# The JSL Database

The JSL has the ability to save statistical data from the simulation runs into a relational database. Any relational database can be utilized; however, the JSL directly supports the embedded Apache Derby database management system (DBMS) as well as the Postgres DBMS. The JSL database functionality is built upon the Java Object-Oriented Query (jooQ) application programming interface, which abstracts one level above directly using Java Database Connectivity (JDBC) calls. Be aware that jooQ is freely available for use only with open source databases. The JSL library provides utilities to create databases, connect to databases, import data from Excel spreadsheets, export data to Excel spreadsheets as well as perform queries.

## The JSL Database Structure

The JSL database consists of six tables that capture information and data concerning the execution of a simulation and resulting statistical quantities. Figure 1 presents the database diagram for the JSL\_DB schema.

* SIMULATION\_RUN – contains information about the simulation runs that are contained within the database. Such information as the name of the simulation, model, and experiment are captured. In addition, time stamps of the start and end of the experiment, the number of replications, the replication length, the length of the warm up period and options concerning stream control.
* MODEL\_ELEMENT contains information about the instances of ModelElement that were used within the execution of the simulation run. A model element has an identifier that is considered unique to the simulation run. That is, the simulation run ID and the model element ID are the primary key of this table. The name of the model element, its class type, the name and ID of its parent element are also held for each entity in MODEL\_ELEMENT. The parent/child relationship permits an understanding of the model element hierarchy that was present when the simulation executed.
* WITHIN\_REP\_STAT contains information about within replication statistical quantities associated with TimeWeighted and ResponseVariables from each replication of a set of replications of the simulation. The name, count, average, minimum, maximum, weighted sum, sum of weights, weighted sum of squares, last observed value, and last observed weight are all captured.
* WITHIN\_REP\_COUNTER\_STAT contains information about with replication observations associated with Counters used within the model. The name of the counter and the its value at the end of the replication are captured for each replication of a set of replications of the simulation.
* ACROSS\_REP\_STAT contains information about the across replication statistics associated with TimeWeighted, ResponseVariable, and Counters within the model. Statistical summary information across the replications is automatically stored.
* BATCH\_STAT contains information about the batch statistics associated with TimeWeighted, ResponseVariable, and Counters within the model. Statistical summary information across the batches is automatically stored.

In addition to the base tables, the JSL database contains views of its underlying constructs to facilitate simpler data extraction. Figure 2 presents the pre-defined views for the JSL database. The views, in essence, reduce the amount of information to the most likely used sets of data for the across replication, batch, and within replication captured statistical quantities. In addition, the PW\_DIFF\_WITHIN\_REP\_VIEW holds all pairwise differences for every response variable, time weighted variable, or counter from across all experiments within the database. This view reports (A – B) for every within replication ending average, where A is a simulation run that has higher simulation ID than B and they represent an individual performance measure. From this view, pairwise statistics can be computed across all replications.

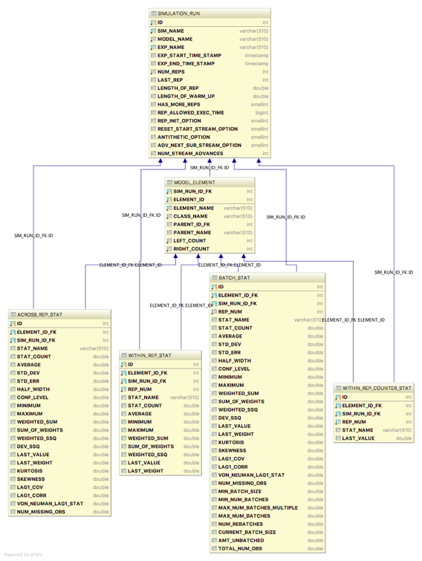


Figure 1: JSL Database Table Relational Diagram

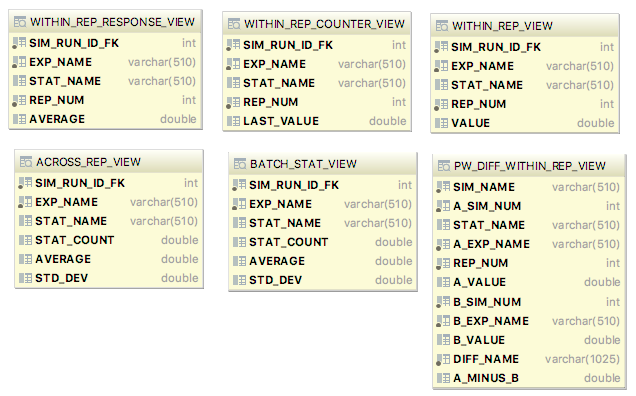


Figure 2: JSL Database Views

The information within the SIMULATION\_RUN, MODEL\_ELEMENT, WITHIN\_REP\_STAT, WITHIN\_REP\_COUNTER\_STAT tables are written to the database at the end of each replication. The ACROSS\_REP\_STAT and BATCH\_STAT tables are filled after the entire experiment is completed. Even though the ACROSS\_REP\_STAT table could be constructed directly from the data captured within the tables holding with replication data, this is not done. Instead, the across replication statistics are directly written from the simulation after all replications of an experiment are completed.

