Introduction

This project aims to develop a portfolio management tool for financial advisors and investment managers. This application integrates an Access database, Excel frontend, and VBA middleware to provide a comprehensive solution for effectively managing and analysing investors' asset allocations. The tool supports the management of various asset types, including stocks, bonds, real estate, etc., making investment management more systematic.

Database

The database structure consists of three main tables: AssetTypes, Investors, and Portfolios.

The AssetTypes table defines categories such as stocks, bonds, and six other asset types. This categorization provides a foundational classification for portfolio management, allowing investment strategies to be tailored according to asset types. The Investors table stores information about investors, such as their ID, name, and email address. This information allows for independent tracking and management of investment activities and facilitates ongoing communication. The Portfolios table documents detailed investment activities per investor, covering asset type, amount, timing, and expected returns. This table serves as a connection between investors and asset types, enabling detailed investment analysis and management.

Regarding queries, the system integrates three main queries.

The AssetSummary query summarizes average returns and total investments by asset type, helping managers identify performance trends and optimize asset allocation. The InvestorAssetDiversity query shows how investments are spread across asset types for each investor, supporting diversification strategies and customized financial advice. The InvestorSummary query provides an overview of each investor's total investment and average returns, aiding in quick evaluation of investment performance and facilitating proactive management.

Front-end

1. “InvestDataSummary” Worksheet

This worksheet serves as the central hub for data management and analysis. Equipped with six functional buttons, it streamlines tasks from data import to cleanup and consolidation, facilitating efficient handling of extensive investment data.

“Import Data Into this Workbook” button directly imports data from the Access database, retrieving data from key tables—AssetTypes, Investors, and Portfolios—and aligns results from queries such as AssetSummary, InvestorAssetDiversity, and InvestorSummary. This data is stored in Excel sheets named after the corresponding Access tables and queries, ensuring consistency and easing further processing and analysis.

“Clear Data In this Worksheet” button clears all data contents in the current worksheet to prevent duplicates and maintain accuracy during new imports. Additionally, data copy buttons re-import data from designated worksheets into this worksheet for in-depth analysis, consolidating data to aid in processing and decision-making.

For data visualization, conditional formatting button highlights critical data, such as the top 10% of investment amounts and investors with returns over 5%. This enables managers to better allocate resources like capital, time, and manpower towards the most lucrative investments, thus boosting the efficiency and profitability of the overall investment portfolio.

2. “InvestorDetails” Worksheet

In this worksheet, users can view all investment details of a selected investor by clicking the button "Choose an Investor and See the details of his/her investments" and then entering the InvestorID in the prompted input box. Once an ID is selected, the top part of the worksheet in grey displays all related investment data for that investor. Investment managers can quickly and intuitively access and analyse an individual investor's portfolio, focusing on each investor's specific circumstances to monitor investment performance and make data-driven decisions, thereby providing investment advice.

Additionally, using pivot tables and slicers enhances the visual accessibility of investment details per investor. For instance, selecting InvestorID 4, Liam Johnson, shows in a pivot table his investments like £70,000 in the Vanguard 500 Index Fund in 2024 with a 7% return, and £10,000 in Brent Crude Oil Futures in 2025 with a 6% return. His total investments amount to £80,000 across two products, averaging a 6.5% return. This suggests he balances risks and returns effectively, aiding investment managers in devising strategies that combine high-return and low-risk products, and in developing diversified asset allocation plans.

3. “New Investments” Worksheet

This worksheet is specially designed for entering new portfolio data and features stringent field validations to minimize entry errors. Fields "InvestorName," "EmailAddress," and "ProductName" require non-empty text to ensure detailed records; notably, "EmailAddress" must include both an "@" and a ".", ensuring accurate identification for future contacts. "InvestorID" needs a positive integer, "AssetTypeID" is chosen from a dropdown list corresponding to specific asset types, and "InvestmentAmount" must be a positive number. "InvestDate" must adhere to the "dd/mm/yyyy" format, and "YieldRate" is capped at values below 100% to accurately reflect the rate of return on investments, facilitating efficient and precise portfolio management.

