EZQL: SQL queries and reports

Syed Iqbal Ahmad
Bilal Bhatti

1. EZQL Language Overview

1.1 Introduction

EZQL was designed to ease database programming by providing a simple language to write queries and run reports.

The language will use syntax and keywords familiar to a database designer, administrator, or developer and allow him/her to easily create programs. These programs will return either a single row or a list containing many rows. We will also allow for post processing to be done on the returned result.

1.2 Language Overview

As stated above, EZQL will allow the database developer to use simple syntax to write database logic. He will be able to receive input arguments and respond to them as needed. The language will support control flow, such as if statements and loops. It will also have basic variable declarations. This will allow for logic prior to running the query.

Further writing the query will not contain any complex JDBC logic. Only a query needs to be specified.
Behind the scenes JDBC will be used to retrieve the results and populate EZQL Wrapper objects.

An EZQL program is interpreted by the EZQL interpreter line by line. We decided to use an interpreter because most programs will be short, and the step of compiling should be saved.

1.3 Why use Java

A Java backend is ideal for a variety of reasons.

Java provides a variety of benefits including security, machine portability, network aware, etc. Further, there is corporate commitment to use Java for enterprise application development, especially for database driven applications, and standard Java architecture supports a model of separation of personnel roles and design tiers.

This support is critical for the allowance of different solutions for any particular technical problem. Often with other vendors there is a strong push to adopt their platform entirely. We wanted to stay with a language which avoids this type of scenario, and maximize flexibility and choice for the user.

For more information about Java please see the Java Whitepaper at
http://java.sun.com/docs/overviews/java/java-overview-1.html

1.4 Main Language Features

The main language features are described below.

1.4.1 Data Types

Common data types and operations are included for use. In addition some special data types are included to provide ease of use in dealing with database returns.

The types are:

- int
- boolean
- string
- row
- list

Operations for int include math operations such as +,-, *, /, and logical operators such as <, >, ==, !=

Operations for boolean include the major logical operations such as &&, ||, ==, !=

Operations for string include string concatenation.

Row and list are described in more detail later on. They are used for database operations.

1.4.2 Methods

Methods can be written to break up larger tasks into smaller blocks of code. These methods can be called by other methods. Further they can hand of a return value to the calling method.

1.4.3 Running queries

```
table role as MyRole {
      row getById(int id_p) {
            string s= ”select a, b, c from user where id = ” + id_p;
```

```
            row r= runQuery(s);
            return r;
      }
}
```

The query is simple declared, passed to the runQuery method, and executed. The result in this case is one row.

We plan to support two return data types, row and list. A list will be used to indicate the return type contains more than one row of data.

1.4.4 Variables, Control Flow

We want to support standard language features including the use of variables, if statements and for loops.

An if statement can be used to perform conditional checks. In this example it is being used to print a warning to the user

```
table role as MyRole {
      row getById(int id) {
            if (id < 0) {
                  print("Warning, ID < 0 ");
            }
            string s= ”select a, b, c from user where id = ” + id;
            row r= runQuery(s);
            return r;
      }
}
```

In this example, a simple calculation is performed before the query is run.

```
table grades as StudentGrades {
      //get classes where the grade is greater than the average of two
exams.
      row getClass(int exam1, int exam2) {
            int avg = (exam1+exam2)/2;
            row r= runQuery(“select class from grades where grade >= “
+avg);
            return r;
      }
}
```

1.4.5 Post-Processing Results

The results can be post processed easily.

We will support some commonly used methods, such as:

```
rowcount(row | list)
```

Further the results can be written to a comma delimited file.

1.4.6 Some Final Notes

A properties file will have to be available specifying how to obtain a connection to the database. This file will also contain a mapping of Java types to database types. This file will need to be available to the backend in order to proceed with interpreting of .ezql files.
Further details about the language as a whole can be found in the tutorial below.

1.5 Related work

Our language serves two related purposes really, to supplant database clients for simple uses, and to be used as a reporting language.

With regards to database clients, many tools exist today with this type of functionality, specifically tools available from the major DB vendors.

We think our language offers some areas where it's more suitable than any normal client or reporting tools:

- EZQL supports methods
- EZQL will support easy post processing of the returned result, without having to step through any complicated menus.
- EZQL is pluggable to allow for extensions to be written. Usually the software you get can only be used as is.
- EZQL is lightweight and runs quickly.

2. Language Tutorial

2.1 Setup

EZQL requires a JDK to be installed. The supported JDK is 1.4.2. Other JDKs may way work as well.

EZQL will always be easy to setup. Just add the ezql.jar to your classpath and run the file ezql.Main. It needs one argument, the filename to be interpreted.

2.2 Tutorial

The tutorial will be brief with some basic snippets.

The header of an EZQL file is:

```
table grades as StudentGrades {
```

"table" is a keyword and must be specified.
"grades" can be any identifier. As a convention it should represent the table name that this file is mostly interested.
"as" is a keyword and must be specified.
"StudentGrades" can be any identifier, although it should match the file name, as this a possible future enhancement to EZQL.

After the table declaration method headers can be defined.

```
int foo(int x, int y, int z) {

}
```

A method must specify a return type. It must have a name. After the name must be "()". Inside of the parenthesis arguments can be specified. A method is began with a "{" and ends with a "}".

Any amount of methods can be declared in a file. However there should be a main() method for the EZQL Interpreter to call, as this is the only method which is directly invoked by the interpreter.
The main method must have the signature below.

```
int main() {
```

```
}
```

A method body can have a variety of simple assignment statements. Some are:

```
int x = 5;
string s="Hello World";
boolean b = true;
int x = x + 10;
```

For displaying output a print statement is available.

```
print ("X : " + x);
```

The print method accepts any variables or string literals. Further it takes a comma delimited set of arguments.

conditional checking can be done by using if statements:

```
boolean b = true;
if (b) {
      print ("Something was printed");
}
```

Loops can be performed using the while loop.

```
int x = 0;
while (x < 5) {
      print ("X "  + x);
      x = x + 1;
}
```

All methods must have a return statement that returns a value matching the declared return type.

```
int x = 54;
return x;
```

Putting all of it together gets us:

```
int doSomething() {
      boolean b = true;
      int i = 0;
      while (b == true) {
            i = i + 1;
            if (i==10) {
                  b = false;
            } else {
                  //do nothing
            }
      } // end while
      return i;
```

```
}

int main() {
      int i = doSomething();
      print("I : " + i);
}
```

Comments are began with a // and terminated with a newline.
Any text after a comment is ignored.

Built in methods are provided to handle the database functionality. These include:

```
list theList = runQuery("Select * From Users");
row theRow = runQuery("Select * From Users where name = '" + name
+ "'");
```

runQuery can return either a row or a list. A list is a collection of rows.

Once a list is obtained, rowCount can be used to get the count of the rows in a list.

```
int x = rowcount(theList);
```

The contents of the row or list can be printed for the user to see.

```
print(theRow);
print(theList);
```

Finally a method called merge can be used to write data to a file.
The style of the data is dependent on templates created in the apache velocity language. Some default ones are included, and further the user can customize the templates as needed.

```
merge(theList, "template.vm", "outfile.txt");
```

merge will write out the contents of the variable theList to outfile.txt, following the rules in template.vm.

3. EZQL Reference Manual

3.1 Introduction

This manual provides information for using the EZQL language for simplified database access. It is the standard for EZQL implementations, and any EZQL implementations must conform to this standard.

The style and syntax of EZQL is similar to Java, although vastly scaled down. EZQL files can be in an ASCII file.

3.2 Lexical Conventions

The first pass of the compiler through an EZQL will result in a set of tokens being produced.

3.2.1 Tokens

The significant tokens are Identifiers, Keywords, String literals, Operators and Constants. There is also white space which is ignored except to separate tokens, and comments which are ignored.

3.2.2 Whitespace

Whitespace can be ' ', '\r', '\n', \t'. They can separate tokens, such as:

int x;

There is a space between int and x. There could be one space or many spaces.

3.2.3 Comments

Only single line comments are allowed. They begin with // and continue until a valid newline character is found.

*COMMENTS : "//" (~('\n'|'\r'))**

3.2.4 Identifier

An identifier can consist of letters, digits and underscores. It must begin with a letter. They can have any length. They are case-sensitive (The identifier 'num' is different than 'NUM').

*ID : ('a'..'z' | 'A'.. 'Z') ('a'..'z' | A'.. 'Z' | '0'..'9' | '_') **

Identifiers are used to identify a single table file, a method name or a variable declaration.

3.2.5 Keywords

All keywords that are specified in the Java specification are reserved. This is to provide support for expanding the language, as well as to change our language from an interpreted one to a compiled one, using the BCEL project.

Some of the Java keywords are keywords in EZQL.

Further we introduce some new keywords:

as
list
query
row
table
view
report

3.2.6 Constants

constantValues
: INTEGER
| STRING_LITERAL
| ("true" |"false")

There are a variety of constant types. They are integers, string and boolean.

Integers are any valid base 10 numbers that consist of a sequence of digits.

Example: 123

Boolean constants are only "true" or "false".

A string is a sequence of characters enclosed in " ".

3.2.7 Operators

Boolean Operators: &&, ||, !
Arithmetic Operators: +, -, *, /, %
Logical Operators: <=, >=, >, <

Rules of precedence are the same as expected for languages in common use.
For math operators *, /, % have more precedence then +,-.
Parenthesis will have highest precedence.

Boolean operators and logical operators have precedence left to right, detailed more below.

3.3 Program Structure

After the lexical analysis, a parsing phase is performed to make sure the program structure is valid according to the rules below, and to generate an AST.

The program structure is as follows.

compilationUnit
: (tableDefinition)

A compilation unit is a valid EZQL program. It consists of a table definition.
The table definition will consist of smaller portions which are needed to make an EZQL program.

tableDefinition
: "table" ID "as" ID "{"
*(tableBody)**
"}"

A table represents a single file. The first ID is used as a convention to represent the database table most operations in this file are performed on. In the future we may provide the ability to have shortcuts based on the declaration.

The second ID must match the name of the file.

tablebody is the main body of the program, specified below.

```
tableBody
        : (methodDefinition)*

methodDefinition
        : (builtInTypes) ID "(" (argumentList)? ")" "{"
          (statement)*
         "}"

builtInTypes
        : "int"
        | "string"
        | "float"
        | "boolean"
        | "row"
        | "list"
```

A method definition begins by indicating the data type the method is expected to return. The ID represents the name of the method. A method consists of statements which are specified below.

Further a method takes arguments. They are as defined below:

```
argumentList
        : argumentSpec (COMMA argumentSpec)*

argumentSpec
        : builtInTypes ID
```

This definition for arguments is basically a comma-separated list of arguments, such as:

int x, string s, boolean b, int i

3.4 Statements

Statements must be part of a method definition. They are executed in order.

```
statement
        :       (       (assignmentStatement
                        | ifStatement
```

```
                    | whileStatement
                    | returnStatement
                    )
            )
```

Valid Statements are:

3.4.1 Assignment Statement

assignmentStatement
: ((builtInTypes ID | ID) "=")? expression ";"

An assignment statement may look like
int x = 5;

However once the variable ID has been declared it can not be re-declared.
For example, both of the statements below are valid statements:
int x = 5;
x = x + 5;

Invalid statements, because the variable y is re-declared:
int y = 5;
int y = y + 5;

The whole assignment portion can also be optional, in order to allow for the running of expressions to generate their side effect.

Expressions will be explained in more detail below.

3.4.2 Return Statement

returnStatement
: "return" (ID)? ";"

ID in return statement must be of builtInTypes and match the type in the methodDeclaration.

3.4.3 While Statement

whileStatement
: "while" "(" expression ")" "{"

*(statement)**
"}"

A while statement is a loop statement. The statements in between "{" and "}" will be run continuously until expression evaluates to true.

Although not all expressions are required to evaluate to a boolean, the expression in a while statement must evaluate to a boolean, otherwise an error will be generated.

3.4.4 If Statement

ifStatement
: "if" "(" expression")" "{"
(thenBlock)
"}" (elseBlock)?

thenBlock
*: (statement)**

elseBlock
: "else"! LCURLY!
*(statement)**
RCURLY!

An if statement consists of an expression. If the expression evaluates to true, the statements which are part of thenBlock will be run. If the expression evaluates to false, and an elseBlock is present the statements which are part of elseBlock will be run.

Although not all expressions are required to evaluate to a boolean, the expression in an if statement must evaluate to a boolean, otherwise an error will be generated.

3.5 Expressions

Expressions follow precedence rules. The expressions are listed below in precedence order, beginning with the lowest precedence first.

Further they are evaluated left to right for any rules at the same level.
The expressions listed below perform their commonly known task. For example the "+" operator is used to add two expressions, the "*" operator is used to multiply two expressions, etc.

expression
*: andExpression ("||" andExpression)**

andExpression
*: relationalExpression ("&&" relationalExpression)**

relationalExpression
*: additionExpression (("="|"!="| "<" |"<=" | ">" | ">=") additionExpression)**

additionExpression
*: multiplyingExpression (("+" |"-") multiplyingExpression)**

The plus operator can be used to also perform string concatenation.

multiplyingExpression
: signExpression (("" | "/" | "%") signExpression)**

signExpression
: ("+" | "-")? negationExpression

negationExpression
: ("!")? expressionElements

expressionElements
: constantValues
| ID
| methodCall
| "(" expression ")"

constantValues
: INTEGER
| CHAR_LITERAL
| ("true" |"false")

3.6 Methods

An expressionElement can consist of a call to another method. The syntax for methodCall is defined below. Further EZQL supports a variety of internal methods which the implementers of the language have provided.

