



Gudu Software General SQL Parser User Guide

Version 1.0
Release Date 19 September 2012

Gudu Software
<http://www.sqlparser.com>

DOCUMENT INFORMATION

Gudu Software

General SQL Parser User Guide

Version 1.0

Printed 19 September 2012

Gudu Software Support Information

For support queries, please contact us at info@sqlparser.com.

Access the Gudu Software Web site at the following URL:

www.sqlparser.com

Copyright

Copyright © 2002-2012 Gudu Software, Inc. All rights reserved.

Trademarks

All trademarks and registered trademarks are the property of their respective owners.

Confidentiality

CONFIDENTIAL AND PROPRIETARY INFORMATION OF Gudu Software. The information set forth herein represents the confidential and proprietary information of Gudu Software. Such information shall only be used for the express purpose authorized by Gudu Software and shall not be published, communicated, disclosed or divulged to any person, firm, corporation or legal entity, directly or indirectly, or to any third person without the prior written consent of Gudu Software.

Disclaimer

Gudu Software provides this publication "as is" without warranty of any kind, either express or implied. In no event shall Gudu Software be liable for any loss of profits, loss of business, loss of use or data, interruption of business, or for indirect, special, punitive, incidental, or consequential damages of any kind.

No part of this work covered by copyright herein may be reproduced in any form or by any means—graphic, electronic, or mechanical—including photocopying, recording, taping, or storage in an information retrieval system, without prior written permission of the copyright owner.

This publication is subject to replacement by a later edition. To determine if a later edition exists, contact www.sqlparser.com.

TABLE OF CONTENTS

Document information	2
<i>Table of Contents</i>	3
<i>Preface</i>	5
Conventions	5
Contacts/Reporting problems	7
<i>Overview of General SQL Parser</i>	9
About the General SQL Parser	9
Problems the General SQL Parser Solves	9
Using the General SQL Parser	10
How the General SQL Parser works	11
Comparing the General SQL Parser to other products	13
<i>Installing General SQL Parser</i>	15
purchasing the General SQL Parser	15
Downloading the General SQL Parser	16
Configuring the General SQL Parser	16
<i>Getting Started with General SQL Parser</i>	18
Tutorial on using the General SQL Parser	18
<i>Advanced Uses of General SQL Parser</i>	25
UML class diagram	25
How the SQL JOIN is decoded	26
SQL expression parse tree traversal	33
Other useful demos	35
<i>Table of Figures</i>	36
<i>Index</i>	37

PREFACE

This document acts as a guide to using Gudu Software General SQL Parser for parsing, formatting, modifying, and analyzing the SQL code used to manipulate the data stored in a database.

Target Audience

This document is intended for all users of General SQL Parser, including the following:

- Programmers who develop database-related tools that analyze SQL queries.
- Programmers familiar with Java, .NET, and SQL for various databases, including Oracle and SQL Server.
- Users of business intelligence or data mining, who perform table or column compact analysis or any other SQL-related analysis.

Pre-Requisite

It is assumed at this point that you have installed Gudu Software General SQL Parser on your machine.

CONVENTIONS

The following tables list the various conventions used in Gudu Software documentation. We follow these conventions to help you quickly and easily identify particular elements, processes, and names that occur frequently in documents.



Typographical conventions

This guide uses the following typographical conventions:

Convention	Description
Bold text	Indicates one of the following: <ul style="list-style-type: none">• Screen element• New terminology• A file or folder name• A control in an application's user interface• A registry key• Important information
<i>Italic text</i>	Indicates a reference or the title of a publication.
Monospaced text	Indicates code examples or system messages.
Monospaced bold text	Indicates system commands that you enter.
<i>Hyperlink</i>	Indicates an Internet link to target material.

Graphical conventions

This guide uses the following graphical conventions:

Convention	Description
	Indicates additional information that may be of interest to the reader.
	Indicates cautions that, if ignored, can result in damage to software or hardware.

CONTACTS/REPORTING PROBLEMS

These sections present contact information for a variety of situations.

Sales

For any sales queries, please contact us at sales@sqlparser.com.

Support

For support queries, please contact us at info@sqlparser.com.

Latest updates and information

For the latest updates and information, please visit us at www.sqlparser.com.

Gudu Software Web site

Access the Gudu Software Web site at the following URL:
www.sqlparser.com

OVERVIEW OF GENERAL SQL PARSER

This chapter covers the following topics:

- What the General SQL Parser is
- What problems the General SQL Parser solves
- Why you would use the General SQL Parser
- How the General SQL Parser works
- How the General SQL Parser compares to alternatives

ABOUT THE GENERAL SQL PARSER

Adding a home-grown SQL parser to your applications is a difficult problem. Not only is decoding SQL grammar difficult, but database vendors are constantly releasing new versions of their databases. Maintaining your own SQL parser is time-consuming, error-prone, and costly.

Gudu Software has developed a SQL parser that simplifies decoding SQL grammar and helps your applications stay current with the latest versions of database programs.

The General SQL Parser is a package that enables you to add powerful SQL functionality to your applications. General SQL Parser offers a custom SQL engine for many widely-used databases, including a query parser for SQL Server, Oracle, DB2, MySQL, Teradata, PostgreSQL, and Access databases.

Using the General SQL Parser library, you can save hundreds of hours of development time, improving your applications and speeding them to your customers.

PROBLEMS THE GENERAL SQL PARSER SOLVES

The General SQL Parser solves many common and difficult SQL development problems, including the following:

- Checks SQL syntax offline so that you can validate syntax without connecting to a database.
- Formats SQL with more than 100 highly customizable format options.
- Performs an in-depth analysis of SQL scripts, including a detailed SQL parse tree node structure.
- Provides full access to the SQL query parse tree, including fetching, modifying, and rewriting SQL segments.

- Prevents SQL injection attacks.
- Translate SQL between different databases

USING THE GENERAL SQL PARSER

The General SQL Parser is available for major programming languages, including C#, VB.NET, Java, C/C++, Delphi, and Visual Basic.

Offline SQL Syntax Check

With our vendor-specific offline SQL syntax check, you can validate SQL syntax without connecting to the database server. Syntax errors can be found even before executing SQL on your production database server. This is especially useful if your SQL was dynamically built based on user input. Supported databases include Oracle (PLSQL), SQL Server, MySQL, DB2, and Access.

SQL Formatter

You can easily integrate our SQL formatter into your application to generate a color-coded, professional, and intuitive SQL layout that gives your product a professional look and feel. Our highly customizable SQL formatter offers more than 100 format options, including text, HTML, and RTF output.

We know how difficult it is to build a flexible and stable SQL formatter. If you need to support more than one SQL dialect – such as Oracle, SQL Server, DB2, and MySQL – then this library could be the smartest investment you ever make, saving you hundreds of hours of coding.

Avoid SQL Injection Attacks

Worried about being vulnerable to SQL injection attacks on your ASP.NET or Java applications? Our operational library automatically detects malicious SQL segments.

