Free Query Tool - tool for enhanced querying SQL databases

Department of Software Engineering
First of all I would like to give my thanks to the supervisor of my bachelor thesis RNDr. Michal Kopecký, Ph.D. (Charles University in Prague) for his valuable comments and help and for his neverending patience. I would like to thank also to professor Terence Parr (University of San Francisco) for such great tool that ANTLR is. Many thanks deserve my girlfriend Veronika Kováčiková for her support. Thank you all.

I declare I wrote the Bachelor’s thesis myself, using only the listed bibliography. I agree with making this thesis publicly available.

In Prague 29.5.2009

Martin Šuška
Obsah

1. Introduction .......................................................................................................... 7
   1.1. Motivation ..................................................................................................... 7
   1.2. Thesis structure .............................................................................................. 7

2. Analysis ................................................................................................................ 8
   2.1. Software future requirements ........................................................................ 9
   2.2. Query writing effectiveness ........................................................................ 9
       2.2.1. Syntax highlighting ............................................................................ 11
       2.2.2. Drag’n’drop ........................................................................................ 12
       2.2.3. Code completing ................................................................................ 13
       2.2.4. Database metadata retrieving ............................................................. 13
       2.2.5. Templates ........................................................................................... 14
   2.3. Existing tools ............................................................................................... 14
       2.3.1. SQL consoles ..................................................................................... 14
           2.3.1.1. SQLTools ................................................................................... 15
           2.3.1.2. SQuirreL ..................................................................................... 17
           2.3.1.3. SQL developer ........................................................................... 19
           2.3.1.4. TOAD ......................................................................................... 20
           2.3.1.5. Comparison ................................................................................ 21
       2.3.2. Editors ................................................................................................ 21
           2.3.2.1. Kate, PSPad, vi ........................................................................... 22
           2.3.2.2. Eclipse ........................................................................................ 22
   2.4. Requested features for implemented program ............................................. 23

3. Design and implementation ............................................................................... 24
   3.1. Basic principles ........................................................................................... 24
   3.2. Conception ................................................................................................... 25
   3.3. Database dependent module ........................................................................ 26
       3.3.1. Communication with database independent module ......................... 27
       3.3.2. Configuration file module.properties .............................................. 27
           3.3.2.1. Settings for Oracle module example ........................................ 28
   3.4. Independent part of the database ................................................................ 28
       3.4.1. Connection to the database ................................................................. 29
       3.4.2. Writing SQL commands ................................................................. 29
3.4.3. Background processes ................................................................. 30
3.4.4. Code complementing ................................................................. 30
3.4.5. Query Execution ....................................................................... 30
3.4.6. Obtaining the object detail ......................................................... 30
3.4.7. Drag’n’drop .............................................................................. 31

4. FQT tool comparison with existing tools ............................................. 32

5. Summary .......................................................................................... 33

6. User manual .................................................................................... 34

6.1. Installation and running the program .............................................. 34

6.1.1. Program parameters ................................................................. 35

6.2. Connect to database ....................................................................... 36

6.2.1. Remove/update of saved connection ........................................... 40

6.2.2. Another connection opening ...................................................... 41

6.3. First Query ................................................................................... 41

6.4. Code completing ........................................................................... 42

6.5. Templates ..................................................................................... 44

6.6. Relations ....................................................................................... 46

6.7. Object details ............................................................................... 48

6.8. Execution plans ............................................................................ 48

6.9. Working with files ......................................................................... 49

6.9.1. File saving ................................................................................ 49

6.9.2. File opening ............................................................................. 50

6.10. Tabs ............................................................................................. 50

6.11. Forced operations ....................................................................... 50

6.11.1. Forced statement execution ..................................................... 51

6.11.2. Forced statement highlighting ............................................... 51

6.12. Program help ............................................................................. 51

6.13. About program dialog ................................................................. 53

7. CD Content .................................................................................... 54

8. Bibliography .................................................................................... 55
Název práce: Free Query Tool – nástroj pro dotazování do SQL databází
Autor: Martin Šuška
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Michal Kopecký, Ph.D.
e-mail vedoucího: kopecky@ksi.mff.cuni.cz

Abstrakt:
Práce popisuje návrh a implementaci nástroje pro dotazování do SQL databází (SQL konzolu), který ulehčí práci zkušenému SQL vývojáři. Především zefektivní psaní SQL dotazů ve volném textu tím, že kontroluje jména již napsaných objektů (tabulek, pohledů, ...), tím, že doplňuje jména objektů právě rozepsaných. Nástroj taky zvýrazňuje syntaxi dotazů, pro lepší orientaci v kódu i včasné upozornění na chyby.

Klíčová slova: SQL editor, SQL konzole, databáze, Oracle

Title: Free Query Tool – tool for enhanced querying SQL databases
Author: Martin Šuška
Department: Department of Software Engineering
Supervisor: RNDr. Michal Kopecký, Ph.D.
supervisors e-mail: kopecky@ksi.mff.cuni.cz

Abstract:
Thesis describes design and implementation of the tool for querying SQL databases (SQL console), that makes querying for senior SQL developer a lot easier and more efficient. Tool checks object names written by user (tables, views, ...) and enables partially written object names completing. Tool, for better orientation in written code, highlights syntax of queries what helps also as error prevention.

Keywords: SQL editor, SQL console, databases, Oracle
1. Introduction

1.1. Motivation

Database user needs are different depending on their role. The needs of DB administrator, database designer or SQL developer are quite different. Despite of great number of visual tools, a lot of advanced users use one of the plain-text editors that use syntax highlighting for writing and debugging their SQL code.

When user writes and tunes SQL statements, he spends a lot of time with looking up object names. He does not remember well table names, table column names, view names and so on. When user wants to join tables he is looking for primary or foreign keys, when he uses filters he would like to know the existing indexes that table contains. When he executes long-lasting query, he would like to know execution plan of such query. Overall the user writes same code snippets all the time.

1.2. Thesis structure

Next chapter analyzes user roles, their needs and possibilities how to makes their work easier and more effective. It also analyzes already existing tools – SQL consoles and SQL editors regarding to requirements. On the basis of analysis, goals and their priorities are determined. Third chapter describes application design and implementation. Thesis summary compares created program with the existing ones, provides an overall review and suggests further development possibilities.

