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

List of Figures

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 convienient 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 Eletric Bill	2020-02-10	25.00	Electricity

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.

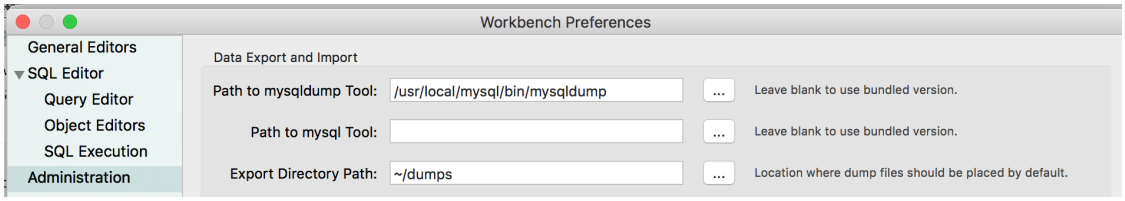

Figure 18: MySQL DUMP Directory

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.

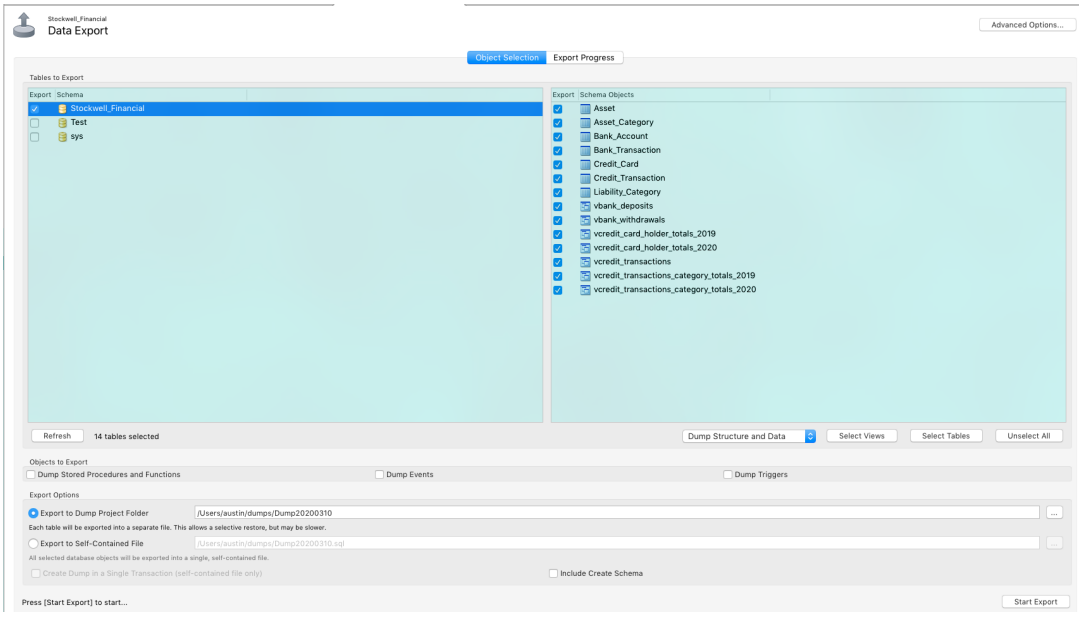

Figure 19: MySQL Export Window

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:

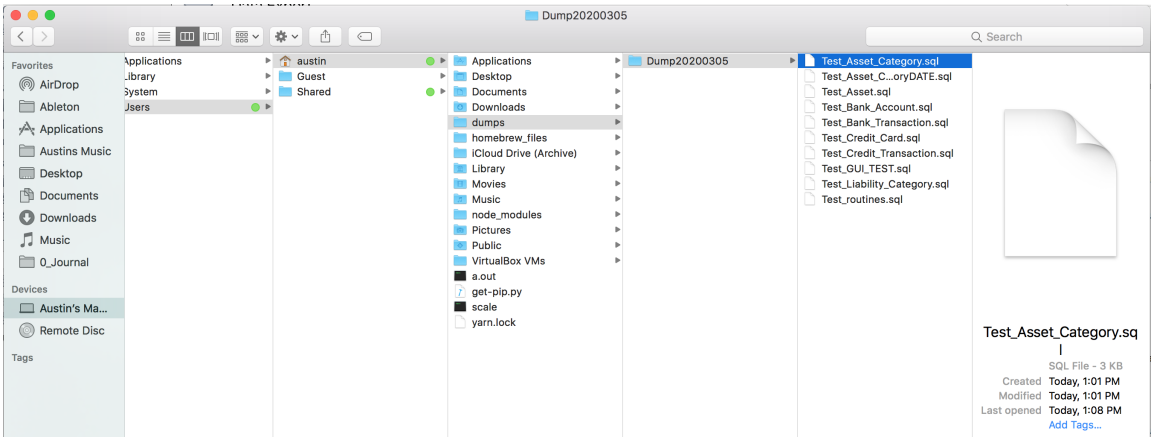

Figure 20: MySQL DUMP Directory in Finder

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/