Universiteit Leiden
The Netherlands

Opleiding Informatica & Economie

Design of a database supporting the exploration of historical documents and linked register data

Ricardo Ruud Johannes Schreuder

Supervisors:
Prof.dr.ir. W. Kraaij & Ir. M.K. van Dijk

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

13/06/2022

Abstract

We hereby present the design of the central database model and its implementation for the Linking University City and Diversity project. The project's focus is to develop software to help to answer several questions about the history of Leiden University. SQL and NoSQL databases were explored to decide which database management system was the best choice for the software solution. The exploration showed that the MySQL relational database was the type of database that provides the required non-functionals best. The aim of the data model was to give answers to research questions about the mobility, social integration, and geographical segregation of scholars and students at Leiden University from 1575 on. The entities, relations between them, and their attributes were investigated based on several high-quality data sources collected by historians. This resulted in a conceptual UML class diagram. MySQL Workbench – a design tool of the MySQL database - was used for this project to implement this conceptual data model by converting it to an EER diagram.

Acknowledgements

This thesis is written for the Leiden Institute of Advanced Computer Science at Leiden University as part of the bachelor's degree in Computer Science & Economics. I would like to express my deepest gratitude to Prof.dr.ir. W. Kraaij & Ir. M.K. van Dijk for the support and feedback received during my thesis. I am also thankful to Liam van Dreumel and Michael de Koning for our sincere collaboration during this project. Lastly, I would like to mention Prof.dr.ir. J.M.W. Visser and Prof.dr. A. Schmidt for their feedback.

Contents

1 Introduction

This thesis is part of a bigger project called the Linking University City and Diversity, also known as the LUCD project. It is an interdisciplinary project between the digital humanities and computer science faculties[Kraaij and Schmidt, 2021]. The project consists of the core and bachelors student team. The core team consists of Wessel Kraaij (project leader), Ariadne Schmidt (historian) and Joost Visser (software expert and data scientist). Richard van Dijk (software architect) acts as a bridge between the core and bachelors student team, reporting to the core team. The bachelors student team consists of Michael de Koning (bachelor student: focusing on the adapter), Liam van Dreumel (bachelor student: focusing on the front end) and Rick Schreuder(me a bachelor student: focusing on the database). The project strives to make a visual representation, in the form of an interactive dashboard. It will contain information regarding Leiden University and the city of Leiden both qualitative and quantitative, from 1575. This will be done through data science techniques and by combining various data sources, which can be structured, semi-structured or unstructured. To realize this there must be a strong foundation, a database, where the front-end can further develop on. This thesis reports on the design and creation of this database.

1.1 Objectives

The LUCD project can be visualized in a software architecture, this is done in Figure 1. As seen in the figure there are three main components the adapter, database and front end. These three components combined will form the foundation of the project. The workflow component connects these three components, this will later be discussed. The ultimate goal is to provide researchers and the general public with information about the history of Leiden University and the city of Leiden interactively. Specifically for the historians, the data that is in the database must be able to answer specific research questions. Example research questions are:

- Geographical segregation: Where did students or professors live in Leiden?
- Geographical segregation: What is the diversity of people coming in and out of the city through the time?
- Social network: Who were students' parents and children and whom did they marry?

To be able to answer these research questions, data has to be converted via an adapter to the database. This is out of scope for this thesis, but will be reported on in the thesis by Michael de Koning. There exist different types of data, which are structured, semi-structured and unstructured data. Data is structured when it follows a predefined schema, typical examples are relational database systems. Semi-structured data is self-describing for both machines and humans and has a schemaless property. A benefit of semi-structured data is the ability to have variations in data structures. Lastly, unstructured data, as the name suggests has no structure, it cannot be stored in rows and columns, examples are images, pdfs and videos [Sint et al., 2009]. The adapter converts these types of data to the database structure created in this thesis. Consequently, the front end, out of scope for this thesis, but will be reported on in the thesis by Liam van Dreumel, displays the data, gathered from the database, on a website. The website is publicly accessible and can therefore be used by both researchers and the general public. The workflow is the mechanism that combines and connects all three components. The workflow can also be called the data manager.

The data manager has three functions, these are scheduling, adding data and linking data with data consistency checks. Scheduling checks repeatedly if certain tasks still need to be executed, if so it executes. The addition of data can be done in two ways. The front end requests data, this will come from the database or the adapter adds data to the database. This new data needs to be linked to existing data followed up by data consistency checks to ensure the data is still consistent. All operations are managed by the workflow.

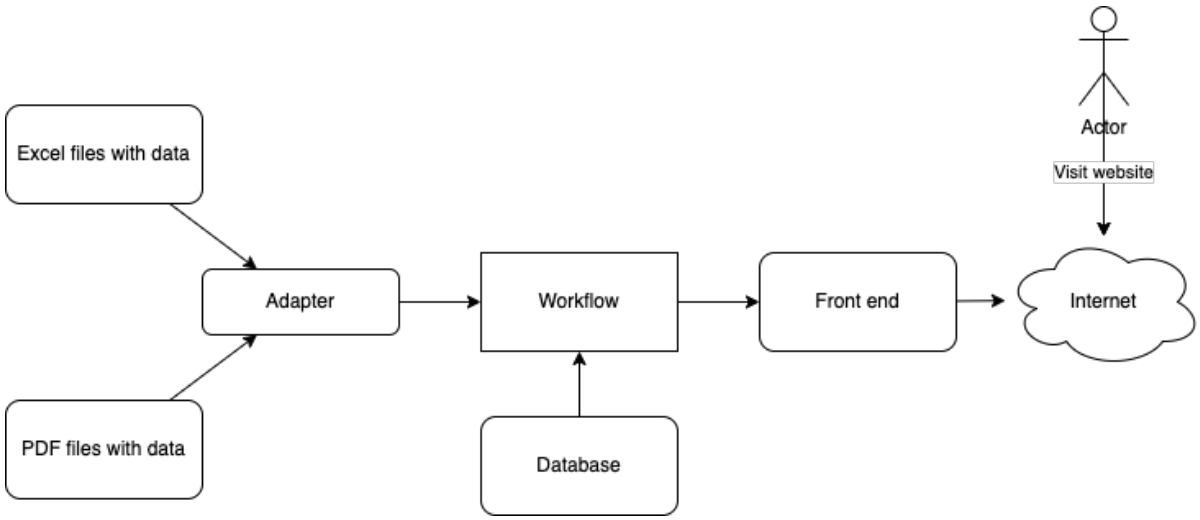

Figure 1: Software architecture of the LUCD project.

This thesis will endeavor to supply a database architecture which can indirectly answer the above mentioned research questions and thus will focus on the back end side of the LUCD project. Currently, there are various sources of data scattered around in all kinds of spreadsheets and database management systems (DBMS), such as Microsoft Excel and Access. A DBMS is a software system that stores, retrieves and can then execute queries on data. It operates as an interface between the end-user and the database. The end-user can create, read, update and delete data in the database, also known as the CRUD operation [app,]. Aforementioned, data sources are non-generalized and scattered across databases. This thesis intends to improve efficiency and create a clear overview of all data sources. To do so, a generic structure for all data sources is made, through a database model, where data is retrieved only for specific research questions. There are many different database management systems. Therefore the non-functional requirements, such as quality characteristics and the CAP theorem must first be determined. The database management system which best suits the requirements will be used to create the architecture. This leads to the research question of this thesis:

What database architecture is most suitable for the LUCD project?

The remainder of this chapter is organized as follows. Section 1.2 will briefly mention the sub research questions of this thesis. Then, the contributions of this thesis are named in Section 1.3. Finally, Section 1.4 elaborates on the thesis overview.

1.2 Sub research questions

The research question can be further divided into sub research questions.

RQ1. What database management system is suitable for the LUCD project?

RQ2. What are the entities and attributes according to the available data sources?[1]

RQ3. How can the entities and attributes and their corresponding relations be defined in a conceptual UML diagram?

1.3 Contributions

The main contribution of this bachelor thesis is the design of a database architecture, specifically made for the Linking University City and Diversity project. This is made in MySQL Workbench, through an EER diagram. There will also be a conceptual UML diagram available for further development, if an expansion of the database architecture is desired.

1.4 Thesis overview

The structure of this thesis is as follows. Firstly, related work will be briefly discussed in Section 2. Secondly, several types of DBMSs are extensively mentioned in Section 3, followed up by selecting a DBMS in Section 4. Next up, in Section 5 the database model will be made, including a UML diagram, EER diagram. This section will specifically go deeper into the designing and explaining of the database model. Followed by the discussion, conclusion and further work in Section 6.

[1] RQ2 does not include relations, since these are logically better explained in RQ3.

2 Related work

This section will briefly discuss related work to this thesis. First, related work on SQL and NoSQL databases will be explored. It is focused on the query performance of the various databases and general usages for NoSQL databases. Then, related work on the design of a database model for historical data will be explored.

2.1 SQL and NoSQL

In 2018, Chang, M.-L.E., & Chua, H. N. found that the performance of SQL and NoSQL databases, with semi-structured data, are different. They examined MongoDB and MySQL, respectively NoSQL and SQL databases, on a logarithmic scale and concluded that these DBMSs are equal in performance up to 10000 rows, after that MongoDB performance outperforms MySQL. However, MySQL has the upper hand regarding performance tuning features.[Chang and Chua, 2018]

Another paper conducted a performance test on three different relational databases. The databases on which the test was performed were MySQL, MonetDB and SQLite. In a three series query, the databases were tested on data retrieval of specific instances, becoming gradually more complex towards the last query. The performance was measured based on execution time. SQLite and MySQL had roughly the same results, except for the last query here MySQL indisputably outperformed SQLite. MonetDB was the best performing database in this research.[Martorelli et al., 2020]

In [Abubakar, 2014] popular open source relational database management systems were evaluated, such as MySQL, PostgreSQL, MariaDB and SQLite. These databases will all be used in this thesis as well and therefore this paper is a suitable benchmark. The paper used the YCBS tool to evaluate each database and the evaluation was done based on three types of workloads, which are INSERT, READ and UPDATE, ranging from 1MB to 1GB. PostgreSQL had the best performance, followed by MySQL, SQLite and MariaDB on the inserts. Next up, regarding updates PostgreSQL and MariaDB are shared first and MySQL and SQLite had considerably worse performance. Lastly, MariaDB topped the ranking, ensuing PostgreSQL, SQLite and MySQL, respectively. This paper from 2014, concludes that PostgreSQL has the best performance of these open source RDBMS.

