DBSchemaScribe—Program Manual

Shengye Wan

I. System Architecture

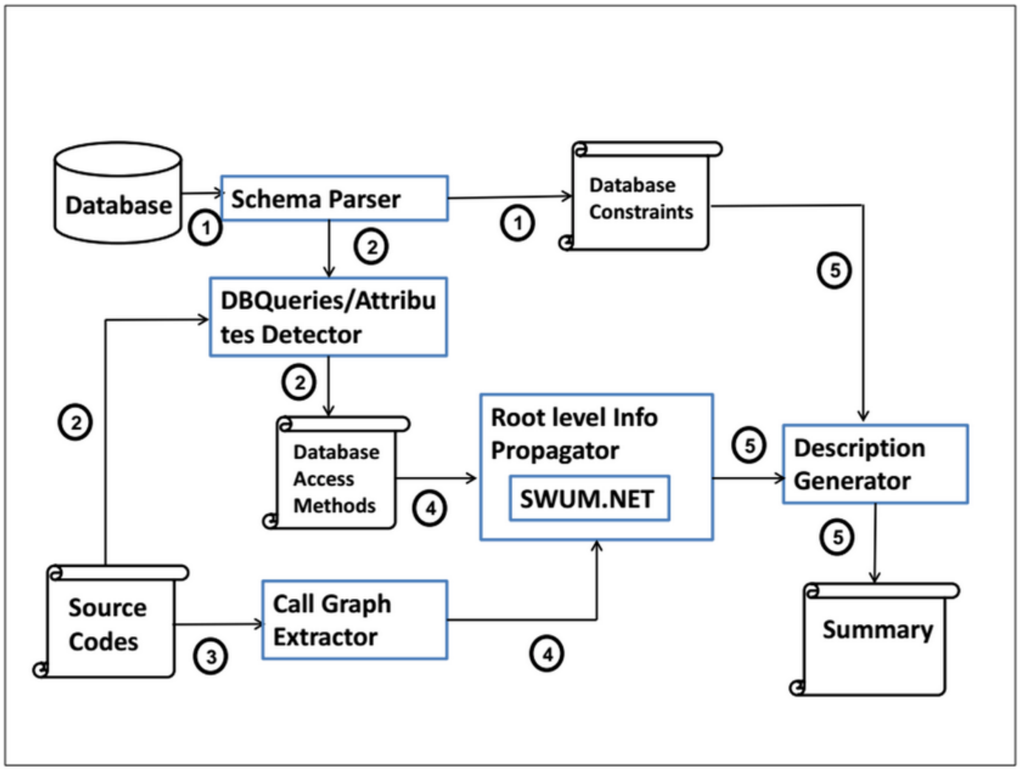


Figure 1: DBSchemaScribe System Architecture

II. Summary of the program

1. The high level working flow

In the program, at first we would connect the program with MySQL database. Then we get the detailed information of our target schema. The next step is going through all the methods in the project that we want to analyze and finding out all the methods have SQL invocation. After this step, we would map the methods with tables and columns according to the SQL statements we found so we would know for each table or column which methods access them and how these methods access them. The final step is generating the report for the target project.

2. Key points

For each table or column, I define the related methods into three types in my code: direct, follow and final. "Direct methods" means the method contains SQL local invocations. "Follow methods" contain the methods that do not have SQL local invocations but they are calling the "direct methods". Another attribute is that "follow methods" could not be the methods in the top-level (because our researching assumption is that the top-level methods are more useful to show how a table or column is operated by user). And the "final methods" are the top-level methods of the call graphs of "direct methods". This conception would appear many times in my code.

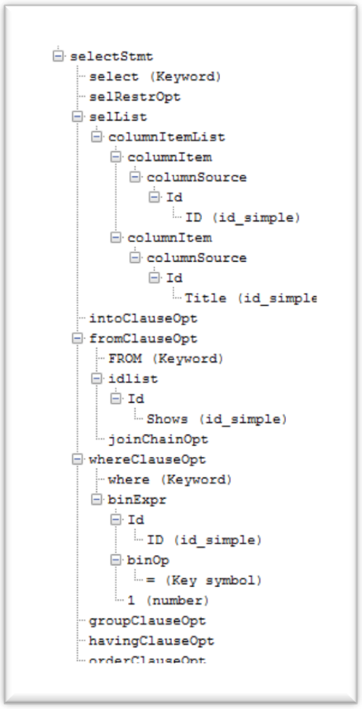
III. Details of Program's Components

1. Schema Parser

This component is developed based on namespace "**MySql.Data.MySqlClient**" which contains INFORMATION_SCHEMA TABLE of each schema in MySql database. These tables provide access to database metadata, information about the MySQL server such as the name of a database or table, the data type of a column, or access privileges.

