



IFT 530 – Advanced Database Management Systems

Group 7

Final Project

Rahul Muddhapuram

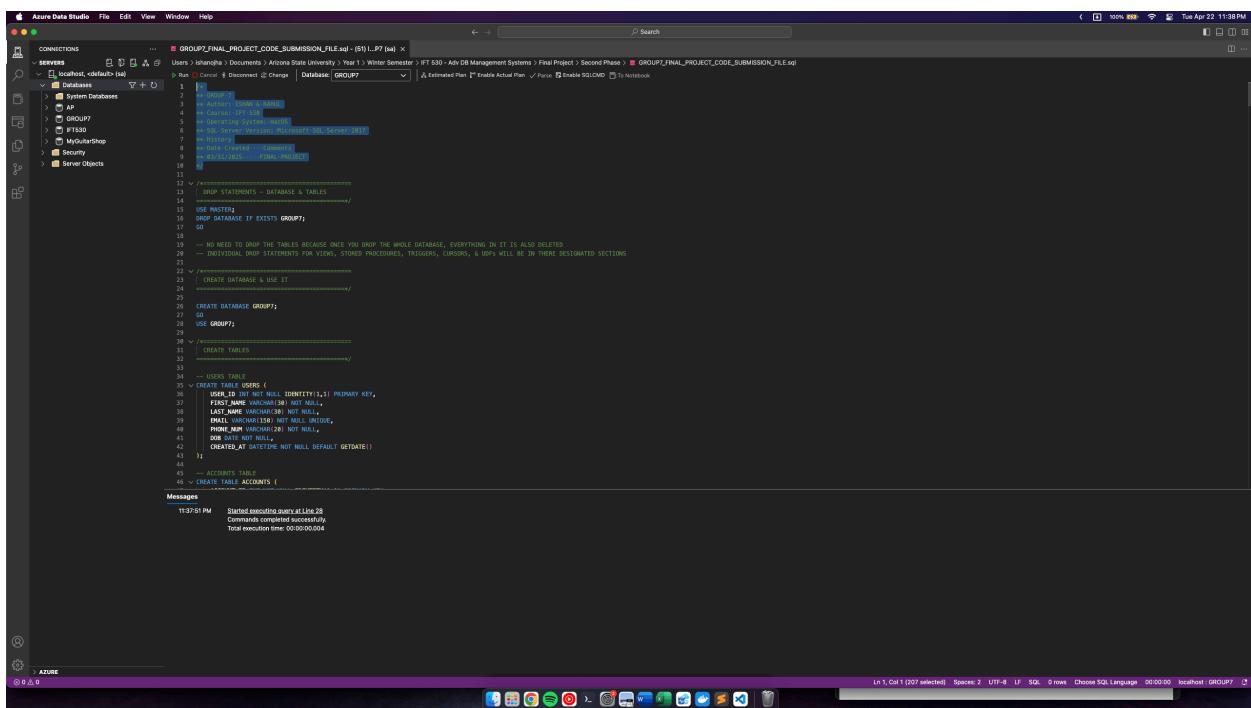
Ishan Ojha

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
*/
```



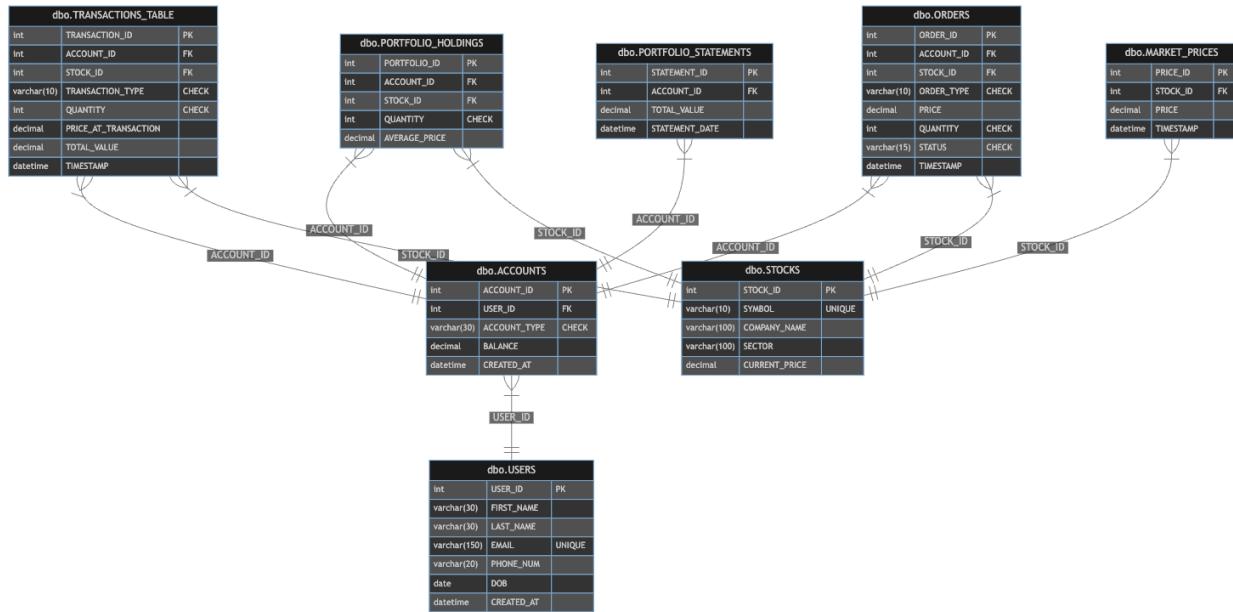
The screenshot shows the Azure Data Studio interface with a SQL editor window. The connection is set to 'localhost, default (sa)'. The script being run is 'GROUP7_FINAL_PROJECT_CODE_SUBMISSION_FILE.sql' (Line 1). The code includes comments explaining the purpose of each section: dropping statements for database and tables, creating the database, and defining the schema for the 'USERS' and 'ACCOUNTS' tables. The 'Messages' pane at the bottom indicates the command completed successfully in 0.000000 seconds.

```
/*
** 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
*/
-- NO NEED TO DROP THE TABLES BECAUSE ONCE YOU DROP THE WHOLE DATABASE, EVERYTHING IN IT IS ALSO DELETED
-- INDIVIDUAL DROP STATEMENTS FOR VIEWS, STORED PROCEDURES, TRIGGERS, CURSORS, & BPs WILL BE IN THERE DESIGNATED SECTIONS
-- CREATE DATABASE & USE IT
CREATE DATABASE GROUP7;
GO
USE GROUP7;
-- CREATE TABLES
CREATE TABLE USERS (
    ID INT IDENTITY(1,1) PRIMARY KEY,
    FIRST_NAME VARCHAR(50) NOT NULL,
    LAST_NAME VARCHAR(50) NOT NULL,
    EMAIL VARCHAR(100) UNIQUE,
    PHONE_NUM VARCHAR(20) NOT NULL,
    DOB DATE NOT NULL,
    CREATED_AT DATETIME NOT NULL DEFAULT GETDATE()
);
-- ACCOUNTS TABLE
CREATE TABLE ACCOUNTS (
```

Messages

11/3791IM Started executing query at Line 28
Commands completed successfully.
Total execution time: 00:00:00.004

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)
```

