Financial Portfolio Management System

**By**

**Mehul 102316082**

**Kunal 102316060**

**Pranjal 102316010**

**Suryansh 102316055**



**Under the guidance of**

**Dr. Manisha Kaushal**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| Chapter No. | Topic | Page No. |
| 1 | Introduction | 3 |
| 2 | Requirement Analysis | 4 |
| 3 | Tables | 5 |
| 4 | ER Diagram | 6 |
| 5 | ER to Table | 7-10 |
| 6 | Code | 12-19 |

**Introduction**

**Problem Statement**

In today’s fast-paced financial markets, **investors and fund managers** juggle multiple spreadsheets, trading platforms, and analytics tools to track their holdings, leaving them exposed to **data inconsistencies**, **delayed pricing updates**, and **manual calculation errors**. Without a single source of truth, reconciling transactions across accounts, portfolios, and assets becomes **time-consuming** and **error-prone**, while **real-time performance metrics** and **risk assessments** often lag behind market movements.

To address these challenges, there is an urgent need for a **centralized, automated portfolio management system** that streamlines **order execution**, **transaction logging**, and **market-data synchronization** in one place. From **seamless asset allocation** and **instantaneous ROI calculation** to **automated risk analysis** and **real-time alerts** for underperforming or high-risk positions, every feature must integrate smoothly to empower users with **accurate insights**, **faster decision-making**, and **robust audit trails**—all accessible through a single, user-friendly platform.

**Key Objectives**

1. **Centralized User & Account Management**

Securely register users and maintain multiple account profiles with consistent data integrity.

1. **Dynamic Portfolio Lifecycle**

Enable users to create, update, and delete portfolios, tracking assets and total value.

1. **Automated Transaction Processing**

On transaction insert, auto-update portfolio assets, total value, and audit logs.

1. **Real-Time Market Data Sync**

Refresh asset prices daily via market data triggers to keep values accurate.

1. **Performance Metrics Computation**

Nightly procedure calculates ROI, annual returns, and profit/loss for each portfolio.

1. **Comprehensive Risk Analysis**

Generate volatility, Sharpe ratio, and beta coefficients through scheduled stored procedures.

1. **Automated Alert Generation**

Trigger alerts for portfolios with low ROI or high risk, ensuring timely notifications.

1. **Secure Deposit & Withdrawal Validation**

Pre-insert triggers enforce balance checks and log every fund movement.

**Requirement Analysis**

* **Portfolio Tracking Across Multiple Users**

The system must allow multiple users to manage financial portfolios consisting of various assets (stocks, mutual funds, etc.), with real-time tracking of asset quantities and current values.

* **Market Data Integration**

The system must allow multiple users to manage financial portfolios consisting of various assets (stocks, mutual funds, etc.), with real-time tracking of asset quantities and current market values.

* **Automated Performance and Risk Calculation**

The system should automatically compute financial metrics (ROI, annual return, profit/loss) and risk metrics (volatility, Sharpe ratio, beta) for each portfolio, stored in separate tables.

* **Trigger-Based Consistency and Logging**

Database triggers must be implemented to ensure data integrity (e.g., balance updates on transactions) and to maintain audit logs for sensitive financial operations.

* **Alert Mechanism for Portfolio Performance**

The system must notify or log alerts when a portfolio’s ROI turns negative, helping users take timely action or review investments.

