PORTFOLIO MANAGEMENT SYSTEM

Prepared in fulfilment of

DATABASE SYSTEMS (CS F214)

By

VARUN VARMA 2020B4A70844P

GAURAV MISHRA 2020B3A70917P

Submitted to

Dr. Amit Dua

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE,

PILANI, PILANI CAMPUS

ACKNOWLEDGEMENT

Firstly, we would like to express our gratitude to Dr. Amit Dua for guiding us throughout the course and equipping us with the necessary tools with which we were able to complete our project in time.

Secondly, we would like to thank the entire DBS team for always being available and helping us clarify the issues that we faced in our journey of completing the project and in the course,

Thirdly, We would like to thank the institute for providing us with the opportunity to work on a project of this magnitude.

Introduction

The Portfolio Management System is a well-designed database that can store and manage relevant data for the investors' portfolios. The database stores information about investments, performance metrics, market data and other financial information. The database design has tables for investments, performance metrics, market data and other financial information. The design process of this project first involved the creation of an Entity Relationship (ER) Diagram. This was followed by conversion into a Relationship Schema where we define the different relations. This was followed by normalization into 3NF.

The database design also includes various measures to ensure data consistency through primary keys, foreign keys and check constraints. Primary keys will be used to uniquely identify each record in the tables, while foreign keys will be used to establish relationships between the tables. Check constraints will be used to ensure that only valid data is stored in the database.

Entity Relationship Diagram

The first step towards database design is to identify the entities and the relationships between said entities. These can be succinctly depicted using an Entity Relationship Diagram (ERD). To get started and to design the diagram we have taken the help of LucidChart, an online platform useful to draw ER diagrams with ease.

The major entities which we recognised along with their attributes are given below:

Investment
- investment_ID (Primary key)
- investment_name
- investment_type
- shares_held

Performance Metrics
- metric_ID (Primary key)
- total_return (Foreign key)
- annualized_return
- risk_level

Market Data
- market_ID (Primary key)
- date
- stock_price
- exchange_rate
- commodity_price

Other Financial Factors
- fin_ID (Primary key)
- date
- interest_rate
- inflation_rate
- GDP_growth_rate

The Investment entity stores information pertaining to each investment made such the name and type of investment and the number of shares held. The performance metrics entity stores data relevant to how well the investment is performing in the market. The market data entity stores information about a particular stock or commodity such as the stock price and commodity. The other financial factors entity stores other financial information relevant while making and keeping track of investments in a portfolio.

The relationships between all these entities is described below:

- Investment -> Performance Metrics: One-to-Many relationship. This is chosen as each investment can have different performance metrics at different periods of time or different states of the market.

- Performance Metrics -> Market Data: One-to-Many relationship. This is because each Performance Metric can be drawn from different time periods of Market Data while each point in the Market Data will result in only one Performance Metric for the investment.

- Investment -> Other Financial Information: One-to-One relationship. Each investment corresponds to unique set of the information attributes specified in the other financial information entity.

The above information is hence presented as an ER Diagram given below. The cardinalities of each relationship is also specified in the diagram.

