Name: Covenant Oludapomola OJO
Course title: Data Mining and Discovery
Student id: 23013682

DESIGNING A REALISTIC BANKING DATABASE USING PYTHON AND SQLITE

BRIEF OVERVIEW

This project focuses on creating a comprehensive banking database using Python-based tools such as Jupyter Notebook, with libraries like Faker, Pandas, and NumPy. The dataset, consisting of 1,000 records, was carefully designed to reflect real-world scenarios by including missing and duplicate data. Key attributes include customer and account details, transaction records, and satisfaction metrics.

The database schema is organized into three relational tables: Customers, Accounts, and Transactions, with primary and foreign keys ensuring data integrity. CSV files generated from the dataset were imported into SQLite for database management.

The project also explores data quality challenges, such as managing missing and duplicate data, and implements measures to simulate realistic banking practices while keeping data privacy.

DATA GENERATION

I used Python (Jupyter Notebook) with libraries such as Faker, Pandas, and NumPy to generate a realistic banking dataset holding 1,000 records. This dataset was designed to reflect real-world scenarios by including both missing and duplicate data, aligning with the assignment requirements.

[1]:
```python
from faker import Faker
import pandas as pd
import numpy as np

# Initialize Faker for generating realistic names
fake = Faker()

# Number of rows to generate
num_rows = 1000

# Generate the data dictionary
data = {
    # Primary Key - Unique identifier for each customer
    'CustomerID': [f'CUST{str(i).zfill(5)}' for i in range(1, num_rows + 1)],

    # AccountID - Unique identifier for each account
    'AccountID': [f'ACC{str(i).zfill(5)}' for i in range(1, num_rows + 1)],

    # TransactionID - Unique identifier for each transaction
    'TransactionID': [f'TRANS{str(i).zfill(5)}' for i in range(1, num_rows + 1)],

    # Customer Name - Nominal data
    'CustomerName': [fake.name() for _ in range(num_rows)],

    # Credit Score - Ordinal data
    'CreditScore': np.random.choice(['Poor', 'Fair', 'Good', 'Very Good', 'Excellent'], num_rows),

    # Satisfaction Score - Interval data
    'SatisfactionScore': np.random.randint(1, 11, num_rows),

    # Account Balance - Ratio data
    'AccountBalance': np.round(np.random.uniform(0, 100000, num_rows), 2),

    # Transaction Frequency - Ordinal data
    'TransactionFrequency': np.random.choice(['Rarely', 'Occasionally', 'Frequently', 'Very Frequently'], num_rows),

    # Account Type - Nominal data
    'AccountType': np.random.choice(['Savings', 'Checking', 'Investment', 'Loan'], num_rows),

    # Last Transaction Date - Interval (date) data
    'LastTransactionDate': pd.to_datetime(
        np.random.choice(pd.date_range("2023-01-01", "2024-01-01"), num_rows)
    )
}

# Convert dictionary to DataFrame
df_banking = pd.DataFrame(data)

# Adding missing values in AccountBalance and SatisfactionScore for realism
df_banking.loc[df_banking.sample(frac=0.05).index, 'AccountBalance'] = np.nan
df_banking.loc[df_banking.sample(frac=0.03).index, 'SatisfactionScore'] = np.nan

# Adding some duplicates to the dataset
duplicates = df_banking.sample(frac=0.01, replace=True)
df_banking = pd.concat([df_banking, duplicates]).reset_index(drop=True)
```

[2]:
```python
df_banking
```

[2]:

	CustomerID	AccountID	TransactionID	CustomerName	CreditScore	SatisfactionScore	AccountBalance	TransactionFrequency	AccountType	LastTransactionDate
0	CUST00001	ACC00001	TRANS00001	Danny Campbell	Good	7.0	45829.29	Occasionally	Savings	2023-05-11
1	CUST00002	ACC00002	TRANS00002	Jenna Monroe	Very Good	10.0	37960.17	Rarely	Loan	2023-03-19
2	CUST00003	ACC00003	TRANS00003	Kenneth Lucas	Excellent	1.0	40037.08	Very Frequently	Savings	2023-09-13
3	CUST00004	ACC00004	TRANS00004	Chad Curry	Good	1.0	60086.02	Occasionally	Savings	2023-11-12
4	CUST00005	ACC00005	TRANS00005	Jamie Velasquez	Good	1.0	72663.46	Frequently	Checking	2023-08-11
...	...	...	...	...	...	...	...	...	...	...
1005	CUST00335	ACC00335	TRANS00335	Kylie Ruiz	Very Good	5.0	22561.67	Occasionally	Savings	2023-02-13
1006	CUST00297	ACC00297	TRANS00297	Dana Roth	Very Good	10.0	2036.77	Rarely	Investment	2023-08-21
1007	CUST00287	ACC00287	TRANS00287	Darren Patrick	Good	7.0	60125.20	Frequently	Loan	2023-04-06
1008	CUST00132	ACC00132	TRANS00132	Shawn Wilson	Good	1.0	3346.57	Frequently	Loan	2023-02-08
1009	CUST00653	ACC00653	TRANS00653	Chloe Bishop	Poor	2.0	70324.19	Very Frequently	Investment	2023-03-19