10

-- PORTFOLIO STATEMENTS TABLE

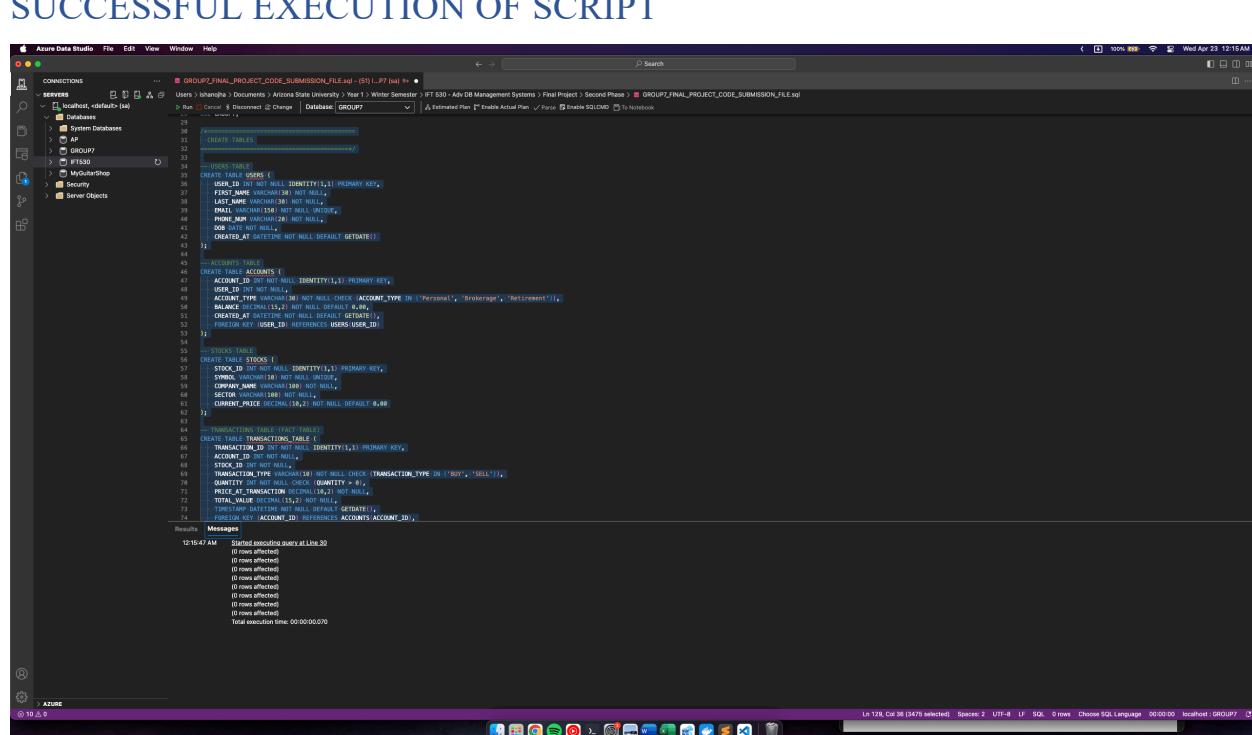
CREATE TABLE PORTFOLIO STATEMENTS

STATEMENT ID INT IDENTITY(1,1) PRIMARY KEY,

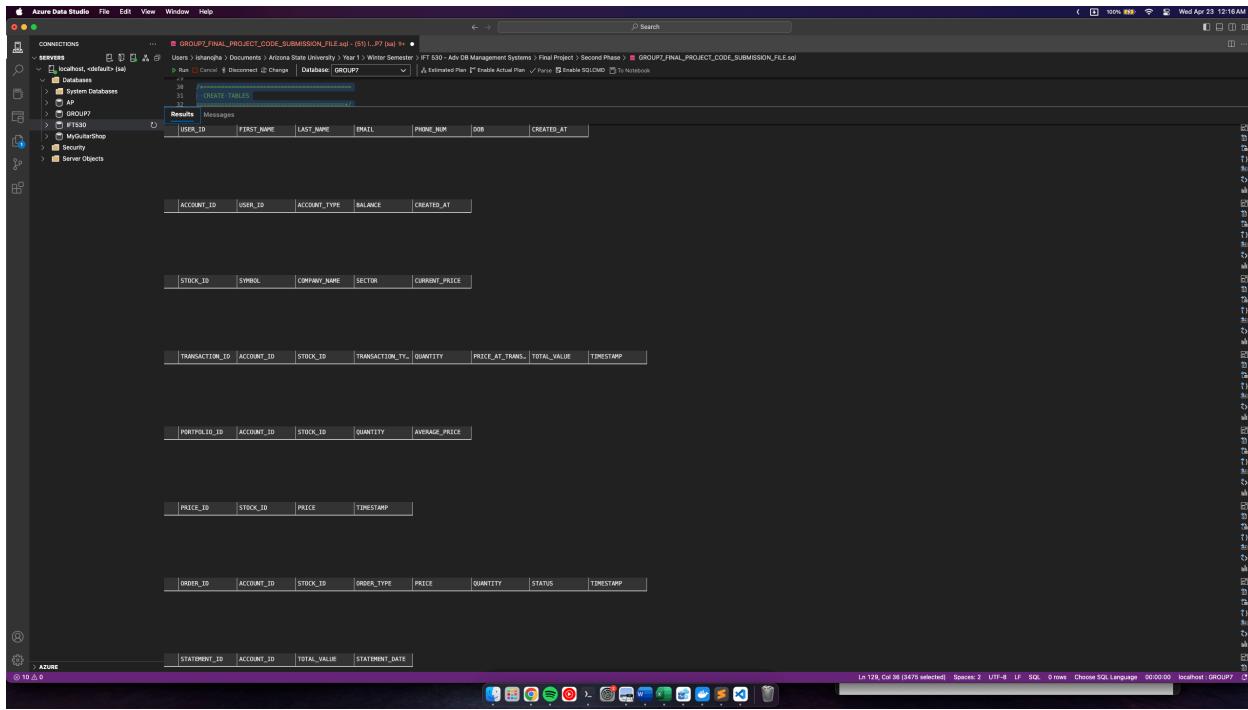
ACCOUNT_ID INT NOT NULL

TOTAL VALUE DECIMAL(18,2).

STATEMENT DATE DATETIME DEFAULT GETDATE();



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

```

    ...
    -- (CRITERIA: 10 IN DIMENSION TABLES & 20-50 ROWS IN THE REST OF THE TABLES) --
    ...
    -- INSERTING DATA INTO TABLES --
    ...
    -- INSERT INTO USERS (FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUM, DOB) VALUES --
    ...
    -- SELECT * FROM USERS -- (ENSURE ALL THE DATA IS POPULATED)
    ...
    -- INSERTING DATA INTO ACCOUNTS TABLE --
    ...
    -- (CRITERIA: 10 IN DIMENSION TABLES & 20-50 ROWS IN THE REST OF THE TABLES) --
    ...
    -- Results: Messages
    12:16:47 AM Started executing query at Line 131
    (20 rows affected)
    (10 rows affected)
    (20 rows affected)
    (20 rows affected)
    (20 rows affected)
    (10 rows affected)
    (10 rows affected)
    (20 rows affected)
    (20 rows affected)
    (10 rows affected)
    Total execution time: 00:00:00.000
  
```

SCREENSHOTS OF POPULATED TABLES

USER_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUM	DOB	CREATED_AT
1	Ishan	Dha	ishan.dha@example.com	223-455-7890	2009-01-31	2025-04-23 07:16:47.263
2	Komal	Patel	komal.patel@example.com	234-567-8901	2008-04-11	2025-04-23 07:16:47.263
3	Behrad	Hakim	behrad.hakim@example.com	345-456-9872	2008-03-09	2025-04-23 07:16:47.263
4	Rahul	Mudhupuram	rahul.mudhupuram@example.com	456-789-0123	2008-09-18	2025-04-23 07:16:47.263
5	Bhargav	Limboski	bhargav.limboski@example.com	567-898-1234	2008-02-05	2025-04-23 07:16:47.263
6	Lovpreet	Arcia	lovpreet.arcia@example.com	678-901-2345	2008-02-25	2025-04-23 07:16:47.263
7	Shuvra	Pragati	shuvra.pragati@example.com	789-012-3456	2008-06-17	2025-04-23 07:16:47.263
8	Yashasvi	Patil	yashasvi.patil@example.com	890-123-4567	2008-01-28	2025-04-23 07:16:47.263
9	Farisal	Walihi	farisal.walihi@example.com	912-345-6789	2008-05-21	2025-04-23 07:16:47.263
10	Amerit	Singh	amerit.singh@example.com	912-345-6789	2008-09-01	2025-04-23 07:16:47.263
11	Zhan	Xiaoyan	zhan.xiaoyan@example.com	912-345-6789	2008-02-27	2025-04-23 07:16:47.263
12	Qusseer	Obaidat	qusseer.obaidat@example.com	912-345-6789	2008-06-08	2025-04-23 07:16:47.263
13	Karen	Zhan	karen.zhan@example.com	912-345-6789	2008-02-27	2025-04-23 07:16:47.263
14	Troyas	Shannohai	troyas.shannohai@example.com	912-345-6789	2001-01-01	2025-04-23 07:16:47.263

ACCOUNT_ID	USER_ID	ACCOUNT_TYPE	BALANCE	CREATED_AT
1	1	Personal	5000.00	2025-04-23 07:16:47.263
2	2	Brokerage	10000.00	2025-04-23 07:16:47.263
3	3	Retirement	12000.00	2025-04-23 07:16:47.263
4	4	Personal	7000.00	2025-04-23 07:16:47.263
5	5	Brokerage	20000.00	2025-04-23 07:16:47.263
6	6	Personal	8800.00	2025-04-23 07:16:47.263
7	7	Brokerage	12000.00	2025-04-23 07:16:47.263
8	8	Retirement	10000.00	2025-04-23 07:16:47.263
9	9	Personal	6000.00	2025-04-23 07:16:47.263
10	10	Brokerage	14000.00	2025-04-23 07:16:47.263

