Project Documentation

A JDBC Driver Supporting Data Integration and Evolution

Jian Jia

University of Iowa, Iowa City, IA
jjia@cs.uiowa.edu

Goals

This project will produce a Unity JDBC Driver that is compliant with the JDBC 3.0 API and offers the ability to translate application requests into several database system specific SQL query requests. The driver will access each of the distributed database sources concurrently through the database specific JDBC Driver provided by the vendor. Query results from each database will be integrated and then sent back to the application.

Project Details

The project is proposed to be finished in four phases.

Phase I

This phase involves building a pass-through JDBC driver that can be used to connect to a data source hosted by MySQL, through its JDBC driver, an open source type 4 driver.

Both MySQL and its JDBC driver were downloaded from internet and installed under WindowsXP. Environment for java program development were also set up under same operating system. The environment includes a compiler j2sdk1.4.0-01 and a package, Jbuilder for code editing.

Three basic JDBC classes must be created to reach goal of phase I:

1. UnityDriver, which implemented the standard java interface *java.sql.Driver* is used to create a driver object. The UnityDriver is able to scan all available JDBC drivers in the driver list text file, and creates connections to databases through those drivers. The URL accepted by UnityDriver is set to any URL by default, since it is used in the local machine.
2. UnityConnection, which implemented the standard java interface *java.sql.Connection* is used to create connection object to other databases.
3. UnityStatement, which implemented the standard java interface *java.sql.Statement* is used to create statement to execute query.

Another auxiliary object, called DataSources is created to store information of each database, including its JDBC driver, URL, connection and statement. That information is needed to distinguish different requests for different databases.

A simple program, named mytest.java was developed to test function of those JDBC API.

Phase II

It was originally proposed to embed a Database Engine into the pass-through driver, so result sets can be stored appropriately in the Database Engine instead of in memory structures. However, query result is stored in the ResultSet object, and there is not directly public access to all data fields. To copy the whole result into a Database Engine table, the method getXXXX (Column index), where XXXX represents data type, must be called every time to acquire a value. This scan through all fields without doing any integration pulls the performance down significantly. Therefore, a new decision is made to integrate results using JDBC ResultSets received without assistance of Database Engine.

Phase III

The Query Translator is ported in phase III.

User queries sent to the Unity driver are now expressed in a semantic query language instead of standard SQL (here version SQL-92 is applied, for it is the basic SQL form that all database packages must support). Features of semantic query are:

i. Semantic query refers a field by its semantic name.

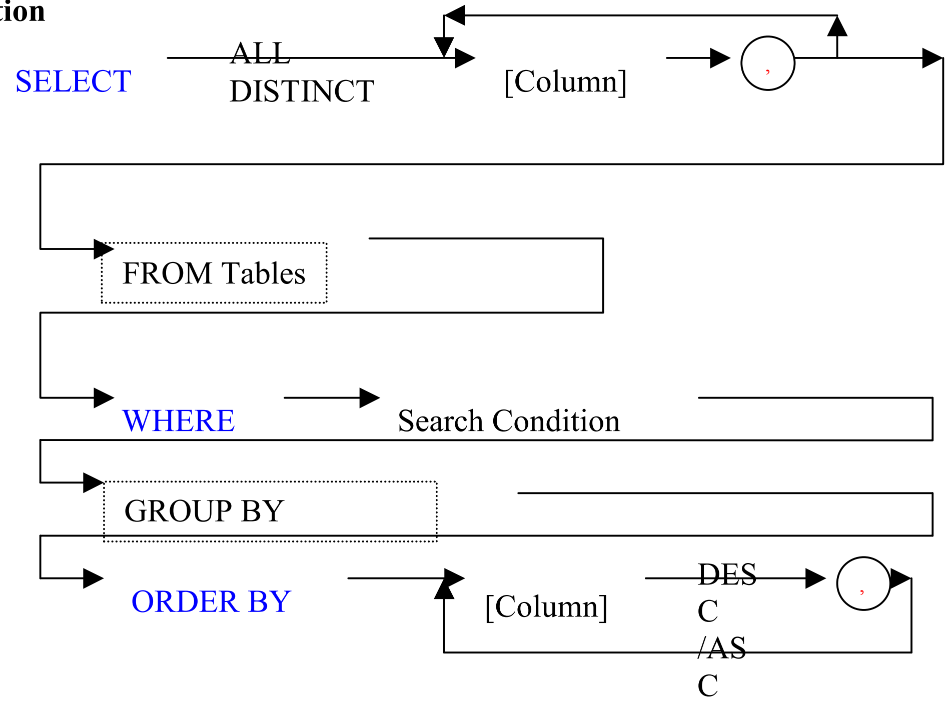
ii. Usually explicit relation specifications such as from table, join and union are not included in the query. Instead they are identified by database information stored in X-Spec while the query is parsed. The only exemption is when such identification is impossible, that is when a join is on a relation itself. Then from table must be given.

iii. Nested sub-query is not allowed in semantic query. This is one approach to the goal of semantic query to make it simpler than SQL, and thus even a dummy user can easily understand and apply it.

