CSCI 465 **COBOL and Db2 Assignment**
200 points

Overview

This assignment will be similar to the COBOL Tables assignment we did earlier this semester, except for a key difference. The JCL needed to compile the COBOL and Db2, create a load module for each, bind them together, and execute the program will be provided to you. You will only need to write the source code for your program. The source code should be similar to what you've done in the COBOL Tables assignment. However, instead of reading the number and name of each mutual fund and daily closing prices associated with fund from a partitioned data set, you will read the data from a Db2 database. **Your output report should be identical to your output in the COBOL Tables assignment**. You will still need to read in the broker deposit and sales file from a PDS member, the name of which is to be provided.

Before You Start

You will need to have the following set up in order to properly compile and run your program. They will need to be in the same location as stated below:

- Source Code Library
    - ***YourKCID*.CSCI465.SRCLIB**
    - Allocate it the following way:
        - Space Units: TRACK
        - Primary: 5
        - Secondary: 10
        - Record Format: FB
        - Record Length: 100
        - Block Size: 1000
        - Data set name type: PDS

- Db2 Compilation Library
    - ***YourKCID*.CSCI465.DBRMLIB**
    - Allocate it the following way:
        - Space Units: TRACK
        - Primary: 5
        - Secondary: 10
        - Record Format: FB
        - Record Length: 100
        - Block Size: 1000
        - Data set name type: PDS

- Load Module Library
    - ***YourKCID.CSCI465.LOAD***
    - Allocate it the following way:
        - Space Units: TRACK
        - Primary: 5

- Secondary: 10
- Record Format: U
- Record Length: 100
- Block Size: 1000
- Data set name type: PDS

REMEMBER: Your data sets must be in the exact same locations in order to properly compile and run your program.

Execution

Because you are not writing any JCL, it will be stored in a publicly accessible data set at `KC02322.CSCI465.JCL(ASSIGNn)`.

In order for your program to run, your source code data set member must be named `ASSIGNn`. You can issue the following command from the terminal to execute your program:

```
TSO SUB 'KC02322.CSCI465.JCL(ASSIGNn)'
```

When running successfully, your program will have a return code of 4. This is because of an informational message that is printed out during the compilation and binding steps.

WORKING-STORAGE Section

You will need to include 3 different Db2 copy libraries, in order to complete this assignment. The first is a set of SQL COBOL variables that will be necessary to check the `SQLCODE` (return code) of any Db2 statement. The last two are the COBOL variables to use to store values when you query the database. You will need to use these variables when you initially read in the data, as the COBOL variable format needs to match the Db2 database. It is recommended that after you read in your data, you move the values to COBOL variables that you have defined yourself, like moving data from a buffer to local storage.

The name of the copy libraries are as follows:

1. `SQLCA`
2. `FUND`
3. `FUNDPRC`

In order to include one of these libraries, you **must** use the following format in the beginning of your `WORKING-STORAGE` Section:

```
EXEC SQL
     INCLUDE libraryname
END-EXEC.
```

The Program

You should first read in your date and time variables using the COBOL intrinsic function, just as you have in previous assignments. Then, open your output data set, and the broker and sales PDS. Process the broker and sales PDS in the same fashion as you have in the COBOL Tables assignment (read loop).

For each record representing a broker's sales for the business day, you must first move his or her city name and broker name to the detail line of the MUTUAL FUND DEPOSIT AND SALES REPORT.

You will then parse each of the four 10-byte fund sales sets of data. For each that has a fund number greater than 0, you will need to query the FUND table to get the fund name. Then you check the SQLCODE to see it was found in the table or not. If the **SQLCODE = 100** then there were no matching records returned by the query (aka it was not found). If a match is found, move the fund number and the fund name from the Mutual Fund Table to the detail line following the broker's name. Note that the broker's region, city, and name should only appear on the first detail line for his or her first, or perhaps only, sale of the day.

Then, move the deposit amount to the detail line. Next, move the number 1 to 5 representing the daily share price used to calculate the number of shares to the detail line. You will need to query the FUND_PRICE table in order to retrieve the fund price so you can calculate the number of shares. As you should always do, make sure the query was successful by checking if SQLCODE is not 100. Finally, move the calculated number of shares to the detail line. You will not need to compute or report broker commission for this report.

Count the number of brokers reported and the grand total of all of the deposits as you process the sales file.

The Mutual Fund Deposit and Sales File

Each record of the broker sales file represents a one-dimensional table. This means that there will be a one-dimensional table definition under the FD for this file.

At the beginning of each sales record are the city name and broker name. The following was taken directly from the COPYLIB member SALESREC:

```
01  SALES-RECORD.
    05  IN-CITY-NAME           PIC X(20).
    05  IN-BROKER-NAME         PIC X(20).
    05  IN-MF-SALE             OCCURS 4.
        10  IN-FUND-NBR        PIC 9(2).
        10  IN-PRICE-FLAG      PIC 9.
        10  IN-DEPOSIT-AMT     PIC 9(5)V99.
```

Following the broker name is where the one-dimensional table that is built into the input sales record is found. Although a bit unrealistic but to simplify things, a broker can only sell up to four different mutual funds per day.