STOCK_ID	SYMBOL	COMPANY_NAME	SECTOR	CURRENT_PRICE
1	AMPL	Apple Inc.	Technology	177.50
2	MSFT	Microsoft Corp.	Technology	318.20
3	GOOGL	Alphabet Inc.	Technology	2798.38
4	TSLA	Tesla Inc.	Automotive	778.40
5	AMZN	Amazon.com Inc.	E-commerce	3458.68
6	FB	Facebook Inc.	Technology	286.00
7	NVDA	NVIDIA Corp.	Semiconductors	496.80
8	NFLX	Netflix Inc.	Entertainment	500.55
9	DIS	Walt Disney Co.	Entertainment	188.00
10	BA	Boeing Co.	Aerospace	218.35

TRANSACTION_ID	ACCOUNT_ID	STOCK_ID	TRANSACTION_TYPE	QUANTITY	PRICE_AT_TRANSACTION	TOTAL_VALUE	TIMESTAMP
1	1	1	BUY	10	175.50	1755.00	2025-04-23 07:16:47.273
2	2	2	SELL	5	318.20	1591.00	2025-04-23 07:16:47.273
3	3	3	BUY	8	2200.40	17603.20	2025-04-23 07:16:47.273
4	4	4	BUY	5	729.40	3647.00	2025-04-23 07:16:47.273
5	5	5	SELL	20	3408.60	68172.00	2025-04-23 07:16:47.273
6	6	6	BUY	12	388.25	4659.00	2025-04-23 07:16:47.273
7	7	7	SELL	10	689.00	6890.00	2025-04-23 07:16:47.273
8	8	8	BUY	10	509.55	5095.00	2025-04-23 07:16:47.273

Two screenshots of Azure Data Studio showing the execution of an SQL script named GROUP7_FINAL_PROJECT_CODE_SUBMISSION_FILE.sql.

Screenshot 1 (Top):

The results pane displays the output of two queries:

- Query 1:** INSERT DATA INTO TABLES
- Query 2:** INSERT INTO PORTFOLIO_TABLE

Table 1: Portfolio Table (PORTFOLIO_TABLE)

PORTFOLIO_ID	ACCOUNT_ID	STOCK_ID	QUANTITY	AVERAGE_PRICE
1	1	1	10	175.00
2	2	2	5	318.20
3	3	3	8	2768.30
4	4	4	15	728.00
5	5	5	20	1662.00
6	6	6	12	386.25
7	7	7	10	680.00
8	8	8	10	580.55
9	9	9	7	188.00
10	10	10	15	218.35

Table 2: Price History (PRICE_HISTORY)

PRICE_ID	STOCK_ID	PRICE	TIMESTAMP
1	1	176.00	2025-04-23 07:16:47.273
2	2	312.50	2025-04-23 07:16:47.273
3	3	2755.00	2025-04-23 07:16:47.273
4	4	725.00	2025-04-23 07:16:47.273
5	5	3468.00	2025-04-23 07:16:47.273
6	6	385.00	2025-04-23 07:16:47.273
7	7	685.00	2025-04-23 07:16:47.273
8	8	585.00	2025-04-23 07:16:47.273
9	9	185.00	2025-04-23 07:16:47.273
10	10	215.00	2025-04-23 07:16:47.273

Table 3: Order History (ORDER_HISTORY)

ORDER_ID	ACCOUNT_ID	STOCK_ID	ORDER_TYPE	PRICE	QUANTITY	STATUS	TIMESTAMP
1	1	1	Market	175.00	10	Executed	2025-04-23 07:16:47.273
2	2	2	Limit	315.00	5	Pending	2025-04-23 07:16:47.273
3	3	3	Market	2758.00	8	Executed	2025-04-23 07:16:47.273
4	4	4	Limit	730.00	15	Canceled	2025-04-23 07:16:47.273
5	5	5	Market	3455.00	20	Executed	2025-04-23 07:16:47.273
6	6	6	Limit	310.00	12	Pending	2025-04-23 07:16:47.273
7	7	7	Market	685.00	10	Executed	2025-04-23 07:16:47.273
8	8	8	Limit	510.00	18	Queued	2025-04-23 07:16:47.273
9	9	9	Market	190.00	7	Executed	2025-04-23 07:16:47.273
10	10	10	Limit	278.00	6	Canceled	2025-04-23 07:16:47.273

Screenshot 2 (Bottom):

The results pane displays the output of two queries:

- Query 1:** INSERT DATA INTO TABLES
- Query 2:** INSERT INTO ORDER_TABLE

Table 1: Order Table (ORDER_TABLE)

ORDER_ID	ACCOUNT_ID	STOCK_ID	ORDER_TYPE	PRICE	QUANTITY	STATUS	TIMESTAMP
1	1	1	Market	175.00	10	Executed	2025-04-23 07:16:47.273
2	2	2	Limit	315.00	5	Pending	2025-04-23 07:16:47.273
3	3	3	Market	2758.00	8	Executed	2025-04-23 07:16:47.273
4	4	4	Limit	730.00	15	Canceled	2025-04-23 07:16:47.273
5	5	5	Market	3468.00	20	Executed	2025-04-23 07:16:47.273
6	6	6	Limit	310.00	12	Pending	2025-04-23 07:16:47.273
7	7	7	Market	685.00	10	Executed	2025-04-23 07:16:47.273
8	8	8	Limit	510.00	18	Pending	2025-04-23 07:16:47.273
9	9	9	Market	190.00	7	Executed	2025-04-23 07:16:47.273
10	10	10	Limit	278.00	6	Canceled	2025-04-23 07:16:47.273

Table 2: Statement History (STATEMENT_HISTORY)

STATEMENT_ID	ACCOUNT_ID	TOTAL_VALUE	STATEMENT_DATE
1	1	17500.00	2025-04-23 07:16:46.000
2	2	31500.00	2025-04-23 07:16:46.000
3	3	27600.00	2025-04-23 07:16:46.000
4	4	19500.00	2025-04-23 07:16:46.000
5	5	23600.00	2025-04-23 07:16:46.000
6	6	25500.00	2025-04-23 07:16:46.000
7	7	27600.00	2025-04-23 07:16:46.000
8	8	21000.00	2025-04-23 07:16:46.000
9	9	21000.00	2025-04-23 07:16:46.000
10	10	23600.00	2025-04-23 07:16:46.000

VIEWS

CODE – VIEW 1

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

VIEW 1 - USER PORTFOLIO SUMMARY

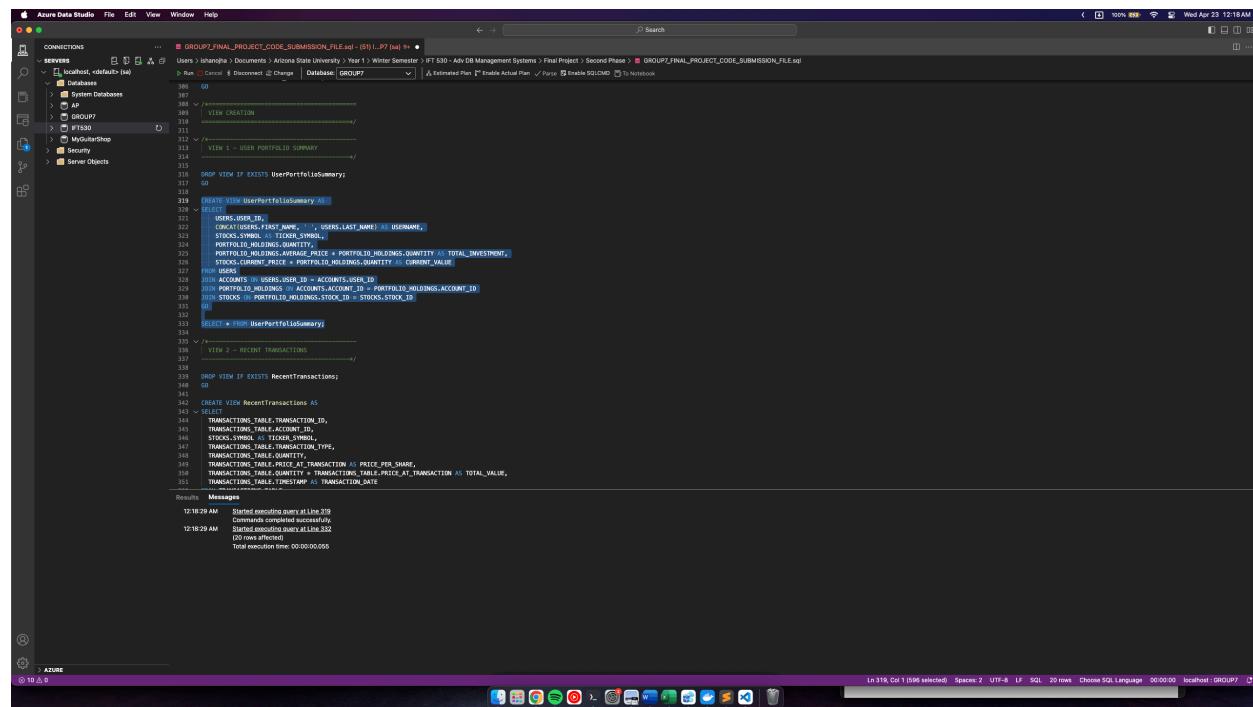
*/

```
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.ACOUNT_ID =
PORTFOLIO_HOLDINGS.ACOUNT_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



The screenshot shows the Azure Data Studio interface with a dark theme. The left sidebar displays a tree view of database connections, servers, and objects. The main pane shows a SQL script being run. The script creates a view named 'UserPortfolioSummary' with the specified columns and joins. The execution status at the bottom indicates the command has completed successfully.

