Criterion C: Development

List of Techniques Utilized:

- Adding/Inputting data into database
- Deleting/Removing data from database
- Querying/Searching for specific data in a database
- Merging two or more data structures/Joining databases
- Recursion
- Parsing a CSV file
- 2D Arrays
- Additional libraries
- Creating tables in database
- Exception handling

NOTE: There are 2 tables, spending and revenue. Subroutines serve the same purpose. Spending has an extra column with GST. Spending subroutines would access from the spending table and revenue subroutines would access from the revenue table.

Adding/Inputting data into database

This shows how the program takes inputs from the user and inserts it into the database file, which will update the database with new information.

```
def askRevData():
    """
    Asks the user for the revenue data that is about to be inputted
    :return: Array
    """
    ENTRY = input("Entry/Context: ")
    YEAR = input("Year: ")
    YEAR = checkInt(YEAR)
    CATEGORY = input("Category Payment/Other: ")
    TRANSACTION = input("Transaction: ")
    AMOUNT = input("Amount: ")
    AMOUNT = checkFloat(AMOUNT)
    REV = [ENTRY, YEAR, CATEGORY, TRANSACTION, AMOUNT]
    return REV
```

The following image above shows the revenue subroutine called upon to get the user inputs for adding a member into the revenue database. After the inputs are all done, the following information is appended to an array that returns the array to be added to the database. The year requests for an integer, the amount requests for a float, and the transaction requires some written text. To check these values, I have implemented a **recursive function**.

```
def askSpendData():
    """
    Asks the user for the information they want to add to the spendings database
    :return: array
    """
    ENTRY = input("Entry (A brief word regarding the context of the revenue): ")
    YEAR = input("Year (Integer): ")
    YEAR = checkInt(YEAR)
    TRANSACTION = input("Transaction (Type of transaction): ")
    checkTransaction(TRANSACTION)
    CATEGORY = input("Category (Donation, Payment, Fee, or other): ")
    AMOUNT = input("Amount (Amount lost before taxes): ")
    AMOUNT = checkFloat(AMOUNT)
    GST = AMOUNT * 0.05
    SPEND = [ENTRY,YEAR, TRANSACTION, CATEGORY, AMOUNT, GST]
    return SPEND
```

The subroutine is the same as the previous one, but calculates the GST based on the amount, and since my client's business is located in Alberta, the GST amount is 5%, which is calculated and appended into the array as well.

```
def addRevData(INFO):
    """
    Adds the revenue data into the table in sequel
    :param INFO: array
    :return: none
    """
    global CURSOR, CONNECTION
    CURSOR.execute("""
        INSERT INTO
            revenue (
                Entry,
                Year,
                Category,
                Trans_action,
                Amount
                )
        VALUES (
            ?, ?, ?, ?, ?
            )
    ;""", INFO)
    CONNECTION.commit()
```

The following image shows the adding of revenue data into the database, this subroutine takes the array of user inputs from the previous subroutine and adds it to the database. When inserting the info, no primary key integer is added, so the system will automatically add a primary key that is one higher than the previous primary key.

```
def addSpendData(INFO):
    """
    Adds the data into the spendings table
    :param INFO: array
    :return: none
    """
    global CURSOR, CONNECTION
    CURSOR.execute("""
            INSERT INTO
                spendings (
                    Entry,
                    Year,
                    Category,
                    Trans_action,
                    Amount,
                    GST
                    )
            VALUES (
                ?, ?, ?, ?, ?, ?
                )
        ;""", INFO)
    CONNECTION.commit()
```

Spendings but with a GST column.

Recursion

```
def checkInt(NUM):
    """
    A recursive function that checks the added input is an int
    :param: NUM: str
    :return: int
    """
    try:
        NUM = int(NUM)
        return NUM
    except ValueError:
        print("Please enter a possible value")
        NEW_NUM = input("> ")
        return checkInt(NEW_NUM)
```

The image shown above shows a recursive function that checks for the “Year” input as an integer. As soon as the year input is taken, the “checkInt” subroutine is called upon, which checks if the input is an integer or not. If it is, the user input type is changed to integer and is returned, and if not, the user is asked to input the year again. This subroutine also implements the usage of **exception handling**, which employs the method of trying to convert a string into an integer, and if a “ValueError” were to occur, it would request another new user input and try to convert it to integer again.

```
def checkFloat(NUM):
    """
    A recursive function that checks the added input is a float
    :param NUM: str
    :return: float
    """
    try:
        NUM = float(NUM)
    except ValueError:
        print("Please enter a possible value")
        NEW_NUM = input("> ")
        return checkFloat(NEW_NUM)
    return NUM

def checkTransaction(NUM):
    """
    A recursive function that checks if the transaction input is null, and if it is, the function will ask for a proper input.
    :param NUM: str
    :return: str
    """
    if NUM == "":
        print("Type of transaction must be identified, please fill out this area")
        NEW_TRANSACTION = input("> ")
        return checkTransaction((NEW_TRANSACTION))
    else:
        return NUM
```

Likewise, the “checkFloat” function and the “checkTransaction” subroutine performs the same tasks as the “checkInt” subroutine, but “checkFloat” tries to convert the input into float types, and “checkTransaction” check for the input to be NOT NULL.