Database Intelligence

Precisely determining and renaming every table and column in stored SQL statements that include considerable nesting sub-queries is a complex procedure. Using the General SQL Parser can make this tedious procedure hassle-free for you, so that you can easily:

- Extract all tables and field names.
- Parse the values and field names in where or having clauses, and verify that the user has specified certain columns in the where clause.
- Determine which tables are accessed, and how these tables are accessed (read, write, or both).
- Determine which table and column are used in a select list field.
- Rename specific table names and rebuild the SQL statements.
- Perform column impact analysis and build source-to-target column/table mappings.
- Read SQL scripts to determine which tables have Create, Read, Update, Delete, and Insert operations against them.

SQL Query Parse Tree

Parsing SQL is a notoriously difficult task. But the General SQL Parser can produce a SQL query parse tree in XML format, so you can further process SQL scripts.

HOW THE GENERAL SQL PARSER WORKS

The General SQL Parser consists of complete library packages available for the following development environments:

- .NET
- Java
- C/C++
- COM
- VCL

The General SQL Parser supports a multitude of databases, including Oracle, SQL Server, DB2, MySQL, Teradata, PostgreSQL, and Access.

Not only do we provide the libraries. We also offer a wide range of free demos that you can copy and adapt to solve your specific problems.

Below is the high-level architecture of the General SQL Parser.

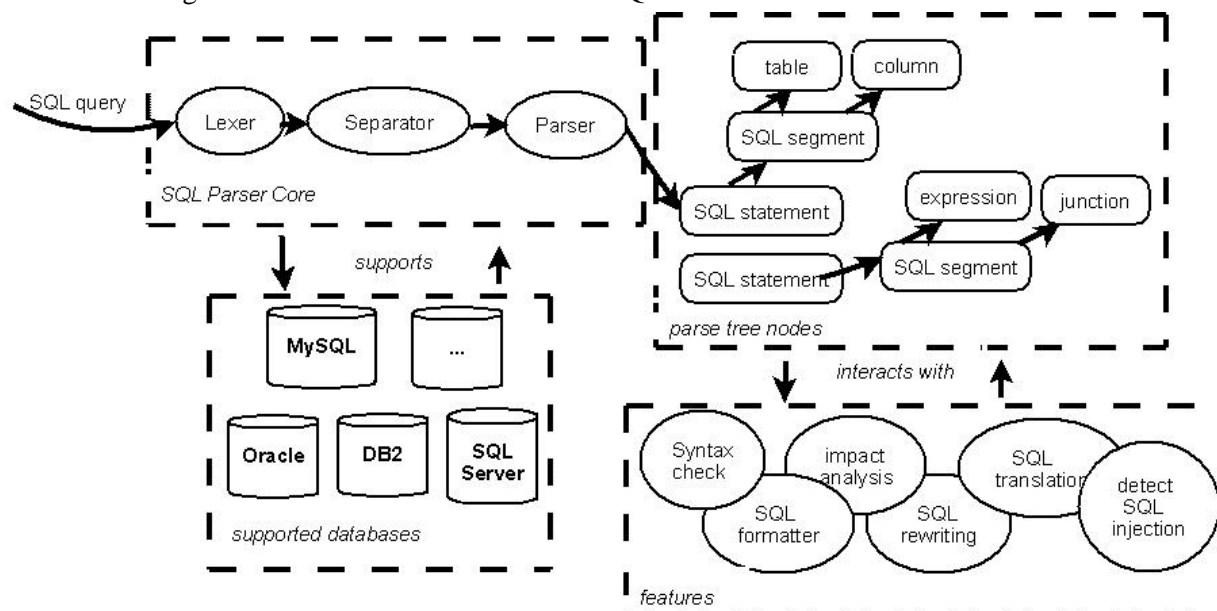


Figure 1: Architecture of General SQL Parser.

The General SQL Parser reads the input SQL text of the query. Lexer (lex parser) tokenizes the input SQL into a list of tokens. Separator turns the source tokens into a list of SQL statements. Each separate SQL statement includes a fragment of source tokens, which is the input to the Yacc Parser.

The Yacc Parser reads the source tokens of each SQL statement. If no syntax error is found, the Parser creates a raw parse tree, based on the BNF of different dialects of databases. Otherwise, the syntax error is detected. A syntax error in one SQL statement doesn't affect others.

Parser then translates the raw query parse tree into a formal parse tree. The top-level node of the formal parse tree is a specific type of SQL statement, such as TSelectSqlStatement, TInsertSqlStatement,

TDeleteSqlStatement, TUpdateSqlStatement, or TCreateTableSqlStatement, which, in turn, includes other sub-parse-tree nodes.

