FACULTY OF SCIENCE & TECHNOLOGY

BSc (Hons) Computing

May 2019

More useful SQL error feedback

By

Adam Williams

Faculty of Science & Technology

Department of Computing and Informatics

Final Year Project

ABSTRACT

SQL (Structured Query Language) databases are usually used with Database Management Software (DBMS) like MySQL Workbench, for parsing queries to determine if they are valid for execution.

As downtime can be costly for businesses (ITIC 2019) queries being valid and knowing the cause when they are not is a significant concern for businesses, in order to prevent database downtime and loss of data. DBMS will parse queries and provide feedback, but this may not be useful in identifying the issue due to being too vague or lacking applicability to the specific issue.

This project details the research of existing parsing methods along with the requirements, design, development, testing etc for a prototype MySQL query parser that demonstrates providing more useful feedback for invalid queries than an existing parser.

The prototype uses the web technologies PHP, JS and HTML with a MySQL database for query execution and operates by the user entering a query which the parser then checks for syntax and semantic issues with feedback for the issues it identifies.

Comparison of feedback from the prototype and another parser on invalid queries by users indicates that the prototype successfully provides more useful feedback.

DISSERTATION DECLARATION

I agree that should the University wish to retain it for reference purposes, a copy of my dissertation may be held by Bournemouth University normally for a period of 3 academic years. I understand that once the retention period has expired my dissertation will be destroyed.

Confidentiality

I confirm that this dissertation does not contain information of a commercial or confidential nature or include personal information other than that which would normally be in the public domain unless the relevant permissions have been obtained. In particular, any information which identifies a particular individual's religious or political beliefs, information relating to their health, ethnicity, criminal history or sex life has been anonymised unless permission has been granted for its publication from the person to whom it relates.

Copyright

The copyright for this dissertation remains with me.

Requests for Information

I agree that this dissertation may be made available as the result of a request for information under the Freedom of Information Act.

Signed: _____.

Name: Adam Williams

Date:

Programme: Computing

Original Work Declaration

This dissertation and the project that it is based on are my own work, except where stated, in accordance with University regulations.

Signed: _____.

Name: Adam Williams

Date:

ACKNOWLEDGEMENTS

I would like to thank Dr Deniz Cetinkaya for providing support and supervision through the project's prototype and report development.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 BACKGROUND

SQL is a database specific scripting language that is used for the management and processing of data and data storage systems like relational databases, with MySQL being a significant portion of the SQL market share (DB-Engines 2019).

Part of SQL usage is making queries for those databases, a query is an SQL script for execution on the database to achieve the desired result like editing or retrieving data, these are usually done in relational database management software (RDBMS) clients like Microsoft SQL Management Studio (Microsoft 2017b), which will assist the user in making queries with feedback and validation of inputted queries.

However, if a query is syntactically or semantically invalid then it may fail to execute or have unintended effects, resulting in lost time and or undesired changes to data.

1.2 PROBLEM DEFINITION

The problem that this project aims to address is that most SQL parsers provide generic feedback for syntactically invalid queries.

I encountered this problem during my university placement year as the interns who would replace me had a break between their university term and the placement, resulting in them forgetting and having to relearn SQL. As they made syntax errors I noticed that there would be basic issues that the SQL management software would not clearly indicate.

For example the query “select * from” is lacking a table name or subquery after the “from”, but the similar software MySQL Workbench (Oracle Undated a) highlights the “select” with the feedback ‘*”select” is not valid at this position for this server version, expecting: ‘(‘, WITH*’ and on query execution it reports the error “*1064. You have an error in your SQL syntax; check the manual for the right syntax to use near '' at line 1*”. Both point to the “select” as the problem and do not indicate that the problem is due to the “from” clause lacking a data source after it.

This kind of lacking feedback can make learning and troubleshooting SQL unnecessarily difficult, as my replacements discovered.

1.3 PROPOSED SOLUTION

My proposed solution is a SQL parser designed to provide more accurate feedback on errors. For the project, a prototype was developed to demonstrate that the principles and methods are able to provide more useful feedback than existing SQL parser/s.

The prototype is a web page using JavaScript, HTML and PHP for parsing queries with a MySQL database for execution. The front end allows the user to enter a query which is checked for syntax issues based on what the clauses (select, from, update etc) require to execute without error.

If the query is identified to be invalid then the parser will provide feedback indicating why. This feedback should be more useful than what MySQL workbench (existing MySQL RDBMS client) provides for basic queries like “select * from” and “select colum1 colum2 from example”. This feedback should make it easier to identify what the issue’s cause is or clearly indicate the cause and suggest how to fix it.

While it would be specific to MySQL its principles and functions could be applied to other SQL languages.

1.4 Aims and Objectives

The general aim is to research and demonstrate parsing invalid MySQL queries and provide more detailed feedback on why they are invalid. The following objectives have been determined to be critical for this project using the Moscow method (IIBA 2015d).

1. Research current SQL query parsing methods and techniques in order to identify how they operate.
2. Develop a prototype web application that takes a syntactically incorrect query and provides more useful feedback than MySQL Workbench as to why it is incorrect and how to fix it for basic syntax errors.
3. Evaluate the prototype to determine if it proves the proposal is feasible and meets requirements.
4. Investigate and make theoretical methods for providing additional feedback on MySQL errors from server error messages.

1.5 Success Criteria

1. Identify 2 methods for parsing SQL and what operations, techniques, checks etc could provide useful feedback for that method.
2. Develop a prototype web MySQL parser that can detail the issues with basic queries like those shown in Appendix C 3.
3. Have around 10 or more people compare feedback from the prototype and MySQL Workbench to determine if feedback from the prototype is more useful.
4. Devise methods of detailing the cause of a MySQL server error from its error message in pseudo code for at least the following error types with general applicability to other errors:
   Error 1054. Unknown column 'column1' in 'field list'
   Error 1111. Invalid use of group function
   Error 1242. Subquery returns more than 1 row
   Error 1637. Too many active concurrent transactions
   Errors 1022, 1060, 1061, 1062, 1291, 1330, 1331, 1332, 1333, 1517. Duplicate object
   Error 1241. Operand should contain 1 column(s)

1.6 Risk Analysis

Risk analysis is the identification, evaluation and prioritisation of risks for a project in order to determine the appropriate precautions to take, based on their significance and likelihood (PivotPointSecurity 2016). For this project it has been used to list potential risks with precautions and rank their significance, based on probability and impact in accordance with the risk matrix in Appendix G.

Risk	Probability	Impact	Significance	Effect	Precaution
Running out of time	Medium	Very High	High 15	Project failure or reduced mark.	A work schedule of 09:00 to 17:00 on weekdays and if that is not enough then it will increase to 09:00 to 20:00.

No users to test prototype	Very low	High	Low 4	Only use cases and other tests	I know a number of people with basic coding knowledge and can offer a reward to other students in exchange for feedback.
Hardware failure and or data loss	Low	High	Medium 8	Loss of time, progress on the project, report etc.	Project files are automatically backed up to Google Drive from my laptop, Google Drive also automatically syncs with my desktop and files are manually backed up to USB.
Unable to meet project requirements	Medium	High	Medium 12	Project failure or reduced mark.	If it seems that I will not meet project requirements within the project timeframe then they will be scaled back and or changed to be more feasible. If they cannot be scaled back then I would do what I can in the available time.
Unable to work on the project (sickness, injury etc)	Very low	High	Low 4	Unable to work on the project, can result in failure or reduced mark.	Unlikely as I have not been sick in years and avoid alcohol, cigarettes etc. Injuries are unlikely due to few chances for them to occur.
Unable to host prototype on server	Very low	Medium	Low 3	Unable to have others test the prototype remotely	I am able to port forward and have multiple computers for hosting. If those fail then I can host the prototype on a university server.

Figure 1 Risk analysis table

1.7 Report Overview

The chapters in this report cover the following:

1. An introduction covering the problem, proposed solution, objectives, risks etc.
2. Background research of SQL parsing methods and issues.
3. Methodology research, chosen methodology for the project and plans.
4. Analysis of the problem, functional and non-functional requirements.
5. Design principles and documentation for the prototype.
6. Implementation and operational details for the prototype.
7. Testing of the prototype and results.
8. Conclusions and evaluation of the project.

2 BACKGROUND STUDY

2.1 OVERVIEW

This chapter summarises some existing SQL parsing methods, general SQL query structure and some issues with parsing SQL.

2.2 SQL PARSING OVERVIEW

SQL parsing is the process of a SQL parser software analysing a text string in accordance with the rules of the SQL language, to determine if the query is syntactically and or semantically valid for execution (Oracle Undated E).

If the query fails syntax parsing or fails during execution then the parser will return an error with an indication of its cause, however, this indication may be lacking details specific to the query like which clause is the cause of the error and how to fix it, see Appendix D 2 for examples.

SQL parsers are commonly used in SQL server administration tools like MySQL Workbench and standalone parsers like SQLFiddle (Feasel 2018). In most cases a parser's operations include:

- Validating a query in order to assist in its correct execution
- Correcting developers on proper SQL syntax
- Assisting in the writing of queries through features like autocomplete (Figure 2)
- Preventing accidents like changing all of a table's data through the safe mode safety feature (Appendix E).

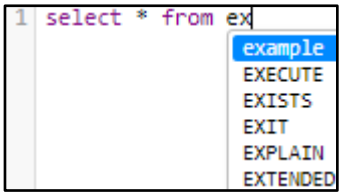


Figure 2 phpMyAdmin MySQL autocomplete

2.3 SQL QUERY STRUCTURE

SQL queries are usually made up of (Oracle Undated F):

- Clauses: reserved keywords like Select and Update
- Operators: reserved functions for specifying conditions like =, >, IN etc.
- Separators: characters to denote separation between objects like commas.
- Objects: table and column names
- Parenthesis: characters for separating a query into subqueries and for denoting datatypes like arrays.
- Values: data, text etc

See Figure 3 below for examples.

select	*column1*	*,*	*column2*	*from*	*table1*	*where*	*column1*	*=*	*1*
Clause	Object	Separator	Object	Clause	Object	Clause	Object	Operator	Value

update	*table1*	*set*	*column1*	*=*	*2*	*where*	*column2*	*=*	*1*
Clause	Object	Clause	Object	operator	value	clause	object	operator	value

insert	*into*	*table1*	*(*	*column1*	*)*	*values*	*(*	*2*	*)*
clause	clause	object	bracket	object	bracket	clause	bracket	value	bracket

Figure 3 Query structure examples

2.4 Syntax And Semantics

When parsing a query there will usually be syntax checks before execution and semantic checks before or during execution. The difference between the two for SQL is that syntax are the rules of the language relating how the query can be constructed and considered executable while semantics are the rules for how the query operates during runtime and if its operations are possible (Cherian 2019).

For example "*select * from table*" uses "table" as a table name but it is a reserved clause, resulting in a syntax error, while "*select * from example group by avg(blah)*" is syntactically valid but the aggregate function "avg" cannot be used with the "group by" clause (Oracle Undated f), resulting in server error 1111.

2.5 Parsing Issues

From Figure 3 it is clear that the valid text following a clause will depend on its syntax and semantics, due to this any SQL parser must check how a clause can be used, what should follow it, what operators can be used with it, its acceptable data types, etc.

For example for the query *"select column1, column2 from (select num from example) where num = '1'",* the parser's checks include:

- "Select" is followed by * or words that are not reserved clauses (column names) with commas between them if more than 1.
- "From" is followed by a non-reserved word (table name) or open and close brackets containing a valid subquery.
- "From" is preceded by a select, delete or another of its required clauses.
- "Where" is followed by a subject (num), operator (=) and target ("1") and that any segments after those follow a similar structure with conditions (AND, OR) between segments.
- The number of single quotes should be even unless escaped (\') and that text between unescaped quotes should not be checked for clauses due to being data.

- Text within parentheses should not affect the clauses outside it beyond being a data source.
- The number of parentheses should be even unless escaped.
- “Select” is followed by a “from” clause.
- “Where” is preceded by a “from” or another required clause.
- No reserved clauses (Oracle Undated b) are incorrectly used, like as a data source or column name.

Additionally, due to the potential complexity of MySQL queries, clauses, conditions, requirements etc, each with their own syntax and rules (Oracle Undated c), query parsing would include numerous checks and processes in order to catch invalid queries and identify the causes.

A demonstration of complex SQL queries can be seen in Appendix C 1 and 2.

2.6 Existing SQL Parsing Methods

2.6.1 phpMyAdmin MySQL Parser

This breaks down the phpMyAdmin MySQL Parser source code available from its GitHub page (phpMyAdmin 2019).

The query is passed to the constructor of Parser.php which determines if a lexer is to be used.

If a lexer is to be used then Lexer.php breaks the query into tokens (on spaces, operators, special characters etc) and checks each token (word/clause) of the query for if there should be delimiters, special characters, reserved keywords, spaces etc.

After the lexer each token is checked for:

- Brackets to determine if the statement is in a subquery by counting the open and close brackets.
- If the token is: a union clause (Union, Union All, Union Distinct, etc), a known statement/clause (Select, Update etc), part of a union requiring additional checks, part of a transaction (Start, Commit and Rollback), etc.
- Acceptable clauses for the statement type (Joins, Where and From for Select)
- If the previous token is relevant to the current token with checks depending on that
- If the query clauses are in the correct order.

And other checks with feedback messages for if the query fails a given check.

A significant part of its checks are the acceptable clauses that each token is checked against, as it determines the acceptable structure and grammar of a query.

To achieve this, each recognised clause (select, from, update etc) has an associated Statement.php file (SelectStatement.php, DeleteStatement.php etc) which contains its grammar and any statement specific functions. For example, Figure 4 shows the operational modifiers and clauses for the Select statement.

```
public static $OPTIONS = array(
    'ALL' => 1,
    'DISTINCT' => 1,
    'DISTINCTROW' => 1,
    'HIGH_PRIORITY' => 2,
    'MAX_STATEMENT_TIME' => array(3, 'var='),
    'STRAIGHT_JOIN' => 4,
    'SQL_SMALL_RESULT' => 5,
    'SQL_BIG_RESULT' => 6,
    'SQL_BUFFER_RESULT' => 7,
    'SQL_CACHE' => 8,
    'SQL_NO_CACHE' => 8,
    'SQL_CALC_FOUND_ROWS' => 9,
);

public static $END_OPTIONS = array(
    'FOR UPDATE' => 1,
    'LOCK IN SHARE MODE' => 1,
);

public static $CLAUSES = array(
    'SELECT' => array('SELECT', 2),
    '_OPTIONS' => array('_OPTIONS', 1),
    '_SELECT' => array('SELECT', 1),
    'INTO' => array('INTO', 3),
    'FROM' => array('FROM', 3),
    'FORCE' => array('FORCE', 1),
    'USE' => array('USE', 1),
    'IGNORE' => array('IGNORE', 3),
    'PARTITION' => array('PARTITION', 3),
    'JOIN' => array('JOIN', 1),
```

Figure 4 SelectStatement.php

This approach to MySQL parsing seems to be the most common as it is used in other papers involving SQL parsing (Anderson and Hills 2017) and MySQL Workbench from a general read of its source code (Oracle Undated a).

2.6.2 Abstract and Concrete Syntax Trees

Abstract and Concrete Syntax Trees (AST and CST) are hierarchical representations of code structure with each node (branch) on the tree being a word, variable, section etc of the code.

For example the SQL statement *"Select blah, column1 from example where column1 = '2'"* can be represented as a syntax tree in Figure 5.

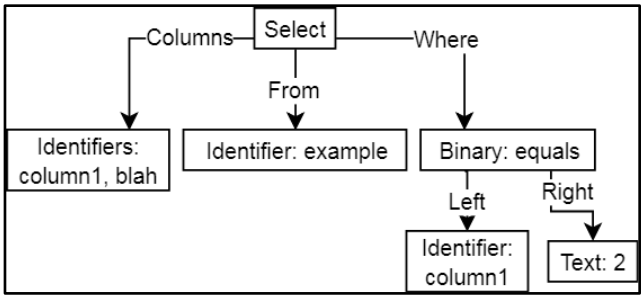


Figure 5 SQL Syntax Tree

The main difference between an AST and a CST is that a CST is the exact syntax of the code while an AST can be more abstract, usually to be more human readable.

Using a syntax tree allows for queries to be broken into nodes with semantic and syntax analysis being done on each node, in accordance with the SQL dialect and the above node in the hierarchy.

I.e. for the “Columns” node in Figure 5 the above node is “Select” and one of the dialect’s rules is that column names are separated by commas.

Another example of this approach to identify invalid query nodes is to parse each node based on what is expected in accordance with the above clause and if a node is invalid then mark it, the marked node can then be checked to determine what it should be. This could be against a template for the query or parent clause (Nagy et all. 2015).

However this method and similar methods require a template and or examples of the correct usage for each clause, a way to do this is by scanning a system for SQL queries to make ASTs from, but an existing system may not be available, with the alternatives being manually making valid templates or mining them from another source like StackOverflow, but this may be self-defeating through requiring a working SQL parser to determine if the mined queries are valid (Nagy and Cleve 2015).

Additionally, syntax trees can be used for general code recognition and parsing, Antlr (Parr undated) being an example of a general purpose code parser that can be configured for different languages through different lexer and parsing logic files.

