IFT 530 – Advanced Database Management Systems

Group 7

Final Project

Rahul Muddhapuram

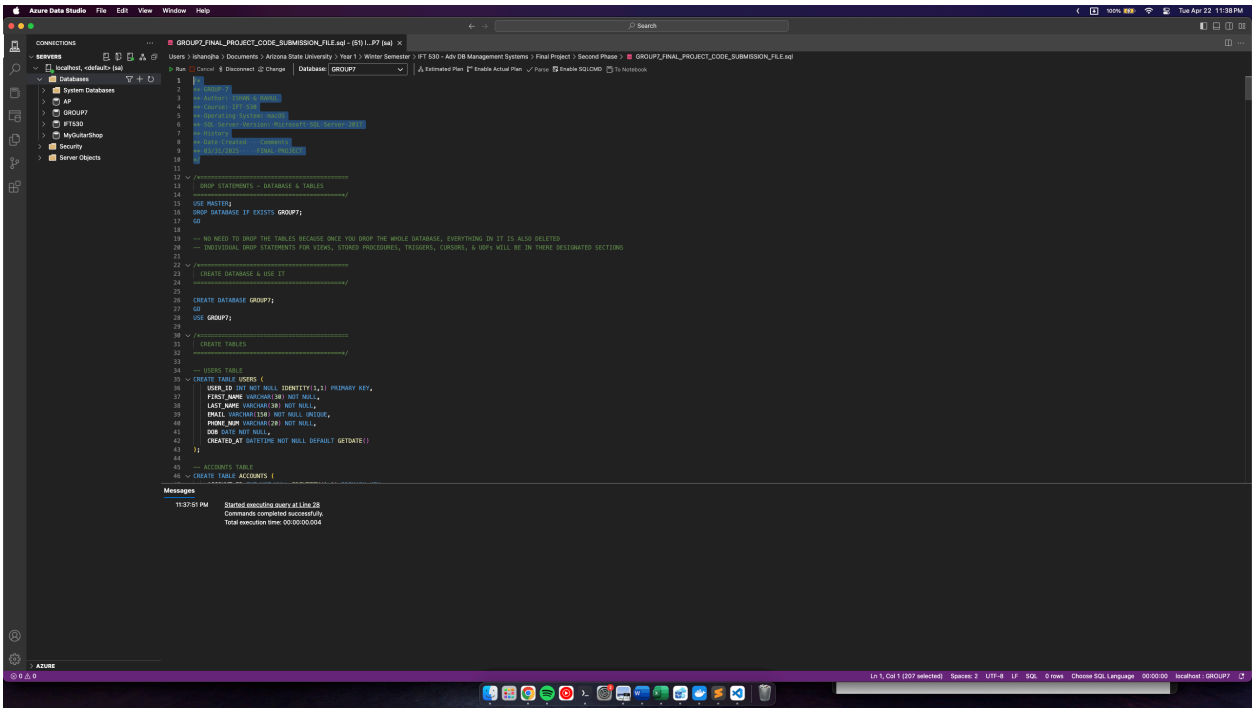
Ishan Ojha

Instructor: Ashish Gulati

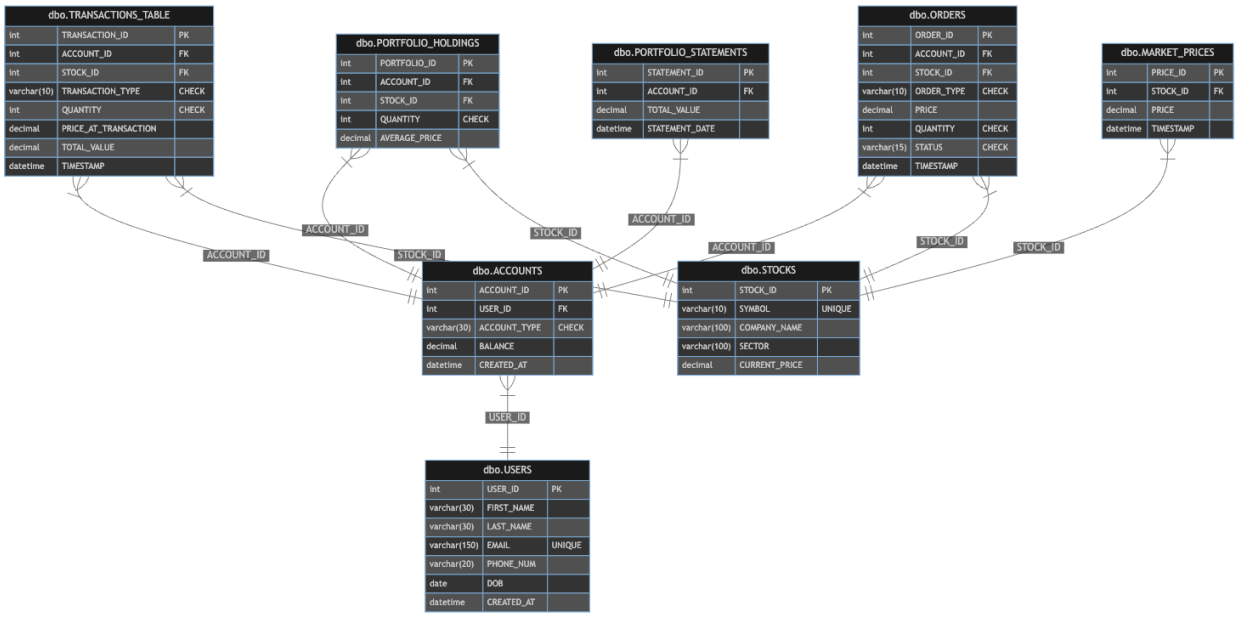
Date of Submission: May 2nd, 2025

SCREENSHOT WITH HEADER

/*
** GROUP 7
** Author: ISHAN & RAHUL
** Course: IFT 530
** Operating System: macOS
** SQL Server Version: Microsoft SQL Server 2017
** History
** Date Created Comments
** 03/31/2025 FINAL PROJECT
*/



ERD DIAGRAM



CREATION OF TABLES

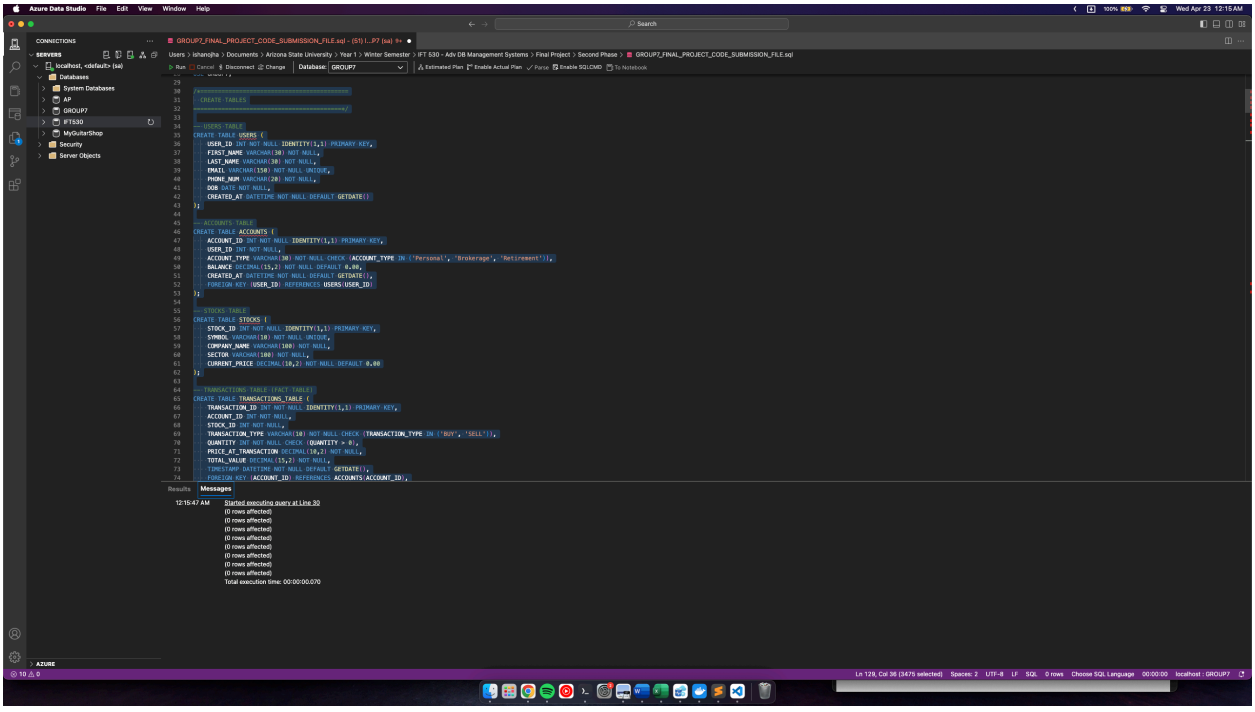
CODE

```
-- USERS TABLE
CREATE TABLE USERS (
    USER_ID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
    FIRST_NAME VARCHAR(30) NOT NULL,
    LAST_NAME VARCHAR(30) NOT NULL,
    EMAIL VARCHAR(150) NOT NULL UNIQUE,
    PHONE_NUM VARCHAR(20) NOT NULL,
    DOB DATE NOT NULL,
    CREATED_AT DATETIME NOT NULL DEFAULT GETDATE()
);

-- ACCOUNTS TABLE
CREATE TABLE ACCOUNTS (
    ACCOUNT_ID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
    USER_ID INT NOT NULL,
    ACCOUNT_TYPE VARCHAR(30) NOT NULL CHECK (ACCOUNT_TYPE IN ('Personal',
'Brokerage', 'Retirement')),
    BALANCE DECIMAL(15,2) NOT NULL DEFAULT 0.00,
    CREATED_AT DATETIME NOT NULL DEFAULT GETDATE(),
    FOREIGN KEY (USER_ID) REFERENCES USERS(USER_ID)
```

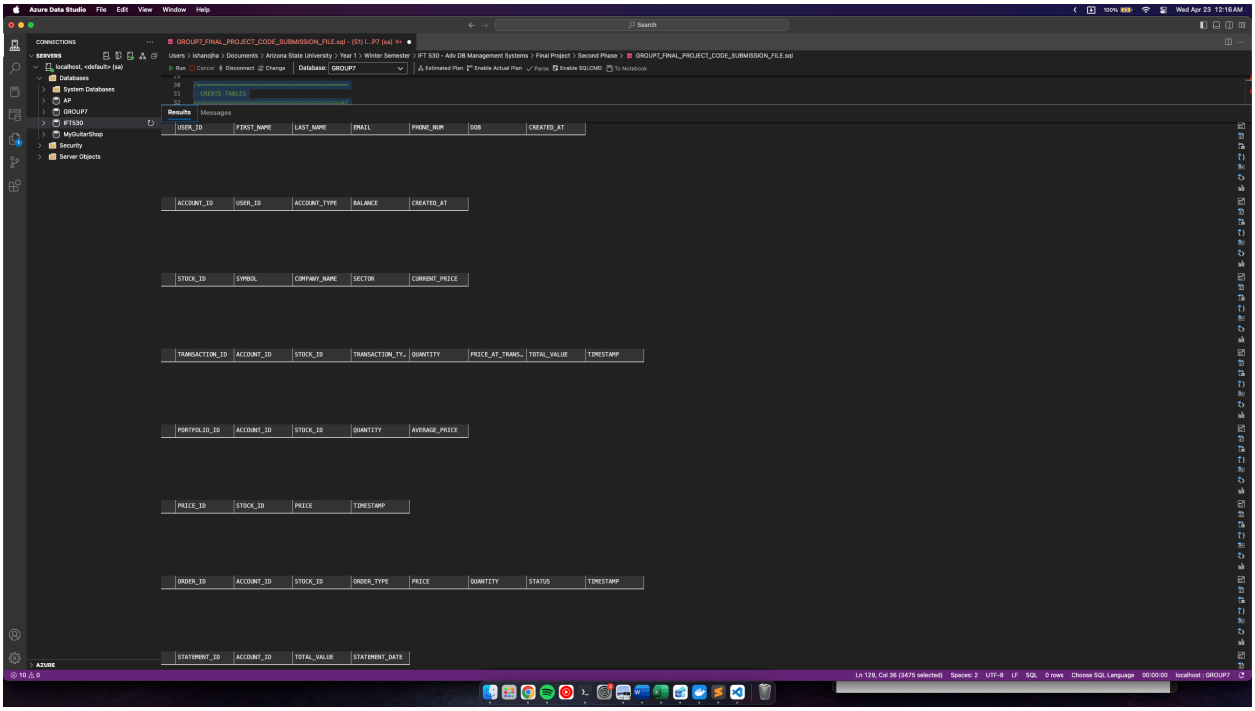
```
);

-- STOCKS TABLE
CREATE TABLE STOCKS (
    STOCK_ID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
    SYMBOL VARCHAR(10) NOT NULL UNIQUE,
    COMPANY_NAME VARCHAR(100) NOT NULL,
    SECTOR VARCHAR(100) NOT NULL,
    CURRENT_PRICE DECIMAL(10,2) NOT NULL DEFAULT 0.00
);

-- TRANSACTIONS TABLE (FACT TABLE)
CREATE TABLE TRANSACTIONS_TABLE (
    TRANSACTION_ID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
    ACCOUNT_ID INT NOT NULL,
    STOCK_ID INT NOT NULL,
    TRANSACTION_TYPE VARCHAR(10) NOT NULL CHECK (TRANSACTION_TYPE IN ('BUY',
'SELL')),
    QUANTITY INT NOT NULL CHECK (QUANTITY > 0),
    PRICE_AT_TRANSACTION DECIMAL(10,2) NOT NULL,
    TOTAL_VALUE DECIMAL(15,2) NOT NULL,
    TIMESTAMP DATETIME NOT NULL DEFAULT GETDATE(),
    FOREIGN KEY (ACCOUNT_ID) REFERENCES ACCOUNTS(ACCOUNT_ID),
    FOREIGN KEY (STOCK_ID) REFERENCES STOCKS(STOCK_ID)
);

-- PORTFOLIO_HOLDINGS TABLE
CREATE TABLE PORTFOLIO_HOLDINGS (
    PORTFOLIO_ID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
    ACCOUNT_ID INT NOT NULL,
    STOCK_ID INT NOT NULL,
    QUANTITY INT NOT NULL CHECK (QUANTITY >= 0),
    AVERAGE_PRICE DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (ACCOUNT_ID) REFERENCES ACCOUNTS(ACCOUNT_ID),
    FOREIGN KEY (STOCK_ID) REFERENCES STOCKS(STOCK_ID)
);

-- MARKET_PRICES TABLE (FACT TABLE)
CREATE TABLE MARKET_PRICES (
    PRICE_ID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
    STOCK_ID INT NOT NULL,
    PRICE DECIMAL(10,2) NOT NULL,
    TIMESTAMP DATETIME NOT NULL DEFAULT GETDATE(),
    FOREIGN KEY (STOCK_ID) REFERENCES STOCKS(STOCK_ID)
```

```
);

-- ORDERS TABLE (FACT TABLE)
CREATE TABLE ORDERS (
    ORDER_ID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
    ACCOUNT_ID INT NOT NULL,
    STOCK_ID INT NOT NULL,
    ORDER_TYPE VARCHAR(10) NOT NULL CHECK (ORDER_TYPE IN ('Market', 'Limit')),
    PRICE DECIMAL(10,2) NOT NULL,
    QUANTITY INT NOT NULL CHECK (QUANTITY > 0),
    STATUS VARCHAR(15) NOT NULL CHECK (STATUS IN ('Pending', 'Executed', 'Canceled')),
    TIMESTAMP DATETIME NOT NULL DEFAULT GETDATE(),
    FOREIGN KEY (ACCOUNT_ID) REFERENCES ACCOUNTS(ACCOUNT_ID),
    FOREIGN KEY (STOCK_ID) REFERENCES STOCKS(STOCK_ID)
);

-- PORTFOLIO_STATEMENTS TABLE
CREATE TABLE PORTFOLIO_STATEMENTS (
    STATEMENT_ID INT IDENTITY(1,1) PRIMARY KEY,
    ACCOUNT_ID INT NOT NULL,
    TOTAL_VALUE DECIMAL(18,2),
    STATEMENT_DATE DATETIME DEFAULT GETDATE(),
    FOREIGN KEY (ACCOUNT_ID) REFERENCES ACCOUNTS(ACCOUNT_ID)
);
```