General SQL Parser Dataflow Overview

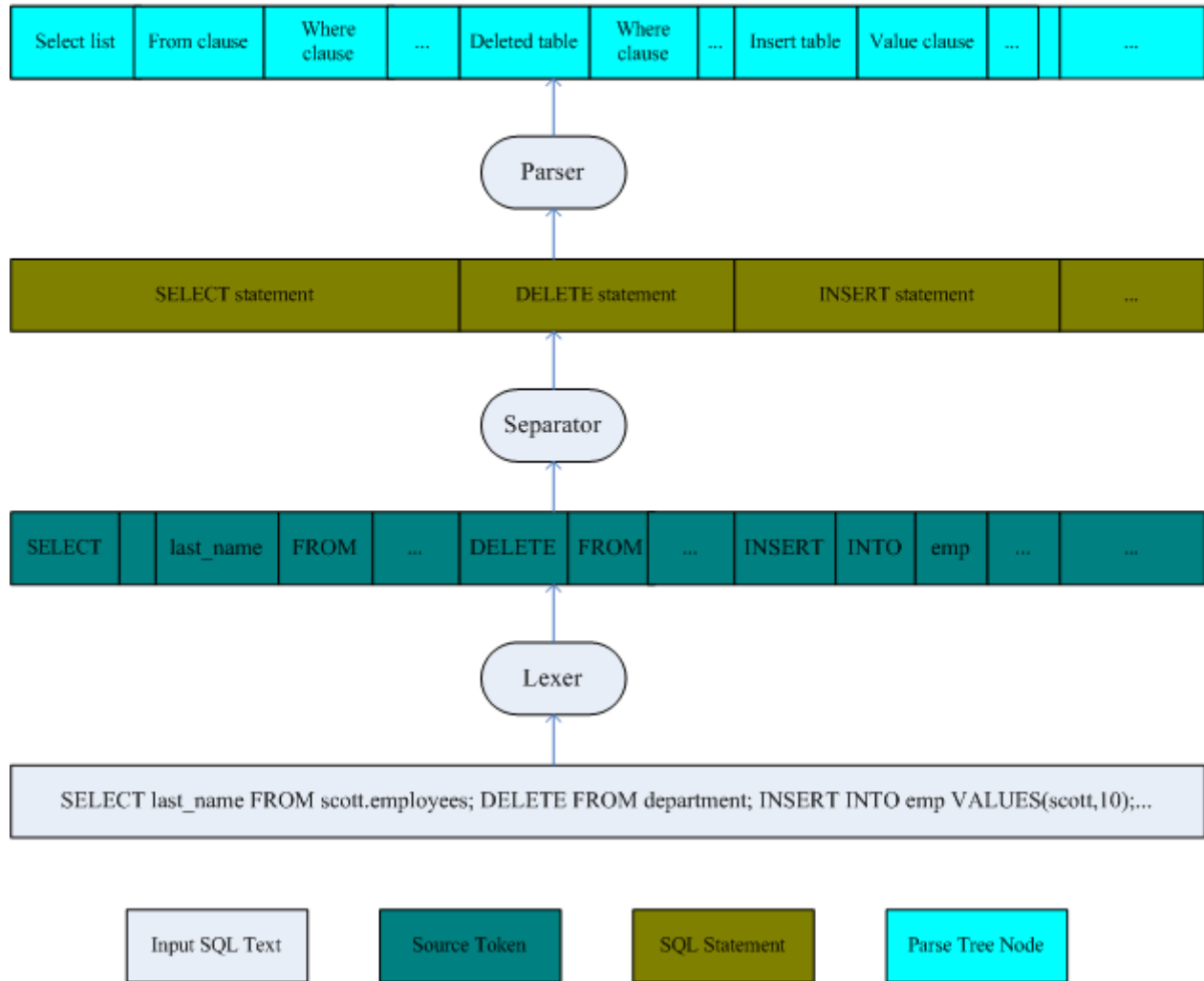


Figure 2: Data Flow in General SQL Parser.

The formal parse tree has a variety of uses, such as the following:

- Obtaining information, such as tables and fields, in a SQL statement is easy. These items are ready for use, along with other database objects in SQL objects.
- Parse tree visitors that descend from TLzVisitorAbs can be used to iterate the parse tree. TLzParseTree2XMLVisitor is a simple visitor that translates the parse tree to XML output. This class is a good reference before you write your own visitor to handle the parse tree.

Many other uses depend only on your requirement and imagination.

COMPARING THE GENERAL SQL PARSER TO OTHER PRODUCTS

When comparing the General SQL Parser to other products, consider the following points:

- Other products are available only for a single language. The General SQL Parser is available for major programming languages, including C#, VB.NET, Java, C/C++, Delphi, and Visual Basic.
- Other products are available only for a single database. The General SQL Parser is available for major databases, including SQL Server, Oracle, DB2, MySQL, Teradata, PostgreSQL, and Access databases.
- The SQL parsers of database vendors are embedded into the database. There is not API available for use by third-party developers and vendors.
- The General SQL Parser's powerful SQL parse and analysis capabilities enable it to parse complex SQL statements that other parsers cannot handle.
- Makers of other parsers actually recommend the General SQL Parser:
 - <http://zql.sourceforge.net>
 - <http://jsqlparser.sourceforge.net>

INSTALLING GENERAL SQL PARSER

This chapter covers the following topics:

- How to download or purchase the General SQL Parser
- How to install the General SQL Parser
- How to configure the General SQL Parser

PURCHASING THE GENERAL SQL PARSER

You can purchase single-developer licenses for the General SQL Parser at the <http://www.sqlparser.com/buynow.php> web site. For a quote for multiple-developer licenses, send an email to sales@sqlparser.com.

To purchase a single-developer license for the General SQL Parser, follow these instructions:

1. Visit the <http://www.sqlparser.com/buynow.php> web site.
2. From the “Please select a product” drop-down list, select one of the following choices:
 - .NET
 - Java
 - COM
 - VCL
3. From the “Please select developer license” drop-down list, select one of the following choices:
 - Enterprise Edition (which supports Oracle, SQL Server/Sybase/Access, DB2, MySQL and Teradata)
 - Professional Edition Oracle
 - Professional Edition SQL Server/Sybase/Access
 - Professional Edition DB2
 - Professional Edition MySQL
4. Click either “Buy Now Via Plimus” or “Buy Now Via PayPal”.
5. Check your email for your receipt and a download link. On rare occasions, orders may take longer to process, so please be patient. Also, check your spam folder because sometimes emails are diverted there.

DOWNLOADING THE GENERAL SQL PARSER

You can download a FREE Trial Version of the General SQL Parser for evaluation purposes. The FREE Trial Version processes SQL queries with fewer than 10,000 characters. The FREE Trial Version expires 90 days after downloading.

To download a FREE Trial Version of the General SQL Parser for evaluation purposes, follow these instructions:

1. Visit the <http://www.sqlparser.com/download.php> web site.
2. From the list of downloads, select one of the following choices:
 - General SQL Parser for .NET
 - General SQL Parser for Java
 - General SQL Parser C/C++ version
 - General SQL Parser COM version, which can be used in any COM-enabled development environment, including Visual Basic and C++
 - General SQL Parser VCL version, for Delphi XE
 - General SQL Parser VCL version, for Delphi XE2
 - General SQL Parser VCL version, for Delphi XE2 update 4
 - General SQL Parser VCL version, for Delphi 7
 - General SQL Parser VCL version, for Delphi 2010
 - General SQL Parser VCL version, for Delphi 2009, Delphi 2007, Delphi 2006, Delphi 2005, Delphi 6, and Delphi 5
3. The Download Trial Version page appears.
4. Optionally, enter your email address to receive free updates, upgrades, and special offers.
5. Click “**Download Now!**”.
6. Save the download zip file to your computer.
7. Extract the contents of the zip file to an empty folder on your computer, such as C:\tmp.

CONFIGURING THE GENERAL SQL PARSER

The configuration of the General SQL Parser depends on the development environment. This section gives instructions on configuring the General SQL Parser for several development environments.

Configuring the General SQL Parser for a Java development environment

To configure the General SQL Parser for a Java development environment, follow these instructions:

1. Ensure that you have an up-to-date version of the Java JDK installed on your computer. Visit the <http://www.oracle.com/technetwork/java/javase/downloads/index.html> web site. Select the JDK download. On the next page, select your computer’s operating system. Download and install the Java JDK as instructed.
2. Open a DOS command window, also called a Command Prompt window.
3. Navigate to the `dist` folder. If you unzipped the contents of the zip file to the folder C:\tmp, then the `dist` folder is located at C:\tmp\gsp_java_trial\dist\.
4. Enter these commands at the DOS command prompt:


```
set java_home="C:\Program Files\Java\<jdk_folder>";  
where jdk_folder is the folder created when installing the Java JDK in step 1 above  
set path=%path%;%java_home%\bin;  
set classpath=%classpath%;%java_home%\jre\lib\rt.jar;.\gsp.jar
```

5. To compile one of the included demos, follow these instructions:
 - a. The included demo programs are in the folder
C:\tmp\gsp_java_trial\dist\demos. For this procedure, we use the
checksyntax demo program.
 - b. Enter this command at the DOS command prompt:
javac demos\checksyntax\checksyntax.java
 - c. Create a simple test SQL file called test.sql in the
C:\tmp\gsp_java_trial\dist folder containing this sample SQL statement:
select count(*) from tab
 - d. Run the compiled checksyntax demo program with the test.sql file by entering this
command at the DOS command prompt:
java demos.checksyntax.checksyntax test.sql
 - e. The demo program gives output such as "Check syntax ok!".
6. You can compile other demos, or your own .java programs, similarly.