2. DB Queries / Attributes Detector

This component is developed based on the only one open source solution I found. It's called "**Irony Kit**" (refer: https://gridwizard.wordpress.com/2014/11/08/looking-for-a-SQL-parser-for-c-dotnet/). Here is an example:
For SQL query: **SELECT ID, Title FROM Shows WHERE ID=1**, It could generate a parsing tree looks like figure 2.



```
selectStmt
  select (Keyword)
  selRestrOpt
  selList
    columnItemList
      columnItem
        columnSource
          Id
            ID (id_simple)
      columnItem
        columnSource
          Id
            Title (id_simple
  intoClauseOpt
  fromClauseOpt
    FROM (Keyword)
    idlist
      Id
        Shows (id_simple)
    joinChainOpt
  whereClauseOpt
    where (Keyword)
    binExpr
      Id
        ID (id_simple)
      binOp
        = (Key symbol)
      1 (number)
  groupClauseOpt
  havingClauseOpt
  orderClauseOpt
```

Figure 2: SQL Parsing Tree

3. Call Graph Extractor

This component is based on the namespace "**ABB.SrcML**". The classes of namespace could help us get detailed information of each methods in project source code. Then we could build the Call Graph based on the methods relationship of SrcML and find out what's the top-level methods of each method.
*Acknowledgement: The function about generating call graphs of methods is written by Boyang.

4. Root Level Info Propagator

This component is developed based on the namespace "**ABB.Swum**". The classes of this namespace could automatically generate natural language descriptions for each method.

5. Description Generator

This component is developed based on the the namespace "**Antlr3.ST**". Basically we could pass values from our application the the script of a webpage and automatically generate a page as report with defined data.

IV. Details of every file in the program

1. Class name: Program, Location: SrcTest/.

The function of this class:
This class is the start of our program. In this class, we would call different methods to get database information, find out all SQL-related methods, map these methods with specific table or column in database schema and generate the report for each test program.

The variables of this class:
public static readonly string ProjLoc. This string saves the location of our program. In the rest part we would call this location as "root".
public static readonly string testProj. This string represents which project we want to test. For example, if we want to analyze project "RiskIt", then this variable should be "RiskIt".
public static readonly string databaseSchemaName. This string is the name of test project's database. For example, when we run the scripts of RiskIt, it would generate a schema called "riskit". Then when we are analyzing RiskIt, we should set this variable as "riskit".
public static string myConnectionString. This string is written according to the grammar of MySQL. In this string, we need to configure the database settings like server address, user ID and password.
public static string SrcmlLoc. This string represents where we save the tool "SrcML". In my GitHub version, I save it in the folder "root/tool".
public static string testLoc. This string represents where we save the test project which is "testProj" variable above.

In my GitHub version, I save all projects in the folder "root/testProgram".
public static string reportLoc. This string represents where we want to save the report of the test project. In my GitHub version, I save all reports in the folder "root/reports".

The methods of this class:
void Main(). In this method we generate 4 classes: 1. dataSchemer, 2. ExtractMethodSQL, 3. dbMethodMapper and 4. infoCollector. The usages of these four classes could refer to their descriptions below.

2. Class name: dataSchemer, Location: SrcTest/DatabaseInfo.
The function of this class:
This class could connect program with MySQL to:
1: Get target database schema's information;
2: Save information into table classes and method classes we defined.

The variables of this class:
public MySqlConnection conn. This variable is response for the class to connect with database. This conn would specify which schema in database is we need.
public List<string> tablesNames. This variable would save all the tables' name of our target schema.
public List<dbTable> tablesInfo. dbTable is a class I defined. This class save all the information we need about one table like what columns one table contain. With this List, we could get any information of one table by search the table's name in list.

The methods of this class:
public void buildTables(). This method would build new dbTable classes for each table and new dbColumn classes for each column. After executing this method, the class dataSchemer would contain all the information about tables and columns.
public List<string> getTableName(). This method would return all tables' names in the target schema.
public List<string> getColumnName(string tableName). This method would return all columns' names in one table.
public string GetOneColumnInfo(string tableName, string columnName, string desireAttribute). This method could return one attribute of one column. "desireAttribute" means which attribute we want to know. The grammar refers to the link https://dev.mySQL.com/doc/refman/5.7/en/columns-table.html.
public string GetOneTableInfo(string tableName, string desireAttribute). This method could return one attribute of one table. The grammar of "desireAttribute" refers to the link https://dev.mySQL.com/doc/refman/5.7/en/tables-table.html.

