**Relational Database to NoSQL Conversion by Schema Migration and Mapping**

**International Journal of Computer Engineering In Research Trends**

**https://ijcert.org/ems/ijcert\_papers/V3I909.pdf**

**Project Group-46**

|  |  |
| --- | --- |
| **Name** | **ID** |
| Aparna Phundir | M23AID006 |
| Abhishek Kumar Gupta | M22AIE237 |
| Bhuvaneswari J | M23AID053 |

**Schema Migration Framework code:**

 **Schema Migration**: The script identifies and orders table dependencies based on foreign keys before migrating the schema.

 **Data Migration**: Data is migrated table by table into MongoDB collections, preserving relationships.

 **Column Renaming**: Columns are renamed to avoid conflicts in MongoDB collections.

 **Modular Design**: The script is designed to handle schema evaluation, foreign key extraction, and data migration separately, making it extensible and easier to debug.

import sqlite3

from pymongo import MongoClient

from collections import deque

class SchemaMigration:

**"""**

**A class to handle schema and data migration from SQLite to MongoDB.**

**Attributes:**

**-----------**

**sqlite\_db\_name : str**

**The name of the SQLite database file.**

**mongo\_uri : str, optional**

**The MongoDB connection URI (default: "mongodb://localhost:27017/").**

**mongo\_db\_name : str, optional**

**The MongoDB database name where the data will be migrated (default: "migrated\_db").**

**"""**

def \_\_init\_\_(self, sqlite\_db\_name, mongo\_uri="mongodb://localhost:27017/", mongo\_db\_name="migrated\_db"):

**"""**

**Initialize the SchemaMigration class with SQLite and MongoDB connections.**

**Parameters:**

**----------**

**sqlite\_db\_name : str**

**Name of the SQLite database file.**

**mongo\_uri : str, optional**

**MongoDB connection URI (default: "mongodb://localhost:27017/").**

**mongo\_db\_name : str, optional**

**MongoDB database name (default: "migrated\_db").**

**"""**

# SQLite connection setup

self.sqlite\_db\_name = sqlite\_db\_name

self.conn = sqlite3.connect(self.sqlite\_db\_name)

self.cursor = self.conn.cursor()

# MongoDB connection setup

self.client = MongoClient(mongo\_uri)

self.mongo\_db = self.client[mongo\_db\_name]

def get\_tables(self):

**"""**

**Retrieve all table names from the SQLite database.**

**Returns:**

**-------**

**list of str**

**A list of table names in the SQLite database.**

**"""**

self.cursor.execute("SELECT name FROM sqlite\_master WHERE type='table';")

return [table[0] for table in self.cursor.fetchall()]

def get\_foreign\_keys(self, table\_name):

**"""**

**Retrieve foreign keys for a given table.**

**Parameters:**

**----------**

**table\_name : str**

**Name of the table to check for foreign keys.**

**Returns:**

**-------**

**list of tuples**

**A list of foreign key relationships for the table.**

**"""**

self.cursor.execute(f"PRAGMA foreign\_key\_list({table\_name});")

return self.cursor.fetchall()

def migrate\_schema(self):

**"""**

**Determine the order of schema migration based on table relationships.**

**Returns:**

**-------**

**list of str**

**A list of tables in the order they should be migrated.**

**"""**

all\_tables = self.get\_tables()

migrated\_tables = set()

linked\_tables = deque() # Linked list to store migration order

for table\_name in all\_tables:

if table\_name in migrated\_tables:

continue

self.evaluate\_table(table\_name, migrated\_tables, linked\_tables)

print("Linked tables for migration:", list(linked\_tables))

return list(linked\_tables)

def evaluate\_table(self, table\_name, migrated\_tables, linked\_tables):