```
File Edit View Window Help
CONNECTIONS
  SERVERS
    localhost<default> (sa)
      GROUP1_FINAL_PROJECT_CODE_SUBMISSION_FILE.sql (51 L, 47 (sa) +)
        Run ▾ Cancel ▾ Disconnect ▾ Change Database: GROUP1
        Estimated Plan ▾ Enable Actual Plan ▾ Parse ▾ Enable SQLCMD ▾ To Notebook
      385 GO
      386
      387
      388 ✓ /*VIEW CREATION
      389
      390   VIEW 1 - USER PORTFOLIO SUMMARY
      391   />
      392   /*
      393     DROP VIEW IF EXISTS UserPortfolioSummary;
      394     GO
      395
      396     CREATE VIEW UserPortfolioSummary AS
      397       SELECT
      398         USERS.USER_ID,
      399         CONCAT(USERS.FIRST_NAME, ' ', USERS.LAST_NAME) AS USERNAME,
      400         STOCKS.SYMBOL AS TICKER_SYMBOL,
      401         PORTFOLIO_HOLDINGS.QUANTITY,
      402         PORTFOLIO_HOLDINGS.AVERAGE_PRICE * PORTFOLIO_HOLDINGS.QUANTITY AS TOTAL_INVESTMENT,
      403         STOCKS.CURRENT_PRICE * PORTFOLIO_HOLDINGS.QUANTITY AS CURRENT_VALUE
      404       FROM USERS
      405       JOIN ACCOUNTS ON USERS.USER_ID = ACCOUNTS.ACOUNT_ID
      406       JOIN PORTFOLIO_HOLDINGS ON ACCOUNTS.ACOUNT_ID = PORTFOLIO_HOLDINGS.ACOUNT_ID
      407       JOIN STOCKS ON PORTFOLIO_HOLDINGS.STOCK_ID = STOCKS.STOCK_ID
      408       WHERE PORTFOLIO_HOLDINGS.QUANTITY > 0
      409
      410     SELECT * FROM UserPortfolioSummary;
      411
      412     /*
      413       VIEW 2 - RECENT TRANSACTIONS
      414       />
      415       /*
      416         DROP VIEW IF EXISTS RecentTransactions;
      417         GO
      418
      419         CREATE VIEW RecentTransactions AS
      420           SELECT
      421             TRANSACTIONS_TABLE.TRANSACTION_ID,
      422             TRANSACTIONS_TABLE.ACOUNT_ID,
      423             STOCKS_SYMBOL.TICKER_SYMBOL,
      424             TRANSACTIONS_TABLE.QUANTITY,
      425             TRANSACTIONS_TABLE.PRICE_AT_TRANSACTION / PRICE_PER_SHARE,
      426             TRANSACTIONS_TABLE.QUANTITY * TRANSACTION_TABLE.PRICE_AT_TRANSACTION AS TOTAL_VALUE,
      427             TRANSACTIONS_TABLE.TIMESTAMP AS TRANSACTION_DATE
      428           FROM TRANSACTIONS_TABLE
      429           JOIN STOCKS_SYMBOL ON TRANSACTIONS_TABLE.STOCK_ID = STOCKS_SYMBOL.STOCK_ID
      430           ORDER BY TRANSACTION_DATE DESC
      431           LIMIT 10
      432
      433     /*
      434     Messages
      12:18:29 AM Started executing query at Line 219
      12:18:29 AM Finished executing query at Line 219
      12:18:29 AM Started executing query at Line 382
      (20 rows affected)
      Total execution time: 00:00:00.055
      Ln 319 Col 1 (596 selected) Spaces: 2 UTF-8 LF SQL 20 rows Choose SQL Language 00:00:00 localhost:GROUP1
```

SCREENSHOTS OF TEST CASES – VIEW 1

```

CREATE VIEW [GROUP2_FINAL_PROJECT_CODE_SUBMISSION_FILE].[dbo].[UserPortfolioSummary] AS
    SELECT
        U.USER_ID,
        U.USERNAME,
        S.TICKER_SYMBOL,
        H.QUANTITY,
        H.TOTAL_INVESTMENT,
        H.CURRENT_VALUE
    FROM
        USERS U
    JOIN
        ACCOUNTS A ON U.USER_ID = A.USER_ID
    JOIN
        HOLDINGS H ON A.ACCT_ID = H.PORTFOLIO_ID
    JOIN
        STOCKS S ON H.HOLDING_ID = S.STOCK_ID
    WHERE
        H.QUANTITY > 0
    GROUP BY
        U.USER_ID,
        U.USERNAME,
        S.TICKER_SYMBOL,
        H.QUANTITY,
        H.TOTAL_INVESTMENT,
        H.CURRENT_VALUE

```

The screenshot shows the SQL code for creating a view named 'UserPortfolioSummary'. The code uses joins to pull data from four tables: USERS, ACCOUNTS, HOLDINGS, and STOCKS. It filters for rows where 'H.QUANTITY > 0' and groups the results by user ID, username, stock symbol, quantity, total investment, and current value.

User ID	Username	Ticker Symbol	Quantity	Total Investment	Current Value
1	Ishan Sijha	AAPL	120	18880.00	21868.00
2	Ishan Sijha	MSFT	50	8625.00	9516.00
2	Komal Patel	AAPL	100	18880.00	20516.00
4	Komal Patel	GOOG	150	32387.50	41255.00
5	Bahadur Hakim	MSFT	120	33280.00	37724.00
6	Bahadur Hakim	TSLA	75	4587.50	54938.00
7	Rahul Mudhugopan	AAPL	80	18880.00	19408.00
8	Rahul Mudhugopan	GOOG	100	30160.00	37718.00
9	Bharat Limbachia	FB	150	34080.00	37558.00
10	Bharat Limbachia	MSFT	250	28580.00	37558.00
11	Luvpreet Arora	FB	100	63135.00	68445.00
12	Luvpreet Arora	NVDA	98	53127.00	54972.00
13	Utpal Sudhar	NFLX	200	95180.00	108118.00
14	Utpal Sudhar	AMZN	100	37150.00	41254.00
15	Dhruv Prasapati	AMZN	140	38712.00	39469.00
16	Dhruv Prasapati	AMZN	110	374680.00	379565.00
17	Faisal Wazili	GOOG	175	472587.50	481382.50
18	Faisal Wazili	TSLA	95	67521.25	68438.00
19	Purnima Meodkar	NVDA	85	51080.00	51868.00
20	Purnima Meodkar	NFLX	150	53175.00	55373.00
21	Ishan Sijha	AAPL	20	3688.00	37518.00
22	Komal Patel	MSFT	1	8.00	338.20
23	Bahadur Hakim	GOOG	15	2256.00	41254.50

CODE – VIEW 2