The aforementioned databases were mostly relational databases, this paper [Gupta et al., 2017] evaluates MongoDB, Cassandra, Neo4j and Redis, all four categories of NoSQL databases, on both nonfunctional and functional requirements. The outcomes are summarized as follows: Redis should be chosen only for high performance and simplicity, Cassandra is well suited for semi structured data that can be represented in a columnar format, MongoDB is preferred when using JSON like formats resulting in high performance, flexibility and usually high scalability and last but not least Neo4j, Neo4j is an optimal database if the graph theory represents the data and provides high stability, however performance and scalability is variable.

2.2 Database model

The project "Digitised Manuscripts to Europeana"(DM2E) converted metadata and contents formats describing and representing digital cultural heritage objects such as manuscripts, letters, books, images and journal articles into the Europeana Data Model. The model operates with a hierarchical structure and with metadata to give information about the data. Resulting in the following: Whenever a data retrieval request is done, specific data can be asked. The model uses SPARQL, a standard query language for linked open data and RDF triplestores. Triplestores is a database, which only returns triples, a triple is a data entity composed of subject-predicate-object: Rick-lives in-Leiden. The core of the infrastructure consists of a Jena TRB triplestore, creating balance between maintainability, scalability and versatility. Thus this model uses triplestore to store historical data, which can be used in the digital humanities.[Baierer et al., 2017]

However, there are more ways to model historical data. The paper of Robert Nasarek suggests that complex historical data should not be stored by triples, but with a property graph-database like Neo4j. He states that a property graph has the ability to assign literals to properties and SPARQL, while most triplestores do not. The benefit of assigning literals to properties is that it stores meta data correctly and avoids undue reification[2].[Nasarek, 2019]

[2]Reification: The process of converting an abstract idea regarding a computer application into an explicit data model

3 Types of DBMS

There are two major types of DBMSs, which are SQL and NoSQL databases. This can be further divided in more detail, see Figure 2. In this thesis, only non-commercial SQL and NoSQL databases will be evaluated and further explained in Section 3.1 and Section 3.2, including the top four SQL databases and the most popular database of every NoSQL type. The weaknesses and strengths of their corresponding databases and their types will be discussed in their subsections. See Figure 2 for a preview of the different types of databases and Figure 3 for the ranking [db-, 2022a] of several popular databases.[Meiryani, 2019]

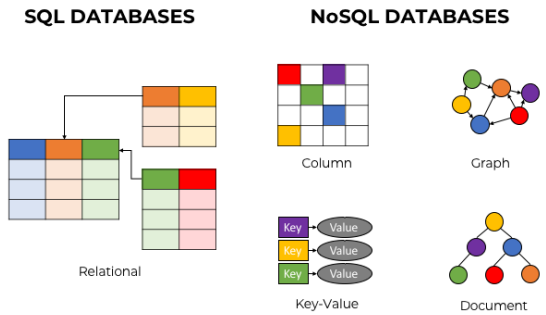

Figure 2: Four types of NoSQL and SQL databases[Lo, 2021].

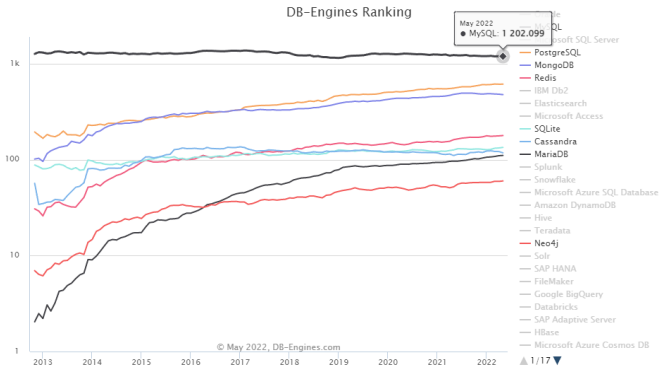

Figure 3: Ranking of databases[db-, 2022a].

3.1 SQL

In 1970 the relational model for database management was invented by Edgar F. Codd, who was a researcher at IBM. Edgar F. Codd wanted to create a better model than IMS, the hierarchical model of IBM, because it lacked flexibility and stated that data independence was needed. Then, changes in query, updates of data and the growth of a variety of data types could lead to disruptive changes in data representation. Therefore he invented database normalization. Database normalization is the process of organizing columns and tables to improve data integrity and reduce data redundancy.[Codd, 1970]

Edgar F. Codd further developed this in 1971 in the paper: "Further normalization of the data base relational model" In this paper he proposed three normal forms, which are still used today. These are the first, second and third normal form, currently known as 1NF, 2NF and 3NF. When no normal form is applied it is called an unnormalized form. To explain the first three normal forms of Egar F. Codd's, three concepts need to be explained. These are *candidate key*, *prime* and *transitively*. A candidate key is a column that can uniquely identify each record in a table, it can always be a primary key. Prime attribute is an attribute which is a part of the primary key. Transitively states that a relation is transitively dependent when an attribute determines another attribute indirectly.

1NF A relation is in 1NF if, no attribute domain has relations as elements, or the atomicity of the table is 1. Stating that a cell cannot hold multiple values.

2NF A relation is in 2NF if, it is in 1NF and every non-prime attribute of the relation is fully dependent on each candidate key of the relation.

3NF A relation is in 3NF if, it is in 2NF and every non-prime attribute of the relation is non-transitively dependent on each candidate key of the relation.[Codd, 1971]

At the time there was no standard way to use the relational model. Therefore, IBM's computer scientists Donald D. Chamberlin and Raymond F. Boyce developed a query language for a relational database, known as SEQUEL. SEQUEL is short for Structured English Query Language, but was shortened to SQL (Structured Query Language), due to trademark patents. A SQL database is the same as a query language for relational databases. As the name suggests, it is the most useful for handling structured data. It is used to store, modify and retrieve data inside a relational database.

A relational database is used for data that is related to one another. This relational data can be presented in a table, which is intuitive and understandable to interpret. In a table, rows serve as records that each have their unique ID and one or several attributes with values, which are represented as columns. This structured approach makes it straightforward to create relations between the individual data points.

SQL is the standard language for relational database management systems (RDBMS) and serves as the core for all these systems. According to a renowned database ranking website [db-, 2022a], popular examples of these systems are MySQL, PostgreSQL, MariaDB and SQLite, these will be further discussed in their respective sections. Queries in SQL are consistent and therefore easier to construct and perform. The relational model itself is also very consistent even when used with multiple database instances because of the relational structure between the data points.

All modifications inside a relational database are done through transactions, which are all the operations that are performed as one sequence. It is important for an SQL database to stay consistent with every database modification. To ensure this, the relational database is defined by four properties called ACID[Mohamed et al., 2014]:

Atomicity A transaction is only executed if all operations within the transaction are executed. Otherwise, none of the operations will be performed.

Consistency The database should always stay consistent. Transactions cannot compromise the consistency of the database, also known as strict consistency.

Isolation Every transaction stands on itself and cannot be altered or affected by other transactions that are performed at the same time.

Durability Changes in data inside the database after a transaction are permanent.

An example of this strict consistency is a bank transaction. A balance of a bank account should always be the same on different servers, if this is not the case a person might think he has less or more money than he actually has.

3.1.1 Advantages

SQL databases have some general advantages. SQL queries are fast and quickly retrieve large amounts of data from the database, SQL uses standardized established language with lots of documentation. Using SQL is user-friendly and easy to learn and understand. SQL also uses tables, which are perfect for relational data. It is also possible to embed SQL into other languages. Another feature of SQL is the possibility for stored procedures and functions within databases. Lastly, SQL treats derivability, redundancy and consistency of relations well.[Codd, 1970]

3.1.2 Disadvantages

There are also some general disadvantages. These are: joins, which are computationally exhaustive, scalability with SQL is more difficult, data views need to be updated when underlying databases are changed, SQL does not keep up with rapid development due to the lack of flexibility.

3.1.3 SQL databases

The following section will elaborate on each relational database management system mentioned earlier. The advantages and disadvantages of every RDBMS will be highlighted. The sections will predominantly focus on the user perspective, based on internet sources. See Section 3.1.3.1 for MySQL, Section 3.1.3.3 for PostgreSQL, Section 3.1.3.2 for MariaDB and Section 3.1.3.4 for SQLite.

3.1.3.1 MySQL

MySQL is among the most popular DBMS in the world [db-, 2022a]. It is an open source and free relational database, which can be run on popular platforms, such as Linux, Windows and MacOS. MySQL was acquired in 2008 by Sun Microsystems and consequently acquired by Oracle in 2010 [Romanowski, 2020]. The database uses standard SQL and is considered consistent, secure and fast. When using simple queries the performance is as good as other databases, however, when using loads of data and multiple joins the performance falls. MySQL also offers a unified visual database GUI called MySQL Workbench, which is developed by Oracle thereby ensuring compatibility with the latest features. MySQL Workbench adds functionalities to MySQL, where the developer can model data, develop with SQL and has administration tools for configuration. Lastly, MySQL uses multiple storage engines InnoDB and MyISAM, whereas MyISAM does not support ACID, InnoDB does. In web applications MySQL is a frequently used database, including large companies Facebook and Uber. MySQL has some disadvantages as well, it does not have a good developing and debugging tool, SQL check constraints are not supported and it is relatively inefficient in handling transactions.[Simplilearn, 2021]

3.1.3.2 MariaDB

MariaDB was created by the original developers of MySQL, they build on top of MySQL to create MariaDB, it essentially is a fork of MySQL. It is guaranteed to stay open source, according to the founders [MariaDB, 2019]. MariaDB is a relational database management system, having similar functions as MySQL, such as MySQL protocol and clients. The RDBMS has been steadily growing over the past decade.

