used. The Database client is also used in web-interface module to retrieve data from the database.

2.2 Daemon

Daemon is an application written in C++ which awaits for the file with GSSIM experiments results. Once the file appears daemon triggers the analyzer module to analyze the file and insert data from it to the database. Daemon configuration is presented on the listing 1, file which contains it must be named daemon.config and placed in RUNNING_DIR (please see the listing)

```
1 /opt/app/gssim/
2 start-db-analyzer.sh
3 /opt/app/gssim/results/
4
5 Lines are corresponding to:
6 1) directory in which gssim is installed
7 2) name of file that should be executed
8 3) base directory which should be watched
9
10 Path to daemon running directory is precompiled
11 and defined in source code as RUNNING_DIR.
12 Currently RUNNING_DIR is /opt/app/gssim/daemon
```

Listing 1: Daemon configuration file

Once daemon is run it closes standard input and output. It also creates daemon.lock file which ensures that only one instance of daemon is running. It contains pid of currently running daemon and it can be used to kill it (with term signal).

2.3 Results parser

Results parser can be found in GSSIM project in gssim.db.reader package. It consist of several classes. In general it is based on StatsParser and StatsReader classess. StatsReader class is wrapped by a specialized readers (for certain files containing data with experiments results). Let explanation of the idea be supported by the example listing 2

```
1 public class StatsParser extends BufferedReader {
2
3     protected String separator = ";";
4
5     public StatsParser(Reader in, boolean skipHeaders) throws IOException {
6         // not important or obvious code
7     }
8
9     public void setSeparator(String arg){
10        // not important or obvious code
11    }
12
13    public String readLine() throws IOException{
14        // it skips empty lines
15    }
16
17    public String[] readValues() throws IOException{
18        // not important or obvious code
19    }
20
21 }
22
23 public abstract class StatsReader {
24
25     protected StatsParser parser;
26     protected String[] values;
27     protected NumberFormat format = GridSchedulingSimulator.DEFAULT_NUMBER_FORMAT;
28     public StatsReader(StatsParser p) {
29         // not important or obvious code
30     }
31
32     public boolean next() {
33         // values array is filled here
34     }
35
36     protected double getAsDouble(int position) {
```

```

37     // not important or obvious code
38 }
39
40 protected float getAsFloat(int position) {
41     // not important or obvious code
42 }
43
44 protected long getAsLong(int position) {
45     // not important or obvious code
46 }
47
48 protected int getAsInt(int position) {
49     // not important or obvious code
50 }
51
52 public class UsageReader extends StatsReader {
53
54     protected String resourceName;
55
56     public UsageReader(StatsParser p) {
57         super(p);
58     }
59
60     public String getName(){
61         return this.values[0];
62     }
63
64     public Timestamp getTimeStamp(){
65         return new Timestamp(getAsLong(1));
66     }
67
68     public Float getUsage(){
69         return getAsFloat(2);
70     }
71 }
72
73 private void processResourceAllTimeline(InputStream is, boolean skipHeaders)
74     throws SQLException, IOException {
75
76     // not important or obvious code
77
78     ResourceAllTimeline resourceAllTimeline = null;
79
80     List<ResourceAllTimeline> resourceAllTimelines = new ArrayList<ResourceAllTimeline>();
81
82     UsageReader resourceAllTimelineReader = new UsageReader(getStatsParser(
83         is, skipHeaders));
84
85     while (resourceAllTimelineReader.next()) {
86
87         resourceAllTimeline = new ResourceAllTimeline();
88         resourceAllTimeline.setFkResourceId(resourceID
89             .get(resourceAllTimelineReader.getName()));
90         resourceAllTimeline.setTimestamp(resourceAllTimelineReader
91             .getTimeStamp());
92         resourceAllTimeline.setUsage(resourceAllTimelineReader.getUsage());
93         resourceAllTimelines.add(resourceAllTimeline);
94     }
95
96     // not important or obvious code
97 }

```

Listing 2: Example of results parser implementation for ResourceAllTimeline

The idea is very simple. StatsParser reads data from input stream line after line skipping empty lines. It has a method readValues() which uses separator char to obtain array of strings containing data. StatsReader allows obtain the data in certain format (if it's possible to convert string into it) by choosing and index (column) in values. Every reader, such as UsageReader is wrapping StatsReader methods with certain data collecting methods (usage of fields names instead of indexes). Thanks to such construction adding additional readers is very simple.