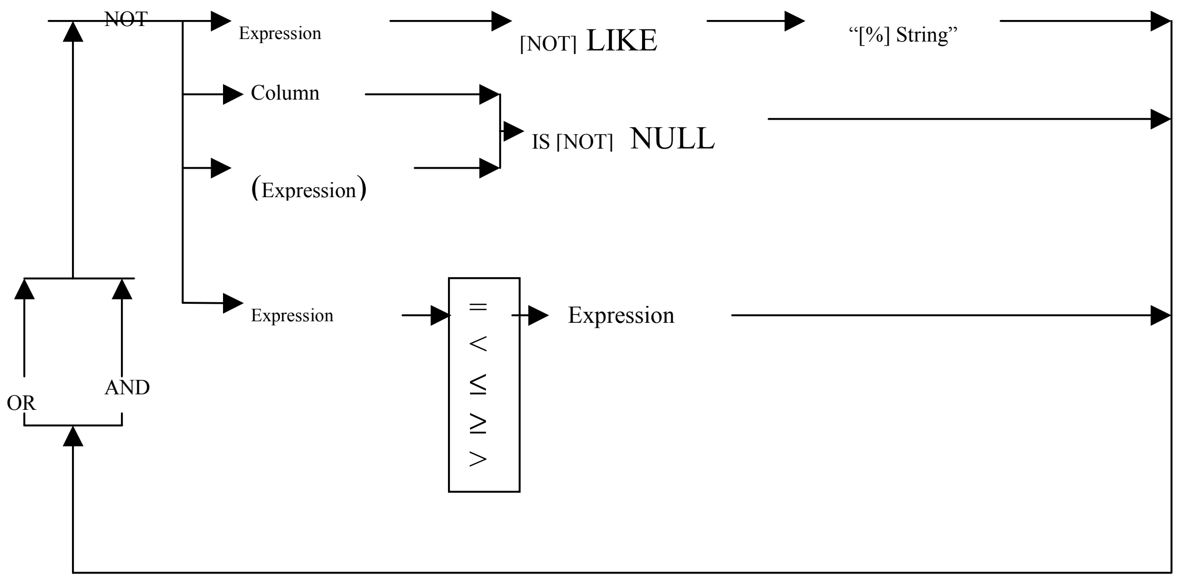
iv. Updates are not supported since they are left as future work beyond this project.

v. Semantic query grammar is thus slightly different from that of SQL-92 due to above features. Following two figures illustrates grammar of selection and search condition: (Expression grammar is not present here since it is the same as that of SQL-92)

Selection



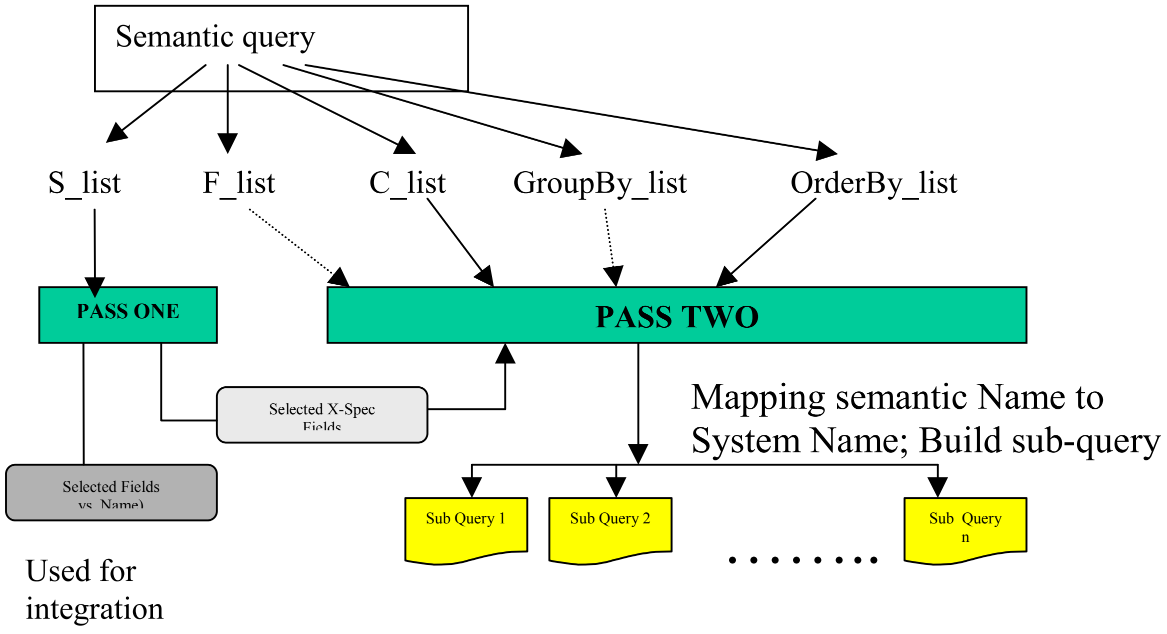
Search Condition



As for the semantic name of a field, it must satisfy the rule that all Fields/Tables that have the same semantic meaning should have same semantic name.

