

---

# Stockwell Financial Database

---

## VOLUME I: CREATING FINANCIAL DATABASE IN MySQL

AUSTIN STOCKWELL  
DECEMBER 2019

## Acknowledgments

Various resources were used to help create this database. The end result was a device that was suited to my specific needs. I would like to thank all of those who helped create the final product!

[?] [?]

# Contents

<b>Acknowledgments</b>	<b>i</b>
<b>List of Figures</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Features</b>	<b>1</b>
<b>3 Database Design</b>	<b>2</b>
3.1 Creating EER Diagram . . . . .	2
3.2 Forward Engineer Feature . . . . .	3
3.3 Creating the Database . . . . .	5
3.3.1 Schema . . . . .	5
3.3.2 Tables . . . . .	5
3.4 Table Relationships . . . . .	9
3.5 Inserting Data into Tables . . . . .	10
3.6 Creating .sql Files . . . . .	10
3.7 Database Structure . . . . .	12
<b>4 Future Improvements</b>	<b>13</b>

## List of Figures

1	EER Diagram . . . . .	2
2	Forward Engineer Feature (1/2) . . . . .	3
3	Forward Engineer Feature (2/2) . . . . .	4
4	Schema Creation . . . . .	5
5	Asset Table . . . . .	5
6	Asset Category Table . . . . .	5
7	Credit Card Table . . . . .	6
8	Credit Transaction Table . . . . .	6
9	Bank Account Table . . . . .	7
10	Bank Transaction Table . . . . .	7
11	Liability Category Table . . . . .	8
12	INSERT Data Command . . . . .	10
13	Viewing Data in MySQL Workbench . . . . .	10
14	.sql File . . . . .	11
15	Database Structure Tree . . . . .	12

# 1 Introduction

This project started as a desire to create a personal database to archive financial assets and liabilities. The main goal of the project was to create a system that actively helped aid future financial decisions while maintaining accurate and easily accessible (yet secure) financial data.

This document describes the process of using MySQL and the MySQL Workbench to create a functioning financial database. As the database progresses, more feature will be added that will be documented in a later volume.

# 2 Features

The following features are integrated into the database:

- Credit card accounts with expense tracking
- Bank accounts with withdrawal and deposit tracking
- Assets with associated income and expenditures

## 3 Database Design

### 3.1 Creating EER Diagram

The EER (Enhanced Entity-Relationship) Diagram is used to create a visual mapping of how the database is to be setup. It includes the various tables within the database and their relationships to one another that determine database functionality and structure.

To create the EER diagram: FILE; NEW MODEL; NEW DIAGRAM.

The user can then create the database tables and define the connections (relationships) between the tables graphically (as shown in Figure 1).

This database consists of seven tables as listed below:

- Asset Table
- Asset Category Table
- Credit Card Table
- Credit Transaction Table
- Bank Account Table
- Bank Transaction Table
- Liability Category Table

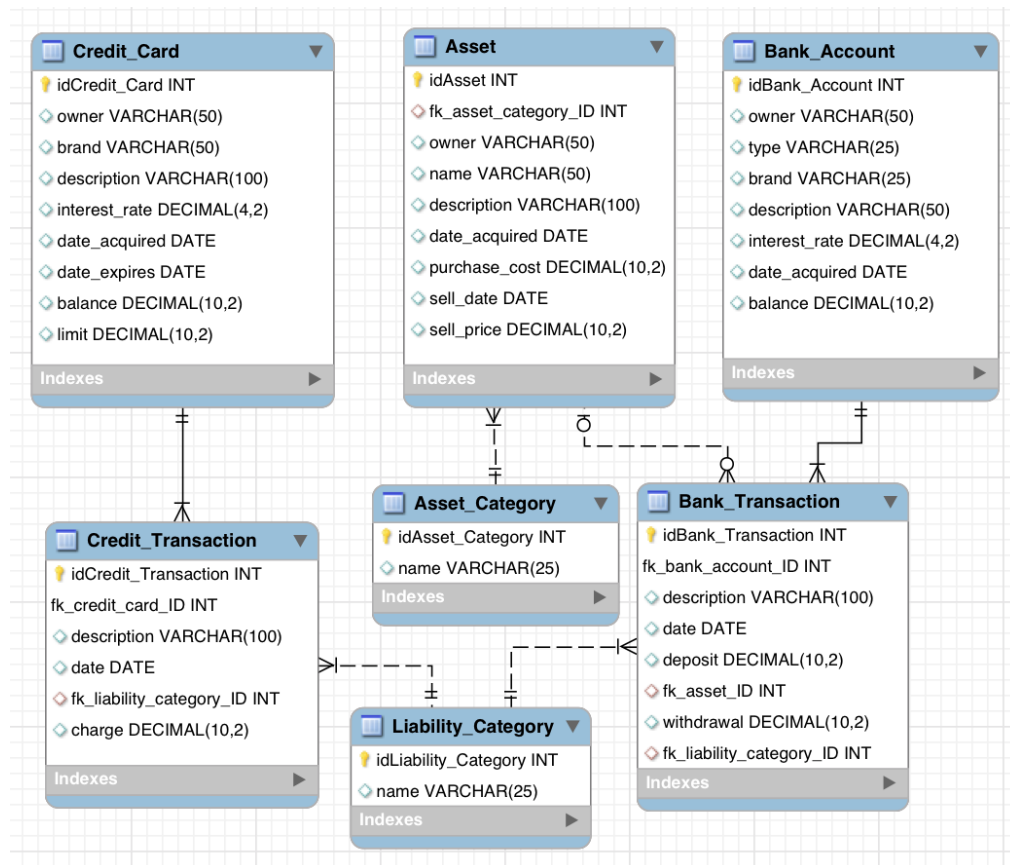


Figure 1: EER Diagram

### 3.2 Forward Engineer Feature

MySQL's 'Forward Engineer' feature allows auto-generation of code based on the EER diagram created in Figure 1. This feature allows the user to get started quickly without having to manually code all the schema/database/table creation statements. Although certain modifications are needed to get the code running effectively and efficiently, the 'Forward Engineer' feature was used as a starting point for this project.

The screenshot shows a macOS-style window titled "Forward Engineer to Database". Inside, there's a sidebar on the left with five radio buttons: "Connection Options" (selected), "Options", "Select Objects", "Review SQL Script", and "Commit Progress". The main area is titled "Set Parameters for Connecting to a DBMS". It contains two dropdown menus at the top: "Stored Connection:" with "Stockwell\_Financial" selected, and "Connection Method:" with "Standard (TCP/IP)" selected. Below these are three tabs: "Parameters" (active), "SSL", and "Advanced". The "Parameters" tab has several fields: "Hostname:" with "127.0.0.1", "Port:" with "3306", "Username:" with "root", and "Password:" with a "Store in Keychain ..." button and a "Clear" button. There's also a "Default Schema:" field. To the right of each field is a small text description. At the bottom right of the window are "Go Back" and "Continue" buttons.