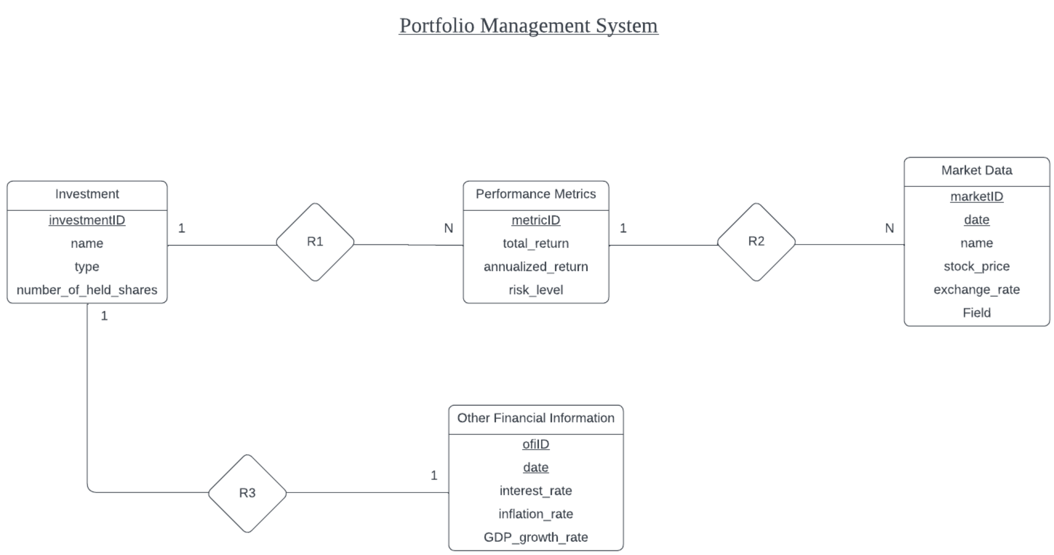


Figure 1: ER Diagram

Relationship Schema and Normalization

Towards a Relationship Schema

The next stage in database design after the creation of the ER diagram is to convert it into a relationship schema which can be modelled in SQL. To perform this conversion we first convert all the entities into relationships. Based on the ER diagram given above we have the following relations:

- *Investment (investment_ID (pk), investment_name, investment_type, shares_held)*

- *Performance Metrics (metric_ID (pk), investment_ID (fk), total_return, annualized_return, risk_level)*

- *Market Data (market_ID (pk), metric_ID (fk), date, stock_price, exchange_rate, commodity_price)*

- *Other Financial Information (fin_ID (pk), investment_ID (fk), date, interest_rate, inflation_rate, GDP_growth_rate)*

These relationships are formed on the basis of the relationships in the ER diagram. To model the One-to-Many relationships two tables are used where the second entity will have a foreign key to the first one and for the One-to-One relationships one table is used for the remaining entity. The primary and foreign keys are indicated in the relationships itself. These relationships and the schema fully describes the ER diagram which was created earlier.

Identifying Functional Dependencies

Now that we have the relationship schema, we model all the functional dependencies present in each relationship. This is an important step as this will pave the way forward towards normalization, which is an important aspect where we aim to reduce redundancy in our database design.

Based on the actual meaning and dependency of each attributes, the following functional dependencies arise for each relationship:

Investment

- investment_ID —> investment_name
- investment_ID —> investment_type
- investment_ID —> shares_held

Performance Metrics

- metric_ID —> total_return
- metric_ID —> annualized_return
- metric_ID —> risk_level
- risk_level —> total_return
- risk_level —> annualized_return

Market Data

- market_ID —> stock_price
- market_ID —> exchange_rate
- market_ID —> commodity_price
- market_ID —> date

Other Financial Information

- fin_ID —> date
- fin_ID —> interest_rate
- fin_ID —> inflation_rate
- fin_ID —> GDP_growth_rate

Note that we can also model the functional dependencies of the Other Financial Information table where both the fin_ID and date form a composite primary key, but we choose not to do so.

Normalization

Now that we have obtained all functional dependencies we are ready to normalize our relationship schema. We observe that every table is already in 1NF as there aren't any composite attributes. We also observe that every table is also in 2NF as every functional dependency in every table is *fully functionally dependent*.

However we observe that not every table in the schema is in 3NF. This is because in the Performance Metrics table, we see that metric_ID —> total_return and metric_ID —> annualized_return are *transitive functional dependencies* as these attributes are also functionally dependent on risk_level which is a *non-prime attribute* apart from metric_ID which is a *prime attribute.*
Hence to perform 3NF normalization we split the Performance Metrics table into two tables as shown:

Performance Metrics

- metric_ID —> total_return
- metric_ID —> annualized_return
- metric_ID —> risk_level_ID

Risk Level

- risk_level_ID —> risk_level
- risk_level_ID —> total_return
- risk_level_ID —> annualized_return

By doing so both these tables are now in 3NF and hence the relationship schema is in 3NF as every other table is already in 3NF. Hence the final set of functional dependencies are:

Investment

- investment_ID —> investment_name
- investment_ID —> investment_type
- investment_ID —> shares_held

Performance Metrics

- metric_ID —> total_return
- metric_ID —> annualized_return
- metric_ID —> risk_level_ID

Risk Level

- risk_level_ID —> risk_level
- risk_level_ID —> total_return
- risk_level_ID —> annualized_return

Market Data

- market_ID —> stock_price
- market_ID —> exchange_rate
- market_ID —> commodity_price
- market_ID —> date

Other Financial Information

- fin_ID —> date
- fin_ID —> interest_rate
- fin_ID —> inflation_rate
- fin_ID —> GDP_growth_rate

We are now done with normalization and are ready to move into implementing the relationship schema in SQL and perform querying operations on it.