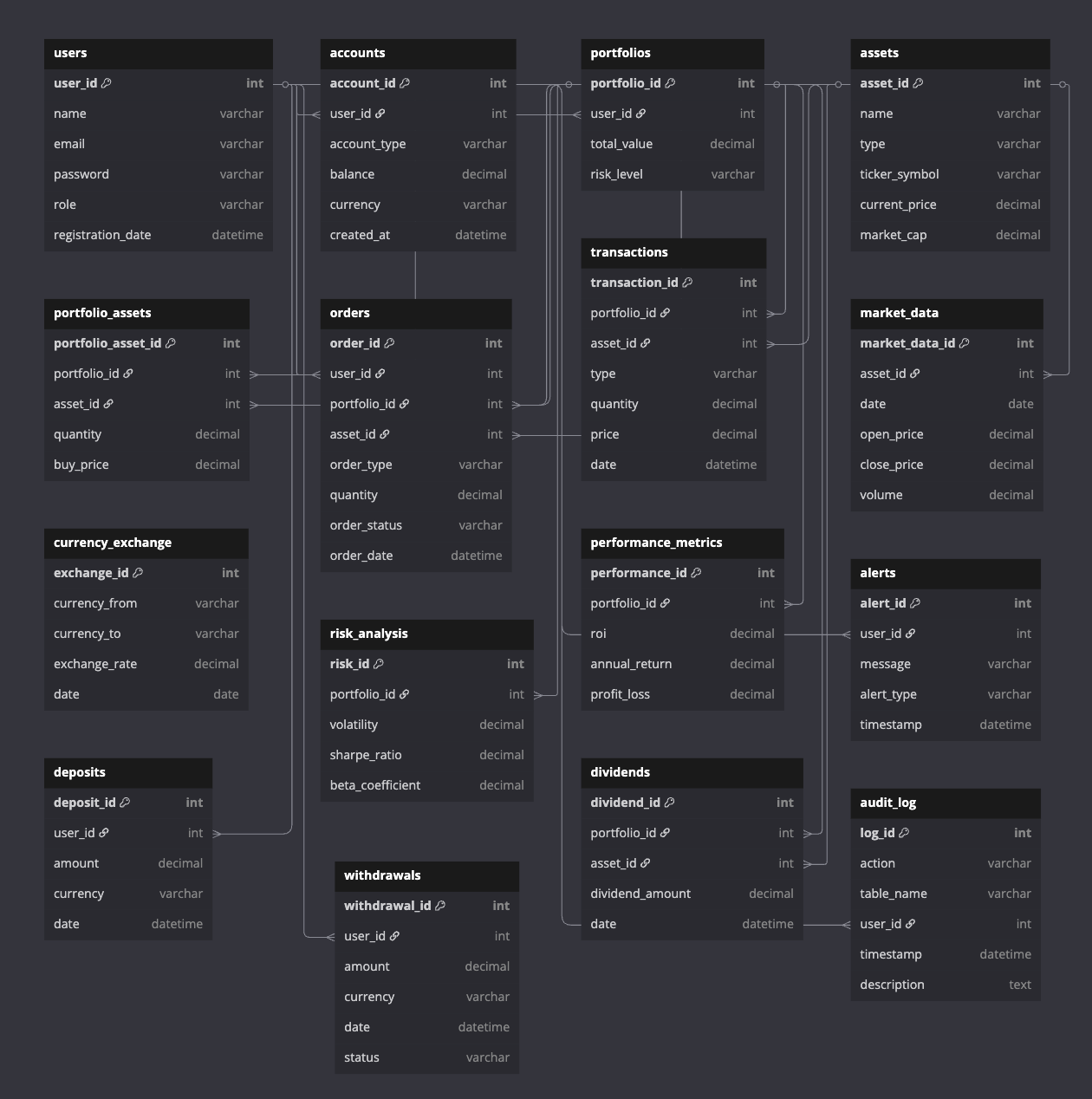
* **Scheduled Maintenance Procedures**

A nightly batch job should automate the update of asset prices, compute new metrics, and generate alerts—ensuring hands-free data refresh and analysis.

