Easing access to relational databases

Investigating access to relational databases in the context of both novice and would-be expert users

Philip Garner

School of Computing and Communications

Lancaster University

This thesis is submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

August 2016

For Amy, I love you!

Declaration

I hereby declare that, except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains fewer than 80,000 words including appendices, footnotes, tables and equations but excluding the bibliography.

Philip Garner

August 2016

Acknowledgements

Firstly I would like to thank John Mariani, my supervisor, for guiding me through this PhD, and my final year project before that. Thank you for steering me through it with a good balance of experience, knowledge and a sense of humour.

Thank you to my family for putting up with me and this work, particularly on those days when it looked like it might never end. Thank you also to all the fantastic people I've lived with over the years at Lancaster University: Olly, Ed, Joe, Amy, Charlotte and Becky; I probably would never have started this if you guys didn't make Lancaster such an awesome place to be. Olly, you convinced me to consider doing a PhD in the first place, it's your fault I'm writing this!

The biggest thanks of all go to Amy, you've supported me through everything this PhD has thrown at me and there is no chance I would have got this far without you. Thank you and I love you.

Thanks also to all the great officemates I've had the pleasure of meeting over the years: Rajiv, Barry, Matjaž, Alex, Chris and Mark. Thanks should also go to all those people at Lancaster University who have helped behind the scenes. A special mention needs to go to the staff (particularly Alistair) and students on the 2014/15 SCC130 course who helped me with the evaluation of SiS along with the anonymous 100-or-so people who took part in the CAFTAN evaluation.

This PhD was funded by the Engineering and Physical Sciences Research Council (EPSRC), thanks for the money!

Abstract

Relational databases are commonplace in a wide variety of applications and are used by a broad range of users with varying levels of technical understanding. Extracting information from relational databases can be difficult; novice users should be able to do so without any understanding of the database structure or query languages and those who wish to become experts can find it difficult to learn the skills required. Many applications designed for the novice user demand some understanding of the underlying database and/or are limited in their ability to translate keywords to appropriate results. Some educational applications often fail to provide assistance in key areas such as using joins, learning textual SQL and building queries from scratch. This thesis presents two applications: Context Aware Free Text ANalysis (CAFTAN) that aims to provide accurate keyword query interpretations for novices, and SQL in Steps (SiS) designed for students learning SQL. Both CAFTAN and SiS are subject to detailed evaluations; the former is shown to be capable of interpreting keyword queries in a way similar to humans; the latter was integrated into an undergraduate databases course and showed the potential benefits of introducing graphical aids into a student's learning process. The findings presented in this thesis have the potential to improve keyword search over relational databases in both a generic and customised context, as well as easing the process of learning SQL for new experts.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Relational databases are commonplace within a huge number of computing applications. They form an important backbone to many different applications from small smart-phone applications to high-traffic web servers. Although not the only database model (Section 2.2), the relational model has, for a long time, been the de facto standard for many applications meaning almost all computer users interact with them on a daily basis.

The widespread use of the relational database means they are used by a huge number of different users (Sections 2.4 and 6.1) (Chen, 1999; Elmasri and Navathe, 2010; Jarke and Vassiliou, 1985; Martin, 1973; Shneiderman, 1978). Some of these users (experts) are equipped with a detailed knowledge of the database structure and how best to use it while others (novices) have no such knowledge and interact with databases through customised interfaces and potentially without realising they are using databases.

The desire to improve the way in which information is extracted from databases has drawn attention within the research community for a number of decades (Chapter 3). Zloof (1975) is generally credited with the first visual query language (VQL): Query By Example (QBE). QBE was designed to remove the need for database users to understand the physical arrangement of data on disk and it remains a popular style of querying databases with similar query styles used in mainstream applications such as Microsoft Access (Section 3.1.1). In the years following the release of QBE a number of form based query languages (e.g. Embley (1989); Epstein (1991); Heiler and Rosenthal (1985); Houben and Paradaens (1989); Shu (1985)) featured in the literature in an attempt to relieve users of the need to build complex textual queries. As the graphical capabilities of computers began to extend beyond simple forms a number of digram (e.g. Clark and Wu (1994); Danaparamita and Gatterbauer (2011); Larson and Wallick (1984)) and icon (e.g. Massari et al. (1994); Tsuda et al. (1989)) based VQLs were introduced along with hybrid languages (e.g. Ahlberg and Shneiderman (1994); King and Melville (1984); Pietriga et al. (2001)) that utilise a combination of forms, diagrams and/or icons. Although VQLs were a popular area of research their target audience remains somewhat unclear; they can be difficult for novice users to understand (Section 3.1.5.4) and they lack the power needed for many expert users (Section 3.1.5.1).

In response to the rise of internet search engines, keyword search over relational databases became increasingly popular, which resulted in a large body of work aimed directly at meeting the needs of the novice user. There are a number of different techniques used to integrate keyword search with relational databases, some of which remodel the data (e.g. Bhalotia et al. (2002); Wang et al. (2006)) while others attempt to use indexing structures to find the most appropriate response to a keyword query (e.g. Agrawal et al. (2002); Hristidis and Papakonstantinou (2002)). Despite the clear aim at the novice user, some keyword search applications are limited in their querying capabilities or demand database knowledge not normally associated with the novice (Section 3.2.5).

The complexity of SQL that places it beyond the reach of the novice user also means learning the language can be problematic for many students (Cembalo et al., 2011; Kearns et al., 1996; Prior, 2003; Russell and Cumming, 2004; Sadiq et al., 2004). Despite a large body of research in the area of education much of the existing work suffers from some key pitfalls such as the use of confusing illustrations (e.g. Cvetanovic et al. (2011)) or only working with a finished statement rather than being used throughout the building process (e.g. Abelló et al. (2008); Mitrovic (1998b); Sadiq et al. (2004)) (Section 3.3.4).

1.1 Research problem

The wide range of users and uses of relational databases allows for many avenues of research within the field. The focus of this work is on the extraction of information from relational databases. Despite a single aim, to ease the extraction of data from relational databases, the work produced in response to the problem is almost entirely dictated by the target user. The extensive capabilities of SQL and the range of databases it can be used with mean that a single solution is unlikely to satisfy the needs of all database users. The all-encompassing research aim investigated in this thesis is the process of **easing access to relational databases**.

This general problem can be broken down further to provide some direction and focus for the work. First, it is important to **identify the users of relational databases and their needs**. The identification of users is important in order to produce software that is designed to meet their specific needs; designing for all users will likely result in a system too complex for novice users but that lacks the power needed for expert users. Once the users and their requirements are clearly identified it is possible to **investigate how software can be used to improve the way in which they interact with relational databases**.

1.2 Research approach

The research approach taken in this thesis involves identifying the specific area of database interactions to investigate along with the users making such interactions. After identifying the users and their needs within the scope of the problem, prototypes can be designed, developed and evaluated. The following steps were taken in order to complete this research:

Define the specific area of database interactions to investigate. Database interactions include the extraction, insertion, deletion or modification of the database contents and schema. Designing a system to remedy problems across all these areas is not only unreasonable but quite possibly unnecessary. The vast majority of users are not concerned with interacting with the schema and primarily build queries to extract the necessary data from databases. As a result, the primary focus of this work is the extraction of data, the SELECT clause; the reason for this is its popularity in both real world applications and long-standing presence on Computer Science courses (e.g. Harvard University (2015); Lancaster University (2015); Stanford University (2015); University College London (2015); University of Cambridge (2015); University of Oxford (2015)).

Identify the different users of relational databases. The ubiquitous use of databases results in a huge number of different database users. There are a number of existing classifications of database users however; some are very fine grained resulting in a large number of categories. Within the context of extracting information many of these categories can often be merged into one. Elmasri and Navathe (2010) describe *"Database Administrators"*, *"Database Designers"* and a selection of *"End Users"*; for the purpose of extracting information, designers, administrators and even some casual end users all demand the ability to construct SELECT statements that accurately reflect their needs. For the purpose of this work the users are divided into two main categories: novice and expert (Section 2.4), an additional category of "student" describes those users transitioning from novice to expert (Section 6.1)

Identify the ways in which software can assist users. In response to the popularity of keyword search the novice demands database applications that return relevant information in exchange for associated keywords. Much of the existing work has some significant limitations (Section 3.2.5) that restrict the extent to which they can be used beyond simple searches. New searching techniques have the potential to improve the way in which search terms are handled. Expert users typically interact with databases using textual environments that enable them to produce very accurate, powerful queries without being restricted by GUIs; because of this they make very few demands of database user interfaces (Section 2.6). However, becoming an expert is a well recognised problem (Danaparamita and Gatterbauer, 2011; Kearns et al., 1996; Mitrovic, 1998b; Russell and Cumming, 2004; Sadiq et al., 2004) and many students find learning SQL difficult. Learning SQL within a textual environment can be intimidating for users whose only prior experience with databases might be through the use of customised UIs aimed at novices. GUIs have a real potential to improve the learning environment but must be carefully designed to avoid users becoming dependant on the system such that they are incapable of interacting with databases without graphical assistance.

Develop prototype applications to meet the needs of the given users. Two prototype applications were developed (CAFTAN and SiS, presented in Chapters 5 and 6 respectively) to meet the needs of our two classes of user. CAFTAN aims to interpret keyword queries over any given relational database, combining them with a mixture of AND and OR operations where appropriate. SiS is an online learning environment in which students can graphically construct queries whilst building a good understanding of textual queries. Key features of SiS include graphical representations of the database structure and Boolean expressions along with live query translations and results that adapt as the student builds their query.

Evaluate the prototypes to quantify their quality. Both CAFTAN and SiS were subject to detailed evaluations that attempted to quantify their quality (Chapter 7). The evaluation of CAFTAN involved comparing the interpretations of keyword queries made by CAFTAN against those made by a humans and two bespoke query applications. SiS was integrated into a live databases course in an attempt to observe the differences between learning SQL with/without graphical aids. Both evaluations show that the tools developed each have the potential to resolve problems with existing applications; CAFTAN offers query interpretations beyond the capabilities of some bespoke applications and SiS provides students with a number of useful resources throughout their learning process.

1.3 Contributions

The contributions of this work in relation to extracting information from relational databases can be categorised as empirical and practical.

1.3.1 Empirical contributions

The evaluations of both CAFTAN and SiS present large quantities of both qualitative and quantitative data. The evaluation of CAFTAN resulted in a data set that not only facilitates a comparison of it against other applications but also offers an insight into the querying habits of novice users. This data was collected independently of CAFTAN and, therefore, is available for use with future iterations of CAFTAN and other keyword search applications. The user study of SiS produced large amounts of qualitative data in response to the various features and their eligibility to be included as part of an educational course. This data was collected from the 101 participants in the form of multiple questionnaires and a focus group.

1.3.2 Practical contributions

Although both applications developed for this work were designed and implemented as prototypes they offer many potential benefits in real world scenarios. CAFTAN out-performs a number of bespoke applications in query interpretation showing that it could be used as a stand-alone search application or used as the basis for a customised user interface. SiS demonstrates the benefits of learning SQL with

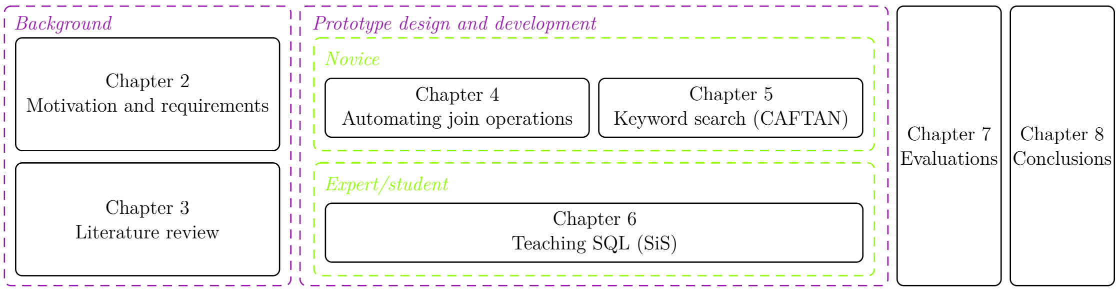


Figure 1.1: Thesis structure

graphical assistance without making students dependant on the UI such that they cannot further develop their skills beyond the need for SiS.

1.4 Outline of thesis

Figure 1.1 shows the structure of this thesis, it is structured over eight chapters, starting with the background and literature review, followed by details of the two key applications. These applications are subject to detailed evaluations and the findings of the work are summarised in the final chapter.

Chapters 2 and 3 provide some background to the research presented in this thesis. Chapter 2 describes some of the different database models currently in use and categorises users as *novices* or *experts* according to their needs and understanding. Chapter 3 provides a detailed review of the literature relating to the extraction of information from relational databases; this is divided into three main parts: visual query languages, keyword search and educational applications.

Chapter 4 describes Shorthand SQL (SSQL), a software library designed to automate the process of joining the required relations in queries involving multiple relations. The use of SSQL is explored throughout various different applications presented in this thesis.

Chapter 5 describes an application, CAFTAN, designed to meet the needs of novice user described in Chapter 2. This application is designed to operate over any given relational database and attempts to address the shortcomings of some of the existing work explored in Chapter 3. The main focus of CAFTAN is to enable the user to build queries involving a mixture of AND and OR operations without any need to understand the structure of the database they are querying.

The process of becoming an expert relational database user involves learning theoretical elements and mastering SQL. Chapter 6 describes the *student* user as someone who is transitioning from a *novice* to an *expert* and describes an application, SiS, designed to assist students in their learning process. SiS aims to allow students to graphically build a query while maintaining an understanding of its textual counterpart by displaying live translations of the query along with the results it yields.

Chapter 7 describes the evaluations of both CAFTAN and SiS. The evaluation of CAFTAN involves comparing the interpretation of keyword queries made by CAFTAN with those made by participants and bespoke query applications. The evaluation shows that CAFTAN is capable of quickly producing query interpretations that closely resemble those made by humans along with highlighting a number of benefits over bespoke systems. The evaluation of SiS involved integrating it into a live undergraduate databases course; throughout this process participants highlighted the need for such an application and identified a number of strengths and weaknesses of the application.

Chapter 8 concludes the thesis and presents a number of avenues for future research stemming from this work.

1.4.1 Movie database

All work explored in this thesis is done in a generic manner and is not targeted at any given database. To provide some context a movie database is used throughout this work. The data used to populate the database was obtained via TMDb (Apiary, 2015) and is described in detail in Appendix A.

Chapter 2

Motivation and requirements

This chapter offers an overview of various database technologies and the users who interact with them. The chapter is structured as follows:

- Section 2.1 discusses the popularity of databases within the context of a number of different domains.
- Section 2.2 compares the relational model against recent NoSQL alternatives in terms of their functionality and the way in which information is extracted from them.
- Section 2.3 describes the use of software designed to ease the construction of queries with a particular focus on Elcee, a piece of software designed for querying email data that provided much of the inspiration for this work.
- Section 2.4 categorises users of databases while Section 2.5 and Section 2.6 discuss the requirements for novice and expert database users respectively.
- Finally, Section 2.7 summarises the contents of this chapter.

2.1 Popularity of databases

Databases are hugely popular and are a staple of almost all computer applications. They are used to varying extents in many applications with some relying heavily on them and others using them to satisfy much smaller use cases. This section discusses the popularity of databases within the following contexts:

- Web applications
- Smartphone applications
- Desktop applications
- Sensor data/IoT

Web 2.0 has been commonplace for many years, websites that allow users to create and modify content are all built using a storage system of some sort. Databases are exceedingly popular for web applications with many websites being built on relational databases. Many popular Content Management Systems (CMSs) such as WordPress, Joomla and Drupal utilise relational databases to store the information upon which these websites heavily rely (Drupal, 2015; Joomla!, 2015; WordPress, 2015). Websites required to handle larger volumes of traffic, such as large scale social media sites (e.g. Facebook[1], Twitter[2] etc.), use a combination of different database applications or customised versions of existing ones that are optimised to handle the heavy traffic the sites receive.

Persistently storing data within a mobile application is a common requirement. All three of the most popular mobile operating systems (Kantar Worldpanel, 2015), Android (Android, 2015), iOS (Apple, 2015) and Windows Phone (Denning, 2013) employ SQLite to enable developers to persistently store information. SQLite is a relational database well suited to mobile applications as the entire database is contained within a single file that requires no administration and provides the vast majority of functionality offered by client-server databases designed for larger scale applications (SQLite, 2015a). In addition to smartphone applications the self-contained database is also used in a vast range of applications in which a

[1]https://www.facebook.com/
[2]https://twitter.com/

client-server database is not an option and storing data in a simple file is too restrictive, such as desktop applications.

The term Internet of Things (IoT) refers to the integration of real-world objects with sensors and connectivity; it has become a popular area for both the research community and the everyday consumer. Sensors are embedded into a huge range of devices with data gathered from them fed to users in an attempt to provide them with meaningful information. The information gathered may relate to data from home appliances such as thermostats or motion sensors (e.g. Nest[3]) or information gathered by mobile devices. Fitness tracking using specialised devices (e.g. Fitbit[4] and Jawbone[5]) or the sensors present in many smartphones (using applications such as Strava[6] or MapMyRide[7]) is also an increasingly popular trend. This use of sensors has the potential to generate vast quantities of data that needs to be stored and analysed to enable meaningful information to be passed to the users. Data gathered from such devices is stored in databases of various types enabling users to extract the desired information at a later date.

2.2 Databases types

There are many different types of databases and the relational model, proposed by Codd (1970), was the de-facto standard for many years. More recently other data models such as graph and document databases have become increasingly popular.

Here we discuss a number of different database models that are currently in widespread use, highlighting their strengths and weaknesses.

[3]https://nest.com/
[4]http://www.fitbit.com/
[5]https://jawbone.com/
[6]https://www.strava.com
[7]http://www.mapmyride.com/

2.2.1 Relational databases

The relational model was introduced in 1970 by Codd in an attempt to remove the need for users to understand the physical arrangement of data on disk. Since then it has gained popularity and is frequently regarded as the *“go-to”* data model, and as a result is used in a massive range of applications.

There are a number of different implementations of the relational database however all are based upon the same data model and offer similar features and methods of interaction. The following are a selection of features of the relational model.

2.2.1.1 ACID properties

Relational databases typically provide the four ACID properties:

- Atomicity
- Consistency
- Isolation
- Durability

The ACID properties, a term first coined by Haerder and Reuter (1983) ensures the reliable handling of transactions within a database.

Atomicity refers to way in which transactions are handled in relational databases. Preserving atomicity ensures that a transaction will be completed in its entirety or not at all. The classic example to demonstrate the benefits of atomicity is the transfer of money from one person to another: *Person A* pays *Person B* £100. The balance for *Person A* must be reduced by £100 and the balance of *Person B* must be increased by £100, this involves two operations that form a single transaction. To preserve atomicity it must be impossible for only one of the two operations within the transaction to be completed.

Consistency ensures that any operations executed on the database will not leave it in an invalid state. This refers to the preservation of relationships and data types specified within the database. If the transaction would result in an invalid state then the database is returned to the state prior to the start of the transaction.

Isolation prevents transactions interacting with each other. The aim of this is to ensure that multiple transactions that are executed on a database concurrently would have the same end result as if they were executed one after another.

Durability of a database simply means that once a transaction is complete the effects of its operations will remain even in the event of subsequent errors or crashes. For instance, if the insertion of data was immediately followed by a power failure the newly inserted data would remain present once power was restored.

The preservation of these ACID properties makes relational databases a popular choice when consistency and quality of data is of paramount importance.

2.2.1.2 Normalisation

Normalisation is a technique used in database design and aims to remove duplicated data by spreading information across multiple tables within a database. This process allows single entities to be represented only once, meaning modifications to the data are less likely to result in inconsistencies across the database.

Spreading data across multiple tables requires connections to be made between them; primary-foreign key relationships facilitate this by maintaining references to one table in another. For instance, a university course might store a reference to the department that manages the course rather than repeating the information relating to the department.

Normalisation can lead to the use of multiple tables even in simple, small scale databases. To extract information spread across multiple relations users must construct queries containing join operations which connect multiple tables in order to extract the desired information.

2.2.1.3 SQL

Relational databases and Structured Query Language (SQL) are almost inseparable, it was first established as an ANSI standard in 1986 and ISO in 1987 (International Organization for Standardization, 1987) and continued to evolve over multiple iterations until the latest release in 2011 (International Organization for Standardization, 2011), all implementations of relational databases provide a variant of the standard. Throughout the various iterations of the SQL standard the language has matured to introduce new features as a result of new demands on database systems. The language allows for the creation, modification and extraction of information from databases. Very few, if any, implementations adhere strictly to the standard however, the majority of systems follow large portions of the standard allowing developers to use different databases without the need to learn a new query language. The standardisation and widespread use of SQL enables developers to choose a database application based upon its strengths and the resources available rather than because of a familiarity with the query language.

2.2.2 NoSQL databases

Although the relational model was, for a long time, seen as the only viable option for database applications there are now a wide range of different models available that can be chosen to best suit the application. The presence of NoSQL options in a world in which relational databases have ruled has threatened its dominance.

Not only SQL (NoSQL) is the term that refers to database systems that utilise models other than the relational model. An aim of many NoSQL technologies is to allow for horizontal scaling to allow the system to handle the potentially vast quantities of data and high volume of data requests.

There are a number of different data models that various NoSQL databases utilise; here we present a selection, by no means a complete set, of the common structures and their potential applications.

2.2.2.1 Key-value

Key-value databases such as MemcacheDB[8] and Redis[9] are perhaps the simplest of all NoSQL databases. They offer the functionality of maps or associative arrays and allow for the retrieval and assignment of values in response to a key value.

2.2.2.2 Document model

The document model is used to store primarily text information in a semi structured format. MongoDB[10] and CouchDB[11] are popular document databases; both store data in the JSON format using the attribute-value notation to enable the contents of a database to be read easily by both humans and computers.

A benefit of utilising semi-structured data for data stores is the flexibility that such a structure brings. Consider a database designed for use with an online shop; using JSON notation allows a single collection to store data for the entire shop regardless of the variation in products. For example, some products such as DVDs have a cast associated with them while musical CDs do not, they have track listings. This variation in data can lead to unusual and complex schema designs when using the relational model but the flexibility of a document store lends itself perfectly to such a scenario; attributes can be added where necessary allowing the database to develop with the needs of the system.

In contrast to the relational model, document databases do not support strictly enforced relationships between documents, they are achieved through the use of nested data or references to other documents. In the case of one-to-many relationships such as a single customer having multiple delivery addresses this information is all stored as multi-value attributes within the customer document. When many-many relationships are needed such as that between products and customers (one customer can order many products and one product can be ordered by many customers) references to both ends of the relationship are required. In

[8]http://memcachedb.org/
[9]http://redis.io/
[10]https://www.mongodb.com
[11]http://couchdb.apache.org/

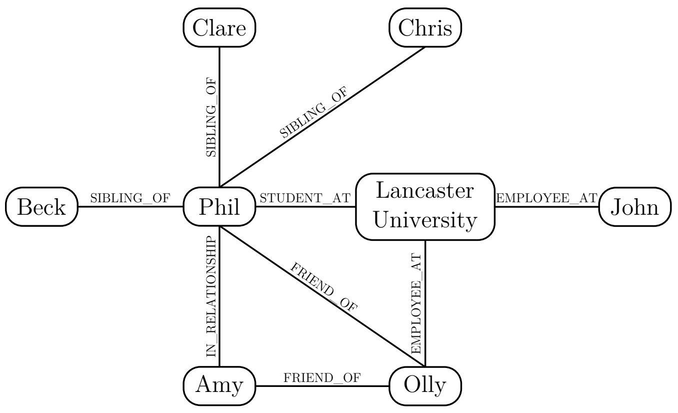


FIGURE 2.1: A snippet of a graph database representing the relationship between various people

this scenario the customer document stores a reference to the products ordered and the product stores a reference to the customers who ordered it; this type of relationship needs to be carefully managed by users or management software to avoid the formation of one-way relationships in which a customer might be linked to a non existent product, or vice-versa.

2.2.2.3 Graph databases

Graph databases model the data as a series of nodes that are connected by edges. Graph databases perform particularly well when modelling large quantities of interconnected data, social networks are good examples of this. Figure 2.1 shows a small extract of data from a sample graph database that might be used to store data to support a social network. The nodes represent seven people and one institution and are connected via five different types of edge depending upon the relationship between the various nodes. In many popular graph databases, such as Neo4j[12], both nodes and edges can contain information; in our example the nodes representing people might contain additional information such as their date of birth while the edges might contain information such as anniversaries for relationships or dates of employment for employees.

The process of querying graph databases is optimised for traversing edges and finding connections between different nodes. To imitate graph databases using

[12]http://neo4j.com/

relational technologies requires the use of a junction table, allowing one record to be linked with one or more other records, this approach is common within relational database design. The use of junction tables allows for immediate connections (e.g. friends) and indirect connections (e.g. friends of friends) to be quickly found but beyond this level (e.g. friends of friends of friends) the performance of relational databases suffer significantly while graph databases remain capable of handling such queries (Robinson et al., 2013, page 20). This example illustrates how the choice of database can impact on the performance of a system.

2.2.2.4 Query languages

NoSQL databases are, unlike their relational counterparts, not manipulated using SQL. SQL depends heavily on the structured nature of data stored within relational databases, NoSQL databases typically contain unstructured or semi-structured data and utilise their own query languages instead. Unlike SQL there is no standardisation of NoSQL query languages, this is largely due to the differing data structures and requirements of the database styles. The following is a selection of different NoSQL query languages along with brief descriptions of their meaning.

Key-value databases As a result of the simple nature of key-value databases the process of extracting data from such databases is also simple. To add a value to a key-value database such as MemcacheDB database the `set` command is used, the following example sets the value of the *mykey* entry to *"myvalue"*.

```
set mykey myvalue
```

Values associated with keys can be extracted by using the `get` command, the following example extracts the value associated with the *mykey* key.

```
get mykey
```

MemcacheDB also includes a number of commands to facilitate the modification and removal of data. As key-value databases do not allow the formation of queries based upon their values more complex criteria regarding the extraction of data is impossible.

Document databases As discussed in Section 2.2.2.2 document databases such as MongoDB and CouchDB store data in the JSON format; to query such databases the user specifies the criteria for the documents that are to be returned. To query a MongoDB database the user specifies their requirements in a query document, any documents in the collection that match the given criteria are returned by the query. In the following example the query specifies that the *type* attribute should be equal to *"DVD"* and the *price* attribute must be less than 10; all DVDs with a price of less than £10 are returned following this query:

```
db.inventory.find( { type: 'DVD', price: { $lt: 10.00 }
   } )
```

MongoDB also allows for more complex queries including grouping operations that allow users to perform summation and counting operations on documents. CouchDB queries are performed through the use of HTTP requests, these have the same functionality as MongoDB in that the user specifies criteria for the desired documents but use a different request format.

Graph databases The following is an extract of the cypher query language used in the Neo4j graph database. Cypher allows users to specify a starting node, according to its attributes, and constraints regarding its edges, such a query returns the nodes connected to the starting nodes by the given edges. The following example extracts the node representing *"Phil"* and all the nodes connected to it via a *SIBLING_OF* edge; the details for *"Phil"* and all his siblings are returned. When executed on the sample database shown in Figure 2.1 this will return the nodes representing *"Phil"*, *"Beck"*, *"Chris"* and *"Clare"*.

```
MATCH (you {name:"Phil"})-[:SIBLING_OF]->(siblings)
RETURN you, siblings
```

This query can be expanded to find more complex connections within the database, such as friends of friends, functionality that has clear benefits in scenarios such as social network applications.

2.2.3 Is the relational model dead?

Although some (e.g. Fleming (2007)) have interpreted the rise of NoSQL databases as the death of the relational model, the introduction of new models should be seen as alternatives, rather than replacements, to the relational model. NoSQL database applications allow developers to choose a database model that best suits their system rather than being forced to use the relational model when it is not best suited to their application.

Different database models offer different benefits, the relational model prioritises high quality, consistent, data strictly maintaining the ACID properties. These features allow for fast and efficient joins and aggregations to be performed. Many NoSQL models such as the document and graph model prioritise flexibility and the handling of vast quantities of data that many modern systems demand.

In summary, different data models offer different sets of strengths and weaknesses and the decision of which database to use should be made based upon the needs of the application. The relational model remains a perfectly viable option in a world where NoSQL databases are of increasing popularity.

The recent work by Google into their F1 database, a system designed to support their AdWords[13] business, highlights the enduring relevance of the relational model. They identified that the scalability of NoSQL databases is unmatched by existing relational models but the lack of ACID compliance introduces problems: *"if you need a highly scalable, high-throughput data store, the only viable option is to use a NoSQL key/value store, and to work around the lack of ACID transactional guarantees and the lack of conveniences"* (Shute et al., 2013). The lack of ACID properties in most NoSQL databases means that developers are responsible for ensuring these

[13]https://www.google.co.uk/adwords/

requirements are adhered to. As a difficult task to solve at the application level this was described as: *"an unacceptable burden to place on developers and that consistency problems should be solved at the database level"* (Shute et al., 2013).

2.2.3.1 Focus on relational databases

Despite the introduction of other databases, the relational model remains exceedingly popular and it maintains its dominance in the database world. In a recent poll the developer community voted SQL as the second most popular technology, behind JavaScript, for the second year running (Stack Overflow, 2015).

As a result of its vast popularity and widespread use the remainder of this work focusses exclusively on the relational model and SQL as a means to manipulate such databases.

2.3 Layers of abstraction

The problem of extracting information from databases, particularly relational databases, is a well recognised problem and there are numerous applications that work in conjunction with them to ease the extraction of their data. Such systems come in various levels of abstraction, typically the lower the level of abstraction the more powerful a system is. Figure 2.2 shows some examples of systems at various levels of abstraction. Syntax highlighting offers very little in addition to a simple command line and therefore sacrifices none of a query language's capabilities but a keyword search systems offering high levels of abstraction offer limited capabilities to the user and experienced users may be unable to specify queries to the level of detail they require.

Section 3.1 describes a number of query systems designed to simplify access to relational databases. The following section describes an application designed to improve the extraction of email data, Elcee; this software provided much of the inspiration for the work described in this thesis and some aspects of it remain in the software accompanying this work.

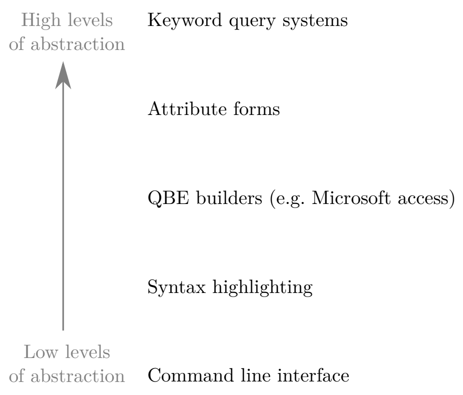

High levels of abstraction
Keyword query systems
Attribute forms
QBE builders (e.g. Microsoft access)
Syntax highlighting
Low levels of abstraction
Command line interface

FIGURE 2.2: Levels of abstraction

2.3.1 Email search with Elcee

Since the early 1970s the popularity of email has exploded, revolutionising the way in which we communicate in both a business and social setting. The total number of email accounts was estimated at over 2.9 billion in 2010 and is predicted to continue growing into the future (The Radacti Group, 2010). Although the massive number of email accounts shows the popularity of email it is not without its drawbacks. Information overload is a term that is commonly used to define excessive amounts of information to the point at which it impedes decision making; the term was initially proposed by Schroder et al. (1967) who stated that the ability to make decisions correlates directly with the amount of information available - to a point. The idea of users being unable to handle excessive amounts of information extends to email data and email overload.

The term email overload has various definitions within the literature, in the context of this work we adopt the definition used by Dabbish and Kraut (2006): *"users' perceptions that their own use of email has gotten out of control because they receive and send more email than they can handle, find, or process effectively"*. This definition directly expands upon that proposed by Schroder et al. (1967) in that an excessive amount of email information can make dealing with the emails very difficult.

As part of my undergraduate final year project an investigation into how advanced search functionality could alleviate email overload took place. The focus

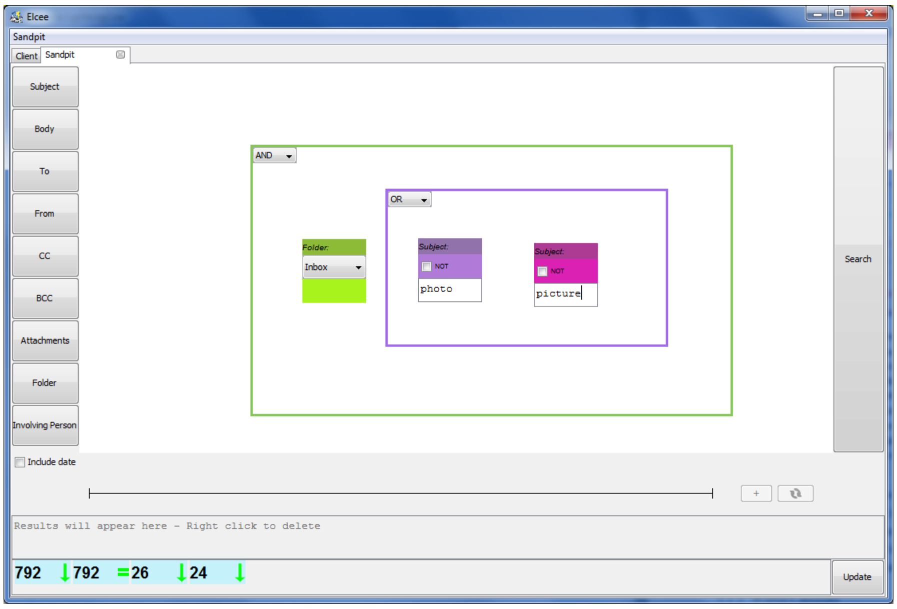


FIGURE 2.3: The Sandpit Search within Elcee showing the following query: *(folder: inbox AND (subject: "photo" OR subject: "picture"))*

of the work was a number of searching and visualisation user interfaces designed to help users extract information through search then explore the results using appropriate visualisation styles. The result of this investigation was an application called Elcee, the software has the appearance of an email client but included two advanced search features and five data visualisation styles.

A small scale evaluation of Elcee showed that the Sandpit Search (Figure 2.3) was the most popular search tool as it allowed for users to specify, in great detail, the requirements of their search leading to small but accurate result sets (high precision and recall). Sandpit Search used individual widgets on screen to represent a single criterion, these could be grouped together with Boolean operations by drawing boxes around them. Boxes represent AND or OR operations and can be nested within other boxes to allow for the construction of complex Boolean operations within a graphical environment. Each attribute of an email has a customised widget to input data when searching, as a result users can utilise drop-down menus, checkboxes and text input where appropriate.

Throughout the evaluation it became clear that Elcee included search tools considerably more capable than those in many email clients but it also became apparent that these features were capable of more than email search alone. All participants in the evaluation believed that the capabilities of Elcee reached beyond email search alone.

Transforming Sandpit Search from an email specific search tool to a generic database query builder presents a number of potential stumbling blocks:

- Visualising the database structure
- Mapping data types to widget styles
- Handling large databases
- Comparing attributes
- Specifying the data to view in the results
- The use of largely meaningless attributes (e.g. tuple identifiers)

This list of potential problems offers a brief insight into some of the problems that arise when developing a generic search tool or visual query language that is applicable to any database.

2.4 Users of databases

As discussed in Section 2.1, there are a huge range of applications for databases, this widespread use inevitably leads to a broad range of users. Different users of databases bring different requirements for using them and their technical ability varies significantly.

This section presents a review of some existing classifications of users and defines categories of users that will be referred to throughout this work.

2.4.1 Categorising users

The categorisation of users of database systems has evolved over many years and there is no single, universally accepted, definition of different classes of user. Some, including Elmasri and Navathe (2010) and Chen (1999, page 124), divide the users into many categories, making small distinctions in the way in which they interact with databases while others define fewer categories (e.g. Shneiderman (1978)).

Martin (1973) categorised novice, non-expert, users as *"operators"* and stated that user interfaces designed for these users should *"appear as natural as possible, or bewilderment will quickly turn into annoyance, criticism, or behaviour that amounts to rejection of the system"*. Jarke and Vassiliou (1985) described the *"casual"* user as *"having only a general idea about structure and content of the database"*, a definition similar to that used by Shneiderman (1978). This demonstrates how the categorisation of users changes over time; the widespread use of databases means that many users interact with them on a daily basis with no knowledge of their structure at all (rather than a *"general idea"*). This change is highlighted by Catarci and Santucci (1995) who describes *"naive"* user as, amongst other things, *"unfamiliar with the details of the internal organization of an information system"*. They also acknowledge the broad range of user classifications and attempt to simplify the definitions by dividing users into two groups: *"those who have had a certain instruction period and have computer and database knowledge, and those who do not have specific training in computer usage and database interaction"*.

A more recent classification involves categorising users according to their requirements for extracting data from the Internet of Things (IoT): *"Casual users will need to access the IoT via a user-friendly graphical user interface [. . .], and more flexible, powerful, and efficient access interfaces will be needed for expert users"* (Cooper and James, 2009).

For the purpose of this work users can be divided into two main categories: novice and expert. By focussing exclusively on the extraction of data we can merge categories described by others in the literature.

2.4.1.1 Novice users

Novice users are required to extract information from databases but are unlikely to have an awareness of the underlying database structure. They may be required to insert of modify data within a database but this will typically be done through the use of customised user interfaces designed to ease the process. Such systems may be customised user interfaces designed specifically for extracting information from a given database or generic applications that are built to interact with any given database. Novice users will never be required to interact with databases using textual query languages.

We refer to novice users according to the following definition, similar to the *"naive"* user described by Catarci and Santucci (1995):

Definition 2.1 (Novice user) *Someone who has no understanding or requirement to understand how to interact with databases using textual query languages. They may interact with databases using various specialist interfaces or graphical systems. These users have no understanding of database design or the structure of the databases they query.*

2.4.1.2 Expert users

An expert user is classified as someone who is required to form textual queries. The use of textual queries to interact with databases is common and numerous users fall into this category. Some expert users may only produce simple queries to extract small sets of data from databases while others may build complex queries that are integrated into other software applications. Regardless of the complexity of the queries produced expert users are required to have an understanding of the database structure and query language syntax to meet their individual needs; as a minimum users are required to understand where in the database certain pieces of information can be found.

The following definition of an expert is used throughout this work:

Definition 2.2 (Expert user) *Someone who is required to interact with databases using textual query languages. Expert users encompass a wide variety of users from those required to submit small textual queries to those designing, building and maintaining large scale databases. An understanding of the database structure is crucial for expert users.*

2.5 Novice requirements

Two defining characteristics of novice users is their lack of understanding of both query language syntax and database structures; as a result systems designed to facilitate novice access to relational databases must not demand prior knowledge in these areas.

Lacking the understanding of query languages and their syntax means that if novice users are to build queries they must be assisted in their construction to help them avoid mistakes. This can be achieved through the use of graphical systems that attempt to hide the intricacies of textual query languages from the user or through the use of keyword search systems that convert textual input to appropriate results.

Users unaware of the underlying database structure can struggle to query databases, particularly as normalisation can lead to data being stored across multiple tables, even in small scale databases. Without a good understanding of the database structure novice users can make false assumptions leading to erroneous queries; to tackle this problem novice users should either be made aware of the structure or have no need to understand it. The use of diagrams and textual descriptions can make users more aware of the underlying structure; however novice users typically have no need to become experts and it is often more beneficial to ensure there is no requirement for them to understand the underlying database structure.

2.6 Expert requirements

As discussed in Section 2.4.1.2 expert users are required to have an understanding of the query language syntax and an appreciation of various database structures that might be used. For users who interact with databases using textual interfaces, graphical systems can often act as less of a help and more of a hindrance (Aversano et al., 2002; Catarci et al., 1996). Graphical systems rarely offer the full scope of a textual language and can slow down those proficient in writing textual queries.

Although experts require little in the way of graphical assistance when building queries it can be difficult for users to initially learn SQL (Cembalo et al., 2011; Kearns et al., 1996; Matos et al., 2006; Mitrovic, 1998a,b; Prior, 2003; Renaud and van Biljon, 2004; Russell and Cumming, 2004; Sadiq et al., 2004; Shneiderman, 1978). The process of learning SQL is fraught with potential stumbling blocks for new users; although GUIs may have little appeal to expert users there are numerous opportunities to use them within the learning environment.

2.6.1 Requirements for educational systems

Designing systems with the aim of teaching users to become experts in building textual queries requires that a number of considerations are taken into account.

The following is a brief outline of the areas for consideration when designing an educational system:

- Who is the system for?
- What is their reason for learning SQL?
- What are the common stumbling blocks?
- Does the purpose of the software extend beyond education?
- What is the scope of the language covered by the software?

2.6.1.1 Who is the system for?

As previously described, experts can find the lack of flexibility in graphical systems limiting in comparison to their textual counterparts. As a result, graphical systems are rarely targeted at this category of user. Despite this, graphical systems can prove highly beneficial throughout the process of educating users in becoming experts.

The relevant experience of users learning SQL may vary significantly, some may have no experience in computer languages while others may want to transfer their skills from one query language to another. A common user of such educational systems are Computer Science undergraduates who may have some experience in programming languages and are expanding their knowledge into SQL.

2.6.1.2 Reasons for learning

The requirement to learn SQL can arise for a number of different reasons, these can loosely be divided into two categories:

- Learning for subject knowledge
- Learning for a specific task

Learning to gain subject knowledge is the scenario in which the user is learning a skill to be able to become competent at analysing and solving related problems. This level of knowledge is not focused at a specific task but rather at a greater understanding of the type of problems that may arise and the way in which they can be tackled. This sort of learning is present in degree schemes and similar courses.

An alternative to learning to gain subject knowledge is learning to complete a given task; in this case a user is less interested in the overall skill and is simply required to learn enough to complete a task or series of tasks. Learning to solve a given task might be used when training an employee to complete a task as part of their job, learning in this setting involves focused tasks in which the user might be expected to work towards their goal. Learning for a given task will likely involve focussing on only the skills needed to complete the task rather than a more generalised set of techniques that can be applied to a number of different problems; this can lead to mastering certain techniques quickly but can result in very few transferable skills.

Within this thesis we focus on learning to master a skill rather than to complete a specific task. This includes learning within a classroom/course setting or as an individual but not to complete a task without a greater appreciation of the query language.

2.6.1.3 Common problems

Learning to extract information from relational databases using SQL presents a number of problems, including:

- Visualising the database structure (Dekeyser et al., 2007; Kearns et al., 1996; Mitrovic, 1998b; Prior, 2003)
- Understanding the syntax of SQL (Mitrovic, 1998b)
- Visualising the results (Cembalo et al., 2011)

As previously discussed, relational databases are typically spread over multiple tables as a result of normalisation (Section 2.2.1.2). This means that to extract anything but the simplest of data users must be capable of visualising how the data is spread across these tables. A key skill in SQL is the ability to construct join operations, enabling users to extract information from multiple tables. A learning environment must supply users with adequate descriptions of the database structure allowing the user to form a good understanding of the data enabling them to form accurate queries.

The extensive capabilities of SQL mean that it includes a number of features many users are unlikely to use. Users must gain an understanding of the core components that they are likely to use on a regular basis but also become confident in their ability to extend beyond this understanding into more powerful and specialist features. Misconceptions as to the meaning of features can lead to users building on these misunderstandings and making false assumptions that might lead to erroneous results. Syntax errors in queries are met with error messages, however, these are a well recognised problem and they frequently offer little insight into the actual cause of a problem or offer a meaningful solution (Cembalo et al., 2011; Mitrovic, 1998b; Prior, 2003; Russell and Cumming, 2004).

Visualising the results of a query can be divided into two important skills: visualising the desired results of a query and also understanding the results of the intermediate steps during query production. Before building a query the user must have a good understanding of the desired output, as with any problem solving activity it is important to understand the end goal before attempting to solve it. Visualising the intermediate steps of a query is also an important skill to learn, users frequently build queries without executing them at the intermediate steps leading to them making assumptions about the result of some operations (Prior, 2003). Users must gain confidence in the correct use of each component of the language, enabling them to accurately visualise the intermediate steps of a query.

2.6.1.4 Outgrowing educational systems

A goal of educational software is often for users to eventually outgrow it. In the context of a tool used to teach computing languages the users should use the system to learn the required skills and then gradually stop using it as their confidence increases. Avoiding reliance on an educational system is important, such software is not designed for real world applications but should be used to build the skills to allow a user to tackle real world problems independently.

2.6.1.5 Meeting the needs of different *"experts"*

As discussed in Section 2.4.1.2 there are a broad range of expert users, some require the ability to write simple queries to extract information from a database while others are required to create and maintain large databases. The wide range of users categorised as expert users means that software designed to teach users SQL must allow users to use it to the extent they require; while the system must cater for users requiring advanced skills it must not deter users seeking a more basic skill set. Similarly, an educational system must allow users who wish to explore more advanced features the ability to do so.

2.7 Chapter summary

In this chapter we have highlighted the overwhelming popularity of databases across a multitude of different applications. A number of different data models were highlighted, addressing strengths and weakness of each. The widespread use of databases means they attract a massive range of users, here we have described the various users of such systems and categorised them as novice or expert within a specific context. The two user groups share very few requirements for database software designed to help them extract information from databases, as a result of this it is impossible to develop a single software solution that meets the needs of both users. A simple solution designed for novice users would lack the capabilities for an expert user as experts require little assistance when building complex textual queries. The enormous breadth of functionality required by expert users also makes developing software for them problematic, however, educational tools can help ease the process of becoming an expert by addressing a number of key problems.

Chapter 3

Literature review

The process of extracting information from relational databases has attracted considerable attention within the research community over the past forty years. This chapter presents a review of the literature relevant to this thesis, the work reviewed here largely falls into one of three categories:

- Visual Query Language (VQL)
- Keyword search
- Educational systems

Visual (or graphical) query languages are designed as direct replacements to their textual counterparts; there are various different approaches to VQLs and their functionality and target audience may vary between implementations. Section 3.1 details a number of VQLs and analyses their strengths and weaknesses.

Keyword search is a popular method of data extraction that allows novice users to access the contents of databases that normally require specialist understanding of database structures and query languages. Keyword search presents a set of results in response to a series of relevant terms provided by the user; different applications use a range of techniques to interpret the given terms as accurately as possible. Section 3.2 discusses the use of keyword queries within the context of relational databases.

As discussed in Section 2.6, expert users require more power and flexibility than is offered by those systems targeted at novice users. Educational systems designed to improve the learning process for future expert users are popular as learning SQL using textual interfaces alone frequently presents problems. Section 3.3 discusses the various approaches to systems designed to teach SQL, highlighting some areas in which they can be improved.

3.1 Visual query languages

Visual query languages (VQLs) provide an alternative to textual interfaces when interacting with databases. There are numerous definitions of VQLs, Epstein (1991) succinctly describes them as *"a visual language which is intended as a database interface and which employs visual metaphors in order to express the semantics of a database language"*.

VQLs provide an often needed graphical interpretation of concepts that can be difficult for users to interpret through text alone. *"The use of visual technologies enables a better human-computer interaction by representing database elements in a more natural way and creating an intuitive correspondence between the visual representation provided by the tool and the concepts of interest to a user"* (Aversano et al., 2002). More specifically, the role of VQLs is twofold: *"first, they aim to help the user understand the database they are working with. [...] The second aim of visual query languages is of course the visual formulation of queries themselves"* (Hartl and Weiand, 2009). Despite being described as the second aim of VQLs by Hartl and Weiand, the ability to graphically build queries is often considered the primary aim with the need to understand the database stemming from this. The normalisation of data within relational databases requires that users have a good understanding of the database structure prior to building queries, this is discussed in Section 2.2.1.2.

3.1.1 Early VQLs

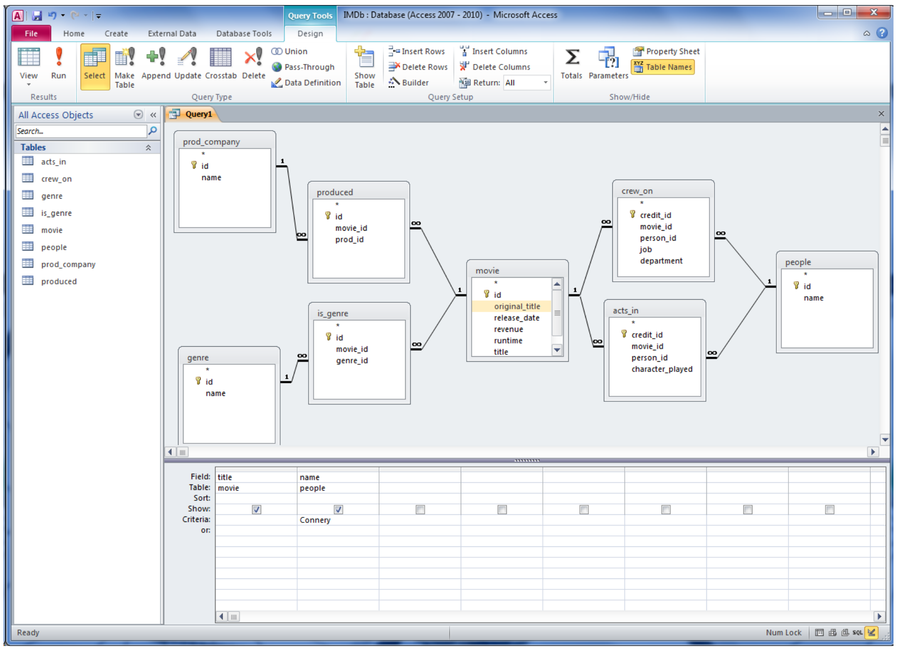
Query By Example (QBE) is generally regarded as the first attempt to introduce graphics into database query languages to increase their appeal to non-professionals (Zloof, 1975).

QBE allows users to specify criteria for attributes in a databases, diagrams are used to represent the database and criteria are entered below each attribute. A notation (entering a P below the attribute) is used to specify the attributes to be displayed in the results allowing the formation of SELECT, FROM, WHERE queries by entering data in the required forms. Additional capabilities such as the negation of criteria ($\neg$) and grouping (double underline) are enabled through the use of various symbols and special notations. Although QBE allows for the formation of queries without the use of textual languages it still requires some understanding of the database structure; join operations are not automatically handled so users must manually specify these.

The use of symbols and special notations allows for the formation of relatively complex queries, however, users are not required to have a complete understanding of the entire notation to use the system at a basic level. This premise is one shared by the majority of visual query languages in their attempt to appeal to experienced and novice users alike. The ability for a user to be able to use a small set of the operations available in a VQL without the unused parts interfering with its simplicity is crucial to its success in appealing to a broad range of users; similarly the advanced features of a visual query language must be easily accessible without basic features impeding advanced actions.

Building queries by specifying how the results are required to appear, as in QBE, remains a popular approach and is used in a wide range of modern applications. Both Microsoft Access[1] (Figure 3.1) and Microsoft SQL Server Management Studio[2] use a UI with clear similarities to QBE.

[1]https://products.office.com/en-gb/access
[2]https://msdn.microsoft.com/en-gb/library/hh213248.aspx



Field:	title	name
Table:	movie	people
Sort:		
Show:	☑	☑
Criteria:		Connery
or:		

FIGURE 3.1: The Microsoft Access visual query editor showing the formulation of a query on the movie database (Appendix A)

3.1.2 Classification of VQLs

There are a huge number of VQLs, Vadaparty et al. (1993) identified over fifty in 1993 and numerous more have been developed since. Catarci et al. (1997) produced a well recognised classification of VQLs, defining the following categories:

- Forms
- Diagrams
- Icons
- Hybrids

Form-based VQLs (e.g. aMAZE LightBench by Lemer et al. (2004)) use text input fields, such as those found in widespread use as part of advanced search systems online, to build a familiar interface that may appeal to novice users. Form-based query languages were a popular area of research when VQLs were initially developing but have become less prominent whilst the development of more abstract icon and diagram-based approaches became popular.

Visually representing database queries through the use of **diagrams** can often be considered more appropriate than form-based techniques; diagrams can often give the user a good overall view of the structure of a query and how different elements interact with each other (Danaparamita and Gatterbauer, 2011).

Icon-based visualisations use graphical representations of components in a database to allow the user to better understand the structure and to query a database. As discussed by Catarci et al. (1997) the use of icons is effective when used to represent an abstraction of a physical entity, however databases are not limited to storing only this sort of data. When representing data that cannot be displayed pictorially icon based languages can become difficult to use.

Hybrid representations of databases and their queries are combinations of other classifications. Most visual query systems fall into the hybrid category whilst still consisting predominately of one of the categories (Catarci et al., 1997). The use of a hybrid visualisation system allows the best components of each type to be combined into a single visualisation technique that utilises elements from each style.

3.1.3 Review of systems

The following is a review of a number of VQLs; due to the vast quantity of systems within the literature this is not a survey of all systems but a subset to illustrate common elements.

3.1.3.1 Forms

Form-based systems were popular with early visual query languages; they could be implemented using the limited software and hardware resources available at the time. As technology advanced, more graphically complex alternatives became possible and increased in popularity. Despite this, elements of form-based systems can be found in many modern applications.

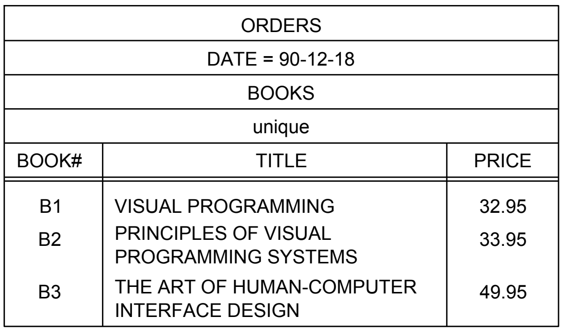
PART	PART_NBR	NAME	WIDTH
		wing	10 U. 10.1
			20 U. 20.1
			? U. WIDTH * 1.5

FIGURE 3.2: A G-WHIZ query to update the width of all the parts with the name *“wing”* (Heiler and Rosenthal, 1985).

As discussed in Section 3.1.1 Query By Example is well recognised as the first visual query language; QBE utilises form elements arranged in a tabular layout to allow users to enter criteria and notations to form a query. G-WHIZ (Heiler and Rosenthal, 1985) extends the functionality of QBE, allowing it to operate in conjunction with the functional data model and introducing the ability for users to construct recursive queries. Heiler and Rosenthal stated that a tabular layout such as that found in QBE removes *“much of the syntax burden from the user, allowing different parts of a complex query to be generated in whatever order is convenient”*. In a similar manner to QBE, G-WHIZ utilises special notations to allow users access to various different features of the language. Unlike QBE, G-WHIZ also allows for the modification of data within the database through the use of update and delete operations. Figure 3.2 shows a G-WHIZ form used to update the *width* attribute of all parts with the *name “wing”*; the update will increase those wings with a *width* of 10 or 20 to 10.1 and 20.1 respectively and update all other wings to 1.5 times their current width.

Many form-based systems offer little abstraction from the database structure, relying on the user maintaining an awareness of the schema or educating them as they use the system. The R^2-interface by Houben and Paradaens (1989) is aimed at *“inexperienced”* users but uses nested headings above forms to replicate the structure of the database; this method of arranging forms below nested headings is also used in FORMAL (Shu, 1985).

NFQL (Embley, 1989) allows designers to build customised forms that can be completed by novice users who are required to extract information from the underlying database.



ORDERS		
DATE = 90-12-18		
BOOKS		
unique		
BOOK#	TITLE	PRICE
B1	VISUAL PROGRAMMING	32.95
B2	PRINCIPLES OF VISUAL PROGRAMMING SYSTEMS	33.95
B3	THE ART OF HUMAN-COMPUTER INTERFACE DESIGN	49.95

FIGURE 3.3: A TableTalk query to find books ordered on a given date (Epstein, 1991).

TableTalk (Epstein, 1991) guides users through the construction of a query using rows in a table to narrow down their search requirements. Figure 3.3 shows a TableTalk query that extracts book numbers, titles and prices for all books ordered on the 18th December 1990 and only shows each book once (regardless of the number of times it was ordered). The top five rows of the table specify the query:

Row 1 Tells the system to use the *orders* table

Row 2 Specifies the date required for the query

Row 3 Connects the *orders* table with the *books*

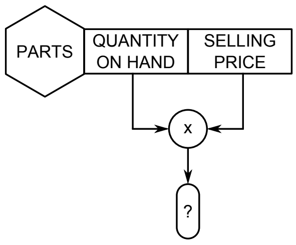
Row 4 Ensures books only appear once in the results

Row 5 Specifies the information to be displayed in the results

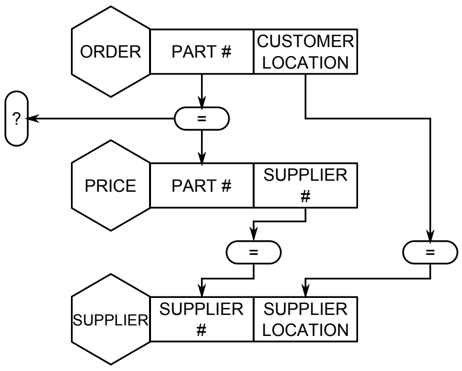
Although some of the work in relation to form-based systems is almost four decades old it can still be found in modern systems. The query designer in Microsoft Access (Figure 3.1) and Microsoft SQL Server Management Studio uses a form layout very similar to QBE, albeit in conjunction with a diagrammatic representation of the database structure.

3.1.3.2 Diagrams

With the rise of increasingly powerful hardware and software the potential to use more graphically complex query systems became increasingly popular in the 1980s.



(A) Find the total value (SELLING times QUANTITY) of all parts.

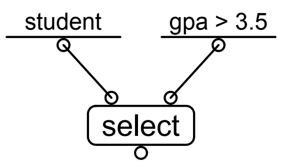


(B) Retrieve the part numbers whose destination and origin are the same city.

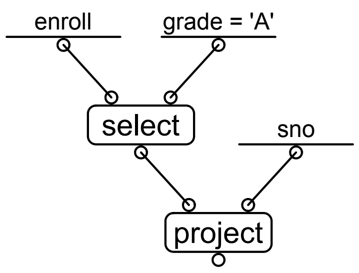
Figure 3.4: Two CUPID queries (McDonald and Stonebraker, 1975).

McDonald and Stonebraker (1975) identified two possible ways in which databases could be made more accessible to casual users: Natural Language Processing (NLP) and picture modelling. NLP refers to the interpretation and processing of queries submitted in a natural, human readable, form while picture modelling is the process of visually representing queries as graphical elements with the aim of making them easier to understand than their textual counterparts. A number of problems relating to NLP were identified including the large vocabulary and ambiguity associated with the English language: *"It is very unlikely that any two English-speaking persons understand precisely the same English"* (Codd, 1974). Due to the potential pitfalls of NLP the Casual User Pictorial Interface Design (CUPID) system was developed. CUPID uses shapes to depict various elements of a database, allowing for the construction of queries that appear to closely resemble their textual counterparts. Figure 3.4 shows two queries drawn using the CUPID notation, Figure 3.4a shows how arithmetic operations can be used in conjunction with attributes while Figure 3.4b shows how joins can be performed and values can be compared across different relations.

Larson and Wallick (1984) attempt to improve the understanding of novice users by breaking problems down into smaller problems and guiding the users towards a syntactically valid solution. Their system involves the use of diagrams to illustrate the structure of the database (in the form of entity relationship diagrams)



(A) Find the students with a GPA of more than 3.5



(B) Find student number (sno) for those students with a grade A

FIGURE 3.5: Two DFQL queries (Clark and Wu, 1994).

and the correct syntax for a query. The syntax diagrams used to ensure users form valid queries share many similarities with those used in some computer language manuals (e.g. the SQLite manual[3]). The approach taken by Larson and Wallick is to attempt to move novice users to more advanced user interfaces; this represents an unusual aim of a visual query language as, frequently, novice users have little or no need to become an expert in the field. The DFQL system (Clark and Wu, 1994) also offers very little abstraction from a textual language and appears almost as a means to visualise a written query rather than as a method to alleviate users of this requirement. Figure 3.5 shows two examples of DFQL in use, the close relationship with textual queries is clear; the functionality of the language extends beyond these examples and it can be used to perform a wide array of operations including join operations and aggregate functions. DFQL includes the ability to define common operations that can be reused as required, a potentially useful feature for those users who regularly perform the same query.

The QueryViz system (Danaparamita and Gatterbauer, 2011) is designed to allow users to understand queries written by others, something that has the potential to become difficult with more complex queries. It uses a UML-like visualisation to illustrate existing queries; this is justified by the statement that *"most database users have seen UML diagrams before"*. Although this may be true for expert users it is unlikely that many novice users are familiar with UML diagrams; this

[3]https://www.sqlite.org/lang_select.html

assumption has the potential to limit the appeal of QueryViz to those with prior experience in system design or database applications.

3.1.3.3 Icons

Unlike diagrammatic approaches to VQLs, the comparatively small body of work relating to icon-based visualisations are frequently targeted at the novice user and attempt to remove the need to understand the database structure.

IconicBrowser (Tsuda et al., 1989) focused on the fact the user should not need to understand the database structure to be able to query it using icons. The IconicBrowser uses icons to represent entities within the database and dropdown menus to select criteria from a list of possible values. Similarly, Query By Icon (QBI) (Massari et al., 1994) is primarily targeted at inexperienced users and was partly developed as diagrams can become *"too complex to be accepted by inexpert users"*.

Although icons can be extremely effective in representing database entities with physical counterparts, problems can occur when the need to visualise more abstract properties arises (Section 3.1.5.3).

3.1.3.4 Hybrids

Hybrid visual query languages combine a mixture of forms, diagrams and/or icons to represent queries. Few VQLs belong exclusively to one category and the vast majority utilise a combination of forms, diagrams and/or icons to some extent.

The FilmFinder system (Ahlberg and Shneiderman, 1994) utilises form elements, in the shape of number sliders and checkboxes, to allow users to specify the criteria for finding film data. The results of a query are shown, live, in the form of a scatter-graph style diagram. Users can then further refine their query by interacting with the diagram. Although the FilmFinder uses forms to query and diagrams to show the results the PESTO (Portable Explorer of STructured Objects) system (Carey et al., 1996) uses forms arranged on a diagram closely resembling the database structure; these forms are then used to enter criteria for the query.

Unlike some systems (e.g. Benzi et al. (1999)) Ski (King and Melville, 1984) automatically generates diagrams for use in its query system rather than relying on expert users to define the layouts to be used by novice users. The Ski system uses a series of predefined steps represented on a diagram to guide a user through the process of extracting or modifying the contents of a database. Some elements of the diagrams include form components to allow for the entry of data.

Although not an interface to databases, but rather a means to manipulate XML data, the VXT system (Pietriga et al., 2001) uses a combination of icons and diagrams to illustrate operations to be performed on XML documents. Figure 3.8 shows a snippet of the VQT interface, its use of extremely abstract notations has the potential to disorient unfamiliar users, this is discussed further in Section 3.1.5.3.

3.1.4 Target audience

The target audience of VQLs varies between implementations; Hartl and Weiand (2009) summarised this: *"the people using VQLs range from the expert database administrator, to the secretary that always uses one and the same query to create an employee list, to the hobbyist web programmer".* Despite this, only a handful are aimed primarily at the expert user (e.g. Kuntz (1993)) and the majority are targeted at novice users (e.g. Benzi et al. (1999); Houben and Paradaens (1989); Massari and Chrysanthis (1995); McDonald and Stonebraker (1975); Shu (1985); Tsuda et al. (1989)). Although there are some systems targeted at the expert user the limitations inherent to visual systems (Section 3.1.5.1) means that they are often more suited to novice users with more basic requirements. Massari and Chrysanthis (1995) recognised the simpler requirements and only allows for simple queries in which the user can specify criteria and the attributes to view in the results (SELECT, FROM, WHERE queries).

3.1.5 Shortcomings of visual query languages

Although visual query languages have experienced widespread popularity in the research community for many years they are not without their shortcomings. Many problems can be associated with visual systems in general rather than individual approaches. Here we discuss the following potential problems with VQLs:

- Their ability to satisfy the needs of expert users
- Managing the screen space, particularly when handling large databases
- Accurately representing abstract concepts in an easy to understand manner
- Ensuring graphical representations remain understandable, even when representing complex queries

3.1.5.1 Expert capabilities

Section 2.6 outlines the requirements of an expert database user. In summary, they are required to build potentially complex queries using many different aspects of the database structure and query language syntax that novice users may be unaware of. To establish this level of skill requires training to learn the query language and time spent studying the database structure. The assistance provided by graphical systems often comes at the cost of flexibility and power, as a result, expert users will rarely require the assistance of a graphical system and can find they act as more of a hindrance than a help (Aversano et al., 2002; Catarci et al., 1996).

