MASTER

Towards successful interaction between humans and databases

Miedema, D.E.

Award date:
2019

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
Towards successful interaction between Humans and Databases

*Master Thesis*

Daphne Miedema

Supervisors:
Dr. G.H.L. Fletcher
Dr.ir. P.A.M. Ruijten

Assessors:
Dr. J.R.C. Ham
Dr. A. Serebrenik

Eindhoven, September 2019
Abstract

Though we live in an era in which every consumer produces large amounts of data every day, data is still mostly stored by companies in a way that consumers can’t access. Usually, only parts of these data are available through company-provided applications. Even if the consumer would be able to gather all their data from the company it would be hard to interact with it as data is traditionally stored in (relational) databases. These databases are optimized for effective storage, powerful search mechanisms and fast access times, but require a lot of training to use effectively.

During our examinations of existing visual querying systems and literature on Human-Computer Interaction, we found that the research on the interaction between humans and databases is highly underexposed. To find out how this interaction functions in real life, we ran an experiment. The results show that novice users of a database system can highly improve their performance by using our support system.

In this thesis, we explore accessible data interaction techniques in order to improve interaction for any type of user. To this end, we create a new database system with a visual support mechanism by means of a user-centered design process. A user study for users with varying levels of experience was carried out to evaluate the proposed technique. Their feedback supports our hypothesis that a visual representation can support users in querying SQL databases.
Acknowledgements

First of all, my deepest gratitude goes to my supervisors dr. George Fletcher and dr. ir. Peter Ruijten. Your questions, encouragement and most of all, guidance, elevated this work to what it is now. A combined thesis, such as this one, requires hard work to balance the scales. I am happy to say that I felt we were able to form a team working towards a single goal. Our shared interest in this topic leads me to pursue a Ph.D., such that I can continue my research to make databases more accessible.

Next, I would like to thank Wilco van Leeuwen for lots of technical and administrative support. Your assistance made the execution of the experiments much easier.

Furthermore, many thanks go out to all individuals who participated in the experiments for sharing their time and experiences with me. Without them, this thesis would not have been possible.

A special mention goes to Anne Grave, for sharing all the tips and tricks for successfully pulling off a double graduation project.

And finally, I would like to thank my boyfriend Dennis for always keeping up the spirits high. No matter where the difficulties lay, writing, coding, experimenting or presenting, you were always there for support. I really appreciate everything you do.
Contents

1 Introduction 1
   1.1 Thesis objectives .............................................. 2
   1.2 Thesis structure .............................................. 2

2 Preliminaries 3
   2.1 Demarcation of the topic ................................. 3
      2.1.1 Target users .................................. 3
      2.1.2 Successful ..................................... 3
      2.1.3 Interaction .................................... 4
   2.2 CSE concepts ............................................. 4
      2.2.1 Database basics .................................. 4
      2.2.2 Database operations .................................. 5
      2.2.3 Graphs and Graph Data ............................. 6
   2.3 History of graph databases .............................. 8
      2.3.1 Foundations .................................... 8
      2.3.2 Object Oriented databases and XML ................. 9
      2.3.3 NoSQL and GraphDBs ............................... 9
      2.3.4 RDF and SPARQL .................................. 9
      2.3.5 Graph Query Languages ............................. 10

3 An exploration of existing systems. 11
   3.1 Introduction ............................................. 11
   3.2 What is known about visualizing graph data? .......... 12
      3.2.1 Visualization design ............................... 12
      3.2.2 The issue of scale ................................ 13
      3.2.3 Representations and layouts ....................... 13
      3.2.4 Information visualization and HCI ................ 14
   3.3 Demarcation of the topic ................................ 17
      3.3.1 Visual query systems ................................ 17
      3.3.2 Graph query languages ............................. 19
   3.4 State of the art in VGQS ................................. 20
      3.4.1 Taxonomy ......................................... 20
      3.4.2 Summaries ......................................... 22
      3.4.3 Findings ........................................ 23
   3.5 Conclusion .............................................. 24

4 An analysis of factors in Human-Database Interaction 25
   4.1 Introduction ............................................. 25
   4.2 Support in literature ..................................... 26
      4.2.1 Human factors in querying and programming ....... 27
## CONTENTS

4.2.2 Observations ........................................... 29  
4.3 Technical factors ........................................... 29  
  4.3.1 Language ........................................... 30  
  4.3.2 Understanding a query ................................. 32  
  4.3.3 Consistency ........................................... 33  
  4.3.4 Technology use through errors and mistakes ........... 34  
  4.3.5 Summary ........................................... 36  
4.4 Task characteristics ........................................... 37  
  4.4.1 Task definition ........................................... 37  
  4.4.2 Querying tasks ........................................... 38  
  4.4.3 Interface tasks ........................................... 39  
  4.4.4 Summary ........................................... 40  
4.5 Human factors ........................................... 40  
  4.5.1 Cognition ........................................... 40  
  4.5.2 Decision making ........................................... 51  
  4.5.3 Summary ........................................... 52  
4.6 Factors of the individual ........................................... 53  
4.7 Conclusion ........................................... 54  

5 Human-Database Interaction in practice .......................... 57  
  5.1 Introduction ........................................... 57  
  5.2 Related work ........................................... 58  
  5.3 Methodology ........................................... 59  
  5.4 Results ........................................... 60  
  5.4.1 Correctness and attempts ........................................... 60  
  5.4.2 Query similarity ........................................... 61  
  5.4.3 Query traces ........................................... 63  
  5.4.4 Questionnaire ........................................... 64  
  5.4.5 Results summary ........................................... 65  
  5.5 Raw text analysis ........................................... 67  
  5.5.1 Text analysis summary ........................................... 69  
  5.6 Query Complexity analysis ........................................... 72  
  5.6.1 Theory ........................................... 72  
  5.6.2 Comparing queries ........................................... 73  
  5.6.3 Experience ........................................... 74  
  5.6.4 Comparison of metrics ........................................... 76  
  5.6.5 Nesting metrics ........................................... 77  
  5.6.6 Outliers ........................................... 80  
  5.6.7 Other remarks ........................................... 85  
  5.6.8 Query complexity summary ........................................... 85  
  5.7 Conclusion ........................................... 87  

6 System Design ........................................... 89  
  6.1 Problem analysis ........................................... 89  
    6.1.1 Stakeholders ........................................... 89  
    6.1.2 Questions ........................................... 89  
    6.1.3 User tasks ........................................... 90  
    6.1.4 Requirements ........................................... 90  
    6.1.5 Use case ........................................... 91  
  6.2 User-Centered Design ........................................... 92  
  6.3 Visual Query representation ........................................... 92  
    6.3.1 Literature ........................................... 92  
    6.3.2 Methodology ........................................... 93  
    6.3.3 Results ........................................... 94
## CONTENTS

6.3.4 Discussion .................................................. 95
6.4 Interface Design - Prototype 1 ................................ 95
6.5 Paper prototype experiment .................................. 96
  6.5.1 Methodology .................................................. 96
  6.5.2 Results ....................................................... 98
  6.5.3 Discussion ................................................... 100
6.6 Interface Design - Prototype 2 ............................... 101
  6.6.1 Implementation .............................................. 101
  6.6.2 Visualization area .......................................... 101
  6.6.3 Database schema ............................................ 104
  6.6.4 Help function ............................................... 104
  6.6.5 Query history ............................................... 104
  6.6.6 Other aspects ............................................... 105
  6.6.7 Colour psychology ......................................... 105
6.7 User test ....................................................... 105
  6.7.1 Methodology .................................................. 106
  6.7.2 Results ....................................................... 106
  6.7.3 Discussion ................................................... 109
6.8 Conclusion ..................................................... 110

7 Conclusions ..................................................... 111
  7.1 Future work .................................................... 112

References ........................................................ 113

Appendices ........................................................ 123

A Raw Text Analysis Examples .................................... 123
  A.1 Examples of Inefficient Formulations ...................... 123
  A.2 Examples of Violations of Standard Patterns ............. 125

B Visualisation Experiment Results ................................ 127

C Paper Prototype Experiment Findings .......................... 137
  C.1 Designs .......................................................... 137
  C.2 Findings by participant ...................................... 141
  C.3 Findings by category ......................................... 143
Chapter 1

Introduction

While the reach of the Internet is rapidly expanding, the amount of data created and stored through it has been increasing exponentially. According to the IDC, the volume of digital data generated annually will grow from 33ZettaByte in 2018 to 175ZettaByte in 2025 (Coughlin, 2018). This data is generated by all members of society, through all kinds of devices and applications including smart thermostats and cars. The software controlling these items generates data. The analyses that can be performed on this data, provide the products’ manufacturers with a lot of information. On a more personal scale, humans also create data using applications for tracking food and calorie intake, movement and location, as well as tracking how well they sleep. Many people are looking for ways to understand and interpret their data correctly.

Large sets of data are most efficiently stored in databases. They allow for advanced interaction with the data, such as: accessing data to pose questions (also known as queries), adding new data, changing the structure of the data, conditionally applying changes to the data, and so on. Databases typically use a specific type of programming language named a query language, to run these interactions. Unfortunately, this means that to use a database, a user needs to become proficient in its associated language. This is simply not feasible for the users that are looking for a simple method to interact with their data.

On top of the significant amount of training needed for effective use of databases, queries also require considerable cognitive effort on the part of the user. Mitrovic (1998) notes that this high cognitive load may originate both in the burden to memorize a database schema, and in the fact that only certain types of errors are intercepted by the database management system. These, among other aspects, contribute to suboptimal usability of databases and prevent untrained users (novices) from analyzing their data. We aim to alleviate these usability problems.

Recent developments in the database research community focus on graph databases. Instead of implying connections between data entries, these graph databases explicitly store all existing connections in the data. This allows graph data to be visualized in a more straight-forward manner, as this data with connections can be drawn as a graph. The more explicit nature of the data also facilitates understanding, as conceptually linked data is also physically linked. The graph query languages for this data are also more visual in nature, leading to more intuitive interaction.

From the perspective of database usability, there is also a lack of research that focuses on this topic. We found only a few researchers working on analyzing the usability of databases. Typically, database systems are developed by database engineers without taking research in Human-Computer Interaction into account. The researchers in these fields work independently, without leveraging principles from one another to improve their work (Bhowmick, 2014). This means that there are systems that adapt to the proficiency of the user, but lack in performance, as well as highly performing systems that lack in user experience design. Any additions to research on database usability, and thus collaboration between the two fields, will lead to improved interactions, as neither field will be an afterthought.

The lack of collaboration also shows in the amount of published research on the topic of database interaction. Some works do exist, but they are far from recent (Reisner, 1981; Ehrenreich,
CHAPTER 1. INTRODUCTION

1981; Landauer, Dumais, Gomez & Furnas, 1982). This means that they only take query languages of that period into account. In the years since these papers were published, many new query languages have been developed. New research on this topic can show whether old principles work for new languages, and can elaborate on interaction with new types of features such as graph databases.

In this thesis, we will contribute to the research on Human-Database Interaction. We review the literature from the past decades, run experiments to compare theory to practice, and develop a database tool through User-Centered Design that should facilitate the interaction with novice users.

1.1 Thesis objectives

In this thesis, we address the following research question:

How can we create tools to facilitate interaction between novice users and databases?

Existing research hypothesizes that the complexity of traditional databases lies in the memorization of the database schema (Mitrovic, 1998). Thalheim (2003) states that visualizations support understanding of a query and its connections. We hypothesize that a visual representation of the interactions will support (novice) users, as it directly displays the relations. Kerren, Stasko, Fekete and North (2008) write that a benefit of data visualization is that it lowers the amount of brainpower required for cognitive processes, through acting as a frame of reference or temporal storage. In short, we hypothesize that the addition of a visualization to a database system would reduce cognitive load and thus improve the interaction between users and databases.

There have been a few recent papers on visualizing SQL that support our hypothesis. For example, Thalheim (2003) found “visualization led [...] to higher conceptual correctness and conceptual completeness.” and Gatterbauer (2011) writes that one of the options to help users interpret a query is to visualize it.

To gather further support for this hypothesis, we pose the following subquestions:
1. What are some existing visual querying systems?
2. Which human factors play a role in the interaction with databases?
3. How do users interact with existing database systems?
4. What is an intuitive way to visualize database searches?
5. What does an ideal database interface look like for a novice user?
6. How do (novice and expert) users interact with this new system?
7. Is the new system an improvement on existing systems?

1.2 Thesis structure

This thesis is structured as follows: in Chapter 2 we set up some context for the rest of the thesis. This is followed by an exploration of existing visual querying systems in Chapter 3 to answer subquestion 1. We cover the interaction with the user in Chapter 4 where we examine the human factors involved in database interaction, corresponding to subquestion 2. Then, we answer subquestion 3 by examining Human-Database interaction in practice, with a case study in Chapter 5. In Chapter 6, we describe our user-centered design cycle: creating, implementing and evaluating various prototypes. This chapter answers subquestion 4, 5, 6 and 7, and discuss its consequences for the interaction design cycles. We finalize this thesis by presenting conclusions, limitations and future work in Chapter 7.
Chapter 2

Preliminaries

In this chapter will describe some concepts that occur regularly in this thesis. This ensures that writer and reader have a common language. In addition, as this is a multidisciplinary work, the readers will not have a common background. Therefore, we introduce some basics of the disciplines of Computer Science and Human-Technology Interaction.

2.1 Demarcation of the topic

The research question of this thesis is:

How can we create tools to facilitate interaction between novice users and databases?

And the title of this thesis is:

Towards successful interaction between Humans and Databases

In this thesis we investigate opportunities to improve this interaction. To establish a common language, we explain some of our choices and elaborate on some concepts in the following sections.

2.1.1 Target users

Learning a query language such as SQL is challenging. As novice users do not master the language completely yet, it makes sense that they would make the most mistakes. This is why the focus of our main research question is on novices.

To make sure that the system is usable by anyone, the experiments that we undertake in this thesis are ran with the participants that suit it the best. These include novices, advanced beginners and Ph.D. students. These names indicate user levels throughout this thesis.

2.1.2 Successful

There are many concepts involved in determining whether an interaction has been successful. These include efficiency, effectiveness, user experience, and many others. In this thesis we do not aim for successful interaction as a concrete metric, but instead focus on improving the user experience and their performance. Therefore we do not define success as a quantitative metric. All studies in the User-Centered Design cycle, described in Chapter 6, are of the qualitative type. Therefore, we can evaluate whether our prototypes are a step in the right direction without measuring any numbers.
2.1.3 Interaction

The choice to title the thesis on ‘interaction between Humans and Databases’ instead of using the concept of Human-Database Interaction, was a conscious one. This field of research has been largely unexplored, and there does not exist a strict definition of what the term means. The related field of Human-Computer Interaction (HCI) has been a topic of research for several decades, and is relevant to the more specific field of Human-Database Interaction.

Intuitively, one can ‘feel’ what the concept of interaction means. The Oxford Dictionary defines it as “reciprocal action or influence”\(^1\). But interaction with a computer or a system, is influenced by more than just the user and the system. Culture, environment, experience and more play a role in the interaction. In Chapter 4, we take a look at many factors that influence this interaction. This, in turn, clarifies what the concept of Human-Database Interaction actually means according to us.

2.2 CSE concepts

2.2.1 Database basics

A database is a collection of data stored in the form of tables. Each table has a primary key, which are unique identifiers for that table, such as a citizen service number to refer to a person. Primary keys can also comprise multiple columns of a table. Tables are linked through the concept of foreign keys, which means that to refer to an item from another table, you use that item’s primary key. If a primary key of one table is used in another table, it refers to an external object and thus is foreign. The database knows which table to reference to connect the data. Hence the name foreign key. In describing databases, for example in the creation of a data model, the primary and foreign keys are often marked. Our preferred notation is to underline primary keys and use a dotted underline for foreign keys, but there are many different options.

The term database management system (DBMS) is an application layer between the database and an end-user. It facilitates interaction by allowing the user to search through the data (also called querying) by means of a dedicated query language. On top of this, it is also a tool for administrators to define, create and update databases. Database management systems often provide a textual interface, which requires the user to know the language associated with the DBMS.

The term database system is an umbrella term that refers to the collection of all applications, content and tools related to databases. It is a high-level definition and usually only distinguishes between types of systems.

Foundations

There are many types of database systems. For this paper it is relevant to know that the traditionally used system is the relational database system (relational databases). A major purpose of a database system is to provide users with an abstract view of the data. This means that the system hides certain details of how the data are stored and maintained (Silberschatz, Korth & Sudarshan, 2011). Programmers and database administrators access the logical level of abstraction, a database user only sees the view level. The first step in this abstraction of database systems was the relational model defined by Codd (1970). He suggested that management of the data should be independent of changes in data types and data representation.

A relational database system works as follows. Each separate type of data has its own table. If we return to our roadtrip example from the introduction, this means that there is a table for cities and a table for roads, as well as a table that represents the connections between roads and cities. In relational database tables, rows are called records, and columns are called attributes. The table for cities might contain attributes such as an ID for the city (which can be used as

\(^1\)https://www.lexico.com/en/definition/interaction
CHAPTER 2. PRELIMINARIES

Table 2.1: A relational database example.

<table>
<thead>
<tr>
<th>CID</th>
<th>name</th>
<th>population</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Amsterdam</td>
<td>853312</td>
</tr>
<tr>
<td>C2</td>
<td>Utrecht</td>
<td>344384</td>
</tr>
<tr>
<td>C3</td>
<td>'s Hertogenbosch</td>
<td>107870</td>
</tr>
<tr>
<td>C4</td>
<td>Leiden</td>
<td>123924</td>
</tr>
<tr>
<td>C5</td>
<td>'s Gravenhage</td>
<td>526439</td>
</tr>
<tr>
<td>C6</td>
<td>Breda</td>
<td>149505</td>
</tr>
<tr>
<td>C7</td>
<td>Eindhoven</td>
<td>227100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RID</th>
<th>name</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>A2</td>
<td>highway</td>
</tr>
<tr>
<td>R2</td>
<td>A4</td>
<td>highway</td>
</tr>
<tr>
<td>R3</td>
<td>A12</td>
<td>highway</td>
</tr>
<tr>
<td>R4</td>
<td>A27</td>
<td>highway</td>
</tr>
<tr>
<td>R5</td>
<td>A58</td>
<td>highway</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CID</th>
<th>RID</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>R1</td>
</tr>
<tr>
<td>C1</td>
<td>R2</td>
</tr>
<tr>
<td>C2</td>
<td>R1</td>
</tr>
<tr>
<td>C2</td>
<td>R3</td>
</tr>
<tr>
<td>C2</td>
<td>R4</td>
</tr>
<tr>
<td>C3</td>
<td>R1</td>
</tr>
<tr>
<td>C4</td>
<td>R2</td>
</tr>
<tr>
<td>C5</td>
<td>R2</td>
</tr>
<tr>
<td>C5</td>
<td>R3</td>
</tr>
<tr>
<td>C6</td>
<td>R4</td>
</tr>
<tr>
<td>C6</td>
<td>R5</td>
</tr>
<tr>
<td>C7</td>
<td>R1</td>
</tr>
</tbody>
</table>

foreign key), the name of the city, and other attributes such as the size of the city. An example database is given in Table 2.1.

For most relational databases, data is managed through the Structured Query Language (SQL or sequel), which includes functionality to insert, update, retrieve and delete data. SQL is mostly used for retrieving information from the database, a process called querying. Querying is done in a very structured form. It contains the attributes to return, the table from which to get them and any additional constraints. An example to retrieve all cities with a population bigger than 200,000 is shown in Query 2.1. Many more advanced functionalities exist, but they are outside of the scope of these preliminaries.

```
SELECT name
FROM cities
WHERE population > 200000;
```
Query 2.1: An example SQL query

Relational databases are named after the relational algebra, a construct in mathematics. These mathematical concepts are not associated to 'relations' as a concept of affiliation between people or items. So, although the name is confusing, relational databases in their core are not concerned with relations at all. On the other hand, for graph databases the focus is on the relationships. The connections, which are only coded implicitly in relational databases, are explicitly mapped out in separate tables in graph databases. Contrary to relational databases, relations are not represented by referring to a foreign key, but with pointers to the data records themselves. This structure of richly connected entities allows graph data to be queried from any point of interest (Neo4j, n.d.).

2.2.2 Database operations

Database management systems allow for various operations. Through a query language, the user is provided with multiple functions to retrieve data. Some of these are relatively straightforward, such as the retrieval in Query 2.1. Other operations need some explanation. An often used example of a more complex operation is the join. A join can be done both implicitly and explicitly.
CHAPTER 2. PRELIMINARIES

A join is a tool to combine data from multiple tables. This is done by means of primary and foreign keys. If an attribute is used in multiple tables, in this case the CID and RID, it can be used to combine the tables. In Query 2.2, an example query is shown that uses an implicit join twice. The join operation combines all three tables through the operations in the WHERE clause. First it combines the tables cities and connections, where it checks all CIDs in rows in the cities table and looks for all rows in the connections table that use the same CID. It then adds a row of these data to a new joined table in working memory. Once the algorithm has iterated over all rows, it continues to the next join. Then, for each RID in the newly joined table, it looks for rows in the roads table with the same RID and then adds a new row with this data to a result table. As mentioned before, a join can also be done explicitly within SQL, the same example as before but with an explicit join is shown in Query 2.3.

Related to these more complex database operations is the query containment problem. We say that query Q1 is contained by query Q2 if the result of Q1 is a subset of the result of Q2. The query containment problem asks a more general question: Given a certain Q1 and a certain Q2, does Q2 contain Q1 for any database D? This is a useful tool, as it can help prove query equivalence (Q1 \( \subseteq \) Q2 and Q2 \( \subseteq \) Q1 means Q1 = Q2). Query containment can also be used to approximate results of hard queries on large databases by running a similar but easier query, that in turn evaluates more quickly (Kolaitis, n.d.). However, answering the query containment problem becomes increasingly difficult when a language gains more functionality (higher expressiveness).

2.2.3 Graphs and Graph Data

Many disciplines of science have their own definition of a graph. The most common item coming to mind when talking about a graph is the two-dimensional figure, representing points of data on a plane. These graphs are also called line charts, and can be created for sets of points as well as for continuous functions.

Fairly different is the graph definition from discrete mathematics, where a graph depicts a set of objects that may be related. Both the objects and the relations can be displayed in an abstract way, where the objects are drawn as circles called nodes or vertices, and the relations are depicted as lines between them, also called edges.

On top of that, there is also a definition for a graph from the field of computer science. In that case, the graph is an abstract data type that is used to store the set of objects from the mathematical graph.

Graph data is named after the discrete mathematical field of graph theory and therefore mostly relates to the middle definition. The term graph data specifically refers to the data itself, the sets of objects and relations that together form a database. There is nothing inherently visual about graph data, although a visualization follows naturally through abstraction of objects.

To specify the difference between relational databases and graph databases, Table 2.2 shows an example of a property graph database. A typical graph database would not necessarily contain the node- and edge properties tables, but contain just the links themselves. This property graph data can also be easily transformed into a visual interpretation, called a node-link representation. A partial node-link representation of the graph database from Table 2.2 is shown in Figure 2.1. The property part is contained in the rectangles with properties. One item missing from the node-link representation is the property `type` for each edge, which is not present in the graph database.
representation that is present in the table, are the edge IDs. These are implicit and not available to users or in interfaces. Edge IDs are internal mechanics for the database system in case two nodes are connected by two different edges. For example, if we would add flight-routes to the database, there might be an additional edge from Beijing to Amsterdam. These edges are then distinguished through their IDs.
2.3 History of graph databases

To provide some context for how we arrived at the current state of the art in databases, in this section we describe the history of graph databases. We start from the foundation, old techniques and the introduction of the relational model. Via Object Oriented databases, the NoSQL movement and RDF we will end up at the modern developments of graph databases and Graph Query Languages.

2.3.1 Foundations

The concept of database systems arose in response to early methods of computerized management of commercial data. Old methods of data management often stored data in operating system file structures. These systems made use of specific application programs to manipulate files in different ways, for example to add a record (Silberschatz, Korth, Sudarshan & Others, 2011). Besides the fact that this is a complex task, there were many problems with storing data like this. Problems included data redundancy, integrity, concurrent access and security of the data.

A major purpose of a database system is to provide users with an abstract view of the data. This means that the system hides certain details of how the data are stored and maintained (Silberschatz, Korth, Sudarshan & Others, 2011). Programmers and database administrators access the logical level of abstraction, a database user sees the view level. The first step in the abstraction of database systems was the relational model defined by Codd (1970). He suggested that management of the data should be independent of changes in data types and data representation. A good database system uses a high level language, which should yield maximal independence between programs and their machine representation.

Although abstraction eased data manipulation, relational databases did not reach performance of the previously used network and hierarchical databases. This changed with the introduction of System R by IBM (Silberschatz, Korth, Sudarshan & Others, 2011). By the early 1980s, the performance of relational databases had become competitive with network and hierarchical database systems.
2.3.2 Object Oriented databases and XML

The 1980s saw much research on parallel and distributed databases, as well as initial work on object oriented databases (Silberschatz, Korth, Sudarshan & Others, 2011). This development was needed because of issues with the relational model. One of the first issues confronting relational databases was the rise of non-textual data, such as images, sound files and videos (Lake & Crowther, 2013). This problem was solved by introducing characteristics such as complex objects and extensibility of data (Atkinson et al., 1989), corresponding with the notion of objects in object oriented programming languages. Object oriented structures such as hierarchies, aggregation and pointers are introduced in database models under the name of object oriented databases. These characteristics allow object oriented databases to be the first database model with native support for graph data.

In the end, object oriented databases did not gain much traction, except in niche markets such as geographic and engineering sectors where there was a need for graphical data to be stored (Lake & Crowther, 2013).

The first half of the 2000s saw the emergence of XML as a new database technology. XML can be viewed as a type of object oriented database. Due to the nature of XML, which allows for creation and usage of any tag, it can support complex objects. Therefore, XML databases can represent any arbitrary data structure. On top of that, XML is tree-structured, which allows for relatively easy representation of graph data. However, despite the wide range of possible applications of XML, relational databases still formed the core of the majority of database applications for quite some time (Silberschatz, Korth, Sudarshan & Others, 2011).

2.3.3 NoSQL and GraphDBs

Another development away from relational databases was the NoSQL movement. NoSQL attempts to label the emergence of an increasing number of non-relational, distributed data stores. The problem of relational databases was the design for character-based data and the modelling by means of columns and rows in a table. This did not scale well, which is a problem in the era of big data (Lake & Crowther, 2013). NoSQL can often offer a quicker solution to a smaller group of applications, rather than the one-database-fits-all approach of the relational model.

Interest in NoSQL rose because of two main reasons. First of all, traditional query languages did not work quickly enough for some new technologies. Secondly, data is no longer only stored, data analysis and retrieval is also essential (Lake & Crowther, 2013). This requires a data model that could execute all of these operations efficiently.

There are various types of NoSQL databases. The subcategory that is most relevant to this overview is the category Graph Data Models. This was the only type of NoSQL database that still applied relations (Moniruzzaman & Hossain, 2013). It also has a focus on visual representations, such that the databases are easier to use.

The NoSQL movement and its goals sparked the development of the first dedicated graph databases. Graph databases allow for data to be stored with a direct link, retaining the interconnectivity in the data (Angles & Gutierrez, 2008). In many cases, structures can also be retrieved with a single operation. However, the underlying storage mechanism varies. Some graph databases depend on a relational engine, while others use a key-value store or a document oriented database.

2.3.4 RDF and SPARQL

In all the years that had passed since graph data were first stored in databases, no standardization efforts had been made. When W3C adopted RDF as a recommendation in 1999, its simplicity and ability to model abstract concepts led to increasing use in data management. The RDF triplets data model was created as a metadata model, but also intrinsically represents a labelled, multi-directed graph. Each triplet can represent an edge in a labelled graph. It is therefore well-suited as standard graph database. It can therefore be named as the first standardized graph database.
Since then, many query languages have been created for RDF. Some were based on SQL, others on logic and rule languages. Angles and Gutierrez (2005) tested a couple of these languages for expressive power regarding graph queries and found that most do not perform well. Only the languages G+ and GraphLog support graph queries well enough to perform as graph query languages.

At the time of the experiment of Angles and Gutierrez (2005), graph query language SPARQL was only a draft. After its introduction, it quickly became the most well-known and used query language for RDF. SPARQL allows for querying data that follows the RDF specification of the W3C. In January of 2008 SPARQL 1.0 became an official W3C Recommendation. SPARQL 1.1 followed in March of 2013.

The RDF data model and SPARQL query language are the last graph-supporting data structures before the development and adoption of dedicated Graph Query Languages.

2.3.5 Graph Query Languages

The first graph query language was G (Cruz, Mendelzon & Wood, 1987). G was not proposed as an alternative to relational query languages, but rather as a complementary language to simplify the formulation of recursive queries. The development of G started a movement of theoretical proposals of other graph query languages (Angles et al., 2016). Some early graph query languages are GRAM and GraphDB, which use regular expressions to define a graph pattern. Other languages, such as QBD, had a visual, diagram interface (Angelaccio, Catarci & Santucci, 1990).

During the 90s, development of GQLs was overshadowed by the appearance of XML, which was seen as the main alternative to relational databases for semi-structured data. However, the emergence of applications with cyclical relationships gave the GQL development new momentum (Angles et al., 2010).

Most recent GQL developments include the rise of multi-vendor languages such as Cypher and SPARQL. More information on recent visual graph query systems can be found in Section 3.4.
Chapter 3

An exploration of existing systems.

In the previous chapters, we have expanded upon the urgency of creating a usable database system, and explored the theory behind databases and various other preliminaries for this thesis. In this chapter, we will review the existing visual graph querying systems. This provides a framework of what our system should be able to do, and what is left to be improved upon.

Although this thesis focuses on improving the interaction with databases in general, this chapter focuses on graph querying systems in particular. As we are looking to develop a successful database interaction, we are searching for the most effective starting point. From Kerren et al. (2008) and Mitrovic (1998), we know that visual representations simplify the database interaction. Of the currently developed systems, graph data have the most intrinsic visual nature. There are many tools that allow for interaction with visual graph queries. Thus, this set of database systems is a good starting point to see which ‘simpler’ database tools exist.

This chapter will answer the subquestion:

What are some existing visual querying systems?

3.1 Introduction

With the exponential growth of data, the variety of different data types also increases rapidly. One of these types is graph data, which is named after the abstract data type graph. This data contains points or nodes and relations or edges which interconnect the data in multiple ways. Usually all of the data is connected and can be shown in the form of one big graph.

Graph data has been represented in different types of databases since the late 1960s (Angles & Gutierrez, 2008). The first steps were made in network model databases, later graph data was also saved in databases that allowed for labeling of nodes and edges. Some of these databases were built upon relational engines, storing data in a table format, whereas others created their systems from the ground up.

Graph data, by design, allows for rapid retrieval of hierarchical structures and links as there is no need for explicit JOIN operations. When graph data is stored in systems based on relational engines, this computational advantage is lost. More recent systems use NoSQL structures, allowing for more complex data as well as different types of queries (Angles & Gutierrez, 2008).

Nowadays, databases are used to store data about almost anything. Not only companies record data, consumers do so too. Both things (cars, smart thermostats and fridges) and smart assistants record lots of data. This may be interesting data for a consumer to analyze. However, over the years, one thing has remained stable: data has been stored in traditional databases. Querying these databases requires knowledge of query languages, most of which require extensive training.
to unlock full potential. A solution would be to use a visual query language, which allows for comfortable navigation even in case of complicated or ambiguous queries (Catarci, 2009).

Several tools have been created to interact visually with graph data. In these tools, one can often create queries via drag-and-drop methodologies to improve user-friendliness (Catarci, 2000). However, multiple issues arise when creating interfaces like these (Bhowmick, 2014). First of all, preselected or canned graph patterns (the elements for interaction) need to be selected by a domain expert for every database and are not portable. Secondly, an interface is static, so parts may become obsolete as the data may not contain these predefined patterns any more. Additionally, the interface must be user-friendly and allow for quick query evaluation.

Solving these problems is important because the recording of data becomes more widespread through the field of ubiquitous computing (Want, 2016). In addition, the market share of graph databases is growing (Agresta, 2015) and every year more tools for graph data are created. This re-establishes the need for user-friendly graph query systems.

In this chapter, we aim to identify research fields related to visualizing data and creating efficient user interfaces. This will help us analyze what the components of an optimal visual query system are. Then, we analyze recent visual query systems and their properties. We conclude the chapter by discussing the state of the art and required improvements upon the existing systems.

### 3.2 What is known about visualizing graph data?

In this section, we will describe existing research on the visualization of graph data. This information helps to structure the evaluation of visual query systems as done in Section 3.4.

The ultimate aim of visualization is to provide insight to users. Visualization research has put much effort into developing new methods to help users obtain insight into their data (Van Wijk, 2006).

#### 3.2.1 Visualization design

The first step in the process of creating a visualization is the design. In order to create the most optimal visualization of given data, design decisions need to be made. Munzner (2009) wrote about four levels of decision making:

1. **Characterize the tasks and data in the vocabulary of the problem domain.** The result should be a detailed set queries or actions carried out by the target users for the collection of data. For graph data, this is generally a concise set of questions that users want answered, such as the option to implement basic graph patterns. More information on these can be found in Section 3.3.2.

2. **Abstract into operations and data types.** The result should be a description of operations and data types. This is the input required for making visual encoding decisions at the next level. The other aspect of this stage is to transform the raw data into the data types that visualization techniques can address. For graph data, there is only one data type, making this a small step in the process.