3.1.3.3 PostgreSQL

PostgreSQL, also known as Postgres, is a free open source RDBMS. PostgreSQL was founded in 1986 by Michael Stonebraker and has since been continuously developed. According to themselves, PostgreSQL is the most advanced open source database. The database supports both non-relational and relational data. PostgreSQL has many extensions, which provide advanced features. The database supports various data types, such as SQL, JSON, XML, key-value and spatial data. PostgreSQL is also known for its reliability, flexibility, stability and matureness. A disadvantage of PostgreSQL is that it has a relatively low reading speed.[Kamaruzzaman, 2021]

3.1.3.4 SQLite

SQLite is a free and open source RDBMS. SQLite is not a standalone app, but rather a library to be embedded in apps, therefore it is an embedded database. SQLite follows the same structure as PostgreSQL. SQLite only loads the data which is needed and overwrites what is edited, so no superfluous data is loaded or overwritten. Furthermore, third-party tools are widely accessible via SQLite. However, SQLite is limited in its data storage up to 281 terabytes, since it is stored in a single disk file.

3.2 NoSQL

NoSQL is the direct opposite of SQL, it is therefore also known as 'Not Only SQL'. The Not only SQL database was created, because the traditional databases were pushed against their boundaries, as a consequence of the increase in data generation. SQL databases could not handle the enormous loads of data and had drops in performance. NoSQL was the only database that could manage unstructured data, without lack of performance [Lacefield, 2018]. This type of database also reduces the complexity, since NoSQL is schema-free, this is a major benefit in comparison to SQL [Jatana et al., 2012]. NoSQL is a non-tabular database class, and comes in a variety of types:

Document database Stores and queries data as JSON-like documents.

Key-value database Simple type of database where each item contains keys and values.

Wide-column database Data is stored in columns rather than rows.

Graph database Stores data in nodes and edges, where edges represent the relation and a node the object. It provides no range queries.

Triplestore Stores data only in triples and is a subset of graph databases, an example is Apache Jena which will be explored in Section 3.2.3.5. It consists of a subject, predicate and object. The predicate is the edge, which defines the relation between the subject and object. Triplestores use SPARQL as their query language. SPARQL is a Resource Description Framework (RDF) query language, which allows users to retrieve and manipulate data stored in RDF format. RDF data, in terms of relational databases, is a table with three columns representing the triples. SPARQL uses common SQL query operations such as JOIN, SORT and AGGREGATE. Lastly, SPARQL is used for Linked Open Data, which is a combination of linked and open data - that is, creating links between datasets and freely usable and distributable data, respectively. The main difference between SPARQL and SQL is that the first operates with RDF data and the latter with relational data.

These types all offer flexible schemas and scalability, which are NoSQL pillars. These flexible schemas allow developers to store enormous amounts of unstructured data, giving them, as said before, lots of flexibility. NoSQL scales horizontally, which allows these loads of unstructured data.

Aforementioned, the most popular database of each type of NoSQL will be examined, these are MongoDB, Neo4j, Redis and Cassandra [db-, 2022a]. These databases all have their strengths and weaknesses, this will be discussed later.

Many NoSQL databases do not support joins, which can sometimes make it difficult to retrieve combined information. Instead, they are denormalized, meaning they have all the information of a certain object in a single document. This increases speed performance, but, as mentioned before, makes it harder to acquire merged data.

As mentioned in Section 3.1 SQL stays consistent through ACID. This is a vital set of guarantees, which is not always supported by NoSQL. Therefore, in NoSQL databases data consistency models can be notably different. However, there exists an opposite of ACID, which is BASE. NoSQL are classified somewhere between both ACID and BASE. BASE is short for [Mohamed et al., 2014]:

Basically Available The database is permanently available.

Soft State The developers are responsible for the consistency of the database. Meaning the state of the system can change over time.

Eventual consistency The transaction will not be directly be processed, but eventually.

BASE does not ensure direct consistency, however, some transactions satisfy eventual consistency. For example, Facebook's feed does not need to be strongly consistent, since users can enjoy the feed without having the latest update of a friend. Also, all distributed systems that aim to provide high availability and partition tolerance (see Section 4) are built on top of eventual consistency.

3.2.1 Advantages

NoSQL has, of course, also some general advantages. NoSQL is renowned for its horizontal scaling, keeping up with rapid development and allowing for much easier flexibility than relational models.

3.2.2 Disadvantages

A general disadvantage of most NoSQL databases is that they do not fully support ACID transactions. They also lack standardization, proper backup systems, reliability and consistency. Regarding the latter, when entering the same dataset again in a database, NoSQL would usually give no warning, leading to duplicates.

3.2.3 NoSQL databases

The following section will elaborate each NoSQL database management system mentioned in Section 3.2. The advantages and disadvantages of every DBMS will be highlighted. See Section 3.2.3.1 for Neo4j, Section 3.2.3.2 for MongoDB, Section 3.2.3.3 for Redis and Section 3.2.3.4 for Cassandra. Apache Jena, mentioned in the related work, will also be briefly discussed in Section 3.2.3.5.

3.2.3.1 Graph database: Neo4j

Neo4j is the leading graph database management system in the world. The prototype was created in 2007 with the idea to store relationships as first-class entities. Neo4j operates with nodes and edges, nodes represent the object or entities and edges connect the nodes, representing the relations between objects or entities. Nodes are always in a direction and can have properties. The graph database uses its own Cypher Query Language, since there are no tables and thus no joins required the queries are usually a lot easier and faster than SQL. Neo4j can also be embedded into an application where it can be used as a library. However, Neo4j does not have in-built data encryption and is only vertically scalable.

3.2.3.2 Document database: MongoDB

MongoDB is a document-based database management system and non-relational. MongoDB is free and open source, which was founded in 2009 and has since been continuously developed. Document databases stores data in JSON-like documents, which makes them flexible and supporting

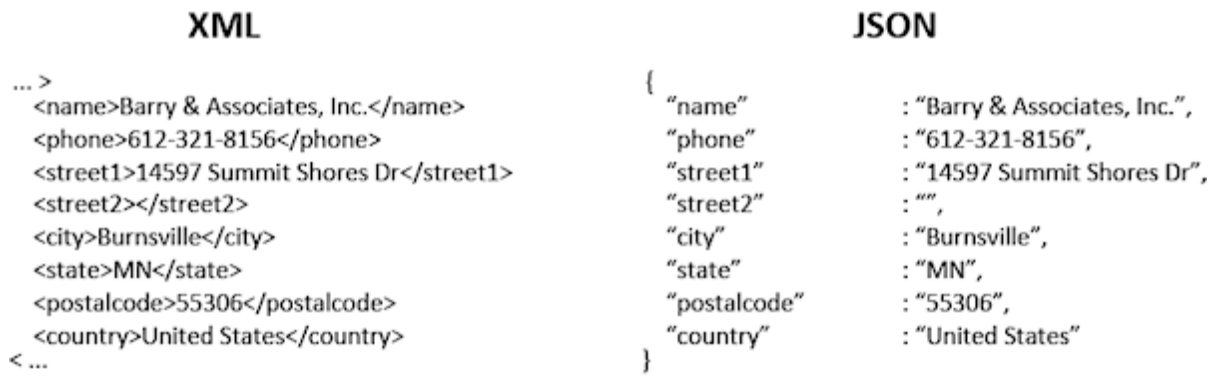

Figure 4: Difference between XML and JSON storage[Barry,].

semi-structured data. These JSON-like documents are self-contained, which allows them to be easily distributed and thus it is scalable. If data is stored in XML it must first be transformed into a JSON store. See the difference in storage between JSON and XML in Figure 4. JSON is a lightweight data-interchange format based on JavaScript that stores data in attribute-value pairs and is easy to read, while XML is a markup language that was designed to store data into a native XML type rather than read it. The performance is considerably better than relational databases and queries are written in MongoDB Query Language based on JavaScript [Chang and Chua, 2018]. Furthermore, MongoDB is ACID-compliant and thus making it an consistent database. MongoDB should not be used for structured or relational data.[Kamaruzzaman, 2021]

3.2.3.3 Key-value database: Redis

Redis is an open source in-memory Key-Value database management system. Key-Value databases store data in collections of Key-Value pairs. The key and value can be anything, from an integer to a date. Redis supports a variety of data structures, such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams, resulting in lots of flexibility.[red,][Kamaruzzaman, 2021]

3.2.3.4 Wide-column: database: Cassandra

Cassandra is a free and open source wide-column database management system, which was founded in 2008 by Avinash Lakshman and Prashant Malik. This database was created on behalf of Facebook, since at that time databases could not scale with the huge amount of data. Cassandra, a highly scalable database for structured, semi-structured or unstructured data solved this problem. It is built to handle up to several petabytes per second. A wide-column database stores data in rows and columns, however, unlike RDBMS, Cassandra allows rows to have different types and names, therefore it could also be considered a two-dimensional Key-Value database [db-, 2022b]. These databases are mostly used in data warehouses, where massive data is stored, for business intelligence analysis. It has a SQL like query language, which is user-friendly, called Cassandra Query Language (CQL). Data is stored and copied in different locations through the hashing algorithm, which makes it reliable. The absence of aggregates and subqueries are drawbacks of Cassandra.[Kamaruzzaman, 2021][doc,]

3.2.3.5 Triplestore: Apache Jena

In addition, Apache Jena will also be briefly explored in this section, as it was mentioned in the related work. However, it will not be taken into account during the evaluation of the different databases, because it was found after the evaluation procedure. Apache Jena is one of the most popular RDF triplestore databases, founded in 2000 by HP labs. It is a database that stores data in triples. The triplestore is classified as a subset of a graph database, as it operates with triples which have the same purpose as graphs. It is specifically designed for creating semantic web and linked data applications. Schemas are built in the RDF schema or Web Ontology Language (OWL). The schemas allow for structured, semi-structured and unstructured data, due to the simplicity of the triplestore model itself. Triplestores are easy to query, standardized (unless many NoSQL databases, making it easier to switch between triplestore databases), provide schema flexibility and allow reasoning. The latter infers based on existing data new knowledge, resulting in possible new relations (note: this is inferred by a database, thus it must always be checked by a historian). Drawbacks are the matureness of reasoning, it is not always 100% accurate. Another drawback is the fact that some knowledge or data cannot be stored in triples. Designing a workaround for this problem is time-consuming.[Ontotext, 2014]

4 Selecting a DBMS

All eight databases have been explained in Section 3. To decide what database management system to use for creating the database model, two quality criteria are defined. The first quality criteria is a set of quality characteristics, which is defined together with the companions of the LUCD project and the ISO-25010 standard [ISO, 2011]: availability, consistency, flexibility, integrity, performance, reliability, security, scalability. The second quality criterion is the CAP Theorem.