SUCCESSFUL EXECUTION OF SCRIPT



SCREENSHOTS OF TABLES CREATED



POPULATING TABLES

CODE

-- (CRITERIA: 10 IN DIMENSION TABLES & 20-50 ROWS IN THE REST OF THE TABLES) --

-- INSERTING INTO USERS TABLE
INSERT INTO USERS (FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUM, DOB) VALUES
('Ishan', 'Ojha', 'ishan.ojha@example.com', '123-456-7890', '2000-01-31'),
('Komal', 'Patel', 'komal.patel@example.com', '234-567-8901', '2000-04-11'),
('Behzad', 'Hakim', 'behzad.hakim@example.com', '345-678-9012', '2000-03-09'),
('Rahul', 'Muddhapuram', 'rahul.muddhapuram@example.com', '456-789-0123', '2000-09-18'),
('Bhargav', 'Limbasia', 'bhargav.limbasia@example.com', '567-890-1234', '2001-11-30'),
('Lovepreet', 'Arora', 'lovepreet.arora@example.com', '678-901-2345', '2002-02-25'),
('Updesh', 'Surdhar', 'updesh.surdhar@example.com', '789-012-3456', '2000-08-05'),
('Dhurv', 'Prajapati', 'dhurv.prajapati@example.com', '890-123-4567', '2001-06-12'),
('Faisal', 'Wakili', 'faisal.wakili@example.com', '901-234-5678', '2002-01-28'),
('Parimal', 'Amodkar', 'parimal.amodkar@example.com', '012-345-6789', '2000-04-09'),
('Andy', 'Du-Ly', 'andy.du-ly@example.com', '112-345-6789', '2001-10-17'),
('Vijay', 'Nandlal', 'vijay.nandlal@example.com', '212-345-6789', '2002-05-25'),
('Tanishq', 'Patil', 'tanishq.patil@example.com', '312-345-6789', '2000-12-04'),
('Unnati', 'Gohil', 'unnati.gohil@example.com', '412-345-6789', '2001-03-21'),
('David', 'Hakim', 'david.hakim@example.com', '512-345-6789', '2002-07-14'),

('Amrit', 'Singh', 'amrit.singh@example.com', '612-345-6789', '2000-09-03'),
('Shobhit', 'Srivastava', 'shobhit.srivastava@example.com', '712-345-6789', '2001-01-19'),
('Jusdeep', 'Dhaliwal', 'jusdeep.dhaliwal@example.com', '812-345-6789', '2002-06-08'),
('Karen', 'Zhan', 'karen.zhan@example.com', '912-345-6789', '2000-02-27'),
('Tejas', 'Bhanushali', 'tejas.bhanushali@example.com', '013-345-6789', '2001-11-06');

SELECT * FROM USERS; -- ENSURE ALL THE DATA IS POPULATED

-- INSERTING INTO ACCOUNTS TABLE
INSERT INTO ACCOUNTS (USER_ID, ACCOUNT_TYPE, BALANCE) VALUES
(1, 'Personal', 5000.00),
(2, 'Brokerage', 10000.00),
(3, 'Retirement', 15000.00),
(4, 'Personal', 7000.00),
(5, 'Brokerage', 20000.00),
(6, 'Retirement', 25000.00),
(7, 'Personal', 8000.00),
(8, 'Brokerage', 12000.00),
(9, 'Retirement', 30000.00),
(10, 'Personal', 6000.00),
(11, 'Brokerage', 11000.00),
(12, 'Retirement', 18000.00),
(13, 'Personal', 7500.00),
(14, 'Brokerage', 9500.00),
(15, 'Retirement', 22000.00),
(16, 'Personal', 6800.00),
(17, 'Brokerage', 14000.00),
(18, 'Retirement', 27000.00),
(19, 'Personal', 5000.00),
(20, 'Brokerage', 15500.00);

SELECT * FROM ACCOUNTS; -- ENSURE ALL THE DATA IS POPULATED

-- INSERTING INTO STOCKS TABLE
INSERT INTO STOCKS (SYMBOL, COMPANY_NAME, SECTOR, CURRENT_PRICE) VALUES
('AAPL', 'Apple Inc.', 'Technology', 175.50),
('MSFT', 'Microsoft Corp.', 'Technology', 310.20),
('GOOGL', 'Alphabet Inc.', 'Technology', 2750.30),
('TSLA', 'Tesla Inc.', 'Automotive', 720.40),
('AMZN', 'Amazon.com Inc.', 'E-commerce', 3450.60),
('FB', 'Meta Platforms', 'Technology', 380.25),
('NVDA', 'NVIDIA Corp.', 'Semiconductors', 600.80),
('NFLX', 'Netflix Inc.', 'Entertainment', 500.55),
('DIS', 'Walt Disney Co.', 'Entertainment', 180.90),

('BA', 'Boeing Co.', 'Aerospace', 210.35),
('AMD', 'Advanced Micro Devices', 'Semiconductors', 130.75),
('INTC', 'Intel Corp.', 'Semiconductors', 48.50),
('PYPL', 'PayPal Holdings', 'Finance', 260.40),
('JPM', 'JPMorgan Chase & Co.', 'Finance', 155.80),
('V', 'Visa Inc.', 'Finance', 230.90),
('MA', 'Mastercard Inc.', 'Finance', 365.70),
('XOM', 'ExxonMobil Corp.', 'Energy', 105.25),
('CVX', 'Chevron Corp.', 'Energy', 120.40),
('WMT', 'Walmart Inc.', 'Retail', 140.35),
('TGT', 'Target Corp.', 'Retail', 135.75);

SELECT * FROM STOCKS; -- ENSURE ALL THE DATA IS POPULATED

-- INSERTING INTO TRANSACTIONS TABLE
INSERT INTO TRANSACTIONS_TABLE (ACCOUNT_ID, STOCK_ID, TRANSACTION_TYPE, QUANTITY, PRICE_AT_TRANSACTION, TOTAL_VALUE) VALUES
(1, 1, 'BUY', 10, 175.50, 1755.00),
(2, 2, 'SELL', 5, 310.20, 1551.00),
(3, 3, 'BUY', 8, 2750.30, 22002.40),
(4, 4, 'BUY', 15, 720.40, 10806.00),
(5, 5, 'SELL', 20, 3450.60, 69012.00),
(6, 6, 'BUY', 12, 380.25, 4563.00),
(7, 7, 'SELL', 10, 600.80, 6008.00),
(8, 8, 'BUY', 18, 500.55, 9009.90),
(9, 9, 'SELL', 7, 180.90, 1266.30),
(10, 10, 'BUY', 6, 210.35, 1262.10);

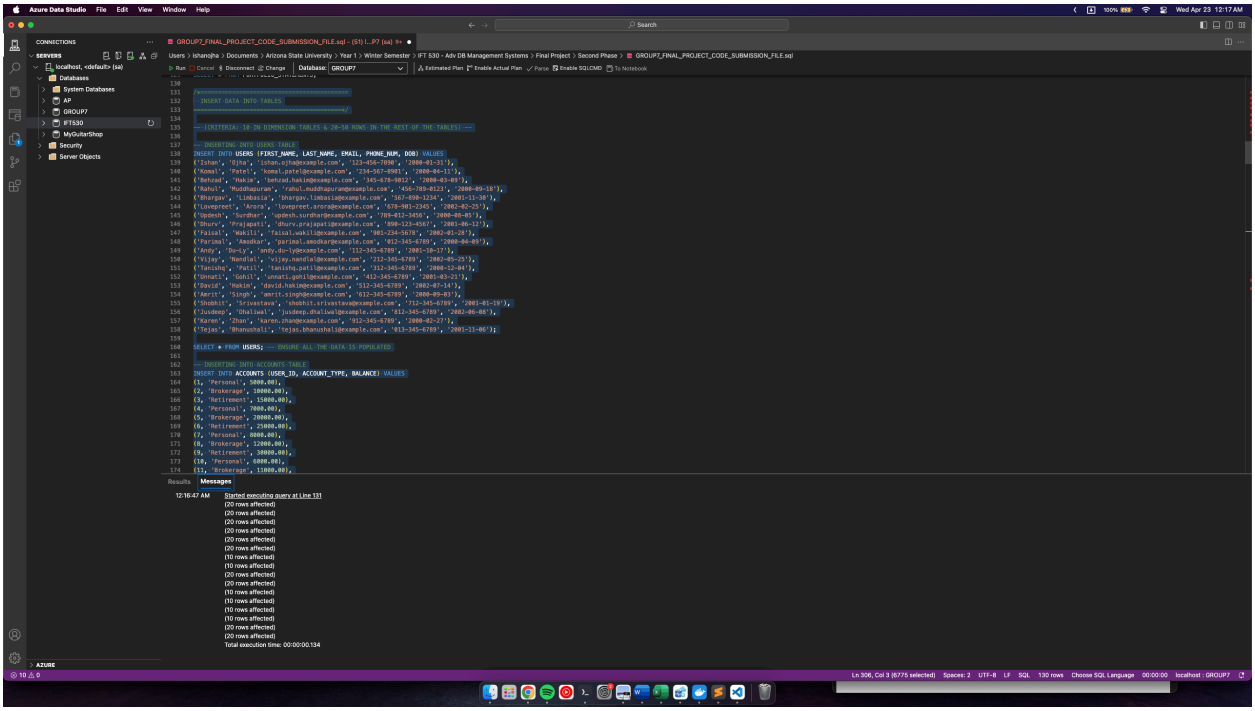
SELECT * FROM TRANSACTIONS_TABLE; -- ENSURE ALL THE DATA IS POPULATED

-- INSERTING INTO PORTFOLIO_HOLDINGS TABLE
INSERT INTO PORTFOLIO_HOLDINGS (ACCOUNT_ID, STOCK_ID, QUANTITY, AVERAGE_PRICE) VALUES
(1, 1, 100, 150.00),
(1, 2, 50, 120.50),
(2, 1, 200, 140.00),
(2, 3, 150, 75.25),
(3, 2, 120, 110.30),
(3, 4, 75, 60.10),
(4, 1, 80, 135.00),
(4, 3, 100, 78.50),
(5, 5, 150, 200.00),
(5, 2, 250, 115.40),
(6, 6, 180, 350.75),