**"""**

**Evaluate table relationships and add related tables for migration.**

**Parameters:**

**----------**

**table\_name : str**

**The current table to evaluate.**

**migrated\_tables : set**

**Set of already migrated tables.**

**linked\_tables : deque**

**Ordered list of tables for migration.**

**"""**

visited = set() # Track visited tables to avoid cycles

queue = deque([table\_name])

while queue:

current\_table = queue.popleft()

if current\_table in visited:

continue

visited.add(current\_table)

linked\_tables.append(current\_table)

migrated\_tables.add(current\_table)

foreign\_keys = self.get\_foreign\_keys(current\_table)

for fk in foreign\_keys:

related\_table = fk[2] # FK points to related table

if related\_table not in migrated\_tables:

queue.append(related\_table)

def rename\_columns\_with\_table\_name(self, table\_name, columns):

**"""**

**Rename columns by prefixing them with the table name to avoid conflicts.**

**Parameters:**

**----------**

**table\_name : str**

**The name of the table.**

**columns : list of str**

**List of column names.**

**Returns:**

**-------**

**list of str**

**List of renamed column names.**

**"""**

return [f"{table\_name}\_{col}" if col != 'id' else col for col in columns]

def migrate\_data(self, linked\_tables):

**"""**

**Migrate data from SQLite tables to MongoDB collections in the specified order.**

**Parameters:**

**----------**

**linked\_tables : list of str**

**The ordered list of tables for migration.**

**"""**

for table\_name in linked\_tables:

self.cursor.execute(f"PRAGMA table\_info({table\_name});")

columns = [col[1] for col in self.cursor.fetchall()]

renamed\_columns = self.rename\_columns\_with\_table\_name(table\_name, columns)

self.cursor.execute(f"SELECT \* FROM {table\_name};")

rows = self.cursor.fetchall()

mongo\_data = []

for row in rows:

document = {col\_name: value for col\_name, value in zip(renamed\_columns, row)}

mongo\_data.append(document)

mongo\_collection = self.mongo\_db[table\_name]

if mongo\_data:

mongo\_collection.insert\_many(mongo\_data)

print(f"Data from table '{table\_name}' migrated to MongoDB.")

def close(self):

**"""**

**Close the SQLite connection.**

**"""**

self.conn.close()

# Usage example

sqlite\_db\_name = 'example.db' # Your SQLite database file name

migration = SchemaMigration(sqlite\_db\_name)

# Migrate schema and data for all tables in linked order

linked\_tables = migration.migrate\_schema()

migration.migrate\_data(linked\_tables)

# Close SQLite connection

migration.close()

**Data Mapping Framework code:**

This script facilitates the conversion of SQL queries into NoSQL queries and execution on a MongoDB database.

It consists of three main classes: `Mediator`, `Convert`, and `DatabaseMetadata`.

**Modules Used:**

- xml.etree.ElementTree (ET): For parsing and generating XML data.

- pymongo: For MongoDB operations.

- sqlparse: For parsing and analyzing SQL queries.

- re: For regular expressions used in parsing SQL statements.

**Classes:**

1. Mediator:

Acts as the intermediary between the application and the query processing logic. It intercepts SQL queries,

converts them to XML format, and delegates the conversion to NoSQL and execution to the `Convert` class.

2. Convert:

Handles parsing and transformation of SQL queries into NoSQL equivalents suitable for MongoDB.

Executes NoSQL queries on the MongoDB database.

3. DatabaseMetadata:

Stores the mapping between relational database table names and their corresponding MongoDB collection names.

import xml.etree.ElementTree as ET

from pymongo import MongoClient

import sqlparse

from sqlparse.sql import Identifier, IdentifierList, Where

from sqlparse.tokens import Keyword, DML

import re

class Mediator:

**"""**

**Intercepts SQL queries, converts them to XML format, and delegates the processing to the Convert class.**

**Attributes:**

**db\_metadata (DatabaseMetadata): The metadata object mapping relational tables to MongoDB collections.**

**convert (Convert): The Convert object for processing and executing NoSQL queries.**

**"""**

def \_\_init\_\_(self, db\_metadata, convert):

self.db\_metadata = db\_metadata

self.convert = convert

def intercept\_query(self, sql\_query, query\_type):