The Query Translator translates the semantic query into SQL, which is executed using the MySQL JDBC driver. Below is the structure of the translator:

Query Translator



For simplification and extendibility of query parsing, the semantic query is broken into small pieces as S_list, F_list, C_list, GroupBy_list and Orderby_list corresponding to statements after keywords: SELECTION, FROM, WHERE, GROUP BY and ORDER BY respectively.

S_list is sent to PassOne, a class to parse selected fields. Legal semantic name must be mapped into its system name, once it is found exist in the database structure. Mapped system names for S_list are stored in matrix (two dimensional array with row corresponding to data source and column corresponding to the semantic name), which is used for results integration.

All lists are sent to PassTwo to identify from-tables, joins and where condition besides system name mapping. Sub-query for each database is generated based on the information from above. The translation follows two rules:

i. Only those semantic fields that are in the data source can be substituted by their system names, and added to corresponding sub-query selection list

ii. An expression can only be added to sub-query condition list only when all semantic arguments are in the data source.

The class UnityTokenizer is used to create token for parsing a string. Mapping from semantic name to system name requires information of database structure. Such information is received by parsing X-Spec Document using X-Spec class and X-Spec Parser.

Phase IV

The Integration Module is ported in phase IV. This final phase involves handling multiple diverse databases and registering their drivers. The Query Translator converts semantic queries from the user to SQL queries for each database (as required). Results from each JDBC driver associated with each database are returned back as ResultSet objects. Final result is integrated on those objects.

Join and Union are two basic methods for integration embedded in UnityStatement. Join applies the algorithm of Sort-Join, i.e. always output the record with smallest key. And thus only when global primary key presented, can results be joined together. Multi-value Field is created during Join when data are inconsistent on data type, size or value across databases. Union is to append all results together when global key couldn't be found in those results. Algorithm of Union is thus very simple and not present here.

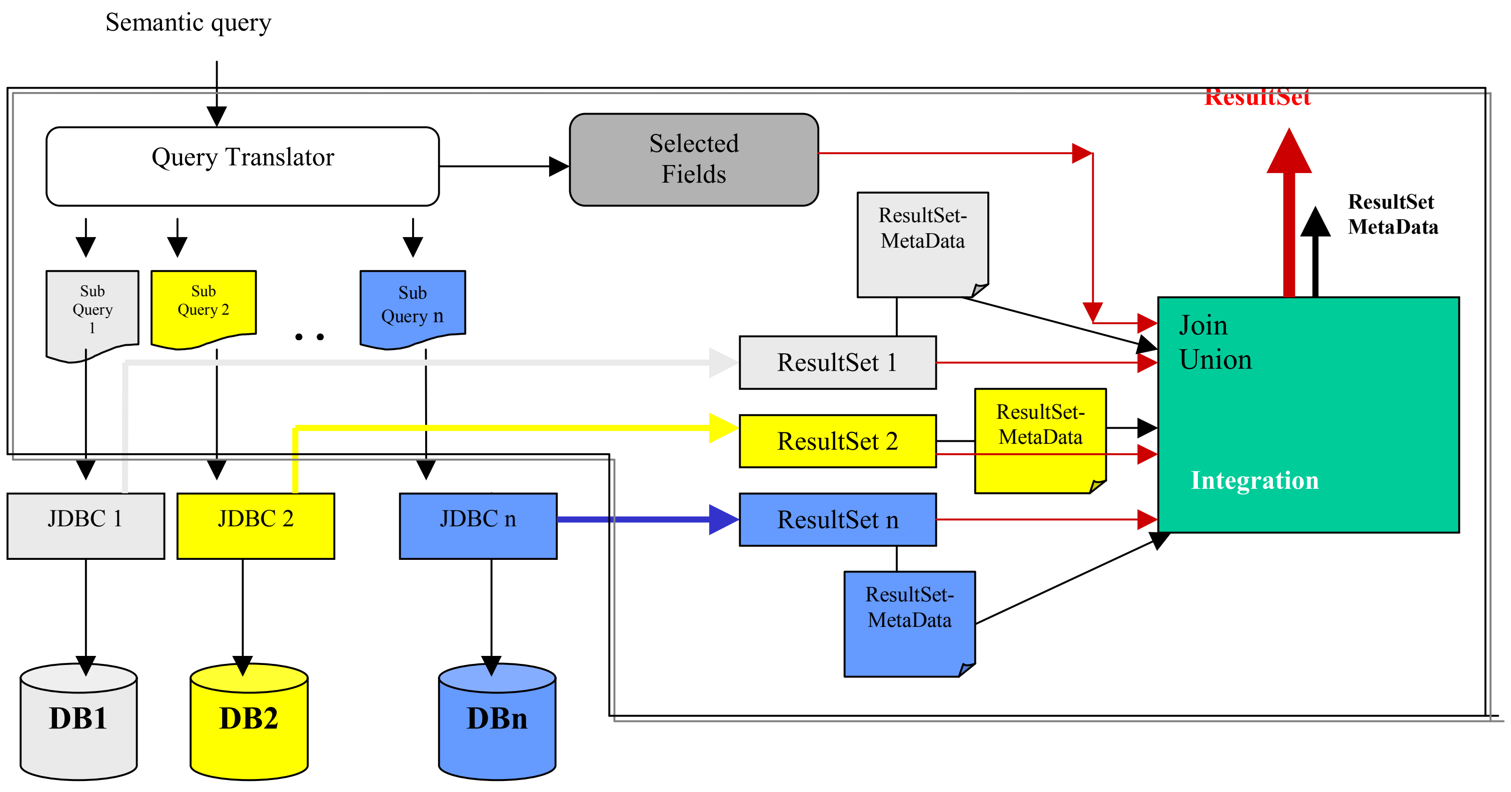
Final result is represented by UnityResultSet class, which implemented *java.sql.ResultSet* interface. To get information about the types and properties of the columns in a UnityResultSet object, the UnityResultSetMetaData class should be implemented. Data in UnityResultSet are stored as a variable array of Field objects.

