CSCI 465 **Assignment 8 – COBOL and Db2** **Fall 2019**

200 points

Overview

As IBM's Db2 is the one of the world's most prolific relational database management systems (RDBMS), it is important that future enterprise developers have some basic experience interacting with Db2. To that end, we are going to do a final assignment with our last COBOL program accessing the mutual fund name, share price and commission percentage stored in Db2 tables instead of building an internal table of that information.

As an introduction to Db2's most recent versions and capabilities in a modern world, the following article was written for *Forbes Magazine* by Adrian Bridgewater, a senior contributor in the subjects of enterprise and cloud who claims to "…track enterprise software application development and data management". It was published on 4 June 2019:

IBM Injects 'Data Science' AI Into Its Db2 Database

IBM doesn't release a whole lot of news detailing the ins and outs of its Db2 database division. Indeed, database vendors including Oracle, SAP, Microsoft and open source champions such as Apache Cassandra, MongoDB and Redis tend to take up more column spaces in the tech-business press. Each vendor is constantly trying to outstrip the other in terms of its database's supposed ability to scale upwards in size, handle deeper analytics complexity and work increasingly effectively (marcoms people like to say 'seamlessly') on different types of data pipeline jobs, all in different cloud environments.

What brought IBM forward to start talking about Db2 then? As is the case with the lion's share of technology augmentations currently playing out at the end of this decade – it's all about adding Artificial Intelligence (AI) via the Machine Learning (ML) methods that fuel it.

How smart is a database?

Databases are already quite clever, i.e., they can store data in different shapes, formats, vectors, directions and degrees of structured, semi-structured or unstructured rawness. The trouble is, we all depend on them a lot (either directly, or at the backend of the apps we use everyday) and so they now need to get a whole lot smarter. Because of this need for extra smartness, IBM is adding AI into Db2 through data science methods that will inject Machine Learning into it.

"Today, expectations for the database are higher," said Prasun Mahapatra, senior database administrator at Micro Focus, and a Db2 user. "Databases must be smarter. Building out Db2 as the new AI database will enable users to optimize everything from the usage of data structures, memory and disk, to the most complex queries."

What IBM has actually done is to provide built-in support for 'data science' development in the Db2 version 11.5 release. Data science is sometimes thought of as the three-way intersection point between mathematics, technology hacking and programming, and business strategy. Data science is essentially a way of getting computers to deduce and work out more from the data crunching tasks we give them.

How much smarter is an AI/ML database?

So if AI + ML is helping computing systems to get increasingly smarter, how does it get into a database? IBM is building that channel by adding new 'drivers' (software connections that manage interfaces) for multiple open source programming languages and frameworks that will allow developers to analyze and build Machine Learning models into software applications that make use of Db2. In theory, at least, such apps should require less management, be more resilient to outages and help to improve user productivity.

Also new with Db2 11.5 is the Augmented Data Explorer, a new natural language querying feature that is designed to give developers a traditional search engine-like experience. Users can pose questions to Db2 and receive results in data visualizations and summaries written in natural language, for easy understanding. The tool itself is containerized for ease of deployment and management and features dynamic visualization, which can help speed the exploration of datasets when building applications.

“In addition to these advances, Db2 users and developers will be able to take advantage of the new capabilities in IBM’s Data Virtualization technology – already available with IBM Cloud Private for Data. Data Virtualization is designed to enable users to easily search across diverse data sources. With it, developers and data engineers can focus on development, minimizing time spent on Extract, Transform and Load (ETL) processes that are associated with moving data. Db2 now also includes blockchain support that helps application developers pull data directly from a blockchain and combine that data with other data sources for analytics or dashboards,” noted IBM, in a press statement.

With the addition of Db2 11.5, IBM has streamlined the portfolio to three editions that share the same common code base: Db2 Community, Db2 Standard, and Db2 Advanced. IBM Db2 is a no charge (and perpetually free) download for trials and developers and is intended for a single application developer to use to design, build, test and prototype applications for deployment on Db2 client or server platforms.