Forward Engineer to Database

**Set Parameters for Connecting to a DBMS**

Connection Options  
Options  
Select Objects  
Review SQL Script  
Commit Progress

Stored Connection: Stockwell\_Financial Select from saved connection settings

Connection Method: Standard (TCP/IP) Method to use to connect to the RDBMS

Parameters SSL Advanced

Hostname: 127.0.0.1 Port: 3306 Name or IP address of the server host - and TCP/IP port.

Username: root Name of the user to connect with.

Password: Store in Keychain ... Clear The user's password. Will be requested later if it's not set.

Default Schema: The schema to use as default schema. Leave blank to select it later.

Go Back Continue

Figure 2: 'Set parameters for connecting to DBMS'

The auto-generated code is shown in Figure 3. This code was used as a basis to build the database. Some modifications are required such as changing 'mydb' to 'Stockwell Financial'.

**NOTE: It is critical that the code is set to 'Foreign Key Checks = 1'. Setting this equal to '1' enforces Foreign Key constraints and ensure that update behavior is upheld!**

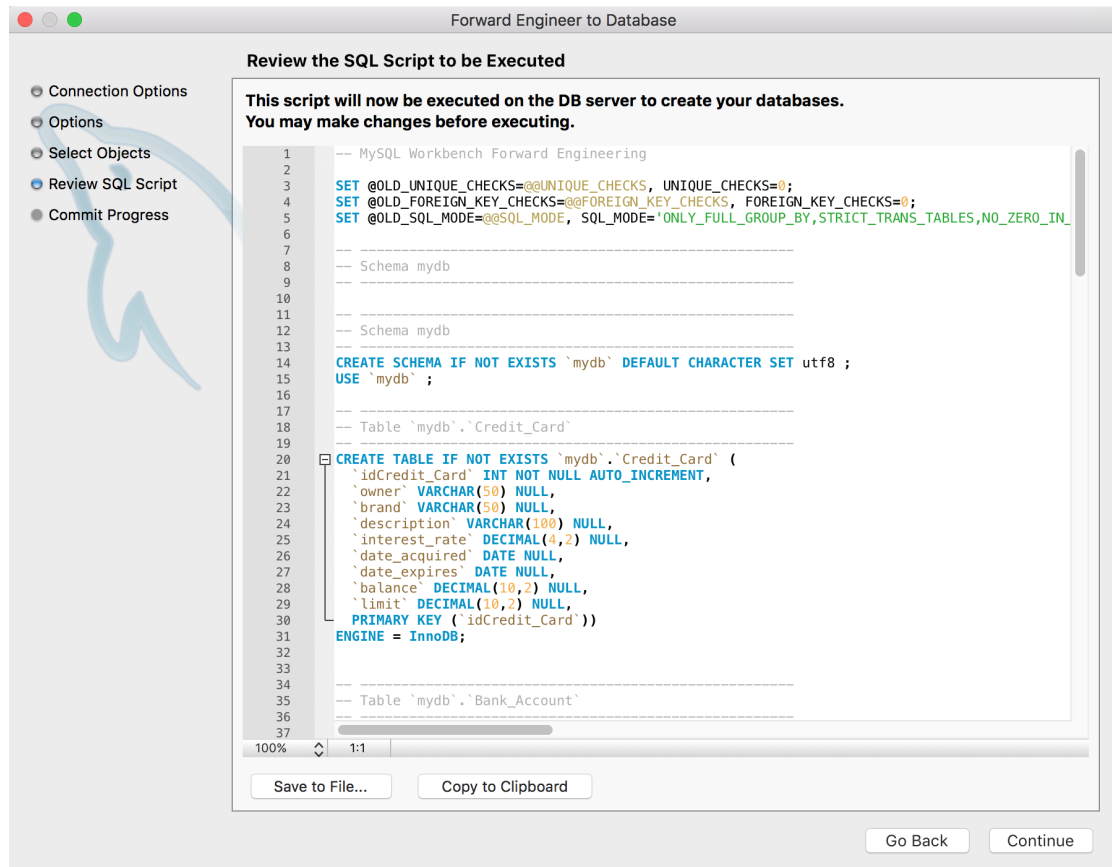


Figure 3: Forward Engineer code



### 3.3 Creating the Database

#### 3.3.1 Schema

The first step is to create the 'SCHEMA' and 'USE' the created schema (Stockwell Financial).

```
-----  
-- Schema Stockwell_Financial  
-----  
CREATE SCHEMA IF NOT EXISTS `Stockwell_Financial` DEFAULT CHARACTER SET utf8 ;  
USE `Stockwell_Financial` ;
```

Figure 4: Creating and using Stockwell Financial schema

#### 3.3.2 Tables

Next, tables used to store data were created and added to the database. The code for the tables are shown in the figures below.

```
-----  
-- Table `Stockwell_Finance`.`Asset`  
-----  
DROP TABLE Stockwell_Finance.Asset;  
CREATE TABLE IF NOT EXISTS `Stockwell_Finance`.`Asset` (  
  `idAsset` INT NOT NULL AUTO_INCREMENT,  
  `owner` VARCHAR(25) NULL,  
  `name` VARCHAR(50) NULL,  
  `description` VARCHAR(100) NULL,  
  `date_acquired` DATE NULL,  
  `date_sold` DATE NULL,  
  `purchase_cost` DECIMAL(10,2) NULL,  
  `sell_price` DECIMAL(10,2) NULL,  
  PRIMARY KEY (`idAsset`))  
ENGINE = InnoDB;
```

Figure 5: Creating Asset table

```
-----  
-- Table `Stockwell_Financial`.`Asset_Category`  
-----  
DROP TABLE IF EXISTS `Stockwell_Financial`.`Asset_Category` ;  
CREATE TABLE IF NOT EXISTS `Stockwell_Financial`.`Asset_Category` (  
  `idAsset_Category` INT NOT NULL AUTO_INCREMENT,  
  `name` VARCHAR(25) NULL,  
  PRIMARY KEY (`idAsset_Category`))  
ENGINE = InnoDB;
```

Figure 6: Creating Asset Category table

```

-----
-- Table `Stockwell_Financial`.`Credit_Card`
-----
DROP TABLE IF EXISTS `Stockwell_Financial`.`Credit_Card` ;

CREATE TABLE IF NOT EXISTS `Stockwell_Financial`.`Credit_Card` (
  `idCredit_Card` INT NOT NULL AUTO_INCREMENT,
  `owner` VARCHAR(50) NULL,
  `brand` VARCHAR(50) NULL,
  `description` VARCHAR(100) NULL,
  `interest_rate` DECIMAL(4,2) NULL,
  `date_acquired` DATE NULL,
  `date_expires` DATE NULL,
  `balance` DECIMAL(10,2) NULL,
  `limit` DECIMAL(10,2) NULL,
  PRIMARY KEY (`idCredit_Card`))
ENGINE = InnoDB;