The end of the thesis contains installation guide together with a user manual and lists the content of enclosed CD.
2. Analysis

As mentioned before there exists different roles of users that regularly work with databases. Ordered accordingly to the database application lifecycle, at the beginning there is a database designer, next role is a developer who develops database middle tier using designed schema. Even when the database application is used in production environment it needs to be further supported. Supported not only by fixing bugs and implementing new functionalities, but it needs to be monitored from the view of resource requirements of the growing database, primarily I/O operations, CPU usage or size of used memory. These tasks remain for DB administrator (HW configuration) together with middle tier developer (SQL queries optimizations, indexes alternation). All these roles have different requirements about work with the database. This thesis is focused primarily to database middle tier developer, who is not the beginner and has experience with development of the SQL code.

We assume good users knowledge of target SQL platform. Such users work with the database during the biggest part of database lifecycle, so there is the best opportunity for improvement. We assume that user wants to edit his code in similar way he is accustomed from other text editors. There have to be possibility to edit more scripts at the same time with functionality of syntax highlighting. We assume that user is interested in database structures defined in database, data stored in database and the data relations. While user is working, he is creating new database structures, alters the existing ones and writes queries and commands that store or show data. Selected data can be of course joined from more tables, could be aggregated or modified in other ways. While user is writing queries he ineffectively spends a lot of time by looking for table names, table column names, columns of primary, candidate and corresponding foreign keys in tables and so on. User usually also makes a lot of mistakes or typing errors whether it’s caused by unfamiliarity with actual database structure or inattention. When table name contains more words, user can hesitate if there are used underscores or not. Another source of naming problems is the case when the column with the same meaning is named differently in different tables, for example ID, OID, CODE, etc. There could be more reasons why it is so – missing
naming conventions, user can work with more databases/schemas or the user is working with a really huge database (considering number of entities). Implemented program, result of this thesis, should solve or at least help with solving such problems.

2.1. **Software future requirements**

There are a lot of database developers that do not use MS Windows as their operating system, they prefer to use other operating systems like Linux or Mac OS. The percentage of such users can be quite bigger comparing to regular users of the database application itself.

Querying different databases like Oracle, MySQL, Sybase is really common and so there is no reason, why the tool for database querying should work just with one database. Tool should be at least easy extensible for using other databases. I was testing and developing the tool primarily for Oracle databases with regard to future extensibility for another database.

The tool should be available for all people that are interested in such tools, so I used open source model. I would like to provide the product that users like, because the things implemented in my tool works correctly. The GUI should be simple and functional.

2.2. **Query writing effectiveness**

My experience with database applications development is, that skilled user do not really needs graphical visualizations of his queries. Visualization in most of cases leads to waste of their time, it spends their time instead of saving it. Query visualization could be interesting for application documentation, but not for really effective query writing. It could help the beginner user to create queries, but it’s not the efficient way to develop applications. It is the truth that visualization is not producing syntactic errors, on the other hand the visualization is limited and might lead to not optimized solutions, what should be a critical fact in the database application. Let’s tell we have table of employees (emp), containing the hire date for
every employee. We would like to retrieve not the first five employees but the 6th to 10th employee from such table. Corresponding query in the Oracle database is:

```
SELECT * FROM (
    SELECT rownum, e.*
    FROM emp e
    ORDER BY hiredate
) WHERE rownum BETWEEN 6 AND 10
```

If we would like to obtain such users with respect to some specific ordering, the query will be even more complicated. It is difficult to imagine that query visualization could make such query creation quicker and the result would be comprehensible. In fact this query is not the most complex one and still the visualization fails.

Similarly I think, that advanced user does not expect the possibility to create all the queries using the mouse. Mouse could be the great help in huge number of applications (operating system GUI is a good example), but it’s not so helpful, when the work subject consists in code writing. I don’t believe that the mouse utilization could be quicker comparing to keyboard writing, even in case when the application allows massive drag’n’drop support. Some simple operations as select all from table, could be quicker, but generally it’s not the truth. Especially conditions defining cannot be quicker even when user use mouse in one hand and is typing keyboard with another one.

Described techniques really exist in some tools. See the following figure*, showing visualizations used in Cognos:

![Visualizations in Cognos](http://www.sybase.com/content/1042113/query_outer_joins1.gif)

* downloaded from http://www.sybase.com/content/1042113/query_outer_joins1.gif
SQL developer* for example allows user to define new table, and other operations, by filling informations in wizards, but as I wrote above, I do not consider these techniques as effective.

Techniques that I consider to be most helpful are 2.2.1 syntax highlighting, 2.2.2 drag’n’drop, 2.2.3 code completing, 2.2.4 work with database metadata and 2.2.5 templates.

2.2.1. Syntax highlighting

Syntax highlighting is a technique for text parts coloring. Relevant parts of the text are highlighted, that means that color or style of the text is changed. That helps user to identify errors in written code yet before query execution. Typically keywords, comments, strings and other text parts are highlighted. If user makes typing error for

example in keyword, keyword is not highlighted and user finds the error a lot quicker.

Syntax highlighting is useful in two cases (both are connected with common idea of error prevention):

- context highlighting

Consider following example:

```
SELECT *  
FROM emp  
AND sal > 1000;
```

When this code is executed in the Oracle database, the following error occurs:

ORA-00933: SQL command not properly ended

I’m convinced that every user checks the end of the query, see the ‘;’ character and is confused because the query is correctly ended. Problem in that query is in missing **WHERE** keyword, instead of it there is **AND** keyword, which is in many tools incorrectly highlighted. If the command is highlighted correctly user fixed the problem yet before query executing.

- typing errors that users make often

Users often make the same mistakes – typing errors. It happens that user writes **SEELCT** instead of **SELECT** keyword or **FORM** instead of **FROM** keyword. When tool for SQL code writing has the highlighting functionality user realizes and corrects the error almost immediately. There is bigger problem with **FORM**, because it’s a keyword in Oracle too and it is usually also highlighted. Solution for this problem is context-free grammar, not only the regular definition of words to be highlighted.

### 2.2.2. Drag’n’drop

Drag’n’drop (drag&drop, DND) consists of clicking on a virtual object in the GUI and dragging it to a different location or onto another virtual object. In general, it can be used to invoke many kinds of actions, or create various types of associations between two abstract objects.
This technique makes it easier to open file (file is simply dragged to application). Drag’n’drop technique can be used in querying for dropping the table name or table column name into query. Also already selected values can be used in where clause of the query. There is a lot of possibilities how this technique can be helpful.

