# Portfolio Management

# CREATE DATABASE portfolio\_management;

USE portfolio\_management;

**1. Investor Table**

This table stores information about investors.

CREATE TABLE investors (

investor\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) NOT NULL,

phone VARCHAR(15),

date\_of\_birth DATE

);

**2. Stock Table**

This table stores information about stocks.

CREATE TABLE stocks (

stock\_id INT AUTO\_INCREMENT PRIMARY KEY,

symbol VARCHAR(10) NOT NULL,

name VARCHAR(100) NOT NULL,

price DECIMAL(10, 2) NOT NULL

);

**3. Portfolio Table**

This table links investors and stocks, representing their investments (i.e., how many stocks they own and at what purchase price).

CREATE TABLE portfolio (

portfolio\_id INT AUTO\_INCREMENT PRIMARY KEY,

investor\_id INT,

stock\_id INT,

quantity INT NOT NULL,

purchase\_price DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (investor\_id) REFERENCES investors(investor\_id) ON DELETE CASCADE,

FOREIGN KEY (stock\_id) REFERENCES stocks(stock\_id) ON DELETE CASCADE

);

**4. portfolio\_investments table:**

`investment\_id` int NOT NULL AUTO\_INCREMENT,

`portfolio\_id` int NOT NULL,

`stock\_id` int NOT NULL,

`quantity` int NOT NULL,

`purchase\_price` decimal(10,2) NOT NULL,

PRIMARY KEY (`investment\_id`),

KEY `portfolio\_id` (`portfolio\_id`),

KEY `stock\_id` (`stock\_id`),

CONSTRAINT `portfolio\_investments\_ibfk\_1` FOREIGN KEY (`portfolio\_id`) REFERENCES `portfolio` (`portfolio\_id`) ON DELETE CASCADE,

CONSTRAINT `portfolio\_investments\_ibfk\_2` FOREIGN KEY (`stock\_id`) REFERENCES `stocks` (`stock\_id`),

CONSTRAINT `portfolio\_investments\_chk\_1` CHECK ((`quantity` > 0)),

CONSTRAINT `portfolio\_investments\_chk\_2` CHECK ((`purchase\_price` > 0))

);

### 1. ****Database Design Overview****

Several key entities and relationships in the system:

* **Investor**: Represents a user who is investing in portfolios.
* **Portfolio**: A collection of investments managed by an investor. A single investor can have multiple portfolios.
* **Investment**: Represents an individual holding in a specific portfolio (e.g., a stock or asset).:
* **Portfolio\_investments: represent a portfolio with corresponding investments made by an investor.**

### 2. ****Entities and Relationships****

#### Tables:

1. **Investor Table**:
   * Represents users in the system.
   * Fields: investor\_id, name, email, etc.
2. **Portfolio Table**:
   * Each investor can have multiple portfolios.
   * Fields: portfolio\_id, investor\_id (Foreign Key), portfolio\_name, creation\_date.
3. **Portfolio\_Investments Table**:
   * Each portfolio can contain multiple investments (such as stocks or other assets).
   * Fields: investment\_id, portfolio\_id (Foreign Key), stock\_id, quantity, purchase\_price.
4. **Stock Table** (Optional but useful):
   * Represents the stocks or assets that can be bought within portfolios.
   * Fields: stock\_id, name, symbol, current\_price, etc.

#### Relationships:

* **Investor - Portfolio**: One-to-many (One investor can have many portfolios).
* **Portfolio - Investment**: One-to-many (One portfolio can contain many investments).
* **Stock - Investment**: Many-to-one (One stock can appear in multiple investments).

### 3. ****Core Functionalities (Methods)****

#### A. **Investor Management**

* **createInvestor**: Allows creating a new investor.pp
* **findInvestorById**: Retrieves an investor's details by their ID.
* **updateInvestorDetails**: Allows updating the details of an investor.
* **deleteInvestor**: Deletes an investor from the system (and cascade deletes their portfolios and investments).

#### B. **Portfolio Management (Multiple Portfolios per User)**

1. **createPortfolio**: Allows an investor to create a new portfolio. The investor can have multiple portfolios, so a portfolio\_name would help distinguish them.
   * **Input**: investor\_id, portfolio\_name
   * **Output**: A new portfolio linked to the investor.
2. **findPortfoliosByUserId**: Retrieves all portfolios for a specific investor.
   * **Input**: investor\_id
   * **Output**: List of all portfolios belonging to the investor.
3. **updatePortfolioName**: Allows updating the name of an existing portfolio.
   * **Input**: portfolio\_id, new\_name
   * **Output**: Updated portfolio with the new name.
4. **deletePortfolio**: Allows deleting a portfolio (and cascade deletes the related investments).
   * **Input**: portfolio\_id
   * **Output**: Deletion of the portfolio and its related investments.

#### C. **Investment Management (Inside a Portfolio)**

1. **createInvestment**: Adds an investment (e.g., a stock) to an existing portfolio.
   * **Input**: investor\_id, portfolio\_id, stock\_id, quantity, purchase\_price
   * **Output**: A new investment added to the specified portfolio.
2. **findInvestmentsByPortfolioId**: Retrieves all investments associated with a particular portfolio.
   * **Input**: portfolio\_id
   * **Output**: List of investments in that portfolio.
3. **updateInvestment**: Allows updating details of an investment in a portfolio (e.g., quantity or purchase price).
   * **Input**: investment\_id, new\_quantity, new\_price
   * **Output**: Updated investment record.
4. **deleteInvestment**: Allows removing an investment from a portfolio.
   * **Input**: investment\_id
   * **Output**: Investment removed from the portfolio.