The CAP theorem is a renowned evaluation method for mostly NoSQL databases. It stands for:

Consistency Data retrieved by a user must always be the most up-to-date and correct data, even right after an update or delete.

Availability Ability to operate continuously, without having downtime.

Partition tolerance Despite a network failure, the system must continue to work, i.e. network partition. This is done by allowing copies of data in multiple places, over a network.

The CAP theorem states that a NoSQL database can only provide two of the previously mentioned guarantees, which implies that there are the following combinations availability and partition tolerance (AP), availability and consistency, (AC) and consistency and partition tolerance (CP). Where users may read inconsistent data, can experience network problems and have the risk of data becoming unavailable, respectively. However, in practice, according to Brewer, the general belief is that partition tolerance cannot be forfeited, for NoSQL. Therefore, a choice has to be made between availability and consistency. NoSQL favors availability (BASE) and relational databases consistency (ACID). All RDBMS and Neo4j are AC, MongoDB and Redis are CP and Cassandra AP.[Lourenço et al., 2015, Brewer, 2012]

The above mentioned quality characteristics will each be briefly explained in Table 1, afterwards these will be evaluated on each databases in Table 2. The outcome is represented in a table.

4.1 Quality characteristics

In Table 1, all chosen quality characteristics are explained. The quality characteristics are ranked in order of importance for this project.

Quality characteristic	Description
Reliability	Degree to which a system, performs a specific function under a specific condition for a specific period of time. It essentially means that is should be doing exactly what is asked. If this does not happen a downtime can occur and can cause end users to not be able to retrieve data and thus unable to use the website.
Consistency	As seen in the CAP theorem, consistency ensures that all nodes see the same data at the same time. However, it also guarantees that database constraints are not violated meaning that a database transaction can only affect the data according to the defined rules.

Security	A system should protect its information and data so that only persons or systems can access data according to their types and levels of authorization.
Availability	The system should be accessible whenever someone wants to use it.
Performance	Performance shows the response time on certain actions, usually the faster the better. It can be measured in many different ways, i.e. latency and channel capacity, responding time on action and number of actions occurred in certain point in time, respectively.
Integrity	Ones who is not authorized must not be able to access or modify certain data or systems.
Scalability	Handling increases of many data should not decrease the performance of a system, it should be scalable either horizontally or vertically.
Flexibility	It represent the amount of adaptability for a certain system to different environments and non-predefined requirements. Moreover, the ability to cooperate with third party components, data modification and changes to the database architecture itself.

Table 1: Description of quality characteristics[Ashanin, 2018, ISO, 2011, Offutt, 2002].

All quality characteristics are defined in Table 1. The following databases will now be evaluated based on the quality characteristics:

SQL MySQL, PostgreSQL, MariaDB, SQLite

NoSQL MongoDB, Neo4j, Redis, Cassandra

See Table 2 for an overview of the evaluation. The evaluation will be based on a 5-point likert scale, ranging from 'good for this quality characteristic' represented by a '++' to 'bad for this quality characteristic' represented by a '-'. In between are the characters '+' (ok for this quality characteristic), '=' (not good not bad for this quality characteristic) and '∼' (fair for this quality characteristic). Concluding, the 5-point likert scale from good to fair: ++, +, =, ∼, -. Each quality characteristic is ranked by the writer of this thesis. The various decisions are based on numerous papers and internet sources. The documentation of all DBMSs have been observed to gather more information. Furthermore, several discussions took place to discuss the outcome of the rankings together with my companions of this project. The ranking is relative, which means that if a column is omitted, the rankings of the whole may change. Lourenco [Lourenço et al., 2015] greatly helped with the ranking of the Cassandra and MongoDB databases.

	MySQL	PostgreSQL	MariaDB	SQLite	MongoDB	Neo4j	Redis	Cassandra
Reliability	++	++	++	+	++	+	+	+
Consistency	++	++	++	+	-	∼	=	∼
Security	++	+	+	∼	∼	∼	-	∼
Availibility	+	+	+	∼	-	∼	+	++
Performance	=	+	∼	+	++	+	+	∼
Integrity	++	+	+	+	-	+	∼	∼
Scalability	-	-	∼	-	∼	+	++	++
Flexibility	-	∼	-	-	+	++	+	+

Table 2: Evaluation of the various database types by the quality characteristics.

4.2 Conclusion

The quality characteristics are listed in order of importance (top-down). When counting all the scores from Table 2 it appears the best relational database management system is MySQL, according to the rankings. The best NoSQL databases are Cassandra and Neo4j. There are minor differences between them. Cassandra ranks low on consistency and security, which are both important for this project. MySQL ranks low on scalability and flexibility, which are both not massively important. Finally, Neo4j ranks low on security, availability and consistency, these are all quite important. Therefore, seen the results of Cassandra and Neo4j, MySQL has the most '++' on the most important quality characteristics. Thus, making it the most popular and mature database on this list. Moreover, the majority of the data is structured and unstructured data can be converted to structured data, with the help of advanced modern techniques. Also, structured data is best handled by RDBMS. MySQL is AC, according to the CAP theorem, which implies that it lacks partition tolerance. Partition tolerance improves query processing performance, however, seen the size of the data performance should not be a problem. Therefore, regarding its popularity, large community, matureness, structured data, being AC and the quality characteristics it seems that MySQL suits the demands of the Linking University City and Diversity. Concluding, the answer to RQ1 is MySQL, this RDBMS will also be used in the remainder of this thesis.[Harkushko,]

5 Database model

As seen in Section 4.2, MySQL will be used as the DBMS for this project. A basic Unified Modelling Language (UML) diagram will be created in Section 5.3 to form the foundation. To make the UML diagram, the entities and attributes must first be defined in Section 5.1. Consequently, the final Enhanced Entity-Relationship (EER) diagram will be created in Section 5.4, which can be directly implemented in the GUI, MySQL Workbench, of MySQL.

5.1 Entities and attributes

This section will cover the entities and attributes of the database model. These are essential to carefully examine, since these will form the basis of the database model. Consequently, the relations between the entities will be defined and explained. In Section 5.1.1, the entities will be highlighted and in Section 5.1.2, the attributes will be highlighted.

5.1.1 Entities

An entity represent an real-world object, e.g. a dog, cat, house, person. An attribute describes the entity, thus providing additional information to give it context, e.g. name and age for a person (entity). The entities and attributes will be based on all the sources and data available. These are digital sources, converted from archives and books located in the Leiden University Library. The digital sources hold information about students, professors and the Leiden University itself. To give insights into the data sources see Figure 5, for an example.

ID	Nobelp	Achternaam	Voornamen	Roepnaam	Geslacht	geboortedatum	Geboorteplaats	Geboorteland	Sterfdatum	Sterfplaats	Land van c
1		Aalst	Theodoor Pieter Hendrik van		Man	1849-01-15	Den Haag	Nederland	1937-03-09	Den Haag	Nederland
3		Ablaing	Willem Matthias d'		Man	1851-08-29	Batavia	Nederlands-Indië	1889-05-28	Leiden	Nederland
6		Abma	Ernst		Man	1924-07-12	Nijmegen	Nederland	2017-03-25		
7		Abragam	Anatole		Man	1914-12-15	Griva-Semgallen	Rusland	2011-06-08	Parijs	Frankrijk
8		Abraham	Robert Emil		Man	1937-05-13	Wenen	Oostenrijk	2015-11-26		
9		Acquoy	Johannes Gerhardus Rijk		Man	1829-01-03	Amsterdam	Nederland	1896-12-15	Leiden	Nederland
10		Addink	Alberti Daniël François		Man	1935-03-31	Eindhoven	Nederland	2006-04-30	Leiden	Nederland
11		Adelman-Glicman	Irma		Vrouw	1930-03-14	Cernowitz	Roemenië	2017-02-05		
12		Adriaanse	Hendrik Johan		Man	1940-04-11	Winschoten	Nederland	2012-08-28		
15		Alberti	Joannes	Joan	Man	1698-03-06	Assen	Nederland	1762-08-13	Leiden	Nederland
16		Albinus	Bernhardus	Bernard	Man	1653-01-07	Dessau	Duitsland	1721-09-07	Leiden	Nederland
17		Albinus	Bernhardus Siegfried		Man	1697-02-24	Frankfurt (Oder)	Duitsland	1770-09-09	Leiden	Nederland
18		Albinus	Fredericus Bernhardus		Man	1715-06-20	Leiden	Nederland	1778-05-23	Leiden	Nederland
21		Aler	Jean Matthieu Marie		Man	1910-07-13	Amsterdam	Nederland	1992-06-30	Amsterdam	Nederland
23		Alkema	Evert Albert		Man	1939-01-18	Eelde	Nederland			
24		Allamand	Johannes Nicolaus Sebastianus		Man	1713-09-18	Lausanne	Zwitserland	1787-03-02	Leiden	Nederland
25		Allegro	Jacques Tobias		Man	1940-11-28	Utrecht	Nederland			
30		Alphen	Jan van		Man	1900-03-08	Leiden	Nederland	1969-05-31	Voorburg	Nederland
31		Alting von Geusau	Frans Alphons Maria		Man	1933-06-26	Bilthoven	Nederland			
33		Altona	Cornelis	Cees	Man	1931-09-23	Makassar	Nederlands-Indië	2008-12-13	Leiderdorp	Nederland
35		Amesz	Jan		Man	1934-03-11	Gouda	Nederland	2001-01-29	Leiderdorp	Nederland
36		Anbeek van der Meijden	Anthonie Gerrit Hendrik		Man	1944-09-18	Veenendaal	Nederland			
38		Anceaux	Johannes Cornelis		Man	1920-07-04	Schiedam	Nederland	1988-08-06	Leiden	Nederland
40		André de la Porte	Gillis		Man	1866-02-25	Boxtel	Nederland	1950-03-12	Den Haag	Nederland
44		Ankum	Johan Albert		Man	1930-07-23	Amsterdam	Nederland			
45		Antal de Felsögeller	Kornél Vilmos		Man	1917-04-19	Boedapest	Hongarije	2003		
47		Anthonides	Johannes	Jan	Man	1569	Alkmaar	Nederland	Tussen 1635 e	Amsterdam	Nederland

Figure 5: Example of a current data source containing historical data.

