# South Dakota School of Mines & Technology

# Database Management Systems, Spring 2022

CSC 484-M01

Final Project Report

Mathew Clutter

Adam Kraus

**1. INTRODUCTION**

This document provides a detailed report on CSC 484 Final Project. Mathew Clutter and Adam Kraus undertook the project.

Mathew Clutter and Adam Kraus worked together heavily to complete this project together. These students worked together to draft the initial requirements document, generate the ER diagrams describing the database setup, and ensure that the resulting tables are properly normalized. While many of the required tasks for this project were completed together, each individual student also worked on some tasks individually. Adam worked to seed the database and start the Python application, and Mathew focused on creating this report and documentation. While each student worked individually on some tasks, this project was a collaborative undertaking.

This application is going to be used for a warehouse management system. It could be used for managing products, orders, and other applicable information. This project consists of a total of six tables. These tables include customers, products, orders, order items, employees, and injury reports. The tables are related according to the following ER diagrams:

Diagram

Description automatically generated

Diagram

Description automatically generated

The first of these diagrams offers a higher-level overview of each table and how it relates to other tables, while the second diagram shows all the connections, along with the primary and foreign keys in each table. The second diagram was generated by the MySQL Workbench application, from the database schema definition. For more information on the layout and format of this database application, please see the initial project proposal document.[[1]](#footnote-2)

All questions, and complaints, can be directed to these individuals.

Mathew Clutter: mathew.clutter@mines.sdsmt.edu

Adam Kraus: adam.kraus@mines.sdsmt.edu

**2. FUNCTIONAL DEPENDENCIES**

Before the process of normalization could begin, our team worked to identify the functional dependencies present within the set of tables proposed.

Starting with the customers table, the first name, last name, email, phone number, and address are all functionally dependent upon the customer ID. For each customer ID, there is only one value of the set containing the first name, last name, email, phone number and address fields. Thus, if the customer ID is known, the values for all the other fields can be determined. Because of this relationship, the customer ID was selected to be the primary key of the customer’s table.

Next, in the employee table, the employees first name, last name, position, phone number, address, salary, and their current employment status can all be determined by the employee ID. The employee ID is the only attribute that can be used to uniquely identify any tuple in the relation. Thus, the employee ID was chosen to be the primary key of the employee table.

Within the injury report table, the employee ID, injury date, and description can all be determined from the injury ID. The injury ID is the only attribute that can uniquely identify the employee ID, injury date, and description within this table. Thus, the employee ID, injury date, and description are functionally dependent upon the injury ID within this table. Thus, the injury ID was chosen as the primary key within this table.

For the products table, the product name, price, and number remaining are all functionally dependent upon the product ID. The product ID is the only attribute that can uniquely identify the combination of product name, price and number remaining. Thus, the product ID was chosen as the primary key of this relation.

For the orders table, the customer ID, employee ID, placed on date, fulfilled on date, and notes can only be uniquely identified by the order ID. Thus, the customer ID was chosen as the primary key of the orders relation.

Finally, for the order items table, the number fulfilled, and number ordered can be uniquely determined by the set containing the order ID and product ID. As it takes a combination of the order ID and product ID to uniquely identify the number fulfilled and number ordered, the number fulfilled, and number ordered are functionally dependent upon the combination of the order and product IDs. Because of this relationship, the order ID and the product ID were chosen together to form a composite primary key.

**3. NORMALIZATION**

In order for the tables to be in third normal form (3NF), they must fulfill the requirements to be in first normal form, and second normal form, along with the additional requirements to fulfill third normal form. To be in first normal form, the intersection of each row and column must contain one, and only one value. For second normal form, first normal form must be achieved, along with every non-primary-key attribute being fully functionally dependent on the primary key. Finally, for third normal form, the relation must be in first and second normal form, along with there being no non-primary-key attribute transitively dependent upon the primary key. As constructed, our tables fulfill the requirement of being in third normal form.

