BMO Coding Assessment - Solution

Answer 1:

LCTR (Large Cash Transaction Report) is a mandatory report that financial entities in Canada must submit to FINTRAC (Financial Transactions and Reports Analysis Centre of Canada) whenever they receive $10,000 or more in cash in a single transaction or in multiple transactions that occur within a 24-hour period and are clearly connected.

In simple terms, the purpose of the LCTR is to monitor and prevent money laundering and terrorist financing. When a large cash transaction is made, banks and other reporting entities (like casinos, securities dealers, money services businesses, etc.) are required by law to collect details about the customer, the nature of the transaction, and the source of the cash. They then file an LCTR report to FINTRAC.

The report includes:

- Customer details (name, ID, occupation)
- Transaction details (amount, method, institution, time)
- Location and purpose of the transaction

The format of the report is governed by a structured JSON schema, and it must pass a set of validation rules defined by FINTRAC to be accepted.

This reporting requirement ensures that cash flow exceeding regulatory thresholds is transparent and traceable, helping the government detect suspicious activity and maintain financial integrity in Canada.

Answer2:

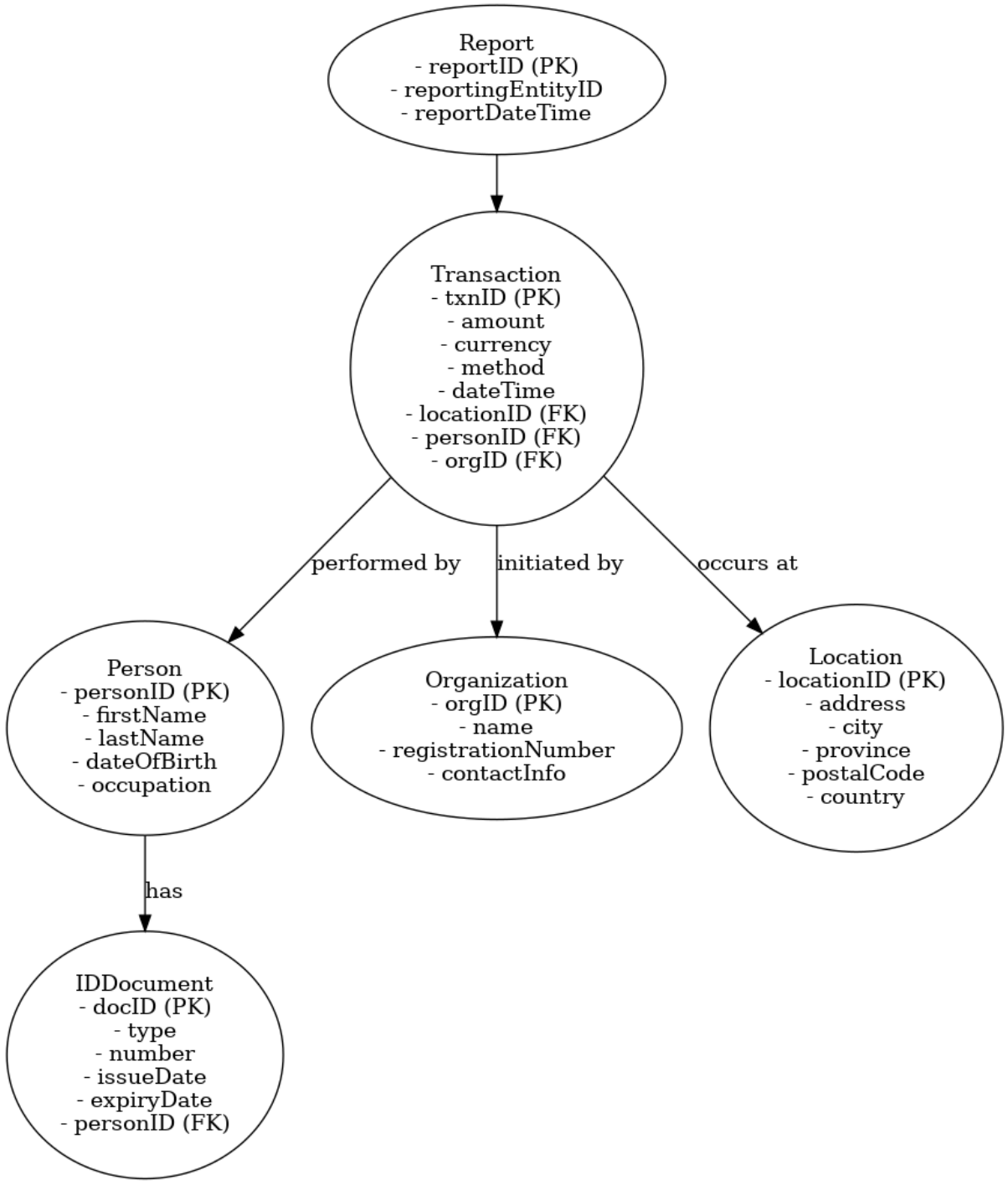
To create the Entity Relationship Diagram (ERD), I reviewed the JSON schema provided for LCTR reporting. The schema defines the structure of a large cash transaction report including its nested objects.