As seen in Figure 5, the data is currently stored in Excel files. There are lots of Excel files, all structured in another way, which makes it confusing to understand. The database model will try to simplify this and remove any uncertainties. However, the data sources are useful to classify entities, which will be used in the database model, e.g. $firstname$, $lastname$, $place\ of\ birth$ and $Nobelaward$ can identify that there exists an entity $professor$. To identify all these entities all data sources have been examined and the following entities are defined:

- Student
- Professor
- Mother
- Father
- Child
- Location
- Employee
- Study
- University
- Faculty
- Institute
- Location
- Publication
- Experiment
- Specialization

5.1.2 Attributes

All these entities have attributes. These attributes describe the entity which essentially give them information. For example, a *student* has an attribute $firstname$, the $firstname$ is an attribute of *student*, which describes the entity *student* by giving it information. An entity can have many attributes to describe it. The attributes have been found by studying the available sources. All the attributes of each entity, mentioned above, are shown in Table 3.

Entity	**Student**	**Professor**	**Father**	**Mother**	**Child**
Attributes	First name Last name Suffix Call sign Gender IsEnrolled Date of birth Date of death Religion Nationality Student number	First name Last name Suffix Call sign Gender IsEnrolled Date of birth Date of death Religion Nationality Nobel prize Appointment Discipline Start of appointment End of appointment	First name Last name Suffix Call sign Gender IsEnrolled Date of birth Date of death Religion Nationality	First name Last name Suffix Call sign Gender IsEnrolled Date of birth Date of death Religion Nationality	First name Last name Suffix Call sign Gender IsEnrolled Date of birth Date of death Religion Nationality
Entity	**University**	**Faculty**	**Institute**	**Location**	**Study**
Attributes	Name Date of creation	Name Date of creation	Name Date of creation	Country City Streetname Postalcode House number Addition Location date	Name Language Croho-number
Entity	**Supporting staff**	**Employee**	**Publication**	**Experiment**	**Specialization**
Attributes	Job name Field	Start of employement End of employement	Publication date Name Score	Name Date	Name

Table 3: Displays the attributes of each entity.

As seen in the above table some entities have the same attributes, which is not optimal for clarity. This problem can be mitigated by inheritance, this will be further explained in detail, alongside many other UML Diagram relationships in Section 5.2.2. Concluding, RQ2 is answered in Section 5.1.1 and Section 5.1.2, respectively.

5.2 Diagrams

In the previous Chapter, the entities and attributes of the database model were defined. This section will build on top of this by using these entities and attributes to create a basic model. There are various ways to visualize such a model. The most common ways to visualize a data model are through Entity-Relationship (ER) diagrams or UML diagrams. The latter will be used to create the foundation of the database model in a simplified conceptual manner. The ER diagram offers lots of features, however, MySQL Workbench only supports EER diagrams, which is an extended version of the ER diagram. Therefore, the EER diagram will build on top of the UML diagram to model the database in-depth with all its relations. Consequently, the EER diagram can be directly implemented in the DBMS through MySQL Workbench.

5.2.1 Choosing a UML diagram

UML, short for Unified Modelling Language is a third-generation object-oriented programming language created in the 1990s, based on the notations of the Booch method, the object-modelling technique and object-oriented software engineering. UML models software solutions, database models, business operations and design of systems. UML models are popular and have a large user base, which is one of the reasons for choosing them [OMG, 2017]. There are two main types of UML diagrams, these are structural and behavioral diagrams. Structural diagrams are static and represent all the static aspects of a system that must be present. On the other hand, behavioral diagrams represent the dynamic aspect of a system, these are aspects that must happen, i.e. how do objects interact with each other. This thesis focuses on the structure of a database and not its actions, therefore a structural diagram is chosen.
Structural diagram that is the umbrella diagram for seven child diagrams, see Table 4 for an overview of the structural child diagrams.

Diagram	Description
Class Diagram	Shows the classes in a system with its corresponding attributes and sometimes operations. The relations between each class, if any, are shown through different types of arrows. These different types of arrows are based on multiplicity.
Component Diagram	Focuses on components of a system, abstract version of class diagrams.
Deployment Diagram	Illustrates the hardware of a system, within the hardware it shows the software.
Object Diagram	Commonly referred to as instance diagrams, since it uses real-world examples. Furthermore, essentially the same as class diagrams without the operations and with specific examples instead of abstraction.
Package Diagram	Visualizes all the packages of a system and its dependencies.
Profile Diagram	Extension of UML, used to describe lightweight extension mechanisms by defining stereotypes, tagged values and constraints.
Composite Structure Diagram	This type of diagrams shows the internal structure of a class.

Table 4: Seven structural diagrams explained[Nishadha, 2021].

As seen in Table 4 class diagrams suits the requirements for this project the most. Keep in mind that the database model has attributes, entities (these represent the classes) and relations. Also, class diagrams are the most used UML diagrams according to [Reggio et al., 2013]. Furthermore, they are considered to be a fundamental diagram for software architecture [Reggio et al., 2013]. Therefore, the UML class diagram will be used to visualize the basic database model.

5.2.2 UML definitions

A UML class diagram can show relations in many types, which are essential for understanding and optimizing the database model. The different types of relations are association, aggregation, composition, generalization (read: inheritance) and specialization [OMG, 2017]. The relations all have different purposes. To make a clear overview of all these relations and their definitions examples are added, see Table 5 for the overview.

Relation	Definition	Example
Association	Object has a relation with another object	Student learns from professor and professor teaches student
Aggregation	Child can independently exist without parent	Student is part of a class, but can exist without a class
Composition	Child cannot independently exist without parent	Student is part of a university, but cannot exist without a university
Inheritance	Parent has all the umbrella attributes of its childs	Person is the umbrella for student, teacher, since they both have a first and last name
Specialization	Child has specific attributes compared to the parent	Student has a student number on top of the first and last name

Table 5: Five types of relations in UML.

5.2.2.1 Multiplicity

Another important aspect of UML is multiplicity. The UML defines multiplicity as the range of allowable cardinalities that a set may assume and is commonly used for association ends. Cardinality is the number of elements in a set. The use of multiplicity in a UML diagram clarifies it, by providing additional necessary information. The number or range at the association end (target end) represents the allowable cardinality or cardinalities that may be associated with the source end. In other words, assume a father and a child and let the child be the association end, if the child has a multiplicity of 1 it means that the child can always have exactly one father. The notation is as follows: $n_1, \ldots, n_2$ for a range of numbers and n for a number, where n_1 represents the lower limit and n_2 the upper limit. A special notation is the '*', it represents an unbounded upper limit.[OMG, 2017, Genova et al., 2002]
To illustrate different kinds of multiplicity, see the following enumeration:

0,1 The cardinality has a range of 0 to 1. Note: the notation is different, since it is a contiguous interval [OMG, 2017].

1 The cardinality equals exactly 1.

0..* Equivalent to *, cardinality is unrestricted, it is also known as 'many'.

1..* The cardinality has a range of 1 to many, also known as '1 or more'.

3..23 the cardinality has a range of exactly 3 to 23.

5.3 Design UML diagram

As the entities and attributes among with UML are defined and explained the UML diagram can be created. The relations will be explained in depth after the figure is shown. Due to the size of the diagram the attributes will not be displayed, since these were already mentioned in Table 3. However, the EER diagram will display these See Figure 9 and the full UML diagram can be found in the Appendix, see Figure 8. Lastly, see Figure 6 for the explainable UML diagram. The UML class diagram uses association, aggregation, composition and inheritance and different kinds of multiplicities to define the relations between the entities.

5.3.1 Clusters

Figure 6 is divided into separate clusters, these can be identified by the same color. The clusters are defined by the following colors: blue, green, red and yellow, which represent the university, employee, family and student cluster, respectively. The relations are based on the available data source mentioned in Appendix C. Moreover, the conceptual model tries to sketch a model with common sense that can be used in real life, i.e. a child has a mother. This conceptual model has been reviewed and observed by the companions of this project. Various changes have been made, with the help of their feedback, in the process of optimizing the model. Lastly, certain assumptions have been made by the writer of this thesis. The multiplicities are based on assumptions of the writer.

First of all, the blue cluster, it represents the university structure. This cluster contains the university, faculty, institute, building and location entity. The university entity is the association end of

the cluster, since it is the most generalized entity. Next up, the university can be specialized by creating a relation to faculty. A faculty is a more detailed representation, since there can be multiple faculties but only one university. Therefore, the relation between university and faculty is a 1-to-1 or more relation, meaning that a university can have 1 or more faculties and a faculty must have exactly one university. It must have exactly one university, since it cannot exist without a university and thus it is also a composition. Another specialization leads to the institute entity. The relation between faculty and institute is almost identical to the aforementioned, instead, it is 1-to-many. The relation between institute and building is a 1 or more-to-many association relation. A building can exist without an institute, but an institute cannot without a building. In addition, a building can have multiple institutes and an institute can be located in different buildings. The same applies to the relation faculty and building, this relation exists, because in the case of no institute no location can be retrieved. The last relation is between building and location, this is a 1 or more-to-1 composition relation. A building should always have exactly one location, it cannot exist without it. On the other hand, a location can have multiple buildings (high-rise, industrial area, etcetera).