1010 rows × 10 columns

Figure 1: banking generative data with 1010 rows and 10 columns

DATA ATTRIBUTE

The dataset includes several key attributes for each customer and their banking activity:

CustomerID: A unique identifier assigned to each customer.

AccountID: A unique identifier for each customer's account.

TransactionID: A unique identifier for each transaction.

CustomerName: A randomly generated name to simulate individual customer identities.

CreditScore: Stands for the customer's credit quality, categorized into levels such as (Poor, Fair, Good, Very Good, and Excellent).

SatisfactionScore: A score between 1 and 10 to show customer satisfaction.

AccountBalance: A floating-point value standing for the account balance, ranging from 0 to 100,000.

TransactionFrequency: A frequency indicator for transactions, with values like (Rarely, Occasionally, Frequently, and Very Frequently).

AccountType: A categorical value writing down the type of account, such as (Savings, Checking, Investment, or Loan).

LastTransactionDate: The most recent date of transaction, randomly assigned within a specified range.

Brief discussion about missing values in the database generated.

Missing values were deliberately introduced in the dataset to mirror real-world data quality issues:

AccountBalance: Approximately 5% of records have missing values in the AccountBalance column. This could stand for scenarios where account balance data is temporarily unavailable or incomplete.

SatisfactionScore: Roughly 3% of records have missing values in the SatisfactionScore column, simulating cases where customer satisfaction feedback was not recorded or is missing.

Brief discussion on duplication of data in the generative data

Duplicate records were added to the dataset to further enhance realism:

Around 1% of the records were duplicated. This simulates potential errors or redundancies that can occur in real-world databases, such as unintentional data entry duplication.

These duplicate records appear across multiple columns, including CustomerID, CustomerName, and AccountID, reflecting how such issues might arise naturally in a dataset.

DATA TYPE

This dataset stands for various data types:

Nominal: CustomerID, CustomerName, CreditScore, AccountType, TransactionFrequency

Ordinal: CreditScore (categorical and ranked from Poor to Excellent)

Interval: SatisfactionScore (on a scale from 1 to 10 without a true zero)

Ratio: AccountBalance (numeric values with a true zero)

DATABASE SCHEMA

After generating banking database, it will hold 3 different tables Customers, Accounts and Transactions. Each table will have their attribute and based on the attribute the columns for each table will be selected from the banking database generated.

For customers, they have the following attribute from the generated data.

1. CustomerID
2. CustomerName
3. CreditScore
4. SatisifactionScore

[6]:
```
Customer= pd.read_csv('Customer.csv')

Customer
```

[6]:

	CustomerID	CustomerName	CreditScore	SatisfactionScore
0	CUST00001	Chad Watkins	Very Good	1.0
1	CUST00002	Heather Anderson	Very Good	10.0
2	CUST00003	Terri Hancock	Poor	4.0
3	CUST00004	Jessica Cuevas	Fair	7.0
4	CUST00005	Michael Miller	Poor	4.0
...	...	...	...	...
1005	CUST00320	Stephen Green	Poor	1.0
1006	CUST00785	Steven Taylor	Very Good	5.0
1007	CUST00720	Mary Mendoza	Fair	4.0
1008	CUST00067	Miguel James MD	Poor	4.0
1009	CUST00389	Kristin Wilson	Excellent	7.0

1010 rows × 4 columns

Figure 2: Customer table

For Accounts, it has the attribute.

