Student’s Full Name: Harsh Homdeo Wasnik
Course Title: SQL/NoSQL Databases for Data and Information Sciences
Term name and year: Fall 2024
Submission Week: Week 16 Final Project Report
Instructor’s Name: Nayem Rahman
Date of Submission: 16th Dec’24

Week 16 Final Project Report: Total Points - 100

Week 8 Final Project Update 1

1. Brief Description of the Project

Business Plan:

The project focuses on building a detailed database of traders and their activities within a stock market trading firm, ***Market Prime Trading Solutions***. The goal is to organize and store information about traders, their equipment needs, specialization areas, and training programs in a structured format.

Objective:

The project’s objective is to create a centralized repository that can be used for efficient management of trading resources and to facilitate data-driven decision-making. This structure allows for easy access to information, making it suitable for:

- Analyzing trader performance across various specialized areas.
- Streamlining equipment allocation to meet the operational needs of different trading departments.
- Identifying and managing training programs to upskill traders in key areas.

2. The Data used and the Source

Data and Source:

The data used in this project is custom created to simulate a stock market trading firm’s operation. It includes detailed information about traders, brokers, stocks, and sectors designed to reflect the realistic needs and structure of a trading environment. The database is structured into two primary tables:

- **Data1 (Trader Profiles):**
  Contains individual trader profiles, including:
  - Contact information
  - Addresses
  - Hire dates
  - Broker
  - Stocks (they traded)

Each trader’s data is structured to provide a comprehensive view of their role within the firm.

- **Data2 (Trading Departments):**
  Focuses on the trading firm's departments, listing the specific equipment required for each trading area, such as:
  - Stocks
  - Sectors
  - Broker

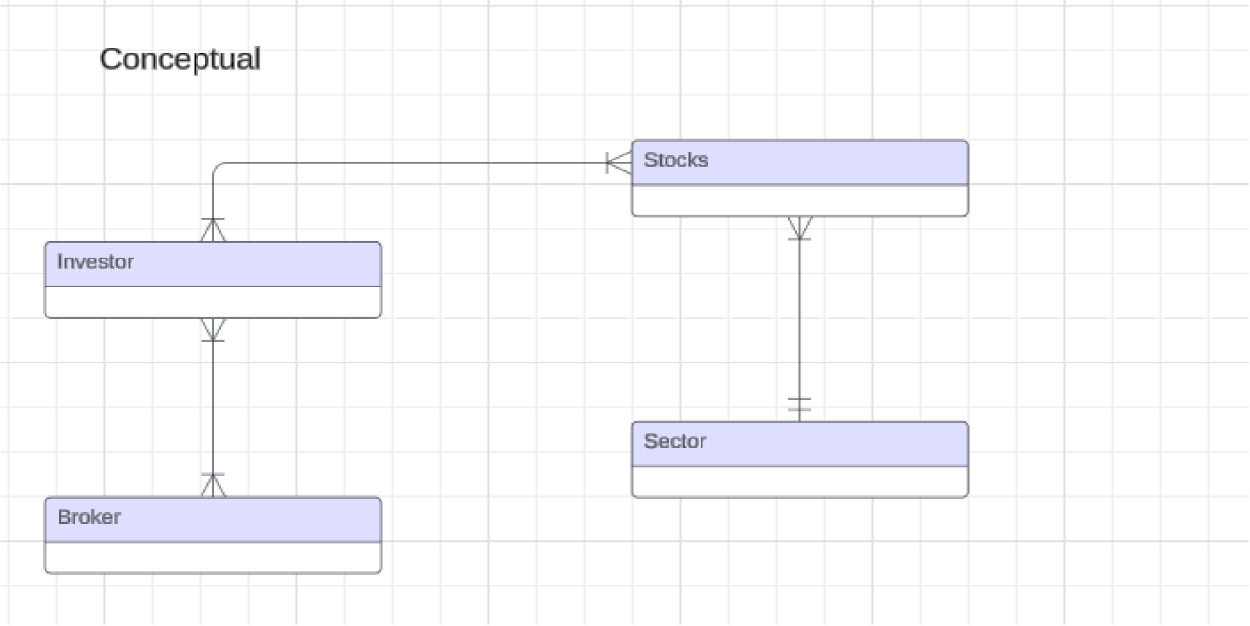
<u>Week 11 Final Project Update 2:</u>

3. Develop a Conceptual Model with 5 or 6 entities in it. Make sure you have at least one many-to-many relationship that exists in your conceptual model. Explain with data why it's a many-to-many relationship.

Entities and Relationships:

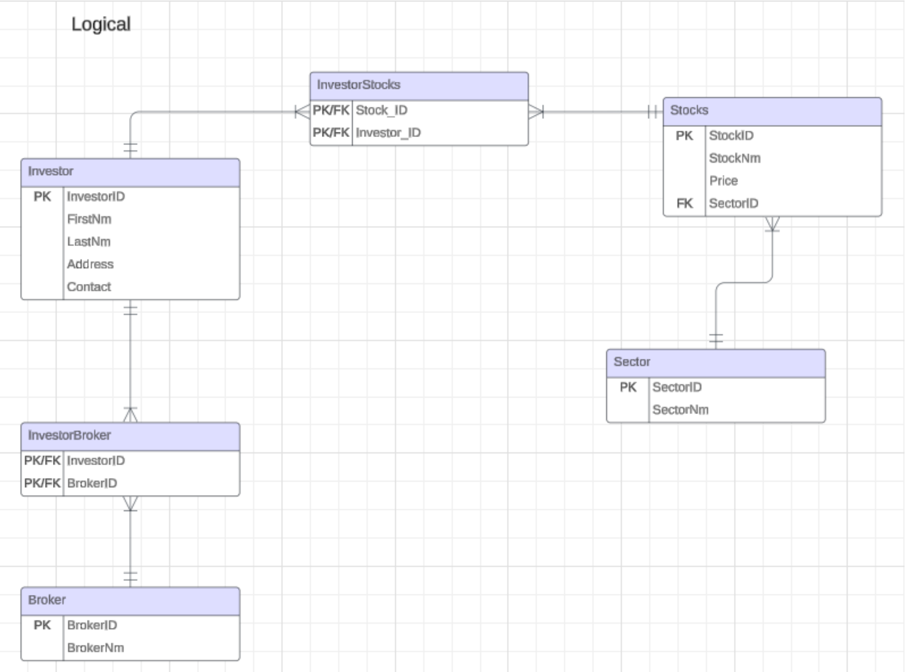
1. <u>Investor and Broker:</u>
   - **Many-to-Many Relationship:** An investor can work with multiple brokers, and a broker can serve multiple investors. This relationship suggests that both investors and brokers engage in transactions and management activities across a diverse range of client-broker pairings.

2. <u>Investor and Stocks:</u>
   - **Many-to-Many Relationship:** Investors can own multiple stocks, and each stock can be owned by multiple investors.



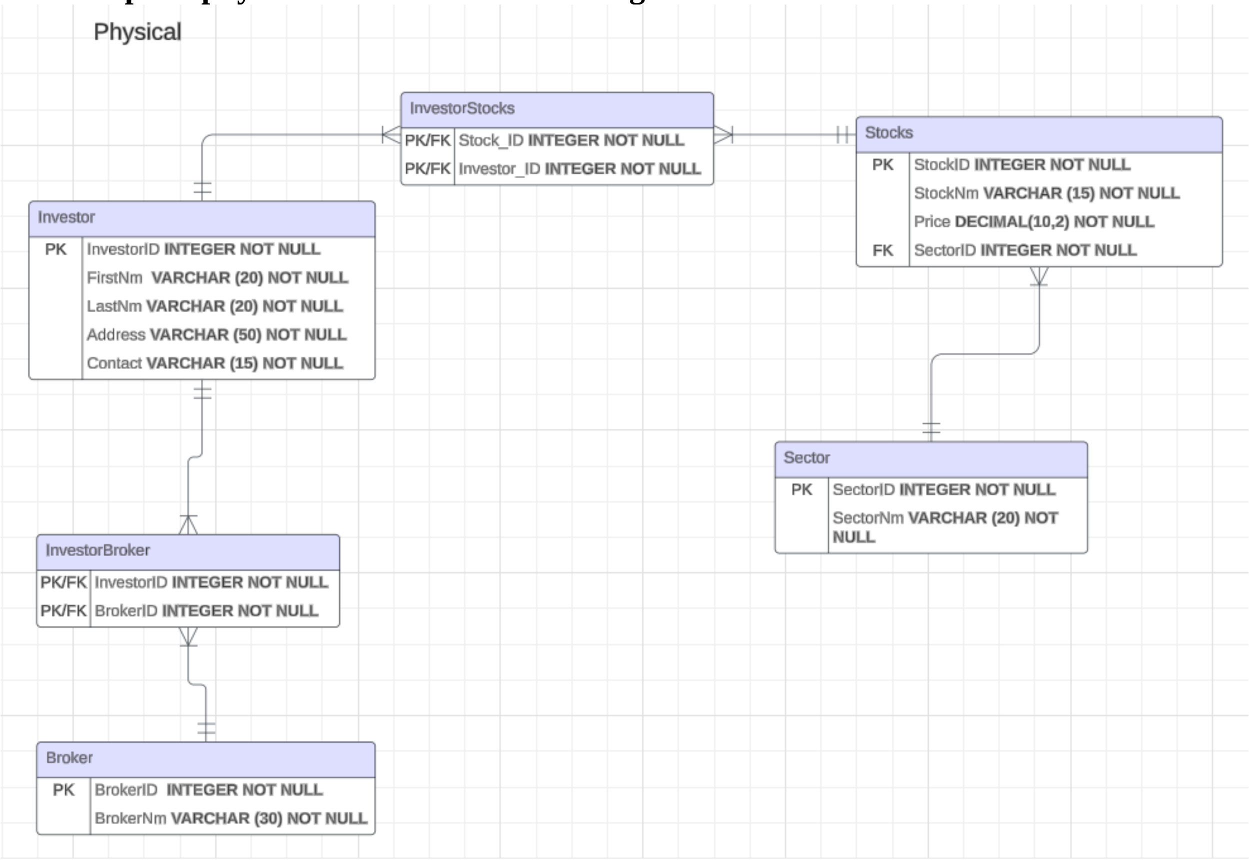
3. <u>Stocks and Sector:</u>
   - **Many-to-One Relationship:** Multiple stocks can belong to one sector, indicating that while a stock has a single sector classification, a sector can encompass various stocks.

4. Develop a Logical Model using the Conceptual Model. Make sure you come up with a junction entity to resolve the many-to-many relationship.



- **Junction Table for Investor and Broker:** This table would manage the many-to-many relationship by recording pairs of Investor IDs and Broker IDs, each pair representing a business relationship.
- **Junction Table for Investor and Stocks:** Similarly, another junction table would be necessary to accurately record which stocks are owned by which investors, capturing the many-to-many nature of this relationship as well.

5. Develop the physical model based on the Logical Model

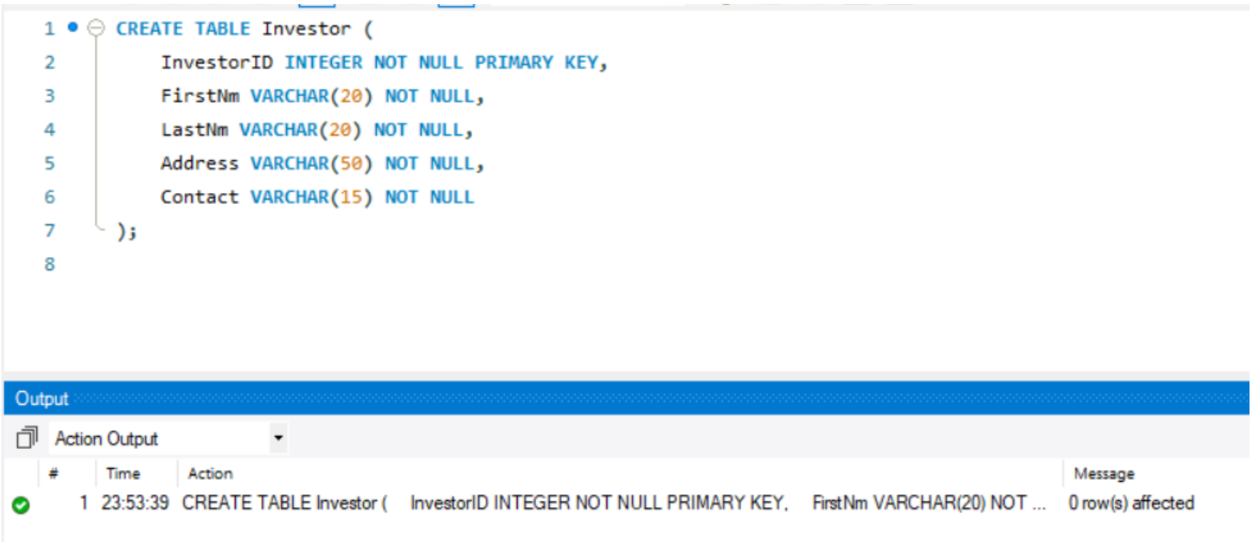

Physical
InvestorStocks
PK/FK Stock_ID INTEGER NOT NULL
PK/FK Investor_ID INTEGER NOT NULL
Stocks
PK StockID INTEGER NOT NULL
StockNm VARCHAR (15) NOT NULL
Price DECIMAL(10,2) NOT NULL
FK SectorID INTEGER NOT NULL
Investor
PK InvestorID INTEGER NOT NULL
FirstNm VARCHAR (20) NOT NULL
LastNm VARCHAR (20) NOT NULL
Address VARCHAR (50) NOT NULL
Contact VARCHAR (15) NOT NULL
Sector
PK SectorID INTEGER NOT NULL
SectorNm VARCHAR (20) NOT NULL
InvestorBroker
PK/FK InvestorID INTEGER NOT NULL
PK/FK BrokerID INTEGER NOT NULL
Broker
PK BrokerID INTEGER NOT NULL
BrokerNm VARCHAR (30) NOT NULL

6. Create tables using a database system. Insert data into the database tables. You must provide the DDL (CREATE TABLE statements), INSERT statements, and SELECT statements.