Second, the green cluster, this represents the employee cluster. The cluster contains the employee, supporting staff, professor, publication and experiment entity. The first three relations are identical, which are inheritance, regarding employee to person, supporting staff to employee and professor to employee. As mentioned in Table 5, when inheriting from your parent entity you inherit all the attributes. For example, supporting staff only has attributes $field$ and $JobName$. It does not need more, since the start and end date of employment are already inherited from employee, the same holds for all person's attributes. This makes it easier to read. Next, the professor has a self-relation because a professor can research with another professor. A professor usually writes and publishes publications, however, this does not always have to be the case, this makes a relation between professor and publication a 1 or more-to-many relation. The relation between publication and experiment is denoted as follows, 1-to-many, a publication does not necessarily need an experiment.

Third, the red cluster, which represents the family structure. The first three relations of this cluster are identical to the aforementioned paragraph. Mother, father and child inherit from person, since they all are persons and thus inherit the attributes. A mother and father can have one or many children, however, a child can only have exactly one mother and father, therefore making it both a 1-to-1 or more composition relation. As per usual a person can have a relationship with another person, which elucidates the self-relation.

At last, the yellow cluster, it regards the student cluster. This cluster has three entities in it: student, study and specialization. The student directly inherits from a person and has a self-relation. A student can work with other students on a project, hence the self-relation. The study has a relation with both entities in this cluster. A study usually has many students, but can also have zero when none enrol and vice versa, thus making it a many-to-many aggregation relation. Sometimes a study has different disciplines within a study a so-called specialization, this specialization inherits from the study and only has an extra attribute, $StudyName$. Certain choices had to be made while clustering and for yellow this meant that this cluster only contains three entities. However, study relates to three more entities in other clusters which are professor, faculty and institute. The study, for example, has professors to teach. Study also has a relation to both faculty and institute, because there must always be a relation to the university. However, if the institute does not exist, it is

not possible and thus the relation to faculty cannot exist without a faculty. However, it can exist without institute or professor. The other way around, a professor does not have to teach and a faculty and institute can have multiple studies. Resulting in a 1 or more-to-many, 1-to-1 or more and many-to-1 or more relation, respectively.

The person entity is colored grey to ensure special attention, since it is a major influential entity being the parent of mother, child, father, employee, student and prestigious people. The latter, falls outside any cluster, since it does not fit in any, hence its normal color. The last entity that is yet to be mentioned is stored data. The stored data entity stores all the unstructured data coming from external sources. This data first needs to be converted to structured data by Natural Language Processing (NLP). However, the unstructured data is already stored so future projects can directly start converting it, without having to search the data. RQ3 has been answered in this section by ultimately creating the UML class diagram (including attributes) in Figure 8.

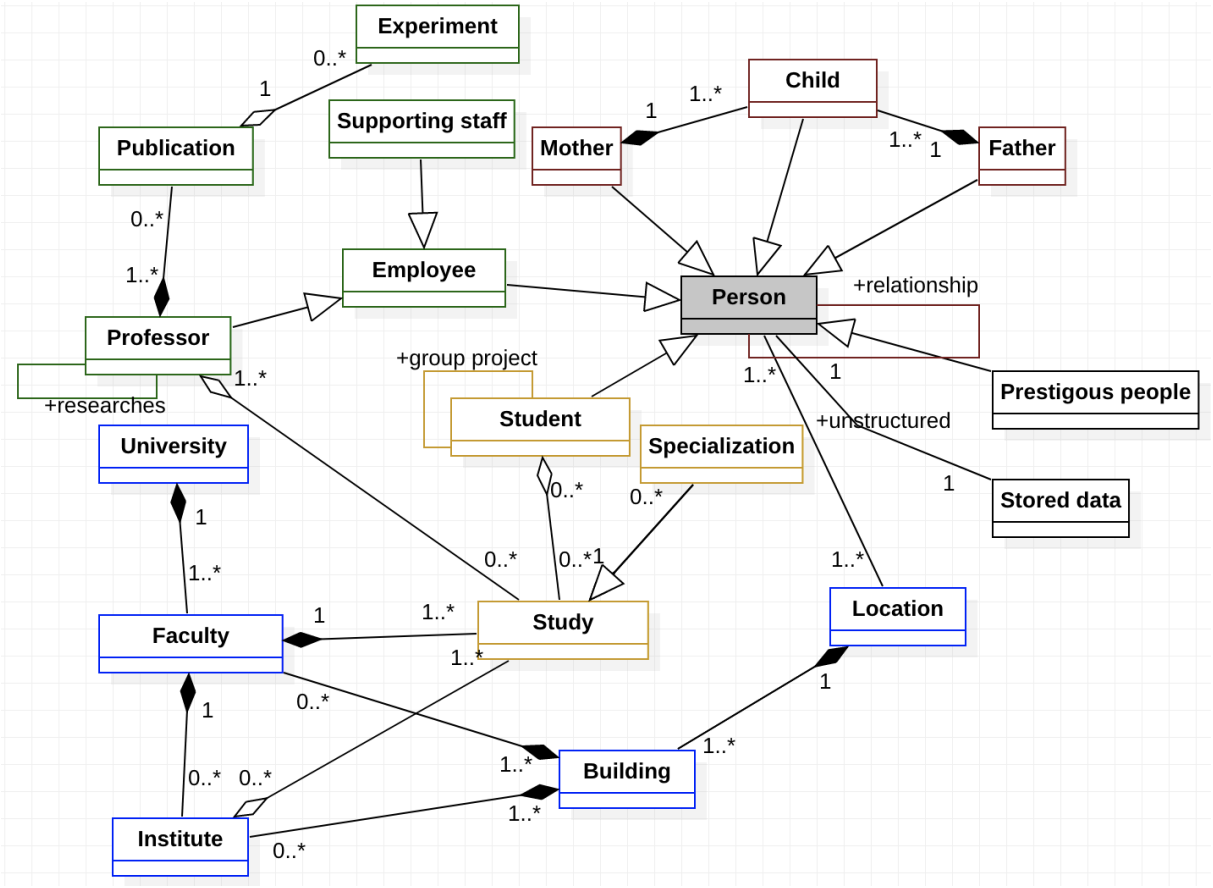

Figure 6: The UML diagram of the database model, including all its relations and entities, for the Linking University City and Diversity project. The figure shows the conceptual database architecture of this thesis.

5.4 EER Diagram

The EER diagram or enhanced entity-relationship diagram is the extended version of the ER diagram. It supports the design of a database through high-level models. The EER diagram was invented in the 1970s by Ramez E. to reflect the data properties and imperatives more accurately. This was desired by upcoming applications of database technology, since they were much more complex [Ramez, 2016]. The benefit is the option to be able to directly transform the EER diagram into tables in MySQL Workbench, the GUI of MySQL, by forward engineering.

5.4.1 MySQL Workbench

As mentioned in Section 4.2 this thesis will use MySQL as the DBMS for the database model. MySQL Workbench will be used as the GUI, since it is developed by Oracle, which is the company behind MySQL, therefore ensuring full compatibility with the latest features. The GUI is easy to use and can be used on Windows, Linux and MacOS. As mentioned before, MySQL Workbench has a built-in feature to transform a model into tables, this is useful.[Corporation, 2022]

5.4.2 The final model

Since the database model is already generally explained in Section 5.2. This section will only cover the challenges during the process of the EER diagram creation. But first, a few definition have to be explained, these are *primary key* and *foreign key*. A primary key is a table column which uniquely identifies each table row. The foreign key is a table column which refers to a primary key in a different table. The database model has a primary key for each entity, but does not have a primary key for junction tables (the table connecting to entity tables, i.e. many-to-many relation).

The following challenges have been encountered during the process of the creation of the database model:

1. Using inheritance in MySQL Workbench: During the creation of the database model, inheritance had to be used as seen in Table 3. However, MySQL Workbench does not have a direct option to inherit from a certain table. This made it difficult to create an inheritance relation. An alternative for inheritance in MySQL Workbench is the use of foreign keys. Foreign keys, usually refer to the primary key in the base table. For example, the Child table inherits from the Person table. To ensure the Child has all the attributes of the Person a foreign key, say Person_PersonID, is linked to the primary key of the Person table, PersonID. This ensures that the Child table has all the attributes of the Person table. This is done the same for every other inheritance problem.

2. Minimalizing the required joins needed for data retrieval: To guarantee readability and reduce complexity the amount of joins required for certain queries is minimalized. The writer of this thesis strongly believes that readability is also an important factor. The database model strives to retrieve every possible combination of data within five joins. Due to possible expansion of the database model the rule might not hold anymore in the future.

3. Linking the location to the person entity: The entity person was defined in Section 5.3.1 and in Table 3 its attributes. Given the attributes, it is noted that the place of birth and death does not consist of a single attribute, it consists of all the attributes located in the entity location. The question raised: How can an attribute in an entity have multiple attributes? Place of birth and death abbreviated as PobID and PodID, respectively, are the attributes that need to represent the location of a person who is born or has died at that location. Since it is not a many-to-many relation it is not possible to solve this problem by making a junction table. Another solution could be to link the PobID and PodID directly to the location entity, however then there would show up an error: "**Error 1215: Cannot add foreign key constraint SQL Statement**". This error occurs, since the data type of Pob and Pob was VARCHAR to allow for both numbers and letters (Bakerstreet 24, etcetera), while the data type of LocationID is INT. According to the documentation of MySQL: "Corresponding columns in the foreign key and the referenced key must have similar data types." Consequently, by analyzing this error and the documentation, the data type of PobID and PodID was changed to INT. This resulted in the attributes PobID and PodID being able to inherit all the attributes of location. However, they first need to be matched in a query to retrieve the data. Figure 7 illustrates this section of the database model, only showing the mentioned entities and relations. The dotted lines represent the relation between the PobID PodID and location. There are two lines, since both PobID and PodID need to inherit from location.