2.7 Error feedback

As current error feedback is lacking in usefulness, it needs to be determined what information feedback should contain and how it should be presented, for example:

1. Additional feedback on an error should be useful to users with the feedback being existing feedback in plain terms instead of programming terms, pointing out the invalid clause, why it is invalid and how to fix it (Prather et al. 2017) etc.
2. If too much detail is provided then a user may only skim read feedback or look for key terms, to avoid this feedback should: be minimal, explicitly point to the issue, use simple vocabulary and reference sections of the query to indicate where the problem is (Marceau et al. 2011).
3. Where possible the feedback should only point to query clauses that are the cause of or related to the error and always refer to terms and clauses in the query instead of expecting the user to know what it is referring to (Traver 2010).
4. Feedback for different errors and related visuals should have a similar format and structure to make recognising key elements easier.
5. Using generic terms like “illegal character” and “extraneous input” should be avoided as they can refer to multiple issues and their meaning may not be understood.
6. The tone of feedback should be kept neutral to avoid offence.

2.8 Research Summary

From my research, it appears that the area of SQL parsing theory is lacking in papers and clear documentation, as most of the parsing logic is in the source code of software like phpMyAdmin and MySQL Workbench, with abstract general code parsing in the form of syntax trees.

Due to this most of the usable research for the prototype is in the form of:

1. Reserved keyword list to determine what is and isn't a reserved word for MySQL (Oracle Undated b).
2. Having a lexer for checking the query against what a clause should have, 2.6.1.
3. Checking each token/word of the query at a time for if it is missing any required clauses or characters, 2.6.1.
4. Breaking the query into sections based on the clauses then checking text between those clauses, 2.6.2.
5. Principles on what to provide in error feedback and how to structure it, 2.7.

However, due to the complexities and variations involved with MySQL queries, my prototype will only cover the basic queries in scope (Select, Delete, From etc), with detailed checking for some clauses.

3 METHODOLOGY

3.1 OVERVIEW

This chapter details the methodologies I have researched as for this project, my chosen methodology for my project and my project plan.

3.2 METHODOLOGY RESEARCH

As with most software development projects in the past year (StackOverflow 2018), an Agile or similar continuous improvement methodology will be used to account for changes in requirements, organised software testing and risk management. Factors relating teamwork have not been taken into account as this is a solo project.

However, as requirements may not change and as the timeframe for development is limited (Appendix F) some waterfall methodologies have been researched.

For this section, I have looked at multiple methodologies and researched in more depth three that I think are the most suitable, those being Rapid Application Development (RAD), Extreme Programming (XP) and Kanban.

3.2.1 Rapid Application Development

RAD is a waterfall methodology that usually consists of 4 stages: requirements, rapid prototyping, feedback and finalisation. These stages tend to result in broad requirements with multiple prototypes being developed quickly for user feedback, in order to determine what to include in the finished product (Coleman and Verbruggen 1998).

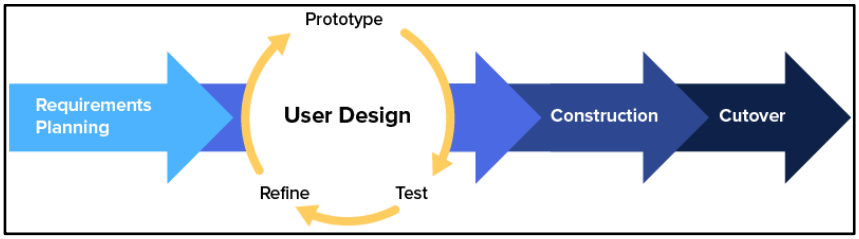


Figure 6 RAD process diagram (KISSFLOW 2018)

RAD has some suitable point for my project as the requirement of “more useful feedback than MySQL Workbench“ is broad in theory but in practice, the development timeframe and scope of the project impose restrictions that make multiple prototypes impractical, for example:

- Feedback will be in text format as other formats like images have little to no benefit for the time to develop them, audio feedback could be used for the visually impaired but would not be included for a proof of concept prototype and as text to speech software makes this redundant.
- Queries will be inputted using a text box, alternative inputs like file upload could be used but require significantly more development time due to the sanitisation and validation involved with processing a file compared to a text string.
- There will be one theme for the visuals (text and colour), colour scheme and font can be quickly changed in code but as the prototype will use HTML any user stylings/plugins can alter the colour scheme (Shutov 2019) and font (FontFace 2018), making multiple themes redundant.

Additionally while making multiple usable prototypes quickly can result in the final product being made quicker it usually requires an experienced developer and or result in under-designed prototypes due to the fast planning and development.

3.2.2 Extreme Programming

XP is an agile methodology that aims for user satisfaction within a feasible time frame, instead of meeting all requirements at a later date (Wells 2013). For software development, it emphasises communication, simplicity, feedback, respect, and courage through various principles including:

- Involving the user in the development process by frequently having them test the software and provide feedback for incremental changes
- Delivering a usable prototype early in development for user feedback and refinements.
- Frequent communication between developers with face to face conversations and meetings
- Planning ahead using user stories, release schedules for software and separating development into segments.
- Restructuring code whenever feasible
- Unit testing all code
- Developing code pairs of people each at a single computer with only one pair integrating their code at a time on a dedicated integration computer.
- Testing whenever a bug is found with new tests made if necessary

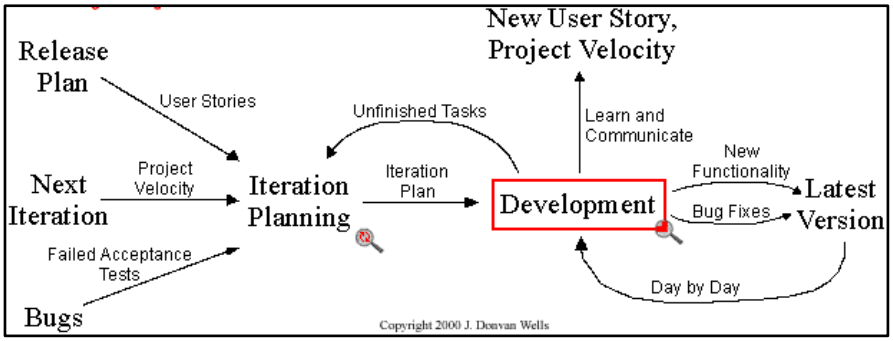


Figure 7 XP Development loop

For my project I do not think XP is suitable as development does not involve frequent user interaction. This is due to the prototype being a proof of concept to demonstrate more useful error feedback and as a result, is lacking the numerous checks to be functional for a user, for example, one of the query checks will be determining if there are unbalanced quotations which have a small number of relevant inputs (‘, “ and escaped quotes) and outputs, significantly reducing the effectiveness of frequent user testing.

3.2.3 Kanban (Gross and Mcinnis 2003)

Kanban is a scheduling process that aims to improve time efficiency for tasks and identify where work can be optimised through using visual elements, it achieves this through:

1. Creating a visual model for work, so that team members can see the work done, being done, to do etc. For example, the to-do board shown in Figure 8.
2. Separating projects into small isolated tasks for quick completion, mitigation of fatigue, reduce bottlenecks and keep developers focused on one task, at a time as multitasking tend to decrease efficiency (American Psychological Association 2006), For example, user story 1, user story 2, comment system etc.

3. Improved flexibility as a team member that has finished their task they can switch to another without breaking their workflow, this is useful if requirements are likely to change with little notice.
4. Gathering information about in-process tasks to determine which stall and why. For example, person A stalled with task C, if person A has had issues in the past with similar tasks then they may not be suited to that type of task.
5. Incrementally refining the Kanban process to improve the efficiency of work using information from 4.

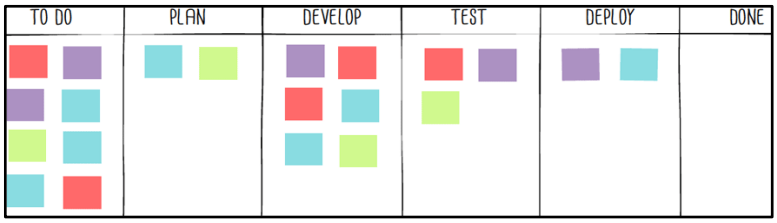


Figure 8 Kanban board (easyproject 2015)

However, for products with significant bottleneck tasks and those that cannot be easily split, Kanban can result in an unbalanced distribution of work and or developers waiting for others to finish their task before continuing. Additionally, if requirements are changing frequently or irregularly then task switching may become necessary and the benefits of focusing on one task at a time lost.

3.3 Chosen Methodology

My chosen methodology for this project is Kanban as the prototype development is relatively small, can be easily broken down into parts with any change in requirements likely to be scaling back instead of increasing and as it is a solo project the issues of bottlenecks and tasks not being suitable to the person are irrelevant.

I normally work on one task at a time so changing my regular workflow to be more aligned with Kanban should not be an issue and to help me focus on each task. Additionally developing the prototype in sections should make testing easier from the sections having a degree of isolation with only data being passed between them.

3.4 Project Plan

As shown in the Appendix F 1 a Gantt chart was made detailing how much time will be spent on a week by week basis for each task, with each weekday usually consisting of around 6 to 7 working hours (09:00 to 16:30 with an hour break), some days may have less due to various factors and I may work late on some other days to make up lost time.

Partway through development it became clear that given the number of complexities with SQL parsing and error codes the prototype and report would not be finished in time, so the development requirements were scaled back with the prototype being more basic in its syntax checks and one function for a specific MySQL server error, with ways to handle others being demonstrated in pseudo code, overall this made the project more theoretical than practical and a revised Gantt chart was made to reflect this (Appendix F 2).

The timeframes for tasks in the version 1 Gantt chart were rough estimations for how long I thought a task would take, for version 2 they are based on my progress before the 6th of March.

4 REQUIREMENTS AND ANALYSIS

4.1 OVERVIEW

This chapter details the prototype requirements gathered from the research in chapter 2 that I aim to achieve and for those I am unable to detail why they are not achieved.

4.2 PROBLEM DOMAIN

As detailed in chapters 1 and 2, the problem is that existing SQL parsing provides vague feedback on syntax and semantic errors that can be little help in determining the cause, depending on the queries complexity, error and user's SQL proficiency.

Chapter 1.1 shows the feedback for a select query lacking a target (table or subquery) after the "from" clause, which points to the select clause being the issue and not the lack of a target. Additional examples of vague error feedback for pre-runtime (client side) and runtime (server side) errors can be seen in Appendix D 2.

The cause of this problem appears to be how the parsers have been made to provide error feedback. The source code for phpMyAdmin's MySQL parser (Chapter 2.6.1) and the number of MySQL server error codes (Oracle Undated d), gives the indication that the developers opted to go for many simple checks and feedback instead of more detailed checks and feedback, most likely as more detailed checks and feedback would take significantly more time to process queries and to develop due to variations in query structure (Figure 3).

4.3 REQUIREMENTS GATHERING

Requirements were determined by looking at the functionality and code of existing MySQL parsers, feedback from them to invalid queries and determining what they should ideally respond with. The parsers looked at are primarily MySQL Workbench and phpMyAdmin's parser, which have some differences in their syntax checks but use the MySQL server for runtime feedback.

4.3.1 Parser testing

In order to determine a baseline for what the parser should achieve and where to focus, I tested MySQL Workbench and phpMyAdmin's parser with multiple invalid queries to see the feedback they provide. The results of this are shown in Appendix D 2.

This testing and observation of results gave me the initial requirements of:

- Identify if something is missing in a query and if so suggest what should be there, like table names, clauses, special characters, operators etc. Q1, Q7, Q8, Q9, Q10, Q11, Q13, Q15.
- Distinguishing between reserved and unreserved keywords. Q2.
- Checking if the unescaped parentheses and quotes are odd/unbalanced. Q4, Q5 and Q6.
- Notifying if both quote types are used in the same query, not a strict requirement as MySQL can use both but using one to close the other will result in an error. Q6.
- Identify the clause an error is caused by and provide its character number for identification.
- Identifying if a clause is valid at its position relative to other clauses. Q16.

4.3.2 Lexer

As detailed in section 2.6 a lexer is used for breaking the query into tokens and attaching information to them that the parser can use. Parsing the whole query as a single string does not seem to be viable as I could not find any theory or code on it so a lexer will be used.

What the lexer breaks on will be determined in the design chapter but spaces, parenthesis, quotes and special characters like commas are likely as they affect how a query is parsed, see Figure 9 for a lexer break example.

select * from example order by column1, blah								
select	*	from	example	order	by	column1	,	blah

Figure 9 lexer breakdown

4.3.3 Miscellaneous

These requirements are obvious for the project proposal, not directly connected to any research or were suggested by others and fit the aims of the prototype.

- Web site, server and associated software to host the prototype over the internet for user testing and demonstration.
- A text box for the query to be entered into with code to trigger the query checks on the text area's contents changing.
- Spell check each word with spelling suggestions if misspelt.
- Have the MySQL server parse queries without execution to prevent queries like table and column deletions.
- Section of the website that on a server error displays results from StackOverflow (or similar) for that error.
- Cover the syntax of the basic clauses like select, from, group, order etc.
- Cover the syntax of advanced clauses and functions like casting, pivot, triggers etc.
- Be able to change the prototypes connected database for database specific checks, like if a named table exists.
- The principles outlined in 2.7.
- An object browser to see objects in the database, similar to the MySQL Workbench browser shown in Figure 10.

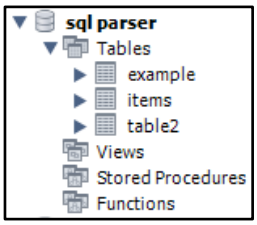


Figure 10 MySQL Workbench object browser

4.4 MOSCOW FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

After gathering the requirements they were rated using the Moscow method (IIBA 2015) and for prioritisation of work sorted into functional (what it does, like “take text query input”) and non-functional (how it does it, like “feedback includes the index number of the invalid clause”) requirements.

The Moscow method separates requirements into 4 categories based in significance to the end result with those being:

1. Must: requirements that are mandatory for the project to be completed.
2. Should: requirements of high importance but are not vital for completion.
3. Could: requirements that could be included if higher priority requirements are met first.
4. Would: requirements that the project could have but are not expected to be included, may be included in future versions.

4.4.1 Functional Requirements

Functional requirements for the prototype.

ID	Requirement	Rating
FR1	A textbox for a query to be entered that when changed triggers code to perform checks on its contents	Must
FR2	Provide suggestions on how to resolve a given syntax error	Must
FR3	Spell check the textbox’s words with spelling suggestions for misspelt words	Must
FR3	Be available over the internet for marking and testing	Must
FR4	Distinguish between text strings and MySQL clauses	Must
FR5	Check if a clause is valid at its position relative to other clauses in the query	Must
FR6	Check if quotes and parentheses are balanced/even	Must
FR7	Passing query to the server for executing without affecting the database	Should

Figure 11 Functional requirements

4.4.2 Non-functional Requirements

Non-functional requirements for the prototype.

ID	Requirement	Rating
NR1	Show the character number of the word an error is caused by	Should
NR2	Providing useful feedback on MySQL server errors from query execution	Should
NR3	A section on the page containing links to StackOverflow pages (or similar site) for a server error that occurs.	Could
NR4	Cover the syntax of selects, inserts, updates, deletes, joins, grouping, ordering, General syntax and subqueries	Should
NR5	Connect system to an external database for database specific queries	Would
NR6	Object browser for seeing entities in the connected database	Would
NR7	Accurate query parsing, i.e. catch missing quotes that do not denote text.	Could
NR8	Cover functions like casting, converting, datepart, views etc	Would
NR9	Feedback on clauses and database objects, i.e. if a select statement then it could indicate the column and table names	Would
NR10	Complete query parsing coverage like MySQL workbench	Would
NR11	Cover advanced functions like stored procedures, triggers, pivot etc.	Would

Figure 12 Non-functional requirements

5 DESIGN

5.1 OVERVIEW

This chapter covers the design of the front and back ends of the prototype along with how some of the requirements are implemented.

5.2 PROTOTYPE FRONT END DESIGN

5.2.1 Wireframes

Wireframes were made as drafts for the prototype web page front end for recording significant visual design changes and visualise the page (Lynch et al. 2016).

5.2.1.1 Main page V1

Version 1 of the front end design is shown in Figure 13.

Due to the limited input and output between the user and prototype (queries in and feedback out) the design is minimalist as great detail may distract from query feedback.

The dynamic elements are the text areas as they would change on a query being entered and with query feedback.

The colour scheme would be limited to a small number of high contrast colours (black and white) for easy readability and compatibility with browser extensions that alter pages colours (Shutov 2019), i.e. black text on white background becomes white text on black background.

Element breakdown:

- Page title: name of the prototype, would be “SQL Parser” or similar.
- Intro text: a brief summary of how the prototype operates, would explain the process of submitting a query and receiving feedback.
- Text area for query: text box for the user to enter a query with code being triggered on its contents changing.
- Submit query to server button: a button that triggers code to send the text box’s contents to the server for parsing and execution.
- Area for feedback on query: when a query is entered or submitted error feedback will appear in this area in a list format.
- Area for StackOverflow results: If server execution of the query returns an error then this area will contain links to the top results from a StackOverflow search of that error.
- Parser Settings Button: Opens a menu containing configurable settings for the prototype.