This limitation of graphical systems means they are not generally appropriate for professional use and should be focused, as they usually are, on novice or inexperienced users.

3.1.5.2 Utilising screen real estate

Diagrammatic systems, in particular, have the potential to become overcrowded when used in conjunction with complex schemas or queries. Query interfaces based

Figure 3.6: The Mac OSX 10.10 Yosemite dock showing a fisheye effect in response to a mouse-over event. The highlighted *"Slack"* application is shown bigger than the others around it.

upon ER or schema diagrams (e.g. Czejdo et al. (1989)) are particularly vulnerable to this as the visualisation must grow to reflect the structure of the database. Burnett et al. (1995) identified the problem of fitting the required information onto the screen space available and highlighted two common solutions to the problem: *"make the drawing smaller and look at the whole, or partition the drawing into pieces and look at each piece separately"*. Although the two approaches to dealing with limited screen real estate offer viable solutions they are not without drawbacks; decreasing the size of a diagram to enable it to fit on screen runs the risk of making it unreadable while partitioning the drawing could potentially remove the much needed context the diagram as a whole provides. As the capability of computers has expanded beyond the capabilities of simplistic 2D graphics more solutions have become possible such as the use of fisheye (e.g. Bederson et al. (2004) and the Mac OSX dock in Figure 3.6) along with 3D visualisations that allow the user to arrange information in 3D space.

3.1.5.3 Accurately representing the database

The use of icons to represent entities within databases allows novice users to visualise them graphically, making it easier to quickly identify the desired components within a user interface. However, icons can become difficult to use when the need to represent an abstract concept arises; something with no physical counterpart is difficult to display using an icon. The Keiron query language (Aversano et al., 2002) is designed to teach users a textual query language such as SQL through the use of icons. It uses the example of querying a database of books by using icons (Figure 3.7). Even in this idyllic example there are clear issues found when representing attributes of a book with icons. The title attribute of a book is

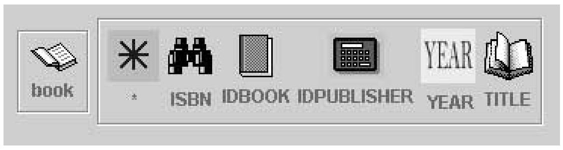


FIGURE 3.7: A selection of icons used in Keiron (a year has no obvious physical counterpart so is simply represented by the word *"YEAR"*)

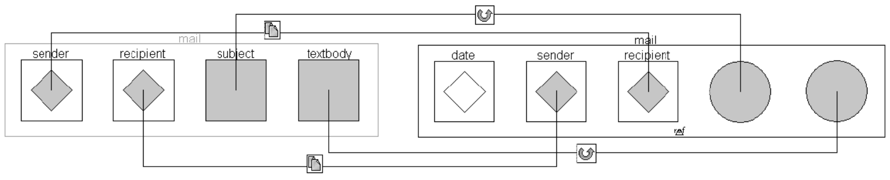


FIGURE 3.8: VXT's XML manipulation language, an example of how visual query languages can become difficult to understand

identified by a small icon of an open book, without the additional information of the attribute name it would be very unclear as to what this icon was representing. The year attribute is simply represented by an icon containing the word *"YEAR"*; the use of text within icons entirely defeats the objective of an icon-based query language and perhaps suggests a hybrid language would be more appropriate.

3.1.5.4 Excessively abstract interfaces

A common problem with many visual query languages is a level of abstraction so high the system is rendered useless, as expressed by Andries and Engels (1996): *"one also gets the impression that some of this research overshoots its mark in the sense that pure graphical formulation of a query sometimes even looks more complex than its textual equivalent"*. The VXT language (Pietriga et al., 2001) (Figure 3.8) is used for manipulating XML documents using icons and diagrams; although not used for querying data, it is a good example of how such tools can become so massively abstract that they are almost unusable.

3.2 Keyword search

In Section 3.1.3.2 we highlighted a quote by McDonald and Stonebraker where it was argued that visual query languages were more appropriate than keyword search due to the associated pitfalls. After identifying a number of problems with Natural Language Processing (NLP) McDonald and Stonebraker developed CUPID to allow novice users to build queries using diagrams. Although in 1975 NLP presented significant challenges, two key changes have occurred since then:

- Software and hardware advancements
- Frequent exposure to keyword search

Both software and hardware capabilities have increased massively in the past few decades, as a result, tasks that were previously seen as impractical can now be completed with relative ease. Hardware and software constraints from the past are often not applicable to modern applications.

Web based search engines have exploded in popularity since first established in the early 1990s (Brin and Page, 2012) and Google now handles in excess of 1.2 trillion searches per year (Google, 2012). As a result of the widespread use of keyword search across the internet the majority of users are accustomed to building keyword queries and submitting them in exchange for a series of relevant results (Park and Lee, 2010). The popularity of keyword search across the internet means that it has become a popular area of research within the databases community.

The application of keyword search over relational databases can largely be divided into two distinct categories: *graph-based* and *relational* approaches. Examples of each techniques, along with discussions regarding their strengths and weaknesses are discussed in this section.

Manning et al. (2008) described keyword queries (also referred to as free text queries) as *"queries that simply consist of query terms with no specification on their relative order, importance or where [. . .] they should be found"*. A keyword query, Q, can be defined as $Q\{t_1 \ldots t_n\}$ where t represents a series of terms specified by the user, this notation will be used throughout this work.

3.2.1 Web search relational databases

To the casual user, both keyword search on the web and over relational databases are much the same; they expect to submit a series of keyword queries and receive a list of appropriate results in response. Although, from the users perspective, the two systems appear to have the same functionality, the implementations of the systems are significantly different (Bhalotia et al., 2002; Khine et al., 2011; Liu et al., 2006; Park and Lee, 2010).

The following are some of the challenges presented in implementing keyword search over relational databases:

- Matching criteria to attributes
- The presentation of results
- Joining the appropriate relations

An important requirement for novice users (as discussed in Section 2.5) is that they must be able to interact with databases without an understanding of its structure or language syntax. As a result of this, a keyword search system should not assume any understanding of the database structure and should therefore match the given criteria to the appropriate attributes automatically.

The presentation of results in both web search engines and bespoke relational database search systems are predefined and always take the same form. However, two different keyword queries executed on relational databases might demand drastically different results visualisations. Jagadish et al. (2007) identified the difference in results styles: *"In the case of a web search, a user expects simply a set of links, with almost no interrelationship between them. In the case of a database search, a user may expect to see a table, a network, a spatial presentation on a map, or a set of points in a multidimensional space"*. This quote clearly identifies the differences between the various styles of presenting results, however, the description of what a novice user might expect in response to a keyword query on relational databases is controversial. In contrast to Jagadish et al., and in keeping with avoiding the need for novice users to learn the database structure, it may be more desirable to respond to keyword queries with a list of appropriate results, much like web search engines. However, the details displayed in these lists might vary considerably depending upon the area of the database queried. To automatically generate appropriate results visualisations presents a number of problems, particularly when choosing the attributes to display in response to a given set of criteria. Often the criteria specified can bear very little resemblance to the desired results; for instance a search for an actor on a movie database might require that a list of their acting credits are displayed.

Due to the normalisation of data in relational databases information is often spread across many different tables (Section 2.2.1.2). As a result of this, users building queries are expected to join tables together for the purpose of specifying criteria and visualising results, novice users are unlikely to have this understanding so the appropriate join operations must be automated. To automatically generate join operations can be difficult as there are often multiple routes through a database to connect the desired relations, frequently passing through junction tables that hold no meaningful data and are only used to facilitate many-many relationships between other relations. Chapter 4 discusses this problem and proposes a solution that is applicable to various different applications.

3.2.2 Approaches to keyword queries

Keyword search over relational databases is largely tackled using one of two approaches: *graph-based* or *relational* systems. In this section we discuss the two approaches within the context of a number of examples.

3.2.2.1 Graph technique

Graph based systems extract the information from within relational databases and build a graph structure to manage keyword queries. Graph structures for handling

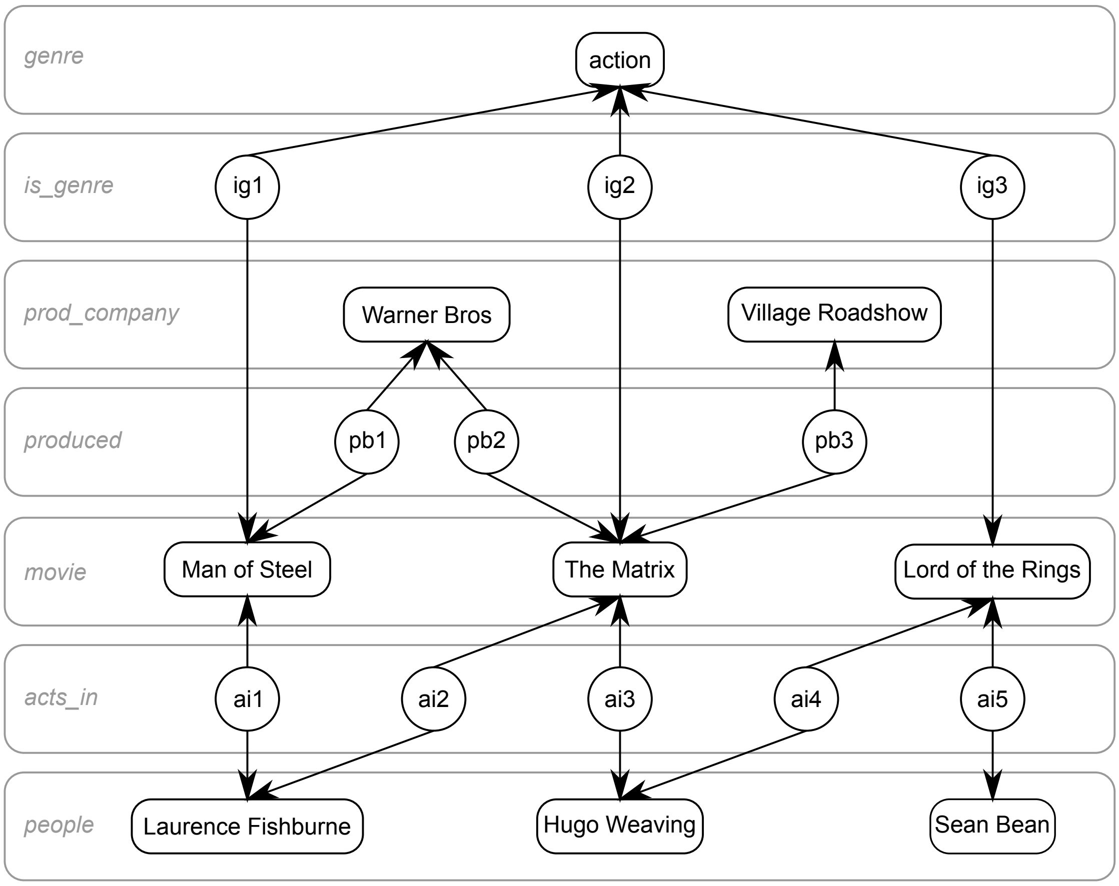


FIGURE 3.9: A graph, G, representation of a small portion of a movie database. The relations the nodes map to are also shown.

keyword queries take various different forms; attributes and terms are typically represented as nodes, edges connect attributes together along primary-foreign keys and to keywords that appear in within them. A solution to a keyword query submitted to a graph-based query system is often defined as a sub-graph that contains all the given keywords.

Figure 3.9 shows the graph, G, which represents a snippet of the movie database found in Appendix A; for illustrative purposes we only show a small section of the graph and actual key values have been replaced with sample data. Adopting the definition of an answer to a keyword query as a sub-graph containing all the terms provided we can assume that the query Q_1*{laurence fishburne warner bros}* has two valid answers:

$\boldsymbol{S_{1.1}}$ *Laurence Fishburne ←ai1 →Man of Steel ←pb1 →Warner Bros*

$\boldsymbol{S_{1.2}}$ *Laurence Fishburne ←ai2 →The Matrix ←pb2 →Warner Bros*

Both sub-graph $S_{1.1}$ and $S_{1.2}$ contain all the given keywords and therefore qualify as solutions to Q_1.

BANKS (Aditya et al., 2002; Bhalotia et al., 2002) matches criteria against textual attributes and defines an answer to a keyword query as a *“rooted tree that connect tuples that match individual keywords in the query”*. The results of a query are presented as a list of ranked *information nodes*, these nodes are chosen to represent the results of the query and can then be further expanded to reveal more information (Figure 3.10). Hyperlinks are used to navigate between nodes representing data from different relations in the database. Although this technique presents an effective means of allowing the user to quickly view the results, while still being able to explore them further, it undoubtedly requires the user to build some understanding of the database structure, something best avoided in novice systems. Ranking in the BANKS system is achieved using a PageRank (Page et al., 1999) style algorithm that applies a prestige value to each node according to the number of edges connecting to it. The system allows some tables to be manually excluded from becoming information nodes, avoiding tables such as *is_genre* or *acts_in* in our sample database (Appendix A) from being used in the results visualisation. Although these tables can be excluded from being presented as nodes in the initial results they must still be navigated through in order to find nodes connected to the information node (in Figure 3.10 the user has navigated through the *writes* table to find the author of a paper presented in the results). The use of web-like links to allow users to navigate through structured data in a familiar environment is similar to the earlier work of DataSpot (Dar et al., 1998) which presents a list of answers that can be clicked to view additional information.

The backwards search algorithms used in BANKS were improved upon in BANKS2 (Kacholia et al., 2005) to use bidirectional search; this increases the speed and efficiency with which results could be found, particularly when some criteria match a large number of nodes in the graph. Something that might arise as a

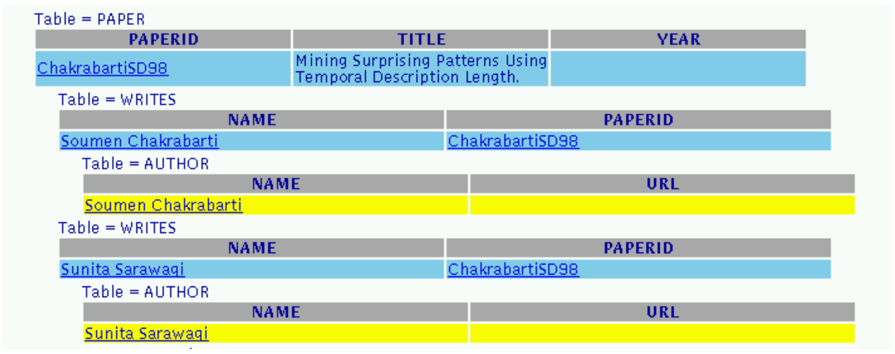


Table = PAPER

PAPERID	TITLE	YEAR
ChakrabartiSD98	Mining Surprising Patterns Using Temporal Description Length.	

Table = WRITES

NAME	PAPERID
Soumen Chakrabarti	ChakrabartiSD98

Table = AUTHOR

NAME	URL
Soumen Chakrabarti	

Table = WRITES

NAME	PAPERID
Sunita Sarawagi	ChakrabartiSD98

Table = AUTHOR

NAME	URL
Sunita Sarawagi	

FIGURE 3.10: The results of the query *"soumen sunita"* in the BANKS system (Bhalotia et al., 2002).

result of the system's ability to match criteria against database meta data such as attribute names.

3.2.2.2 Relational technique

The relational technique of keyword search differs from graph-based approaches in that it does not rely on alternative models of the data but utilises the existing database structure. The aim of such systems is to build query that can be executed directly on the database, in the case of relational databases this means the production of SQL statements suitable for execution on the target database. To build these statements relational systems utilise various indexing structures; these index structures may be generated and maintained by the query system or they may utilise the indexes already in place.

Many commercial products such as MySQL (MySQL, 2012) and Oracle (Oracle, 2015b) include text indexing systems that allow users to submit text queries directly on the database without the use of third party applications. The following is an example of a text query that might be submitted on a movie database such as that described in Appendix A:

```sql
SELECT *
FROM movie
WHERE MATCH (title) AGAINST ('matrix');
```

This query illustrates how, although text indexing can be useful, it still requires an understanding of the underlying database schema; if a relational system were to utilise such an index the system would be required to automatically match keywords to attributes and joins relations where necessary.

Early work that uses the relational technique includes DBXplorer (Agrawal et al., 2002), developed at Microsoft; the main focus of this work was the way in which the system builds an index appropriate for translating keyword queries into SQL SELECT statements. DBXplorer first adds auxiliary tables to the database to store the symbol table (or index) used to process keyword queries. To find answers to a query, DBXplorer uses the symbol table to find all entries in the database that contain the keywords provided; an answer to a query may consist of a single tuple or a series of tuples joined by primary-foreign key relationships.

Another early example of the relational technique is DISCOVER (Hristidis and Papakonstantinou, 2002) which uses a master index to locate the occurrence of the given keywords. Using some understanding of the database schema the system attempts to locate a Minimum Total Joining Networks of Tuples (MTJNT) that contain all the specified criteria. An MTJNT refers to a series of joined tuples that contain all the given terms and cannot be reduced in size by removing any tuples from the join without the loss of some keywords. In an attempt to reduce the likelihood of irrelevant results being presented in the form of a large MTJNT, DISCOVER includes the ability to limit their size to a given value; although this reduces the likelihood of irrelevant results, choosing the maximum allowable size can be difficult (see Section 3.2.5.8). DISCOVER was improved upon in DISCOVER2 (Hristidis et al., 2003) which utilises the advanced free-text indexing available in database management systems to improve the ranking features of DISCOVER and offer Boolean OR functionality in addition to the ANDs that many previous systems support.

The SPARK (Luo et al., 2007) system was designed to improve the ranking systems used in other systems; the identification of a number of problems with existing ranking systems led to a number of improvements in SPARK, which utilises free-text indexes to locate criteria within the database. Two factors considered by

SPARK are the *"completeness"* and the size of a result. The completeness refers to the number of the supplied terms that appear in the results. The need for this metric stems from the potential for systems, such as DISCOVER2 (Hristidis et al., 2003), to rank an answer containing multiple occurrences of one term (and none of the other criteria) over one which contains all the specified criteria but only once. SPARK will prioritise answers containing all the criteria over those with multiple occurrences of a single term. The number of join operations is often used as a rudimentary ranking system, however, Luo et al. identify the potential pitfalls of simply prioritising those answers with fewer joins and present a more advanced alternative. The proposed algorithm not only takes into account the number of joins in the result but also the distribution of the given terms across the results.

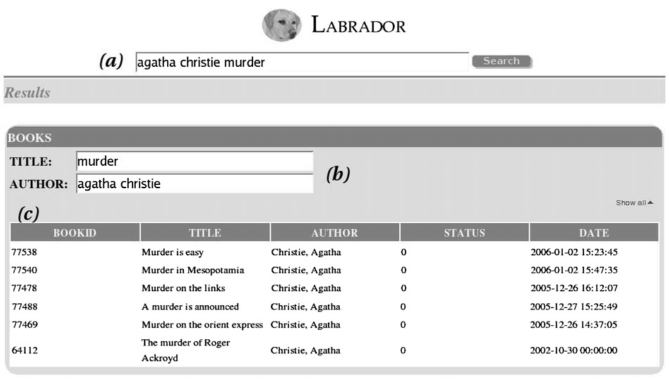
LABRADOR (Mesquita et al., 2007) matches the given criteria, one by one, to attributes within the database. Matching terms to corresponding attributes within the database allows LABRADOR to build a candidate query, for instance the query *Q_2{tom cruise}* might produce the following candidate queries to be considered for execution:

$S_{2.1}$ (name: tom), (name: cruise)

$S_{2.2}$ (name: tom), (title: cruise)

$S_{2.3}$ (title: tom), (title: cruise)

The possible solutions to *Q_2* are then passed to the database management system in the form of SQL SELECT statements and are then executed and ranked according to their perceived relevance. LABRADOR is presented as a system upon which other interfaces might be built; a sample interface is presented (Figure 3.11) in which the user submits keyword queries and then views the structured interpretations generated by LABRADOR along with the ranked results this query produces. The flexibility of LABRADOR means that it can be used to produce a wide range of different query interfaces (an example of browsing possible interpretations using hyperlinks is given) and can utilise a number of different database tools depending



agatha christie murder

BOOKS

TITLE: murder
AUTHOR: agatha christie

BOOKID	TITLE	AUTHOR	STATUS	DATE
77538	Murder is easy	Christie, Agatha	0	2006-01-02 15:23:45
77540	Murder in Mesopotamia	Christie, Agatha	0	2006-01-02 15:47:35
77478	Murder on the links	Christie, Agatha	0	2005-12-26 16:12:07
77488	A murder is announced	Christie, Agatha	0	2005-12-27 15:25:49
77469	Murder on the orient express	Christie, Agatha	0	2005-12-26 14:37:05
64112	The murder of Roger Ackroyd	Christie, Agatha	0	2002-10-30 00:00:00

FIGURE 3.11: An interface built using LABRADOR. The user submits their query (a), views the structured interpretation (b) and the ranked results (c).

upon those available with the chosen RDBMS, for example terms can be matched to attributes using simple equality comparisons or more advanced string operations such as the LIKE or CONTAINS functions.

3.2.3 Ranking

Keyword search, by its very nature, is ambiguous and is liable to return irrelevant results. In an attempt to combat this many systems use various ranking metrics in an attempt to place the most relevant result at the top of the list shown to the user; this process is often referred to as *top-k* ranking. There are various ranking metrics employed by different systems; some use simple techniques such as ranking solutions with the fewest edges or joins (e.g. Agrawal et al. (2002); Dar et al. (1998); Hristidis and Papakonstantinou (2002)) as the most appropriate. Other systems use more complex ways of ranking the results including BANKS which employs algorithms similar to Google's PageRank (Page et al., 1999) to rank the results appropriately.

There is a large amount of work into *top-k* ranking systems (Bhalotia et al., 2002; Ding et al., 2007; Golenberg et al., 2008; Kacholia et al., 2005; Kimelfeld and Sagiv, 2006); some of this work focusses on the implementation of ranking within new keyword systems and others discuss the inclusion of new ranking systems to existing, typically graph based, systems. Often these systems offer high recall and low precision rates; they display almost, if not all, the desired results but these

DBLP	IMDb
Baid et al. (2010); Balmin et al. (2004); Bhalotia et al. (2002); He et al. (2007); Hristidis et al. (2003, 2008); Hulgeri et al. (2001); Kacholia et al. (2005); Khine et al. (2011); Li et al. (2009, 2011, 2008); Luo et al. (2007); Park and Lee (2010); Qin et al. (2010, 2012); Wang et al. (2006); Wheeldon et al. (2004); Xie et al. (2012); Xu et al. (2012); Zeng et al. (2012)	Chaudhuri et al. (2006a); Demidova et al. (2010); Deokmin Haam et al. (2010); Fakhraee and Fotouhi (2012); He et al. (2007); Kacholia et al. (2005); Li et al. (2009, 2008); Luo et al. (2007); Qin et al. (2012); Zeng et al. (2012); Zhou and Pei (2011)

Table 3.1: The use of various datasets in keyword search systems

are hidden by many irrelevant results. Ranking systems are crucial when handling such large result sets as they attempt to display the most relevant results at the top of the list, limiting the negative impact of such large result sets.

3.2.4 Demonstrations and evaluations

There are various methods of evaluating keyword search systems, some of the literature includes attempts to quantify the quality of a system but many simply provide a demonstration. Generally, demonstrating and evaluating a keyword system involves using a given database and executing a series of keyword queries on it; measuring various metrics and drawing conclusions from them.

3.2.4.1 Data sets

Although some systems have requirements regarding the database engine or indexing structure (e.g. Hristidis et al. (2003)) they are generally capable of operating with any database. The DBLP and IMDb are popular datasets for demonstrations and evaluations, Table 3.1 shows how these datasets are used within the literature. These data sets are often chosen because they offer a familiar environment in which to execute queries that users can easily relate to.

3.2.4.2 Performance overhead

Much of the early work concentrated on evaluations based upon a system's performance (e.g. Agrawal et al. (2002); Bhalotia et al. (2002); Khine et al. (2011); Li et al. (2008); Markowetz et al. (2007)) but in light of advancements in both hardware and software recent evaluations attempt to quantify the accuracy of keyword systems focussing less on their performance overhead. Provided a system offers adequate performance it is more reasonable to focus the evaluation of a keyword search system on the accuracy it offers rather than the resources it requires.

3.2.4.3 Precision and recall

Precision and recall values can be used to compare a model result set to one provided by a query system; these metrics are a popular means to quantify the quality of keyword query results (e.g. Chaudhuri et al. (2006a); Fakhraee and Fotouhi (2012); Kacholia et al. (2005); Liu et al. (2006); Mesquita et al. (2007); Patil and Chen (2012); Xie et al. (2012)).

Precision is a measure of the percentage of results returned that are relevant to the given search terms while **recall** refers to the percentage of relevant results returned. The following formulae are used to calculate the two accuracy measures:

$$Precision = \frac{\text{number of correct results returned}}{\text{total number of results returned}}$$

$$Recall = \frac{\text{number of correct results returned}}{\text{total number of correct records}}$$

Systems with low precision values would return many irrelevant results alongside any valid ones; this could mean the user finds it difficult to extract the desired information within the unwanted information. Queries resulting in low recall values would not contain all the correct results from the dataset, this would often result in the user needing to re-query the database to find the desired information.

3.2.5 Problems with existing work

Although there is a large body of work in relation to incorporating keyword search into relational database there are some areas in which many systems are limited. The following highlights a selection of these problems, some of which are applicable to many systems while others only to a few.

- Section 3.2.5.1 discusses the problem of automating join operations in relational-based systems.
- Section 3.2.5.2 addresses the reliance that some applications have on the databases upon which they operate.
- Maintaining graphs and indexes can sometimes be problematic, Section 3.2.5.3 outlines some problems with this.
- Section 3.2.5.4 discusses the potential problems introduced by some graph systems that only support AND operations but not OR.
- Many-many queries occur when two relations, either side of a junction table, are involved in a query, Section 3.2.5.5 highlights a number of problems introduced by these queries.
- Section 3.2.5.6 discusses the need for both Boolean AND and OR operations and the support offered for such queries.
- Keyword should not require an understanding of the database structure, despite this some functionality often relies on this understanding: Section 3.2.5.7.
- Section 3.2.5.8 discusses the potential for graph-based systems to produce undesirable answers as a result of joining nodes together that lack a real-world connection.
- The presentation of results is often overlooked, Section 3.2.5.9 highlights why the results visualisation is important and introduces a number of challenges to overcome.

- Relational databases utilise many different data types Section 3.2.5.10 discusses how various systems handle the processing of different data types.
- Generic search systems can struggle to match bespoke ones that have been specifically designed to handle certain criteria, Section 3.2.5.11 addresses some of the differences in these systems.

3.2.5.1 Joins (in the relational technique)

Join operations are a vital component in querying relational databases, normalisation means that even simple queries can require information from multiple relations. A crucial, and commonly accepted, requirement of novice users is that they should not need to understand the database structure (Section 2.5). In order to prevent the user from needing to understand the database structure keyword systems must handle all join operations automatically.

In graph-based systems that maintain a series of nodes and edges representing tuples and their connections join operations are automated in the process of finding paths that connect the supplied keywords. For instance the query Q_3*{hugo weaving action}*, submitted to the graph G, would find a path between the person *"Hugo Weaving"* and the genre *"action"* through the various intermediate tables. In traversing the graph to find the connections joins are automatically completed by connecting tuples in different relations.

The relational-based systems that utilise various indexing structures must rely on other techniques to find connections between the attributes in which the supplied terms occur. Indexes can be used to find the location of words but they typically offer no knowledge of the database structure. Some relational techniques (e.g. Agrawal et al. (2002)) utilise a schema graph that is used to identify connections between the relations in which the keyword terms appear.

3.2.5.2 Reliance on application specific tools

Some keyword search tools are designed to be very generic implementations that are applicable to a wide range of data models. BANKS2 (Kacholia et al., 2005) can be applied to a wide range of different data stores including relational databases, XML (often connected through IDREF tags) and HTML documents connected with hyperlinks. In contrast, other systems (e.g. Hristidis and Papakonstantinou (2002)) rely on a specific underlying database implementation and utilise functionality limited to the chosen RDBMS. Although systems reliant on a given data structure still offer an interesting insight into solving the problem of keyword search over relational databases, they limit the potential for expansion into other areas.

3.2.5.3 Maintaining data models

Many systems, both graph and relational, build their own models of the data; graph based systems use nodes and edges to represent the data while relational systems used indexes to locate terms within the database. Both of these structures require some degree of maintenance, if records are modified, added or deleted from the databases these models must be changed to reflect this, otherwise the likelihood of erroneous results increases the more out of date the data model becomes.

Manning et al. (2008) suggested that graph based systems are not scalable due to the large memory footprint and the significant maintenance overhead required. Independently constructed indexes that are not maintained by the RDBMS can be considered to have the same problem, ensuring they remain up-to-date involves monitoring changes and adapting the data structure to reflect them. Other than the work that utilises the indexes provided by the chosen RDBMS the maintenance of indexes or graph structures is rarely addressed.

3.2.5.4 OR operations in graph systems

As previously discussed (Section 3.2.2.1) graph-based systems often define an answer to a keyword query as a sub-graph that contains all of the terms provided. This approach effectively applies AND operations between the terms; although this satisfies a wide range of keyword queries the OR operation is not redundant.

The definition that a valid response to a keyword query is a sub-graph containing all the given keywords limits such systems to AND operations only. OR operations are possible only when they can be re-written as AND operations: for example Q_4*{james bond skyfall casino royale}* may represent a query attempting to find films called *"Skyfall"* OR *"Casino Royale"* containing a character called *"James Bond"*. Q_4 would successfully be executed on a graph-based system because both *"Skyfall"* and *"Casino Royale"* have characters called *"James Bond"*. However, consider the query Q_5*{james bond skyfall matrix}*, this query might be interpreted as meaning movies with a character called *"James Bond"* AND a title of *"Skyfall"* OR *"Matrix"*. As in Q_4, there is a movie called *"Skyfall"* containing a character called *"James Bond"*, however, this character does not appear in any movies containing *"Matrix"* in the title. The query Q_5 cannot be re-written to use AND operations alone, one of the conditions fails, in this scenario graph based systems defining an answer as a sub-graph containing all the given terms would fail and not provide a valid answer.

Consider the generic query $Q_6\{x\ y_1\ y_2\}$ in which x is required in the search results along with either y_1 or y_2. The success of a graph-based system correctly handling Q_6 is dependant upon whether $Q_{6a}\{x\ y_1\}$ and $Q_{6b}\{x\ y_2\}$ produce valid responses. If they both produce valid responses then x acts as the root of a tree connecting to both y_1 and y_2. If x cannot be connected to both y_1 and y_2 then there is no sub-graph that will connect all three criteria, representing a failure of such graph-based systems.

This restriction of graph-based systems has the potential to be perceived as inconsistent handling of queries by novice users who have no understanding of the underlying query algorithms.

3.2.5.5 Many-many queries

Many-many relationships, facilitated by junction tables, are common in relational databases; in the movie database (Appendix A) there are four examples of this, the relationship between movies and genres, facilitated by the *is_genre* junction table, is one such example.

We use the term *"many-many queries"* to describe those queries that include relations which have a many-many cardinality between them. An example of such a query might be when a user searches for a movie co-staring two given actors Q_7*{fishburne weaving}*. The relationship between *movie* and *people* is many-many.

The SQL statement to execute such a query cannot be made using Boolean operations alone; this is because there is no single tuple in the joined relations that contains all the required terms. The combination of GROUP BY and HAVING clauses ensures that only movies staring the given two actors are present in the results. The SQL required is as follows[4]:

```
SELECT movie.*
FROM movie INNER JOIN acts_in ON movie.id=acts_in.movie_id
  INNER JOIN people ON people.id=acts_in.person_id
WHERE people.name='Fishburne'
    OR  people.name='Weaving'
GROUP BY movie.id
HAVING COUNT(movie.id)=2;
```

The WHERE clause in the above query requires that the joined tuples must contain at least one of the two given actors while the grouping clause combines multiple results relating to the same movie into a single row. The HAVING clause restricts the output to only those movies that have satisfied two (all) of the conditions in the WHERE clause. If three actors were given then the HAVING clause would be changed to restrict the output to those movies containing all three of the given actors.

The IMDb (IMDb, 2013) uses a separate user interface[5] to allow users to search for movies containing two actors. Two limitations introduced by this approach are accessibility and query limitations. As the many-many query functionality is not

[4]In this sample query the actors are referred to by their last name, in practice they would likely be identified by their full name or ID.

[5]http://www.imdb.com/search/common

integrated within the main search interface the user is required to navigate through the advanced search options to find this functionality. A second limitation is that this user interface can only accept the names of two people; it is not possible to find movies in which three people appear. The approach taken by IMDb shows both a recognition of the need to facilitate such many-many queries and a lack of integration with existing free-text user interfaces.

Recognition and appropriate handling of this type of query is important for free-text systems. If they do not effectively handle these queries they may return no results and if the user has little or no knowledge of the database schema the reason for this might be unclear.

In graph based systems such as BANKS (Aditya et al., 2002) and EASE (Li et al., 2008) these connections would, assuming they exist, be identified. This is because all knowledge of the original schema is discarded in favour of a graph structure that connects related terms, bypassing the need for SQL. Relational systems (e.g. DBXplorer (Agrawal et al., 2002) and DISCOVER (Hristidis and Papakonstantinou, 2002)) that use an index to locate occurrences of keywords must recognise and handle these queries appropriately or they risk returning no results for queries that do have valid answers in the database.

3.2.5.6 Using both AND and OR

The vast majority of keyword search systems only facilitate the AND operation (e.g. Agrawal et al. (2002); Hristidis and Papakonstantinou (2002); Khine et al. (2011); Li et al. (2008); Markowetz et al. (2007); Mesquita et al. (2007); Wheeldon et al. (2004); Widom (2005)), that is they require that all the supplied keywords are present to constitute a valid response to a query. By default the NUITS system Wang et al. (2006) applies an AND operation between all the given criteria, however, it includes a *"Boolean mode"* which allows the user to submit queries such as Q_8*{james bond AND (skyfall OR matrix)}*. Although the use of a Boolean mode enabling the user to submit queries such as Q_8 undoubtedly increases the

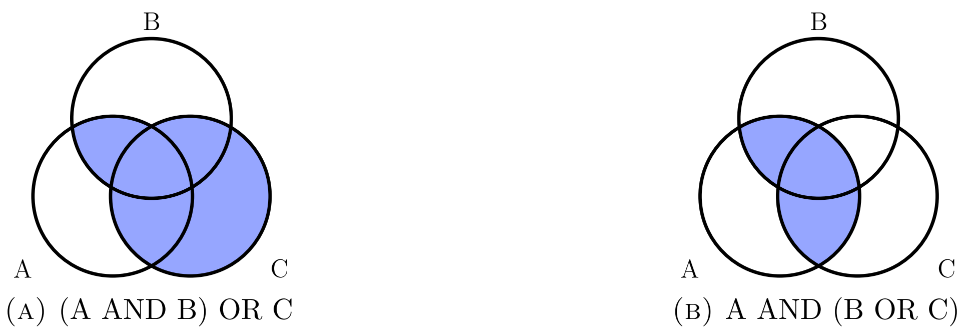


FIGURE 3.12: Two Venn diagrams showing the importance of the position of parenthesis when mixing AND and OR operations. Both diagrams show *A AND B OR C* with different parenthesis positions.

power of a search system it also increases the level of understanding required by the user. It is well recognised that novice users have an incomplete understanding of Boolean expressions (Hildreth, 1989; Nielsen, 1997), so making the use of them the only means to use OR operations seems counter-intuitive when the target audience of such systems is novice users. *"One of the common mistakes users made was to substitute the AND logical operator for the OR logical operator when translating an English sentence to a linear text query"* (Young and Shneiderman, 1993).

The use of a single Boolean operation (AND or OR) between the given criteria allows systems to apply universal rules defining how multiple criteria are handled. Dynamically allocating a mixture of AND and OR operations between keywords requires a more detailed understanding of the contents of the database than many systems are equipped with. Using a mixture of different Boolean operations also requires an ability to group criteria together with parentheses, as is manually defined in Q_8, to ensure consistent handling of Boolean expressions (Figure 3.12).

3.2.5.7 Schema understanding

A crucial requirement of novice users is their need to query a database with no understanding of its structure (Section 2.5). Although this is a well recognised requirement for novices many systems include the ability to submit keyword queries that reference elements of the database schema (e.g. (Bhalotia et al., 2002; Wang et al., 2006; Xie et al., 2012)). Meta-data such as attribute names can often be used to force the system to find the given word in a specified location; for example a query such as Q_9{*name: weaving*} would force the system to locate the term *"weaving"* in the *name* attribute. The most obvious problem with this stems from the very real possibility that a novice user might only be equipped with an incomplete understanding of the database schema. For instance, the name of a person may be stored in a single *name* field or they may be split across two attributes: *last_name* and *first_name*. If the user has only a partial understanding of the database they may either get unexpected results or even no results at all.

3.2.5.8 Unrestricted graph sizes

Sub-graphs defined as answers to keyword queries (Section 3.2.2.1) are designed to connect the supplied terms to form an appropriate response to the query given. Consider the query Q_{10}{*matrix action*} executed on the graph G, there are four sub-graphs that satisfy the query Q_{10}, they are as follows:

$\boldsymbol{S_{10.1}}$ *action ←ig2 →The Matrix*

$\boldsymbol{S_{10.2}}$ *action ←ig1 →Man of Steel ←pb1 →Warner Bros ←pb2 →The Matrix*

$\boldsymbol{S_{10.3}}$ *action ←ig1 →Man of Steel ←ai1 →Laurence Fishburne ←ai2 →The Matrix*

$\boldsymbol{S_{10.4}}$ *action ←ig3 →Lord of the Rings ←ai4 →Hugo Weaving ←ai3 →The Matrix*

Although these sub-graphs are all valid responses to the given query it is clear that $S_{10.1}$ is more appropriate than $S_{10.2}$, $S_{10.3}$ or $S_{10.4}$.

In graph based systems it is often assumed that an answer to a keyword query is any sub-graph which contains all the keywords provided, however sub-graphs can become very large if they are not restricted or managed to some extent. Luo (2009) discusses this problem and states that SPARK does not limit the size of the sub-graph. This could result in a massive sub-graph; consider the many-many relationship between people and movies in the movie database (Appendix A) which is facilitated by the *acts_in* junction table.

Two actors who have no credits on the same movies, *Laurence Fishburne* and *Sean Bean* could be connected through a series of movies as follows: *Sean Bean →ai5 →Lord of the Rings →ai4 →Hugo Weaving ←ai3 →The Matrix ←ai2 →Laurence Fishburne* (i.e. *Laurence Fishburne* co-starred in a movie with someone (*Hugo Weaving*) who co-starred in a movie with *Sean Bean*). These chains across many-many relationships are unrestricted in size and could potentially present problems with respect to results presentation and ranking systems. DISCOVER (Hristidis and Papakonstantinou, 2002) and QUICK (De Virgilio et al., 2012, page 109) include the means to set the maximum size of a candidate network, which prevents the generation of such large sub-graphs but this introduces the problem of giving a value to such a maximum. Ranking answers to queries based upon the number of join operations or edges can help make such results become less prominent in the results, this technique is employed by DISCOVER and DBXplorer (Agrawal et al., 2002).

3.2.5.9 Presentation of results

The presentation of query results is often overlooked, with many systems focusing entirely on the generation of results without considering how to display them (Manning et al., 2008; Park and Lee, 2010). Although rarely discussed, this step is crucial within the process of extracting information from a database:

1. Decide upon the search criteria relating to the desired results
2. Submit the keywords to the search system
3. Review the results and extract the desired information

Without an appropriate results visualisation the user will not be able to extract the desired information from the final result set.

In simple search applications a tabular user interface would be sufficient for displaying results but this is not the case for more complex systems such as those operating over relational databases. In graph based systems the various sub-graphs which are chosen as potential results could consist of nodes representing tuples from different relations in the database. An example of this is Q_{10} which searches for the name of a movie, *"matrix"* and a genre, *"action"* (Section 3.2.5.8). In this scenario there are four possible solutions from three different sets of attributes. This mis-match of different attributes within the results would make them very difficult to display in the same results presentation; for instance a tabular layout could not be used in this setting as the the headings may not apply to all the results. One potential solution to this may be to spread the results across multiple visualisations but this could, particularly in large databases, result in many different windows and, as a result, make it difficult for a user to extract the desired information. The use of multiple results representations would also make it difficult to represent any ranking or ordering functions which have been applied to the results.

An alternative way to overcome this problem might be join all relations in the database together resulting in all headings becoming appropriate for all results. Although this would make it possible to display various different solutions to queries on the same visualisation it has the potential to include vast quantities of irrelevant information, particularly in large databases such as MONDIAL[6], a database of geographical data comprising of 33 relations and used by Luo et al. (2007) and Sagiv (2013).

BANKS (Bhalotia et al., 2002) uses a tree structure (Figure 3.10) to represent the results of a query; this structure very closely follows that of the database and shows an entry for each table in the database. For instance the results of a query relating to two actors would display a movie they co-starred in at the root, followed by two entries in the *acts_in* relation and two in the *people* relation. Although an effective means of displaying the results this approach has two main drawbacks:

- A requirement for schema understanding
- The potential for repetitive browsing actions

[6]http://www.dbis.informatik.uni-goettingen.de/Mondial/

As the results are presented on a visualisation closely related to the database schema the user must have some awareness of it to efficiently find the desired information. Within the context of a movie database a novice user might believe that a movie is linked directly to the actors who starred in it but, when following the structure of the database, they must navigate through the *acts_in* junction table. The potential for repetitive browsing actions is also a problem that might arise when users are repeatedly performing similar queries on a regular basis.

The NUITS system (Wang et al., 2006) uses a graphical representation of the nodes in a graph to display the answers to a query (Figure 3.13). Consider the query Q_{11}*{man of steel hugo weaving}*, the movie *"The Matrix"* and the actor *"Laurence Fishburne"* are used to connect the given criteria, these paths are graphically represented, highlighting the occurrences of the keywords. In an attempt to simplify the visualisation and hide unnecessary information NUITS classifies the nodes as one of two types: *K-nodes* and *C-nodes*. *K-nodes* contain keywords and show some of the contents of the node (relation and attribute names along with the contents of that attribute) while *C-nodes* are used to connect them and are denoted by the relation name alone. As this approach to visualising the results heavily relies on the structure of an answer, which may vary, a single query has the potential to produce a large number of graphs as potential answers. To reduce the overhead of navigating through the results, they are clustered together according to their underlying structure, for instance $S_{10.3}$ and $S_{10.4}$ both represent answers from the same sets of tuples so they would be clustered together.

3.2.5.10 Data types

Relational databases use a wide variety of data types to best represent the data stored within them. These data types can largely be categorised into three groups:

- Textual
- Numeric

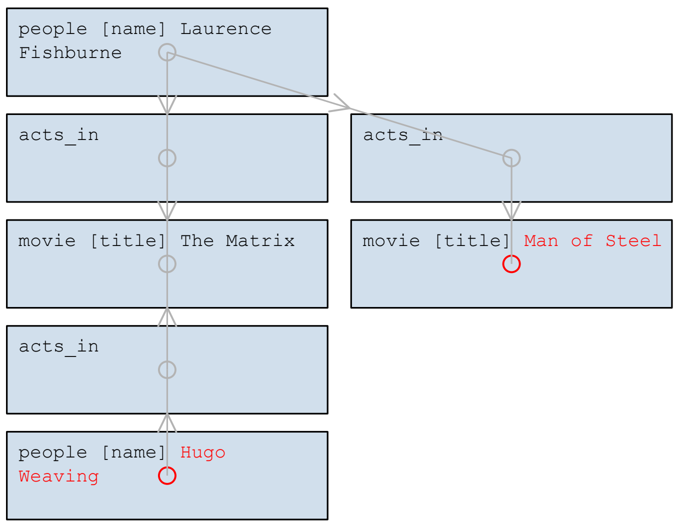

people [name] Laurence Fishburne
acts_in
acts_in
movie [title] The Matrix
movie [title] Man of Steel
acts_in
people [name] Hugo Weaving

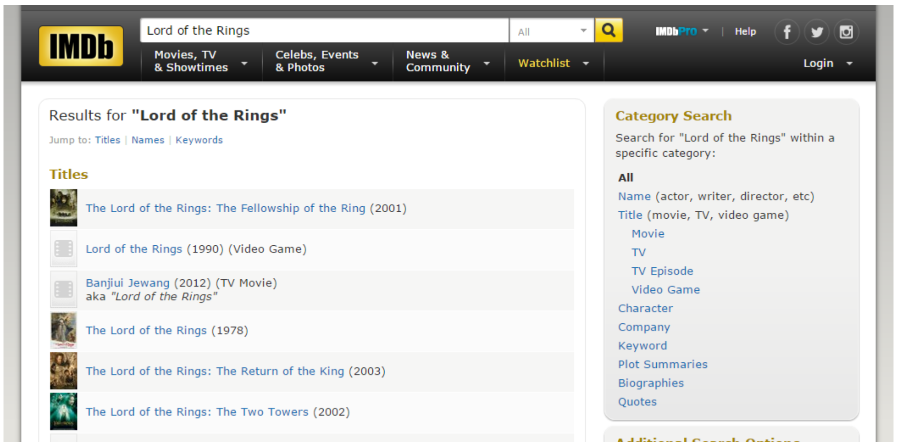
FIGURE 3.13: The tuple-connection tree view in NUITS following a query, Q_{11}, on the movie database.

- Temporal

Although many database systems allow for other data types (e.g. BLOBs), these are unlikely to be used as the criteria for a query so are understandably disregarded from all work we are aware of.

In keyword systems, all criteria is entered as text and the user does not specify whether the criteria should be treated otherwise. It is important for data types to be treated appropriately and not simply as text; this becomes clear when you consider a date attribute within a database. A date may be stored using the *dd/mm/yyyy* format but if a user were to enter criteria using a single digit for a day or month (e.g. 1/12/2015) value then it would be overlooked by the system because it does not match the format the database uses. A keyword system should be able to recognise the different formats of a date and handle them all appropriately, similarly numeric values should be handled appropriately within keyword search systems.

Existing work generally fails to support data types other than text although there are some systems that support numeric input (Chaudhuri et al., 2006b). Some systems (e.g. Qin et al. (2012)) exclusively support textual values and others (e.g. Wang et al. (2006); Xie et al. (2012)) support numeric values but only when an attribute and equality operation is specified, for example $Q_{12}\{year < 1989\}$.



Results for **"Lord of the Rings"**

Jump to: Titles | Names | Keywords

Titles

The Lord of the Rings: The Fellowship of the Ring (2001)

Lord of the Rings (1990) (Video Game)

Banjiui Jewang (2012) (TV Movie)
aka *"Lord of the Rings"*

The Lord of the Rings (1978)

The Lord of the Rings: The Return of the King (2003)

The Lord of the Rings: The Two Towers (2002)

Category Search

Search for "Lord of the Rings" within a specific category:

All
Name (actor, writer, director, etc)
Title (movie, TV, video game)
Movie
TV
TV Episode
Video Game
Character
Company
Keyword
Plot Summaries
Biographies
Quotes

FIGURE 3.14: The results of a search on the IMDb website

3.2.5.11 Generic vs bespoke systems

Generic keyword query systems, the focus of this section, are designed to be applicable to potentially any database (as long as certain structural requirements are met, Section 3.2.5.2). Building a query system for any database means they cannot be optimised for a particular data set; this allows for transferability between databases but lacking the understanding of the database contents can limit the potential of such a system. Bespoke systems that are tuned to meet the needs of a single data set can include an awareness of how to handle certain sets of criteria and how to display the results.

Figure 3.14 shows how the IMDb uses its understanding of the contents of the database to display the results of a query relating to a movie with the title, year of release and a thumbnail of the movie poster. Without an understanding of the contents of the database this style of visualisation is impossible in a generic system.

3.3 Systems for education

Section 2.6 describes the needs of expert users of databases; they typically interact with databases by submitting textual queries in an environment with very few, if any, graphical aids. The reason for this is that the power and flexibility is difficult, if not impossible, to match in a GUI. Although a textual interface is well suited to the expert user it is far from ideal as a means to teach a new generation of experts.

The many problems associated with learning SQL have prompted a large body of work focused on the development of systems designed to ease the process of learning the language. The majority of this work can be divided into two categories: *analytical* and *animated* systems.

In this section we provide an outline of some of the problems encountered when learning SQL, followed by an analysis of the existing work targeted at solving the problems.

3.3.1 Problems faced when learning

As a result of the popularity of the relational model, mastering SQL is an important skill required as part of a Computer Science degree and many related professions. Despite its importance and widespread use across many different applications learning SQL is often a difficult skill for novice users to master.

As applications become increasingly reliant on data (collected by sensors, interactions and people) it becomes crucial that students have a good grasp on the basics of SQL. With Computer Science being added to both primary and secondary school syllabi in the UK (National Curriculum, 2015), the need for effective teaching tools is vital.

3.3.1.1 Declarative nature

The process of learning SQL often takes place as part of a university degree scheme, as part of this students will, most likely, be learning one or more object-oriented or procedural programming languages. In such languages the student describes the steps required to achieve their goal rather than the result of that goal. The following shows a Java method to find the longest runtime from an array of *movie* objects, it describes the steps of checking the various runtimes in-turn and returning the largest value.

```
public int getMaxRuntime(Movie [] movies){
  int longest = 0;
  for(Movie m : movies){
    if(m.getRuntime() > longest)
      longest = m.getRuntime();
  }
  return longest;
}
```

In contrast, the following is an SQL statement to find the maximum runtime from a table of movies:

```
SELECT MAX(runtime) FROM movie;
```

These examples show how the declarative nature of SQL means the user describes the data they want to view rather than the process of obtaining it. Mentally switching from procedural to declarative languages is a recognised problem associated with learning SQL (Matos and Grasser, 2002; Sadiq et al., 2004). Cembalo et al. (2011) identified that students solve procedural problems by breaking them down into smaller steps but *"this approach cannot be followed with SQL, because in a complex query there are no intermediate steps to solve separately, but instead temporary sets of data which result from the execution of the different operators of the same query". "SQL requires learners to think in sets rather than steps"* (Sadiq et al., 2004).

3.3.1.2 Visualising the database/performing joins

Normalising databases means that relational databases frequently consist of many tables that are connected via primary-foreign key relationships. As a result all but the very simplest of queries require joining multiple tables together. To confidently build queries involving join operations users of SQL must maintain a good understanding of the database structure. *"The user has to remember too many things, the names of the record types and attributes have to be remembered before the* *user can express a query"* (Wong and Kuo, 1982). Building and maintaining this mental picture can be difficult for novice users, particularly as they may be exposed to a wide range of schemas throughout their learning process: *"It is a burden for students to memorize the database schema, possibly resulting in erroneous solutions due to incorrect table or attribute names."* (Dekeyser et al., 2007). Without the required level of understanding of the database structure join operators can become difficult and/or confusing for newcomers to SQL (Kearns et al., 1996; Mitrovic, 1998b; Prior, 2003).

3.3.1.3 Help and error messages

SQL is often learnt in a textual, command driven, environment that offers very little help or guidance for new users. Textual interfaces have no awareness of the database structure and frequently lack basic features such as syntax highlighting. This lack of feedback during the query process means that a user may be unaware of any problems in their query until they attempt to execute it (Wong and Kuo, 1982). When an erroneous query is submitted to the RDBMS the user will be met with an error message, these messages are a well recognised problem and frequently offer little insight into the actual cause of a problem or offer a meaningful solution (Cembalo et al., 2011; Mitrovic, 1998b; Prior, 2003; Russell and Cumming, 2004). These features of the interface to SQL can make interacting with relational databases a daunting prospect, particularly for new users.

3.3.1.4 Understanding functions

Many implementations of SQL include a wide array of functions for manipulating, processing and formatting data. An incomplete understanding of these functions can result in the formation of misconceptions regarding their proper use; if the correct use of such functions is not clarified then it may result in users building erroneous queries with little understanding of the reason for such errors. If students extrapolate on these misconceptions it has the potential to lead to queries that do not return the desired results when executed on different datasets (Mitrovic, 1998b).

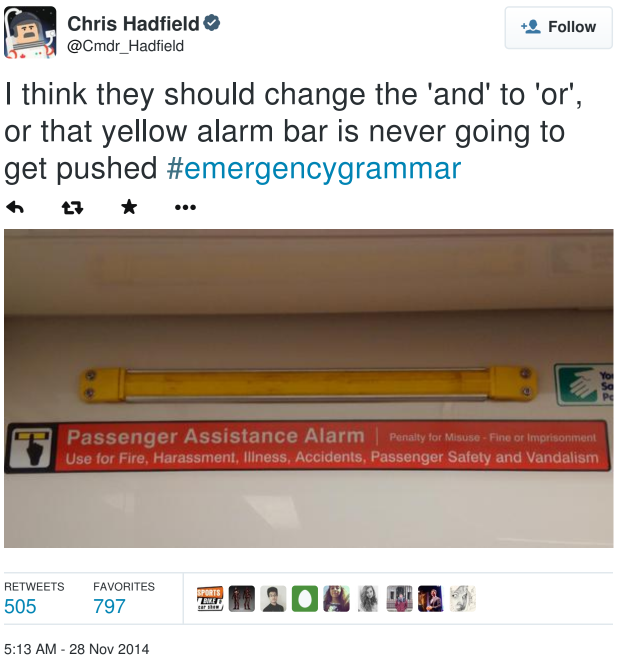
3.3.1.5 Daunting user interface

When students begin to learn a computer language it is often their first exposure to Command Line Interfaces (CLIs). Many students are likely to be familiar and confident in their use of Windows, Icons, Menus and Pointer (WIMP) based user interfaces but the use of CLIs can initially be daunting to new users. Using a CLI, as opposed to a graphical application such as an IDE, can improve a students understanding of low level operations needed to solve computing problems but at the cost of a steeper learning curve (Dillon et al., 2012). Although, in the long term, users can benefit from using CLIs they remain a daunting prospect for many users who are more accustomed to scanning sets of icons to find the desired feature rather than having to remember the appropriate commands.

With the introduction of highly graphical programming environments such as Scratch (Malan and Leitner, 2007; Resnick et al., 2009) even those students with programming experience might have little experience in using CLIs. The use of the command line is something that students must become comfortable with as they develop into expert users; avoiding the command line can appear beneficial at the start of the learning process but it has been shown that the transition from graphical environments to CLIs is significantly more difficult than the reverse (Dillon et al., 2012). This presents a problem for educational institutions, using command line interfaces can result in long term benefits but also has the potential to overwhelm users not used to such an interface.

3.3.1.6 Boolean expressions

Boolean expressions are often considered difficult for novice users to master (Nielsen, 1997), one reason for this is the difference between their use in the spoken language compared to their logical meanings. Young and Shneiderman (1993) summarises



Chris Hadfield
@Cmdr_Hadfield

I think they should change the 'and' to 'or', or that yellow alarm bar is never going to get pushed #emergencygrammar

RETWEETS 505 FAVORITES 797

5:13 AM - 28 Nov 2014

Follow
Passenger Assistance Alarm
Penalty for Misuse - Fine or Imprisonment
Use for Fire, Harassment, Illness, Accidents, Passenger Safety and Vandalism

FIGURE 3.15: This sign reads *"Use for Fire, Harassment, Illness, Accidents, Passenger Safety and Vandalism"* and it illustrates the differences between the spoken word and logical operations. Although the use of AND may be considered correct English this is not be the case in logical set operations where OR would be more appropriatex. (Hadfield, 2014)

this problem as follows: *"One of the reasons for this difficulty is that novice users use terms that they are familiar with (indeed the terms "and" and "or" are used often in natural language); but these terms take on different meanings when used to form a query (Figure 3.15). Thus, when constructing queries in SQL, users tend to make errors because they resort to their knowledge of English. This result was noted in experiments conducted by Boyle et al. (1983); Greene et al. (1990); Michard (1982). One of the common mistakes users made was to substitute the AND logical operator for the OR logical operator when translating an English sentence to a linear text query."*

Building Boolean expressions is an important aspect of creating an SQL query, they are used in the WHERE and HAVING clauses of the SQL SELECT statement and a good understanding of them is crucial in ensuring the desired results are returned. Visualising Boolean expressions can also be difficult for students, particularly when these expressions contain multiple, even nested, sets of parenthesis.

3.3.2 SELECT statement priority

Although SQL consists of many statements that are used for building, manipulating and extracting information from databases the most common statement is SELECT. The SELECT statement has the potential to become more complex than other clauses, as a result, many educational systems focus exclusively on this. Learning the SELECT statement has the additional benefit of including a number of skills that can be transferred to other statements; the WHERE clause is used in many different statements, mastering it within the context of SELECT enables users to utilise these skills in other statements (Cembalo et al., 2011; Danaparamita and Gatterbauer, 2011; de Raadt et al., 2007; Kearns et al., 1996; Mitrovic, 1998b; Prior, 2003; Russell and Cumming, 2004).

3.3.3 Existing techniques

Approaches to software designed to teach users how to build SQL queries can largely be divided into two categories: *analytical* and *animated* systems. Analytical systems are designed to provide feedback or marking based upon a given query while animated systems attempt to graphically represent the steps taken to achieve the results of a query.

3.3.3.1 Analytical

Analytical systems operate by processing a given SQL statement to produce useful feedback, be it a binary right/wrong mark (e.g. Prior (2003); Sadiq et al. (2004)), a more fine-grained score (e.g. Russell and Cumming (2004)) or more in-depth help messages.

SQLTutor (Mitrovic, 1998b) is an early example of an analytical system with the aim of providing detailed and helpful error messages in a personalised environment. The desire to improve error messages within SQLTutor is clear recognition of their, frequently documented, shortcomings (Cembalo et al., 2011; Mitrovic, 1998b; Prior, 2003; Russell and Cumming, 2004). Utilising a series of predefined questions

SQLTutor is aware of the solution to a question a student is attempting to answer; this enables it to present students with challenging questions and provide useful and detailed error messages where appropriate.

Abelló et al. (2008) highlight the fact there are often multiple valid SQL solutions to a given data request. To address this problem Abelló et al. designed LEARN-SQL to focus on analysing the results of a query rather than the SQL used to achieve the results. This allowed students to obtain the correct results using techniques best suited to them without being penalised if their chosen technique differed from that used in the model answer. An example of where this might be useful is the use of implicit and explicit inner join operations; both constitute valid queries but they use markedly different notations.

Many analytical systems such as SQLator (Sadiq et al., 2004), WinRDBI (Dietrich et al., 1997) and AsseSQL (Prior, 2003) use the analysis of queries as support for teachers and course administrators by automatically marking questions. SQLify (de Raadt et al., 2007) extends this further by allowing students to review queries written by their peers, potentially leading to benefits for both the author and reviewer of the query.

3.3.3.2 Animation

Many animation based systems require the input of a textual SQL statement that is then animated in one or more steps.

A common approach, adopted by eSQL (Kearns et al., 1996) and SAVI (Cembalo et al., 2011), is to use a system similar to a programming language debugger. These systems allow users to step through the various stages of a query to observe the impact of each statement until the final results are achieved. An animation is often used for each clause in a SELECT statement but they are frequently displayed in an order different to that which they actually appear in the query; for example the SELECT clause is the first to appear in a query but is often shown towards the end of a series of animations.

The development of eSQL by Kearns et al. (1996) focusses on algorithms used to select a subset of table a table or tables that illustrate a given query. This technique means that users can observe the impact of a query on a small subset of data and apply this understanding to larger, real-world datasets.

3.3.3.3 Commercial applications in education

When faced with the difficulties of learning SQL some institutions resort to the use of commercial applications as a result of a lack of appropriate educational software. Applications such as Microsoft Access[7], HeidiSQL[8] and phpMyAdmin[9] are designed to best facilitate the extraction of information from relational databases and are *"not designed for educational purposes but for the professional management of databases"* (Grillenberger and Brinda, 2012). The use of such applications as a teaching tool can lead to significant problems when students are required to transfer their skills to a textual environment (Mayes and Fowler, 1999; Renaud and van Biljon, 2004). These applications can allow students the ability to use the Query By Example builder to produce the SQL without gaining any understanding of its meaning (Cigas and Kushan, 2010).

3.3.4 Problems with existing systems

Despite the popularity work focused on developing applications to aid students in their learning of SQL there remain a number of common problems within the literature. These include:

- Confusing illustrations
- A significant time overhead for teachers
- The need for a complete SQL statement in animated systems
- Quantifying the quality of a query

[7]http://office.microsoft.com/en-gb/access
[8]http://www.heidisql.com/
[9]http://www.phpmyadmin.net/

```
SELECT t.services, SUM(j.distance) AS milage
FROM journey j, truck t
WHERE t.IDTru = j.IDTru
AND t.model = 'MAN'
AND t.Year > 1990
GROUP BY t.services
HAVING AVG(j.distance) > 50;
```

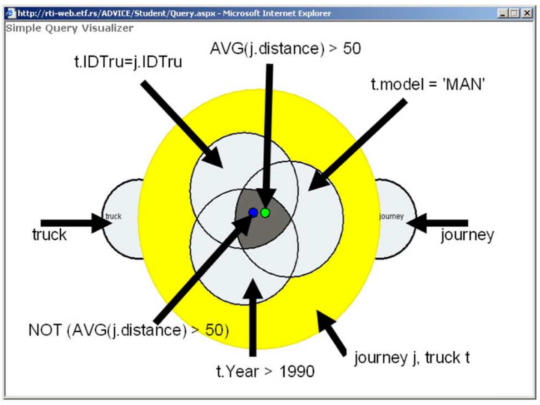


FIGURE 3.16: A query and its representation in the ADVICE educational system.

3.3.4.1 Confusing illustrations

In Section 3.1.5.4 the problem of VQLs becoming so abstract that they become harder to understand than their textual counterparts is addressed. This has the potential to also become an issue in educational systems. Figure 3.16 shows the query visualisation feature within the ADVICE system (Cvetanovic et al., 2011); this effectively illustrates how the concept of an illustration that might seem simple and easy to follow at first can become abstract to the point at which it is unusable.

3.3.4.2 Time overhead for teachers

In order to obtain enough understanding of the user's needs and therefore provide comprehensive feedback many analytical systems rely on the use of predefined banks of questions that students can answer (e.g. Abelló et al. (2008); Allen (2000); Mitrovic (1998b); Sadiq et al. (2004)). Although this leads to increased analytical capabilities it comes at the cost of a significant time overhead for teachers or course administrators who are required to enter the questions and model answers. It could also be argued that, if students only interact with a database to answer a predefined question, it may somewhat limit their desire to explore other areas of the database.

3.3.4.3 Need for complete SQL statements

A key problem with animation based systems is their ability to work only with a fully formed SQL statement. This restriction means that such systems may be more appropriate for demonstrating examples or pre-prepared queries rather than as educational tools for teaching novice users.

Danaparamita and Gatterbauer (2011) appear to have recognised the problem that animated systems require fully formed SQL statements and present QueryViz as a tool for *"novice SQL users to browse through existing repositories and thus intuitively familiarise with the logical patterns behind the SQL syntax"*. Unlike eSQL or SAVI, QueryViz uses a single static illustration, rather than a dynamic animation, to illustrate a SELECT query.

3.3.4.4 Quantifying the quality of a result

Analytical systems rely on their ability to quantify the quality of a query, be it for feedback or marking purposes. There are a number of means to complete this, assuming a model answer is also known. The system can compare either the query string or the results of the query, both techniques have potential pitfalls.

Comparing the query itself may introduce problems as, particularly for more complex queries, there can often be a number of different ways to solve the problem, neither of which are definitively better than the other. Consider a inner join submitted to the movie database (Appendix A), the following two queries both perform identically and return the same results:

```sql
SELECT name, character_played
FROM people, acts_in
WHERE people.id = acts_in.person_id;

SELECT name, character_played
FROM people INNER JOIN acts_in
        ON people.id = acts_in.person_id;
```

The first query uses implicit joins and specifies the connection criteria in the WHERE clause whereas the second explicitly defines the join. Both queries can be considered correct and personal preference often dictates the best choice. If the model answer is specified using one notation and a student submits another they should not be penalised for choosing an alternative method, this makes comparing queries based upon their query string alone difficult.