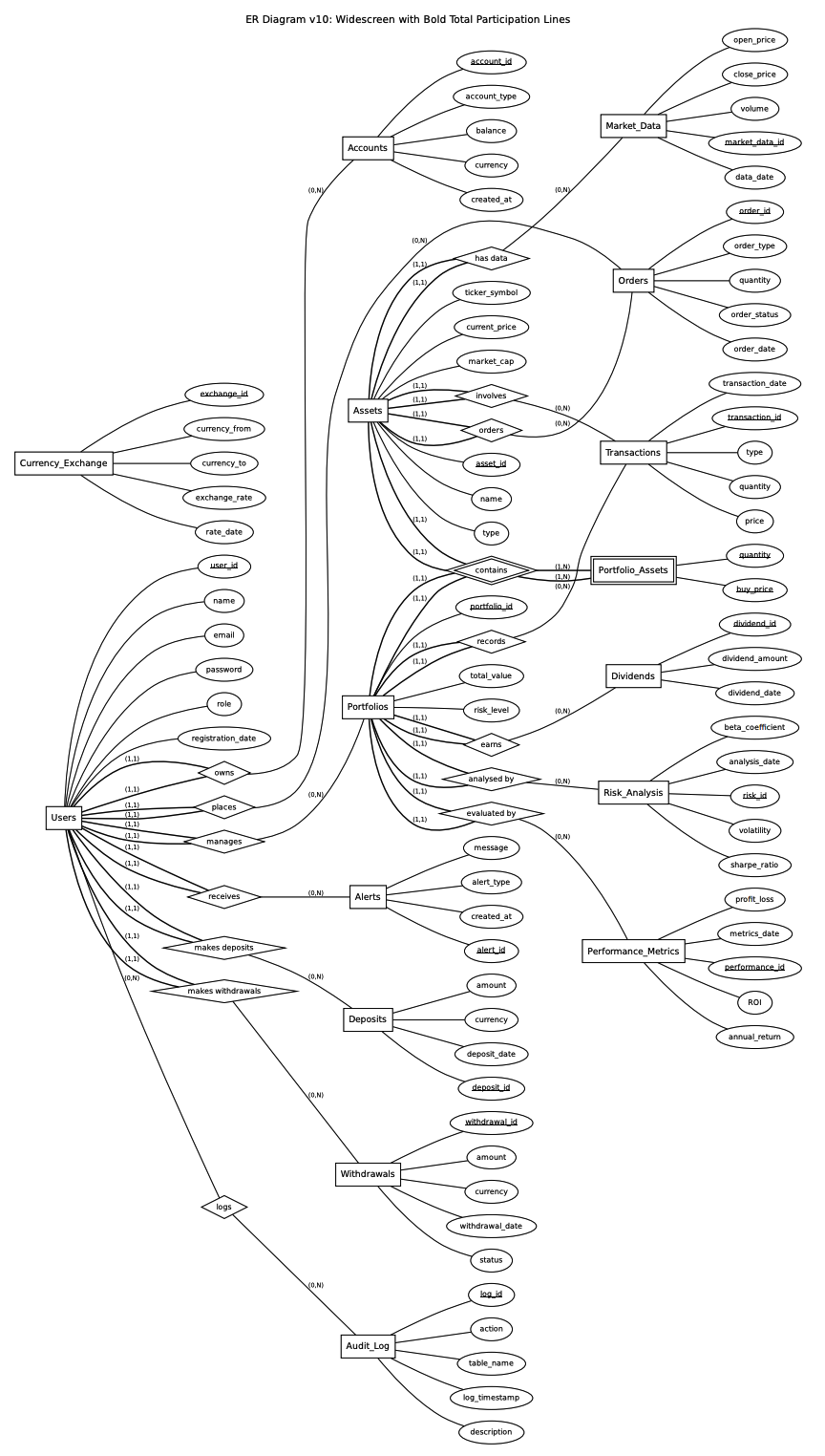
* **User-Friendly Schema with PL/SQL Logic**

The database design should be normalized and supported with PL/SQL procedures and sequences to handle backend logic efficiently without a separate application server.

**Tables**



**ER Diagram**



**ER To TABLES**

1. Users

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| user\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| name | VARCHAR2(100) | NOT NULL |
| email | VARCHAR2(100) | UNIQUE, NOT NULL |
| password | VARCHAR2(200) | NOT NULL |
| role | VARCHAR2(20) | DEFAULT 'user', NOT NULL |
| registration\_date | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

1. Accounts

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| account\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| user\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Users(user\_id) |
| account\_type | VARCHAR2(50) | NOT NULL |
| balance | NUMBER(18,2) | DEFAULT 0, NOT NULL |
| currency | VARCHAR2(10) | NOT NULL |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

1. Assets

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| asset\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| name | VARCHAR2(100) | NOT NULL |
| type | VARCHAR2(50) | NOT NULL |
| ticker\_symbol | VARCHAR2(20) | UNIQUE, NOT NULL |
| current\_price | NUMBER(18,4) | NOT NULL |
| market\_cap | NUMBER(20,2) |  |

1. Portfolios

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| portfolio\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| user\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Users(user\_id) |
| total\_value | NUMBER(20,2) | DEFAULT 0 |
| risk\_level | VARCHAR2(20) |  |

1. Portfolio\_Assets

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data type** | **Constraints** |
| portfolio\_asset\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| portfolio\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Portfolios(portfolio\_id) |
| asset\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Assets(asset\_id) |
| quantity | NUMBER(18,4) | DEFAULT 0 |
| buy\_price | NUMBER(18,4) |  |

1. Orders

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| order\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| user\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Users(user\_id) |
| portfolio\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Portfolios(portfolio\_id) |
| asset\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Assets(asset\_id) |
| order\_type | VARCHAR2(10) | NOT NULL /\* 'BUY' or 'SELL' \*/ |
| quantity | NUMBER(18,4) | NOT NULL |
| order\_status | VARCHAR2(20) | DEFAULT 'PENDING' |
| order\_date | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

1. Transactions

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| transaction\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| portfolio\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Portfolios(portfolio\_id) |
| asset\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Assets(asset\_id) |
| type | VARCHAR2(20) | NOT NULL /\* 'BUY','SELL','DIVIDEND' \*/ |
| quantity | NUMBER(18,4) | NOT NULL |
| price | NUMBER(18,4) | NOT NULL |
| date | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

