Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Bachelor Thesis

Relational Database Design: Case of Jablotron Group

Artem Makurin

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Artem Makurin

Informatics

Thesis title

Relational Database Design: Case of Jablotron Group

Objectives of thesis

Main objective of the thesis is to propose, build and analyse a database for Jablotron Group's first retail store.

To achieve this objective, several tasks must be first accomplished. These tasks include:

- Give a definition to the Relational database model.
- Follow the historical development of the related technology.
- Explore the business applications of the Relational database model.
- Design a Conceptual data model of the proposed Jablotron system.
- Select the most suitable DBMS and build the Logical data model of the future system.
- Produce the Physical data model using previous findings and build the database system according to it.
- Test the solution to assure required performance.

Methodology

The methodology will consist of such scientific methods of research as analysis, inspection of available reading materials and collection of secondary information.

The proposed extent of the thesis

30-40

Keywords

database, relational database design, practical database design, database performance analysis, DBMS analysis, database ERD

Recommended information sources

BAGUI, S. – EARP, R. Database Design Using Entity-Relationship Diagrams, 2011. ISBN 978-1439861769
HERNANDEZ, M. J. Database Design for Mere Mortals: 25th Anniversary Edition, 2020. ISBN 978-0136788041
CHURCHER, C. Beginning Database Design: From Novice to Professional, 2012. ISBN 978-1430242093

Expected date of thesis defence

2022/23 SS – FEM

The Bachelor Thesis Supervisor

Ing. Martin Pelikán, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 27. 2. 2023
Ing. Martin Pelikán, Ph.D.
Head of department

Electronic approval: 13. 3. 2023
doc. Ing. Tomáš Šubrt, Ph.D.
Dean

Prague on 15. 03. 2023

Declaration

I declare that I have worked on my bachelor thesis titled "Relational Database Design: Case of Jablotron Group" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break any copyrights.

In Prague on ___________________________

Acknowledgement

I would like to thank Martin Pelikan, my thesis supervisor, for help and support.

Relational Database Design: Case of Jablotron Group

Abstract

Databases are an essential tool for businesses that need to manage large amounts of data efficiently. Databases are used by businesses to store customer information, financial data, inventory records, and other types of information necessary for their operations. In today's business world, it is essential to have a robust and efficient database system that can handle complex data structures and provide accurate and timely information. Database systems have become an integral part of modern businesses, providing a reliable and secure way to manage data. The focus of this paper is to document the development of a functional database system that has been designed for Jablotron Group's proposed retail store in Prague. The chosen DBMS for the implementation of the database system is MySQL 8.0.36 The paper also mentions the use of triggers and views to achieve the desired behaviour that is necessary for the day-to-day activities of a retail store. The primary objective of this project is to explore the relational database model and its different aspects in real-life solutions. By designing and implementing a functional database system for a retail store, the paper aims to demonstrate the potential of relational database models in handling complex data structures and providing a robust and efficient solution for data management. Overall, the paper can be considered as a case study that explores the application of the relational database model in real-life scenarios. The proposed solution can be a valuable resource for businesses that are looking for efficient and reliable data management solutions for their retail operations.

Keywords: database, relational, design, implementation, data, retail, DBMS, triggers, views

Návrh relační databáze: Případ Jablotron Group

Abstrakt

Databáze jsou základním nástrojem pro podniky, které potřebují efektivně spravovat velké množství dat. Databáze slouží podnikům k ukládání informací o zákaznících, finančních údajích, skladových zásobách a dalších typů informací nezbytných pro jejich činnost. V dnešním světě podnikání je nezbytné mít solidní a efektivní databázový systém, který dokáže zpracovávat složitá data a poskytovat přesné a včasné informace. Databázové systémy se staly nedílnou součástí moderních podniků a poskytují spolehlivý a bezpečný způsob správy dat. Tato práce se zaměřuje na dokumentaci vývoje funkčního databázového systému, který byl navržen pro vybranou maloobchodní prodejnu společnosti Jablotron Group v Praze. Pro implementaci databázového systému byl zvolen DBMS MySQL 8.0.36.

Bakalářská práce zmiňuje použití spouštěčů a pohledů k dosažení požadovaného výsledku, který je nezbytný pro každodenní činnost maloobchodu. Hlavním cílem tohoto projektu je prozkoumat relační databázový model a jeho různé aspekty reálných řešení. Cílem práce je na základě návrhu a implementace funkčního databázového systému pro maloobchodní prodejnu, demonstrovat potenciál relačních databázových modelů při zpracování složitých datových struktur a poskytnutí solidního a efektivního řešení pro správu dat. Celkově lze práci považovat za případovou studii, která zkoumá použití relačního databázového modelu v reálných scénářích. Navrhované řešení může být cenným zdrojem pro podniky, které hledají efektivní a spolehlivé řešení správy dat pro své maloobchodní provozy.

Klíčová slova: Databáze, relační, návrh, implementace, data, maloobchod, DBMS, spouštěče, pohledy

1 Table of Contents

List of images

List of tables

2 Introduction

Throughout history, humans have always needed to somehow store and organise information. It should go without saying that learning how to transfer knowledge not from one person directly to another, but through a different medium is one of humanity's greatest achievements. Be it cave paintings, paper books, or words on a screen, being able to communicate thoughts, facts and ideas is extremely important. Historically, banks and governmental bodies were always expected to keep track of a lot of information related to their clients and citizens respectively, and it has been that way for as long as they've existed. Things that we now know as databases have accompanied humanity for a long time, greatly preceding the computer era.

Judging by the remains of the earliest dated libraries, humans have been implementing some types of information classification systems for thousands of years, easing the burden of librarians. These basic data classification systems, along with the aforementioned bank and government archives were the first examples of humankind's usage of databases. As humanity evolved, so did the need to record and sort large amounts of data. Starting all the way back in 1963 with Charles Bachman's Integrated Data Store (1), computer-aided data management has come a long way to reach today's heights of development. New and even more advanced database models of the present day are changing the way we expect to work with databases.

The retail industry has witnessed a significant shift in the way business is conducted in recent years. The rise of e-commerce, omnichannel retailing, and big data analytics has led to a fundamental change in the way customers shop, as well as how retailers manage their sales and inventory data. As a result, there is a need for retail stores to have a reliable and efficient method of storing and organising their data. A database system can provide a solution for storing, organising, and accessing data in a structured and secure way.

In this diploma paper, we aim to develop a database for a retail store and evaluate its performance and effectiveness. The practical part of the thesis is focusing on the implementation, while the theoretical part is providing necessary context and explaining the logic of database design. The resultant database should check all marks of a successful and correct design.

3 Objectives and Methodology

3.1 Objectives

Jablotron Group is a successful Czech company employing over 500 people, distributing products in 73 countries, and with a turnover of over 134 million €. Its product line includes but is not limited to security and smart home systems. While being a successful business, Jablotron Group doesn't have any retail stores. The way the products are distributed is through retail shops operated by different companies and through official distributors. This thesis' main objective is to propose, build and analyse a theoretical database system for Jablotron Group's first retail store in Prague.

To achieve this objective, several tasks must be first accomplished. These tasks include:

- Give a definition of the Relational database model.
- Explore the business applications of the Relational database model.
- Analyse the requirements and data of a retail store.
- Design a database schema for the retail store using entity-relationship (ER) modelling.
- Select the most suitable DBMS and build the Logical data model of the future system.
- Produce the Physical data model using previous findings and build the database system according to it.
- Test and validate the database using SQL queries and reporting tools.
- Draw conclusions from performed work.
- Propose suggestions for future improvements and extensions.

3.2 Methodology

The research methodology of this project is a combination of qualitative and quantitative research methods. The qualitative research methods include observations which will be used to gather the requirements and data of the retail store. The quantitative research methods consist of experiments which will be used to evaluate the performance and effectiveness of the developed database.

The data collection process will involve observing the systems of rival businesses. The data will be used to develop the ER model and the sample data for the database.

The ER model will be used to design the database schema, which will include the entities, attributes, and relationships of the retail store data. The database schema will be implemented using DBMS of choice, which will involve creating the tables, columns, keys, and constraints. The sample data will be populated into the database using SQL statements and scripts.

The database will be tested and validated using SQL queries and reporting tools, such as MySQL Workbench. The testing will involve checking the correctness and completeness of the data, as well as the efficiency and scalability of the database.

4 Literature Review

4.1 Defining the Database

One of the tasks that this work seeks to accomplish is forming our own solid definition of the database, using existing materials by other authors as a basis. The reasoning for this task is simple, and it's because, as it will become obvious further into this work, there are no two different definitions that would be focusing on the same aspects of the database concept. Some of these upcoming definitions will, naturally, be more tailored to fit the context of the book that we find them in, but that doesn't matter too much within the scope of this work, as we only need them as general guidelines to base our own thoughts on.

4.1.1 Historical overview

According to the ***Oxford English Dictionary***, the word *database* has been first used in printed media to describe the technology all the way back in 1962. Generally, the word database is used to describe information tracking systems found in our everyday lives and in our computers, but that definition is too broad in scope of this thesis, so we'll have to thoroughly narrow it down as we progress. Certainly, databases in a modern meaning of the word have existed for as long as humanity has been storing data, but historically, they've been called different things, like *archives*. One of the reasons researchers, engineers and linguists have not been able to come up with a solid unchangeable definition that everyone agrees on yet is quite simply the fact that the word is so young. It must be taken into consideration that due to the close link between the two, the word *database* is casually used to describe both the database itself and the database management system (hereinafter referred to as DBMS) that is used to manage it. We, however, must not confuse these two terms, as the sheer difference between the two will become quite clear progressing further into this work.

4.1.2 Literature definition analysis

How is the term *database* usually defined? In order to put down a water-tight definition of the database, we must first take a look at some definitions put in place by various entities. Chris J. Dale, a prominent relational database theory researcher and author on many books on the subject, defines the database as *a collection of persistent data that is used by the* *application systems of a given enterprise* in his monumental work that is still widely used by colleges for teaching database design***: An Introduction to Database Systems*** (2). Moreover, Robert Sheldon and Geoff Moes give a similar definition in their book called ***Beginning MySQL*** (3). They write: *a database is a collection of related data organised and classified in a structural format that is defined by metadata*. Moving a bit away from researchers, Merriam-Webster's dictionary describes the database as *a usually large collection of data organised especially for rapid search and retrieval* and the Cambridge dictionary says it's *a large amount of information stored in a computer system in such a way that it can be easily looked at or changed*. According to a Czech web marketing specialist Jan Štráfelda, a database is *a sophisticated software system for data storage and subsequent processing*. Finally, we'll take a look at how database software vendors describe their work process to get a first-hand perspective. According to Oracle, one of the database industry's leaders and a business which was once the third biggest software company in the world by market capitalization and revenue; *a database is an organised collection of structured information, or data, typically stored electronically in a computer system*.

As we can see, each of these definitions focuses on a slightly different aspect of database technology. This is certainly due to the fact that no universal definition of the database exists. In order to put down our own definition we must first analyse, which of these aspects are, in fact, *purposes* of the database.