```

Figure 7: Creating Credit Card table

```

-----
-- Table `Stockwell_Financial`.`Credit_Transaction`
-----
DROP TABLE IF EXISTS `Stockwell_Financial`.`Credit_Transaction` ;

CREATE TABLE IF NOT EXISTS `Stockwell_Financial`.`Credit_Transaction` (
  `idCredit_Transaction` INT NOT NULL AUTO_INCREMENT,
  `fk_credit_card_ID` INT NOT NULL,
  `description` VARCHAR(100) NULL,
  `fk_liability_category_ID` INT NULL,
  `date` DATE NULL,
  `charge` DECIMAL(10,2) NULL,
  PRIMARY KEY (`idCredit_Transaction`, `fk_credit_card_ID`),
  INDEX `credit_card_ID_idx` (`fk_credit_card_ID` ASC),
  INDEX `Credit_Transaction_credit_card_bill_category_ID_idx` (`fk_liability_category_ID` ASC),
  CONSTRAINT `fk_credit_card_ID`
    FOREIGN KEY (`fk_credit_card_ID`)
      REFERENCES `Stockwell_Financial`.`Credit_Card` (`idCredit_Card`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_credit_card_bill_category_ID`
    FOREIGN KEY (`fk_liability_category_ID`)
      REFERENCES `Stockwell_Financial`.`Liability_Category` (`idLiability_Category`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;

```

Figure 8: Credit Transaction table

```

-----
-- Table `Stockwell_Financial`.`Bank_Account`
-----
DROP TABLE IF EXISTS `Stockwell_Financial`.`Bank_Account` ;

CREATE TABLE IF NOT EXISTS `Stockwell_Financial`.`Bank_Account` (
  `idBank_Account` INT NOT NULL AUTO_INCREMENT,
  `owner` VARCHAR(50) NULL,
  `type` VARCHAR(25) NULL,
  `brand` VARCHAR(25) NULL,
  `description` VARCHAR(50) NULL,
  `interest_rate` DECIMAL(4,2) NULL,
  `date_acquired` DATE NULL,
  `balance` DECIMAL(10,2) NULL,
  PRIMARY KEY (`idBank_Account`))
ENGINE = InnoDB;

```

Figure 9: Creating Bank Account table

```

-----
-- Table `Stockwell_Financial`.`Bank_Transaction`
-----
DROP TABLE IF EXISTS `Stockwell_Financial`.`Bank_Transaction` ;

CREATE TABLE IF NOT EXISTS `Stockwell_Financial`.`Bank_Transaction` (
  `idBank_Transaction` INT NOT NULL AUTO_INCREMENT,
  `fk_bank_account_ID` INT NOT NULL,
  `date` DATE NULL,
  `description` VARCHAR(100) NULL,
  `deposit` DECIMAL(10,2) NULL,
  `fk_asset_ID` INT NULL,
  `withdrawal` DECIMAL(10,2) NULL,
  `fk_liability_category_ID` INT NULL,
  PRIMARY KEY (`idBank_Transaction`, `fk_bank_account_ID`),
  INDEX `bank_account_ID_idx` (`fk_bank_account_ID` ASC),
  INDEX `Bank_Transaction_asset_ID_idx` (`fk_asset_ID` ASC),
  INDEX `fk_liability_category_ID_idx` (`fk_liability_category_ID` ASC),
  CONSTRAINT `fk_bank_account_ID`
    FOREIGN KEY (`fk_bank_account_ID`)
    REFERENCES `Stockwell_Financial`.`Bank_Account` (`idBank_Account`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_asset_ID`
    FOREIGN KEY (`fk_asset_ID`)
    REFERENCES `Stockwell_Financial`.`Asset` (`idAsset`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_liability_category_ID`
    FOREIGN KEY (`fk_liability_category_ID`)
    REFERENCES `Stockwell_Financial`.`Liability_Category` (`idLiability_Category`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;

```

Figure 10: Creating Bank Transaction table

```
-----  
-- Table `Stockwell_Financial`.`Liability_Category`  
-----  
DROP TABLE IF EXISTS `Stockwell_Financial`.`Liability_Category` ;  
  
CREATE TABLE IF NOT EXISTS `Stockwell_Financial`.`Liability_Category` (  
  `idLiability_Category` INT NOT NULL AUTO_INCREMENT,  
  `name` VARCHAR(25) NULL,  
  PRIMARY KEY (`idLiability_Category`))  
ENGINE = InnoDB;
```

Figure 11: Creating Liability Category table

**NOTE:** If you are running into **CONSTRAINT ERRORS** while attempting to create the tables, follow the outlined procedure:

- Set the 'FOREIGN KEY CHECKS = 0'
- CREATE all tables and INSERT all data
- Set 'FOREIGN KEY CHECKS = 1'

### 3.4 Table Relationships

This database is composed mostly of one-to-many relationships. For example, the relationship between the Credit Card Table and the Credit Transaction table is a one-to-many relationship. There can be MANY transactions on any given credit card, but only ONE credit card for any given transaction!

The tables are related to one another in a manner that DELETION of data in one table may result in DELETION of data in other tables with the corresponding data. This is due to the CASCADE behavior of the Foreign Keys in the Bank Transaction and Credit Transaction tables.

- **Deleting an Asset Category deletes corresponding Assets**
- **Deleting a Credit Card deletes corresponding Credit Transactions**
- **Deleting a Liability Category deletes corresponding Credit Transactions**
- **Deleting a Bank Account deletes corresponding Bank Transactions**
- **Deleting an Asset deletes corresponding Bank Transactions**
- **Deleting an Asset Category deletes corresponding Bank Transactions**
- **Deleting a Liability Category deletes corresponding Bank Transactions**

### 3.5 Inserting Data into Tables

The next step in creating the database was to insert data into the tables. This is done with the 'INSERT' command, as shown in Figure 12. Once the appropriate data was inserted into the tables, the data was viewed by using the 'SELECT' command as (also shown in Figure 12).

```
-- INSERT DATA INTO TABLE  
  
SELECT * FROM Asset;  
INSERT INTO Asset VALUES (1, 11, 'Austin', 'Ethereum', 'Ethereum on Binance & Coinbase', '2017-12-01', 500.00, NULL, NULL);  
INSERT INTO Asset VALUES (2, 11, 'Austin', 'Ripple', 'Ripple on Binance', '2017-12-01', 500.00, NULL, NULL);  
SELECT * FROM Asset;
```

Figure 12: INSERT data into Asset table

The screenshot shows a database application interface. At the top, there's a 'Result Grid' with a search bar and 'Filter Rows' button. Below it, a table displays data for the 'Asset' table. The table has columns: idAsset, fk\_asset\_category\_ID, owner, name, description, date\_acquired, purchase\_cost, sell\_date, and sell\_price. Two rows are visible. Below the table, there's a section for 'Asset 1' with 'Apply' and 'Revert' buttons. At the bottom, an 'Action Output' section shows a successful query execution.

idAsset	fk_asset_category_ID	owner	name	description	date_acquired	purchase_cost	sell_date	sell_price
1	11	Austin	Ethereum	Ethereum on Binance & Coinbase	2017-12-01	500.00	NULL	NULL
2	11	Austin	Ripple	Ripple on Binance	2017-12-01	500.00	NULL	NULL

Time	Action	Response	Duration / Fetch Time
08:59:14	SELECT * FROM Stockwell_Financial.Asset LIMIT 0, 1000	2 row(s) returned	0.00024 sec / 0.000...