2.4 Data analyzer

Data analyzer class (`gssim.db.analyzer.Analyzer`) is a core of processing data from files and putting them into a database process. As a parameter must be provided a *.tar file or directory with experiment results. Additional parameters are connected with the configuration. Depending on number of extra (apart experiment results path) arguments used, the configuration will be loaded in different ways:

- no configuration argument specified: default configuration will be loaded
- one additional argument: database address, username and password will be loaded from properties file which path is an argument
- two additional arguments: first argument is encrypted properties file (as above), second one is the private key of RSA algorithm which will be used to decrypt the file

Configuration of Analyzer is stored in a class which implements `gssim.db.config.Config` class. Fields of this class have JavaDOC, so they will not be described here. Worth to mention is fact, that Config class contains data needed to connect to the database and names of the files which should be analyzed. It also determines whether *.tar file with experiment results will be extracted to RAM or disk.

DataAnalyzer is a class which should be run as an application. Apart run and main it has only private methods and they will not be described here.

In *.tar archive properties file should be present. If it contains *experiment.id* it will use the existing data from database connected with experiment. Otherwise a simple experiment will be created with default name.

2.5 Database client

Database client is a general DAO pattern designed access to database with results of GSSIM experiments. It's strictly connected with GSSIM results database scheme. It contains of the following packages and classes:

- `gssim.db.dao` – package with interfaces containing CRUD database method for the certain classes
- `gssim.db.data` – classes representing certain relations in database
- `gssim.db.find` – package containing
- `gssim.db.mysql` – classes implementing interfaces from `gssim.db.dao` and provides the connection with MySQL database
- `gssim.db.utils` – package containing helpful classes. It contains (among others) `ResultSetAdapter` and `PreparedStatementAdapter` classes, which are used to handle simple types as objects (standard `ResultSet` and `PreparedStatement` interfaces doesn't provide methods obtaining objects such as Integer, Double, Boolean, etc. and to set an argument to be null method setting null argument must be called). Adapters doesn't implement all the methods from `ResultAdapter` and `PreparedStatement` interfaces, but only these which are used. Other important class in this package is `DbCredentialsManager` used to generate RSA keys and encrypt or decrypt properties files.

MySQL implementation in package `gssim.db.mysql` doesn't provide the implementation of all the methods, but only these which are necessary. If there will be demand on certain functions that are not supported now they can be easily implemented basing on this document and JavaDOC.

Usage of the API will be presented by example in listing 3.

```
1 package gssim.db.mysql;
2
3 import gssim.db.dao.data.Configuration;
4 import gssim.db.dao.data.Experiment;
5 import gssim.db.dao.data.Resource;
6 import gssim.db.dao.data.User;
7 import gssim.db.find.Finder;
8
9 import java.io.IOException;
10 import java.sql.Connection;
11 import java.sql.SQLException;
12 import java.util.ArrayList;
13 import java.util.List;
14
```

```

15 public class TestClass {
16
17     public static void main(String[] args) throws SQLException, IOException,
18         InstantiationException, IllegalAccessException,
19         ClassNotFoundException {
20
21         MySQLDAOFactory factory = new MySQLDAOFactory();
22         Connection connection = null;
23         try {
24
25             // COMMON FUNCTIONS FOR CRUD OPERATIONS
26             connection = factory.createDBConnection(
27                 "localhost:3306/gssim-results", "root", null);
28
29             ConfigurationDAO configurationDAO = new ConfigurationDAO(connection);
30             UserDAO userDAO = new UserDAO(connection);
31             ExperimentDAO experimentDAO = new ExperimentDAO(connection);
32             ResourceDAO resourceDAO = new ResourceDAO(connection);
33
34             // DATA CREATION (INSERTION)
35             Configuration config = new Configuration();
36             configurationDAO.insert(config, false);
37
38             User user = new User();
39             user.setDN("user-dn");
40             userDAO.insert(user, false);
41
42             Experiment experiment = new Experiment();
43             experiment.setConfigId(config.getId());
44             experiment.setUserId(user.getId());
45             experiment.setStatus("FINISHED");
46             experimentDAO.insert(experiment, false);
47
48             List<Resource> resources = new ArrayList<Resource>();
49             Resource tempRes = new Resource();
50             tempRes.setAllocationLoad(0.5);
51             tempRes.setReservationLoad(0.5);
52             tempRes.setExpId(experiment.getPK());
53             tempRes.setName("res-1");
54             tempRes.setType("to-find");
55             resources.add(tempRes);
56             // adding other resources
57             tempRes = new Resource();
58             tempRes.setAllocationLoad(0.5);
59             tempRes.setReservationLoad(0.5);
60             tempRes.setExpId(experiment.getPK());
61             tempRes.setName("res-n");
62             tempRes.setType("other");
63             resources.add(tempRes);
64
65             connection.commit();
66
67             // MULTIPLE TUPLES DATA RETRIEVAL (SEARCHING)
68             Finder finder = Finder.getResourceFinder(null, null, "to-find");
69             resources = resourceDAO.find(finder);
70             for (Resource resource : resources)
71                 System.out.println(resource);
72
73             // SINGLE TUPLE DATA RETRIEVAL (WITH UNIQUE ID)
74             int idToSelect = resources.get(0).getPK();
75             Resource resource = new Resource();
76             resource.setPK(idToSelect);
77             resource = resourceDAO.select(resource);
78             System.out.println(resource);
79
80             // DATA UPDATE
81             resource.setName("res-1-changed");
82             resourceDAO.update(resource, true);
83
84             // DATA REMOVAL
85             resourceDAO.delete(resource, true);
86
87         } catch (SQLException e) {
88             e.printStackTrace();
89             // exception handling

