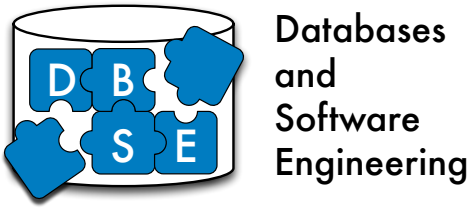
University of Magdeburg

Faculty of Computer Science



Master's Thesis

Designing a Database Schema for Survey Questions

Author:

Kurian John

February 24, 2022

Advisors:

Prof. Dr. rer. nat. habil. Gunter Saake

Department of Technical and Business Information Systems
Otto-von-Guericke-University Magdeburg

Dr.-Ing. David Broneske

Department of Infrastructure and Methods
German Centre for Higher Education Research and Science Studies

John, Kurian:
Designing a Database Schema for Survey Questions
Master's Thesis, University of Magdeburg, 2022.

Abstract

Surveys are the research methodology used to gather data from a predefined group of people about specific subjects. A survey has multiple survey questions related to the inquisitive topic. The survey respondents answer these questions to generate the survey response data, which can be analyzed statistically to derive useful information and insights. The surveys and their questions are important for the reproducibility and comparability within surveys of different years or between different survey projects. Hence, we need a database to store them permanently and access them when designing new surveys. However, the storage of survey questions in the database is a tedious task as the questions can be of different kinds without any fixed schema. Therefore, in this thesis, we design an SQL and a NoSQL database schema for the management of survey questions. Furthermore, both the schemas are compared against each other in terms of different parameters to find out the best-suited one for managing the survey questions.

Acknowledgments

This thesis work would not have been possible without the support of many people. First of all, I would like to thank my advisors Prof. Dr. Gunter Saake and Dr.-Ing. David Broneske for giving me the opportunity to write this thesis at the Department of Databases and Software Engineering.

I must express my very profound gratitude to Dr.-Ing. David Broneske for the supervision and support he has given me throughout this thesis work.

I would also like to thank Dr. Anja Gottburgsen and Rachid Laajouzi for their valuable inputs and comments on this thesis.

I am grateful to my parents and my wife for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

Finally, I am grateful to God for the good health and well being that was necessary to complete this thesis work.

Contents

List of Figures

List of Tables

List of Code Listings

1. Introduction

Surveys are the research methodology used to collect data from a predefined group of people about specific topics. The data collected from various survey respondents are then analyzed statistically to derive useful information and insights [Gla05]. The data is collected from the target audience via questionnaires that contain a set of questions related to the topic under investigation. The surveys could be conducted through different means such as online, offline, phone and interviews.

Each survey consists of multiple survey questions related to the survey's topic. The survey respondents answer these questions to generate the survey response data. Apart from the response data as the major outcome of a survey, also the survey questions are important for the reproducibility and comparability within surveys of different years or between various survey projects. For example, the questions of a particular survey might be useful in situations like conducting the same survey in the future, creating similar surveys or using it as a reference for the researchers who analyzes the survey response. Hence, the permanent storage of these survey questions is important for the survey research.

Database systems are one of the most popular choices for the storage and management of a huge amount of data [MK19]. Databases offer many features such as fast data retrieval, data sharing, non-redundant data storage, data security and easy maintenance for the efficient management of data. Therefore, databases would be a good option for the storage of survey data as well. SQL and NoSQL databases are the two different types of databases used for data management. SQL databases or relational databases store the data in the form of tables or relations. On the other hand, all the non-relational databases that implement a non-relational data model (e.g., document-oriented model, graph model, key-value pairs, column stores) for data storage are considered as NoSQL databases.

The storage of survey responses in the databases is often a straightforward task as the data is well structured. The data usually consists of the answers given by the survey respondents for the different survey questions. Such well-structured data could be easily managed using the databases. However, the storage of survey questions

in a database raises some problems. The survey questions are not well structured as compared to the survey response. The surveys can have different types of questions with different structures. A survey question can be open-ended, closed-ended or a combination of both [Gla05]. The closed-ended questions can have different ways of answering, such as multiple-choice, single choice, drop-downs or range selection. Furthermore, a question can have multiple sub-questions as well, and these sub-questions could be of different types as well. Therefore, the storage of survey questions in the database is a tedious task.

Goal of this Thesis

The goal of this thesis is to design an SQL and a NoSQL database schema for the management of survey questions at DZHW. Furthermore, both the schemas are compared against each other in terms of different parameters to find out the best-suited one for managing the survey questions. The major contributions that are made over the course of this thesis are discussed below:

- A detailed analysis of the survey questions at DZHW is conducted to identify all the business requirements, which would provide a clear and precise understanding of the database for managing the survey questions.
- A high-level conceptual data model is designed for the survey questions based on the outcomes of the analysis of the survey questions, such that it can be adapted to any of the required final databases. The conceptual data model is independent of any technical or implementation details of the final database.
- An SQL and NoSQL database schemas are designed for managing the survey questions by expanding the conceptual data model. The SQL schema is implemented in the MySQL database, and the NoSQL schema is implemented in the MongoDB database.
- Our evaluation and comparison of MySQL and MongoDB databases are carried out in terms of different parameters such as data loading times, storage space, query execution times, full-text search and selectivity analysis to find out the best-suited database for managing the survey questions.

Structure of the Thesis

The structure of the thesis is as follows:

- **Chapter 2: Background**
  The necessary background details required for the thesis is provided in Chapter 2.
- **Chapter 3: Related Work**
  In Chapter 3, several relevant research works related to this thesis work is discussed.

- **Chapter 4: Requirement Analysis and Conceptual Data Model**
  Chapter 4 aims at a detailed analysis of the survey questions at DZHW to identify all the requirements that would provide a clear and precise understanding of the database which is to be designed.

- **Chapter 5: Selection of the Databases**
  In Chapter 5, we compare different the SQL and NoSQL databases to select the best suited ones for implementing the database for managing the survey questions.

- **Chapter 6: Database Design**
  In Chapter 6, we design SQL and NoSQL database schemas for managing the survey questions by expanding the conceptual data model.

- **Chapter 7: Loading Survey Questions into Database**
  Chapter 7 discusses the process of loading the survey questions, present in the portable document format (PDF) to the designed databases using ETL workflows.

- **Chapter 8: Evaluation**
  In Chapter 8, we compare and evaluate the designed MySQL and MongoDB databases in terms of different parameters.

- **Chapter 9: Conclusion and Future Work**
  Chapter 9 is the final chapter in which the thesis work is summarized, and the possible future works are also discussed.

2. Background

This chapter discusses the necessary background details required to design suitable database schemas for holding the survey questions and also the important concepts of ETL Tools, which could be used to store the semi-structured survey questions into the designed database. Initially, we look into the surveys conducted at DZHW, followed by the concepts of SQL and NoSQL databases. Further, we discuss different design techniques for SQL and NoSQL databases. Finally, the concepts of ETL Tools are also explained.

2.1 DZHW Surveys

Deutsche Zentrum für Hochschul- und Wissenschaftsforschung (DZHW), funded by federal and state governments, conducts various surveys across Germany in order to collect data for research and analysis to provide different research oriented services for Universities across the country and in the field of science research[1].

The social surveys, graduate panel surveys, DZHW panel study of school leavers with a higher education entrance qualification surveys, scientists surveys and EUROGRADUATE survey are some of the surveys that are conducted at DZHW. These surveys are discussed in detail below.

Social Surveys

Social surveys are about the social and economic life of the students in Germany[2]. These surveys were conducted from 1951 to 2016 among the students who were registered at a higher education institution in Germany[3]. The survey contains questions regarding students' study-related aspects, social and demographic features, financial status, accommodation, time abroad for studies or study-related matters and time management between studies and employment.

[1] https://www.dzhw.eu/en/gmbh
[2] http://www.sozialerhebung.de/
[3] https://www.dzhw.eu/en/forschung/projekt?pr_id=460

DZHW Graduate Panel Surveys

DZHW graduate panel surveys are conducted to obtain information about the academic and career developments of university graduates[4]. These surveys are conducted among graduate students in Germany from the year 1989. The main topics covered by the surveys are students' academic careers, study conditions, career demands and opportunities and career changes and their associated issues.

DZHW Panel Study of School Leavers with a Higher Education Entrance Qualification Surveys

These surveys focus on the academic progress of school leavers who have a higher education entrance qualification from schools to higher education institutions and vocational education institutions[5]. These surveys are conducted among the students who have higher educational qualifications after school from 1976.

DZHW Scientists Surveys

DZHW scientists surveys are trend surveys conducted among teachers and researchers to learn about the influence of changes in the German science system on the academic research conditions[6]. These surveys are conducted from 2010 among the scientists (professors and research assistants) at German universities. The main topics covered by these surveys are scientists' working and research conditions, issues regarding current research policies and research priorities.

EUROGRADUATE Survey

The eurograduate survey was a pilot survey aimed at testing out the possibilities of a European-wide graduate survey[7]. The survey was conducted in 2018 among the students who were recently graduated from eight European countries such as Austria, Czech Republic, Germany, Greece, Croatia, Lithuania, Malta, and Norway. The aim of the survey was to learn about how happy are the students in their studies, how did they manage to sustain themselves during their studies, whether they travelled to other countries and what do they do after graduation.

Summary of DZHW Surveys

Most of these surveys are part of survey series which are carried out periodically and many among these are related to each others as well. As a result these surveys contain many common questions among them. However there is no centralized and non redundant storage system which could be used to store all these survey questions along with the relationships among the survey questions. Currently all the survey questions are stored independently without considering the relationship among them. Hence, a centralized and non redundant storage system would be very helpful for the researchers at DZHW for reference and future survey creation.

[4] https://www.dzhw.eu/en/forschung/projekt?pr_id=467
[5] https://www.dzhw.eu/en/forschung/projekt?pr_id=465
[6] https://www.wb.dzhw.eu/en/about/index_html
[7] https://www.eurograduate.eu/

2.2 Databases

A database is defined as a set of data or information stored according to a particular structure in a computer system [CK19]. The data is stored permanently and could be accessed by another person or a program to perform various operations on it such as insertion, deletion and updation. The data in the database is managed via a software called Database Management System (DBMS) using special query languages. Two different types of databases that are currently being used are SQL and NoSQL databases.

2.2.1 SQL Databases

SQL databases are often known as relational databases. As the name implies, relational databases store data in the form of relations or tables. As a result, an SQL database can be defined as a set of relations or tables with rows and columns where each row represents a data record or entity. Each row or data record could be uniquely identified by a column, called as primary key, containing unique values. Two tables in the database could be linked together by using foreign keys which is stored in both the tables [KK01].

The main advantages of relational databases when compared with other non relational databases are follows:

- SQL databases have an advanced and powerful yet simple querying language that allows for easy access of data.
- SQL databases usually adhere to ACID (atomicity, consistency, isolation and durability) properties which ensure data stability and security.
- SQL databases enables normalization of data, which prevents data redundancies.

2.2.1.1 Characteristics of Relational Databases

Dr. Edgar F. Codd, an IBM research scientist, have proposed 12 different principles that every relational database should follow [KK01]. A database must follow all twelve of these rules in order to become completely relational. These principles are generally known as Codd's rules for relational databases and are discussed further below.

1. All data and metadata stored in database should be in a table or relational format.
2. Each data element in the database should be accessible using a combination of table name, primary key and column name.
3. NULL values in the database should be systematically considered as "missing Information" in all the places.
4. Data which describes the database (metadata) should also be stored in the database as tables like regular data.

5. The entire database should be accessed by a common language. This language should be used for data definition, manipulation and other transaction management operations on data.

6. All the changes made in the database tables should be automatically reflected on the views which are defined on the tables.

7. High level insertion, updation, selection and deletion queries should be supported by the database.

8. Physical independence of data stored in the database is desired. Physical data independence is defined as the ability to make changes in the physical schema of the database without affecting any schemas in the higher levels such as conceptual schema and external views.

9. Logical independence of data stored in the database is desired. Logical data independence is defined as the ability to make changes in the conceptual schema of the database without affecting any schemas in the higher levels such as external views or application programs.

10. The database should be independent of all the applications which uses it. All the integrity constrains should be incorporated within the database itself.

11. Distribution independence of the data stored in the database is desired. For the end users data should appear to be stored in a central location even if it is distributed.

12. Operations which provide access to the low level records(row level processing) in the database should not pose any threat to the integrity and security constraints.

2.2.1.2 Relational Database Model

Relational database model war first proposed by Dr. Edgar F. Codd in the year 1969 in his work "A Relational Model of Data for Large Shared Databanks" [Cod70]. This new model was initially introduced as a mathematical model based on set theory and first order predicate logic to store huge volumes of data.

In relational model, data is stored in the form of tables [Her13]. The tables are also called relations. Each data record, also known as tuple or entity, is stored as a row in the relational model and is identified by a column which contains unique values. This unique field is called primary key. Each column of the table represents an attribute of the tuple.

Figure 2.1 shows an example table in the relational model. The relation, Employee Relation, stores the data of two employees(2 tuples). Each employee has a name and salary and is uniquely identified by using an EmployeeId (3 attributes with EmployeeId as the primary attribute).

In relational model, data should be normalized and is stored as potentially multiple tables to avoid data redundancy. Thus in a relational database, information about

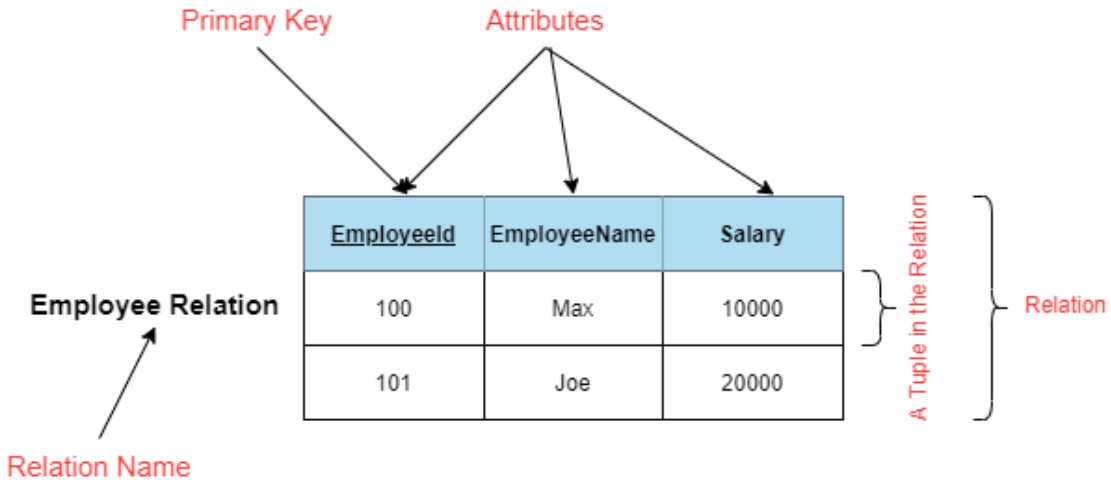


Figure 2.1: Example Relation for Relational Database Model

a single entity could be stored in different tables and these different tables are connected or related together using common attributes which are present in both the tables[Her13]. The relationships between tables in the relational model could be classified into three types as follows:

- One-to-one relations: Two relations are said to be one-to-one related if a record in the first relation is related to only one record in the second relation and vice versa. Such relationships are not very common and if they exists then the two records should to stored together as a single record in a single relation unless there exist some security or other organizational constraints.
- One-to-many relations: Two relations are said to be one-to-many related if a record in the first relation is related to one or more records in the second relation, whereas each record in the second relation should be related to only one record in the the first relation. Such relationships are very common and are implemented using the primary keys and foreign keys.
- Many-to-many relations: Two relations are said to be many-to-many related if a record in the first relation is related to one or more record in the second relation and vice versa. In case of many-to-many relations, the relation is converted into two one-to-many relations by introducing a third table between the two existing tables.

SQL Databases offers many advantages like efficient management of structured data, high data integrity, atomicity, normalized storage of data and data consistency. On the other hand, the SQL databases are not well suited for handling semi-structured or unstructured data, lacks horizontal data scalability, and the fixed schema of the SQL databases might sometimes create problems when the application requirement changes. Therefore, SQL databases are mainly used when the data has a fixed structure, and feature like data non-redundancy, data integrity and data security is important.

2.2.2 NoSQL Databases

NoSQL stands for 'Not Only SQL' [HHLD11]. It is a common term used to describe non relational databases. Thus a database could be called as NoSQL database if the the data model used to store the data in the data is non relational, that is the data is not stored as tables or relations and is not managed using SQL.

2.2.2.1 Characteristics of NoSQL Databases

Since any non relational database falls under the category of NoSQL databases it is very difficult to list out specific properties that are followed by all the NoSQL databases [HHLD11, MBNG15]. However, some of the high level characteristics followed by most of the NoSQL databases are discussed below [SMP14].

- Flexible Schema: Even though NoSQL databases have different data models, all of them allow to have a flexible schema. This enables easy modification of the databases according to the timely change in the business requirements.
- Horizontal Scalability: NoSQL databases are scaled out horizontally which makes it more suitable than relational databases.
- Replication of Data: In order to achieve data redundancy and load balancing, many NoSQL systems store copies of the same data at different sites. However this may affect the consistency of the stored data [GL12].

2.2.2.2 Classification of NoSQL Databases

As discussed in Section 2.2.2.1, NoSQL databases implements different data models. Based on these data models, NoSQL databases can be mainly classified into four different types.

1. **Key-Value Stores:** Key-value databases store data in collections in the form of key-value pairs. Data is stored as values and each value contains a key which could be used to access the data [See09]. All the keys in a collection are unique and thus could not be duplicated in order to prevent access conflicts. The main advantages of key-value stores are that they implement a simpler data model as compared to its counterparts and also offers fast read and write capabilities as compared to relational databases. However, they do not provide the necessary database characteristics such as atomicity and data consistency. Another shortcoming is that the data search using values stored in the collections are very difficult or sometimes even impossible. The data could be only accessed using the keys [OBALB15]. Redis, Amazon DynamoDB, Microsoft Azure Cosmos DB and Memcached are some of the examples of key-value databases.
2. **Document Oriented Databases:** Document databases store data in the form of documents. Depending on the type of document used, the data model may differ. But generally, each of these documents has a unique key which could be used to identify the documents. Different document formats which are being

used among document oriented databases are JSON (JavaScript Object Notation), XML (eXtensible Markup Language) and BSON (Binary JavaScript Object Notation) [Sha15]. Document databases store the data in a format which is very close to the data format used in the applications thereby reducing the application development efforts. In a document, data is stored as key-value/array pairs and may even contain other embedded documents as well. Unlike key-value stores, document databases allow search operations using keys, values or even metadata and hence they are more flexible [OBALB15]. MongoDB, Couchbase, Firebase Realtime Database and CouchDB are some of the examples of popular document databases.

3. **Column Oriented Databases:** Column oriented databases are also called wide column stores or extensible record stores. Such databases store data in columns. Like relational databases, the data is stored as tables, but the table partition is done vertically and each column is stored separately [Sri17]. Like the document oriented model, this model is also derived from the key-value model by incorporating the concept of columns into the key-value model architecture. Column oriented databases offer high performance in case of aggregation queries since the data is stored as columns and thus are mainly opted for data warehousing and other analytical applications. Cassandra, HBase, Datastax Enterprise and Microsoft Azure Table Storage are some of the examples of column oriented databases.

4. **Graph Databases:** Graph databases are used to store data which could be represented in the form of graphs. Graph databases implement the graph model and store data in the form of graphs with nodes, edges and relations. In a graph database a data entity is stored as a graph node. Two nodes are connected together by an edge if there exists a relationship between the entities stored in those node and this relationship is represented by an edge label [kK15]. The main advantage of the graph database along with its highly flexible schema is that it allows the users to not only store and analyse the data but also the relationships between different data entities as well [OBALB15]. Neo4j, Virtuoso, ArangoDB and OrientDB are some of the examples of graph databases.

NoSQL databases offers many advantages such as scalability, fast read and write access and also offer a flexible schema as compared to the traditional SQL databases and are suitable for applications that deal with huge volumes of data. On the other hand they also have some disadvantages as well like lack of transactions, slower or no join operations, data redundancy and data duplication. Therefore, NoSQL databases are well suited when the data is semi-structured or unstructured and the schema flexibility is important the data model.

2.3 Database Design

Database design may be defined as the process of developing a suitable design for representing the data in the database by analysing the problem statement [Let15].

The problem statement includes the scope of the problem along with all the requirements as well as the constraints of the data being analysed. The database is then developed using this design.

Database design is a very important step in application development. It is very crucial for the stability, correctness and integrity of the stored data [Her13]. A false or improper design of the database might results in incorrect data retrieval which would jeopardize the entire application. On the other hand a good and reliable design would makes the database easy to manage and it also make the data retrieval as well as the application development much more easier.

2.3.1 SQL Database Design

The traditional way of designing relational databases includes three steps: requirement analysis, data modeling and data normalization [Her13].

2.3.1.1 Requirement Analysis

Requirement analysis is the primary step in designing a database. It involves collection and analysis of all the business and information requirements of the problem. This is done to get a clear and complete idea about the database which is to be designed along with its practical uses.

2.3.1.2 Conceptual Data Modeling

Data modeling involves building an initial data model of the database using a data modelling technique. This model is basic and only gives a general understanding about the data. Different data modelling techniques which could be used in this step are Entity Relationship (ER) modeling [SEP95], Object role modeling [BGJ03] and UML modeling [EJN01].

2.3.1.3 Data Normalization

Data Normalization is the process of splitting of large tables into small tables to avoid data redundancy and data duplication so that the data manipulation operations on the data would become easier and error free.

2.3.2 NoSQL Database Design

NoSQL databases uses different data models and these data models offer high schema flexibility as well. So, like SQL databases, there is no fixed database design strategy which could be used to design NoSQL databases [ZBM20].

Over the years many researchers have proposed several design methodologies for NoSQL databases. For example NoSQL Abstract Model (NoAM) is a design technique which is used to design different types of NoSQL systems discussed in Section 2.2.2.2 except Graph databases [BCAT14]. NoAM creates an abstract data model independent of the underlying database. However, the model is too abstract that it does not focus on any of the main implementation characteristics of the the underlying database. Similarly many other modeling techniques were also proposed

over the years such as NoSQL Schema Evaluator (NoSE) [MSAL16] and Model-Driven Framework for NoSQL Database Design (Mortadelo) [dlVGSB+18]. However, none these works were able to produce a concrete modelling technique which could be used to design NoSQL databases in general due to their flexible schema and diverse data models.

After analysing the above techniques we have decided to follow a simple database and use-case specific methodology to design the NoSQL database used in this thesis work. It includes the requirement analysis (Section 2.3.1.1), high level data modelling (Section 2.3.1.2) and then designing a physical schema of the database according to the data model implemented by the selected NoSQL database and the use-cases of the system which is to be build. The processes of requirement analysis and high level data modelling could be done in the same way they are carried out for SQL database design as these steps only aim at gaining a better understanding about the problem and data without considering any implementation aspects of the database.

2.4 ETL

ETL stands for extract, transform and load. ETL may be defined as the process of extracting data from different sources such as flat files, databases or APIs and load the data into desired locations such as data warehouses, databases or flat files after transforming the data into the desired format. ETL is generally used for data warehousing. Data warehouses are a central storage system which stores the integrated data from various data sources [Vas09, KSS14]. Data warehouses are mainly used for analytical and reporting tasks on the collected data. The ETL process was introduced to aid the population of data warehouses with data from multiple sources by reducing the development efforts.

2.4.1 ETL Process

ETL process involves three steps for managing data: Extraction, Transformation and Loading [WP15]. Figure 2.2 shows the basic workflow of an ETL process.

2.4.1.1 Data Extraction

Data extraction is the first step in the ETL process. It involves the extraction or collection of data from different sources. Different data sources include flat files, APIs (Application Programming Interface), databases or could be even generated by the software itself. The loaded data is moved into a staging area for the next step. The staging area could be considered as an intermediate storage area for the loaded data. The data in the staging area is used in all the future data processing steps and the intermediate data is also stored in this staging area.

2.4.1.2 Data Transformation

In the data transformation step, the loaded data is processed and transformed according to the business requirements by using different operations. Different operations which could be done during the data transformation involves splitting or merging of data, data aggregation, filtering of data, cleaning of data, data transpose operations, data verification and data validation.

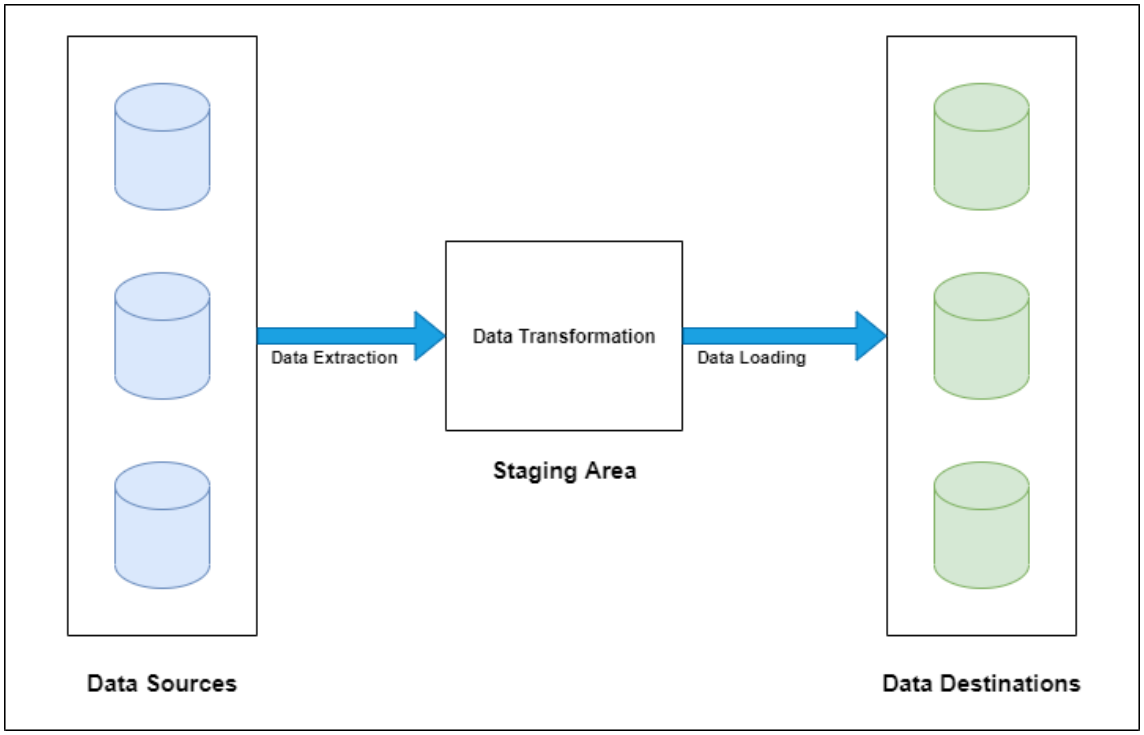


Figure 2.2: ETL Workflow

2.4.1.3 Data Loading

In the third and final step, the transformed data is loaded or stored into the preferred destinations as per the business requirements. Different data destinations include flat files, APIs, databases and data warehouses.

2.4.2 ETL Tools

ETL tools follow the ETL process for the fast processing of huge volumes of data. ETL tools are used to collect data from different sources and process it effectively to convert it into the desired format and finally storing the processed data in a location preferred by the users [MS15]. Many ETL tools are GUI based and offer different components for fast and efficient processing of huge volumes of data. The GUI based tools also reduce the development effort by limiting the hand coding.

Talend Open Studio, Informatica Power Center, Microsoft-SQL Server Integrated Services, Oracle Data Integrator and Xplenty are some of the ETL tools that are currently available in the market today. In this thesis work, we use the Talend Open Studio for the data processing operations as it is one of the most widely used open-source ETL tools that are being used today in many applications. The main features of the Talend Open Studio are discussed in the following section.

2.4.3 Talend Open Studio

Talend open studio is a GUI based open source ETL tool that is used for creating efficient data integration processes [Bow12]. The tool has more than 800 connectors or components for carrying out different operations such as connecting to different

databases, loading data from multiple sources, transforming data, writing data to multiple data sources and defining complex data integration operations. Since it is a GUI based tool that generates the code automatically, it is easy to learn and is suitable for both experienced as well as less experienced developers.

2.4.3.1 Talend Open Studio Environment

The common elements of the Talend Open Studio such as workspace, project, job, repository, design workspace, palette and configuration tabs are discussed in this section.

- **Workspace:** Workspace is a directory in the computer which contains all the Talend open studio projects.
- **Project:** A project in Talend Open Studio is a collection of several jobs.
- **Job:** A Talend job can be defined as the collection of several components that are connected. Each job performs a particular data integration process.
- **Repository:** The repository window is present in the top left corner of the Talend open studio. All the jobs present in the current projects are listed in this window.
- **Design Workspace:** It is the main window at the centre of the Talend open studio. The developers place different Talend components in this window and configure them to create the required job.
- **Palette:** The palette window contains all the components available in the Talend open studio. This window is present in the top right corner of the application. The developers can search the required components from the palette and place them in the design workspace to use them in the job.
- **Configuration Tabs:** The configuration tabs are mainly used to configure the individual components in the design workspace. A component in the design workspace can be configured in the configuration tab by selecting it. These are present at the bottom of the Talend open studio application.

The repository, design workspace, palette and configuration tabs are shown in Figure 2.3.

2.4.3.2 Talend Open Studio Components

A component or connector in Talend can be defined as the basic functional block which performs a specific operation. For example, the *tMysqlInput* component is used to read data from the MySQL database and the *tFilterRow* component filter the input data based on certain conditions. All the components are available in the palette window in the Talend open studio. The developers can select different components from the palette and place them in the design window to use them in a Talend job. Talend open studio has more than 800 components in total[8]. Some example components are shown in Figure 2.3.

[8]https://www.talendforge.org/components/index.php

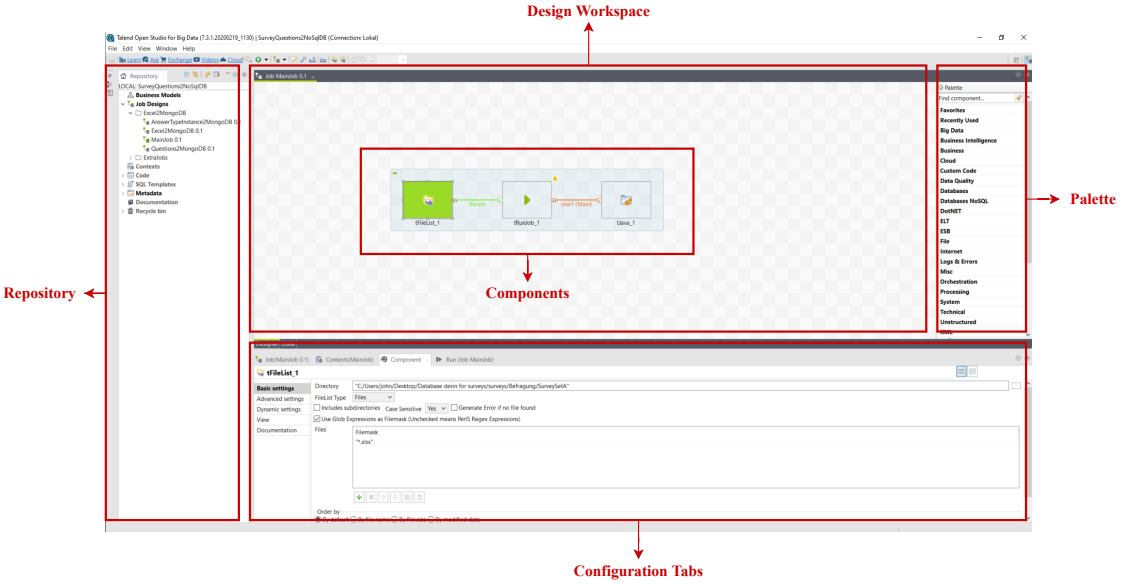


Figure 2.3: Talend Open Studio Elements

2.5 Summary

In this chapter, we provided the necessary background details required for this thesis work. Initially, a brief introduction was given to the different surveys conducted at DZHW. Further, we discussed the concepts of SQL and NoSQL databases, such as their characteristics and corresponding data models. Additionally, the SQL and NoSQL database design processes were also explained. Finally, we discussed the important concepts of the ETL process and also introduced Talend Open Studio, which is an open-source ETL tool.

3. Related Work

This chapter discusses several relevant research works related to this thesis. Some of the concepts discussed in these have aided in the successful completion of this thesis work.

[PWJD03] presents a database model for dynamic online web surveys. The survey questions are divided into multiple components so that each component can be represented as a database table record. Different independent question components are combined to form a complete question. The same question component can be used in multiple survey questions as well. However, this paper only provides limited information regarding the representation of survey questions in the database.

The goal of this thesis is to design and compare an SQL database and a NoSQL database for the management of survey questions at DZHW. In other words, we can say that we need to store the unstructured survey questions in a database. Therefore, we look into the different ways in which unstructured data can be stored in relational databases. Since NoSQL databases are designed exclusively for handling unstructured data, we do not investigate further into this topic. Different ways in which unstructured data can be permanently stored using relational databases are following:

3.1 Unstructured Data in Relational Databases

Relational databases are primarily designed for structured data. Storing unstructured data in SQL databases is a difficult task mainly due their fixed data model. However, over the years, many researchers have tried to store unstructured data in relational databases. Some of such works are discussed in this section.

[AM10] presents a comparative analysis of three distinct methods for storing unstructured data in databases. In the first method, unstructured data is stored inside the database as BLOB objects. The large files in the database cause speed and storage concerns in the database, which is one of the main disadvantages of this technique. The second option proposes storing unstructured data outside the database and associating these files to the database by recording their position or URLs. However,

the externally stored files are not part of the database transactions. The final and most effective technique is to save unstructured data using a novel hybrid storage method. This hybrid approach maintains unstructured data files in the file system outside the database, but guarantees that they are consistent with database transactions. However, the above-discussed methods only focus on storing the files of unstructured data. They fail to handle the cases where one needs to access a part of data stored in one of the files without loading the entire file.

[YAOI13] presents a detailed analysis of three different methods to handle unstructured data using relational databases. They are discussed as follows:

3.1.1 Database Schema

The first technique defines a database schema that can incorporate the unstructured data and then extract and map different identified entities from the unstructured data records into the database. For example, [MS06] proposes a system that could be used to store text records like citations into relational databases by using machine learning models. Initially, a structure could be inferred for the text records and the relevant entities are identified from this structure. Then, a relational database is designed using these identified entities. The text records are analysed using machine learning models to extract the entities from them. Each of these extracted entities is then mapped to the corresponding columns in the database and the corresponding relationships between the different columns are also defined. This method is really useful for data records from which a structure could be inferred.

3.1.2 Alternative Data Model

The second technique proposed by [YAOI13] suggests using new alternative data models to handle unstructured data. [LL10] proposes a new data model called tetrahedral data model for managing unstructured data. According to the tetrahedral model, each data record is decomposed into four different components: basic attributes that are common to all kinds of unstructured data, semantic attributes which describe the data, low level attributes that are extracted from the data and raw data containing the actual data. Relationships are also defined among these components for fast access to data.

3.1.3 SQL Queries

The third way of managing unstructured data is to use the SQL queries to access data stored in text databases [YAOI13]. This method is called query based approach and usually require an intermediate system to extract information from the unstructured data record according to the structured input query. QXtract [AG03] is a system that implements this methodology. QXtract is used to retrieve documents from large text databases from trained queries. The system learns the database search queries from the user input of some example tuples. After extracting the best-fitted documents from the learned queries, these documents are further processed by an information retrieval system to generate the final relation or table with the values corresponding to the user input.

3.2 Hybrid Approach

Sometimes the data can contain both structured and unstructured parts or some portion of the unstructured data could be converted into a structured format. For example, a video file may have several attributes such as name, file type, size, duration and description along with the actual file. All the attributes could be considered structured information, and the video file could be considered as unstructured information. In this case, a hybrid approach could be employed to store the data by combing SQL and NoSQL databases. The SQL database accommodates all the structured data whereas the unstructured information is managed by the NoSQL systems. For example, [SMP15] proposes a hybrid approach, hybrid database architecture (HDA), for storing unstructured data using MongoDB. The structured data is stored in a relational database and all the unstructured data is stored as files in MongoDB. The GridFS feature of MongoDB is used for storing the files. The metadata of the files in MongoDB is stored in the relational database. In this way, the connection is established between the two databases. For selecting a file, the relational database containing the metadata or details regarding the file is searched first, and once the file is identified, the corresponding file is selected from the MongoDB database.

3.3 Summary

In this chapter, we discussed several related works to this thesis. Initially, we discussed a database model which manages dynamic online web surveys. Further, we discussed the different ways in which unstructured data can be stored using relational databases as the survey questions can be considered as unstructured data. The first method involves storing the unstructured data in relational databases by designing a database schema for the unstructured data or by using an alternative data model or by using SQL queries to access data stored in text databases. The second method involves storing the data using a hybrid approach using SQL and NoSQL databases.

4. Requirement Analysis and Conceptual Data Model

Requirement analysis is the primary step in designing a database. In this step, the features, purpose and possible constraints of the database are identified and explained clearly so that a correct, complete and definite database design could be developed in the subsequent steps.

This chapter aims at a detailed analysis of the survey questions at DZHW. To this end, we analyse different survey questions in detail to collect all the business requirements, which would provide a clear and precise understanding of the database which is to be designed to store the survey questions. Furthermore, a conceptual data model is also designed for the survey question, which would aid the database design in the future.

Initially, we analyze a survey as a whole to understand the structure of the survey. Furthermore, we analyze different questions in the survey to understand the general structure of the survey questions and to identify different types of questions, answers and some special cases that exist in the survey. Based on this analysis, we then identify and list out all the requirements for the database design. Finally, we design a conceptual data model using the outcomes of the requirement analysis.