4.1.3 Purpose of the database

One thing all aforementioned definitions have in common is obvious, and it would serve as the foundation for ours as well. The main purpose of any database is first and foremost to collect and store data. Certainly, there is a massive variety of different information that can be stored, and, as we know, not all databases are computer-based. Remembering that, let's dig deeper into the definitions. The next important thing to recognise is that the database is not just a simple collection of data. Indeed, it's the organised, structured nature of the collection that makes it a database. But is the organised structure of the database its purpose? Hardly, as it's merely a consequence of information being stored in an organised fashion, which a database is. The result of having that organised structure, though, is the ease of retrieving, manipulating, and updating information that is contained within the database, which is definitely a purpose. What the aforementioned definitions fail to address is the extended period of time for which the database serves its purpose. The permanent, or at least continuous nature of data storage is definitely the purpose of the database as well.

4.1.4 Conclusion

So, what did this analysis achieve? We have drawn out three purposes of the database. Let's put them together: the purpose of the database is to store information for a long time in a manner that makes manipulating it easy. Now, there is only one thing that separates us from building that final definition. Only one author that has been referenced before in this work has mentioned this vital part. A database, computer or paper-based, never consists of a single element, on the contrary, it's always a *system* that has many parts interacting with each other in some way. So, our final definition is a database is *a system that continuously stores information in a manner that makes manipulating it easy*.

4.2 Computer Data Storage

Since we have established that a database is a concept that is best characterised by its use in computer-based systems, it would be helpful to explore how computers have started to be used as data processing machines in the first place. In order to describe data storage as fully as possible, we must separate three different ways a computer can read and store data.

4.2.1 Differences in data storage types

The storage of data on contemporary personal computers is a complex and multifaceted process that involves a variety of different types of storage media and technologies. At its core, the structure of computer data storage can be divided into three primary categories: primary, secondary, and offline storage. More generally, data storage can be described as either volatile (destroyed or corrupted upon computer reset) or non-volatile (independent and long-term memory).

4.2.2 Primary storage

Primary storage or Random Access Memory is the only space directly accessible by the Central Processing Unit (hereinafter referred to as CPU), but it is also the only volatile data source of the computer. It is usually much smaller compared to long-term storage options. As a result of a close link between Random Access Memory and CPU, this type of memory is the fastest for read and write operations and is most commonly used for real-time computer calculations. Earliest computers used obsolete technologies for this, such as delay lines, drum memory, Williams (cathode ray vacuum) tubes, and magnetic tape storage. Later, with the emergence of a more advanced magnetic-core memory, it completely overtook the market and became the go-to solution for primary storage on computers for 20 years from about 1955 to 1975 (4). In 1970, however, metal-oxide-semiconductor field effect (or MOS) transistor-based memory started to take over as semiconductor production increased following Intel's development of their dynamic random access memory chip in 1966. In fact, this same metal oxide semiconductor technology is the foundation of the modern commonplace dynamic random-access memory (or DRAM for short).

4.2.3 Secondary storage

Contemporary secondary memory usually is based on magnetic disk storage devices or on some kind of solid-state drive (hereinafter referred to as SSD) technology, mSATA and M.2 formats based on Flash memory are currently leading. Historically, computers didn't always have secondary, non-volatile storage. First computers only had their primary memory and were programmed either by wire switches or by off-line data sources such as punch tapes. Secondary memory only started to appear in commonplace computers in the 1960s, following IBM's development of the first ever commercially available hard disk drive in 1956. The first SSDs came to the computer memory market in 1991. It was a 20 MB drive by SanDisk, intended to be used with an IBM ThinkPad laptop. At the same time as the general capacity of computers' primary storage was growing linearly, the capacity of secondary storage grew exponentially.

4.2.4 Offline storage overview

To work with data, a computer must read it. Nowadays, there is a wide variety of off-line non-volatile data storage mediums, from flash drives and memory cards to optical discs, but what humanity used in the early days of computing was much more primitive. Data storage has started with punch cards, which are pieces of cardboard with holes that are *punched* out by special machines. The holes represent sequences of instructions that can be interpreted by a machine. This invention has been used since 1725, when Basile Bouchon, a French textile worker, came up with an idea to automate his loom to do patterns. He would use punch cards to *program* his loom, but nothing more. No widespread use was seen for this invention, however, and even the 1837 proposal of the idea of the Analytical Engine, by English inventor and mathematician Charles Babbage, did little to change this fact.

4.2.5 First means of offline storage

The first extensive use of punch cards came half a century later, when for the 1890 U.S. Census Herman Hollerith, a statistician and inventor from Buffalo, NY developed Babbage's idea to the point where punch cards' holes meant not only sequences of instructions, but rather saved information that then processed by a computer. Using this, he developed a punch card data processing system that wasn't only used by the US Census Bureau but was in fact so successful that Hollerith went on to form Tabulating Machine Company in 1896, which was later renamed IBM and found immense success on the market. During the following years, the punch cards have grown to be irreplaceable to industries and governmental entities both in America and worldwide.

The first programmable computers have also resorted to the punch card concept as, despite some data storage technologies starting their development before WWII, they were still in their infancy, and the punch card was a reliable, proven solution. Z3, a beloved project of an independent German scientist and inventor Konrad Zuse which was designed in 1938 and completed by 1941, effectively the first working programmable computer in existence, used punch cards made from a different compound as means of entering data into the system. So did the impressive ENIAC and Pilot ACE systems that came after, in 1945.

Even well before WWII, other storage solutions were already evolving, and new ones were starting to appear. Originally developed by a Danish engineer Valdemar Paulsen in 1898 for his first-of-its-kind telegraphone, which was essentially a sound recorder, magnetic wire data storage showed great promise. Austrian engineer and inventor Fritz Pfleumer expanded on Paulsen's idea and patented his own development, recording audio on magnetic tape in 1928 in Germany (5). This would lead to the creation of magnetic tapes, an early storage medium, and subsequently of cassette tapes which are a relatively modern storage format that is familiar to most people.

4.2.6 Floppy drives

In 1971, IBM, and in the following year, Memorex would release their first attempts at a floppy drive and a floppy diskette to the computer storage market. The Memorex model, called Memorex 650, was astonishingly for its time capable of writing the data to the diskette in addition to reading from it. Other suppliers and manufacturers were quick to adopt this new technology to store and load data for key entry, data logging, and for other data-related computer activities.

4.2.7 Optical disks

Optical disks also predate modern time quite a lot, with the earliest recorded work regarding the subject being dated to as early as 1884, when scientists Bell, Bell, and Tainter, have managed, as part of an experiment, to record the word *barometer* on a glass disk using a beam of light (6). In 1958, the optical disc that could store video (an early form of a digital video disc, hereinafter DVD) was invented and subsequently patented by David Paul Gregg, an American engineer. Ten years later, in the Netherlands, Pieter Kramer, a physicist working for Philips, invented a videodisc that can be read by a laser beam. By 1972 Pieter had filed a patent for his invention (7), and gradually, the physical format following his ideas had become dominant in the production of optical discs. Sony and Philips were ahead of the competition at that time, and it was these two companies that developed the first generation of CD discs in the mid-1980s.

4.2.8 Flash memory

Flash memory, an important invention widely used today, has its roots dating back to 1959 when the floating gate transistor was developed by engineers Mohamed M. Atalla and Dawon Kahng. This set the path to the creation of rewritable floating gate memory by Japanese engineer Fujio Masuoka in 1980. Masuoka was researching for Toshiba at the time, which commercially launched the NAND flash memory in 1987. Flash memory is generally one of two memory architectures: NAND and NOR. NAND accesses data sequentially, which increases read times but allows for greater storage density. On the other hand, NOR is characterised by its random-access method which leads to read times but slower data program and erase times much faster.

Table 1 Comparison of NOR and NAND memory types.

Parameter	NOR	NAND
Density	1 Mbit – 1 Gbit	64 Mbit - 16 Gbit
Read initial access	55 ns	10,000 ns
Read sequential access	9 ns	50 ns
Program	0.3 Mbytes/s	2.6 Mbytes/s
Erase	0.2 Mbytes/s	8.2 Mbytes/s
Access Method	Random	Sequential

Source: Cypress Corporation, 2017

In 1988, the first commercially available NOR type flash chip was introduced by Intel Corporation. Due to the nature of NOR architecture, it soon replaced the dated Read Only Memory that was used as early computer firmware storage. The flash chip is also the basis for the USB flash drive. The credit for the invention of the USB drive is disputed between multiple individuals and companies, including M-Systems, IBM, and Trek 2000 International.

So, where and how were databases stored in computers? To answer that question, let’s dive into how the database came to exist.

4.3 Relational Database Model

The need for good data management and storage systems became apparent with the development of mankind. The explosion in the volume of stored data must be attributed to the invention of the transistor, then of the integrated microchip and subsequent computerization of the way companies handle information. As described in the introduction to this paper, the history of database development started in 1963 with Charles Bachman's proposal of the Integrated Data Store.

4.3.1 How databases came to exist

Earliest computer-based databases have followed the *flat file* model, which is a basic consecutive list of entries, uniquely identified by the ID of each record. With the development of newer and more advanced *navigational data* models came parent-child pointers (later often used to point to physical addresses on a disk) between records in the database. The two dominating navigational database models were the hierarchical model developed by IBM and the network model, developed by, and named after CODASYL, the Committee on Data Systems Language (8)

These databases have used sequential loading of data that was recorded on magnetic or paper tapes. From the mid-1960s the emerging availability of direct-access storage in the form of magnetic disks and drums has made the database an encyclopaedic term, as per ***Oxford English Dictionar***y. This new type of database has allowed shared access of users, which drastically improved usability.

4.3.2 Emergence of Relational model

The most important step in database development happened in 1970, when Edgar Codd, who was a computer scientist at IBM at the time, proposed the *relational* database model in his groundbreaking publication titled ***A Relational Model of Data for Large Shared Data Banks***. He worked at the office that was directly involved in the development of a hard disc and learned the shortcomings of the navigational database model first-hand. In Codd's papers, he insisted that applications should search for information in the database by content, instead of following links. The model proposed by Codd has obsoleted the pointers, as now data would have been separated into simple tables, which then would have been linked together only by their respective data fields. Relational model made the database much more scalable and easier to maintain, so much so that many different companies and people have seen the potential of this new architecture way ahead of Codd's parent company, IBM, which was reluctant to adopt this new paradigm due to their heavy investment in their own database management system - the so-called Information Management System, which employed the rapidly facing obsoletion navigational model.

It took three years for IBM to finally start working in the direction Codd wanted. The System R project was IBM's first step in mainstream relational database development. The papers published by the group behind the project have ultimately influenced two scientists at the University of California in Berkeley, Michael Stonebraker and Eugene Wong to start their own relational database project (9). Although INGRES was the first to enter development, other companies like Relational Software (now Oracle Corporation) were quicker to enter the commercial market (10).