These are the following steps that I undertook to create the ERD:

1) Identified major object groups such as reports, individuals, organizations, transactions, and locations, and treated them as entities.
2) Analyzed how these entities reference each other and mapped their relationships (one-to-many, many-to-one).
3) Created the ERD using an online tool.

The ERD includes the following entities: Report, Transaction, Person, Organization, Location, and IDDocument, with primary and foreign key relationships clearly defined. This

helps in breaking down the nested structure of the JSON into a relational model that can be implemented in any RDBMS like MySQL.



The Entity Relationship Diagram (ERD) translates the nested and hierarchical structure of the LCTR JSON schema into a normalized relational model. Each major object or group in the schema becomes a distinct table (entity), with attributes becoming columns and relationships represented through foreign keys.

This mapping helps:

1. Identify primary keys that uniquely identify records (e.g., reportID, txnID).
2. Define foreign key relationships between tables (e.g., Transaction → Person or Location), enabling JOIN operations.
3. Avoid data duplication by separating repeating groups (e.g., a person can have multiple ID documents).
4. Support efficient querying and data integrity in relational databases like MySQL or PostgreSQL.

Answer3:

I undertook the following steps to flatten the sample LCTR JSON data using python:

1. Pandas and json libraries were used.
2. The JSON file was loaded from a local file (Sample_Data.json), parsed, and normalized using pandas.json_normalize().
3. Nested structures like arrays and dictionaries were flattened into a single table using underscore (_) separators for key paths.
4. The normalized data was printed using .head() for a quick preview.
5. The flattened output was saved as a CSV file named flattened_lctr_data.csv.

Code file: **Question3_answer.py**
Output file: **flattened_lctr_data.csv**

Answer4:

I wrote a C# console application to flatten the nested LCTR JSON data using Newtonsoft.Json and recursive parsing via JToken.

1. The script reads the input JSON file, traverses nested fields, and converts them into a flat key-value structure using dot notation.
2. I chose this method because recursive flattening made it easier to handle unpredictable nesting levels while keeping the output readable.
3. The flattened data is saved into a CSV file (flattened_lctr_data_q4.csv) for further processing.
4. I used Visual Studio Code and the .NET CLI (dotnet run) to develop and execute the program.
5. This approach ensures that the data is normalized and suitable for downstream validation or analytics.

Code file: **Program.cs**
Output file: **flattened_lctr_data_q4.csv**

Answer5:

To implement a relational database from the sample LCTR JSON file, I wrote a Python script that:

1. Parses and extracts relevant data from the JSON file (Sample_Data.json).

2. Separates information into two tables:

    a. Customers: capturing individual details such as name, date of birth, and occupation.
    b. Transactions: capturing transaction amount, currency, transaction date, and location, while referencing the customer via a person_id foreign key.

3. Handles primary and foreign key constraints using SQLite:
    a. person_id as the primary key in the Customers table.
    b. txn_id as the primary key in the Transactions table.

4. Ensures data integrity by:
    a. Removing duplicate entries based on person_id and txn_id.
    b. Printing location_id values for verification.
    c. Clearing existing tables before loading to avoid conflicts.

Code file: **Question5_answer.py**
Output file: **lctr.db**

Answer6:

I took the following steps to count the number of validation rules for each JSON field from the LCTR schema:

1. I used the official validation rules CSV published by FINTRAC.
2. The CSV was loaded and processed using a Python script with the pandas library.
3. I grouped the rules by the Field Id column (which corresponds to JSON field paths).
4. A count was calculated to determine how many rules apply to each field.
5. The grouped output was exported to a CSV file named Validation_rules_count_from_csv.csv.

Code file: **Question6_answer.py**
Output file: **Validation_rules_count_from_csv.csv**

Answer7:

To validate the sample LCTR JSON data, I wrote a Python script that first flattens the nested structure using pandas.json_normalize, and then compares the fields against the validation rules listed in the official FINTRAC CSV.

1. I focused on two types of rules:
2. Presence: Ensuring that all mandatory fields are present and not empty
3. Format: Checking simple data types such as date, numeric, and email fields

Each rule was applied to the corresponding JSON field, and any violations were recorded and saved to a file named validation_errors_sample_data.csv. This file includes a detailed list of validation errors for the sample data.

Code file: **Question7_answer.py**
Output file: **validation_errors_sample_data.csv**

Answer8:

I created a reusable Python script that can validate any LCTR JSON file by dynamically applying FINTRAC's official rules.

1. The script uses pandas.json_normalize to flatten the structure and checks for:
2. Presence: Required fields must not be missing or empty
3. Format: Basic checks for dates, numbers, and email formats
4. Initially, I considered validating nested JSON directly, but I found flattening made rule matching much easier and more scalable.
5. Validation results are saved in a file named like validation_errors_<input_filename>_Q8.csv.

Code file: **Question8_answer.py**
Output file: **validation_errors_Sample_Data_Q8.csv**