A JSLDatabase instance is constructed to hold the data from any JSL simulation. As such, a simulation execution can have many observers and thus could have any number of JSLDatabase instances that collect data from the execution. The most common case for multiple databases would be the use of an embedded database as well as a database that is stored on a database server.

# Creating and Using a Default JSL Database

It is easy to have a database associated with a simulation. Just indicate that a default database should be created when constructing an instance of the Simulation class by providing “true” for the create default database parameter of the Simulation class:

Simulation sim = **new** Simulation(**"Drive Through Pharmacy"**, **true**);

This causes an instance of the class, JSLDatabase, to be created and attached to the instance of the Simulation as an observer via an intermediate class called JSLDatabaseObserver. The running of the simulation then causes data from the simulation to be stored in an Apache Derby embedded database that is contained within the directory, jslOutput/db. The name of the database will be based on the name provided for the simulation. For the previous code snippet, a database called JSLDb\_DriveThroughPharmacy will be created. Any spaces in the name of the simulation are removed for the name of the database and JSLDb\_ is appended. This naming is the default behavior for the default database. The database can be accessed just like any database. For example, IntelliJ’s Datagrip tool can be used to connect to the database, execute queries, and export data.

One very important point to note is that constructing an instance of Simulation and providing “true” for the create default database option, **always** creates a new instance of the default database. This process involves deleting the previous default database and re-creating it without any data. If you do not want the previously created default database to be deleted and recreated, then you must take explicit steps to prevent this from happening. The most obvious steps include:

* Do not call the constructor of Simulation with “true” for the create default database option. This will cause no new default JSL database to be created. Thus, a previous database instance will not be deleted. Or,
* Change the name of the simulation so that a differently named default database will be constructed.
* Move, copy, or rename the previously created database using operating system commands.

Note that if you execute **any** simulation that has the **same name** as a previously executed simulation and you were using the default database option, then the previous database will be deleted and recreated. This might cause you to lose previous simulation results. This behavior is the default because generally, when you re-run your simulation you want the results of that run to be written into the database.

# Creating and Using JSL Databases

To better control the creation and use of an instance of the JSLDatabase, I suggest that you consider creating your own instance rather than relying on the default database. The main reason to do this is if you plan to add results from multiple simulation executions to the **same** database. A database represented by a JSLDatabase instance is just a database that has a JSL\_DB schema within it to hold data from a JSL simulation run. Provided that you can properly configure a JSL\_DB schema within a database, you can use **any** database as the backing store for JSL results. By default the JSL library facilitates the creation of embedded Apache Derby databases and Postgres databases. In addition, the JSL library facilitates the creation of JSLDatabase instances based on these two database management systems.

A number of methods are provided to create instances of a JSLDatabase. Obviously, the constructor of JSLDatabase can be used. The constructor has two parameters, an instance of an object that implements the DatabaseIfc interface and a Boolean parameter (clearDataOption) which controls whether or not all of the data within a possible JSL\_DB schema is removed when the JSLDatabase instance is created. The default behavior is to not remove previous data. If the supplied DatabaseIfc interface instance does not already have a JSL\_DB schema, then one is created. If it already has a JSL\_DB schema, then the clear data option controls what happens to any previously stored data. Any of the methods of the DatabaseFactory class can be used to create an instance of the DatabaseIfc interface. Because you are most likely interested in directly making a JSLDatabase instance, there are a number of static methods of the JSLDatabase class that are provided for common use cases. For example, the following code snippet illustrates how to make an instance of a JSLDatabase based on the embedded Derby database system. This will create a database named “MCB\_Db” within the jslOutput/db directory. If a database already exists with that name, then it will be deleted and a new database created.

JSLDatabase mcb\_db = JSLDatabase.*createEmbeddedDerbyJSLDatabase*(**"MCB\_Db"**);

If you want to connect to a previously created database, then use the methods of the DatabaseFactory class. For example, the following connects to an existing database found within the jslOutput/db directory and then supplies it to the JSLDatabase constructor. No previous data is lost via this process since we are only connecting to a database (not creating one).