Details: Create the tables that you have come up with (the table must be based on the Physical Model).
(a) Columns, Primary Key (PK), Data Type and length, and NULL/NOT NULL need to be implemented, per the Physical Model.
(b) Show the table definition (DDL) that you implemented (not in a graphical view).
(c) Insert the complete set of data that you have come up with and show the insert statements used.

1. Table Creation (DDL)

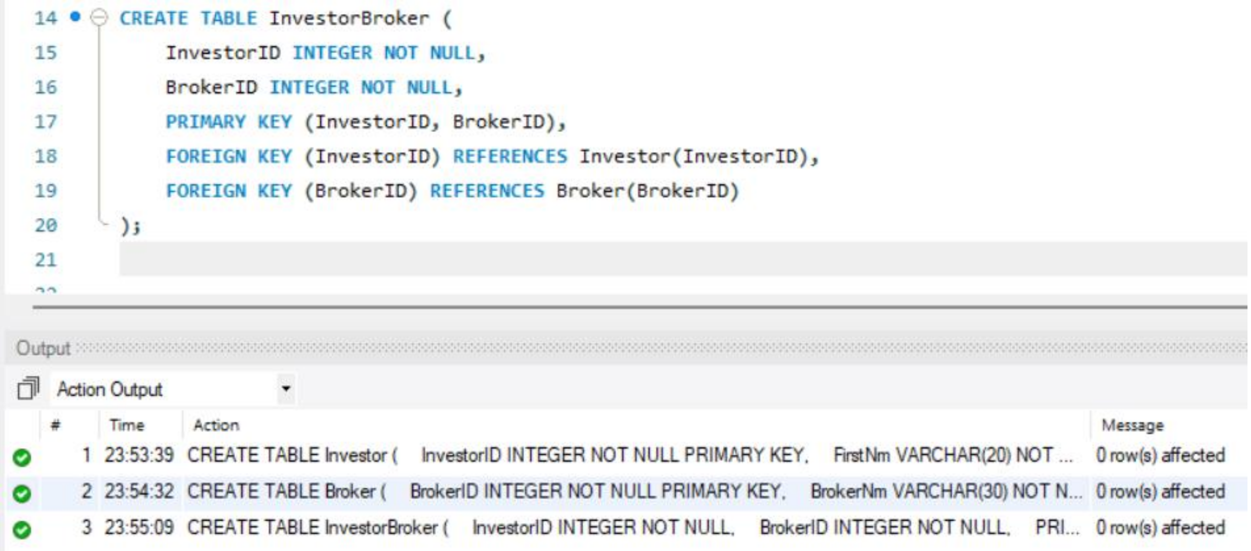
Table: Investor



```
CREATE TABLE Investor (
    InvestorID INTEGER NOT NULL PRIMARY KEY,
    FirstNm VARCHAR(20) NOT NULL,
    LastNm VARCHAR(20) NOT NULL,
    Address VARCHAR(50) NOT NULL,
    Contact VARCHAR(15) NOT NULL
);
```

```
   #  Time      Action                                                                                    Message
✓  1  23:53:39  CREATE TABLE Investor (   InvestorID INTEGER NOT NULL PRIMARY KEY,   FirstNm VARCHAR(20) NOT ...   0 row(s) affected
```

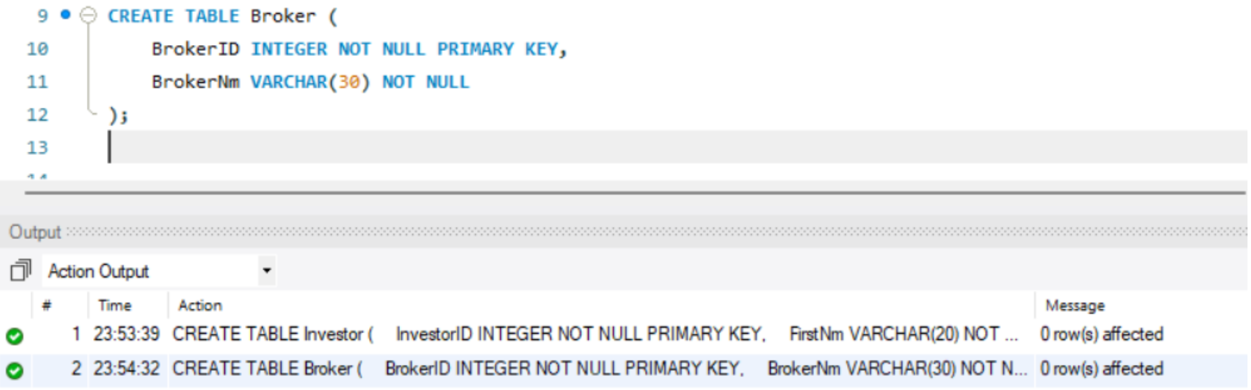
Table: InvestorBroker



```
CREATE TABLE InvestorBroker (
    InvestorID INTEGER NOT NULL,
    BrokerID INTEGER NOT NULL,
    PRIMARY KEY (InvestorID, BrokerID),
    FOREIGN KEY (InvestorID) REFERENCES Investor(InvestorID),
    FOREIGN KEY (BrokerID) REFERENCES Broker(BrokerID)
);
```

```
   #  Time      Action                                                                                        Message
✓  1  23:53:39  CREATE TABLE Investor (   InvestorID INTEGER NOT NULL PRIMARY KEY,   FirstNm VARCHAR(20) NOT ...   0 row(s) affected
✓  2  23:54:32  CREATE TABLE Broker (   BrokerID INTEGER NOT NULL PRIMARY KEY,   BrokerNm VARCHAR(30) NOT N...     0 row(s) affected
✓  3  23:55:09  CREATE TABLE InvestorBroker (   InvestorID INTEGER NOT NULL,   BrokerID INTEGER NOT NULL,   PRI...  0 row(s) affected
```

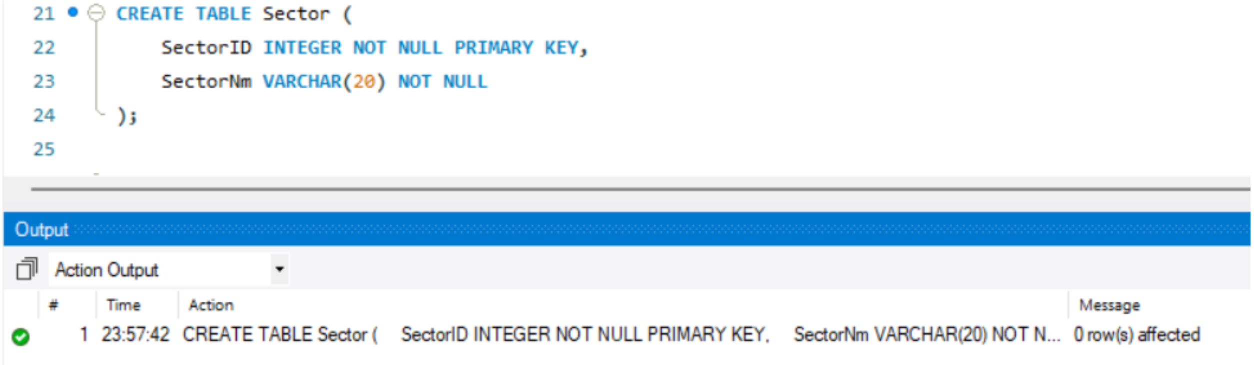
Table: Broker



```sql
CREATE TABLE Broker (
    BrokerID INTEGER NOT NULL PRIMARY KEY,
    BrokerNm VARCHAR(30) NOT NULL
);
```

```
#   Time      Action                                                                                    Message
1   23:53:39  CREATE TABLE Investor (   InvestorID INTEGER NOT NULL PRIMARY KEY,   FirstNm VARCHAR(20) NOT ...   0 row(s) affected
2   23:54:32  CREATE TABLE Broker (   BrokerID INTEGER NOT NULL PRIMARY KEY,   BrokerNm VARCHAR(30) NOT N...      0 row(s) affected
```

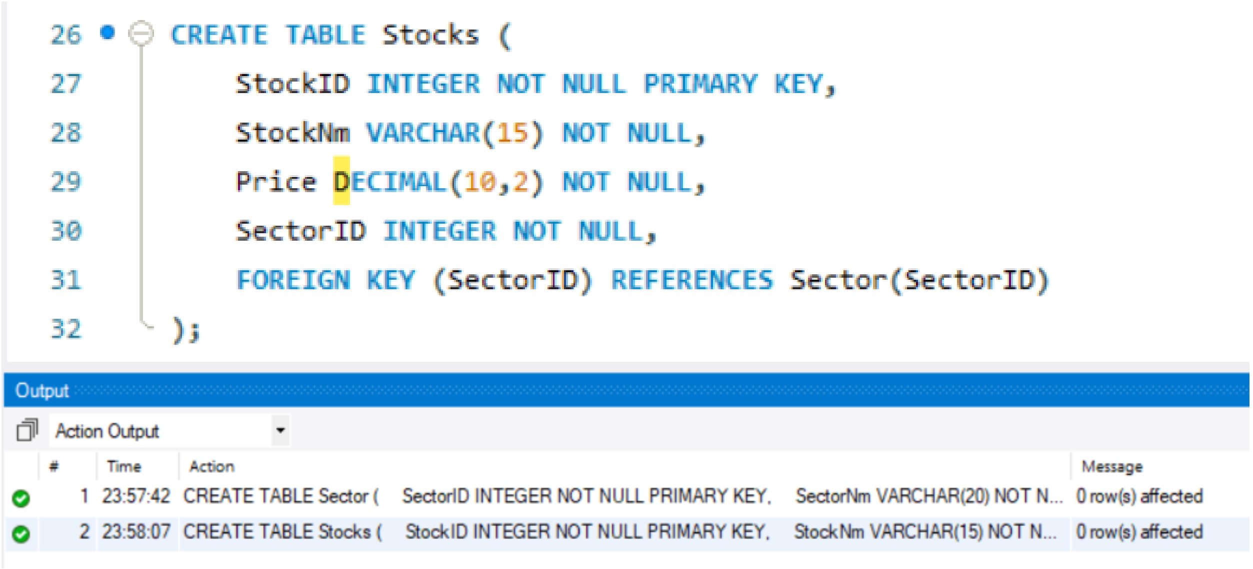
Table: Sector



```sql
CREATE TABLE Sector (
    SectorID INTEGER NOT NULL PRIMARY KEY,
    SectorNm VARCHAR(20) NOT NULL
);
```

```
#   Time      Action                                                                                Message
1   23:57:42  CREATE TABLE Sector (   SectorID INTEGER NOT NULL PRIMARY KEY,   SectorNm VARCHAR(20) NOT N...   0 row(s) affected
```