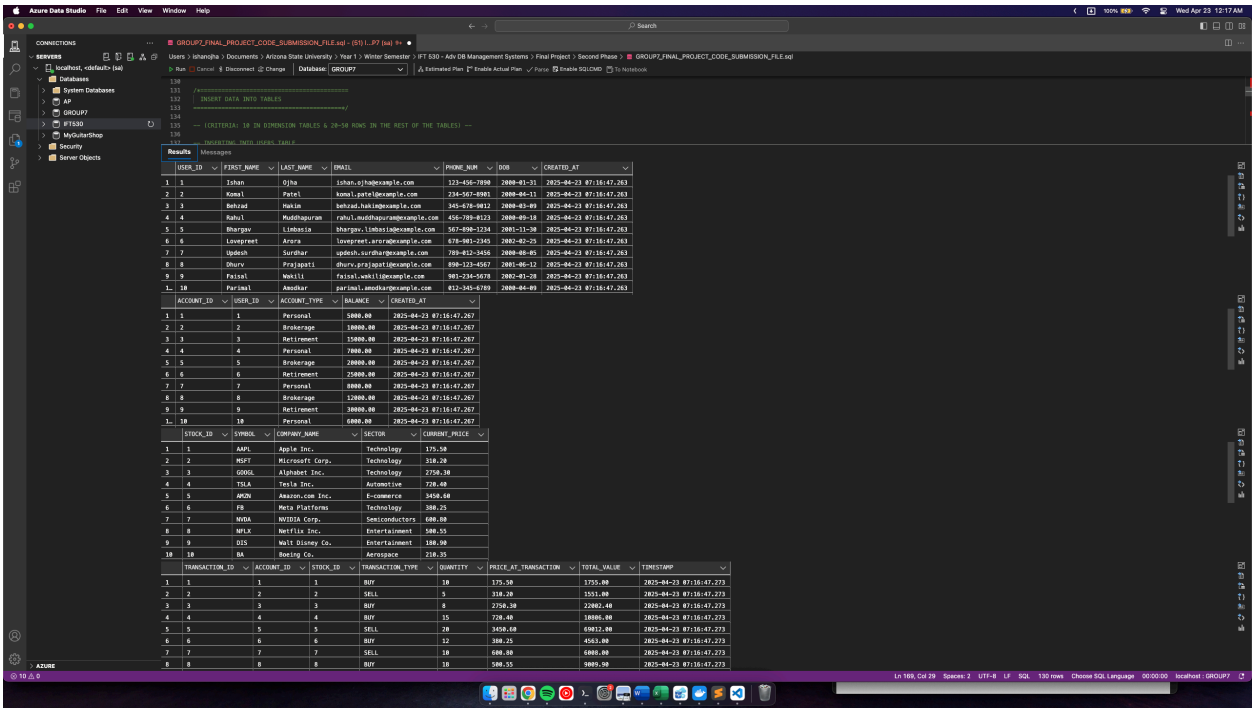
```
(6, 7, 90, 590.30),
(7, 8, 200, 480.50),
(7, 9, 120, 175.25),
(8, 10, 140, 205.10),
(8, 5, 110, 3400.00),
(9, 3, 175, 2700.50),
(9, 4, 95, 710.75),
(10, 7, 85, 600.00),
(10, 8, 130, 495.75);

SELECT * FROM PORTFOLIO_HOLDINGS; -- ENSURE ALL THE DATA IS POPULATED

-- INSERTING INTO MARKET_PRICES TABLE
INSERT INTO MARKET_PRICES (STOCK_ID, PRICE) VALUES
(1, 176.00),
(2, 312.50),
(3, 2755.00),
(4, 725.00),
(5, 3460.00),
(6, 385.00),
(7, 605.00),
(8, 505.00),
(9, 185.00),
(10, 215.00);

SELECT * FROM MARKET_PRICES; -- ENSURE ALL THE DATA IS POPULATED

-- INSERTING INTO ORDERS TABLE
INSERT INTO ORDERS (ACCOUNT_ID, STOCK_ID, ORDER_TYPE, PRICE, QUANTITY,
STATUS) VALUES
(1, 1, 'Market', 175.50, 10, 'Executed'),
(2, 2, 'Limit', 315.00, 5, 'Pending'),
(3, 3, 'Market', 2750.00, 8, 'Executed'),
(4, 4, 'Limit', 730.00, 15, 'Canceled'),
(5, 5, 'Market', 3455.00, 20, 'Executed'),
(6, 6, 'Limit', 390.00, 12, 'Pending'),
(7, 7, 'Market', 605.00, 10, 'Executed'),
(8, 8, 'Limit', 510.00, 18, 'Pending'),
(9, 9, 'Market', 190.00, 7, 'Executed'),
(10, 10, 'Limit', 220.00, 6, 'Canceled');

SELECT * FROM ORDERS; -- ENSURE ALL THE DATA IS POPULATED

-- INSERTING INTO PORTFOLIO_STATEMENTS TABLE
```

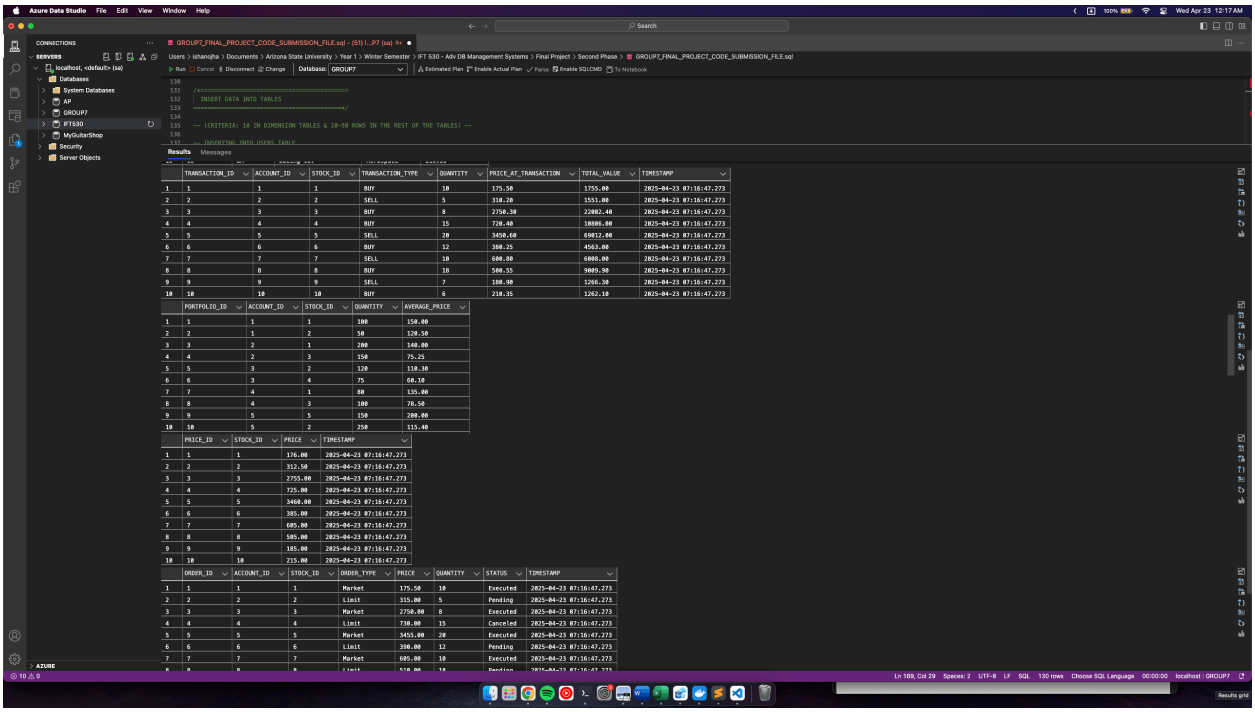
```
INSERT INTO PORTFOLIO_STATEMENTS (ACCOUNT_ID, TOTAL_VALUE,
STATEMENT_DATE) VALUES
(1, 17500.00, '2025-01-31'),
(2, 18500.00, '2025-01-31'),
(3, 22000.00, '2025-01-31'),
(4, 19500.00, '2025-01-31'),
(5, 23000.00, '2025-01-31'),
(6, 25500.00, '2025-01-31'),
(7, 27000.00, '2025-02-28'),
(8, 19000.00, '2025-02-28'),
(9, 21000.00, '2025-02-28'),
(10, 23000.00, '2025-02-28'),
(11, 24500.00, '2025-02-28'),
(12, 21500.00, '2025-02-28'),
(13, 26000.00, '2025-03-31'),
(14, 23500.00, '2025-03-31'),
(15, 25000.00, '2025-03-31'),
(16, 26500.00, '2025-03-31'),
(17, 24000.00, '2025-03-31'),
(18, 25500.00, '2025-03-31'),
(19, 27000.00, '2025-03-31'),
(20, 25000.00, '2025-03-31');

SELECT * FROM PORTFOLIO_STATEMENTS; -- ENSURE ALL THE DATA IS POPULATED
GO
```

