

---

# 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

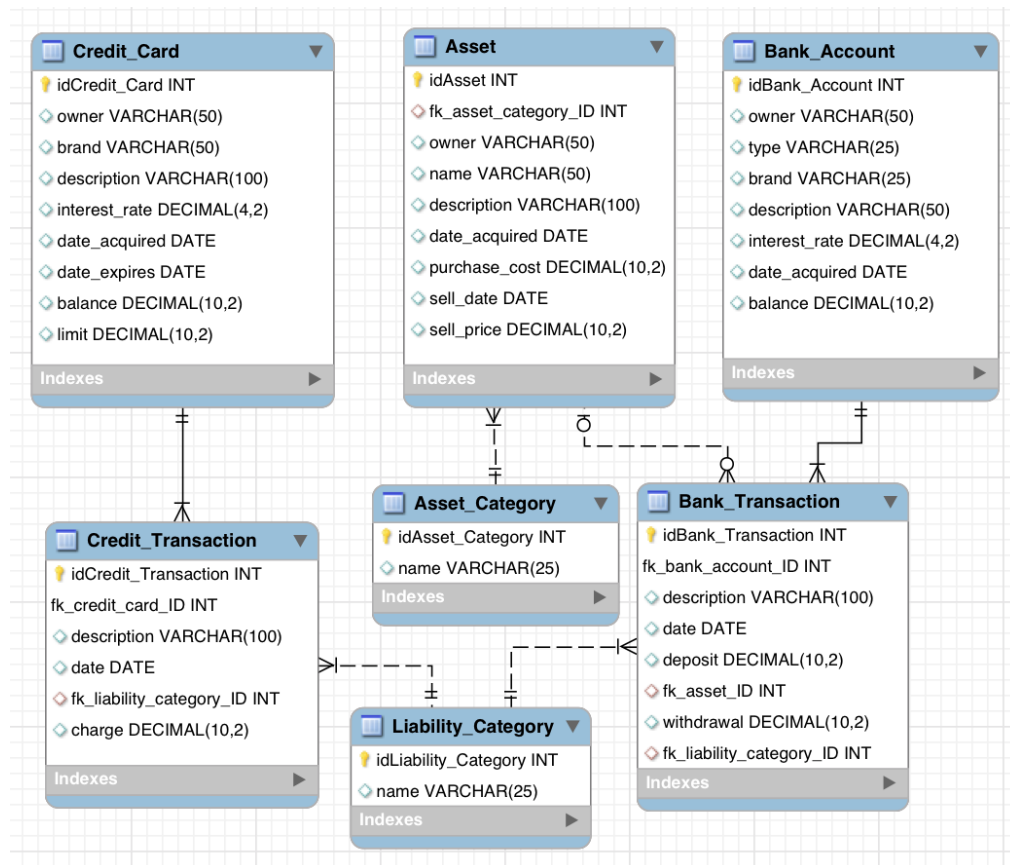


Figure 1: EER Diagram

### 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". On the left is a sidebar 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 the following fields and controls:

- Stored Connection:** A dropdown menu showing "Stockwell\_Financial" with a blue arrow icon. To the right is the text "Select from saved connection settings".
- Connection Method:** A dropdown menu showing "Standard (TCP/IP)" with a blue arrow icon. To the right is the text "Method to use to connect to the RDBMS".
- Below these are three tabs: "Parameters" (active, highlighted in blue), "SSL", and "Advanced".
- Hostname:** A text field containing "127.0.0.1".
- Port:** A text field containing "3306". To the right is the text "Name or IP address of the server host - and TCP/IP port."
- Username:** A text field containing "root". To the right is the text "Name of the user to connect with."
- Password:** A text field with two buttons: "Store in Keychain ..." and "Clear". To the right is the text "The user's password. Will be requested later if it's not set."
- Default Schema:** An empty text field. To the right is the text "The schema to use as default schema. Leave blank to select it later."

At the bottom right of the window are two buttons: "Go Back" and "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!**