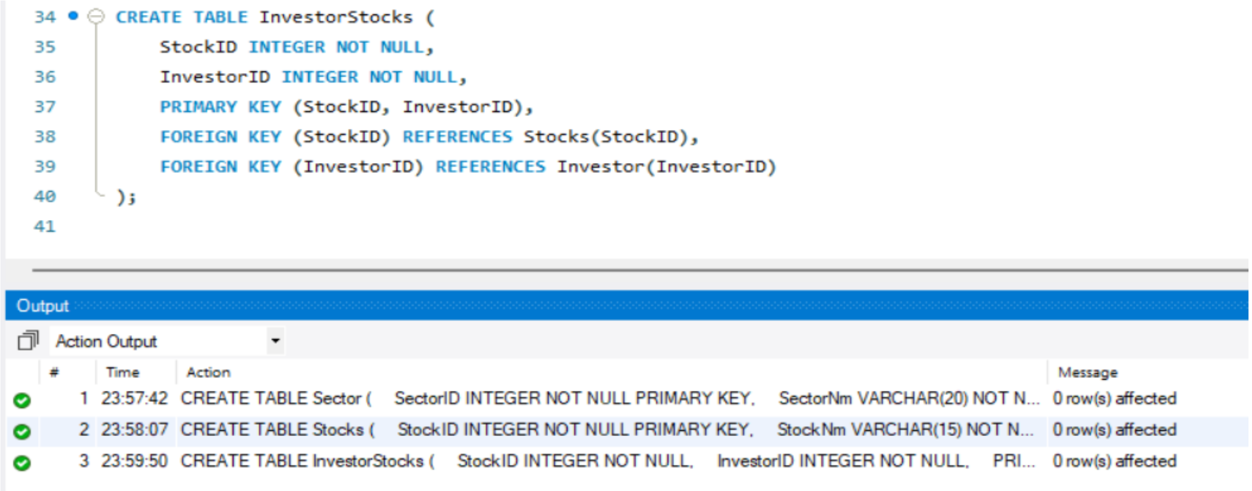
Table: Stocks



```sql
CREATE TABLE Stocks (
    StockID INTEGER NOT NULL PRIMARY KEY,
    StockNm VARCHAR(15) NOT NULL,
    Price DECIMAL(10,2) NOT NULL,
    SectorID INTEGER NOT NULL,
    FOREIGN KEY (SectorID) REFERENCES Sector(SectorID)
);
```

```
#   Time      Action                                                                                Message
1   23:57:42  CREATE TABLE Sector (   SectorID INTEGER NOT NULL PRIMARY KEY,   SectorNm VARCHAR(20) NOT N...   0 row(s) affected
2   23:58:07  CREATE TABLE Stocks (   StockID INTEGER NOT NULL PRIMARY KEY,   StockNm VARCHAR(15) NOT N...     0 row(s) affected
```

Table: InvestorStocks

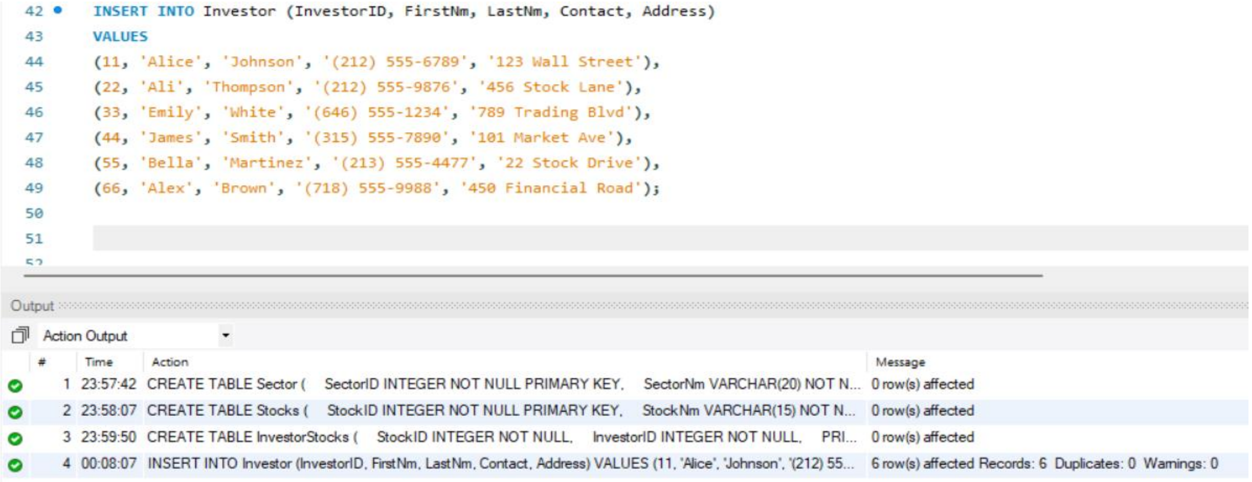


```sql
CREATE TABLE InvestorStocks (
    StockID INTEGER NOT NULL,
    InvestorID INTEGER NOT NULL,
    PRIMARY KEY (StockID, InvestorID),
    FOREIGN KEY (StockID) REFERENCES Stocks(StockID),
    FOREIGN KEY (InvestorID) REFERENCES Investor(InvestorID)
);
```

```
 #  Time      Action                                                                                                Message
 1  23:57:42  CREATE TABLE Sector (   SectorID INTEGER NOT NULL PRIMARY KEY,   SectorNm VARCHAR(20) NOT N...    0 row(s) affected
 2  23:58:07  CREATE TABLE Stocks (   StockID INTEGER NOT NULL PRIMARY KEY,   StockNm VARCHAR(15) NOT N...      0 row(s) affected
 3  23:59:50  CREATE TABLE InvestorStocks (   StockID INTEGER NOT NULL,   InvestorID INTEGER NOT NULL,   PRI...  0 row(s) affected
```

2. **Insert Statements**

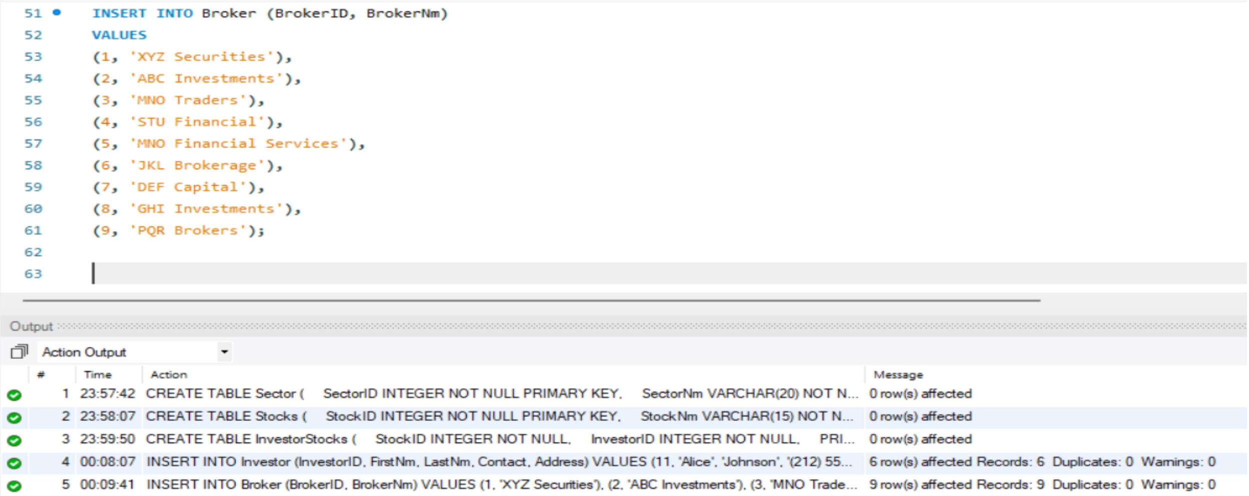
Insert into Investor



```sql
INSERT INTO Investor (InvestorID, FirstNm, LastNm, Contact, Address)
VALUES
(11, 'Alice', 'Johnson', '(212) 555-6789', '123 Wall Street'),
(22, 'Ali', 'Thompson', '(212) 555-9876', '456 Stock Lane'),
(33, 'Emily', 'White', '(646) 555-1234', '789 Trading Blvd'),
(44, 'James', 'Smith', '(315) 555-7890', '101 Market Ave'),
(55, 'Bella', 'Martinez', '(213) 555-4477', '22 Stock Drive'),
(66, 'Alex', 'Brown', '(718) 555-9988', '450 Financial Road');
```

```
 #  Time      Action                                                                                                Message
 1  23:57:42  CREATE TABLE Sector (   SectorID INTEGER NOT NULL PRIMARY KEY,   SectorNm VARCHAR(20) NOT N...    0 row(s) affected
 2  23:58:07  CREATE TABLE Stocks (   StockID INTEGER NOT NULL PRIMARY KEY,   StockNm VARCHAR(15) NOT N...      0 row(s) affected
 3  23:59:50  CREATE TABLE InvestorStocks (   StockID INTEGER NOT NULL,   InvestorID INTEGER NOT NULL,   PRI...  0 row(s) affected
 4  00:08:07  INSERT INTO Investor (InvestorID, FirstNm, LastNm, Contact, Address) VALUES (11, 'Alice', 'Johnson', '(212) 55...  6 row(s) affected Records: 6  Duplicates: 0  Warnings: 0
```

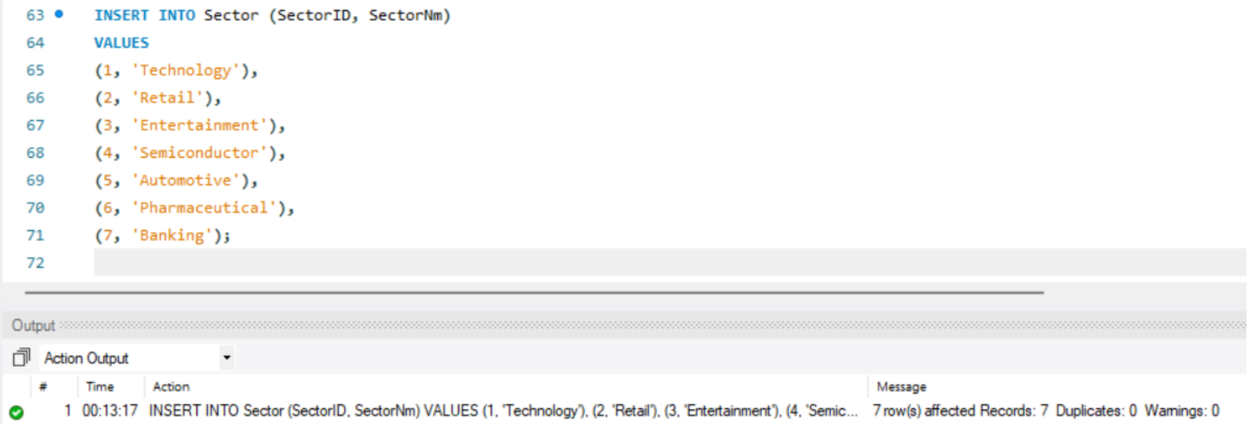
Insert into Broker



```sql
INSERT INTO Broker (BrokerID, BrokerNm)
VALUES
(1, 'XYZ Securities'),
(2, 'ABC Investments'),
(3, 'MNO Traders'),
(4, 'STU Financial'),
(5, 'MNO Financial Services'),
(6, 'JKL Brokerage'),
(7, 'DEF Capital'),
(8, 'GHI Investments'),
(9, 'PQR Brokers');
```

```
#  Time      Action                                                                                                        Message
1  23:57:42  CREATE TABLE Sector (   SectorID INTEGER NOT NULL PRIMARY KEY,    SectorNm VARCHAR(20) NOT N...                  0 row(s) affected
2  23:58:07  CREATE TABLE Stocks (   StockID INTEGER NOT NULL PRIMARY KEY,    StockNm VARCHAR(15) NOT N...                    0 row(s) affected
3  23:59:50  CREATE TABLE InvestorStocks (   StockID INTEGER NOT NULL,    InvestorID INTEGER NOT NULL,    PRI...              0 row(s) affected
4  00:08:07  INSERT INTO Investor (InvestorID, FirstNm, LastNm, Contact, Address) VALUES (11, 'Alice', 'Johnson', '(212) 55...  6 row(s) affected Records: 6  Duplicates: 0  Warnings: 0
5  00:09:41  INSERT INTO Broker (BrokerID, BrokerNm) VALUES (1, 'XYZ Securities'), (2, 'ABC Investments'), (3, 'MNO Trade...    9 row(s) affected Records: 9  Duplicates: 0  Warnings: 0
```

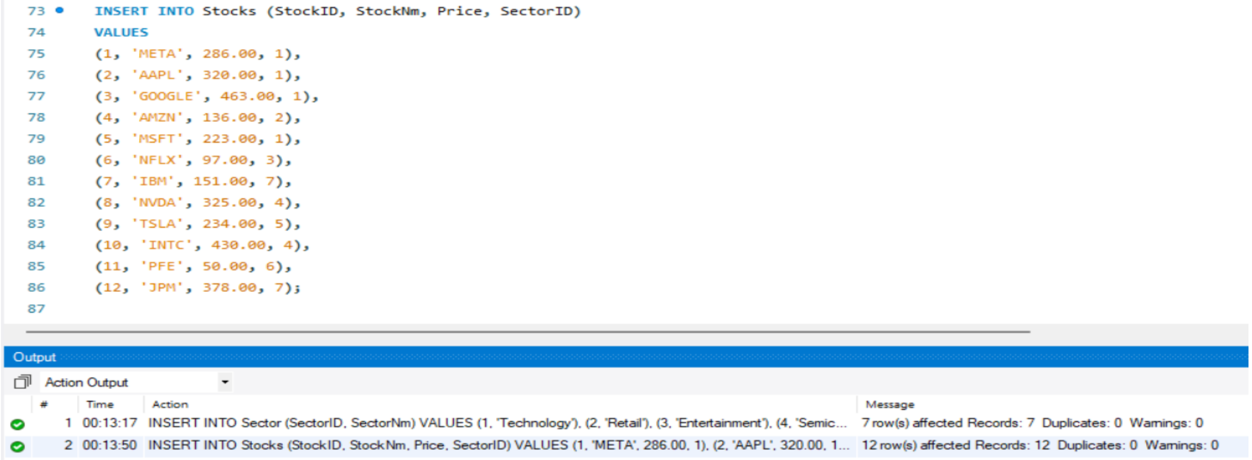
Insert into Sector



```sql
INSERT INTO Sector (SectorID, SectorNm)
VALUES
(1, 'Technology'),
(2, 'Retail'),
(3, 'Entertainment'),
(4, 'Semiconductor'),
(5, 'Automotive'),
(6, 'Pharmaceutical'),
(7, 'Banking');
```

```
#  Time      Action                                                                                                    Message
1  00:13:17  INSERT INTO Sector (SectorID, SectorNm) VALUES (1, 'Technology'), (2, 'Retail'), (3, 'Entertainment'), (4, 'Semic...  7 row(s) affected Records: 7  Duplicates: 0  Warnings: 0
```