4.1 Analysis of the Survey

For designing the database schema, we analyze the questions from the survey, "Die Studierendenbefragung in Deutschland"[1]. This survey is the most extensive one carried out among the students in Germany. It is a combination of different surveys carried out separately in the past such as the Social Survey, the Student Survey, the EUROSTUDENT survey and the Studying with Impairments survey. The social survey aims for studying the social and economic conditions of the students. The Student Survey focus on the study conditions of the students. The EUROSTUDENT

[1] https://www.die-studierendenbefragung.de/en/for-higher-education-institutions

survey is conducted to understand the conditions of students in all the 28 European countries. The Studying with Impairments survey is carried out among the students to understand their academic conditions and challenges of the students with any impairment that might affect their studies.

The main focus areas of the survey, "Die Studierendenbefragung in Deutschland", include social, economic and political situations of the students, study conditions and challenges of the students, career plans and options of the students and health situations of the students that might affect their studies. The data collected from the surveys are regularly used for educational reporting. Furthermore, it is also used by various educational institutions across the country for their quality management. The collected data is also published in the Research Data Centre of DZHW for research purposes.

The survey, "Die Studierendenbefragung in Deutschland", contains different modules which address different categories or subtopics in the survey. For example, the module called "Kulturelle Rahmenbedingungen" contains questions about the cultural conditions of the students and the module called "Studierende mit Kind" contains questions to collect details about studying with children. Similarly, each of the survey modules contains different questions corresponding to the category or subtopic it addresses. The survey contains nineteen different modules in total. Figure 4.1 shows the overall structure of the analyzed survey.

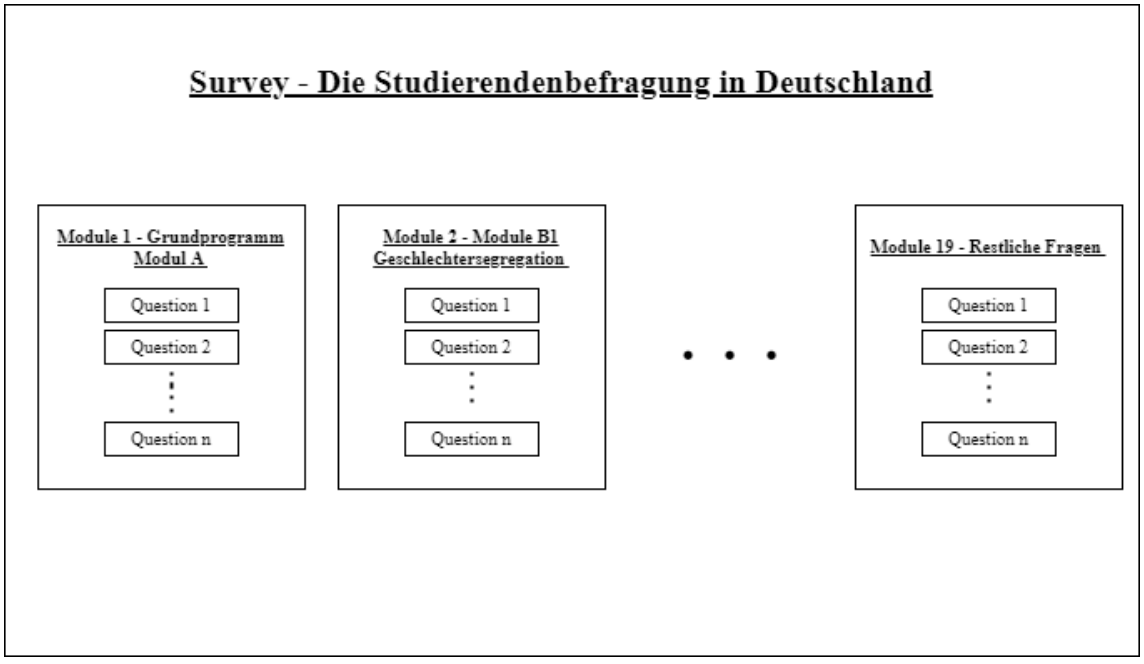


Figure 4.1: Structure of the Survey

Since the analyzed survey is a collection of different surveys conducted separately in the past, it is very important to identify and represent the relationships among them. The new survey contains many questions that were already present in the past surveys. Thus these shared questions should be stored non redundantly in the database. Moreover, the relationship between various surveys via such shared

questions should be well defined as well. Furthermore, the database which is to be designed should take into account the structure of the survey as well.

4.2 Analysis of the Survey Questions

After analyzing the survey as a whole, in this section, we consider the survey questions that belong to different modules of the survey.

4.2.1 Structure of Survey Questions

A survey question mainly consists of two parts: the question part and the answer part. An example of a simple question is shown in Figure 4.2. The question part contains a mandatory main question part and optional sub-questions. The question in Figure 4.2 only consists of the main question. The answer part varies depending on the type of answer. However, in general, an answer part contains the information regarding the type of answer and zero or more options/choices associated with the answer. The respondents of the survey have to select one or more options as the answer to that particular question. Sometimes instead of selecting options, the answer has to be given as free texts as well.

The question in Figure 4.2 has an answer part with the answer type as a "single choice type" (only one could be selected from a group of choices) and has two choices/options associated with it.

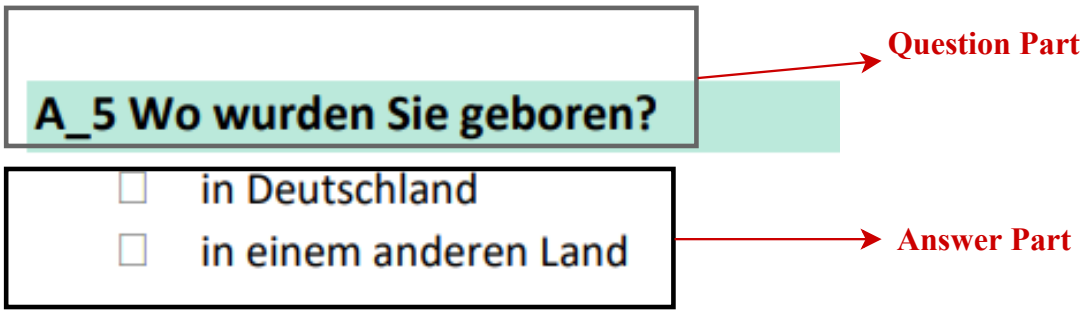


Figure 4.2: Structure of a Question

4.2.2 Classification of Questions

The analysed survey has different types of questions. Depending on how the question part is structured, the survey questions can be classified into different groups. The questions were analysed and grouped and each one of these groups was given a unique name so that they could be easily referenced in the subsequent sections and chapters of this thesis. Different groups of questions are level 0 questions, level 1 questions, level 1 grouped questions, level 0 multiple questions. They are discussed below:

4.2.2.1 Level 0 Questions

Level 0 questions are simple questions with only a main question part. This main question has an answer part associated with it as well. Such questions do not have any sub-questions. Figure 4.3 is an example of a level 0 question with a main question and a corresponding answer part.

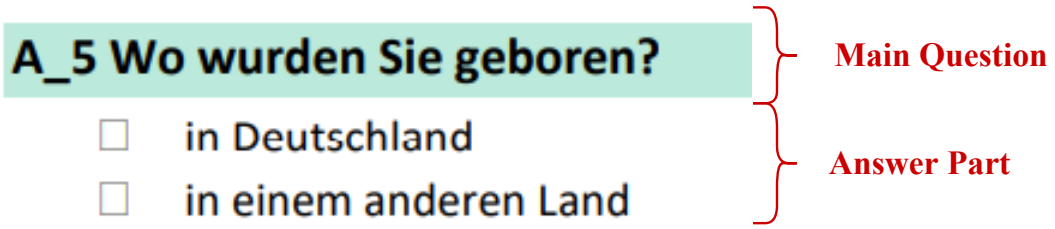


Figure 4.3: Example of a Level 0 Question

4.2.2.2 Level 1 Questions

Level 1 questions are survey questions with sub-questions. Such questions have a main question and this main question has one or more sub-questions. Each one of the sub-question has an answer part associated with it. Figure 4.4 is an example of a level 1 question with a main question having two sub-questions with corresponding answer parts. A level 1 question can have any number of sub-questions ranging from zero to n number of sub-questions. In the analyzed survey the highest number of sub-questions which belonged to a question was twelve.

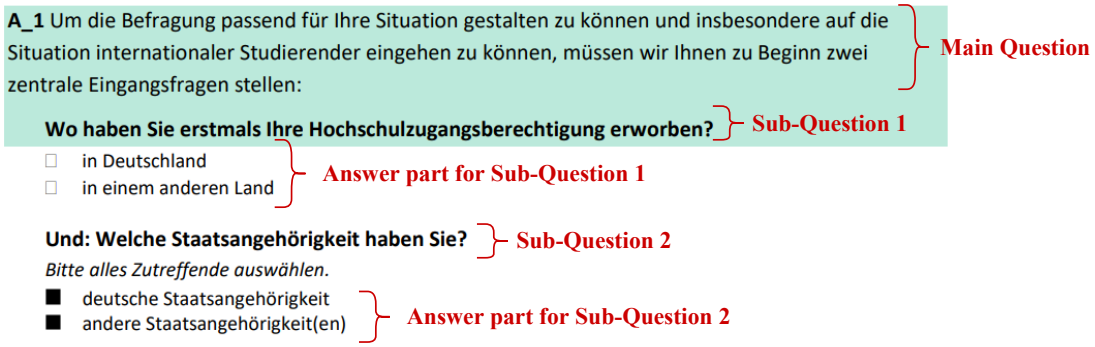


Figure 4.4: Example of a Level 1 Question

4.2.2.3 Level 1 Grouped Questions

Level 1 grouped questions are the survey questions with sub-questions where the sub-questions are arranged in different groups. Such questions have a main question part and this main question part has one or more sub-question groups and each of these groups has one or more sub-questions in them. Like the level 1 questions,

each one of the sub-questions has an answer part associated with it. Figure 4.5 is an example of level 1 grouped question with a main question having twelve sub-questions which are grouped into three different groups. However, there is no limit to the number of groups. A level 1 grouped survey question can have any number of sub-questions and these sub-question should be grouped into at least two groups as well.

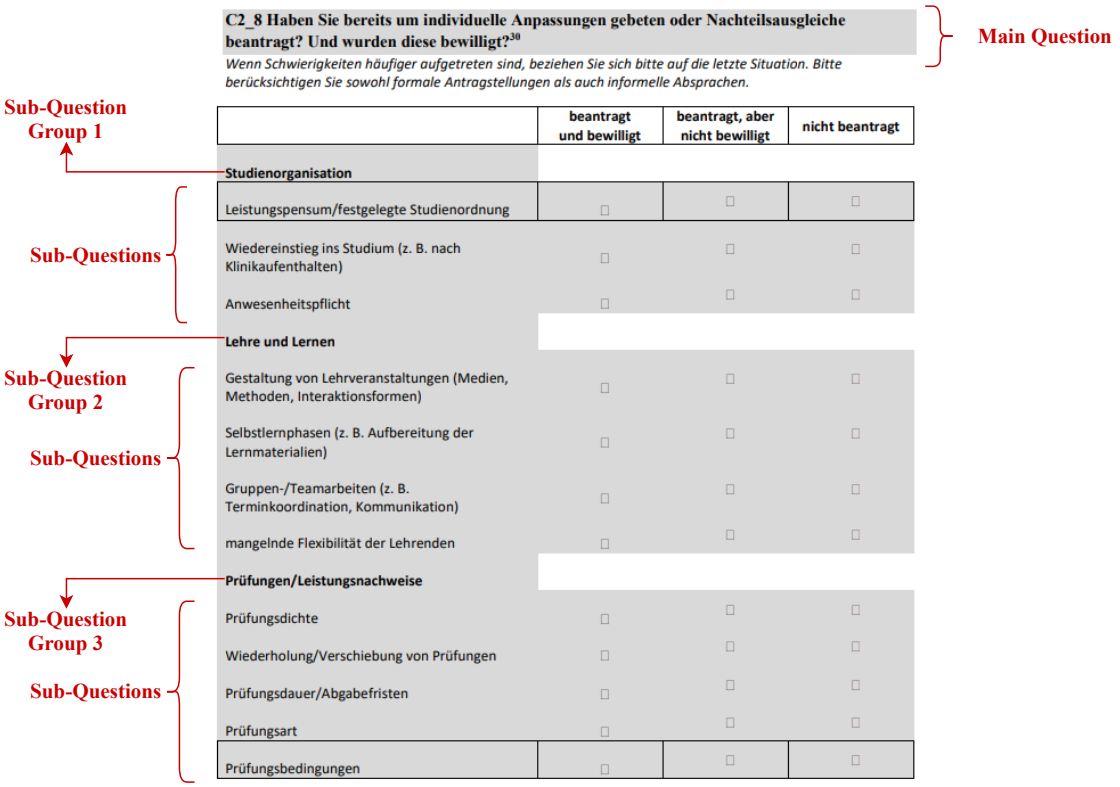


Figure 4.5: Example of a Level 1 Grouped Question

4.2.2.4 Level 0 Multiple Questions

Level 0 multiple questions are those survey questions that contain more than one level 0 question under the same question number. A single question has multiple main questions. A level 0 multiple question should have at least two main questions. Figure 4.6 is an example of level 0 multiple Question with three main questions with corresponding answer parts under one question number.

Even though, these are the only identified question groups in the analysed survey, the database schema which is to be designed should not only accommodate these identified groups but also should accommodate other possible questions which can be created by combining two or more of these question groups or by extending the structure of an existing question group. For example, by combing level 1 questions and Level 0 multiple questions we could create a new question type that contains multiple main question parts and each of these main question have multiple sub-questions. A structure for such a question is shown in Figure 4.7. Another example is to extend the structure of level 1 questions so that a new question type is created in

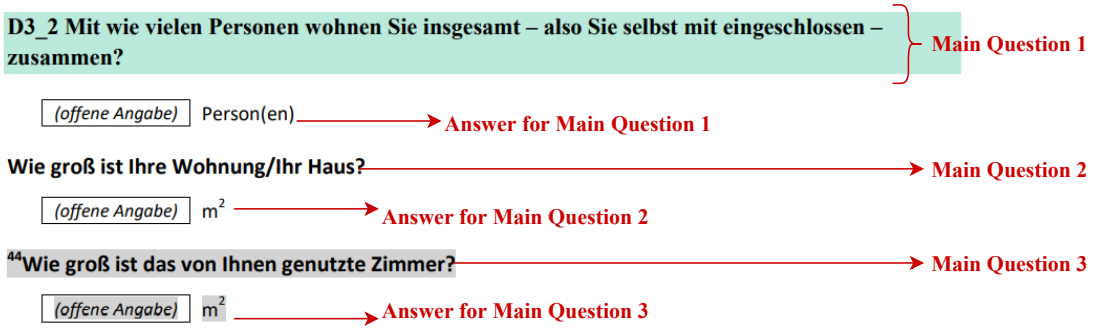


Figure 4.6: Example of a Level 0 Multiple Question

which the question contains multiple sub-questions and each of these sub-questions can have multiple sub-sub-questions as well. A structure for such a question is shown in Figure 4.8.

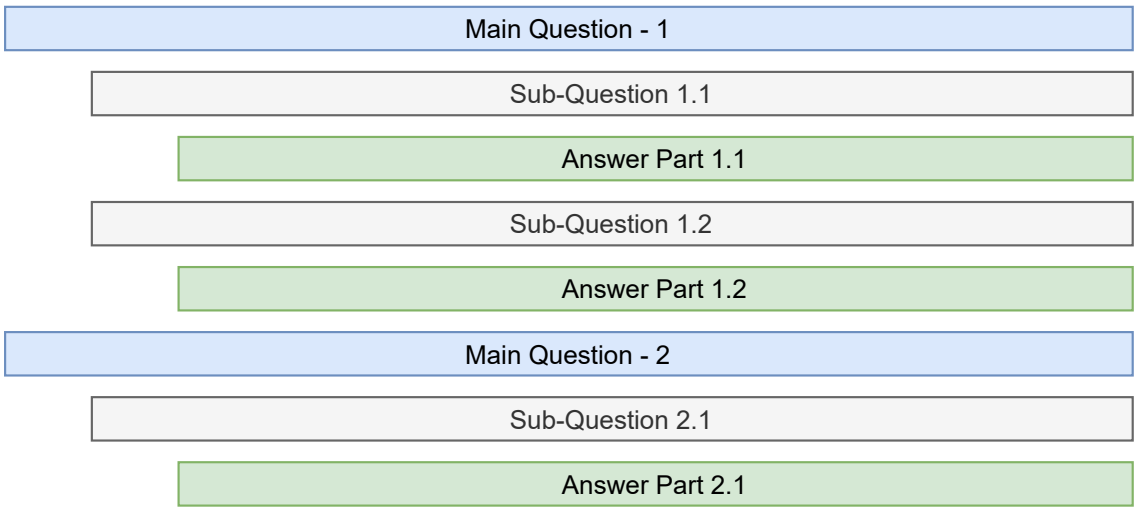


Figure 4.7: New Question Type Structure - 1

4.2.3 Classification of Answers

The questions analyzed in the survey have different types of answers. An answer type determines how a survey respondent can provide the answer to a particular question. Identification and analysis of different answer types would be useful while designing the database model for the survey questions. The identified answer types are single choice answer type, multiple choice answer type, drop-down answer type, free text answer type and range/likert-scale answer type. The are discussed as follows:

4.2.3.1 Single Choice Answer Type

A question has a single choice answer type when its answer part has multiple choices. The person taking the survey can only select one of these choices as the answer. Figure 4.9 is an example for a question with the single choice answer type. The survey respondent can select either the option ”in Deutschland” or the option ”in einem anderen Land” as the answer.

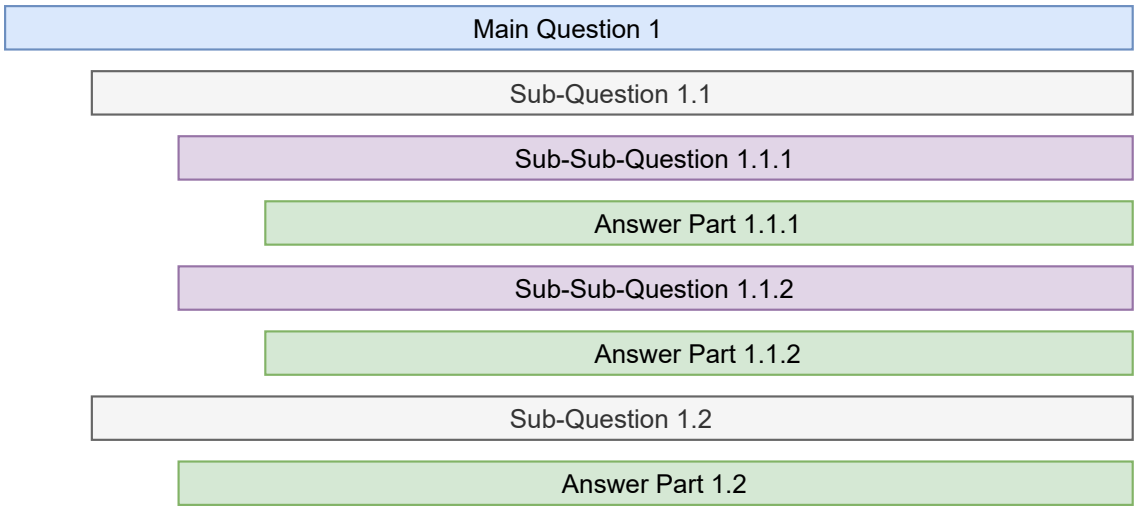


Figure 4.8: New Question Type Structure - 2

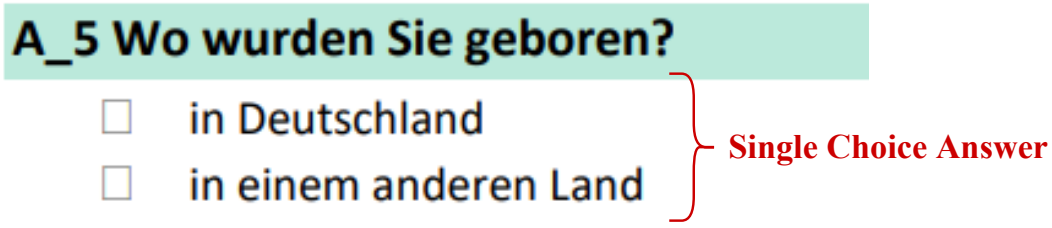


Figure 4.9: Example of a Single Choice Answer Type

4.2.3.2 Multiple Choice Answer Type

A question with a multiple choice answer type has multiple choices in the answer part. The survey respondent may select one or more choices as the answer. Figure 4.10 is an example for a question with multiple choice answer type. The person taking the survey can select multiple options from the list of nine choices as the answer to this question.

4.2.3.3 Drop-Down Answer Type

In the case of drop-down answer type, the answer part of the question has a drop-down list and the respondent of the survey can select one of the options from it as the answer. An example for such a question is given in Figure 4.11. The survey respondent can select one of the options from the drop-down list, "Geburtsland", as the answer.

4.2.3.4 Free Text Answer Type

A survey question has a free text answer type if the survey respondent has to answer that particular question as a free text in a provided text box. Figure 4.12 describes such a question. The question has two sub-questions and each of them has a free

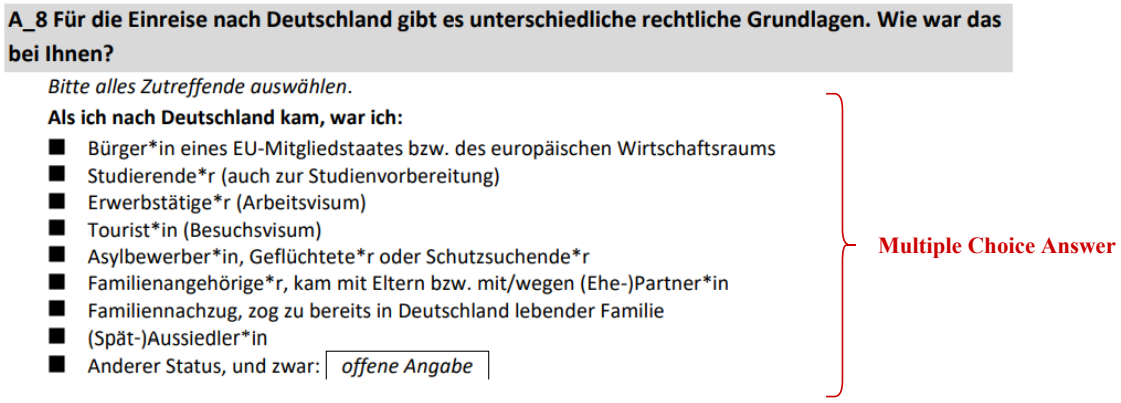


Figure 4.10: Example of a Multiple Choice Answer Type

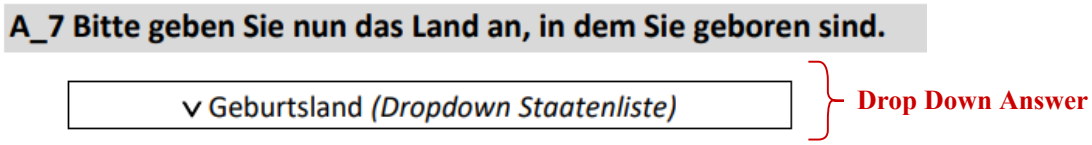


Figure 4.11: Example of a Drop Down Answer Type

text answer type. Thus the respondents of the survey can give free texts as their answers to this question in the given text boxes.

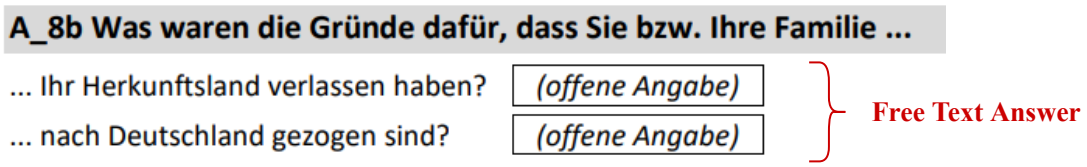


Figure 4.12: Example of a Free Text Answer Type

4.2.3.5 Range/Likert-Scale Answer Type

To provide an answer to a survey question with range/likert-scale as the answer type, the survey respondent has to select a value from a range of provided values. The range could be having numeric or text values. A question with range answer type is described in Figure 4.13. The question contains five sub-questions and each one could be answered by selecting a value ranging from ”nie” to ”sehr häufig”.

4.2.4 Special Cases

During the analysis of the survey questions, some special cases were identified in some of the questions. These special cases should be also addressed in the database schema to be designed as they contain important information about the survey or the questions. The identified special cases in the survey questions are instructions,

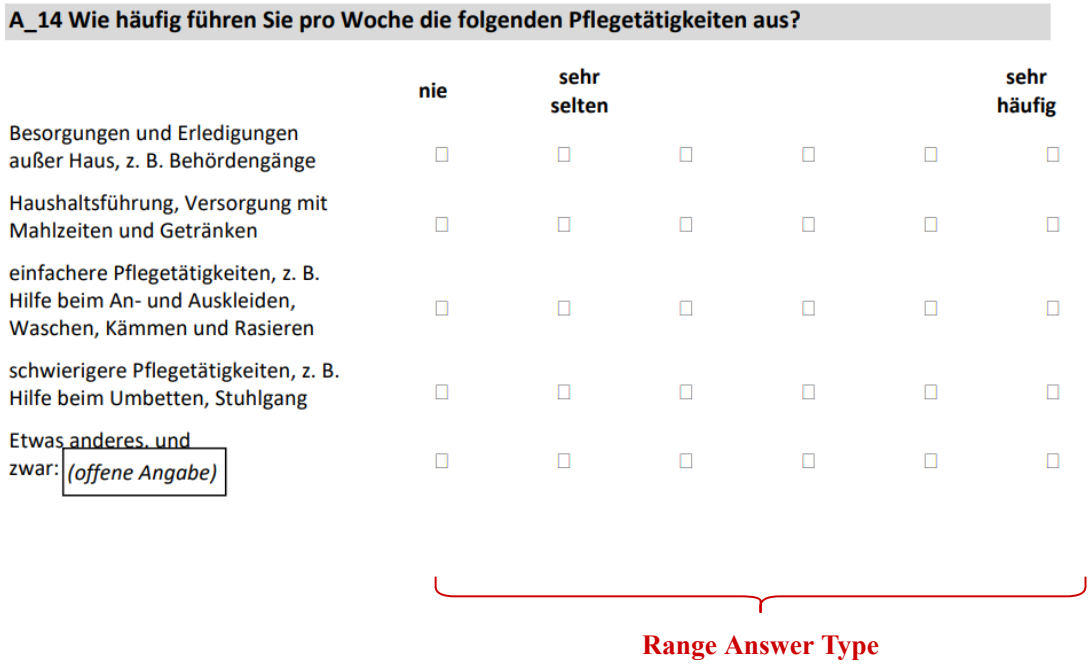


Figure 4.13: Example of a Range/Likert-Scale Answer Type

footnotes/conditions, additional free texts or drop-downs and grouped answer type choices. They are discussed beneath:

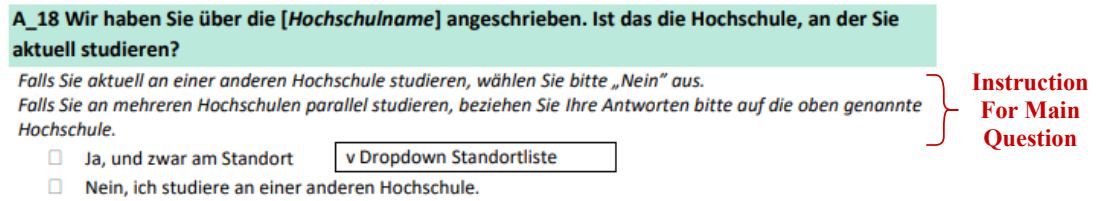
4.2.4.1 Instructions

Some survey questions have instructions on how to answer them. The instructions are given to help the person answering the question. The instructions could be associated with both main or sub-questions. Figure 4.14 gives an example of this special case. Figure 4.14(a) shows a question with an instruction for the main question and Figure 4.14(b) shows a survey question with an instruction for one of its sub-questions.

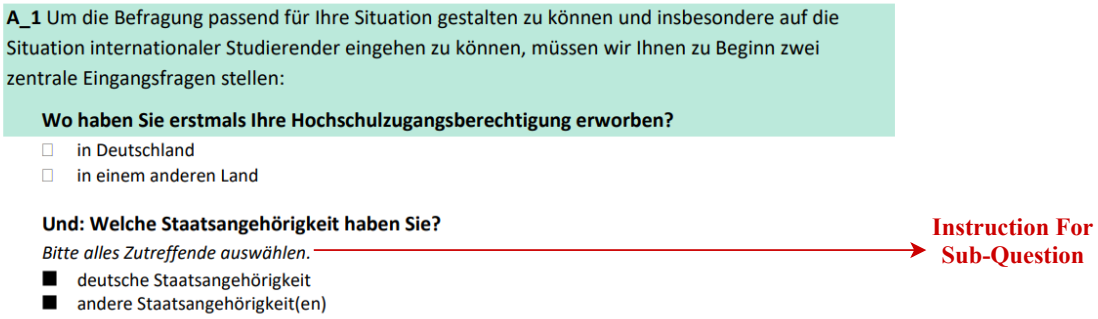
4.2.4.2 Footnotes/Conditions

Some survey questions have footnotes associated with them. The footnotes describe certain conditions or additional information regarding the question. So it is important to save these footnotes as well. In the examined survey, it is found that the footnotes could be associated with the main questions and/or the sub-questions.

If a question has a footnote associated with one of its question parts, it is indicated by a superscript, usually a number, at the beginning or the end of that question part. The actual footnote text is present at the bottom of the page. Figure 6.4 is an example of this special case. Figure 6.4(a) shows a survey question with footnotes associated with the main question as well as a sub-question. Figure 6.4(b) shows an example of the actual footnote texts which are displayed at the bottom of the page.



(a) Example: Instruction For Main Question



(b) Example: Instruction For Sub-Question

Figure 4.14: Example of Instructions

4.2.4.3 Additional Free Texts or Drop Downs

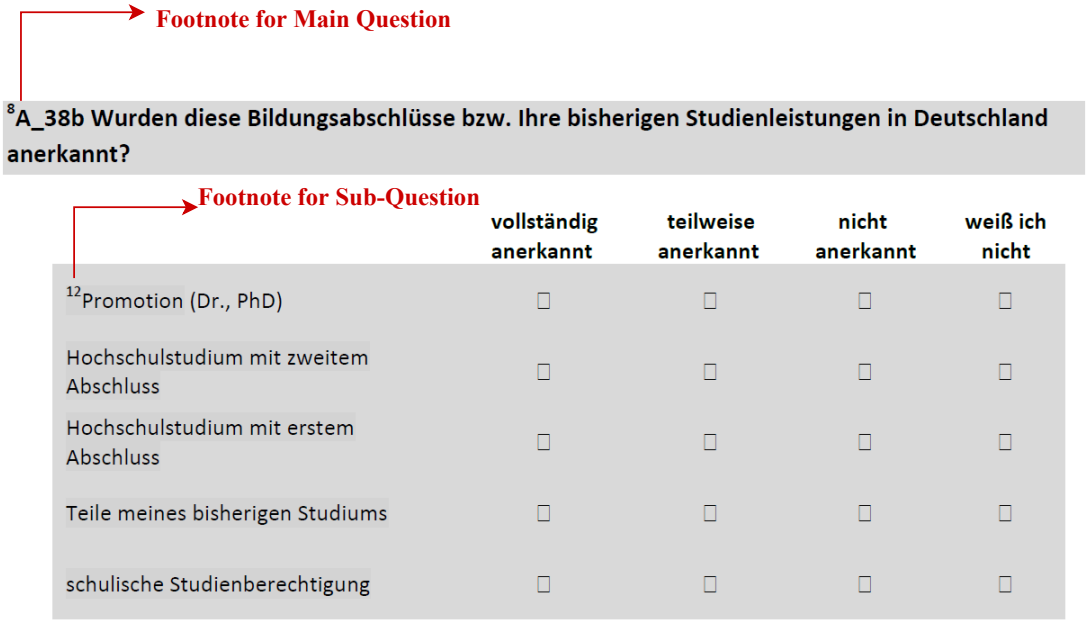
In the analysed survey some questions have additional free text or drop-downs associated with some question parts or choices. Such special cases should also be handled in the database. Some questions with such special cases are shown in Figure 4.16. Figure 4.16(a) shows an example of a question that has an additional free text associated with its last sub-question and the Figure 4.16(b) is a survey question in which an answer choice has an additional drop-down associated with it.

4.2.4.4 Grouped Answer Type Choices

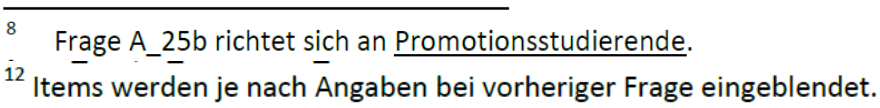
Some questions in the analysed survey contain answer type choices that are grouped into different categories. Figure 4.17 is a question having such a special case. The seven answer type choices are arranged into two groups as shown in the example.

4.2.5 Variable Names

Another important feature of the surveys conducted at DZHW is variable names. A variable name is used to uniquely identify a question/sub-question. So no two questions/sub-questions have the same variable name. The variable names are used to identify the answers correctly given by each survey respondent. An answer given by a survey respondent for a particular question or sub-question could be uniquely identified using the respondent id and the corresponding variable name. Furthermore, the variable names also characterize a questioned person. Each variable is a property of the respondent. Hence, they have a sociological meaning and importance as well.



(a) Question with Footnote



(a) Footnote at the Bottom of Page

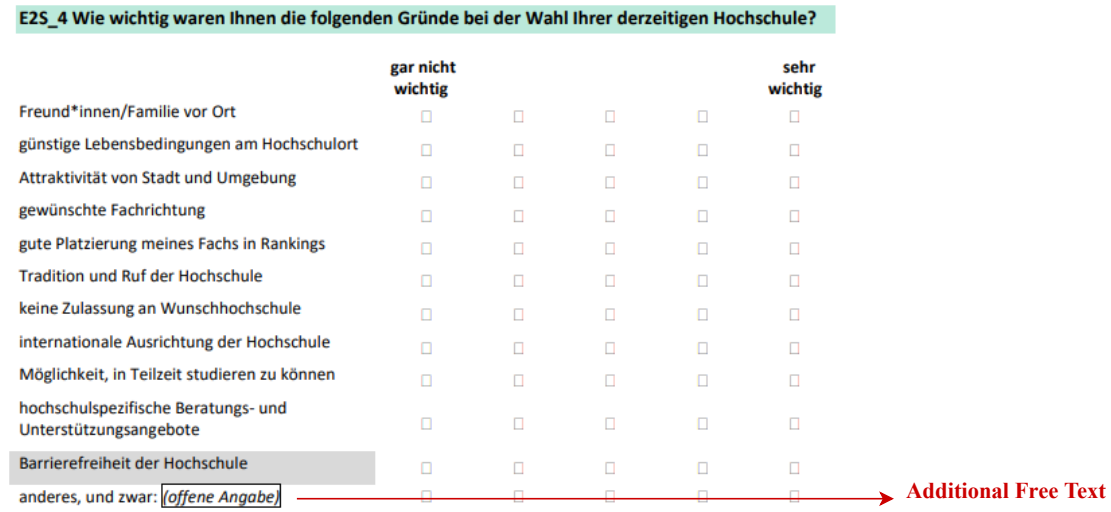
Figure 4.15: Example of Footnotes

A survey question can have multiple variable names associated with it depending on the number of main/sub-questions in it. If a question has only a main question, then it only has one variable name corresponding to that main question. On the other hand, if a survey question has sub-questions, then each one of those sub-questions has a variable name associated with it. Also the additional free texts and drop-downs discussed in Section 4.2.4.3 also have separate variable names associated with them.

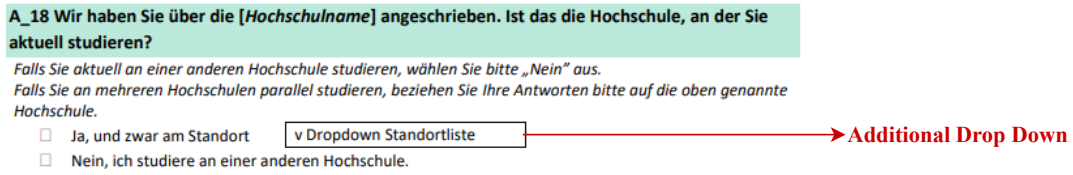
Figure 4.18 shows a survey question with variable names. The question has five sub-questions and an additional free text associated with the last sub-question as shown in the figure. Thus the question has six variable names, one for each sub-question (pflegt1, pflegt2, pflegt3, pflegt4, pflegt5) and one for the additional free text (pflegt5o). These variable names are also shown in the figure.

4.3 Requirements for Database Design

After analyzing the survey and the different questions in it, various implementation requirements were identified. These requirements contain important information about the survey and the survey questions and should be considered while designing



(a) Additional Free Text for a Sub-Question



(b) Additional Free Text for a Answer Choice

Figure 4.16: Example of Additional Free Texts or Drop Downs

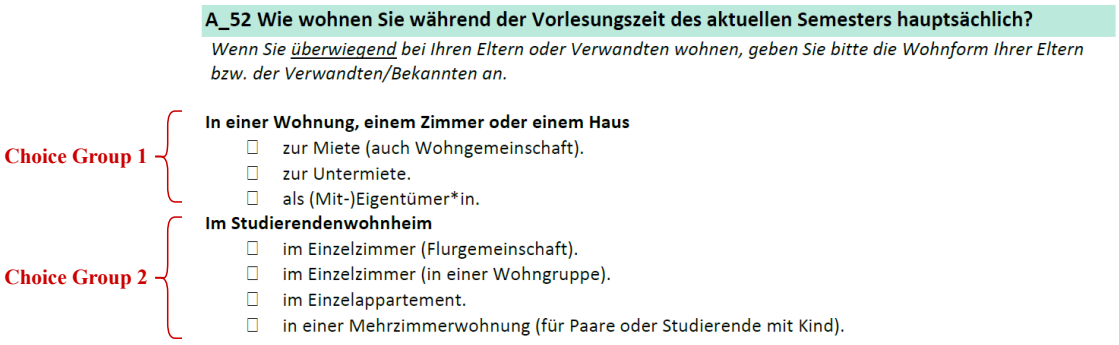


Figure 4.17: Example of Grouped Answer Type Choices

the database for storing the survey questions. The requirements identified for the database design are as follows:

- **Structure of the Survey:** The structure of the survey should be maintained in the database. A survey contains different modules with multiple questions. The database should be able to incorporate this hierarchy.