Each table is in first normal form, as every table only contains atomic values (that is, each column only contains one entry per row). Additionally, there are not any repeated columns in any table. This combination ensures that each table fulfills the requirements to be in first normal form, as each intersection of row and column contains only one value.

Additionally, each table is also in second normal form. For the customers, employees, injury reports, products, and orders relations, there is only one primary key (the primary key is a single attribute, not a composite key). Thus, since the primary key in each of these tables is able to uniquely identify the rest of the values in each tuple, the non-primary key attributes are not dependent on a subset of the primary key, only the primary key itself. For the order items table, showing second normal form is not quite as clear, as the primary key in the order items table is a composite key consisting of the order ID and the product ID. However, the non-key attributes in this table require the values for both the order and product ID to be uniquely identified. It is possible for different orders to have the same product ID as another order; thus, the entry in the order item table cannot be uniquely determined by only the product ID. Similarly, for one order ID, there can be numerous product IDs associated with that singular order. Thus, to uniquely identify the number ordered and the number fulfilled for any order item, the combination of the product ID and the order ID is needed. As the non-key attributes are not dependent upon a subset of the primary key, this table is in second normal form.

Finally, each of the tables are in third normal form. There are no transitive functional dependencies present in any of the tables. Each table contains the minimum amount of information required to be uniquely identifiable, with minimal data duplication. In every table, the non-key attributes are only dependent upon the primary key. There are no tables where the primary key can be used to uniquely identify another attribute, which can then be used to uniquely identify another attribute within the relation. Each attribute can only be determined by the primary key itself. This ensures that each relation is in third normal form.

**4. APPLICATION IMPLEMENTATION DETAILS**

Our team worked to create a Python console application to interact with our database. To connect with the MySQL database, the mysql.connector module was utilized. This module provides a method for Python and MySQL to communicate with each other, allowing the Python program to generate queries to the MySQL database that the MySQL database can understand and perform. The following code allows for the connection between the Python script and MySQL database to be created:

    usr = input("Enter SQL username: ")

    pwd = input("Enter SQL password: ")

    mydbConn = mysql.connector.connect(

        host="localhost",

        user=usr,

        password=pwd,

        database="warehouse"

    )

This code specifies the host, user, password, and database to connect to. Our app prints out a selection of different options to do. You can view the entirety of all relevant tables, as well as some selections filter the outputs. You can create a new customer, update an order, and delete an injury report as well. If given more time, we would add validation to the creation of new entities, so that we ensure all entries fit in their respective fields.

For the SQL statements, we used python format strings, so we can simply insert the relevant data. We had to manually escape the quotes in the string, and the input data goes between them. This does leave it open to attacks though.

**5. WORKFLOW/DEMO**

This program functions by printing a menu that lists various options that a user can select to create, read, update, or delete various options in the database. The user makes their selection, and if additional information is required to fulfill the action, the user is prompted again for the needed information. Once all the information required has been provided, the appropriate CRUD operation is conducted on the database.

A screenshot of the menu presented to the user is below:

Text

Description automatically generated

**6. CONCLUSION**

In conclusion, this project helped to teach us about the process of designing and creating an application that utilizes a database. This project helped us to learn about how to design and create a database from scratch. We learned how to go from a basic concept, to identifying tables, finding functional dependencies, and ensuring that the tables are normalized properly, to creating a functional application that connects with a database. This project gave our team experience from start to finish of designing and creating a database, and an application that utilizes the database.

Throughout this project, we initially had some confusion regarding how to ensure each table was normalized, and what the functional dependencies all consisted of. We were able to resolve these issues with careful review of the textbook and lecture slides, and we were able to successfully identify functional dependencies and normalize the tables.

There are numerous potential improvements that this project could have in the future. Potentially, a GUI could be added to the application to make it a bit more user friendly than a console application. Additional queries could also be added into the application, allowing for more CRUD operations on and between different tables. This program could be made more robust (potentially with additional error checking and validation). While there are some improvements that could be made, this program demonstrates a functional application that interacts with a database.

1. Note that the injury report table is a more recent addition, which was not present in the initial proposal. [↑](#footnote-ref-2)