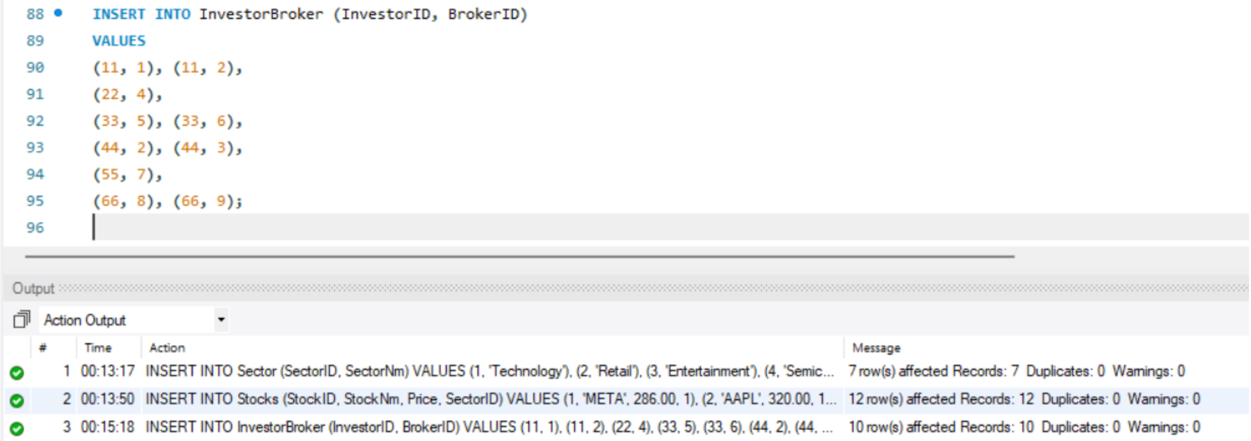
Insert into Stocks



```sql
INSERT INTO Stocks (StockID, StockNm, Price, SectorID)
VALUES
(1, 'META', 286.00, 1),
(2, 'AAPL', 320.00, 1),
(3, 'GOOGLE', 463.00, 1),
(4, 'AMZN', 136.00, 2),
(5, 'MSFT', 223.00, 1),
(6, 'NFLX', 97.00, 3),
(7, 'IBM', 151.00, 7),
(8, 'NVDA', 325.00, 4),
(9, 'TSLA', 234.00, 5),
(10, 'INTC', 430.00, 4),
(11, 'PFE', 50.00, 6),
(12, 'JPM', 378.00, 7);
```

```
#  Time      Action                                                                                                    Message
1  00:13:17  INSERT INTO Sector (SectorID, SectorNm) VALUES (1, 'Technology'), (2, 'Retail'), (3, 'Entertainment'), (4, 'Semic...  7 row(s) affected Records: 7  Duplicates: 0  Warnings: 0
2  00:13:50  INSERT INTO Stocks (StockID, StockNm, Price, SectorID) VALUES (1, 'META', 286.00, 1), (2, 'AAPL', 320.00, 1...   12 row(s) affected Records: 12  Duplicates: 0  Warnings: 0
```

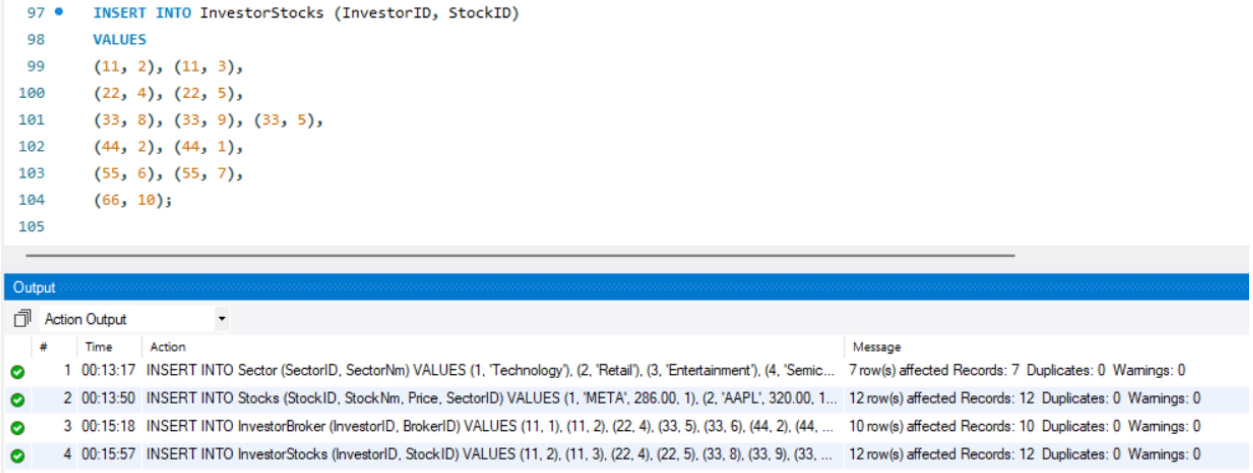
Insert into InvestorBroker



```sql
INSERT INTO InvestorBroker (InvestorID, BrokerID)
VALUES
(11, 1), (11, 2),
(22, 4),
(33, 5), (33, 6),
(44, 2), (44, 3),
(55, 7),
(66, 8), (66, 9);
```

	#	Time	Action	Message
✓	1	00:13:17	INSERT INTO Sector (SectorID, SectorNm) VALUES (1, 'Technology'), (2, 'Retail'), (3, 'Entertainment'), (4, 'Semic...	7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0
✓	2	00:13:50	INSERT INTO Stocks (StockID, StockNm, Price, SectorID) VALUES (1, 'META', 286.00, 1), (2, 'AAPL', 320.00, 1...	12 row(s) affected Records: 12 Duplicates: 0 Warnings: 0
✓	3	00:15:18	INSERT INTO InvestorBroker (InvestorID, BrokerID) VALUES (11, 1), (11, 2), (22, 4), (33, 5), (33, 6), (44, 2), (44, ...	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0

Insert into InvestorStocks



```sql
INSERT INTO InvestorStocks (InvestorID, StockID)
VALUES
(11, 2), (11, 3),
(22, 4), (22, 5),
(33, 8), (33, 9), (33, 5),
(44, 2), (44, 1),
(55, 6), (55, 7),
(66, 10);
```

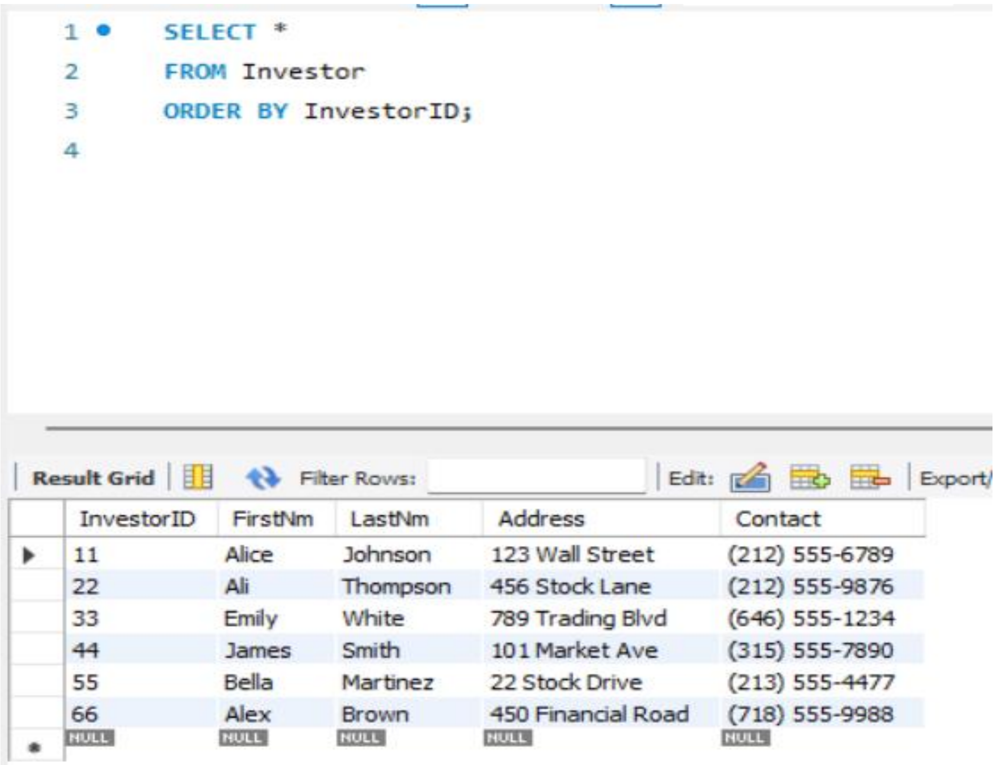
	#	Time	Action	Message
✓	1	00:13:17	INSERT INTO Sector (SectorID, SectorNm) VALUES (1, 'Technology'), (2, 'Retail'), (3, 'Entertainment'), (4, 'Semic...	7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0
✓	2	00:13:50	INSERT INTO Stocks (StockID, StockNm, Price, SectorID) VALUES (1, 'META', 286.00, 1), (2, 'AAPL', 320.00, 1...	12 row(s) affected Records: 12 Duplicates: 0 Warnings: 0
✓	3	00:15:18	INSERT INTO InvestorBroker (InvestorID, BrokerID) VALUES (11, 1), (11, 2), (22, 4), (33, 5), (33, 6), (44, 2), (44, ...	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0
✓	4	00:15:57	INSERT INTO InvestorStocks (InvestorID, StockID) VALUES (11, 2), (11, 3), (22, 4), (22, 5), (33, 8), (33, 9), (33, ...	12 row(s) affected Records: 12 Duplicates: 0 Warnings: 0

7. Create a variety of SQL queries to retrieve data from one or many tables:

1. Retrieve the data from each table by using the SELECT * statement and order by PK column(s).

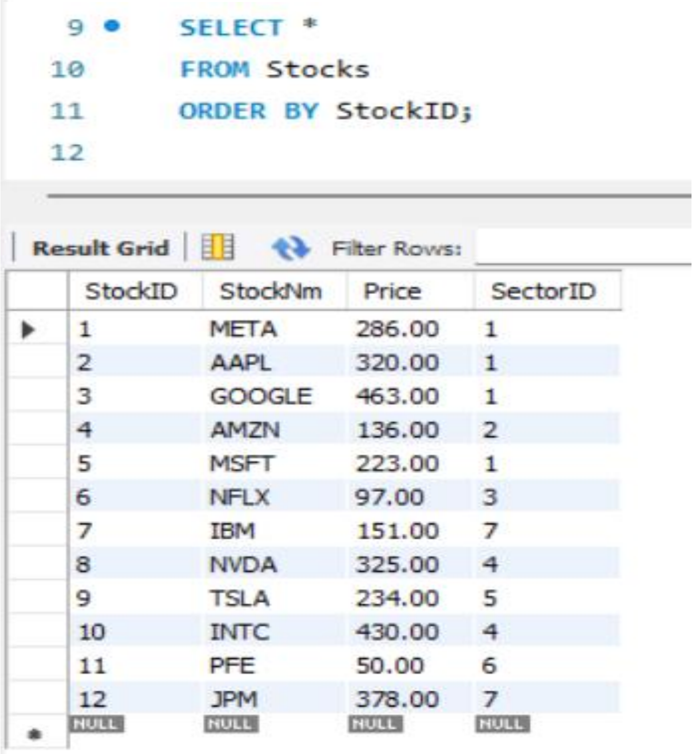
Show the output. Make sure you show the print screen of the complete set of rows and columns.

The rows must be ordered by PK column(s).