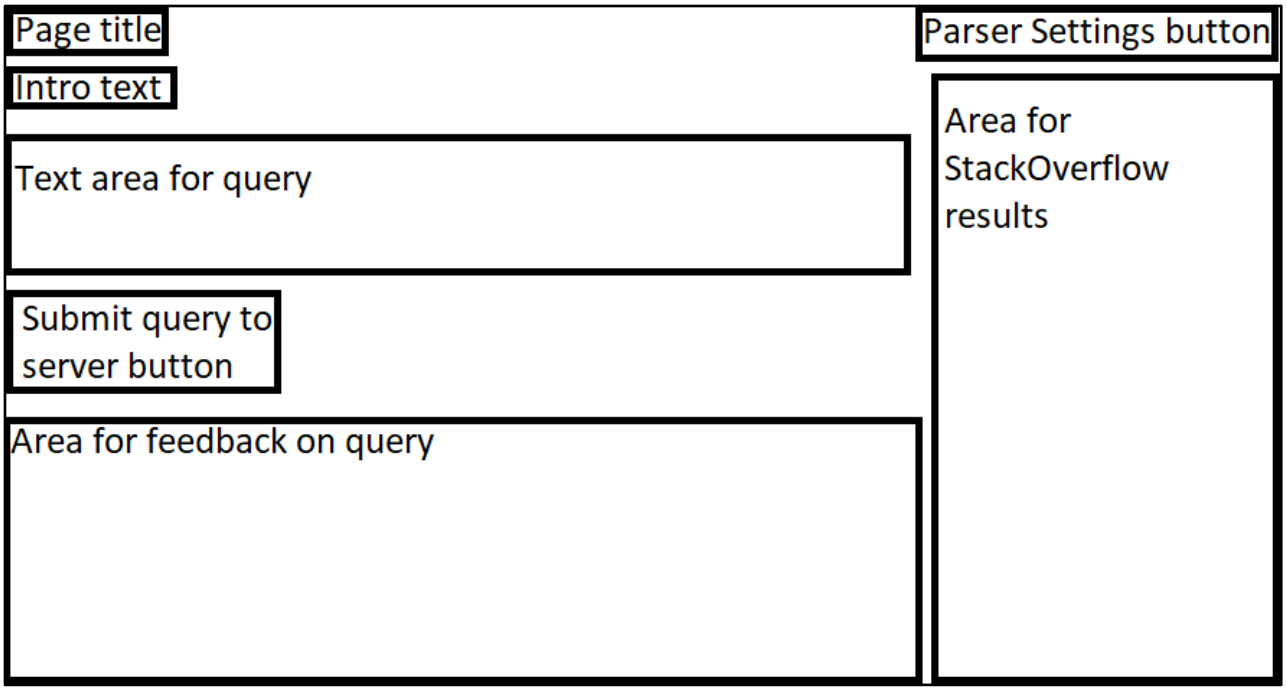


Figure 13 Parser wireframe V1

5.2.1.2 Settings menu

Version 1 of the settings menu is shown in Figure 14.

This menu would contain user configurable settings for the prototype like the server connection.

Element breakdown:

- Settings: the title of the menu
- Server connection settings: multiple text boxes for the user to enter the details of a MySQL server to connect to, for database specific queries and checks.
- Other settings: an area for settings that are not detailed.
- Apply and Cancel: buttons to apply or cancel setting changes.

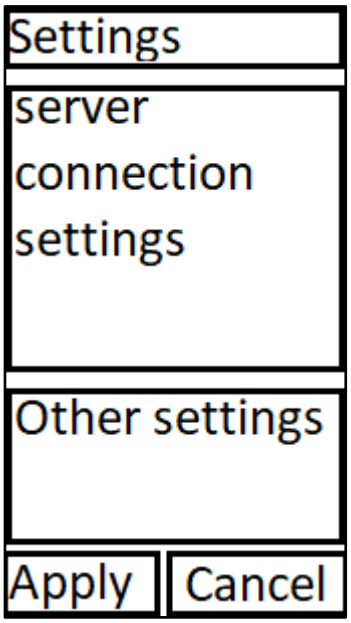


Figure 14 settings menu wireframe V1

5.2.1.3 Main page V2

Due to a scale back in requirements from the StackOverflow results, changing database connection and settings requirements were removed from the design.

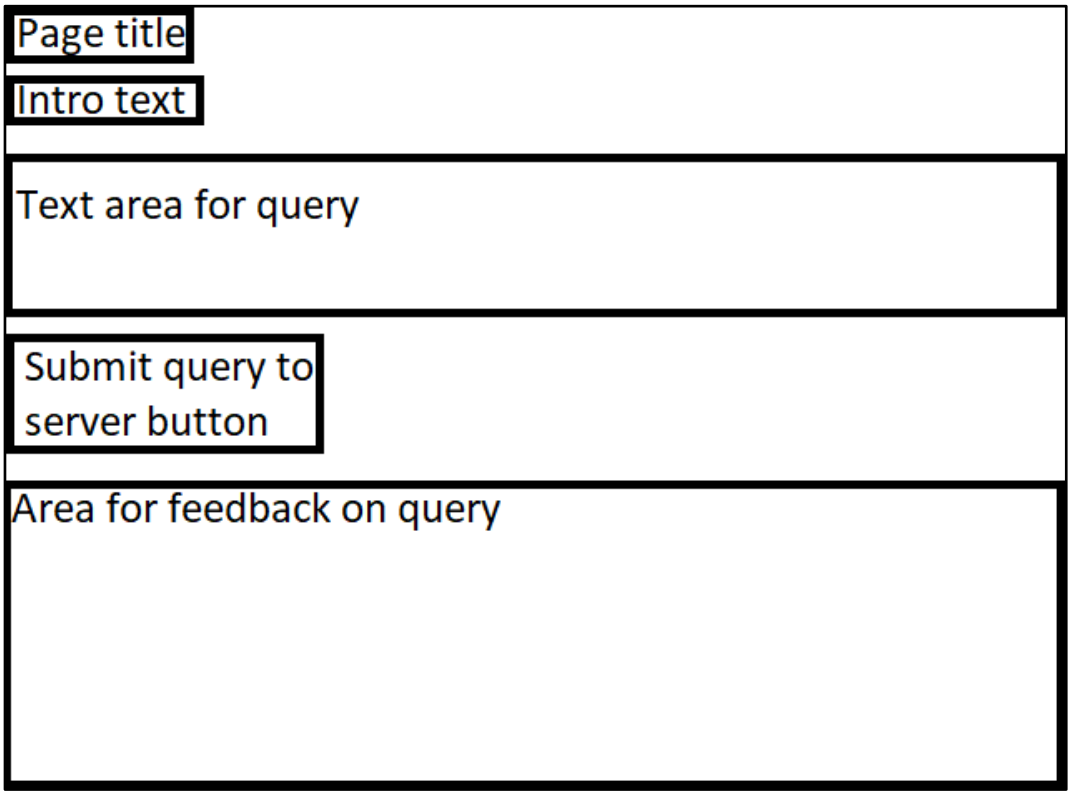


Figure 15 Parser wireframe V2

5.2.2 Lexer

The lexer breaks a string into tokens based on criteria and attaches additional information to each token.

For my prototype, the lexer would break the query into tokens on spaces, commas, colons, parenthesis, quotes and other special SQL characters which affect how a query is parsed (Oracle 2019). Each token would be added to an array with its index number for referencing the token in error checks and feedback, see Figure 16 below for an example.

select column1, blah from example where blah = 2										
Index	0	7	14	16	21	26	34	40	45	47
Token	select	Column1	,	blah	from	example	where	blah	=	2

Figure 16 lexer example

5.2.3 Syntax Parser

The lexer will break a query into an array of tokens with index numbers that the parser checks for basic syntax issues, like their order, required clauses etc.

Parsing the query for syntax issues will involve a number of steps:

1. Check each token against a reserved clause list (Oracle Undated b) for if it is a reserved clause (select, from, order etc) and that it is not between quotes as data does not need to be checked.

2. If it is a reserved clause, then check each next token against a list of expected clauses for if it matches an expected end clause (i.e. “select” is start and “from” is the end), if no end clause is found and the clause needs one then return an error detailing the missing clause.
3. If there is an end clause then check the tokens between the start and end clauses against a list for if they are valid there (between select and from there should be column names, subquery, * etc), if a token is not valid then return an error detailing the invalid tokens and the expected tokens.
4. If there is no end clause but there should be tokens after the clause (i.e. column names) then check each next token against the lexer for if it is valid until finds an invalid token (reserved clause) or the query ends, if it finds an invalid token then return an error with its details.

Due to the variations in query structure and error causes, this is only a general approach and depends on the clauses, reserved tokens, end clauses etc but this would work for following examples queries in Appendix D 2:

- Q1, a non-reserved clause after the “from”
- Q2, a non-reserved clause between “select” and “from” but there is the reserved clause “column” instead
- Q3, “from” should be followed by a table or subquery but the brackets are empty.
- Q7, “from” should be followed by a table there is the reserved clause “order” and “by” should be followed by column name/s but is empty.
- Q8, “by” requires that any non-reserved tokens after it are separated by commas.

For the queries involving quotes, there would be two additional procedures:

1. Check if the numbers of double and single quotes not following an escape (\) is even, if one is odd then return the character and number of the last quote as an error. This would work for Q4 and Q6 as it would point out the quotes that are unmatched.
2. If quote numbers are even, then check each quote for where its matching quote is to build an array of quote ranges for other procedures that need to determine if a token is a clause or is data in a text string.

For parenthesis, the same could be done with the first check being if the number of unescaped parenthesis of each type is even, but the second would have to take into account the position of the open and close parenthesis relative to each other, as other checks will need to determine if a token is within the same subquery.

This could be done by checking each token for if it is an open parenthesis then setting a variable (parCount) to 0, if the next parenthesis is another open then parCount + 1, if it is a matching close parenthesis then parCount - 1 and if it’s a matching close along with parCount being 0 then the close has been found with the open and close parenthesis being added to an array of ranges.

For example in the query *“Select ("test1", ("test2")) from example”* there are two bracket ranges, 7 to 26 and 17 to 25. The string would be split on the spaces, brackets and other characters, then each word would be checked for if it is a bracket, on the (at character 7 it would then look for a matching close bracket by checking each next word, once it gets to the (at 17 then it increases parCount by 1 then decreases it by 1 at) 25 so that when it reaches the) at 26 it correctly identifies it as the matching close bracket. If parCount was not used then it would match the next close bracket to each open bracket, resulting in two ranges with the same close bracket.

5.2.4 Feedback principles

Based on section 2.7 in order for feedback to be maximally useful it should:

1. Aim to use as little text as feasible to be concise.
2. Use plain language and terms to describe the issue in order to avoid miscommunication and technical terms where possible.
3. Explain relevant principles in the feedback instead of expecting the user to know.
4. Refer to clauses in the query when indicating an issue.
5. Indicate how to fix the issue.

For example the query *"Select column from example"* is invalid from the reserved clause "column" being used instead of a column name, In accordance with the principles feedback for this query would be *"The reserved word 'column' at character 7 should be replaced with a column name as a column cannot be called 'column'"*.

5.3 Prototype Back End Design

5.3.1 Server error feedback theory

Initially, I planned to make a basic server-side parser in PHP, for which if a server error occurred it would analyse the query in relation to the error to determine the cause and provide feedback.

It would have checks to achieve this depending on the error, for example:

- If the error and query have similar terms then perform specific checks. I.e. if query and error contain "group" then check the tokens after any "group by" clauses.
- If the error has a term like "incorrect number of" then check for clauses with multiple values and if the number of clauses matches the number of value, i.e. insert statement with 2 columns and 1 value.
- If the error contains a term like "duplicate" and the query includes "create" then check the database to see if an object with the same name already exists.

However, due to a scale back in requirements, this was reduced to the code for one MySQL error with other solutions being made in pseudo code to demonstrate the theory.

5.3.2 Pseudo code for errors

5.3.2.1 Error 1054. Unknown column in field list

```
if (connected to a MySQL server) {
    variable lastWordTableClause = false
    variable columnName = ""
    variable columnSuggestions = ""
    For each word of the MySQL query {
        if (columnSuggestions not = "") {
            if (lastWordTableClause = true AND columnName = "") {
                columnName = word
                columnSuggestions = Results of MySQL query ("select column_name, table_name from information_schema.columns where
                column_name like" fuzzy recognition of ("column1"))
            } else if (word = "from" OR word = "update" OR word = "into") {
                lastWordTableClause = true
            }
        }
    }
    returnFeedback ("There is no column with the name "column1" for this table, there are columns with a simular name: " + columnSuggestions)
} else {
    variable spellingSuggestions = spell check suggestions for "column1"
    returnFeedback ("There is no column with the name "column1" for this table, as it may be misspelt here are spelling suggestions: " +
    spellingSuggestions)
}
```

Figure 17 Error 1054 pseudo code

If a database is connected then this would check its information_schema (view for database objects) for columns with a similar name to the missing column and inform the user of them, If not connected then it would provide spelling suggestions as the column name may be misspelt.

5.3.2.2 Error 1242. Subquery returns more than 1 row

```
variable bracketOpen = 0
variable subQuery = ""
variable subQueryArray
For each word in query {
    if (bracketOpen = 1 AND word = "select") {
        subQuery = word
    } else if (word = "(") {
        bracketOpen = bracketOpen + 1
    } else if (word = ")") {
        bracketOpen = bracketOpen -1
    } else if (bracketOpen > 0 AND subQuery NOT = "") {
        subQuery = subQuery + " " + subQuery
    } else if (bracketOpen = 0 AND subQuery NOT = "") {
        subQueryArray append subQuery
        subQuery = ""
    }
}
For each query in subQueryArray {
    queryResults = execute query
    if (count queryResults > 1) {
        return feedback ("Subquery " + query + " returns more than 1 value, it may be the cause of the error")
    }
}
```

Figure 18 Error 1242 pseudo code

Find the subqueries (those between parentheses) then execute each of them and return a message for those that return more than 1 result.

5.3.2.3 Error 1637. Too many active concurrent transactions

```
variable processList = MySQL query "SHOW PROCESSLIST"
returnFeedback ("There are " + count(processList) + " current transactions, please re-execute the query
when there are less or manually terminate stuck transactions if you have the permissions to do so")
```

Figure 19 Error 1637 pseudo code

Return the number of concurrent transactions to the user.

5.3.2.4 Errors 1022, 1060, 1061, 1062, 1088, 1092, 1330, 1331, 1332, 1333, 1826 etc. Duplicate object

```
variable duplicateObject
variable queryResults = MySQL query (select type, objectName from information_schema.(TABLES, COLUMNS,
 VIEWS, TABLE_CONSTRAINTS, ROUTINES, triggers etc) where objectName like "% duplicateObject %")
variable objectType = queryResults[type]
returnFeedback(duplicateObject + " already exists in the database as a " + objectType)
```

Figure 20 Error duplicate pseudo code

Check the database for an object with the same name and return its details to the user, see Appendix C 1 for the full query.

5.3.2.5 Error 1241. Operand should contain 1 column

```
variable bracketOpen = 0
variable subQuery = ""
variable subQueryArray
For each word in query {
    if (bracketOpen = 1 AND word = "select") {
        subQuery = word
    } else if (word = "(") {
        bracketOpen = bracketOpen + 1
    } else if (word = ")") {
        bracketOpen = bracketOpen -1
    } else if (bracketOpen > 0 AND subQuery NOT = "") {
        subQuery = subQuery + " " + subQuery
    } else if (bracketOpen = 0 AND subQuery NOT = "") {
        subQueryArray append subQuery
        subQuery = ""
    }
}
For each query in subQueryArray {
    queryResults = execute query
    if (columnCount (queryResults > 1)) {
        return feedback ("Subquery " + query + " returns more than 1 columns, it may be the cause of the error")
    }
}
```

Figure 21 Error 1241 pseudo code

Find the operands between brackets that return multiple columns then notify the user.

5.3.2.6 Errors 1153, 1162, 1301. Object bigger than max_allowed_packet

These errors are caused by the current object being processed having a packet size larger than the database allows. Get the name and size of the error object along with the database limit and inform the user of them.

```
variable largeObject = MySQLErrorObjectName
variable largeObjectSize = MySQLErrorObjectSize
variable databaseMaxPacket = MySQL Query (SHOW VARIABLES LIKE 'max_allowed_packet';)
returnFeedback("The object " + largeObject + " with a size of " + largeObjectSize +
 " is over the database packet limit of " + databaseMaxPacket)
```

Figure 22 Error max_allowed_packet pseudo code

5.3.3 Single error feedback

Error 1111 “Invalid use of group function” was used as the cause is usually using aggregate or similar functions with the “group by” instead of a “having” clause and would demonstrates providing feedback on functions with invalid clauses.

```
aggregateAndReservedClausesArray
For each word of query {
    if (groupByFound = true AND havingFound = false) {
        if (aggregateAndReservedClausesArray contains word) {
        return feedback (word + " should not be with the group by, if it is an
        aggregate function then it should with a select or having clause")
        }
    } else if (word = "by") {
        variable groupByFound = true
    } else if (word = "having") {
        variable havingFound = true
    }
}
```

Figure 23 Error 1111 pseudo code

Check if the “group by” clause is followed by an aggregate or reserved clause and that no “having”, aggregate or reserved clause is after the “group by” but before the “having”.

6 IMPLEMENTATION

6.1 OVERVIEW

This chapter details the development, sections of code, operation and tools used for the prototype parser along with significant challenges encountered and how they were resolved.

6.2 PROTOTYPE DEVELOPMENT

6.2.1 Development Environment and Tools

Initially, I was not using an environment as I thought that the necessary functionality would exist in the default MySQL and PHP servers, however, this is incorrect as there were issues with connecting and configuring them.