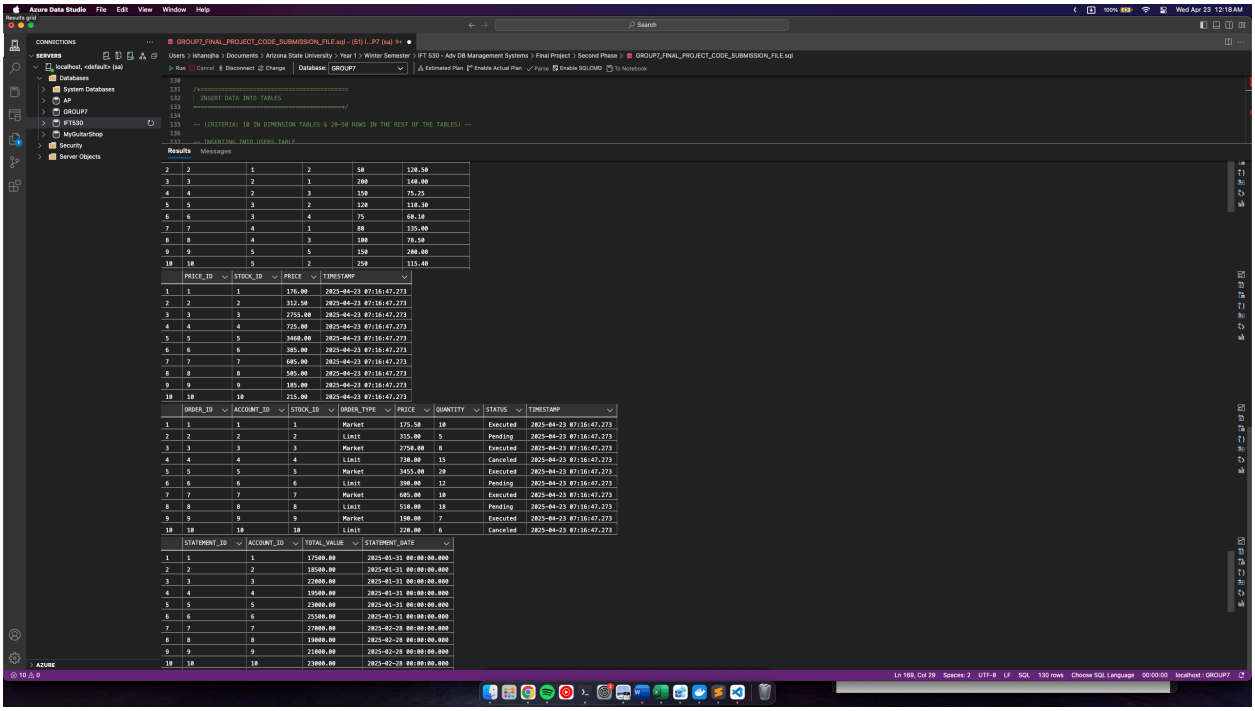
SUCCESSFUL EXECUTION OF SCRIPT



SCREENSHOTS OF POPULATED TABLES







VIEWS

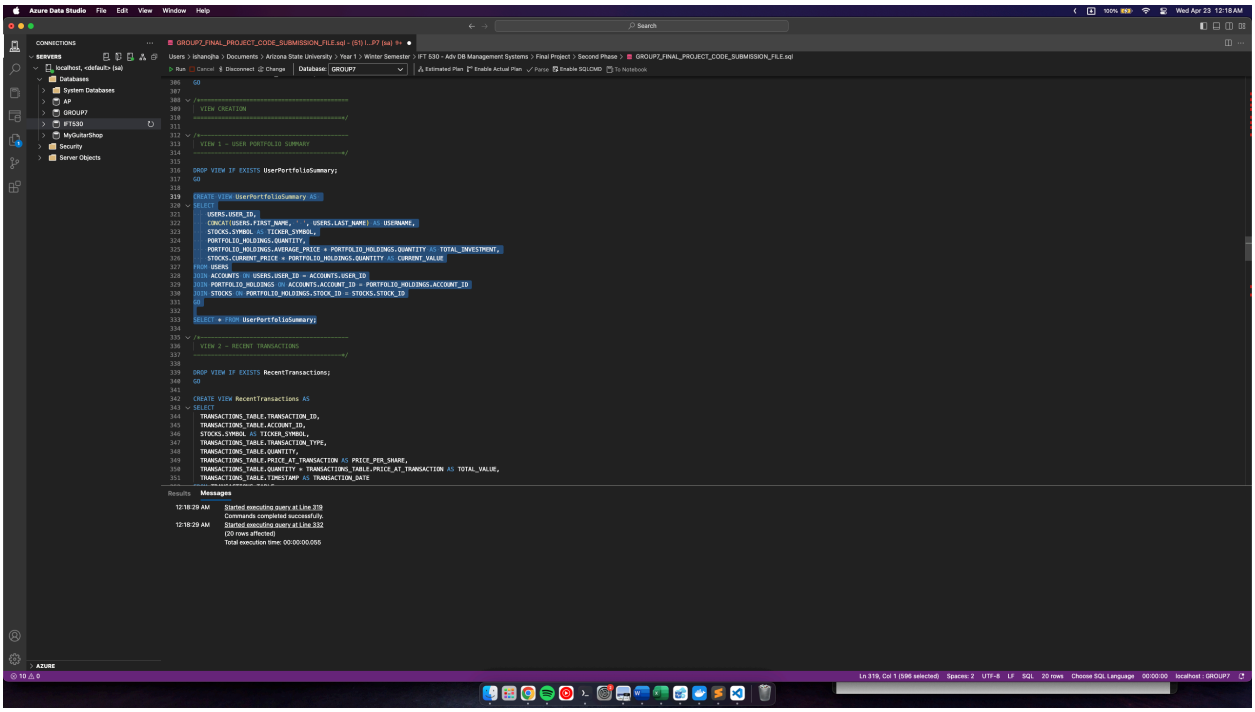
CODE – VIEW 1

/*--

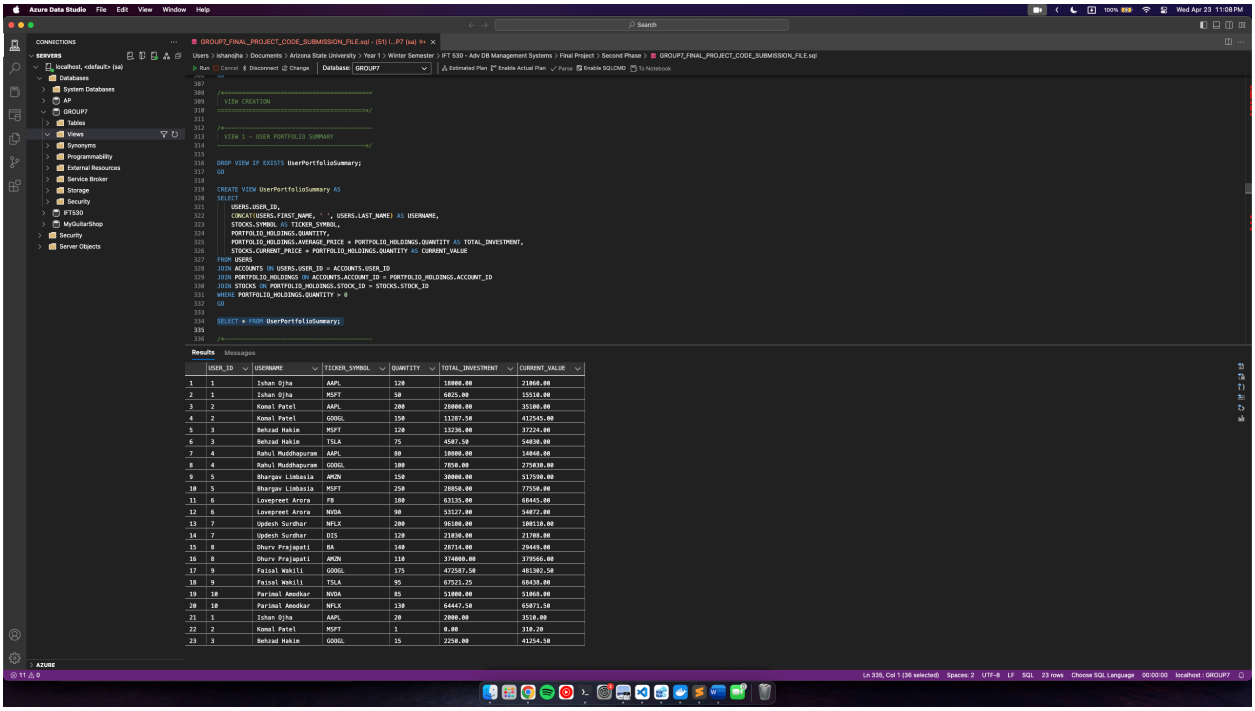
VIEW 1 - USER PORTFOLIO SUMMARY

```
-----------------------------------------*/

DROP VIEW IF EXISTS UserPortfolioSummary;
GO

CREATE VIEW UserPortfolioSummary AS
SELECT
   USERS.USER_ID,
   CONCAT(USERS.FIRST_NAME, ' ', USERS.LAST_NAME) AS USERNAME,
   STOCKS.SYMBOL AS TICKER_SYMBOL,
   PORTFOLIO_HOLDINGS.QUANTITY,
   PORTFOLIO_HOLDINGS.AVERAGE_PRICE * PORTFOLIO_HOLDINGS.QUANTITY AS
TOTAL_INVESTMENT,
   STOCKS.CURRENT_PRICE * PORTFOLIO_HOLDINGS.QUANTITY AS CURRENT_VALUE
FROM USERS
JOIN ACCOUNTS ON USERS.USER_ID = ACCOUNTS.USER_ID
JOIN PORTFOLIO_HOLDINGS ON ACCOUNTS.ACCOUNT_ID =
PORTFOLIO_HOLDINGS.ACCOUNT_ID
JOIN STOCKS ON PORTFOLIO_HOLDINGS.STOCK_ID = STOCKS.STOCK_ID
WHERE PORTFOLIO_HOLDINGS.QUANTITY > 0
GO

SELECT * FROM UserPortfolioSummary;
```

SUCCESSFUL EXECUTION OF VIEW 1



SCREENSHOTS OF TEST CASES – VIEW 1

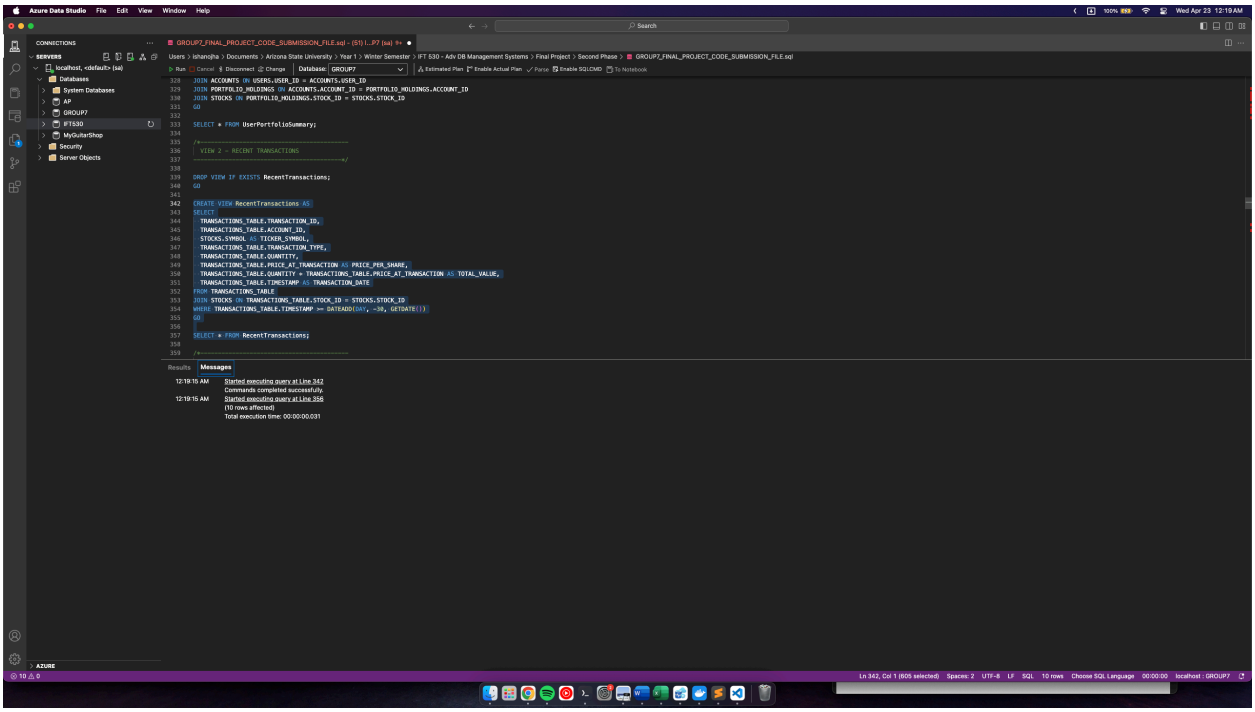


CODE – VIEW 2

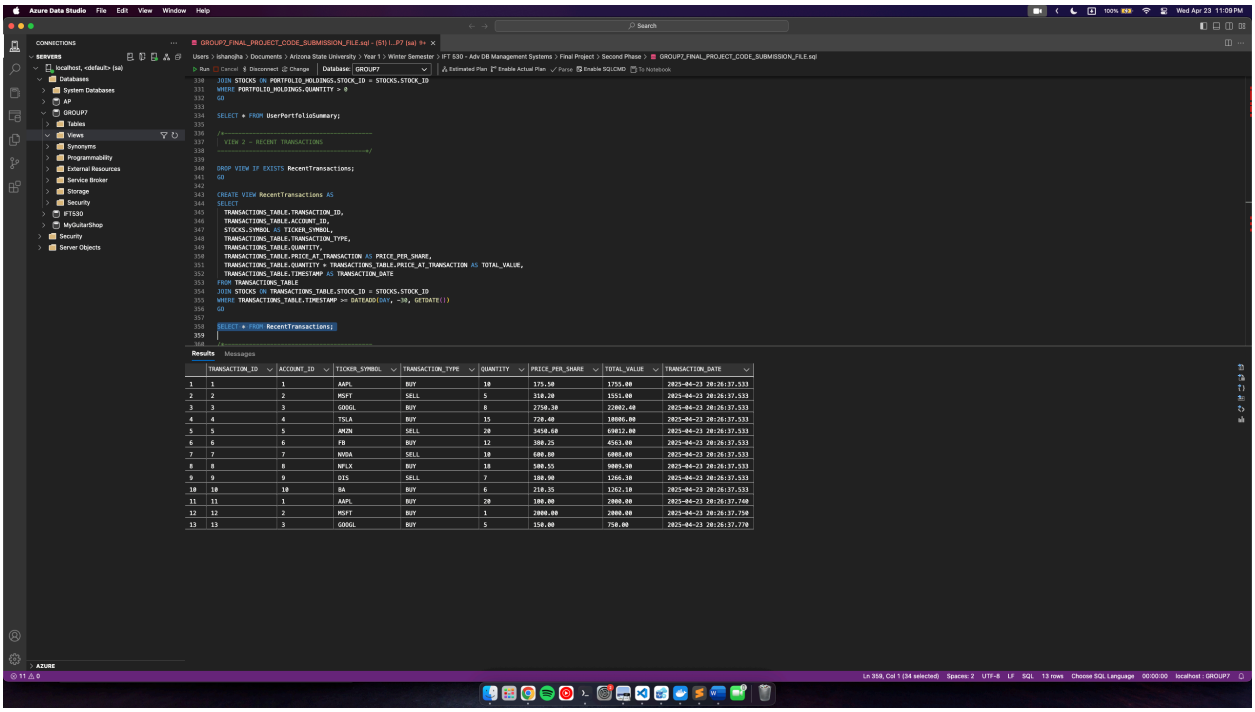
```
/*----------------------------------------
  VIEW 2 - RECENT TRANSACTIONS
----------------------------------------*/

DROP VIEW IF EXISTS RecentTransactions;
GO

CREATE VIEW RecentTransactions AS
SELECT
  TRANSACTIONS_TABLE.TRANSACTION_ID,
  TRANSACTIONS_TABLE.ACCOUNT_ID,
  STOCKS.SYMBOL AS TICKER_SYMBOL,
  TRANSACTIONS_TABLE.TRANSACTION_TYPE,
  TRANSACTIONS_TABLE.QUANTITY,
  TRANSACTIONS_TABLE.PRICE_AT_TRANSACTION AS PRICE_PER_SHARE,
  TRANSACTIONS_TABLE.QUANTITY * TRANSACTIONS_TABLE.PRICE_AT_TRANSACTION
AS TOTAL_VALUE,
  TRANSACTIONS_TABLE.TIMESTAMP AS TRANSACTION_DATE
FROM TRANSACTIONS_TABLE
JOIN STOCKS ON TRANSACTIONS_TABLE.STOCK_ID = STOCKS.STOCK_ID
WHERE TRANSACTIONS_TABLE.TIMESTAMP >= DATEADD(DAY, -30, GETDATE())
GO
```

SELECT * FROM RecentTransactions;