3. **Design visual encoding and interaction techniques.** The choice of interaction techniques depends on the goal of the system.

4. **Create algorithms to execute these techniques efficiently.** The discussion of different algorithms is beyond the scope of this survey.

If these steps are used during the visualization process, designers can ensure a robust end result. In addition, this model can also be used to guide the evaluation and validation of visualizations. All of these steps can uncover threats to validity (Munzner, 2009). A mistake in level one can mean that your visualization allows for actions that users would not do, a mistake in level four can make the visualization process too slow.

Another type of evaluation is too see whether the visualization could be improved. There are three types of evaluation:

1. Formatte: “Can I make it better?”
2. Summative: “Is it right?”
3. Exploratory: “Can I understand more?”

These measures are partially related to cognitive processes and the human mind. For more detail on how visualization can be improved through User-Centered Design, see Section 3.2.4.

However, even with the guidelines mentioned above, it is still hard to create the perfect design. For example, there can be trade-offs in the cooperation of visualization experts and domain experts (Van Wijk, 2006). In addition, the data itself may have properties that make it hard to visualize, such as its scale and dimensions.

3.2.2 The issue of scale

Visualizing graph data quickly becomes complex when graph size increases (Chau, 2012). The number of nodes and edges in the graph may quickly exceed then number of pixels on the screen. Subsequently, finding a good starting point to investigate the data becomes a difficult task.

Chau (2012) suggests that attention routing can help users locate good starting points for analysis. As abnormalities in the data often represent new knowledge, channeling attention based on anomaly detection may directly lead to insights.

Another solution was proposed by Dunne and Shneiderman (2013). Their paper describes how visualization can be improved by means of graph simplification. The process uses motif extraction or simplification, which means that common patterns of nodes and links are replaced by meaningful glyphs, such as in Figure 3.1.

![Figure 3.1: Graph simplification by means of motif extraction. Key features remained visible.](image)

After applying this process, the key structure of the graph is still visible. However, the image is much less cluttered and therefore easier to understand. The graph also requires less screen space while it preserves a lot of underlying information (Dunne & Shneiderman, 2013).

3.2.3 Representations and layouts

Another issue that the design guidelines of Munzner (2009) do not deal with is the fact that many different data representations exist. Below, we will discuss all representations and layouts that are regularly used for the visualization of graph data.

The representation that is used most often for graph data is a node-link diagram, also called an explicit representation. In a node-link diagram, the nodes are represented either by 2D geometric shapes such as circles or squares, or by textual labels. Edges are most often represented by line segments or arrows.

To cope with the messiness of node-link diagrams, Di Battista, Eades, Tamassia and Tollis (1999) suggests these essentials for good node-link layout:

- Minimized edge crossings
- Minimized distance of neighboring nodes
Figure 3.2: A conflict between the rules of good node-link layout. In this case, minimization of edge crossings conflicts with uniform edge length.

- Minimized drawing area
- Uniform edge length
- Minimized edge bends
- Maximized angular distance between different edges
- Aspect ratio of height:width about 1 (not too long and not too wide)
- Symmetry: similar graph structures should look similar

However, these guidelines can be conflicting, such as minimized edge crossings and uniform edge length. This example is shown by Schulz and Schumann (2006) and can be seen in Figure 3.2.

For node-link representations, multiple different graph layouts exist. First, of all, a hierarchical graph layout draws all vertices of a graph in horizontal layers. This is most often used for directed graphs, where the edges are directed downwards. A force-directed layout focuses on aesthetics, such that the edges are all of more or less equal length, as well as minimizing edge crossings. A constraint-based layout captures restrictions that are added to node and edge positioning. Finally, an attribute-driven layout uses extra attributes from the data to add dimensions to the positioning of the data. In addition, hybrid layouts, which combine the previous approaches, are possible.

As opposed to explicit representations, implicit representations exist, where edges are not represented as explicit objects. As with explicit representations, nodes are represented as graphics primitives. Edges can be represented by means of inclusion, overlap or adjacency of the nodes. Examples are treemaps and sunbursts (Lex, 2015).

A more compact and less elaborate representation can be achieved by means of matrices. Any graph can also be represented by an adjacency matrix. The focus of the adjacency matrix is on the set of edges, instead of on nodes. This means it can show cliques and clusters well. However, matrix representations are not suited for sparse graphs as they have a quadratic screen space requirement (Lex, 2015). For more discussion about matrix representations versus node-link diagrams, see (Munzner, 2014a).

In addition, there are hybrid representations, such as smaller matrices for cliques connected by links. One example of such a tool is NodeTrix (Henry, Fekete & McGuffin, 2007). In Figure 3.3, a hybrid representation is shown.

3.2.4 Information visualization and HCI

In previous sections, we have stated multiple times that visualizations of queries may help users understanding of the data. However, we have not yet discussed the mechanics behind this phenomenon. There are multiple ways in which information visualizations and well-designed interfaces may support understanding and we discuss them in detail below.

Cognitive support

According to Card, Mackinlay and Shneiderman (1999), there are six major ways in which visualizations can support cognition.
CHAPTER 3. AN EXPLORATION OF EXISTING SYSTEMS.

1. Increased resources. A visualization can serve as external memory, or to offload work from the cognitive to the perceptual system.
2. Reduced search. Visualizations can group information in a way that imposes structure, allowing for easy search and access.
4. Perceptual inference. Visualization can support perceptual inferences that are easy for humans (and hard for computers).
5. Perceptual monitoring. Visualizations can allow for the monitoring of a large number of potential events if these can stand out by appearance or motion.

From this list, we can conclude that the main benefit of data visualization is that it lowers the amount of brainpower required for cognitive processes, through acting as a frame of reference or temporal storage (Kerren et al., 2008). Therefore, visualization is a valuable type of assistance in data analysis. However, peoples interactions with a visualization tool are also influenced by personal factors.

Personal factors are cognitive processes that differ for each individual. These factors include the person's frame of reference, conceptual models and his mental model. Perception and interaction with data will affect user's understanding of information presented visually (Tory & Möller, 2004).

A mental model is a set of expectancies, held by a user, regarding the actions that are required to accomplish a goal. In our case, the goal may be to find the result of a certain query. An effective mental model allows the user to correctly predict the result of various actions within the system (Wickens, Lee, Liu & Gordon-Becker, 2014). The development of this mental model can be supported by the designers through a well-defined conceptual model. The conceptual model depicts the framework through which functionalities are presented (Mayhew, 1992). One of the ways in which this can be done is presenting functionality through a familiar metaphor. Examples of such a metaphor are the cut and paste mechanisms in text editors. Another important way to clarify the conceptual model is through providing feedback, especially throughout processing actions such as the loading of data.

Perceptual support

In addition to the ways presented above in which cognition is supported by visualization, perception can also be aided by visualization (Kerren et al., 2008). There are two properties of the visual system that can be aided by proper visualizations.

The first property is denoted as pre-attentive processing theory. This theory discusses visual features that can be distinguished rapidly and accurately by the low-level visual system. One of these features is color. An example of a pre-attentive processing task is shown in Figure 3.4. The pre-attentive processing does have limits: if you would include many different colors, it would no longer work and we would have to resolve to sequential scanning.

![Figure 3.4: Finding the red dot between all blue dots is fast due to pre-attentive processing.](image-url)
The second property is Gestalt theory, which is a high-level cognitive process for perception. It explains the principles that the brain follows when trying to interpret an image (Chang, Dooley & Tuovinen, 2002). There are eleven laws that were identified by Chang et al. (2002) as having significant effects on interface design:

- **Balance/Symmetry.** An object seems incomplete if the object is not balanced or symmetrical.
- **Continuation.** The eye has an instinctive action to follow continuous figures, such as paths, through an image.
- **Closure.** A closed contour tends to be seen as an object.
- **Figure-ground.** We distinguish between a foreground and background.
- **Focal point.** Every visualization requires a centre of interest or point of emphasis.
- **Isomorphic correspondence.** We interpret meaning of symbols based on previous experiences.
- **Good form.** This means a simple design or symmetrical layout.
- **Proximity.** Things that are close together are perceptually grouped together.
- **Similarity.** Similar elements are perceptually grouped together.
- **Simplicity.** Uncluttered graphics can easily be simplified in the brain.
- **Unity/Harmony.** A congruity or arrangement exists among the elements in a design.

If visualizations follow these guidelines, processing of the data becomes both easier and faster.

**Interface design**

To conclude, visualization can support both cognition and perception, thereby helping users understand the data. The best way to use the guidelines above is to apply User-Centered Design. A user-centered perspective tries to optimize the system to make sure that it works well in combination with the users capabilities, instead of forcing the user to adapt their work style to the systems capabilities (Freitas, Pimenta & Scapin, 2014). This means that we do not only look at the representation of the data, but accommodate user’s preferences for the interface as well.

(Smith & Mosier, 1986) present five high-level goals for data display:

1. Consistency of data display
2. Efficient information assimilation by the user
3. Minimal memory load on user
4. Compatibility of data display with data entry
5. Flexibility for user control of data display

However, data display design is a large topic with many special cases. We should see these objectives as a starting point. Applying them to information visualizations can yield rules such as “Present data only if they assist the user” or “Use consistent formatting in the interface” (Shneiderman & Plaisant, 2010a).

If we move beyond the modeling of the display and look at the workings of the full interface, we find that interface design is guided by the “Golden Rules” as presented by Shneiderman and Plaisant (2010a). In his work, eight golden rules are mentioned which will be discussed below.

The first rule is to strive for **consistency.** This is equal in wording to the first rule by Smith and Mosier (1986) but has a different meaning. Consistent interfaces do not only use consistent color and layout principles, but also require similar actions in similar situations. In addition, terminology should be consistent throughout menus, prompts and help screens.

The second rule is to cater to universal **usability.** Users of a system are usually diverse in characteristics. Differences between novices and experts, users of different age ranges and disabilities such as color blindness require plasticity within the interface.

**Informative feedback** is very important as well. For every user action, there should be system feedback. Feedback for frequent and minor actions should be minimal, whereas for important and major actions, it should be more substantial.

Related to the aspect of feedback is the concept of **closure.** Designing actions with a beginning, middle and an end can give users a sense of accomplishment and satisfaction.

Another rule that can increase user satisfaction is an interface that tries to **prevent errors.** A good example for the type of interface we evaluate here is the graying out of inappropriate menu options.
items. Additionally, in case of an error, simple and specific instructions for recovery should be provided.

The sixth golden rule states that actions should be \textit{reversible}. This feature is important for user satisfaction as it decreases anxiety and fear of errors. It may also encourage the user to explore unfamiliar options, as mistakes can be reversed quickly.

Support of the internal locus of \textit{control} is important as well. Experienced users desire the sense that they are in charge of the interface. Surprises or changes in familiar behavior will annoy them.

The last golden rule is to reduce \textit{short-term memory load}. Humans in general can store seven plus or minus two chunks of information in short-term memory. This means that interfaces should avoid situations in which users must remember information from one screen and then use that information on another screen.

Shneiderman and Plaisant (2010a) state that these principles, although refined over three decades, require interpretation and tuning for each specific design domain. Their statement that these rules are a good starting point for designers, seems to resonate with the research community through collections of these or similar rules (Abras, Maloney-Krichmar & Preece, 2004; Chalmers, 2003; Dix, 2009).

### 3.3 Demarcation of the topic

In this section some more definitions are introduced to indicate the boundaries of this survey. The concepts defined below will be used in Section 3.4 to discuss the state of the art in user-friendly graph querying.

#### 3.3.1 Visual query systems

Different papers have written about the topic of visual query systems under different names. Some papers use the term \textit{Visual Query Language} (VQL) such as (Catarci, Costabile, Levialdi & Batini, 1997). Others use the word \textit{tool} or \textit{interface} to describe their implementation (Truong & Dkaki, 2006; Braga, Campi & Ceri, 2005). To generalize this field, we will use the title of \textit{Visual Query Systems} (VQS) or \textit{Visual Graph Query Systems} (VGQS). Visual Query Systems are query systems that use visual representations to denote the domain of interest and to express requests. According to Catarci et al. (1997), they allow for querying by non-technical users and introduce a mechanism for comfortable navigation in all situations.

As a VQS is complete and includes a visual query language that it abstracts from, we start by looking at the work by (Catarci, 2009) on visual query languages. She writes: “Visual Query Languages are languages for querying databases that use a visual representation to depict the domain of interest and express related requests. [They] provide a language to express the queries in visual format [...]”.

\textbf{Visual query representations}

Catarci (2009) notes that, in contrast to query representations, visual representations can differ from a database representation. In Section 3.2.3, we showed that graph data can be represented explicitly, implicitly, by means of matrices and in hybrid form. Visual query representations often are in one or multiple of three forms: Form-based, Diagrammatic and Iconic representations. Below, we will examine whether these representations will also work for visual \textit{graph} query languages.

Form-based representations are most common for queries, it is a simple way to allow novice users to formulate queries. This is because users do not need any insight in the structure of the data, but can use keyword search to find what they are looking for. According to (Catarci et al., 1997), the main characteristic of a form-based representation is that it is a structured representation which corresponds to an abstraction of conventional paper forms. This may include form elements such as radio buttons, check boxes and drop-down menus (El-Mahgary & Soisalon-Soininen, 2015). A form-based approach is attractive because it does not require much attention.
to detail from the user’s perspective. It may provide the user with a high level of expressive power, while also being easy to use (El-Mahgary & Soisalon-Soininen, 2015).

Diagrammatic representations use diagrams to represent a query. A diagram is characterized by a parallel correspondence between the structure of the representation and the structure of the represented (Kulpa, 1994). For example, relative positions and distances are in direct correspondence between the two. Analogical representations, of which diagrams are a subset, model the thing they represent. Richards (2002) argues that there are many types of diagrams, that should be classified through their use, namely to display relations. He suggests: “What makes a diagram a diagram is the ability of users to recognise in it spatial relations which in some way correspond to the relationships represented.” (Richards, 2002). If we combine the viewpoints of both writers, we could conclude that diagrammatic representations of graph queries are in the form of node-link diagrams.

Iconic representations use icons, pictorial representations of an object or concept. By the usage of icons, information is represented through form, structure and color more than through numbers and text (Catarci, Massari & Santucci, 1991). One example is the the icon of a knife and fork representing the presence of a restaurant. Icons have a high expressive power and cultural independence from the viewer, making it very advantageous to use. However, it can be hard to find a universally accepted set of icons (Catarci et al., 1991). An iconic query is created by selecting and combining icons to produce a new icon (Catarci et al., 1997).

Visualizing results

The adequate visualization of query results is important for usability of the system as well. In addition to the visual query representation classification, Catarci et al. (1997) also present visual representations of query results. They categorize them under form-based, diagram-based and icon-based again, but the meanings differ slightly from those for query representations. Therefore, we will discuss them below.

A form-based result representation means that the outcomes of the query are presented in tables or scrollable lists. This is a very compact form, especially in case of removal of duplicates.

A diagram-based representation refers to any graphic that uses position and magnitude to encode information. For graph data, some VQSs may show the query result in the same diagram type as the original query.

An icon-based representation uses icons to visualize the query result. The icon could show the type of data, with textual labels identifying each occurrence. This way, it is easy to see which kind of data was retrieved, making it easier to analyze.

In order to create the most effective system, it would be valuable to visualize results in the same way as the query was represented. This especially holds for icon- and diagram-based representations. If the interface is well-designed, querying could become an iterative and possibly exploratory action, where each result prompts the user for new questions or refinement of the query.

Each of the above mentioned representations works well for certain situations and has both pros and cons. (Catarci et al., 1997) suggest that an effective system should adapt the data representation to the task at hand, in order to support understanding of the data. For example, including both a natural language option and a visual option allows a system to be used effectively by both novice and expert users.

Orientation

However, well-designed query interfaces are not the only important factor. Catarci (2009): “Apart from the visual representation, any VQL is characterized by the way in which it allows the user to express his/her requests. Very often, the actual query specification is the second step of the user interaction, while there is a first step devoted to the understanding of the overall database content.” Before one is able to create a query, one needs to know the structure of the database. Understanding what data the database represents, as well as knowing the names of objects of
interest, is required to efficiently create a query. Data exploration is the first step a user takes, and it should therefore be easy to do.

In this survey, we will not include visual query systems that have been created for the purpose of browsing only. Our goal in this survey is to show the state of the art in visual graph query systems. However, we believe that browsing is not typically done by means of a query. There are many visual graph data exploration tools available, some include NetworkX (Schult & Swart, 2008), NodeTrix (Henry et al., 2007) and Refinery (Kairam, Riche, Drucker, Fernandez & Heer, 2015).

3.3.2 Graph query languages

Next, we move on beyond visualization to see what constitutes a graph query. The two main conceptual features of modern graph query languages are graph patterns and path expressions. The focus of most graph query languages has been on matching these types of queries against a data set.

Graph patterns

A graph pattern has the same structure as the type of graph database it has to match. However, instead of constants, the pattern contains variables as nodes or edge labels. A match for this pattern is a mapping from variables in the pattern to constants in the database, such that the pattern is contained within the original database. An example is shown in Figure 3.5.

![Figure 3.5: A basic graph pattern for an edge-labelled graph (Angles et al., 2016)](image)

These basic graph patterns can already cover the relational operations of natural join and selection based on equality. More complex patterns are needed to execute other relational operations such as projection, union, difference and filter (Angles et al., 2016).

Path expressions

Path expressions or navigational queries allow for navigation of the topology of data, which means that values are filled in for each node. (Angles et al., 2016). The most basic type of navigational query is the path query, to find out whether there is a path between two named nodes. This can be done with or without regard for edge-labels. One example of a path query with regard of edge labels to find out whether two named persons are friends.

Path queries can also be used to augment graph patterns, which creates navigational graph patterns, as seen in Figure 3.6. They are similar to basic graph patterns, but edges labels can be either constants, path queries or $*$ to denote arbitrary paths.

Required functionalities

According to Wood (2012), there are a number of functionalities that a graph query language must fulfill in order to gain full expressiveness. Two of these were mentioned in the previous sections. Subgraph matching is a graph pattern and Finding nodes connected by paths is a path expression or navigational query. Other functionalities named by Wood (2012) are:
3.4 State of the art in VGQS

Many of the languages that were discussed in the introductory sections, such as SPARQL and Cypher, are query languages with text-only interfaces. As mentioned previously, using these languages requires training. Unlocking full potential of these languages can take multiple months or years of practice. In order to allow novice users and non-programmers to do data analysis as well, this section of the report will discuss recent visual graph query systems.

In the sections below we will discuss all the complete visual graph query systems that we have found during our investigation. In Section 3.4.1, the categories on which we evaluated them are explained. The table shows all collected data, and is followed by summaries of all VGQS and their notable qualities. We will conclude this section with a small evaluation section, in which we present what we have learnt.

3.4.1 Taxonomy

First of all, it is important to note that all systems mentioned below use explicit data representations for their graph data. In the third column, you can see each systems query representation. As mentioned in Section 3.2, this can be either diagram-based, form-based or icon-based. The same holds for the representation of the query result, which is evaluated in the fourth column. In the fifth column, we have evaluated the query systems expressiveness through their implementation of basic graph functionalities. As can be seen in Section 3.3.2, these include subgraph matching, path expression or navigational query, aggregation, node creation and ranking. We have excluded Woods requirement for approximate matching. This is not something we consider essential for full expressive power.

As you may see in the table, some entries contain a question mark. For these cases there was no explicit confirmation or negation made about the functionality, or we were not able to check for the functionality.

In the case of multiple representation types supported by a system, or if the system uses hybrid representations, the most prominent representation is named first.

20 Towards successful interaction between Humans and Databases
<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>Query repres.</th>
<th>Result repres.</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Subgraph</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Matching</td>
</tr>
<tr>
<td>Cytoscape</td>
<td>(Shannon et al., 2003; Smoot et al., 2011)</td>
<td>form</td>
<td>diagram</td>
<td>✓</td>
</tr>
<tr>
<td>GBLENDER/</td>
<td>(Jin et al., 2011, 2012)</td>
<td>diagram</td>
<td>diagram</td>
<td>✓</td>
</tr>
<tr>
<td>PRAGUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphVista</td>
<td>(Paradies et al., 2015)</td>
<td>form/ icon</td>
<td>diagram</td>
<td>✓</td>
</tr>
<tr>
<td>NITELIGHT</td>
<td>(Russell et al., 2008)</td>
<td>diagram</td>
<td>form</td>
<td>✓</td>
</tr>
<tr>
<td>OptiqueVQS</td>
<td>(Soylu et al., 2016)</td>
<td>form/ diagram</td>
<td>form</td>
<td>✓</td>
</tr>
<tr>
<td>Orion</td>
<td>(Jayaram et al., 2016)</td>
<td>diagram/form</td>
<td>diagram</td>
<td>✓</td>
</tr>
<tr>
<td>PopoJS</td>
<td><a href="http://www.popotojs.com">www.popotojs.com</a></td>
<td>diagram/icon</td>
<td>form/icon</td>
<td>✓</td>
</tr>
<tr>
<td>QueryVOWL</td>
<td>(Haag et al., 2015)</td>
<td>diagram/form</td>
<td>form</td>
<td>✓</td>
</tr>
<tr>
<td>Visage</td>
<td>(Pienta et al., 2016)</td>
<td>diagram/icon</td>
<td>diagram/form</td>
<td>✓</td>
</tr>
<tr>
<td>VOGUE</td>
<td>(Bhowmick et al., 2013)</td>
<td>diagram</td>
<td>diagram</td>
<td>✓</td>
</tr>
</tbody>
</table>

1By means of the package “General SPARQL”  
2By means of the package “Metanode2”  
3Supported in vSPARQL but not implemented in the interface  
4Mentioned as second-level goal in (Soylu et al., 2016)  
5Implemented in Cypher, but not supported in the interface.
3.4.2 Summaries

In this section, we will present the different interesting or outstanding aspects of the systems discussed in the table above.

Cytoscape is an open source platform for visualizing networks and integrating them with other data (Shannon et al., 2003; Smoot et al., 2011). It was originally meant for the analysis of biological data, but now also allows for general network analysis and visualization. The core distribution provides a basic set of features, but many plugins are available to extend this. Apps have been developed to couple Cytoscape with Neo4j, RDF or SPARQL. As can be seen in the table, these plugins can allow for full graph functionality. The General SPARQL app allows for searching in a form-based menu interface. The result is interactive and can be used for follow-up searches. The Metanode plugin supports aggregation within queries. Cytoscape is based on Java technology.

GBLENDER/PRAGUE are two subsequent systems that support evaluation of subgraph containment and similarity queries, as well as visual query modification (Jin et al., 2011, 2012). The systems facilitate pruning and retrieval of partial results, to allow for more efficient querying.

GraphVista is a visual query and exploration tool (Paradies et al., 2015). It can store intermediate query results in a repository for later use. In addition to caching results, it allows the user to analyze large graphs without overloading the application through loading large fractions of the data set. It also uses ad-hoc querying to identify entry points of interest for the user. Additionally, GraphVista also implements partial approximate matching: when a user applies a filter, an estimated result size is displayed. GraphVista was designed with the goal in mind that it should be intuitive to use without requiring expert knowledge of a graph query language.

NITELIGHT is a visual tool for SPARQL that supports end users by providing graphical notations to query the data (Russell et al., 2008). It uses a visual version of SPARQL called vSPARQL. NITELIGHT provides an interactive graphical editing environment that combines navigation and query visualization techniques. Although NITELIGHT only uses tables as query result representation, they acknowledge that other output formats might be more suited to the processing capabilities of human end-users. Examples they propose are map-based, timelines and natural language serializations (Russell et al., 2008). Although NITELIGHT is based on vSPARQL and therefore supports full graph expressiveness, filtering expressions is not supported.

The usage of vSPARQL has as drawback that it still uses SPARQL terminology and notation, which might make it hard to understand for novice users. However, this seems to be the goal of the developers. They say: “[NITELIGHT] is primarily intended for use by those with previous experience of SPARQL. [...] we do not propose to develop a simplified query language for end-users; rather we aim to support end-users with respect to the creation of complex queries using supportive user interfaces and user interaction mechanisms.” (Smart et al., 2008)

OptiqueVQS is a visual tool that uses widgets to enhance working with SPARQL (Soylu et al., 2013). The user interactions are mostly done through a form-based interface. OptiqueVQS only supports graph pattern queries, as that was their primary goal. They state: “There are also query types for which there is no simple way to formulate them purely with the aforementioned topological forms, such as queries with quantification (i.e., universal and existential), negation, and aggregation (e.g., count, sum, max etc.).” (Soylu et al., 2013).

In OptiqueVQS, the arcs that connect nodes do not have any direction, since for each active node only outgoing relationships, including inverse relationships, are presented; this allows queries to be always read from left to right (Soylu et al., 2013). They employ a node duplication approach for cyclic queries for the sake of having tree-shaped representations for queries, as these may be easier for novice users to comprehend.

Orion is a visual query builder that uses suggestions to allow users to reach a higher success rate in creating complex queries (Jayaram et al., 2016). It supports both an active and passive mode, where the active mode is included in the diagram representation of the query. The passive mode can be used through a menu. This menu uses hierarchy to support the user in finding the right node or edge label.

popotojs (www.popotojs.com) is a JavaScript tool that builds on Cypher and the Neo4j REST
API to create a graph query builder. Graph queries can be created through clicking objects. The query builder disables automatically when no results are available, so it implements a reactive interface. It also implements functionality to translate the graph query to natural language. Results are displayed as a list, in order to show additional data. If geographic data is available, results are displayed on a map.

QueryVOWL covers a part of the SPARQL query language, but, to date, also omits some elements (Haag et al., 2015). The addition of envisioned graphical elements is on the researchers agenda. So far, a visual representation for advanced types of relationships, such as inequality or asymmetric relationships (greater than, less than, etc.), has not yet been defined. This means they only support graph pattern queries.

QueryVOWL uses a reactive interface and allows for interactive use of results to change variables and edit values. In order to implement a reactive interface, quick result retrieval is necessary. To support this, each query is normalized before sending it to the server. Normalized SPARQL queries are then stored in the cache.

Visage is a tool that allows users to create graph queries through a diagram-based interface (Pienta et al., 2016). Visage supports a wide variety of node constraints, from wildcard (meaning a node can be of any type) to single specified values. Visage has a feature they call ‘graph autocomplete’, which guides the query specification process. It guides both in structure and in attributes, such that the user will not draw a query with empty results.

This guiding feature requires Visage to be interactive: matches are found in the background while the user interacts with the query. Results can also be requested through a button. In order to parse these results more quickly, the query is divided in smaller pieces. Nodes which have a dedicated name and selected features are processed first, as that set of results is small. Wildcard nodes are parsed later, to lower the amount of processing required.

Visage fully supports Neo4j and partially supports SPARQL and is build in JavaScript and Python.

VOGUE is a tool for interaction-aware querying that blends query formulation and query processing together for quicker response times (Bhowmick et al., 2013). It builds upon previous work in PRAQUE and GBLENDER. VOGUE allows for visual query formulation though canned patterns. These patterns are mined in the data, to find the most frequently occurring fragments. It mostly addresses subgraph containment and similarity search programs.

In practice, the above mentioned systems are not used that often yet. One of the more popular languages that are currently used by experts for graph querying is Cypher, in conjunction with the Neo4j database. The visual interface is nice and uses diagram-based visualization of the graphs, but the query language is too complex for novice users for it to be included in this survey. However, it is a well-used system and it should be credited here.

3.4.3 Findings

We will now summarize and generalize our findings. Unfortunately, only a few of the papers provided adequate information regarding the types of queries that were possible in the systems. As software or demos were often not available, we hope that our conclusions in the table and below are correct.

First of all, we found that few of the systems reviewed above contain database interaction techniques beyond querying for answers. It was not possible to create new data, as typical querying systems allow. The two most important functionalities of subgraph matching and path expressions are supported, so the systems can be classified as visual graph query systems. However, the lack of interaction with the data hinders full usability of databases for novice users.

Related to the previous point is the fact that many systems build upon a more well-established language such as Cypher and SPARQL. These languages do support all graph functionalities, or full expressiveness, but that has not been used in these tools. As was mentioned in Haag et al. (2015), this is hard to implement, because some of these functionalities, especially aggregation, does not allow for easy visualization. Also, the result of a query with the COUNT operation would often result in just a numerical result, which makes iterative query and result processes harder. It
therefore makes sense that none of the systems reached full expressiveness, although it would be nice to see what solutions the developers could come up with.

We also found that many of the tools use hybrid representations for queries and results. This is a positive development, as it allows users to choose the interpretation that works best for them.

Unfortunately, only four out of the ten evaluated systems use explicit diagram-based representations for both the query and the result. This is a shame, because using the same, explicit representation, allows users to query iteratively. Forming a query, seeing the result and then being able to adapt the query slightly, instead of having to rewrite it, allows for quicker analysis. This may give users a more satisfying experience.

3.5 Conclusion

In this chapter, we have surveyed the state of the art in visual graph query systems. We have analyzed their visual characteristics and explored what those mean for usability, as well as measuring expressiveness through typical graph query tasks.

Development of user-friendly systems has come a long way since the first database developments in the 1960s. However, many of the tools surveyed in this paper are still in their infancy or only prototypes, with quite some improvements possible. The guidelines for interfaces showed us that there are still steps that can be taken, even in mature tools.

We did not undertake in-depth analysis of usability and performance of the systems. Usability analysis usually includes efficiency, effectiveness and satisfaction tests by means of user experiments. We also did not evaluate the computational performance of the systems here, but focused on the design and capabilities of the system. For our goal of the development of user-friendly systems, appropriate for novices, running time is not the most essential aspect to improve upon as most novices don’t work very fast.

We uncovered multiple challenges that the systems evaluated here do not deal with adequately. The first is the support of full graph functionality. Realization of support for aggregation, node creation and ranking does require more than implementation. The main challenge is coming up with appropriate visualization methods for these graph operations. This lack of full graph functionality in the systems hinders the adoption of VGQS as replacements for traditional database systems. As a result, it hinders usability of databases for novice users.

Another challenge is development of closed systems, where query and result use the same representation. These systems, especially those using diagram-based representation, allow users to query iteratively. This increases usability because it allows for quick adaptation and appropriate feedback mechanisms.

To conclude, this overview of ways in which visualizations support cognition and perception gives us concrete evidence why visual graph queries support database understanding. We have also seen that most of the recently developed tools can be improved significantly. This again confirms the need for a well-designed, novice-friendly database system. In the next chapters, we will be undertaking steps towards building such a system.
Chapter 4

An analysis of factors in Human-Database Interaction

In the previous chapter, we found that although many graph query system prototypes exist, they all leave things to be desired. The chapter serves as a starting point for designing a tool that facilitates novice users. In this chapter, we will lay the theoretical groundwork for the design. On top of that, we answer the following subquestion.

Which human factors play a role in the interaction with databases?

4.1 Introduction

In this chapter, we will focus on the underexposed area of human-database interaction. There are many processes that are part of this interaction. For novice users, the first step in interacting with a database is to study the query language. Other interaction processes include studying the data in the database, writing accurate queries and analyzing the data that is returned by the database system. There are many facets to these interactions with the interface that influence human performance.

As not much research has been done into human-database interaction specifically, the goal of this review is to research the factors that play a role in human interaction with information systems (a more generalized field). Along the way, we will analyze which factors can be applied to databases specifically. These outcomes may then be used to combine the efforts towards efficient querying and good interface design, by creating guidelines for implementation of a visual querying tool.

In this chapter, we will look at all the factors that may influence querying efficiency. We will focus on how these factors affect efficiency, effectiveness, and satisfaction, as well as performance. In our search for the complexities involved in querying a database, one of the most important involved fields is human factors engineering. The way the human mind works and interprets its environment, the culture of the individual, as well as cognition as a general topic are important subjects in human factors engineering. The goal of human factors is to enhance performance, increase safety and increase user satisfaction (Wickens, Lee, Liu & Gordon-Becker, 2004e), which are necessary in order to create an objectively good system. However, we should not underestimate the influence that design and technology can have on understanding the task at hand, so we will analyze those as well.

All four of the dimensions we discuss here (technical, task, human and individual factors) have been explored in one way or another. For each of these dimensions, there are factors that play a big role in the interaction between human and a database. We will discuss each of these dimensions in their own chapter. First, we discuss aspects of technology such as language and design. Then, we
look at task characteristics that influence performance. Subsequently, we take a look at properties of the human mind such as perception and decision making, and their roles in interaction with databases. We finish by looking at the properties of the individual, where characteristics such as confidence may play a role in human-database interaction. However, to place the topic in context, we will start this review by looking into previous works on human-database interaction.

4.2 Support in literature

During the writing of this chapter, we noticed that very few researchers have explored human factors in query languages. Previous studies were done a long time ago (Reisner, 1981; Ehrenreich, 1981; Laughery Jr. & Laughery Sr., 1985), and should be updated with more recent findings from both computer science and human factors research. However, recent papers on these topics are scarce. The broader topic of programming and its relations to human factors research have been explored in more depth. We believe that the process of writing queries is a more specific form of programming. This means that studies about programming also apply to our overview of querying research. However, to the best of our knowledge, no studies have been undertaken that directly compare the mental processes and models involved in querying versus programming. In the following paragraphs, we provide sources that indirectly support our standpoint.

First of all, we can infer a relation between programming languages and query languages from their syntax and through source-code from different programming language libraries. For example, the Python language has a library that allows for the creation and querying of a SQL database (MapBox, 2018), and Neo4j can be used through Java (Neo4j, 2018). The query languages themselves may also tick many boxes. For example, SQL has commands for control flow, control blocks, and Boolean operators, as well as the usage of variables. All of these are elements of a programming language too.