Abelló et al. (2008) addressed this in the production of LEARN-SQL and compare the results of a query to determine if they offer the same functionality. In the same way that different, perfectly valid, queries might produce the same answer there can also be instances of two queries that would produce identical answers but one technique is favourable over the other. Consider the scenario in which a user is attempting to extract the longest runtime for a movie; the following queries both produce identical results:

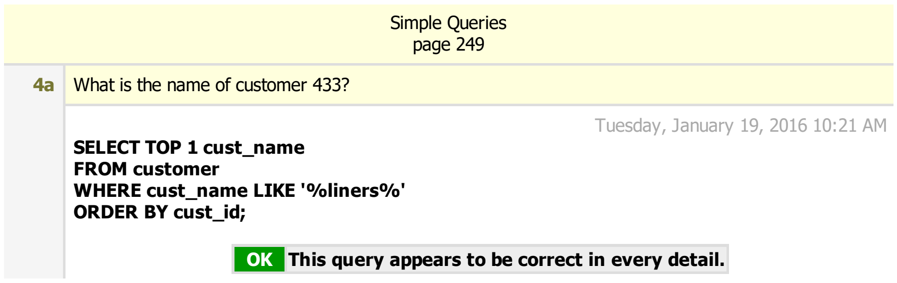
```sql
SELECT MAX(runtime)
FROM movie;

SELECT runtime
FROM movie
ORDER BY runtime DESC
LIMIT 1;
```

Despite the two above queries providing the same, correct, result the first one is generally considered a more desirable solution. In an attribute that is not indexed (as is likely the case with the *runtime* attribute) the first query requires a single pass of the data to locate the lowest value whereas the second query demands the use of a sorting algorithm (e.g. MySQL (2015)) to order the results before only displaying the top one as a result of the LIMIT clause. This example clearly shows that the results of a query alone cannot be used to determine its accuracy. WebSQL (Allen, 2000) is an example of a system that marks the response to a query according to the results alone; Figure 3.17 demonstrates the pitfalls of such an approach and shows a query marked as *"correct in every detail"* by WebSQL despite clear flaws in the solution used.

3.4 Chapter summary

In this chapter we highlighted an overview of the literature relevant to the work presented in this thesis. Visual Query Languages were initially introduced to allow


Simple Queries
page 249

4a What is the name of customer 433?

Tuesday, January 19, 2016 10:21 AM

```
SELECT TOP 1 cust_name
FROM customer
WHERE cust_name LIKE '%liners%'
ORDER BY cust_id;
```

OK This query appears to be correct in every detail.

FIGURE 3.17: WebSQL has marked this query as *"correct in every detail"* whereas it is clearly not the most appropriate way to solve the problem.

users easier access to relational databases but many of them lack the power and flexibility required for expert users and/or the simplicity for novice users.

In light of advancements in both hardware and software capabilities keyword search applications have become more popular. There are two main approaches to keyword search: graph and relational techniques, both allow a set of terms to be submitted and return a set of appropriate results. Much of the work regarding keyword search is limited in its capabilities, this might be as a result of its results visualisation style, requirements of the user or its ability to handle certain search scenarios.

Learning SQL can also be a difficult, but often essential, task for users attempting to become experts in the field. There are a number of systems, both animation and analytical based, that attempt to alleviate some of this burden; analytical systems attempt to offer feedback or marking to the student while animations are used to provide an insight into the steps taken to achieve query results. Accurately analysing queries presents a number of difficulties as there are often multiple approaches to solving a query problem. Animating queries can be hugely beneficial to demonstrate an existing query but is often incapable of helping students in building their own queries.

Chapters 5 and 6 detail two systems designed to tackle the problem of keyword search over relational databases and teaching the SQL language respectively.

Chapter 4

Automating join operations

Join operations are a crucial part of constructing SQL SELECT statements. They allow users to extract data from multiple relations in a database, a common requirement due to the normalisation of data. In this chapter we outline the need for join operations along with difficulties faced by both the user of databases and developers of applications designed to operate in conjunction with them. In a bid to tackle this problem Shorthand SQL (SSQL), a software library that automates join operations, is introduced in a bid to assist developers when building generic database applications.

- Section 4.1 highlights some of the difficulties faced when constructing join operations, both from the user and developer's point of view.
- Section 4.2 presents SSQL as a system to provide an improved understanding of the underlying database structure. SSQL can be used to automate the construction of join operations as well as offering some ambiguity resolution.
- Section 4.3 outlines how SSQL is designed, not as a stand alone application, but as a software library that can be integrated into other applications. To illustrate this point, examples of SSQL being integrated into four different applications are given.

4.1 Join operations

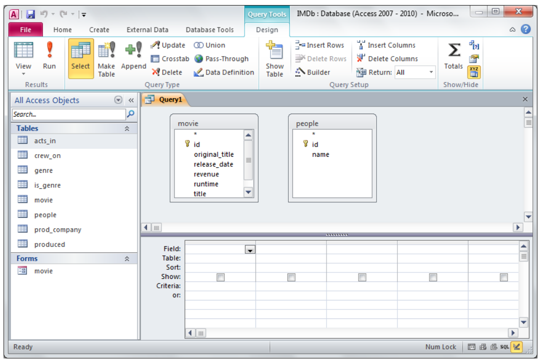
One of the advantages of the relational model over its predecessors, the network and hierarchical models, is that it does not require that the user understands how the information is arranged on the disk (Codd, 1970). Although the relational model successfully avoids the need for actual pointers it introduces foreign keys to connect relations. We can think of these foreign keys as symbolic pointers, linking tuples in one relation to tuples elsewhere.

This need to connect relations together, even to complete relatively simple queries, can be a burden for both end users and developers of database applications. End users are required to build a mental map of the database and developers are required to integrate joins, often on the fly, into their applications.

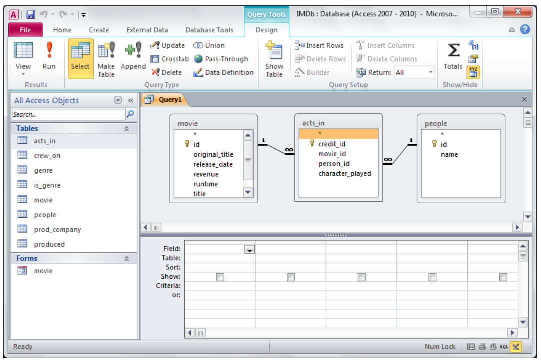
4.1.1 User's perspective

Users who interact with databases using SQL are required to maintain an understanding of the database structure to enable them to build join operations across multiple relations. As discussed in Chapter 3, there are a number of different systems that use various different techniques in an attempt to alleviate the user of this burden. Diagrams can offer an effective means to allow users to visualise the database structure as a whole; some systems such as Microsoft Access provide automatic generation of joins between relations used in a query.

Microsoft Access (and Microsoft Server Management Studio) offers both a diagrammatic representation of the database structure and automatic join operations. Despite this, neither of these functions are entirely automated. The creation of a database diagram relies on the user manually adding and arranging tables one by one, directly related tables are graphically connected with the appropriate links but indirectly related tables are not. Figure 4.1 shows how the intermediate tables must be added to the diagrams to show the links between them; this is also a requirement for the automatic generation of joins. Microsoft Access will join the tables as shown in the diagram, all relations are joined together if present in the



(A) Microsoft Access query designer failing to show links between indirectly connected tables



(B) Microsoft Access showing links between directly connected tables

FIGURE 4.1: Microsoft Access fails to show connections between indirectly connected attributes so the user is required to add junction tables to the diagram to view the connections.

diagram regardless of whether they are used in other elements of the query. If tables are indirectly connected and their junction tables are not included in the diagram no joins will be performed and the resultant output will be a Cartesian product between the given tables.

Although somewhat limited, this automatic generation of join operations undoubtedly provides some greatly needed assistance to many users, as highlighted by Aversano et al. (2002): *"The reason why students perform better when expressing this type of queries [joins] in MS Access seems to be the fact that MS Access graphical query composer automatically generates inner join constructs whenever a foreign key relationship has been previously defined between two tables"*.

4.1.2 Developer's perspective

Although the burden to remember the database structure is significant for end users, it can be even greater for a developer. Developers who build applications that allow novice users to interact with relational databases must develop algorithms that can, on the fly, identify the use of different relations and how they should sensibly be connected.

To automatically generate join operations, such that the end user of the application is presented with the results they expect, demands an understanding of the database structure that can be used to construct join operations in a wide variety of situations. The difficulty of the problem increases in the development of generic applications that are not tailored to a specific database, these should be capable of building joins over potentially huge and complex database schemas.

In application specific software that is designed to operate in conjunction with a given database the developer might use views to avoid the need to repeatedly construct the same join operations. Views act as stored queries and can be built from multiple relations, this means joins can be managed during development rather than at runtime. Although views might offer a viable alternative to managing joins on the fly for bespoke systems they are of little help to generic systems because they require prior knowledge of the database schema.

The Universal Relation Model (URM) has the potential to be more valuable than views to the developer of a generic database application. The URM offers the user a single view of the entire database: *"The universal relation model aims at achieving complete access-path independence in relational databases by relieving the user of the need for logical navigation among relations."* (Maier et al., 1984). The user is then able to query this view without the need to understand the multi-relation structure of the database. Although the exclusive use of the URM would, in most circumstances[1], remove the need for join operations to be calculated it also has some significant drawbacks. One of the main drawbacks stems from the way in which the URM removes many of the benefits introduced with the relational model. Normalisation allows users to query the portion of the database that is of interest to them without interacting with other relations, the use of the URM means that users are constantly interacting with the entire database irrespective of whether their interest is much more focused.

Alleviating the need for developers to manage join operations in their database applications would allow them to focus exclusively on providing the optimum

[1]excluding some less common problems such as self joins

environment for users to extract information rather than also managing the structure of the database.

4.2 Shorthand SQL (SSQL)

Shorthand SQL (SSQL) was developed to remove the need for developers and, in turn, end users to gain an in-depth understanding of the database structure before being able submit queries to it. The functionality of SSQL is twofold, its primary focus is centred around the use of path finding algorithms to enable **automatic joins** between different relations in the database. By gaining an understanding of the database such that the system is capable of automatically traversing connections between relations SSQL is also able to offer some **ambiguity resolution**.

4.2.1 Automating joins using path finding algorithms

To facilitate automatic join operations across a relational database the schema must first be modelled as a graph in which relations are represented by nodes that are connected by edges representing the primary-foreign key relationships between them (Figure 4.2). SSQL considers an edge to be bi-directional, this allows one table to be connected to its neighbour regardless of whether that involves travelling from primary to foreign key or vice versa. In the movie database (Appendix A) this allows a character to be connected to the movie they appear in in the same way that a movie can be linked to the characters that appear in it. Figure 4.2 shows the schema graph generated for the movie database.

To automatically generate joins between any given tables in the database SSQL uses a combination of Dijkstra's (Dijkstra, 1959) and Prim's (Jarník, 1930; Prim, 1957) algorithms. The following steps, along with Algorithm 1, describe how SSQL builds the joins for any given query, to provide some context these steps are given in relation to a query on the movie database that demands the use of three relations: *acts_in*, *prod_company* and *genre*.

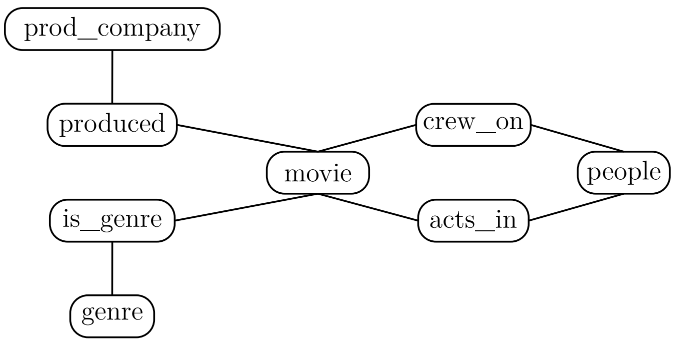
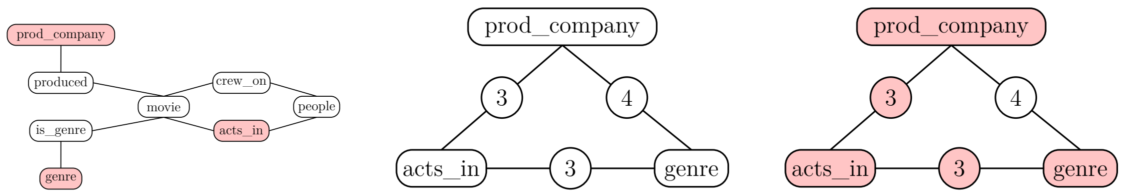


FIGURE 4.2: A graph representation of the schema of the movie database in Appendix A.

```
Require: $graph$                                                    ▷ The schema graph
Require: $relations$                                   ▷ The directly accessed relations
  $prims \leftarrow []$                        ▷ The graph used to perform Prim's algorithm on
  for each $relation1 \in relations$ do
    for each $relation2 \in relations$ do
      if $relation1! = relation2$ then                       ▷ When the nodes are different
        $path \leftarrow graph.doDijkstras(relation1, relation2)$
        $prims.addNode(relation1)$                ▷ Add the node (if it doesn't already exist)
        $prims.addNode(relation2)$                ▷ Add the node (if it doesn't already exist)
        $prims.addEdge(relation1, relation2, path.getLength())$          ▷ Connect the two
      end if
    end for
  end for
  $mst \leftarrow prims.doPrims()$               ▷ Use Prim's to find the minimum spanning tree
  $required\_relations \leftarrow []$           ▷ The set of relations needed to build the query
  for each $edge \in mst$ do
    $required\_relations.addAll(edge.getRelations())$              ▷ Store the used relations
  end for
  return $required\_relations$
```

ALGORITHM 1: How SSQL finds the relations needed in a query given a set of directly accessed relations and a schema graph



(A) Pick the relations to use (B) Build a weighted graph (C) Perform Prim's

FIGURE 4.3: SSQL steps

1. Establish the relations required in the query. This can be achieved through unambiguous notation or through ambiguity resolution (Section 4.2.2). In our example the three relations used are: *acts_in*, *prod_company* and *genre*. (Figure 4.3a).

2. Take the chosen relations from the database and perform Dijkstra's algorithm to find the shortest path between each.

3. Build a new graph with each of the chosen tables as nodes, they are connected with edges with a weight defined by the length of the path calculated in step 2. (Figure 4.3b).

4. Perform Prim's algorithm on the weighted graph to find the minimum spanning tree (i.e. the shortest route that connects all the nodes in the graph). (Figure 4.3c). Any duplicate connections (e.g. that between *movie* and *acts_in*) are only represented once.

4.2.1.1 Dijkstra's and Prim's algorithms

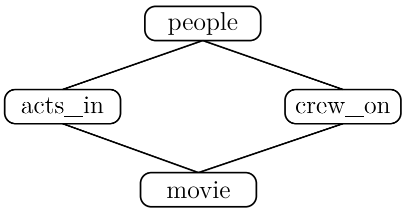
Modelling the database schema as a graph introduces the potential to use many different path finding algorithms. To find the connection between two given nodes (relations) SSQL uses Dijkstra's algorithm (Dijkstra, 1959); the algorithm is simple to implement and is guaranteed to find the shortest path without any prior knowledge of the graph structure or estimates of path lengths. Although other path finding algorithms, such as A*, may offer improved efficiency in certain scenarios these can largely be discounted because the graph only represents the schema, not the data, meaning it is unlikely to be of such a size that this performance difference becomes noticeable, let alone significant.

Although Dijkstra's algorithm alone is sufficient to find the shortest path through a schema to connect two given relations it cannot be used to find connections for three or more given relations. To provide SSQL with this functionality Prim's algorithm (Jarník, 1930; Prim, 1957) is used. SSQL builds a complete graph with the directly accessed relations as nodes and edges connecting every node to every other node (Figure 4.3b). As Prim's algorithm is only ever run on a graph built by SSQL it can be certain that all nodes are connected and a single spanning tree can be found, therefore more advanced algorithms for finding the minimum spanning forest in a potentially disconnected graph are unnecessary. As with the use of Dijkstra's algorithm in SSQL, the size of the graph upon which Prim's will be used is likely to be very small; for n given relations the number of nodes in the graph is n and the number of edges is equal to $\frac{n(n-1)}{2}$. The use of exclusively small graphs in SSQL means that the performance relative to other algorithms is largely insignificant, enabling algorithms to be chosen based on their capabilities and ease of implementation.

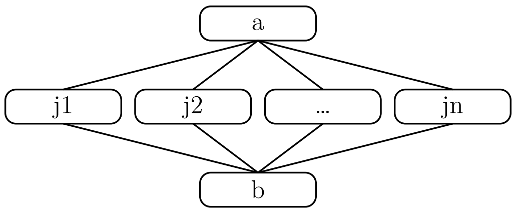
4.2.1.2 Multiple equal length paths

Although SSQL is capable of traversing a large database schema to find the shortest path connecting given relations some database designs can cause problems with the algorithms used. One such scenario is when a decision has to be made between two paths of equal length.

When traversing the schema graph SSQL attempts to identify the smallest join network that contains all the given relations. In many scenarios this is sufficient, however, when equal length paths are found SSQL must pick one. Consider the relationship between the *movie* and *people* relations in the movie database (Appendix A, also shown in Figure 4.4a), people can be associated with movies as actors and/or crew members. A query involving attributes in the *movie* and *people*



(A) Multiple paths in the movie database (Appendix A)



(B) Multiple paths in a generic setting

Figure 4.4: Multiple junction tables between two relations

relations have two possible connections, one via *acts_in* and one via *crew_on*. In its current form SSQL makes no considerations for which might be the most appropriate path and therefore the choice is essentially random.

There are two potential solutions to this problem that future iterations of SSQL could employ: identify the *best* route or use both routes. Choosing the *best* route from a set of equally expensive paths becomes problematic as the system must also maintain an understanding of the database contents that define the most appropriate path. In the movie database example the *best* connection between *movie* and *people* is the one that returns the most appropriate results, for many people one path will return no results while the other will return appropriate results (i.e. many people are cast *or* crew on movies, rarely both). Performing this check at runtime has the potential to be costly, particularly in a generic application that has no awareness of the potential size or indexing of the database upon which it might be operating. Without the system containing a set of hard-coded preferences it would have to execute all possible paths and make a judgement of which is the most appropriate based upon the given results.

An alternative approach is to use all the given paths in a single query through the use of union operations. Consider again the act of joining the *movie* and *people* relations, if a temporary relation were to be used that combined the *acts_in* and *crew_on* relations then the two initial tables could be joined using both possible junction tables. The following query can be used to find all movies *"Laurence Fishburne"* is connected to irrespective of whether the connections is made as an actor or a crew member. The temporary junction table, *u*, is used to combine the appropriate keys in the *acts_in* and the *crew_on* relations.

```
SELECT people.name, movie.title
FROM people INNER JOIN
        (SELECT person_id AS i, movie_id AS o
        FROM acts_in
        UNION
        SELECT person_id AS i, movie_id AS o
        FROM crew_on) AS u ON people.id = u.i
        INNER JOIN movie ON movie.id = u.o
WHERE people.name = 'Laurence Fishburne';
```

In a more generic sense the following is a FROM clause that can be used to connect relation *a* with relation *b* through any number of junction tables (*j1, j2, . . . , jn* as shown in Figure 4.4b). In this example individual junction tables could also be substituted with a series of joined relations if the two initial relations are connected through more than one junction table.

```
FROM a INNER JOIN
        (SELECT j1.in AS i, j1.out AS o
        FROM j1
        UNION
        SELECT j2.in AS i, j2.out AS o
        FROM j2
        UNION
        ...
        SELECT jn.in AS i, jn.out AS o
        FROM jn) AS u
        ON a.out = u.i
        INNER JOIN b ON b.in = u.o
```

4.2.2 Ambiguity resolution

SSQL accepts the input of ambiguous queries, this can arise when using attribute names that appear in multiple relations (e.g. *"id"* or *"name"* in the movie database example). When an attribute is named in a query without the user specifying a relation SSQL will analyse the possible locations for this attribute. When SSQL encounters an ambiguous attribute it will generate every combination of these unknown attributes.

The inclusion of ambiguous attributes inevitably leads to numerous interpretations for single query; SSQL generates all possible interpretations of a query and attempts to rank them accordingly. Firstly queries are ranked according to the number of join operations the query uses, where fewer joins are considered more desirable. This ranking metric can be calculated quickly using the path finding algorithms that automatically generate joins. After the initial ranking it is possible that multiple interpretations will be equally ranked; if the developer allows, these interpretations can be further ranked according to the number of results they return (the more results the higher the ranking). This has the potential to be a computationally expensive operation, depending upon the size of the database and the indexing structures used, for this reason the developer, with an understanding of such characteristics of the database, must enable this functionality.

4.2.3 Sample SSQL queries

SSQL is designed to be used in conjunction with other software applications, as such it does not interact directly with the user but requires another application to act as an intermediary. The following are a selection of queries, shown here as textual SSQL statements, to illustrate the various features and capabilities of the library.

4.2.3.1 From clause generation

Query 4.1 uses the *movie* and *genre* tables. Dijkstra's algorithm is used to connect them through *is_genre* relation. SSQL generates a FROM clause including the intermediate junction table, the resultant SQL query is shown in Query 4.2.

```sql
SELECT genre.name
WHERE movie.title = 'The Matrix';
```

Query 4.1: An SSQL query to find the genre(s) of the movie called *"The Matrix"*

```sql
SELECT genre.name
FROM genre INNER JOIN is_genre ON is_genre.genre_id=genre.id
        INNER JOIN movie ON is_genre.movie_id=movie.id
WHERE movie.title = 'The Matrix';
```

Query 4.2: The SQL translation of Query 4.1, a query to find the genre(s) of the movie *"The Matrix"*

Section 4.2.1 describes a query involving three disconnected tables, Query 4.3 uses the three tables used in the example (*acts_in*, *prod_company* and *genre*). This illustrates the extent to which a query can be simplified and, as a result, alleviating the developer of generating the lengthy FROM clause.

```
SELECT character_played
WHERE genre.name = 'action' AND prod_company.name = 'Warner
   Bros.';
```

Query 4.3: An SSQL query to find all the characters in movies with the genre *"action"* produced by *"Warner Bros."*

```
SELECT acts_in.character_played
FROM acts_in INNER JOIN movie ON acts_in.movie_id=movie.id
        INNER JOIN produced ON produced.movie_id=movie.id
        INNER JOIN prod_company ON prod_company.id=produced.
           prod_id
        INNER JOIN is_genre ON movie.id=is_genre.movie_id
        INNER JOIN genre ON is_genre.genre_id=genre.id
WHERE genre.name = 'action' AND prod_company.name = 'Warner
   Bros.';
```

Query 4.4: The SQL translation of Query 4.3 including all the necessary joins

4.2.3.2 Ambiguity resolution

Within the movie database there are numerous attributes that appear in multiple relations; the *name* attribute is a good example of this, it is present in three relations: *people*, *prod_company* and *genre*. The short query in Query 4.5, therefore has three possible interpretations, each with no join operations and therefore must be ranked according to the number of results they return. Query 4.6 shows the different interpretations in the order deemed most appropriate by SSQL; there are more actors than production companies and more production companies than genres.

```
SELECT name;
```

Query 4.5: An SSQL statement to find a list of names within the movie databse

```
SELECT name
FROM people;

SELECT name
FROM prod_company;

SELECT name
FROM genre;
```

Query 4.6: The SQL translations of Query 4.5 in order from most to least likely

If multiple ambiguous terms are used in the same query the number of interpretations can quickly increase. Using the *name* attribute twice in a query means that both occurrences can be attributed to any one of the three relations in which *name* appears. Query 4.7 has nine possible interpretations, Query 4.8 shows the three that return at least one result.

```
SELECT name
WHERE name = 'Laurence Fishburne';
```

QUERY 4.7: An SSQL query to show the *name* where the *name* is *"Laurence Fishburne"*

```
SELECT people.name
FROM people
WHERE people.name = 'Laurence Fishburne';

SELECT prod_company.name
FROM people INNER JOIN acts_in ON people.id=acts_in.
   person_id
        INNER JOIN movie ON movie.id=acts_in.movie_id
        INNER JOIN produced ON movie.id=produced.movie_id
        INNER JOIN prod_company ON prod_company.id=produced.
            prod_id
WHERE people.name = 'Laurence Fishburne';

SELECT genre.name
FROM people INNER JOIN acts_in ON people.id=acts_in.
   person_id
        INNER JOIN movie ON movie.id=acts_in.movie_id
        INNER JOIN is_genre ON movie.id=is_genre.movie_id
        INNER JOIN genre ON genre.id=is_genre.genre_id
WHERE people.name = 'Laurence Fishburne';
```

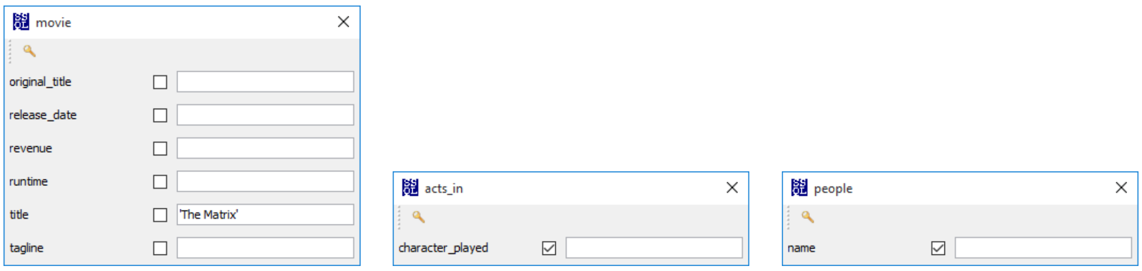
QUERY 4.8: The SQL translations of Query 4.7 in order from most to least likely. For illustrative purposes only those interpretations that yield at least one result are shown.

4.3 SSQL as part of other software

As previously discussed, SSQL is not designed as a stand-alone application but as a means to allow developers to focus on building an application to best suit the needs of the user rather than handling the generation of FROM criteria. Throughout the development of SSQL it has been integrated into a number of different applications to demonstrate its effectiveness. Table 4.1 shows how four applications utilise the various different features of SSQL, these four programs are described in more detail in the following sections:

	QBE-SSQL	T-SSQL	Sandpit Search	CAFTAN
Automatic joins	✓	✓	✓	✓
Ambiguity resolution	×	✓	×	×

TABLE 4.1: Different prototypes that use SSQL


movie

original_title	☐	
release_date	☐	
revenue	☐	
runtime	☐	
title	☐	'The Matrix'
tagline	☐	

acts_in

character_played	☑	

people

name	☑	

(A) Show only the movies with the title *"The Matrix"*

(B) Display the contents of the *character_played* attribute

(C) Display the contents of the *name* attribute

FIGURE 4.5: A QBE-SSQL query to find the list of all the actors in the film *"The Matrix"* along with the characters they played.

4.3.1 QBE-SSQL

Query By Example (QBE) (Zloof, 1975) is well recognised as the first attempt at using graphical representations to ease the process of extracting data from relational databases. QBE-SSQL takes a very similar approach but with the addition of SSQL to automate the joining of the necessary relations.

Each relation is represented by a window containing a row for each attribute within that relation. Each attribute has an associated checkbox and text field, the checkbox is used to specify that the attribute should be shown in the results (instead of the *"P"* notation used in QBE) and the text field is used to enter criteria for that attribute. Figure 4.5 shows three three windows used to build a query to list the actors, along with the characters they portray, in the movie *"The Matrix"*. Figure 4.5a shows the *movie* relation, no checkboxes are ticked meaning none of the attributes are shown in the results and the criteria for the *title* is entered in the text field. Figures 4.5b and 4.5c show the *acts_in* and *people* relations used to request the character and actor names appear in the results.

The following SQL is generated from the graphical representations in Figure 4.5, the SELECT and WHERE clauses reflect the contents of the GUI while the three tables are joined together automatically in the FROM clause.

```
SELECT acts_in.character_played, people.name
FROM movie INNER JOIN acts_in ON movie.id = acts_in.movie_id
        INNER JOIN people ON people.id = acts_in.person_id
WHERE movie.title='The Matrix';
```

4.3.2 T-SSQL

Although graphically the simplest of all systems presented in this chapter, Textual SSQL (T-SSQL) utilises more of SSQL's functionality than the other systems. None of the other systems utilise the ambiguity handling offered by SSQL but T-SSQL demands the use of both ambiguity resolution and FROM clause generation. T-SSQL allows users to submit what are essentially incomplete SQL statements, allowing the system to generate the missing content. Section 4.2.3 demonstrates a number of textual SSQL statements and their SQL counterparts.

T-SSQL is built to resemble a database client such as MySQL, upon logging in and choosing a database the user manually enters queries that, when executed, are processed with SSQL to generate an SQL equivalent. To process a textual SSQL statement it is parsed with CUP (Hudson, 1999) and JLex (Berk, 2003) before passing the relevant information to SSQL for processing. As discussed in Section 4.2.2 ambiguous queries have the potential to produce many interpretations, when an ambiguous query is encountered by T-SSQL all interpretations are executed (in order from most to least likely), showing the results in a tabular layout.

4.3.3 Sandpit Search

The Sandpit Search was initially developed as part of the Elcee application for searching over email data (see Section 2.3.1). To extend the capabilities of Sandpit Search a fully functional working prototype was built that was capable of operating over any relational database. To enable the system to operate in a generic fashion required a number of modifications, to avoid the need for the system to be able to

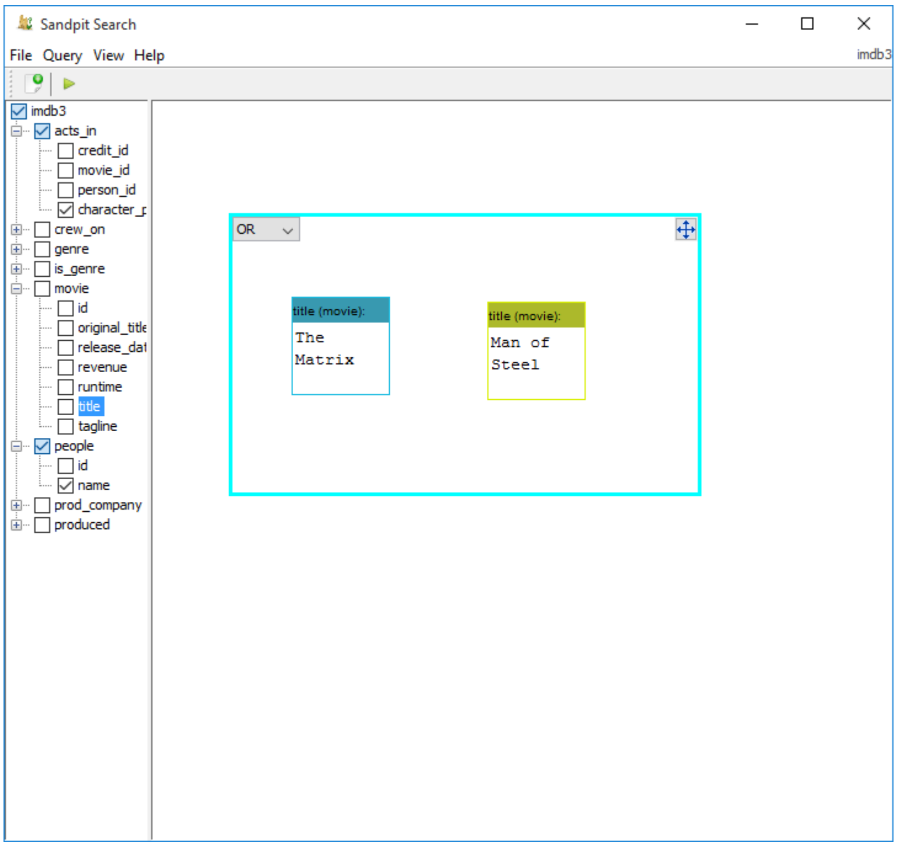


FIGURE 4.6: Sandpit Search

understand the underlying schema it visualises the database as a tree structure that allows the user to explore areas of the database of interest to them while ignoring others. The tree structure also enables the user to tell the system which attributes they wish to view in the results by ticking the appropriate box next to the attribute name.

Figure 4.6 shows the layout of the generic Sandpit Search that displays a graphical representation of a query requesting a list of actors, and the characters they play, starring in *"The Matrix"* or *"Man of Steel"*. The following SQL is produced from this graphical representation, the widgets and boxes are translated to textual Boolean expressions and the chosen attributes appear in the SELECT clause. The FROM clause is generated automatically by SSQL and the user does not need to specify the linkage criteria.

```
SELECT acts_in.character_played, people.name
FROM movie INNER JOIN acts_in ON movie.id=acts_in.movie_id
        INNER JOIN people ON acts_in.person_id=people.id
WHERE movie.title = 'The Matrix'
        OR movie.title = 'Man of Steel';
```

4.3.4 CAFTAN

Chapter 5 describes the keyword search system Context Aware Free Text ANalysis (CAFTAN) designed to allow novice users to build and submit SQL queries without any understanding of the database schema or query language syntax. CAFTAN uses fine-grained indexes to identify the most likely interpretation of a keyword and SSQL is used to connect all the appropriate relations together (Section 5.4.1.4).

4.4 Chapter summary

In this chapter we reiterate the challenge of building queries involving joins from both end user and developer perspectives. With the focus on developers and, in particular, the challenges they face when building generic database applications, SSQL is presented. SSQL allows developers to outsource the process of managing on the fly construction of joins to a small library included in their code. By modelling the database schema as an undirected graph and utilising two different path finding algorithms SSQL can generate automatic joins between any number of directly or indirectly connected relations. The understanding of the database structure SSQL has also enables it to offer some ambiguity resolution that can aid developers in choosing the most likely interpretation of a query. SSQL has successfully been integrated into four different applications and, in each case, generates a linkage criteria without the need for intervention by either the user or the developer.

Chapter 5

Keyword search (CAFTAN)

Keyword search is an exceedingly popular means of finding relevant information, the popularity of online search engines means that almost all computer users are familiar with the process of submitting relevant keywords in exchange for a series of appropriate results. The following chapter describes Context Aware Free Text ANalysis (CAFTAN), a keyword search system capable of operating over any relational database, and is structured thus:

- Section 5.1 describes the importance of keyword search in relation to the target audience and what they expect from such applications.
- Bespoke search systems can provide highly appropriate responses to queries because they are specifically tailored to meet the needs of a given database. Section 5.2 highlights some challenges that generic systems face when attempting to match the capabilities of customised systems.
- Section 5.3 describes some requirements that generic keyword search systems should endeavour to meet.
- The CAFTAN system is described in Section 5.4 along with descriptions of the indexing structure and algorithms used to produce appropriate query responses.
- Finally, Section 5.5 summarises this chapter and its contributions.

5.1 Why keyword search?

The popularity of keyword search has exploded with the rise of internet search engines in the 1990s. This popularity means that almost all computer users, both novice and expert, are familiar with the concept of submitting relevant keywords to extract desired information from a dataset.

5.1.1 Who is the target audience?

Although keyword search is applicable to all computer users, within the context of relational databases, the novice user (Section 2.4.1) is the primary target audience. Experts typically have precise requirements that demand the formation of textual queries to meet their exact needs whereas novice users have no understanding of how to do this so rely on the automatic processing of terms and the production of results.

5.1.2 What do they expect from keyword search?

A defining characteristic of the novice user is their lack of understanding of both the database structure and query language syntax. Novice users expect a keyword search system to respond to these limitations and provide appropriate responses in relation to all areas of the database.

Novice users expect to be able to submit queries to the database by simply providing a series of related terms without any description about how they should be processed or connected. They expect the response of such queries to take the form of a suitable visualisation that accurately represents the expected results. The results should display only the information relevant to the query, hiding irrelevant information such as meaningless key fields and parts of the database unrelated to the search terms.

5.1.3 Why do they expect it?

The expectation that it should be possible to find relevant information by entering a set of related keywords undoubtedly stems from the ability to do so while searching the web. Although applying keyword search to web documents and relational databases are very different problems, to the uninformed, they can appear the same. These expectations put pressure on developers to produce keyword interfaces to different data structures, including relational databases.

5.2 Bespoke vs generic applications

Section 3.2.5.11 describes the differences between bespoke and generic search applications. They are two very different tasks and developing a generic application to match the capabilities of bespoke software is difficult for a number of reasons, which all stem from the application's awareness of the system upon which it operates. Bespoke applications are aware of likely searches and can prioritise and ignore certain attributes or terms in the query to optimise the results. This also extends to the visualisation of the results, a knowledge of the domain allows carefully crafted results to be presented whereas generic systems must automatically choose the appropriate attributes to show along with an appropriate style and layout.

5.3 Requirements of keyword search

The user requirements of keyword search systems are relatively simple as there are only a limited number of ways in which the user can interact with the database. It is imperative that the requirements of keyword search systems reflect the level of understanding associated with the target audience, novice users (Section 2.4.1.1).

5.3.1 Technical understanding

Building keyword queries should be possible without an understanding of any technical aspects normally associated with querying a relational database. Users must not require an understanding of database design techniques or the structure of the database upon which they are working. It must be assumed that the user has no understanding of the relations, their contents or how they are connected. Similarly no understanding of query languages, their syntax or Boolean operations and how they are used to connect criteria should be needed to use a keyword search system.

5.3.2 Query responses

The response to a query should include data from the appropriate relations, joined together to maintain referential integrity where necessary. The data displayed should avoid synthetic primary and foreign key values used as joining fields and must not simply join all relations in the database together to achieve the result set.

If the response to a query is inaccurate, as is an unavoidable possibility with keyword search, the user should be able to re-query the database for an improved interpretation.

5.3.3 Customisation

In an attempt to match the capabilities of bespoke applications administrators should be able to customise a keyword search system to improve the way in which it handles search terms and displays the results. This customisation should allow for a generic application to match the capabilities of those designed for a specific purpose.

5.4 CAFTAN

Context Aware Free Text ANalaysis (CAFTAN) is a generic keyword search prototype that is capable of operating over any relational database. It uses fine-grained indexes and an awareness of the database structure to enable it to accurately interpret keyword queries and provide results that closely resemble those of bespoke systems. CAFTAN differs from existing work in the field as a result of its ability to:

- Build queries with a combination of AND and OR operations
- Appropriately apply search criteria to different data types
- Provide customised query results that closely resemble bespoke applications

5.4.1 How it works

CAFTAN uses an index that associates a weight value to each occurrence of a term. Weights are used to build an SQL query suitable for execution on the database. To build such statements four steps are used:

1. Construction of the index
2. Utilisation of the index
3. Joining the desired terms using SSQL
4. Adjustments made in response to customisations

5.4.1.1 Weights

Each word in an attribute, in a given tuple, has an associated weight. The weighting is based upon the number of occurrences of a word in the database and, crucially, the relationship it has with other tuples. This value defines the likelihood of a word being matched to a given attribute following the execution of a query.

The importance of integrating an awareness of the database structure into the index becomes clear when you consider the word *"Noah"* within the movie database; this is the name of twenty-six actors, twenty-seven characters and is in the title of two movies within our dataset (Appendix A). If the weighting was based purely on number of occurrences then it might be reasonable to assume that a search made for the word *"Noah"* refers to characters with this name; however this does not take into account the number of movies they appear in. It is important to adjust these weights according to the number of references in other relations as it ensures that more prominent tuples are more likely to be matched against keywords provided; in our movie example this means actors with more credits are more likely to form the results of a query. CAFTAN analyses database relationships and increases the weight accordingly, thus the weight of the term *"Noah"* the title of a movie is increased to 122 as a result of the number of actors and crew appearing in these movies. The weight of *"Noah"* referring to an actor is increased to 94 because of the number of movies these actors have appeared in but the most likely interpretation is calculated to be the movie title.

5.4.1.2 Building the index

The following describes how the index is built, this involves scanning the database contents and meta data.

We store the location (relation, tuple and attribute) of every word in the database. Initially these occurrences have a weight of one; if the same word is found in the same location then the weight is further increased by one. The database is then scanned for primary-foreign key relationships; every time a tuple is referenced in another relation (e.g. a person is referenced in the *"acts_in"* relation) the weight of all this tuple's contents are increased accordingly.

The index has the potential to be very large, for this reason and for simplicity while the system is at prototype stage, it is stored persistently in an SQLite database (Section 5.4.4) and must be rebuilt if records are added or removed from the database (see Section 5.4.5.2).

Algorithm 2 describes how the index is built for a relational database (*database*). The indexing structure can be accessed by supplying a word (*word*) and an attribute (*attribute*); in this case *attribute* refers to the given attribute within a specific tuple.

```
Require: $database$                                            ▷The relational database
  $index \leftarrow [][]$                                      ▷Initialise the empty index
  for each $relation \in ddatabase$ do                         ▷For each relation in the database
    for each $tuple \in relation$ do                           ▷For each tuple in the relation
      for each $attribute \in tuple$ do                        ▷For each attribute in the tuple
        for each $word \in attribute$ do                       ▷For each word in the attribute
          if $index[word][attribute] exists$ then              ▷If an entry in the index already exists
            $index[word][attribute] + +$                       ▷Increment the index value
          else
            $index[word][attribute] \leftarrow 1$              ▷Initialise the index value to 1
          end if
        end for
      end for
    end for
  end for
  for each $relation \in database$ do                          ▷For each relation in the database
    if $relation$ has primary key then
      for each $tuple \in relation$ do                         ▷For each tuple in the relation
        $increment \leftarrow$ count references to t in other relations
        for each $attribute \in tuple$ do                      ▷For each attribute in the tuple
          for $word \in attribute$ do                          ▷For each word in the attribute
            $index[word][attribute] += increment$              ▷Adjust weight for relationships
          end for
        end for
      end for
    end if
  end for
  return $index$                                               ▷Return the populated index
```

Algorithm 2: Building the CAFTAN index

5.4.1.3 Utilising the index

The following describes how the index is used to produce appropriate responses to keyword queries; throughout this description the following terms are referred to:

Definition 5.1 (Search term) *A single criterion extracted from the user input. These often take the form of a single words but can also be temporal or numeric ranges (e.g. "100-200")*

Definition 5.2 (Tuple) *A single record within the database. These are identified by a primary key value or tuple identifier.*

Definition 5.3 (Tuple set) *A set of one or more tuples that can all be retrieved using the same search term(s). The weight of a tuple set is equal to the sum of weights of the tuples within it.*

Definition 5.4 (Weight) *A numeric value that represents the likelihood of a search term or search terms being matched against a tuple or tuple set.*

Definition 5.5 (Strength) *A optional, manually defined, value that is used to adjust the automatically generated weights. These are specified by administrators using their knowledge of the database structure and contents. Strength values are associated with attributes, not tuples.*

Definition 5.6 (Adjusted weight) *The weight value multiplied by the strength value. If no strength is provided the adjusted weight is equal to the weight.*

The fine granularity of the index used by CAFTAN allows it to identify the attribute within a specific tuple in which a word arises. This high level of detail is required to allow the effective use of AND and OR Boolean operations. The use of the index involves building tuple sets which satisfy different search terms. Tuple sets are combined using union and intersection operations to imitate AND and OR operations where necessary, something not possible in some graph based systems (Section 3.2.5.4).

The following, along with Algorithm 3, describes how the index is utilised by CAFTAN to produce accurate free-text queries. To provide some context, this process is illustrated by the example query of Q_{13}*{thriller gary oldman}*.

```
Require: $index$                                              ▷ The CAFTAN index
Require: $stops$                                              ▷ A set of stop words
Require: $strengths$                                          ▷ User defined strengths (optional)
Require: $q$                                                  ▷ The keyword query
  $matches \leftarrow []$                                     ▷ Attributes that match search terms
  $q \leftarrow q$.remove($stops$)                            ▷ Remove stop words
  for each $term \in q$ do                                    ▷ Loop through the search terms
    for each $m \in index[term]$ do                           ▷ Loop through the matches in the index
      $a \leftarrow m.getAttribute()$                         ▷ Get the attribute (and tuple) for this match
      if $a \in matches$ then                                 ▷ If an entry for this attribute already exists
        $matches[a] += m.getWeight() \times strength[a]$
      else
        $matches[a] \leftarrow m.getWeight() \times strength[a]$
      end if
      $matches[a].addSearchTerm(term)$                        ▷ Store a reference to the search term
    end for
  end for
  $tuple\_set \leftarrow []$
  for each $match \in matches$ do
    $tuple\_set[match.getSearchTerms()].add(match)$           ▷ Aggregate by search term(s)
  end for
  $tuple\_set.sort()$          ▷ Sort by the number of search terms and then by accumulated weight
  $i \leftarrow 0$
  $chosen\_tuples \leftarrow []$
  while $!q.empty()$ AND $i < tuple\_set.size()$ do
    if $tuple\_set[i].getTerms() \subseteq q$ then            ▷ If the tuple set contains some search terms
      $chosen\_tuples.add(tuple\_set[i])$
      $q.remove(tuple\_set[i].getTerms())$                    ▷ Remove the matched terms
    end if
    $i++$
  end while
  return $chosen\_tuples$                                     ▷ Return the tuples that make up the SELECT statement
```

Algorithm 3: Querying the CAFTAN index

1. The criteria first has any stop words removed (IDOM, 2002), this ensures only meaningful terms are used in the query. Once stop words are removed the input is separated into search terms. In Q_{13} the search terms are *"thriller"*, *"gary"* and *"oldman"* (no stop words are present in the query).

2. The index is next queried to find all tuples that are referenced by a search term. All tuples extracted from the index have an associated weight (and adjusted weight if strengths are supplied). Table 5.1 shows an extract of the index, the strengths applied are described in Section 5.4.1.5.

Search Term	**Attribute**	**Tuple identifier**	**Weight**	**Adjusted weight**
thriller	genre.name	53	934	1120
gary	people.name	4068	34	40
thriller	movie.tagline	19380	25	12
thriller	movie.tagline	11219	20	10
gary	people.name	4507	18	21
gary	people.name	2048	15	18
gary	people.name	64	14	16
gary	people.name	21163	14	16
oldman	people.name	64	14	16
thriller	movie.tagline	110465	13	6
gary	people.name	3953	12	14
gary	people.name	37932	12	14
gary	people.name	33	11	13
gary	people.name	1077782	11	13
gary	people.name	5501	10	12

TABLE 5.1: The results of querying the CAFTAN index

3. When a single tuple is matched against multiple criteria (e.g. tuple *64* in the *people* relation is matched against both *"gary"* and *"oldman"* in Q_{13}) the weight associated with this tuple is set to the sum of all the contributing weights extracted from the index (Table 5.2). This process effectively applies an AND operation between the contributing search terms.

4. Where tuples are accessed by the same search term(s) we can build a tuple set. In our example the term *"gary"* is matched against many (*Gary Oldman* and 152 others called *Gary*) people, all of these entries make up a single tuple

Terms	**Attribute**	**Tuple identifier**	**Weight**
thriller	genre.name	53	934
gary	people.name	4068	34
gary AND oldman	people.name	64	28
thriller	movie.tagline	19380	25
thriller	movie.tagline	11219	20
gary	people.name	4507	18
gary	people.name	2048	15
gary	people.name	21163	14
thriller	movie.tagline	110465	13
gary	people.name	3953	12
gary	people.name	37932	12
gary	people.name	33	11
gary	people.name	1077782	11
gary	people.name	5501	10

Table 5.2: Search terms referencing the same tuple

set with a weight value equal to the sum of all tuples within it. Table 5.3 shows the tuple sets generated by Q_{13}, the tuple sets are ordered first by the number of terms and then by their weight.

Terms	**Relation**	**Weight**	**No. of tuples**
gary AND oldman	people	28	1
thriller	genre	934	1
gary	people	556	152
thriller	movie	58	3
gary	acts_in	36	36
gary	prod_company	6	1
oldman	acts_in	1	1

Table 5.3: Applying a union operation between tuple sets

5. The tuple sets are analysed in turn to assess their eligibility for inclusion in the final SQL statement. If a tuple satisfied one or more previously unsatisfied search terms it is chosen for inclusion in the final query. Table 5.4 shows the tuple sets chosen for the final query.

Once the interpretations of the search terms are calculated these can be used to extract the tuples containing the relevant information. In the above scenario the WHERE clause for the resolved SELECT statement is as follows:

```
WHERE genre.id = 53 AND people.id = 64
```

Terms	**Relation**	**Weight**	**No. of tuples**
gary AND oldman	people	28	1
thriller	genre	934	1

TABLE 5.4: The final tuple sets chosen: there is one actor called *"Gary Oldman"*, and one genre called *"thriller"*.

If multiple tuple sets are chosen from the same relation an OR operation is applied between them as the use of AND would simply return no results (the potential for AND operations has already been explored in step 3). When multiple relations are referred to, as in our example, SSQL (Section 5.4.1.4) is used to connect the given relations. This provides the following SQL SELECT statement in response to Q_{13}[1]:

```
SELECT *
FROM people INNER JOIN acts_in ON people.id = acts_in.
   person_id
        INNER JOIN movie ON movie.id = acts_in.movie_id
        INNER JOIN is_genre ON is_genre.movie_id = movie.id
        INNER JOIN genre ON genre.id = is_genre.genre_id
WHERE genre.id = 53 AND people.id = 64;
```

5.4.1.4 SSQL

Chapter 4 described SSQL, a means to automatically generate join operations to connect a set of given relations within a relational database. It uses path finding algorithms (Dijkstra's and Prim's) to determine the shortest possible connection between the given relations. SSQL plays an important role in CAFTAN; the index is used to locate the terms provided and SSQL is used to connect the given relations to form the FROM clause in the SELECT statement. The use of SSQL as a software library allows CAFTAN to focus on locating the appropriate terms and combining them into a Boolean expression, outsourcing the processing of the database structure to SSQL.

5.4.1.5 Customisation

Bespoke query systems have the potential to be customised to provide optimum query interpretations and results presentation. This provides customised systems

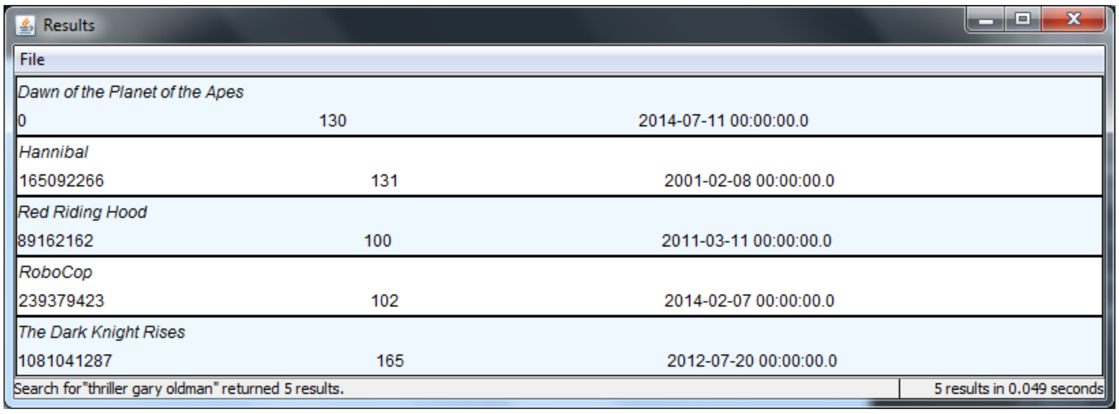
[1]Note: The SELECT clause can be customised by administrators (see Section 5.4.1.5)

with significant advantage when compared with generic interfaces to the same database. CAFTAN introduces a number of customisation capabilities that allow various different aspects to be fine tuned according to the database upon which it is operating. These features are designed to be customised by administrators who have an understanding of the database structure and likely queries, *not* novice users of the system.

Weight strengths In relational databases there are often attributes that store data that can be considered meaningless to many users; examples of this include primary and foreign keys that are primarily used to connect multiple relations and are unlikely to form search criteria for a keyword query. In a generic implementation of keyword search these attributes have the potential to skew interpretations; an example of this within the movie database might be Q_{14} *{100-180}*, most would consider this to refer to movies with a runtime of between 100 and 180 minutes however there is the potential for a generic system to interpret this as a range of IDs rather than runtimes.

CAFTAN includes the ability for administrators to customise weights of various attributes by specifying an associated strength. The strengths are applied when the weights are extracted from the index and they multiply the weights by the given strength value. Attributes such as IDs and those unlikely to be used for search terms can be given a low weight which makes them unlikely to be chosen when interpreting keyword searches and more important attributes can be given higher weights to increase their likelihood of being chosen as query interpretations. The following describes the strengths associated with the movie database (Appendix A):

- All IDs: 0.2 (searches based upon the ID are extremely unlikely)
- Movie tagline: 0.5 (taglines contain many terms that can skew results but rarely constitute search terms)
- Release date, genre name, people's names: 1.2 (these attributes are frequently used as search terms)



```
Dawn of the Planet of the Apes
0                    130                2014-07-11 00:00:00.0
Hannibal
165092266                131                2001-02-08 00:00:00.0
Red Riding Hood
89162162                 100                2011-03-11 00:00:00.0
RoboCop
239379423                102                2014-02-07 00:00:00.0
The Dark Knight Rises
1081041287               165                2012-07-20 00:00:00.0
Search for "thriller gary oldman" returned 5 results.        5 results in 0.049 seconds
```

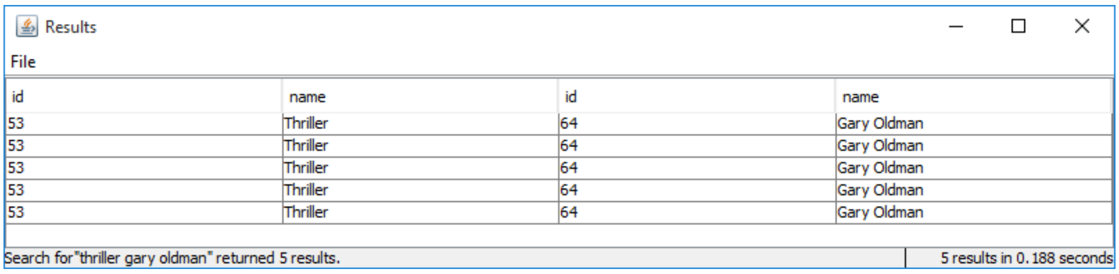
FIGURE 5.1: CAFTAN's customised results view in response to Q_{13}

- Movie title: 1.5 (titles are common search terms)

The strengths are specified in an XML file that is read when extracting data from the index; Appendix B shows the sample XML file used to specify the above strengths.

Customised results visualisation As discussed in Section 3.2.5.9 the presentation of results is an important yet often overlooked element of keyword search. The desired results can often have little in common with the search criteria provided, for instance, Q_{13} described in Section 5.4.1.3 contains three keywords relating to a genre and an actor; despite this it is likely that details about movies relating to the given actor and genre are more desirable than simply displaying the actor name and genre.

Automatically determining the optimal results visualisation for a given query is impossible without some level of human intervention, to meet these particular needs of a keyword search system CAFTAN includes the ability to customise the results for given search criteria. Administrators can specify that when search terms are matched to a given relation or attribute they direct the system to present the results in a customised format. Figure 5.1 shows how the results to Q_{13} have been customised to display the relevant movie information. The customisation allows administrators to specify the attributes to display, a layout and font style along



id	name	id	name
53	Thriller	64	Gary Oldman
53	Thriller	64	Gary Oldman
53	Thriller	64	Gary Oldman
53	Thriller	64	Gary Oldman
53	Thriller	64	Gary Oldman

Search for"thriller gary oldman" returned 5 results.
5 results in 0.188 seconds

FIGURE 5.2: CAFTAN's non-customised results view for Q_{13}

with an ORDER BY clause if necessary. Appendix B shows the XML file used to specify this layout.

If no custom visualisation is specified for the given query a tabular layout is used to display all attributes from all relations in the query (effectively performing SELECT *). This table (Figure 5.2) allows the user to reorder the columns by dragging them and reorder the rows by clicking the associated headings.

5.4.2 Features of CAFTAN

In addition to the core functionality of CAFTAN it includes a number of additional features that separate it from existing work. These features include:

- Handling many-many queries
- Handling different data types
- Managing numeric and temporal ranges
- Re-querying the database using the same criteria

5.4.2.1 Many-many queries

As discussed in Section 3.2.5.5 many-many queries are those which require the use of an OR operation and the HAVING clause to display relevant results. The need for such an approach arises when the cardinality between two relations used in the results is many-many, an example of this may be when the results of a query consist of information from both the *people* and *movie* relations within the movie database.

Consider the query *Q_{15}\{laurence fishburne hugo weaving\}*; both *Laurence Fishburne* and *Hugo Weaving* are actors however simply applying an OR operation between the two criteria would result in the user being shown a list of all the movies staring at least one of the given actors. CAFTAN recognises this situation and will include the appropriate HAVING clause to display the movies staring both actors, if the user requests a re-query then this restriction is removed showing movies staring either actor. As with all elements of CAFTAN, this is implemented in an generic fashion, as a result, it can be applied to any database sharing these characteristics and is not limited to two search terms.

The following represents the resolved SQL for *Q_{15}* where 2975 and 1331 represent the IDs of *Laurence Fishburne* and *Hugo Weaving* respectively.

```sql
SELECT movie.*, people.*
FROM movie INNER JOIN acts_in ON movie.id = acts_in.movie_id
    INNER JOIN people ON people.id = acts_in.person_id
WHERE people.id IN (2975,1331)
GROUP BY movie.id
HAVING COUNT(movie.id) = 2;
```

5.4.2.2 Data types

Relational databases are used to store a wide variety of data, broadly speaking these can be divided into textual, numeric and temporal (date/time) attributes. CAFTAN supports all of these attribute types using the appropriate SQL functions. The entry of a date, *Q_{16}\{19/12/1989\}*, is a good example of CAFTAN's ability to handle input appropriately rather than always as textual values (as in a lot of other systems). In the case of *Q_{16} 19* is recognised as the day, *12* as the month and *1989* as the year, recognising this input as a date allows CAFTAN to utilise SQL functions to extract matching date values. This processing of different data types is unique to CAFTAN with many systems treating all values as text and some completely ignoring non-textual values (Qin et al., 2012).

5.4.2.3 Ranges

In addition to supporting numeric and temporal values, CAFTAN also supports the use of ranges. As with all elements of the system no additional knowledge of the database is required to build queries involving ranges and they can be used in conjunction with other search terms. Ranges are specified using a single dash to separate the lower and upper limit. *Q_{17}{100-180}* specifies a numeric range between *100* and *180*; in the movie database this is associated with the runtime of a movie and returns all those movies with a runtime of between 100 and 180 minutes (inclusive).

5.4.2.4 Re-querying

Ambiguity is an unavoidable problem within keyword search, the query *Q_{18}{Noah}* may be used to refer to movies containing *"Noah"* in the title but the same search term may be used in reference to people called *"Noah"*. Re-queries can be requested using the menu in the results window. Table 5.5 shows the different interpretations of *Q_{18}* and their associated weights. When a re-query is requested the highest weighted tuple set is removed from the list and the process of calculating the appropriate SQL is repeated. In this simple example, the re-query of *Q_{18}* would remove the association of the search term with the title of a movie and the response to the query is recalculated. As a result, *"Noah"* is associated with people names in the first re-query.

Relation	**Attribute**	**Weight**	**No. of tuples**
movie	title	122	2
people	name	94	26
acts_in	character_played	27	27

TABLE 5.5: The different interpretations of *Q_{18}*

5.4.3 Classification of CAFTAN

Section 3.2.2 describes the two main approaches to keyword search over relational databases: *graph* and *relational*. Graph techniques remodel the database contents as a series of interconnected nodes; nodes can represent attributes or keywords and the solution to a query is typically defined as a sub-graph that contains all the supplied criteria. Relational techniques use indexing structures to build an SQL statement suitable for execution on the database.

CAFTAN can primarily be classified as a relational system as a result of its heavy reliance on the index used to locate the most appropriate interpretation of a query. The result of a query submitted to CAFTAN is an SQL SELECT statement that can be executed directly on the database rather than a customised representation of the data that might be used in a purely graph based system. Despite being primarily a relational system CAFTAN also utilises graph structures in the form of a schema graph used by SSQL (Chapter 4).

5.4.4 Implementation

CAFTAN is implemented in Java and operates in conjunction with MySQL databases. Communication between the Java application and MySQL is handled by JDBC meaning that the application could be used with any relational database that provides a JDBC connector with minimal modification.

In its prototype form CAFTAN uses SQLite to store the indexing structure. This provides efficient storage of the large amount of information required for such a fine grained index whilst also allowing for the rapid development of the software. Figure 5.3 shows the simple database structure used to act as the index for CAFTAN, an index is applied to the *word* attribute in the *words* relation to allow for quicker lookups.

As discussed in Section 5.4.1.4 CAFTAN relies on SSQL to build the join operations between the locations of the various terms provided by the user. SSQL communicates independently with the database to construct the graph structures

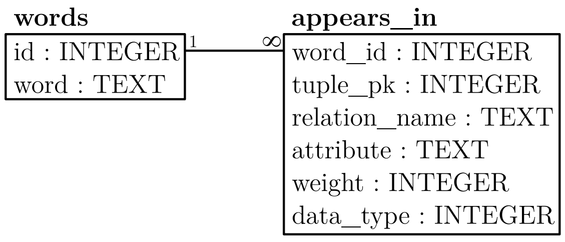


FIGURE 5.3: The CAFTAN indexing structure

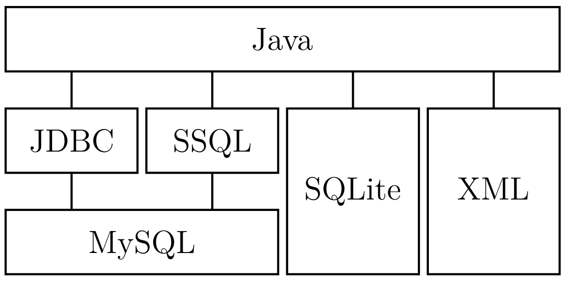


FIGURE 5.4: The CAFTAN prototype's system architecture

required to find connections between different relations. This independence of SSQL means that CAFTAN has no requirement to build and maintain database connections for SSQL and simply passes a request to it and receives the appropriate response.

A highlight of CAFTAN is its ability to allow a number of customisation options in relation to both the searching and results presentation; all customisation options are defined in an XML file, the CAFTAN application reads the customisation files as and when they are needed (Appendix B shows some sample XML files).

Figure 5.4 shows the various elements of CAFTAN and how they are connected.

5.4.5 Limitations

Although CAFTAN offers some significant improvements over existing work in the field of keyword search over relational databases it is not without some drawbacks. Some of these drawbacks are due to the implementation of the prototype and could be eradicated in future iterations of the software while others were design decisions intentionally included in CAFTAN.

5.4.5.1 Non-integer ranges

CAFTAN supports range operations that are defined by the use of a single dash separating the lower and upper bounds of the range (Section 5.4.2.3). This feature enabled CAFTAN to outperform bespoke systems when handling queries such as Q_{19}*{180-240}* which would most likely be interpreted as a range of runtimes in the context of a movie database (see Section 7.1.3 for details of query interpretations). CAFTAN also interpreted such queries to refer to a range of numbers representing the runtime of a movie.

Although this ability to recognise and handle ranges without demanding any specialist knowledge of the database structure from the user sets CAFTAN apart from other systems tested it is only effective when operating on integer fields. To enable effective use of the indexing structure CAFTAN effectively[2] interprets queries such as Q_{19} as:

```
WHERE runtime = 180 OR runtime = 181 OR runtime = 182 OR ...
    OR runtime = 240;
```

This technique has a number of limitations, it is not applicable to non-integer values and it has the potential to computationally expensive as the difference between the lower and upper limits increases. Both of these problems could be solved by utilising SQL functions instead of interacting CAFTANs index, for instance the query Q_{19} could be rewritten as:

```
WHERE runtime BETWEEN 180 AND 240;
```

The drawback to using such functions, and the reason this approach was not taken in CAFTAN, was the risk that such a function might be run on a non-indexed field which would result in a potentially very slow query execution. One of the requirements for CAFTAN was that it would not interfere with the existing database structure, contents or indexes; as a result CAFTAN could not add indexing to improve the performance of functions such as BETWEEN.

[2]The actual CAFTAN interpretation would refer to the IDs of movies with a runtime within the given range.

5.4.5.2 Index maintenance

In its prototype, proof-of-concept, stage CAFTAN has no means to adjust the index as the contents of the database changes (Section 5.4.1.2). The contents of the database are analysed and the index reflects the dataset at the moment it is built but remains unchanged until it is entirely rebuilt. To adjust the index as the database changes a listener (possibly using database triggers) would be required to monitor any changes and apply adjustments to weights and new entries in the index.

Without the ability for the index to reliably represent the contents of the database the potential for CAFTAN to be widely adopted is limited although this does not detract from the benefits the searching algorithms in CAFTAN can bring. Section 7.1.5 shows that, although CAFTAN does not adapt the index to reflect a changing database it can be built from scratch relatively quickly.

