2020

Managing Inventory: A Study of Databases and Database Management Systems

Jemal M. Jemal
The College of Wooster, jjemal20@wooster.edu

Managing Inventory: A study of databases and database management systems

Independent Study Thesis

Presented in Partial Fulfillment of the Requirements for
the Degree Bachelor of Arts in Computer Science in the
Department of Mathematical & Computational Sciences at
The College of Wooster

by
Jemal Mahdi Jemal
The College of Wooster
2020

Advised by:

Dr. Tom Montelione

THE COLLEGE OF
WOOSTER

Abstract

Databases play an important role in the storage and manipulation of data. Databases and database management systems allow for fast and efficient data querying that has recently become increasingly important in most companies and organizations. This paper introduces a few of the different types of database management systems that are in widespread use today. It introduces some important terminology related to databases and database management systems. This paper also briefly discusses web user interfaces, highlighting important user interface design principles. Finally, an inventory management system is implemented for a local stationery store and is integrated with a web application to serve as the front end.

This work is dedicated to the future generations of Wooster students.

Acknowledgments

First and foremost, I would like to thank God for blessing me with the strength and dedication to successfully complete my independent research. I would also like to express my deepest appreciation to my advisor, Dr. Tom Montelione for guiding me through this process and continuously pushing me to explore deeper concepts in my research area. I appreciate all the mentorship and edits I received from my advisor.

I would also like to express my gratitude to my parents and siblings for always encouraging me to push myself to the best of my abilities. They have constantly been supporting me throughout this process.

Finally, I would like to express my gratitude to my friends and family that have been motivating and supporting me during the challenging parts of my research. Thank you Sam for bearing with me as I complained about deadlines and workload throughout the semester. I appreciate all the guidance that I received while completing my research.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Humans began to use databases as a means of organizing information long before the creation of computers. Accountants carried ledgers, librarians created catalogs, and business owners logged their inventories. In the early 1960's computers became affordable to businesses and with it came the automatization of databases. Since then, databases and database management systems have become an increasingly popular tool for the storage and handling of data. Today, computer databases have allowed humans to access and interact with data seamlessly from anywhere around the world.

The main aim of this research paper is to provide an introduction to database management systems and discuss their usage in business settings as a method of managing different types of data. In the software portion of this independent study, an inventory management system is created for a client that owns a stationery store in Addis Ababa, Ethiopia. This store functions as both a wholesaler and retailer as it sells different types of stationery products to individual people as well as other businesses, schools, and offices. Currently, the business uses excel spreadsheet files to manage its inventory. When ordered stationery products arrive at the store, the accountant logs all products on an excel sheet and the products are stocked in a warehouse. Once the products are stocked, there is currently no method that

keeps track of what specific items have been sold. Moreover, the business does not currently use a point of sale system which makes it difficult to know when it is best to restock. The client has identified the drawbacks of using this approach and wants to automate this system to streamline the process of managing inventory within her business, thus making the process more convenient for her and her employees.

Another portion of this research examines the design of web-user interfaces. Designing a good web-user interface requires adherence to certain design principles in terms of usability and visual aesthetics. This study looks at the research that has been done in terms of how certain design principles contribute to user satisfaction when visiting a particular website, and how certain design elements motivate users to re-visit a website. The implemented inventory management system is web-based and thus requires a certain type of web design. Certain design principles from this study are used when implementing the user interface of the inventory management system.

1.1 What is an inventory management system?

An inventory management system is an automated software system used for monitoring stock levels, product orders, and sales records. There are many different hardware and software components that make up a fully developed inventory management system. An inventory management system integrated with a point of sale system is able to detect when a particular product is sold and automatically removes that product from stock. When stock levels are low, the system automatically notifies an employee so that products are reordered. Advanced systems are also able to automatically reorder products when stock levels are low. Inventory management systems are very advantageous for businesses as they are used to generate different types of reports such as sales history reports. Business owners are able to see which

products are selling well and which items might need to be discounted to prevent certain types of products from just sitting in stock. This consequently increases the profitability of a business. Additionally, complex data mining techniques can be performed on the data that exists in an inventory management system. These techniques look for certain patterns in the data that is stored and can forecast future sales for specific products. This allows business owners to understand customer buying habits and thus enables them to cater more towards their customer demand. This ultimately increases sales and customer loyalty.

There are various commercial inventory management systems available for purchase. Some systems also offer cloud-based data storage for their customers. Some of the more popular systems include QuickBooks Enterprise and Vend.

1.2 Outline of Approach

This paper begins by introducing and exploring certain features of the different types of databases and database management systems available. More attention and focus is given to relational database management systems since the inventory management system software is built on the relational database model. Following this, the concept of database architecture and database normalization are discussed. This paper then looks into database security and suggests some best practices for protecting data stored in a database. For the web design portion of the study, different types of web-based user interfaces and user interface design principles are explored. Finally, the software section of this paper explores the programming languages and web-frameworks used to design and implement the inventory management system software. An entity-relationship diagram is also implemented for the underlying relational database, and the user interface design for the web application is discussed.

CHAPTER 2

Databases and Database Management Systems

This chapter gives a general overview of databases and database management systems. It starts by defining terminology that is crucial to the understanding of databases. It then introduces a few common types of database management systems and discusses their main features including their applications in the business environment.

2.1 Databases

A database is a collection of data that is stored on a computer system and can be used to retrieve and manipulate data in a structured manner. It is often hidden behind the tools and services used by humans every day. Databases span almost all areas where computers are in use [32]. This includes areas such as education, medicine, law, business and many more. Databases are very popular because they simplify the process of data management, especially when dealing with large amounts of data due to the systematic storage of the data. Section 2.2 discusses some of the different methods used to store data. A database can store data with different formats including textual, audio, graphical and many other data formats.

A collection of data can be called a database if it contains the following three implicit properties:

- The data stored represents some aspect of the real world. Alterations observed in the real world are represented in the database
- The data is logically coherent and not a mere random collection
- The database serves a specific purpose to a group of users at some point in the processing of the stored data [32].

There are no restrictions on how large or small databases can get. It can be as small as information on everyone in a household and can get as big as billions of records being stored by companies such as Amazon and Facebook.

Before further discussing databases and database management systems, the following database terminology needs to be defined:

- **Records (Tuples)**: The rows in a database. Each row stores data for a single instance
- **Fields (Attributes)**: The columns in a database. Each column stores one type of data
- **Schema**: The structure and organization of a database. Specifies what fields a database contains
- **Relation**: An individual table in a database. Usually given a name and has a collection of records
- **Query**: A specific request for a subset of data within the database
- **View**: A virtual table that contains specific records resulting from a query. The virtual table resulting from a view statement is not stored permanently.

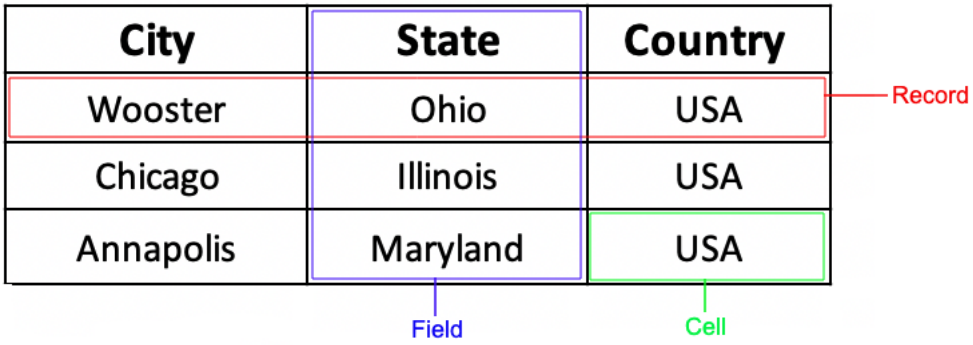


Figure 2.1: Visualization of fields, records and cells

Figure 2.1 is a sample relation that contains 3 fields and 3 records.

The most basic database type is called a flat-file database. This type of database consists of a single table containing data and its records do not have a structured relationship. Flat-file databases store data in fields and records. One big advantage of a flat-file database is its ease of creation, making it a simple and convenient way to store small amounts of data. However, flat-file databases become increasingly inefficient when dealing with large amounts of data. Moreover, flat-file databases do not have the mechanism to prevent data redundancy which is often a key factor in reducing search times in databases. Data redundancy is the unnecessary duplication of data which slows down the process of searching and adds unnecessary storage size to the database. CSV (comma-separated value) files and other spreadsheet files are common examples of flat-file databases.

2.2 Database Management Systems

A database management system is software that is used to manage the storage, manipulation, and retrieval of data from a database. Once a database has been created, the database management system stores a database catalog containing descriptive information about the database [32]. This is more commonly known as

metadata. Metadata contains information such as the database structure and the different data types contained in the database. Moreover, a database management system provides a wide range of functionalities to users such as facilitating administrative operations, data security and data recovery. It serves as an interface between the database and the application programs that make use of the database. Figure 2.2 contains a simplified version of a database system environment. This figure shows that end-users are able to access data from a database by use of application programs. Application programs can access data stored in a particular database by sending queries to the database management system that is, in turn, able to access the requested subset of data from the stored database. [32].

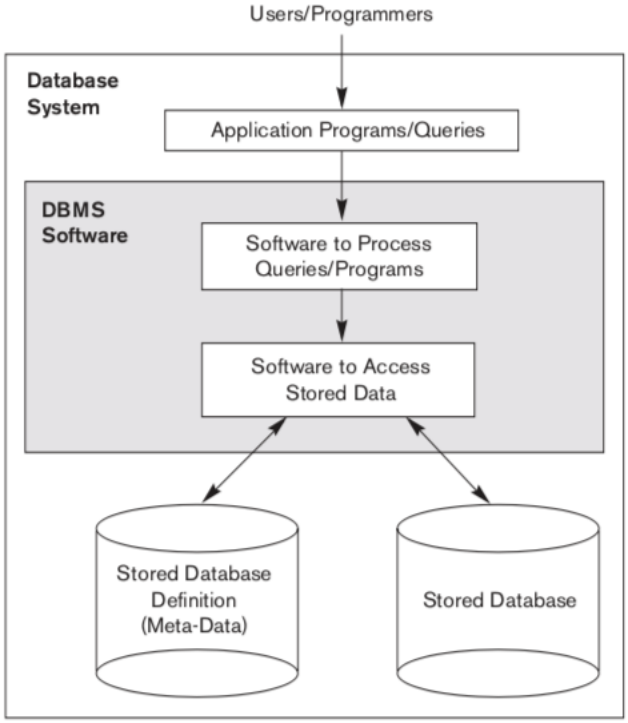
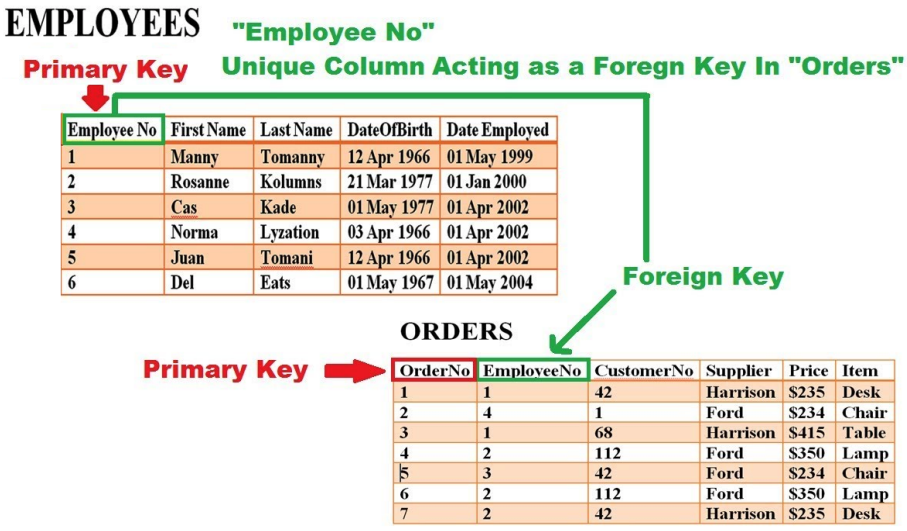


Figure 2.2: Simplified version of a database system environment [32]

The following subsections discuss four different types of database management systems.

2.2.1 Relational Database Management System (RDBMS)

A relational database management system (RDBMS) is a software system used to manage relational databases. A relational database is a type of database that consists of data that have relationships between them. Often a relational database is organized in different tables where relationships are made between the tables by the use of primary keys and foreign keys. Primary keys are unique keys assigned to each row to help identify each record in a table. Foreign keys are used to create references between different tables. Foreign keys are defined as primary keys in one table and exist as a field in the second table to provide the required link between the two tables. The purpose of foreign keys is to identify a particular record from the referenced table. Figure 2.3 contains two tables that are connected via a foreign key. *Employee No* is a primary key in the Employees table but is a foreign key in the Orders table. The use of foreign keys to create a link between different tables is what makes relational databases unique and is an efficient way to help eliminate the issue of data redundancy. In addition to using foreign keys, a RDBMS removes data redundancy using a process called normalization *(refer to chapter 4)*.



Employee No	First Name	Last Name	DateOfBirth	Date Employed
1	Manny	Tomanny	12 Apr 1966	01 May 1999
2	Rosanne	Kolumns	21 Mar 1977	01 Jan 2000
3	Cas	Kade	01 May 1977	01 Apr 2002
4	Norma	Lyzation	03 Apr 1966	01 Apr 2002
5	Juan	Tomani	12 Apr 1966	01 Apr 2002
6	Del	Eats	01 May 1967	01 May 2004

OrderNo	EmployeeNo	CustomerNo	Supplier	Price	Item
1	1	42	Harrison	$235	Desk
2	4	1	Ford	$234	Chair
3	1	68	Harrison	$415	Table
4	2	112	Ford	$350	Lamp
5	3	42	Ford	$234	Chair
6	2	112	Ford	$350	Lamp
7	2	42	Harrison	$235	Desk

Figure 2.3: Primary key and foreign key[23]

A relational database management system allows users to create, read, update and delete data from a relational database. This functionality also goes by the acronym 'CRUD'. Users can perform CRUD functionality by using SQL *(Structured Query Language)* [7]. SQL is a language used to communicate with a database and is also the standard language used for relational database management systems. It is a declarative language that tells the database what to do, but not how to do it. For example, to list the customer number and supplier for all orders with a price greater than $300 in the orders table from figure 2.3, the following SQL query is used:

```
SELECT CustomerNo, Supplier FROM ORDERS WHERE Price > 300
```

A RDBMS also provides many advanced features such as concurrency control. Concurrency control is the system's ability to determine how to handle concurrent manipulation of data. This usually happens when multiple people are trying to access and edit the same data from a database at the same time. For example, if two different people try to reserve the same Airbnb for the same night, the RDBMS has rules to determine who gets the reservation by properly handling all the concurrent requests that it receives.