SUCCESSFUL EXECUTION OF VIEW 2



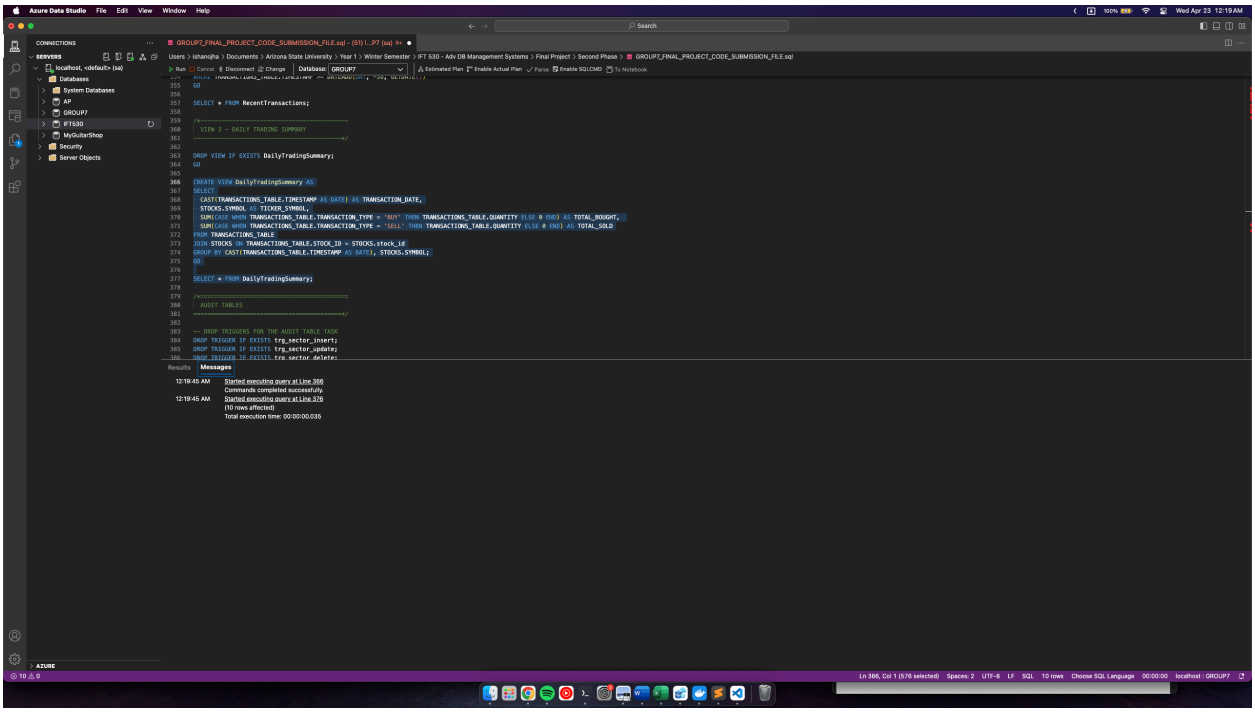
SCREENSHOTS OF TEST CASES – VIEW 2



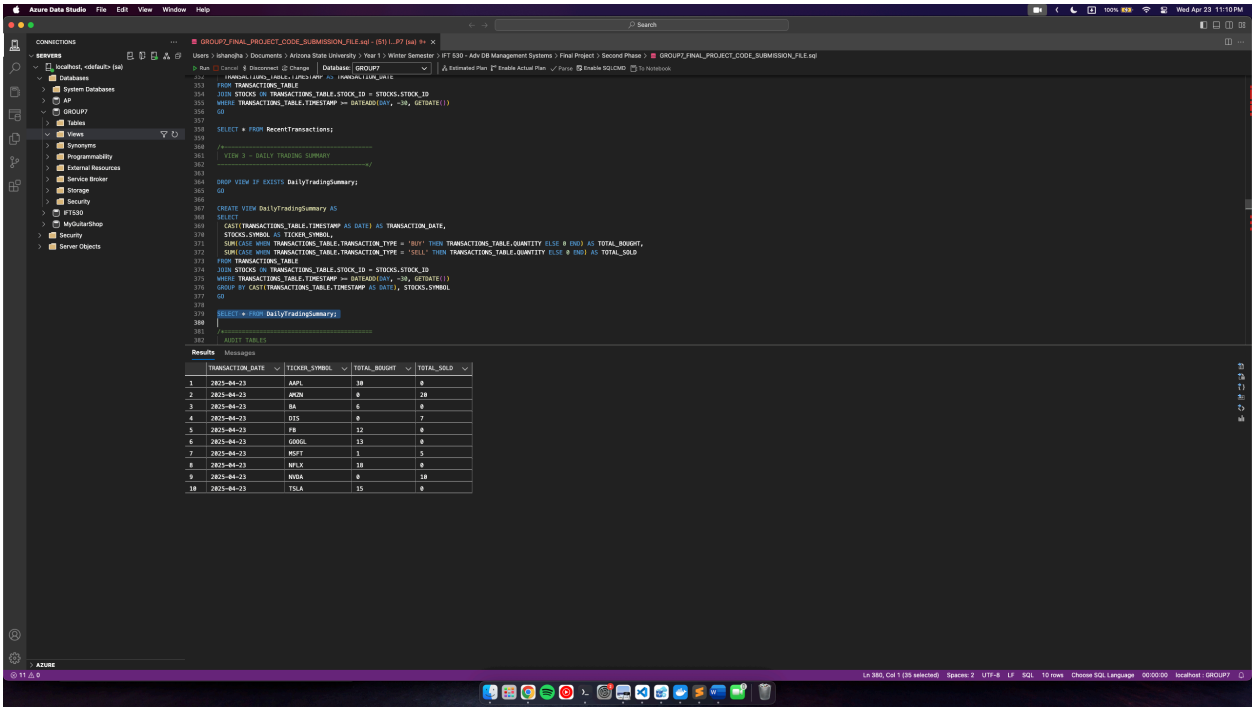
CODE – VIEW 3

```
/*----------------------------------------
  VIEW 3 - DAILY TRADING SUMMARY
----------------------------------------*/

DROP VIEW IF EXISTS DailyTradingSummary;
GO

CREATE VIEW DailyTradingSummary AS
SELECT
  CAST(TRANSACTIONS_TABLE.TIMESTAMP AS DATE) AS TRANSACTION_DATE,
  STOCKS.SYMBOL AS TICKER_SYMBOL,
  SUM(CASE WHEN TRANSACTIONS_TABLE.TRANSACTION_TYPE = 'BUY' THEN
TRANSACTIONS_TABLE.QUANTITY ELSE 0 END) AS TOTAL_BOUGHT,
  SUM(CASE WHEN TRANSACTIONS_TABLE.TRANSACTION_TYPE = 'SELL' THEN
TRANSACTIONS_TABLE.QUANTITY ELSE 0 END) AS TOTAL_SOLD
FROM TRANSACTIONS_TABLE
JOIN STOCKS ON TRANSACTIONS_TABLE.STOCK_ID = STOCKS.STOCK_ID
WHERE TRANSACTIONS_TABLE.TIMESTAMP >= DATEADD(DAY, -30, GETDATE())
GROUP BY CAST(TRANSACTIONS_TABLE.TIMESTAMP AS DATE), STOCKS.SYMBOL
GO

SELECT * FROM DailyTradingSummary;
```

SUCCESSFUL EXECUTION OF VIEW 3



SCREENSHOTS OF TEST CASES – VIEW 3



AUDIT TABLE

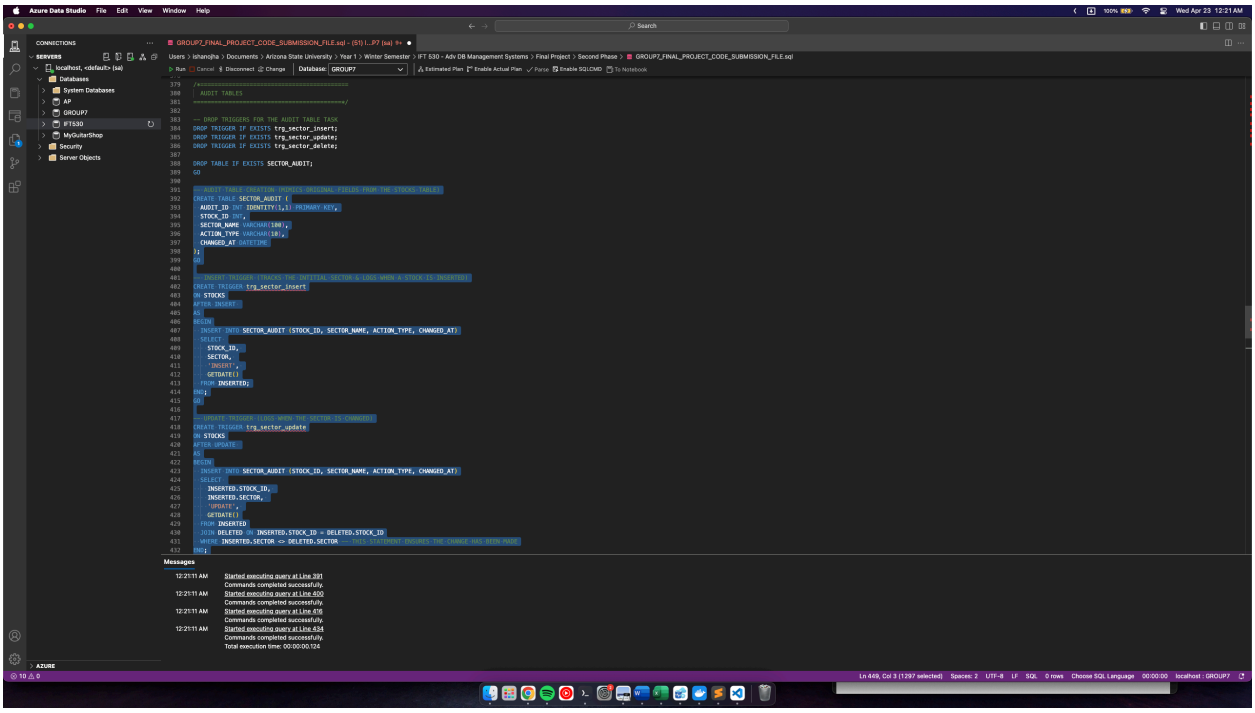
CODE – AUDIT TABLE

```
/*=============================================
  AUDIT TABLES
=============================================*/

-- DROP TRIGGERS FOR THE AUDIT TABLE TASK
DROP TRIGGER IF EXISTS trg_sector_insert;
DROP TRIGGER IF EXISTS trg_sector_update;
DROP TRIGGER IF EXISTS trg_sector_delete;

DROP TABLE IF EXISTS SECTOR_AUDIT;
GO

-- AUDIT TABLE CREATION (MIMICS ORIGINAL FIELDS FROM THE STOCKS TABLE)
CREATE TABLE SECTOR_AUDIT (
  AUDIT_ID INT IDENTITY(1,1) PRIMARY KEY,
  STOCK_ID INT,
  SECTOR_NAME VARCHAR(100),
  ACTION_TYPE VARCHAR(10),
  CHANGED_AT DATETIME
```

```
);
GO

-- INSERT TRIGGER (TRACKS THE INTITIAL SECTOR & LOGS WHEN A STOCK IS INSERTED)
CREATE TRIGGER trg_sector_insert
ON STOCKS
AFTER INSERT
AS
BEGIN
  INSERT INTO SECTOR_AUDIT (STOCK_ID, SECTOR_NAME, ACTION_TYPE, CHANGED_AT)
  SELECT
    STOCK_ID,
    SECTOR,
    'INSERT',
    GETDATE()
  FROM INSERTED;
END;
GO

-- UPDATE TRIGGER (LOGS WHEN THE SECTOR IS CHANGED)
CREATE TRIGGER trg_sector_update
ON STOCKS
AFTER UPDATE
AS
BEGIN
  INSERT INTO SECTOR_AUDIT (STOCK_ID, SECTOR_NAME, ACTION_TYPE, CHANGED_AT)
  SELECT
    INSERTED.STOCK_ID,
    INSERTED.SECTOR,
    'UPDATE',
    GETDATE()
  FROM INSERTED
  JOIN DELETED ON INSERTED.STOCK_ID = DELETED.STOCK_ID
  WHERE INSERTED.SECTOR <> DELETED.SECTOR -- THIS STATEMENT ENSURES THE
CHANGE HAS BEEN MADE
END;
GO

-- DELETE TRIGGER (LOGS WHEN SECTOR(S) ARE DELETED FROM STOCKS)
CREATE TRIGGER trg_sector_delete
ON STOCKS
AFTER DELETE
AS
BEGIN
```

```
  INSERT INTO SECTOR_AUDIT (STOCK_ID, SECTOR_NAME, ACTION_TYPE, CHANGED_AT)
  SELECT
    STOCK_ID,
    SECTOR,
    'DELETE',
    GETDATE()
  FROM DELETED
END;
GO
```

SUCCESSFUL EXECUTION OF AUDIT TABLE SCRIPT

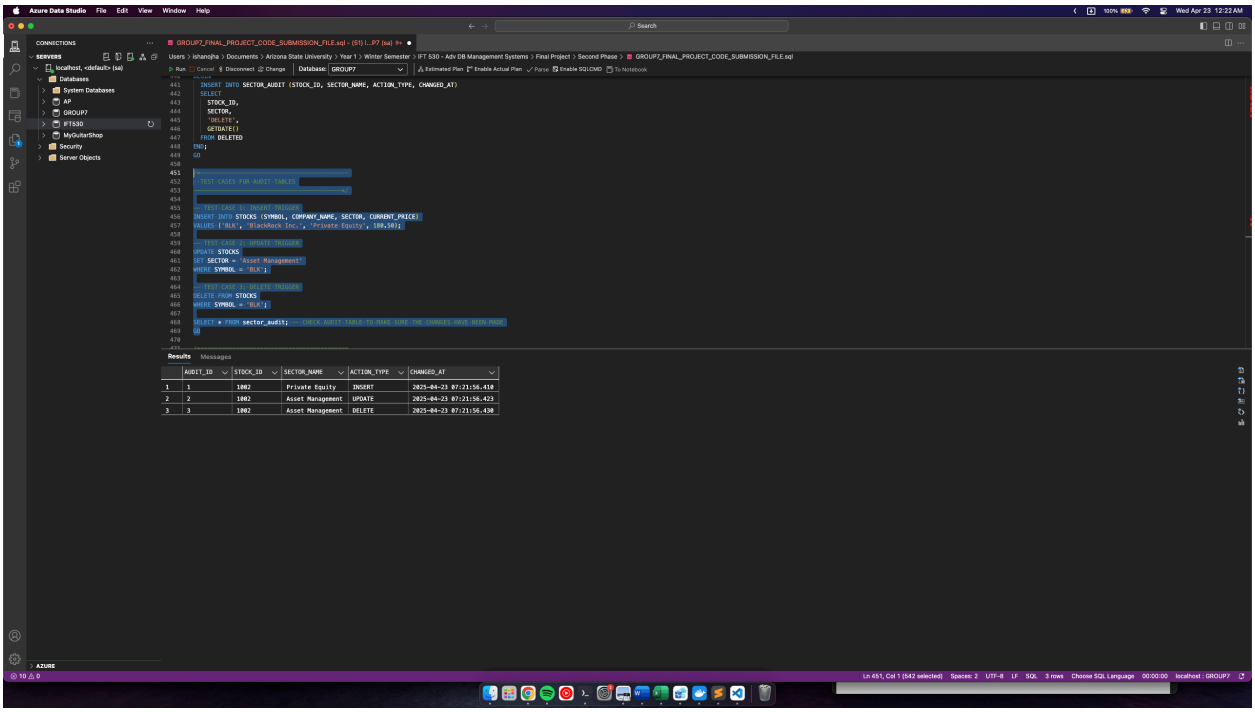


CODE – TEST CASES FOR AUDIT TABLES

```
/*----------------------------------------
  TEST CASES FOR AUDIT TABLES
----------------------------------------*/

-- TEST CASE 1: INSERT TRIGGER
INSERT INTO STOCKS (SYMBOL, COMPANY_NAME, SECTOR, CURRENT_PRICE)
VALUES ('BLK', 'BlackRock Inc.', 'Private Equity', 180.50);

-- TEST CASE 2: UPDATE TRIGGER
UPDATE STOCKS
SET SECTOR = 'Asset Management'
WHERE SYMBOL = 'BLK';
```

```
-- TEST CASE 3: DELETE TRIGGER
DELETE FROM STOCKS
WHERE SYMBOL = 'BLK';

SELECT * FROM sector_audit; -- CHECK AUDIT TABLE TO MAKE SURE THE CHANGES HAVE BEEN MADE
GO
```