```

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

```

```

DROP VIEW IF EXISTS RecentTransactions;
GO

```

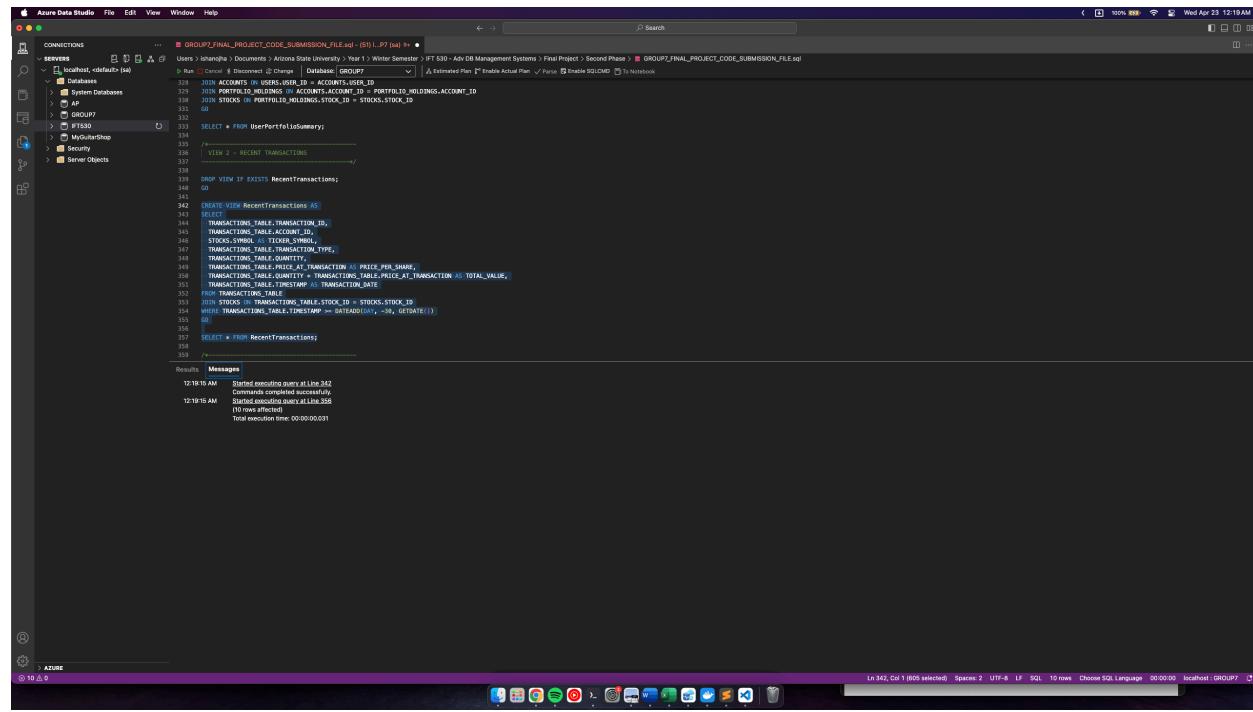
```

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;
```

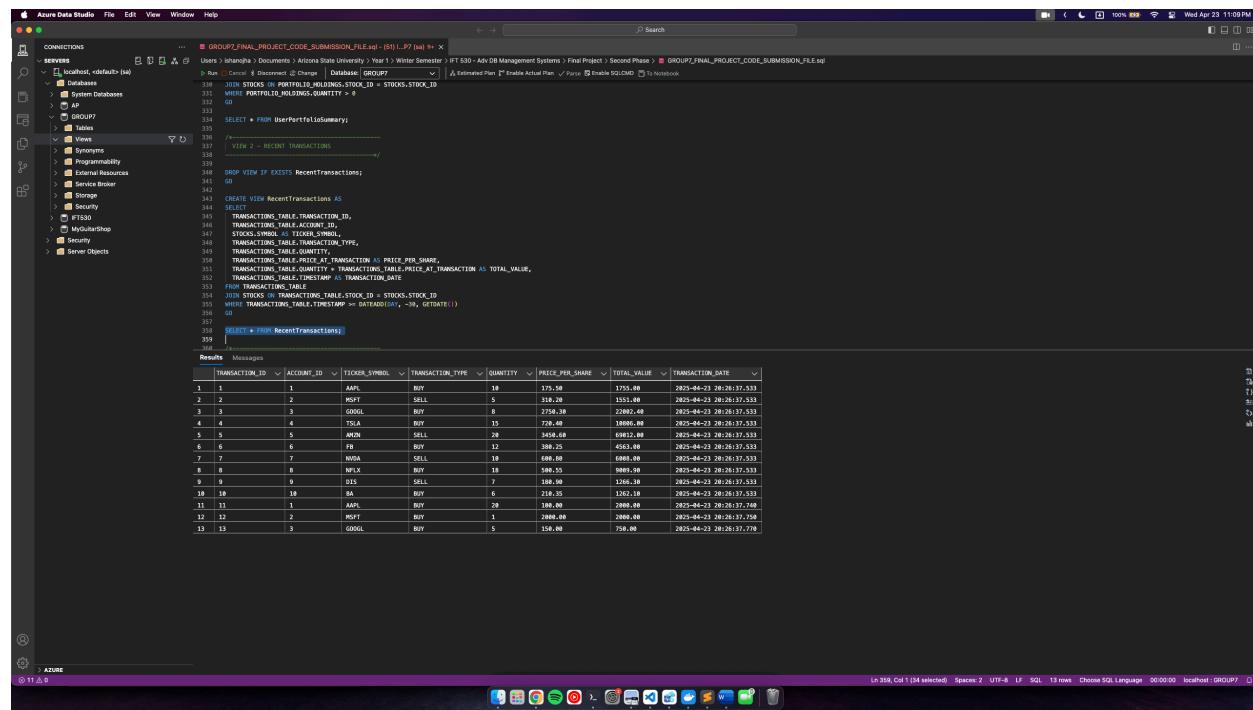
SUCCESSFUL EXECUTION OF VIEW 2



Azure Data Studio interface showing the execution of a SQL script named GROUP7_FINAL_PROJECT_CODE_SUBMISSION_FILE.sql. The results pane displays the creation of a view named RecentTransactions, which selects data from the UserPortfolioSummary table. The execution completed successfully with 10 rows affected.

```
12:19:10 AM Started executing query at Line 342
12:19:15 AM Commands completed successfully.
12:19:15 AM (10 rows affected)
Total execution time: 00:00:00.03
```

SCREENSHOTS OF TEST CASES – VIEW 2



Azure Data Studio interface showing the results of the executed SQL script. The results pane displays 13 rows of transaction data, including columns such as TRANSACTION_ID, ACCOUNT_ID, TICKER_SYMBOL, TRANSACTION_TYPE, QUANTITY, PRICE_PER_SHARE, TOTAL_VALUE, and TRANSACTION_DATE.

TRANSACTION_ID	ACCOUNT_ID	TICKER_SYMBOL	TRANSACTION_TYPE	QUANTITY	PRICE_PER_SHARE	TOTAL_VALUE	TRANSACTION_DATE
1	1	AMPL	BUY	10	171.50	1715.00	2025-04-23 20:26:37.533
2	1	MSFT	SELL	5	318.50	1592.50	2025-04-23 20:26:37.533
3	3	CSCO	BUY	1	7226.20	7226.20	2025-04-23 20:26:37.533
4	4	TSLA	BUY	15	729.40	10931.00	2025-04-23 20:26:37.533
5	5	AMZN	SELL	20	3458.60	69172.00	2025-04-23 20:26:37.533
6	6	FB	BUY	12	386.25	4633.00	2025-04-23 20:26:37.533
7	7	NVDA	SELL	10	6865.00	68650.00	2025-04-23 20:26:37.533
8	8	NFLX	BUY	10	580.50	5805.00	2025-04-23 20:26:37.533
9	9	USS	SELL	7	238.50	1669.50	2025-04-23 20:26:37.533
10	10	SA	BUY	6	219.35	1316.10	2025-04-23 20:26:37.533
11	11	AMPL	BUY	20	180.00	3600.00	2025-04-23 20:26:37.748
12	12	MSFT	BUY	2	2800.00	5600.00	2025-04-23 20:26:37.750
13	13	GOOGL	BUY	5	150.00	750.00	2025-04-23 20:26:37.778

CODE – VIEW 3

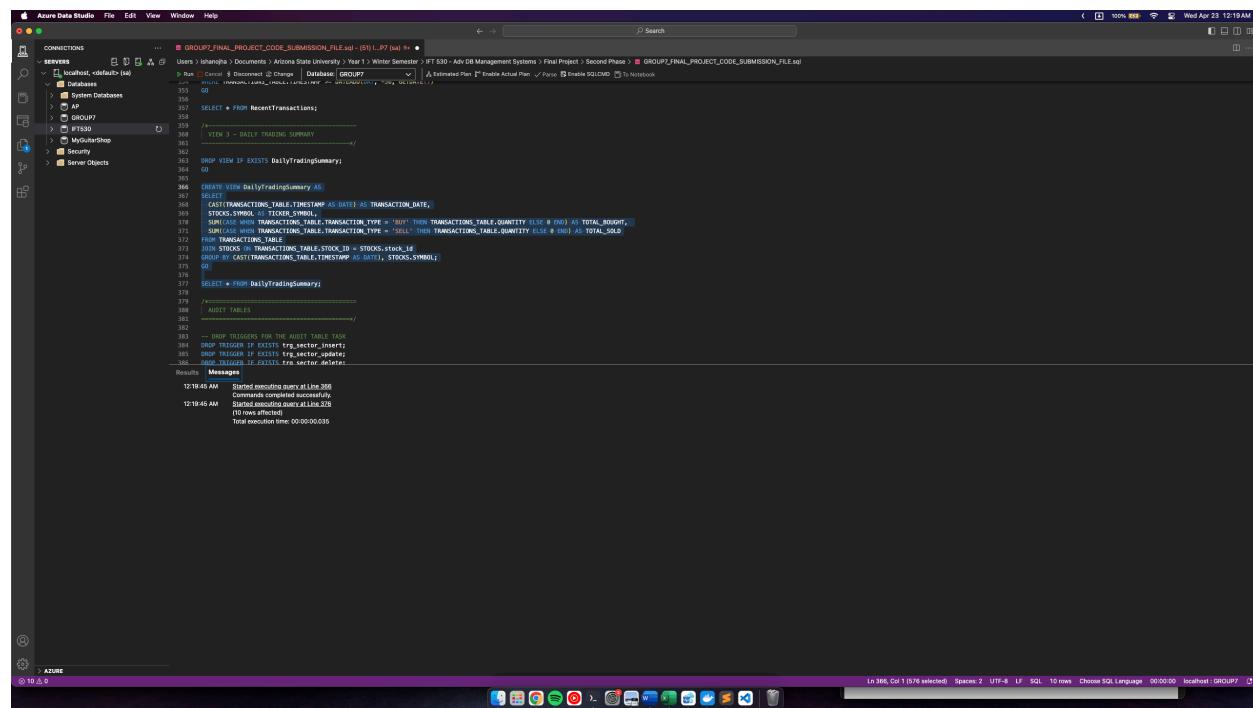
```
/*
-----  
VIEW 3 - DAILY TRADING SUMMARY  
-----*/
```