To resolve this and simplify server configuration I used Xampp (Apache Friends 2015) for the prototype's base as it comes with Apache (web server), MySQL and Filezilla (FTP).

For an integrated development environment I used PHPStorm (JetBrains 2018) with Skeleton (Gamache 2014) as a frontend framework, as I have experience with them from previous projects.

6.2.2 Prototype front end operation

6.2.2.1 Visuals

As shown in Appendix J 1 the design of the website is minimalist as the prototype functionality is limited.

When the user changes the contents of the text area it is checked for SQL syntax issues and if any of those checks fail then a function is called to append feedback to the pages list element.

6.2.2.2 External scripts

1. clausesStartEnd.js: An array of start clauses, acceptable end clauses for the start clause, expected tokens between them, special conditions like if an end clause is optional and what separator the clause uses between values. I.e. the “select” clause has: [“select”, [“databaseObjectName”], [“from”], [“”], [“,”]].
2. requiredClauses.js: An array of clauses and the clauses that they require to be syntactically valid. I.e. the “where” clause has: ["where", [["from"], ["update"]]] as it requires a “from” or “update” in the same query.
3. reservedCharacters.js: An array of special characters reserved by MySQL like “~” and “/”.
4. reservedClauses.js: An array of tokens reserved by MySQL like “alter”, “order” and “table”.
5. JavaScriptSpellCheck (nanospell Undated): Spell checking words and with spelling suggestions.

6.2.2.3 Code operation order

1. Store query from the text area to a variable.
2. Call function balancedParenthesiseCheck on the query then store the quote and parenthesise indexes it returns.
3. Call function getQuoteRanges on the quote indexes to get their ranges.
4. Create a version of the query with text between quotes replaced with spaces as strings do not need to be checked.

5. Use the lexer (6.2.2.4) to break the new query into tokens with index numbers.
6. For each token spell check and call the functions oldClauseCheck and getClauseRanges to determine if it is valid in its position and if the query contains the required clauses.

6.2.2.4 Function lexer

```
let splits = textClauses.split(/ |(?=,)|(?=;)|(?=\))|(?=\()/g); //Split text into tokens
let index = 0; //Number/position of word
for(let i = 0; i < splits.length; i++) { //While i is less than text length
    if (splits[i] !== "") {
        words.push([index, splits[i]]);
    } //Add each word and its position to an array
    index += splits[i].length + 1; //Index/word number increased by the word length + 1
}
```

Figure 24 lexer code snippet

The lexer splits the query into tokens with index numbers and adds them to a 2D array, it uses regular expression (Goyvaerts 2019) to break the query on spaces, commas, semicolons and parentheses. By default, the .split function removes the character it splits (PHP Group Undated) but this is only needed for spaces so (?=) is used to keep characters needed for parsing the query like parentheses and commas.

6.2.2.5 Function balancedParenthesiseCheck

A modified version of a script (Hitt 2017) to check if parentheses are balanced in a query and record the positions of double and single quotes to arrays.

The modifications include accounting for escape characters, storing the positions of parenthesis, checking if double and single quotes are unbalanced and returning feedback to the user detailing the issue if any of its checks fail, for example, with the query “select * from (select column1 from table1” feedback like “The (at character 15 is unbalanced and may cause an error” would be returned.

It operates by breaking the query into characters then counting the parentheses and quotes, With parentheses it stores a count of how many open parentheses it has found with each close parentheses of the same type decreasing that count, if there are no more parenthesis and the count is not 0 then it returns feedback with the unbalanced parenthesis. It breaks the query into characters instead of using the lexer array because it needs individual characters not whole words.

6.2.2.6 Function getQuoteRanges

A function that takes the double and single quote index arrays from balancedParenthesiseCheck and creates an array of quote ranges along with providing feedback if quotes are unbalanced or both types are used.

It does this by counting the quotes of each type and on each even-numbered same type putting its and the previous quotes indexes into a 2D array, for example the query “select col1, ‘blah’ from example where col2 = ‘2’“ the quoteRanges array would be [[13, 18], [46, 48]].

```
if ((doubleQuoteCount % 2) === 0 && (doubleQuoteCount > 0)) {
    count = 0;
    previousValue = '';
    doubleQuoteIndex.forEach(function(value) {
        if (count % 2 !== 0) { //If odd then put quote position and previous quote positions into the array
            quoteRanges.push([previousValue, value]);
        }
        else {previousValue = value;} //If even quote then store its position
        count++
    });
} else if (doubleQuoteCount > 0) { //If remainder more than 0 then there are an odd number of double quotes
    appendFeedback('There is an unclosed ", if it is not in a text string and escaped (\\") then it may cause an error.');
}
if (doubleQuoteCount > 0 && singleQuoteCount > 0) { //If both quote types are used then display message.
    appendFeedback("Single and double quotes are used, if possible use one type for consistency.")
}
return quoteRanges;
```

Figure 25 getQuoteRanges snippet

6.2.2.7 Function oldClauseCheck

A function that checks each token against the requiredClauses.js array for if it is a MySQL clause, if the token matches a clause the query will be checked for one of its required clauses, for example, if the query contains “from” then it will be checked for “select” or “delete” and an error message returned if it does find either.

```
clause[1].forEach(function(subClause) { //For each required sub-clause of a clause, i.e. From is a required sub-clause of Select.
    if (word[1] === clause[0] && !textClauses.includes(subClause[0]) && missingClauseCheck === false) {
        missingClause += subClause[0] + ", ";
    } //If word matches clause, text does not include sub-clause and no required clause found then add missing clause to string.
    else if (word[1] === clause[0] && textClauses.includes(subClause[0])) {
        missingClauseCheck = true;
    } //If word equals clause and sub-clause is present then missingClauseCheck =  true.
});
if (missingClause !== '' && missingClauseCheck === false) {
    appendFeedback('"' + clause[0] + '" at column ' + word[0] + ' is missing one of the following clauses: "' + missingClause + '"');
} //If a required sub-clause is missing then append a message detailing that.
```

Figure 26 oldClauseCheck snippet

6.2.2.8 Function getClauseRanges

getClauseRanges is the primary function of the parser and a more complex but not as complete version of the oldClauseCheck function.

The main difference is that instead of just checking if the query contains a required clause it checks if a clause is followed by its required clause and that text between two clauses is valid. For example “from select” is valid under oldClauseCheck as it does not check the order.

It operates by being called with a token and an array of tokens after it in the query. Then a number of checks are applied until the end of the query or an expected clause (in the clausesStartEnd.js array) is found, provided that the tokens being checked are in the same query section as the start clause (not a subquery or between quotes), this process is done in two main sections.

Firstly if the next token checked is an expected end clause then the start and end clauses along with their index numbers are added to an array and the boolean clauseMatch is set to true.

```
if (endClauses.indexOf(compWord[1]) > -1) { //If word is an end clause for the start clause
    clauseRanges.push([word[0], compWord[0], word[1], compWord[1]]); //add start and end clause and their ranges to array
    clauseMatch = true; //Match has been found
} else if (reservedClauses.indexOf(compWord[1]) > - 1) { //If word is a reserved clause not in the end clause array
    appendFeedback('"' + word[1] + '" at character ' + word[0] + ' cannot have the clause "' + compWord[1] + '" at character ' + compWord[0]);
}
```

Figure 27 getClauseRanges snippet 1

Secondly, if the end clause has not been found then it adds the current word to the betweenClauseText array and performs checks on them to see if they have separators (if required) or if there are too many values for the start clause.

```
if (clauseMatch === false) { //If match not found in previous if statement
    betweenClauseText.push([compWord[0], compWord[1]]); //Put text into array
    if ((betweenClauseText.length % 2 === 0) && objectSeparators.length > 0 && objectSeparators.indexOf(compWord[1]) === -1) {
        //Even number of values and word is not a separator
        appendFeedback('Objects after clause "' + word[1] + '" at character ' + word[0] + ' need one of the following separators
        between its values:  ' + objectSeparators);
    } else if (specialConditions.indexOf("singleObject") && betweenClauseText.length > 1) {//Clause has single object flag and
    there is more than 1 non-clause value after if.
        appendFeedback('"' + word[1] + '" at character ' + word[0] + ' can only have 1 value between it and the next clause');
    }
}
```

Figure 28 getClauseRanges snippet 2

If any of those checks fail or the end clause is not found and the start clause should have one then feedback is provided to the user. Additionally, the function returns the start to end clause ranges, special conditions for the end clause and other information.

6.2.2.9 Remaining checks

Using the data returned from getClauseRanges a number of additional checks are performed depending on if start and end clauses have been found, if there is text between the clauses etc.

If there is a start clause, end clause, text between the clauses and both are in the same section (not in different sub queries) then for each token between the clauses check it against the list of expected tokens for the clause to see if it is acceptable, for example, if the start is “select” and the end is “from” then the expected tokens are database object names (not reserved clauses). If invalid clauses are found and there is no correct clause then notify the user with a list of expected tokens.

```
betweenClauseText.forEach(function(clauseText) { //For each word between the clauses
    let clauseMatch = false;
    let requiredClauses = [];
    if ('([{'.indexOf(clauseText[1]) > -1) {insideBracket++} //If open bracket then +1
    if (')]}'.indexOf(clauseText[1]) > -1) {insideBracket--} //If close bracket then -1
    if (insideBracket === 0) { //If not in a sub-query
        betweenClauses.forEach(function (betweenClause) { //Check word against each expected clause
            if (clauseMatch === false) { //Check word matches an expected clause if no match has been found
                if (clauseText[1].includes(betweenClause) || betweenClause.includes("databaseObjectName")) {
                    clauseMatch = true;
                }
                else {requiredClauses.push(clauseText[1]);} //If no match then put missing clause into array
            }
        });
    }
    if (clauseMatch === false && requiredClauses.length > 0) {
        appendFeedback('Clause "' + clauseText[1] + '" at character ' + clauseText[0] + ' does not appear to
        be valid, the expected values can include: ' + requiredClauses);
    }
});
```

Figure 29 remaining code snippet 1

If there is no text between the clauses and some is expected then return feedback detailing that.

```
} else if (betweenClauses.length > 0) {
    appendFeedback('There is no text between "' + clauseRanges[0][2] + '" at character ' + clauseRanges[0][0] + ' and "' + clauseRanges
    [0][3] + '" at character ' + clauseRanges[0][1] + ' but one or more of the following are required: ' + betweenClauses);
}
```

Figure 30 remaining code snippet 2

Then if there is a start clause that is missing an end clause or a set of start and end clauses are missing tokens/clauses between them then return feedback to the user. Some clauses will have markers to denote different formats and so they are handled differently, for example “endEmpty” denotes that a clause having an end clause is optional.

```
} else if (clauseRanges.length <= 0 && endClauses.length > 1 && specialConditions.indexOf('endEmpty') === -1) {
    appendFeedback('The "' + word[1] + '" clause at character ' + word[0] + ' is missing one of the following clauses
    after it 1: ' + endClauses); //If no clause range for word but has end clauses then required end clause/s are missing
} else if (betweenClauses.length > 0 && betweenClauseText.length === 0) {
    appendFeedback('The "' + word[1] + '" clause at character ' + word[0] + ' is missing one of the following: ' +
    betweenClauses + ". This may be a table name, column name, data type etc.");
}
```

Figure 31 remaining code snippet 3

6.2.3 Prototype back end operation

For the prototype, the server.php file handles queries sent from the main page using AJAX POST requests, as part of the scaled-back requirements it only handles server error 1111 and for others it just returns the MySQL server error message to the user.

The following are the most significant parts of its operation.

6.2.3.1 Database connection

Figure 32 shows the code for if a POST request is received and the default database connection details, as the initial requirements included having the prototype connect to external MySQL servers these details would have been received from the client instead of being hardcoded.

```
if (isset($_POST["query"])) {
    $username = "root"; $password = ""; $host = "127.0.0.1"; $db = "sql parser";
    $connection = mysqli_connect($host, $username, $password, $db);
```

Figure 32 Database connection code

6.2.3.2 Escaping and word array

Figure 33 shows the query and the token array from the client being escaped (treating special characters as text and not code). The query is escaped as a string for execution and the token array is remade with each value being escaped as I could not find a way to escape an array without breaking it up.

```
$userQuery = mysqli_real_escape_string($connection, $_POST["query"]); //Store query to variable with escaping
if (!mysqli_query($connection,$userQuery)) { //If query executes with error
    $wordsArrayUnescaped = json_decode($_POST['jsonWords']); //Get words with index values, decoded from JSON.
    $queryWords = []; //Variable for words
    foreach ($wordsArrayUnescaped as $word) { //escape each query value/word and put into new array
        array_push($queryWords, [mysqli_real_escape_string($connection, $word[0]), mysqli_real_escape_string(
        $connection, $word[1])]);
    }
```

Figure 33 Escape and array code

6.2.3.3 Error 1111

Figure 34 shows the code to handle error 1111 “Invalid use of group function”. It does this by checking each token of the array to see if it matches an aggregate function and is after a “group” but before a “having”. A common cause of this error is an aggregate function being used with a “group by” instead of a “having”. If it locates an aggregate with the “group by” then it returns feedback detailing the issue to the user.

```
if (mysqli_errno($connection) == 1111) { //Hard coded example for error 1111, invalid group by usage
    $groupIndex = ""; //Indexes of where "group" words are in query
    $aggregateIndex = ""; //index for aggregate words in query
    $havingFound = false;
    $groupFound = false;
    $aggregateFunctions = ["avg", "bit_and", "bit_or", "bit_xor", "count", "count", "group_concat", "json_arrayagg", "json_objectagg",
     "max", "min", "std", "stddev", "stddev_pop", "stddev_samp", "sum", "var_pop", "var_samp", "variance"]; //Aggregate functions
    foreach ($queryWords as $word) { //For each word
        if ($word[1] == "group") { //If "group" then add its index to variable and set groupFound to true.
            $groupIndex .= ($word[0] . " ");
            $groupFound = true;
        } elseif ($word[1] == "having") { //If "having" then set havingCheck to true as having has been found.
            $havingCheck = true;
        } elseif ($havingFound === false AND $groupFound === true AND in_array($word[1], $aggregateFunctions)) { //If an aggregate
        has been found, after "group" and before "having" then add it to variable, aggregates should be after the "having" clause by.
            $aggregateIndex .= ($word[1] . " at " . $word[0] . ", ");
        }
    }
    $returnString = "Error 1111 one of the group by clauses are not used correctly, character/s: " . $groupIndex . ". <br /> Group by
    should be followed by column names with , separating them.";
    if ($aggregateIndex !== "") { //If aggregate functions after group and before having then add text with their indexes.
        $returnString .= "<br />The following aggregate functions appear to be with the group clause but should be used with a Having
        clause: " . $aggregateIndex;
    }
```

Figure 34 Error 1111 code

7 PROTOTYPE TESTING

7.1 OVERVIEW

This section details the tests performed on the prototype in order to determine if it meets requirements and how it compares to the existing MySQL parsers.

7.2 ERROR FEEDBACK COMPARISON

In order to determine if the prototype demonstrates more useful feedback than MySQL workbench, a number of invalid queries were inputted into both parsers and the results recorded.

The results shown in Appendix H demonstrate that additional information for errors and possible ways to resolve them are feasible, for example as shown in Figure 35, C7 demonstrates that it can detect when a clause/name is missing with suggestions of what should be there, C5 demonstrates finding unbalanced parenthesis and C10 demonstrates the handling of a semantic server error from its error number.

ID	Query	Problem	Workbench	Prototype
C5	*select * from example where column1 = 1)*	Unclosed bracket after “1”	“Extraneous input found - expected end of input” for “)”.	The) at column 39 is unbalanced and may cause an error.
C7	*select * from order by*	No data source after “from” and no column/s after “by”	“Unexpected ‘order’ (order)” on “order”. “unexpected end of input” on “by”	There is no text between "from" at character 9 and "order" at character 14 but one or more of the following are required: databaseObjectName The "by" clause at character 20 is missing one of the following: databaseObjectName. This may be a table name, column name, data type etc.
C10	*select * from example group by avg(blah)*	Cannot use the aggregate function “avg” with “group by”	Error Code: 1111. Invalid use of group function	The clause "by" at character 28 cannot have the reserved clause "avg" at character 31 Server message: Error 1111 one of the group by clauses are not used correctly, character/s: 22. Group by should be followed by column names with , separating them. The following aggregate functions appear to be with the group clause but should be used with a Having clause: avg at 31,

Figure 35 Feedback Comparison Highlights

However, the procedures behind this feedback only provide additional information for the situations they have been made to address (unbalanced quotes, missing clauses etc) and are unfinished or lacking applicability for other situations like C11 to C15, as this feedback does not address the issues.

7.3 REQUIREMENT TEST CASES

In order to determine if the prototype meets the Moscow functional and non-functional requirements use case tests were made against them with a few of the more significant cases shown in Figure 36 and all results shown in Appendix I, requirement NR10 is omitted from testing due to not having a clear way to implement it.

Moscow	ID	Req ID	Test	Expected outcome	Actual outcome
MUST	RT 5	FR4	Enter the query “select by from example” into the text area	Feedback is provided indicating “by” is a MySQL reserved clause and not valid next to the “select”	Pass, feedback is provided indicating “by” is reserved and not valid with “select”: *The clause "select" at character 0 cannot have the reserved clause "by" at character 7* *The clause "by" at character 7 cannot have the reserved clause "from" at character 10*
	RT 7	FR6	Enter the query “select (blah" from example” into the text area	Feedback indicating that there are unbalanced (and “	Pass, feedback indicates the (and “ are unbalanced: *The (at column 7 is unbalanced and may cause an error.* *There is an unclosed ", if it is not in a text string and escaped (\") then it may cause an error.*
SHOULD	RT 8	FR7	Submit the query “delete from example” in the prototype then “select * from example” in MySQL Workbench	The delete query being valid in the prototype and the 2nd query returning data.	Pass, prototype finds no issue with query and example table has data.