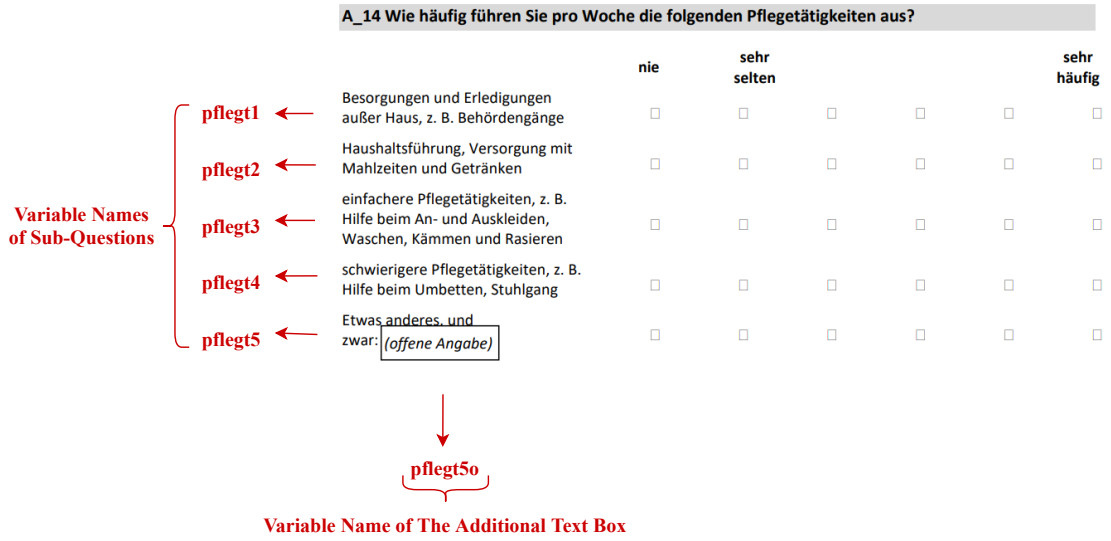


Figure 4.18: Example: Variable Names

- **Structure of Survey Questions:** Each survey question has a question part and an answer part. The database should be able to store these components non redundantly.
- **Non-Redundant Questions:** A survey can contain questions that could be already present in some other surveys. Thus these shared questions should be stored non redundantly in the database by defining all the necessary relationships.
- **Non-Redundant Answer Type Instances:** Sometimes, multiple questions can have the same answer type instances. For example, the same set single choice options could be used in two separate survey questions. In such cases, the shared answer type instances should be stored non redundantly in the database.
- **Question types:** The database should be flexible enough to handle the different question types discussed in Section 4.2.2. The different question types are level 0 questions, level 1 questions, level 1 grouped questions and level 0 multiple questions. Apart from the identified question types, the database should also handle new question types as discussed in Section 4.2.2.
- **Answer Types:** Like the question types, the new database should be able to incorporate all the different answer types such as single choice, multiple choice, drop-down, free text and range/likert-scale discussed in Section 4.2.3.
- **Special Cases:** The special cases discussed in Section 4.2.4 such as instructions, footnotes/conditions, additional free texts or drop-downs and grouped answer type choices should also be successfully stored in the database along with the questions as they contain important information about the survey.
- **Variable Names:** The database should be able to incorporate the variable name feature of the survey. All the properties of variable names should be met as well.

4.4 Conceptual Data Model

In this section, we present a conceptual data model based on the outcomes of the requirement analysis. This data model is a high-level model which provides only a general understanding of the examined survey data. It can be considered as a stepping stone in designing the final database. The conceptual model is only a basic model and does not provide any implementation or technical details of the final database. Instead, it explains the data in a way that is closer to how the users view the data. The model presented in this section is expanded in the future steps to build the final database model.

The data modelling technique used to create the conceptual data model is the Entity-Relationship (ER) Model. The ER model consists of an ER diagram. The ER diagram mainly consists of entities, attributes of the entities and relationships between the entities. An entity is a data object and is represented using a rectangle in the ER diagram. Attributes describe the properties of the entities and are represented using oval shapes in the ER diagram. A relationship describes the relationship between two entities and is represented using a diamond shape.

Figure 4.19 shows the conceptual data model for the analysed survey questions. The different identified entities in the data are Survey, Survey Module, Question, Question Part, Answer Part, Answer Part Choice, Additional Answer Part and Additional Answer Part Choice. Various relationships among these entities and their properties are discussed below:

- **Survey Entity:** The survey entity has *survey_name* as its attribute, which stores the name of the survey. Each survey contains different survey modules. A survey should have one or more survey modules in it.

  For example, the survey analysed in this thesis work could be considered as an instance of the survey entity with the survey name "Die Studierendenbefragung in Deutschland". The survey has nineteen different modules in it.

- **Survey Module Entity:** The survey module entity has a *survey_module_name* attribute, which stores the name of the survey module. Each survey module has multiple questions in it. A survey module should have at least one question. Furthermore, each survey module belongs to one and only one survey.

  For example, let us consider the survey "Die Studierendenbefragung in Deutschland". Each survey module in this survey could be considered as an instance of the survey module entity. Each survey module has multiple survey questions as well.

- **Question Entity:** Question entity has a *question_id* attribute, which is used to uniquely identify the question. Each question has multiple question parts, like the main question part, sub-question part and sub-sub-question part, depending on the type of question. The same question can belong to different survey modules in multiple surveys. In such cases, the question number would be different depending on the survey. For example, consider a question Q present in survey module SM1 and also in survey module SM2. Question Q is displayed

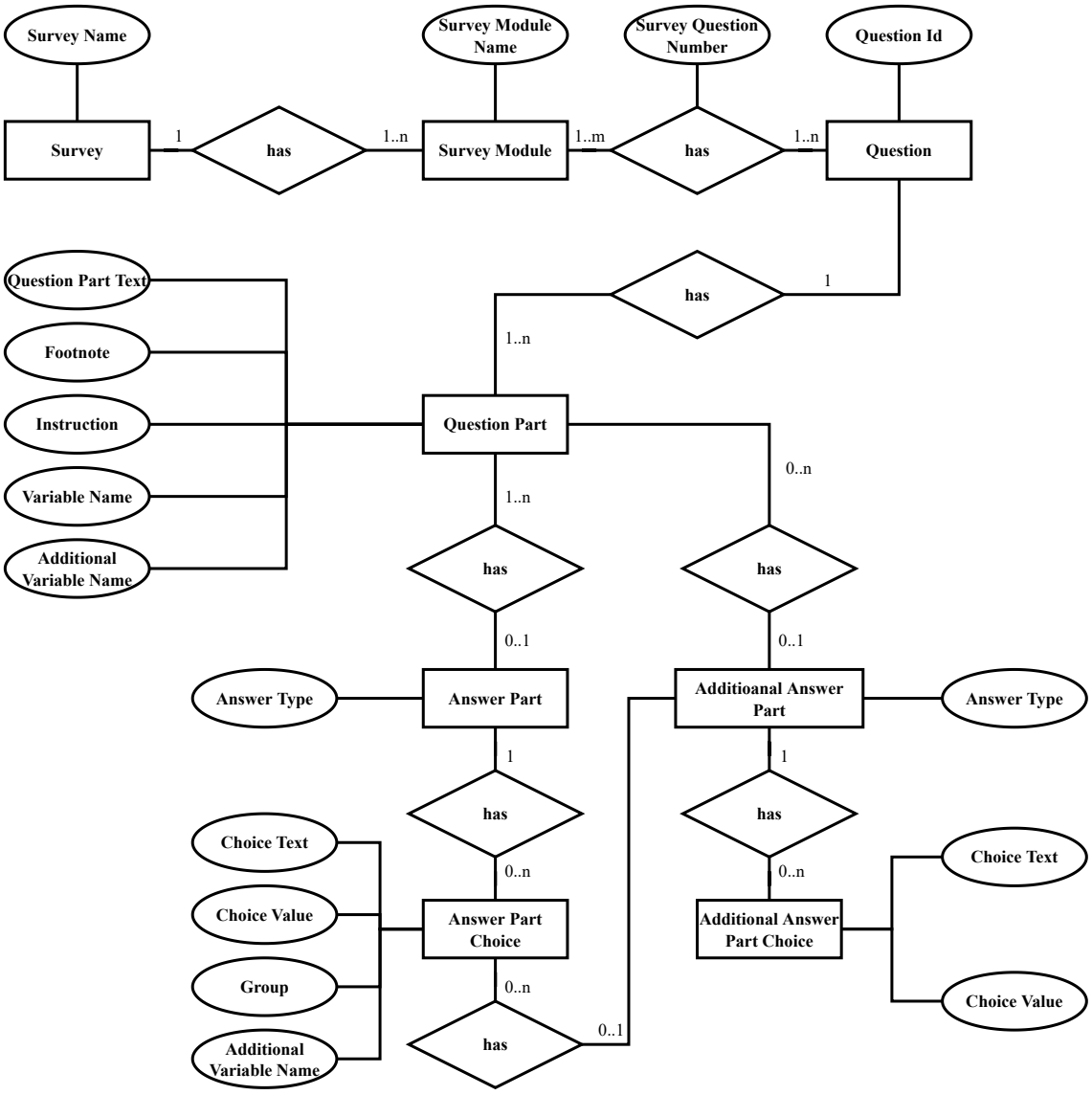


Figure 4.19: Conceptual Data Model for Survey Questions

as the second question in SM1, whereas in SM2, Q is displayed as the seventh question. In this case, even though the same question is used in both surveys, the question numbers are different. This is why the *survey_question_number* attribute, which stores the question number of a question in a survey, is associated with the relationship.

The question shown in Figure 4.20 could be considered as an instance of the question entity. The question belongs to a module in the survey, "Die Studierendenbefragung in Deutschland". The question has three question parts as well as shown in the figure.

- **Question Part Entity:** The question part entity has five attributes: *question_part_text, footnote, instruction, variable_name and additional_variable_name*. The *question_part_text* stores the text which is displayed in the survey. The *footnote* attribute and *instruction* attribute store the footnote information and instruction text, if any, associated with the corresponding question

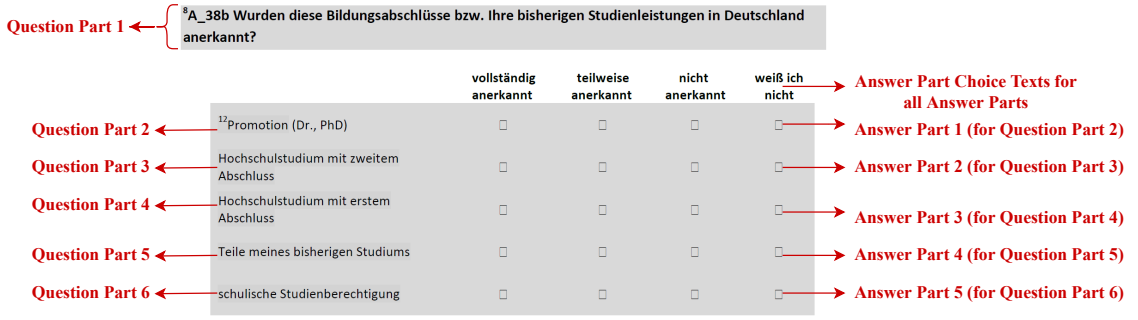


Figure 4.20: Example Question with Six Question Parts and Five Answer Parts

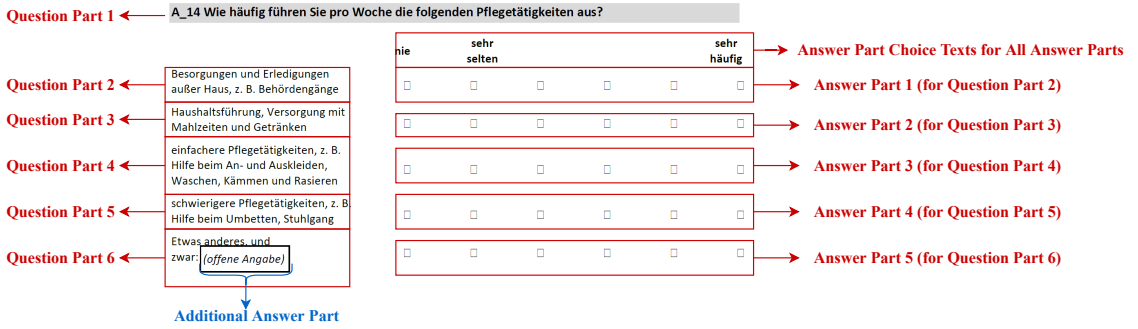


Figure 4.21: Example Question with an Additional Answer Part Associated with a Question Part

part. If the question part has an answer part, then it has a variable name which is stored in the *variable_name* attribute. If an additional free text or drop-down is associated with the question part, then also it has a corresponding variable name. For the distinction between the normal variable name and the variable name associated with the additional free text or drop-down, we call it an additional variable name as is stored in the *additional_variable_name* attribute.

While considering entity relationships, each question part belongs to a question and can also have an optional answer part and/or an optional additional answer part associated with it. The additional answer part includes the additional free text or drop-down discussed as special cases in Section 4.2.4.

Consider the question in Figure 4.20. The question contains six question parts as shown in the figure. Question Part 1 is the main question part and it does not have any answer parts. Whereas, Question Part 2-6 have corresponding answer parts and thus they have variable names associated with them. For the question in Figure 4.21 there is an additional answer part of type free text associated with the last question part (Question Part 6) as shown. Therefore, the question part has an additional variable name associates with it.

- **Answer Part Entity:** An answer part entity has an *answer_type* attribute. It is used to store the answer type which describes how the question is answered. Apart from this, an answer part may have multiple answer part choices depending on the answer type. For example, if the answer type is a free text,

then there are no answer type choices associated with that particular answer part. On the other hand, the answer part contains multiple choices if the answer type is drop-down, range, multiple-choice or single choice. An answer part can be a part of multiple survey questions as well.

The question in Figure 4.22 contains an answer part of type single choice. The answer part has two choices as well.

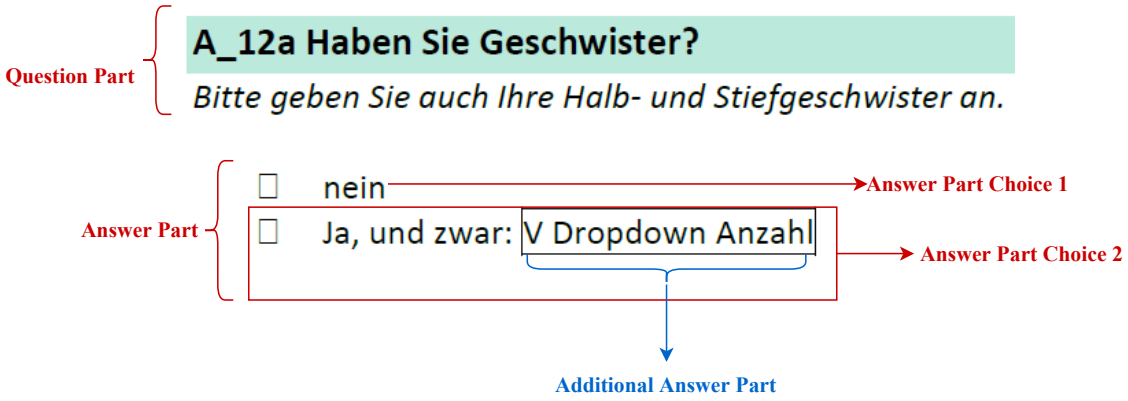


Figure 4.22: Example Question with an Additional Answer Part Associated with an Answer Part Choice

- **Answer Part Choice Entity:** The answer part choice entity has *choice_text, choice_value, choice_group* and *additional_variable_name* as its attributes. The *choice_text* attribute stores the text that is displayed as the choice in the survey. Each choice text has a value, usually a numeric value, associated with it and it is stored in the *choice_value* attribute. This value is used for the internal representation of the choice text in programs and databases as they can handle numeric values much easier than text values. If the choices of a particular answer part are categorized into different groups then the *choice_group* attribute contains the name of the group to which a choice belongs. If the choice contains an additional answer part (additional free text or drop-down) associated with it, then the *additional_variable_name* attribute stores the variable name of the additional answer part. In the case of entity relationships, an answer part choice entity may have an additional answer part entity.

  The answer part of the question in Figure 4.22 has two answer part choices as shown. Each of these choices has a corresponding choice text and choice value associated with it. The values are not shown as they are used for the internal representation. The second choice, Answer Part Choice 2, has an additional answer part associated with it and thus has a corresponding additional variable name as well.

- **Additional Answer Part Entity:** An additional answer part entity has an attribute called *answer_type*, which stores the type of the additional answer. In the survey which is analyzed for this thesis work, there exist only two types for an "additional answer part", free text and drop-down (called additional free text and additional drop-down). The additional answer part entity may have multiple additional answer part choices as well depending on the answer

type. An additional answer part might belong to one or more answer parts or question parts.

The question in Figure 4.22 has an additional answer part associated with the second answer part choice. It is of type drop-down and thus has multiple additional answer type choices as well. Also, in Figure 4.21, the last question part contains an additional answer part of type free text. Since the answer is given as a free text there is no additional answer part choices associated with it.

- **Additional Answer Part Choice Entity:** The additional answer part choice entity has two attributes *choice_text* and *choice_value*. The *choice_text* attribute stores the text which is displayed as the answer choice in the survey. The *choice_value* attribute stores the corresponding value of the choice text.

  The answer type of the additional answer part in Figure 4.22 is a drop-down and thus it has multiple additional answer type choices. Each of these choices has a choice text and a corresponding value as well.

4.5 Summary

In this chapter, we have done a detailed requirements analysis and conceptual data modelling for designing a database for managing survey questions. We started the requirement analysis by analyzing a survey, "Die Studierendenbefragung in Deutschland", conducted at DZHW. Different features of the survey were examined, and the structure of the survey was identified.

After analyzing the survey as a whole, we moved on to the analysis of individual questions in the survey. Initially, we comprehended the general structure of the survey questions. Furthermore, the different types of survey questions and answer parts were identified. Different types of questions are level 0 questions, level 1 questions, level 1 grouped questions, level 0 multiple questions. Different types of answer parts that describe how a question can be answered are single choice answer type, multiple choice answer type, drop-down answer type, free text answer type and range/likert-scale answer type. Additionally, we also discussed some special cases such as instructions, footnotes/conditions, additional free texts or drop-downs and grouped answer type choices that are present in the survey questions. These special cases are important as they contain important information about the survey. Finally, we also discussed an important feature of the surveys called variable names, which are used to uniquely identify the survey questions/sub-questions.

After analyzing the survey and the survey questions, the requirements for database design were identified and listed. Based on the outcomes of the requirement analysis, a conceptual data model was designed. The designed conceptual data model is a basic model without any implementation or technical details, which only provides a general understanding of the examined survey data. This conceptual data model is expanded to build the final database model in the future steps.

5. Selection of Databases

In this chapter, we select the SQL and NoSQL databases for implementing the database schemas for managing the survey questions. For this, different SQL and NoSQL databases are considered, and the most suitable ones are selected for implementing the final database model. The selection was done before the final database design, as the NoSQL database schema could only be designed after selecting the database. The physical schema of the NoSQL database is designed according to the data model implemented by the selected NoSQL database.

5.1 Selection of SQL Database

5.1.1 Analysis and Comparison

Different SQL databases which are considered and analysed for this thesis work are discussed below. At the time of writing this thesis work, the selected relational databases were the most popular among the available databases[1]. They are as follows:

5.1.1.1 Microsoft SQL Server

Microsoft SQL Server is another popular relational database managed and developed by Microsoft Corporation. It offers superior performance and reliability [VM21] and is often considered as an enterprise database used in large-scale organizations with complex functionalities [SJDM15]. Thus, like the Oracle database, Microsoft SQL Server is also not desired for small to medium sized applications.

5.1.1.2 MySQL

MySQL is a widely used open-source SQL database. It is currently under the ownership of the Oracle Corporation and was first released in the year 1995 [Let15]. MySQL is supported by many different operating systems and programming languages without any memory leaks [WAA02]. Because of its open-source nature, high availability, and superior performance, MySQL is considered the best relational database for medium and small-scale applications [IAB+17].

[1] https://db-engines.com/en/ranking

5.1.1.3 Oracle

Oracle is one of the most popular commercial relational databases. Oracle was first released in the year 1979 and is owned by the Oracle Corporation. Many companies opt for Oracle as their database because of its scalability and definitive architecture [KK01]. However, using Oracle for small scale applications is not advisable due to its high complexity [BRA12] and difficulty in managing the database [KK01].

5.1.1.4 PostgreSQL

PostgreSQL is also an open-source relational database managed by the PostgreSQL Global Development Group. It is a feature-packed database and is well ahead of any other relational databases in terms of implementing SQL standards [KK01]. When comparing open source databases, PostgreSQL is slower as compared to MySQL [PPJ17] and thus is less preferred when high performance is desired.

In Table 5.1, we provide a basic overview of the relational databases discussed in this section. All four databases have the relational model as their primary data model. MySQL and PostgreSQL are open-source systems, and the Oracle and Microsoft SQL Server have a commercial licence. Both MySQL and Oracle databases were implemented in C and C++. However, PostgreSQL is implemented only in C and Microsoft SQL Server is implemented only in C++. In the case of programming language support, Oracle is the winner with 24 supported programming languages, followed by MySQL, which supports 19 programming languages. Microsoft SQL Server supports 11 programming languages, whereas PostgreSQL supports only 10 languages. All the databases support ACID properties, which is an important feature of all relational databases. Furthermore, all of them offers durability and consistency as well.

5.1.2 Selection

Among the four previously discussed SQL databases, MySQL is selected to implement and analyze the SQL database schema designed to store the survey questions. This selection is made based on the following reasons:

- At the time of writing this thesis, MySQL was the second most popular relational database, and among the open-source relational databases, it ranked as the best as well[2].
- MySQL is considered the best relational database for medium and small scale applications. Even though Oracle and Microsoft SQL Server offer superior performance, they are less suited for small or medium scale applications.
- MySQL has an open-source license which makes it more favorable for this thesis work. On the other hand, Oracle and Microsoft SQL Server have a commercial license with restricted free versions. Even though PostgreSQL has an open-source license, it lacks behind MySQL in terms of performance.

[2] https://db-engines.com/en/ranking

	MySQL	Oracle	PostgreSQL	Microsoft SQL Server
Primary Data Model	Relational	Relational	Relational	Relational
License	open-source	Commercial	open-source	Commercial
Implementation Language	C and C++	C and C++	C	C++
Supported Programming Languages	Ada C C# C++ D Delphi Eiffel Erlang Haskell Java JavaScript Objective-C OCaml Perl PHP Python Ruby Scheme Tcl	C C# C++ Clojure Cobol Delphi Eiffel Erlang Fortran Groovy Haskell Java JavaScript Lisp Objective C OCaml Perl PHP Python R Ruby Scala Tcl Visual Basic	.Net C C++ Delphi Java JavaScript Perl PHP Python Tcl	C# C++ Delphi Go Java JavaScript PHP Python R Ruby Visual Basic
Transaction	ACID	ACID	ACID	ACID
Durability	yes	yes	yes	yes
Consistency	yes	yes	yes	yes

Table 5.1: Overview of Relational databases

5.2 NoSQL Database Selection

5.2.1 Analysis and Comparison

Document oriented databases are selected to store the survey questions from the different types of NoSQL databases discussed in Section 2.2.2.2. Document oriented databases could be well suited to represent the survey questions as each of the questions could be considered as a document with several attributes and multiple embedded documents. A survey question, for example, could be described as a document with attributes such as question number, survey module, and survey to which the question belongs, while the actual structure of the question and the corresponding answer type could be stored as embedded documents.

Because of their simple architecture, the key-value stores were not chosen. A question with a complex structure would be extremely difficult to represent using key-

value pairs. Furthermore, key-value stores also do not allow the search using values, which would make the search of survey questions very difficult. Column stores are primarily used for analytical and data warehousing applications, and designing a schema for survey questions in column stores would be less feasible. Graph databases are best suited for data that can be represented as graphs, as well as for the analysis of relationships between data entities. However, in the case of survey questions, the better representation model would be a document, and the analysis of relationships between entities is only of minor importance.

Different document oriented databases that are considered and analysed for this thesis work are discussed below. At the time of writing this thesis work, the selected databases are the most popular open-source document oriented databases among the existing ones[3]. They are as follows:

5.2.1.1 Couchbase

Couchbase is a multi-model NoSQL database from Couchbase, Inc. which supports multiple data models. It was first released in the year 2010. The primary database model of Couchbase is a document store. However, it also supports key-value storage and spatial data storage [Gü94] as secondary database models. This thesis work mainly concentrates on the document oriented feature of Couchbase. The main characteristics of Couchbase [CB19] are as follows:

- Couchbase stores data in the form of JSON (JavaScript Object Notation) documents.
- Couchbase uses an SQL like query language called N1QL for querying data. This makes the development easier.
- Couchbase offers features like Map-Reduce, secondary indexes and server-side scripting which improves the performance of the database.
- Like MongoDB, Couchbase also supports sharding which enables horizontal scalability.

Even though development with Couchbase is easier due to its SQL like language, it offers fewer functionalities, programming language support and security as compared to MongoDB.

5.2.1.2 CouchDB

CouchDB is also a multi-model NoSQL database, developed by the Apache Software Foundation. The initial version of CouchDB was released in the year 2005. Like Couchbase, the primary database model of CouchDB is also the document store along with Spatial data storage as the secondary model. The key characteristics [CB19] of CouchDB are discussed in the following points.

- JSON (JavaScript Object Notation) is the data model implemented by CouchDB to store documents.

[3] https://db-engines.com/en/ranking

- CouchDB manages the documents via Restful HTTP APIs and uses JavaScript as the querying language.
- Like Couchbase, CouchDB also implements features that improve its performance like Map-Reduce, secondary indexes and server-side scripting.
- Shading in CouchDB enables horizontal scaling.

The main disadvantages of CouchDB are lack of ad-hoc queries, no transaction support, lack of predefined data types and expensive arbitrary queries.

5.2.1.3 MongoDB

MongoDB is a popular document oriented database developed by MongoDB Inc. The first version of MongoDB was released in the year 2009. The main features of MongoDB [BBC13] include:

- MongoDB stores data in the form of BSON (Binary JavaScript Object Notation) documents. The BSON format is similar to the JSON (JavaScript Object Notation) format and is generally easy to work with. A group of documents is called a collection in MongoDB. A collection could be considered as an equivalent of the table in the relational database. A group of collections forms a database.
- MongoDB uses a JavaScript-based querying language to query data.
- MongoDB offers different indexing features such as single field indexes on a single field of a document, compound indexed on multiple fields of a document, multi-key indexes to index the array elements, geospatial indexes on geospatial coordinate data, text indexes for performing efficient text searches and unique indexes to ensure that the indexed fields do not have duplicate values.
- The aggregation pipeline feature of MongoDB enables complex aggregations of data. An aggregation pipeline in MongoDB contains multiple document processing stages. Each stage performs an data processing operation on the input documents and passes the processed documents to the next stage for further processing.
- MongoDB also offers special-purpose collections such as fixed-size collections which could accommodate only a fixed amount of data and short-to-live collections which would only live for a fixed duration.
- MongoDB offers queries which enable efficient management of data.
- The auto sharding feature of MongoDB efficiently handles the horizontal scaling of the database automatically.

However, a major disadvantage of MongoDB is security [KES16]. MongoDB was developed without giving much importance to security. Furthermore, the memory usage of MongoDB is also very high.

In Table 5.2, we provide a basic overview of the three document databases discussed in this section. All three databases have the document as their primary data model. As explained before, all of them have open-source licenses as well. The MongoDB is implemented in C++, and the CouchDB is implemented using Erlang. However, Couchbase is implemented using C, C++, Go and Erlang. In the case of programming language support, MongoDB is the winner with 29 supported languages. CouchDB supports 17 programming languages, and PostgreSQL supports only 14 languages. Only MongoDB and Couchbase support ACID properties. However, all of them offers durability and consistency. Additionally, MongoDB and Couchbase offer SQL support as well. In MongoDB, SQL queries could be executed using special connectors, and the Couchbase supports N1QL, which is an extension of ANSI SQL to JSON.

5.2.2 Selection

Among the above-discussed document databases, MongoDB is selected to store the survey questions in this thesis work. The main reasons for choosing MongoDB are as follows:

- MongoDB is the most popular and widely used document database with good community support as compared to its counter parts[4].
- MongoDB supports more programming languages than Couchbase and CouchDB.
- The aggregation pipeline in MongoDB enables efficient processing of complex data aggregations that would be useful for analyzing the survey data.
- Even though Couchbase offers a simple SQL like querying language, MongoDB also provides connectors that enable the use of SQL queries against MongoDB collections.
- MongoDB uses BSON that supports more data types than JSON, used in Couchbase and CouchDB.
- MongoDB performs better than Couchbase and CouchDB in terms of functionalities [CB19].

5.3 Summary

In this chapter, we have selected the SQL and NoSQL databases for implementing the final database model for managing the survey questions. Initially, four different SQL databases were considered. They are Microsoft SQL Server, MySQL, Oracle and PostgreSQL. After comparing the four databases, MySQL was selected to implement our database for managing the survey questions. The main reasons for selecting the MySQL database are its high popularity, suitability, open-source nature and comparable performance.

[4]https://db-engines.com/en/ranking

In the case of NoSQL databases, we selected the document-oriented databases for managing the survey question from the different types of NoSQL databases as it is well suited to represent the survey questions. From document-oriented databases, three popular ones were considered and compared. They are MongoDB, Couchbase and CouchDB. MongoDB is selected to implement our database after the comparison. The main reasons for choosing the MongoDB database are its high popularity, good programming language support, aggregation pipelines for the efficient processing of complex data aggregations, SQL support and better performance.

	MongoDB	**Couchbase**	**CouchDB**
Primary Data Model	Document	Document	Document
License	open-source	open-source	open-source
Implementation Language	C++	C C++ Go Erlang	Erlang
Supported Programming Languages	Actionscript C C# C++ Clojure ColdFusion D Dart Delphi Erlang Go Groovy Haskell Java JavaScript Lisp Lua MatLab Perl PHP PowerShell Prolog Python R Ruby Rust Scala Smalltalk Swift	.Net C Clojure ColdFusion Erlang Go Java JavaScript Perl PHP Python Ruby Scala Tcl	C C# ColdFusion Erlang Haskell Java JavaScript Lisp Lua Objective-C OCaml Perl PHP PL/SQL Python Ruby Smalltalk
Transaction	Multi-document ACID Transactions with snapshot isolation	ACID	-
Durability	yes	yes	yes
Consistency	yes	yes	yes
SQL Support	SQL queries via MongoDB Connector	N1QL (Extension of ANSI SQL to JSON)	-

Table 5.2: Overview of Document Oriented databases

6. Database Design

In this chapter, we design an SQL and NoSQL database models for the survey questions. The conceptual data model presented in **??** is expanded for implementing the database models. The SQL database model is implemented in a MySQL database and the NoSQL model is implemented in a MongoDB database. The design, especially the NoSQL one, is done by considering the characteristics of the database in which the models are implemented.

6.1 SQL Database Design

The conceptual data model discussed in **??** does not represent the fully normalized data. One of the main characteristics of the SQL database is data normalization. The data stored in a relational database should be normalized to prevent data redundancies. Therefore, the conceptual model is normalized to obtain the final database model. Figure 6.1 shows the final SQL database model.

According to the standard procedure, all entities in the conceptual data model are converted into tables. Apart from these tables, we also introduce six new tables in the SQL database design to ensure data normalization. The newly included tables are Question Part Text, Footnote, Choice Group, Choice Text, Question Part Instruction and Survey Module Question. Furthermore, some other modifications are also done to ensure the correct implementation of the database such as extending the structures of question part, answer part choices and additional answer part choices. All the tables in the SQL database schema are discussed below.

6.1.1 Survey

The survey table stores the details of the surveys. It has two attributes. The *survey_name* attribute stores the name of the survey and, the *survey_id* attribute stores the ids that are used to uniquely identify the corresponding survey records. Each survey record in the table should have one or more survey modules associated with it.

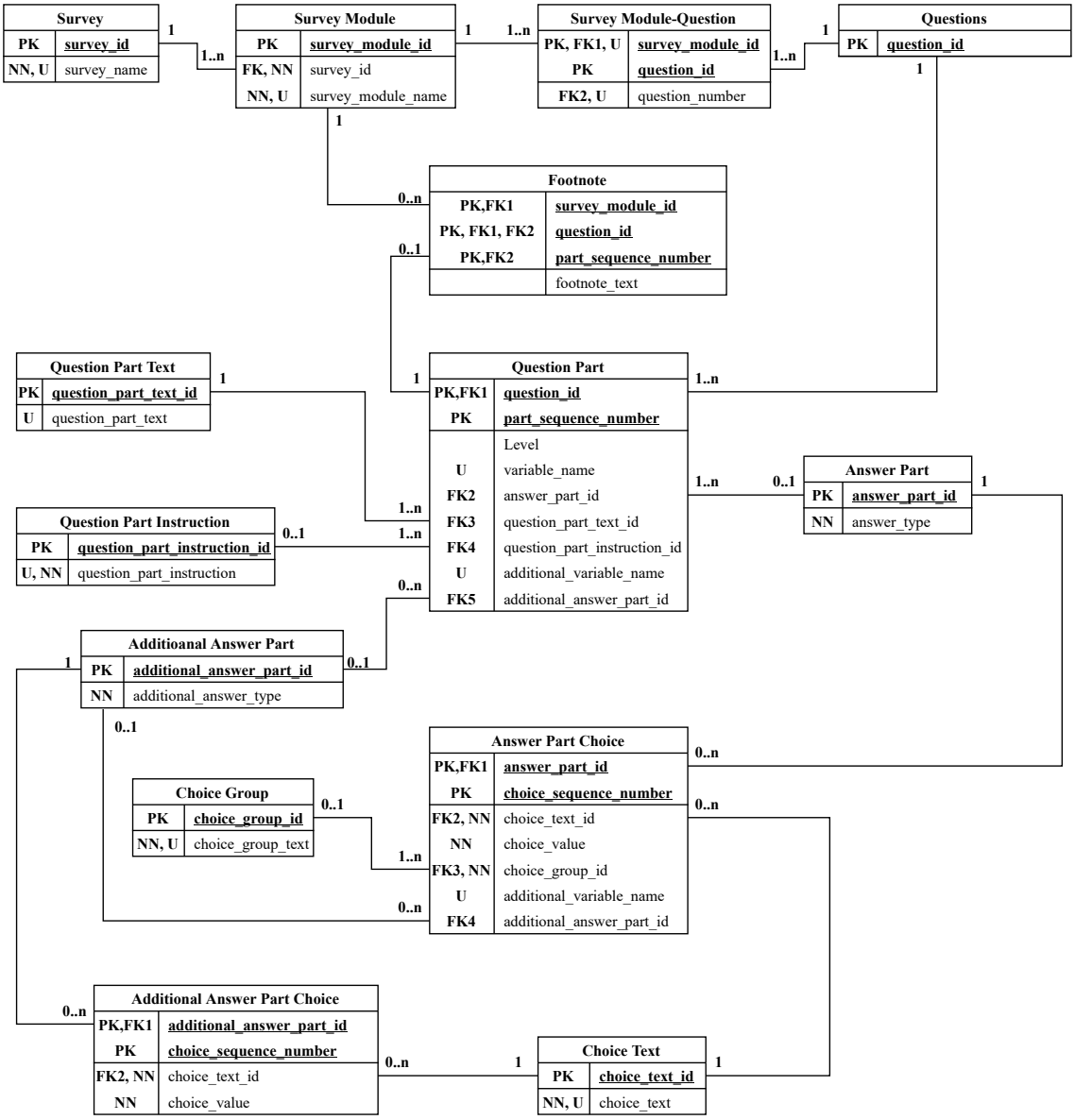


Figure 6.1: SQL Database Design

6.1.2 Survey Module

The survey module table stores the details of the different survey modules. The table has three attributes. The *survey_module_id* attribute is used to uniquely identify a survey module record. The *survey_id* attribute stores the id of the survey to which the survey module belongs. The *survey_module_name* stores the name of the survey modules. Each survey module should have multiple questions and, it should belong to a survey as well.

6.1.3 Question

The question table stores the unique id of each survey question in the *question_id* attribute. A question should belong to at least one of the survey modules. So, each *question_id* should be present in at least one entry of the survey module question

table. Furthermore, each question record should have at least one question part as well. So, each *question_id* should be present in at least one record of the question part table.

6.1.4 Survey Module Question

The survey module question table is used to handle the *question_number* attribute associated with the relationship connecting the survey module entity and the question entity in the conceptual data model (Chapter 4). The *question_number* attribute is associated with the relationship because a question can belong to different survey modules in multiple surveys. In such cases, the question number would be different depending on the survey module. The new table, survey module question, stores all the questions in a survey module with their corresponding question number. In this way, the survey questions are made independent of the surveys. The table contains three attributes *survey_module_id*, *question_id* and *question_number*. The *survey_module_id* stores the id of a survey module from the survey module table. The *question_id* stores the id of a question from the question table. Finally, the *question_number* stores the question number of the question in the survey module. Each table record is uniquely identified using the combination of the *survey_module_id* and the *question_id* attributes.