1. AccountID
2. CustomerID
3. AccountType
4. AccountBalance

```
[8]: Account= pd.read_csv('Account.csv')

Account
```

[8]:

	AccountID	CustomerID	AccountType	AccountBalance
0	ACC00001	CUST00001	Investment	11775.88
1	ACC00002	CUST00002	Checking	59933.28
2	ACC00003	CUST00003	Investment	854.99
3	ACC00004	CUST00004	Savings	36176.76
4	ACC00005	CUST00005	Investment	35448.84
...	...	...	...	...
1005	ACC00320	CUST00320	Loan	22021.97
1006	ACC00785	CUST00785	Checking	71817.09
1007	ACC00720	CUST00720	Checking	54267.94
1008	ACC00067	CUST00067	Loan	49390.81
1009	ACC00389	CUST00389	Loan	28832.35

1010 rows × 4 columns

Figure 3: Accounts table

For Transaction, it has attribute.

1. TransactionID
2. AccountID
3. TransactionFrequency
4. LastTransactionDate

```
[10]: Transaction= pd.read_csv('Transaction.csv')

Transaction
```

[10]:

	TransactionID	AccountID	TransactionFrequency	LastTransactionDate
0	TRANS00001	ACC00001	Frequently	2023-09-24
1	TRANS00002	ACC00002	Rarely	2023-12-07
2	TRANS00003	ACC00003	Very Frequently	2023-06-19
3	TRANS00004	ACC00004	Rarely	2023-10-16
4	TRANS00005	ACC00005	Rarely	2023-12-17
...	...	...	...	...
1005	TRANS00320	ACC00320	Very Frequently	2023-08-19
1006	TRANS00785	ACC00785	Occasionally	2023-12-18
1007	TRANS00720	ACC00720	Occasionally	2023-09-15
1008	TRANS00067	ACC00067	Frequently	2023-12-25
1009	TRANS00389	ACC00389	Occasionally	2023-01-17

1010 rows × 4 columns

Figure 4: Transaction table

Importing csv file to DB browser SQLite

After each file is saved as csv (comma separated value), it was imported to the database browser SQLite. And it is displayed as followed when imported.