1. Market\_Data

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| market\_data\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| asset\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Assets(asset\_id) |
| date | DATE | NOT NULL |
| open\_price | NUMBER(18,4) |  |
| close\_price | NUMBER(18,4) |  |
| volume | NUMBER(20,0) |  |

1. Currency\_Exchange

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| exchange\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| currency\_from | VARCHAR2(10) | NOT NULL |
| currency\_to | VARCHAR2(10) | NOT NULL |
| exchange\_rate | NUMBER(18,6) | NOT NULL |
| date | DATE | NOT NULL |

1. Risk\_Analysis

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| risk\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| portfolio\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Portfolios(portfolio\_id) |
| volatility | NUMBER(10,4) |  |
| sharpe\_ratio | NUMBER(10,4) |  |
| beta\_coefficient | NUMBER(10,4) |  |

1. Performance\_Metrics

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| performance\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| portfolio\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Portfolios(portfolio\_id) |
| roi | NUMBER(10,4) |  |
| annual\_return | NUMBER(10,4) |  |
| profit\_loss | NUMBER(18,4) |  |

1. Alerts

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| alert\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| user\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Users(user\_id) |
| message | VARCHAR2(255) | NOT NULL |
| alert\_type | VARCHAR2(50) |  |
| timestamp | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

1. Deposits

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| deposit\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| user\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Users(user\_id) |
| amount | NUMBER(18,4) | NOT NULL |
| currency | VARCHAR2(10) | NOT NULL |
| date | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

1. Withdrawals

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| withdrawal\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| user\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Users(user\_id) |
| amount | NUMBER(18,4) | NOT NULL |
| currency | VARCHAR2(10) | NOT NULL |
| date | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |
| status | VARCHAR2(20) | DEFAULT 'PENDING' |

1. Dividends

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| dividend\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| portfolio\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Portfolios(portfolio\_id) |
| asset\_id | NUMBER | NOT NULL, FOREIGN KEY REFERENCES Assets(asset\_id) |
| dividend\_amount | NUMBER(18,4) | NOT NULL |
| date | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

  16.          Audit\_log

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data Type** | **Constraint(s)** |
| log\_id | NUMBER | PRIMARY KEY, NOT NULL, GENERATED ALWAYS AS IDENTITY |
| action | VARCHAR2(100) | NOT NULL |
| table\_name | VARCHAR2(50) | NOT NULL |
| user\_id | NUMBER | FOREIGN KEY REFERENCES Users(user\_id) |
| timestamp | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |
| description | VARCHAR2(4000) |  |

[**Codes**](https://github.com/mehul79/Financial-Portfolio-Management-System)

**-- Schema Creation**

-- 1. Users

CREATE TABLE Users (

user\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

name VARCHAR2(100) NOT NULL,

email VARCHAR2(255) NOT NULL UNIQUE,

password VARCHAR2(255) NOT NULL,

role VARCHAR2(50) NOT NULL,

registration\_date DATE DEFAULT SYSDATE NOT NULL

);

-- 2. Accounts

CREATE TABLE Accounts (

account\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

user\_id NUMBER NOT NULL,

account\_type VARCHAR2(50) NOT NULL,

balance NUMBER(18,2) DEFAULT 0 NOT NULL,

currency VARCHAR2(3) DEFAULT 'USD' NOT NULL,

created\_at DATE DEFAULT SYSDATE NOT NULL,

CONSTRAINT fk\_accounts\_user FOREIGN KEY (user\_id) REFERENCES Users(user\_id)

);

-- 3. Assets

CREATE TABLE Assets (

asset\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

name VARCHAR2(100) NOT NULL,

type VARCHAR2(50) NOT NULL,

ticker\_symbol VARCHAR2(20) NOT NULL UNIQUE,

current\_price NUMBER(18,4) DEFAULT 0 NOT NULL,

market\_cap NUMBER(20,2) DEFAULT 0 NOT NULL

);

-- 4. Portfolios

CREATE TABLE Portfolios (

portfolio\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

user\_id NUMBER NOT NULL,

total\_value NUMBER(18,2) DEFAULT 0 NOT NULL,

risk\_level VARCHAR2(20) DEFAULT 'Medium',

CONSTRAINT fk\_portfolios\_user FOREIGN KEY (user\_id) REFERENCES Users(user\_id)

);

-- 5. Portfolio\_Assets

CREATE TABLE Portfolio\_Assets (

portfolio\_asset\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

portfolio\_id NUMBER NOT NULL,

asset\_id NUMBER NOT NULL,

quantity NUMBER(20,4) NOT NULL,

buy\_price NUMBER(18,4) NOT NULL,

CONSTRAINT fk\_pa\_portfolio FOREIGN KEY (portfolio\_id) REFERENCES Portfolios(portfolio\_id),