For example, consider a survey module (*survey_module_id* = 1) with a question (*question_id* = 100) as its tenth question. Assume that the same question is also present in another survey module (*survey_module_id* = 2) as its seventh question. In this case, the entries in the survey module question table are shown in Table 6.1. There is an entry for the same question for both the survey modules with the corresponding question number. Here, the considered question is a standalone object which could be used in multiple survey modules.

survey_module_id	**question_id**	question_number
1	100	10
2	100	7

Table 6.1: Example of Survey Module Question Table

6.1.5 Question Part Text

The question part text table stores the texts of the question parts which are associated with each of the question parts. The table contains two attributes *question_part_text_id* and *question_part_text*. The *question_part_text* stores the texts associated with the question parts and the corresponding *question_part_text_id* attribute stores the ids which are used to uniquely identify the texts. For a saved text, the corresponding id is stored in the question part table records that has the text. This is done to prevent redundant storage of question part texts in the database.

6.1.6 Question Part Instruction

The question part instruction table stores the instructions associated with the question parts. Each instruction text has a corresponding unique id. The table contains

two attributes *question_part_instruction_id* and *question_part_instruction*. The *question_part_instruction* stores the instructions associated with the question parts and the corresponding *question_part_instruction_id* attribute stores the ids which are used to uniquely identify the instructions. The instructions are linked to the question part table records using the ids. Redundant storage of instruction texts in the database is avoided using this strategy.

6.1.7 Question Part

A question can have multiple question parts (Figure 6.2). Saving the question parts according to the definition given in the conceptual data model will create problems as it would be difficult to retrieve the structure of the question from the saved information. For example, consider the question given in Figure 6.2. The question has two question parts, and according to the conceptual data model, we could save these question parts as shown in Table 6.2. However, at a later stage, when we query these question parts to retrieve the question, we would not be able to reconstruct the question as we do not have sufficient information. We could not identify the order of the question parts and, we also could not identify which question part is the main question part and which one is the sub-question part.

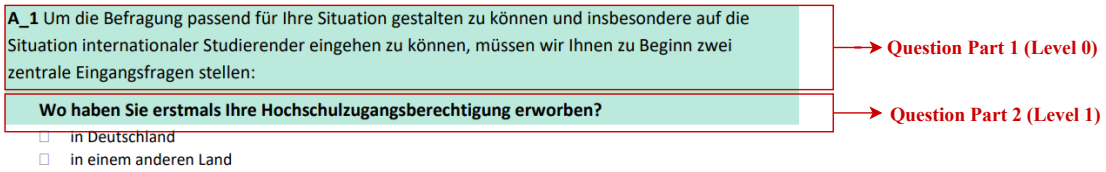


Figure 6.2: Example Question IV

Question Part Text	FootNote	Instruction	Variable Name	Additional Variable Name
Um die Befragung passend...	Null	Null	Null	Null
Wo haben Sie erstmals Ihre...	Null	Null	vsbdeba	Null

Table 6.2: Question Parts According to Conceptual Data Model

Therefore, the sequential order of the question part in a question should be saved along with the question part information. Furthermore, the level of the question part should also be saved so that the correct structure of the question can be retrieved. Therefore, the structure of the question part table should include this information as well. As a result, in the database design, we have extended the attributes of the question parts to include *part_sequence_number* and *level* as new attributes. The *part_sequence_number* stores the sequential order of the question parts in the question and the *level* attribute stores the level of the question part.

The remaining attributes of the question part table are as follows. The *question_id* stores the id of the corresponding question. The *variable_name* attribute stores the corresponding variable name if the question part has an answer part. The *answer_part_id* stores the id of the answer part if the question part has one. The *question_part_text_id* stores the id of the question part text associated with the question part. The *question_part_instruction_id* stores the id of the instruction if

the question part has one. The *additional_variable_name* stores the corresponding variable name if the question part has an additional answer part. Finally, the *additional_answer_part_id* stores the id of the additional answer part if the question part has one. Each table record is uniquely identified using the combination of *question_id* and *part_sequence_number* attributes.

According to the SQL database model, the question parts of the question in Figure 6.2 could be saved as shown in Table 6.3.

question_id	**part_sequence_number**	level	variable_name	answer_part_id	question_part_text_id	question_part_instruction_id	additional_variable_name	additional_answer_part_id
100	1	0	Null	Null	20	Null	Null	Null
100	2	1	vsbdeba	1000	30	Null	Null	Null

Table 6.3: Question Parts According to Final Database Model

6.1.8 Choice Text

The choice text table stores the texts associated with the answer part choices. The table has two attributes *choice_text_id* and *choice_text*. The texts associated with the choices in the questions are stored in the *choice_text* attribute and, the corresponding *choice_text_id* attribute stores ids that could be used to uniquely identify the texts. The id of a choice text is stored in the answer part choice and additional answer part choice records that have the particular text. The choice texts are isolated to ensure data normalization by preventing duplicate storage.

6.1.9 Choice Group

Like choice texts, the texts associated with the choice groups in case of grouped answer part choices are also stored in a separate table. This table is named as choice group and contains two attributes *choice_group_id* and *choice_group_text*. The *choice_group_text* attribute stores the choice group descriptions. Each one of these texts is identified by a unique id which is stored in the *choice_group_id* attribute. Only the answer part choices could be grouped and, the choice records which are grouped store the id of the group in which they belong. Duplicate storage of texts could be avoided by this.

6.1.10 Answer Part

The answer part choice table stores the information regarding the answer parts in the survey questions. It has two attributes: the *answer_part_id* attribute stores the ids, which could be used to uniquely identify the answer parts and the *answer_type* attribute stores the corresponding answer type of the answer part, which explains how the answer could be given. An answer part record may or may not have answer part choices depending on its *answer_type* attribute value. But, each answer part should belong to at least one of the question part records in the question part table.

6.1.11 Answer Part Choice

An answer part can also have multiple choices. Like the question parts, it is important to save the sequential order of the answer part choices as well. Otherwise, we would not be able to reconstruct the question with the correct order of the answer part choices. For example, consider the question in Figure 6.3. The question has a main question part with an answer type as a single choice. The answer part has seven different options/choices. According to the structure of the answer part choice in the conceptual data model, these choices could be saved as shown in Table 6.4. However, at a later stage, when we retrieve the choices to reconstruct the question, we would not be able to do so as we do not have the order in which the choices are arranged.

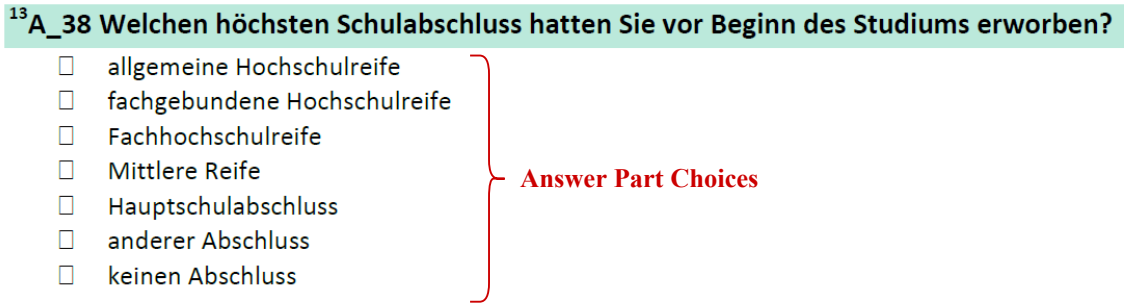


Figure 6.3: Example Question with Answer Parts Choices

Choice Text	Choice Value	Choice Group	Additional Variable Name
allgemeine Hochschulreife	10	Null	Null
fachgebundene Hochschulreife	20	Null	Null
Fachhochschulreife	30	Null	Null
Mittlere Reife	40	Null	Null
Hauptschulabschluss	50	Null	Null
anderer Abschluss	60	Null	Null
keinen Abschluss	70	Null	Null

Table 6.4: Answer Part Choices According to Conceptual Data Model

Thus, in this case also we need to save the sequential order of the choices along with the existing information. Therefore, the answer part choice table should include the choice order as well. This information is stored in the *choice_sequence_number* attribute of the answer part choice table.

The remaining attributes of the answer part choice table are follows. The *answer_part_id* stores the id of the answer part to which the choice belongs. The *choice_text_id* stores the id of the choice texts. The *choice_value* stores the value of the particular choice. The *choice_group_id* attribute stores the id of the choice group if the choices of the answer parts are grouped together. The *additional_variable_name* stores the corresponding variable name of the additional answer part if the choice has got one. Finally, the *additional_answer_part_id* attribute stores the id the additional answer part if the choice has got one. Each table record is uniquely identified by a combination of *answer_part_id* and *choice_sequence_number* attributes.

According to the structure of the answer part choice table, the choices in the question Figure 6.3 are saved as shown in Table 6.5. With this structure we are able to successfully reconstruct the choices in the question.

answer_part_id	**choice_sequence_number**	choice_text_id	choice_value	choice_group_id	additional_variable_name	additional_answer_part_id
100	1	1000	10	Null	Null	Null
100	2	1001	20	Null	Null	Null
100	3	1002	30	Null	Null	Null
100	4	1003	40	Null	Null	Null
100	5	1004	50	Null	Null	Null
100	6	1005	60	Null	Null	Null
100	7	1006	70	Null	Null	Null

Table 6.5: Answer Part Choices According to SQL Database Model

6.1.12 Additional Answer Part

The answer part choice table stores the information regarding the additional answer parts in the survey questions. It has two attributes. The *additional_answer_part_id* attribute stores the ids, which could be used to uniquely identify the additional answer parts. The *additional_answer_type* attribute stores the corresponding answer type of the additional answer part which, explains how the answer could be given. An additional answer part record may or may not have answer part choices depending on its *additional_answer_type* attribute value. However, each additional answer part should either belong to at least one of the question part records or answer part choice records.

6.1.13 Additional Answer Part Choice

The additional answer part choice table stores the choices of the additional answer parts. Like in the answer part choice table, it is important to store the sequential order of the choices in this table as well for the correct reconstruction of survey questions. This information is stored in the *choice_sequence_number* attribute of the table. The additional answer part choice table has three more attributes. The *additional_answer_part_id* stores the id of the additional answer part to which the choice belongs. The *choice_text_id* attribute stores the id of the text associated with the choice from the choice text table. The *choice_value* stores the value of the choice. Each table record is uniquely identified using the *additional_answer_part_id* and *choice_sequence_number* attributes combined.

6.1.14 Footnote

In the conceptual data model, the footnote information in a question is stored in the corresponding question part entity to which it belongs. However, it might create some problems when the same question is used in different surveys. The footnotes

usually contain information which is specific to the surveys such as how a question is related to other questions in the survey. For example, Figure 6.4 shows two example footnotes from the survey. The first footnote is a common footnote for three questions (A_38a, A_38b and A_38c) and explains that these questions are only given to international students. The second footnote explains that the question which has this footnote is related to the previous question in the survey. Thus, saving the footnotes in the question part entity might affect the independence of questions as our main objective is to save the questions as standalone objects in the database.

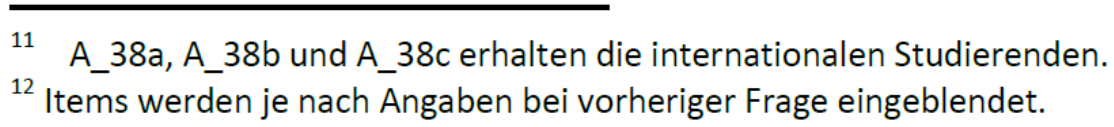


Figure 6.4: Footnote Examples from the Survey

Therefore, a new table called footnote is created for saving the footnote information. An instance of a footnote table belongs to a question part and contains attributes to store the survey module details and question details along with the actual footnote text so that a footnote could be easily mapped to its question in a survey module. This makes the survey questions standalone so that they could be used in different surveys without any problems.

The footnote table has four attributes *survey_module_id*, *question_id*, *part_sequence_number* and *footnote_text*. The *survey_module_id* stores the id of the survey module. The *question_id* stores the id of the question that belongs to the survey module. The *part_sequence_number* stores the position of the question part in the question which contains the footnote. Finally, the *footnote_text* stores the actual footnote text which is associated with the question. Each footnote record is uniquely identified using a combination of *survey_module_id*, *question_id* and *part_sequence_number* attributes. The combination of *survey_module_id* and *question_id* can verify whether the question is present in the corresponding survey module by using the survey module question table. Furthermore, the combination of *question_id* and *part_sequence_number* can verify the existence of question part from the question part table.

6.2 NoSQL Database Design

We select MongoDB as the NoSQL database for managing the survey questions. MongoDB is a document-oriented database that uses the document data model. Therefore, the survey questions should be saved in MongoDB in the form of BSON (Binary JavaScript Object Notation) documents. The BSON format is similar to the JSON (JavaScript Object Notation) format and is generally easy to work with. A database in MongoDB contains multiple collections where each one holds multiple documents in it.

Unlike SQL databases, NoSQL databases prefer de-normalized data, which leads to data redundancy. However, one of our main objectives is to store the survey questions non redundantly so that they could be shared among different surveys.

Furthermore, the answer parts and additional answer parts should also be stored non-redundantly so that the questions that use the same answer parts or additional answer parts have the same choice values. It is to establish uniformity in the answers across the surveys. Therefore, we need to create multiple collections for making these question sections non-redundant. Since the data is shared among different MongoDB collections, we should establish connections between the related documents stored in different collections. For this, MongoDB offers a feature called database reference which, allows a document to refer to another document in the same or different database. For example, A document D1 can refer to another document D2 by saving the unique document id (called as *_id*) of D2 as a field in D1. It works similar to the primary key-foreign key concept in SQL databases but does not ensure referential integrity.

For designing the NoSQL database, we have de-normalized the conceptual database model. Different related entities in the conceptual data model are combined to form MongoDB collections. While converting an entity into a MongoDB collection, all the attributes in the entity will become document fields in the collection. Furthermore, when two related entities are combined to form a MongoDB collection, all the attributes in the main entity will become document fields in the new collection. The second entity which belongs to the main entity will be saved as a nested document in the collection document corresponding to the main entity. The entities are combined to create the collections without violating the requirements, such as non-redundant questions and answer parts discussed in the previous paragraph.

The MongoDB database, which is designed for managing the survey questions has five different collections: survey collection, question collection, answer part collection, additional answer part collection and footnote collection. These collections are disused in detail in the following.

6.2.1 Survey Collection

The survey collection contains all the details about the surveys and survey modules. The survey entity and survey module entity in the conceptual data model are merged to form the survey collection. When comparing with the SQL database model, we can see that the survey collection contains the data stored in survey, survey module and survey module-questions tables in the SQL database model. The structure of the survey collection is shown in Figure 6.5.

The survey collection contains *survey* documents. Each *survey* document contains three attributes: *SurveyId*, *SurveyName* and *SurveyModules*. *SurveyId* stores the id, that could be used to uniquely identify the survey document. *SurveyName* attribute stores the name of the survey. The *SurveyModules* attribute contains an array of *SurveyModule* documents.

Each *SurveyModule* document has three attributes: *SurveyModuleId*, *SurveyModuleName* and *surveyModuleQuestions*. The *SurveyModuleId* stores the id, which could be used to uniquely identify the corresponding survey module. The *SurveyModuleName* attribute stores the name of the survey module, and the *SurveyModuleQuestions* attribute contains an array of *SurveyModuleQuestion* documents.

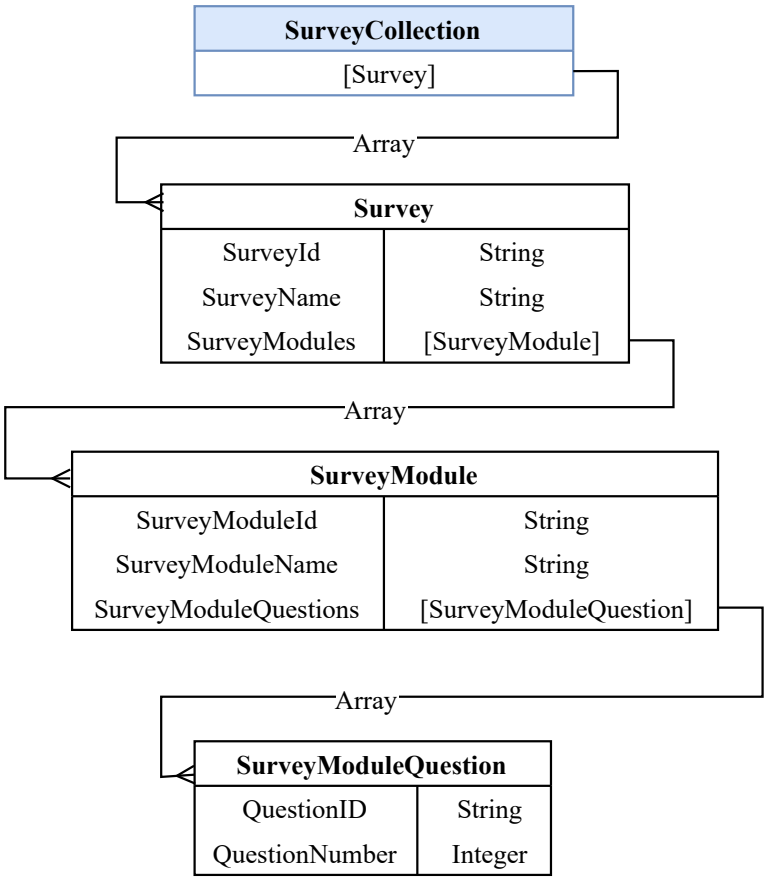


Figure 6.5: Structure of Survey Collection

A *SurveyModuleQuestion* document contains information about a question that is a part of the corresponding survey module. It has two attributes: *QuestionId* and *QuestionNumber*. The *QuestionId* attribute refers to a question document in the question collection by saving the question's unique id. The *QuestionNumber* attribute stores the question number that denotes the position of the question in the survey module.

6.2.2 Question Collection

The question collection stores all the information about the survey questions and their corresponding question parts. The question and question part entities in the conceptual data model are merged to create the question collection. The question collection stores the details present in the question, question part, question part text and question part instruction tables in the previously discussed SQL database design. The structure of the question collection is shown in Figure 6.6.

The question collection contains multiple *Question* documents. Each *Question* document has two attributes: *QuestionId* and *QuestionParts*. The *QuestionId* attribute stores the ids that are used to uniquely identify the corresponding *Question* documents. The *QuestionParts* attribute contains an array of *QuestionPart* documents.

A question may have multiple question parts. The *QuestionPart* document stores the details of a question part. A *QuestionPart* document contains eight attributes:

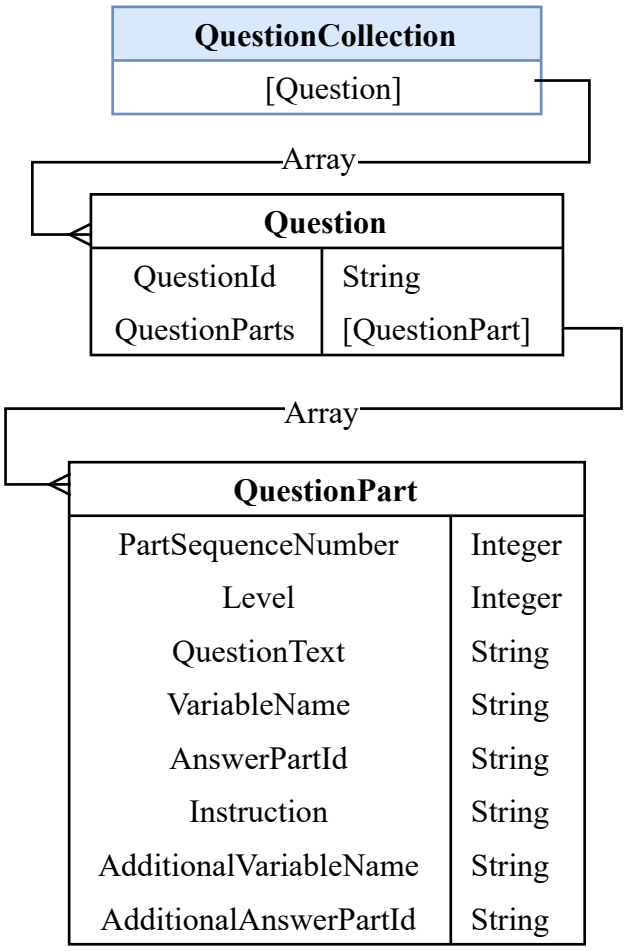


Figure 6.6: Structure of Question Collection

PartSequenceNumber, *Level*, *QuestionText*, *VariableName*, *AnswerPartId*, *Instruction*, *AdditionalVariableName* and *AdditionalAnswerPartId*. The *PartSequenceNumber* attribute stores the sequential order of the question part in the question. The *level* attribute stores the corresponding question part level. Both of these attributes could be used to retrieve the correct structure of the question. The *QuestionText* attribute stores the text associated with the question part. The *VariableName* contains the corresponding variable name if the question part has an associated answer part. The *AnswerPartId* stores the id of the answer part from the answer part collection if the question part has one. If the question part has an instruction text associated with it, then it is stored in the *Instruction* attribute. The *AdditonalVariableName* attribute stores the corresponding variable name if the question part has an additional answer part. The *AdditionalAnswerPartId* attribute refers to the additional answer part associated with the question part if the question part has one by saving the corresponding id of the additional answer part document, stored in a separate collection.

6.2.3 Answer Part Collection

The details regarding the answer parts of different survey questions are stored in the answer part collection. This collection could be considered as the combination

of the answer part and the answer part choice entities in the conceptual data model. When we compare it with the SQL database design, we can see that the answer part collection stores the details that are distributed across the answer part, answer part choice, choice text and choice group tables in the SQL database. Figure 6.7 shows the structure of the answer part collection.

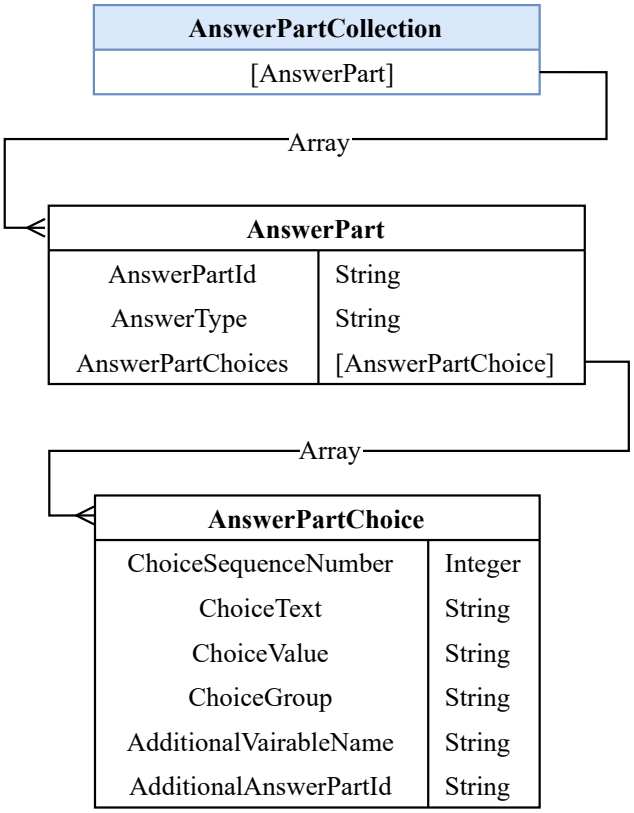


Figure 6.7: Structure of Answer Part Collection

The collection contains multiple documents called *AnswerPart* documents and, each document contains the details of an answer part. An *AnswerPart* document has three attributes: *AnswerPartId*, *AnswerType* and *AnswerPartChoices*. Each *AnswerPart* document has a unique id, and it is stored in the *AnswerPartId* attribute. This id could be used to uniquely identify the corresponding *AnswerPart* document. The *AnswerType* attribute stores the corresponding answer type of the answer part, which explains how the answer could be given. The *AnswerPartChoices* attribute contains an array of *AnswerPartChoice* documents.

If an answer part has multiple choices, then the details regarding each of these choices are saved in an *AnswerPartChoice* document. An *AnswerPartChoice* document has six attributes: *ChoiceSequenceNumber*, *ChoiceText*, *ChoiceValue*, *ChoiceGroup*, *AdditonalVariableName* and *AdditionalAnswerPartId*. When an answer part has multiple choices, it is important to save the sequential order of these choices so that the question could be correctly reconstructed. The *ChoiceSequenceNumber* attribute stores the position of the choice so that we can easily identify the sequential order of the choices. The text associated with the choice which is displayed in the survey is stored in the *ChoiceText* attribute. The *ChoiceValue* attribute stores the value of

the choice, which is used for its internal representation. Sometimes, the choices of an answer part could be grouped. In such cases, the *ChoiceGroup* attribute stores the name of the group to which the corresponding choice belongs. If the answer part has an addition answer part, then the corresponding variable name is stored in the *AdditonalVariableName* attribute and the *AdditionalAnswerPartId* attribute refers to the corresponding additional answer part which is present in a separate collection.

6.2.4 Additional Answer Part Collection

The Additional Answer Part Collection contains documents that store information regarding the additional answer parts. The additional answer parts include free texts or drop-downs that are associated with question parts or answer parts to get some additional information along with the actual answer. This collection is created by combing the details of the additional answer part and additional answer part choice entities in the conceptual data model. The structure of the additional answer part collection is shown in Figure 6.8. While analyzing the structure of the collection, we can see that the additional answer part collection combines the details in the additional answer part, additional answer part choices and choice text tables in the previously discussed SQL database design.

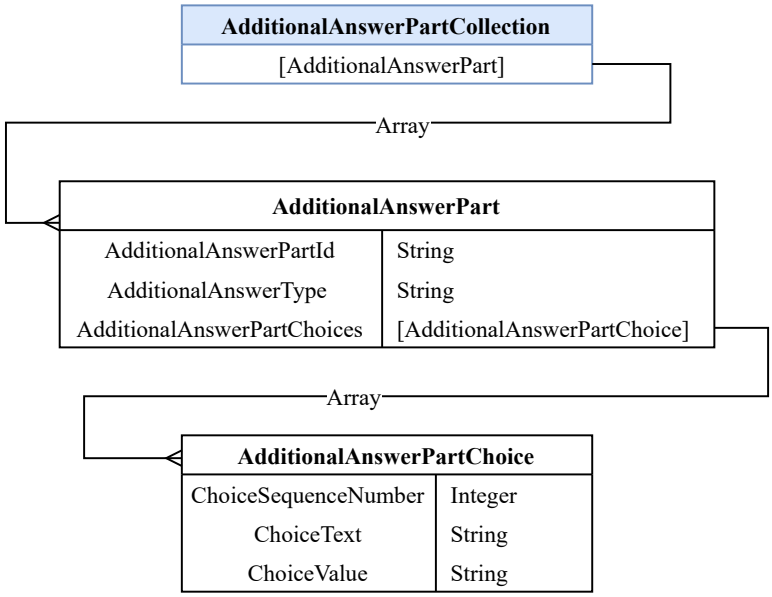


Figure 6.8: Structure of Additional Answer Part Collection

The additional answer part collection has multiple *AdditionalAnswerPart* documents. Each *AdditionalAnswerPart* document stores the details of an additional answer part and has three attributes: *AdditionalAnswerPartId*, *AdditionalAnswerType*, and *AdditionalAnswerPartChoices*. Each *AdditionalAnswerPart* document could be uniquely identified using an id which is stored in the *AdditionalAnswerPartId* attribute. The *AdditionalAnswerType* attribute stores the corresponding answer type of the additional answer part, which explains how the answer could be given. The *AdditionalAnswerPartChoices* attribute contains an array of *AdditionalAnswerPartChoice* documents.

An additional answer part might contain several choices from which the survey respondents can select one/many as their answer. Each *AdditionalAnswerPartChoice* document stores the information regarding these choices. The *AdditionalAnswerPartChoice* document has three attributes: *ChoiceSequenceNumber*, *ChoiceText* and *ChoiceValue*. The *ChoiceSequenceNumber* attribute stores the position of the choice so that we could easily identify the sequential order of the choices. The text which is displayed as the choice in the survey is saved in the *ChoiceText* attribute. The *ChoiceValue* attribute stores the value of the choice and is used for internal representation.

6.2.5 Footnote Collection

The footnote collection stores the footnote details of the survey questions. The survey questions should be stored as standalone objects so that they could be used in multiple surveys. However, the footnotes usually contain survey specific information such as how a question is related to other questions in the survey. The same question present in different surveys might have different footnote information depending on the survey it belongs. Therefore, it is not advisable to store the footnotes in the question collection as it might affect the independence of the questions. To overcome this problem, the footnote information is stored in a separate collection called footnote collection. The structure of the collection is shown in Figure 6.9. The footnote collection is an exact replication of the footnote table in the SQL database design.

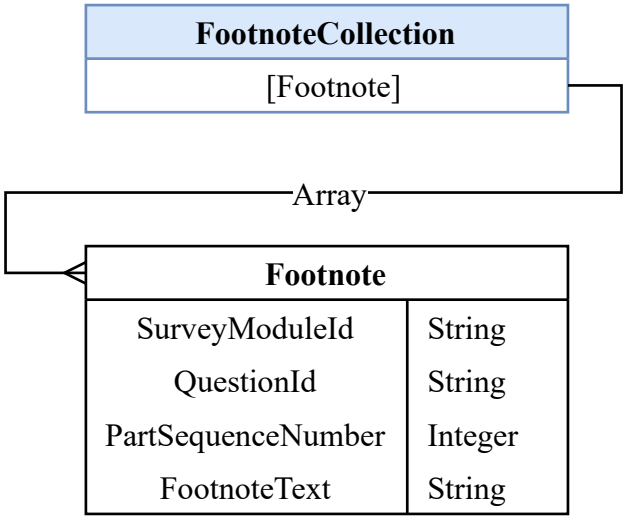


Figure 6.9: Structure of Footnote Collection

The footnote collection has multiple *Footnote* documents. Each *Footnote* document contains the information regarding the footnote of a question in a particular survey module. The document has three attributes: *SurveyModuleId*, *QuestionId*, *PartSequenceNumber* and *FootnoteText*. The *SurveyModuleId* stores the id of the survey module, which has the question with the footnote. The id of the question is stored in the *QuestionId* attribute. The *PartSequenceNumber* attribute stores the position of the question part in the question with which the footnote is associated. Finally, the *FootnoteText* stores the actual footnote text displayed in the survey.

6.3 Summary

In this chapter, we have designed database schemas for managing the survey questions in the SQL and NoSQL (MongoDB) databases. For the SQL database, the schema was designed by normalizing the conceptual data model. The conceptual data model was created by analyzing various identified requirements of the system. On the other hand, before designing the NoSQL database, we selected the data model and the database which are best suited for the survey questions and the schema was designed according to the selected database. We have selected the document-oriented data model as it could best represent the survey questions. Among different document-oriented databases, MongoDB was selected for managing the survey questions. After the database selection, different MongoDB collections were designed for storing the survey questions by de-normalizing the conceptual data model.

MongoDB offers a flexible schema which would be beneficial for the survey questions as they have a flexible structure. The flexibility of the database allows us to easily incorporate future changes and future questions with a new structure. However, the SQL database has a predefined and fixed schema which would be less beneficial in terms of incorporating future questions with a new structure. The predefined schema offers less flexibility and only incorporates questions with certain predefined formats.

SQL offers a declarative and structured querying language. The SQL query language would make data management easier as the language is popular, interactive and standardized. On the other hand, MongoDB has a JSON based querying language for data management. This query language is not as widely used as the SQL query language. Additionally, it would be difficult to manage the data shared among different collections, especially in the case of joins/look-ups that would lead to very complex queries.

In the case of data retrieval or selection queries, the SQL system might perform better than the NoSQL systems as it offers faster query processing, especially for join operations. The data retrieval from MongoDB would be very expensive as the data is shared among different collections and the joins or look-ups in MongoDB is slower as compared to the SQL system.

Avoiding redundant storage of survey questions and answer parts are important for the designed database. The SQL system is more suited for preventing data redundancies. The duplicate storage of survey questions and answer parts could be avoided by the primary and foreign key concepts of the SQL database. Furthermore, the consistency of the data is ensured by the referential integrity constraint. On the other hand, MongoDB also allows us to prevent duplicate storage of data objects by object reference, but it does not guarantee any data consistency. Thus, we need to have extra efforts at the application level to ensure data consistency.

In the case of storage space, the SQL database is expected to consume less space for storing the survey questions as it contains normalized data. However, the data stored in MongoDB is not as normalized as the data in the SQL system and therefore is expected to consume more space. However, a conclusion is only possible after

thorough evaluation as the storage space for foreign keys might also affect the results in the SQL database.

The extent to which the data is normalized in both of the databases might affect the data processing steps as well. For storing the survey questions in the SQL database, more data processing is required as the data needs to be normalized. On the other hand, fewer data processing efforts might be required for MongoDB as it stores less normalized data.

7. Loading Survey Questions into Database

In this chapter, we discuss the process of loading the survey questions, present in the portable document format (PDF), to the designed databases. The survey questions present in PDF documents need to be extracted, and the extracted texts should be processed so that they can be stored in the SQL and MongoDB databases. To this end, we need to identify different defined entities of each question such as survey, survey module, question parts, answer parts, additional answer parts, footnotes and choices should be identified from the PDF files, and these entities should be processed according to the schema of the target database so that they can be permanently stored in the database. The processing of data mainly involves data normalization. The extent to which the data is normalized is less in the case of the MongoDB database as compared to the SQL database.

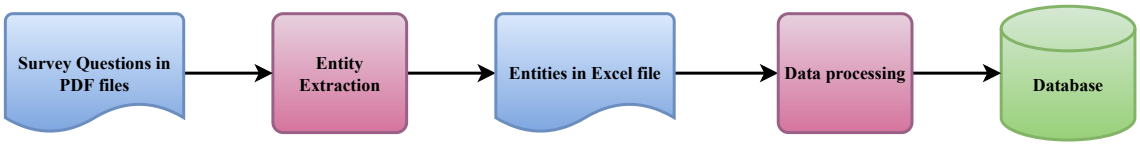


Figure 7.1: Data Loading Process

The flow of the data loading process is shown in Figure 7.1. The survey questions are initially present in PDF files. Different entities are extracted from these PDF files, and these identified entities are stored in an Excel file. The data in the Excel file is processed using an ETL job which will finally load the data into the database. Two processes involved in data loading are entity extraction and data processing. These processes are explained in detail as follows.

7.1 Entity Extraction

The entity extraction process involves the identification of different defined entities for each survey question present in the PDF documents. The identified entities are

then stored in an Excel file for further processing. The different entities extracted for each question are the survey details of the question, survey module details of the question, question parts in the question, answer parts associated with the question parts, choices of answer parts, additional answer parts, choices of additional answer parts and footnotes present in the question.

The reason for selecting the Excel file to store the question entities is that we already have an Excel file with some of the entities available at DZHW. These Excel files contain the corresponding variable names of the survey questions and the question texts associated with them. These Excel files are expanded by adding the remaining question entities to make them suitable for our use case. For example, consider an example question from the analyzed survey shown in Figure 7.2. The data present in the already available Excel file corresponding to this question is shown in Figure 7.3. The remaining entities should be identified and added to the Excel file. In this thesis work, the entity extraction process is carried out manually. The survey question in the PDF files are analyzed manually, and the different entities are identified and stored in an Excel file.

A_14 Wie häufig führen Sie pro Woche die folgenden Pflegetätigkeiten aus?

	nie	sehr selten				sehr häufig
Besorgungen und Erledigungen außer Haus, z. B. Behördengänge	☐	☐	☐	☐	☐	☐
Haushaltsführung, Versorgung mit Mahlzeiten und Getränken	☐	☐	☐	☐	☐	☐
einfachere Pflegetätigkeiten, z. B. Hilfe beim An- und Auskleiden, Waschen, Kämmen und Rasieren	☐	☐	☐	☐	☐	☐
schwierigere Pflegetätigkeiten, z. B. Hilfe beim Umbetten, Stuhlgang	☐	☐	☐	☐	☐	☐
Etwas anderes, und zwar: *(offene Angabe)*	☐	☐	☐	☐	☐	☐

Figure 7.2: Example Question for Data Loading

Seitenidentifier	Variablenname	Fragetext	Itemtext
A_14	pflegt (pflegt1; pflegt2; pflegt3; pflegt4; pflegt5; pflegt5o)	Wie häufig führen Sie pro Woche die folgenden Pflegetätigkeiten aus?	(pflegt1): Besorgungen und Erledigungen außer Haus, z. B. Behördengänge (pflegt2): Haushaltsführung, Versorgung mit Mahlzeiten und Getränken (pflegt3): einfachere Pflegetätigkeiten, z. B. Hilfe beim An- und Auskleiden, Waschen, Kämmen und Rasieren (pflegt4): schwierigere Pflegetätigkeiten, z. B. Hilfe beim Umbetten, Stuhlgang (pflegt5): Etwas anderes, und zwar:

Figure 7.3: Already Available Entities of the Question

The question in Figure 7.2 is analyzed manually, and different entities are extracted from it. These entities are then saved into an Excel file. The Excel file that contains all the entities of the analyzed question is shown in Figure 7.4. The first column in the Excel file contains the details of the survey. The second column contains the details about the survey module of the question. The question number of the

Survey	SurveyModule	QuestionNumber	PartSequence	Level	Variable name	Additional Variablename	QuestionPartText	Instruction	Footnote	Answer Type	AnswerPart
Die Studierendenbefragung in Deutschland	Grundprogramm Modul A	18	1	0			Wie häufig führen Sie pro Woche die folgenden Pflegetätigkeiten aus?				
			2	1	pflegt1		Besorgungen und Erledigungen außer Haus, z. B. Behördengänge			Range	1;1;nie 2;2;sehr selten 3;3; 4;4; 5;5; 6;6;sehr häufig
			3	1	pflegt2		Haushaltsführung, Versorgung mit Mahlzeiten und Getränken			Range	1;1;nie 2;2;sehr selten 3;3; 4;4; 5;5; 6;6;sehr häufig
			4	1	pflegt3		einfachere Pflegetätigkeiten, z. B. Hilfe beim An- und Auskleiden, Waschen, Kämmen und Rasieren			Range	1;1;nie 2;2;sehr selten 3;3; 4;4; 5;5; 6;6;sehr häufig
			5	1	pflegt4		schwierigere Pflegetätigkeiten, z. B. Hilfe beim Umbetten, Stuhlgang			Range	1;1;nie 2;2;sehr selten 3;3; 4;4; 5;5; 6;6;sehr häufig
			6	1	pflegt5	pflegt5o	Etwas anderes, und zwar:			Range	1;1;nie 2;2;sehr selten 3;3; 4;4; 5;5; 6;6;sehr häufig

Figure 7.4: Identified Entities of the Question

question in that particular survey module is present in the third column. From Figure 7.2, we can see that the question has six question parts. The details regarding these question parts are present in columns 4-9. Different attributes of the question part entity such as part sequence number, level, variable name, additional variable name, question part text and instruction are stored in these columns. The footnote information associated with each question part is present in the tenth column. If the question part has an answer part, then the corresponding answer type is present in the eleventh column. Finally, the choices associated with the answer part are present in the last column.