DatabaseIfc database = DatabaseFactory.*getEmbeddedDerbyDatabase*(**"JSLDb\_DriveThroughPharmacy"**);  
*// use the database as the backing database for the new JSLDatabase instance*JSLDatabase jslDatabase = **new** JSLDatabase(database);

The previously illustrated code examples only **create** an instance of JSLDatabase. The instance is **not** connected to an instance of Simulation and thus simulation results will not be added to the database unless additional steps are taken to hook up the JSLDatabase instance with instances of the Simulation class prior to running experiments.

The approach to connect a JSLDatabase instance with a Simulation instance involves creating an instance of JSLDatabaseObserver to monitor the simulation’s execution. JSLDatabaseObserver has three required parameters in its constructor: 1) an instance of JSLDatabase, 2) and instance of Simulation, and 3) a boolean parameter (clearDataBeforeExperimentOption), which controls whether or not data from prior executions of the observed simulation will be cleared if they have the same simulation name and experiment name before each experiment (i.e. when the run() method is called on the simulation). The default value for the clear data before experiment option is true. Thus, data will be cleared from the database if the simulation name and experiment name are **the same**. If you do not want the data cleared, then set the option to false. However, if you then attempt to execute a simulation that has the same name and experiment name as one already stored in the database, an exception will be thrown. To prevent this exception change the name of the simulation or the experiment prior to running the simulation or decide to clear the data. The preferred method is to change the name of the experiment since this facilitates other analysis using JSL constructs.

Since creating and using a JSLDatabaseObserver is a common use case, the JSL library provides methods on the Simulation class to facilitate this:

* **public** JSLDatabaseObserver createJSLDatabaseObserver(String dbName)
* **public** JSLDatabaseObserver createJSLDatabaseObserver(JSLDatabase jslDatabase)
* **public** JSLDatabaseObserver createJSLDatabaseObserver(JSLDatabase jslDatabase, **boolean** clearDataBeforeExperiment)

The first method creates an embedded Derby database with the provided name in the jslOutput/db directory. The created JSLDatabaseObserver is returned and through that reference the underlying JSLDatabase can be accessed. The first of these three methods creates a new database. The latter two only creates a new JSLDatabaseObserver instance and uses the supplied JSLDatabase. Thus, data will be added to the database. The class, UsingJSLDbExamples within the ex.running package illustrates many of the use cases presented.

As an illustration consider running a simulation multiple times within the same program execution but with different parameters. The following code illustrates how this might be achieved. The first line creates a simulation with a default database. Then the simulation is set up and executed. Notice that the experiment name is set prior to running the simulation. Then, the service time parameter is changed and the simulation is executed again. Notice that the experiment name was changed before the second simulation run. Data from both executions are captured within the default database. This code can be re-executed because a new default database is created each the Simulation instance is constructed. This clears all previous data and then all subsequent runs are captured because the experiment name is changed. If the name had not been changed before the second simulation run, then when the second simulation run executes the data from the first run will be cleared and the second run data captured because the clear data before experiment flag is true. If the clear data before experiment flag is false and you attempt to execute this code an exception would be thrown because there would be an attempt to enter data into the database that has the same simulation and experiment name. This would force you to change the name of the experiment before executing the second experiment. Because this code does change the names of the experiments, the clear data before experiment flag setting is irrelevant because the experiments have different names.

*// make the simulation with a default database*Simulation sim = **new** Simulation(**"MultiRun"**, **true**);  
*// set the parameters of the experiment*sim.setNumberOfReplications(30);  
sim.setLengthOfReplication(20000.0);  
sim.setLengthOfWarmUp(5000.0);  
  
*// create the model element and attach it to the main model*DriverLicenseBureauWithQ driverLicenseBureauWithQ = **new** DriverLicenseBureauWithQ(sim.getModel());  
sim.setExperimentName(**"1stRun"**);  
*// tell the simulation to run*System.***out***.println(**"Simulation started."**);  
sim.run();  
System.***out***.println(**"Simulation completed."**);  
  
sim.setExperimentName(**"2ndRun"**);  
driverLicenseBureauWithQ.setServiceDistributionInitialRandomSource(**new** ExponentialRV(0.7));  
  
*// tell the simulation to run*System.***out***.println(**"Simulation started."**);  
sim.run();  
System.***out***.println(**"Simulation completed."**);  
  
*// get the default JSL database*Optional<JSLDatabase> db = sim.getDefaultJSLDatabase();  
  
**if** (db.isPresent()) {  
 System.***out***.println(**"Printing across replication records"**);  
 db.get().getAcrossRepStatRecords().format(System.***out***);  
 System.***out***.println();  
}

Once you have a database that contains the schema to hold JSL based data, you can continue to write results to that database as much as you want. If your database is on a server, then you can easily collect data from different simulation executions that occur on different computers by referencing the database on the server. Therefore, if you are running multiple simulation runs in parallel on different computers or in the “cloud”, you should be able to capture the data from the simulation runs into one database.