### 2.2.3. Code completing

Code completing is a technique how to make querying more effective. When the user invokes this functionality editor tries to guess what is the user trying to write. In SQL queries writing there is opportunity for doing so. Queries have some structure and it is good presumption to guess what’s coming next. If there is more possibilities, the list can be shown to user and he can choose the correct one.

Code completing can be implemented based on two principals:

- **code completing based on grammar**

  When the grammar is used for code completing, in dependence on the grammar we know what is expected – it is the table or view name after `FROM` keyword or expression in `SELECT`, `WHERE` or `GROUP BY` clauses. When function name is entered, code completing can show list of parameters.

- **code completing based on templates**

  User can define templates besides grammar. Application can either contain build-in templates or allow user to define his own templates. For example if the user uses `to_char()` function a lot, he can define “tc” template. When the code completing is invoked the token tc can be easily replaced by `to_char(sysdate, 'yyyy-mm-dd')` function call.

### 2.2.4. Database metadata retrieving

Database systems serve primarily for data storing and provide functions for data managing. Besides own data, database systems usually gather data about data stored in database so-called metadata. These metadata typically contain lists of database objects (tables, views, …) and another information as table columns, integrity constraints and a lot more. Tool for database querying should support retrieving of these metadata and use them for drag’n’drop or code completing.
2.2.5. Templates

Templates are code snippets written automatically when user invokes such functionality for example when key shortcut is pressed. SQL language is suited for templates. SQL language has defined structure and this structure can be generated which improves efficiency of code writing.

Templates functionality can be further improved by allowing selected text to be part of template expansion. If there is select command

\[
\text{SELECT sysdate FROM dual}
\]

Template can be defined in such a way that when the user presses corresponding key shortcut and some text (i.e. sysdate) is selected, the text sysdate is replaced with \(\text{to_char(sysdate, 'yyyy-mm-dd')}\) text.

2.3. Existing tools

There exist a lot of tools, ordinary editors or specialized querying consoles that provide better or worse support for SQL queries writing. With respect to presumed user request for compound of superior editor with SQL console this part of the thesis presents both existing SQL editors and available SQL consoles. While SQL editors allow user to modify text files, SQL consoles enable user to send commands to be executed in the target database. As I have chosen to implement the database tool primarily for the Oracle database, next sections introduce tools usable for this database. All the tools I used for testing were used in default settings.

2.3.1. SQL consoles

There exist more SQL consoles for querying Oracle databases. I concerned on testing the freely available ones.
2.3.1.1. SQLTools*

It is small tool intended for developing Oracle applications. SQLTools program is available only for Windows OS and it is available under GNU GPL†. Program uses OCI‡ for connecting to database.

I tested version 1.4.3, marked as a last stable at the time of writing this thesis. SQLTools introduces really good work with drag’n’drop.

![SQLTools screenshot](image)

Program supports dragging table column names or selected values from retrieved result set (1) or object name from list of objects (2) or column name from details panel (3) to the query. There are small inconsistencies in drag’n’drop.

* [http://www.sqltools.net/](http://www.sqltools.net/)
† [http://www.gnu.org/licenses/](http://www.gnu.org/licenses/)
‡ Oracle Call Interface – API pro komunikaci se serverem, dodávané spolu s klientskými aplikacemi Oracle.

15
implementation. When the column name is dragged from result set table the text is copied in upper case, but when column name is copied from detail panel, text is copied in lower case.

Dragging object name to details panel doesn’t work from SQL area. For retrieving object details a shortcut F12 is defined, but this shortcut is not working from list of objects.

SQL editor in SQLTools program supports syntax highlighting for keywords that are transformed to underlined upper case letters. Function names are underlined and capitalized. Syntax highlighting is implemented using final automata. Every keyword written wherever in SQL area is highlighted. Here is statement example

```sql
SELECT * FROM dept AND deptno = 10
```

in which `USER` word is function without parameters, but is highlighted as keyword. Similarly sometimes happen when I’m writing command that when I’m writing command:

```sql
SELECT DISTINCT procedure_name
FROM all_procedures
ORDER BY procedure_name
```

if character ‘_’ after procedure word is replaced by mistake with space character what is later fixed, identifier in mixed case (PROCEDURE_name) shows.

Code completing functionality is missing.

There exist templates in SQLTools program, invoked by Ctrl + Spacebar shortcut. I’m using this shortcut in other programs for code completing functionality, so I use to call this a lot. When I wrote in query above the prefix “pro” (beginning of procedure_name) and invoked the code completing, it replaced “pro” with a template for empty procedure and it wasn’t what I wanted. This definitely is not the program error, I’m just accustomed to use this shortcut for something else. I didn’t find the way how to change this setting.

There exist templates in SQLTools program, they are invoked by Ctrl + Spacebar shortcut, but it caused more errors for me. I’m using this shortcut in other programs for code completing functionality, so I use to call this a lot. When I wrote in query
above pro (beginning of procedure_name) and invoked code completing it replaced pro with template for empty procedure and it wasn’t what I wanted.

Another compared functionality was the ability of the program to recognize the commands. SQLTools correctly recognizes last entered command. If there are more commands to run, previous commands need to be terminated by ‘;’ character, empty line between commands is not enough. If there are two commands in the same line separated by ‘;’, execution of such commands ends with error ORA-00911: invalid character.

Functionality for execution plan retrieving is invoked by F9 shortcut, but this functionality needs extra plan_table in the database.

Tools also do not know the desc command.

2.3.1.2. SQuirreL*

SQuirreL is written in Java and so it is platform independent. It supports connection to many different databases and supports different ways how to connect to database. For example for Oracle database it supports connection using JDBC, OCI, and Java5Driver.

I tested version 3.0.1. Comparing to SQLTools it supports code completing functionality. If there is an alias defined for table and code completing is invoked, help shows just columns for this table. If the alias is not defined, code completing shows also other possibilities like function names, but when such function is selected, for example sysdate, text {call SYSDATE()} is written to the SQL area.