The following two figures illustrate Join algorithm and the whole new picture of unity JDBC API.

```
        // Find all matched Columns from all Resultsets
         byte[][] keyvalues = new byte[result_num][];
         boolean match =true;

         for (int i=0; i<result_num; i++)// # of resultsets
       {
           if(res[i].next())
                 keyvalues[i] = res[i].getBytes(key[i]);
            else  keyvalues[i] = null;
         }

         while(!is_Empty(keyvalues))
         {
       // find the smallest key values and compare if other fields are also consistent
         int smallest = smallest_key_pos(keyvalues);
         // every field is of type vector since there could be multiple values.
         Vector [] f_values = new Vector[result_num];
         for (int k=0; k < rsf_length; k++)
         {
            if (selectfds[smallest][k] != null)
            {
                        f_values[k].addElement(res[smallest].getBytes(selectfds[smallest][k]));
             }
          }
         for (int i=smallest+1; i < result_num; i++)
         {
                if (keyvalues[i] == keyvalues[smallest])   // Key matches
              {
                         // check if other fields also matches
                 for (int k=0; k < rsf_length; k++)
                 {   byte [] value_s = null;
                     byte [] value_i = null;
                 if (selectfds[smallest][k] != null)
                                         value_s = res[smallest].getBytes(selectfds[smallest][k]);
                 if (selectfds[i][k] != null)      value_i = res[i].getBytes(selectfds[i][k]);
                 if (value_s != value_i &&  selectfds[i][k] != null)                    f_values[k].addElement(value_i);
                   if (f_values[k].size() > 1) RSFields[k].set_Multivalue();
                          }
             if (res[i].next()) keyvalues[i] = res[i].getBytes(key[i]);
             else keyvalues[i] = null;
             }
        }
       rows.addElement(f_values); // get f_values for all fields of one row
       if (res[smallest].next()) keyvalues[smallest] = res[smallest].getBytes(key[smallest]);
       else keyvalues[smallest] = null;
} // end of while iterator

// all rows are created;
//return (ResultSet) new UnityResultSet(RSFields, rows);
} catch (SQLException sqlEx){}
return (ResultSet) new UnityResultSet(RSFields, rows);
```

Sort-Join Algorithm

Inside JDBC Driver



Further Work on Unity JDBC Driver

The integration algorithms should be extended to allow compuations across databases and GROUP BY operation. PassTwo needs be modified as the consequence of integration extension. A full testing should be the next step to verify functions of Unity JDBC driver, after several more sample databases are created and their X-Spec documentations generated.

Summary

In order to use the integration capabilities of Unity in a Java development environment, a Unity JDBC driver prototype is developed in this project. This JDBC driver prototype offers the ability to access multiple database systems, and is capable of translating semantic user queries into SQL queries for each database system. Thus the complexities of accessing and integrating information from multiple databases are hidden from client applications. The project uses MySQL for testing. The flexibility of JDBC will allow the Unity JDBC driver to access other databases such as Sybase, Microsoft SQL server, Oracle and Informix on different platforms.