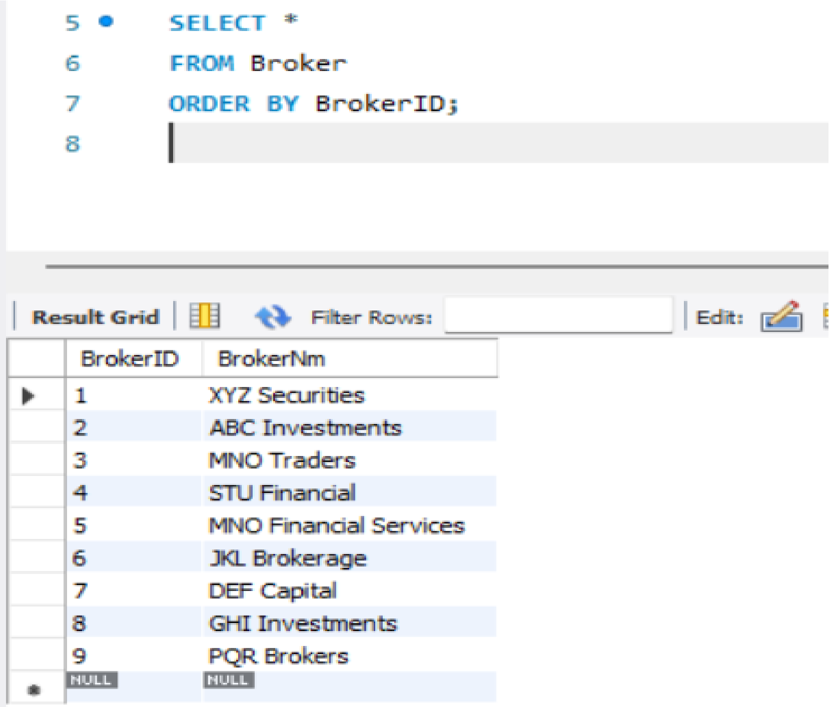
```
SELECT *
FROM Investor
ORDER BY InvestorID;
```

InvestorID	FirstNm	LastNm	Address	Contact
11	Alice	Johnson	123 Wall Street	(212) 555-6789
22	Ali	Thompson	456 Stock Lane	(212) 555-9876
33	Emily	White	789 Trading Blvd	(646) 555-1234
44	James	Smith	101 Market Ave	(315) 555-7890
55	Bella	Martinez	22 Stock Drive	(213) 555-4477
66	Alex	Brown	450 Financial Road	(718) 555-9988
NULL	NULL	NULL	NULL	NULL



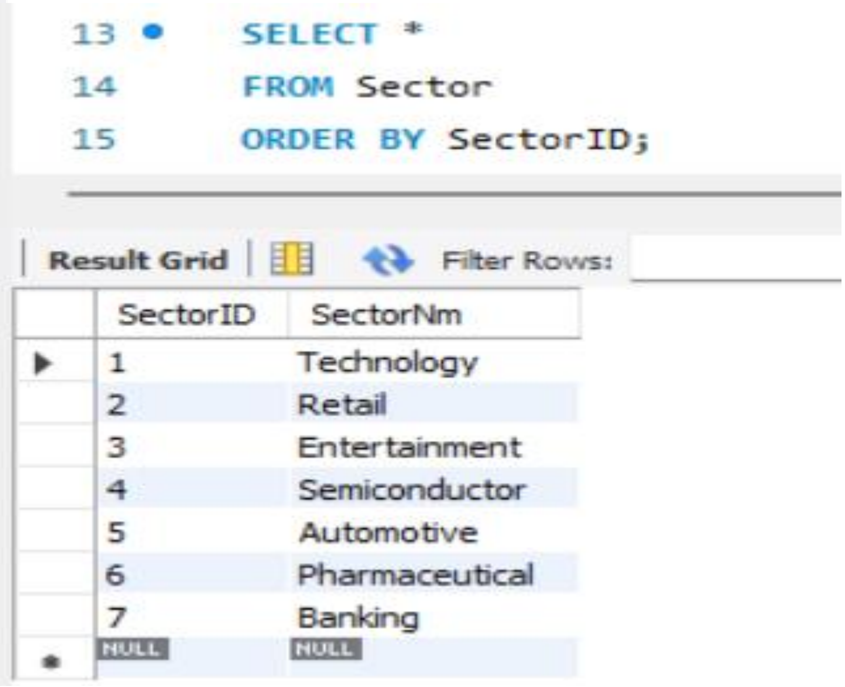
```
SELECT *
FROM Stocks
ORDER BY StockID;
```

StockID	StockNm	Price	SectorID
1	META	286.00	1
2	AAPL	320.00	1
3	GOOGLE	463.00	1
4	AMZN	136.00	2
5	MSFT	223.00	1
6	NFLX	97.00	3
7	IBM	151.00	7
8	NVDA	325.00	4
9	TSLA	234.00	5
10	INTC	430.00	4
11	PFE	50.00	6
12	JPM	378.00	7
NULL	NULL	NULL	NULL



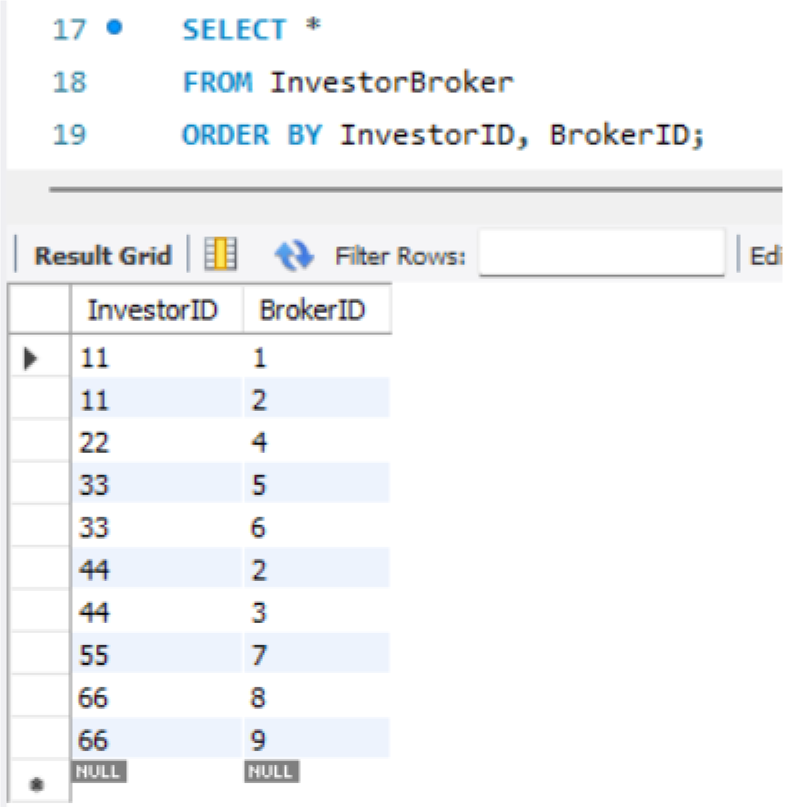
```sql
SELECT *
FROM Broker
ORDER BY BrokerID;
```

BrokerID	BrokerNm
1	XYZ Securities
2	ABC Investments
3	MNO Traders
4	STU Financial
5	MNO Financial Services
6	JKL Brokerage
7	DEF Capital
8	GHI Investments
9	PQR Brokers
NULL	NULL



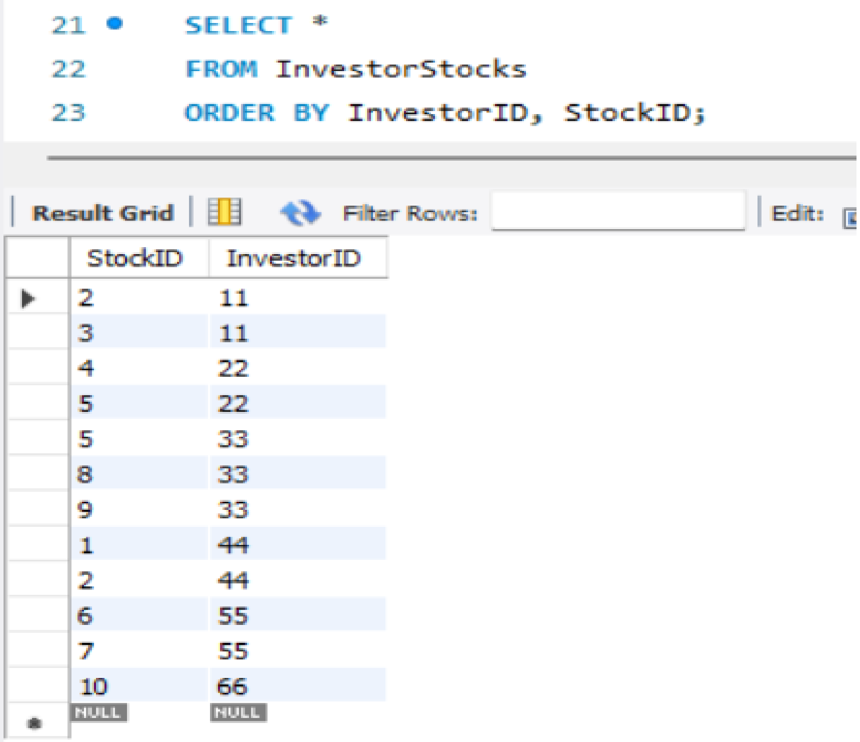
```sql
SELECT *
FROM Sector
ORDER BY SectorID;
```

SectorID	SectorNm
1	Technology
2	Retail
3	Entertainment
4	Semiconductor
5	Automotive
6	Pharmaceutical
7	Banking
NULL	NULL



```sql
SELECT *
FROM InvestorBroker
ORDER BY InvestorID, BrokerID;
```

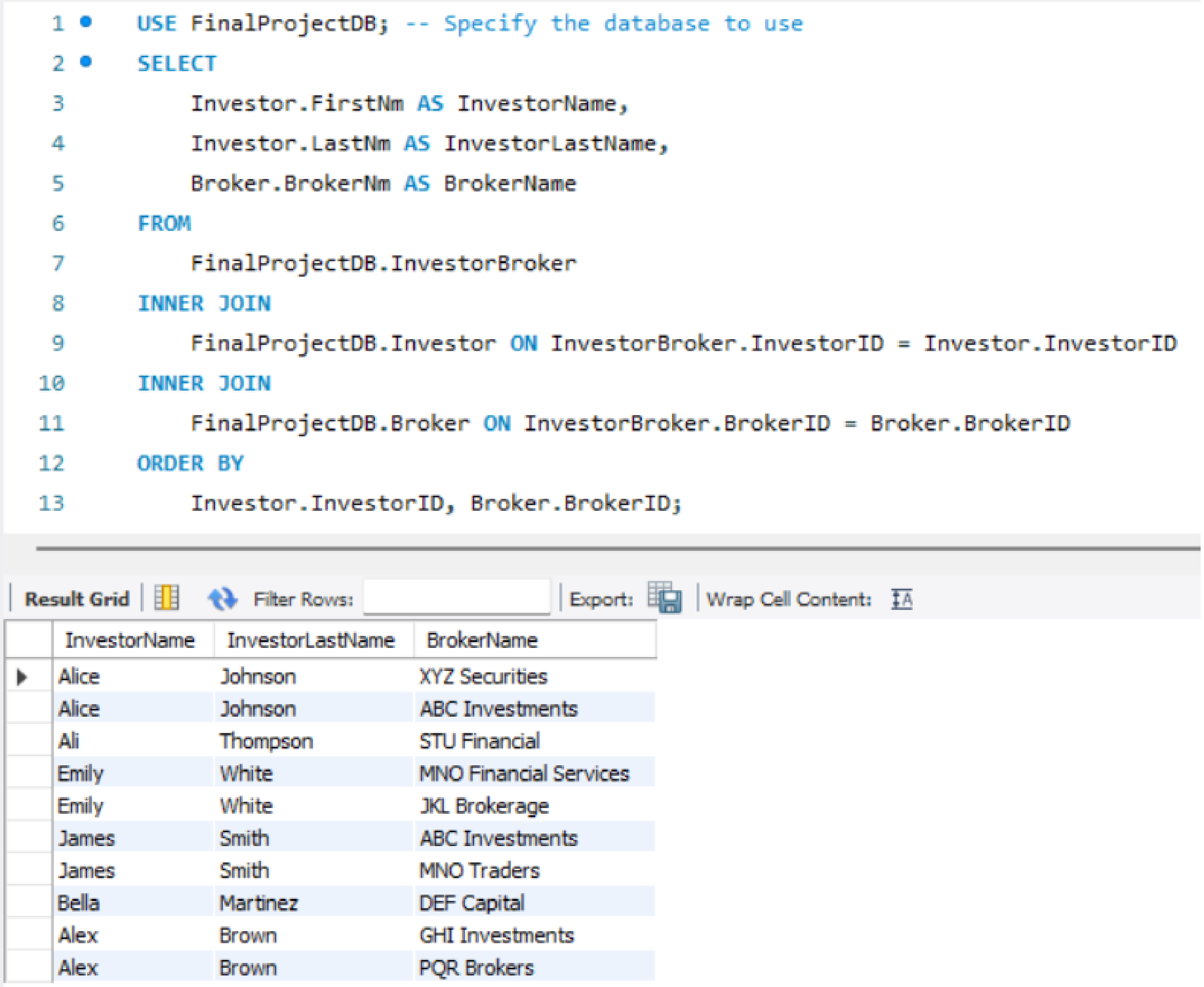
InvestorID	BrokerID
11	1
11	2
22	4
33	5
33	6
44	2
44	3
55	7
66	8
66	9
NULL	NULL



```sql
SELECT *
FROM InvestorStocks
ORDER BY InvestorID, StockID;
```