4.3.3 Structured Query Language

An original design by IBM for System R, Structured Query Language (hereinafter SQL) was intended to simplify user interaction with the database by using plain English language where applicable to modify and access database information. SQL is a standard language for managing relational databases, which provides a set of commands for creating, modifying, querying, and deleting data. Structured Query Language is used to create database objects, such as tables, columns, keys, and constraints, as well as to insert, update, and delete data. SQL is also used to retrieve data from the database using queries, which can join, filter, and aggregate the data based on specific criteria. It is a fundamental tool for managing and manipulating data inside of a database system.

Later, when Relational Software was developing its Oracle Database Version 2 (released in 1979), they employed an updated version of SQL to serve as a primary protocol of communication between users and the database management system. So did many vendors that started offering their own database software later in the future, such as open-source PostgreSQL, which is based on INGRES ideas and developed by Michael Stonebraker. ISO and ANSI standard groups would go on to standardise the *SQL Database Language definition* in 1986. With various additions and updates, SQL is still largely used today.

4.3.4 Further development

Naturally, the development of the relational model continued. In 1989, Microsoft partnered with Sybase and Ashton-Tate to release Microsoft SQL Server for OS/2, effectively being the first largely successful effort in bringing database software to that platform (11). By 1995, David Axmark, Michael Widenius, and Allan Larsson released MySQL, an open-source relational database management software, which would go on to quickly gain dominating popularity and rival Oracle as the world's most popular database engine due to its ease of deployment, scalability, and reconfigurability. It is recognized as a go-to solution for most small starting businesses.

Contemporary database software can be divided into many types, but only a handful of radically new approaches have emerged since then. With the development of object-oriented programming, programmers began to interpret the data stored in databases as objects for unity causing the emergence of object-oriented databases. In recent years, the organisation of data in the database has been slowly losing its importance, as businesses incline towards blazing-fast, modern, and horizontally scalable NoSQL databases. These are characterised by not requiring fixed schemas and query statements as a cause of storing denormalized data. Bridging the gap between SQL and NoSQL databases, NewSQL architecture attempts to produce features of both in one package.

4.4 How and why businesses use databases

4.4.1 Origins of database use

As the popularity of the relational database grew, its use began to shift from mainly government bodies and research centres to more mainstream users like businesses of all sizes. Back when computers were mainly used for computational tasks of businesses, all information that was used to get a result of a specific calculation but had no use on its own was considered useless and discarded after the calculation was completed. But as computer storage availability and client expectations of service quality increased, companies have realised that they can somehow store and use all this massive amount of data that daily business activities generate. Current and historical data that the business produces as a by-product of day-to-day operations can be useful for the business in many ways.

4.4.2 Databases as the core of business

How are databases useful for businesses? Database management systems are specifically made to accurately store large amounts of data, with scalability in mind, which is greatly beneficial for the enterprise market with its never-ending expansion (12). Database is responsible for keeping accurate and always updated transaction info so a business can see which client has paid and which hasn't. Sales transactions, inventory changes, manufacturing schedules and billing reports are managed with databases.

Sales and inventory management is a critical aspect of retail operations, which involves the tracking and analysis of sales and inventory data. Sales data includes information on the products sold, the customers who bought them, and the time and location of the transactions. Inventory data includes information on the quantities, locations, and statuses of the products in the store. Effective sales and inventory management can help retail stores optimise their supply chain, reduce waste and loss, and improve customer satisfaction. A database system can provide a solution for storing, analysing, and utilising the sales and inventory data.

The sorted and centralised structure of the database transforms indifferent data into knowledge while enhancing data consistency and quality. Information contained in the database is much easier to obtain quickly when an employee needs it, and the information is guaranteed to be accurate, with less room for human error due to implemented constraints for data. Managing personnel has analysis tools to work with data captured by the database. Billing, payroll, and scheduling reports can be formed this way and analysed by managers.

While businesses are under legal obligations to treat customers' personal data carefully, it's still possible to freely record speculated information like customers' age range and income, buying preferences and interests. This information would then help to customise the buying experience for customers: when you know what a specific client likes to buy, you're more likely to predict what they'll want to buy next. Judging by the information in the database, the sales department can provide special offers to clients who buy most frequently or try to win back clients who haven't purchased recently. The customer database has a monetary value which is inherited from its ability to produce sales.

For some businesses like law and accounting firms an updated knowledge database is important. It is an internal knowledge management system compiled and maintained by experts in the field which can always be consulted in case of questions arising from the normal day-to-day operation of the business. This type of database is usually not accessible by normal customers of the company as the information it contains is priceless for business activities and is very valuable to competitors.

Inventory databases provide much useful information that is much broader than just the amount of goods in stock. Customers can make great use of the database management system's data search and sort features with virtually no performance hit for the company that owns the database. Inventory database access doesn't have to be limited to employees only: suppliers can monitor the stock to know when to make a restock offer to the business, while customers can look at the status, content, and history of their orders. Keeping the database of a company updated saves a tremendous amount of time for the employees who are responsible for the website – they benefit from not having to tediously update web content by hand and being able to simply use the data from the database.

4.5 Database technologies overview

Modern database management systems generally adopt one of a few database paradigms that vary from an architectural point of view. As previously mentioned in this work, *relational* database model is a rather old invention. So, naturally, there exists a constant effort to design new and improved models that don't necessarily follow old conventions, which makes classifying database somewhat more difficult. Some of the different paradigms include but are not limited to:

- *Hierarchical Database Paradigm:* Has data organised in a tree-like structure, where one parent record can have multiple child records. It is often used in mainframe applications and is useful for certain types of data with fixed and predictable structure.

- *Network Database Paradigm:* Similar to hierarchical model but allows for more complex relationships between records. Data is stored in sets, with each set having a parent and multiple children. It is useful for building complex data relationships but is difficult to support and maintain.
- *Object-Oriented Database Paradigm:* Data is stored in objects, following how object-oriented programming languages work. Objects contain both data and methods, allowing a wide variety of data types and encapsulation. It is useful for modelling real world but can be more difficult to query and maintain than other databases.
- *Document Database Paradigm:* Stores data in document formats such as JSON or XML instead of tables. Each document can have its own schema, allowing for more flexibility in data storage. Often used for storing unstructured data, such as social media posts or sensor readings.
- *Key-Value Database Paradigm:* Data is being stored in form of key-value pairs, where each value is associated with a unique key. It is a simple and fast database paradigm, often used for caching and real-time applications.
- *Graph Database Paradigm:* Characterised by storing data in nodes and edges, similar to how graphs are represented in mathematics. Each node represents an entity, while edges represent relationships between entities. Useful for handling very complex data relationships and graph-based queries, such as social network analysis or recommendation systems.

As previously mentioned, not every existing database management system can be characterised by one of these paradigms. The two main directions that are currently seeing most of development are NoSQL and Hybrid model databases, both of which are blanket terms covering a number of various implementations.

- *NoSQL Databases*: NoSQL, or "not only SQL" databases are a broad category of databases that do not use the traditional table-based relational model. Instead, they use a variety of data models, such as document-based, key-value, graph-based, or column-family databases. These databases are often used for large-scale web applications, where performance and scalability are critical. They can be useful for handling unstructured or semi-structured data, such as social media feeds, sensor data, or log files. Some popular NoSQL databases include MongoDB, Cassandra, and Redis.

- *Hybrid Model Databases:* Hybrid model databases combine elements of different database paradigms, such as relational, object-oriented, and NoSQL databases, to create a more flexible and scalable database solution. For example, a hybrid database might use a document-based model for unstructured data, a relational model for structured data, and a graph-based model for relationship data. These databases are often used for complex data structures, such as social networks or recommendation systems, where multiple data models are needed. Some examples of hybrid databases include Apache CouchDB and ArangoDB.

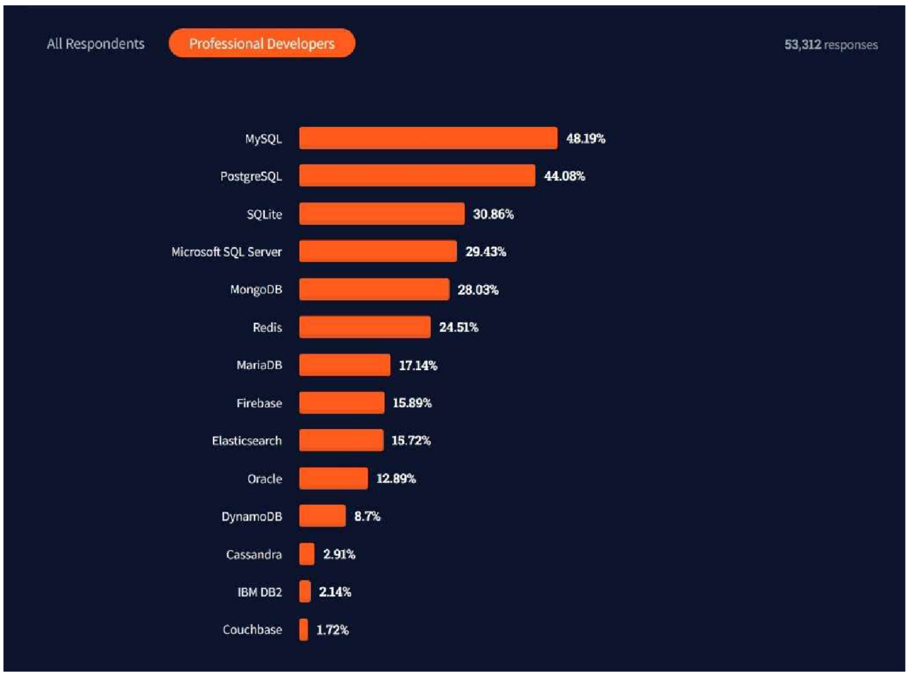
4.5.1 Comparing SQL and NoSQL databases

While relational databases were a definite improvement over older and more basic data models, businesses of today have a choice of either sticking with a relational or a newer non-relational database, also known as a NoSQL database. These two architectures were designed at different times and for different things, and naturally, there is a distinction between the two in the level of support. SQL is an older and standardised technology, thus there is a supply of experienced developers and great documentation regarding it, while NoSQL database management systems are oftentimes dependent on community support due to a lack of proper documentation arising from the much shorter history of the NoSQL concept. The SQL database management software can have various levels of compliance with the SQL standard, but the syntax of general commands is usually pretty similar throughout. On the other hand, having experience in one of the NoSQL carries no value when the user is presented with another NoSQL system, as there is no standardisation between different NoSQL-based database systems. NoSQL's development has taken place due to the restrictive nature of relational model's constraints and structure, which is a pro and con on its own.

As expected, the strict table definition language which is SQL provides data redundancy, rule enforcement and security at the cost of ease of scalability and convenience of storing large amounts of data. At the same time, NoSQL systems offer various sub-models to tailor to the specific use of the database. NoSQL provides the most benefits when used by a business for which the speed of loading data and the number of concurrent users is most important. NoSQL can also prove useful for rapidly expanding companies, as its denormalized and non-standardised structure allows the database servers to be upgraded horizontally, by adding more servers to the pool, while relational databases can only be easily scaled by adding more performance power to a single main server, hence called a vertical upgrade.