Syntax highlighting works a lot better comparing to SQLTools program, for example in command:

```sql
SELECT * from
```

form word is marked as incorrect (because expected keyword FROM is missing). When I tested the tool using user function:

```
```

* http://www.squirrelsqll.org/
program works as incorrectly as SQLTools. The user word is incorrectly highlighted as a keyword. When I tested table creation command:

```
CREATE TABLE test (  
  num int,  
  txt varchar2(100),  
  datum date  
)
```

only the date datatype was recognize, int wasn’t highlighted and varchar2 datatype was highlighted as incorrect input. Program distinguishes existing and not existing objects in from clause. Highlighting has also another small error – if there is a column name alias without AS keyword, alias is marked as wrong input.

Drag’n’drop functionality in SQuirreL is not supported.

Command recognition works differently from SQLTools. If two commands are separated only with only a new line, tool – correctly – fails. If the ‘;’ character is added, then both commands are executed. If there is ‘/’ character used for command termination, the command execution fails with error message ORA-00933: SQL command not properly ended. Statements with comments at the end are executed correctly. Desc command is not supported.

When I was using this program I didn’t like the behavior where all command results are shown in separate tabs. This feature used a lot of memory. If these results are closed, the amount of memory doesn’t decrease.

The biggest problem using this tool I found was the performance problem. When there was a bunch of commands it starts to freeze when editing other commands.

Templates in SQuirreL program are working as described in 2.2.3 code completing chapter. When user writes sf and invokes code completing, statement:

```
SELECT * FROM
```

is generated.

Exporting result set to .csv or .HTML files worked fine, just exporting as SQL statements for timestamp columns generated constants in the ODBC format.
and other tools have problem recognizing this commands, because it’s not correct SQL statement.

2.3.1.3. SQL developer*

SQL Developer is developed by Oracle company. It has plugins for other databases, for example for MySQL. Tool should be platform independent, because it’s also written in Java, but when I was trying to download that tool I had to choose the platform. When I compared downloaded files for Windows OS and distribution for other operation systems, both archives were the same. Probably the difference is only in archives, containong RPM package for Linux, or tar.gz archive for Mac OS. Program needs JDK instead of JRE. It’s caused by debugging of PL/SQL code support.

Syntax highlighting functionality is somewhere between SQLTools and SQuirreL programs, because in statement:

```
SELECT * form
```

the word form was highlighted.

Code completing works fine, comparing to Squirrel. When code completing is invoked in select clause without prefixes, system functions i.e. sysdate are not shown. Code completing for inner select shown at least ‘*’ comparing to Squirrel that didn’t show any possibility.

Result exports works correctly except for dates. When the date column is exported as insert statement, values in statement are formatted according to current NLS† settings, so in my database it cuts off the hours, minutes and seconds.

I didn’t like the long starting of the program that is much longer comparing to other tools. When Squirrel was connected to the database it takes a long time downloading the metadata, but SQL developer starts long time without connecting to


† National Language Support
database. It happened several times that SQL developer starts to freeze without any reason and sometimes it frozen completely and all the work was lost, that’s the reason why a lot of people do not like it. On the other hand I have to admit that SQL developer contains a lot of prepared functions to be used just by clicking.

2.3.1.4. TOAD

This paid tool is developed by the Quest Software and it is one of the best tools available. It has a lot of modules as Debugger, DB Admin module and SQL Optimizer module, but I think it’s popularity comes from the fact it was available before free SQL developer developed by Oracle. TOAD is just for Windows. I was testing Trial version of the TOAD 9.7.2.5.

Code completing (called Code Assist in TOAD) is invoked automatically when user types ‘.’ character after object name or ‘(’ character after function name. Code completing is so weaker because it requires using table aliases and schema names.

Syntax highlighting in TOAD incorrectly highlights statements:

```sql
SELECT * form
SELECT user FROM dual
```

words user and form are incorrectly highlighted as keywords. It is also apparent that final automaton is used for highlighting. Statement for table creation from chapter 2.3.1.2 SQuirreL knows all the datatypes – int, varchar2 and number.

Drag’n’drop is supported to be used from list of objects panel (it is also possible to drag’n’drop column names), but it’s not possible use drag’n’drop from result set table.

I was surprised this tool returns only 30 rows from result set and I didn’t find the setting for changing this. I’m convinced that it is possible to change this setting, but

* http://www.toadsoft.com/

† http://www.quest.com/
when I tried to look in setting I simply didn’t find it. In other tools – SQuirreL I had no problem in doing so. That only shows it’s not intuitive as other tools are.

Tool handled export to HTML, CSV and insert as statements as the best tool comparing to other tools. When I changed the date format shown in result set table, insert statements generate by the tool for dates were correct.

2.3.1.5. Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SQL Tools</td>
</tr>
<tr>
<td>Portability</td>
<td>Windows</td>
</tr>
<tr>
<td>Drag’n’drop</td>
<td>Excelent</td>
</tr>
<tr>
<td>Code completing</td>
<td>No</td>
</tr>
<tr>
<td>Syntax highlighting</td>
<td>Yes</td>
</tr>
<tr>
<td>Result set export</td>
<td>Insert statements export is not supported, works fine.</td>
</tr>
<tr>
<td>Other features summary (besides features mentioned above in table)</td>
<td>Tool has really small size what I rate as positive property.</td>
</tr>
</tbody>
</table>

Table 1 existing tools comparison

2.3.2. Editors

This section describes editors that lack consoles abilities but still facilitate the user work.
2.3.2.1. Kate*, PSPad†, vi‡
These editors, as well as many others, support syntax highlighting. In spite of they
cannot work with metadata, they are not primarily developed for databases and they
use final automata for highlighting, they are still very often used by the developers.

```
select * from emp e
where e.deptno = 30
```

Figure 3 syntax highlighting in Kate editor

```
select * from emp e
where e.deptno = 30
```

Figure 4 syntax highlighting in PSPad editor

```
select * from emp e
where e.deptno = 30
```

Figure 5 syntax highlighting in vi editor

2.3.2.2. Eclipse
Eclipse is an integrated development environment (IDE) for Java and other
programming languages. It supports not only syntax highlighting, but also advanced
error reporting together with hints for their fixing. It also provides the user with
many possibilities regarding code generating and code completing.