Figure 13: Data of Asset table

### 3.6 Creating .sql Files

To ensure integrity of data and allow for future rebuilds, all of the separate tables and their corresponding code was placed into separate .sql files. When new data is to be added to the database, the user opens the .sql file corresponding to the table and modifies the table code using various commands.

The files that were created were as follows:

- Asset
- Asset Category
- Bank Account
- Bank Transaction
- Credit Card
- Credit Transaction
- Liability Category

```

/* AUTHOR: Austin Stockwell
   DATE: 12-29-2019
   SCOPE: Financial Database
*/
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=1;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_

-- Schema Stockwell_Financial

CREATE SCHEMA IF NOT EXISTS `Stockwell_Financial` DEFAULT CHARACTER SET utf8 ;
USE `Stockwell_Financial` ;

-- Table `Stockwell_Financial`.`Asset_Category`

DROP TABLE IF EXISTS `Stockwell_Financial`.`Asset_Category` ;

CREATE TABLE IF NOT EXISTS `Stockwell_Financial`.`Asset_Category` (
  `idAsset_Category` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(25) NULL,
  PRIMARY KEY (`idAsset_Category`))
ENGINE = InnoDB;

-- INSERT DATA INTO TABLE

SELECT * FROM Asset_Category;
INSERT INTO Asset_Category VALUES (1, 'Stock');
INSERT INTO Asset_Category VALUES (2, 'Bond');
INSERT INTO Asset_Category VALUES (3, 'Index Fund');
INSERT INTO Asset_Category VALUES (4, 'Mutual Fund');
INSERT INTO Asset_Category VALUES (5, 'IRA');
INSERT INTO Asset_Category VALUES (6, 'ROTH IRA');
INSERT INTO Asset_Category VALUES (7, '401K');
INSERT INTO Asset_Category VALUES (8, 'Roth 401K');
INSERT INTO Asset_Category VALUES (9, 'CD');
INSERT INTO Asset_Category VALUES (10, 'Commodity');
INSERT INTO Asset_Category VALUES (11, 'Cryptocurrency');
INSERT INTO Asset_Category VALUES (12, 'Real Estate');
SELECT * FROM Asset_Category;

```

Figure 14: File containing code of Asset Category table

As shown above, each of the separate table files contains:

- SET (Unique/Foreign Key/SQL Mode)
- CREATE SCHEMA (Including default character sets)
- USE (Selects database to be modified)
- DROP/CREATE (Used to create tables of database)
- INSERT (Populates table with data)
- SELECT (Allows viewing of table data within MySQL Workbench)

### 3.7 Database Structure

Once the tables were created and data was inserted correctly, the structure tree of the database was shown on the left-side of the MySQL Workbench (shown below).

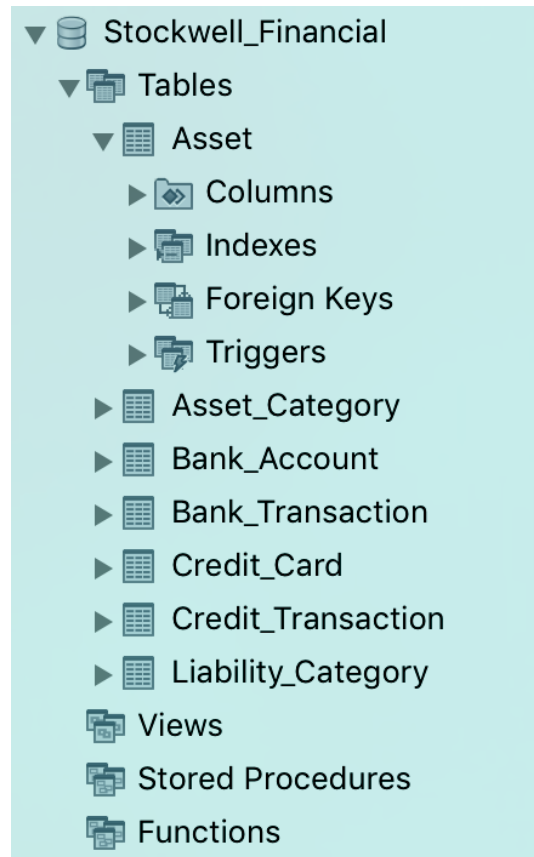


Figure 15: Database structure tree shown in MySQL Workbench



## 4 Future Improvements

The current state of the database is completely functional and satisfies the original goal of the project. However, there are improvements that will be made and integrated into future iterations of the database. These features will be documented in the upcoming volumes.

The following improvements will be integrated into this project:

- Views (To allow easier viewing of database)
- Stored Procedures (To allow balances of bank accounts and credit cards to be calculated)
- Functions (To allow balances of bank accounts and credit cards to be calculated)
- Automating entry of data
- Remote access to database

## References

- [1] M. Z. L. Philip J. Pratt, *A Guide to MySQL*. Course Technology, Cengage Learning, 2006.
- [2] —, *Concepts of Database Management*, 8th ed. Cengage Learning, 2012.

---

# Stockwell Financial Database

---

VOLUME II: MySQL VIEWS / TRIGGERS / DUMPS

AUSTIN STOCKWELL  
MARCH 2020

## Acknowledgments

Various resources were used to help create this database. The end result was a device that was suited to my specific needs. I would like to thank all of those who helped create the final product!