There is no column for the additional answer part entity in the Excel file. This is because all the additional answer parts in the analyzed survey questions are free texts. The free text answer type does not have any choices associated with it. Therefore, we do not need separate columns for storing the answer type or choices of the additional answer parts. If there is an additional variable name associated with the question part, then the question part has an additional answer part of type free text. For example, in the analyzed question, the last question part has an additional free text associated with it. This free text has a corresponding variable name (pflegt5o) stored in the corresponding column. By reading the additional variable name only, we can say that the last question part has an additional answer part of type free text associated with it.

If a question part has an answer part associated with it, then the corresponding answer type is stored in the eighth column and the answer part choices, if any, are present in the last column. Each non-empty cell in the last column may contain multiple choices of the corresponding answer part. Since multiple choices are saved in a single cell, the data in each of these cells are arranged in a particular format so that the different choices could be easily retrieved. Each record in the cell contains multiple rows, and each row contains the details of a single choice. A choice can

have five attributes as part sequence number, choice value, choice text, additional variable name and choice group. Each row in the cell contains values for all these attributes separated using a semicolon (;). The structure of each cell record is shown in Figure 7.5. The choice sequence number, which explains the position of the choice, is present in the first place. In the second place, the value of the choice, used for the internal representation, is given. The text that is displayed as the choice is present in the third position. If the choice has an additional answer part associated with it, then the corresponding variable name is present in the fourth position. Finally, if the choices are grouped, the corresponding group details to which the choice belongs is stored in the last position. The choice text, additional variable name and choice group are optional fields, and if they do not have any values, these fields could be ignored as well. For example, all the answer parts in Figure 7.4 have the same answer part with six choices. None of the choices has an additional answer part, thus the additional variable name field (fourth field) is empty for all choices. Additionally, the choices are not grouped, and hence the choice group field (fifth field) is also empty for all the choices. Furthermore, the third, fourth and fifth choices have no choice texts, and thus the choice text field (third field) is empty for these choices.

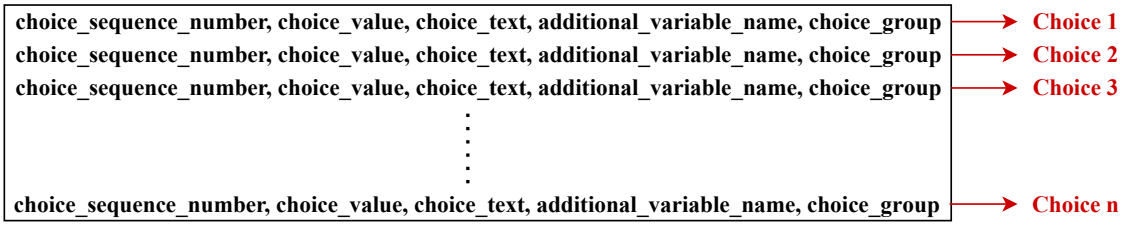


Figure 7.5: Structure of an Answer Part Choices Record in the Excel file

After extracting all the entities and storing these entities in an Excel file according to the defined format, the data in the Excel files are processed by an ETL job, which loads them into the database after processing.

7.2 Data Processing

In the data processing step, the extracted entities of the survey questions are processed according to the schema of the database to which the data is to be saved. The main task involved in the data processing step is data normalization according to the target database. The processed data is then saved into the corresponding database. In this thesis work, ETL jobs are used to process the extracted entities. The ETL job will load the survey question entities from the Excel files, and process them according to the final database schema, and finally save the processed data into the database. The ETL jobs for processing the extracted entities are designed using Talend Open Studio.

The SQL and MongoDB databases have different schema, and the extent to which the data is normalized is also different in these databases. The data need to be less normalized in the case of the MongoDB database as compared to the SQL database. For example, while saving a survey question into the SQL database we need to save the question part texts and instructions in separate tables, and their

primary keys should be saved in the corresponding question part record for reference. However, while storing the same survey question in the MongoDB database, we do not need to store the question part texts and instructions separately. Instead, they could be stored in the same single collection along with their corresponding question parts. Therefore, we need separate ETL jobs for saving the data into the SQL and MongoDB databases. They are discussed as follows.

7.2.1 Data Processing for SQL Database

For the SQL database, the data should be normalized more as compared to the MongoDB database. The question part entities are processed by an ETL job, and the processed data is finally stored into the MySQL database. The flow of the ETL job is explained in Algorithm 1. Each step in the algorithm is also discussed in detail as follows.

Algorithm 1 Algorithm Explaining the flow of ETL job for saving the survey questions to SQL Database

STEP 1: Load the Data from the Excel File
STEP 2: Extract Survey and Survey Module Details
STEP 3: Save Survey Details to Database
STEP 4: Save Survey Module Details to Database
STEP 5: Extract Footnote Details and Question Numbers
STEP 6: For Each Question Number
- **STEP 6.1:** Extract the Corresponding Question Details
- **STEP 6.2:** Extract Question Part Texts, Instructions, Variable Names and Additional Variable Names
- **STEP 6.3:** Save Question Part Texts, Instructions and Additional Variable Names to Database
- **STEP 6.4:** For Each Variable Name
  - **STEP 6.4.1:** Extract Answer Part Details corresponding to the Variable Name
  - **STEP 6.4.2:** Convert Answer part Choice Record into Multiple Columns
  - **STEP 6.4.3:** Extract Answer Type, Choice Texts, Additional Variable Names and Choice Groups
  - **STEP 6.4.4:** Save Choice Texts, Additional Variable Names and Choice Groups to Database
  - **STEP 6.4.5:** Create Partial Answer Part Choice Records
  - **STEP 6.4.6:** Get Answer Part Id
  - **STEP 6.4.7:** Create and Save Complete Answer Part Choice Records to Database
- **STEP 6.5:** Create Partial Question Part Records
- **STEP 6.6:** Get Question Id
- **STEP 6.7:** Create and Save Complete Question Part Records to Database

STEP 7: Save the Footnote Details to Database
STEP 8: Save the Question Number to Database

1. **Load the Data from the Excel File:** The data present in the Excel file is loaded by the ETL job. For this the *tFileInputExcel* component of Talend Open Studio is used. The schema of the Excel file should be also defined for the correct retrieval of the data. Figure 7.6 shows the *tFileInputExcel* component in the Talend job and Figure 7.7 shows its schema.

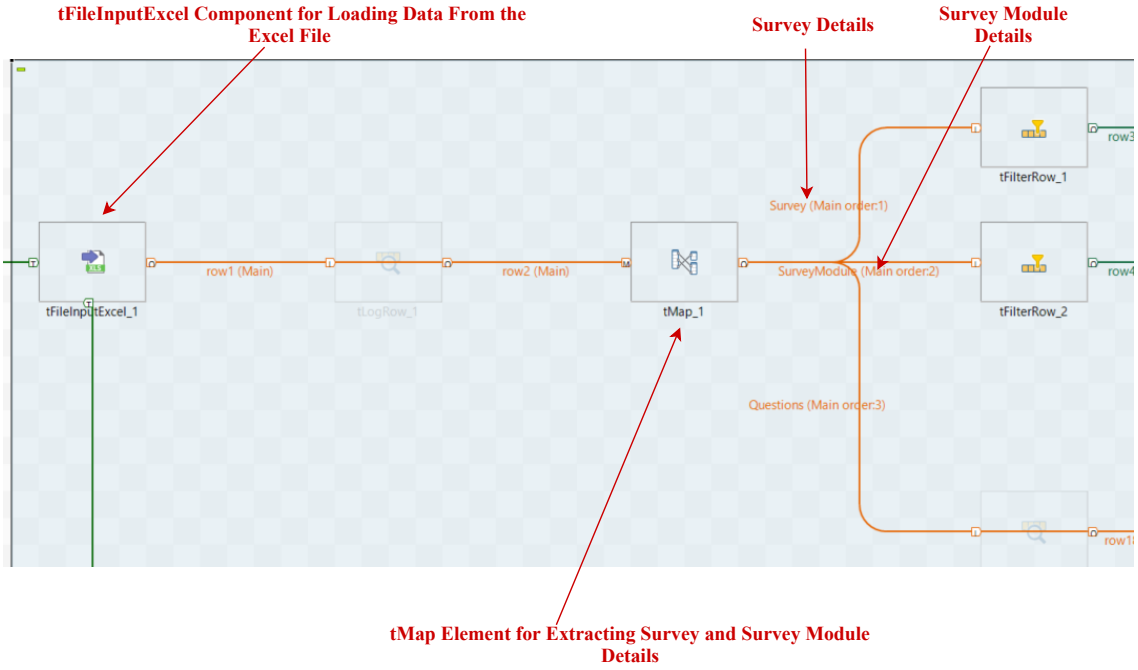


Figure 7.6: Talend Components for Loading the Input Data and Extracting the Survey and Survey Module Details

2. **Extract Survey and Survey Module Details:** After loading the data from the Excel file, the survey and survey module details are extracted from it. The *tMap* component of Talend is used for extracting these data. The extracted survey and survey module details are then processed separately. Figure 7.6 shows the *tMap* component in the Talend job, which extracts the survey and survey module details, and Figure 7.8 shows the structure of the *tMap* component.

3. **Save Survey Details to Database:** The survey details extracted in step 2 are inserted into the database in this step. After the insertion, the corresponding survey id is saved in the buffer for future use. The ETL job components handling the insertion of the survey details into the database are shown in Figure 7.9. Initially, the survey details from step 2 are filtered using the *tFilterRow* component to filter out the empty rows. The empty rows are created while extracting the survey details in 2. The survey details are common to all the questions and are present in only one row in the input Excel file. Therefore, while extracting the survey details there will be empty rows in the data. The *tFilterRow* component filters out these empty rows in the extracted data. The survey details are then inserted into the *survey* table using the *tDBOutput* component. The creation of duplicate records is also prevented

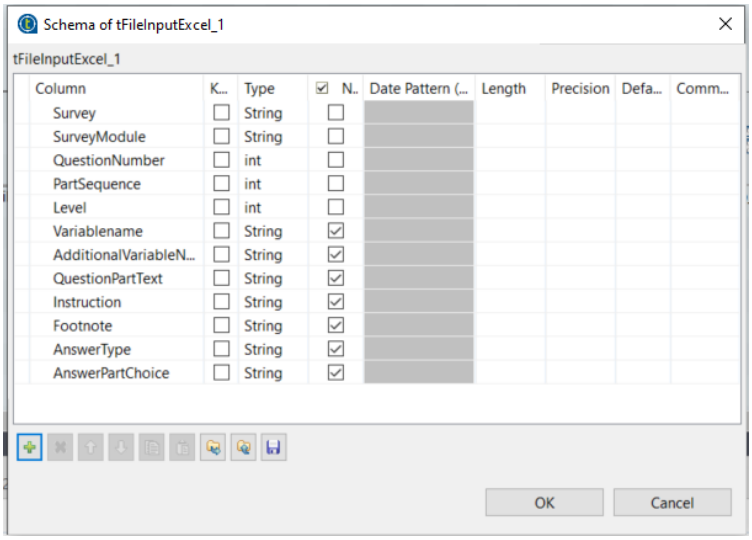


Figure 7.7: Schema of the *tFileInputExcel* Component

by the *tDBOutput* component. After the insertion, the corresponding auto-generated id of the survey details is selected from the database using the *tDBRow* component. The result from the database is of type object, and the type should be converted into an integer as the survey id is an integer. This is done the by the *tParseRecordSet* component. Finally, the id of the survey is saved in the buffer using the *tHashOutput* (renamed as SurveyId in Figure 7.9) component.

4. **Save Survey Module Details to Database:** This step is to save the survey module details, extracted in step 2, into the *survey module* table in the database. Each record in the survey module table needs the corresponding id of the survey to which the survey module belongs. For this, the survey id from step 3 is added to the survey module details using a *tMap* component. The structure of the *tMap* component is shown in Figure 7.10. Finally, the survey module details are inserted into the database. The procedure used for the insertion is similar to the one used in step 3. After the insertion, the auto-generated survey module id is stored in the buffer for future use.

5. **Extract Footnote Details and Question Numbers:** In this step, the footnote details and question numbers are extracted from the input data. In the case of footnote details, the corresponding question numbers and part sequence numbers of the question parts to which the footnotes are associated are also saved. These additional details are needed to correctly identify the position of the footnote when it is later saved in the database. The extracted details are stored in the buffer for future use. Like in step 2, the ETL job uses the *tMap* component for the extraction of required data.

6. **Iterate Over Each Question Numbers:** In this step, the ETL job loops over the question numbers stored in the buffer in step 5. For example, if there are a hundred questions, then the loop will run hundred times. For each iteration, a question number is taken, and the corresponding question is processed. After

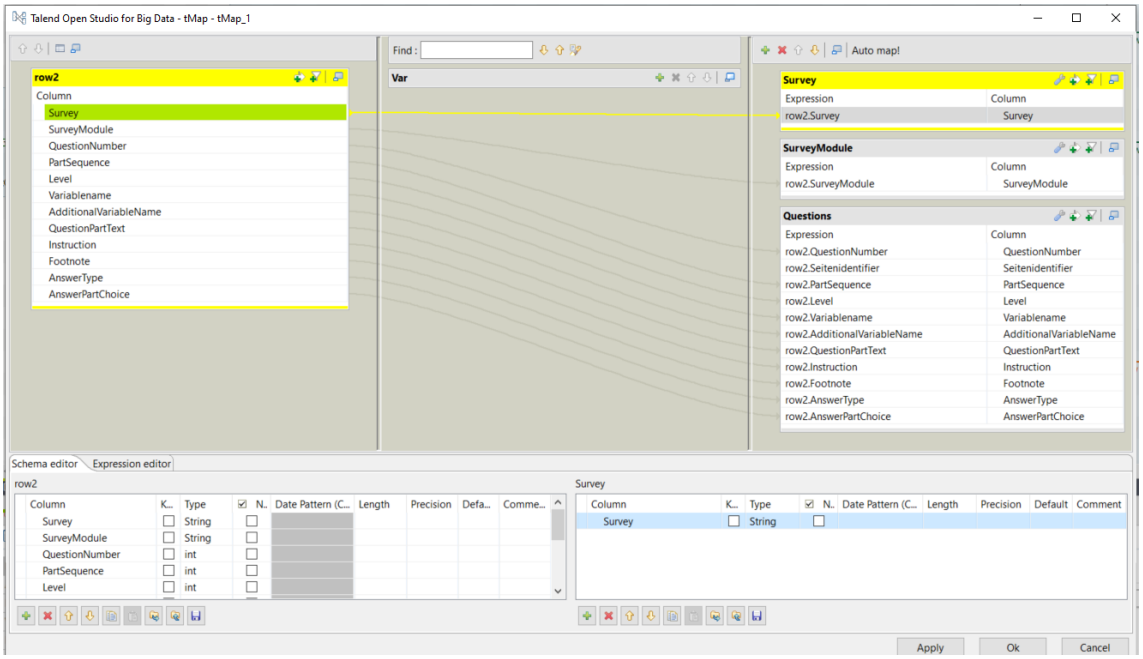


Figure 7.8: Structure of *tMap* component used for Extracting Survey and Survey Module Details

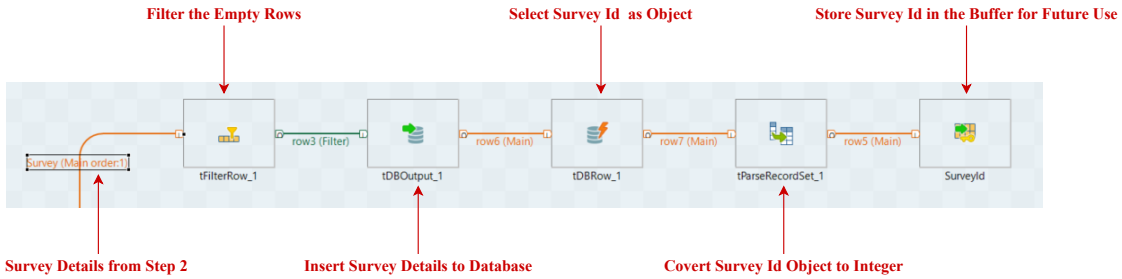


Figure 7.9: Talend Components for Saving the Survey Details into the Database

saving each question to the database, the question numbers are saved in the buffer along with the auto-generated question ids for future use. The Talend components for this step is shown in Figure 7.11. Initially, the question numbers from step 5 are read using a *tHashInput* (renamed as *QuestionNumbers* in Figure 7.11) component. The *tFlowToIterate* component then iterates over the input question numbers and for each iteration the corresponding question number is sent to the *tRunJob* component. The *tRunJob* component processes the question details corresponding to the input question number. Finally, after saving the question in the database the corresponding question id is stored in the buffer along with the question number for future use using the *tHashOutput* (renamed as *QuestionNumberWithIds* in Figure 7.11) component. The tasks executed in the loop are discussed in the sub-tasks of this step.

6.1. **Extract the Corresponding Question Details:** The input data contains details of multiple questions. From this data, the details of the question with the question number from step 6 is extracted in this step for further processing. For this, we perform an inner join with the question number

Figure 7.10: Structure of *tMap* Component for Adding the Survey Id to the Survey Module Details

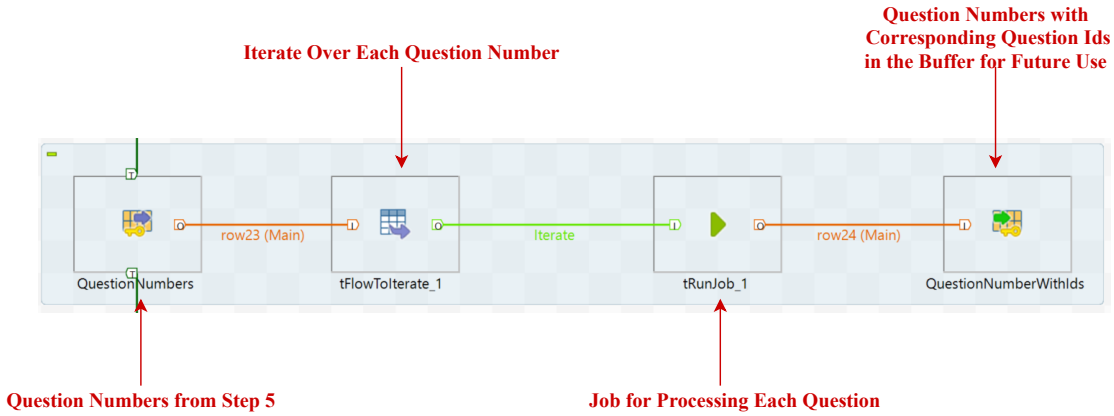


Figure 7.11: Talend Component to Iterate Over the Question Numbers

from step 6 and the question number column of the input data. The result will be the question details corresponding to the question number. This could be done using a *tMap* component. The structure of the *tMap* component is shown in Figure 7.12.

6.2. **Extract Question Part Texts, Instructions, Variable Names and Additional Variable Names:** After getting the details of a single question in step 6.1., the question part texts, instructions, variable names and additional variable names from each question parts in the question are extracted. This is done by using a *tMap* component. The procedure is similar to that of the one discussed previously in step 2 and step 5.

6.3. **Save Question Part Texts, Instructions and Additional Variable Names to Database:** The question part texts, instructions and additional variable names are saved into the database in this step. The question part texts and instructions are saved into the corresponding tables. After the insertion, the question part texts and instructions are stored in the buffer along with their auto-generated ids for future use. For each additional variable name, a record is inserted, if not present, in the *additional answer part* table with answer type as free text. After the insertion, the

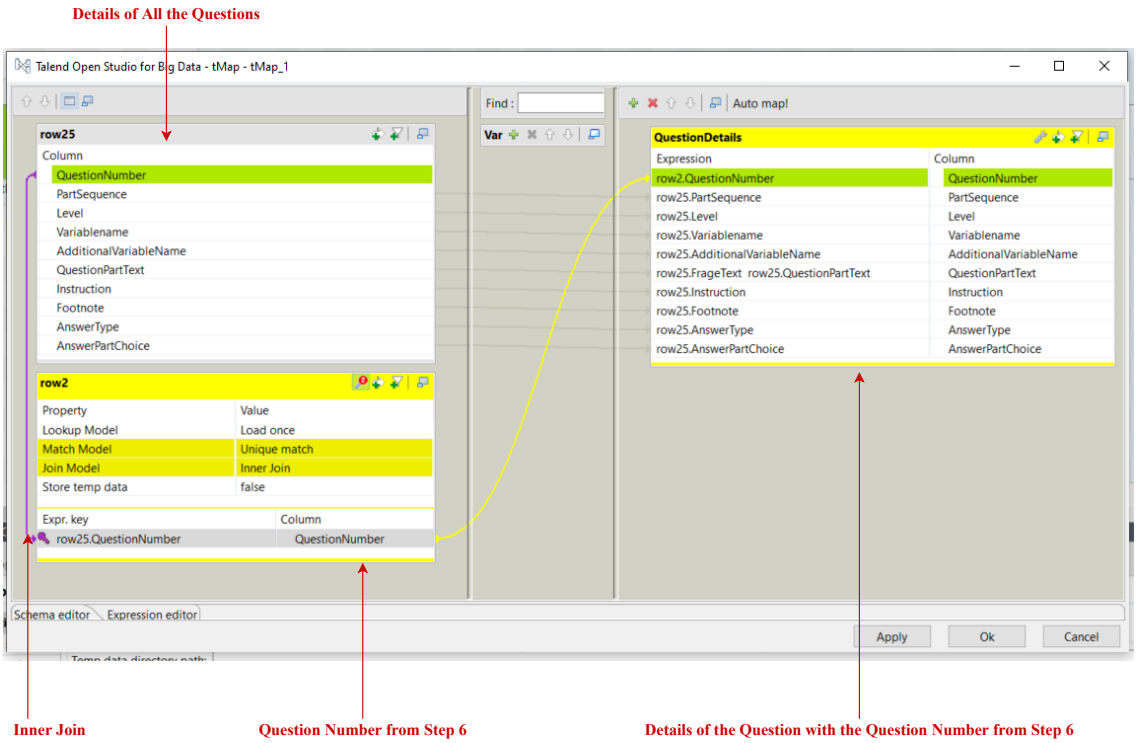


Figure 7.12: Structure of the *tMap* component for Extracting the Details of a Single Question Using the Question Number

additional variable name is stored in the buffer with its corresponding additional answer part id. The procedure used for insertion is similar to the one used in step 3.

6.4. **Iterate Over Each Variable Name:** In this step, the ETL job iterates over each variable name extracted in step 6.2.. For each iteration, a variable name is taken, and the corresponding answer part is extracted. This answer part is then processed and saved into the database. At the end of each iteration, the variable name is saved in the buffer along with the auto-generated id of the corresponding answer part. The ETL flow used for looping over the variable names is similar to the one used in 6. The tasks executed in the loop are discussed in the sub-tasks of this step.

6.4.1. **Extract Answer Part Details corresponding to the Variable Name:** The answer part details corresponding to the variable name from step 6.4. is extracted in this step. The procedure used in this step is similar to the one used in step 6.1.. Using a *tMap* component we perform an inner join of the input variable name with the variable name field of the input data. The result contains the answer part details (AnswerType and AnswerPartChoice columns from the input data) corresponding to the input variable name. The structure of the *tMap* component is shown in Figure 7.13.

6.4.2. **Convert Answer part Choice Record into Multiple Columns:** If the answer part has choices, their details are present as a single record of the AnswerPartChoice column of the input Excel file. The format

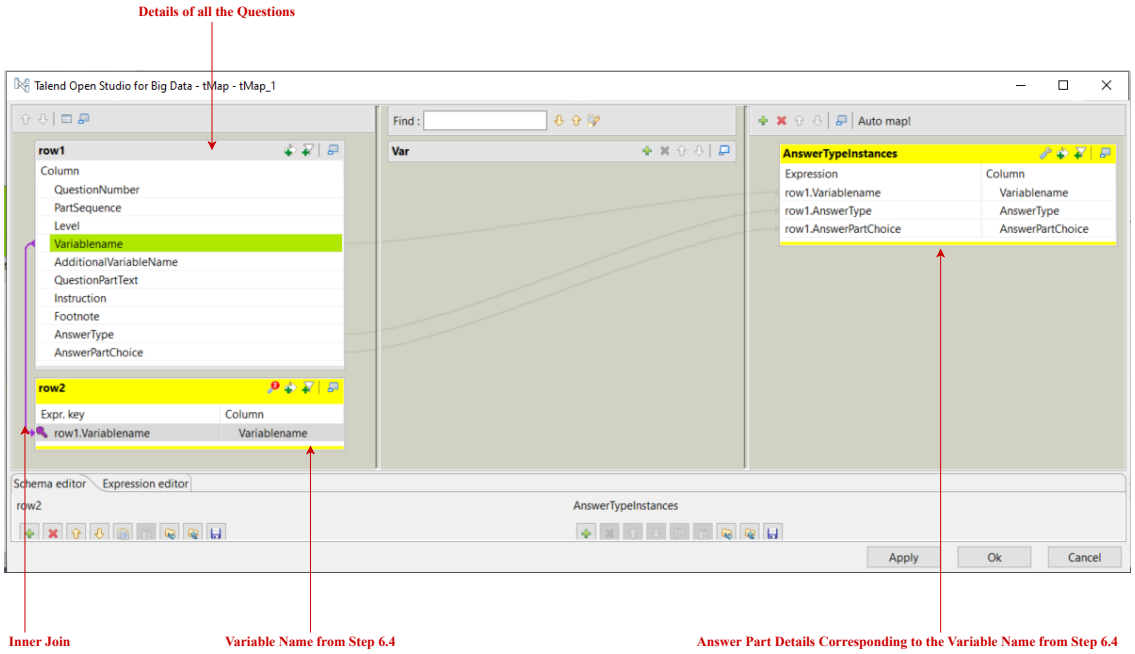


Figure 7.13: Structure of the *tMap* component for Extracting the Details of a Single Answer Part Using the Variable Name

in which the details of the answer part choices are saved is shown in Figure 7.5. In this step, we extract each field in this record and store them separately. The Talend job flow for this is shown in Figure 7.14. Initially, the record is split into multiple rows such that each row contains the details of a single choice. This is done by using the *tNormalize* component in Talend. Next, the different attributes of the choices, which are separated using the semicolon (;), are extracted into multiple columns using the *tExtractDelimitedFields* component. An example for the input data, its intermediate result and the final output are also given in Figure 7.14.

6.4.3. **Extract Answer Type, Choice Texts, Additional Variable Names and Choice Groups:** The next step is to extract the answer type, choice texts, additional variable names and choice groups from the data generated as the output of step 6.4.2.. This is done by using a *tMap* component, and the procedure is similar to the ones discussed in step 2.

6.4.4. **Save Choice Texts, Additional Variable Names and Choice Groups to Database:** The extracted choice texts and choice groups are saved into the corresponding tables in the database. After the insertion, the choice texts and choice groups are stored in the buffer along with their auto-generated ids for future use. For each additional variable name, a record is inserted, if not present, in the *additional answer part* table with answer type as free text. After the insertion, the additional variable name is stored in the buffer with its corresponding additional answer part id. The procedure used for insertion is similar to the one used in step 3.

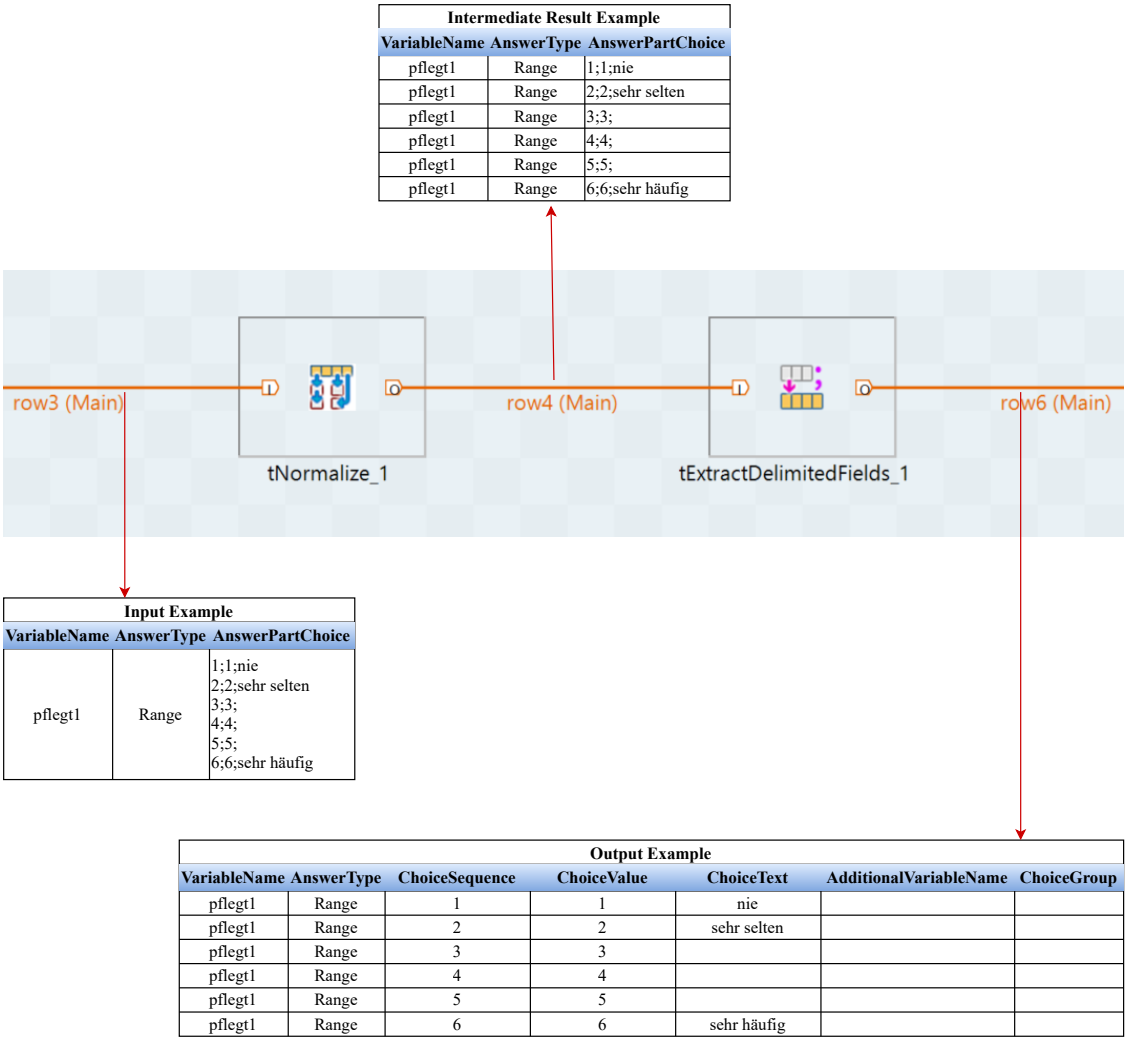


Figure 7.14: Talend Components for Processing the Answer Part Choice Record

6.4.5. **Create Partial Answer Part Choice Records:** This step is to create the partial answer part choice records. A partial answer part choice record consists of all the fields in the answer part choice table except the answer part id. Therefore, it contains the choice sequence number, choice value, choice text id, choice group id, additional variable name and additional answer part id. The structure of the partial records are shown in Figure 7.15. We have the choice sequence numbers, choice values and additional variable names from the input data. Additionally, the corresponding ids of the choice texts, choice groups and additional answer parts can be taken from the buffer as they are saved in step 6.4.4..

6.4.6. **Get Answer Part Id:** This step is to get the id for the new answer part. If the answer part is already present in the database, we can take the existing id. Otherwise, we need to create a new record in the answer part table using the answer type from step 6.4.3.. For this, we save the partial answer part choice records from step 6.4.6. into a temporary database table. Then, a MySQL function checks if there

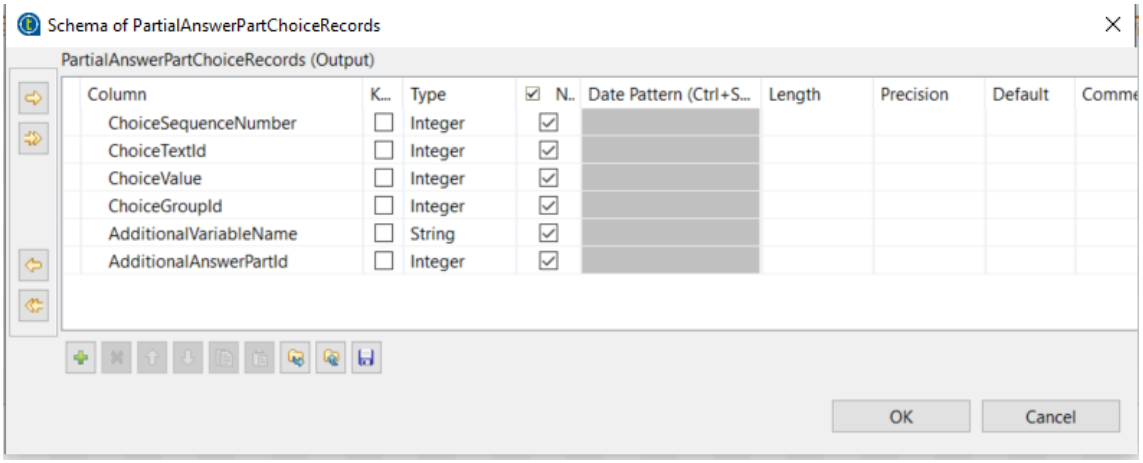


Figure 7.15: Structure of the Partial Answer Part Choice Record

exists an answer part in the database with the given answer type and choices. The answer type from step 6.4.3. is given as the input to the MySQL function, and the choices are present in the temporary table. If already present, then we take the existing answer part id. Otherwise, the function creates a new record in the answer part table and return its id. The answer part id is stored in the buffer along with the variable name from step 6.4. for future use. If the answer part is already present in the database, then the step 6.4.7. could be ignored.

6.4.7. **Create and Save Complete Answer Part Choice Records to Database:** The answer part id from step 6.4.6. is added to the partial answer part records from step 6.4.6. to get the complete answer part choice records. The structure of the records is shown in Figure 7.16. These records are then inserted into the *answer part choice* table in the database. The insertion could be done similar to the way it is done in step 3.

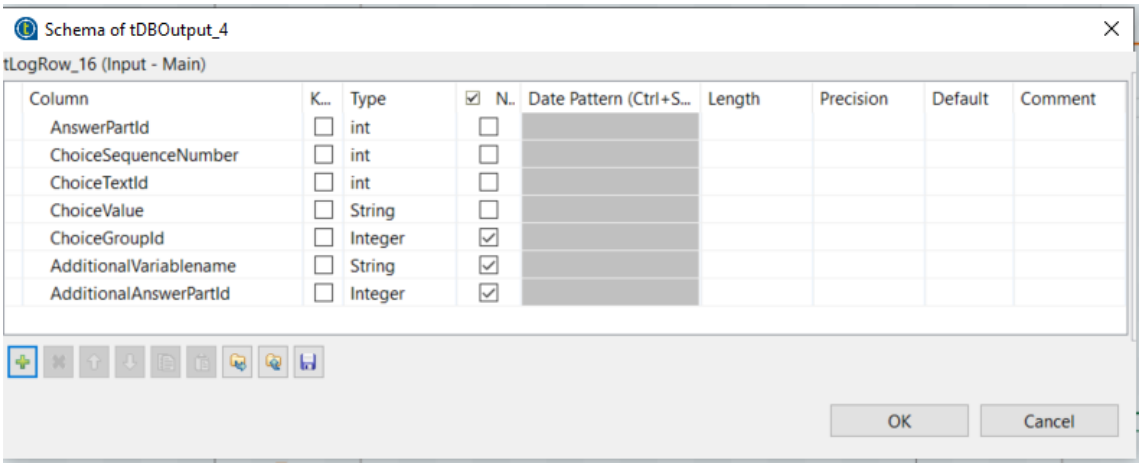


Figure 7.16: Structure of the Complete Answer Part Choice Records

6.5. **Create Partial Question Part Records:** After inserting the answer parts of the question into the database and getting their corresponding answer part ids, the next step is to save the question parts to the database. For this, the ETL job firstly creates the partial question part records. A partial question part record consists of all the fields in the question part table except the question id. Thus, it contains the part sequence number, level, question part text id, question part instruction id, variable name, answer part id, additional variable name and additional answer part id. The structure of the partial records is shown in Figure 7.17. The ETL job now has the part sequence numbers, level, variable names and additional variable names from the input data. Additionally, the corresponding ids of the question part texts, question part instructions, answer parts and additional answer parts can be taken from the buffer as they are saved in step 6.3. and 6.4..