```
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



The screenshot shows the Azure Data Studio interface with a query editor window. The code for creating and selecting from the DailyTradingSummary view is pasted into the editor. The status bar at the bottom indicates the execution completed successfully with 10 rows affected.

```
/*
-----  
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  
GROUP BY CAST(TRANSACTIONS_TABLE.TIMESTAMP AS DATE), STOCKS.SYMBOL  
GO  
  
SELECT * FROM DailyTradingSummary;
```

Results

```
12:19:45 AM Started executing query at line 366  
12:19:45 AM All columns displayed  
12:19:45 AM Started dropping table L1_Lines  
12:19:45 AM (10 rows affected)  
Total execution time: 00:00:00:035
```

Messages

```
12:19:45 AM Started executing query at line 366  
12:19:45 AM All columns displayed  
12:19:45 AM Started dropping table L1_Lines  
12:19:45 AM (10 rows affected)  
Total execution time: 00:00:00:035
```

SCREENSHOTS OF TEST CASES – VIEW 3

The screenshot shows the Azure Data Studio interface with a SQL editor window. The code being run is:

```

-- GROUP FINAL PROJECT CODE SUBMISSION FILE id : (5) L_P7 [sa] + X
USE [Winter Semester]
GO
SELECT * FROM #DailyTradingSummary
GO

```

The code is for creating a view named #DailyTradingSummary. It uses a temporary table #DailyTradingSummary to store data from the TRANSACTIONS_TABLE. The view then selects the data from this temporary table.

Results

TRANSACTION_DATE	TICKER_SYMBOL	TOTAL_BOUGHT	TOTAL_SOLD
2023-08-23	AMZN	10	0
2023-08-23	GOOGL	0	20
2023-08-23	MSFT	0	5
2023-08-23	TSLA	0	7
2023-08-23	FB	12	0
2023-08-23	GOOGL	13	0
2023-08-23	MSFT	1	5
2023-08-23	AMZN	10	0
2023-08-23	MSFT	0	10
2023-08-23	TSLA	15	0

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 INITIAL 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

```

--> Audit Tables
381
382  DROP TRIGGERS FOR THE STOCK TABLE
383  DROP TRIGGER IF EXISTS trg_sector_Insert;
384  DROP TRIGGER IF EXISTS trg_sector_update;
385  DROP TRIGGER IF EXISTS trg_sector_Delete;
386
387  DROP TABLE IF EXISTS SECTOR_AUDIT;
388
389  GO
390
391  --CREATE THIS TRIGGER TO LOG THE ORIGINAL FIELDS FROM THE STOCK TABLE
392  CREATE TABLE SECTOR_AUDIT (
393      AUDIT_ID INT IDENTITY(1,1) PRIMARY KEY,
394      STOCK_ID INT,
395      SECTOR NVARCHAR(50),
396      ACTION_TYPE NVARCHAR(10),
397      CHANGED_AT DATETIME
398 );
399
400
401  --CREATE TRIGGER THAT LOGS THE ORIGINAL RECORD & LOGS WHEN A STOCK IS INSERTED
402  CREATE TRIGGER trg_sector_Insert
403  ON STOCKS
404  AFTER INSERT
405  AS
406  BEGIN
407      INSERT INTO SECTOR_AUDIT (STOCK_ID, SECTOR_NAME, ACTION_TYPE, CHANGED_AT)
408      SELECT
409          STOCK_ID,
410          SECTOR,
411          'INSERT',
412          GETDATE()
413      FROM INSERTED
414  END
415
416
417  --CREATE TRIGGER LOGS WHEN THE SECTOR IS UPDATED
418  CREATE TRIGGER trg_sector_update
419  ON STOCKS
420  AFTER UPDATE
421  AS
422  BEGIN
423      INSERT INTO SECTOR_AUDIT (STOCK_ID, SECTOR_NAME, ACTION_TYPE, CHANGED_AT)
424      SELECT
425          INSERTED STOCK_ID,
426          INSERTED SECTOR,
427          'UPDATE',
428          GETDATE()
429      FROM INSERTED
430      WHERE DELETED = INSERTED STOCK_ID = DELETED STOCK_ID
431      AND INSERTED SECTOR <> DELETED SECTOR
432      THIS STATEMENT保障 THE CHANGE HAS BEEN MADE
433
434
435

```

Messages

```

12:21:11 AM Started executing query at Line 391
12:21:11 AM Commands completed successfully.
12:21:11 AM Started executing query at Line 434
12:21:11 AM Commands completed successfully.
12:21:11 AM Started executing query at Line 435
12:21:11 AM Commands completed successfully.
12:21:11 AM Started executing query at Line 434
12:21:11 AM Commands completed successfully.
Total execution time: 00:00:00.192

```

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

The screenshot shows the Azure Data Studio interface with a SQL editor window. The code in the editor is as follows:

```

-- 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
```

The results pane displays the data from the sector_audit table:

AUDIT_ID	STOCK_ID	SECTOR_NAME	ACTION_TYPE	CHANGED_AT
1	1002	Private Equity	INSERT	2025-04-23 07:21:56.418
2	2	Asset Management	UPDATE	2025-04-23 07:21:56.423
3	3	Asset Management	DELETE	2025-04-23 07:21:56.430

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 TRANSACTION 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

```

-- GROUP7_FINAL_PROJECT_CODE_SUBMISSION_FILE.sql - (St) - P7 (sa) ->
Run | Cancel | Document | Database: GROUP7 | Estimated Plan | Enable Actual Plan | Parse | Enable SQLCMD | To Notebook
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
170

```

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
```

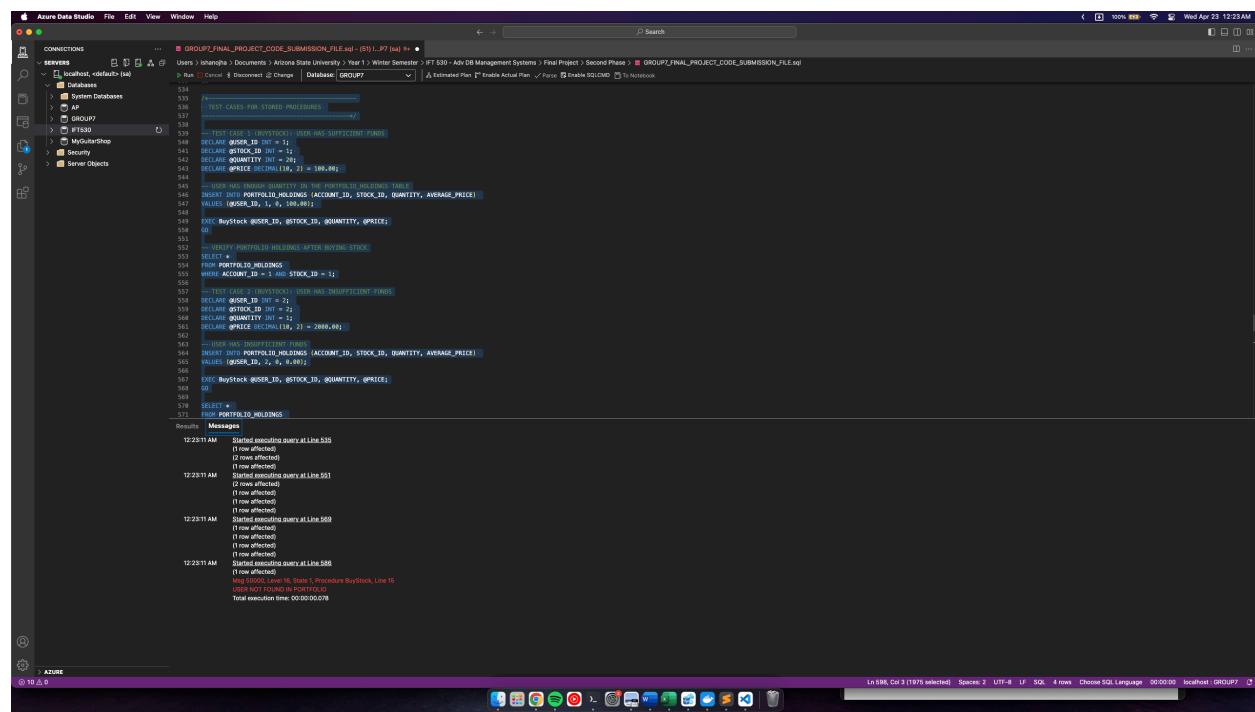
```
EXEC BuyStock @USER_ID, @STOCK_ID, @QUANTITY, @PRICE;
GO
```