AI everywhere

Vice president for IBM data and AI Daniel Hernandez has said the changes IBM has made here are ‘reflective’ of IBM’s clients’ growth patterns and the acceleration of AI adoption we all expect. It’s true; barely a day (perhaps an hour) goes past without the technology industry now telling us that it has engineered more AI into every niche of its coalface.

The database used to be just that, i.e., a base of data, a traditional system of record and a data storage repository.

That time has now passed, we expect intelligence every level of the modern technology stack from front-end user interfaces with natural language and speech recognition, throughout our selection of applications and now all the way back to through into our data management layer.

The Assignment

This assignment is similar to Assignment 7except for two key differences: 1) the mutual fund data, prices and commission percentages are stored in Db2 database tables, and 2) the JCL needed to compile and bind the `SALESRPT` program and the Db2 parts of the assignment, assemble and bind the now dynamically-called `CALCVALS` program and then execute the application will be provided to you.

You will only need to write the source code for your programs which is very similar to what you wrote in Assignment 7. You will no longer need the sort step, the mutual fund table and the entire `BUILDTBL` program and process.

Your output report should be identical to your output from Assignment 7. The JCL provided to you will read the broker deposit and sales file from `KC02322.CSCI465.DATA7(SALES)`

Before You Start

You may need to sign into TSO/ISPF using a different area procedure. On the TSO/E screen where you enter your password, enter your password and, before pressing Enter, change `PROCEDURE ===>` to:

```
DBPROCCG
```

You will need to have the following set up in order to properly compile and run your programs. **They must be the same data set names as stated below:**

- Source Code Library
    - `KC03nnn.DB2.SRCLIB`
    - Allocate it the following way:
        - Space Units: TRACK
        - Primary: 5
        - Secondary: 10
        - Directory Blocks: 1
        - Record Format: FB
        - Record Length: 80
        - Block Size: 800
        - Data Set Name Type: LIBRARY (PDSE)
- Db2 Compilation Library
    - `KC03nnn.DB2.DBRMLIB`
    - Allocate it the following way:
        - Space Units: TRACK
        - Primary: 5
        - Secondary: 10
        - Directory Blocks: 1
        - Record Format: FB
        - Record Length: 80
        - Block Size: 960
        - Data Set Name Type: LIBRARY (PDSE)
- Load Module Library – Different than your current LOADLIB PDSE!
    - `KC03nnn.DB2.LOAD`
    - Allocate it the following way:
        - Space Units: TRACK
        - Primary: 5
        - Secondary: 10
        - Directory Blocks: 1
        - Record Format: U
        - Record Length: 100
        - Block Size: 1000

- Data Set Name Type: LIBRARY (PDSE)

Remember, your data sets must have the exact names provided above 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.DB2.JCLLIB(ASSIGN8)`. You must store and edit your two programs – named `SALESRPT` and `CALCVALS` as in Assignment 7 – in the source library you allocated above named `KC03nnn.DB2.SRCLIB`. The names of the source library members must match the `PROGRAM-ID` of the COBOL program `SALESRPT` and the `CSECT` of the Assembler program `CALCVALS`. There will be only code stored in these two source library members…NO JCL!

In order for your application to execute, you will issue the following command from the `COMMAND ===>` prompt in any subdirectory of TSO/ISPF:

```
TSO SUB 'KC02322.DB2.JCLLIB(ASSIGN8)'
```

When the program completes, an alert will pop up as a black screen that shows the `MAXCC` (Max Condition Code) of the job. When running successfully, your program will have a `MAXCC` of 4. This is because of a few informational messages that are printed out during the compilation and binding steps. Press Enter to go back to the commant prompt and go to SD.ST to view your output.

Note that the JCL used to submit your program is available for your information in the Assignment 8 folder on Blackboard as `ASSIGN8.txt`. Also available for your information is the JCL and instream COBOL program used to load the two Db2 tables, `KC02322.FUND` and `KC02322.FUND_PRC`. It is also in the Assignment 8 folder on Blackboard as `DB2LOAD.txt`.

