

**South Dakota School of Mines & Technology**  
**Database Management Systems, Spring 2022**

CSC 484-M01

**Final Project Report**

Ryan Finn

## 1. INTRODUCTION

This document provides a detailed report on CSC 484 Final Project. The project was undertaken by Ryan Finn.

All questions, and complaints, can be directed to this (these) individual(s).

Ryan Finn: ryan.finn@mines.sdsmt.edu.com

## 2. FUNCTIONAL DEPENDENCIES

Staff Relation:

- first\_name, last\_name, sex, birth\_date, and hire\_date are all functionally dependent on staff\_no in a 1:\* relationship.
- staff\_no is the primary key and is in a 1:1 relationship with the other attributes.

Customers Relation:

- first\_name and last\_name are both functionally dependent on customer\_no in a 1:\* relationship.
- customer\_no is the primary key and is in a 1:1 relationship with the other attributes.

Products Relation:

- unit\_price, quantity, and locations are all functionally dependent on product\_no in a 1:\* relationship.
- product\_no is the primary key and is in a 1:1 relationship with the other attributes.

Shipments Relation:

- status, shipment\_date, and delivery\_date are all functionally dependent on shipment\_no in a 1:\* relationship.
- shipment\_no is the primary key and is in a 1:1 relationship with the other attributes.

Order Relation:

- shipment\_no, customer\_no, order\_date, and products are all functionally dependent on order\_no in a 1:\* relationship.
- order\_no is the primary key and is in a 1:1 relationship with the other attributes.

### 3. NORMALIZATION

The 3NF relations are of the form:

- staff (staff\_no, first\_name, last\_name, sex, birth\_date, hire\_date)
- customers (customer\_no, first\_name, last\_name)
- products (product\_no, unit\_price, quantity, locations)
- shipments (shipment\_no, status, shipment\_date, delivery\_date)
- orders (order\_no, order\_date, products, shipment\_no, customer\_no)

Note that staff\_no is never a foreign key, it is only ever used in the staff relation. The orders relation is also the only relation that links the customers, products, and shipments relation together, though the link with products is indirect.

### 4. APPLICATION IMPLEMENTATION DETAILS

This application is a console app written in Python. It only requires Python 3.10 and the mysql.connector package be installed.

This application is designed with particular commands and functionality in mind. It has numerous built in functions for the user to invoke that will perform complex tasks so the user does not need to manually do so. However, the option to perform a basic query is also available. This decision was made in part to reduce user error and be more beginner friendly. Complex tasks, such as querying multiple tables or conditional updating, should be done by the application itself. The other reason for this design was to increase security; to make sure users could not perform malicious actions on the database. A set of predefined, hardcoded functions allows for both of these requirements.

Python -> MySQL connection:

```
try:
    cnx = mysql.connector.connect(**config)
    cursor = cnx.cursor()
except mysql.connector.Error as err:
    print(f" Could not connect to database `{database_name}`.")
    print(f" {err}\n")
    exit(1)

try:
    cursor.execute(f"USE {database_name}")
```

```

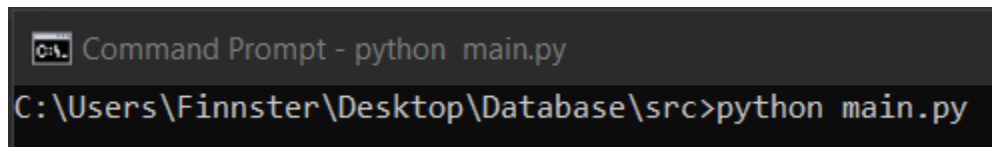
        print("CONNECTED TO DATABASE")
    except mysql.connector.Error as err:
        print(f" Database `{database_name}` does not exist.")
        if (
            err.errno == errorcode.ER_DB_DROP_EXISTS
            or err.errno == errorcode.ER_BAD_DB_ERROR
        ):
            self.__createDatabase(cursor, cnx, database_name)
            self.__createTables(cursor)
            cnx.commit()
            print("CONNECTED TO DATABASE")
        else:
            print(f" {err}\n")
            self.close()
            exit(1)
    print()
    self.__cnx = cnx
    self.__cursor = cursor

```

The MySQL statements can be found in the MySQL directory of the app, under src.

## 5. WORKFLOW/DEMO

To start the application, open a terminal (either through an IDE like PyCharm or a normal Command Prompt), navigate to the folder that contains the program (Project/src), and use the command `python main.py` to begin running.

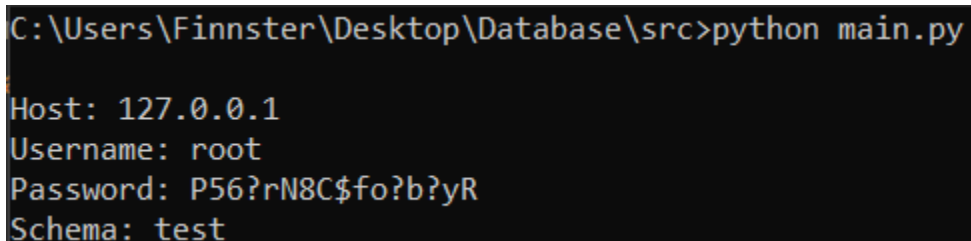


```

C:\Users\Finnster\Desktop\Database\src>python main.py

```

This will prompt the user for the MySQL database's login information, such as the host, username, password, and the specific schema to connect to.



```

C:\Users\Finnster\Desktop\Database\src>python main.py
Host: 127.0.0.1
Username: root
Password: P56?rN8C$fo?b?yR
Schema: test