CONSTRAINT fk\_pa\_asset FOREIGN KEY (asset\_id) REFERENCES Assets(asset\_id),

CONSTRAINT uq\_pa\_portfolio\_asset UNIQUE (portfolio\_id, asset\_id)

);

-- 6. Transactions

CREATE TABLE Transactions (

transaction\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

portfolio\_id NUMBER NOT NULL,

asset\_id NUMBER NOT NULL,

type VARCHAR2(20) CHECK (type IN ('BUY','SELL','DIVIDEND')) NOT NULL,

quantity NUMBER(20,4) NOT NULL,

price NUMBER(18,4) NOT NULL,

transaction\_date DATE DEFAULT SYSDATE NOT NULL,

CONSTRAINT fk\_trx\_portfolio FOREIGN KEY (portfolio\_id) REFERENCES Portfolios(portfolio\_id),

CONSTRAINT fk\_trx\_asset FOREIGN KEY (asset\_id) REFERENCES Assets(asset\_id)

);

-- 7. Orders

CREATE TABLE Orders (

order\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

user\_id NUMBER NOT NULL,

asset\_id NUMBER NOT NULL,

order\_type VARCHAR2(20) CHECK (order\_type IN ('BUY','SELL')) NOT NULL,

quantity NUMBER(20,4) NOT NULL,

order\_status VARCHAR2(20) DEFAULT 'PENDING' NOT NULL,

order\_date DATE DEFAULT SYSDATE NOT NULL,

CONSTRAINT fk\_orders\_user FOREIGN KEY (user\_id) REFERENCES Users(user\_id),

CONSTRAINT fk\_orders\_asset FOREIGN KEY (asset\_id) REFERENCES Assets(asset\_id)

);

**-- Triggers**

**TRG\_TRX\_AFTER\_INSERT**

* Logs a new transaction and updates portfolio asset quantities after a BUY or SELL transaction is inserted.

CREATE OR REPLACE TRIGGER trg\_trx\_after\_insert

AFTER INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_new\_qty NUMBER;

v\_total\_val NUMBER;

BEGIN

-- Update or insert Portfolio\_Assets

BEGIN

SELECT quantity INTO v\_new\_qty

FROM Portfolio\_Assets

WHERE portfolio\_id = :NEW.portfolio\_id AND asset\_id = :NEW.asset\_id

FOR UPDATE;

UPDATE Portfolio\_Assets

SET quantity = v\_new\_qty + CASE WHEN :NEW.type = 'BUY' THEN :NEW.quantity ELSE -:NEW.quantity END

WHERE portfolio\_id = :NEW.portfolio\_id AND asset\_id = :NEW.asset\_id;

EXCEPTION WHEN NO\_DATA\_FOUND THEN

IF :NEW.type = 'BUY' THEN

INSERT INTO Portfolio\_Assets (portfolio\_id, asset\_id, quantity, buy\_price)

VALUES (:NEW.portfolio\_id, :NEW.asset\_id, :NEW.quantity, :NEW.price);

END IF;

END;

-- Recalculate total portfolio value

SELECT NVL(SUM(pa.quantity \* a.current\_price), 0) INTO v\_total\_val

FROM Portfolio\_Assets pa JOIN Assets a ON pa.asset\_id = a.asset\_id

WHERE pa.portfolio\_id = :NEW.portfolio\_id;

UPDATE Portfolios

SET total\_value = v\_total\_val

WHERE portfolio\_id = :NEW.portfolio\_id;

-- Log the transaction

INSERT INTO Audit\_Log(action, table\_name, description)

VALUES('INSERT', 'Transactions',

'Transaction ' || :NEW.transaction\_id || ' of type ' || :NEW.type);

END;

**----------------------------------------------------------------------------------------**

**TRG\_MD\_AFTER\_UPDATE**

* Updates the current price of the asset in the Assets table when its market data is updated.

CREATE OR REPLACE TRIGGER trg\_md\_after\_update

AFTER UPDATE OF close\_price

ON Market\_Data

FOR EACH ROW

BEGIN

-- 2a. Update Assets.current\_price

UPDATE Assets

SET current\_price = :NEW.close\_price

WHERE asset\_id = :NEW.asset\_id;

-- 2b. Audit log entry

INSERT INTO Audit\_Log(action, table\_name, user\_id, description)

VALUES (

'UPDATE',

'Market\_Data',

NULL,

'Market\_Data ' || :NEW.market\_data\_id ||

' close\_price changed from ' || :OLD.close\_price ||

' to ' || :NEW.close\_price

);

END trg\_md\_after\_update;

**----------------------------------------------------------------------------------------**

**TRG\_WD\_BEFORE\_INSERT**

* Validates user balance before allowing a withdrawal; blocks if balance is insufficient for the requested amount.