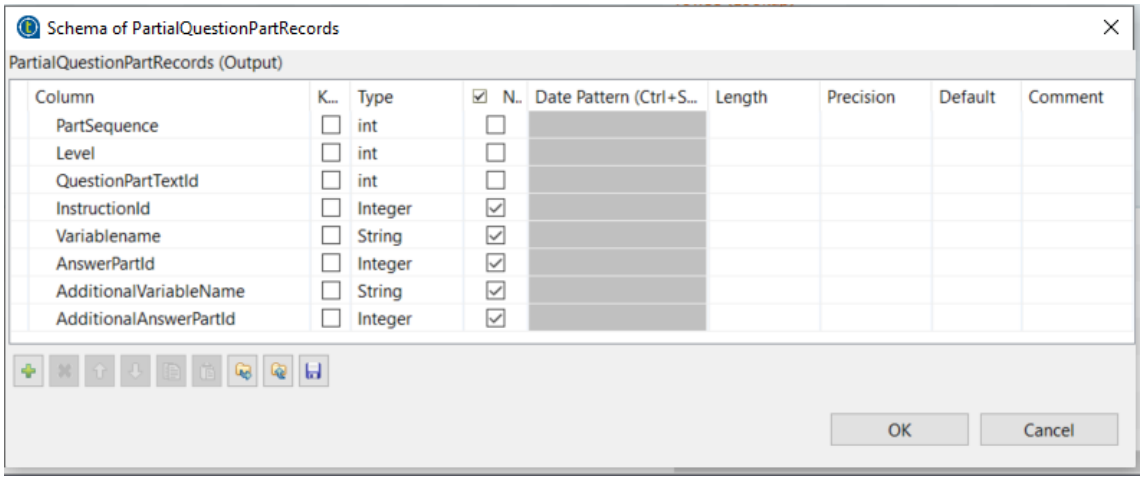


Figure 7.17: Structure of the Partial Question Part Records

6.6. **Get Question Id:** After creating the partial question part records, the ETL job checks whether the question is already present in the database. For this, the ETL job stores the partial question records from step 6.5. into a temporary database table. Then, a MySQL function checks if there exists the same question in the database. If there exists a question with the same question parts saved in the temporary table, then the ETL job takes the existing question id. Otherwise, the function creates a new record in the question table and returns the corresponding question id. The question id is stored in the buffer along with the question number from step 6 for future use. If the question is already present in the database, then the step 6.7. could be ignored.

6.7. **Create and Save Complete Question Part Records to Database:** After getting the new question id in step 6.6., it is then added to the partial question part records from step 6.5. to create the complete question part records. The structure of the records is shown in Figure 7.18. These records are then inserted into the *question part* table in the database. The insertion could be done similar to the way it is done in step 3.

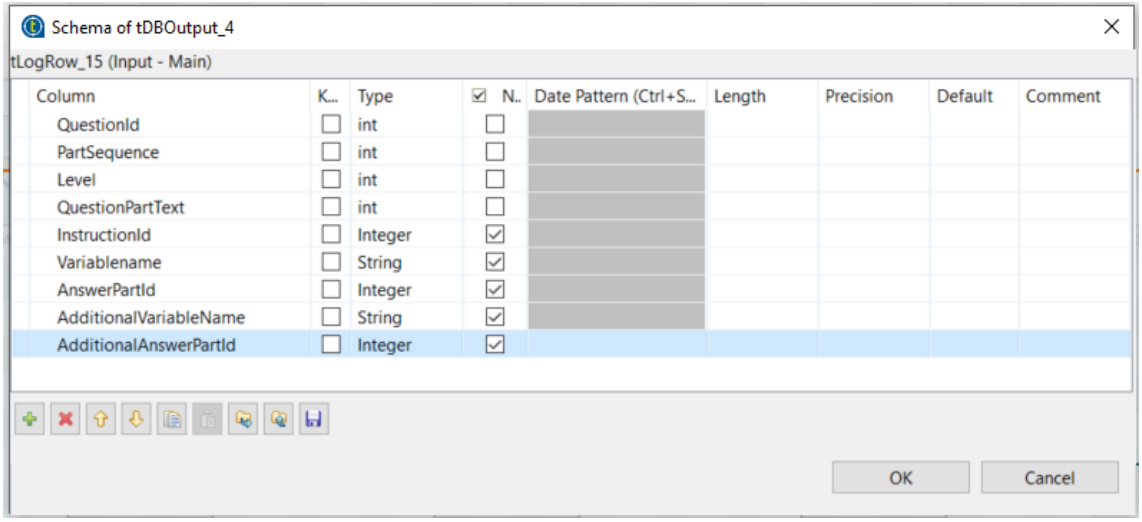


Figure 7.18: Structure of the Complete Question Part Records

7. **Save the Footnote Details to Database:** The footnote details saved in the buffer in step 5 contains the footnote, question number and the part sequence number of the question part with which the footnote is associated. At first, the ETL job adds the survey module id from step 4 to the footnote details. Then, the question numbers are replaced with the corresponding question ids from step 6. Now, the footnote details could be inserted into the *footnote* table in the database. The structure of the *footnote* table records is shown in Figure 7.19. These records are then inserted into the footnote table in the database. The insertion could be done similar to the way it is done in step 3.

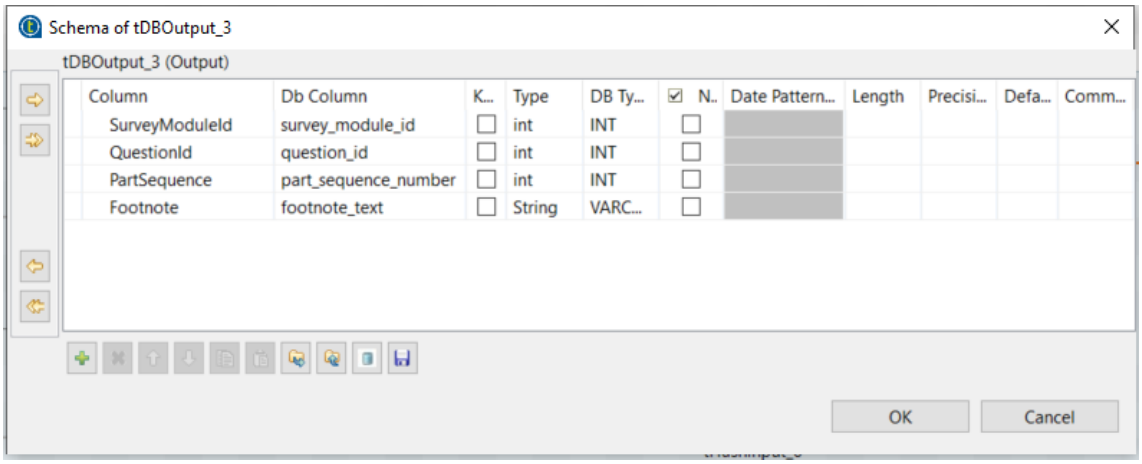


Figure 7.19: Structure of the Footnote Records

8. **Save the Question Number to Database:** In this step, the ETL job saves the question numbers of the questions to the 'survey module question' table in the database. We use the question numbers with corresponding question ids from step 6. The ETL job then adds the survey module id from step 4 to these records. The resultant records are then inserted into the *survey module*

question table of the database. The structure of the table records is shown in Figure 7.20.

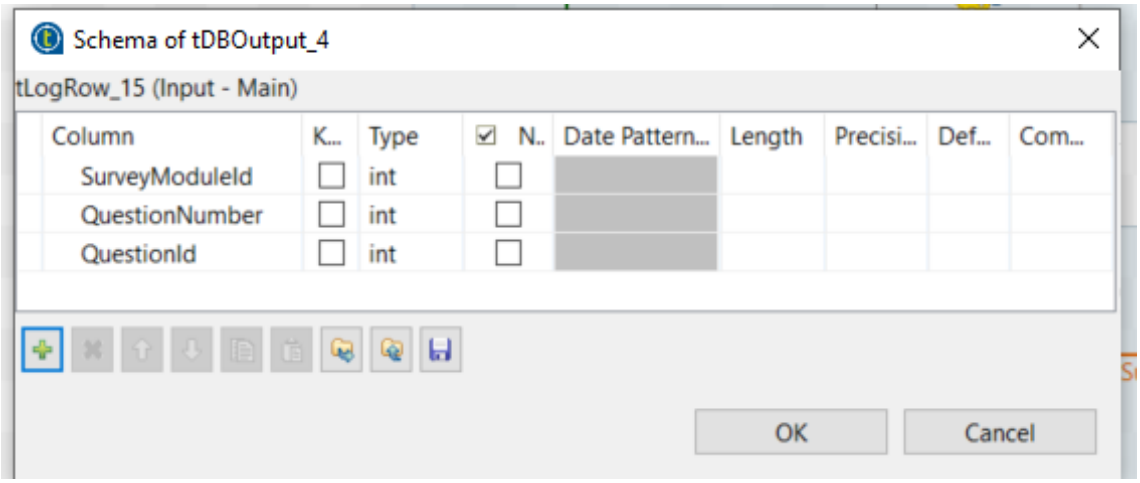


Figure 7.20: Structure of the Records for 'Survey Module Question Table' in the Database

7.2.2 Data Processing for MongoDB Database

In the case of the MongoDB database, the data only needs to be less normalized as compared to the SQL database. Since the data is processed differently, we use a separate ETL job to process the extracted entities of the survey questions and store them in the MongoDB database. The flow of the ETL job is explained in Algorithm 2.

Some of the steps in this algorithm are the same or similar as compared to the steps in the previous algorithm for the SQL database. The steps are step1, step 2, step 3, step 3.1, step 3.2, step 3.4, step 3.4.1, step 3.4.2 and step 3.4.3. Since the operations performed in these steps are discussed in detail in Section 7.2.1, we only focus on the remaining steps in this section. The only change in these steps is that the extracted survey and survey module details are saved as global variables for future use instead of storing them in the buffer as done in the previous algorithm. The concept of the global variables is explained in step 3.4.6. Furthermore, the step 3.4.4 and step 3.5 are similar to step 3.3 and step 3.4.5 respectively. Hence, they are also not explained in detail. The remaining steps are discussed in detail as follows.

3.3 **Create and save documents in the additional answer part collection for each additional variable name:** If the question has additional variable names associated with any of the question parts, then it means that the corresponding question parts have an additional answer part of type free text. In this case, the ETL job creates a corresponding document in the additional answer part collection. The flow of the ETL job that saves a document into the additional answer part collection is shown in Figure 7.21. Initially, all the question part records with an additional variable name are filtered out using a *tFilterRow* component. Then, a field called answer type is created for each of these records

Algorithm 2 Algorithm Explaining the flow of ETL job for saving the survey questions to MongoDB Database.

STEP 1: Load the data from the Excel file.
STEP 2: Extract survey details, survey module details, footnote details and question numbers.
STEP 3: For each question number.
STEP 3.1: Extract the corresponding question details.
STEP 3.2: Extract variable names and additional variable names.
STEP 3.3: Create and save documents in the additional answer part collection for each additional variable name.
STEP 3.4: For each variable name.
STEP 3.4.1: Extract answer part details corresponding to the variable name.
STEP 3.4.2: Convert answer part choice record into multiple columns.
STEP 3.4.3: Extract additional variable names.
STEP 3.4.4: Create and save documents in the additional answer part collection for each additional variable name.
STEP 3.4.5: Get additional answer part ids.
STEP 3.4.6: Create and save answer part collection document from the answer part details.
STEP 3.4.7: Get the answer part id.
STEP 3.5: Get additional answer part ids.
STEP 3.6: Create and save question collection document from the question details.
STEP 3.7: Get the Question Id.
STEP 4: If a survey document is not present with the input survey details in database, then create a document in the survey collection.
STEP 5: If the survey document contains a survey module with the input survey module details, then insert the survey module details to the survey document.
STEP 6: Insert the question details into the survey document.
STEP 7: Create and save footnote collection documents from the footnote details.

with the value as 'free text' using a *tMap* component. Since the additional answer part does not have choices, the document only has the answer type field. The document is then inserted into the 'additional answer part' collection in the MongoDB database. The insertion is done using the *tMongoDBOutput* component in Talend. The insertion of duplicate documents is handled by the *tMongoDBOutput* component itself. The id of the document is automatically generated by the database at the time of insertion.

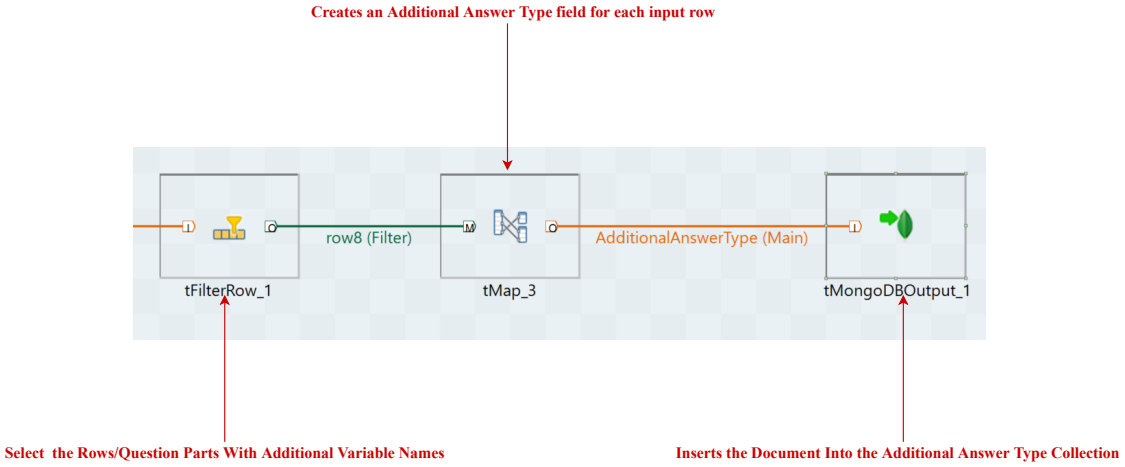


Figure 7.21: Talend Components for Saving a Document into the Additional Answer Part Collection

3.4.5 **Get additional answer part ids:** The input data from step 3.4.1 contains all the details for creating the 'answer part collection' document, except the 'additional answer part ids' of the 'additional variable names'. The 'additional answer part' corresponding to the 'additional variable names' is of type free text, and the corresponding document should be read from the 'additional answer part collection' to get its id. The ETL job flow for reading the document from the database is shown in Figure 7.22. The *tMongoDDInput* component is used to read the document from the database. The selection query and the collection from which the data is to be selected should be defined in the *tMongoDDInput* component. This is shown in Figure 7.23. The document is then saved in the buffer using a *tHashOutput* (renamed ad 'AdditionalAnswerPart') component.

3.4.6 **Create and save answer part collection document from the answer part details:** The ETL job flow for creating the document for the 'answer part collection' and inserting it into the database is shown in Figure 7.24. Initially, the 'additional answer part' ids of the 'additional variable names' from step 3.4.5 are added to the answer part details. This is done by using a *tMap* component. The input answer part details and the answer part details with the additional answer part ids are shown in Figure 7.24 (Table 1 & Table 2). The ETL job now has all the details for creating the 'answer part collection' document. These fields are then processed by a *tWriteJSONField* component, which will create the JSON fields for the answer part document. Each answer part document has two fields apart from its id: answer type and answer part

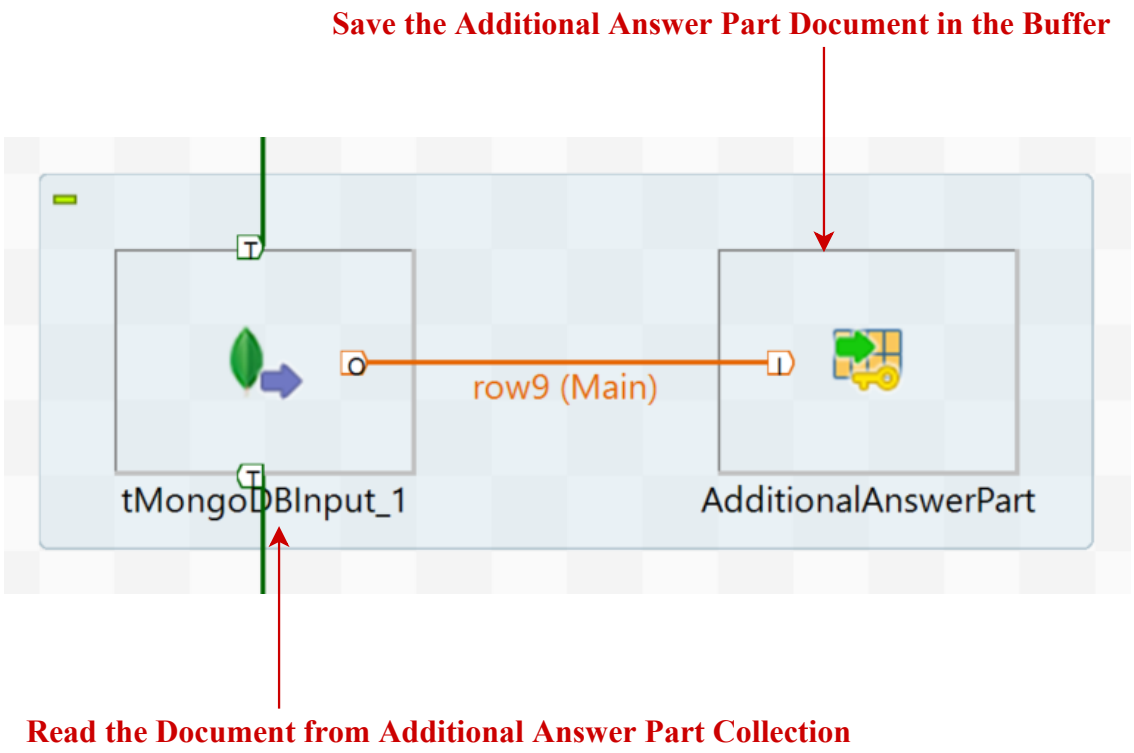


Figure 7.22: Talend Components for Reading the Additional Answer Part Collection Document

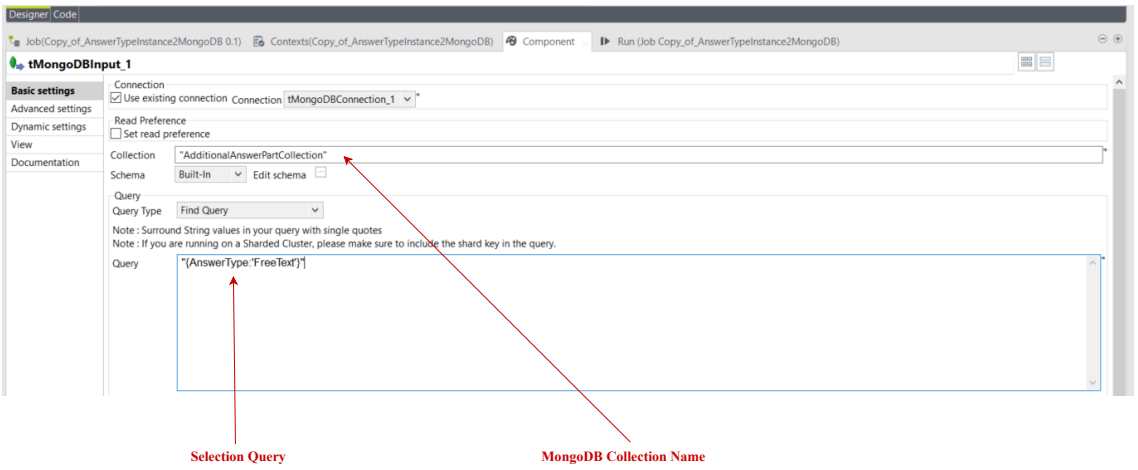
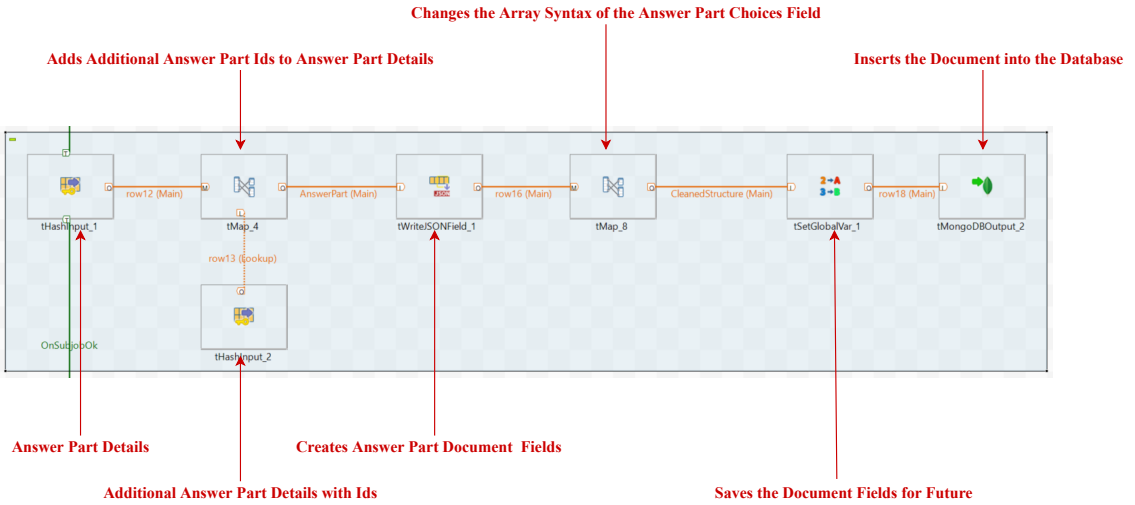


Figure 7.23: Structure of the *tMongoDDInput* Component for Reading the Documents from Additional Answer Part Collection

choices. The answer type is a text field, and the answer part choices are represented as an array of nested documents where each document contains the details of a choice. The output of the *tWriteJSONField* component is shown in Figure 7.24 (Table 3). However, the syntax of the array element in Figure 7.24 (Table 3) is not correct according to MongoDB. In MongoDB, each array should be represented by using square brackets ([ ]). Hence, the syntax of the array should be changed. This is done by another *tMap* component and the output of the *tMap* with correct array syntax is also shown in Figure 7.24

(Table 4). The ETL job now has the complete and correct JSON fields for creating the answer part document. Before inserting the document into the database, the fields are saved as global variables for future use by using a *tSetGlobalVar* component. Finally, the document is inserted into the answer part collection using *tMongoDBOutput* component. The id of the document is automatically generated by the database at the time of insertion.



(Table 1) Input Answer Part Details

Answer Type	Choice Sequence	Choice Value	Choice Text	Additional Variable Name	Choice Group
Range	1	1	nie		
Range	2	2	sehr selten		
Range	3	3			
Range	4	4			
Range	5	5			
Range	6	6	sehr häufig		

(Table 2) Answer Part Details with Additional Answer part Ids

Answer Type	Choice Sequence	Choice Value	Choice Text	Additional Variable Name	Additional Answer PartId	Choice Group
Range	1	1	nie			
Range	2	2	sehr selten			
Range	3	3				
Range	4	4				
Range	5	5				
Range	6	6	sehr häufig			

(Table 3) JSON Fields

AnswerType	AnswerPartChoices

Range	loop: {
	{
	ChoiceSequenceNumber: 1,
	ChoiceValue: 1,
	ChoiceText: "nie",
	AdditionalVariableName: "",
	AdditionalAnswerPartId: "",
	ChoiceGroup: ""
	},
	{
	ChoiceSequenceNumber: 2,
	ChoiceValue: 2,
	ChoiceText: "sehr selten",
	AdditionalVariableName: "",
	AdditionalAnswerPartId: "",
	ChoiceGroup: ""
	},
	.
	.
	.
	}

(Table 4) JSON Fields with new Syntax for Array

AnswerType	AnswerPartChoices
Range	[{ ChoiceSequenceNumber: 1, ChoiceValue: 1, ChoiceText: "nie", AdditionalVariableName: "", AdditionalAnswerPartId: "", ChoiceGroup: "" }, { ChoiceSequenceNumber: 2, ChoiceValue: 2, ChoiceText: "sehr selten", AdditionalVariableName: "", AdditionalAnswerPartId: "", ChoiceGroup: "" }, . . .]

Figure 7.24: Talend Components for Creating and Inserting Answer Part Collection Document

3.4.7 **Get the answer part id:** After saving an answer part into the database, the id of the answer part document should be retrieved and saved along with the corresponding variable name from step 3.4 so that they could be used in the

future while creating the question documents. For this, the id of the answer part document, saved in step 3.4.6, should be retrieved. The ETL job flow for this process is shown in Figure 7.25. Initially, a *tMongoDBInput* component reads the answer part document, which was inserted in the previous step, from the database. The document fields, saved as global variables in the previous step, are used for the selection of the document. The selection query is shown in Figure 7.26. The selected document has the id, and it is then added to the variable name from step 3.4 using a *tMap* component. Finally, the variable name and the answer part id is saved in the buffer using a *tBufferOutput* component.

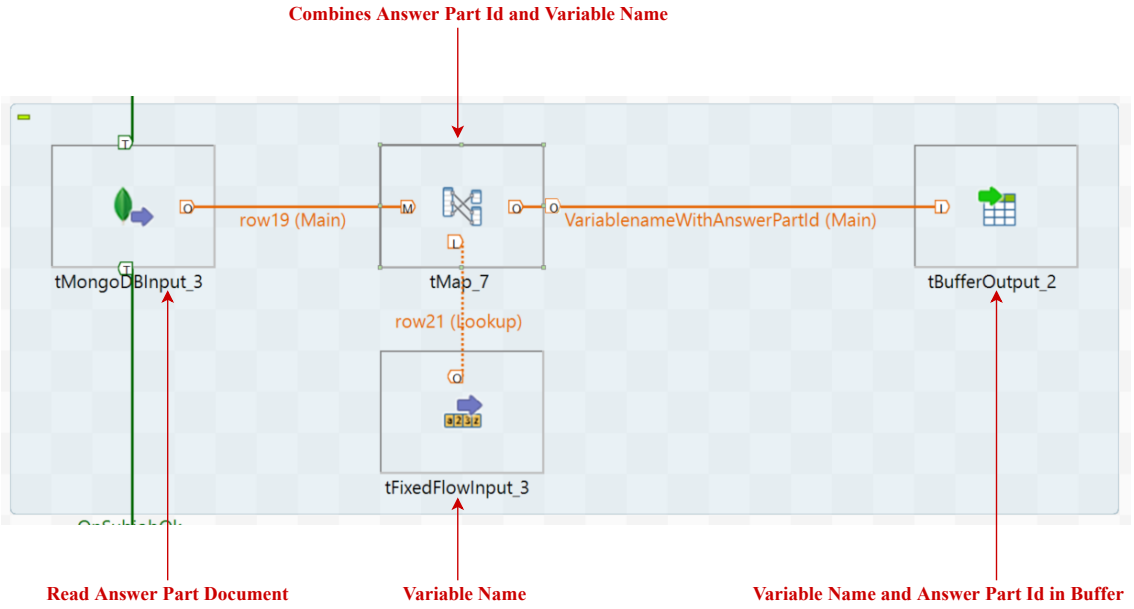


Figure 7.25: Talend Components for Reading Answer Part Id

3.6 **Create and save question collection document from the question details:** The process involved in this step is similar to that of step 3.4.6. Initially, the additional answer part ids from step 3.5 are added to the question details from step 3.1. Then, the answer part ids from step 3.4.7 are also added to the question details. Now, the ETL job has all the details for creating the 'question collection' document. These details are converted into JSON fields and then inserted into the question collection in the database.

3.7 **Get the Question Id:** After inserting a question document into the question collection in the database, the corresponding auto-generated id of the question should be retrieved for future use. This is done similar to the way it is done in step 3.4.7. The question document inserted in the previous step is retrieved using *tMongoDBInput* component, and the id is retrieved from it. The retrieved id is then stored in the buffer along with the corresponding question number from step 3.

4 **If a survey document is not present with the input survey details in database, then create a document in the survey collection:** In this step, the ETL job

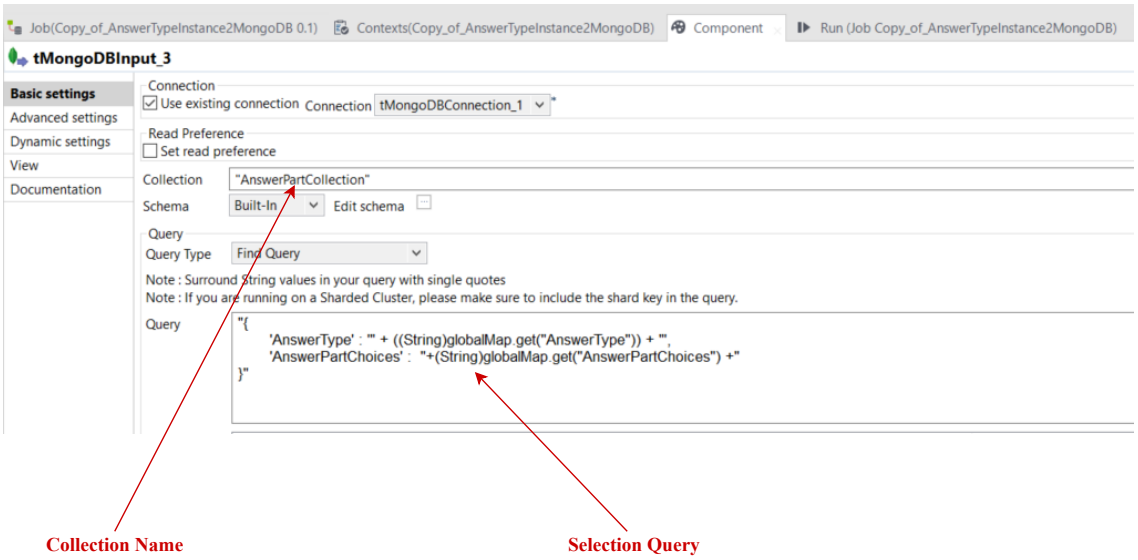


Figure 7.26: Structure of the *tMongoDDInput* Component for Reading the Document from Answer Part Collection

checks whether the survey collection already has a document with the survey details extracted in step 2. The flow of the ETL job for doing this is shown in Figure 7.27. A *tMongoDBInput* component tries to retrieve a document with the input survey details. If a corresponding document is not found, then a new document is created with the input survey details. The conditional flow of the ETL job is achieved by an 'if trigger connection' in Talend. The insertion of the new document is done by a *tMongoDBRow* component. The *tMongoDBRow* component executes the MongoDB command for creating the new survey document. The structure of the *tMongoDBRow* with the insert command is shown in Figure 7.28. On the other hand, if the survey document is already found, then the ETL job moves to the next step.

5 **If the survey document contains a survey module with the input survey module details, then insert the survey module details to the survey document:** In this step, the input survey module details from step 2 are added to the corresponding survey document. The previous step ensures that there is a document in the survey collection for the input survey details. This step adds the input survey module details into this document if it is not already present. The process involved in this step is similar to that of the previous step. The ETL job flow is shown in Figure 7.29. Initially, the ETL job retrieves the document from the survey collection with the given survey and survey module details using a *tMongoDBInput* component. If no document is found, then it confirms that the survey module details are not present in the survey document. In this case, the ETL job creates a survey module document from the input survey module details and adds it into the 'SurveyModules' array of the survey document. Each survey module document has a survey module id and survey module name. The ETL job has the survey module name from the input survey module details and creates a new id for the survey module by using a *tJava* component. The Java code executed by the *tJava* component,

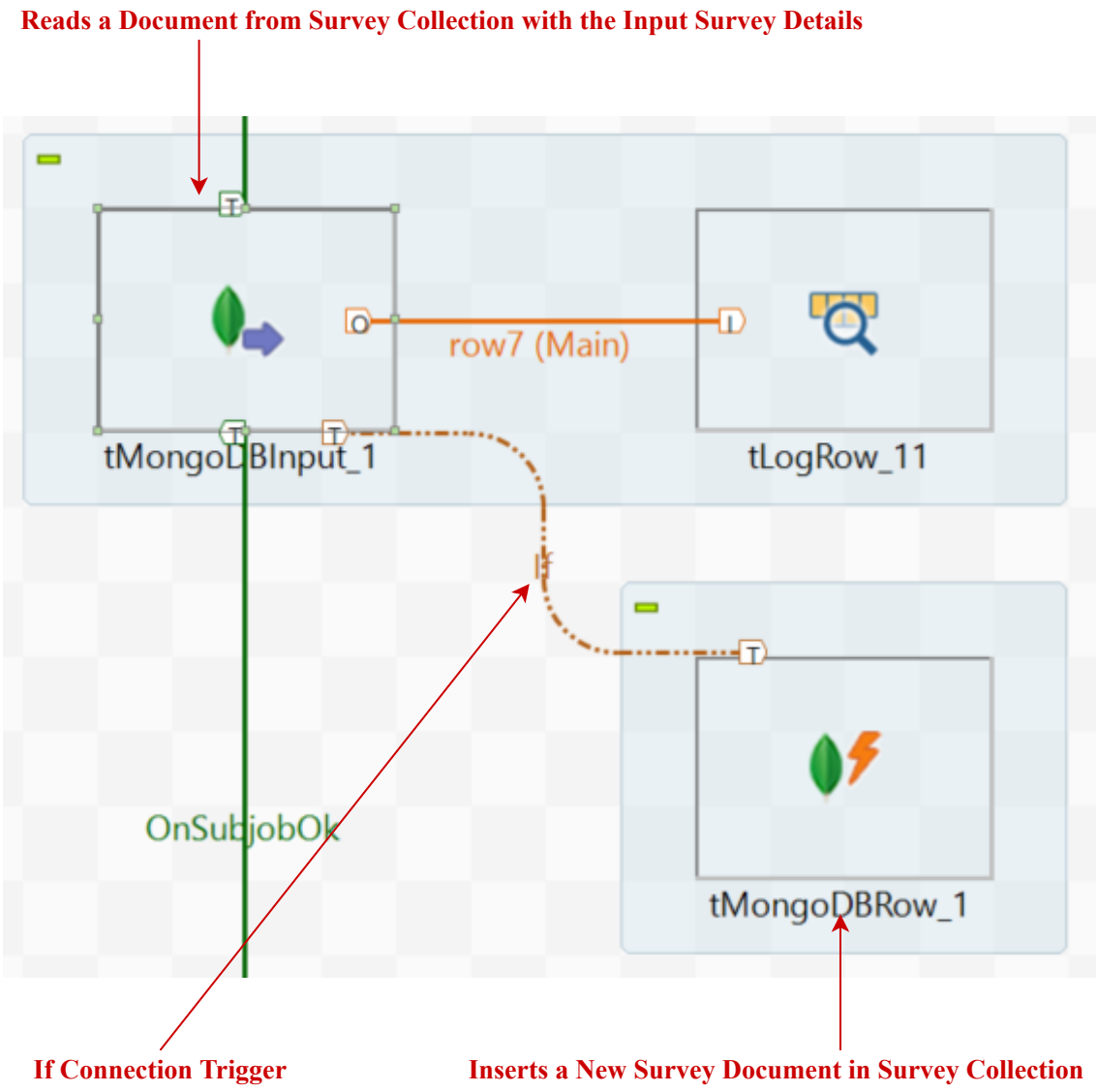


Figure 7.27: Talend Components for Creating a Survey Document with Input Survey Details, if it is not Already Present

which creates a new id for the survey module and saves it as a global variable, is shown in Figure 7.30. The *tLibraryLoad* component shown in Figure 7.29 is for loading the necessary Java libraries for executing the Java code in *tJava* component. Finally, the ETL job updates the survey document and adds the survey module details to it using a *tMongoDBRow*. The structure of the *tMongoDBRow* component with the update command is shown in Figure 7.31.

6 **Insert the question details into the survey document:** In this step, the question details are inserted into the survey document. The question numbers of all the questions in the input data along with their corresponding question ids are inserted into the 'SurveyModuleQuestions' array in the survey document. Each pair of question numbers and question ids is stored as a document in the array. The question numbers and question ids from step 3.7 could be directly added to the survey document using a *tMongoDBRow* component. The up-

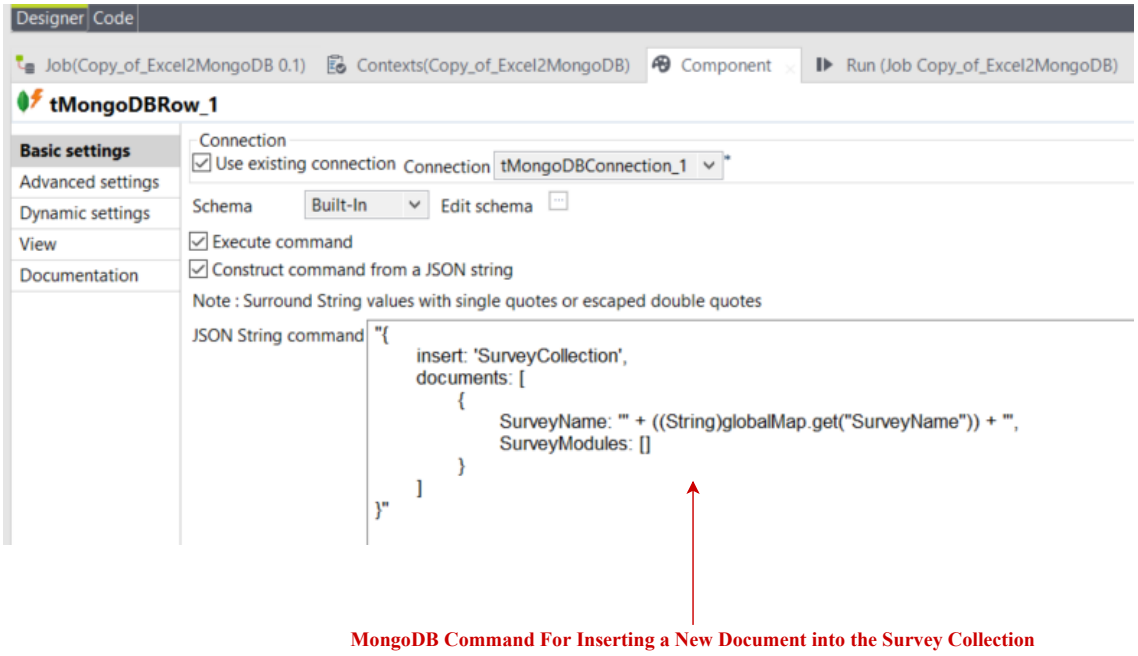


Figure 7.28: Structure of the *tMongoDBRow* Component for Creating the Survey Document

date query to add the question details to the survey document is similar to the one shown in Figure 7.31.