#### D. **Stock Management (Optional)**

1. **createStock**: Adds a new stock to the system.
   * **Input**: stock\_name, symbol, price
   * **Output**: A new stock entry in the stocks table.
2. **findStockById**: Retrieves stock details by stock\_id.
   * **Input**: stock\_id
   * **Output**: Stock details.
3. **updateStockPrice**: Allows updating the price of an existing stock (useful for price changes).
   * **Input**: stock\_id, new\_price
   * **Output**: Updated stock price.
4. **deleteStock**: Removes a stock from the system.
   * **Input**: stock\_id
   * **Output**: Stock removed from the system.

### 4. ****Business Logic Layer (Service Layer)****

The **Service Layer** will interact with the **Data Access Layer (DAO)**, which performs the actual database operations. The Service Layer will include methods that provide business logic such as validating data, handling exceptions, and ensuring integrity between entities (investors, portfolios, and investments).

#### Key Service Methods:

* **createInvestor**: Validates investor data and calls DAO to persist.
* **createPortfolio**: Checks if the investor exists and if the portfolio name is unique, then calls DAO to create the portfolio.
* **createInvestment**: Ensures that the portfolio exists, checks for valid stock data, and then creates the investment.
* **findPortfoliosByUserId**: Calls the DAO to retrieve all portfolios for a specific investor.
* **findInvestmentsByPortfolioId**: Retrieves all investments under a specific portfolio.
* **deleteInvestment**: Removes an investment from the portfolio and checks for dependencies.

### 5. ****End-User Functionalities****

End users (investors) should be able to perform the following actions:

* **Create multiple portfolios** for different investment strategies (retirement, short-term, etc.).
* **View their portfolios**, including all investments within each portfolio.
* **Add investments** to a portfolio (e.g., buy stocks).
* **Update investment details**, such as increasing or decreasing the number of shares owned.
* **Delete investments** when they sell them.
* **Delete portfolios** when they no longer wish to manage them.

### 6. ****System Flow****

Here’s a simplified flow of how an investor might interact with the system:

1. **Investor signs up**:
   * **Create Investor**: Create a new investor in the system.
2. **Investor creates portfolios**:
   * **Create Portfolio**: Investor can create multiple portfolios like "Retirement Portfolio" or "Short-Term Portfolio".
3. **Investor adds investments to portfolios**:
   * **Create Investment**: Investor buys stocks, bonds, etc., and adds them to specific portfolios.
4. **Investor views portfolios and investments**:
   * **Find Portfolios**: View all portfolios.
   * **Find Investments by Portfolio**: View investments under each portfolio.
5. **Investor updates portfolio or investment**:
   * **Update Portfolio Name**: Change portfolio name (e.g., "Short-Term Portfolio" to "Growth Portfolio").
   * **Update Investment**: Change quantity or price of an investment.
6. **Investor deletes investments or portfolios**:
   * **Delete Investment**: Remove an investment from a portfolio.
   * **Delete Portfolio**: Close a portfolio and remove all related investments.

### 7. ****Exceptions and Error Handling****

* **Portfolio not found**: When trying to access or modify a portfolio that doesn't exist.
* **Investment not found**: When trying to access or modify an investment that doesn't exist.
* **Invalid data**: For invalid quantities, stock IDs, or prices.
* **Foreign Key Constraints**: Ensure that all foreign key references (investor\_id in portfolios, portfolio\_id in investments) are valid.

### Conclusion

This system provides a comprehensive way to manage portfolios, investments, and stocks for multiple users, with a focus on:

* **Portfolio Management**: Multiple portfolios per user.
* **Investment Tracking**: Add, update, and delete investments within portfolios.
* **Flexible Stock Management**: Manage stocks and link them to portfolio investments.

This is a high-level view of the requirements and functionalities without any code.

# Queries for Database Design:

# Create database portfolio\_management;

USE portfolio\_management;

The tables used in application are:

1. **investors**
2. **portfolio**
3. **portfolio\_investments**
4. **stocks**

**CREATE TABLE** queries for tables:

Create database portfolio\_management;

Use database portfolio\_management;

CREATE TABLE investors (

investor\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone VARCHAR(15) UNIQUE NOT NULL,

date\_of\_birth DATE NOT NULL

);

CREATE TABLE portfolio (

portfolio\_id INT AUTO\_INCREMENT PRIMARY KEY,

investor\_id INT NOT NULL,

portfolio\_name VARCHAR(100) NOT NULL,

creation\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (investor\_id) REFERENCES investors(investor\_id) ON DELETE CASCADE

);

CREATE TABLE stocks (

stock\_id INT AUTO\_INCREMENT PRIMARY KEY,

symbol VARCHAR(10) UNIQUE NOT NULL,

name VARCHAR(100) NOT NULL,

price DECIMAL(10,2) NOT NULL

);

CREATE TABLE portfolio\_investments (

investment\_id INT AUTO\_INCREMENT PRIMARY KEY,

portfolio\_id INT NOT NULL,

stock\_id INT NOT NULL,

quantity INT NOT NULL CHECK (quantity > 0),

purchase\_price DECIMAL(10,2) NOT NULL CHECK (purchase\_price > 0),

FOREIGN KEY (portfolio\_id) REFERENCES portfolio(portfolio\_id) ON DELETE CASCADE,

FOREIGN KEY (stock\_id) REFERENCES stocks(stock\_id)

);

### ****Notes:****

* **Auto-incremented primary keys** makes sure unique identification.
* **Foreign keys** maintain relationships for referential integrity.
* **ON DELETE CASCADE** makes sure that deleting an investor deletes their portfolios, and deleting a portfolio deletes its investments.
* **CHECK constraints** prevent invalid data (e.g., quantity and purchase price must be positive).