5.4.5.3 Rankings

CAFTAN produces an SQL SELECT statement in response to keywords, the database structure, contents (Section 5.4.1.2) and any manually defined customisations (Section 5.4.1.5). The only ordering applied to the results of the query are those defined by the customisations, if none are specified then the order of the results are not guaranteed (in SQL no assumptions regarding the order of the results should be made when not using an ordering clause). Candidate queries are ranked in order of their perceived relevance but the results of a query are not; in the production of an SQL query CAFTAN demands that all results of a query contain all the given terms in at least their associated fields but these results are not sorted further.

To order the results of a query such that the most relevant result is at the top of the list of results is difficult to do without an understanding of the database structure or contents, something CAFTAN attempts to avoid. For instance, in the movie database used throughout this work (Appendix A), the user might want the results of a query for movies to be displayed in order of revenue (i.e. the most successful, well known, movies first). Without an understanding of the meaning behind the database this type of ordering is difficult to automate.

To automatically sort the results of a query a generic application must rely on the information that can be derived from the structure and contents of the database. For instance, it would be possible for CAFTAN to order the results of a query according to the number of connections the results have with tuples in other relations, as is the case in DISCOVER (Hristidis and Papakonstantinou, 2002), or even between the joined tuples. This would result in movies being ordered according to the number of genres, actors and crew members it is related to. Although this rather crude method of ranking would provide some ordering it is heavily reliant on the quality of the data and the number of connections is not always a marker of significance. The movie *Gravity* is a good example of when the number of connections to other relations does not correlate to relevance; the small cast of seven might falsely give it a low ranking despite it being considerably more successful (with a revenue of over $700m) than movies with a much larger cast.

5.5 Chapter summary

This chapter introduces Context Aware Free Text ANalysis, a keyword search system that can be used in conjunction with any relational database. CAFTAN differs from existing keyword search systems in that it supports a number of different criteria formats (different data types and ranges) and allows for extensive customisation to allow it to provide more appropriate results. A primary aim for CAFTAN is to operate without any understanding of the database structure or query language syntax. CAFTAN uses its understanding of the database structure to build SQL SELECT statements that involve Boolean expressions including a mixture of AND and OR operations as well as those spanning multiple relations. Table 5.6 summarises the features of CAFTAN in comparison to other work in this field. To quantify the quality of CAFTAN an evaluation of its performance and how it interprets keyword queries was carried out; this is described in Chapter 7.

			Data types							**Booleans**		
System name	**Reference**	**Graph/Relational**	**Text**	**Number**	**Date**	**Ranges**	**Results ranking**	**Results visualisation**	**Schema knowledge needed**	**ANDs**	**ORs**	**Customisable**
CAFTAN	-	R[3]	✓	✓	✓	✓	×	✓	×	✓	✓	✓
DISCOVER	Hristidis and Papakonstantinou (2002)	R[3]	✓	×[4]	×[4]	-	✓	-	×	✓	×	
DBXplorer	Agrawal et al. (2002)	R	-	-	-	-	×	-	×	✓	×	-
S-KWS	Markowetz et al. (2007)	R	-	-	-	-	×	×	×	✓	×	-
EASE	Li et al. (2008)	G	✓	×[5]	×[5]	-	✓	-	×	×	✓	✓[6]
BANKS	Aditya et al. (2002)	G	✓	×[5]	×[5]	-	✓	✓[7]	×[8]	✓	×	×
-	Li et al. (2011)	G	✓	×[5]	×[5]	×	✓	-	×	×	✓	-
-	Khine et al. (2011)	G	-	-	-	-	✓	-	×	✓	×	-
-	Liu et al. (2006)	G	✓	×	×	×	✓	-	×[8]	×	✓	✓[9]

[3]With schema graph
[4]Limited by interMedia Text (Oracle, 2002)
[5]Treated as text
[6]Different ranking algorithms can be used
[7]Visualisation closely resembles the database structure
[8]Optional
[9] Can generate synonyms to map to schema elements

			Data types							**Booleans**		
System name	**Reference**	**Graph/Relational**	**Text**	**Number**	**Date**	**Ranges**	**Results ranking**	**Results visualisation**	**Schema knowledge needed**	**ANDs**	**ORs**	**Customisable**
NUITS	Wang et al. (2006)	G	✓	✓[10]	×	✓[10]	✓	✓[7]	×[8]	✓	✓[11]	✓[12]
EKSO	Widom (2005)	R	✓	×[13]	×[13]	×	✓	×	×	✓	×	-
DbSurfer	Wheeldon et al. (2004)	R[3]	✓	×	×	×	✓	✓[7]	×	×	✓	×[14]
LABRADOR	Mesquita et al. (2007)	R	✓	×[5]	×[5]	×	✓	×[15]	×	✓	×	×[15]

TABLE 5.6: A comparison of CAFTAN against other published work. A single dash denotes a feature that is not described.

[10]When specified with a given database attribute

[11]When used in *"Boolean mode"*

[12]Some options to customise *"UI elements"* and configuration options

[13]Limited to text by Net Search Extender (IBM, 2015)

[14]Although not directly customisable DbSurfer can be a *"foundation for a customised interface"*

[15]LABRADOR is presented as an API to be used by other systems so has no native UI but can be customised by those systems that use it

Chapter 6

Teaching SQL (SiS)

Learning SQL is well recognised as an important, yet often difficult, skill to master. There are a number of proposed solutions from within both academic and commercial circles. Despite this, it remains a difficult task for many students and there is little uptake of proposed systems designed to alleviate the problem.

In this chapter SQL in Steps (SiS) is presented, a web based application designed to break down the process of building SQL queries into smaller, more manageable, steps that the user can easily understand. The chapter is structured as follows:

- Section 6.1 introduces the concept of a third category of user: *students* who are in a transitional stage between novice and expert users.
- There are many differences between the requirements for educational software packages and those designed for the professional management of databases, Section 6.2 highlights some of these.
- Section 6.3 discusses the potential for using SSQL (described in Chapter 4) in an educational environment.
- A series of requirements for software designed to help teach SQL are outlined in Section 6.4.
- SQL in Steps (SiS) is introduced in Section 6.5. SiS is an educational application that aims to improve a users understanding by allowing them to

graphically build a query whilst maintaining an understanding of its textual counterpart.

- Section 6.7 highlights a number of necessary limitations of SiS and the reasoning behind them.
- SiS is a web based application that operates in conjunction with SQLite. The architecture of SiS is presented in Section 6.6.
- Section 6.8 summarises this chapter and its contributions.

6.1 Students

In Section 2.4.1 two classes of user were defined: novices and experts. Novice users have no awareness of the database structure or query language syntax and often interact with databases through customised applications, such as CAFTAN (Chapter 5), that ensure no specialist knowledge is required to extract information. Expert users are aware of the database structure and interact with databases using textual query languages with little or no assistance from other applications. Users learning SQL are in a transitional stage between the novice and expert users, as a result a third category of user can be introduced:

Definition 6.1 (Student) *A database user transitioning from a novice user to an expert. They might have little/no understanding of database design or query languages at first but they must gain skills to such a level that they can then be classified as experts.*

The process of transitioning from a novice to an expert requires that the user gains an understanding of the theoretical concepts that underpin databases along with the practical skills associated with designing, building, maintaining and using them. The ability to create and modify databases using query languages such as SQL is a crucial part of the learning process, occupying a large portion of many database courses.

6.2 Educational vs professional use

Educational software designed to teach students the concepts and skills required to become an expert database user differs greatly from software designed for the professional management of databases. Despite this, as discussed in Section 3.3.3.3, management software is often used in an educational setting for lack of a more appropriate alternative. The use of such software can appear to benefit students greatly but when required to transfer their skills from a management application to a textual environment they can struggle (Mayes and Fowler, 1999; Renaud and van Biljon, 2004). Students are required to build textual queries because the demands of more advanced queries can quickly outgrow the capabilities of graphical software applications. As a result, it is important that the appropriate software is used at the various stages in the learning process. The reason management applications can ultimately hinder students stems from their differing aims. Broadly speaking, management software is designed to enable users to quickly extract the desired information without using SQL whereas educational software is focussed on teaching users how to form SQL statements independent of any software help.

6.3 SSQL in education

Understanding and building join operations is an important skill as it enables users to extract information from multiple relations, something that is a crucial requirement in normalised databases. Chapter 4 introduced Shorthand SQL (SSQL), a software library that allows join operations to be automatically calculated. If used in an educational system, students could specify the relations they wish to directly access allowing SSQL to automatically generate the missing join operations. This approach has the potential to show users the correct formulation of join operations they might otherwise be unsure of. The use of SSQL within an educational setting has the potential to expose users to correctly formed joins that meet the individual needs of any given query, however this benefit could be outweighed by the disadvantage of students not building their own joins. To fully understand a join operation, as with many components of a query, the student must build them independently with minimal automatically generated content. When using SSQL in an educational setting with a large schema the user might build a query with two directly accessed relations that results in the automatic generation of a series of join operations, the temptation to use the resultant query without a strong understanding of its meaning has the potential to severely hinder the user when they are required to build joins without the assistance of such applications. For this reason, SSQL was not included in the educational work presented in this thesis.

6.4 Requirements

The following are the requirements that the educational software presented here, SiS, aimed to adhere to. They can be divided into three categories: those designed to help specific problems when learning of SQL, those relating to assistance given to students and those relating to the implementation of the application.

6.4.1 Problem based requirements

There are a number of particularly challenging topics within the process of learning SQL, the following are some key problems that SiS aims to tackle.

6.4.1.1 Breaking down problems

Procedural and object-oriented programming languages involve the author describing the steps needed to reach their goal (Section 3.3.1.1). These steps mean that breaking problems down into smaller steps for educational purposes is often simple; declarative languages such as SQL describe the end result rather than the steps to achieve it, making breaking problems down into smaller steps more difficult.

The ability for a user to incrementally build a query would enable students to gain a good understanding of what each component of a query does, decreasing the likelihood of them forming misconceptions regarding various areas of SQL. This potential benefit of an incremental query builder was identified by Wong and Kuo (1982): *"The chances of formulating a complex query correctly on the first try (or even the first few trials) are slim. When using current languages, there is the fear and doubt as to whether the query is complete or whether some conditions are missing. Users could benefit from a facility that allows them to build a query in a piecemeal fashion with feedback of partial results available to them at any time."*.

6.4.1.2 Visualising the database and building joins

When learning SQL users are likely to be exposed to a number of different databases, varying in complexity and design. SQL demands that users build a mental picture of the database in order to locate the appropriate attributes and build joins. Building joins requires a good overall understanding of the database structure, the textual descriptions provided by many database, and educational, applications can make visualising these connections difficult. To enable students to quickly visualise the database and how the different relations are connected educational software should provide an appropriate description of the database, in a graphical form.

6.4.1.3 Close relationship to textual queries

As discussed in Section 3.3.1.5, learning SQL in a purely textual environment can be a daunting prospect for many new students; the interface offers little feedback or assistance in resolving errors and there are no graphical cues to help users remember commands or schema design. Despite this, using graphical interfaces can make transferring skills to a textual environment difficult (Cigas and Kushan, 2010; Dillon et al., 2012; Mayes and Fowler, 1999; Renaud and van Biljon, 2004). Graphical systems make interacting with databases more accessible to new students however, for educational purposes, they are frequently too detached from their textual counterparts. For example, Microsoft Access allows users to build queries using the graphical query builder and also view the generated SQL but users can easily avoid the textual SQL entirely, beneficial for users uninterested in SQL but potentially limiting for students. An educational system employing graphical components should ensure that the textual translation of a query is both easily visible and accurately represents the query.

In addition to viewing the textual version of a query the system should also encourage the frequent execution of queries so the user can compare their expectations of a query against its actual results. This would help users eliminate any false assumptions or misconceptions about the functionality of their query; furthermore, in an educational setting that uses comparatively small databases there is little downside to frequently executing queries as they are built.

6.4.1.4 Boolean expressions

As discussed in Section 3.3.1.6, mastering the use of Boolean expressions is key part of learning SQL that is often considered difficult for many users. They are used widely throughout SQL, the WHERE and HAVING clauses that are used to filter the results of a query are good examples of this. Software designed to educate users in building SQL statements should include functionality to assist users in the construction of Boolean operations. Although assistance is required it should not automate the process to the extent that the users would be unable to independently build Boolean expressions without the software.

6.4.2 Assistance based requirements

The following requirements of educational software relate to its features that provide assistance rather than those that focus on tackling specific problems relating to learning SQL.

6.4.2.1 Generic application

An educational application must be capable of handing the wide range of databases a student will encounter during their learning process, offering relevant assistance appropriate to each database.

6.4.2.2 Contextual help

Examples are an invaluable resource when learning any computer language (Song, 2015; van Gog et al., 2011). However, if a student is learning SQL with one database and an example is given within the context of another they must first familiarise themselves with the contents and schema of the new database before making full use of the example. Examples provided within the context of the database currently in use can allow them to easily relate to it, benefiting from its content without the need to learn another database structure.

6.4.2.3 New and experienced students

Students learning SQL can have very different levels of understanding when they start the learning process and they can also have significantly different goals. Some users only require enough knowledge to perform simple queries while others might demand a larger skill set enabling them to use the more complex features of the language. Educational software should cater for a broad range of these users, also allowing users who have confidence in a particular area to explore it further without deterring new users.

6.4.2.4 Minimal teacher input

Throughout many Computer Science courses the students are presented with information that often requires further, independent, study to become sufficiently knowledgeable in. Educational software must respond to the potentially large amount of independent learning and allow students to explore the software and the related concepts without the input of a teacher. This concept of *learner directed* education allows students of varying ability to explore the system at their own rate without feeling rushed or held back by their peers (Beale and Sharples, 2002).

Within educational software the demands of a teacher can often extend beyond directly helping students; for instance, AsseSQL (Prior, 2003) and SQLator (Sadiq et al., 2004), amongst others, both require a predefined bank of questions and model answers that can be used for automatic marking. The time overhead associated with producing such questions is potentially huge and a burden that should not be given to teachers. The set-up of educational software should be quick, allowing teachers to adapt the software as the needs of the students develop throughout the course.

6.4.3 Implementation requirements

There are a number of implementation requirements that can impact the success of software, educational or otherwise. The following two requirements relate to the accessibility of educational applications.

6.4.3.1 Specialist hardware/software

Educational software is likely to be used by a wide range of users on a wide range of hardware and software set-ups; many students are equipped with their own computers and do not only complete work in a laboratory environment. Demanding certain requirements of the student's computers is only likely to deter them from using the software, as a result it should be capable of running on all major operating systems without making any unreasonable hardware or software demands.

6.4.3.2 Web based

Web based applications are seeing a recent rise in popularity to reflect the needs of users who require consistent interfaces to software when accessed from any location on a multitude of different devices. Browser based software (e.g. Google Docs, Sheets and Slides) uses various web technologies to allow the applications to be run in almost any browser on almost any operating system. This offers a means to satisfy the requirement outlined in Section 6.4.3.1 by only demanding that a computer can run a modern browser to enable it to run the application. Additional benefits from web based software include the fact that any data required by the application can be stored on the web server and accessed directly rather than requiring that the user stores data on their device.

6.5 SiS

In order to address both the problems of learning SQL in a textual environment and the flaws outlined in the existing literature (as discussed in Chapter 3), SQL in Steps (SiS) (Garner and Mariani, 2015a,b) was developed. The main focus of SiS is to allow the student to build an SQL statement in small steps using a GUI while gaining an understanding of the textual query they are building in the background. The user interface is designed with the textual translation at its heart. Every change made to the user interface prompts a change in the textual translation which, in turn, refreshes the results of the query. By bringing the textual translation of a query to the forefront and avoiding an excessively abstract user interface, SiS addresses the difficulty of transferring from a graphical to a textual environment.

6.5.1 Design process

Beale and Sharples (2002) defined a guide designing educational software; this process was followed closely throughout the design of SiS:

Define the educational aims and objectives Allow novice users to become experts in SQL with a particular focus on the SELECT statement. Students should be able to explore the various components of the statement, incrementally building on their understanding.

Identify the learning needs Areas considered particularly challenging include visualising the database, performing joins and maintaining an understanding of a textual query. Misunderstandings relating to Boolean expressions and the correct use of some clauses must be clarified.

Decide which needs could be addressed by computer Graphically visualising the database structure has the potential to help users in understanding the database structure as well as assisting in building join operations. Graphical representations of many components of a query can be used to improve the accessibility of query production, however, these must be closely coupled with an understanding of textual queries as students must learn to build queries independent of graphical aids.

Determine the general teaching and learning approach SiS is designed to be *learner directed* software for an *individual* to *discover* elements of SQL. *Learner directed* software refers to the software being used without the assistance of a teacher and *discovery* software allows students to explore concepts and practice skills.

Determine the teaching strategy SiS provides *tools and resources* to support students in addition to the resources usually available throughout the databases course.

Choose the teaching components SiS acts as both a *model* and a *learning resource*. Models provide dynamic representations of a complex system, a query, that can be manipulated by the user to reach their goal. A learning resource provides online information to assist students.

Design and test the software SiS was designed and built to fulfil the needs identified. Once developed, some small-scale informal testing was carried out in the form of questions with a selection of first year undergraduate students on the databases course (2013/14 cohort).

Evaluate the entire system After further development in response to some of the students comments a formal evaluation of SiS was carried out involving 101 students from the undergraduate databases course (2014/15 cohort). Details of this can be found in Chapter 7.

6.5.2 Focus on the SELECT statement

SQL consists of many different statements that can be used to create, access and modify the database schema and its contents. The SELECT statement is used to read data from a pre-existing database, it consists of a number of different clauses (typically between two and seven) and can vary in complexity depending upon the individual needs of the user and the database structure. Cembalo et al. (2011); Danaparamita and Gatterbauer (2011); de Raadt et al. (2007); Kearns et al. (1996); Mitrovic (1998b); Prior (2003); Russell and Cumming (2004); Sadiq et al. (2004) all focus primarily on the SELECT statement, as does SQL in Steps. There are a number of reasons for this decision:

- Transferable skills
- Frequency of use
- Potential for complex statements

Mastering the SELECT clause provides the student with a number of transferable skills that can be applied to other statements within SQL. For example, the use of Boolean expressions in the WHERE and HAVING clauses are directly applicable to the UPDATE and DELETE statements and the process of unambiguously referring to attributes using the *<relation>.<attribute>* notation is used throughout the language.

The frequency of use of the SELECT clause is another reason for many systems focusing primarily on this clauses, the SELECT clause is the most common SQL statement. The construction or modification of the schema is rarely needed, once a good database structure is established it can remain unmodified throughout its entire lifetime; conversely, users frequently need to extract information. The ability to confidently and quickly build SELECT statements is an important skill for all expert database users.

The SELECT statement also has the potential to become more complex than other statements. The large number of clauses that can make up the statement along with elements such as handling joins and subqueries means that SELECT statements can become large and complex. Despite the potential complexity of a SELECT clause they have no possibility of permanently modifying the database structure meaning that no SELECT statement submitted, regardless of errors or inaccuracies, will impede the users ability to further work with the database.

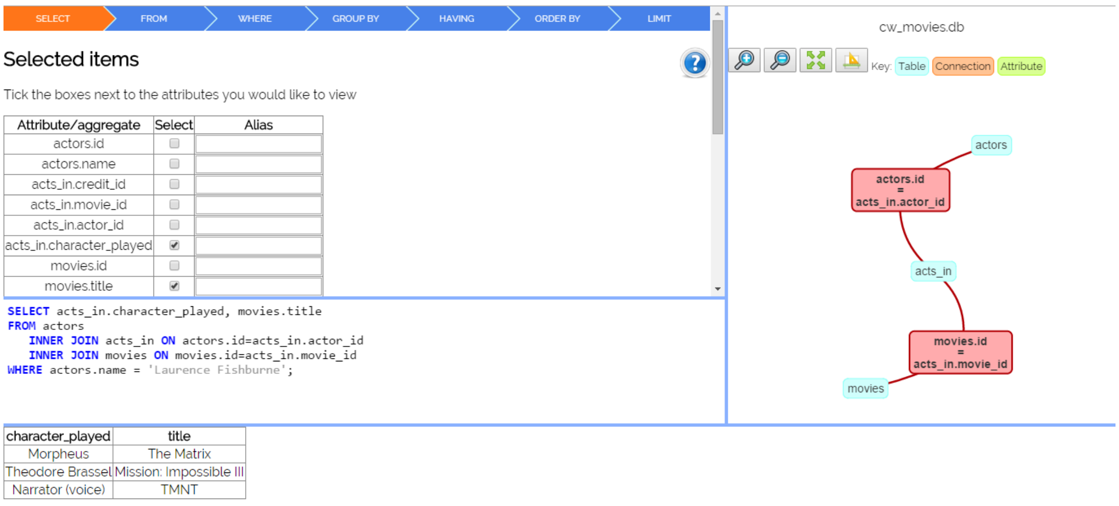
6.5.3 Overview of the SiS user interface

The SiS system is consists of four main panels (clockwise from top left in Figure 6.1):

1. Clause selection and query builder
2. Database visualisation
3. Results panel
4. Textual query

6.5.3.1 Clause selection and query builder

This panel is used to specify the details of the query (all bar the FROM clause). The clauses are arranged in the menu at the top of the panel in the order in which they appear in the textual query ensuring consistency between the GUI and the textual translation. Each clause has a customised UI that is dynamically generated according to the database structure; the UI is intentionally not very abstract to make clear the connection between the graphical and textual versions of a query.


SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT
cw_movies.db
Key: Table Connection Attribute
actors
actors.id = acts_in.actor_id
acts_in
movies.id = acts_in.movie_id
movies

Selected items

Tick the boxes next to the attributes you would like to view

Attribute/aggregate	Select	Alias
actors.id	☐	
actors.name	☐	
acts_in.credit_id	☐	
acts_in.movie_id	☐	
acts_in.actor_id	☐	
acts_in.character_played	✓	
movies.id	☐	
movies.title	✓	

```
SELECT acts_in.character_played, movies.title
FROM actors
   INNER JOIN acts_in ON actors.id=acts_in.actor_id
   INNER JOIN movies ON movies.id=acts_in.movie_id
WHERE actors.name = 'Laurence Fishburne';
```

character_played	title
Morpheus	The Matrix
Theodore Brassel	Mission: Impossible III
Narrator (voice)	TMNT

Figure 6.1: The SiS learning environment

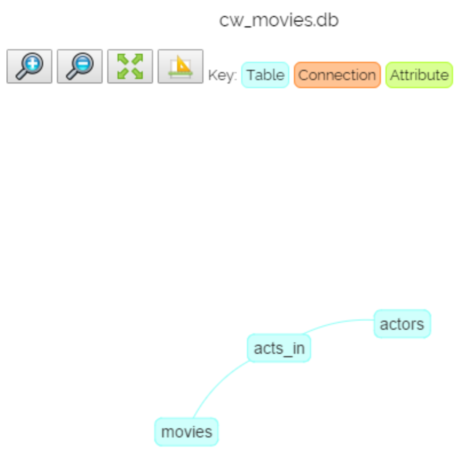
6.5.3.2 Database visualisation

The database visualisation is permanently visible on the right hand side of the SiS window, it shows the different relations within the database and how they are connected. The visualisation can be manipulated by the user, repositioning nodes if necessary, and can zoomed in and out to show more or less detail as required. Figure 6.2 shows the various levels of zoom available, the default (Figure 6.2b) shows relations and the join criteria used to connect them, something often overlooked by students.

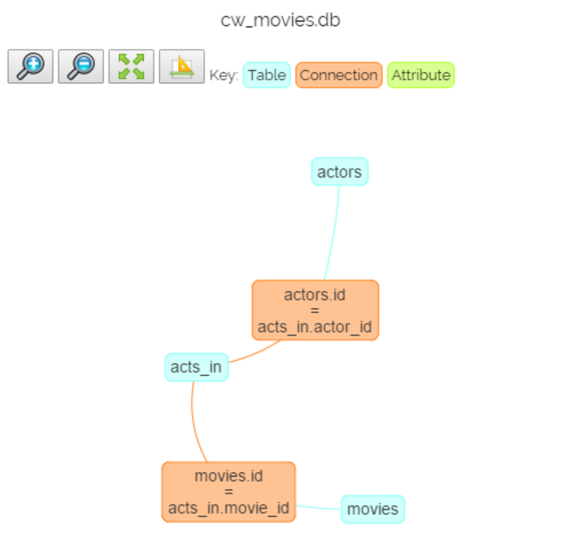
In addition to being used as a visual aid the database visualisation is also used as a means to build a FROM clause. To query a single relation the user can select the appropriate node in the visualisation and to build a join operation the user can select one or more of the connection nodes between relations.

6.5.3.3 Results panel

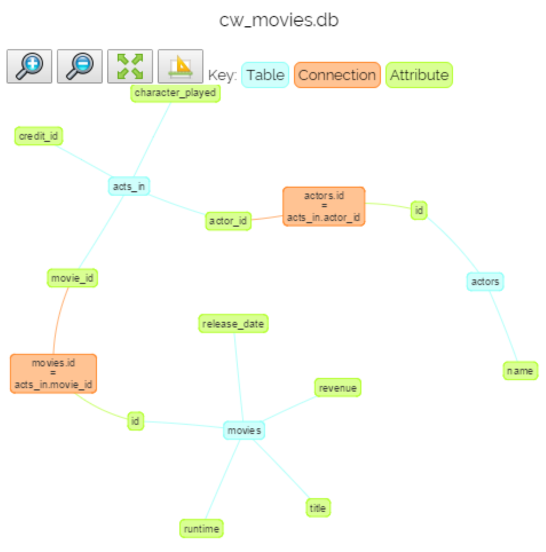
The results panel shows the results, displayed in a table, of the current SELECT statement in the *textual query panel*. These results are updated as the query changes, if the query contains an error or returns no results an appropriate message is displayed (Figure 6.3).



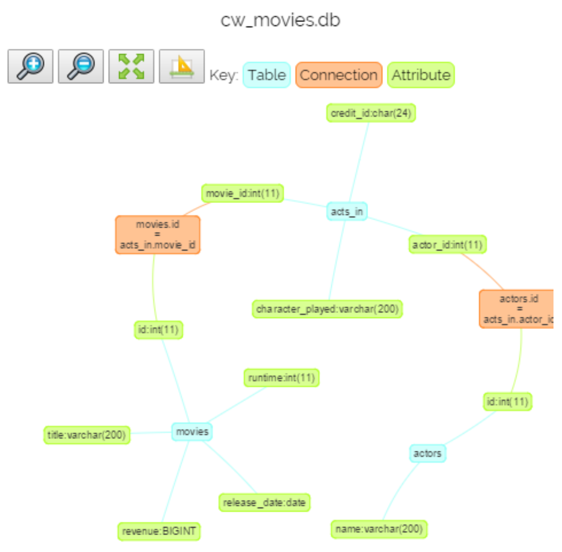
(A) The minimum zoom level showing the relations in the database


cw_movies.db
Key: Table Connection Attribute
actors
actors.id = acts_in.actor_id
acts_in
movies.id = acts_in.movie_id
movies

(B) The database visualisation showing relations and connections (default view)

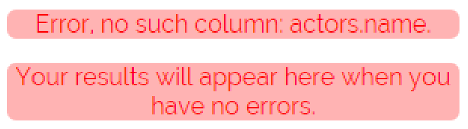


(C) Showing relations, connections and attributes



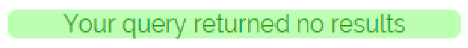
(D) Showing relations, connections, attributes and their data types

FIGURE 6.2: The various levels of zoom on the database visualisation in SiS


Error, no such column: actors.name.

Your results will appear here when you have no errors.

(A) When an error is present in the SQL. The top message is that which is received from SQLite when attempting to execute the query.


Your query returned no results

(B) When the query returns no results

FIGURE 6.3: Messages returned when results are unavailable in SiS

6.5.3.4 Textual query

The textual query in SiS is kept permanently up to date to reflect any changes made to the graphical components of the system. Every change to the query builder is reflected in the textual query which also uses highlighting to show the recently changed parts of the query along with syntax highlighting in an attempt to make the textual query as readable as possible.

6.5.4 Features of SiS

In order to meet the requirements described in Section 6.4 the following key features are included in SiS:

- Textual translations
- Live results
- Visualisation of the database (including join assistance)
- Graphical construction of Boolean operations
- Contextual help and examples
- Features appealing to a wide variety of users
- A not excessively abstract user interface
- Customisation

6.5.4.1 Textual translations

The aim of teaching SQL is to provide students with an understanding of SQL such that they are capable of building textual queries independently; from the students perspective this can initially seem a daunting task. As previously discussed (Section 3.3.3.3), attempts to simplify this process have involved introducing students to graphical systems before transferring to a textual user interface. A significant problem of using graphical systems comes in the transition between a GUI and a text-based system.

SiS attempts to avoid this problem by allowing the user to build a query using a simple user interface while the textual translation remains clear, unavoidable, and up to date. Every change made to the user interface prompts a change to the textual translation and the results the query currently yields. This use of a textual translation in conjunction with live results enables the user to gain confidence in the different clauses of SQL, their syntax and their impact on the results. This evolutionary approach to building queries allows novice users to start with a relatively simple query that can be developed into a more complex SQL statement through small, gradual, changes. This style of building a query in a step-by-step fashion ensures the user is always aware of the textual query and means SiS is the only system (we are aware of) that satisfies the description of a desirable system by Wong and Kuo (1982) in Section 6.4.1.1.

6.5.4.2 Live results

Every change made to a query using the graphical query builder is reflected in a re-execution of the query, prompting an update of the results. This repeated execution of a query as it is built has numerous benefits within an educational environment. Repeatedly executing the query has the potential to reduce the build up of misconceptions or false assumptions that students might make as they write textual queries (Section 6.4.1.1). Ensuring the user is constantly aware of the impact of their query removes the need to make assumptions regarding their query. Allen (2000) identified similar benefits in the development of WebSQL: *"students can easily test assumptions made about syntax"*. SiS extends this concept further by executing queries after every change, this means that students do not need to actively test their assumptions and they can continually analyse the results of a query as it is built. The identification of errors is also improved when the user can clearly see the point at which the error is introduced to the query; error messages are notoriously difficult understand (Cembalo et al., 2011; Mitrovic, 1998b; Prior, 2003; Russell and Cumming, 2004) and identifying the exact location of the cause of the error can greatly ease the process of resolving it. Prior (2003) identified that the way in which students and database professionals build queries differs in that students often write a query in its entirety and then execute it while experts use multiple executions to *"verify the results of preliminary queries"* before refining it. The live results of a query in SiS can promote this refinement style of query production in student users.

6.5.4.3 Visualisation of the database

When learning SQL students may be exposed to a wide variety of different database structures, they may be introduced to more complex schemas as their understanding increases. A common problem faced by many new users is the need to visualise the database schema before querying it; students should not be penalised for failing to remember the structure of a database but should be encouraged to learn how to best utilise the various different schemas they are exposed to. As discussed in Section 3.3.1.2 this is a problem acknowledged in eSQL (Kearns et al., 1996), SQLify (de Raadt et al., 2007) and SQLTutor (Mitrovic, 1998b) by the inclusion of a database description. Although a description of the individual relations in a database provides useful information they offer little assistance in visualising the connections between them.

To familiarise the user with the structure of the database SiS presents them with a graphical representation of the database structure (Figure 6.2). This illustration can be zoomed in and out to discover details such as the attributes within a relation and the ways in which they are connected to other relations. This part of the user interface is not only used as a prompt when the user is interrogating the database but is also used as a means to build the FROM clause of a query. The use of this visualisation, visible at all times, allows the user to quickly build queries and explore the database without studying lengthy textual descriptions.

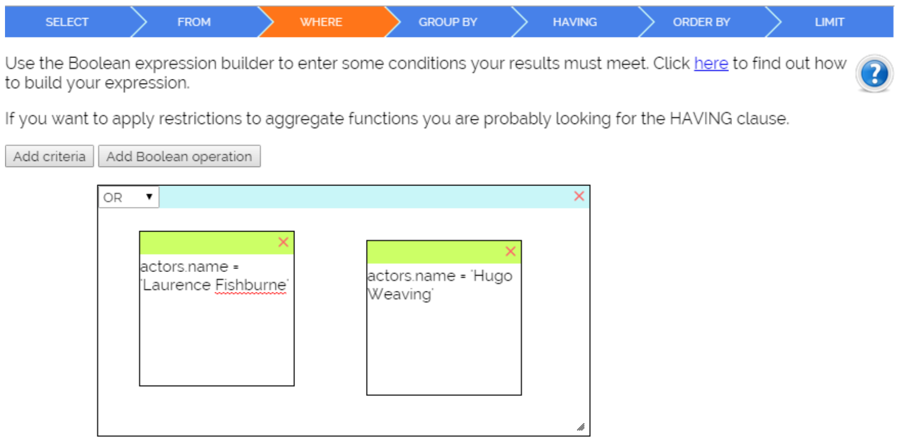
The visual representation of the database structure consists of nodes that represent the different components of the schema and are connected by edges where appropriate. The following nodes are used:

- Tables. Shown in blue at all zoom levels.
- Connections. Shown in orange at all bar the first zoom level. The connection criteria for a join is shown on these nodes.
- Attributes. Shown in green in the last two zoom levels.

Building a query using one table can be done from any zoom level by clicking the appropriate node representing a table in the diagram. To build a query involving a join operation the student can select the orange connection nodes that lie between relations, this can be achieved in all except the first zoom level (Figures 6.2b to 6.2d). When querying an individual relation clicking on a different table node changes the contents of the FROM clause to reflect the new selection whereas join operations across more than two relations can be achieved by selecting more than one connection node.

6.5.4.4 Graphical construction of Boolean operations

Building Boolean expressions is an important aspect of creating an SQL query, they are used in the WHERE and HAVING clauses of the SQL SELECT statement and a good understanding of them is crucial in ensuring the desired results are returned. The understanding of Boolean expressions is something that many novice users struggle with (Nielsen, 1997; Young and Shneiderman, 1993). As discussed in Section 6.4.1.4 the difference in how Booleans are used in logical operations and spoken language can introduce confusion leading to some users using incorrect set operations (Section 3.3.1.6). Visualising Boolean expressions can also be difficult for students, particularly when these expressions contain multiple, even nested, sets of parenthesis.



Use the Boolean expression builder to enter some conditions your results must meet. Click here to find out how to build your expression.

If you want to apply restrictions to aggregate functions you are probably looking for the HAVING clause.

actors.name = 'Laurence Fishburne'

actors.name = 'Hugo Weaving'

FIGURE 6.4: Sandpit Search within SiS showing *actors.name = 'Laurence Fishburne' OR actors.name = 'Hugo Weaving'*.

To allow the user to visualise their Boolean expressions in a more graphical sense SiS introduces a graphical representation of Boolean expressions, the sandpit (Figure 6.4). This component is a continuation of the work developed to enable users to build powerful queries for searching through email data (Section 2.3.1) and acts as a generic interface to any Boolean expression. As with all elements of SiS the sandpit is intentionally closely linked to its textual counterpart; each individual component of a Boolean expression is represented using a single widget on screen, the user can input their criteria on this widget. Individual widgets are combined by drawing boxes around them, these boxes represent any Boolean expressions and their associated parenthesis, boxes can be in an AND or an OR state and can contain a mixture of both widgets and other boxes enabling them to potentially build complex Boolean expressions in a graphical environment closely linked to the textual equivalent. In addition to graphically representing the Boolean expressions the sandpit also includes an auto-complete feature that prompts users with the names of attributes in the database to avoid them making avoidable mistakes by misspelling the attribute names.

The assistance of a graphical environment coupled with live results ensures that students can easily build Boolean expressions and verify them against the results the query yields as they do so.

6.5.4.5 Contextual help and examples

As discussed in Section 6.4.2.2, examples and demonstrations are an invaluable tool when learning any computer language (van Gog et al., 2011), SQL is no exception.

SiS includes the ability to provide contextual examples for all clauses in relation to the database currently in use. These examples are generated automatically by analysing the structure and contents of the chosen database. SiS will analyse the database and select the attributes and values that best demonstrate a particular clause. For instance, when demonstrating the GROUP BY clause, the system will attempt to identify a foreign key containing duplicate values in numerous tuples, a grouping and counting function will be applied to demonstrate how many of each foreign key there are. The content and structure of the database is used to provide a description of the query's meaning along with the query itself and the results it yields.

As with many of the functions within SiS the help functionality can be customised by course administrators; once the examples are generated they are stored in an XML file, this file can be edited to include more appropriate examples if necessary. Results of the sample queries are not stored but obtained by executing the query on the database as required.

Appendix C shows the automatically generated examples for a simplified version of the movie database in Appendix A.

6.5.4.6 Catering for a wide variety of users

Section 2.4.1.2 defines expert users as a wide range of users who need to be able to build textual queries. The broad range of users and the differing rates at which they learn SQL means that educational software such as SiS must cater for a large spectrum of users from the new student to those on the cusp of becoming experts. SiS attempts to appeal to all such users by breaking each clause down into their own tabs allowing less confident users to avoid more advanced features while other users can explore further at their own pace. The customisation of SiS by teachers (Section 6.5.4.8) also helps them ensure the interface is appropriate for their students.

6.5.4.7 Abstract user interface

In Section 3.3.4.1 the potential problems relating to the use of abstract graphical representations of queries are highlighted; in summary they can reach a point at which they become so abstract they act as more of a hindrance than a help in the process of learning SQL. The user interface for SiS is intentionally not overly abstract and closely follows the structure of an SQL clause. This allows the user to easily map the functions of the user interface to the different components of an SQL statement. For instance, to include attributes in the SELECT clause involves selecting attributes using a check-box, selecting relations to be used in the query is done by clicking on a graphical representation of the database structure; both actions are easily mapped to their effect on the query.

6.5.4.8 Customisation

As previously discussed, there are a wide range of needs for expert users, some have much more complex requirements than others. The ability to customise an educational application to meet the needs of these various students is important; SiS includes a number of customisable features. The generic implementation of SiS that allows it to be used in conjunction with any SQLite database allow teachers or course administrators to pick a database that suits the needs of the students. In addition to this SiS has a number of elements that can be enabled/disabled using a configuration file (Appendix C), these include:

- Individual join types (inner, left, right, outer)
- Clauses (all except SELECT and FROM can be disabled)
- Sub queries
- The ability to save queries

- Explicit/implicit joins

6.5.5 Classification of SiS

As discussed in Section 3.3.3 software designed to aid students in learning query languages can, largely, be categorised as *analytical* or *animated*. SiS can loosely be categorised as an animated system. Nevertheless, it differs from those previously discussed in that it guides the user towards incrementally building a query before submitting it to view the results. This guidance through the process of building a query provides the user with the steps taken to achieve the final result set; this differs from other animated systems in that the steps for animation are prompted entirely from the user and not through the use of predefined stages. The student can choose to build their SQL SELECT statement in almost any order; for instance, they could build the WHERE clause (and view the effects this has on the results) then add some ordering information before changing back to the WHERE clause to refine it further.

Animations used in educational software have the potential to become a distraction rather than a benefit to the user Justice (2000); Rieber (1988). To avoid this problem occurring in SiS the animations are subtle and non-intrusive; they can be viewed as transitions from one query to another rather than elaborate animations designed to demonstrate a concept.

6.6 System architecture

SiS is a web based system that is built using HTML and various JavaScript libraries (Section 6.6.0.1); SQLite was chosen for the database upon which SiS operates as databases are self-contained within a single file allowing many databases to be quickly added to the system.

Figure 6.5 shows how the various components of SiS connect to allow for live updates to the SQL translation and results. The HTML elements of the user

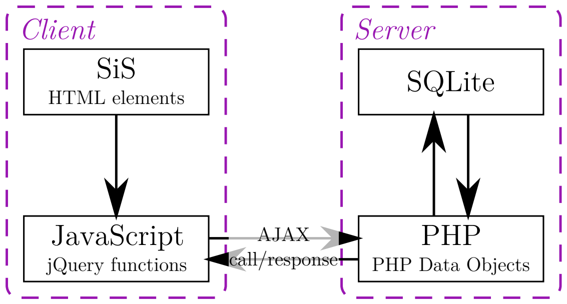


FIGURE 6.5: The SiS prototype's system architecture

interface are processed on the client machine using JavaScript functions to produce an SQL SELECT statement, this statement is then sent to the server, via an AJAX call, where a PHP script uses PDO to execute the query on the appropriate SQLite database. Upon receiving the results from the database the script compiles the results into an HTML table before passing them back to the client to be displayed along with the SQL executed.

6.6.0.1 Libraries used

The implementation of SiS is made possible through the use of a number of freely available JavaScript libraries, the following were used to varying extents:

jQuery (https://jquery.com/) Handling AJAX and various elements of dynamic page generation.

jQuery user interface (http://jqueryui.com/) Allows for the movement and resizing required for the Sandpit components.

vis.js (http://visjs.org/) Used to draw the database visualisation.

fancybox (http://fancybox.net/) Popup windows used throughout the system (e.g. help windows).

jQuery Splitter (https://github.com/jcubic/jquery.splitter) To form the different panels and allow them to be resized by dragging the dividers.

jQuery.textcomplete (http://yuku-t.com/jquery-textcomplete/) Auto-complete attribute names in the Sandpit.

attrchange (http://meetselva.github.io/attrchange/) Listen for changes to the database visualisation size to ensure it is redrawn when necessary.

6.7 Limitations of SiS

Despite the potential benefits of SiS, it is not without limitations. Some of these limitations stem the implementation choices but many were conscious design decisions, sometimes with unintended consequences.

6.7.1 SQLite limitations

SiS is built using SQLite because of the benefits that self contained databases can bring (see Section 6.6). As a result, there are a number of limitations that SiS inherits from SQLite, the most notable of which is the limited support for join operations.

SQLite supports the INNER and LEFT join but not RIGHT or FULL OUTER joins (SQLite, 2015c). This limitation of SQLite represents a potentially significant drawback in its suitability for educational use. Section 3.3.1.2 discusses the importance of gaining a good understanding of join operations and without support for RIGHT and FULL OUTER joins a students understanding has the potential to be somewhat limited. To address this issue a solution is integrated into SiS that enables it to simulate both RIGHT and FULL OUTER joins without the student needing to understand the conversion performed. This simulation of SiS is the only scenario in which the SQL shown to the user is not the same as that which is executed on the SQLite database.

The following query is used to find all the people named *"Smith"* and the characters they have played regardless of whether they have had any acting roles; if the person found has had no acting roles then null values will be shown in the *acts_in* relation.

```
SELECT *
FROM acts_in RIGHT JOIN people
        ON people.id = acts_in.person_id
WHERE people.name LIKE ' %Smith';
```

The query can simply be re-written using a left join by swapping the order of the relations and using LEFT JOIN instead of RIGHT JOIN. If the above query were to be built in SiS it would be displayed to the user as a RIGHT JOIN but executed using the LEFT JOIN notation:

```
SELECT *
FROM people LEFT JOIN acts_in
        ON people.id = acts_in.person_id
WHERE people.name LIKE ' %Smith';
```

Full outer joins can also be simulated using a union of a left and right (simulated) join, for example:

```
SELECT *
FROM a OUTER JOIN b ON a.id = b.id;
```

can be re-written as:

```
SELECT *
FROM a LEFT JOIN b ON a.id = b.id
        UNION
        b LEFT JOIN a ON a.id = b.id;
```

As with the right joins the use of FULL OUTER JOIN is displayed to the user as a full join but executed on the database as the union of two left joins. This simulation of different join styles allows students to develop a good understanding of the various types of join, something widely found to be a difficult skill to master, while still using SQLite and the limited join types it supports.

6.7.2 Subqueries

In SiS each query is represented by the SiS window and all the elements within it, this style of visualisation means that representing subqueries would be very difficult without using a potentially complex series of interlinked windows. In many relational database implementations the number of subqueries allowed is, essentially[1], unlimited meaning any user interface designed to represent this must

[1]Although, in practical terms, there is no limit on the depth of subqueries there is often a limit, albeit a large one, enforced by the database implementation. SQLite has a limit of 1000 levels (SQLite, 2015b), OracleDB imposes no limit in the FROM clause but a limit of 255 levels in

be capable of scaling to an infinite size whilst remaining easy for a new student to understand.

The use of subqueries in SiS is limited to the FROM clause and these can only be used by referring to an SQL file containing the subquery (i.e. the subquery must be build graphically and then saved to a file that can be imported into the new query). The more common use for subqueries, in the WHERE clause, is not possible in SiS meaning the following query could not be represented in SiS:

```
SELECT name
FROM people
WHERE id IN(
        SELECT person_id
        FROM acts_in
        WHERE character_played = 'Morpheus');
```

6.7.3 Unusual joins

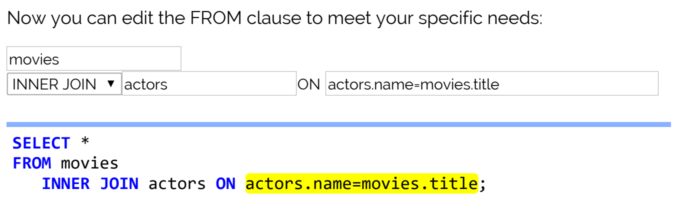
Join operations in SiS are primarily built using the graphical representation of the database on the right of the window (Section 6.5.3.2). This graph is automatically generated from the schema of the database and allows for the quick building of joins involving primary-foreign key relationships. This construction of joins ensures that the user is aware of the join condition used as they are required to click this to build the join.

In some unusual circumstances users might be required to form joins that are not built around a primary-foreign key relationship. For example, the following query can be used to find people with a name that is also a movie title (this could also be achieved using a number of alternative techniques):

```
SELECT *
FROM movie INNER JOIN people ON people.name=movie.title;
```

If a student is required to build such a join operation they must first start with a standard primary-foreign key join and customise it in the FROM clause tab to meet their specific needs (Figure 6.6). Although this functionality might not be

the WHERE clause (Oracle, 2015a) and Microsoft SQL Server has a limit of 32 levels (Microsoft, 2014).

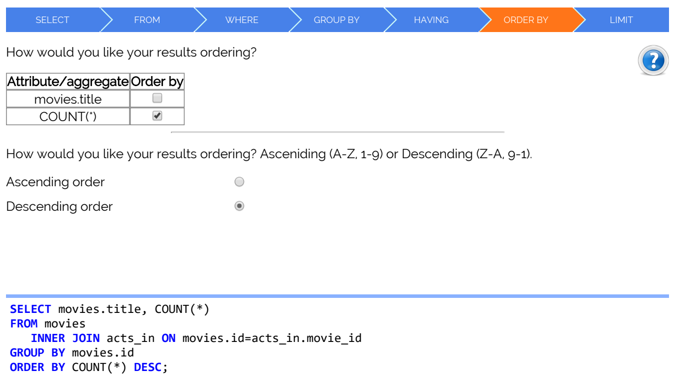


Now you can edit the FROM clause to meet your specific needs:

movies			
INNER JOIN ▾	actors	ON	actors.name=movies.title

```
SELECT *
FROM movies
   INNER JOIN actors ON actors.name=movies.title;
```

FIGURE 6.6: Building unusual joins with SiS



How would you like your results ordering?

Attribute/aggregate	Order by
movies.title	☐
COUNT(*)	✓

How would you like your results ordering? Asceniding (A-Z, 1-9) or Descending (Z-A, 9-1).

Ascending order [✓ Descending order]

```
SELECT movies.title, COUNT(*)
FROM movies
   INNER JOIN acts_in ON movies.id=acts_in.movie_id
GROUP BY movies.id
ORDER BY COUNT(*) DESC;
```

FIGURE 6.7: The ORDER BY clause in SiS

immediately obvious to new users it is something that is unlikely to be often used, particularly when learning SQL.

6.7.4 Advanced ordering and grouping

SQL allows for both the ordering and grouping of the results according to multiple attributes therefore allowing for the results of a query to be finely tuned according to the users specific needs. SiS allows results to be ordered and grouped according to multiple attributes but these cannot be rearranged and the type of ordering (ascending or descending) cannot be applied to each attribute in turn. In addition the ORDER BY clause is restricted to those attributes found in the SELECT clause, attributes must appear in the results to form the ordering criteria. This limitation is imposed in an attempt to reduce the confusion that might occur by ordering results by an attribute that is not visible.

6.8 Chapter Summary

In this chapter the concept of a third category of user, the student, who is transitioning from a novice to an expert user is introduced. In response to the needs of the student a number of requirements for educational SQL software are outlined along with the introduction of SQL in Steps (SiS) a web based application designed specifically for education. SiS allows students to graphically build queries whilst remaining aware of the textual translation of a query and the results it yields. The low level of abstraction allows students to easily transition from SiS to textual SQL as their confidence increases. Table 6.1 summarises the contributions of SiS in comparison to other work within the field. An evaluation of SiS involving first year undergraduate students from the databases course is detailed in Chapter 7.

System name	**Reference**	**Animated**	**Analytical**	**Supplementary/replacement**	**Learning**	**Assessment**	**Focus on SELECT**	**Pool of queries**	**Building or demo**	**Database visualisation**	**Boolean help**	**Error help**
SQL in Steps	Garner and Mariani (2015a,b)	✓	×	S	✓	×	✓	×	B	✓[2]	✓[3]	×[4]
SQLator	Sadiq et al. (2004)	×	✓	S	×	✓	✓	✓[5]	D	✓[6]	-	-
SQL-Tutor	Mitrovic (1998b)	×	✓	S	✓	×	✓	✓	D	✓[7]	-	✓[8]
-	Russell and Cumming (2004)	×	✓	R	✓	✓	✓	✓	D	-	-	-
LEARN-SQL	Abelló et al. (2008)	×	✓	S	✓	✓	×	✓	D	-	-	-
eSQL	Kearns et al. (1996)	✓	×	-	✓	×	✓	×	D	✓[7]	-	-
QueryViz	Danaparamita and Gatterbauer (2011)	✓	×	S	-[9]	-[9]	✓	×	D	-	-	-
SAVI	Cembalo et al. (2011)	✓	×	S	✓	×	✓	×	D	✓[10]	-	×
ADbC	Murray and Guimaraes (2008)	✓	×	S	✓	×	×	✓	B	✓[11]	-	-
WinRDBI	Dietrich et al. (1997)	×	✓	S	×	✓	×	×	D	✓[7]	-	-
AsseSQL	Prior (2003)	×	✓	S	×	✓	✓	✓	D	-	-	-
SQLify	De Raadt et al. (2006)	✓	✓	R	✓	✓	✓	✓	D	✓[7]	-	-

TABLE 6.1: A comparison of SiS against other published work. A single dash denotes a feature that is not described.

[2]Graphical representation (Section 6.5.4.3)
[3]See Section 6.5.4.4
[4]See Section 7.2.7.4
[5]SQLator used 300 queries, sorted by difficulty
[6]Entity relationship diagrams and snapshots of the table contents used
[7]Textual description
[8]*"Hints"* give more information about the type of error
[9]QueryViz is targeted at practical use rather than at education
[10]Displays the relation contents as the query is animated
[11]ER diagrams are used

Chapter 7

Evaluations

This thesis presents two different software prototypes that both stem from the same problem: how to ease access to relational databases. The solution to this problem differs greatly depending upon the type of user the solution is built for. In Chapter 2 two classes of user were identified and the resultant applications were built to reflect their differing needs. CAFTAN was built to meet the needs of novice users with no database understanding and SiS was developed to help students (Section 6.1) to become expert users. The following chapter presents evaluations of both systems and is structured as follows:

- Section 7.1 details the evaluation of Context Aware Free Text ANalysis (CAFTAN) which involves comparing query interpretations made by participants with those made by CAFTAN along with performance tests in relation to both indexing and query processing.

- Section 7.2 presents the evaluation of SQL in Steps which involved the deployment of the system into a live undergraduate databases course.

- Section 7.3 summarises these evaluations.

7.1 Keyword search

The Context Aware Free Text ANalysis (CAFTAN) system was introduced in Chapter 5, it aims to interpret keyword queries and represent their results in a generic way that offers improvements over existing systems.

7.1.1 Methodology

The evaluation of CAFTAN is twofold, it involves quantifying the quality of interpretations made by CAFTAN and also its performance. To quantify the quality of CAFTAN we compared interpretations of keyword queries made by humans against those made by CAFTAN and other systems; this methodology avoids the need to make assumptions regarding how queries are interpreted by users (Section 3.2.4) and enables the evaluation of the system without the need for participants to use it (Section 7.1.2). Participants were presented with a series of keyword queries and three possible interpretations of their meaning; they were asked to rank the interpretations from most to least likely. Once multiple users had ranked the different interpretations averages could be calculated to find the most common interpretation of a particular set of keywords.

These averages can be used to compare various different search systems against the interpretations provided by the participants. If we assume that the participants choices represent the ideal standard for interpreting keyword queries we can measure how close to this ideal standard various systems come. As discussed in Section 5.4.1.3, CAFTAN utilises its index to produce an SQL statement which is executed directly on the database; as a result, if the interpretation of the query is accurate the precision and recall values will always be at their maximum. Testing the precision and recall of an accurately formed SQL SELECT statement simply tests the quality of the database implementation, not the software generating those statements, as a result this evaluation ranks the quality of the interpretations of queries and operates under the assumption that accurate interpretations lead to accurate results. This part of the evaluation is designed under the assumption that keyword search systems operate best when they interpret queries in the same way as the humans using the system.

The performance of CAFTAN was evaluated by repeatedly indexing and querying a database in order to find average times for each operation. The index for the movie database was constructed, from scratch, 200 times while 100 queries were executed 100 times each in both the generic and customised versions of CAFTAN, a total of 20000 query executions.

7.1.1.1 Database

Keyword search aims to make relational databases accessible to users who have little or no understanding of database structures or how to extract information from them. In order to make the evaluation accessible to a wide range of participants the database of movies referred to throughout this thesis was chosen, this is something the majority of users can relate to without the need to familiarise themselves with the database contents (Appendix A). Participants were not aware of the database structure or its exact contents.

7.1.1.2 Participants

The participants were recruited through links shared on social media and the questionnaire was available for seven weeks. Although the participants were completely anonymous it can be assumed that they represent a wide variety of different ages and computer literacy. Throughout the duration of the study the survey was completed by 96 different users, each user was asked to complete up to ten queries and on average they completed 9.05 questions each, as a result each set of keyword terms was interpreted on average 8.7 times. Thirty-seven participants registered and agreed to the conditions of the study but failed to answer any questions.

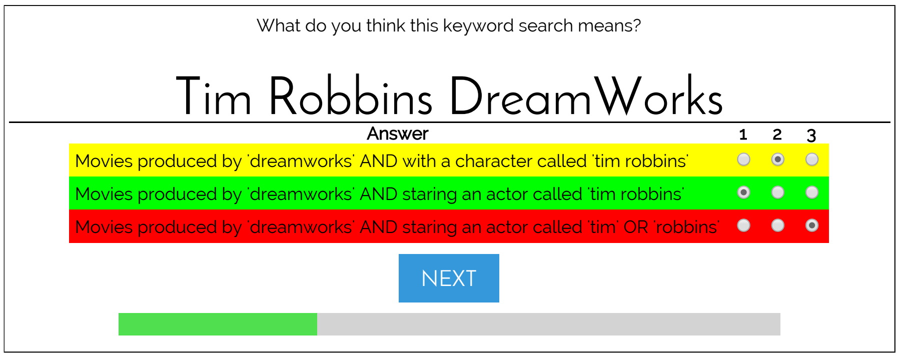
7.1.1.3 Question generation

Questions were manually generated by joining various different tables within the movie database, from these joined tuples a selection of keywords were used to generate query terms. The average length of queries was 2.64 terms with six terms representing the longest query and one the shortest. A pool of 100 keyword queries was generated and, because each set of terms was generated from joined tuples, they all had at least one valid answer in our database. Appendix B shows the questions generated for this evaluation.

Each set of terms were given three possible interpretations, these interpretations were manually generated by assessing their possible different meanings.

7.1.1.4 Questionnaire structure

Upon accessing the questionnaire participants were first given a description of the research along with contact details should they have any questions, comments or criticisms. After this the participants were shown a consent form that they must agree to before proceeding to the questionnaire (both research description and consent form can be found in Appendix B). Finally, users were presented with instructions for completing the questionnaire including screenshots demonstrating how to answer each question.


What do you think this keyword search means?

Tim Robbins DreamWorks

Answer	1	2	3
Movies produced by 'dreamworks' AND with a character called 'tim robbins'	○	●	○
Movies produced by 'dreamworks' AND staring an actor called 'tim robbins'	●	○	○
Movies produced by 'dreamworks' AND staring an actor called 'tim' OR 'robbins'	○	○	●

FIGURE 7.1: An example of a question used for the evaluation of SiS

Once users proceed past the instructions they were presented with their first question, Figure 7.1 shows an example query. The keyword terms are shown in large font at the top and participants can order the queries according to how they interpret the given query by using the radio buttons to the right of the interpretations. The radio buttons are used to rank the interpretations and they restrict the choices such that each interpretation can only hold one ranking position. As the participant ranked the interpretations they were colour coded to reassure them that 1 represents the most likely and 3 the least. Participants were unable to advance to the next question until they ranked the existing interpretations. Below the “Next” button a progress bar shows users how far through the questionnaire they are (Figure 7.1).

Questions are chosen from the pool of 100 possible queries at random and the possible interpretations are also ordered at random to ensure the order in which they are presented has minimal effect on the order participants give. No two participants answered the same series of questions.

7.1.1.5 Performance tests

The performance tests for CAFTAN are divided into two main categories:

- Index building
- Query response times

To test the speed at which a database index can be constructed for use with CAFTAN the movie database was repeatedly indexed 200 times. After the index was built it was completely removed only to be re-built in its entirety again.

To test the query response times a test mode integrated into the CAFTAN application enabled the automatic execution of queries, as a result, we can analyse the performance of a large number of query executions. The test mode does not remove any graphical elements of the query system, it automatically enters the query terms into the text field and submits the query, waiting for the results window to be shown before repeating the process for the next query. The 100 queries used to evaluate the accuracy of CAFTAN were also used in the performance tests. Each query was executed 100 times on CAFTAN implementations with and without customisations resulting in a total of 20000 query executions than can be used to analyse the performance of CAFTAN.

All performance tests were executed on the same computer, a mid-range laptop (Samsung NP-RF511) with the following specifications:

- Hardware:
  - Intel Core i5 2450M @ 2.50GHz
  - 8.00GB Dual-Channel DDR3 @ 665MHz
  - 1TB Seagate ST1000LM024 HN-M101MBB (SATA)
- Software:
  - Windows 10 Home 64-bit
  - Java SE Runtime Environment (build 1.8.0_51-b16)
  - MySQL v14.14
  - JDBC for SQLite version 3.7.2
  - JDBC for MySQL version 5.1.24

The results are presented in Section 7.1.5.

7.1.2 Transferable methodology

The aim of the evaluation of CAFTAN was primarily, although not exclusively, to ascertain the quality of the query interpretations it generates. It was also designed to:

- Provide an insight into users' searching habits.
- Allow for comparisons between CAFTAN and other systems.
- Produce a reusable data set that could be used with future iterations of CAFTAN or with other applications.

- Provide a comprehensive evaluation of CAFTAN without users needing to interact with the system.

The evaluation involved capturing the way in which users interpret keyword queries, this provides an insight into how they process the meaning of keyword queries. This can be used to ensure that keyword search applications respond in a manner appropriate to the users expectations (Section 7.1.2).

The users collectively produced a *"gold standard"* of query interpretation that all keyword query systems should aim for. If a system is capable of consistently interpreting queries in the same way as humans then it would always respond in a predictable and effective manner. This *"gold standard"* data set also allows for a detailed and accurate evaluation of CAFTAN without the need for participants to use the system; this allows the results to be used in conjunction with future iterations of CAFTAN without re-running the evaluation. The data generated by users for the evaluation of CAFTAN can be applied to any keyword search application and, as such the complete dataset is freely available (see Section B.3 for details).

7.1.3 User interpretations

The evaluation involved collecting 869 interpretations of 100 different queries, this provides an insight into how keyword queries are handled by different users.

Perhaps the most interesting observation taken from the results of evaluation is the lack of use of Boolean OR operations. A third (99/300) of the possible interpretations of queries contained at least one OR operation, however, none of the highest ranked interpretations contained an OR operation. Of the 896 interpretations only 62 (7.1%) were thought to contain an OR operation. Although these queries are not interpreted as requiring an OR operation some of their SQL translation may require an OR operation, examples of such a situation may include a many-many query used to find the set of films falling into two genre categories (e.g. Q_{20}*{war horror}*, see Section 3.2.5.5 for an explanation of many-many queries).

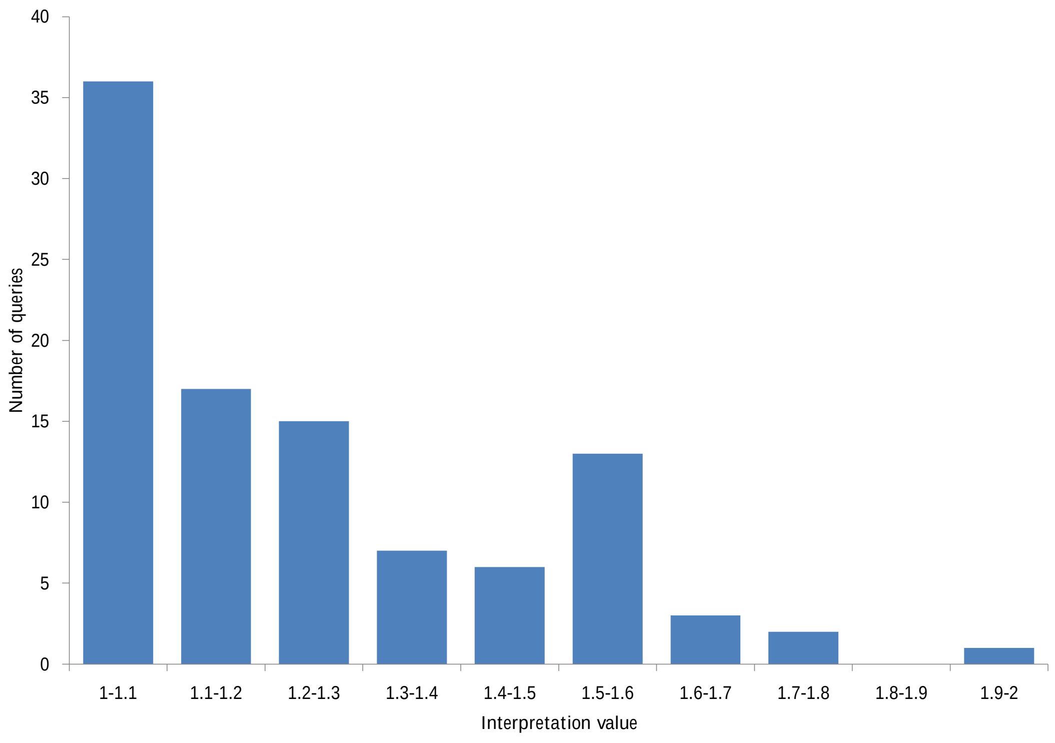


FIGURE 7.2: The frequency of the participants' favourite interpretations

This observation suggests that when users construct keyword queries they are often seeking answers that contain all the given keywords and that AND operations should be favoured over OR. This is the case in CAFTAN which will apply an AND where possible and if impossible use an OR operation (Section 5.4.1.3).

Figure 7.2 and Table 7.1 show the variation in interpretations of the 100 queries used in the evaluation; the interpretation value refers to the average of the user ranks (1 = most likely, 3 = least likely), therefore a value of 1 indicates a unanimous decision. Thirty-six of the one hundred queries had a unanimous decision as to the most likely interpretation and only one had no favourite with each interpretation receiving an average ranking of 2. This graph shows that interpretations were generally close to a unanimous decision with 75% of the queries having an average highest ranking of less than 1.4.

Average user interpretation	Frequency
1.0 - 1.1	36
1.1 - 1.2	17
1.2 - 1.3	15
1.3 - 1.4	7
1.4 - 1.5	6
1.5 - 1.6	13
1.6 - 1.7	3
1.7 - 1.8	2
1.8 - 1.9	0
1.9 - 2.0	1

Table 7.1: The frequency of the participants' favourite interpretations

7.1.4 Comparisons

The aim of any keyword search system is to interpret queries in a predictable manner such that users are shown the results appropriate to their interpretations of the entered terms. In order to quantify the quality of CAFTAN the interpretations made were compared to those made by IMDb and TMDb[1]. Almost all search terms can be interpreted in various different ways and all of the different systems ranked the interpretations according to different criteria. As discussed in Section 5.4.1.3 CAFTAN ranks the different interpretations according to how likely they are to be the best interpretation; both IMDb and TMDb apply search terms to a single field and rank these fields according to the number of results they each contain. Figure 7.3 and Table 7.2 show how the highest ranked (participant) interpretations were handled by various different systems. In an ideal scenario a search application would rank the participants best match as its first choice. The graph shows that the generic implementation of CAFTAN interpreted queries in the same way as the participants 64% of the time and adding customised strengths and user interface (Section 5.4.1.5) increased this to 81%.