Figure 36 Requirement test highlights

8 CONCLUSIONS

8.1 SUMMARY

The project has produced a prototype MySQL parser demonstrating methods for providing additional feedback on invalid SQL queries, along with research detailed general SQL structure, some of the issues with SQL parsing along with existing methods and theory for parsing SQL queries.

The prototype uses JavaScript and PHP procedures to analyse a MySQL query inputted through a HTML text area for syntax and semantic issues, it then provides feedback on the cause/s of and possible solution/s for identified issues.

However the identifiable issues are limited as the prototype has not been fully developed, for example, the prototype is connected to its own MySQL database with no option for the user to change it, so queries requiring specific database objects (table, columns etc) will return object not found errors, this means the prototype will not provide useful feedback on anything more complex than basic queries like selects with optional clauses.

If more time were to be spent developing the prototype then it could be a viable MySQL parser that provides more feedback on errors than existing parsers, however, I expect that the methods in use require more development time than those used in existing parsers due to greater complexity.

8.2 EVALUATION

In order to determine if the prototype is a success, its feedback has been evaluated by users with at least some coding experience and the prototype's features evaluated against the project requirements.

8.2.1 User evaluation

To determine if query feedback from the prototype is more useful than feedback from a MySQL IDE, users were asked for 10 invalid MySQL queries to choose which feedback is more useful for identifying the errors cause, see Appendix J2 for the web-page and Appendix H C1 to C10 for the queries with prototype and MySQL Workbench feedback.

Full results from user submissions are in Appendix K and the summary in Figure 37 indicates that users did find the feedback from the prototype (B) to be more useful than feedback from MySQL Workbench (A) as prototype feedback was chosen 75.5% of the time.

ID	Query	Times A was picked	Times B was picked
Q1	select * from	1	8
Q2	Select column from example	2	7
Q3	select * from ()	1	8
Q4	select "blah from example	4	5
Q5	select * from example where column1 = 1)	5	4
Q6	select "blah' from example	3	6
Q7	select * from order by	1	8

Q8	Select * from example order by column1 blah	1	8
Q9	select * update from example	2	7
Q10	select * from example group by avg(blah)	2	7
Total Picks		22	68
Percentage		24.5%	75.5%

Figure 37 User feedback summary

For the 3 queries where A was picked more than 25% of the time (Q4, Q5 and Q6) unbalanced quotes or parentheses are used, this is likely due to B not always specifying the quote and parenthesis locations depending on the error while A does, user comments support this as they indicate that A gives location details while B may not, For example:

- Q4, “Tells you where the problem is”
- Q5, “provides the user with detailed explanation in a more user understandable way why the problem has occurred and where”
- Q6, “A details where the quotes are”

For the other 7 queries where B was picked more than 75% of the time user comments indicate that this is due to B feedback providing more details, indicating more clearly what the cause is and how to fix it, For example:

- Q1, “More verbose and human readable “
- Q2, “More information given”
- Q3, “provides the user with detailed explanation why the problem has occurred and where”
- Q7, “shows where issue is”
- Q8, “tells you what to do to fix it”
- Q9, “helps show update is not valid”
- Q10, “More verbose, easier to read, more information”

8.2.2 Requirement evaluation

Appendix I details the tests to determine if requirements set out in section 4.4 have been met, with Figure 38 showing the number completed along with their importance values. Importance values are used to emphasise the significance of higher priority Moscow requirements over lower priority requirements and signify that despite only achieving half of the requirements, most of the more significant requirements were completed.

Importance values Would = 0, Could = 1, Should = 2, Must = 3. Partially completed requirements are counted as 0.5.						
Type	**Must**	**Should**	**Could**	**Would**	**Overall**	**Percentage**
Completed	6/6	3/4	0/2	0/6	9/18	50%
Importance score	18/18	6/8	0/2	0/0	24/28	85.71

Figure 38 Requirements test summary

For the Should requirements two partially passed due to insufficient development time, those being NR2 due to the server side checks being mostly theoretical and NR4 due to clauses like update and insert not being fully covered.

8.3 Future Work

The artefact produced is a proof of concept prototype as the timeframe available for the project is insufficient to develop a complete MySQL parser, this insufficiency can be seen with queries involving functions, updates, inserts, subqueries and other clauses beyond the basics.

Additionally, a server-side parser would need to be developed to handle query execution errors from their description and error code, this would be made to handle the most common server errors with general applicability to the less common errors by looking at key terms in the error message, see section 5.3 for details.

Other improvements and refinements that could be made to the prototype include but are not limited to:

1. Additional feedback on server errors from the error code and description, could use terms in the error description to look for issues in the query.
2. A section on the web page that updates with the top results from StackOverflow for a server error that occurs.
3. Fully cover the syntax of selects, inserts, updates, deletes, joins, group, order, subqueries, unions and all other significant MySQL clauses.
4. Fully cover the syntax of MySQL functions like pivots, aggregates, casting, datepart etc.
5. Be able to change the MySQL database that the prototype connects to using a settings menu on the web page.
6. An object browser on the web page for seeing entities in the connected database.
7. Determine the appropriate clauses for a query based on the existing clauses, i.e. for the query *"select * from update example"* it would see that it is a select query based on the "select" and "from" clauses and determine that "update" should not be there.
8. Autocomplete/clause suggestion when typing a query or for incomplete queries. See Figure 2 for an example.
9. Distinguish between clauses, object names and special characters for clause parsing.
10. More accurate syntax parsing for clauses with multiple values, i.e. currently the query "select column1, column2 from example" has the error "Clause "select" at character 0 can only have 1 value between it and the next clause" as accounting for multiple values has not been implemented.
11. Parsing of data sources involving parentheses that are not brackets like arrays.
12. Coverage of AS clause and aliases.
13. Coverage of stored procedures and stored variables.
14. Determine if the text entered is a MySQL query, for example, the text "hello world" is not a query but the prototype will not find a syntax error with it.
15. Proper web hosting of the prototype, for this project it was accessible through an IP address instead of a web page address.

Word count (main body of the report excluding figures and tables): 9524

REFERENCES

1. Anderson, D. and Hills, M., 2017. Supporting Analysis of SQL Queries in PHP AiR. 2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM) [online]. Available from: https://ieeexplore.ieee.org/document/8090149 [Accessed 1 Mar 2019].
2. American Psychological Association, 2006. Multitasking: Switching costs [online]. https://www.apa.org. Available from: https://www.apa.org/research/action/multitask [Accessed 13 Mar 2019].
3. Apache Friends, 2015. Xampp-win32-5.6.8-0-VC11-installer.exe. Windows. Apache Friends. Available from: https://sourceforge.net/projects/xampp/files/XAMPP%20Windows/5.6.8/ [Accessed 1 Mar 2019].
4. Brass, S. and Goldberg, C., 2004. Detecting Logical Errors in SQL Queries. Grundlagen von Datenbanken [online], 28-32. Available from: https://www.researchgate.net/publication/221397890_Detecting_Logical_Errors_in_SQL_Queries [Accessed 1 Mar 2019].
5. Brass, S. and Goldberg, C., 2005. Semantic errors in SQL queries: A quite complete list. Journal of Systems and Software [online], 79 (5), 630-644. Available from: https://www.sciencedirect.com/science/article/pii/S016412120500124X [Accessed 1 Mar 2019].
6. Cao, D. and Bai, D. (2010). Design and implementation for SQL parser based on ANTLR. In: 2010 2nd International Conference on Computer Engineering and Technology. [online] Chengdu: IEEE, pp.276 - 279. Available at: https://ieeexplore.ieee.org/abstract/document/5485593 [Accessed 19 Mar. 2019].
7. Coleman, G. and Verbruggen, R., 1998. A Quality Software Process for Rapid Application Development. Software Quality Journal [online], 7 (2), 107 - 122. Available from: https://link.springer.com/article/10.1023/A:1008856624790 [Accessed 14 Mar 2019].
8. Daniel, 2016. Best way to understand big and complex SQL queries with many subqueries [online]. Stack Overflow. Available from: https://stackoverflow.com/questions/37338331/best-way-to-understand-big-and-complex-sql-queries-with-many-subqueries [Accessed 11 Mar 2019].
9. easyprojects, 2015. simple-physical-board-w-card-types-e87dbe30.png [online]. image. Available from: https://explore.easyprojects.net/wp-content/uploads/2015/06/simple-physical-board-w-card-types-e87dbe30.png [Accessed 13 Mar 2019].
10. Feasel, J., 2018. SQL Fiddle | A tool for easy online testing and sharing of database problems and their solutions. [online]. SQL Fiddle. Available from: http://sqlfiddle.com/ [Accessed 8 Mar 2019].
11. FontFace, 2018. Fontface Ninja [online]. Chrome.google.com. Available from: https://chrome.google.com/webstore/detail/fontface-ninja/eljapbgkmlngdpckoiiibecpemleclhh [Accessed 13 Mar 2019].
12. Gamache, D., 2014. Skeleton: Responsive CSS Boilerplate [online]. Getskeleton.com. Available from: http://getskeleton.com/ [Accessed 22 Mar 2019].

13. Gross, J. and Mcinnis, K., 2003. Kanban made simple: demystifying and applying Toyota's legendary manufacturing process [online]. 1st ed. ebook. New York: Amacom. Available from: https://ebookcentral.proquest.com/lib/bournemouth-ebooks/detail.action?docID=3001746 [Accessed 13 Mar 2019].

14. HiSlide, Undated. Risk assessment 5x5 matrix template [online]. Hislide.io. Available from: https://hislide.io/product/risk-assessment-5x5-matrix-template/ [Accessed 27 Feb 2019].

15. Hitt, R., 2017. String Balancing in Javascript (STACK) [online]. Robhitt.com. Available from: https://www.robhitt.com/blog/balance-parenthesis-js/ [Accessed 4 Feb 2019].

16. IIBA, 2015. A Guide to the Business Analysis Body of Knowledge [online]. 3rd ed. ebook. Oakville, Ontario: International Institute of Business Analysis. Available from: http://www.innovativeprojectguide.com/documents/BABOK_Guide_v3_Member.pdf [Accessed 26 Feb 2019].

17. JetBrains, 2018. PHPStorm. Windows. JetBrains. Available from: https://www.jetbrains.com/phpstorm/

18. KISSFLOW, 2018. Rapid-application-development.png [online]. image. Available from: https://kissflow.com/wp-content/uploads/2018/07/Rapid-application-development.png [Accessed 13 Mar 2019].

19. Microsoft, 2017. SQL Server Management Studio (SSMS) - SQL Server [online]. Docs.microsoft.com. Available from: https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-2017 [Accessed 26 Feb 2019].

20. Mitrovic, A., 1998. Learning SQL with a computerized tutor. In: SIGCSE '98 Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education [online]. New York: SIGCSE, 307-311. Available from: https://dl.acm.org/citation.cfm?id=274318 [Accessed 1 Mar 2019].

21. Nagy, C. and Cleve, A., 2015. Mining Stack Overflow for discovering error patterns in SQL queries. In: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME) [online]. Bremen, Germany: IEEE, 516 - 520. Available from: https://ieeexplore.ieee.org/abstract/document/7332505 [Accessed 5 Mar 2019].

22. Nagy, C., Meurice, L. and Cleve, A., 2015. Where was this SQL query executed? a static concept location approach. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER) [online]. Montreal, QC: IEEE, 580-584. Available from: https://ieeexplore.ieee.org/abstract/document/7081881 [Accessed 1 Mar 2019].

23. nanospell, Undated. JavaScript Spell Check [online]. Javascriptspellcheck.com. Available from: https://www.javascriptspellcheck.com/ [Accessed 29 Jan 2019].

24. Oracle, Undated a. MySQL :: Download MySQL Workbench [online]. Dev.mysql.com. Available from: https://dev.mysql.com/downloads/workbench/ [Accessed 26 Feb 2019].

25. Oracle, Undated b. MySQL :: MySQL 8.0 Reference Manual :: 9.3 Keywords and Reserved Words [online]. Dev.mysql.com. Available from: https://dev.mysql.com/doc/refman/8.0/en/keywords.html [Accessed 11 Mar 2019].

26. Oracle, Undated c. MySQL :: MySQL 8.0 Reference Manual :: 13 SQL Statement Syntax [online]. Dev.mysql.com. Available from: https://dev.mysql.com/doc/refman/8.0/en/sql-syntax.html [Accessed 11 Mar 2019].

27. Oracle, Undated d. MySQL :: MySQL 8.0 Reference Manual :: B.3 Server Error Message Reference. [online] Dev.mysql.com. Available at: https://dev.mysql.com/doc/refman/8.0/en/server-error-reference.html [Accessed 15 Mar. 2019].

28. Oracle, Undated e. SQL Processing [online]. Docs.oracle.com. Available from: https://docs.oracle.com/database/121/TGSQL/tgsql_sqlproc.htm [Accessed 12 Apr 2019].

29. Oracle, Undated f. Built-In Aggregate Functions [online]. Docs.oracle.com. Available from: https://docs.oracle.com/cd/E14571_01/apirefs.1111/e12048/funcbltag.htm [Accessed 1 May 2019].

30. Parr, T., Undated. ANTLR [online]. Antlr.org. Available from: https://www.antlr.org [Accessed 7 Mar 2019].

31. phpMyAdmin, 2019. phpmyadmin/sql-parser [online]. GitHub. Available from: https://github.com/phpmyadmin/sql-parser [Accessed 1 Mar 2019].

32. Sadiq, S., Orlowska, M., Sadiq, W. and Lin, J., 2004. SQLator: an online SQL learning workbench. In: ITiCSE '04 Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education [online]. Leeds: ITiCSE, 223-227. Available from: https://dl.acm.org/citation.cfm?id=1008055 [Accessed 1 Mar 2019].

33. Shutov, A., 2019. Dark Reader — dark theme for every website [online]. Dark Reader. Available from: https://darkreader.org/ [Accessed 13 Mar 2019].

34. StackOverflow, 2018. Stack Overflow Developer Survey 2018 [online]. Stack Overflow. Available from: https://insights.stackoverflow.com/survey/2018#development-practices [Accessed 14 Mar 2019].

35. Wells, D., 2000. iteration.gif [online]. image. Available from: http://www.extremeprogramming.org/map/iteration.html [Accessed 11 Mar 2019].

36. Wells, D., 2013. Extreme Programming: A Gentle Introduction. [online]. Extremeprogramming.org. Available from: http://www.extremeprogramming.org/ [Accessed 11 Mar 2019].

37. DB-Engines, 2019. Most popular database management systems globally 2019 | Statistic [online]. Statista. Available from: https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/ [Accessed 4 Apr 2019].

38. ITIC, 2019. Global hourly enterprise server downtime cost 2017-2018 | Statistic [online]. Statista. Available from: https://www.statista.com/statistics/753938/worldwide-enterprise-server-hourly-downtime-cost/ [Accessed 4 Apr 2019].

39. Cherian, C., 2019. Oracle® Database Programmer's Guide to the Oracle Precompilers [online]. 19th ed. ebook. Oracle. Available from: https://docs.oracle.com/en/database/oracle/oracle-database/19/zzpre/programmers-guide-oracle-precompilers.pdf [Accessed 12 Apr 2019].

40. Oracle, 2019. Special Characters in Oracle Text Queries [online]. Oracle Help Center. Available from: https://docs.oracle.com/en/database/oracle/oracle-database/19/ccref/special-characters-in-oracle-text-queries.html [Accessed 15 Apr 2019].

41. Prather, J., Pettit, R., Holcomb McMurry, K., Peters, A., Homer, J., Simone, N. and Cohen, M., 2017. On Novices' Interaction with Compiler Error Messages: A Human Factors Approach. In: Proceedings of the 2017 ACM Conference on International Computing Education Research [online]. New York: ACM, 74-82. Available from: https://dl.acm.org/citation.cfm?doid=3105726.3106169 [Accessed 15 Apr 2019].

42. Marceau, G., Fisler, K. and Krishnamurthi, S., 2011. Mind your language: on novices' interactions with error messages. In: Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software [online]. New York: ACM, 3-18. Available from: https://dl.acm.org/citation.cfm?id=2048241 [Accessed 15 Apr 2019].

43. Traver, V., 2010. On Compiler Error Messages: What They Say and What They Mean. Advances in Human-Computer Interaction [online], 2010, 1-26. Available from: https://www.hindawi.com/journals/ahci/2010/602570/.

44. Goyvaerts, J., 2019. Using Regular Expressions with PHP [online]. Regular-expressions.info. Available from: https://www.regular-expressions.info/php.html [Accessed 16 Apr 2019].

45. PHP Group, Undated. PHP: split - Manual [online]. Php.net. Available from: https://www.php.net/manual/en/function.split.php [Accessed 16 Apr 2019].

46. PivotPointSecurity, 2016. Why Use Matrix Models for Risk Assessment? I Pivot Point Security [online]. Pivot Point Security. Available from: https://www.pivotpointsecurity.com/blog/using-matrix-models-for-risk-assessment/ [Accessed 16 Apr 2019].