A RDBMS is one of the most commonly used types of database management systems. This is primarily due to its ease of use. SQL is simple to learn and provides a high-level of abstraction for its users. This enables users to focus more on what needs to be done rather than having to worry about the internal processes of the system. Some of the most popular relational database management systems include MySQL, Oracle DB, SQLite, PostgreSQL, and SQL Server by Microsoft [7]. While MySQL, SQLite, and PostgreSQL are all open-source systems, Oracle DB is owned by Oracle Corporation and can be expensive. Oracle DB is most commonly used in the banking industry as it includes functionality specifically tailored towards banks [7]. Microsoft's SQL Server is also proprietary software that is part of Microsoft's data platform suite.

2.2.1.1 Entity Relationship Diagram (ERD)

An entity-relationship diagram (ERD) is a diagram that illustrates the logical structure of a database. It helps to visualize the relationships that exist in a relational database. In this section, some basic terminology of ERD components and their symbols are presented.

- **Entity**: An entity is a concept about which we want to store an object [3]. It is represented by a rectangle. Entities are usually nouns. Ex: Patient, Doctor
- **Relationship**: A relationship defines how different entities interact with each other. It is represented by a diamond. Relationships are usually verbs. Ex: Supervises, registers
- **Attribute**: An attribute is a property of an entity. One entity consists of many different attributes. Attributes can be represented as ovals or can be presented under each entity name. A horizontal line separates the entity from its attributes.
- **Cardinality**: A cardinality specifies the type of relationship between different attributes. Cardinality includes one-to-one, one-to-many and many-to-many relationships as seen in figure 2.4.
- **Ordinality**: Ordinality describes the relationship between two entities as either optional or mandatory. While cardinality specifies the maximum number of relationships, ordinality specifies the absolute minimum number of relationships [3].

Various styles can be used to express cardinality and ordinality. Some of these styles include Chen style, Martin style, and Bachman style. Figure 2.4 shows how ordinality and cardinality are depicted using the information engineering style.

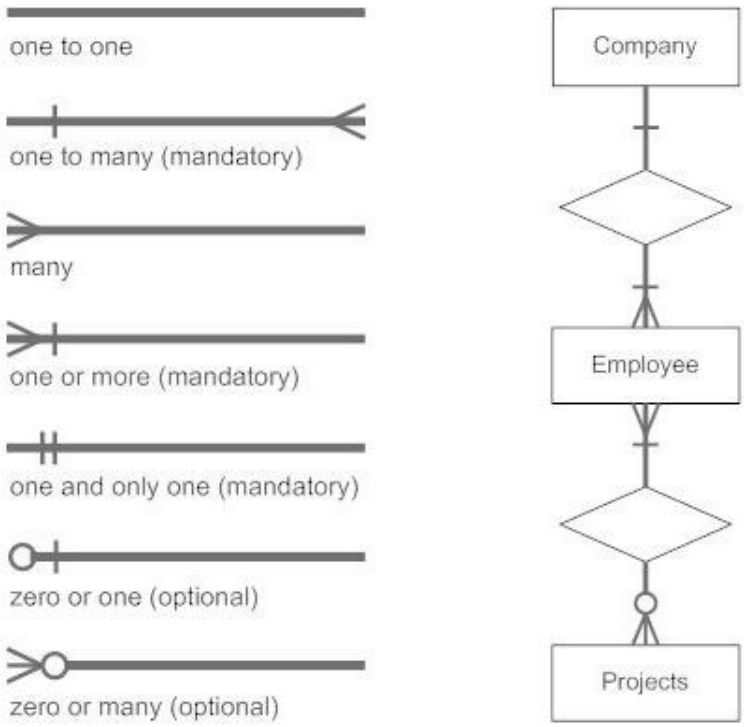


Figure 2.4: Cardinality and ordinality presented using information engineering style [3]

2.2.2 Hierarchical Database Management System (HDBMS)

The hierarchical database management system (HDBMS) is a software system used to manage hierarchical databases. A hierarchical database employs a tree-like structure and stores data using the parent-child relationship. The tree-like structure is made up of nodes and branches. Each node contains different attributes that describe a record. Each record can contain several fields that in turn house data values such as floats, texts, integers or even pointers to other records. The node found at the very top is referred to as the root node and contains dependent nodes in succeeding levels [5]. A parent node can have one or many children. If the parent node has only one child, the parent and the child have a one-to-one relationship. If the parent has more than one child node, the parent and its children have a one-to-many relationship. Figure 2.5 illustrates the parent-child relationship in the

tree-like structure where *Bank* is the root of this database tree. In this example, all parents have a one-to-many relationship with their respective children.

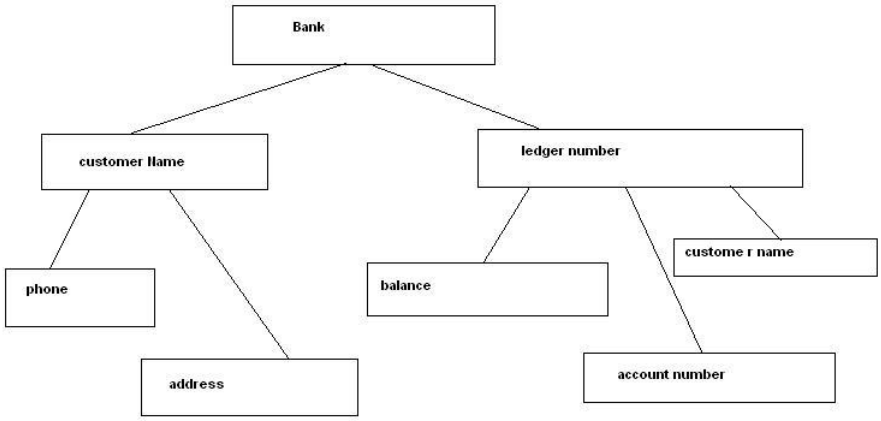


Figure 2.5: Parent child relationships in the hierarchical database model [5]

A HDBMS can be very complicated to design and use. It is inefficient when dealing with many-to-many relationships because it often results in data redundancy [5]. The parent-child relationship of the hierarchical database makes it difficult to delete nodes that are not leaf nodes. For example, the deletion of a parent node results in the deletion of all its children. Moreover, if data is inserted at a child node, the child can only be accessed using its parent which results in an unnecessary overhead for insertion time [5].

Even though HDBMSs are becoming less popular these days, big companies such as IBM and Microsoft still use them. For example, Windows registry makes use of the hierarchical database model. IBM's information management system (IMS) uses the hierarchical database model as well [38].

2.2.3 Object Oriented Database Management System (OODBMS)

An object-oriented database management system is a software system designed to manage object-oriented databases. An object-oriented database stores data in the form of objects. The object-oriented database model extends the approach of the relational database model by using its table-oriented design and implementing an object-oriented approach. An OODBMS follows the principles of object-oriented programming and makes use of classes and objects. Figure 2.6 outlines some features of the object-oriented database model. This figure visually shows that object-oriented databases are a combination of relational database features and object-oriented programming features.

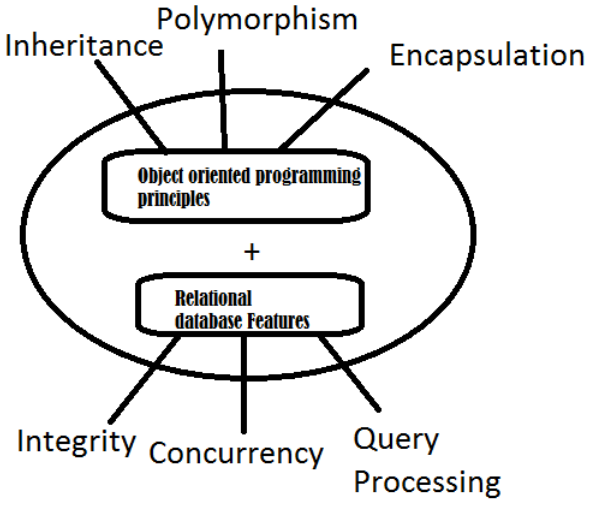


Figure 2.6: Object-oriented database model [40]

Some of the most fundamental features of an object-oriented database include classes, encapsulation, inheritance, association, and persistence. A class is a template used for creating objects. An object has many states and behaviors. Each object in an object-oriented database has a unique object identifier [40]. Inheritance is also an important object-oriented database feature that is used to create new classes that

inherit the properties of their respective parent class and add additional properties. Association is a feature that defines the relationship between different objects in the database. Associations are defined within classes so all objects created from the class include that specific association as one of their properties. Persistence is another object-oriented database feature that enables the creation of persistent objects. Persistent objects are objects that continue to exist in memory even after a program completes its execution. Persistent objects are really useful when data needs to be restored if an application needs to be rerun. It is important in solving issues related to data recovery and concurrency control [40].

There are many advantages to using an OODBMS. An OODBMS is good at handling very complex data relationships since complex data can be accessed merely as objects. If programmers use object-oriented programming to build software, it is much easier to incorporate an OODBMS than another type of DBMS because of the object-relational mapping tools available to manipulate object-oriented data in the database. Data can be accessed as objects from the database when developing software that needs to interact with a database. Additionally, an OODBMS provides quick access to information stored in a database. This is because objects are tracked via their unique object IDs. This eliminates the need for complex foreign key traversals as in the case for RDBMS's [35]. An OODBMS also allows users to create user-defined complex data types. Users can define complex data types since the type of data stored is not restricted to specific data types as in the case with RDBMSs. One drawback to using OODBMS is the complexity required to program OODBMS systems, especially when compared to the techniques used to program RDBMS systems [35].

2.2.4 Network Database Management System (NDBMS)

A network database management system is a software system used to manage network databases. A network database is a database model that extends the hierarchical database model and is designed to solve some of the drawbacks to the hierarchical model. A network database model is a more flexible method of implementing the parent-child relationship available in the hierarchical database model. In the network database model, a child can have one or more parents resulting in a graph structure made up of different nodes. The network model represents data as a tree of records where records are related to each other through pointers.

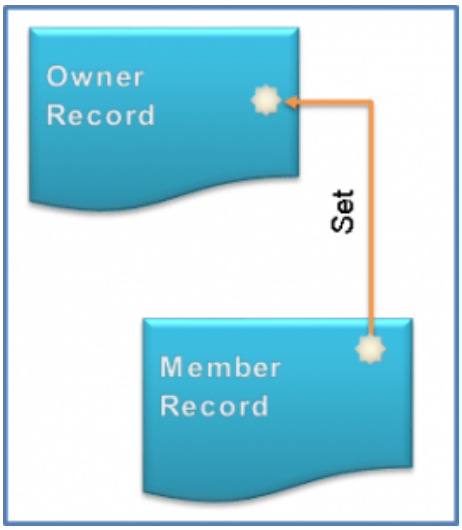


Figure 2.7: Network database model record relationship [1]

Figure 2.7 depicts the relationship between two related records. Relationships in the network database model are referred to as 'sets' [1]. Pointers are used to access records in this model. The relational database equivalence to sets is the use of foreign keys. A network database model has a significant advantage here since there are no columns devoted to relating two different tables. The use of pointers results in reduced disk space consumption and memory usage. Moreover, the

querying performance of a network database model is significantly better than its relational database model counterpart [1]. The network database model provides key advantages over the hierarchical database model due to its flexibility and its ease of access to different nodes through the use of pointers. A major drawback to the network database model is that it is difficult to design, implement and maintain. Due to this reason, many programmers prefer to use relational and object-oriented database models instead.

2.3 NoSQL Databases

NoSQL databases have become an increasingly popular method of building and managing databases. NoSQL, or 'Not Only SQL' refers to a group of non-relational databases that are highly useful when working with big data analytics [37]. With the rise in our dependence on the world wide web, data gathered by large companies and corporations is growing at a record pace. At this scale, conventional data management tools are not able to efficiently handle such large amounts of data. The main advantage of NoSQL databases is that they are designed to handle and process large-scale data. Due to this reason, NoSQL databases provide greater flexibility than their relational database counterpart. NoSQL databases do not require adherence to a specific schema and thus can be used for semi-structured and non-structured data [21]. NoSQL databases are also horizontally scalable. This enables processing efficiency especially when dealing with large-scale datasets. Horizontal scaling provides the ability to use more resources (machines) to the already-existing pool of resources which in-turn increases processing capacity.

NoSQL databases have four different classifications. They are key-value stores, document stores, wide-column stores and graph databases. Key-value stores are types of databases that assign keys to the items being stored. Its main use is

in situations where extremely fast and scalable retrieval of data is needed [21]. Document stores are designed to store data that are encoded in a standard exchange format such as JSON or BSON. This format is used in situations that deal with semi-structured and unstructured data. Specific use-cases are, for example, in building content management systems that manage blogs. Each blog has different components and can be processed without needing to adhere to a specific schema. Wide-column stores are able to store records with really large numbers of dynamic columns. This type of database is commonly used by predictive analytics and large-scale batch data processing [21]. Graph databases use graph structures to store and navigate relationships between different nodes. Graph databases use nodes and edges to store data and their relationships with one another. Graph databases are useful when dealing with data relationships in cases like social networking and fraud detection [22]. Figure 2.8 shows a visualization of a simple graph database created that shows how edges and nodes come together to depict the relationships that Jack has with different people in a social networking site. In this case, the nodes store data (the names of people) and the edges store the relationship between the different nodes.