Despite the fact that systems such as IMDb and TMDb are designed specifically to handle data relating to movies they only matched the same interpretation as the participants 24% and 26% of the time respectively. Their second choice rarely

[1]These comparisons were made in April 2015 and reflect the capabilities of IMDb and TMDb at that time.

matched the most likely interpretation according to the participants, as a result, they frequently failed to return any appropriate results. IMDb and TMDb failed to interpret the queries over 70% of the time while the customised implementation of CAFTAN only failed 16% of the time.

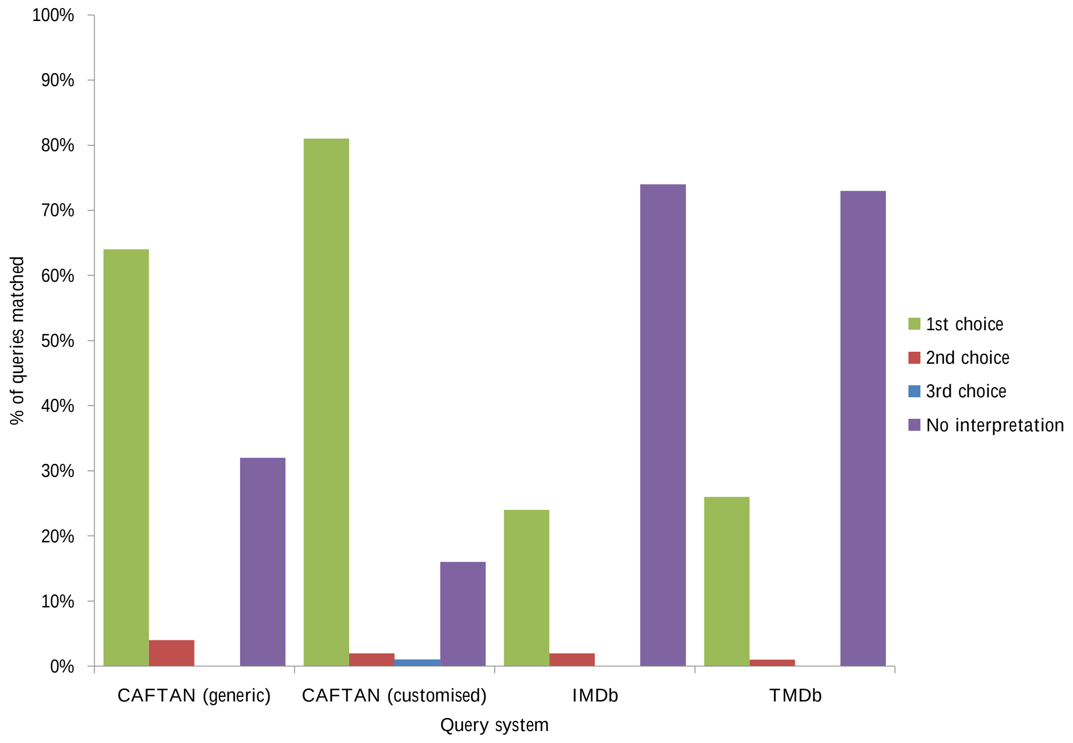


FIGURE 7.3: How participants' best match was handled by various different systems

	CAFTAN (generic)	**CAFTAN (customised)**	**IMDb**	**TMDb**
1st choice	64	81	24	26
2nd choice	4	2	2	1
3rd choice	0	1	0	0
No interpretation	32	16	74	73

TABLE 7.2: How participants' best match was handled by various different systems

The number of terms in a keyword query appear to have an impact on the success rate of bespoke systems such as IMDb and TMDb. Figure 7.4 and Table 7.3 show how different query lengths were handled; neither IMDb nor TMDb correctly interpreted any queries containing three or more terms. The most common number of search terms within the 100 queries used in the evaluation was 3 (38/100 queries contained 3 terms), the generic implementation of CAFTAN correctly resolved 63% of these queries while the addition of strengths increased this to 82%. The benefits of the inclusion of strengths are clear when observing the accuracy of CAFTAN when interpreting queries containing 4 terms; the generic system correctly interpreted 50% of the 16 queries while the customised implementation correctly interpreted 94%, only failing on one account.

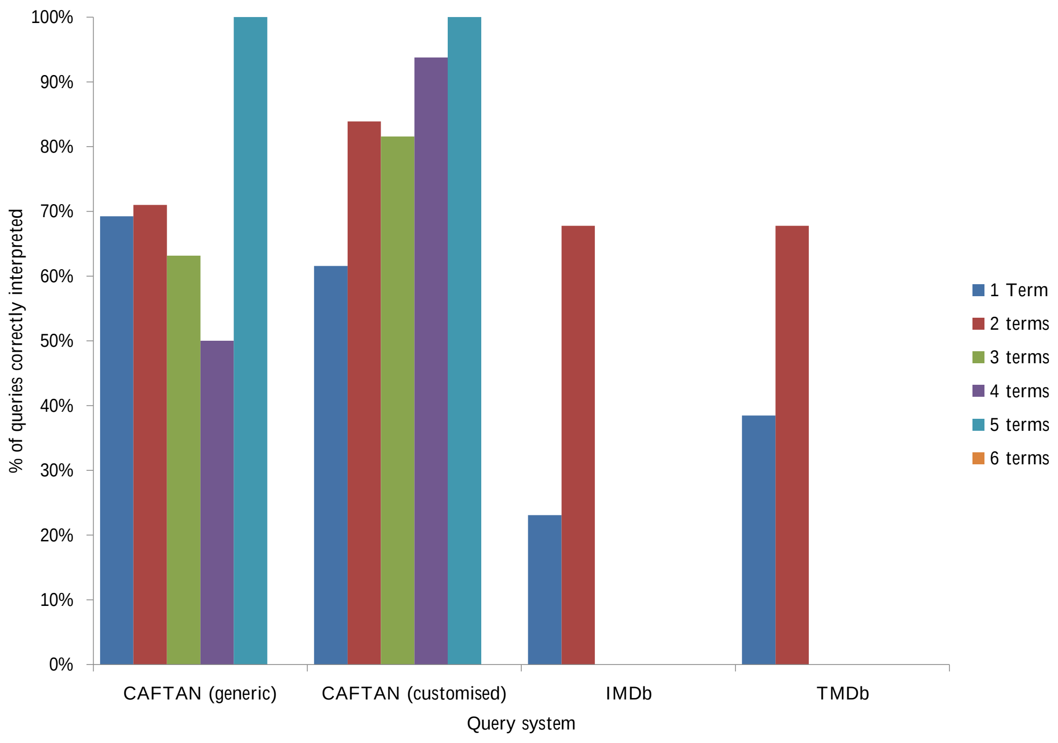


FIGURE 7.4: How different systems handle different number of search terms

No. of terms	Total	CAFTAN (generic)	CAFTAN (customised)	IMDb	TMDb
1	13	9	8	3	5
2	31	22	26	21	21
3	38	24	31	0	0
4	16	8	15	0	0
5	1	1	1	0	0
6	1	0	0	0	0

TABLE 7.3: How different systems handle different number of search terms

7.1.4.1 vs CAFTAN (generic)

The generic implementation of CAFTAN involves executing the queries using no customisation other than that which is programmatically generated by analysing the database structure and contents. In this scenario the system has no awareness of the relative importance of different attributes so all weights are generated automatically. Despite this CAFTAN interpreted 64 of the 100 queries in the same way as the participants and was more accurate than the customised version when interpreting single term queries.

In the following sections we look at how the participants' interpretations compare with CAFTAN (both generic and customised), IMDb and TMDb.

7.1.4.2 vs CAFTAN (customised)

Unlike the generic implementation of CAFTAN, the customised version allows for some customisation by defining the relative importance of the attributes and the information that should be displayed in certain circumstances (Section 5.4.1.5). The strengths applied to the database were manually assigned as follows:

- All IDs: 0.2 (searches based upon the ID are extremely unlikely)
- Movie tagline: 0.5 (taglines contain many terms that can skew results but rarely constitute search terms)
- Release date, genre name, people's names: 1.2 (these attributes are frequently used as search terms)
- Movie title: 1.5 (titles are common search terms)

The strengths were not generated in response to the results of the generic queries but were generated by increasing the weight of those attributes believed to be commonly used in queries relating to movie data. When using CAFTAN without strengths some of the query terms were incorrectly interpreted because the system treated all the attributes equally, in practice this is rarely the case. A good example of this is the *tagline* attribute in the movie table, this attribute contains a large number of terms that can be easily confused with a genre: *"adventure"*, *"war"*, *"romance"* etc. Despite this, it seems unlikely that a user would search for a movie based upon it's tagline rather than its genre, to customise CAFTAN for the movie database the weight of the tagline was reduced and the weight of a genre name increased.

The improvements described here increased the rate at which CAFTAN interprets queries in the same way as the participants from 64% to 81%.

7.1.4.3 vs IMDb

The Internet Movie Database (IMDb) is a popular website containing a wide variety of movie data and is consistently ranked in the top fifty websites in the world (Alexa, 2015a). As a result the expectation might be that such a popular website would utilise systems that result in consistently accurate interpretations of keyword queries. This evaluation shows that IMDb is only capable of handling relatively simple queries which contain data from one domain (e.g. actors *or* movies, not both). This limitation was also identified by Luo et al. (2007) who found the searching capabilities of IMDb were limited to *"pre-built template queries"* and if the user strays beyond these its capabilities are significantly diminished; the query Q_{21}*{2001 hanks}* was used to illustrate how IMDb fails to provide appropriate results for queries beyond the scope of these templates.

All of the searches made using IMDb were done so using the keyword search available from any page on IMDb, the site offers advanced search options but these cannot be considered keyword search as the user is required to specify the field in which the terms appear.

IMDb employs a simple ranking technique for the different interpretations of the keywords provided. This system appears to sort the possible interpretations according to the number of results they yield with the most results earning the highest ranking, this simplistic approach seems to be sufficient for the majority of cases.

When presented with the same 100 keyword queries as our participants IMDb interpreted them in the same way only 24 times; in addition the participant's top interpretation was twice chosen as IMDb's second choice. The primary reason for this low success rate is the system's inability to apply different search terms to multiple fields, all search terms are applied to a single field which often leads to the search terms returning no results. An example of how this severely limits the capabilities of the search system is the query Q_{22}*{Thriller Gary Oldman}*, all of the nine participants who interpreted this believed it should show movies with the genre *"thriller"* staring the actor *"Gary Oldman"*, despite this seemingly obvious interpretation IMDb returns no results because all the keywords cannot be found within the same field. As a result of this restriction IMDb is incapable of interpreting the majority of the queries used in this evaluation.

7.1.4.4 vs TMDb

The Movie Database (TMDb) offers similar services to those offered by IMDb with the inclusion of an API that can be freely used to access the content of the database behind TMDb. The website is considerably less popular than IMDb and is ranked at approximately 20,000 in the Alexa rankings (Alexa, 2015b). Despite the difference in popularity, and possibly due to the apparent simplicity of TMDb's system, the two operate in a very similar manner with TMDb slightly out-performing IMDb and interpreting queries in the same way as humans 26% of the time.

In a similar fashion to IMDb, TMDb appears to rank the different interpretations of keyword queries according to the number of results each interpretation brings. Similarly the system also runs each keyword query in a fixed number of interpretations (people, movies, TV shows, collections, companies, keywords and lists), not allowing terms to be found in different fields. One of the reasons TMDb appeared to slightly out perform IMDb was, in fact, due to its simplistic approach to queries; when presented with a search term that is unusual or would yield few results IMDb will attempt to find a more appropriate search term without notifying the user. These automatic corrections were observed when searching for *"Domino"*, IMDb displayed a selection of actors with the nickname *"Domino"* but also displayed a selection of actors and actresses called *"Dominic"* or *"Dominique"* etc. IMDb adjusts the search criteria despite also returning, in the second category, five movies containing the word *"domino"* in the title, in the same situation TMDb does not alter the search criteria and displays the appropriate movies. TMDb never alters the search criteria and will display no results without correcting or suggesting corrections for search terms. This example shows that it can be detrimental to alter the users keyword terms, particularly without notifying the user; perhaps the best approach is for systems to adopt a *"Did you mean . . . "* system similar to those used by search engines such as Google. These systems use statistical machine learning to observe how users make corrections to search terms when they make a spelling mistake allowing them to intelligently make recommendations or corrections to certain criteria (Google, 2007). Such applications rely on large numbers of users to build an accurate model of users behaviour, perhaps a reason why TMDb hasn't adopted such techniques.

As described above TMDb will only match keywords against a static set of fields within the database, this set of fields does not include the name of characters so searching for a character name may not yield any appropriate results.

7.1.5 Performance

Irrespective of the accuracy with which a query can be interpreted the system must provide query responses within a reasonable time. The following performance tests of CAFTAN were carried out to ensure the system represents a viable option for real world applications. All performance tests were carried out on the same test machine (Section 7.1.1.5) using a movie database (Appendix A).

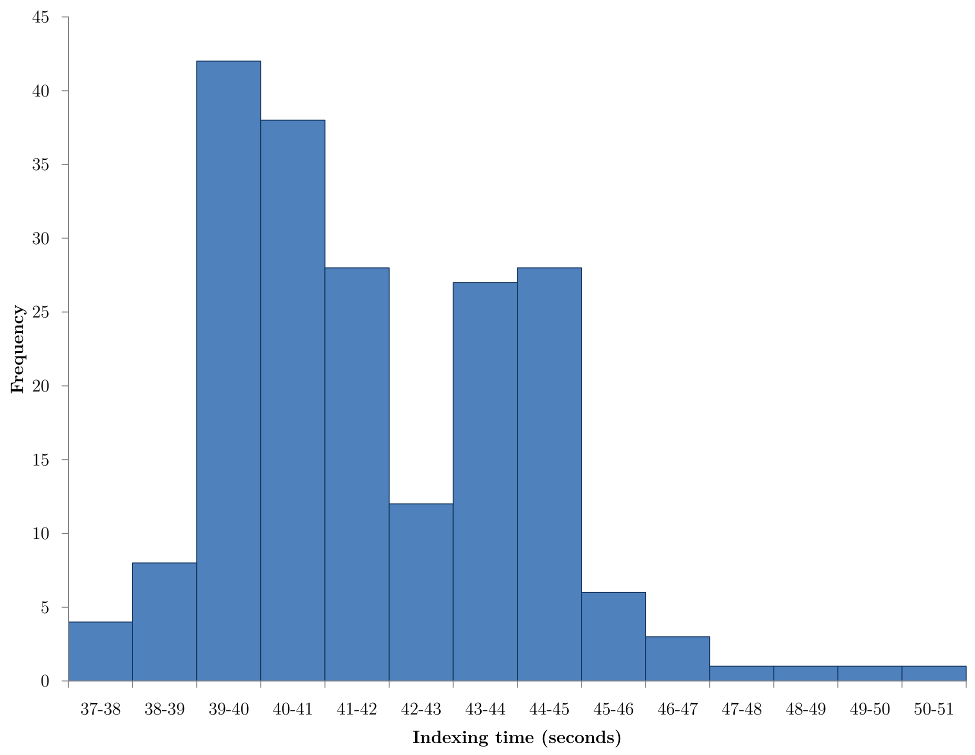


FIGURE 7.5: A frequency distribution graph for the time taken by CAFTAN to index the movie database

7.1.5.1 Index construction

The index was built 200 times on the test machine to reduce the impact of any anomalous results and to provide accurate average times for index construction. The mean time taken to build the index was 41.8 seconds ($\sigma = 2.3$ seconds), the movie database containing a total of 198668 tuples spread over eight relations meaning, on average, one tuple was indexed every 0.21 milliseconds. Figure 7.5 and Table 7.4 shows a frequency distribution for the 200 index constructions performed for this evaluation.

7.1.5.2 Query response times

A testing mode in CAFTAN was used to automate the repeated running of the 100 queries that were also used in the evaluation of the accuracy of CAFTAN. Each query was submitted to CAFTAN 100 times for both the generic and customised implementations of the system. Queries were executed one after another 100 times

Indexing time (seconds)	**Frequency**
37 - 38	4
38 - 39	8
39 - 40	42
40 - 41	38
41 - 42	28
42 - 43	12
43 - 44	27
44 - 45	28
45 - 46	6
46 - 47	3
47 - 48	1
48 - 49	1
49 - 50	1
50 - 51	1

TABLE 7.4: Indexing times for 200 repeated indexes of the movie database

rather than one query being executed 100 times before executing the next query 100 times; query caching[2] was disabled to ensure that disproportionately fast queries were not included in the evaluation in response to caching. Timings were made from the point of submitting the query until the results window, containing all appropriate information, was displayed; this includes query interpretation and execution times along with building and displaying the appropriate GUI components. Section B.1.3 shows both the queries executed along with their average execution times after 100 executions in both implementations of CAFTAN.

Figure 7.6 and Table 7.5 shows how the number of terms supplied impact the performance of CAFTAN. The graph shows that the performance of CAFTAN in both customised and generic form were largely unaffected by the number of terms supplied. The average execution time was less than half a second for all queries in both the generic and customised implementations of CAFTAN, irrespective of the number of terms.

The 100 test queries included three main types of query: dates, numeric ranges and textual queries. Table 7.6 shows the average execution time for each types of query; in both implementations of CAFTAN queries involving dates performed best with each implementation returning the associated results in less than 100ms

[2]http://dev.mysql.com/doc/refman/5.5/en/query-cache.html

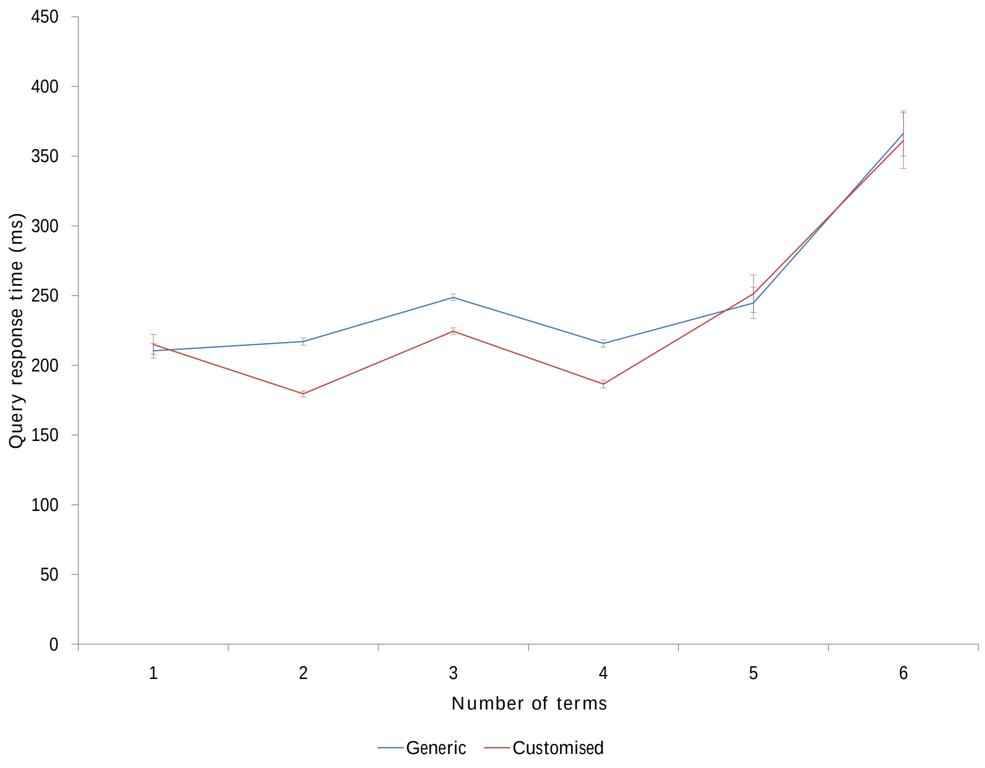


FIGURE 7.6: Number of query terms vs execution time

Terms	Query count	Mean		Standard deviation	
		Generic	Customised	Generic	Customised
1	13	210.5	215.0	190.2	259.2
2	31	216.9	179.5	143.8	120.7
3	38	248.7	224.5	144.4	143.4
4	16	215.8	186.6	107.7	111.4
5	1	244.8	251.4	113.2	136.0
6	1	366.3	361.1	161.3	200.6

TABLE 7.5: Response times for queries of various length

99.5% of the time for the generic implementation of CAFTAN and 97.5% of the time with the addition of customisations. Ranges performed the least well, most likely in response to the way in which they are calculated in CAFTAN and the large result sets (Section 5.4.5.1); despite giving the slowest response times the generic implementation always returned results in less than two seconds in both implementations of CAFTAN. All other queries within our sample data can be considered textual queries, these are the most common type of query and CAFTAN responded to them quickly, providing results in less than half a second 99.2% of the time in the generic implementation and 98.8% of the time in the customised version.

Query type	**Generic average time (ms)**	**Customised average time (ms)**
Dates	63	38
Ranges	451	429
Text	211	186
All	**230**	**205**

TABLE 7.6: Types of queries and their average execution times

7.1.6 Study limitations

The methodology used to evaluate CAFTAN differs from that which was used to evaluate much of the existing work; it is focussed on quantifying the quality of query interpretations with respect to accuracy and performance. This technique introduced a number of advantages such as the production of a reusable dataset that could be used for the evaluation of other applications or during the further development of CAFTAN and the in depth analysis of how users interpret keyword queries. Despite these benefits the evaluation was not without its drawbacks; participants never used the application and it was only shown operating with one database.

A major strength of the evaluation was the comparison of human interpretations of queries and those made by CAFTAN but the users never directly interacted with the CAFTAN system. This lack of interaction with CAFTAN means that users could not provide feedback regarding the quality of some features such as the results visualisation.

CAFTAN was developed as an entirely generic application that can be applied to any relational database and it was in no way tailored to meet the needs of the database used in the evaluation (Appendix A). However, the use of CAFTAN with one database only demonstrates its capabilities within the confines of the chosen database; elements of database design that are not present in the movie database are not tested in the evaluation of CAFTAN.

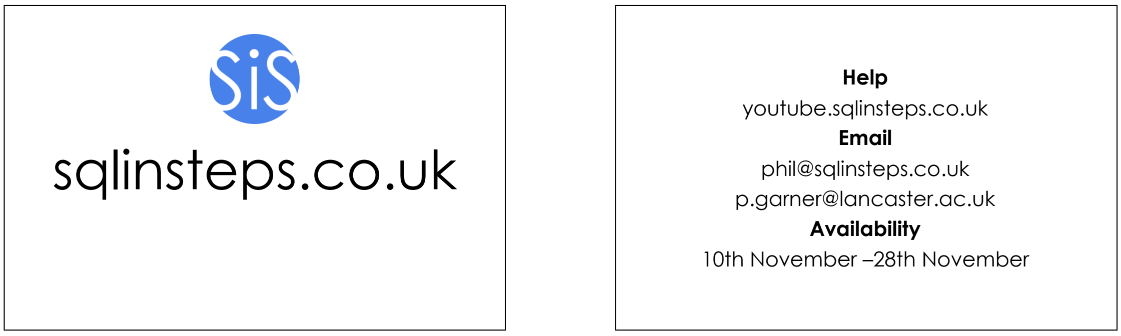
7.1.7 CAFTAN evaluation summary

This evaluation involved analysing the way in which keyword queries are interpreted by humans and comparing these to those generated by CAFTAN along with two bespoke systems (IMDb and TMDb) along with assessing the performance of the system. Key points of the evaluation include:

- Exploring the searching habits of novice users and finding out what they expect from keyword search, avoiding the need for developers to make such assumptions.
- Illustrating that CAFTAN is capable of interpreting keyword queries in a manner close to that of the human participants.
- Showing that CAFTAN outperforms bespoke systems, with regards to query interpretations, and adding customised strengths further improves its ability to search over a specific data set.
- The identification of weaknesses in some bespoke systems, particularly in relation to their inability to search over multiple fields or beyond predefined template queries.
- Performance test showed that CAFTAN is capable of both indexing and query resolution in within reasonable time; responding to the 20000 test executions in less than half a second 96% of the time.

7.2 SQL in Steps

In contrast to CAFTAN, which is designed for the novice user, SQL in Steps is designed to provide students with an understanding of SQL that will enable them to become expert users. The following evaluation describes the process of introducing SiS as part of an undergraduate databases course. The evaluation focusses on both the differing experiences of learning SQL with and without SiS and on some key areas highlighted by users of SiS.



SiS

sqlinsteps.co.uk

Help
youtube.sqlinsteps.co.uk
Email
phil@sqlinsteps.co.uk
p.garner@lancaster.ac.uk
Availability
10th November –28th November

(A) Front (B) Back

FIGURE 7.7: The SiS business card given to participants at the start of the study

7.2.1 Methodology

The evaluation of SiS involved 101 first year Computer Science undergraduates enrolled on a databases course. Upon agreeing to participate in the study they completed an assessment consisting of ten questions; the questions required that participants provide the SQL to answer questions regarding a given relational database (see Appendix C for the questions). On completion of the assessment students were randomly assigned to a group; one group would be granted access to SQL in Steps while the other would complete the study without access to SiS using only the usual course materials (lectures, work books and help from teaching assistants/lecturers). Users with access to SiS were given a business card (Figure 7.7) showing the URL of the tool along with information regarding help available, the dates for the study and email contacts.

Every week the students completed formative assessments, these assessments were used in conjunction with questionnaires to assess the improvements made by the students throughout the study. In addition, an assessment with an identical structure to the one completed prior to the study commencement was given to participants at the end of the study. This approach allowed for the measure of both the students understanding and their confidence in their understanding.

After the three weeks of using SiS a selection of participants were invited to attend a focus group to further discuss various aspects of SiS and how they could be improved in further iterations of the software. The focus group took place two weeks after access to SiS was withdrawn allowing for the discussion of the participants experiences during the remainder of the course without access to SiS (for details on where to find a full transcription of the focus group see Appendix C).

7.2.1.1 Participants

All first year undergraduates who were enrolled on the databases course (149 students in total) were given the opportunity to participate in the study, a brief description of the study was presented to the students during a lecture prior to the study commencing. On the databases course the students are split into four groups for practical assignments; in the first of these practicals the students were given the opportunity to participate in the study. Students were made fully aware that participating in the study did not guarantee them access to SQL in Steps and that they would be randomly assigned to a group: sixty-one had access to SiS, forty did not.

To ensure that the random assignment of students into groups resulted in an even distribution of ability all participants were required to complete an assessment prior to the study commencing. Participants in the SiS and non-SiS groups scored 6.6% and 6.1% respectively. These scores show that there was no significant difference between the capabilities of the groups prior to the commencement of the databases course.

7.2.1.2 Recording SiS use

Prior to accessing SiS students were required to log in using their university network credentials (a facility provided by the university, no personal information or passwords were stored by SiS), the reason for this was twofold; it prevented participants who have not got access to SiS from using the tool and it provides the ability to match use against specific participants.

Every time a query is executed, and hence every time a change is made to the UI, a record of this is made. These records show the popularity of the tool and also how individuals interact with it. All logs were compiled with participant assessments and anonymised allowing for comparison of use against performance but with no means to connect this information to actual participants.

7.2.1.3 Course structure

SQL in Steps is not intended to replace any components of an existing databases course but is designed to supplement it. As a result, with the exception of access to SiS, the course and resources made available were the same irrespective of group or participation in the study. These resources include one lecture per week and a workbook to be completed in a two hour practical session. In the practical sessions the students had the opportunity to use MySQL or SQLite with many opting to use both throughout the course.

The databases course spans five continuous weeks of a term and the practicals took the following structure:

Week 1 building and modifying databases using CREATE, INSERT and UPDATE statements

Week 2 Basic SELECT statements including SELECT, FROM (no joins), WHERE, ORDER BY, GROUP BY clauses

Week 3 Advanced SELECT statements including various different joins and set operations

Week 4-5 Assessed coursework (combining work from previous weeks)

The study spanned the first three weeks of the course ensuring that all students, irrespective of their involvement in the study, had access to identical resources during the assessed coursework.

7.2.1.4 Help with SiS

Throughout the study there were a number of different resources available to participants to help them in using SiS. The help available took the form of videos, written guides and both verbal and written communication.

Throughout the study, in the database lab sessions, there was always a researcher present to answer any questions or queries the participants had regarding SiS. All participants had access to an email address that could be used to contact the researchers outside of timetabled sessions.

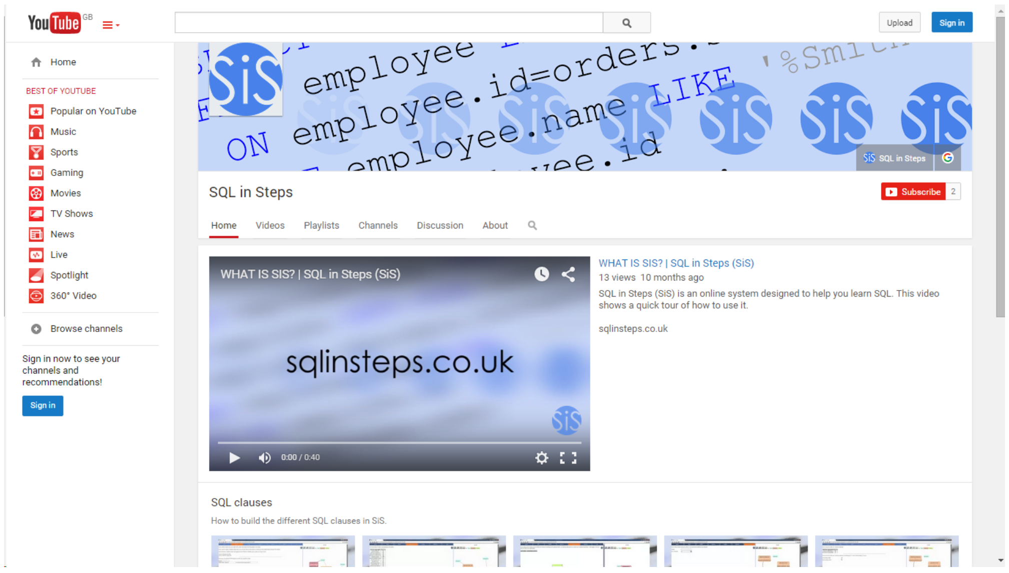
A five page guide to using SiS was produced and made available on the SiS homepage, the Moodle pages for the database course and in the weekly lab sessions (see Appendix C for the guide). This guide includes descriptions of all the elements of the user interface along with step-by-step instructions to building a simple query. Some frequently asked questions were also included in the document to answer some common questions regarding the software and study.

A series of eleven videos were made to demonstrate the various features of SiS, these allowed the user to watch a walk through showing the various features of SiS. One video was made for each clause along with separate videos showing various other features including a *"Getting started"* video showing how to build a simple query in SiS. The videos could be accessed through a YouTube channel (Figure 7.8), a link to the videos could be found on the SQL in Steps home page was shared with students at the beginning of the study.

Analysis regarding the way in which this help was used can be found in Section 7.2.7.

7.2.2 Usage

Of the sixty-one participants granted access to SiS twenty-seven of them (44%) executed at least one query totalling 5674 queries over the three week period, this represents an average of 210 queries per user and a total of 7 hours and 29 minutes spent using SiS. Of the twenty-seven participants nineteen students (31% of all



WHAT IS SIS? | SQL in Steps (SiS)
13 views 10 months ago
SQL in Steps (SiS) is an online system designed to help you learn SQL. This video shows a quick tour of how to use it.
sqlinsteps.co.uk

SQL clauses
How to build the different SQL clauses in SiS.

FIGURE 7.8: The YouTube channel homepage for SQL in Steps

participants with access) executed more than ten queries with an average of 297 queries and 23 minutes use per person. Seven students (11% of all SiS users) submitted more than one hundred queries with an average of 740 queries and 56 minutes use per person. The most active participant submitted 1871 queries and spent 2 hours 59 minutes using SiS over 19 sessions.

Although the relatively low number of students may initially be interpreted as them finding the tool of little help, it is unlikely that this is the main reason for low use of SiS. Of the 61 students with access 34 (55.7%) failed to use the tool at all and a further 3 (4.9%) logged in but did not interact with the tool in any way. The fact that over half of the participants with access to SiS never had first hand experience with it means they did not dismiss the tool in response to it's perceived quality. Other factors that may have contributed to a small number of active SiS users include:

- **Lack of a need for additional resources.** Many participants commented on the high quality of the resources available throughout the course: *"the workbooks are actually pretty good so there is no reason for you to drag away from [them]"*. Participants were able to complete the coursework exercises without the need for additional resources therefore they rarely encountered the desire to seek out additional help.

- **Lack of knowledge of the study.** Throughout the study participants were never directly encouraged to use SiS, as a result some simply forgot about it and others were unsure of when SiS was available to use.

The relatively low uptake of SiS means making conclusive quantitative claims about the benefits of SiS is difficult, however, there are some areas of learning SQL that the use of SiS appears to have impacted.

At the end of the study participants were asked to rate their confidence in various different aspects of SQL (48 (78.7%) SiS users and 34 (85%) non-SiS users responded, 82 (81.2%) responses in total); scores were given on a Likert scale (1 = very low confidence to 5 = very high confidence) (see Appendix C for the questionnaires). SiS users rated themselves as having a better understanding of both visualising the database (4.2 vs 3.75) and join operations (3.0 vs 2.79); this is likely a response to being shown the means to to visualise the database and graphical representations of joins. Although SiS appears to assist users in visualising and performing joins on the database it appears to slightly hinder the users' ability to interpret error messages. SiS users rated their understanding at 3.0 while non-SiS users rated their understanding at 3.53. SiS simply forwards the error messages from SQLite to the user without providing any additional information (Section 6.6) so irrespective of their group participants were exposed to the same error messages. The reason for the improved understanding of errors in non-SiS users may be as a result of increased exposure to them; SiS users are less likely to make syntactical errors, as they are lead through the building process, so are less likely to have seen error messages.

7.2.3 Student performance

Assessments given to students before and after the study were designed to show improvements made over the three week period. All students were expected to make significant improvements as many had no prior experience in the field (74% of participants scored 0% in the first assessment). In analysing the improvements made it was hoped that any additional progress made in the SiS group could be attributed to the availability of the tool throughout their learning process.

Section 7.2.8.2 discusses some limitations introduced by the willingness, or lack thereof, of participants to complete questionnaires and assessments required for the study. The low number (25%) of participants who completed the final assessment somewhat limits the conclusions that can be drawn from their results. Nevertheless greater improvements were made by those with access to SiS. The average improvement for a non-SiS user (from 15 completed assessments) was 50.6% whereas the improvement made by SiS users (from 5 completed assessments by active users of SiS[3]) was 66.9%.

The low rate at which the formative assessments were completed also limits the extent to which concrete conclusions can be made from them. Of the 101 participants 18 completed all three formative assessments; Table 7.7 shows the average results for these participants according to their assigned group. The second and third formative assessments are of particular interest as their focus was the SELECT statement (Section 7.2.1.3), problems for which SiS could have been of assistance. Some participants with access to SiS used it very little (less than 10 query executions) or never at all, as a result these participants can be considered as non-SiS users, these adjusted figures are shown in the last two columns of Table 7.7. These results show that active users of SiS increased their performance between the second and third assessments whereas students who did not use SiS achieved slightly lower results in the third assessment. This could be an indicator that SiS is particularly beneficial when students are presented with more complex problems, such as those found in the third and final formative assessment.

Anonymised data regarding student performance and their use of SiS are available in their entirety, see Section C.3.1 for details.

[3]5 participants with access to SiS but who showed little or no activity with the tool are excluded from this statistic

	Non-SiS	SiS	Actual non-SiS	Actual SiS
Number of participants	7	11	13	5
Initial assessment	7.5%	6.8%	4.0%	15%
Formative assessment 1	67%	83%	73%	87%
Formative assessment 2	90%	85%	89%	82%
Formative assessment 3	90%	85%	86%	90%

TABLE 7.7: Formative assessment results for the 18 participants who completed all assessments

7.2.4 Concerns with the command line

A Command Line Interface (CLI) such as the MySQL or SQLite client can be an intimidating environment for students who have previously only interacted with databases and other software through the use of graphical user interfaces. Participants described the command line as *"disorientating"* particularly *"when coming from Microsoft Access [or other GUIs]"*. Participants highlighted the fact that formatting of the results of a query often make them hard to interpret. Other factors contributing to the confusion introduced by a command driven environment include the ability to view both the query and results on screen at the same time; even common tasks such as repositioning the cursor and copying information to the clipboard are achieved in an unconventional manner. An aim of SiS was to ease the learning curve of the command line by using the same notation in a more familiar environment. Throughout the study some participants explored other GUIs such as phpMyAdmin to aid their learning, however, some described how they returned to SiS because they *"loved the visualisation"*.

The use of a CLI can mean that students are not only learning SQL but also how to interpret and interact with the command line, for many this can be an unwelcome addition to their workload. The use of user interfaces, such as SiS, designed as a transitional tool between highly graphical and textual interfaces can serve to ease this step.

The live updates to the SQL translation and the results are a cornerstone of SiS (Section 6.5.4.1) and were largely well received by the participants. They found that the live UI enabled them to build a good understanding of an SQL SELECT statement while they constructed one using the user interface: *"its really helpful to have the results always showing up as well [as the SQL]"*.

7.2.5 Database visualisation

Some students find visualising the database structure possible without help but many find drawing the database structure beneficial: *"I use the .schema command and then draw out the table [and some contents]"*, this is a common approach, an example of this is shown in Figure 7.9. The database visualisation included in SiS was generally well received, however, many participants believe improvements are needed to maximise its potential. Joins often represent a stumbling block for many students and it requires that they are able to visualise the database as a whole, the visualisation was praised for easing this problem: *"It was great how it showed inner joins, that was the hardest part for me"*. Despite the praise for visualising the database as a whole, for some participants it wasn't clear that this was more than just a visual aid and you had to click it to build a FROM clause: *"It wasn't readily apparent [how to do joins]"*. Other participants failed to realise that to perform multiple joins they were required to click more than one connection in the visualisation: *"I don't think there is much support for multiple joins"*.

One area participants believed needed improvement was in the appearance of some elements of the database visualisation. As the nodes representing connections between tables were biggest many of them were initially drawn to them rather than the smaller table nodes; suggestions to improve this aspect of the system included changing the shape of the nodes or ensuring the table node is the biggest. Participants felt that improvements to the way in which information was added and removed from the visualisation were also needed; in its current state *"zoom"* buttons allow the user to add/remove information from the visualisation. This was described as confusing by some participants: *"you expect it to make a section of the visualisation larger [...] but it actually goes to a different view"*. After participants discussed various different methods of changing the information viewed on the

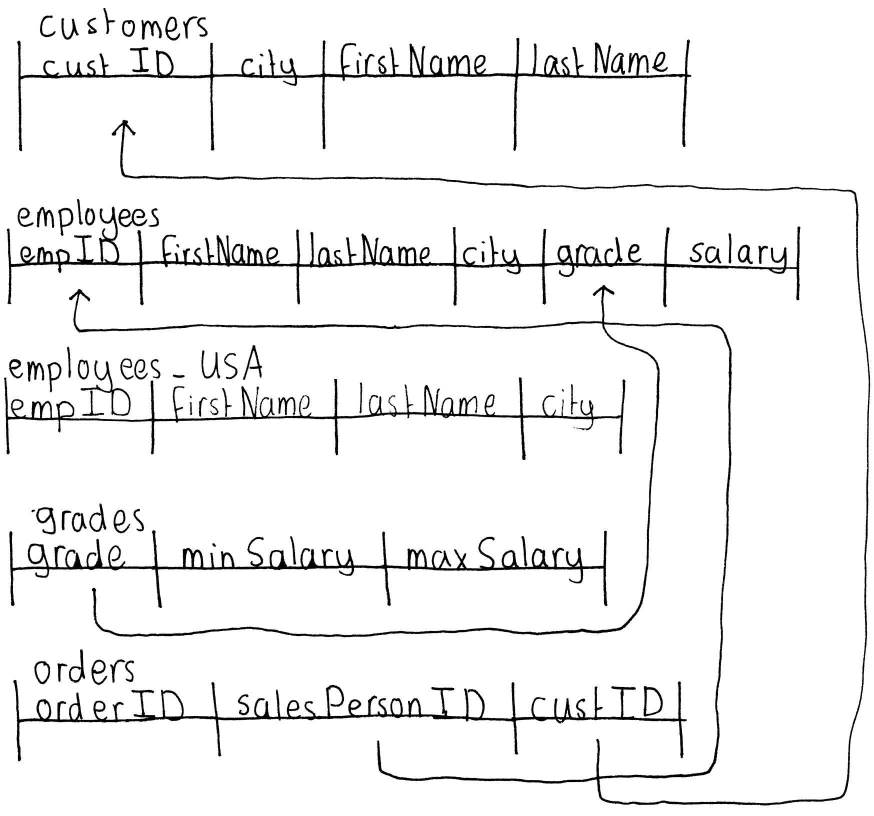


FIGURE 7.9: A participants drawing to help them visualise the database structure

visualisation check-boxes were the most popular choice: *"check-boxes can be used to show different levels of information [...] you can select all the information you want"*.

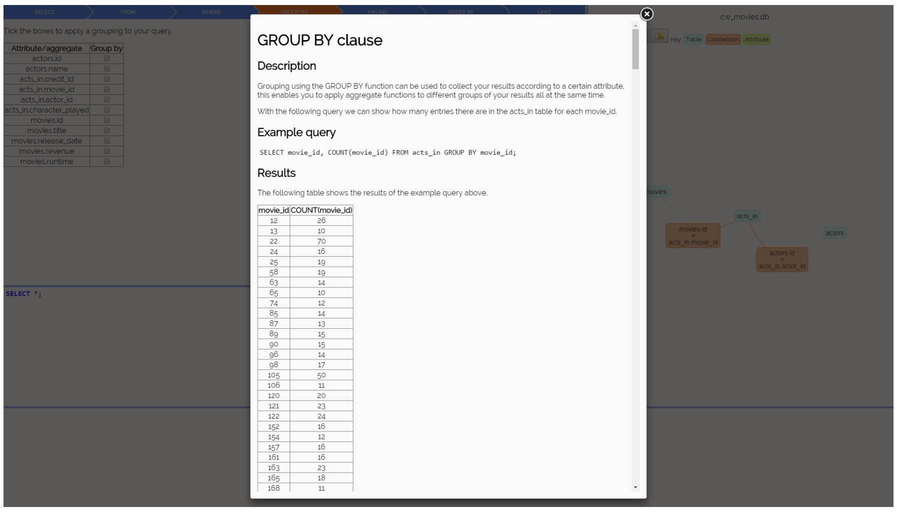
Despite participants believing that the visualisation needed some minor refinement it was highly praised as one of the best features of SiS. The visualisation allowed many participants to build a mental picture of the database without the use of additional diagrams or drawings enabling them to concentrate on building the appropriate queries (see Section 7.2.2 for details). This visualisation style also presents itself as something that could easily be manually recreated by the students should they need to visualise a database without SiS.

7.2.6 Boolean expressions

As discussed in Section 3.3.1.6 Booleans are often difficult for novice users to visualise often resulting in the misuse of AND and OR operations or misunderstandings regarding nested expressions.

To combat the difficulty presented by Boolean expressions SiS includes a graphical representation of Boolean expressions that is aimed at allowing users to build expressions in a graphical environment while still gaining an understanding of how to construct them textually (Section 6.5.4.4). This element of the system was the least well received and was described as *"complicated"* by almost all participants within the focus group. The perception of complexity may have been due to the participants unwillingness to access the help available (Section 7.2.7). The help for this feature described how to create Boolean expressions and included a video showing how this was done; as with many of the help features, these features were rarely accessed with the help button never being clicked throughout the entire study and the video for the WHERE clause only viewed twice (Section 7.2.7). Participants also suggested that their prior experience in textually constructing Boolean expressions meant there was no need for them to use a graphical representation: *"we're doing Boolean expressions in C [and other modules] so we know about AND, OR and XOR etc. Its easy for us to just write it in textual form, there is no need to make Boolean expressions graphical."*. With this in mind it may be more beneficial to replace the graphical representation of Boolean expressions with a simple text input with syntax highlighting in future iterations of SiS.

Despite participants feeling that the Boolean expression visualisation was unnecessary this opinion is largely dictated by their prior understanding of Booleans; had the databases course taken place prior to the other modules in which students learnt about Booleans this feature may have been very useful. While Boolean visualisations may not be necessary for university students it may be more suited to a more inexperienced or younger user base.



GROUP BY clause

Description

Grouping using the GROUP BY function can be used to collect your results according to a certain attribute, this enables you to apply aggregate functions to different groups of your results all at the same time.

With the following query we can show how many entries there are in the acts_in table for each movie_id.

Example query

```
SELECT movie_id, COUNT(movie_id) FROM acts_in GROUP BY movie_id;
```

Results

The following table shows the results of the example query above.

movie_id	COUNT(movie_id)
12	26
13	10
22	70
24	16
25	19
58	19
63	14
65	10
74	12
85	14
87	13
89	15
90	15
96	14
98	17
105	50
106	11
120	20
121	23
122	24
152	16
154	12
157	16
161	16
163	23
165	18
168	11

FIGURE 7.10: Automatically generated help for the GROUP BY clause in SiS

7.2.7 Help

Help provided throughout the study came in two forms: help with SQL (Section 6.5.4.5) and help with SiS (Section 7.2.1.4). SiS included various features to help users understand the various aspects of SQL and as part of the study participants had access to numerous resources to provide them with a comprehensive introduction to SiS.

7.2.7.1 Help with SQL

All of the help with SQL was accessed through the help button displayed on each of the clause pages. Some users found that, despite its presence in the form of a large question mark button (Figures 6.1 and 7.10), the location of the help functionality was not clear enough. Increasing the prominence of help features introduces the risk of it becoming an annoyance or inconvenience for users who do not require any assistance.

Throughout the focus group it became apparent that the lack of use of the help features was not solely as a result of participants being unable to locate it but also of their reluctance to use in-built help in software packages. Participants discussed how they would use a search engine to find help with SQL instead of accessing the help within SiS. There is a stigma attached to in-software help that makes users assume it will be of little help and many users totally avoid it. The quality of the help is irrelevant if users never access it and ensuring that users do use the help available when needed is a difficult task.

This negative attitude towards help in software is not a new phenomenon; Carroll and Rosson (1987) described how the user is often, paradoxically, preoccupied with the end result of their task (learning SQL in this case) that they have little time to use the help functionality, despite it potentially helping them complete their tasks. Grayling (1998) also observed the reluctance to use help: *"After carefully reviewing all the tools, menus, and dialogs, and not finding any commands or functions that would seem to help them with the scenario task at hand, the test subjects would sigh (audibly), and then start back at the beginning, reviewing all the menus, tools, and dialogs again. [. . .] they preferred to revert to the as-yet-unsuccessful trial and error strategy rather than go to the Help menu and review the very material that was designed to assist them."*. User Interface Engineering (2004) noted that the word *"help"* implies that the user must admit failure and recommended the use of *"tips"* or *"hints"* instead; this approach was taken by the University of Arizona when redesigning their library system to increase the number of students accessing the feature (Dickstein and Mills, 2000).

Some users suggested the help would be more obvious if it were to take the form of the Office Assistant *"Clippy"* as used by Microsoft (Figure 7.12). Whilst making the help more obvious is a desirable feature this level of intrusiveness would likely deter the user further. A balance between hidden help and the intrusiveness of the unpopular Office Assistants must be found to ensure the help is used but not disruptive to the users workflow. This could take the form of an introductory tour or a occasional suggestions for help following the use of a new feature.

In order to make the help functionality more appealing in future iterations a number of considerations must be taken into account:

- The terms *"help"* should not be used, favouring *"tips"* or *"hints"* has the potential to increase the likelihood of users accessing this functionality.

- Short, relevant help (with links to more in depth help) ensure users can access the information quickly without breaking away from the task in hand for long periods of time.
- Some prompting for use (e.g. an introductory tour) may make users more aware of the available help but also speed up the process of familiarising themselves with the system.

7.2.7.2 Help with SiS

Throughout the SiS study all participants had a wide range of assistance in the form of videos, documents and contact with the researchers (Section 7.2.1.4). Despite the large amount of help available some students struggled with certain elements of SiS but failed to use the help systems in place.

Videos for each clause were viewed an average of twice over the duration of the entire study (Table 7.8) and no participants emailed researchers for assistance in using SiS. The help functionality within SiS was used 10 times by eight different students throughout the study, this represents an average of one help request for every 567 queries executed. During the focus group one of the participants acknowledged that their request for help was accidental and they immediately closed the associated dialogue box.

The low use of help is as a result of two main factors: the prominence of help features and the reluctance of users to use the help available.

7.2.7.3 Introduction to the software

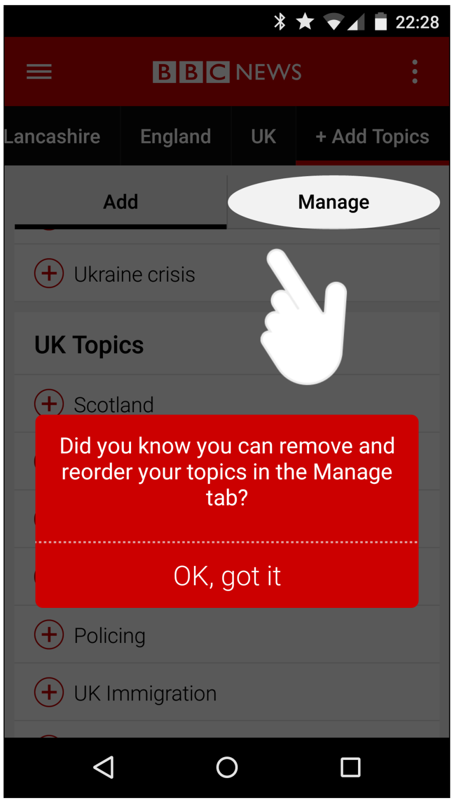
Some participants described their first impression of SiS as a feeling of *“where do I start”* whereas others described their desire to explore: *“I saw SELECT so I started with SELECT”*. Although some users found exploring without instruction satisfactory the general consensus of opinion was that some help or instructions are needed. In the focus group some users wanted discrete help that they could ignore when not needed while others wanted something more prominent: *“you could use*

Video	Views
SELECT clause	2
FROM clause	1
WHERE clause	2
GROUP BY clause	1
ORDER BY clause	2
HAVING clause	2
LIMIT clause	4
Database visualisation	2
Help with clauses	1
What is SiS?	3
Getting Started	2
Total	**22**

Table 7.8: YouTube views during the SiS study

what Microsoft used to and have a little paper clip at the side" (Figure 7.12). One participant suggested the use of *"speech bubbles over aspects [of the user interface]"* as a means to provide information on how to use the system. This approach is relatively common and is often used for new applications or when major changes are made to a user interface, the BBC used this technique following the release of an update to their news application for Android (Figure 7.11) (Google Play, 2015). Other participants believed that a less intrusive introduction to the system would be more appropriate, a popular suggestion was a *"take a tour"* feature. The lack of an introduction embedded into the software may be a contributing factor to the fact that 19 (31% of 61) participants only submitted between 1 and 99 queries.

Requests for help in the form of an animated assistant such as the Microsoft Office Assistant was an unexpected contribution, especially considering the overwhelming unpopularity of these within Microsoft Office. *"Clippy"* was widely considered annoying, intrusive, distracting and even rude (Whitworth, 2005; Xiao et al., 2003) but some success has been found when introducing similar tools into educational applications (Mitrovic and Suraweera, 2000). Given the attitude towards software help observed in this study it seems likely that, despite requests for an assistant, had such a feature been included in SiS it would likely have been ignored or disabled. The suggestion of an intelligent help system that reacts to



Did you know you can remove and reorder your topics in the Manage tab?

OK, got it

FIGURE 7.11: The help offered to users after major changes were made to the user interface of the BBC News Android App in early 2015 (Google Play, 2015)

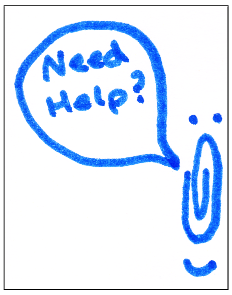


FIGURE 7.12: A drawing by a participant that closely resembles *"Clippy"*, the Microsoft assistant

a users actions is likely a response to the large quantity of information that the participants felt they needed to effectively use all the features of SiS.

7.2.7.4 Error messages

The quality of error messages in SQL are a well recognised problem (Section 2.6.1.3) and some of the participants in the SiS study echoed this: *"If you compare it [SQL errors] to the syntax errors in C it'll print out the line and it'll [C] have the line number and it'll have a little arrow showing where the error is if you're missing a semi-colon or missing a brace or whatever but with SQL it just says "syntax error", it doesn't really say anything apart from that."*.

A request from the students was to improve the usability of error messages by utilising the knowledge of the database that SiS has in order to provide more helpful error messages. Due to the commonly held belief that error messages are unhelpful this issue was not overlooked in the design of SiS but the error messages were intentionally left unchanged. The reason for SiS using the same error messages as SQLite (the database SiS uses) was in an attempt to ease the transition between the graphical environment of SiS and the textual environment in which students must become comfortable using. If the errors were changed to show improved messages that accurately described the problem and how to remedy them then transitioning to the less obvious SQL errors would likely make bug fixing difficult for users of SiS.

A more reasonable solution that would both provide the user with the skills to decode error messages without SiS would be to introduce a means to help the student decipher the errors meaning and correct any problems. The understanding that SiS has of the database structure, and its contents, could allow it the ability to help users find the true meaning of an error, its cause and a means to fix it. In providing students with a set of skills that enables them to handle SQL errors SiS could improve their error handling as they move onto textual environments.

7.2.8 Study limitations

The process of evaluating SiS presented a number of limitations that had the potential to hinder the success of the study. These can largely be divided into two categories: ethical constraints and the willingness of students to participate in the study.

7.2.8.1 Ethical constraints

As with all studies involving current students on an active course there were a large number of ethical considerations to take into account during the planning stage of the evaluation. The process of ethical approval was time consuming and restricted the study structure.

The ethical limitations stemmed from a number of different sources:

- Potential to disadvantage students
- Influence on assessed coursework
- Assignment of groups

The study conducted is a modification of the original design as a result of changes that were dictated by ethical considerations. The initial study design would have spanned the full five weeks of the course utilising the assessed coursework as a major indicator of student progress. Students were to be divided into two groups according to their timetabled practical session with half of the groups using SiS and the other half completing the course as normal.

A major concern, understandably so, within the course administrators stemmed from the potential of creating an unfairly balanced course as a result of the difference in resources available. Students in different groups have the potential to be advantaged or disadvantaged depending upon SiS' ability to enhance or disrupt the learning process. Concerns were raised as to how, if at all, grades would be adjusted to reflect the different resources available

In an attempt to reduce the impact of SiS on the assessed coursework the length of the study was reduced to three weeks, ensuring that all students had access to the same resources throughout the study; as a result, all access to SiS was revoked prior to the assessed coursework being distributed in the fourth week of the course. This significant reduction in study length limited the observations possible and, had it been available to the students, the frequency of use might have increased in response to the motivation to complete the assessed coursework. One participant described their disapproval of SiS' withdrawal in the last two weeks, saying it was *"when we could have used it the most"*.

Week	**Queries**	**No. of users**
Week 1	30	5
Week 2	2523	19
Week 3	3121	12
Total	**5674**	–

TABLE 7.9: The number of queries submitted in SiS per week showing a steady increase as the weeks progressed (week one did not require any SELECT statements).

Table 7.9 shows the steady increase in the number of queries submitted to SiS as the weeks progressed. Had the study been able to continue into the fourth and fifth weeks of the course, when the content became more difficult and the contribution towards a final grade was introduced there would likely have been a further increase of SiS' use. This would have increased the significance of the evaluation results and potentially exposed the software to more students resulting in more feedback in relation to the benefits it brings.

Excluding the last two weeks of the study also made the assessed coursework unavailable for inclusion in the study; as a result the formative assessments became the primary means to quantify student progress. Using these assessments as part of the study introduces a new set of problems with regards to establishing a reliable dataset, these problems are discussed in Section 7.2.8.2.

Prior to taking part in the study students were made fully aware that participation did not guarantee them access to SiS but that their access would be decided at random with the toss of a coin. This technique ensured that students were fairly divided into groups and they had no grounds to dispute the assignment of groups or the benefits this may bring. The initial study design involved splitting the participants into two depending upon their assigned lab sessions; had this approach been taken demonstrations and examples could have been shown to the entire group without having an impact on non-SiS users, this would likely have increased the exposure of SiS and therefore the extent to which it was used. In practice each lab session included a mixture of SiS and non-SiS users meaning demonstrating the tool and providing instructions to the group as a whole was impossible.

An alternative study structure that was considered was to make the tool available to all participants and then compare the improvements made by those who chose to use the tool with those who chose not to. Although this structure would potentially expose more students to SiS and ease some logistical problems of conducting the study this would be at the expense of introducing a self-selection bias. It is likely that students willing to try experimental software such as SiS are also those willing to spend more time ensuring they have a good understanding of the subject matter, resulting in higher assessment scores. This methodology has the potential to falsely show that SiS significantly increased the participants understanding of SQL, when in reality the attitude of the students may have had more impact on their improved understanding.

Ethical constraints dictate that the introduction of experimental software such as SiS cannot be made a compulsory part of the course so students were never instructed to use the tool or provided with demonstrations, as a result, many students never accessed the tool (see Section 7.2.2).

7.2.8.2 Willingness of SiS participants

The use of students within academic studies can introduce a number of limitations, many of which stem from their motivation to participate in the study. In the case of SiS, students had little motivation to complete the various assessments and questionnaires upon which the study heavily relied. The study consisted of two assessments and one questionnaire and participants also granted the researchers access to the weekly formative assessment results.

The first assessment was to be completed before the study commenced, immediately after signing a consent form, without submitting a completed assessment students were not accepted as a participant on the study, this provided the motivation for students to complete the assessment. In each of the three weeks of the study all students (not only those participating in the study) were given a formative assessment; it was made clear to students that the results of these assessments did not contribute towards their final grade for the course but were used as a means to modify and improve the course content based upon their understanding. From 101 participants 85 completed the first formative assessment with 61 and 20 students completing assessments two and three respectively. At the end of the study the students were presented with a questionnaire designed to establish their confidence relating to the different aspects of SQL and an assessment to quantify their improvement, 82 students completed the questionnaire however only 25 completed the final assessment.

Following the study a random selection of 49 students were asked why, as the weeks progressed, there was an increasing reluctance to complete the formative assessments (see Appendix C for the questionnaire). A wide variety of reasons were given for this; some believed the content was too difficult (4 out of 49 students) while others felt it wasn't challenging enough (3 out of 49 students). The most common reason given, with 41% of the students asked (20 out of 49) agreeing, was the lack of contribution towards the course grade: *"It didn't count towards our grades so there was less incentive to complete them"*. Research into the link between motivation in students and the use of formative assessment has been popular in recent years and the general consensus is that the use of such assessment serves to increase the motivation in students (Black and Wiliam, 1998; Brookhart, 1997;

Cauley and McMillan, 2010; McMillan, 1996); despite this, our findings show that motivating students to complete the formative assessments can be difficult.

In contrast to the relatively low number of participants completing the formative assessments and the questionnaires for the study all 101 participants completed the assessed coursework presented to them in the final weeks of the course, after the study finished, however these results are not eligible for inclusion in the study. This stark contrast in student uptake shows that the low number of students completing the resources required for their studies is not as a result of a lack of understanding of the course or a lack of engagement with the study but a lack of motivation. The motivation to complete the initial assessment was to gain a place on the study while the motivation for the assessed coursework its contribution towards the students final grade for the course.

Motivation can be, amongst other types, intrinsic or extrinsic; intrinsic motivation is *"behaviour that is energised by the pleasure derived from engaging in the activity"* while extrinsic motivation is driven by rewards or punishments given by others (Hill, 2013). The results of this study show that the participants were largely extrinsically motivated and the reward of grades or participation in a study drive them to complete certain tasks. Despite much of the existing research within the literature concluding that formative assessments can increase motivation in many students this study showed that some students struggled to find the motivation to complete the formative assessments as the rewards were not considered sufficient for many students. However, the sense of a tangible reward such as participation in a research study or a contribution towards their module grade encouraged all participants to complete the necessary assessments.

7.2.9 SiS evaluation summary

The evaluation of SiS involved introducing it into a live databases course and observing how it was used. It presented a numerous key points, the following were discussed:

- The need for an alternative interface for the process of teaching SQL
- The benefit of presenting the users with a graphical representation of the database structure
- The need, or lack thereof, for a visualisation to improve the understanding of Boolean expressions
- The low use of help and the reasons for this
- Limitations of the study

Participants in this study stressed some of the difficulties associated with learning SQL using a command driven user interface alone; they are considered a daunting prospect as a result of their lack of assistance and poor feedback in response to errors. SiS showed that presenting the students with a graphical representation of the database schema can assist them in understanding the database structure and provide them with a good understanding of join operations and how to build them. SiS also includes a graphical representation of Boolean expressions along with the database structure; although Booleans can be a difficult concept for many users to grasp, particularly when transferring skills from the spoken word, this was not the case for many participants. The prior experience in constructing Boolean expressions that the participants had allowed them to build the necessary criteria without graphical support, as a result many participants found the graphical representation of Booleans unnecessarily complex and difficult to grasp. Some participants with access to SiS found the lack of introduction meant that getting started with the software was difficult, in anticipation of this problem there were a number of tools available to help the students. Despite the ample help available it was rarely accessed resulting in some students failing to get beyond an initial glance at the software. The reason for this lack of use was, in part, due to the way in which it was presented but also in response to the stigma associated with help in software. Help is regularly ignored and the challenge in improving its use lies in making it more appealing to the user rather than in improving the quality of its contents. Limitations throughout the study restricted both the design and results available from the evaluation; these limitations came in the form of both participant introduced problems, such as a lack of motivation to participate in the study, and ethical considerations, such as providing additional resources to some students but not others.

In summary, this evaluation of SiS illustrates the potential for introducing graphical applications into the learning environment for database students. Although some elements of the system may require refinement the features of SiS were largely met with a positive reception.

7.3 Chapter summary

This chapter presents the evaluation of two database applications, SiS (Chapter 5) and CAFTAN (Chapter 6). Both evaluations show that the systems introduce real benefits into their respective fields and offer the potential for further improvement.

CAFTAN was designed for the novice user who must interact with relational databases without an understanding of its structure of the appropriate query language. The interpretations of keyword queries CAFTAN made were compared against both bespoke systems and, crucially, those made by humans. The evaluation of CAFTAN provides:

- An insight into how humans expect keyword queries to be interpreted.
- An evaluation that illustrates that CAFTAN outperforms bespoke applications by consistently interpreting queries in a way close to human interpretations.
- A rigorous performance evaluation showing that CAFTAN is capable of returning appropriate results in a timely fashion.
- A data set that can be reused in future iterations of CAFTAN or in the development of other applications.

SiS is designed as an educational application to ease students into the process of building textual queries. The evaluation involved comparing the process of learning SQL with and without access to SiS. It highlighted:

- The need for improved interfaces for educational purposes and the benefits they can bring.
- The impact of introducing graphical representations of the database structure and Boolean expressions.
- The stigma attached to help integrated into software packages and the reluctance to use such systems.

Chapter 8

Conclusions

This research aimed to explore how the process of extracting information from relational databases can be improved; this area of research is important due to the widespread use of relational databases in a broad array of applications by a large number of users.

The initial aim of this work outlined in Section 1.1 was to *"ease access to relational databases"*; this problem was broken down into two further problems:

1. Identify the users of relational databases and their needs.

2. Investigate how software can be used to improve the way in which people interact with relational databases.

This thesis identifies two main classes of user and describes the differences in the way these users interact with relational databases. Two applications designed to assist these two types of user in extracting information were developed and subsequently evaluated.

8.1 Background, motivation and literature review

Chapter 2 highlights some background information in relation to this research including the various database models along with the different users of such systems. The relational model is by no means the only way in which data can be modelled but until recently it enjoyed a relatively uninterrupted reign as the default database choice for almost all applications. The rise of NoSQL databases in the early 21st century provided developers with an alternative to the relational model. It is important to view NoSQL databases as alternatives rather than replacements to the relational model; they offer some functionality that is unavailable in the relational model, just as the relational model offers some functionality unavailable in NoSQL alternatives (Section 2.2.3).