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Figure 6 syntax highlighting and code completing in Eclipse IDE

* http://kate-editor.org/
† http://www.pspad.com/
‡ http://www.vim.org/
2.4. **Requested features for implemented program**

Summary table of the features that the tool have to implement:

<table>
<thead>
<tr>
<th>Feature requirement</th>
<th>Tool implemented in this thesis</th>
<th>Tools that meet the requirement</th>
<th>Other tools from thesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform independent</td>
<td>Yes</td>
<td>SQuirreL, SQL Developer</td>
<td>SQL*Tools, TOAD</td>
</tr>
<tr>
<td>Syntax highlighting</td>
<td>Yes</td>
<td>All</td>
<td>-</td>
</tr>
<tr>
<td>Syntax highlighting based on grammar</td>
<td>Yes</td>
<td>SQuirreL</td>
<td>SQL*Tools, SQL Developer, TOAD</td>
</tr>
<tr>
<td>Code completing</td>
<td>Yes</td>
<td>SQuirreL, SQL Developer, TOAD</td>
<td>SQL*Tools</td>
</tr>
<tr>
<td>Drag’n’drop</td>
<td>Yes</td>
<td>SQL Developer, SQL*Tools</td>
<td>SQuirreL, TOAD</td>
</tr>
<tr>
<td>Multi database support</td>
<td>Yes</td>
<td>SQL Developer, SQuirreL, TOAD</td>
<td>SQL*Tools</td>
</tr>
<tr>
<td>Execution plan</td>
<td>Yes</td>
<td>All</td>
<td>-</td>
</tr>
</tbody>
</table>

*Table 2 requested features for implemented program*
3. Design and implementation

3.1. Basic principles

In this chapter I will describe main decisions on which I would like to establish my case for querying.

- Program shouldn't be intended only for a specific operating system

  Considering the Java is the most platform-independent platform nowadays, my program is written in Java and thus ensures its portability to Windows, Linux, Mac OS and other systems for which the JVM is implemented.

- Program won't be intended for only one database

  Despite the fact that I will write a program specifically for Oracle, the program will be designed in such a way that its extension to other database will be the least complicated.

- The user interface of the program should not be too difficult

  Because I believe that the beauty is in simplicity and the program should prove it's quality mainly by its functionality, I would like to create a program that would show evidence of well functioning.

- Syntax highlighting based on grammar

  As revealed from the comparison, it is more correct, when the program makes highlighting of syntax using language parser and not only by simple automaton.

I chose Java version 1.6, because lower versions (1.4 and lower) do not maintain generic types, which I consider very useful when writing code, because it enables better type control and thus reduce the number of errors in the program itself. During the development it showed that version 1.5 creates some obstacles when implementing of some functionality.

I chose Oracle because it is the most common database, and thus the program could attend many users.
Other significant feature of the program would be its unpretentiousness for maintenance. Its installation and configuration should be transparent – ideally it should be enough to extract it to the target directory. This way the user exactly knows where the program and all its files are located. On the other hand, the goal is to provide optionally saving the configuration files in user profile.

The response of programs that communicates with the database according to the user instructions is unfortunately tied with responses of the database we are working with. These response times are influenced for example: by the connection throughput we are connecting with, by database workload in the query time but also by the quality of our own queries. The program can not be limited to sending only optimal queries to the database and cannot have a target to accomplish all commands within to the certain limit. On the other hand the program shouldn't forbid further working with the program, even during time-consuming queries. It should be possible to switch to another tab, edit the content in the same tab, explore the database metadata, run more queries at the same time etc. In the same session the queries will wait each other, in different sessions queries will run in parallel.

For the implementation of the SQL language parser I select a parser generator ANTLR. When choosing, I focused mainly on language for which the parser is determined and consequently the available documentation, because I hadn't previous experiences with any tool for generating grammars in Java.

### 3.2. Conception

From the fact that the program is not intended for only one database follows that it must contain a part which is dependent on the chosen database – on Oracle. This database dependent part should be as small as possible because it must be rewritten or extended whenever new database support is implemented. The rest of the application should be database-independent and communicate with the database using clearly defined interfaces. Into the database-independent part belongs e.g. GUI, background processes (it's just naming, background processes are implemented as threads) and interfaces for database dependent part.
The database dependent part includes mainly parser for the given grammar and the part responsible for extracting metadata from the database.

Figure 7 FQT modules

The figure shows the program modules. At the lowest level is an Oracle database with which the program communicates using JDBC. In the picture you can also see how is the program divided into the database dependent and independent parts. These parts communicate through interfaces. The arrow indicates the start of the program by user. After program execution the main program window is displayed. Before the first attempt to establish the connection to the database the database module is initialized. Using reflection instances of classes, which will be responsible for future SQL commands parsing are initialized according to saved settings. After connection to the database the area for writing SQL queries becomes accessible for user. For each change in this area the SQL parser for the specified text is called. The task of the parser is to recognize given command and support the database independent part with informations necessary for its correct highlighting.

3.3. Database dependent module

Database dependent module of application contains sub–modules for SQL parsing, subsequent work with parsed command and obtaining the metadata from the
database. It implements the interface `IModule` for communication between database dependent and database independent modules of the program. Except the cooperation between sub-module working with parsed command and sub-module for obtaining metadata, all events are coming from the database independent module.

### 3.3.1. Communication with database independent module

Sub-module for working with database communicates with an independent module only in the case when it tries to determine whether there is cached metadata or not during the work with parsed command.

Entire database independent module is going to be initialized before the first connection to the database. This is done by the settings in file `module.properties`, described in the following section. Subsequently, when the user changes a command in command editor area, SQL lexer receives the part of the user written code and performed its lexical analysis. The result is passed to SQL parser that creates the parsed SQL command, which implements interface `IParsedCommand` and it is returned to an independent database part as a result of user changes.

Part of text which lexer and parser have to analyze is given by area for writing SQL commands and it is determined by finite automaton. It refers to a minimum text part of the entire area for writing code, to prevent of parsing entirely the whole area. Although it is not send a whole area, but only its necessary minimal part, it cannot be expected that it’s going to be parsed only one command. Due to the fact that you can (eg by insertion from clipboard) insert more than one character at the time and parsing should be effective, the parser is running only once - at the end and not after each inserted character.