These are shown below, in the format returnType ID "(" parameterList ")"

list | row runQuery (string);
void print(string);
int rowcount(list);

void merge(list, string, string);

runQuery takes a query as an argument and executes it using JDBC as the backend. It can return a list or row.

print sends the input to the system console.

rowCount returns the number of rows in the particular list.

Merge takes a list, a template file and an output file, and writes the list to the output file in the format specified by the template file.

methodCall
: (ID "(" parameterList ")"

parameterList
: (expression ("," expression))?*

3.7 Scope

The assignmentStatement can only be inside of a methodDefinition. The scope of any variables declared is only for the enclosing method. After the method is terminated, the current stack will be discarded, and a new stack made for the currently executing method.

4 Project Plan

4.1 Development Process

Our idea for EZQL was conceived by discussions between the team of problems faced between the layer of Java and the database. This lead to the idea that we should create a language which has a familiar syntax and supported database access in a simple way, without having to write the same code which is commonly cut and pasted. Finally we realized that this approach lends itself to being used as a tool for reporting, so we decided we wanted support for outputting the returned data to a file.

After we agreed on the above, we laid out sample source files for what we believed our language should like. These sample files were the main "specs" initially relied on. There was an iterative cycle of reviewing our initial idea against our source files, to make sure that the language we created solved our initial problem. With the above done, we agree to some more detailed specs.

The development process used initially was pair programming, since this project consisted of two members. After some significant portions were completed the work was divided up so that we could work quickly to meet deadlines.

Development began with the Lexer and Parser, mirroring what was agreed upon in the Language Reference Manual. At times we found that we needed to be more flexible then what was agreed upon in the LRM, so we revised it and moved forward accordingly.

After this was done, it was fairly clear what we needed to provide in terms of the backend. We implemented a two pass Walker to parse an AST. The first pass handled declarations, the second pass handled the actual running of the language.

The testing of the language proceeded in two phases. We were able to use the sample files to test the front end of the compiler. We added to those files to give us a more robust suite of test cases, especially the handling of special cases. The backend of the compiler was tested using JUnit. We felt this would be better then reinventing the wheel, since using JUnit is recognized commercially as a standard testing tool.

Development and testing proceeded iteratively until we reached our final product.

4.2 Style Guide

The adoption of a style guide is critical to any successful team project. Without a guide, developers will write code according to their own preferences.

There were two sets of code mainly developed, the Java backend and the ANTLR grammar files.

Fortunately for Java there is a well established standard which all developers of the team are familiar with and used on former projects.

http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

This guide was adhered to for the Java code.

For ANTLR we looked at previously created files which existed to determine what styles were appropriate. The conclusions we came to are listed below.

- Following similar naming conventions as Java. For our rules, the first letter is lowercase and any new words in that rule are uppercase.

  e.g.
  returnSpec

- Adding a newline after the rule declaration.

  e.g.
  returnSpec
  :

- Adding a newline after each OR block (|).

  e.g.
  builtInTypes
  : "int"
  | "string"
  | "float"
  | "boolean"
  | "row"
  | "list"
  ;

- Adding a "{" "}" at any needed places, such as if statements, regardless of whether it is optional. This leads to clarity and saves debugging headaches later.

  e.g.

```
methodDefinition
    : #(METHOD_DEF rt:returnTypes id:ID (al:argumentList)? (mb:.)? )
    {
        if (id.getText().equals("main") && al == null) {
            methodBody(#mb);
        }
```

```
}
;
```

- Adhering to a limit of 80 characters per line. This is needed to make the code readable at all times.

Following the above guidelines ensured that our code was consistent, easy to read and maintain.

4.3 Project Timeline

The following deadlines were set for key project development goals.

09-28-2004	Language whitepaper, core language features defined
10-10-2004	Development environment and code conventions defined
10-21-2004	Language reference manual, grammar complete
12-07-2004	Alpha release for backend
12-14-2004	Beta release for backend, Error Handling
12-21-2004	Code freeze, project complete

4.4 roles

There were not defined roles since the project consisted of two members. The requirement we felt was appropriate was for each member to be familiar with all aspects of the project and work side by side.

4.5 software development environment

We agreed to standardize on some tools to avoid any environment problems. Specifically we used:

- Java, JDK 1.4.2
- ANTLR
- Eclipse IDE
- ANTLR plugin for eclipse
- CVS
- MySQL Database
- JUnit 3.8.1

The decision to use Eclipse and the ANTLR plugin proved critical in working quickly and being able to automatically recreate the ANTLR generated code after modifying the ANTLR grammar files.

CVS was also needed to allow us to look at previous checkins in case we needed to rollback.

4.6 Project Log

The list below represents a rough outline of the tasks completed for this project, listed in the order of completion.

- Project Main ideas, Brainstorming
- Ideas about language formatting and syntax
- Whitepaper creation
- Defining of basic syntax rules
- identifiers, string literals, ints
- comments
- assignment statement
- program structure, method body
- test scripts for existing functionality
- Language reference Manual
- creation of EZQL types
- getting method calls to work
- return statement
- if statement, while loop
- test scripts for existing functionality
- ideas about AST, initial efforts to get usable AST
- creation of symbol table
- creation of EZQLRuntime to manage program
- refactored symbol table to allow method scope

- test scripts for existing functionality
- changed grammar to be multi pass so that methods could be called before they are declared
- added support for internal method calls, Java/JDBC calls
- Completion of backend
- final testing

5. Architectural Design

5.1 Interpreted

EZQL is an interpreted language as opposed to a compiled one. We considered generating Java source from EZQL files or byte code. We discussed generating Java source code by hand; also there are a few libraries available that would have allowed us to generated byte code on the fly as we parsed through the file. BCEL and CGLIB are two such libraries. They both provide a mechanism for programmatically manipulating and generating byte code that can be loaded into the JVM and executed along with other code. That, performance wise, would have been the optimal solution. However, considering the small size of the EZQL as a language and size of a typical EZQL program, it would not have made a big difference. In addition since EZQL is parsed and executed in the same JVM it further reduces the performance gap, because the JVM doesn't have to be loaded multiple times into the memory. So we decided to interpret the source code instead of translating it to Java. That did pose a number of challenges as we had to now maintain a lot of runtime information, that otherwise would have been a non-issue. We do think this was the best approach, as it really exposed us to different aspects of compiler design and the complexities involved. We had to implement our own typing system and a mechanism for scope management.

5.2 Statically Typed

EZQL is implemented as a statically typed language. It supports a basic set of types. Namely:

```
string
int
boolean
list
row
```

We found that it was easier to catch errors earlier in execution with static typing. The first pass performs syntax analysis and ensures that code doesn't contain any syntactic errors.

5.3 Architecture

The EZQL interpreter is built on ANTLR and Java. The front-end functions of lexing and parsing are implemented with ANTLR and the backend runtime environment is developed in Java. Its architectural layout is depicted in figure 5.2 below.

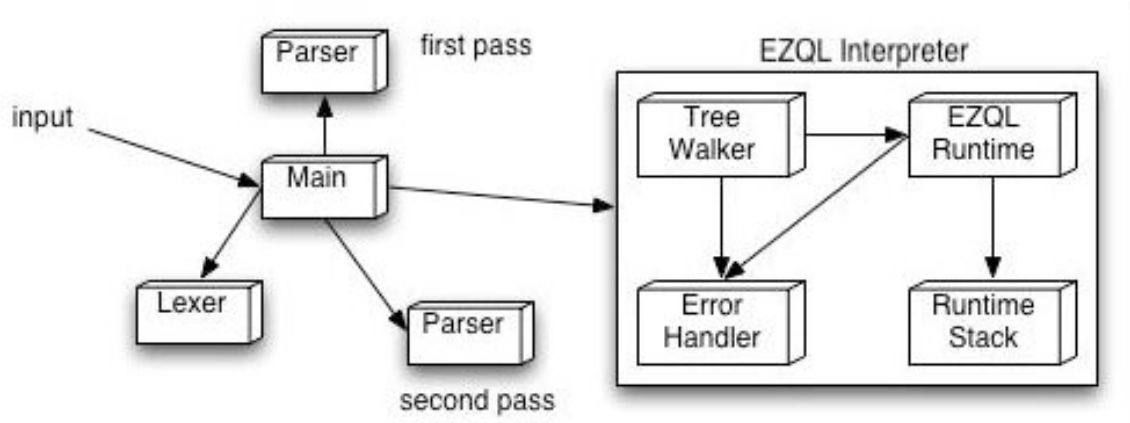


Figure 5.2

We implemented the compiler as a two-pass compiler. We perform code analysis and dependency checking in the first pass and the actual interpretation in the second pass. This gives us the ability to arrange the source code in any manner without concern for evaluation order; in addition we can implement loading of external libraries written in EZQL without much effort.

The backend is implemented in Java. We designed the backend with extensibility in mind. We employ a few standard patterns, such as Command, Builder, State etc. Our architecture allows EZQL to be extended fairly easily. We can add additional language functional features without touching the core code base, by using these patterns. EZQL can dynamically load new feature implementations. This is easily done by implementing a simple interface and registering the implementation by placing it in a specific location. The implementation is than loaded and registered using the Java classloader through the reflection API.

The main components of EZQL are the parser, the runtime and the type engine. The tree walker interacts with the runtime and invokes appropriate actions. It acts as the interface between the other components. The implementation of type checking and expression evaluation is handed off to the **`EzqlTypeHelper`**, which enforces type rules and invokes code for expression evaluation. Data types extend a simple abstract class which declares some basic methods for type checking and providing runtime information to the interpreter.

```
public abstract String getType();
public void setReturned(boolean ret);
public boolean isReturned();
```

The type helper exposes the following operations:

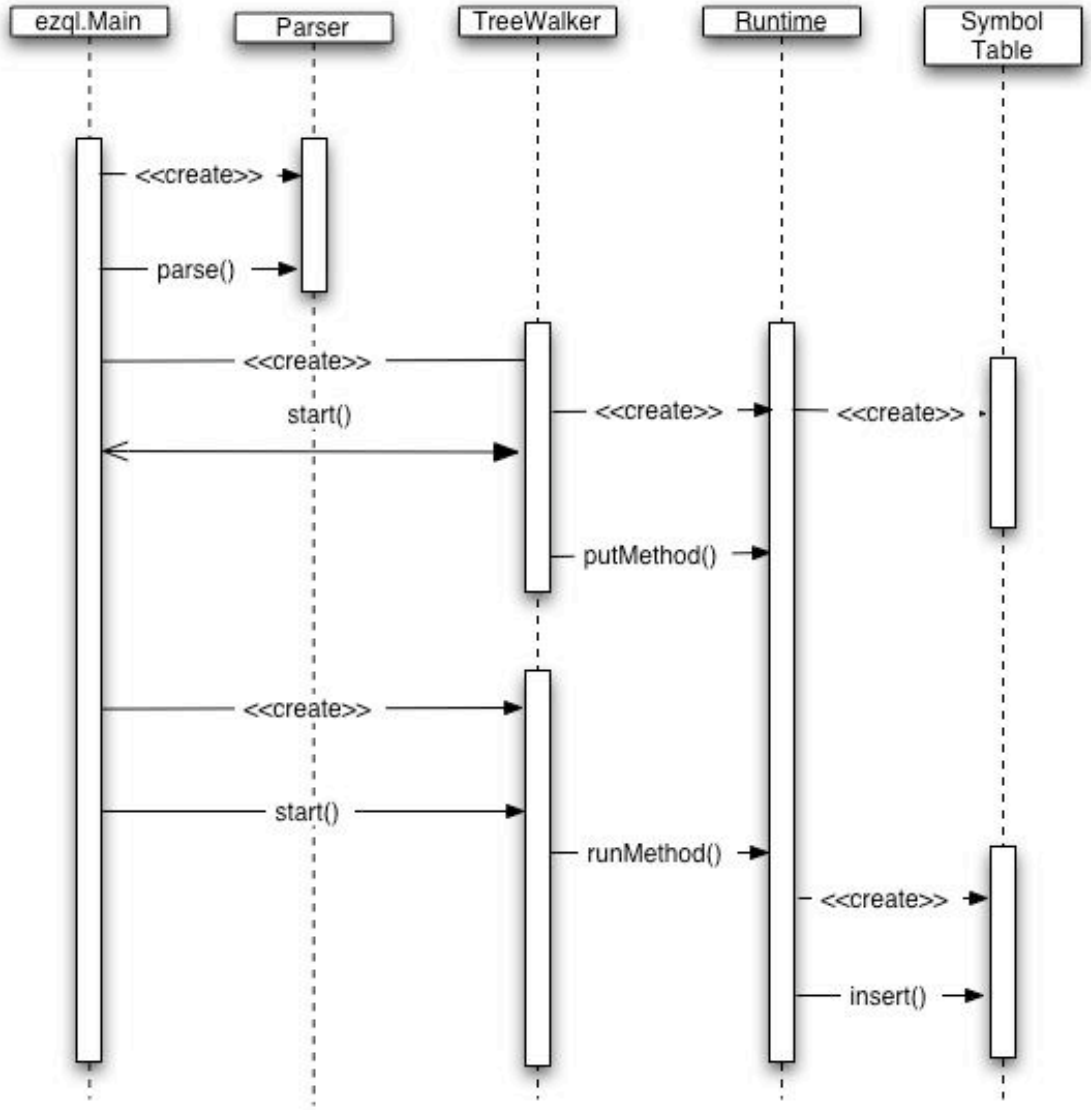
```
public static boolean sameType(EzqlType a, EzqlType b);
public static EzqlType plus(EzqlType a, EzqlType b);
public static EzqlType minus(EzqlType a, EzqlType b);
public static EzqlType star(EzqlType a, EzqlType b);
public static EzqlType div(EzqlType a, EzqlType b);
public static EzqlType mod(EzqlType a, EzqlType b);
public static EzqlType equal(EzqlType a, EzqlType b);
public static EzqlType notEqual(EzqlType a, EzqlType b);
public static EzqlType ge(EzqlType a, EzqlType b);
public static EzqlType gt(EzqlType a, EzqlType b);
public static EzqlType le(EzqlType a, EzqlType b);
public static EzqlType lt(EzqlType a, EzqlType b);
public static EzqlType lor(EzqlType a, EzqlType b);
public static EzqlType land(EzqlType a, EzqlType b);
public static EzqlType negate(EzqlType a);
```

