SC2207 Lab5 Report

Lab index	SCMB	
Group number	3	
Team Members	Guo Yichen	U2320626C
	Mahi Pandey	U2321382F
	Mehta Rishika	U2323133H
	Zhao Qixian	U2321752L

Tutorial Group: SCMB

Group Number: #3

Lab Professor: Wei Dong

Teaching Assistant: Hu Yihua

College of Computing and Data Science

Nanyang Technological University

2024/2025 Semester 2

APPENDIX C: INDIVIDUAL CONTRIBUTION FORM

Full Name	Individual Contribution to Lab 1 Submission	Percentage of Contribution	Signature
Guo Yichen	Entity refinements and ERD drawing	25	
Mahi Pandey	Created Entities and Attributes in the ERD	25	
Mehta Rishika	Entity attribute, and ERD plotting	25	
Zhao Qixian	Entity attribute tables and ERD plotting	25	

Full Name	Individual Contribution to Lab 3 Submission	Percentage of Contribution	Signature
Guo Yichen	Refinements of relation schema	25	
Mahi Pandey	Cross check relational schema tables	25	
Mehta Rishika	Making the relational Schema	25	
Zhao Qixian	Find potential 3NF violations	25	

Full Name	Individual Contribution to Lab 5 Submission	Percentage of Contribution	Signature
Guo Yichen	Sql refine and report writing,query video	25	
Mahi Pandey	Populating the Tables & Report	25	
Mehta Rishika	SQL table creation and initialisation, report	25	

Zhao Qixian	SQL Queries and report writing, query video	25	

APPENDIX D: USE OF AI TOOL(S) IN LAB WORK

Each team member should indicate either A or B:

A. I affirm that my contribution(s) to the lab work is my own, produced without help from any AI tool(s).

B. I affirm that my contribution(s) to the lab work has been produced with the use of AI tool(s).

Team member	Signature	Date	A or B
Guo Yichen		4/2/2025	A
Mahi Pandey		4/2/2025	A
Mehta Rishika		4/2/2025	A
Zhao Qixian		4/2/2025	A

By signing this form, you declare that the above affirmation made is true and that you have read and understood NTU's policy on the use of AI tools.

If any team member answered B, the team member(s) must indicate and replicate the table below for every instance that AI tool(s) is used:

Name of AI tool	< For example, ChatGPT >
Input prompt	< Insert the question that you asked ChatGPT >
Date generated	< Insert the date when the response was generated >
Output generated	< Insert the response verbatim from ChatGPT >
Output screenshots	< Attach or reference screenshots of the response if applicable >
Impact on submission	< Briefly explain which part of your submitted work was ChatGPT's response applied >