3. Class name: dbTable, Location: SrcTest/DatabaseInfo.
The function of this class:
This class is defined to save all the data we need of each table.

The variables of this class:
public List<dbColumn> columns. This list contains all the columns belong to the table.
public List<desMethod> directMethods. This list contains all the methods that directly access this table.
public List<desMethod> followMehtods. This list contains all the methods that might access this table by call the methods in "directMethods". Furthermore, methods in list are not the methods that in the top-level of a call graph.
public List<desMethod> finalMethods. This list contains all the methods that might access this table and methods in this list have to be the top-level methods.
public List<dbMethodSql> relationships. This list is used to save how each method in "directMethods" access the table. For example, if method A delete table B, then for the class of table B we save A with "delete". PS: one method might have several relationships with one table.
public string name. Table's name.
public string title. This is the title of this table that we showed in the final report.
public string attribute. This string would summary one table like when it creates and how many items it contains.
public string methodsDes. This string would summary all the methods that are related to this table and we show this string in the final report.

The methods of this class:
public List<string> getRelationships(string name). This method is used to return the relationship between the table and the method has the name same as variable "name". We need this method when we are generating final reports.
public void insertRelationships(string name, string SQL). This method is used to handle a new relationship we found between a method "name" and this table.
public string TakeSpaceOff(string ori). This function would delete the last space of a string. When we are generating final report some strings contain useless spaces in the last position so I delete them with this method.
public string generateLeftIndex (). This method is used to generate the index of this table and the index is put in the left side of the report. With this index we could put the table and the columns belong to this table together.
public void generateDescription (dataSchemer db). This function would generate the description about this table and the description is saved in attribute.
public void insertMethod(desMethod m, string opt, string instruction). This method is used to handle a new SQL statements "opt" that we found in the method "m". The instruction could be "direct", "follow" and "final" so we could relatively insert the method into three lists of methods.

4. Class name: dbColumn, Location: SrcTest/DatabaseInfo.

The function of this class:
This class is defined to save all the data we need of each column. this method could directly refer to dbTable, I create two classes just because I want to handle table and columns separately and their descriptions are different.

The variables of this class:
All variables are similar to dbTable, the difference are:
1) dbColumn does not contain the list of columns because column does not have sub-columns;
2) dbColumn has a new variable: public string tableName. This variable means which table the column belongs to.

The methods of this class:
All methods of dbColumn are similar to dbTable too.

5. Class name: ExtractMethodSQL, Location: SrcTest/MethodInfo.

The function of this class:
This class would go through all the methods in your target project to:
1: find out all the SQL statements;
2: build call graph;
3: generate method descriptions for methods with local SQL invocations.

The variables of this class:
public string LocalProj. This is the variable save where is the project we want to analyze which includes the project location and the project's name.
public string SrcmlLoc. This is the variable refers to where is the SrcML which is our tool for extract all the methods in the target project.
public List<desMethod> methodsInfo. This is the list for saving all the methods we have analyzed. We need this list because we need the information of each method many times (one method might happen in many call graphs). So we need this list to avoid repeating some process for one method (especially for getting description of one method, this step is achieved by "ABB.Swum" and it takes half time of my program).
public List<desMethod> allDirectMethods. This is the list for saving all the methods that have SQL local invocations (other methods that are also in the call graph of direct methods but do not have local invocation would not be saved in this list). We would pass this list to the class "MethodMapper" so we could see how each method interacts with tables and columns in details. In this class, we only extract all the SQL statement and save them in the class "desMethod".
public int sqlCount. This is the variable for saving how many sqlCount we found and this variable would be used to generate the summary of the target project. This summary would be showed in the report.
public CGManager cgm. This is the Call Graph Manager of the target project. With this variable we could easily know for each method, what are the methods it calls and what are the methods call it.

dataSchemer db. This is the database information we extract in class "Program". Here I set another variable to point it just because this class needs it many times so I want to save it in the class.