The interpreter is invoked by running **`ezql.Main`** class. It invokes the lexer and parser then initializes the runtime environment and hands control over to the tree walker. From that point the runtime system manages the flow and state of the interpreter.

```
EzqlLexer lexer = new EzqlLexer(new FileInputStream(srcFile));
EzqlParser parser = new EzqlParser(lexer);
parser.compilationUnit();
AST t = parser.getAST();

// FIRST PASS FOR CODE ANALYSIS AND VALIDATION
EzqlInitialTreeParser eitp = new EzqlInitialTreeParser();
EzqlRuntime runtime = eitp.start(t);

// SECOND PASS FOR EVALUATION
EzqlTreeParser tp = new EzqlTreeParser();
tp.start(t, runtime);
```



5.4 The Runtime Environment

The EZQL runtime environment tracks code evaluation and maintains scope information. It consists of two components, the runtime stack and the symbol table. The symbol table is responsible for storing variable values and results of expression within a specific scope. The runtime is responsible for initializing and destroying scope. There are two types of scopes supported by EZQL, global and method. Runtime maintains a stack and at method calls instantiates a new scope and pushes it onto the stack, which can be referenced by other components of the system, in a generic way. Runtime exposes methods that can be invoked to interact with the current scope, namely:

```
// USED DURING THE FIRST PASS TO VALIDATE AND REGISTER METHODS
```

```
public void putMethod(String name, ArrayList params, String returnType,
AST body);

// USED DURING EXECUTION TO MAINTAIN SCOPE AND STATE
public EzqlType runMethod(EzqlTreeParser walker, String name, ArrayList
list) throws RecognitionException;
public void insert(AST type, String var, EzqlType value);
public EzqlType lookup(String var);
public void clear();
public void print();
```

After successful invocation of a method call the returned values are extracted and set in parent scope and the current scope is popped off the stack and discarded.

5.5 Error Handling

Error handling interface is provided by the **`ErrorHandler`** and **`LineNumberAST`** classes. **`LineNumberAST`** extends the **`CommonAST`** to extract and store line number information during parsing phase. Then during interpretation time when an error is encountered, the current AST node is passed to the **`ErrorHandler`** class along with an error message. The **`ErrorHandler`** raises an **`EzqlException`** which extends the **`RuntimeException`** with the provided message, which is caught by **`ezql.Main`** and prints the error message. This effectively stops execution of the code. The **`LineNumberAST`** provides one method of interest, which we added:

```
public int getLineNumber();
```

The ErrorHandler also exposes three methods that can be invoked when an error is encountered:

```
public static void handle(String msg);
public static void handle(AST t, String msg);
public static void print(AST t, String msg);
```

5.6 EZQL Core Function Library

The function library adds facilities primarily for SQL query post processing and generating text based reports. It is implemented using the Command pattern. All of the classes reside in the **`ezql.internal`** package. It is easily extensible and new functions can be added easily. One interface needs to be implemented to extend library capabilities. The package also contains all of the database related classes and helper classes for manipulating result sets.

```
public interface EzqlInternalMethod {
      public EzqlType runMethod(ArrayList args) throws EzqlException;
}
```

The following is an example of how easy it is to add further capabilities.

```
public class EzqlRunQuery implements EzqlInternalMethod {
    public EzqlType runMethod(ArrayList args) {
        EzqlType param = (EzqlType)args.get(0);
        String query = param.toString();
        return new EzqlList(EzqlQueryHelper.runquery(query));
    }
}
```

In addition one of the nice features we provide is the facility to generated text based reports by simply calling a built-in function to merge a list returned by a query and template. The library also provides functions for list and row manipulation.

5.7 Work Division

Bilal Bhatti

- Data Typing component
- Runtime state/stack component
- Core function Library
- Error Handling

Syed Iqbal Ahmad

- Core function library
- Symbol table

6. Test Plan

6.1 Goal and Approach

Our goal for testing was two fold, driving our development process and verifying correctness of developed code. Naturally, it is very difficult if not impossible to test fully and completely and have no bugs in the end product. You have to try nonetheless. We wanted to integrate testing in our coding process. We tried our best to actually implement that, however there were times when that was not the case and testing took a back seat.

Since our team consisted only of two members it was easy to ensure that we both stuck by whatever testing/development approach we decided, though deciding on that was sometimes difficult given the project. We setup some utilities that we would both use during development and sometimes we both had to be creative in our testing given particular problems. In the end, though it was difficult, testing was not one of our major problems.

Developing a testing plan for EZQL was challenging because it involves a a few different technologies and has a number of third party dependencies. As mentioned already, the front-end is entirely built on ANTLR and testing the lexer, parser and tree walk is not so straight forward, because there are a lot of little nuances with the ANTLR grammar that can be tricky and it is not easy to unit test as is possible for Java or most other languages. So because of that we had to test different parts of the project differently. Our test automation approach can be broken down into three parts:

- Lexer and parser
- Back end java implementation
- Interpreter

6.2 Test Automation

We used JUnit, shell scripts and batch files as needed to setup our test harness. Below is a snippet of our main testing shell script.

```
// TEST LEXER AND PARSER
java ezql.TestParser sample/recursion.ezql
java ezql.TestParser sample/query.ezql
java ezql.TestParser sample/merge.ezql
java ezql.TestParser sample/table.ezql
java ezql.TestParser sample/newtable.ezql

//UNIT TEST JAVA CODE
java ezql.test.MainTestSuite
```

```
//TEST EZQL SOURCE FILES
./runezql.sh sample/recursion.ezql
./runezql.sh sample/query.ezql
./runezql.sh sample/merge.ezql
./runezql.sh sample/table.ezql
./runezql.sh sample/newtable.ezql
```

6.2.1 Lexer and Parser

To test the lexer and parser, we developed a number of java classes that are invoked from our main test script. The classes process the file and provide feedback about the correctness of the grammar. ANTLR provides a mechanism for viewing the parsed input as a visual tree representation. That was our main method of testing and verifying our grammar for parsing and AST construction. We set up this utility fairly early in the project and it was an invaluable tool for us. It provided immediate feedback and helped greatly with constructing the tree in the way we needed and debugging.

6.2.2 Java

We used JUnit heavily for unit and regression testing of our Java code. Professionally, both of us have worked on a number of projects, so it wasn't very difficult to see the need for method level testing of our code and JUnit is a very easy tool, yet it works great. We have three main test suites, for testing third party dependencies, type rule checking and expression evaluation. We tested these components at the method level.

6.2.3 EZQL Interpreter

Towards end of the project, naturally, we had to test the interpreter as a whole and that proved to be the most challenging part, for the reasons mentioned earlier. For a while because of the deadline, we each just tested it any way we could think of. Eventually, however, we decided on a simple approach and just wrote a bunch of files and stuck them in our main testing shell script. Now we can run one shell script and test different parts of EZQL together. That has really been very helpful. Below are listings of some of our EZQL files that were used to test. We added to these files incrementally as new language features were added. So they don't focus on doing anything useful or logical, they are meant simply to verify correctness of the compiler for given some input.

6.3 Test Cases

```
expression.ezql
----------------------------------------

table expression as Expression {
```

```
	int testBoolean() {
		boolean b = false;
		boolean c = b && true;
		boolean d = x < 10;
		boolean e = x > 10;
		boolean f = x == 10;
		boolean g = x <= 10;
		boolean h = x >= 10;
		boolean i = c && d;
		boolean j = e || f;
		boolean k = i==5 && true;
		return;
	}

	int testMath() {
		int x = 5;
		int y = 10;
		int z = x + y;
		int a = x * y;
		int b = x - y;
		int c = x / y;
		int d = a + b - c + d;
	}

	int testStatement() {
		boolean b = true;
		int i = 0;
		while (b==true) {
			i = i + 1;
			if (i==10) {
				b = false;
			} else {
				//do nothing
			}
		} // end while
	}

} // end file


recursion.ezql
-----------------------------------------
table recursion as Recursion {
	int main() {
		int x = foo(5);
		print("outer x : " + x);
	}

	int foo(int x) {
		if (x == 0) {
			return x;
		} else {
			print("inner x : " + x);
			foo(x - 1);
		}
	}
```

```
}

merge.ezql
----------------------------------------
table users as Users {
      int main() {
            list l = runquery("select * from users");
            print(rowcount(l));
            merge(l, "ezql.vm", "ezql.output");
      }
}

newtable.ezql
----------------------------------------
table role as MyRole {

      int main() {
            foo(4, 1, 6);
            return 0;
      }

      int foo(int x, int y, int z) {
            boolean a = false;
            if (a) {
                  string b = "if clause evaluated";
                  string c = "if clause evaluated, 2nd statement";

            } else {
                  string d = "else clause evaluated";
                  string e = "else clause evaluated, 2nd statement";
                  if (1 != 2) {
                        string f = "nested if clause evaluated";
                        string g = "nested if clause evaluated, 2nd
statement";
                  }
            }
            print ("A:" + a);

            while (x < 5) {
                  print ("x:" + x);
                  x = x+1;
                  if (true) {
                        x = 5;
                  }

                  while (y < 5) {
                        y = y * 2;
                        print("value of y is: " + y);
                  }
            }
            int b = 5 + 5 * 4 / 4 % 2;
            return b;
      }
}

query.ezql
----------------------------------------
```

```
table users as Users {
      int main() {
            list l = runquery("select name from users");
            int x = rowcount(l);
            print("row count: " + (x - 1));

            while(x > 0) {
                  x = x - 1;
                  print(columnvalue(l, "name"));
            }
            return 0;
      }
}
```

6.4 Work Division

Bilal Bhatti

JUnit Integration and tests

Parser testing class

EZQL source files for testing

Shell scripts

Syed Iqbal Ahmad

JUnit tests

EZQL testing class

EZQL source files for testing

MS DOS batch files

7. Lessons Learned

Iqbal Ahmad

I could say many things about the lessons I learned from this project. Since our project consisted of only two people, I got to be involved in nearly all aspects of the project. I of course benefited from either working directly or indirectly on all of the different portions, the Lexer, Parser, TreeWalker and backend Java code. I found it interesting to see all of the decisions which go into the making of a language, both the technical side and also the syntax side. The syntax side is interesting because it may not

offer anything from a performance point of view but increases the usability of the language, and if you don't like something or ever wished there was a feature in a language, you can add/change your language. The technical side offers a variety of implementation challenges which are rewarding once overcome.

Advice:

First I would emphasize highly on the importance of getting started early, especially making a schedule and sticking to it. In addition, get started with ANTLR as early as possible, because it has a lot of little nuances, and you might think your project is moving along ok, only to get stuck fiddling around with trying to get ANTLR to accept your grammar. Work incrementally, but always think towards the big picture so that you don't lose sight of the main goal.

Bilal Bhatti

This was a very rewarding course project. I can honestly say I was challenged and I learned a lot. Compilers prior to this class were pretty much black boxes to me, source code went in and some how executables came out. This project changed that. The most interesting thing I learned from this project would have to be the various parsing algorithms; because I have been in situations where I needed to parse some text files in order to extract information and the approach I always took was very different from the way these lexers and parsers work. And a tool like ANTLR is just plain cool. When I started the project I didn't realize what could be done with a tool like that, that's partly why we didn't provide our language specific query mechanism because we thought parsing SQL wouldn't be clean and easy. Now, however, it doesn't seem so daunting. We could've easily built an ANTLR grammar file for SQL, may be not for the entire syntax but the more common SELECT statement syntax would not be too difficult.

My advice to anyone taking this course would have to be to get started on the grammar quickly. Go through tutorials to get a feel of ANTLR. There are a lot of little weird things about the way it works. Documentation for it is not so great; your best sources are the previous semester projects. Also, build the parser and the tree walker together. The parser needs to generate a tree the walker can walk effectively and easily. If the parser is not designed with that in mind, you will have to constantly adjust your parser. We learned that the hard way. Also, try to specify the syntax and features of your language as much as possible prior to jumping in to write the grammar.