Definitions in literature also equate querying and programming. Some authors are very straightforward in their comparison, such as Risch (2009): “A query language is a specialized programming language for searching and changing the contents of a database.” Other comparisons can be drawn more indirectly. Zelkowitz (2007) defines a programming language as “the notations used to communicate algorithms to a computer […] The result of expressing the algorithm in a programming language is called a program. The process of writing the algorithm using a programming language is called programming […]”. This can also capture the gist of query languages and the process of writing a query. Zelkowitz (2007) also states that a program contains three basic components: (1) a mechanism for declaring data objects to contain information, (2) data operations that transform data objects into another and (3) an execution sequence. Although this does not fit one-to-one for query languages, the gist is similar. If we look at the result of a query as a variable, it is empty to begin with (1) and filled through the process of querying (2). Depending on whether the language is declarative or imperative, the execution sequence is defined to a certain degree (3).

Our final exhibit is in the discussion of (programming) language generations, in particular the third and fourth. Each new language generation aims for a higher level of abstraction. This can be achieved in two separate ways. First of all, the language can grow to be more like human (natural) language. The second option is to abstract away from code as a language and create programs through other interfaces such as tables or menus (Bohner & Mohan, 2009). The first-generation languages contain only binary code, second-generation languages are assembly languages. Both of these are machine-dependent. The creation of high-level programming languages led to the introduction of the concept third-generation languages (3GL). These were much less machine-dependent and more user-friendly to write and maintain, but in turn required compilation or interpretation. Early forms of 3GLs are Fortran and COBOL (Bohner & Mohan, 2009), later examples include C++ and Java. Then came the advanced 3GLs, such as Python and Perl, which have some fourth-generation abilities as they are closer to human language (Li, Shi, Hu, Wang & Zhai, 2017).

The term fourth-generation language was used by Martin (1982) to refer to non-procedural,
high-level specification languages. Heering and Mernik (2002) state that most of the languages in this category are query languages, as does Ehrenreich (1981): “The non-procedural aspect of query languages is useful in distinguishing them from programming languages.” This supports that a query language is a more specific form of programming language. In addition, fourth-generation languages include visual, graphical interfaces, which are more easily used for querying than for programming.

From these discussions above, we can safely assume that querying and programming are similar. Although we do not have any information about the mental processes involved, we do suppose that querying is a more specific form of programming, and therefore the processes should be similar.

4.2.1 Human factors in querying and programming

Now that we have established that the reports of human factors in programming are relevant for our findings on querying as well, below we will elaborate on all surveys we have found on human factors in querying and programming. A quick overview of findings is displayed in Table 4.1.

A first thing to note was that all papers were written to convey different messages to different audiences. Some where contemplations of hypothetical factors that served to urge research onwards (Ehrenreich, 1981), others were targeting companies (or managers specifically) to have them facilitate a more optimal working environment (Weinberg, 1971). Some were more focused on psychological factors (Tripathy & Naik, 2014), others were focused on technical factors (Welty & Stemple, 1981). Some talked about experimental methodologies (Weissman, 1974), while others shared experiment results (Reisner, 1981). In short, all of them had something to say on how the research on human-computer interaction should be furthered.

However, from the dates on the papers, we can see that this human factors movement was active for about a decade, before dying out suddenly. It is not entirely clear to us why this was the case. One speculation is that the arrival of late third-generation languages, such as C++ in 1983, may have satisfied these researchers in their quest for more accessible languages. This is not to say that no more papers were written on human factors and computers, but no significant amount of surveys was published after 1990.

To create a relevant, complete and up-to-date overview, we first oriented ourselves on the topic.
## Technological factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>✓</th>
<th>✓</th>
<th>✓</th>
<th>✓</th>
<th>✓</th>
<th>✓</th>
<th>✓</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Query Elements</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Readability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Interface</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Error processing</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cues/Support</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Design</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Visualization method</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Consistency</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Control</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Simplicity</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

## Human Factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>✓</th>
<th>✓</th>
<th>✓</th>
<th>✓</th>
<th>✓</th>
<th>✓</th>
<th>✓</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cognition</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Perception</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Memory</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Problem Solving</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Deductive Reasoning</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Attention</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Individual</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Stress/Workload</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Novice-Expert</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Confidence/Optimism</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Motivation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Gender</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Task Characteristics</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4.1: An overview of all survey papers found. Colored columns are survey papers that specifically mention querying and databases.
CHAPTER 4. AN ANALYSIS OF FACTORS IN HUMAN-DATABASE INTERACTION

One major source of the current list of factors is a book on Human Factors Engineering (Wickens et al., 2004e). Most of the categories that we will elaborate on later through applied literature, are inspired by sections in the book. Later orientation on the papers added some extra categories to the list, which has resulted in a broad set of subjects to discuss. Table 4.1 lists all factors and shows which papers touch upon them. Keeping in mind that some of these papers are very old, not all of the remarks in these papers are applicable. Some of the advice and design recommendation are applicable only to lower level languages, which are no longer used. Other recommendations have long for years been part of guidelines that are being taken into account when designing a system and are therefore not relevant to this chapter.

4.2.2 Observations

From the table, we can see that some papers are very broad in topics they discuss, while others are much more focused on a few factors. As these factors are all still very broad, this does not mean that those papers with few checkmarks are not useful. On the contrary, some of these elaborate on the factors in much more details. Some of the checkmarks in the table refer to a sentence or two on the topic, while others capture full chapters.

The table also shows that there are a few topics that are discussed consistently in most papers. These are readability, cues/support, cognition, memory, and the novice-expert gradient. Items that were barely touched upon are interfaces, control, perception, deductive reasoning, attention, confidence, motivation, and gender. In general, apart from the novice-expert factor, individual factors have hardly been discussed. The main difference between high and low exposure seems to be the difficulty of running experiments on these attributes. However, in this list, interfaces does stand out as a barely covered factor.

There also seem to be some differences between the topics covered in database versus programming papers, although we cannot say anything conclusively about it due to the low amount of papers at our disposal.

This short overview of existing survey papers has shown what the topics of interest are. It has given some pointers as to what aspects were of interest in the past, and which subjects could use some more attention. In the following sections, we will treat all of the factors in depth. We start by introducing and defining each factor. Then we will look at the ways in which it could play a role in interaction with a databases system, elaborate on previous experiments when available and finish with pointers on how to possibly improve understanding through these factors.

4.3 Technical factors

This section discusses in detail which technological factors may have an influence on the human-database interaction. Some of these are very specific to databases and queries, others are more general or discuss programming. This depends on the amount of coverage of the topic in literature.

From experience, we know that there is an effect of the technology itself on the interaction with the user. This is why the field of human-technology interaction exists. The main point of interaction is through a system’s interface. The interface is a set of items that influence the user’s senses. This includes the screen, from which we can read, and any other sources of outgoing information, as well as the peripherals that can be used for data input.

Factors that influence the interaction in a more indirect way are language, support, and control. If these are all favorable to the user, interaction could be a much smoother process. In this chapter, we will explore how to improve the interaction through each of these factors. The sections follow the structure of answering the following questions: What is it? How does it play a role? What experiments have been done? What pointers for improvement can we take away from this?
CHAPTER 4. AN ANALYSIS OF FACTORS IN HUMAN-DATABASE INTERACTION

4.3.1 Language

In Section 4.2 we elaborated on programming- and query languages and their similarities and differences. In this discussion, we distinguished between programming language generations: first through fourth generation languages were defined. The goal underlying these generations was that programming languages should evolve, become more abstract, in order to simplify the process of programming. This drive has led to a developing environment with hundreds of programming- and query languages. Many of these are similar, as they are based on the same lower-level language, but they are still different.

Complications may arise in query formulation due to the complexity of the language itself. There are two ways in which we can distinguish between languages. The first is external, the type of interaction the language requires. The second type is the language’s inner complexity, such as syntax and data model. In this section, as well as in Section 4.3.2 on understanding queries, we will focus on the inner complexity. The external complexity will be covered in the other sections in this chapter, as the interaction is based on the design and visual characteristics of the language.

In this section, we will explore whether there are any indications that one program language is objectively (or subjectively) better than another. It might be the case that novices benefit from different language characteristics than experts would. Some languages might be better suited for human-database interaction than others. In this section, we will include programming languages as they may have features that could be applied to query languages too, even if the language itself might not be applicable for use in database systems.

There have been various researchers defining categories in which we can classify query languages. According to Reisner (1981), query languages can differ with regard to syntactic form, procedurality, and their data model:

- **Syntactic form.** There are three forms of syntax that occur regularly: linear form, two-dimensional form, and positional form (Reisner, 1981). Linear syntax is written from left to right, top to bottom, similar to normal texts. It is used in languages such as SQL. Two-dimensional syntax is more of a diagram, where queries are written by filling in forms or tables (as in QBE). Positional syntax uses sub- and superscript to denote the composition of a query.

- **Procedurality.** A language can either be procedural or non-procedural. Non-procedural queries don’t state how to obtain a result, procedural queries do. A procedural query specifies step-by-step how to achieve the result. Languages such as SQL and QBE are non-procedural, they describe the desired result without specifying how it is to be achieved (Welty & Stemple, 1981). The subset of query languages that are used on graph data (graph query languages) is procedural, as they describe the specific nodes that have to be traversed to reach the result.

- **Data model.** Each query language also relies on its own data model. The most fundamentally established data models are the relational model, the hierarchical model, and the network model (Reisner, 1981). The relational model consists of a collection of relationships, where data can be queried using only the natural structure of the data (Codd, 1970). The hierarchical data model organizes data according to the structural relationships of hierarchical definition trees (Tsichritzis & Lochovsky, 1976). Lastly, the network model stores data in the form of graph structures (Reisner, 1981). Other, more recent data models include the Entity-Relationship model, the object-based model, and the semi-structured data model (Silberschatz, Korth & Sudarshan, 2011).

Contrasting the categorization on language characteristics by Reisner (1981) is the interaction-based categorisation by Uren et al. (2017). They state that query languages and tools can be categorized as one of four types: keyword-based, form-based, view-based or natural language-based.

- **Keyword-based systems.** The user puts in one or more keywords, and the system returns the items that are most closely related. A major advantage of this method is that it is straight-forward, users are often familiar with this type of interaction (Uren et al., 2017; Kaufmann & Bernstein, 2007). A disadvantage is that these systems do not support users that want to explore the domain, as this would require guessing of keywords.
• **Form-based systems.** The user creates a query through a form containing items such as drop-down menus and check-boxes. Again, this is a familiar interaction for most users as forms are used in many systems and websites. Form-based systems also support exploration, as drop-down menus show the users all possible options right from the start. A major drawback of form-based systems is that they are made to fit the domain and thus are not portable (Uren et al., 2017).

• **View-based systems.** The user creates a query by means of navigating through the data, which (in this type of system) often has a graph structure. The advantage of this type of systems is that the data is represented intuitively, which in turn supports domain exploration. A drawback of using view-based systems is that formulating queries can be very time-consuming (Uren et al., 2017).

• **Natural language-based systems.** The user types a question in natural language, which is then translated into a query to run through the database. Its advantage is that sentences allow for using multiple parameters, as natural language is quite flexible. This flexibility is a disadvantage too, as language parsers cannot reliably interpret sentences with many clauses (Uren et al., 2017).

Many papers discuss the option of using natural language for querying. Results of various studies show that no clear conclusion can be drawn on whether natural language querying supports querying performance more than traditional querying languages do.

One of these studies was done by Chan, Tan and Wei (1999), who argue in favor of natural language querying. Not only because this does not require users to learn a language, but also because it reduces cognitive load while creating a query. The users can focus on formulating correct queries, instead of having to worry about whether the query is syntactically correct (Chan et al., 1999). Their results comparing natural language querying and linear-keyword languages do not clearly favor one over the other, although they did find that expressive ease does influence querying performance (Chan et al., 1999).

Kaufmann and Bernstein (2007) also support the use of natural language as an intuitive query language. However, due to the fact that even the most casual computer users can deal with keyword and form-based searches, they wonder whether natural language is still an applicable research field. As natural language parsing is difficult, the results retrieved through natural language queries may often be of very low quality (Kaufmann & Bernstein, 2007). Another issue with natural language as a querying medium that Kaufmann and Bernstein (2007) pose is that a natural language interface is usually domain-tailored. On the other hand, their experiment showed that users prefer natural language interfaces over keyword-based systems. Kaufmann and Bernstein (2007) reduce their finding’s implications by referencing another study, which found that users preferred using keywords unless the retrieval results of full query sentences was better.

Ehrenreich (1981) also discusses natural language. Although the paper is old, some of his arguments are still valid. He states that the differences between a query language and natural language are more elaborate than syntax and vocabulary. The types also differ in their ordering of information, default actions and operand arrangement (Ehrenreich, 1981). This makes it hard to compare the two experiments. One criticism for using natural language is that it is too ambiguous to be used as a query language (Ehrenreich, 1981).

A very different study with programmers was done by Floyd, Santander and Weimer (2017). They compared brain patterns on tasks involving either programming languages or natural language. They found that the brain patterns for programming languages were significantly different from the patterns for natural language. However, the two drew closer for more experienced programmers, meaning that for these experts, programming language reads similar to natural language (Floyd et al., 2017).

A note of caution on interpreting language performance comes from (Weinberg, 1971), who states that we must keep in mind that ease of learning does not equal ease of use for any language. We must keep in mind that these experiments show only short insights into the minds. These findings might not always apply to the long term. However, ease of learning is much easier to measure, as the process is on the shorter side.

From this discussion, we can conclude that there is no convincing proof that natural language
CHAPTER 4. AN ANALYSIS OF FACTORS IN HUMAN-DATABASE INTERACTION

querying is easier for users than dedicated query languages. The most promising approach is to create a query language that leans closely towards natural language and is intuitive to novice users.

4.3.2 Understanding a query

Each query language has different characteristics, which may make it more or less complex to learn. They can differ in syntax and semantics, may have added functionality that other languages don’t have, or may lack functionality. The semantics of a language may shed some light on the goal that the developers of the language designed it for. According to Uren et al. (2017), there are three different types of queries: searching for entities, searching for relations and parameterized searches. Entity searches serve cases where a user searches for a particular item or individual, possibly by means of an ID. Relation-based searches look for relations between entities. Parameterized searches are created for more complex cases, where the user specifies several parameters (Uren et al., 2017).

A well-designed query language should allow all these types of interaction, such that there is no lack of functionality. However, not all types of searches are equally easy for different users. For example, we know that it is not wise to introduce a user to a new language by explaining the most complex concept first. The user must first understand the basics before they can continue to more advanced operations.

This advised process is supported by findings from previous studies, in which novice users struggled with the advanced concepts in SQL. For example, Welty and Stemple (1981) found that novices struggle with SQL’s (and TABLET’s) GROUPBY clause, as well as with the implicit JOIN structure. Reisner (1977) found similar effects on the GROUPBY clause. She suggests SQL should be treated as a layered language: novices should only access the easier layers, more experienced users can access more advanced layers when they need more sophisticated functionality. Thus, every user can advance to their own limits (Ehrenreich, 1981).

Something else that actively helps in understanding code is to apply good coding practices. Dunsmore and Gannon (1980) researched the factors that lead to neat code and discuss five of these factors: nesting depth, ratio of local versus global variables, ratio of data communication through global variables, amount of variables referenced per statement, and the number of live variables. They found that all five factors influence the amount of effort required to write new code (Dunsmore & Gannon, 1980). Although the concept of global variables does not exist in query languages as queries are always local, these factors still give some insight into the process of writing code and queries.

Related to the concept of nesting mentioned by Dunsmore et al. is that of indentation. A complex program structure may be understood faster when the indentation is adequate, as it makes it easier to ascertain functional blocks. Indentation can increase the readability of a piece of code significantly and is one of the easiest coding practices to implement. Other research into code readability has been done by Posnett, Hindle and Devanbu (2011), who created a really simple model of readability based on the volume (which relates program length and vocabulary), the number of lines and entropy of the code. They also talk about indentation, but only for languages that require it (such as Python) and find that these languages are not necessarily better with regard to readability. However, our argument is that for the same lines of code, understanding is increased when the code is indented properly, versus reading all lines left-aligned. This statement is not rejected in literature (Buse & Weimer, 2008; Posnett et al., 2011). Other coding practices that facilitate readability of code are mnemonic variables and commenting (Laughery Jr. & Laughery Sr., 1985).

Analyzing textual features on top of the previously mentioned structural features gives us an even better model for readability (Scalabrino, Linares-Vásquez, Poshyvanyk & Oliveto, 2016). Among the code characteristics they propose are: whether variable names are words from a dictionary, whether these variable names are specific, and the level of cohesion.

Weissman (1974) also discussed factors that may influence readability in much depth. He suggests that the following structural and textual elements can play a role:

- Presence of comments
CHAPTER 4. AN ANALYSIS OF FACTORS IN HUMAN-DATABASE INTERACTION

- Placement of declaration
- Paragraphing and indentation
- Choice of variable names
- Redeclaration of a variable name within the program’s scope
- Use of magic numbers
- Length of program segments
- Passing functions as parameters
- Recursion
- Levels of nesting
- Use of data structures
- Locality of operations
- Changing the iteration variable
- Method of parameter passing

Some of these elements were already discussed and not all of these factors are applicable to querying. However, there is one that does which stands out: Recursion. As with the previously mentioned query concepts of GROUPBY and JOIN, recursion is another relatively complex concept for users. This is not just because it is hard to interpret conceptually, but also because the concept can only be written in a way that is suboptimal for reading comprehension. Users that are required to use recursion features of a language may benefit from a visual interface to represent these recursive measures.

To summarize, not all languages are equally easy to use for all users. We can safely assume that a more advanced user can handle more complexity than a novice. Through the variety of readability models described in the papers discussed in this section, we can give each piece of code, each query, a readability grade. This, combined with the guidelines by Reisner (1977) for layering languages, allow us to support novice database users in different ways from experts. For example, novice users could have their access restricted to simpler commands, and be supported in making their code cohesive. If novice users require the use of more complex features, they could be supported through a visual interface instead of using textual queries.

4.3.3 Consistency

To facilitate human-database interaction, it is important to take consistency into account in the design of the system. There are multiple applications of consistency within a system. Some of them are:

- The consistency of the icons and models within the system with those know in the outside world, to facilitate mental models.
- Consistency within the application, such as the order of buttons or the alignment of text.
- Consistency of the semantics with those of other well-known programming/query languages.

We will discuss each of these in a bit more detail below.

Conforming to the real world. Our knowledge of the world around us gives us a framework as to what possible actions and affordances we can afford in any situation. This knowledge is structured in mental models, each of which applies to a system (we describe mental models in more detail in Section 4.5.1). For example, for a text editor, we usually expect a white ‘sheet’ with some margins to type on, a button in the form of a floppy disk to save and a red button with an x on it to exit the program. An application on our desktop with a trash can symbol should represent the action of throwing something away, putting files in a folder means you are organizing them and clearing clutter away. All these actions conform to those in the real world. This is useful as it does not put any strain on a user to learn all sorts of new interactions. The user can easily equate the action on the computer to the action in the real world, immediately understanding how it works and what to do. For a system design, it is important to conform to this as well. It eases the use of the system and decreases the chances of users becoming frustrated with the system.

Consistency within the application. It is also important for a system to be consistent within itself. As noted before, this may include the order of button presentation and the alignment
CHAPTER 4. AN ANALYSIS OF FACTORS IN HUMAN-DATABASE INTERACTION

of text, but there are many factors.

One of the types of inconsistencies that can occur in a system is non-uniformity. It increases strain on the user and may lead to a bad experience. Some examples of non-uniformity were presented by Weinberg (1971): use of abbreviations for some variable names while using full names for others, arbitrary indenting and spacing, commenting code selectively. The biggest problem according to Weinberg (1971) is the use of mnemonic variable names that do not actually contain what they describe, such as sumAB not containing the sum of A and B.

Another factor in variable naming is abbreviation. Abbreviating all words in the same way helps the user understand how the system works, allowing them to translate the abbreviations back to full words. This consistency should result in a better program (Ehrenreich, 1981). Examples of consistent metrics for mnemonic variables are contraction (delete the vowels), truncation (deleting all but the first three letters of a word), and military abbreviations. Ehrenreich (1981) showed that truncation was the preferred and most optimal solution for the participants in their experiment. Performance increased even more when participants were aware of how abbreviations were formed.

The behavior within a system should also be consistent. For example, Elvins and Jain (1998) mention a system where the ‘undo’ operation can only be applied to certain operations. This is not something the user would expect. When a system provides an ‘undo’ operation, it should work on anything, not selectively. Another example is given by B. Johnson (2015), who describes a system which provided error messages as a service. Some of these error messages were very detailed, whereas others did not provide enough information, and no pattern was found in these inconsistencies (B. Johnson, 2015). These types of inconsistencies should be taken into account when designing a system, as they increase the mental load on the user.

Consistency with other well-known languages. If a user is familiar with programming or querying before learning to use a system, they approach it with two types of assumptions. First of all, they will assume that the system conforms to the real world. Secondly, they will assume that the language behaves as other languages do. This does not mean that the syntax and semantics for each language should be the same (this would defeat the purpose of having multiple languages) but does assume something about the underlying logic, in the same sense that mnemonic variable names should at least hint at their supposed contents.

As Weinberg (1971) states: “When we start using a language it is hard to observe the uniformities. Instead, we will notice those characteristics that don’t match with what is right according to memory: things we learnt from natural language or programming languages we have learnt before.” Thus, creating a new language to be consistent with older languages may simplify the learning process for the user (Weinberg, 1971).

From these three categories, we can see that it is important to take into account the assumptions that users may have about the system. We should aim to design a system that is as consistent as possible. In this process, it is hard to lose track of all the aspects that require consistency as there are many more than those presented here, which means that the process should involve many user tests to account for these issues.

4.3.4 Technology use through errors and mistakes

The use of an information system is never without problems. There are numerous causes that may influence a user’s performance. One of those is mentioned above: the problems with consistency that reside in the assumptions that the user brings to the table. Consistency may capture both interface and language aspects of the system, but in those cases, errors are caused by the system design. In this section, we will look at all the types of errors that the user may make within a database system. Then, we will look at how the system can help the user in recovering from this through documentation, error messages and other support mechanisms. Finally, we will look at how this affects the user’s feeling of control over the system.

Error types. One category of errors are those made when writing a query. There are many researchers that have written about this. For example, (Reisner, 1977) presents the following types: ending errors (using multiples or past tense for words), spelling errors and synonym errors. To fix this problem, she suggests computer aids that can help with writing. These could match
word stems, correct spelling errors and contain a synonym dictionary (Ehrenreich, 1981). Another writing issue is mentioned by Landauer et al. (1982), who names synonyms and polysemy as issues with querying, as humans often express an idea in different ways or use the same word in different ways.

Writing errors may not only occur in the text but may also mix up syntax or semantics. Gould and Ascher (1975) found that participants confuse operations such as ‘or more’ and ‘or less’ in a significant amount of cases. This occurs in queries where they have to apply logical concepts to translate natural language into symbols such as $\geq$ and $\leq$. Other common problems were those where a value had to be translated into another value. An example is: “Return all people who have worked here for more than 5 years”, which has to be translated into the appropriate year of hiring. Another source of confusion was found by Thomas and Gould (1975), who found that participants confused semantically similar concepts such as SUM and COUNT. Thomas (1976) uncovered problems with quantifiers such as ‘all’, ‘some’ and ‘none’. Statements were often incorrect or incomplete, while also not describing the stimulus uniquely. Thomas (1976) notes that logically correct usage of quantifiers is rare in natural language. As Ehrenreich (1981) states: “The computer’s precise interpretation of a [...] statement may not coincide with the user’s imprecise understanding of logic.”

Unfortunately, these errors in logic and semantics seem to require significantly more effort to correct than syntactic errors (Laughery Jr. & Laughery Sr., 1985). In most modern code editors and database systems, syntax errors are already recognized by the software, and thus do not require any effort from the user. This means that in the design of a new system, one should keep in mind the effort required for semantic errors.

**Cues and support.** The number of resources that a user requires for solving problems and repairing errors can be reduced through assistance by the system. This support can already start in the exploratory phases of database interaction, such as showing part of the scope of possibilities (Ichinco, 2016). Another item that may help in the starting phases of database interaction is a visual representation of the data model.

The process of querying a database may be supported by auto-complete functionality, although Ehrenreich (1981) warns us that inexperienced users may have a hard time using it appropriately. Related is the concept of mixed-initiative systems, which may assist the users in writing correct queries (Ehrenreich, 1981). In these systems, the computer will take initiative when it has determined that the user has overlooked some aspect of the task. The user may also explicitly ask the system for guidance.

Another source of support is that of a system’s status information. A sense of presence (i.e. the other person is paying attention) is essential in person-to-person communication. Ehrenreich (1981) states that it also plays a role in user-computer interactions. Therefore, the system should always communicate whether a query has been registered and what it is doing with it (Ehrenreich, 1981). Status information can be communicated in multiple ways. Some of these are simple and contain one type of information, such as a spinning icon to indicate that the system is working on something, or a red wavy underline to indicate the location of an error (Barik, 2015). Another option is to communicate statuses through notifications. These can range from very simple to very elaborate, where the optimal amount of content depends on the user. The required amount of descriptiveness partially depends on the amount of experience the user has (Shneiderman, 1980a). To illustrate the effect of notification content: if a notification is not descriptive enough, the developer may have to look for outside sources to find what caused the problem. Descriptions that are too long, on the other hand, require a lot of reading. The need to actively search for the appropriate information may lead to disregarding of notifications (B. Johnson, 2015).

Another important aspect to support is the documentation of the system. This includes the user manuals, online help and tutorials (Shneiderman, Plaisant & Cohen, 2010). Shneiderman et al. (2010) provides an extensive taxonomy covering all aspects of documentation. One of these is the list of many situations in which documentation can support the user:

- Before starting: tutorials
- At the beginning: getting started documents
- During the task: context-sensitive help
CHAPTER 4. AN ANALYSIS OF FACTORS IN HUMAN-DATABASE INTERACTION

- After failure: help buttons
- When the user returns: start-up tips

All of these types of documentation may contain different information. For a user who needs to recover from an error, context-sensitive information and help are most effective. This includes documentation on all error messages that the system may provide to the user and information on how to solve them. This documentation is called pragmatic information (Shneiderman et al., 2010). The information about all aspects of the interface is called syntactic information, the information about action sequences (tutorials) are semantic information (Shneiderman et al., 2010).

For documentation, it is important to provide information through different media, such as text, graphics, voice recording, and animations (Shneiderman et al., 2010). As different users have different learning styles, providing different sources of information will facilitate understanding for any user.

Control. We have seen that the level of appropriate documentation depends on the user requiring it. To support all users, the most extensive documentation is required, but expert users do not require long error messages. The solution here would be to allow the user control over the amount of information available to them. According to Shneiderman (1980a), the desire for control increases with experience. For experienced users, the computer is an aid in accomplishing personal objectives. Allowing the user to tweak this aid to their liking can increase performance and satisfaction.

Other things that could increase a sense of control is to allow users to change the type and positioning of control. This can also increase performance significantly. The same holds for allowing the users flexibility in semantics. For example, allowing programmers to use both a full word command and its abbreviation, such as squareroot and sqrt, increases control and flexibility for the programmer (Weinberg, 1971). Shneiderman (1980a) does note that allowing users a syntactic choice may lead to more difficulty debugging, as a wrong form is harder to spot when there are many right forms.

Other aspects that influence a user’s sense of control are the timing and stability of a system, although the best performance is the same for any user. It is important for system designers to pay attention to these factors, as quicker responses and better stability may increase satisfaction.

4.3.5 Summary

In this section, we have discussed many different factors on the topic of technology. We talked about query languages and their contest with natural language as a querying medium. There are no clear results on which of the two is better with regard to both performance and user experience. Then we discussed understanding queries and the difficulties that complex language concepts (such as joins and recursion) bring to the learning process. As these concepts provide functionalities that are essential to solving more complex problems, the suggestion by Reisner (1977) to treat query languages in a layered sense may be of use to novice users. The use of good coding practices and appropriate documentation can help any user learn and understand even the most complex query languages.

Then, we discussed consistency as a factor in system design. The various applications of this concept were discussed: conforming to the world, consistency within the application, consistency with other languages. If we use this knowledge correctly, a system can be designed which is almost self-explanatory. In reducing the mental load of our system’s user, we also increase satisfaction.

Finally, we discussed errors and mistakes that could occur during the use of the system. One thing that is certain in this context is that errors occur. Therefore, the most important function in a system is to help the user repair their mistakes. This can be done through various types of documentation and the application of mixed-initiative systems. However, not all users require the same amount of help and assistance. The support system should allow advanced users to control the content of messages and the types of interaction that the user has with the system. This allows the users to use the system at their own level of expertise and thus leads to a much better user experience.
4.4 Task characteristics

This section discusses which task characteristics may influence human performance on that task. This chapter wraps up some of the factors that did not fit in other sections appropriately and revisits some that have been partially explained in other contexts.

Convincing someone that a complex task, on average, takes longer to complete than an easier task, is not very difficult. We all have experiences in this regard that show us the statement is true. It is much harder to define what it is that makes a task more complex, which characteristics make a problem hard to solve. In this section, we will examine papers that discussed these topics.

There are multiple aspects to tasks, each with their own characteristics. We will start our discussion on the formulations of the tasks in any system. Then, we will look at tasks that are specific to querying. We will finish our discussion of task characteristics with specific interface tasks.

First, a recap of what we have learned about task characteristics in other sections. Being aware of task characteristics is important when analyzing user performance, as these are dependent variables that might not be taken into account otherwise. In other sections in this chapter, we have already gained some insights into how we can best present tasks to users to facilitate understanding. For example, making tasks smaller reduces cognitive load on working memory. Also, presenting tasks in an order that matches decision-making processes, concept schemas and mental models can reduce the number of errors and thus increase performance. All of these factors can be taken into account when designing a system.

4.4.1 Task definition

In the papers that we have found on experiments on task complexity, tasks were communicated to participants through words. The tasks were either written down or explained to the participants. The pro of using words in this context is that it is a very specific way to communicate your goals. However, when using words, it is very hard to communicate something in a neutral manner.

For example, the writer of the task may have made assumptions about the knowledge that the user has. This then guides the writing of tasks and their structure. Various researchers distinguish between clearly versus unclearly specified tasks (Schacter, Chung & Dorr, 1998; Tanin et al., 2000). Schacter et al. (1998) called them ill-defined vs well-defined tasks. They researched the effect of specification on information seeking behavior in elementary school-aged children. They also recorded self-reported performance on these tasks. They found that the children performed significantly worse for well-defined tasks, and searched more thoroughly for ill-defined tasks. The children, on the other hand, rated their performances as similar for both tasks (Schacter et al., 1998). Although these findings may not translate one-to-one to the population of novice users that we are researching, they do represent a population that is not trained. The mistakes that these kids made (trusting all answers, discontinuing searches) are actions that novice database users may perform too. Thus, we should make sure that the learning process covers these types of mistakes.

The lack of neutrality in a text may also mean that the task descriptions contain goals or emotions from the experimenter. Vakkari (2005) distinguishes between abstract and functional task definitions. The functional perspective includes the goal that is pursued, the abstract perspective does not. Olah (2006) found that these goals play a big role in the interactions with the system. She says that the goal is continuously important throughout the interaction, as shown by the reflective actions that users undertake during any task (Olah, 2006). This means that the user does not only adapt to the system limitations, but also to the task objectives, which might limit them in their task.

Another study on task goals was done by Choi (2010). She analyzed the influence of task goals on the number of query iterations that were used (image search), as well as on the length of the search term. The three goals she found were: academic, personal interest, and work-related. She found users used significantly more iterations for finding the correct image for academic goals than for the other two goals (Choi, 2010). This leads us to believe that users have different definitions
of satisfactory outcomes for different contexts. Thus, presenting one query in various contexts may lead to different answers. This should be kept in mind when designing tasks within a system.

To summarize, in order to provide users with the best user experience, we should keep in mind that the way in which a task is formulated has a significant impact on the way the user completes the task. As it is almost impossible to formulate a task in a completely neutral manner, we will not give guidelines for that. On the contrary, we would like to encourage readers to use this fact to their advantage.

4.4.2 Querying tasks

In this section, we take a look at what tasks are regularly undertaken with code, such that we can understand the points of view from the user. Many different lists of task distinctions have been proposed, we will present some of them in more detail.

Shneiderman (1980b) suggests the following tasks for programmers:

- Learning
- Design (assessing user needs, costs, determine schedules)
- Composition (creating programs)
- Comprehension (comprehension involves control structures, module design and data structures.)
- Testing (validation)
- Debugging (syntactic and semantic errors)
- Documentation (comments, charts, guides)
- Modification (enhancing old software)

An evolved list was presented by Laughery Jr. and Laughery Sr. (1985) who proposed the following set of categories:

- Learning the programming language/techniques
- Problem definition
- Problem analysis
- Program structure development
- Composition/coding
- Program testing and debugging
- Documentation
- Modification

Tripathy and Naik (2014) discuss program comprehension and distinguish between five types of tasks: adaptive, perfective, corrective, reuse and code leverage tasks.

Unfortunately, these lists were created for programming and software engineering in particular, but parallels to querying can be drawn. For example, composition is similar for programming and querying. The list by Laughery Jr. and Laughery Sr. (1985) is general and completely applicable to querying.

A more functional distinction on querying has been proposed by Vakkari (2005). He states that search actions can be of two types:

- conceptual: articulating and selecting search terms, evaluating the results
- instrumental: manipulating the representations through operators and field restrictions, understanding relations in the data.