StockID	InvestorID
2	11
3	11
4	22
5	22
5	33
8	33
9	33
1	44
2	44
6	55
7	55
10	66
NULL	NULL

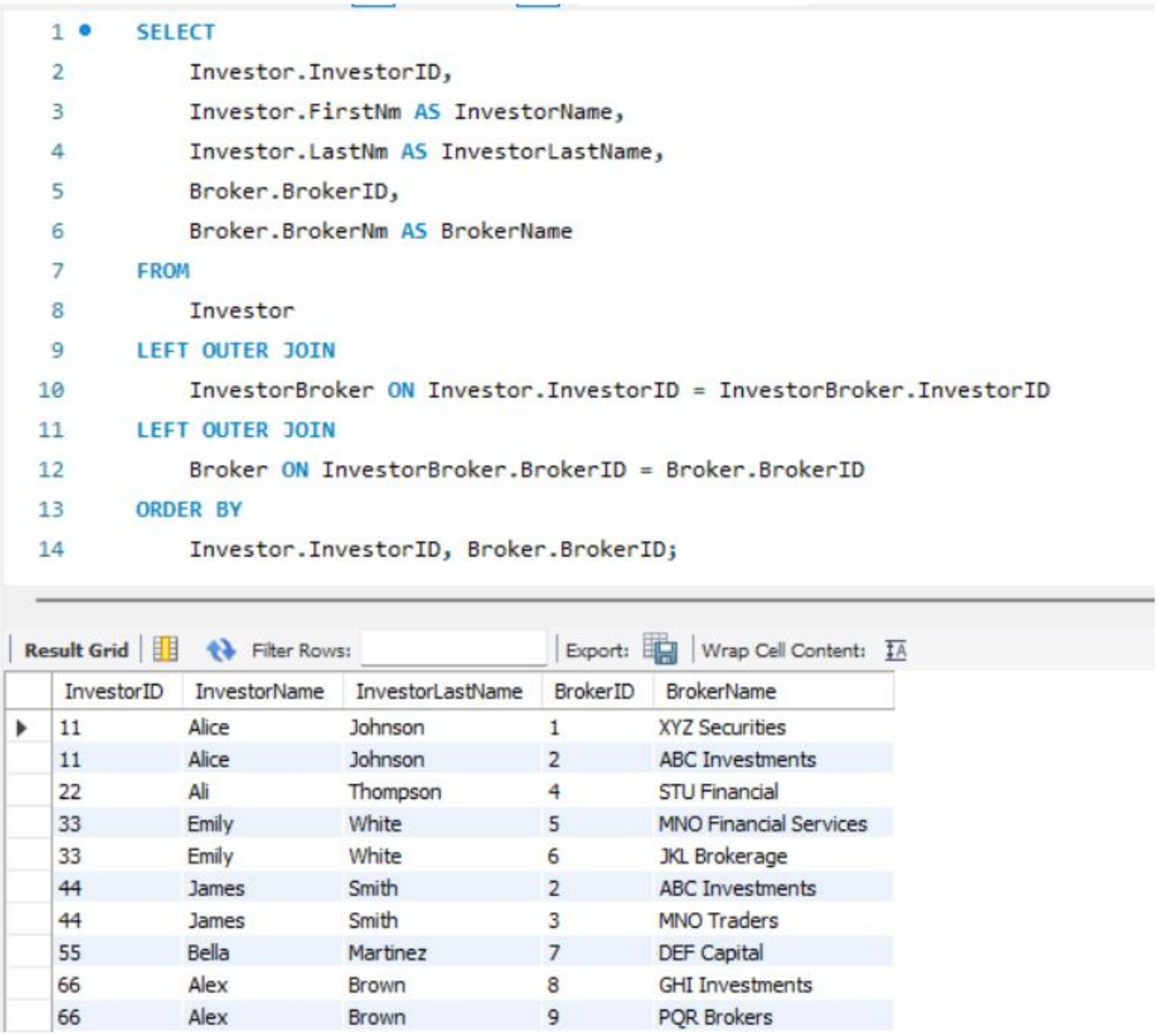
2. Write an SQL involving the junction table and two other related tables. You must use the INNER JOIN to connect with all three tables. The database that you created must be included in your SQL queries.



```
USE FinalProjectDB; -- Specify the database to use
SELECT
    Investor.FirstNm AS InvestorName,
    Investor.LastNm AS InvestorLastName,
    Broker.BrokerNm AS BrokerName
FROM
    FinalProjectDB.InvestorBroker
INNER JOIN
    FinalProjectDB.Investor ON InvestorBroker.InvestorID = Investor.InvestorID
INNER JOIN
    FinalProjectDB.Broker ON InvestorBroker.BrokerID = Broker.BrokerID
ORDER BY
    Investor.InvestorID, Broker.BrokerID;
```

InvestorName	InvestorLastName	BrokerName
Alice	Johnson	XYZ Securities
Alice	Johnson	ABC Investments
Ali	Thompson	STU Financial
Emily	White	MNO Financial Services
Emily	White	JKL Brokerage
James	Smith	ABC Investments
James	Smith	MNO Traders
Bella	Martinez	DEF Capital
Alex	Brown	GHI Investments
Alex	Brown	PQR Brokers

3. Write an SQL by including two or more tables and using the LEFT OUTER JOIN. Show the results and sort the results by key field(s). Interpret the results compared to what an INNER JOIN does.



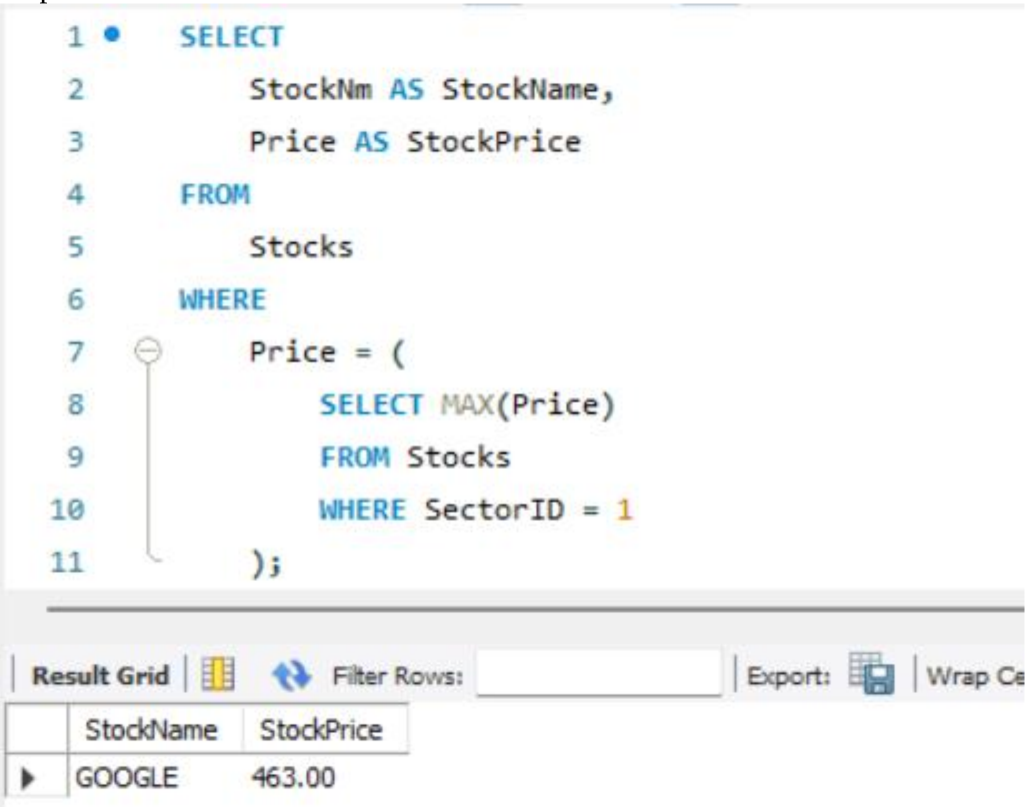
```sql
SELECT
    Investor.InvestorID,
    Investor.FirstNm AS InvestorName,
    Investor.LastNm AS InvestorLastName,
    Broker.BrokerID,
    Broker.BrokerNm AS BrokerName
FROM
    Investor
LEFT OUTER JOIN
    InvestorBroker ON Investor.InvestorID = InvestorBroker.InvestorID
LEFT OUTER JOIN
    Broker ON InvestorBroker.BrokerID = Broker.BrokerID
ORDER BY
    Investor.InvestorID, Broker.BrokerID;
```

InvestorID	InvestorName	InvestorLastName	BrokerID	BrokerName
11	Alice	Johnson	1	XYZ Securities
11	Alice	Johnson	2	ABC Investments
22	Ali	Thompson	4	STU Financial
33	Emily	White	5	MNO Financial Services
33	Emily	White	6	JKL Brokerage
44	James	Smith	2	ABC Investments
44	James	Smith	3	MNO Traders
55	Bella	Martinez	7	DEF Capital
66	Alex	Brown	8	GHI Investments
66	Alex	Brown	9	PQR Brokers

LEFT OUTER JOIN includes all rows from the left table (Investor), filling unmatched rows with NULL. All investors are included in the result, regardless of whether they have a broker assigned.

An INNER JOIN would only include rows where a match exists in all tables (Investor, InvestorBroker, and Broker).

4. Write a single-row subquery. Show the results and sort the results by key field(s). Interpret the output.

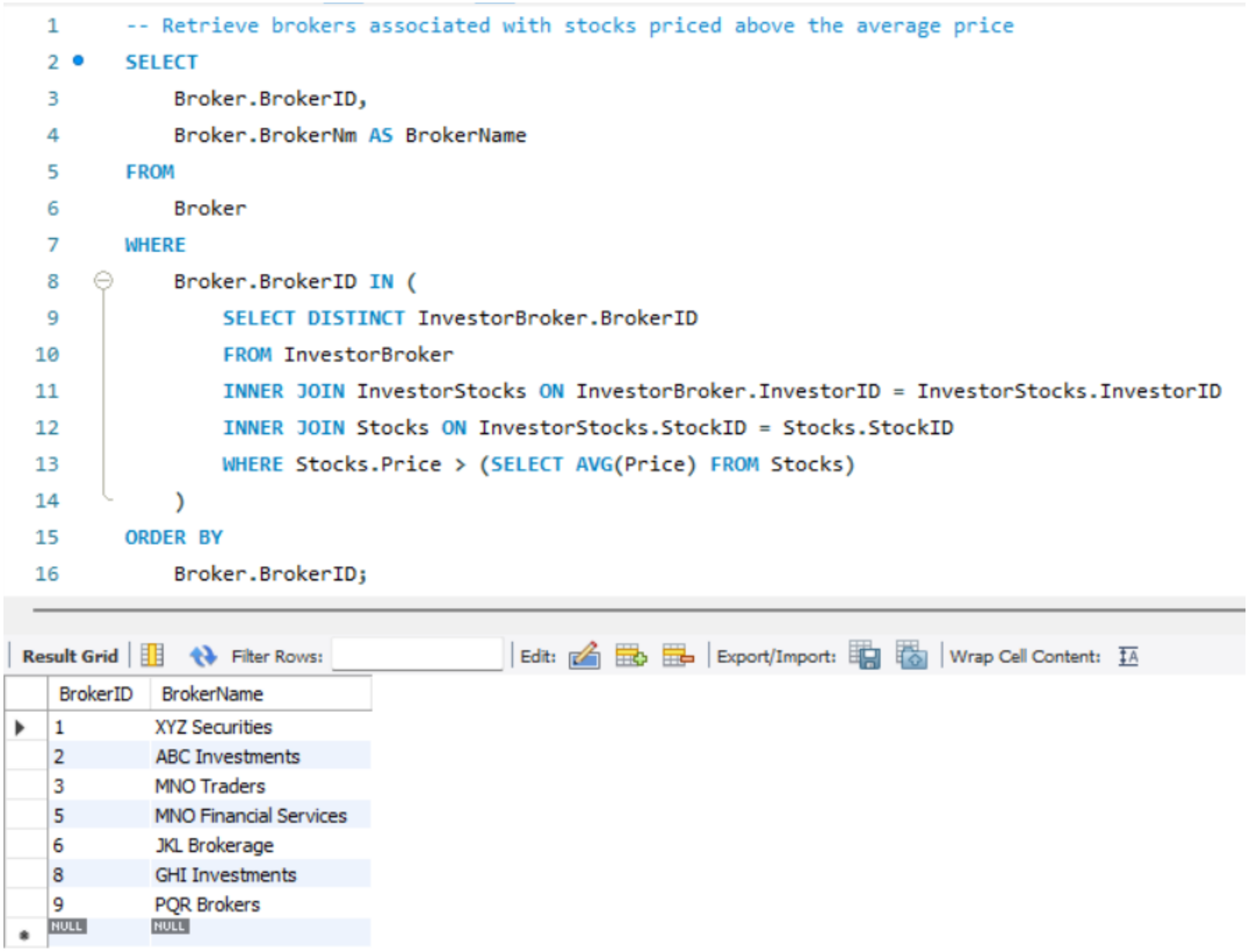


```
SELECT
    StockNm AS StockName,
    Price AS StockPrice
FROM
    Stocks
WHERE
    Price = (
        SELECT MAX(Price)
        FROM Stocks
        WHERE SectorID = 1
    );
```

StockName	StockPrice
GOOGLE	463.00

Google is the most expensive stock, as determined by the single-row subquery logic.