8. Appendix

The appendix lists all of our source files. We didn't include the authors' name, since there is no particular author for any of the files. Almost all of the files were created/modified by both authors.

expression.ezql

```
-----------------------------------------------------
table expression as Expression {


      int testBoolean() {

            boolean b = false;

            boolean c = b && true;

            boolean d = x < 10;

            boolean e = x > 10;

            boolean f = x == 10;

            boolean g = x <= 10;

            boolean h = x >= 10;

            boolean i = c && d;

            boolean j = e || f;

            boolean k = i==5 && true;

            return;

      }


      int testMath() {

            int x = 5;

            int y = 10;

            int z = x + y;

            int a = x * y;

            int b = x - y;
```

```
            int c = x / y;

            int d = a + b - c + d;

      }


      int testStatement() {

            boolean b = true;

            int i = 0;

            while (b==true) {

                  i = i + 1;

                  if (i==10) {

                        b = false;

                  } else {

                        //do nothin

                  }

            } // end while

      }


} // end table



merge.ezql
-----------------------------------------------------
table role as MyRole {
      int main() {
            list l = runquery("select name from users");
            print(rowcount(l));
            merge(l, "ezql.vm", "ezql.output");
      }
}



newtable.ezql
-----------------------------------------------------
table role as MyRole {
```

```
	int main() {
		foo(4, 1, 6);
		return 0;
	}

	int foo(int x, int y, int z) {
		boolean a = false;
		if (a) {
			string b = "if clause evaluated";
			string c = "if clause evaluated, 2nd statement";

		} else {
			string d = "else clause evaluated";
			string e = "else clause evaluated, 2nd statement";
			if (1 != 2) {
				string f = "nested if clause evaluated";
				string g = "nested if clause evaluated, 2nd
statement";
			}
		}
		print ("A:" + a);

//		x=0;
//		y=0;
//		while (x < 5) {
//			print ("x:" + x);
//			x = x+1;
//			if (true) {
//				x = 5;
//			}
//
			while (y < 5) {
				y = y * 2;
				print("asdfsdf" + y);
			}
//		}
		int b = 5 + 5 * 4 / 4 % 2;
		return b;
	}
}



query.ezql
-------------------------------------------------
table users as Users {
	int main() {
		list l = runquery("select name from users");
		int x = rowcount(l);
		print("row count: " + (x - 1));

		int y = 0;
		while(x > y) {
			x = (x - 1);
			print(l);
		}
		return 0;
```

```
	}
}



recursion.ezql
-------------------------------------------------
table recursion as Recursion {
	int main() {
		int x = foo(5);
		print("outer x : " + x);
	}

	int foo(int x) {
		if (x == 0) {
			return x;
		} else {
			print("inner x : " + x);
			foo(x - 1);
		}
	}
}


table.ezql
-------------------------------------------------
//package com.xyz.admin;

table role as MyRole {

	row getById(int id_p) {
		// No need for quotes to escape. Just keep searching until
the semicolon
		// return select a, b, c from user where id = :id_p;
		// execute query, process resultset and return a row
		return row_v;
	}

	list getBy2Ids(int id_p, int id_p) {
		// execute query and return processed resultset
		return list_p;
	}

	int testBoolean() {
		if (4 > 4) {
			x = 5;
		}
		X = "HELLO STRING";
		int x = 5;
		x = 5;
		b = 5 + 5 * 4 / 4 % 2;
		c = 5 < 6 == 5 && 6;
		boolean b = false;
		boolean c = b && true;
		boolean d = x < 10;
		boolean e = x > 10;
		boolean f = x == 10;
```

```
            boolean g = x <= 10;
            boolean h = x >= 10;
            boolean i = c && d;
            boolean j = e || f;
            boolean k = i==5 && true;
            return;
      }

      int testMath() {
            int x = 5;
            int y = 10;
            int z = x + y;
            int a = x * y;
            int b = x - y;
            int c = x / y;
            int d = a + b - c + d;
      }

      int testStatement() {
            boolean b = true;
            int i = 0;
            while (b==true) {
                  i = i + 1;
                  if (i==10) {
                        b = false;
                  } else {
                        //do nothin
                  }
            } // end while
      }

} // end class
```

ErrorHandler.java

```
-----------------------------------------------------
package ezql;

import antlr.collections.AST;

/**
 * @author bbhatti
 *
 */
public class ErrorHandler {

      public static void handle(String msg) {
            //System.out.println("Line # ?" + ": " + msg);
            throw new RuntimeException("Line # ? " + msg);
      }

      public static void handle(AST t, String msg) {
            //System.out.println("Line # " +
((LineNumberAST)t).getLineNumber() + ": " + msg);
            throw new RuntimeException("Line # " +
((LineNumberAST)t).getLineNumber() + ": " + msg);
```

```
	}

	public static void print(AST t, String msg) {
		System.out.println("Line # " +
((LineNumberAST)t).getLineNumber() + ": " + msg);
	}
}
```

EzqlBoolean.java

```
package ezql;

/**
 * @author iqbal
 *
 */
public class EzqlBoolean extends EzqlType {
	private boolean boolVal;

	public EzqlBoolean() {
		boolVal = true;
	}

	public EzqlBoolean(String bstr) {
		this.boolVal = new Boolean(bstr).booleanValue();
	}

	public EzqlBoolean(boolean bool) {
		this.boolVal = bool;
	}
	public String getType() {
		return EzqlType.BOOLEAN_TYPE;
	}

	public boolean getValue() {
		return boolVal;
	}

	public String toString() {
		return "" + boolVal;
	}
}
```

EzqlException.java

```
package ezql;

/**
 * @author bbhatti
 *
 */
public class EzqlException extends RuntimeException {
	String msg;
```

```
	public EzqlException(String msg) {
		this.msg = msg;
	}
}
```

EzqlInteger.java

```
package ezql;


public class EzqlInteger extends EzqlType {

	int x;

	public EzqlInteger() {
		x = 0;
	}

	public EzqlInteger(String s) {
		x = Integer.parseInt(s);
	}

	public EzqlInteger(int i) {
		x = i;
	}

	public String getType() {
		return EzqlType.INT_TYPE;
	}

	public int getValue() {
		return x;
	}

	public String toString() {
		return "" + x;
	}
}
```

EzqlList.java

```
package ezql;

import java.util.ArrayList;

/**
 * @author iqbal
 */
public class EzqlList extends EzqlType {
	private ArrayList rsList;

	public EzqlList() {
	    rsList = new ArrayList();
```

```
      }
      public EzqlList (ArrayList list) {
            rsList = list;
      }

      public ArrayList getValue() {
          return this.rsList;
      }

      public String getType() {
            return EzqlType.LIST_TYPE;
      }

      public String toString() {
          return rsList.toString();
      }
}
```

EzqlMethod.java

```
package ezql;

import java.util.ArrayList;

import antlr.collections.AST;

/**
 * @author bbhatti
 */
public class EzqlMethod {
    //args
    //name
    //return type
    //AST
    private ArrayList args;
    private String name;
    private String returnType;
    private AST body;

    public EzqlMethod() {
    }

    public EzqlMethod(ArrayList args, String name, String returnType,
AST body) {
        super();
        this.args = args;
        this.name = name;
        this.returnType = returnType;
        this.body = body;
    }
    public ArrayList getParams() {
        return args;
    }
    public void setArgs(ArrayList args) {
        this.args = args;
```

```
    }
    public AST getBody() {
        return body;
    }
    public void setBody(AST body) {
        this.body = body;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getReturnType() {
        return returnType;
    }
    public void setReturnType(String returnType) {
        this.returnType = returnType;
    }

    public String toString() {
        return "Method info: " +
                        "return type=" + returnType +
                        ", name=" + name +
                        ", args=" + args +
                        ", body=" + body;
    }
}
```

EzqlParameter.java

```
package ezql;

import antlr.collections.AST;

public class EzqlParameter {

      public String type;
      public String name;
      public AST typeAST;

      public EzqlParameter() {
      }

    public EzqlParameter(String type, String name, AST typeAST) {
        super();
        this.type = type;
        this.name = name;
        this.typeAST = typeAST;
    }

    /**
       * @return Returns the name.
       */
      public String getName() {
```

```
            return name;
      }
      /**
       * @param name The name to set.
       */
      public void setName(String name) {
            this.name = name;
      }
      /**
       * @return Returns the type.
       */
      public String getType() {
            return type;
      }
      /**
       * @param type The type to set.
       */
      public void setType(String type) {
            this.type = type;
      }

      public String toString() {
            return "Param : " + name + "Type:  " + type;
      }
      /**
       * @return Returns the typeAST.
       */
      public AST getTypeAST() {
            return typeAST;
      }
      /**
       * @param typeAST The typeAST to set.
       */
      public void setTypeAST(AST typeAST) {
            this.typeAST = typeAST;
      }
}
```

EzqlRow.java

```
-----------------------------------------------------
package ezql;

import java.util.HashMap;

/**
 * @author bbhatti
 */
public class EzqlRow extends EzqlType {

    private HashMap row;

    public EzqlRow() {
        this(new HashMap());
    }
```

```
    public EzqlRow(HashMap row) {
        this.row = row;
    }

    public HashMap getValue() {
        return this.row;
    }

    public String getType() {
        return EzqlType.ROW_TYPE;
    }

    public String toString() {
        return this.row.toString();
    }
}
```

EzqlRuntime.java

```
---------------------------------------------------
package ezql;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Stack;

import ezql.grammar.EzqlTreeParser;
import ezql.internal.EzqlInternalMethod;
import ezql.internal.EzqlMethodHelper;

import antlr.RecognitionException;
import antlr.collections.AST;

/**
 * @author bbhatti
 */
public class EzqlRuntime {

    HashMap methodMap = new HashMap();

    Stack callStack = new Stack();

    EzqlSymbolTable mainScope;

    public void initRuntime() {
        mainScope = new EzqlSymbolTable();
        callStack.push(mainScope);
    }

    public void putMethod(String name, ArrayList params, String
returnType,
            AST body) {

      //body.

        EzqlMethod meth = new EzqlMethod();
```

```
        meth.setName(name);
        meth.setBody(body);
        meth.setReturnType(returnType);
        meth.setArgs(params);
        methodMap.put(name, meth);
    }

    public EzqlType runMethod(EzqlTreeParser walker, String name,
ArrayList list)
            throws RecognitionException {

        EzqlMethod method = (EzqlMethod) methodMap.get(name);
        EzqlSymbolTable scope = null;
        EzqlType retVal = null;
        if (method != null) {
            scope = new EzqlSymbolTable(name);
            if (!validMethod (method, list, scope)) {
                ErrorHandler.handle("Argument mismatch for method " +
method.getName());
            }

            callStack.push(scope);
            AST ast = (AST) method.getBody();
            retVal = (EzqlType) walker.methodBody(ast);

            String declRet = method.getReturnType();
              if (!retVal.getType().equals(declRet)) {
                  ErrorHandler.handle("return type doesnt match
declared method " + method.getName());
              }
              scope = (EzqlSymbolTable) callStack.pop();
        } else {
            // CHECK FOR INTERNAL METHODS
            retVal = runInternalMethod(walker, name, list);
        }
        return retVal;
    }

    private EzqlType runInternalMethod(EzqlTreeParser walker, String
name, ArrayList list) {
        EzqlInternalMethod method = EzqlMethodHelper.getMethod(name);
        if (method == null) {
            ErrorHandler.handle("Call to an undeclared method: " +
name);
        }
        return method.runMethod(list);
    }

    public boolean validMethod (EzqlMethod method, ArrayList argsList,
EzqlSymbolTable scope) {
      if (method.getParams().size() != argsList.size()) {
            return false;
      }
      int argSize = method.getParams().size();
      for (int i = 0; i < argSize; i++) {
            EzqlParameter param =
(EzqlParameter)method.getParams().get(i);
```

```
//                System.out.println(param);
            EzqlType arg = (EzqlType)argsList.get(i);

            if (param.getType().equals(arg.getType())) {
                  scope.insert(param.getTypeAST(), param.getName(),
arg);
            } else {
                  return false;
            }
      }
      return true;
    }

    public void insert(AST type, String var, EzqlType value) {
        EzqlSymbolTable current = (EzqlSymbolTable) callStack.peek();
        current.insert(type, var, value);
    }

    public EzqlType lookup(String var) {
        EzqlSymbolTable current = (EzqlSymbolTable) callStack.peek();
        return current.lookup(var);
    }

    public void clear() {
        EzqlSymbolTable current = (EzqlSymbolTable) callStack.peek();
        current.clear();
    }

    public void print() {
        EzqlSymbolTable current = (EzqlSymbolTable) callStack.peek();
        //current.print();
    }

    public String toString() {
        return methodMap.toString();
    }
}


EzqlString.java
------------------------------------------------------
package ezql;

/**
 * @author iqbal
 */
public class EzqlString extends EzqlType {
      private String s;

      public EzqlString() {
      }

      public EzqlString(String s) {
            this.s = s;
      }

      public String getType() {
```

```
            return EzqlType.STRING_TYPE;
      }

      public String toString() {
            return this.s;
      }
}
```

EzqlSymbolTable.java

```
package ezql;

import java.util.*;

import antlr.collections.AST;

public class EzqlSymbolTable {

    private String scopeName;
      private HashMap varType;
      private HashMap varValue;

      public EzqlSymbolTable() {
          this("MainScope");
      }

      public EzqlSymbolTable(String scopeName) {
          this.scopeName = scopeName;
            varType = new HashMap();
            varValue = new HashMap();
      }

      private boolean DEBUG = false;

      public void insert (AST type, String var, EzqlType value) {
            if (DEBUG) {
                  System.out.println("IN SYMBOL TABLE");
                  System.out.println("TYPE: " + type);
                  System.out.println("VAR: " + var);
                  System.out.println("VAL TYPE: " + value.getType());
                  System.out.println("VALUE: " + value.toString());
            }

            //If the type is null, then
            // 1) it was already declared
            // 2) syntax error
            if (type == null) {
                  Object prevType = varType.get(var);
                  if (prevType == null) {
                        ErrorHandler.handle("Var " + var + " needs to
be declared before use ");
                  } else {
                        String prevEzqlType =
((EzqlType)varValue.get(var)).getType();
                        if (prevEzqlType != value.getType()) {
```

```
                              ErrorHandler.handle(type, "Illegal type
reassignment for Var: " + var);
                        } else {
                              varValue.put(var, value);
                        }
                  }
            } else {
                  //since the type is not null, make sure its the first
declaration
                  if (varType.get(var) != null) {
                        ErrorHandler.handle(type, "Var: " + var + "
Cannot be redeclared");
                  }

                  if (value.getType() == EzqlType.VOID_TYPE) {
                        ErrorHandler.handle(type, "Cannot assign void
for variable " +  var);
                        return;
                  }

                  // TODO
                  //before adding new types, make sure they are valid
                  //eg dont allow boolean b = "stringliteral";
                  if (type.toString().equals("string") &&
value.getType() != EzqlType.STRING_TYPE) {
                        ErrorHandler.handle(type, "Cannot assign string
for variable " +  var);
                  } else if (type.toString().equals("int")  &&
value.getType() != EzqlType.INT_TYPE) {
                        ErrorHandler.handle(type, "Cannot assign int
for variable " +  var);
                  } else if (type.toString().equals("boolean")  &&
value.getType() != EzqlType.BOOLEAN_TYPE) {
                        ErrorHandler.handle(type, "Cannot assign
boolean for variable " +  var);
                  } else if (type.toString().equals("list")  &&
value.getType() != EzqlType.LIST_TYPE) {
                        ErrorHandler.handle(type, "Cannot assign list
for variable " +  var);
                  } else if (type.toString().equals("row")  &&
value.getType() != EzqlType.ROW_TYPE) {
                        ErrorHandler.handle(type, "Cannot assign row
for variable " +  var);
                  } else {
                        //valid first time declaration, just add all
the info
                        varType.put(var,type);
                        varValue.put(var, value);

                  }
            }


      }

      public EzqlType lookup (String var) {
            return (EzqlType)varValue.get(var);
```

```
	}

	public void clear() {
		varType = new HashMap();
		varValue = new HashMap();
	}

	public void print () {
		Iterator iter = varType.keySet().iterator();
		while (iter.hasNext())	{
			String key = (String)iter.next();
			System.out.println("TABLE : " + varType.get(key) + "
" + key + " = " + varValue.get(key));
		}
	}

	public String toString() {
	    return this.scopeName;
	}

}
```