Many more lists and categories have been presented by various authors. Unfortunately, none of these researchers present any findings relating task type to user performance. Some of these tasks might be easier, more fun, or quicker than others, but we have not found any task comparisons.

One paper that does present some findings related to task characteristics is Chandler and Sweller (1996). They found that cognitive load increases when the elements that need to be learned have high interactivity, not so much when there are a lot of elements. As a consequence, Chandler and Sweller (1996) suggest that task difficulty is largely independent of cognitive load: learning a lot of things is hard, but not as hard as when these items would have high interactivity. A bigger set of items may be easier to learn than a smaller set if the smaller set has high interactivity (Chandler & Sweller, 1996). More specifically, some tasks are more interactive than others and
thus require more knowledge of the system (Chandler & Sweller, 1996). In short, we should take into account the current knowledge of the user, and based on that edit the task.

There are lots of papers that have analyzed task complexity and its effect on user performance (Chan, Siau & Wei, 1997; Chan et al., 1999; Gwizdka, 2009; Kim & Rieh, 2006; Palvia & Liao, 2000; Siau, Chan & Wei, 2004; Topi, Valacich & Hoffer, 2005). Their overarching finding is that increasing complexity leads to higher interaction times and lower correctness scores. Gwizdka (2009) found that self-reported task complexity was correlated with the amount of effort the participant had extended to finish the task. Topi et al. (2005) found that user confidence is largely determined by task complexity. Both these self-reports are interesting measures in the context of human-database interaction. If we combine the findings by Gwizdka (2009) and Topi et al. (2005), then we hypothesize that users who put a lot of time and effort into a task may have their confidence decreased as a result.

Gwizdka (2009) did multiple studies and found another effect on task complexity. He found that the higher the difficulty of the task, the more pages were visited in search for results.

Another general distinction in task types was made by Schacter et al. (1998), who distinguishes between finding tasks and searching tasks. The former involves an information need where the user knows exactly what they need to fill the 'knowledge gap'. A searching task is more open, the user only has a vague sense of what they are looking for.

A similar characterization was proposed by Gwizdka (2009), based on the nature of the task: Fact Finding and Information Gathering (finding multiple pieces of information on a specific topic). Additionally, he distinguishes between the structures of information that need to be found: Simple (only one piece of information is needed), Hierarchical (finding multiple characteristics of a single concept on different hierarchical levels) and Parallel (finding multiple concepts on the same level in the hierarchy) (Gwizdka, 2009).

This same Simple, Hierarchical and Parallel distinction was also used by Liu, Gwizdka, Liu, Xu and Belkin (2010) in a task on web search. They developed a taxonomy of query reformulation types:

- **Generalization**: removed a search term
- **Specialization**: added a search term
- **Word Substitution**: old and new search terms have at least one word in common and are the same length
- **Repeat**: old and new search terms are the same but formatted differently
- **New Query**: previous and new query do not have words in common

They found that the most frequent reformulation was Specialization, and the most effective reformulation types were New query and Word Substitution (Liu et al., 2010). These are mostly applicable to web search but may apply to some query languages as well.

To summarize, there are many different taxonomies of task types. Unfortunately, most papers have not examined whether these task types are executed by participants in different ways. Most findings in this area of research are on task complexity, where they found that higher task complexity leads to reduced performance. One paper that did experiment on different query types found that some types of reformulation are more common than others. This finding could be incorporated in a system by making the most often used functionalities most easily accessible.

### 4.4.3 Interface tasks

Some tasks do not only have an information retrieval component but an interface component too. In this section, we describe three studies on this topic.

The first study was done by Tanin et al. (2000). They researched the influence of query previews on task performance for both clearly and unclearly specified tasks. Their querying system was form-based and thus fully textual, the query preview was in a visual format (bar chart). They found that query previews can cut down completion times on querying tasks that were exploratory. The previews increased completion times for tasks that were well-defined (Tanin et al., 2000).

Ju and Gluck (2005) proposed a new type of interface based on a model of human processing. Their system used user input and problem-solving methods to relabel and reorganize the menu.
structure. They had participants perform four data interaction tasks in both the new and an old interface. They found no effect of the interface on the success rate of the tasks (Ju & Gluck, 2005). However, they did find that the interaction effect of task type and interface design had an effect on satisfaction, accuracy and completion time. This means that there were differences between the old and the new interface, performance-wise, but only for certain tasks.

Koshman (2005) also ran a user test on a new information retrieval system. The system was designed for experts but was tested on other users too. This allowed the designer to verify their assumptions on the user characteristics. The tasks they used tested whether the users could correctly interpret all interface functionality. It turned out that often, users did select the correct feature to solve the task, but did not (know how to) apply it correctly to solve the task (Koshman, 2005). Thus, the users had enough knowledge to interact with the system, but not enough to complete the tasks in a satisfactory manner.

Although this section is small, we can see from these results that the creation of a database system is not trivial. All of these systems mentioned above worked but were not optimal in all situations.

4.4.4 Summary

Presenting tasks in a neutral manner is very complex. The description of a task will almost certainly contain assumptions, goals or emotions from the task writer. In this section, we have learned that these aspects influence the user. The user’s own goals also influence their actions, as we found that users have different definitions of satisfactory outcomes for different audiences.

Unfortunately, most of the research on this topic has not been quantifiable, but many researchers did find relationships between task complexity and user performance. In addition, some research was done on the query formulations that were undertaken most often, and we concluded that functionalities that are used often should be easily accessible. Other papers about new systems had mixed results on whether their systems were better than existing systems. This demonstrates that creating an optimal database system is not trivial.

4.5 Human factors

In this section, we analyze how interaction is influenced by human factors. These are the factors that are similar for all humans, such as the processes guiding cognition, decision making and problem-solving. For most of these, we have been able to find studies on their influence on information retrieval performance, which we can subsequently generalize to the interaction with database systems.

The effect of human factors on the human-database interaction is mediated by memory and attention. Most of the processes involved, such as problem-solving and decision making, depend on mental models and schemas for performance. More accurate models lead to fewer errors in the interaction process. Thus, we should aim for creating correct mental models through the use of metaphors and documents that explain the workings of the system in detail.

We will look into all elements that play a role in memory and understanding, as well as the more elaborate processes of problem-solving and decision making. Increased accuracy in these processes leads to more successful human-database interaction. The sections follow the structure of these questions: What is it? How does it play a role? What experiments have been done? What pointers for improvement can we take away from this?

4.5.1 Cognition

Cognition is the set of basic mechanisms/processes of perception, thinking and remembering. It comprises mental activities surrounding knowledge: acquisition, storage, transformation and information use (Matlin, 2009). Wickens, Lee, Liu and Gordon-Becker (2004a) represent it as a
pipeline where the processes perception of information, transforming the information and responding to the information follow each other. We will first discuss the overall process of cognition and dive into its separate elements and their meaning later on in this chapter.

Cognition is involved in all everyday processes. Its role in human-database interaction is based on the fact that cognition captures the full pipeline of perceiving, processing and transforming the information into something we can act upon. It is thus involved in various aspects of human-database interaction, but most prominently involves the language learning process and the act of using the database system.

There are many different models representing the process of cognition, as well as debates as to which one would fit best. Tripathy and Naik (2014) captured a few of these models in the context of programming.

- Letovsky Model (1986). Contains five main components: a knowledge base, a mental model, and assimilation process, external software documentation, and a dangling purpose unit. It is meant as a model of enhancement tasks.
- Shneiderman and Mayer Model (1995). Contains three components: short-term memory, long-term memory for both semantic and syntactic knowledge, and internal semantics of the code. It is a model of understanding code.
- Brooks Model: This model views the code as a mapping from problem to program. First, the domain knowledge from both sides is matched, then the code is mapped to this intermediate domain knowledge. This allows for the verification of hypotheses, where the mental model is a hierarchy of primary and subsidiary hypotheses. It is a model of understanding code.
- Soloway, Adelson and Ehrlichs Model (1988). This model assumes the programmer is familiar with the (type of) code. It combines the rules of discourse and the code’s external representations, as well as the programming plans into an understanding process where documents are matched to plans. This results in an internal representation, where a current mental representation may lead to new programming plans, increasing understanding. This model captures understanding code.
- Pennington Model (1987). This model contains a loop, which suggests that programmers incrementally create their mental models. The programmer’s understanding of the code, knowledge of the text structure and their situation model are assimilated into a text base. The documentation, plan knowledge and text base together create a situation model. Finally, the text base and the situation model are cross-referenced and updated. It is a model of understanding code.
- Integrated Meta-Model (1995). This model combines top-down and bottom-up approaches. The idea behind the model is that when a programmer reads a piece of code, they start understanding the code at many different levels simultaneously. For example, control flow, functionality and goal of the program can be understood and connected at the same time. Three models are built at the same time: the top-down model, the situation model, and the program model; the programmer takes an opportunistic approach. These models all possible tasks a programmer can undertake.

Tripathy and Naik (2014) do not discuss which models would be best supported by research, nor which would make the most sense. However, from the descriptions, there are some conclusions that we can draw. First of all, most models have focused on the process of understanding code. All models include a memory component, which makes sense because of the limited working memory that humans possess. Finally, it is important to notice that only one of the models, that of Pennington, contains a loop. Although the cognition process is usually proposed as a pipeline, it is still an iterative process. The action undertaken by a person influences the environment, which in turn leads to changed perceptions, new information to interpret and new actions. Therefore, the loop in the model seems essential.

Facilitating cognition is an indirect process. We can improve the interaction of a person with a system not through improving cognition, but by facilitating its sub-processes: Attention, Perception, Memory, and Problem Solving.
Attention

According to Wickens et al. (2004a), there are two aspects to attention: it requires selection of sensory channels for incoming information, as well as dividing up between all aspects of performance.

Selective attention is guided by salience, effort, expectancy, and value. Salience is a bottom-up process that explicitly captures attention, through abrupt onset, distinctiveness, or when it is auditory or tactile. Expectancy and value are knowledge-based: we know from experience where changes will most probably occur, and focus our attention on those aspects. For example, when waiting at a traffic light, we keep an eye on the situation around us but focus on the traffic light. There, a change should occur that allows us to drive past the intersection. Attention may be inhibited by high amounts of required effort, as turning the head or looking further ahead requires more energy. In short, there is a degree to the amount of selective attention paid.

Dividing attention, on the other hand, is a task that may succeed or fail. According to Wickens et al. (2004a) there are four factors that influence its success:

1. resource demand: there is a relationship between how hard a single task is, and a person’s ability to carry out another task at the same time.
2. structure: if two processes both rely on the eyes, dividing attention will fail. Mitigating between eye and ear is possible.
3. similarity: two similar tasks having divided attention at the same time will lead to confusion.
4. resource allocation: if two processes are undertaken at the same time, one is primary, while the other is secondary. As a consequence, the secondary process will suffer when the demand for attention gets too high.

A. Johnson, Proctor and Vu (2004) write that recent research compares attention to a spotlight. In this analogy the spotlight represents the area of direct attention, where the area of direct attention does not necessarily equal the direction of a person’s gaze. The spotlight can be moved through the observance of cues. These may be endogenous (symbols such as arrows, that have to be interpreted before they can be followed) or exogenous (external cues, usually of sudden onset, that draw attention to their location) (A. Johnson et al., 2004).

The interaction with a database system requires attention from the user. Both aspects of attention are essential here: selective attention should be facilitated by a system to allow the user to work in a focused manner. Attention should also not be unnecessarily divided, as this increases demand on cognitive resources.

Indirectly, attention and user performance on a system are related through the environment the user is in. According to Shneiderman (1980b), the amount of selective attention on a task is influenced by:

- room size
- room structure (windows, doors, ceilings)
- light conditions
- air temperature and humidity
- arrangement of desks
- access to facilities
- noise: quality and intensity
- interference from others
- degree of privacy

Research more directly related to attention and performance has been done by Bailey and Konstan (2006) who focused on resource demand. Their experiment focused on interruption and the speed of recovery, showing that interrupting attention will lead to a larger task completion time as well as an increased number of errors. In addition, Bailey and Konstan (2006) have shown that interrupting attention has a negative influence on affective state. This effect may lead to a negative experience with tools that provide too many interruptions at the wrong time.

Already in 1982, Allen (1982) stated that tasks requiring high processing demands, may reduce the attentional resources available to other (simpler) tasks and can thus lead to more errors. A tool should draw the attention to the most important instructions or tasks, such that the user
may stay on track (Abbott, Bogart & Walkingshaw, 2015).

A final study by Martin-Michiellot and Mendelsohn (2001) studied cognitive load while learning to use an interface. They studied the split-attention effect, which suggests that when two conceptually connected materials are shown as physically separate, they are much more challenging to learn. When these two split materials represent some redundancy, the understanding becomes even more challenging. They did not find any significant effects, but they suggest this might be due to too low cognitive loads during the experiment (Martin-Michiellot & Mendelsohn, 2001). The effect was shown to be significant in its original paper (Sweller, Chandler, Tierney & Cooper, 1990) and is interesting in its applications in the context of a database system.

Deriving from the papers read on the topic, there are many guidelines to keep in mind when designing for optimal attention. First of all, we want to make sure selective attention is in the right place. Through salience, we can focus the user’s attention on the most strategic issues. Use of pre-attentive processing mechanisms can be facilitated by designing for a pop-out effect (where an object contrasts with objects in the background) such that it can grab attention (Koshman, 2005). In addition, we know that when we present a lot of information, the salience of individual parts goes down. A high amount of information also requires more of the limited resources that one has for attention. However, there is a balance required for the amount of information provided as, to some extent, more information leads to a more accurate understanding of a concept (Kulesza et al., 2013).

We also want to make sure attention is divided as little as possible, although we know that dividing attention is required to some extent during querying (Siegmund et al., 2014). One option to reduce division of attention is choosing moments of interruption that hardly disturb the user. In order to do this, we can make the system attention-aware (Bailey & Konstan, 2006). In addition, interruptions should be short, otherwise the user may lose attention (Zgraggen, Galakatos, Crotty, Fekete & Kraska, 2016). If possible, it is also a good idea to create the interface in a way that offloads heavily demanded resources (Wickens et al., 2004a). For example, if the user has to keep an eye on many things at the same time, we can make an important message auditory. We could even distinguish between voice and text (Allen, 1982).

The key to reducing the division of attention is to avoid asking the user to do multiple things at the same time (Wickens et al., 2004a). However, sometimes divided attention is necessary. When two sources of information are presented and must be integrated, such as a graph and its legend, these items with high mental proximity should also have high physical proximity (Wickens, Lee, Liu & Gordon-Becker, 2004c).

The learning process for a new system can be made easier by focusing the user’s attention on the similarities between the current system and one they have used before (Martin-Michiellot & Mendelsohn, 2001). In the case of laymen, showing the similarities between the system and the real world may be helpful, as they can then draw parallels between the two.

Finally, the environment that the user is working in can be optimized. This may free up some attentional resources, allowing more attention to be paid to the task itself.

Perception

The act of focusing selective attention on something is part of the process of perception, which takes the information and extracts meaning from it (Wickens et al., 2004a). However, perception can also occur without attention, in a bottom-up process. The most well-known example of this is the cocktail party effect. It is the effect of hearing someone say your name, even if you were not listening to their conversation. Besides the bottom-up process, there are two other, simultaneous, perceptual processes: unitization and top-down processing. Unitization is the process of grouping together items in view that we know occur together often. Older theories promoted one over the other, but most recent theories combine all three processes into one model of perception (Biswas & Robinson, 2009).

The bottom-up process is guided by physical features (observations) dictating what we see, whereas the top-down process is based on expectations and experiences. Thus, bottom-up processing is heavily influenced by the quality of the features that are registered. Therefore, degrad-
CHAPTER 4. AN ANALYSIS OF FACTORS IN HUMAN-DATABASE INTERACTION

...ation occurs in the bottom-up process through low-quality sensory input, but hardly occurs in top-down processes as they are not based on observation. This also means that it is easier to identify groups of stimuli (such as words) than individual stimuli (a sequence of numbers without a pattern) (Wickens et al., 2004a), as words have a certain probability after a few letters have been identified, whereas there is no such probability for individual stimuli through bottom-up processing.

A system should always be designed with perception and attention in mind. If the design is not made to facilitate perception, usage will be much less efficient and mistakes are made more often. However, as perception is a part of the cognition process, its effect is hard to measure by itself. Thus, perception influences the human-database interaction through the system’s design.

Although perception is hard to measure (gaze direction does not necessarily mean that directed attention is focused there), there have been models that implicitly predict areas of attention within an interface. Biswas and Robinson (2009) created a perception model that predict what a user will look at in order to find the answer to their question. This model was based on mouse events, images of an interface and location of certain objects in the interface. Other parameters include distance to the screen and a set of angles between the user and the outsides of the screen (Biswas & Robinson, 2009). Although this model is very interesting to find the effects of interface design on perception, the model can only be applied at the stage in creation when a first prototype has been designed. It does not help us in our search for design pointers to facilitate interface usage.

Wickens et al. (2004c) on the other hand, do give pointers on how perception can be facilitated on a display (by proxy in an interface), although their sources are unclear:

- Legibility facilitates unitization and bottom-up processing: important aspects are contrast, visual angle, and illumination.
- Humans cannot reliably judge an absolute value through a sensory modality: guessing a noise-level or color is hard, especially when there are many options. We should not require the judgment of absolute limits. In any type of interface, these variables should be represented in numbers and text.
- Support top-down processing by showing things in more than one way. This is especially important for things that defy expectations.
- Redundancy over separate modalities increases saliency.
- Similarity between items causes confusion.

Another guideline is to facilitate bottom-up processing through legibility and logical groupings. When this is not possible, we should facilitate top-down processing through context and redundancy (Wickens et al., 2004a). This does pose a trade-off, as increasing redundancy and context on the one hand increase accuracy of perception and facilitate top-down processing, but at the same time also decreases efficiency (Wickens et al., 2004a).

Working memory

Working memory is a part of the storage system in the human brain dedicated to holding a small amount of information for other processes to work on. It holds both verbal and spatial information, the first in the form of the phonological loop, the second on the visuospatial sketchpad (Wickens et al., 2004a). Figl (2012) adds both a central executive and episodic buffer to those basic elements of working memory. Unfortunately, working memory is limited in both duration and capacity. Similarity and working load play a role in the effectiveness of working memory as well (Chandler & Sweller, 1996; Wickens et al., 2004a).

Working memory is an important factor in human-database interaction as it is involved in understanding and processing data and queries. Long, complex queries may contain so many parts that it is hard to keep them all in memory, which in turn may make it hard to interpret them correctly. Databases may be big and complex, with lots of elements, which requires a lot of attention. To design for low working memory load requires some guidelines.

One of the oldest theories on working memory is that it can capture seven plus or minus two chunks of information, depending on their complexity (Wickens et al., 2004a). A chunk is defined by its physical and cognitive properties: four random letters (P D L I) form four chunks, while
four letters that also spell a word (L O O P) together form only one chunk. This principle is related to long-term memory as familiarity and chunking logic is based on experience. A set of digits of the same length as a postal code is easy to remember if it forms your own postal code, but may be much more challenging to remember for someone else who has another postal code. The same holds for PIN codes, personal identifier codes, and phone numbers.

Although the 7 ± 2 theory is a good starting point, it is also controversial. There are some experiments that have shown that the number of items recalled decreases when the complexity of the items increases (Allen, 1982). It may even be an optimistic limit in any situation (Wickens et al., 2004a). Short-term memory may be affected by the ways in which its content has to be shared between multiple active processes (Allen, 1982). These concerns reflect the dimensions of working memory we discussed before: the number of elements, duration, similarity, and amount of attention.

The role of working memory in human-computer interaction as applied science has been touched upon by a few studies. Its first application is its involvement in the process of learning. Harms (2013) researched the cognitive processes involved in learning programming and believes that current tutorials overwhelm working memory by including too many steps, making learning much more challenging. Cognitive Load Theory implies that learning is facilitated through managing the working memory needs closely (van Merrienboer & Sweller, 2005).

There are two factors that affect working memory: intrinsic cognitive load and extraneous cognitive load. The intrinsic cognitive load is the complexity of the task when placed in the context of the learner’s expertise. The extraneous cognitive load is the complexity of the task arising from the manner of presentation (Chandler & Sweller, 1996; Harms, 2013). By its nature, it is hardly possible to change a task’s intrinsic cognitive load (although the order of learning concepts may help). Extraneous cognitive load, on the other hand, can be decreased by dropping all elements of instruction that are not necessary to learn a concept. For example, to teach someone how to program a loop, we do not need to teach language syntax first. Instead, we can use drag and drop tools or pseudo code to teach the concept (Harms, 2013). This decreases strain and thus facilitates the learning process. Harms (2013) suggests two generalized approaches to support programming students. First, modeling a user’s programming experience gives insight into their intrinsic cognitive load for any concept. Secondly, he suggests decreasing extraneous cognitive load through the analysis of concepts and their load on working memory, and selecting the appropriate concepts for each learner.

Another extensive definition of cognitive load was given through Strother, Ulijn and Fazal (2012). They state that cognitive load is not only influenced by the content and structure of a task, but also by the characteristics of the individual. This includes the distinction between declarative and procedural descriptions and whether the actions are stored in memory or undertaken immediately.

An experiment related to cognitive load research was run by Siegmund et al. (2014), who experimented with functional Magnetic Resonance Imaging (fMRI) on programmers doing coding tasks. Tasks included code comprehension and syntax error detection. They found a relationship between working memory and code understanding, especially through the phonological loop. The experimenters forced bottom-up comprehension of code snippets by removing identifier names that might explain the functionality of a function or variable (Siegmund et al., 2014). They suggest that training (bottom-up) working memory, for example through solving puzzles, may facilitate code understanding. A follow-up study with fMRI techniques was done by Floyd et al. (2017), who aimed to understand how the human brain treats software engineering tasks. They found that the neural representations of programming and natural language tasks in working memory are distinct, although the neural representations for skilled participants were much more similar for the two tasks. This may imply that, for skilled programmers, code review tasks and analysing natural language texts have similar (low) intrinsic complexity. Thus, it seems that working memory load decreases when expertise grows.

Working memory load dimensions have been examined by Kim and Rieh (2006). They undertook a dual-task experiment to measure the mental effort required for certain tasks. Their main task was a search task, the secondary task was visual observation. Missed occurrences on the sec-
secondary task reflected high mental effort. They found that a primary task of viewing search results allowed near-perfect results on the secondary visual observation task, whereas document reading as a primary task led to much more missed occurrences. Their hypothesis is that viewing search results is a keyword based process, where the participants were only scanning for words (Kim & Rieh, 2006). This left them enough working memory capacity to keep up with the secondary task.

Shneiderman (1980a) relates working memory capacity to emotions. He argues that releasing items from working memory can provide a sense of great relief. This means that the brain wants to focus on finishing a task and giving closure. “This pressure for closure means that users, especially novices, may prefer multiple small operations over one large operation.” (Shneiderman, 1980a)

Since we know that the amount of information that can be kept in memory is finite, having an abundance of data may hinder performance on various tasks. However, there are also a lot of opportunities in these situations (Calders, Fletcher, Kamiran & Pechenizkiy, 2012). If we have the right tools for extracting models and patterns, we can make sense of big data. Calders et al. (2012) state that the limiting factor for information overload is the human inability to “define in an unambiguous way what is interesting”. Their solution for information overload is to structure all data through various techniques, depending on its characteristics. Pre-structuring of the data could mean less strain on the brain, which leads to a better user experience.

To summarize these findings into guidelines, we start with the purely resource-based pointers. Wickens et al. (2004a) compiled a list of guidelines for systems that facilitate working memory.

- If a task requires more than one step, support the user in letting them know what steps they have completed and how many still need to be done.
- Exploit the chunking mechanism by splitting up data in recognizable parts.
- Use physical distinctions to reduce confusability.
- Keep working memory limits in mind: when writing instructions, make sure to write them in the appropriate order such that little information has to be kept in memory.
- Negation should be avoided as often as possible as it requires extra thinking steps.
- Replace items in memory with visual information, take the knowledge out into the world (Wickens et al., 2004c).

The work of Harms (2013) and Chandler and Sweller (1996) advises us to take the user’s existing knowledge into account, to speed up the learning process. We should also focus on the amount of training and experience a subject has (Siegmund et al., 2014; Floyd et al., 2017). Scanning tasks are easier to combine with other tasks than reading (Kim & Rich, 2006), and shorter tasks are better than longer ones as they give relief through freeing working memory every so often (Shneiderman, 1980a). Finally, reading or analyzing data that is presented in a more structured sense requires less working memory resources, and will thus indirectly facilitate understanding (Calders et al., 2012).

Long-term memory

Long-term memory is the storage for any information leaving working memory, it allows storing this information for later use (Wickens et al., 2004a; Figl, 2012). It is filled through learning and training, and according to (Wickens et al., 2004a), consists of semantic memory and event memory. Each piece of information has a strength and a set of associations to other pieces of information. When these decay, the information will be forgotten (Wickelgren, 1972).

Semantic knowledge in long-term memory is organized in three structures: schemas, mental models, and cognitive maps (Wickens et al., 2004a). Schemas cover entire knowledge structures about a topic. Schemas about a certain task, or set of tasks, are called scripts. These usually describe what ‘normal’ behavior in a certain situation entails. Mental models are schemas about systems, and are thus the most relevant type of long-term memory for this research. There are two types of mental models: those models when a user knows the functionality of a system, but not its details, and those models that represent how and why something works (Kulesza et al., 2013). Finally, cognitive maps represent spatial information, such as a map of the city or the shape of a constellation.

Episodic knowledge stems from an experience. It is often visual in nature and is biased from...
CHAPTER 4. AN ANALYSIS OF FACTORS IN HUMAN-DATABASE INTERACTION

the perspective of the memory owner. (Episodic) memories rely on extracting the gist, then later reconstructing the details (Wickens et al., 2004a). On the other side is prospective memory. Failure in those parts means that one will forget to do something, such as leaving at the right time or putting the trash outside. Prospective memory can be relieved a bit through the implementation of reminders (Wickens et al., 2004a; Parnin, 2013).

The impact of long-term memory on human-database interaction is again based on memory load. In this section, we will focus on semantic knowledge in general, as well as on mental models as they have a high impact on both learning and using an information system. These are the two most essential parts of long-term memory in our context. Episodic memory is not very relevant for learning, on top of being a highly biased system. Schemas and cognitive maps as concepts are not very relevant either, as the information we are discussing here is more abstract and talks about systems and processes specifically. Mental models, on the other hand, are a resource that is essential for the understanding of a system. Erroneous mental models may lead to behavior with unintended consequences, as an assumption about how the system works may lead to an incorrect schema about how the system will behave, giving some functionality unexpected side-effects (Kulesza et al., 2013).

Many researchers have written about long-term memory and programming or querying languages. One of the first was Laughery Jr. and Laughery Sr. (1985), who stated that learning a programming language may be separated into two parts: semantic and syntactic programming knowledge. The semantic knowledge covers the general programming concepts that are independent of the language. Syntactic knowledge, on the other hand, concerns the details of the language: variable assignments, which characters are valid and the layout of functions and other code. “The best evidence for this model is the relative difficulty of learning the first programming language” (Laughery Jr. & Laughery Sr., 1985). For the first language, a programmer needs to learn both semantics and syntax, while later languages mostly require learning a new syntax.

Tripathy and Naik (2014) approaches this differently. They state that programmers pose two kinds of knowledge: general knowledge and software-specific knowledge. This general knowledge concept seems broader than the semantic knowledge defined by Laughery Jr. and Laughery Sr. (1985), as it also covers algorithms, programming principles, solution approaches, networks etcetera. It is not only connected to the language, as the semantic knowledge is, but also to the architecture and structure surrounding it. Then, Tripathy and Naik (2014) talks about software-specific knowledge, which covers everything about the software to be created. Its application domain, goals, and solution strategy. This is disjoint from Laughery Jr. and Laughery Sr. (1985) concepts as well.

A study by Landauer et al. (1982) showed that human memory is good at associating a fact with other information (what do you think of when I say “Ford”), but bad at listing the full contents of a class (name all car brands). This has to do with the fact that new knowledge is not ordered hierarchically into the brain, but immediately makes pervasive connections to existing knowledge (Landauer et al., 1982). We have to take into account what the natural representation of the information we want to model is, and model it such that it does not clash with its representation in human memory. Landauer et al. (1982) distinguishes between two structures: (3D) Euclidian distance metrics, and a hierarchical (tree-based) system.

Another study by Landauer et al. (1982) discusses the learning process of programming languages. They found that if the language has a low amount of degrees of freedom, it requires a longer learning process. If the user can input mostly free-form, learning is faster, but success depends on the natural-language processing capabilities of the system (Landauer et al., 1982). Natural language processing can be improved if we know which words and sentences are natural for people to use in the information retrieval context. However, the likelihood of two people using the same word to describe the same thing is less than one in five (Landauer et al., 1982). For verbs and program names, it is even less than one in ten. Therefore, there is no real shortcut available for improving natural language programming languages. Thus, the system should adapt to the user and the user to the system.

The written structure of a program itself also plays a role in the strain on long-term memory. The program concepts of locality (all relevant parts of a program are close together) and linearity
(no needless switching from linear code) correspond to synesthetic and sequential memory, respectively. Synesthetic memory is “that part of our memory that enables us to remember things as a whole” (Weinberg, 1971). Good locality within a program means less strain on synesthetic memory. This can also be partially achieved by using well-chosen mnemonic variable names. Good linearity reduces strain on sequential memory, program statements in linear order are typically easier to handle than a branching or looping sequence, concluded Weinberg (1971) in an experiment.

Research into mental models has also been done. One experiment by Tripathy and Naik (2014) found that mental models are unique. Thus, different programmers may have different mental models for the same program, whereas the same programmer may have the same mental model for different programs. They separate mental models in static and dynamic elements. The static elements are text-structures, chunks, schemas, plans, and hypotheses. The dynamic elements are chunking, cross-referencing and strategies. According to Tripathy and Naik (2014), experienced programmers are much better at extracting and manipulating information from code and adapting the static and dynamic elements to form a more accurate mental model.

Another study was done by Kulesza et al. (2013), on mental models and how they are affected by explanations of a system’s workings. They hypothesized that an accurate mental model is built through a balance of explanation size. Making the explanations short will lead to incomplete mental models, making the explanations too big may lead to a complete disregard of the information and an even more lacking mental model (Kulesza et al., 2013). This is due to the fact that a long explanation requires a lot of attention and thus it depends on the benefits that the user expects from attending to the information, whether a mental model will be built. Kulesza et al. (2013) found two barriers to building accurate models. First was the participant’s assumptions about the aspects of the information the system used. The second was the lack of knowledge of the process that the system used. They also found that explanations with high soundness and high completeness were best for building mental models, but when attention or viewed benefits are lacking, Kulesza et al. (2013) suggest reducing soundness will still lead to a mostly correct model with a higher perceived cost/benefit ratio. Users seem most focused on learning about the system when they can use this information for adapting the system to their needs (Kulesza et al., 2013).

Various papers have been written on mental models and their effect on information retrieval. Although capturing a participant’s mental model is nearly impossible, examining the effects of mental model characteristics is a valid approach. For example, Dimitroff (1992) showed that participants with more accurate models found more of the items that they were searching for and made fewer mistakes.

Zhang and Chignell (2001) analyzed the effect of user characteristics on mental models of information retrieval systems. They found that the accuracy of various mental models was significantly influenced by professional status, academic background, and computer experience. Combining this finding with that of Dimitroff (1992) mentioned above, we can now conclude that different user characteristics lead to models with different accuracies, and thus do different levels of performance. The goal of Zhang and Chignell (2001) was to tailor the system to these user-characteristics, in order to both predict performance and accommodate information retrieval for each group.

Both of these last findings are especially useful for the development of a database querying system. We can deduce that in order to improve users’ performance on an information retrieval task, we should be facilitating accurate mental model development (Dimitroff, 1992; Zhang & Chignell, 2001). How we can achieve this was described by Kulesza et al. (2013), who advocate system explanations that are both sound and complete. We also know that experienced programmers build better mental models (Tripathy & Naik, 2014), so these explanations should be focusing on more novice users.

The facilitation of schemas is also important. Novice users may not have any experience with both programming and querying, and thus lack the general, semantic knowledge that helps more experienced programmers to learn a new language (Laughery Jr. & Laughery Sr., 1985). The use of schemas also reduces the load on working memory, as we can directly draw information...
from long-term memory (Chandler & Sweller, 1996). We could support the users by giving them clear instructions on what the general language structure is like, so as to provide them with more elaborate schemas.

For the contents of a querying system, the layout and structure of the represented data are important too. Showing data in a way that matches the structure that this data has in the brain, can help the user understand both the data and the problems more quickly (Landauer et al., 1982). Not only the data should be organized well, the code should be too. Weinberg (1971) showed that locality and linearity, when applied well, simplify the process of understanding code. Other items, such as mnemonic variable names, reduce strain on long-term memory as well.

Mayhew (1992) adds that a conceptual model can be made clearer through: making invisible parts visible, providing feedback to the user of underlying processes, use patterns and rules consistently, using familiar metaphors. However, metaphors may also have negative consequences, as the example used may have fewer capabilities than its counterpart on the computer. This may lead the user to overlook these extra capabilities, as well as skewing the mental model (Wickens, Lee, Liu & Gordon-Becker, 2004d). Thus, mental models can be kick-started through the use of metaphors, but we must be careful that they are formed correctly, in order to facilitate system use as opposed to hindering it.