SQL DDL Commands for Table Creation

```sql
-- Switch to the correct database
USE TotalWealthDB;
GO

-- Drop tables if they exist (in reverse order of creation to avoid
foreign key conflicts)
DROP TABLE IF EXISTS TransactionFees;
DROP TABLE IF EXISTS Transaction1;
DROP TABLE IF EXISTS UNREALIZED_GAIN_LOSS;
DROP TABLE IF EXISTS INVESTED_VALUE;
DROP TABLE IF EXISTS FUND_IN_PORTFOLIO;
DROP TABLE IF EXISTS BOND_IN_PORTFOLIO;
DROP TABLE IF EXISTS STOCK_IN_PORTFOLIO;
DROP TABLE IF EXISTS PortfolioFeeStructure;
DROP TABLE IF EXISTS Portfolio1;
DROP TABLE IF EXISTS FUND;
DROP TABLE IF EXISTS BOND;
DROP TABLE IF EXISTS STOCK;
DROP TABLE IF EXISTS ASSET;
DROP TABLE IF EXISTS FINANCIAL_GOAL;
DROP TABLE IF EXISTS RISK_TOLERANCE;
DROP TABLE IF EXISTS INVESTOR;

-- Create INVESTOR Table
CREATE TABLE INVESTOR (
    Phone VARCHAR(15) PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Gender CHAR(1),
    DoB DATE,
    AnnualIncome DECIMAL(15, 2),
    Company VARCHAR(100)
);

-- Create RISK_TOLERANCE Table
CREATE TABLE RISK_TOLERANCE (
    Phone VARCHAR(15),
    RiskLevel VARCHAR(20),
    Q1A VARCHAR(255),
    Q2A VARCHAR(255),
    Q3A VARCHAR(255),
    Q4A VARCHAR(255),
    Q5A VARCHAR(255),
    PRIMARY KEY (Phone, RiskLevel),
    FOREIGN KEY (Phone) REFERENCES INVESTOR(Phone)
);

-- Create FINANCIAL_GOAL Table
CREATE TABLE FINANCIAL_GOAL (
    Goal VARCHAR(100),
```

```sql
    Phone VARCHAR(15),
    Amount DECIMAL(15, 2),
    Timeline INT,
    PRIMARY KEY (Goal, Phone),
    FOREIGN KEY (Phone) REFERENCES INVESTOR(Phone)
);

-- Create ASSET Table
CREATE TABLE ASSET (
    AssetID VARCHAR(20) PRIMARY KEY,
    Name VARCHAR(100),
    Price DECIMAL(15, 2)
);

-- Create STOCK Table
CREATE TABLE STOCK (
    AssetID VARCHAR(20) PRIMARY KEY,
    P_ERatio DECIMAL(10, 2),
    EPS DECIMAL(10, 2),
    EBITDA DECIMAL(15, 2),
    FOREIGN KEY (AssetID) REFERENCES ASSET(AssetID)
);

-- Create BOND Table
CREATE TABLE BOND (
    AssetID VARCHAR(20) PRIMARY KEY,
    InterestRate DECIMAL(5, 2),
    MaturityDate DATE,
    FOREIGN KEY (AssetID) REFERENCES ASSET(AssetID)
);

-- Create FUND Table
CREATE TABLE FUND (
    AssetID VARCHAR(20) PRIMARY KEY,
    ExpenseRatio DECIMAL(5, 2),
    DividendYield DECIMAL(5, 2),
    FOREIGN KEY (AssetID) REFERENCES ASSET(AssetID)
);

-- Create PORTFOLIO Table (Decomposed into Portfolio1 and
PortfolioFeeStructure)
CREATE TABLE Portfolio1 (
    Phone VARCHAR(15),
    PID VARCHAR(20),
    MarketValue DECIMAL(15, 2),
    InceptionDate DATE,
    AnnualizedReturn DECIMAL(5, 2),
    PRIMARY KEY (PID, Phone), -- Composite primary key (order
corrected)
    FOREIGN KEY (Phone) REFERENCES INVESTOR(Phone)
);

CREATE TABLE PortfolioFeeStructure (
```

```sql
    MarketValue DECIMAL(15, 2) PRIMARY KEY,
    Fee DECIMAL(5, 2)
);

-- Create STOCK_IN_PORTFOLIO Table
CREATE TABLE STOCK_IN_PORTFOLIO (
    StockID VARCHAR(20) PRIMARY KEY,
    PID VARCHAR(20),
    Phone VARCHAR(15),
    StartDate DATE,
    AllocationRatio DECIMAL(5, 2),
    PostTradeCO VARCHAR(100),
    FOREIGN KEY (StockID) REFERENCES STOCK(AssetID),
    FOREIGN KEY (PID, Phone) REFERENCES Portfolio1(PID, Phone) --
References Portfolio1
);

-- Create BOND_IN_PORTFOLIO Table
CREATE TABLE BOND_IN_PORTFOLIO (
    BondID VARCHAR(20) PRIMARY KEY,
    PID VARCHAR(20),
    Phone VARCHAR(15),
    StartDate DATE,
    AllocationRatio DECIMAL(5, 2),
    PostTradeCO VARCHAR(100),
    FOREIGN KEY (BondID) REFERENCES BOND(AssetID),
    FOREIGN KEY (PID, Phone) REFERENCES Portfolio1(PID, Phone) --
References Portfolio1
);

-- Create FUND_IN_PORTFOLIO Table
CREATE TABLE FUND_IN_PORTFOLIO (
    FundID VARCHAR(20) PRIMARY KEY,
    PID VARCHAR(20),
    Phone VARCHAR(15),
    StartDate DATE,
    AllocationRatio DECIMAL(5, 2),
    PostTradeCO VARCHAR(100),
    FOREIGN KEY (FundID) REFERENCES FUND(AssetID),
    FOREIGN KEY (PID, Phone) REFERENCES Portfolio1(PID, Phone) --
References Portfolio1
);

-- Create INVESTED_VALUE Table
CREATE TABLE INVESTED_VALUE (
    Phone VARCHAR(15),
    PID VARCHAR(20),
    Date DATE,
    Amount DECIMAL(15, 2),
    PRIMARY KEY (Phone, PID, Date),
    FOREIGN KEY (PID, Phone) REFERENCES Portfolio1(PID, Phone) --
References Portfolio1
);
```

```sql
-- Create UNREALIZED_GAIN_LOSS Table
CREATE TABLE UNREALIZED_GAIN_LOSS (
    Phone VARCHAR(15),
    PID VARCHAR(20),
    Date DATE,
    Amount DECIMAL(15, 2),
    PRIMARY KEY (Phone, PID, Date),
    FOREIGN KEY (PID, Phone) REFERENCES Portfolio1(PID, Phone) --
References Portfolio1
);

-- Create TRANSACTION Table (Decomposed into Transaction1 and
TransactionFees)
CREATE TABLE Transaction1 (
    ID VARCHAR(20),
    Date DATE,
    PID VARCHAR(20),
    Phone VARCHAR(15),
    Type VARCHAR(50),
    PRIMARY KEY (ID, Date),
    FOREIGN KEY (PID, Phone) REFERENCES Portfolio1(PID, Phone) --
References Portfolio1
);

CREATE TABLE TransactionFees (
    Type VARCHAR(50) PRIMARY KEY,
    Fee DECIMAL(5, 2)
);
```

Table Records

INVESTOR

	Phone	Name	Gender	DoB	AnnualIncome	Company
1	92000000	Aisha Lim	M	2002-05-16	104997.03	Standard Chartered
2	92000001	Chloe Goh	M	1982-01-04	135843.48	OCBC
3	92000002	Benjamin Teo	F	2008-10-18	65335.35	HSBC
4	92000003	Chloe Lim	M	1983-10-01	89112.39	Revolut
5	92000004	Grace Wong	M	1986-01-22	97408.07	Maybank
6	92000005	Jayden Goh	M	1986-02-28	119617.71	DBS
7	92000006	Grace Ng	F	1991-05-16	134361.78	OCBC
8	92000007	Benjamin Chong	M	2000-02-16	86763.07	HSBC
9	92000008	Jayden Teo	F	1994-03-31	135873.25	HSBC
10	92000009	Chloe Goh	F	2001-06-05	52077.45	UOB