EzqlType.java

```
package ezql;

/**
 * @author bbhatti
 */
public abstract class EzqlType {
	public static final String INT_TYPE = "int";
	public static final String STRING_TYPE="string";
	public static final String BOOLEAN_TYPE="boolean";
	public static final String ROW_TYPE="row";
	public static final String LIST_TYPE="list";
	public static final String VOID_TYPE = "void";

	public boolean returned = false;

	public abstract String getType();

	public void setReturned(boolean ret) {
		this.returned = ret;
	}

	public boolean isReturned() {
		return this.returned;
	}
}
```

EzqlTypeHelper.java

```
package ezql;
```

```
/**
 * @author bbhatti
 */
public class EzqlTypeHelper {

	//public static EzqlSymbolTable table = new EzqlSymbolTable();

	public static boolean sameType(EzqlType a, EzqlType b) {
		return a.getType() == b.getType();
	}

	public static EzqlType plus(EzqlType a, EzqlType b) {
		if (sameType(a,b)) {
			if (a.getType() == EzqlType.INT_TYPE) {
				return new
EzqlInteger(((EzqlInteger)a).getValue() + ((EzqlInteger)b).getValue());
			} else if (a.getType() == EzqlType.STRING_TYPE) {
				return new EzqlString(a.toString() +
b.toString());
			}
			throw new RuntimeException("Invalid data types");
		} else {
			if (a.getType() == EzqlType.STRING_TYPE ||
b.getType() == EzqlType.STRING_TYPE) {
				return new EzqlString(a.toString() +
b.toString());
			}

		}
		throw new RuntimeException("Type mismatch");
	}

	public static EzqlType minus(EzqlType a, EzqlType b) {
		if (sameType(a,b)) {
			if (a.getType() == EzqlType.INT_TYPE) {
				return new
EzqlInteger(((EzqlInteger)a).getValue() - ((EzqlInteger)b).getValue());
			}
			throw new RuntimeException("Invalid data types");
		}
		throw new RuntimeException("Type mismatch");
	}

	public static EzqlType star(EzqlType a, EzqlType b) {
		if (sameType(a,b)) {
			if (a.getType() == EzqlType.INT_TYPE) {
				return new
EzqlInteger(((EzqlInteger)a).getValue() * ((EzqlInteger)b).getValue());
			}
			throw new RuntimeException("Invalid data types");
		}
		throw new RuntimeException("Type mismatch");
	}

	public static EzqlType div(EzqlType a, EzqlType b) {
		if (sameType(a,b)) {
```

```
                  if (a.getType() == EzqlType.INT_TYPE) {
                        return new
EzqlInteger(((EzqlInteger)a).getValue() / ((EzqlInteger)b).getValue());
                  }
                  throw new RuntimeException("Invalid data types");
            }
            throw new RuntimeException("Type mismatch");
      }

      public static EzqlType mod(EzqlType a, EzqlType b) {
            if (sameType(a,b)) {
                  if (a.getType() == EzqlType.INT_TYPE) {
                        return new
EzqlInteger(((EzqlInteger)a).getValue() % ((EzqlInteger)b).getValue());
                  }
                  throw new RuntimeException("Invalid data types");
            }
            throw new RuntimeException("Type mismatch");
      }

      public static EzqlType equal(EzqlType a, EzqlType b) {
            if (sameType(a,b)) {
                  if (a.getType() == EzqlType.INT_TYPE) {
                        return new
EzqlBoolean(((EzqlInteger)a).getValue() ==
((EzqlInteger)b).getValue());
                  } else if (a.getType() == EzqlType.STRING_TYPE) {
                        return new
EzqlBoolean(a.toString().equals(b.toString()));
                  } else if (a.getType() == EzqlType.BOOLEAN_TYPE) {
                        return new
EzqlBoolean(a.toString().equals(b.toString()));
                  }
                  throw new RuntimeException("Invalid data types");
            }
            throw new RuntimeException("Type mismatch");
      }

      public static EzqlType notEqual(EzqlType a, EzqlType b) {
            return negate(equal(a,b));
      }

      public static EzqlType ge(EzqlType a, EzqlType b) {
            if (sameType(a,b)) {
                  if (a.getType() == EzqlType.INT_TYPE) {
                        return new
EzqlBoolean(((EzqlInteger)a).getValue() >=
((EzqlInteger)b).getValue());
                  }
                  throw new RuntimeException("Invalid data types");
            }
            throw new RuntimeException("Type mismatch");
      }

      public static EzqlType gt(EzqlType a, EzqlType b) {
            if (sameType(a,b)) {
                  if (a.getType() == EzqlType.INT_TYPE) {
```

```
                        return new
EzqlBoolean(((EzqlInteger)a).getValue() > ((EzqlInteger)b).getValue());
                  }
                  throw new RuntimeException("Invalid data types");
            }
            throw new RuntimeException("Type mismatch");
      }

      public static EzqlType le(EzqlType a, EzqlType b) {
            if (sameType(a,b)) {
                  if (a.getType() == EzqlType.INT_TYPE) {
                        return new
EzqlBoolean(((EzqlInteger)a).getValue() <=
((EzqlInteger)b).getValue());
                  }
                  throw new RuntimeException("Invalid data types");
            }
            throw new RuntimeException("Type mismatch");
      }

      public static EzqlType lt(EzqlType a, EzqlType b) {
            if (sameType(a,b)) {
                  if (a.getType() == EzqlType.INT_TYPE) {
                        return new
EzqlBoolean(((EzqlInteger)a).getValue() < ((EzqlInteger)b).getValue());
                  }
                  throw new RuntimeException("Invalid data types");
            }
            throw new RuntimeException("Type mismatch");
      }

      public static EzqlType lor(EzqlType a, EzqlType b) {
            if (sameType(a,b)) {
                  if (a.getType() == EzqlType.BOOLEAN_TYPE) {
                        return new
EzqlBoolean(((EzqlBoolean)a).getValue() ||
((EzqlBoolean)b).getValue());
                  }
                  throw new RuntimeException("Invalid data types");
            }
            throw new RuntimeException("Type mismatch");
      }

      public static EzqlType land(EzqlType a, EzqlType b) {
            if (sameType(a,b)) {
                  if (a.getType() == EzqlType.BOOLEAN_TYPE) {
                        return new
EzqlBoolean(((EzqlBoolean)a).getValue() &&
((EzqlBoolean)b).getValue());
                  }
                  throw new RuntimeException("Invalid data types");
            }
            throw new RuntimeException("Type mismatch");
      }

      public static EzqlType negate(EzqlType a) {
            return new EzqlBoolean(!((EzqlBoolean)a).getValue());
```

```
      }
}
```

EzqlVoid.java

```
package ezql;

/**
 * @author iqbal
 *
 */
public class EzqlVoid extends EzqlType {

      public String getType() {
            return EzqlType.VOID_TYPE;
      }

      public String toString() {
            return "";
      }
}
```

LineNumberAST.java

```
package ezql;

import antlr.*;
import antlr.collections.AST;

/**
 * @author bbhatti
 */
public class LineNumberAST extends CommonAST
{
   private int lineNumber;

    public LineNumberAST() { }
    public LineNumberAST(Token tok) { super(tok); }
    public void initialize(AST t) {
        super.initialize(t);
        if (t.getClass().getName().compareTo("LineNumberAST") == 0)
        {
            LineNumberAST t2 = (LineNumberAST)t;
            lineNumber = t2.lineNumber;
        }
    }
    public void initialize(Token tok) {
        super.initialize(tok);
        lineNumber = tok.getLine();
    }

    public int getLineNumber()
    {
```

```
        return lineNumber;
    }
}
```

Main.java

```
-------------------------------------------------
package ezql;
import java.io.File;
import java.io.FileInputStream;
import java.util.Arrays;
import java.util.List;

import antlr.collections.AST;
import ezql.grammar.EzqlInitialTreeParser;
import ezql.grammar.EzqlLexer;
import ezql.grammar.EzqlParser;
import ezql.grammar.EzqlTreeParser;

class Main {
      public static void main(String[] args) {
            EzqlParser parser = null;
            try {
//            EzqlLexer lexer = new EzqlLexer(new
DataInputStream(System.in));
//            EzqlLexer lexer = new EzqlLexer(new StringReader("table
role as foo { }"));

                  List argList = Arrays.asList(args);
                  File srcFile = null;

                  if (!argList.isEmpty() && argList.get(0) != null) {
                        srcFile = new File((String)argList.get(0));
                  } else {
                        System.out.println("Using default file:
recursion.ezql");
                        srcFile = new File("sample" + File.separator +
"recursion.ezql");
                  }

                  EzqlLexer lexer = new EzqlLexer(new
FileInputStream(srcFile));
                  parser = new EzqlParser(lexer);
                  parser.compilationUnit();

            } catch (Exception e) {
                  System.out.println("\nException In Lexer/Parser: ");
                  System.out.println(e.getMessage());
                  e.printStackTrace();
            }

            try {
                  AST t = parser.getAST();
                  EzqlTreeParser tp = new EzqlTreeParser();
                  EzqlInitialTreeParser eitp = new
EzqlInitialTreeParser();
```

```
                EzqlRuntime runtime = eitp.start(t);
                tp.start(t, runtime);
                System.out.println("\nDone....");
            } catch (Exception e) {
                System.out.println("\nException In TreeWalker: ");
                System.out.println(e.getMessage());
                e.printStackTrace();
            }
      }
}


TestParser.java
-------------------------------------------------
package ezql;

import java.io.File;
import java.io.FileInputStream;
import java.util.Arrays;
import java.util.List;

import ezql.grammar.EzqlLexer;
import ezql.grammar.EzqlParser;

/**
 * @author bbhatti
 *
 */
public class TestParser {
      public static void main(String[] args) {
            EzqlParser parser = null;
            try {
                List argList = Arrays.asList(args);
                File srcFile = null;

                if (!argList.isEmpty() && argList.get(0) != null) {
                      srcFile = new File((String)argList.get(0));
                } else {
                      System.out.println("Using default file:
recursion.ezql");
                      srcFile = new File("sample" + File.separator +
"recursion.ezql");
                }

                System.out.println("Parsing : " +
srcFile.getAbsolutePath());
                EzqlLexer lexer = new EzqlLexer(new
FileInputStream(srcFile));
                parser = new EzqlParser(lexer);
                parser.compilationUnit();

                System.out.println("\nDone....\n");
            } catch (Exception e) {
                System.out.println("\nException In Parser: ");
                System.out.println(e.getMessage());
                e.printStackTrace();
            }
```

```
	}
}
```