So, choosing a SQL or NoSQL database for businesses comes down to their use case, whether data integrity of normalised tables that are minimising data repetition is needed or the benefits NoSQL provides, such as fast access for non-normalised information or scalability (13). Database management systems based on both ideas are still being developed and maintained, with many concepts and features usually reserved for one or the other making their way to their counterparts. Many NoSQL database management systems, such as MongoDB support regular SQL functions, like joins and other commands, and the trend of systems different by nature incorporating features they haven't originally had is expected to continue (14).

Image 1 A chart comparing usage of various database management systems.



Source: Stack Overflow, 2022

According to Stack Overflow's (a question-answer type website for developers with more than 14 million active users worldwide) 2022 developer survey, relational databases are still being used the most, with 4 first places on the survey being SQL based systems. However, the gap between relational and non-relational models is expected to shrink with the continuous development of more complex NoSQL systems in the future. However, as both SQL and NoSQL databases have their respective use cases, the relational model is not going to go extinct just yet.

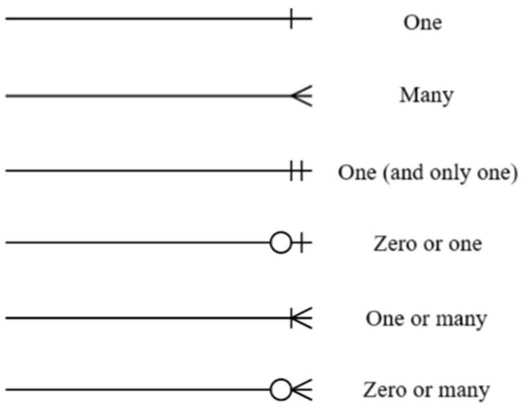
4.6 Database design theory

4.6.1 Beginning the design

What does the design of the database system begin with? Different database specialists and book authors have differing opinions regarding that. Some prefer to start their design process by thoroughly analysing the requirements, some are inclined to believe that the first step should be defining a mission statement and objectives of the database. Others say that the first thing to do is to build an ERD: an entity relationship diagram of the proposed design (15). Entity-relationship (or ER) modelling is a technique for designing the conceptual schema of a database, which describes the entities, attributes, and relationships in the domain of interest. An entity is a thing or object that exists independently, such as a product, customer, or sale. An attribute is a characteristic or property of an entity, such as the price, name, or date. A relationship is an association between two entities, such as a product that is bought by a customer, or an order that includes multiple products. In database design, it’s important to keep track of the number of relationships that will exist between entities, as correct constraints need to be set up. For example, a user can be registered in the system and not have any orders, but an existing order is always associated with one and only one user. This is a mutual relationship and should be properly recorded in an ERD. For this purpose, a number of different relationship notations exist. Entity relationship diagrams built for this project follow *Crow’s Foot Notation.*

In this notation, a circle means zero, three diverging lines mean zero, and a line represents one. Combining these symbols, complex relationships can be described as per image 2.

Image 2 Overview of Crow's Foot Notation.



Source: Author, 2023

Generally, the ER model is used to create a graphical representation of the database, which helps to visualise the structure and relationships of the future system.

4.6.2 Further analysis

While all mentioned early steps of database design are indeed important, they're not mutually exclusive to one another and a good database design should go through all of them. Requirement analysis is a technique which includes going through all the things the future database should and should not be like. For this task, specifying the details and requirements of the database as thoroughly as possible is crucial (16). To define what the mission statement of the database is means to determine its true purpose, whether it's to solve a particular business problem or to manage the organisation's day-to-day activities. On the contrary, defining the mission objectives is performed by compiling statements representing the general tasks users might accomplish using the database. An *entity relationship diagram* of the proposed database is a type of flowchart that describes the inner structure of the future system, the entities, attributes, and their relationships. The ERD can be of three types, depending on the level of generalisation, who are the diagrams prepared by and who is going to use them. When the future system is entering development, *conceptual* ERD is used to compile together information gathered from analysing business requirements. This model is furthest from the actual physical implementation of the database and is aimed to mostly describe what business needs from the database as opposed to describing fine details of the future system. Moving forward with the design process, *logical* ERD is built by expanding the *conceptual* model and putting down column types, as well as some logical optimisation of tables and columns. Finally, a *physical* ERD is constructed. It goes into more detail about the actual implementation of the future database in a specific DBMS, including finalising columns' & rows' names and data types, adding join tables if necessary, selecting primary keys and appropriate constraints.

The existing database must be analysed, if applicable, to perform a successful database design process. It might be a legacy database or a paper-based database. Still, the way the organisation currently uses and manages its data can give the database designer a lot of information vital for the development process. A different vital step in the design process is to conduct interviews with future platform users from all the departments that would have access to the database to find out more about how they enter and manage data and what the project requires in their eyes. From their answers, the database designer complies a list of required attributes, compares it with their own list of assumed required attributes and presents the following result to users and management. Upon presentation, comments and suggestions may arise, and it's up to the designer to acknowledge these things and modify the structure accordingly if deemed necessary.

4.6.3 Database implementation

The next step in the design process is to begin creating table structures according to the ERD and set their respective fields according to the prior database and user analysis. This separate process starts with identifying subjects of the future database and establishing them as tables in the database, Subsequently, each table is populated with fields. The exact number and type of fields in each table come from prior discussions and distinct characteristics of the table subject. A particular field or fields that uniquely identify each record in the table are designated the primary key of the table.

After tables are created, their relationships need to be identified and properly established. This is achieved via a logical connection using primary and foreign keys or through the use of linking tables. The physical implementation of relationships between entities depends on the type of relationship a designer needs to implement. Sometimes the relationship is obvious, but sometimes thorough interviews with database users are required in order to correctly identify it.

Subsequently, business rules are implemented into the database. These are a set of various nuances regarding the actual database use, such as uniqueness or non-nullability of certain fields and other limitations. For the business rules to be accurate and up to-date, frequent user and management consultation is necessary, as the database designer might have no understanding of the way various areas of the company function. The managers' knowledge needs to be transformed into a set of general business rules, while users will reveal area-specific rules that need to be implemented. It's up to the database designer to make these business rule into real field constraints in the database. Ultimately, it's these constraints and validation tables with fixed values that are maintaining the data integrity of the database.

After the constraints are established, time comes to determine views. A view is a collection of specific database entities selected by a query that is stored in the database's dictionary. It is dynamically updated with relevant entries each time access to a view is requested. Certainly, users from different parts of the company need different information. And again, most commonly the database designer simply doesn't know specifically what each user needs, which leads to interviews having to be conducted. Some users require summary information to enable them to make strategic decisions, while others need detailed data. When this survey is concluded, the developer implements the required database information selections via queries and records them for constant use as views.

Finally, a review is concluded. At this step assuring that tables, fields, relationships, and business rules meet the criteria of good design is important. Every found error that arises must be fixed before the logical structure design is considered over.

5 Practical Part

The goal of this thesis is to go through all the steps of creating a database system, including triggers and views that would be necessary for daily operation of a real business, in this case a typical retail store for Jablotron Group. The resulting system should be instantly recognisable as belonging to Jablotron brand and should follow all principles of correct database design.

5.1 Overview

Jablotron Group a. s., a Jablonec-nad-Nisou based company sells complicated manufactured goods and intellectual goods in the form of on-line services across many European countries. The company is successful in its field without operating any retail stores, due to its broad reseller network. Thus, we assume that the proposed retail store is not going to be very big as there is no great immediate demand for Jablotron products. In this case, there is no need for keeping track of employee records in the database that this thesis seeks to design. The system starts development as a primarily inventory-focused database, with more functionality that is deemed necessary at a workplace added further throughout the development cycle. For these reasons and for reasons discussed in the literature review, MySQL 8.0.36 was selected as a DBMS of choice for this project.

5.2 Pre-design analysis

Expanding from the overview, a list of necessary entities must be drawn in order to propose an inventory database with sale transaction recording and support for online orders. Jablotron sells alarms, cameras, smart home systems, vehicle security and monitoring products, cooling and heating units, phones, and baby respiratory monitors, but the list of required items does not stop there. Since the retail store has to sell items ordered online as well as from the physical shelfs, order tables, customer, and payment info tables, as well as shopping carts must be implemented. Moreover, database tables will be designed according to what a typical business would need to store about their products and clients.

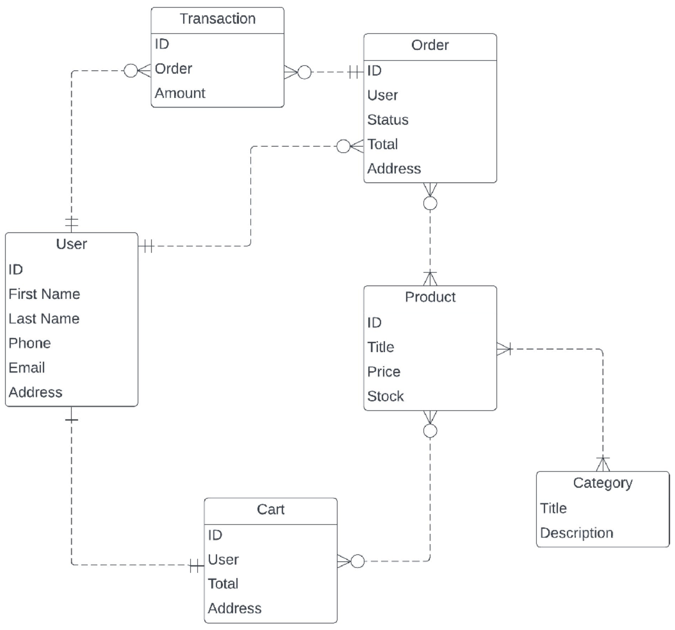
The mission statement of the system we are designing would be to ensure productive day-to-day operations of a retail store using the database. Objectives include recording sale transactions, saving user-specific information to help the business and giving customers a way to interact with the store online.

5.3 Entity relation diagrams

When creating a database, the first step is to analyse the requirements and identify what data needs to be stored and how it will be used. This process involves identifying all the relevant entities and their attributes and determining the relationships between them.

The *conceptual* model is the first ERD that needs to be created. This diagram focuses on the high-level representation of the system and identifies the main entities, their attributes, and the relationships between them. In this model, the emphasis is on understanding the business needs of the system, not the technical implementation details.

Image 3 Conceptual ERD.