RISK_TOLERANCE

	Phone	RiskLevel	Q1A	Q2A	Q3A	Q4A	Q5A
1	92000000	Conservative	D	C	C	C	A
2	92000001	Aggressive	A	A	C	C	A
3	92000002	Conservative	B	B	A	B	A
4	92000003	Moderate	D	B	C	B	C
5	92000004	Aggressive	C	D	A	A	A
6	92000005	Conservative	A	B	A	A	D
7	92000006	Conservative	B	A	D	B	C
8	92000007	Aggressive	C	C	B	B	B
9	92000008	Aggressive	C	A	D	B	A
10	92000009	Aggressive	C	C	A	D	A

FINANCIAL_GOAL

	Goal	Phone	Amount	Timeline
1	Buy Property	92000000	800000.00	25
2	Buy Property	92000010	1000000.00	15
3	Buy Property	92000014	200000.00	25
4	Buy Property	92000045	500000.00	30
5	Buy Property	92000048	800000.00	10
6	Children Education	92000005	100000.00	5
7	Children Education	92000012	1000000.00	10
8	Children Education	92000024	100000.00	20
9	Children Education	92000029	1000000.00	10
1…	Children Education	92000038	1000000.00	25

ASSET

	AssetID	Name	Price
1	BND001	SG Bond 2035	100.00
2	BND002	SG Bond 2035	100.00
3	BND003	US Treasury 2030	98.50
4	BND004	US Treasury 2030	98.50
5	BND005	Green Bond 2040	102.00
6	BND006	Green Bond 2040	102.00
7	BND007	Singapore Savings Bond 2028	100.00
8	BND008	Singapore Savings Bond 2028	100.00
9	FND001	Vanguard S&P 500	55.25
10	FND002	Vanguard S&P 500	55.25

	AssetID	Name	Price
14	FND006	Global Tech Fund	65.75
15	FND007	Emerging Markets ETF	50.00
16	FND008	Emerging Markets ETF	50.00
17	STK001	Tesla	900.00
18	STK002	Tesla	900.00
19	STK003	Tesla	900.00
20	STK004	Tesla	900.00
21	STK005	Apple	180.00
22	STK006	Apple	180.00
23	STK007	Apple	180.00
24	STK008	Nvidia	700.00

STOCK

	AssetID	P_ERatio	EPS	EBITDA
1	STK001	56.18	16.02	41288549.36
2	STK002	78.47	11.47	72052369.78
3	STK003	61.94	14.53	28662995.63
4	STK004	104.41	8.62	94911601.17
5	STK005	9.35	19.26	44765259.94
6	STK006	12.50	14.40	64803893.90
7	STK007	20.57	8.75	96589211.18
8	STK008	36.63	19.11	15682347.17
9	STK009	59.37	11.79	53280868.77
10	STK010	29.46	11.88	29080530.80

BOND

	AssetID	InterestRate	MaturityDate
1	BND001	1.20	2035-12-31
2	BND002	1.20	2035-12-31
3	BND003	4.99	2030-12-31
4	BND004	4.99	2030-12-31
5	BND005	1.31	2040-12-31
6	BND006	1.31	2040-12-31
7	BND007	4.62	2028-12-31
8	BND008	4.62	2028-12-31

FUND

	AssetID	ExpenseRatio	DividendYield
1	FND001	0.76	3.78
2	FND002	0.56	2.45
3	FND003	0.73	1.03
4	FND004	0.61	3.05
5	FND005	1.26	4.62
6	FND006	0.47	3.66
7	FND007	1.38	3.72
8	FND008	1.10	4.43

Portfolio1

	Phone	PID	MarketValue	InceptionDate	AnnualizedReturn
1	92000000	P2001	643833.12	2023-12-06	5.10
2	92000001	P2002	493761.63	2024-01-16	-3.42
3	92000002	P2003	419454.39	2022-02-19	-3.38
4	92000003	P2004	266349.87	2023-09-14	-4.29
5	92000004	P2005	772707.47	2022-07-01	-4.51
6	92000005	P2006	718072.94	2023-05-08	6.50
7	92000006	P2007	659499.06	2024-02-08	10.63
8	92000007	P2008	578932.01	2023-10-26	2.74
9	92000008	P2009	603143.43	2023-07-28	3.79
10	92000009	P2010	190675.44	2023-11-05	13.09

PortfolioFeeStructure

	MarketValue	Fee
1	103758.02	0.88
2	136157.83	0.88
3	146490.62	0.88
4	190675.44	0.88
5	263171.20	0.88
6	266349.87	0.88
7	275234.84	0.88
8	305273.05	0.88
9	310754.76	0.88
10	315914.36	0.88

STOCK_IN_PORTFOLIO

	StockID	PID	Phone	StartDate	AllocationRatio	PostTradeCO
1	STK001	P2012	92000011	2023-03-10	0.24	Clearstream
2	STK002	P2014	92000013	2020-09-19	0.15	Clearstream
3	STK003	P2016	92000015	2023-08-06	0.34	Interactive Broker
4	STK004	P2016	92000015	2021-05-28	0.46	Saxo
5	STK005	P2007	92000006	2024-01-30	0.21	Clearstream
6	STK006	P2007	92000006	2022-06-05	0.32	Saxo
7	STK007	P2001	92000000	2021-04-25	0.42	Clearstream
8	STK008	P2013	92000012	2022-04-13	0.27	Saxo
9	STK009	P2003	92000002	2022-11-03	0.45	Clearstream
10	STK010	P2002	92000001	2020-07-08	0.50	Clearstream