After data entry, the "Update Database" button at the top right automatically transmits the new data to the backend Access database. This button triggers the "UpdateAccessWithNewData" VBA subroutine, which further validates the data and then updates the "Investors" and "Portfolios" tables, ensuring seamless integration between the worksheet and the database.

Additionally, the usage instructions on the right side of the worksheet provide clear definitions and input rules for each field, such as the mapping of "AssetTypeID" to specific asset categories and the requirement for "YieldRate" to be entered as a percentage. This layout aids user understanding and significantly minimizes operational errors.

VBA Middleware

As the codes are very complex, only a small number of codes are showed in this report.

1. ImportDataModule

This module supports “Import Data In this Workbook” button.

Initially, the program scans all worksheets in the current workbook and automatically deletes any previously imported ones with the same names using the Worksheets.Delete method.

Subsequently, the program uses ADO to establish a connection with the Access database. It dynamically creates the database file path by combining the current workbook path and the database filename, allowing it to operate effectively in any environment.

```
DbPath = ThisWorkbook.Path & "\Database_Investment Portfolios.accdb"
provider = "Provider=Microsoft.ACE.OLEDB.12.0;"

Set MyDBConnection = New ADODB.connection
MyDBConnection.ConnectionString = provider & "Data Source=" & DbPath & ";"
MyDBConnection.Open
```

Then after defining the MySQLTables() and MySQLInstructions() array, this dynamic database connection sequentially reads tables and queries, writing their results into new worksheet. Data import is executed using a SELECT * query, with data entered row by row into worksheets named after each table or query.

```
For i = LBound(TableNames) To UBound(TableNames)
    Set ws = ThisWorkbook.Worksheets.Add
    ws.Name = TableNames(i)
    ws.Cells.Clear

    Set Results = MyDBConnection.Execute(MySQLTables(i))
    OutputResults ws, Results
    Results.Close
Next i
```

```
For i = LBound(QueryNames) To UBound(QueryNames)
    Set ws = ThisWorkbook.Worksheets.Add
    ws.Name = QueryNames(i)
    ws.Cells.Clear

    Set Results = MyDBConnection.Execute(MySQLInstructions(i))
    OutputResults ws, Results
    Results.Close
Next i
```

During this process, OutputResults is a crucial subroutine, which first outputs column names and then proceeds to output records row by row.

2. SummarizeDataModule

This module's function is to consolidate and organise data scattered across different worksheets. Initially, the ClearSummaryData subroutine supports “Clear Data In this Worksheet” button. It precisely removes data from columns A to O starting from the second row, preserving header integrity and preventing data overlay.

Next, data copy buttons are supported by CopyPortfoliosToMainSheet, UpdateSummaryFromInvestorSummary, UpdateAssetDetails, and UpdateAssetTypeDiversity subroutines. Some processes could be accomplished using the XLOOKUP function, but to make the operation more intuitive, code is used instead. For example, in UpdateSummaryFromInvestorSummary, nested loops are used to match the InvestorID and import investor information into H to K of this summary worksheet.

```
For i = 2 To lastRowSummary
    For j = 2 To lastRowInvestorSummary
        If wsSummary.Cells(i, "B").Value = wsInvestorSummary.Cells(j, "A").Value Then 'check whether the InvestorIDs are matched
            wsSummary.Cells(i, "H").Value = wsInvestorSummary.Cells(j, "B").Value
            wsSummary.Cells(i, "I").Value = wsInvestorSummary.Cells(j, "C").Value
            wsSummary.Cells(i, "J").Value = wsInvestorSummary.Cells(j, "D").Value
            wsSummary.Cells(i, "K").Value = wsInvestorSummary.Cells(j, "E").Value
            Exit For 'Exit the inner loop once a match is found to avoid redundancy
        End If
    Next j
Next i
```

Finally, the HighlightHighYieldAndTopInvestors subroutine assists users in identifying investors in the top 10% of total investment and those with an Average Yield Rate over 5%, marking them in green and blue, respectively. Instead of using Excel’s built-in tools, this function is also implemented via VBA code to enhance user operation.