GETTING STARTED WITH GENERAL SQL PARSER

This chapter contains a tutorial on using the General SQL Parser.

TUTORIAL ON USING THE GENERAL SQL PARSER

This tutorial illustrates how to use the Java version of the General SQL Parser to decode the SQL grammar of the SELECT SQL statement. When you are more familiar with the General SQL Parser, you can use this SQL library to decode all supported SQL statements to as detailed a SQL fragment as you need.

Sample SQL and output SQL

For this tutorial, we use the following sample SQL:

```
SELECT e.last_name      AS name,
       e.commission_pct comm,
       e.salary * 12    "Annual Salary"
FROM   scott.employees AS e
WHERE  e.salary > 1000
ORDER BY
       e.first_name,
       e.last_name;
```

The decoded output of this sample SQL is the following:

```
Select statement:
  Column: e.last_name, Alias: name
  Column: e.commission_pct, Alias: comm
  Column: e.salary * 12, Alias: "Annual Salary"
```

```

table:
    scott.employees, alias: e
where clause:
    e.salary > 1000
order by:
    e.first_name
    e.last_name

```

Procedure

Step 1.

This step consists of the following tasks:

1. Create an instance of TGSqlParser.
2. Assign the sample SQL text.
3. Parse the sample SQL text.
4. Create the syntax to show an error message, if any error is detected. Otherwise, call the analyzeStmt method to decode the various SQL statement types accordingly.

Here is the code to perform these tasks:

```

public static void main(String args[]){

    TGSqlParser sqlparser = new TGSqlParser(EDbVendor.dbvoracle);

    sqlparser.sqltext = "SELECT e.last_name AS name,\n" +
        "        e.commission_pct comm,\n" +
        "        e.salary * 12 \"Annual Salary\"\n" +
        "FROM    scott.employees AS e\n" +
        "WHERE   e.salary > 1000\n" +
        "ORDER  BY\n" +
        "    e.first_name,\n" +
        "    e.last_name;";

    int ret = sqlparser.parse();
    if (ret == 0){
        for(int i=0;i<sqlparser.sqlstatements.size();i++){
            analyzeStmt(sqlparser.sqlstatements.get(i));
            System.out.println("");
        }
    }else{
        System.out.println(sqlparser.getErrorMessage());
    }
}

```

Step 2.

With the analyzeStmt method, different SQL statements have different SQL structures, so we must handle them separately. In this tutorial, we just provide an analyzeSelectStmt method to illustrate how to decode a SELECT statement.

Here is the code for this step:

```
protected static void analyzeStmt(TCustomSqlStatement stmt){

    switch(stmt.sqlstatementtype){
        case sstselect:
            analyzeSelectStmt((TSelectSqlStatement)stmt);
            break;
        case sstupdate:
            break;
        case sstcreatetable:
            break;
        case sstaltertable:
            break;
        case sstcreateview:
            break;
        default:
            System.out.println(stmt.sqlstatementtype.toString());
    }
}
```

Step 3.

Using the analyzeSelectStmt method, we can use APIs provided by the Java version of the General SQL Parser to access different parse tree nodes of the SELECT statement such as the select list, from clause, and where clause.

Here is the code for this step:

```
protected static void analyzeSelectStmt(TSelectSqlStatement pStmt){
    System.out.println("\nSelect statement:");
    if (pStmt.isCombinedQuery()){
        String setstr="";
        switch (pStmt.getSetOperator()){
            case 1: setstr = "union";break;
            case 2: setstr = "union all";break;
            case 3: setstr = "intersect";break;
            case 4: setstr = "intersect all";break;
            case 5: setstr = "minus";break;
            case 6: setstr = "minus all";break;
            case 7: setstr = "except";break;
            case 8: setstr = "except all";break;
        }
        System.out.printf("set type: %s\n",setstr);
        System.out.println("left select:");
        analyzeSelectStmt(pStmt.getLeftStmt());
        System.out.println("right select:");
        analyzeSelectStmt(pStmt.getRightStmt());
        if (pStmt.getOrderbyClause() != null){
            System.out.printf("order by clause
                %s\n",pStmt.getOrderbyClause().toString());
        }
    }else{
        //select list
    }
}
```

```

for(int i=0; i < pStmt.getResultColumnList().size();i++){
    TResultColumn resultColumn =
    pStmt.getResultColumnList().getResultColumn(i);
    System.out.printf("\tColumn: %s, Alias:
    %s\n",resultColumn.getExpr().toString(),
    (resultColumn.getAliasClause() ==
    null)?"":resultColumn.getAliasClause().toString());
}

//from clause, check this document for detailed information
//http://www.sqlparser.com/sql-parser-query-join-table.php
for(int i=0;i<pStmt.joins.size();i++){
    TJoin join = pStmt.joins.getJoin(i);
    switch (join.getKind()){
        case TBaseType.join_source_fake:
            System.out.printf("\ntable: \n\t%s, alias:
            %s\n",join.getTable().toString(),(join.getTab
            le().getAliasClause() !=null)?join.getTable()
            .getAliasClause().toString():"");
            break;
        case TBaseType.join_source_table:
            System.out.printf("\ntable: \n\t%s, alias:
            %s\n",join.getTable().toString(),(join.getTab
            le().getAliasClause() !=null)?join.getTable()
            .getAliasClause().toString():"");
            for(int
            j=0;j<join.getJoinItems().size();j++){
                TJoinItem joinItem =
                join.getJoinItems().getJoinItem(j);
                System.out.printf("Join type:
                %s\n",joinItem.getJoinType().toString()
                );
                System.out.printf("table: %s, alias:
                %s\n",joinItem.getTable().toString(),(j
                oinItem.getTable().getAliasClause() !=n
                ull)?joinItem.getTable().getAliasClause
                ().toString():"");
                if (joinItem.getOnCondition() != null){
                    System.out.printf("On:
                    %s\n",joinItem.getOnCondition().to
                    String());
                }else if
                (joinItem.getUsingColumns() != null){
                    System.out.printf("using:
                    %s\n",joinItem.getUsingColumns().t
                    oString());
                }
            }
            break;
        case TBaseType.join_source_join:
            TJoin source_join = join.getJoin();
            System.out.printf("\ntable: \n\t%s, alias:
            %s\n",source_join.getTable().toString(),(sour
            ce_join.getTable().getAliasClause() !=null)?s