<u>Deleting/Removing Data from a database</u>

```
127 def askRevId():
128     """
129     Displays all revenue data to the user and lets the user choose a primary key to update
130     :return: int -- > Primary Key
131     """
132     global CURSOR, CONNECTION
133     INFO = CURSOR.execute("""
134         SELECT
135             id,
136             Entry,
137             Year
138         FROM
139             revenue
140         ORDER BY
141             Year DESC
142     ;""").fetchall()
143     HEADER = ["id", "Entry", "Year"]
144     print("Please select a id")
145     print(tabulate(INFO,HEADER,tablefmt="fancy_outline"))
146     CHOICE = input("> ")
147     ID = []
148     if CHOICE.isnumeric():
149         CHOICE = int(CHOICE)
150     else:
151         print("Please enter a number")
152         return askRevId()
153     for i in range(len(INFO)):
154         ID.append(INFO[i][0])
155     if CHOICE not in ID:
156         print("Please enter a possible number")
157         return askRevId()
158     else:
159         return CHOICE
```

The following image above shows a subroutine giving the user the current data existing. The program selects the id, Entry, and Year columns of the tables, and orders them from most recent to oldest in terms of “Year”, and then uses “tabulet” to display the information (line 133-145). The program then asks the user to input an id, and error-checking is implemented as the program checks if the input “isnumeric()” which verifies the input as an integer. If it is not, it will return the entire subroutine again, but if it is an integer, it will return the input as an integer (line 146-159).

```
536 def deleteRev(ID):
537     """
538     Deletes the row of data that the user has chosen to delete
539     :param ID: int -- > primary key
540     :return: none
541     """
542     global CURSOR, CONNECTION
543     CURSOR.execute("""
544         DELETE FROM
545             revenue
546         WHERE
547             id = ?
548     """, [ID])
549     CONNECTION.commit()
```

The above image shows the deletion of a row of data from the database. This subroutine uses the returned value from the previous subroutine (the ID) and deletes the row with that current existing ID.

Querying/Searching for specific data in a database

This shows how the program will find a specific set of data based on the inputted year from the user.

```
264 def askSpendYr():
265     """
266     Asks for the year of the data the user wants to see
267     :return: int
268     """
269     print("Input the year of the data:")
270     YR = input("> ")
271     try:
272         YR = int(YR)
273     except ValueError:
274         print("Please input a valid number")
275         return askSpendYr()
276     return YR
```

The image above shows a subroutine that requests a year's input from the user.

```
458 def queryRev(YR):
459     """
460     Searches for the info in the database that contains that year
461     :param YR: int
462     :return: 2d array
463     """
464     global CURSOR, CONNECTION
465     QUERY = CURSOR.execute("""
466         SELECT
467             *
468         FROM
469             revenue
470         WHERE
471             Year = ?
472     ;""",[YR]).fetchall()
473     return QUERY
```

The image above shows a subroutine that takes the inputted year from the previous subroutine and uses it to find all existing data in a certain year. When ".fetchall()" is used, a **2D Array** containing all current existing data on the inputted year is created. If there is no data, "None" is returned, otherwise the array is returned.

<u>Parsing a CSV File</u>

```
350 def getValues(FILENAME):
351     """
352     Extracts contents of file and put it into 2d array
353     :param FILENAME: str
354     :return: 2d array
355     """
356     FILE = open(FILENAME,'r', encoding='utf-8')
357     TEXT_LIST = FILE.readlines()
358     FILE.close()
359     for i in range(len(TEXT_LIST)):
360         if TEXT_LIST[i][-1] == "\n":
361             TEXT_LIST[i] = TEXT_LIST[i][:-1]
362         TEXT_LIST[i] = TEXT_LIST[i].split(',')
363         for j in range(len(TEXT_LIST[i])):
364             try:
365                 TEXT_LIST[i][j] = float(TEXT_LIST[i][j])
366             except ValueError:
367                 pass
368     return TEXT_LIST
```

First, the CSV file is opened (line 356), and we use the "readlines()" function. This extracts the data into a list, which is then turned into a **2d array.** Then the file is closed. Before returning, data types are turned into float/text to meet the database constraints. When entering the data into the database, the first array in the 2d array is omitted, as it is the index of the data.