**"""**

**Intercepts and processes an SQL query by converting it to XML format and delegating to Convert.**

**Args:**

**sql\_query (str): The SQL query to intercept.**

**query\_type (str): The type of SQL query (SELECT, INSERT, UPDATE, DELETE).**

**Returns:**

**dict: The formatted result of the executed query.**

**"""**

xml\_request = self.create\_xml\_request(sql\_query, query\_type)

print("Query intercepted and converted to XML format.")

result = self.convert.process\_query(xml\_request)

return self.format\_result(result)

def create\_xml\_request(self, query, query\_type):

**"""**

**Generates an XML request for the given SQL query.**

**Args:**

**query (str): The SQL query.**

**query\_type (str): The type of SQL query.**

**Returns:**

**bytes: The XML request in bytes format.**

**"""**

root = ET.Element("Request")

ET.SubElement(root, "xmlns").text = "queryInterceptor"

ET.SubElement(root, "method").text = "Intercepta"

ET.SubElement(root, "query").text = query

ET.SubElement(root, "queryType").text = query\_type

return ET.tostring(root, encoding="utf-8", method="xml")

def format\_result(self, result):

**"""**

**Formats the NoSQL query result into a relational format.**

**Args:**

**result (list): The NoSQL query result.**

**Returns:**

**dict: A dictionary containing headers and rows.**

**"""**

return {

"headers": list(result[0].keys()) if result else [],

"rows": [list(row.values()) for row in result]

}

class Convert:

**"""**

**Handles the transformation of SQL queries into NoSQL queries and executes them.**

**Attributes:**

**mongo\_client (MongoClient): The MongoDB client.**

**db\_metadata (DatabaseMetadata): The metadata object for table-to-collection mapping.**

**"""**

def \_\_init\_\_(self, mongo\_client, db\_metadata):

self.mongo\_client = mongo\_client

self.db\_metadata = db\_metadata

def process\_query(self, xml\_request):

**"""**

**Processes an XML query request by parsing and executing it.**

**Args:**

**xml\_request (bytes): The XML query request.**

**Returns:**

**list: The result of the NoSQL query execution.**

**"""**

query\_data = self.parse\_xml\_request(xml\_request)

sql\_query = query\_data['query']

query\_type = query\_data['queryType']

nosql\_query = self.translate\_to\_nosql(sql\_query, query\_type)

return self.execute\_nosql\_query(nosql\_query, query\_type)

def parse\_xml\_request(self, xml\_request):

**"""**

**Parses the XML request to extract query data.**

**Args:**

**xml\_request (bytes): The XML query request.**

**Returns:**

**dict: Extracted query data containing 'query' and 'queryType'.**

**"""**

root = ET.fromstring(xml\_request)

return {

"query": root.find("query").text,

"queryType": root.find("queryType").text

}

# [Other methods in the Convert class handle query translation and execution.]

class DatabaseMetadata:

**"""**

**Stores metadata mapping relational database tables to MongoDB collections.**

**Attributes:**

**table\_collection\_mapping (dict): A dictionary mapping table names to collection names.**

**"""**

def \_\_init\_\_(self, table\_collection\_mapping):

self.table\_collection\_mapping = table\_collection\_mapping

def get\_collection\_name(self, table\_name):

**"""**

**Retrieves the collection name corresponding to a table name.**

**Args:**

**table\_name (str): The relational database table name.**

**Returns:**

**str: The MongoDB collection name.**

**"""**

return self.table\_collection\_mapping.get(table\_name)

**Execution Steps:**

* Create DB in Sqlite with table structure and data.
* Run mongo db using mongod command in command prompt.
* Execute schema migration framework python code to migrate sqlite db data to mongo db.
* Execute mapping framework python code to to facilitate the execution of SQL queries on a NoSQL database.
* Execute sql2nosql\_plot.py to validate performance of sqlite and mongodb transaction.