Modelling the Schema in SQL and SQL Queries

We now implement the relationship schema which was finalized earlier in SQL and perform queries on it. The code snippets of each part are given below:

Creation of tables specified in the schema with appropriate key attributes and check constraints.

```sql
-- create the investment table
CREATE TABLE Investment (
    investment_ID    INT NOT NULL AUTO_INCREMENT,
    investment_name  VARCHAR(255) NOT NULL,
    investment_type  VARCHAR(255) CHECK(investment_type IN ("stock", "commodity", "bonds", "FD")),
    shares_held      INT NOT NULL CHECK(shares_held > 0),
    CONSTRAINT pk_investment PRIMARY KEY (investment_ID)
);

-- create the risk level table
CREATE TABLE Risk_Level (
    risk_level_ID     INT NOT NULL AUTO_INCREMENT,
    risk_level        DECIMAL(4,3),
    total_return      DECIMAL(10,3),
    annualized_return DECIMAL(10,3),
    CONSTRAINT pk_risk PRIMARY KEY(risk_level_ID)
);

-- create the performance metrics table
CREATE TABLE Performance_Metrics (
    metric_ID      INT NOT NULL AUTO_INCREMENT,
    investment_ID  INT NOT NULL,
    risk_level_ID  INT NOT NULL,
    CONSTRAINT pk_metric     PRIMARY KEY (metric_ID),
    CONSTRAINT fk_investment FOREIGN KEY (investment_ID) REFERENCES Investment(investment_ID) ON DELETE CASCADE,
    CONSTRAINT fk_risk       FOREIGN KEY (risk_level_ID) REFERENCES Risk_Level(risk_level_ID) ON DELETE CASCADE
);

-- create the market data table
CREATE TABLE Market_Data (
    market_ID       INT NOT NULL AUTO_INCREMENT,
    metric_ID       INT NOT NULL,
    market_date     DATE NOT NULL,
    stock_price     DECIMAL(10,3),
    exchange_rate   DECIMAL(10,3),
    commodity_price DECIMAL(10,3),
    CONSTRAINT pk_market PRIMARY KEY (market_ID),
    CONSTRAINT fk_metric FOREIGN KEY (metric_ID) REFERENCES Performance_Metrics(metric_ID) ON DELETE CASCADE
);
```

We then defined procedures for adding and deleting an investment which can be seen in the SQL script file. The procedure consists of basic insert and delete operations with appropriate sub-queries whenever required.

Queries

Add, update and delete an investment by calling our stored procedures.

```sql
-- insert an investment
CALL add_investment("Tata", "stock", 2, 2.00, 140, NULL, NULL, 0.65, 0.54, 0.23);

-- update the shares in the investment table
UPDATE Investment SET shares_held = 3 WHERE investment_ID = 1;

-- delete the above investment
CALL delete_investment(1);
```

1	19:35:36	CREATE TABLE Investment (investment_ID INT NOT NULL AUTO_INCREMENT, investment_name VARCHAR(255) NOT NULL, investment_type VARCHAR...	0 row(s) affected
2	19:35:36	CREATE TABLE Risk_Level (risk_level_ID INT NOT NULL AUTO_INCREMENT, risk_level DECIMAL(4,3), total_return DECIMAL(10,3), annualized_...	0 row(s) affected
3	19:35:36	CREATE TABLE Performance_Metrics (metric_ID INT NOT NULL AUTO_INCREMENT, investment_ID INT NOT NULL, risk_level_ID INT NOT NULL, C...	0 row(s) affected
4	19:35:36	CREATE TABLE Market_Data (market_ID INT NOT NULL AUTO_INCREMENT, metric_ID INT NOT NULL, market_date DATE NOT NULL, stock_pri...	0 row(s) affected
5	19:35:36	CREATE TABLE Other_Financial_Information (fin_ID INT NOT NULL AUTO_INCREMENT, fin_date DATE NOT NULL, investment_ID INT NOT NULL, intere...	0 row(s) affected
6	19:35:36	CREATE PROCEDURE add_investment(IN i_name VARCHAR(255), IN i_type VARCHAR(255), IN shares INT, IN risk DECIMAL(4,3), IN stock_price DECIMAL(10,3)...	0 row(s) affected
7	19:35:36	CREATE PROCEDURE add_investment_with_date(IN i_name VARCHAR(255), IN i_type VARCHAR(255), IN shares INT, IN risk DECIMAL(4,3), IN stock_price DEC...	0 row(s) affected
8	19:35:36	CREATE PROCEDURE delete_investment(IN i_id INT) MODIFIES SQL DATA DETERMINISTIC COMMENT "Delete an investment" BEGIN DELETE FROM Risk...	0 row(s) affected
9	19:35:44	CALL add_investment("Tata", "stock", 2, 2.00, 140, NULL, NULL, 0.65, 0.54, 0.23)	1 row(s) affected
10	19:35:44	UPDATE Investment SET shares_held = 3 WHERE investment_ID = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
11	19:35:44	CALL delete_investment(1)	1 row(s) affected

We then add more investments and market data to perform the query operations. The code for that is again present in the SQL script file.