```

If the connection can be made it will try to find the desired schema. If one does not exist it will create and initialize one.

```
C:\Users\Finnster\Desktop\Database\src>python main.py

Host: 127.0.0.1
Username: root
Password: P56?rN8C$fo?b?yR
Schema: test

OPENING CONNECTION...
  Database `test` does not exist.
  Database `test` created successfully.
  Table `staff` created successfully.
  Table `customers` created successfully.
  Table `products` created successfully.
  Table `shipments` created successfully.
  Table `orders` created successfully.
CONNECTED TO DATABASE

Type `help` for more information.
Type `quit` to exit the program.

cmd>>
```

The user may now begin using the application to manipulate the database. Typing help will show the schema structure and give a list of available commands to use.

```
cmd>> help

Database tables:
=====
  staff:      staff_no, first_name, last_name, sex, birth_date, hire_date
  customers:  customer_no, first_name, last_name
  products:   product_no, unit_price, quantity, locations
  shipments:  shipment_no, status, shipment_date, delivery_date
  orders:     order_no, shipment_no, customer_no, order_date, products

Attribute formats:
=====
  staff_no, product_no, shipment_no, order_no: set automatically
  birth_date, hire_date, ship_date, delivery_date, order_date: YYYY-MM-DD
  sex:      M, F, or OTHER
  status:   PROCESSING, IN-TRANSIT, or DELIVERED
  locations, products:  "a, b, c, etc."

Documented commands (type help <topic>):
=====
getCustomersOfProduct   newLocation           updateLastName
getCustomersOfShipment  newOrder              updateOrderShipment
getDeliveredOrders      newProduct            updateProductPrice
getShipmentsOfCustomer  newShipment           updateProductQuantity
getShipmentsOfProduct   newStaff              updateSex
getShippedOrders        query                 updateShipmentStatus
getUnshippedOrders      removeLocation
newCustomer             updateFirstName

Undocumented commands:
=====
help  quit

cmd>>
```

Typing help <command> will show more information about that command

```
cmd>> help newStaff
      newStaff <first_name> <last_name> <sex> <birth_date>

cmd>>
```

From here the user may query or manipulate the database as they like, within the restrictions of what the program allows. For example, a new shipment may have arrived today, so the user may use newShipment to add it into the database. Then, once the shipment is filled with customer orders it can be shipped from the warehouse and its status set to 'in-transit'. A user can also query which orders are currently in transit.

```
cmd>> newShipment
INSERT INTO shipments (shipment_no) VALUES (NULL);:
  Inserted into row 5

cmd>> updateOrderShipment 4 5
UPDATE orders SET shipment_no = 5 WHERE order_no = '4';
Updated 1 rows

cmd>> updateOrderShipment 5 5
UPDATE orders SET shipment_no = 5 WHERE order_no = '5';
Updated 1 rows

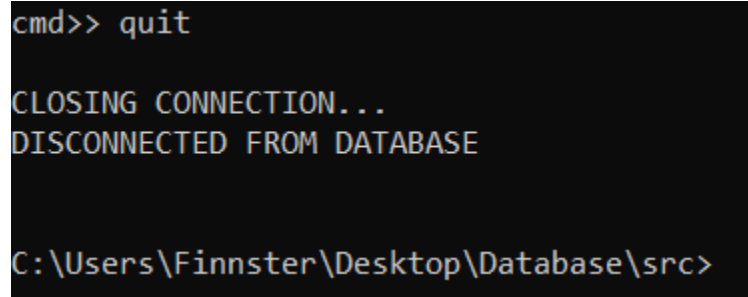
cmd>> updateOrderShipment 6 5
UPDATE orders SET shipment_no = 5 WHERE order_no = '6';
Updated 1 rows

cmd>> updateShipmentStatus 5 IN-TRANSIT
UPDATE shipments SET status = 'IN-TRANSIT' WHERE shipment_no = 5;
Updated 1 rows

cmd>> getShippedOrders
SELECT o.*
FROM shipments s JOIN orders o USING (shipment_no)
WHERE status = 'IN-TRANSIT'
GROUP BY order_no;
:
(4, 5, 3, datetime.datetime(2022, 5, 2, 2, 4, 32), '1, 2, 3, 4')
(5, 5, 5, datetime.datetime(2022, 5, 2, 2, 4, 57), '5, 8, 6')
(6, 5, 4, datetime.datetime(2022, 5, 2, 2, 5, 13), '1')

cmd>>
```

Commands will also show their respective MySQL statements. Once done the user may use the quit command to end the session.



```
cmd>> quit  
  
CLOSING CONNECTION...  
DISCONNECTED FROM DATABASE  
  
C:\Users\Finnster\Desktop\Database\src>
```

## 6. CONCLUSION

The application is very basic, it could probably be improved by adding an actual GUI so it's more user-friendly. I would have liked to have made a web app for this project, but unfortunately I didn't have the time to learn how, so I opted for Python instead. But it works and provides all the functionality a normal employee could need.

Most of my problems when creating this app were simple MySQL syntax errors. Normally the MySQL Workbench will warn you about syntax errors and other things, but since all the SQL code was done as Python strings it became harder to debug. I found that writing the SQL statements in the workbench and then copy-pasting them over to Python worked best, but it's still not ideal.

I learned making a SQL database app like this can take a lot of work in terms of user support. I needed to make 23 different functions just to make sure the database was secure from user mistakes.