**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 |

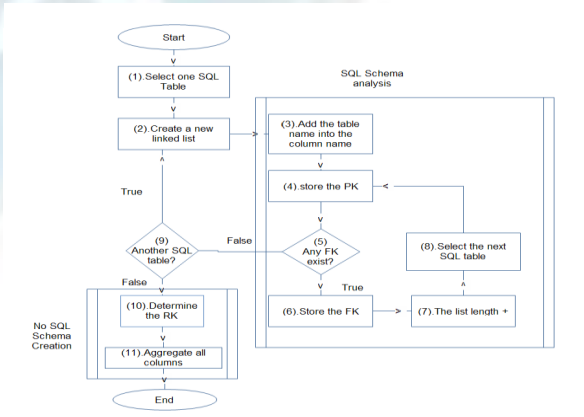
**About Project:**

* To build a Schema-Migration and Mapping Framework using python to support automatic data migration from relational databases to NoSQL.
* The Schema Migration class automates the migration of data from a SQLite database to Mongo DB, preserving relationships and dependencies between tables through foreign key analysis.
* The Data mapping system is designed to facilitate the execution of SQL queries on a NoSQL database.

**Prerequisites**

* Sqlite
* MongoDB
* Python 3.x
* xml.etree.ElementTree
* Sqlparse
* Pymongo
* re

**Schema Migration Framework:**

****

The SchemaMigration class automates the migration of data from a SQLite database to MongoDB, preserving relationships and dependencies between tables through foreign key analysis. The migration involves:

1. Extracting the schema and table relationships.
2. Determining the order of table migrations based on dependencies.
3. Renaming columns for clarity during migration.
4. Transferring data from SQLite to MongoDB.

**System Architecture**

**Components**

1. **SQLite Database**:
   * Stores relational data with foreign key dependencies.
   * Acts as the source database for migration.
2. **MongoDB**:
   * NoSQL database used as the destination.
   * Documents in collections represent the migrated tables.
3. **SchemaMigration Class**:
   * Handles schema analysis, table dependency resolution, and data transfer.

**Execution Flow**

1. **Initialization**:
   * Establishes connections to SQLite and MongoDB.
2. **Schema Extraction**:
   * Identifies all tables and their relationships via foreign key analysis.
3. **Migration Order Determination**:
   * Evaluates tables and resolves dependencies to ensure correct migration order.
4. **Data Migration**:
   * Transfers data from SQLite tables to MongoDB collections.
5. **Design Details**

| **Class/Method** | **Responsibility** |
| --- | --- |
| **SchemaMigration** | Encapsulates migration logic and manages SQLite and MongoDB connections. |
| **get\_tables** | Retrieves all table names from the SQLite database. |
| **get\_foreign\_keys** | Extracts foreign key constraints for a given table. |
| **migrate\_schema** | Determines the migration order of tables based on relationships and dependencies. |
| **evaluate\_table** | Explores table dependencies using foreign key constraints and builds a migration sequence. |
| **rename\_columns\_with\_table\_name** | Adds table prefixes to column names to avoid naming conflicts and improve clarity. |
| **migrate\_data** | Transfers data from SQLite tables to MongoDB collections while maintaining the migration sequence. |
| **close** | Closes the SQLite connection. |

#### ****Data Flow****

1. **Schema Extraction**:
   * Fetch all table names using sqlite\_master.
   * Identify relationships using PRAGMA foreign\_key\_list.
2. **Dependency Resolution**:
   * Utilize a breadth-first search approach to evaluate dependencies.
   * Build a migration sequence ensuring dependent tables are migrated before their references.
3. **Data Migration**:
   * For each table in the migration sequence:
     + Retrieve column names and data.
     + Rename columns for clarity.
     + Insert transformed data into the corresponding MongoDB collection.

### ****Technical Details****

#### ****Database Handling****

1. **SQLite**:
   * Uses the sqlite3 module for querying metadata (sqlite\_master, PRAGMA) and data.
2. **MongoDB**:
   * Uses the pymongo library to insert documents into collections.

#### ****Key Algorithms****

1. **Dependency Resolution**:
   * Implements a BFS-like traversal to process tables based on foreign key dependencies.
   * Ensures that no table is migrated before its dependencies.
2. **Data Transformation**:
   * Renames columns (except primary keys like id) by prefixing them with table names.

### ****Usage Instructions****

1. Replace sqlite\_db\_name with the name of the SQLite database file.
2. Ensure MongoDB is running locally or update the mongo\_uri parameter.
3. Execute the script to migrate schema and data from SQLite to MongoDB.