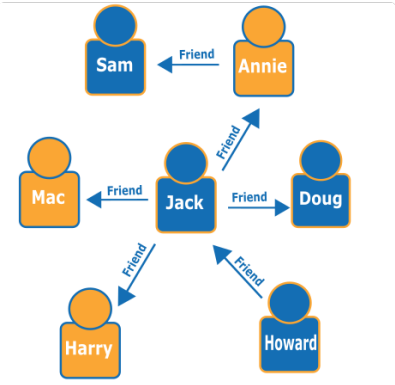


Figure 2.8: Social network graph for Jack [22]

CHAPTER 3

Database Architecture

Different databases are designed according to different database architectures to meet the specific need of a business or organization. A database architecture governs how a database management system is accessed and who is able to access it. A 'tier' refers to the different layers that exist in a specific type of architecture. When designing a database it is important to identify what type of database architecture to use to provide quick and secure access to data. The architecture of a database can either be single or multi-tier. This chapter introduces and discusses 4 different tiers of database architecture, namely, 1-tier, 2-tier, 3-tier and N-tier architectures.

3.1 1-tier Architecture

The 1-tier database architecture is the simplest form of database architecture. This is when the client, the server, and the database all reside on the same machine. While this design is not commonly used in large scale databases, it is very useful in local application development where programmers set up a database for testing purposes. This architecture allows programmers to directly access and communicate with the database, thus providing a quick way for programmers to access required data while in the development phase of the database management system [24]. One

major drawback to using this architecture is the security risks it poses. If the server runs into a problem, communication with the database will be lost. Moreover, the database is vulnerable to attacks as the client has direct and unrestricted access to the database, resulting in the possible loss of data [24].

3.2 2-tier Architecture

The 2-tier database architecture consists of an application layer between the end-user and the database management system [4]. Similar to the client-server architecture, the 2-tier architecture consists of the client tier and the data (database) tier. The end-user can only access the database through the application layer. The application layer, also known as an open database connectivity *(ODBC)*, provides an API that allows the end-user to make calls to the database management system. The 2-tier architecture is able to provide extra security as the end-user is unable to directly access the database. A user's interaction with the backend database is limited by the design of the client tier. Figure 3.1 illustrates this representation. In this figure, the server-side handles the database management system and the client-side handles the application and client layers [17]. In this particular architecture design, the application layer serves as an intermediary allowing communication between the client layer and the database system.

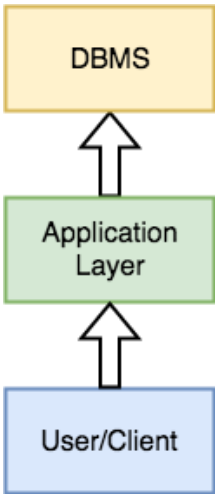


Figure 3.1: 2-tier architecture representation [4]

3.3 3-tier Architecture

3-tier architecture is the most commonly used database architecture. It consists of a presentation layer *(also known as the graphical user interface layer)*, an application layer and a database layer. The user interacts with the presentation layer. For example, the presentation layer can be a form that the user fills out on their browser. Once the user submits the form, the data is sent to the application layer for processing [14]. This technique provides a high level of abstraction because the end-user has no knowledge of the application layer or the database management system but is able to generate different types of views from the database by using the provided graphical user interface. The application layer is responsible for processing the requests submitted from the presentation layer. It provides validation checks before sending the request to the database [14]. It is also responsible for retrieving the requested data from the database and sending that data back to the presentation layer for the end-user to view. The database layer houses the actual database. Depending on the specific request from the user, the data contained in the database layer can be updated, deleted, read or new data can be added. Figure 3.2 depicts

this representation. Despite the added complexity, using this type of architecture is advantageous because it provides increased security. The application layer has specific verification checks in place to ensure that data is not mishandled by the client. Moreover, the separate layers allow for easy scalability. For example, if there is a need to scale the database in an organization, this can be done seamlessly without affecting the other layers in the architecture. Additionally, this architecture makes it easy for experts to be responsible for implementing and maintaining each layer of the system. For example, a web-designer can handle the client layer while the database administrator can be in charge of the database layer. This way, each layer can be handled with expertise, thus increasing the quality and efficiency of each separate layer.

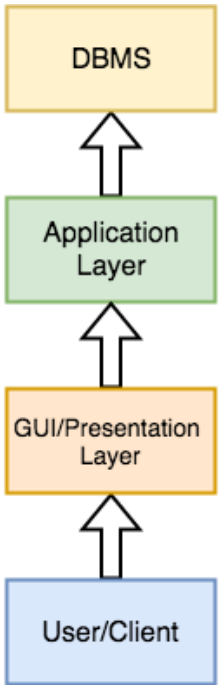


Figure 3.2: 3-tier architecture representation [4]

3.4 N-tier Architecture

An n-tier database architecture refers to multi-tier database architectures. Thus, 2 and 3-tier architectures are classified as n-tier or multi-tier database architectures. The most common type of n-tier architecture is the 3-tier architecture where the presentation layer, application layer, and data layer are all separate. Depending on the size and complexity of the application, these three layers can be placed in different computers that reside in different locations around the world. There are many advantages to using the n-tier architecture including enhanced security such as disaster recovery, as well as easier scalability and maintainability [41].

The n-tier architecture enables the enforcement of security to be different for each layer in the architecture. Most commonly, the data tier needs to be the most secure as all data used by an organization lies in the database. Thus, it is possible to add more security to the data tier without having to make changes in other layers. This method allows designers to have full control over security at each of the different layers.

Scalability is an important component when designing applications that make use of databases. The n-tier architecture enables scalability for each layer in the architecture without affecting the other layers. For example, if a company decides to expand its operation globally, the company can add additional databases to the data tier as can be seen in figure 3.3. The application tier can be scaled by adding new business rules, and the presentation layer can be expanded so the particular user interface can be viewed across multiple types of devices. Figure 3.4 illustrates how the presentation layer can be expanded to support viewing on a computer, mobile devices, and tablets.

Figure 3.3: Scaling the data tier [20]

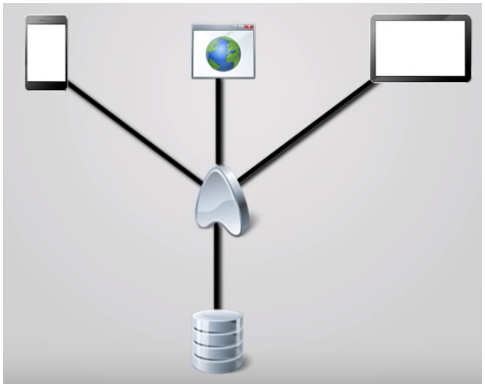


Figure 3.4: Scaling the presentation tier [20]

CHAPTER 4

Database Normalization

Database normalization is a technique used to organize the way data is stored in a database. Database normalization is an important part of database design as it helps eliminate data redundancy. Data redundancy is when multiple copies of the same data exist in a database. This causes undesirable characteristics such as insertion, deletion and update anomalies. Additionally, memory space and speed of data access are also compromised as a result of a database that is unnormalized. In addition to eliminating data redundancy, normalization also plays a key role in ensuring the correct logic in the storage of data.

To better explain insertion, deletion and update anomalies the limitations of the design for table 4.1 are analyzed. This table contains information for a hypothetical car rental company that stores data on cars and their assigned drivers.

V_ID	Make	Model	Year	Driver	Rent Start Date	Rent End Date
001	Toyota	Camry	2012	Katie Rech	08/04/18	08/06/18
002	Mercedes	Benz	2008	William Brook	05/15/18	05/16/18
003	Chevrolet	Cruze	2010	William Brook	06/20/18	07/02/18
004	Honda	Accord	2000	William Brook	10/10/18	10/11/18
005	Audi	Q5	2018	Katie Rech	09/01/18	09/20/18
006	Nissan	Maxima	2016	William Brook	03/13/18	03/15/18

Table 4.1: Car rental details

An insertion anomaly is when inserting new data is not possible because of the absence of other data in the insertion process. Suppose the rental company purchases a new car and wants to enter that information in the table. Unless the new car is assigned to the driver and has a rental date, the driver and rental date information will be set to NULL. Moreover, if the new car is assigned to Katie Rech, her name will need to be inserted in the driver column, increasing data redundancy.

A deletion anomaly occurs when deleting a particular subset of data results in the unintentional loss of other data stored in the database. For example, if the rental company decides to sell the Toyota Camry and the Audi Q5 these cars will be deleted from the database. This will also unintentionally delete the details of a particular driver, Katie Rech.

Update anomalies arise when a table contains many duplicate fields and an update to one field does not update all the corresponding fields in the database. This is known as a partial update and results in data inconsistency. In the table above, if William Brook changes his first name to James, each and every instance of the name William Brook must be updated to James Brook. This is a tedious process and is prone to errors when dealing with large databases.

The following subsections depict how these anomalies can be eliminated by normalizing a table in different normal forms.

4.1 First Normal Form (1NF)

The first normal form contains a set of fundamental rules for database design. The formal definition for the first normal form is as follows:

Definition: Let relation r have attributes A_1, ..., A_n of types T_1, ..., T_n respectively. Then r is in the first normal form if and only if, for all tuples t appearing in r, the value of attribute A_i in t is of type T_i for $i = 1, ..., n$ [30].

Let table 4.1 be relation *r* as reference. The attributes for this table are the column headers, namely, *VID*, *Make*, *Model*, ..., *Rent End Date*. All the tuples (rows) in this table do not violate the first normal form because for each tuple, *VID* is an integer, *Make*, *Model*, and *Driver* are text types and so on.

Additionally, the table contains only single-valued attributes. This means that the entries for each column in the table only contain a single value. Having more than one value is a violation of the first normal form. Moreover, the first normal form demands that each column should have a unique name. Having two or more columns with the same name is a violation of the first normal form.

Table 4.2 is a hypothetical record of customers that rented DVDs from a store. This table violates two rules of the first normal form. The first violation is that two fields have the same name. The second violation arrises in the *Rented Movies* field. Some customers have rented more than one movie. The *Rented Movies* field thus contains multiple values for some customers. Table 4.3 below shows how the violations can be resolved to ensure the relation is in the first normal form.

User No	**Name**	**Name**	**Rented Movies**
001	Fred	Calvin	Black Panther
002	Kevin	Daniel	Iron Man 3, Hereditary, The Conjuring
003	James	Harry	Incredibles 2, Deadpool 2
004	Katherine	Oliver	Sorry to bother you

Table 4.2: Unnormalized movie rental relation

User No	**First Name**	**Last Name**	**Rented Movies**
001	Fred	Calvin	Black Panther
002	Kevin	Daniel	Iron Man 3
003	James	Harry	Incredibles 2
004	Katherine	Oliver	Sorry to bother you
002	Kevin	Daniel	Hereditary
002	Kevin	Daniel	The Conjuring
003	James	Harry	Deadpool 2

Table 4.3: Movie Rental relation in 1NF

4.2 Second Normal Form (2NF)

The second normal form is the second major step in the normalization process of a database. It is an extension of the first normal form and has the purpose of better clarifying the database design and continuing to eliminate data redundancy within the existing relations in a database.

Before a formal definition for the second normal form is given, the following terms need to be defined:

- **Primary Key**: One or more attributes that are deemed most suitable to identify each record in a relationship. Most times, a primary key consists of a single attribute and is commonly a U_ID attribute (*unique identifier*) for each record. A primary key cannot have a null value

- **Foreign Key**: One or more attributes that are used to link two relationships together. A foreign key exists as a candidate (*or primary*) key in one relationship (*known as the parent table*) and exists as a foreign key in another relationship (*known as the dependent table*) [15]. The foreign key acts as a cross-reference between two relationships and is an essential element in eliminating data redundancy

- **Candidate Key**: The minimum number of attributes necessary to uniquely identify each record in a relationship. A relationship can have multiple candidate keys and they can take null values.

- **Composite Key**: A primary key that consists of two or more attributes

- **Super Key**: One or more attributes that are used to uniquely identify each record in a relationship. A super key is different from a candidate key since the attributes of a super key are not necessarily irreducible. Thus, a candidate key is indeed a subset of a super key (*Super key* $\subseteq$ *Candidate key*)

- **Subkey**: A subset of at least one key in a relationship
- **Partial Dependency**: When non-key attributes in a relation are functionally dependent on a subset of a candidate key
- **Functional Dependency (FD)**: For a relation r, functional dependency exists between two or more attributes if the value of one attribute uniquely determines the value of other attributes. For example if $X \rightarrow Y$, then Y is functionally dependent on X. X is called the determinant and Y is called the dependent. In table 4.1, the functional dependency $\{Make\} \rightarrow \{Model\}$ holds because each car model is functionally dependent on a specific make. Additionally, $\{V\_ID\} \rightarrow \{Make, Model\}$ holds as each make and model combination can be uniquely identified by the V_ID [30].
- **Trivial Functional Dependency**: The functional dependency $X \rightarrow Y$ is trivial if and only if there is no way the functional dependency can be violated [30].

Given the list of definitions above, the second normal form is defined as follows:

Definition: The relation r is in second normal form if and only if, for every nontrivial functional dependency $X \rightarrow Y$ that holds in R, at least one of the following is true: (a) X is a super key; (b) Y is a super key; (c) X is not a subkey [30].

In simpler terms, a relation is said to be in the second normal form if it is already in the first normal form and all partial dependencies are removed from the relation.

Consider the following relation on student major information:

S_ID	First Name	Last Name	Gender	Major	Fee ($)
01	Robert	Brown	Male	Physics	1000
02	Karen	Martinez	Female	Economics	700
03	Helen	Cranberry	Female	Neuroscience	1300
04	Louise	Lee	Female	Anthropology	650
05	Karen	Martinez	Female	Psychology	650
06	Jacob	Green	Male	Chemistry	1050
07	Robert	Brown	Male	Mathematics	900
08	Karen	Martinez	Female	Mathematics	900
09	Theodore	Putname	Male	Biology	850

Table 4.4: Student major table

Table 4.4 fulfills all the requirements of the first normal form. However, this table has lots of duplications and partial dependencies. For example, the *fee* attribute is partially dependent on *major* which is a non-key attribute ($\{Major\} \rightarrow \{Fee\}$). Additionally, *Gender* is partially dependent on *First Name* and *Last Name* ($\{FirstName, LastName\} \rightarrow \{Gender\}$). As the size of this table increases, more and more duplications will occur causing insertion, deletion and update anomalies. If the fee for a mathematics major increases, this information needs to be updated on all its occurrences in this database causing a potential update anomaly.

To put table 4.4 into the second normal form, information needs to be split into multiple tables. This eliminates partial dependencies in the separate tables. The first table that needs to be created holds student information. The second table links students to their respective majors and the third table holds course information.