The widespread use of databases means that there are large number of different users who interact with them. These users range from the expert who is knowledgable in the area of database design and query languages to the novice who has little or no understanding of such areas. There are numerous classifications of users in the literature (Catarci and Santucci, 1995; Chen, 1999; Elmasri and Navathe, 2010; Jarke and Vassiliou, 1985; Martin, 1973; Shneiderman, 1978) but many define a large number of categories in response to the wide range of database operations that different users can make (designing, building, modifying and extracting data). Section 2.4.1 describes a refined categorisation of users that is the result of purely focussing on the extraction of data and ignoring other database interactions. Focussing on this small area of database interactions means that many of the previous categories can be merged into one, this results in the following categories of user:

- **Novice** (Section 2.4.1.1): Those with no understanding of the database structure or query languages.
- **Expert** (Section 2.4.1.2): Those with knowledge of textual query languages and the underlying database schema.
- **Student** (Section 6.1): Those users transitioning from novice to expert. They are in the process of learning textual SQL and schema design.

The potential to improve the process of extracting information from relational databases was initially, albeit briefly, explored in an undergraduate project within the context of email data (Section 2.3.1). The Elcee application uses a graphical representation of Boolean expressions that enables users, with little expert knowledge, to be able to build potentially complex Boolean expressions. This Boolean visualisation style, the Sandpit Search, was also used in a proof-of-concept application for SSQL (Section 4.3.3) along with the SiS prototype (Section 6.5.4.4)

Chapter 3 presents a detailed literature review of a number of areas relevant to the process of extracting information from databases:

- Visual Query Languages (Section 3.1)
- Keyword search (Section 3.2)
- Educational applications (Section 3.3)

The first advances in the area of extracting such information came in the form of visual query languages (VQLs); these were initially form based (e.g. QBE by Zloof (1975)) but, in response to hardware and software advancements, they became graphically more complex and used icons and diagrams. Although an improvement on their textual counterparts VQLs still demand some knowledge of the database structure. In order to make databases more accessible to the novice user keyword search systems became increasingly popular and they use a range of different techniques to translate keyword search terms to appropriate results. The expert user typically interacts with databases using textual query languages; the reason for this is that their power and flexibility that means they are capable of describing complex and accurate queries. Learning such a language can be difficult and, as a result, there are a number of applications that are designed to ease the process. Educational applications typically use animations or analysis to provide students with knowledge that aims to improve their understanding of query languages.

Although there is a large body of work that explores both keyword search and educational applications they are not without limitations. Many keyword applications are limited in their ability to build Boolean expressions, frequently limited to the use of AND or OR operations rather than an appropriate mix of both. Some applications are also limited in the variety of data types they can interpret with the vast majority of systems either interpreting non-textual data as text or simply ignoring it. Much of the existing work within the field of educational applications is also limited with many only suitable for the exploration of complete queries rather than being aimed at helping students with the construction of queries. Many educational applications also fail to address some well recognised difficulties associated with learning SQL such as visualising the database, constructing join operations and building Boolean expressions.

8.2 Prototype development

In response to the needs of various different database users a number of prototypes were developed as part of this work.

Building join operations can be difficult for both developers and end users of databases; Chapter 4 is dedicated to exploring a potential solution to this problem. It introduces Shorthand SQL (SSQL), a software library that uses path finding algorithms to automate the production of joins. As a proof-of-concept SSQL was integrated into a number of different applications to simplify the query process for the end user without burdening the developer with the task of generating links between relations. The use of SSQL was also explored within the context of CAFTAN and SiS; CAFTAN uses SSQL to connect the relations containing given keywords, enabling it to perform cross-relation queries in a generic setting (Section 5.4.1.4). The use of SSQL in an educational environment was also explored but subsequently dismissed because of its potential to build a reliance on the system to the point at which transitioning to a textual query language would become difficult (Section 6.3).

Chapter 5 describes the Context Aware Free Text ANalysis (CAFTAN) prototype that uses a weighted index to provide appropriate responses to keyword queries in a generic environment. CAFTAN builds a detailed index that associates a weight with each occurrence of textual, numeric or temporal data. The weights are combined in response to search terms to locate the most likely interpretation of a query and the necessary relations are joined using SSQL. Features of CAFTAN include:

- **Boolean operations** (Section 5.4.1.3): The index for CAFTAN is used in such a way that enables it to build Boolean expressions with an appropriate mix of AND and OR operations.
- **Many-many queries** (Section 5.4.2.1): CAFTAN automtically recognises the cardinality between relations in a query and is capable of building queries that can effectively extract information from relations connected in this way.
- **Data types** (Section 5.4.2.2): Unlike many existing systems CAFTAN will recognise the presence of different data types in a query and will build appropriate queries in response to this.
- **Ranges** (Section 5.4.2.3): The recognition of data types also extends to ranges; CAFTAN is capable of recognising ranges, denoted with a single dash, and applying them appropriately to the data within the database.

Chapter 6 begins by introducing the *student* user in addition to the primary categories of *novice* and *expert* users (Section 6.1). In response to the needs of the student user the SQL in Steps (SiS) prototype was developed; its primary aim was to enable students to graphically build queries whilst building an understanding of their textual counterparts. Key features of SiS include:

- **Textual translations and live results** (Sections 6.5.4.1 and 6.5.4.2): The main focus of SiS is its live display of textual queries and their results. These remain up to date as the student graphically builds a query.
- **Visualisation of the database** (Section 6.5.4.3): Visualising the database structure as a whole is often considered difficult by many students; the graphical representation of the database in SiS enables students to view the schema as a whole in various levels of detail.

- **Graphical representations of Booleans** (Section 6.5.4.4): Booleans are frequently difficult for novice users to understand; in an attempt to combat this SiS includes the graphical representation of Boolean expressions originally developed for Elcee (Section 2.3.1).
- **Contextual help** (Section 6.5.4.5): To provide students with examples that might help them with their learning process SiS automatically generates examples for each of the clauses within the context of the database the student is currently using. This avoids the need for users to learn a new database schema to understand an example.
- **Customisable/generic implementation** (Section 6.5.4.8): No elements of SiS are developed for a specific database and it is capable of working with any SQLite database. This generic implementation along with numerous customisable features means that SiS is suitable for a wide array of students with a wide array of needs.

8.3 Evaluations and conclusions

The evaluations of both CAFTAN and SiS are presented in Chapter 7. The evaluation of CAFTAN (Section 7.1) was twofold and involved quantifying the accuracy of query interpretations along with the performance of indexing and query execution. The methodology used to evaluate CAFTAN is both unique and powerful in that it offers an insight into the way in which people interpret keyword queries along with producing a dataset that can be used in future iterations of CAFTAN or with other search applications. Comparing query interpretations made by humans and two popular bespoke applications showed that, even without customisations, CAFTAN consistently interpreted queries more accurately than bespoke applications. When compared to the *"gold standard"* dataset generated by participants CAFTAN interpreted 64% of queries in the same way, with the addition of customisations increasing this to 81%. This represents significant benefits when compared to IMDb and TMDb, which interpreted queries in the same way as humans 24% and 26% of the time respectively. CAFTAN performed particularly well when presented with queries that span multiple relations or those beyond the scope of the *"template queries"* used by many bespoke applications (Luo et al., 2007) (Section 7.1.4.3). Rigorous performance tests on CAFTAN show that, for the movie database (Appendix A), CAFTAN is capable of supplying fast query responses. Tests showed that CAFTAN consistently returns results in less than half a second, irrespective of the number of search terms or type of query.

To evaluate SiS it was integrated into a live undergraduate databases course and the intention was to compare those students who used SiS against those who did not (Section 7.2). Some limitations of the study (Section 7.2.8) meant this comparison was not entirely possible but the potential benefits of the system were made clear. Participants expressed concerns with learning SQL in a purely textual environment and described how features such as the graphical representation of the database structure improved their understanding of the language. Despite the majority of SiS features being well received not all features were popular; the Boolean visualisation, for example, was seen as confusing by many users. The help system used in SiS also prompted much discussion; participants agreed there was a need for such help but further development into the way in which it is presented was necessary. A common dislike of help systems means they demand careful consideration to be effectively integrated into a system. Although the study structure limited the ability to make quantifiable claims regarding the benefits of SiS the qualitative data gathered shows the system has real potential to provide students with benefits that are unavailable, yet sorely needed, in a textual environment.

8.4 Future areas of research

Although the benefits of the work presented here are significant within the context of both the novice and student users, there is undoubtedly scope for further work and improvements in this field. The following outlines a number of areas with the potential for further work within the field.

8.4.1 CAFTAN and real world applications

The methodology used in the evaluation of CAFTAN (Section 7.1.1) is powerful and brings numerous benefits such as the production of a reusable data set along with an insight into the habits and expectations of the keyword search user.

Despite these benefits, the software was never used by participants and this somewhat limits the extent to which it can be considered a viable alternative for real world applications. To assess the potential for distributing CAFTAN as part of a real world system it must be used in such an environment for testing purposes. To encourage the mass use of CAFTAN the application would likely have to be modified or reimplemented to aid the distribution of the software. It would be possible to reimplement the algorithms CAFTAN uses in almost any programming language that facilitates a database connection so a version of the system could conceivably be integrated into a web based application allowing users to query a database stored on a web server without demanding any specialist hardware or software on the participants machine.

By integrating the capabilities of CAFTAN into a web based application with a facility for users to rank the quality of the results of a query it would be possible to establish a good understanding of the potential for it to be used in a real world scenario.

8.4.2 Automating CAFTAN strengths

CAFTAN enables database administrators to specify strengths that can be used to influence the interpretation of keyword queries (Section 5.4.1.5). These strengths act as a multiplier and increase or decrease the weights associated with search terms in the CAFTAN index; using sensible values for these weights can improve the quality of the interpretations made by CAFTAN, preventing the association of terms with largely irrelevant attributes.

These strengths must be manually specified by the database administrator using their knowledge of the schema and attribute contents. In future iterations of CAFTAN it would be possible to automate the production of these strengths by monitoring how the re-query functionality is used (**??**). Without strengths applied to the movie database there is the potential for CAFTAN to misinterpret a query for a movie title as one for a tagline; following this erroneous interpretation the user is likely to request a re-query which indicates that the tagline interpretation was incorrect. Observing similar behaviour by multiple users would make it possible to automatically reduce the strength associated with the tagline to avoid future misinterpretations.

8.4.3 Redesigned SiS study

Although SiS showed a lot of potential for improving the way in which learning SQL can be approached the participants also addressed a number of ways in which it could be improved. These include improvements in relation to the construction of Boolean expressions (Section 7.2.6) and improvements regarding the help and introductions given to students (Section 7.2.7). In light of the study and focus group the appeal and benefits of SiS have the potential to be greatly improved, in light of these improvements a more comprehensive study involving SiS could be carried out. The study could be improved in the following ways:

- The duration of the study could be extended (ethical and departmental approval permitting)
- The exposure of SiS could be improved, presenting it as an option when students approach a new or challenging topic.

- Decreased reliance on formative assessment and study based assessment. Section 7.2.8.2 describes the lack of motivation for students to complete these assessments so the study should not rely on them.

- Analysing the behaviour of those participants not using SiS (including speed of query production and number of executions etc. See Section 8.4.4)

- Analysing the participants use of textual SQL after the study finishes. This, coupled with the analysis of non-SiS participants, would enable a direct comparison between the habits of those participants exposed to SiS against those without access.

8.4.4 Observational study on textual queries

The evaluation of SiS (Section 7.2) attempts to show how the introduction of a graphical system can improve the way in which students learn SQL. While this evaluation undoubtedly highlighted a number of ways in which the learning process could be improved its comparison against textual systems could have been improved. All analysis of learning SQL in a textual environment was based upon existing literature (Section 3.3.1) and the opinions of students (Section 7.2.4) without an in-depth analysis of the problem.

To establish a detailed understanding of how new students interact with textual SQL would require an observational study. Such a study would be need to be unobtrusive to ensure the study itself did not impact on the learning process. This could be achieved using a simple keylogging application that captures all the interactions between a student and a database and would enable the collection of data relating to:

- Number of query executions

- How databases are visualised (use of *.schema* or *describe* commands)

- Error handling (how quickly errors are resolved)

Although this study would provide a valuable source of data relating to how new students interact with databases in a textual environment it would undoubtedly bring ethical constraints as a result of the use of keylogging software. The process of capturing key strokes would need to be entirely voluntary and the participants would need to be able to pause or stop the logging at any time.

8.4.5 Observational study on the use of help

The help made available to participants during the evaluation of SiS was rarely used and the reason for this is largely due to the stigma associated with such assistance integrated into software applications. There is little research into the reasons for this negative association or the means to remedy it. To provide an insight into the attitude that users have towards help systems an observational study could be carried out to ascertain when, if ever, a user would use the software help.

To provide this information users could be presented with a small application (of any type) with the task of completing an impossible task. After realising the task is not obvious the user will have a number of options:

- Continue, in vain, manually searching through the functions of the program to find the means to complete the task.
- Access the help.
- Give up and conclude that the task is impossible.

If the user were to access the appropriate help menu in the application they would be met with a message informing them that their participation in the study is over and the task is in fact impossible. Various metrics could be measured while participants complete the study, including the time taken to locate the help menu or give up. Throughout the study different help menus could be used to investigate the extent to which the style of help effects how quickly users are willing to access it. The help styles that could be used include:

- Traditional help menu in toolbar
- *"Take a tour"* That can be dismissed at any point
- The use of *"Tips"* or *"Hints"* rather than the term *"Help"*
- Microsoft Office Assistant style help that attempts to offer automatic guidance to users

The findings of the SiS study suggest that encouraging users to access the help features will be difficult and very few users will use it before giving up and failing to complete the task. Finding the best means of presenting help would have huge benefits for the way in which software makes assistance available and could potentially be applied to a broad range of different software applications.

8.5 Closing remarks

This thesis described the need to simplify access the contents of relational databases; their widespread use means a massive number of different users demand access to their contents through the use of applications that reflect their technical understanding. Two main classes of user were defined: novice and expert with a third category of students describing those users transitioning from novice to expert.

In response to the needs of novice and student users two prototypes were designed, developed and evaluated. CAFTAN is a keyword search application that translates search terms to SQL that can be executed directly on a relational database. The evaluation of CAFTAN showed that it is capable of consistently and quickly interpreting search terms in the same way as humans, out performing two bespoke applications in many areas. An educational application, SiS, was developed to help students become expert users; it utilises many graphical components in order to assist users in learning the intricacies of textual SQL. The integration of SiS into a live undergraduate databases course showed that the inclusion of carefully designed graphical elements can assist new users in many areas such as building join operations and maintaining an awareness of the result that a query will yield.

Although both applications are designed to meet similar goals, easing access to relational databases, they are significantly different in response to the category of user they are designed for. These prototypes offer interesting improvements over much of the existing work in the field and also provide the potential for further research and integration into real world applications.

References

Abelló, A., Rodríguez, M. E., Urpí, T., Burgués, X., Casany, M. J., Martín, C., and Quer, C. (2008). LEARN-SQL: Automatic Assessment of SQL Based on IMS QTI Specification. In *2008 Eighth IEEE International Conference on Advanced Learning Technologies*, pages 592–593. IEEE.

Aditya, B., Bhalotia, G., Chakrabarti, S., Hulgeri, A., Nakhe, C., Parag, P., and Sudarshan, S. (2002). BANKS: browsing and keyword searching in relational databases. In *VLDB '02 Proceedings of the 28th international conference on Very Large Data Bases*, pages 1083–1086. VLDB Endowment.

Agrawal, S., Chaudhuri, S., and Das, G. (2002). DBXplorer: a system for keyword-based search over relational databases. In *Proceedings 18th International Conference on Data Engineering*, pages 5–16, Washington DC, USA. IEEE Comput. Soc.

Ahlberg, C. and Shneiderman, B. (1994). Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *CHI '94 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 313–317.

Alexa (2015a). Alexa: imdb.com Site Overview. http://www.alexa.com/siteinfo/imdb.com. Accessed: 01 Apr 2015.

Alexa (2015b). Alexa: themoviedb.org Site Overview. http://www.alexa.com/siteinfo/themoviedb.org. Accessed: 01 Apr 2015.

Allen, G. N. (2000). WebSQL : An Interactive Web Tool for Teaching Structured Query Language. In *Americas Conference on Information Systems*, pages 2066–2071.

Andries, M. and Engels, G. (1996). A Hybrid Query Language for an Extended Entity-Relationship Model. *Journal of Visual Languages and Computing*, 7(3):321–352.

Android (2015). android.database.sqlite | Android Developers. http://developer.android.com/reference/android/database/sqlite/package-summary.html. Accessed: 26 Jun 2015.

Apiary (2015). The Movie Database API. http://docs.themoviedb.apiary.io. Accessed: 24 Aug 2015.

Apple (2015). Persistent Store Features. https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CoreData/Articles/cdPersistentStores.html. Accessed: 26 Feb 2015.

Aversano, L., Canfora, G., De Lucia, A., and Stefanucci, S. (2002). Understanding SQL through iconic interfaces. In *Proceedings 26th Annual International Computer Software and Applications*, pages 703–708. IEEE Comput. Soc.

Baid, A., Rae, I., Li, J., Doan, A., and Naughton, J. (2010). Toward scalable keyword search over relational data. In *Proceedings of the VLDB Endowment*, volume 3, pages 140–149. VLDB Endowment.

Balmin, A., Hristidis, V., and Papakonstantinou, Y. (2004). Objectrank: authority-based keyword search in databases. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, pages 564–575.

Beale, R. and Sharples, M. (2002). Design Guide for Developers of Educational Software. *British Educational Communications and Technology Agency (Becta).*

Bederson, B. B., Clamage, A., Czerwinski, M. P., and Robertson, G. G. (2004). DateLens: A Fisheye Calendar Interface for PDAs. *Transactions on Computer-Human Interaction*, 11(1):90–119.

Benzi, F., Maio, D., and Rizzi, S. (1999). VISIONARY: a Viewpoint-based Visual Language for Querying Relational Databases. *Journal of Visual Languages & Computing*, 10(2):117–145.

Berk, E. (2003). JLex: A Lexical Analyzer Generator for Java(TM). http://www.cs.princeton.edu/~appel/modern/java/JLex/. Accessed: 30 May 2013.

Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., and Sudarshan, S. (2002). Keyword searching and browsing in databases using BANKS. In *Proceedings 18th International Conference on Data Engineering*, pages 431–440. IEEE Comput. Soc.

Black, P. and Wiliam, D. (1998). Assessment and Classroom Learning. *Assessment in Education: Principles, Policy & Practice*, 5(1):7–74.

Boyle, J. M., Bury, K. F., and Evey, R. J. (1983). Two Studies Evaluating Learning and Use of QBE and SQL. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 27(7):663–667.

Brin, S. and Page, L. (2012). Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 56(18):3825–3833.

Brookhart, S. M. (1997). A Theoretical Framework for the Role of Classroom Assessment in Motivating Student Effort and Achievement. *Applied Measurement in Education*, 10(2):161–180.

Burnett, M. M., Baker, M. J., Bohus, C., Carlson, P., Yang, S., and Vanzee, P. (1995). Scaling-Up Visual Programming-Languages. *Computer*, 28(3):45–54.

Carey, M., Haas, L., Maganty, V., and J.Williams (1996). PESTO: An integrated query/browser for object databases. *Proceedings of VLDB*, pages 203–214.

Carroll, J. M. and Rosson, M. B. (1987). Paradox of the active user. In *Interfacing thought: cognitive aspects of human-computer interaction*, pages 81–111, Cambridge (MAS). MIT Press.

Catarci, T., Chang, S. K., Costabile, M. F., Levialdi, S., and Santucci, G. (1996). A graph-based framework for multiparadigmatic visual access to databases. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):455–475.

Catarci, T., Costabile, M. F., Levialdi, S., and Batini, C. (1997). Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8:215–260.

Catarci, T. and Santucci, G. (1995). Are visual query languages easier to use than traditional ones?: an experimental proof. In *Proceedings of the HCI'95 conference on People and computers X*, pages 323–338, Huddersfield, United Kingdom. Cambridge University Press.

Cauley, K. M. and McMillan, J. H. (2010). Formative Assessment Techniques to Support Student Motivation and Achievement. *The Clearing House: A Journal of Educational Strategies, Issues and Ideas*, 83:1–6.

Cembalo, M., De Santis, A., and Ferraro Petrillo, U. (2011). SAVI: A new System for Advanced SQL Visualization. In *Proceedings of the 2011 conference on Information technology education - SIGITE '11*, pages 165–170, New York, New York, USA. ACM Press.

Chaudhuri, S., Das, G., Hristidis, V., and Weikum, G. (2006a). Probabilistic information retrieval approach for ranking of database query results. *ACM Transactions on Database Systems*, 31(3):1134–1168.

Chaudhuri, S., Das, G., Hristidis, V., and Weikum, G. (2006b). Probabilistic information retrieval approach for ranking of database query results. *ACM Transactions on Database Systems*, 31(3):1134–1168.

Chen, Z. (1999). *Computational Intelligence for Decision Support*. CRC Press, Boca Raton, Florida, USA, 1 edition.

Cigas, J. and Kushan, B. (2010). Experiences with online SQL environments. *Journal of Computing Sciences in Colleges*, 25(5):251–257.

Clark, G. J. and Wu, C. (1994). DFQL: Dataflow query language for relational databases. *Information & Management*, 27(1):1–15.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.

Codd, E. F. (1974). Seven steps to rendezvous with the casual user. In *IFIP Working Conference Database Management*, Research report San Jose Research Laboratory, pages 179–200. IBM Thomas J. Watson Research Division.

Cooper, J. and James, A. (2009). Challenges for Database Management in the Internet of Things. *IETE Technical Review*, 26(5):320–329.

Cvetanovic, M., Radivojevic, Z., Blagojevic, V., and Bojovic, M. (2011). ADVICE—Educational System for Teaching Database Courses. *IEEE Transactions on Education*, 54(3):398–409.

Czejdo, B., Rusinkiewicz, M., Embley, D., and Reddy, V. (1989). A visual query language for an ER data model. In *IEEE Workshop on Visual Languages*, pages 165–170.

Dabbish, L. A. and Kraut, R. E. (2006). Email Overload at Work: An Analysis of Factors Associated with Email Strain. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 431–440, Banff, Alberta, Canada.

Danaparamita, J. and Gatterbauer, W. (2011). QueryViz: helping users understand SQL queries and their patterns. In *Proceedings of the 14th International Conference on Extending Database Technology*, pages 558–561, Uppsala, Sweden.

Dar, S., Entin, G., Geva, S., and Palmon, E. (1998). DTL's DataSpot: Database Exploration Using Plain Language. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 645–649.

de Raadt, M., Dekeyser, S., and Lee, T. Y. (2006). Do students SQLify ? improving learning outcomes with peer review and enhanced computer assisted assessment of querying skills. In *Proceedings of the 6th Baltic Sea conference on Computing education research Koli Calling 2006 - Baltic Sea '06*, pages 101–108, New York, New York, USA. ACM Press.

de Raadt, M., Dekeyser, S., and Lee, T. Y. (2007). A system employing peer review and enhanced computer assisted assessment of querying skills. *Informatics in Education*, 6(1):163–178.

De Virgilio, R., Guerra, F., and Velegrakis, Y. (2012). *Semantic Search over the Web*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Dekeyser, S., de Raadt, M., and Lee, T. Y. (2007). Computer assisted assessment of SQL query skills. In *ADC '07 Proceedings of the eighteenth conference on Australasian database - Volume 63*, pages 53–62. Australian Computer Society, Inc.

Demidova, E., Fankhauser, P., Zhou, X., and Nejdl, W. (2010). DivQ: diversification for keyword search over structured databases. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 331–338, Geneva, Switzerland. ACM.

Denning, A. (2013). Using the SQLite database engine with Windows Phone 8 apps. http://blogs.windows.com/buildingapps/2013/03/12/using-the-sqlite-database-engine-with-windows-phone-8-apps/. Accessed: 26 Jun 2015.

Deokmin Haam, Ki Yong Lee, and Kim, M. (2010). Keyword search on relational databases using keyword query interpretation. In *5th International Conference* *on Computer Sciences and Convergence Information Technology*, pages 957–961. IEEE.

Dickstein, R. and Mills, V. (2000). Usability testing at the University of Arizona Library: how to let the users in on the design. *Information technology and libraries*, 19(3):144–151.

Dietrich, S. W., Eckert, E., and Piscator, K. (1997). WinRDBI: A Windows-based Relational Database Educational Tool. *ACM SIGCSE Bulletin*, 29(1):126–130.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

Dillon, E., Anderson-Herzog, M., and Brown, M. (2012). Studying the Novice's Perception of Visual Vs. Command Line Programming Tools in CS1. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 56(1):605–609.

Ding, B., Yu, J. X., Wang, S., Qin, L., Zhang, X., and Lin, X. (2007). Finding Top-k Min-Cost Connected Trees in Databases. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 836–845. IEEE.

Drupal (2015). Step 2: Create the database | Drupal.org:. https://www.drupal.org/documentation/install/create-database#mysql_commands. Accessed: 08 Jul 2015.

Elmasri, R. and Navathe, S. B. (2010). *Fundamentals of Database Systems*. Addison-Wesley.

Embley, D. W. (1989). NFQL: the natural forms query language. *ACM Transactions on Database Systems*, 14(2):168–211.

Epstein, R. G. (1991). The TableTalk query language. *Journal of Visual Languages & Computing*, 2(2):115–141.

Fakhraee, S. and Fotouhi, F. (2012). DBSemSXplorer: Semantic-based Keyword Search System over Relational Databases for Knowledge Discovery. In *Proceedings* *of the Third International Workshop on Keyword Search on Structured Data - KEYS '12*, pages 54–62, Scottsdale, Arizona, USA. ACM Press.

Fleming, S. (2007). The End of an Architecture Era (It's Time for a Complete Rewrite). *Architectural Theory Review*, 12(2):195–208.

Garner, P. and Mariani, J. (2015a). Learning SQL in steps. In *The 6th International Conference on Education, Training and Informatics*, Orlando, Florida, USA.

Garner, P. and Mariani, J. (2015b). Learning SQL in steps. *Journal on Systemics, Cybernetics and Informatics*, 13(4):19–24.

Golenberg, K., Kimelfeld, B., and Sagiv, Y. (2008). Keyword proximity search in complex data graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, pages 927–940, New York, New York, USA. ACM Press.

Google (2007). Search 101. https://www.youtube.com/watch?v=syKY8CrHkck&t=22m03s. Accessed: 18 Jun 2015.

Google (2012). Zeitgeist 2012 - Google. http://www.google.com/zeitgeist/2012/#the-world. Accessed: 24 Jul 2015.

Google Play (2015). BBC News - Android Apps on Google Play. https://play.google.com/store/apps/details?id=bbc.mobile.news.uk. Accessed: 30 Apr 2015.

Grayling, T. (1998). Fear and Loathing of the Help Menu : A Usability Test of Online Help. *Technical Communication*, (2):166–177.

Greene, S. L., Devlin, S. J., Cannata, P. E., and Gomez, L. M. (1990). No IFs, ANDs, or ORs: A study of database querying. *International Journal of Man-Machine Studies*, 32(3):303–326.

Grillenberger, A. and Brinda, T. (2012). eledSQL – A New Web-Based Learning Environment for Teaching SQL at Secondary School Level. In *Proceedings of the*

7th Workshop in Primary and Secondary Computing Education on - WiPSCE '12, pages 101–104, Hamburg, Germany. ACM Press.

Hadfield, C. (2014). Chris Hadfield on Twitter. https://twitter.com/Cmdr_Hadfield/status/538319778629566464. Accessed: 28 Nov 2014.

Haerder, T. and Reuter, A. (1983). Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317.

Hartl, A. and Weiand, K. (2009). *A Visual Rendering of a Semantic Wiki Query Language*. PhD thesis, Ludwig Maximilian University, Munich.

Harvard University (2015). Syllabus. http://cdn.cs50.net/2015/fall/lectures/0/w/syllabus/cs50/cs50.html. Accessed: 29 Sep 2015.

He, H., Wang, H., Yang, J., and Yu, P. S. (2007). BLINKS: Ranked Keyword Searches on Graphs. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data - SIGMOD '07*, pages 305–316, New York, New York, USA. ACM Press.

Heiler, S. and Rosenthal, A. (1985). G-WHIZ, a Visual Interface for the Functional Model with Recursion. In *Proceedings of the 11th International Conference on Very Large Data Bases*, volume 11, pages 209–218, Stockholm, Sweden. VLDB Endowment.

Hildreth, C. R. (1989). *Intelligent Interfaces and Retrieval Methods for Subject Searching in Bibliographic Retrieval Systems.* Library of Congress, Washington, DC, USA.

Hill, A. P. (2013). Motivation and university experience in first-year university students: A self-determination theory perspective. *Journal of Hospitality, Leisure, Sport & Tourism Education*, 13:244–254.

Houben, G.-J. and Paradaens, J. (1989). A graphical interface formalism: specifying nested relational databases. In *Proceedings of the IFIP TC2/WG 2.6 Working Conferenc on Visual Database Systems*, pages 257–276, Tokyo, Japan.

Hristidis, V., Gravano, L., and Papakonstantinou, Y. (2003). Efficient IR-style keyword search over relational databases. In *Proceedings of the 29th VLDB Conference*, pages 850–861, Berlin, Heidelberg.

Hristidis, V., Hwang, H., and Papakonstantinou, Y. (2008). Authority-based keyword search in databases. *ACM Transactions on Database Systems*, 33(1):1–40.

Hristidis, V. and Papakonstantinou, Y. (2002). Discover: keyword search in relational databases. In *VLDB '02 Proceedings of the 28th international conference on Very Large Data Bases*, pages 670–681. VLDB Endowment.

Hudson, S. E. (1999). CUP User's Manual. http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html. Accessed: 30 May 2013.

Hulgeri, A., Bhalotia, G., Nakhe, C., Chakrabarti, S., and Sudarshan, S. (2001). Keyword Search in Databases. *IEEE Data Engineering Bulletin*, 24(3):22–31.

IBM (2015). Net Search Extender key concepts. http://www-01.ibm.com/support/knowledgecenter/SSEPGG_10.5.0/com.ibm.db2.luw.admin.nse.topics.doc/doc/c0052076.html?lang=en. Accessed: 12 Oct 2015.

IDOM, G. (2002). Glasgow IDOM - IR linguistic utilities. http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words. Accessed: 14 Jan 2016.

IMDb (2013). IMDb - Movies, TV and Celebrities. http://www.imdb.com/. Accessed: 11 Apr 2013.

International Organization for Standardization (1987). ISO 9075:1987 - Information processing systems – Database language – SQL. http://www.iso.org/iso/iso_ catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=16661. Accessed: 13 Jan 2016.

International Organization for Standardization (2011). ISO/IEC 9075-1:2011 - Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework). http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=53681. Accessed: 13 Jan 2016.

Jagadish, H. V., Chapman, A., Elkiss, A., Jayapandian, M., Li, Y., Nandi, A., and Yu, C. (2007). Making database systems usable. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 13–24, Beijing, China.

Jarke, M. and Vassiliou, Y. (1985). A framework for choosing a database query language. *ACM Computing Surveys*, 17(3):313–340.

Jarník, V. (1930). O jistém problému minimálním. *Práca Moravské Prírodovedecké Spolecnosti*, 6:57–63.

Joomla! (2015). J3.x:Installing Joomla - Joomla! Documentation:. https://docs.joomla.org/J3.x:Installing_Joomla#Hosting_Requirements. Accessed: 08 Jul 2015.

Justice, L. (2000). Uses of animated imagery within the software learning environment. *Digital Creativity*, 11(1):35–42.

Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., and Karambelkar, H. (2005). Bidirectional expansion for keyword search on graph databases. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 505–516.

Kantar Worldpanel (2015). Smartphone OS sales market share. http://www.kantarworldpanel.com/global/smartphone-os-market-share/. Accessed: 26 Jun 2015.

Kearns, R., Shead, S., and Fekete, A. (1996). A teaching system for SQL. In *Proceedings of the second Australasian conference on Computer science education - ACSE '97*, pages 224–231, New York, New York, USA. ACM Press.

Khine, P. T. T., Win, H. P. P., and Tun, K. N. N. (2011). Keyword searching and browsing system over relational databases. In *2011 Sixth International Conference on Digital Information Management*, pages 121–126. IEEE.

Kimelfeld, B. and Sagiv, Y. (2006). Finding and approximating top-k answers in keyword proximity search. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '06*, pages 173–182, New York, New York, USA. ACM Press.

King, B. and Melville, S. (1984). Ski: A Semantics-Knowledgeable Interface. In *Proceedings of the 10th Conference on Very Large Data Bases*, pages 30–33, Singapore.

Kuntz, M. (1993). Introduction to GIUKU: Graphical interactive intelligent utilities for knowledgeable users of data base systems. *Information Systems*, 18(4):249–268.

Lancaster University (2015). SCC.130: Information Systems. http://www.lusi.lancaster.ac.uk/CoursesHandbook/ModuleDetails/ModuleDetail?yearId=000114&courseId=016556. Accessed: 29 Sep 2015.

Larson, J. A. and Wallick, J. B. (1984). An interface for novice and infrequent database management system users. In *Proceedings of the July 9-12, 1984, national computer conference and exposition*, pages 523–529, New York, New York, USA. ACM Press.

Lemer, C., Antezana, E., Couche, F., Fays, F., Santolaria, X., Janky, R., Deville, Y., Richelle, J., and Wodak, S. J. (2004). The aMAZE LightBench: a web interface to a relational database of cellular processes. *Nucleic acids research*, 32(Database issue):D443–D448.

Li, G., Feng, J., and Wang, J. (2009). Structure-aware indexing for keyword search in databases. In *Proceeding of the 18th ACM conference on Information and knowledge management - CIKM '09*, pages 1453–1456, New York, New York, USA. ACM Press.

Li, G., Ji, S., Li, C., and Feng, J. (2011). Efficient fuzzy full-text type-ahead search. *The VLDB Journal*, 20(4):617–640.

Li, G., Ooi, B. C., Feng, J., Wang, J., and Zhou, L. (2008). EASE: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured Data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, pages 903–914, New York, New York, USA. ACM Press.

Liu, F., Yu, C., Meng, W., and Chowdhury, A. (2006). Effective keyword search in relational databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06*, pages 563–574, New York, New York, USA. ACM Press.

Luo, Y. (2009). *SPARK: A Keyword Search System on Relational Databases.* PhD thesis, The University of New South Wales.

Luo, Y., Lin, X., Wang, W., and Zhou, X. (2007). SPARK: Top-k Keyword Query in Relational Databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data - SIGMOD '07*, pages 115–126, New York, New York, USA. ACM Press.

Maier, D., Ullman, J. D., and Vardi, M. Y. (1984). On the foundations of the universal relation model. *ACM Transactions on Database Systems*, 9(2):283–308.

Malan, D. J. and Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin*, 39(1):223–227.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *An Introduction to Information Retreival.* Cambridge University Press.

Markowetz, A., Yang, Y., and Papadias, D. (2007). Keyword search on relational data streams. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data - SIGMOD '07*, pages 605–616, New York, New York, USA. ACM Press.

Martin, J. (1973). *Design of man-computer dialogues.* Prentice-Hall, Englewood Cliffs. New Jersey.

Massari, A. and Chrysanthis, P. (1995). Visual query of completely encapsulated objects. In *Proceedings RIDE-DOM'95. Fifth International Workshop on Research Issues in Data Engineering-Distributed Object Management*, pages 18–25. IEEE Comput. Soc. Press.

Massari, A., Pavani, S., and Saladini, L. (1994). QBI: An Iconic Query System for Inexpert Users. In *Proceedings of the workshop on Advanced visual interfaces - AVI '94*, pages 240–242, New York, New York, USA. ACM Press.

Matos, V. and Grasser, R. (2002). Teaching Tip A Simpler (and Better) SQL Approach to Relational Division. *Journal of Information Systems Education*, 13(2):85–88.

Matos, V., Grasser, R., and Jalics, P. (2006). The case of the missing tuple: teaching the SQL outer-join operator to undergraduate information systems students. *Journal of Computing Sciences in Colleges*, 22(1):23–32.

Mayes, J. and Fowler, C. (1999). Learning technology and usability: a framework for understanding courseware. *Interacting with Computers*, 11(5):485–497.

McDonald, N. H. and Stonebraker, M. (1975). CUPID-The Friendly Query Language. In *ACM Pacific*, pages 127–131, San Francisco, California.

McMillan, J. H. (1996). *Classroom Assessment. Principles and Practices for Effective Instruction.* Allyn & Bacon.

Mesquita, F., da Silva, A. S., de Moura, E. S., Calado, P., and Laender, A. H. (2007). LABRADOR: Efficiently publishing relational databases on the web by using keyword-based query interfaces. *Information Processing & Management*, 43(4):983–1004.

Michard, a. (1982). A new database query language for non-professional users: design principles and ergonomic evaluation. *Behavioral and Information Technology*, 1(3):279–288.

Microsoft (2014). Maximum Capacity Specifications for SQL Server. https://msdn.microsoft.com/en-us/library/ms143432.aspx. Accessed: 04 Sep 2015.

Mitrovic, A. (1998a). A Knowledge-Based Teaching System for SQL. In *ED-MEDIA 98*, pages 1027–1032.

Mitrovic, A. (1998b). Learning SQL with a computerized tutor. *ACM SIGCSE Bulletin*, 30(1):307–311.

Mitrovic, A. and Suraweera, P. (2000). Evaluating an animated pedagogical agent. In *Intelligent Tutoring Systems, Proceedings*, pages 73 – 82, Montreal, Canada.

Murray, M. and Guimaraes, M. (2008). Animated database courseware: using animations to extend conceptual understanding of database concepts. *Journal of Computing Sciences in Colleges*, 24(2):144–150.

MySQL (2012). MySQL :: MySQL 5.5 Reference Manual :: 12.9.1 Natural Language Full-Text Searches:. https://dev.mysql.com/doc/refman/5.5/en/fulltext-natural-language.html. Accessed: 14 Jan 2016.

MySQL (2015). MySQL :: MySQL 5.7 Reference Manual :: 8.2.1.15 ORDER BY Optimization:. http://dev.mysql.com/doc/refman/5.7/en/order-by-optimization.html. Accessed: 31 Jul 2015.

National Curriculum (2015). National curriculum in England: computing programmes of study. https://www.gov.uk/government/publications/

national-curriculum-in-england-computing-programmes-of-study. Accessed: 01 May 2015.

Nielsen, J. (1997). Search and You May Find. http://www.nngroup.com/articles/search-and-you-may-find/. Accessed: 13 Jan 2016.

Oracle (2002). Oracle8i interMedia Text Reference. http://docs.oracle.com/cd/A87860_01/doc/inter.817/a77063/cdefaul5.htm. Accessed: 15 Oct 2015.

Oracle (2015a). Database SQL Language Reference - Using Subqueries. http://docs.oracle.com/cd/B28359_01/server.111/b28286/queries007.htm. Accessed: 04 Sep 2015.

Oracle (2015b). Oracle Text. http://www.oracle.com/technetwork/testcontent/index-098492.html. Accessed: 27 Jul 2015.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The PageRank citation ranking: bringing order to the web. Technical report, Stanford University InfoLab.

Park, J. and Lee, S.-g. (2010). Keyword search in relational databases. *Knowledge and Information Systems*, 26(2):175–193.

Patil, R. and Chen, Z. (2012). STRUCT: Incorporating Contextual Information for English Query Search on Relational Databases. In *Proceedings of the Third International Workshop on Keyword Search on Structured Data - KEYS '12*, pages 11–22, Scottsdale, Arizona, USA. ACM Press.

Pietriga, E., Vion-Dury, J.-Y., and Quint, V. (2001). VXT: a visual approach to XML transformations. In *Proceedings of the 2001 ACM Symposium on Document engineering*, pages 1–10, Atlanta, Georgia, USA. ACM.

Prim, R. C. (1957). Shortest Connection Networks And Some Generalizations. *Bell System Technical Journal*, 36(6):1389–1401.

Prior, J. C. (2003). Online assessment of SQL query formulation skills. In *Proceedings of the fifth Australasian conference on Computing education-Volume 20*, pages 247–256. Australian Computer Society, Inc.

Qin, L., Yu, J. X., and Chang, L. (2010). Scalable keyword search on large data streams. *The VLDB Journal*, 20(1):35–57.

Qin, L., Yu, J. X., and Chang, L. (2012). Computing Structural Statistics by Keywords in Databases. *IEEE Transactions on Knowledge and Data Engineering*, 24(10):1731–1746.

Renaud, K. and van Biljon, J. (2004). Teaching SQL - Which Pedagogical Horse for This Course? In Williams, H. and MacKinnon, L., editors, *Key Technologies for Data Management*, Lecture Notes in Computer Science, pages 244–256. Springer Berlin Heidelberg, Berlin, Heidelberg.

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., and Silver, J. (2009). Scratch: Programming for all. *Communications of the ACM*, 52(11):60–67.

Rieber, L. P. (1988). Animation in Computer-Based Instruction. *Educational Technology Research and Development*, 38(1):77–86.

Robinson, I., Webber, J., and Eifrem, E. (2013). *Graph Databases.* O'Reilly Media, 1 edition.

Russell, G. and Cumming, A. (2004). Improving the student learning experience for SQL using automatic marking. In *Cognition and Exploratory Learning in Digital Age*, pages 281–288.

Sadiq, S., Orlowska, M., Sadiq, W., and Lin, J. (2004). SQLator: an online SQL learning workbench. *ACM SIGCSE Bulletin*, 36(3):223–227.

Sagiv, Y. (2013). A personal perspective on keyword search over data graphs. In *Proceedings of the 16th International Conference on Database Theory - ICDT '13*, pages 21–32, New York, New York, USA. ACM Press.

Schroder, H. M., Driver, M. J., and Streufert, S. (1967). *Human Information Processing*. New York: Holt, Rinehart & Winston.

Shneiderman, B. (1978). Improving the human factors aspect of database interactions. *ACM Transactions on Database Systems*, 3(4):417–439.

Shu, N. C. (1985). Formal: a Forms-Oriented, Visual-Directed Application Development System. *Computer*, 18(8):38–49.

Shute, J., Ellner, S., Cieslewicz, J., Rae, I., Stancescu, T., Apte, H., Vingralek, R., Samwel, B., Handy, B., Whipkey, C., Rollins, E., Oancea, M., Littlefield, K., and Menestrina, D. (2013). F1: A Distributed SQL Database That Scales. *Proceedings of the VLDB Endowment*, 6(11):1068–1079.

Song, Y. (2015). *An Authoring and Presentation Environment for Interactive Worked Examples*. PhD thesis, University of Glasgow.

SQLite (2015a). Appropriate Uses For SQLite. https://www.sqlite.org/whentouse.html. Accessed: 26 Jun 2015.

SQLite (2015b). Implementation limits for SQLite. http://www.sqlite.org/limits.html#max_expr_depth. Accessed: 04 Sep 2015.

SQLite (2015c). SQL Features That SQLite Does Not Implement. https://www.sqlite.org/omitted.html. Accessed: 07 Sep 2015.

Stack Overflow (2015). Stack Overflow Developer Survey 2015. http://stackoverflow.com/research/developer-survey-2015#tech-lang. Accessed: 13 Apr 2015.

Stanford University (2015). CS145 Introduction to Databases (Autumn 2015). http://web.stanford.edu/class/cs145/. Accessed: 29 Sep 2015.

The Radacti Group (2010). The Radacti Group, Inc. http://www.radicati.com/?p=5282. Accessed: 01 Jul 2015.

Tsuda, K., Hirakawa, M., Tanaka, M., and Ichikawa, T. (1989). IconicBrowser: an iconic retrieval system for object-oriented databases. *Proceedings 1989 IEEE Workshop on Visual Languages*, pages 59–76.

University College London (2015). COMP3013 - Database and Information Management Systems. http://www.cs.ucl.ac.uk/students/syllabus/undergrad/3013_database_and_information_management_systems/. Accessed: 29 Sep 2015.

University of Cambridge (2015). Computer Laboratory – Course pages 2015–16: Databases. https://www.cl.cam.ac.uk/teaching/1516/Databases/. Accessed: 29 Sep 2015.

University of Oxford (2015). Databases. http://www.cs.ox.ac.uk/teaching/courses/2015-2016/databases/. Accessed: 29 Sep 2015.

User Interface Engineering (2004). Making Tips Work. http://world.std.com/~uieweb/tips.htm. Accessed: 23 Sep 2015.

Vadaparty, K., Aslandogan, Y. A., and Ozsoyoglu, G. (1993). Towards a unified visual database access. *ACM Sigmod Record*, 22(2):357–366.

van Gog, T., Kester, L., and Paas, F. (2011). Effects of worked examples, example-problem, and problem-example pairs on novices' learning. *Contemporary Educational Psychology*, 36(3):212–218.

Wang, S., Peng, Z., Zhang, J., Qin, L., Wang, S., Yu, J. X., and Ding, B. (2006). NUITS: a novel user interface for efficient keyword search over databases. In *Proceedings of the 32nd international conference on Very large data bases*, pages 1143–1146, Seoul, Korea. VLDB Endowment.

Wheeldon, R., Levene, M., and Keenoy, K. (2004). DbSurfer: A Search and Navigation Tool for Relational Databases. In Williams, H. and MacKinnon, L., editors, *Key Technologies for Data Management*, Lecture Notes in Computer Science, pages 144–149. Springer Berlin Heidelberg, Berlin, Heidelberg.

Whitworth, B. (2005). Polite computing. *Behaviour & Information Technology*, 24(5):353–363.

Widom, J. (2005). Indexing Relational Database Content Offline for Efficient Keyword-Based Search. In *9th International Database Engineering & Application Symposium (IDEAS'05)*, pages 297–306. IEEE.

Wong, H. K. T. and Kuo, I. (1982). GUIDE : Graphical User Interface for Database Exploration. In *VLDB*, pages 22–32, Mexico City, Mexico.

WordPress (2015). Database Description - WordPress Codex. https://codex.wordpress.org/Database_Description. Accessed: 08 Jul 2015.

Xiao, J., Catrambone, R., and Stasko, J. (2003). Be Quiet? Evaluating Proactive and Reactive User Interface Assistants. In *Proc. INTERACT*, pages 383–390, Zurich, Switzerland.

Xie, D., Luo, J.-L., and Zhu, Y. (2012). A Keyword Retrieval Semantics over Relational Databases. *Physics Procedia*, 25:1863–1870.

Xu, Y., Guan, J., and Ishikawa, Y. (2012). Scalable Top-k Keyword Search in Relational Databases. *Database Systems for Advanced Applications*, 1:65–80.

Young, D. and Shneiderman, B. (1993). A graphical filter/flow representation of Boolean queries: A prototype implementation and evaluation. *Journal of the American Society for Information Science*, 44(6):327–339.

Zeng, Z., Bao, Z., Ling, T. W., and Lee, M. L. (2012). iSearch: An Interpretation based Framework for Keyword Search in Relational Databases. In *Proceedings of the Third International Workshop on Keyword Search on Structured Data*, pages 3–10, Scottsdale, Arizona, USA. ACM Press.

Zhou, B. and Pei, J. (2011). Aggregate keyword search on large relational databases. *Knowledge and Information Systems*, 30(2):283–318.

Zloof, M. M. (1975). Query by example. In *Proceedings of the May 19-22, 1975, national computer conference and exposition on - AFIPS '75*, pages 431–438, New York, New York, USA. ACM Press.

Appendix A

Movie database

The following outlines a sample database which is referred to throughout this thesis. This movie database is used to provide context in a number of examples and also in the evaluation of the CAFTAN system (Chapter 5).

A.1 The Movie Database (TMDb)

TMDb is a website similar in functionality to the more well known IMDb; it allows users to contribute information and provides a set of APIs to extract its contents (Apiary, 2015). These APIs were used to obtain information from TMDb and populate a relational database representing a subset of the data.

A.1.1 API responses

The contents of the TMDb database can be accessed using a REST API that returns JSON strings containing the relevant information. HTTP GET parameters are used to pass any data required with a request such as IDs and an access token (required for all API calls). To populate the database used in this work three API calls were used:

1. Production companies - providing a list of movies produced by a given company

2. Movies - providing details of a given movie, including genres

3. Credits - providing a list of actors and the characters they portray along with crew members and their jobs for the given movie

The following shows sample API calls for each of the above, for brevity the JSON responses are not given in their entirity.

A.1.1.1 Production companies

The following call returns a list of the movies produced by *"Warner Bros"*, with the ID 6194. The ID of the production companies can be found manually on the TMDb website: http://api.themoviedb.org/3/company/6194/movies

```
{
  "id" : 6194,
  "page" : 1,
  "results" : [{
      "adult" : false,
      "backdrop_path" : "/7u3pxc0K1wx32IleAkLv78MKgrw.jpg",
      "genre_ids" : [12, 28, 53, 878],
      "id" : 603,
      "original_language" : "en",
      "original_title" : "The Matrix",
      "overview" : "Thomas A. Anderson is a man living two lives. By day he is an average computer programmer
             and by night a malevolent hacker known as Neo, who finds himself targeted by the police when he
            is contacted by Morpheus, a legendary computer hacker, who reveals the shocking truth about our
            reality.",
      "release_date" : "1999-03-30",
      "poster_path" : "/gynBNzwyaHKtXqlEKKLioNkjKgN.jpg",
      "popularity" : 5.088707,
      "title" : "The Matrix",
      "video" : false,
      "vote_average" : 7.6,
      "vote_count" : 4507
    }, {
      "adult" : false,
      "backdrop_path" : "/65JWXDCAfwHhJKnDwRnEgVB411X.jpg",
      "genre_ids" : [28, 80, 18],
      "id" : 272,
      "original_language" : "en",
      "original_title" : "Batman Begins",
      "overview" : "Driven by tragedy, billionaire Bruce Wayne dedicates his life to uncovering and defeating
             the corruption that plagues his home, Gotham City.  Unable to work within the system, he instead
            creates a new identity, a symbol of fear for the criminal underworld - The Batman.",
      "release_date" : "2005-06-15",
      "poster_path" : "/xios0eLfzPbfLfqui41kSWn0OsZ.jpg",
      "popularity" : 4.573057,
      "title" : "Batman Begins",
      "video" : false,
      "vote_average" : 7.2,
      "vote_count" : 3428
    }
  ],
  "total_pages" : 61,
  "total_results" : 1220
}
```

A.1.1.2 Movies

Once a list of movies, and their associated IDs, is established these IDs can be used to find further information regarding a specific movie. The following finds the information about the movie “The Matrix” (ID 603 - found in the production companies API call): http://api.themoviedb.org/3/movie/603

```
{
  "adult" : false,
  "backdrop_path" : "/7u3pxc0K1wx32IleAkLv78MKgrw.jpg",
  "belongs_to_collection" : {
    "id" : 2344,
    "name" : "The Matrix Collection",
    "poster_path" : "/lh4aGpd3U9rm9B8Oqr6CUgQLtZL.jpg",
    "backdrop_path" : "/bRm2DEgUiYciDw3myHuYFInD7la.jpg"
  },
  "budget" : 63000000,
  "genres" : [{
      "id" : 12,
      "name" : "Adventure"
    }, {
      "id" : 28,
      "name" : "Action"
    }, {
      "id" : 53,
      "name" : "Thriller"
    }, {
      "id" : 878,
      "name" : "Science Fiction"
    }
  ],
  "homepage" : "http://www.warnerbros.com/movies/home-entertainment/the-matrix/37313ac7-9229-474d-a423-
      44b7a6bc1a54.html",
  "id" : 603,
  "imdb_id" : "tt0133093",
  "original_language" : "en",
  "original_title" : "The Matrix",
  "overview" : "Thomas A. Anderson is a man living two lives. By day he is an average computer programmer and
       by night a malevolent hacker known as Neo, who finds himself targeted by the police when he is
      contacted by Morpheus, a legendary computer hacker, who reveals the shocking truth about our reality."
      ,
  "popularity" : 3.617395,
  "poster_path" : "/gynBNzwyaHKtXqlEKKLioNkjKgN.jpg",
  "production_companies" : [{
      "name" : "Village Roadshow Pictures",
      "id" : 79
    }, {
      "name" : "Groucho II Film Partnership",
      "id" : 372
    }, {
      "name" : "Silver Pictures",
      "id" : 1885
    }, {
      "name" : "Warner Bros.",
      "id" : 6194
    }
  ],
  "production_countries" : [{
      "iso_3166_1" : "AU",
      "name" : "Australia"
    }, {
      "iso_3166_1" : "US",
      "name" : "United States of America"
    }
  ],
  "release_date" : "1999-03-30",
  "revenue" : 463517383,
  "runtime" : 136,
  "spoken_languages" : [{
      "iso_639_1" : "en",
      "name" : "English"
    }
  ],
  "status" : "Released",
  "tagline" : "Welcome to the Real World.",
  "title" : "The Matrix",
  "video" : false,
  "vote_average" : 7.6,
  "vote_count" : 4508
}
```

A.1.1.3 Credits

The ID of a movie can also be used to find the credits (cast and crew) for a movie. The following API call produces the credits for the movie *"The Matrix"* (ID 603): http://api.themoviedb.org/3/movie/603/credits

```
{
  "id" : 603,
  "cast" : [{
      "cast_id" : 34,
      "character" : "Neo",
      "credit_id" : "52fe425bc3a36847f80181c1",
      "id" : 6384,
      "name" : "Keanu Reeves",
      "order" : 0,
      "profile_path" : "/id1qIb7cZs2eQno90KsKwG8VLGN.jpg"
    }, {
      "cast_id" : 21,
      "character" : "Morpheus",
      "credit_id" : "52fe425bc3a36847f801818d",
      "id" : 2975,
      "name" : "Laurence Fishburne",
      "order" : 1,
      "profile_path" : "/mh0lZ1XsT84FayMNiT6Erh91mVu.jpg"
    }, {
      "cast_id" : 22,
      "character" : "Trinity",
      "credit_id" : "52fe425bc3a36847f8018191",
      "id" : 530,
      "name" : "Carrie-Anne Moss",
      "order" : 2,
      "profile_path" : "/8iATAc5z5XOKFFARLsvaawa8MTY.jpg"
    }, {
      "cast_id" : 23,
      "character" : "Agent Smith",
      "credit_id" : "52fe425bc3a36847f8018195",
      "id" : 1331,
      "name" : "Hugo Weaving",
      "order" : 3,
      "profile_path" : "/3DKJSeTucd7krnxXkwcir6PgT88.jpg"
    }
  ],
  "crew" : [{
      "credit_id" : "52fe425bc3a36847f8018117",
      "department" : "Directing",
      "id" : 9339,
      "job" : "Director",
      "name" : "Andy Wachowski",
      "profile_path" : "/nh5SBuv9cm1FByTc7dlV0zy03GO.jpg"
    }, {
      "credit_id" : "52fe425bc3a36847f801811d",
      "department" : "Directing",
      "id" : 9340,
      "job" : "Director",
      "name" : "Lana Wachowski",
      "profile_path" : "/fxZ7SpJCZ9DgJERWEpGmn5a4mdp.jpg"
    }
  ]
}
```

A.2 Structuring the data

The API requests submitted to TMDb, and their responses, offer little insight as to the structure TMDb uses to store the data. As a result it is necessary to design a database schema suitable for storing the subset of the data used in this work. The standard practice of normalisation was carried out in order to eliminate duplicate entries from the database. The result of this process is a database consisting of eight relations; four of them contain meaningful information relating to genres, production companies, people and movies while the remaining four act primarily as junction tables with *acts_in* and *crew_on* including the role associated with the relationship. Figure A.1 shows the structure of the database.

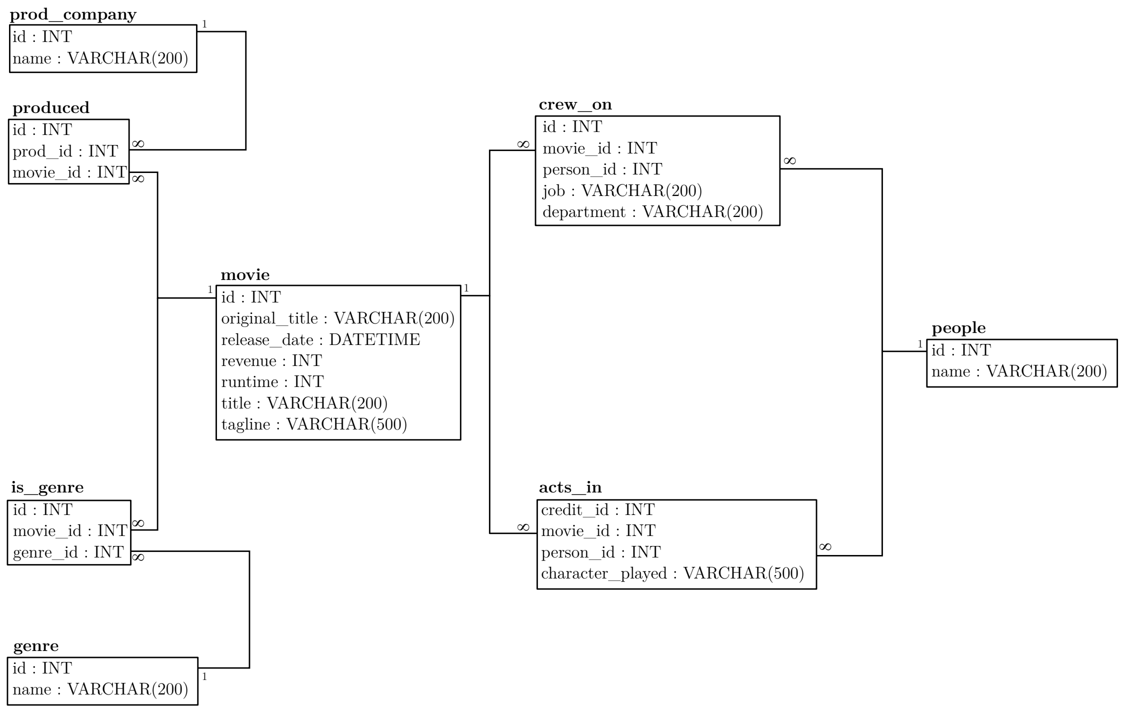


FIGURE A.1: The movie database schema

A.3 The relational movie database

To populate the database three API calls (Section A.1.1) were used to find the information relating to movies produced by a selection of the highest grossing production studios[1]. A small application was written to automate the process, extracting all films produced by the given studios along with their associated genres and credits and populating a database with the results. A relational data structure was designed and implemented for this purpose, in practice the information was stoed in a MySQL database.

By using the TMDb APIs and the schema described here a database containing 5735 movies and their associated actors, crew, genres and production companies was produced. Generating the database from the highest grossing film studios increases the likelihood that its contents would be familiar to many users.

[1]http://www.the-numbers.com

Appendix B

CAFTAN

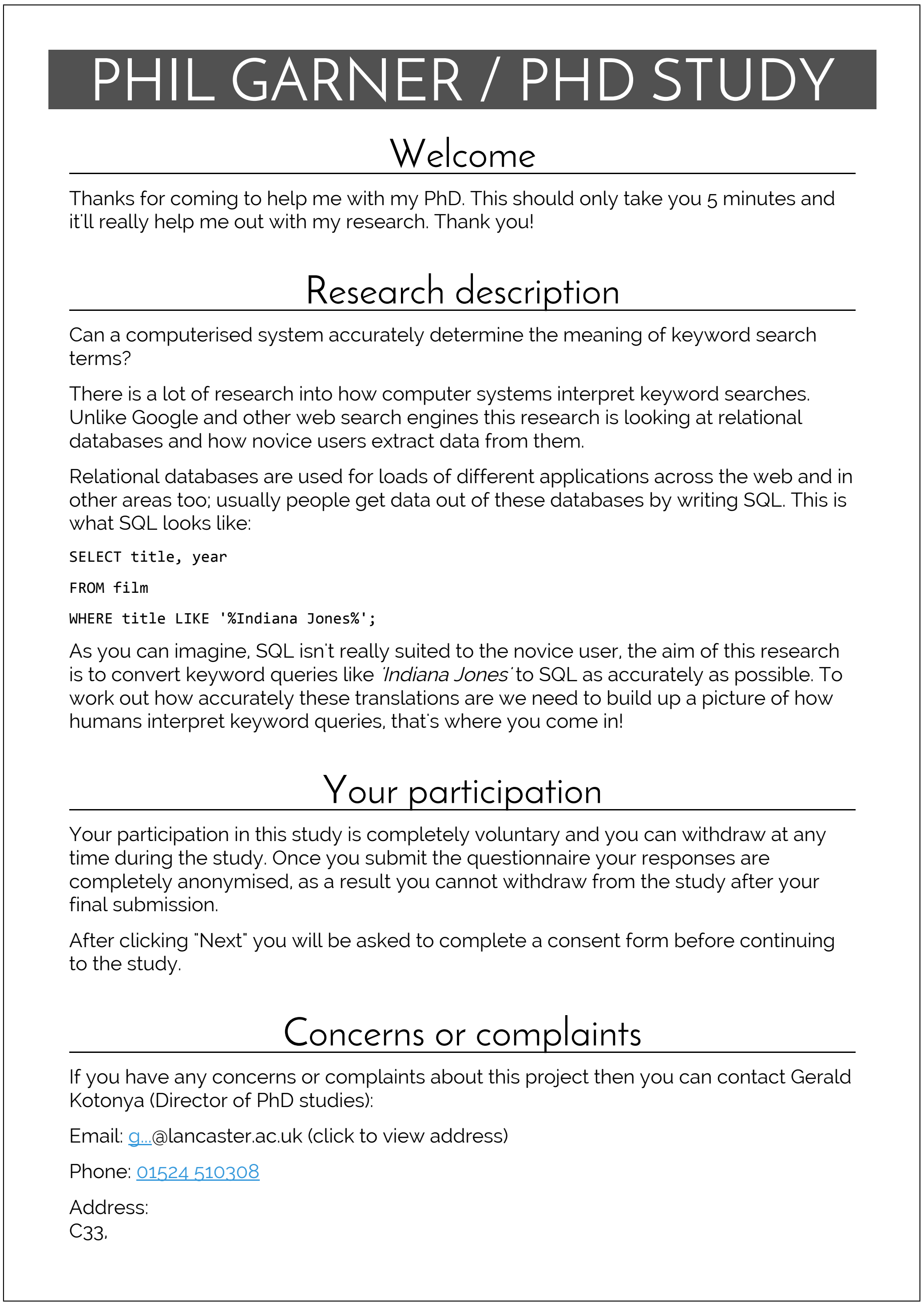
This appendix includes information relevant to the CAFTAN prototype described and evaluated in Chapters 5 and 7 respectively. It includes sample customisation descriptions, forms and questionnaires used in the evaluation.

B.1 CAFTAN study forms

In Chapter 7 the evaluation for CAFTAN is described, the following forms were presented to users, online, prior to their commencement in the study.

B.1.1 Research description

The aim of this document was to make clear the aims of the study as well as the methodology used to achieve them. The document also included contact details of both the researcher and an independent third party should any complaints arise.



PHIL GARNER / PHD STUDY

Welcome

Thanks for coming to help me with my PhD. This should only take you 5 minutes and it'll really help me out with my research. Thank you!

Research description

Can a computerised system accurately determine the meaning of keyword search terms?

There is a lot of research into how computer systems interpret keyword searches. Unlike Google and other web search engines this research is looking at relational databases and how novice users extract data from them.

Relational databases are used for loads of different applications across the web and in other areas too; usually people get data out of these databases by writing SQL. This is what SQL looks like:

```
SELECT title, year
FROM film
WHERE title LIKE '%Indiana Jones%';
```

As you can imagine, SQL isn't really suited to the novice user, the aim of this research is to convert keyword queries like *'Indiana Jones'* to SQL as accurately as possible. To work out how accurately these translations are we need to build up a picture of how humans interpret keyword queries, that's where you come in!

Your participation

Your participation in this study is completely voluntary and you can withdraw at any time during the study. Once you submit the questionnaire your responses are completely anonymised, as a result you cannot withdraw from the study after your final submission.

After clicking "Next" you will be asked to complete a consent form before continuing to the study.