```

```

source_join.getTable().getAliasClause().toString():");

for(int
j=0;j<source_join.getJoinItems().size();j++){
    TJoinItem joinItem =
    source_join.getJoinItems().getJoinItem(
    j);
    System.out.printf("source_join type:
%s\n",joinItem.getJoinType().toString(
    ));
    System.out.printf("table: %s, alias:
%s\n",joinItem.getTable().toString(),(j
oinItem.getTable().getAliasClause() !=n
ull)?joinItem.getTable().getAliasClause
().toString():");
    if (joinItem.getOnCondition() != null){
        System.out.printf("On:
%s\n",joinItem.getOnCondition().to
String());
    }else if
    (joinItem.getUsingColumns() != null){
        System.out.printf("using:
%s\n",joinItem.getUsingColumns().t
oString());
    }
}

for(int
j=0;j<join.getJoinItems().size();j++){
    TJoinItem joinItem =
    join.getJoinItems().getJoinItem(j);
    System.out.printf("Join type:
%s\n",joinItem.getJoinType().toString(
    ));
    System.out.printf("table: %s, alias:
%s\n",joinItem.getTable().toString(),(j
oinItem.getTable().getAliasClause() !=n
ull)?joinItem.getTable().getAliasClause
().toString():");
    if (joinItem.getOnCondition() != null){
        System.out.printf("On:
%s\n",joinItem.getOnCondition().to
String());
    }else if
    (joinItem.getUsingColumns() != null){
        System.out.printf("using:
%s\n",joinItem.getUsingColumns().t
oString());
    }
}

break;
default:

```

```

                System.out.println("unknown type in join!");
                break;
            }
        }

        //where clause
        if (pStmt.getWhereClause() != null){
            System.out.printf("\nwhere clause: \n\t%s\n",
                pStmt.getWhereClause().getCondition().toString());
        }

        // group by
        if (pStmt.getGroupByClause() != null){
            System.out.printf("\ngroup by:
                \n\t%s\n",pStmt.getGroupByClause().toString());
        }

        // order by
        if (pStmt.getOrderbyClause() != null){
            System.out.printf("\norder by:");
            for(int
                i=0;i<pStmt.getOrderbyClause().getItems().size();i++){
                System.out.printf("\n\t%s",pStmt.getOrderbyClause()
                    .getItems().getOrderByItem(i).toString());
            }
        }

        // for update
        if (pStmt.getForUpdateClause() != null){
            System.out.printf("for update:
                \n%s\n",pStmt.getForUpdateClause().toString());
        }

        // top clause
        if (pStmt.getTopClause() != null){
            System.out.printf("top clause:
                \n%s\n",pStmt.getTopClause().toString());
        }

        // limit clause
        if (pStmt.getLimitClause() != null){
            System.out.printf("top clause:
                \n%s\n",pStmt.getLimitClause().toString());
        }
    }
}

```

Step 4.

With this Java code available, we can assemble them into a runnable program by putting them into this structure:

```

import gudusoft.gsqlparser.*;
import gudusoft.gsqlparser.nodes.TJoin;

```

```
import gudusoft.gsqlparser.nodes.TJoinItem;
import gudusoft.gsqlparser.nodes.TResultColumn;
import gudusoft.gsqlparser.stmt.*;

public class tutorial {

}
```

Then you can run this Java program with any `SELECT` statement to see how it works.

Conclusion of tutorial

This tutorial is simple, but still powerful. The General SQL Parser can do far more than this. Please read the next chapter to see an overview of all the classes available in this SQL library, as well as other advanced topics.

ADVANCED USES OF GENERAL SQL PARSER

This chapter includes further details about how to use the General SQL Parser.

UML CLASS DIAGRAM

This section presents a UML class diagram that shows how all the classes are related. This class diagram gives you a better understanding of how the General SQL Parser decodes SQL scripts, as well as showing how SQL statements are organized in the Java classes.

General SQL Parser Java version Classes Overview

This diagram only list some major classes. Check API reference for a full list of class

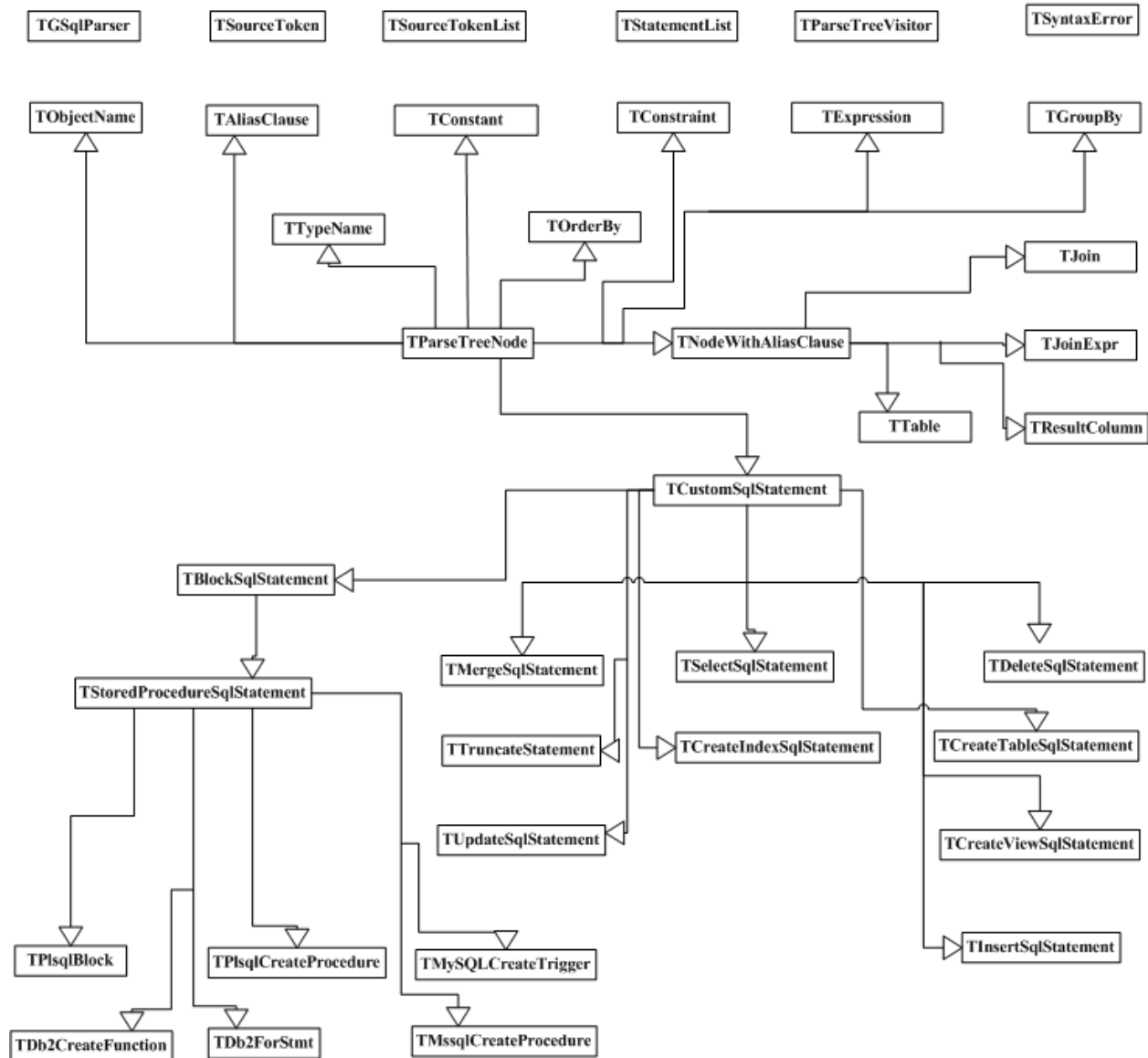


Figure 3: Java Classes for General SQL Parser.