WORKING-STORAGE SECTION.

You will need to include 3 different Db2 copy library members in order to complete this assignment. The first is a set of SQL COBOL variables that will be necessary to check the SQL CODE (return code) of any Db2 statement. The last two are the COBOL variables you need 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've defined yourself, similar to moving data from a buffer to local storage.

The name of the DB2 copy library members in `KC02322.DB2.DCLLIB` are as follows:

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

In order to include one or more of these libraries, you **must** use the following format at the **beginning** of your `WORKING-STORAGE SECTION`:

```
EXEC SQL
   INCLUDE membername
END-EXEC.
```

The Program

You will no longer need the `BUILDTBL` program and you will no longer need the `FUND-TBL` defined in the main program. Instead, the information stored in the `FUND-TBL` will now be stored in existing Db2 tables. You will be use this Db2 database to get the fund information and price. Process the sales file in the same fashion as you did in Assignment 7.

As before, you will parse the up to three sales per broker. For each of the up to three sales that has a fund number greater than 0, you will need to query the `FUND` Db2 table to get the fund name. Then you check the SQL code to see it was found in the table or not. `SQLCODE = 100` indicates that there were no matching records returned by the query, a.k.a., fund not found. If a match is found, handle it as you did in Assignment 7.

You will also need to query the `FUND_PRC` (Price) Db2 table in order to retrieve the fund price and call `CALCVALS` dynamically. Again, `SQLCODE = 100` indicates that there were no matching records returned by the query, a.k.a., price not found. (Note that it states you will be calling `CALCVALS` dynamically now!)

The Mutual Fund Broker Sales File

As in Assignment 7, each record of the broker sales file represents a one-dimensional table with up to three sales for the day. The following was taken directly from the `KC02322.CSCI465.COPYLIB` member `SALESREC`. This `COPYLIB` is included in the concatenation on the `SYSLIB DD` card for the COBOL Compiler step. The other is the aforementioned `KC02322.DB2.COPYLIB`.

```
01  SALES-RECORD.
    05  SALES-BRANCH-NME          PIC X(25).
    05  SALES-BROKER-NME          PIC X(25).
    05  SALES-FUND-SALE           OCCURS 3
                                  INDEXED BY SALE-NDX.
        10  SALE-FUND-NBR         PIC S9(3).
        10  SALE-DEP-AMT          PIC S9(8)V99.
    05  FILLER                    PIC X(13).
```

The Db2 query for a matching mutual fund in the Mutual Fund Table will return either a found or not found condition. Handle it exactly as you did in Assignment 7.

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, use the `COPY` statement to import them from the `COPY` library 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

<u>Db2 Format</u>
```
FUND_NBR    INTEGER NOT NULL,
FUND_NME    VARCHAR(25)
```

<u>COBOL Format</u>
```
01  DCLFUND.
```

```
    10 FUND-FUND-NBR                PIC S9(9) BINARY.
    10 FUND-FUND-NME.
       49 FUND-FUND-NME-LEN         PIC S9(4) BINARY.
       49 FUND-FUND-NME-TEXT        PIC X(25).
```

`KC02322.FUND_PRC` Table

Db2 Format
```
FUND_NBR      INTEGER NOT NULL,
FUND_PRC      DECIMAL(5, 2),
FUND_COMM_PCT DECIMAL(6, 5)
```

COBOL Format
```
01  DCLFUND-PRC.
    10 PRC-FUND-NBR         PIC S9(9) BINARY.
    10 PRC-FUND-PRC         PIC S9(3)V9(2) PACKED-DECIMAL.
    10 PRC-FUND-COMM-PCT    PIC S9(1)V9(5) PACKED-DECIMAL.
```

SQL Notes

- **EVERY** SQL statement embedded in COBOL is done inside an `EXEC-SQL` block as follows:

  ```
  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) as above and as follows:

  ```
  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 as necessary and automatically for you.

- `SQLCODE` will be updated after every SQL statement that is executed. If `SQLCODE = 100`, the query returned no results

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