BOND_IN_PORTFOLIO

	BondID	PID	Phone	StartDate	AllocationRatio	PostTradeCO
1	BND001	P2018	92000017	2022-05-19	0.34	Clearstream
2	BND002	P2014	92000013	2024-06-13	0.46	Interactive Broker
3	BND003	P2012	92000011	2022-11-06	0.22	Interactive Broker
4	BND004	P2012	92000011	2024-03-24	0.11	Interactive Broker
5	BND005	P2001	92000000	2020-08-21	0.13	Interactive Broker
6	BND006	P2016	92000015	2024-06-28	0.22	Interactive Broker
7	BND007	P2009	92000008	2021-04-07	0.17	Interactive Broker
8	BND008	P2018	92000017	2022-12-12	0.24	Saxo

FUND_IN_PORTFOLIO

	FundID	PID	Phone	StartDate	AllocationRatio	PostTradeCO
1	FND001	P2011	92000010	2020-04-05	0.47	Saxo
2	FND002	P2008	92000007	2022-07-09	0.41	Interactive Broker
3	FND003	P2020	92000019	2023-11-18	0.48	Interactive Broker
4	FND004	P2011	92000010	2023-04-27	0.43	Clearstream
5	FND005	P2018	92000017	2022-04-21	0.20	Interactive Broker
6	FND006	P2016	92000015	2023-01-07	0.14	Saxo
7	FND007	P2010	92000009	2022-09-09	0.26	Interactive Broker
8	FND008	P2018	92000017	2022-11-03	0.15	Clearstream

INVESTED_VALUE

	Phone	PID	Date	Amount
1	92000000	P2001	2024-01-01	643833.12
2	92000000	P2001	2024-03-31	688401.90
3	92000001	P2002	2024-01-01	493761.63
4	92000001	P2002	2024-03-31	496235.20
5	92000002	P2003	2024-01-01	419454.39
6	92000002	P2003	2024-03-31	361863.01
7	92000003	P2004	2024-01-01	266349.87
8	92000003	P2004	2024-03-31	241038.77
9	92000004	P2005	2024-01-01	772707.47
10	92000004	P2005	2024-03-31	914045.96

UNREALIZED_GAIN_LOSS

	Phone	PID	Date	Amount
1	92000000	P2001	2024-01-01	47445.90
2	92000000	P2001	2024-03-31	55766.53
3	92000001	P2002	2024-01-01	31665.15
4	92000001	P2002	2024-03-31	29493.45
5	92000002	P2003	2024-01-01	-26358.36
6	92000002	P2003	2024-03-31	-25403.26
7	92000003	P2004	2024-01-01	46939.54
8	92000003	P2004	2024-03-31	49507.30
9	92000004	P2005	2024-01-01	45046.10
10	92000004	P2005	2024-03-31	37229.24

Transaction1

	ID	Date	PID	Phone	Type
1	T100	2024-04-18	P2001	92000000	SELL
2	T101	2024-02-23	P2002	92000001	BUY
3	T102	2024-01-03	P2003	92000002	SELL
4	T103	2024-01-03	P2004	92000003	REBALANCE
5	T104	2024-02-29	P2005	92000004	BUY
6	T105	2024-03-05	P2006	92000005	BUY
7	T106	2024-02-12	P2007	92000006	BUY
8	T107	2024-03-05	P2008	92000007	BUY
9	T108	2024-02-24	P2009	92000008	REBALANCE
10	T109	2024-02-29	P2010	92000009	SELL

TransactionFees

	Type	Fee
1	BUY	0.50
2	REBALANCE	0.40
3	SELL	0.70
4	TOPUP	0.20

SQL Queries

Query 1

Find investors who are making on average a loss across all their portfolios in 2024.

Assumptions:

- The **UNREALIZED_GAIN_LOSS** table contains all relevant transactions for **2024**
- The Amount in **UNREALIZED_GAIN_LOSS** uses **negative values** to denote losses
- Investors' portfolios are evaluated only for **2024**. Losses from prior years or future projections are irrelevant to this analysis

Explanation:

- The query identifies investors who, on average, have incurred losses across all their portfolios in 2024. It retrieves their phone number, name, and the average unrealized gain/loss amount (AvgUnrealizedGainLoss).
- **Step 1:** The **SELECT** clause retrieves the investor's phone number (i.Phone), the investor's name (i.Name), and the average unrealized gain or loss (AVG(ul.Amount)) which is aliased as AvgUnrealizedGainLoss. The **FROM** clause specifies the INVESTOR table (i) as the main data source.

```sql
-- 1. Find investors who are making on average
--a loss across all their portfolios in 2024.
SELECT
    i.Phone,
    i.Name,
    AVG(ul.Amount) as AvgUnrealizedGainLoss
FROM
    INVESTOR i
```

- **Step 2:** The **JOIN** clauses link the Portfolio1 table (p) to INVESTOR based on the matching Phone column, and then the UNREALIZED_GAIN_LOSS table (ul) is joined to the Portfolio1 table using both PID and Phone as matching keys. This ensures that the data is accurately connected across all three tables.

```sql
JOIN
    Portfolio1 p ON i.Phone = p.Phone
JOIN
    UNREALIZED_GAIN_LOSS ul ON p.PID = ul.PID AND p.Phone = ul.Phone
```

- **Step 3:** The **WHERE** clause filters the records to include only those where the year of the Date column in UNREALIZED_GAIN_LOSS is 2024. The **GROUP BY** clause groups the data by the investor's phone number and name to calculate the average unrealized gain/loss for each investor.

```sql
WHERE
    YEAR(ul.Date) = 2024
GROUP BY
    i.Phone, i.Name
```

- **Step 4:** The **HAVING** clause filters the results to include only those investors with an average unrealized gain or loss less than 0 (i.e., a negative average unrealized gain/loss). Finally, the **ORDER BY** clause sorts the results by the calculated

AvgUnrealizedGainLoss in ascending order, with the most negative values appearing first.

```
HAVING
    AVG(ul.Amount) < 0
ORDER BY
    AvgUnrealizedGainLoss;
```

The final query identifies investors who, on average, have incurred losses across all their portfolios in 2024. It retrieves their phone number, name, and the average unrealized gain/loss (where losses are represented by negative values). The query specifically filters for losses in 2024, groups the data by investor, and includes only those with a negative average unrealized gain/loss. The results are then sorted in ascending order by the magnitude of the loss, showing the investors with the largest losses at the top.