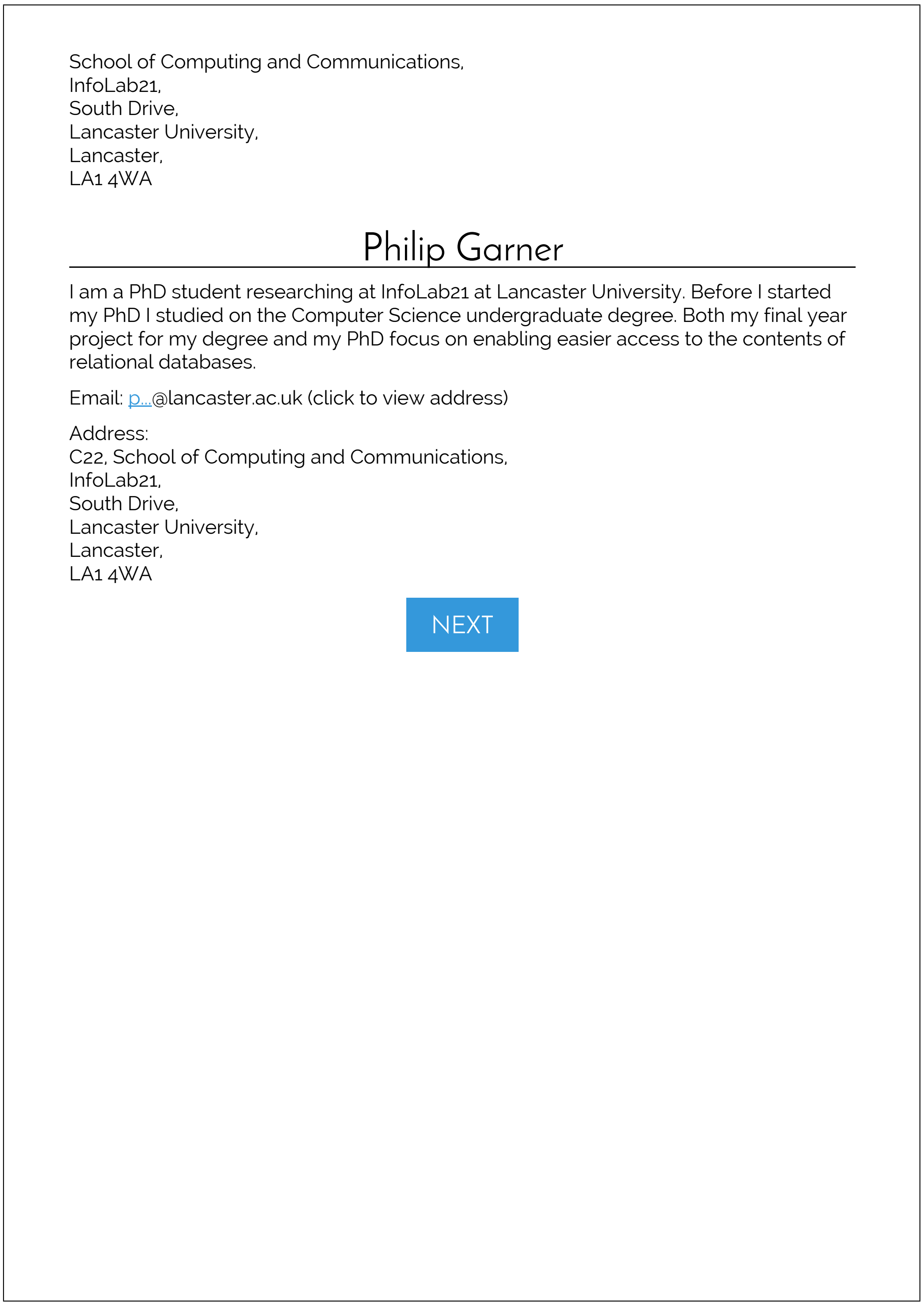
Concerns or complaints

If you have any concerns or complaints about this project then you can contact Gerald Kotonya (Director of PhD studies):

Email: g...@lancaster.ac.uk (click to view address)

Phone: 01524 510308

Address:
C33,



School of Computing and Communications,
InfoLab21,
South Drive,
Lancaster University,
Lancaster,
LA1 4WA

Philip Garner

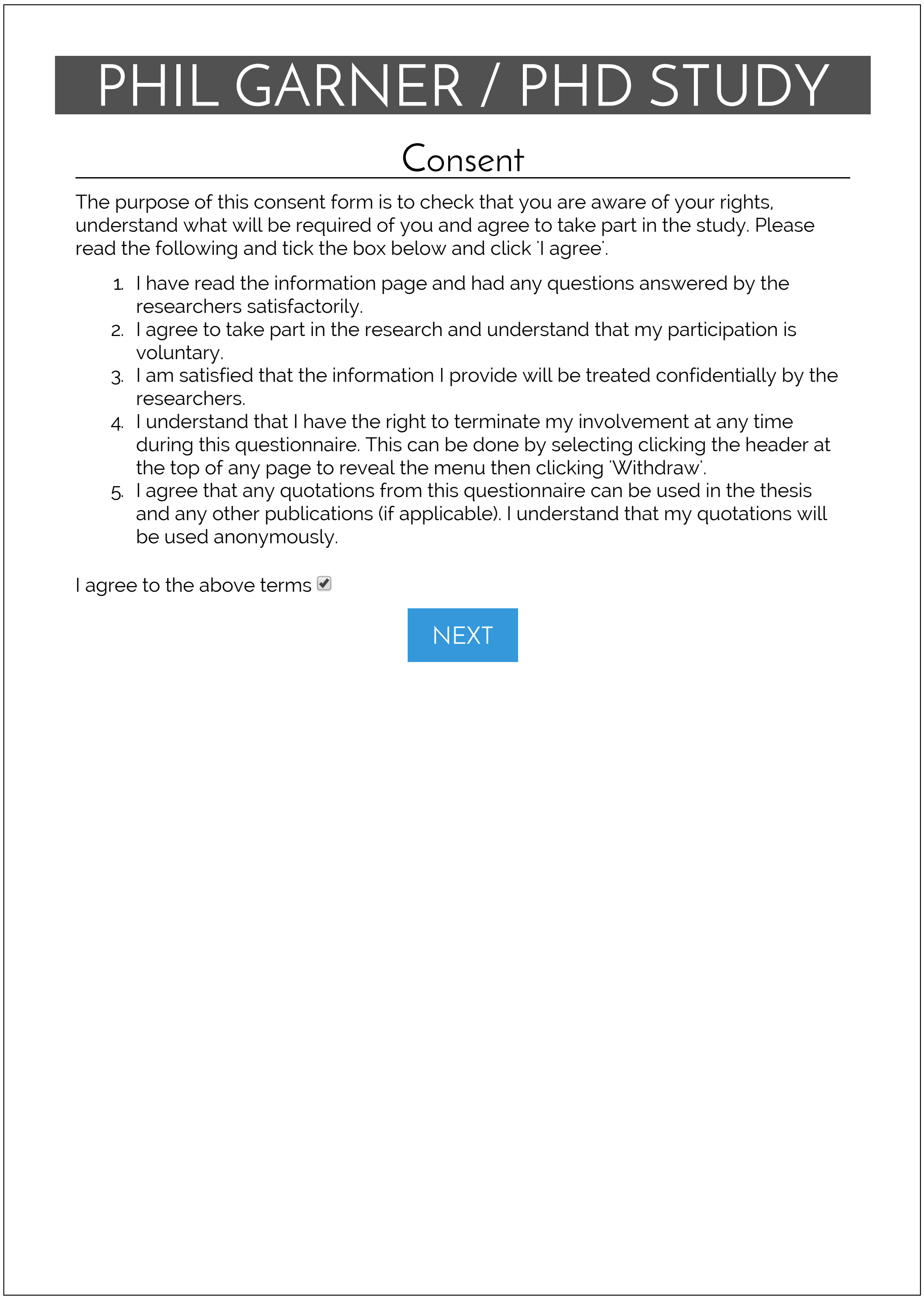
I am a PhD student researching at InfoLab21 at Lancaster University. Before I started my PhD I studied on the Computer Science undergraduate degree. Both my final year project for my degree and my PhD focus on enabling easier access to the contents of relational databases.

Email: p...@lancaster.ac.uk (click to view address)

Address:
C22, School of Computing and Communications,
InfoLab21,
South Drive,
Lancaster University,
Lancaster,
LA1 4WA

B.1.2 Consent form

The following consent form was used to clarify that the participants understood the details of the study before it commenced.



PHIL GARNER / PHD STUDY

Consent

The purpose of this consent form is to check that you are aware of your rights, understand what will be required of you and agree to take part in the study. Please read the following and tick the box below and click 'I agree'.

1. I have read the information page and had any questions answered by the researchers satisfactorily.
2. I agree to take part in the research and understand that my participation is voluntary.
3. I am satisfied that the information I provide will be treated confidentially by the researchers.
4. I understand that I have the right to terminate my involvement at any time during this questionnaire. This can be done by selecting clicking the header at the top of any page to reveal the menu then clicking 'Withdraw'.
5. I agree that any quotations from this questionnaire can be used in the thesis and any other publications (if applicable). I understand that my quotations will be used anonymously.

I agree to the above terms ☑

B.1.3 Evaluation questions

The following 100 keyword searches were used in the evaluation of CAFTAN. They are shown here in alphabetical order but were used at random in the accuracy study. In the performance evaluation of CAFTAN (Section 7.1.5) each query was executed 100 times on both the generic and customised implementations of CAFTAN; the average execution times are shown (in milliseconds) for each CAFTAN implementation respectively.

1. 08/07/2011 *(59, 37)*
2. 100-150 *(371, 380)*
3. 100-180 romance comedy *(453, 442)*
4. 120-180 Drama *(329, 291)*
5. 120-180 Drama Thriller *(357, 315)*
6. 14/10/1999 *(68, 40)*
7. 180-240 *(235, 209)*
8. 180-240 Thriller *(241, 218)*
9. 1999 Thriller *(147, 114)*
10. 2001 *(135, 111)*
11. 2001 Gary Oldman *(253, 234)*
12. 2002 Drama *(145, 130)*
13. 2003 Will Ferrell *(249, 245)*
14. 2005 *(230, 219)*
15. 2005 Bruce Willis *(184, 169)*
16. 2007 Adventure *(144, 117)*
17. 60-120 *(989, 979)*
18. 60-120 Family *(465, 452)*
19. 60-120 action thriller *(616, 572)*
20. Action Donald Sutherland *(374, 332)*
21. Action Eva Green *(344, 306)*
22. Action Fox 2000 Pictures *(178, 151)*
23. Action Paramount Pictures *(179, 148)*
24. Action Thriller Drama *(141, 115)*
25. Action Universal Pictures *(142, 125)*
26. Adam Scott Dimension Films *(233, 234)*
27. Adventure Buena Vista *(152, 134)*

28. Alexander *(238, 288)*
29. Animation Thriller *(317, 143)*
30. Anna Kendrick Universal Pictures *(368, 250)*
31. Anne Hathaway Legendary Pictures *(373, 276)*
32. Astro *(180, 128)*
33. Avatar *(155, 111)*
34. Cameron Diaz *(244, 211)*
35. Chris Evans Mark Ruffalo *(213, 179)*
36. Christian Bale Heath Ledger *(176, 147)*
37. Christopher Walken Amanda Plummer *(169, 154)*
38. Colin Farrell *(208, 195)*
39. Comedy John Cleese *(278, 279)*
40. Comedy Justin Timberlake *(258, 242)*
41. Comedy Sophie Evans *(260, 246)*
42. Crime Jim Broadbent *(288, 262)*
43. Crime Pruitt Taylor Vince *(308, 260)*
44. Crime Uma Thurman *(294, 249)*
45. Crime Universal Pictures *(180, 148)*
46. Crime Warner Bros *(156, 128)*
47. Dominic West *(217, 195)*
48. Domino *(126, 101)*
49. Drama Arron Shiver *(216, 196)*
50. Drama Island Pictures *(146, 127)*
51. Drama Legendary Pictures *(116, 96)*
52. Drama Phyllis Somerville *(195, 188)*
53. Drama Richard Gere *(227, 249)*
54. Emma Thompson Village Roadshow Pictures *(245, 251)*
55. Family Adam West *(274, 249)*
56. Family Comedy Romance *(145, 126)*
57. Fantasy Joel Fry *(230, 216)*
58. Fantasy Megan Fox *(246, 227)*
59. Fight Club *(138, 114)*
60. George Clooney Brad Pitt *(179, 169)*
61. Harry Potter *(130, 117)*

62. History War *(111, 93)*
63. Horror Cold Day Ltd *(102, 85)*
64. Horror Dimension Films *(104, 87)*
65. Iron Man *(111, 104)*
66. Jackie Brown *(99, 93)*
67. Jamie Parker *(159, 160)*
68. Jasmine Dustin *(165, 162)*
69. Jimmy Fallon Perdido Prod *(185, 187)*
70. Josh Gad *(188, 183)*
71. Julie Walters Harry Potter *(231, 223)*
72. Katie Holmes *(227, 203)*
73. Keanu Reeves Laurence Fishburne *(169, 144)*
74. Knocked Up 17 Again *(113, 93)*
75. Lauren Tom *(203, 191)*
76. Lincoln *(119, 95)*
77. Lord of the Rings Deep Impact *(366, 361)*
78. Lucy Liu *(293, 241)*
79. Matt Winston *(279, 237)*
80. Musical Woody Buck *(301, 254)*
81. Mystery Drama *(145, 118)*
82. Noah Ringer *(226, 210)*
83. Pat Corley *(240, 215)*
84. Robbie Coltrane *(251, 221)*
85. Romance Miramax Films *(164, 117)*
86. Romance Trent Ford *(257, 198)*
87. Ron Cook *(250, 195)*
88. Science Fiction Adventure *(156, 120)*
89. Shrek *(126, 96)*
90. Skye Dennis *(203, 173)*
91. Star Trek *(133, 94)*
92. Suspense Alec Baldwin *(230, 183)*
93. The Core Up *(410, 355)*
94. Thriller Gary Oldman *(310, 273)*
95. Thriller Joe Dixon *(318, 288)*
96. Tim Robbins DreamWorks *(302, 288)*
97. Tim Robbins Morgan Freeman *(213, 198)*
98. Tom Hardy Michael Caine *(242, 234)*
99. War Horror *(141, 125)*
100. William Hurt *(230, 250)*

B.2 CAFTAN customisations

Chapter 5 describes CAFTAN, a generic keyword search system that can be customised with respect to both the way in which it handles search terms and also the style with which the results are displayed. The following XML documents show how the customisations are described.

B.2.1 Weight strengths

The following is an example of an XML file used to customise the weights of attributes within the movie database

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<strengths>
  <!-- acts_in -->
  <strength relation="acts_in" attribute="credit_id" strength="0.2"></strength>
  <strength relation="acts_in" attribute="movie_id" strength="0.2"></strength>
  <strength relation="acts_in" attribute="person_id" strength="0.2"></strength>

  <!-- crew_on -->
  <strength relation="crew_on" attribute="credit_id" strength="0.2"></strength>
  <strength relation="crew_on" attribute="movie_id" strength="0.2"></strength>
  <strength relation="crew_on" attribute="person_id" strength="0.2"></strength>
  <strength relation="crew_on" attribute="job" strength="0.8"></strength>
  <strength relation="crew_on" attribute="department" strength="0.8"></strength>

  <!-- genre -->
  <strength relation="genre" attribute="id" strength="0.2"></strength>
  <strength relation="genre" attribute="name" strength="1.2"></strength>

  <!-- is_genre -->
  <strength relation="is_genre" attribute="id" strength="0.2"></strength>
  <strength relation="is_genre" attribute="movie_id" strength="0.2"></strength>
  <strength relation="is_genre" attribute="genre_id" strength="0.2"></strength>

  <!-- movie -->
  <strength relation="movie" attribute="id" strength="0.2"></strength>
  <strength relation="movie" attribute="original_title" strength="1.5"></strength>
  <strength relation="movie" attribute="release_date" strength="1.2"></strength>
  <strength relation="movie" attribute="revenue" strength="1.0"></strength>
  <strength relation="movie" attribute="runtime" strength="1.0"></strength>
  <strength relation="movie" attribute="title" strength="1.5"></strength>
  <strength relation="movie" attribute="tagline" strength="0.5"></strength>

  <!-- people -->
  <strength relation="people" attribute="id" strength="0.2"></strength>
  <strength relation="people" attribute="name" strength="1.2"></strength>

  <!-- prod_company -->
  <strength relation="prod_company" attribute="id" strength="0.2"></strength>
  <strength relation="prod_company" attribute="name" strength="1.0"></strength>

  <!-- produced -->
```

```xml
  <strength relation="produced" attribute="id" strength="0.2"></strength>
  <strength relation="produced" attribute="movie_id" strength="0.2"></strength>
  <strength relation="produced" attribute="prod_id" strength="0.2"></strength>

</strengths>
```

B.2.2 Visualisation

The following is a sample XML file used to produce customised results visualisations.

```xml
<query_descriptions>
  <query_description>
    <search>
      <relation name="genre"></relation>
    </search>
    <show>
      <row>
        <attribute relation="movie" attribute="title" span="3" style="italic"></attribute>
      </row>
      <row>
        <attribute relation="movie" attribute="revenue"></attribute>
        <attribute relation="movie" attribute="runtime"></attribute>
        <attribute relation="movie" attribute="release_date"></attribute>
      </row>
    </show>
    <order_by>
      <order relation="movie" attribute="title" type = "ASC"></order>
    </order_by>
  </query_description>
  <query_description>
    <search>
      <relation name="people"></relation>
    </search>
    <show>
      <row>
        <attribute relation="movie" attribute="title" span="3" style="italic"></attribute>
      </row>
      <row>
        <attribute relation="movie" attribute="revenue"></attribute>
        <attribute relation="movie" attribute="runtime"></attribute>
        <attribute relation="movie" attribute="release_date"></attribute>
      </row>
    </show>
    <order_by>
      <order relation="movie" attribute="title" type = "ASC"></order>
    </order_by>
  </query_description>
</query_descriptions>
```

B.3 Complete dataset

The raw data used in the evaluation of CAFTAN, including performance data and user interpretations, is openly available from Lancaster University data archive at http://dx.doi.org/10.17635/lancaster/researchdata/79

Appendix C

SiS

Chapter 6 describes SQL in Steps which is evaluated in Chapter 7. Throughout this evaluation a number of forms, questionnaires and assessments were used. This appendix includes all material relating to SiS and it's evaluation.

- Section C.1 includes all the forms, questionnaires and assessments used throughout the study.
- Section C.2 describes the various configurable options in SiS.
- Section C.3.2 is a full transcription of the focus group that took place following the SiS study.

C.1 SiS study forms

The following forms, questionnaires and assessments were given to students at various points in the SiS evaluation.

C.1.1 Start of study

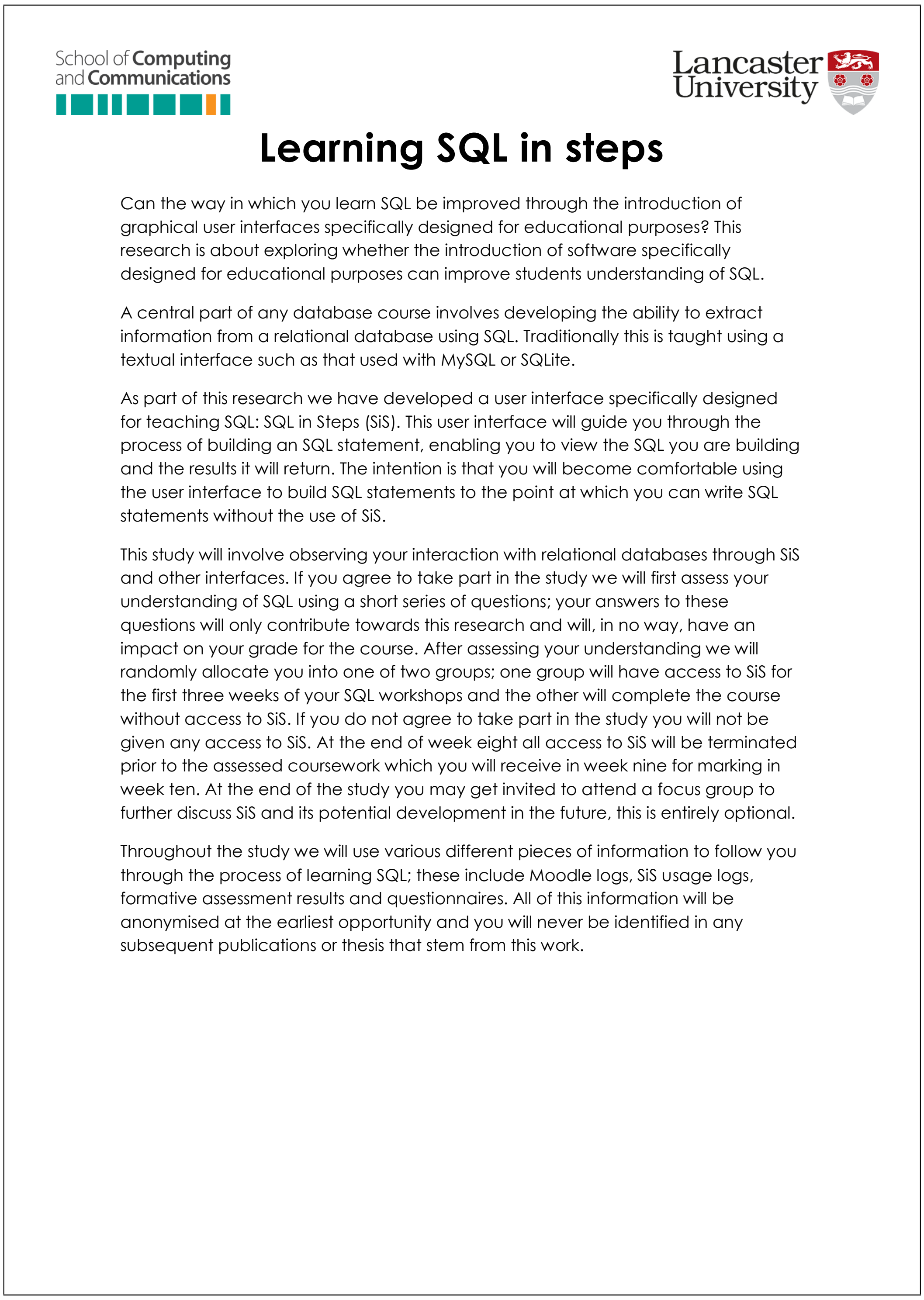
At the start of the study, participants were given three documents:

- A research description
- A participation consent form

- An initial assessment

C.1.1.1 Research description

The aim of this document was to make clear the aims of the study as well as the methodology used to achieve them. The document also included contact details of both the researcher and an independent third party should any complaints arise.



School of Computing and Communications

Lancaster University

Learning SQL in steps

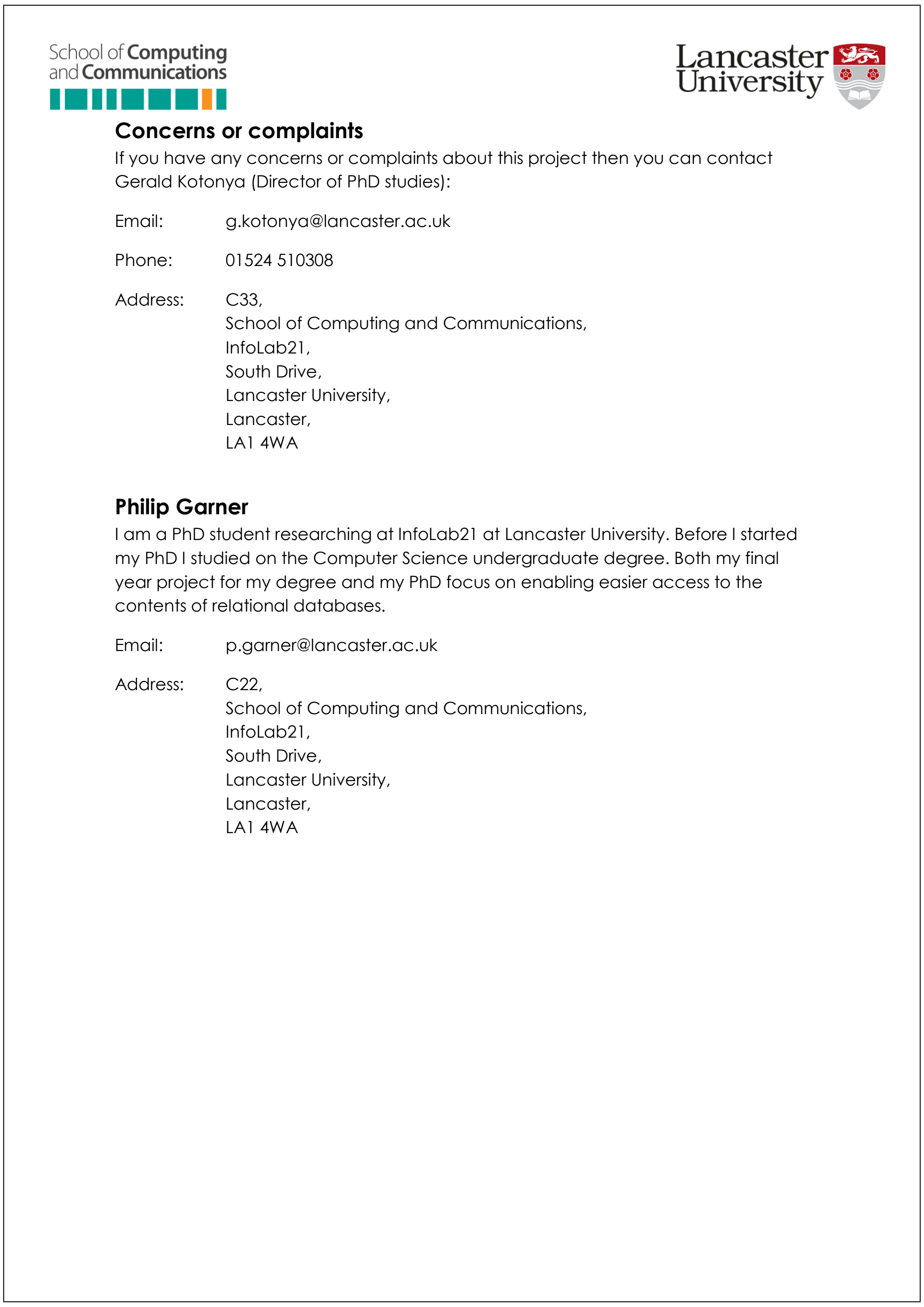
Can the way in which you learn SQL be improved through the introduction of graphical user interfaces specifically designed for educational purposes? This research is about exploring whether the introduction of software specifically designed for educational purposes can improve students understanding of SQL.

A central part of any database course involves developing the ability to extract information from a relational database using SQL. Traditionally this is taught using a textual interface such as that used with MySQL or SQLite.

As part of this research we have developed a user interface specifically designed for teaching SQL: SQL in Steps (SiS). This user interface will guide you through the process of building an SQL statement, enabling you to view the SQL you are building and the results it will return. The intention is that you will become comfortable using the user interface to build SQL statements to the point at which you can write SQL statements without the use of SiS.

This study will involve observing your interaction with relational databases through SiS and other interfaces. If you agree to take part in the study we will first assess your understanding of SQL using a short series of questions; your answers to these questions will only contribute towards this research and will, in no way, have an impact on your grade for the course. After assessing your understanding we will randomly allocate you into one of two groups; one group will have access to SiS for the first three weeks of your SQL workshops and the other will complete the course without access to SiS. If you do not agree to take part in the study you will not be given any access to SiS. At the end of week eight all access to SiS will be terminated prior to the assessed coursework which you will receive in week nine for marking in week ten. At the end of the study you may get invited to attend a focus group to further discuss SiS and its potential development in the future, this is entirely optional.

Throughout the study we will use various different pieces of information to follow you through the process of learning SQL; these include Moodle logs, SiS usage logs, formative assessment results and questionnaires. All of this information will be anonymised at the earliest opportunity and you will never be identified in any subsequent publications or thesis that stem from this work.



School of Computing and Communications

Lancaster University

Concerns or complaints

If you have any concerns or complaints about this project then you can contact Gerald Kotonya (Director of PhD studies):

Email:	g.kotonya@lancaster.ac.uk
Phone:	01524 510308
Address:	C33, School of Computing and Communications, InfoLab21, South Drive, Lancaster University, Lancaster, LA1 4WA

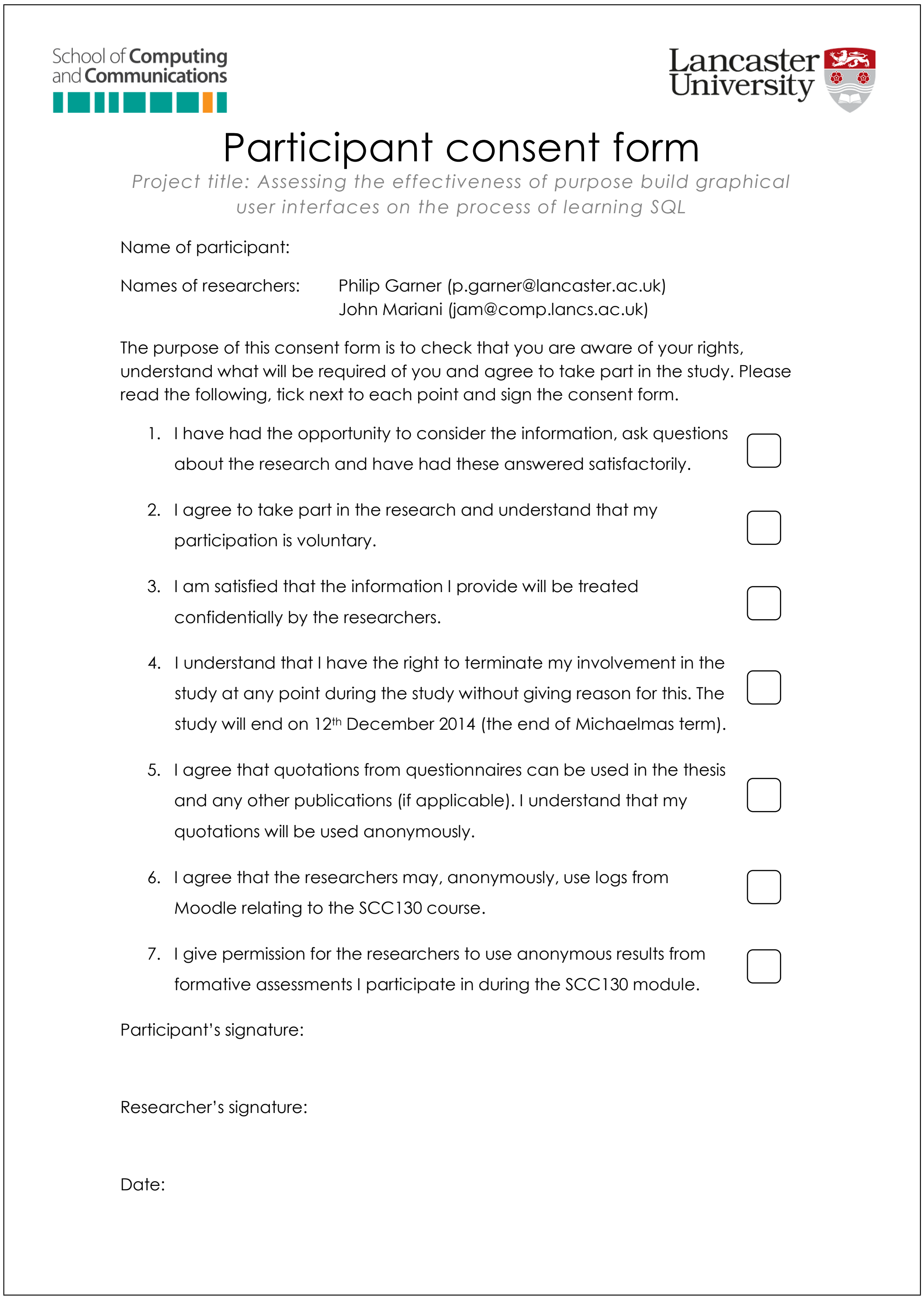
Philip Garner

I am a PhD student researching at InfoLab21 at Lancaster University. Before I started my PhD I studied on the Computer Science undergraduate degree. Both my final year project for my degree and my PhD focus on enabling easier access to the contents of relational databases.

Email:	p.garner@lancaster.ac.uk
Address:	C22, School of Computing and Communications, InfoLab21, South Drive, Lancaster University, Lancaster, LA1 4WA

C.1.1.2 Consent form

The following consent form was used to clarify that the participants understood the details of the study before it commenced.



School of Computing and Communications

Lancaster University

Participant consent form

Project title: Assessing the effectiveness of purpose build graphical user interfaces on the process of learning SQL

Name of participant:

Names of researchers: Philip Garner (p.garner@lancaster.ac.uk)
John Mariani (jam@comp.lancs.ac.uk)

The purpose of this consent form is to check that you are aware of your rights, understand what will be required of you and agree to take part in the study. Please read the following, tick next to each point and sign the consent form.

1. I have had the opportunity to consider the information, ask questions about the research and have had these answered satisfactorily. ☐
2. I agree to take part in the research and understand that my participation is voluntary. ☐
3. I am satisfied that the information I provide will be treated confidentially by the researchers. ☐
4. I understand that I have the right to terminate my involvement in the study at any point during the study without giving reason for this. The study will end on 12th December 2014 (the end of Michaelmas term). ☐
5. I agree that quotations from questionnaires can be used in the thesis and any other publications (if applicable). I understand that my quotations will be used anonymously. ☐
6. I agree that the researchers may, anonymously, use logs from Moodle relating to the SCC130 course. ☐
7. I give permission for the researchers to use anonymous results from formative assessments I participate in during the SCC130 module. ☐

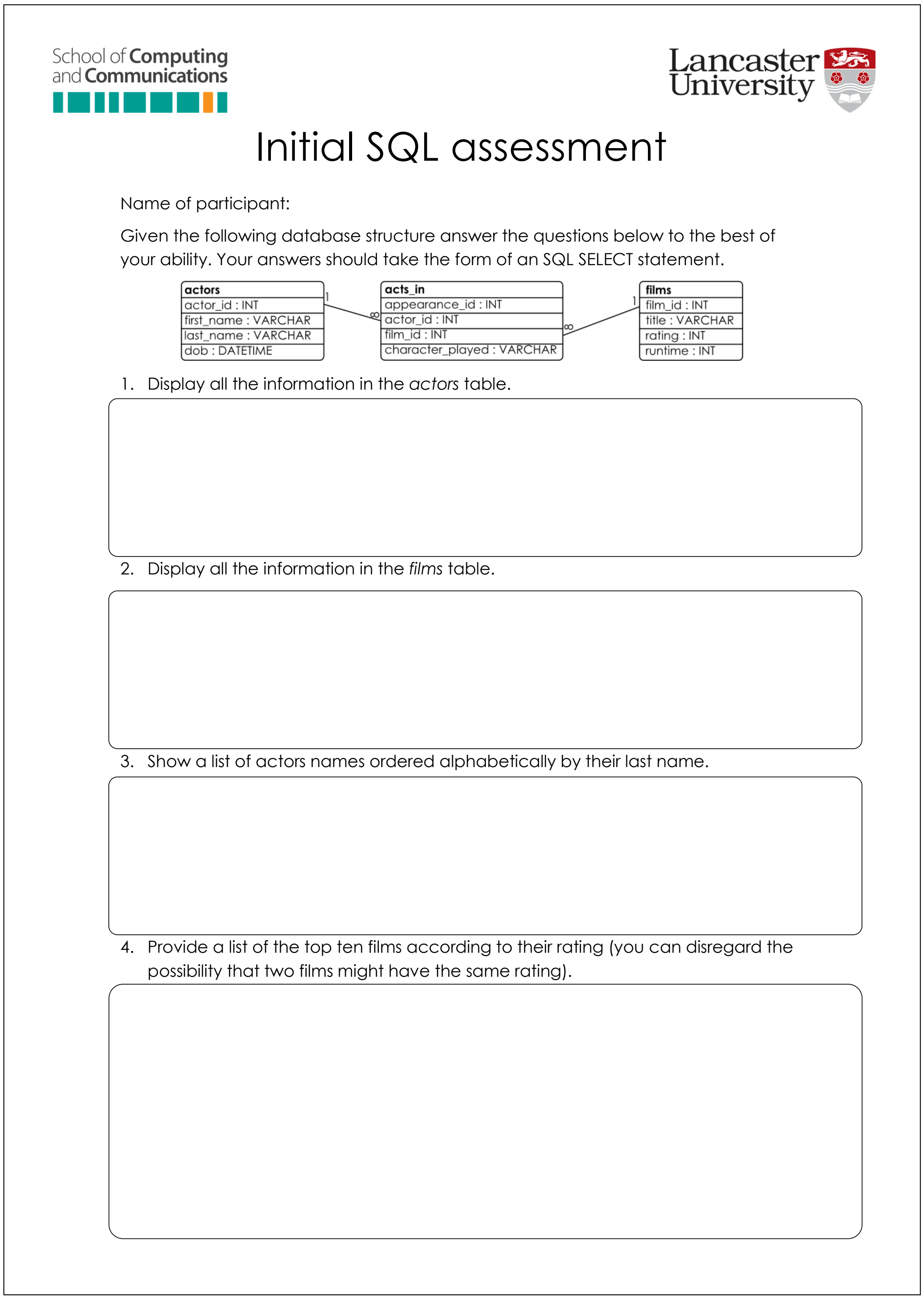
Participant's signature:

Researcher's signature:

Date:

C.1.1.3 Assessment

The participants completed the following initial assessment to establish their understanding prior to the commencement of the study.



School of Computing and Communications

Lancaster University

Initial SQL assessment

Name of participant:

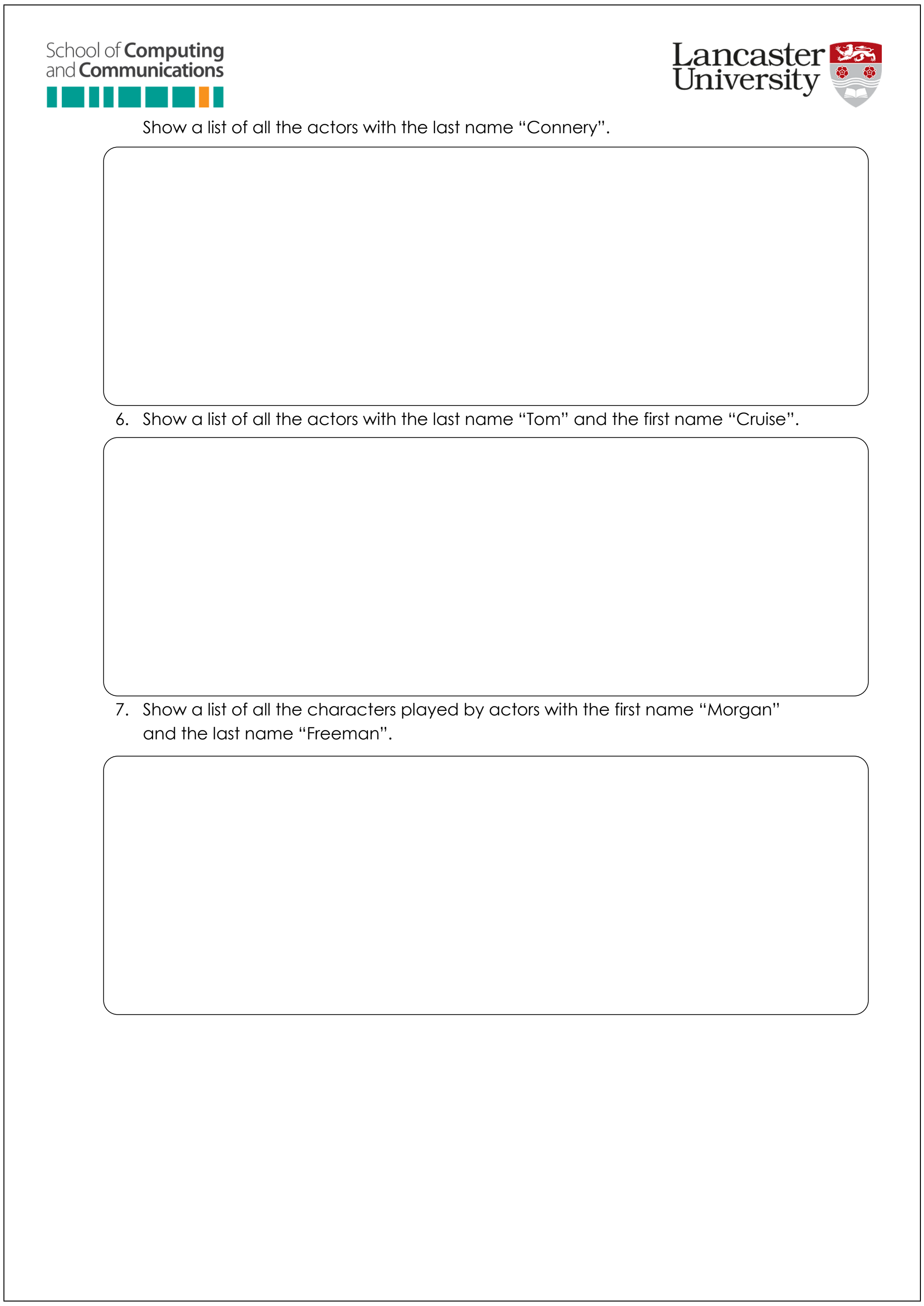
Given the following database structure answer the questions below to the best of your ability. Your answers should take the form of an SQL SELECT statement.

actors
actor_id : INT
first_name : VARCHAR
last_name : VARCHAR
dob : DATETIME

acts_in
appearance_id : INT
actor_id : INT
film_id : INT
character_played : VARCHAR

films
film_id : INT
title : VARCHAR
rating : INT
runtime : INT

1. Display all the information in the *actors* table.
2. Display all the information in the *films* table.
3. Show a list of actors names ordered alphabetically by their last name.
4. Provide a list of the top ten films according to their rating (you can disregard the possibility that two films might have the same rating).



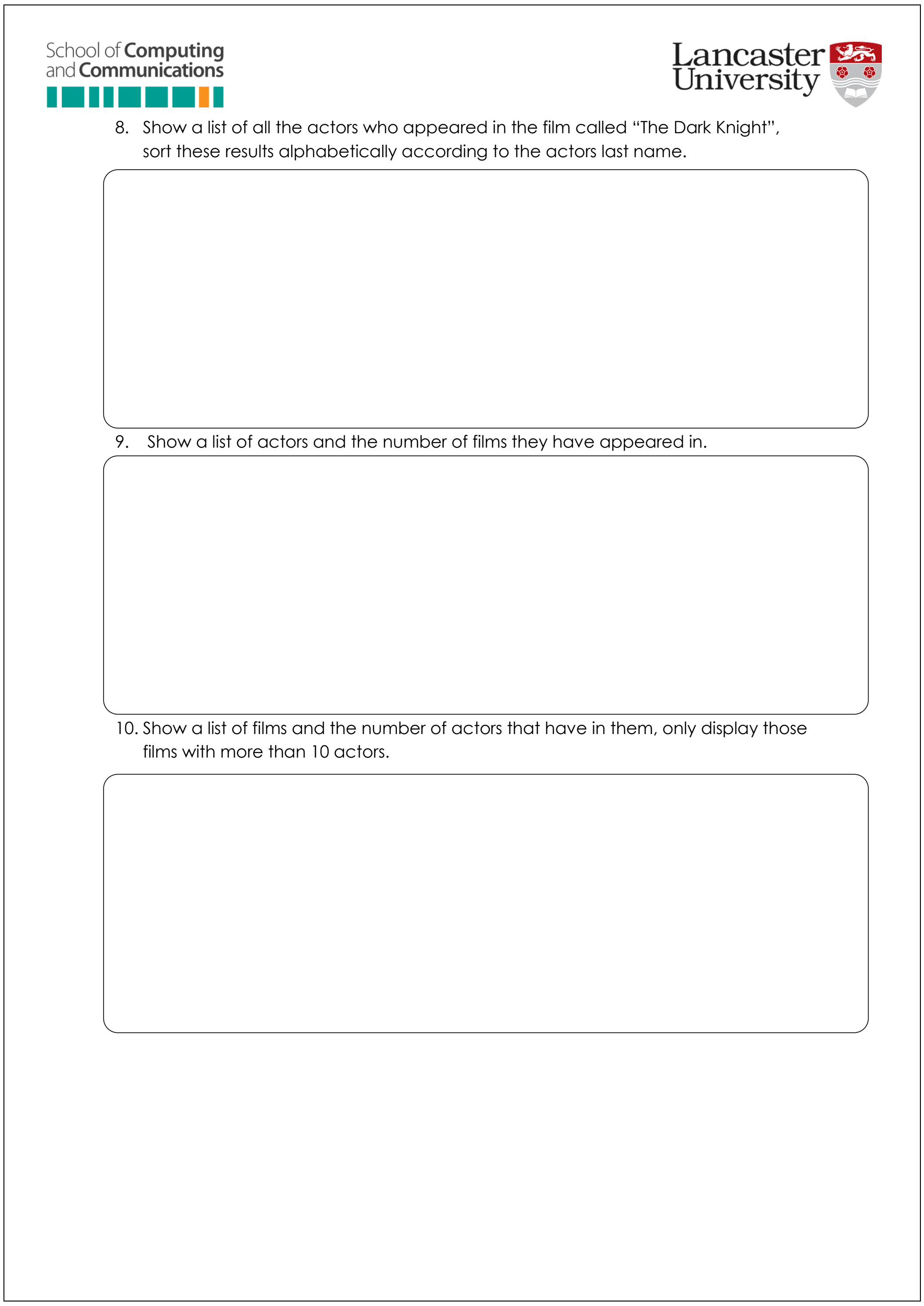
School of Computing and Communications

Lancaster University

Show a list of all the actors with the last name "Connery".

6. Show a list of all the actors with the last name "Tom" and the first name "Cruise".

7. Show a list of all the characters played by actors with the first name "Morgan" and the last name "Freeman".



School of Computing and Communications

Lancaster University

8. Show a list of all the actors who appeared in the film called “The Dark Knight”, sort these results alphabetically according to the actors last name.

9. Show a list of actors and the number of films they have appeared in.

10. Show a list of films and the number of actors that have in them, only display those films with more than 10 actors.

C.1.1.4 Getting started guide

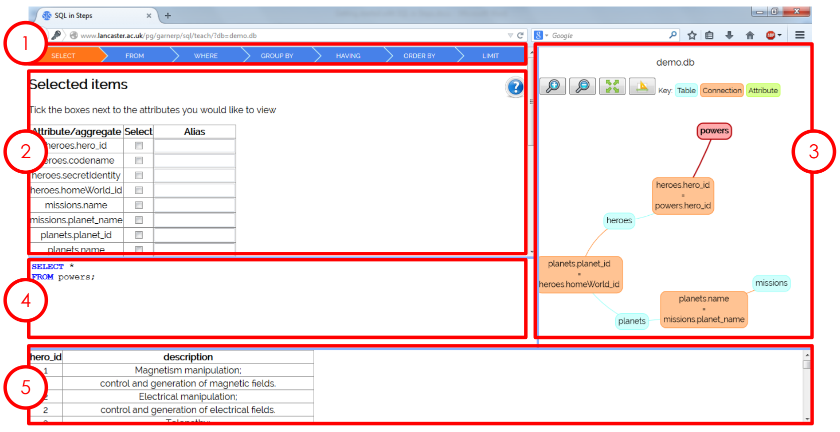
The following guide was made available to students on the home page for SQL in Steps and also on the Moodle pages for the module.

Getting started with SQL in Steps

sqlinsteps.co.uk

Thanks for taking part in the study regarding the way in which you learn SQL. I hope you find SQL in Steps (SiS) useful throughout the next couple of weeks.

Familiarising yourself with the user interface


SQL in Steps
www.lancaster.ac.uk/pg/garnerp/sql/teach/?db=demo.db
Google
1
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT
2
Selected items
Tick the boxes next to the attributes you would like to view
3
demo.db
Key: Table Connection Attribute
powers
heroes.hero_id = powers.hero_id
heroes
planets.planet_id = heroes.homeWorld_id
missions
planets.name = missions.planet_name
planets
4
5

Attribute/aggregate	Select	Alias
heroes.hero_id	☐	
heroes.codename	☐	
heroes.secretIdentity	☐	
heroes.homeWorld_id	☐	
missions.name	☐	
missions.planet_name	☐	
planets.planet_id	☐	
planets.name	☐	

```
SELECT *
FROM powers;
```

hero_id	description
1	Magnetism manipulation;
	control and generation of magnetic fields.
2	Electrical manipulation;
2	control and generation of electrical fields.

The user interface for SiS consists of five main components:

1. The workflow bar – used for switching between clauses
2. The query builder – Used for most of your query building
3. The database visualisation – Used for building the FROM clause and visualising the database structure
4. The SQL – The textual version of your query, this is the actual SQL executed
5. The query results – The results of your query are shown here

Any changes you make to the query builder or the database visualisation will be immediately reflected in the SQL and results panel.

Getting started

Here we'll talk you through building your first query with SiS.

Head to sqlinsteps.co.uk

Go to sqlinsteps.co.uk and pick a database to use.

sqlinsteps.co.uk

Log in using your university credentials

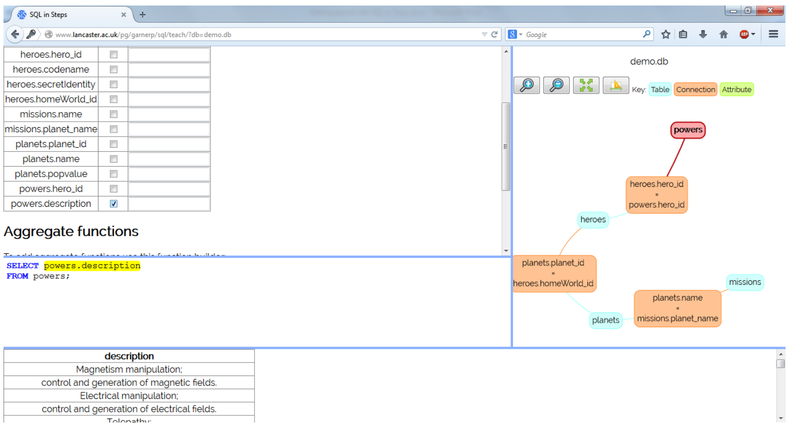
Use your usual university username and password to log into SiS.

Pick a table to query

On the right of SiS is a graphical representation of the database structure, tables are shown in blue and relationships in orange. Click on a table to view its contents. You should have noticed that the SQL has updated to show the table you chose in the FROM clause and the results display the contents of the table. You have now built your first query using SiS, you should be able to understand how your actions with the graphical user interface (GUI) have affected the SQL and the results.

Choosing the attributes (or columns) to view

The select clause can be used to show different columns in your results, if you don't pick any then SiS uses SELECT * which will show all columns. To edit your SELECT clause place a tick next to the boxes you would like to see in the results. As you do this you will see both the SQL and results update. Remember, the columns you pick must be from tables that appear in your FROM clause.


SQL in Steps
www.lancaster.ac.uk/pg/garnerp/sql/teach/?db=demo.db
Google
demo.db
Key: Table Connection Attribute
powers
heroes.hero_id = powers.hero_id
heroes
planets.planet_id = heroes.homeWorld_id
missions
planets.name = missions.planet_name
planets

heroes.hero_id	☐	
heroes.codename	☐	
heroes.secretIdentity	☐	
heroes.homeWorld_id	☐	
missions.name	☐	
missions.planet_name	☐	
planets.planet_id	☐	
planets.name	☐	
planets.popvalue	☐	
powers.hero_id	☐	
powers.description	✓	

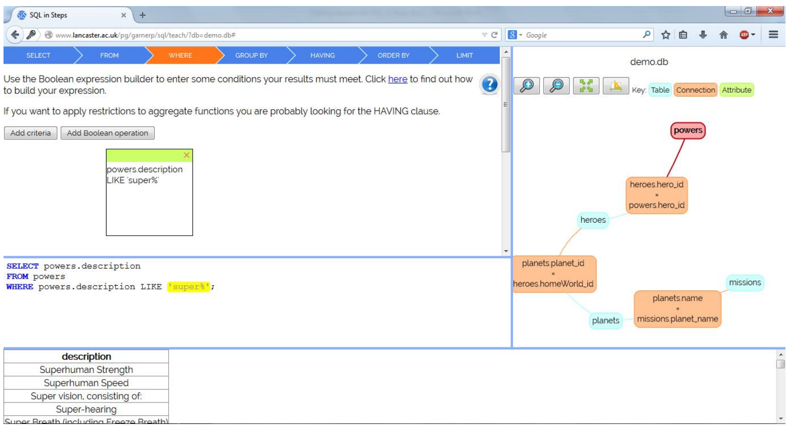
Aggregate functions

```
SELECT powers.description
FROM powers;
```

description
Magnetism manipulation;
control and generation of magnetic fields.
Electrical manipulation;
control and generation of electrical fields.

Adding a condition

Often you don't want to see all the rows in the table, we use the WHERE clause to only show the ones we are interested in. SiS uses a graphical representation of WHERE clauses to help you understand them. To add criteria switch to the WHERE clause using the workflow at the top of the page. Click "Add criteria" to insert a new criteria widget, enter your criteria on the widget, these generally take the form of <attribute> <comparison> <criteria>; for example: *id = 1*.


SQL in Steps
www.lancaster.ac.uk/pg/garnerp/sql/teach/?db=demo.db#
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT
Add criteria
Add Boolean operation
demo.db
Key: Table Connection Attribute
powers
heroes.hero_id = powers.hero_id
heroes
planets.planet_id = heroes.homeWorld_id
missions
planets.name = missions.planet_name
planets

Use the Boolean expression builder to enter some conditions your results must meet. Click here to find out how to build your expression.

If you want to apply restrictions to aggregate functions you are probably looking for the HAVING clause.

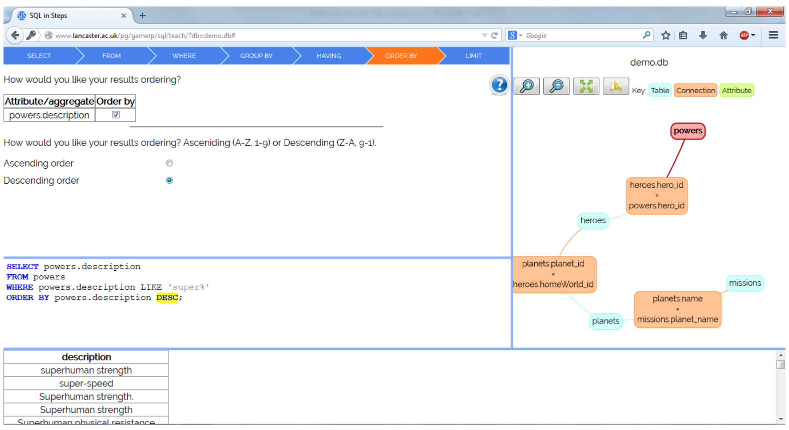
powers.description LIKE 'super%'