S_ID	First Name	Last Name	Gender
01	Robert	Brown	Male
02	Karen	Martinez	Female
03	Helen	Cranberry	Female
04	Louise	Lee	Female
05	Jacob	Green	Male
06	Theodore	Putnam	Male

Table 4.5: Student information table

S_ID	Major 1	Major 2	Major 3
01	Physics	Mathematics	Null
02	Economics	Mathematics	Psychology
03	Neuroscience	Null	Null
04	Anthropology	Null	Null
05	Chemistry	Null	Null
06	Biology	Null	Null

Table 4.6: Student major table

Major	Fee ($)
Physics	1000
Economics	700
Neuroscience	1300
Anthropology	650
Psychology	650
Chemistry	1050
Mathematics	900
Biology	450

Table 4.7: Major information table

Tables 4.5, 4.6, and 4.7 all comply with the rules of the second normal form. If specific information about a student or a major needs to be updated, it can be done easily at one location, eliminating the risk of anomalies. Moreover, all three tables do not contain partial dependancies. Finally all data is stored such that there is no data duplication.

4.3 Third Normal Form (3NF)

The third normal form is the third step in the normalization process of a database. It is the stepping stone to achieving the Boyce-Codd normal form. The third normal form extends the second normal form to further eliminate anomalies and data redundancy. The following term is defined before giving a formal definition of the third normal form:

Transitive dependency: A non-key attribute that is functionally dependent on another non-key attribute in the same relation

The third normal form is defined as follows:

Definition: A relation R is in the third normal form if and only if, for every nontrivial functional dependency $X \rightarrow Y$ that holds in R, either (a) X is a super key or (b) Y is a subkey [30].

For a relation to be in third normal form, it must first fulfill the requirement for the second normal form. Additionally, it must also eliminate all transitive dependencies.

Tables 4.5, 4.6, and 4.7 in the previous subsection are all in the third normal form. However, if table 4.7 is expanded to contain department chairs, then transitive dependancies will exist in the relation. The following relation contains student major information including department chairs:

Major	Fee ($)	Chair_ID	Department Chair
Physics	1000	01	Dr. Erik
Economics	700	02	Dr. Jennifer
Neuroscience	1300	03	Dr. John
Anthropology	650	04	Dr. Pierce
Psychology	650	05	Dr. Timothy
Chemistry	1050	06	Dr. George
Mathematics	900	07	Dr. Radu
Biology	450	08	Dr. Deloris

Table 4.8: Major and department chair information table

In table 4.8 transitive dependancies exist in the *Chair_ID* and *Department Chair* attributes. *Department Chair* is functionally dependent on *Chair_ID* which is itself a non-key attribute. The functional dependency is represented as $\{Major\} \rightarrow \{Chair\_ID\} \rightarrow \{DepartmentChair\}$. Additionally $\{Major\} \rightarrow \{DepartmentChair\} \rightarrow \{Chair\_ID\}$ is also another valid transitive dependency. *Chair_ID* is functionally dependent on the non-key attribute *Department Chair* which is in turn dependent on *Major*. To eliminate this transitive dependency, table 4.8 needs to be split into two different tables as follows:

Major	Fee ($)	Chair_ID
Physics	1000	01
Economics	700	02
Neuroscience	1300	03
Anthropology	650	04
Physchology	650	05
Chemistry	1050	06
Mathematics	900	07
Biology	450	08

Table 4.9: Major information table

Chair_ID	Department Chair
01	Dr. Erik
02	Dr. Jennifer
03	Dr. John
04	Dr. Pierce
05	Dr. Timothy
06	Dr. George
07	Dr. Radu
08	Dr. Diloris

Table 4.10: Department chair table

Tables 4.9 and 4.10 are now in third normal form. All of the transitive dependancies have been eliminated.

4.4 Boyce-Codd Normal Form (BCNF)

The Boyce-Codd normal form (sometimes referred to as 3.5 normal form) is an advanced version of the third normal form. While the third normal form is usually adequate for most relational databases, data redundancy can still be reduced. BCNF extends the third normal form by ensuring that all data present in a relation provides a fact about the key, the whole key, and nothing but the key [30]. The Boyce-Codd normal form was developed by computer scientists Raymond Boyce and Edgar Codd in 1974 to address some anomalies that the third normal form was unable to fix.

The formal definition for the Boyce-Codd normal form is given as follows:

Definition: Relation R is in Boyce-Codd normal form (BCNF) if and only if, for every nontrivial functional dependency $X \rightarrow Y$ that holds in R, X is a super key [30].

This means that if Y happens to be a prime attribute, X cannot be a non-prime attribute. To better explain this concept, relation 4.11 is converted into BCNF.

Student_ID	Major	Graduation Year	Academic Advisor
001	Chemistry	2020	Dr. Clara
002	Math	2022	Dr. Mary
003	Sociology	2020	Dr. Pierce
002	Physics	2020	Dr. Johnston
004	Sociology	2021	Dr. Pierce

Table 4.11: Student major and advisor information

Relation 4.11 contains information on university students and their assigned academic advisors. This table is in the third normal form as it does not have any transitive dependancies. However, it is not in BCNF. This is because of the following functional dependancies:

- $\{Student\_ID, Advisor\} \rightarrow \{Major\}$
- $\{Major\} \rightarrow \{Advisor\}$

It is important to note here that *Student_ID* and *Advisor* are candidate keys, thus making them prime attributes. The second functional dependency shows that the prime attribute *Advisor* is functionally dependent on the non-prime attribute *Major*. Thus, relation 4.11 does not satisfy BCNF. The previous relation can be broken down into two smaller relations to satisfy BCNF as follows:

Student_ID	Graduation Year	Academic Advisor
001	2020	Dr. Clara
002	2022	Dr. Mary
003	2020	Dr. Pierce
002	2020	Dr. Johnston
004	2021	Dr. Pierce

Table 4.12: Student advisor and graduation date information

Major	Academic Advisor
Chemistry	Dr. Clara
Math	Dr. Mary
Sociology	Dr. Pierce
Physics	Dr. Johnston
Sociology	Dr. Pierce

Table 4.13: Available majors and advisor information

Relations 4.12 and 4.13 are broken down such that no prime attribute is functionally dependent on a non-prime attribute. The third column in relation 4.12 could have included student major information instead of academic advisors and the relations would still fulfill BCNF requirements. It is assumed that students are not able to have dual majors and no one student can have multiple advisors for simplicity. While the third normal form is usually sufficient for eliminating anomalies, the Boyce-Codd normal form is also used to further eliminate anomalies.

CHAPTER 5

Database Security

Database security is an important step in securing the massive amounts of personal and classified information that companies, governments, educational institutions, and healthcare providers collect on their users. With the rise in accessibility of the internet around the world, it is becoming more and more crucial to ensure information security. It is very important that the data stored in databases keep their integrity, do not get lost or misplaced, and are accessed only by authorized services and personnel. Database security comes in many different forms including the physical security of data centers, using software to prevent unauthorized intrusions, and implementing administrative controls in the different types of software interacting with the database systems. It is important to note that databases that are accessible on the world wide web need more stringent security infrastructures in place than databases that only provide local services to businesses without having the need to be connected on the internet.

Currently, the most popular security measure in the industry is the use of firewalls. Firewalls act as a barrier between the trusted internal network of an organization and the untrusted internet. Firewalls decide whether or not to allow incoming and outgoing network traffic between an organization's network and the wider internet network. Firewalls, however, are limited in securing a database

from internal violations in an organization or in cases where unauthorized users successfully penetrate the firewall and have access to the internal resources of an organization [27]. According to a U.S. Air Force study on computer crimes, the majority of database security breaches occur from insiders [27]. This demonstrates the need to adopt systems that ensure database security from an internal standpoint.

There are many different types of database information security control methodologies. This chapter focuses on a few of these practices while also looking at the concept of SQL injection attacks and their prevention techniques.

5.1 Access Control

Database access control is a security mechanism by which a database user is allowed access to only a subset of the database that the user can query. Unauthorized data access is often a major concern in the design of a database management system. This occurs when a user is able to obtain information that he/she is not entitled to access. The main objective of access control is to restrict access to database users based on their specific access privileges. The concept of access control is tightly associated with the concept of user authorization which determines whether a user is allowed access to specific types of data and whether or not the user can make modifications and deletions to the data. Additionally, the issue of confidentiality is also tightly associated with access control in the case of databases. User data confidentiality is the method of protecting user data from unauthorized access. There are three main types of database access control that will be covered in this section. These are discretionary access control *(DAC)*, mandatory access control *(MAC)* and role-based access control *(RBAC)*.

5.1.1 Discretionary Access Control

Discretionary access control is a type of access control by which access to data stored in a relational database is based on the user's identity and the authorization rules assigned to the user. DAC is used in many commercial database management systems due to its flexibility in granting users access to data [26]. DAC allows users to grant authorizations on a specific subset of data to other users. Users can be granted access rights to specific objects, data files, and records. Additionally, specific modes of access are assigned such as viewing, inserting and deleting access [39]. One important aspect of DAC is its authorization administration policy. The authorization administration policy is a policy that governs how a user is granted or revoked access to data. Some of the more common types of authorization administration policies include centralized administration and ownership administration [26]. Centralized administration is when only certain users *(with higher privilege access)* have the authority to grant or revoke permissions whereas ownership administration insists that only the owner of an object can grant or revoke a user's access to a specific object. A user is said to be the owner of the object if they created that object or if they have higher privileges than the creator of the object *(such as an administrator or any other official higher up in the privilege rank for a database system)*. Ownership administration offers additional features for administration delegation [26]. This is a system by which the owner of an object can delegate other users to be able to grant or revoke authorizations for the specific object.

The discretionary access control framework has a few drawbacks. The DAC framework enables object owners to control access authorizations to specific objects and this can lead to Trojan horse attack susceptibility. A Trojan horse is a type of malware designed to inflict harmful action on data stored. Users that have access to the database but are not authorized to view the contents of a specific object can

use Trojan horse attacks to copy the contents of that object on behalf of the object owner [18]. This takes place without the knowledge of the object owner but using the owner's access privileges. While there are different models proposed to protect the discretionary access control policy from Trojan horse attacks, these attacks are still the biggest drawback to using DAC.

5.1.2 Mandatory Access Control

Mandatory access control is another type of access control developed using a non-discretionary model. Unlike DAC, the MAC framework assigns access authorizations based on a user's information clearance. MAC policies are often defined and controlled by either the database administrator or the security policy administrator *(if a security policy administrator exists within the organization)*. This framework is often referred to as the most stringent form of access control. Due to this reason, MAC is the goto access control mechanism employed by the military, intelligence agencies and other government offices [25]. Each database user is classified based on an ordered set of access classes *(also known as labels)*. An access class is made up of two different components: a security level and a set of categories [26]. An example of security levels used by intelligence agencies are Top Secret *(TS)*, Secret *(S)*, Confidential *(C)*, and Unclassified *(U)*. The most dominant class is the Top Secret class and has access to the most sensitive data stored in a database. Users that have access authorization to these dominant classes are usually trusted within their respective organizations. The set of categories, on the other hand, refer to the different sectors of an organization that a specific user is classified under. In the case of the US military, the set of categories include departments such as NATO and the Nuclear Army.

5.1.3 Role-based Access Control

Role-based access control allows access to data based on the role of a user. This method of access control is a relatively new innovation and is currently highly sought after. A user's role within an organization represents the set of actions and responsibilities associated with the user [26]. Unlike MAC and RAC, access authorizations in this model are granted or revoked to specific roles instead of specific users. When a user needs to complete a task, the user is assigned a role and thus acquires the access authorization of the role [26]. The RBAC model works by authorizing users to play certain roles when they need to perform certain actions. This method of access control is efficient compared to MAC and DAC as all the roles are predefined and the only thing that needs to be done is to assign a user to a role depending on the action that needs to be performed. Additionally, the RBAC method consolidates all the different access authorizations on the database into organized roles making it easier to manage and modify if necessary. Figure 5.1 depicts three users that are assigned the same role *Role 1* and are able to perform different transactions on *Object 1* and *Object 2*.

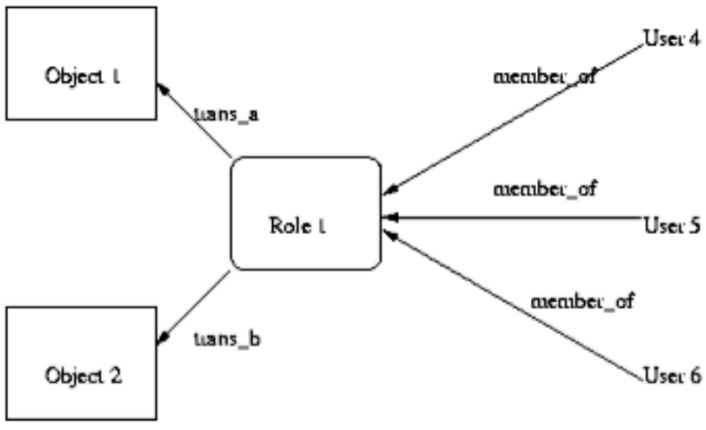


Figure 5.1: Role based authorization [25]

One additional security feature provided by RBAC is the separation of duty

constraints *(SoD)*. SoD constraints enforce that no one user has too many roles and thus has too many access authorizations. This is because if a user that has too many authorizations is compromised, that will highly expose the data stored in a database [26]. SoD constraints can either be static or dynamic. Static SoD constraints restrict the number of users that can be assigned to one role and also govern whether or not two different roles can have common users [26]. This fits into the 'separation of duties' principle which states that users should not have more rights than they need to perform a certain task. Dynamic SoD constraints restrict role access based on the user's interaction with their assigned role during previous and current sessions. A session in this context refers to the duration in which a user requires to complete one unit of work.

5.2 Encryption

Encryption is the method of converting plaintext into ciphertext by the use of an encryption algorithm. Encryption algorithms are implemented based on the practice of cryptography. The plaintext is scrambled using a mathematical algorithm, and only users with an encryption key are able to unscramble the text. Encryption is currently extensively being used in e-commerce, banking services, some secure messaging platforms, and many more industries. Database encryption is a technique used as an added layer of security for a database. Security measures such as access control and firewalls provide adequate security to a database but are unable to protect the data stored if an attacker bypasses their way into a database. As stated at the beginning of this chapter, the majority of database security breaches occur from insiders. Thus, it would be trivial for insiders such as system administrators and database administrators to have access to a database system. Database-level encryption protects data stored on a database by ensuring that only authorized users

are able to read the stored data. If database encryption is correctly implemented, data obtained by an attacker is meaningless without the proper key to decrypt it. Additionally, an attacker would not be able to modify the database by adding records because the information would be illegitimate when the database is decrypted for use [31].