TreeViewer.java

```
-----------------------------------------------------
package ezql;

/**
 * @author bbhatti
 */

import java.io.FileInputStream;

import ezql.grammar.EzqlLexer;
import ezql.grammar.EzqlParser;
import antlr.ASTFactory;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

public class TreeViewer {

	public static void main(String[] args) {
		if (args.length == 0) {
			System.out.println("Need file name");
			return;
		}

		try {
			EzqlLexer lexer = new EzqlLexer(new
FileInputStream(args[0]));
			EzqlParser parser = new EzqlParser(lexer);
			parser.compilationUnit();


			AST t = parser.getAST();
			CommonAST.setVerboseStringConversion(true,
parser.getTokenNames());
			ASTFactory factory = new ASTFactory();
			AST r = factory.create(0,"AST ROOT");
			r.setFirstChild(t);

			ASTFrame frame = new ASTFrame("Tree Viewer", r);
			frame.setSize(600, 600);
			frame.setVisible(true);
		} catch (Exception e) {
			System.out.println("Ezql ERROR: " + e.getMessage());
		}
	}
}
```

ezql.g

```
-----------------------------------------------------
```

```
// Authors : Bilal Bhatti
//           Iqbal Ahmad Syed

header { package ezql.grammar; }

class EzqlParser extends Parser;


options {
      k = 2;
      exportVocab = Ezql;
      buildAST = true;
}

tokens {
      WHILE; LOOP_BODY; LOOP_CONDITION;METHOD_CALL;PARAM_LIST;
      TABLE_DEF;TABLE_ALIAS_DEF;METHOD_DEF;TYPE;TABLE_BODY;ARGS;ARG;RET
URN_DEF;
      METHOD_BODY;
STATEMENT;BOOLEAN_VALUE;IF_STATEMENT;THEN_BLOCK;ELSE_BLOCK;
}

compilationUnit
      { getASTFactory().setASTNodeType("ezql.LineNumberAST"); }
      : (tableDefinition)
      ;

tableDefinition!
      : "table"^ ID ta:tableAliasClause LCURLY!
        tb:tableBody
        RCURLY!
        {#tableDefinition = #([TABLE_DEF,"TABLE_DEF"],ID,ta,tb);}
      ;

tableAliasClause
      : "as" ID
      {#tableAliasClause = #([TABLE_ALIAS_DEF,"TABLE_ALIAS_DEF"],ID);}
      ;

tableBody
      : (methodDefinition)*
      {#tableBody = #([TABLE_BODY,"TABLE_BODY"],tableBody);}
      ;

methodDefinition
      : (rs:returnSpec ID LPAREN! (argumentList)? RPAREN! LCURLY!
            methodBody
        RCURLY!
        )
        {#methodDefinition = #([METHOD_DEF,"METHOD_DEF"],rs);}
      ;

returnSpec
      : rt:returnTypes
        {#returnSpec = #([TYPE,"TYPE"],rt);}
      ;
```

```
returnTypes
	: (builtInTypes)
	;

builtInTypes
	: "int"
	| "string"
	| "float"
	| "boolean"
	| "row"
	| "list"
	;

argumentList
	: argumentSpec (COMMA! argumentSpec)*
	  {#argumentList = #([ARGS,"ARGS"],argumentList);}
	;

argumentSpec
	: a:builtInTypes ID
	  {#argumentSpec = #([ARG,"ARG"],argumentSpec);}
	;

returnStatement!
	: "return" (a:returnVals)? SEMI
	  {#returnStatement = #([RETURN_DEF,"RETURN_DEF"],a);}
	;

returnVals
	: (expression )
	;

methodBody
	: (statement)*
	  {#methodBody = #([METHOD_BODY,"METHOD_BODY"],methodBody);}
	;

statement
	:	(	(ifStatement
			| assignmentStatement
			| whileStatement
			| returnStatement
			)
		{#statement = #([STATEMENT,"STATEMENT"],statement);}
		)
	;

assignmentStatement
	: ((builtInTypes ID | ID) ASSIGN^)? expression SEMI!
	;

expressionElements
	: constantValues
	| ID
	| methodCall
	| LPAREN! expression RPAREN!
	;
```

```
constantValues
	: INTEGER
	| STRING_LITERAL
	| b:booleanValues {#b.setType(BOOLEAN_VALUE);}
	;

booleanValues
	: ("true" |"false")
	;

methodCall
	: (ID^ LPAREN! pl:parameterList! RPAREN!)
	  {#methodCall = #([METHOD_CALL,"METHOD_CALL"],methodCall,pl);}
	;

parameterList
	: (expression (COMMA! expression)*)?
	  {#parameterList = #([PARAM_LIST,"PARAM_LIST"],parameterList);}
	;

negationExpression
	: (LNOT^)? expressionElements
	;

signExpression
	: (PLUS^ | MINUS^)? negationExpression
	;

multiplyingExpression
	: signExpression ((STAR^ | DIV^ | MOD^) signExpression)*
	;

additionExpression
	: multiplyingExpression ((PLUS^ | MINUS^) multiplyingExpression)*
	;

relationalExpression
	: additionExpression ((EQUAL^ | NOT_EQUAL^ | LT^ | LE^ | GT^ |
GE^) additionExpression)*
	;

andExpression
	: relationalExpression (LAND^ relationalExpression)*
	;

expression
	: andExpression (LOR^ andExpression)*
	;

ifStatement
	: "if"! LPAREN! expression RPAREN! LCURLY!
	      thenBlock
	  RCURLY!
	  (elseBlock)?
	  {#ifStatement = #([IF_STATEMENT,"IF_STATEMENT"],ifStatement);}
```

```
      ;

thenBlock
      : (statement)*
        {#thenBlock = #([THEN_BLOCK,"THEN_BLOCK"],thenBlock);}
      ;

elseBlock
      : "else"! LCURLY!
            (statement)*
        RCURLY!
        {#elseBlock = #([ELSE_BLOCK,"ELSE_BLOCK"],elseBlock);}
      ;


whileStatement
      : "while"! LPAREN! loopCondition RPAREN! LCURLY!
          loopBody
        RCURLY!
        {#whileStatement = #([WHILE,"WHILE"],whileStatement);}

      ;

loopCondition
      : expression
        {#loopCondition =
#([LOOP_CONDITION,"LOOP_CONDITION"],loopCondition);}

      ;

loopBody
      : (statement)*
        {#loopBody = #([LOOP_BODY,"LOOP_BODY"],loopBody);}

      ;

class EzqlLexer extends Lexer;

options {
      charVocabulary = '\0'..'\377';
      testLiterals = false;
      k = 2;
}

ID
      options {testLiterals=true;}
      : ('a'..'z' | 'A'..'Z') ('a'..'z' | 'A'..'Z' | '0' .. '9' | '_')*
      ;

protected
DIGIT
      : ('0'..'9')
      ;

INTEGER
      : (DIGIT)+
      ;
```

```
STRING_LITERAL
      : '"'! ( '"' '"'! | ~('"'|'\n'|'\r'))* ( '"'!
      | // nothing -- write error message
      )
      ;

// OPERATORS
LPAREN                  :     '('         ;
RPAREN                  :     ')'         ;
LCURLY                  :     '{'         ;
RCURLY                  :     '}'         ;
COMMA             :     ','         ;
DOT              :   '.'       ;
ASSIGN                  :     '='         ;
EQUAL             :     "=="        ;
LNOT                    :     '!'         ;
DIV                     :     '/'         ;
PLUS                    :     '+'         ;
MINUS             :     '-'         ;
STAR                    :     '*'         ;
MOD                     :     '%'         ;
GE                      :     ">="        ;
GT                      :     ">"         ;
LE                      :     "<="        ;
LT                      :     '<'         ;
LOR                     :     "||"        ;
LAND                    :     "&&"        ;
SEMI                    :     ';'         ;
NOT_EQUAL               :     "!="        ;

COMMENTS
      : "//" (~('\n'|'\r'))* { $setType(Token.SKIP);}
      ;

// Whitespace -- ignored
WS    :     (     ' '
            |     '\t'
            |     '\f'
            |     (     options {generateAmbigWarnings=false;}
                  :     "\r\n"  // DOS
                  |     '\r'    // Macintosh
                  |     '\n'    // Unix
                  )
                  { newline(); }
            )+
            { _ttype = Token.SKIP; }
      ;



ezql.initial.walker.g
-----------------------------------------------------
// Authors : Bilal Bhatti
//           Iqbal Ahmad Syed

header { package ezql.grammar; }
```

```
{
      import ezql.*;
      import java.util.ArrayList;
      import java.util.HashMap;
}

class EzqlInitialTreeParser extends TreeParser;

options {
      ASTLabelType = "LineNumberAST";
      importVocab = Ezql;
}

start returns [EzqlRuntime runtime = new EzqlRuntime()]
{
      runtime.initRuntime();
      getASTFactory().setASTNodeType("ezql.LineNumberAST");
      ArrayList argList;
}
      : #(TABLE_DEF id:ID tac:. tableBody[runtime])
      ;

tableBody [EzqlRuntime runtime]
      : #(TABLE_BODY (md:methodDefinition[runtime])*)
      ;

methodDefinition [EzqlRuntime runtime]
      {
            ArrayList al=null;

      }
      : #(METHOD_DEF rt:returnTypes id:ID (al=argumentList)? (mb:.)?)
      {
            if (!id.getText().equals("main")) {

      runtime.putMethod(id.getText(),al,rt.getFirstChild().getText(),
#mb);
            }
      }
      ;

returnTypes
      : #(TYPE (builtInTypes))
      ;

builtInTypes
      : "int"
      | "string"
      | "float"
      | "boolean"
      | "row"
      | "list"
      ;

argumentList returns [ArrayList al = null]
      {
```

```
            al = new ArrayList();
      }

      : #(ARGS (ar:.(
            {
                  argument(#ar, al);
            }
                        )
                  )*
            )
      ;

argument [ArrayList list]
      : #(ARG (bt:builtInTypes id:ID))
      {
                  EzqlParameter param = new EzqlParameter();
                  param.setType(bt.getText());
                  param.setName(id.getText());
                  param.setTypeAST(#bt);
                  list.add(param);
      }
      ;



ezql.tree.g
------------------------------------------------------
// Authors : Bilal Bhatti
//           Iqbal Ahmad Syed

header { package ezql.grammar; }

{
      import ezql.*;
      import ezql.internal.*;

      import java.util.ArrayList;
      import java.util.HashMap;
}

class EzqlTreeParser extends TreeParser;

options {
      ASTLabelType = "LineNumberAST";
      importVocab = Ezql;
}

{
      EzqlRuntime runtime = null;
}

start [EzqlRuntime rt]
{
      runtime = rt;
      getASTFactory().setASTNodeType("ezql.LineNumberAST");
}
      : #(TABLE_DEF id:ID tableAliasClause tableBody)
```

```
	;


tableAliasClause
	: #(TABLE_ALIAS_DEF id:ID)
	;

tableBody
	: #(TABLE_BODY (md:methodDefinition)*)
	;

methodDefinition
	: #(METHOD_DEF rt:returnTypes id:ID (al:argumentList)? (mb:.)? )
	{
		if (id.getText().equals("main") && al == null) {
			methodBody(#mb);
		}
	}
	;

returnTypes
	: #(TYPE builtInTypes)
	;

builtInTypes
	: "int"
	| "string"
	| "float"
	| "boolean"
	| "row"
	| "list"
	;

argumentList
	: #(ARGS (argument)*)
	;

argument
	: #(ARG (bt:builtInTypes id:ID))
	;

returnStatement returns [EzqlType res=null]
	: #(RETURN_DEF res=expr { if (res != null) {
res.setReturned(true);}})
	;

methodBody returns [EzqlType res=null]
	: #(METHOD_BODY (res=statement

	{
		if (res != null && res.isReturned()) {
			break;
		}
	}
	)*)
	;
```

```
statement returns [EzqlType res=null]
      : #(STATEMENT (assignmentStatement
            | res = expr
            | res = ifStatement
            | res = whileStatement
            | res = returnStatement
                  )
            )
      ;

ifStatement returns [EzqlType res=null]
      {
            EzqlType ifBool;
      }
      : #(IF_STATEMENT ifBool=expr thenp:. (elsep:.)?)
      {
            if (ifBool.getType() != EzqlType.BOOLEAN_TYPE) {
                  ErrorHandler.handle (#IF_STATEMENT, "A boolean
statement is needed for an if statement");
            }
            if ( ((EzqlBoolean)ifBool).getValue() ) {
                  res = ifstatements(#thenp);
            } else {
                  if (#elsep != null) {
                        res = ifstatements(#elsep);
                  }
            }
      }
      ;

ifstatements returns [EzqlType res=null]
      : #(THEN_BLOCK (res=statement)*)
      | #(ELSE_BLOCK (res=statement)*)
      ;

whileStatement returns [EzqlType res=null]
      {
            EzqlType ifBool;
      }
      : #(WHILE ifBool=l:loopCondition s:.)
      {
            if (ifBool.getType() != EzqlType.BOOLEAN_TYPE) {
                  ErrorHandler.handle (#WHILE, "Invalid boolean
condition for while statement");
            }
            while ( ((EzqlBoolean)ifBool).getValue() ) {
                  res = loopBody(#s);
                  if (res != null && res.isReturned()) {
                        break;
                  }
                  ifBool = loopCondition(#l);
            }
      }
      ;

loopCondition returns [EzqlType retVal=null]
      : #(LOOP_CONDITION retVal = expr)
```

```
      ;

loopBody returns [EzqlType res=null]
      : #(LOOP_BODY (res = statement)*)
      ;

assignmentStatement
      { EzqlType a; }
      : #(ASSIGN (t:builtInTypes id:ID) a=expr)
      {
            runtime.insert(t,id.getText(),a);
      }
      ;

expr returns [EzqlType retVal=null]
      {
            EzqlType a,b;
            ArrayList pal;
      }
      : #(PLUS  a=expr b=expr) {retVal = EzqlTypeHelper.plus(a, b);}
      | #(MINUS a=expr b=expr) {retVal = EzqlTypeHelper.minus(a, b);}
      | #(STAR  a=expr b=expr) {retVal = EzqlTypeHelper.star(a, b);}
      | #(DIV   a=expr b=expr) {retVal = EzqlTypeHelper.div(a, b);}
      | #(MOD   a=expr b=expr) {retVal = EzqlTypeHelper.mod(a, b);}
      | #(EQUAL a=expr b=expr) {retVal = EzqlTypeHelper.equal(a, b);}
      | #(GE    a=expr b=expr) {retVal = EzqlTypeHelper.ge(a, b);}
      | #(GT    a=expr b=expr) {retVal = EzqlTypeHelper.gt(a, b);}
      | #(LE    a=expr b=expr) {retVal = EzqlTypeHelper.le(a, b);}
      | #(LT    a=expr b=expr) {retVal = EzqlTypeHelper.lt(a, b);}
      | #(LOR   a=expr b=expr) {retVal = EzqlTypeHelper.lor(a, b);}
      | #(LAND  a=expr b=expr) {retVal = EzqlTypeHelper.land(a, b);}
      | #(LNOT  a=expr) {retVal = EzqlTypeHelper.negate(a); }
      | #(NOT_EQUAL a=expr b=expr) {retVal =
EzqlTypeHelper.notEqual(a,b); }

      | i:INTEGER {retVal = new EzqlInteger(i.getText()); }
      | bool:BOOLEAN_VALUE {retVal = new EzqlBoolean(bool.getText()); }
      | s:STRING_LITERAL {retVal = new EzqlString(s.getText()); }
      | id:ID {retVal=runtime.lookup(id.getText());}
      | #(mc:METHOD_CALL m:ID pal=paramArrayList)
            {
                  retVal = runtime.runMethod(this, m.getText(), pal);
            }
  ;

paramArrayList returns [ArrayList al = new ArrayList()]
{ EzqlType res=null; }
      : #(PARAM_LIST
            (res=expr {al.add(res);}
        )*)
      ;



DBService.java
--------------------------------------------------
package ezql.internal;
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;

/**
 * @author bbhatti
 */
public class DBService {

	public static Connection getConnection() {
		return getConnection(EzqlProperties.props);
	}

	public static Connection getConnection(Properties props) {
		String driver = props.getProperty("driver");
		String url = props.getProperty("url");
		String database = props.getProperty("database");
		String user = props.getProperty("user");
		String password = props.getProperty("password");

		Connection con = null;
		try {
			Class.forName(driver);
			con = DriverManager.getConnection(url, user,
password);
		} catch (Exception ex) {
			ex.printStackTrace();
			try {
				if (con != null)
					con.close();
			} catch (Exception e) {
			}
		}
		return con;
	}

	public static void closeConnection(Connection con) {
		try {
			if (con != null)
				con.close();
		} catch (Exception e) {
		}
	}

	public static void cleanup(ResultSet rs, Statement stmt,
Connection con) {
		try {
			rs.close();
			stmt.close();
			con.close();
		} catch (Exception e) {
		}
	}
```

```
      public static void main(String[] args) {
            //System.out.println(EzqlProperties.props.toString());
            Connection con =  null;
            Statement stmt =  null;
            ResultSet rs =  null;
            try {
                  con = DBService.getConnection();
                  stmt = con.createStatement();
                  rs = stmt.executeQuery("select Name from Users");
                  while (rs.next()) {
                        System.out.println("user: " +
rs.getString("Name"));
                  }
            } catch (Exception e) {
                  e.printStackTrace();
            } finally {
                  DBService.cleanup(rs, stmt, con);
            }
      }
}
```