Now we perform the remaining queries.

We use join here to join multiple tables and query info from all of them.

```sql
-- SQL QUERIES

-- Join the investments table with the risk level table to retrieve the total return for each investment.
SELECT Investment.investment_name, Investment.investment_type, Risk_Level.total_return
FROM Investment
INNER JOIN Performance_Metrics ON Investment.investment_ID = Performance_Metrics.investment_ID
INNER JOIN Risk_Level ON Risk_Level.risk_level_ID = Performance_Metrics.risk_level_ID;
```

100% 1:151

Result Grid Filter Rows: Search Export:

investment_na...	investment_ty...	total_return
Tata	stock	4200.000
VI	stock	4800.000
Taj	stock	36000.000
Reliance	stock	8400.000
Titan	stock	35200.000
Gold	commodity	133920.000
Oil	commodity	81000.000
Diamonds	commodity	728000.000

Similar usage of INNER JOIN.

```sql
-- Join the investments table with the market data table to retrieve the stock prices for a particular date.
SELECT Market_Data.stock_price, Market_Data.market_date, Investment.investment_name
FROM Investment
INNER JOIN Performance_Metrics ON Investment.investment_ID = Performance_Metrics.investment_ID
INNER JOIN Market_Data on Performance_Metrics.metric_ID = Market_Data.metric_ID;
```

100% 87:161

Result Grid Filter Rows: Search Export:

stock_pri...	market_date	investment_na...
140.000	2023-01-17	Tata
170.000	2023-01-31	Tata
210.000	2023-02-13	Tata
320.000	2023-01-11	VI
450.000	2023-02-01	VI
360.000	2023-03-15	VI
120.000	2023-01-19	Taj
450.000	2023-02-01	Taj
450.000	2023-02-15	Taj
210.000	2023-03-19	Reliance
130.000	2023-01-25	Reliance
145.000	2023-02-07	Reliance
110.000	2023-01-13	Titan
90.000	2023-03-18	Titan
60.000	2023-04-01	Titan
432.000	2023-04-13	Gold
600.000	2023-05-11	Gold
900.000	2023-06-15	Gold
900.000	2023-01-13	Oil
450.000	2023-01-23	Oil
500.000	2023-02-01	Oil
650.000	2023-03-17	Diamonds
450.000	2023-04-01	Diamonds

We use GROUP BY to group the investments by investment_type.

```sql
-- Group the investments by type and retrieve the average annualized return for each type.
SELECT Investment.investment_type, AVG(Risk_Level.annualized_return) AS average_annualized_return
FROM Investment
INNER JOIN Performance_Metrics ON Investment.investment_ID = Performance_Metrics.investment_ID
INNER JOIN Risk_Level ON Risk_Level.risk_level_ID = Performance_Metrics.risk_level_ID
GROUP BY Investment.investment_type;
```

100% 37:174

Result Grid Filter Rows: Search Export:

investment_ty...	average_annualized_ret...
stock	17720.0000000
commodity	314306.6666667

We order by risk level to get the top performing investments.

```sql
-- Filter the investments by risk level and retrieve the top-performing investments.
SELECT Investment.investment_name, Investment.investment_type, Risk_Level.risk_level
FROM Investment
INNER JOIN Performance_Metrics ON Investment.investment_ID = Performance_Metrics.investment_ID
INNER JOIN Risk_Level ON Risk_Level.risk_level_ID = Performance_Metrics.risk_level_ID
ORDER BY Risk_Level.risk_level;
```

100% 37:174

Result Grid Filter Rows: Search Export:

investment_na...	investment_ty...	risk_level
Gold	commodity	0.120
Oil	commodity	0.610
Taj	stock	1.000
Titan	stock	1.010
VI	stock	1.210
Reliance	stock	2.000
Tata	stock	2.010
Diamonds	commodity	4.010

Multiply the shares held and stock_price in total

```sql
-- Calculate the total value of all investments based on the number of shares held and the current stock prices from the market data table.
SELECT SUM(Investment.shares_held) * SUM(Market_Data.stock_price) AS total_value
FROM Investment
INNER JOIN Performance_Metrics ON Investment.investment_ID = Performance_Metrics.investment_ID
INNER JOIN Market_Data on Performance_Metrics.metric_ID = Market_Data.metric_ID;
```