While encryption is an effective way of protecting data, database encryption must satisfy specific constraints. The following list discusses a few of these constraints:

- The encryption algorithm used must be adequate enough to require an extremely high work factor *(the effort required to break into a crypto-system)*. In order to achieve this level, difficult mathematical problems are used in encryption algorithms.
- Encryption and decryption must be fast so as to not heavily impact the performance of the database system. If the process of encrypting and decrypting data is slow, this technique would introduce issues related to the speed and performance of the system.
- The size of the encrypted data is not significantly greater than the size of the data before encryption.
- The encryption algorithm used must be able to encrypt and decrypt single records *(single rows)* without regard to their physical and logical position in the database. A single record would be able to provide all data for a single entity and thus encryption must be implemented on the record-level.
- The encryption algorithm must be able to handle logical subschemas. Subschemas are different subsets of the schema for a database. Different users of the database might want to view different subschemas thus obtaining different subsets of data stored on the database. Decrypting different subschemas must present the stored data correctly.

- The database management system must be able to identify and properly handle records that are attempted to be created or updated using false encryption keys. An authorized user must be able to identify false records when decrypting records.

- The encryption mechanism must not demand that duplicate copies of data be created to allow subschema representations as this introduces storage, performance and integrity issues.

- The encryption scheme must allow data to be decrypted from incomplete records the same way it is decrypted from fully complete records [31].

This list is by no means an exhaustive list of constraints. It is, however, an accumulation of some of the most important requirements to consider when designing and implementing database encryption algorithms.

Protecting database encryption keys is as important as encrypting the database itself. Since cryptography relies on keys to encrypt and decrypt data, database encryption is only as good as the protection of the encryption keys [33]. Encryption key management is the process of administering and protecting cryptographic keys. Database administrators need to determine where to store the encryption keys and what access restrictions to impose on these keys. One method currently being used is to store the keys in a restricted database table and encrypt that table with a master key [33]. The master key gets stored on the database server. The disadvantage of this approach is that the database administrators and other administrators with privileged permissions are able to access this master key and use it to decrypt data stored on the database.

An alternate approach to database encryption is to use a hardware security module *(HSM), a hardware device designed to protect and store digital keys*, to store the master encryption key. Hardware security modules provide crypto-processing

functionalities that allow the encrypted keys stored in a server to be dynamically decrypted. These decrypted keys are then removed from server memory once the intended database operations are completed [33].

Another solution is to have a separate server that takes care of security-related tasks. In the security server, encryption keys are stored using the hardware security module. Additionally, the security server manages users, groups, privileges and encryption policies [33]. Figure 5.2 is a high-level diagram depicting the separated database and security servers. The database server includes a security module that is responsible for communicating with the security engine in the security server to provide the key to encrypt and decrypt data. The security server is also responsible for authenticating users, and ensuring that user privileges are being enforced.

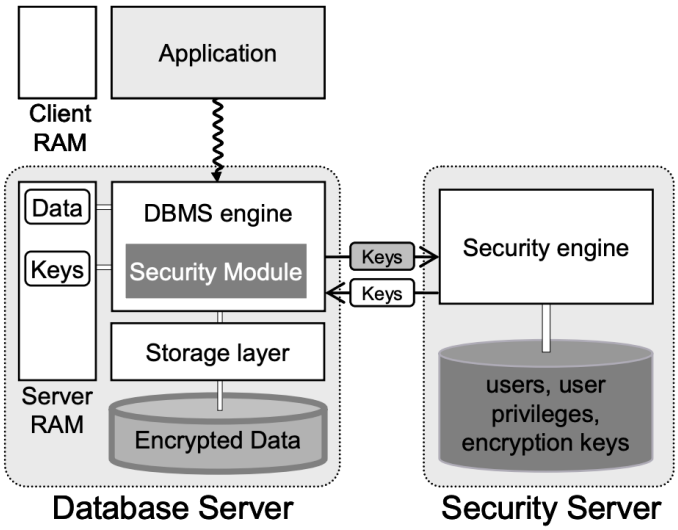


Figure 5.2: Security server implementation [33]

5.3 SQL Injection

SQL injection is a technique used in databases to access information that is not intended to be displayed. Malicious SQL queries are injected into a database

and can result in displaying sensitive and private data to an unauthorized user. SQL injections are also used to delete or manipulate data from a database. This vulnerability can have far-reaching consequences for organizations as these attacks threaten the integrity and privacy of data stored in a database. SQL injection attacks are most common in web applications that contain a database in the backend. The following user input demonstrates how an unauthorized user is able to inject malicious code into a database to display all user information from an unprotected database system If a user inputs *'105 OR 1=1'* as a UserId, the following SQL query is generated:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

[13]

This SQL query displays all the tuples in the *Users* table stored in the database because *'1=1'* will always return true. Depending on what fields are available in the *Users* table, the SQL injection described above can compromise private user information such as passwords, and home addresses.

The use of SQL injections to penetrate through database systems is very widespread and has been identified as the top ten most critical web application security risks in 2007 and 2010 [29]. In relation to web applications, there are three main avenues where SQL injection attacks occur. These are through cookies, server variables and through physical user input [29]. Cookies are packets of data that store state information on a user's machine. They are very useful as they allow websites to perform long-term user recognition. Cookie-based SQL injections *(also known as Cookie Poisoning)* occur when a hacker modifies a cookie by injecting malicious code. This is done by intercepting an HTTP request before it reaches a server and adding malicious SQL statements in its cookie field [12]. This enables the hacker to impersonate a valid user and gain information on behalf of the hacked user.

The most common type of SQL injection attack occurs through server variables. Server variables are all the variables that define an HTTP request such as the GET and POST submission methods. Additionally, HTTP headers and HTTP cookie parameters are also used as an avenue to inject malicious SQL queries. Malicious queries are entered into the client-side of the web application or specific requests containing malicious SQL queries are made to the server [29].

Physical user input is another avenue used in SQL injections. Websites that use user input directly in an SQL query are most vulnerable to this type of attack. There must be a specific mechanism in place that validates user input before passing it into an SQL query. It is important to note that malicious SQL code can also be injected through the form of barcodes *(if there are barcode input fields in the website)* and RFID tags [29].

SQL injection attacks can be classified into three different types, namely, in-band, inferential and out-of-band attacks [29]. In-band attacks refer to the type of attacks where the same communication channel is used to send an SQL injection attack and gather the results. This is the simplest form of SQL injection. Inferential attacks relate to methods of injection where the hacker does not directly receive data in response to the injection attempt. Instead, the hacker reconstructs the database structure based on the response of the web application after sending payloads [6]. Out-of-band attacks are uncommon and occur when the data is retrieved through a different channel than was used to send the attack. This method is used in cases where information retrieval in a web application is limited [29].

5.3.1 Prevention Techniques

Extensive research has been and is currently being done to develop different techniques to protect systems against SQL injection attacks. However, there is currently no one technique that fully immunizes systems from injection attacks.

Developers must use a variety of different tools and programming practices to make it as difficult as possible for an attacker to access a database system.

SQL prevention methods can be static, where developers work towards sealing up all vulnerabilities before deploying an application [29]. There are also dynamic techniques that work towards identifying and blocking SQL injection attempts while a particular application is providing service.

Parameterizing queries *(also known as a prepared statement with parameter binding)* is a static technique used to prevent SQL injection attacks by allowing a developer to specify the structure of SQL queries before passing parameters to those queries. This prevents unsanitary user input from modifying the query structure, thus ensuring that the intent of a query remains unchanged. For example, if an attacker inputs *'105 OR 1=1'* as a UserId, the parameterized query will look for a UserId that matches the entire string *'105 OR 1=1'* instead of exposing all the information stored in the *User* table.

Using an SQL DOM is another static technique that prevents SQL injection attacks. The SQL DOM gives the responsibility of database connectivity and database manipulation to a set of classes that are strongly correlated to a database schema. This allows the developer to generate SQL queries through the use of the classes instead of string manipulation [29]. This is important as it alters the query-building process from an unregulated one that uses string concatenation to a systematic one that uses a type-checked API [36]. The type-checked API applies certain security procedures such as input filtering. This prevents malicious SQL queries from affecting the system. The main drawback of using the SQL DOM is that it reduces the performance of the database system and requires developers to learn this new programming paradigm.

A dynamic approach that can be used is a machine learning-based anomaly SQL injection detection and prevention technique. This technique studies the behavior

of normal database operations and detects an attack by identifying uncommon behaviors. There are three steps that need to be carried out to use this technique. Initially, normal database behavior is defined and parameterized. Next, a model is created and is trained to detect abnormal behaviors. Finally, a process of prevention is defined so that a machine learning model can correctly deal with the identified abnormal behaviors [29]. Research has shown that clustering and outlier detection techniques are most effective when used in developing these machine learning algorithms. Additionally, support vector machines *(supervised machine learning classification algorithms)* are shown to have high confidence intervals in correctly classifying malicious SQL code that can be used for injection attempts.

5.4 Best Practices

Database security is a major topic and there are many different approaches to protecting data residing in a database. Extensive research is being done to advance the method of database security and ensure data integrity. The best practices for securing a database range from physical modes of security to securing systems through software and have more or less become standard in the current industry. Making sure that the physical machines hosting the database are kept in a secured and well-ventilated area is an important physical security measure. Physical damage done to the machine hosting a database can result in permanent loss of data. Additionally, the machines hosting the database need to be under supervision to prevent unauthorized entry.

It is advised to keep the database server separate from the web server. The database server should be kept behind a firewall to prevent an attacker from having access to data even if the attacker was to break into the web server. Separating the database server from the web server enables developers to implement different

types of security measures to the two different servers. Encrypting the database together with any backups that might be available is also equally important.

A web application firewall *(WAF)* is a type of firewall that secures web applications by filtering and monitoring HTTP traffic. Using this type of firewall not only protects the web application but also the underlying database in the backend from the likes of SQL injection attacks. When designing web applications, developers need to minimize their use of third-party apps. Third-party apps make it easy for developers to add functionality to their websites, but also come with the risk of the security of the database, especially when dealing with third-party apps that pull information from the database. Developers need to make sure that third-party apps being utilized are being updated and continuously supported to minimize the security risk they pose.

Database activity monitoring *(DAM)* is a technology that monitors all the activity being carried out on a particular database. This technology can be highly effective at protecting a database as it logs and provides reports on all the activity done on a database. Real-time alerts are sent out if the technology detects a threat or a violation of policy and authorized personnel can handle the issue immediately. DAM is only responsible for monitoring and alerting violations but does not handle the elimination of the threat. Thus, DAM needs to be used in conjunction with other security measures.

This chapter touched on some important concepts about database security and approaches that can be taken to minimize the risk of database security breaches. It is important to note that the approach to securing a database can vary depending on the nature of the data stored including the size of the data, and the level of confidentiality that is required by the data. While no one approach can guarantee a fully secure database system, employing multiple security measures as discussed in this chapter can provide adequate protection for a database system.

CHAPTER 6

User Interface Design

With the rise in popularity of web-based services, user interface design has become a major topic of interest amongst developers and web designers. User interface design has become a strategic component for many businesses and organizations as websites are often the first point of contact between a particular user and a business. If the business website or application is correctly designed it might motivate the user to easily perform their required task, thus increasing the user's satisfaction with the service they are receiving. This is a huge advantage to the success of the business itself.

Current user interface usability studies have focused more on identifying the type of user accessing the website and also the type of service that the user interface intends to provide. Identifying these key components is crucial as it enables the designer to focus on enabling the user to navigate through the website and complete tasks as quickly and efficiently as possible while preserving a decent aesthetic for the particular interface. This chapter explores some of the research being done regarding web-based user interfaces while explaining some of the fundamental design principles that web designers need to consider when designing for the web.

6.1 Web-Based User Interfaces

More and more businesses and organizations place great importance on their web presence. While providing access to millions of users worldwide, the web has allowed businesses to provide service to their customers remotely. Advancements made in the UI *(user interface)* and UX *(user experience)* of websites are major reasons the web has such a huge presence in business settings. Web-based user interface design focuses on the interaction between a user and the web application. Web designers are responsible for how a web application looks and the amount of convenience it provides for its users. A common trend in web design is the concept of simplicity. The famous French writer, Antonie de Saint-Exupery said, 'Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away [19]. The concept of simplicity in web user interface design relates closely to Antonie's quote. Simplicity refers to a user interface with a clean layout that does not contain unnecessary elements and has adequate white space. Simplicity is important as it improves legibility, and allows developers to guide their users to the important parts of the website. This can be done through the systematic use of specific colors and shapes [19]. Simplicity leads to more productivity and a better first impression of a website.

Figure 6.1 is the homepage for Craigslist and is an example of a poor user interface design. Craigslist is an advertisement website with many different segments including job vacancies and cars for sale. It is evident that simplicity has not been taken into consideration when designing this homepage as it is cluttered with many different links and lacks the visual appeal that customers are used to receiving in other web services. The number of different categories that are presented at the homepage can be unpleasant for the eye and makes the design look bloated and inconvenient to navigate.

Figure 6.1: Poor web user interface design [8]

6.2 User Interface Design Principles

In his book, 'GUI Bloopers 2.0' Jeff Johnson identifies the following nine basic user interface design principles. This section looks into the first three design principles identified in the list below [34]:

- Focus on the users and their tasks, not on the technology
- Consider function first, presentation later
- Design for responsiveness
- Don't distract users from their goals
- Try it out on users, then fix it!
- Conform to the users' view of the task
- Design for the common case
- Facilitate learning

– Deliver information, not just data

The first principle is identified as the main principle from which all the other principles are derived from. During the pre-design process, web designers need to understand their target users and what they would be using the website for. Designers need to recognize what specific problems they intend to solve through the use of their website. Additionally, the types of skills of their target audience need to be taken into consideration. Is the general user tech-savvy? How well does the user know about the product/service being offered by the app? These types of questions need to be investigated by identifying and surveying potential users whose demographics make them an attractive target market [34]. This allows designers to learn about the tasks that the user intends to complete using the website. Designers can then decide which tasks they want to support and design the specific steps that need to be taken to complete these tasks.