3. SpecificInvestorModule

This module facilitates the button "Choose an Investor and See the details of his/her investments".

The following code enables an input box feature:

```
investorID = Application.InputBox("Please enter the InvestorID:", "Input InvestorID", Type:=1)
If investorID = False Then
    MsgBox "Operation cancelled by the user.", vbInformation
    Exit Sub
End If
```

Here, Application.InputBox provides the interface for user input and restricts the input to

numerical values using Type:=-1. If the user cancels the input, the subroutine exits with Exit Sub to prevent any unnecessary operations.

To prevent mixing of new and old data, code also clears old data from the "InvestorDetails" worksheet. Moreover, it loops through the "InvestDataSummary" worksheet to find matches for the entered InvestorID:

```
For i = 2 To lastRowSummary
    If wsSummary.Cells(i, 2).Value = investorID Then 'Check if the value in column B of the current row equals the user-entered investorID
        For j = 1 To 14
        wsDetails.Cells(outputRow, j).Value = wsSummary.Cells(i, j + 1).Value
        Next j
        outputRow = outputRow + 1
    End If
Next i
```

If a match is found, the Resize method copies the entire row to the next empty row in "InvestorDetails". Lastly, a MsgBox provides feedback once data export is complete, enhancing user experience.

4. UpdateDatabaseModule

This module supports the “Update Database” button. It starts by connecting to the Access database and loading existing investor information to quickly verify whether investor IDs from the "NewInvestments" Excel table already exist in the database. An ExistsInCollection function is used; if an investor ID is not found, it adds new investor information to the "Investors" table.

Additionally, the module loads all records from the "Portfolios" table and generates a unique key for each feature. This key checks if new entries from "NewInvestments" already exist in "Portfolios." If not found, they are added; if found, duplicates are not added to prevent redundancy.

```
For i = 2 To lastRow
    If Not ExistsInCollection(existingInvestors, CStr(ws.Cells(i, 1).Value)) Then  'update investors table when the key doesn't include in the collection
        SQL = "INSERT INTO Investors (InvestorID, InvestorName, EmailAddress) VALUES (" & _
              ws.Cells(i, 1).Value & ", '" & ws.Cells(i, 2).Value & "', '" & ws.Cells(i, 3).Value & "')"
        connection.Execute SQL
    End If

    uniqueKey = Join(Array(ws.Cells(i, 1).Value, ws.Cells(i, 4).Value, ws.Cells(i, 5).Value, Format(ws.Cells(i, 6).Value, "dd/mm/yyyy"), ws.Cells(i, 7).Value, ws.Cells(i, 8).Value)
    If Not ExistsInCollection(existingPortfolios, uniqueKey) Then 'update portfolios table
        SQL = "INSERT INTO Portfolios (InvestorID, AssetTypeID, InvestmentAmount, InvestDate, ProductName, YieldRate) VALUES (" & _
              ws.Cells(i, 1).Value & ", " & ws.Cells(i, 4).Value & ", " & ws.Cells(i, 5).Value & ", #" & ws.Cells(i, 6).Value & "#, '" & _
              ws.Cells(i, 7).Value & "', " & ws.Cells(i, 8).Value & ")"
        connection.Execute SQL
    Else
        duplicateFound = True  'Only exactly identical records are flagged as duplicates
    End If
Next i
```

If duplicates are detected at any stage, a MsgBox alerts the user, preventing insertion and enhancing user understanding and trust in the system’s operations.

Conclusion

This portfolio management tool demonstrates significant potential for expansion. As the business grows, the tool can be further developed to support more users and complex business needs. All designs and implementations consider the demands of real-world business environments, with potential future enhancements including additional asset types and investment strategies to provide richer data analysis capabilities. This project improves the efficiency of financial advisors and investment managers. Also, it provides data-driven support for investment decision-making. For further details and collaboration, the project is hosted on GitHub, accessible at [https://github.com/sylvialiu1216/MTH785-Coursework-2].

Reference

RisingWave Labs. (2025). *Awesome Stream Processing*. Available at: https://github.com/risingwavelabs/awesome-stream-processing