100% 81:187

Result Grid Filter Rows: Search Export:

total_value
46089835.000

We sum over the annualized_returns here. (Note values are dummy values need not be to scale)

```sql
-- Calculate the portfolio's overall annualized return based on the investments' individual returns and the number of shares held.
SELECT SUM(annualized_return) AS overall_annualized_return
FROM Risk_Level;
```

100% 1:188

Result Grid Filter Rows: Search Export:

overall_annualized_ret...
1031520.000

Order the selection by the date to get the most recent inflation rate and impose a limit of 1.

```sql
-- Retrieve the most recent inflation rate from the other financial information table.
SELECT inflation_rate
FROM Other_Financial_Information
ORDER BY fin_date
DESC LIMIT 1;
```

inflation_rate
0.120

We use where clause to set the date range.

```sql
-- Calculate the percentage change in stock prices for a particular investment between two dates.
SELECT Investment.investment_name, Market_Data.stock_price
FROM Investment
INNER JOIN Performance_Metrics ON Investment.investment_ID = Performance_Metrics.investment_ID
INNER JOIN Market_Data on Performance_Metrics.metric_ID = Market_Data.metric_ID
WHERE Market_Data.market_date > "2023-01-01" AND Market_Data.market_date < "2023-06-01" AND Investment.investment_ID = 3;
```

investment_na...	stock_pri...
VI	320.000
VI	450.000
VI	360.000

```sql
-- Filter the market data table by date range and retrieve the stock prices for a particular investment (ID 3).
SELECT Market_Data.stock_price
FROM Investment
INNER JOIN Performance_Metrics ON Investment.investment_ID = Performance_Metrics.investment_ID
INNER JOIN Market_Data on Performance_Metrics.metric_ID = Market_Data.metric_ID
WHERE Investment.investment_ID = 3;
```

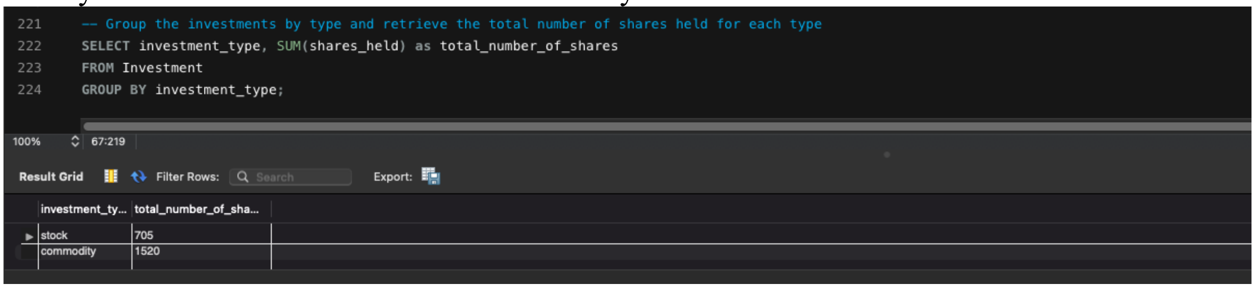
stock_price
320.000
450.000
360.000

Multiple INNER JOIN and ORDER BY clause is used to check with attributes for different tables.

```sql
-- Retrieve the top-performing investments based on annualized return and risk level.
SELECT Investment.investment_name, Investment.investment_type, Risk_Level.annualized_return, Risk_Level.risk_level
FROM Investment
INNER JOIN Performance_Metrics ON Investment.investment_ID = Performance_Metrics.investment_ID
INNER JOIN Risk_Level ON Risk_Level.risk_level_ID = Performance_Metrics.risk_level_ID
ORDER BY (Risk_Level.annualized_return AND Risk_Level.risk_level);
```

investment_na...	investment_ty...	annualized_retu...	risk_level
Tata	stock	4200.000	2.010
VI	stock	4800.000	1.210
Taj	stock	36000.000	1.000
Reliance	stock	8400.000	2.000
Titan	stock	35200.000	1.010
Gold	commodity	133920.000	0.120
Oil	commodity	81000.000	0.610
Diamonds	commodity	728000.000	4.010

Finally a GROUP BY clause is used here directly.



Conclusion and Scope for Improvement

We have successfully implemented the relationship schema and performed query operations on the data stored. We have used multiple procedures, check constraints, and triggers to ensure data consistency.

The database can be further improved by setting a temporal stream of real-time market data and perform more fast and relevant information queries. More economic factors can be incorporated in the database to produce more robust and informative queries.

THANK YOU