Before UI designers start designing the website's user interface, they need to focus on designing the functionality of the application. Designers need to define how their users will be interacting with the data and also decide which users can access and manipulate certain pages of the website. Developing a conceptual model is an important part of this process. Conceptual models are models that organize the knowledge generated by investigating the website's functionality [34]. A designer can model a representation of the website in a way that facilitates the successful completion of a user's intended task. It is important to note that the conceptual model is not a UI design. Rather it is a model that explains the functionality of the application and the concepts that users need to understand to use the website efficiently [34]. Once the conceptual model is complete, designers can move on to thinking about the specific UI component to include to help provide the intended functionality.

Research in the UI field has shown that a website's responsiveness is the single

most important factor in determining user satisfaction [34]. Responsiveness refers to how long it takes a website to react to user input. Responsiveness in this context should not be confused with performance because applications have two kinds of speed: real speed and perceived speed [28]. While responsiveness relates to perceived speed, performance is closely associated with real speed. A responsive website provides feedback to a user's request even if it needs time to process the request. Responsive designs include elements such as progress bars to let the user know how long a request will take to process. Research has shown that poor responsive designs often negatively impact worker satisfaction. Graph 6.2 below shows the effect of a website's response time on a user's productivity. The longer the response wait time, the lower the worker productivity. While performance is very important to user productivity, responsive designs can be used to communicate the process to the user, thus increasing user satisfaction.

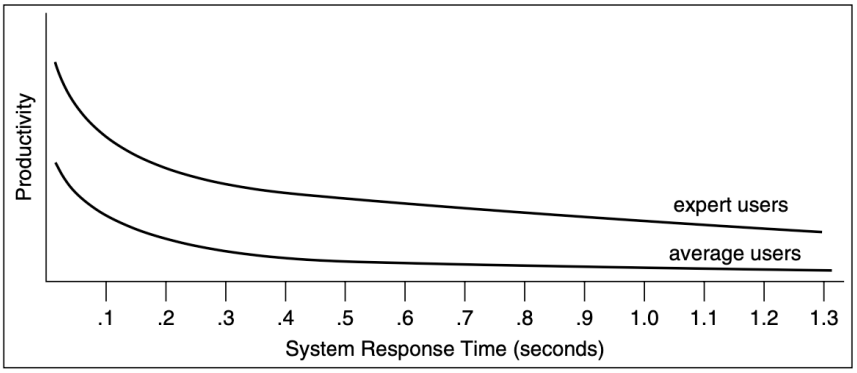


Figure 6.2: Effect of response time on user productivity [34]

CHAPTER 7

Software

The software section of this independent study implements an inventory management system for a local stationery shop located in Addis Ababa, Ethiopia. A web application is designed and developed to allow the stationery store employees to automate the process of inventory management.

This chapter focuses on presenting the underlying relational database model that is designed and implemented in the backend of the web application software. There is also a discussion of the user interface design as well as the web framework, templating engine, and database management system used to code the inventory management system.

7.1 Database management systems and web frameworks

Python is the programming language chosen for implementing the software. The main reason for using this language is due to my experience using Python, and because of the Django web framework that is only available for Python. Additionally, Python is a language that can achieve lots of functionality with few lines of code as compared to the likes of C++ and Java. As mentioned above, Django is used as the chosen web framework for this software. SQLite is used for the relational database

management system and the Django template language as a templating engine. The following subsections look into these different frameworks and how they are used in the implementation of the inventory management system web application.

7.1.1 Django

Django is an open-source python-based web framework that is an extremely popular choice for creating web applications. Django is very effective as it takes care of many of the nuisances that relate to web development. This allows the web developer to focus on creating a functional web-app without needing to reinvent the wheel [11]. Django comes with a plethora of built-in components that take care of security, user registration, model creation, and many more additional functionalities. Django is most famous for its versatility, security, and scalability.

Django uses object-relational mapping *(ORM)*. ORM is a technique used to convert data between two incompatible type systems. Django uses ORM to encapsulate database tables and database functionality through models and Python classes. This technique is very helpful for web developers as it saves time and allows them to write database queries using the object-orient paradigm of Python. Figure 7.1 shows how the order header relation is implemented using Djangos ORM functionality. Each relation is encapsulated in a class that uses Django's models API. Each database field is specified as a class attribute. Within each attribute, all the unique requirements of a database field are specified. The 'order_flag' attribute is a flag that employees use to identify if a specific order has been invoiced, *(the customer has paid for the order)*, voided *(the customer does not need the order anymore)*, or is in progress *(the customer has placed an order but has not yet paid for it)*. These choices appear as a drop-down list in the front-end. All the relations for the inventory management system are creating using this approach.

```python
class Order_header(models.Model):
    '''
    Database model for the order header relation
    '''
    # Choices for the order flag
    O = 'O'          # order
    I = 'I'          # invoice
    V = 'V'          # void
    flag_choices = (
        (O, 'O'),
        (I, 'I'),
        (V, 'V'),
    )

    order_date = models.DateField(auto_now=True)
    order_time = models.TimeField(auto_now=True)
    customer_name = models.ForeignKey(Customer, null=True, on_delete=models.SET_NULL)
    order_employee_id = models.ForeignKey(User, null=True, on_delete=models.SET_NULL)
    order_flag = models.CharField(max_length=1, choices=flag_choices, default=O)
```

Figure 7.1: Implementation of the order header relation

Django has a built-in class-based views API that developers can utilize to create their different views to handle GET, POST, PUT and DELETE HTTP methods. Class-based views are a great way to handle HTTP requests as it allows developers to respond to different types of HTTP requests with different class methods instead of using nested conditional statements that can become messy. Another advantage of using class-based views is their ability to extend their functionality through the use of Mixins. Mixins are an easy way to combine different behaviors and functionality from multiple parent classes. Figure 7.2 implements a create view for creating new employees to the database system. In this figure, the *EmployeeCreateView* inherits Djangos *CreateView* class that is responsible for displaying a form for creating an object and finally saving the object. This class also inherits two built-in Mixins, namely *LoginRequiredMixin* and *PermissionRequiredMixin*. The *LoginRequiredMixin* ensures that users are logged in before they can access the form to create new employee objects. The *PermissionRequiredMixin* handles user permissions and ensures that only users that have the permission to add employees can access the form. The *permission_required* variable only allows users with the *add_employee* permission to be able to access the employee creation form. As can be seen in this figure, Mixins are very useful and take care of much of the security aspect related to

creating forms, helping the developer focus more on adding functionality to the forms.

```python
class EmployeeCreateView(LoginRequiredMixin, PermissionRequiredMixin, CreateView):
    '''
    Renders a form to enable creation of an employee object
    '''
    form_class = EmployeeForm

    template_name = 'web_inv/Employee_form.html'
    permission_required = 'web_inv.add_employee'

    def get_context_data(self, *args, **kwargs):
        context = super().get_context_data(**kwargs)
        context['heading'] = 'New Employee'
        context['submit'] = 'Add'
        return context

    def form_valid(self, form):
        return super().form_valid(form)
```

Figure 7.2: Class-based view implementation for creating new employees

Django includes a Python module dedicated to handling URL configurations. When a new Django project is created, a module named 'urls.py' is automatically created. This module maps specific URL paths with Python views. Figure 7.3 depicts the URL path configurations for CRUD operations on customers. Updating and deleting customers requires specific customer primary keys. Django's URL configuration allows developers to dynamically handle this by using <*int:pk*> attribute. In this example, *CustomerListView*, *CustomerDetailView*, ... , *CustomerDeleteView* are all class-based views that are implemented in a separate module. Django has a method called *as_view()* that allows classes to be converted to views before passing them to a URL path configuration. The *name* variable for each of the different paths specify the name of the page that each view is routed to.

```python
# Customer URL path
path('customer/', CustomerListView.as_view(), name='customer-list'),
path('customer/<int:pk>/', CustomerDetailView.as_view(), name='customer-detail'),
path('customer/new/', CustomerCreateView.as_view(), name='create-customer'),
path('customer/<int:pk>/update/', CustomerUpdateView.as_view(), name='update-customer'),
path('customer/<int:pk>/delete/', CustomerDeleteView.as_view(), name='delete-customer'),
```

Figure 7.3: Mapping urls with class-based views

Another powerful tool that Django provides is its admin site. Django admin is a web-based approach to conduct different administrative functions. The admin interface allows authorized users to handle other users, models, and group permissions just to name a few of its functionalities. The admin page can be accessed by administrators to add or remove permissions and organize users into different groups to allow users to inherit group permissions. This approach saves time when developing the web-app as it allows developers to quickly test their models without having to create views for their models. Figure 7.4 shows the admin page for the inventory management system that is implemented. All of the models, as well as the different users and groups created, can easily be accessed and manipulated using this interface. Django admin can only be accessed by select users that have *staff* status enabled or users that are created as a *superuser*. When this inventory management system is deployed, only the web administrator will be able to access this page.

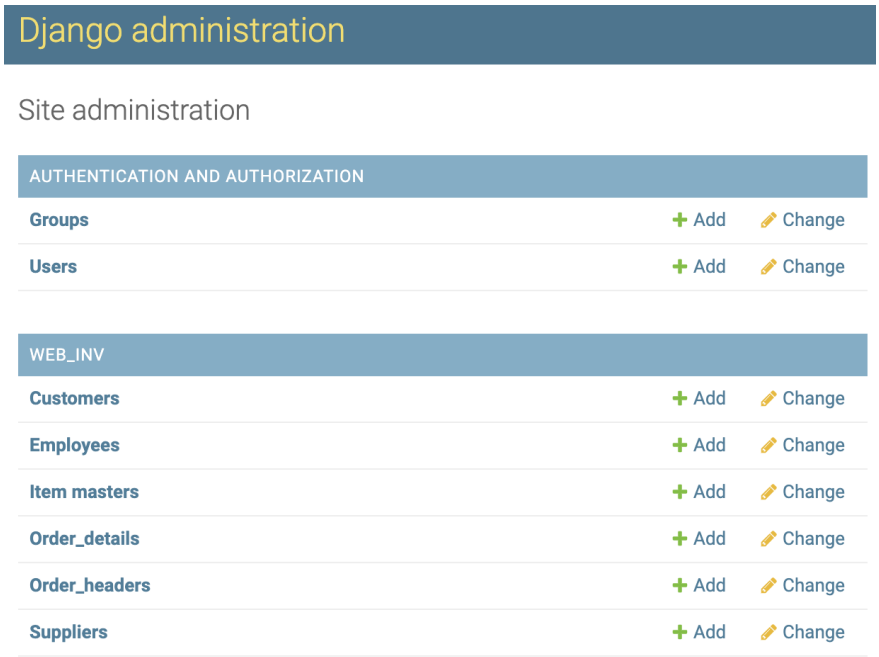


Figure 7.4: Django admin page

Unlike many other web frameworks, Django includes its own templating engine. Similar to the popular Jinja2 templating engine, the Django template language *DTL* allows developers to generate HTML content dynamically. Django allows users to switch out its default templating engine with other engines such as Jinja2 if need be, but the built-in engine was sufficient for the scope of the web-app created in this project. DTL is a powerful yet simple engine that provides a wide range of features. Figures 7.5 and 7.6 show some of the instances in which DTL is used. Figure 7.5 is the HTML code for the signup page. This module extends *master.html* which includes the base structure for the web-app. Crispy forms is a Django application that allows developers to render elegant Django forms in addition to providing full control over the different fields in the form. Variables are surrounded by double curly brackets. In figure 7.6, *{{ form | crispy }}* is a variable that contains the signup form. The vertical line used is a filter that tells Django to use crispy forms to render the signup form. Tags *({% ... %})* are used for arbitrary logic. The for-loop in the code loops through all the fields in the form and if a specific field has a help text, it is displayed in gray color. DTL syntax is very similar to python except indentation is not required and the *for* and *if* blocks need to be closed using *endfor* and *endif*.

Django used cross-site request forgery protection *(CSRF protection)* as a security measure when rendering forms. CSRF tokens protect websites from cross-site request forgery attacks. CSRF attacks occur when a logged-in user visits another website that contains a malicious link or Javascript that performs an unauthorized action on the web-app using the credentials of the logged-in user [9]. To protect against CSRF attacks, GET, POST, PUT and DELETE methods need to be protected using a CSRF token. CSRF tokens are unique and unpredictable tokens that are automatically generated by the server-side application. When the client makes an HTTP request, this token is sent to the client. The server-side application then validates all requests made by the client using this token, and any request that does

not include this unique token is rejected [16]. To use CSRF protection, the tag *{% csrf_token %}* is used.

```
{% extends 'web_inv/master.html' %}
{% load crispy_forms_tags %}

{% block content %}

  <h1 class="display-4" align="center">Sign Up</h1>

  <form  method="POST">
    {%  csrf_token %}
    {{ form|crispy }}

    {% for field in form %}
      {% if field.help_text %}
          <small style="color: grey">{{ field.help_text }}</small>
      {% endif %}
    {% endfor %}

    <div>
    <button type="submit" class="btn btn-primary">Sign up</button>
    </div>

  </form>

{% endblock content %}
```

Figure 7.5: Sign up page

In addition to using the *PermissionRequiredMixin* to ensure user permissions, additional filter tags can be used to hide links that are not supposed to be active for specific users based on their permissions. For example, in figure 7.6 *{% perms.web_inv.add_employee %}* ensures that the button that provides the form to add new employees is hidden from users that do not have the *add_employee* permission.

```
<form>
  {% if perms.web_inv.add_employee %}
    <button formaction="{% url 'create-employee' %}" class="btn btn-outline-primary"> + Add New Employee </button>
  {% endif %}
</form>
```

Figure 7.6: Permissions using Django template language

7.1.2 SQLite

SQLite is used to handle the database side of the web-application. SQLite is one of the many relational database management systems that are widely used. Unlike other database management systems such as MySQL and PostgreSQL, SQLite is not a client-server database engine and thus does not require a server to run. SQLite has what is known as a 'server-less architecture' and uses the database files stored on the disk to read and write to the database [2]. Due to this reason, SQLite is lightweight and is very easy to set up and operate. Figure 7.7 is a simplified diagram of the client-server architecture employed by relational database management systems like MySQL. In contrast, SQLite uses the server-less architecture as seen in figure 7.8.