7 **Create and save footnote collection documents from the footnote details:** In the final step, the ETL job creates the footnote documents from the footnote details extracted in step 2 and saves these documents in the footnote collection of the database. The footnote details extracted in step 2 contains the question number, the part sequence number of the question part to which the footnote is associated with and the footnote text. However, a footnote collection document contains the survey module id, question id, part sequence number and footnote text as its attributes. Therefore, the ETL job converts the input footnote details into the required format before inserting them into the MongoDB collection. The flow of the ETL job for this process is shown in Figure 7.32. Initially, the question numbers in the input footnote details are replaced by corresponding question ids. This is done by using a *tMap* component. The *tMap* component takes the question numbers and their corresponding ids from step 3.7 and replaces each question number in the extracted footnote details with the corresponding question id. Finally, the ETL job inserts the footnote documents into the database using a *tMongoDBRow* component. The survey module id attribute for the footnote documents is already saved as a global variable in the previous step.

7.3 Summary

In this chapter, we have discussed the process of loading the survey questions from portable document format files to the designed MySQL and MongoDB databases.

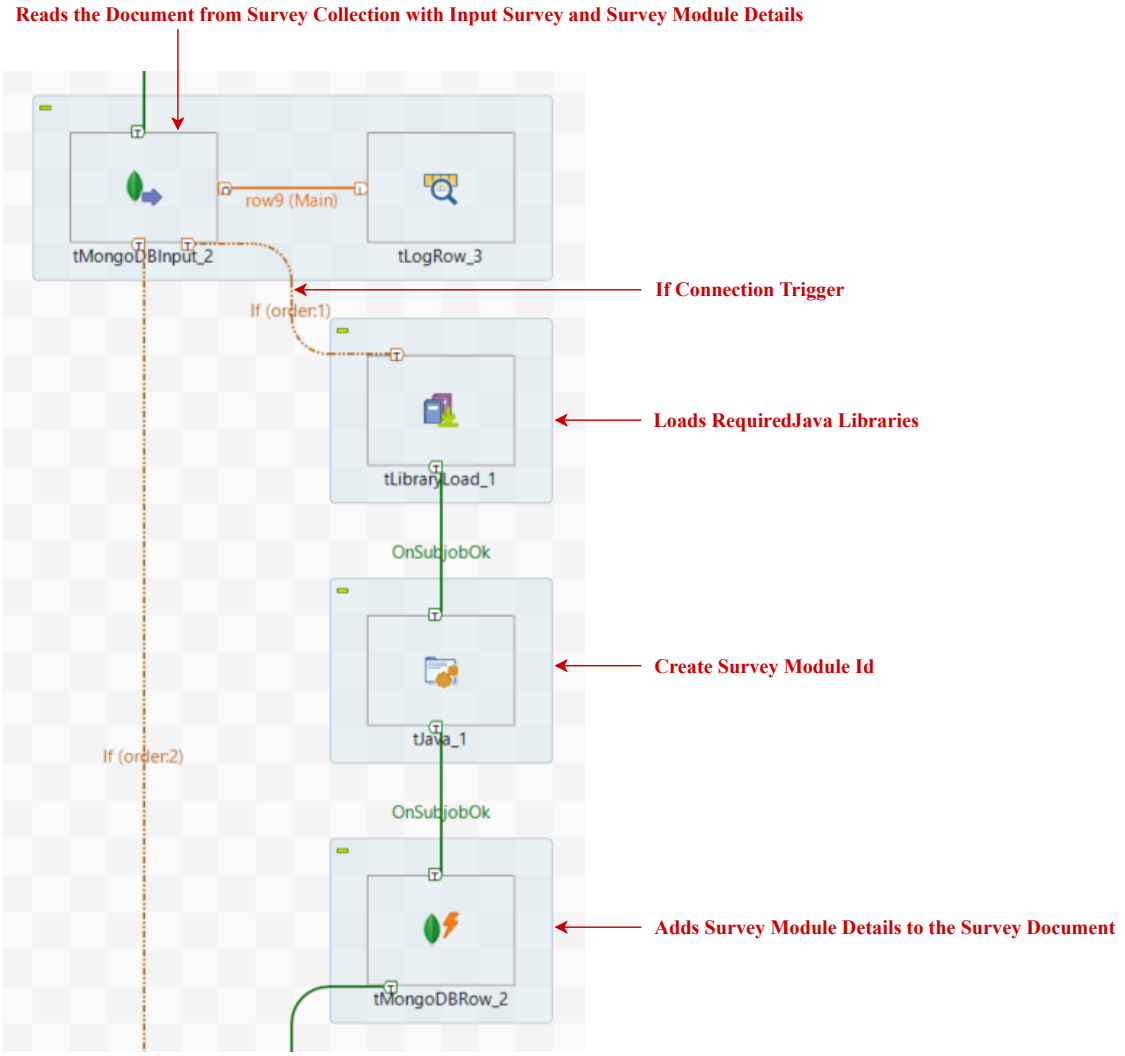


Figure 7.29: Talend Components for Adding the Survey Module Details to the Survey Document, if it is not Already Present

Initially, the different defined entities of the survey questions present in the PDF documents are extracted. The different entities extracted for each question are the survey details of the question, survey module details of the question, question parts in the question, answer parts associated with the question parts, choices of answer parts, additional answer parts, choices of additional answer parts and footnotes present in the question. These extracted entities are then saved in excel files.

The extracted entities of the survey questions in the Excel files are then processed and loaded into MySQL and MongoDB databases using ETL jobs. The extent to which the data is normalized is different in both databases. The data is less normalized in the MongoDB database as compared to the SQL database. Since the data processing is different for both databases, separate ETL jobs were designed to load the data into the MySQL and MongoDB databases. Since the number of data processing is less in the MongoDB database, the ETL job for loading the data into the MongoDB database is expected to perform better than the one for the MySQL

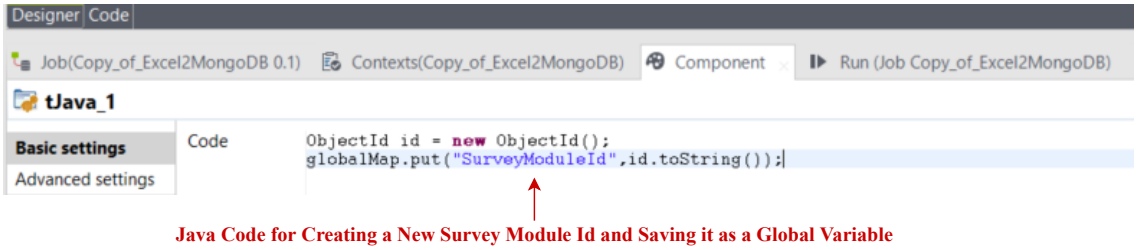


Figure 7.30: Structure of the *tJava* Component for Creating a new Survey Module Id

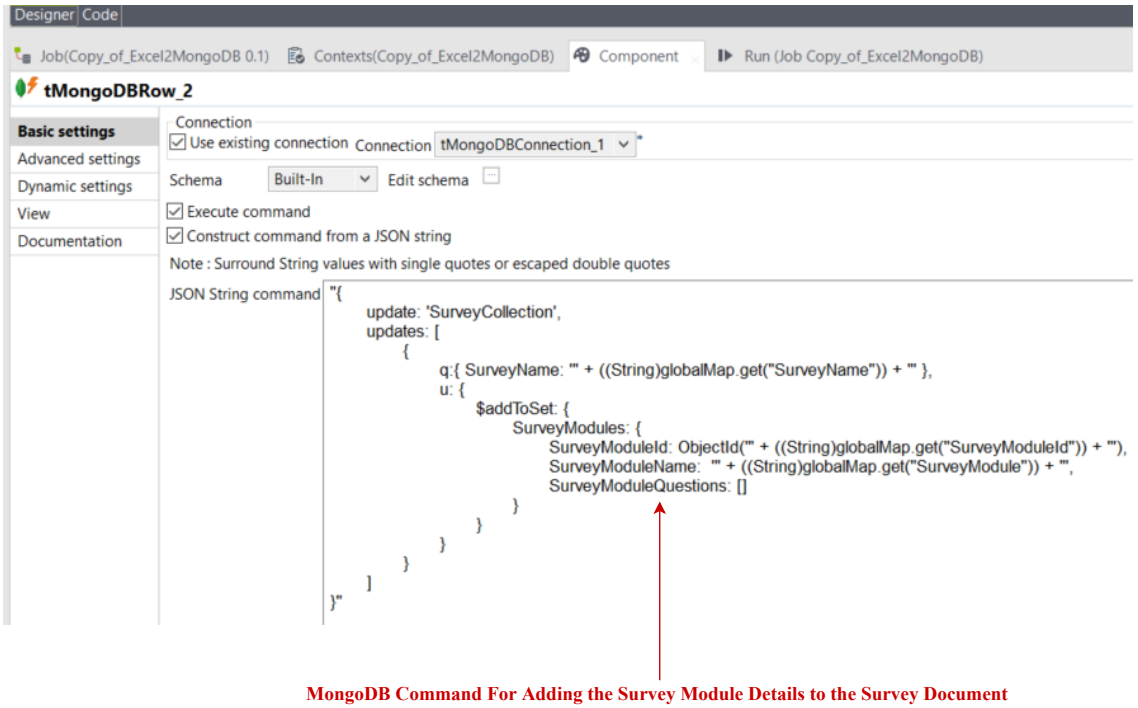


Figure 7.31: Structure of the *tMongoDBRow* Component for Adding the Survey Module Details to the Survey Document

database in terms of job execution times. However, it is only a hypothesis and should be verified with proper evaluation of the ETL jobs.

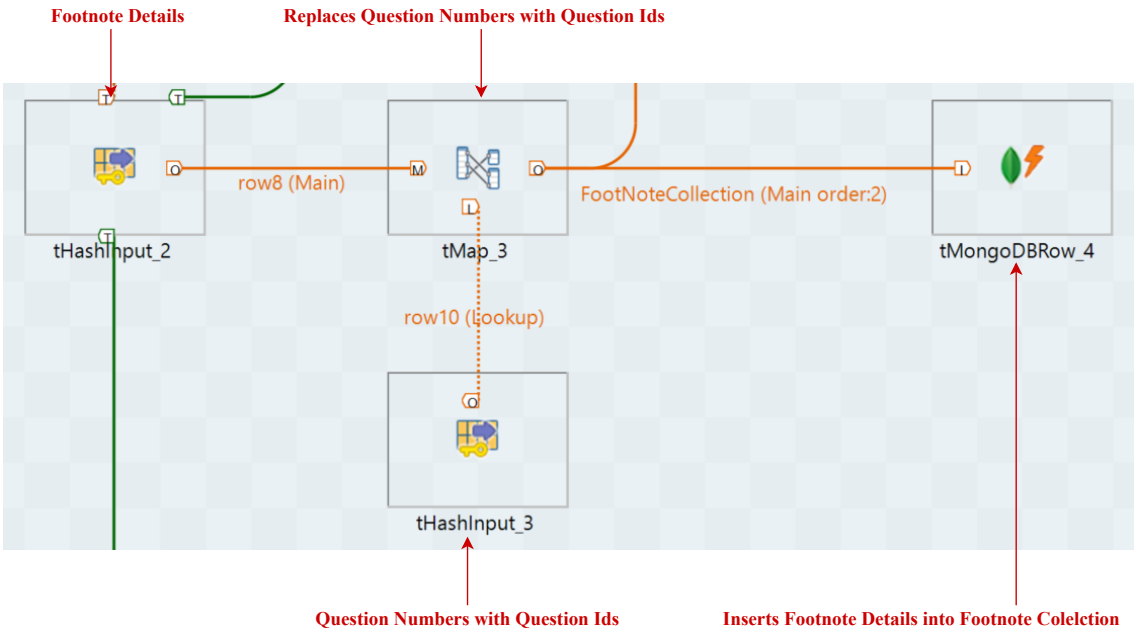


Figure 7.32: Talend Components for Creating and Saving the Footnote Documents into the Footnote Collection

8. Evaluation

In this thesis work, SQL and NoSQL databases were designed and implemented for managing the survey questions at DZHW. The relational database is implemented in MySQL, and the document-oriented database is implemented in MongoDB. Furthermore, ETL based workflows in Talend Open Studio were also designed and implemented for loading the survey questions present into the designed databases.

The goal of this thesis is to design an SQL database and a NoSQL database for the management of survey questions at DZHW. Furthermore, the best-suited database should be identified by evaluating and comparing both databases. Therefore, in this chapter, we compare and evaluate the designed SQL and NoSQL databases based on different parameters. Firstly, the database systems are compared based on the performance of their corresponding ETL workflows that load the survey data into them. Secondly, the storage space requirements of the designed databases are evaluated. Furthermore, the data retrieval performance of the two databases is analyzed by comparing the performance of different selection queries in both databases. Additionally, the performance of both databases in terms of their inbuilt text search capabilities is also evaluated. Finally, a selectivity analysis of the databases is also done to compare them in terms of the time taken to load a different number of questions. Before discussing the evaluation steps in detail, the setup or framework used for the evaluation is described.

8.1 Evaluation Setup

The framework used for conducting the various experiments for the evaluation of the designed databases is explained in detail in this section. Initially, the hardware specifications used for the experiments are discussed, followed by the explanation of different software specifications. Furthermore, a description of the data set used for the evaluation experiments is also given.

8.1.1 Hardware Specifications

The experiments were done in a machine with Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz. The read/write memory of the system was 32GB.

8.1.2 Software Specifications

The operating system used for running the experiments was Windows 10. The edition of the operating system was Enterprise Edition, and the version was 21H1. 'MySQL 8.0.25(MySQL Community Server - GPL)' is used for implementing the relational database schema, and 'MongoDB 5.0.1 Community Edition' was used to implement the NoSQL database schema. The ETL jobs for loading the data into the databases were designed using the 'Talend Open Studio for Big Data version 7.3.1'.

8.1.3 Data Set

In order to perform different experiments for the evaluation of the databases, four different survey sets were prepared: survey set A, survey set B, survey set C and survey set D. The total number of surveys, survey modules and questions in each of the survey sets are given in Table 8.1. Survey set A contains one survey with three survey modules and 80 questions. Survey set B contains one survey, six survey modules and 160 questions. Survey set C has two surveys, twelve survey modules and 320 questions. Survey set D contains four surveys, twenty-four survey modules and 640 questions. Finally, the survey set E has eight surveys, forty-eight survey modules and 1280 questions. Each survey set might contain repeated questions, but these repeated questions are present in different survey modules.

The number of surveys, survey modules, and survey questions increases from survey set A to survey set E. This is done to learn the changes in the performance of the databases as the number of survey elements increases. Furthermore, the size of the data also increases as we move from survey set A to survey set E. It allows us to identify the changes in the databases' performance with an increasing data load as well. The survey sets were created by using the survey questions from the "Die Studierendenbefragung in Deutschland" survey discussed in Chapter 4.

	Number of Surveys	Number of Survey Modules	Number of Questions	Size
Survey Set A	1	3	80	0.0336 MB
Survey Set B	1	6	160	0.0674 MB
Survey Set C	2	12	320	0.1349 MB
Survey Set D	4	24	640	0.2698 MB
Survey Set E	8	48	1280	0.5398 MB

Table 8.1: Total Number of Surveys, Survey Modules and Questions in the Survey Sets

8.2 Data Loading Analysis

In this section, we compare the performance in terms of the execution times of the ETL workflows used for loading the data into the databases. It is done to find the time required to load the data into the two databases. Even though the databases store the same data, their structure of the data is different. In the MySQL database, the data is stored in the form of tables, whereas in the MongoDB database, it is stored as BSON documents. Moreover, the extent to which the data is normalized in the MongoDB database is also less as compared to that in the SQL database.

Therefore, how the ETL jobs process and save the data is different in the case of both databases. The data insertion time of MySQL and MongoDB databases also plays a crucial role.

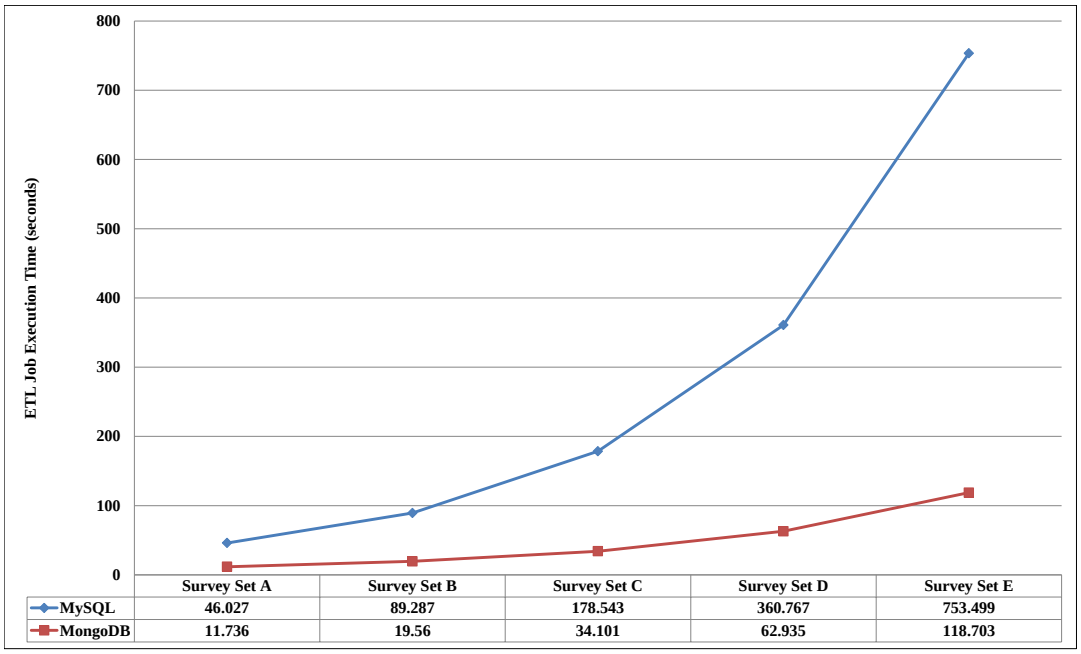
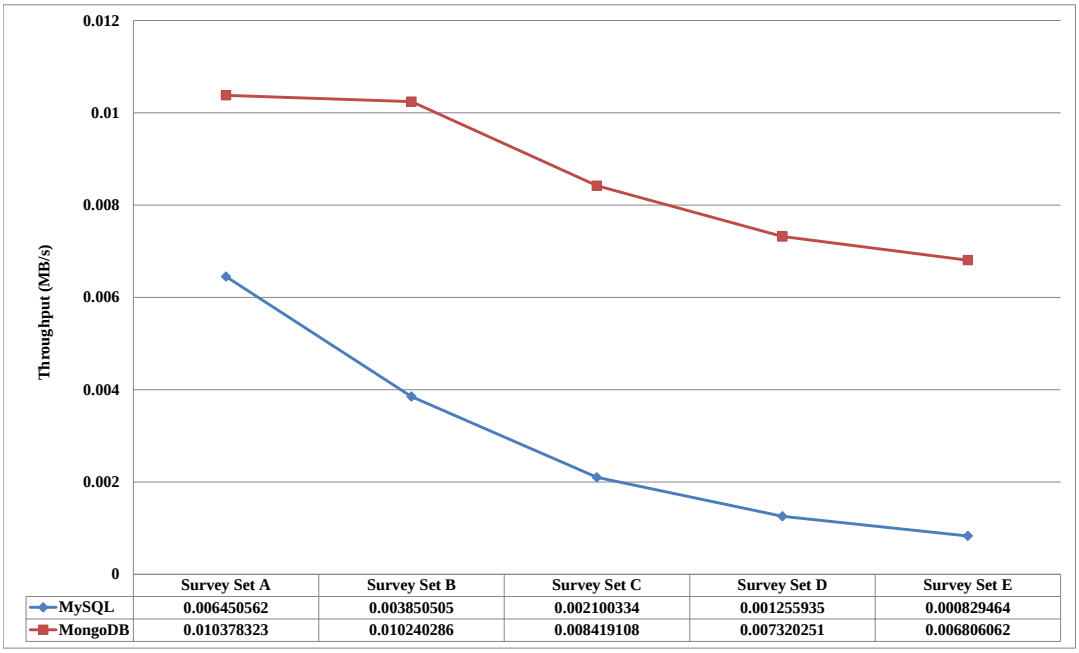


Figure 8.1: ETL Job Execution Times for MySQL and MongoDB

Each survey set, discussed in Section 8.1.3, was loaded into the MySQL and MongoDB databases, and the execution times of both the ETL jobs were measured. The execution times of the ETL jobs for the different survey sets are shown in Figure 8.1. In the graph, we can see that the execution times of both the ETL jobs increases with the size of the survey sets. However, the ETL job for loading the data into the MongoDB database performs better than the ETL job for the MySQL database for all the survey sets. This can be explained by the fact that the MongoDB database is faster than MySQL in terms of insert operations [GGPO15]. Furthermore, the data is less normalized in the MongoDB database and therefore, only less number of data processing operations are present in the MongoDB ETL job. This could have also played a major factor in its faster execution time.

Since the MySQL and MongoDB schemas are different, the amount of the data written into both the databases are different even though the input data is the same. Therefore, we also compare the throughput of both ETL jobs. Figure 8.2 shows the throughput of the ETL jobs for different survey sets. The throughput values are calculated using the ETL job execution times in Figure 8.1 and the data size values from Table 8.2. In Figure 8.2, we can see that the MongoDB database has higher throughput values for all the survey sets. Even though the data sizes in the MySQL database are higher for survey sets A, B and C, the corresponding ETL job execution times are much higher as compared to that in the MongoDB database,

which resulted in low throughput. Therefore, we can conclude that the MongoDB database performs better than the MySQL database in the data loading analysis



	Survey Set A	Survey Set B	Survey Set C	Survey Set D	Survey Set E
MySQL	0.006450562	0.003850505	0.002100334	0.001255935	0.000829464
MongoDB	0.010378323	0.010240286	0.008419108	0.007320251	0.006806062

Figure 8.2: Throughput of the ETL Jobs

8.3 Storage Space Analysis

In this section, we compare both the databases in terms of the storage space taken for storing the survey questions. Since the extent to which the same data is normalized in both databases are different, it would be useful to calculate the size of the same data and indexes in both databases.

Initially, we consider the data size of both databases. The data size may be defined as the size taken by the databases to represent the data. The data size of the MySQL database can be calculated by using the SQL command in Listing 8.2. The data size of the MongoDB database can be calculated by using the function shown in Listing 8.1. This function returns the details of the current database, which includes the data size as well.

```
db.stats();
```

Listing 8.1: MongoDB Function for Calculating the Data Size

```
SELECT table_schema "DB Name", ROUND(SUM(data_length)/1024/1024, 4)
"Data Size in MB" FROM information_schema.tables WHERE
table_schema = [DB Name];
```

Listing 8.2: SQL Command for Calculating the Data Size

	MySQL	MongoDB
Survey Set A	0.2969 MB	0.1218 MB
Survey Set B	0.3438 MB	0.2003 MB
Survey Set C	0.3750 MB	0.2871 MB
Survey Set D	0.4531 MB	0.4607 MB
Survey Set E	0.6250 MB	0.8079 MB

Table 8.2: Data Size of MySQL and MongoDB Databases

The data size of the MySQL and MongoDB databases for each survey set is given in Table 8.2. While analyzing the values in Table 8.2, we can see that the MySQL database has taken more space for storing the survey questions as compared to the MongoDB database for survey sets A, B and C. But, MySQL performs better in the case of survey sets D and E. This can be explained by the extent to which the data is normalized in both databases. As the data size increases, the number of redundant survey questions, question parts and answer parts increases. In such cases, the MySQL database, in which data is normalized more, performs better than the MongoDB database, as the data size is reduced by non-redundant storage of the repeated elements of the questions. Therefore, the MySQL database could be considered as a better option for managing a large amount of survey data in terms of the data size.

Secondly, we compare the index sizes of both databases for each survey set. The size of the indexes in the MySQL database could be calculated by using the SQL command in Listing 8.3. The MongoDB index size could be calculated by using the function shown in Listing 8.1, which returns the total index size of the database as well.

```
SELECT table_schema "DB Name", ROUND(SUM(index_length)/1024/1024, 4)
"Index Size in MB" FROM information_schema.tables WHERE
table_schema = [DB Name];
```

Listing 8.3: SQL Command for Calculating the Data Size

	MySQL	MongoDB
Survey Set A	0.2813 MB	0.1563 MB
Survey Set B	0.3125 MB	0.1563 MB
Survey Set C	0.3281 MB	0.2188 MB
Survey Set D	0.4688 MB	0.2305 MB
Survey Set E	0.7188 MB	0.2461 MB

Table 8.3: Total Index Size of MySQL and MongoDB Databases

The total index size of the databases for different survey sets are shown in Table 8.3. From the analysis of these index sizes, we can see that the MySQL database needs

more space to store the indexes as compared to the MongoDB database for all the survey sets. This is because the MySQL database has fourteen tables, which would in turn result in a large number of indexes. However, the MongoDB database has only five collections and therefore would only be having fewer indexes as compared to the MySQL database. Therefore, we can conclude that the MongoDB database performs better in terms of the total index size.

8.4 Selection Query Tests

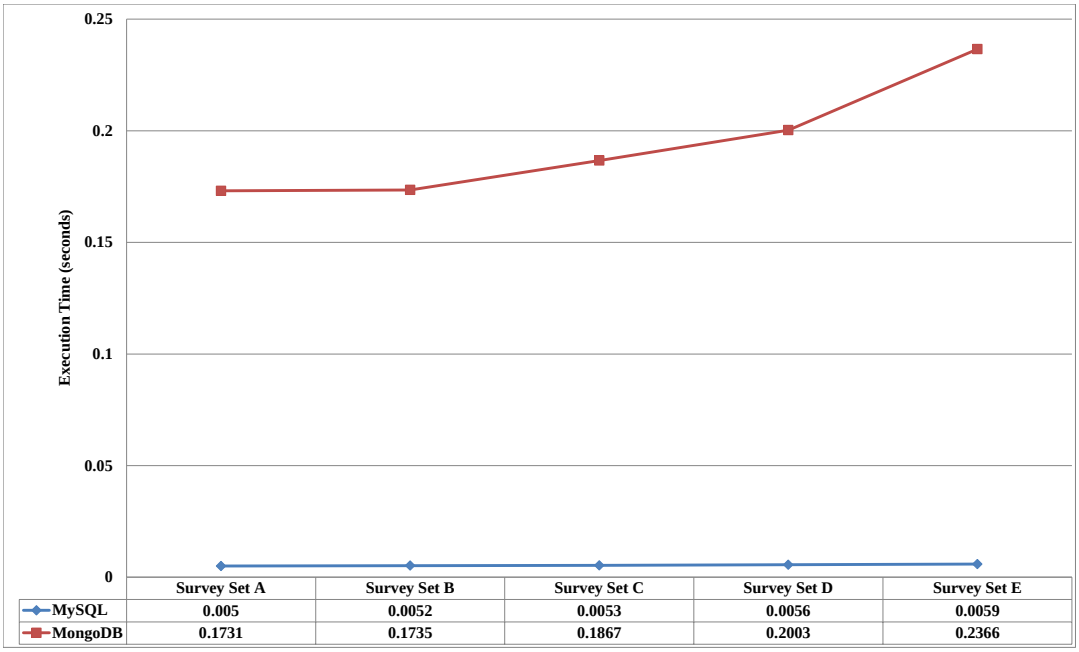
In this section, we evaluate the performance of the MySQL and MongoDB databases for different data retrieval queries. Since the data is shared among different tables in the MySQL database, join operations between these tables should be done in most of the selection queries to retrieve the required data. Similarly, in the MongoDB database also, the data is shared among different collections. Therefore, data in these multiple collections should be joined as well in most of the cases to get the required data. Lookup operation in MongoDB is used for this purpose which works similar to the left outer join operation in SQL. Thus, the main aim of these tests is to compare the performance of the selection queries with join operations in MySQL and lookup operations in MongoDB. For this, different use cases were designed that retrieve data shared among different tables in MySQL or different collections in MongoDB. Each of these use cases retrieves the same data from both databases and requires join or lookup operations depending on the database.

Three different use cases were designed in total. The first use case retrieves all the questions in a survey module. This use case is relevant when a person wants to analyze all the questions present in a survey module. An example scenario is when a person wants to create a survey module in a new survey, and a similar survey module is already present in an existing survey. In such situations, the person can analyze all the questions in the existing survey module to check if the same survey module can be reused in the new survey. The second use case is to select a question using its unique question id. This use case would be used for the fast retrieval of a survey question. For example, if a person wants to make changes in a particular survey question and the unique id of the question is known, then the person can easily retrieve the exact question using the id without any further searches. The third use case is to retrieve survey questions using question part texts. This would be a commonly used scenario to search the survey questions. An example scenario is when a person wants to find all the survey questions in the database that contains some keywords or text patterns. In such cases, the corresponding questions could be selected using this use case. The different uses cases are discussed in detail below.

8.4.1 Use Case 1: Select All Questions in a Survey Module

In this use case, all the questions in a survey module are selected from the database. The id of the survey module is given as the input parameter in the query, and the query returns all the questions in that particular survey module. The corresponding survey details are returned as well. Furthermore, the footnote details should be displayed along with the question details.

MySQL and MongoDB queries (see Chapter A) were designed for this use case, and each query was executed multiple times for each survey set. The id of the same survey module is given as the input parameter while executing the MySQL and MongoDB queries for each survey set so that the queries return the same result. Each query was executed twenty times, and the average value of the execution times was taken for the evaluation. The query execution times for each survey set is shown in Figure 8.3.



	Survey Set A	Survey Set B	Survey Set C	Survey Set D	Survey Set E
MySQL	0.005	0.0052	0.0053	0.0056	0.0059
MongoDB	0.1731	0.1735	0.1867	0.2003	0.2366

Figure 8.3: Query Execution Time - Select All Questions in a Survey Module

By analyzing the values in Figure 8.3, we can see that the MySQL database performs better than the MongoDB database for all the survey sets in this particular selection use case. The low performance of the MongoDB database could be explained by the expensive lookup operations done in the MongoDB selection query.

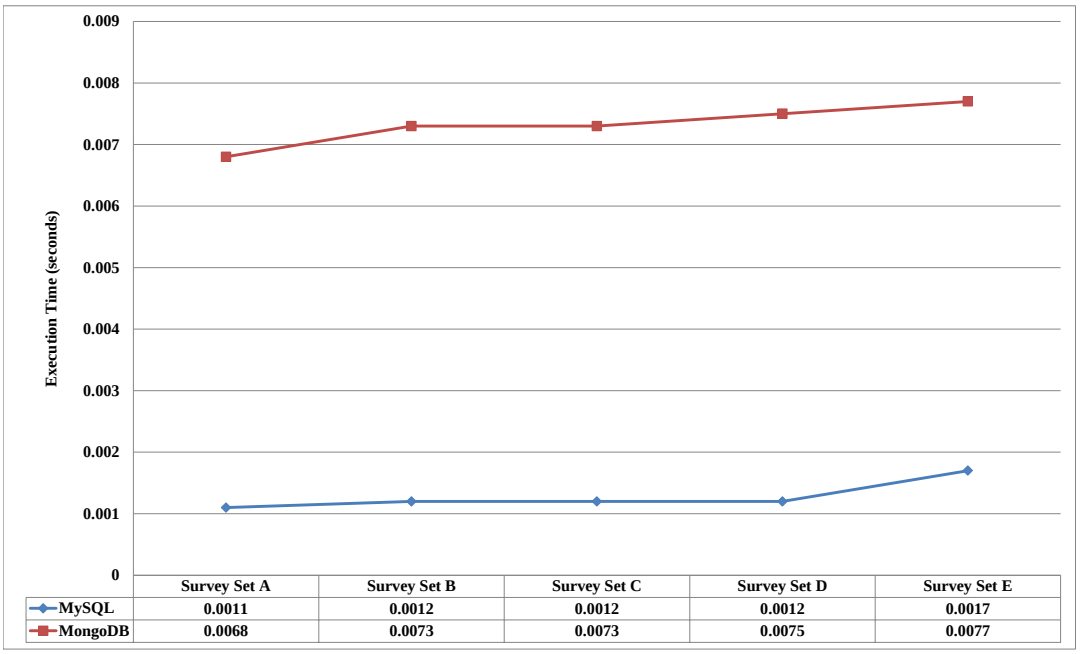
Next, we consider the performance of each database separately. We can see that as the size of the data set increases, the MySQL query performs better than the MongoDB query in terms of execution times. The increase in the execution time with the increase in the size of the data is less in MySQL as compared to that in MongoDB. In the case of the MySQL database, the increase in the query execution time from survey set A to survey set E is just 0.0009 seconds. Whereas, in the case of the MongoDB database, the increase in the execution time is 0.0635 seconds. With this result, we argue that MySQL performs better than MongoDB as the data size increases in this particular use case. However, we believe that this is not enough to make a conclusion as the increase in the execution times are very small in both the databases and these values are heavily dependent on the dynamic characteristics

of the system in which they are executed, such as system load and available main memory.

8.4.2 Use Case 2: Select Question by Id

This use case is designed to retrieve a question by giving its unique id. By giving the question id as the input parameter to the selection query, the corresponding question details, footnote details, survey and survey module details to which the question belongs should be returned as the result.

Like in the previous use case, MySQL and MongoDB queries were designed for the use case. The queries were executed twenty times each for each survey set, and the average value of the execution times was taken for the evaluation. While executing the queries for a survey set, the id of the same question is given as the input parameter to the MySQL and MongoDB queries to get the same results. The query execution times are shown in Figure 8.4.



	Survey Set A	Survey Set B	Survey Set C	Survey Set D	Survey Set E
MySQL	0.0011	0.0012	0.0012	0.0012	0.0017
MongoDB	0.0068	0.0073	0.0073	0.0075	0.0077

Figure 8.4: Query Execution Time - Select Question by Id

The MySQL database outperforms the MongoDB database in this use case as well. However, when we consider the performance of each query separately for different survey sets, we can see that the increase in the data size does not have much effect on the query performance of both databases. As the data size increases, the execution time of both the queries increases only by a small amount. The increase in the execution time from survey set A to survey set E in the case of the MySQL database is 0.0006 seconds, and in the case of the MongoDB database it is 0.0009 seconds.

8.4.3 Use Case 3: Select Questions by Question Part Text

In this use case, the survey questions are selected from the database by using the question part texts. The query takes a string as the input parameter, and all the questions that contain the string as a part of their question part texts are returned as the result. Along with the question details, the corresponding footnote and survey details of the questions should be also retrieved as well.

MySQL and MongoDB queries were designed for this use case. The queries do not perform the full-text search using the input string. Instead, this use case focuses on the performance of the pattern matching functions in the databases. In MySQL, this is done using the 'LIKE' operator, and in MongoDB, this is done using a simple '$match' operator. The performance of the full-text search feature of the databases is evaluated in detail in Section 8.5. The designed queries were executed twenty times each for each survey set, and the average execution time is taken for the evaluation. Figure 8.5 shows the query execution times for the MySQL and MongoDB databases. The same string is given as the input parameter of both the queries for each survey set to get the same results.

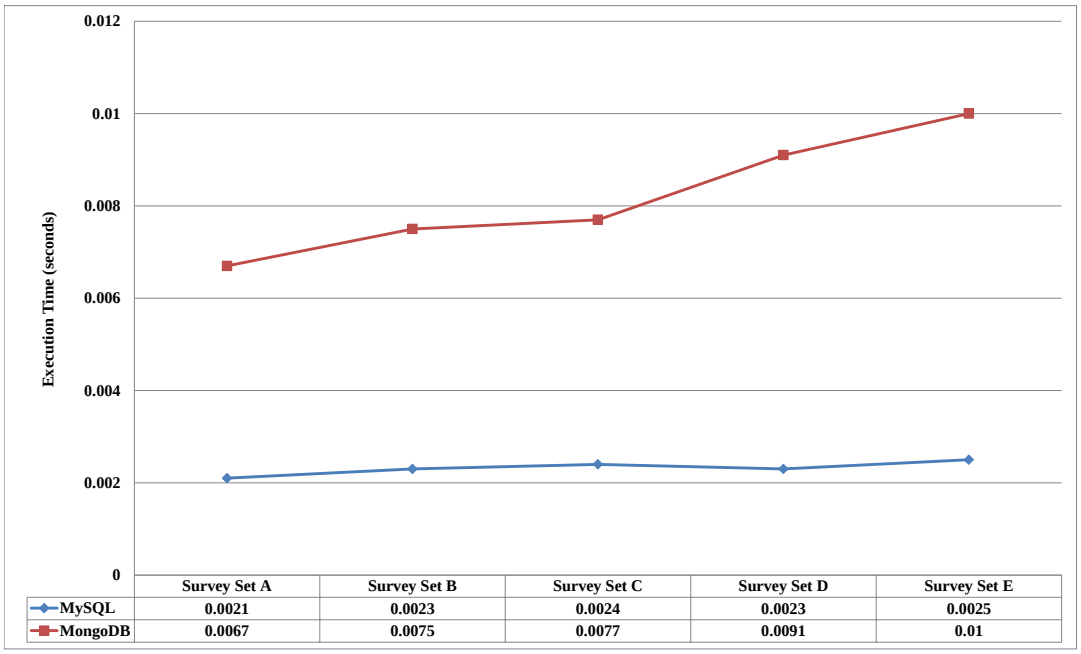


Figure 8.5: Query Execution Time - Select Questions by Question Part Text

By analyzing the results in Figure 8.5, we can see that the MySQL database performs better than the MongoDB database in this use case also. On the other hand, while analysing the performance of each database separately for different survey sets, we can see that the MySQL query performance is not much affected by the increase in the data size. The difference between the query execution times of survey set E and survey set A is just 0.0004 seconds. However, the performance of the MongoDB

query degrades gradually as the data size increases (from 0.0067 seconds for survey set A to 0.01 seconds for survey set E). But, the increase in the execution times are small, and these values are heavily dependent on the dynamic characteristics of the system.