5. Write a multiple-row subquery. Show the results and sort the results by key field(s). Interpret the output.

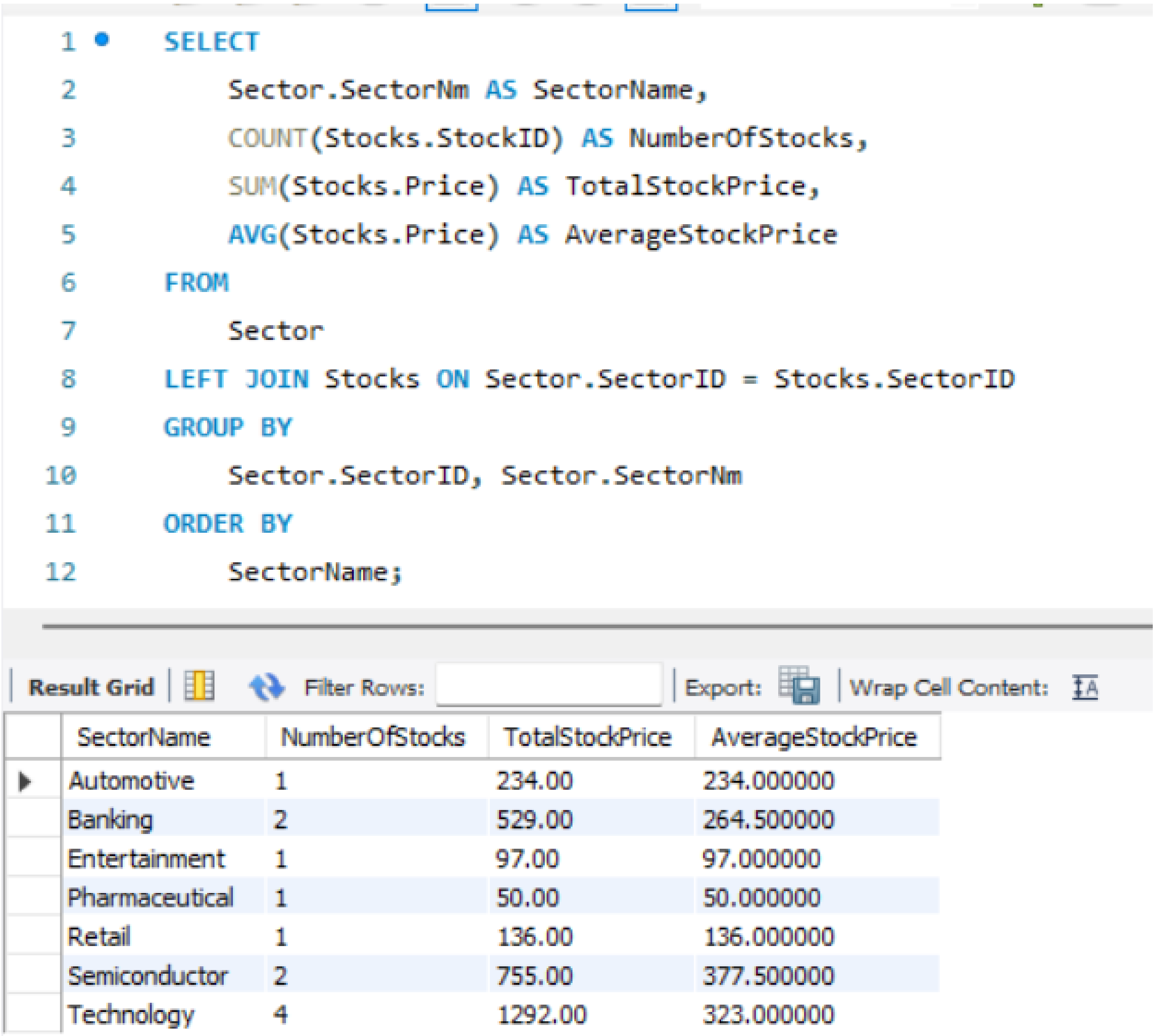


```sql
-- Retrieve brokers associated with stocks priced above the average price
SELECT
    Broker.BrokerID,
    Broker.BrokerNm AS BrokerName
FROM
    Broker
WHERE
    Broker.BrokerID IN (
        SELECT DISTINCT InvestorBroker.BrokerID
        FROM InvestorBroker
        INNER JOIN InvestorStocks ON InvestorBroker.InvestorID = InvestorStocks.InvestorID
        INNER JOIN Stocks ON InvestorStocks.StockID = Stocks.StockID
        WHERE Stocks.Price > (SELECT AVG(Price) FROM Stocks)
    )
ORDER BY
    Broker.BrokerID;
```

BrokerID	BrokerName
1	XYZ Securities
2	ABC Investments
3	MNO Traders
5	MNO Financial Services
6	JKL Brokerage
8	GHI Investments
9	PQR Brokers
NULL	NULL

The query identifies brokers who are associated with stocks priced higher than the average price. It uses a multiple-row subquery to fetch the broker IDs based on the stock price filter.

6. Write an SQL to aggregate the results by using multiple columns in the SELECT clause. Interpret the output.

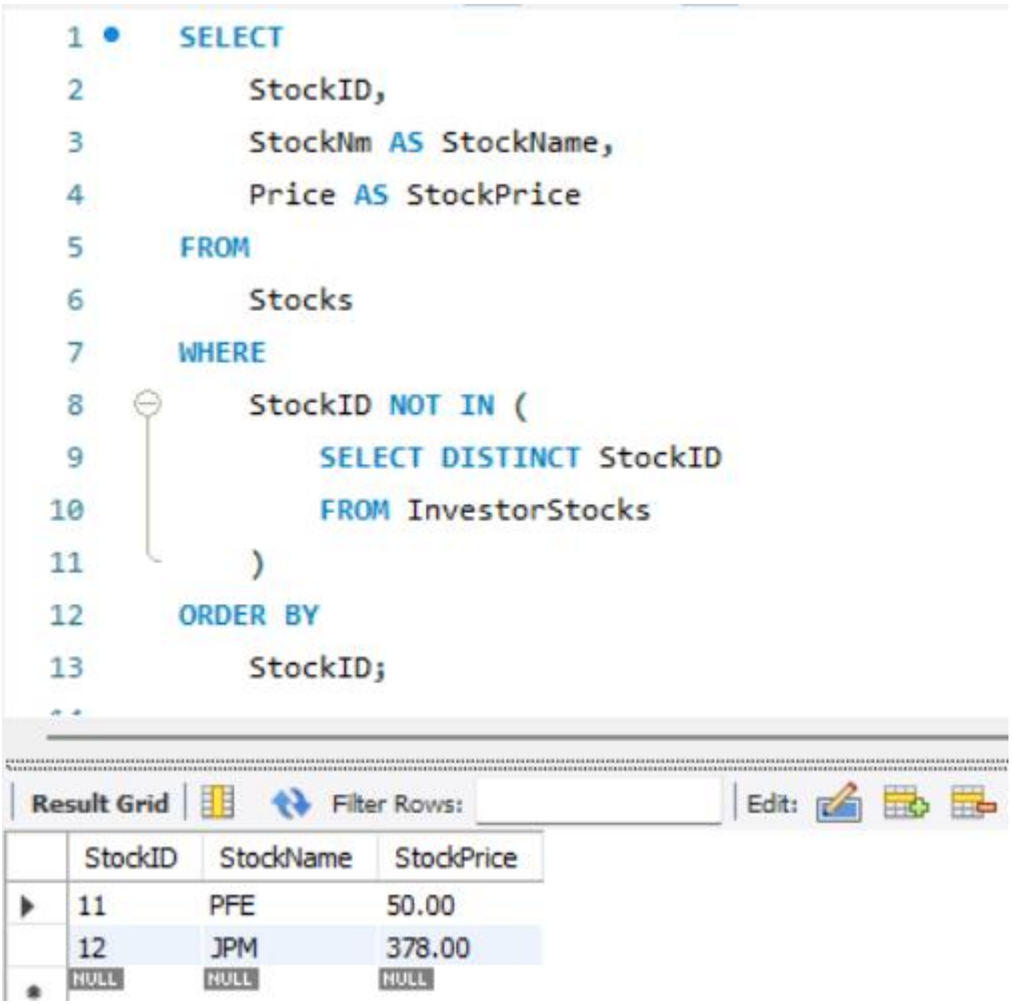


```sql
SELECT
    Sector.SectorNm AS SectorName,
    COUNT(Stocks.StockID) AS NumberOfStocks,
    SUM(Stocks.Price) AS TotalStockPrice,
    AVG(Stocks.Price) AS AverageStockPrice
FROM
    Sector
LEFT JOIN Stocks ON Sector.SectorID = Stocks.SectorID
GROUP BY
    Sector.SectorID, Sector.SectorNm
ORDER BY
    SectorName;
```

SectorName	NumberOfStocks	TotalStockPrice	AverageStockPrice
Automotive	1	234.00	234.000000
Banking	2	529.00	264.500000
Entertainment	1	97.00	97.000000
Pharmaceutical	1	50.00	50.000000
Retail	1	136.00	136.000000
Semiconductor	2	755.00	377.500000
Technology	4	1292.00	323.000000

This query helps to summarize the stock data by sector, providing valuable insights into the distribution and value of stocks across different sectors.

7. Write a subquery using the NOT IN operator. Show the results and sort the results by key field(s). Interpret the output.

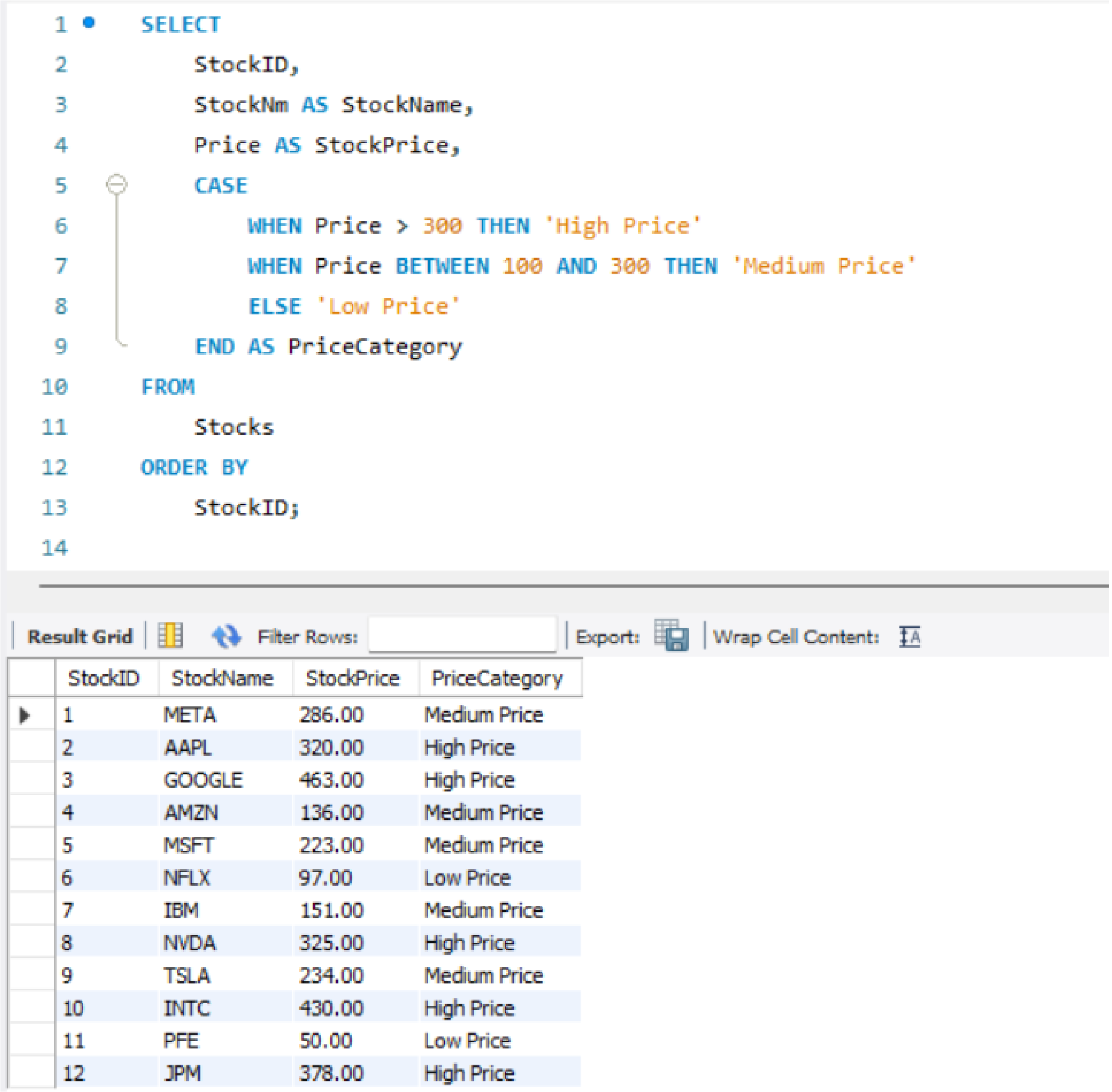


```sql
SELECT
    StockID,
    StockNm AS StockName,
    Price AS StockPrice
FROM
    Stocks
WHERE
    StockID NOT IN (
        SELECT DISTINCT StockID
        FROM InvestorStocks
    )
ORDER BY
    StockID;
```

StockID	StockName	StockPrice
11	PFE	50.00
12	JPM	378.00
NULL	NULL	NULL

The query identifies stocks that are not associated with any investor in the InvestorStocks table. The NOT IN operator is useful for excluding rows based on a list of values retrieved from a subquery.