### 3.3.2. Configuration file `module.properties`

In this file modules parameters are defined:
<table>
<thead>
<tr>
<th>Parameter*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModuleNames</td>
<td>Designates modules defined in configuration file. It is list of comma separated values.</td>
</tr>
<tr>
<td>.lexer</td>
<td>Canonical name of class which implements interface I Lexer.</td>
</tr>
<tr>
<td>.parser</td>
<td>Canonical name of class which implements interface I Parser.</td>
</tr>
<tr>
<td>.commandParser</td>
<td>Canonical name of class which implements interface ICommandParser.</td>
</tr>
<tr>
<td>.module</td>
<td>Canonical name of class which implements interface I Module.</td>
</tr>
<tr>
<td>.metadataRetriever</td>
<td>Canonical name of class which implements interface IMetadataRetriever.</td>
</tr>
<tr>
<td>.driver</td>
<td>Database connection driver name</td>
</tr>
<tr>
<td>.connectString</td>
<td>Defines format of connection string for concrete module. It is parametrized for host name, port and database name</td>
</tr>
<tr>
<td>.defaultType</td>
<td>Defines which types of database objects have to be shown by default.</td>
</tr>
</tbody>
</table>

Table 3 module parameters description

3.3.2.1. Settings for Oracle module example

ModuleNames=Oracle

Oracle.lexer=net.freequerytool.modules.lexers.OracleSQLLexer
Oracle.parser=net.freequerytool.modules.parsers.OracleSQLParser
Oracle.commandParser=net.freequerytool.dbmodules.DefaultCommandParser
Oracle.module=net.freequerytool.modules.OracleModule
Oracle.metadataRetriever=net.freequerytool.modules.OracleMetadataRetriever

Oracle.driver=oracle.jdbc.driver.OracleDriver
# connection String to DB, where
# 0 is host name or IP address
# 1 is port number
# 2 is SID
Oracle.connectString=jdbc:oracle:thin:@\{0}\:{1}\:{2}
Oracle.defaultType=TABLE

* each parameter except the first has a prefix formed from module name for which the settings is valid
3.4. **Independent part of the database**

This section is formed by program’s GUI mainly with which the user works. How to work with the program is described in the 6 User manual chapter at the end of the thesis. User interaction with the program generates events, which are subsequently processed. If the independent part of the database has no capability to process the event, these events are forwarded to dependent part of the database, or it generates a new event for this part.

3.4.1. **Connection to the database**

Description of how the user connects to the database is in chapter 6 User manual. In this chapter I will focus on what must be done for connection to the database.

Just before connection dialog is shown to the user, it is required to load module properties and load modules defined in `module.properties` configuration file. Depending on loaded modules, connection dialog contains combo box to define which module has to be used for connection. Already saved connections are loaded from the file. If the passwords were saved for connections, they are loaded from separate password file, which contains passwords encrypted by hill’s cipher.

3.4.2. **Writing SQL commands**

After connecting to the database, it’s opened a new tab in the new panel for writing SQL code for user. The tab includes a section for editing the SQL code and a section for displaying the results. The area for writing SQL has assigned a parser according to initialized module and any change in the area generates new requirements for the command parsing.

**Example:**

Suppose that user has written a start of a SQL command:

```
SELECT * from ...
```

When the user appends to the command letter m, then SQL area finds the beginning and the end of the command. As it was entered only one letter, the program knows that it is only one command to be parsed and the command is sent for analysis to the parser of the SQL language for a particular module. As described above, parser returns as a result an instance implementing `IParsedCommand` interface that
represents the analyzed command. In parsed command are marked significant command parts as keywords, comments, identifiers, strings, etc. and these can be highlighted by highlighter.

3.4.3. Background processes
To prevent freezing of the program by calling long-lasting actions and not to stopping to respond, it was developed the concept of processes running in the background. Some of the tasks, which must the program fulfill and which may take a long time – e.g. connecting to the database, executing the query, etc. All potentially long operations are performed in the background and to the user is displayed a window which shows that the task is still running. On one hand we want restrict the user and not let him e.g. run by mistake two queries at the same time in the same session, because by calling the second query he can ruin the result of the first call. On the other hand, we do not want the user to lose the possibility of working with the program and thus with another database somewhere in another tab. Another thing that can be done by the user is stopping the task he runs before.

3.4.4. Code complementing
If the user invokes code completing, then – according to the position of the cursor – the command which is being edited is found and requirement for obtaining help for the query is forwarded to a specific parsed command. Because the program knows the statement structure it can decide for example if column names or names of the tables should be provided. The necessary information is obtained either from metadata cache or (if the cache doesn’t contains the necessary data) from the database using IMetadataRetriever instance for obtaining metadata from the database. All obtained metadata are stored afterwards in the cache.

3.4.5. Query Execution
A command for execution of a query is sent to the correctly initialized instance of background process, which executes the query in another thread. The query text position in the text area is recognized simply according to the cursor position.
3.4.6. Obtaining the object detail

By invoking the functionality for obtaining details about an object the program finds the command under the cursor and according to the position of the cursor within the command details about required object are taken from metadata cache or by calling the database.

If the user selects the text, which doesn’t correspond to any object in the database – it’s the key word or text string for example, the program will notify this and will not make any database query to retrieve metadata. In case that shown details seem to be incorrect to the user (e.g. because they are outdated) they can enforce the refresh of them. In this case the data are not taken from the cache but the database is contacted, thereby the value in metadata cache is restored.

Invoking function manually by users is not the only way to obtain details about the object. Details are obtained also for helping user with code complementing. Although the same function is called, its invocation is triggered by other action.

3.4.7. Drag’n’drop

Drag'n'drop functionality is almost entirely implemented in the database independent module of the program, namely GUI. If the drag'n'drop functionality needs a database metadata for its operation, it asks the database dependent module to obtain them either form the cache or from the database. Whenever user uses drag'n'drop functionality to change the text in the SQL area, the system processes the event the same way as if the text were written manually or pasted from the clipboard.

It means namely that this event triggers both parsing and highlighting the text, as it was described above.
4. FQT tool comparison with existing tools

The biggest contribution of the tool is the implementation and using full-featured parser. It’s used for main functionalities as syntax highlighting and code completing Tool overcomes existing tool in syntax highlighting, what can be demonstrated on examples like:

```
SELECT * FROM emp and deptno = 10;
```

where and keyword is not highlighted, because where keyword is missing.

```
SELECT sysdate FROM dual
```

where sysdate is correctly recognized as function

```
SELECT dummy „select“ FROM dual
```

SQuirreL highlighted second occurrence of select word in this command and because of this the question about using final automaton or grammar remains opened.