```
SELECT powers.description
FROM powers
WHERE powers.description LIKE 'super%';
```

description
Superhuman Strength
Superhuman Speed
Super vision, consisting of:
Super-hearing
Super Breath (including Freeze Breath)

Order your results

SQL makes no promises about the order the results are returned in so, if you want to display your results in a sensible order you can use an ORDER BY CLAUSE. Switch to the ORDER BY clause in the workflow bar at the top of the page. Here you will see a set of attributes you can order by, pick one. Then select the way in which you want the results to be ordered (ascending or descending). As you do this you will see the SQL and results update accordingly.


SQL in Steps
www.lancaster.ac.uk/pg/garnerp/sql/teach/?db=demo.db#
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT
demo.db
Key: Table Connection Attribute
powers
heroes.hero_id = powers.hero_id
heroes
planets.planet_id = heroes.homeWorld_id
missions
planets.name = missions.planet_name
planets

How would you like your results ordering?

Attribute/aggregate	Order by
powers.description	☑

How would you like your results ordering? Asceniding (A-Z, 1-9) or Descending (Z-A, 9-1).

Ascending order ○
Descending order ●

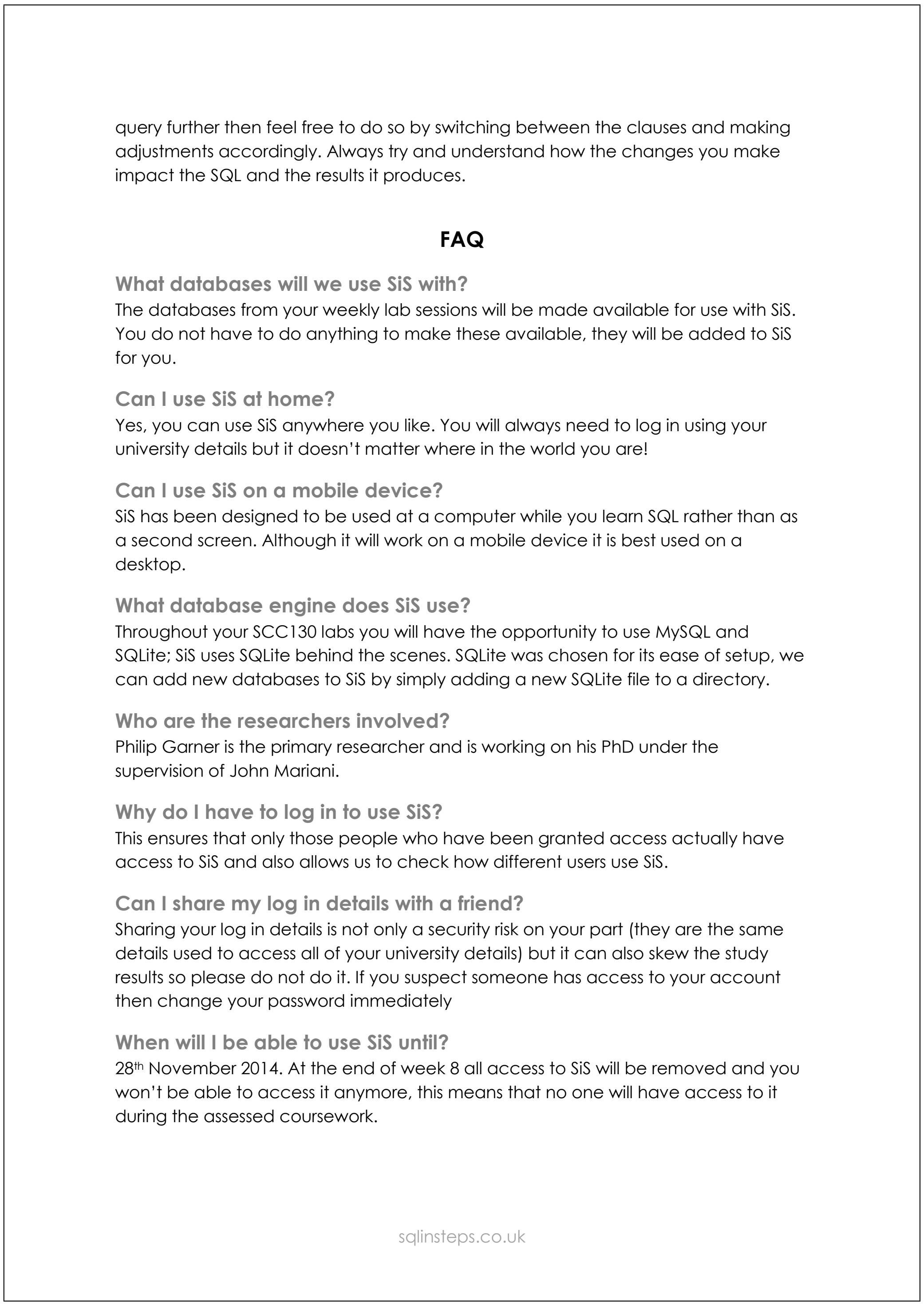
```
SELECT powers.description
FROM powers
WHERE powers.description LIKE 'super%'
ORDER BY powers.description DESC;
```

description
superhuman strength
super-speed
Superhuman strength.
Superhuman strength
Superhuman physical resistance

Customise your query more

In this quick tutorial we have built a SELECT statement consisting of a SELECT, FROM, WHERE and ORDER BY clause, it gives you an insight into how SiS can be used and hopefully provides you with an idea of its capabilities. If you wish to customise your

sqlinsteps.co.uk



query further then feel free to do so by switching between the clauses and making adjustments accordingly. Always try and understand how the changes you make impact the SQL and the results it produces.

FAQ

What databases will we use SiS with?
The databases from your weekly lab sessions will be made available for use with SiS. You do not have to do anything to make these available, they will be added to SiS for you.

Can I use SiS at home?
Yes, you can use SiS anywhere you like. You will always need to log in using your university details but it doesn't matter where in the world you are!

Can I use SiS on a mobile device?
SiS has been designed to be used at a computer while you learn SQL rather than as a second screen. Although it will work on a mobile device it is best used on a desktop.

What database engine does SiS use?
Throughout your SCC130 labs you will have the opportunity to use MySQL and SQLite; SiS uses SQLite behind the scenes. SQLite was chosen for its ease of setup, we can add new databases to SiS by simply adding a new SQLite file to a directory.

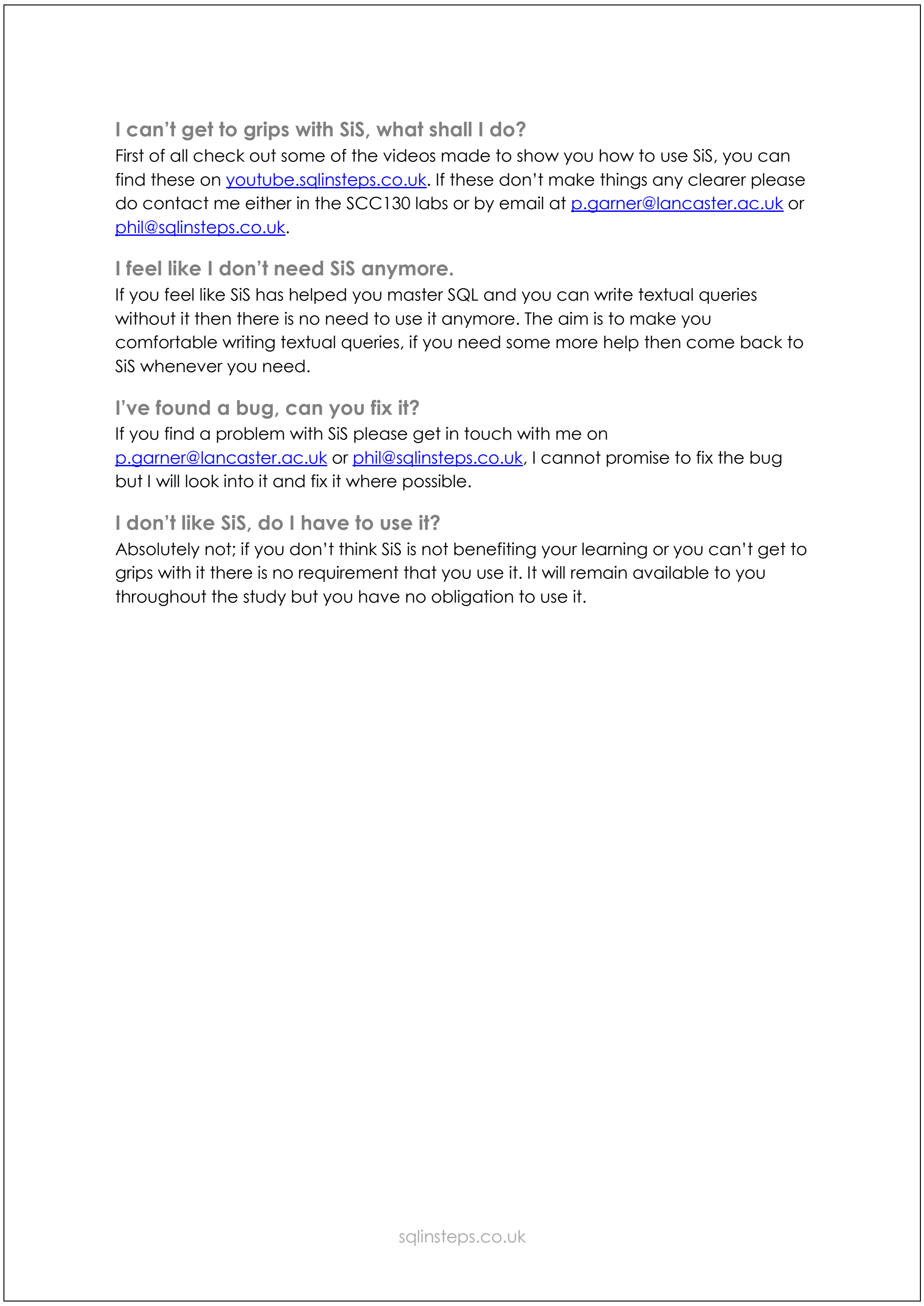
Who are the researchers involved?
Philip Garner is the primary researcher and is working on his PhD under the supervision of John Mariani.

Why do I have to log in to use SiS?
This ensures that only those people who have been granted access actually have access to SiS and also allows us to check how different users use SiS.

Can I share my log in details with a friend?
Sharing your log in details is not only a security risk on your part (they are the same details used to access all of your university details) but it can also skew the study results so please do not do it. If you suspect someone has access to your account then change your password immediately

When will I be able to use SiS until?
28th November 2014. At the end of week 8 all access to SiS will be removed and you won't be able to access it anymore, this means that no one will have access to it during the assessed coursework.

sqlinsteps.co.uk



I can't get to grips with SiS, what shall I do?
First of all check out some of the videos made to show you how to use SiS, you can find these on youtube.sqlinsteps.co.uk. If these don't make things any clearer please do contact me either in the SCC130 labs or by email at p.garner@lancaster.ac.uk or phil@sqlinsteps.co.uk.

I feel like I don't need SiS anymore.
If you feel like SiS has helped you master SQL and you can write textual queries without it then there is no need to use it anymore. The aim is to make you comfortable writing textual queries, if you need some more help then come back to SiS whenever you need.

I've found a bug, can you fix it?
If you find a problem with SiS please get in touch with me on p.garner@lancaster.ac.uk or phil@sqlinsteps.co.uk, I cannot promise to fix the bug but I will look into it and fix it where possible.

I don't like SiS, do I have to use it?
Absolutely not; if you don't think SiS is not benefiting your learning or you can't get to grips with it there is no requirement that you use it. It will remain available to you throughout the study but you have no obligation to use it.

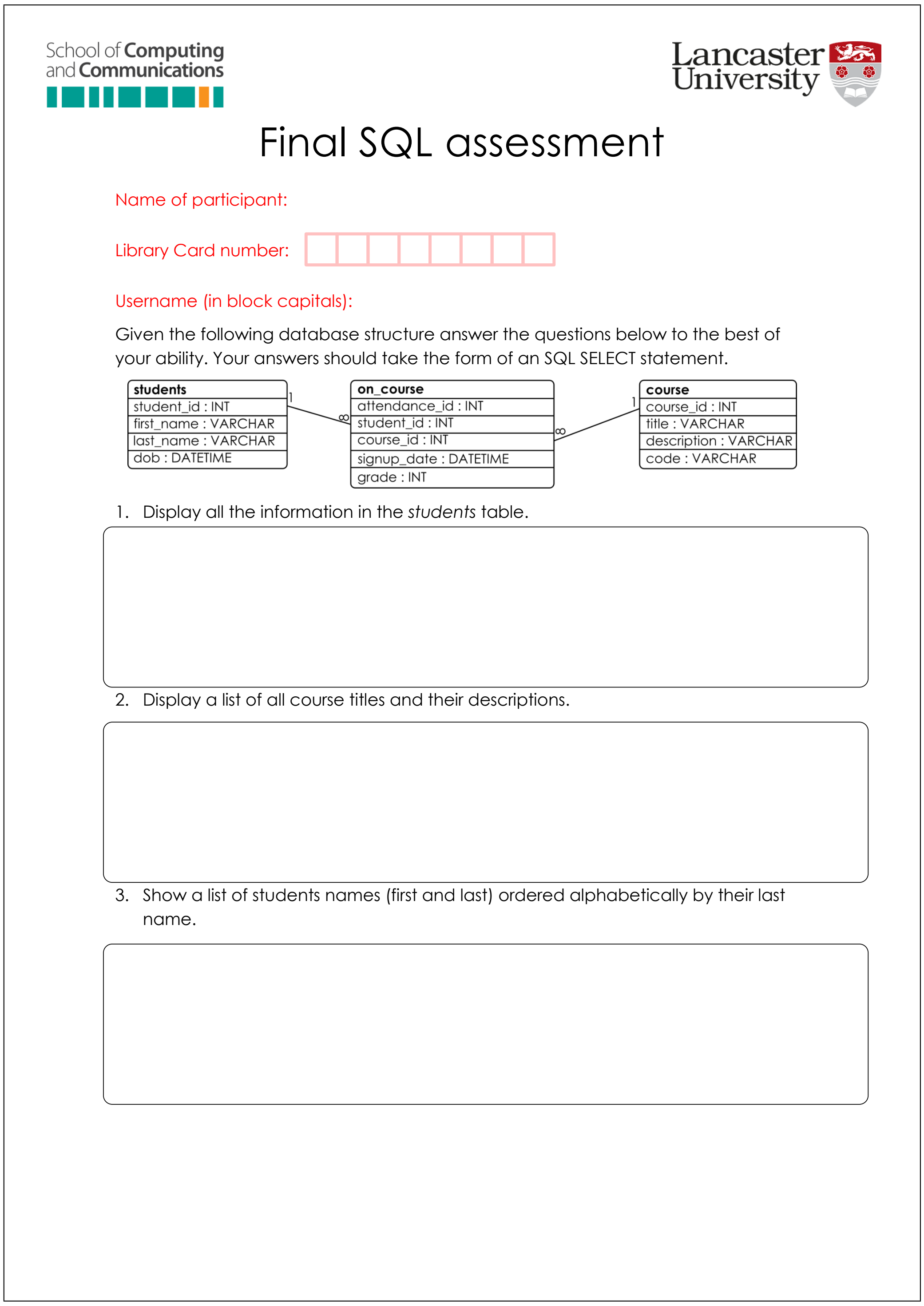
sqlinsteps.co.uk

C.1.2 End of study

To establish the extent to which the user improved throughout the duration of the study they were each given an assessment and a questionnaire at the end of the three week study.

C.1.2.1 Assessment

The following assessment aimed to establish the understanding of SQL after three weeks of practical assignments with/without SiS use.



School of Computing and Communications

Lancaster University

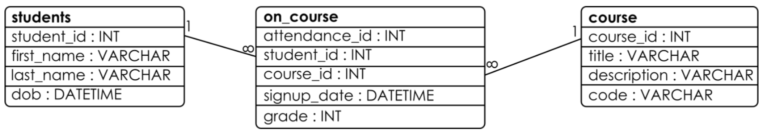
Final SQL assessment

Name of participant:

Library Card number:

Username (in block capitals):

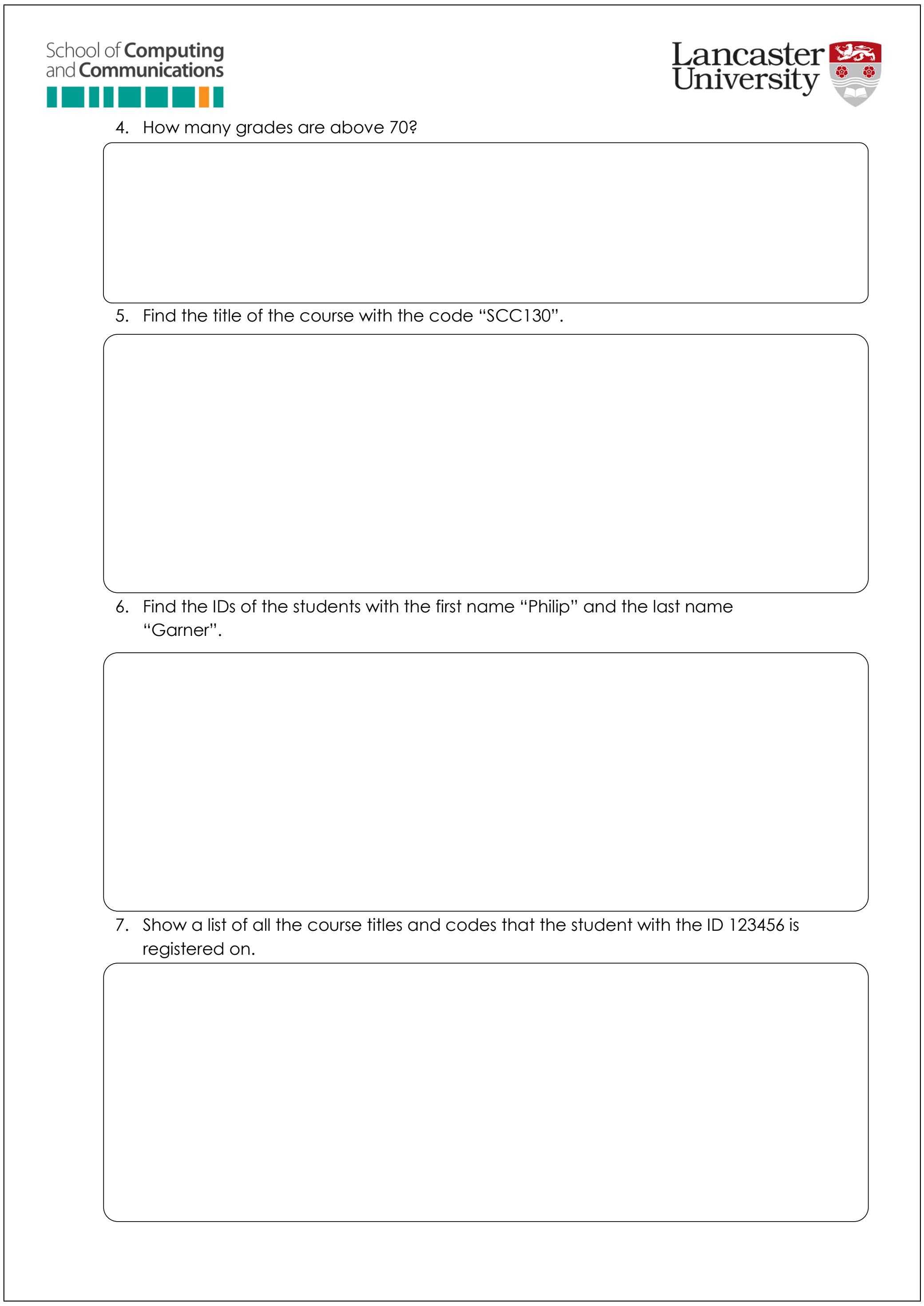
Given the following database structure answer the questions below to the best of your ability. Your answers should take the form of an SQL SELECT statement.



1. Display all the information in the *students* table.

2. Display a list of all course titles and their descriptions.

3. Show a list of students names (first and last) ordered alphabetically by their last name.



School of Computing and Communications

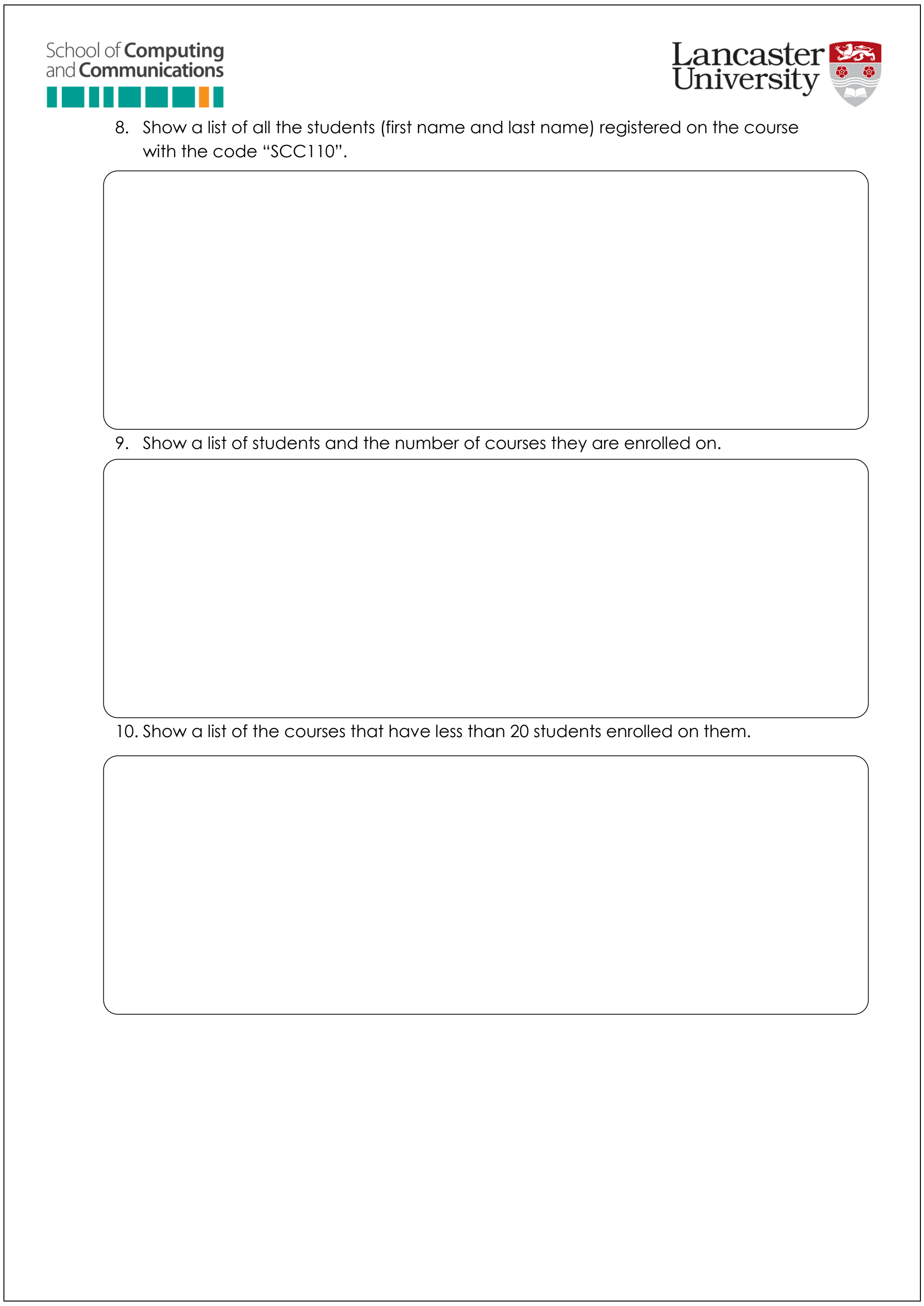
Lancaster University

4. How many grades are above 70?

5. Find the title of the course with the code “SCC130”.

6. Find the IDs of the students with the first name “Philip” and the last name “Garner”.

7. Show a list of all the course titles and codes that the student with the ID 123456 is registered on.



School of Computing and Communications

Lancaster University

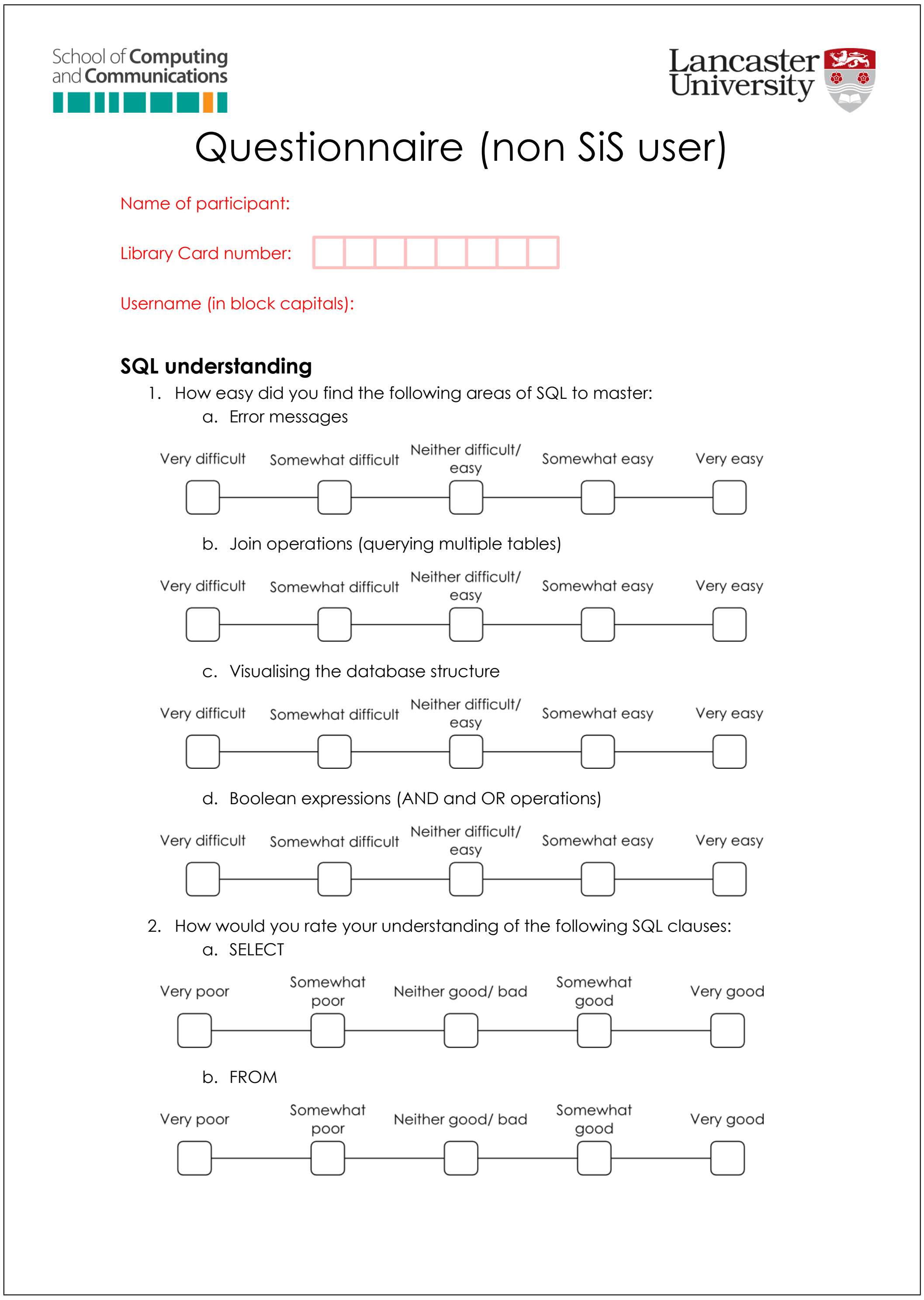
8. Show a list of all the students (first name and last name) registered on the course with the code "SCC110".

9. Show a list of students and the number of courses they are enrolled on.

10. Show a list of the courses that have less than 20 students enrolled on them.

C.1.2.2 Non-SiS user questionnaire

Those participants who did not use SiS were given a questionnaire including questions relating to their understanding of SQL and learning it in a textual environment.



School of Computing and Communications

Lancaster University

Questionnaire (non SiS user)

Name of participant:

Library Card number:

Username (in block capitals):

SQL understanding

1. How easy did you find the following areas of SQL to master:
   a. Error messages

Very difficult	Somewhat difficult	Neither difficult/ easy	Somewhat easy	Very easy
☐	☐	☐	☐	☐

   b. Join operations (querying multiple tables)

Very difficult	Somewhat difficult	Neither difficult/ easy	Somewhat easy	Very easy
☐	☐	☐	☐	☐

   c. Visualising the database structure

Very difficult	Somewhat difficult	Neither difficult/ easy	Somewhat easy	Very easy
☐	☐	☐	☐	☐

   d. Boolean expressions (AND and OR operations)

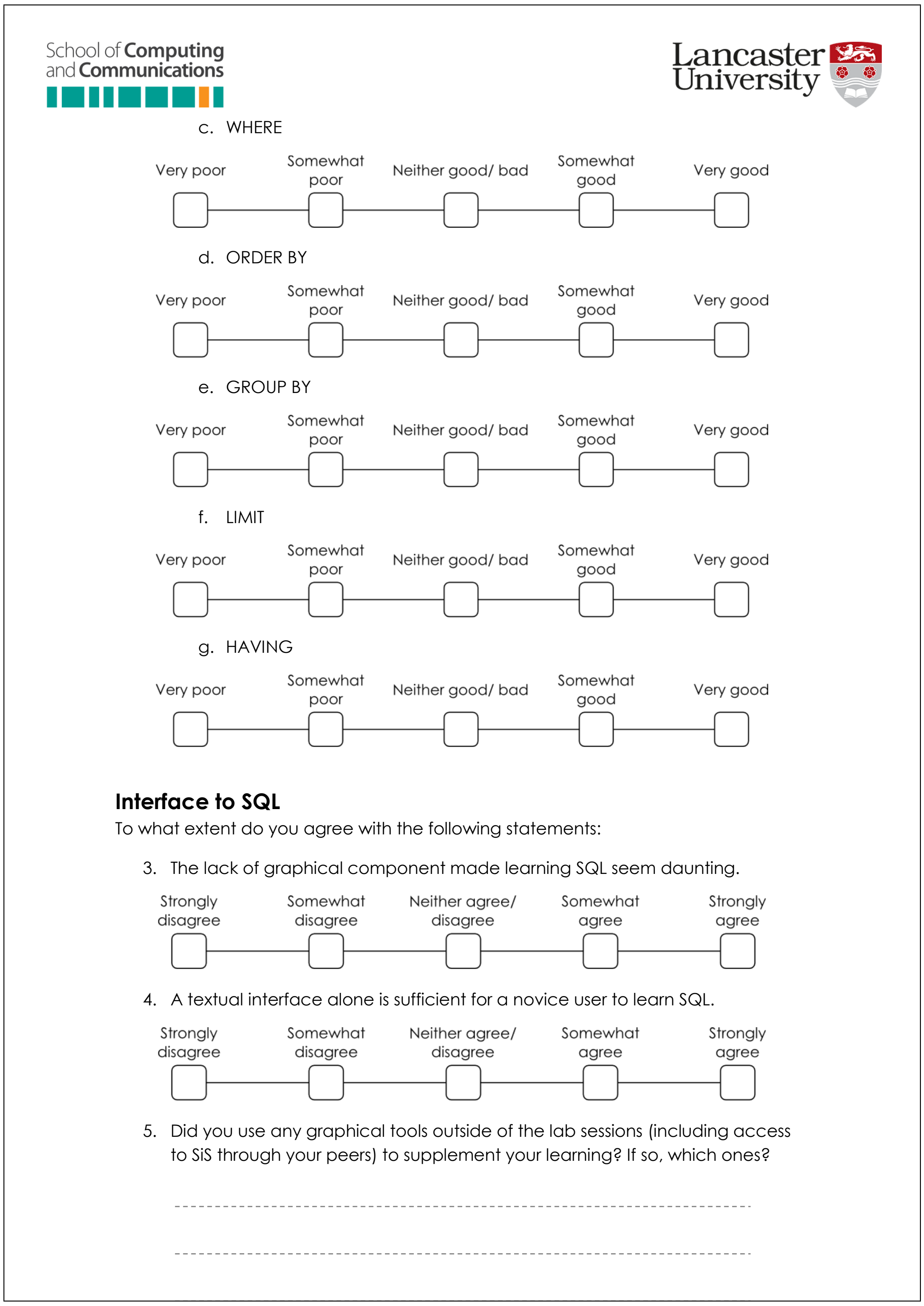
Very difficult	Somewhat difficult	Neither difficult/ easy	Somewhat easy	Very easy
☐	☐	☐	☐	☐

2. How would you rate your understanding of the following SQL clauses:
   a. SELECT

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐

   b. FROM

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐



School of Computing and Communications

Lancaster University

c. WHERE

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐

d. ORDER BY

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐

e. GROUP BY

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐

f. LIMIT

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐

g. HAVING

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐

Interface to SQL

To what extent do you agree with the following statements:

3. The lack of graphical component made learning SQL seem daunting.

Strongly disagree	Somewhat disagree	Neither agree/ disagree	Somewhat agree	Strongly agree
☐	☐	☐	☐	☐

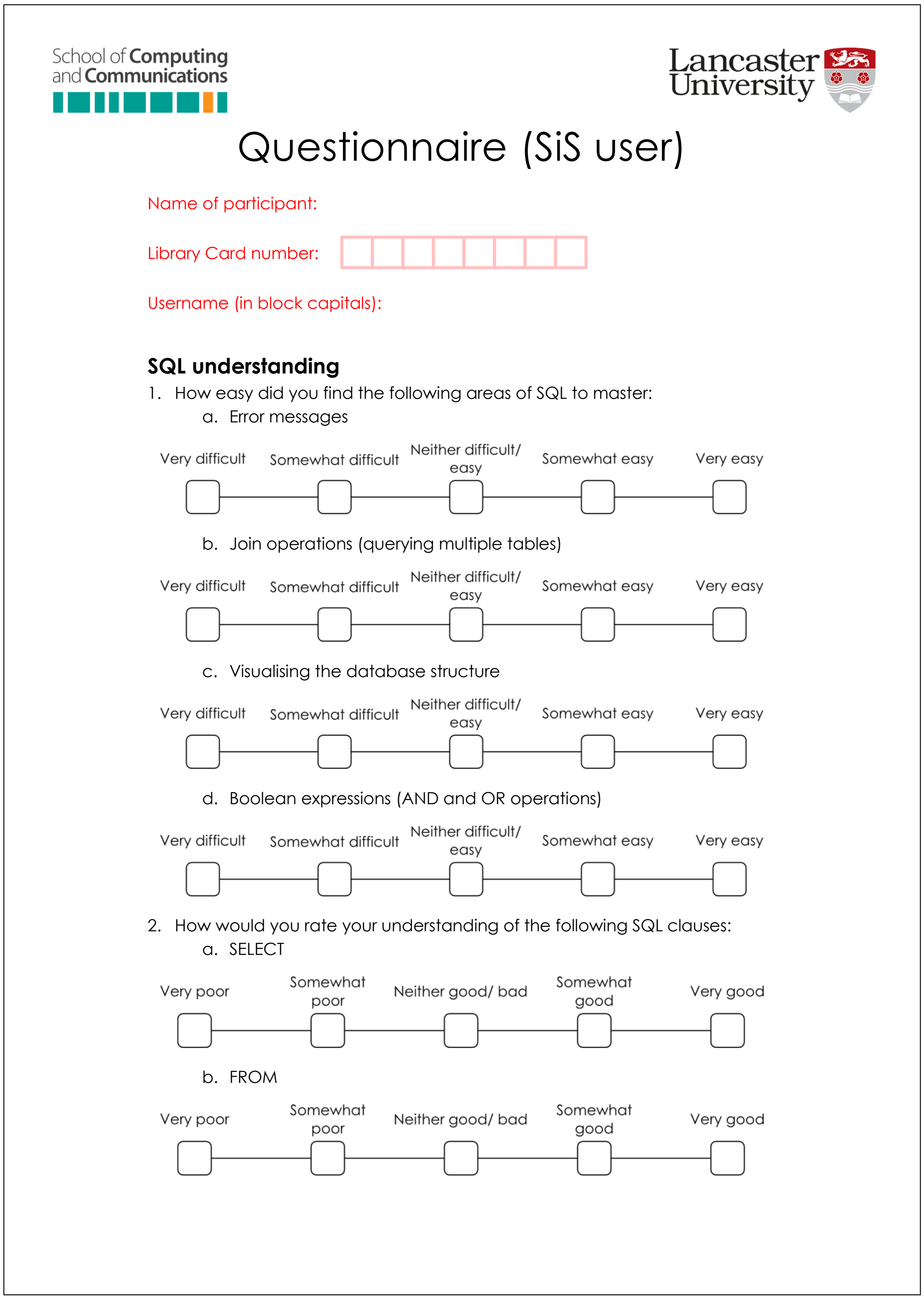
4. A textual interface alone is sufficient for a novice user to learn SQL.

Strongly disagree	Somewhat disagree	Neither agree/ disagree	Somewhat agree	Strongly agree
☐	☐	☐	☐	☐

5. Did you use any graphical tools outside of the lab sessions (including access to SiS through your peers) to supplement your learning? If so, which ones?

C.1.2.3 SiS user questionnaire

Participants with access to SiS were given a questionnaire similar, in large part, to that given to the non-SiS users but with the addition of questions relating to SiS.



School of Computing and Communications

Lancaster University

Questionnaire (SiS user)

Name of participant:

Library Card number:

Username (in block capitals):

SQL understanding

1. How easy did you find the following areas of SQL to master:
   a. Error messages

Very difficult	Somewhat difficult	Neither difficult/ easy	Somewhat easy	Very easy
☐	☐	☐	☐	☐

   b. Join operations (querying multiple tables)

Very difficult	Somewhat difficult	Neither difficult/ easy	Somewhat easy	Very easy
☐	☐	☐	☐	☐

   c. Visualising the database structure

Very difficult	Somewhat difficult	Neither difficult/ easy	Somewhat easy	Very easy
☐	☐	☐	☐	☐

   d. Boolean expressions (AND and OR operations)

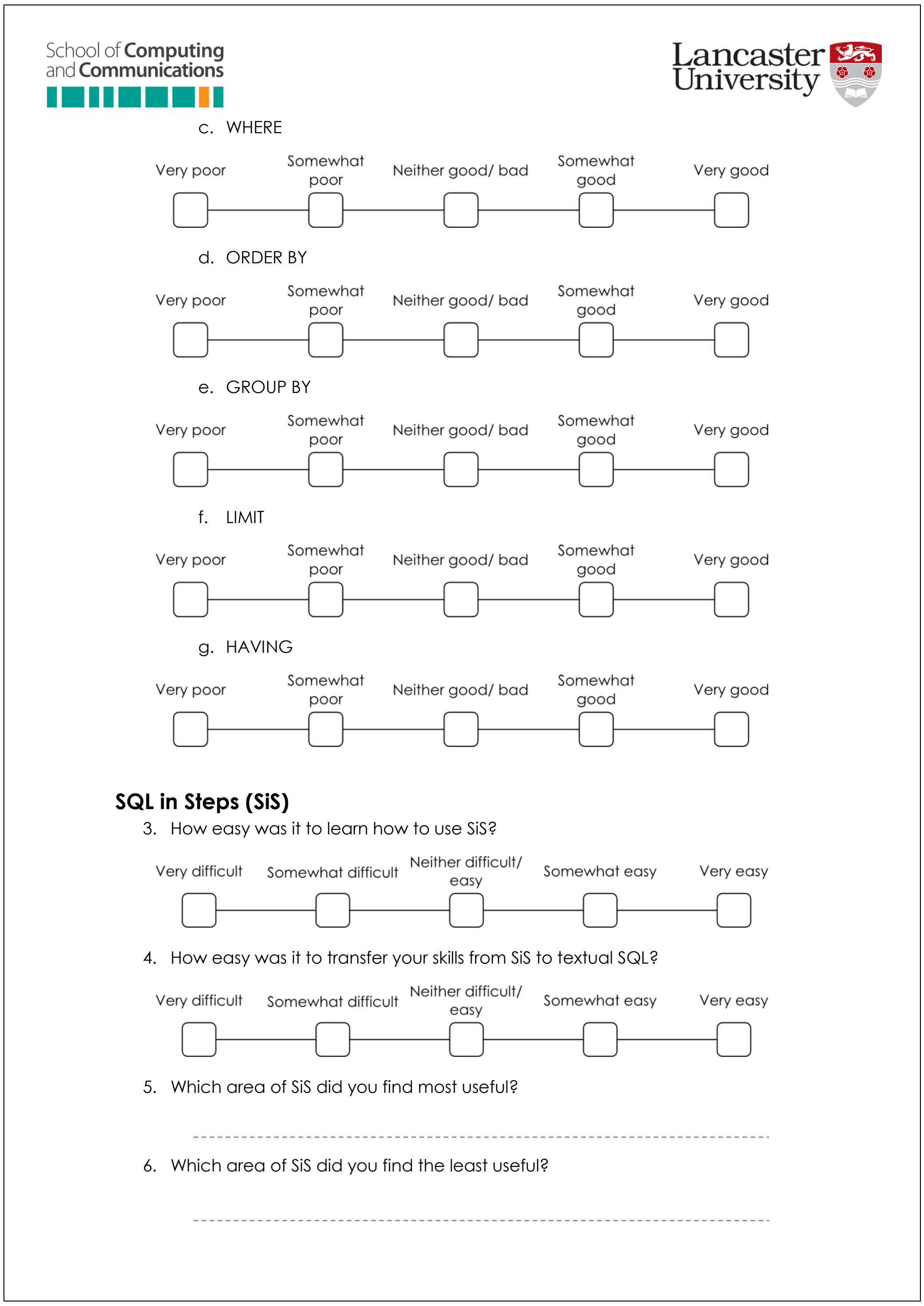
Very difficult	Somewhat difficult	Neither difficult/ easy	Somewhat easy	Very easy
☐	☐	☐	☐	☐

2. How would you rate your understanding of the following SQL clauses:
   a. SELECT

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐

   b. FROM

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐



School of Computing and Communications

Lancaster University

c. WHERE

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐

d. ORDER BY

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐

e. GROUP BY

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐

f. LIMIT

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐

g. HAVING

Very poor	Somewhat poor	Neither good/ bad	Somewhat good	Very good
☐	☐	☐	☐	☐

SQL in Steps (SiS)

3. How easy was it to learn how to use SiS?

Very difficult	Somewhat difficult	Neither difficult/ easy	Somewhat easy	Very easy
☐	☐	☐	☐	☐

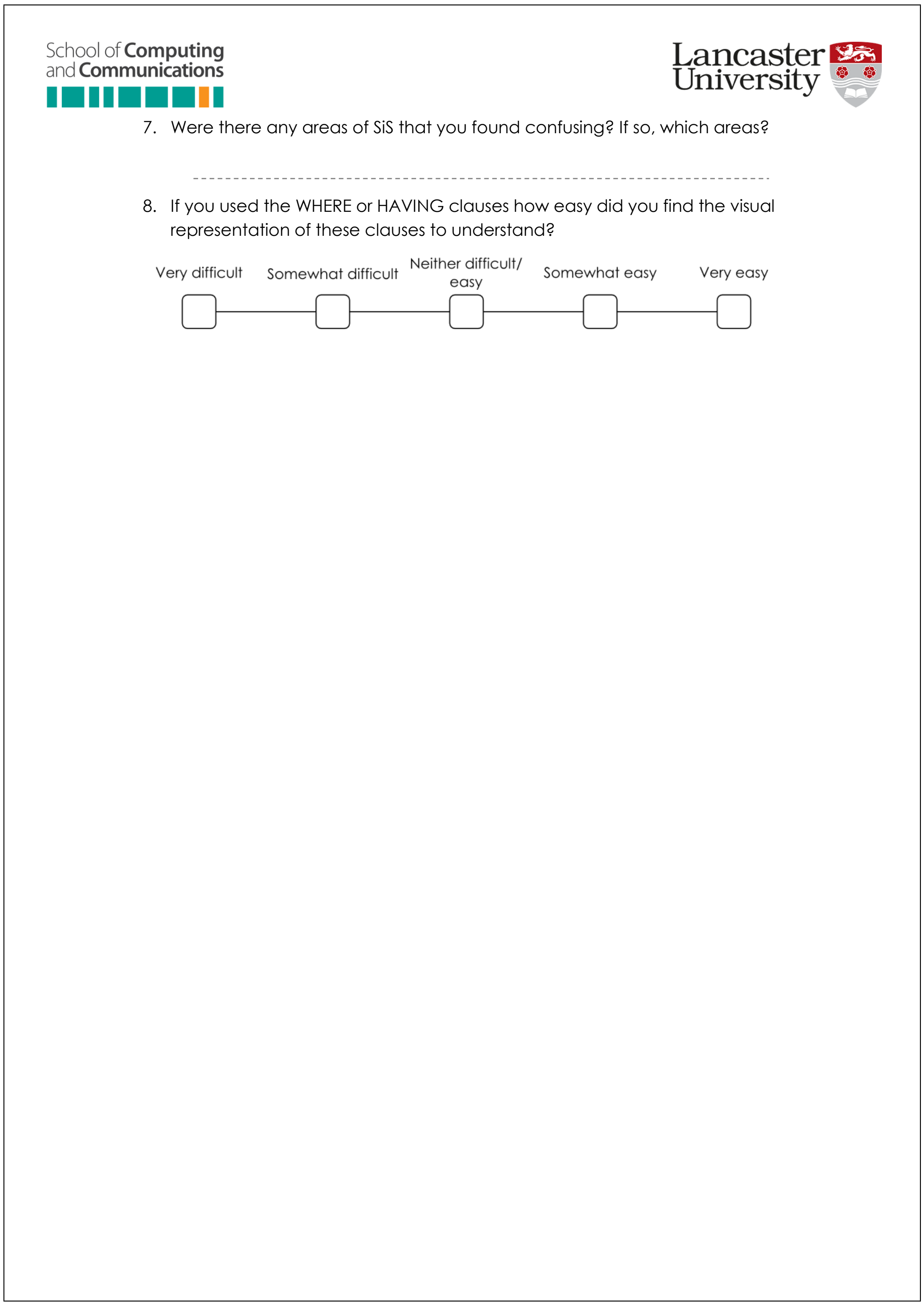
4. How easy was it to transfer your skills from SiS to textual SQL?

Very difficult	Somewhat difficult	Neither difficult/ easy	Somewhat easy	Very easy
☐	☐	☐	☐	☐

5. Which area of SiS did you find most useful?

----

6. Which area of SiS did you find the least useful?



School of Computing and Communications

Lancaster University

7. Were there any areas of SiS that you found confusing? If so, which areas?

----

8. If you used the WHERE or HAVING clauses how easy did you find the visual representation of these clauses to understand?

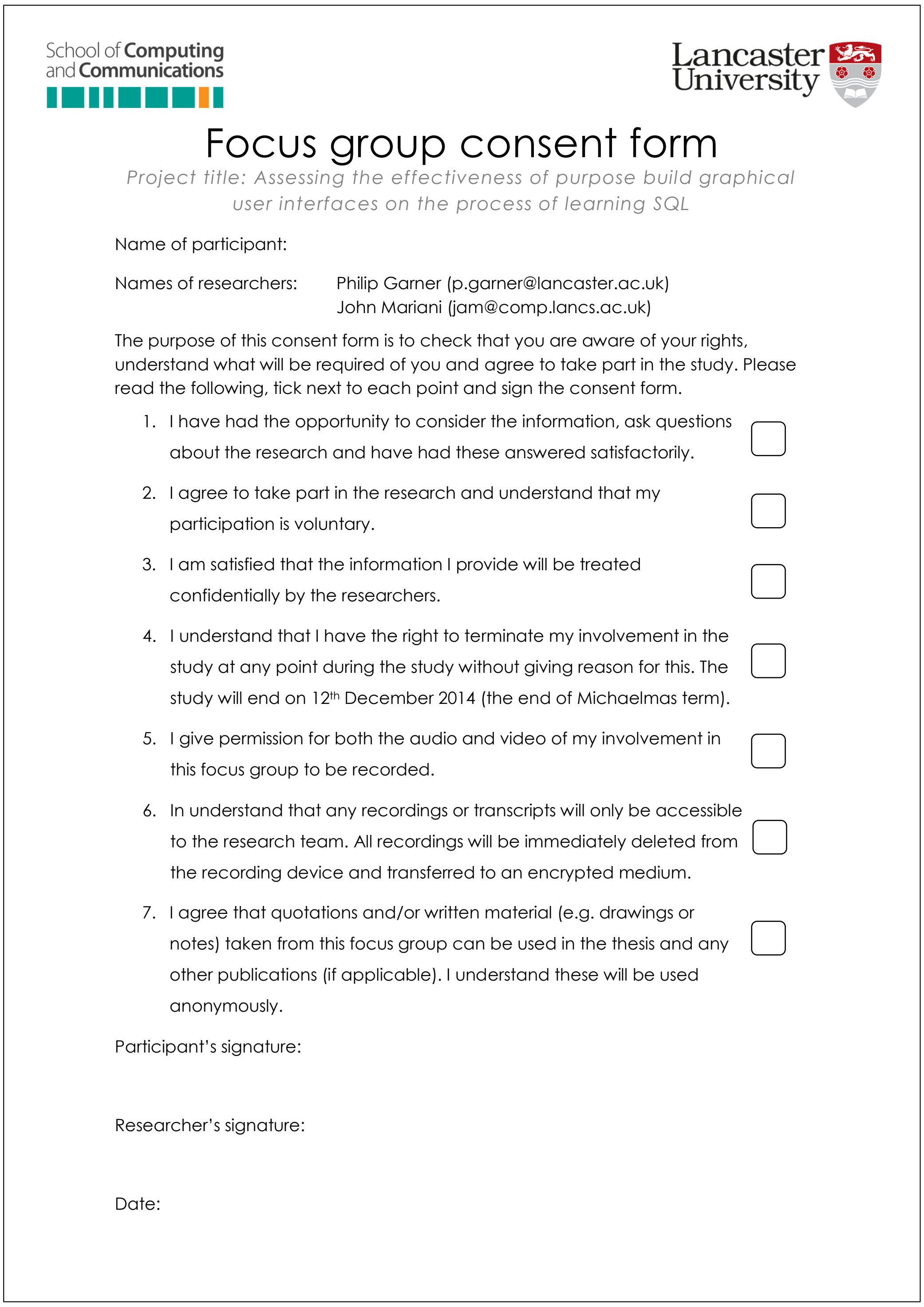
Very difficult	Somewhat difficult	Neither difficult/ easy	Somewhat easy	Very easy
☐	☐	☐	☐	☐

C.1.3 Focus Group

After the completion of the study a selection of students were invited to attend a focus group to further discuss SiS and how it might be further developed in the future.

C.1.3.1 Consent form

The focus group participants were asked to complete a consent form allowing the group to be recorded.



School of Computing and Communications

Lancaster University

Focus group consent form

Project title: Assessing the effectiveness of purpose build graphical user interfaces on the process of learning SQL

Name of participant:

Names of researchers:	Philip Garner (p.garner@lancaster.ac.uk)
	John Mariani (jam@comp.lancs.ac.uk)

The purpose of this consent form is to check that you are aware of your rights, understand what will be required of you and agree to take part in the study. Please read the following, tick next to each point and sign the consent form.

1. I have had the opportunity to consider the information, ask questions about the research and have had these answered satisfactorily. ☐
2. I agree to take part in the research and understand that my participation is voluntary. ☐
3. I am satisfied that the information I provide will be treated confidentially by the researchers. ☐
4. I understand that I have the right to terminate my involvement in the study at any point during the study without giving reason for this. The study will end on 12th December 2014 (the end of Michaelmas term). ☐
5. I give permission for both the audio and video of my involvement in this focus group to be recorded. ☐
6. In understand that any recordings or transcripts will only be accessible to the research team. All recordings will be immediately deleted from the recording device and transferred to an encrypted medium. ☐
7. I agree that quotations and/or written material (e.g. drawings or notes) taken from this focus group can be used in the thesis and any other publications (if applicable). I understand these will be used anonymously. ☐

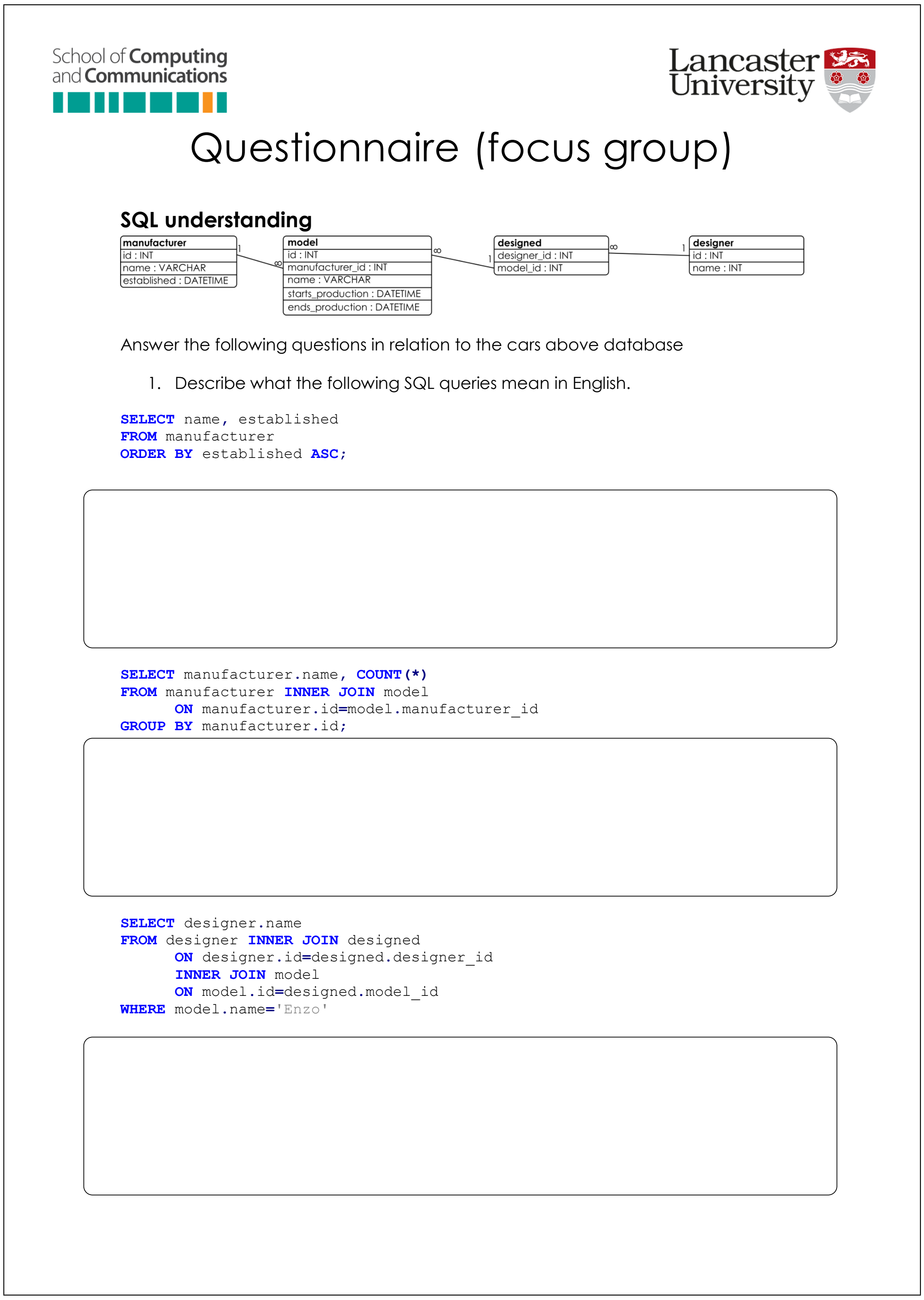
Participant's signature:

Researcher's signature:

Date:

C.1.3.2 Questionnaire

A more detailed questionnaire was also given to the focus group participants to establish an in-depth understanding of their knowledge of SQL.

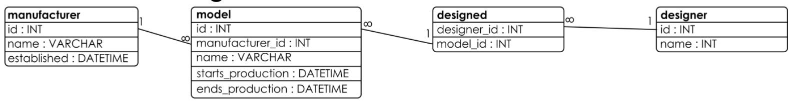


School of Computing and Communications

Lancaster University

Questionnaire (focus group)

SQL understanding



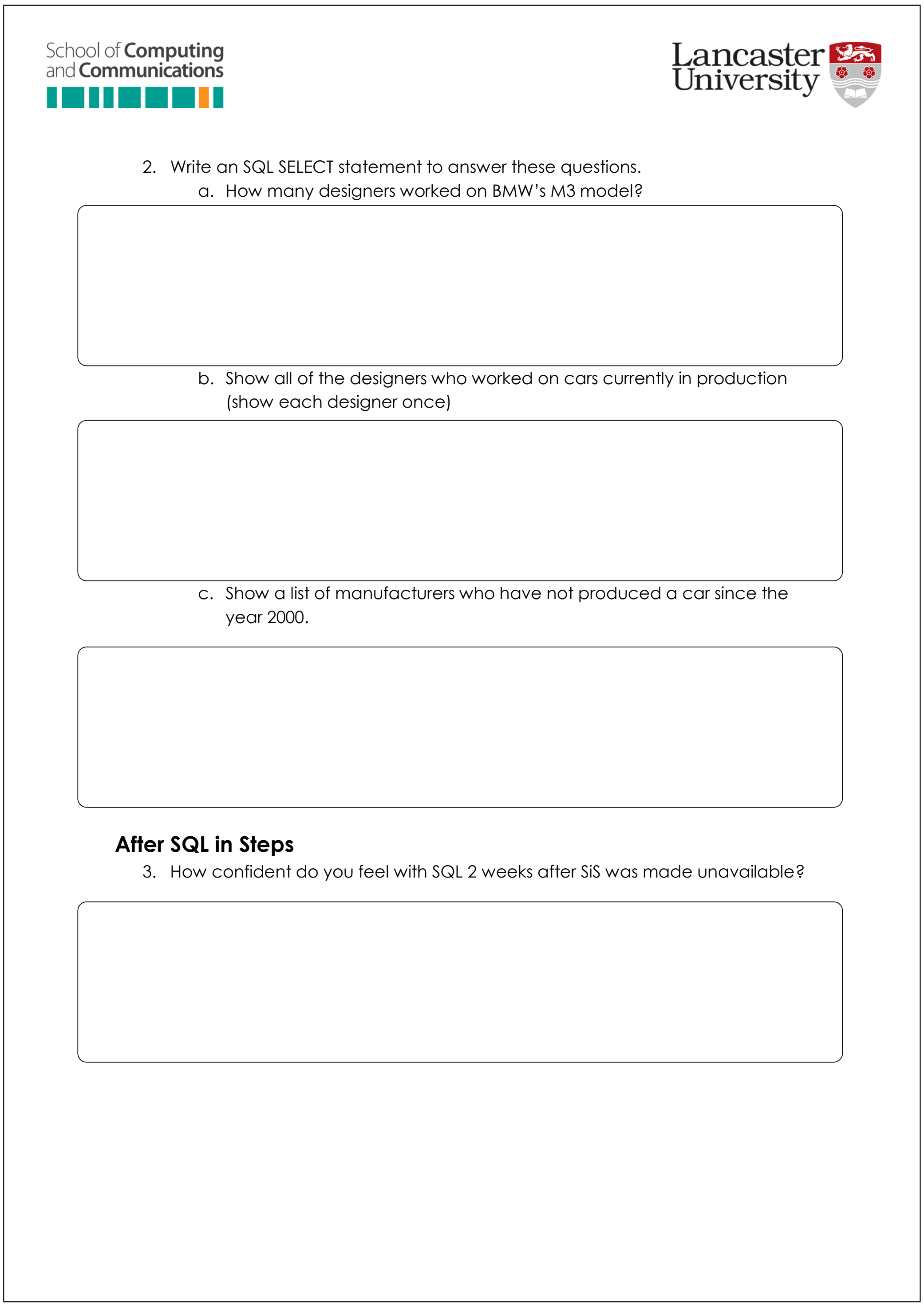
Answer the following questions in relation to the cars above database

1. Describe what the following SQL queries mean in English.

```sql
SELECT name, established
FROM manufacturer
ORDER BY established ASC;
```

```sql
SELECT manufacturer.name, COUNT(*)
FROM manufacturer INNER JOIN model
      ON manufacturer.id=model.manufacturer_id
GROUP BY manufacturer.id;
```

```sql
SELECT designer.name
FROM designer INNER JOIN designed
      ON designer.id=designed.designer_id
      INNER JOIN model
      ON model.id=designed.model_id
WHERE model.name='Enzo'
```



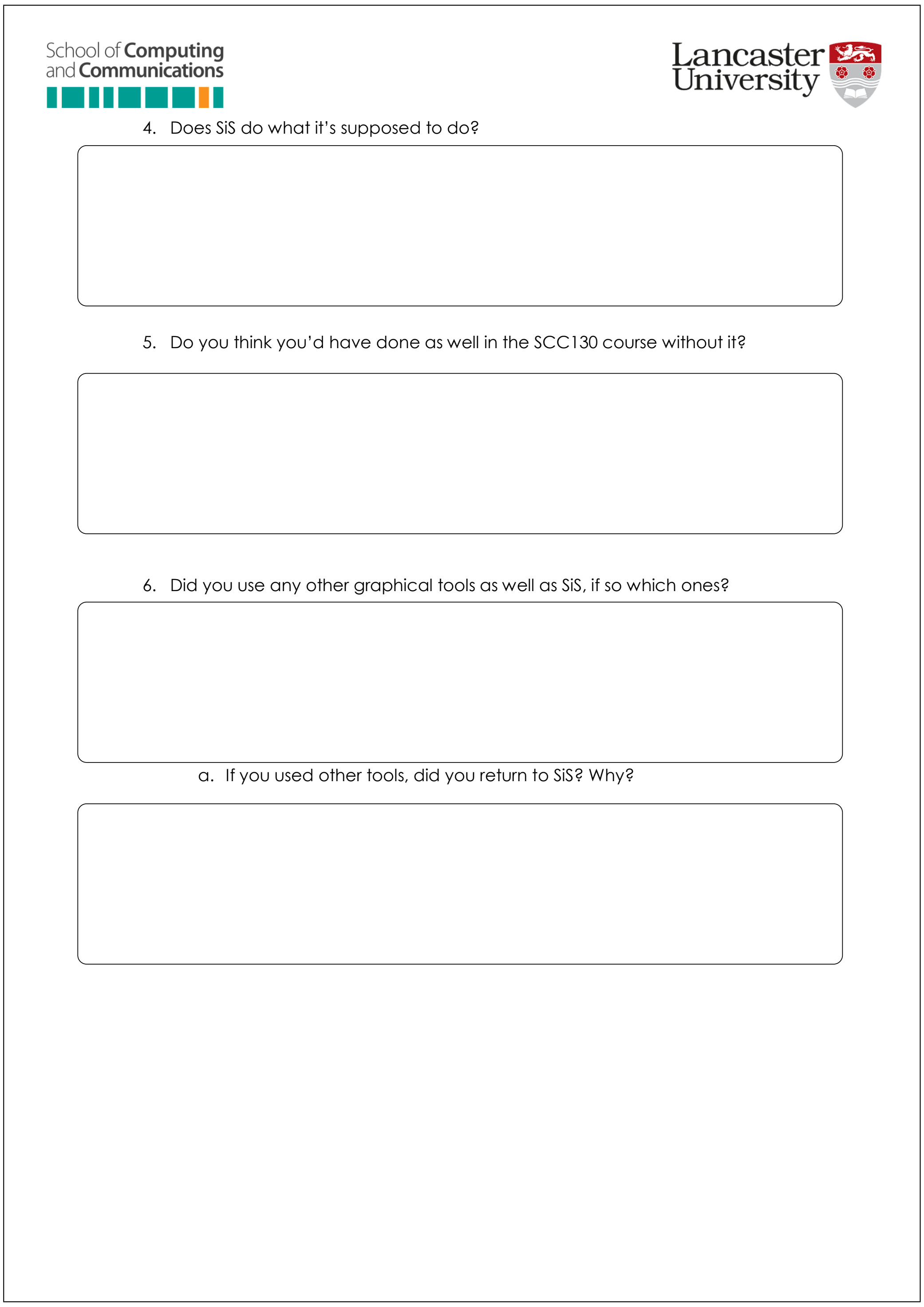
School of Computing and Communications

Lancaster University

2. Write an SQL SELECT statement to answer these questions.
   a. How many designers worked on BMW's M3 model?
   b. Show all of the designers who worked on cars currently in production (show each designer once)
   c. Show a list of manufacturers who have not produced a car since the year 2000.

After SQL in Steps

3. How confident do you feel with SQL 2 weeks after SiS was made unavailable?



School of Computing and Communications

Lancaster University

4. Does SiS do what it's supposed to do?

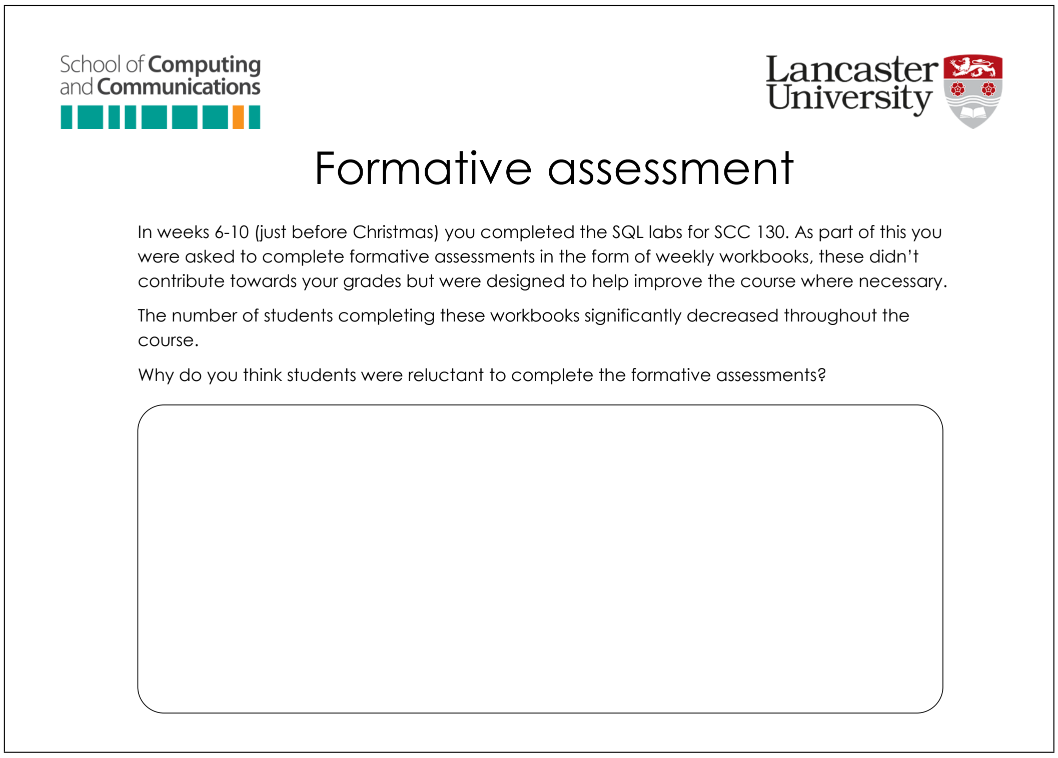
5. Do you think you'd have done as well in the SCC130 course without it?

6. Did you use any other graphical tools as well as SiS, if so which ones?

a. If you used other tools, did you return to SiS? Why?

C.1.4 Formative assessment question

After the completion of the study a random selection of 49 anonymous students were asked why they believed the number of students completing the formative assessments steadily decreased as the weeks progressed. The following is the questionnaire they were presented with:



School of Computing and Communications

Lancaster University

Formative assessment

In weeks 6-10 (just before Christmas) you completed the SQL labs for SCC 130. As part of this you were asked to complete formative assessments in the form of weekly workbooks, these didn't contribute towards your grades but were designed to help improve the course where necessary.

The number of students completing these workbooks significantly decreased throughout the course.

Why do you think students were reluctant to complete the formative assessments?

C.2 SiS configuration file

In Chapter 6 SiS is described, one of the features of SiS is the level of customisation available that allows it to appeal to a broad range of students.

C.2.1 SiS configuration

The following is the configuration file used during the evaluation described in Chapter 7.

```
[joins]
;Format joins explicitly (FROM/ON clause) or implicitly (WHERE clause)
explicitJoin = true
;Allow the use of full joins (note: not natively supported in SQLite - will translate to INNER JOIN
     operations)
fullJoin = false
```

```
;Allow the use of right joins (note: not natively supported in SQLite - will translate to LEFT JOIN
     operations)
rightJoin = false

[ui]
;Show a warning when the user attempts to navigate away from the page
redirectWarning = false
;Allow the input of a custom SELECT item, this is required to enable SELECT clauses such as SELECT a+1 FROM
     table;
custom_select = true;
;Show the option for nested queries
nested = false;
;Show the save button on the SQL translation
save_button = false;

[clauses]
;Show the WHERE clause panel
showWhere = true
;Show the GROUP BY clause panel - If set to false the HAVING panel will also be hidden, you cannot display
     the HAVING panel without the GROUP BY
showGroupBy = true
;Show the HAVING clause panel - showGroupBy must also be set to true
showHaving = true
;Show the ORDER BY clause panel
showOrderBy = true
;Show the LIMIT clause panel
showLimit = true

[database]
;The directory containing the database - this is relative to the base directory containing all the teaching
     SQL files (the parent of this directory)
sqlite_dir = db

[developer]
;If in debug mode will print out information to the console
debug = false
;Keep a log of the activity performed by the users
log = true
;The name for the log file
log_file = log.txt
```

C.2.2 SiS automatically generated help

The following are automatically generated examples for a simplified version of the movie database found in Appendix C.

C.2.2.1 SELECT

In the SELECT query you define the information you would like to see in your results, this can consist of a list of attributes and/or functions. The different attributes/functions you wish to see can be given aliases using AS.

With the following query we can show only the name column from the actors table (use SELECT * to display all columns within the tables).

```
SELECT name FROM actors;
```

C.2.2.2 FROM

The FROM clause is used to define the tables you want to extract information from. If you want to extract information from more than one table then you need a join. For example ... FROM table1 JOIN table2 ON table1.pk=table2.fk ...

With the following query we can extract all the information from the acts_in table.

```
SELECT * FROM acts_in;
```

C.2.2.3 WHERE

In the WHERE clause you must provide a Boolean expression which all records in your results must meet. This is not required but is a commonly used clause in the SELECT statement.

With the following query we can show only those rows in the actors table with a id value of 3 OR 1366382.

```
SELECT * FROM actors WHERE id = 3 OR id = 1366382;
```

C.2.2.4 GROUP BY

Grouping using the GROUP BY function can be used to collect your results according to a certain attribute, this enables you to apply aggregate functions to different groups of your results all at the same time.

With the following query we can show how many entries there are in the acts_in table for each movie_id.

```
SELECT movie_id, COUNT(movie_id) FROM acts_in GROUP BY
   movie_id;
```

C.2.2.5 HAVING

The HAVING clause can be used to apply restrictions to the aggregate functions used in a GROUP BY clause.

With the following query we can show the movie_id entries which have more than 4 entries in the acts_in table.

```
SELECT movie_id, COUNT(movie_id) FROM acts_in GROUP BY
   movie_id HAVING COUNT(movie_id) > 4;
```

C.2.2.6 ORDER BY

The ORDER BY clause can be used to display your results by one or more attributes in ascending or descending order.

With the following query we can order the actors table by the name attribute (use DESC instead of ASC to order in descending order).

```
SELECT * FROM actors ORDER BY name ASC;
```

C.2.2.7 LIMIT

The LIMIT clause can be used to only show a portion of the full results. If your query will return many results you can restrict this number using LIMIT. Note: it should not be used in place of functions such as MIN or MAX.

With the following query we can show 5 records of the actors table, starting at the 10th record.

```
SELECT * FROM actors LIMIT 10, 5;
```

C.3 Evaluation data

Data gathered throughout the evaluation of SiS is openly available from Lancaster University data archive.

C.3.1 Complete dataset

The anonymised student data used in the evaluation of SiS can be found at http://dx.doi.org/10.17635/lancaster/researchdata/80

C.3.2 SiS focus group transcription

A complete transcription of the focus group can be found at http://dx.doi.org/10.17635/lancaster/researchdata/64

Glossary

Application programming interface A set of rules that programs follow to communicate with each other. This includes web services and the use of a software library in another application.

attribute A single data field within a tuple.

cardinality The type of relationship between two relations in a relational database. These can take the form of one-one(1-1), one-many(1-∞) or many-many(∞-∞).

column *See attribute.*

explicit join When join conditions in an SQL SELECT statement are defined using ON in the FROM clause.

field *See attribute.*

IDREF A means to connect elements in an XML document. Allows elements to have parent and child elements, similar in functionality to relationships in relational databases.

implicit join When join conditions in an SQL SELECT statement are defined in the WHERE clause.

JavaScript Object Notation An attribute-value pair notation for representing data. Many REST interfaces provide responses in JSON format.

junction table A relation in a database used to facilitate a many-many relationship between two other relations. *See cardinality.*

normalisation The process of organising data within a relational database with the aim of reducing data redundancy.

relation A set of tuples that belong to the same domain. For example, in a movie database (such as that described in Appendix A) a relation might contain a series of tuples that describe production companies.

Representational State Transfer The use of GET, POST, PUT, DELETE requests sent over HTTP. REST interfaces are often used to programmatically manipulate datasets.

row *See tuple.*

table *See relation.*

tuple The information relating to a single entry in a table in a database.

Unified Modelling Language A standard means to visualise the design of a system (not specific to database applications).

Acronyms

ACID Atomicity, Consistency, Isolation, Durability.

API Application programming interface.

CAFTAN Context Aware Free Text ANalysis.

CLI Command Line Interface.

CMS Content Management System.

ER Entity Relationship.

GUI Graphical User Interface.

IDE Integrated Development Environment.

IoT Internet of Things.

JSON JavaScript Object Notation.

NLP Natural Language Processing.

NoSQL Not only SQL.

QBE Query By Example.

RDBMS Relational database management system.

REST Representational State Transfer.

SiS SQL in Steps.

SQL Structured Query Language.

SSQL Shorthand SQL.

UI User Interface.

UML Unified Modelling Language.

URM Universal Relation Model.

VQL Visual Query Language.

WIMP Windows, Icons, Menus and Pointer.

XML eXtensible Markup Language.