Merging two or more data structures/Joining databases

```
805         INFOSPEND = CURSOR.execute("""
806                 SELECT
807                     spendings.Amount,
808                     spendings.GST
809                 FROM
810                     revenue
811                 JOIN
812                     spendings
813                 ON
814                     revenue.Year = spendings.Year
815                 WHERE
816                     revenue.Year = ?
817             ;""", [YR]).fetchall()
```

```
826         NEWINFOREV = []
827         for i in range(len(INFOREV)):
828             NEWINFOREV.append(INFOREV[i][0])
829         NEWINFOREV = sum(NEWINFOREV)
830         if INFOSPEND == []:
831             NEWINFOSPEND = 0
832         else:
833             NEWINFOSPEND = []
834             NUM = int(len(INFOSPEND)/len(REVENUENUMBER))
835             for i in range(NUM):
836                 for j in range(2):
837                     NEWINFOSPEND.append(INFOSPEND[i][j])
838             NEWINFOSPEND = sum(NEWINFOSPEND)
839         ANSWER = NEWINFOREV - NEWINFOSPEND
840         ANSWER = round(ANSWER, 2)
```

The databases “revenue” and “spendings” are **joined** at Year. Spending amount and GST are selected. The array is then added together and subtracted from the revenue amount to find the total profits. It is rounded to the nearest hundredth.

2D Arrays

```
571 def configureRev():
572     """
573     Configures the data for revenue, finds the different years, and finds the total from each year and puts it into an array.
574     :return: 2d array
575     """
576     global CURSOR, CONNECTION
577     INFO = CURSOR.execute("""
578         SELECT
579             Year
580         FROM
581             revenue
582     ;""").fetchall()
583     NEWINFO = []
584     for k in range(len(INFO)):
585         for j in range(len(INFO)):
586             if INFO[k][0] == INFO[j][0]:
587                 if INFO[j][0] in NEWINFO:
588                     pass
589                 else:
590                     NEWINFO.append(INFO[j][0])
591     GRAPH = []
592     for i in range(len(NEWINFO)):
593         TOTAL = CURSOR.execute("""
594             SELECT
595                 Amount
596             FROM
597                 revenue
598             WHERE
599                 Year = ?
600             ;""",[NEWINFO[i]]).fetchall()
601         NEWTOTAL = []
602         for l in range((len(TOTAL))):
603             NEWTOTAL.append(TOTAL[l][0])
604         NEWTOTAL = sum(NEWTOTAL)
605         GRAPH.append([NEWINFO[i], NEWTOTAL])
606     return GRAPH
```

When"fetchall()" is used (line 582), a **2D array** is created with all the years in the table. This 2d array is then traversered into a new array, where the new array has only one unique year for each respective year in the dataset. "fetchall()" (line 600) gets the "amount" as a **2d array** (in that respective year), which is then traversed into a new array and added together. The year and total amount is then appended to the final array. This is done in a **for loop** to cover all unique years. It is returned to be graphed.

Tables creation in databases

```
369 def setupRevenue(REVENUE):
370     """
371     Sets up the tables of revenue
372     :param REVENUE: 2d array
373     :return: none
374     """
375     global CURSOR, CONNECTION
376     CURSOR.execute("""
377         CREATE TABLE
378             revenue (
379                 id INTEGER PRIMARY KEY,
380                 Entry TEXT,
381                 Year INTEGER NOT NULL,
382                 Category TEXT,
383                 Trans_action TEXT NOT NULL,
384                 Amount REAL NOT NULL
385             )
386     ;""")
387     for i in range(1, len(REVENUE)):
388         CURSOR.execute("""
389             INSERT INTO
390                 revenue (
391                     Entry,
392                     Year,
393                     Category,
394                     Trans_action,
395                     Amount
396                 )
397             VALUES (
398                 ?, ?, ?, ?, ?
399             )
400         ;""", REVENUE[i])
401     CONNECTION.commit()
```

This shows the creation of tables in the database. As the client is recording their money distribution, the year/time and amount must be “NOT NULL”, to ensure that each of those columns are filled out.

Additional Libraries

```
6  import sqlite3
7  import pathlib
8  import matplotlib.pyplot as plt
9  from tabulate import tabulate
```

“Sqlite3” was imported for the creation and handling of databases efficiently.

“Pathlib” was imported to detect if the file connecting to the database already exists, if yes, the database file would not be created, otherwise the file would be created.

“Matplotlib” was imported to allow the graphing of data, which provides a visual representation of the data, and user-friendly experience.

“Tabulet” was imported to create a user-friendly experience and provide nicely designed tables for my client.

Word Count: 99