# Querying the JSL Database

The JSL database is a database and thus it can be queried from within Java or from other programs. If you have an instance of the JSLDatabase as in:

DatabaseIfc database = DatabaseFactory.*getEmbeddedDerbyDatabase*(**"JSLDb\_DriveThroughPharmacy"**);  
*// use the database as the backing database for the new JSLDatabase instance*JSLDatabase jslDatabase = **new** JSLDatabase(database);

You can extract information about the simulation run using the methods of the JSLDatabase class. Since the underlying data is stored in a relational database, SQL queries can be used on the database. The discussion of writing and executing queries from within Java is beyond the scope of this discussion. To facilitate queries within Java, the JSL leverages the open source jooQ library. A few methods to be aware of include:

* writeAllTablesAsCSV() – writes all the tables to separate CSV files
* writeDbToExcelWorkbook() – writes all the tables and views to a single Excel workbook
* getWithinRepViewRecords() – returns a jooQ Result containing all the within replication statistical records
* getWithinRepViewRecordsAsTablesawTable() – returns a Tablesaw table representation of the within replication statistical data
* getAcrossRepViewRecordsAsTablesawTable() - returns a Tablesaw table representation of the across replication statistical data
* getAcrossRepViewRecords() returns a jooQ Result containing all the across replication statistical records
* getMultipleComparisonAnalyserFor(set of experiment name, response name) – returns an instance of the MultipleComparisonAnalyzer class in order to perform a multiple comparison analysis of a set of experiments on a specific response name.

The JSL database can be accessed via R or other software programs and additional analysis performed on JSL simulation data.

# Appendix

The functionality of the JSL\_DB depends upon how ResponseVariable, TimeWeighted, and Counter instances are named within a JSL model. A JSL model is organized into a tree of ModelElement instances with the instance of the Model at the top of the tree. The Model instance for the simulation model contains instances of ModelElement, which are referred to as children of the parent model element. Each model element instance can have zero or more children, and those children can have children, etc. Each ModelElement instance must have a unique integer ID and a unique name. The unique integer ID is automatically provided when a ModelElement instance is created. The user can supply a name for a ModelElement instance when creating the instance. The name must be unique within the simulation model.

A recommended practice to ensure that model element names are unique is to use the name of the parent model element as part of the name. If the parent name is unique, then all children names will be unique relative to any other model elements. For example, in the following code getName() references the name of the current model element (an instance of QueueingSystemWithQ), which is serving as the parent for the children model element declared within the constructor.

**public** QueueingSystemWithQ(ModelElement parent, **int** numServers, RVariableIfc ad, RVariableIfc sd) {  
 **super**(parent);  
  
 **myWaitingQ** = **new** Queue(**this**, getName() + **"\_Q"**);  
 **myNumBusy** = **new** TimeWeighted(**this**, 0.0, getName() + **"\_NumBusy"**);  
 **myNS** = **new** TimeWeighted(**this**, 0.0, getName() + **"\_NS"**);  
 **mySysTime** = **new** ResponseVariable(**this**, getName() + **"\_System Time"**);

The name supplied to the TimeWeighted and ResponseVariable constructors will cause the underlying statistic to have the same name. The statistic’s name cannot be changed once it is set. The statistic name is important for referencing statistical data within the JSL database. One complicating factor involves using the JSL database to analyze the results from multiple simulation models. In order to more readily compare the results of the same performance measure between two different simulation models, the user should try to ensure that the names of the performance measures are the same. If the above recommended naming practice is used, the names of the statistics may depend on the order in which the model element instances are created and added to the model element hierarchy. If the model structure never changes between different simulation models then this will not present an issue; however, if the structure of the model changes between two different simulation models (which can often be the case), the statistic names may be affected. If this issue causes problems, you can always name the desired output responses or counters exactly what you want it to be and use the same name in other simulation models.

Since the model element ID is assigned automatically based on the number of model elements created within the model, the model element numbers between two instances of the same simulation model will most likely be different. Thus, there is no guarantee that the IDs will be the same and using the model element ID as part of queries on the JSL database will have to take this into account. You can assume that the name of the underlying statistic is the same as its associated model element and since it is user definable, it is better suited for queries based on the JSL database.