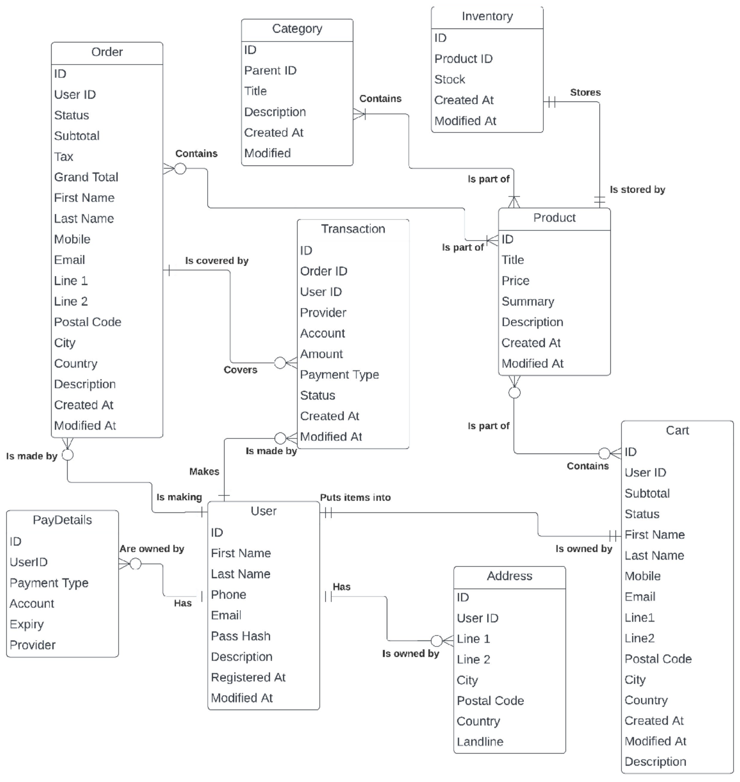


Source: Author, 2024

The six main entities in the conceptual model are *Product*, *Category*, *Order*, *User*, *Cart*, and *Transaction*. These entities represent the main objects that the system will manage, and each entity has its own attributes that define the characteristics of the object. For example, the Product entity might have attributes such as *Product ID*, *Name, Description*, and *Price* at this time. In order to capture future relationships between entities, the diagram employs Crow's Foot Notation which was explained in article 4.6.1 of this work. We are establishing that a user can be responsible for zero or many orders, but each existing order is associated with one and only one user. One user can have one and only one cart while a single cart is owned by one and only one user. User can have made zero or many transactions, but each existing transaction is associated with one and only one user. Order is paid by zero or many transactions, but a single transaction can only be associated with a single order. Finally, at current, *conceptual* level we allow a special type of relationship – many-to-many, which is the case for orders - products, categories – products and carts - products. Naturally, we want many products to be in many orders, but implementing this exact scenario in the DBMS without being careful will cause problems.

Once the Conceptual Model is complete, the next step is to transform it into a Logical Model. This step involves refining the entity relationships and defining all the attributes for each table. In the Logical Model, each table has its own set of attributes that define the data that is stored in it. This includes not only the attributes defined in the Conceptual Model but also any additional attributes that are added to meet the business needs of the system.

Image 4 Logical ERD.

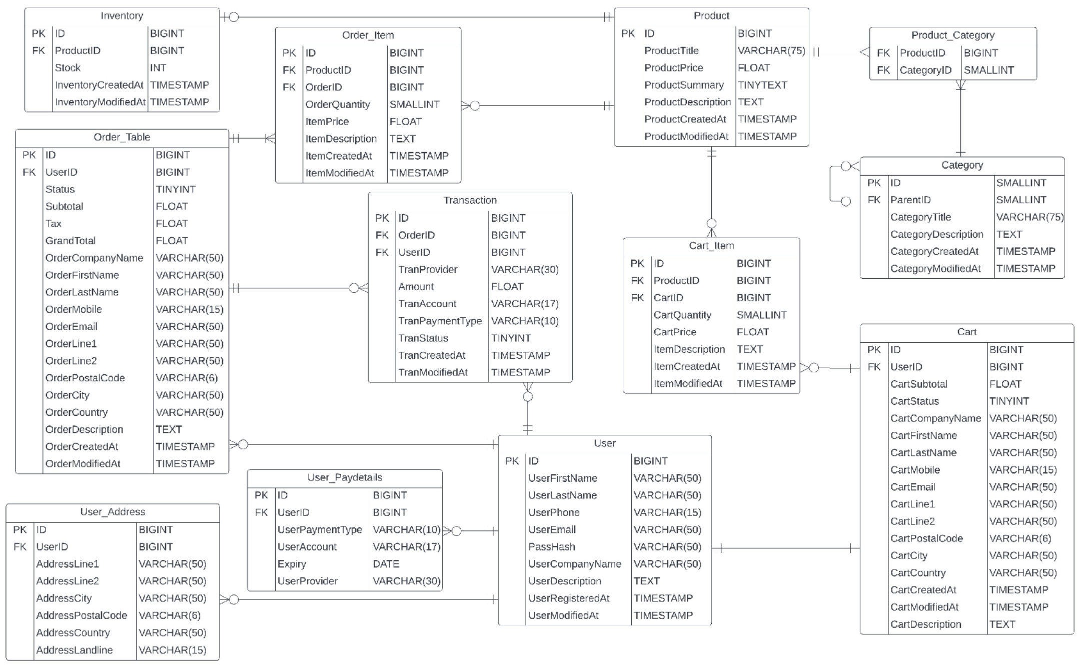


Source: Author, 2024

It is important to note that the *logical model* is still independent of any specific DBMS. Entities are becoming more defined, with some attributes being expanded into multiple. Some additional entities have been added as a result of this step – *PayDetails*, *Address* and *Inventory*. Many-to-many relationships are still allowed, but the logic of many-to-many relationships and other relationships between entities needs to be established and recorded. For example, user can have none or many addresses, but each address is owned by only one user. On the contrary, only one Inventory record is associated with each Product record.

The final stage, *physical* ERD, will leave us with a ready for implementation database system overview with details of specific DBMS, such as MySQL, in mind. In this stage, the limitations of the specific DBMS are taken into consideration, any optimizations that could be needed are performed and exact attribute types are selected. For example, *Order* is a reserved keyword in MySQL and the table must be renamed. The process also includes setting up primary and foreign keys.

Image 5 Physical ERD.



Source: Author, 2024

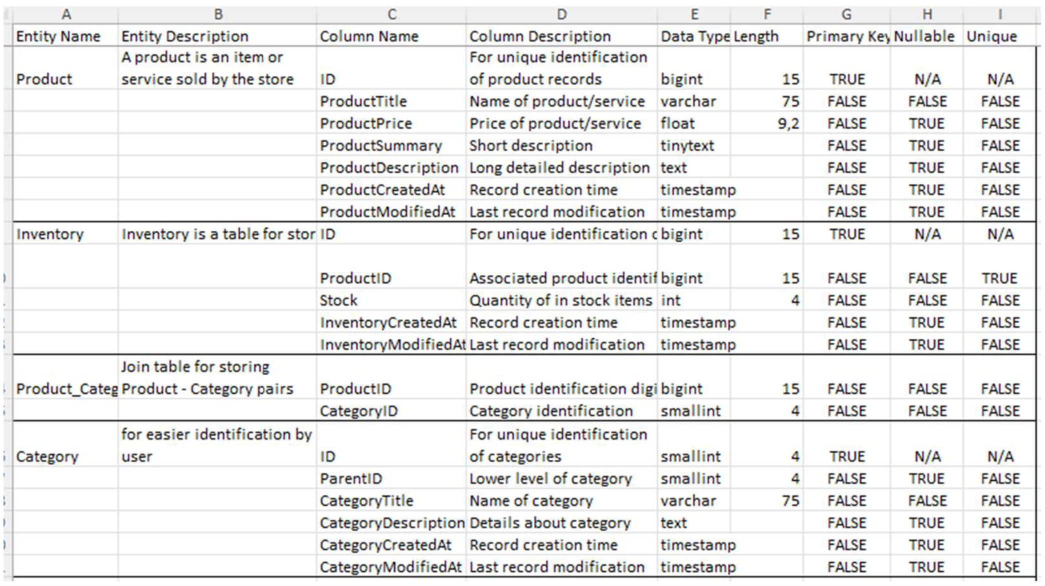
In the end, all many-to-many relationships were replaced by one-to-many with help of association tables. Due to the fact that some items stored in the *Products* table will be services, there is a possibility that no *Inventory* records with specific *ProductID* will exist in the database. *Product_Category* table will use a combination of two foreign keys for unique record identification. Each *Category* table record contains an attribute storing the *ParentID* of this record. As there are multiple category levels, categories that are root-level will have *ParentID* of *null*. Each category can only have zero or one parent, but each category can have zero or many children. By following this systematic and detailed approach, we have created a database that is efficient, flexible, and able to meet the needs of the organisation or project.

5.4 Data Dictionary

A data dictionary is a centralised repository of metadata that contains information about the data elements in a database. It typically includes a collection of entities, attributes, and their properties, with detailed explanations about their role and purpose. The entities refer to the objects or concepts in a database, while the attributes refer to the characteristics or properties of those objects. The data dictionary also contains information about the relationships between the entities, such as the cardinality and participation constraints.

The purpose of a data dictionary is to provide a clear and concise definition of the data elements in a database, ensuring consistency and accuracy in data usage. It serves as a reference for database designers, developers, and end-users, enabling them to understand the structure and meaning of the data. The data dictionary also facilitates communication between different stakeholders involved in the database development and maintenance process. Additionally, it can be used to generate reports, documentation, and data dictionaries for various applications and systems that use the database.

Image 6 Data dictionary definitions.



A	B	C	D	E	F	G	H	I
Entity Name	Entity Description	Column Name	Column Description	Data Type	Length	Primary Key	Nullable	Unique
Product	A product is an item or service sold by the store	ID	For unique identification of product records	bigint	15	TRUE	N/A	N/A
		ProductTitle	Name of product/service	varchar	75	FALSE	FALSE	FALSE
		ProductPrice	Price of product/service	float	9,2	FALSE	TRUE	FALSE
		ProductSummary	Short description	tinytext		FALSE	TRUE	FALSE
		ProductDescription	Long detailed description	text		FALSE	TRUE	FALSE
		ProductCreatedAt	Record creation time	timestamp		FALSE	TRUE	FALSE
		ProductModifiedAt	Last record modification	timestamp		FALSE	TRUE	FALSE
Inventory	Inventory is a table for stor	ID	For unique identification c	bigint	15	TRUE	N/A	N/A
		ProductID	Associated product identif	bigint	15	FALSE	FALSE	TRUE
		Stock	Quantity of in stock items	int	4	FALSE	FALSE	FALSE
		InventoryCreatedAt	Record creation time	timestamp		FALSE	TRUE	FALSE
		InventoryModifiedAt	Last record modification	timestamp		FALSE	TRUE	FALSE
Product_Categ	Join table for storing Product - Category pairs	ProductID	Product identification digi	bigint	15	FALSE	FALSE	FALSE
		CategoryID	Category identification	smallint	4	FALSE	FALSE	FALSE
Category	for easier identification by user	ID	For unique identification of categories	smallint	4	TRUE	N/A	N/A
		ParentID	Lower level of category	smallint	4	FALSE	TRUE	FALSE
		CategoryTitle	Name of category	varchar	75	FALSE	FALSE	FALSE
		CategoryDescription	Details about category	text		FALSE	TRUE	FALSE
		CategoryCreatedAt	Record creation time	timestamp		FALSE	TRUE	FALSE
		CategoryModifiedAt	Last record modification	timestamp		FALSE	TRUE	FALSE

Source: Author, 2024

Now that the details of database entities are clearly set, building the system can continue.