```

```

90     } finally {
91         if (connection != null) {
92             connection.close();
93         }
94     }
95 }
96 }
97 }

```

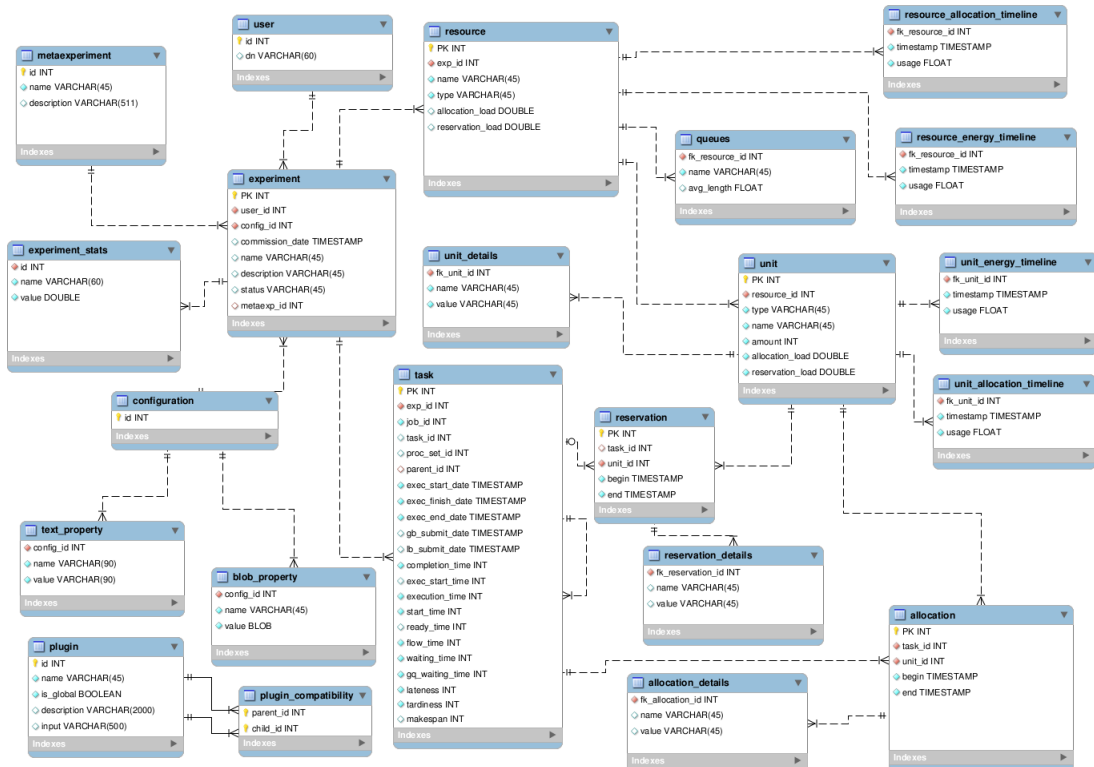
Listing 3: Usage of database API

Classes "Configuration", "User", "Experiment" and "Resource" represent entities "task", "user", "experiment" and "resource". As you can see in order to connect to the database you must use Factory class. Using DAO objects for each class allows you performing simple (CRUD – create, retrieve, update, delete) database operations. Worth to mention is the fact that each operation, apart the object of the specific class, uses also boolean value which determines whether to commit operation or not. Commit affects also all the previous operations which were not committed yet. Another important thing is, that setting the values which are primary keys generated from the sequence do not affect insert operation. Field with primary key is set during insert operation and if it was set before it will be changed. More complicated is find operation. For this a special class – Finder – is implemented. It has no public constructor. In order to retrieve object of this class you must use one of its static methods (for each entity there is different method). Arguments of the methods are finding criteria (their names are obvious, so you can easily know which argument stands for which criteria). Important fact is, that