SCREENSHOTS OF TEST CASES – AUDIT TABLES



STORED PROCEDURES

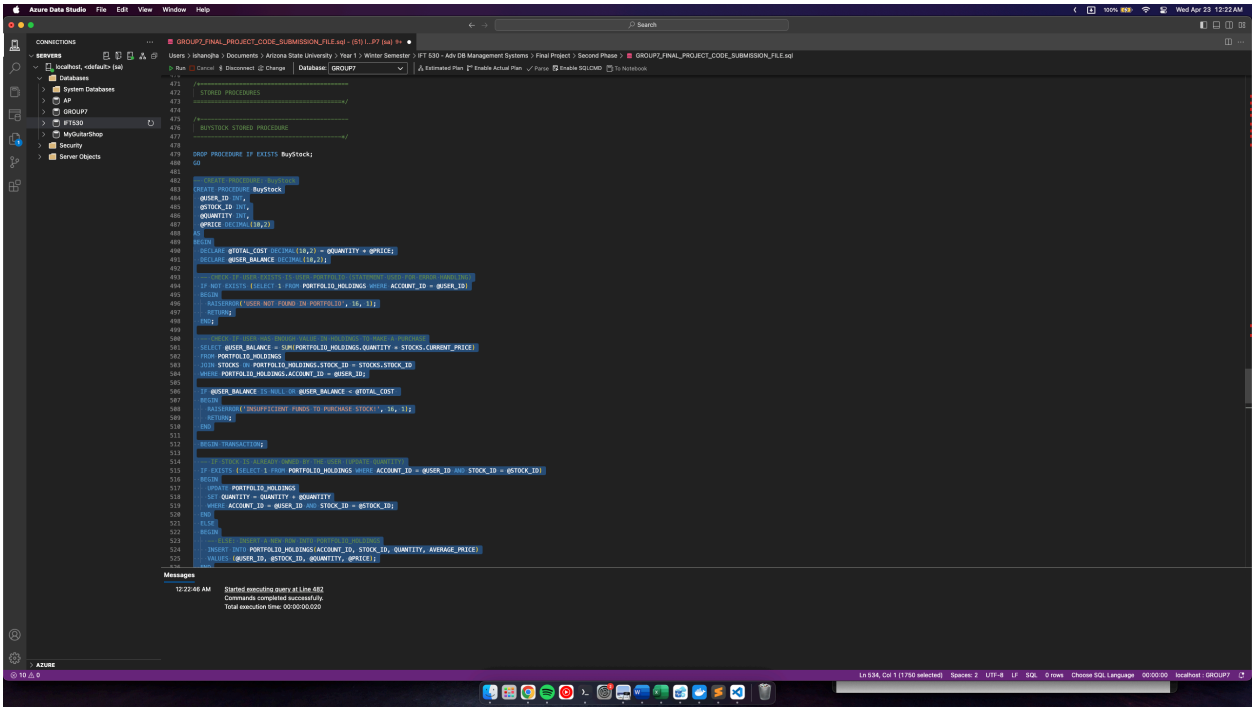
CODE – STORED PROCEDURE 1

```
/*=============================================
  STORED PROCEDURES
=============================================*/

/*-----------------------------------------
  BUYSTOCK STORED PROCEDURE
-----------------------------------------*/

DROP PROCEDURE IF EXISTS BuyStock;
GO

-- CREATE PROCEDURE: BuyStock
```

```
CREATE PROCEDURE BuyStock
  @USER_ID INT,
  @STOCK_ID INT,
  @QUANTITY INT,
  @PRICE DECIMAL(10,2)
AS
BEGIN
  DECLARE @TOTAL_COST DECIMAL(10,2) = @QUANTITY * @PRICE;
  DECLARE @USER_BALANCE DECIMAL(10,2);

  -- CHECK IF USER EXISTS IS USER PORTFOLIO (STATEMENT USED FOR ERROR
HANDLING)
  IF NOT EXISTS (SELECT 1 FROM PORTFOLIO_HOLDINGS WHERE ACCOUNT_ID =
@USER_ID)
  BEGIN
    RAISERROR('USER NOT FOUND IN PORTFOLIO', 16, 1);
    RETURN;
  END;

  -- CHECK IF USER HAS ENOUGH VALUE IN HOLDINGS TO MAKE A PURCHASE
  SELECT @USER_BALANCE = SUM(PORTFOLIO_HOLDINGS.QUANTITY *
STOCKS.CURRENT_PRICE)
  FROM PORTFOLIO_HOLDINGS
  JOIN STOCKS ON PORTFOLIO_HOLDINGS.STOCK_ID = STOCKS.STOCK_ID
  WHERE PORTFOLIO_HOLDINGS.ACCOUNT_ID = @USER_ID;

  IF @USER_BALANCE IS NULL OR @USER_BALANCE < @TOTAL_COST
  BEGIN
    RAISERROR('INSUFFICIENT FUNDS TO PURCHASE STOCK!', 16, 1);
    RETURN;
  END

  BEGIN TRANSACTION;

  -- IF STOCK IS ALREADY OWNED BY THE USER (UPDATE QUANTITY)
  IF EXISTS (SELECT 1 FROM PORTFOLIO_HOLDINGS WHERE ACCOUNT_ID = @USER_ID
AND STOCK_ID = @STOCK_ID)
  BEGIN
    UPDATE PORTFOLIO_HOLDINGS
    SET QUANTITY = QUANTITY + @QUANTITY
    WHERE ACCOUNT_ID = @USER_ID AND STOCK_ID = @STOCK_ID;
  END
  ELSE
  BEGIN
```

```
    -- ELSE: INSERT A NEW ROW INTO PORTFOLIO_HOLDINGS
    INSERT INTO PORTFOLIO_HOLDINGS(ACCOUNT_ID, STOCK_ID, QUANTITY,
AVERAGE_PRICE)
    VALUES (@USER_ID, @STOCK_ID, @QUANTITY, @PRICE);
  END

  -- INSERT TRANSACATION RECORD WITH CORRECT TOTAL_VALUE AND TIMESTAMP
  INSERT INTO TRANSACTIONS_TABLE (ACCOUNT_ID, STOCK_ID, TRANSACTION_TYPE,
QUANTITY, PRICE_AT_TRANSACTION, TOTAL_VALUE, TIMESTAMP)
  VALUES (@USER_ID, @STOCK_ID, 'BUY', @QUANTITY, @PRICE, @TOTAL_COST,
GETDATE());
  COMMIT;
END;
GO
```

SUCCESSFUL EXECUTION OF STORED PROCEDURE 1



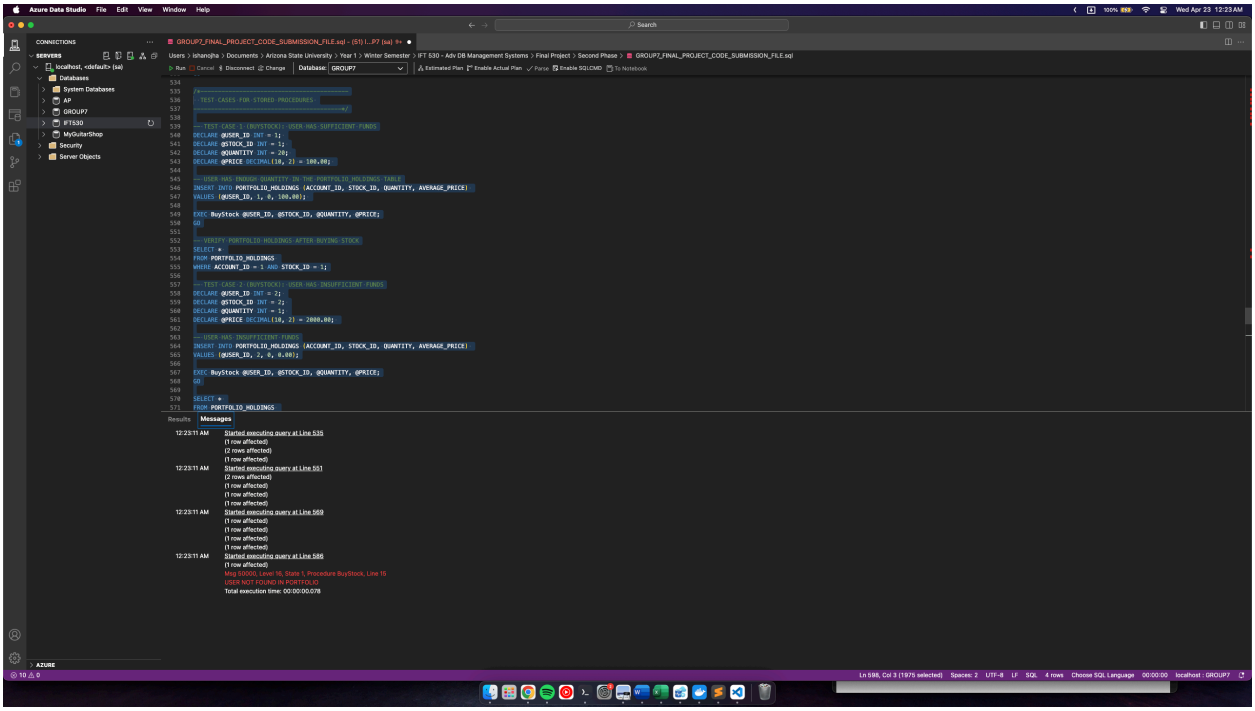
CODE – TEST CASES FOR STORED PROCEDURE 1

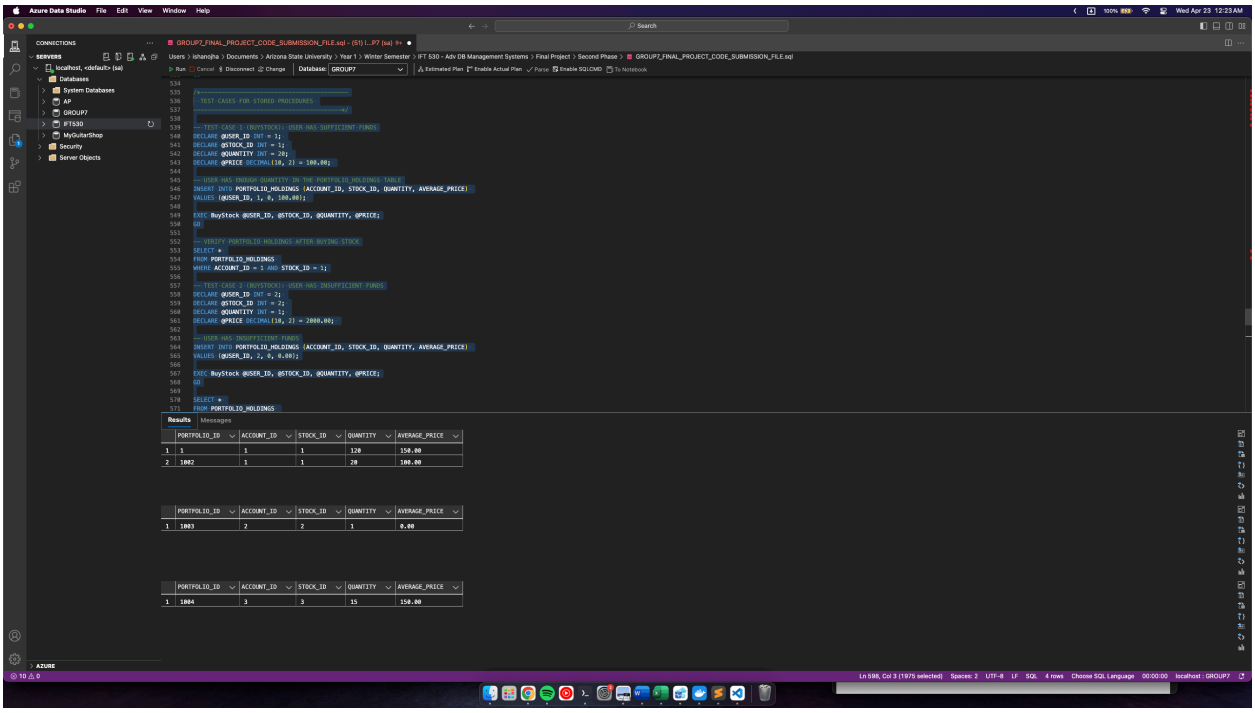
```
/*-----------------------------------------
  TEST CASES FOR STORED PROCEDURES
-----------------------------------------*/

-- TEST CASE 1 (BUYSTOCK): USER HAS SUFFICIENT FUNDS
DECLARE @USER_ID INT = 1;
DECLARE @STOCK_ID INT = 1;
```

```
DECLARE @QUANTITY INT = 20;
DECLARE @PRICE DECIMAL(10, 2) = 100.00;

-- USER HAS ENOUGH QUANTITY IN THE PORTFOLIO_HOLDINGS TABLE
INSERT INTO PORTFOLIO_HOLDINGS (ACCOUNT_ID, STOCK_ID, QUANTITY,
AVERAGE_PRICE)
VALUES (@USER_ID, 1, 0, 100.00);

EXEC BuyStock @USER_ID, @STOCK_ID, @QUANTITY, @PRICE;
GO

-- VERIFY PORTFOLIO HOLDINGS AFTER BUYING STOCK
SELECT *
FROM PORTFOLIO_HOLDINGS
WHERE ACCOUNT_ID = 1 AND STOCK_ID = 1;

-- TEST CASE 2 (BUYSTOCK): USER HAS INSUFFICIENT FUNDS
DECLARE @USER_ID INT = 2;
DECLARE @STOCK_ID INT = 2;
DECLARE @QUANTITY INT = 1;
DECLARE @PRICE DECIMAL(10, 2) = 2000.00;

-- USER HAS INSUFFICIENT FUNDS
INSERT INTO PORTFOLIO_HOLDINGS (ACCOUNT_ID, STOCK_ID, QUANTITY,
AVERAGE_PRICE)
VALUES (@USER_ID, 2, 0, 0.00);

EXEC BuyStock @USER_ID, @STOCK_ID, @QUANTITY, @PRICE;
GO

SELECT *
FROM PORTFOLIO_HOLDINGS
WHERE ACCOUNT_ID = 2 AND STOCK_ID = 2;

-- TEST CASE 3 (BUYSTOCK): USER ALREADY OWNS THE STOCK (UPDATING QUANTITY)
DECLARE @USER_ID INT = 3;
DECLARE @STOCK_ID INT = 3;
DECLARE @QUANTITY INT = 5;
DECLARE @PRICE DECIMAL(10, 2) = 150.00;

-- USER ALREADY OWNS STOCK
INSERT INTO PORTFOLIO_HOLDINGS (ACCOUNT_ID, STOCK_ID, QUANTITY,
AVERAGE_PRICE)
VALUES (@USER_ID, 3, 10, 150.00); -- USER ALREADY OWNS STOCK
```

```
EXEC BuyStock @USER_ID, @STOCK_ID, @QUANTITY, @PRICE;
GO

SELECT *
FROM PORTFOLIO_HOLDINGS
WHERE ACCOUNT_ID = 3 AND STOCK_ID = 3;

-- TEST CASE 4 (BUYSTOCK): USER DOES NOT EXIST IN THE PORTFOLIO
DECLARE @USER_ID INT = 999; -- THIS USER DOES NOT EXIST CAUSING THE CODE TO
FAIL SO IT WILL RAISE AN ERROR
DECLARE @STOCK_ID INT = 4; -- Assuming AMZN has StockID 4
DECLARE @QUANTITY INT = 5;
DECLARE @PRICE DECIMAL(10, 2) = 2500.00;

EXEC BuyStock @USER_ID, @STOCK_ID, @QUANTITY, @PRICE;
GO
```