47. Oracle, Undated F. MySQL :: MySQL 8.0 Reference Manual :: 9 Language Structure [online]. Dev.mysql.com. Available from: https://dev.mysql.com/doc/refman/8.0/en/language-structure.html [Accessed 16 Apr 2019].

48. Lynch, P., Horton, S. and Marcotte, E., 2016. Web Style Guide: Foundations of User Experience Design, 4th Edition. 4th ed. New Haven: Yale University Press. Available from: https://books.google.co.uk/books?hl=en&lr=&id=EY_BDAAAQBAJ [Accessed 16 Apr 2019].

APPENDIX A - PROJECT PROPOSAL

UNDERGRADUATE PROJECT PROPOSAL FORM

Please refer to the **Project Handbook Section 4** when completing this form

Degree Title: Computing	**Student's Name: Adam Williams**
	Supervisor's Name: Deniz Cetinkaya
	Project Title/Area: SQL Parser

SECTION 1: PROJECT OVERVIEW

1.1 Problem definition - use <u>one sentence</u> to summarise the problem:
Existing SQL IDEs and teaching systems do not provide specific details on errors like what exactly the cause is, in most cases, there will be a generic error message.

1.2 Project description - briefly explain your project:
A web-based Oracle SQL IDE/query parser that aims to provide specific feedback on errors, existing IDEs and code checkers normally provide generic/vague error messages on errors whereas I will try to provide accurate feedback on the errors cause and failing that more information on the issue than the base Oracle SQL Developer IDE.

1.3 Background - please provide brief background information, e.g., client:
The idea came about due to teaching SQL to my replacements during my placement.
The real-world application of this would be for finding the cause of an error for those using SQL when the generic error message does not provide enough information to determine the cause.
Target demographic being SQL developers, mostly those learning SQL as advanced developers are more likely to know or be able to find the cause.

1.4 Aims and objectives – what are the aims and objectives of your project?
It would be a website to check for errors and provide feedback on SQL queries with specific details as to what causes the error and how to fix it.
The text boxes would have validation to inform the user if they make a mistake with the query, for example, would tell the user if there is no FROM clause or an unclosed bracket.
I would aim for it to cover at the basics of Oracle which includes: selects, inserts, updates, deletes, joins, grouping, ordering, General syntax, views, subqueries and functions like casting, converting, datepart etc.
If I have enough time then it would cover advanced functions like stored procedures, triggers, pivot, blob data etc.

SECTION 2: ARTEFACT

2.1 What is the artefact that you intend to produce?
A web-based Oracle SQL IDE/query parser that aims to provide specific feedback on errors, existing IDEs and code checkers normally provide generic/vague error messages on errors whereas I will try to provide accurate feedback on the errors cause.

It would consist of:
A text box/area for entering the SQL query to check and run.
A section for feedback to appear as the query is entered and executed.

A client-side part using JavaScript for basic/quick checks, syntax and grammar errors that are not specific to the database, i.e. an odd number of (or selct instead of select.
A server-side part using vb.net for more advanced logic and using error messages from the Oracle SQL server to determine the error cause, i.e. if "table not found" then it would check the database to see if a table with a similar name exists and suggest that table.
A settings menu for changing the Oracle database it is connected to.
Object browser for the connected database in order to show the tables, views, procedures etc.
A section that shows and links to the top/first results for an SQL error from stack overflow (or similar site), does not show unless an SQL server error occurs.

2.2 How is your artefact actionable (i.e., routes to exploitation in the technology domain)?
The artefact is a route to better error recognition/detailing for SQL statements and speeding up learning and refining a developer's SQL.
This is actionable as it would provide information as to what is wrong with a query allowing the developer and learners to take note and not make the mistake again.

SECTION 3: EVALUATION

3.1 How are you going to evaluate your work?
I plan to assess how well the artefact parses SQL by having a few people with little SQL knowledge using it to see if they can successfully create and execute queries, this would focus around the basics and extend to the advanced functions if there is time.
I have a couple of friends who could test this and when the system is in a usable state would try to have first years test it for a reward, I aim to have around 10 people test it and provide feedback in the form of a 1 to 5 rating and any comments they may have.

Additionally, I will compare how the system performs relative to SQL IDEs and online validators like the Oracle SQL Developer and W3schools SQL system.
This would be done by inputting multiple broken queries into the systems and recording the output, ideally, my system should provide more information about the error cause than the other systems.

3.2 Why is this project honours worthy?
I think coding a system to determine the cause of an error in the SQL statement based on the error message will not be an easy task and will require a lot of work.
Additionally, the project:

- Is original, but not entirely unique as it would incorporate the existing Oracle SQL Developer IDE.
- Will require me to research more and refine my programming techniques/methods for the system to not be a mess.
- Will demonstrate programming ability, programming logic and thinking about writing SQL from a beginner's perspective

3.3 How does this project relate to your degree title outcomes?
It is applicable to the Advanced Development and Web Information Systems units as it is a web software development project.
It meets the requirements of investigating a computing problem by building a system with the aim to resolve it and will demonstrate:

- Knowledge/understanding of multiple computing/programming languages, techniques and principles.
- Working as an individual to Identify and solve a computing problem with the evaluation of resources and design methods.
- Planning, monitoring and evaluation of a computing project.

3.4 How does your project meet the BCS Undergraduate Project Requirements?
The project will solve a real-life need (lack of specific feedback for SQL errors) through the application of analytical and practical skills learned from the course and placement.
Additionally, doing the project will demonstrate an ability to plan and manage work along with producing a project report containing: elucidation, investigation, verification, appraisal, research, and analysis etc of a computing problem and development of a solution.

3.5 What are the risks in this project and how are you going to manage them?
Running out of time:
Due to my work attitude of 09:00 to 17:00 do it until it is done I do not think this is likely to happen, however, if it looks like I may run out of time then I will work on it almost non-stop until it is done.
Multiple end users cannot test:
If I cannot get anyone with little to no SQL knowledge to test it then I will do the end user testing myself by deliberately making as many mistakes as I can to find bugs/flaws.
Hardware failure and data loss:
If the computer or storage I am using for the project fails I have spare storage (USB, drives) and computers and will be regularly backing up university work to USB and google drive.

SECTION 4: REFERENCES

4.1 Please provide references if you have used any.

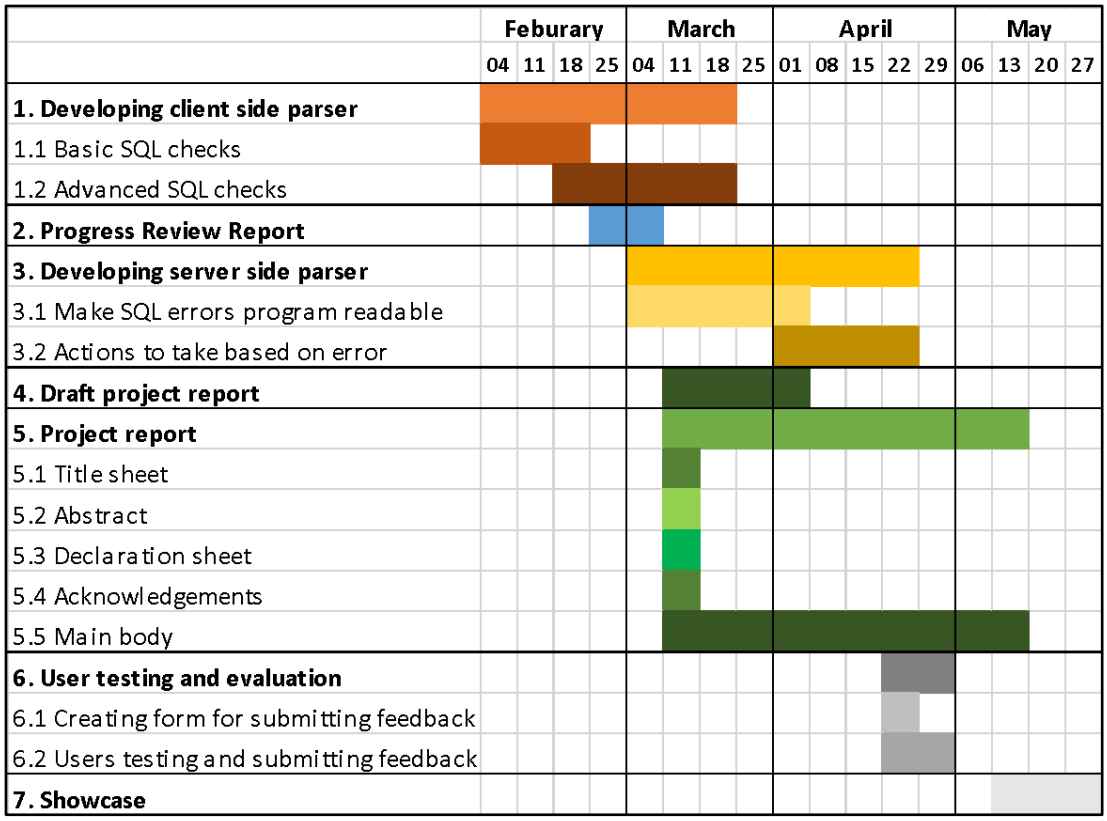
SECTION 5: ETHICS (PLEASE DELETE AS APPROPRIATE)

5.1 Have you submitted the ethics checklist to your supervisor? Yes

5.2 Has the checklist been approved by your supervisor? Yes

SECTION 6: PROPOSED PLAN (PLEASE ATTACH YOUR GANTT CHART BELOW)

I do not expect to stick to a plan as my general work ethic is work 09:00 to 17:00 until it's done with work on the weekends, however, this usually means the amount of work done in a day is not consistent as it can be affected by other factors like being tired, distractions, assignments, exams etc.
If it looks like I am falling behind then I will have to work on the project more after 17:00 and on the weekends.


Feburary
March
April
May
04 11 18 25 04 11 18 25 01 08 15 22 29 06 13 20 27
1. Developing client side parser
1.1 Basic SQL checks
1.2 Advanced SQL checks
2. Progress Review Report
3. Developing server side parser
3.1 Make SQL errors program readable
3.2 Actions to take based on error
4. Draft project report
5. Project report
5.1 Title sheet
5.2 Abstract
5.3 Declaration sheet
5.4 Acknowledgements
5.5 Main body
6. User testing and evaluation
6.1 Creating form for submitting feedback
6.2 Users testing and submitting feedback
7. Showcase

APPENDIX B - BU RESEARCH ETHICS CHECKLIST

1. Student Details

Name	Adam Williams
School	Faculty of Science & Technology
Course	BSc Computing
Have you received external funding to support this research project?	No
Please list any persons or institutions that you will be conducting joint research with, both internal to BU as well as external collaborators.	

2. Project Details

Title	SQL Parser
Proposed Start Date	28-January-2019
Proposed End Date	31-August-2019
Supervisor	Deniz Cetinkaya

Summary (including detail on background methodology, sample, outcomes, etc.)

A web-based Oracle SQL IDE/query parser that aims to provide specific feedback on errors, existing IDEs and code checkers normally provide generic/vague error messages on errors whereas I will try to provide accurate feedback on the errors cause.

This kind of system partly exists in the form of websites like W3Schools and the query validators in SQL workbench programs, but their error messages are vague to the point where the interns would be asking me why the query failed to execute when the error is something simple that the error message does not cover.

3. External Ethics Review (Answer “Yes” go to 4, “No” go to 5)

Does your research require external review through the NHS National Research Ethics Service (NRES) or through another external Ethics Committee?	No

4. External Ethics Review Continued

Answered “Yes” to question 3 will conclude the BU Ethics Review so you do not need to answer the following questions. Note you will need to obtain external ethical approval before commencing your research.

5. Research Literature (Answer “Yes” go to 6, “No” go to 7)

Is your research solely literature based?	No

6. Research Literature Continued (Either answer will conclude the review)

Will you have access to personal data that allows you to identify individuals OR access to confidential corporate or company data (that is not covered by confidentiality terms within an agreement or by a separate confidentiality agreement)?	No
Describe how you will collect, manage and store the personal data (taking into consideration the Data Protection Act 2018, General Data Protection Regulation (GDPR) and the Data Protection Principles).	
NA	

7. Human Participants Part 1 (Answer "Yes" go to 8, "No" go to 12)

Will your research project involves interaction with human participants as primary sources of data (e.g. interview, observation, original survey)?	Yes

8. Human Participants Part 2 (Answer **any** "Yes" go to 9)

Does your research specifically involve participants who are considered vulnerable (i.e. children, those with cognitive impairment, those in unequal relationships—such as your own students, prison inmates, etc.)?	No
Does the study involve participants age 16 or over who are unable to give informed consent (i.e. people with learning disabilities)? NOTE: All research that falls under the auspices of the Mental Capacity Act 2005 must be reviewed by NHS NRES.	No
Will the study require the co-operation of a gatekeeper for initial access to the groups or individuals to be recruited? (I.e. students at school, members of a self-help group, residents of Nursing home?)	No
Will it be necessary for participants to take part in your study without their knowledge and consent at the time (i.e. covert observation of people in non-public places)?	No
Will the study involve discussion of sensitive topics (i.e. sexual activity, drug use, criminal activity)?	No

9. Human Participants Part 2 Continued

Describe how you will deal with the ethical issues with human participants?
None of the issues raised in section 8 are applicable.

10. Human Participants Part 3 (Answer **any** "Yes" go to 11, **all** "No" go to 12)

Could your research induce psychological stress or anxiety, cause harm or have negative consequences for the participant or researcher (beyond the risks encountered in normal life)?	No
Will your research involve prolonged or repetitive testing?	No
Will the research involve the collection of audio materials?	No
Will your research involve the collection of photographic or video materials?	No
Will financial or other inducements (other than reasonable expenses and compensation for time) be offered to participants?	No

11. Human Participants Part 3 Continued

Please explain below why your research project involves the above-mentioned criteria (be sure to explain why the sensitive criterion is essential to your project's success). Give a summary of the ethical issues and any action that will be taken to address these. Explain how you will obtain informed consent (and from whom) and how you will inform the participant(s) about the research project (i.e. participant information sheet). A sample consent form and participant information sheet can be found on the Research Ethics website.
Evaluation of the SQL Parser will involve a couple of friends and computing first years using the website and then provide a rating (1 to 5) and feedback on the system, i.e. how it compares to a typical SQL IDE in their opinion. I will ask a couple of my friends (preferably those with little to no SQL experience/knowledge) to test the system and offer monetary compensation for their time. For the first years, I will send out an email and or put up posters/ads around the computing areas of the university detailing the testing and offering monetary compensation for the participant's time. I aim to have around 10 people test the system and will use the feedback/ratings in my project report.

12. Final Review

Will you have access to personal data that allows you to identify individuals OR access to confidential corporate or company data (that is not covered by confidentiality terms within an agreement or by a separate confidentiality agreement)?	No
Will your research take place outside the UK (including any and all stages of research: collection, storage, analysis, etc.)?	No
Please use the below text box to highlight any other ethical concerns or risks that may arise during your research that have not been covered in this form.	

Review Completion Date: 21-September-2018

APPENDIX C - SQL QUERIES

1. MySQL Database Object Search

```
SELECT OBJECT_TYPE,OBJECT_NAME FROM (
    SELECT 'TABLE' AS OBJECT_TYPE,TABLE_NAME AS OBJECT_NAME FROM information_schema.TABLES
    UNION
    SELECT 'COLUMN' AS OBJECT_TYPE,COLUMN_NAME AS OBJECT_NAME FROM information_schema.COLUMNS
    UNION
    SELECT 'VIEW' AS OBJECT_TYPE,TABLE_NAME AS OBJECT_NAME FROM information_schema.VIEWS
    UNION
    SELECT 'INDEX[Type:Name:Table]' AS OBJECT_TYPE,CONCAT (CONSTRAINT_TYPE,' : ',CONSTRAINT_NAME,
    ' : ',TABLE_NAME) AS OBJECT_NAME FROM information_schema.TABLE_CONSTRAINTS
    UNION
    SELECT ROUTINE_TYPE AS OBJECT_TYPE,ROUTINE_NAME AS OBJECT_NAME FROM information_schema.ROUTINES
    UNION
    SELECT 'TRIGGER[Schema:Object]' AS OBJECT_TYPE,CONCAT (TRIGGER_NAME,' : ',EVENT_OBJECT_SCHEMA,
    ' : ',EVENT_OBJECT_TABLE) AS OBJECT_NAME FROM information_schema.triggers
) R where object_name like "% %"
```

2. Complex SQL Query (Daniel 2016)

```
SELECT COLUMNS FROM (
    SELECT COLUMNS FROM (
        SELECT DISTINCT COLUMNS FROM table000 alias000 inner join table000 alias000 ON column000 = table000.column000 left join (
            SELECT COLUMNS FROM (
                SELECT DISTINCT COLUMNS FROM COLUMNS WHERE conditions
            ) AS alias000
            GROUP BY COLUMNS
        ) alias000
        ON conditions WHERE conditions
    ) AS alias000
    left join (
        SELECT COLUMNS FROM many_tables WHERE  many_conditions
    )
) AS alias000 ON condition left join (
    SELECT COLUMNS FROM (
        SELECT COLUMNS FROM many_tables WHERE many_conditions
    )
) AS alias001, (
    SELECT many_columns FROM many_tables WHERE many_conditions
) AS alias001 ON condition ORDER BY column001
```

3. Basic SQL Queries

Select * from

Update table1 set column = 'blah'

Select * from ()

Select "blah from table1

Select * from table1 where blah = 1)

Select column1, "blah' from table1

APPENDIX D - MySQL QUERY FEEDBACK

1. 1064 ERROR

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near

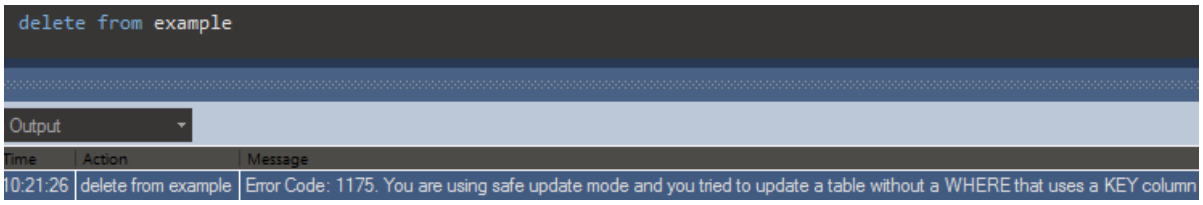
2. MySQL SERVER, CLIENT AND IDEAL FEEDBACK COMPARISON

MySQL Workbench was also used for server feedback but is not included as its feedback is almost identical to phpMyAdmin's parser feedback.