Problem solving

The act of problem-solving combines the previously mentioned cognitive processes: attention and perception as well as all types of memory. The process compares the current state and the goal state and finds the steps to get from the one to the other. Before being able to solve a problem, a process called troubleshooting has to take place. Troubleshooting requires a series of tests to find out what the problem is, after which problem-solving requires a series of actions to reach the goal state. It relies on the hypotheses that a user can form, and are thus subject to many biases, most notably those in the areas of attention and perception (Wickens et al., 2004a). Problem-solving is an activity that is high in cognitive resource cost, which limits human performance (Wickens et al., 2004a).

According to Jonassen (2000), problem-solving requires two processes. First of all, a mental representation of the situation and the problem. Secondly, it requires action, a manipulation of the problem space. However, all problems are different, and as such, the problem-solving process is never the same (Jonassen, 2000). Jonassen (2000) distinguishes eleven types of problems:
1. Logical problems
2. Algorithmic problems
3. Story problems
4. Rule-using problems
5. Decision-making problems
6. Trouble-shooting problems
7. Strategic Performance problems
8. Diagnosis-Solution problems
9. Case Analysis problems
10. Design problems
11. Dilemmas

From this list, we can see that Jonassen (2000) distinguishes between problem-solving and troubleshooting, but names these troubleshooting tasks in his problem types list. Thus, he does not make the same time-line distinction between the two that Wickens et al. (2004a) does.

In a sense, everything we as humans do, has some underlying hypotheses and goals that make each task a problem-solving task. The interaction of humans with databases involves many such tasks. As query languages are used to search a database and find the answer to a question, they are tools that help in problem-solving. The creation of a query is a problem that needs solving directly. In addition, the process of learning to use such a query language is, to some extent, also a problem to be solved. Thus, problem-solving strategies are very relevant to keep in mind when designing an accessible database system.
Although it is a bit more general than the querying problems that are our focus, user actions in information retrieval systems are also described as problem-solving actions (Brajnik, Mizzaro, Tasso & Venuti, 2002). For these types of retrieval tasks, search strategies are essential. Many authors have come up with names for strategies, as well as categories of user behavior. An attempt at unification of these strategies was made by Wilson (1999), who states that the various models are complementary instead of competing. His final search strategy model defines a route from uncertainty to certainty with three recognizable stages in the process: problem identification, problem definition, problem resolution (and a potential fourth state of solution statement) (Wilson, 1999). According to Brajnik et al. (2002) supporting the user on information retrieval tasks is hard, due to the large number of distinct routes that the user could take to arrive at their goal. The routes depend on user characteristics, task, and information complexity as well as on the cognitive state of the user.

One important user characteristic is experience. As we know from previous sections, novice users do not have very accurate mental models or schemas yet. This also means that their view on the characteristics of the program or tool is lacking, and therefore they are unable to recognize problem types. As a consequence, novice users must rely on general problem-solving strategies (Jonassen, 2000). This may be much less efficient than using specialized strategies.

Laughery Jr. and Laughery Sr. (1985) discuss a model by Newell and Simon (1972), which describes problem-solving in a programming context. Similar to Brajnik et al. (2002), they believe that different individuals will use different strategies for solving the same problem. They also have the same thoughts on the iterative nature of the process. However, the details on how these cycles take place differ. Laughery Jr. and Laughery Sr. (1985) state that humans select the task that “maximizes goal attainment”, after which the state of the environment changes and a new action can be selected.

A more recent report on strategies in information retrieval behavior was done by Olah (2006). An experiment analyzed the transitions between the stages that participants went through in solving an information retrieval problem, leading to a new model for information seeking behavior. Olah (2006) identified fourteen search stages, each with their own cognitive and operational elements. Her exploration into the degree of predictability helps the design of information systems, as these systems can mediate the process of information retrieval better when we know what the most likely next step will be. Their most important finding in our context is that the process is not only iterative, users often go back to previous stages too (even multiple times) (Olah, 2006).

Marchionini (1992) discussed the essential interface features for supporting problem-solving in information retrieval. He poses three assumptions: users want to solve their problem to accomplish their goal, users want to minimize cognitive load and maximize enjoyment, the economic advantage of big data promotes even faster growth of the amount of data available. Although the article is old, Marchionini (1992) makes some points that are still valid. First of all, he states that casual and novice users should use different systems than experts would. Secondly, already in 1992, they understood that showing results in a visual layout is a promising technique. Finally, different types of data have different preferred methods of visualization and retrieval (Marchionini, 1992).

Another example of problem-solving in a programming context is the task of query reformulation, an iterative process. Liu et al. (2010) ran an experiment with three query structure types: simple, hierarchical and parallel, as well as a role for user satisfaction. Query reformulation happens in three instances: when the amount of data retrieved was too large, too small, or the query did not capture what it should have (Fidel, 1985). In the experiment, they analyzed reformulations in two cases: either the previous search found something useful, or they had not found useful information. They found that both search experience and domain knowledge influence the problem-solving strategy (Liu et al., 2010).

A noteworthy finding was made by Laughery Jr. and Laughery Sr. (1985): The posed goal influences the quality of the code. A set goal to code ‘fast’ or ‘efficiently’ result in different amounts of bug hunting and execution time. This may imply that instructions that present a hierarchy in goals, leads to different problem-solving strategies.

From these papers, we can extract another set of guidelines. First of all, from the work of Laughery Jr. and Laughery Sr. (1985), we know that we have to be careful in the statement of
CHAPTER 4. AN ANALYSIS OF FACTORS IN HUMAN-DATABASE INTERACTION

a problem, as the strategy adopted by the user depends on the wording of the goal. Secondly, it can be useful to use models such as those by Olah (2006), as they may guide design through predicting the next steps in the process with high probability.

The selected problem-solving route depends on user characteristics (including experience (Marchionini, 1992) and domain knowledge (Liu et al., 2010)), task and information complexity, and on the cognitive state of the user (Brajnik et al., 2002). Marchionini (1992) adds the type of data to this list. Taking these characteristics into account may lead to higher efficiency and enjoyment. As Marchionini (1992) stated, users solving a problem want to do it fast, with minimal cognitive load, and maximal enjoyment.

4.5.2 Decision making

Each and every one of us makes hundreds of decisions every day, many of them unconsciously. Wickens, Lee, Liu and Gordon-Becker (2004b) describes a decision-making task as having the following characteristics: there are a number of options to choose from, there is some information about every option, there is more than one second of time to make a decision, and there is an element of risk as there is no clear best option. Another important aspect to decision making is that it is active and involves “an intention to pursue a particular course of action” (Hardman, 2009).

The process of decision making is similar to that for troubleshooting, as both are based in hypotheses. Wickens et al. (2004b) defines the process as follows:

1. finding all cues that are relevant for the decision and integrating them
2. generating and selecting hypotheses about the current and future state
3. generating a plan and the corresponding choices of action

The rising complexity of new systems for interaction complicates both the problem solving and decision-making processes. If we think of a visual querying system as an interactive piece of software, querying does not only involve problem-solving skills but decision making as well. In such a complex system, assisting the user in their decision-making processes may lead to more satisfactory outcomes of the human-database interaction.

The first models describing the decision-making process were focusing on rational decisions. The idea was that if we can calculate the cost and gain of each option, we can choose the decision that maximizes this gain. However, human decision making violates assumptions of these rational models (Wickens et al., 2004b). This is due to the fact that humans have limited cognitive resources, and calculating all possible options is a costly operation. In daily life, we have only limited time to make a decision, which is where heuristics come into play. Heuristics are rules-of-thumb, simplified rules for interpreting a situation or making decisions (Wickens et al., 2004b). Simon (1957) theory of satisficing builds on this principle, suggesting that humans make decisions that are satisfactory, that they deem “good enough”. This theory finds a solution for the violated assumptions, as it is much quicker and would thus speed up the decision making process.

Heuristics are a powerful but simple system for decision making, but they are vulnerable to biases. There is an immense amount of decision making biases that should be taken into account. Familiarity with the heuristics can assist the designer in creating an interface or system that counteracts these biases. Some of the more well-known heuristics are the following:

- **Cue primacy**: the first few cues are attributed a greater importance.
- **Cue salience**: the most salient cues will likely capture attention and be deemed more important.
- **Availability heuristic**: people judge something based on how easy it is to retrieve from memory. Something that has been considered many times before will be seen as a likely option in a new case.
- **Cognitive tunneling**: once many cues point to a certain hypothesis, it becomes very hard to change your mind, even in the light of new cues disagreeing with the chosen hypothesis.
- **Confirmation bias**: the process of only looking for cues if they support the hypothesis you selected. Instead of using the information that the cues bring to reject hypotheses, the information is only used to confirm hypotheses.
Framing bias: the way a decision is framed (80 percent win vs. 20 percent loss) influences the decisions taken. A specific example is the sunk cost fallacy, which shows that persons are more likely to choose a risky loss (loss might get bigger) over a sure loss (cutting loose now). This bias is relatively easy to overcome: frame decisions in terms of gains.

Even with all these biases that may occur in the decision-making process, there are various studies that support the theory of satisficing. One study done by Agosto (2002) on decision making on the web found support for satisficing behavior in the search strategies of their participants. They used reduction behaviors such as often returning to known sites for information instead of finding new sources, as well as using search engine synopses to determine whether a site was useful and appealing (Agosto, 2002).

Another experiment on satisficing was done by Warwick, Rimmer, Blandford, Goff and Buchanan (2009). They undertook a long study of two years into their students’ information seeking behavior. In their experiment, they found that students strategically used satisficing, as they estimated the minimum amount of work required to pass. It seems that instead of learning new searching behaviors to find exactly what they needed, they learned to satisfy a goal in the least complicated way. Although satisficing, in this case, does save time and effort, it also limits growth (Warwick et al., 2009).

Satisficing research also regularly focuses on the stopping behavior. As the goal of satisficing strategies is to find a satisfying result, this means that users applying such a strategy would not search until they have found all information on a certain topic, but only continue until they feel they have enough. Experimenters observing such behavior could thus conclude the presence of satisficing strategies. However, these behaviors could also be related to decision making biases as explored previously. Biases such as cue primacy, cognitive, and confirmation bias could also lead to early stopping behavior. This and other satisficing characteristics are important for researchers and designers to keep in mind.

Wickens et al. (2004b) describes the following three options to improve decision making:

• task redesign: systems with greater stability handle errors and biases better, and facilitate mental model development.

• decision-support systems: can be done through either removing inconsistent decision making by the person or providing useful instruments that complement the user.

• training the user: there are many ways in which to do this, but training through simulation seems like the best medium for practice.

From the satisficing experiments, we know that users are conservative in their exploring of information. To promote users exploring the data from more viewpoints, it might be useful to add an interface mechanism that can show unexplored parts of the data. This decision-support system may help the user in gaining a full perspective on the data in a much quicker way.

Information systems are used by both experts and novices. Those that have experience with many different search and decision making strategies, as well as those using satisficing approaches, who are less determined to finish a complex search (Warwick et al., 2009). A system should be designed in such a way that it is approachable for all these different groups.

4.5.3 Summary

In this section, we have discussed all human factors that are involved in human-database interaction. The first aspect we examined was cognition, which consists of attention, perception, memory, and problem-solving. We learned that cognition is an iterative concept: the action influences the environment, the environment changes and the perception of the changed environment leads to new actions.

When reading about attention, we found that there are two aspects to it: selective attention and divided attention. As interaction with a database requires attention from the user, both selective and divided attention are essential for understanding. For example, interrupting attention leads to longer completion times and a negative affective response. Thus, in the design of a system, we should aim to divide attention as little as possible.
CHAPTER 4. AN ANALYSIS OF FACTORS IN HUMAN-DATABASE INTERACTION

Facilitating perception is essential for optimal understanding and usage of the system. We found that we can support perception through high-quality sensory input and redundancy over different modalities.

Working memory is essential for cognitive processing. It is involved in understanding and processing data and queries. To ease the load on working memory, we need design guidelines. Research in this area found that working memory load decreases when expertise grows. For novices, we can support working memory by splitting tasks into smaller parts and letting the user know which tasks they completed and which tasks still require some attention. Long-term memory is also widely involved in human-database interaction. First of all, it is involved in the learning process. In addition, long-term memory contains schemas that represent how things work. These two functionalities are enhanced if a new system looks like and works similarly to systems that the user already knows. For novice users, we can facilitate mental models by giving insights into the underlying mechanisms of the system and through providing adequate metaphors.

Finally, we discussed problem-solving and decision making. Human-database interaction requires both of these processes. Every user has their own approach to solving problems, depending on the user characteristics, domain knowledge, task complexity, and the user’s cognitive state. Adapting to these characteristics can increase efficiency and enjoyment for the user. The decision making process is based on heuristics and thus susceptible to biases. To improve decision-making capabilities, we can redesign tasks, such that they are more stable in the light of decision-making biases. Another option is to train the user, such that they are aware of decision-making biases and can navigate around them. This should help the user work with a database system more effectively, and thus improves the user’s judgment of the system.

4.6 Factors of the individual

In this section, we wanted to discuss the characteristics of individual users that may play a role in human-database interaction. These characteristics may influence interaction both positively and negatively, depending on the person. For example, someone with high motivation for learning may have more effective interactions with a database system than someone with low motivation. Other characteristics, such as level of expertise, influence the interaction without being positive or negative in itself. Analyzing all these characteristics can give us pointers on what a user with certain characteristics would prefer in their interaction with the system. Thus, knowing the user characteristics, adapting the system to it could positively influence the interaction.

- **Stress.** Users may have varying tolerances for stress, which may affect their productivity. In the area of human-database interaction, stress usually arises from high task loads. One of the most famous works on task load and task performance is the Yerkes-Dodson law, or inverted U-curve (Yerkes & Dodson, 1908). This work showed that, between underload and overload, there is an area where task load and task performance are balanced. Another stressor for database users is time pressure. Topi et al. (2005) measured the effect of limited time on query writing performance, and found that limiting the time allotted for writing reduces the mean query correctness scores. Epps (2017) presented some general implications for system design such as minimizing memory load, and advocates automatic estimation of task load and stress such that the system can adapt to each user’s stress tolerance.

- **Expertise.** The preference of a user for levels of interaction depends on the level of skill that the user possesses. Interaction for both novices and experts, and every level of expertise in between, can be facilitated by implementing various types of support for these users. Novices may prefer a completely different interface-type, such as drag and drop (Weintrop, 2015), or they may be supported through items such as start-up tips and tutorials. Documentation and coding help may support beginners (Ichinco, 2016), and keyboard short-cuts may help experts in their interaction (Shneiderman & Plaisant, 2010b). This allows each user to use the system in their own preferred way, increasing their performance.

- **Confidence.** Although confidence usually increases with the level of expertise, it still varies widely within each level. Beginners can be overconfident, experts may suffer from imposter
syndrome. In general, people confident of their skill may perform better because they do not need to worry about executing their task correctly. This also relates to the research by Topi et al. (2005), who found a correlation between task complexity and confidence. Although we cannot always lower task complexity, there are measures to boost confidence, such that the users can still use the system effectively. One effective option is to regularly refresh a user’s knowledge of the system. A study by Siau et al. (2004) found that such a training session increased the users’ confidence.

- **Motivation.** This factor is in effect through the whole process of learning a language, gathering experience using it, and productivity when a user reaches expert status. Perceiving the relevance of usage of a system affects motivation, and can be increased with higher student engagement: manipulating an interface instead of only viewing it (Settle, Vihavainen & Sorva, 2014). Carbone, Hurst, Mitchell and Gunstone (2009) found a relationship between how motivated a student is, and their level of success learning a programming language. Settle et al. (2014) write that boosting students’ motivation is possible through the learning materials, the interactions with fellow students and staff, and the visibility of the learning process. However, we need to be careful: Weinberg (1971) writes that motivation and productivity have a relation where more motivation means more productivity until a maximum is reached, beyond which more motivation will bring performance to zero.

- **Personality traits.** Many of the factors described above were personality traits, but there are more of those that we did not cover. Examples that are mentioned in literature as being positive for programming performance are humility, assertiveness, a sense of humor, flexibility, being precise (Weinberg, 1971; Shneiderman, 1980b; Tripathy & Naik, 2014). We can conclude that knowledge of the user is useful in order to positively influence the interaction.

### 4.7 Conclusion

In this chapter, we have produced a more up-to-date overview of factors in Human-Database Interaction. The field is highly underexposed, as collaboration between researchers in computer science and human-computer interaction on this topic has been scarce. On top of that, most of the research in this field was found to be from the 1980’s. Significant contributions have been made since then on the topic of query languages and information retrieval systems.

In this chapter, we presented an overview of many factors that play a role in the field of human-database interaction. We discussed factors in the fields of technology, task analysis, human factors and summarized the issues on the individual scale. We uncovered a wide range of design guidelines. Some were very specific, others are more general. Below, we will discuss the main points that require attention in the design of a system.

First of all, any system that is developed should be well documented. Documentation should be available for learning to use the system, as well as for purposes of error handling and bug fixing. Extensive documentation can also help any user in building appropriate mental models, which facilitates the learning process.

Secondly, we need to adapt the system to the user’s expertise and personal characteristics. Novice users do not have the experience to use advanced functionality, whereas expert users do not need (or want) extensive feedback during their interaction with the system.

Another essential guideline is that we should not require our users to divide their attention if it is not necessary to do so. Interruptions have a negative effect on affect, and thus should be uncommon and as fast as possible.

The final guideline is that we should keep in mind that any definition of tasks and problems is never completely neutral. Although this does require knowledge of the user, we urge system designers to use this fact to their advantage.

Although we have covered a lot of factors in this overview, there is still a large set of factors that we have not examined. Some examples include privacy, security and information integrity. For future research, we recommend the inclusion of these factors. We also noticed a lack of papers
on experiments for individual factors. Adapting any system to these is very interesting, but further research is required in order to do this. Furthermore, we recommend examining the appropriate form of visualizing data that allows for optimal interaction.
Chapter 5

Human-Database Interaction in practice

In the previous chapters, we have analyzed existing tools and the theoretical requirements for building a successful querying system. These chapters provide us with the design guidelines we need to build such a system. However, most of the analysis so far has been theoretical. We are also interested in how these factors play out in the real world. In order to analyze this in more depth, we undertook a case study on students learning about databases.

In this chapter, we discuss the contents of the study and the results that were found. We start by explaining the theory behind the experiment and by comparing our study to various studies undertaken in the past. In the next section, we explain the methodology and the various types of analyses that were applied to the data. Then we discuss our results in more detail. We finish this chapter by comparing findings from previous chapters to the results of our practical experiment and placing it in the context of improving the user experience. In summary, this chapter answers the subquestion:

How do users interact with existing database systems?

5.1 Introduction

As we know by now, humans generate a lot of data daily. The manufacturers of the various products that we use store this data in such a way that it is not easily available to consumers. Although users receive some insights from these manufacturers into the data that they produce, these are often limited to graphs and lists made available by the manufacturers. Users can only view this data, they cannot analyze their data by asking more complex questions. Even if the users had access to their data, they would typically not be able to interact with it, as database interaction requires training and experience that novice users do not have. Our goal is to amend this usability problem, to make data analysis through databases more accessible to novice users. One aspect of the interaction that can give us insights into the difficulties users experience is the content of the interaction. For relational databases (the most ubiquitous type), these are queries in the Structured Query Language (SQL). Some typical queries are shown in Table 5.1.

This chapter presents a case study of students solving homework problems in SQL. The use of SQL is dictated by its ubiquitousness and the fact that the participants are taught SQL in the course under study. The set of participants can give insight into the perspective of advanced beginners on SQL. This analysis of query content as a standalone topic will contribute to the improvement of interaction with database systems. Uncovering the querying process and finding points of failure in the interaction helps us to improve the system design. This, in turn, will improve the interaction between the human and the database.
Table 5.1: Two example queries: a template and a concrete query.

<table>
<thead>
<tr>
<th>SELECT column_name</th>
<th>SELECT Customer_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM table_name</td>
<td>FROM Customer</td>
</tr>
<tr>
<td>WHERE condition;</td>
<td>WHERE City=&quot;Glasgow&quot;;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer_name</th>
<th>City</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daphne Miedema</td>
<td>Eindhoven</td>
<td>the Netherlands</td>
</tr>
<tr>
<td>John Smith</td>
<td>Glasgow</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>Anna Williams</td>
<td>London</td>
<td>United Kingdom</td>
</tr>
</tbody>
</table>

5.2 Related work

To the best of our knowledge, no analyses on the content of the SQL querying process exist. However, various researchers have written about aspects of the interaction between human and databases, with a focus on errors and complexity. Although we have to keep in mind that many of the papers are decades old. The research we discuss below is focused on relational databases and SQL. Most of the work that has been done to date has been of the quantitative type. Researchers have related error frequency to various other variables, such as teaching method, difficulty of a query, amount of time available, and the novice-expert spectrum.

There are many ways to score the performance on a querying task. The most often used grading scheme is to either count a query as correct or count it as incorrect (Chan, 2007). A second method is to grade the query on a scale, for example from 0 to 5 (Chan et al., 1997). Reisner (1977) differentiated between minor and major errors, which she then translated to ‘essentially correct’ and ‘incorrect’ categories. Bowen, Ferguson, Lehmann and Rohde (2003) calculated a score by counting all micro errors, which they defined as: “the number of changes required to make the query semantically correct”.

The main difference between this study and the ones discussed above is the type of analysis. In this paper, we take an exploratory, qualitative approach. We focus on the characteristics of the queries themselves, the order in which they were performed and the number of attempts made by the participants. This study also includes an elaborate questionnaire to analyze participants’ characteristics, although the data has not been fully analyzed yet. As the analysis is exploratory, no hypotheses were posed.

In the past years, there have been some qualitative studies, for example by Brass and Goldberg (2006). Their analysis focused on non-intended queries: queries that do not have syntactic mistakes but are inherently incorrect, such as contradictory queries that return the empty set. Their list contains 39 errors, in the categories unnecessary complications, inefficient formulations, violations of standard patterns, duplicates and runtime errors (Brass & Goldberg, 2006). An even bigger list was generated by Taipalus and Siponen (2018), who distinguish between syntactic, semantic and logical errors. They present the concept of logical errors, which are different from semantic errors. They redefine semantic errors to be those queries that are wrong, independently of the question posed (similar to the non-intended queries mentioned previously in (Brass & Goldberg, 2006)). Logical errors according to Taipalus and Siponen are those queries that are incorrect as they do not match the question posed (Taipalus & Siponen, 2018).

Other studies on database interaction focus on the comparison of database schemas or query languages. Studies exist comparing different data models (Chan, 2007), comparing textual languages to visual interfaces (Greene, Devlin, Cannata & Gomez, 1990), comparing procedurality (the degree to which the query specifies where the data can be found) (Welty & Stemple, 1981) and comparing language generations (Li et al., 2017).

There also exist works that analyze individual differences between the users of the language, such as age, experience, cognitive skill and confidence (Greene et al., 1990; Chan, 2007; Chan et al., 1997; Reisner, 1981). Their findings include an interaction effect between cognitive skill, language
and query operator type (Greene et al., 1990), a significant effect of age on query completion time (Greene et al., 1990), an interaction effect between confidence and query complexity (Chan et al., 1997) and many others. The current study includes these and other metrics to measure individual characteristics.

Another difference between this study and existing studies on SQL is their goals. Many of the papers mentioned above were meant to improve the teaching process and other aspects of education. Our study, on the other hand, aims to contribute to usability, effectiveness and user experience. One takeaway from previous studies is that SQL (and other languages) should be approached as a layered language, allowing more novice users only easy functionalities, with more advanced users having access to more advanced features (Reisner, 1981).

5.3 Methodology

Participants. This study has a total of 175 participants, of which 159 completed the questionnaire and 104 submitted SQL logs for week 4, leading to a complete set of data for 76 participants. Among the participants were 65 males (age $\mu = 20.8$, $\sigma = 1.9$), 10 females (age $\mu = 21.8$, $\sigma = 2.2$) and 1 participant preferred not to disclose their gender. For week 6, the submitted number of logs was 50, all of which discussed one query: query 8. The majority of the participants in the study were second-year Computer Science BSc students, taking a databases course for the first time (68) or retaking the course (8). All students had basic knowledge of SQL before the experiment, making them advanced beginners.

Questionnaires. To analyze individual characteristics, the students submitted a questionnaire, consisting of three parts. The first part was a set of questions about the participant asking their age, gender, and experience with programming and querying. We gathered this data to analyze whether these aspects had an effect on querying success. As discussed in Section 5.2, previous studies have suggested that effects of these aspects may occur. The second part of the questionnaire was a set of questions on motivation, based on the BREQ (Mullan, Markland & Ingledew, 1997). We believe there may be a correlation between motivation and grades, as well as between motivation and perseverance. The questionnaire ended with a set of questions on self-efficacy, based on the MSLQ (Pintrich, Smith, Garcia & McKeachie, 1991). These questions were included as we believe there may be a relationship between self-efficacy, scores, and perseverance. In addition, motivation and self-efficacy may be correlated as well. The questions on motivation and self-efficacy used a five-point Likert scale.

Homework analysis. The main part of the experiment was homework analysis. Students had to write queries for graded assignments during the course. All query attempts were collected through a customized IPython notebook that executed the students’ queries in SQLite. There were six distinct questions posed. The students did not receive real-time feedback whether their answers were correct. A total of 11041 queries were submitted by the students. An overview of the questions posed to the students is available in Table 5.2. For each of the questions, multiple answers were counted as correct. The correctness of the answers was computed by comparing the result tables.
Table 5.2: The questions posed to the students.

<table>
<thead>
<tr>
<th>query 1</th>
<th>query 2</th>
<th>query 3</th>
<th>query 4</th>
<th>query 5</th>
<th>query 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>List all the product ids that were bought by at most two different customers.</td>
<td>List the customers (id and name) who purchased on the same date both a product with name 'Onions' and a product with name 'Coffee' (not necessarily at the same store).</td>
<td>List the ids of the customers that made a purchase at every store.</td>
<td>Find the names of the store-chains that on average sell products in quantities of more than 4.</td>
<td>Find the name of the product that is sold for the highest price (ever) and the name of the store that sold that item for that price.</td>
<td>Find the largest difference in price for a product in stock (i.e., in the inventory of a store) on 2018-08-23 between two different stores.</td>
</tr>
</tbody>
</table>

5.4 Results

5.4.1 Correctness and attempts

First, we analyzed the overall scores for the queries. The resulting plot is Figure 5.1. Query 2 seems to have been most successfully solved by students, whereas query 4 was only solved by one-third of all participants.

With regard to the number of attempts, all queries show similar student behavior. In Figure 5.2, all bars contain 5 numbers, and the first bar contains 0 through 4 attempts. The figure shows that many students attempted the questions between 0 and 9 times. The trend in these figures is almost purely descending. Hardly any sets of more than 80 attempts were found, so these were cut off the figure.

In addition, we separately counted the number of students who did not attempt the query. The results are in Table 5.3. Almost all queries have the same number of missed attempts, except for query 1. As this was the first question in the homework, it can be expected that students started with this question but may have given up later on.

Table 5.3: Summary of the error fraction outliers.

<table>
<thead>
<tr>
<th>query 1</th>
<th>query 2</th>
<th>query 3</th>
<th>query 4</th>
<th>query 5</th>
<th>query 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>15</td>
<td>14</td>
<td>14</td>
<td>15</td>
<td>13</td>
</tr>
</tbody>
</table>

Figure 5.1: A bar chart representing the fraction of correct answers for each query.

Figure 5.2: A bar chart showing the number of attempts for each query.
CHAPTER 5. HUMAN-DATABASE INTERACTION IN PRACTICE

5.4.2 Query similarity

A second metric that was applied to the data was text similarity. We calculated various similarity metrics over the raw queries, to see whether students completely turn around their submission and whether this influences the end result. Some examples are Figure 5.3, Figure 5.4 and Figure 5.5.

One technical observation is that the inverted Levenshtein Distance (ILD) seems much less robust as a metric than both Jaccard Similarity and the Common Fragments metric. All three of the metrics return different similarity scores for all participants at all times, which makes drawing conclusions more complex. Fortunately, sometimes the metrics do agree, as is the case for participant 280 in Figure 5.5. Their fourth attempt at the query is a fresh start. The ILD finds a similarly new query for attempt twelve, but the other two metrics disagree.

Compared to participant 280, participant 320 seems to change their query around much more. Their graph (Figure 5.4) has lower overall scores on Common Fragments and many downward peaks for the ILD. This may mean that the participant did not really know what they were doing, or tried many different approaches resulting in consistently low scores. An interesting point in the graph for participant 100 (Figure 5.3) is the agreement on attempt 17, where all three techniques show a similarity of (close to) 100%.

Figure 5.2: Histogram for the number of attempts made per query.
CHAPTER 5. HUMAN-DATABASE INTERACTION IN PRACTICE

Figure 5.3: An overview of similarity metrics for pp100 on query 2.

Figure 5.4: An overview of similarity metrics for pp320 on query 1.

Towards successful interaction between Humans and Databases
5.4.3 Query traces

A third metric is the analysis of sequence patterns within a participant’s attempts. This can give insight into the order of solving, whether a student knew their answer was correct and the number of attempts they performed. In combination with the motivation and self-efficacy scores from the questionnaire, this plot can also give some insights into their degree of perseverance, although we have to keep in mind that the students did not know whether their final answers were correct. In Figure 5.7, Figure 5.6, Figure 5.9 and Figure 5.8, some of the paths that students took in solving the questions are displayed.

Some things stand out from these plots. First of all, there are many different ways of approaching the homework. Figure 5.7 and Figure 5.6 show a nice contrast, with the former participant using a very structured approach, and the latter working more chaotically. The linear approach by participant 277 may be explained by their high motivation score, which may lead them to work in a more precise manner. Participant 378, on the other hand, shows a very low score for motivation and self-efficacy. In their query attempt log in Figure 5.8 the participant also shows to not have made many attempts.

An interesting pattern also arises for participant 127 in Figure 5.9. We can see here that a repeating pattern occurs in the execution of queries. This may be due to the fact that participants used IPython notebooks. In the course Data Analytics for Engineers, students learn to work with these notebooks. The course teaches them the “Notebook Workflow”, which requires students to run all cells sequentially, which is one of the reasons we might be seeing such a pattern. Besides
5.4.4 Questionnaire

On top of the homework analysis, we analyzed some of the questionnaire questions and searched for correlations, as in Figure 5.12. As seen in the previous section, we also applied the values of motivation and self-efficacy to other findings.

First, we check whether the variables are normally distributed. From both Shapiro-Wilk tests and normality plots, we find that the null-hypothesis cannot be rejected, and thus we can assume the data is normally distributed. Normality can be rejected for the grades data, but we can argue that this makes sense. The participants were not randomly selected, as the students could choose whether they wanted to participate or not. This may lead to a skewed sample, as we believe successful students are more likely to share their data.

Eyeballing the boxplots in Figure 5.10 and Figure 5.11 further convinces us that the data is normally distributed. It is important to note that the metrics have different scales on which they are calculated. The BREQ questions on motivation are averaged and then weighted to give an index to the degree of self-determination. This means scores can go above the 5 on the Likert scale, as well as allowing for negative scores. The self-efficacy scale is graded by taking the mean of the scores on all questions. This leads to a smaller range of possible values and a maximum score of 5.0.

Now we move on to the analysis of the questionnaire. We would expect some correlations between the three aforementioned variables, as students with high motivation and self-efficacy might put more effort into the work and thus perform better. As can be seen from the plots the correlations are very low, but the correlation between self-efficacy and grades is significant with $p = 0.036$. This means that students that had a higher belief in their own skills, received higher grades for their homework. This is interesting, as self-efficacy is a subjective concept. People with high confidence in their own skills do not necessarily perform better by definition, but in this case, they actually did.

We also expected a correlation between motivation and self-efficacy. A calculation shows a near-significant relationship: 0.217 with $p = 0.061$.

Figure 5.7: The query attempt log of participant 79.

Figure 5.8: The query attempt log of participant 378.

working according to the standards presented to them previously, participant 127 seems to also have answered most queries correct in the end, which can be deduced from the number of blue dots in the figure.
CHAPTER 5. HUMAN-DATABASE INTERACTION IN PRACTICE

A final graph displays age versus grade. Figure 5.13 shows the wide range of grades that each age group was awarded. No significant differences were found between age groups. From eyeballing the plot we do see that the grades from the 21-year-olds are on average much better than those from the other groups.

5.4.5 Results summary

In this section, we analyzed the characteristics of the data and the contents of the questionnaire. There are three important findings in this section. First of all, when we look at the number of attempts, the behavior of the students is similar. The number of students who did not attempt the questions was approximately equal, and the fraction of correct answers was also similar for most queries. Standing out in these plots are query 1 (with most students attempting it), and query 4 (with the lowest fraction of correct answers).

Then, through the plots of the students’ traces, we found that students have widely varying methods of working. Although to some extent, we knew about learning styles, the impact of this finding is significant. It means that for improving the interaction between human and database, we have to keep the user’s method of working in mind.

Finally, we found a correlation between self-efficacy and homework score. This means that we might be able to predict success from the self-efficacy scores as well.
CHAPTER 5. HUMAN-DATABASE INTERACTION IN PRACTICE

Figure 5.12: An overview of similarity metrics.

Figure 5.13: A boxplot of all grades split up by age group.
5.5 Raw text analysis

The next step in the data analysis was to look at the raw text, the style of working, the types of errors that were made, and any other notable characteristics. As a starting point, we used the SQL error type analysis by Brass and Goldberg (2006). They define a broad range of mistakes that were made in the queries they analyzed. Below, we give an overview of the categories they define, and some of the errors that fall into that category (Brass & Goldberg, 2006).