Final Code:

```
SELECT
    i.Phone,
    i.Name,
    AVG(ul.Amount) as AvgUnrealizedGainLoss
FROM
    INVESTOR i
JOIN
    Portfolio1 p ON i.Phone = p.Phone
JOIN
    UNREALIZED_GAIN_LOSS ul ON p.PID = ul.PID AND p.Phone = ul.Phone
WHERE
    YEAR(ul.Date) = 2024
GROUP BY
    i.Phone, i.Name
HAVING
    AVG(ul.Amount) < 0
ORDER BY
    AvgUnrealizedGainLoss;
```

Output:

	Phone	Name	AvgUnrealizedGainLoss
1	92000028	Faiz Koh	-39845.610000
2	92000042	Isla Tan	-39705.840000
3	92000048	Aisha Chong	-36485.230000
4	92000044	Jayden Koh	-35877.815000
5	92000009	Chloe Goh	-34979.090000
6	92000011	Daniel Goh	-34738.775000
7	92000041	Chloe Goh	-34216.180000
8	92000037	Jayden Tan	-33964.260000
9	92000046	Grace Wong	-30227.985000
10	92000002	Mahi Pandey	-25880.810000
11	92000012	Isla Teo	-20284.570000
12	92000029	Aisha Lee	-18551.305000
13	92000008	Jayden Teo	-17704.385000
14	92000033	Benjamin Y…	-15762.900000
15	92000006	Grace Ng	-15144.725000
16	92000010	Hiro Koh	-13575.975000
17	92000040	Aisha Koh	-13287.195000
18	92000018	Isla Ng	-13037.645000
19	92000049	Hiro Wong	-11124.885000
20	92000027	Jayden Koh	-3118.565000

Query 2

Find investors who are seeing an annualized return of more than 10% from their portfolios in 2024.

Assumptions:

- **AnnualizedReturn** values in **Portfolio1** represent percentage returns (positive values denote gains).
- Investors' portfolios are evaluated **only for 2024**. Returns from prior years or future projections are irrelevant to this analysis.

Explanation:

- The query identifies investors with portfolios that achieved an annualized return **greater than 10% in 2024**. It retrieves their name, phone number, portfolio ID, and annualized return value.
- **Step 1:** The **SELECT** clause specifies the columns to retrieve: the investor's name (i.Name) aliased as InvestorName, the investor's phone number (i.Phone), the portfolio ID (p.PID) aliased as PortfolioID, and the portfolio's annualized return (p.AnnualizedReturn).

```sql
-- 2. Find investors who are seeing an annualized return of
-- more than 10% from their portfolios in 2024.
SELECT
    i.Name AS InvestorName,
    i.Phone,
    p.PID AS PortfolioID,
    p.AnnualizedReturn
```

- **Step 2:** The **FROM** clause specifies the INVESTOR table (i) as the main source of data. The **INNER JOIN** clause combines data from the Portfolio1 table (p) based on the matching Phone column in both tables. This ensures only records with a corresponding investor and portfolio are included.

```sql
FROM
    INVESTOR i
INNER JOIN
    Portfolio1 p ON i.Phone = p.Phone
```

- **Step 3:** The **WHERE** clause filters the results to include only portfolios with an annualized return greater than 10.0. Finally, the **ORDER BY** clause sorts the results by the annualized return in descending order, displaying the highest returns first.

```sql
WHERE
    p.AnnualizedReturn > 10.0
ORDER BY
    p.AnnualizedReturn DESC;
```

The final query identifies investors whose portfolios have achieved an annualized return greater than 10% in 2024. It retrieves the investor's name, phone number, portfolio ID, and the corresponding annualized return. The query filters portfolios with returns exceeding 10% and

sorts the results in descending order by annualized return, showcasing investors with the highest returns at the top. This provides insight into investors who have seen significant gains in 2024.

Final Code:

```
SELECT
    i.Name AS InvestorName,
    i.Phone,
    p.PID AS PortfolioID,
    p.AnnualizedReturn
FROM
    INVESTOR i
INNER JOIN
    Portfolio1 p ON i.Phone = p.Phone
WHERE
    p.AnnualizedReturn > 10.0
ORDER BY
    p.AnnualizedReturn DESC;
```

Output:

	InvestorName	Phone	PortfolioID	AnnualizedReturn
1	Hiro Wong	92000049	P2050	14.62
2	Jayden Koh	92000044	P2045	14.59
3	Aisha Chong	92000016	P2017	13.99
4	Daniel Lee	92000013	P2014	13.34
5	Chloe Goh	92000009	P2010	13.09
6	Chloe Lim	92000021	P2022	12.53
7	Aisha Tan	92000031	P2032	11.90
8	Hiro Teo	92000019	P2020	11.70
9	Aisha Koh	92000040	P2041	11.64
10	Daniel Teo	92000024	P2025	11.54
11	Daniel Goh	92000011	P2012	11.39
12	Aisha Lee	92000029	P2030	11.09
13	Grace Teo	92000047	P2048	10.95
14	Benjamin Ng	92000030	P2031	10.90
15	Isla Lee	92000023	P2024	10.90
16	Grace Ng	92000006	P2007	10.63
17	Jayden Tan	92000037	P2038	10.31

Query 3

Find the monthly average unrealized gain/loss of portfolios for each month in 2024.