SCREENSHOTS OF TEST CASES – STORED PROCEDURE 1





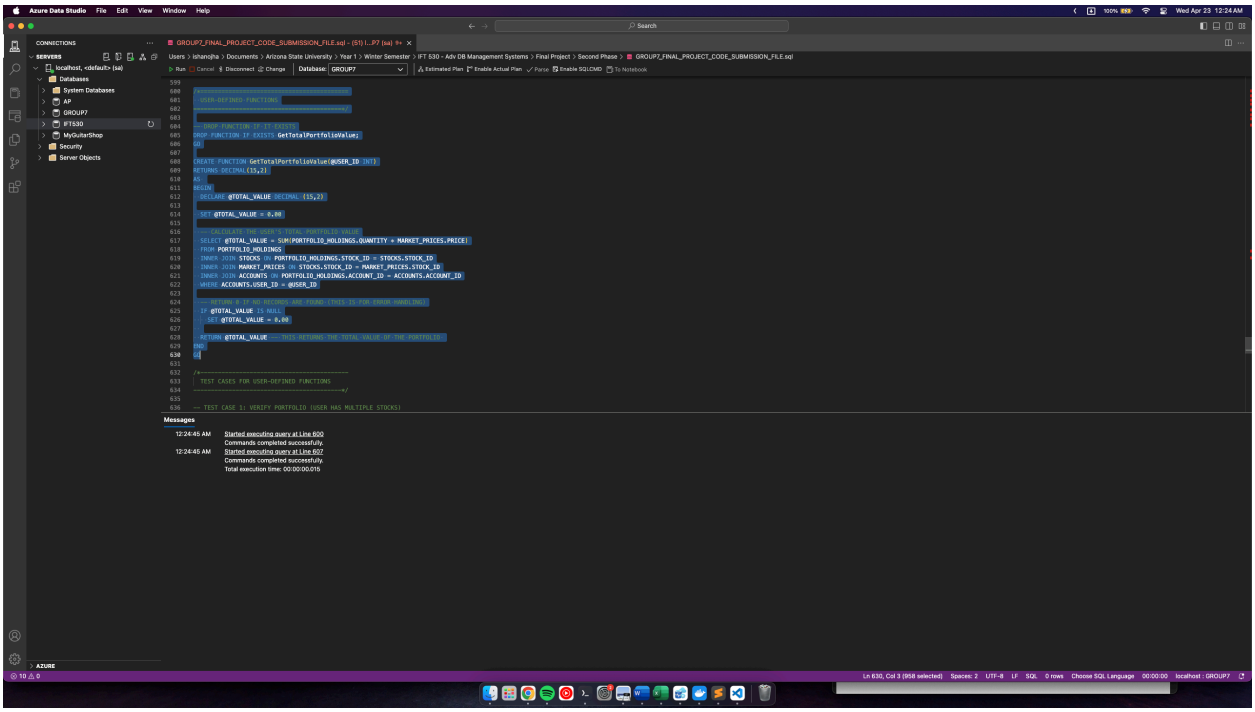
USER-DEFINED FUNCTIONS

CODE – USER-DEFINED FUNCTION

```
/*=============================================
  USER-DEFINED FUNCTIONS
=============================================*/

-- DROP FUNCTION IF IT EXISTS
DROP FUNCTION IF EXISTS GetTotalPortfolioValue;
GO

CREATE FUNCTION GetTotalPortfolioValue(@USER_ID INT)
RETURNS DECIMAL(15,2)
AS
BEGIN
  DECLARE @TOTAL_VALUE DECIMAL (15,2)

  SET @TOTAL_VALUE = 0.00

  -- CALCULATE THE USER'S TOTAL PORTFOLIO VALUE
  SELECT @TOTAL_VALUE = SUM(PORTFOLIO_HOLDINGS.QUANTITY *
MARKET_PRICES.PRICE)
  FROM PORTFOLIO_HOLDINGS
  INNER JOIN STOCKS ON PORTFOLIO_HOLDINGS.STOCK_ID = STOCKS.STOCK_ID
```

```
  INNER JOIN MARKET_PRICES ON STOCKS.STOCK_ID = MARKET_PRICES.STOCK_ID
  INNER JOIN ACCOUNTS ON PORTFOLIO_HOLDINGS.ACCOUNT_ID =
ACCOUNTS.ACCOUNT_ID
  WHERE ACCOUNTS.USER_ID = @USER_ID

  -- RETURN 0 IF NO RECORDS ARE FOUND (THIS IS FOR ERROR HANDLING)
  IF @TOTAL_VALUE IS NULL
    SET @TOTAL_VALUE = 0.00

  RETURN @TOTAL_VALUE -- THIS RETURNS THE TOTAL VALUE OF THE PORTFOLIO
END
GO
```

SUCCESSFUL EXECUTION OF UDF SCRIPT

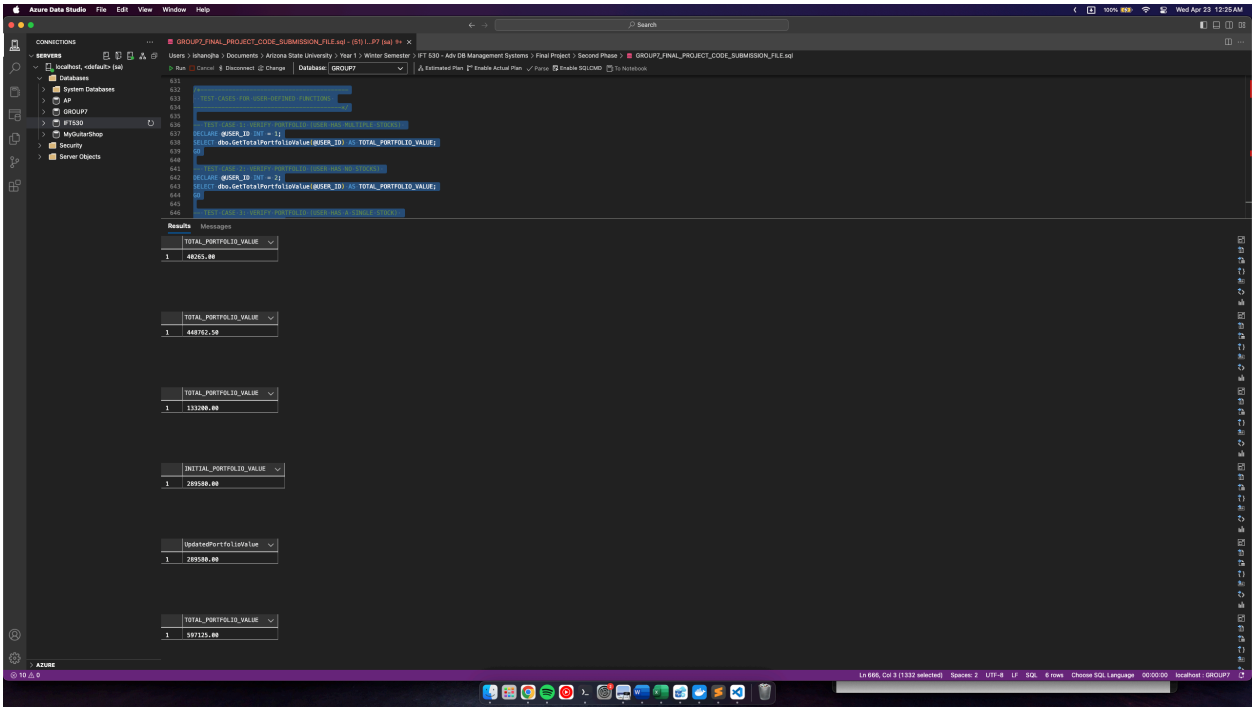


CODE – TEST CASES FOR UDF

```
/*----------------------------------------
  TEST CASES FOR USER-DEFINED FUNCTIONS
----------------------------------------*/

-- TEST CASE 1: VERIFY PORTFOLIO (USER HAS MULTIPLE STOCKS)
DECLARE @USER_ID INT = 1;
SELECT dbo.GetTotalPortfolioValue(@USER_ID) AS TOTAL_PORTFOLIO_VALUE;
GO

-- TEST CASE 2: VERIFY PORTFOLIO (USER HAS NO STOCKS)
```

```
DECLARE @USER_ID INT = 2;
SELECT dbo.GetTotalPortfolioValue(@USER_ID) AS TOTAL_PORTFOLIO_VALUE;
GO

-- TEST CASE 3: VERIFY PORTFOLIO (USER HAS A SINGLE STOCK)
DECLARE @USER_ID INT = 3;
SELECT dbo.GetTotalPortfolioValue(@USER_ID) AS TOTAL_PORTFOLIO_VALUE;
GO

-- TEST CASE 4: VERIFY PORTFOLIO (AFTER UPDATING STOCK PRICE)
DECLARE @USER_ID INT = 4;
-- INITIAL PORTFOLIO VALUE
SELECT dbo.GetTotalPortfolioValue(@USER_ID) AS INITIAL_PORTFOLIO_VALUE;
-- UPDATING THE STOCK PRICES SO WE CAN SEE THE CHANGES
UPDATE MARKET_PRICES SET PRICE = 120 WHERE STOCK_ID = 101; -- ASSUMPTION:
STOCK 101 PRICE CHANGE
UPDATE MARKET_PRICES SET PRICE = 160 WHERE STOCK_ID = 102; -- ASSUMPTION:
STOCK 102 PRICE CHANGE

-- CHECK UPDATED PORTFOLIO VALUE
SELECT dbo.GetTotalPortfolioValue(@USER_ID) AS UpdatedPortfolioValue;
GO

-- TEST CASE 5: VERIFY PORTFOLIO (VALUE FOR USER WITH NEGATIVE STOCK
QUANTITIES)
DECLARE @USER_ID INT = 5;
SELECT dbo.GetTotalPortfolioValue(@USER_ID) AS TOTAL_PORTFOLIO_VALUE;
GO
```