if criteria is null it is treated as irrelevant (queries are constructed in a specific way). To sum up, Finder class makes finding very easy, as it is presented in line 69. The result of query will be all the resources which type contain string "to-find". Other parameters are null, so they are not taken into account.

3 Database schema

Database schema is presented on figure 1. Up-to-date scripts connected with database (schema and example data) can be found in the following directory: \$GSSIM-path/trunk/db/example/

Figure 1: Database schema



4 Tutorials

4.1 Usage of MySQLWorkbench

On the following pictures (figure 2 and 3) will be presented how to add new class to database schema (basing on metaexperiment example). Database schema is designed in MySQLWorkbench and is saved in file: gssim_db_model.mwb, it can be exported to sql file by pressing ctrl + shift + g.

Figure 2: To place new relation user clicks the button pointed by cursor

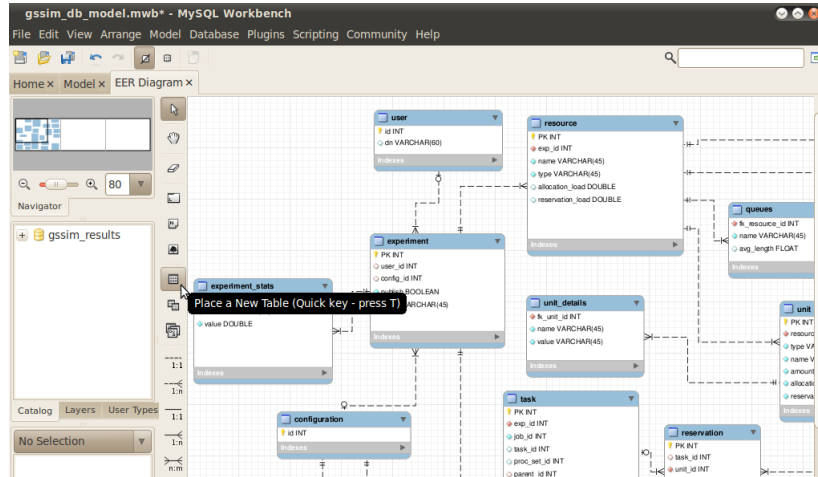
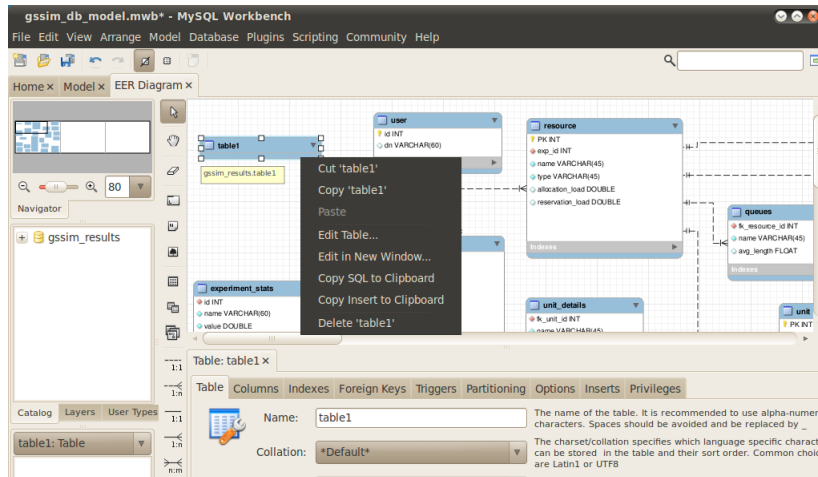


Figure 3: To edit new relation user clicks on it with right mouse button and then Edit table... with left button. It will make window which is seen on the bottom of the figure appear. In this window user can rename the table, manage the columns, foreign keys and so on



4.2 Adding classes to handle new entity

Let's take a look what classes must be added to handle "metaexperiment" entity:

- gssim.db.dao.data.Metaexperiment

```
1 package gssim.db.dao.data;  
2  
3 public class Metaexperiment {  
4  
5     protected Integer id;  
6     protected String name;  
7     protected String description;  
8  
9     // getters and setters
```

```

10
11     @Override
12     public int hashCode() {
13         // standard function
14     }
15
16     @Override
17     public boolean equals(Object obj) {
18         // standard function
19     }
20
21     @Override
22     public String toString() {
23         // standard function
24     }
25 }

```

Listing 4: Metaexperiment class

It's a class which represents an object connected with the entity

- gssim.db.dao.MetaexperimentDAO

```

1 package gssim.db.dao;
2
3 import gssim.db.dao.data.Metaexperiment;
4
5 public interface MetaexperimentDAO extends DAOInterface<Metaexperiment>{
6
7 }

```

Listing 5: MetaexperimentDAO class (just wrapper) from package gssim.db.dao

It's a class which wraps generic DAOInterface with entity name and does nothing more

- gssim.db.mysql.MetaexperimentDAO

```

1 package gssim.db.mysql;
2
3 // not important code
4
5 public class MetaexperimentDAO extends AbstractDAO implements
6     gssim.db.dao.MetaexperimentDAO {
7
8     protected MetaexperimentDAO(Connection conn) throws SQLException {
9         super(conn);
10        log = LogFactory.getLog(MetaexperimentDAO.class);
11        setRelationName("metaexperiment");
12    }
13
14    @Override
15    protected PreparedStatementAdapter createDeleteStmt() throws SQLException {
16        // similar to createInsertStmt()
17    }
18
19    @Override
20    protected PreparedStatementAdapter createFindStmt() throws SQLException {
21        // similar to createInsertStmt()
22    }
23
24    @Override
25    protected PreparedStatementAdapter createInsertStmt() throws SQLException {
26        String sql = "INSERT INTO metaexperiment(name, description) VALUES(?, ?)";
27        PreparedStatementAdapter stmt = new PreparedStatementAdapter(this
28            .getConnection().prepareStatement(sql,
29                Statement.RETURN_GENERATED_KEYS));
30
31        return stmt;
32    }

```



```

33
34 @Override
35 protected PreparedStatementAdapter createSelectStmt() throws SQLException {
36     // similar to createInsertStmt()
37 }
38
39 @Override
40 protected PreparedStatementAdapter createUpdateStmt() throws SQLException {
41     // similar to createInsertStmt()
42 }
43
44 @Override
45 public boolean delete(Metaexperiment obj, boolean commit) throws SQLException {
46     // similar to insert
47 }
48
49 @Override
50 public void fillObjectFromResultSet(Metaexperiment obj, ResultSet resultSet)
51     throws SQLException {
52     // function which converts JDBC data to entity object
53 }
54
55 @Override
56 public int fillStatementWithObject(PreparedStatement pstmt, Metaexperiment obj,
57     int offset) throws SQLException {
58     // function which converts entity object to JDBC data
59 }
60
61 @Override
62 public List<Metaexperiment> find(Finder finder) throws SQLException {
63
64     PreparedStatementAdapter stmt = getFindStmt();
65     List<Metaexperiment> list = new ArrayList<Metaexperiment>();
66
67     stmt.setString(1, finder.getString("name"));
68     stmt.setString(2, finder.getString("name"));
69
70     ResultSetAdapter result = new ResultSetAdapter(stmt.executeQuery());
71
72     while (result.next()) {
73         Metaexperiment obj = new Metaexperiment();
74         fillObjectFromResultSet(obj, result);
75         list.add(obj);
76     }
77
78     return list;
79 }
80
81
82 @Override
83 public int insert(Metaexperiment obj, boolean commit) throws SQLException {
84
85     PreparedStatementAdapter stmt = getInsertStmt();
86
87     fillStatementWithObject(stmt, obj, 0);
88
89     int result = stmt.executeUpdate();
90
91     if (result == 1) {
92         ResultSet gk = stmt.getGeneratedKeys();
93         Integer id = gk.next() ? gk.getInt(1) : -1;
94         obj.setId(id);
95
96         if (commit)
97             commit();
98
99         logSuccess("insert");
100
101     } else
102         logError("insert", result, obj);
103
104     return result;
105 }
106
107 public int[] insertBatch(List<Metaexperiment> objList, boolean commit)