ID	Query	Problem	Server phpMyAdmin	Client MySQL Workbench	Ideal
Q1	*select * from*	No data source after "from"	1064 - (APPENDIX D 1) '' at line 1	"Unexpected end of input" on the "from" clause.	There is no table or subquery after the "from" at character 9.
Q2	*Select column from example*	"column" is a reserved keyword	1064 - (APPENDIX D 1) 'column from example' at line 1	"Unexpected 'column' (column)" on "column".	"Column" at character 7 is a reserved keyword, there should be a column name there instead.
Q3	*select * from ()*	No subquery between the brackets	1064 - (APPENDIX D 1) ')' at line 1	"Unexpected ')' (closing parenthesis)" on ")".	The subquery at character 9 is empty.
Q4	*select "blah from example*	Unclosed double quote before "blah"	Unclosed quote @ 7 1064 - (APPENDIX D 1) '"blah from example' at line 1	"Unexpected end of input" on the "select" clause. "Unfinished double quote string" on the " and after.	The " at character 7 does not have a matching " in the query, the number of " needs to be even.
Q5	*select * from example where column1 = 1)*	There is an unclosed bracket after the "1"	1064 - (APPENDIX D 1) ')' at line 1	"Extraneous input found - expected end of input" for ")".	The ")" at character 39 does not have a matching "(".
Q6	*select "blah' from example*	Two different quotes where there should be one type	Unclosed quote @ 7 1064 - (APPENDIX D 1) '"blah' from example' at line 1	"Unexpected end of input" on the "select" clause. "Unfinished double quote string" on " and after.	The " at character 7 and ' at character 12 do not have matching quotes, the same quote type must be used for each quote.

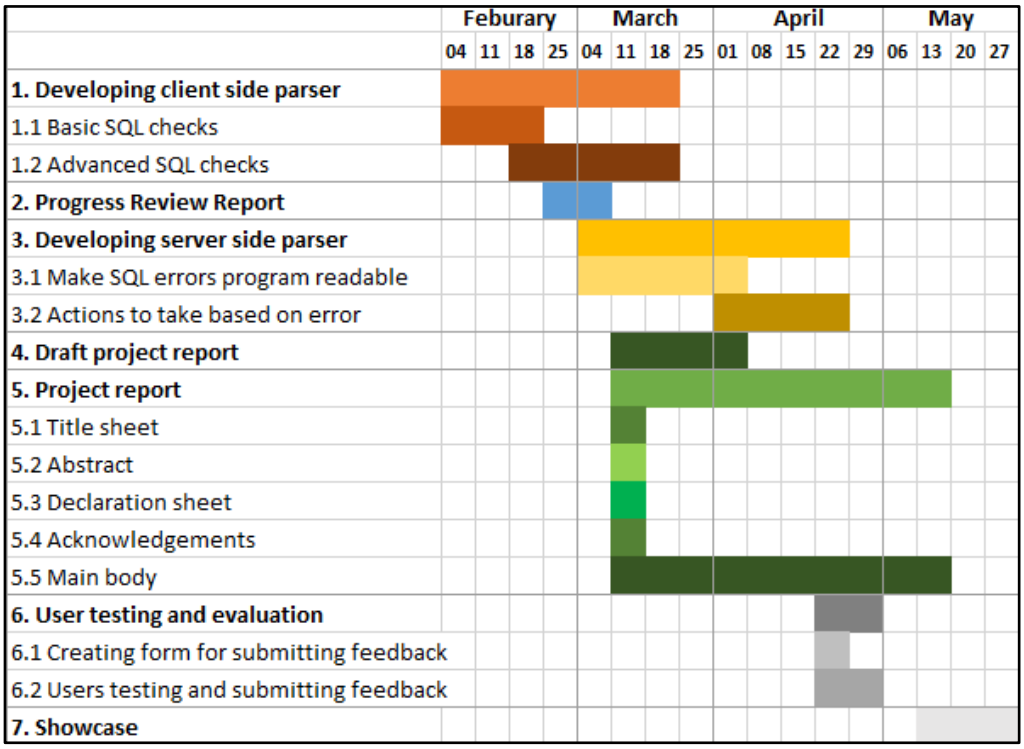
Q7	*select * from order by*	No table name and no column names after the order by clause	1064 - (APPENDIX D 1) 'order by' at line 1	“Unexpected ‘order’ (order)” on “order”. “unexpected end of input” on “by”	The “from” at character 9 is not followed by a table name or subquery The “order by” at character 14 needs column names after it.
Q8	*Select * from example order by column1 blah*	No comma between “column1” and “blah” after “order by”	1064 - (APPENDIX D 1) 'blah' at line 1	“Extraneous input found - expected end of input” for “blah”.	The column names “column1” and “blah” at character 31 should have a comma between them.
Q9	*select * from example where = 2*	no column name after “where”	1064 - (APPENDIX D 1) '= 2' at line 1	“unexpected ‘=’ (equal operator)” on “=”	The “where” clause at character 22 should be followed by a column name
Q10	*update example set column1 =*	no data after “=”	1064 - (APPENDIX D 1) '' at line 1	“unexpected end of input” on “=”	The operator “=” at character 27 should be followed by a value.
Q11	*insert into example (column1) values (1, 2)*	More data values than columns to insert into.	1136 - Column count doesn't match value count at row 1	None, is handled semantically on execution.	The “insert” statement starting at character 0 has 1 column and 2 values, the number of columns and values should match
Q12	*select * from example where (blah like'1' | blah like '2')*	| is not a valid operator, should be ||	1064 - (APPENDIX D 1) '| blah like '2')' at line 1	“missing closing parenthesis” on “‘1’”	The “|” at character 42 is not a valid character, “||” may be suitable.
Q13	*select * from example where blah*	“where” used but no conditions for “blah”	None, is syntactically valid but no results	None, syntactically is valid but no results	The column name “blah” at character 28 should be followed by an operator and value.
Q14	*update example set column1 = '2' blah like '1'*	There should be a “where” clause before “blah”	1064 - (APPENDIX D 1) 'blah like '1'' at line 1	“‘blah’ (identifier) is not valid input at this position” on “blah”	The “update” statement at character 0 seems to be lacking a “where” clause.
Q15	*delete example where blah = '1'*	“Delete” should be followed by a “from”	1064 - (APPENDIX D 1) 'where column1 = '1'' at line 1	“‘where’ (where) is not valid input at this position” on “where”	The “delete” clause at character 0 should be followed by a “from” clause
Q16	*select * update from example*	“update” is not valid in the select query	1064 - (APPENDIX D 1) 'update from example' at line 1	“unexpected ‘update’ (update) on “update”	The “update” at character 9 is not valid in a “select” query.

APPENDIX E - MYSQL WORKBENCH SAFE MODE

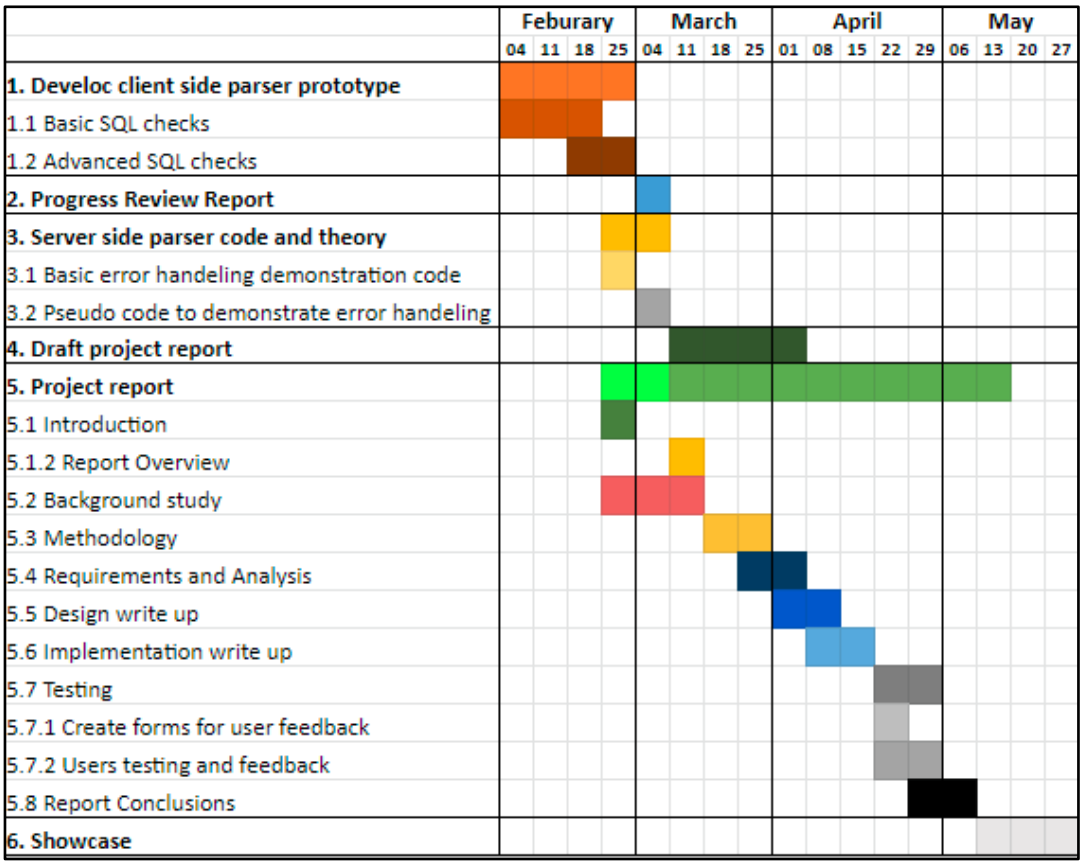


APPENDIX F - PROJECT PLANS AND GANTT CHARTS

1. GANTT CHART V1



2. GANTT CHART V2 6/3/2019



APPENDIX G - RISK MATRIX

Based on an existing template risk matrix (HiSlide undated).

Very low 1 - 2	Low 3 - 4	Medium 5 - 12	High 13 - 21	Very high 21 - 25

Probability	**Very High 5**	medium 5	medium 10	high 15	high 20	very high 25
	High 4	low 4	medium 8	medium 12	high 16	high 20
	Medium 3	low 3	medium 6	medium 9	medium 12	high 15
	Low 2	very low 2	low 4	medium 6	medium 8	medium 10
	Very Low 1	very low 1	very low 2	low 3	low 4	medium 5
		Very Low 1	**Low 2**	**Medium 3**	**High 4**	**Very High 5**
		Impact				

APPENDIX H - PROTOTYPE AND MYSQL WORKBENCH COMPARISON

ID	Query	Problem	Workbench	Prototype
C1	*select * from*	No table name after the “from”	“Unexpected end of input” on the “from” clause.	The "from" clause at character 9 is missing one of the following: databaseObjectName. This may be a table name, column name, data type etc.
C2	*Select column from example*	“Column” is a reserved keyword	“Unexpected ‘column’ (column)” on “column”.	The clause "select" at character 0 cannot have the reserved clause "column" at character 7
C3	*select * from ()*	No subquery between brackets	“Unexpected ‘)’ (closing parenthesis)” on “)”.	There is no text between "from" at character 9 and ")" at character 16 but one or more of the following are required: databaseObjectName
C4	*select "blah from example*	Unclosed double quote before “blah”	“Unexpected end of input” on the “select” clause. “Unfinished double quote string” on the “ and after.	There is an unclosed ", if it is not in a text string and escaped (\") then it may cause an error. ""blah" at column 7 may be misspelt, here are some spelling suggestions: undefined
C5	*select * from example where column1 = 1)*	Unclosed bracket after “1”	“Extraneous input found - expected end of input” for “)”.	The) at column 39 is unbalanced and may cause an error.
C6	*select "blah' from example*	Two quotes where there should be one type	“Unexpected end of input” on the “select” clause. “Unfinished double quote string” on “ and after.	There is an unclosed ', if it is not in a text string and escaped (') then it may cause an error. There is an unclosed ", if it is not in a text string and escaped (\") then it may cause an error. Single and double quotes are used, if possible use one type for consistency.
C7	*select * from order by*	No data source after “from” and no column/s after “by”	“Unexpected ‘order’ (order)” on “order”. “unexpected end of input” on “by”	There is no text between "from" at character 9 and "order" at character 14 but one or more of the following are required: databaseObjectName The "by" clause at character 20 is missing one of the following: databaseObjectName. This may be a table name, column name, data type etc.
C8	*Select * from example order by*	No comma between “column1” and “blah” after “by”	“Extraneous input found - expected end of input” for “blah”.	Object names after clause "by" at character 28 should have one of the following separators between its values: , "column1" at column 31 may be misspelled, here are some spelling suggestions:

	column1 blah			column's,column,columns,columned
C9	*select * update from example*	“update” is not valid in the select query	“unexpected ‘update’ (update) on “update”	The clause "select" at character 0 cannot have the reserved clause "update" at character 9 The clause "update" at character 9 cannot have the reserved clause "from" at character 16 The "update" clause at character 9 is missing one of the following clauses after it: set
C10	*select * from example group by avg(blah)*	Cannot use the aggregate function “avg” with “group by”	Error Code: 1111. Invalid use of group function	The clause "by" at character 28 cannot have the reserved clause "avg" at character 31 Server message: Error 1111 one of the group by clauses are not used correctly, character/s: 22. Group by should be followed by column names with , separating them. The following aggregate functions appear to be with the group clause but should be used with a Having clause: avg at 31,
C11	*insert into example (column1) values (1, 2)*	More data values than columns to insert into.	None, while appearing to be a syntax error it is handled semantically on execution.	" insert" at column 0 may be misspelt, here are some spelling suggestions: insert,inset,inert,insect Objects after clause "into" at character 8 need one of the following separators between its values: , The "into" clause at character 8 is missing one of the following clauses after it: values,value "(column1" at column 21 may be misspelt, here are some spelling suggestions: undefined "values" at character 32 can only have 1 value between it and the next clause The "values" clause at character 32 is missing one of the following clauses after it:)
C12	*select * from example where (blah like '1' | blah like '2')*	| is not a valid operator, should be ||	“missing closing parenthesis” on “‘1’”	The clause "where" at character 22 cannot have the reserved clause "like" at character 34 "where" at character 22 can only have 1 value between it and the next clause The clause "where" at character 22 cannot have the reserved clause "like" at character 50 "(blah" at column 28 may be misspelt, here are some spelling suggestions: undefined
C13	*select * from example where blah*	“where” used but no conditions for “blah”	None, is valid but no results	None, is valid but no results
C14	*update example set column1 =*	Should be a “where” before “blah”	“‘blah’ (identifier) is not valid input at this position” on “blah”	"set" at character 15 can only have 1 value between it and the next clause

	'2' blah like '1'			Objects after clause "set" at character 15 need one of the following separators between its values: , The "like" clause at column 38 is missing one of the required following clauses: "where, " The clause "set" at character 15 cannot have the reserved clause "column" at character 19 The clause "set" at character 15 cannot have the reserved clause "like" at character 38
C15	*delete example where blah = '1'*	“Delete” should be followed by a “from”	“‘where’ (where) is not valid input at this position” on “where”	The "where" clause at column 15 is missing one of the required following clauses: "from, update, " "where" at character 15 can only have 1 value between it and the next clause