CREATE OR REPLACE TRIGGER trg\_wd\_before\_insert

BEFORE INSERT

ON Withdrawals

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

-- 3a. Check account balance

SELECT balance

INTO v\_balance

FROM Accounts

WHERE user\_id = :NEW.user\_id

FOR UPDATE;

IF v\_balance < :NEW.amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds for withdrawal.');

END IF;

-- 3b. Deduct amount

UPDATE Accounts

SET balance = v\_balance - :NEW.amount

WHERE user\_id = :NEW.user\_id;

-- 3c. Audit log entry

INSERT INTO Audit\_Log(action, table\_name, user\_id, description)

VALUES (

'INSERT',

'Withdrawals',

:NEW.user\_id,

'Withdrawal of ' || :NEW.amount || ' ' || :NEW.currency

);

END trg\_wd\_before\_insert;

**----------------------------------------------------------------------------------------**

**TRG\_DIV\_AFTER\_INSERT**

* Credits the dividend amount to the corresponding user’s account balance and logs the event in Audit\_Log.

CREATE OR REPLACE TRIGGER trg\_div\_after\_insert

AFTER INSERT

ON Dividends

FOR EACH ROW

BEGIN

-- 4a. Insert matching dividend transaction

INSERT INTO Transactions (portfolio\_id, asset\_id, type, quantity, price)

VALUES (

:NEW.portfolio\_id,

:NEW.asset\_id,

'DIVIDEND',

0, -- dividends don’t change share count

:NEW.dividend\_amount

);

-- 4b. Audit log entry

INSERT INTO Audit\_Log(action, table\_name, user\_id, description)

VALUES (

'INSERT',

'Dividends',

NULL,

'Dividend ' || :NEW.dividend\_id ||

' of amount ' || :NEW.dividend\_amount

**--Procedures PL/SQL**

**PROC\_UPDATE\_MARKET\_PRICES -**

Updates each asset's current price using today's corresponding market data close price.

**PROC\_COMPUTE\_METRICS\_AND\_RISK –**

Calculates portfolio ROI, returns, profit/loss, and risk metrics, then updates related tables.

**PROC\_AUTO\_ALERTS –**

Generates alerts for portfolios with negative ROI based on today’s performance data.

**NIGHTLY\_MAINTENANCE –**

Runs all key procedures sequentially to update prices, metrics, risks, and generate alerts.

**CODE -**

-- Update Market Prices

CREATE OR REPLACE PROCEDURE proc\_update\_market\_prices AS

BEGIN

UPDATE Assets a SET a.current\_price = (

SELECT md.close\_price FROM Market\_Data md

WHERE md.asset\_id = a.asset\_id AND TRUNC(md.data\_date) = TRUNC(SYSDATE)

)

WHERE EXISTS (

SELECT 1 FROM Market\_Data md

WHERE md.asset\_id = a.asset\_id AND TRUNC(md.data\_date) = TRUNC(SYSDATE)

);

COMMIT;

END;

-- Compute Metrics & Risk

CREATE OR REPLACE PROCEDURE proc\_compute\_metrics\_and\_risk AS

CURSOR c IS SELECT portfolio\_id, total\_value FROM Portfolios;

v\_old NUMBER; v\_new NUMBER; v\_roi NUMBER; v\_ann NUMBER; v\_pl NUMBER;

v\_vol NUMBER := 0.1; v\_sharpe NUMBER; v\_beta NUMBER := 1.0;

BEGIN

FOR r IN c LOOP

SELECT NVL(SUM(pa.quantity \* a.current\_price), 0)

INTO v\_new FROM Portfolio\_Assets pa JOIN Assets a ON pa.asset\_id = a.asset\_id

WHERE pa.portfolio\_id = r.portfolio\_id;

v\_old := r.total\_value;

v\_roi := CASE WHEN v\_old > 0 THEN (v\_new - v\_old)/v\_old ELSE NULL END;

v\_ann := v\_roi \* 365; v\_pl := v\_new - v\_old;

v\_sharpe := CASE WHEN v\_vol > 0 THEN v\_roi / v\_vol ELSE NULL END;

MERGE INTO Performance\_Metrics pm

USING (SELECT r.portfolio\_id pid, TRUNC(SYSDATE) d FROM DUAL) src

ON (pm.portfolio\_id = src.pid AND TRUNC(pm.metrics\_date)=src.d)

WHEN MATCHED THEN UPDATE SET ROI=v\_roi, annual\_return=v\_ann, profit\_loss=v\_pl