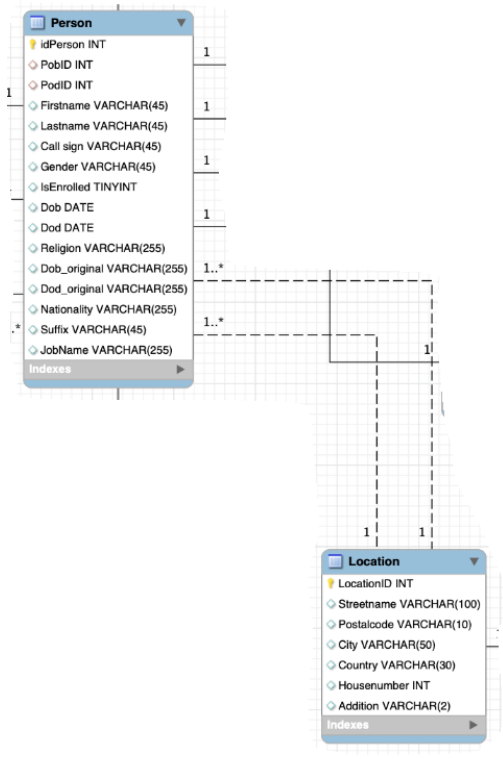

Figure 7: Linking the person entity to the location.

6 Discussion, conclusions and Further Research

The goal of this thesis was to create a database architecture suitable for the Linking University City and Diversity project. To create this database architecture three sub research questions were formulated to gradually develop the architecture. These will all be answered, shortly, down below.

RQ1 was answered based on personal experience with the database, the quality characteristics and the CAP theorem. I think it is very important that a database should be reliable and consistent. For example, executing the same query should always return the correct output. As illustrated in Table 2, MySQL has maximum points on both reliability and consistency, therefore satisfying these conditions. Moreover, SQL is very popular, has a large user base and has a large team constantly developing and testing its systems. Making it a reliable and well-suited database. RQ2, concerns the definition of the entities and attributes, these were defined and found according to my own interpretation of the available data sources C. These are listed in Section 5.1.1 and Table 3. Ultimately, to answer RQ3, with all the gathered knowledge the database model was made and shown in Figure 9.

In Section 1.1 three research questions from historians were formulated. This thesis has endeavored to be able to answer these by creating the database architecture. The first research question is: *Where did students or professors live in Leiden?*. This can be answered by the entities: *Person*, *Professor*, *Student* and *Location*. The relations defined between them ensure that data retrieval of multiple entities is possible. The specific attribute that answers this research question is *Residence*, which receives its information from the entity *location*.
The next research question is: *What is the diversity of people coming in and out of the city through the time?* This is essentially a specific sequel to the previous research question, because it also needs *Location* and *Person*. The *Person* entity defines the diversity by gender, religion and nationality and the *Location* defines the location of a certain person during the time. The most important attribute here is *location date*. This attribute stores the date of location, i.e. the date when someone moved to a new location, and that is exactly what the research question is about.
The last research question is stated as follows: *Who were students' parents and children and whom did they marry?* To answer this question information concerning family structure and relationships is needed. This is modelled in the family cluster and the *Student* entity. As seen in Figure 6 the *Person* entity is able to have a relationship with another *Person*. This type of relationship can be a marriage or simply a friendship. The person's mother or father can be retrieved via the family cluster. The *Mother* and *Father* entities are linked to the *Child* entity to ensure this. For example, Thea is a mother and Diana is a child, but Diana is also a mother and Thea is also a child. This circle ensures that a family structure can always be traced. Therefore, a students' mother and father can be traced including their relationship via the blue cluster and the *Student* entity.

As mentioned in Section 2.2 other papers used triplestores and Neo4j. Echoes a project within Erfgoed Leiden, also uses triplestores. They use the Europeana data model, which has a broad documentation, a large user base and already several historical databases. However, the user interface only shows triples. This limits a user's experience, since they must continue to click through the data to see what they want. A relational database model can show lots of data at the same time by using tables. This can consequently be filtered down by a filter, which can be made by the front end. In several cases, it is better that a user is presented with all the information. If a user wants to

see less he can filter unnecessary information out. On the other hand, Europeana data model can automatically link information to existing information making it easier to model.[Clayphan, 2017]

Concluding, the database architecture consists of relational tables, ensuring simplicity and clearness. Also creating balance between performance, reliability and consistency.

6.1 Limitations

Due to the limited time, not everything could be structured in the architecture. For example, data imported from external sources having other data types must first be converted to the correct data type. The database architecture does not take historical data on enrollment into consideration, for example, which might be desirable by historians. Also, this database architecture intends to provide the LUCD project with a foundation that can be further expanded on. Therefore, the emphasis is more on the accurateness than the full completeness.
Another limitation this thesis has is that it has not explored star or snowflake schemas, where a star schema is a simplified snowflake schema. Star and snowflake schemas can help optimize database models. This could, however, be taken into consideration during the eventual extension. Star and snowflake schemas consist of a fact table and multiple dimension tables. A fact table stores numeric data usually for analysis and dimension tables provide additional information on this quantitative data.[Iqbal et al., 2019]
Lastly, the database architecture has not been properly tested on a server. This was unfortunately not possible during my project, since there was no server to run on. This needs to be explored when uploading the database architecture to a server.

6.2 Further research

This paper did not take HISCO, short for Historical International Standard Classification of Occupations, into account. It is a program that has classified many occupations ranging from "butcher" to "priest" and its geographical scope varies between "German villages" to "Great Britain". It contains all the historical occupations between 1690 and 1970. They build the scheme on top of the existing scheme, ISCO68. This could be useful if further research wants to classify occupation from incoming external sources through the adapter (explained in Section 1.1). That can consequently correctly be added to the database. Further research can also build on top of this model, since it is publicly available.
The Europeana data model can be further explored to define the advantages and disadvantages in comparison to this data model.

References

[doc,] About apache cassandra. https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/cassandraAbout.html.

[red,] Introduction to redis. https://redis.io/docs/about/.

[app,] What is database management systems (dbms)?

[OMG, 2017] (2017). Omg® unified modeling language® (omg uml®). *About the unified modeling language specification version 2.5.1*, pages 1–796.

[db-, 2022a] (2022a). Engines ranking. https://db-engines.com/en/ranking/relational+dbms.

[db-, 2022b] (2022b). Wide column stores - db-engines encyclopedia. https://db-engines.com/en/article/Wide+Column+Stores.

[Abubakar, 2014] Abubakar, Y. (2014). Benchmarking popular open source rdbms: A performance evaluation for it professionals benchmarking popular open source rdbms: A performance evaluation for it professionals. *International Journal of Advanced Computer Technology (IJACT)*, 3:39.

[Ashanin, 2018] Ashanin, N. (2018). Quality attributes in software architecture. https://medium.com/@nvashanin/quality-attributes-in-software-architecture-3844ea482732.

[Baierer et al., 2017] Baierer, K., Dröge, E., Eckert, K., Goldfarb, D., Iwanowa, J., Morbidoni, C., and Ritze, D. (2017). Dm2e: A linked data source of digitised manuscripts for the digital humanities. *Semantic Web*, 8(5):733–745.

[Barry,] Barry, D. K. Javascript object notation (json). https://www.service-architecture.com/articles/web-services/javascript-object-notation-json.html.

[Brewer, 2012] Brewer, E. (2012). Cap twelve years later: How the" rules" have changed. *Computer*, 45(2):23–29.

[Chang and Chua, 2018] Chang, M.-L. E. and Chua, H. N. (2018). Sql and nosql database comparison: From performance perspective in supporting semi-structured data. In *Advances in Information and Communication Networks*, Advances in Intelligent Systems and Computing, pages 294–310. Springer International Publishing, Cham.

[Clayphan, 2017] Clayphan, R., C. V. I. A. (2017). European data model – mapping guidelines v2.4.

[Codd, 1970] Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387.

[Codd, 1971] Codd, E. F. (1971). Further normalization of the data base relational model. *Research Report / RJ / IBM / San Jose, California*, RJ909.

[Corporation, 2022] Corporation, O. (2022). *MySQL Workbench*. MySQL Workbench.

[Genova et al., 2002] Genova, G., Llorens, J., and Martinez, P. (2002). The meaning of multiplicity of n-ary associations in uml. *Software and Systems Modeling*, 1(2):86–97.

[Gupta et al., 2017] Gupta, A., Tyagi, S., Panwar, N., Sachdeva, S., and Saxena, U. (2017). Nosql databases: Critical analysis and comparison. pages 293–299.

[Harkushko,] Harkushko, L. Vital things to consider when choosing a database for your app. https://yalantis.com/blog/how-to-choose-a-database/.

[Iqbal et al., 2019] Iqbal, M. Z., Mustafa, G., Sarwar, N., Wajid, S. H., Nasir, J., and Siddque, S. (2019). A review of star schema and snowflakes schema. In *International Conference on Intelligent Technologies and Applications*, pages 129–140. Springer.

[ISO, 2011] ISO (2011). Iso 25000 portal. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?start=6.

[Jatana et al., 2012] Jatana, N., Puri, S., Ahuja, M., Kathuria, I., and Gosain, D. (2012). A survey and comparison of relational and non-relational database. *International Journal of Engineering Research & Technology*, 1(6):1–5.

[Kamaruzzaman, 2021] Kamaruzzaman, M. (2021). Top 10 databases to use in 2021. https://towardsdatascience.com/top-10-databases-to-use-in-2021-d7e6a85402ba.

[Kraaij and Schmidt, 2021] Kraaij, W. and Schmidt, A. (2021). Linking university, city and diversity. https://www.universiteitleiden.nl/onderzoek/onderzoeksprojecten/wiskunde-en-natuurwetenschappen/liacs-linking-university-city-and-diversity.

[Lacefield, 2018] Lacefield, J. (2018). The evolution of nosql. https://www.datastax.com/blog/evolution-nosql.

[Lo, 2021] Lo, V. (2021). *SQL and NoSQL databases*.

[Lourenço et al., 2015] Lourenço, J. R., Cabral, B., Carreiro, P., Vieira, M., and Bernardino, J. (2015). Choosing the right nosql database for the job: a quality attribute evaluation. *Journal of big data*, 2(1):1–26.

[MariaDB, 2019] MariaDB (2019). Mariadb foundation. https://mariadb.org/.

[Martorelli et al., 2020] Martorelli, I., Helwerda, L. S., Kerkvliet, J., Gomes, S. I. F., Nuytinck, J., van der Werff, C. R. A., Ramackers, G. J., Gultyaev, A. P., Merckx, V. S. F. T., and Verbeek, F. J. (2020). Fungal metabarcoding data integration framework for the mycodiversity database (mddb). *Journal of Integrative Bioinformatics*, 17(1):20190046.