APPENDIX I - REQUIREMENTS CASE TESTS

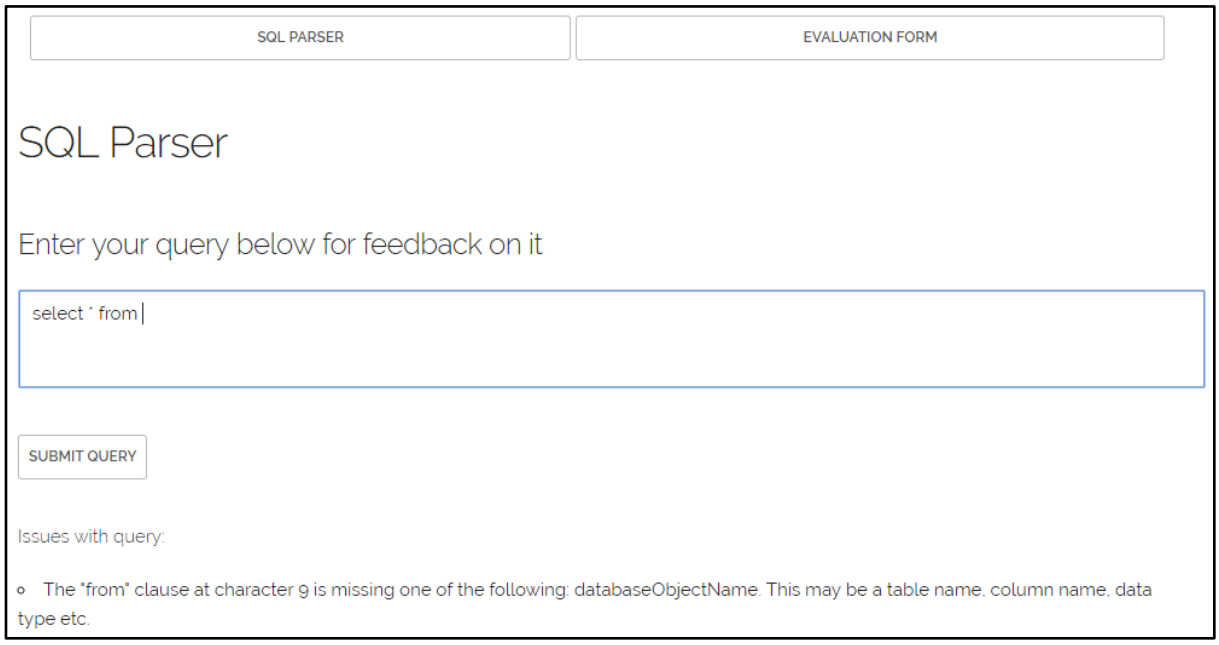
Mos cow	ID	Req ID	Test	Expected outcome	Actual outcome
MUST	RT 1	FR1	Type text into the pages text area	Feedback is provided	Pass, List element is updated with feedback after each keypress
	RT 2	FR2	Enter the invalid query “selec * from example” into the text area	Feedback is provided indicating the “selec” should be “select”	Pass, feedback is provided that indicates the query is lacking a “select” and that “selec” is misspelt: *"selec" at column 0 may be misspelt, here are some spelling suggestions: silica,sleek,sulk,slick* *The "from" clause at column 8 is missing one of the required following clauses: "select, "*
	RT 3	FR3	Enter the query “select mispelt from example” into the text area	Feedback is provided indicating “mispelt” is misspelt with a spelling suggestion provided.	Pass, feedback with correct spelling is provided: *"mispelt" at column 7 may be misspelt, here are some spelling suggestions: misspelt,misspell*
	RT 4	FR3	Host and connect to the prototype over the internet	Prototype web page and functions work over the internet	Pass, prototype functions work over the internet.
	RT 5	FR4	Enter the query “select by from example” into the text area	Feedback is provided indicating “by” is a MySQL reserved clause and not valid next to the “select”	Pass, feedback is provided indicating “by” is reserved and not valid with “select”: *The clause "select" at character 0 cannot have the reserved clause "by" at character 7* *The clause "by" at character 7 cannot have the reserved clause "from" at character 10*
	RT 6	FR5	Enter the query “select * from example by order blah” into the text area	Feedback is provided indicating that “order” should be before “by”	Pass, feedback is provided indicating “by” should be after “order”: *The "order" clause at character 25 is missing one of the following clauses after it: by*
	RT 7	FR6	Enter the query “select (blah" from example” into the text area	Feedback indicating that there are unbalanced (and “	Pass, feedback indicates the (and “ are unbalanced: *The (at column 7 is unbalanced and may cause an error.* *There is an unclosed ", if it is not in a text string and escaped (\") then*

					it may cause an error.
SHOULD	RT 8	FR7	Submit the query “delete from example” in the prototype then “select * from example” in MySQL Workbench	The delete query being valid in the prototype and the 2nd query returning data.	Pass, prototype finds no issue with query and example table has data.
	RT 9	NR1	Enter the query “select column from example” into the text area	Feedback indicating the “column” at character 7 is invalid.	Pass, feedback indicates the “column” at character 7 is the issue: *The clause "select" at character 0 cannot have the reserved clause "column" at character 7*
	RT 10	NR2	Execute the query “select * from example group by avg(blah)”	Feedback indicating the “avg” is invalid after the “group by” and without a “having”.	Partial, feedback indicates the lack of a having and avg to be the issue but this has only been coded for 1 error. *Server message: Error 1111 one of the group by clauses are not used correctly, character/s: 22. Group by should be followed by column names with , separating them.* *The following aggregate functions appear to be with the group clause but should be used with a Having clause: avg at 31,*
	RT 12	NR4	Enter invalid queries involving selects, inserts, updates, deletes, joins, grouping, ordering etc.	Feedback indicating the cause for each invalid clause	Partial, coverage of updates, inserts and joins not implemented due to insufficient time.
COULD	RT 11	NR3	Execute a semantically invalid query	A section on the page updates with results from a StackOverflow search of the error number	Fail, not implemented due to insufficient time
	RT 15	NR7	Enter the query “*select * from ‘example*” into the text area	Feedback provided indicating that there should not be a quote after the “from”	Fail, not implemented due to insufficient time.
	RT 13	NR5	Change the connected database in the settings menu	Database the prototype is connected to changes	Fail, not implemented due to insufficient time
	RT 14	NR6	Open object browser for connected database	Database objects are shown in the object	Fail, not implemented due to insufficient time

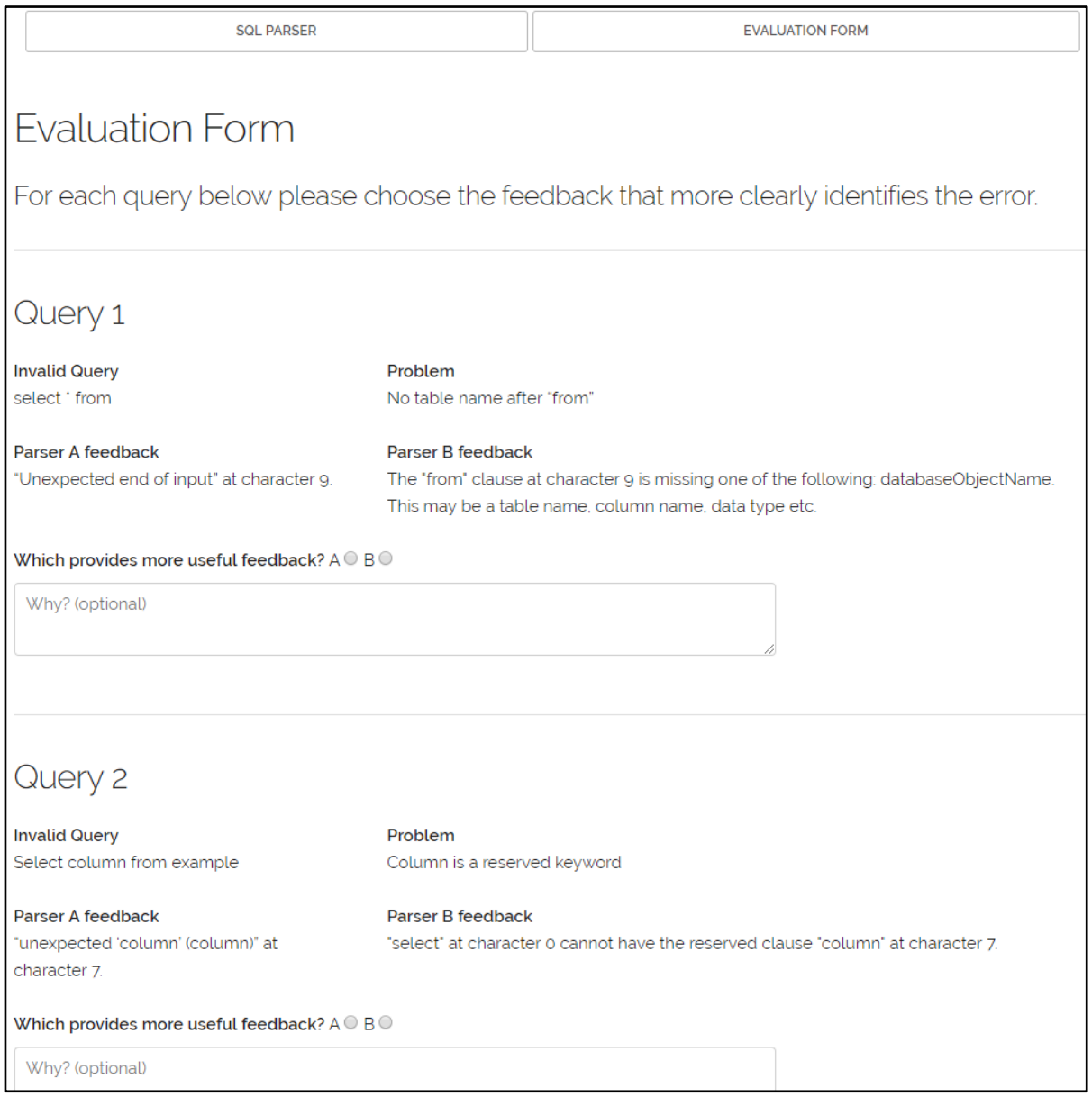
WOULD				browser	
	RT 16	NR8	Enter invalid queries involving casting, converting, datepart, views etc into the text area	Feedback provided that indicating the error cause for each function	Fail, not implemented due to insufficient time
	RT 17	NR9	Type the query “select * from” into the text area	Feedback provided suggesting a table name	Fail, not implemented due to insufficient time
	RT 19	NR11	Enter invalid queries involving stored procedures, triggers, pivot and other advanced functions.	Feedback provided that indicates the cause for the advanced function	Fail, not implemented due to insufficient time

APPENDIX J - PROTOTYPE SCREENSHOTS

1. PARSER WEB PAGE



2. EVALUATION FORM WEB PAGE



APPENDIX K - EVALUATION FORM RESULTS

ID: Identification for the row inserted to the database
Survey ID: Identification for the whole survey that the user completes
Query: The invalid MySQL query the user is evaluating the feedback for
Feedback: Query the user chooses, A is MySQL Workbench and B is the prototype
Date: Date and time the survey was completed and added to database
Comments: Any comments the user made on why they choose the feedback

ID	Survey ID	Query	Feedback	Date	Comments
166	2	select * from	B	01/04/2019 15:54	Details that a table name is required
167	2	Select column from example	B	01/04/2019 15:54	Details that column is wrong due to being a reserved clause
168	2	select * from ()	B	01/04/2019 15:54	Details that there should be text between the brackets
169	2	select "blah from example	A	01/04/2019 15:54	A details where the quote is but B does not
170	2	select * from example where column1 = 1)	B	01/04/2019 15:54	B details that the bracket is unbalanced while A does not
171	2	select "blah' from example	A	01/04/2019 15:54	A details where the quotes are
172	2	select * from order by	B	01/04/2019 15:54	B details that the from and by are missing column names
173	2	Select * from example order by column1 blah	B	01/04/2019 15:54	B details that a comma is required
174	2	select * update from example	B	01/04/2019 15:54	B details that there are missing clauses
175	2	select * from example group by avg(blah)	B	01/04/2019 15:54	B details why the error has occurred
176	3	select * from	A	02/04/2019 18:07	
177	3	Select column from example	B	02/04/2019 18:07	
178	3	select * from ()	A	02/04/2019 18:07	
179	3	select "blah from example	A	02/04/2019 18:07	
180	3	select * from example where column1 = 1)	B	02/04/2019 18:07	
181	3	select "blah' from example	B	02/04/2019 18:07	
182	3	select * from order by	B	02/04/2019 18:07	
183	3	Select * from example order	B	02/04/2019 18:07	

		by column1 blah			
184	3	select * update from example	A	02/04/2019 18:07	
185	3	select * from example group by avg(blah)	A	02/04/2019 18:07	
186	4	select * from	B	03/04/2019 17:12	More information given
187	4	Select column from example	B	03/04/2019 17:12	More information given
188	4	select * from ()	B	03/04/2019 17:12	More information given
189	4	select "blah from example	A	03/04/2019 17:12	Tells you where the problem is
190	4	select * from example where column1 = 1)	B	03/04/2019 17:12	
191	4	select "blah' from example	B	03/04/2019 17:12	More information given
192	4	select * from order by	B	03/04/2019 17:12	More information given
193	4	Select * from example order by column1 blah	B	03/04/2019 17:12	Tells you where and what the problem is
194	4	select * update from example	B	03/04/2019 17:12	More information given
195	4	select * from example group by avg(blah)	B	03/04/2019 17:12	More information given
196	5	select * from	B	05/04/2019 09:59	
197	5	Select column from example	A	05/04/2019 09:59	
198	5	select * from ()	B	05/04/2019 09:59	
199	5	select "blah from example	A	05/04/2019 09:59	
200	5	select * from example where column1 = 1)	A	05/04/2019 09:59	
201	5	select "blah' from example	A	05/04/2019 09:59	
202	5	select * from order by	A	05/04/2019 09:59	
203	5	Select * from example order by column1 blah	B	05/04/2019 09:59	
204	5	select * update from example	A	05/04/2019 09:59	
205	5	select * from example group by avg(blah)	A	05/04/2019 09:59	
216	6	select * from	B	05/04/2019 11:40	Clear instructions, could reduce time spent

217	6	Select column from example	B	05/04/2019 11:40	
218	6	select * from ()	B	05/04/2019 11:40	
219	6	select "blah from example	B	05/04/2019 11:40	
220	6	select * from example where column1 = 1)	B	05/04/2019 11:40	
221	6	select "blah' from example	B	05/04/2019 11:40	
222	6	select * from order by	B	05/04/2019 11:40	
223	6	Select * from example order by column1 blah	B	05/04/2019 11:40	
224	6	select * update from example	B	05/04/2019 11:40	
225	6	select * from example group by avg(blah)	B	05/04/2019 11:40	This type of error log is much clearer
226	7	select * from	B	05/04/2019 14:33	Provides more information about the problem
227	7	Select column from example	B	05/04/2019 14:33	provides the user with detailed explanation why the problem has occurred and where
228	7	select * from ()	B	05/04/2019 14:33	provides the user with detailed explanation why the problem has occurred and where
229	7	select "blah from example	B	05/04/2019 14:33	provides the user with detailed explanation in a more user understandable way why the problem has occurred and where
230	7	select * from example where column1 = 1)	A	05/04/2019 14:33	provides the user with detailed explanation in a more user understandable way why the problem has occurred and where
231	7	select "blah' from example	B	05/04/2019 14:33	provides the user with detailed explanation in a more user understandable way why the problem has occurred and where
232	7	select * from order by	B	05/04/2019 14:33	provides the user with detailed explanation in a more user understandable way why the problem has occurred and where

233	7	Select * from example order by column1 blah	B	05/04/2019 14:33	provides the user with detailed explanation in a more user understandable way why the problem has occurred and where
234	7	select * update from example	B	05/04/2019 14:33	provides the user with detailed explanation in a more user understandable way why the problem has occurred and where
235	7	select * from example group by avg(blah)	B	05/04/2019 14:33	provides the user with detailed explanation in a more user understandable way why the problem has occurred and where
236	8	select * from	B	06/04/2019 22:20	More verbose and human readable
237	8	Select column from example	B	06/04/2019 22:20	Clearer intention
238	8	select * from ()	B	06/04/2019 22:20	more verbose
239	8	select "blah from example	B	06/04/2019 22:20	more useful for a beginner
240	8	select * from example where column1 = 1)	A	06/04/2019 22:20	more information
241	8	select "blah' from example	B	06/04/2019 22:20	more detail
242	8	select * from order by	B	06/04/2019 22:20	clearly laid out information
243	8	Select * from example order by column1 blah	B	06/04/2019 22:20	tells you what to do to fix it
244	8	select * update from example	B	06/04/2019 22:20	easier to understand
245	8	select * from example group by avg(blah)	B	06/04/2019 22:20	More verbose, easier to read, more information
246	9	select * from	B	07/04/2019 17:26	Specifies what the problem might be.
247	9	Select column from example	B	07/04/2019 17:26	Specification
248	9	select * from ()	B	07/04/2019 17:26	Specification
249	9	select "blah from example	B	07/04/2019 17:26	Specification
250	9	select * from example where column1 = 1)	A	07/04/2019 17:26	Specification
251	9	select "blah' from example	B	07/04/2019 17:26	Specification

252	9	select * from order by	B	07/04/2019 17:26	Specification
253	9	Select * from example order by column1 blah	B	07/04/2019 17:26	Specification
254	9	select * update from example	B	07/04/2019 17:26	Specification
255	9	select * from example group by avg(blah)	B	07/04/2019 17:26	Specification
256	10	select * from	B	10/04/2019 10:53	more detail and easier to understand
257	10	Select column from example	A	10/04/2019 10:53	feedback is simple to understand and tells message properly
258	10	select * from ()	B	10/04/2019 10:53	Data Valid
259	10	select "blah from example	B	10/04/2019 10:53	yes
260	10	select * from example where column1 = 1)	A	10/04/2019 10:53	feedback is good
261	10	select "blah' from example	A	10/04/2019 10:53	yes
262	10	select * from order by	B	10/04/2019 10:53	shows where issue is
263	10	Select * from example order by column1 blah	A	10/04/2019 10:53	the feedback is useful
264	10	select * update from example	B	10/04/2019 10:53	helps show update is not valid
265	10	select * from example group by avg(blah)	B	10/04/2019 10:53	shows errors in data