[1] [2] [3]

# Contents

<b>Acknowledgments</b>	<b>i</b>
<b>List of Figures</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Features</b>	<b>1</b>
<b>3 Database Design</b>	<b>2</b>
3.1 Table Structure Modifications . . . . .	2
3.2 VIEW Creation . . . . .	6
3.2.1 Definition . . . . .	6
3.2.2 Bank Deposits View . . . . .	7
3.2.3 Bank Withdrawals View . . . . .	8
3.2.4 Credit Transactions View . . . . .	9
3.3 TRIGGER Creation . . . . .	10
3.3.1 Definition . . . . .	10
3.3.2 Bank Transaction Deposit Trigger . . . . .	11
3.3.3 Bank Transaction Withdrawal Trigger . . . . .	11
3.3.4 Credit Card Payment Trigger . . . . .	12
3.3.5 Credit Card Purchase Trigger . . . . .	12
<b>4 Data Backup Using MySQL's Data Export Feature</b>	<b>13</b>
<b>5 Future Improvements</b>	<b>15</b>

## List of Figures

1	Asset Tabe . . . . .	2
2	Asset Category Tabe . . . . .	2
3	Bank Account Table . . . . .	3
4	Bank Transaction Table . . . . .	3
5	Credit Card Table . . . . .	4
6	Credit Transaction Table . . . . .	4
7	Liability Category Table . . . . .	4
8	Bank Deposits VIEW Code . . . . .	7
9	Bank Deposits VIEW . . . . .	7
10	Bank Withdrawals VIEW Code . . . . .	8
11	Bank Withdrawals VIEW . . . . .	8
12	Credit Transactions VIEW Code . . . . .	9
13	Credit Transactions VIEW . . . . .	9
14	Bank Transaction Deposit Trigger . . . . .	11
15	Bank Transaction Withdrawal Trigger . . . . .	11
16	Credit Card Payment Trigger . . . . .	12
17	Credit Card Purchase Trigger . . . . .	12
18	MySQL DUMP Directory . . . . .	13
19	MySQL Export Window . . . . .	13
20	MySQL DUMP Directory in Finder . . . . .	14

# 1 Introduction

This project started as a desire to create a personal database to archive financial assets and liabilities. The main goal of the project was to create a system that actively helped aid future financial decisions while maintaining accurate and easily accessible (yet secure) financial data.

This document will show what changes were made to the original database in terms of database design. These slight changes helped accommodate improved functionality that ultimately made the Stockwell Financial database much more powerful and user-friendly. This documents will also discuss how the database can be backed up for data protection and database migration.

# 2 Features

The following features are integrated into the database:

- VIEWS
- TRIGGERS

## 3 Database Design

### 3.1 Table Structure Modifications

The database still consists of seven tables as listed below:

- Asset Table
- Asset Category Table
- Bank Account Table
- Bank Transaction Table
- Credit Card Table
- Credit Transaction Table
- Liability Category Table

```
CREATE TABLE IF NOT EXISTS `Stockwell_Financial`.`Asset` (  
  `idAsset` INT NOT NULL AUTO_INCREMENT,  
  `fk_asset_category_ID` INT NULL,  
  `owner` VARCHAR(50) NULL,  
  `name` VARCHAR(50) NULL,  
  `description` VARCHAR(100) NULL,  
  `date_acquired` DATE NULL,  
  `purchase_cost` DECIMAL(10,2) NULL,  
  `sell_date` DATE NULL,  
  `sell_price` DECIMAL(10,2) NULL,  
  PRIMARY KEY (`idAsset`),  
  INDEX `fk_asset_category_ID_idx` (`fk_asset_category_ID` ASC),  
  CONSTRAINT `fk_asset_category_ID`  
    FOREIGN KEY (`fk_asset_category_ID`)  
      REFERENCES `Stockwell_Financial`.`Asset_Category` (`idAsset_Category`)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE)  
ENGINE = InnoDB;
```

Figure 1: Asset table

```
CREATE TABLE IF NOT EXISTS `Stockwell_Financial`.`Asset_Category` (  
  `idAsset_Category` INT NOT NULL AUTO_INCREMENT,  
  `name` VARCHAR(250) NULL,  
  PRIMARY KEY (`idAsset_Category`))  
ENGINE = InnoDB;
```

Figure 2: Asset Category table



```

CREATE TABLE IF NOT EXISTS `Stockwell_Financial`.`Bank_Account` (
  `idBank_Account` INT NOT NULL AUTO_INCREMENT,
  `owner` VARCHAR(50) NULL,
  `type` VARCHAR(25) NULL,
  `brand` VARCHAR(25) NULL,
  `description` VARCHAR(50) NULL,
  `interest_rate` DECIMAL(4,2) NULL,
  `date_acquired` DATE NULL,
  `balance` DECIMAL(10,2) NOT NULL,
  PRIMARY KEY (`idBank_Account`))
ENGINE = InnoDB;

```

Figure 3: Bank Account table

```

CREATE TABLE IF NOT EXISTS `Stockwell`.`Bank_Transaction` (
  `idBank_Transaction` INT NOT NULL AUTO_INCREMENT,
  `fk_bank_account_ID` INT NOT NULL,
  `description` VARCHAR(100) NULL,
  `deposit` DECIMAL(10,2) NULL,
  `withdrawal` DECIMAL(10,2) NULL,
  `fk_asset_ID` INT NULL,
  `fk_asset_category_ID` INT NULL,
  `fk_credit_card_ID` INT NULL,
  `fk_liability_category_ID` INT NULL,
  `date` DATE NULL,
  PRIMARY KEY (`idBank_Transaction`, `fk_bank_account_ID`),
  INDEX `bank_account_ID_idx` (`fk_bank_account_ID` ASC),
  INDEX `Bank_Transaction_asset_ID_idx` (`fk_asset_ID` ASC),
  INDEX `fk_liability_category_ID_idx` (`fk_liability_category_ID` ASC),
  INDEX `fk_asset_category_ID_idx` (`fk_asset_category_ID` ASC),
  CONSTRAINT `constraint_fk_bank_account_ID`
    FOREIGN KEY (`fk_bank_account_ID`)
      REFERENCES `Stockwell_Financial`.`Bank_Account` (`idBank_Account`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `constraint_fk_asset_ID`
    FOREIGN KEY (`fk_asset_ID`)
      REFERENCES `Stockwell_Financial`.`Asset` (`idAsset`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `constraint_fk_liability_category_ID`
    FOREIGN KEY (`fk_liability_category_ID`)
      REFERENCES `Stockwell_Financial`.`Liability_Category` (`idLiability_Category`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `constraint_fk_asset_category_ID`
    FOREIGN KEY (`fk_asset_category_ID`)
      REFERENCES `Stockwell_Financial`.`Asset_Category` (`idAsset_Category`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `constraint_fk_credit_card_ID`
    FOREIGN KEY (`fk_credit_card_ID`)
      REFERENCES `Stockwell_Financial`.`Credit_Card` (`idCredit_Card`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;

```

Figure 4: Modified Bank Transaction table

```

CREATE TABLE IF NOT EXISTS `Stockwell_Financial`.`Credit_Card` (
  `idCredit_Card` INT NOT NULL AUTO_INCREMENT,
  `owner` VARCHAR(50) NULL,
  `brand` VARCHAR(50) NULL,
  `description` VARCHAR(100) NULL,
  `interest_rate` DECIMAL(4,2) NULL,
  `date_acquired` DATE NULL,
  `date_expires` DATE NULL,
  `balance` DECIMAL(10,2) NOT NULL,
  `credit_limit` DECIMAL(10,2) NULL,
  PRIMARY KEY (`idCredit_Card`))
ENGINE = InnoDB;