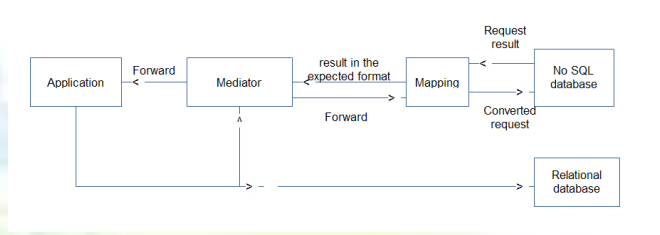
### ****Assumptions and Constraints****

1. Foreign key constraints in SQLite are correctly defined.
2. MongoDB collections are named after their respective SQLite tables.
3. Data integrity is preserved by migrating tables in dependency order.

### ****Error Handling****

1. **SQLite Connection Errors**:
   * Handled by Python's built-in exceptions when opening the database.
2. **MongoDB Connection Errors**:
   * If the connection fails, the script will raise an exception.
3. **Circular Dependencies**:
   * Circular foreign key dependencies may cause infinite loops. The design tracks visited tables to prevent this.

**Data Mapping Framework:**

****

The SQL-to-NoSQL Query Translation and Execution System is designed to facilitate the execution of SQL queries on a NoSQL database. This is achieved by intercepting SQL queries, converting them to a NoSQL equivalent format, and executing them on a MongoDB database. The system comprises three core components: the Mediator, the Convert module, and the Database Metadata.

### ****3. Components and Responsibilities****

#### ****3.1 Mediator****

* **Purpose**: Intercepts SQL queries, converts them into an XML format, and delegates processing to the Convert module.
* **Responsibilities**:
  + Create an XML representation of intercepted SQL queries.
  + Communicate with the Convert module for query translation and execution.
  + Format NoSQL query results into a relational format with headers and rows.

#### ****3.2 Convert****

* **Purpose**: Handles the core logic for translating SQL queries to NoSQL queries and executing them.
* **Responsibilities**:
  + Parse the XML query request received from the Mediator.
  + Extract relevant SQL components like table name, WHERE clauses, and SET or VALUES data.
  + Map SQL queries to equivalent MongoDB queries.
  + Execute the MongoDB queries and return results.

#### ****3.3 DatabaseMetadata****

* **Purpose**: Maintains a mapping between relational table names and NoSQL collection names.
* **Responsibilities**:
  + Provide collection names corresponding to SQL table names for query translation.

### ****4. Architecture****

The system follows a **modular architecture** with three main classes:

1. **Mediator**: Manages interaction between the SQL-based application and the Convert module.
2. **Convert**: Implements the logic for SQL parsing, translation to NoSQL, and execution.
3. **DatabaseMetadata**: Serves as a lookup for table-to-collection mappings.

### ****5. Workflow****

#### ****5.1 Query Interception****

1. The Mediator receives an SQL query and query type.
2. The query is converted into XML format.
3. The Mediator sends the XML to the Convert module.

#### ****5.2 Query Processing****

1. The Convert module parses the XML and extracts query details (e.g., query type, table name, WHERE clause).
2. The SQL query is translated to a MongoDB query using:
   * Table-to-collection mapping.
   * Clause parsing (e.g., WHERE, SET).
3. The MongoDB query is executed, and results are retrieved.

#### ****5.3 Result Formatting****

1. The Convert module returns the execution result to the Mediator.
2. The Mediator formats the result into a relational format with headers and rows.
3. The formatted result is returned to the SQL-based application.

### ****6. Supported Features****

#### ****6.1 Query Types****

* **SELECT**: Translated to MongoDB find() queries.
* **INSERT**: Translated to MongoDB insert\_one() operations.
* **UPDATE**: Translated to MongoDB update\_many() operations.
* **DELETE**: Translated to MongoDB delete\_many() operations.

#### ****6.2 SQL Clause Translation****

* **Table Name**: Extracted from FROM clause or INSERT INTO.
* **WHERE Clause**: Translated to MongoDB filters using operators like $eq, $gt, $lt.
* **SET Clause**: Parsed for field-value pairs for UPDATE queries.
* **VALUES Clause**: Extracted for INSERT queries.

### ****7. Implementation Details****

#### ****7.1 Key Methods****

* **Mediator Class**:
  + intercept\_query: Main entry point for SQL queries.
  + create\_xml\_request: Converts SQL queries to XML format.
  + format\_result: Formats NoSQL results into relational format.
* **Convert Class**:
  + process\_query: Parses XML and delegates to translation and execution methods.
  + translate\_to\_nosql: Converts SQL queries to MongoDB queries.
  + execute\_nosql\_query: Executes MongoDB queries.
* **DatabaseMetadata Class**:
  + get\_collection\_name: Retrieves the collection name for a given table.

#### ****7.2 Query Translation****

* SQL parsing is achieved using the sqlparse library.
* WHERE clauses are parsed to construct MongoDB filters.
* INSERT and UPDATE data fields are extracted and mapped to MongoDB formats.

### ****8. Error Handling****

* Invalid SQL queries (e.g., missing table names) raise a ValueError.
* Mismatched columns and values in INSERT queries raise a ValueError.
* Unsupported query types raise a ValueError.