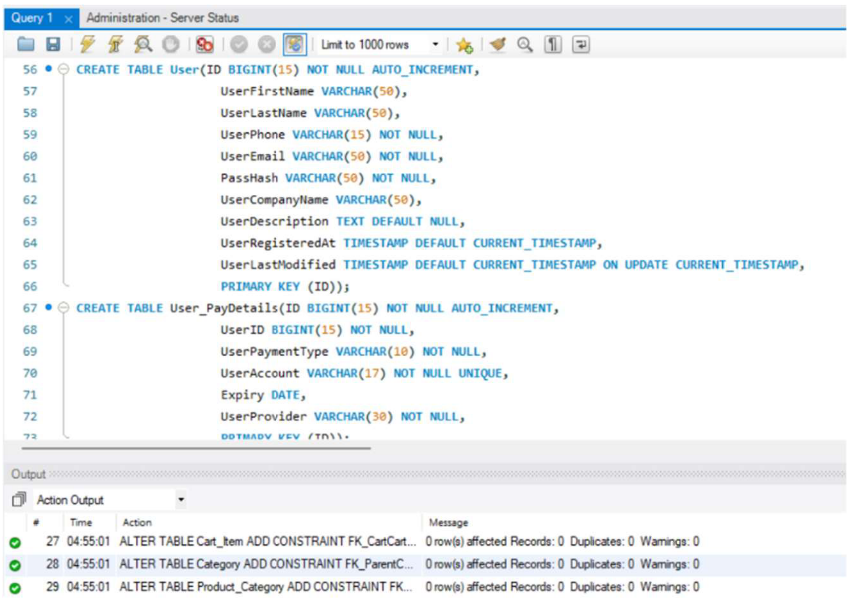
5.5 DDL: Data Definition Language

The process of creating a database in MySQL typically begins with the execution of the command *CREATE DATABASE name*, where *name* is replaced with the desired name of the new database. This command instructs the MySQL DBMS to create a new empty database with the specified name.

Once the database has been created, the next step is to inform the DBMS that we want to work with this database. This is achieved using the command *USE name*. This command sets the default database for the current MySQL session, so that any subsequent commands will be executed within the context of this database.

After the database schema has been created and selected, the next step is to create appropriate tables and columns using Data Definition Language (DDL) commands such as *Create Table*, *Alter Table*, and *Drop Table*. These commands allow us to define the structure of the tables within our database, including the names and data types of columns, constraints on the data, and relationships between tables.

Image 7 A portion of DDL commands used for database creation.



```
CREATE TABLE User(ID BIGINT(15) NOT NULL AUTO_INCREMENT,
                    UserFirstName VARCHAR(50),
                    UserLastName VARCHAR(50),
                    UserPhone VARCHAR(15) NOT NULL,
                    UserEmail VARCHAR(50) NOT NULL,
                    PassHash VARCHAR(50) NOT NULL,
                    UserCompanyName VARCHAR(50),
                    UserDescription TEXT DEFAULT NULL,
                    UserRegisteredAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                    UserLastModified TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
                    PRIMARY KEY (ID));
CREATE TABLE User_PayDetails(ID BIGINT(15) NOT NULL AUTO_INCREMENT,
                    UserID BIGINT(15) NOT NULL,
                    UserPaymentType VARCHAR(10) NOT NULL,
                    UserAccount VARCHAR(17) NOT NULL UNIQUE,
                    Expiry DATE,
                    UserProvider VARCHAR(30) NOT NULL,
                    PRIMARY KEY (ID));
```

#	Time	Action	Message
27	04:55:01	ALTER TABLE Cart_Item ADD CONSTRAINT FK_CartCart...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
28	04:55:01	ALTER TABLE Category ADD CONSTRAINT FK_ParentC...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
29	04:55:01	ALTER TABLE Product_Category ADD CONSTRAINT FK...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

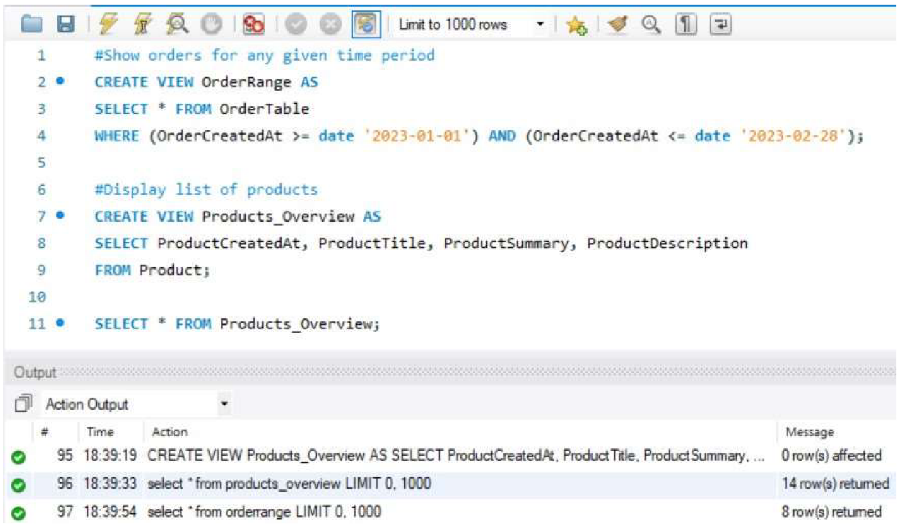
Source: Author, 2024

5.6 DML: Data Manipulation Language

Once the necessary tables are created and any relevant constraints are put in place to ensure data integrity, it is desirable to set up additional tools and instruments that can make working with the database easier and more efficient.

One such tool is the use of views, which allow a great deal of customisation to be enjoyed by users. Views can be customized to display only the columns and rows that are relevant to a particular task, and they can be updated as necessary to reflect changes in the underlying data, thus providing you a schema-free database experience. For example, *OrderRange* view from Image 8 selects all orders from the *OrderTable* that were created within a given period. This view can be used by managers and analysts to quickly obtain a list of orders placed during a specific period for. The *Products_Overview* view selects information about each product from the Product table and groups them by their creation date. This demonstrates how views can be used by marketing and sales teams to review the products offered by the retail business and make informed decisions about promotions or product retirements.

Image 8 OrderRange and Products_Overview views.



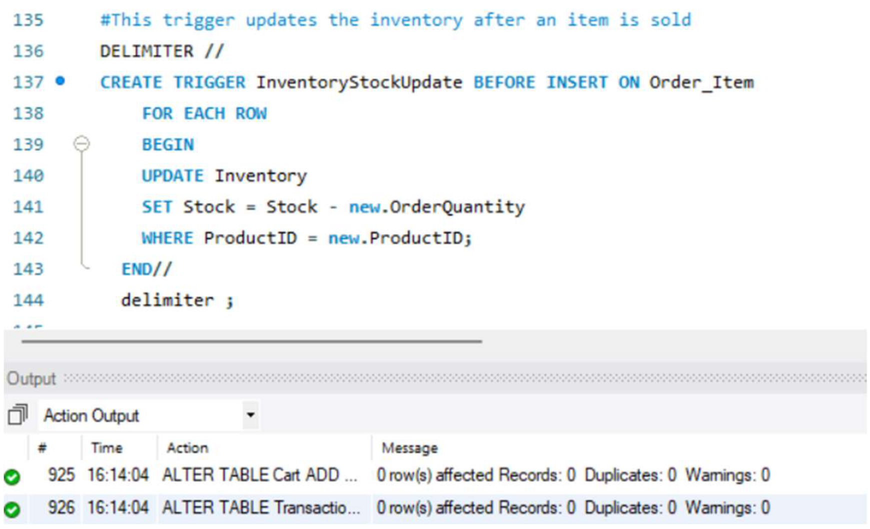
```
#Show orders for any given time period
CREATE VIEW OrderRange AS
SELECT * FROM OrderTable
WHERE (OrderCreatedAt >= date '2023-01-01') AND (OrderCreatedAt <= date '2023-02-28');

#Display list of products
CREATE VIEW Products_Overview AS
SELECT ProductCreatedAt, ProductTitle, ProductSummary, ProductDescription
FROM Product;

SELECT * FROM Products_Overview;
```

```
#   Time      Action                                                                          Message
95  18:39:19  CREATE VIEW Products_Overview AS SELECT ProductCreatedAt, ProductTitle, ProductSummary, ...  0 row(s) affected
96  18:39:33  select * from products_overview LIMIT 0, 1000                                   14 row(s) returned
97  18:39:54  select * from orderrange LIMIT 0, 1000                                          8 row(s) returned
```

Source: Author, 2024

Another important feature of DML is the use of triggers, which are actions that are automatically performed by the database in response to specific events or changes in the data. They allow automation of a lot of user interaction with the database.

Image 9 InventoryStockUpdate trigger.



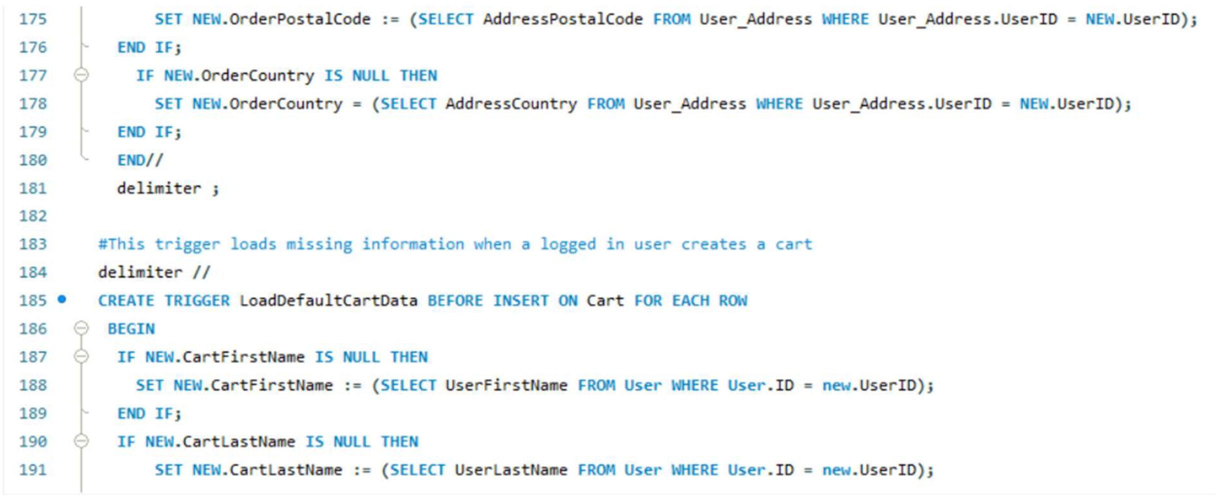
```
#This trigger updates the inventory after an item is sold
DELIMITER //
CREATE TRIGGER InventoryStockUpdate BEFORE INSERT ON Order_Item
    FOR EACH ROW
    BEGIN
    UPDATE Inventory
    SET Stock = Stock - new.OrderQuantity
    WHERE ProductID = new.ProductID;
  END//
  delimiter ;
```

```
Output
Action Output
    #    Time      Action                     Message
    925  16:14:04  ALTER TABLE Cart ADD ...   0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
    926  16:14:04  ALTER TABLE Transactio...  0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
```

Source: Author, 2024

This trigger is used to automatically update the Inventory table when a sale has been successfully completed, simply using the information that is already stored by the database. Since the database knows precisely which and how much of goods were sold, it feels very natural to use that knowledge and let the database quietly update itself when a shopping cart turns into an order.