```

Figure 5: Credit Card table

```

CREATE TABLE IF NOT EXISTS `Stockwell_Financial`.`Credit_Transaction` (
  `idCredit_Transaction` INT NOT NULL AUTO_INCREMENT,
  `fk_credit_card_ID` INT NOT NULL,
  `description` VARCHAR(100) NULL,
  `fk_liability_category_ID` INT NULL,
  `charge_date` DATE NULL,
  `charge` DECIMAL(10,2) NULL,
  PRIMARY KEY (`idCredit_Transaction`, `fk_credit_card_ID`),
  INDEX `credit_card_ID_idx` (`fk_credit_card_ID` ASC),
  INDEX `Credit_Transaction_credit_card_bill_category_ID_idx` (`fk_liability_category_ID` ASC),
  CONSTRAINT `fk_credit_card_ID`
    FOREIGN KEY (`fk_credit_card_ID`)
      REFERENCES `Stockwell_Financial`.`Credit_Card` (`idCredit_Card`)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
  CONSTRAINT `fk_credit_card_bill_category_ID`
    FOREIGN KEY (`fk_liability_category_ID`)
      REFERENCES `Stockwell_Financial`.`Liability_Category` (`idLiability_Category`)
      ON DELETE CASCADE
      ON UPDATE CASCADE))
ENGINE = InnoDB;

```

Figure 6: Credit Transaction table

```

CREATE TABLE IF NOT EXISTS `Stockwell_Financial`.`Liability_Category` (
  `idLiability_Category` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(25) NULL,
  PRIMARY KEY (`idLiability_Category`))
ENGINE = InnoDB;

```

Figure 7: Liability Category table

The following columns were added to the 'Bank Transaction' table:

- 'fk asset category ID'
- 'fk credit card ID'

The addition of the two columns in the 'Bank Transaction' table allowed the table to be combined with the 'Credit Transaction' table as well as the 'Asset Table'. Doing so allowed the three tables to be combined into a VIEW that greatly improved the overall readability and flow of data when tracking bank transactions.

## **3.2 VIEW Creation**

### **3.2.1 Definition**

VIEWS in MySQL are essentially 'virtual tables'. They do not contain the data that is returned. Instead, the data in views are stored in the tables of the database.

Using VIEWS allow the user to create 'virtual tables' that are much more readable and provide a smoother workflow. Such views often incorporate information from multiple tables into one coherent 'view' while at the same time allowing unnecessary information to be excluded from the view. Views also improve security of the database by showing only intended data to authorized users.

Three views were created for the presentation of data in the database:

- vBank Deposits
- vBank Withdrawals
- vCredit Charges

### 3.2.2 Bank Deposits View

The 'vBank Deposits' view shows similar data that is included in the 'Bank Transaction' table. This view shows only income in the form of deposits to any of the bank accounts in the 'Bank Account' table. This view combines two separate tables ('Bank Account' and 'Bank Transaction') into one window that allows a streamlined workflow for the user. With the creation of the view, the user no longer has to open the two separate tables to gather the data that is now situated in the single view.

```
CREATE VIEW vBank_Deposits
AS
SELECT
    Bank_Transaction.idBank_Transaction,
    Bank_Account.owner account_holder,
    Bank_Account.type account_type,
    Bank_Account.brand bank,
    Bank_Transaction.description deposit_description,
    Bank_Transaction.date,
    Bank_Transaction.deposit deposit_amount,
    Asset_Category.name asset_category
FROM Bank_Transaction
JOIN Bank_Account ON Bank_Transaction.fk_bank_account_ID = Bank_Account.idBank_Account
JOIN Asset_Category ON Bank_Transaction.fk_asset_category_ID = Asset_Category.idAsset_Category
WHERE Bank_Transaction.deposit > 0.00
ORDER BY date DESC;
```

Figure 8: Bank Deposits VIEW Code

idBank_Transaction	account_holder	account_type	bank	deposit_description	date	deposit_amount	asset_category
16	Austin/Belle	Checking	CHASE	Austin Paycheck	2020-02-21	715.90	Paycheck
14	Austin/Belle	Checking	CHASE	Austin Paycheck	2020-02-07	715.90	Paycheck
8	Austin/Belle	Checking	CHASE	Austin Paycheck	2020-01-24	715.90	Paycheck

Figure 9: Bank Deposits VIEW

### 3.2.3 Bank Withdrawals View

The 'vBank Withdrawals' view shows data that is included in the 'Bank Transaction' table. This view combines three separate tables ('Bank Transaction', 'Bank Account', and 'Liability Category') into one window that allows a streamlined workflow for the user. With the creation of the view, the user no longer has to open the three separate tables to gather the data that is now situated in the single view.

```
CREATE VIEW vBank_Withdrawals
AS
SELECT
    Bank_Transaction.idBank_Transaction,
    Bank_Account.owner account_holder,
    Bank_Account.type account_type,
    Bank_Account.brand bank,
    Bank_Transaction.description withdrawal_description,
    Bank_Transaction.date,
    Bank_Transaction.withdrawal withdrawal_amount,
    Liability_Category.name liability_category
FROM Bank_Transaction
JOIN Bank_Account ON Bank_Transaction.fk_bank_account_ID = Bank_Account.idBank_Account
JOIN Liability_Category ON Bank_Transaction.fk_liability_category_ID = Liability_Category.idLiability_Category
ORDER BY date DESC;
```

Figure 10: Bank Withdrawals VIEW Code

	idBank_Transaction	account_holder	account_type	bank	withdrawal_description	date	withdrawal_amou...	liability_category
►	17	Austin/Belle	Checking	CHASE	Austin CHASE FREEDOM Payment	2020-02-23	971.33	Credit Card Payment
	12	Austin/Belle	Checking	CHASE	Belle State Farm Car Insurance	2020-02-19	88.53	Car Insurance
	11	Austin/Belle	Checking	CHASE	Planet Fitness	2020-02-18	22.99	Hobbies
	15	Austin	Checking	Centier	Austin DISCOVER Payment	2020-02-18	13.16	Credit Card Payment
	10	Austin/Belle	Checking	CHASE	Gas Bill	2020-02-07	67.14	Gas Bill

Figure 11: Bank Withdrawals VIEW

### 3.2.4 Credit Transactions View

The 'vCredit Transactions' view shows similar data that is included in the 'Credit Transaction' table. This view combines three separate tables ('Credit Transaction', 'Credit Card', and 'Liability Category') into one window that allows a streamlined workflow for the user. With the creation of the view, the user no longer has to open the three separate tables to gather the data that is now situated in the single view.

```
/*
AUTHOR: Austin Stockwell
SCOPE: View
DESCRIPTION: This is a view that shows credit transactions in a convenient way
*/

CREATE VIEW vCredit_Transactions
AS
SELECT
    Credit_Transaction.idCredit_Transaction,
    Credit_Card.owner card_holder,
    Credit_Card.brand,
    Credit_Transaction.description charge_description,
    Credit_Transaction.charge_date,
    Credit_Transaction.charge,
    Liability_Category.name liability_category
FROM Credit_Transaction
JOIN Credit_Card ON Credit_Transaction.fk_credit_card_ID = Credit_Card.idCredit_Card
JOIN Liability_Category ON Credit_Transaction.fk_liability_category_ID = Liability_Category.idLiability_Category
ORDER BY charge_date DESC;
```