[Meiryani, 2019] Meiryani, A. S. (2019). Database management system - ijstr. https://www.ijstr.org/final-print/june2019/Database-Management-System.pdf.

[Mohamed et al., 2014] Mohamed, M. A., Altrafi, O. G., and Ismail, M. O. (2014). Relational vs. nosql databases: A survey. *International Journal of Computer and Information Technology*, 3(03):598–601.

[Nasarek, 2019] Nasarek, R. (2019). Data modeling of complex historical information. In *RODBH*, pages 15–25.

[Nishadha, 2021] Nishadha (2021). Uml diagram types: Learn about all 14 types of uml diagrams.

[Offutt, 2002] Offutt, J. (2002). Web software applications quality attributes. *Quality Engineering in Software Technology (CONQUEST 2002)*, pages 187–198.

[Ontotext, 2014] Ontotext (2014). The truth about triplestores.

[Ramez, 2016] Ramez, E. (2016). Fundamentals of database systems seventh edition.

[Reggio et al., 2013] Reggio, G., Leotta, M., Ricca, F., and Clerissi, D. (2013). What are the used uml diagrams? a preliminary survey. volume 1078.

[Romanowski, 2020] Romanowski, J. (2020). The most popular databases in 2020. https://learnsql.com/blog/most-popular-sql-databases-2020/.

[Simplilearn, 2021] Simplilearn (2021). A complete guide on mysql workbench. https://www.simplilearn.com/tutorials/mysql-tutorial/mysql-workbench.

[Sint et al., 2009] Sint, R., Schaffert, S., Stroka, S., and Ferstl, R. (2009). Combining unstructured, fully structured and semi-structured information in semantic wikis. In *CEUR Workshop Proceedings*, volume 464, pages 73–87. Heraklion Crete, Greece.

A UML class diagram with attributes

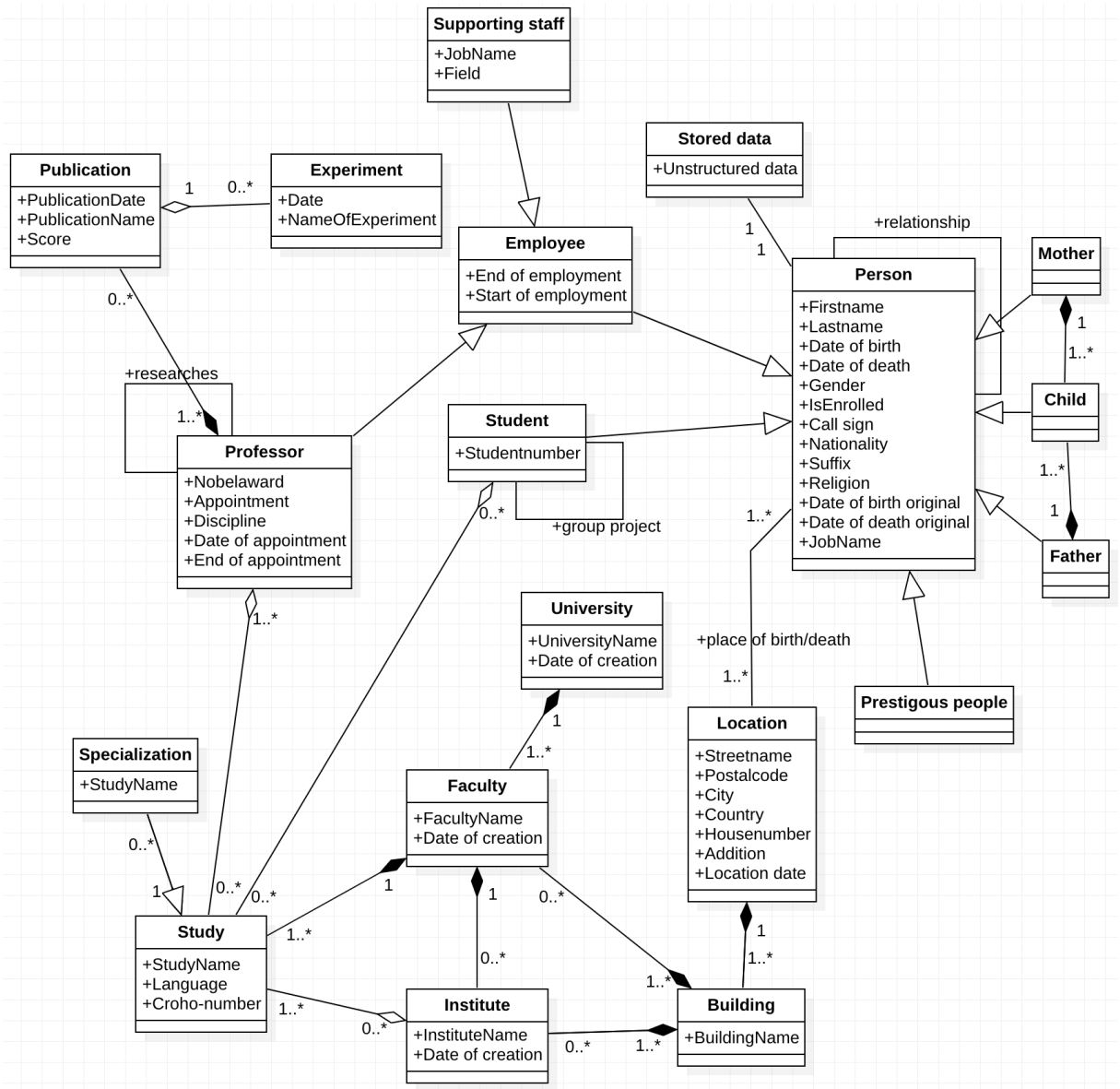

Figure 8: The UML diagram with attributes of the Linking University City and Diversity project.

B Full EER diagram

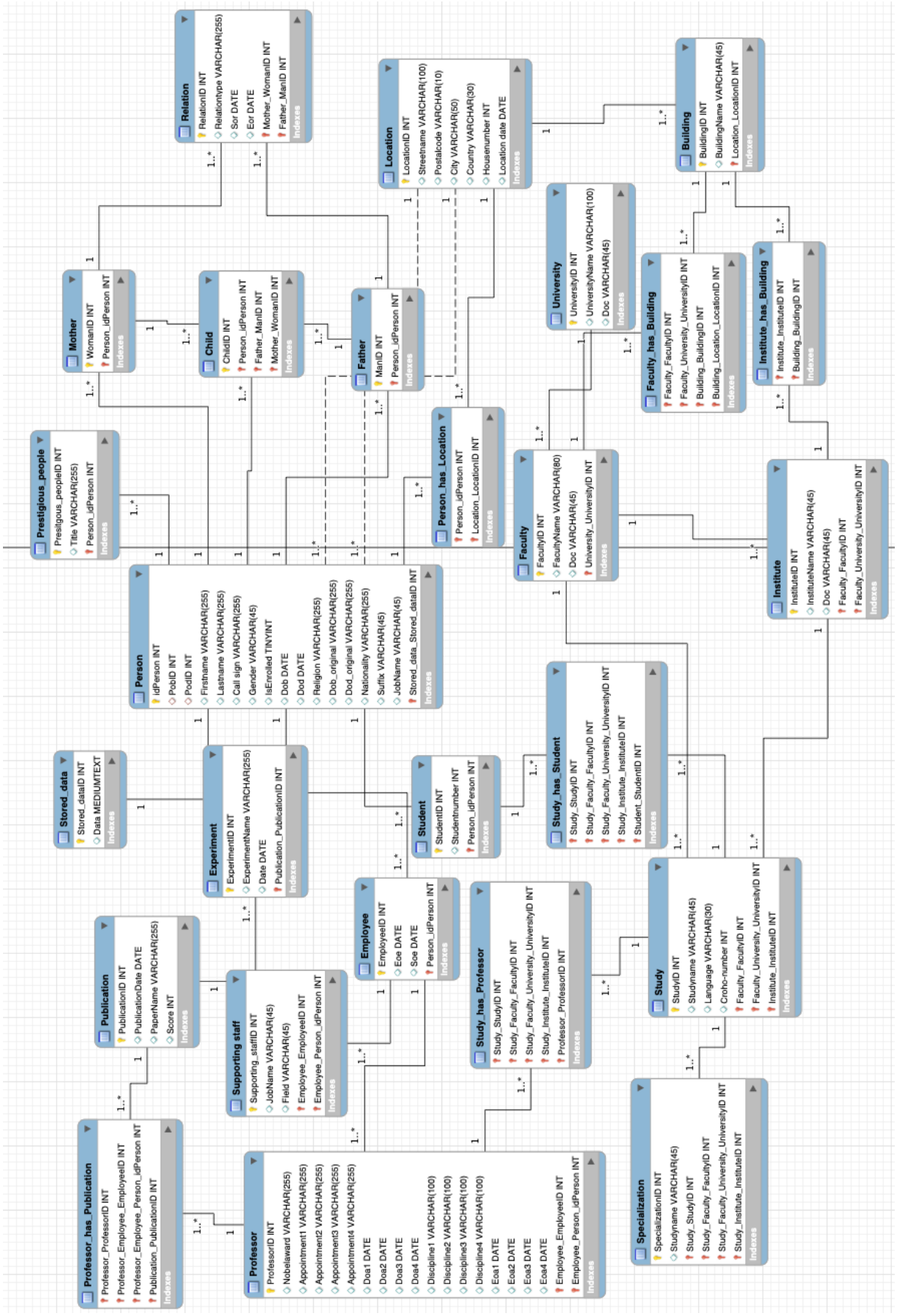

Figure 9: The EER diagram of the database model for the Linking University City and Diversity project.

C Github: Data sources and code

This is the link to my public github repository: "https://github.com/LiacsProjects/linkingUCD_code/tree/Rick". In Github the source code is available for reproduction of the database. There is an adapter, via which one can make a connection with the existing database. Lastly, the data sources used for the design and creation of this database are listed in Github.