8. Write a query using a CASE statement. Show the results and sort the results by key field(s). Interpret the output.



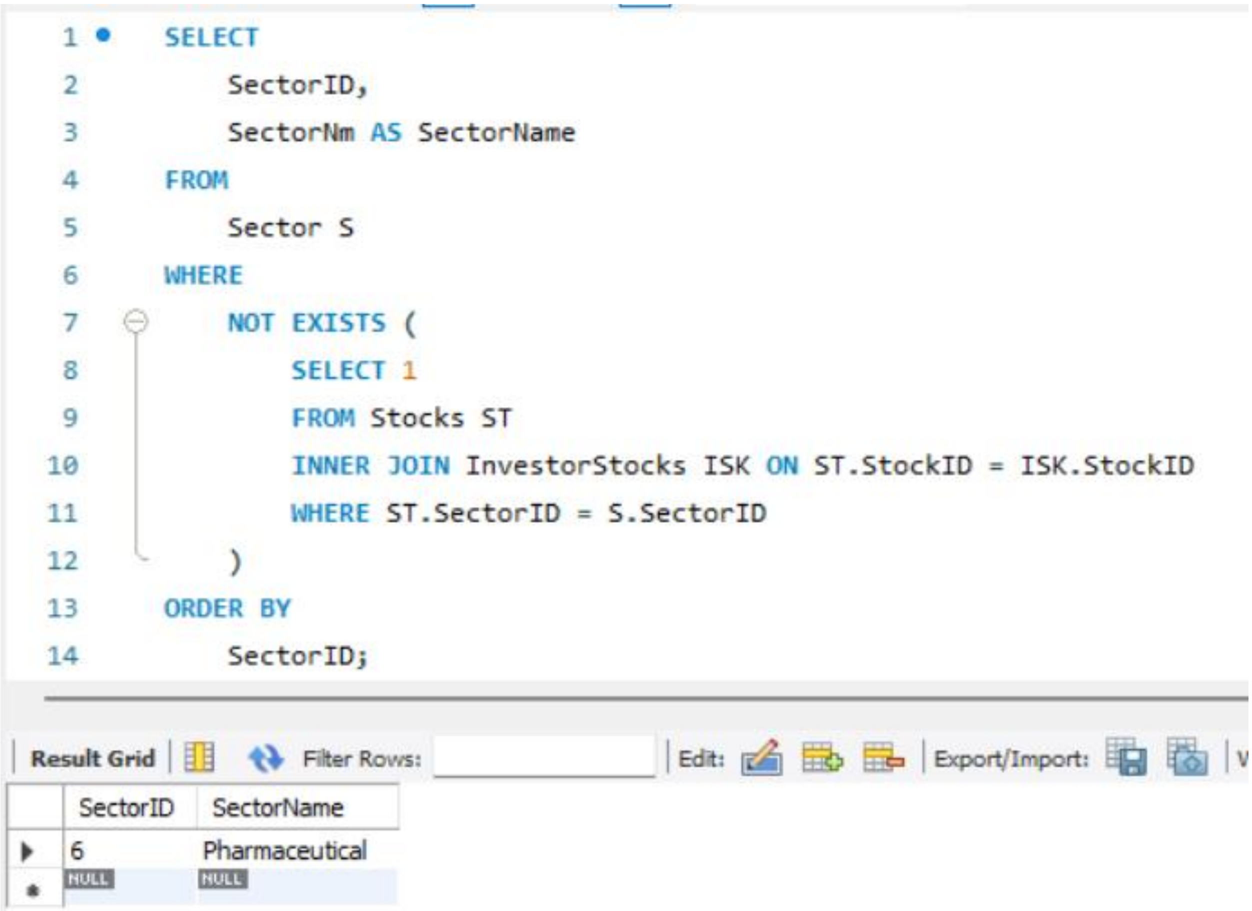
```sql
SELECT
    StockID,
    StockNm AS StockName,
    Price AS StockPrice,
    CASE
        WHEN Price > 300 THEN 'High Price'
        WHEN Price BETWEEN 100 AND 300 THEN 'Medium Price'
        ELSE 'Low Price'
    END AS PriceCategory
FROM
    Stocks
ORDER BY
    StockID;
```

StockID	StockName	StockPrice	PriceCategory
1	META	286.00	Medium Price
2	AAPL	320.00	High Price
3	GOOGLE	463.00	High Price
4	AMZN	136.00	Medium Price
5	MSFT	223.00	Medium Price
6	NFLX	97.00	Low Price
7	IBM	151.00	Medium Price
8	NVDA	325.00	High Price
9	TSLA	234.00	Medium Price
10	INTC	430.00	High Price
11	PFE	50.00	Low Price
12	JPM	378.00	High Price

Categories Created:

- Stocks are categorized into three groups:
    - **High Price:** Stocks priced above $300.
    - **Medium Price:** Stocks priced between $100 and $300.
    - **Low Price:** Stocks priced below $100.
- Helps identify premium stocks (e.g., GOOGLE and AAPL) versus affordable ones (NFLX).
- Assists in decision-making for stock tier-based analysis.

9. Write a query using the NOT EXISTS operator. Show the results and sort the results by key field(s). Interpret the output.

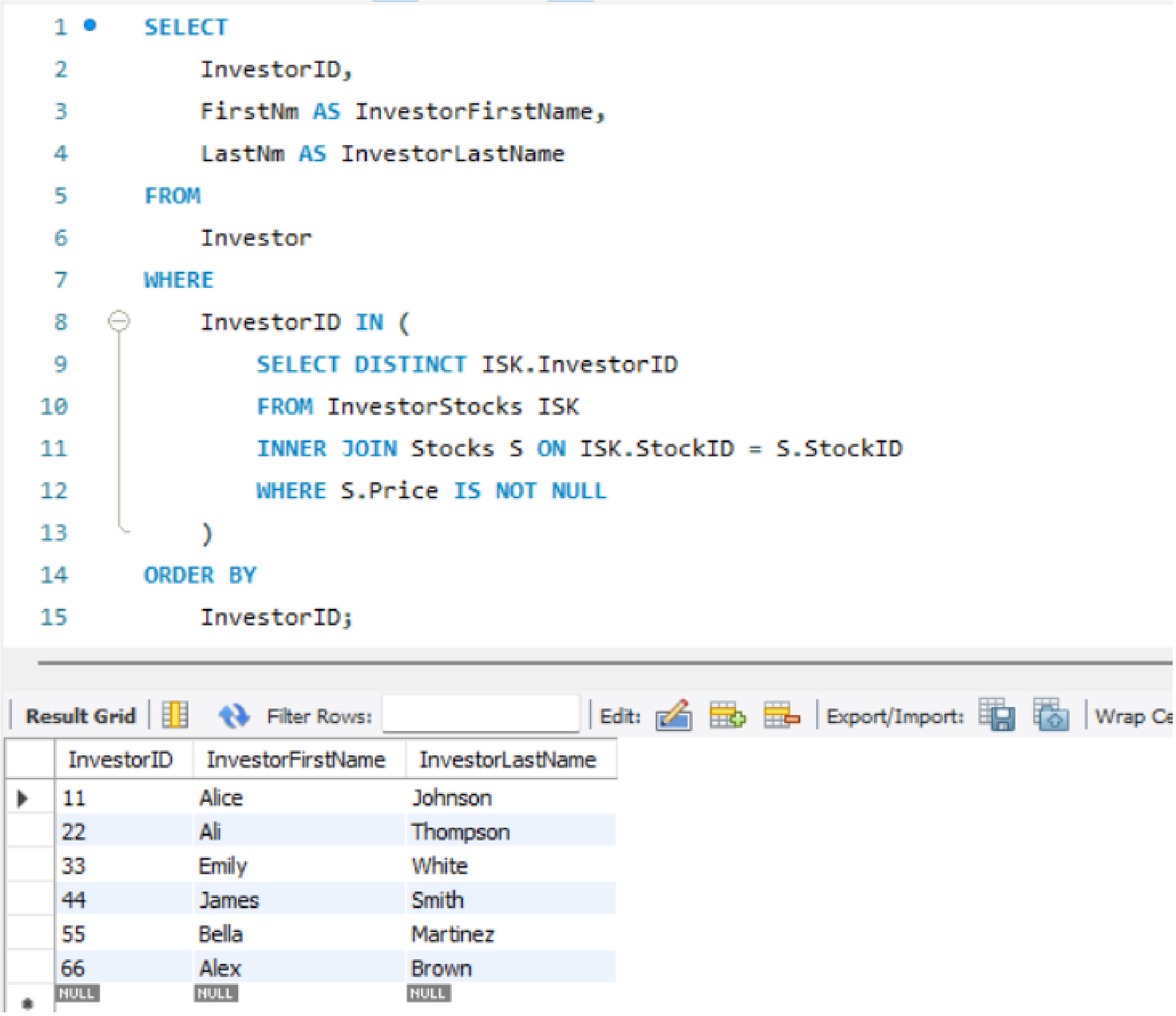


```sql
SELECT
    SectorID,
    SectorNm AS SectorName
FROM
    Sector S
WHERE
    NOT EXISTS (
        SELECT 1
        FROM Stocks ST
        INNER JOIN InvestorStocks ISK ON ST.StockID = ISK.StockID
        WHERE ST.SectorID = S.SectorID
    )
ORDER BY
    SectorID;
```

SectorID	SectorName
6	Pharmaceutical
NULL	NULL

The query identifies sectors where **no stocks are owned by any investors**.
Sector Pharmaceutical has stocks (e.g., PFE), but no investor owns them.

10. Write a subquery using the NOT NULL operator in the inner query. Show the results and sort the results by key field(s). Interpret the output.



```sql
SELECT
    InvestorID,
    FirstNm AS InvestorFirstName,
    LastNm AS InvestorLastName
FROM
    Investor
WHERE
    InvestorID IN (
        SELECT DISTINCT ISK.InvestorID
        FROM InvestorStocks ISK
        INNER JOIN Stocks S ON ISK.StockID = S.StockID
        WHERE S.Price IS NOT NULL
    )
ORDER BY
    InvestorID;
```

InvestorID	InvestorFirstName	InvestorLastName
11	Alice	Johnson
22	Ali	Thompson
33	Emily	White
44	James	Smith
55	Bella	Martinez
66	Alex	Brown
NULL	NULL	NULL

This query lists all investors who own stocks with valid prices. The NOT NULL condition ensures that only stocks with valid prices are considered.

Summary

This report is a detailed case study on stock management database system with the focus on advanced use of SQL for selecting and filtering the data across the tables. The work included writing queries that would provide relevant information like unowned stocks, classification of stock prices, and sectors or brokers with no business. SQL features applied include subquery, JOIN, aggregation and conditional logic using CASE statements to enhance data selection and interpretation. These queries were intended to bring forward key information patterns, notably investors, stock sector correlations and stock price bands, while ensuring clarity through ordered outputs.

Further ramping up the volume of information presented in the report, it described the versatility of subqueries and operators such as NOT IN, NOT EXISTS and IS NOT NULL for different business situations. For example, the subqueries were employed to find out the sectors that do not contain the investor-owned stocks and those investors who own stocks with valid price figure that can be inserted into the SQL algorithm to solve real-life relational database problems. It mirrors a systematic method of database querying that pays strong emphasis on precision, relevance, and conclusions making the program tremendously strong and ultra-stable.