Figure 12: Credit Transactions VIEW Code

	idCredit_Transaction	card_holder	brand	charge_description	charge_date	charge	liability_category
▶	747	Austin	Discover	Smiths	2020-02-23	34.15	Groceries
	749	Austin	CHASE FREEDOM	Amazon	2020-02-22	21.66	Books
	746	Austin	Discover	Albertsons	2020-02-21	12.47	Groceries
	748	Austin	CHASE FREEDOM	Dan Lok HTC	2020-02-21	2495.00	Self Improvement
	745	Austin	Discover	Trader Joes	2020-02-20	61.43	Groceries
	744	Austin	Discover	Walmart	2020-02-20	5.51	Groceries
	743	Austin	Discover	Albertsons	2020-02-17	15.90	Groceries
	742	Austin	Discover	Trader Joes	2020-02-15	50.86	Groceries
	741	Austin	Discover	Albertsons	2020-02-15	6.98	Groceries
	730	Austin	CHASE FREEDOM	Amazon	2020-02-14	9.02	Miscellaneous
	740	Austin	Discover	Walgreens	2020-02-14	9.74	Groceries
	731	Austin	CHASE FREEDOM	Momo Yama Sushi	2020-02-14	36.71	Restaurant
	739	Austin	Discover	Walmart	2020-02-14	17.10	Groceries
	738	Austin	Discover	Dairy Queen	2020-02-11	2.03	Restaurant
	737	Austin	Discover	Albertsons	2020-02-11	6.26	Groceries
	729	Austin	CHASE FREEDOM	Great Clips	2020-02-10	18.00	Miscellaneous
	728	Austin	CHASE FREEDOM	Boulder City Electric Bill	2020-02-10	25.00	Electricity
	726	Austin	Discover	Smiths	2020-02-10	67.65	Groceries

Figure 13: Credit Transactions VIEW

### 3.3 TRIGGER Creation

#### 3.3.1 Definition

Triggers are database objects that are associated with a table. Triggers can be executed when you run INSERT, UPDATE, and DELETE commands on a particular table (or set of tables). They can be invoked BEFORE or AFTER the event.

The second iteration of the database has four triggers (all of which are AFTER INSERT) triggers:

- Bank Transaction Deposit
- Bank Transaction Withdrawal
- Credit Card Payment
- Credit Card Purchase

**NOTE: IF YOU DROP A CORRESPONDING TABLE YOU MUST RE-RUN THESE SCRIPTS IN ORDER FOR THE SCRIPTS TO WORK!**

**NOTE: 'BALANCE' COLUMNS OF 'BANK ACCOUNT' AND 'CREDIT CARD' TABLES MUST BE SET TO 'NOT NULL' FOR TRIGGERS TO WORK CORRECTLY! THIS IS SHOWN IN THE CODE IN FIGURES 3 AND 5!**



### 3.3.2 Bank Transaction Deposit Trigger

The 'tBank Transaction deposit' trigger updates the corresponding bank account balance of the 'Bank Account' table when a deposit is made into that bank account. This trigger is an AFTER INSERT trigger. When an INSERT statement is run on the 'Bank Transaction' table an UPDATE command occurs and the NEW balance is SET. The new balance is equal to the previous balance plus the NEW value of the 'deposit' column in the INSERT statement.

```
USE Stockwell_Financial;

DELIMITER $$

CREATE TRIGGER tBank_Transaction_deposit
AFTER INSERT
ON Bank_Transaction FOR EACH ROW
BEGIN
    UPDATE Bank_Account
    SET balance = balance + NEW.deposit
    WHERE Bank_Account.idBank_Account = NEW.fk_bank_account_ID;
END;
$$
DELIMITER ;
```

Figure 14: Bank Transaction Deposit Trigger

### 3.3.3 Bank Transaction Withdrawal Trigger

The 'tBank Transaction withdrawal' trigger updates the corresponding bank account balance of the 'Bank Account' table when a withdrawal is made on that bank account. This trigger is an AFTER INSERT trigger. When an INSERT statement is run on the 'Bank Transaction' table an UPDATE command occurs and the NEW balance is SET. The new balance is equal to the previous balance minus the NEW value of the 'withdrawal' column in the INSERT statement.

```
USE Stockwell_Financial;

DELIMITER $$

CREATE TRIGGER tBank_Transaction_withdrawal
AFTER INSERT
ON Bank_Transaction FOR EACH ROW
BEGIN
    UPDATE Bank_Account
    SET balance = balance - NEW.withdrawal
    WHERE Bank_Account.idBank_Account = NEW.fk_bank_account_ID;
END;
$$
DELIMITER ;
```

Figure 15: Bank Transaction Withdrawal Trigger

### 3.3.4 Credit Card Payment Trigger

The 'tCredit Card payment' trigger updates the corresponding credit card balance of the 'Credit Card' table when a payment is made on that card from a corresponding bank account. This trigger is an AFTER INSERT trigger. When an INSERT statement is run on the 'Bank Transaction' table, an UPDATE command occurs and the new credit card balance is SET. The new credit card balance is equal to the previous balance minus the NEW value of the 'withdrawal' column of the 'Bank Transaction' table in the INSERT statement.

```
USE Stockwell_Financial;

DELIMITER $$

CREATE TRIGGER tCredit_Card_payment
AFTER INSERT
ON Bank_Transaction FOR EACH ROW
BEGIN
    UPDATE Credit_Card
    SET balance = balance - NEW.withdrawal
    WHERE Credit_Card.idCredit_Card = NEW.fk_credit_card_ID
    AND NEW.fk_liability_category_ID = 16;
END;
$$
DELIMITER ;
```

Figure 16: Credit Card Payment Trigger

### 3.3.5 Credit Card Purchase Trigger

The 'tCredit Card purchase' trigger updates the corresponding credit card balance of the 'Credit Card' table when a purchase is made on that card. This trigger is an AFTER INSERT trigger. When an INSERT statement is run on the 'Credit Transaction' table an UPDATE command occurs and the new credit card balance is SET. The new credit card balance is equal to the previous balance plus the NEW value of the 'charge' column of the 'Credit Transaction' table in the INSERT statement.

```
USE Stockwell_Financial;

DELIMITER $$

CREATE TRIGGER tCredit_Transaction_purchase
AFTER INSERT
ON Credit_Transaction FOR EACH ROW
BEGIN
    UPDATE Credit_Card
    SET balance = balance + NEW.charge
    WHERE Credit_Card.idCredit_Card = NEW.fk_credit_card_ID;
END;
$$
DELIMITER ;
```

Figure 17: Credit Card Purchase Trigger

## 4 Data Backup Using MySQL's Data Export Feature

It is possible to backup all database information instantly with the 'DATA EXPORT' feature of MySQL. There are three options to do so:

- Dump Structure and Data
- Dump Data Only
- Dump Structure Only

The first step of the process is creating the correct path to the 'mysqldump Tool' and 'Export Directory Path' within the MySQL Workbench Preferences Menu (shown below).