1. Unnecessary complications: there is an equivalent but simpler query possible. Reasons may include:
   - The user knows the alternative is wrong, but the errors in the current query are hidden by complexity.
   - The user did not know the alternative is possible.
   - The user believes the current query runs faster than the alternative.
   - The user believes the current query is easier to read/maintain for humans.

2. Inefficient formulations: slowing down execution if the optimizer fails to catch these mistakes.
   - Unnecessary DISTINCT
   - Unnecessary JOIN
   - Tautological or inconsistent subcondition
   - LIKE without wildcards
   - IN/EXISTS can be replaced by comparison
   - DISTINCT in aggregation function

   - Missing JOIN condition
   - Condition in the subquery can be moved up
   - HAVING without GROUP BY
   - DISTINCT in aggregation
   - OUTER JOIN can be replaced by INNER JOIN

4. Duplicate errors

5. Runtime errors

We started reading through the query logs with this list in mind. For many of the errors, we found an example of a participant who showed this behavior. On the other hand, some of the errors would not occur in the logs due to the nature of the questions posed. For example, the queries used AVG and COUNT aggregations, but the use of DISTINCT in this case is not a problem, thus DISTINCT in aggregation did not occur in the data. For visual examples of occurring inefficient formulations and violations of standard patterns, see Appendix A.

It is interesting to see that some students (for example participant 99 in Figure 5.14) regularly write incredibly complex queries with many complications (the first category by Brass and Goldberg (2006)). We believe that in these cases, the students do not know nor understand that the queries can be written in much simpler ways, as the example in Figure 5.15 indicates. Writing the queries in such complex ways increases both the difficulty of solving the questions and the execution time.

Of course, examples from the category runtime errors by Brass and Goldberg (2006) occurred regularly as well. The students do not have a thorough understanding of the language yet, and make mistakes, as in Figure 5.16.

We also looked at the logs without searching for specific patterns and found some interesting phenomena. First of all, some students solve the problems by building a query step by step. Especially for the more complex queries, they first select the appropriate table, then build a subquery, add a condition to the where clause, and so on. This way of working gives the participants a good insight into how their additions to the query change the result table. A similar phenomenon is students building a query that has SELECT * as selection, then when they believe the query is done, replace the * by the appropriate selection condition. An example is shown in Figure 5.18. The result tables were left out here as they were very long.

Students also posed exploratory queries, to analyze the schema and see how many records the data contains. For example, query 3 asks for customers that bought something at every store.
CHAPTER 5. HUMAN-DATABASE INTERACTION IN PRACTICE

Figure 5.14: Participant 99 writes complex queries that are hard to read.

```
query5 = ""
SELECT pID, sID
FROM purchase
WHERE price = (SELECT MAX (price)
    FROM purchase) and pName = (SELECT pName
    FROM product
    WHERE pID = (SELECT pID
    FROM purchase
    WHERE price = (SELECT MAX (price)
    FROM purchase)) and sName = (SELECT sName
    FROM store
    WHERE sID = (SELECT sID
    FROM purchase
    WHERE price = (SELECT MAX (price)
    FROM purchase)))
"
...
```

Figure 5.15: An example that shows it is probable that participant 99 lacks understanding of the query writing process.

```
practicequery1a = ""
SELECT i.cID
FROM purchase i, shoppinglist j
WHERE i.date = j.date and i.cID = j.cID
EXCEPT
SELECT i.cID
FROM purchase i, shoppinglist j
WHERE i.date = j.date and i.cID = j.cID and i.pID = j.pID
"
```

Figure 5.16: Runtime errors occur in the work of participant 115.

```
query1 = ""
SELECT pID
FROM purchase EXCEPT(SELECT pID
    FROM purchase AS p1, purchase AS p2, purchase AS p3
    WHERE p1.pID = p2.pID
    AND p1.pID = p3.pID
    AND p1.cID <> p2.cID
    AND p1.cID <> p3.cID
    AND p2.cID <> p3.cID)
"
```
CHAPTER 5. HUMAN-DATABASE INTERACTION IN PRACTICE

Figure 5.17: Participant 143 also writes unnecessarily complex queries, but these are easy to read.

Then, some participants wrote queries first to find or count the number of distinct stores. Query four asks for a certain quantity of items, participants can be seen playing with the condition, changing it around and substituting other numbers to see what the results would be in that case (see Figure 5.21). The students also keep posing exploratory queries that reveal the database schema, throughout their homework. This seems to indicate that they regularly forget the exact contents of the schema.

From the notebooks, we can also see that students worked together, and gave each other their solutions. This is clear due to the style of typing changing a lot (caps versus lowercase SQL keywords and different types of aliases). One such example is Figure 5.19.

One complication is the fact that some students renamed the variables that we put in the notebooks. This hindered our quantitative analysis of their queries, and we did not include these logs.

5.5.1 Text analysis summary

In this section, we looked at the queries in their unprocessed form. We first compared our data with the error list as presented by Brass and Goldberg (2006), and found that our work supports their findings. We found proof of most error types existing, and ones that we did not find were due to the types of questions asked.

Then we looked at the queries in the context of the problem-solving process. We found that most of the students build queries with very high complexity. The number of concepts, brackets and question structures they apply is too complex. This might be explained by the fact that they approach querying as a programming problem, applying UNION and JOIN manually. However, SQL has some mechanisms to apply these operations implicitly. The students might not have enough knowledge of SQL yet to write their queries in this shorter form.

On top of these ‘errors’, we also found traces of productive problem-solving. Some students systematically build up their queries, splitting their solution up in building blocks, such that they can see what each addition to the query changes in the result table. Other noteworthy findings are helper queries, exploratory queries and working with fellow students.
CHAPTER 5. HUMAN-DATABASE INTERACTION IN PRACTICE

70 Towards successful interaction between Humans and Databases

Figure 5.18: Participant 338 builds a query with SELECT *

```
query6 = ""
        SELECT *
        FROM inventory AS i

pd.read_sql_query(query6, conn)
```

Figure 5.19: An example of a participant’s style changing, indicating sharing of queries.

```
query2 = ""
        SELECT ABS(s1.unit_price-s2.unit_price) AS diff
        FROM inventory as s1, inventory as s2
        WHERE s1.date = '2018-08-23' AND s2.date = '2018-08-23'
        GROUP BY s1.pid
        HAVING MAX(diff)

pd.read_sql_query(query2, conn)
```

Figure 5.20: Participant ran various queries under different variable names than the ones we chose.

```
correctSolutionUsingTestQuery = ""
SELECT DISTINCT c.CID, c.CName
FROM customer c
WHERE c.CID IN
    (33, 42, 95, 86, 110, 152, 169, 171, 175, 190, 111, 176, 177, 196, 15, 23, 129, 52, 59, 92, 37, 122, 124, 176, 190, 31, 172, 181)
```
Figure 5.21: Participant 115 is exploring the data.
5.6 Query Complexity analysis

During the raw text analysis, we found many indications of too high levels of complexity in the raw text. Thus, we decided that the next step in the analysis was to compare query complexities. The complexity of the students’ work can be compared to the complexity of the correct answers, as well as comparing complexity between students. The changes in complexity of students’ answers throughout their homework session give insights into their problem-solving process. The relation between the complexity metrics, as well as the differences in complexity between questions, have the potential to show differences in working style between queries and between students.

5.6.1 Theory

There are various metrics of query complexity, an overview of all metrics used in this chapter is in Table 5.4. Bowen et al. (2003) performed a similar study to the one undertaken in this chapter, and also calculate a query complexity count. Their definition of query complexity is to count the total number of operators and operands in each query. These include SQL keywords, names, operands, brackets and punctuation; any word or character in the query that is not a space. An example of a calculation is displayed in Figure 5.22.

Another suggestion for the analysis of query complexity was posed by Chan, Lim, Lim and Loh (1994). They proposed the use of Software Science Metrics, a theory that was introduced by M.H. Halstead in the 1970s in order to predict software complexity (Chan et al., 1994). The metrics are based on the number of operators and operands in the code, which are combined in various formulas to calculate different metrics: the Volume Equation, the Level Equation, and the Effort Equation. They find that although the metrics were developed for software complexity, their application to calculate query complexity leads to accurate measures. The approach by Chan et al. (1994) is similar to that of Jain, Moritz, Halperin, Howe and Lazowska (2016), who suggest a query complexity measure that combines the number of operations in a query with the number of distinct operations. Unfortunately, they do not state exactly how to combine the two.

On top of these metrics, we looked at the characteristics of SQL as a domain specific language. What aspects of the language indicate complexity? One hypothesis is that brackets are used to different extents by different users. The number of brackets used may give some indication as to how complex the written query is.

An additional metric of interest to us is the running time for any query. More complex queries will often lead to more complex execution plans, which in turn leads to longer running times.

We will calculate all the measures needed for the Software Science Metrics, as well as the complexity metric suggested by Bowen et al. (2003), and compare these for the data that we have.

**SQL Query**

<table>
<thead>
<tr>
<th>SQL Query</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT ITEMB.itemno, recepno</td>
<td>6</td>
</tr>
<tr>
<td>FROM ITEMB, RECEIPTSB</td>
<td>4</td>
</tr>
<tr>
<td>WHERE ITEMB.itemno = RECEIPTSB.itemno</td>
<td>8</td>
</tr>
<tr>
<td>AND paydate &lt; (rechate + (0.5*termdays))</td>
<td>11</td>
</tr>
<tr>
<td>Total Complexity For The Query</td>
<td>29</td>
</tr>
</tbody>
</table>

Figure 5.22: An example of the query complexity as calculated by Bowen.
CHAPTER 5. HUMAN-DATABASE INTERACTION IN PRACTICE

Table 5.4: All metrics we evaluate in this chapter.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Source</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bowen</td>
<td>Bowen et al. (2003)</td>
<td>Count all operands and operators in the query.</td>
</tr>
<tr>
<td>SSM-Volume</td>
<td>Chan et al. (1994)</td>
<td>Reflects the unique operators and operands in the query.</td>
</tr>
<tr>
<td>SSM-Level</td>
<td>Chan et al. (1994)</td>
<td>Reflects how well a query is written, (L=1) means the most ideally written query.</td>
</tr>
<tr>
<td>Total sets of brackets</td>
<td>-</td>
<td>All complete sets of brackets occurring in the query.</td>
</tr>
<tr>
<td>Total nesting</td>
<td>-</td>
<td>Maximum level of nesting occurring in a query.</td>
</tr>
<tr>
<td>Nested negation</td>
<td>-</td>
<td>Maximum level of nested negation in a query.</td>
</tr>
<tr>
<td>Running time</td>
<td>-</td>
<td>Time the SQL engine takes to run the query.</td>
</tr>
</tbody>
</table>

Figure 5.23: An overview of the maximum occurring nesting level per student per query.

5.6.2 Comparing queries

In the raw text analysis of the queries, we already hypothesized that many of the students do not have an appropriate understanding of SQL and thus write unnecessarily complex queries. This finding is supported by the metrics of nesting that we find. Especially the subquery nesting finds very high numbers of up to five (see Figure 5.23), and total bracket use of up to ten pairs (see Figure 5.24). This would not be necessary in any case for the asked questions, as the required queries are not that complex.

To compare whether students were consistent in their behavior over all different questions, we compared the average values for various complexity metrics per student per query. For the running time, we can see in Figure 5.25 that they are low to begin with, but some of them are more consistently low than others. In particular, query 2, 5 and 6 have lower running times. This may be explained by the number of results in the result table, as shorter tables require less time to be retrieved. Related to the execution of queries is the number of queries that do not run. The plot displaying these numbers is Figure 5.26. When we put the data in these plots together, we can see that although there is a wide variation in running time both between students and between queries, this does not hold for the number of errors. The number of errors up until the top of the box-plot whisker is almost equal for all queries. This suggests that the querying process of the students did not differ much between queries in this regard. Of course, some outliers exist, and
CHAPTER 5. HUMAN-DATABASE INTERACTION IN PRACTICE

Figure 5.24: An overview of the maximum occurring total sets of brackets per student per query.

Figure 5.25: An overview of the average running times per student per query.

to place these in context, we counted the fraction of errors that occurred in Figure 5.27. In this context, we see that there is high variability in the fraction of errors that students make. Some students have close to 0 percent errors, others are up to 90 percent.

5.6.3 Experience

To go in more depth, we decided to remake these plots while distinguishing between students with SQL experience and those without, as extracted from their answers to the questionnaire. In Figure 5.28 and Figure 5.29, we do find some results. Unexpectedly, the two groups have similar performance on the first two queries, with similar outliers too (although the non-experienced set contains more outliers). On all subsequent queries, the students with experience perform better. Their relatively worse experience on the first two queries might be explained by the fact that the questions were harder than they expected, or that the knowledge was not as fresh in memory. Students with experience might not give the first (relatively easy) questions much thought, as they have done it before. By the time they reach the harder questions, they may be both more aware of the difficulties of the questions and more warmed up and practiced through the previous
CHAPTER 5. HUMAN-DATABASE INTERACTION IN PRACTICE

Figure 5.26: An overview of the number of erroneous queries per student per query.

Figure 5.27: An overview of the number of erroneous queries per student per query.
CHAPTER 5. HUMAN-DATABASE INTERACTION IN PRACTICE

Figure 5.28: The number of erroneous queries for students without experience in SQL.

Figure 5.29: An overview of the number of erroneous queries for students with experience in SQL.

questions. In addition, students with experience may have drafted complex queries before, whereas novices have not.

In this context, it is important to note that although the sizes of the two groups (with experience vs. without) were similar, half of the students that are counted as experienced did not send in Jupyter logs. The course allowed the students to finish their homework in multiple ways, one of which was command line. This method did not produce the logs that were used to analyze the students’ process, and thus the data of these students could not be analyzed. This leads to a sample of experienced students that is only half the size of the sample of non-experienced students. This should be kept in mind when looking at these figures.

5.6.4 Comparison of metrics

We also compared complexity metrics per student, to see whether there were relations between the metrics. Three examples are in Figure 5.30, Figure 5.31 and Figure 5.32. The changing complexity throughout the attempts gives some insights into the process that these three participants went through. For participant 189, a clear growth can be seen in the Bowen and SSM-Volume metrics.
The student has kept increasing the complexity to reach their desired answer. Then, at attempt 10 and 11, they fall back to a low complexity (also for running time and nested negation), presumably to check the data once more, before finishing their answer by adding a subquery to their answer. They end up with a query that has a good running time, appropriate Bowen complexity and the correct level of negation- and subquery nesting.

Participant 255 used more attempts than participant 189. The complexity metrics for their queries show ups and downs in multiple aspects, which may reflect that they tried different approaches. From the subquery nesting depth, we can see that at attempt 14, the student seems to start over. They started with a nesting depth of one, but reset to zero to build up over the following attempts to using three sets of brackets in the end. Thus, at attempt 14 they might have found an error in their previous approach, leading to insights into the correct approach. For running time, we see four extreme outliers around attempt 10, with a pattern for these four attempts arising in the Bowen, SSM-Volume and SSM-Level subgraphs. It seems the participant executed an inefficient query four times. Looking at the raw data, we see that this is approximately correct, attempt 9, 10 and 11 are the same query, which has one more concept than attempt 8. As to why a student would execute the same query three times, we can only guess. One possibility is that, as the running time was relatively long, the result was unexpected, thus leading the student to verify the results. Another option is the student taking a break from the question, but we have no data that could prove this.

The first thing that stands out when analyzing the complexity for participant 320 on query 6 is the fact that they have not used any negation concepts in their query, whereas the correct answers do. Another striking detail is the wide range in complexity scores that they received throughout their attempts, for all measures except running time. As the running times are very close to zero (and lower than that of the correct answer), their queries must be very efficient. Combining this information with the missing negation, the answer that this person worked towards may not have been correct. From the grading of the homework, we could extract that the student indeed submitted an incorrect answer to this question. An interesting pattern shows a slow increase in the number of brackets that the student used (up to six sets for a few of the last attempts), with nesting levels up to three. The correct answers only used one set of brackets. When we discard the first two attempts from this analysis, the student seems to have built the query and its brackets up in a slow and systematic way, sometimes removing a set, later adding one again. This increasing complexity is not reflected in the Bowen and SSM metrics, which stabilize after attempt 35 or so. This could be explained by the student adding brackets, but at the same time removing other concepts. This makes the metrics that count the number of concepts relatively static. Finally, it is interesting to see that this student created so many non-running queries (all those with red dots in the running time plot). Their approach of building up the query seems to have been necessary in order to create running queries. A pattern can be seen where once the student comes across a non-running query, they decide to try some things and then remove a level of complexity (brackets) in order to take a step back and try again.

In general, we found that for some students higher complexity scores on the Bowen metric and SSM-Volume led to lower running times. The primary cause most likely is the fact that SQL running time is heavily influenced by the size of the result table. The later, more complex questions most likely had smaller result tables, leading to lower running times. Other explanations include better-defined questions (easier to interpret correctly), easier questions or accessing a different number of tables. Although for some students we found a negative relationship between these metrics, this relationship does not always occur/exist. However, running time is a useful tool to analyze how many results the student likely had.

5.6.5 Nesting metrics

We did want to compare the process that the students went through for all queries. Three metrics were well-suited for that as they were relatively restricted: the nesting metrics. That means the total sets of brackets used, the maximum level of nested brackets, and the nested negation. In order to compare the queries without comparing students, we created density plots. These plots
can uncover patterns in the nesting behavior of students throughout their sessions, as it layers all of them on top of each other with low opacity. In such a plot, both a high number of dots and a low number of dots can be of interest, so we will discuss all three of them in depth. As with the previous plots, the orange lines show the level of nesting of the proposed solutions.

The plot for the total number of brackets is Figure 5.33. It is the most densely populated one, as it has the widest y-axis, going up to 10 sets of brackets. From all of the sub-figures, it is clear that students often used more brackets than strictly necessary. Especially query 2, 3, 4 and 6 show this behavior. Query 5 had mostly one to three sets of brackets, close to the answer. It did have the highest outliers (10 levels of brackets), but this is uncharacteristic behavior when compared to the overall pattern, and most likely originates in one student’s behavior. In short, the behaviors on this complexity metric for query 1 and 5 were similar and predictable. Participants without experience are expected to use more brackets, as this leads to higher trust and security that the result table reflects the students’ queries. For the other figures, interesting things are going on. In general, all of the plots show sharp dots at the start and end of the plot. This is mostly an artifact of the normalization process. Students with a low number of attempts lead to sharp dots at significant parts of the plots, such as the start, the end, and the exact middle. When looking at the plots closely, you can see that some areas on set intervals are darker as a result of this behavior. Then we get to the more complex aspects of these plots. We can interpret these density plots a little like heatmaps. Darker areas mean that more people were in this spot at that point in their session. These darker areas thus tell us something about the process that multiple participants went through. For example, for query 2 there is a ‘cloud’ in the middle of the session for nesting level 1. Although we can see that the majority of query attempts contained zero brackets, many participants seem to have used a single set of brackets in the middle of solving the query. This suggests that the students ran into issues that required them to add brackets in order to make heads or tails of the question. Another interesting observation is the lack of queries
with 0 brackets for query 3 through 6. For most of the queries the number of attempts with 0 brackets slowly decreases as the line becomes more and more see-through, but for query 5, hardly any attempts without brackets were made. Finally, there is a very nice pattern to be seen for query 6. Many of the queries are all over the place, with very high scores for the sets of brackets used in these queries. However, we also see a correlation between the progression in the session and the sets of brackets used. Throughout the sessions, many participants seem to have, slowly but consistently, been increasing the brackets. Many participants seem to have used this process to get to their answers, with students submitting a final answer with up to four sets of brackets where only one was required. This might indicate that the query was relatively complex for the students to finish and that they required some more help.

The density plot for the maximum number of nesting is Figure 5.34. As in Figure 5.33, we can see the artifacts from normalizing the number of attempts. Important note: nesting levels of 0 and 1 are the same as total sets of brackets being 0 and 1, so the bottom parts of the two figures look similar. This is reflected already in the first plot, where we can see that for query 1, the majority of participants used 0 brackets for the whole query, whereas one set was required. Due to this duplication, in this plot we will look for patterns above nesting level 1, and outliers are the most interesting aspect. In the plot for query 2, we see a small blob of outliers for nesting level 4. As these are very close together and evenly spaced, this leads us to believe that this was the same student. Looking at the data shows that this is indeed the case, participant 143 created these data points with a very complex query. This writing of very complex queries seems to be constant throughout the data. The goal of the participants does not seem to be to solve the problem in the easiest way but to solve the problem at all. Their inexperience shows, which indicates that some more help in writing shorter queries is needed. Another interesting group of attempts is found at the start of query 5, nesting level 4. Eight of these attempts were made by participant 207, who started their problem solving with a high number of nested brackets. Looking at the query shows
that their solution is not as complex as the query by participant 143. On the contrary, their query is relatively easy but apparently did not work as the participant expected. The student tried various combinations of brackets around their concepts until they arrived at a working query that produced a result table. This again shows that the student does not have enough understanding, confidence or patience to look at the code and find the error themselves.

The nested negation overview in Figure 5.36 is of a different scale. Negation does occur regularly in the queries but was never nested more than two levels. However, some interesting things are still visible in these plots, especially for the queries where the correct answers involved one or two levels of negation. In all the plots, we see an overwhelming lack of negation used by the students. Only query 6, at the end of the session, shows a decrease of queries with 0 negation. There also is a visible pattern between the horizontal lines in the plots: the bottom ones are darkest, the middle ones lighter and the top (nested negation level 2) is almost non-existent, except for query 1 and 3. These required two negations for the correct answer, so that makes sense. In general, it does seem that students are hesitant to use negation.

5.6.6 Outliers

We already discussed some outliers in the discussion of plots we found before. However, some other plots were given limits on the y-axis to discard the outliers and show clearer patterns. The analysis of some of these outliers will be done here.

Running time We start with running time. First, we calculated three standard deviations from the mean, to find the running time that would be the upper limit for ‘normal’ running times. This turned out to be 1.777 seconds. Then, we filtered all running times to retrieve the query name and participant number of all the queries that took longer than this time limit. Through this method, we found 69 outliers by 24 participants. A summary of the numbers we found is

Figure 5.32: Complexity metrics for participant 320 on query 6.
Figure 5.33: Density overlay plots for the total number of brackets used by participants.
Figure 5.34: Density overlay plots for the maximum number of nested brackets used by participants.
Table 5.5: Summary of the metrics of the running time outliers.

<table>
<thead>
<tr>
<th>query</th>
<th>query 2</th>
<th>query 3</th>
<th>query 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 sets, 23 outliers</td>
<td>4 sets, 5 outliers</td>
<td>2 sets, 10 outliers</td>
<td>6 sets, 11 outliers</td>
</tr>
<tr>
<td>min</td>
<td>max</td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>correct mean</td>
<td>mean</td>
<td>correct mean</td>
<td>mean</td>
</tr>
<tr>
<td>0.08</td>
<td>4.522</td>
<td>0.01</td>
<td>3.197</td>
</tr>
<tr>
<td>query 5</td>
<td>query 6</td>
<td>others</td>
<td></td>
</tr>
<tr>
<td>4 sets, 9 outliers</td>
<td>2 sets, 3 outliers</td>
<td>2 sets, 8 outliers</td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>max</td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>1.819</td>
<td>2.472</td>
<td>3.786</td>
<td>12.329</td>
</tr>
<tr>
<td>correct mean</td>
<td>mean</td>
<td>correct mean</td>
<td>mean</td>
</tr>
<tr>
<td>0.0097</td>
<td>2.215</td>
<td>0.008</td>
<td>8.428</td>
</tr>
</tbody>
</table>

available in Table 5.5. Two participants had created outliers on two different questions, leading to a counting total of 26 sets of outliers in the table.

The first observation to make from the table is the division of outliers over the different questions. We can see from the second row in the table that a pattern can be found that shows that question 1 and 4 had the most outliers in sets, and query 1 had most outliers overall. Comparing this to the means of the correct answers, only question one should take significantly (up to ten times) longer than the others. This longer running is reflected in the outliers of the students. On the other hand, the longest mean running time in the outliers occurs in query 6, with an average of 8.5 seconds. Comparing this to the mean running time of the correct answer, we see that this question should actually have taken the least time of all. As this set is the smallest of the seven, it might not be representative. Also, this set does not contain the highest outlier overall, which is 14.6 seconds and was found for query 4. This question also contained the second highest amount of outliers, over the biggest set of participants. Thus, we can assume that students thought query 4 was complex. This assumption is supported by a boxplot of attempts (Figure 5.35), which shows that query 4 contains the highest median number of attempts of all queries.

Query 1 having most outliers can be explained in a simple way. As we already mentioned, the correct answers to query 1 showed that the query took the longest to run. On top of that, we discussed the relation between running time and the size of the result table. Query 1 was the first question in the homework that had the students interact with a real SQL database. The question was open and the exploration of the database might have led some students to ask for a print of a big part of the database. As the database is of considerable size, this will lead to longer running times for this first question. We should keep in mind that this type of exploration is very useful to the database users, and may result in more efficient querying in the long run. If there would be a way to exclude these more exploratory runs from the records (excluding participants querying and printing a full table), the results will be more reflective of the query-specific process.

It is also interesting to look at the distribution of outliers over the participants. We found that the number of participants raising 1 outlier is the highest, with the number of participants creating 5, 7 and 12 outliers, respectively, is the lowest. This shows that often, writing an incorrect query that takes so long is a one-off experience for most students. It also makes sense that students would like to solve such issues quickly, as long running times have an effect on the speed of problem-solving. When one writes a query that takes more than ten seconds, this might inhibit the students' flow.

Finally, we found a pattern within the sets of outliers made by different participants. Some of the sets were very diverse, participant 329 on query 1 ran three different queries with running times of 1.974, 11.566 and 3.227. But overall, most sets were very similar, especially those with running times between two and three seconds. For example: participant 121 on query 4 had the outlier set of [2.400, 2.428, 2.392, 2.628]. For participant 207 on query 5 we found [2.470, 2.388, 2.379]. From the original data, we can see that these outliers were found in (technically) consecutive query runs.
CHAPTER 5. HUMAN-DATABASE INTERACTION IN PRACTICE

This again points us to the hypothesis that students keep rerunning erroneous and slow queries as they do not understand what is going on.

In summary, we found two patterns in the running time outliers: most outliers were made in query 1, and queries with relatively low outliers (less than three seconds) were often repeated by the participant.

Error fraction We move on to the outliers in the erroneous queries written by the participants. In an earlier paragraph, we discussed the error fractions, which relate the number of errors to the total number of attempts. This is a fairer comparison than just looking at the number of errors someone made, as a participant with 15 errors in 20 attempts performed much worse than a participant with 15 errors in 100 attempts. As an outlier, we took the mean plus two standard deviations. We first looked at using three standard deviations, but this led to a boundary of approximately 0.97. From a practical standpoint, this fraction is way too high. A programmer with 97 percent erroneous queries would not be able to function appropriately. Thus, we went with two standard deviations, which resulted in a boundary of 0.729. This led to 24 student queries being regarded as outliers on the error fraction aspect. From these 24, 6 students were outliers on more than one query.

Table 5.6: Summary of the error fraction outliers.

<table>
<thead>
<tr>
<th>query 1</th>
<th>query 2</th>
<th>query 3</th>
<th>query 4</th>
<th>query 5</th>
<th>query 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

When we count the distribution of the outliers over the queries in Table 5.6, we see a decreasing trend. Most outliers occur in query 1, and then decrease when approaching query 6. From the trace plots made earlier in this chapter, we know that most students started on the queries in linear order, that is to say, as in the order present in the homework. This means that most students started their query practice with query 1. As with the running time outliers, most error fraction outliers also occur in query 1. This, combined with the fact that they have low- to no experience with SQL to begin with, might point out a need for the students to jog their memory and get some practice. The reducing pattern from query 2 through 6 can also be found in the lowering whisker of the plotted data in Figure 5.27.

Number of attempts Another observation that we made earlier on, based on the traces plot, is the variance in the number of attempts made by every participant. There are many queries that only had one attempt, but there were also students that made up to 1500 attempts in total, over all queries. Here we dive into the number of attempts as divided over the queries, to see whether patterns arise.

Table 5.7: Summary of the attempts outliers.

<table>
<thead>
<tr>
<th>query 1</th>
<th>query 2</th>
<th>query 3</th>
<th>query 4</th>
<th>query 5</th>
<th>query 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

For the outlier boundary, we chose the mean plus two times standard deviation again. This lead to the number 71.2, or 71 attempts. We found 13 instances of outliers on this feature, with three participants generating two outliers. Most of the sets were between 71 and 100 attempts, with four sets exceeding 100, and one extreme outlier of 301 attempts for query 2. These four sets were created by two participants, taking the blame for two sets each: query 1, 2x query 2 and query 6. Although no real pattern is visible in Table 5.7, we do see that query 2 is higher than the others. As they also include two of the extreme outliers, we might say that query 2 was complex in the sense that it required a lot of attempts.

The participants To round up this analysis of the outliers, we checked whether any participants made outliers in more than one category. If there is a small group of students creating all the outliers, catering to this group does not help novice users in general. In total, there were 104 participants who sent in logs. From these 104, 42 participants were responsible for producing...
outliers in the three categories discussed above. This forty percent seems like a big enough group to confirm that the issues discussed above are representative of errors that advanced beginners make. There were some participants who created outliers in multiple categories. Two students made outliers on all three categories. Six students had outliers in two categories, the combination of error fraction and running time occurred most frequently (four times).

5.6.7 Other remarks

The query process also depends on the amount of time the student had to finish the exercises, but in this dataset, we do not have any timestamps available. A related metric is the number of attempts a student used, and the trace that shows the order in which the students attempted each query.

5.6.8 Query complexity summary

In this section, we analyzed the complexity of the queries submitted by the participants. We found various patterns in the data, which gives us insight into the problem-solving process that the participants went through. Our main finding in this section is that students prioritize solving the problem over solving it efficiently. We have seen cases of students brute-forcing errors through a random addition of brackets, without them looking at their code and thinking about why they were getting error messages. Similar behavior is the repeated execution of ‘bad’ queries, where running time is high or they receive errors. Repeatedly executing the same query does not lead to different results, but they did it anyway. Their queries are often overly complex with high numbers of brackets and complexity scores far beyond those of the correct answers. These findings again show that the students do not really know what they are doing.

Another important finding is that students hardly ever use negation, even when the solutions suggest its use. The usage of negation is relatively complex, and may thus be too hard for these advanced beginners to fully understand. The fact that they evaded the concept and often did not even try to use it, may suggest a fear of negation.

Figure 5.35: Boxplot of number of attempts made per query.
Figure 5.36: Density overlay plots for the total number of nested negation used by participants.
5.7 Conclusion

In this chapter, we examined the interaction between users and existing database systems. From each of the aspects examined above, we found proof of ongoing issues in the existing systems. First of all, we found that students have widely different ways of working. Some are very neat, some make a mess of their homework. Some work through the problems in an organized way, whereas others’ attempts are all over the place. In the first analysis, we also found a correlation between self-efficacy and homework score. This may help us predict the success of the students in their interaction with the database.

Furthermore, in the introduction of this chapter, we wrote about the complexity of using traditional databases via SQL. Mitrovic (1998) discussed the high cognitive load and its potential causes in memorizing a database schema. We found proof of this statement in the recurring theme of exploratory queries posed by the students. Repeatedly having to execute these queries hinders the participants in their flow of solving the problems posed to them, which in turn reduces the usability of a system. Displaying the database schema within the interface might thus increase the accessibility of the system for (novice) users.

In the raw text analysis, we found that a majority of the students wrote queries that were much more complex than required. The way in which some of these queries were written indicates that some students have no idea what they are doing. Then there are other students who do work in a structured manner, and systematically build their queries. The changes in the result table point them to the next step in the process of problem-solving. Other helpful actions that we found were the use of helper queries and exploratory queries related to the question itself (such as finding all distinct stores). Students helping each other by sharing queries is helpful for them in the short run but hinders learning in the long run.

During the complexity analysis, we found proof of students repeatedly executing the same query. They should know that this will not change the outcome, but it seems they did sometimes expect different results. This holds mostly for queries with a long running time or those that resulted in a database error. We also found that students seem to have avoided negation. As this concept is relatively complex, it makes sense that students would be hesitant. It might be possible to increase their confidence by solving more problems before moving on to problems requiring negation.

Overall, we found that query 1 was an outlier in most aspects. Unfortunately, this might be due to the set-up of the experiment. When users come into contact with a database at first, they will want to run some exploratory queries to get used to the database schema. On top of that, especially for novice users, we expect them to need some time to refresh their memory on SQL. They will need to remember the precise syntax and how it works, as well as getting used to the column- and table names. This may have led to a large number of outliers on query 1. As we have a hypothesis as to why this occurred, it is something that we can improve upon in the second iteration of this experiment.

We have now generated a set of insights into the behavior of users on a real-world database. Their interaction with the system is representative of the behavior of advanced beginners. Thus, we can now use these insights to improve the interaction between humans and databases. We will take these insights into the next chapter, where we will choose a subset to apply in our interaction design.
Chapter 6

System Design

In Chapter 3, we discussed various papers on the topic of visualization and interaction design. In this chapter, we will touch upon those points again and work out how this influences the design of our system. Additionally, we answer four subquestions in this chapter:

- What is an intuitive way to visualize database searches?
- What does an ideal database interface look like for a novice user?
- How do (novice and expert) users interact with this new system?
- Is the new system an improvement on existing systems?