```
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



```
GOPT1P7
-- TEST CASE 1: BUYSTOCK: USER HAS SUFFICIENT FUNDS
534
535    DECLARE @USER_ID INT = 1;
536    DECLARE @STOCK_ID INT = 2;
537    DECLARE @QUANTITY INT = 20;
538    DECLARE @PRICE DECIMAL(10, 2) = 100.00;
539
540    -- USER HAS ENOUGH FUNDS IN THE PORTFOLIO_HOLDINGS TABLE
541    INSERT INTO PORTFOLIO_HOLDINGS (ACCOUNT_ID, STOCK_ID, QUANTITY, AVERAGE_PRICE)
542    VALUES (@USER_ID, 2, 100, 100);
543
544    EXEC BuyStock @USER_ID, @STOCK_ID, @QUANTITY, @PRICE;
545
546    -- VERIFY PORTFOLIO HOLDINGS AFTER BUYING STOCK
547    SELECT *
548    FROM PORTFOLIO_HOLDINGS
549    WHERE ACCOUNT_ID = 1 AND STOCK_ID = 2;
550
551    -- TEST CASE 2: BUYSTOCK: USER HAS INSUFFICIENT FUNDS
552
553    DECLARE @USER_ID INT = 2;
554    DECLARE @STOCK_ID INT = 2;
555    DECLARE @QUANTITY INT = 20;
556    DECLARE @PRICE DECIMAL(10, 2) = 2800.00;
557
558    -- USER HAS INSUFFICIENT FUNDS
559    INSERT INTO PORTFOLIO_HOLDINGS (ACCOUNT_ID, STOCK_ID, QUANTITY, AVERAGE_PRICE)
560    VALUES (@USER_ID, 2, 0, 0.00);
561
562    EXEC BuyStock @USER_ID, @STOCK_ID, @QUANTITY, @PRICE;
563
564    -- VERIFY PORTFOLIO HOLDINGS
565    SELECT *
566    FROM PORTFOLIO_HOLDINGS;
```

Results

```
Messages
12/23/11 AM Started executing query at Line 535
(1 row affected)
12/23/11 AM Started executing query at Line 541
(1 row affected)
12/23/11 AM Started executing query at Line 547
(1 row affected)
12/23/11 AM Started executing query at Line 553
(1 row affected)
12/23/11 AM Started executing query at Line 559
(1 row affected)
12/23/11 AM Started executing query at Line 565
(1 row affected)
12/23/11 AM User not found in PORTFOLIO
User not found in level 16, state 1, Procedure BuyStock, Line 15
User not found in PORTFOLIO
Total execution time: 00:00:00.078
```

```

-- TEST CASE 1: BUY STOCK IF USER HAS SUFFICIENT FUNDS
538
539 --TEST CASE 1: BUY STOCK IF USER HAS SUFFICIENT FUNDS
540
541 DECLARE @USER_ID INT = 1;
542 DECLARE @STOCK_ID INT = 1;
543 DECLARE @QUANTITY INT = 2;
544 DECLARE @PRICE DECIMAL(4, 2) = 100.00;
545
546 INSERT INTO PORTFOLIO_HOLDINGS (ACCOUNT_ID, STOCK_ID, QUANTITY, AVERAGE_PRICE)
547 VALUES (@USER_ID, 1, 2, 100.00);
548
549 EXEC BuyStock @USER_ID, @STOCK_ID, @QUANTITY, @PRICE;
550 GO
551
552 -- VERIFY PORTFOLIO HOLDINGS AFTER BUYING STOCK
553
554
555 FROM PORTFOLIO_HOLDINGS
556 WHERE ACCOUNT_ID = 1 AND STOCK_ID = 1;
557
558 --TEST CASE 2: (BUY STOCK) USER HAS INSUFFICIENT FUNDS
559
560 DECLARE @USER_ID INT = 1;
561 DECLARE @STOCK_ID INT = 2;
562 DECLARE @QUANTITY INT = 1;
563 DECLARE @PRICE DECIMAL(4, 2) = 200.00;
564
565 INSERT INTO PORTFOLIO_HOLDINGS (ACCOUNT_ID, STOCK_ID, QUANTITY, AVERAGE_PRICE)
566 VALUES (@USER_ID, 2, 1, 200.00);
567
568 EXEC BuyStock @USER_ID, @STOCK_ID, @QUANTITY, @PRICE;
569 GO
570
571 SELECT *
572 FROM PORTFOLIO_HOLDINGS

```

Results

PORTFOLIO_ID	ACCOUNT_ID	STOCK_ID	QUANTITY	AVERAGE_PRICE
1	1	1	2	100.00
2	1002	1	2	100.00

PORTFOLIO_ID	ACCOUNT_ID	STOCK_ID	QUANTITY	AVERAGE_PRICE
1	1003	2	1	200.00

PORTFOLIO_ID	ACCOUNT_ID	STOCK_ID	QUANTITY	AVERAGE_PRICE
1	1004	2	1	200.00

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

```

-- GROUP.FINAL_PROJECT_CODE_SUBMISSION_FILE.sql (59) : (1) [Run]
680 USE [GROUP]
681 GO
682 -- USER-DEFINED FUNCTIONS
683 GO
684 DROP FUNCTION IF EXISTS GetTotalPortfolioValue;
685 GO
686 CREATE FUNCTION GetTotalPortfolioValue(@USER_ID INT)
687 RETURNS DECIMAL(15,2)
688 AS
689 BEGIN
690     DECLARE @TOTAL_VALUE DECIMAL(15,2)
691     SET @TOTAL_VALUE = 0.00
692     SELECT @TOTAL_VALUE = SUM(PORTFOLIO_HOLDINGS.QUANTITY * MARKET_PRICES.PRICE)
693     FROM PORTFOLIO_HOLDINGS
694     INNER JOIN STOCKS ON PORTFOLIO_HOLDINGS STOCK_ID = STOCKS STOCK_ID
695     INNER JOIN ACCOUNTS ON PORTFOLIO_HOLDINGS ACCOUNT_ID = ACCOUNTS ACCOUNT_ID
696     WHERE ACCOUNTS USER_ID = @USER_ID
697     RETURN @TOTAL_VALUE
698 END
699 GO
700 -- TEST CASES FOR USER-DEFINED FUNCTIONS
701 GO
702 -- TEST CASE 1: VERIFY PORTFOLIO (USER HAS MULTIPLE STOCKS)
703 GO

```

Messages

```

12:24:45 AM Started executing query at Line 680
12:24:45 AM Commands completed successfully.
12:24:45 AM Started executing query at Line 682
12:24:45 AM Commands completed successfully.
Total execution time: 00:00:00.019

```

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

The screenshot shows the Azure Data Studio interface. In the top navigation bar, 'File', 'Edit', 'View', 'Window', and 'Help' are visible. The main area shows a connection to 'localhost <default> (sa)'. The database is set to 'GROUP7'. A query window displays several lines of SQL code related to 'TEST CASES FOR USER-DEFINED FUNCTIONS'. The results pane shows the output of these queries, including values for 'TOTAL_PORTFOLIO_VALUE' and 'INITIAL_PORTFOLIO_VALUE' across four rows.

```

631 --TEST CASE 1: VERIFY PORTFOLIO USER HAS MULTIPLE STOCKS
632 DECLARE @USER_ID INT = 1
633 SELECT abo.GetTotalPortfolioValue(@USER_ID) AS TOTAL_PORTFOLIO_VALUE
634
641 --TEST CASE 2: VERIFY PORTFOLIO USER HAS NO STOCKS
642 DECLARE @USER_ID INT = 2
643 SELECT abo.GetTotalPortfolioValue(@USER_ID) AS TOTAL_PORTFOLIO_VALUE;
644
645 --TEST CASE 3: VERIFY PORTFOLIO USER HAS A SINGLE STOCK
646