The 'mysqldump Tool' is by default located at /usr/local/mysql/bin/mysqldump. Once this file path is chosen, all exported files (referred to as DUMPS in MySQL) will be placed on the 'Export Directory Path', which in this example is located at (tilda)/dumps.

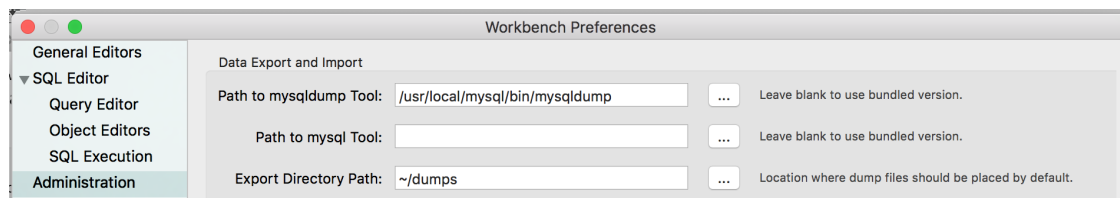


Figure 18: MySQL DUMP Directory

**NOTE: It may be necessary to show 'hidden files' in OS X finder window! This can be done typing the following command in the terminal: defaults write com.apple.finder AppleShowAllFiles TRUE**

To access the 'Data Export tool' navigate to 'Server' — 'Data Export'. This pulls up the MySQL Data Export window shown below. In this window, the user can select multiple databases and tables to be exported. It is also possible to export views, triggers, stored procedures, functions, and events.

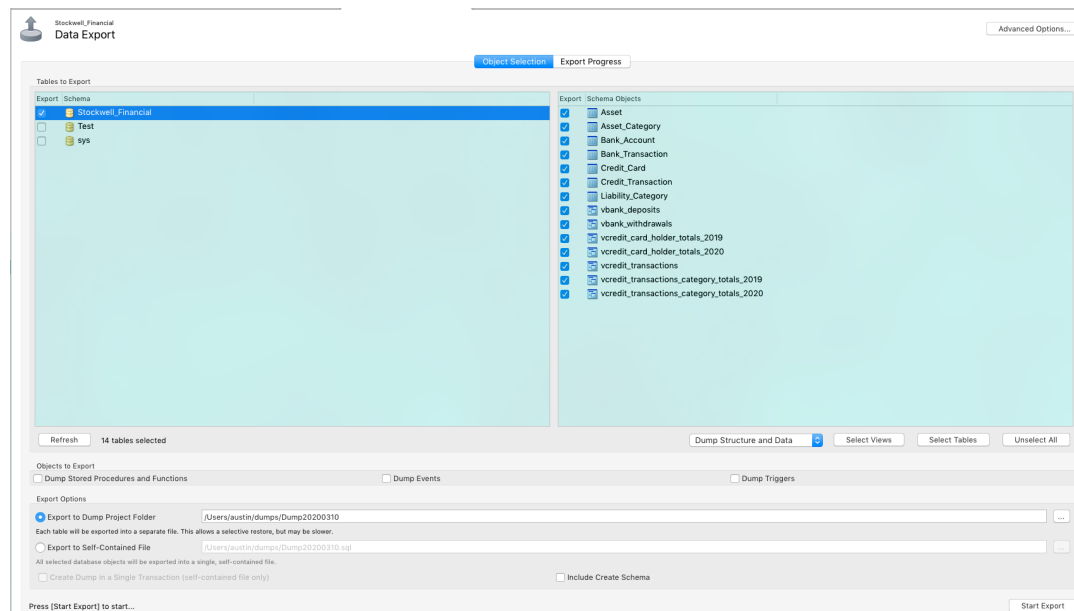


Figure 19: MySQL Export Window

Once the export has completed, they can be found inside of the directory that was set in the first step. The directory in this example is shown in finder below:

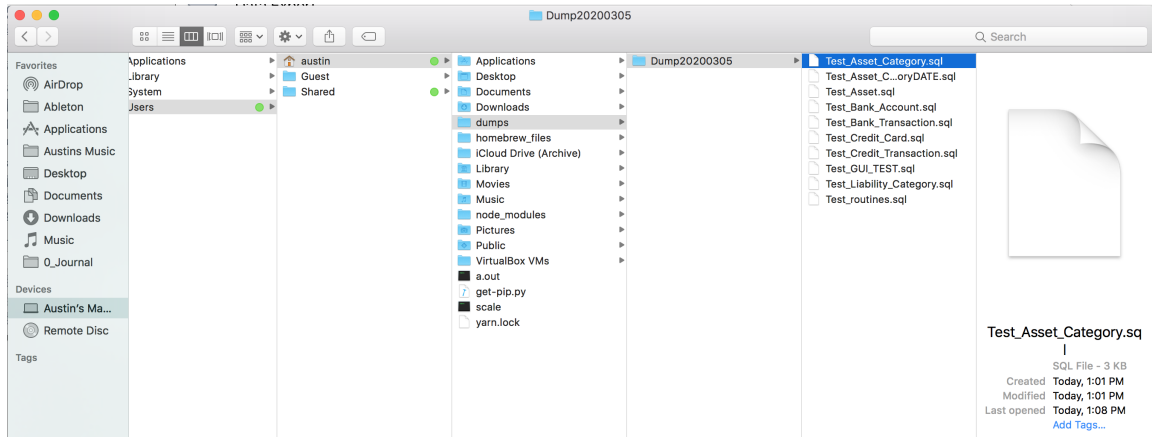


Figure 20: MySQL DUMP Directory in Finder

The files located in this directory can be used directly to migrate or backup the desired database and database components to prevent loss of data.

## 5 Future Improvements

The current state of the database is completely functional and satisfies the original goal of the project. However, there are improvements that will be made and integrated into future iterations of the database. These features will be documented in the upcoming volumes.

The following improvements will be integrated into this project:

- Commit() and rollback() functionality
- Creation of a Graphical User Interface (GUI) that will be used to enter data into the database
- Visualization of data
- Remote access to database

## References

- [1] M. Z. L. P. J. Pratt, *A Guide to MySQL*. Course Technology, Cengage Learning, 2006.
- [2] —, *Concepts of Database Management*. Cengage Learning, 2012, vol. 8.
- [3] (2020, March). [Online]. Available: <http://www.mysqltutorial.org/>