6.1 Problem analysis

From the experiment done in Chapter 5, we found a list of problems that commonly occurred. Looking at the questionnaire results, we can conclude that our population is representative of the group of advanced beginners. The only factor that was not evenly distributed in the population was age. The types of errors made by these participants reflect the fact that they are new to SQL and do not have enough experience to write concise queries.

This problem of novelty could be relieved by supporting these novice users in multiple ways through visualizations and annotations. This section will describe the development of an interactive database querying tool that works through SQL and supplies a visual graph representation of the query as well.

6.1.1 Stakeholders

There is a single stakeholder, who is the direct user. This user will benefit from the proposed solution as it should help them simplify the querying process. As we noticed in Section 5.4.4, the users have very different characteristics such as experience and age. The system is designed specifically for advanced beginners, but can be used by more expert users too. Users of all levels of experience should feel comfortable using this system. The users also may have very different ways of working, which should all be supported by the system. In addition, there may be effects of age or gender.

6.1.2 Questions

Our main goal in this thesis is to facilitate interaction between novice users and databases. This will enable them to more effectively use query languages, and pick up on concepts more quickly.

In the introduction we already hypothesized that adding a visualization would be a helpful tool for novices, as it can support them in analyzing results and understanding why problems occur.
There are many types of additional improvements that can be made to support the user in their querying process. For instance, we could pinpoint the location within the query where issues arise. We could add interaction so the user can rearrange the graph. We could allow the user to compare queries. We could allow the user to combine (sub)queries. On the more technical side, we could improve the interaction by increasing the performance of the system.

In the design process that we are describing in this section, we do not yet take these additional changes into account. Are these system conditions could be very useful, but in order to measure their effects we should only apply one significant change at a time. Therefore, we leave these additions to later iterations of the system.

6.1.3 User tasks

There are two sides to the system that we are designing here. The first is the overall system, the interface, and its capabilities. Then, separately, there is the design of the graphical query representation.

We will start by dividing the visual tasks of the interface into categories according to the topology of tasks written by (Munzner, 2014b).

1. Analyze (consume and produce)
   (a) Present a graph representation of the query at the press of a button
   (b) Present the result table containing the outcome of the query
   (c) Discover the relations between tables within a query
   (d) Derive answers from the result tables
   (e) Derive new queries from subqueries

2. Search (returns targets)
   (a) Explore the data model
   (b) Browse the visual representation by dragging and dropping
   (c) Browse the contents of the tables in the data

3. Query (content of the targets)
   (a) Identify the characteristics of the result
   (b) Identify mistakes through error messages
   (c) Compare the variables in the result
   (d) Compare two or more query representations
   (e) Compare whether the current query is the same as the previous query?

On top of these visual tasks, there are some technical aspects that we could not express in the taxonomy by Munzner (2014b). These include the actions of writing a query. As the goal of the system is to support novice users, most of the advanced SQL commands are not needed by them. Thus, for this design, we will focus on basic SQL functionality, subqueries and the usage of negation. This is because the students in the experiment in Chapter 5 seemed to have avoided negation, even when it was required for solving the posed problem.

6.1.4 Requirements

The two main requirement for the system are the realization of the set of user tasks listed above and the technical setup for running SQL queries on a database. On top of that, there are some implicit requirements. First of all, in our study in the previous chapter, we found that various students write helper-queries and exploratory queries to aid them in solving the problem. If we would only display one graph translation of the query, the user would have to memorize their helper query. It would thus be useful if users could ‘stick’ their graph representations and start new ones as well. This interaction needs additional buttons.

Similarly, we found that students have widely different ways of working. Some users have been working through the problems in a structural manner, whereas others had a more haphazard approach of randomly attempting any query. The system should be able to support all of its users, independent of working order or query-writing method. In this scenario, it would again be
helpful to store or save older graph representations, such that users returning to a certain query can retrieve their (latest) query representation.

We also found that users repeatedly ran the same query. As the result should not change when the query is the same, this behavior is a waste of both time and resources. The system should be aware of this practice and post feedback when the user uses this approach.

Finally, we have found that these novice users make many mistakes in the writing process. The system should be able to intuitively indicate that an error has occurred. Preferably, it will also indicate an approximate location for the error.

On top of these functional requirements, it is important to keep the system light. As we design the system for novice users, it would be best if it does not require installation. Creating the system as a web page allows high access as it is usable on most common PCs.

6.1.5 Use case

In this section, we will discuss the typical walk-through of the user in the system.

1. The system contains a cursor in a text field.
2. The user types a (sub)query into the text field (allows data creation, querying and exploration options)
3. The user executes the query by pressing a button
4. The result table is displayed as a result.
5. The user asks for a visual representation of the query by pressing a button.
6. The system displays the visual representation.
7. The user returns to step 2

Alternative courses:

1. If the user wants to ask for help (at any point during the usage of the system)
   1.1 The user clicks a Help button.
   1.2 The system displays a window with an example query and the option to search for keywords
   1.3 The user types in the keyword that they want to know more about.
   1.4 The system displays information on the keyword.
   1.5 The user closes the help window.
   1.6 The system returns to step 1.

4. If the user caused an error.
   4.1 The system displays an error message.
   4.2 The user finds the error.
   4.3 The user returns to step 2.

7. If the user wants to write two subqueries and merge them.
   7.1 The user clicks a button to pin the visual representation and its corresponding query.
   7.2 The system displays an empty box to submit a new query.
   7.3 The user types in their second subquery.
   7.4 The user executes the query by pressing a button.
   7.5 The result table is displayed as a result.
   7.6 The user asks for a visual representation of the query by pressing a button.
   7.7 The system displays the visual representation.
   7.8 The user presses a button to merge the two subqueries.
   7.9 The system prompts the user for the required method of joining.
   7.10 The user confirms the method of joining.
   7.11 The system creates a new box in which it displays the new query and its new visual representation.
   7.12 The user can return to step 2.
6.2 User-Centered Design

After the establishment of the technical characteristics of the design, we will now take a look at the human aspects of the design.

The design process in this project is guided by the principles of the user-centered design paradigm as first described by Norman and Draper (1986). This means that, from the start of the prototyping process, stakeholders are involved in the design, and drive the design through the analysis of their requirements (Marchionini, 1992). One key concept in this context is that of mental models (Ju & Gluck, 2005). The functional requirements and the technical implementations should be connected by a representation that any user can understand.

We apply this paradigm by recording interactions with the prototypes, as well as various measures such as questionnaires, such that feedback from users can directly be applied to the prototypes. This is an iterative process, where interactions influence the prototypes, and prototypes influence the interactions. This cycle will be applied to various components of the tool development.

In practice, this means that we will be running user tests for all steps of the interface- and interaction design. In the second half of this chapter, we describe the experiments that were undertaken, their results, and the effects of these results on the subsequent prototypes.

6.3 Visual Query representation

We start this design process with the key element of our new tool: the visual representation of the query. The goal of this visualization is to create more insight into the SQL text, such that users make fewer mistakes.

Although there has been some research on the visualization of graph data (which we elaborated on in Section 3.2), the visual representation of queries did not get much attention. As our goal is to simplify the interaction, we want to see users’ intuitive representation of a query. With such an approach, the design of this element can be done in collaboration with future users.

Requiring the users to intuitively visualize all details available to us in the SQL language would be too big a task, so we decided to focus on negation and subqueries. These are two components which we have shown the students struggled with in Chapter 5.

Besides the feedback from the users, we also wanted to base our design on theory. Thus, in the next section, we first analyze some existing research on the visualization of negation and subqueries.

6.3.1 Literature

Subqueries can be difficult to understand due to their intrinsic nested nature. Queries can contain many levels of nesting, making them hard to write, but even harder to understand. For the visual representation of subqueries, there is a clear-cut answer: delimiting each level of nesting and each subquery within that with a box, nesting the boxes as the queries are nested. This is supported by literature, such as Thalheim (2003). Another method is to represent nesting through directed arrows, which indicate a reading order, as in Gatterbauer (2011).

The question of how to represent negation is much less clear-cut. Is it relatively easy to show the presence of a concept, but how do you show something that must not be there? Various papers have considered this issue. Some researchers suggest the use of complementary colors, distinguishing between existing and not existing (Kindlmann & Scheidegger, 2014). Others suggest different borders for different types of complex operations (including conjunction and disjunction) (Boley, n.d.; Gatterbauer, 2011). Negation can also be expressed in a more textual manner, by using mathematical symbols where one expresses the conditions.
6.3.2 Methodology

With the execution of this user test we aim to answer the subquestion:

What is an intuitive way to visualize database searches?

Participants and design

There were five participants, all Ph.D. students at the Database group in the Department of Mathematics and Computer Science at the Eindhoven University of Technology.

The main idea behind this thesis is that a well thought out visualization can support novice users in understanding SQL. Previously we found that novice users regularly memorized the structures that they need to write, without understanding what the queries mean. Thus, asking novice users to draw their intuitive representations of several queries will most likely not yield the results we are looking for. Therefore, we asked more experienced database users to participate in this experiment.

To analyze these two aspects, we created four queries. These queries, plus their corresponding data model and instructions, were printed and handed to the participants. They were then asked to visualize the four queries in the way that they found most intuitive. Once they were done, the visualizations were sent to the experimenter by email.

Materials

We presented the participants with a supermarket database schema (Table 6.1). Additionally, we gave them four queries that increased in complexity: we started with a basic query (Query 6.1), then added a subquery (Query 6.2), and finally added levels of negation (Query 6.3, Query 6.4).

Tables

<table>
<thead>
<tr>
<th>Tables</th>
<th>Data Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer (cID, cName, street, city)</td>
<td></td>
</tr>
<tr>
<td>store (sID, sName, street, city)</td>
<td></td>
</tr>
<tr>
<td>product (pID, pName, suffix)</td>
<td></td>
</tr>
<tr>
<td>shoppinglist (cID, pID, quantity, date)</td>
<td></td>
</tr>
<tr>
<td>purchase (tID, cID, sID, pID, date, quantity, price)</td>
<td></td>
</tr>
<tr>
<td>inventory (sID, pID, date, quantity, unit-price)</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Datamodel for the Supermarket database.

```
SELECT c.cID, c.cName
FROM customer AS c
WHERE city = Eindhoven
```

Query 6.1: The IDs and names of customers from Eindhoven.

```
SELECT c.cName
FROM customer AS c
WHERE EXISTS (
    SELECT *
    FROM purchase AS p
    WHERE p.cID = c.cID
)
```

Query 6.2: The names of customers that have made a purchase.
CHAPTER 6. SYSTEM DESIGN

SELECT c.cName
FROM customer AS c
WHERE NOT EXISTS (SELECT *
FROM shoppinglist AS s
WHERE s.cID = c.cID)
)

Query 6.3: All customers that do not have a shopping list.

SELECT s.sID, s.sName
FROM store AS s
WHERE NOT EXISTS (SELECT p.pID
FROM product AS p
WHERE NOT EXISTS (SELECT *
FROM inventory AS i
WHERE s.sID = i.sID
AND i.pID = p.pID)
)
)

Query 6.4: Stores that have every product in inventory.

Procedure

The materials and short instructions were sent to the participants by email. The participants were instructed to draw a visualization that they found intuitive. We did not explicitly mention graphs as an option, as this might bias the participants.

They could draw the visualization through any medium they preferred. Then, the images were returned to the experimenter by email.

6.3.3 Results

The results we retrieved were widely varying, and can be found in Appendix B. Two of the five participants drew something graph-like. Four out of five participants used arrows and wrote the appropriate conditions near these arrows, as we did in our first prototype.

Of the ones not using graph-like representations, one participant represented the query through the result tables. One participant represented the queries with flow-charts that specified the execution plans. One participant represented the queries with pseudo-code and logic.

The fact that two of the five participants drew graph-like representations when asked for the most intuitive visualization (without mentioning graphs), supports our hypothesis that a graph representation is an intuitive characterization of a query. Four out of five participants drew details similar to our first prototype, supporting the accuracy of our initial drawings.

• Participant 1: Represents the query by showing the intermittent and final result tables. Connected by arrows which contain the constraints, in that sense similar to the prototype.
• Participant 2: Similar to a flowchart, sort of execution plan. Connected by arrows with constraints.
• Participant 3: Graph-like representation, with squares surrounding subqueries.
• Participant 4: Pseudo-code/logic representation.
• Participant 5: Cross between flowchart and graph-like representation, including result table.

Within the queries listed above, we focused on the two relatively difficult topics of negation and subqueries. Visualization of these items is not straight-forward, so we will take a look at how the participants represented these.

Subqueries

Queries 2, 3 and 4 of this experiment contained subqueries, indicated with round brackets.

Unfortunately, the drawings were of such different types that some of them do not distinguish between the main query and subqueries. Three of the five participants did not distinguish explicitly between subqueries. The other two used boxes drawn around the subqueries to separate the parts. This is similar to what we found in the literature. Interestingly, due to their nature, flowcharts apply the other method of distinction that we found in literature: arrows for reading order. Thus, we found support for both methods of subquery representation.
As support for our visualization as graph-based still stands, we feel that the method of using arrows is confusing when also using edges. Thus, we decided to visualize subqueries with thinly bordered boxes.

Negation

Query 3 and 4 in this experiment focused on negation, through the usage of the WHERE NOT EXISTS predicate in SQL.

Some participants used their knowledge of mathematics and logic and represented the negation through symbols, such as $\notin$ and $\neg \exists$. The participants that used flow-chart representations either put in blocks with “NOT EXISTS”, or posed the existence as a question. The fifth participant represented the negation by drawing the arrow for the condition and then drawing a cross through it.

Thus, we have found five different approaches. The participants were not unanimous, so selecting a preferred method from these results is difficult. Considering the knowledge level of novice users, we discarded the usage of $\notin$ and $\neg \exists$ as they were too complex. We judged the crossed out lines as somewhat crude and possibly distracting. Thinking back to our findings in the literature described above, we decided to go with colored lines to indicate meaning. Negation, as defined in the subqueries we discussed above, was part of a subquery, so this method also combined the two aspects of subqueries and negations well.

6.3.4 Discussion

In this section we discussed the following sub-question:

What is an intuitive way to visualize database searches?

One weakness of the approach is that the digital manner of experimenting meant a lack of interaction with the participant. Thus, not everyone may have understood correctly what the goal was. On top of that, we do not know the reasons why the participants drew these visualizations. What are their ideas, intuitions, and preferences for this visualization? However, this weakness was not a major problem, as we did not want to personalize the visualization to every user. We wanted to make a well thought out visualization that works for most people, and the findings in this experiment helped consolidate what the most important components of the visualization should look like.

The strength of this approach is that there were no bounds as to what the participant may send in. The differing backgrounds and focuses of the participants lead to different approaches, showing their preferences for visualizations.

We found that all participants returned widely varying results. Although we did not show it to the participants, we did have some ideas as to how to visualize the query. Our method was supported by various details of drawings of the different participants combined. Thus we conclude that a good way to visualize a query is to draw it as a graph, with the addition of various details.

6.4 Interface Design - Prototype 1

For the design of the first prototype, we take into account the results of the database experiment as described in Section 5.7. As there is no real interface to Jupyter, our first focus is on functionality. The design is guided by the golden rules by Shneiderman and the Gestalt rules as described in Section 3.2.4. We focus on symmetry, consistency, and cleanliness, and this first design does not include any colors.

The first design consideration is what functionalities to include. As is the case in Jupyter: We need basics such as a text area and an area for the results. Buttons are required for all functionalities, such as the execution of queries. We do not want to use a shortcut for it, as is done in Jupiter.
To simplify the query process we add three functionalities: An elaborate help function, a display of the data model, and a visualization of the query. On top of that, we add the functionality to temporarily store a query, which may speed up the workflow. This is done by clicking the thumb tack in the visualization area. At this point, we are not yet sure about the placement of the buttons. Thus, we take into account Fitts’ law and place the buttons as close to their applicable area as possible. The history area is placed at the top of the screen, to match with mental models on tabs. The design is made as symmetrical as possible and can be seen in Figure 6.1.

6.5 Paper prototype experiment

With the paper prototype designs and the results of the visualization user tests analyzed, we could start the first round of user testing. There were two parts to this first experiment: asking the users to draw their preferred interface, as well as having these users interact with a paper prototype made by the experimenter. In this section, we describe the methodology and outcomes of the study.

6.5.1 Methodology

Through this study we aim to answer the following sub question:

What does an ideal database interface look like for a novice user?

Participants and design

For this experiment, we recruited students of various ages with basic SQL knowledge. There were three participants, two female and one male. They had all used SQL for some time, either within
a university course or within another project. None of the participants were very secure in their abilities to write a SQL query from scratch.

This study was of the qualitative and exploratory type, so no variables are measured in the typical sense of the word. All sessions were video-recorded and transcribed afterward. The drawings made by the participants could be compared to the prototype as designed by the experimenter.

Materials

As mentioned before, all participants’ interactions were video-recorded. The room was set up as in Figure 6.2. On the short side of the table, the participant was seated. As all participants were right-handed, the camera was placed on their left, angled down towards the table. In front of the participant were the papers, first an empty paper to draw on and instructions to their right. Later, the experimenter’s prototype on the left and the instructions on the right. To the right of the participant, the experimenter was seated. They sat behind their computer with all instructions readily available.

The rooms were closed off, with the participants sitting with their backs to the door, facing one or two windows.

Figure 6.2: The layout of the room during the experiment. A: video camera, B: participant, C: experimenter, D: outer windows.

Procedure

This paper prototype experiment took approximately thirty minutes and contained two parts. In the first part of the study, the participants received a list of requirements for a database interface and drew their own interpretation of this interface. The participants were asked to think aloud, while the experimenter prompted them to discuss the arguments behind the item placement.

The instructions for the drawing were as follows: *Draw your ideal interface containing at least:*

- An area to write a query in SQL
- A table for displaying query results
- An area to visualize the query
- A button to run the query
- A button to generate the visualization
- A help-button
• A button to temporarily store the query
• A button to save the query
• Anything else you might want

In the second part of the experiment, a prepared prototype was shown to the participants. Then, a paper prototype user test commenced. The participants received a list of tasks to go through, all the while interacting with the prototype.

1. Write a query in the appropriate field
2. Execute the query
3. Visualize the query
4. Store this query in minimized form
5. Request the data model
6. Write a new query
7. Visualize the query
8. Execute the query
9. Do you want to try out some more?

Then, the participants were asked to compare the prototypes. Which aspects of either prototype did the participants prefer, which aspects did they not like? Was anything missing in the interface?

They were also asked their opinions on specific elements of the interface. These included the query visualization and the various options for storing queries.

The participants were free to ask any further questions if they wished to do so.

6.5.2 Results

The drawings by the participants can be found in Section C.1. A list of bullet point findings is compiled in the second part of Appendix C. We will now discuss these findings in the context of existing literature on interface design.

**Button placement:** The participants are unanimous about the characteristics of the help button. It needs to be easy to find, and close to the query area. The same principle holds for the other buttons: they need to be close to the items that they apply to, such as the button to run the query. We can relate this to the theory in Section 4.5.2, where we discussed Decision Making. Having the buttons for the appropriate actions nearby makes it easier for the user to decide on the correct course of action. On top of that, according to Fitts’ Law, there is a relation between target size, distance and the time to reach this target (Fitts & Peterson, 1964). We can use this law to our advantage when placing the buttons, increasing the speed of the interaction in case users want that. As Marchionini (1992) found, users solving a problem want to do it fast.

Participants 1 and 2 found a linear order in the querying process, and thus represented this order in their system by alternating areas and buttons in the order that they thought they would be used. Similarly, they found order in the buttons store and save. Participant 2 said “Save comes at the end of everything,” and participant 1 said “Store to the left of save, as temporarily stored means you still want to do work on it, whereas saving insinuates it is the final product”. These findings are in line with the western way of reading from left to right, such that as a consequence, items on the left come first.

The actual button placement in the drawn prototypes was very different for the three participants. Participant 1 drew the buttons in a row and right-aligned them. Participant 2 scattered the buttons but had them both left- and right-aligned. Participant 3 added the buttons to menus.

On top of the required buttons, participant 3 also added a button to validate the query. This might be a helpful addition to the system.

**Ordering and placement:** Although the drawings of the participants were far from equal, none of them stands out or is unexpected. All three wanted the most important item to be first, and they all decided that this central item was the area to write the query in. Then, the screen was divided in half horizontally or vertically, with the one half containing either all text (query and results) or all ‘core’ items (query and visualization) and the other half containing the remaining items. This ‘most important’ half was always placed on the left or top division, the first thing
to see following our reading direction. In their own way, each of the participants used these separations as a means of guiding Attention. As we discussed in Section 4.5.1, Attention as a process is a combination of selective attention and dividing attention. The salience of putting an aspect ‘first’, as well as making the aspects distinct, leads to more attention being paid to this aspect. Having the user pay attention to the correct aspect at the correct time can help increase the speed of the interaction, as well as making sure the user does not have a negative experience due to interruptions (Bailey & Konstan, 2006). On top of this, there is the research by Martin-Michiellot and Mendelsohn (2001), who found that two conceptually connected materials (such as the query and the visualization) should not be displayed in a physically separate way, as this inhibits the learning process.

Only one of the participants drew space for the pinned queries to be stored, but as this was not in the instructions, there was no problem with that. When asked where in their prototype the users would incorporate this, the overall preference was to the side. Notable is the decision of participant 2 to draw an interface in portrait mode (although they only used part of the paper). This was directed by their previous experience with Jupyter.

**Additions and changes to the prototype:** All participants interacted successfully with the proposed prototype. One issue was found, which might be due to the formulation of the instructions: When asked to “store this query in minimized form”, two out of three participants did not find the thumb tack button that would activate this functionality. Participant 1 noted: “[The thumb tack] wasn’t necessarily intuitive, because I might not have understood what minimizing meant, so pinning was not what I thought of.” All participants agreed that the button was small and did not stand out. It should at least have a border and shadows, and the placement might not be optimal as it seems part of the visualization to some. Attention again plays a role in this aspect. Putting the button on the right-hand side and adding a border should increase visibility.

For the help-button, participant 3 suggested making it small, with an i or question mark on it. This is supported by the research on mental models and metaphors (described in Section 4.5.1), as these are symbols that reflect the help-functionality without spelling it out in words. They are a form of cognitive support that allows users to correctly predict the outcome of their actions (Wickens et al., 2014). On top of that, many systems already use these symbols, and thus users are quickly able to understand what they mean.

As we already saw in the paragraph on button placements, all participants agree that the buttons should be placed close to the areas that they correspond to. Some discussion was had about the Merge and Data model buttons. The Data model does not concern the queries directly but is more a general set of info on the loaded database. Thus, it could be put in a menu or any other place further from the query. For the Merge button, there were multiple points of view on what it relates to. As no two queries are in focus at the same time, it uses at least one, sometimes two, pinned queries. Thus, it would make sense to have Merge close to both the query area and the pinned queries.

The three participants were unanimous on the storing of the pinned items. They all thought the tabs with query snippets could become too confusing. The pins with visualizations included too much information. The named tabs were liked: “It is nice and simple.” (participant 3). This reflects one of the central goals for data display (Smith & Mosier, 1986), minimal memory load on the user. The principle of minimizing cognitive load is also supported by Marchionini (1992) and Epps (2017). In contrast to these findings, participant 3 had some reservations about the named tabs option as it does not leave much space for encoding additional information such as the number of records returned by a query. They also suggested that horizontal scrolling through lots of records might be confusing, so they did prefer their own (vertical) history mechanism for the pinned queries.

Another point of feedback was for the line of buttons in the original prototype. Participant 2 mentioned a dislike for the fact that all buttons have the same size, and thus the same importance, although they did admit that this made it easier to find the functionality that they were looking for. The underlying thought for making these buttons the same is to remain consistent within the application, as this reduces the mental load on the user. Details on this mechanism can be found in Section 4.3.3.
Positive feedback: From the remarks overall, we can say that the participants liked the system. Participant 3 states: “I like the fact that you have all these options, quite clear there on the screen” and “I like the fact that you're thinking to keep things simple and neat.”

Two functionalities stood out positively for the participants: the pinning of queries and the interactive visualization. As participant 1 states: “The pinning, that's the biggest asset”. This is relative to their experience with an existing system that allowed for typing many queries but then left the user to their own devices trying to find the query they were looking for again. Excessive scrolling is no longer necessary in this system, as the queries can be named and stored neatly. Here, again, we know that making things easy to find means fewer interruptions for the user, leading to a more positive user experience (Bailey & Konstan, 2006).

The visualization was understandable to the participants. They thought it was useful and nice, especially when it is interactive. According to participant 3, visual feedback can help in decision making, as well as to drive hypotheses. Participant 2 elaborated on why it was helpful: “I feel like the visualization helps. Me, and a lot of people, while we were learning about queries, had a hard time understanding exactly what it is and what they look like. The visualization really helps with that. Such as with how the connection goes, and how it works, basically. I think this is a really helpful way to do it.”

In conclusion, the participants’ feedback directly supports our hypothesis that novice users can be supported in their interaction through visual representation.

6.5.3 Discussion

In this section we discussed the following sub-question:

What does an ideal database interface look like for a novice user?

The usage of the paper prototyping approach was very valuable in this stage of the research. Earlier in this chapter, we discussed the principle of User-Centered Design. Involving the user in this stage of the design process, providing early feedback, helps to find issues even before a first implemented prototype is available. The qualitative approach has as an advantage that many quotes by the participants are reported, providing solid proof of how well the prototype was received. This has preference over using a Likert scale here, as it does not require the interpretation of numbers and the assumptions that go with it.

The choice of study also had drawbacks. The interpretation of spoken text, including the transcription of it, was very time-consuming. Another drawback was due to the text-based nature of SQL. This means it was not very well suited for paper prototyping in the traditional way. Normally, in a user test, we would require the user to write a SQL query from scratch. However, as our prototype contributes by adding visualizations, drawing a visualization from scratch during the experiment was not feasible. Therefore, our solution was to have some pre-written queries with their corresponding visualizations and have the participants pick one and place it in the correct interface area. As this lessens the types of interactions that the user can have with the interface, it is not an optimal solution. However, it was a solution that allowed for early feedback from the user. This was more important to us than allowing the participants completely free interaction with the prototype. Besides, this made the experiment less daunting, as all participants were a bit hesitant about their SQL skills.

The prototypes drawn by the participants reflected their level of experience. The more novice users drew simple interfaces, the more advanced user drew more details and even came up with additional functionality.

It was interesting to see that the initial drawings by the participants reflected many of the components that were present in the first prototype, even though the participants had not seen it yet. This includes ordering and button placement.

In the next section we have extrapolated from this data and present an interface that should reflect the preferences of the users.
6.6 Interface Design - Prototype 2

Based on the feedback of the participants in the previous section, we did a redesign of the interface. The second prototype can be viewed in Figure 6.3. Some aspects stayed the same, while others changed drastically. Elements that may be recognized from the previous design includes the text area and visualization area. We will discuss this new design in separate pieces, starting with our main contributions.

Figure 6.3: The second, fully functional prototype.

6.6.1 Implementation

For ease of access we decided to implement the system as a web application. This means that the system contains a front-end, a back-end, and an API to communicate between the two. The choice of the programming language depended on the visualization options that it provided. One of the most well-known and powerful visualization libraries is d3.js\(^1\). JavaScript also offers the powerful framework Bootstrap\(^2\), which significantly simplifies the building of an interface. Therefore, we chose JavaScript as our front-end programming language.

However, due to security concerns, JavaScript is by design not allowed access to the local file system. As a database requires data, which may be stored locally instead of on the server, the back-end required a different programming language. A well-known language for server- and API implementation is Python. Coding of the RESTful API was supported through Flask\(^3\), a micro web framework in Python.

Through this combination of tools, we were able to implement the system according to the design described in the following sections.

6.6.2 Visualization area

One of our three main contributions is a visual representation of the query. As you can see in Figure 6.3 the placement is similar to that in the previous prototype. One participant mentioned

---

\(^1\)https://github.com/d3/d3/wiki/Gallery
\(^2\)https://getbootstrap.com
\(^3\)https://flask.palletsprojects.com/en/1.1.x/
the option of resizing the field but we wanted to keep this prototype more simple to check the
Effectiveness of the functionality.
Through the experiment in Section 6.3 we got a feeling for the preferred visualization method.
For the implementation of the visualization, we again focused on subqueries and negation. On top
of the graph structure as a representation, we added visual indicators for various other components
of the query.

This is a representation of a simple query. It
involves one table and thus has one node and no
links. The label of the node is the name of the
database table that it represents, in this case the
Customer table. If the query contains an alias for
the table, this will be represented on the node as
well with a slightly different text color.

Each node can be expanded by clicking it. This
will not only reveal details about the database
schema but also visually represent other compon-
ents of your query. In this example, we see orange
highlights for the columns on which you have ap-
plicated a selection on in the query. In this case, the
query contained SELECT *. Only top-level selec-
tions are highlighted, those within the subqueries
are not.

Each query may access more than one table. This
example shows the usage of two tables. The join
condition applied on the tables is represented by
a labeled link. Nodes that have a border are ele-
ments that are used in a subquery. This border
shows that the subquery is of the positive type,
no negation is involved.
CHAPTER 6. SYSTEM DESIGN

Subqueries can be nested for many levels. This query shows two nested subqueries. The dotted border, contrary to the previous example, indicates a negation for the subquery it contains.

Besides the highlighting for selected columns, the system can also display all conditions that are applied within the query. This is done with a green color as in this example. The values of the condition are also written down in the tables.

Sometimes, a table column may have both a selection and a condition. In that case, we nest the borders and display both colors.

Finally, a query may contain aggregation. In that case, we display the aggregation method within the selection column that it applies to.

The visualization can also be manipulated by the user, via a dragging interaction. For the colors of the visualization, we reuse those of the background and interface.
6.6.3 Database schema

As included in the first prototype, we implemented a button that manipulates a pop-up containing the data model. We decided to represent the schema in its technical manner, which is plaintext with underlinings for primary keys. Each table name is written down in boldface, with all its columns displayed below. For each column, we display the name and type between brackets. An excerpt is shown in Figure 6.4.

Another option would have been to display the scheme in a more visual way, for example through an ER-diagram. Such a representation would show links for any primary and foreign key. However, displaying these lines correctly can be difficult technically and can be confusing for users, especially for large databases with many tables.

6.6.4 Help function

To support the users navigating the tool, a basic help menu was set up. It contains descriptions of the tool, as well as extensive help on writing SQL queries.

The help pop-up contained four parts: The basic intro, a description of all interface elements, a tutorial on how to write SQL queries, and an index of all possible SQL commands. All this information combined should support the user in writing their queries.

Implementation differed from the first prototype in its button placement. The first idea was to offer writing help near the query area, and general help further away. In this prototype, we merged the two elements. The first argument in favor of this decision is to not split up the help, such that everything is in one place and thus easy to find. Second of all, the implementation of the error messages was not clear enough yet to be linked with the query writing help.

6.6.5 Query history

In the first prototype, we already suggested the storage of queries. There, we placed them on the top of the page. Feedback during the first user test suggested that horizontal scrolling may not be optimal in this case. Therefore in this new prototype, query storage was placed on the right side of the page, allowing for vertical scrolling. The term Query History was suggested by one of our users, and as it can show all the queries one has ran before, it seemed a fitting name.

Storing a query into history happens through the store button. All new queries are automatically numbered, such that they have unique names. This name is used to represent the query in the history list. Users can also edit this name to make it more descriptive. Once they click the Store button, a new block with this query name appears in the history list. This block also contains a little button with the trashcan icon. This allows users to remove the query from storage. This should be intuitive as the users’ mental model should match the trashcan to the functionality of throwing something away. Opening the query is done by pressing the block with the name of the respective query. This also automatically executes the query. This action should also be intuitive as it matches similar systems.

inventory
  sID (Integer)
  pID (Integer)
  date (Date)
  quantity (Integer)
  unit_price (Decimal(10, 2))

Figure 6.4: An excerpt of the schema for the Inventory table
6.6.6 Other aspects

Another feature that should simplify the interaction with SQL is the syntax highlighting. This is a feature included in all commercial systems, as it supports the user in knowing which words are reserved. This was not implemented by hand but through the library highlight.js.

Another item of convenience that we added was the Clear button. This button clears the query name, query contents, visualization, and results. It leaves the query history intact. This allows the user to quickly start from scratch.

Finally, there are things that could have been implemented based on research and the feedback of the users: Loading different databases, Detailed error messages, split help functionality, a merge function and advanced data displayed in the query history. These were not our first priority but will be taken into account in further iterations of development.

6.6.7 Colour psychology

The colors chosen for the interface are blue tones, yellow and green. This section will describe our findings on color psychology, and explain the effect of the chosen colors.

Elliot and Maier (2014) wrote a review analyzing the effect of color on achievement. They state that no prominent literature of the effects of color perception on psychological functioning exists, although some literature on color and physiology is available. For example, there are the more intuitive findings of warm and cool colors and their effect on arousal, as well as the distinction between positive and negative colors.

Elliot and Maier (2014) focused on productivity and achievements. It is important to keep in mind that color effects depend on the context. One of their findings is that “[...] blue stores and websites are rated as more relaxing, less crowded, and even more trustworthy”. These positive associations with blue (peace (Mehta & Zhu, 2009)) and green (calmness (Clarke & Costall, 2008)) may come from the links to the natural realm. These colors may positively affect performance during task engagement. Besides, blue logos have been linked to high competence. Their review also showed evidence that blue and green may be beneficial to creativity. Mehta and Zhu (2009) agree that tasks requiring creativity and imagination may benefit from the color blue.