The methods of this class:
public void run(). This method is running after we generate the class and it contains two parts: 1. generating call graph of the target project; 2. calling GoThroughMethods to check that does each method contain SQL local invocation or not.
public void GoThroughMethods(IEnumerable<MethodDefinition> methods). Checking each method to figure out all SQL local invocations. The analysis contains: 1. finding all the variables that SQL invocation like variable with type "String" (The details about handle these variables are achieved in method "statIsDeclaration"); 2. finding all the string usage directly like "Select * from tableName (This is achieved by statIsOthers.)
After the first step, we would check all the text we get with the class "SqlStmtParser" to see which text is SQL invocation.
public void statIsDeclaration(MethodDefinition m, Statement stat, List<string[]> variables). Finding all the variables that SQL invocation. At first we check the variable's type, if it could not save string like type "int" then we do not check it. Then we figure out all the statement that are related to this variable by method "FindRelated" and then rebuild this variable based on these statements by calling "Rebuldstring". Finally, for each variable that could save string, we would know it's final status.
public void statIsOthers(MethodDefinition m, Statement stat, List<string[]> variables). For all the other statement, we directly check all the expression in the statement to see is it a SQL invocation or not. One special case is that the expression is a "MethodCall". In this case, the variable passed into the method also could be a SQL invocation and we handle this case by method "handleFunctionCall".
public string handleFunctionCall(Expression exp, List<string[]> variables). When we know an expression is a "MethodCall", we check it's arguments and handle the case like argument is m.call (string1 + string2). String1 plus string2 might be an invocation.
public desMethod getMethodInfo(MethodDefinition m). This would return the class desMethod of the method "m". The details of class desMethod is written below this class.
public desMethod newMethodInfo(MethodDefinition m). When the list "methodsInfo" does not contain the information of a method "m", we could generate a new "desMethod" class for "m" and save the new class into lists.
public int CheckVariableExist(string Name, List<string[]> variables). This method would check the variables for us. For some variables, it might be defined as: variable1 = variable2 + variable3. Then we need to check what is variable2 and variable3 to help us rebuild variable1.
public string GetVariableCont(string Name, List<string[]> variables). If list "variables" contain the variable of "Name" then we would return it's value.
public void UpdateVariable(string Name, List<string[]> variables, string cont). This would update the list "variables" with variable "Name" whose content is "cont".
public void FindRelated(MethodDefinition m, VariableDeclaration declcont, List<string[]> variables). This method is used to find all the statement related the the statement "declcont".
public string TakeQuotOff(string ori). This method is remove the quote of a string. For some cases, if the string is wrapped by the quote then our SQL analyzer could not find it.
public string GetExpCont(Expression exp, List<string[]> variables). This method is used to return the value of an expression. For example, a "MethodCall" expression would return "FunctionCall" for rebuild a string.
public string Rebuildstring(string varname, Statement targetstat, List<string[]> variables).This is used for rebuilding a complete string for the variable with name "varname".

6. Class name: desMethod, Location: SrcTest/MethodInfo.
The function of this class:
This class is used to save all the information we need about each method.

The variables of this class:
public string swumsummary. This variable save the summary generated by "ABB.Swum" and it should describe what's the usage of this method.
public List<MethodDefinition> followmethods. This list saves all the methods that call this method but not in the top-level.
public List<MethodDefinition> finalmethods. This list saves all the methods that call this method and in the top-

level.
public MethodDefinition methodself. MethodDefintion class is the class defined in "ABB.Srcml" and we wrapped it by our class.
public string name. The full name of this method.
public List<sqlStmtParser> sqlStmts. All the SQL invocations of this method.

The methods of this class:
public string translateStmt(string stmt, string TorC). For each SQL invocation, our "SqlStmtParser" only could return a simple typle like "selectStmt" and this method would translate the type into human languages.
public string getHtmlDescribe(List<string> allSql, string TorC). This method would generate the description of this method and we could show the description in the final report.

7. Class name: dbMethodSql, Location: SrcTest/MethodInfo.
The function of this class:
This function is used to save that how each method interacts with table and column in details. This class has to be contained by class "dbTable" or "dbColumn".

The variables of this class:
public string methodName. The name of this method.
public List<string> SQLSequence. All the SQL statements that "methodName" interact with parent table or parent column.

8. Class name: dbMethodMapper, Location: SrcTest/MethodInfo.
The function of this class:
This class would:
1: save methods information into tables class and column class;
2: generate methods description for methods without local SQL but still in call graph.