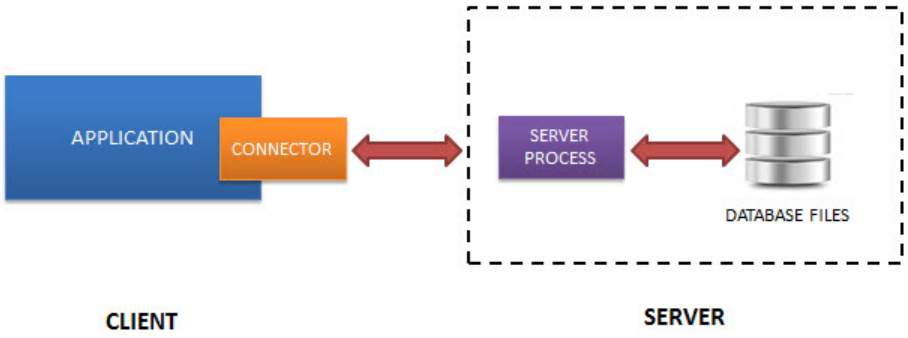


Figure 7.7: Client server architecture [2]

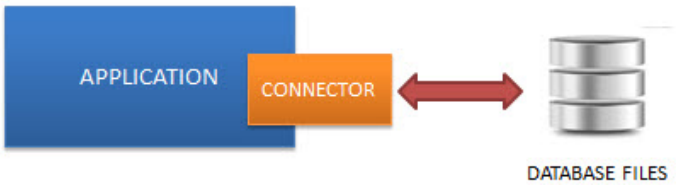


Figure 7.8: Server-less architecture of SQLite [2]

Depending on its specific use, SQLite has a few disadvantages. While SQLite emphasizes reliability, simplicity, and efficiency, it is mainly used in situations where

there is low HTTP request traffic. Additionally, SQLite is not as powerful as other database management systems and is not scalable if an organization decides to expand. In the case of the inventory management system implemented, SQLite is sufficient as the stationery store that will use this system is very small, and this system is to be used solely internally in the stationery store. Additionally, Django makes it really simple to switch database management systems if need be. Figure 7.9 shows the section that needs to be switched in the Django settings module if a different database management system deems more appropriate.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Figure 7.9: Django settings for the DBMS used

7.2 User interface design

User interface design is an important component of the inventory management system web application. As discussed in chapter 6, user interface design has become a strategic component for many businesses. While this web application is meant for in-house use and the user interface design is not as important, certain elements have been put in place to increase user satisfaction when interacting with the system. This section looks into some of these components and the front-end frameworks that are used to design the user interface.

7.2.1 Bootstrap 4

Bootstrap is one of the most popular open-source tools that are used to create responsive web applications. Responsive web applications are able to automatically adjust to fit different screen sizes. This enables the user to view all the contents of a website in a comfortable and organized manner on any screen size. With the dramatic rise in popularity of web browsing on smartphones and tablets, designing responsive web applications is crucial as it allows a wide range of users to seamlessly access content provided by a web application. Bootstrap offers a wide variety of components such as buttons, forms, and navigation bars. Figure 7.10 is a Bootstrap form that is used to enter a new stationery item into the database. The Bootstrap template used offers a clean and modern user interface.

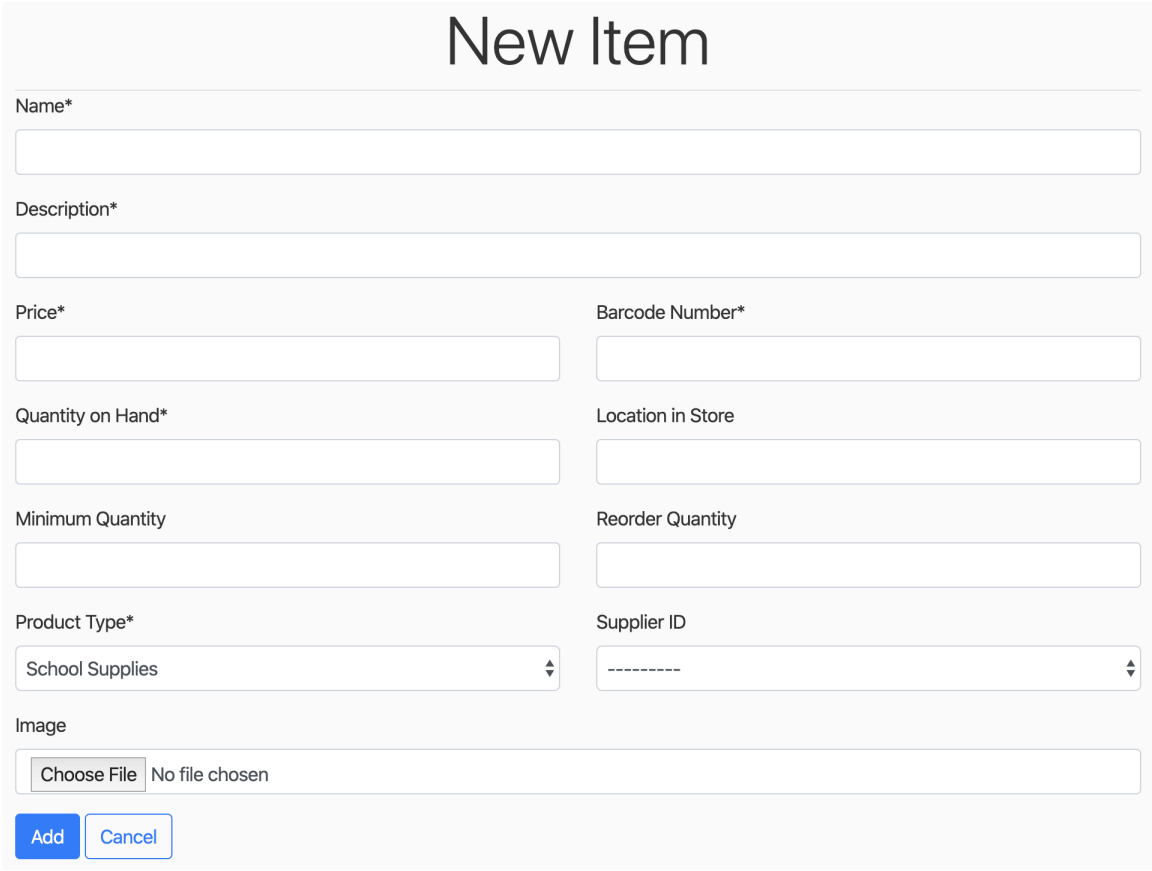


Figure 7.10: Form to create a new item

7.2.2 Order Page

The order page allows authorized employees to create and process orders. To create an order, a customer is selected from the customer table and an order header is created for the customer. One order can include one or many different items with different quantities. Figure 7.11 depicts the order page for a customer that is buying two different items of different quantities. By clicking on the 'Add Item To Order' button, an employee can add additional items to the order. This page dynamically updates the subtotal, VAT *(value-added tax)*, and total as additional items are added or removed. By default the order flag is set to 'O' *(meaning the order is in progress and not yet completed)*. Once the customer is ready to pay for the order, the 'Invoice Order' button is clicked and an invoice is generated. This automatically updates the order flag to 'I' *(meaning the order has been invoiced)*. If the customer decides not to complete the order, the 'Void Order' button is clicked to successfully void the order.

Additional checks are put in place to make sure that the quantity ordered is not more than what is available in the store. If a quantity greater than is available is requested, an error message lets the employee know that the requested amount is more than the available amount and the requested amount is capped at the maximum available quantity for the item in-store. Figure 7.12 shows the error message that is generated. Visual cues are generated using Django's messages framework. The messages framework provides functionality for creating flash messages that disappear after a certain time. The messages have different tags such as 'success', 'warning' and 'error' that correspond to the type of flash message intended to be displayed.

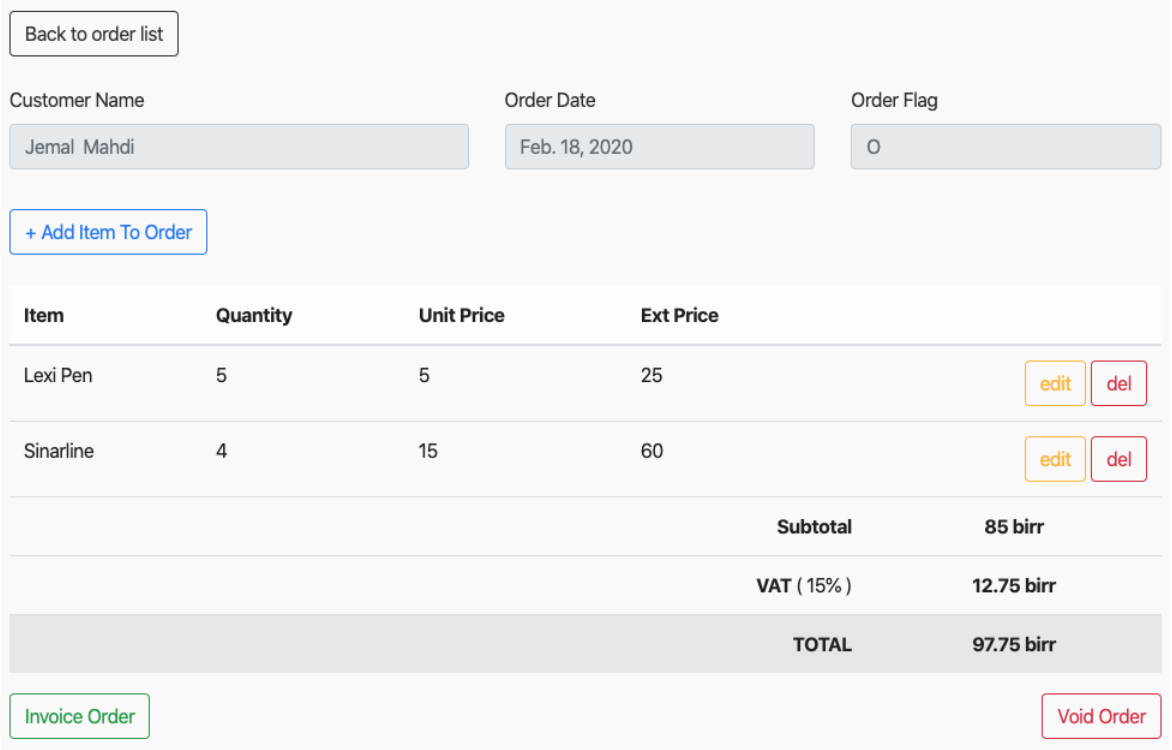


Figure 7.11: Order page view

Figure 7.12: Visual queues to depict error

7.3 Reporting and Invoicing

Reporting and invoicing are a few of the features that the inventory management system includes. As mentioned in the previous section, an invoice is generated once the customer is ready to pay for the order. Figure 7.13 shows a sample order invoice. The customer information is fetched from the customer table through the use of foreign keys. A print button is also available on this page to allow employees to easily print the invoice.

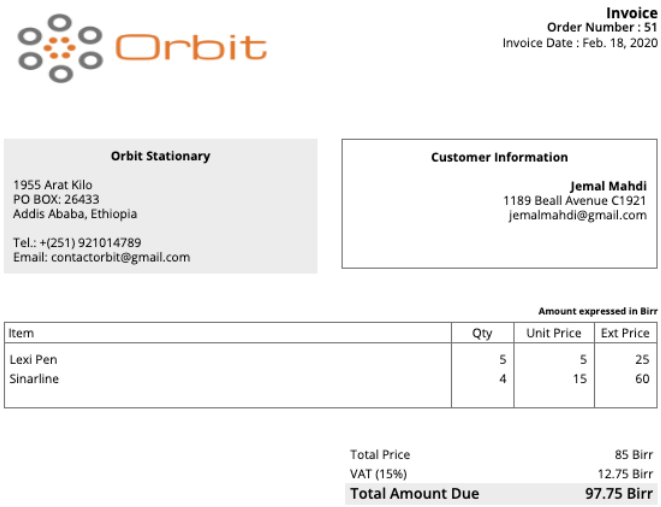


Figure 7.13: Invoice for order

The reporting feature generates an end-of-day report for the orders processed throughout the day. This includes a sales report that presents month-to-date sales, year-to-date sales, and sales made during the current day. It also includes a low stock report. A list of items with low inventory is generated to remind authorized employees to order additional products. When an item is created, the quantity on hand and the minimum quantity are logged together with the item detail. If the quantity on hand is lower than the minimum quantity, the item is flagged and displayed as a low stock item. Figure 7.14 presents an example end-of-day report generated by an employee.

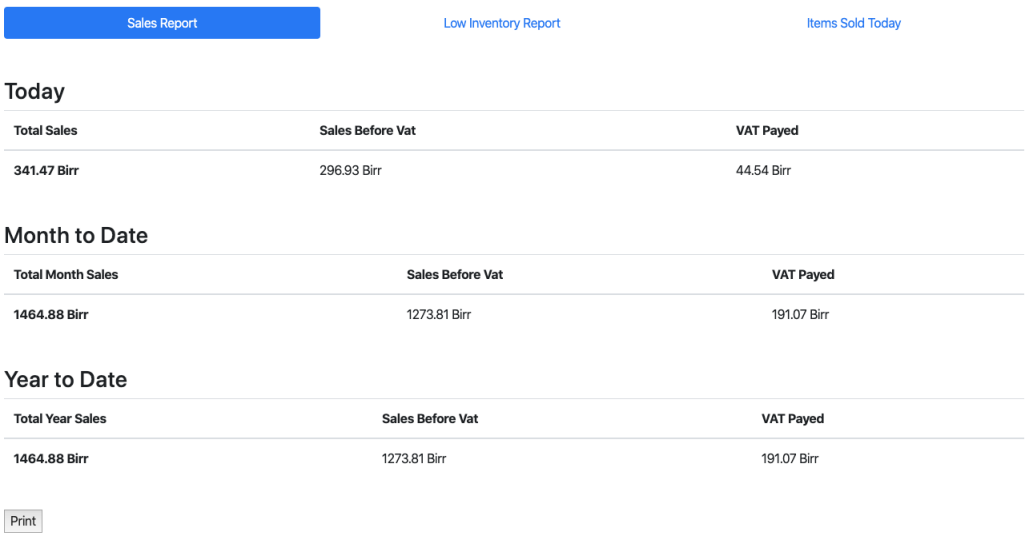


Figure 7.14: End of day report

7.4 User Management

User management is an important component of this inventory management system software. Django handles most of the user authentication, user groups, and user permissions to perform certain tasks. Django allows the creation and manipulation of user objects. The web administrator creates user objects for all employees authorized to use the inventory management system. This allows the user to be able to login to the system. Additionally, each user is assigned to a user group that has predefined permissions. For example, an accountant will have their own user object and will be assigned to an accountant group. The accountant group is assigned specific permissions for viewing, editing and deleting different forms in the system. The web administrator handles the creation of all user objects including user groups and permissions through Django's admin site. Figure 7.4 in section 7.1.1 shows the admin page that web administrator uses to assign users to groups and assign group permissions.