Assumptions:

- The **UNREALIZED_GAIN_LOSS** table contains transactions **exclusively for 2024**.
- The **Amount** field uses **negative values to denote losses** and positive values for gains.
- The **Date** column is stored in a valid date format, allowing date-related functions (FORMAT, MONTH, DATENAME) to work correctly.

Explanation:

- The query calculates the **monthly average unrealized gain/loss** across all transactions in 2024. It retrieves the year-month, month number, month name, and average unrealized gain/loss amount (AverageUnrealizedGainLoss) for each month.
- **Step 1**: The **SELECT** clause specifies the columns to include in the results. It extracts the year and month from the Date column using **FORMAT**(Date, 'yyyy-MM') and aliases it as YearMonth, the month number with MONTH(Date) as MonthNumber, and the full month name with **DATENAME**(MONTH, Date) as MonthName. Additionally, AVG(Amount) calculates the average of the Amount column for each group, aliased as AverageUnrealizedGainLoss.

```sql
-- 3. Find the monthly average unrealized gain/loss of portfolios for each month in 2024.
SELECT
    FORMAT(Date, 'yyyy-MM') AS YearMonth,
    MONTH(Date) AS MonthNumber,
    DATENAME(MONTH, Date) AS MonthName,
    AVG(Amount) AS AverageUnrealizedGainLoss
```

- **Step 2:** The **FROM** clause specifies the source table, which is UNREALIZED_GAIN_LOSS, where the data is being queried from. The **WHERE clause** filters records to include only unrealized gain/loss transactions from the year **2024**.

```sql
FROM
    UNREALIZED_GAIN_LOSS
WHERE
    YEAR(Date) = 2024
```

- **Step 3:** The **GROUP BY** clause results by YearMonth (formatted as yyyy-MM), MonthNumber, and MonthName to aggregate data at the **monthly level** and compute the average gain/loss for each month.

```sql
GROUP BY
    FORMAT(Date, 'yyyy-MM'),
    MONTH(Date),
    DATENAME(MONTH, Date)
ORDER BY
    MonthNumber;
```

- **Step 4:** The **ORDER BY** clause sorts results by MonthNumber in **ascending order**, ensuring months are displayed chronologically (January to December).

The final query calculates the average unrealized gain/loss for each month in 2024 by extracting

data from the UNREALIZED_GAIN_LOSS table. It retrieves the year-month, month number, month name, and the average unrealized gain/loss for each month. The query filters records for the year 2024 and groups the results by the formatted year-month, month number, and month name to calculate the monthly average gain/loss. The results are then sorted in chronological order from January to December, providing insights into the monthly performance of portfolios throughout the year.

Final Code:

```
SELECT
    FORMAT(Date, 'yyyy-MM') AS YearMonth,
    MONTH(Date) AS MonthNumber,
    DATENAME(MONTH, Date) AS MonthName,
    AVG(Amount) AS AverageUnrealizedGainLoss
FROM
    UNREALIZED_GAIN_LOSS
WHERE
    YEAR(Date) = 2024
GROUP BY
    FORMAT(Date, 'yyyy-MM'),
    MONTH(Date),
    DATENAME(MONTH, Date)
ORDER BY
    MonthNumber;
```

Output:

	YearMonth	MonthNumber	MonthName	AverageUnrealizedGainLoss
1	2024-01	1	January	5663.964000
2	2024-03	3	March	5815.150000

Query 4

What is the top three most popular first financial goals for investors in 2024?

Assumptions:

- The **Goal** column is populated with valid, non-null values (e.g., "Retirement," "Wealth Preservation," "Education Funding").
- Investors may have **multiple goals**, but each row in the table corresponds to a single goal for a single investor.
- The data reflects **current and active goals** (no historical or inactive goals are included).

Explanation:

- The query identifies the **top 3 most common financial goals** among investors and counts how many investors are associated with each goal. It returns the name of the financial goal (e.g., "Retirement") and the total number of investors pursuing that goal.
- **Step 1**: The **TOP 3** clause limits the results to the **three most popular goals**, highlighting the highest-priority objectives among investors.

```sql
-- 4. What is the top three most popular first
-- financial goals for investors in 2024?
SELECT TOP 3
    Goal,
    COUNT(*) AS NumberOfInvestors
FROM
    FINANCIAL_GOAL
```

- **Step 2:** The **GROUP BY** clause groups results by the Goal column to count how many investors are associated with each goal.

```sql
GROUP BY
    Goal
ORDER BY
    NumberOfInvestors DESC;
```

- **Step 3:** The **ORDER BY** clause sorts results by NumberOfInvestors in **descending order**, prioritizing the most frequently occurring goals.

The final query identifies the top 3 most popular financial goals among investors in 2024. It retrieves the financial goal name (e.g., "Retirement," "Wealth Preservation") and the total number of investors associated with each goal. The query groups the data by the Goal column to count how many investors are pursuing each goal, then sorts the results in descending order by the number of investors. The TOP 3 clause limits the results to the three most popular goals, providing insight into the primary financial objectives of investors in 2024.

<u>Final Code:</u>

```sql
SELECT TOP 3
    Goal,
    COUNT(*) AS NumberOfInvestors
FROM
    FINANCIAL_GOAL
```

```sql
GROUP BY
    Goal
ORDER BY
    NumberOfInvestors DESC;
```

Output:

	Goal	NumberOfInvestors
1	Medical Emergency	11
2	Start Business	9
3	Children Education	7

Query 5

Find investors who consistently top up their investment at the beginning of every month (dollar-cost averaging) in 2024 for at least one of their portfolios.