```

	TOTAL_PORTFOLIO_VALUE
1	48265.00

	TOTAL_PORTFOLIO_VALUE
1	448762.50

	TOTAL_PORTFOLIO_VALUE
1	153200.00

	INITIAL_PORTFOLIO_VALUE
1	289500.00

	UpdatedPortfolioValue
1	289500.00

	TOTAL_PORTFOLIO_VALUE
1	597125.00

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

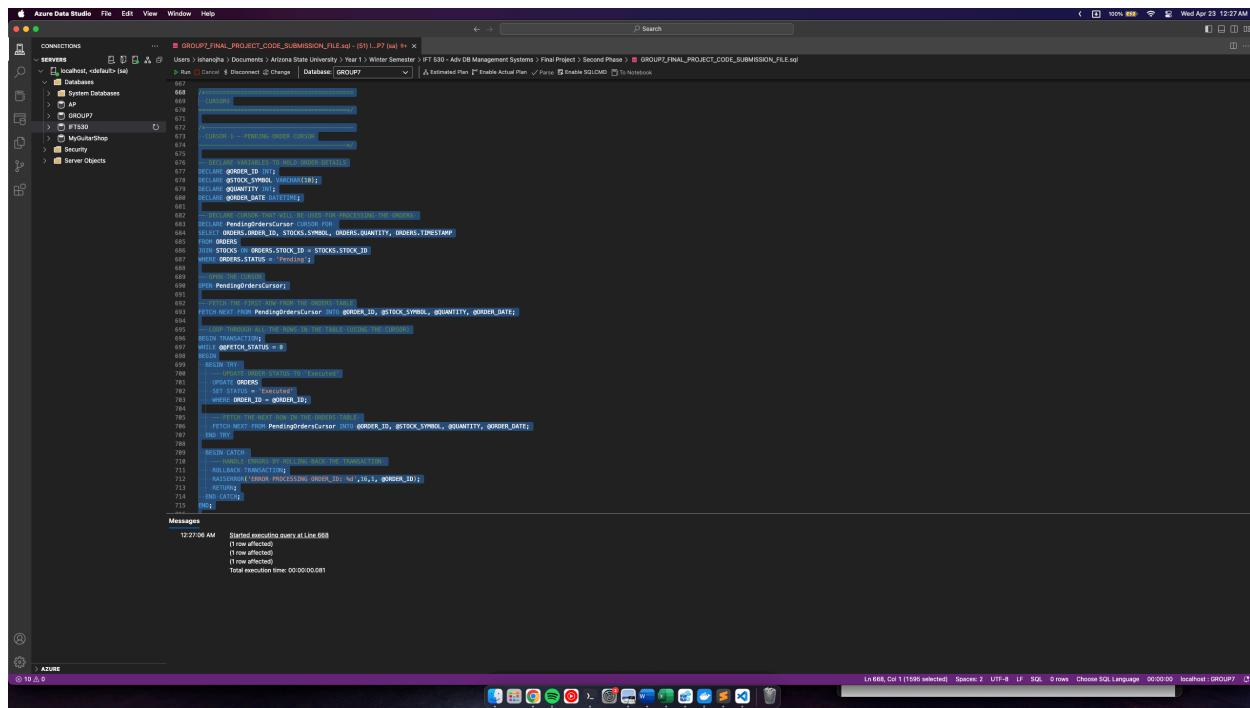
BEGIN CATCH
    -- HANDLE ERRORS BY ROLLING BACK THE TRANSACTION
    ROLLBACK TRANSACTION;
    RAISERROR('ERROR PROCESSING ORDER_ID: %d',16,1, @ORDER_ID);
    RETURN;
END CATCH;
END;

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



A screenshot of the Azure Data Studio interface. The left sidebar shows a tree view of database connections, servers, databases, and objects. The main pane displays a SQL script titled 'GROUP7_FINAL_PROJECT_CODE_SUBMISSION_FILE.sql' with line numbers 661 to 715. The script contains a cursor declaration and processing logic. Below the script, the 'Messages' section shows the execution log:

```
12:27:06 AM Started executing query at Line 661
1 row(s) affected
(1 row affected)
(1 row affected)
Total execution time: 00:00:00.081
```

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

Three screenshots of Azure Data Studio showing the execution of a SQL script named GROUP7_FINAL_PROJECT_CODE_SUBMISSION_FILE.sql. The script contains code for updating order status and fetching pending orders.

```

-- GROUP7_FINAL_PROJECT_CODE_SUBMISSION_FILE.sql (51 L, 877 B) --+
-- GROUP7_FINAL_PROJECT_CODE_SUBMISSION_FILE.sql

685 JOIN STOCKS_IN_ORDERS STOCK_ID = STOCKS STOCK_ID
686 WHERE STOCK_SYMBOL = 'TICKING';
687
688 OPEN PendingOrdersCursor;
689
690 -- PULL THE FIRST ROW FROM THE ORDERS TABLE
691 FETCH NEXT FROM PendingOrdersCursor INTO @order_id, @stock_symbol, @quantity, @order_date;
692
693 -- LOOP THROUGH ALL THE ROWS IN THE TABLE (USING THE CURSOR)
694 BEGIN TRANSACTION;
695 WHILE @@FETCH_STATUS = 0
696 BEGIN TRY
697 UPDATE ORDERS
698 SET STATUS = 'Executed'
699 WHERE ORDER_ID = @order_id;
700
701 -- PULL THE NEXT ROW IN THE ORDERS TABLE
702 FETCH NEXT FROM PendingOrdersCursor INTO @order_id, @stock_symbol, @quantity, @order_date;
703
704 END TRY
705
706 BEGIN CATCH
707 ROLLBACK TRANSACTION;
708 RAISERROR('ERROR PROCESSING ORDER_ID: %d',16,1,@order_id);
709 RETURN;
710 END CATCH
711 GO
712
713 /*
714 * TEST CASES FOR CURSORS
715 */
716
717 -- TEST CASE 1 (PENDING ORDER CURSOR): VIEW PENDING ORDERS BEFORE CURSOR EXECUTION
718 SELECT * FROM ORDERS WHERE STATUS = 'Pending';
719
720 -- TEST CASE 2 (PENDING ORDER CURSOR): VERIFY PROCESSED ORDERS
721 SELECT * FROM ORDERS WHERE STATUS = 'Executed';
722
723 */

Results Messages
ORDER_ID ACCOUNT_ID STOCK_ID ORDER_TYPE PRICE QUANTITY STATUS TIMESTAMP
1 2 2 2 Limit 315.00 5 Pending 2023-04-23 07:16:47.273
2 6 6 6 Limit 390.00 12 Pending 2023-04-23 07:16:47.273
3 8 8 8 Limit 510.00 18 Pending 2023-04-23 07:16:47.273

-- GROUP7_FINAL_PROJECT_CODE_SUBMISSION_FILE.sql (51 L, 877 B) --+
-- GROUP7_FINAL_PROJECT_CODE_SUBMISSION_FILE.sql

695 BEGIN TRY
696 -- UPDATE ORDER STATUS TO 'Executed'
697 UPDATE ORDERS
698 SET STATUS = 'Executed'
699 WHERE ORDER_ID = @order_id;
700
701 -- PULL THE NEXT ROW IN THE ORDERS TABLE
702 FETCH NEXT FROM PendingOrdersCursor INTO @order_id, @stock_symbol, @quantity, @order_date;
703
704 END TRY
705
706 BEGIN CATCH
707 ROLLBACK TRANSACTION;
708 RAISERROR('ERROR PROCESSING ORDER_ID: %d',16,1,@order_id);
709 RETURN;
710 END CATCH
711 GO
712
713 /*
714 * TEST CASES FOR CURSORS
715 */
716
717 -- TEST CASE 1 (PENDING ORDER CURSOR): VIEW PENDING ORDERS BEFORE CURSOR EXECUTION
718 SELECT * FROM ORDERS WHERE STATUS = 'Pending';
719
720 -- TEST CASE 2 (PENDING ORDER CURSOR): VERIFY PROCESSED ORDERS
721 SELECT * FROM ORDERS WHERE STATUS = 'Executed';
722
723 */

Results Messages
ORDER_ID ACCOUNT_ID STOCK_ID ORDER_TYPE PRICE QUANTITY STATUS TIMESTAMP
1 1 1 Market 175.50 10 Executed 2023-04-23 07:16:47.273
2 2 2 2 Limit 315.00 5 Executed 2023-04-23 07:16:47.273
3 3 3 3 Market 2700.00 8 Executed 2023-04-23 07:16:47.273
4 5 5 5 Limit 150.00 20 Executed 2023-04-23 07:16:47.273
5 6 6 6 Limit 390.00 12 Executed 2023-04-23 07:16:47.273
6 7 7 7 Market 685.00 10 Executed 2023-04-23 07:16:47.273
7 8 8 8 Limit 510.00 18 Executed 2023-04-23 07:16:47.273
8 9 9 9 Market 190.00 7 Executed 2023-04-23 07:16:47.273

```