Image 10 Triggers for time optimisations.



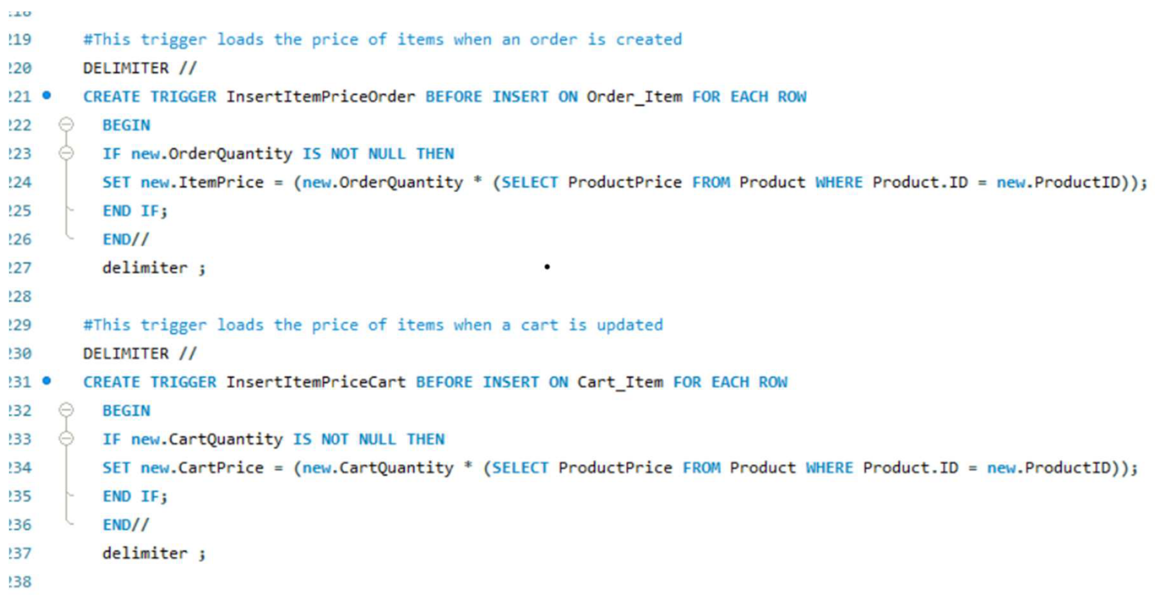
```
      SET NEW.OrderPostalCode := (SELECT AddressPostalCode FROM User_Address WHERE User_Address.UserID = NEW.UserID);
  END IF;
    IF NEW.OrderCountry IS NULL THEN
      SET NEW.OrderCountry = (SELECT AddressCountry FROM User_Address WHERE User_Address.UserID = NEW.UserID);
  END IF;
  END//
  delimiter ;

#This trigger loads missing information when a logged in user creates a cart
delimiter //
CREATE TRIGGER LoadDefaultCartData BEFORE INSERT ON Cart FOR EACH ROW
 BEGIN
  IF NEW.CartFirstName IS NULL THEN
    SET NEW.CartFirstName := (SELECT UserFirstName FROM User WHERE User.ID = new.UserID);
  END IF;
  IF NEW.CartLastName IS NULL THEN
      SET NEW.CartLastName := (SELECT UserLastName FROM User WHERE User.ID = new.UserID);
```

Source: Author, 2024

As the retail store that is being designed throughout this thesis is targeting long term user commitments, it is safe to assume that all clients will be registered in our system. So,

naturally, when the company and the user are old friends, the company is already well aware of who they are, why spend time by going over the same details again?
As per my project, I’ve implemented two triggers that load the user information if that data is already saved in the system. One trigger covers carts and another finalised orders. A good rule of thumb is to never try to force these efforts in automatization to other users – after all, they might have different opinion of what is convenient. So, for these triggers, I’ve decided that they should only activate and quietly complete the form if a user purposely sends that form with missing information – this way nobody gets upset over improvements they’ve never wanted.

Image 11 InsertItemPrice triggers.

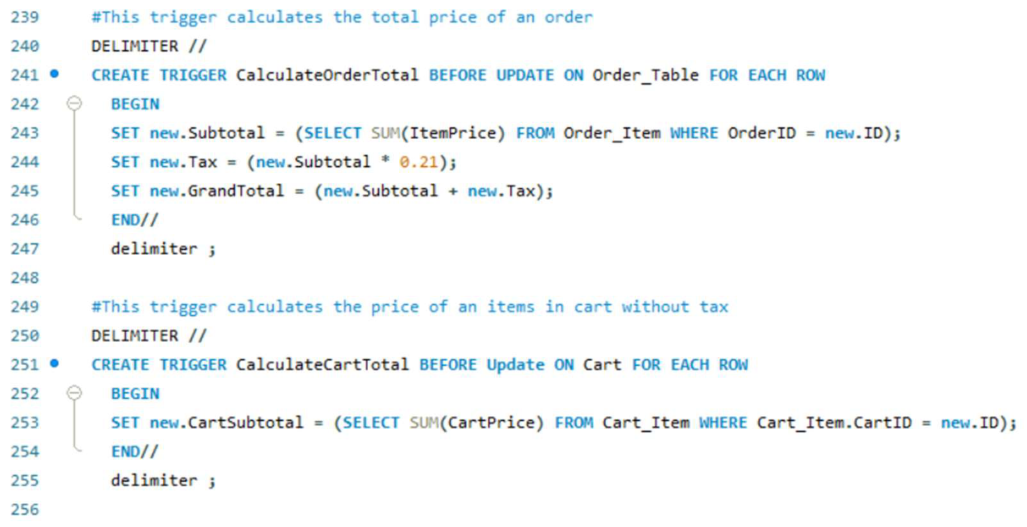


```
#This trigger loads the price of items when an order is created
DELIMITER //
CREATE TRIGGER InsertItemPriceOrder BEFORE INSERT ON Order_Item FOR EACH ROW
  BEGIN
  IF new.OrderQuantity IS NOT NULL THEN
  SET new.ItemPrice = (new.OrderQuantity * (SELECT ProductPrice FROM Product WHERE Product.ID = new.ProductID));
  END IF;
  END//
  delimiter ;

#This trigger loads the price of items when a cart is updated
DELIMITER //
CREATE TRIGGER InsertItemPriceCart BEFORE INSERT ON Cart_Item FOR EACH ROW
  BEGIN
  IF new.CartQuantity IS NOT NULL THEN
  SET new.CartPrice = (new.CartQuantity * (SELECT ProductPrice FROM Product WHERE Product.ID = new.ProductID));
  END IF;
  END//
  delimiter ;
```

Source: Author, 2024

The idea behind these two triggers is quite simple. First of all, again, there are two because we need to implement the same thing for both shopping carts and shipped orders. So, what is happening here is that we want to see a monetary equivalent of things in the cart or in the order, but due to the architecture of the system that is not going as straightforward as it seems. When an item is added into the shopping cart, it is not the same exact item that is stored in the database. More specifically, it is an intermediate object, something between a cart and an item. This is usually done to avoid many-to-many entity relations. In our case, just before a *cart item* is added to the *Cart_Item* table, the trigger quickly checks that the number of items added to the cart is not zero, then collects the price of items from another entity in the database and finally, updates the price of *cart item* itself. From this point onwards, two different triggers will take over.

Image 12 Triggers for calculating total sum to be paid by client.



```
#This trigger calculates the total price of an order
DELIMITER //
CREATE TRIGGER CalculateOrderTotal BEFORE UPDATE ON Order_Table FOR EACH ROW
  BEGIN
  SET new.Subtotal = (SELECT SUM(ItemPrice) FROM Order_Item WHERE OrderID = new.ID);
  SET new.Tax = (new.Subtotal * 0.21);
  SET new.GrandTotal = (new.Subtotal + new.Tax);
  END//
  delimiter ;

#This trigger calculates the price of an items in cart without tax
DELIMITER //
CREATE TRIGGER CalculateCartTotal BEFORE Update ON Cart FOR EACH ROW
  BEGIN
  SET new.CartSubtotal = (SELECT SUM(CartPrice) FROM Cart_Item WHERE Cart_Item.CartID = new.ID);
  END//
  delimiter ;
```

Source: Author, 2024

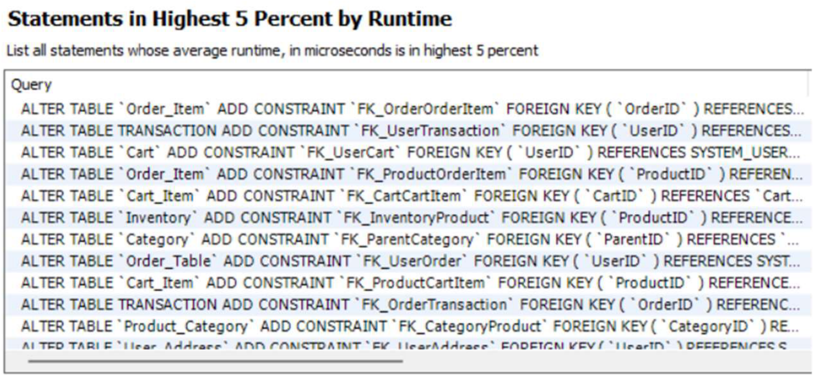
In order to calculate the total monetary sum representing each shopping cart and each order, a bit of data manipulation and math is needed. The situation may look easier with the shopping cart total, but that is only due to the fact that the shopping cart doesn't include taxes. What's actually happening is that each time, just before tables *Cart* or *Order_Table* are updated, MySQL is looking for cart items that our previous triggers have helpfully updated with their price multiplied by their quantity. Once the cart items are found, their prices are summed up. And for calculating the *order* total the process would be very similar, only with some addition and multiplication.