```

```

108         throws SQLException {
109
110         PreparedStatementAdapter stmt = getInsertStmt();
111
112         for (int i = 0; i < objList.size(); i++) {
113             fillStatementWithObject(stmt, objList.get(i), 0);
114             stmt.addBatch();
115         }
116
117         int[] result = stmt.executeBatch();
118
119         stmt.clearBatch();
120
121         int j = 0;
122
123         while (j < result.length)
124             if (result[j] == 1)
125                 j++;
126
127         if (j == result.length) {
128
129             ResultSet gk = stmt.getGeneratedKeys();
130             for (int i = 0; i < objList.size(); i++) {
131                 Integer id = gk.next() ? gk.getInt(1) : -1;
132                 objList.get(i).setId(id);
133             }
134
135             logSuccess("batch insert", j);
136
137             if (commit)
138                 commit();
139
140         } else
141             logError("batch insert", j);
142
143         return result;
144     }
145
146     @Override
147     public Metaexperiment select(Metaexperiment obj) throws SQLException {
148         // similar to insert
149     }
150
151     @Override
152     public boolean update(Metaexperiment obj, boolean commit) throws SQLException {
153         // similar to insert
154     }
155 }

```

Listing 6: MetaexperimentDAO class from package gssim.db.mysql

Class which handles simple database operations. Method names are strictly connected to operations:

- create(...)Stmt – this is a group of methods which are responsible for SQL statements connected with basic database operations, worth to mention is fact that if retrieval of automatically generated primary key is desired then statement which is created should contain special argument as specified in line 29
 - fillObjectFromResultSet, fillStatementWithObject – these functions are used by other function to make transferring data between JDBC objects and entity objects easier
 - select, delete, update – these are very simple methods which use prepared statements to perform database operations
 - insert – it is very similar operation to mentioned right above however, it also retrieves an automatically generated primary key
 - batchInsert – it is very similar to insert, however it must handle retrieval of multiple primary keys
 - find – it is similar to select operation, however it uses Finder class object
- gssim.db.dao.find.Finder

```

1 package gssim.db.find;
2
3 // not important code
4
5 public class Finder {
6
7     Map<String, Object> map = new HashMap<String, Object>();
8
9     private Finder() {
10    };
11
12    // not important code
13
14    public static Finder getMetaexperimentFinder(String name) {
15
16        Finder finder = new Finder();
17
18        finder.setCriteria("name", name);
19
20        return finder;
21    }
22
23    // not important code
24
25    public Boolean getBoolean(String key) {
26
27        if (map.containsKey(key)) {
28            // not important code
29        }
30        return null;
31    }
32
33    public Integer getInteger(String key) {
34
35        if (map.containsKey(key)) {
36            // not important code
37        }
38        return null;
39    }
40
41    public Integer getMinInteger(String key) {
42
43        if (map.containsKey(key)) {
44            // not important code
45        }
46        return Integer.MIN_VALUE;
47    }
48
49    public Integer getMaxInteger(String key) {
50
51        if (map.containsKey(key)) {
52            // not important code;
53        }
54        return Integer.MAX_VALUE;
55    }
56
57    public void setCriteria(String key, Object value) {
58        map.put(key, value);
59    }
60
61 }

```

Listing 7: Finder class

This class stores methods which return Finder class storing information about search details for entities.

Any other classes can be added in the same way. Any modification of entity fields is connected with change of each class mentioned above (except DAOInterface wrapper).

4.3 Installation of the database module on remote server

Now it's just running the *ant grms.prepare* in command line (in the directory with build.xml file, which is now *trunk*) and copy to-grms directory to desired location. It is recommended to run after copying *ant grms.clean*. Currently gssim is installed on sftp://grass1.man.poznan.pl/opt/app/gssim, in this location classpath is included in special configuration files called modules (! to do: find name of these files).

If library *jsch.jar 0.1.42* or later is installed (or contained in classpath during task execution) it is possible to install GSSim on any sftp server which is in build.xml file under variable *sftp-grms*. It is performed by ant task *grms.install* which is executed with such command: *ant grms.install -Dusername="username" -Dpassword="password"*

Important: installation of daemon must be performed separately (it's just copying the daemon files to chosen directory). Please remember that user which started daemon must also have right to execute bash script with database analyzer as well as rights to open result files from grms.

5 Additional functionality – encryption manager

The class *DbCredentialsManager* in package *gssim.db.utils* contains UI which allows to generate private and public RSA keys as well as encryption and decryption of files using RSA algorithm.