EzqlColumnValue.java

```
package ezql.internal;

import java.util.ArrayList;

import ezql.EzqlException;
import ezql.EzqlRow;
import ezql.EzqlString;
import ezql.EzqlType;

/**
 * @author bbhatti
 */
public class EzqlColumnValue implements EzqlInternalMethod {
    public EzqlType runMethod(ArrayList args) throws EzqlException {
        EzqlRow row = (EzqlRow)args.get(0);
        EzqlString columnName = (EzqlString)args.get(1);
        return new
EzqlString((String)row.getValue().get(columnName.toString()));
    }
}
```

EzqlInternalMethod.java

```
package ezql.internal;

import java.util.ArrayList;

import ezql.EzqlException;
import ezql.EzqlType;

/**
```

```
 * @author bbhatti
 */
public interface EzqlInternalMethod {
    public EzqlType runMethod(ArrayList args) throws EzqlException;
}
```

EzqlMerge.java

```
-----------------------------------------------------
package ezql.internal;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.StringWriter;
import java.io.Writer;
import java.util.ArrayList;

import org.apache.velocity.Template;
import org.apache.velocity.VelocityContext;
import org.apache.velocity.app.VelocityEngine;

import ezql.ErrorHandler;
import ezql.EzqlException;
import ezql.EzqlList;
import ezql.EzqlString;
import ezql.EzqlType;
import ezql.EzqlVoid;

/**
 * @author bbhatti
 */
public class EzqlMerge implements EzqlInternalMethod {

    public EzqlType runMethod(ArrayList args) throws EzqlException {
        VelocityEngine ve = new VelocityEngine();
        Writer output = null;

        String workingDir = (String)
EzqlProperties.props.get("workingdir");

        EzqlList list = (EzqlList) args.get(0);
        String template = "" + (EzqlString) args.get(1);
        String outputFileName = ((EzqlString) args.get(2)).toString();

        if (list != null && template != null && outputFileName != null)
{
                try {
                    ve.init("velocity.properties");
                    Template t = ve.getTemplate(template);
                    VelocityContext context = new VelocityContext();
                    context.put("list", list.getValue());
                    StringWriter writer = new StringWriter();
                    t.merge(context, writer);

                    File aFile = new File(outputFileName);
```

```
                    output = new BufferedWriter(new FileWriter(aFile));
                    output.write(writer.toString());
                } catch (Exception e) {
                    e.printStackTrace();
                } finally {
                    try {
                        output.close();
                    } catch (Exception e) {}
                }
        } else {
            ErrorHandler.handle("invalid parameters");
        }
        return new EzqlVoid();
    }
}
```

EzqlMethodHelper.java

```
package ezql.internal;

import java.util.HashMap;


/**
 * @author bbhatti
 */
public class EzqlMethodHelper {
    private static HashMap methodMap = new HashMap();

    static {
        methodMap.put("runquery", new EzqlRunQuery());
        methodMap.put("rowcount", new EzqlRowCount());
        methodMap.put("print", new EzqlPrint());
        methodMap.put("merge", new EzqlMerge());
    }

    public static EzqlInternalMethod getMethod(String name) {
        return (EzqlInternalMethod)methodMap.get(name);
    }
}
```

EzqlPrint.java

```
package ezql.internal;

import java.util.ArrayList;
import java.util.Iterator;

import ezql.EzqlType;
import ezql.EzqlVoid;

/**
 * @author iqbal
 */
```

```
public class EzqlPrint implements EzqlInternalMethod {

      public EzqlType runMethod(ArrayList args) {
            for (Iterator iter = args.iterator(); iter.hasNext();) {
                  EzqlType element = (EzqlType) iter.next();
                  System.out.print(element.toString());
            }
            System.out.println();
            return new EzqlVoid();
    }
}
```

EzqlProperties.java

```
-----------------------------------------------------
package ezql.internal;

import java.io.File;
import java.io.FileInputStream;
import java.util.Properties;

/**
 * @author bbhatti
 */
public class EzqlProperties {
      public static Properties props;
      static {
            try {
                  FileInputStream fis = new FileInputStream(new
File("ezql.properties"));
                  props = new Properties();
                  props.load(fis);
            } catch (Exception e) {
                  props = null;
                  e.printStackTrace();
            }
      }
}
```

EzqlQueryHelper.java

```
-----------------------------------------------------
package ezql.internal;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;


/**
 * @author iqbal
 */
public class EzqlQueryHelper {
```

```
      public static ArrayList runquery (String query) {
            Connection conn = null;
            Statement stat = null;
            ResultSet rs = null;
            ArrayList list = new ArrayList();

            try {
                  conn = DBService.getConnection();
                  stat = conn.createStatement();
                  rs = stat.executeQuery(query);
                  int columns = rs.getMetaData().getColumnCount();
                  while (rs.next()) {
                        HashMap map = new HashMap();
                        for (int i = 1; i <= columns; i++) {
                              String key =
rs.getMetaData().getColumnName(i);
                              String val = rs.getString(i);
                              map.put(key, val);
                        }
                        list.add(map);
                  }

            } catch (Exception e) {
                  e.printStackTrace();
            } finally {
                  DBService.cleanup(rs, stat, conn);
            }
            return list;
      }
}
```

EzqlRowCount.java

```
---------------------------------------------------
package ezql.internal;

import java.util.ArrayList;

import ezql.EzqlException;
import ezql.EzqlInteger;
import ezql.EzqlList;
import ezql.EzqlType;

/**
 * @author bbhatti
 */
public class EzqlRowCount implements EzqlInternalMethod {
    public EzqlType runMethod(ArrayList args) throws EzqlException {
        EzqlList list = (EzqlList)args.get(0);
        return new EzqlInteger(list.getValue().size());
    }
}
```

EzqlRunQuery.java

```
package ezql.internal;

import java.util.ArrayList;

import ezql.EzqlList;
import ezql.EzqlType;

/**
 * @author bbhatti
 */
public class EzqlRunQuery implements EzqlInternalMethod {
    public EzqlType runMethod(ArrayList args) {
        EzqlType param = (EzqlType)args.get(0);
        String query = param.toString();
        return new EzqlList(EzqlQueryHelper.runquery(query));
    }

}
```

ColumnValueTest.java

```
package ezql.test;

import java.util.ArrayList;
import java.util.HashMap;

import junit.framework.TestCase;
import ezql.EzqlList;
import ezql.EzqlRow;
import ezql.EzqlString;
import ezql.internal.EzqlColumnValue;
import ezql.internal.EzqlInternalMethod;
import ezql.internal.EzqlQueryHelper;

/**
 * @author bbhatti
 *
 */
public class ColumnValueTest extends TestCase {

    /**
     * Constructor for ColumnValueTest.
     *
     * @param arg0
     */
    public ColumnValueTest(String arg0) {
        super(arg0);
    }

    public void testColumnValue() {
        ArrayList args = new ArrayList();
        EzqlList ql = new EzqlList(EzqlQueryHelper.runquery("Select
name from users"));
```

```
        EzqlRow er = new EzqlRow((HashMap) ql.getValue().get(1));
        args.add(er);
        args.add(new EzqlString("name"));

        EzqlInternalMethod method = new EzqlColumnValue();
        EzqlString name = (EzqlString) method.runMethod(args);
        //        System.out.println("name: " + name);
        assertNotNull(name);
    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run(ColumnValueTest.class);
    }
}


DatabaseTest.java
-------------------------------------------------
package ezql.test;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;

import junit.framework.TestCase;
import ezql.internal.DBService;

/**
 * @author bbhatti
 */
public class DatabaseTest extends TestCase {

    /**
     * Constructor for DatabaseTest.
     *
     * @param arg
     */
    public DatabaseTest(String s) {
        super(s);
    }

    /*
     * @see TestCase#setUp()
     */
    protected void setUp() throws Exception {
        super.setUp();
    }

    public void testConnection() {
        //System.out.println(EzqlProperties.props.toString());
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            con = DBService.getConnection();
            assertNotNull(con);
        } catch (Exception e) {
```

```
            e.printStackTrace();
        } finally {
            DBService.cleanup(rs, stmt, con);
        }
    }

    public void testQuery() {
        //System.out.println(EzqlProperties.props.toString());
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            con = DBService.getConnection();
            stmt = con.createStatement();
            rs = stmt.executeQuery("select Name from Users");
            assertTrue(rs.next());
            //System.out.println("'select name from users' returned
            // successfully");
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            DBService.cleanup(rs, stmt, con);
        }
    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run(DatabaseTest.class);
    }
}
```

FeatureTestSuite.java

```
-----------------------------------------------------
package ezql.test;

import junit.framework.TestSuite;

/**
 * @author bbhatti
 */
public class FeatureTestSuite extends TestSuite {

    public static FeatureTestSuite suite() {
        FeatureTestSuite suite = new FeatureTestSuite();
        suite.addTest(new TestSuite(DatabaseTest.class));

        // BUILTIN FUNCTION LIBRARY TEST
        suite.addTest(new MergeTest("testMerge"));
        suite.addTest(new RunQueryTest("testRunQuery"));
        suite.addTest(new RowCountTest("testRowCount"));
        suite.addTest(new ColumnValueTest("testColumnValue"));
        suite.addTest(new PrintTest("testPrint"));

        return suite;
    }

    public static void main(String[] args) {
```

```
        junit.textui.TestRunner.run(FeatureTestSuite.suite());
    }
}
```

MainTestSuite.java

```
-----------------------------------------------------
package ezql.test;

import junit.framework.TestResult;
import junit.framework.TestSuite;

/**
 * @author bbhatti
 */
public class MainTestSuite extends TestSuite {

    public static MainTestSuite suite() {
        MainTestSuite suite = new MainTestSuite();
        suite.addTest(FeatureTestSuite.suite());
        suite.addTest(new TestSuite(TypeEngineTest.class));
        suite.addTest(new SymbolTableTest("testSymbolTable"));
        return suite;
    }

    public static void main(String[] args) {
        TestResult result =
junit.textui.TestRunner.run(MainTestSuite.suite());
        System.out.println("JUnit test cases completed with following
results: ");
        System.out.println("Total test count       : " +
result.runCount());
        System.out.println("Successfull test count : "+
(result.runCount() - result.failureCount()));
        System.out.println("Error count            : " +
result.errorCount());
        System.out.println("Test failure count     : " +
result.failureCount());
    }
}
```

MergeTest.java

```
-----------------------------------------------------
package ezql.test;

import java.util.ArrayList;

import junit.framework.TestCase;
import ezql.EzqlList;
import ezql.EzqlString;
import ezql.internal.EzqlInternalMethod;
import ezql.internal.EzqlMerge;
import ezql.internal.EzqlQueryHelper;

/**
 * @author bbhatti
```

```
 */
public class MergeTest extends TestCase {

    public MergeTest(String s) {
        super(s);
    }

    public void testMerge() {
        ArrayList args = new ArrayList();
        args.add(new EzqlList(EzqlQueryHelper
                .runquery("Select name from users")));
        args.add(new EzqlString("ezql.vm"));
        args.add(new EzqlString("ezql.rpt"));
        EzqlInternalMethod method = new EzqlMerge();
        assertNotNull(method.runMethod(args));
    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run(MergeTest.class);
    }
}


PrintTest.java
-------------------------------------------------------
package ezql.test;

import java.util.ArrayList;

import junit.framework.TestCase;
import ezql.EzqlString;
import ezql.EzqlType;
import ezql.internal.EzqlPrint;

/**
 * @author iqbal
 *
 */
public class PrintTest extends TestCase {

    /**
     *
     */
    public PrintTest() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @param arg0
     */
    public PrintTest(String arg0) {
        super(arg0);
        // TODO Auto-generated constructor stub
    }

    public void testPrint() {
```

```
        EzqlPrint print = new EzqlPrint();
        EzqlType type = null;
        try {
            print.runMethod(null);
        } catch (Exception e) {
            //null args should generate an error
            assertNotNull(e);
        }
        ArrayList list = new ArrayList();
        list.add(new EzqlString("THIS IS A TEST STRING"));

        type = print.runMethod(list);
        assertNotNull(type);

    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run(PrintTest.class);
    }

}


RowCountTest.java
-----------------------------------------------------
package ezql.test;

import java.util.ArrayList;

import junit.framework.TestCase;
import ezql.EzqlInteger;
import ezql.EzqlList;
import ezql.internal.EzqlInternalMethod;
import ezql.internal.EzqlQueryHelper;
import ezql.internal.EzqlRowCount;

/**
 * @author bbhatti
 */
public class RowCountTest extends TestCase {

    /**
     * Constructor for TestRowCount.
     *
     * @param arg0
     */
    public RowCountTest(String arg0) {
        super(arg0);
    }

    public void testRowCount() {
        ArrayList args = new ArrayList();
        args.add(new EzqlList(EzqlQueryHelper.runquery("Select name
from users")));
        EzqlInternalMethod method = new EzqlRowCount();
        assertEquals(((EzqlInteger) method.runMethod(args)).getValue(),
4);
```

```
    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run(RowCountTest.class);
    }
}
```