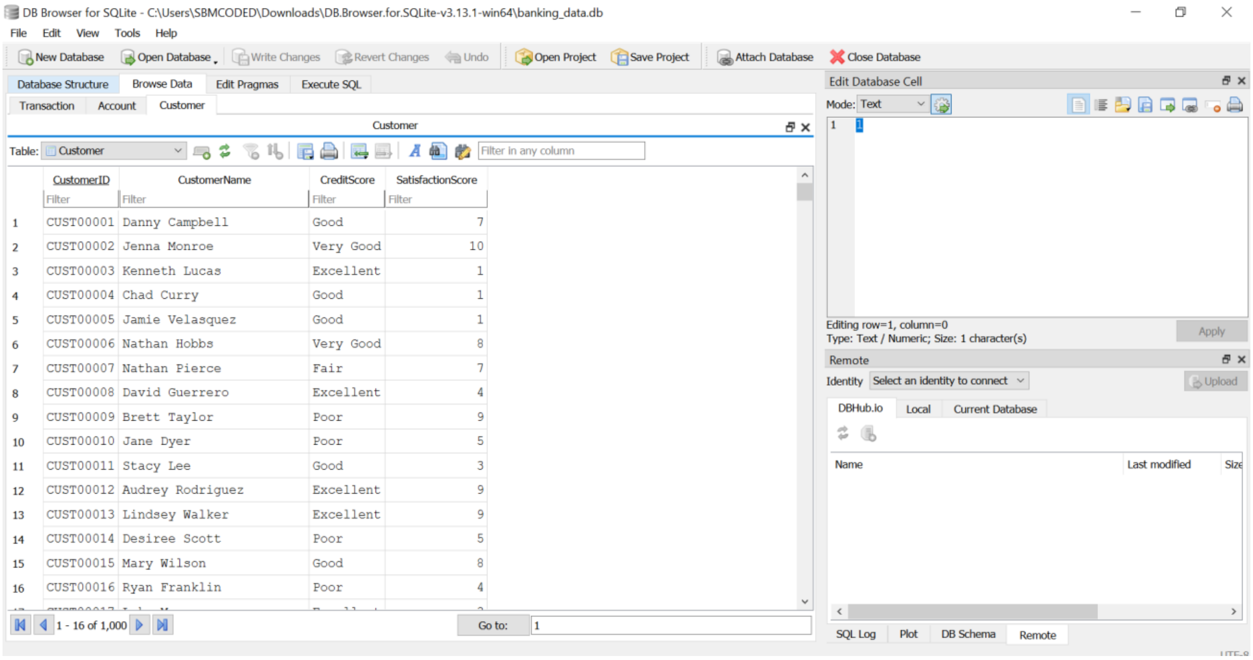

Figure 5: for customer table

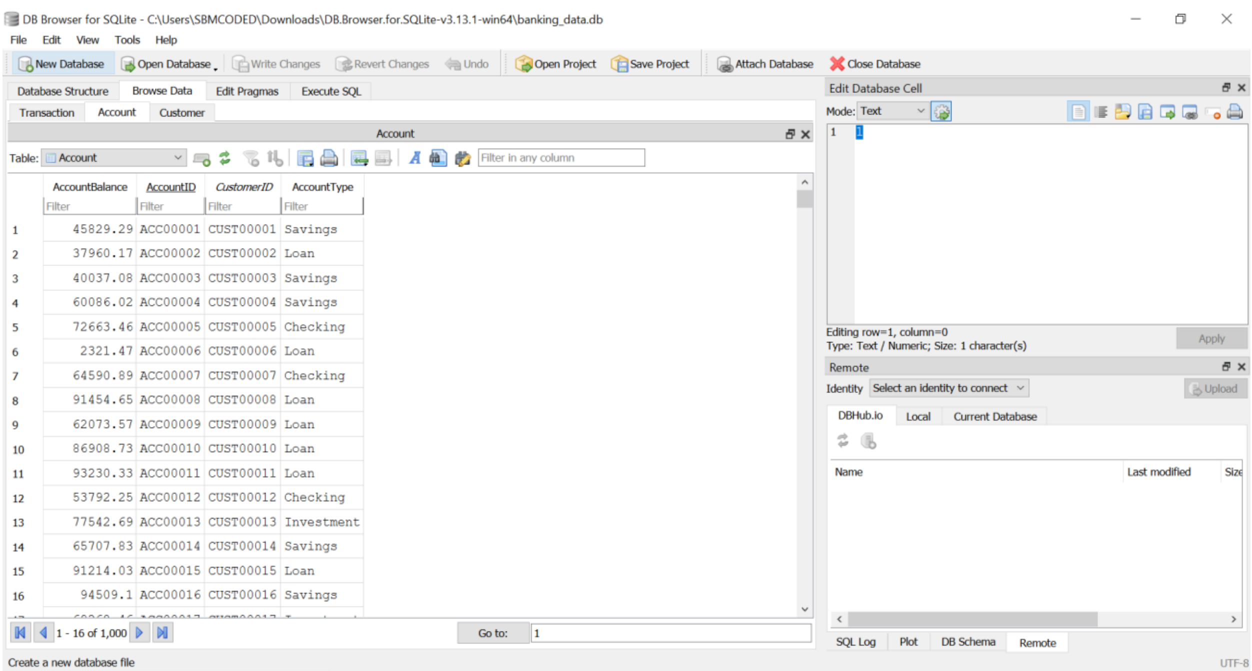

Figure 6: for Account table

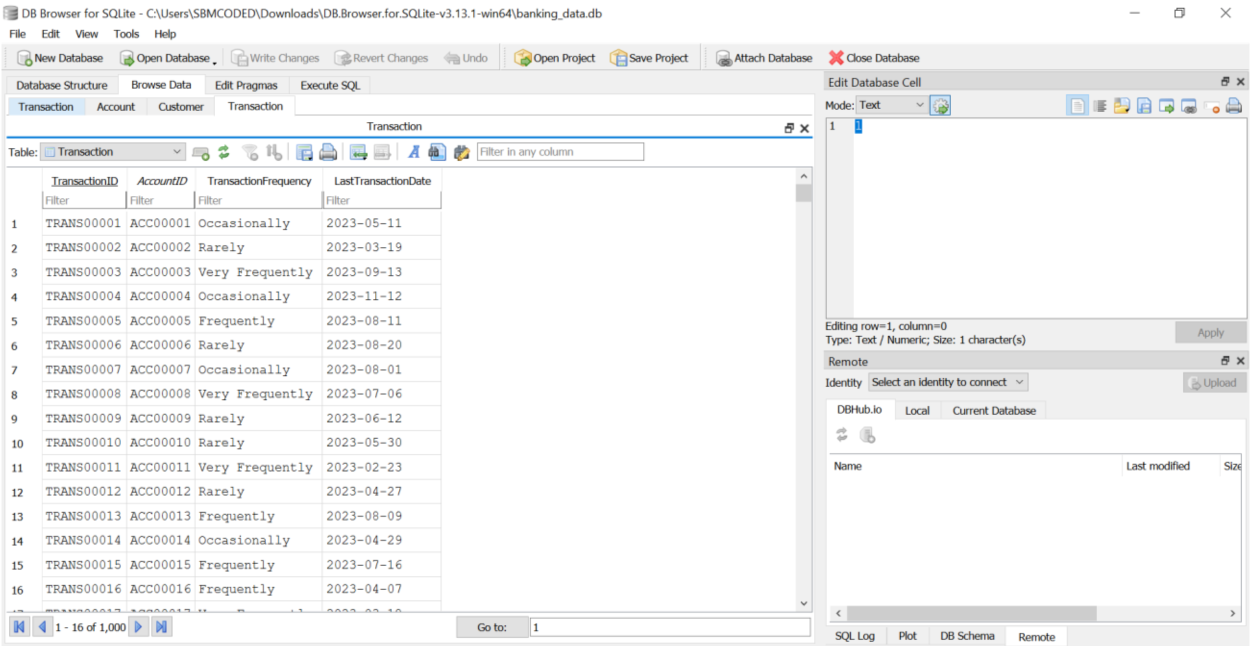

Figure 7: for Transaction table

JUSTIFICATION FOR TABLES

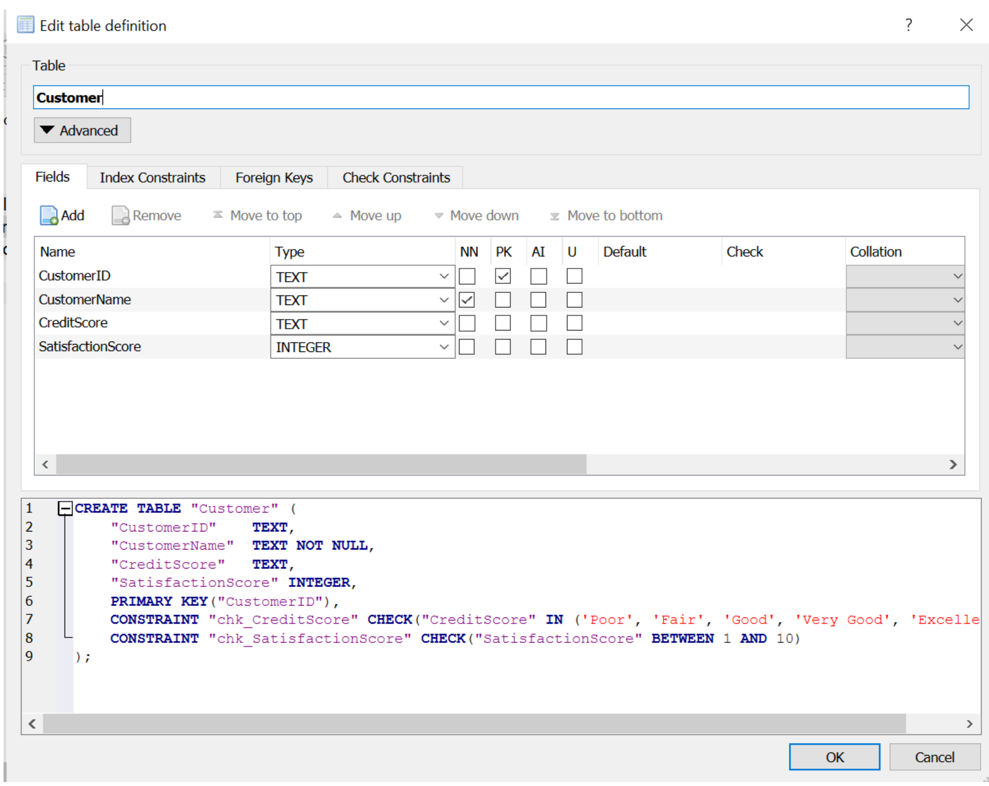

Figure 8: customer table

Customers Table: This table stores key information for each customer, including CreditScore and SatisfactionScore. Keeping customer data separate from account and transaction data helps in managing relationships and enhances security.