HOW THE SQL JOIN IS DECODED

This section describes how the SQL JOIN is decoded.

The JOIN keyword is used in a SQL statement to query data from two or more tables, based on a relationship between certain columns in these tables. The types of JOIN include inner join, left join, right

join, and full join. In General SQL Parser, those join types are represented by classes such as TLzJoin, TLzJoinList, TLzJoinItem, and TLzJoinItemList.

The following three diagrams illustrate TLzJoin, TLzJoinList, TLzJoinItem, and TLzJoinItemList, which are used to represent a join table in a FROM clause. You can check TSelectSqlStatement.JoinTables, which is a type of TLzJoinList, to start fetching all the information.

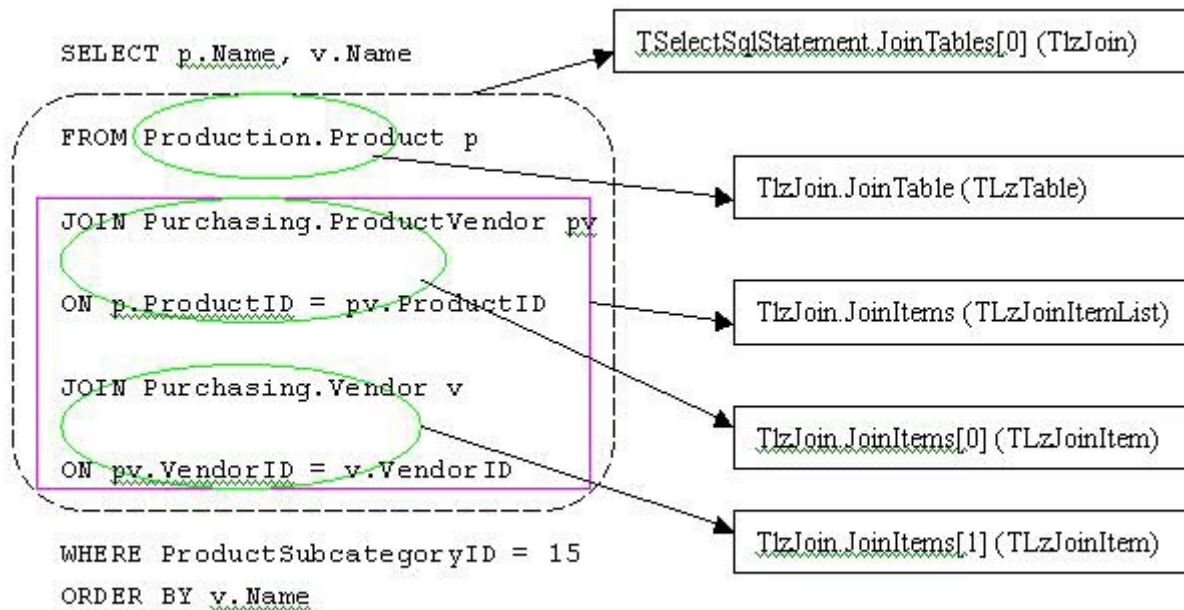


Figure 4: Using TSelectSqlStatement.JoinTables.

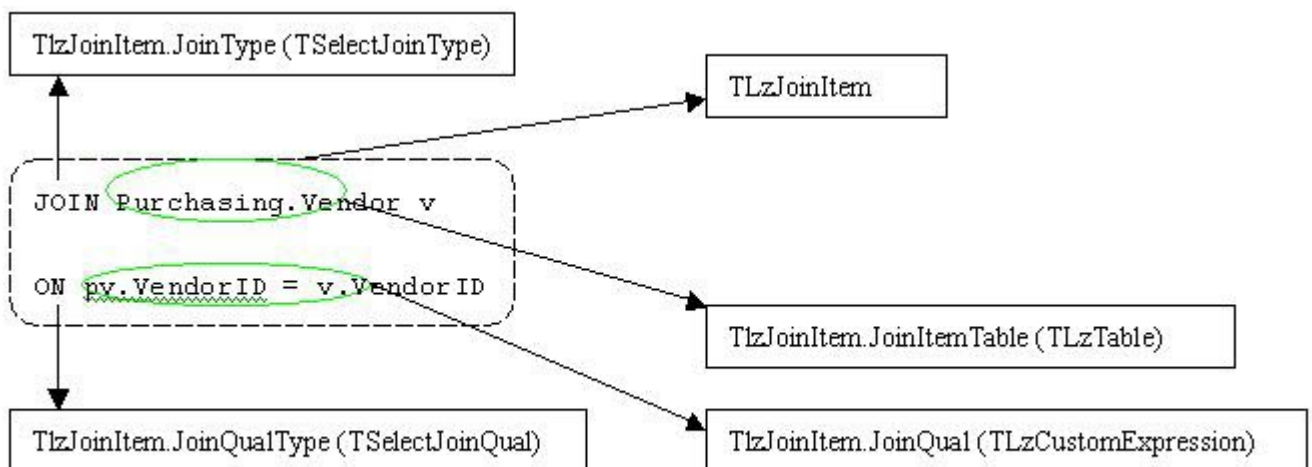


Figure 5: Using TLzJoinItem.

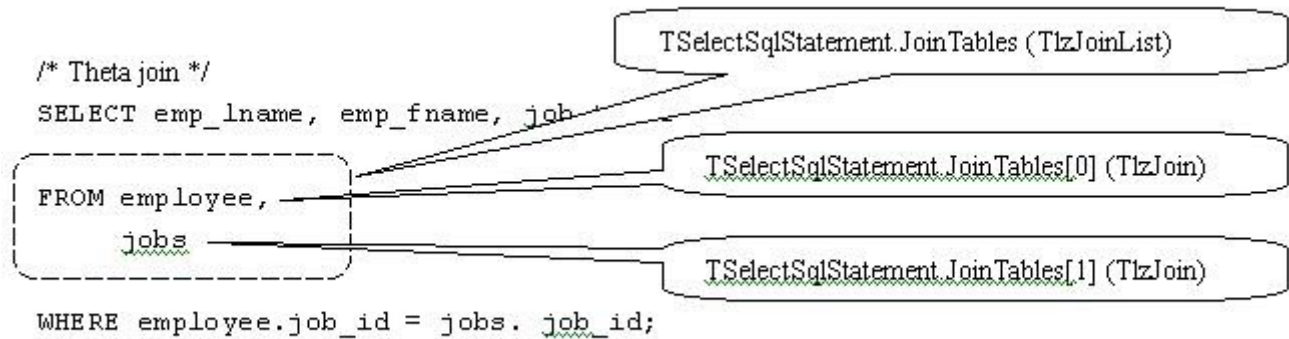


Figure 6: Using TSelectSqlStatement.JoinTables.