WHEN NOT MATCHED THEN INSERT VALUES (

Performance\_Metrics\_SEQ.NEXTVAL, r.portfolio\_id, v\_roi, v\_ann, v\_pl, TRUNC(SYSDATE)

);

MERGE INTO Risk\_Analysis ra

USING (SELECT r.portfolio\_id pid, TRUNC(SYSDATE) d FROM DUAL) src

ON (ra.portfolio\_id = src.pid AND TRUNC(ra.analysis\_date)=src.d)

WHEN MATCHED THEN UPDATE SET volatility=v\_vol, sharpe\_ratio=v\_sharpe, beta\_coefficient=v\_beta

WHEN NOT MATCHED THEN INSERT VALUES (

Risk\_Analysis\_SEQ.NEXTVAL, r.portfolio\_id, v\_vol, v\_sharpe, v\_beta, TRUNC(SYSDATE)

);

END LOOP;

UPDATE Portfolios p SET total\_value = (

SELECT NVL(SUM(pa.quantity \* a.current\_price),0)

FROM Portfolio\_Assets pa JOIN Assets a ON pa.asset\_id = a.asset\_id

WHERE pa.portfolio\_id = p.portfolio\_id

);

COMMIT;

END;

-- Auto Alerts

CREATE OR REPLACE PROCEDURE proc\_auto\_alerts AS

BEGIN

FOR r IN (

SELECT pm.portfolio\_id, pm.ROI, p.user\_id

FROM Performance\_Metrics pm JOIN Portfolios p ON pm.portfolio\_id = p.portfolio\_id

WHERE TRUNC(pm.metrics\_date) = TRUNC(SYSDATE) AND pm.ROI < 0

) LOOP

INSERT INTO Alerts VALUES (

Alerts\_SEQ.NEXTVAL, r.user\_id,

'Portfolio '||r.portfolio\_id||' negative ROI: '||TO\_CHAR(r.ROI\*100,'FM990.00')||'%',

'Negative ROI', SYSDATE

);

END LOOP;

COMMIT;

END;

-- Maintenance Runner

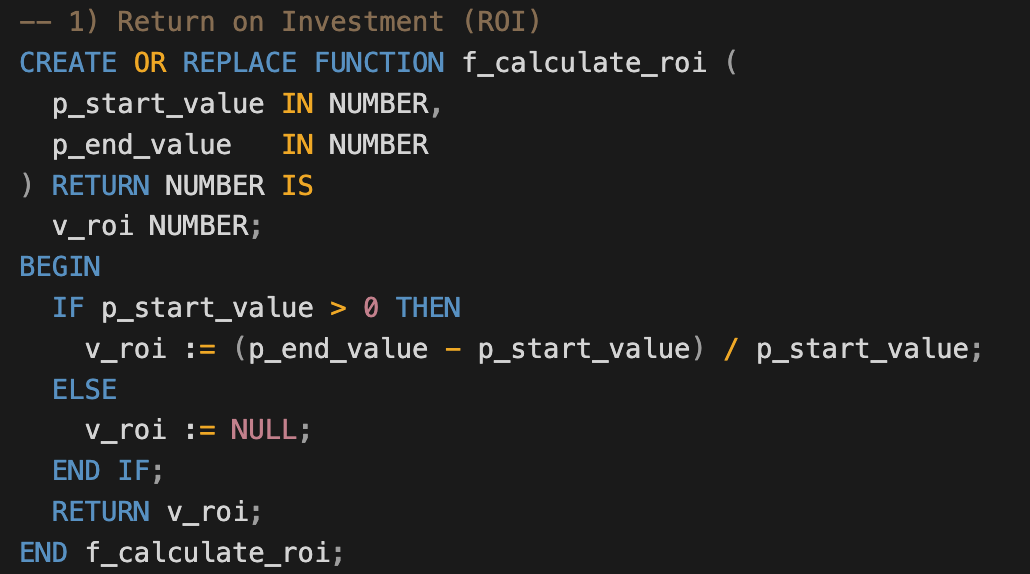
CREATE OR REPLACE PROCEDURE nightly\_maintenance AS