Attributes own by customer includes:

CutomerID: primary key (text)

CustomerName: not null (text)

CreditScore: (text)

Satisfactionscore: (integer)

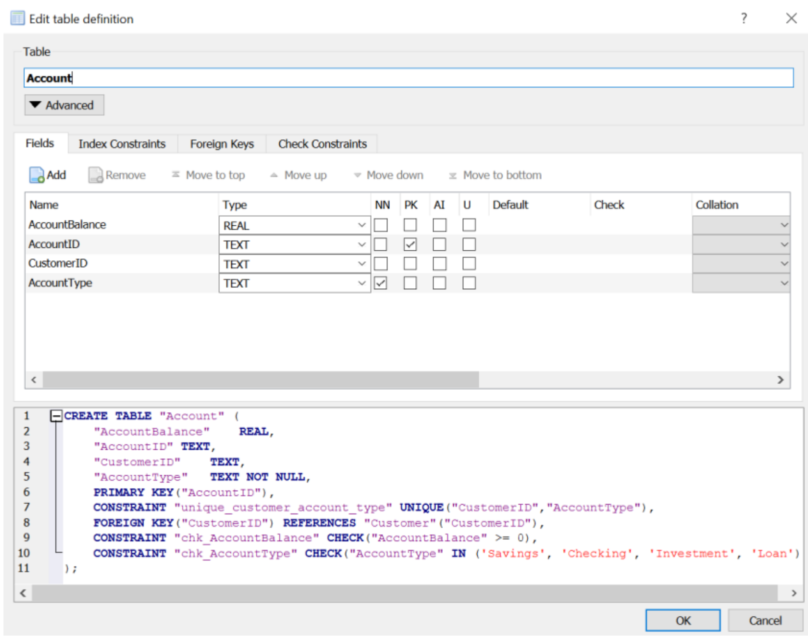

Figure 9: Account table

Accounts Table: This table links to the Customers table through CustomerID and includes AccountType and AccountBalance. A composite unique constraint (CustomerID and AccountType) prevents duplicate account types per customer, simulating realistic banking practices.

AccountID; primary key (text)

CustomerID: foreign key

AccountType:(not null) (text)

AccountBalance; (Real)

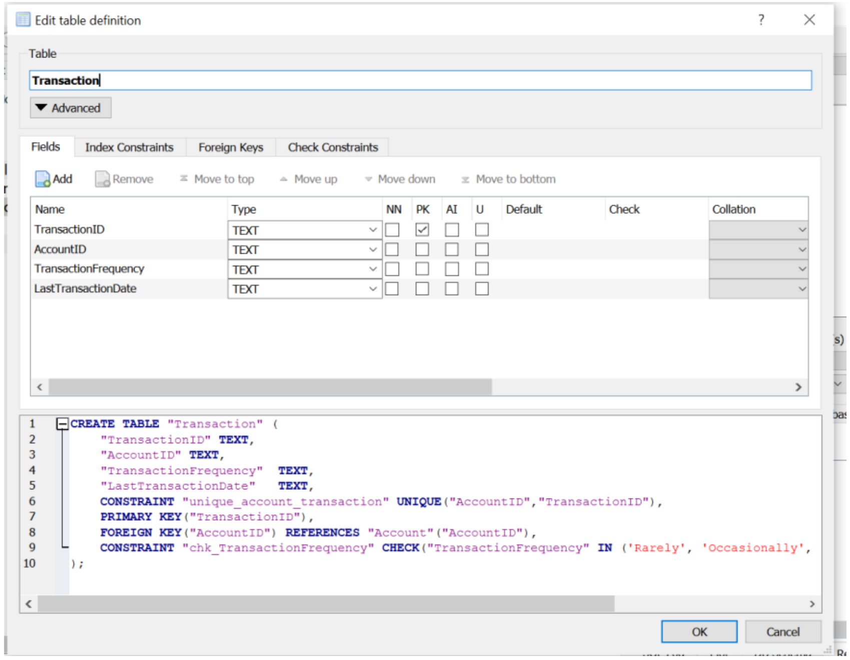

Figure 10: Transaction table

Transactions Table: This table records each transaction, linked to AccountID from the Accounts table. A composite unique (key) constraint on AccountID and TransactionID ensures that each transaction is unique for the given account.

TransactionID; primary key

AccountID; foreign key

TransactionFrequency: (text)

LastTransactionDate; (text)

Data Privacy Discussion:

- All data generated for this assignment is fictional, safeguarding against any misuse of personal information. The use of randomly generated names and details helps support privacy while simulating a real banking scenario.
- Following data protection principles, sensitive information has been organized to prevent any inadvertent exposure. Implementing composite keys and foreign keys in the database design reinforces relational integrity and ensures data consistency.