Along with the city name and broker name, adding up to 40 bytes, these four 10-byte sets of fund sales information make up 80 bytes total. Please note that the deposit amount is now only seven bytes long. Also note that one or more of these four fund sales sets of data may contain all zeros if the broker sells fewer than four mutual funds during the day. If the fund number is set to zoned decimal zeros, you can ignore the rest of the data on the record and move on to the next sales record in the file.

Details about the Report (The same as the COBOL Tables assignment)

1. Center the titles under one another at the top of each page, including the totals pages. This should include the name of the firm as used in previous assignments as the title and 'MUTUAL FUND DEPOSIT AND SALES REPORT' as a subtitle.

2. Each page should have the date in `MM/DD/YYYY` format at the far left of the first header line and '`PAGE:` ' with page number in format `ZZ9` at the far right of the first header line.
3. Each page should have the time in `HH:MM:SS` format at the far left of the second header line.
4. Double-spaced after the second of the page headers, or titles, should be an appropriate column header (which can be two lines if necessary) and, single-spaced after that, a line of appropriately placed hyphens.
5. Maximum of 16 double-spaced broker detail lines per page. It is permissible to page break while listing a broker's four possible sales for the day.
6. Use the full 132 bytes available to spread out your headers and detail lines.
7. All arithmetic calculations for dollars and cents should be rounded to two decimal places. All for share amounts should be rounded to three decimal places with a fourth decimal place set to 0 when displayed.
8. All numeric-edited fields should be correctly edited with a floating dollar sign (when dollars and cents), commas between the thousands, and decimal points.
9. Remember to ONLY print the broker's region, city name, and broker name on the first of the four possible detail lines reporting his or her daily sale(s).

If the Db2 query for a matching mutual fund in the Mutual Fund Table returns a not found condition, move just the numeric-edited mutual fund to the detail line followed by 'MUTUAL FUND NOT FOUND IN TABLE!' followed by a numeric-edited display of the mutual fund number not found to the detail line beginning where the fund number from a good sales transaction would be displayed. Display NOTHING else on the detail line following this information. Note that you SHOULD add the deposit amount to the grand total of deposits in spite of not being able to find a matching fund.

At the end of this report and at the top of a new page, double space and center a third title of 'MUTUAL FUND SALES TOTALS'. Double-spaced after this third header, center three column headers and, single-spaced after that, a line of appropriately placed hyphens. Double-spaced below this display the number of brokers, the total of all deposits rounded, of course, to two decimal places, and the average deposit per broker reported rounded, of course, to two decimal places.

Db2 Table Format

When referring to a table in your program, you should precede the table with the table space name, which in this case is `KC02322` (`KC02322.`*`tableName`*). You **SHOULD NOT** declare these tables in working storage manually. Instead, you the copy library to import them, as stated in the beginning of the handout. It is required that you use the COBOL variables laid out below when trying to perform any SQL Db2 statements, or you will receive a compile error in your program.

`KC02322.FUND` Table

Db2 Format

```
FUND_NBR   INTEGER NOT NULL,
FUND_NAME  VARCHAR(25)
```

COBOL Format

```
01  DCLFUND.
    10 FND-FUND-NBR              PIC S9(9) USAGE COMP.
    10 FND-FUND-NAME.
       49 FND-FUND-NAME-LEN      PIC S9(4) USAGE COMP.
```

```
       49 FND-FUND-NAME-TEXT     PIC X(25).
```

KC02322.FUND_PRC Table

Db2 Format

```
FUND_NBR   INTEGER NOT NULL,
FUND_FLAG  INTEGER NOT NULL,
FUND_PRC   DECIMAL(5, 2)
```

COBOL Format

```
01  DCLFUND-PRICE.
    10 PRC-FUND-NBR          PIC S9(9) USAGE COMP
    10 PRC-FUND-FLAG         PIC S9(9) USAGE COMP.
    10 PRC-FUND-PRC          PIC S9(3)V9(2) USAGE COMP-3.
```

SQL Notes

- **EVERY** SQL statement embedded in COBOL is done inside what is known as an `EXEC-SQL` block.

```
EXEC-SQL
     SELECT TABLE_COLUMN
     INTO :COBOL-VARIABLE
     FROM KC02322.MY_TABLE
END-EXEC.
```

- When you need to reference a COBOL variable inside an `EXEC-SQL` block, you precede the variable name with a **`:`** (colon).

```
EXEC-SQL
     SELECT TABLE_COLUMN_1
     INTO :COBOL-VARIABLE_1
     FROM KC02322.MY_TABLE
     WHERE TABLE_COLUMN_2 = :COBOL-VARIABLE-2
END-EXEC.
```

  - Where `COBOL-VARIABLE-1` and `COBOL-VARIABLE-2` are defined in `WORKING-STORAGE`.
  - **These COBOL variables need to match the data type of the Db2 column, so it is important that you use the COBOL variables copied in for use with the tables**.
  - You can move data from and to these fields, and COBOL will handle converting the types properly automatically for you.

- `SQLCODE` will be updated after every SQL statement that is executed, if it is `100`, that means that the query returned no results

(continued)

Other Notes

You MUST use the `COPYLIB` member `SALESREC` in copy library, or "copylib", PDS `KC02322.CSCI465.COPYLIB` at least once each in your program.

Call the COBOL intrinsic date function only once in your program and NOT inside of a loop.

Submit the .txt file with the above three steps on Blackboard before the time and date at which it is due.