Assumptions:

- "Consistently top up" is defined as performing at least one top-up within the first 5 days of a month.
- Only transactions in 2024 are considered.
- A “month with top-up” is counted only once per investor per portfolio.
- We are interested in investors who have done this in at least one month.

Explanation:

The query is structured in two main parts:

1. A **CTE (Common Table Expression)** to extract monthly top-up activity
2. A **main query** to count the number of months with such activity per investor

Step 1: Identify Monthly Top-Ups

```sql
WITH MonthlyTopups AS (
    SELECT
        t.Phone,
        t.PID,
        MONTH(t.Date) AS MonthNumber,
        COUNT(*) AS TopupCount
    FROM
        Transaction1 t
    WHERE
        t.Type = 'TOPUP'
        AND YEAR(t.Date) = 2024
        AND DAY(t.Date) <= 5  -- Assuming "beginning of month" means within first 5 days
    GROUP BY
        t.Phone,
        t.PID,
        MONTH(t.Date)
)
```

This CTE filters Transaction1:

- For transactions of type 'TOPUP'
- Within the first 5 days of each month in 2024
- And groups by Phone, PID, and month
- COUNT(*) gives how many top-ups occurred in that month for that portfolio

The result captures the months when early top-ups occurred for each investor and portfolio.

Step 2: Aggregate and Display Results

```sql
SELECT
    i.Name AS InvestorName,
    i.Phone,
    COUNT(DISTINCT m.MonthNumber) AS MonthsWithTopups
FROM
    INVESTOR i
JOIN
    MonthlyTopups m ON i.Phone = m.Phone
GROUP BY
    i.Name,
    i.Phone
HAVING
    COUNT(DISTINCT m.MonthNumber) > 0
ORDER BY
    MonthsWithTopups DESC;
```

This part:

- Joins the CTE with the INVESTOR table to fetch the investor's name
- Counts the distinct months with early top-ups
- Filters to include only those investors who had at least one month with a qualifying top-up
- Sorts them by the number of such months in descending order

Final Code:

```sql
WITH MonthlyTopups AS (
    SELECT
        t.Phone,
        t.PID,
        MONTH(t.Date) AS MonthNumber,
        COUNT(*) AS TopupCount
    FROM
        Transaction1 t
    WHERE
        t.Type = 'TOPUP'
        AND YEAR(t.Date) = 2024
        AND DAY(t.Date) <= 5  -- Assuming "beginning of month" means
within first 5 days
    GROUP BY
        t.Phone,
        t.PID,
        MONTH(t.Date)
)
SELECT
```

```sql
    i.Name AS InvestorName,
    i.Phone,
    COUNT(DISTINCT m.MonthNumber) AS MonthsWithTopups
FROM
    INVESTOR i
JOIN
    MonthlyTopups m ON i.Phone = m.Phone
GROUP BY
    i.Name,
    i.Phone
HAVING
    COUNT(DISTINCT m.MonthNumber) > 0
ORDER BY
    MonthsWithTopups DESC;
```

Final Output:

	InvestorName	Phone	MonthsWit…
1	Aisha Lee	92000029	1

Query 6

Find the most popular financial goals for investors working in the same company and whose age is between 30 to 40 years old.

Assumptions:

- Age is calculated based on the difference between current year and investor's date of birth.
- Only investors aged between 30 and 40 (inclusive) are considered.
- Each investor may have one or more financial goals in the FINANCIAL_GOAL table.
- We assume the goal is "popular" if it has the highest count per company — although this query shows all goals grouped by company, not just the top one per company.

Explanation:

This query uses two CTEs to filter eligible investors by age and then aggregates their financial goals by company.

Step 1: Compute Investor Age

```sql
WITH InvestorAge AS (
    SELECT
        i.Phone,
        i.Name,
        i.Company,
        i.DoB,
        DATEDIFF(YEAR, i.DoB, GETDATE()) AS Age
    FROM
        INVESTOR i
),
```

This CTE (InvestorAge) calculates each investor's age by subtracting their date of birth from the current date (GETDATE()), using DATEDIFF(YEAR, ...). It keeps track of their name, phone, and company for future use.

Step 2: Filter Investors Aged 30 to 40

```sql
AgeFiltered AS (
    SELECT
        ia.Phone,
        ia.Name,
        ia.Company,
        ia.Age
    FROM
        InvestorAge ia
    WHERE
        ia.Age BETWEEN 30 AND 40
)
```

This CTE (AgeFiltered) filters out only those investors whose computed age falls between 30 and 40 years old (inclusive).

Step 3: Aggregate Financial Goals by Company

```sql
SELECT
    af.Company,
    fg.Goal,
    COUNT(*) AS NumberOfInvestors
FROM
    AgeFiltered af
JOIN
    FINANCIAL_GOAL fg ON af.Phone = fg.Phone
GROUP BY
    af.Company,
    fg.Goal
ORDER BY
    af.Company,
    NumberOfInvestors DESC;
```

The final query:

- Joins the filtered investors with their financial goals using Phone as the key,
- Groups by company and goal to count how many investors at each company share the same goal,
- Orders the results by company and popularity of the goal (descending count).

This helps identify what financial goals are most common within different companies for mid-career investors.