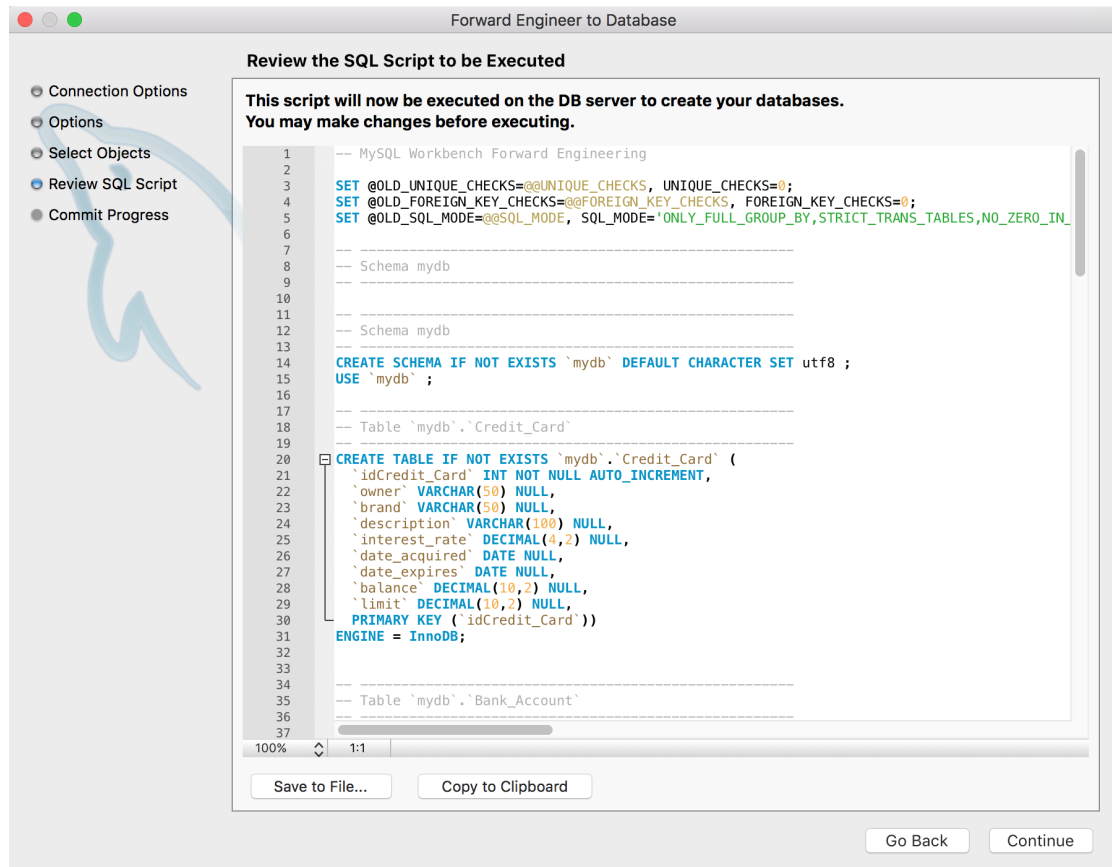


Figure 3: Forward Engineer code



### 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.

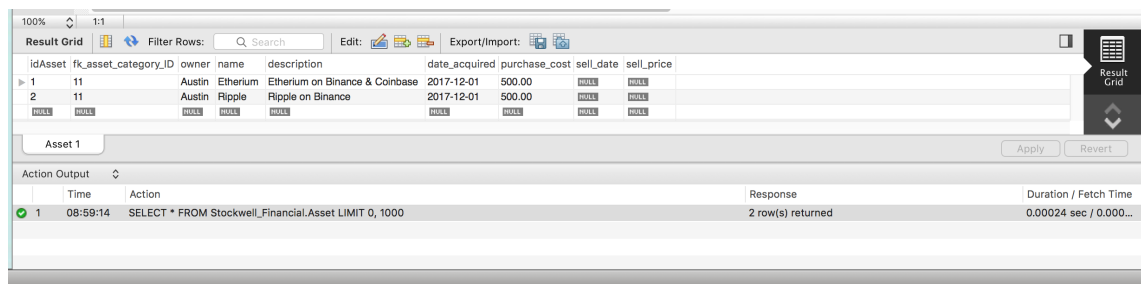
- **Deleting an Asset Category deletes corresponding Assets**
- **Deleting a Credit Card deletes corresponding Credit Transactions**
- **Deleting a Liability Category deletes corresponding Credit Transactions**
- **Deleting a Bank Account deletes corresponding Bank Transactions**
- **Deleting an Asset 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 window. At the top, there's a toolbar with 'Result Grid', 'Filter Rows', 'Search', 'Edit', and 'Export/Import' buttons. Below this is a table with columns: idAsset, fk\_asset\_category\_ID, owner, name, description, date\_acquired, purchase\_cost, sell\_date, and sell\_price. The table contains two rows of data. Below the table, there's a section labeled 'Asset 1' with 'Apply' and 'Revert' buttons. At the bottom, there's an 'Action Output' section with columns: Time, Action, Response, and Duration / Fetch Time. It 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).

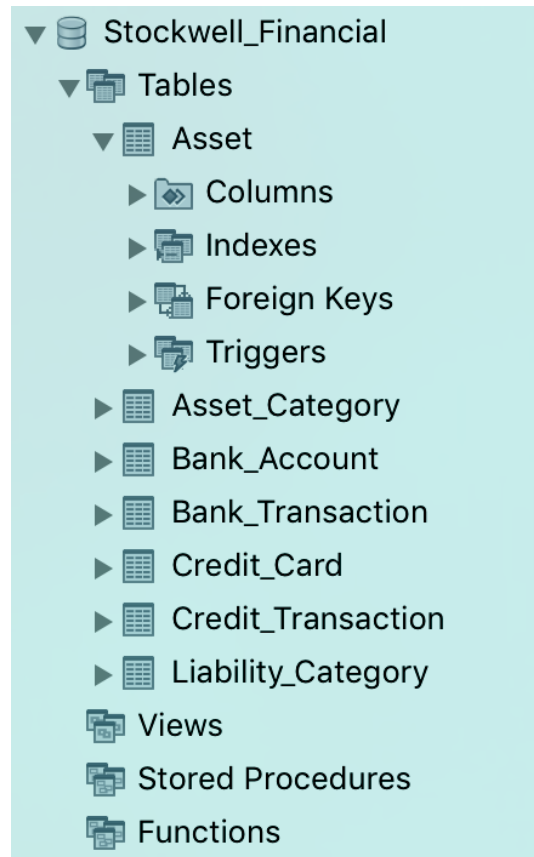


Figure 15: Database structure tree shown in MySQL Workbench



## 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.