6 Analysis and further improvement

6.1 Performance analysis

As the retail store is expected to grow and evolve, it is essential to ensure that the database is performing optimally to ensure fast and efficient access to data. Previously in this thesis we have utilized MySQL Workbench, a visual tool for designing, managing, and monitoring MySQL databases. Among other things, Workbench provides a built-in dashboard that displays high-level performance metrics, as well as detailed performance metrics contained in the sys schema. Author has used this tool to identify and analyse key performance metrics such as CPU usage, memory utilization, query performance, and index usage.

Image 13 Table statistics of the database.



Statements in Highest 5 Percent by Runtime

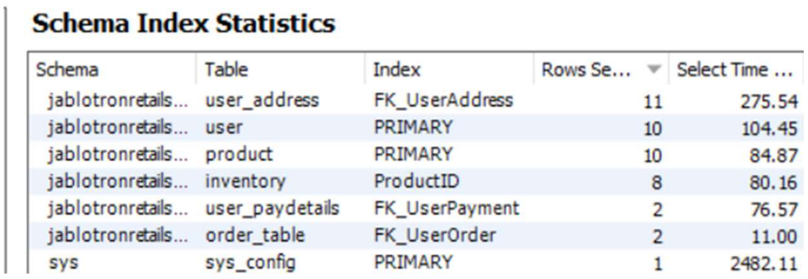
List all statements whose average runtime, in microseconds is in highest 5 percent

```
Query
ALTER TABLE `Order_Item` ADD CONSTRAINT `FK_OrderOrderItem` FOREIGN KEY ( `OrderID` ) REFERENCES...
ALTER TABLE TRANSACTION ADD CONSTRAINT `FK_UserTransaction` FOREIGN KEY ( `UserID` ) REFERENCES...
ALTER TABLE `Cart` ADD CONSTRAINT `FK_UserCart` FOREIGN KEY ( `UserID` ) REFERENCES SYSTEM_USER...
ALTER TABLE `Order_Item` ADD CONSTRAINT `FK_ProductOrderItem` FOREIGN KEY ( `ProductID` ) REFEREN...
ALTER TABLE `Cart_Item` ADD CONSTRAINT `FK_CartCartItem` FOREIGN KEY ( `CartID` ) REFERENCES `Cart...
ALTER TABLE `Inventory` ADD CONSTRAINT `FK_InventoryProduct` FOREIGN KEY ( `ProductID` ) REFERENCE...
ALTER TABLE `Category` ADD CONSTRAINT `FK_ParentCategory` FOREIGN KEY ( `ParentID` ) REFERENCES `...
ALTER TABLE `Order_Table` ADD CONSTRAINT `FK_UserOrder` FOREIGN KEY ( `UserID` ) REFERENCES SYST...
ALTER TABLE `Cart_Item` ADD CONSTRAINT `FK_ProductCartItem` FOREIGN KEY ( `ProductID` ) REFERENCE...
ALTER TABLE TRANSACTION ADD CONSTRAINT `FK_OrderTransaction` FOREIGN KEY ( `OrderID` ) REFERENC...
ALTER TABLE `Product_Category` ADD CONSTRAINT `FK_CategoryProduct` FOREIGN KEY ( `CategoryID` ) RE...
ALTER TABLE `User_Address` ADD CONSTRAINT `FK_UserAddress` FOREIGN KEY ( `UserID` ) REFERENCES S
```

Source: Author, 2024

This project has provided us with an interesting insight into database performance. As it turns out, Alter Table operations and foreign key constraints are very expensive, and it brings to mind an interesting experiment – try to figure out if we could go faster with less normalisation.

Image 14 Database index statistics.



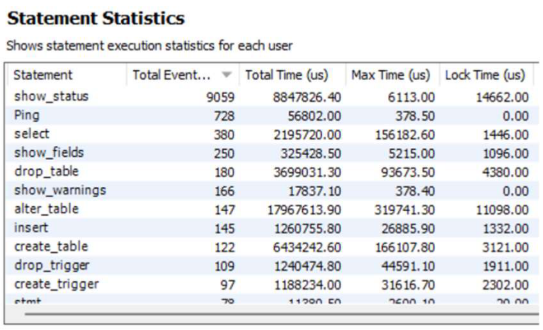
Schema Index Statistics

Schema	Table	Index	Rows Se...	Select Time ...
jablotronretails...	user_address	FK_UserAddress	11	275.54
jablotronretails...	user	PRIMARY	10	104.45
jablotronretails...	product	PRIMARY	10	84.87
jablotronretails...	inventory	ProductID	8	80.16
jablotronretails...	user_paydetails	FK_UserPayment	2	76.57
jablotronretails...	order_table	FK_UserOrder	2	11.00
sys	sys_config	PRIMARY	1	2482.11

Source: Author, 2024

In terms of performance, the select time for the tables seems within reason with the largest table *user_address* taking only 0.2 milliseconds to select 11 rows.

Image 15 Database statement statistics by MySQL Workbench.



Statement Statistics

Shows statement execution statistics for each user

Statement	Total Event...	Total Time (us)	Max Time (us)	Lock Time (us)
show_status	9059	8847826.40	6113.00	14662.00
Ping	728	56802.00	378.50	0.00
select	380	2195720.00	156182.60	1446.00
show_fields	250	325428.50	5215.00	1096.00
drop_table	180	3699031.30	93673.50	4380.00
show_warnings	166	17837.10	378.40	0.00
alter_table	147	17967613.90	319741.30	11098.00
insert	145	1260755.80	26885.90	1332.00
create_table	122	6434242.60	166107.80	3121.00
drop_trigger	109	1240474.80	44591.10	1911.00
create_trigger	97	1188234.00	31616.70	2302.00

Source: Author, 2024

In conclusion, our analysis using MySQL Workbench revealed several areas where the retail store database could be optimized to improve performance, including optimizing queries and reducing CPU overhead. With ongoing monitoring and optimization of key performance metrics, it is possible to ensure that the database continues to operate at peak efficiency and can handle increasing workloads as the retail store continues to grow and evolve. To draw a conclusion, MySQL Workbench is an effective tool for development, analysing database performance and learning crucial insights about database design.

6.2 Further development

A potential and logical upgrade for the built database system would be a connected web front for users to purchase and order items, manage their subscriptions. Store workers will benefit a lot from a similar web-based system that makes working with orders a more trivial and accessible task. Unfortunately, despite author's attempts to receive some kind of feedback from Jablotron Group, it does not seem likely that the company is interested in opening a retail store at this time.

7 Conclusion

The retail industry is constantly evolving, and businesses must stay up to date with the latest technologies and trends in order to remain competitive. One of the key technologies that has revolutionized the retail industry in recent years is the database. By storing and managing large amounts of data on customers, products, and sales, databases have enabled retailers to make informed decisions based on accurate and up-to-date information. Through this thesis, we have explored the design and implementation of a retail database, including the creation of triggers, views, and procedures that can help to automate various tasks and ensure data integrity. The database that has been developed has the potential to bring significant benefits to retail businesses by streamlining operations, improving customer service, and increasing sales and revenue. By automating tasks such as inventory management, sales tracking, and customer data management, the database can free up workers' time and allow them to focus on more important tasks, such as customer service and sales. The use of triggers, views, and procedures can also help to ensure that data is accurate and consistent across different parts of the business, making it easier for retail workers to make informed decisions. To summarise, the database that has been developed through this thesis represents a powerful tool for retail businesses looking to stay ahead of the curve and succeed in an increasingly competitive industry.

8 References

1. *ACM (Association for Computer Machinery). ACM Awards A. M. Turing Award Laureate - Charles W. Bachman* [online] 1973. https://amturing.acm.org/award_winners/bachman_1896680.cfm. Accessed 30 December 2022.
2. DATE, Chris, 2003. *An Introduction to Database Systems.* Boston: Addison-Wesley Longman. ISBN 9780321197849.
3. SHELDON, Robert. MOES, Geoff, 2005. *Beginning MySQL.* Indianapolis, Wiley. ISBN 0764579509.
4. LANGLOIS, Richard, 2002. *Computers and semiconductors. Technological innovation and economic performance*. Princeton: Princeton University Press. p. 265-284. ISBN 0691090912.
5. DANIEL, Eric, MEE, Denis, CLARK, Mark, 1998. *Magnetic Recording: The First 100 Years.* New York: IEEE Press. ISBN 978-0780347090.
6. GROSVENOR, Edwin, WESSON, Morgan, 2016. *Alexander Graham Bell.* Boston: New Word City. ISBN 1612309844.
7. *KRAMER, Pieter. Reflective optical record carrier* [online]. 26 November 1991. https://patents.google.com/patent/US5068846. Accessed 30 December 2022.
8. BERG, Kristi, SEYMOUR, Tom, GOEL, Richa. History Of Databases. *International Journal of Management & Information Systems (IJMIS).* Volume 17, Issue 1, First Quarter 2013, p. 29-35. ISSN 1546-5748.
9. ROWE, Lawrence. History of the Ingres Corporation. *IEEE Annals of the History of Computing*. Volume 34, Issue 4, October-December 2012, p. 58-70. ISSN 1058-6180.
10. *ACM (Association for Computer Machinery). ACM Awards A. M. Turing Award Laureate - Edgar F. Codd* [online]. 1981. https://amturing.acm.org/award_winners/codd_1000892.cfm. Accessed 30 December 2022.
11. SCOTT, Harris, PRESTON, Curtis, 2007. *Backup & Recovery: Inexpensive Backup Solutions for Open Systems.* Sebastopol: O'Reilly, ISBN 978-0596102463.

12. NANDI, Veena. Maintaining Database: Business Intelligence Tool for Competitive Advantage. *Business Intelligence Journal.* Volume 5, Issue 2, July 2012, p. 352-357. ISSN 1918-2325.
13. GARBA, Musa. A Comparison of NoSQL and Relational Database Management Systems (RDBMS). *Kasu Journal of Mathematical Sciences (KJMS).* Volume 1, Issue 2, December 2020. ISSN 2734-3439.
14. *MongoDB. Supported SQL Functions and Operations* [online]. 2022. https://www.mongodb.com/docs/bi-connector/current/supported-operations/. Accessed 30 December 2022.
15. HARRINGTON, Jan, 2016. *Relational Database Design and Implementation: Clearly Explained, Fourth Edition.* Burlington: Morgan Kaufman. ISBN 978-0128043998.
16. HERNANDEZ, Michael, 2014. *Database Design for Mere Mortals. Third Edition.* Ann Arbor: Addison-Wesley Professional. ISBN 978-0-321-88449-7.

8.1 Table references

Cypress Semiconductor AN99111 – Parallel NOR Flash Memory: An Overview [online]. 2017. https://www.infineon.com/dgdl/Infineon-AN99111_Parallel_NOR_Flash_Memory_An_Overview-ApplicationNotes-v03_00-EN.pdf?fileId=8ac78c8c7cdc391c017d0742858b6597&utm_source=cypress&utm_medium=referral&utm_campaign=202110_globe_en_all_integration-files. Accessed 30 December 2022.

8.2 Image references

Stack Overflow Developer Survey 2022 [online]. 2022. https://survey.stackoverflow.co/2022/#most-popular-technologies-database-prof. Accessed 30 December 2022.