BEGIN

**-- Functions**

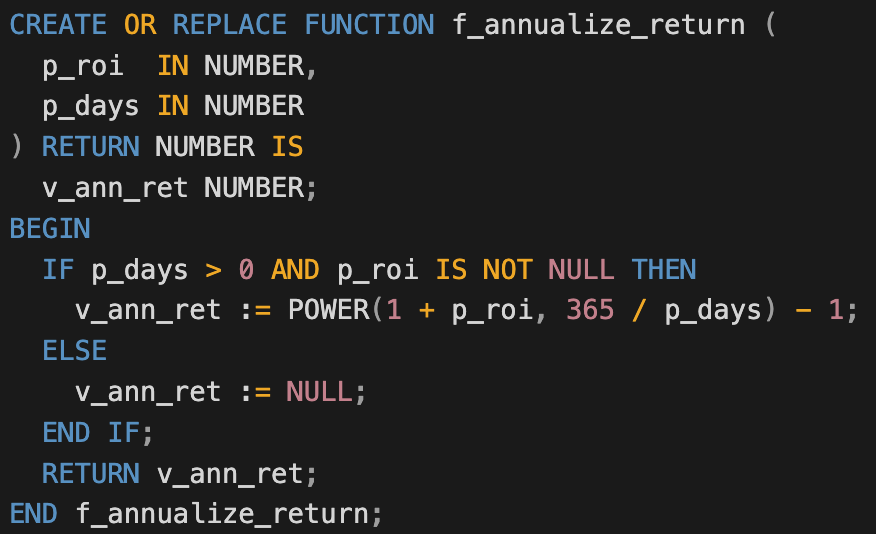
**F\_CALCULATE\_ROI –**

* Calculates the Return on Investment (ROI) by comparing the start and end values, returning the percentage gain or loss.

****

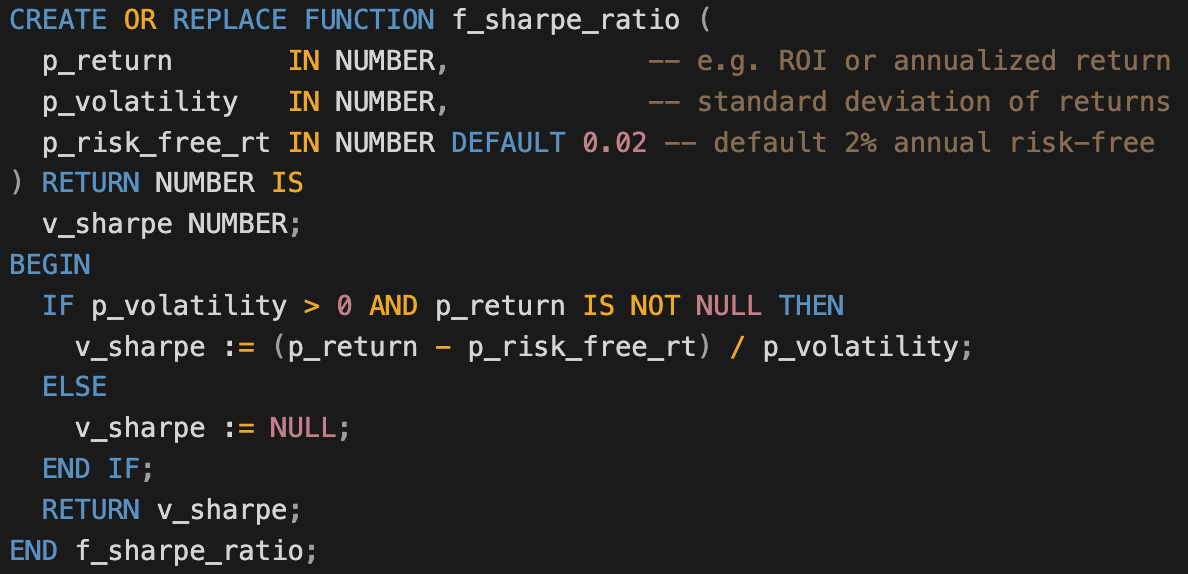
**F\_ANNUALIZE\_RETURN –**

* Converts the given ROI and holding period (in days) into an annualized return using the formula: (1 + ROI)^(365/days) - 1.

****

**F\_SHARPE\_RATIO –**

* Computes the Sharpe Ratio by dividing the excess return (return minus risk-free rate) by the standard deviation (volatility) of the return.

****

**Conclusion**

To conclude,  
the development of the **Financial Portfolio Management System** using **Oracle SQL** and **PL/SQL** demonstrates the effectiveness of a **database-driven approach** in managing complex financial data. With a **normalized schema** of 16 interrelated tables, the system efficiently tracks **users, portfolios, assets, transactions**, and **market data**. Key operations such as updating **portfolio values**, calculating **performance metrics** (ROI, profit/loss, annual returns), and analyzing **risk** (volatility, Sharpe ratio) are handled through **well-structured stored procedures** and **scheduled jobs**.

Triggers ensure **real-time data consistency**, while **SQL functions** support modular and reusable financial logic. This **backend-only system**, integrated entirely within the **Oracle environment**, offers **scalability, accuracy**, and **maintainability**, making it a robust solution for **portfolio analysis** and **financial reporting**.