Comparing the created tool with SQL Developer or SQuirreL tools would be too risky, but on the other hand this tools have long history and they are developed by many people. SQL Developer is based on JDeveloper* which is developed by Oracle since 1998†. SQuirreL SQL client is a bit younger – the SourceForge project was established in 2001‡.

---

† [zdroj http://www.orafaq.com/wiki/JDeveloper](http://www.orafaq.com/wiki/JDeveloper)
5. Summary

 Implemented tools fulfilled goals of the thesis. Oracle SQL grammar was implemented in the tool and its parser is used in tasks that makes query writing more effective as proper syntax highlighting, code completing etc. The tool combines featured plain-text editor together with an SQL console that directly invokes written content in the database.
6. User manual

6.1. Installation and running the program

There is no need to install program. It needs to be just unzipped from archive and it is ready to run. Program is available at CD (/FQT-1.0.zip) that is attached to the thesis and also at

http://fqt.googlecode.com/

To run program double click on start_FQT.bat file (if you are using Windows OS) or start_FQT.sh file (in Linux). These files simply contain just one command to run FQT program:

```
java -Dfqt.home="..." -Dlog4j.debug=false -cp
lib\FQT.jar;lib\FQTCommon.jar;lib\FQTOracleModule.jar;lib\FQTMySQLModule.jar;lib\antlr-3.1.1.jar;lib\log4j-1.2.15.jar;lib\ojdbc5.jar;lib\swing-worker-1.1.jar
net.freequerytool.core.Main --lang_dir="lang" --lib_dir="lib" --md_dir="metadata" --prop_dir="props"
```

where … characters are replaced with actual working directory. Program needs Java runtime environment (JRE) in version 1.6.0_03 or later to be run. Linux users need also X window system. You do not need Java development kit (JDK) but it is possible to run FQT using it as well. Check if you have java installed on your computer by running `java -version` command from command line (cmd for Windows or shell in Linux). Example of calling this command should show output similar to listed one when java runtime environment is available:

```
$ java -version
java version "1.6.0_13"
Java(TM) SE Runtime Environment (build 1.6.0_13-b03)
Java HotSpot(TM) Client VM (build 11.3-b02, mixed mode, sharing)
```

Program was tested using 1.6.0_03 JRE. In case you have not JRE installed on your computer you can download it for free at
When the program is ready, following window is shown:

![FQT welcome page](http://java.com/getjava)

Windows contains brief help information what to do to connect to database.

### 6.1.1. Program parameters

Table describes FQT program parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Dfqt.home=[path]</code></td>
<td>Specifies the directory for home directory of FQT program. All subdirectories of FQT program have to be subdirectories of this directory.</td>
<td>Working directory, by default it is set by script to the result of <code>dir</code> command call (in Windows) or <code>pwd</code> (in Linux).</td>
</tr>
<tr>
<td>`-Dlog4j.debug=[true</td>
<td>false]`</td>
<td>Parameter for debugging logger subsystem debugging.</td>
</tr>
<tr>
<td><code>-lang_dir</code></td>
<td>Directory containing language mutations for program (relative path to <code>lang</code>)</td>
<td>lang</td>
</tr>
</tbody>
</table>
### Parameter Description Default values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-lib_dir</td>
<td>Directory contains FQT program libraries and module library (relative path to fqt.home directory).</td>
<td>lib</td>
</tr>
<tr>
<td>-md_dir</td>
<td>Directory containing cached metadata (relative path to fqt.home directory).</td>
<td>metadata</td>
</tr>
<tr>
<td>-prop_dir</td>
<td>Directory containing FQT program property files (relative path to fqt.home directory).</td>
<td>props</td>
</tr>
</tbody>
</table>

Table 4 FQT program parameters description

The most important parameter is the first one (fqt.home). It specifies the directory in which other directories are located. When program is starting first of all logger subsystem is initialized, parameter log4j.debug specifies if this initialization have to print messages about initializations.

### 6.2. Connect to database

Started program shows brief help containing information what to do to connect to the database. User needs to open connect dialog using left Alt + C shortcut or by using Tools > Connect option from the menu.

Following dialog is shown for connection:
At the right side of the dialog:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name:</td>
<td>connection name to save connection with</td>
</tr>
<tr>
<td>host:</td>
<td>computer (database is running on) to connect to it is allowed to enter IP address or computer name</td>
</tr>
<tr>
<td>port:</td>
<td>port where the database is listening on (default for Oracle is 1521)</td>
</tr>
<tr>
<td>SID</td>
<td>database identifier we want to connect to</td>
</tr>
<tr>
<td>User</td>
<td>user (also schema in Oracle database) we want to connect as</td>
</tr>
</tbody>
</table>
| Password | user password  
- if entered it will be saved  
- if not entered user is asked for password always he wants to connect to database |

**Table 5 connection properties**

When save button is pressed, connection is saved to connection table. Following pictures show connection dialog before and after the save button is pressed.
Figure 10 Example of filled connection dialog
In previous example connection name was set as `scott@localhost`, (userName@host) but user do not need to follow this convention. However, connection name have to be unique in connection table. Thus, whenever user tries to save connection with name already present in connection table, warning dialog is shown. User have to choose if he wants to replace existing connection or to cancel saving.
To connect to database user have to choose saved connection from table and press Connect button. Program creates connection to the database and opens a tab with the title corresponding to saved connection name:

![Figure 13 Just connected to database screenshot](image)

### 6.2.1. Remove/update of saved connection

It might happen that user needs to remove saved connection (i.e. in case he knows that he will not need this connection any more), or that he needs to update existing one. For such functionality dialog contains two buttons – remove and edit. When delete button is pressed, selected connection in connection table is removed from the table as well as from the connection file. When edit button is pressed values from saved connection appear in fields at the right side of the connection dialog. It enables user to edit these values and save changed connection with changed values. Of course there will be warning message shown and user has to confirm changes.

Saved connections are stored in props directory default location of the directory is in FQT home directory.
6.2.2. Another connection opening
FQT program allows user to connect to multiple databases. Program makes difference between multiple tabs using shared connection and multiple tabs using its own connections. Tabs for one connection have the same color, while tabs for different connections have different colors. Active tab has color different from other connections. It is used a round robin algorithm for color assignment. While there is limited number of colors (6), it might happen that there are tabs with same color but different connection.