RunQueryTest.java

```
package ezql.test;

import junit.framework.TestCase;
import ezql.EzqlList;
import ezql.internal.EzqlQueryHelper;

/**
 * @author bbhatti
 */
public class RunQueryTest extends TestCase {

    public RunQueryTest(String s) {
        super(s);
    }

    public void testRunQuery() {
        EzqlList lst = new EzqlList(EzqlQueryHelper
                .runquery("select name from users"));
        assertNotNull(lst);
    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run(RunQueryTest.class);
    }
}
```

SymbolTableTest.java

```
package ezql.test;

import junit.framework.TestCase;
import antlr.collections.AST;
import ezql.EzqlInteger;
import ezql.EzqlString;
import ezql.EzqlSymbolTable;
import ezql.EzqlType;
import ezql.LineNumberAST;

/**
 * @author iqbal
 *
 */
public class SymbolTableTest extends TestCase {

    /**
     * @param arg0
```

```
     */
    public SymbolTableTest(String arg0) {
        super(arg0);
        // TODO Auto-generated constructor stub
    }

    public void testSymbolTable() {
        EzqlSymbolTable table = new EzqlSymbolTable();
        try {
            table.insert(null, "x", new EzqlInteger(5));
        } catch (RuntimeException re) {
            //this should be not null, the above call will throw an
exception
            assertNotNull(re);
        }

        AST ast = new LineNumberAST();
        ast.setText("string");

        table.insert(ast, "s", new EzqlString("HELLOWORLD"));
        EzqlType type = table.lookup("s");
        assertNotNull(type);
        assertEquals(type.toString(), "HELLOWORLD");

        table.insert(null, "s", new EzqlString("HELLOWORLD AGAIN"));
        type = table.lookup("s");
        //the new value should be in the table
        assertEquals(type.toString(), "HELLOWORLD AGAIN");

        type = table.lookup("y");
        //should be null, since it doesnt exist
        assertNull(type);

        String scopeName = table.toString();
        assertEquals(scopeName, "MainScope");
    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run(SymbolTableTest.class);
    }

}
```

TypeEngineTest.java

```
-----------------------------------------------------
package ezql.test;

import java.util.ArrayList;

import junit.framework.TestCase;
import ezql.EzqlBoolean;
import ezql.EzqlInteger;
import ezql.EzqlList;
import ezql.EzqlString;
import ezql.EzqlTypeHelper;
```

```
/**
 * @author bbhatti
 */
public class TypeEngineTest extends TestCase {

    private EzqlList lst;

    private EzqlString str;

    private EzqlInteger int_55;

    private EzqlInteger int_110;

    private EzqlBoolean bool_true;

    private EzqlBoolean bool_false;

    /**
     * Constructor for TypeCheckingTest.
     *
     * @param arg0
     */
    public TypeEngineTest(String arg0) {
        super(arg0);
    }

    /*
     * @see TestCase#setUp()
     */
    protected void setUp() throws Exception {
        super.setUp();
        lst = new EzqlList(new ArrayList());
        str = new EzqlString("test ezql str");
        int_55 = new EzqlInteger(55);
        int_110 = new EzqlInteger(110);
        bool_true = new EzqlBoolean(true);
        bool_false = new EzqlBoolean(false);
    }

    public void testSameType() {
        assertFalse(EzqlTypeHelper.sameType(lst, str));
        assertFalse(EzqlTypeHelper.sameType(lst, int_55));
        assertFalse(EzqlTypeHelper.sameType(lst, bool_true));
    }

    public void testPlus() {
        EzqlInteger resi = (EzqlInteger) EzqlTypeHelper.plus(int_55,
int_55);
        assertEquals(resi.getValue(), 110);

        EzqlString ress = (EzqlString) EzqlTypeHelper.plus(str,
int_55);
        assertTrue(ress.toString().equals("test ezql str55"));
    }

    public void testMinus() {
```

```
        EzqlInteger resi = (EzqlInteger) EzqlTypeHelper.minus(int_110,
int_55);
        assertEquals(resi.getValue(), 55);
    }

    public void testStar() {
        EzqlInteger resi = (EzqlInteger) EzqlTypeHelper.star(
                new EzqlInteger(2), int_55);
        assertEquals(resi.getValue(), 110);
    }

    public void testDiv() {
        EzqlInteger resi = (EzqlInteger) EzqlTypeHelper.div(int_110,
                new EzqlInteger(2));
        assertEquals(resi.getValue(), 55);
    }

    public void testMod() {
        EzqlInteger resi = (EzqlInteger) EzqlTypeHelper.mod(int_110,
                new EzqlInteger(50));
        assertEquals(resi.getValue(), 10);
    }

    public void testEqual() {
        EzqlBoolean resb = (EzqlBoolean) EzqlTypeHelper.equal(int_55,
int_55);
        assertTrue(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.equal(int_55, int_110);
        assertFalse(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.equal(str, new EzqlString(
                "test ezql str"));
        assertTrue(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.equal(bool_true,
bool_true);
        assertTrue(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.equal(bool_true,
bool_false);
        assertFalse(resb.getValue());
    }

    public void testNotEqual() {
        EzqlBoolean resb = (EzqlBoolean) EzqlTypeHelper
                .notEqual(int_55, int_55);
        assertFalse(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.notEqual(int_55, int_110);
        assertTrue(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.notEqual(str, new
EzqlString(
                "test ezql str"));
        assertFalse(resb.getValue());
```

```
        resb = (EzqlBoolean) EzqlTypeHelper.notEqual(bool_true,
bool_true);
        assertFalse(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.notEqual(bool_true,
bool_false);
        assertTrue(resb.getValue());
    }

    public void testGe() {
        EzqlBoolean resb = (EzqlBoolean) EzqlTypeHelper.ge(int_55,
int_55);
        assertTrue(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.ge(int_110, int_55);
        assertTrue(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.ge(int_55, int_110);
        assertFalse(resb.getValue());
    }

    public void testGt() {
        EzqlBoolean resb = (EzqlBoolean) EzqlTypeHelper.gt(int_55,
int_55);
        assertFalse(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.gt(int_110, int_55);
        assertTrue(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.gt(int_55, int_110);
        assertFalse(resb.getValue());
    }

    public void testLe() {
        EzqlBoolean resb = (EzqlBoolean) EzqlTypeHelper.le(int_55,
int_55);
        assertTrue(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.le(int_110, int_55);
        assertFalse(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.le(int_55, int_110);
        assertTrue(resb.getValue());
    }

    public void testLt() {
        EzqlBoolean resb = (EzqlBoolean) EzqlTypeHelper.lt(int_55,
int_55);
        assertFalse(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.lt(int_110, int_55);
        assertFalse(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.lt(int_55, int_110);
        assertTrue(resb.getValue());
    }
```

```
    public void testLor() {
        EzqlBoolean resb = (EzqlBoolean) EzqlTypeHelper.lor(bool_true,
                bool_false);
        assertTrue(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.lor(bool_true, bool_true);
        assertTrue(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.lor(bool_false,
bool_false);
        assertFalse(resb.getValue());
    }

    public void testLand() {
        EzqlBoolean resb = (EzqlBoolean) EzqlTypeHelper.land(bool_true,
                bool_false);
        assertFalse(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.land(bool_true, bool_true);
        assertTrue(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.land(bool_false,
bool_false);
        assertFalse(resb.getValue());
    }

    public void testNegate() {
        EzqlBoolean resb = (EzqlBoolean)
EzqlTypeHelper.negate(bool_true);
        assertFalse(resb.getValue());

        resb = (EzqlBoolean) EzqlTypeHelper.negate(bool_false);
        assertTrue(resb.getValue());
    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run(TypeEngineTest.class);
    }
}
```

build.xml

```
<project name="calc" default="run" basedir="./">

      <property name="src.dir" value="src"/>
      <property name="build.dir" value="bin"/>
      <property name="lib.dir" value="lib"/>
      <property name="antlrjar" value="${lib.dir}/antlr.jar"/>
      <property name="grammar.dir" value="${src.dir}/ezql/grammar"/>
      <property name="grammar.file" value="${grammar.dir}/ezql.g"/>

      <path id="project.class.path">
            <pathelement location="${antlrjar}"/>
```

```
            <pathelement path="${build.dir}"/>
      </path>

      <target name="compile" depends="grammar, prepare">
            <javac debug="on" srcdir="${src.dir}"
destdir="${build.dir}">
                  <classpath refid="project.class.path"/>
            </javac>
      </target>

      <target name="run">
            <java classname="ezql.Main">
                  <classpath refid="project.class.path"/>
            </java>
      </target>

      <target name="tree" depends="compile">
            <java classname="calc.CalcTreeViewer">
                  <classpath refid="project.class.path"/>
            </java>
      </target>

      <target name="clean">
            <delete dir="${build.dir}"/>
      </target>

      <target name="prepare" depends="clean">
            <mkdir dir="${build.dir}"/>
      </target>

      <target name="grammar">
            <delete>
                  <fileset dir="${grammar.dir}">
                        <include name="*.java"/>
                        <include name="*.txt"/>
                  </fileset>
            </delete>
            <antlr target="${grammar.file}"
outputdirectory="${grammar.dir}"/>
      </target>
</project>


runTV.sh
-----------------------------------
java ezql.TreeViewer sample/query.ezql



runTV.bat

java -classpath
%CLASSPATH%;C:\eclipse\workspace\ezql\bin;.;C:\dev\antlr-
2.7.4\antlr.jar ezql.TreeViewer sample/query.ezql
```

```
runezql.sh
-----------------------------------
java -classpath ./bin:./lib/antlr.jar:./lib/velocity-dep-
1.4.jar:./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.Main $1

runtests.sh
-----------------------------------

java -classpath ./bin:./lib/antlr.jar:./lib/velocity-dep-
1.4.jar:./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/recursion.ezql
java -classpath ./bin:./lib/antlr.jar:./lib/velocity-dep-
1.4.jar:./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/query.ezql
java -classpath ./bin:./lib/antlr.jar:./lib/velocity-dep-
1.4.jar:./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/merge.ezql
java -classpath ./bin:./lib/antlr.jar:./lib/velocity-dep-
1.4.jar:./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/table.ezql
java -classpath ./bin:./lib/antlr.jar:./lib/velocity-dep-
1.4.jar:./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/newtable.ezql

java -classpath
./junit3.8.1/junit.jar:./bin:./lib/antlr.jar:./lib/velocity-dep-
1.4.jar:./lib/mysql-connector-java-3.0.9-stable-bin.jar
ezql.test.MainTestSuite

./runezql.sh sample/recursion.ezql
./runezql.sh sample/query.ezql
./runezql.sh sample/merge.ezql
./runezql.sh sample/table.ezql
./runezql.sh sample/newtable.ezql


runtests.bat
--------------------------------------

java -classpath ./bin;./lib/antlr.jar;./lib/velocity-dep-
1.4.jar;./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/recursion.ezql
java -classpath ./bin;./lib/antlr.jar;./lib/velocity-dep-
1.4.jar;./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/query.ezql
java -classpath ./bin;./lib/antlr.jar;./lib/velocity-dep-
1.4.jar;./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/merge.ezql
java -classpath ./bin;./lib/antlr.jar;./lib/velocity-dep-
1.4.jar;./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/table.ezql
java -classpath ./bin;./lib/antlr.jar;./lib/velocity-dep-
1.4.jar;./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/newtable.ezql

rem commented out temporarily to avoid database failures.
```

```
rem java -classpath
./junit3.8.1/junit.jar;./bin;./lib/antlr.jar;./lib/velocity-dep-
1.4.jar;./lib/mysql-connector-java-3.0.9-stable-bin.jar
ezql.test.MainTestSuite


testparser.sh
-------------------------------------

java -classpath ./bin:./lib/antlr.jar:./lib/velocity-dep-
1.4.jar:./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/recursion.ezql
java -classpath ./bin:./lib/antlr.jar:./lib/velocity-dep-
1.4.jar:./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/query.ezql
java -classpath ./bin:./lib/antlr.jar:./lib/velocity-dep-
1.4.jar:./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/merge.ezql
java -classpath ./bin:./lib/antlr.jar:./lib/velocity-dep-
1.4.jar:./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/table.ezql
java -classpath ./bin:./lib/antlr.jar:./lib/velocity-dep-
1.4.jar:./lib/mysql-connector-java-3.0.9-stable-bin.jar ezql.TestParser
sample/newtable.ezql


ezql.properties
-------------------------------------

driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/ezql
database=ezql
user=ezql
password=test
workingdir=/Users/bbhatti/Desktop/school/ezql


velocity.properties
-----------------------------------

file.resource.loader.path = ./templates
runtime.log = ezql.log
```