<u>Final Code:</u>

```sql
WITH InvestorAge AS (
    SELECT
        i.Phone,
        i.Name,
        i.Company,
        i.DoB,
        DATEDIFF(YEAR, i.DoB, GETDATE()) AS Age
    FROM
        INVESTOR i
),
AgeFiltered AS (
    SELECT
        ia.Phone,
        ia.Name,
        ia.Company,
        ia.Age
    FROM
        InvestorAge ia
    WHERE
        ia.Age BETWEEN 30 AND 40
)
SELECT
    af.Company,
    fg.Goal,
    COUNT(*) AS NumberOfInvestors
```

```sql
FROM
    AgeFiltered af
JOIN
    FINANCIAL_GOAL fg ON af.Phone = fg.Phone
GROUP BY
    af.Company,
    fg.Goal
ORDER BY
    af.Company,
    NumberOfInvestors DESC;
```

Final Output:

	Company	Goal	NumberOfInvestors
1	CIMB	Children Education	1
2	CIMB	Start Business	1
3	CIMB	Buy Property	1
4	DBS	Start Business	1
5	DBS	Children Education	1
6	DBS	Financial Freedom	1
7	HSBC	Start Business	1
8	HSBC	Buy Property	1
9	Maybank	Retirement	1
10	OCBC	Financial Freedom	1
11	Standard Chartered	Travel the World	3
12	UOB	Financial Freedom	1
13	UOB	Medical Emergency	1

Query 7

Are male investors in their 20s making more money from their investments than their female counterparts in 2024?

Assumption:

- "Investors in their 20s" means investors aged between 20 and 29 years old, inclusive.
- Age is calculated as of '2025-03-29'.
- Investment performance is measured using:
  - Unrealized Gain/Loss as of '2024-03-31'
  - Annualized Return from the Portfolio1 table
- If there's no unrealized gain/loss record, it is treated as 0 using COALESCE.

Explanation:

This query is designed to **compare investment performance** between young male and female investors. It does this using two Common Table Expressions (CTEs): one for age filtering, and another for linking performance metrics.

Step 1: Filter Investors Aged 20–29

```sql
WITH InvestorAge AS (
    SELECT
        i.Phone,
        i.Name,
        i.Gender,
        DATEDIFF(YEAR, i.DoB, '2025-03-29') AS Age,
        i.AnnualIncome
    FROM
        INVESTOR i
    WHERE
        DATEDIFF(YEAR, i.DoB, '2025-03-29') BETWEEN 20 AND 29
),
```

This CTE:

- Computes the investor's age by subtracting date of birth from a fixed reference date
- Filters investors who are between 20 and 29 years old
- Keeps their phone, name, gender, and income for later use

Step 2: Join with Investment Performance Data

```sql
InvestmentPerformance AS (
    SELECT
        ia.Phone,
        ia.Name,
        ia.Gender,
        ia.Age,
        COALESCE(ugl.Amount, 0) AS UnrealizedGainLoss,
        p.AnnualizedReturn
    FROM
        InvestorAge ia
    JOIN
        Portfolio1 p ON ia.Phone = p.Phone
    LEFT JOIN
        UNREALIZED_GAIN_LOSS ugl ON ia.Phone = ugl.Phone
                                    AND p.PID = ugl.PID
                                    AND ugl.Date = '2024-03-31'  -- Most recent data point
)
```

This part:

- Joins the previously filtered investors with their portfolios
- Left joins UNREALIZED_GAIN_LOSS to capture gains/losses as of 2024-03-31
- Uses COALESCE to treat missing gain/loss values as 0
- Also includes AnnualizedReturn from each portfolio

Step 3: Aggregate Performance by Gender

```sql
SELECT
    Gender,
    COUNT(*) AS InvestorCount,
    AVG(UnrealizedGainLoss) AS AvgUnrealizedGainLoss,
    AVG(AnnualizedReturn) AS AvgAnnualizedReturn
FROM
    InvestmentPerformance
GROUP BY
    Gender
ORDER BY
    AvgUnrealizedGainLoss DESC;
```

This final part:

- Groups the investors by Gender
- Counts how many investors fall into each gender category
- Computes the average unrealized gain/loss and average annualized return
- Orders the results by average gain/loss to see which group performed better

<u>Final Code:</u>

```sql
WITH InvestorAge AS (
    SELECT
        i.Phone,
        i.Name,
        i.Gender,
        DATEDIFF(YEAR, i.DoB, '2025-03-29') AS Age,
        i.AnnualIncome
    FROM
        INVESTOR i
    WHERE
        DATEDIFF(YEAR, i.DoB, '2025-03-29') BETWEEN 20 AND 29
),
InvestmentPerformance AS (
    SELECT
        ia.Phone,
        ia.Name,
        ia.Gender,
        ia.Age,
        COALESCE(ugl.Amount, 0) AS UnrealizedGainLoss,
        p.AnnualizedReturn
    FROM
        InvestorAge ia
    JOIN
        Portfolio1 p ON ia.Phone = p.Phone
    LEFT JOIN
        UNREALIZED_GAIN_LOSS ugl ON ia.Phone = ugl.Phone
                                AND p.PID = ugl.PID
                                AND ugl.Date = '2024-03-31'  -- Most
recent data point
)
SELECT
    Gender,
    COUNT(*) AS InvestorCount,
    AVG(UnrealizedGainLoss) AS AvgUnrealizedGainLoss,
    AVG(AnnualizedReturn) AS AvgAnnualizedReturn
FROM
    InvestmentPerformance
GROUP BY
    Gender
ORDER BY
    AvgUnrealizedGainLoss DESC;
```

Final Output:

	Gender	InvestorCount	AvgUnrealizedGainLoss	AvgAnnualizedReturn
1	M	5	8611.296000	5.400000
2	F	10	107.267000	8.217000