![Figure 14 Screenshot showing different tab colors](image)

6.3. First Query
SQL query writing is really simple. User starts to write the query to the SQL area. That is the area where is the focus when connection is established.

When user finishes the query, the query can be executed by pressing Ctrl + Enter shortcut. Result of the query is shown in result tab under SQL area. Following screenshot shows such situation:
6.4. Code completing

Code completing functionality is invoked by Ctrl + Space shortcut. If the program has some help available, it shows possible options in popup, for example table column names or table names.
User can select from shown possibilities using keyboard or mouse. Selection has to be confirmed by pressing Enter. If code completing help is not needed, it’s possible to close popup using the Esc key.

Program shows code completing help only when the user invoked this functionality, it’s not invoking the functionality spontaneously in any moment.

It is possible to filter the help by continuing in writing. That means that code completing help is filtered in dependence on written text. In case shown at figure 9 there were names of tables/views shown for ‘de’ prefix. If the user types ‘p’ as the next letter, code completing help is filtered and shows only DEPT possibility (figure 10). When the table is created with mixed case characters, name is enter with double quotes.
6.5. Templates

Templates represent code snippets. The user can define his own templates and bind shortcuts for them. Later, when user is writing the query, using these templates makes his writing more effective. Example of such template could be template bound to left Alt + S shortcut that generates select command:

```
SELECT * FROM
```

or template bound to Alt + W shortcut, which could generate WHERE, keyword or left Alt + G shortcut, for keywords GROUP BY. User can define as many templates as he wants and is limited only by number of unique shortcuts. Templates are defined and manageable in settings. Settings are available either using right Alt + Enter shortcut* or in menu Tools > Settings...

* that generates Alt + Ctrl + Enter key event in Java
For template creation it is required to define template name e.g. ‘generate select’, template text, which have to be generated when template is invoked and shortcut bindings, that invokes template. Shortcut is entered in Shortcut text field. It is not needed to write shortcut text, just press keyboard key and recognized keys are shown in text field. New defined template appears in shortcut settings. Non-standard shortcuts use the 'template – ' prefix in the list.
6.6. Relations

Program allows user also to save so-called relations. Relations could be created for simplification of writing complex queries in the future. Relations are shown as special database objects with object type RELATION. In fact that could be used for client-side views, i.e. views not defined in the database, but remembered by this tool only.

Relation can be saved from selected text in the SQL area using menu option Tools > Create relation. To creation a new relation, the part of the SQL code has to be selected and its name has to be assigned. Corresponding dialog is shown for name specification.

![Figure 20 Relation creation](image)

New Relation is then shown in DB objects panel.
It is possible to modify or delete relations, tab ‘Relations’ in setting dialog serves for this.
6.7. **Object details**

When user is interested in some DB object details, he needs just to move the caret in SQL area on the object name and invoke functionality for object details retrieving – key F3. Panel for DB object details shows retrieved details (shown at figure 16).

![Figure 23 DB object details (DEPT table)](image)

6.8. **Execution plans**

Execution plans are generated from database automatically whenever query is executed. It is also possible to retrieve execution plan for query without its invocation. The shortcut for execution plan retrieving is F9.
6.9. Working with files

6.9.1. File saving
User can save his work if he wants to. There is menu option File > Save file... which opens file browser window where user select directory where the file have to be saved and defines name for file. FQT change the tab name according to file name to signalize that this tab is associated with a file. When the user changes content of the file, tab name is prefixed with '*' character to signalize not saved file. Prefix disappears when file is saved.
6.9.2. File opening
User’s saved work can be opened in the program to continue working on his scripts. To open file there is File > Open file ... option in menu. Invocation of it opens file chooser window in which user can select file or more files to be opened in program. Each file is opened in separate tab. When file is opened, content is shown and highlighted in the SQL area.

6.10. Tabs
FQT supports work with tabs. It enables user to split his work to parts and so to make his work more effective. New tab is opened with Ctrl + N or Ctrl + T shortcuts. To close the tab the shortcut Ctrl + W is present.

6.11. Forced operations
Because databases are evolving or there could be something missing in SQL grammar for some database by mistake and therefore a correct statement could be incorrectly marked as incorrect, FQT introduced forced operations. Forced
operations are the way how FQT wants to prevent user to be unable to work with program. SQL area uses for command locating final automaton, therefore it might happened that it recognize the command incorrectly.

6.11.1. **Forced statement execution**
If the parser is unable to recognize some statement, there is possibility to execute such statement too. It is enabled with forced statement execution what is functionality that enables user to select text he wants to execute and selected text is sent to database for execution – FQT is not analyzing such statement. To execute the statement there are two possibilities. Select the text and invoke statement execution – Ctrl + Enter shortcut or choose menu option – Tools > Force > execution.

6.11.2. **Forced statement highlighting**
For highlighting it is similar. If there is some error in command recognition in FQT, it could happen that FQT is unable to distinguish two commands written in SQL area. In such case FQT tries to highlight two statements as one statement, what is impossible. Highlighting is one of the most important way, how to make user effective. Therefore FQT introduced option how to highlight statement that was not recognized by FQT. User have to select text representing statement and invoke menu option – Tools > Force > highlighting.

6.12. **Program help**
Program help is available to introduce FQT program to its users. It serves as first point where to find information about the program. Program has to allow users to create their scripts in more effective way, program help is there to allow users to use FQT itself in an effective way.
Welcome in FQT Help.

Dear user, I'm glad you chose to use FQT. I hope that this document helps you in using FQT program or answers your questions.

Topics

first steps - new database connection, first query
templates - using templates
tabs - work with tabs
bugs - any problem, let me know

Figure 26 Help dialog
6.13. **About program dialog**

Dialog contains information about program version, autor, copyright and license.

![Figure 27 About dialog](image)
7. CD Content

CD contains zipped (FQT-1.0.zip) and unzipped version of the program (FQT directory), Sources directory contains all source files. Program in default settings is not prepared to be run from read only medium. CD also contains generated Javadoc documentation, also zipped and unzipped version in Javadoc directory. Directory Thesis contains electronic version of this thesis and the root of the disk contains readme.txt describing the content of CD.
8. Bibliography


[JavaDoc] http://java.sun.com/j2se/1.5.0/docs/api/


[LGPL] http://www.gnu.org/licenses/licenses.html#LGPL


[Parr07] Terence Parr. The definitive ANTLR reference