Yellow (and red), as positive colors, are posed to be stimulating and invite action. However, some other studies suggest that yellow may inhibit performance (Elliot & Maier, 2014). This types of contradicting findings occur regularly, as is also shown by Mehta and Zhu (2009). They, similar to Elliot and Maier (2014), state that the effect of a color may be dependent on its context.

Also of interest in this context are the effects of green as found by Moller, Elliot and Maier (2009). Their paper describes the links between green and red, and both positive and negative valence. They found an association between green and words representing success, for both nouns and adjectives, although the effect was reduced when white was added as a control condition.

One limitation of recent color studies is the approach of showing participants one color at a time (Clarke & Costall, 2008). Depending on the context of the research, such as the effect colored light, this may be appropriate. However, an interface does never contain only one color.

In summary, we chose blue as the background color for its relaxing effect. Green was added for its effect on creativity, and its link to success. Finally, some details were drawn in yellow, as they represented more advanced functionality or were used to highlight small details that require attention.

6.7 User test

Now that the second prototype has been implemented, we can start on a more involved user test, where the participants write queries and execute them. In this section, we describe the study and our findings.

4https://highlightjs.org
6.7.1 Methodology

In this section we aim to answer the following subquestions:

- How do (novice and expert) users interact with this new system?
- Is the new system an improvement on existing systems?

Participants and design

The experiment was undertaken with five participants of various occupations. All participants were male and between the ages of 20 and 28.

This user test was a paid study taking about 40 minutes. Participants were paid the seven euros for participation. The study contained four parts: a questionnaire, an interaction with the system, another questionnaire, and an interview.

Materials

As this experiment involved a working prototype, it was undertaken on a laptop, specifically a 2017 MacBook Air. The system runs in the browser, for which we chose Chromium, which ran in full-screen mode such that no extra tool bars were visible. The interaction between the participants and the system was recorded via screen capture.

The participants were asked to solve four questions, for which we present an example solution in Table 6.3. The questions were selected to focus on subqueries and negation because these were the queries of which we wanted to reduce the mental load.

We used two different questionnaires. The first asked for personal characteristics, such as age, gender and self-reported experience levels. The second questionnaire was the SUS. This stands for System Usability Scale, and was introduced by Brooke (1996). It contains 10 questions that evaluate their experience with the system, rated on a five-point Likert-scale.

A final source of data comes from the system itself. Each query that the participants executed was automatically recorded in a digital log.

Procedure

Upon entering the room, the participants was welcomed and introduced to the experiment. They signed their informed consent form and were then given the first questionnaire.

Then followed the main part of the experiment, which was the interaction with the system. We first introduced the system to the participants, by showing them its various functionalities. Then, the participants had approximately 30 minutes to answer four questions. After the time was up, the participants could write down which queries they thought they answered correctly.

The third part was to evaluate the system. First, they received the SUS questionnaire. After they filled this in, we moved on to a short interview. Participants were first asked for their general impression of the system, then we discussed all the specific additions that were made to improve the interaction. They were asked to evaluate the visualization, the database schema, query storage, and the help function.

6.7.2 Results

From the complete set of data gathered during the experiment, the interviews provide the most valuable data for this iteration of the design process. In this section we analyze the quantitative data gathered during the interviews. Then, we reflect on the design and determine what items should change in the next iteration.
Question  
List all the product IDs that were bought by at most two different customers

```
SELECT p.pID
FROM purchase AS p
EXCEPT
SELECT p1.pID
FROM purchase AS p1, purchase AS p2, purchase AS p3
WHERE p1.pID = p2.pID
AND p1.pID = p3.pID
AND p1.cID <> p2.cID
AND p1.cID <> p3.cID
AND p2.cID <> p3.cID
```

Possible answer

List of the customers (ID and name) who purchased on the same date both a product with name ‘Onions’ and a product with name ‘Coffee’ (not necessarily at the same store)

```
SELECT DISTINCT c.cID, c.cName
FROM product p1, product p2, purchase pu1, purchase pu2, customer c
WHERE p1.pName = 'Onions'
AND p2.pName = 'Coffee'
AND p1.pID = pu1.pID
AND p2.pID = pu2.pID
AND pu1.date = pu2.date
AND pu1.cID = pu2.cID
```

List of the IDs of the customers that made a purchase at every store.

```
SELECT c.cID
FROM customer c
JOIN ( 
SELECT cID
FROM purchase
GROUP by cID
HAVING count (distinct sID) = (select count(*) from store)
) d on c.cID = d.cID
```

Find the name of the product that is sold for the highest price (ever) and the name of the store that sold that item for that price.

```
WITH max_price (value) as ( 
SELECT MAX(p.price / p.quantity)
FROM purchase p)
SELECT s.sName, pr.pName
FROM purchase p, store s, max_price m
WHERE p.price = m.value
AND s.sID = p.sID
AND p.pID = pr.pID
```

Table 6.3: The questions for the user test
CHAPTER 6. SYSTEM DESIGN

General

In general, the participants stated that the system looks nice, easy and clean. They called it a ‘positive experience’ (participant 4) and felt that the system was ‘without much distraction’ (participant 5). The short introduction was sufficient to get a feeling for how everything worked, according to participant 4.

Visualization

It was very striking to find that the majority of the participants did not use the visualization. Reasons for this behavior were various. Some participants acknowledged that they were not used to this functionality, and thus forgot to use it (participants 2 and 5). This reflects the mere exposure effect (also known as the familiarity principle), introduced by Zajonc (1968). The mere exposure effect poses that repeated exposure to a stimulus makes an individual prefer it over a new stimulus. The effect of this principle is that “familiar objects are preferred over novel objects.” (Jones, Young & Claypool, 2011, p. 1). As no other systems that we know of present this functionality, this mechanism is likely at play here. Some participants noted that longer usage of the system (beyond the 30-minute experiment) should make some familiar with this functionality, such that they will use it more often.

Another reason for not using the visitation was that some participants thought it would not help them (participants 4 and 5). They argued that they connected the concepts in their head, without visualizing them.

In addition, many of the participants were out of practice with SQL. Comparing the self-reported skill measures to the amount of correctly answered questions, we found proof of the overconfidence effect, specifically the overestimation aspect (Moore & Schatz, 2017). The average self-reported skill level on SQL was 8/10, but in the end, four of the five participants reported that they had not managed to finish more than one (of four) questions correctly. This is a big discrepancy between the expectations of the participants and their final results. One participant explicitly mentioned that their trial and error method of writing the query prevented them from actively using the visualization (participant 2).

Beyond longer use, one solution to alleviate these issues is to make the visualization reactive. This means that any time the query in the text field is executable without errors, we automatically generate the corresponding visualisation. This automatically updating visualization may support the users in expanding their query. However, there is a drawback. The updating of the visualisation means that the interface is constantly changing, which may distract or interrupt the user. The corresponding loss of attention may reduce performance instead of improving it. Future iterations of the system should be tested to see which solution is preferable.

Although the visualization was not used to the extent that we were hoping, there was positive feedback as well. Based on the examples in the help menu, participants noted it looked nice and clear. There were also some indications that the visualization may help reduce the cognitive load. For example, participant 3 said it “helps to keep track of what you were doing”, and participant 5 said: “For very complex queries, this may help find mistakes in the query.”

One more suggestion was proposed by participant 3. Many of the queries they wrote contained views. Although the visualization could deal with that, the node representing this view could not be expanded. They suggested implementing this interaction, such that working with views is simplified.

Database schema

The participants had a positive experience with the database schema. They said it worked as expected and was nicely structured and clear (participants 3, 4 and 5).

The participants did find one drawback with the current implementation: one needs to choose between typing and looking at the database schema (participant 1, 2). When the schema is hidden, it is easy to write column- and table names incorrectly, which leads to server errors. They
suggested keeping the schema visible, to prevent this from occurring. Also, participant 2 suggested reducing the size of the database schema by making it into an expandable list of table names.

Help

During this short experiment, the help function was used sparingly. All participants agreed that the functionality is useful and that it contained most of the information they needed. Participant 4 acknowledged that the help function was useful when, after the experiment, they found out that the information they needed was present there.

Participant 5 noted that the list of SQL commands could benefit from some examples.

Query history

The query history mechanism received very positive feedback. Participants called it useful (participant 1, 5), easy and helpful (participant 2, 5).

An explicit example came from participant four, who had forgotten how to use an INNER JOIN and thus used the mechanism to store an example query.

Participant 5 said it was easy that the mechanism used local storage in the browser, as you did not have to decide where to store query locally. Besides, giving the queries a name makes it easy to describe them accurately and recall them quickly.

Participant 3, on the other hand, preferred to store a query primarily as a result table. The implementation made sense to them but was different from their expectations.

Other suggestions

We already discussed suggestions by the participants: reactive visualization generation and a visible database schema. One more thing was consistently mentioned by the participants. When the query contained an error, the only message returned was a server 500 error. As this message is non-specific, it does not support the user in correcting their error. It would be helpful for the users to know which word or line was wrong. This should be taken into account in the next prototype.

6.7.3 Discussion

In this section we discussed the following subquestion:

How do (novice and expert) users interact with this new system?

The fact that this second prototype was fully functional, was a great addition to this user-centered design process. The system allowed us to evaluate our design decisions.

However, the setup of the experiment does not fully reflect the usage of database systems in the real world. For novice users, the usage is close, as they would often use a database system for practice of homework problems. For experts, on the other hand, daily usage of the database system looks very different. Expert users are typically familiar with their database schema, as well as with their queries. Typical production databases are also much more complex and larger in size. This makes the queries they typically write much more complex than the ones that we practiced here.

Although support for simple queries probably looks different from support for complex queries, we believe that this user test will still indicate whether this system and our additions to the querying process are a step in the right direction.

From the findings presented, we can say that the second prototype was well received. Of course the system still contains bugs, and there are certainly improvements to be made. However, it seems that we are on the right track for developing a system that supports users in writing queries.

Towards successful interaction between Humans and Databases 109
6.8 Conclusion

In this chapter we presented the concept of user centered design. We ran three user tests, in order to develop each part of the interface in an appropriate manner. During each of the studies we found proof for our intuition and for existing literature. Our participants confirm our decisions, and the iterative design cycle lends more credence to our findings.

When we compare the current solution to the problem analysis posed in Section 6.1, we can conclude that we took into account the stakeholders and user tasks correctly. Most of the requirement posed there are fulfilled as well. Two requirements that were not implemented are feedback on repeated queries, and intuitive error messages (including pointing out the location of the error). The first of these requirements is not essential for a simple working prototype, of which this is the first iteration. The second requirement would have been very nice to implement, but wasn’t possible due to time constraints. As there is no simple solution to the problem, it will take time to develop a satisfactory implementation.

In this chapter we also aimed to answer the final subquestion:

Is the new system an improvement on existing systems?

In this section we focused on analyzing the interview questions of the final study. From the results of the qualitative studies in this chapter, we believe that this design is a step in the right direction. Although we did not run the quantitative analysis that can support such a claim, we are able to use various quotes by participants to support our design decisions. Future work may use the quantitative data that we gathered in the final study to make a bolder claim. We leave judgment of the tool up to the reader.
Chapter 7

Conclusions

In this thesis we addressed the following main question:

How can we create tools to facilitate interaction between novice users and databases?

In this final chapter, we will summarize our findings in order to answer our main question.

We started this thesis in Chapter 3 by exploring the question: What are some existing visual querying systems? We chose graph data as a starting point, due to its intrinsic visual nature. Our examination of research on visualizing graph data uncovered that many details need to be taken into account. On top of a review on appropriately visualizing the query and its results, we provided several sets of guidelines for interface design. Our taxonomy showed details on ten recent visual graph querying systems, most of which did not have demos available. We found that none of the systems implement full querying functionality, but many systems did support iterative querying processes. Examining these systems allowed us to capture the good aspects of each design, as well as ideas for improving upon it.

In Chapter 4 we continued the design process by examining which humans factors play a role in the interaction with databases. As the field of Human-Database Interaction has been underexposed, we gathered support from papers on the interaction with Information Systems and those on human factors in programming. We determined that there are four dimensions to the interaction: technical, task, human and individual. The two main technical factors that play a role in interaction are documentation and error processing. A prominent task factor is wording and presentation. A central human factor is the iterative nature of cognition, where the action and the environment influence each other. Of further interest are dividing attention and reducing working memory load. Important individual factors are confidence and motivation. Each of these factors and all of the ones presented in that chapter should be taken into account when designing a new database system.

In Chapter 5 we examined how users interact with existing database systems, to see how the factors we listed in the previous chapter would play out in the real world. We ran an experiment where Computer Science students interacted with an SQL database. We found proof that SQL presents a high cognitive load to these users, which might be reduced by providing them with the database schema within the system. Many different types of transgressions were found, but one of the most significant was the avoidance of negation. Other unproductive behaviors include repeatedly executing similar queries, working sloppily, and writing queries many orders of magnitude more complex than required. These insights into the behavior of real-world users were taken into account when designing the prototypes.

We started Chapter 6 by examining the stakeholders and requirements. With this information in hand, we started the User-Centered Design process. Our first user test aimed to answer the question: what is an intuitive way to visualize database searches? This email-based test asked a set of Ph.D. students to intuitively visualize a small set of SQL queries. Although the results were widely varying, we found support for using our graph-based visualization as a solution.
CHAPTER 7. CONCLUSIONS

We continued with a second experiment, where we presented users with a paper prototype. The goal of this experiment was to find out what an ideal database interface looks like for a novice user. First, we asked participants to draw their ideal interface, later they worked with an existing paper prototype. The initial drawings by the participants reflected many of the components that are present in the first prototype. It is also noteworthy that the participants were mostly in agreement with each-other on aspects such as button placement. Two functionalities stood out positively for the participants: the pinning of queries and the interactive visualization.

With this feedback of the participants, we built a second prototype. This prototype was a fully working database system implemented in JavaScript and Python. Again we ran a user test to examine how users interact with this new system. The second prototype was well received. Four elements were added to the system to simplify the querying process: a visual representation, the database schema, a help menu, and query storing functionality. Of these, the main attraction was the visual representation, which participants reported they were hardly using. Various reasons for his behavior were given, one of which was that they were not used to this functionality. Changing the system to make the visualization reactive might be beneficial for these users, as in that case the visualization no longer requires a button. Although there are still elements to improve, we can use various quotes by our participants to support our design decisions. We believe that this design is a step in the right direction towards successful interactions between humans and databases.

Taking a step back to our main question, it was answered by the process presented in this thesis. To reach our goal of facilitating interaction, we first examined existing systems that aimed for similar goals. As we found shortcomings in these systems that could be solved, we created a new design, also taking into account all (human) factors that may play a role. In order to build a successful system, theories should be validated in practice and users should be kept in the loop during the design process. Any design should be incremental, taking into account all feedback. Interviews with users of our system show that we have made a positive contribution on top of existing systems.

7.1 Future work

Throughout this thesis, we have provided pointers for future work. The literature on Human-Database Interaction is limited, as is research on individual factors. Data from the various experiments performed in this project can be examined in more depth. And of course, the prototype can be taken through various additional cycles of the User-Centered Design process.

This project will be continued in a Ph.D. project in the Department of Mathematics and Computer Science, where we will continue working towards successful interactions between Humans and Databases.
References


Bhowmick, S. S. (2014). DB x HCI: towards bridging the chasm between graph data management and HCI. In Database and expert systems applications (pp. 1–11). 1, 12


References

Systems, 4, 251–273. 58, 72, 73
References


Towards successful interaction between Humans and Databases 115
References


Kulesza, T., Stumpf, S., Burnett, M., Yang, S., Kwan, I. & Wong, W.-K. (2013, sep). Too much,
too little, or just right? Ways explanations impact end users’ mental models. In 2013 IEEE Symposium on Visual Languages and Human Centric Computing (pp. 3–10). IEEE. 43, 46, 47, 48


References


References

of the may 19-22, 1975, national computer conference and exposition (pp. 439–445). ACM.


Yerkes, R. M. & Dodson, J. D. (1908, nov). The relation of strength of stimulus to rapidity of habit-formation. *Journal of Comparative Neurology and Psychology*, 18(5), 459–482. 53


Appendix A

Raw Text Analysis Examples

A.1 Examples of Inefficient Formulations

```
query4 = "SELECT DISTINCT s.sName
FROM store as s, purchase as p
WHERE s.sid=p.sid AND s.sName IN
  (SELECT DISTINCT sName FROM store GROUP BY sName HAVING COUNT(DISTINCT(sID))>=2)
GROUP BY s.sName
HAVING AVG(p.quantity) > 4"
```

Figure A.1: Unnecessary DISTINCT by participant 15.
APPENDIX A. RAW TEXT ANALYSIS EXAMPLES

Figure A.2: Unnecessary JOIN by participant 68.

```sql
query2 = ```
SELECT c.cID, c.cName
FROM customer AS c, purchase AS pu, product AS pr1, product AS pr2
WHERE
pu.cID=c.cID AND
pu.pID=pr1.pID AND
pr1.pName="Onions";
```

Figure A.3: Tautological or inconsistent subcondition, by participant 105.

```sql
query2 = ```
SELECT 
    pur.cID, pur.date
FROM 
    purchase pur, product prod
WHERE 
    prod.pID == pur.pID
GROUP BY 
    pur.cID, pur.date
HAVING 
    prod.pName == 'Coffee' AND prod.pName == 'Onions'
```

Figure A.4: IN condition can be replaced by a comparison, from participant 31.

```sql
query2 = ```
SELECT c.cID, c.cName
FROM customer c, purchase p, product pr
WHERE c.cID=p.cID AND p.pID=pr.pID AND pr.pName="Onions" AND (c.cID, c.cName) IN(
    SELECT c.cID, c.cName
    FROM customer c, purchase p, product pr
    WHERE c.cID=p.cID AND p.pID=pr.pID AND pr.pName="Coffee"
)
```
A.2 Examples of Violations of Standard Patterns

Figure A.5: HAVING without GROUP BY, from participant 28.

```
query2 = ""
    SELECT c.ID, c.cName, PURCHASE.pID, PURCHASE.date
    FROM PURCHASE
    JOIN CUSTOMER ON CUSTOMER.cID = PURCHASE.cID
    JOIN PRODUCT ON PRODUCT.pID = PURCHASE.pID
    HAVING pName = 'Onions' OR pName = 'Coffee'
```

Figure A.6: The condition in the subquery can be moved up (participant 39)

```
query1 = ""
SELECT pID
FROM ( 
    SELECT pID, COUNT(DISTINCT cID) as c
    FROM purchase p
    GROUP BY pID
    WHERE c < 3
) 
```

Figure A.7: Missing JOIN condition (participant 121).
Appendix B

Visualisation Experiment Results

Table B.1: Participant one’s visualisation of query 1.

<table>
<thead>
<tr>
<th>clID</th>
<th>cName</th>
<th>street</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Jonas</td>
<td>Lijnbeek</td>
<td>Eindhoven</td>
</tr>
<tr>
<td>4567</td>
<td>Coco</td>
<td>Central</td>
<td>Amsterdam</td>
</tr>
</tbody>
</table>
APPENDIX B. VISUALISATION EXPERIMENT RESULTS

Figure B.2: Participant one’s visualisation of query 2.

Figure B.3: Participant one’s visualisation of query 3.
Figure B.4: Participant one's visualisation of query 4.
Figure B.5: Participant two’s visualisation of query 1 and 2.
Figure B.6: Participant two’s visualisation of query 3 and 4.
Figure B.7: Participant three’s visualisation of query 1 and 2.
Figure B.8: Participant three’s visualisation of query 3 and 4.
Figure B.9: The visualisations as drawn by participant four.
Figure B.10: The visualisations as drawn by participant five.
Appendix C

Paper Prototype Experiment Findings

C.1 Designs
Figure C.1: The interface as drawn by participant 1.
Figure C.2: The interface as drawn by participant 2.
Figure C.3: The interface as drawn by participant 3.
C.2 Findings by participant

Findings from participant 1:
- Buttons need to be placed close to the items that they apply to. So execute should be close to the text of the query. "I like the buttons very close in relation to what’s going on."
- Help buttons are regularly placed at the bottom of the screen.
- The Help button needs to be easy to find.
- Separates query and results on one side, and the visualization on the other side (left/right).
- Participant ’recognized’ a linear order in the process, finishing each aspect before moving on to another aspect of the tool.
- Participant thought visualization was about the results, not the query.
- Save and store could be placed either under the query, or at the ‘end’ of everything.
- The hard part about using Jupyter was that sometimes you had to scroll a lot, and it was hard to see where everything was.
- Participant grouped Execute and Generate button together, as well as Save and Store.
- Participant prefers rounded corners for all buttons, and maybe colors, to make them more easy to see (distinguish between them)
- Participant places store button to the left of save button, as store means you still want to work on it, whereas save indicates a final product.
- Additional functionality could include the option to resize parts of the interface, such as increasing or decreasing the size of the visualization. Possibly by moving around the ‘middle line’, or adding zooming buttons.
- Thumb tack ‘button’ for temporarily storing the query was not clear, possibly due to a mismatch of storing and pinning as concepts. Also, the button was not clear enough and should maybe have a border and/or color.
- Participant likes the idea of pinning the queries. "The pinning, that’s the biggest asset”
- Comparing the prototypes shows similar layout.
- Participant really likes the prototype as made by experimenter.
- When asked to incorporate space for temp storing the queries, they indicate a bar at the bottom of the screen, or to the side.
- Participant thinks that placement of the merge button is not optimal, as it relates to pinned queries.
- Participant thinks Datamodell should be higher up as it is "an overview of everything you have”.
- Participant likes the visualization of the graph with expandable nodes. "This is very easy and makes it interactive for the viewer”. "That I really like, that’s actually really well done”.
- Participant would place the pin button on the right upper side of the vis. "Right now it is more in the middle, and it is harder to see, whereas if it is on the side it is easier to pick out”. "It should really be set apart from the visualization”
- Participant prefers named tab options for pinning.
- Participant prefers prototype over Jupyter.

Findings from participant 2:
- The most important part is the area to write the query in
- Participant prefers screen in portrait mode, as this was their experience with Jupyter.
- Help button should be near the query area.
- Visualization should come before results, as it gives an overview.
- Save button comes at the end of everything (in time and space).
- Temporarily storing is less important than saving
- Less important items are displayed smaller
- Participant draws buttons left- and right aligned.
- The help button should stand out, as it is something you want to notice quickly when you have a problem.
- For this participant the help button breaks through their button alignment as used throughout the design. Why? Participant confirms that closeness to the area it relates to is more...
important than the button standing out.

- Thumb tack 'button' for temporarily storing the query was clear.
- Participant confirms that visualization corresponds to the query.
- Visualization should not contain full data, only headers of the table.
- Visualization is useful and nice.
- The clean interface of the experimenter prototype gives a better impression when you open it.
- Participant does not like the fact that all buttons have the same 'importance'.
- All buttons in a row does have the advantage that they are easy to find, you don’t have to look all over the place.
- Participant would put the help button on the left side of the query area, aligned with the top.
- The layout of the major areas differs a bit.
- Participant likes the visualization next to the query, so you can quickly compare them.
- "It helps to see the query, visualize it, and from that it is easier to write it down correctly. For the data, it is not that important to have it right next to it."
- Experimenter prototype lacks a save button.
- A menu bar could be on both the left side and the top, participant prefers top.
- Participant prefers named tabs over other pinning options. They contain too much information and could be confusing.
- "I feel like the visualization helps. Me, and a lot of people, while we were learning about queries, had a hard time understanding exactly what it is and what they look like. The visualization really helps with that. Such as with how the connection goes, and how it works, basically. I think this is a really helpful way to do it."

Findings participant 3:

- I like the interfaces where you have the most important thing on top.
- Query area fills the top of the screen, with a smaller text box within it on the left.
- Participant adds buttons to validate and save the query to the right of the text box.
- The size of the result area depends on the size of the database.
- The visualization location depends on the purpose, how you want to visualize the query, do you want to keep track of it, do you want to modify it...
- The first thing that comes to mind for visualization is using a kind of set representation.
- Participant suggested an interaction where selecting data changes the query, and changing the query changes the visualization. This is the goal of the tool.
- "I always thought some visual feedback on how you build a query [...] can help to make a decision. I think it can also help to drive hypotheses, understanding which kind of data you need."
- "I think, if [the visual representation] is the core, to build a query, it could also take a bigger part of the screen [...]"
- Help button is next to the query area.
- "It is a help button to formulate a query, so I would keep it in the area dedicated to the query."
- Help button can be a small info button or something like a question mark.
- A general help button can also be added to the menu bar.
- The save and store buttons are drawn in an options bar in the result area.
- Participant draws a history 'list' next to the result area, which shows that one has saved or stored the query. "It contains instances with some information such as how many records I have and the query that you built. And then maybe the one that you still have [in the full view] can be highlighted."
- Participant proposes a circular travel through the system, when moving from aspect to aspect. This also applies to storing a query and then retrieving another, bringing it to the top.
- For other functionalities, participant suggested an excel-like, spreadsheet representation of the result table. This allows for sorting, zooming and interaction such as being able to select
different records.
• Participant did not recognize the thumb tack as a button.
• Participant wanted to try the merge functionality.
• Participant found the layout of the experimenter’s prototype intuitive.
• “I like the fact that you have all these options, quite clear there on the screen”
• Maybe I would move the buttons to the top of the areas, but that is personal preference.
• Pinned queries are similar to participant’s history suggestion.
• Participant preferred their own solution, as ”if you have a lot of queries, I think we are more used to scrolling down”.
• Participant preferred the named tabs over the other options ”It is nice and simple.” ”the query tabs are tricky, because you have a lot of text”.
• Comparing the named tabs to history, ”I don’t know how many things you can encode. [...] you still want to always be aware of the results. You could maybe solve this by adding a small badge to the tab (that shows the number of records)”. Information could also be conveyed through tooltips or other pop-ups.
• The thumb tack button needs to be made more clear.
• If the visualization is the core aspect of the system, you could consider swapping these (vis and results table). It is then true that this (vis) is placed further away from this (pinned vis), then you can rely on this (history) vertically such that it is still connected.
• ”All the components are there, all the things that I would expect are there.”
• ”I think it is important to focus on what you think is the core element, such as the visualization and the merging. To highlight these capabilities.”
• ”I like the fact that you’re thinking to keep things simple and neat.”
• Participant likes the visualization ”I think it is good, especially if you can interact with that.”
• Participant suggests to link the visualization and the result table ”In the sense that if you click, you get some kind of interaction here (between vis and result table). Then you dont overlap too much information.”

C.3 Findings by category

Button placement.
• Participant 1
  – Buttons need to be placed close to the items that they apply to. So execute should be close to the text of the query. ”I like the buttons very close in relation to what’s going on.”
  – Help buttons are regularly placed at the bottom of the screen.
  – The Help button needs to be easy to find.
  – Save and store could be placed either under the query, or at the ’end’ of everything.
  – Participant grouped Execute and Generate button together, as well as Save and Store.
  – Participant prefers rounded corners for all buttons, and maybe colors, to make them more easy to see (distinguish between them)
  – Participant places store button to the left of save button, as store means you still want to work on it, whereas save indicates a final product.
• Participant 2
  – Help button should be near the query area.
  – Save button comes at the end of everything (in time and space).
  – Temporarily storing is less important than saving.
  – Less important items are displayed smaller
  – Participant draws buttons left- and right aligned.
  – The help button should stand out, as it is something you want to notice quickly when you have a problem.
  – For this participant the help button breaks through their button alignment as used
throughout the design. Why? Participant confirms that closeness to the area it relates to is more important than the button standing out.

- **Participant 3**
  - Help button is next to the query area.
  - “It is a help button to formulate a query, so I would keep it in the area dedicated to the query.”
  - Help button can be a small info button or something like a question mark.
  - A general help button can also be added to the menu bar.
  - The save and store buttons are drawn in an options bar in the result area.

**Ordering and placement.**

- **Participant 1**
  - Separates query and results on one side, and the visualization on the other side (left/right)
  - Participant ‘recognized’ a linear order in the process, finishing each aspect before moving on to another aspect of the tool.
  - Save and store could be placed either under the query, or at the ‘end’ of everything.
  - When asked to incorporate space for temp storing the queries, they indicate a bar at the bottom of the screen, or to the side.

- **Participant 2**
  - The most important part is the area to write the query in.
  - Participant prefers screen in portrait mode, as this was their experience with Jupyter.
  - Visualization should come before results, as it gives an overview.
  - Save button comes at the end of everything (in time and space).

- **Participant 3**
  - I like the interfaces where you have the most important thing on top.
  - Query area fills the top of the screen, with a smaller text box within it on the left.
  - “The visualization location depends on the purpose, how you want to visualize the query, do you want to keep track of it, do you want to modify it... ”
  - Participant proposes a circular travel through the system, when moving from aspect to aspect. This also applies to storing a query and then retrieving another, bringing it to the top.

**Possible additions, changes.**

- **Participant 1**
  - Additional functionality could include the option to resize parts of the interface, such as increasing or decreasing the size of the visualization. Possibly by moving around the ‘middle line’, or adding zooming buttons.
  - Thumb tack ‘button’ for temporarily storing the query was not clear, possibly due to a mismatch of storing and pinning as concepts. Also, the button was not clear enough and should maybe have a border and/or color.
  - Participant thinks that placement of the merge button is not optimal, as it relates to pinned queries.
  - Participant thinks Datamodel should be higher up as it is ”an overview of everything you have”.
  - Participant would place the pin button on the right upper side of the vis. ”Right now it is more in the middle, and it is harder to see, whereas if it is on the side it is easier to pick out”. ”It should really be set apart from the visualization”
  - Participant prefers named tab options for pinning.

- **Participant 2**
  - Thumb tack ‘button’ for temporarily storing the query was clear.
  - Participant does not like the fact that all buttons have the same ‘importance’.
  - Participant would put the help button on the left side of the query area, aligned with the top.
  - Experimenter prototype lacks a save button.
  - Participant prefers named tabs over other pinning options. They contain too much information and could be confusing.
• Participant 3
  – Participant adds buttons to validate and save the query to the right of the text box.
  – The first thing that comes to mind for visualization is using a kind of set representation.
  – Participant draws a history 'list' next to the result area, which shows that one has saved or stored the query. "It contains instances with some information such as how many records I have and the query that you built. And then maybe the one that you still have [in the full view] can be highlighted."
  – Participant proposes a circular travel through the system, when moving from aspect to aspect. This also applies to storing a query and then retrieving another, bringing it to the top.
  – For other functionalities, participant suggested an excel-like, spreadsheet representation of the result table. This allows for sorting, zooming and interaction such as being able to select different records.
  – Participant did not recognize the thumb tack as a button.
  – Maybe I would move the buttons to the top of the areas, but that is personal preference.
  – Participant preferred the named tabs over the other options "It is nice and simple." "the query tabs are tricky, because you have a lot of text".
  – Comparing the named tabs to history, "I don’t know how many things you can encode. [...] you still want to always be aware of the results. You could maybe solve this by adding a small badge to the tab (that shows the number of records)". Information could also be conveyed through tooltips or other pop-ups.
  – The thumb tack button needs to be made more clear.
  – If the visualization is the core aspect of the system, you could consider swapping these (vis and results table). It is then true that this (vis) is placed further away from this (pinned vis), then you can rely on this (history) vertically such that it is still connected.
  – "I think it is important to focus on what you think is the core element, such as the visualization and the merging. To highlight these capabilities."
  – "I think, if [the visual representation] is the core, to build a query, it could also take a bigger part of the screen [...]"
  – Participant suggests to link the visualization and the result table "In the sense that if you click, you get some kind of interaction here (between vis and result table). Then you don't overlap too much information."

Positive feedback
• Participant 1
  – Participant likes the idea of pinning the queries. "The pinning, that’s the biggest asset"
  – Participant really likes the prototype as made by experimenter.
  – Participant likes the visualization of the graph with expandable nodes. "This is very easy and makes it interactive for the viewer". "That I really like, that’s actually really well done".
• Participant 2
  – Participant confirms that visualization corresponds to the query.
  – Visualization is useful and nice.
  – The clean interface of the experimenter prototype gives a better impression when you open it.
  – All buttons in a row does have the advantage that they are easy to find, you don’t have to look all over the place.
  – Participant likes the visualization next to the query, so you can quickly compare them.
  – "It helps to see the query, visualize it, and from that it is easier to write it down correctly. For the data, it is not that important to have it right next to it."
  – "I feel like the visualization helps. Me, and a lot of people, while we were learning about queries, had a hard time understanding exactly what it is and what they look like. The visualization really helps with that. Such as with how the connection goes, and how it works, basically. I think this is a really helpful way to do it."
• Participant 3
APPENDIX C. PAPER PROTOTYPE EXPERIMENT FINDINGS

- “I always thought some visual feedback on how you build a query [...] can help to make a decision. I think it can also help to drive hypotheses, understanding which kind of data you need.”
- Participant found the layout of the experimenter’s prototype intuitive.
- “I like the fact that you have all these options, quite clear there on the screen”
- “All the components are there, all the things that I would expect are there.”
- “I like the fact that you’re thinking to keep things simple and neat.”
- Participant likes the visualization “I think it is good, especially if you can interact with that.”

Comparison to existing systems.

• Participant 1
  - The hard part about using Jupyter was that sometimes you had to scroll a lot, and it was hard to see where everything was.
  - Participant prefers prototype over Jupyter.

• Participant 2
  - When comparing to Jupyter: “I feel like the visualization helps. Me, and a lot of people, while we were learning about queries, had a hard time understanding exactly what it is and what they look like. The visualization really helps with that. Such as with how the connection goes, and how it works, basically. I think this is a really helpful way to do it.”