8.4.4 Selection Query Tests: Summary

We have tested the performance of MySQL and MongoDB databases for different selection use cases. The use cases were formulated, and queries were designed for each of these use cases in both databases. The performances of the databases are then compared based on the query execution times. The main outcomes of this evaluation are listed below:

- The MySQL database performs better than the MongoDB database in all the discussed use cases in terms of query execution times.
- While considering the individual performances of the databases separately in each use case for different survey sets, we can say the performance of MySQL and MongoDB are not much affected by the data size. However, the MySQL database has a slight advantage over MongoDB in use cases 1 and 3.

8.5 Full-Text Search Analysis

In this section, we evaluate the full-text search performances of the designed MySQL and MongoDB databases. This comparison is crucial as the survey questions could be considered as text data, and most of the data retrieval operations from these databases would be based on text search. For example, one of the most common data selection operations that would be performed is the retrieval of survey questions using some keywords. Full-text search is the best technique that could be used to facilitate such searches that involve searching text data.

Full-text search is an advanced way of searching text data in databases by using text indexes. To perform a full-text search in a database, text indexes should be created for each column or field which stores the text data which is to be searched. The full-text search operation finds all the instances of the input text keywords by performing searches on the text indexes. This searching technique is faster than the normal pattern matching functions like 'LIKE' and '$match' operations.

Both MySQL and MongoDB offer the capabilities to perform the full-text search. They are discussed below:

Full Text Search in MySQL

MySQL offers three types of full-text searches: Natural language full-text search, Boolean full-text search and query expansion search. Natural language full-text search considers the input text keywords as simple free texts. The Boolean full-text search allows Boolean operators in the input text keywords. These Boolean operators have special meanings, such as the '+' sign could be added to the word to

make that particular word compulsory in each returned row, and the '-' sign could be added to a word to make that word not present in any of the returned rows.

Query expansion search uses a different search strategy as compared to the other full-text search techniques. In query expansion search, the text indexes are searched at first using the input keywords to return the rows that contain the input keywords. Then, relevant words are extracted from this initial result, and a full-text search is again performed with these extracted words. The result from the second search is then added to the first result to get the final result of the search operation.

Full Text Search in MongoDB

MongoDB also supports full-text search using text indexes. Unlike MySQL, there is only one type of full-text search in MongoDB. The '$text' operator is used to perform the full-text search in the MongoDB database.

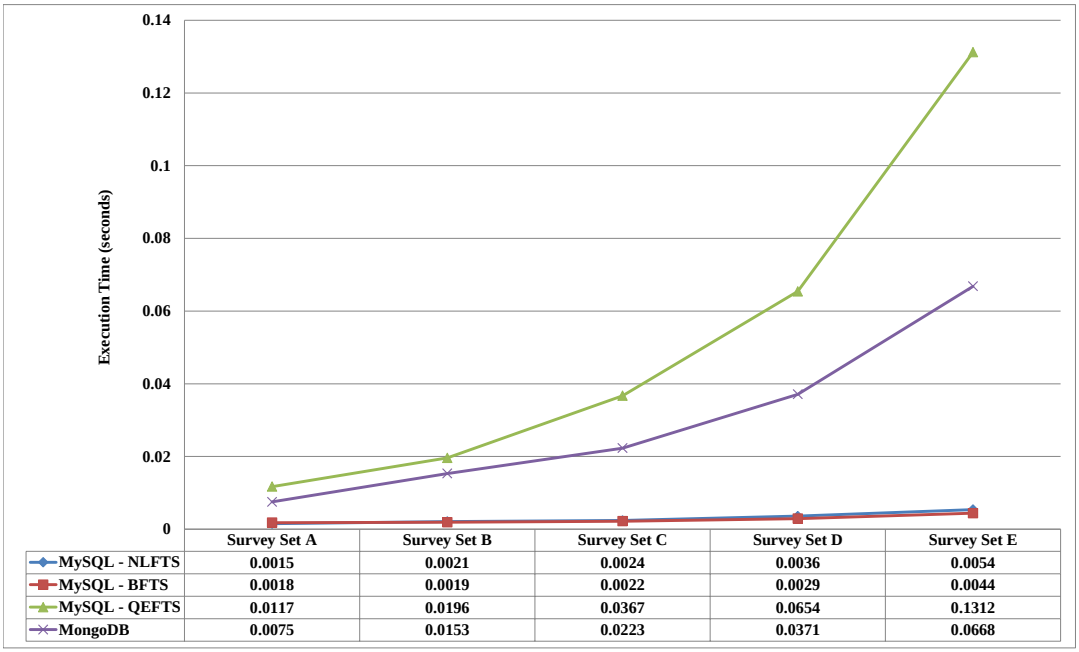
Full-Text Search: MySQL vs MongoDB

To compare the full-text search performances in our MySQL and MongoDB databases, we take the same use case discussed in Section 8.4.3. The survey questions should be retrieved from the database using their question part texts. There are four queries in total: three queries for the MySQL database as there are three types of full-text searches in MySQL and one query for the MongoDB database. Each query takes a keyword as input parameter and performs the full-text search. The result of each query contains the details of all the questions that are found using the corresponding full-text search along with their footnote, survey and survey module details.

In order to successfully execute these queries, appropriate text indexes should also be created. For this, a text index was created in the MySQL database on the 'question_part_text' column of the Question Part Text table. Similarly, a text index was created on the 'QuestionText' field in the Question collection of the MongoDB database as well.

The designed queries were executed twenty times each for each survey set, and the average execution time was taken for the evaluation. Figure 8.6 shows the query execution times of the MySQL and MongoDB full-text search queries. In Figure 8.6 NLFTS stands for natural language full-text search, BFTS stands for Boolean full-text search and QEFTS stands for query expansion full-text search.

The results in Figure 8.6 shows that the Boolean full-text search of the MySQL database is the best among the compared techniques. The natural language full-text search also performed well and comes in the second position with slightly higher execution times. For survey set A, the natural language full-text search performs better than the Boolean full-text search as well. The MongoDB full-text search and the query expansion full-text search techniques did not perform well as compared to the other two techniques and come in the third and fourth positions respectively. The high execution time values of the query expansion full-text search could be explained by the multiple steps involved in its search procedure. Since a second search is also done by extracting the important keywords from the results of the first search, the query execution is expected to take longer than the other techniques



	Survey Set A	Survey Set B	Survey Set C	Survey Set D	Survey Set E
MySQL - NLFTS	0.0015	0.0021	0.0024	0.0036	0.0054
MySQL - BFTS	0.0018	0.0019	0.0022	0.0029	0.0044
MySQL - QEFTS	0.0117	0.0196	0.0367	0.0654	0.1312
MongoDB	0.0075	0.0153	0.0223	0.0371	0.0668

Figure 8.6: Query Execution Time - Full Text Search

that only has one level of search. However, in general, we can say that the MySQL database performs better than the MongoDB database in terms of full-text search capabilities.

Since text indexes should be created for conducting full-text searches in both databases, it would be noteworthy to compare the sizes of these text indexes as well. For this, the size of the text indexes in MySQL and MongoDB databases are calculated and tabulated for each survey set. The size of the text index in the MySQL database could be calculated using the SQL command in Listing 8.4. The command in Listing 8.4 returns the sum of all the indexes in a table. The size of the text index could be calculated using this command by dropping all the other indexes in the table. The size of the text index in MongoDB could be calculated by using the function shown in Listing 8.5. This function returns the details of all the indexes in the given collection which includes the index size as well. Table 8.4 shows the size of the text indexes in both the databases.

```
SELECT ROUND(SUM(index_length)/1024/1024, 4) "Index Size in MB"
FROM information_schema.tables WHERE table_schema=[DB Name] and
table_name=[Table Name];
```

Listing 8.4: SQL Command for Calculating the Text Index Size

```
db.[Collection Name].stats();
```

Listing 8.5: MongoDB Function for Calculating the Text Index Size

	MySQL	MongoDB
Survey Set A	0.0156 MB	0.0313 MB
Survey Set B	0.0156 MB	0.0391 MB
Survey Set C	0.0156 MB	0.0469 MB
Survey Set D	0.0156 MB	0.0625 MB
Survey Set E	0.0469 MB	0.0898 MB

Table 8.4: Size of the Text Indexes in MySQL and MongoDB for Different Survey Sets

The size of the MySQL text indexes is less than that of the MongoDB text indexes for each survey set. Therefore, we can conclude that the MySQL database is better than the MongoDB database in terms of the text index sizes for the full-text search as well.

8.6 Selectivity Analysis

In this section, we perform the selectivity analysis of the MySQL and MongoDB databases. Selectivity of a column in a database is defined as the fraction of the rows returned by a selection operation on that column. However, in this experiment, we do not perform the selectivity analysis on a column level. Instead, we do it on the question level. The main aim of this analysis is to compare the time taken to load a different number of questions from the MySQL and MongoDB databases. The questions are selected using the full-text search technique discussed in the previous section. Only the Boolean full-text search is selected for doing the selectivity analysis of the MySQL database as it performed better than the other two full-text search techniques in our evaluation in Section 8.5. Furthermore, the tests are done using the survey set E as it contains more questions. The survey set E contains 1280 questions in total. However, some of these questions are repeated ones, and the total number of unique questions in the survey set is 720.

For conducting the experiment, different keywords were selected that will retrieve a different number of questions from the databases. The keywords, number of questions selected from the databases using the keywords and the corresponding selectivity values are shown in Table 8.5. The average query execution times for these different selectivity values are shown in Figure 8.7.

Keyword	Number of Questions Retrieved	Selectivity (%)
Monat	9	1.25
Deutschland	63	8.75
Studium	126	17.5
Haben	216	30
Sie	513	71.25

Table 8.5: Selected Keywords and Corresponding Selectivities

In Figure 8.7, we can see that the execution times of the MySQL queries are less than the ones of the MongoDB database. Therefore, we can conclude that the MySQL

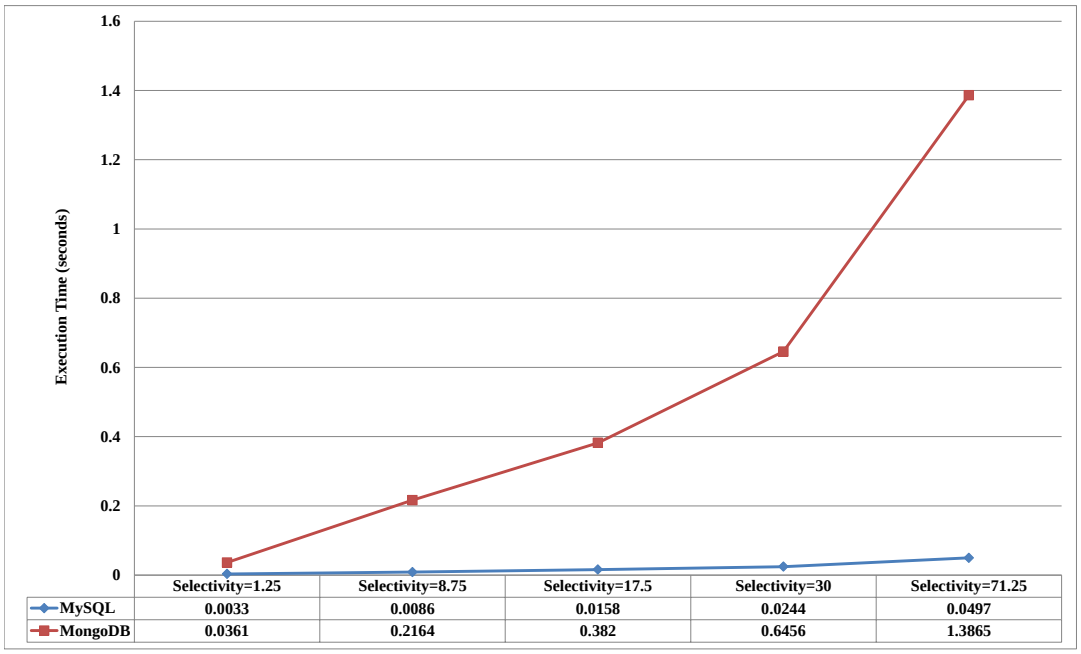


Figure 8.7: Query Execution Time - Selectivity Analysis

database performs better than the MongoDB database for different selectivity values. When we consider the performances of each database separately, we can see that the query execution times increases almost exponentially with the selectivity values for both the databases.

8.7 Summary

In this section, we have compared and evaluated the MySQL and MongoDB databases in terms of different parameters. Initially, the data loading analysis was done to compare the performance of the ETL workflows used for loading the data into the databases in terms of their execution time and throughput. In the data loading analysis, the ETL job for the MongoDB database performed better than the one for the MySQL database. Secondly, the data size and the index size of both databases were compared. In the case of data size, the MongoDB database performed better than the MySQL database for the smaller data sets. However, when it comes to larger data sets, the MySQL database has shown better performance. In the case of the total index size of the databases, the MongoDB database has a clear advantage over the MySQL database.

Furthermore, the performance of MySQL and MongoDB databases were evaluated for different data retrieval use cases. The MySQL database performed better than the MongoDB database in all the discussed data selection use cases in terms of query execution times. Additionally, the evaluation of the databases was also done in terms of the full-text search queries. The MySQL database showed better performance for

full-text search queries as well. Finally, the selectivity analysis was also done to compare both databases in terms of the time taken to load a different number of questions. The MySQL database was the winner in the selectivity analysis as well.

After the evaluation of our MySQL and MongoDB databases designed for managing the survey questions, we have found that the MySQL database performed better than the MongoDB database in four experiments out of the six experiments conducted. Therefore, we can summarize that the designed MySQL database is naturally the better option for managing the survey questions.

9. Conclusion and Future Work

In this thesis work, we have developed and compared two database systems for managing the survey questions at DZHW. We have started by providing the necessary background information for designing and developing the required database systems. It involves a brief description of the different types of surveys conducted at DZHW, followed by a discussion of the important characteristics of the database systems mainly focusing on SQL and NoSQL databases. Additionally, the major concepts and steps involved in the design process of SQL and NoSQL databases are explained. Finally, a brief introduction about the ETL process and Talend Open Studio, a commonly used open-source ETL tool, was also given as it is used for loading the data into the designed databases.

After discussing the necessary background details, a detailed requirement analysis was conducted. For this, the survey, "Die Studierendenbefragung in Deutschland", conducted at DZHW was analyzed. The structure of the survey and survey questions were identified, along with different types of questions, answer parts and some special cases present in the survey. Using these details, the various requirements and constraints for the database design were identified. Additionally, a high-level conceptual data model was designed based on the identified requirements. This model is used to create the required physical database schemas.

After the requirement analysis, we selected an SQL and NoSQL database for implementing the designed database schemas. MySQL was selected from different SQL databases and MongoDB was selected from different NoSQL databases for implementing the databases for managing the survey questions.

The SQL and NoSQL database schemas are then designed by expanding the conceptual data model. For the SQL schema, all the relations and their attributes are explained in detail. Similarly, for the NoSQL (MongoDB) database, all the collections and the description of the documents present in each of these collections were also explained in detail.

After designing the databases, the ETL workflows for loading the survey questions into the databases were introduced. The ETL jobs were developed using Talend

Open Studio for Big Data. Separate ETL jobs were developed for MySQL and MongoDB databases as the structure of the data is different in both databases. The different data processing steps involved in both the ETL jobs were also explained in detail.

After the implementation of the databases in MySQL and MongoDB, we compared and evaluated both the databases in terms of different parameters such as data loading times, storage space, query execution times, full-text search and selectivity analysis. The MySQL database outperformed the MongoDB database in terms of the data size, query execution times, full-text search and selectivity analysis. Therefore, we can conclude that the MySQL database is the better option for managing the survey questions as it offers better performance than MongoDB, especially in terms of query execution times and full-text search as they are the most frequently used features in the database. On the other hand, the MongoDB database showed an upper hand in the data loading analysis and the index sizes. However, the data loading operations only happen less frequently in the database and, the data retrieval speed should be given more importance over the storage space requirements of the database. Therefore, we conclude that the SQL database schema would be the better option for the management of survey questions.

As future work, we can implement the designed SQL and NoSQL database models in other databases as well. Since, in this thesis work, the main focus was given for the designing of the database schemas, we were not able to implement the designed models in other potential databases. So it would be worthwhile to investigate the performance of the database models in other SQL and document-oriented databases as well.

Furthermore, all the surveys at DZHW were not investigated for this thesis work. Only the survey, "Die Studierendenbefragung in Deutschland", was investigated for designing the database schemas. So in the future, all the remaining surveys and their questions should also be analyzed to verify whether our designed database models could incorporate them as well.

A. Appendix

```
db.SurveyCollection.aggregate(
[
    { "$unwind" :
        { path: '$SurveyModules', preserveNullAndEmptyArrays: true }
    },
    { "$unwind" : {
        path: '$SurveyModules.SurveyModuleQuestions',
        preserveNullAndEmptyArrays: true
        }
    },
    {
        $match : {
        'SurveyModules.SurveyModuleId' :
          ObjectId("61e485e02c858a2ffdfafc96")
        }
    },
    {
        "$lookup": {
            "from": 'QuestionCollection',
            "localField":
                'SurveyModules.SurveyModuleQuestions.QuestionId',
            "foreignField": '_id',
            "as": 'QuestionParts'
        }
    },
    { "$unwind" : {
            path: 'QuestionParts',
            preserveNullAndEmptyArrays: true
        }
    },
        { "$unwind" :
            {
                path:'QuestionParts.QuestionParts',
                preserveNullAndEmptyArrays: true
        }
    },
    {
```

```
        $set: {
            'SurveyModuleName':  {
                "$ifNull": ["$SurveyModules.SurveyModuleName", ""]
            },
            'SurveyModuleId': {
                "$ifNull": ["$SurveyModules.SurveyModuleId", ""]
            },
            'QuestionId':
                {"$ifNull":
                [
                "$SurveyModules.SurveyModuleQuestions.
                    QuestionId", ""
                ]
            },
            'QuestionNumber': {
                "$ifNull":
                [
                "$SurveyModules.SurveyModuleQuestions.
                    QuestionNumber", ""
                ]
            },
            'QuestionPart': {
                "$ifNull": ["QuestionParts.QuestionParts", {}]
            }
        }
    },
    {$unset: "SurveyModules" },
    {$unset: "QuestionParts"},
    {$unset: "_id"},
    {
        "$lookup": {
        "from": 'AnswerPartCollection',
        "localField": 'QuestionPart.AnswerPartId',
        "foreignField": '_id',
        "pipeline":
        [
            {"$unwind": "$AnswerPartChoices" },
            {
                "$lookup": {
                    "from": 'AdditionalAnswerPartCollection',
                    "localField":
                        'AnswerPartChoices.AdditionalAnswerPartId',
                    "foreignField": '_id',
                    "as": 'AdditionalAnswerPart'
                }
            },
            { "$unwind" :
                {
                    path: '$AdditionalAnswerPart',
                    preserveNullAndEmptyArrays: true
                }
            },
            {
                $set: {
                    'AnswerPartChoices.AdditionalAnswerPart':
                    {
                        "$ifNull": ["$AdditionalAnswerPart", {}]
```

```
                }
            }
        },
        {$unset: "AdditionalAnswerPart"},
        {$unset: "AnswerPartChoices.AdditionalAnswerPartId"},
        {$unset: "AnswerPartChoices.AdditionalAnswerPart._id"},
        {
            $group : {
                _id : {
                    _id: "$_id",
                    AnswerType: "$AnswerType"
                },
                AnswerPartChoices:
                {
                    $push: "$AnswerPartChoices"
                }
            }
        },
        {
            $set: {
                'id': {"$ifNull": ["$_id._id", ""]},
                'AnswerType':
                {
                    "$ifNull": ["$_id.AnswerType", ""]
                }
            }
        },
        {$unset: "_id"}
    ],
    "as": 'AnswerPart'
    }
},
{
    "$unwind" :
    {
        path: '$AnswerPart',
        preserveNullAndEmptyArrays: true
    }
},
{
    $set: {
        'QuestionPart.AnswerType': {
            "$ifNull": ["$AnswerType", {}]
        }
    }
},
{ $unset: "AnswerPart"},
{ $unset: "QuestionPart.AnswerPartId"},
{ $unset: "QuestionPart.AnswerPart.id"},
{
    "$lookup": {
        "from": 'AdditionalAnswerPartCollection',
        "localField": 'QuestionPart.AdditionalAnswerPartId',
        "foreignField": '_id',
        "as": 'AdditionalAnswerPart'
    }
},
```

```
{
    "$unwind" : {
        path: '$AdditionalAnswerPart',
        preserveNullAndEmptyArrays: true
    }
},
{
    $set: {
        'QuestionPart.AdditionalAnswerPart': {
            "$ifNull": ["$AdditionalAnswerPart", {}]
        }
    }
},
{ $unset: "AdditionalAnswerPart" },
{ $unset: "QuestionPart.AdditionalAnswerPartId" },

{
    $lookup: {
        from: "FootnoteCollection",
        let: {
            surveyModuleId: "$SurveyModuleId",
            questionId: "$QuestionId",
            partSequence: "QuestionPart.PartSequence"
        },
        pipeline: [
            {
                $match: {
                    $expr: {
                        $and: [
                            {
                                $eq: [
                                    "$SurveyModuleId",
                                    "$$surveyModuleId"
                                ]
                            },
                            {
                                $eq: [
                                    "$QuestionId",
                                    "$$questionId"
                                ]
                            },
                            {
                                $eq: [
                                    "$PartSequenceNumber",
                                    "$$partSequence"
                                ]
                            }
                        ]
                    }
                }
            }
        ],
        as: "Footnote"
    }
},
{ "$unwind" : {
        path: '$Footnote',
```

```
            preserveNullAndEmptyArrays: true
        }
    },
    {
        $set: {
            'QuestionPart.Footnote': {
                "$ifNull": ["$Footnote.FootnoteText", ""]
            }
        }
    },
    { $unset: "Footnote" },
    { $unset: "SurveyModuleId" },
    { $unset: "QuestionId" },
    {
        $group : {
            _id : {
                SurveyName: "$SurveyName",
                SurveyModuleName: "$SurveyModuleName",
                QuestionNumber: "$QuestionNumber"
            },
            QuestionParts: { $push: "QuestionPart" }
        }
    },
    {
        $set: {
            'SurveyName': {"$ifNull": ["$_id.SurveyName", ""]},
            'SurveyModuleName': {
                "$ifNull": ["$_id.SurveyModuleName", ""]
            },
            'QuestionNumber': {
                "$ifNull": ["$_id.QuestionNumber", ""]
            }
        }
    },
    {$unset: "_id"},
    {
        $set: {
            'QuestionDetails': {
                'QuestionNumber': "$QuestionNumber",
                'QuestionStructure': "$QuestionStructure"
            }
        }
    },
    {$sort: {"QuestionDetails.QuestionNumber":1}},
    {
        $group : {
            _id : {
                SurveyName: "$SurveyName",
                SurveyModuleName: "$SurveyModuleName"
            },
            Questions: { $push: "$QuestionDetails" }
        }
    },
    {
        $set: {
            'SurveyName': {"$ifNull": ["$_id.SurveyName", ""]},
            'SurveyModuleName': {
```

```
                "$ifNull": ["$_id.SurveyModuleName", ""]
            }
        }
    },
    {$unset: "_id"}
]);
```

Listing A.1: MongoDB Code for Selecting All Questions in a Survey Module

```
select
    S.survey_name, SM.survey_module_name, SMQ.question_number,
    QST.part_sequence_number, QST.level, QPTT.question_part_text,
    QPIT.question_part_instruction, FT.footnote_text,
    QST.variable_name, ATIT.answer_type,
    ATIST.choice_sequence_number, ATIST.choice_value,
    CTT.choice_text, CGT.choice_group_text,
    ATIST.additional_variable_name as
        AnswerPartAdditionalVariableName,
    AATIT.additional_answer_type as
        AnswerPartAdditionalAnswerType,
    AATIST.choice_sequence_number as
        AnswerPartAdditionalPartChoiceSequence,
    AATIST.choice_value as
        AnswerPartAdditionalAnswerPartChoiceValue,
    CTT2.choice_text as
        AnswerPartAdditionalAnswerPartChoiceText,
    QST.additional_variable_name, AATIT2.additional_answer_type,
    AATIST2.choice_sequence_number as AdditionalChoiceSequenceNo,
    AATIST2.choice_value as AdditionalChoiceValue,
    CTT3.choice_text as AdditionalChoiceText
from
    (
        select
            *
        from
            survey_questions_db.survey_module_table sm
        where
            survey_module_id=101
    )
    as SM natural join
    survey_questions_db.survey_table S natural join
    survey_questions_db.survey_module_questions_table SMQ
        natural join
    survey_questions_db.question_part_table QST natural join
    survey_questions_db.question_part_text_table QPTT left join
    survey_questions_db.question_part_instruction_table QPIT on
        (
            QST.question_part_instruction_id =
                QPIT.question_part_instruction_id
        ) left join
    survey_questions_db.footnote_table FT on
        (
            SM.survey_module_id = FT.survey_module_id
            and SMQ.question_id=FT.question_id
            and QST.part_sequence_number=FT.part_sequence_number
        ) left join
    survey_questions_db.answer_part_table ATIT on
```

```
        (
            QST.answer_part_id=ATIT.answer_part_id
        ) left join
    survey_questions_db.answer_part_choice_table ATIST on
        (
            ATIT.answer_part_id =
                ATIST.answer_part_id
        ) left join
    survey_questions_db.choice_text_table CTT on
        (ATIST.choice_text_id = CTT.choice_text_id) left join
    survey_questions_db.choice_group_table CGT on
        (ATIST.choice_group_id = CGT.choice_group_id) left join
    survey_questions_db.additional_answer_part_table AATIT on
        (
            ATIST.additional_answer_part_id =
                AATIT.additional_answer_part_id
        ) left join
    survey_questions_db.additional_answer_part_choice_table AATIST o
        (
            ATIST.additional_answer_part_id =
                AATIST.additional_answer_part_id
        ) left join
    survey_questions_db.choice_text_table CTT2 on
        ( AATIST.choice_text_id = CTT2.choice_text_id ) left join

    survey_questions_db.additional_answer_part_table AATIT2 on
        (
            QST.additional_answer_part_id =
                AATIT2.additional_answer_part_id
        ) left join
    survey_questions_db.additional_answer_part_choice_table AATIST2 o
        (
            QST.additional_answer_part_id =
                AATIST2.additional_answer_part_id
        ) left join
    survey_questions_db.choice_text_table CTT3 on
        ( AATIST2.choice_text_id = CTT3.choice_text_id )
;
```

Listing A.2: SQL Code for Selecting All Questions in a Survey Module

Bibliography

[AG03] Eugene Agichtein and Luis Gravano. Querying text databases for efficient information extraction. In *Proceedings of the International Conference on Data Engineering*, pages 113–124. IEEE, 2003. (cited on Page 18)

[AM10] Jasmin Azemović and Denis Mušić. Comparative analysis of efficient methods for storing unstructured data into database with accent on performance. In *Proceedings of the International Conference on Education Technology and Computer*, volume 1, pages V1–403–V1–407. IEEE, 2010. (cited on Page 17)

[BBC13] Shannon Bradshaw, Eoin Brazil, and Kristina Chodorow. *MongoDB: the definitive guide: powerful and scalable data storage*. O'Reilly Media, Inc., 2013. (cited on Page 43)

[BCAT14] Francesca Bugiotti, Luca Cabibbo, Paolo Atzeni, and Riccardo Torlone. Database design for NoSQL systems. In *Proceedings of the International Conference on Conceptual Modeling*, pages 223–231. Springer International Publishing, 2014. (cited on Page 12)

[BGJ03] Bernadette Byrne, Mary Garvey, and Mike Jackson. Using object role modelling to teach database design. *LTSN Workshop on Teaching, Learning and Assessment of Databases, Coventry*, 2003. Available online https://www.academia.edu/10766378/USING_OBJECT_ROLE_MODELLING_TO_TEACH_DATABASE_DESIGN. (cited on Page 12)

[Bow12] Jonathan Bowen. *Getting started with Talend Open Studio for data integration*. Packt Publishing Ltd, 2012. (cited on Page 14)

[BRA12] Alexandru Boicea, Florin Radulescu, and Laura Ioana Agapin. MongoDB vs Oracle – Database comparison. In *Proceedings of the International Conference on Emerging Intelligent Data and Web Technologies*, pages 330–335. IEEE Computer Society, 2012. (cited on Page 40)

[CB19] André Calçada and Jorge Bernardino. Evaluation of Couchbase, CouchDB and MongoDB using OSSpal. In *Proceedings of the International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 427–433. SciTePress, 2019. (cited on Page 42 and 44)

[CK19] Roman Ceresnak and Michal Kvet. Comparison of query performance in relational a non-relation databases. *Transportation Research Procedia*, 40:170–177, 2019. (cited on Page 7)

[Cod70] Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. (cited on Page 8)

[dlVGSB+18] Alfonso de la Vega, Diego García-Saiz, Carlos Blanco, Marta Zorrilla, and Pablo Sánchez. Mortadelo: A model-driven framework for NoSQL database design. In *Proceedings of the International Conference on Model and Data Engineering*, pages 41–57. Springer International Publishing, 2018. (cited on Page 13)

[EJN01] Robert A. Maksimchuk Eric J. Naiburg. *UML for Database Design*. Addison-Wesley Professional, 2001. (cited on Page 12)

[GGPO15] Cornelia Győrödi, Robert Gyorodi, George Pecherle, and Andrada Olah. A comparative study: MongoDB vs. MySQL. In *Proceedings of the International Conference on Engineering of Modern Electric Systems*, pages 1–6. IEEE, 2015. (cited on Page 93)

[GL12] Seth Gilbert and Nancy Lynch. Perspectives on the cap theorem. *Computer*, 45(2):30–36, 2012. (cited on Page 10)

[Gla05] Priscilla A Glasow. Fundamentals of survey research methodology. Technical report, The MITRE Corporation, 2005. (cited on Page 1 and 2)

[Gü94] Ralf Hartmut Güting. An introduction to spatial database systems. *The VLDB Journal*, 03:357–399, 1994. (cited on Page 42)

[Her13] Michael J. Hernandez. *Database Design for Mere Mortals: A Hands-on Guide to Relational Database Design, Third Edition*. Addison-Wesley Professional, 2013. (cited on Page 8, 9, and 12)

[HHLD11] Jing Han, Ee Haihong, Guan Le, and Jian Du. Survey on NoSQL database. In *Proceedings of the International Conference on Pervasive Computing and Applications*, pages 363–366. IEEE, 2011. (cited on Page 10)

[IAB+17] Khawar Islam, Kamran Ahsan, Syed Bari, Muhammad Saeed, and Syed Asim. Huge and real-time database systems: A comparative study and review for SQL Server 2016, Oracle 12c MySQL 5.7 for personal computer. *Journal of Basic and Applied Sciences*, 13:481–490, 2017. (cited on Page 39)

[KES16] Hema Krishnan, M.Sudheep Elayidom, and T. Santhanakrishnan. MongoDB – A comparison with NoSQL databases. *International Journal of Scientific and Engineering Research*, 07:1035–1037, 2016. (cited on Page 43)

[KK01] Kevin Kline and Daniel Kline. *SQL in a Nutshell*. O'Reilly Associates, Inc., 2001. (cited on Page 7 and 40)

[kK15] Rohit kumar Kaliyar. Graph databases: A survey. In *Proceedings of the International Conference on Computing, Communication Automation*, pages 785–790. IEEE, 2015. (cited on Page 11)

[KSS14] Veit Köppen, Gunter Saake, and Kai-Uwe Sattler. *Data Warehouse Technologien*. MITP-Verlags GmbH Co. KG, 2014. (cited on Page 13)

[Let15] Jerzy Letkowski. Doing database design with MySQL. *Journal of Technology Research*, 06, 2015. (cited on Page 11 and 39)

[LL10] Wei Li and Bo Lang. A tetrahedral data model for unstructured data management. *Science China Information Sciences*, 53:1497–1510, 2010. (cited on Page 18)

[MBNG15] Michael Madison, Mark Barnhill, Cassie Napier, and Joy Godin. NoSQL database technologies. *Journal of International Technology and Information Management*, 24:1–14, 2015. (cited on Page 10)

[MK19] Andreas Meier and Michael Kaufmann. *SQL & NoSQL databases*. Springer, 2019. (cited on Page 1)

[MS06] Imran R. Mansuri and Sunita Sarawagi. Integrating unstructured data into relational databases. In *Proceedings of the International Conference on Data Engineering*, pages 29–29. IEEE, 2006. (cited on Page 18)

[MS15] Nilesh Mali and SachinBojewar. A survey of ETL tools. *International Journal of Computer Techniques*, 02:20–27, 2015. (cited on Page 14)

[MSAL16] Michael J. Mior, Kenneth Salem, Ashraf Aboulnaga, and Rui Liu. NoSE: Schema design for NoSQL applications. In *Proceedings of the International Conference on Data Engineering*, pages 181–192. IEEE, 2016. (cited on Page 13)

[OBALB15] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih. Comparison and classification of NoSQL databases for Big Data. In *Proceedings of the International Conference on Big Data, Cloud and Applications*, 2015. (cited on Page 10 and 11)

[PPJ17] R. Poljak, Patrizia Poščić, and Danijela Jakšić. Comparative analysis of the selected relational database management systems. In *Proceedings of the International Convention on Information and Communication Technology, Electronics and Microelectronics*, pages 1496–1500. IEEE, 2017. (cited on Page 40)

[PWJD03] Roy P. Pargas, James C. Witte, K. Jaganathan, and J.S. Davis. Database design for dynamic online surveys. In *Proceedings of the International Conference on Information Technology: Coding and Computing*, pages 665–671. IEEE, 2003. (cited on Page 17)

[See09] Marc Seeger. Key-value stores: A practical overview. Technical report, Computer Science and Media, Stuttgart Media University, Stuttgart, 2009. (cited on Page 10)

[SEP95] Il-Yeol Song, Mary Evans, and E. Park. A comparative analysis of Entity-Relationship diagrams. *Journal of Computer and Software Engineering*, 03:427–459, 1995. (cited on Page 12)

[Sha15] Sugam Sharma. An extended classification and comparison of NoSQL big data models. *Computing Research Repository*, abs/1509.08035, 2015. Availabe online http://arxiv.org/abs/1509.08035. (cited on Page 11)

[SJDM15] Amlanjyoti Saikia, S. Joy, Dhondup Dolma, and R. Mary. Comparative performance analysis of MySQL and SQL Server relational database management systems in windows environment. *International Journal of Advanced Research in Computer and Communication Engineering*, 04:160–164, 2015. (cited on Page 39)

[SMP14] Biswajeet Sethi, Samaresh Mishra, and Prasant Kumar Patnaik. A study of NoSQL database. *International Journal of Engineering Research Technology*, 03:1131–1135, 2014. (cited on Page 10)

[SMP15] Milorad Stević, Branko Milosavljević, and Branko Perisic. Enhancing the management of unstructured data in E-learning systems using MongoDB. *Program electronic library and information systems*, 49:91–114, 2015. (cited on Page 19)

[Sri17] K. T. Sridhar. Modern column stores for big data processing. In *Proceedings of the International Conference on Big Data Analytics*, pages 113–125. Springer International Publishing, 2017. (cited on Page 11)

[Vas09] Panos Vassiliadis. A survey of extract–transform–load technology. *International Journal of Data Warehousing and Mining*, 05(3):1–27, 2009. (cited on Page 13)

[VM21] Igor Vershinin and A. R. Mustafina. Performance analysis of PostgreSQL, MySQL, Microsoft SQL Server systems based on TPC-H tests. In *Proceedings of the International Russian Automation Conference*, pages 683–687. IEEE, 2021. (cited on Page 39)

[WAA02] Michael Widenius, David Axmark, and Kaj Arno. *MySQL Reference Manual*. O'Reilly Media, Inc., 2002. (cited on Page 39)

[WP15] Rahmadi Wijaya and Bambang Pudjoatmodjo. An overview and implementation of extraction-transformation-loading (ETL) process in data warehouse (Case study: Department of agriculture). In *Proceedings of the International Conference on Information and Communication Technology*, pages 70–74. IEEE, 2015. (cited on Page 13)

[YAOI13] Wael Yafooz, Siti Zaleha Zainal Abidin, Nasiroh Omar, and Zanariah Idrus. Managing unstructured data in relational databases. In *Proceedings of the IEEE Conference on Systems, Process Control*, pages 198–203. IEEE, 2013. (cited on Page 18)

[ZBM20] Cristofer Zdepski, Tarcizio Alexandre Bini, and Simone Nasser Matos. New perspectives for NoSQL database design: A systematic review. *American Academic Scientific Research Journal for Engineering, Technology, and Sciences*, 68(1):50–62, 2020. (cited on Page 12)

Statement of Authorship

Thesis: **Designing a Database Schema for Survey Questions**

Name: **Kurian** Surname: **John**

Date of birth: **23.12.1994** Matriculation no.: **221425**

I herewith assure that I wrote the present thesis independently, that the thesis has not been partially or fully submitted as graded academic work and that I have used no other means than the ones indicated. I have indicated all parts of the work in which sources are used according to their wording or to their meaning.

I am aware of the fact that violations of copyright can lead to injunctive relief and claims for damages of the author as well as a penalty by the law enforcement agency.

Place, Date

Signature, Name