7.5 Inventory Management System ERD

An introduction to entity-relationship diagrams is presented in section 2.2.1.1. In this section, an entity-relationship diagram for the inventory management database is provided as seen in figure 7.15. Cardinality and ordinality are presented using the information engineering style. The relational database for the web application is built using this underlying structure.

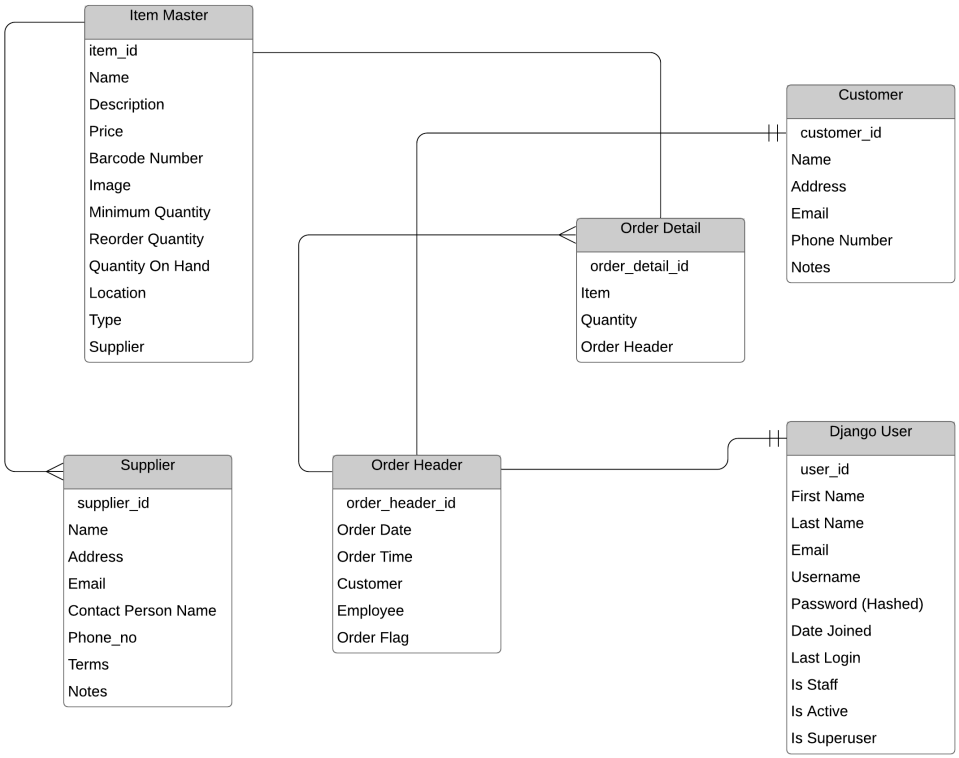


Figure 7.15: Entity relationship diagram for inventory management database

'Django User' is a built-in Django relation that handles user objects. The 'Order Header' relation includes the name of the employee that created the order and thus uses a foreign key to reference the 'Django User' relation. The relation depicted by the diagram shows that one 'Order Header' can be created by one and only one 'Django User'. 'Order Header' and 'Order Detail' has a one-to-many relationship. When a new 'Order Header' object is created, it initially has no items added to the

order. The employee can then add items to the order by creating new 'Order Detail' objects that are assigned to the 'Order Header'.

'Django User' is related to Django's built-in user groups and user permissions relations but the relationship is not depicted in figure 7.15 as Django handles these components by default. While Django allows the creation of custom user and group models, the default functionality that Django provides out the box is sufficient for the requirements of this inventory management system.

CHAPTER 8

Conclusion

This paper studies different types of databases and database management systems. It introduces and discusses theoretical aspects of the relational database model including database architecture and normal forms. Designing a database model for an organization is challenging because of all the different components that need to be put in place to enable the system to provide efficient service. Database developers first need to determine the purpose and context of the database being created. This allows them to organize the data received into logical structures. This is the most important and challenging part of the database design process as it lays the foundational framework for the storage, manipulation, and retrieval of data stored in the database. This paper extensively discusses normal forms, which is an important concept in the design of data tables and data relationships. Eliminating data redundancy plays a key role in ensuring data consistency, data integrity, query execution efficiency, and in upholding high performance. With the rise of the cloud computing industry, tech giants are offering cloud database solutions for businesses, organizations, and governments from all around the world. This is a very advantageous solution for clients of all sectors since the burden of data storage, scalability, performance, and security is taken care of by the cloud database providers.

Our interconnected world raises big threats to the vast amounts of personal data stored in databases. Database security has been and will continue to be one of the major challenges database systems face as hackers constantly develop new methods to breach existing security systems. This paper investigates database security threats and provides solutions and best practices for keeping the stored data as secure as possible from external and internal attacks. With the current rise in interest for the Internet of Things *(IOT)*, more and more personal data will be stored and thus the need for stronger database security mechanisms is incumbent upon the database designer. Additionally, efficiently accommodating the large volumes of unstructured data generated from a wide variety of IoT sources pose challenges to database developers as they need to adapt existing systems to handle this type of data.

The software portion of this independent study implements an inventory management software system for a small stationery store in Ethiopia. The software is a web application that allows different types of employees to access the system to store, edit and retrieve data from the database. A web user interface is built to provide employees with a clean and organized structure to enable them to efficiently carry out daily tasks. While the system provides basic functionality to automate the management of inventory, it still has lots of potential for future improvement as will be discussed in the upcoming section.

8.1 Limitations and Future Improvements

During the process of designing and implementing the inventory management system software, various limitations have been identified and methods for future improvement have been considered. The software system created is a simplistic but functional model for properly and efficiently managing inventory. One major

limitation of this software is its reporting feature. Currently, the reporting feature is meant to be run at the end of the day to show sales statistics for the current day. It also lists the items that are low in stock. Future improvements to this area include the option to generate sales statistics for specific time periods and improve the reporting feature by adding dynamic graphs and charts to help the client visualize trends in sales and inventory. The software currently has month-to-date and year-to-date sales, but it would be helpful to give the user more control with viewing sales statistics for user specified time periods.

Another major area for future improvement relates to the concept of concurrency control. Currently, the stationary business is small enough that concurrent manipulation of the same data is unlikely to happen at the same time. However, if this does happen, there are currently no concurrency control mechanisms to ensure the proper handling of concurrent access and manipulation of data. Additionally, concurrency control will be necessary when the business expands in the near future. Development in this area includes the implementation of a locking mechanism in the database management system. Lock-based protocols help synchronize access to the database items by concurrent transactions. When a specific transaction is being made to data stored in a specific part of the database, the transaction obtains a lock on the object before beginning its operation. This prevents any other transaction from manipulating the object. Once the operation is complete, the data is unlocked for future transactions [10].

The user interface for the system also has some limitations. Currently, a user needs to visit more pages to create a full order than is necessary. For example, figure 7.11 shows the user interface for the order page. When the 'Add Item To Order' button is clicked, the user is taken to another page to pick the item. This process can be eliminated by the use of inline formsets that allow a small form to be generated

on the same page rather than switch between different pages. This increases the speed by which orders are processed.

Finally, another limitation of this software is the lack of alerting features. Currently, the only method to check for low stock is by running the end of day report. However, a more efficient method is to alert the proper employee either through email or by implementing a notification system on the web application. This method allows users to immediately be notified in cases of low stock so low inventory can be quickly be restocked for items in high demand.

With the addition of the features described above, the inventory management system that is implemented has the capacity to fully replace the manual method of inventory management employed by the business today. The developed system requires constant technical support to scale the system as the stationery business grows and requirements change. Inventory management is a major concern for the stationery business and migrating to an automated approach can help the business identify its strengths and weaknesses to serve its customers more efficiently, thus generating more revenues and customer satisfaction.

References

1. Raima. URL https://raima.com/network-model-vs-relational-model/. viii, 15, 16

2. What is sqlite? URL http://www.sqlitetutorial.net/what-is-sqlite/. viii, 61

3. Entity relationship diagram. URL https://www.smartdraw.com/entity-relationship-diagram/. viii, 10, 11

4. Understanding dbms architecture. URL https://www.studytonight.com/dbms/architecture-of-database.php. viii, 19, 20, 21

5. Hierarchical database management systems. URL http://dbconcepts.generalconnection.com/topic/rdbms/what-is/30-hierarchical-database-management-systems.html. viii, 11, 12

6. Types of sql injection? URL https://www.acunetix.com/websitesecurity/sql-injection2/. 45

7. What is a relational database management system? URL https://www.codecademy.com/articles/what-is-rdbms-sql. 9

8. washington, dc jobs, apartments, for sale, services, community, and events. URL https://washingtondc.craigslist.org/. viii, 51

9. Cross site request forgery protectiondocumentation. URL https://docs.djangoproject.com/en/3.0/ref/csrf/. 59

10. Dbms concurrency control: Two phase, timestamp, lock-based protocol. URL https://www.guru99.com/dbms-concurrency-control.html. 72

11. Django introduction. URL https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction. 55

12. Sql injection. URL https://owasp.org/www-community/attacks/SQL_Injection. 44

13. URL https://www.w3schools.com/sql/sql_injection.asp. 44

14. Architecture of database. URL https://www.tutorialcup.com/dbms/architecture-of-database.htm. 20

15. Sql foreign key constraint. URL https://www.w3schools.com/sql/sql_foreignkey.asp. 27

16. Csrf tokens. URL https://portswigger.net/web-security/csrf/tokens. 60

17. Dbms architecture - javatpoint. URL https://www.javatpoint.com/dbms-architecture. 19

18. *A guide to understanding discretionary access control in trusted systems*. National Computer Security Center, 1987. 38

19. The importance of simplicity in web design, Jan 2013. URL http://fearlessflyer.com/the-importance-of-simplicity-in-web-design/. 50

20. May 2013. URL https://www.youtube.com/watch?v=KlHvRKSH4pk. viii, 23

21. What is nosql?, 2017. URL https://aws.amazon.com/nosql/. 16, 17

22. Nosql: database for storage and retrieval of data in cloud, 2017. URL https://aws.amazon.com/nosql/graph/. viii, 17

23. Nov 2017. URL https://www.youtube.com/watch?v=x2udY8IBXQ4. viii, 8

24. Single-tier vs. multi-tier architecture: Choosing the right bitnami package, Sep 2019. URL https://docs.bitnami.com/google-templates/singletier-vs-multitier/. 18, 19

25. Ryan Ausanka-Crues and Harvey S. Mudd. Methods for access control : Advances and limitations. 2006. viii, 38, 39

26. Elisa Bertino and Ravi Sandhu. Database security - concepts, approaches, and challenges. *CERIAS Tech Report*, 2(1), 2005. URL https://www.cerias.purdue.edu/assets/pdf/bibtex_archive/2005-99.pdf. 37, 38, 39, 40

27. Elisa Bertino, Sushil Jajodia, and Pierangela Samarati. Database security: Research and practice, May 1995. URL https://www.sciencedirect.com/science/article/pii/0306437995000294. 36

28. Peter Bickford. *Interface design: the art of developing easy-to-use software*. AP Professional, 1997. 53

29. Roshni Chandrashekhar, Manoj Mardithaya, Santhi Thilagam, and Dipankar Saha. Sql injection attack mechanisms and prevention techniques, 2012. URL https://link.springer.com/chapter/10.1007/978-3-642-29280-4_61#Bib1. 44, 45, 46, 47

30. Christopher Date. *Database design and relational theory: normal forms all that jazz*. OReilly, 2013. 25, 28, 31, 33

31. George I Davida, David L Wells, and John B Kam. A database encryption system with subkeys, Jun 1981. URL https://dl.acm.org/doi/10.1145/319566.319580. 41, 42

32. Ram Elmasri and Shamkant B. Navathe. *Fundamentals of database systems*. Pearson-Addison-Wesley, 2011. viii, 4, 5, 6, 7

33. Yanli Guo and Luc Bouganim. Database encryption, Sep 2011. URL https://hal.archives-ouvertes.fr/hal-00623915. viii, 42, 43

34. Jeff Johnson. *GUI bloopers 2.0: common user interface design donts and dos*. Elsevier/Morgan Kaufmann Publishers, 2008. viii, 51, 52, 53

35. Atul Kahate. *Introduction to database management systems*. Pearson Education (Singapore), 2004. 14

36. Stephanos Mavromoustakos, Aakash Patel, Kinjal Chaudhary, Parth Chokshi, and Shaili Patel. Causes and prevention of sql injection attacks in web applications. pages 55–59, 12 2016. doi: 10.1145/3026724.3026742. 46

37. ABM Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics - classification, characteristics and comparison. *International Journal of Database Theory and Application*, 6. URL https://arxiv.org/abs/1307.0191v1. 16

38. Arjun Panwar. Types of database management systems, Nov 2011. URL https://www.c-sharpcorner.com/UploadFile/65fc13/types-of-database-management-systems/. 12

39. S. K. Singh. *Database systems: concepts, design and applications*. Dorling Kindersley (India), 2009. URL https://learning.oreilly.com/library/view/database-systems-concepts/9788177585674/. 37

40. Vignesh Sivabalan. What is an object-oriented database? URL https://study.com/academy/lesson/what-is-an-object-oriented-database.html. viii, 13, 14

41. Stephen Watts. N-tier architecture: Tier 2, tier 3, and multi-tier explained, Jul 2017. URL https://www.bmc.com/blogs/n-tier-architecture-tier-2-tier-3-and-multi-tier-explained/. 22