The variables of this class:
ExtractMethodSQL extractor. This is the information of target project we get in class "Program".
dataSchemer db. This is the information of target database we get in the class "Program".
List<dbTable> tablesInfo. This is the tables information contains in variable "db".

The methods of this class:
public void run(). This method would analyze all the methods" saved in the extractor and mapping these methods with correct table or column.
updateConnection(SqlStmtParser p1, desMethod m, string opt). This method would extract all the ids contain in one SQL statement. These ids might be table name, column name or just some id useless. Then, we would check how many table name and column name are contained in the SQL statement "p1". Finally, we would connect each table and column in the id list with this method "m".

9. Class name: SqlStmtParser, Location: SrcTest/SQLAnalyzer.
The function of this class:
For each possible SQL invocation, we would generate one SqlStmtParser class. This class would parse the result if statement we passed in is a SQL invocation.

The variables of this class:
public string nowStmt. The statement we want to analyze.
public ParseTree wholeTree. If the "nowStmt" is a SQL-invocation, we could build the parse tree for it.
public Parser parser. The core component about parsing a statement and it's provided in library "Irony".
public bool isStmt. Tell us the "nowStmt" is a SQL invocation or not.
public ParseTreeNode stmtType. If "nowStmt" is a SQL invocation this would be its SQL type like "SelectStmt" or "UpdateStmt".

public ParseTreeNodeList stmtPoints. All the tree nodes in the variable "wholeTree".

The methods of this class:
public string ReplaceQues(string beforestring). This method is used to replace the question mark in a statement. For some target project, their SQL invocation looks like "insert (?,?,?) into tableName" but this is not acceptable for the grammar of Irony, so I replace these "?" by this method.
public List<string> CheckEntireTree(ParseTreeNode node, string targetText). Irony does not provide a method to check all the nodes in their parse tree so I wrote this method to check all the nodes.
public List<string> AddList(List<string> mainList, List<string> addList). When we get a new Id in getAllIds we need this method to add them into list.
public List<string> getAllIds(). Get all the parse node with mark "Id" in the parse tree. In Irony, all the table names and column names would be marked as "Id" so this method could get them together and we would separate them in the class "dbMethodMapper".

10. Class name: infoCollector, Location: SrcTest/ReportGenerator.
The function of this class:
This class would collect all the information we need for generating final report and generate class "FinalGenerator" to generate the report.

The variables of this class:
public HashSet<SingleSummary> AllTableSummary. This list contains all the summaries about tables that we would show in the final report.
public HashSet<SingleSummary> AllColumnSummary. This list contains all the summaries about columns that we would show in the final report.
public string outputLoc. This variable represents where we output the report.

The methods of this class:
public void run(). This method would generate the summary for the project and prepare all the other data for final report.
public void GenerateSummary(string stsum). This function is used to generate the summary of whole target project. The summary is showed in the left part of our report, the details is defined in the two HashSet above.

11. Class name: FinalGenerator, Location: SrcTest/ReportGenerator.
The function of this class:
This class would map all the information comes from "infoCollect" into the template that is saved in the location "SrcTest/Templet".

The variables of this class:
public string AllShiftFunction. This string contains all the JavaScript functions for shifting methods' short names into full names and shifting them back. I define this part because the methods' names take too much space of the report so I want to simplify it.
public string projectSum. The summary we generate in class "infoCollector".

The methods of this class:
public void Generate(string path). Generating the final report based on the templet named "CourseHome".

12. Class name: SingleSummary, Location: SrcTest/Summary.
The function of this class:
This class is used to save the information of each table or column that we want to show in the report.

The variables of this class:
public string title. The title of a table or column in the report.
public string attributions. The attribution of a table or column.

public string methodInfo. The related method information of a table or column.
public string tableName. The table's name or the table a column belongs to.
public string tableIndex. Only the classes of tables have this variable, the classes of columns would set this as "".

Other files in the project:

13. **File: Class name: SqlGrammar, Location: SrcTest/SQLAnalyzer.** The function of this class is defining all the grammars of SQL statements. This is provided by Irony but I revised some grammar to detect more SQL invocations. For different unknown SQL grammar you might need to keep revising this class.

14. **File: Class name: SwumSummary, Location: SrcTest/Summary.** The function of this class is generating the description of a method. This class is provided by Boyang.

15. **Folder: "SrcTest/CallGraph".** The function of this folder is going through all the methods of a project and generating the call graph. This component is provided by Boyang.

16. **Folder "SrcTest/Templet".** The function of this folder is saving all the templets for generating reports.