Scenario 1

Consider this sample SQL code for Scenario 1:

```

Select t1.f1
from my.table1 t1,my.table2 t2
where t1.f1 = t2.f1

```

After parsing this SQL, the data in TSelectSqlStatement.Tables is:

```

Tables.Count = 2;

Tables[0].TableName = 'table1';
Tables[0].TablePrefix = 'my';
Tables[0].TableAlias = 't1';

Tables[1].TableName = 'table2';
Tables[1].TablePrefix = 'my';
Tables[1].TableAlias = 't2';

```

And the information in TSelectSqlStatement.JoinTables is:

```

JoinTables.Count = 2;

JoinTables.items[0].JoinTableType = jttTable; // the collection
of join is table : my.table1
JoinTables.items[0].JoinTable = Tables[0]; //my.table1
JoinTables.items[0].JoinItems.count = 0;

JoinTables.items[1].JoinTableType = jttTable; // the collection
of join is table : my.table1

```

```
JoinTables.items[1].JoinTable = Tables[1]; //my.table2
JoinTables.items[1].JoinItems.count = 0;
```

Scenario 2

Consider this sample SQL code for Scenario 2:

```
Select t1.f1
from my.table1 t1
join my.table2 t2
on t1.f1 = t2.f1
```

After parsing this SQL, the data in TSelectSqlStatement.Tables is:

```
Tables.Count = 2;

Tables[0].TableName = 'table1';
Tables[0].TablePrefix = 'my';
Tables[0].TableAlias = 't1';

Tables[1].TableName = 'table2';
Tables[1].TablePrefix = 'my';
Tables[1].TableAlias = 't2';
```

Notice that everything is the same as in scenario 1.

Now, check the information in JoinTables:

```
JoinTables.Count = 1;

JoinTables.items[0].JoinTableType = jttTable; // the collection
of join is table : my.table1
JoinTables.items[0].JoinTable = Tables[0]; //my.table1
JoinTables.items[0].JoinItems.count = 1;

JoinTables.items[0].JoinItems[0].JoinItemTableType =
jttTable; //the collection of this joinitem is table:my.table2
JoinTables.items[0].JoinItems[0].JoinItemTable = my.table2;
JoinTables.items[0].JoinItems[0].JoinType = sjtjoin;
JoinTables.items[0].JoinItems[0].JoinQualType = sjqOn;
JoinTables.items[0].JoinItems[0].JoinQual = AEXPRESSION ; //a
custom expressoin point to t1.f1 = t2.f1
```

Scenario 3

Consider this sample SQL code for Scenario 3:

```
select t1.f1
from my.table1 t1
  join (my.table2 t2
  left join my.table3 t3
  on t2.f1 = t3.f1) as joinalias1
  on t1.f1 = t2.f1;
```

After parsing this SQL, the data in TSelectSqlStatement.Tables is:

```
Tables.Count = 3;

Tables[0].TableName = 'table1';
Tables[0].TablePrefix = 'my';
Tables[0].TableAlias = 't1';

Tables[1].TableName = 'table2';
Tables[1].TablePrefix = 'my';
Tables[1].TableAlias = 't2';

Tables[2].TableName = 'table3';
Tables[2].TablePrefix = 'my';
Tables[2].TableAlias = 't3';
```

Now, check the data in JoinTables:

```
JoinTables.Count = 1;

JoinTables.items[0].JoinTableType = jttTable; // the collection
of join is table : my.table1
JoinTables.items[0].JoinTable = Tables[0]; //my.table1
JoinTables.items[0].JoinItems.count = 1;

JoinTables.items[0].JoinItems[0].JoinItemTableType =
jttJoin; //the collection of this joinitem is join
JoinTables.items[0].JoinItems[0].JoinItemJoin = AJoin; // ajoin
point to (my.table2 t2
left join my.table3 t3
on t2.f1 = t3.f1) as joinalias1

JoinTables.items[0].JoinItems[0].JoinType = sjtjoin;
JoinTables.items[0].JoinItems[0].JoinQualType = sjqOn;
```

```

JoinTables.items[0].JoinItems[0].JoinQual = AEXPRESSION ; //a
custom expressoin point to t1.f1 = t2.f1

JoinTables.items[0].JoinItems[0].JoinItemJoin.JoinTableType =
jttTable;
JoinTables.items[0].JoinItems[0].JoinItemJoin.JoinTable =
Tables[1]; //my.table2
JoinTables.items[0].JoinItems[0].JoinItemJoin.Alias = joinalias1
JoinTables.items[0].JoinItems[0].JoinItemJoin.AliasWithAs = true;
JoinTables.items[0].JoinItems[0].JoinItemJoin.NestedLevel = 1;
JoinTables.items[0].JoinItems[0].JoinItemJoin.JoinItems.count = 1;

JoinTables.items[0].JoinItems[0].JoinItemJoin.JoinItems[0].JoinIt
emTableType = jttTable;
JoinTables.items[0].JoinItems[0].JoinItemJoin.JoinItems[0].JoinIt
emTable = Tables[2]; //my.table3
JoinTables.items[0].JoinItems[0].JoinItemJoin.JoinItems[0].JoinTy
pe = sjtleft;
JoinTables.items[0].JoinItems[0].JoinItemJoin.JoinItems[0].JoinQu
alType = jqton;
JoinTables.items[0].JoinItems[0].JoinItemJoin.JoinItems[0].JoinQu
al = AEXPRESSION ; //a custom expressoin point to t2.f1 = t3.f1

```

Scenario 4

Consider this sample SQL code for Scenario 4:

```

select t1.f1
from my.table1 t1
right join ((my.table2 t2
left outer join my.table3 t3
on (t2.f1 = t3.f2))
left join (my.table4 t4
full outer join my.table5 t5
on (t4.f1 = t5.f1)) t4alias
on (t4.b1 = t2.c1))
on (t1.a1 = t3.b3);

```

After parsing this SQL, the data in TSelectSqlStatement.Tables is:

```

Tables.Count = 5;

Tables[0].TableName = 'table1';
Tables[0].TablePrefix = 'my';
Tables[0].TableAlias = 't1';

```

```
Tables[1].TableName = 'table2';
Tables[1].TablePrefix = 'my';
Tables[1].TableAlias = 't2';
```

```
Tables[2].TableName = 'table3';
Tables[2].TablePrefix = 'my';
Tables[2].TableAlias = 't3';
```

```
Tables[3].TableName = 'table4';
Tables[3].TablePrefix = 'my';
Tables[3].TableAlias = 't4';
```

```
Tables[4].TableName = 'table5';
Tables[4].TablePrefix = 'my';
Tables[4].TableAlias = 't5';
```