SCREENSHOTS OF TEST CASES – UDF



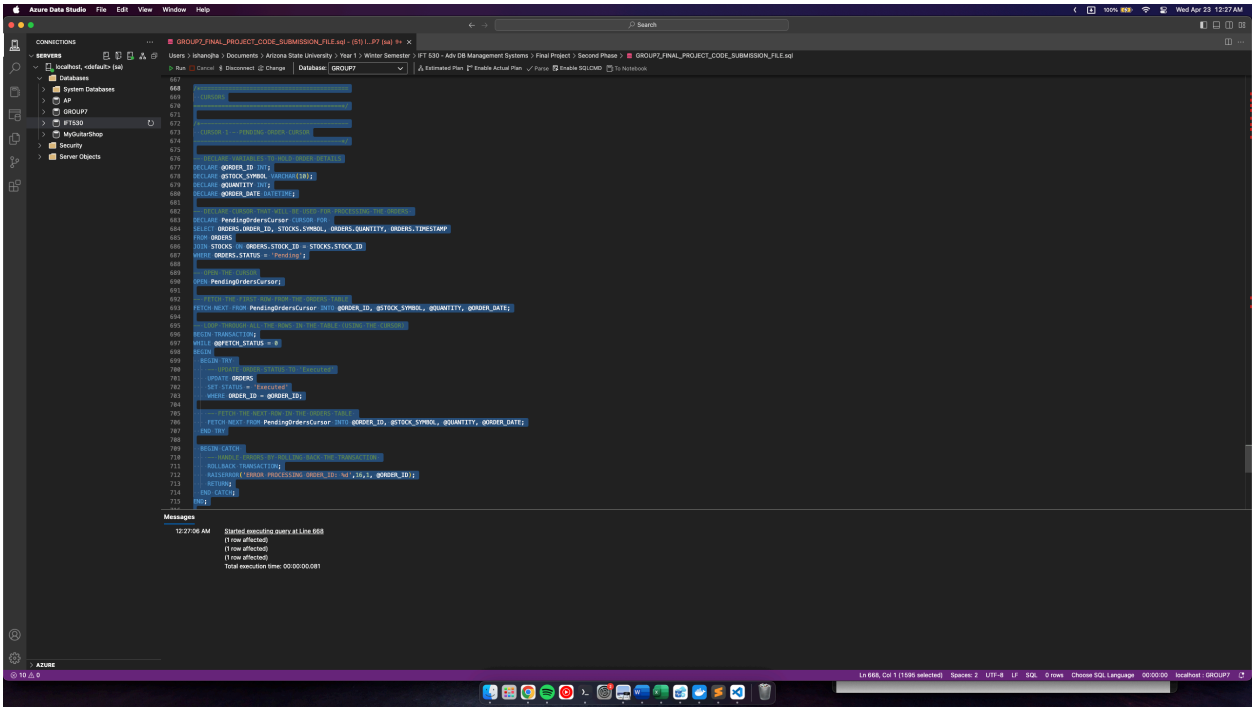
CURSORS

CODE – CURSOR 1

```
/*=============================================
  CURSORS
=============================================*/

/*-----------------------------------------
  CURSOR 1 - PENDING ORDER CURSOR
-----------------------------------------*/

-- DECLARE VARIABLES TO HOLD ORDER DETAILS
DECLARE @ORDER_ID INT;
DECLARE @STOCK_SYMBOL VARCHAR(10);
DECLARE @QUANTITY INT;
DECLARE @ORDER_DATE DATETIME;

-- DECLARE CURSOR THAT WILL BE USED FOR PROCESSING THE ORDERS
DECLARE PendingOrdersCursor CURSOR FOR
SELECT ORDERS.ORDER_ID, STOCKS.SYMBOL, ORDERS.QUANTITY, ORDERS.TIMESTAMP
FROM ORDERS
JOIN STOCKS ON ORDERS.STOCK_ID = STOCKS.STOCK_ID
```

```
WHERE ORDERS.STATUS = 'Pending';

-- OPEN THE CURSOR
OPEN PendingOrdersCursor;

-- FETCH THE FIRST ROW FROM THE ORDERS TABLE
FETCH NEXT FROM PendingOrdersCursor INTO @ORDER_ID, @STOCK_SYMBOL,
@QUANTITY, @ORDER_DATE;

-- LOOP THROUGH ALL THE ROWS IN THE TABLE (USING THE CURSOR)
BEGIN TRANSACTION;
WHILE @@FETCH_STATUS = 0
BEGIN
  BEGIN TRY
    -- UPDATE ORDER STATUS TO 'Executed'
    UPDATE ORDERS
    SET STATUS = 'Executed'
    WHERE ORDER_ID = @ORDER_ID;

    -- FETCH THE NEXT ROW IN THE ORDERS TABLE
    FETCH NEXT FROM PendingOrdersCursor INTO @ORDER_ID, @STOCK_SYMBOL,
@QUANTITY, @ORDER_DATE;
  END TRY

  BEGIN CATCH
    -- HANDLE ERRORS BY ROLLING BACK THE TRANSACTION
    ROLLBACK TRANSACTION;
    RAISERROR('ERROR PROCESSING ORDER_ID: %d',16,1, @ORDER_ID);
    RETURN;
  END CATCH;
END;

COMMIT TRANSACTION; -- COMMIT THE TRANSACTION AFTER ALL THE UPDATES ARE
DONE

-- CLOSE & DEALLOCATE THE CURSOR
CLOSE PendingOrdersCursor;
DEALLOCATE PendingOrdersCursor;
GO
```

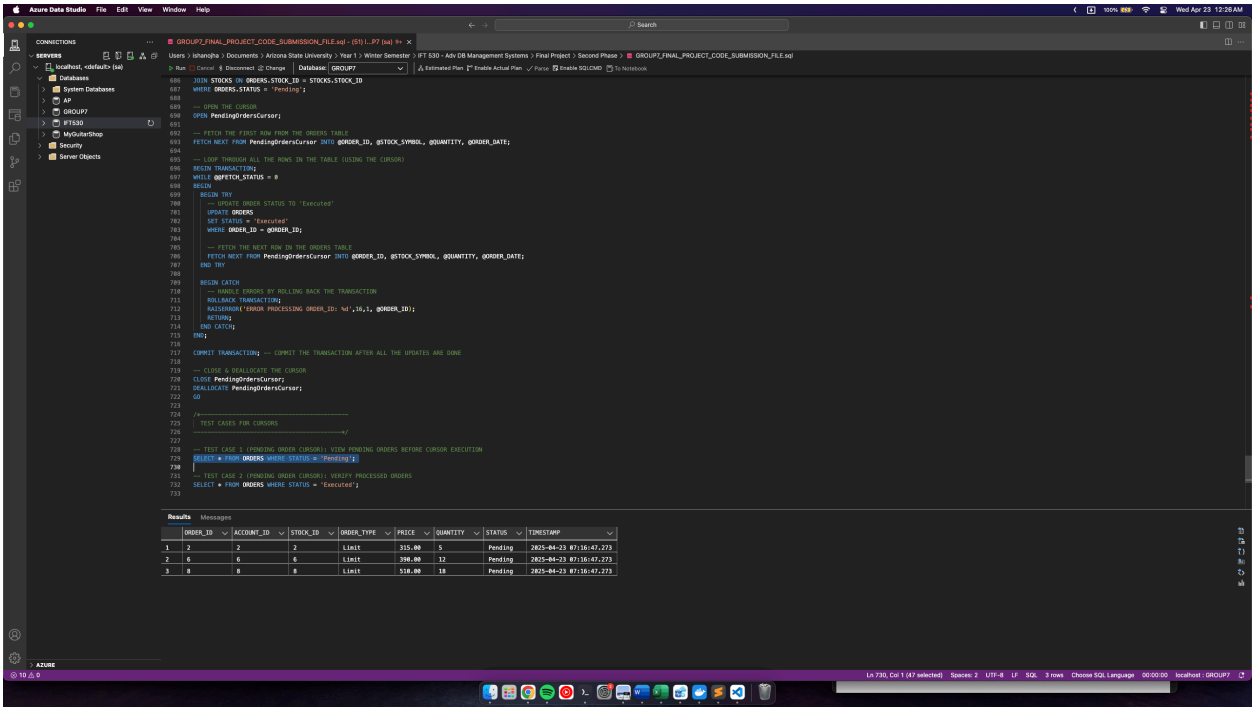
SUCCESSFUL EXECUTION OF CURSOR 1



CODE – TEST CASES FOR CURSOR 1

```
/*-----------------------------------------
  TEST CASES FOR CURSORS
-----------------------------------------*/

-- TEST CASE 1 (PENDING ORDER CURSOR): VIEW PENDING ORDERS BEFORE CURSOR EXECUTION
SELECT * FROM ORDERS WHERE STATUS = 'Pending';

-- TEST CASE 2 (PENDING ORDER CURSOR): VERIFY PROCESSED ORDERS
SELECT * FROM ORDERS WHERE STATUS = 'Executed';
```

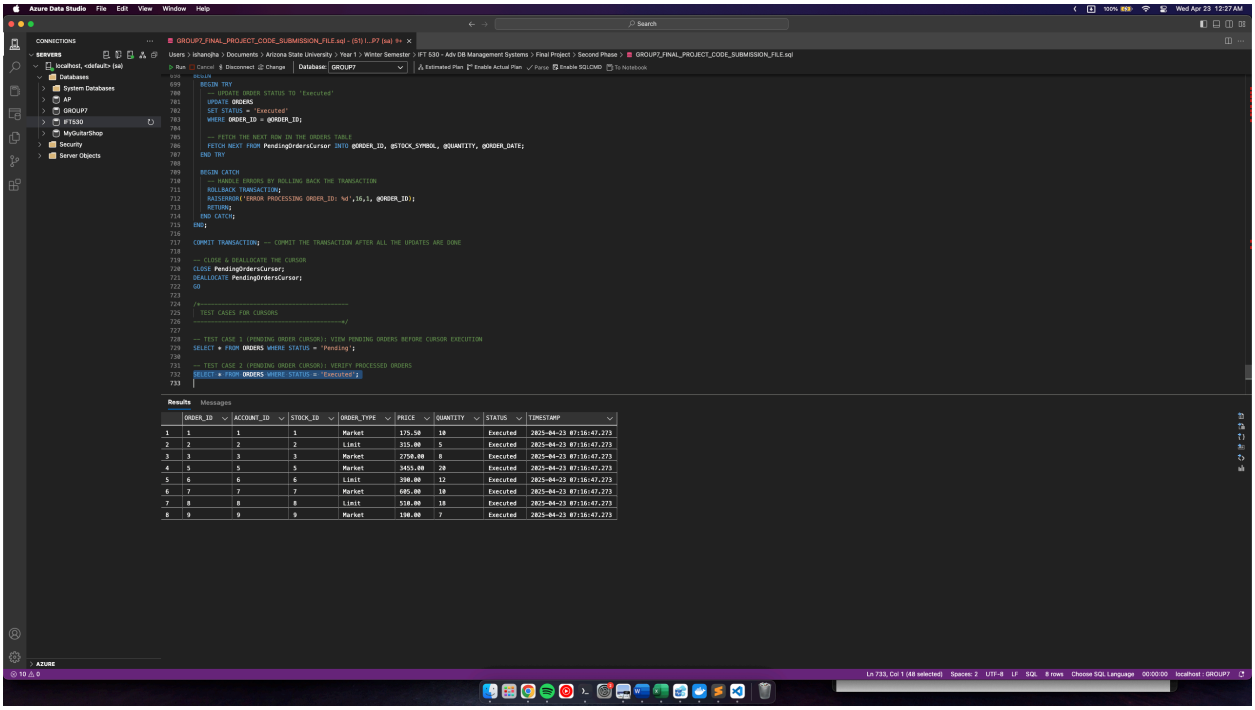
SCREENSHOTS OF TEST CASES – CURSOR 1



```
JOIN STOCKS ON ORDERS.STOCK_ID = STOCKS.STOCK_ID
WHERE ORDERS.STATUS = 'Pending';

-- OPEN THE CURSOR
OPEN PendingOrdersCursor;

-- FETCH THE FIRST ROW FROM THE ORDERS TABLE
FETCH NEXT FROM PendingOrdersCursor INTO @ORDER_ID, @STOCK_SYMBOL, @QUANTITY, @ORDER_DATE;

-- LOOP THROUGH ALL THE ROWS IN THE TABLE (USING THE CURSOR)
BEGIN TRANSACTION;
WHILE @@FETCH_STATUS = 0
BEGIN
  BEGIN TRY
    -- UPDATE ORDER STATUS TO 'Executed'
    UPDATE ORDERS
    SET STATUS = 'Executed'
    WHERE ORDER_ID = @ORDER_ID;

    -- FETCH THE NEXT ROW IN THE ORDERS TABLE
    FETCH NEXT FROM PendingOrdersCursor INTO @ORDER_ID, @STOCK_SYMBOL, @QUANTITY, @ORDER_DATE;
  END TRY

  BEGIN CATCH
    -- HANDLE ERRORS BY ROLLING BACK THE TRANSACTION
    ROLLBACK TRANSACTION;
    RAISERROR('ERROR PROCESSING ORDER_ID: %d',16,1, @ORDER_ID);
    RETURN;
  END CATCH;
END;

COMMIT TRANSACTION; -- COMMIT THE TRANSACTION AFTER ALL THE UPDATES ARE DONE

-- CLOSE & DEALLOCATE THE CURSOR
CLOSE PendingOrdersCursor;
DEALLOCATE PendingOrdersCursor;
GO

/*-----------------------------------------
  TEST CASES FOR CURSORS
-----------------------------------------*/

-- TEST CASE 1 (PENDING ORDER CURSOR): VIEW PENDING ORDERS BEFORE CURSOR EXECUTION
SELECT * FROM ORDERS WHERE STATUS = 'Pending';

-- TEST CASE 2 (PENDING ORDER CURSOR): VERIFY PROCESSED ORDERS
SELECT * FROM ORDERS WHERE STATUS = 'Executed';
```

Results

	ORDER_ID	ACCOUNT_ID	STOCK_ID	ORDER_TYPE	PRICE	QUANTITY	STATUS	TIMESTAMP
1	2	2	2	Limit	315.00	5	Pending	2025-04-23 07:16:47.273
2	6	6	6	Limit	398.00	12	Pending	2025-04-23 07:16:47.273
3	8	8	8	Limit	518.00	18	Pending	2025-04-23 07:16:47.273



```
BEGIN TRY
  -- UPDATE ORDER STATUS TO 'Executed'
  UPDATE ORDERS
  SET STATUS = 'Executed'
  WHERE ORDER_ID = @ORDER_ID;

  -- FETCH THE NEXT ROW IN THE ORDERS TABLE
  FETCH NEXT FROM PendingOrdersCursor INTO @ORDER_ID, @STOCK_SYMBOL, @QUANTITY, @ORDER_DATE;
END TRY

BEGIN CATCH
  -- HANDLE ERRORS BY ROLLING BACK THE TRANSACTION
  ROLLBACK TRANSACTION;
  RAISERROR('ERROR PROCESSING ORDER_ID: %d',16,1, @ORDER_ID);
  RETURN;
END CATCH;
END;

COMMIT TRANSACTION; -- COMMIT THE TRANSACTION AFTER ALL THE UPDATES ARE DONE

-- CLOSE & DEALLOCATE THE CURSOR
CLOSE PendingOrdersCursor;
DEALLOCATE PendingOrdersCursor;
GO

/*-----------------------------------------
  TEST CASES FOR CURSORS
-----------------------------------------*/

-- TEST CASE 1 (PENDING ORDER CURSOR): VIEW PENDING ORDERS BEFORE CURSOR EXECUTION
SELECT * FROM ORDERS WHERE STATUS = 'Pending';

-- TEST CASE 2 (PENDING ORDER CURSOR): VERIFY PROCESSED ORDERS
SELECT * FROM ORDERS WHERE STATUS = 'Executed';
```

Results

	ORDER_ID	ACCOUNT_ID	STOCK_ID	ORDER_TYPE	PRICE	QUANTITY	STATUS	TIMESTAMP
1	1	1	1	Market	175.50	10	Executed	2025-04-23 07:16:47.273
2	2	2	2	Limit	315.00	5	Executed	2025-04-23 07:16:47.273
3	3	3	3	Market	2750.00	8	Executed	2025-04-23 07:16:47.273
4	5	5	5	Market	3455.00	20	Executed	2025-04-23 07:16:47.273
5	6	6	6	Limit	398.00	12	Executed	2025-04-23 07:16:47.273
6	7	7	7	Market	605.00	10	Executed	2025-04-23 07:16:47.273
7	8	8	8	Limit	518.00	18	Executed	2025-04-23 07:16:47.273
8	9	9	9	Market	190.00	7	Executed	2025-04-23 07:16:47.273