Now, check the data in JoinTables:

```
JoinTables.Count = 1;
```

```
JoinTables.items[0].JoinTableType = jttTable;
JoinTables.items[0].JoinTable = Tables[0]; //my.table1
```

```
JoinTables.items[0].JoinItems.count = 1;
```

```
JoinTables.items[0].JoinItems[0].JoinType = sjtrightjoin;
JoinTables.items[0].JoinItems[0].JoinQualType = sjqOn;
JoinTables.items[0].JoinItems[0].JoinQual = AEXPRESSION ; //a
custom expressoin point to t1.a1 = t3.b3
```

```
JoinTables.items[0].JoinItems[0].JoinItemType = jttJoin;
JoinTables.items[0].JoinItems[0].JoinItemJoin = AJoin; //
((my.table2 t2
left outer join my.table3 t3
on (t2.f1 = t3.f2))
left join (my.table4 t4
full outer join my.table5 t5
on (t4.f1 = t5.f1)) t4alias
on (t4.b1 = t2.c1))
```

```
AJoin.NestedLevel = 1;
AJoin.JoinTableType = jttJoin;
AJoin.JoinJoin = ASubJoin; //(my.table2 t2
```



```
left outer join my.table3 t3
on (t2.f1 = t3.f2))
```

```
ASubJoin.NestedLevel = 1;
ASubJoin.JoinTableType = jttTable;
ASubJoin.JoinTable = Tables[1]; //my.table2
ASubJoin.JoinItems.count = 1;
ASubJoin.JoinItems.items[0].JoinType = sjtleftjoin;
ASubJoin.JoinItems.items[0].JoinQualType = jtqon;
ASubJoin.JoinItems.items[0].JoinQual = AExpression;//t2.f1 =
t3.f2
ASubJoin.JoinItems.items[0].JoinItemTableType = jttTable;
ASubJoin.JoinItems.items[0].JoinItemTable = Tables[2];
//my.table3
```

```
AJoin.JoinItems.count = 1;
```

```
AJoin.JoinItems.items[0].JoinTableType = jttJoin;//again, this is
a join
```

```
AJoin.JoinItems.items[0].JoinJoin = ASub2Join;//(my.table4 t4
full outer join my.table5 t5
on (t4.f1 = t5.f1)) t4alias
```

```
ASub2Join.Alias = t4alias;
ASub2Join.NestedLevel = 1;
ASub2Join.JoinTableType = jttTable;
ASub2Join.JoinTable = Tables[3]; //my.table4
ASub2Join.JoinItems.count = 1;
ASub2Join.JoinItems.items[0].JoinType = sjtfullOuter;
ASub2Join.JoinItems.items[0].JoinQualType = jtqon;
ASub2Join.JoinItems.items[0].JoinQual = AExpression;//t4.f1 =
t5.f1
ASub2Join.JoinItems.items[0].JoinItemTableType = jttTable;
ASubJoin.JoinItems.items[0].JoinItemTable = Tables[4];
//my.table5
```

```
AJoin.JoinItems.items[0].JoinType = sjtleftouter;//left join
```

```
AJoin.JoinItems.items[0].JoinQualType = sjqOn;
```

```
AJoin.JoinItems.items[0].JoinQual = sjqOn;// t4.b1 = t2.c1
```

SQL EXPRESSION PARSE TREE TRAVERSAL

This section discusses SQL expression parse tree traversal in pre-order, in-order, and post-order orders.

An expression is a combination of one or more values, operators, and SQL functions that evaluates to a value. An expression generally assumes the datatype of its components.

A SQL expression or condition is often used in the WHERE clause to filter which records are returned in the query's result set. Manipulating an expression is a common task when you process a SQL query in your program.

There are many kinds of SQL expressions among different database vendors. The General SQL Parser supports almost all of those expressions, and represents them in classes that are easy to process. For a detailed explanation of those expressions, check our online API reference at <http://sqlparser.com/kb/javadoc/gudusoft/gsqlparser/nodes/TExpression.html>.

In this section, we illustrate how to iterate an expression in a uniform way that lists the atomic elements of the expression in the pre-order, in-order, and post-order orders.

The expression in the WHERE clause of this sample SQL is iterated.

```
SELECT f1 FROM t1
WHERE t1.a = 10.2
      AND name = 'k'
      OR id IN ( 2, 4, 5 )
```

Pre-order traversal of parse tree of SQL expression

```
OR
AND
=
t1.a
10.2
=
name
'k'
IN
id
(
,
2
,
4
5
```

In-order traversal of parse tree of SQL expression

```
t1.a
```

```

=
10.2
AND
name
=
'k'
OR
id
IN
2
,
4
,
5
()

```

Post-order traversal of parse tree of SQL expression

```

t1.a
10.2
=
name
'k'
=
AND
id
2
4
5
,
,
()
IN
OR

```

OTHER USEFUL DEMOS

We have created many useful demos to help you make better use of this library. In addition, we frequently add demos upon request from users. You can find more demos at <http://www.dpriver.com/blog/list-of-demos-illustrate-how-to-use-general-sql-parser/>.

TABLE OF FIGURES

Figure 1: Architecture of General SQL Parser.....	11
Figure 2: Data Flow in General SQL Parser.....	12
Figure 3: Java Classes for General SQL Parser.....	26
Figure 4: Using TSelectSqlStatement.JoinTables.....	27
Figure 5: Using TLzJoinItem.....	27
Figure 6: Using TSelectSqlStatement.JoinTables.....	28

INDEX

- .NET, 5, 11, 15, 16
- Access, 2, 7, 9, 10, 11, 12, 15
- analyzeSelectStmt, 19, 20
- analyzeStmt, 19, 20
- ASP.NET, 10
- BNF, 11
- C, 10, 11, 12, 16, 17
- C++, 10, 11, 12, 16
- COM, 11, 15, 16
- contact information, 7
- DB2, 9, 10, 11, 12, 15
- decode, 18, 19
- Delphi, 10, 12, 16
- demos, 11, 17, 36
- event, 2
- in-order, 34
- In-order, 35
- Java, 5, 10, 11, 12, 15, 16, 17, 18, 20, 24, 25, 26
- Java classes, 25
- Java code, 24
- JDK, 16, 17
- JOIN, 26
- JoinTables, 27, 28, 29, 30, 31, 32
- MySQL, 9, 10, 11, 12, 15
- Oracle, 5, 9, 10, 11, 12, 15
- Parse, 10, 11, 12, 19
- parse tree, 9, 11, 12, 20, 34, 35
- Parser, 1, 2, 5, 9, 10, 11, 12, 13, 15, 16, 18, 20, 24, 25, 26, 27, 34
- PostgreSQL, 9, 11, 12
- post-order, 34
- Post-order, 35
- pre-order, 34
- Pre-order, 34
- scripts, 9, 10, 11, 25
- segments, 9, 10
- SQL grammar, 9, 18
- SQL injection, 10
- SQL library, 18, 24
- SQL Server, 5, 9, 10, 11, 12, 15
- Sybase, 15
- syntax, 9, 10, 11, 17, 19
- TCreateTableSqlStatement, 12
- TDeleteSqlStatement, 12
- Teradata, 9, 11, 12, 15
- TGSqlParser, 19
- TInsertSqlStatement, 11
- TLzJoin, 27
- TLzJoinItem, 27
- TLzJoinItemList, 27
- TLzJoinList, 27
- Trial Version, 16
- TSelectSqlStatement, 11, 20, 27, 28, 29, 30, 31
- TSelectSqlStatement.Tables, 28, 29, 30, 31
- TUpdateSqlStatement, 12
- UML class diagram, 25
- VCL, 11, 15, 16
- Visual Basic, 10, 12, 16
- Yacc, 11